Computer Networks 281 (2026) 112216

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Scalable optimization for congestion-aware NFV deployment

Mohammad A. Raayatpanah [a], Thomas Weise [b], Jocelyne Elias [c,*], Fabio Martignon [d], Andrea Pimpinella [d]

[a] Mathematical Sciences and Computer, Kharazmi University, Teheran, Iran
[b] Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University, Hefei, Anhui, China
[c] Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
[d] Department of Management, Information and Production Engineering, University of Bergamo, Bergamo, Italy

ARTICLE INFO

Keywords:
Network functions virtualization (NFV)
Non-convex optimization
Mixed-integer nonlinear programming (MINLP)
Resource allocation
Cost modeling
Cutting-plane heuristic

ABSTRACT

This paper introduces a novel optimization framework for Network Functions Virtualization (NFV) that addresses the efficient implementation of end-to-end service requests in physical networks. Our approach characterizes each server node by a reliability function reflecting its computational load, which aids in balancing workloads and mitigating congestion. By optimizing the reliability metric along the route, our approach ensures robust end-to-end service quality. We formulate the NFV deployment problem as a non-convex mixed-integer non-linear programming (MINLP) model aimed at minimizing both deployment and operational costs while maximizing resource utilization, addressing also per-node installation conflicts and inter-VNF incompatibilities. Given the NP-hard nature of the problem, we develop efficient linearization techniques and bounding schemes, using also dynamic programming, to convert the formulation into a tractable mixed-integer linear programming (MILP) model. Additionally, a cutting-plane-based heuristic with a warm-start strategy is proposed to further accelerate convergence. Experimental evaluations on real-world network topologies demonstrate that our framework offers scalable and cost-effective solutions compared to existing approaches.

1. Introduction

Network Functions Virtualization (NFV) has revolutionized modern network architectures by decoupling network services from dedicated hardware and implementing them as software running on general-purpose servers [1]. This paradigm shift enables dynamic deployment of Virtual Network Functions (VNFs) that can be instantiated and executed by Communication Service Providers to efficiently scale services based on demand. Beyond optimizing resource utilization, NFV enhances network flexibility, and reduces both capital (CAPEX) and operational (OPEX) expenditures. Besides its immediate application to the management of network infrastructures, NFV enables new opportunities across diverse domains, such as cloud computing, edge and fog environments, and even industry-specific contexts like smart manufacturing or healthcare [2], where dynamic and reliable service provisioning is essential [3,4]. However, despite its promises, the deployment and optimization of NFV systems pose complex challenges, particularly in ensuring efficient resource allocation, network congestion management and service reliability [5]. The design of effective NFV systems necessitates robust optimization models to efficiently allocate VNFs to commercial off-the-shelf server nodes, while accounting for computational capacity, traffic demand, and service reliability constraints. This problem is inherently complex [6], requiring advanced mathematical models to balance cost minimization, efficient resource utilization, and service quality. Also, the non-convex nature of the problem adds complexity, requiring specialized techniques for effective decision-making.

In this paper, we address these challenges by introducing novel, advanced optimization techniques tailored for NFV deployments. We propose a comprehensive framework that assigns VNFs to server nodes based on their congestion levels and demands, optimizing workload distribution and ensuring robust end-to-end service delivery. In particular, the framework characterizes each server node through a reliability function reflecting its computational load: by optimizing the reliability metric along the service route, our approach balances network workload, mitigates congestion and preserves end-to-end service quality. By focusing on these latter objectives, our model also indirectly contributes to improve energy efficiency by minimizing server overloading and reducing unnecessary redundancy, thereby lowering overall energy consumption in the network infrastructure. Lastly, our framework specifically targets the minimization of infrastructure costs, thus capturing the

* Corresponding author.
E-mail addresses: tweise@ustc.edu.cn (T. Weise), jocelyne.elias@unibo.it (J. Elias), fabio.martignon@unibg.it (F. Martignon), andrea.pimpinella@unibg.it (A. Pimpinella).

https://doi.org/10.1016/j.comnet.2026.112216
Received 10 October 2025; Received in revised form 18 February 2026; Accepted 11 March 2026
Available online 14 March 2026

crucial trade-off between service quality, operational expenditures and infrastructure maintenance that virtual and cloud service providers must consider. Precisely, our contributions can be summarized as follows:

- *Mathematical Modeling*: We formulate the NFV deployment problem as a non-convex Mixed-Integer Non-Linear Programming (MINLP) model [7], incorporating routing constraints, VNF placement decisions, and network capacity limitations. We further extend the model by introducing binary parameters that specify whether a given function can be installed on a given node, enabling per-node VNF allocation restrictions. We also model conflicts between VNFs that cannot be deployed together on the same physical node because of functional or security incompatibilities.
- *Linearization Techniques*: We introduce a novel linearization approach, including dynamic programming-based reformulations, to transform non-linear constraints in the original MINLP into a tractable Mixed-Integer Linear Programming (MILP) model. This transformation enhances computational efficiency and scalability for large-scale networks.
- *Heuristic Optimization*: To address the NP-hard nature of the problem, we develop a cutting-plane-based heuristic algorithm that combines bounding schemes with a warm-start strategy. This approach accelerates convergence by generating high-quality initial cuts and pruning the search space effectively.
- *Baseline Approaches*: We introduce and compare two baseline models. The first baseline omits the key end-to-end reliability constraint, revealing its critical role in preserving service guarantees. The second baseline simplifies the reliability condition by imposing it directly on individual server nodes, rather than on the end-to-end path. These comparisons show the effectiveness of our model design.
- *Performance Evaluation*: We validate our framework on realistic topologies derived from the SNDLib database and other NFV-relevant datasets. Our model accounts for heterogeneous nodes (e.g., general-purpose servers, SmartNICs, and PISA switches) and a wide range of VNF types with specific computational demands. Experimental results demonstrate the framework's effectiveness in minimizing costs, improving reliability, and reducing congestion.

We conduct a detailed analysis of performance metrics such as total cost, execution time, node and link utilization. Numerical results show the impact of key parameters, such as end-to-end reliability requirements, volume and requested bandwidth of service demands, links and nodes capacity, and VNF placement cost, on network configuration. Considering realistic network scenarios, we show that at the highest feasible end-to-end reliability guarantee our framework trades an average increase in overall infrastructure costs of 1% with a corresponding average reduction of maximum server resource consumption equal to 46% in the largest instance, which determines an average 23% higher fairness of load distribution among network nodes. The interplay of links capacity, nodes computational resources and feasibility of reliability guarantees is also investigated, such to provide generalizable insights for service providers. Overall, the proposed model achieves the optimal solution in short computing times – most of the experiments are solved within 4 min and consistently faster than 7 min regardless of network topology and service request demand – compared to compact ILP formulations, demonstrating its suitability for real-time NFV orchestration. In summary, our paper presents an integrated and realistic framework for NFV optimization, combining advanced linearization and heuristic techniques with rigorous evaluation on practical network scenarios. The resulting solutions are both scalable and cost-effective, offering a valuable contribution toward reliable, efficient, and flexible NFV deployments.

The structure of this paper is as follows: Section 2 defines the NFV resource allocation problem, outlining its key components. Section 3 presents the mathematical formulation, detailing the objective function and constraints governing VNF placement and traffic routing. Section 4 introduces a cutting-plane heuristic that constructs lower and upper bounding models to relax complex constraints in the NP-hard NFV deployment problem. It also incorporates a warm-start strategy to generate initial tight cuts, enhancing convergence. Section 6 analyzes numerical results that demonstrate the model's effectiveness in two real network topologies. Section 7 discusses related works, while Section 8 concludes the manuscript.

2. Problem statement

In this work, we address the problem of routing and resource optimization for VNFs in end-to-end service requests, where requests can be estimated based on their service contracts. Each server node is associated with a concave, non-increasing *reliability function* that captures the total computational resources required to process all VNFs hosted on that server. This function design effectively mitigates network load by promoting an optimal distribution of workloads across servers. The probability of successfully routing a request from its source to its destination is then computed as the product of the reliability functions of the servers along its path. Our objective is to minimize the total cost of traffic routing and VNF deployment. This includes capital expenditures (deployment costs of VNFs on servers), link costs (bandwidth usage), and server activation costs. While the objective explicitly targets deployment and operational expenditures, it also implicitly promotes efficient utilization of computational and bandwidth resources, ensuring that end-to-end service delivery satisfies a prescribed reliability threshold.

We first formulate the problem as a non-convex MINLP and show that it is NP-hard. To overcome the nonlinearity, we introduce a linearization technique based on dynamic programming, enabling a Mixed-Integer Linear Program (MILP) reformulation. Since solving the MILP can become impractical for large-scale network instances, we further propose a cutting-plane algorithm-based heuristic that computes near-optimal solutions within less than 7 min of execution time. Finally, we evaluate the performance of our approach on real-world network topologies and compare it against baseline solutions.

In the remainder of this section, we introduce the main definitions and mathematical notations that will be used in the formulation presented in the next section.

We model the physical network as an undirected graph $G = (N, L)$, where N and L denote the set of nodes and links. Each node $n \in N$ is characterized by two resource capacities: the computational resource capacity (e.g., CPU) denoted by sc_n, and the bandwidth capacity (e.g., memory or disk capacity) denoted by sb_n. A node may be a pure switch without processing capability ($sc_n = 0$), or a switch with attached NFV middleboxes, or a server ($sc_n > 0$). Nevertheless, irrespective of its processing capability and whether it hosts VNF instances, the node incurs a switching activation cost sa_n for each routed service request.

The set of all Virtual Network Functions (VNFs) that may be deployed in the network is denoted by F. Each VNF type $f \in F$ is associated with a processing capacity θ_f, i.e., the maximum amount of traffic (in Mbps) that a single instance of VNF f can handle. VNFs may be hardware-based (directly connected to switches) or software-based (deployed on server components within nodes).

The operator is responsible for handling a set of end-to-end requests K. Each request $k \in K$originates at a source node o^k and terminates at a destination node t^k. We assume that every request is routed along a single path between its source and destination. The bandwidth demand of request k is denoted by b^k.

Given the processing capacity θ_f, the number of instances of VNF f required to process the traffic of request k is $m_f^k = \left\lceil \frac{b^k}{\theta_f} \right\rceil$, where $m_f^k \in \mathbb{Z}^+$. Note that in our model m_f^k does not represent a chain of serial middleboxes that increases the throughput of a single flow by sequential processing. Instead, it denotes the total number of instances of VNF f required to process request k, given its traffic demand b^k and

the per-instance processing capacity θ_f. These instances contribute to the computational load of the nodes on which they are deployed and are used to model heterogeneous resource usage and load balancing across the infrastructure, rather than an increase of per-flow bandwidth through serialization. This abstraction is consistent with NFV deployments where the software implementation of a VNF can be scaled out horizontally across virtual resources. If an operator prefers to enforce that all instances of a given VNF type for a specific request must be colocated on a single node, this can be achieved by adding integrality constraints that force ρ^k_{fn} to take either value 0 or m^k_f, ensuring that the full VNF processing demand is allocated to one node only.

Some VNFs cannot be deployed together on the same physical node because of functional or security incompatibilities. We refer to these as **conflicting functions**. For example, two VNFs that introduce mutual performance degradation, or a firewall and a traffic monitoring function that must remain logically separated, are considered conflicting. Let $F^k \subseteq F$ be the set of VNFs required to serve request k; then, the set of conflicting VNF pairs for such request is denoted by $FC^k \subseteq F^k \times F^k$.

Thus, each request $k \in K$can be described by the tuple $\{o^k, t^k, b^k, (f, m^k_f)_{f\in F^k}\}$, which specifies its source, destination, bandwidth demand, and the required VNFs with their respective number of instances.

Each VNF $f \in F_k$ incurs a deployment cost reflecting infrastructure usage, resource allocation (CPU, memory, storage), licensing, and operational expenses. We denote by $\alpha_{n,f}$ the cost of deploying VNF $f \in F$ on server node $n \in N$. Parameter $\eta^k_{n,f}$ represents the amount of computational resources that must be allocated to VNF f on node n for request $k \in K$. The feasibility of deploying VNF f on node n is expressed by the binary parameter δ_{nf}, which equals 1 if node n can support VNF f, and 0 otherwise.

Each physical link $(i, j) \in L$ has a bandwidth capacity $lc_{i,j}$ and an activation cost $la_{i,j}$. We assume that switches and links are perfectly reliable, while servers are subject to load-dependent reliability degradation. Each server $n \in N$ is associated with what we call a *reliability function* R_n, which is a non-increasing function of the total amount of computational resources required to process all the VNFs hosted and executed on the server. This function is therefore directly related to the probability of successful data transit through the server. This captures the empirical observation that failure probability is a non-decreasing function of workload [8]: servers operating at high utilization exhibit significantly higher error and failure rates compared to lightly loaded ones. Hence, reliability decreases as server load increases. We further assume that server reliabilities are independent. For a request k routed along a path P with intermediate servers $n = 1, 2, \dots i$, the reliability of the entire path (a measure of the overall congestion level on such path) is given by the product of the reliabilities of the servers on that path: $\prod_{n=1}^{i} R_n$. A path is considered *reliable* if it satisfies the request-specific reliability threshold τ_k. Thus, for each request $k \in K$, data from source o^k must reach destination t^k with probability at least τ_k, where $0 < \tau_k \leq 1$.

Finally, Table 1 summarizes the parameters and variables used throughout the paper.

Table 1
Notations for parameters and variables.

Parameter	Description
$G(N, L)$	Physical network represented as a graph G with nodes $n \in N$ and links $(i, j) \in L$
sb_n	Bandwidth capacity of server $n \in N$
sc_n	Computational resource capacity of server $n \in N$ (e.g., CPU)
sa_n	Activation cost of server node $n \in N$
$la_{i,j}$	Cost of using physical link $(i, j) \in L$
$lc_{i,j}$	Bandwidth capacity of physical link $(i, j) \in L$
K	Set of end-to-end service requests $k \in K$
o^k, t^k	Source and destination nodes of request $k \in K$
b^k	Bandwidth demand of request $k \in K$
τ_k	Minium acceptable reliability threshold for request $k \in K$ with $0 < \tau_k \leq 1$
F	Set of Virtual Network Function (VNF) types $f \in F$
F^k	Subset of VNFs from F ($F^k \subseteq F$) required to fulfill request $k \in K$
FC^k	Set of conflicting VNFs for request k that cannot be deployed simultaneously on the same node. $FC^k = \{\{f_1, g_1\}, \dots, \{f_m, g_m\}\}$, where each pair $f_l, g_l \in F^k, l = 1 \dots, m$ represents mutually conflicting VNFs.
δ_{nf}	Boolean parameter indicating whether virtual function f is available on node n.
$\alpha_{n,f}$	Cost to deploy a VNF instance $f \in F$ on server node $n \in N$
$\eta^k_{n,f}$	Required computational resources for an instance of VNF $f \in F$ deployed at node $n \in N$ for request $k \in K$
θ_f	Maximum traffic that one instance of VNF $f \in F$ can process
m^k_f	Number of instances of VNF $f \in F$ required to process request $k \in K$
Variable	**Description**
x^k_n	Binary variable: $x^k_n = 1$ if request $k \in k$ is routed through node $n \in N$, 0 otherwise
$y^k_{i,j}$	Binary variable: $y^k_{i,j} = 1$ if request $k \in K$ is routed through link $(i, j) \in L$, 0 otherwise
ρ^k_{fn}	Integer variable: number of instances of VNF $f \in F$ deployed on server node $n \in N$ for request $k \in K$
L_n	Total computational resources consumed by VNFs on server $n \in N$
U_n	Bandwidth consumed on server node $n \in N$
R_n	Reliability of server node $n \in N$
p^k_n	Probability that request $k \in K$ successfully reaches node $n \in N$ from its source o^k.

3. Mathematical formulation

Our goal is to determine the placement of the network function instances subject to the following constraints.
We start by addressing the routing problem. Each end-to-end request $k \in K$ must transmit its traffic along a single physical path. This path is defined using the indicator variables $y^k_{i,j}$ and x^k_n, which specify the links $(i, j) \in L$ and nodes $n \in N$ traversed, respectively. These variables are constrained by flow conservation rules:

$$\left(\sum_{\{j|(i,j)\in L\}} y^k_{i,j} - \sum_{\{j|(j,i)\in L\}} y^k_{j,i}\right) = \begin{cases} 1, & \text{if } i = o^k, \\ -1, & \text{if } i = t^k, \\ 0, & \text{otherwise} \end{cases} \qquad \forall k \in K, i \in N. \tag{1}$$

Here, the binary variable $y^k_{i,j} \in \{0, 1\}$ indicates whether request k is routed through link (i, j). The binary variable $x^k_n \in \{0, 1\}$ specifies whether node $n \in N$ belongs to the physical path of request k as enforced by the following node-based routing constraints:

$$x^k_n \leq \left(\sum_{\{j|(n,j)\in L\}} \left(y^k_{n,j} + y^k_{j,n}\right)\right) \qquad \forall k \in K, n \in N, \tag{2}$$

$$x^k_n \geq \left(y^k_{n,j} + y^k_{j,n}\right) \qquad \forall k \in K, n \in N, (n, j) \in L. \tag{3}$$

Intuitively, if the path of request $k \in K$ does not traverse node $n \in N$, then $\sum_{\{j|(n,j)\in L\}} \left(y^k_{n,j} + y^k_{j,n}\right) = 0$, forcing $x^k_n = 0$ in constraint (2). Conversely, if the path does pass through node n, then one of either $y^k_{n,j}$ or $y^k_{j,n}$ must equal 1 in constraint (3), which enforces $x^k_n = 1$. Similarly, in constraint (3), when $x^k_n = 0$, it indicates that $y^k_{n,j} + y^k_{j,n} = 0$ for all j. Hence, constraints (2) and (3) together establish the relationship between the node/switch indicators x^k_n and route indicators $y^k_{n,j}$.
The link capacity constraint is expressed as:

$$\sum_{k\in K} y^k_{i,j} * b^k \leq lc_{i,j} \qquad \forall (i, j) \in L. \tag{4}$$

This ensures that the total bandwidth allocated on each link does not exceed its capacity $lc_{i,j}$. Next, we guarantee that instances of each required $f \in F^k$ for request $k \in K$ are deployed on servers located along the request's path. Formally, the number of required instances satisfies: $m^k_f = \sum_{n\in N} \rho^k_{fn} * x^k_n$, $\forall k \in K, f \in F^k$. Since this relationship is not linear, we apply the following linearization:

$$\rho^k_{fn} \leq m^k_f * x^k_n * \delta_{nf} \qquad \forall n \in N, k \in K, f \in F^k, \tag{5}$$

$$\sum_{n\in N} \rho^k_{fn} = m^k_f \qquad \forall k \in K, f \in F^k, \tag{6}$$

where the integer variable ρ^k_{fn} denotes the number of instances of function $f \in F$ deployed on server $n \in N$ for request $k \in K$. Constraint (5) ensures that if request k traverses server n ($x^k_n = 1$), then up to m^k_f instances of VNF f can be placed there, whenever available on such node ($\delta_{nf} = 1$). Constraint (6) guarantees that the total number of deployed instances across all servers equals the demand m^k_f.

Constraints (7) ensure that virtual functions f and g cannot be deployed simultaneously on the same node n for a given request k. The set FC^k consists of all such pairs of conflicting virtual functions, as summarized in Table 1. Note that we could define a more general set FC, independent of any specific request k. This set would encode all potential incompatibilities between virtual functions across the system. This would allow us to reason about incompatibilities at the global level and then potentially filter or apply them per request as needed:

$$\frac{\rho^k_{fn}}{m^k_f} + \frac{\rho^k_{gn}}{m^k_g} \le 1 \qquad \forall n \in N, k \in K, f, g \in FC^k. \tag{7}$$

It is essential to guarantee that the total bandwidth consumed on each server node n, denoted by U_n, does not exceed its available capacity, sb_n :

$$U_n = \sum_{k\in K} \sum_{f\in F^k} \theta_f \rho^k_{fn} \qquad \forall n \in N, \tag{8}$$

$$U_n \le sb_n \qquad \forall n \in N. \tag{9}$$

Similarly, the total computational resources required to process VNFs on a physical server $n \in N$, denoted by L_n, are defined as:

$$L_n = \sum_{k\in K} \sum_{f\in F^k} \eta^k_{n,f} * \rho^k_{fn} \qquad \forall n \in N. \tag{10}$$

These computational requirements must also respect the server capacity:

$$L_n \le sc_n \qquad \forall n \in N. \tag{11}$$

The reliability function of each server/switch node $n \in N$ is defined as a non-increasing function of the total computational resources needed to handle VNFs in constraint (10).

$$R_n = H(L_n) \quad \forall n \in N. \tag{12}$$

Nodes acting purely as switches (i.e., without executing any VNFs) are assumed to be perfectly reliable. For numerical evaluation, we adopt the function $1 - \left(L_n/sc_n\right)^2$.

Since server nodes operate independently, the probability of successfully delivering an end-to-end request to its destination can be defined as the product of the reliabilities of all nodes along the chosen path. To enforce the minimum reliability threshold τ_k for each request $k \in K$, we impose:

$$\prod_{n\in N} R_n^{x^k_n} \ge \tau_k \qquad \forall k \in K. \tag{13}$$

The assumption that server reliabilities are independent and that the end-to-end success probability along a path is given by the product of the reliabilities of the traversed servers is a standard simplification in reliability-aware network design and NFV placement models. In our context, this multiplicative structure is not intended to provide an exact physical failure model of the underlying hardware and protocol stack. Rather, it serves as a tractable surrogate that captures how placement decisions and induced loads affect the relative reliability of different paths: heavily loaded nodes have reduced R_n, so paths using them become less attractive when enforcing end-to-end reliability guarantees. Congestion, routing logic, and hardware characteristics are thus reflected implicitly through the dependence of R_n on the node load (L_n). A more detailed modeling of correlated failures or protocol-specific mechanisms could be incorporated in future work by appropriately redefining the node reliability function or by complementing the product form with additional constraints. Finally, note that constraints (8), (10), and (12), together with the nonnegativity of the b^k-, m-, η-, and x-values, imply the nonnegativity of the U-, L- and R-variables.

Latency-aware formulation

While our main design objective is to balance cost and congestion-aware reliability, latency is also a critical quality-of-service dimension, especially in 5G/6G applications. In the present formulation we do not explicitly consider delay, which allows us to isolate and analyze the impact of reliability-aware placement on load balancing and resource utilization. However, the model can be extended to incorporate latency guarantees by associating each physical link $(i, j) \in L$ with a propagation and processing delay ($d_{i,j}$) as well as each node $n \in N$ with a VNF processing delay (v_n) that builds up linearly with respect to L_n irrespective of the VNF type as

$$v_n = v^{base}_n + (v^{max} - v^{base}_n) \times \frac{L_n}{sc_n}.$$

We can then impose, for each request k, a bound on the total delay along its routed path of the form:

$$\sum_{(i,j)\in L} d_{ij} y^k_{ij} + \sum_{n\in N} v_n x^k_n \le D^{\max}_k, \quad \forall k \in K.$$

Bilinear terms between bounded (integer) variables and binary variables can be linearized via standard big-M reformulations introducing auxiliary variables ($w^k_n = L_n x^k_n$). In fact, in our model, ρ (number of instances of VNF f deployed on server node n for request k) is an integer variable, and L_n (the total computational resources required to process VNFs on a physical server n) is a variable defined linearly as a function of ρ. Hence, the only non-linear element is the product $L_n x^k_n = \rho^{k'}_{f'n} x^k_n$.

Possible linearization. As stated before, we introduce an auxiliary variable $w^k_n = L_n x^k_n$. Then we impose the following constraints:

$$0 \le w^k_n \le \overline{L}_n x^k_n,$$
$$w^k_n \le L_n,$$
$$w^k_n \ge L_n - \overline{L}_n(1 - x^k_n), \quad \forall n, k$$

where $\overline{L}_n$ is a known upper bound on L_n (for example, s^c_n).

Typical values for v^{base}_n, $v^{\max}$ and $D^{\max}_k$ in 5G softwarized networks are of the order of sub-milliseconds, several milliseconds [9,10] and tens of milliseconds [11], respectively. This extension introduces an additional trade-off between reliability and latency: in some cases, routing a request on a longer but less congested path may improve end-to-end reliability at the cost of increased delay. A quantitative study of this trade-off is left to future work.

3.1. Cost modeling

Resource allocation in NFV is driven both by cost considerations and service performance. The number of deployed VNF instances plays a key role in this trade-off: deploying more VNFs improves performance but increases costs, while deploying fewer VNFs reduces costs at the risk of degrading service quality. To capture the overall cost structure, we classify expenses into three categories: capital expenditures, link activation costs, and node activation costs [1,5]. This classification provides a structured framework for assessing the economic implications of NFV deployments and supports informed decision-making in balancing cost and performance.

Capital expenditures

Deploying a VNF instance on a server node incurs costs related to standby energy consumption, licensing, image transfer, and booting processes [12,13]. These costs, referred to as capital expenditures, are denoted by $\alpha_{n,f}$ for each instance of VNF $f \in F$ deployed on server node $n \in N$. The total capital expenditure of the network is given by:

$$C_{capex} = \sum_{k\in K} \sum_{f\in F} \sum_{n\in N} \rho^k_{fn} \alpha_{n,f}. \tag{14}$$

This expression quantifies the total cost of deploying VNF instances across server nodes to satisfy service demands.

Link activation cost

Empirical studies on network power consumption show that devices consume significant energy upon activation, while the incremental energy consumption due to traffic load is relatively small [14].

Hence, we express the total link activation cost as: i io

$$C_{link} = \sum_{k\in K} \sum_{(i,j)\in L} y^k_{i,j} la_{i,j}. \tag{15}$$

This formulation captures both the energy and economic impact of activating links to support service requests. In contrast to traditional activation cost models, it also incentivizes the distribution of requests across links, thereby promoting load balancing and reducing per-link congestion.

Node activation cost

Activating network nodes also incurs costs. The cost associated with node $n \in N$ is denoted by sa_n, and reflects the expense of routing service requests through it. These costs depend on the type and capability of the node (e.g., server, switch). The total node activation cost is:

$$C_{node} = \sum_{k\in K} \sum_{n\in N} x^k_n sa_n. \tag{16}$$

Besides accounting for the energy and economic impact of activating a network node, this modeling also captures the processing overhead (e.g., memory storage), energy consumption (from processing units, cooling, and supporting hardware), and resource management overhead (such as queues management) incurred per each handled service request. Moreover, it allows consideration of heterogeneous server types, each with distinct cost structures.

Total cost function

The total cost is the sum of the above components:

$$C_{total} = C_{capex} + C_{link} + C_{node}. \tag{17}$$

Eq. (17) combines both deployment (capex) and operational (link and node activation) costs. To extend the model over a time horizon, operational costs can be scaled by a constant factor, which we normalize to unity for simplicity.

The objective of the NFV resource allocation optimization problem is to minimize C_{total}, influencing both VNF placement and routing decisions. This leads to the following MINLP formulation:

$$\min \quad C_{total} \qquad \text{(MINLP)} \tag{18}$$

s.t. (1) – (13).

Our problem is a specialized version of the integer multicommodity flow problem, which minimizes total cost while satisfying demand constraints. However, unlike standard formulations, our model incorporates function placement, reliability, and node capacity constraints. Since the integer multicommodity flow problem is $\mathcal{NP}$-complete [15], our problem inherits the same complexity. Furthermore, constraints (12) and (13) introduce non-convexity into the formulation. To guarantee a globally optimal solution, we reformulate the problem as an equivalent MILP in Section 3.2.

Comment on activation costs formulation

Note that an alternative formulation for C_{link} and C_{node} that just accounts for activating costs can be easily obtained by modifying equations (15) and (16) as follows:

- Eq. (15): $C'_{link} = \sum_{(i,j)\in L} w_{i,j} la_{i,j}$
- Eq. (16): $C'_{node} = \sum_{n\in N} z_n sa_n$

where $w_{i,j}$ and z_n are binary variables defined for every link $(i,j) \in L$ and node $n \in N$, respectively, with the following constraints:

$$w_{i,j} \geq y^k_{i,j}, \qquad \forall k \in K$$

$$z_n \geq y^k_{i,j} \qquad \forall k \in K.$$

With such alternative formulation, preliminary results show a performance similar to our proposed formulation (18), but at the expense of higher computational times and a reduced feasible region, and is therefore not further considered in the numerical evaluation.

3.2. Linearization of the mathematical model

In this section, we introduce a linearization technique inspired by the approach outlined in [16] to reformulate the non-linear terms in constraints (13). To this end, we define a new variable p^k_n, which represents the probability that request $k \in K$ successfully reaches node $n \in N$ from its source, given that node $n \in N$ lies on its path. With this definition, constraint (13) can be replaced by the following set of constraints:

$$p^k_{o^k} = R_{o^k}, \qquad \forall k \in K, \tag{19a}$$

$$p^k_n \leq R_n p^k_j + (1 - y^k_{j,n}), \qquad \forall k \in K, (j,n) \in L, \tag{19b}$$

$$p^k_n \geq \tau_k, \qquad \forall k \in K, n \in N. \tag{19c}$$

If link $(j,n) \in L$ lies on the unique path of request k from its source o^k, i.e., $y^k_{j,n} = 1$, then the probability that the data generated by o^k successfully reach node n is bounded by the probability of reaching its preceding node j multiplied by the reliability of node n. Conversely, if $y^k_{j,n} = 0$, constraint (19b) becomes redundant. The minimum acceptable probability is enforced by constraint $p^k_n \geq \tau_k$, which ensures that each request $k \in K$ satisfies its reliability threshold τ_k.

However, constraint (19b) includes the bilinear term $R_n p^k_j$ which reintroduces non-convexity. To achieve a globally optimal solution, the non-linear terms must be linearized. The linearization process is non-trivial, as neither R_n nor p^k_j are binary variables. To address this, we propose a transformation that reformulates the non-linear model into a mixed-integer linear program. Since each end-to-end request is assumed to follow a single path, the computational resources consumed on each server $n \in N$ are restricted to a finite set of values. Consequently, the reliability of each server can also only take a finite number of values, determined by the reliability function in equation (12). By utilizing the finite values for server computational resources and reliability, and employing linearization techniques from non-convex programs as described in [7,17,18], we can determine potential values for L_n for all $n \in N$ through the following two-stage approach.

- **First stage**
  The process begins with a *depth-first search* (DFS). First, the DFS is performed in the reverse direction along incoming links to identify all sources that route traffic through server n. Then, a forward DFS is executed from server n toward the destination to verify whether the destination is reachable from n. Based on these searches, we define K'_n as the set of requests for which the source can reach server n and the destination is reachable from server n. In other words, K'_n represents the set of requests for which server n is an intermediate node on the path.
- **Second stage**
  In this stage, we determine all potential values of L_n by computing every possible sum of the form $\sum_{k\in K''} \sum_{f\in F'} \eta^k_{n,f}$, for each subset $K'' \subseteq K'_n$ and $F' \subseteq F^k$. This problem can be efficiently addressed using *Dynamic Programming*, which breaks the computation into smaller subproblems. Without loss of generality, assume $K'_n = \{1, 2, \ldots, |K'_n|\}$ and $F^k = \{f_1, f_2, \ldots, f_{|F^k|}\}$. Define $w_n(q_1, q_2, s)$ as a binary variable that equals 1 if there exists a subset $\bar{K} \subseteq \{1, 2, \ldots, q_1\}$ and $\bar{F} \subseteq \{f_1, f_2, \ldots, f_{q_2}\}$ such that $\sum_{k\in\bar{K}} \sum_{f\in\bar{F}} \eta^k_{n,f} = s$, where $q_1 = 0, 1, \ldots, |K'_n|$, $q_2 = 0, 1, \ldots, |F^k|$, and $s = 0, \ldots, \min\{sc_n, \sum_{k\in K} \sum_{f\in F} \eta^k_{n,f} * m^k_f\}$. It is worth noting that

$w_n(q_1, q_2, 0) = 1$ for all q_1 and q_2 since the empty set is a subset of every set. Additionally, $w_n(q_1, q_2, s) = 0$ for any $s < 0$ because all $\eta^k_{n,f}$ are non-negative. Furthermore, $w_n(0, q_2, s) = w_n(q_1, 0, s) = 0$ for all $s > 0$ since the only subset of the empty set is itself, summing to 0. For any $q_1 \geq 1$, $q_2 \geq 1$, and $s > 0$, the recursion is:

$$w_n(q_1, q_2, s) = \max\left\{ w_n(q_1 - 1, q_2, s), w_n(q_1, q_2 - 1, s), w_n(q_1 - 1, q_2 - 1, s - \eta^{q_1}_{n,f_{q_2}}) \right\}.$$

In this recursion there are three alternatives depending on whether request q_1 or function f_{q_2} is being considered. As a result, all feasible values for L_n are associated with those values of s for which $w_n(|K'_n|, |F|, s) = 1$.

To model only discrete values for L_n, we introduce binary variables u^g_n for each $n \in N$ and $g \in \{0, \ldots, G_n\}$ where $u^g_n = 1$ if γ_g is chosen for L_n (and hence $R_n = \phi_g$), and 0 otherwise. Constraints (10) and (12) can then be reformulated as:

$$\sum_{g=0}^{G_n} \gamma_g u^g_n = \sum_{k\in K} \sum_{f\in F} \eta^k_{n,f} \rho^k_{fn}, \qquad \forall n \in N, \tag{20a}$$

$$R_n = \sum_{g=0}^{G_n} \phi_g u^g_n, \qquad \forall n \in N. \tag{20b}$$

To ensure that at most one discrete value from the sets Γ_n and Φ_n is selected for the variables L_n and R_n, respectively, we introduce the following constraints:

$$\sum_{g=0}^{G_n} u^g_n = 1, \qquad \forall n \in N, \tag{21a}$$

$$u^g_n \in \{0, 1\}, \qquad \forall n \in N, g \in \{0, \ldots, G_n\}. \tag{21b}$$

After substituting equation (20b) into constraints (19b), the resulting terms are either linear or nonlinear depending on whether they involve the product $u^g_n p^k_j$. In the nonlinear case, we encounter the product of a binary variable and a bounded continuous variable, which can be replaced by introducing a new continuous variable w^{gk}_{jn}, defined as $w^{gk}_{jn} = u^g_n p^k_j$. To enforce this equivalence, the following constraints are added, ensuring that w^{gk}_{jn} takes the value of $u^g_n p^k_j$ when the link $(j, n) \in L$ belongs to the unique path for source o^k, i.e., when $y^k_{j,n} = 1$:

$$w^{gk}_{jn} \leq u^g_n \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}, \tag{22a}$$

$$w^{gk}_{jn} \leq y^k_{j,n} \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}, \tag{22b}$$

$$w^{gk}_{jn} \leq p^k_j, \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}, \tag{22c}$$

$$w^{gk}_{jn} \geq p^k_j - (1 - u^g_n) - (1 - y^k_{j,n}), \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}. \tag{22d}$$

Let χ denote the set of (Y, X, ρ, U) satisfying constraints (1)–(9). Incorporating the above linearization, our MINLP can be reformulated as the following equivalent mixed-integer linear program:

$$\min\ C_{total} = C_{capex} + C_{link} + C_{node}$$

s.t.

$$(Y, X, \rho, U) \in \chi \tag{23a}$$

$$\sum_{g=0}^{G_n} \gamma_g u^g_n = \sum_{k\in K} \sum_{f\in F} \eta^k_{n,f} \rho^k_{fn}, \qquad \forall n \in N, \tag{23b}$$

$$\sum_{g=0}^{G_n} \gamma_g u^g_n \leq sc_n, \qquad \forall n \in N \tag{23c}$$

$$R_n = \sum_{g=0}^{G_n} \phi_g u^g_n, \qquad \forall n \in N \tag{23d}$$

$$\sum_{g=0}^{G_n} u^g_n = 1, \qquad \forall n \in N, \tag{23e}$$

$$p^k_{o^k} = R_{o^k}, \qquad \forall k \in K, \tag{23f}$$

$$p^k_n \leq \sum_{g=0}^{G_n} \phi_g w^{gk}_{jn} + (1 - y^k_{j,n}), \qquad \forall k \in K, (j, n) \in L, \tag{23g}$$

$$p^k_n \geq \tau_k, \qquad \forall k \in K, n \in N, \tag{23h}$$

$$w^{gk}_{jn} \leq u^g_n \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}, \tag{23i}$$

$$w^{gk}_{jn} \leq y^k_{j,n} \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}, \tag{23j}$$

$$w^{gk}_{jn} \leq p^k_j, \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}, \tag{23k}$$

$$w^{gk}_{jn} \geq p^k_j - (1 - u^g_n) - (1 - y^k_{j,n}), \qquad \forall k \in K, (j, n) \in L, g \in \{0, \ldots, G_n\}. \tag{23l}$$

$$u^g_n \in \{0, 1\}, \qquad \forall n \in N, g \in \{0, \ldots, G_n\}. \tag{23m}$$

As observed, the size of the MILP problem depends on the value of G_n. For large values of G_n, solving the MILP directly may be computationally prohibitive. Therefore, in Section 4, we introduce an alternative solution methodology based on a cutting-plane approach, designed to handle large-scale instances efficiently.

Discussion of the second stage's computational complexity

The computational complexity of the second stage is $\mathcal{O}(|K| \min\{sc_n, \sum_{k\in K} \sum_{f\in F} \eta^k_{n,f} * m^k_f\})$. In fact, this stage can be viewed as a variant of the *Subset Sum Problem*, where instead of verifying the existence of a specific target sum, we compute all achievable sums.[1] Since the maximum possible subset sum is given by $min\{sc_n, \sum_{k\in K} \sum_{f\in F} \eta^k_{n,f} \cdot m^k_f\}$, it is bounded by sc_n and in practice remains relatively small,[2] as in [19]. To address this, we define a Boolean Dynamic Programming (DP) table (denoted as w_n) to store all achievable subset sums. We then iterate over all elements in K, ensuring that we efficiently track possible sums without needing to store explicit subsets.

In fact, the goal is to avoid the exponential storage required by enumerating all possible subsets. A naive method would be to explicitly list every subset of requests and VNFs, which requires $\mathcal{O}(2^{|K|\cdot|F|})$ space and time in the worst case. In contrast, we use a Boolean DP table that only records whether a given total computational resource consumption s is achievable. Therefore, we define $w_n(q_1, q_2, s)$ as follows:

- $w_n(q_1, q_2, s) = 1$ if, using the first q_1 requests and the first q_2 VNFs, there exists a subset whose total demand equals s.
- $w_n(q_1, q_2, s) = 0$ otherwise.

In other words, $w_n(q_1, q_2, s) = 1$ means that the value s can be achieved using the first q_1 requests and the first q_2 VNFs, while $w_n(q_1, q_2, s) = 0$ means it cannot be achieved. Note that the DP table does not store subsets themselves but only feasibility information (i.e., the achievable sums). The table is filled recursively: for each new element, either we exclude it (inheriting feasibility from the previous state) or we

[1] The Subset Sum Problem is NP-complete and enumerating all subsets is exponential in general. However, when the largest possible subset sum is not too large, a pseudo-polynomial time dynamic programming (DP) algorithm can solve it in $\mathcal{O}(nS)$, where n is the number of items and S is the maximum sum [19].

[2] Our numerical results confirm this, showing that even in the worst-case scenario, the maximum is 16, while the minimum is 2. Hence, the set of all possible subset sums consists only of even numbers between 2 and 16, depending on sc_n which is computationally manageable.

include it (adding its demand value to an already feasible sum). With this construction, the required storage and computations are reduced from exponential ($\mathcal{O}(2^{|K|\cdot|F|})$) to pseudo-polynomial, namely $\mathcal{O}\Big(|K|\cdot \min\Big\{sc_n, \sum_{k\in K}\sum_{f\in F} n^k_{n,f}\cdot m^k_f\Big\}\Big)$. This efficiency comes from the fact that we only need to store whether each sum up to the capacity bound sc_n is achievable, instead of explicitly storing all subsets.

3.3. Service-chain ordering

In the formulation presented so far, each request $k \in K$ is associated with a set F_k of VNFs and a required number of instances m^k_f for each $f \in F_k$, but no explicit ordering is imposed among VNFs. In many NFV deployments, however, service chains must respect precedence relations (e.g., DPI before Firewall, Firewall before Proxy). According to what is proposed in [20], we propose the following ordering for VNFs: 1) NAT, 2) Firewall, 3) IDS and 4) Proxy. Furthermore, we show how the model can be extended to support such ordering constraints without altering its mixed-integer linear structure.

For each request $k \in K$ we define the set $O_k \subseteq F_k \times F_k$ of ordered VNF pairs, where $(f,g) \in O_k$ indicates that VNF f must be executed before VNF g along the path of request k (from o_k to t_k). To capture where each VNF is executed, we introduce binary variables

$$z^k_{fn} = \begin{cases} 1, & \text{if at least one instance of VNF } f \text{ for} \\ & \text{request } k \text{ is deployed on node } n, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall k \in K,\ f \in F_k,\ n \in N.$$

These variables are linked to the existing routing and placement variables as follows:

$$z^k_{fn} \le x^k_n, \qquad \forall k \in K,\ f \in F_k,\ n \in N, \tag{24}$$

$$\rho^k_{fn} \le m^k_f\, z^k_{fn}\, \delta_{nf}, \qquad \forall k \in K,\ f \in F_k,\ n \in N, \tag{25}$$

where x^k_n indicates whether request k uses node n, ρ^k_{fn} denotes the number of instances of VNF f deployed on node n for request k, and δ_{nf} is the deployment indicator of VNF f on node n. Constraint (24) ensures that a VNF can only be hosted at nodes that belong to the path of the request, while (25) forbids allocating VNF instances on nodes where either $z^k_{fn} = 0$ or the VNF is unavailable ($\delta_{nf} = 0$). Together with the original demand constraint

$$\sum_{n\in N} \rho^k_{fn} = m^k_f, \qquad \forall k \in K,\ f \in F_k,$$

this preserves the possibility of distributing the m^k_f instances of VNF f across multiple nodes, as in the base model.

To reason about the position of each node along the path of request k, we introduce hop-index variables h^k_n, which take integer values bounded by $|N|-1$:

$$0 \le h^k_n \le |N|-1, \qquad \forall k \in K,\ n \in N, \tag{26}$$

and we assign hop index zero to the source node:

$$h^k_{o_k} = 0, \qquad \forall k \in K. \tag{27}$$

For each directed link $(i,j) \in L$, we impose

$$h^k_j \ge h^k_i + 1 - M\big(1 - y^k_{ij}\big), \qquad \forall k \in K,\ (i,j) \in L, \tag{28}$$

where M is a sufficiently large constant. If link (i,j) is used by request k (i.e., $y^k_{ij} = 1$), then (28) enforces $h^k_j \ge h^k_i + 1$, so hop indices strictly increase along the path. When $y^k_{ij} = 0$, constraint (28) becomes non-binding.

We now define, for each request k and each VNF type $f \in F_k$, the *last* and *first* hop indices at which instances of f are executed. Intuitively,

$$H^{\max,k}_f = \max\{h^k_n : z^k_{fn} = 1,\ n \in N\},\quad H^{\min,k}_f = \min\{h^k_n : z^k_{fn} = 1,\ n \in N\},$$

so that $H^{\max,k}_f$ and $H^{\min,k}_f$ capture, respectively, the farthest and earliest positions at which VNF f is hosted along the path of request k.

To linearize these definitions, we introduce binary selectors ℓ^k_{fn} and u^k_{fn} such that:

$$\ell^k_{fn} = \begin{cases} 1, & \text{if node } n \text{ is selected as the last node hosting VNF } f \text{ for} \\ & \text{request } k, \\ 0, & \text{otherwise,} \end{cases}$$

$$u^k_{fn} = \begin{cases} 1, & \text{if node } n \text{ is selected as the first node hosting VNF } f \text{ for} \\ & \text{request } k, \\ 0, & \text{otherwise,} \end{cases}$$

for all $k \in K, f \in F_k, n \in N$. Since $m^k_f \ge 1$ by assumption, each VNF must have at least one hosting node, hence a first and a last node. We impose:

$$\ell^k_{fn} \le z^k_{fn}, \qquad \forall k \in K,\ f \in F_k,\ n \in N, \tag{29}$$

$$\sum_{n\in N} \ell^k_{fn} = 1, \qquad \forall k \in K,\ f \in F_k, \tag{30}$$

$$u^k_{fn} \le z^k_{fn}, \qquad \forall k \in K,\ f \in F_k,\ n \in N, \tag{31}$$

$$\sum_{n\in N} u^k_{fn} = 1, \qquad \forall k \in K,\ f \in F_k. \tag{32}$$

The variables $H^{\max,k}_f$ and $H^{\min,k}_f$ are then linked to the hop indices as follows:

$$H^{\max,k}_f \ge h^k_n - M\big(1 - z^k_{fn}\big), \qquad \forall k \in K,\ f \in F_k,\ n \in N, \tag{33}$$

$$H^{\max,k}_f \le h^k_n + M\big(1 - \ell^k_{fn}\big), \qquad \forall k \in K,\ f \in F_k,\ n \in N, \tag{34}$$

$$H^{\min,k}_f \le h^k_n + M\big(1 - z^k_{fn}\big), \qquad \forall k \in K,\ f \in F_k,\ n \in N, \tag{35}$$

$$H^{\min,k}_f \ge h^k_n - M\big(1 - u^k_{fn}\big), \qquad \forall k \in K,\ f \in F_k,\ n \in N. \tag{36}$$

Constraint (33) ensures that $H^{\max,k}_f$ is at least the hop index of every node hosting VNF f (i.e., with $z^k_{fn} = 1$), while (34) pins $H^{\max,k}_f$ to the hop index of the unique selected last node with $\ell^k_{fn} = 1$. Similarly, constraint (35) ensures that $H^{\min,k}_f$ is no larger than the hop index of any node hosting VNF f, and (36) ties $H^{\min,k}_f$ to the hop index of the unique selected first node with $u^k_{fn} = 1$. Taken together, these constraints enforce

$$H^{\max,k}_f = \max\{h^k_n : z^k_{fn} = 1, n \in N\},\quad H^{\min,k}_f = \min\{h^k_n : z^k_{fn} = 1, n \in N\},$$

for each request k and VNF f.

We can now impose the desired service-chain ordering. For each request $k \in K$ and each ordered pair $(f,g) \in O_k$, we require that every occurrence of VNF f along the path of k strictly precede every occurrence of VNF g. This is enforced by the linear constraint

$$H^{\max,k}_f + 1 \le H^{\min,k}_g, \qquad \forall k \in K,\ (f,g) \in O_k. \tag{37}$$

Since $H^{\max,k}_f$ and $H^{\min,k}_g$ represent, respectively, the last hop at which VNF f is hosted and the first hop at which VNF g is hosted for request k, constraint (37) guarantees that all nodes hosting f are placed before all nodes hosting g along the path. This extension preserves the linear structure of the overall formulation and can be activated whenever explicit service-chain ordering is required.

Illustrative example with distributed VNF instances

To illustrate how the proposed extended formulation handles distributed VNF instances while enforcing a strict service-chain order, we consider a simple NFV infrastructure with four nodes and a single directed path (see Fig. 1):

$$N = \{1,2,3,4\}, \qquad L = \{(1,2),(2,3),(3,4)\}.$$

We consider a single request k with source $o_k = 1$ and destination $t_k = 4$. The request is routed along the path

$$1 \to 2 \to 3 \to 4,$$

so that $y^k_{12} = y^k_{23} = y^k_{34} = 1$ and $x^k_n = 1$ for all $n \in \{1, 2, 3, 4\}$.

The request requires two VNFs: DPI and Proxy. Let f_1 denote DPI and f_2 denote Proxy, so that $F_k = \{f_1, f_2\}$, and assume that the service chain must execute DPI before Proxy. Formally,

$$O_k = \{(f_1, f_2)\}.$$

To highlight the possibility of distributing instances across multiple nodes, we set

$$m^k_{f_1} = 2, \qquad m^k_{f_2} = 1.$$

Using constraints (26)–(28), we assign hop indices along the path as

$$h^k_1 = 0, \quad h^k_2 = 1, \quad h^k_3 = 2, \quad h^k_4 = 3.$$

Feasible distributed placement. Suppose that the two instances of DPI are placed on nodes 2 and 3, while Proxy is placed on node 4. A feasible placement that satisfies the demand constraints is then

$$\rho^k_{f_1,2} = 1, \quad \rho^k_{f_1,3} = 1, \quad \rho^k_{f_2,4} = 1,$$

with all other $\rho^k_{f_i,n} = 0$. In particular,

$$\sum_{n\in N} \rho^k_{f_1,n} = 2 = m^k_{f_1}, \qquad \sum_{n\in N} \rho^k_{f_2,n} = 1 = m^k_{f_2},$$

so that the required number of instances for each VNF is satisfied.

By constraint (25), we must have $z^k_{f_1,2} = z^k_{f_1,3} = 1$ and $z^k_{f_2,4} = 1$, while all other $z^k_{f_i,n} = 0$. The sets of nodes hosting each VNF are therefore

$$\{n \in N : z^k_{f_1,n} = 1\} = \{2, 3\}, \qquad \{n \in N : z^k_{f_2,n} = 1\} = \{4\}.$$

We now choose the selector variables so that node 2 is the first node hosting f_1 and node 3 is the last node hosting f_1, while node 4 is both the first and last node hosting f_2:

$$u^k_{f_1,2} = 1, \quad \ell^k_{f_1,3} = 1, \qquad u^k_{f_2,4} = 1, \quad \ell^k_{f_2,4} = 1,$$

and all other $\ell^k_{f_i,n} = u^k_{f_i,n} = 0$. By constraints (33)–(36), we obtain

$$H^{\min,k}_{f_1} = h^k_2 = 1, \qquad H^{\max,k}_{f_1} = h^k_3 = 2,$$
$$H^{\min,k}_{f_2} = H^{\max,k}_{f_2} = h^k_4 = 3.$$

The ordering constraint (37) for the pair $(f_1, f_2) \in O_k$ then reads

$$H^{\max,k}_{f_1} + 1 \;\le\; H^{\min,k}_{f_2} \quad\Longleftrightarrow\quad 2 + 1 \le 3,$$

which is satisfied. Thus, even though DPI is distributed over two different nodes (2 and 3), and Proxy is placed on a third node (4), the model correctly enforces that all DPI processing precedes any Proxy processing along the path.

Infeasible placement with reversed order. Consider instead a placement where the two DPI instances are still hosted on nodes 2 and 3, but Proxy is hosted on node 2, i.e.,

$$\rho^k_{f_1,2} = 1, \quad \rho^k_{f_1,3} = 1, \quad \rho^k_{f_2,2} = 1,$$

and all other $\rho^k_{f_i,n} = 0$. In this case,

$$\{n \in N : z^k_{f_1,n} = 1\} = \{2, 3\}, \qquad \{n \in N : z^k_{f_2,n} = 1\} = \{2\}.$$

A consistent choice of selectors is to pick node 2 as both the first and last node hosting f_2, and node 2 as the first and node 3 as the last node hosting f_1:

$$u^k_{f_1,2} = 1, \quad \ell^k_{f_1,3} = 1, \qquad u^k_{f_2,2} = 1, \quad \ell^k_{f_2,2} = 1.$$

The corresponding hop indices become

$$H^{\min,k}_{f_1} = h^k_2 = 1, \qquad H^{\max,k}_{f_1} = h^k_3 = 2,$$
$$H^{\min,k}_{f_2} = H^{\max,k}_{f_2} = h^k_2 = 1.$$

The ordering constraint (37) then reads

$$H^{\max,k}_{f_1} + 1 \;\le\; H^{\min,k}_{f_2} \quad\Longleftrightarrow\quad 2 + 1 \le 1,$$

which is violated. Intuitively, placing Proxy at node 2 means that some traffic would be processed by Proxy before being fully processed by DPI, which contradicts the required service-chain order $f_1 \prec f_2$. This placement is therefore correctly declared infeasible by the model.

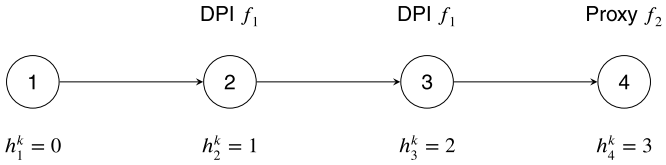


Fig. 1. Illustrative example with distributed DPI instances. Request k is routed along the path $1 \to 2 \to 3 \to 4$ and must execute DPI f_1 before Proxy f_2. Two instances of DPI are placed at nodes 2 and 3, while Proxy is placed at node 4. The hop indices $(h^k_1, h^k_2, h^k_3, h^k_4) = (0, 1, 2, 3)$ encode the position of each node along the path. In the placement shown, the ordering constraint $H^{\max,k}_{f_1} + 1 \le H^{\min,k}_{f_2}$ is satisfied, so all DPI processing precedes any Proxy processing along the path.

4. Cutting plane-based heuristic

As previously discussed, the proposed model is NP-Hard, and the exact formulation presented in Section 3.2 may become impractical for large-size network instances. To address this issue, we introduce a cutting-plane-based heuristic designed to obtain an optimal solution to our problem. As observed previously, one of the main challenges in solving the MINLP problem lies in handling constraints (12) and (13). Hence, the proposed heuristic leverages reformulated lower- and upper-bounding models for these constraints, allowing the problem to be approximated efficiently. The algorithm then iteratively refines these bounds by adding cutting planes, thereby converging toward the optimal solution.

4.1. Lower and upper-bounding models

To tackle the complexity of the original MINLP model (18), we design two bounding formulations: a lower-bounding model and an upper-bounding model. Both formulations approximate the cumulative computational resources required at each server node and translate them into corresponding reliability values. These bounding models serve as relaxations or approximations of the original formulation, and their objective values provide valid lower and upper bounds for the original problem. The construction of both models follows the same general idea:

- First, we replace constraint (13) with constraints (19a), (19b), and (19c).
- Then, we approximate the actual cumulative load L_n of each server node n by rounding it either downward (lower-bounding model) or upward (upper-bounding model).
- Finally, the estimated value of the node reliability R_n is substituted into the MINLP model (18), leading to two bounding formulations that can be linearized using the same technique introduced in Section 3.2.

The details of the construction are presented separately for the lower- and upper-bounding formulations.

Lower-Bounding Formulation

For the lower-bounding formulation, we begin by identifying $\mu_n + 1$ possible sums of the form $\sum_{k\in K} \sum_{f\in F^k} \rho^k_{fn}$, which represent potential values for the cumulative computational resources L_n of server n. These values are collected in the ordered set $\{d_0, ..., d_{\mu_n}\}$, where $d_g < d_{g+1}, \forall g = 0, ..., \mu_n - 1$. It is evident that for each server n, the smallest possible value of d_g is 0, and since each request visits every server at most once, the largest possible is $\sum_{k\in K} \sum_{f\in F^k} \rho^k_{fn}$. Thus, the actual cumulative computational resources required to process VNFs of server n lie within the interval $[d_0, d_{\mu_n}]$, where $d_0 = 0$ and $d_{\mu_n} = \sum_{k\in K} \sum_{f\in F^k} \rho^k_{fn}$. The corresponding value of R_n can then be computed using the reliability function in equation (12), evaluated at each d_g, yielding the set $\{H_0, ..., H_{\mu_n}\}$. Notably, $H_0 = 1$, and since H is non-increasing in L_n, we have $H_g \ge H_{g+1}$ for $g = 0, ..., \mu_n - 1$.

Next, we approximate L_n by rounding it down to the nearest value in $\{d_0, ..., d_{\mu_n}\}$ that does not exceed the actual load of server n. To achieve this, we introduce a binary variable $\tilde{u}_n^g$, which equals 1 if $L_n \geq d_g$, and 0 otherwise, for each $n \in N$ and $g \in \{1, 2, \dots, \mu_n\}$. This relationship is enforced by the following set of constraints:

$$\tilde{u}_n^g \geq \frac{L_n - d_g + 1}{\sum_{k\in K}\sum_{f\in F^k} \rho_{fn}^k - d_g + 1}, \qquad \forall n \in N, g \in \{1, 2, \dots, \mu_n\}. \tag{38}$$

Accordingly, the estimated reliability for the lower-bounding problem, denoted by R_n^L, can be computed as:

$$R_n^L = 1 + \sum_{g=1}^{\mu_n} \tilde{u}_n^g * (H_g - H_{g-1}) \tag{39}$$

The lower-bounding model is then obtained by modifying the original MINLP formulation (18) as follows:

- Replace constraint (13) with constraints (19a), (19b), and (19c),
- Substitute R_n in the MINLP model (18) with R_n^L,
- Add constraint (38) to the model,
- Replace constraint (12) with equation (39),
- Impose binariness restrictions on the $\tilde{u}$ -variables.

Since L_n is rounded downward, the estimated reliability R_n tends to overestimate the actual reliability of each node n. Consequently, if constraints (19a), (19b), and (19c) are fulfilled based on the reliability values of server nodes, they should also be satisfied using the overestimated reliability values. Also, each feasible solution of the original MINLP model (18) is a feasible solution of the lower-bounding problem. Therefore, the feasible region of the lower-bounding problem is a relaxation of that of model (18), and the objective value of the lower-bounding solution provides a valid lower bound for the original problem.

Upper-Bounding Formulation

Similarly, the upper-bounding model is obtained by rounding L_n upward. To this end, the definition of the auxiliary binary variables $\tilde{u}_n^g$ is revised as follows:

$$\tilde{u}_n^g = \begin{cases} 1, & \text{if} L_n > d_g \\ 0, & \text{otherwise}, \end{cases} \qquad \forall n \in N, g \in \{0, \dots, \mu_n - 1\}.$$

This condition is enforced by the following forcing constraints:

$$\tilde{u}_n^g \geq \frac{L_n - d_g}{\sum_{k\in K}\sum_{f\in F^k} \rho_{fn}^k - d_g}, \qquad \forall n \in N, g \in \{1, 2, \dots, \mu_n\}. \tag{40}$$

The estimated reliability value in the upper-bounding model is then computed as:

$$R_n^U = 1 + \sum_{g=0}^{\mu_n - 1} \tilde{u}_n^g * (H_{g+1} - H_g). \tag{41}$$

Accordingly, the upper-bounding model is obtained from the MINLP (18) by applying the following modifications:

- Replace constraint (13) with constraints (19a), (19b), and (19c),
- Substitute R_n in the MINLP model (18) with R_n^U,
- Add constraint (40) to the model,
- Replace constraint (12) with equation (41),
- Impose binariness restrictions on the $\tilde{u}$ -variables.

The lower- and upper-bounding formulations can both be linearized using the same technique introduced in Section 3.2.

Cutting Plane-Based Iterative Method

To find an optimal solution for the original MINLP model (18) we design an iterative method based on a cutting plane algorithm applied to the lower-bounding formulation. The procedure can be summarized as follows:

- We first construct the lower bounding model using the subsets $\{d_0, ..., d_{\mu_n}\}$ and $\{H_0, ..., H_{\mu_n}\}$ and obtain an integer-feasible solution, $\left(\overline{Y}, \overline{X}, \overline{\rho}, \overline{U}\right)$, using the branch-and-bound method [21].
- We compute the actual values of L_n and R_n using equations (10) and (12), respectively.
- We evaluate the reliability of each end-to-end request $k \in K$ and node $n \in N$, and identify any requests that fail to meet their reliability threshold τ_k. Since the lower-bounding problem is a relaxation of the original MINLP, the actual reliability of some paths may fall below τ_k. We denote the set of such violated requests as $\bar{K} \subset K$.
  - If $\bar{K} = \emptyset$, all requests satisfy the reliability constraints, and the integer-feasible solution of the lower-bounding problem is also feasible for the full model (1)–(13).
  - If $\bar{K} \neq \emptyset$, then the reliability of some requests $k \in K$ falls short of τ_k, and the current solution must be refined.
- We apply a cutting plane to exclude the current infeasible solution of the lower-bounding model, as discussed in Section 4.2. In addition, we adopt a *Warm-Start strategy* (Section 4.3) to accelerate convergence and improve the quality of the initial valid inequalities. For this purpose, we define two parameters, LB and UB, which represent the lower and upper bounds of the objective function, respectively:
  - The initial value of LB, denoted by LB', is obtained from the objective value of the linear relaxation of the MILP reformulation presented in Section 3.2.
  - The initial value of UB, denoted by UB', is derived under the assumptions that $y_{i,j}^k$ and x_n^k are binary variables , and that ρ_{fn}^k is an integer variable bounded by m_f^k. Thus, UB' can be expressed as:

$$UB' = \sum_{k\in K}\sum_{f\in F}\sum_{n\in N} m_f^k \alpha_{n,f} + \sum_{k\in K}\sum_{(i,j)\in L} la_{i,j} + \sum_{k\in K}\sum_{n\in N} sa_n. \tag{42}$$

An algorithmic description of the heuristic is presented in Algorithm 1.

Algorithm 1 Cutting plane-based heuristic for the proposed problem.

Input: All parameters, $\bar{K} = \emptyset$, tolerance ϵ, initial bounds $LB = LB'$ and $UB = UB'$.
Output: Optimal reliable design.
Formulate lower and upper bounding models using subsets $\{d_0, \dots, d_{\mu_n}\}$ and $\{H_0, \dots, H_{\mu_n}\}$.
Add initial cuts (44)–(49) to both models.
while $UB - LB \geq \epsilon$ **do**
  for each $n \in \mathcal{N}$ **do**
    Compute $\overline{L_n}$ and $\overline{R_n}$ using Eqs. (10) and (12).
  end for
  Set $\bar{K} = \emptyset$.
  for each $k \in K$ **do**
    if $\prod_{n\in\mathcal{N}} R_n^{x_n^k} < \tau_k$ **then**
      $\bar{K} \leftarrow \bar{K} \cup \{k\}$.
    end if
  end for
  if $\bar{K} = \emptyset$ **then**
    Return solution with current bounds LB, UB.
  else
    for each $k \in \bar{K}$ **do**
      Add cut (43) to both models.
    end for
  end if
end while

4.2. Valid inequality

Consider the case where a subset of end-to-end requests cannot be assigned with a *reliable* path, i.e., $\bar{K} \neq \emptyset$. In such situations, it becomes necessary to introduce a valid inequality that excludes the current solution from the feasible region of the lower-bounding problem and rectifies the violation. Adjusting a violated path can involve at least one of the following two actions:

- **Path modification**: modify the violated path of the end-to-end request $k \in \bar{K}$ from its origin o^k to its destination t^k.
- **Resource adjustment**: reduce the total computational resources allocated to physical servers along the violating path.

To formalize this, for each request $k \in \bar{K}$, we define the following sets:

- $\tilde{L}^k$: the set of links used by request k from its source o^k to its destination t^k. Formally, $\tilde{L}^k = \{(i,j) \in L | \overline{y^k_{i,j}} = 1\}$, where $\overline{y^k_{i,j}}$ of the corresponding decision variable $y^k_{i,j}$ in the current solution of the lower-bounding problem.
- P^k: the set of physical servers located along the violated path of request k. Specifically, P^k comprises nodes n for which the variable x^k_n in the current solution of the lower-bounding problem has a value greater than zero. This can be expressed as $P^k = \{n | \overline{x^k_n} > 0\}$, where $\overline{x^k_n}$ represents the value of the variable x^k_n in the current solution of the lower-bounding problem.
- $\bar{F}^k$: the set of all virtual functions $f \in F^k$ deployed on servers in P^k whose associated $\overline{\rho}$-variables equal 1 in the current solution. Formally, $\bar{F}^k = \{f | f \in F^k, n \in P^k, \overline{\rho^k_{fn}} = 1\}$, where $\overline{\rho^k_{fn}}$ is the value of variable ρ^k_{fn} in the current solution of the lower-bounding problem.

To adjust a violated path for request $k \in \bar{K}$, at least one of the variables ρ^k_{fn} or $y^k_{i,j}$ must be forced to zero in the current solution. This requirement is captured by the following inequality:

$$\sum_{f \in \bar{F}^k} (1 - \rho^k_{fn}) + \sum_{\{(i,j) \in L\}} (1 - y^k_{i,j}) \geq 1. \tag{43}$$

Inequality (43) constitutes a valid cut for the MINLP model (18). Since request k is a violating request, if the left-hand side of inequality (43) equals zero, then request k follows exactly the same path as before, and the total computational resources on every physical server in P^k are at least as high as they were in the previous solution. Consequently, such a solution should be infeasible for model (18). Therefore, inequality (43) is a valid constraint that eliminates the current infeasible solution of the lower-bounding problem, as its left-hand side evaluates to zero in that solution.

4.3. Warm-start strategy

Slow convergence in the early iterations may result from the generation of low-quality valid inequalities. To address this issue, we implement a warm-start strategy that generates an initial set of tight cuts. The warm-start includes two families of inequalities that exploit basic flow and capacity properties of the network: connectivity constraints and cover constraints.

- **Connectivity constraints**: According to the flow conservation principle (constraint (1)) for any intermediate node $n \in N - \{o^k, t^k\}$, the presence of any incoming link for request k implies at least one outgoing link, and vice versa. This requirement yields the following connectivity constraints:

$$y^k_{n,j} \leq \sum_{\{h|(h,n) \in L\}} y^k_{h,n}, \qquad \forall k \in K, (n,j) \in L, n \in N - \{o^k, t^k\}, \tag{44}$$

$$y^k_{j,n} \leq \sum_{\{n|(n,h) \in L\}} y^k_{n,h}, \qquad \forall k \in K, (j,n) \in L, n \in N - \{o^k, t^k\}. \tag{45}$$

- **Cover constraints**: Cover inequalities enforce that, for each source-destination pair, the total bandwidth leaving the source or entering the destination is at least the request bandwidth b^k:

$$\sum_{\{j|(o^k,j) \in L\}} lc_{o^k,j} y^k_{o^k,j} \geq b^k, \quad \forall k \in K, \tag{46}$$

$$\sum_{\{j|(j,t^k) \in L\}} lc_{j,t^k} y^k_{j,t^k} \geq b^k, \quad \forall k \in K. \tag{47}$$

Additionally, if for a node $n \in N$, we have $\{k \in K | n = o^k\} \neq \emptyset$, then the aggregate outgoing capacity from n must cover the sum of their bandwidth demands:

$$\sum_{\{j|(o^k,j) \in L\}} lc_{o^k,j} y^k_{o^k,j} \geq \sum_{k \in \{k \in K | n = o^k\}} b^k. \tag{48}$$

Analogously, if we have $\{k \in K | n = t^k\} \neq \emptyset$, the following constraint is considered:

$$\sum_{\{j|(j,t^k) \in L\}} lc_{j,t^k} y^k_{j,t^k} \geq \sum_{k \in \{k \in K | n = t^k\}} b^k. \tag{49}$$

5. LB-greedy: Load-balanced greedy heuristic

As an additional approach, we consider a simple load-balanced greedy heuristic, denoted LB-Greedy. LB-Greedy is designed to approximate reliability-awareness through load balancing, without explicitly computing end-to-end reliabilities or performing any local search.

The heuristic processes the set of requests K sequentially and maintains, at each step, the current load L_n on every node $n \in N$. The load-dependent reliability model used in the MILP implies that nodes with lower normalized load L_n/s^c_n are more reliable: LB-Greedy exploits this by penalizing heavily loaded nodes during both path selection and VNF placement. The algorithm proceeds as follows:

1. Initialize all routing variables y^k_{ij}, placement variables ρ^k_{fn}, and node loads L_n to zero. Compute residual capacities for all nodes and links.
2. Fix an order in which to process the requests (e.g., non-increasing bandwidth demand b_k, or simply a random permutation).
3. For each request k in the chosen order:
   (a) **Path selection.** For each request k, we first build a residual graph that only contains links with enough remaining bandwidth to carry k. Let r_{ij} denote the residual capacity of link (i,j) at the current iteration, and define
   $$r_{ij} = l^c_{ij} - \sum_{k' \in K_{\text{served}}} b_{k'}\, y^{k'}_{ij},$$
   where K_{served} is the set of requests already routed and placed. We then restrict the set of links to
   $$L_k = \{(i,j) \in L : r_{ij} \geq b_k\},$$
   and run a shortest-path algorithm on the reduced graph $G_k = (N, L_k)$ using load-aware link weights
   $$w_{ij} = l^a_{ij} + \beta \frac{1}{2}\left(\frac{L_i}{s^c_i} + \frac{L_j}{s^c_j}\right), \qquad (i,j) \in L_k,$$
   where $\beta > 0$ is a tuning parameter and L_n/s^c_n is the normalized load of node n. If no path exists between the source and destination of request k in G_k, the request is marked as unsatisfied and the algorithm proceeds to the next one. Otherwise, the resulting path P_k is selected and the corresponding routing variables y^k_{ij} are set to 1 for all $(i,j) \in P_k$.
   (b) Set $y^k_{ij} = 1$ for all links $(i,j) \in P_k$, and mark all nodes on P_k as used by request k.
   (c) **Greedy VNF placement.** For each VNF $f \in F_k$ and for each required instance of f:
      i. Consider the set of nodes on P_k that are allowed to host f and have sufficient residual processing and memory capacity for request k.

ii. For each candidate node n in this set, compute:
- the incremental cost $\Delta C(n)$ of placing the instance on n, including the per-node activation cost if this is the first VNF placed on n for any request;
- the normalized load L_n/s_n^c before placement.

iii. Define a simple score

$$S_n = \Delta C(n) + \gamma \frac{L_n}{s_n^c},$$

where $\gamma > 0$ is a load-penalty parameter. Select the node $n^\star$ that minimizes S_n.

iv. Place the instance of f on $n^\star$, update $\rho^k_{fn^\star}$, update the load $L_{n^\star}$ and the residual capacities.

v. If no feasible node exists for a given instance (i.e., all candidate nodes violate capacity constraints), mark request k as unsatisfied and optionally remove its routing and previously placed VNFs.

4. After all requests have been processed, compute the total cost and the end-to-end reliability Rel_k for each request k using the same definitions as in the MILP model.

Although LB-Greedy does not explicitly optimize end-to-end reliability, it indirectly promotes reliability by routing and placing VNFs on less loaded nodes, which are more reliable under the load-dependent failure model. This makes LB-Greedy a stronger baseline than purely unreliable or locally reliable schemes, while remaining extremely easy to implement and significantly faster than both the exact MILP and more elaborate metaheuristic.

Note that, in all heuristic experiments, we set the load-aware parameters β and γ so that the load terms in the path and placement scores have the same order of magnitude as the corresponding economic costs. More details on the tuning of such parameters are provided in Section 6.

6. Numerical results

In this section, we evaluate the performance of the proposed optimization model (23) (namely, *End-to-End Reliable Model*, ***E***). First, we compare its performance with those obtained in our previous work [22], where each VNF instance is able to process the same amount of traffic, requires the same computational resources, and is not subject to either deployment or incompatibility constraints. We refer to this latter baseline as *Homogeneous VNF Model* (***H***). Then, we compare the performance of model ***E*** against two baseline approaches, designed to partially or fully relax the end-to-end service reliability constraint, i.e., constraint (13). Specifically, we define the two following baselines:

1. *Unreliable Model* (***U***): this strategy ignores constraints (13), i.e., it removes the end-to-end reliability requirements, enforcing only node capacity limits.
2. *Locally Reliable Model* (***L***): it imposes reliability locally at each node rather than across the entire service chain. In practice, constraints (13) are substituted by:

$$R_n^{x_n^k} \geq \tau_k, \qquad \forall k \in K, n \in N. \tag{50}$$

Finally, in addition to the considered baseline approaches, we consider the simple load-balanced greedy heuristic proposed in Section 5, and compare the corresponding performance with those obtained by our proposed approach.

The network topologies, link capacities and associated traffic demands (i.e., source-destination pairs – o^k and t^k – and flow amounts – b^k) are extracted from the SNDLib database [23], a key resource in network design research. In the following, we describe the experimental setup, the service reliability model, and the metrics used for the performance evaluation.

Table 2
Parameters setup.

Nodes [22]		
Cost [$] ($sa_n$): $\mathcal{U}([3000, 5000])$		
Network Node	**CPU Capacity** [#] (sc_n)	**Processing Capacity** [$Gbps$] (sb_n)
Server	8	40
SmartNIC	12	80
PISA	16	160
Links [22,23]		
Capacity [$Gbps$] ($lc_{i,j}$): As from [23]		
Cost [$] ($la_{i,j}$): $\mathcal{U}([100, 1200])$		
VNF Requirements [24,25]		
Cost [$] ($\alpha_{n,f}$): $\mathcal{U}([50, 1000])$		
Network Function	**CPU Required** [#] ($\eta^k_{n,f}$)	**Processing Capacity** [$Mbps$] (θ_f)
Firewall	4	900
Proxy	4	900
NAT	2	900
IDS	8	600

Experiments design

From the options available, we select from the SNDLib database two topologies, namely *Abilene*, consisting of $N = 12$ nodes and 15 bidirectional links, and *Germany50*, consisting of $N = 50$ nodes and 88 bidirectional links. For each topology, we conduct several experiments investigating the interplay between three critical aspects related to VNF placement: i) the variation in the service demand (i.e., the number $|K|$ of service requests in the network), ii) the variation in network nodes computational resources (sc_n) and links physical capacity ($lc_{i,j}$), and iii) the variation in the service requests reliability threshold τ_k (i.e., the tightness of constraints (13)). For the *Abilene* topology, we set $|K| \in \{2, 4, 6, 8\}$, whereas for *Germany50* we let $|K|$ take values in $\{8, 12, 16, 20, 24, 28\}$. In both scenarios, τ_k is set to the same value for all requests ranging from 0 to 1 in increments of 0.1. We consider three types of network nodes, each endowed with specific computational and storage capabilities [1], namely: off-the-shelf servers, SmartNICs and PISA switches. Also, we distinguish four VNF types, characterized by different processing capacity (θ_f) and resource requirements ($\eta^k_{n,f}$) [24], that are: Firewall, Proxy, Network Address Translation (NAT) and Intrusion Detection System (IDS). Regardless of the selected type, the cost $\alpha_{n,f}$ to install one VNF instance is randomly drawn in the interval $[50, 1000]$ US dollars ($), as done in [22,25]. Each request k that must be routed in the network is randomly assigned a subset of the mentioned VNFs, whose instances are then installed according to each request's traffic demand, b^k. Table 2 summarizes the values of i) computation capability (sc_n) and bandwidth (sb_n) available at each node type, ii) the costs to activate links ($la_{i,j}$) and nodes (sa_n), and to deploy VNFs ($\alpha_{n,f}$), and iii) the processing capacity (θ_f) and required computational resources ($\eta^k_{n,f}$) for each VNF instance. We underline that the node capacities reported in Table 2, and in particular the CPU and processing capabilities assigned to SmartNICs and PISA switches, are chosen to reflect one representative class of programmable accelerators and security-oriented devices, rather than to model any specific commercial product. They should therefore be interpreted as scenario parameters used to explore the impact of heterogeneous compute and forwarding resources on VNF placement and routing decisions. Alternative capacity profiles could be adopted without changing the formulation or solution methodology, and would simply lead to different numerical trade-offs in the evaluation. To study the sensitivity of the model's performance to sc_n and $lc_{i,j}$, we define two additional experimental scenarios, namely ***S2*** and ***S4***, where we increase the values of both parameters by 2 and 4 times with respect to the values reported in Table 2 (which we refer to as scenario ***S1***), respectively.

Table 3
Experimental setup of parameters $lc_{i,j}$ and sc_n, regardless of the selected topology. In Scenario ***S1***, both parameters are setup as defined in Table 2.

	Capacity Parameters	
Scenarios	**Links** $[Gbps]$ $(lc_{i,j})$	**Nodes** $[\#\ CPU]$ (sc_n)
S1	as in Table 2	as in Table 2
S2	2 x ***S1***	2 x ***S1***
S4	4 x ***S1***	4 x ***S1***

Table 4
VNF-to-node allocation constraints (top) and conflicting VNFs (bottom).

VNF \ Node	PISA	SmartNIC	Server
Firewall	✓	–	✓
IDS	✓	✓	–
NAT	–	✓	✓
Proxy	–	✓	✓

VNF \ VNF	Firewall	IDS	NAT	Proxy
Firewall	–	–	✗	–
IDS	–	–	✗	✗
NAT	✗	✗	–	–
Proxy	–	✗	–	–

For the sake of clarity, we summarize in Table 3 how both ***S2*** and ***S4*** setups differ from that of scenario ***S1***.

In line with realistic deployments, we assume that each node type is allowed to instantiate a specific subset of VNF options, reflecting their heterogeneous computational and functional capabilities. Specifically, we do not allow off-the-shelf servers to execute Intrusion Detection System (IDS) instances, and limit them to less resource-intensive NAT, Proxy, and Firewall operations, similarly to their role in Content Delivery Networks or private service provider infrastructures. Instead, IDS functions are offloaded to SmartNICs and PISA devices. Among these, PISA nodes are treated as security-oriented platforms, supporting also Firewall instances but not NAT or Proxy. SmartNICs, by contrast, provide greater flexibility and are permitted to run NAT and Proxy services in addition to IDS. To further reflect practical deployment constraints, we impose incompatibility rules that prevent certain VNFs from coexisting on the same node. In particular, if a node executes NAT, it cannot simultaneously run IDS or Firewall, although it may still host a Proxy instance. Similarly, a node acting as a Proxy cannot also perform IDS, but it may host NAT or Firewall functions. Conceptually, this results in a separation of roles: nodes executing NAT or Proxy behave as part of a forwarding domain, while nodes running IDS or Firewall represent a distinct security domain. Table 4 reports both types of deployment constraints, i.e., the mapping between VNF types and node types capable of executing them (top), and the incompatibility rules that prevent conflicting VNFs from being installed on the same node (bottom).

Reliability model and performance metrics

The reliability of each server $n \in N$ is modeled as a non-increasing, concave, and continuous function of its computational load L_n. Specifically, we define:

$$R_n = 1 - \left(\frac{L_n}{sc_n}\right)^2, \quad \forall n \in N, \tag{51}$$

with $L_n \leq sc_n$ to ensure that $R_n \geq 0$. The quadratic form above is chosen to model a non-increasing, concave degradation of reliability with respect to server load, with an accelerated decline as L_n approaches sc_n. Empirical studies and analytical performance models show that failure rates and error probabilities tend to increase slowly at moderate utilizations and then grow more sharply as the load approaches the hardware limits, due to queueing, contention and thermal effects [8,26,27]. A concave, monotonically decreasing function of the normalized load (L_n/sc_n) captures this behavior while remaining simple enough to be embedded inside our optimization framework and linearization scheme. We emphasize that the proposed approach is not restricted to the specific functional form in Eq. (51): any alternative concave, non-increasing function $H(L_n/sc_n)$ could be used in Eq. (12), and the linearization and cutting-plane heuristic would still apply with minor modifications to the discretization of the reliability domain.

Thus, to assess the efficiency of our model in allocating VNF instances, we analyze the utilization of server computational resources. For each server $n \in N$, we define the resource utilization ξ_n as:

$$\xi_n = \frac{L_n}{sc_n}, \tag{52}$$

which represents the fraction of the computational capacity consumed at server n. To characterize the distribution of ξ_n across the network, we compute the Coefficient of Variation (CV) and the maximum utilization value $\xi^{\max}$ as follows:

$$\mathrm{CV} = \frac{\sigma_{\xi_n}}{(\sum_n \xi_n)/|N|}, \tag{53}$$

$$\xi^{max} = \max_{n \in N} (\xi_n), \tag{54}$$

where σ_{ξ_n} is the standard deviation of ξ_n. In the next section, we show the capability of model ***E*** to preserve end-to-end service reliability as opposed to baseline approaches, and discuss how such model improves overall performance by (i) tending to make ξ_n more uniform and reducing the CV, which indicates a more balanced resource allocation, and (ii) lowering $\xi^{\max}$, which mitigates potential congestion bottlenecks.

6.1. Impact on node resources utilization

Fig. 2 illustrates a representative scenario, showing the routing solution obtained when the following $|K| = 4$ requests are issued with no reliability requirements (i.e., $\tau_k = 0$ for each request) in the *Abilene* network topology:

R1: Green, dot-dash - from *IPLSng* to *STTLng*;
R2: Orange, straight - from *NYCMng* to *STTLng*;
R3: Blue, dash - from *HSTNng* to *STTLng*;
R4: Purple, dot - from *LOSang* to *KSCYng*.

The figure demonstrates how stricter reliability requirements affect routing. Specifically, service request R3 (shown as blue, dashed line when routed with no reliability requirements) is re-routed by model ***E*** as τ_k increases to 0.4. The new routing path, depicted in Fig. 2 as a red, straight line, now includes intermediate nodes *LOSAng* and *SNVAng*, such to decrease computational load of nodes *KSCYng* and *DNVRng* (previously included in the original routing path selected for R3) in order to satisfy the higher quality-of-service requirement.

In the following sections, we evaluate the performance of the proposed optimization model considering different experimental scenarios, designed to focus on the following aspects: i) how performance compares when the model is applied to allocate VNF instances with homogeneous versus diverse functional requirements (Section 6.1.1); ii) how network workload distribution is impacted by the tightness of constraints (13) (Section 6.1.2); and iii) how performance scales to larger network instances (Section 6.3.1). Finally, Section 6.3 provides results and insights regarding model's cost components and running time.

6.1.1. Homogeneous vs differentiated VNFs options

First, we consider the *Abilene* topology and compare the results obtained from models ***E*** and ***H*** when applied to scenario ***S1***, as reported in the first and second column of Table 5, respectively. As one can see, model ***E*** is mostly infeasible with this setup, due to the inherent greater complexity of the problem with respect to model ***H***. For example, when

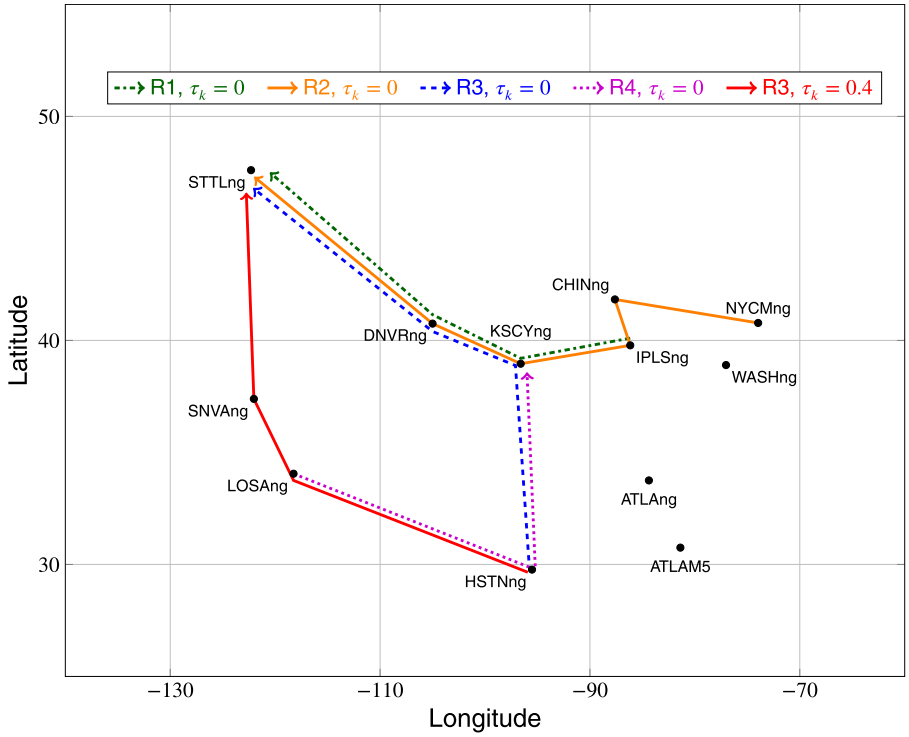


Fig. 2. Example network scenario with service requests routing paths for $|K| = 4$. The red, straight line represents how request R3 is re-routed after increasing τ_k from 0 (blue, dashed line) to 0.4. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$|K| = 2$, ***H*** provides feasible solutions up to $\tau_k = 0.8$, while no solution is available when model ***E*** is used for $\tau_k > 0.3$. In contrast, when $|K| = 8$, model ***E*** is not feasible for any value of parameter τ_k, while model ***H*** is infeasible for $\tau_k \geq 0.2$.

Therefore, to better highlight the impact of model ***E*** on workload distribution, we evaluate its performance in scenarios ***S2*** and ***S4***. Fig. 3 shows the results obtained with our proposed optimization model, when both link and node CPU capacities are scaled by factors of 2 (Fig. 3a and b) and 4 (Fig. 3c and d). The corresponding numerical values are reported in columns ***E-S2*** and ***E-S4*** of Table 5, respectively.

Fig. 3a and c plot the CV (solid lines) as a function of τ_k, while Fig. 3b and d represent how $\xi^{\max}$ (dashed lines) varies with τ_k. The line tones indicate the number of service requests in the network, with darker lines representing lower service loads. Considering CV, lower values indicate a more uniform workload distribution, while higher CV values suggest disparities in server utilization. Regardless of the number of requests, and for both network expansion instances, the CV consistently decreases as τ_k increases, demonstrating the model's ability to distribute the load more evenly when higher reliability is required, even if VNFs are subject to differentiated requirements and deployment constraints. In particular, compared to the baseline case of $\tau_k = 0$ (i.e., when no requirement is set on reliability), we observe that when model ***E*** is applied to ***S2*** (top row) the CV improves between 37% ($|K| = 2$, $\tau_k = 0.7$) and 58% ($|K| = 6$, $\tau_k = 0.2$) at the highest feasible reliability threshold, while the improvement lays between 49% ($|K| = 4$, $\tau_k = 0.8$) and 63% ($|K| = 8$, $\tau_k = 0.6$) when scenario ***S4*** (bottom row) is considered.

Reducing $\xi^{\max}$ is equally critical, as it enables more efficient resource management and mitigates the risk of network congestion. Similar to CV, $\xi^{\max}$ exhibits a clear downward trend as τ_k increases. When the network is setup as in scenario ***S2***, results demonstrate that the proposed optimization framework can substantially reduce maximum server utilization from 25% in the most demanding case ($|K| = 8$, $\tau_k = 0.1$) to 56% in the lightest load case ($|K| = 2$, $\tau_k = 0.8$). A similar trend is observed for scenario ***S4***, where $\xi^{\max}$ is reduced up to 56% for $|K| \geq 4$ at the highest feasible value of τ_k .

In general, we observe a tradeoff emerging from the interplay between network capacity, workload distribution and problem's feasibility, that is strongly influenced by the tightness of the reliability constraints. In lower-capacity settings (e.g., scenarios ***S1*** and ***S2***), high workloads rapidly lead to infeasibility under stringent reliability requirements (e.g., in ***S2***, when $|K| = 8$ the problem is infeasible for $\tau_k \geq 0.3$), indicating that systems with constrained resources are more vulnerable to performance degradation. However, even in such restrictive contexts, meaningful improvements in workload distribution can still be achieved for relatively small reliability thresholds (e.g., τ_k equal to 0.1 or 0.2), suggesting that partial relaxation of the constraints may provide a practical compromise between performance and feasibility. By contrast, in higher-capacity settings (e.g., scenario ***S4***), the benefits of an improved workload distribution are more pronounced at larger reliability thresholds (e.g., $\tau_k \geq 0.3$), which implies that additional capacity enables the system to sustain stricter requirements without compromising performance. These findings suggest that the appropriate choice of the reliability threshold should be informed by available capacity: tighter thresholds can be used in resource-rich environments, whereas modest threshold relaxations are suggested in resource-constrained ones, as a strategy to improve workload balance while preserving feasibility.

6.1.2. *Tightness of the reliability constraint*

A distinctive characteristic of our proposed model ***E*** regards the implementation of reliability guarantees with an end-to-end perspective, i.e., along the full service path rather than node-wise in a local fashion. To assess the impact that the tightness of reliability constraints has on VNFs placement, we compare the distributions of network nodes utilization ξ_n obtained by baseline approaches ***U*** and ***L*** against that corresponding to model ***E***, considering the *Abilene* topology and scenario ***S4***. Yet, we highlight that similar considerations apply also to scenario ***S2***.

Regardless of the number of requests $|K|$ that are issued in the network, we generally observe that model ***E*** provides a better load balance than the considered baselines, moving the distribution of ξ_n towards lower utilization values as much as it is permitted by capacity

Table 5
Servers' utilization statistics from models ***H*** (as proposed in [22], where VNFs are subject to homogeneous processing and computational requirements) and ***E*** (proposed in this work), when applied to scenario ***S1***. Columns ***E-S2*** and ***E-S4*** show how results from model ***E*** scale when links and nodes capacities are set to twice (***S2***) and four (***S4***) times the values reported in Table 2.

$\|K\|$	τ_k	*H-S1*		*E-S1*		*E-S2*		*E-S4*	
		CV	ξ^{max} [%]	CV	ξ^{max} [%]	CV	ξ^{max} [%]	CV	ξ^{max} [%]
2	0	1.56	0.67	1.16	0.83	1.46	0.75	1.46	0.38
	0.1	1.56	0.67	1.07	0.67	1.46	0.75	1.46	0.38
	0.2	1.56	0.67	0.78	0.67	1.46	0.75	1.46	0.38
	0.3	1.56	0.67	0.76	0.50	1.46	0.75	1.46	0.38
	0.4	1.46	0.63	/	/	1.32	0.67	1.46	0.38
	0.5	1.35	0.63	/	/	1.16	0.42	1.46	0.38
	0.6	1.24	0.50	/	/	1.16	0.42	1.46	0.38
	0.7	1.04	0.33	/	/	0.92	0.33	1.46	0.38
	0.8	0.96	0.25	/	/	/	/	1.46	0.38
4	0	1.03	1.00	0.60	1.00	0.97	1.00	1.34	0.88
	0.1	0.95	0.83	/	/	0.84	0.75	1.34	0.88
	0.2	0.87	0.75	/	/	0.81	0.75	1.12	0.88
	0.3	0.76	0.67	/	/	0.75	0.75	1.27	0.88
	0.4	0.62	0.58	/	/	0.66	0.63	1.11	0.75
	0.5	/	/	/	/	/	/	1.00	0.63
	0.6	/	/	/	/	/	/	0.91	0.50
	0.7	/	/	/	/	/	/	0.84	0.38
	0.8	/	/	/	/	/	/	0.69	0.38
6	0	0.84	1.00	0.31	1.00	0.84	1.00	1.29	1.00
	0.1	0.65	0.75	/	/	0.58	0.75	1.24	0.88
	0.2	0.43	0.67	/	/	0.35	0.63	1.18	0.75
	0.3	/	/	/	/	/	/	0.91	0.75
	0.4	/	/	/	/	/	/	0.84	0.63
	0.5	/	/	/	/	/	/	0.76	0.50
	0.6	/	/	/	/	/	/	0.61	0.44
	0.7	/	/	/	/	/	/	/	/
	0.8	/	/	/	/	/	/	/	/
8	0	0.76	1.00	/	/	0.79	1.00	1.22	1.00
	0.1	0.48	0.75	/	/	0.44	0.75	1.15	0.88
	0.2	/	/	/	/	/	/	1.10	0.75
	0.3	/	/	/	/	/	/	0.93	0.63
	0.4	/	/	/	/	/	/	0.79	0.56
	0.5	/	/	/	/	/	/	0.61	0.50
	0.6	/	/	/	/	/	/	0.45	0.44
	0.7	/	/	/	/	/	/	/	/
	0.8	/	/	/	/	/	/	/	/

constraints. To illustrate this result, we consider the case with $|K| = 4$, which is particularly insightful since it is the only configuration for which the problem remains feasible across all tested threshold values of τ_k. We plot in Fig. 4 the distribution of ξ_n obtained in such scenario according to baselines ***U*** (blue bars) and ***L*** (orange), and our model ***E*** (green), for τ_k equal to 0.1 (top-left), 0.2 (top-right), 0.4 (bottom-left), and 0.8 (bottom-right). For a fixed value of τ_k, the x-axis represents the computational load of a node, while the y-axis corresponds to the (normalized) frequency of occurrence of a given load value into bins of size 10%. Note that the value of τ_k has no impact on the results provided by model ***U***, where constraints (13) are omitted from the model. As shown in Fig. 4, when model ***E*** is selected and the highest feasible reliability is required (i.e., $\tau_k = 0.8$), all the active nodes (i.e., 9) have a load ratio lower than 40%, while this happens only in 66% (4 out of 6) and 86% (6 out of 7) of the nodes selected by baselines ***U*** and ***L***, respectively. Conversely, when reliability is not taken into account (baseline ***U***), 33% of the active nodes consume more than 60% of their computational resources, with one of them being loaded as much as $\xi^{max} = 88\%$. In the same scenario, ξ^{max} is reduced by 56% when model ***E*** is adopted, which also yields 4% absolute improvement over baseline ***L***. Finally, we show in Fig. 5 the distributions of reliabilities obtained in the same scenario from the three optimization options (similar results are obtained for other scenarios). Coherently with the observed improvement in computational load fairness among network nodes, our proposed model always improves over both baselines the median reliability across all the requests when $\tau_k \geq 0.2$, on average by 60% and 21% with respect to models ***U*** and ***L***, respectively.

To conclude, we observe that reliable models adopt different allocation strategies to satisfy the reliability constraints. On the one hand, when service reliability is locally guaranteed (as done by baseline ***L***), the model tends to strategically re-allocate VNFs among the same nodes selected by the unreliable strategy (model ***U***), i.e., without altering the path, as long as this suffices to meet the per-node reliability bound. On the other hand, due to the stringent reliability constraints enforced by model ***E***, achieving end-to-end guarantees may require activating additional nodes, thus rerouting the corresponding service requests along alternative paths. For instance, as shown in Fig. 4 when $\tau_k = 0.8$, while model ***L*** activates one more node than model ***U***, model ***E*** requires three additional activations (i.e., two more than ***L***) to satisfy the reliability requirements.

6.1.3. Scalability to larger network instance

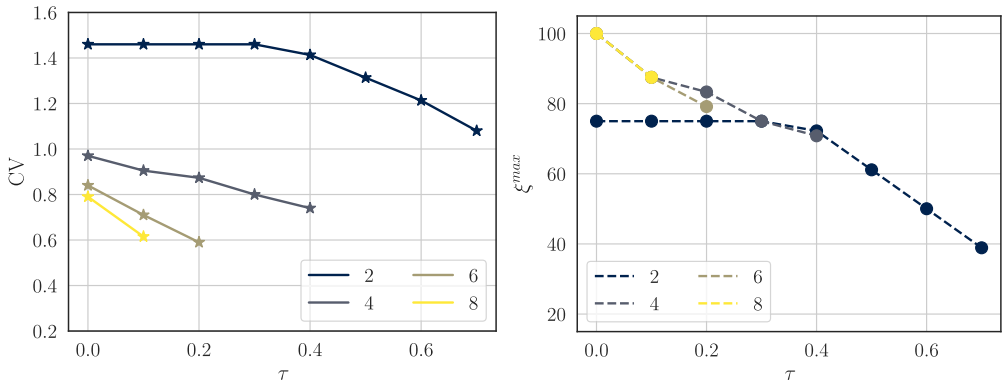
In our final experiment, we examine the impact of using model ***E*** on node utilization ξ_n when the number of service requests $|K|$ exceeds 8. To do so, we select the *Germany50* topology and apply the model to scenario ***S2*** (i.e., when the values of links and nodes capacity parameters are twice of those reported in Table 2). Moreover, to avoid capacity-constraint issues during the allocation process, we randomly select $|K|$ source-destination pairs from the options available in SNDLib, considering only those that offer a traffic demand b^k equal to 2 Gbps.

We plot in Fig. 6 the CV (solid lines) and ξ^{max} (dashed lines) obtained from using model ***E*** to allocate VNFs in scenario ***S2***, with network load varying from $|K| = 8$ to $|K| = 28$ with a step-size increase of 2 requests, versus τ_k, respectively. As shown, both CV and ξ^{max} show a decreasing trend with respect to τ_k, similarly to what observed in previous scenarios. In particular, in the most loaded scenario (i.e., $|K| = 28$) and at the corresponding maximum feasible reliability threshold (i.e., $\tau_k = 0.2$), our solution contributes to decrease CV by 21% and reduce ξ^{max} by more than 30%, at the cost of only 2 more node activations with respect to baseline ***U***. Fig. 7 shows the benefits on servers load distribution induced by model ***E*** (green bars) with respect to baselines ***U*** (blue) and ***L*** (orange), for different reliability thresholds and for $|K| = 24$ (similar results could be observed for different values of $|K|$). We observe that when the model yields feasible solutions (i.e., for $\tau_k \leq 0.6$ in this case), it activates network nodes such to ensure a more fair distribution of the computational load, on average decreasing ξ^{max} by 28% and 15% with respect to unreliable (i.e., model ***U***) and locally reliable (i.e., model ***L***) allocations, respectively.

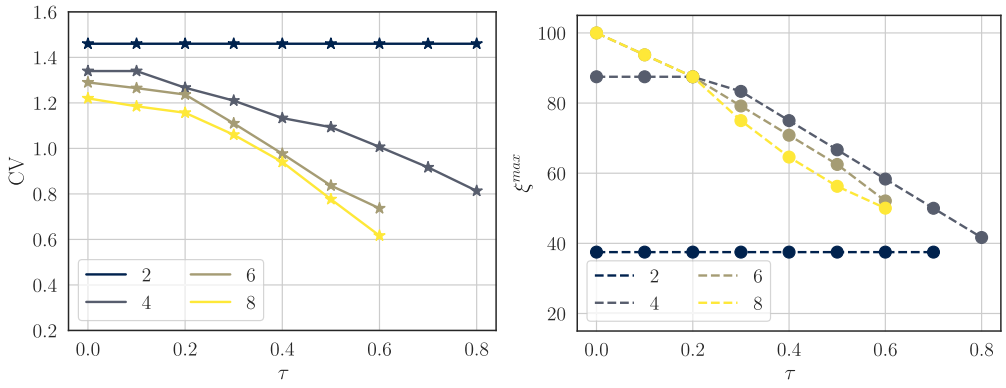
6.2. LB-greedy vs optimal approach

In this Section, we evaluate the performance of LB-Greedy, and compare them with the (optimal) ones obtained by model ***E***. We consider 100 independent instances per value of $|K|$ driven by *Germany50* topology (scenario ***S2***), and we let $|K|$ vary between 12 and 28. For a fixed value of $|K|$, the algorithm processes requests sequentially: for each request, it iteratively explores the top 100 shortest paths in increasing order of length until the request is either satisfied or marked as blocked, then proceeds to the next request. We report in Table 6 the results obtained for different values of hyper-parameters β and γ, which control the load terms in path and placement scores. More precisely, we take $\beta = \omega \cdot l_{max}$ and $\gamma = \omega \cdot \alpha_{max}$, where l_{max} and α_{max} denote the average link and VNF deployment costs, respectively, while ω is a scaling parameter which we set in our experiments as equal to either 1, 100, or 1000. Note that LB-Greedy took less than 5 s to solve each of the considered network instances, being therefore substantially faster than the proposed, optimal approach.

As shown in Table 6, higher values of ω cause the algorithm to prioritize load balancing, generally increasing the total deployment cost. When $|K| \leq 20$, raising ω from 1 to 100 improves on average the CV



(a) Model: ***E***, Scenario: ***S2***, Metric: CV. (b) Model: ***E***, Scenario: ***S2***, Metric: ξ^{max}.



(c) Model: ***E***, Scenario: ***S4***, Metric: CV. (d) Model: ***E***, Scenario: ***S4***, Metric: ξ^{max}.

Fig. 3. Results on CV (left column) and $\xi^{\max}$ (right column) versus reliability threshold obtained in *Abilene* when model ***E*** is applied to scenarios ***S2*** (top row) and ***S4*** (bottom row), respectively.

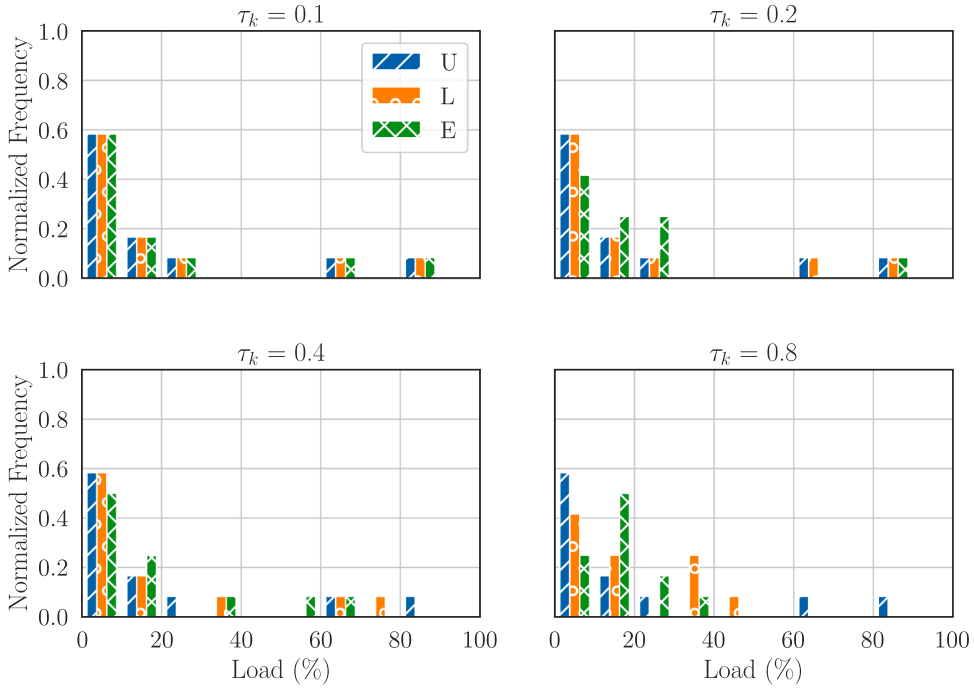


Fig. 4. Distribution of ξ_n obtained by models ***U*** (blue), ***L*** (orange) and ***E*** (green) in *Abilene*, when applied to scenario ***S4*** with $|K| = 4$ service requests, respectively. We set τ_k as equal to 0.1 (top-left), 0.2 (top-right), 0.4 (bottom-left), and 0.8 (bottom-right). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

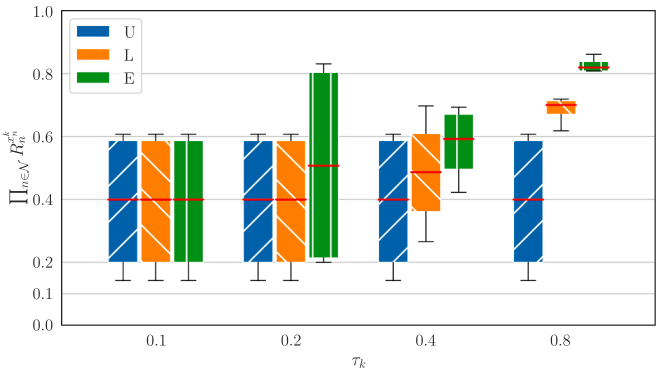


Fig. 5. Distribution of per-request reliability obtained by models ***U*** (blue), ***L*** (orange) and ***E*** (green) in *Abilene*, when applied to scenario ***S4***, with $|K| = 4$ service requests and τ_k taking values in $[0.1, 0.2, 0.4, 0.8]$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

by 10% and requests' reliability by 22% (20th p.) and 4% (50th p.), but requires 18% higher average total deployment cost. However, the marginal gains diminish with each increment of ω: in the same scenarios, when ω raises from 100 to 1000, 1% higher average costs lead on average to 3% lower CV and 9%-3% higher 20th and 50th percentile of reliability, respectively. Similar improvements are obtained for $|K| = 24$ when increasing ω from 1 to 100, when it contributes to raise τ_k (20th percentile) from 0 (i.e., no guarantee) to 0.15, while no impact is observed for $|K| = 28$. Conversely, setting ω to 1000 (or higher values) has a negligible impact when for $|K| > 20$: results suggest $\omega = 100$ as a reasonable setup for the scenario we selected (Germany50-S2), considering also that moderate variations around these values do not qualitatively change the behavior of the heuristic. Nevertheless, despite the effectiveness of such setting (none of the requests was blocked when $|K| \leq 24$, while one request was blocked in only one instance when $|K| = 28$) and the overall computational efficiency of the algorithm, LB-Greedy is anyway able to provide only average reliability guarantees at consistently higher (+17%) average deployment costs than our optimal approach, which leads on average to 13% better CV while providing minimum reliability guarantees to all service demands.

6.3. Cost components and running time

Table 7 summarizes cost components (server and link activation costs, CAPEX and total cost) for the proposed optimization model, and the corresponding computation times, when applied to place VNFs in *Abilene* (top, scenario ***S4***) and *Germany50* (bottom, scenario ***S2***) topologies. In both scenarios, and independently of $|K|$, raising τ_k with fixed server and link activation costs results in non-decreasing CAPEX, due to the potential deployment of additional VNF instances to provide the requested reliability guarantees. Moreover, we observe that when constraints (13) become more stringent, the algorithm prioritizes allocating additional resources to reinforce existing service chains rather than distributing requests across alternative paths, unless new nodes activations are required to meet the reliability requirements. As a result, server and link costs often remain unchanged while CAPEX grows due to the deployment of extra virtual functions. However, if stricter reliability constraints necessitate utilizing the capacity of previously inactive nodes, server costs may also increase (as observed for request R3 in Fig. 2), an effect likely occurring the closer we are to the highest feasible value of τ_k. In some of such scenarios, the algorithm achieves a marginal reduction in link expenditures by leveraging more efficient VNF placements, yet the overall cost consistently increases for stricter values of τ_k. Also, in both topologies we observe that when computational resources of new servers are used (i.e., when servers activation cost increases), CAPEX may decreases in some cases, regardless of the service demand $|K|$. In fact, the cost $\alpha_{n,f}$ to activate a VNF instance f depends on the type of node n hosting that instance: when a new server is selected, the migration of one or more VNF instances from a previously selected node to the new one may result in a VNF installation cost gain. For instance, in *Abilene* this happens when $K = 4$ and τ_k increases from 0.7 to 0.8, or equivalently when $K = 8$ and τ_k increases from 0.5 to 0.6. Similarly, we observe this behavior in *Germany50* for $K = 16$, when τ_k increases from 0.5 to 0.6. All these outcomes are consistent with the model's approach to balancing the contributions of the three cost components, while adhering to reliability requirements.

Regarding computation times, the experiments were conducted on a commodity server equipped with an Intel i5-5200U and 16 GB of memory. Overall, the combination of the MILP reformulation and the cutting-plane heuristic exhibits favorable scalability properties on the considered topologies. For the *Abilene* network (12 nodes) and for the larger *Germany50* network (50 nodes), all feasible instances are solved on a commodity server in less than 4 and 7 min, respectively, regardless of the number of service requests and the tightness of the reliability constraint, with faster solving observed on *Abilene* instances compared to *Germany50*. As expected, the solution time tends to increase with both $|K|$ and τ_k, but the growth remains moderate thanks to the limited size of the discretized reliability domain (Section 3.2) and the effectiveness of the bounding and warm-start strategies in pruning the search space.

6.3.1. Running time scalability analysis

To illustrate further how the proposed approach scale with increasing network sizes and service demand volumes, we analyze in details the relationship between computation time, number of network nodes and $|K|$. Focusing on Germany50 as base topology, we generate

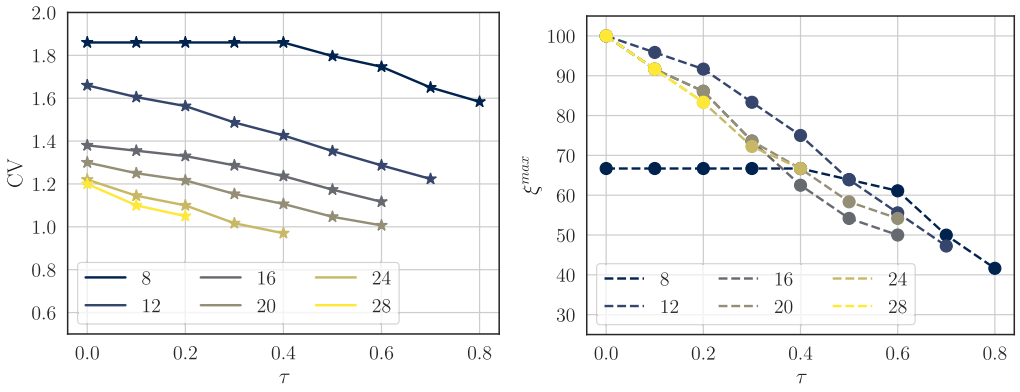


Fig. 6. Results on CV (left) and $\xi^{\max}$ (right) versus reliability threshold obtained when model ***E*** is applied in *Germany50* topology to scenario ***S2***.

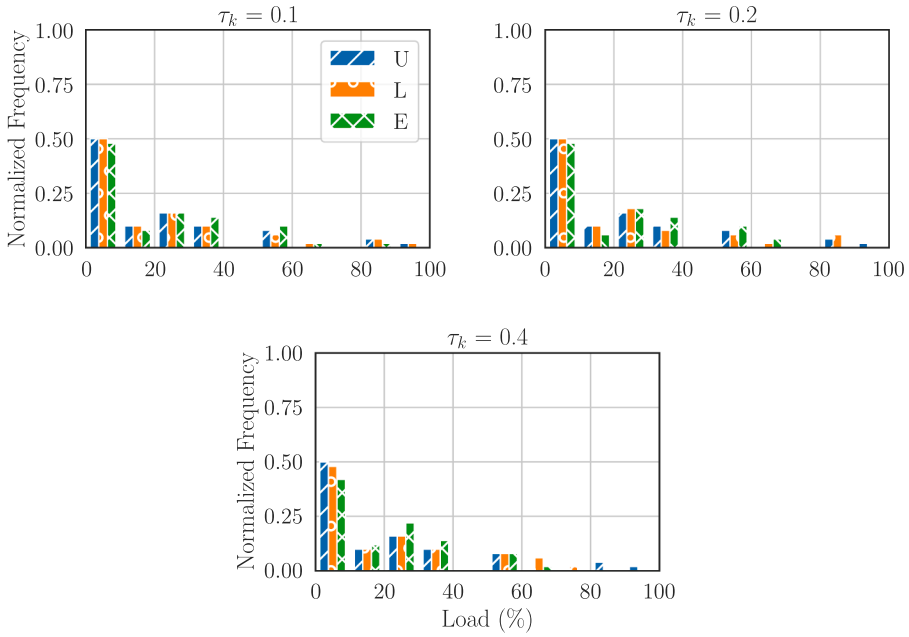


Fig. 7. Distribution of ξ_n obtained by models ***U*** (blue), ***L*** (orange) and ***E*** (green) when applied in *Germany50* to scenario ***S2***, for $|K| = 24$. We set τ_k to 0.1 (top-left), 0.2 (top-right), 0.4 (bottom), while the case of tau = 0.8 is omitted due to infeasibility. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 6
Topology: *Germany50*, Scenario: ***S2***. Columns 3, 4 and 5 report the performance of LB-Greedy when β and γ are scaled by factors 1, 100 and 1000 with respect to maximum links (1200 [$]) and nodes (5000 [$]) activation costs, and |**K**| is varied between 12 and 28. Total Cost (80th p.) and CV (80th p.) correspond to the 80th percentile values of the distributions computed over 100 independent runs, while τ_k (20th p.) and τ_k (50th p.) represents the minimum reliability achieved by 80% and 50% of the requests, respectively. The last three columns report optimal results obtained by model ***E***, at the highest feasible reliability threshold τ_k (opt).

Scale (ω)	|**K**|	Total Cost (80th p.)	CV (80th p.)	τ_k (20th p.)	τ_k (50th p.)	Total Cost (opt.)	CV (opt.)	τ_k (opt.)
	12	302.0	1.67	0.40	0.73	317.2	1.18	0.70
	16	403.5	1.44	0.27	0.69	406.5	1.08	0.60
1	20	510.8	1.25	0.13	0.59	499.0	0.99	0.60
	24	619.5	1.15	0.00	0.49	603.4	0.93	0.40
	28	732.4	1.02	0.00	0.44	695.4	0.95	0.20
	12	365.0	1.50	0.44	0.75	317.2	1.18	0.70
	16	488.4	1.29	0.31	0.70	406.5	1.08	0.60
100	20	605.0	1.13	0.23	0.64	499.0	0.99	0.60
	24	730.9	1.05	0.15	0.54	603.4	0.93	0.40
	28	854.7	0.97	0.00	0.44	695.4	0.95	0.20
	12	367.7	1.42	0.46	0.77	317.2	1.18	0.70
	16	490.9	1.26	0.38	0.72	406.5	1.08	0.60
1000	20	608.6	1.13	0.24	0.65	499.0	0.99	0.60
	24	732.8	1.05	0.17	0.55	603.4	0.93	0.40
	28	859.2	0.97	0.00	0.44	695.4	0.95	0.20

different synthetic topologies of varying sizes by randomly sampling $\psi = [10\%, 20\%, \ldots 100\%]$ of nodes from set N (and corresponding incident links from L). Being $|V|$ the number of sampled nodes, for each synthetic topology we generate variable service demands by linearly scaling $|K|$ with $|V|$ by the network service load density ϕ, defined as the number of requests per single network node. We plot in Fig. 8 the solving times resulting from running model E as a function of $|V|$, where ϕ is set to $[0.1, 0.2, \ldots, 0.5]$. Results show that both increasing the load density for fixed topology and considering larger topologies lead to longer solving time. For any fixed value of ψ, increasing ϕ steadily raises running times, due to the raise in problem complexity induced by denser service demands. Such growth remains modest at $\psi \leq 30\%$ but steepens at higher ψ, indicating that high service load on larger effective topologies proves especially computationally demanding. Regardless of ϕ, higher ψ (i.e., larger topologies) incurs substantially longer running times–often by orders of magnitude relative to $\psi \leq 30\%$, particularly under higher service demand densities. Furthermore, we observe that when $\psi \geq 30\%$ the running time shows an almost exponential trend with increasing ϕ. Overall, results suggest that the proposed approach is suitable for offline or near-real-time NFV orchestration in medium-scale infrastructures, and can serve as a high-quality benchmark or building block for hybrid schemes that combine exact optimization with faster heuristic or learning-based methods.

7. Related work

The optimal placement of Virtual Network Functions (VNFs) has been extensively studied in the context of network service provisioning, to optimize network usage and resource allocation ([26–29]), or economic aspects such as provider's profits ([30]), capital and operational

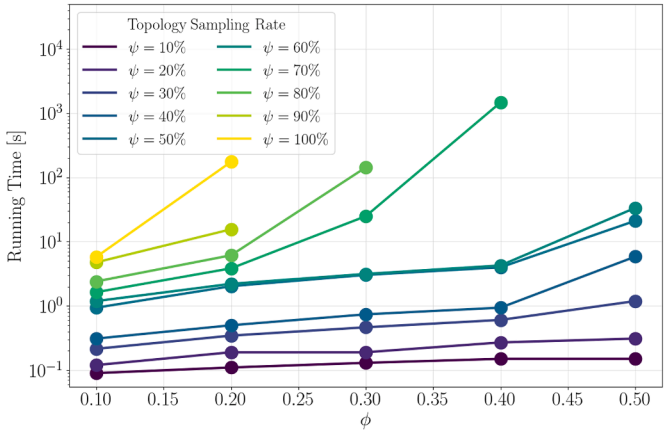


Fig. 8. Running time versus service load density, measured as the number ϕ of requests-per-node, for 10 synthetic topologies of varying sizes. Higher input values of ψ map to larger topologies.

costs ([12,25,31]). Concurrently, optimal placement Service Function Chaining (SFC) services is also a popular research interest [13,31].

In [28], authors address the VNF placement and routing problem in NFV systems, proposing a mathematical programming model that accounts for practical constraints such as compression/decompression and forwarding latency. A tailored heuristic is introduced to handle large-scale instances, illustrating trade-offs between traffic engineering and infrastructure efficiency. Conversely, authors in [29] propose an optimal VNF placement strategy that maximizes the number of successfully executable services under QoS requirements. Results show that the proposed Branch and Bound algorithm based on Artificial Intelligence searches outperforms traditional offline resource allocation in terms of scalability to large online scenarios, performing within 5% gap with respect to the best obtained ILP solution. Differently than previous works, our model emphasizes reliability-aware placement to achieve load balancing throughout the network, and routing under node-specific constraints, with explicitly modeling of load-induced reliability degradation and minimizing infrastructure costs.

Relatively to end-to-end service provisioning, Service Function Chaining (SFC) in hybrid environments is examined in [13], which leverages SDN-NFV integration to improve chain mapping and resource utilization. While this complements our work, we focus instead on the intrinsic robustness of placement strategies rather than dynamic flow steering mechanisms. Reliable placement of VNF chains is addressed in [31], where static failure probabilities of single VNFs rather than workload-dependent reliability of traversed network nodes are multiplied to model the reliability of the whole service chain. While our work does not explicitly consider SFC, we still address end-to-end reliability and consider nodes functional incompatibilities to resort to realistic service deployment scenarios.

In particular, reliability-aware modeling has been addressed in different contexts. For instance, the work in [26] proposes an optimization model for backup resource allocation where failure probability depends on workload, a concept we extend by incorporating end-to-end service reliability guarantees into the primary path optimization. Similarly, the routing-based reliability model in [16] focuses on finding two disjoint paths to improve robustness, while a solution based on shared backup paths is proposed in [30] to implicitly enforce reliability guarantees against single VNF failures. Differently than both approaches, our model embeds reliability directly into the placement and routing decisions over a single (optimized) service path. We model network node reliability as a non-increasing function of its workload, as often done in prior studies, which confirm that workload-dependent failure probability is a non-decreasing function [26,27] and empirically observe that highly utilized servers experience failure rates up to an order of magnitude higher than lightly loaded servers [8]. Accordingly, our model reflects the fact that higher load reduces overall reliability of network servers.

As for the optimization objective, energy- and cost-aware NFV placement strategies have also been explored in prior works. The study in [12] presents a cost-based placement framework for virtual DPI functions,

Table 7
Impact of τ_k on cost components and model runtime ($\mathcal{T}$) for *Abilene* (a) and *Germany50* (b), when model ***E*** is applied to scenarios ***S4*** and ***S2***, respectively. Costs are scaled by 10^3.

(a) Topology: *Abilene*, Scenario: ***S4***.

|K|	τ_k	Server	Link	CAPEX	Total	$\mathcal{T}$ [s]
2	≤ 0.8	39.50	7.52	2.71	49.73	0.51
4	0.0	67.1	14.0	7.0	88.0	1.4
	0.1	67.1	14.0	7.0	88.0	1.6
	0.2	67.1	14.0	7.0	88.1	3.3
	0.3	67.1	14.0	7.2	88.3	2.4
	0.4	67.1	14.0	7.5	88.5	3.1
	0.5	67.1	14.0	7.5	88.6	10.5
	0.6	67.1	14.0	7.9	89.0	6.9
	0.7	67.1	14.0	8.3	89.4	8.9
	0.8	71.2	13.5	6.7	91.3	29.4
6	0.0	103.8	21.9	8.5	134.2	2.3
	0.1	103.8	21.9	8.7	134.4	3.3
	0.2	103.8	21.9	9.1	134.8	4.4
	0.3	103.8	21.9	9.5	135.2	8.1
	0.4	103.8	21.9	9.9	135.6	8.4
	0.5	103.8	21.9	10.5	136.2	23.7
	0.6	103.8	21.9	11.5	137.2	49.6
8	0.0	130.5	26.4	10.8	167.7	4.5
	0.1	130.5	26.4	11.2	168.1	4.3
	0.2	130.5	26.4	11.6	168.6	6.7
	0.3	130.5	26.4	12.2	169.2	29.8
	0.4	130.5	26.4	13.0	169.9	70.4
	0.5	130.5	26.4	14.1	171.0	206.3
	0.6	134.7	25.9	12.1	172.6	243.0

(b) Topology: *Germany50*, Scenario: ***S2***.

|K|	τ_k	Server	Link	CAPEX	Total	$\mathcal{T}$ [s]
12	0.0	184.7	124.6	5.2	314.2	9.1
	0.1	184.7	124.6	5.3	314.3	69.0
	0.2	184.7	124.6	5.5	314.8	19.7
	0.3	184.7	124.6	5.7	315.0	52.3
	0.4	184.7	124.6	5.9	315.2	86.6
	0.5	184.7	124.6	6.1	315.4	81.4
	0.6	184.7	124.6	6.4	315.7	77.7
	0.7	185.4	125.0	6.8	317.2	88.6
16	0.0	237.4	157.2	9.2	403.4	16.6
	0.1	237.4	157.2	9.4	404.0	66.7
	0.2	237.4	157.2	9.6	404.3	56.9
	0.3	237.4	157.2	9.9	404.5	66.2
	0.4	237.4	157.2	10.4	405.0	163.2
	0.5	237.4	157.2	11.0	405.7	151.1
	0.6	238.1	157.7	10.8	406.5	320.3
20	0.0	289.9	193.6	11.8	495.3	19.6
	0.1	289.9	193.6	12.0	495.5	44.9
	0.2	289.9	193.6	12.3	495.8	69.7
	0.3	289.9	193.6	12.5	496.0	239.3
	0.4	289.9	193.6	13.1	496.6	218.9
	0.5	289.9	193.6	13.8	497.3	403.0
	0.6	290.6	194.0	14.4	499.0	403.4
24	0.0	343.0	228.6	14.1	585.7	24.5
	0.1	343.0	228.6	14.5	586.1	106.6
	0.2	343.0	228.6	14.8	586.4	332.0
	0.3	343.0	228.6	15.3	586.9	376.7
	0.4	350.4	236.1	16.9	603.4	404.2
28	0.0	406.8	272.0	15.5	694.3	30.1
	0.1	406.8	272.0	16.1	694.9	405.4
	0.2	406.8	272.0	16.6	695.4	373.8

emphasizing economic factors without modeling reliability trade-offs. Power-awareness in network design is analyzed in [14], highlighting the importance of energy-efficient configurations. Our work builds upon these principles by demonstrating how reliability-aware placement can indirectly reduce energy consumption by minimizing server overuse and unnecessary redundancy.

From a mathematical optimization perspective, our problem shares structural characteristics with general non-convex MINLPs, as discussed in [7]. To manage such complexity, we draw upon classical techniques for factorable program reformulations [17,18] and exploit MILP-based relaxations following principles discussed in [21]. In particular, we combine linearization with dynamic programming and develop a cutting-plane-based heuristic with a warm-start strategy, yielding scalable solutions for large network instances.

Beyond exact and decomposition-based optimization, a parallel line of research tackles VNF placement and Service Function Chain (SFC) embedding through (i) metaheuristics and hybrid heuristic–metaheuristic schemes, and (ii) learning-based approaches. These methods are typically motivated by scalability and by the need to operate in dynamic environments where re-optimization must be performed frequently, whereas our work focuses on a rigorous reliability-aware formulation (load-dependent node reliability), explicit incompatibility constraints and a solver-driven approach (MILP reformulation plus cutting-plane acceleration) that can provide exact or near-exact solutions in practical time for realistic topologies.

Metaheuristics and hybrid schemes. Genetic algorithms and neighborhood-search metaheuristics (e.g., tabu search) are among the approaches used to explore large solution spaces with reduced computational effort, at the cost of weaker optimality guarantees. In [32] the authors propose a genetic-algorithm (GA) approach for sequential SFC placement in virtualized edge networks. A key strength of GA-based placement is the ability to explore large combinatorial spaces and to incorporate multiple (possibly non-convex) objectives or constraints through flexible encodings, while remaining relatively easy to parallelize and to adapt across heterogeneous infrastructures. However, the solution quality can be sensitive to representation choices and hyper-parameters (population size, crossover/mutation rates, stopping criteria), and feasibility is often enforced via penalties or repair operators, which may become brittle when constraints are tight (e.g., strict end-to-end guarantees or strong placement restrictions). Moreover, reliability is usually handled implicitly (e.g., via load balancing or availability proxies), whereas our model embeds a workload-dependent reliability constraint directly in the optimization problem. A complementary perspective based on tabu search is presented in [33], where the authors formulate the Subchain-Aware NFV service Placement (SAP) optimization model to explicitly account for (i) the configuration cost of stitching reused network functions into an SFC and (ii) the recovery cost of network functions with limited reliability, while encouraging reuse of already deployed consecutive subchains. They develop Tabu-SAP to solve the SAP problem and report substantial capacity/cost improvements (e.g., significantly more supported SFCs compared to reuse baselines) with execution times much lower than exact optimization for longer chains. The main limitations, relative to our focus, are that the reliability aspect is captured via recovery-cost modeling rather than via an explicit *workload-dependent* reliability constraint, and the model does not target the same set of hard constraints we consider (e.g., explicit incompatibility rules and per-node VNF allocation constraints), which are central to our formulation. In addition to the optimal placement of SFC requests, the ability to adapt to online demand variations is a crucial aspect of virtual network infrastructures. In [34] the authors design a multi-criteria heuristic for online operation that jointly selects VNF placements and routes upon the arrival of new SFC demands using a weighted preference function over several features (including server/node activation, available CPU, link bandwidth, link latency, and hop count). A GA is employed offline to automatically tune the model's weights and hyperparameters, resulting in higher acceptance rates and improved server utilization compared to the untuned version of the algorithm, which represents a major strength of this work. However, the approach is not fully tailored to online adaptation: the proposed GA runs once offline to optimize only a relatively small set of heuristic hyperparameters, limiting on-the-fly adaptations of the learning policy in presence of non-stationary traffic or topology changes. Moreover, although load balancing is implicitly targeted by minimizing the total power consumption–modeled as dependent on the server load–no explicit reliability guarantees, per-node VNF allocation constraints, or VNF conflict considerations are included in the optimization problem. In contrast to single-objective optimization, the study in [35] leverages a multiobjective GA to investigate SFC *reconfiguration*, jointly optimizing migration and rerouting decisions to react to demand variations. The main strength of this line is the explicit treatment of multiobjective trade-offs: Pareto sets allow operators to select configurations that balance (re)configuration cost, resource usage and service impact according to operational preferences, being thus naturally aligned with dynamic settings where reconfiguration is frequent. On the other hand, reconfiguration-focused objectives can differ from single-shot provisioning quality: when the baseline configuration is already given, the GA search space and outcomes may be biased by the initial state and by the assumed migration model/costs. In addition, the approach remains heuristic and typically provides no deterministic optimality or feasibility guarantees under stringent constraints; by contrast, our MILP-based framework can serve as an exact/near-exact benchmark to quantify the optimality gap of reconfiguration heuristics under reliability constraints and incompatibility rules.

Learning-based approaches. Reinforcement learning and deep learning-based methods have proven particularly attractive for very large-scale or highly time-varying environments, where solving exact models to optimality at each decision epoch proves impractical. These approaches increasingly frame placement, chaining, and orchestration as sequential decision problems, training policies that deliver near-instantaneous runtime decisions. Building on such paradigm, in [36] the authors employ Deep Reinforcement Learning (DRL) for VNF placement in dynamic SDN/NFV environments. The primary strength of this work is the *online responsiveness*: once trained, inference is fast and can adapt decisions to time-varying traffic without solving a new optimization problem at every step. Nevertheless, DRL policies strongly depend on reward shaping and on the availability of representative training scenarios; constraint satisfaction is often handled through penalties or soft constraints, which may lead to occasional infeasibility or hard-to-predict behavior when operating conditions shift (e.g., new topologies, different demand distributions, or tighter reliability requirements). In [37] the authors also investigate VNF placement optimization, modeling it as a constrained combinatorial task and developing a DRL framework to minimize the overall infrastructure power consumption. In particular, they use a sequence-to-sequence encoder-decoder architecture with attention blocks to map each service chain into a placement decision and then combine the learned policy with a simple heuristic into a hybrid agent. Strengths include scalability to large state spaces (relative to exact solvers) and the possibility to incorporate operational signals (e.g., instantaneous utilization, queueing proxies) directly into the state representation. Conversely, weaknesses include limited interpretability of the resulting policy, non-trivial reproducibility due to training variance, and generalization issues, as policies trained on a given topology or workload regime may not transfer reliably without retraining or fine-tuning. Moreover, neither SFC reliability nor infrastructure cost optimization are addressed by the optimization framework. The DRL-based joint placement and routing of SFCs is also addressed in [38], where the authors achieved a substantial improvement of the model's representational capability by incorporating

an attention mechanism within the optimization framework. In fact, attention can highlight salient nodes/links and improve decision quality when the network state is high-dimensional, which is particularly relevant for coupled placement and routing problems. However, the additional model complexity increases training cost and can exacerbate overfitting to the training topologies; moreover, attention improves *how* the policy represents the problem but does not by itself provide hard guarantees on optimality or constraint satisfaction. More recently, in [39] the authors applied DRL to SFC embedding to learn a policy that accounts for both chain-level decisions and resource constraints. A strength of this approach is the ability to capture *long-term* effects of embedding decisions (e.g., how early placements affect future acceptance and congestion) through cumulative reward optimization, which lets it provide higher quality solutions at substantially faster convergence speed than baseline DRL methods. Conversely, ensuring strict feasibility (especially under multiple coupled constraints) often requires additional mechanisms (e.g., action masking, constrained RL, or post-processing), and the learned policy may be difficult to certify in terms of worst-case behavior. Finally, in [40] the authors propose a DRL-driven scheduling framework (combined with route optimization) for latency-sensitive SFC processing. The main strength is the explicit focus on *runtime* service performance (e.g., latency/rejection reduction in dynamic settings) and the ability to integrate scheduling decisions with network-aware routing strategies. Limitations include dependence on accurate operational feedback and on a training setup that matches the target deployment; additionally, latency-centric objectives may trade off against reliability constraints unless reliability is explicitly modeled and enforced.

Overall, metaheuristic and learning-based methods offer scalability and fast online decisions, but they usually provide weaker guarantees and rely on tuning or training assumptions. Our framework is complementary: by explicitly modeling load-dependent node reliability, incompatibility constraints and placement restrictions, and by providing an exact (or near-exact via cutting planes) solver-driven baseline on realistic topologies, it can be used to (i) benchmark heuristic/learned policies, (ii) quantify their optimality gaps under strict reliability targets, and (iii) serve as an optimization oracle within hybrid schemes where a learned policy is periodically calibrated against an exact solver.

Finally, we evaluated our model on realistic topologies and traffic datasets derived from the SNDLib library [23]. In contrast to path-based formulations such as [25], our model adopts a flow-based approach with embedded reliability and load-aware constraints. Moreover, we extend the model derived from our previous work [22], which we enhance with 0–1 per-node VNF installation constraints, inter-VNF compatibility constraints, and empirical evaluation against two newly defined baseline strategies lacking full reliability guarantees.

8. Conclusion

This paper addressed the challenge of optimizing Network Functions Virtualization deployments, focusing on efficient Virtual Network Function placement and resource allocation. By leveraging advanced optimization techniques, including novel linearization strategies and a cutting-plane heuristic with warm-start strategies, we transformed a non-convex MINLP formulation into a tractable MILP model, thereby effectively mitigating the inherent computational complexities of NFV systems.

Our framework systematically maps a wide range of VNF types to heterogeneous server nodes according to node congestion, and service requirements, thereby optimizing resource usage, balancing workloads and limiting redundancy while ensuring resilient end-to-end service performance. By explicitly addressing infrastructure cost minimization, the framework incorporates the fundamental trade-off between service quality, operational expenditures, and infrastructure maintenance, that is often encountered by virtual and cloud service providers.

We demonstrated via extensive experimental results on two real-world topologies (i.e., *Abilene* and *Germany50* networks from SNDLib) the effectiveness of our proposed solution. Regardless of the scenario, both workload distribution among servers and maximum server utilization improve with our optimization strategy–showing a 21% to 63% improvement in workload distribution, as measured by reductions in the Coefficient of Variation, and a 33% to 58% reduction in maximum server utilization–when the highest feasible service reliability guarantee is required. By comparing the results with those obtained by two baseline approaches that either do not consider end-to-end reliability requirements or impose them locally, we show that our model consistently improves the median per-request reliability by an average of 60% and 21%, respectively, with only a modest increase in the number of activated server nodes. Overall, the proposed optimization strategy demonstrated scalability by solving realistic instances in less than 7 min on a commodity server, irrespective of the size of the topology, the number of service requests and the tightness of the reliability constraint.

Several interesting tradeoffs emerged from the analysis of the results. On the one hand, we highlight that resource-constrained systems face higher risk of infeasibility under tight reliability requirements, while systems with greater capacity can sustain stricter guarantees without compromising performance. Adjusting reliability thresholds based on available capacity offers a practical approach to balancing workload distribution and maintaining deployment feasibility. On the other hand, we observe that when reliability is guaranteed locally, VNFs tend to be reassigned within the same nodes and paths used by non-reliable strategies, as long as server-level reliability bounds are satisfied. In contrast, stricter end-to-end reliability requirements may necessitate activating additional servers, thus rerouting service requests along alternative paths to ensure the desired guarantees. We believe these insights could serve as guidelines for virtual service providers to deliver scalable, cost-effective, and reliable NFV deployments.

To conclude, we underline that the present study has some limitations that open interesting directions for future work. First, we restrict each request to a single path and do not consider multi-path routing or traffic splitting, which could further enhance reliability and load balancing. Second, while our reliability model captures workload-dependent degradation at the node level, it assumes independent node failures and adopts a specific concave functional form; extending the framework to correlated failures or alternative reliability models would be valuable. Third, latency is not optimized explicitly, even though delay constraints can be readily added to the formulation as discussed in Section 3. Finally, our experimental comparison focuses on exact and relaxation-based baselines; integrating and empirically evaluating metaheuristic or learning-based placement strategies on the same scenarios is an important avenue for future research.

CRediT authorship contribution statement

Mohammad A. Raayatpanah: Writing – review & editing, Writing – original draft, Software, Methodology, Conceptualization; **Thomas Weise:** Writing – original draft, Supervision; **Jocelyne Elias:** Writing – review & editing, Writing – original draft, Supervision; **Fabio Martignon:** Writing – review & editing, Writing – original draft, Supervision; **Andrea Pimpinella:** Writing – review & editing, Writing – original draft, Supervision.

Data availability

Data will be made available on request.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Given her role as Editorial Board Member, Prof. Jocelyne Elias had no involvement in the peer review of this article and had no access to information regarding its peer review. Full responsibility for the editorial process for this article was delegated to another journal editor If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by the University of Montpellier's MAK'IT (Montpellier Advanced Knowledge Institute on Transitions) visiting scientist program, by projects *SERICS* (PE00000014) under the NRRP MUR program, funded by the EU - NGEU, and the PRIN project *NEWTON* (2022ZA8T22), funded by the European Union - NextGenerationEU.

References

[1] J.G. Herrera, J.F. Botero, Resource allocation in NFV: a comprehensive survey, IEEE Trans. Netw. Serv. Manage. 13 (3) (2016) 518–532. https://doi.org/10.1109/TNSM.2016.2598420

[2] B. Németh, N. Molner, J. Martín-Pérez, C.J. Bernardos, A. de la Oliva, B. Sonkoly, Delay and reliability-constrained VNF placement on mobile and volatile 5G infrastructure, IEEE Trans. Mob. Comput. 21 (9) (2022) 3150–3162. https://doi.org/10.1109/TMC.2021.3055426

[3] X. Gao, R. Liu, A. Kaushik, Virtual network function placement in satellite edge computing with a potential game approach, IEEE Trans. Netw. Serv. Manage. 19 (2) (2022) 1243–1259. https://doi.org/10.1109/TNSM.2022.3141165

[4] G. Frick, A. Lehmann, U. Trick, B. Ghita, Resilient placement of network functions in a WMN-based NFV infrastructure, IEEE Access (2025) 52538–52562. https://doi.org/10.1109/ACCESS.2025.3553810

[5] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, X. Fu, Recent advances of resource allocation in network function virtualization, IEEE Trans. Parallel Distrib. Syst. 32 (2) (2020) 295–314. https://doi.org/10.1109/TPDS.2020.3017001

[6] W. Attaoui, E. Sabir, H. Elbiaze, M. Guizani, VNF and CNF placement in 5G: recent advances and future trends, IEEE Trans. Netw. Serv. Manage. 20 (4) (2023) 4698–4733. https://doi.org/10.1109/TNSM.2023.3264005

[7] S. Burer, A.N. Letchford, Non-convex mixed-integer nonlinear programming: a survey, Surv. Oper. Res. Manage. Sci. 17 (2012) 97–106. https://doi.org/10.1016/j.sorms.2012.08.001

[8] B. Schroeder, E. Pinheiro, W.-D. Weber, DRAM errors in the wild: a large-scale field study, ACM SIGMETRICS Perform. Eval. Rev. 37 (1) (2009) 193–204. https://doi.org/10.1145/2492101.1555372

[9] S. Troia, M. Savi, G. Nava, L.M.M. Zorello, T. Schnpageser, G. Maier, Performance characterization and profiling of chained CPU-bound virtual network functions, Comput. Netw. 231 (2023) 109815. https://doi.org/10.1016/j.comnet.2023.109815

[10] Q. Ye, W. Zhuang, X. Li, J. Rao, End-to-end delay modeling for embedded VNF chains in 5G core networks, IEEE Internet Things J. 6 (1) (2018) 692–704. https://doi.org/10.1109/JIOT.2018.2853708

[11] H. Yu, T. Taleb, J. Zhang, Deterministic latency/jitter-aware service function chaining over beyond 5G edge fabric, IEEE Trans. Netw. Serv. Manage. 19 (3) (2022) 2148–2162. https://doi.org/10.1109/TNSM.2022.3151431

[12] M. Bouet, J. Leguay, T. Combe, V. Conan, Cost-based placement of vDPI functions in NFV infrastructures, Int. J. Netw. Manage. 25 (6) (2015) 490–506. https://doi.org/10.1002/nem.1920

[13] C. Ren, H. Li, Y. Li, Y. Wang, H. Xiang, X. Chen, On efficient service function chaining in hybrid software defined networks, IEEE Trans. Netw. Serv. Manage. 19 (2) (2021) 1614–1628. https://doi.org/10.1109/TNSM.2021.3123502

[14] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, S. Wright, Power awareness in network design and routing, in: IEEE Conference on Computer Communications (INFOCOM), IEEE, 2008, pp. 457–465. https://doi.org/10.1109/INFOCOM.2008.93

[15] S. Even, A. Itai, A. Shamir, On the complexity of time table and multi-commodity flow problems, in: 16th Annual Symposium on Foundations of Computer Science (sfcs 1975), 1975, pp. 184–193. https://doi.org/10.1109/SFCS.1975.21

[16] A.K. Andreas, J.C. Smith, Mathematical programming algorithms for two-path routing problems with reliability considerations, INFORMS J. Comput. 20 (4) (2008) 553–564. https://doi.org/10.1287/ijoc.1080.0266

[17] G.P. McCormick, Computability of global solutions to factorable nonconvex programs: Part I—Convex underestimating problems, Math. Program. 10 (1976) 147–175. https://doi.org/10.1007/BF01580665

[18] H.D. Sherali, H. Wang, Global optimization of nonconvex factorable programming problems, Math. Program. 89 (2001) 459–478. https://doi.org/10.1007/PL00011409

[19] H. Kellerer, U. Pferschy, D. Pisinger, The bounded knapsack problem, in: Knapsack Problems, Springer, 2004, pp. 185–209. https://doi.org/10.1007/978-3-540-24777-7_7

[20] M. Savi, M. Tornatore, G. Verticale, Impact of processing costs on service chain placement in network functions virtualizationDelay guaranteed,, in: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), IEEE, 2015, pp. 191–197. https://doi.org/10.1109/NFV-SDN.2015.7387426

[21] L.A. Wolsey, Integer Programming, John Wiley & Sons, 2020.

[22] M.A. Raayatpanah, T. Weise, J. Elias, F. Martignon, A. Pimpinella, A mixed-integer linear programming approach for congestion-aware optimized NFV deployment, in: 2025 23rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), IEEE, 2025, pp. 1–8. https://doi.org/10.23919/WiOpt66569.2025.11123231

[23] S. Orlowski, R. Wessäly, M. Pióro, A. Tomaszewski, SNDlib 1.0- Survivable network design library, Netw.: Int. J. 55 (3) (2010) 276–286.

[24] M.F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, On orchestrating virtual network functions, in: The 11th International Conference on Network and Service Management (CNSM), IEEE, 2015, pp. 50–56. https://doi.org/10.1109/CNSM.2015.7367338

[25] A. Mouaci, É. Gourdin, I. LjubiĆ, N. Perrot, Virtual network functions placement and routing problem: path formulation, in: 2020 IFIP Networking Conference (Networking), 2020, pp. 55–63.

[26] M. Zhu, F. He, E. Oki, Optimization model for multiple backup resource allocation with workload-dependent failure probability, IEEE Trans. Netw. Serv. Manage. 18 (3) (2021) 3733–3752. https://doi.org/10.1109/TNSM.2021.3079937

[27] M. Zhu, E. Oki, Robust function deployment against uncertain recovery time in different protection types with workload-dependent failure probability, Comput. Netw. 231 (2023) 109826. https://doi.org/10.1016/j.comnet.2023.109826

[28] M. Gao, B. Addis, M. Bouet, S. Secci, Optimal orchestration of virtual network functions, Comput. Netw. 142 (2018) 108–127. https://doi.org/10.1016/j.comnet.2018.06.006

[29] M. Taghavian, Y. Hadjadj-Aoul, G. Texier, N. Huin, P. Bertin, An approach to network service placement reconciling optimality and scalability, IEEE Trans. Netw. Serv. Manage. 20 (3) (2023) 2218–2229. https://doi.org/10.1109/TNSM.2023.3284602

[30] Y. Wu, W. Zheng, Y. Zhang, J. Li, Reliability-aware VNF placement using a probability-based approach, IEEE Trans. Netw. Serv. Manage. 18 (3) (2021) 2478–2491. https://doi.org/10.1109/TNSM.2021.3093199

[31] P.K. Thiruvasagam, A. Chakraborty, A. Mathew, C.S.R. Murthy, Reliable placement of service function chains and virtual monitoring functions with minimal cost in softwarized 5G networks, IEEE Trans. Netw. Serv. Manage. 18 (2) (2021) 1491–1507. https://doi.org/10.1109/TNSM.2021.3056917

[32] L. Magoula, S. Barmpounakis, I. Stavrakakis, N. Alonistioti, A genetic algorithm approach for service function chain placement in 5G and beyond, virtualized edge networks, Comput. Netw. 195 (2021) 108157. https://doi.org/10.1016/j.comnet.2021.108157

[33] T.V. Doan, G.T. Nguyen, M. Reisslein, F.H.P. Fitzek, SAP: subchain-aware NFV service placement in mobile edge cloud, IEEE Trans. Netw. Serv. Manage. 20 (1) (2023) 319–341. https://doi.org/10.1109/TNSM.2022.3201388

[34] S.R. Zahedi, S. Jamali, P. Bayat, A power-efficient and performance-aware online virtual network function placement in SDN/NFV-enabled networks, Comput. Netw. 205 (2022) 108753. https://doi.org/10.1016/j.comnet.2021.108753

[35] K. Alizadeh Noghani, A. Kassler, J. Taheri, P. Öhlén, C. Curescu, Multiobjective genetic algorithm for fast service function chain reconfiguration, IEEE Trans. Netw. Serv. Manage. 20 (3) (2023) 3501–3522. https://doi.org/10.1109/TNSM.2022.3195820

[36] J. Pei, P. Hong, M. Pan, J. Liu, J. Zhou, Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks, IEEE J. Sel. Areas Commun. 38 (2) (2020) 263–278. https://doi.org/10.1109/JSAC.2019.2959181

[37] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, F. Liberal, Virtual network function placement optimization with deep reinforcement learning, IEEE J. Sel. Areas Commun. 38 (2) (2020) 292–303. https://doi.org/10.1109/JSAC.2019.2959183

[38] N. He, S. Yang, F. Li, S. Trajanovski, L. Zhu, Y. Wang, X. Fu, Leveraging deep reinforcement learning with attention mechanism for virtual network function placement and routing, IEEE Trans. Parallel Distrib. Syst. 34 (4) (2023) 1186–1201. https://doi.org/10.1109/TPDS.2023.3240404

[39] Y. Liu, J. Zhang, Service function chain embedding meets machine learning: deep reinforcement learning approach, IEEE Trans. Netw. Serv. Manage. 21 (3) (2024) 3465–3481. https://doi.org/10.1109/TNSM.2024.3353808

[40] Z. Liu, Z. Shu, S. Chen, Y. Zhong, J. Lin, Low-latency virtual network function scheduling algorithm based on deep reinforcement learning, Comput. Netw. 246 (2024) 110418. https://doi.org/10.1016/j.comnet.2024.110418