Dynamic VNF Placement, Resource Allocation and Traffic Routing in 5G

Original
Dynamic VNF Placement, Resource Allocation and Traffic Routing in 5G / Golkarifarda, Morteza; Chiasserini, Carla Fabiana; Malandrino, Francesco; Movaghar, Ali. - In: COMPUTER NETWORKS. - ISSN 1389-1286. - STAMPA. - (2021). [10.1016/j.comnet.2021.107830]

Availability:
This version is available at: https://hdl.handle.net/20.500.14243/421734

Publisher:
Elsevier

Published
DOI:10.1016/j.comnet.2021.107830

Publisher copyright
Elsevier postprint/Author's Accepted Manuscript

© 2021. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/.The final authenticated version is available online at:
http://dx.doi.org/10.1016/j.comnet.2021.107830

(Article begins on next page)

04 December 2024

Dynamic VNF Placement, Resource Allocation and Traffic Routing in 5G

Morteza Golkarifard[a,*], Carla Fabiana Chiasserini[b,c,d], Francesco Malandrino[c,d], Ali Movaghar[a]

[a] *Sharif University of Technology, Tehran, Iran*
[b] *Politecnico di Torino, Torino, Italy*
[c] *CNR-IEIIT, Italy*
[d] *CNIT, Italy*

Abstract

5G networks are going to support a variety of vertical services, with a diverse set of key performance indicators (KPIs), by using enabling technologies such as software-defined networking and network function virtualization. It is the responsibility of the network operator to efficiently allocate the available resources to the service requests in such a way to honor KPI requirements, while accounting for the limited quantity of available resources and their cost. A critical challenge is that requests may be highly varying over time, requiring a solution that accounts for their dynamic generation and termination. With this motivation, we seek to make joint decisions for request admission, resource activation, VNF placement, resource allocation, and traffic routing. We do so by considering real-world aspects such as the setup times of virtual machines, with the goal of maximizing the mobile network operator profit. To this end, first, we formulate a one-shot optimization problem which can attain the optimum solution for small size problems given the complete knowledge of arrival and departure times of requests over the entire system lifespan. We then propose an efficient and practical heuristic solution that only requires this knowledge for the next time period and works for realistically-sized scenarios. Finally, we evaluate the performance of these solutions using real-world services and large-scale network topologies. Results demonstrate that our heuristic solution performs better than a state-of-the-art online approach and close to the optimum.

1. Introduction

5G networks are envisioned to support a variety of services belonging to vertical industries (*e.g.*, autonomous driving, media, and entertainment) with a diverse set of requirements. Services are defined as a directed graph of virtual network functions (VNFs) with specific and varying key performance indicators (KPIs), *e.g.*, throughput, and delay. Requests for these services arrive over time and mobile network operators (MNOs) are responsible for efficiently satisfy such a demand, by fulfilling their associated KPI while minimizing the cost for themselves.

As a result of the softwarization of 5G-and-beyond networks, enabled by software-defined networking (SDN) and network function virtualization (NFV), it is now feasible to use general-purpose resources (*e.g.*, virtual machines) to implement the VNFs required by the different service. The decision on which resources to associate with which VNF and service is made by a network component called *orchestrator*, as standardized by ETSI [1]. Without loss of generality, we focus only on computational and communication resources (*e.g.*, virtual machines and the links connecting them); notice, however, that our proposed framework is applicable to other resource types (*e.g.*, storage).

The network orchestrator makes the following decisions [1]:

- *admission* of requests;
- *activation/deactivation* of VMs;
- *placement* of VNF instances therein;
- *assignment* of CPU to VMs for running the hosted VNF instances;
- *routing* of traffic through physical links.

These decisions are clearly mutually dependent, and therefore should be made *jointly*, in order to account for the – often nontrivial – ways in which they influence one another. The focus of this paper is thus to consider the *joint* requests admission, VM activation/deactivation, VNF placement, CPU assignment, and traffic routing problem in order to maximize the MNO profit, while considering:

- the properties of each VNF,
- the KPI requirements of each service,
- the capabilities of VMs and PoPs (points of presence, *e.g.*, datacenters) and their latency,
- the capacity and latency of physical links,

*Corresponding author.
Email address: golkari@ce.sharif.edu (Morteza Golkarifard)

- the VMs setup times,
- the arrival and departure times of service requests.

As better discussed in Sec. 2, some of these factors are simplified, or even neglected, in existing works on 5G orchestration. Notably, we account for the VM setup time, which becomes a significant factor in (for example) IoT applications, when requests are often short-lived. Ignoring setup (and tear-down) times can reduce the optimality of existing solutions.

Furthermore, we account for the fact that different VNFs may have different levels of complexity, therefore, different quantities of computational resources may be needed to attain the same KPI target. Inspired by several works in the literature [2], we model individual VNFs as queues and services as queuing networks. Critically, unlike traditional queuing networks, the quantity of traffic (*i.e.*, the number of clients in queues) *can change* across queues, as VNFs can drop some packets (*e.g.*, firewalls) or change the quantity thereof (*e.g.*, video transcoders). Our model accounts for this important aspect by replacing traditional flow conservation constraints with a *generalized flow conservation* law, allowing us to describe arbitrary services with arbitrary VNF graphs.

Given this model, we formulate a one-shot optimization problem which, assuming perfect knowledge of future requests, allows us to maximize the MNO profit. Given the NP-hardness of such a problem and the fact that knowledge of future requests is usually not available, we propose MaxSR, an efficient heuristic algorithm which will be invoked periodically based on the knowledge of requests within each time period. The proposed method can achieve a near-optimal solution for large-scale network scenarios. We evaluate MaxSR compared to the optimum and other benchmarks using real-world services and different network scenarios.

In summary, the main contributions of this paper are as follows:

- we propose a complete model for the main components of 5G, both in terms of vertical services (dynamic requests, VNFs, and services KPIs) and in terms of resources (*e.g.* VMs and links);
- our model accounts for the time variations of service requests, and dynamically allocates the computational and network resources while considering VMs setup times. It can also accommodate a diverse set of VNFs in terms of computational complexity and KPI requirements, multiple VNF instances, and arbitrary VNF graphs with several ingress and egress VNFs, rather than a simple chain or directed acyclic graph (DAG);
- we formulate a one-shot optimization problem as a Mixed-Integer Non-linear Programming (MINLP) to make a joint decision on VM state, VNF placement, CPU assignment, and traffic routing based on the complete requests statistics over the entire system lifespan;
- we propose MaxSR, an efficient near-optimal heuristic algorithm to solve the aforementioned problem based on the knowledge of the near future for large scale network scenarios;
- finally, we compare MaxSR with optimum and the online approach Best-Fit, through extensive experiments using synthetic services and requests, and different network scenarios.

The rest of the paper is organized as follows. Sec. 2 reviews related works. Sec. 3 describes the system model and problem formulation, while Sec. 4 clarifies our solution strategy. Finally, Sec. 5 presents our numerical evaluation under different network scenarios, Sec. 6 discusses the applicability and scalability of the proposed solution, and Sec. 7 concludes the paper.

2. Related Work

Several works have addressed VNF placement and traffic routing, as exemplified by the survey paper [3]. In most of these works, the problem is formulated as a Mixed-Integer Linear Program (MILP) with a different set of objectives and constraints. Such an approach can yield exact solutions, but merely works for small instances; therefore, heuristic algorithms that offer a near-optimal solution have also been presented.

In particular, a first body of works provides a one-time VNFs placement, given the incoming service requests. Since this method leaves already placed VNFs intact, it can lead to a sub-optimal solution when the traffic varies over time. Examples of such an approach can be found in [4–10], which aim at minimizing a cost function, *e.g.*, operational cost, QoS degradation cost, server utilization, or a combination of them, and assume that there are always enough resources to serve the incoming requests. Among them, Cohen *et al.* [4] propose an approximation algorithm to place sets of VNFs in an optimal manner, while approximating to the constraints by a constant factor. Pham *et al.* [8] introduce a distributed solution based on a Markov approximation technique to place chains of VNFs where the cost enfolds the delay cost, in addition to the cost of traffic and server. [9], instead, addresses the same problem but aims at minimizing the energy consumption, given constraints on end-to-end latency for each flow and server utilization. Pei *et al.* [10] propose an online heuristic for this problem, by which VNF instances are deployed and connected using the shortest path algorithm, in order to minimize the number of VNF instances and satisfy their end-to-end delay constraint.

Another thread of works focuses on an efficient admission policy that maximizes the throughput or revenue of admitted requests [11–14]. In particular, Sallam *et al.* [11] formulate joint VNF placement and resource allocation

problem to maximize the number of fully served flows considering the budget and capacity constraints. They leverage the sub-modularity property for a relaxed version of the problem and propose two heuristics with a constant approximation ratio. [12] studies the joint VNF placement and service chain embedding problem, so as to maximize the revenue from the admitted requests. A similar problem is tackled in [14] and [13] but for an online setting where the requests should be admitted and served upon their arrival. Zhou *et al.* [13], on the other hand, first formulate a one-shot optimization problem over the entire system lifespan and then leverage the primal-dual method to design an online solution with a theoretically proved upper bound on the competitive ratio.

A different approach is adopted in [15–21] where VNF placement can be readjusted through VNF sharing and migration, to optimally fit time-varying service demands. [15] and [16] propose algorithms that properly scale over-utilized or under-utilized VNF instances based on the estimation of future service demands. Jia *et al.* [17] propose an online algorithm with a bounded competitive ratio that dynamically deploys delay constrained service function chains across geo-distributed datacenters minimizing operational costs.

Request admission control has instead been considered in [18–21]. More in detail, Li *et al.* [18] propose a proactive algorithm that dynamically provisions resources to admit as many requests as possible with a timing guarantee. Similarly, [19] admits requests and places their VNFs in the peak interval, but minimizes the energy cost of VNF instances by migration and turning off empty ones in the off-peak interval. Liu *et al.* [20] envision an algorithm that maximizes the service provider's profit by periodically admitting new requests and rearranging the current-served ones, while accounting for the operational overhead of migration. Finally, leveraging VNF migration and sharing, [21] proposes an online algorithm to maximize throughput while minimizing service cost and meeting latency constraints.

Relevant to our work are also studies that target specifically 5G systems, although they merely consider the link delay and neglect processing delays in the servers. An example can be found in [2], which models VMs as M/M/1 PS queues, and proposes an optimization formulation and a heuristic solution to minimize the average service delay, while meeting the constraints on the links and host capacities. The works in [22] and [23] aim instead to minimize, respectively, the operational cost and the energy consumption of VMs and links while ensuring end-to-end delay KPI. [22] also allows for VNF sharing and studies the impact of applying priorities to different services within a shared VNF. Zhang *et al.* [24] tackle the request admission problem to maximize the total throughput, neglecting instead queuing delay at VMs. We summarize most related solutions and compare them with our work in Table 1.

We remark that most of the above works present proactive approaches, and only deal with either cost minimization or request admission. On the contrary, we focus on dynamic resource activation, VNF placement, and CPU assignment to maximize the revenue from admitted requests over the entire system lifespan, while minimizing the deployment costs and accounting for some practical issues. Our proactive MINLP formulation of the problem extends existing models by accounting for the maximum end-to-end delay as the main KPI, while our heuristic is a practical and scalable solution, which periodically admits new requests and readjusts the existing VNF deployment. To the best of our knowledge, this is the first dynamic solution for service orchestration in 5G networks.

3. System Model and Problem Formulation

In this section, first we describe our system model supported by a simple example. Later, we formulate the *joint* requests admission, VM activation, VNF placement, CPU assignment, and traffic routing problem; a discussion of the problem time complexity follows. The frequently used notation is summarized in Table 2.

3.1. System Model

Physical infrastructure. Let $\mathcal{G} = (\mathcal{M}, \mathcal{E})$ be a directed graph representing the physical infrastructure network, where each node $m \in \mathcal{M}$ is either a VM or a network node (*i.e.*, a router or a switch). A VM m has maximum computational capacity $C_{vm}(m)$. Set $\mathcal{E}$ denotes the physical links connecting the network nodes. We define $B(e)$ and $D_{phy}(e)$ as, respectively, the bandwidth and delay of physical link $e \in \mathcal{E}$. Time is discretized into *steps*, $\mathcal{T} = \{1, 2, \ldots, T\}$, and we assume that at every time step a VM may be in one of the following states: *terminated*, *turning-on*, or *active*. Specifically, VMs can only be used when they are active, and they need to be turned-on one time step before being active. Based on the measurements reported in [16], we also consider the traffic flow migration time to be negligible with respect to the VM setup time.

Each VM can host one VNF and belongs to a datacenter $d \in \mathcal{D}$; we denote the available amount of computational resources in datacenter d by $C_{dc}(d)$ and the set of VMs within d with $\mathcal{M}_d$. In the physical graph G, physical links within datacenters are assumed to be ideal, *i.e.*, they have no capacity limit and zero delay. Let logical link $l \in \mathcal{L}$ be a sequence of physical links connecting two VMs, $src(l)$ and destination $dst(l)$, then we define end-to-end path $p \in \mathcal{P}$ as a sequence of logical links.

Services. We represent each service $s \in \mathcal{S}$ with a *VNF Forwarding Graph (VNFFG)*, where the nodes are VNFs $q \in \mathcal{Q}$, and the directed edges show how traffic traverses the VNFs. VNFFG can be any general graph with possibly several ingress and egress VNFs. We denote the total new traffic, entering the ingress VNFs of service s, by $\lambda_{new}(s)$. A traffic packet of service s, processed in VNF q_1, is forwarded to VNF q_2 at time t with probability of $\mathbb{P}(s, q_1, q_2, t)$. Similarly, $\mathbb{P}(s, \circ, q, t)$ is the probability that

Table 1: Summary of related solutions

Ref.	Objective	request admission	resource activation	VNF placement	traffic routing	CPU assignment	arbitrary VNFFG	VNF complexity[a]	VNF traffic scaling[b]	target max. delay[c]	link latency	processing delay	queuing model	link capacity	node capacity	VM setup time	online solution	dynamic workload
ours	maximize revenue while minimizing cost	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
[2]	minimize service average delay	×	×	✓	×	✓	✓	×	×	×[1]	✓	✓	✓	✓	✓	×	×	×
[8]	minimize operational and delay costs	×	×	✓	×	×	×	×	×	×	✓	×	×	✓	✓	×	×	✓
[9]	minimize energy	×	✓	✓	✓	×	×	×	×	✓	✓	×	×	✓	✓	×	✓	✓
[10]	minimize embedding cost	✓	×	✓	✓	×	×	×	×	×	✓	✓	×	✓	✓	×	×	✓
[11]	maximize served traffic	×	×	✓	×	✓	×	×	×	×	×	×	×	×	✓	×	×	×
[12]	maximize revenue	✓	×	✓	✓	×	✓	×	×	×	×	×	×	✓	✓	×	×	✓
[13]	maximize revenue	✓	×	✓	×	×	×	×	×	×	×	×	×	✓	✓	×	✓	✓
[14]	maximize revenue	✓	×	✓	✓	×	×	×	×	×	×	×	×	✓	✓	×	✓	✓
[15]	minimize cost	×	×	✓	✓	×	×	×	×	×	×	×	×	✓	✓	×	✓	✓
[16]	minimize # of used VMs	×	×	✓	×	✓	×	×	×	×	×	×	×	×	✓	✓	×	✓
[17]	minimize operational and delay costs	×	×	✓	✓	×	×	×	✓	×	✓	×	×	×	✓	×	✓	✓
[18]	maximize # of admitted reqs.	✓	×	✓	✓	×	×	×	×	✓	✓	✓	×	✓	✓	×	✓	✓
[19]	minimize energy and VNF reconfiguration costs	✓	×	✓	✓	✓	×	×	×	×	×	×	×	✓	✓	✓	×	✓
[20]	maximize profit	✓	×	✓	✓	×	✓	×	×	×	×	×	×	✓	✓	×	×	✓
[21]	maximize throughput while minimizing operational cost	✓	×	✓	✓	×	×	×	×	✓	✓	×	×	×	✓	×	×	✓
[22]	minimize operational cost	×	✓	✓	×	✓	✓	✓	×	×[1]	×	✓	✓	×	✓	×	×	×
[23]	minimize energy	×	✓	✓	✓	✓	✓	✓	✓	×[1]	✓	✓	×	✓	✓	×	×	×
[24]	maximize total throughput	✓	×	✓	×	×	×	×	×	✓	✓	✓	×	✓	✓	×	×	×

[a] VNFs can have different computational complexities.
[b] VNFs can change the quantity of outgoing traffic.
[c] maximum end-to-end delay which traffic packets of a service can tolerate.
[1] average tolerable end-to-end delay is considered.

a new traffic packet of service s starts getting service in ingress VNF q at time t, and $\mathbb{P}(s, q, \circ, t)$ is the probability that a traffic packet of service s, already served at egress VNF q, departs service s at time t. For each service s, we consider its target delay, $D_{QoS}(s)$, as the most critical KPI, specifying the maximum tolerable end-to-end delay for the traffic packets of s.

VNFs can have different processing requirements depending on their computational complexity. We denote by $\omega(q)$ the computational capability that VNF q needs to process one unit of traffic. Some VNFs may not find sufficient resources on a single VM to completely serve the traffic while satisfying the target delay. Thus, multiple instances can be created, with $N(s, q)$ being the maximum number of instances of VNF q at each point in time. Instances of the same VNF can be deployed either within the same datacenter or at different datacenters; in the latter case, the traffic between each pair of VNFs must be splitted through different logical links that connect the VMs running the corresponding VNF instances.

Different requests for the same services may arrive over time; we denote with K_s the set of all service requests for service s, and characterize the generic service request $k \in \mathcal{K}$ with its arrival time $t_{arv}(k)$ and departure time $t_{dpr}(k)$. Due to slice *isolation* requirements [25], we assume that the VNF instances of different service requests are not shared with other service requests.

Example. Figure 1 represents a possible deployment of two sample services, vehicle collision detection (VCD) and video on-demand (VoD), on the physical graph (Figure 1c) in a single time step. VCD is a low-latency service with a very low target delay D_{QoS}, and VoD is a traffic intensive service with a high λ_{new}. Figure 1a and Figure 1b depict the VNFFGs of the VCD and VoD services,

Table 2: Notation (sets, variables, and parameters)

Symbol	Description
$\mathcal{D}$	Set of datacenters
$\mathcal{E}$	Set of physical links
$\mathcal{K}$	Set of service requests
$\mathcal{L}$	Set of logical links
$\mathcal{M}$	Set of VMs
$\mathcal{P}$	Set of end-to-end paths
$\mathcal{Q}$	Set of VNFs
$\mathcal{S}$	Set of services
$\mathcal{T}$	Set of time steps
$\mathcal{W}_s$	Set of paths from ingress VNFs to egress VNFs in VNF graph of service s
$A(k,m,q,t)$	Whether to deploy VNF q of service request k at VM m at time t
$D(k,m,q,t)$	Traffic departing VM m for VNF q of service request k at time t
$F(k,l,q_1,q_2,t)$	Equal to 1 when $\rho(k,l,q_1,q_2,t) > 0$
$I(k,m,q,t)$	Traffic entering VM m for VNF q of service request k at time t
$L(e,t)$	Traffic on physical link e at time t
$O(m,t)$	Whether VM m is *active* at time t
$R(m,t)$	Average time for a request to be processed at VM m at time t
$U(m,t)$	Whether VM m is *turning-on* at time t
$V(k,t)$	Whether service request k is *active* at time t
$\mu(k,m,q,t)$	Service rate to assign to VM m for VNF q of service request k at time t
$\rho(k,l,q_1,q_2,t)$	Fraction of traffic from VNF q_1 to q_2 of service request k, through logical link l at time t
$B(e)$	Bandwidth of physical link e
$C_{dc}(d)$	Computational capacity of datacenter d
$C_{vm}(m)$	Computational capacity of VM m
$D_{QoS}(s)$	Target delay for service s
$D_{log}(l)$	Delay of logical link l
$D_{phy}(e)$	Delay of physical link e
$N(s,q)$	Maximum number of instances for VNF q of service s
$X_{cpu}(m)$	Cost for VM m to process one unit of computation in one time step
$X_{idle}(m)$	Fixed cost incurred when VM m is *turning-on* or *active* in one time step
$X_{link}(e)$	Cost of data transmission through physical link e in one time step
$X_{rev}(s)$	Revenue from serving one traffic unit of service s
$\Lambda(s,q_1,q_2,t)$	Traffic from VNF q_1 to q_2 for service s at time t
$\mathbb{P}(s,q_1,q_2,t)$	Probability that traffic processed at VNF q_1 is forwarded to VNF q_2 of service s at time t
$\alpha(s,q)$	Ratio of outgoing traffic to incoming traffic for VNF q of service s
$\lambda_{new}(s)$	New traffic for service s
$\omega(q)$	Computation capability required for one traffic unit at VNF q
$t_{arv}(k)$	Arrival time of service request k
$t_{dpr}(k)$	Departure time of service request k

respectively, where the numbers on the edges represent the transition probability of traffic packets between corresponding VNFs. The physical graph contains a set of datacenters $\mathcal{D} = \{d_1, d_2, d_3\}$ with computational capability C_{dc}. Datacenters are connected to each other using a switch and physical links with bandwidth B and a latency D_{phy}. VMs within each datacenter are denoted by sets $\mathcal{M}_{d_1} = \{m_1, m_2\}$, $\mathcal{M}_{d_2} = \{m_3, m_4\}$, and $\mathcal{M}_{d_3} = \{m_5, m_6, m_7\}$, each with computational capability C_{vm}. As depicted in Figure 1c, service VCD is deployed within datacenter d_2 to avoid inter-datacenter network latency. Service VoD is deployed across datacenter d_1 and third-party datacenter d_3. VNF *transcoder*, having high computational complexity ω, requires two instances in datacenters d_3 to fully serve the traffic.

3.2. Problem Formulation

In this section, we first describe the decisions that have to be made to map the service requests onto network resources. Then we formalize the system constraints and the objective using the model presented in Sec. 3.1, along with the decision variables we define. In general, given the knowledge of the future arrival and departure times of service requests, we should make the following decisions:

- service request activation, *i.e.*, when service requests get served;
- VM activation/deactivation, *i.e.*, when VMs are set up or terminated;
- VNF instance placement, *i.e.*, which VMs have to run VNF instances;
- CPU assignment, *i.e.*, how much computational capability shall be assigned to a VM to run the deployed VNF;
- traffic routing, *i.e.*, how traffic between VNFs is routed through physical links.

Service request activation. Let binary variable $V(k,t) \in \{0,1\}$ denote whether service request k is being served at

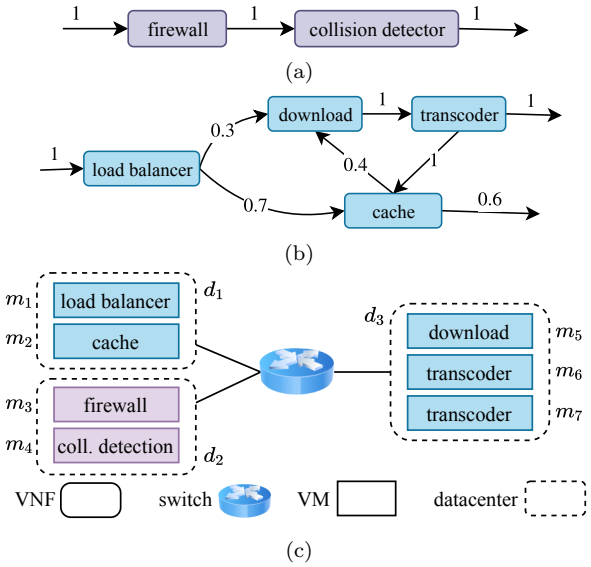


Figure 1: VNFFG of (a) vehicle collision detection (VCD) service and (b) video on-demand (VoD) service. The number on edges represents transition probability of traffic packets. (c) Physical graph including three datacenters connected using a switch.

time t. Once admitted, a service request has to be provided for all its lifetime duration. Given service request arrival time $t_{arv}(k)$ and departure time $t_{dpr}(k)$, this translates into:

$$V(k,t) = 0, \forall k \in \mathcal{K}, t \in \mathcal{T} : t < t_{arv}(k) \vee t \geq t_{dpr}(k). \quad (1)$$

VNF instances. The following constraint limits the number of deployed instances of VNF q of any service request $k \in \mathcal{K}_s$ to be less than $N(s,q)$ at any point in time:

$$\sum_{m \in \mathcal{M}} A(k,m,q,t) \leq N(s,q), \quad \forall t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, q \in \mathcal{Q}, \quad (2)$$

where binary variable $A(k,m,q,t)$ represents whether VNF q of service request k is placed on VM m at time t. The network slice isolation property of 5G networks prevents VNF sharing among requests for different services. In addition, at most one VNF instance can be deployed on any VM, *i.e.*,

$$\sum_{k \in \mathcal{K}} \sum_{q \in \mathcal{Q}} A(k,m,q,t) \leq 1, \qquad \forall m \in \mathcal{M}, t \in \mathcal{T}. \quad (3)$$

VM states. We define two binary variables $U(m,t)$ and $O(m,t)$ to represent whether VM m is *turning-on* or *active* at time t, respectively. We formulate a simple constraint to prevent VMs from being concurrently turning-on and active at any time, *i.e.*,

$$O(m,t) + U(m,t) \leq 1, \qquad \forall m \in \mathcal{M}, t \in \mathcal{T}. \quad (4)$$

The following constraint enforces that VM m can be *active* at time t only if it has been turning-on or active in the previous time step:

$$O(m,t) \leq O(m,t-1) + U(m,t-1), \forall m \in \mathcal{M}, t \in \mathcal{T}. \quad (5)$$

VMs are able to run VNFs only when they are active, *i.e.*,

$$\sum_{k \in \mathcal{K}} \sum_{q \in \mathcal{Q}} A(k,m,q,t) \leq O(m,t), \quad \forall m \in \mathcal{M}, t \in \mathcal{T}. \quad (6)$$

Computational capacity. Let real variable $\mu(k,m,q,t)$ represent the service rate assigned to VM m to run VNF q of service request k at time t. Multiplying it by $\omega(q)$, we have the amount of computation capability assigned to VM m to run VNF q at time t. The limited computational capability of datacenters and VMs denoted, respectively, by $C_{dc}(d)$ and $C_{vm}(m)$, should not be exceeded at any point in time. We describe such a limitation by imposing:

$$\sum_{m \in \mathcal{M}_d} \sum_{k \in \mathcal{K}} \sum_{q \in \mathcal{Q}} \mu(k,m,q,t) \cdot \omega(q) \leq C_{dc}(d), \quad \forall t \in \mathcal{T}, d \in \mathcal{D}, \quad (7)$$

where the sum on the left-hand side of the inequality is over all VMs within datacenter d. Similarly, for the VMs we have

$$\mu(k,m,q,t) \cdot \omega(q) \leq A(k,m,q,t) \cdot C_{vm}(m), \quad \forall t \in \mathcal{T}, k \in \mathcal{K}, q \in \mathcal{Q}, m \in \mathcal{M}, \quad (8)$$

where $A(k,m,q,t)$ on the right-hand side of the inequality enforces zero service rate for VM m when no VNF is placed therein.

KPI target fulfillment. Whenever a service request is being served, *i.e.*, $V(k,t) = 1$, all the traffic in the corresponding VNFFG should be carried by the underlying physical links. The following constraint ensures this condition for the traffic between each pair of VNFs at any point in time:

$$\sum_{l \in \mathcal{L}} \rho(k,l,q_1,q_2,t) \geq V(k,t), \quad \forall t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, q_1, q_2 \in \mathcal{Q} : \mathbb{P}(s,q_1,q_2,t) > 0. \quad (9)$$

Real variable $\rho(k,l,q_1,q_2,t)$ shows the fraction of traffic from VNF q_1 to q_2 of service request k that is routed through logical link l at time t. As mentioned, the traffic flow from VNF q_1 to VNF q_2 may be splitted into several logical links (see Eq. (2)). Moreover, since we consider multi-path routing, there may be multiple logical links between each pair of VNF instances. Therefore, constraint (9) implies that for any service request k requesting traffic from VNF q_1 to q_2 at time t (*i.e.*, $\mathbb{P}(s,q_1,q_2,t) > 0$), the sum of all fractional traffic going though any logical link, should be equal to 1 at any time when the service request is being served.

The above constraint does not include ingress and egress traffic. To account for such contributions, we need to introduce *dummy* nodes in the VNFFG and the physical graph. We add an end-point *dummy* VNF, $\circ$ in every VNFFG, which is directly connected to all ingress and egress VNFs and a *dummy* VM in the physical graph which is directly connected to all VMs. We define $\mathcal{L}_\circ$ as the set of *dummy* logical links which start from or end at the *dummy* VM. We assume that *dummy* logical links are ideal, *i.e.*, they have no capacity limit and zero delay and cost. We can now formulate the associated traffic constraints as:

$$\sum_{l \in \mathcal{L}_\circ} \rho(k,l,\circ,q,t) \geq V(k,t), \quad \forall t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, q \in \mathcal{Q} : \mathbb{P}(s,\circ,q,t) > 0, \quad (10)$$

$$\sum_{l \in \mathcal{L}_\circ} \rho(k,l,q,\circ,t) \geq V(k,t), \quad \forall t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, q \in \mathcal{Q} : \mathbb{P}(s,q,\circ,t) > 0, \quad (11)$$

where $\rho(k,l,\circ,q,t)$ and $\rho(k,l,q,\circ,t)$ are the fraction of new traffic entering ingress VNF q and the fraction of traffic departing from egress VNF q, respectively, going through logical link l at time t.

Placement. We can now correlate the routing decisions ρ and the placement decisions A as

$$\rho(k,l,q_1,q_2,t) \leq A(k,m,q_2,t), \quad \forall t \in \mathcal{T}, k \in \mathcal{K}, q_1 \in \mathcal{Q} \cup \{\circ\}, q_2 \in \mathcal{Q}, m \in \mathcal{M}, l \in \mathcal{L} \cup \mathcal{L}_\circ : dst(l) = m. \quad (12)$$

The above constraint implies that whenever there is an incoming traffic to VNF q_2 through logical link l whose destination is VM m, *i.e.*, $dst(l) = m$, VNF q_2 is deployed at VM m. Similarly, whenever there is an outgoing traffic from VNF q_1 through logical link l whose source is VM m, *i.e.*, $src(l) = m$, VNF q_1 is deployed at VM m:

$$\rho(k,l,q_1,q_2,t) \leq A(k,m,q_1,t), \quad \forall t \in \mathcal{T}, k \in \mathcal{K}, q_1 \in \mathcal{Q}, q_2 \in \mathcal{Q} \cup \{\circ\} \quad m \in \mathcal{M}, l \in \mathcal{L} \cup \mathcal{L}_\circ : src(l) = m. \quad (13)$$

System stability. Let $\lambda(s,q,t)$ denote the total incoming traffic of VNF q of service s at time t. $\lambda(s,q,t)$ equals the sum of ingress traffic and the traffic coming from other VNFs to VNF q of service s at time t:

$$\lambda(s,q,t) = \lambda_{new}(s) \cdot \mathbb{P}(s,\circ,q,t) + \sum_{q' \in \mathcal{Q} \setminus \{q\}} \lambda(s,q',t) \cdot \mathbb{P}(s,q',q,t). \quad (14)$$

Using $\lambda(s,q,t)$, the amount of traffic from VNF q_1 to VNF q_2 of service s at time t can be represented as:

$$\Lambda(s,q_1,q_2,t) = \lambda(s,q_1,t) \cdot \mathbb{P}(s,q_1,q_2,t). \quad (15)$$

We can now define an auxiliary variable to represent the incoming traffic of VNF q of service request k, which enters VM m at time t:

$$I(k,m,q,t) = \sum_{q' \in \mathcal{Q} \cup \{\circ\}} \sum_{\substack{l \in \mathcal{L} \cup \mathcal{L}_\circ : \\ dst(l) = m}} \rho(k,l,q',q,t) \cdot \Lambda(s,q',q,t), \quad t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, q \in \mathcal{Q}, m \in \mathcal{M}, \quad (16)$$

where the summation is over all logical links ending at VM m. Finally, we describe the *system stability* requirement, which imposes the incoming traffic not to exceed the assigned service rate for each VNF q of service request k on VM m, at any point in time:

$$I(k,m,q,t) \leq \mu(k,m,q,t), \quad \forall t \in \mathcal{T}, k \in \mathcal{K}, q \in \mathcal{Q}, m \in \mathcal{M}. \quad (17)$$

Generalized flow conservation. Our model captures the possibility of having VNFs for which, due to processing, the amount of incoming and that of outgoing traffic are different. We define the scaling factor $\alpha(s,q,t)$ as the ratio of outgoing traffic to incoming traffic for VNF q of service s at time t:

$$\alpha(s,q,t) = \frac{\sum_{q' \in \mathcal{Q} \cup \{\circ\}} \Lambda(s,q,q',t)}{\sum_{q' \in \mathcal{Q} \cup \{\circ\}} \Lambda(s,q',q,t)}, t \in \mathcal{T}, s \in \mathcal{S}, q \in \mathcal{Q}. \quad (18)$$

We also define auxiliary variable $D(k,m,q,t)$ to represent the outgoing traffic of VNF q of service request k departing VM m at time t:

$$D(k,m,q,t) = \sum_{q' \in \mathcal{Q} \cup \{\circ\}} \sum_{\substack{l \in \mathcal{L} \cup \mathcal{L}_\circ : \\ src(l) = m}} \rho(k,l,q,q',t) \cdot \Lambda(s,q,q',t), \quad t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, q \in \mathcal{Q}, m \in \mathcal{M}, \quad (19)$$

where the right-hand side enfolds all traffic flowing through logical links starting from VM m. We can then formulate the *generalized flow conservation law* for each VNF q of service request k on VM m at time t:

$$D(k,m,q,t) = \alpha(s,q,t) \cdot I(k,m,q,t), \quad \forall t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, q \in \mathcal{Q}, m \in \mathcal{M}, \quad (20)$$

which implies that for each VNF q of service request k on VM m, at any time, the outgoing traffic is equal to the incoming traffic multiplied by the scaling factor $\alpha(s,q,t)$.

Latency. End-to-end network latency for a traffic packet of a service request is the time it takes to the packet to be served by all VNFs along the path from the ingress to the egress VNFs. Such a latency includes two contributions, namely, the network delay between pairs of VMs on which subsequent VNFs are deployed and the processing time at the VNFs themselves. The former can be defined based on the delay of the logical links l, denoted by $D_{log}(l)$.

Such a delay is the sum of the delay of the underlying physical links:

$$D_{log}(l) = \sum_{e \in l} D_{phy}(e). \quad (21)$$

We also introduce binary variable $F(k, l, q_1, q_2, t)$ to represent whether logical link l is used for routing the traffic from VNF q_1 to q_2 of service request k at time t. F can be described as

$$\rho(k, l, q_1, q_2, t) \leq F(k, l, q_1, q_2, t), \quad \forall t \in \mathcal{T}, k \in \mathcal{K}, q_1, q_2 \in \mathcal{Q}, l \in \mathcal{L}. \quad (22)$$

The traffic packets in the VNFFG follow a path p of logical links in the underlying physical graph, which connect all VNFs in the VNFFG. Let $w \in \mathcal{W}_s$ be the sequence of VNFs, from an ingress VNF to an egress VNF in the VNFFG of service s. The network delay of traffic packets of service request k, which traverse the VNFs as specified by w and go through the links belonging to p, is given by:

$$\sum_{(q_1,q_2) \in w} \sum_{l \in p} F(k, l, q_1, q_2, t) \cdot D_{log}(l). \quad (23)$$

The processing time of VM m, denoted by $R(m, t)$, is the time it takes for a traffic packet to be completely processed in the VM. Modeling each VM as a queue with discipline PS (or, equivalently, FIFO), the processing time of VM m at time t is [2]:

$$R(m, t) = \frac{1}{\sum_{k \in \mathcal{K}} \sum_{q \in \mathcal{Q}} (\mu(k, m, q, t) - I(k, m, q, t))}, \quad m \in \mathcal{M}, t \in \mathcal{T}. \quad (24)$$

Then, the processing time incurred by the traffic packets following the VNF sequence w, is given by:

$$\sum_{q \in w} \sum_{m \in p} A(k, m, q, t) \cdot R(m, t). \quad (25)$$

Finally, the experienced delay must be less than the target delay, *i.e.*,

$$\sum_{(q_1,q_2) \in w} \sum_{l \in p} F(k, l, q_1, q_2, t) \cdot D_{log}(l) + \sum_{q \in w} \sum_{m \in p} A(k, m, q, t) \cdot R(m, t) \leq D_{QoS}(s), \quad \forall t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, w \in \mathcal{W}_s, p \in \mathcal{P}. \quad (26)$$

Our model supports the case where a sequence of VNFs in the VNFFG enforces a specific delay requirement. Let $\overline{w} \in \overline{\mathcal{W}}_s$ be a sequence of VNFs in the VNFFG of service s that requires additional delay requirement $\overline{D}_{QoS}(s, \overline{w})$. The experienced delay in $\overline{w}$ must be less than $\overline{D}_{QoS}(s, \overline{w})$, *i.e.*,

$$\sum_{(q_1,q_2) \in \overline{w}} \sum_{l \in p} F(k, l, q_1, q_2, t) \cdot D_{log}(l) + \sum_{q \in \overline{w}} \sum_{m \in p} A(k, m, q, t) \cdot R(m, t) \leq \overline{D}_{QoS}(s, \overline{w}), \quad \forall t \in \mathcal{T}, s \in \mathcal{S}, k \in \mathcal{K}_s, \overline{w} \in \overline{\mathcal{W}}_s, p \in \mathcal{P}. \quad (27)$$

Link capacity. The traffic on any physical link should not exceed the maximum link capacity, $B(e)$. To formalize this constraint, we define the auxiliary variable $L(e, t)$ to represent the traffic on physical link e at time t. This variable is equal to the total traffic between each pair of VNFs which goes through the logical link l containing the physical link e:

$$L(e, t) = \sum_{s \in \mathcal{S}} \sum_{k \in \mathcal{K}_s} \sum_{q_1,q_2 \in \mathcal{Q}} \sum_{\substack{l \in \mathcal{L}: \\ e \in l}} \Lambda(s, q_1, q_2, t) \cdot \rho(k, l, q_1, q_2, t). \quad (28)$$

The link capacity constraint is expressed as

$$L(e, t) \leq B(e), \quad \forall e \in \mathcal{E}, t \in \mathcal{T}. \quad (29)$$

Objective. The goal of the optimization problem is to maximize the service revenue while minimizing the total cost. Services may belong to different categories such as ultra-reliable low-latency communications (URLLC), enhanced mobile broadband (eMBB), or massive machine-type communications (mMTC) with different requirements for which the revenue should account. The revenue obtained by serving one unit of traffic of service s is indicated as $X_{rev}(s)$, which allows for the support of any arbitrary revenue scenario. For example, such a quantity can be inversely proportional to the target delay of service s, *i.e.*, $1/D_{QoS}(s)$, implying that serving services with lower target delay yield higher revenue for the MNO. The total revenue is expressed as

$$R = \sum_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} \sum_{k \in \mathcal{K}_s} X_{rev}(s) \cdot V(k, t) \cdot \lambda_{new}(s). \quad (30)$$

The total cost is the sum of the transmission cost in physical links, computational and idle costs in VMs, which are described, respectively, as:

$$C_{link} = \sum_{t \in \mathcal{T}} \sum_{e \in \mathcal{E}} X_{link}(e) \cdot L(e, t), \quad (31)$$

$$C_{cpu} = \sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}} \sum_{k \in \mathcal{K}} \sum_{q \in \mathcal{Q}} X_{cpu}(m) \cdot \mu(k, m, q, t) \cdot \omega(q), \quad (32)$$

$$C_{idle} = \sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}} X_{idle}(m) \cdot (U(m, t) + O(m, t)). \quad (33)$$

The above costs are expressed per unit of time and depend, respectively, on a proportional cost $X_{link}(e)$ paid

for each physical link e per unit of traffic, a proportional cost $X_{cpu}(m)$ for each VM m paid per unit of computation, and a fixed cost $X_{idle}(m)$ for each VM m paid if VM m is *turning-on* or *active*. Finally, we write the optimization formulation as:

$$\max\left[R - (C_{link} + C_{cpu} + C_{idle})\right]$$
$$\text{subject to (1)–(13), (17), (20), (26) and (29)} \qquad (34)$$

We note that the above optimization contains both continuous (*e.g.*, $\mu(k, m, q, t)$) and integer (*e.g.*, $A(k, m, q, t)$) variables with non-linear constraints (26) and (27), hence it is an MINLP.

3.3. Problem Complexity

The problem of jointly making decisions about VM activation, VNF placement, CPU assignment, and traffic routing formulated above contains both integer and real decision variables, hence it is non-convex. In the following, we prove that the problem is NP-hard, through a reduction from the weight constrained shortest path problem (WCSPP) to a simpler version of our own.

Theorem 1. *The problem mentioned in Sec. 3.1 is NP-hard when the objective value is greater than zero.*

Proof. We reduce an NP-hard problem, called weight constrained shortest path problem (WCSPP) [26], to our problem. Given a graph $G(V, E)$, and the cost and weight associated with the edges, the WCSPP asks to find the minimum cost route between two specified nodes while ensuring that the total weight is less than a given value. We consider a special case of our problem where only one service request with a chain of two VNFs arrives at $t = 1$ and departs in the next time step. We set the maximum number of instances for both VNFs to one. There are only two VMs in the physical infrastructure, with $C_{vm}(m) = \infty$ and $X_{cpu}(m) = X_{idle}(m) = 0$; the remaining are network nodes. We set $C_{dc}(d) = \infty, \forall d \in \mathcal{D}$. Then, it is easy to see that WCSPP is equivalent to the special case of our problem when the objective value is greater than zero.

Beside complexity, solving the problem formulated in Sec. 3.2 assumes that the entire knowledge of arrival and departure times of all service requests is available, which is not realistic in many scenarios. As detailed below, to cope with this issue, our strategy is to periodically solve our problem, with each problem instance leveraging only the information about the past and the current service requests.

4. The MaxSR Solution

In light of the problem complexity discussed above, we propose a heuristic solution called MaxSR, which makes decisions (i) only concerning a subsequent time interval encompassing the present and the near future, which can be predicted with high accuracy (as it will be discussed in Sec. 6), and (ii) based on the knowledge of the service requests occurring within such time interval. More precisely, starting from time step t, MaxSR makes decisions concerning the current service requests and accounting for a time horizon H, *i.e.*, extending till $t + H$. After τ time steps, where $\tau \leq H$, MaxSR is executed again accounting for the next time interval, *i.e.*, $[t + \tau, t + \tau + H)$. This implies that, although decisions are made accounting for a time horizon equal to H, they will be enacted just until the next execution of MaxSR. In other words, τ specifies the number of time steps between consecutive executions of the algorithm. At each execution, the algorithm computes the decisions, even for the current service requests allowing them to migrate to released cheaper resources. Note that a migration cost can be accounted for by incorporating this value into the cost of the resources towards which a migration could take place. Even with such a limited time horizon, directly solving the problem defined in Sec. 3.2 is still NP-hard. To walk around this limitation, at every execution, MaxSR processes the service requests received in the last τ time steps sequentially, *i.e.*, one request at a time. In the following, we provide an overview of MaxSR in Sec. 4.1, and we detail the algorithms composing our heuristic in Sec. 4.2.

4.1. Overview

At every execution, MaxSR first considers service requests in decreasing order based on the corresponding service revenue. It then activates the necessary VMs for serving the first service request, trying to map the VNF sequence w onto a path p connecting the VMs deemed to host the required VNFs. While doing this, more than one instance can be created for a VNF if necessary to meet the service target delay. To this end, we associate with each VNF a *delay budget*, which is proportional to the VNF computational complexity $\omega(q)$. Such budget, however, is *flexible*, since the delay contribution of a VNF exceeding its delay budget may be compensated for by a subsequent VNF on w, which is deployed in a VM able to process traffic faster than what indicated by the VNF budget. Additionally, MaxSR exploits a *backtrack* approach: in case of lack of sufficient resources at a certain point of current path p, the algorithm can go back to the last successfully deployed VNF and looks for an alternative deployment (hence path), leaving more spare budget for subsequent VNFs. None the less, it may prove impossible to find enough resources to accommodate the traffic and delay constraint of a given VNF instance; in this case, the service request is rejected.

The decisions that MaxSR makes are summarized below.

Placement. MaxSR aims to minimize the placement cost. This implies that the number of deployed VNF instances should be low, and the selected VMs should have a low cost. The algorithm thus starts from one instance and chooses the lowest-cost VM among the available ones.

If this placement is not feasible, it tries the highest capacity VM to avoid the use of an extra instance. If the latter strategy is also infeasible, it increases the number of instances and repeats the process until a successful deployment is possible, or the limit on the maximum number of instances is reached (Alg. 2 and Alg. 3).

Routing. Recall that each VNF may have several instances and that such instances may be deployed on VMs connected through multiple logical links. MaxSR adopts a water-filling approach to route the traffic between each pair of VNFs through different logical links between a pair of VMs. To limit the processing time at each VM, the traffic entering each VM is properly set based on the VM available capacity (Alg. 3).

CPU assignment. MaxSR aims to keep the service rate of the used VMs as low as possible, in order to reduce the consumption of computing resources, hence the cost. This means setting the lowest service rate compatible with the per-VNF delay budget, except when we have to compensate for a VNF exceeding its delay budget; in the latter case, the algorithm opts for the maximum service rate on the VM (Alg. 4).

Algorithm 1: Main body of MaxSR algorithm

Input: t, H, $\mathcal{K}_{t,H} \leftarrow \{k \in \mathcal{K} : [t, t+H) \cap \left[t_{arv}(k), t_{dpr}(k)\right) \neq \emptyset\}$
Output: result sets $\mathcal{R}_p := \{\mu(k,m,q)\}$, $\mathcal{R}_r := \{r(k,l,q_1,q_2)\}$, VM states

1 $\mathcal{R}_p \leftarrow \emptyset, \mathcal{R}_r \leftarrow \emptyset$
2 $R(k) \leftarrow X_{rev}(s) \cdot (\min\{t+H, t_{dpr}(k)\} - \max\{t, t_{arv}(k)\}) \cdot \lambda_{new}(s), \forall s \in \mathcal{S}, k \in \mathcal{K}_s \cap \mathcal{K}_{t,H}$
3 **sort** $k \in \mathcal{K}_{t,H}$ **by** $R(k)$ in desc. order
4 **forall** $k \in \mathcal{K}_{t,H}$ **do**
5 **call** BSRD(k) and update $\mathcal{R}_p$ and $\mathcal{R}_r$
6 VM-Activation($\mathcal{R}_p$)

4.2. Algorithms

Alg. 1. It is the main body of the MaxSR heuristic, taking as input time horizon H, the current time step t, and the set $\mathcal{K}_{t,H}$ of service requests which should be served in the time horizon $[t, t+H)$. Line 2 calculates service revenue $R(k)$ for each request k, based on the expected traffic to be served in the time horizon and the expected revenue, *i.e.*, $X_{rev}(s)$ for service s. The algorithm sorts the service requests in Line 3 in descending order, according to $R(k)$. It then calls BSRD for each request, in order to determine whether and how to serve it within the time horizon. If the request can be served, the resulting VNF placement/CPU assignment and routing decisions are stored in $\mathcal{R}_p$ and in $\mathcal{R}_r$, respectively. For each served request, $\mathcal{R}_p$ will then contain a tuple per each VNF instance that specifies the allocated VM and its assigned service rate, while $\mathcal{R}_r$ will contain a tuple for each pair of VNF instances, determining the amount of traffic on their connecting logical link(s). Finally, the VMs required for running the service request are activated if not already active; we recall that it takes one time step to activate them (turning-on state), and they will remain up till the service departure time.

Algorithm 2: Backtracking-based service request deployment (BSRD)

Input: service request k of service s
Output: $\mathcal{R}_p, \mathcal{R}_r$

1 $i \leftarrow 1$; *status* $\leftarrow$ `normal`; *can-backtrack* $\leftarrow$ `false`; $\mathcal{C} \leftarrow \emptyset$, $\mathcal{R}_p \leftarrow \emptyset$; $\mathcal{R}_r \leftarrow \emptyset$
2 $\Delta(s,q) \leftarrow \omega(q) / \sum_{j=1}^{|Q_s|} \omega(Q_s(j)), \forall q \in$ VNF chain of s
3 **while** $i \leq$ number of VNFs **do**
4 **if** *status* is `normal` **then**
5 **for** $n \leftarrow 1$ to $N(s, Q_s(i))$ **do**
6 **for** *strategy* $\in \{$`cheapest`, `largest`$\}$ **do**
7 **call** VPTR($k, i, n,$ *strategy*) and CA(k, i)
8 **if** deployment is successful **then**
 break
9 **else if** *status* is `critical` **then**
10 *can-backtrack* $\leftarrow$ `false`
11 **call** VPTR($k, i, N(s, Q_s(i))$, `largest`) and CA(k, i)
12 **if** i-th VNF is successfully deployed **then**
13 **if** *status* is `normal` **then** *can-backtrack* $\leftarrow$ `true`
14 Update $\mathcal{R}_p, \mathcal{R}_r$, *status* $\leftarrow$ `normal`, $i \leftarrow i+1$
15 **else** ▷ i-th VNF is not deployed
16 *status* $\leftarrow$ `critical`
17 **if** *can-backtrack* **then**
18 Discard $\mathcal{R}_p, \mathcal{R}_r$ for $(i-1)$-th VNF, $i \leftarrow i-1$
19 **else if** fail is due to delay budget **then**
20 Update $\mathcal{R}_p, \mathcal{R}_r$, $i \leftarrow i+1$
21 **else** ▷ fail is due to traffic
22 **terminate** and discard $\mathcal{R}_p$ and $\mathcal{R}_r$
23 **if** result sets are not feasible **then**
24 **terminate** and discard $\mathcal{R}_p$ and $\mathcal{R}_r$

Alg. 2. Given service request k for service s as an input, the goal of Alg. 2 is to check whether all VNFs of s can be deployed with the available resources. If it is possible, the request is served and the result sets $\mathcal{R}_p$ and $\mathcal{R}_r$ are returned. The global boolean variables *status* and *can-backtrack* represent the deployment status and the possibility of backtracking, respectively. *status* is `critical` if the last VNF deployment has failed, and `normal` otherwise. The global cache $\mathcal{C}$ is a set of results that facilitates the backtracking operation (see Alg. 3). The algorithm starts in `normal` mode; clearly, backtracking is not allowed

for the first VNF in the VNFFG and cache $\mathcal{C}$ is empty (Line 1). The algorithm starts by assigning a delay budget to each VNF of the service, which is proportional to the VNF computational complexity (Line 2), where $Q_s(j)$ denotes the j-th VNF in the VNFFG. Then, it goes across the sequence of VNFs starting from the ingress VNF and deploys them one by one.

For each VNF, Lines 4-11 decide on the number of required instances and the VM selection *strategy*, based on the deployment *status*. The strategy can be `cheapest` or `largest`: the algorithm selects VMs with the lowest cost when the strategy is `cheapest`, and with the highest capacity when the strategy is `largest`. The first part (Lines 5-8) deploys the VNF in the `normal` mode. Since the algorithm aims to keep the number of required VNF instances as low as possible, it starts with one instance and the `cheapest` strategy and calls VPTR to determine placement and routing, and CA to determine the CPU assignment. The deployment is successful if neither of these algorithms fails. If the `cheapest` strategy does not yield a successful deployment for the VNF, the algorithm keeps the number of instances fixed and tries the `largest` strategy. If both strategies fail, the number of instances is increased by one and the process is repeated. The algorithm ends whenever a successful deployment is found (Line 8), or the maximum number of instances is reached.

Lines 12-22 decide how to proceed in the VNF sequence according to the result of deployment, *status* and *can-bakctrack*. If the deployment is successful (Line 12), the algorithm updates the result set, sets *status* to `normal` and proceeds to the next VNF in the VNFFG (Line 14). *can-backtrack* is also updated in Line 13, which means that backtracking is allowed for the next VNFs only when we have a successful deployment in the `normal` mode for the current VNF: this prevents the algorithm to backtrack again to a VNF, which has already been deployed in `critical` mode. Otherwise (Line 15), *status* is set to `critical` and the algorithm proceeds as follows. As the first attempt, it tries to refine the placement in the previous step. Thus, if backtracking is allowed, it reverts the result sets related to the previous VNF in the VNFFG and goes back to deploy it again (Line 18). When the deployment fails but backtracking is not possible, due to a violation of the delay budget, the algorithm preserves the current deployment in the result set and proceeds to the next VNF, hoping to compensate for the exceeded delay budget (Line 20). If neither option is viable, the algorithm decides not to serve the current service request and reverts all result sets related to its deployment (Line 22).

Lines 10-11 deploy the VNF when status is `critical`, *i.e.*, when the previous VNF deployment has failed. This VNF is either the next VNF in the VNFFG when the algorithm is in the backtracking phase, or the previous VNF when the algorithm is going to compensate for the exceeded delay budget by the current deployment. In either case, the algorithm chooses the fastest option to deploy the VNF, regardless of the cost, using the maximum number of instances and `largest` strategy. Finally, the algorithm checks the feasibility of the decisions made with regard to the datacenter capacity and service target delay after each VNF deployment in Line 23. For the former, it is enough to check that the total computational capability assigned to VMs within each datacenter does not exceed its maximum capacity, *i.e.*, for each datacenter d,

$$\sum_{\mu(k,m,q)\in\mathcal{R}_p: m\in\mathcal{M}_d} \mu(k,m,q)\cdot\omega(q) \leq C_{dc}(d). \tag{35}$$

Traffic packets belonging to a service may go through different end-to-end paths in the physical network and experience different end-to-end delays. We define $\bar{\delta}(k,m,q)$ as the maximum end-to-end delay that traffic packets belonging to service request k experience from the ingress VNF until they depart VM m which hosts an instance of VNF q. Thus, after deploying VNF q of service request $k \in \mathcal{K}_s$, it is enough to check that this delay for any VM m, hosting an instance of q, does not exceed the service target delay:

$$\bar{\delta}(k,m,q) \leq D_{QoS}(s). \tag{36}$$

Alg. 3. It determines the placement and traffic routing for the i-th VNF of request k of service s, using n instances and the given *strategy*. Line 1 initializes (q_1, q_2) to the i-th VNFs pair in the VNFFG of service s, the routing result set $\mathcal{R}_r$ to $\emptyset$, and the remaining unserved traffic between q_1 and q_2, *i.e.*, Λ', to $\Lambda(k, q_1, q_2, t)$. The first pair of VNFs is $(\circ, q_1)$ with the assumption that the *dummy* VNF $\circ$ is placed on the *dummy* VM. In Lines 2-3, first the remaining capacity of each logical link l is calculated and stored in $B'_{log}(l)$ and then the ones that have a remaining capacity greater than zero, host VNF q_1 on their source VM, and host no VNF on their destination, are picked and stored in the set $\mathcal{L}_{top}$. The links in $\mathcal{L}_{top}$ and their destination VMs are the only potential candidates for this algorithm to place instances of the i-th VNF and accommodate its incoming traffic Λ'. In other words, in the rest of the algorithm, we consider the joint logical link and its destination VM as one entity and pick the best ones according to the *strategy* and n. If the selected entities cannot fit the incoming traffic, the placement fails; none the less, we still preserve the amount of satisfied traffic in the cache and exploit this information in the backtracking phase.

The implementation speed of the backtrack operation is greatly improved by *caching*. Specifically, when Alg. 3 is called in the backtracking phase to refine the placement of i-th VNF, the cache contains results which determine the routing of a portion of the outgoing traffic of the $(i+1)$-th VNF to the $(i+2)$-th VNF, which was satisfied by the previous deployment of the $(i+1)$-th VNF in the VNFFG. Lines 4-6 exploit the cached results and accommodate the unserved portion of incoming traffic by using different instances, which helps the next deployment of the $(i+1)$-th VNF to fully serve its traffic. For instance, assuming $\alpha(s, \mathcal{Q}_s(i+1)) = 1$ and that the placement of the $(i+1)$-th VNF has failed by Λ' unserved traffic, the backtrack-

Algorithm 3: VNF placement and traffic routing (VPTR)

Input: $k \in \mathcal{K}_s, i, n, strategy$

1 $(q_1, q_2) \leftarrow (Q_s(i-1), Q_s(i))$; $\mathcal{R}_r \leftarrow \emptyset$; $\Lambda' \leftarrow \Lambda(s, q_1, q_2, t)$;

2 $B'_{log}(l) \leftarrow$ remaining capacity of $l, \forall l \in \mathcal{L}$

3 $\mathcal{L}' \leftarrow \{l \in \mathcal{L} \cup \mathcal{L}_\circ :$ q_1 is on *src(l)* $\wedge$ *dst(l)* is free $\wedge\ B'_{log}(l) > 0\}$

4 **if** $\mathcal{C} \neq \emptyset$ **then** ▷ cache is not empty

5 Fill $l \in \mathcal{L}'$: *dst*$(l) = m$, consiedring limit $\frac{D(k,m,q)}{\alpha(s,q)}$, $\forall D(k, m, q) \in \mathcal{C} : q = q_2$

6 Update $\mathcal{R}_r, \Lambda', n, \mathcal{L}'$; $\mathcal{C} \leftarrow \emptyset$

7 **if** *strategy* is `cheapest` **then**

8 **sort** $l \in \mathcal{L}'$ **by** $\omega(q_2) \cdot X_{cpu}(dst(l)) + \sum_{e \in l} X_{link}(e)$ in asc. order

9 **else if** *strategy* is `largest` **then**

10 **sort** $l \in \mathcal{L}'$ **by** $\min\{B'_{log}(l), \frac{C_{vm}((dst(l))}{\omega(q_2)}\}$ in desc. order

11 $\mathcal{L}_{top} \leftarrow$ Pick top $l \in \mathcal{L}'$ as much as possible such that $|\{dst(l) : l \in \mathcal{L}_{top}\}| = n$

12 $\mathcal{M}_{top} \leftarrow \{dst(l) : l \in \mathcal{L}_{top}\}$

13 $\hat{I}(k, m, q_2) \leftarrow \frac{C_{vm}(m)}{\sum_{m' \in \mathcal{M}_{top}} C_{vm}(m')} \cdot \Lambda', \forall m \in \mathcal{M}_{top}$

14 $C'_{vm}(m) \leftarrow C_{vm}(m), \forall m \in \mathcal{M}_{top}$

15 **forall** $l \in \mathcal{L}_{top}$ **do**

16 $c(l) \leftarrow \min\{B'_{log}(l), \frac{C'_{vm}(dst(l))}{\omega(q_2)}\}$

17 $r(k, l, q_1, q_2) \leftarrow$ Fill l by remaining outgoing traffic of q_1 on *src(l)* considering $c(l)$ and limit $\hat{I}(k, dst(l), q_2)$

18 Update Λ', $B'_{log}(l)$, $C'_{vm}(dst(l))$; $\mathcal{R}_r \leftarrow \mathcal{R}_r \cup \{r(k, l, q_1, q_2)\}$

19 **if** $\Lambda' > 0$ **then**

20 Preserve $D(k, m, q_1)$ in cache $\mathcal{C}$ such that q_1 is on m

21 **return** `fail`, $\emptyset$

22 **return** `success`, $\mathcal{R}_r$

ing step will have to accommodate only Λ' traffic on extra VMs, *i.e.*, the routing and placement results for the served traffic portion, $D(k, m, q_2) \in \mathcal{C}$, will not change.

The pairs of logical links and the connected VMs will be selected for placement and routing based on the given *strategy*. If the *strategy* is `cheapest`, they will be sorted according to the cost of the logical link plus the VM CPU cost in ascending order (Line 8). If the strategy is `largest`, we sort them in descending order by the minimum of the remaining capacity of the logical link and the VM (Line 10). Line 11 picks the biggest set of top logical links such that the number of unique destination VMs is equal to the number of instances, *i.e.*, n, and stores them in $\mathcal{L}_{\text{top}}$. Note that there may be multiple logical links with the same destination VM in this set, and therefore we should pick the largest set to increase the chance of fitting the traffic. If the number of unique destination VMs is less than n, $\mathcal{L}_{\text{top}}$ will be empty and the placement fails. Otherwise, we store destination VMs corresponding to logical links $l \in \mathcal{L}_{\text{top}}$ in set $\mathcal{M}_{\text{top}}$ (Line 12).

To avoid an exceedingly high processing time, Line 13 introduces a limit for the amount of traffic entering a given VM $m \in \mathcal{M}_{\text{top}}$, proportional to the VM maximum computational capacity. Notice that all logical links ending at the same destination VM have the same limit. The remaining computational capacity of each selected VM, $C'_{vm}(m)$, is initialized to its maximum $C_{vm}(m)$ (Line 14). The algorithm adopts a *water-filling* approach to fill the logical links in Lines 15-18. First, for each logical link l and its connected VM *dst*(l), the remaining capacity, *i.e.*, the minimum of the remaining capacities of l and *dst(l)*, is stored in $c(l)$ (Line 16). Then, logical link l is filled by the remaining unserved outgoing traffic of VNF q_1 on VM src(l), so that neither $c(l)$ limit on the capacity of logical link l nor the $\hat{I}(k, m, q_2)$ limit on the incoming traffic of VM dst(l) are violated. Line 18 updates the remaining unserved traffic from q_1 to q_2 (Λ'), the remaining capacity of logical link l ($B'_{\text{log}}(l)$), the remaining capacity of destination VM ($C'_{vm}(dst(l))$), and routing result set ($\mathcal{R}_r$). Finally, if there is still some unserved traffic from VNF q_1 to q_2 (*i.e.*, not all the traffic can be served), the algorithm returns `fail` (Lines 19-21). Line 20 preserves the satisfied outgoing traffic of VM m hosting an instance of VNF q_1, *i.e.*, $D(k, m, q_1)$, in the cache, so as to use it later on in case of backtracking. Otherwise, the algorithm returns `success` with the placement result set $\mathcal{R}_p$.

Alg. 4. It is called in Line 7 and Line 11 of Alg. 2 when the deployment of VNF q in Alg. 3 is successful. Given the result set $\mathcal{R}_r$, this algorithm is responsible for assigning the service rates to VMs for running the deployed instances of VNF q. After initialization, in Line 2, $\mathcal{L}_{dep}$ defines the set of the logical links used for routing a part of traffic from any instance of VNF q_1 to any instance of VNF q_2. We store the VMs on which VNF q_2 is already deployed in the set $\mathcal{M}_{dep}$ (Line 3). Then, for each $m \in \mathcal{M}_{dep}$, we calculate the incoming traffic through the sum of traffic from all logical links ending in VM m, and store it in $I(k, m, q_2)$ in Line 5.

$\check{\delta}(k, m, q_2)$ represents the maximum end-to-end delay that traffic packets experience from the ingress VM to VM m, which hosts an instance of VNF q_2, but before being processed by m. For each logical link $l \in \mathcal{L}_{dep}$ where *dst(l)* $= m$, this delay is equal to the sum of the maximum end-to-end delay of traffic packets after being processed by VNF q_1 on VM *src*(l), *i.e.*, $\bar{\delta}(q_1, src(l))$, and the delay of logical link l, *i.e.*, $D_{log}(l)$. Taking the maximum over all such logical links, we have $\check{\delta}(k, m, q_2)$ in Line 6.

Similar to the VNF deployment in Alg. 3, the algorithm assigns service rates to VMs based on the deployment *status*. In the `critical` mode, the algorithm aims to reduce the delay contribution, which depends on logical links delay and processing time on VMs. The logical

Algorithm 4: CPU assignment (CA)

Input: $k \in \mathcal{K}_s, i, \mathcal{R}_r$

1 $(q_1, q_2) \leftarrow (Q_s(i-1), Q_s(i))$; $\mathcal{R}_p \leftarrow \emptyset$;
2 $\mathcal{L}_{dep} \leftarrow \{l \in \mathcal{L} : \exists r(k', q_1', q_2', l) \in \mathcal{R}_r : k' = k \wedge q_1' = q_1 \wedge q_2' = q_2 \wedge r(k', q_1', q_2', l) > 0\}$
3 $\mathcal{M}_{dep} \leftarrow \{m \in \mathcal{M} : \exists l \in \mathcal{L}_{dep} : dst(l) = m\}$
4 **for** $m \in \mathcal{M}_{dep}$ **do**
5 $\quad I(k, m, q_2) \leftarrow \sum_{r(k,l,q_1,q_2) \in \mathcal{R}_r : dst(l) = m} r(k, l, q_1, q_2)$
6 $\quad \check{\delta}(k, m, q_2) \leftarrow \max_{l \in \mathcal{L}_{dep} : dst(l) = m} \left(\bar{\delta}(k, src(l), q_1) + D_{log}(l)\right)$
7 $\quad$ **if** *status* is `critical` **then**
8 $\quad\quad \mu(k, m, q_2) \leftarrow \frac{C_{vm}(m)}{\omega(q_2)}$
9 $\quad$ **else** ▷ *status* is `normal`
10 $\quad\quad \mu(k, m, q_2) \leftarrow I(k, m, q_2) + \frac{1}{\sum_{j=1}^{i} \Delta(s, Q_s(j)) - \check{\delta}(k, m, q_2)}$
11 $\quad\quad$ **if** $\mu(k, m, q_2) \notin (I(k, m, q_2), \frac{C_{vm}(m)}{\omega(q_2)}]$ **then**
12 $\quad\quad\quad \mu(k, m, q_2) \leftarrow \frac{C_{vm}(m)}{\omega(q_2)}$
13 $\quad \mathcal{R}_p \leftarrow \mathcal{R}_p \cup \{\mu(k, m, q_2)\}$
14 $\quad \bar{\delta}(k, m, q_2) \leftarrow \check{\delta}(k, m, q_2) + \frac{1}{\mu(k, m, q_2) - I(k, m, q_2)}$
15 **if** $\max_{m \in \mathcal{M}_{dep}} \bar{\delta}(k, m, q_2) > \sum_{j=1}^{i} \Delta(s, Q_s(j))$ **then**
16 $\quad$ **return** `fail`, $\mathcal{R}_p$
17 **return** `success`, $\mathcal{R}_p$

links are already selected by the VPTR algorithm, thus here we assign the maximum possible service rate for the VM to reduce the processing time (Line 8). Instead, when the algorithm is in `normal` mode, it chooses the minimum possible service rates for VM m (Line 10), such that the VNFs delay budget do not violate, *i.e.*

$$\sum_{j=1}^{i} \Delta(s, Q_s(j)) - \check{\delta}(k, m, q_2) = \frac{1}{\mu(k, m, q_2) - I(k, m, q_2)}. \tag{37}$$

In the above equation, the right- and left-hand sides represent the processing time of VM m and the remaining delay budget of VNFs, respectively. To compute the latter, first it is calculated the total delay budget of the VNFs up to the i-th one (*i.e.*, the current one). Then, it is subtracted by the maximum end-to-end delay of traffic packets, before being processed by VNF q_2 on VM m, *i.e.*, $\check{\delta}(k, m, q_2)$.

The computed service rate for VM m may be invalid because (i) no delay budget is left to process the current VNF on VM m, *i.e.*, the left-hand side of equality in (37) becomes non-positive, or (ii) the assigned service rate exceeds the maximum capability of the VM. In both cases, the CA algorithm fails, however, the VM is assigned to its maximum computational capability to process the VNF (Line 12). Recall that, although the CPU assignment failed for the current VNF, the algorithm keeps the results to be used in Alg. 2 (Line 19) when backtracking is not allowed. In this case, the algorithm continues with the next VNF and tries to compensate for the exceeded delay budget. Line 13 stores the results, and Line 14 updates $\bar{\delta}(k, m, q_2)$ for this VM that shows the maximum end-to-end delay after the packets are processed by VM m. Finally, when all service rates have been assigned, the algorithm returns `fail` if the remaining delay budget is violated for at least one VM (Line 15), and `success` otherwise.

4.3. Computational Complexity

The MaxSR heuristic takes the set of physical links $\mathcal{E}$, service requests $\mathcal{K}$, and their VNFFG $\mathcal{Q}_s$, VMs $\mathcal{M}$, and logical links $\mathcal{L}$ as inputs. Note that $\mathcal{L}$ is considered as an input since it is computed once for all executions of MaxSR algorithm. Below, we prove that this algorithm has a worst-case polynomial complexity in terms of input parameters.

Theorem 2. *The MaxSR algorithm has a worst-case polynomial computation complexity.*

Proof. First, we determine the complexity of the VPTR and CA algorithms. VPTR constructs and sorts the set $\mathcal{L}'$ in $O(|\mathcal{L}| \log |\mathcal{L}|)$ and adopts water filling to fill the logical links in $O(\mathcal{L})$, thus the total time complexity of this algorithm is $O(|\mathcal{L}| \log |\mathcal{L}|)$. CA also has $O(\mathcal{L})$ complexity, hence the total computational complexity of VPTR and CA remains equal to that of VPTR. Alg. 1 sorts the service requests in $O(|\mathcal{K}| \log |\mathcal{K}|)$ and calls BSRD for each service request. In the worst-case, BSRD tries every possible number of instances and strategies for all VNFs in the VNFFG of the given service request. Let N and Q be upper bounds on the maximum number of instances, *i.e.* $N(s, q), \forall s \in \mathcal{S}, q \in \mathcal{Q}$, and the number of VNFs in a VNFFG, *i.e.* $|\mathcal{Q}_s|, \forall s \in \mathcal{S}$, respectively. Thus, the total time complexity of BSRD is $O(NQ|\mathcal{L}| \log |\mathcal{L}|)$ and total time complexity of Alg. 1 is $O\big(|\mathcal{K}|NQ|\mathcal{L}| \log |\mathcal{L}| + |\mathcal{K}| \log |\mathcal{K}|\big)$. Therefore, the worst-case total time complexity is polynomial in terms of input parameters. In other words, the complexity of the heuristic depends primarily on the number of service requests, the number of VNFs in each VNFFG, the number of deployment attempts for each VNF, and the number of logical links.

5. Numerical Results

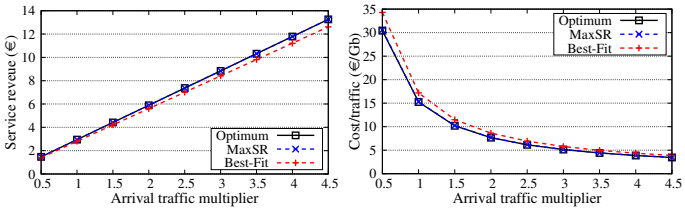
We now present the results of the numerical experiments we conducted, and show that the proposed scheme consistently outperforms a state-of-the-art approach and closely matches the optimum. We take the well-known Best-Fit approach as a baseline, as this is followed by many state-of-the-art works (*e.g.*, [5, 8, 15, 18]). We compare our heuristic algorithm against the following benchmarks:

Table 3: List of services

Service	D_{QoS} (ms)	λ_{new} (Mb/s)	X_{rev} (€/Gb)	Application
s_1	10	3	100	safety apps. (*e.g.*, vehicle collision detection)
s_2	45	10	22.2	real-time apps (*e.g.*, gaming)
s_3	80	15	12.5	soft real-time apps
s_4	2500	400	0.4	delay-tolerant apps (*e.g.*, video streaming)

Table 4: Different VM types in datacenters

VM type	C_{vm} (MIPS)	X_{cpu} (€/MIPS/hour)	X_{idle} (€/hour)
Small	600	2×10^{-5}	0.018
Medium	1200	4×10^{-5}	0.036
Large	1800	6×10^{-5}	0.054



(a) (b)

Figure 2: Small-scale scenario. Impact of service requests arrival traffic on absolute value of service revenue and cost/traffic ratio. Physical link delay = 2 ms.

- **Global optimum**. The solution of the optimization problem defined in Sec. 3.2 obtained by brute-force search, assuming exact knowledge of arrival and departure times of all service requests.
- **Best-fit**. It is an online algorithm which decides about each service request upon its arrival, without any information about the future service requests. Best-Fit deploys VNFs of a service request one by one, using a single instance of each VNF and the `cheapest` strategy. If the request can be served, the selected resources will be dedicated to the service request until its departure.

In our performance evaluation, we use the following performance metrics:

- **Service revenue**, defined as the sum of revenues achieved by serving service requests. For a single request of service s, this metric equals the amount of served traffic multiplied by $X_{rev}(s)$.
- **Cost/traffic**, which reflects the average cost incurred to serve a unit of traffic.

In the following, we first consider a small-scale network scenario, for which the optimum solution can be obtained in a reasonable time. This scenario will give interesting and easy-to-interpret insights regarding how each service type impacts the revenue and cost/traffic ratio. Then, we run MaxSR and Best-Fit in a large-scale real network scenario, where achieving the optimum solution is impractical. Table 3 summarizes the services we consider for our performance evaluation, inspired to real-world 5G applications. The revenue gained from serving one unit of traffic of service s, *i.e.*, $X_{rev}(s)$, is set inversely proportional to the service target delays. We assume that the service requests arrive according to a Poisson process, and the duration of requests follows an exponential distribution.

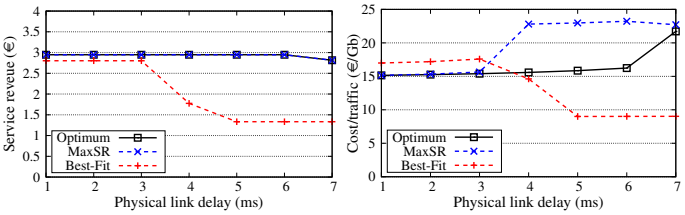
In both scenarios, we study the impact of traffic and delay on the performance metrics by multiplying traffic arrival rates λ_{new} and physical link delays $D_{phy}(e)$ by different factors. We run each experiment 50 times and report the average value for each point in the figures. In general, MaxSR, taking advantage of backtracking, achieves close to the optimum service revenue better than Best-Fit. However, the value of cost/traffic ratio depends on how tight the target delay is. When the target delay is small, the chance of backtracking increases; therefore, MaxSR incurs more cost to serve the requests.

5.1. Small-scale Scenario

We consider two pairs of VMs of different types, *i.e.*, *small* and *medium* as described in Table 4. Pairs of VMs inside are connected using a physical link: physical links between *small* and *medium* types VMs have cost of 0.02 €/Gb and 0.04 €/Gb per hour, respectively, while their latency varies from 1 ms to 7 ms with the default value set to 2 ms, and we disregard the link capacity. The time needed to setup a VM is one minute.

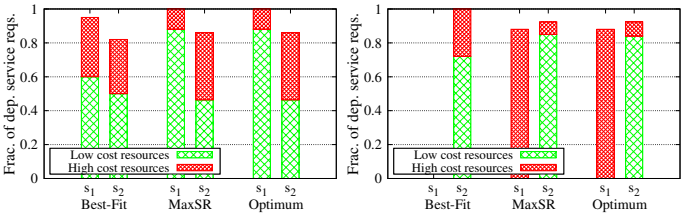
We consider two simple services s_1 and s_2, each having a chain of two VNFs with target delays 10 ms and 45 ms, and with input traffic rates 3 Mb/s and 15 Mb/s, respectively (as summarized in Table 3). In this scenario, we set $N(s, q) = 1$ for all VNFs, an average duration of 3 minutes for each service, and we assign them randomly to the arrival points of a Poisson process with an average rate of 0.5 requests per minute, while the total system lifespan is set to 10 minutes.

Impact of Physical Link latency and Arrival Traffic. Figure 2 shows the impact of the traffic arrival intensity on the service revenue and cost/traffic ratio. MaxSR matches the optimum, and Best-Fit performs close to the optimum in both service revenue and cost/traffic ratio. As it has no backtracking mechanism, Best-Fit does not serve a request whenever any of its VNFs cannot be served within its delay budget, *i.e.*, it has no budget flexibility;



(a) (b)

Figure 3: Small-scale scenario. Impact of physical link latecy on absolute value of service revenue and cost/traffic ratio. Arrival traffic multiplier = 1.



(a) Physical link delay = 3 ms. (b) Phsysical link delay = 7 ms.

Figure 4: Fraction of deployed service requests for each service and algorithm. Arrival traffic multiplier = 1.

therefore, it achieves lower service revenue than the optimum. While the cost of physical links increases proportionally to the traffic, the costs of VMs in *turning-on* mode remains constant, and their cost in *active* mode increases less than proportionally with the traffic; the resulting effect is that cost/traffic ratio decreases with the traffic – which conforms to the intuitive notion that serving larger amounts of traffic is more cost-efficient. Best-Fit incurs more cost compared to MaxSR and optimum because it does not support VNF migration, causing a VNF to continue running on a high-cost VM even if a low-cost VM becomes available. The excess VMs CPU cost and transmission cost scale with the traffic, whereas the excess VMs idle cost remains constant; therefore, the difference between Best-Fit and optimum becomes smaller as traffic increases.

Figure 3 shows the impact of physical link latency on the service revenue and cost/traffic ratio. For all latency values, MaxSR is still able to achieve optimum service revenue. As shown in Figure 4a, no strategy (not even optimum) can serve all requests when the physical link delay is 3 ms especially, for service s_2. The reason is that when the number of concurrent requests becomes more than two, both optimum and MaxSR give the priority to the high-revenue service s_1 and requests for s_2 will only be processed if resources are available. When the physical link delay increases, service requests need more computational capacity on VMs to meet their target delay, in order to offset longer network delays. Specifically, when the physical link delay is 7 ms, requests of type s_1 can only be served on high-capacity VMs, and therefore, concurrent requests

Figure 5: Cogent Network Topology.

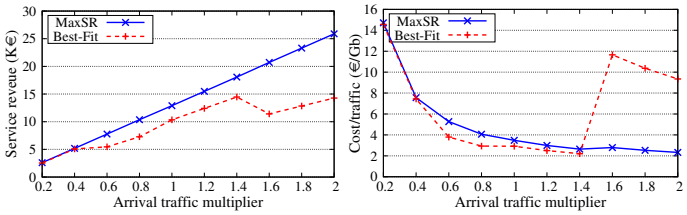
of this type can not be served. This is confirmed by the degradation of the optimum in Figure 3a, and in Figure 4b, where the fraction of served requests of type s_1 becomes less than 1 when the physical link delay is 7 ms.

Best-Fit gains substantially lower service revenue compared to others, especially for higher values of physical link delay. As shown in Figure 4b, this is due to the fact that Best-Fit cannot deploy requests of type s_1 in those cases. This, in turn, is due to the fact that it does not support backtracking: when the delay budget for the second VNF in the chain of s_1 is violated, no corrective action is taken and the whole request fails.

Figure 3b shows MaxSR has a higher cost/traffic ratio when the physical link delay is over 4 ms. The reason is that the need for backtracking increases with the physical link delay, and VMs become more likely to be scaled to their maximum capacity, which results in a higher cost. As one might expect, the cost/traffic ratio for Best-Fit decreases when physical link delay ≥ 4 ms because it does not serve requests of higher cost service s_1. Recall that the cost of a service depends on the amount of required CPU on VMs, and therefore services with lower target delays incur more costs to serve one unit of traffic.

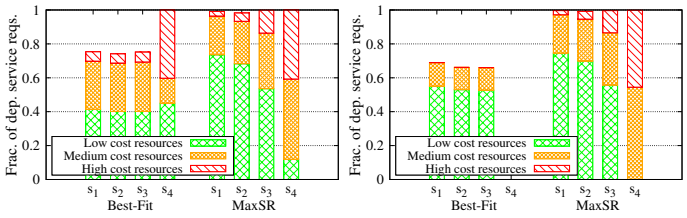
Large-scale scenario. We consider the real-world inter-datacenter network *Cogent*, a tier 1 Internet service provider (Figure 5). This network topology contains 197 access nodes with 245 physical links and 32 datacenters. We set the cost of links connecting the datacenters to 0.02 €/GB. The delay of logical links connecting the datacenters is set to be proportional to their geographical lengths, while the links inside each datacenter are assumed to be ideal having no capacity limit, latency, and cost. We assume each datacenter hosts 42 VMs, each of which is connected to some edge switches. We categorize VMs within each datacenter in *small*, *medium*, and *large* types according to their capacity and cost, as described in Table 4. We assume VMs need one minute to setup before being *active*.

We consider the four different services described in Table 3, each of which is a representative of a category of real 5G applications. In this scenario, we assume that the VNFFG of each service is a chain of five VNFs. We further assume that the computational complexity, *i.e.*, ω and maximum number of instances, *i.e.*, $N(s,q)$ for two VNFs of service s_4 are three, while other VNFs have $\omega = 1$ and $N(s,q) = 1$. Similar to the previous scenario, we consider



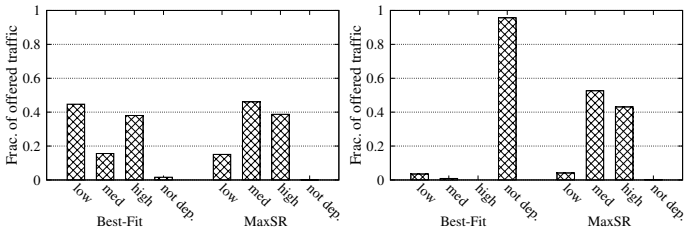
(a) (b)

Figure 6: Large-scale scenario. Impact of service requests arrival traffic on absolute value of service revenue and cost/traffic ratio. Physical link delay multiplier = 1.



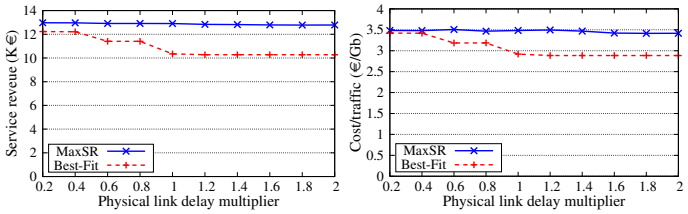
(a) Traffic multiplier = 1.0 (b) Traffic multiplier = 1.6

Figure 7: Fraction of deployed service requests for each service and algorithm. Physical link delay multiplier = 1.



(a) Traffic multiplier = 1.0 (b) Traffic multiplier = 1.6

Figure 8: Fraction of offered traffic deployed on each resource type for each algorithm. *low*, *med*, *high* and *not dep.* mean the fraction of offered traffic served on low-cost resources, medium-cost resources, high-cost resources, and which is not served, respectively. Physical link delay multiplier = 1.



(a) (b)

Figure 9: Large-scale scenario. Impact of physical link delay on absolute value of service revenue and cost/traffic ratio. Arrival traffic multiplier = 1.

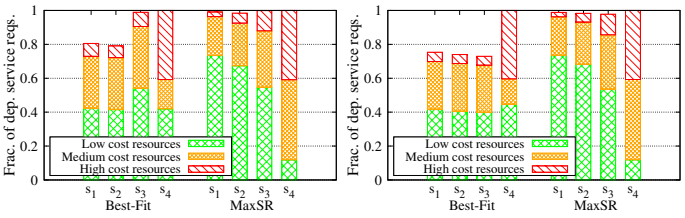
an equal number of requests for each service where requests arrive across time steps with the average inter-arrival time of three minutes and end after an average duration of two hours, and the total system lifespan is assumed to be one day. In this experiment, we set H to 40 minutes and τ to 20 minutes.

Impact of Physical Link latency and Arrival Traffic. As explained above, the optimum values cannot be obtained for this scenario in a reasonable time and therefore we rely on results for MaxSR and Best-Fit. Figure 6a shows the effect of arrival traffic on the service revenue, while Figure 7 shows the fraction of requests of each service that can be successfully deployed. We observe that service revenue for MaxSR changes almost proportionally with the traffic because increasing the traffic almost does not impact the fraction of served requests by this algorithm. Best-Fit serves a lower fraction of service requests, and therefore achieves lower revenue. Besides, Best-Fit shows a drop-off in service revenue when the arrival traffic multiplier is 1.6: as confirmed by Figure 7b, this is because Best-Fit does not serve requests of high traffic service s_4 when the traffic multiplier is over 1.6, due to its lack of support for multiple VNF instances.

Figure 6b shows the impact of arrival traffic on the cost/traffic ratio. Best-Fit has lower cost/traffic ratio when arrival traffic multiplier is less than 1.4, since MaxSR must use resources with higher cost and higher computational capabilities to serve more service requests; in other words, Best-Fit serves less traffic but that traffic is served cheaply. Similar to service revenue the values of cost/traffic ratio for Best-Fit have a significant rise when the arrival traffic multiplier is 1.6, as confirmed by Figure 8, when the traffic multiplier increases from 1.0 to 1.6, Best-Fit is no longer able to serve a significant fraction of the total traffic. As shown by Figure 7, the traffic Best-Fit is unable to serve mainly belongs to the low-cost service s_4, which results in a higher cost for served traffic.

Figure 9a shows the impact of physical link latency on the service revenue. Similar to the small-scale scenario, MaxSR outperforms Best-Fit especially for higher values of physical link delays, because the latter cannot serve the requests for services with low target delay (hence, higher revenue). As these service types have a higher cost, they will also cause the cost/traffic ratio for Best-Fit to be lower than MaxSR as shown in Figure 9b.

Running Time. We run our experiments using a server with 40-core Intel Xeon E5-2690 v2 3.00GHz CPU and 64 GB of memory. To compare the running time of different algorithms, we consider the case where the arrival traffic and physical link delay multipliers are equal to one. For each scenario, we run the algorithm 50 times and report the average running time in Table 5. MaxSR and Best-Fit are substantially faster than brute-force in the small-scale scenario. The prohibitively long running time for brute-force highlights its poor scalability, and makes it inapplicable for the large-scale scenario in practice. The results for the large-scale scenario show that although MaxSR has a higher running time compared to Best-Fit due to backtracking, both of them are scalable



(a) Physical link delay multiplier = 0.6 (b) Physical link delay multiplier = 1.2

Figure 10: Fraction of deployed service requests for each service and algorithm. Arrival traffic multiplier = 1.

Table 5: Running time (in seconds)

Scenario	Brute-force	MaxSR	Best-Fit
Small-scale	399	0.2	0.14
Large-scale	-	21	2

and adequately fast for large-scale networks.

6. Discussion

In this section, we discuss the applicability and scalability of MaxSR algorithm.

Applicability. The MaxSR solution is based on the assumption that the knowledge of upcoming service requests is available for the near future. However, in some practical scenarios, the arrival time and service type of requests are unknown beforehand. Thus, we need to forecast them based on previous network statistics. A broad spectrum of related work on traffic prediction [27–36] confirm that the high-quality prediction of future traffic exists. Especially, by taking advantage of deep learning [29–36], predictions are evolving to be more accurate and reliable, and further improvements of prediction techniques are currently under study. In particular, He *et al.* [31] propose an efficient deep learning-based solution to predict the network traffic at the user level over a short horizon, which dynamically adapts to highly varying data traffic and performs very close to the oracle case. Thus, upcoming service requests in a short time horizon can be reasonably well estimated in advance.

Scalability. As shown in Sec. 4.3, MaxSR algorithm has a worst-case polynomial computation complexity, and its scalability is highlighted by the running times reported in the previous section. It is important to highlight how such a scalability is achieved without resorting to distributed solutions (*e.g.*, [8]), which represent an alternative strategy to achieve scalability and short runtimes, at the cost of potentially conflicting decisions by different decision-makers.

7. Conclusion

We proposed a dynamic service deployment strategy in 5G networks, accounting for real-world aspects such as VM setup times, and jointly making all the required decisions. We first formulated the problem of joint requests admission, VM activation, VNF placement, resource allocation, and traffic routing as an MINLP based on the complete knowledge of requests arrival and departure times. We took the MNO profit as the main objective to be optimized over the entire system lifespan, leveraging a queueing model to ensure all requests adhere to their latency targets. Our model also accounted for the key features of 5G services such as complex VNF graphs and arbitrary input traffic.

Due to the problem complexity, we further proposed a heuristic, MaxSR, which has polynomial complexity and attains near-optimal solutions, while only needing the knowledge/prediction of the upcoming service requests in a short time horizon. The algorithm works in a sliding-horizon fashion, rearranging the current-served requests across existing VMs to reduce the deployment costs, and admitting the new ones as they arrive at the system. Furthermore, the parameters of MaxSR allow for different trade-offs between solution optimality and running time. We demonstrated the effectiveness and efficiency of our approach through a numerical evaluation including different network scenarios.

Acknowledgments

This work was partially supported by the EU 5GROWTH project (Grant No. 856709).

References

[1] ETSI, Network Functions Virtualisation (NFV); Management and Orchestration (2017).

[2] S. Agarwal, F. Malandrino, C. F. Chiasserini, S. De, Vnf placement and resource allocation for the support of vertical services in 5g networks, IEEE/ACM Transactions on Networking 27 (1) (2019) 433–446.

[3] B. Yi, X. Wang, K. Li, M. Huang, et al., A comprehensive survey of network function virtualization, Computer Networks 133 (2018) 212–262.

[4] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, Near optimal placement of virtual network functions, in: IEEE Conference on Computer Communications (INFOCOM), 2015, pp. 1346–1354.

[5] M. Xia, M. Shirazipour, Y. Zhang, H. Green, A. Takacs, Network function placement for nfv chaining in packet/optical datacenters, Journal of Lightwave Technology 33 (8) (2015) 1565–1570.

[6] Lin Gu, Sheng Tao, Deze Zeng, Hai Jin, Communication cost efficient virtualized network function placement for big data processing, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2016, pp. 604–609.

[7] M. Mechtri, C. Ghribi, D. Zeghlache, A scalable algorithm for the placement of service function chains, IEEE Transactions on Network and Service Management 13 (3) (2016) 533–546.

[8] C. Pham, N. H. Tran, S. Ren, W. Saad, C. S. Hong, Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach, IEEE Transactions on Services Computing 13 (1) (2020) 172–185.

[9] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, B. Akbari, Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining, IEEE Transactions on Network and Service Management 16 (1) (2019) 374–388.
[10] J. Pei, P. Hong, K. Xue, D. Li, Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system, IEEE Transactions on Parallel and Distributed Systems 30 (10) (2019) 2179–2192.
[11] G. Sallam, B. Ji, Joint placement and allocation of virtual network functions with budget and capacity constraints, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 523–531.
[12] M. A. Tahmasbi Nejad, S. Parsaeefard, M. A. Maddah-Ali, T. Mahmoodi, B. H. Khalaj, vspace: Vnf simultaneous placement, admission control and embedding, IEEE Journal on Selected Areas in Communications 36 (3) (2018) 542–557.
[13] R. Zhou, Z. Li, C. Wu, An efficient online placement scheme for cloud container clusters, IEEE Journal on Selected Areas in Communications 37 (5) (2019) 1046–1058.
[14] T. Kuo, B. Liou, K. C. Lin, M. Tsai, Deploying chains of virtual network functions: On the relation between link and server usage, IEEE/ACM Transactions on Networking 26 (4) (2018) 1562–1576.
[15] X. Fei, F. Liu, H. Xu, H. Jin, Adaptive vnf scaling and flow routing with proactive demand prediction, in: IEEE Conference on Computer Communications (INFOCOM), 2018, pp. 486–494.
[16] H. Tang, D. Zhou, D. Chen, Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers, IEEE Transactions on Parallel and Distributed Systems 30 (3) (2019) 530–543.
[17] Y. Jia, C. Wu, Z. Li, F. Le, A. Liu, Online scaling of nfv service chains across geo-distributed datacenters, IEEE/ACM Transactions on Networking 26 (2) (2018) 699–710.
[18] Y. Li, L. T. X. Phan, B. T. Loo, Network functions virtualization with soft real-time guarantees, in: IEEE Conference on Computer Communications (INFOCOM), IEEE, 2016, pp. 1–9.
[19] V. Eramo, E. Miucci, M. Ammar, F. G. Lavacca, An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures, IEEE/ACM Transactions on Networking 25 (4) (2017) 2008–2025.
[20] J. Liu, W. Lu, F. Zhou, P. Lu, Z. Zhu, On dynamic service function chain deployment and readjustment, IEEE Transactions on Network and Service Management 14 (3) (2017) 543–553.
[21] M. Huang, W. Liang, Y. Ma, S. Guo, Maximizing throughput of delay-sensitive nfv-enabled request admissions via virtualized network function placement, IEEE Transactions on Cloud Computing (2019) 1–1.
[22] F. Malandrino, C. F. Chiasserini, G. Einziger, G. Scalosub, Reducing service deployment cost through vnf sharing, IEEE/ACM Transactions on Networking 27 (6) (2019) 2363–2376.
[23] F. Malandrino, C. F. Chiasserini, C. Casetti, G. Landi, M. Capitani, An optimization-enhanced mano for energy-efficient 5g networks, IEEE/ACM Transactions on Networking 27 (4) (2019) 1756–1769.
[24] Q. Zhang, F. Liu, C. Zeng, Adaptive interference-aware vnf placement for service-customized 5g network slices, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 2449–2457.
[25] N. Alliance, Description of network slicing concept, NGMN 5G P 1 (2016) 1.
[26] I. Dumitrescu, N. Boland, Algorithms for the weight constrained shortest path problem, International Transactions in Operational Research 8 (1) (2001) 15–29.
[27] K. Papagiannaki, N. Taft, Zhi-Li Zhang, C. Diot, Long-term forecasting of internet backbone traffic, IEEE Transactions on Neural Networks 16 (5) (2005) 1110–1124.
[28] F. Xu, Y. Lin, J. Huang, D. Wu, H. Shi, J. Song, Y. Li, Big data driven mobile traffic understanding and forecasting: A time series approach, IEEE Transactions on Services Computing 9 (5) (2016) 796–805.
[29] Y. Xu, F. Yin, W. Xu, J. Lin, S. Cui, Wireless traffic prediction with scalable gaussian process: Framework, algorithms, and verification, IEEE Journal on Selected Areas in Communications 37 (6) (2019) 1291–1306.
[30] J. Feng, X. Chen, R. Gao, M. Zeng, Y. Li, Deeptp: An end-to-end neural network for mobile cellular traffic prediction, IEEE Network 32 (6) (2018) 108–115.
[31] Q. He, A. Moayyedi, G. Dán, G. P. Koudouridis, P. Tengkvist, A meta-learning scheme for adaptive short-term network traffic prediction, IEEE Journal on Selected Areas in Communications 38 (10) (2020) 2271–2283.
[32] K. Gao, D. Li, L. Chen, J. Geng, F. Gui, Y. Cheng, Y. Gu, Predicting traffic demand matrix by considering inter-flow correlations, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2020, pp. 165–170.
[33] A. Rago, G. Piro, G. Boggia, P. Dini, Multi-task learning at the mobile edge: An effective way to combine traffic classification and prediction, IEEE Transactions on Vehicular Technology 69 (9) (2020) 10362–10374.
[34] A. Azari, P. Papapetrou, S. Denic, G. Peters, User traffic prediction for proactive resource management: Learning-powered approaches, in: 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–6.
[35] Q. Zeng, Q. Sun, G. Chen, H. Duan, C. Li, G. Song, Traffic prediction of wireless cellular networks based on deep transfer learning and cross-domain data, IEEE Access 8 (2020) 172387–172397.
[36] M. Li, Y. Wang, Z. Wang, H. Zheng, A deep learning method based on an attention mechanism for wireless network traffic prediction, Ad Hoc Networks 107.