VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks

VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks

Satyam Agarwal, *Member, IEEE,* Francesco Malandrino, *Member, IEEE,*
Carla-Fabiana Chiasserini, *Fellow, IEEE,* Swades De, *Senior Member, IEEE*

Abstract—One of the main goals of 5G networks is to support the technological and business needs of various industries (the so-called verticals), which wish to offer to their customers a wide range of services characterized by diverse performance requirements. In this context, a critical challenge lies in mapping in an automated manner the requirements of verticals into decisions concerning the network infrastructure, including VNF placement, resource assignment, and traffic routing. In this paper, we seek to make such decisions *jointly*, accounting for their mutual interaction, and efficiently. To this end, we formulate a queuing-based model and use it at the network orchestrator to optimally match the vertical's requirements to the available system resources. We then propose a fast and efficient solution strategy, called MaxZ, which allows us to reduce the solution complexity. Our performance evaluation, carried out accounting for multiple scenarios representative of real-world services, shows that MaxZ performs substantially better than state-of-the-art alternatives and consistently close to the optimum.

1 Introduction

5G networks are envisioned to provide the computational, memory, and storage resources needed to run multiple third parties (referred to as vertical industries or *verticals*) with diverse communication and computation needs. Verticals provide network operators with the specification of the services they want to provide, e.g., the virtual (network) functions (VNFs) they want to use to process their data and the associated quality of service.

Mobile network operators are in charge of mapping the requirements of the verticals into infrastructure management decisions. This task is part of the network *orchestration*, and includes making decisions concerning (i) the *placement* of the VNFs needed by the verticals across the infrastructure; (ii) the *assignment* of CPU, memory and storage resources to the VNFs; (iii) the *routing* of data across network nodes.

These decisions interact with each other in ways that are complex and often counterintuitive. In this paper, we focus on the allocation of computational and network resources, and make such decisions jointly, accounting for (i) the requirements of each VNF and vertical; (ii) the capabilities of the network operator's infrastructure; (iii) the capacity and latency of the links between network nodes. A key aspect of our work, often disregarded by previous literature on 5G and VNF placement, is that our approach allows *flexible allocation* of the computational capabilities of each host among the VNFs it runs.

We identify queuing theory as the best tool to model 5G networks, owing to the nature of its traffic and the processing it needs. Indeed:

- most of Internet-of-things (IoT) and machine-type communication (MTC) traffic will consist of RESTful, atomic (in principle) requests [2, Sec. 6.11], as opposed to long-standing connections;
- such requests will traverse one or more processing stages, as implemented in the emerging multi-access edge computing (MEC) implementation Amazon Greengrass [3], and can trigger additional requests in the process;
- the time it takes to process each request depends on the capabilities of the computational entity serving it [3].

Requests and processing stages naturally map onto clients and queues they have to traverse. Furthermore, the fact that queues can be assigned different service rates aptly models our flexible allocation of computational resources.

We take *service delay* as our main key performance indicator (KPI), and we formulate an optimization problem that minimizes the maximum ratio between actual and maximum allowed end-to-end latency, across all services. Furthermore, and without loss of generality, we focus on CPU as the resource to assign to VNFs. In light of the complexity of the problem, we then propose an efficient solution strategy, closely matching the optimum: based on (i) *decoupling* the VNF placement and CPU assignment decisions, while keeping track of their interdependence, and (ii) *sequentially* making such decisions for each VNF. Traffic routing decisions are simply derived once all placement and assignment decisions are made. Although made in a decoupled and sequential fashion, our decisions are joint as their mutual impact is properly accounted for, e.g., we consider how deploying a new VNF on a host impacts the possible CPU assignments therein.

Our main contributions can be summarized as follows:

- our model accounts for the main resources of 5G networks, namely, hosts and links;

- *S. Agarwal is with IIT Guwahati, India. F. Malandrino and C.-F. Chiasserini are with Politecnico di Torino, Italy. S. De is with IIT Delhi, India.*
- *A preliminary version [1] of this work was presented at the IEEE INFOCOM 2018 conference.*

- we model the diverse requirements of different VNFs, and allow them to be composed in arbitrarily complex graphs, as mandated by [4, Sec. 6.5], instead of simpler chains or directed acyclic graphs (DAGs);
- unlike existing work, we allow *flexible* allocation of CPU to VNFs, and model the resulting impact on service times;
- we propose a solution strategy, called MaxZ, that is able to efficiently and effectively make VNF placement and CPU allocation decisions, and show how it consistently performs very close to the optimum across a variety of traffic requirements;
- focusing on the special case of fully-load conditions, we state and prove several properties of the optimal CPU allocation decisions, and use them to further speed up the decision process.

The remainder of the paper is organized as follows. Sec. 2 reviews related work, highlighting the novelty of our contribution. Sec. 3 positions our work within the context of the ETSI management and orchestration (MANO) framework. Sec. 4 describes the system model, while Sec. 5 introduces the problem formulation and analyzes its complexity. Sec. 6 presents our solution concept, while Sec. 7 describes how we deal with the special case of full-load conditions. Sec. 8 addresses scenarios with multiple VNF instances. Finally, Sec. 9 presents performance evaluation results, while Sec. 10 concludes the paper.

2 Related work

Network slicing and orchestration. A first body of works concerns the network slicing paradigm and its role within 5G. Several works, including [5]–[7], focus on the architecture of 5G networks based on network slicing, pointing out their opportunities and challenges. Other works, e.g., [8], [9], address decision-making in 5G networks and the associated challenges, including computational complexity. Finally, orchestration, including the decision-making involved entities and the arising security concerns have been tackled in, e.g., [10] and [11], respectively.

Network-centric optimization. Many works, including [12]–[16], tackle the problems of VNF placement and routing from a network-centric viewpoint, i.e., they aim at minimizing the load of network resources. In particular, [12] seeks to balance the load on links and servers, while [13] studies how to optimize routing to minimize network utilization. The above approaches formulate mixed-integer linear programming (MILP) problems and propose heuristic strategies to solve them. [14], [15], and [16] formulate ILP problems, respectively aiming at minimizing the cost of used links and network nodes, minimizing resource utilization subject to QoS requirements, and minimizing bitrate variations through the VNF graph.

Service provider's perspective. Several recent works take the viewpoint of a service provider, supporting multiple services that require different, yet overlapping, sets of VNFs, and seek to maximize its revenue. The works [17], [18] aim at minimizing the energy consumption resulting from VNF placement decisions. [19], [20] study how to place VNFs between network-based and cloud servers so as to minimize the cost, and [21] studies how to design the VNF graphs themselves, in order to adapt to the network topology.

User-centric perspective. Closer to our own approach, several works take a user-centric perspective, aiming at optimizing the user experience. [22], [23] study the VNF placement problem, accounting for the computational capabilities of hosts as well as network delays. In [24], the authors consider inter-cloud latencies and VNF response times, and solve the resulting ILP through an affinity-based heuristic.

Virtual EPC. The Evolved Packet Core (EPC) is a prime example of a service that can be provided through software defined networking and network function virtualization (SDN/NFV). Interestingly, different works use different VNF graphs to implement EPC, e.g., splitting user- and control-plane entities [25]–[27] or joining together the packet and service gateways (PGW and SGW) [28], [29]. Our model and algorithms work with any VNF graph, which allows us to model any real-world service, including all implementations of vEPC.

2.1 Novelty

The closest works to ours, in terms of approach and/or methodology, are [22], [23], [24], and [28].

In particular, [22], [23], and [28] model the assignment of VNFs to servers as a generalized assignment problem, a resource-constrained shortest path problem and a MILP problem, respectively. This implies that either a server has enough spare CPU capacity to offer a VNF, or it does not. Our queuing model, instead, is the first to account for the flexible allocation of CPU to the VNFs running on a host, e.g., the fact that VNFs will work faster if placed at a scarcely-utilized server. Furthermore, [22] and [28] have as objective the minimization of costs and server utilization, respectively. Our objective, instead, is to minimize the delay incurred by requests of different classes, which changes the solution strategy that can be adopted. The work [23] aims at solving essentially the same problem as ours, albeit in the specific scenario where all traffic flows through a deterministic sequence of VNFs, i.e., VNF graphs are chains.

The queuing model used in [24] is similar (in principle) to ours; however, [24] does not address overlaps between VNF graphs and only considers DAGs, i.e., requests cannot visit the same VNF more than once. Furthermore, in both [23] and [24] no CPU allocation decisions are made, and the objective is to minimize a global metric, ignoring the different requirements of different service classes. Finally, the affinity-based placement heuristic proposed in [24] neglects the inter-host latencies, and this, as confirmed by our numerical results in Sec. 9, can yield suboptimal performance.

Finally, it is worth mentioning that a preliminary version of this paper appeared in [1]. While sharing the same basic solution concept, this version includes a substantial amount of new and revised material, including a discussion on how our work fits in the 5G MANO framework (Sec. 3), an extended discussion of full-load conditions (Sec. 7), and new results for large-scale scenarios.

3 Our work and the ETSI MANO framework

ETSI has standardized [4] the management and orchestration (MANO) framework, including a set of functional blocks and the reference points, i.e., the interfaces between functional blocks (akin to a REST API) that they use to communicate. Its high-level purpose is to translate business-facing KPIs chosen by the vertical (e.g., the type of processing needed and the associated end-to-end delay) into resource-facing decisions such as virtual resource instantiation, VNF placement, and traffic routing. In this section, we first present a brief overview of the ETSI MANO framework; then, in Sec. 3.1, we focus on the NFV orchestrator and detail the decisions it has to make and the input data at its disposal.

Fig. 1 presents the functions composing the MANO framework (within the blue area) as well as the functions outside the framework they interact with. Operation and business support (OSS/BSS) service block, which represent the interface between verticals and mobile operators. High-level, end-to-end requirements and KPIs are conveyed, through the Os-Ma-nfvo reference point, to the NFV orchestrator (NFVO). The NFVO is in charge of deciding the number and type of VNFs to instantiate as well as the capacity of virtual links (VLs) connecting them.

Such decisions are conveyed, via the Or-vnfm interface, to the VNF manager (VNFM) function, which is in charge of actually instantiating the required VNFs. The VNFM requests from the virtual infrastructure manager (VIM) any resource, e.g., virtual machine (VM) or VL needed by the VNFs themselves. The VNFM also interacts with the element management (EM) function, a non-MANO entity that is in charge of Fault, Configuration, Accounting, Performance and Security (FCAPS) management for the functional part of the VNFs, i.e., for the actual tasks they perform.

Finally, the VIM interacts with the NFV infrastructure (NFVI), which includes the hardware (e.g., physical servers, network equipment, etc.) and software (e.g., hypervisors) running the VNFs.

3.1 The NFVO: input, output, and decisions

As its name suggests, the main entity in charge of orchestration decisions is the NFV orchestrator (NFVO), which belongs to the MANO framework depicted in Fig. 1. In the following, we provide more details on the decisions the NFVO has to make and the information it can rely upon, which correspond (respectively) to the output and input of our algorithms.

The NFVO receives from the OSS/BSS a data structure called *network service descriptor* (NSD), defined in [4, Sec. 6.2.1]. NSDs include a graph-like description of the processing each service requires, e.g., the VNFs that the traffic has to traverse, in the form of a *VNF Forwarding Graph (VNFFG)* descriptor [4, Sec. 6.5.1]. They contain *deployment flavor* information, including the delay requirements associated with every service [4, Sec. 6.2.1.3]. Additionally, from the virtual infrastructure manager (VIM), the NFVO fetches information on the state and availability of network infrastructure, including VMs able to run the VNFs and the links connecting them.

With such information, the NFVO can make what ETSI calls *lifecycle management* decisions [4, Sec. 7.2] about the VNFs composing each network slice, i.e., how many instances of these VNFs to instantiate, where to host them, and how much resources to assign to each of them. Such decisions will correspond to decision variables in our system model, as detailed next.

4 System model

We model VNFs as M/M/1 *queues*, belonging to set $\mathcal{Q}$, whose customers correspond to service requests. The *class* of each customer corresponds to the service with which each request is associated; we denote the set of such classes by $\mathcal{K}$. The *service rate* $\mu(q)$ of each queue q reflects the amount of CPU (expressed in, e.g., ticks or microseconds of CPU-time) each VNF is assigned to. Thus, $\mu(q)$ influences the time taken to process one service request. Notice that $\mu(q)$ does not depend on the class k; that is, CPU is assigned on a per-VNF rather than per-class basis. This models those scenarios where the same VNF instance can serve requests belonging to multiple services.

Arrival rates at queue $q \in \mathcal{Q}$ are denoted by $\lambda_k(q)$. Note that these values are class-specific, and reflect the amount of traffic of different services. Class-specific *transfer probabilities* $\mathbb{P}(q_2|q_1, k)$ indicate the probability that a service request of class k enters VNF q_2 after being served by VNF q_1. Furthermore, $\mathbb{P}(q|\circ, k)$ indicates the probability that a request of class k starts its processing at VNF q.

Physical, or more commonly virtual, *hosts* are represented by set $\mathcal{H}$. Each host h has a finite *CPU capacity* κ_h. Host-specific κ_h values account for both different capabilities and different hosts, and the fact that some hosts may be assigned a low-power CPU state [30] for energy-saving purposes. This implies that energy constraints can be accounted for by properly setting the values of the κ_h parameters.

Going from host $h \in \mathcal{H}$ to host $l \in \mathcal{H}$ entails a deterministic *network latency* $\delta(h, l)$, which depends on the (virtual) link between the two hosts. Furthermore, the link between hosts h and l has a finite capacity $C(h, l)$. The delay and capacity parameters, i.e., $\delta(h, l)$ and $C(h, l)$, are able to describe in a consistent way two different cases, namely:

- there is a direct, physical link between hosts h and l;
- h and l are connected through multiple physical links, which are abstracted as one virtual link.

In the latter case, the capacity of the virtual link corresponds to that of the physical link with lowest capacity, while the delay is the sum of individual delays. Such information is part of the input data to our problem.

The notation used is summarized in Tab. 1, and exemplified below.

Example 1. Assume that the network has to support three services: video streaming, gaming, and vehicle collision detection. Then the set of service classes is $\mathcal{K} = \{\text{video}, \text{game}, \text{veh}\}$. For the sake of clarity, let us associate to each service the following, highly simplified, VNF graphs:

- *video streaming*: firewall – transcoder – billing;

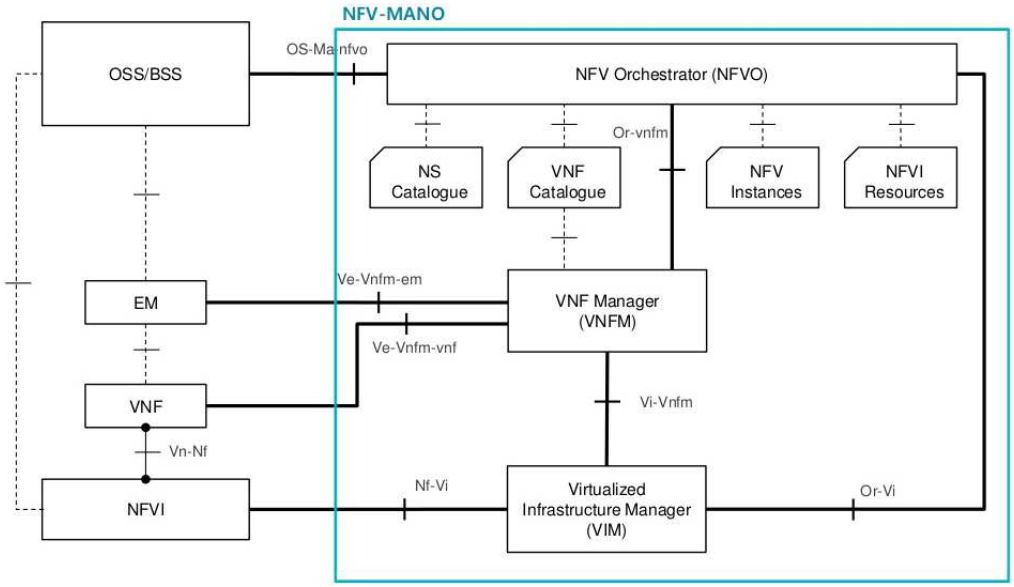

Fig. 1. The NFV-MANO architectural framework. Source: [4]

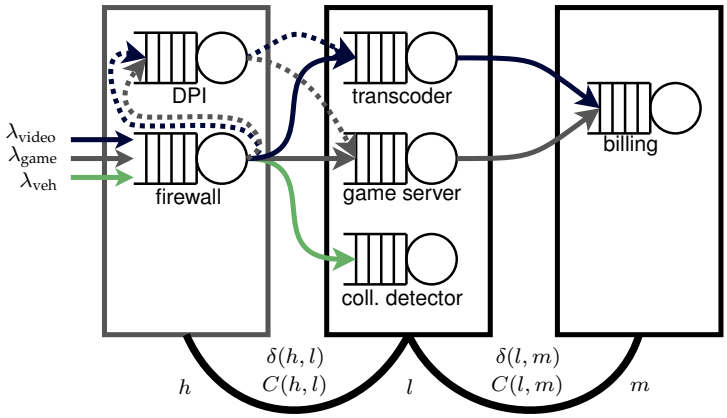

Fig. 2. Example 1: three service graphs, and six VNFs, corresponding to the six queues, placed across three hosts. Dashed and dotted lines represent the different paths that service requests can take.

- *gaming*: firewall – game server – billing;
- *vehicle collision detection*: firewall – collision detector.

Suspicious-looking packets belonging to the video streaming and gaming services can further be routed through a deep packet inspection (DPI) VNF. Hence, $\mathcal{Q}$={firewall, transcoder, billing, game server, collision detector, DPI}.

There are three hosts $\mathcal{H} = \{h, l, m\}$, connected to each other through links characterized by a latency δ and a link capacity C. Fig. 2 illustrates the above quantities and shows a possible VNF placement across the three hosts. Routing can be deterministic, e.g., $\mathbb{P}(\text{billing}|\text{transcoder}, \text{video})$=1, or it can be probabilistic, e.g., $\mathbb{P}(\text{DPI}|\text{firewall}, \text{gaming})$=0.1 and $\mathbb{P}(\text{game server}|\text{firewall}, \text{gaming})$=0.9.

TABLE 1
Notation

$A(h,q)$	Binary variable	Whether to deploy VNF q at host h
$C(h,l)$	Parameter	Capacity of the link between hosts h and l
D_k	Aux. var.	Total delay incurred by requests of class k
D_k^{QoS}	Parameter	Target delay for requests of class k
$h \in \mathcal{H}$	Set	Physical hosts
$k \in \mathcal{K}$	Set	Services (traffic classes)
$q \in \mathcal{Q}$	Set	VNFs (queues)
$\mathbb{P}(q_1\|q_2,k)$	Parameter	Probability that requests of class k visit VNF q_2 (immediately) after VNF q_1
$R_k(q)$	Aux. var.	Processing time for requests of class k at VNF q
$\gamma_k(q)$	Aux. var.	Number of times requests of class k visit VNF q
$\delta(h,l)$	Parameter	Delay associated with the link between hosts h and l
$\Lambda_k(q)$	Aux. var.	Rate at which requests of any class arrive at VNF q
$\lambda_k(q)$	Aux. var.	Rate at which *new* requests of class k arrive at VNF q
$\hat{\lambda}_k(q)$	Aux. var.	Total rate at which requests of class k arrive at VNF q
κ_h	Parameter	Computational capacity of host h
$\mu(q)$	Real variable	Computational capacity to assign to VNF q

5 Problem Formulation and Complexity

When allocating the resources necessary to run each service, the NFVO has to make three main decisions:

- VNF placement, i.e., which physical hosts have to run the required VNFs;
- CPU assignment, i.e., how the computational capabilities of each host have to be shared between VNF;
- how traffic shall be steered between VNFs.

In this section, we describe how such decisions and the constraints they are subject to can be described through our

system model. We take service delay as our main performance metric, and we formulate the problem for scenarios where exactly one instance of each VNF is to be deployed in the network. The general case where multiple instances of the same VNF can be deployed is then discussed in Sec. 8.

Decisions and decision variables. We have two main decision variables: a binary variable $A(h,q) \in \{0,1\}$ represents whether VNF $q \in \mathcal{Q}$ is deployed at host $h \in \mathcal{H}$, and a real variable $\mu(q)$ expresses the amount of CPU assigned to VNF $q \in \mathcal{Q}$. Notice how $\mu(q)$ maps onto the service rate of the corresponding queue.

System constraints. A first, basic constraint that allocation decisions must meet is that the computational capacity of hosts is not exceeded, i.e.,

$$\sum_{q \in \mathcal{Q}} A(h,q)\mu(q) \leq \kappa_h, \quad \forall h \in \mathcal{H}. \tag{1}$$

As mentioned above, we present our model in the case where there is exactly one instance of each VNF deployed in the system. This translates into:

$$\sum_{h \in \mathcal{H}} A(h,q) = 1, \quad \forall q \in \mathcal{Q}. \tag{2}$$

Arrival rates and system stability. In addition not to overload hosts, allocation decisions must ensure that individual VNFs are assigned enough computational capacity to cope with their load, i.e., that the system is *stable*. Recall that input parameters $\lambda_k(q)$ express the rate at which *new* requests of service class k arrive at queue $q \in \mathcal{Q}$. We can then define an auxiliary variable $\hat{\lambda}_k(q)$, expressing the *total* rate of requests of class k that enter queue q, either from outside the system or from other queues. For any $k \in \mathcal{K}$, we have:

$$\hat{\lambda}_k(q) = \sum_{q \in \mathcal{Q}} \lambda_{k,q} + \sum_{p \in \mathcal{Q}} \mathbb{P}(q|p,k)\hat{\lambda}_k(p). \tag{3}$$

We can then define another auxiliary variable $\Lambda(q)$, expressing the total arrival rate of requests of any class entering queue q:

$$\Lambda(q) = \sum_{k \in \mathcal{K}} \hat{\lambda}_k(q).$$

Using $\Lambda(q)$, we can impose *system stability*, requesting that, for each queue, the arrival rate does not exceed the service rate:

$$\Lambda(q) < \mu(q), \quad \forall q \in \mathcal{Q}. \tag{4}$$

In other words, each VNF should receive *at least* enough CPU to deal with the incoming traffic. If additional CPU is available at the host, it will be exploited to further speed up the processing of requests.

Latency. The previous constraints ensure that individual VNFs are stable, i.e., they process incoming requests in a finite time. We can now widen our focus, and study how the processing times of different VNFs and the network times combine to form our main metric of interest, i.e., the *delay* each request is subject to.

The processing time, i.e., the time it takes for a request of service k to traverse VNF q is represented by an auxiliary variable $R_k(q)$. For FCFS (first come, first serve) and PS (processor sharing) queuing disciplines, we have:

$$R_k(q) = \frac{1}{\mu(q) - \Lambda(q)}, \quad \forall q \in \mathcal{Q} \tag{5}$$

Note that the right-hand side of (5) does not depend on class k; intuitively, this is because the queuing disciplines we consider are unaware of service classes. The response times for other queuing disciplines, including those accounting for priority levels and/or preemption, cannot be expressed in closed form. It is also worth stressing that present-day implementations of multi-access edge computing (MEC) [3] are based on FIFO discipline, and do not support preemption.

To compute the network latency that requests incur when transiting between hosts, we first need the expected number of times, $\gamma_k(q)$, that a request of class k visits VNF $q \in \mathcal{Q}$, i.e.,

$$\gamma_k(q) = \mathbb{P}(q|\circ,k) + \sum_{p \in \mathcal{Q} \setminus \{q\}} \mathbb{P}(q|p,k)\gamma_k(p). \tag{6}$$

In the right-hand side of (6), the first term is the probability that requests start their processing at queue q, and the second is the probability that requests arrive there from another queue p. Note that $\gamma_k(q)$ is not an auxiliary variable, but a quantity that can be computed offline given the transfer probabilities $\mathbb{P}$. Using $\gamma_k(q)$, the expected network latency incurred by requests of service class k is:

$$\sum_{q,r \in \mathcal{Q}} \gamma_k(q)\mathbb{P}(r|q,k) \sum_{h,l \in \mathcal{H}} \delta(h,l)A(h,q)A(l,r). \tag{7}$$

We can read (7) from left to right, as follows. Given a service request of class k, it will be processed by VNF q for $\gamma_k(q)$ number of times. Every time, it will move to VNF r with probability $\mathbb{P}(r|q,k)$. So doing, it will incur latency $\delta(h,l)$ if q and r are deployed at hosts h and l, respectively (i.e., if $A(h,q) = 1$ and $A(l,r) = 1$).

The average total delay of requests of the generic service class k is therefore given by:

$$D_k = \sum_{q \in \mathcal{Q}} \gamma_k(q)R_k(q) + \sum_{q,r \in \mathcal{Q}, q \neq r} \gamma_k(q)\mathbb{P}(r|q,k) \sum_{h,l \in \mathcal{H}} A(h,q)A(l,r)\delta(h,l). \tag{8}$$

Link capacity. Given the finite link capacity $C(h,l)$, which limits the number of requests that move from any VNF at host h to any VNF at host l,we have:

$$\sum_{k \in \mathcal{K}} \sum_{q,r \in \mathcal{Q}} \hat{\lambda}_k(q)\mathbb{P}(r|q,k)A(q,h)A(r,l) \leq C(h,l). \tag{9}$$

Constraint (9) contains a summation over all classes k and all VNFs $q, r \in \mathcal{Q}$, such that q is deployed at h and r is deployed at l, as expressed by the A-variables. For each of such pair of VNFs, $\hat{\lambda}_k(q)$ is the rate of the requests of class k that arrive at q. Multiplying it by $\mathbb{P}(r|q,k)$, we get the rate at which requests move from VNF q to VNF r, hence from host h to host l.

Objective. D_k defined above represents the average delay incurred by requests of class k. In our objective function, we have to combine these values in a way that reflects the differences between such classes, most notably, their different QoS limits. Thus, we consider for each class k the

ratio of the actual delay D_k to the limit delay D_k^{QoS}, and seek to minimize the maximum of such ratios:

$$\min_{A,\mu} \max_{k \in \mathcal{K}} \frac{D_k}{D_k^{\text{QoS}}}. \tag{10}$$

Importantly, the above objective function not only ensures fairness among service classes while accounting for their limit delay, but it also guarantees that the optimal solution will match *all* QoS limits if possible. More formally:

Property 1. If there is a non-empty set of solutions that meet constraints (1)–(9) and honor the services QoS limits, then the optimal solution to (10) falls in such a set.

Proof: We prove the property by contradiction, and assume that there is a feasible solution such that $D'_k \leq D_k^{\text{QoS}}$ for all service classes, but that the optimal solution has $D^\star_{\hat{k}} > D_{\hat{k}}^{\text{QoS}}$ for at least one class $\hat{k} \in \mathcal{K}$.

In this case, the optimal value of the objective (10) would be at least $\frac{D^\star_{\hat{k}}}{D_{\hat{k}}^{\text{QoS}}} > 1$. However, we know by hypothesis that there is a feasible solution where $D_k \leq D_k^{\text{QoS}}$ for all classes, which would result in an objective function value of $\min_{k \in \mathcal{K}} \frac{D'_k}{D_k^{\text{QoS}}} \leq 1$. It follows that the solution we assumed to be optimal cannot be so. □

Furthermore, when no solution meeting all QoS limits exists, the solution optimizing (10) will minimize the damage by keeping all delays as close as possible to their limit values.

5.1 Problem complexity

The VNF placement/CPU assignment problem is akin to max-flow problem; however, it has a much higher complexity due to the following: (i) binary variables control whether edges and nodes are activated, and (ii) the cost associated with edges changes according to the values of said variables. More formally, the problem of maximizing (10) subject to constraints (1)–(9) includes both binary ($A(h,q)$) and continuous ($\mu(q)$) variables. More importantly, constraints (1) and (9), as well as objective (10) (see also (8)), are nonlinear and non-convex, as both include products between different decision variables.

Below we prove that such a problem is NP-hard, through a reduction from the generalized assignment problem (GAP).

Theorem 1. The problem of joint VNF placement and CPU assignment is NP-hard.

Proof: It is possible to reduce the GAP, which is NP-hard [31], to ours. In other words, we show that (i) for each instance of the GAP problem, there is a corresponding instance of our VNF placement problem, and (ii) that the translation between them can be done in polynomial time.

GAP instance. The GAP instance includes *items* $i_1, \ldots, i_N$ and *bins* $b_1, \ldots, b_M$. Each bin b has a budget (size) s_b; placing item i at bin b consumes a budget (weight) w_{bi} and yields a cost p_{bi}. The decision variables are binary flags x_{bi} stating whether item i shall be assigned to bin b; also, each item shall be assigned to exactly one bin. The objective is to minimize the cost.

Reduction. In our problem, items and bins correspond to VNFs and hosts respectively, and the decision variables x_{bi} correspond to VNF placement decisions $A(i,b)$. The capacity of each host is equal to the size s_b of the corresponding bin. Furthermore, we must ensure that:

- the weight w_{bi} of item i when placed at bin b corresponds to the quantity of CPU assigned to VNF i, i.e., $w_{bi} = \mu^b(i)$;
- the cost p_{bi} coming from placing item i in bin b corresponds to the opposite[1] of the processing time at VNF i, i.e., $w_{bi} = -\frac{1}{\mu^b(i) - \Lambda(i)}$, or equivalently, with a linear equation, $\Lambda(i) - \mu^b(i) = \frac{1}{p_{bi}}$.

Finally, we set all inter-host delays to zero.

Complexity of the reduction. Performing the reduction described above only requires to solve a linear system of equations in the $\mu^b(i)$ and $\Lambda(i)$ variables, which can be performed in polynomial (indeed, cubic) time [32]. We have therefore presented a polynomial-time reduction of any instance of the GAP problem to our problem. It follows that our problem is NP-hard, q.e.d. □

It is interesting to notice how, in the proof of Theorem 1, we obtain a *simplified* version of our problem, with non-flexible CPU assignment (if VNF i is placed at host b it gets exactly $\mu^b(i)$ CPU) and no network delay. This suggests that our problem is indeed more complex than GAP.

The NP-hardness of the problem rules out not only the possibility to directly optimize the problem through a solver, but also commonplace solution strategies based on *relaxation*, i.e., allowing binary variables to take values anywhere in $[0,1]$. Even if we relaxed the $A(h,q)$ variables, we would still be faced with a non-convex formulation, for which no algorithm is guaranteed to find a global optimum.

One approach to overcome such an issue could be simplifying the model, e.g., by assuming that any host has sufficient computing capability to run simultaneously all VNFs, therefore dispensing with the $A(h,q)$ variables. However, by doing so we would detach ourselves from real-world 5G systems, thus jeopardizing the validity of the conclusions we draw from our analysis. We instead opt to keep the model unchanged and present an efficient, *decoupled* solution strategy, leveraging on sequential decision making.

6 Solution strategy

Our solution strategy is based on decoupling the problems of VNF placement and CPU allocation, and then sequentially – and yet jointly, i.e., accounting for their mutual impact – making these decisions. We begin by presenting our VNF placement heuristic, called *MaxZ*, in Sec. 6.1, and then discuss CPU allocation in Sec. 6.2.

6.1 The MaxZ placement heuristic

As mentioned earlier, the two main sources of problem complexity are binary variables and non-convex functions in both objective (10) and constraints (1) and (9). In order to solve the VNF placement problem, our heuristic walks around these issues by:

1) formulating a convex version of the problem;

1. So that minimizing the cost is the same as minimizing the service time.

2) solving it through an off-the-shelf solver;
3) computing, for each VNF q and host h, a *score* $Z(h,q)$, expressing how confident we feel about placing q in h;
4) considering the maximum score $Z(h^\star, q^\star)$ and placing VNF $q^\star$ at host $h^\star$;
5) repeating steps 2–4 until all VNFs are placed.

The name of the heuristic comes from step 4, where we seek for the highest score Z.

6.1.1 Steps 1–2: convex formulation

To make the problem formulation in Sec. 5 convex, first we need to get rid of binary variables; specifically, we replace the binary variables $A(h,q) \in \{0,1\}$ with continuous variables $\tilde{A}(h,q) \in [0,1]$.

We also need to remove the products between $\tilde{A}$-variables (e.g., in (7), (8), and (9)), by replacing them with a new variable. To this end, for each pair of VNFs q and r and hosts h and l, we introduce a new variable $\Phi(h,l,q,r) \in [0,1]$, and impose that:

$$\Phi(h,l,q,r) \leq \tilde{A}(h,q), \quad \forall h,l \in \mathcal{H}, q,r \in \mathcal{Q}; \tag{11}$$

$$\Phi(h,l,q,r) \leq \tilde{A}(l,r), \quad \forall h,l \in \mathcal{H}, q,r \in \mathcal{Q}; \tag{12}$$

$$\Phi(h,l,q,r) \geq \tilde{A}(h,q) + \tilde{A}(l,r) - 1, \quad \forall h,l \in \mathcal{H}, q,r \in \mathcal{Q}. \tag{13}$$

The intuition behind constraints (11)–(13) is that $\Phi(h,l,q,r)$ mimics the behavior of the product $\tilde{A}(h,q)\tilde{A}(l,r)$: if either $\tilde{A}(h,q)$ or $\tilde{A}(l,r)$ are close to 0, then (11) and (12) guarantee that $\Phi(h,l,q,r)$ will also be close to zero; if both values are close to one, then (13) allows also $\Phi(h,l,q,r)$ to be close to one.

Another product between variables, i.e., a term in the form $A(h,q)\mu(q)$, appears in (1). Following a similar approach, we introduce a set of new variables, $\psi(h,q)$, mimicking the ratio between the $A(h,q)\mu(q)$ product and the host capacity κ_h. We then impose:

$$\psi(h,q) \leq \tilde{A}(h,q), \quad \forall h \in \mathcal{H}, q \in \mathcal{Q}; \tag{14}$$

$$\sum_{q \in \mathcal{Q}} \psi(h,q) \leq 1, \quad \forall h \in \mathcal{H}, \tag{15}$$

which mimic (1). By replacing all products between $\tilde{A}$-variables with a Φ-variable and all products between $\tilde{A}$- and μ-variables with a ψ-variable, we obtain a *convex problem*, which can efficiently be solved through commercial solvers.

6.1.2 Steps 3–4: Z-score and decisions

Let us assume that no VNF has been placed yet. We then solve an instance of the convex problem described in Sec. 6.1.1, and use the values of the variables $\tilde{A}(h,q)$ and $\psi(h,q)$ to decide which VNF to place at which host.

Recall that $\tilde{A}(h,q)$ is the relaxed version of our placement variable $A(h,q)$, so we would be inclined to use that to make our decision. However, we also need to account for how much computational capacity VNFs would get, as expressed by $\psi(h,q)$. If such a value falls below the threshold $T_\psi(h,q) = \frac{\Lambda(q)}{\kappa_h}$, then VNF q may not be able to process the incoming requests, i.e., constraint (4) may be violated.

To prevent this, we define our Z-score, i.e., how confident we are about placing VNF q at host h, as follows:

$$Z(h,q) = \tilde{A}(h,q) + \mathbb{1}_{[\psi(h,q) \geq T_\psi(h,q)]}, \tag{16}$$

where $\mathbb{1}$ is the indicator function. Recalling that $\tilde{A}$-values are constrained between 0 and 1, favoring high values of (16) means that we prefer a deployment that results in ψ-values greater than the threshold, if such a deployment exists. Otherwise, we make the placement decision based on the $\tilde{A}$-values only.

Specifically, we select the host $h^\star$ and VNF $q^\star$ associated with the maximum Z, i.e., $h^\star, q^\star \leftarrow \arg\max_{h \in \mathcal{H}, q \in \mathcal{Q}} Z(h,q)$, and place VNF $q^\star$ in host $h^\star$. We fix this decision and repeat the procedure till all VNFs are placed (i.e., we perform exactly $|\mathcal{Q}|$ iterations).

We now present two example runs of MaxZ, for two scenarios with different inter-host latencies.

Example 2. Consider a simple case with two hosts $\mathcal{H} = \{h_1, h_2\}$ with the same CPU capacity $\kappa_h = 5\,\text{requests/s}$, two VNFs $\mathcal{Q} = \{q_1, q_2\}$, and only one request class k with $\lambda_k = 1\,\text{requests/s}$. Requests need to subsequently traverse q_1 and q_2. The inter-host latency $\delta(h_1,h_2)$ is set to 5 ms, while $D^{\text{QoS}} = 50\,\text{ms}$. Then, intuitively, the optimal solution is to deploy one VNF per host.
We solve the problem in Sec. 6.1.1. After the first iteration, we obtain $\tilde{\mathbf{A}} = \left[\begin{smallmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{smallmatrix}\right]$, $\psi = \left[\begin{smallmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{smallmatrix}\right]$, and $\mathbf{Z} = \left[\begin{smallmatrix} 1.5 & 1.5 \\ 1.5 & 1.5 \end{smallmatrix}\right]$ [2]. In such a case, using a tie-breaking rule, we place VNF q_1 at host h_1. In the second iteration, we have $\tilde{\mathbf{A}} = \left[\begin{smallmatrix} 1 & 0.38 \\ 0 & 0.62 \end{smallmatrix}\right]$, $\psi = \left[\begin{smallmatrix} 0.8 & 0.19 \\ 0 & 0.61 \end{smallmatrix}\right]$, and $\mathbf{Z} = \left[\begin{smallmatrix} 2 & 1.38 \\ 0 & 1.62 \end{smallmatrix}\right]$. We ignore the entries pertaining to VNF q_1 that has been already placed and, since $Z(h_2,q_2) > Z(h_1,q_2)$, we deploy VNF q_2 at host h_2, which corresponds to the intuition that, given the small value of δ, VNFs should be spread across the hosts.

Example 3. Let us now consider the same scenario as in Example 2, but assume a much longer latency $\delta(h_1,h_2) = 100\,\text{ms}$. The best solution will now be to place both VNFs at the same host.
After the first iteration, we obtain $\tilde{\mathbf{A}} = \left[\begin{smallmatrix} 0.7 & 0.7 \\ 0.3 & 0.3 \end{smallmatrix}\right]$, $\psi = \left[\begin{smallmatrix} 0.5 & 0.5 \\ 0.3 & 0.3 \end{smallmatrix}\right]$, and $\mathbf{Z} = \left[\begin{smallmatrix} 1.7 & 1.7 \\ 1.3 & 1.3 \end{smallmatrix}\right]$. Again using a tie-breaking rule, we place VNF q_1 at host h_1. In the second iteration, we have $\tilde{\mathbf{A}} = \left[\begin{smallmatrix} 1 & 0.7 \\ 0 & 0.2 \end{smallmatrix}\right]$, $\psi = \left[\begin{smallmatrix} 0.6 & 0.4 \\ 0 & 0.2 \end{smallmatrix}\right]$, and $\mathbf{Z} = \left[\begin{smallmatrix} 2 & 1.8 \\ 0 & 1.2 \end{smallmatrix}\right]$. We again ignore the entries in the first column and, since $Z(h_1,q_2) > Z(h_2,q_2)$, we place VNF q_2 at host h_1, making optimal decisions.

6.2 CPU allocation

Once the MaxZ heuristic introduced in Sec. 6.1 provides us with deployment decisions, we need to decide the CPU allocation, i.e., the values of the $\mu(q)$ variables in the original problem described in Sec. 5. This can be achieved simply by solving the problem in (10) but keeping the deployment decision fixed, i.e., replacing the $A(h,q)$ variables with

2. In all matrices, rows correspond to hosts and columns to VNFs.

parameters whose values come from the MaxZ heuristic. Indeed, we can prove the following property.

Property 2. If the deployment decisions are fixed, then the problem of optimizing (10) subject to (1)–(9) is convex.

Proof: Several constraints of the original problem only involve $A(h,q)$ variables, and thus simply become conditions on the input parameters: this is the case of (2), (3), and (9). Also, constraints (1) and (4) are linear in the variables $\mu(q)$. With regard to the objective function, (8) is now linear with respect to $\mu(q)$, while (5) is convex, even if it does not look so *prima facie*. Indeed, its second derivative is $\frac{\mathbf{d}}{\mathbf{d}^2\mu(q)}\frac{1}{\mu(q)-\lambda(q)} = \frac{2}{(\mu(q)-\Lambda(q))^3}$, which is positive for any $\mu(q) > \Lambda(q)$. That condition is required for system stability; therefore, we can conclude that constraint (5) is convex over the all region of interest. Finally, the objective function in (10) is in min-max form, which preserves convexity. □

Property 2 guarantees that we can make our CPU allocation decisions, i.e., decide on the $\mu(q)$ values, in polynomial time. We can further enhance the solution efficiency by reducing the optimization problem to the resolution of a system of equations, through the Karush-Kuhn-Tucker (KKT) conditions.

6.2.1 KKT conditions

In several nonlinear problems, including convex ones, optimal solutions are guaranteed to have certain properties, known as KKT conditions [33]. This greatly simplifies and speeds up the search for the optimum, as such a search can be restricted to solutions satisfying the KKT conditions.

The KKT conditions are:

1) stationarity;
2) primal feasibility;
3) dual feasibility;
4) complementary slackness.

Stating them requires associating (i) re-writing the objective and constraints in *normal form*, and (ii) associating a *KKT multiplier* with each of the constraints.

Therefore, we introduce an auxiliary variable ρ representing the maximum $\frac{D_k}{D_k^{\text{QoS}}}$ ratio, and imposing that for each service class $k \in \mathcal{K}$:

$$\rho \geq \frac{D_k}{D_k^{\text{QoS}}} = \frac{1}{D_k^{\text{QoS}}}\left(\sum_{q\in\mathcal{Q}}\frac{\gamma_k(q)}{\mu(q)-\Lambda(q)} + \sum_{q,r\in\mathcal{Q}}\gamma_k(q)\mathbb{P}(r|q,k)\sum_{h_1,h_2\in\mathcal{H}}A(h_1,q)A(h_2,r)\delta(h_1,h_2)\right) \quad (17)$$

At this point, the objective is simply to minimize ρ.

We also need to re-write constraints (1), (4) and (17) in normal form, and associate to them the multipliers M_q, M_h and M_k respectively. The resulting Lagrangian function is:

$$L = \rho + \sum_{q\in\mathcal{Q}} M_q X_q + \sum_{h\in\mathcal{H}} M_h Y_h + \sum_{k\in\mathcal{K}} M_k W_k, \quad (18)$$

where:

$$X_q = -\mu(q) + \Lambda(q);$$

$$Y_h = \sum_{q\in\mathcal{Q}} A(h,q)\mu(q) - \kappa_h;$$

$$W_k = \sum_{q\in\mathcal{Q}}\frac{\gamma_k(q)}{D_k^{\text{QoS}}}\frac{1}{\mu(q)-\Lambda(q)} + \sum_{q,r\in\mathcal{Q}}\gamma_k(q)\mathbb{P}(r|q,k)\sum_{h_1,h_2\in\mathcal{H}}A(h_1,q)A(h_2,r)\frac{\delta(h_1,h_2)}{D_k^{\text{QoS}}} - \rho.$$

Stationarity, the first KKT condition, requires that $\nabla_{r\rho\mu(q)}L = 0$, which translates into the following equations:

$$\frac{\partial}{\partial\rho}L = 0 \iff 1 - \sum_{k\in\mathcal{K}} M_k = 0. \quad (19)$$

Furthermore, for each $q \in \mathcal{Q}$, we have:

$$\frac{\partial}{\partial\mu(q)}L = 0 \iff -M_q + \sum_{h\in\mathcal{H}} M_h A(h,q) + - \sum_{k\in\mathcal{K}}\frac{M_k\gamma_k(q)}{D_k^{\text{QoS}}}\frac{1}{(\mu(q)-\Lambda(q))^2} = 0 \quad (20)$$

The primal feasibility condition requires that all constraints are met. The third one, dual feasibility, is only relevant for problems that contain equality constraints, which is not our case.

Finally, complementary slackness requires that either the inequality constraints are active, or the corresponding multipliers are zero, i.e.,

$$M_q X_q = 0, \quad \forall q \in \mathcal{Q}, \quad (21)$$

$$M_h Y_h = 0, \forall h \in \mathcal{H}, \quad (22)$$

$$M_k W_k = 0, \quad \forall k \in \mathcal{K}. \quad (23)$$

Based on (21)–(23), the multipliers assume the following meaning:

- M_q is zero for all *stable* queues, i.e., the queues fulfilling the condition $\mu(q) > \Lambda(q)$;
- M_h is zero for all *non-strained* hosts, i.e., hosts used for less than their CPU capacity κ_h;
- M_k is zero for all *non-critical* classes, i.e., classes for which the $\frac{D_k}{D_k^{\text{QoS}}}$ ratio is strictly lower than ρ.

We can now determine the *global* computational complexity of our approach, including the VNF placement through the MaxZ heuristic and the CPU allocation by optimizing (10).

Property 3. Our solution strategy, including the MaxZ VNF placement heuristic in Sec. 6.1 and the CPU allocation strategy in Sec. 6.2 has polynomial computational complexity.

Proof: Running the MaxZ heuristic requires solving exactly $|\mathcal{Q}|$ convex problems, and the complexity of doing so is cubic in the number of variables, which in turn is linear in $|\mathcal{H}|$ and $|\mathcal{Q}|$. It follows that the total complexity of MaxZ is $O\left(|\mathcal{Q}|(|\mathcal{Q}||\mathcal{H}|)^3\right) = O(\max\left\{|\mathcal{Q}|^4, |\mathcal{H}|^3\right\})$. MaxZ also dominates the total computational complexity, because deciding the CPU allocation as described in Sec. 6.2, requires only solving a system of equations, which has [32] cubic complexity in the number $|\mathcal{Q}|$ of variables. □

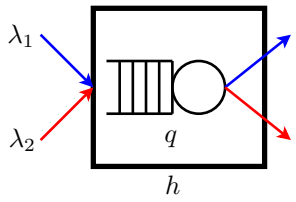

Fig. 3. A simple system where two classes of clients traverse the same queue. Host h will always be strained; additionally, depending on the values of $D_k^{\max}$, either one or both the classes will be critical.

7 Special case: full-load conditions

In this section, we seek to further reduce the complexity of the CPU allocation problem. Let us start from the Lagrange multipliers derived earlier, and recall that we require *stability*, i.e., $\Lambda(q) < \mu(q)$, hence $M_q = 0$ for all queues $q \in \mathcal{Q}$.

Given the above and (20), we can write that, for each queue $q \in \mathcal{Q}$ deployed at host h,

$$M_h = \sum_{k \in \mathcal{K}} M_k \frac{\gamma_k(q)}{D_k^{\max}} \frac{1}{\left(\mu(q) - \Lambda(q)\right)^2}. \tag{24}$$

Recalling the meaning of the multipliers, we can state the following lemma and properties.

Lemma 1. If CPU assignment decisions are made optimizing the objective (10), then there exists at least one critical class, i.e., for which equality holds in (17).

Proof: Constraint (17) must be active for at least one $k \in \mathcal{K}$, otherwise, the selected value of r would not be optimal. □

Property 4. All hosts traversed by service requests of critical classes are strained.

Proof: Let k be a critical class (hence, $M_k > 0$), q be a host serving it, and $H(q)$ the host q is deployed at. From (24), it follows:

$$M_{H(q)} \geq M_k \frac{\gamma_k(q)}{D_k^{\max}} \frac{1}{\left(\mu(q) - \Lambda(q)\right)^2}.$$

The quantity at the second member is positive ($M_k > 0$ because class k is strained, and $\gamma_k(q) > 0$ because by hypothesis clients of class k are served at q). This implies that $M_{H(q)} > 0$, and therefore, by (22), that host h is strained. □

Property 5. VNFs deployed at a strained host serve at least one critical class each.

Proof: Let us consider a queue q, By hypothesis, its host $H(q)$ is strained, i.e., $M_{H(q)} > 0$. From (24), it follows:

$$\sum_{k \in \mathcal{K}} M_k \frac{\gamma_k(q)}{D_k^{\max}} \frac{1}{\left(\mu(q) - \Lambda(q)\right)^2} > 0,$$

which can only be if there is at least one class k that is critical (hence $M_k > 0$) and whose clients are served by q (hence $\gamma_k(q) > 0$). □

In summary, we are guaranteed that there is at least one critical class, and that *all* the hosts it traverses are strained, and that *each* of the VNFs deployed on the strained hosts (not only the ones serving requests of the original critical class) serve at least one critical class. This can lead to a cascade effect, as shown in Example 4.

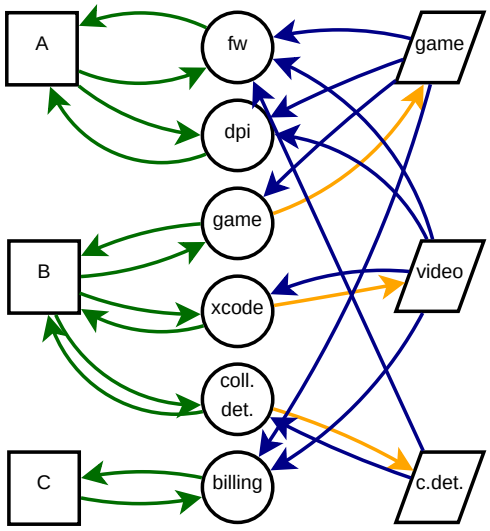

Fig. 4. The graph $\mathcal{G}$ generated for the system depicted in Fig. 2. Left, center and right edges correspond to hosts, VNFs and classes respectively. Green edges are created according to rule (ii), blue edges according to rule (iii), and yellow edges according to rule (iv).

Example 4. Consider the case in Fig. 2. By Lemma 1, at least one class is critical; let us assume that such a class is collision detection. From Property 4, all hosts traversed by collision detection requests, i.e., hosts h and l, are strained. Since host l is strained, from Property 5 it follows that each of its VNFs serves at least one critical class. Since the transcoder queue only serves the video class, the video class is critical. Similarly, since the game server only serves the game class, that class is critical as well. Finally, both video and game classes traverse host m; therefore, by Property 4, that host is critical as well.

The cascade effect shown in Example 4 might lead us to conjecture that all classes are critical and all hosts are strained. However, this is not true in general. A simple counterexample is represented in Fig. 3, where two classes share the same queue. By Lemma 1, one of the two classes will be critical, and, hence, by Property 4, host h will be strained. Property 5 tells us what we already know, i.e., that one of the two classes will be critical, but it does not imply that *both* will be. Indeed, that depends on the values of $D_k^{\max}$: if $D_{k_1}^{\max} = D_{k_2}^{\max}$, then both classes are critical; otherwise, the class with the lowest $D_k^{\max}$ value will be critical and the other will not.

However, we can state a *sufficient* condition for all classes to be critical (and, hence, all hosts to be strained), *regardless* the $D_k^{\max}$ values. It is based on (i) building a graph $\mathcal{G}$ representing the hosts, VNFs and classes in our system (as shown in Fig. 4), and (ii) verifying a simple property over it.

Theorem 2. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed graph where:

(i) there is a vertex for every host, queue, and class, i.e., $\mathcal{V} = \mathcal{H} \cup \mathcal{Q} \cup \mathcal{K}$;

(ii) for every host h and queue q s.t. $\mathbf{A}(h, q) = 1$, add

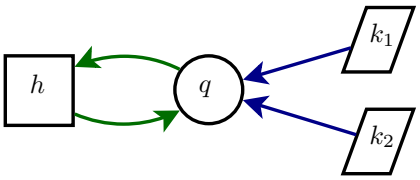

Fig. 5. The graph $\mathcal{G}$ generated for the system in Fig. 3, which is not strongly connected (it is impossible to reach k_1 from k_2).

to $\mathcal{E}$ a pair of edges (q,h) and (h,q);
(iii) for every queue q and class k s.t. $\gamma_k(h) > 0$, add to $\mathcal{E}$ an edge (k,q);
(iv) for every queue q and class k s.t. $\gamma_k(h) > 0$ and k is the only class using q, i.e., $\gamma_j = 0, \forall j \neq k$, add to $\mathcal{E}$ an edge (q,k).
If graph $\mathcal{G}$ is strongly connected, then all classes in $\mathcal{K}$ are critical and all hosts in $\mathcal{H}$ are strained.

Proof: Lemma 1 guarantees us that there is at least one critical class $k^\star$; let us then start walking through the graph from the corresponding edge and mark all vertices we can reach as critical (if corresponding to classes) or strained (if corresponding to hosts). Through edges added according to rule (ii) and (iii), we will be able to reach all hosts traversed by clients of the critical class, and those hosts will be strained as per Property 4. Edges outgoing from the host vertices, created according to rule (ii), will make us reach all queues deployed at these hosts. By Property 5, each of these queues serves at least one critical class. If this class is unique, i.e., if we have an edge created according to rule (iv), then those classes are critical as well, and we can repeat the process.

The strong connectivity property implies that we can reach all vertices (including all classes and all hosts) from any vertex of $\mathcal{G}$, including the one critical class whose existence is guaranteed by Lemma 1. □

Fig. 4 presents the graph $\mathcal{G}$ resulting from the system in Fig. 2, which is strongly connected. Fig. 5 presents the graph for the system in Fig. 3, which is not strongly connected and, thus, it does not meet the sufficient condition stated in Theorem 2. Recall that, because that condition is sufficient but not necessary, k_1 and k_2 could still be both critical, depending on their $D_k^{\max}$ values.

In scenarios like the one in Example 4, where all classes are critical and all hosts are strained, we have:

$$\sum_{q\in\mathcal{Q}} \frac{\gamma_k(q)}{D_k^{\text{QoS}}} \frac{1}{\mu(q)-\Lambda(q)} + \\ + \sum_{h_1,h_2\in\mathcal{H}} \sum_{q,r\in\mathcal{Q}} A(h_1,q)A(h_2,r)\gamma_k(q)\mathbb{P}(r|q,k)\frac{\delta(h_1,h_2)}{D_k^{\text{QoS}}} = \rho \quad \forall k\in\mathcal{K},$$

$$\sum_{q\in\mathcal{Q}} A(h,q)\mu(q) = \kappa_h, \quad \forall h\in\mathcal{H}.$$

The above equations can be combined with the KKT conditions stated in Sec. 6.2.1, thus forcing $Y_h = 0\ \forall h \in \mathcal{H}$ and $W_k = 0\ \forall k \in \mathcal{K}$. This greatly simplifies and speeds up the process of finding the optimal CPU allocation values $\mu(q)$.

8 Multiple VNF Instances

So far, we presented our system model and solution strategy in the case where exactly one instance of each VNF has to be deployed. This is not true in general; some VNFs may need to be replicated owing to their complexity and/or load.

If the number N_q of instances of VNF q to be deployed is known, then we can replace VNF q in the VNF graph with N_q replicas thereof, labeled $q^1, q^2, \ldots q^{N_q}$, each with the same incoming and outgoing edges. With regard to the $\Lambda(q)$ requests/s that have to be processed by *any* instance of VNF q, they are split among the instances. If $f(q,i)$ is the fraction of requests for VNF q that is processed by instance q^i (and thus $\sum_{i=1}^{N_q} f(q,i) = 1$), then instance q^i gets requests at a rate $f(q,i)\Lambda(q)$. It is important to stress that once the $f(q,i)$ splitting fractions are known, then the resulting problem can be solved with the approach described in Sec. 6.

Establishing the $f(q,i)$ values is a complex problem; indeed, straightforward solutions like uniformly splitting flows (i.e., $f(q,i) = \frac{1}{N_q}$), are in general suboptimal. We therefore resort to a *pattern search* [34] iterative approach.

Without loss of generality, we describe our approach in the simple case $N_q = 2$. In this case, the splitting values are $f(q,1) = f$ and $f(q,2) = 1-f$. Given an initial guess f_0, an initial step Δ and a minimum step $\epsilon < \Delta$, we proceed as follows:

1) we initialize the splitting factor f to the initial guess f_0;
2) using the procedure detailed in Sec. 6, we compute the objective value (10) for the splitting values f, $f+\Delta$, and $f-\Delta$;
3) if the best result (i.e., the lowest value of (10)) is obtained for splitting value $f+\Delta$ or $f-\Delta$, then we replace f with that value and loop to step 2;
4) otherwise, we reduce Δ by half;
5) if now Δ is lower than ϵ, the algorithm terminates;
6) otherwise, we loop to step 2.

The intuition of the pattern-search procedure is similar, in principle, to gradient-search methods. If we find that using $f+\Delta$ or $f-\Delta$ instead of f produces a lower delay, then we replace the current value of f with the new one; otherwise, we try new f-values closer to the current one. When we are satisfied that there are no better f-values further than ϵ from the current one, the search ends.

Notice that in step 2 of our procedure we run the decision-making procedure described in Sec. 6; this implies that, once we find the best value of f, we also know the best VNF placement and CPU allocation decisions.

From the viewpoint of complexity, the multi-instance case requires running the procedure detailed in Sec. 6 for as many times as there are values of f to try. Since at every iteration we half the step Δ, the number of such values is at most $\lceil \log_2 \frac{\Delta_0}{\epsilon} \rceil$, Δ_0 being the initial step. Such a logarithmic term does not impact the order of the global complexity, which remains polynomial as proven in Property 3.

9 Numerical Results

After describing the reference topology and benchmark alternatives in Sec. 9.1, this section reports the performance of

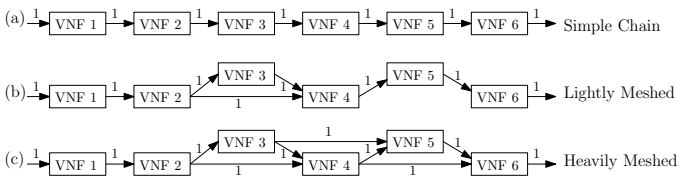

Fig. 6. The VNF graphs used in our performance evaluation, reflecting real-world service implementations.

MaxZ as a function of the inter-host latency and arrival rates (Sec. 9.2). Next, the effects of multi-class and multi-instance scenarios (Sec. 9.3) and larger-scale topologies (Sec. 9.4) are presented. Finally, in Sec. 9.5, we generalize our results and study the number of solutions that are examined by MaxZ and its counterparts, as well as the associated running times.

9.1 Reference scenario

We consider a reference topology with three hosts having CPU capacity $\kappa_h = 10\,\text{requests/ms}$ each. As in Example 1, hosts are connected to each other through physical links having latency that varies between $50\,\text{ms}$ and $400\,\text{ms}$. For simplicity, we disregard the link capacity, i.e., we assume computation to be the bottleneck in our scenario, as it is commonly the case in single-tenant scenarios. Throughout our performance evaluation, we benchmark the MaxZ placement heuristic in Sec. 6.1 against the following alternatives:

- *global optimum*, found through brute-force search of all possible deployments;
- *greedy*, where the number of used hosts is minimized, i.e., VNFs are concentrated as much as possible;
- *affinity-based* [24], trying to place at the same host VNFs with high transition probability between them.

After the VNF placement decisions are made, we compute the optimal CPU allocation, i.e., the optimal $\mu(q)$ values, as explained in Sec. 6.2. It is important to remark that the CPU allocation strategy is the same for all placement strategies.

We first focus on a single request class k, fix the arrival rate to $\lambda_k = 1\,\text{requests/ms}$, and compare the three VNF graphs depicted in Fig. 6, ranging from a simple chain to a complex meshed topology. Notice how in graphs (b) and (c) requests can *branch* and *merge*, i.e., the number of requests outgoing from a VNF does not match the number of incoming ones. This is the case with several real-world functions; in particular, the light and heavy mesh topologies are akin to vEPC implementations where user- and control-plane entities are joint [29] and split [26], [27], i.e., implemented, respectively, by the same VNF or by separate ones.

9.2 Effect of physical link latency and arrival rate

Fig. 7 shows the average service delay as a function of the physical link latency for the VNF graphs presented in Fig. 6. We can observe that the performance of Greedy is always the same regardless of such latency, as all VNFs are deployed at the same host. On the other hand, the performance of Affinity-based is quite good for low values of the physical links latency, but then quickly degrades, due to the fact that the affinity-based heuristic disregards link latency. As far as MaxZ is concerned, it exhibits an excellent performance: it consistently yields a substantially lower service delay compared to Greedy and Affinity-based, and is always very close to the optimum.

Fig. 8 focuses on the heavy mesh topology, and breaks down the service delay yielded by MaxZ into its computation and traversing latency components. Processing latency only depends upon the VNF placement, while traversing latency depends upon both the VNF placement (which determines how many inter-host links are traversed) and the per-link latency. When such latency is low, MaxZ tends to spread the VNFs across all available hosts, in order to assign more CPU. As the physical link latency increases, the placement becomes more and more concentrated (thus resulting in lower $\mu(q)$ values and higher processing times), until, when the link latency is very high, all VNFs are placed at the same host and there is no traversing latency at all.

Fig. 7 and Fig. 8 clearly illustrate the importance of flexible CPU allocation. If we only accounted for the minimum CPU required by VNFs, as in [22], [24], we could place all of them in the same host, as the Greedy strategy does. This would result, as we can see from the far right in Fig. 8, in high processing times and *two unused* hosts.

We now fix the physical link latency to $50\,\text{ms}$, and change the arrival rate λ between $0.1\,\text{requests/ms}$ and $2\,\text{requests/ms}$; Fig. 9 summarizes the service delay yielded by the placement strategies we have studied. A first observation concerns the Greedy strategy: since all VNFs are placed in the same host, as λ increases, VNFs receive an amount of CPU that is barely above the minimum limit $\Lambda(q)$. This, as per (5), results in processing times that grow very large. The difference between the other placement strategies tends to become less significant; intuitively, this is because processing times dominate the total delay, and thus spreading the VNFs as much as possible is always a sensible solution. MaxZ still consistently outperforms Affinity-based, and performs very close to the optimum.

9.3 Multiple class and VNF instances

In Fig. 10, we move to a multi-class scenario where $|\mathcal{K}| = 3$ service classes share the same VNF graph. The three classes have limit delays D^{QoS} of $10\,\text{ms}$, $45\,\text{ms}$, and $2\,\text{s}$, respectively corresponding to safety applications (e.g., collision detection), real-time applications (e.g., gaming), and delay-tolerant applications (e.g., video streaming). Fig. 10 shows that all placement strategies result in delays that are roughly proportional to the limit ones. Also, the relative performance of the placement strategies remains unmodified – MaxZ outperforms Affinity-based and is close to the optimum, while Greedy yields much higher delay. Notice that, for very high values of physical link latency, it is impossible to meet all QoS constraints, i.e., $\frac{D_k}{D_k^{\text{QoS}}} > 1$ for at least one class k. In these cases, MaxZ limits the damage by keeping the $\frac{D_k}{D_k^{\text{QoS}}}$ ratios as low as possible.

It is also interesting to notice in Fig. 10(center) that the service delay yielded by MaxZ is actually lower than the optimum. However, this does not mean that MaxZ outperforms the optimum; indeed, due to the min-max structure of our objective (10), the objective value is determined by the class with the highest $\frac{D_k}{D_k^{QoS}}$ ratio. For low-to-medium link

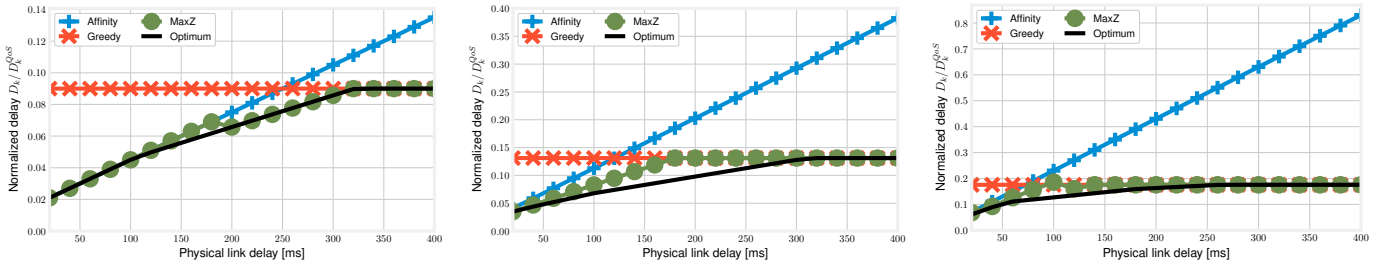

Fig. 7. Normalized service delay as a function of the physical link latency, for the chain (left), light mesh (center), heavy mesh (right) VNF graphs. Note that the y-axis scale varies across the plots.

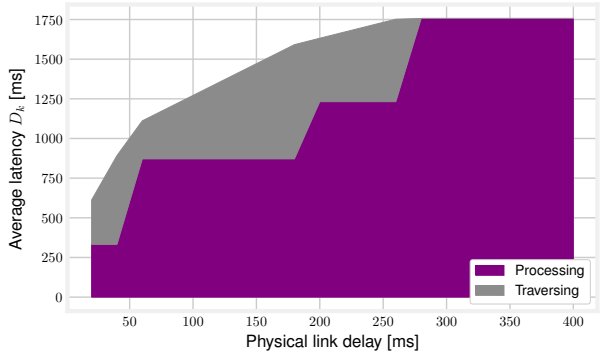

Fig. 8. Breakdown of the total service delay as a function of the physical link latency, for the heavy mesh topology and the MaxZ deployment strategy.

latency, such class is the low-delay one, with a normalized service delay (shown in Fig. 10(left)) around 0.9. Thus, MaxZ and the optimum strategy obtain the same objective value.

In Fig. 11, we drop the assumption that there is only one instance of each VNF; specifically, for VNF_4 and VNF_6 we allow two instances each. By comparing Fig. 11 to Fig. 7, we can immediately notice that allowing multiple VNF instances substantially decreases the total delay. More interestingly, we can observe that MaxZ always outperforms its alternatives, and is very close to the optimum, except for some cases when the topology is very complex.

9.4 Larger-scale scenarios

We now move to larger-scale, more complex scenarios, where:

- the VNF graph is the *extreme mesh* depicted in Fig. 12 and inspired to real-world VNF graphs [35];
- there can be up to two copies of each VNF;
- each host is connected through physical links with either 4 or 6 other hosts chosen in such a way that any pair of hosts is connected through one virtual link.

The fact that the physical topology is not strongly connected implies that the latency of virtual links between hosts depends on many physical links they are made of.

Fig. 13, obtained when each host is connected to four others, shows that the relative performance of the solution strategies does not change, with MaxZ still outperforming its alternatives. By comparing Fig. 13 with Fig. 7(right), we can observe much higher service delays, often exceeding D_k^{QoS}. This further highlights how MaxZ can provide near-optimal performance in all conditions, including sparsely-connected topologies where wrong placement decisions can result in very high network delays.

Moving to Fig. 14, obtained when each host is connected to six others, we can observe a slightly better performance for all strategies, due to shorter network delays. MaxZ is still able to clearly outperform all the alternatives, due to its ability to account for both network latencies and computation times.

9.5 Generalization: examined solutions and running times

In the following, we seek to generalize the results shown above and study the number of solutions (i.e., values given to the $A(h,q)$ variables) that our benchmark approaches examine while searching for the best one. Recall that, as shown in Property 3, MaxZ has a complexity of $O(\max\{|\mathcal{Q}|^4, |\mathcal{H}|^3\})$.

Brute-force. We use brute-force to seek for the globally-optimal solution. There are $|\mathcal{Q}|$ number of VNFs to be placed in $|\mathcal{Q}|$ hosts. Thus, in total there are $|\mathcal{H}|^{|\mathcal{Q}|}$ possible number of VNF-host placements. For each of the VNF-host placement, we obtain the optimal resource allocation for each VNF at each host by solving a convex optimization problem. Convex problems have cubic complexity in the number of variables, and each problem has $|\mathcal{Q}|+1$ variables, corresponding one $\mu(q)$ each VNF and a variable corresponding to r. The problem complexity in this case is $\mathcal{O}((|\mathcal{Q}|+1)^3)$. Thus, the overall computational complexity of global optimization utilizing brute-force is $\mathcal{O}(|\mathcal{H}|^{|\mathcal{Q}|}(|\mathcal{Q}|+1)^3)$.

Greedy scheme. In the greedy scheme, firstly the VNFs are placed on the hosts such that the number of hosts utilized is minimized. Thereafter, optimal resources are allocated by solving the convex optimization problem. The VNFs are first sorted in the decreasing order of their load which has the computational complexity of $\mathcal{O}(|\mathcal{Q}|\log(|\mathcal{Q}|))$. They are then allocated to $|\mathcal{H}|$ hosts, iteratively. The greedy placement algorithm has a complexity of $\mathcal{O}(|\mathcal{Q}|\log(|\mathcal{Q}|)+|\mathcal{H}|)$. As seen before, the computational complexity of resource allocation algorithm is $\mathcal{O}((|\mathcal{Q}|+1)^3)$. Thus, the overall computational complexity of greedy algorithm is $\mathcal{O}(|\mathcal{Q}|\log(|\mathcal{Q}|)+|\mathcal{H}|+(|\mathcal{Q}|+1)^3)$, which is equivalent to $\mathcal{O}(|\mathcal{H}|+(|\mathcal{Q}|+1)^3)$.

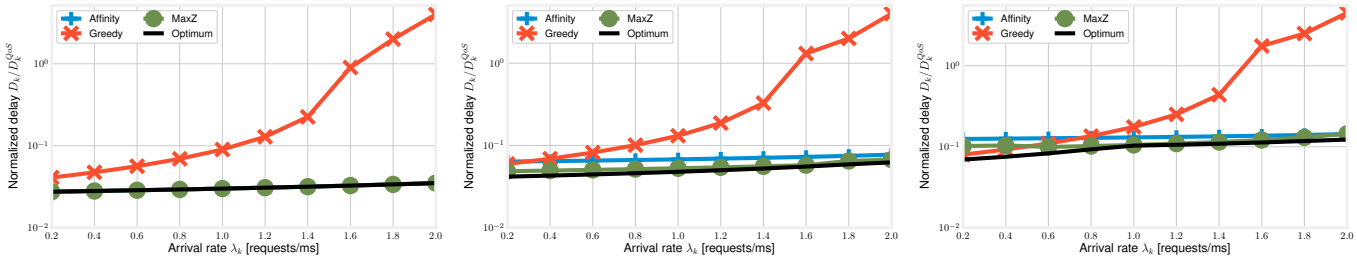

Fig. 9. Normalized service delay (log scale) as a function of arrival rate λ for the chain (left), light mesh (center), heavy mesh (right) VNF graphs.

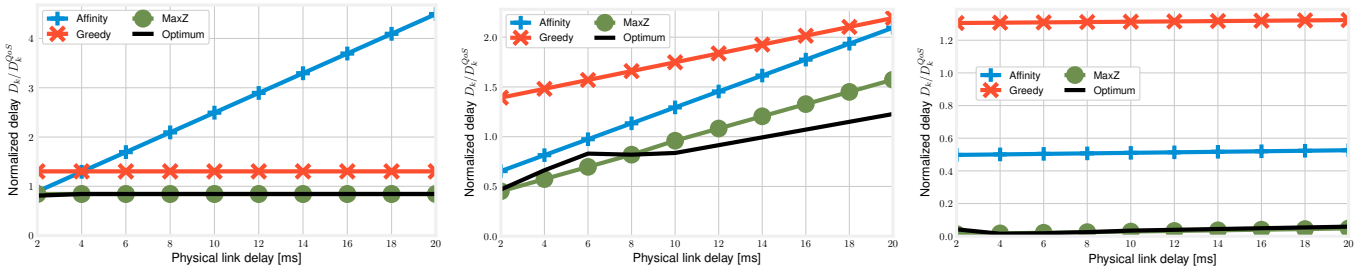

Fig. 10. Multi-class scenario, heavy mesh graph: normalized service delay vs. arrival rate λ for the low-delay (left), medium-delay (center), high-delay (right) service classes. Note that the y-axis scale varies across the plots.

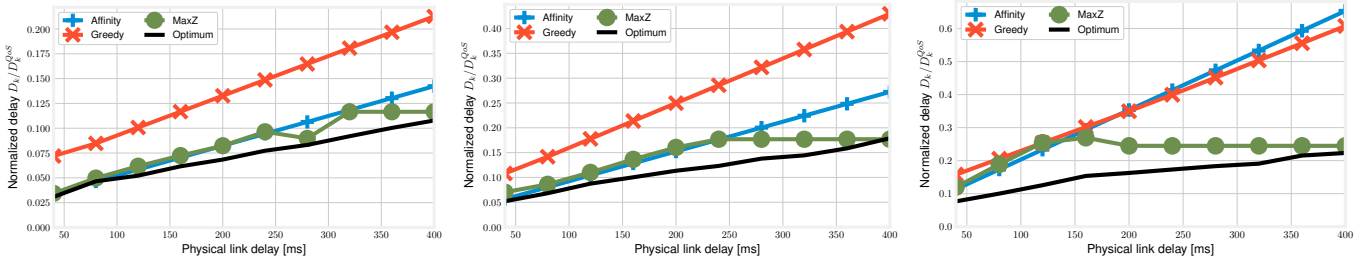

Fig. 11. Multi-instance scenario: normalized service delay vs. the physical link latency for the chain (left), light mesh (center), heavy mesh (right) VNF graphs. Note that the y-axis scale varies across the plots.

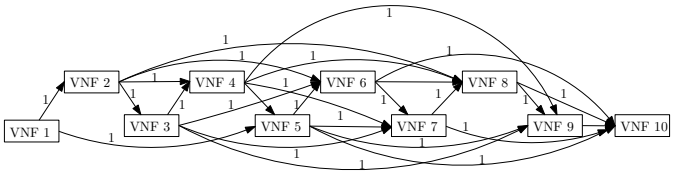

Fig. 12. Extreme mesh VNF graph.

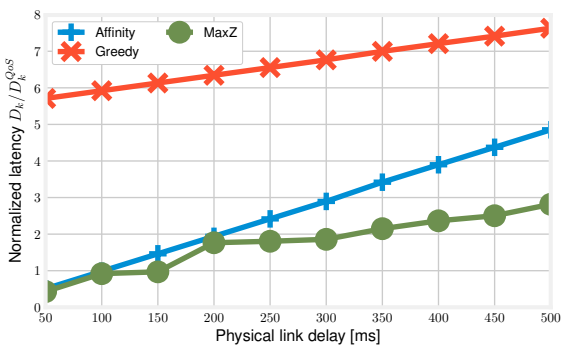

Fig. 13. Extreme mesh VNF graph, 20-host topology with connectivity degree 4: normalized service delay vs. physical link latency.

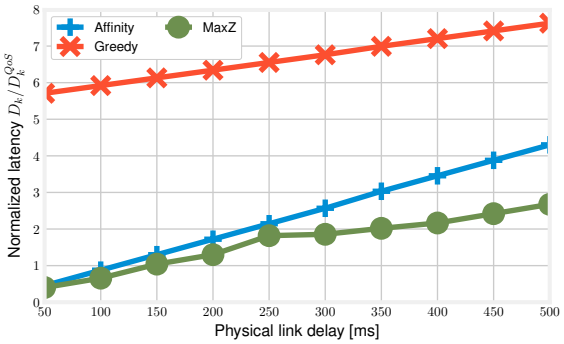

Fig. 14. Extreme mesh VNF graph, 20-host topology with connectivity degree 6: normalized service delay vs. physical link latency δ.

Affinity-based scheme. In the affinity-based scheme, firstly the VNFs are placed on the hosts such that the VNFs having high transition probability between them are placed on the same host. Thereafter, optimal resources are allocated by solving the convex optimization problem. There are $|\mathcal{Q}|$ number of VNFs to be placed in $|\mathcal{H}|$ hosts. The affinity based VNF-host placement algorithm has a complexity of

TABLE 2
Execution time (in seconds) for different schemes

Scenario	Aff.	Greedy	MaxZ	Brute
Base scenario (chain graph)	0.32	0.34	4.22	238.83
Base scenario (light mesh graph)	0.32	0.29	4.31	238.26
Base scenario (heavy mesh graph)	0.35	0.34	4.53	246.18
Base scenario (multiple classes)	0.61	0.62	10.14	415.05
Large scenario, 4-degree connectivity (extreme mesh graph, two instances per VNF)	0.62	0.50	23.36	
Large scenario, 6-degree connectivity (extreme mesh graph, two instances per VNF)	0.62	0.50	21.34	

$\mathcal{O}(|\mathcal{Q}||\mathcal{H}|)$. As seen before, the computational complexity of resource allocation algorithm is $\mathcal{O}((|\mathcal{Q}|+1)^3)$. Thus, the overall computational complexity of this scheme is $\mathcal{O}(|\mathcal{H}||\mathcal{Q}|+(|\mathcal{Q}|+1)^3)$.

Execution times. All the above computations refer to the order of magnitude of the worst-case computational complexity. However, it is also interesting to assess how such complexity translates into actual execution times. To this end, Tab. 2 reports the execution times of MaxZ and its counterparts for different topologies and VNF graphs, measured on a server equipped with a Xeon E5-2600 processor and 48 GByte of RAM. We can clearly observe that, while MaxZ takes longer than the affinity-based and greedy heuristics to run, their execution times are comparable in the base scenario. Furthermore, MaxZ runs over two orders of magnitude faster than the brute-force procedure. It is also interesting to notice that the execution times in the large scenario are still limited, while the brute-force procedure is utterly unable to tackle that case.

10 Conclusion

We targeted the problem of orchestration in 5G networks, that requires to make decisions about VNF placement, CPU assignment, and traffic routing. We presented a queuing-based model accounting for all the main features of 5G networks, including (i) arbitrarily complex service graphs; (ii) flexible allocation of CPU power to VNFs sharing the same host, and its impact on processing time; (iii) the possibility of having multiple instances of the same VNF.

Based on our model, we presented a methodology to make the requirement decisions jointly and effectively, based on two pillars: a VNF placement heuristics called MaxZ, and a convex formulation of the CPU allocation problem given placement decisions. We also showed, based on KKT conditions, that the CPU allocation problem is further simplified in full-load conditions, where all hosts are completely utilized. We evaluated our methodology against multiple VNF graphs and physical topologies of varying complexity, and found the performance of MaxZ to consistently exceed that of state-of-the-art alternatives, and closely match the optimum. Future research directions include multi-tenant scenarios, where multiple verticals share the same infrastructure. Furthermore, future work will aim at optimizing the number of instances to be deployed for each VNF, and designing a low-complexity heuristic for it, and will investigate other KPIs than service delay.

Acknowledgment

This work was supported by the European Commission through the H2020 5G-TRANSFORMER project (Project ID 761536).

References

[1] S. Agarwal, F. Malandrino, C.-F. Chiasserini, and S. De, "Joint VNF Placement and CPU Allocation in 5G," in *IEEE INFOCOM*, 2018.
[2] ETSI. GS MEC 009: Mobile Edge Computing (MEC); General principles for Mobile Edge Service APIs.
[3] Amazon. AWS Greengrass. https://aws.amazon.com/greengrass/.
[4] ETSI. (2017) Network Functions Virtualisation (NFV); Management and Orchestration. http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf.
[5] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, 2017.
[6] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5g and future mobile networks: mobility, resource management, and challenges," *IEEE Communications Magazine*, 2017.
[7] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega *et al.*, "Network slicing to enable scalability and flexibility in 5g mobile networks," *IEEE Communications magazine*, 2017.
[8] K. Samdanis, S. Wright, A. Banchs, A. Capone, M. Ulema, and K. Obana, "5g network slicing–part 2: Algorithms and practice," *IEEE Communications Magazine*, 2017.
[9] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Communications Magazine*, 2017.
[10] X. Li, J. Mangues-Bafalluy, I. Pascual, G. Landi, F. Moscatelli, K. Antevski, C. J. Bernardos, L. Valcarenghi, B. Martini, C. F. Chiasserini *et al.*, "Service orchestration and federation for verticals," in *IEEE WCNC Workshops*, 2018.
[11] M. A. S. Santos, A. Ranjbar, G. Biczók, B. Martini, and F. Paolucci, "Security requirements for multi-operator virtualized network and service orchestration for 5g," in *Guide to Security in SDN and NFV*. Springer, 2017.
[12] A. Hirwe and K. Kataoka, "LightChain: A lightweight optimization of VNF placement for service chaining in NFV," in *IEEE NetSoft*, 2016.
[13] T. W. Kuo, B. H. Liou, K. C. J. Lin, and M. J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *IEEE INFOCOM*, 2016.
[14] A. Baumgartner, V. S. Reddy, and T. Bauschert, "Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization," in *IEEE NetSoft*, 2015.
[15] F. Ben Jemaa, G. Pujolle, and M. Pariente, "Analytical Models for QoS-driven VNF Placement and Provisioning in Wireless Carrier Cloud," in *ACM MSWiM*, 2016.
[16] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *IEEE CloudNet*, 2015.
[17] A. Marotta and A. Kassler, "A Power Efficient and Robust Virtual Network Functions Placement Problem," in *IEEE ITC*, 2016.
[18] N. E. Khoury, S. Ayoubi, and C. Assi, "Energy-Aware Placement and Scheduling of Network Traffic Flows with Deadlines on Virtual Network Functions," in *IEEE CloudNet*, 2016.
[19] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, 2016.

[20] L. Gu, S. Tao, D. Zeng, and H. Jin, "Communication cost efficient virtualized network function placement for big data processing," in *IEEE INFOCOM Workshops*, 2016.
[21] J. Cao, Y. Zhang, W. An, X. Chen, J. Sun, and Y. Han, "VNF-FG design and VNF placement for 5G mobile networks," *Science China Information Sciences*, 2017.
[22] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM*, 2015.
[23] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, and P. Castoldi, "Latency-aware composition of virtual functions in 5G," in *IEEE NetSoft*, 2015.
[24] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Computer Communications*, 2017.
[25] A. Baumgartner, V. S. Reddy, and T. Bauschert, "Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization," in *IEEE NetSoft*, 2015.
[26] G. Hasegawa and M. Murata, "Joint Bearer Aggregation and Control-Data Plane Separation in LTE EPC for Increasing M2M Communication Capacity," in *IEEE GLOBECOM*, 2015.
[27] A. Ksentini, M. Bagaa, and T. Taleb, "On Using SDN in 5G: The Controller Placement Problem," in *IEEE Globecom*, 2016.
[28] D. Dietrich, C. Papagianni, P. Papadimitriou, and J. S. Baras, "Network function placement on virtualized cellular cores," in *COMSNETS*, 2017.
[29] J. Prados-Garzon, J. J. Ramos-Munoz, P. Ameigeiras, P. Andres-Maldonado, and J. M. Lopez-Soler, "Modeling and Dimensioning of a Virtualized MME for 5G Mobile Networks," *IEEE Transactions on Vehicular Technology*, 2017.
[30] Intel. Power Management States: P-States, C-States, and Package C-States. https://software.intel.com/en-us/articles/power-management-states-p-states-c-states-and-package-c-states.
[31] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European journal of operational research*, 1992.
[32] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*. Siam, 1994, vol. 43.
[33] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Berkeley Symposium on Mathematical Statistics and Probability*, 1951.
[34] R. M. Lewis and V. Torczon, "Pattern search methods for linearly constrained minimization," *SIAM Journal on Optimization*, 2000.
[35] C. Casetti, C. F. Chiasserini, N. Molner, J. Martin-Perez, T. Deiss, C.-T. Phan, F. Messaoudi, G. Landi, and J. B. Baranzano, "Arbitration among vertical services," in *IEEE PIMRC 5G Cell-Less Nets Workshop*, 2018.

Satyam

Francesco Malandrino received his M.S. and Ph.D. degrees from Politecnico di Torino, Italy, in 2008 and 2012 respectively. He is now a tenured researcher at the Institute of Electronics, Computer and Communication Engineering of the National Research Council of Italy (CNR-IEIIT), headquartered in Torino, Italy. Prior to his current appointment, he has been an assistant professor and a research fellow at Politecnco di Torino, a Fibonacci Fellow at the Hebrew University of Jerusalem and a research fellow at Trinity College, Dublin. His research interests include the architecture and management of wireless, cellular, and vehicular networks.

Carla-Fabiana Chiasserini (M'98, SM'09, F'18) graduated from the University of Florence in 1996 and received her Ph.D. from Politecnico di Torino, Italy, in 2000. She worked as a visiting researcher at UCSD in 1998–2003, and as a Visiting Professor at Monash University in 2012 and 2016. She is currently an Associate Professor with the Department of Electronic Engineering and Telecommunications at Politecnico di Torino. Her research interests include architectures, protocols, and performance analysis of wireless networks. Dr. Chiasserini has published over 300 papers in prestigious journals and leading international conferences.

Swades