This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301506

Double Deep Q-Learning-based Path Selection and Service Placement for Latency-Sensitive Beyond 5G Applications

Masoud Shokrnezhad[1], Tarik Taleb[1], and Patrizio Dazzi[2]
1 Oulu University, Oulu, Finland
2 University of Pisa, Pisa, Italy
{masoud.shokrnezhad; tarik.taleb}@oulu.fi, patrizio.dazzi@unipi.it

***Abstract*— Nowadays, as the need for capacity continues to grow, entirely novel services are emerging. A solid cloud-network integrated infrastructure is necessary to supply these services in a real-time responsive, and scalable way. Due to their diverse characteristics and limited capacity, communication and computing resources must be collaboratively managed to unleash their full potential. Although several innovative methods have been proposed to orchestrate the resources, most ignored network resources or relaxed the network as a simple graph, focusing only on cloud resources. This paper fills the gap by studying the joint problem of communication and computing resource allocation, dubbed CCRA, including function placement and assignment, traffic prioritization, and path selection considering capacity constraints and quality requirements, to minimize total cost. We formulate the problem as a non-linear programming model and propose two approaches, dubbed B&B-CCRA and WF-CCRA, based on the Branch & Bound and Water-Filling algorithms to solve it when the system is fully known. Then, for partially known systems, a Double Deep Q-Learning (DDQL) architecture is designed. Numerical simulations show that B&B-CCRA optimally solves the problem, whereas WF-CCRA delivers near-optimal solutions in a substantially shorter time. Furthermore, it is demonstrated that DDQL-CCRA obtains near-optimal solutions in the absence of request-specific information.**

***Index Terms*—Beyond 5G, 6G, Computing First Networking, Cloud-Network Integration, Cloud Network Fabric, Resource Allocation, Path Selection, Traffic Prioritization, VNF Placement, Optimization Theory, Reinforcement Learning, and Q-learning.**

I. Introduction

Nowadays, an increase in data flow has resulted in a 1000-fold increase in network capacity, which is the primary driver of network evolution. While this demand for capacity will continue to grow, the Internet of Everything is forging a paradigm shift to new-born perceptions, bringing a range of novel services with rigorous deterministic criteria, such as connected robotics, smart healthcare, autonomous transportation, and extended reality [1]. These services will be provisioned by establishing functional components, Virtual Network Functions (VNFs), which will generate and consume vast amounts of data that must be processed in real-time to ensure service responsiveness and scalability.

In these circumstances, a distributed cloud architecture is essential [2], which could be implemented via a solid cloud-network integrated infrastructure built of distinct domains in Beyond 5G (B5G) [3]. These domains can be distinguished by the technology employed, including radio access, transport, and core networks, as well as edge, access, aggregation, regional, and central clouds. Moreover, these resources can be virtualized using technologies such as Network Function Virtualization (NFV), which enables the construction of separate virtual entities on top of this physical infrastructure [4], [5]. Since distributed cloud and network domains would be diverse in terms of characteristics but limited in terms of capability, communication and computing resources should be jointly allocated, prioritized, and scheduled to ensure maximum Quality of Service (QoS) satisfaction while maximizing resource sharing and maintaining a deterministic system state, resulting in energy savings as one of the most significant examples of cost minimization objectives [6].

The joint problem of resource allocation in cloud-network integrated infrastructures has been extensively studied in the literature. Emu *et al.* [7] analyzed the VNF placement problem as an Integer Linear Programming (ILP) model that guarantees low End-to-End (E2E) latency while preserving QoS requirements by not exceeding an acceptable latency violation limit. They proposed an approach based on neural networks and demonstrated that it can result in near-optimal solutions in a timely way. Vasilakos *et al.* [8] examined the same problem and proposed a hierarchical Reinforcement Learning (RL) method with local prediction modules as well as a global learning component. They demonstrated that their method significantly outperforms conventional approaches. Sami *et al.* [9] investigated a similar topic to minimize the cost of allocations, and a Markov decision process design was provided. They claimed that the proposed method provides efficient placements. Performing cost-effective services was also investigated by Liu *et al.* [10] and He *et al.* [11]. In the former, the authors considered the cost of computing and networking resources as well as the cost of using VNFs and proposed a heuristic algorithm, whereas, in the latter, they considered latency as a cost and proposed a Deep Reinforcement Learning (DRL) solution to the problem. Iwamoto *et al.* [12] investigated the problem of scheduling VNF migrations in order to optimize the QoS degradation of all traffic flows and proposed a stochastic method on the basis of the load degree of VNF instances.

TABLE I
LITERATURE REVIEW.

Paper	Objective	Constraints							Solution Approach		
		Traffic Priority	Routing	Network Capacity	Network Latency: Device	Network Latency: Link	Compute Capacity	Compute Latency	Optimal	Heuristic	Learning
[13]	max profit	✓	✓	✓			✓			✓	
[10]	min cost			✓		✓	✓			✓	
[11]	max profit - cost					✓	✓	✓			✓
[14]	min energy		✓			✓	✓	✓			✓
[15]	min cost		✓	✓			✓			✓	
[12]	max fairness					✓	✓	✓		✓	
[16]	max profit - cost		✓	✓			✓			✓	
[7]	min latency			✓		✓	✓				✓
[8]	min latency						✓	✓			✓
[9]	min cost	✓					✓				✓
[17]	max rate		✓	✓			✓			✓	
[18]	min cost			✓		✓	✓	✓	✓		
[19]	max rate			✓		✓	✓	✓		✓	
[20]	min latency + cost					✓	✓		✓	✓	
[21]	min latency		✓			✓	✓	✓		✓	
this work	min cost	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Although innovative techniques for addressing computing resource restrictions have been proposed by the above-mentioned authors, the network is solely considered as a pipeline in their studies, with no cognitive ability to the cloud domains. Nevertheless, there are additional studies in the literature that have been concentrating on communication and computing resources jointly. Kuo *et al.* [17] studied the joint problem of VNF placement and path selection in order to better utilize the network resources, and a heuristic approach was proposed to tackle it. Mada *et al.* [18] and Zhang *et al.* [19] addressed the problem of VNF placement with the objective of maximizing the sum rate of accepted requests. Mada *et al.* solved the problem by using an optimization solver, and Zhang *et al.* adopted a heuristic strategy. Yuan, Tang and You [20] formulated the latency-optimal placement of functions as an ILP problem and proposed a genetic meta-heuristic algorithm to solve it. Gao *et al.* [21] focused on the VNF placement and scheduling to reduce the cost of computing resources by proposing a latency-aware heuristic algorithm. Minimizing the cost of allocations was also investigated by Miyamura *et al.* [15] and Yang *et al.* [16]. They took into account traffic routing constraints and proposed heuristic approaches to address the problem. By considering energy consumption as the most significant cost associated with networking and computing resources, Xuan *et al.* [14] addressed the same problem by proposing an algorithm based on a multi-agent DRL and a self-adaptation division strategy. Nguyen *et al.* [13] investigated the problem of VNF placement, where requests are weighted according to their priority and the goal is to maximize the total weight of services accepted for deployment on the infrastructure.

The methods presented in the cited studies are effective for resolving the resource allocation problem. However, such approaches cannot be utilized in B5G systems. Due to the stringent QoS requirements in the delay-reliability-rate space [22], the large number of concurrent services and requests, and the ever-changing dynamics of both infrastructure and end-user service usage behavior in terms of time and space, every detail of communication and computing resources must be determined and controlled in order to realize a deterministic B5G system [3]. In some studies, latency-related limitations and requirements were simply ignored [17], [15], [16], [13]. Despite the fact that delay is addressed in the other studies mentioned, they simplified it to be a connection feature, and queuing delay in network devices is completely eliminated. Furthermore, path selection is disregarded in some studies [18], [19], [20], and cost optimization is overlooked in others [19], [20].

This paper fills in the gap in the current literature by investigating the joint problem of allocating communication and computing resources, including VNF placement and assignment, traffic prioritization, and path selection. The problem is faced while taking into account capacity constraints and link and queuing delays, to minimize overall cost. As an extension of the work presented in [23], the following are the primary contributions of this research:

- Formulating the joint resource allocation problem of the cloud-network integrated infrastructure as a Mixed Integer Non-Linear Programming (MINLP) problem.
- Proposing a method based on the Branch & Bound (B&B) algorithm to discover the optimal solution of the problem, and devising a heuristic approach based on the Water-Filling (WF) algorithm in order to identify near-optimal solutions to the problem. When the system is fully known, both techniques can be applied to solve the problem.
- Developing an architecture based on the Double Deep Q-learning (DDQL) technique comprising agent design, training procedure, and decision-making strategy for allocating resources when the system is only partially known, i.e., there is no prior knowledge about the requests' requirements.

The reminder of this paper is organized as follows. Section II introduces the system model. The resource allocation problem is formulated in Section III. Next, the B&B and heuristic approaches are presented in Section IV. Section V presents a DDQL-based resource allocation architecture. Finally, numerical results are illustrated and analyzed in Section VI, followed by concluding remarks in Section VII.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301506

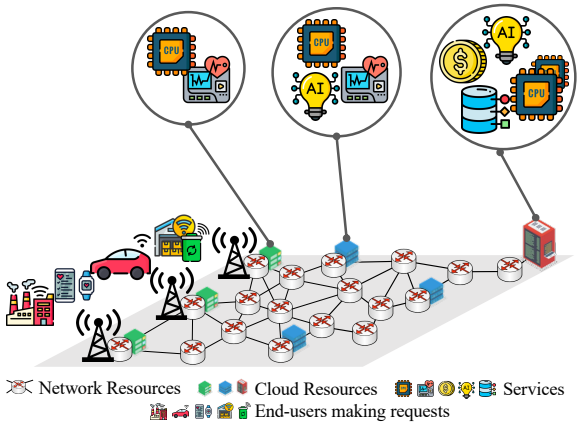


Fig. 1. The envisioned system model.

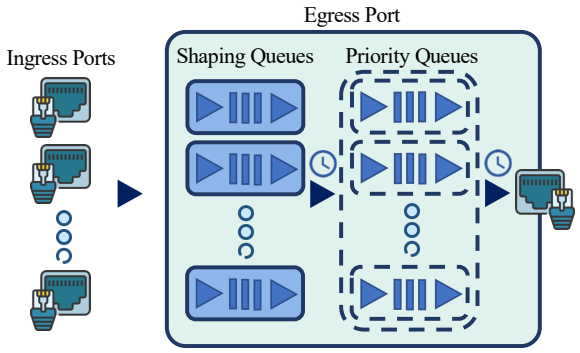


Fig. 2. ATS switch model.

II. System Model

In the following, we describe the main components of the system envisioned in this paper. As depicted in Fig. 1, the system consists of an infrastructure (integrated networking and computing resources), services running on computing resources, and end-user requests that must be connected to the services via networking resources. The parameters defined in this section are summarized in Table II.

A. Infrastructure Model

The considered infrastructure is composed of the edge (non-radio side) and core network domains consisting of $\mathcal{V}$ nodes, $\mathcal{L}$ links, and $\mathcal{P}$ paths denoted by $\mathcal{G} = \langle \boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{L}}, \boldsymbol{\mathcal{P}} \rangle$. $\boldsymbol{\mathcal{V}} = \{v | v \in \{1, 2, ..., \mathcal{V}\}\}$ is the set of nodes. $\boldsymbol{\mathcal{L}} \subset \{l : (v, v') | v, v' \in \boldsymbol{\mathcal{V}}\}$ indicates the set of links, where the bandwidth of link l is constrained by $\widehat{B_l}$, and it costs Ξ_l per capacity unit. Although a variety of factors (distance, technology, redundancy, accessibility, etc.) contribute to this cost as a capital expenditure, the energy used by network devices to process the traffic carried by this link is one of the significant operating expenses affecting this cost and must be precisely addressed in order to realize future networks [24]. $\boldsymbol{\mathcal{P}} = \{p : (\vdash_p, \dashv_p) | p \subset \boldsymbol{\mathcal{L}}\}$ denotes the set of all paths in the network, where $\vdash_p$ and $\dashv_p$ are the head and tail nodes of path p, and $\delta_{p,l}$ is a binary constant equal to 1 if path p contains link l. It should be noted that all paths are directed vectors of nodes with no loops.

Each node in the network is an IEEE 802.1 Time-Sensitive Networking (TSN) device comprising an IEEE 802.1 Qcr Asynchronous Traffic Shaper (ATS) at each egress port. As depicted in Fig. 2, An ATS uses a two-level queuing model [25]: 1) an array of shaped queues, each associated with a priority level and an ingress port, and 2) one queue per priority level. Each priority queue combines the output of all shaped queues with the same priority level. All queues implement the First-In-First-Out (FIFO) strategy. The next packet for transmission is identified by comparison of 1) the associated priority levels, and 2) the eligibility times of the Head-of-Queue (HoQ) packets. This could be accomplished, for instance, using comparator networks or linear iteration over all queues/HoQ packets while the transmission of a previous packet is in progress.

We consider $\boldsymbol{\mathcal{K}} = \{k | k \in \{1, 2, ..., \mathcal{K}\}\}$ as the set of priority levels and assume that k_r is the assigned priority of the traffic associated with request r, and the size of the queues for priority level k is the same and equal to $\widehat{\mathcal{T}_k}$. Note that lower levels have higher priorities. Moreover, each node v is equipped with computing resources as one of the prospective hosts to deploy service VNFs and limited to a predefined capacity threshold $\widehat{\zeta_v}$, which costs Ψ_v per capacity unit. Ψ_v is an increasing function of the energy consumed by various components of computing nodes (such as the processor, memory, and storage) to process requests and required by cooling systems to maintain appropriate temperatures. This cost is one of the most significant obstacles that must be overcome to make future applications feasible [26].

It is worth mentioning that the network is divided into several tiers, with nodes distributed across them so that the edge nodes (the entry nodes of requests) are located in tier 0. The higher the tier index, the greater the capacity of the associated nodes, and the lower their cost. In other words, the nodes closest to end-users (or to the nodes that serve as entry points - e.g., far edge node or in-network computing nodes [27]) are provisioned with high-cost, limited-capacity computing facilities, while low-cost, high-capacity depots are deployed in the core.

B. Service Model

The set of services is dubbed $\boldsymbol{\mathcal{S}} = \{s | s \in \{1, 2, ..., \mathcal{S}\}\}$, where $\mathcal{S}$ indicates the number of services. If an end-user requests a service, its VNF has to be replicated in the network-embedded computing resources. Each VNF is empowered to serve more than one request, and $\widehat{\mathcal{C}_s}$ indicates the maximum capacity of each VNF of service s.

C. Request Model

The set of requests asking for services is represented by $\boldsymbol{\mathcal{R}} = \{r | r \in \{1, 2, ..., \mathcal{R}\}\}$, where $\mathcal{R}$ is the number of requests. Each request r arrives in the network through node v_r (one of the nodes equipped with a radio access base station) and intends service s_r, specifying its minimum necessitated service capacity, network bandwidth, and maximum tolerable delay, indicated by $\widetilde{\mathcal{C}_r}$, $\widetilde{\mathcal{B}_r}$, and $\widetilde{\mathcal{D}_r}$, respectively. In addition, $\widetilde{\mathcal{T}_r}$ and $\widetilde{\mathcal{H}_r}$, denoting the burstiness of traffic and the largest

TABLE II
PARAMETER/VARIABLE* DESCRIPTION.

#	P/V*	Description
Section II	$\boldsymbol{\mathcal{V}}$	set of nodes; $\lvert\boldsymbol{\mathcal{V}}\rvert = \mathcal{V}$
	$\boldsymbol{\mathcal{L}}$	set of links; $\lvert\boldsymbol{\mathcal{L}}\rvert = \mathcal{L}$
	$\widehat{B_l}$	bandwidth of link l
	Ξ_l	cost of link l per capacity unit
	$\boldsymbol{\mathcal{P}}$	set of paths; $\lvert\boldsymbol{\mathcal{P}}\rvert = \mathcal{P}$
	$\vdash_p$	head node of path p
	$\dashv_p$	tail node of path p
	$\delta_{p,l}$	1 if path p contains link l, otherwise 0 (binary input)
	$\boldsymbol{\mathcal{K}}$	set of priority levels; $\lvert\boldsymbol{\mathcal{K}}\rvert = \mathcal{K}$
	k_r	priority of request r
	$\widehat{\mathcal{T}_k}$	queue size of priority k
	$\widehat{\zeta_v}$	computing capacity of node v
	Ψ_v	cost of node v per capacity unit
	$\boldsymbol{\mathcal{S}}$	set of services; $\lvert\boldsymbol{\mathcal{S}}\rvert = \mathcal{S}$
	$\widehat{\mathcal{C}_s}$	capacity of each VNF of service s
	$\boldsymbol{\mathcal{R}}$	set of requests; $\lvert\boldsymbol{\mathcal{R}}\rvert = \mathcal{R}$
	v_r	point of arrival of request r
	s_r	target service of request r
	$\widetilde{\mathcal{C}_r}$	minimum required service capacity of request r
	$\widetilde{\mathcal{B}_r}$	minimum required network bandwidth of request r
	$\widetilde{\mathcal{D}_r}$	maximum tolerable delay of request r
	$\widetilde{\mathcal{T}_r}$	traffic burstiness of request r
	$\widetilde{\mathcal{H}_r}$	largest packet size of request r
Section III	$g_{r,v}$	assigned node of request r (binary var.)
	$z_{s,v}$	assigned node of service s (binary var.)
	$\varrho_{r,k}$	assigned priority of request r (binary var.)
	$\overrightarrow{f_{r,p,k}}$	inquiry path and priority of request r (binary var.)
	$\overleftarrow{f_{r,p,k}}$	response path and priority of request r (binary var.)
	$D_{r,k,l}$	delay of request r on priority k and link l
	D_{r,s_r}	computing delay of request r
	D_r	E2E delay of request r
	$\boldsymbol{\mathcal{R}_1}$	requests with a priority higher than or equal to request r
	$\boldsymbol{\mathcal{R}_2}$	requests with a priority lower than request r
	$\boldsymbol{\mathcal{R}_3}$	requests with a priority higher than request r

packet size for request r, are also assumed to be known a priori. Utilizing historical data, along with predictive data analytics methods, is one of the viable options for obtaining such accurate and realistic statistical estimates of traffic.

III. PROBLEM DEFINITION

This section describes the joint problem of VNF placement and assignment, traffic prioritization, and path selection. In what follows, the constraints and objective function are formulated as a MINLP problem and the problem is stated at the end of the section. The variables and parameters defined in this section are summarized in Table II.

A. VNF Placement and Assignment Constraints

To arrange VNFs, each request must be first assigned a single node to serve as its service location (C1). This assignment is acceptable if the assigned node hosts a VNF for the requested service (C2). When the requests of a specific service are assigned to a particular node, they will be handled by a shared VNF. C3 ensures that the total service capacity required by these requests does not surpass the VNF's capacity. Additionally, C4 guarantees that the computing capacity of a node is not exceeded by the VNFs placed on it. Without these two constraints, both VNFs and nodes are at risk of becoming overloaded, leading to the potential termination of VNFs and congestion of requests. Such a scenario would significantly decrease the system's reliability and availability. The problem formulation becomes as follows:

$$\sum\nolimits_{\boldsymbol{\mathcal{V}}} g_{r,v} = 1, \forall r \in \boldsymbol{\mathcal{R}}, \tag{C1}$$

$$g_{r,v} \leq z_{s_r,v}, \forall r, v \in \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{V}}, \tag{C2}$$

$$\sum\nolimits_{\{r|r\in\boldsymbol{\mathcal{R}}\wedge s_r=s\}} \widetilde{\mathcal{C}_r} g_{r,v} \leq \widehat{\mathcal{C}_s}, \forall v, s \in \boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{S}}, \tag{C3}$$

$$\sum\nolimits_{\boldsymbol{\mathcal{S}}} \widehat{\mathcal{C}_s} z_{s,v} \leq \widehat{\zeta_v}, \forall v \in \boldsymbol{\mathcal{V}}, \tag{C4}$$

where $g_{r,v}$ and $z_{s,v}$ are binary variables. $g_{r,v}$ is 1 if node v is selected as the service node of request r, and $z_{s,v}$ is 1 if service s is replicated on node v.

B. Traffic Prioritization and Path Selection Constraints

To direct traffic, we must first ensure that each request is assigned to exactly one priority level (C5). Then, each request's (request and reply) paths are determined (C6 and C7). For each request, a single inquiry path is chosen that starts at the request's entry node and ends at the request's VNF node. The response path follows the same logic but in reverse order. The following two constraints guarantee that the two paths are chosen on the priority level assigned to each request (C8 and C9). Finally, the constraints maintaining the maximum capacity of links and queues are enforced (C10 and C11). With C10, the sum of the required bandwidth for all requests whose inquiry or response path, or both, contains link l is guaranteed to be less than or equal to the link's capacity. In C11, the capacity of queues is guaranteed in the same way for each link and each priority level. The set includes:

$$\sum\nolimits_{\boldsymbol{\mathcal{K}}} \varrho_{r,k} = 1, \forall r \in \boldsymbol{\mathcal{R}}, \tag{C5}$$

$$\sum\nolimits_{\{p|p\in\boldsymbol{\mathcal{P}}\wedge\vdash_p=v_r\wedge\dashv_p=v\},\boldsymbol{\mathcal{K}}} \overrightarrow{f_{r,p,k}} = g_{r,v}, \forall r, v \in \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{V}}, \tag{C6}$$

$$\sum\nolimits_{\{p|p\in\boldsymbol{\mathcal{P}}\wedge\vdash_p=v\wedge\dashv_p=v_r\},\boldsymbol{\mathcal{K}}} \overleftarrow{f_{r,p,k}} = g_{r,v}, \forall r, v \in \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{V}}, \tag{C7}$$

$$\sum\nolimits_{\boldsymbol{\mathcal{P}}} \overrightarrow{f_{r,p,k}} = \varrho_{r,k}, \forall r, k \in \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{K}}, \tag{C8}$$

$$\sum\nolimits_{\boldsymbol{\mathcal{P}}} \overleftarrow{f_{r,p,k}} = \varrho_{r,k}, \forall r, k \in \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{K}}, \tag{C9}$$

$$\sum\nolimits_{\boldsymbol{\mathcal{R}}} \widetilde{\mathcal{B}_r} \sum\nolimits_{\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{K}}} \delta_{p,l} \cdot (\overrightarrow{f_{r,p,k}} + \overleftarrow{f_{r,p,k}}) \leq \widehat{B_l}, \forall l \in \boldsymbol{\mathcal{L}}, \tag{C10}$$

$$\sum\nolimits_{\boldsymbol{\mathcal{R}}} \widetilde{\mathcal{T}_r} \sum\nolimits_{\boldsymbol{\mathcal{P}}} \delta_{p,l} \cdot (\overrightarrow{f_{r,p,k}} + \overleftarrow{f_{r,p,k}}) \leq \widehat{\mathcal{T}_k}, \forall k, l \in \boldsymbol{\mathcal{K}}, \boldsymbol{\mathcal{L}}, \tag{C11}$$

where $\varrho_{r,k}$ is a binary variable that equals 1 only when the priority level assigned to request r is k, and $\overrightarrow{f_{r,p,k}}$ and $\overleftarrow{f_{r,p,k}}$ are binary variables that reflect the inquiry and response paths for request r on priority level k, respectively.

C. Delay Constraints

To guarantee the minimum delay requirement of requests, the following settings should be adhered:

$$D_{r,s_r}\widetilde{\mathcal{C}_r} = \widetilde{\mathcal{H}_r}, \forall r \in \boldsymbol{\mathcal{R}}, \tag{C12}$$

$$D_{r,k,l} = \frac{\sum_{\boldsymbol{\mathcal{R}_1}} \widetilde{\mathcal{T}_{r'}} + \bigwedge_{\boldsymbol{\mathcal{R}_2}} \widetilde{\mathcal{H}_{r'}}}{\widehat{B_l} - \sum_{\boldsymbol{\mathcal{R}_3}} \widetilde{\mathcal{B}_{r'}}} + \frac{\widetilde{\mathcal{H}_r}}{\widehat{B_l}}, \ \forall r, k, l \in \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{K}}, \boldsymbol{\mathcal{L}}, \tag{C13}$$

$$D_r = \sum\nolimits_{\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{L}},\boldsymbol{\mathcal{K}}} D_{r,k,l}\delta_{p,l} \cdot (\overrightarrow{f_{r,p,k}} + \overleftarrow{f_{r,p,k}}) + D_{r,s_r}, \forall r \in \boldsymbol{\mathcal{R}} \tag{C14}$$

$$D_r \leq \widetilde{\mathcal{D}_r}, \forall r \in \boldsymbol{\mathcal{R}}, \tag{C15}$$

where $D_{r,k,l}$, D_{r,s_r} and D_r are continuous variables denoting the delay experienced by a given flow of request r associated

with priority level k passing through ATS-based link l [25], its computing delay, and the corresponding E2E delay calculated as the sum of the delays on the links that comprise both paths of the request and its computing delay. Besides, $\bigwedge$ is a function which returns the max value over the given set, $\boldsymbol{\mathcal{R}_1}$ equals $\{r'|r' \in \boldsymbol{\mathcal{R}} \wedge k_{r'} \leq k \wedge \delta_{p,l}(\overrightarrow{f_{r',p,k_{r'}}} + \overleftarrow{f_{r',p,k_{r'}}}) > 0\}$, $\boldsymbol{\mathcal{R}_2}$ represents $\{r'|r' \in \boldsymbol{\mathcal{R}} \wedge k_{r'} > k \wedge \delta_{p,l}(\overrightarrow{f_{r',p,k_{r'}}} + \overleftarrow{f_{r',p,k_{r'}}}) > 0\}$, and $\boldsymbol{\mathcal{R}_3}$ denotes $\{r'|r' \in \boldsymbol{\mathcal{R}} \wedge k_{r'} < k \wedge \delta_{p,l}(\overrightarrow{f_{r',p,k_{r'}}} + \overleftarrow{f_{r',p,k_{r'}}}) > 0\}$. These sets represent requests that share the same link as request r, whereas $\boldsymbol{\mathcal{R}1}$ includes requests with a higher or equal priority, $\boldsymbol{\mathcal{R}_2}$ contains requests with a lower priority, and $\boldsymbol{\mathcal{R}_3}$ shares requests with a higher priority.

D. Objective Function

The objective function is to minimize the total cost of allocated computing nodes and network links, that is:

$$\sum\nolimits_{\boldsymbol{\mathcal{R}},\boldsymbol{\mathcal{V}}} \Psi_v g_{r,v} + \sum\nolimits_{\boldsymbol{\mathcal{R}},\boldsymbol{\mathcal{L}}} \Xi_l \sum\nolimits_{\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{K}}} \delta_{p,l}(\overrightarrow{f_{r,p,k}} + \overleftarrow{f_{r,p,k}}), \quad \text{(OF)}$$

As mentioned in Section II, this cost is directly related to the energy consumption of networking and computing elements, and its reduction is a crucial open challenge that must be carefully addressed to enable B5G systems [28], [29], [30], [31].

E. Problem

Considering the constraints and objective function, the problem of Communication and Computing Resource Allocation (CCRA) is:

$$\text{CCRA: } \textit{min} \text{ OF } \textit{s.t.} \text{ C1 - C15.} \quad (1)$$

IV. Fully-Informed Methods

In this section, the system is assumed to be fully known, i.e., the list of services and their characteristics are available, and the current state of the network and cloud resources as well as requests and their requirements are being monitored and collected on a regular basis. This could be the case of an industrial environment whereby tasks and communications among robots and devices are pre-planned [32], [33], [34]. Under such scenarios, the following section proposes two methods, B&B-CCRA and WF-CCRA, to solve the problem specified by (1). Clearly, an efficient strategy for implementing these methods is to centralize their development as system orchestrator components. Then, when end-users request access to the services, the methods can be executed, and the resulting decisions can be applied to the network and cloud resources using Software-Defined Networking (SDN) and NFV technologies.

A. B&B-CCRA

Suppose that C1 and C5 - C15 are eliminated from (1) and only C2 - C4 affect the problem. Given this, the problem can be reformulated as minimizing the cost of assigned nodes within the capacity constraints of nodes and VNFs, that is $min \sum_{\boldsymbol{\mathcal{R}},\boldsymbol{\mathcal{V}}} \Psi_v g_{r,v}$ *s.t.* C2 - C4. If a new parameter denoted $\Psi'_v = \mathcal{M} - \Psi_v$, where $\mathcal{M}$ is a big positive number, is defined and substituted for Ψ_v, the relaxed problem can be rewritten equivalently as $max \sum_{\boldsymbol{\mathcal{R}},\boldsymbol{\mathcal{V}}} \Psi'_v g_{r,v}$ *s.t.* C2 - C4, which is the Multi-Dimensional Knapsack (MDK) problem with at least $\mathcal{S}$ items and $\mathcal{V}$ knapsacks. Since the MDK problem is NP-hard [35] and a relaxed version of our problem is as hard as this problem, it is proved that our problem is also NP-hard, and finding its optimal solution in polynomial time is mathematically intractable. One potential strategy for addressing such a problem is to restrict its solution space using the B&B algorithm, which relaxes and solves the problem to obtain lower bounds, and then improves the bounds using mathematical cuts to reach acceptable solutions. The method is described in Algorithm 1. In this algorithm, the solution space is discovered by maintaining an unexplored candidate list $\boldsymbol{\mathcal{N}} = \{N_t|t \geq 1\}$, where each node N_t contains a problem, denoted by Φ_t, and t is the iteration number. This list only contains N_1, the root candidate, at the beginning with the primary problem to be solved. To reduce its enormous computational complexity, instead of directly applying the B&B algorithm to CCRA, we consider its integer linear transformation as the problem of N_1.

CCRA comprises non-linear constraints C13 and C14. To linearize C13, the summations and max function with variable boundaries should be converted to a linear form. A simple, effective technique is to replace each term with an approximated upper bound. Since the aggregated traffic burstiness is bounded by $\widehat{\mathcal{T}_k}$ for each priority level k in C11, $\sum_{\boldsymbol{\mathcal{R}_1}} \widehat{\mathcal{T}_{r'}}$ can be replaced by the sum of this bound for all priority levels greater than or equal to k, that is $\sum_{\{k'|k' \leq k\}} \widehat{\mathcal{T}_{k'}}$. In a similar way, we define a new constraint (C13′) for the aggregated bandwidth allowed on priority level k over link l, dubbed $\widehat{f_{l,k}}$, and replace the sum of allocated bandwidths with $\sum_{\{k'|k'<k\}} \widehat{f_{l,k'}}$. Besides, the maximum packet size for a particular subset of requests can be replaced by the maximum permitted packet size in the network, denoted by $\widehat{\mathcal{H}}$. Therefore, the followings define the linear transformation of C13:

$$\sum\nolimits_{\boldsymbol{\mathcal{R}}} \widehat{\mathcal{B}_r} \sum\nolimits_{\boldsymbol{\mathcal{P}}} \delta_{p,l}(\overrightarrow{f_{r,p,k}} + \overleftarrow{f_{r,p,k}}) \leq \widehat{f_{l,k}}, \forall k \in \boldsymbol{\mathcal{K}}, \forall l \in \boldsymbol{\mathcal{L}}, \quad \text{(C13}')$$

$$\widehat{D_{k,l}} = \frac{\sum_{\boldsymbol{\mathcal{K}_1}} \widehat{\mathcal{T}_{k'}} + \widehat{\mathcal{H}}}{\widehat{B_l} - \sum_{\boldsymbol{\mathcal{K}_2}} \widehat{f_{l,k'}}} + \frac{\widehat{\mathcal{H}}}{\widehat{B_l}}, \forall k \in \boldsymbol{\mathcal{K}}, \forall l \in \boldsymbol{\mathcal{L}}, \quad \text{(C13}'')$$

where $\widehat{D_{k,l}}$ is the delay upper bound on link l with priority level k, $\boldsymbol{\mathcal{K}_1}$ is $\{k'|k' \leq k\}$, and $\boldsymbol{\mathcal{K}_2}$ is $\{k'|k' < k\}$. Since D_{r,s_r} is linear, C14 can be linearized by substituting the actual delay for the upper bound derived in C13″, and the new constraint for E2E delay is:

$$D_r = \sum\nolimits_{\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{L}},\boldsymbol{\mathcal{K}}} \widehat{D_{k,l}} \delta_{p,l}(\overrightarrow{f_{r,p,k}} + \overleftarrow{f_{r,p,k}}) + D_{r,s_r}, \forall r \in \boldsymbol{\mathcal{R}}. \quad \text{(C14}')$$

Given this, the linear transformation of CCRA, dubbed LiCCRA, is as follows:

$$\text{LiCCRA: } \textit{min} \text{ OF } \textit{s.t.} \text{ C1 - C12, C13}', \text{ C13}'', \text{ C14}', \text{ C15.} \quad (2)$$

Now, with LiCCRA as Φ_1, each iteration of the B&B algorithm begins with the selection and removal of a candidate from the unexplored list. Then, the problem of this candidate is naturally relaxed and solved, i.e., all the integer variables in the set $\{0, 1\}$ are replaced with their continuous equivalents restricted by the box constraint $[0, 1]$, and the relaxed problem

Algorithm 1 B&B-CCRA.

$\boldsymbol{\mathcal{N}} \leftarrow \{N_1\}$, $\eta^\star \leftarrow +\infty$, $t \leftarrow 0$
while $\boldsymbol{\mathcal{N}}$ is not empty **do**
  $t \leftarrow t+1$
  $N_t \leftarrow$ selects a node form $\boldsymbol{\mathcal{N}}$
  $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \backslash \{N_t\}$
  $(\boldsymbol{\mu}_t^\star, \boldsymbol{\lambda}_t^\star), \phi_t^\star \leftarrow$ solve the relaxed problem of Φ_t
  if $\boldsymbol{\mu}_t^\star$ is integer for all elements **then**
    $\eta^\star \leftarrow min(\eta^\star, \phi_t^\star)$
  else if $\phi_t^\star < \eta^\star$ is preserved **then**
    $N_t^1, N_t^2 \leftarrow$ two children of N_t
    $\boldsymbol{\mathcal{N}} \leftarrow \boldsymbol{\mathcal{N}} \cup \{N_t^1, N_t^2\}$

is solved using a Linear Programming (LP) solver to obtain the solution of the relaxed problem $(\boldsymbol{\mu}_t^\star, \boldsymbol{\lambda}_t^\star)$ and the optimal objective value $\phi_t^\star$, where $\boldsymbol{\mu}$ is the relaxed integer variables set, and $\boldsymbol{\lambda}$ is the set of continuous variables. Next, if all relaxed variables have integer values, the obtained objective in this iteration is considered to update the best explored integer solution. Otherwise, a variable index j is selected such that $\boldsymbol{\mu}_t^\star[j]$ is fractional, and the feasible constraints set π_t is divided into two parts as $\pi_t^1 = \pi_t \cap \{\boldsymbol{\mu}_t[j] \leq \lfloor \boldsymbol{\mu}_t^\star[j] \rfloor\}$ and $\pi_t^2 = \pi_t \cap \{\boldsymbol{\mu}_t[j] \geq \lceil \boldsymbol{\mu}_t^\star[j] \rceil\}$. Then, two problems are formed as $\Phi_t^1 = min$ OF *s.t.* π_t^1 and $\Phi_t^2 = min$ OF *s.t.* π_t^2. Now, two child nodes N_t^1 and N_t^2, whose problems are Φ_t^1 and Φ_t^2 respectively, are put into the unexplored list. The B&B algorithm is iterated until $\boldsymbol{\mathcal{N}}$ is empty.

Alternatively, we can run this algorithm until a desired solving time is reached or an acceptable objective value is acquired. The key advantage of this algorithm is that it produces at least a lower bound even when the solving time is limited. As a result, it may be used to establish baselines allowing for the evaluation of alternative approaches.

B. WF-CCRA

Since the B&B method searches the problem's solution space for the optimal solution, its complexity can grow up to the size of the solution space in the worst case [36]. Given that the size of the solution space in CCRA (or LiCCRA) for each request is $\mathcal{V}^2\mathcal{P}^2\mathcal{K}$ considering its integer variables, the problem's overall size is $\mathcal{R}!\mathcal{V}^2\mathcal{P}^2\mathcal{K}$, considering the number of permutations of $\mathcal{R}$ requests. Therefore, finding its optimal solution for large-scale instances using B&B is impractical in a timely manner, and the goal of this section is to devise an efficient approach based on the WF concept in order to identify near-optimal solutions for this problem.

The WF-CCRA method is elaborated in Algorithm 2. The first step is to initialize the vectors of parameters and variables used in (1) (or in (2)). Following that, two empty sets, $\boldsymbol{\mathcal{R}'}$ and $\boldsymbol{\Omega}$, are established. The former maintains the set of accepted requests, and the latter stores the feasible resource combinations for each request during its iteration. Now, the algorithm iterates through each request in $\boldsymbol{\mathcal{R}}$, starting with the one with the most stringent delay requirement, and keeps track of the feasible allocations of VNF, priority, as well as inquiry and response paths based on the constraints of (1) (or (2)). The final steps of each iteration are to choose the allocation with the lowest cost and fix it for the request, as well as to update remaining resources and the set of pending and accepted requests. When there is no pending request, the algorithm terminates.

Algorithm 2 WF-CCRA.

initialize variable and parameter vectors
$\boldsymbol{\mathcal{R}'} \leftarrow \{\}$, $\boldsymbol{\Omega} \leftarrow \{\}$
sort $\boldsymbol{\mathcal{R}}$ in ascending order according to $\widetilde{\mathcal{D}_r}$
while $\boldsymbol{\mathcal{R}}$ is not empty **do**
  for $v \in \boldsymbol{\mathcal{V}}$ **do**
    if $z_{s_r,v} == 1$ **and** $\widehat{\mathcal{C}_r} \leq \widehat{\mathcal{C}_{s_r}}$ on v **then**
      $g_{r,v} = 1$
    if $z_{s_r,v} \neq 1$ **and** $\widehat{\mathcal{C}_{s_r}} \leq \widehat{\zeta_v}$ **then**
      $z_{s_r,v} = 1$, $g_{r,v} = 1$
    else go to the next iteration
    for $k \in \boldsymbol{\mathcal{K}}$ **do**
      $\varrho_{r,k} = 1$
      for $p \in \boldsymbol{\mathcal{P}} \wedge \vdash_p = v_r \wedge \dashv_p = v$ **do**
        if $\widetilde{\mathcal{B}_r} \leq \widehat{\mathcal{B}_l}$ & $\widetilde{\mathcal{T}_r} \leq \widehat{\mathcal{T}_k}$ on $l \; \forall l \in \boldsymbol{\mathcal{L}} \wedge \delta_{p,l} = 1$ **then**
          $\overrightarrow{f_{r,p,k}} = 1$
          for $p' \in \boldsymbol{\mathcal{P}} \wedge \vdash_{p'} = v \wedge \dashv_{p'} = v_r$ **do**
            if $\widetilde{\mathcal{B}_r} \leq \widehat{\mathcal{B}_l}$ & $\widetilde{\mathcal{T}_r} \leq \widehat{\mathcal{T}_k}$ on $l \; \forall l \in \boldsymbol{\mathcal{L}} \wedge \delta_{p,l} = 1$ **then**
              $\overleftarrow{f_{r,p',k}} = 1$
              calculate $\underline{D_r}$ based on (C14) (or C14′)
              if $D_r \leq \widetilde{\mathcal{D}_r}$ **then**
                $\boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega} \cup \{(z_{s_r,v}, g_{r,v}, \varrho_{r,k}, \overrightarrow{f_{r,p,k}}, \overleftarrow{f_{r,p',k}})\}$
  fix assignments of $argmin_{\boldsymbol{\Omega}}$ OF for r
  update capacities, $\boldsymbol{\Omega} \leftarrow \{\}$, $\boldsymbol{\mathcal{R}} \leftarrow \boldsymbol{\mathcal{R}} / \{r\}$, $\boldsymbol{\mathcal{R}'} \leftarrow \boldsymbol{\mathcal{R}'} \cup \{r\}$

The complexity of the WF-CCRA algorithm is $O(\mathcal{R}\mathcal{V}\mathcal{K}\mathcal{P}^2)$. Although this approach is significantly more efficient than the B&B algorithm in terms of complexity (it can be executed within milliseconds), its complexity can be further reduced by restricting the number of valid paths between each pair of nodes to a fixed-size set of paths with the lowest costs or smallest number of links. In addition, despite the fact that this algorithm implements only one of the $\mathcal{R}!$ possible permutations (serving requests in descending order of their urgency) and it converges to a solution where the cost of allocating resources to each request is locally minimized, it is expected to provide efficient solutions in terms of accuracy as well. The reason is that since requests for the same service are of the same quality and requests for all services have stringent QoS requirements, the unique permutations do not vary significantly.

V. Partially-Informed Method

In the previous section, we assumed that the system is fully known. In this section, we consider a scenario wherein the system is only partially known, i.e., the state of the available network and cloud resources is tracked in real-time, and the list of services and their associated characteristics have been introduced in advance. However, end-users and the orchestrator do not exchange information pertaining to requests and their requirements. In this particular scenario, to solve the problem

stated in (1), we employ the DDQL technique, proposed by Google in the DeepMind project [37]. In what follows, The DDQL concept and its agent, which serves as the core building block of the learning logic, are briefly introduced. The design of the learning algorithm and the architecture of the DDQL-based resource allocation approach are then discussed, along with an analysis of various implementation strategies.

A. Double Deep Q-Learning Agent

RL is a technique wherein an agent is trained to tackle sequential decision problems through trial-and-error interactions with the environment. Q-learning is a widely used RL algorithm wherein the agent learns the value of each action, defined as the sum of future rewards associated with performing that action, and then follows the optimal policy, which is choosing the action with the highest value in each state.

According to Watkins and Dayan [38], one method for obtaining the optimal action-value function is to define a Bellman equation as a straightforward value iteration update using the weighted average of the old value and the new information, that is

$$Q(\theta_\tau, a_\tau) \; += \; \sigma[Y_\tau^Q - Q(\theta_\tau, a_\tau)], \tag{3}$$

where θ_τ and a_τ are the agent's state and action at time slot τ respectively, σ is a scalar step size, and Y_τ^Q is the target, defined by

$$Y_\tau^Q = \beta_{\tau+1} + \gamma \max_{a\in\boldsymbol{\mathcal{A}}} Q(\theta_{\tau+1}, a), \tag{4}$$

where $\beta_{\tau+1}$ is the reward at time slot $\tau+1$, $\gamma \in [0,1]$ is a discount factor that balances the importance of immediate and later rewards, and $\boldsymbol{\mathcal{A}}$ is the set of actions. Since most interesting problems are too large to discover all possible combinations of states and actions and learn all action-values, one potential alternative is to use a Deep Neural Network (DNN) to approximate the action-value function. In a Deep Q-Network (DQN), the state is given as the input and the Q function of all possible actions, denoted by $Q(\theta, .; \boldsymbol{\mathcal{W}})$, is generated as the output, where $\boldsymbol{\mathcal{W}}$ is the set of DNN parameters. The target of the DQN is as follows:

$$Y_\tau^{DQN} = \beta_{\tau+1} + \gamma \max_{a\in\boldsymbol{\mathcal{A}}} Q(\theta_{\tau+1}, a, \boldsymbol{\mathcal{W}}_\tau), \tag{5}$$

and the update function of $\boldsymbol{\mathcal{W}}$ is

$$\boldsymbol{\mathcal{W}}_{\tau+1} = \boldsymbol{\mathcal{W}}_\tau + \sigma[Y_\tau^{DQN} - Q(\theta_\tau, a_\tau; \boldsymbol{\mathcal{W}}_\tau)] \nabla_{\boldsymbol{\mathcal{W}}_\tau} Q(\theta_\tau, a_\tau; \boldsymbol{\mathcal{W}}_\tau). \tag{6}$$

To further enhance the efficiency of DQN, it is necessary to consider two additional improvements. The first is the use of an experience memory [39], wherein the observed transitions are stored in a memory bank, and the neural network is updated by randomly sampling from this pool. The authors demonstrated that the concept of experience memory significantly improves the DQN algorithm's performance. The second is to employ the concept of Double Deep Q-Learning (DDQL), introduced in [37]. In both standard Q-learning and DQNs, the max operator selects and evaluates actions using the same values (or the same Q). Consequently, overestimated values are more likely to be selected, resulting in overoptimistic value estimations. DDQL implements decoupled selection and evaluation processes. The following is the definition of the target in DDQL:

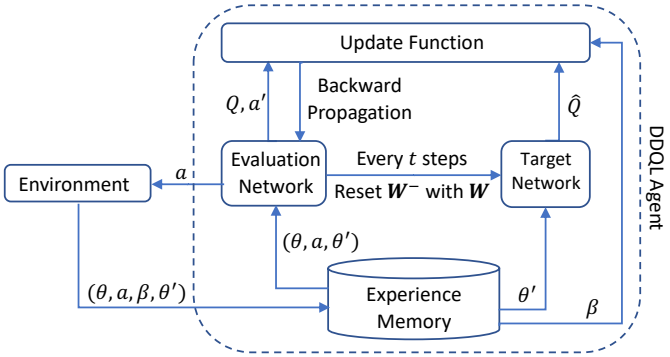


Fig. 3. DDQL Agent.

$$Y_\tau^{DDQL} = \beta_{\tau+1} + \gamma \, \widehat{Q}(\theta_{\tau+1}, a', \boldsymbol{\mathcal{W}}_\tau^-), \tag{7}$$

where $a' = argmax_{a\in\boldsymbol{\mathcal{A}}} Q(\theta_{\tau+1}, a, \boldsymbol{\mathcal{W}}_\tau)$, and the update function is

$$\boldsymbol{\mathcal{W}}_{\tau+1} = \boldsymbol{\mathcal{W}}_\tau + \sigma[Y_\tau^{DDQL} - Q(\theta_\tau, a_\tau; \boldsymbol{\mathcal{W}}_\tau)] \nabla_{\boldsymbol{\mathcal{W}}_\tau} Q(\theta_\tau, a_\tau; \boldsymbol{\mathcal{W}}_\tau). \tag{8}$$

In this model, $\boldsymbol{\mathcal{W}}$ is the set of weights for the main (or evaluation) Q and is updated in each step, whereas $\boldsymbol{\mathcal{W}}^-$ is for the target $\widehat{Q}$ and is replaced with the weights of the main network every t steps. In other words, $\widehat{Q}$ remains a periodic copy of Q. The authors demonstrated that the DDQL algorithm not only mitigates observed overestimations but also significantly improves accuracy. The training procedure of the DDQL agent is depicted in Fig. 3, which includes receiving the environment response and storing it in the memory bank, passing transitions to the evaluation network and updating its weights with the update function, and adjusting the weights of the target network. In this figure, θ' is the resulted state after applying action a.

B. DDQL-CCRA

Since the CCRA problem comprises different sets of variables and their corresponding constraints, to solve it based on the DDQL agent depicted in Fig. 3, the first step is to design a chain of agents, each of which is responsible for addressing one group of the variables. Our proposed chain consists of four DDQL agents. The first agent, denoted by Λ^{SP}, is intended to determine the location of service VNFs in response to requests ($\boldsymbol{g}$ and $\boldsymbol{z}$), and thus its action set is the set of network nodes. In other words, $a^{SP} \in \boldsymbol{\mathcal{A}}^{SP} = \boldsymbol{\mathcal{V}}$. Λ^{PA} is the second agent with action set $\boldsymbol{\mathcal{K}}$, and it is responsible for assigning the priority level of traffic. The remaining two agents route traffic by determining the inquiry path from the entering node to its VNF location and the response path in the opposite direction, denoted by Λ^{QPS} and Λ^{PPS}, respectively. The action set of these agents comprises all possible network paths. To interact with the system, each agent provides an action that contains the index of the request for which it is attempting to satisfy its resource requirements and a value from its action space. For example, $a^{SP} = \{r : 1, \xi : 3\}$ means that the VNF for request 1 should be located in node 3, or $g_{1,3} = 1$. Moreover,

$\boldsymbol{a} = \{a^{SP}, a^{PA}, a^{QPS}, a^{PPS}\}$ represents the set of all agents' actions.

Next, the system state has to be formulated. As the infrastructure is the only side of the system that is known, the state is a collection of network and cloud resources, that is:

$$\theta = \left[\widehat{\zeta_v} \forall v \in \boldsymbol{\mathcal{V}}\right] \oplus [\Psi_v \forall v \in \boldsymbol{\mathcal{V}}] \oplus \left[\widehat{B_l} \forall l \in \boldsymbol{\mathcal{L}}\right] \oplus [\Xi_l \forall l \in \boldsymbol{\mathcal{L}}] \oplus \left[\widehat{D_{k,l}} \forall k \in \boldsymbol{\mathcal{K}}, \forall l \in \boldsymbol{\mathcal{L}}\right], \tag{9}$$

where $\oplus$ returns the concatenation of two arrays. When the system receives actions, the state of the available network and cloud resources is updated by deactivating the resources assigned to the associated request, and resulted state θ' is generated.

The final step is to design the reward, which is a reaction to the effectiveness of action after receiving it and shifting from state θ to resulted state θ'. In other words, agents are wired to the system via the reward. To address the problem defined in (1), we propose the reward as follows for request r:

$$\beta = 100 \left(1 - \frac{\text{OF}_{r,\boldsymbol{a}} - min\ \text{OF}_r}{max\ \text{OF}_r - min\ \text{OF}_r}\right) \chi_{r,\boldsymbol{a}} \tag{10}$$

where max OF$_r$ and min OF$_r$ are the maximum and minimum costs that can be achieved by allocating the available network and cloud resources to request r without considering any constraints or requirements, OF$_{r,\boldsymbol{a}}$ is the cost of the allocations provided by the agents, and $\chi_{r,\boldsymbol{a}}$ represents the response of request r to actions $\boldsymbol{a}$. $\chi_{r,\boldsymbol{a}}$ ensures that all constraints of (1) are met. Consider $\boldsymbol{a}$ containing an action that violates one of the constraints (for example, a node or a VNF or a path is overloaded, or a priority level is assigned in such a way that the E2E delay requirement is violated). In this circumstance, the affected request will respond with $\chi_{r,\boldsymbol{a}} = 0$, and the reward for $\boldsymbol{a}$ will be 0. Therefore, the probability of selecting that action decreases, and after a certain number of iterations, actions with infeasible allocations are implicitly removed from the set of possible actions. Besides, OF$_{r,\boldsymbol{a}}$ controls the efficiency of $\boldsymbol{a}$. Similarly, after a number of iterations, allocations with lower costs will have a greater chance of being selected. Therefore, after training, agents will choose feasible actions (within the constraints of (1)) with lower costs (minimizing OF).

Now, Algorithm 3 details DDQL-CCRA, the learning algorithm proposed to solve the CCRA problem based on DDQL. The algorithm is divided into two phases:

1) Training Phase: In this phase (lines 1 to 24), T represents the number of training steps, whereas ϵ' and $\widetilde{\epsilon}$ are small positive integers to control the ϵ-greedy algorithm. Through each step, the set of actions is determined and transmitted to the system, after which the reward and the updated state are received and used to train the agents employing the ADAM optimizer [40] and update their DNN weights via the memory bank. This process is repeated over the set of requests until the specified maximum number of steps is reached. It is worth mentioning that the action in each agent is selected by an ϵ-greedy policy that follows the evaluation function of the corresponding agent with probability $(1-\epsilon)$ and selects a random action with probability ϵ. The probability is decreased linearly from ϵ to $\widetilde{\epsilon}$ during the training process. Using the ϵ-greedy method and the ADAM optimizer ensures the convergence of DDQL-CCRA to feasible, low-cost solutions (based on the defined reward) [41].

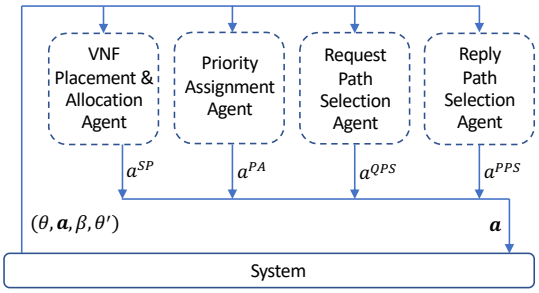


Fig. 4. Data flow between the agents and the system.

2) Decision Making Phase: In each step of this phase (lines 25 to 36), one request is selected, and its required resources are allotted by the agents. The decision is then transmitted to the system to collect the infrastructure's response and the request. Following this, the reward and the mean reward, denoted by $\overline{\beta}$, are determined. Fig. 4 depicts the actions generated by the agents, their transmission to the environment, and their subsequent return to the agents in preparation for the next decision-making. Due to the fact that we have no knowledge of the requests' requirements, every change in the criteria is managed by examining the average reward; if it falls below a specified threshold, denoted by $\widetilde{\beta}$, it indicates that end-users have adopted a new policy and the training phase must be repeated. This procedure continues until the required resources for each request have been determined.

C. DDQL-CCRA Resource Allocation Architecture

The architecture of the DDQL-CCRA resource allocation method is depicted in Fig. 5. Due to the fact that the characteristics of different services may be entirely different, an isolated DDQL-CCRA algorithm is designed to be executed for each service. The broker receives requests, classifies them, and forwards each service's requests to its respective controller. In addition, the broker collects the most recent state of the network and cloud resources from the resource orchestrator and transmits it to the controllers. The controller is responsible for executing the DDQL-CCRA algorithm by implementing the memory bank, maintaining the state of requests, calculating the reward, and returning action sets to the broker. Action sets are collected by the broker from all controllers and relayed to the resource orchestrator to apply to the infrastructure. Since actions are chosen at random during the training phase, digital twins could be used to evaluate them to prevent the infrastructure from entering unpredictable states that result in disruptions to its operation [42].

In order to enhance the scalability of this architecture, rather than considering the set of all nodes as the action set of Λ^{SP} and the set of all paths of the network as the action set of Λ^{QPS} and Λ^{PPS}, these spaces can be pruned to create fixed-size sets consisting of the most likely options for VNF placement and path selection.

Algorithm 3 DDQL-CCRA.

initialize T, ϵ' and $\widetilde{\epsilon}$
$\tau \leftarrow 0$, $\epsilon \leftarrow 1$, $memory \leftarrow \{\}, \overline{\beta} \leftarrow 0$
$\theta_\tau \leftarrow$ get the environment's current state
while there is active request **do**
  while $\tau \leq T$ **do**
    $\boldsymbol{a} \leftarrow \{\}$
    $r \leftarrow$ select one of the active requests
    for i in $\{SP, PA, QPS, PPS\}$ **do**
      $n \leftarrow$ generate one random number between 0 and 1
      if $n > \epsilon$ **then**
        $\xi \leftarrow argmax_{a \in \boldsymbol{\mathcal{A}}^i} Q(\theta_\tau, a, \boldsymbol{\mathcal{W}}_\tau)$ by Λ^i
      else
        $\xi \leftarrow$ select a random a from $\boldsymbol{\mathcal{A}}^i$
      $a^i \leftarrow \{r, \xi\}$
    $\boldsymbol{a} \leftarrow \{a^{SP}, a^{PA}, a^{QPS}, a^{PPS}\}$
    apply $\boldsymbol{a}$
    collect the infrastructure and the request responses
    calculate β
    $memory \leftarrow memory \cup \{(\theta, \boldsymbol{a}, \beta, \theta')\}$
    choose a sample form $memory$, and train agents
    if $\epsilon > \widetilde{\epsilon}$ **then**
      $\epsilon \leftarrow \epsilon - \epsilon'$
    $\tau \leftarrow \tau + 1$
    $\theta_\tau \leftarrow \theta'$
  $\boldsymbol{a} \leftarrow \{\}$
  $r \leftarrow$ select one of the active requests
  for i in $\{SP, PA, QPS, PPS\}$ **do**
    $\xi \leftarrow argmax_{a \in \boldsymbol{\mathcal{A}}^i} Q(\theta_\tau, a, \boldsymbol{\mathcal{W}}_\tau)$ by Λ^i
    $a^i \leftarrow \{r, \xi\}$
  $\boldsymbol{a} \leftarrow \{a^{SP}, a^{PA}, a^{QPS}, a^{PPS}\}$
  apply $\boldsymbol{a}$
  collect the infrastructure and the request responses
  calculate β
  $\overline{\beta} \leftarrow \gamma\beta + (1-\gamma)\overline{\beta}$, $\theta_\tau \leftarrow \theta'$
  if $\overline{\beta} < \widetilde{\beta}$ **then**
    go to 1

- For Λ^{SP}, the lower and upper boundaries of the QoS requirements for each service can be extracted (or considered inputs to the problem), and then a set with size $\mathcal{V}'$, named $\boldsymbol{\mathcal{V}'}$, including feasible nodes to maintain the QoS boundaries at the lowest cost (Ψ_v) is generated.
- For Λ^{QPS} and Λ^{PPS}, a set of size $\mathcal{P}'$ is created for each service containing feasible paths in order to maintain the QoS boundaries at the lowest cost (Ξ_l). Note that these paths should begin at the edge devices (the entry nodes of requests) and terminate at one of the nodes of $\boldsymbol{\mathcal{V}'}$ for Λ^{QPS}. In Λ^{PPS}, the same logic is followed, but in reverse order.

The complexity and accuracy of the DDQL-CCRA algorithm can be modified by adjusting the size of these sets. $\mathcal{V}'$ and $\mathcal{P}'$ can be set to large numbers if high precision is required or if the complexity of running the DDQL-CCRA algorithm can be handled by high-powered software/hardware. Alternatively, small sets can be utilized to return the result in a relatively shorter amount of time.

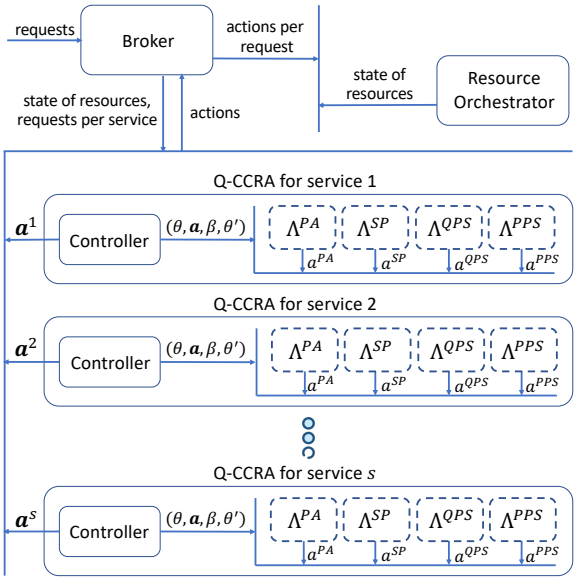


Fig. 5. DDQL-CCRA Resource Allocation Architecture.

TABLE III
SIMULATION PARAMETERS.

Parameter	Value
number of links ($\mathcal{L}$)	$\sim \mathcal{U}\{3\mathcal{V}, 5\mathcal{V}\}$, where the resulted graph is connected.
number of priority levels ($\mathcal{K}$)	4
number of services ($\mathcal{S}$)	3
number of tiers	3
capacity per link ($\widehat{B_l}$)	$\sim \mathcal{U}\{250, 300\}$ mbps
cost per link (Ξ_l)	$\sim \mathcal{U}\{10, 20\}$
computing capacity per node ($\widehat{\zeta_v}$)	$\sim 100\,\mathcal{U}(x, x+1)$ mbps
cost per node (Ψ_v)	$10^{\,x+1}$
bandwidth bound per link-priority ($\widehat{f_{l,k}}$)	$\widehat{B_l}/\mathcal{K}$
queue size per priority ($\widehat{T_k}$)	$200/\mathcal{K}$
VNF capacity per service ($\widehat{\mathcal{C}_s}$)	20 mbps
capacity requirement per request ($\widetilde{\mathcal{C}_r}$)	$\sim \mathcal{U}\{4, 8\}$ mbps
bandwidth requirement per request ($\widetilde{\mathcal{B}_r}$)	$\sim \mathcal{U}\{2, 10\}$ mbps
traffic burstiness per request ($\widetilde{\mathcal{T}_r}$)	$\sim \mathcal{U}\{1, 4\}$
packet size per request ($\widetilde{\mathcal{H}_r}$)	1

x is the number of tiers minus the tier number of the node

TABLE IV
TRAINING CONFIGURATION.

Parameter	Value
number of training steps	10^4
learning rate	10^{-4}
memory size	5×10^4
batch size	32
discount factor (γ)	0.99
epsilon decrement (ϵ')	5×10^{-6}
epsilon bound ($\widetilde{\epsilon}$)	5×10^{-2}

VI. NUMERICAL RESULTS

In this section, the efficiency of the proposed methods is numerically investigated. The system model parameters are listed in Table III, and the configuration of the agents' training procedure is shown in Table IV. Note that the results are

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301506

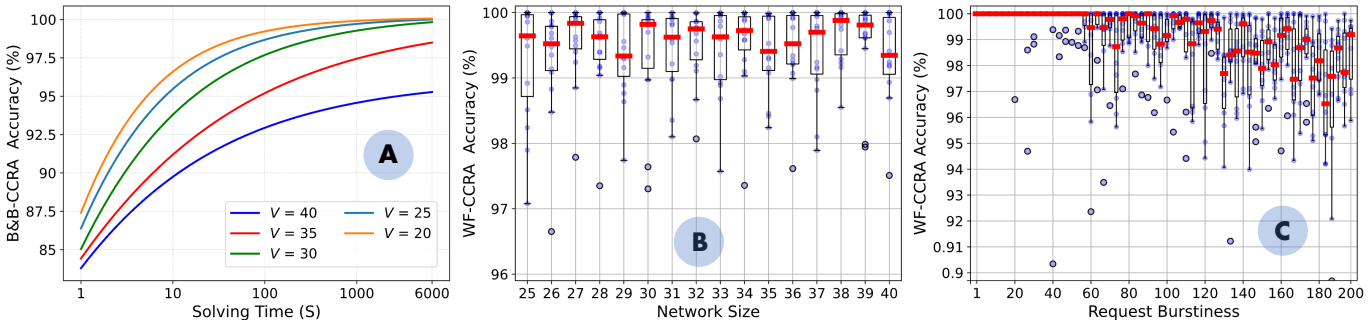


Fig. 6. Solution accuracy of A) B&B-CCRA vs. solving time, B) WF-CCRA vs. network size, and C) WF-CCRA vs. request burstiness. In subfigures A and B, the number of requests is set to 200, and the number of network nodes in subfigure C is 20. In subfigure B and C, for each number of nodes or requests, 50 random systems are generated, and the problem is solved using WF-CCRA, with B&B-CCRA providing the optimal solution. The results of random samples are represented by blue dots, and the aggregated results are represented through boxplots, where red points indicate medians.

obtained on a computer with 8 processing cores, 16 GB of memory, and a 64-bit operating system.

The accuracy of the B&B-CCRA and WF-CCRA methods is illustrated in Fig. 6. The methods are evaluated based on the accuracy of the solutions they provide. Note that the accuracy of a solution for a scenario named η is defined as $1-((\eta-\eta^\star)/\eta^\star)$, where $\eta^\star$ is the scenario's optimal solution, which is obtained by solving it with CPLEX 12.10. In Fig. 6.A, the accuracy of B&B-CCRA is plotted vs. the solving time (in logarithmic scale) for five scenarios with different network sizes. As illustrated, the accuracy of B&B-CCRA starts at 80% after the first iteration, which is obtained by solving the LP transformation of LiCCRA with CPLEX 12.10 in just a few milliseconds, and increases as the solving time passes, reaching 92% for all samples after 100 seconds. It proves that this method can be easily applied to provide baseline solutions for small and medium size use cases. However, the accuracy growth is slowed by increasing the network size, which is expected given the problem's NP-hardness and complexity. In the two remaining subfigures, the accuracy of WF-CCRA is depicted against the number of requests attending to use system resources, known as request burstiness, and network size. It is evident that regardless of network size, WF-CCRA has an average accuracy greater than 99%, implying that it can be used to allocate resources in a near-optimal manner even for large networks. For different numbers of requests, the average accuracy remains significantly high and greater than 96%. It does, however, slightly decrease as the number of requests increases, which is the cost of decomplexifying the problem by allocating the resources through isolating requests. Since the decrease is negligible, it is expected that the algorithm is capable of allocating resources efficiently for large numbers of requests.

The DDQL-CCRA resource allocation architecture, depicted in Fig. 5, is examined in Fig. 7. In this figure, the mean cost and E2E delay per each supported request, as well as the percentage of supported requests, are plotted against the DDQL-CCRA iteration counter for three scenarios with varying E2E delay requirements. In order to supplement the analysis, this figure additionally includes the outcomes of WF-CCRA in parallel to R-, CM-, and DM-CCRA. In R-CCRA, all allocations are determined at random, but in CM- and DM-CCRA, allocations are made to minimize cost and delay, respectively, without considering other constraints. Note that in order to implement DDQL-CCRA, we deployed the DDQL-CCRA resource allocation architecture on all edge devices (the entry nodes).

When $\widetilde{D_r}$ is less than 1 ms, the only feasible solution is to assign all requests to the most costly nodes of the first tier. Consequently, the mean cost for all techniques is high, with the exception of CM-CCRA, which attempts to fit all requests into one of the third-tier nodes with the lowest cost, resulting in the inability to support any request and the mean cost of 0. Since the mean delay for all nodes in the first tier is too low, the average delay per each supported request for all methods excluding CM-CCRA is less than 1 ms and similar. However, the supported request rate is entirely different for each method. R-CCRA, which assigns nodes evenly to requests, places a third of requests on the first tier, therefore its rate is approximately 33%. DM-CCRA selects the node with the shortest E2E delay; hence, its support rate is the number of requests that can be serviced by a single node in the first tier, which is approximately 45%. Given that DDQL-CCRA employs the ϵ-greedy technique, it also generates random results during the initial learning iterations. However, as the learning progresses, it receives the reward based on end-user responses and begins to place more and more requests on the first tier until it reaches the near-optimal solutions supplied by WF-CCRA.

When the E2E delay requirement threshold is changed to 3 ms, both the first and second tier nodes can be occupied to support requests. Since DM- and CM-CCRA always select a node in the first and third tiers, respectively, their outcomes are identical to those of the preceding scenario. R-CCRA doubles the percentage of supported requests because it randomly assigns 66% of requests to the first and second tiers. In addition, its mean delay is slightly smaller than that of WF- and DDQL-CCRA since it utilizes the first tier nodes more than these two cost-effective approaches. Note that the difference is negligible, as the delay of nodes in the first tier is vanishingly small and cannot significantly affect the mean delay. In contrast, when DDQL-CCRA identifies a changing need (lines 35 and 36 of Algorithm 3), it restarts the learning process and enables the ϵ-greedy technique. Therefore, it

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301506

Fig. 7. DDQL-CCRA convergence compared with WF-CCRA, R-CCRA, CM-CCRA, and DM-CCRA. The number of requests is 150, and the number of network nodes is 30, and all requests can be serviced by a single tier's resources. The results are calculated as a moving average with a window size of 100 in order to capture trend lines.

begins anew with random results and optimizes allocation by fitting as many requests as feasible into the second-tier nodes in ascending cost order. Also in this scenario, it can be observed that the learning technique yields near-optimal efficiency outcomes.

The final scenario is eliminating the delay requirement and releasing the entire infrastructure to serve requests. In this case, although the results for DM-CCRA are identical, the support rate for CM-CCRA is approximately 50%, indicating that the node with the lowest cost can service approximately 50% of requests. Similar to the prior scenario, the outcomes of R-CCRA are enhanced. Now it can support all requests, but its mean cost and delay are not optimal because it consumes the resources of all tiers equally. Similarly, the trend for DDQL-CCRA is the same. As soon as it senses a change in requirements, it begins to randomly assign resources, recognizing that requests should be sent as much as possible to the core clouds. It initially determines that the node with the lowest cost yields the best outcome. Therefore, it places all requests on a single node, thereby reducing the number of supported requests and enhancing the mean delay and cost. Subsequently, the reward of this allocation begins to decline as certain requests cannot be supported, the value of dispersing requests throughout the third tier increases progressively, and the optimal policy leads to an increase in the support rate, coupled with a reduction in the mean cost and delay. In Fig. 7, it is evident that the DDQL-CCRA approach in partially known systems can lead to near-optimal solutions obtained when those systems are fully known.

In Fig. 8, DDQL-CCRA is investigated with regard to request burstiness. The mean cost and E2E delay of each supported request are depicted in Fig. 8.A and Fig. 8.C respectively, whereas Fig. 8.B illustrates the number of supported requests. In this figure, the results of DDQL-CCRA are compared with those of WF-CCRA, FSA [13], BSA [13], CEP [10], A-DDPG [11], and MDRL-SaDS [14]. FSA is a heuristic algorithm that randomly assigns resources to requests in descending order of their required computing capacity. BSA is a similar method that assigns resources in descending order of their remaining capacity to the sorted requests. In CEP, resources are allocated with the aim of minimizing the total cost of links. A-DDPG is an RL method that adjusts the reward for each request to maximize its overall utility. In this solution, utility is defined as the profit of serving the request as a function of its required bandwidth minus the E2E path delay experienced. MDRL-SaDS is another RL technique in which the reward is the computing and networking cost of serving each request divided by the total cost of allocated resources across the infrastructure. This strategy seeks to minimize the cost of allocated resources in relation to their energy consumption.

Evidently, the number of requests supported by FSA is relatively high, as are its mean cost and E2E delay. FSA distributes requests across all tiers, resulting in significant utilization of all resources, a high mean cost, and a high mean E2E delay. Since all links have a similar cost, CEP exhibits comparable performance; however, because it considers the feasibility of links, it achieves slightly better results. A-DDPG, where requests are assigned to nodes with a lower E2E delay, is another costly method. Increasing the number of requests causes second- and first-tier nodes to become occupied and requests to be assigned to the resources of other tires, thereby

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301506

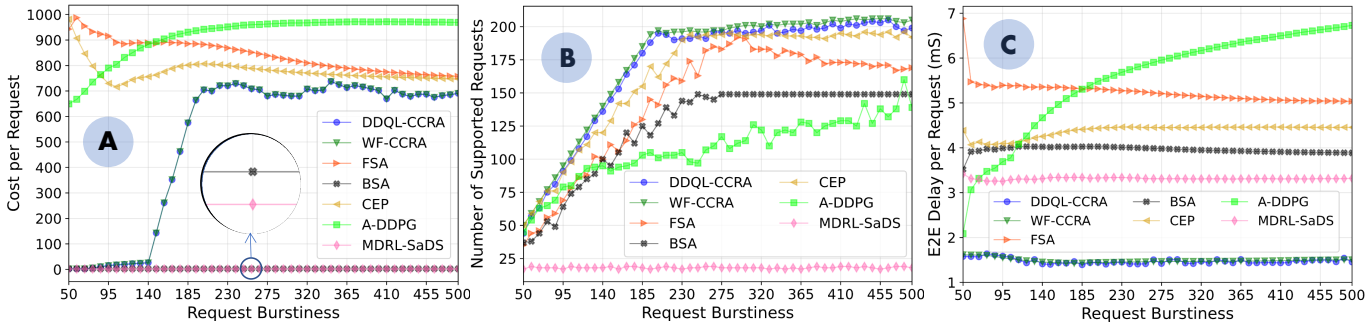


Fig. 8. A) Mean cost of each supported request, B) number of supported requests, and C) mean E2E delay of each supported request for DDQL-CCRA, WF-CCRA, FSA & BSA [13], CEP [10], A-DDPG [11], and MDRL-SaDS [14] vs. request burstiness. The delay requirement of requests is 10 ms, and the number of network nodes is 9. The results are calculated as a moving average with a window size of 100, where each sample is the average of 50 arbitrary systems.

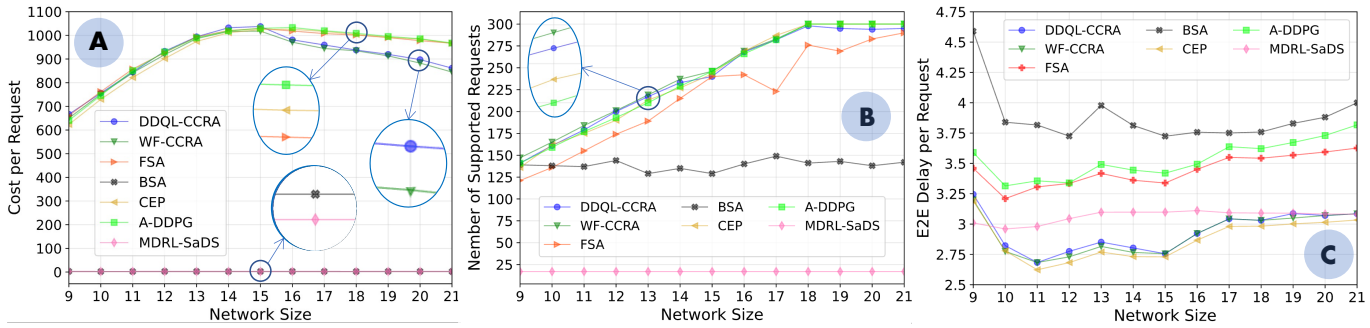


Fig. 9. A) Mean cost of each supported request, B) number of supported requests, and C) mean E2E delay of each supported request for DDQL-CCRA, WF-CCRA, FSA [13], BSA [13], CEP [10], A-DDPG [11], and MDRL-SaDS [14] vs. network size. In this scenario, the first four nodes are added to the first tier, followed by the second four nodes to the second tier, and then the last four nodes to the third tier. The delay requirement is 10 ms, and there are a total of 300 requests. The results are calculated as a moving average with a window size of 20, where each sample is the average of 50 arbitrary systems.

increasing the E2E delay. In BSA, because the performance metric is the remaining capacity of computing nodes and the nodes are ordered from low capacity to high capacity across the tiers, it occupies the nodes from the cloud to the edge, resulting in outcomes with very low cost and moderate delay. A-DDPG and BSA cannot support a substantial number of requests because the feasibility of links is not explicitly evaluated. MDRL-SaDS is the most inefficient method by which requests are routed to the node with the lowest cost. Therefore, the number of supported requests is proportional to the node's capacity and delay. The behavior of the two remaining methods, DDQL- and WF-CCRA, is comparable. They support requests by initially assigning third-tier nodes. Then, once this tier is occupied, they proceed to occupy the second tier, resulting in an exponential cost increase. The mean cost converges to a fixed value when all resources are occupied and the first tier is in use. This approach results in a very low E2E delay because it assigns request priorities based on their delay requirements (unlike other approaches, which are unaware of ATS queues). DDQL-CCRA can provide near-optimal solutions regardless of the number of requests received, as demonstrated.

The final figure compares the proposed approaches to the approaches depicted in the preceding figure for various network sizes. In this scenario, if $\mathcal{V} \leq 15$, the first-tier network has a very high capacity, whereas if $\mathcal{V} > 15$, there are sufficient resources to fulfill all requests. Therefore, CEP and A-DDPG, which focus on minimizing the cost of allocated links and E2E delay respectively, as well as FSA, which allocates resources randomly, can support a large number of requests despite the high cost of allocations. Since the capacity of the low-cost tier for BSA and MDRL-SaDS is not excessive (and the capacity ratio of this tier to the others is less than the previous figure), the request support rate is unpromising despite the low cost. When the infrastructure is full ($\mathcal{V} \leq 15$), the results of WF- and DDQL-CCRA are comparable to those of other algorithms. However, when there are more resources ($\mathcal{V} > 15$), these two approaches move requests to low-cost resources, thereby reducing the total cost of allocations. When it comes to E2E delay, even though the results are similar for all methods, by adding a node to the initial network, the resources of the first tier are extended and more requests can be supported with smaller delays, resulting in a sudden decrease for $\mathcal{V} = 10$. By adding more nodes, however, more resources are added to the other tires, and FSA (which allocates resources randomly), BSA (which assigns resources in descending order of their remaining capacity), and A-DDPG (which tries to maximize the overall utility) migrate requests to the lower-cost tiers, resulting in a slight increase in E2E delay. Despite the increase in WF- and DDQL-CCRA techniques, their outcome is the lowest E2E delay because they manage priority queues according to the delay requirements of requests.

VII. Conclusion

In this paper, the joint problem of communication and computing resource allocation comprising VNF placement and assignment, traffic prioritization, and path selection considering capacity and delay constraints was studied. The primary objective was to minimize the total cost of allocations. We initially formulated the problem as a MINLP model, and used a method, named B&B-CCRA, to solve it optimally. Then, due to the complexity of B&B-CCRA, a WF-based approach was developed to find near-optimal solutions in a timely manner. These two methods can be utilized to solve the problem when the system is fully known. However, for scenarios wherein there is no request-specific information, a DDQL-based architecture was presented that yields near-optimal solutions. The efficiency of the proposed methods was demonstrated by numerical results.

As potential future work, we intend to address the problem by accounting for more dynamic environments in which end-users are mobile and all of their needs are subject to change. In addition, the proposed methods could be supplemented by taking into account dynamic infrastructure resources, in which the cost of resources (such as their energy consumption) or their availability can fluctuate over time. In such highly dynamic scenarios, we intend to enhance the proposed DDQL-based method with Continual Learning in order to reduce the adaptation time required to adjust agents after each change. Another possible research direction is to extend the problem to include radio domain resources (such as power control, channel assignments, rate control, and relay selection in multi-hop scenarios), thereby providing end-user-to-end-user resource allocations. Furthermore, we intend to improve the proposed method for allocating resources to VNF chains rather than individual VNFs.

Acknowledgment

This research work is partially supported by the Business Finland 6Bridge 6Core project under Grant No. 8410/31/2022, the Academy of Finland IDEA-MILL project under Grant No. 352428, the European Union's Horizon 2020 ICT Cloud Computing program under the ACCORDION project with grant agreement No. 871793, and the Academy of Finland 6G Flagship program under Grant No. 346208.

References

[1] T. Taleb, A. Boudi, L. Rosa, L. Cordeiro, T. Theodoropoulos, K. Tsepes, P. Dazzi, A. Protopsaltis, and R. Li, "Towards Supporting XR Services: Architecture and Enablers," *IEEE Internet of Things Journal*.

[2] L. Corneo, M. Eder, N. Mohan, A. Zavodovski, S. Bayhan, W. Wong, P. Gunningberg, J. Kangasharju, and J. Ott, "Surrounded by the Clouds: A Comprehensive Cloud Reachability Study," in *Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, Apr. 2021, pp. 295–304.

[3] X. Yang, Z. Zho, and B. Huang, "URLLC Key Technologies and Standardization for 6G Power Internet of Things," *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 52–59, June 2021.

[4] T. Taleb, I. Afolabi, K. Samdanis, and F. Z. Yousaf, "On Multi-Domain Network Slicing Orchestration Architecture and Federated Resource Control," *IEEE Network*, vol. 33, no. 5, pp. 242–252, Sept. 2019.

[5] T. Taleb, P. A. Frangoudis, I. Benkacem, and A. Ksentini, "CDN Slicing over a Multi-Domain Edge Cloud," *IEEE Transactions on Mobile Computing*, vol. 19, no. 9, pp. 2010–2027, Sept. 2020.

[6] Y. Li, J. Huang, Q. Sun, T. Sun, and S. Wang, "Cognitive Service Architecture for 6G Core Network," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 10, pp. 7193–7203, Oct. 2021.

[7] M. Emu, P. Yan, and S. Choudhury, "Latency Aware VNF Deployment at Edge Devices for IoT Services: An Artificial Neural Network Based Approach," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, June 2020, pp. 1–6, iSSN: 2474-9133.

[8] X. Vasilakos, M. Bunyakitanon, R. Nejabati, and D. Simeonidou, "Towards Low-latent & Load-balanced VNF Placement with Hierarchical Reinforcement Learning," in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, Sept. 2021, pp. 162–167.

[9] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-Driven Deep Reinforcement Learning for Scalable Fog and Service Placement," *IEEE Transactions on Services Computing*, pp. 1–1, 2021.

[10] M. Liu and S. B. Alias, "Cost-Efficient Virtual Network Function Placement in an Industrial Edge System: A Proposed Method," *IEEE Systems, Man, and Cybernetics Magazine*, vol. 9, no. 1, pp. 10–17, Jan. 2023.

[11] N. He, S. Yang, F. Li, S. Trajanovski, L. Zhu, Y. Wang, and X. Fu, "Leveraging Deep Reinforcement Learning With Attention Mechanism for Virtual Network Function Placement and Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1186–1201, Apr. 2023.

[12] M. Iwamoto, A. Suzuki, and M. Kobayashi, "Optimal VNF Scheduling for Minimizing Duration of QoS Degradation," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, Jan. 2023, pp. 855–858, iSSN: 2331-9860.

[13] D. H. P. Nguyen, Y.-H. Lien, B.-H. Liu, S.-I. Chu, and T. N. Nguyen, "Virtual Network Function Placement for Serving Weighted Services in NFV-Enabled Networks," *IEEE Systems Journal*, pp. 1–12, 2023.

[14] H. Xuan, Y. Zhou, X. Zhao, and Z. Liu, "Multi-agent deep reinforcement learning algorithm with self-adaption division strategy for VNF-SC deployment in SDN/NFV-Enabled Networks," *Applied Soft Computing*, vol. 138, p. 110189, May 2023.

[15] T. Miyamura and A. Misawa, "Joint optimization of optical path provisioning and VNF placement in vCDN," *Optical Switching and Networking*, vol. 49, p. 100740, May 2023.

[16] C. Yang, B. Hu, Y. Feng, H. Huang, H. Lai, and J. Tan, "An online service function chain orchestration method for profit maximization in edge computing networks," *Engineering Reports*, vol. n/a, no. n/a, p. e12653.

[17] 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," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1562–1576, Aug. 2018.

[18] B. E. Mada, M. Bagaa, T. Tale, and H. Flinck, "Latency-aware Service Placement and Live Migrations in 5G and Beyond Mobile Systems," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, June 2020, pp. 1–6, iSSN: 1938-1883.

[19] Q. Zhang, F. Liu, and C. Zeng, "Adaptive Interference-Aware VNF Placement for Service-Customized 5G Network Slices," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Apr. 2019, pp. 2449–2457, iSSN: 2641-9874.

[20] Q. Yuan, X. Ji, H. Tang, and W. You, "Toward Latency-Optimal Placement and Autoscaling of Monitoring Functions in MEC," *IEEE Access*, vol. 8, pp. 41 649–41 658, 2020.

[21] T. Gao, X. Li, Y. Wu, W. Zou, S. Huang, M. Tornatore, and B. Mukherjee, "Cost-Efficient VNF Placement and Scheduling in Public Cloud Networks," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4946–4959, Aug. 2020.

[22] C. D. Alwis, A. Kalla, Q.-V. Pham, P. Kumar, K. Dev, W.-J. Hwang, and M. Liyanage, "Survey on 6G Frontiers: Trends, Applications, Requirements, Technologies and Future Research," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 836–886, 2021.

[23] M. Shokrnezhad and T. Taleb, "Near-optimal Cloud-Network Integrated Resource Allocation for Latency-Sensitive B5G," in *2022 IEEE Global Communications Conference (GLOBECOM)*, Rio De Janeiro, Brazil, Dec. 2022.

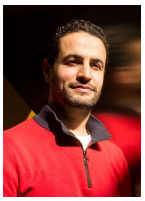
[24] J. Lei, S. Deng, Z. Lu, Y. He, and X. Gao, "Energy-saving traffic scheduling in backbone networks with software-defined networks," *Cluster Computing*, vol. 24, no. 1, pp. 279–292, Mar. 2021.

[25] J. Specht and S. Samii, "Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016, pp. 75–85, iSSN: 2159-3833.

[26] U. Arshad, M. Aleem, G. Srivastava, and J. C.-W. Lin, "Utilizing power consumption and SLA violations using dynamic VM consolidation in

cloud data centers," *Renewable and Sustainable Energy Reviews*, vol. 167, p. 112782, Oct. 2022.
[27] S. Kianpisheh and T. Taleb, "A Survey on In-network Computing: Programmable Data Plane And Technology Specific Applications," *IEEE Communications Surveys and Tutorials (COMST)*.
[28] J. R. Bhat and S. A. Alqahtani, "6G Ecosystem: Current Status and Future Perspective," *IEEE Access*, vol. 9, pp. 43 134–43 167, 2021.
[29] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G Networks: Use Cases and Technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, Mar. 2020.
[30] U. Gustavsson, P. Frenger, C. Fager, T. Eriksson, H. Zirath, F. Dielacher, C. Studer, A. Pärssinen, R. Correia, J. N. Matos, D. Belo, and N. B. Carvalho, "Implementation Challenges and Opportunities in Beyond-5G and 6G Communication," *IEEE Journal of Microwaves*, vol. 1, no. 1, pp. 86–100, Jan. 2021.
[31] N. H. Mahmood, S. Böcker, I. Moerman, O. A. López, A. Munari, K. Mikhaylov, F. Clazzer, H. Bartz, O.-S. Park, E. Mercier, S. Saidi, D. M. Osorio, R. Jäntti, R. Pragada, E. Annanperä, Y. Ma, C. Wietfeld, M. Andraud, G. Liva, Y. Chen, E. Garro, F. Burkhardt, C.-F. Liu, H. Alves, Y. Sadi, M. Kelanti, J.-B. Doré, E. Kim, J. Shin, G.-Y. Park, S.-K. Kim, C. Yoon, K. Anwar, and P. Seppänen, "Machine type communications: key drivers and enablers towards the 6G era," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, p. 134, June 2021.
[32] H. Yu, C. Wang, T. Taleb, and J. Zhang, "Deep Reinforcement Learning based Deterministic Routing and Scheduling for Mixed-Criticality Flows," *IEEE Transactions on Industrial Informatics*.
[33] Q. Guo, R. Gu, H. Yu, T. Taleb, and Y. Ji, "Probabilistic-Assured Resource Provisioning with Customizable Hybrid Isolation for Vertical Industrial Slicing," *IEEE Transactions on Network and Service Management (TNSM)*.
[34] T. Taleb, I. Afolabi, and M. Bagaa, "Orchestrating 5G Network Slices to Support Industrial Internet and to Shape Next-Generation Smart Factories," *IEEE Network*, vol. 33, no. 4, pp. 146–154, July 2019.
[35] H. Kellerer, U. Pferschy, and D. Pisinger, "Multidimensional Knapsack Problems," in *Knapsack Problems*, H. Kellerer, U. Pferschy, and D. Pisinger, Eds. Berlin, Heidelberg: Springer, 2004, pp. 235–283.
[36] G. Pataki, M. Tural, and E. B. Wong, "Basis Reduction and the Complexity of Branch-and-Bound," in *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, ser. Proceedings. Society for Industrial and Applied Mathematics, Jan. 2010, pp. 1254–1261.
[37] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016, number: 1.
[38] C. J. C. H. Watkins and P. Dayan, "Technical Note," in *Reinforcement Learning*, ser. The Springer International Series in Engineering and Computer Science, R. S. Sutton, Ed. Boston, MA: Springer US, 1992, pp. 55–68.
[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[41] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.
[42] H. X. Nguyen, R. Trestian, D. To, and M. Tatipamula, "Digital Twin for 5G and Beyond," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 10–15, Feb. 2021.

Masoud Shokrnezhad received his B.Sc. degree in information technology from Shahid Madani University of Azerbaijan, Tabriz, Iran, and his M.Sc. and Ph.D. degrees (as a bright talent) in computer networks from Amirkabr University of Technology (Tehran Polytechnic), Tehran, Iran, in 2011, 2013, and 2019, respectively. He is currently a postdoctoral researcher with the Center of Wireless Communications, University of Oulu, Oulu, Finland. Between June 2021 and December 2021, he was a postdoctoral researcher at the School of Electrical Engineering, Aalto University, Espoo, Finland. Prior to that, he had been working as a senior system designer and engineer with FavaPars and Pouya Cloud Technology, Tehran, Iran, since 2013. He has been engaged in many national, international, and European projects working on designing and developing computing and networking frameworks, and has been directly co-managing a startup focusing on developing SDWAN solutions for B2B use cases. His research interests lie in the fields of artificial intelligence and machine learning, operations research and optimization theory, semantic networking and communications, as well as resource allocation. In addition to this, he is a full-stack developer who has extensive practical experience working with JS-based technologies.



Tarik Taleb (Senior Member, IEEE) received the B.E. degree (with distinction) in information engineering and the M.Sc. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 2001, 2003, and 2005, respectively. He is currently a professor with the Center of Wireless Communications, University of Oulu, Oulu, Finland. He is the founder and director of MOSA!C Lab, Espoo, Finland. He has also been directly engaged in the development and standardization of the Evolved Packet System as a member of 3GPP's System Architecture Working Group 2. His research interests lie in the fields of telco cloud, network softwarization, network slicing, AI-based software-defined security, immersive communications, mobile multimedia streaming, and next-generation mobile networking. Prof. Taleb is the recipient of the 2021 IEEE ComSoc Wireless Communications Technical Committee Recognition Award in December 2021, and the 2017 IEEE ComSoc Communications Software Technical Achievement Award in December 2017 for his outstanding contributions to network softwarization. He is also the co-recipient of the 2017 IEEE Communications Society Fred W. Ellersick Prize in May 2017, and many other awards from Japan.

Patrizio Dazzi received the B.Sc. degree in computer science and the M.Sc. degree in computer technologies (computer systems and networks) degree from the University of Pisa, Pisa, Italy, in 2003 and 2004, respectively, and the Ph.D. degree in computer science and engineering from the IMT School for Advanced Studies Lucca, Lucca, Italy, in 2008. He is a researcher with a strong interest in high-performance distributed systems, particularly in models and algorithms for the decentralized management of computations and data. He is currently a senior researcher with the University of Pisa, Pisa, Italy, and the co-founder and co-leader of the Pervasive AI Laboratory, a joint initiative of the University of Pisa and the National Research Council of Italy. He has been the Scientific Coordinator of the EU-South Korea H2020 BASMATI project, he served as a Project Coordinator for the EU H2020 ACCORDION project, and he is currently serving as an Innovation Coordinator for the EU H2020 TEACHING project. Dr. Dazzi served as a program committee member at many conferences in the field of large, distributed systems, including IEEE ICDCS, IEEE CLOUD, ACM SoCC, EuroPar, ACM SIGKDD, ACM WSDM, and IEEE HPCS. He is a member of the Executive Committee of the IEEE Technical Committee on Cloud Computing. He has a long-lasting experience as a speaker in international venues.