Computer Communications 184 (2022) 107–117

Contents lists available at ScienceDirect

Computer Communications

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

Latency and quality-aware task offloading in multi-node next generation RANs[☆],[☆☆]

Ayman Younis[*], Brian Qiu, Dario Pompili

Electrical and Computer Engineering, Rutgers University–New Brunswick, NJ, USA

ARTICLE INFO

Keywords:
NG-RAN
Tasks offloading
Convex optimization
OpenAirInterface (OAI)
Testbed

ABSTRACT

Next-Generation Radio Access Network (NG-RAN) is an emerging paradigm that provides flexible distribution of cloud computing and radio capabilities at the edge of the wireless Radio Access Points (RAPs). Computation at the edge bridges the gap for roaming end users, enabling access to rich services and applications. In this paper, we propose a multi-edge node task offloading system, i.e., QLRan, a novel optimization solution for latency and quality tradeoff task allocation in NG-RANs. Considering constraints on service latency, quality loss, edge capacity, and task assignment, the problem of joint task offloading, latency, and Quality Loss of Result (QLR) is formulated in order to minimize the User Equipment (UEs) task offloading utility, which is measured by a weighted sum of reductions in task completion time and QLR cost. The QLRan optimization problem is proved as a Mixed Integer Nonlinear Program (MINLP) problem, which is a NP-hard problem. To efficiently solve the QLRan optimization problem, we utilize Linear Programming (LP)-based approach that can be later solved by using convex optimization techniques. Additionally, a programmable NG-RAN testbed is presented where the Central Unit (CU), Distributed Unit (DU), and UE are realized by USRP boards and fully container-based virtualization approaches. Specifically, we use OpenAirInterface (OAI) and Docker software platforms to deploy and perform the NG-RAN testbed for different functional split options. Then, we characterize the performance in terms of data input, memory usage, and average processing time with respect to QLR levels. Simulation results show that our algorithm performs significantly improves the network latency over different configurations.

1. Introduction

Motivation: Mobile platforms (e.g., smartphones, tablets, IoT mobile devices) are becoming the predominant medium of access to Internet services due to a tremendous increase in their computation and communication capabilities. However, enabling applications that require real-time, in-the-field data collection and mobile platform processing is still challenging due to (i) the insufficient computing capabilities and unavailable aggregated/global data on individual mobile devices and (ii) the prohibitive communication cost and response time involved in offloading data to remote computing resources such as cloud datacenters for centralized computation. In light of these limitations, the *edge computing* term was introduced to unite telco, IT, and cloud computing and provide cloud services directly from the network edge. In general, the edge cloud servers or nodes are usually deployed directly at the mobile Base Stations (BSs) of a Radio Access Network (RAN), or at the local wireless Access Points (APs) using a generic-computing platform. Hence, the edge cloud node has ability to execute the offloading applications in close proximity to end users. In this way, the network end-to-end (e2e) latency and the back/mid/fronth-haul cost will be reduced. Recently, Cloud Radio Access Network (C-RAN) [2] has been emerged as a clean-slate redesign of the mobile network architecture in which parts of physical-layer communication functionalities are decoupled from distributed, possibly heterogeneous, Radio Access Points (RAPs), i.e., BSs or WiFi hotspots, and are then consolidated into a baseband unit pool for centralized processing. However, the centralized C-RAN design follows a *"one size fits all"* architectural approach, which makes it difficult to address the wide range of Quality of Service (QoS) requirements and support different types of traffic [3]. Also, a fully centralized architecture imposes high capacity requirements on fronthaul links [4]. Therefore, Next Generation RANs (NG-RAN) [5] has been introduced as a resource-efficient solution to address the above issues and reduce deployment costs. *It is worthy of note that, due to the flexibility of NG-RAN architecture,*

[☆] A preliminary/shorter version of this work appeared in the Proc. of the *IEEE/IFIP Wireless On-demand Network Systems and Services Conference (WONS)*, Mar'21 [1].

[☆☆] This work was supported by the US NSF Grant No. ECCS-2030101.

[*] Corresponding author.
E-mail addresses: a.younis@rutgers.edu (A. Younis), brian.qiu@rutgers.edu (B. Qiu), pompili@rutgers.edu (D. Pompili).

https://doi.org/10.1016/j.comcom.2021.11.026
Received 30 July 2021; Received in revised form 27 October 2021; Accepted 29 November 2021
Available online 24 December 2021

mobile network operators will have high degree of freedom to move from a "full centralization" in C-RAN to a "partial centralization" in NG-RAN with a specific functional splitting option to a "distributed approach" in edge cloud [6]—enabling rich services and applications in close proximity to the end users.

Task offloading can enhance the performance of mobile devices because servers in the edge cloud have higher computation capabilities than mobile devices. Therefore, enabling task offloading in NG-RAN is proposed to address the limitations (e.g., storage and computing resources) in the existing RANs. Meanwhile, in some cases, processing the entire input data in edge cloud servers would require more than the available computing resources to meet the desired latency/throughput guarantees. In the context of NG-RAN applications (e.g., IoT, AR/VR), transferring, managing, and analyzing large amounts of data in an edge cloud would be prohibitively expensive. Hence, the tradeoff between service latency and the tolerance of quality loss can improve key network performance metrics like the user's QoS [7,8]. In this paper, we define the Quality Loss of Results (QLR) term as the level of relaxing/approximating in data processing while the user's QoS is still at an acceptable level. Accordingly, *our key idea is motivated by the observation that in several NG-RAN applications such as media processing, image processing, and data mining, a high-accuracy result is not always necessary and desirable; instead, obtaining a suboptimal result with low latency cost is more acceptable by vendors or end users.* Consequently, relaxing QLR in such applications alleviates the required computation workload and enables a significant reduction of latency and computing cost in NG-RAN.

Our Vision: Our objective is to design a holistic decision-maker for an optimal joint task offloading scheme with quality and latency awareness in a multi-edge NG-RAN to minimize the UEs' overall offloading cost. Specifically, we consider a multi-edge node network where each RAP is equipped with an edge node to provide computation offloading services to UEs. In this way, several key benefits could be brought up to NG-RAN system over the multi-node servers; (i) preventing the resource-limited edge node/servers from becoming the bottleneck. Usually, the cloud servers overload when serving a large number of UEs with high processing priority. By directing many UEs to nearby edge nodes, the overloaded can be alleviated; (ii) reducing the energy consumption and network latency. Each UE has the capability to offload its task to the RAP with a more favorable uplink channel condition; (iii) getting better network collaboration. The NG-RAN with multi-RAP set could coordinate with each other to manage and balance the computation resources between the edge servers. *In this work, a Latency and Quality tradeoffs task offloading problem, QLRan, is formulated to trade off between the service latency and the acceptable level of QLR under specific application requirements (e.g., QoS, computing, and transmitting demands). Additionally, the process of task allocation across edge nodes is formulated as an objective optimization problem. The optimization objectives include both minimizing the average service latency and reducing the overall quality loss.*

Our Contributions: The main objective of this paper is to design the QLRan algorithm, optimizing the trade-off between the application completion time and QLR cost. The main contributions of this paper are summarized as follows.

- Subject to transmission and processing delays, quality loss, and computing capacity constraints, we formulate and analyze mathematically the QLRan optimization problem in NG-RAN as a Mixed Integer Nonlinear Program (MINLP) that jointly optimizes the computational task allocation and QLR levels. The problem formulation and analysis trade off optimizing the service latency and the overall quality loss.
- The QLRan optimization problem is proved as a non-deterministic polynomial-time hard (NP-hard) problem. To solve the problem efficiently, we first relax the binary computation offloading decision variable and QLR level to real numbers. Then, we utilize the Linear Programming (LP)-based method to solve the relaxed QLRan problem by using convex optimization techniques.
- We provide a set of tools to deploy the NG-RAN mobile network. To explore the virtualization in the 5G system, we assign several OpenAirInterface (OAI) [9] containers composing of a RAN and the core of the 5G system. Specifically, we implement a programmable testbed to demonstrate a connection between UE, RAN, and Evolved Packet Core (EPC) implemented in the NG-RAN virtualization environment. The real-time experiments are carried out under various configurations in order to profile functional splitting, the data input, memory usage, and average processing time with respect to QLR levels.
- We provide formal proofs on the convergence and optimality of our algorithm and evaluate its performance under different network conditions. In terms of computing capacity and number of tasks, the numerical results show that latency can be reduced while decreasing the QLR level under practical physical constraints.

Paper Organization: The remainder of this article is organized as follows. The related work is introduced in Section 2. Section 3 includes system overview in terms of functional split options and task offloading process. We present the system model in Section 4. The QLRan problem is formulated in Section 5, followed by presenting a linear programming-based solution for QLRan optimization problem. The performance evaluation is discussed in Section 6; finally, we conclude the paper in Section 7.

2. Related work

In this section, we introduce the key concepts and papers from both industry and academia over the past several years.

2.1. Related concepts and technologies

Several cloud-based task offloading frameworks have been proposed in recent years. For example, Mobile Cloud Computing (MCC) has been proposed as a cloud-based network that can provide mobile devices with significant capabilities such as storage, computation, and task offloading to a centralized cloud [10]. However, MCC has faced several noticeable challenges to address the mobile next generation in terms of end-to-end network latency, coverage, and security. To tackle these challenges, Multi-access Edge Computing (MEC) has been introduced by European Telecommunications Standards Institute (ETSI) as an integration of the edge cloud computing systems and wireless mobile networks [11]. One of the key–value features of MEC is to enable rich services and applications in close proximity to end users. With the MEC paradigm, mobile devices have options to offload their computation-intensive tasks to a MEC server to meet the demanding Key Performance Indicators (KPIs) of 5G and beyond, especially in terms of low latency and energy efficiency. Similar to MEC systems, fog computing networks are proposed by CISCO systems to bring cloud services to the edge of an enterprise network [12]. In fog networks, the computation processing is mainly executed in the local area networks and in IoT gateways or fog nodes. Recently, the concept of NG-RAN has been defined by 3GPP as a promising approach to merge edge cloud features and RAN functionaries. In industry, many RAN organizations have made significant progress in implementing open source-software that supports NG-RAN technology. For instance, EURECOM has implemented the OpenAirInterface (OAI) platform [9], which provides an open, full software implementation of 5G and beyond systems compliant with 3GPP standards under real-time algorithms and protocols. Plus, ORAN [13], founded by AT&T, aims to drive the mobile industry towards an ecosystem of innovative, multi-vendor, interoperable, and autonomous NG-RAN with reduced cost, improved performance, and greater agility. In general, these open RAN-software projects have a high degree of flexibility, such as being able to run CU and DU entities over a fully virtual environment such as VMs or Linux containers, as

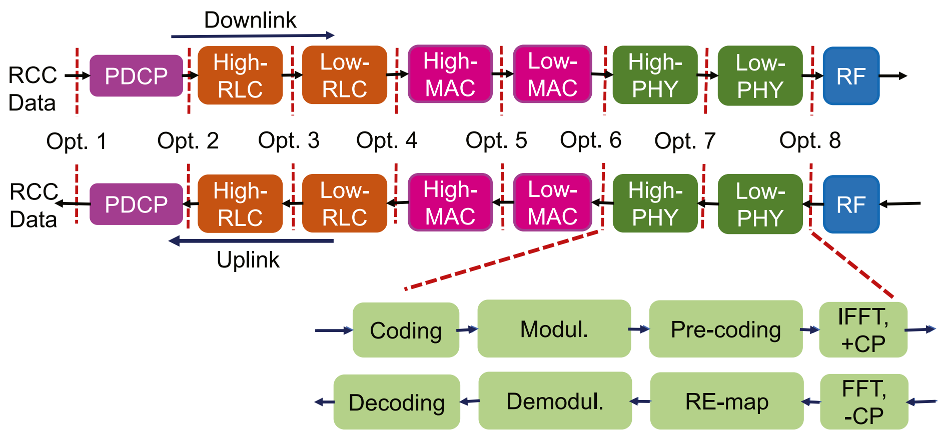

Fig. 1. Logical diagram for uplink/downlink of gNB with eight functional split options.

well as enabling promising next-generation features (e.g., network slicing and functional splitting). Such NG-RAN software will undoubtedly speed up the transition from monolithic and inflexible networks to agile, distributed elements depending on *virtualization*, *softwarization*, openness, and intelligence-fully interoperable RAN components.

2.2. Task offloading in cloud-based RANs

As part of task offloading in cloud-based RAN, several papers have focused on enhancing overall system performance in network energy, system latency, and energy efficiency. For instance, the work in [14] formulates a joint task offloading and resource allocation to maximize the users' task offloading gains in MEC. Then the main optimization problem has been decomposed into several sub-optimal problems that are solved using convex and quasi-convex optimization techniques. The authors in [15] study the energy-latency tradeoff problem for IoT partial task offloading in the MEC network by jointly optimizing the local computing frequency, task splitting, and transmit power. Then, the optimization is solved by an alternate convex search-based algorithm. In [16], by considering a cloud–fog computing network, the authors design a computation offloading algorithm to minimize total cost with respect to the energy consumption and offloading latency. To maximize the energy efficiency of task offloading, Vu et al. propose an approach based on the interior point method and bound algorithm. Exploiting machine learning methods in task offloading has also attracted several types of research in cloud-based RAN systems. Using reinforcement learning, the work in [17] introduces a MEC-based blockchain network where multi-mobile users act as miners to offload their data processing and mining tasks to a nearby MEC server via wireless channels. Although the focus of our article is in the line direction of mentioned works, applying different offloading schemes and constraints within the joint optimization NG-RAN framework could open up new, interdisciplinary avenues for researchers in the context of the 5G and beyond systems. Previously mentioned works consider a single remote server as the offloading destination. In contrast, with considering constraints on service latency, quality loss, and edge capacity, our paper proposes an algorithmic approach for latency and quality tradeoff task offloading in multi-node NG-RANs. Furthermore, our work is based on real-world NG-RAN testbed experiments that allow us to characterize the performance in terms of data input, memory usage, and average processing time with respect to QLR levels.

3. System overview

We describe here the functional split options and introduce the task allocation procedure for NG-RAN. Table 1 summarizes the key notations used.

Table 1
Summary of key notations.

Symbol	Description
$\mathcal{U}$	Set of UEs
$\mathcal{S}$	Set of edge nodes
$\mathcal{K}$	Set of computational tasks
a_{uk}	Indicator to show whether the task k is generated by UE u
a_{us}	Indicator to show whether edge node s is available for UE u
a_{sk}	Indicator to show whether task k is assigned to the edge node s
q_k	QLR level assigned to task k
$D_u(q_k)$	Input data transfer the computing task k from UE u to the edge
$C_u(q_k)$	Workload of computation to accomplish the task k
R_{us}	Transmission data rate of the link between edge node s and UE u
τ_k^{up}	Uplink transmission time
τ_k^{exe}	Execution time of task k at the edge
f_{us}	Assigned CPU-clock frequency on edge s of UE u
$B(q_k)$	Computing demanded from task k with QLR level
δ^{τ}	Weight of latency consumption time for task k
δ^{q}	Weight of QLR level for task k

3.1. NG-RAN functional split options

As a part of the NG-RAN study, 3GPP proposed several functional splits between CUs and DUs. Accordingly, it has been proposed 8 possible options shown in Fig. 1 [18]. The choice of how to split the NG-RAN architecture depends on several factors related to radio network status, traffic size and network providers' services, such as low latency, high throughput, UE density, and the geographical location of DUs. By moving from Option 1 to Option 8, a tradeoff can be established between fronthaul latency and processing complexity. Basically, by adding more baseband functions at the DUs, the required fronthaul rate can be reduced, while the processing complexity will be increased, and the energy consumption at the DUs will be increased [19]. Specifically, computationally costly operations like Fast Fourier Transformation (FFT), Inverse Fast Fourier Transformation (IFFT), Rate Matching, and Turbo encoding/decoding are shifted to the CU side, resulting in variation in energy consumption at the CU and the DU. It is worth noting that, in an actual NG-RAN testbed implementation, both the CU and DU can be implemented through virtualization. For example, in the OAI platform, each CU can be initialized by one container image and linked to a DU container image. We will provide the functional spilled-based testbed design in detail to provide real-time NG-RAN experiments.

3.2. Task allocation process

The main process of the task allocation in our proposed NG-RAN system can be summarized as follows:

1. *Edge cloud nodes:* Initially, a UE searches its communication area for the best edge cloud node to connect to. Hence, the UE will send a pilot signal and collects response from edge cloud nodes. Any edge cloud node that responds will be considered to be a potential candidate. For instance, in Fig. 2, the edge cloud candidate of the UE is the edge node within the coverage area of LTE eNB DU.
2. *Task classification:* After the edge cloud node assignment, the UE starts uploading the task information to the edge node. Some key information include; (i) the unique ID of the uploading task; (ii) the application's layers and requirements; (iii) the task profile, which include the task constraints (e.g., tolerable latency, QLR level, workload).
3. *Task executing:* After task classification, the RAP will run a resource allocation algorithm to determine: (i) the service time required for task accomplishment; (ii) computing capacity that is available for the task executing; (iii) the compare these estimates to the tasks' tolerated latency requirement.

4. System model

In this section, we describe the network setting, quality loss of result tradeoff, and task uploading model.

4.1. Network description

For the NG-RAN system model, we consider a multi-cell, multi-node edge system as illustrated in Fig. 2, in which each RAP (e.g., BS, eNodeB (eNB), gNodeB (gNB), etc.) engages with a set $\mathcal{S} = \{1, 2, \dots, S\}$ of S edge nodes (e.g., neighboring DU servers) to supply computation offloading services to the limited-resource mobile devices such as smartphones, tablets, and IoT devices. Specifically, each edge cloud node can be realized either by a physical server, or by Virtual Machine (VM)/container, which can communicate with the UE through wireless channels provided by the corresponding RAP. Plus, each UE can select to offload its computation task to an edge node from the candidate nearby severs. Accordingly, we denote the set of UEs in the mobile system and the set of computation tasks as $\mathcal{U} = \{1, 2, \dots, U\}$ and $\mathcal{K} = \{1, 2, \dots, K\}$, respectively. To define the association between UEs and RAPs, we define two binary indicators as follows, $a_{uk} \in \{0, 1\}$ is presented to indicate whether the task k is generated by UE u, while $b_{us} \in \{0, 1\}$ is presented to indicate whether edge node s is available for UE u (i.e., the edge node s has an acceptable channel state condition to be in the list of edge candidate). Hence,

$$a_{uk} = \begin{cases} 1, & k \in \mathcal{K}_u \\ 0, & \text{Otherwise} \end{cases}, \quad a_{us} = \begin{cases} 1, & s \in \mathcal{S}_u \\ 0, & \text{otherwise} \end{cases}, \tag{1}$$

where $\mathcal{S}_u \subseteq \mathcal{S}$ is denoted as the set of edge candidates for UE u, and $\mathcal{K}_u \subseteq \mathcal{K}$ is defined as the set of tasks generated by UE u. Thus, from (1), we can denote a_{sk} as a binary variable to indicate whether task k is assigned to edge node s or not. If the edge node s is available for UE u, the task k will be successfully assigned to the edge node s. Hence, a_{sk} will be satisfy the following requirement,

$$a_{sk} \leq \min\{a_{uk}, a_{us}\}, \forall u \in \mathcal{U}, k \in \mathcal{K}, s \in \mathcal{S}. \tag{2}$$

The modeling of user computation tasks, task uploading transmissions, edge computation resources, and offloading utility are presented here below.

4.2. Quality loss of result tradeoff

Many emerging applications in cloud-based computing networks (e.g., online recommender, video streaming, object recognition, and image processing) exhibit variant optional parameters that authorize end-users to take advantage of the tradeoff between QLR and service latency. For instance, many object recognition algorithms basically demand specific extraction methods of several numbers of layers with given wavelengths and orientations from image datasets for advanced analysis [20]. Hence, the achieved QLR managing the processing time in the object recognition can be relaxed if the number of extracted layers are properly adapted. Another example is multi-bitrate video streaming, in which the Over-The-Top (OTT) video content providers (e.g., YouTube, Amazon Prime, Netflix, …) offer to end-users different video quality levels to fit within the device's display and network connection [6,21]. Adjusting the video quality levels can save extra computational energy and time for OTT video providers at the same time make users experience good video watching without interruption. In this paper, we denote q_k as QLR level assigned to task k. Hence, we allow each UE u to select different q_k values to exploit the trade-off between processing cost and latency. We define QLR as five levels in which level 1 refers to the strictest demand for quality, while level 5 represents the highest tolerance for quality loss. In practice, QLR levels are determined at an application-specific level.

4.3. Task uploading

The computation task uploading in NG-RAN system can be described as a tuple of two parameters, $\langle D_u(q_k), C_u(q_k)\rangle$, where $Du(q_k)$ [bits] represents the amount of input data required to transfer the application execution (including system settings, application codes, and input parameters) from the local device to the edge node, and $C_u(q_k)$ [cycles] denotes the workload, i.e., the amount of computation to accomplish the task. Each UE $u \in \mathcal{U}$ has one computation task at a time that is atomic and cannot be divided into subtasks. The values of $D_u(q_k)$ and $C_u(q_k)$ can be obtained through carefully profiling of the task execution [14].

In Section 6, we will provide more details about the modeling of these metrics. Besides, the computation task associated with UE can be executed locally or offloaded to an edge cloud node. The Mobile device would save battery bower by offloading part of its task application to the remote edge; however, a considered cost, time and energy, from uploading the input data would be added in the task offloading scenario. Therefore, similar to [14], we consider several time parameters in the case of the UE u offloads its task k to one of the edge nodes, the overall uploading time delay consists of the follows: (i) the time τ_k^{up} [s] to transmit the input to the edge node on the uplink, (ii) the time τ_k^{exe} to perform the computation task at the edge node, and (iii) the time to bring the output data from the edge node back to UE on the downlink. In general, the size of the output data is much smaller than the input data, and the downlink data rate is much higher than that of the uplink. Therefore, similar to [14,22,23], we neglect the delay of sending the output in our computation model. Note that when the delay of the downlink transmission of output data is non-negligible, our proposed approach can still be directly applied for a given downlink rate allocation scheme and known output data size. The transmission time of UE u, that is required to send its task data input $D(q_k)$ in the uplink, can be determined as,

$$\tau_k^{up} = \frac{D_u(q_k)}{R_{us}}, \forall u \in \mathcal{U}, k \in \mathcal{K}, s \in \mathcal{S}, \tag{3}$$

where R_{us} is the transmission data rate of the link between the selected edge node s and UE u. Given the computing resource assignment, the execution time of task k at edge node s is,

$$\tau_k^{exe} = \frac{C_u(q_k)}{f_{us}}, \forall u \in \mathcal{U}, k \in \mathcal{K}, s \in \mathcal{S}, \tag{4}$$

where f_{us} denotes the assigned CPU-clock frequency on edge s to UE u of task k.

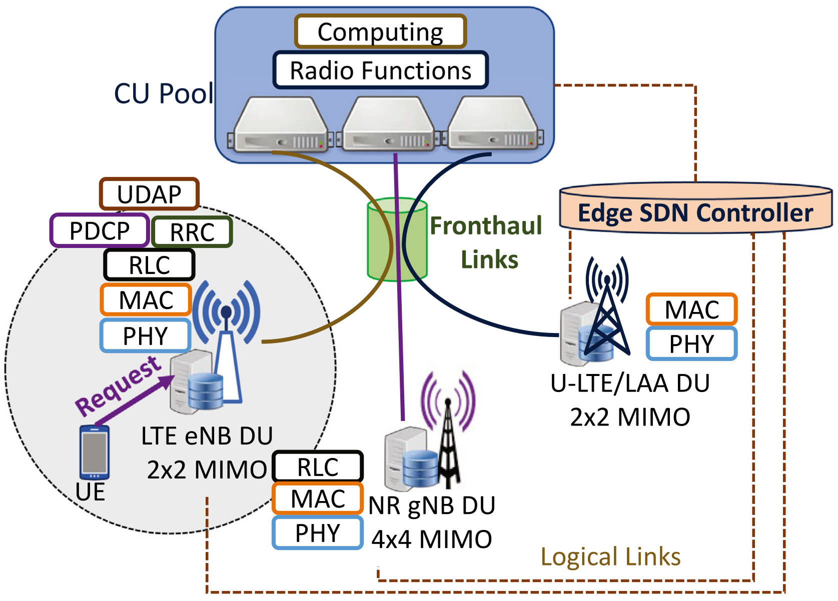

Fig. 2. System overview of QLRan, in which the gray circle represents the communication range of the RAP.

4.4. System constraints

We now introduce the following four constraints to capture the features of a task offloading multi-node NG-RAN system.

1. *QLR constraint:* As we will describe in 4.2, q_k can be modeled based on a specific key metric in an application. In our scenario, we adopt the video resolution level as q_K in the video streaming application. Under these considerations, which will be described in more details in Section 6 the QLR constraint for the task k is defined as, $q_k = \{1, 2, 3, 4, 5\}, \forall k \in \mathcal{K}$
2. *Task association constraint:* We assume each computation task of the UE must be assigned to one edge cloud node. Hence, the offloading policy would satisfy the task association constraint, expressed as,
$$\sum_{s\in\mathcal{S}} a_{sk} = 1, \forall k \in \mathcal{K}. \tag{5}$$
3. *Service latency constraint:* In many graphic applications with multiple tasks, the reduction of computation workload at the edge node considerably affects the task execution latency. For instance, real-time gaming applications have a preferred response time around 50 ms latency to enjoy a higher Quality of Experience (QoE) [24]. Achieving appropriate latency for a graphic video application demands tradeoffs processing time, uploading time, and quality. In this paper, we denote parameter $\tau_k^{\max}$ to define the maximum tolerable system latency for the task k. To guarantee that the task is accomplished in the allowed threshold time, the service latency constraint is expressed as,
$$\tau_k^{up} + \tau_k^{exe} \le \tau_k^{\max}, \forall k \in \mathcal{K}. \tag{6}$$
4. *Resource constraint:* In multi-node NG-RAN with intensive workloads, the computation capacity should be taken into account while designing a latency-quality optimization algorithm. The computation capacity could refer to several hardware metrics such as GPU, CPU, and memory. Adjusting these parameters is directly affected by the service latency and the required quality. However, the computational processing capacity at the edge cloud node cannot exceed its limited capacity. Therefore, we present the parameter, $B_s^{\max}$, as the maximum computation capacity of edge node s, while $B(q_k)$ is defined as the require computation capacity generated from processing task k at QLR q_k. Hence, the capacity constraint is model as,
$$\sum_{k\in\mathcal{K}} B(q_k) a_{sk} \le B_s^{\max}, \forall s \in \mathcal{S}. \tag{7}$$

5. Problem formation

In this section, we mathematically formulate the QLRan optimization problem, which optimizes the trade-off between the service latency and quality loss while offloading tasks in NG-RAN edge nodes. Due to the intractability of the problem and the need for a practical solution, we then present a step-by-step solution based on a linear programming-based solution, which is employed to transform the QLRan problem into a convex optimization problem.

5.1. Latency and quality tradeoffs problem

For a given $\mathcal{A} = \{a_{sk} | s \in \mathcal{S}, k \in \mathcal{K}\}$, the set of selected edge nodes, and $\mathcal{Q} = \{qk | k \in \mathcal{K}\}$, the set of selected QLR levels, we define the system utility as the weighted-sum of all the UEs' offloading utilities,

$$J_k(\mathcal{A}, \mathcal{Q}) = \delta^\tau \tau_k + \delta^q q_k \sum_{s\in\mathcal{S}} a_{sk}, \forall s \in \mathcal{S}, k \in \mathcal{K}, \tag{8}$$

where $\tau_k = \left(\tau_k^{up} + \tau_k^{exe}\right)$, $0 \le \delta^\tau \le 1$ and $0 \le \delta^q \le 1$ denote the weights of latency consumption time and QLR levels for task k, respectively. Note that we define the latency and quality tradeoffs utility, $J_k(\mathcal{A}, \mathcal{Q})$ of task k as a linear combination of the two metrics because both of them can concurrently reflect the latency-quality tradeoff of executing a task, i.e., both higher longer computation completion time and high accuracy of result lead to higher computational cost. To meet task-specific demands, we allow different UEs to select different weights, which are denoted by δ^τ and δ^q, in decision making. For example, a UE with low accuracy application demand would like to choose a larger δ^q to save more computational cost. On the other hand, when a UE is running some delay-sensitive applications (e.g., online movies), it may prefer to set a larger δ^τ to reduce the latency. We now formulate the Latency and Quality Tradeoffs (QLRan) problem as a system utility minimization problem, i.e.,

$$\mathcal{P}1 : \min_{\mathcal{A},\mathcal{Q}} \sum_{k\in\mathcal{K}} J_k(\mathcal{A}, \mathcal{Q}) \tag{9a}$$

s.t. :

$$a_{sk} \in \{0,1\}, q \in \{1,2,3,4,5\}, \forall s \in \mathcal{S}, k \in \mathcal{K}, \tag{9b}$$

$$\sum_{s\in\mathcal{S}} (\tau_k^{up} + \tau_k^{exe}) a_{sk} \le \tau_k^{\max}, \forall k \in \mathcal{K}, \tag{9c}$$

$$\sum_{k\in\mathcal{K}} B(q_k) a_{sk} \le B_s^{\max}, \forall s \in \mathcal{S}, \tag{9d}$$

$$\sum_{s\in\mathcal{S}} a_{sk} = 1, \forall k \in \mathcal{K}. \tag{9e}$$

The constraints in the formulation above can be explained as follows: constraint (9b) secure that the computation task can be accomplished in the time that cannot exceed than the demanded maximum threshold time, $\tau_k^{\max}$; constraint (9c) implies that the demand for computation capacity must not exceed its edge node capacity; finally, constraint (9d) indicates that each task must be assigned as a whole to one edge node.

Proposition 1. *$\mathcal{P}1$ is an NP-hard problem.*

Proof. To demonstrate that $\mathcal{P}1$ is an NP-hard, let first consider the case, where $\delta^\tau = 0$, $\delta^q = 1$. That means the time spent for uploading and executing a task is neglected for this case and the focusing is done only on the second part of $J_k(\mathcal{A},\mathcal{Q})$, where the QLR term is important. We assume that $\hat{q}_k$ represents the opposite value of q_k and denotes as the quality level in the result of task k. Accordingly, we can reformulate the $\mathcal{P}1$ as $\hat{\mathcal{P}}1$, in which the new objective function $\hat{J}(\mathcal{A},\mathcal{Q})$ will be maximized. Plus, constraint (9c) can be omitted for simplicity. Besides, Constraint (9d) is rewritten to imply that the resource requirement of task k is exactly equal to its quality value $\hat{q}_k$. Each edge cloud node in the NG-RAN system can only handle one task generated from the UE in the RAP coverage area. Let $\hat{a}_k$ is defended as a binary indicator to show whether the task k is assigned to the edge node, and B to denote the resource capacity of the edge node. With these considerations, the optimization problem in (9) can be relaxed as,

$$\hat{\mathcal{P}}1 : \max \sum_{s\in\mathcal{S}} \hat{q}_k \hat{a}_k \tag{10a}$$

$$\text{s.t. :} \sum_{k\in\mathcal{K}} \hat{q}_k \hat{a}_k \le B, \tag{10b}$$

$$\hat{a}_k \in \{0,1\}. \tag{10c}$$

It is obvious that problem $\hat{\mathcal{P}}1$ is a standard weighted-sum problem that is an NP-complete problem [25]. Therefore, $\mathcal{P}1$ also can be characterized as an NP-hard problem. The proof is completed. ■

Next, we will propose an iterative approach to solve $\mathcal{P}1$ based on Linear Programming-based (LP) optimization. By utilizing the standard optimization solver (e.g., MOSEK [26]), the proposed system can generate an efficient task allocation decision with an acceptable latency tolerance constraint.

5.2. Linear programming-based solution

The key challenge in solving the optimization problem in $\mathcal{P}1$ is that the integer constraints $a_{sk} \in \{0,1\}$ and $q \in [1,5]$ make $\mathcal{P}1$ a MIP problem, which is in general non-convex and NP complete [27]. Thus, similar to works in [8,28], we first relax the binary computation offloading decision variable, a_{sk}, and QLR level, q_k, to real numbers, i.e., $0 \le a_{sk} \le 1$. Then we will discuss the convexity of $\mathcal{P}1$ with the relaxed optimization variables a_{sk} and q_k. Then, we consider the following; $D(q_k) = y_d q_k + z_d$, $C(q_k) = y_t q_k + z_t$, $B(q_k) = y_b q_k + z_b$, and $x_{sk} = q_k a_{sk}$. The parameters y_d, z_d, y_t, z_t, y_b, and z_b can be estimated by offline profiling of the NG-RAN testbed, as detailed in Section 6. The LP problem for the primal problem is given by,

$$\mathcal{P}2 : \min_{\mathcal{A},\mathcal{Q},\mathcal{X},t} \delta^\tau t + \delta^q \sum_{s\in\mathcal{S}} x_{sk} \tag{11a}$$

s.t. :

$$0 \le a_{sk} \le 1, 1 \le q_k \le 5, t \le \tau_k^{max}, \forall s \in \mathcal{S}, k \in \mathcal{K}, \tag{11b}$$

$$0 \le x_{sk} \le 5a_{sk}, \forall s \in \mathcal{S}, k \in \mathcal{K}, \tag{11c}$$

$$q_k - 5(1-a_{sk}) \le x_{sk} \le q_k \forall s \in \mathcal{S}, k \in \mathcal{K}, \tag{11d}$$

$$\sum_{s\in\mathcal{S}} \left(\frac{y_d}{R_{us}} + \frac{y_t}{f_{us}}\right) x_{sk} + \left(\frac{z_d}{R_{us}} + \frac{z_t}{f_{us}}\right) a_{sk} \le \tau_k^{\max}, \tag{11e}$$

$$\sum_{s\in\mathcal{S}} a_{sk} = 1, \forall k \in \mathcal{K}. \tag{11f}$$

Proposition 2. *Constraints (11c) and (11d) can be relaxed to the constraint $x_{sk} = a_{sk} q_k$.*

Proof. Case 1: ($a_{sk} = 0$, **and** $q_k \in [1,5]$). Form constraints (11c) and (11d), we can conclude the follows,

$$x_{sk} \le 0, x_{sk} \ge 0, \text{and } x_{sk} \le q_k, x_{sk} \ge q_k - 5, \tag{12}$$

After solving (12), we can get $x_{sk} = 0$.

Case 2: ($a_{sk} = 1$, **and** $q_k \in [1,5]$).

$$x_{sk} \le 5, x_{sk} \ge q_k, \text{and } x_{sk} \ge q_k, x_{sk} \ge 0, \tag{13}$$

From (13), we can conclude $x_{sk} q_k = q_k$. From **Case 1** and **Case 2**, we demonstrate that the constraints (11c) and (11d) are equivalent to the constraint $x_{sk} = a_{sk} q_k$. The proof is complete. ■

6. Performance evaluation

In this section, we describe the testbed experiments and simulation results to provide more details about the QLR level model in terms of memory and CPU usage, as well as to test the effectiveness of the QLRan algorithm.

6.1. Testbed experiment

We present here our QLRan testbed, including the architecture, configuration, and experiment methods. Then, we analyze the performance of QLRan in terms of CPU processing time and latency.

6.1.1. Architecture

We conducted experiments on a testbed consisting of various components, i.e.,

- *End users:* For our experiment we use a Samsung Galaxy S9 running on Android 10 that acts as the UE.
- *Edge nodes:* To simulate the edge node, we use Asus Laptop equipped with an Intel Pentium III processor running Ubuntu 18.04. The cloud is represented by the more powerful desktop PC Intel Xeon E5-1650, 12-core at 3.5 GHz and 32 GB RAM.
- *Network:* The structure of OAI consists of two components: one, called *oai*, is used for building and running gNB units; the other, called *openair-cn*, is responsible for building and running the Evolved Packet Core (EPC) networks, as shown in Fig. 3. The Openair-cn component provides a programmable environment to implement and manage the following network elements: Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (SPGW-C), and PDN Gateway (PGW-U). We use WiFi as well as LTE to act as our physical link between the UE and the edge. The edge is connected to the cloud through Ethernet. As illustrated in Fig. 3, all the EPC and gNB components are implemented by as container image by using Docker and docker compose [29]. The UE and RF RAN are implemented in hardware, conventional cell phone and USRP 210, respectively.

6.1.2. NG-RAN testbed for different functional options

We endowed our testbed with several functional split options so as to realize the CU and DU in gNB. All containers in Fig. 3 are hosted by the desktop PC Intel Xeon E5-1650, 12-core at 3.5 GHz and 32 GB

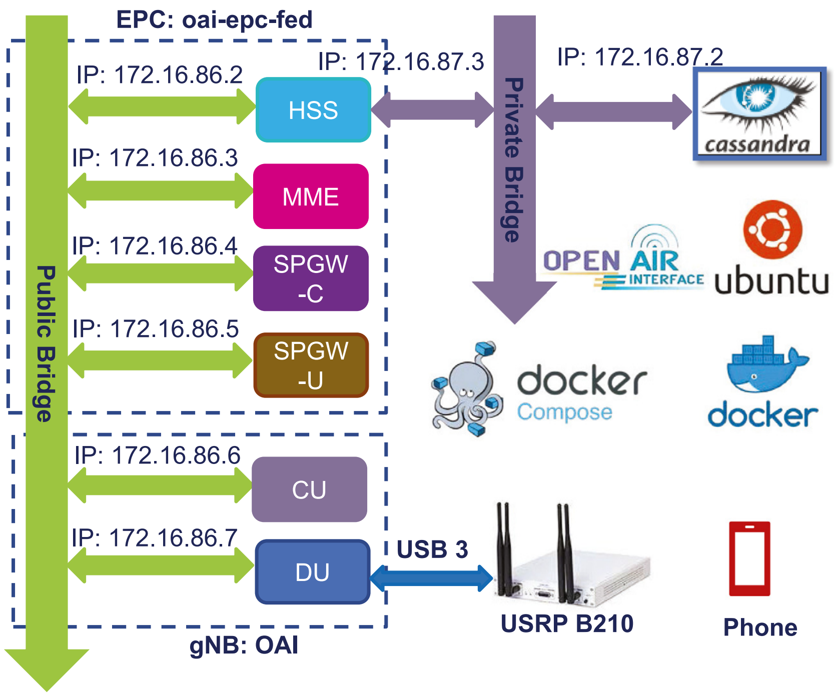

Fig. 3. Logical illustration of the fully containerized-based NG-RAN testbed.

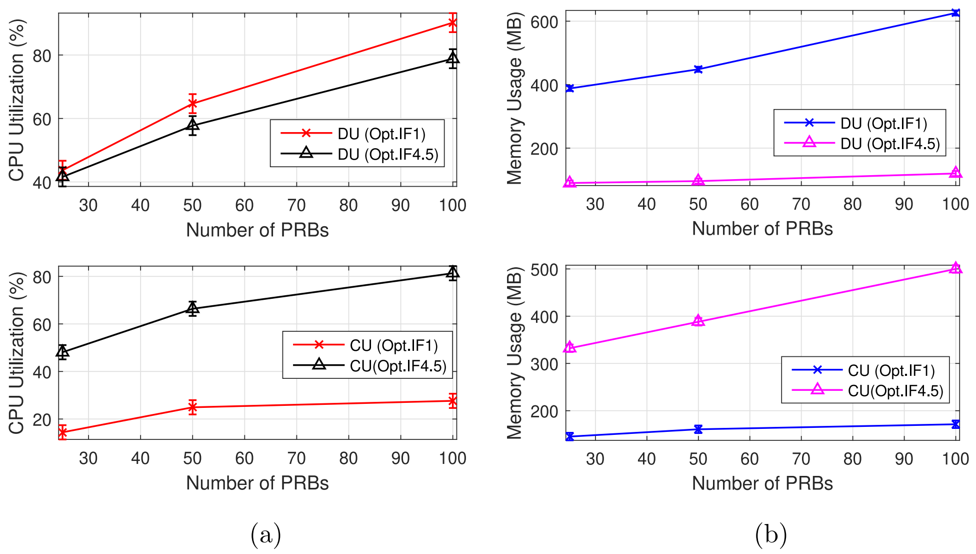

Fig. 4. (a) CPU utilization vs. number of PRBs for DU and CU in Options IF1 and IF4.5; (b) Memory usage vs. number of PRBs for DU and CU in Options IF1 and IF4.5.

RAM. For the UE, we use a Samsung Galaxy S9 running on Android 10. For network configuration, we run our NG-RAN prototype for three functional splits; Option F1, (PDCP/RLC, Option 2 in 3GPP TR 38.801 standard), Option IF4.5 (Lower PHY/Higher PHY, a.k.a Option 7.x in 3GPP TR 38.801 standard), and Option LTE eNB. We summarize the testbed configuration parameters in Table 2.

Fig. 4(a) shows the CPU utilization percentage at DU and CU containers. The CPU utilization percentage is measured by the *docker stats command* in Ubuntu, which provides a live data stream for running containers. The downlink UDP traffic repeatedly is sent from the SPGW-U container to the UE with various PRB settings in two functional split Options, F1 and IF4.5. It can be observed that the CPU consumption for DU and CU is continuing to increase linearly as the number of PRBs is increased in the two functional split options. However, Option IF1 consumes more CPU percentage at DU than at the CU. For example, the CPU utilization percentage is 43.67% in DU while it is 14.42% in CU. That is because the higher PHY operations such as RLC/MAC, L1/high, tx precode, rxcombine, and L1/low operations reside in DU for split Option IF1 [30]. In Option IF4.5, the pattern is reversed. We can see from Fig. 4(a) that the CPU usage at CU is higher than at DU. Fig. 4(c) shows the memory usage of DU and CU containers when the NG-RAN testbed performs in Options IF1 and IF4.5 at different values of PRBs. Similar to the CPU consumption pattern, the memory usage at DU is higher than at DU in Option IF1. For example, the memory usage is 388 MB in DU while it is 145.3 MB in CU at Option IF1 and 25 PRB.

6.2. *Application profiling*

To test QLRan, we consider two applications: video streaming and facial detection in smart surveillance cameras. These two tasks are both video-based tasks that require varying degrees of quality.

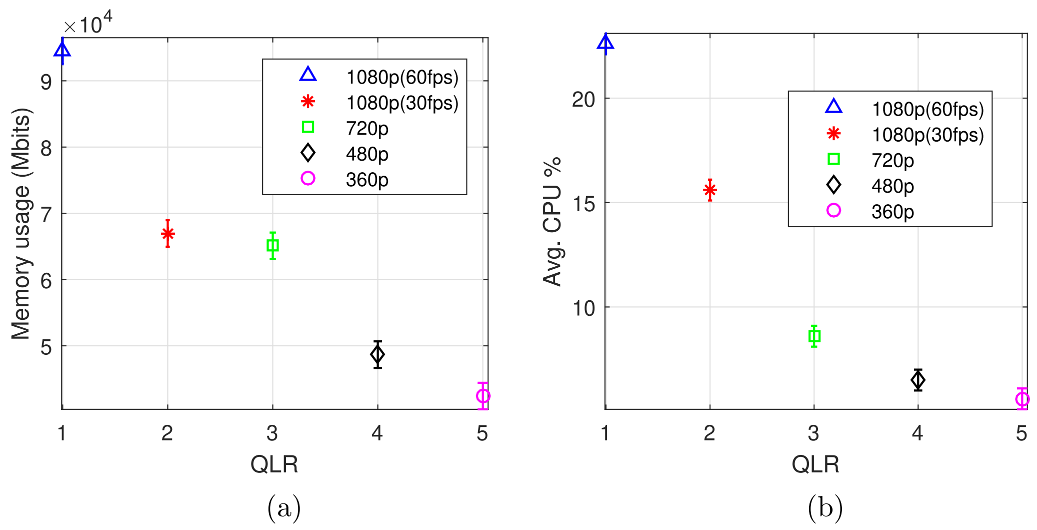

Fig. 5. (a) Memory usage for various QLR levels in video streaming; (b) CPU usage for various QLR levels in video streaming.

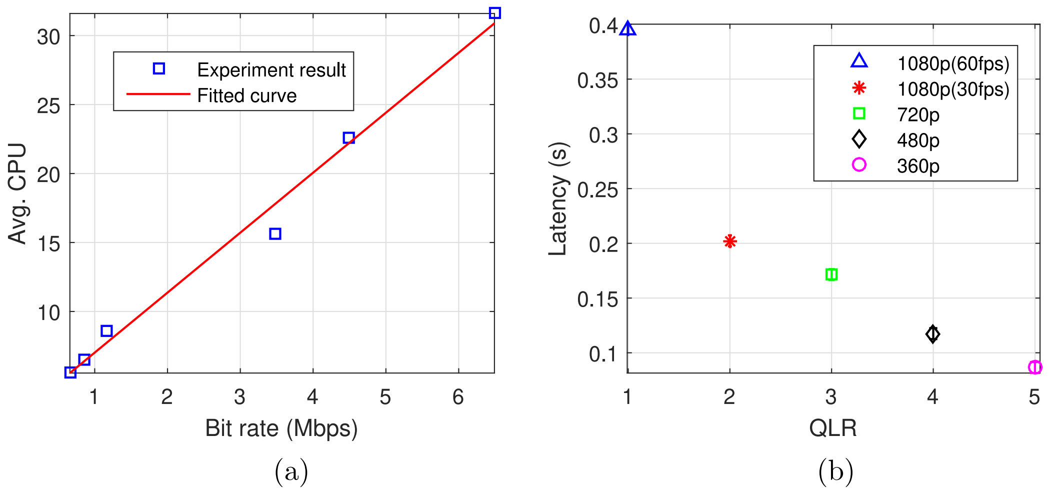

Fig. 6. (a) Relation between a video's bitrate and CPU consumption in video streaming; and (b) Latency in facial recognition.

Table 2
Testbed configuration parameters for gNB.

Mode	FDD	Options	IF1, IF4.5, eNB
Frequency	2.68 GHz	PRB	25, 50, 100
TX Power	150 dBm	Env.	Multi-container
MCS	28	SINR	15 – 20

Table 3
Configuration parameters for simulation.

y_d, z_d	4.3, 2.75	Capacity [GB]	1.5
y_t, z_t	−5.24 , 3.31	δ^r/δ^q	[50, 100, 150]
y_b, z_b	−10.41 , 95.9	Data rate [Mbps]	2
U, K, S	10, 10, 20	Delay tolerance [ms]	300
$B_s^{\max}$ [GB]	3	QLR	[1, 2, 3, 4, 5]

Video streaming application: Video streaming is run on two Dell Workstations, each with two Xeon E6-1650 processors. Each workstation is equipped with 32 GB of RAM running Ubuntu 18.04. In our experiments, a prerendered movie of one minute is streamed between these two computers using *ffmpeg*, a video transcribing and streaming application. On the other end, a *ffplay* is used to receive and render the video stream. Four different video resolutions are used: 360 × 240, 480 × 360, 960 × 720, and 1920 × 1080. Additionally for the highest resolution of 1920 × 1080, 30 fps as well as 60 fps is used as well as a stereographic stream for 60 fps for potential 3D reconstruction applications.

Facial recognition application: In addition to network streaming, a basic facial detection and recognition application is tested against the very same resolutions in the network stream. The facial recognition algorithm is based on the popular and simple *dLib library* available for python [31].

For both applications, we have chosen QLR 1 to represent the best networking conditions while a QLR of 5 represents the worst network conditions. Using the `top` utility, we were able to log in 1 second intervals the CPU consumption as well as the memory consumption of the process on the server streaming the video. Note that in both Fig. 5(a) and (b) we witness a linear increase in both memory and CPU consumption which can be expressed in the following equations,

$$B(q_k) = -10.4q_k + 95.9,\ \ C(q_k) = -5.2q_k + 33.3, \forall k \in \mathcal{K}. \tag{14}$$

In Fig. 6(a), since we downsampled the video resolutions ourselves, we are able to extract the exact average bitrate for various stream profiles to arrive at an equation,

$$D(q_k) = 4.30x + 2.75, \forall k \in \mathcal{K}, \tag{15}$$

where x represents the achievable bit rate in Mbps. Similarly—as shown in Fig. 6(b)—as video resolutions increase in facial recognition

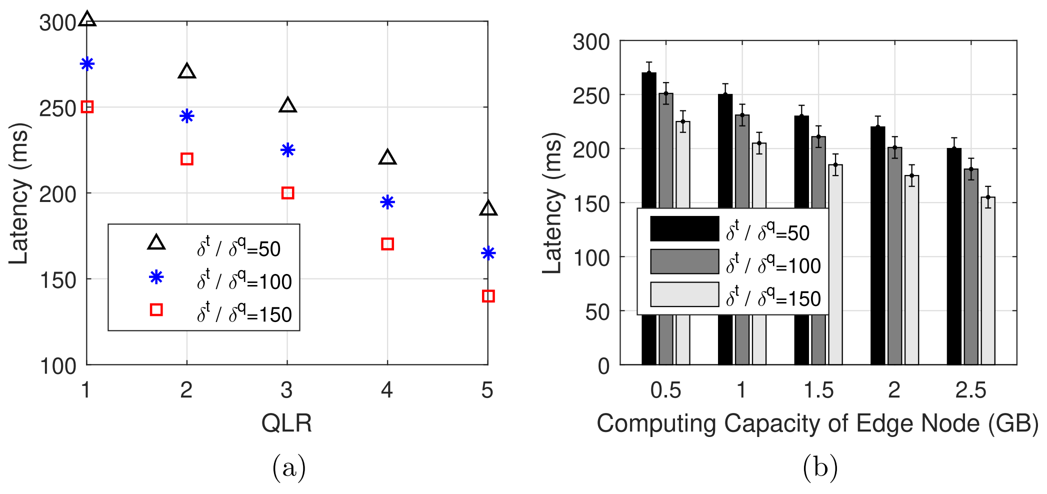

Fig. 7. System latency performance versus: (a) QLR levels; and (b) Computing capacities.

application, so does processing time. Hence, the QLR processing time can be modeled as,

$$T^{proc} = -0.08q_k + 0.51, \forall k \in \mathcal{K}. \tag{16}$$

6.3. Numerical result

We consider a NG-RAN system consisting of 100 m × 100 m cell with a RAP in the center. The mobile devices, $N = 25$, are randomly located inside the cell. The channel gains are generated using a distance-dependent path-loss model given as L [dB] $= 140.7 + 36.7\log_{10} d_{[\mathrm{km}]}$, where d is the distance between the mobile device and the BS, and the log-normal shadowing variance is set to 8 dB. The other network parameter values are listed in Table 3.

In general, the computational tasks can be classified into two different categories: (i) approximatable, tasks that can be approximated to achieve significant savings in execution time, with however a potential loss of accuracy in the result; and (ii) non-approximatable, tasks whose execution without any approximation is necessary for the success of the application, i.e., if any approximation technique were applied on these tasks, the application would not generate meaningful results. We refer the interested readers to the work in [32], which introduces a lightweight online algorithm that selects between these tasks to enable real-time distributed applications on resource-limited devices. Accordingly, we consider video streaming and facial recognition applications, which can be considered as approximatable tasks, for profiling. The reason for choosing these task applications is that they can highly benefit from the collaboration between mobile devices and edge platforms. In experimental results, we study the impact of the difference of service quality level, which can be considered as the resolution level of video streaming and facial applications, on the system latency and edge node computing capacity.

6.3.1. Impact of control parameters δ^t and δ^q

We discussed the definitions of the scalar weights δ^t and δ^q in Section 5. In general, these parameters are used to make a tradeoff between the service latency and quality. Specifically, When δ^t/δ^q is increased, the QLRan algorithm will be more sensitive to system latency; otherwise, it will be the quality of result sensitive. Fig. 7(a) shows that the latency cost decreases with a larger QLR parameter for different values of δ^t/δ^q, which are 50, 100, and 150. Specifically, the average system latency value at the δ^t/δ^q ratio is 50 and the QLR level is 1 is around 300 ms, while the average system latency values are 275 ms and 250 ms for QLR level is 1 and δ^t/δ^q ratio are 100 and 150, respectively. That is because when QLR level is 1, which refers to the best accuracy that can be obtained from processing the computational task in the edge cloud node, the computational complexity at QLRan will increase as well as the system latency. The system latency decreases with the tolerance of quality becoming low. Plus, QLRan shows good performance when the algorithm is acting towards the latency performance. For instance, when the QLR level increases to 4, the average system latency of QLRan turns down to 220 ms, 200 ms, and 180 ms, for $\delta^t/\delta^q = 50$, 100, and 150, respectively.

6.3.2. Impact of computing capacity of edge cloud node

To evaluate the offloading performance in term of memory usage, $B(q_k)$, We run the QLRan algorithm for different values of computing capacity $B(q_k)$ at δ^t/δ^q ratio of 50, 100, and 150. We observe that as long as the memory requirements are sufficient, the computing capacity (CPU/GPU) requirements can be satisfied. Hence, the performance can be evaluated with several memory sizes. As mentioned in Table 3, We set the memory size of a edge node to $B_s^{\max} = 1.5$ GB by default, while the ratio of $\delta^t/\delta^q = 50$, 100, and 150 are tuned to measure the system latency and QLR with several memory capacity values. Also, the memory size of each edge node is tuned from 0.5 to 2 GB. As illustrated in Fig. 7(b), the system latency decreases when the memory capacity of the QLRan algorithm increases. Specifically, the service latency decreases by around 12% from tuning the δ^t/δ^q ratio from 50 to 100 at computing capacity is 0.5 GB, while the overall pattern continues to decrease as the computing capacity value is increased.

6.3.3. Impact of increasing number of tasks

For the computation task, we use the face detection and recognition application for airport security and surveillance [33], which can highly benefit from the collaboration between mobile devices and edge platforms. The setting value of 12 computational tasks are selected to be in the range of 90 and 250 kB for the data size and between 890 and 1150 Megacycles for the CPU cycles. Fig. 8(a) shows the performance of different schemes versus the number of tasks. In this figure, the parameter of task data input is a random variable following linearity increasing with QLR levels. It can be seen that the case $\delta^t/\delta^q = 150$ has less latency cost compared to the other.

6.4. Comparing QLRan with other baseline approaches

We compare the QRan algorithm with the following existing benchmarks:

- Cloud Edge Executing Only (CEO): Each UE $u \in \mathcal{U}$ has only one option: to offload its task to cloud edge node within its communication coverage without considering the tradeoff between latency and approximate computing;

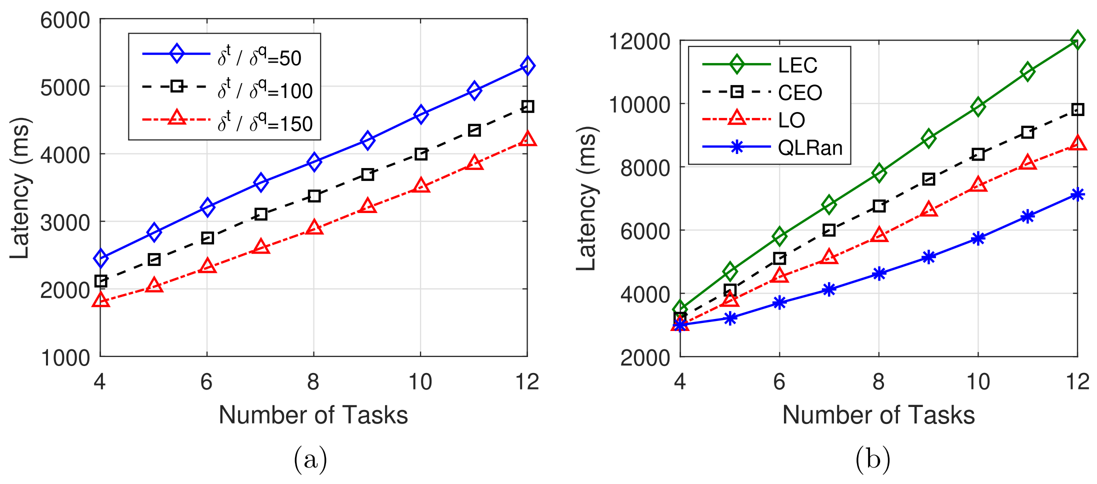

Fig. 8. (a) System latency performance versus number of computational tasks; and (b) System latency versus number of computation tasks under different execution schemes.

- Local Executing Only (LEO): Each UE $u \in \mathcal{U}$ has only one option: to execute its task locally within its communication coverage without considering the tradeoff between latency and approximate computing;
- Latency-aware ask Offloading (LO): Each UE $u \in \mathcal{U}$ can offload its task to edge cloud within its communication coverage. Here, only the latency is considered in the objective function, while approximate computing is ignored.

As illustrated in Fig. 8(b), we evaluate the running performance of 12 computational tasks under different offloading schemes. Our joint latency and quality-aware offloading scheme outperforms other schemes. Specifically, the performance gap between QLRan and other schemes increases when the number of task increases. That is because the QLRan algorithm is designed to trade off between the latency and QLR level, while the other schemes only focus on the offloading and executing scenarios.

7. Conclusions

We presented latency-quality tradeoffs and task offloading in multi-node next-generation RANs. We designed our algorithm, QLRan, to reduce system service latency while adjusting the overall quality level. Practical NG-RAN system constraints have been considered to formulate the proposed task offloading problem. The constraints depend on network latency, quality loss, and edge node computing capacity, while the objective function is the weighted sum of all the UEs' offloading utilities. The QLRan is cast as an NP-hard problem; therefore, we propose a Linear Programming (LP)-based approach that can be solved via convex optimization techniques. Simulation results are generated from running several real-time applications on the NG-RAN testbed, which is completely implemented under container-based virtualization and functional-split option technologies. We considered video-streaming and facial-recognition applications as building blocks of many cloud-based applications. We evaluated our solution and thorough simulation results showed that the performance of the QLRan algorithm significantly improves the network latency over different configurations.

Declaration of competing interest

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

References

[1] A. Younis, B. Qiu, D. Pompili, QLRan: Latency-quality tradeoffs and task offloading in multi-node next generation RANs, in: Proc. IEEE/IFIP WONS, 2021, pp. 1–8.

[2] A. Younis, T. Tran, D. Pompili, Energy-efficient resource allocation in C-RANs with capacity-limited fronthaul, IEEE Trans. Mob. Comput. 20 (2) (2021) 473–487.

[3] I.A. Alimi, A.L. Teixeira, P.P. Monteiro, Toward an efficient C-RAN optical fronthaul for the future networks: A tutorial on technologies, requirements, challenges, and solutions, IEEE Commun. Surv. Tutor. 20 (1) (2017) 708–769.

[4] A. Younis, T.X. Tran, D. Pompili, Fronthaul-aware resource allocation for energy efficiency maximization in C-RANs, in: Proc. IEEE ICAC, 2018, pp. 91–100.

[5] 3GPP TS 38.300 V2.0.0, NR; NR and NG-RAN overall description; Stage 2, 2017, Release 15.

[6] A. Younis, T.X. Tran, D. Pompili, On-demand video-streaming quality of experience maximization in mobile edge computing, in: Proc. IEEE WoWMoM, 2019, pp. 1–9.

[7] Y. Li, Y. Chen, T. Lan, G. Venkataramani, MobiQoR: Pushing the envelope of mobile edge computing via quality-of-result optimization, in: Proc. IEEE ICDCS, 2017, pp. 1261–1270.

[8] A. Younis, T.X. Tran, D. Pompili, Energy-latency-aware task offloading and approximate computing at the mobile edge, in: Proc. IEEE MASS, 2019, pp. 299–307.

[9] EURECOM, OAI, 2020, Available: http://www.openairinterface.org/.

[10] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: Architecture, applications, and approaches, Wirel. Commun. Mob. Comput. 13 (18) (2013) 1587–1611.

[11] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, et al., MEC in 5G Networks, ETSI White Paper 28, 2018, pp. 1–28.

[12] CISCO, Fog Computing and the Internet of Things: Extend The Cloud to Where the Things Are, Whit Paper, 2015, pp. 1–6.

[13] O-RAN alliance, O-RAN Use Cases and Deployment Scenarios, White Paper, 2020.

[14] T.X. Tran, D. Pompili, Joint task offloading and resource allocation for multi-server mobile-edge computing networks, IEEE Trans. Veh. Technol. 68 (1) (2018) 856–868.

[15] M. Qin, N. Cheng, Z. Jing, T. Yang, W. Xu, Q. Yang, R.R. Rao, Service-oriented energy-latency tradeoff for IoT task partial offloading in MEC-enhanced multi-RAT networks, IEEE Internet Things J. 8 (3) (2021) 1896–1907.

[16] Z. Zhao, S. Bu, T. Zhao, Z. Yin, M. Peng, Z. Ding, T.Q. Quek, On the design of computation offloading in fog radio access networks, IEEE Trans. Veh. Technol. 68 (7) (2019) 7136–7149.

[17] D.C. Nguyen, P.N. Pathirana, M. Ding, A. Seneviratne, Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2536–2549.

[18] 3GPP TR 38.801, Study of new radio access technology: Radio access architecture and interfaces, 2017, Release 14.

[19] L. Wang, S. Zhou, Flexible functional split and power control for energy harvesting cloud radio access networks, IEEE Trans. Wirel. Commun. 19 (3) (2019) 1535–1548.

[20] V. Kshirsagar, M. Baviskar, M. Gaikwad, Face recognition using eigenfaces, in: Proc. IEEE ICCRD, 2011, pp. 302–306.

[21] I. de Fez, R. Belda, J.C. Guerri, New objective QoE models for evaluating ABR algorithms in DASH, Elsevier Comput. Commun. 158 (2020) 126–140.

[22] X. Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2014) 974–983.
[23] X. Lyu, H. Tian, C. Sengul, P. Zhang, Multiuser joint task offloading and resource optimization in proximate clouds, IEEE Trans. Veh. Technol. 66 (4) (2016) 3435–3447.
[24] Y.W. Bernier, Latency compensating methods in client/server in-game protocol design and optimization, in: Game Developers Conference, Vol. 98033, 2001.
[25] C.-P. Schnorr, M. Euchner, Lattice basis reduction: Improved practical algorithms and solving subset sum problems, Math. Program. 66 (1–3) (1994) 181–199.
[26] MOSEK Aps, The MOSEK optimization toolbox v 9, 2019.
[27] E.D. Andersen, K.D. Andersen, Presolving in linear programming, Math. Program. 71 (2) (1995) 221–245.
[28] T.Q. Dinh, J. Tang, Q.D. La, T.Q. Quek, Offloading in mobile edge computing: Task allocation and computational frequency scaling, IEEE Trans. Commun. 65 (8) (2017) 3571–3584.
[29] Docker, 2021, https://docs.docker.com/.
[30] OAI tutorials, 2021, https://gitlab.eurecom.fr/oai/openairinterface5g/blob/develop/doc/FEATURE_SET.md#enb-phy-layer.
[31] D.E. King, Dlib-ml: A machine learning toolkit, J. Mach. Learn. Res. 10 (2009) 1755–1758.
[32] P. Pandey, D. Pompili, Exploiting the untapped potential of mobile distributed computing via approximation, Pervasive Mob. Comput. 38 (2017) 381–395.
[33] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, W. Heinzelman, Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture, in: Proc. IEEE ISCC, 2012, pp. 59–66.

Ayman Younis received the B.Eng. and M.Sc. degrees in Electrical Engineering from the U. of Basrah, Iraq, in 2008 and 2011, respectively. He is pursuing the Ph.D. degree in ECE at Rutgers, NJ, USA, under the guidance of Dr. Pompili. His research focuses on wireless communications and mobile cloud computing, with emphasis on software-defined testbeds. He received the Best Paper Award at the IEEE/IFIP Wireless On-demand Network Systems and Services Conference (WONS) in 2021.

Brian Qiu obtained a M.S. in the Electrical and Computer Engineering (ECE) at Rutgers University, NJ, in 2021, where he also received his B.S. in 2019. He is interested in distributed mobile computing and in general in networks with a focus on anonymity and privacy.

Dario Pompili is an associate professor with the Dept. of ECE at Rutgers University. Since joining Rutgers in 2007, he has been the director of the CPS Lab, which focuses on mobile edge computing, wireless communications and networking, acoustic communications, and sensor networks. He received his Ph.D. in ECE from the Georgia Institute of Technology in 2007. He had previously received his 'Laurea' (combined BS and MS) and Doctorate degrees in Telecommunications and System Engineering from the U. of Rome "La Sapienza," Italy, in 2001 and 2004, respectively. He has received a number of awards in his career including the NSF CAREER'11, ONR Young Investigator Program'12, and DARPA Young Faculty'12 awards. In 2015, he was nominated Rutgers-New Brunswick Chancellor's Scholar. He served on many international conference committees taking on various leading roles. He published about 200 refereed scholar publications, some of which received best paper awards: with more than 13K citations, Dr. Pompili has an h-index of 44 and an i10-index of 111 (Google Scholar, Oct'21). He is a Fellow of the IEEE Communications Society (2021) and a Distinguished Member of the ACM (2019). He is currently serving as Associate Editor for IEEE Transactions on Mobile Computing (TMC).