QLRan: Latency-Quality Tradeoffs and Task Offloading in Multi-node Next Generation RANs

Ayman Younis, Brian Qiu, and Dario Pompili
Department of Electrical and Computer Engineering
Rutgers University–New Brunswick, NJ, USA
E-mails: {a.younis, brian.qiu, pompili}@rutgers.edu

***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, and edge capacity, 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 virtualized using the OpenAirInterface (OAI) software platform to 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.**

***Index Terms*—NG-RAN; Tasks Offloading; Convex optimization; OpenAirInterface (OAI); Testbed.**

I. 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 processing using mobile platforms 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 (EC) concept has emerged, which aims to unite telco, IT, and cloud computing to provide cloud services directly from the network edge. EC nodes are implemented directly at the cellular Base Stations (BSs) of a Radio Access Network (RAN), or at the local wireless Access Points (APs) using a generic-computing platform. This way, the edge node allows for the execution of applications in close proximity to end users, which reduces the end-to-end (e2e) delay and the costly backhaul bandwidth consumption. Recently, Cloud Radio Access Network (C-RAN) [1] has been introduced as a clean-slate redesign of the cellular architecture where some to all of the 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 base band 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 [2]. Also, a fully centralized architecture imposes high capacity requirements on fronthaul links [3]. Therefor, Next Generation RANs (NG-RAN) [4] 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, 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 EC [5]—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 much higher computation capability than the mobile device. Therefore, enabling task offloading in NG-RAN can be proposed to address limitations (e.g., storage and computing resources) in the existing RANs. Meanwhile, in some cases, processing the entire input data in EC servers would require more than the available computing resources to meet the desired latency/throughput guarantees. In the context of IoT applications, transferring, managing, and analyzing large amounts of data in an EC would be prohibitively expensive. According to [6], [7], by relaxing the tolerance of quality loss, service latency can be reduced. We explore a novel dimension, Quality Loss of Results (QLR), to represent the user service with suboptimal networking conditions. *Our key idea is motivated by the observation that in many edge applications like media processing, machine learning, and data mining, an exact result is not always necessary, and a suboptimal result is acceptable.* As a consequence, 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: Unlike the traditional approaches mentioned above, (i.e., full centralized/distributed RAN), our objective is to design a holistic optimization solution for latency-quality aware joint task offloading 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. The benefits brought by a multi-edge node NG-RAN system over the single-edge NG-RAN (aka single-cloud) system are multi-fold: (i) firstly, edge nodes may be overloaded when serving a large number of offloading UEs. One can reduce overloaded servers by redirecting some UEs to offload to the neighboring node edges from the nearby RAPs, thus preventing the resource-limited edge node from becoming the bottleneck; (ii) secondly, any UE can opt to offload its task to the RAP with a more favorable uplink channel condition, saving transmission energy consumption; (iii) finally, coordination of resource allocation to offload UEs across multiple neighboring RAPs can help mitigate the effect of interference and resource contention among users and hence, improve offloading gains when multiple UEs offload their tasks simultaneously. *In this paper, a Latency and Quality tradeoffs task offloading problem, called (QLRan), is designed to optimize service latency and QLR under application-specific requirements, e.g., communicating and computing 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.

Related Works: The NG-RAN paradigm has attracted considerable attention in both academia and industry over the past several years. For instance, the open source FlexRAN platform [8] is designed to activate the dynamic functional splitting option in NG-RAN. ORAN [9], 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. Meanwhile, some recent works focused on the energy consumption and offloading decision problem to minimize an execution delay. For instance, in [10], the authors formulate an optimization model for NG-RAN, called Apt-RAN, to optimize the energy consumption of the Central Unit (CU) pool and the number of handovers, considering different functional splits. The work in [11] minimizes the execution delay by one-dimensional search algorithm, which finds an optimal offloading decision policy according to the application buffer queuing state and characteristic of the channel between the user and the EC server.

The computation offloading decision that minimizes the energy consumption at the user while satisfying the execution delay of the application was proposed in [12]–[14]. The optimization problem in [12] was formulated as a Constrained Markov Decision Process (CMDP) while a decision on the computation offloading in [13] done periodically in each time slot, during which all the users are divided into two groups: the first user group is allowed to offload computation to the EC server while the second group has to perform computation locally due to unavailable computation resources at the EC server. For the femto-cloud computing systems, the cloud server is jointly optimized the femto access points, the transmit power, precoder and computation load distribution in [14].

Existing work has shown the feasibility to save energy and/or reduce latency by relaxing the workload computation accuracy requirement [6], [15], [16], either via optimization and software-based techniques [6], [15], or hardware-based approximation techniques [16].

Although the focus of our paper is orthogonal to this line of work, considering different workload partitioning techniques within the joint optimization framework, what we propose in QLRan opens up interesting avenues for researches in the context of NG-RAN. Most of the 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 allowed us to characterize the performance in terms of data input, memory usage, and average processing time with respect to QLR levels.

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.

- Subjecting to transmission delay, processing delay, quality loss, and computing capacity constraints, we mathematically formulate and analyze the QLRan optimization problem in NG-RAN as a Mixed Integer Nonlinear Program (MINLP) problem that jointly optimizes the computational task allocation, and QLR levels. The problem formulation and analysis trade off in 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 a NG-RAN mobile network. To explore the virtualization in the 5G system, we assign several OpenAirInterface (OAI) [17] 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 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 term of computing capacity and number of tasks, the numerical results show

that the latency can be reduced, while decreasing the QLR level under practical physical constraints.

Paper Organization: The remainder of this paper is organized as follows. We present the system model in Sect. II; the QLRan problem is formulated in Sect. III, followed by presenting a linear programming-based solution for QLRan optimization problem; the performance evaluation is discussed in Sect. IV; finally, we conclude the paper in Sect. V.

II. System Model

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

A. Network Description

We consider a multi-cell, multi-node edge system as illustrated in Fig. 1, in which each RAP (e.g., BS, eNodeB (eNB), gNodeB (gNB), etc.) engages with a set $\mathcal{S} = \{1, 2, .., S\}$ of S edge nodes (e.g., neighboring DU servers) to provide computation offloading services to the resource constrained mobile devices such as smart phones, tablets, and wearable devices. In general, each edge node can be either a physical server, a Virtual Machine (VM) or a container with moderate computing capabilities provisioned by the network operator and can communicate with the UE through wireless channels provided by the corresponding RAP. Each UE can choose to offload computation tasks to a edge node from one of the nearby RAPs it can connect to. We denote the set of UEs in the mobile system and the set of computation tasks as $\mathcal{U} = \{1, 2, ..., U\}$ and $\mathcal{K} = \{1, 2, ..., K\}$, respectively. We denote $a_{uk} \in \{0, 1\}$ to indicate whether the task k is generated by UE u, and $b_{us} \in \{0, 1\}$ is defined to indicate whether edge node s is available for UE u (i.e., the edge node s is in the list of edge candidate). Then, we have,

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

where $\mathcal{S}_u \subseteq \mathcal{S}$ is defined 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. Furthermore, a binary variable a_{sk} is defined to indicate whether task k is assigned to edge node s. The task k will be successfully assigned to the edge node s only if the edge node s is available for the generator of the task k. Therefore,

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

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

B. Quality Loss of Result Tradeoff

Many emerging applications in cloud-based computing networks such as recommendation, data mining, object recognition, media (e.g., video and image) processing and data analytics expose different control parameters that allow end-users to exploit the tradeoff between QLR and network latency. For example, object recognition algorithms [18] often require extraction of a given number of layers with different

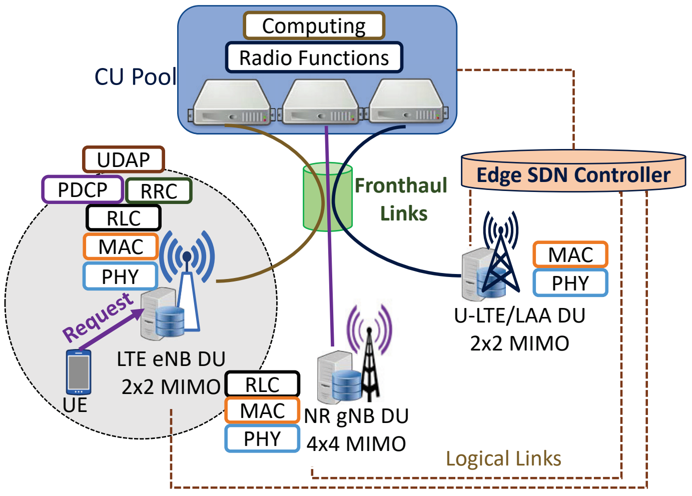


Fig. 1. System overview of QLRan, in which the gray circle represents the communication range of the RAP.

wavelengths and orientations from the original input images for analysis. Therefore, by adjusting the number of layers extracted, the achieved QLR which controls the processing time in the object recognition can be relaxed. In recommendation algorithms [19], the amount of reference data used to make the recommendation can serve as the control knob for achieving this tradeoff [15]. These methods are referred to as parameter level substitution in prior work. Other techniques of relax computing include discarding or substituting certain different subsets of tasks in non-relax computation, to achieve varied QLR-latency tradeoff. 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 exploiting 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, the set of measuring metric QLR for an application is calculated for each application domain.

C. Task Uploading

We consider that each UE $u \in \mathcal{U}$ has one computation task at a time that is atomic and cannot be divided into subtasks. By taking into account the user acquired service quality (i.e., quality degradation), we characterize each computation task k as a tuple of two parameters, $\langle D_u(q_k), C_u(q_k)\rangle$, in which $Du(q_k)$ [bits] specifies the amount of input data necessary to transfer the program execution (including system settings, program codes, and input parameters) from the local device to the edge node, and $C_u(q_k)$ [cycles] specifies the workload, i.e., the amount of computation to accomplish the task. The values of $D_u(q_k)$ and $C_u(q_k)$ can be obtained through carefully profiling of the task execution [20]. In Sect. IV, we will provide more details about the modeling these metrics. Each task can be performed locally on the UE or offloaded to a edge node. By offloading the computation task to the edge node, the

UE would save its energy for task execution; however, it would consume additional time and energy for sending the task input.

In case UE u offloads its task k to one of the edge nodes, the incurred delay comprises: (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 execute the task at the edge node, and (iii) the time to transmit the output from the edge node back to UE on the downlink. Since the size of the output is generally much smaller than the input, plus the downlink data rate is much higher than that of the uplink, we omit the delay of transferring the output in our computation, as also considered in [21], [22]. Note that, when the delay of the downlink transmission of output data is non-negligible, our proposed algorithm can still be directly applied for a given downlink rate allocation scheme and known output data size. Given limited bandwidth, the transmission delay depends on the size of data to be transmitted.

The transmission time of UE u when sending its task input $D(q_k)$ in the uplink can be calculated 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.

D. System Constraints

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

1) *Quality loss constraint:* As we described in II-B, q_k can be defined based on video resolution, e.g., in the case of video streaming. Under these considerations, which will be described in more details in Sect. IV the QLR constraint for the task k is represented as, $q_k = \{1, 2, 3, 4, 5\}, \forall k \in \mathcal{K}$
2) *Assignment constraint:* One computational task is supposed to be the basic unit for task allocation. Hence, the computational task must be assigned to one node node,
$$\sum\nolimits_{s \in \mathcal{S}} a_{sk} = 1, \forall k \in \mathcal{K}. \tag{5}$$
3) *Service latency constraint:* The maximum tolerable service latency for an AR navigation application is 250 ms [23]. Whereas, the maximum tolerable service latency for video streaming application can be as much as 1 sec. We use parameter $\tau_k^{\max}$ to denote the maximum tolerable service latency for the task k. In order to ensure that the task can be completed in time, the service constraint is as,
$$\tau_k^{up} + \tau_k^{exe} \leq \tau_k^{\max}, \forall k \in \mathcal{K}. \tag{6}$$
4) *Capacity constraint:* The demand for capacity (i.e., GPU, CPU, and memory) is affected by the service latency and the expected quality. The total demand received by a edge node cannot exceed its capacity. We defined $B_s^{\max}$ as the capacity of edge node s, and $B(q_k)$ as the demand from task k with QLR q_k. Hence, the capacity constraint is model as,
$$\sum\nolimits_{k \in \mathcal{K}} B(q_k) a_{sk} \leq B_s^{\max}, \forall s \in \mathcal{S}. \tag{7}$$

III. 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 linear programming-based solution, which is employed to transform the QLRan problem to convex optimization problem.

A. Latency and Quality Tradeoffs Problem

For a given $\mathcal{A} = \{a_{sk} | s \in \mathcal{S}, k \in \mathcal{K}\}$,the 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,

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

where $\tau_k = (\tau_k^{up} + \tau_k^{exe})$, $0 \leq \delta^\tau \leq 1$ and $0 \leq \delta^q \leq 1$ 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.,

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

s.t :

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

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

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

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

The constraints in the formulation above can be explained as follows: constraint (9b) ensures that the task can be completed in time that cannot exceed than required maximum time deadline,$\tau_k^{\max}$; constraint (9c) implies that the demand for capacity (i.e., GPU, CPU, and memory) is affected by the service latency and the expected quality. Therefore, the total demand received by a edge cannot exceed its capacity, $B_s^{\max}$. where $B(q_k)$ is defined as the demand, memory usage, from task k with QLR level q_k; finally, constraint (9d) implies that each task must be assigned as a whole to one edge node.

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

Proof. A special case, where $\delta^\tau = 0$, $\delta^q = 1$, is considered, which means that the goal is to minimize the sum of QLR levels. Here, $\hat{q}_k$ is defined as the quality gain in the result of task k. It represents the opposite of QLR. Therefore, the goal of minimizing the sum of QLR levels can be transformed into maximizing the sum of quality gains. For simplicity, Constraint (9c) is removed. In addition, Constraint (9d) is relaxed by assuming that the resource requirement of task k is exactly equal to its quality gain $\hat{q}_k$. Besides, we assume that each UE can generates only one task and one edge node in each coverage area. We define $\hat{a}_k$ to indicate whether the task k is assigned to the edge node, and B to represent the resource capacity of the edge node. Hence, The optimization problem in (9) can be simplify as,

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

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

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

Problem $\widehat{\mathcal{P}}1$ is a classic subset sum problem that has proved to be an NP-complete problem [24]. Hence, $\mathcal{P}1$ can be classified as NP-hard problem. The proof is completed. □

Next, we will propose a approach to solve $\mathcal{P}1$ based on Linear Programming-based (LP) optimization. With the help of the linear programming solver (e.g., MOSEK [25]), the system can make an efficient task allocation decision with a balanced multi objective optimization in a short time.

B. 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 [26]. Thus, similar to works in [7], [27], 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 Sect. IV.

The LP problem for the primal problem is given by,

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

s.t :

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

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

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

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

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

Proposition 2. *Constraints* (11c) *and* (11d) *are equal 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 prove that the constraints (11c) and (11d) are equal to the constraint $x_{sk} = a_{sk} q_k$. The proof is complete. □

IV. Performance Evaluation

Testbed experiments and simulation results are provided here to show the effectiveness of the QLRan algorithm.

A. Testbed Experiment

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

1) Architecture: We conducted experiments on a testbed consisting of several 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 a 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 mainly 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. 2(a). The Openair-cn component provides a programmable environment to implement and manage the following network elements: Mobility Management Entity (MME), Home Subscriber

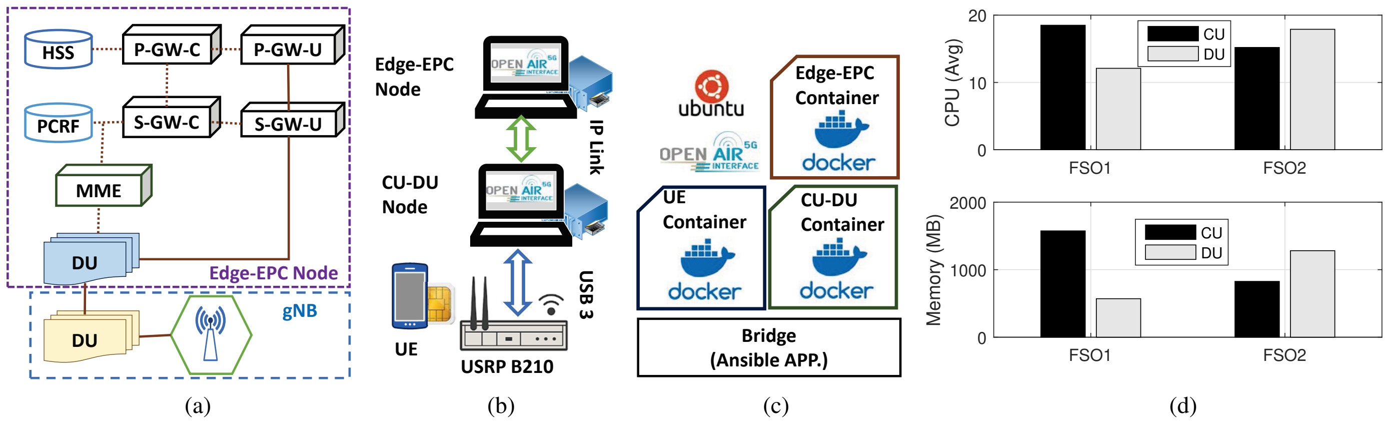


Fig. 2. (a) NG-RAN configuration; (b) OAI testbed setup; (c) OAI-emulation setup; and (d) CPU-Memory utilization versus different functional splitting options.

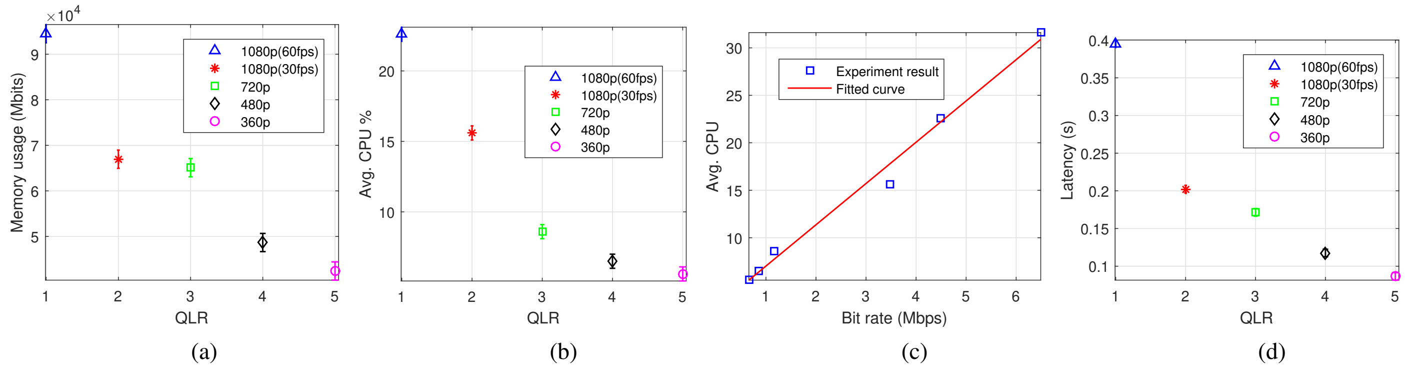


Fig. 3. (a) Memory usage for Various QLR levels in video streaming; (b) CPU usage for Various QLR levels in video streaming (c) Relation between a video's bitrate and CPU consumption in video streaming; and (d) Latency in facial recognition.

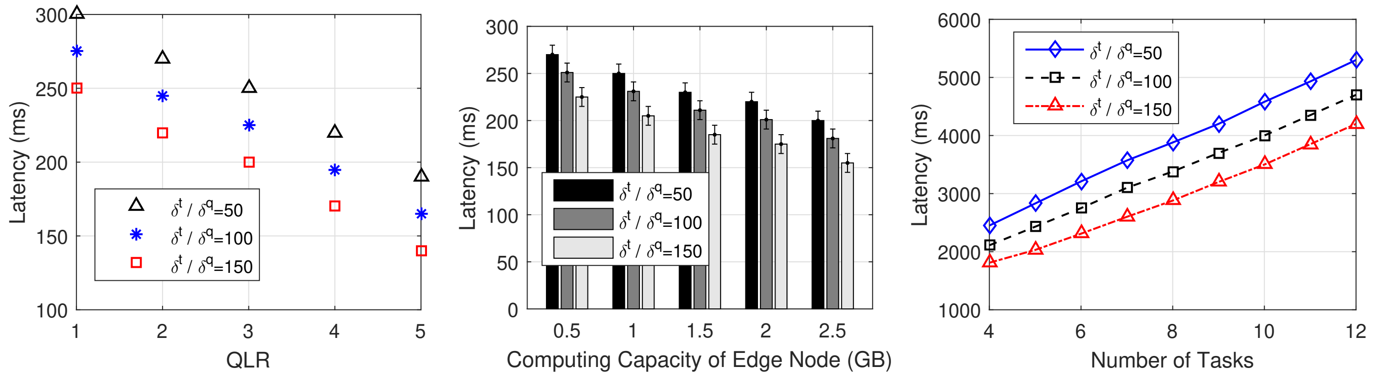
Server (HSS), Serving Gateway (S-GW-C), PDN Gateway (P-GW-C), and Policy and Charging Rules Function (PCRF). We use WiFi as well as LTE to act as our physical link between the UE and the edge. The edge is connected to the cloud through Ethernet. As shown in Fig. 2(b), the UE and RAN are implemented in hardware, conventional cell phone, and USRB 210, while EPC core is implemented in software.

2) Function Splitting Options (FSOs): In this part setting, We create an operational gNB, i.e., the main element of a RAN, based on the NG-RAN architecture and using open-source software OAI emulation, in which all RAN components are deployed using Docker containers that can be hosted on cloud infrastructure as show in Fig 2(c). Specifically, we run two instances of *lte-softmodem* which split the base station into two roles. One of these splits is based on the Remote Radio Unit (RRU)/Radio Cloud Center (RCC) split from C-RAN architecture. The other split is closer to the L1 using the F1 interface. In both configurations, we use a real UE to simulate traffic. CPU averages are recorded over the course of a minute. Initial memory footprints are taken in Fig. 2(d). We observe that the CU consumes more CPU and memory in FSO 1 when compared to the DU. However, the trend inverses for FSO 2. The CPU and memory consumption is less in the CU than the DU. This is because in FSO 2, PDCP is running in the CU while the heavier functions (e.g., RLC, MAC,and RF) are running in the DU. One of the key observations from these experiments is that the CPU consumption of the CU is reduced by nearly 21% when we move from Option 1 to Option 2, as lower PHY layer functions such as FFT and IFFT are moved to DU side. However, higher PHY operations like turbo encoding/decoding operations still reside in the CU for both the splits. Similarly, nearly 47% of the CU memory footprint is reduced for Option 1 when compared to Option 2.

B. 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 qualities.

Video streaming application: Video streaming is run on two Dell Workstations each with two Xeon E6-1650 processors. Each workstation is equipped with 32GB 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



(a) (b) (c)

Fig. 4. Service latency performance versus: (a) Scalar weight; (b) Computing capacity; and (c) Number of computational tasks.

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 of the popular and simple *dLib library* available for python [28].

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 Figs. 3(a) and 3(b) we witness a linear increase in both memory and CPU consumption which can be expressed in the following equations,

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

In Fig. 3(c), 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,

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

where x represents the achievable bit rate in Mbps. Similarly—as shown in Fig. 3(d)—as video resolutions increase in facial recognition application, so does processing time. Hence, the QLR processing time can be modeled as,

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

C. Numerical Result

We consider a NG-RAN system consisting of $100\ \mathrm{m} \times 100\ \mathrm{m}$ 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 $L[\mathrm{dB}] = 140.7 + 36.7 \log_{10} \mathrm{d}_{[\mathrm{km}]}$, where d is the distance between the mobile device and the BS, and the log-normal shadowing variance is set to $8\ \mathrm{dB}$. The other network parameter values are listed in Table I.

TABLE I
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^τ/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	$[1, 2, 3, 4, 5]$

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 [29], which introduces a light-weight 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 consider 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 platform. In the experiments, the impact of the variation of service quality, which can be defined in video streaming and facial recognition applications as increasing the resolution levels, on the service latency and the amount of resource consumption are explored. $f_{us} = 1000\ \mathrm{Megacycles}, \forall \mathrm{u} \in \mathcal{U}, \mathrm{s} \in \mathcal{S}$.

1) Impact of Parameters δ^t and δ^q: As mentioned in Sect. III, the scalar weights δ^t and δ^q refer to the optimization tendency toward service latency and quality, respectively. When δ^t/δ^q is higher, the task allocation strategy is latency sensitive; otherwise, it is quality sensitive. For QLRan optimization problem, we tune the ratio of the scalar weights, δ^t/δ^q, from 50 to 150, and compare the results in Fig. 4(a). When the density of edge nodes is high, as illustrated in Fig. 4, the average service latency in case of QLRan algorithm is around 300 ms when the QLR level is 1 and $\delta^t/\delta^q = 50$. The service latency decreases with the tolerance of quality loss.

Besides, QLRan show good performance when the algorithm is acting towards the latency optimization. For example, when the QLR level increases to 4, the average latency of QLRan would drop up to 220 ms,200 ms, and 180 ms, for $\delta^t/\delta^q = 50$, 100, and 150, respectively.

2) Impact of Computing Capacity of Edge Nodes: For the two tasks we have tested, memory usage, $B(q_k)$, becomes a performance bottleneck. We have noticed that computing capacity requirements (e.g., CPU/GPU) can be satisfied as long as the memory requirements are met. Therefore, the performance is evaluated with varying memory sizes. As shown in Table I, the memory size of a edge node node is set to $B_s^{\max} = 1.5$ GB by default, with $\delta^t/\delta^q = 50$, 100, and 150 are chosen as examples to evaluate the service latency and QLR with different memory capacities settings. The memory size of each edge node is tuned from 0.5 to 2 GB. As shown in Fig. 4(b). When the memory capacity of QLRan increases, the service latency decreases. In details, the service latency decreases by around 12%.

3) Impact of Increasing Number of Tasks: For computation task, we consider the face detection and recognition application for airport security and surveillance [30], which can highly benefit from the collaboration between mobile devices and edge platform. Fig. 4(c) 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 a linearity increasing with QLR levels. It can be seen that the case $\delta^t/\delta^q = 50$ has better performance compared to the other.

V. Conclusions

In this paper, we highlight QLRan algorithm, latency-quality tradeoffs and task offloading in multi-node next generation RANs. Our algorithm is designed to minimize average service latency while reducing the overall quality loss. We considered the constraints on service latency, quality loss, and edge node capacity and formulate the task allocation process as a objective optimization problem, where a tradeoff is maintained between the service latency and quality loss. As it is proved to be an NP-hard problem, we propose a Linear Programming (LP)-based approach that can be later solved by using convex optimization techniques. For computation task, we consider video streaming and facial recognition applications which can be the building blocks of many cloud-based applications. We evaluate our solution with simulation results show that the performance of QLRan algorithm significantly improves the network latency over different configurations.

Acknowledgment: This work was supported by the US National Science Foundation under Grant No. ECCS-2030101.

References

[1] A. Younis, T. Tran, and D. Pompili, "Energy-efficient resource allocation in C-RANs with capacity-limited fronthaul," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 473–487, 2021.

[2] I. A. Alimi, A. L. Teixeira, and P. P. Monteiro, "Toward an efficient C-RAN optical fronthaul for the future networks: A tutorial on technologies, requirements, challenges, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 708–769, 2017.

[3] A. Younis, T. X. Tran, and D. Pompili, "Fronthaul-aware resource allocation for energy efficiency maximization in C-RANs," in *proc. IEEE ICAC*, pp. 91–100, 2018.

[4] 3GPP TS 38.300 V2.0.0, "NR; NR and NG-RAN overall description; Stage 2," *Release 15*, 2017.

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

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

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

[8] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *Proc. CoNEXT*, pp. 427–441, 2016.

[9] O-RAN alliance, "O-RAN use cases and deployment scenarios," *White Paper*, 2020.

[10] H. Gupta, M. Sharma, B. R. Tamma, *et al.*, "Apt-RAN: A flexible split-based 5G RAN to minimize energy consumption and handovers," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 473–487, 2019.

[11] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE ISIT*, pp. 1451–1455, 2016.

[12] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *Proc. IEEE ICC*, pp. 5529–5534, 2015.

[13] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Joint allocation of computation and communication resources in multiuser mobile cloud computing," in *Proc. IEEE Workshop SPAWC*, pp. 26–30, 2013.

[14] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, 2015.

[15] P. Pandey and D. Pompili, "MobiDiC: Exploiting the untapped potential of mobile distributed computing via approximation," in *Proc. IEEE PerCom*, pp. 1–9, 2016.

[16] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: A cache for approximate computing," in *Proc. MICRO*, pp. 50–61, 2015.

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

[18] V. Kshirsagar, M. Baviskar, and M. Gaikwad, "Face recognition using eigenfaces," in *Proc. IEEE ICCRD*, pp. 302–306, 2011.

[19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. ACM WWW*, pp. 285–295, 2001.

[20] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, 2014.

[21] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, 2014.

[22] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3435–3447, 2016.

[23] M. Noreikis, Y. Xiao, and A. Ylä-Jaäiski, "Qos-oriented capacity planning for edge computing," in *Proc. IEEE ICC*, pp. 1–6, 2017.

[24] C.-P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Mathematical programming*, vol. 66, no. 1-3, pp. 181–199, 1994.

[25] MOSEK Aps, *The MOSEK optimization toolbox v 9*, 2019.

[26] E. D. Andersen and K. D. Andersen, "Presolving in linear programming," *Mathematical Programming*, vol. 71, no. 2, pp. 221–245, 1995.

[27] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, 2017.

[28] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[29] P. Pandey and D. Pompili, "Exploiting the untapped potential of mobile distributed computing via approximation," *Pervasive and Mobile Computing*, vol. 38, pp. 381–395, 2017.

[30] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proc. IEEE ISCC*, pp. 59–66, 2012.