Edge Gaming: a Greening Perspective[⋆]

Francesco Spinelli[a,b,*], Antonio Bazco-Nogueras[a], Vincenzo Mancuso[a]

[a] *IMDEA Networks Institute, Avda. del Mar Mediterraneo, 22, 28918 Madrid, Spain*
[b] *Universidad Carlos III de Madrid, Avda. Universidad, 30, 28911 Leganés, Madrid, Spain*

Abstract

We tackle the problem of how to support gaming at the edge of the cellular network. The reduced latency and higher bandwidth that the edge enjoys with respect to cloud-based solutions implies that transferring cloud-based games to the edge could be a premium service for end-users. The goal of this work is to design a scheme compatible with MEC and network slicing principles of 5G and beyond, and which maximizes the utility of a service/infrastructure provider with time-varying edge node capacities due to the access to intermittent renewable energy. We formulate a *multi*-dimensional integer linear programming problem, proving that it is NP-hard in the strong sense. We prove that our problem is sub-modular and propose an efficient heuristic, `GREENING`, which considers the allocation of gaming sessions and their migration. For the mentioned scenario, we analyze a wide variety of realistic configurations at the edge, studying how the performance depends on *i)* whether the games have a static or dynamic workload, *ii)* the distribution of renewable energy through nodes and time, or *iii)* the topology of the edge network. Through simulations, we show that our heuristic achieves performance close to that achieved by solving the NP-hard optimization problem, except with extremely lower complexity, and performs up to 25% better than state-of-the-art algorithms.

Keywords: MEC, Edge computing, Edge gaming, Sustainable computing, Renewable energy, beyond 5G, Resource allocation and migration

1. Introduction

Cloud gaming allows users to play newest-generation games requiring only an internet connection and a screen (e.g., a TV screen, laptop, or mobile phone) by leveraging on a cloud infrastructure. Games are located and processed in a cloud server, which streams the content to the end-user screen. Initial attempts were unsuccessful (e.g., OnLive) mainly due to the lack of infrastructure, but nowadays cloud gaming is having a second life. It is a growing market—reaching by 2023 a total revenue of 8 billion dollars [2])—and many tech companies are launching cloud gaming services within their network infrastructure, for example Google with Google Stadia, NVidia with Geforce Now, and AWS with Amazon Luna, to name just a few. At the same time, one of the main features of 5G and beyond 5G networks is the ability to place computing resources closer to end-users, at the edge of the network. In this scenario, paradigms such as Multi-access Edge Computing (MEC) [3] arose, bringing forth novel use cases or *verticals* [4]. MEC gives the opportunity to exploit cloud gaming at the edge, developing the concept of *edge gaming*.

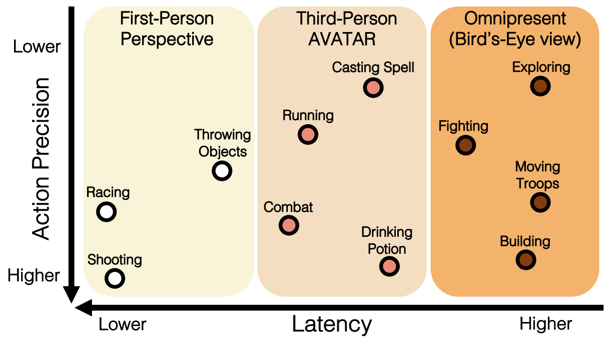


Figure 1: High level example of latency requirements for different game actions.

We can highlight three main benefits of edge gaming with respect to cloud gaming: *i)* notably reduced latency, which makes the gaming experience immersive and interactive, with different nuances, as shown in Figure 1; edge gaming will enable in particular fast-paced games, where timing is fundamental—e.g. First Person Shooter (FPS)—and competitive online multiplayer gaming. *ii)* At the same time as allowing for tighter latency requirements, allocating games at the edge will reduce network core congestion [5], therefore reducing the possibilities of packet losses while allowing higher video quality. Finally, *iii)* the edge gaming paradigm could easily leverage the roll out of new networking principles such as network slicing and

[⋆]This work is supported in part by the Regional Government of Madrid through the grant 2020-T2/TIC-20710 for Talent Attraction and by the TAPIR-CM project (S2018/TCS-4496). A preliminary conference version has been published at IEEE WoWMoM'22 [1].

[*]Corresponding author
Email addresses: `francesco.spinelli@imdea.org` (Francesco Spinelli), `antonio.bazco@imdea.org` (Antonio Bazco-Nogueras), `vincenzo.mancuso@imdea.org` (Vincenzo Mancuso)

standardization bodies such as ETSI MEC [3].

However, constrained edge resources are scarce and variable, and new techniques should be proposed to efficiently provision resources to meet several quality and cost constraints.

In particular, it is expected that energy will become a crucial bottleneck for the deployment of this kind of systems [6], and there is a growing interest in making infrastructures more sustainable by leveraging renewable energies [7, 8]. Indeed, edge nodes and datacenters could be endowed with their own sources of renewable energy, which means that MEC nodes would have time-varying capabilities depending on the fluctuating energy resources, rising the possibility to migrate resources across several edge nodes if necessary. This is particularly true for gaming applications, as games have a highly dynamic behavior in terms of both workload and instantaneous resource requirements [9]. As a matter of example, we can imagine that the workload required for the same game differs when it renders a static scenario with respect to the case where the scenario quickly changes (e.g., where the character moves); such change of requirements may happen also if the frame rate has changed from 30 to 60 fps.

Games could exploit migration at the edge because migration delays are negligible in such scenario due to the proximity of edge servers in residential areas [4] and new efficient migration techniques that can be exploited [10, 11, 12]. Hence, the possibility of migrating online gaming servers in few milliseconds would enable seamless game sessions and therefore an increased QoE for end-users even in a dynamic scenario with twofold variability—in games requirements and nodes capabilities. In such (possibly unsteady) scenario, resource allocation and migration becomes one of the most challenging tasks.

Main Contributions

Motivated by the aforementioned, we focus on the problem of resource allocation in a *sustainable edge-based online gaming* scenario, which we refer to as **green edge gaming**. In this scenario, we aim at maximizing the utility of the system taking into account revenue and costs of energy, deployment and migration. Accordingly, we develop a smart allocation and migration algorithm of online game sessions under several realistic constraints.

The main contributions of our article are as follows.

- We develop the concept of green edge gaming with time-varying edge node capabilities due to the fluctuating availability of renewable energy. We show that this concept is compliant with ETSI MEC standardization and with modern networking principles such as network slicing.
- We argue that this concept could lead to a *premium* business scenario for which we formulate an accurate *multi*-dimensional linear integer programming problem, showing that it is NP-hard in the strong sense and sub-modular.
- We develop `GREENING`, an efficient online heuristic for game session allocation and migration which maximizes the utility by maximizing the use of renewable energy (green energy) instead of the one proceeding from polluting sources (brown energy).
- We study the proposed algorithm in realistic settings, where (*i*) the amount of available renewable energy is obtained from a database of real values for solar and wind energy generation whose average levels depend on the time of the day; (*ii*) maximum capabilities of edge nodes are taken from actual commercial equipment; (*iii*) resource requirements of game sessions can be either static (e.g., as an approximation for their maximum possible values) or dynamically change at a fast pace, in accordance with real measures on online gaming. We are the first modeling and studying the job's dynamics in a gaming scenario at the edge.
- We evaluate the proposed algorithm against several benchmarks, and we show it achieves near-optimal performance without high complexity. The results show that the proposal obtains values up to 25% better than state-of-the-art approaches.

A preliminary version of our work has appeared in [1]. In this article, we provide new insights on realistic dynamic aspects, and in particular on green energy dynamics and dynamic gaming workloads, and explore a much richer set of operational scenarios, therefore offering a more comprehensive overview of the green edge gaming paradigm. Please note that the `GREENING` algorithm proposed in this article represents an evolution of the preliminary `GREENING` algorithm presented in [1].

Article Organization: The rest of this article is organized as follows. We provide an overview of previous related works in Section 2, and we define the system model in Section 3. In Section 4, we formulate an instantaneous optimization problem, proving its NP-hardness and its submodularity. In Section 5, we tackle the more general online problem of green game session allocation. Section 6 highlights our main results and finally Section 7 presents our concluding remarks.

2. Related Work

We present in the following an overview of the related works, with an emphasis on cloud gaming, edge gaming, and edge resource allocation and migration, as well as a brief description of the differences of our work with respect to the literature.

Cloud gaming: Cloud gaming has been well studied and is becoming a reality, with some projections highlighting that 20% of gaming sessions will be soon on cloud [6].

Several recent papers addressed this paradigm, focusing especially on: *i)* server allocation, *ii)* costs and *iii)* QoS guarantees. In [13], the authors study the server provisioning problem for cloud gaming with the double goal of reducing server running and software storage costs. Similarly, the authors in [14] propose several heuristic algorithms to solve the problem with both server allocation costs (for server renting fees and data transfer) and the bandwidth costs, taking into account real-world latency constraints, while Wu *et al.* [15] design an online control algorithm to reduce both latency (focusing especially on queuing delay) and server provisioning costs.

Some works study resources utilization, using real testbeds: Li *et al.* [16] focus on minimizing resources usage when interference between co-located games at the same server happens (i.e., decreasing QoS). According to [9], games could fall into two categories: CPU-critical and memory-input-output critical. Therefore, they propose several task scheduling strategies in order to optimize resource allocation. In [17], the authors propose a framework called T-Gaming that uses off-the-shelf consumer GPUs, prioritized video encoding, and adaptive real-time streaming based on deep reinforcement learning with the goal of reducing hardware and network costs.

Other works analyze the resources placement problem. For example, Hong *et al.* [18] study the virtual machine (VM) placement problem for maximizing both the profit for service providers and the overall gaming QoE for end-users while, in [19], the authors propose a distributed algorithm to optimize VM placement in mobile cloud gaming through resource competition.

Finally, only few papers consider energy in cloud gaming systems. At high level, the authors in [20] discuss green energy solutions for cloud gaming while in [21], Chuah *et al.* propose a control algorithm to decrease GPU power consumption while guaranteeing the Service-Level Agreement (SLA).

Edge gaming: Compared to cloud gaming, edge gaming is a newer concept tied up with edge computing. Indeed, edge computing could help the cloud gaming paradigm in both storage [22] and computation (by offloading tasks [23] or rendering whole games), minimizing the overall response time. In [5], the authors leverage edge servers to offload computation intensive tasks for gaming, showing that this strategy could reduce network delay and bandwidth consumption. Yates *et al.* [24] develop a Markov model to optimize frame rate and lag synchronization of server and player in low-latency edge cloud gaming systems, employing an age of information metric to characterize the system performance.

In an edge computing scenario with constrained computing resources, migrations between different servers are both possible (due to the proximity and the fast connection between nodes) and necessary (because of the limited amount of resources and the variability of the scenario), which opens new scenarios with multiple challenges. Braun *et al.* [11] propose a new migration protocol to migrate a MEC gaming application through different edge servers while in [12] the authors have developed Talaria, an in-engine content synchronisation solution. The latter allows for unnoticeable game instance migration between edge servers, which is mandatory in order to maintain a satisfactory QoE for end-users in fast-paced games.

Edge resource allocation and migration: Allocation and migration of resources is a well-known problem, which has been studied especially for centralized cloud systems [25]. Notwithstanding, edge nodes could be constrained for instance in bandwidth, energy, storage, or computing capabilities, and it is difficult to apply the same strategies designed for cloud systems into an edge scenario due to these multiple constraints on the edge resources.

Among the works that consider this challenge in edge networks, [26] proposes a joint service placement and request scheduling scheme, while [27] proposes a randomized rounding technique for the joint optimization of service placement and request routing in a MEC network. Both papers consider several constraints on edge nodes. Other papers provide solutions from a different perspective, and they make use of a machine learning approach; for example, Wang *et al.* [28] propose a joint task offloading and migration schemes in a mobility-aware MEC network. Their scheme is based on Reinforcement Learning (RL) and their goal is to obtain the maximum system utility minimizing the migration costs. In [29], the authors use a deep learning framework for proactive migration (based on service replication) of MEC resources in a 5G vehicle scenario, with the goal of minimizing the total energy expenditure, without considering hardware limitations such as on memory and CPU cycles. In [30], the authors leverage on deep reinforcement learning to minimize the average completion time of tasks under migration energy budget, while in [31] the authors investigate the task migration issue for multiple UAVs in the MEC-based UAV delivery system. Specifically, they study an energy-aware decision-making strategy for the dynamic task migration in order to optimize the UAV energy consumption. Wang *et al.* [32] propose a Markov decision process framework for dynamic service migration in order to follow users movement, without considering edge nodes capabilities. In [33], the authors propose a resource-aware VM migration technique, without taking into account energy consumption, while the authors in [34], on the opposite, focus on a VM migration mechanism that is aware and adapts to the fluctuating available green energy, minimizing therefore the energy consumption from non-renewable sources, but without considering other constraints.

Novelty of our work: To the best of our knowledge, the *green edge gaming* scenario, in which games are allocated in nodes with time-varying capacity and in the presence of both fluctuating energy and workloads and of several nearby edge nodes, has not yet been analyzed. In this scenario migration of tasks may play a significant role, since they can not only reduce costs and pollution, but also avoid congestion in the network core, with improved QoE for end-users.

Many works in the cloud gaming area have tackled QoS or QoE metrics (on delay and bandwidth especially), although they do not incorporate other aspects as storage, energy, or computation limitations, since these constraints are usually not challenging in a cloud infrastructure due to higher amount of resources. However, they are indeed very important for the edge infrastructure. In this work, we attempt to consider all the types of resources that may become the bottleneck in the green edge gaming scenario, namely bandwidth, delay, storage, node computation capabilities, and energy available. Besides, we consider a twofold dynamic setting, where we consider that both the capacity of the nodes (due to variable amount of renewable energy) and the requirements of the jobs (due to the inherent varying nature of games specifications) are time-varying, which has not been evaluated before in edge/cloud gaming scenarios.

Most of previous works focused on a subset of the constraints here considered and the migration of tasks, if taken into account, was mainly based on user's mobility. We will show in this work that migrations have a fundamental role independently of user mobility. We assume that the migration of already-on-the-system jobs can be performed within nearby edge servers for two main reasons: first, to optimize the use of energy and other resources, but also to make space in the system for newly arrived jobs while respecting all the considered constraints. This is only possible in the edge context, since migrations cannot be considered in legacy cloud gaming contexts [18]. Finally, we are the first ones that study the jobs' dynamics in an edge gaming scenario, exposing how resources should be delicately allocated at the edge with jobs having dynamic workload. In this scenario, migrations are necessary in order to maintain a high QoE for end-users.

3. System Model

Figure 2 shows a schematic of our reference scenario. We model a layered 5G edge network infrastructure [35] containing a set of edge game servers, with a set of network links connecting these servers between them and with the end-users. We denote the set of network links by $\mathcal{Z}$, and the size of this set as $Z = |\mathcal{Z}|$. The set of edge computing servers is denoted by $\mathcal{N}$ and it is composed of N servers. We consider two types of servers, which differ in their capabilities and proximity to end-users. Among the N nodes, B servers reside on *far-edge nodes*, each deployed at a base station (BS), and primarily meant to serve users of that BS, whereas the rest of the servers are each located at a different *M1 node*, placed in the edge/transport network where the traffic of multiple BSs converges [35]. M1 nodes have bigger capacities compared to edge nodes, in terms of computation, energy, and memory capabilities, although these capabilities must be shared across users belonging to multiple cells.

We distinguish two different types of energy powering the servers, whether it comes from renewable (green) and non-renewable (polluting) energy sources. Regarding the green energy availability, we restrict ourselves to locally

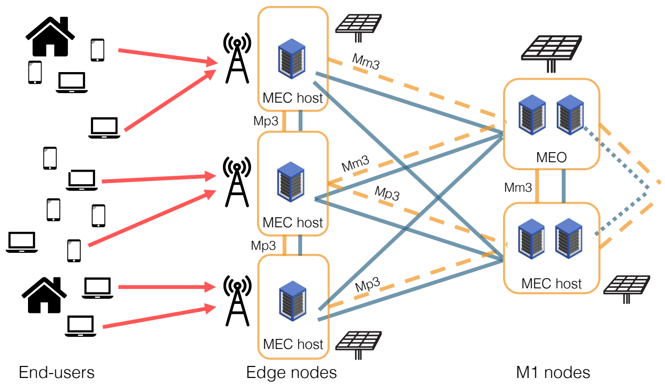


Figure 2: 5G Edge Infrastructure compliant with ETSI MEC.

generated energy, and thus we only consider wind and solar as renewable sources, which have already been applied in edge computing contexts [36]. Non-renewable energy is always available at M1 nodes, while edge nodes might not have access to it. When a server can only access green energy, its computing capacity is proportional to the available green energy. In all cases, we assume that green energy can be used at no cost, but brown energy has a non-negligible cost.

The infrastructure is used to run online game sessions, each of which is referred to as a *job*. The operation time is slotted, and we consider that the game sessions' requirements are random variables that may follow different distributions. The jobs arriving over time are modeled by a set $\mathcal{J}$. The operator accepts to process the job in exchange to a monetary payment, such that a certain job j provides a revenue R_j to the operator that includes many factors, e.g. user's fees, percentages of game purchases, advertising, etc. At the same time, jobs incur a cost of deployment, management, and processing which depends on the amount of computation, memory, and communication resources, the energy required, and the duration of the job, which are variable quantities that evolve over time. Furthermore, jobs interruption (due to shortage of energy) or migrations also incur a certain non-negligible cost.

In the following, we explain in detail the time-slotted operation of the system, and the statistical model considered for energy fluctuations, jobs requirements, and nodes capabilities.

3.1. Resource allocation for Green Edge Gaming

The system operates in a time-slotted manner. We consider a centralized decision maker that is aware of the state (in terms of capability and load) of each server. At the beginning of each time slot, there exists a set of newly arrived job requests, as well as another set of jobs that are already being served. Furthermore, the amount of renewable energy at each node and the energy and computation requirements of each job may vary from one time slot to the next one. The system's task is to migrate ongoing jobs, interrupt them, and accept and allocate current requests with the goal of optimizing the utility of the scenario. The optimization is applied each time slot in which something has changed, i.e., upon new jobs arrive, the available level of green energy changes, or job requirements change. Note that considering dynamic energy levels and workloads is challenging because past optimal job allocations might soon turn into non-optimal and call for reconfiguration at a frequent pace. We will analyze different settings in which the frequency of these changes varies.

The network does not know the future duration of each job, as game sessions have an unknown duration in nature. However, we assume that at each time slot the decision maker knows the jobs requirements (bandwidth, delay, memory, computation, and energy) for the starting slot. This assumption can model a scenario where the network can estimate and/or predict with high enough precision the average consumption of a certain job based on the information available (type of game and device, previous values, etc.) and the considered optimization time slot is short enough, e.g., a few tens of seconds for a game whose computing and rendering power typically change significantly only upon significant changes of scene.

3.2. Energy fluctuation model

We consider that each edge node is equipped with on-site renewable energy sources. In particular, we consider that each edge node has installed a personal-use-size windmill and a one-square-meter solar panel. Both energy generators amount to a total maximum capacity of 1.5 kW, as per specifications of current commercial devices.[1]

Due to the unpredictability of wind/solar resources [37], we model the green energy behavior in a stochastic manner [38]. We make use of the dataset provided by a Belgian operator called Elia to create samples that match the trends and randomness of green energy generation in a real power grid. In particular, Elia provides weekly forecasts of both wind [39] and solar [40] energy generation in Belgium, with a granularity of 15 minutes, and we have used the data generated for the period from 21st to 27th of March 2022 for specific areas in Belgium.

In addition to the most probable forecast, the dataset provides confidence intervals. We will use such information to generate random realizations of energy forecast that conform to daily changes of green energy availability. Figure 3 shows a weekly solar and wind power generation forecast together with confidence intervals, with data taken from the dataset made available by Elia. We can observe that the amount of available green energy varies considerably depending on the time of the day but also between different days. For simplicity, we do not consider the use of long-duration batteries at edge nodes, and therefore green energy is not stored from time slot to time slot.

[1]E.g., see *Tumo-Int* 1000 *W Vertical Wind Turbine Generator* or *GONGJU* 1000 *W Vertical Axis Wind Turbine Generator* for windmills, and *Jinko TIGER Pro 545W* or *Longi Hi-MO 4 455W* for solar panels.

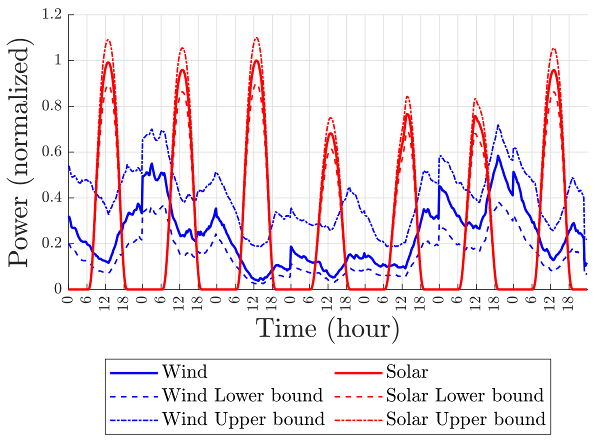


Figure 3: Weekly solar and wind power generation forecast provided by Elia for Flanders (wind data) and for a federal region of Belgium (solar data), from the 21st to the 27th of March, 2022. Values reported in the figure are normalized to the solar peak average expected on the third day (about 3 MW for the entire region to which the dataset applies). In this article, this forecast dataset was rescaled to account for the fact that only a limited number of solar panels and a windmill can be mounted at an edge node, and used to produce the numerical results presented in Section 6.

3.3. Job monetization and cost

Each accepted job brings a revenue R_j to the operator, which may depend on the requirements of the job. At the same time, jobs require an operating cost that is proportional to such requirements. In particular, the total cost associated to job j, which is denoted by C_j, is composed of a cost of deployment, the cost of the non-renewable energy consumption, the possible cost from migrating the job, and the interruption cost.

a) Deployment cost. It represents the cost of instantiating and deploying resources to support the accepted jobs, and it is denoted by $C_j^{(d)}$.

b) Energy cost. This cost is proportional to the amount of polluting energy consumed by the job at each time unit, and hence is not constant over time. It is denoted by $C_j^{(b)}$.

c) Migration cost. The migration cost represents the induced operating cost derived from re-deploying, rescheduling and migrating resources among nodes within the edge network.

For simplicity, we do not consider migrations triggered by handovers as an optimization problem. We do so not only because that topic has been covered in other studies [11], but also because we are interested in the evaluation of interactive game sessions, which are typically several minutes long and are played at home or in a static environment [41].

d) Interruption cost. A job interruption leads to a loss of performance and the termination of the user experience. Consequently, the cost associated to interruptions

($C_j^{(p)}$) is considerably higher than that of migrations and can take out the revenue associated to that job, because of the *premium* nature of the user's subscriptions. Accordingly, jobs have to be scheduled immediately or rejected rather than queued. A job interruption can occur both when the availability of green energy decreases and/or allocated jobs change their workload, causing the server's computing or energy capacity to become insufficient for all running jobs, and migration cannot be enforced.

Note that our approach to revenue and cost values does not consider topological factors such as the distance between nodes. In fact, those factors lead to negligible differences in the edge scenario (cf. [4]).

3.4. Game requirements model

Jobs originate from devices such as mobile phones, laptops, or smart TV. Thus, considering a large potential number of users, we consider that jobs arrive according to a Poisson process and have a duration extracted from a Weibull distribution, which realistically models the duration of online game sessions [41].

Every job will need powerful dedicated resources to work smoothly, in particular for energy [6] and computing power. The jobs, which we recall that refer to online game sessions, must meet QoS requirements in terms of delay and bandwidth, and they are characterized by their requirements in terms of energy, memory, and computation consumption.

Let us describe separately these aspects.

3.4.1. Delay

The overall response delay is the total time between an end-user submits his/her commands and the time the corresponding game frame is displayed to the user [42].

Response delay (D_r) is composed of network delay (D_n), processing delay (D_p), game logic (D_g) and playout delay (D_o), i.e., $D_r = D_n + D_p + D_g + D_o$ [42]. The network delay is basically the round-trip-time (RTT), which depends on where the server is placed; the processing delay is the delay to encode/decode and packetize commands and frames (which could take from 5 ms to 100 ms, depending on many factors [12, 42]); game logic delay denotes the time required by the game software to process a user's command and render the next game frame that contains responses to the command. This delay strongly depends on the game, with a range from 5 ms [12] to 50 ms with cases reaching even 130 ms [42]. Finally, the playout delay is the time required for the client to receive, decode, and display a frame, and it takes an average of 4 ms [12].

For simplicity, we work with average delays and focus on the network delay budget of each job, D_j, considering the other delay components (which are less correlated to network management and more dependant on the particular game) as constant.

Furthermore, we assume that queuing delays at switches are negligible, since our *premium* service could prioritize packets, avoiding unnecessary delays. The network delay budget is therefore spent over the links that connect the user to the game server, the resulting delay being the sum of average per-link delays d_z.

Since the time scale of our scheduling problem and the duration of the time slots is in the order of minutes, we also neglect the migration time because it is possible to obtain seamless game migration across several edge servers at millisecond timescale (cf. [12]).

3.4.2. Bandwidth

Focusing instead on the bandwidth requirements, we assume that each job requires a constant downlink bandwidth t_j, chosen at random from a uniform distribution with realistic bounds, while the uplink bandwidth is assumed to be negligible [42]. This assumption of constant t_j follows from the fact that it is possible to play games in streaming mode with several screen resolutions.

Furthermore, since we focus on the edge environment, we assume that the downlink bandwidth of far-edge nodes and M1 nodes is the bottleneck, rather than the per-link bandwidth, and consequently we ignore the latter.

3.4.3. Memory

Each job j requires a per-time-slot memory s_j at the node where it is running. We assume that this memory requirement is known and constant for the whole duration of the job, although it randomly varies for each job.

3.4.4. Computation requirements and energy consumption

Both energy consumption and computation requirements of a game session are strongly correlated. In particular, we consider that there exists a linear relation between both parameters, and that they may be different for each job. At a given time, we denote the energy consumption of job j as e_j, and its required computing power (in terms of processing cycles) as p_j.

We consider practical values for these requirements, extracted from some studies on gaming energy consumption [6] (in particular, from the resources in [43, 44]), such that we define both a minimum and a maximum value for both computation and energy consumption levels, as well as a mean value. Furthermore, we consider that the energy and computation for each job has a random value within the range of practical levels.

We consider two different scenarios regarding the jobs' requirements. First, we will consider that these values remain constant during the whole duration of the job. The second scenario is a practical generalization where the energy and computation requirements of a job vary over time. In such case, we assume that the required values evolve as a random walk process constrained within the maximum and minimum values.

In general, by considering fixed computing workloads and use of resources for each job, we make a tractable simplification which makes sense to evaluate a system in

Table 1: Notation used in this work

Notation	Meaning
C_j	Total Cost of job j
$C_j^{(b)}$	Energy cost (per slot) for job j
$C_j^{(d)}$	Deployment cost for job j
$C_j^{(m)}, C_j^{(p)}$	Migration and interruption costs for job j
$\mathcal{J}, J$	Set of jobs and its size
$\mathcal{N}, N$	Set of nodes (game servers), and its size
R_j	Revenue of job j
$\mathcal{T}$	Set of consecutive time slots
$\mathcal{Z}, Z$	Set of links and its size
T_n	Bandwidth of node n
t_j	Downlink throughput for job j
d_z	Delay incurred on link z
D_j	Maximum delay for job j
G_n, E_n	Green and total power at node n
e_j	Power required by job j
P_n	Computing power at node n
p_j	Computing power for job j
S_n	Memory capacity at node n
s_j	Memory required for job j
$w_{jz} = \{0,1\}$	(Variable) 1 if job j passes through link z
$x_{jn} = \{0,1\}$	(Variable) 1 if node n handles job j

which resources are always guaranteed to the user, hence they are allocated based on the peak demand of the online game session, which makes sense for a premium service like the one studied in this article. It has been shown in the literature that co-locating several games at the same server that has to share un-isolable resources leads to a general performance degradation of the QoS [16]. However in our work we do not consider such degradation since we do not have un-isolable resources.

With the above, we next formulate a utility optimization problem on how to allot jobs to nodes so as to maximize the overall utility by serving as many jobs in full and minimizing total costs. This means that the use of green energy has to be prioritized, migrations should be used only if they bring more revenue than cost, and job interruptions should be avoided.

4. Instantaneous Utility Optimization

First, we consider the instantaneous version of our problem, meaning that revenues and costs are allocated at each time slot, every job is allocated and executed in a single time slot, and there are neither migrations nor job interruptions.

4.1. Problem formulation

We consider the following variables: R_j is the revenue of accepted job j while C_j is its total cost. C_j includes deployment $C_j^{(d)}$ and brown energy costs $C_j^{(b)}$ associated to the computation required for the job.

Our decision variables, denoted by x_{jn} for all $j \in \mathcal{J}$ and all $n \in \mathcal{N}$, are binary variables that indicate whether job j is allocated at edge node n ($x_{jn} = 1$) or not ($x_{jn} = 0$). w_{jz} is another binary variable, whose value is 1 if job j passes through link z and 0 otherwise.

Table 1 summarizes the notation used in the article. The problem is therefore formulated as follows:

$$\max \sum_{n \in \mathcal{N}} \sum_{j \in \mathcal{J}} (R_j - C_j)\, x_{jn}; \tag{1a}$$

$$\text{s.t.:} \sum_{n \in \mathcal{N}} x_{jn} \leq 1, \qquad \forall j \in \mathcal{J}; \tag{1b}$$

$$\sum_{j \in \mathcal{J}} t_j x_{jn} \leq T_n, \qquad \forall n \in \mathcal{N}; \tag{1c}$$

$$\sum_{j \in \mathcal{J}} p_j x_{jn} \leq P_n, \qquad \forall n \in \mathcal{N}; \tag{1d}$$

$$\sum_{j \in \mathcal{J}} e_j x_{jn} \leq E_n, \qquad \forall n \in \mathcal{N}; \tag{1e}$$

$$\sum_{j \in \mathcal{J}} s_j x_{jn} \leq S_n, \qquad \forall n \in \mathcal{N}; \tag{1f}$$

$$\sum_{z \in \mathcal{Z}} d_z w_{jz} \leq D_j, \qquad \forall j \in \mathcal{J}; \tag{1g}$$

where:

- The objective function (1a) expresses the net utility;
- Constraint (1b) states that job j can only be allocated to a single node n;
- Constraints (1c) to (1f) ensure that a job's placement does not violate the server's capacity in terms of: downlink bandwidth (T_n), processing power (P_n), available energy (instantaneous power E_n), and memory (S_n);
- Constraint (1g) ensures that the average delay is guaranteed for each job;
- All weights t_j, p_j, e_j, s_j, and d_z, capacities T_n, P_n, E_n, S_n, and delay budgets D_j take positive values.

The above described problem is non-trivial to solve if no server can accommodate all jobs. In that case, the problem is NP-Hard, as shown next.

Theorem 1. *Constraints* (1b) *and* (1c) *alone make the problem NP-hard (in the strong sense).*

Proof. We reduce the Multiple Knapsack Problem (MKP) to our problem formalization. According to [45], the MKP could be written as follows: considering a set of K knapsacks with capacity W_k each, $k \in \{1, \ldots, K\}$, and a set of I items to store ($K \leq I$) where each item i has positive reward r_i and positive weight w_i, $i \in \{1, \ldots, I\}$. The objective expression is $\sum_{k=1}^{K} \sum_{i=1}^{I} r_i x_{ik}$, which has to be maximized under the constraints that $\sum_{i=1}^{I} w_i x_{ik} \leq W_k, \forall k$, and $\sum_{k=1}^{K} x_{ik} \leq 1, \forall i$, with x_{ik} being a binary variable indicating whether item i is allocated to knapsack k.

We consider the special case where $C_j = 0$ and p_j, e_j, s_j, and d_z are all equal to 1, whereas P_n, E_n, and S_n are equal to J and $D_j = Z$. In this special case, constraints (1d)-(1g) are all redundant and always satisfied.

With this special configuration, our problem is a MKP with $K = N$ knapsacks of capacity T_n and $I = J$ items with weights t_j and rewards R_j. This means that the MKP is a particular case of our problem. Therefore, we could argue that our problem is complex as much as the MKP, which is NP-hard. Since this reduction can be built in polynomial time, it follows that our problem is NP-hard. However, we highlight that due to this reduction to MKP, our problem is NP-hard in the strong sense, meaning that no polynomial-time approximation scheme is known [45] unless $P = NP$. □

4.2. Sub-modularity

We now show that the problem in Section 4.1 is sub-modular, which leads to useful performance guarantees. First, let us re-formulate the problem as a set-optimization problem. Let $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{N}$ denote the set of selected single-service placements, where $(j, n) \in \mathcal{S}$ means that job j is placed at node n. Let $\Theta(\mathcal{S})$ denote the objective value of (1a), so that (1) becomes

$$\max \Theta(\mathcal{S}) \tag{2a}$$
$$\text{s.t.: } \mathcal{S} \subseteq \mathcal{J} \times \mathcal{N} \tag{2b}$$
$$(1b) \text{ to } (1g). \tag{2c}$$

Theorem 2. *The optimal value of Θ is a monotone increasing and sub-modular set function.*

Proof. Consider that a real-valued set function f is monotone increasing if $\forall\ \mathcal{S}_1 \subseteq \mathcal{S}_2 \subseteq \mathcal{S}$, $f(\mathcal{S}_1) \leq f(\mathcal{S}_2)$. Moreover, the function $f(\cdot)$ is sub-modular if $\forall\ \mathcal{S}_1 \subseteq \mathcal{S}_2 \subseteq \mathcal{S}$ and $u \in \mathcal{S} \setminus \mathcal{S}_2$, it holds that $f(\{u\} \cup \mathcal{S}_1) - f(\mathcal{S}_1) \geq f(\{u\} \cup \mathcal{S}_2) - f(\mathcal{S}_2)$.

The monotonicity of the solution of our problem is clear because expanding $\mathcal{S}$ (i.e., putting more jobs and/or nodes) enlarges the solution space of (2a) and therefore increases its optimal value. The solution is also sub-modular since, for a given amount of green energy, any increase in the number of allocated jobs will increase the amount of required polluting energy at the nodes, and therefore the overall utility obtained by including more jobs will be progressively reduced. For this class of problems, it is known that we can construct a greedy algorithm that iteratively selects the element that maximizes (subject to the constraints) the objective function, such that this algorithm achieves a performance guarantee of $1 - 1/e$ [46]. □

We present in Algorithm 1 the legacy `GREEDY` algorithm that solves problem (2) in polynomial time with performance guarantees using its submodularity property. Since the structure of the algorithm is well known and derives from [46], we omit a detailed explanation about it and refer to [46] for further information.

Algorithm 1 `GREEDY Algorithm`

```
1: Input: Network topology, $\mathcal{N}$, jobs $\mathcal{J}$ (with parameters $t_j, p_j, s_j, e_j, D_j\ \forall j \in \mathcal{J}), T_n, P_n, S_n, G_n, E_n\ \forall n \in \mathcal{N}, d_z\ \forall z \in \mathcal{Z}$
2: Output: Job-to-node placement map $\mathcal{S}$
3: Initialize: $\mathcal{S} = \emptyset$;   $\mathcal{J}^\emptyset = J$;   $\mathcal{S}^\emptyset = \mathcal{J} \times \mathcal{N}$
4: while $\exists (j, n) \in \mathcal{S}^\emptyset$ s.t. $\mathcal{S} \cup (j, n)$ satisfies (2c) do
5:     $(j^\star, n^\star) \leftarrow \arg\max_{(j,n) \in \mathcal{S}^\emptyset} \Theta(\mathcal{S} \cup (j, n))$
6:     $\mathcal{S} \leftarrow \mathcal{S} \cup (j^\star, n^\star)$
7:     $\mathcal{J}^\emptyset \leftarrow \mathcal{J}^\emptyset \setminus j^\star$
8:     $\mathcal{S}^\emptyset = \mathcal{J}^\emptyset \times \mathcal{N}$
9: end while
```

5. Online Problem with Migrations and Penalties

The problem in Section 4 can be extended to the case where jobs last more than the duration of a time slot and arrive asynchronously. This situation is important because it represents the practical problem to be solved online in a real system. For this case, the objective function of the optimization problem becomes

$$\max \sum_{\tau \in \mathcal{T}} \sum_{n \in \mathcal{N}} \sum_{j \in \mathcal{J}} \left(R_j(\tau) - C_j(\tau)\right) x_{jn}(\tau) \tag{3}$$

where $\mathcal{T}$ is the time interval (a set of consecutive slots) over which we optimize the utility of the system, and where we recall that the total cost $C_j(\tau)$ is obtained as $C_j(\tau) = C_j^{(d)}(\tau) + C_j^{(b)}(\tau) + C_j^{(m)}(\tau) + C_j^{(p)}(\tau)$. In (3), we consider a one-time revenue rather than a per-time-slot revenue, such that $R_j(\tau)$ is a non-zero value only at the time slot of the arrival of request of acceptance for job j, and it does not depend on the job duration. Similarly, the deployment cost ($C_j^{(d)}$) is only non-zero at acceptance time, whereas the penalties for migration ($C_j^{(m)}$) and for job interruption ($C_j^{(p)}$) are only applied in the time slots in which the corresponding events occur. The only term that appears in every time slot (due to its possible fluctuation) is the variable cost incurred by consuming non-renewable energy ($C_j^{(b)}$). In this optimization problem, the objective function in (3) must satisfy the same constraints as the problem in (1a), i.e., (1b)–(1g), except for the fact that these constraints have to hold at any time slot $\tau \in \mathcal{T}$.

In this new formulation, jobs can arrive in different time slots, and decisions must be made online at the slot boundaries. It is worth noting that, if migration and job interruption costs are neglected, the problem is equivalent to the one shown in Section 4, because it is enough to maximize the objective function slot by slot. Therefore the sub-modularity property will hold also for the online job allocation problem under such simplifying assumptions. Instead, if we consider those penalties, sub-modularity is not guaranteed. However, with realistically small migration costs and rare job interruptions, the problem can be considered, *in practice*, still sub-modular or as a small perturbation of a sub-modular case. From that, it is intuitive

Algorithm 2 GREENING – Proposed heuristic algorithm

```
1:  Input: $\mathcal{N}$, active ($\mathcal{J}^{(\tau)}$) and new ($\mathcal{J}^{(+)}$) jobs with
    current parameters $t_j, s_j, p_j(\tau), e_j(\tau), D_j\ \forall j \in \{\mathcal{J}^{(\tau)} \cup \mathcal{J}^{(+)}\}$,
    network parameters $T_n, P_n, S_n, E_n^{(\tau)}\ \forall n \in \mathcal{N}$,
    $d_z\ \forall z \in \mathcal{Z}$, and previous allocation $\mathcal{S}^{(\tau-1)}$.
2:  Output: $\mathcal{S}^{(\tau)}, \Theta^{(\tau)}$
3:  Initialize: $\mathcal{S}^{(\tau)} \leftarrow \mathcal{S}^{(\tau-1)}$; $\Theta^{(\tau)} \leftarrow 0$;
4:  if Jobs or Nodes energy levels change then
5:    for $n \in \mathcal{N}$ do
6:      $E_{n,\text{eff}} \leftarrow \sum_{j:(j,n)\in\mathcal{S}^{(\tau)}} e_j(\tau)$
7:      $P_{n,\text{eff}} \leftarrow \sum_{j:(j,n)\in\mathcal{S}^{(\tau)}} p_j(\tau)$
8:      while $E_n^{(\tau)} < E_{n,\text{eff}}$ or $P_n^{(\tau)} < P_{n,\text{eff}}$ do
9:        $\mathcal{J}_n \leftarrow \{\, j \mid (j,n) \in \mathcal{S}^{(\tau)}\}$
10:       $\mathcal{N}_g^{(\tau)} \leftarrow$ sort($\mathcal{N}, G_n^{(\tau)} - E_{n,\text{eff}}$)
11:       $j^{(m)}, n^{(m)} \leftarrow$ Migration-GREENING($\mathcal{J}_n$, $n$)
12:       if $n^{(m)} == -1$ {interruption} then
13:         $j^{(m)} \leftarrow \arg\min_{j:(j,n)\in\mathcal{S}^{(\tau)}}\{R_j\}$
14:         $\mathcal{J}^{(\tau)} \leftarrow \mathcal{J}^{(\tau)} \setminus \{j^{(m)}\}$
15:         $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} - C_{j^{(m)}}^{(p)}$
16:         $\mathcal{S}^{(\tau)} \leftarrow \mathcal{S}^{(\tau)} \setminus \{(j^{(m)}, n)\}$
17:       else if $n^{(m)} \neq n$ {migration} then
18:         $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} - C_{j^{(m)}}^{(m)}$
19:         $\mathcal{S}^{(\tau)} \leftarrow \{\mathcal{S}^{(\tau)} \setminus (j^{(m)}, n)\} \cup \{(j^{(m)}, n^{(m)})\}$
20:       end if
21:       $E_{n,\text{eff}} \leftarrow \sum_{j:(j,n)\in\mathcal{S}^{(\tau)}} e_j(\tau)$
22:       $P_{n,\text{eff}} \leftarrow \sum_{j:(j,n)\in\mathcal{S}^{(\tau)}} p_j(\tau)$
23:     end while
24:   end for
25: end if
26: Execute: Acceptance-GREENING (Algorithm 2-a)
27: $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} - \sum_{(j,n)\in\mathcal{S}^{(\tau)}} C_j^{(b)}$ {Substract energy cost}
```

to consider that we can extend the greedy heuristic approach also to the online version of the problem, as shown in the following.

5.1. Proposed online heuristic

For sub-modular problems, it is known that the simple strategy of maximizing the instantaneous utility at each time that a new job arrives achieves high performance. This approach precludes the possibility of rejecting a job just because it might prevent the acceptance of future jobs. However, this strategy does not limit us to only use the GREEDY algorithm in Algorithm 1. Instead, we propose a heuristic algorithm that, although it retains the *greedy* spirit of Algorithm 1, is able to conform to the current state of the servers and the available green energy. Our proposal, named GREENING, operates in a per-time-slot basis, and it takes its decision only based on the current state of the network. For the sake of readability, we have split the description of the algorithm in two sequential stages and one auxiliary function: The general description of our proposed algorithm is shown in Algorithm 2, which includes the entire procedure; however, the latest part of the algorithm, which handles the acceptance of newly arrived jobs, is disclosed in Algorithm 2-a due to space limitations. Finally, a migration function called by both Algorithm 2 and Algorithm 2-a will be presented in Algorithm 2-m. The migration function is called in two circumstances: when an arriving job is not allocated with a direct placement and when there is a change of green energy levels due to changes in energy generation or in gaming workloads. Next, we detail the algorithm and each one of its parts.

Algorithm 2-a Acceptance-GREENING – Proposed heuristic algorithm (Part II: Acceptance of new jobs)

```
1:  Continue from line 25 in Algorithm 2
2:  $E_{n,\text{eff}} \leftarrow \sum_{j:(j,n)\in\mathcal{S}^{(\tau)}} e_j(\tau)\ \forall n \in \mathcal{N}$
3:  $\mathcal{N}_g^{(\tau)} \leftarrow$ sort($\mathcal{N}, G_n^{(\tau)} - E_{n,\text{eff}}$)
4:  Define: $\bar{\mathcal{S}}_{(j_1\to j_2),n}^{(\tau)}$ as $\{(j_2, n) \cup \{\mathcal{S}^{(\tau)} \setminus (j_1, n)\}\}$
5:  for $j_{\text{arr}} \in \mathcal{J}^{(+)}$ do
6:    for $n \in \mathcal{N}_g^{(\tau)}$ do
7:      if $(j_{\text{arr}}, n)$ satisfies (1c)–(1g) then
8:        $\mathcal{J}^{(\tau)} \leftarrow \mathcal{J}^{(\tau)} \cup \{j_{\text{arr}}\}$
9:        $\mathcal{S}^{(\tau)} \leftarrow \mathcal{S}^{(\tau)} \cup \{(j_{\text{arr}}, n)\}$
10:       $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} + R_{j_{\text{arr}}} - C_{j_{\text{arr}}}^{(d)}$
11:       break loop over $n$
12:     end if
13:   end for
14:   if $j_{\text{arr}} \notin \mathcal{J}^{(\tau)}$ {New job not placed} then
15:     for $n \in \mathcal{N}_g^{(\tau)}$ do
16:       $\mathcal{J}_{\text{arr},n} \leftarrow \{\, j \mid (j,n) \in \mathcal{S}^{(\tau)}$ and
                 $\bar{\mathcal{S}}_{(j\to j_{\text{arr}}),n}^{(\tau)}$ satisfies (1c)–(1g)}
17:       $j^{(m)}$, $n^{(m)} \leftarrow$ Migration-GREENING($\mathcal{J}_{\text{arr},n}$, $n$)
18:       if $n^{(m)} \neq -1$ then
19:         $\mathcal{J}^{(\tau)} \leftarrow \mathcal{J}^{(\tau)} \cup \{j_{\text{arr}}\}$
20:         $\mathcal{S}^{(\tau)} \leftarrow \mathcal{S}^{(\tau)} \cup \{(j_{\text{arr}}, n)\}$
21:         $\mathcal{S}^{(\tau)} \leftarrow \{\mathcal{S}^{(\tau)} \setminus (j^{(m)}, n)\}$
                 $\cup\ \{(j^{(m)}, n^{(m)}),\ (j_{\text{arr}}, n)\}$
22:         $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} + R_{j_{\text{arr}}} - C_{j_{\text{arr}}}^{(d)} - C_{j^{(m)}}^{(m)}$
23:         break loop over $n$
24:       end if
25:     end for
26:     if $j_{\text{arr}} \notin \mathcal{J}^{(\tau)}$ then
27:       Reject job $j_{\text{arr}}$
28:     end if
29:   end if
30: end for
31: Continue in Algorithm 2
```

The algorithm is triggered at the beginning of each time slot. It has two main stages. One is dedicated to react and re-schedule active jobs in the possible event that either the energy availability at the nodes or the energy requirements

Algorithm 2-m `Migration-GREENING`

```
1: Input:      $\mathcal{J}^{(m)}$ (Set of candidate jobs to migrate)
               $n$ (node that needs to migrate jobs)
2: Inherit:    State and variables of Algorithm 2
3: Output:     $j^{(m)}$ (job to migrate)
               $n^{(m)}$ (node where $j^{(m)}$ migrates)
4: Initialize: $n^{(m)} \leftarrow n$
5: for $j_c \in \mathcal{J}^{(m)}$ do
6:   for $n' \in \mathcal{N}_g^{(\tau)} \setminus \{n\}$ do
7:     if allocating $j_c$ to $n'$ satisfies (1c)–(1g) then
8:       $n^{(m)} \leftarrow n'$
9:       $j^{(m)} \leftarrow j_c$
10:      break double loop over $\mathcal{J}^{(m)}$ and $\mathcal{N}_g^{(\tau)}$
11:    end if
12:  end for
13: end for
14: if $n^{(m)} == n$ {No node to migrate} then
15:   $n^{(m)} \leftarrow -1$
16: end if
```

for the jobs change with respect to the previous time slot. The second part focuses on the admission control and optimizes the resource allocation in order to accept new jobs if it is possible.

Re-allocating ongoing jobs. (Algorithm 2). First, the `GREENING` algorithm checks whether the amount of available renewable energy has changed at any node. In the case in which the jobs energy and computation requirements can dynamically change, the algorithm also monitors if these values have evolved. If any of these events happen, some nodes might no longer have enough power to serve all their allocated jobs, and therefore some jobs must be migrated or interrupted.

The algorithm proceeds node by node and, for each node with not enough resources (in terms of either computation or energy resources), it examines if some job can be migrated to other—less loaded—nodes in order to avoid job interruptions. This search of both jobs to migrate and feasible destination nodes is carried out by the migration function `Migration-Greening` presented in Algorithm 2-m. This function takes as input a node n and a set of candidate jobs $\mathcal{J}^{(m)}$ to be migrated from node n, and it outputs which one of the candidate jobs has to be migrated ($j^{(m)}$) and toward which node is the migration conducted ($n^{(m)}$).

Importantly, before starting the search for possible migrations the nodes are sorted by the amount of *available* green energy, in descending order.[2] For that, let us introduce some useful notations. We define $E_{n,\text{eff}}$ as the total energy required by all the jobs currently running in node n. $P_{n,\text{eff}}$ is similarly defined for the computation resources required at node n. From the definition of $E_{n,\text{eff}}$, it follows that the amount of green energy currently available at node n is obtained by subtracting $E_{n,\text{eff}}$ from the total amount of green energy in the node ($G_n^{(\tau)}$), where a negative value of $G_n^{(\tau)} - E_{n,\text{eff}}$ indicates the amount of polluting energy consumed at node n. Let us further denote the set of nodes ordered based on $G_n^{(\tau)} - E_{n,\text{eff}}$ as $\mathcal{N}_g^{(\tau)}$, and the node index in the i-th position of $\mathcal{N}_g^{(\tau)}$ as η_i. From this notation, it follows that $\mathcal{N}_g^{(\tau)}$ is ordered such that $(G_{\eta_i}^{(\tau)} - E_{\eta_i,\text{eff}}) \geq (G_{\eta_{i+1}}^{(\tau)} - E_{\eta_{i+1},\text{eff}})$ for any $i < N$. This ordering is motivated by the fact that nodes that have more available green energy incur less costs.

If no other node can accommodate any of the jobs in node n, Algorithm 2-m returns that the destination node is -1. In this latter case, when no job can be migrated, the job with the smallest revenue (since the interruption cost is comparable to the revenue) in the node is interrupted. This process is repeated until all energy and computation constraints are satisfied.

Migrating function. (Algorithm 2-m). The previously mentioned migration function operates a simple search on the set of potential migration destination nodes and checks the feasibility of migration based on the problem's constraints. For each candidate job j_c in the input set $\mathcal{J}^{(m)}$, we evaluate if j_c can be migrated to other node n'.

In order to check the feasibility of the migration, the search starts from the node with more *available* green energy and the list of nodes follows by the amount of *available* green energy $\mathcal{N}_g^{(\tau)}$. In this manner, we give priority to the nodes that reduce the cost of energy consumption. The search stops as soon as a destination node is found. Once we find a node $n' \in \mathcal{N}_g^{(\tau)} \setminus n$ that can allocate a job $j_c \in \mathcal{J}^{(m)}$, we set job j_c as the migrating job ($j^{(m)}$) and node n' as the destination node ($n^{(m)}$), which are the outputs of the function. If there is no feasible pair ($j^{(m)}$, $n^{(m)}$), the function returns $n^{(m)} = -1$.

Acceptance of new jobs. After handling the continuity of the jobs that are already in the system, `GREENING` focuses on the admission of newly arrived jobs. For that, it tries to allocate them one by one, in a sequential order. For each one of the arrived jobs, the algorithm verifies if the job fits in any of the servers. This verification follows the same *available green energy* order $\mathcal{N}_g^{(\tau)}$ as described in the previous stage, such that the nodes with the highest available green power have priority in the job allocation.

The algorithm tries a direct placement on the node at the top of the list, and moves to the next node only if the allocation is not possible according to any of the constraints. This is aligned with the greedy heuristic of the instantaneous problem, although considering just energy levels rather than overall allocation utilities. Yet, the probability of making the same decision as the greedy algorithm

[2]Sorting nodes according to the *total* available level of green energy is also possible, as shown in our preliminary work [1], although using the residual energy is more robust to dynamic workloads.

is high, because nodes with higher unused green energy are likely to be the ones offering the highest utility.

However, if no node in the list can take a newly arrived job, GREENING tries to migrate some of the already allocated jobs so that the new job can fit in the system. This section of the algorithm substantially differs from a standard greedy heuristic. In order to do this, the algorithm invokes again the migration function Migration-Greening from Algorithm 2-m on the already allocated jobs. In this case, however, there exists a difference with respect to the other call to the function. Before, the set of candidate jobs $\mathcal{J}^{(m)}$ was the whole set of jobs allocated to node n, i.e., $\mathcal{J}^{(m)} = \{j \,|\, (j,n) \in \mathcal{S}^{(\tau)}\}$. Now, since we need to have enough space to allocate the new job, we restrict the set of candidate jobs to be composed only of the jobs enabling the new admission. This set is given by $\mathcal{J}^{(m)} = \{j \mid (j,n) \in \mathcal{S}^{(\tau)}\} \cap \{j \mid \bar{\mathcal{S}}^{(\tau)}_{(j\to j_{\text{arr}}),n}$ satisfies (1c)–(1g)$\}$, where we have defined $\bar{\mathcal{S}}^{(\tau)}_{(j_1\to j_2),n}$ as the resulting allocation set obtained from substituting the already allocated job j_1 by the new job j_2, i.e., $\bar{\mathcal{S}}^{(\tau)}_{(j_1\to j_2),n} = \big\{(j_2,n) \cup \{\mathcal{S}^{(\tau)} \backslash (j_1,n)\}\big\}$. As before, the nodes are ordered by the amount of available green energy. If the migration function does not find any migration combination that makes enough room for the new job, the job is rejected and its revenue is lost. Otherwise, the job is allocated, bringing a revenue of R_j and a cost of deployment of $C_j^{(d)}$, and the migration is committed with an incurred cost $C_j^{(m)}$.

Eventually, the algorithm discounts from the objective function the cost due to the amount of polluting (non-renewable) energy consumed during the time slot.

Note that the described migration function is greedy and so Algorithm 2 is still a greedy algorithm, in the sense that it makes instantaneous decisions without considering what could happen in the future. However, allowing migrations can only improve the utility obtained with a scheme without migrations, be it Algorithm 1 or Algorithm 2 simplified by skipping the call to the migration function. Therefore, we can expect that Algorithm 2 will offer better performance guarantees than the value $1-1/e$ of Algorithm 1.

To conclude, the complexity of our GREENING heuristic described in Algorithm 2 is $O(N^2J^2)$, which would reduce to $O(NJ^2)$ in case of direct placement of the arriving jobs, without migrations.

5.2. ETSI MEC and network slicing compatibility

In this subsection we comment on how green edge gaming is compliant to both ETSI MEC and network slicing concepts.

In the case of ETSI MEC, the MEC Orchestrator (MEO), which has an overview of the complete MEC system and therefore could be deployed in more centralized nodes, could consider the objective function (1a) to place games in its system. Indeed, one of the MEO's roles consists in selecting appropriate MEC host(s) for application instantiation based on constraints, such as latency, available resources, and available services [3]. To this aim, the MEO talks directly to the Virtual Infrastructure Manager (VIM), whose role is to physically deploy resources. MEC hosts provide compute, storage and network resources for the MEC applications and they could be deployed in edge and M1 nodes, where games are actually installed. Finally, games could be deployed as MEC applications, leveraging on several *on-board* MEC services, such as Radio Network Information, location and traffic management to sustain the appropriate QoE level. Edge and M1 nodes could be connected through several reference points: with the MEO through a *Mm3* link and they could communicate between each other through a *Mp3* link [3]. Indeed, Figure 2 shows also a high level example of the edge gaming scenario implemented through ETSI MEC.

Green edge gaming could also leverage network slicing to guarantee resources to servers. A service provider could reserve a slice of resource in order to satisfy end-users in terms of bandwidth computing power.

6. Numerical Evaluation

In this section we evaluate numerically the proposed algorithm on a set of green edge gaming scenarios, and we provide a performance comparison with alternative online gaming solutions. To perform our experiments we built a simulator with Matlab 2021a, in which we implemented our solution as well as several baselines and state-of-the-art alternatives.

We study the performance of the considered solutions in a set of different configurations. We are interested in analyzing how the different parameters and possible topologies of the edge computing system impact the results. For that, we run a set of experiments, where in each of the experiments we vary one aspect of the network (e.g., the arrival rate of jobs, the energy dynamics, the relation between number of far-edge nodes and M1 nodes, etc.). Among the compared algorithms, we consider cases where migrations are not considered, or where the type energy (renewable or not) is not taken into account, so as to better understand the impact of each of the features.

We start by describing the general parameters of the scenarios considered, and later we will detail each variation and its implications.

6.1. Simulation scenario and setup

We study the problem in a metropolitan area where users leverage online game servers in far-edge and M1 nodes, with the QoS requirements described before in terms of computing power, latency, memory, and bandwidth. For each of the settings considered in the following, we evaluate different sizes of the edge network, i.e., a set of values of the number of nodes N, always within the range compatible with the number of edge and M1 nodes

Table 2: Simulation parameters

	Edge server	**Job**
Bandwidth	350 Mbps	$\mathcal{U}(10, 30)$ Mbps
Computing	3 × 3.5 GHz (Far-edge) 5 × 3.5 GHz (M1)	Random walk within 315 to 385 Mflops
Memory	3 × 64 GB (Far-edge) 5 × 64 GB (M1)	$\mathcal{U}(750, 850)$ MB
Power	1.5 kW (Far-edge) 2 kW (M1)	Random walk within 70 to 130 W
Delay	$\mathcal{U}(2, 15)$ ms	$\mathcal{U}(50, 150)$ ms
Revenue	-	$\mathcal{U}(0.03, 0.0367)$ $
Duration	-	$\mathcal{W}\mathrm{eib}(2504.8, 2.9637)$
Deployment	0.01 $ (Far-edge) 0.015 $ (M1)	-
Migration	-	0.0003 $
Interruption	-	100% of the revenue
Energy	-	0.35 $/kWh

that will be initially deployed in a metropolitan framework, in line with previous studies [35]. We will vary the total number of nodes between 4 and up to 48.

We simulate a green edge gaming environment during a whole day, and repeat the experiment several times until we obtain small confidence intervals. We solve Problem (3) with multiple approaches, on a slot-by-slot basis. We consider that each time slot lasts one minute, which is much shorter than a typical online game session ($\sim$40 minutes [41]) and much longer than any job migration mechanism (lasting from tens of milliseconds [12] up to seconds) or game session launching (which takes less than a second [10]).

6.1.1. Network topology and server specifications

The network topology is hierarchical, as displayed in Figure 2, and the connectivity in between servers is assumed to be a full mesh. Throughout the experiments, we will vary the portion of the nodes that belong to the M1 type.

Server capabilities are based on a NVidia blade server [47] for edge computing. In particular, we consider that far-edge servers dedicate 3 blades to our use case, whereas M1 nodes dedicate 4 blades. Each of these blades is endowed with a CPU of 3.5 GHz for computing power, 64 GB of RAM memory and requires 450 watts (W) of energy. From this, we consider that the far-edge nodes require 1.5 kW in order to work at full capacity, while M1 nodes require 2 kW.

Besides, we assume that the bandwidth and incurred delays for the edge nodes are constant and inline with 5G values; specifically, we consider that each node has a downlink bandwidth of 350 Mb/s and incurs a latency of the order of 5-10 ms.

6.1.2. Job statistics

We assume that the time of arrival of jobs follows a Poisson process, such that the number of arrivals in each time slot is given by a Poisson random variable with rate λ. In general, we will scale the arrival rate proportionally to the number of nodes, such that λ can be generically written as $\lambda = \alpha N$, where α is a constant.

The duration of a job is extracted from a Weibull distribution, which is known to precisely characterize the distribution of duration of online game sessions [41]. In particular, we consider a Weibull distribution with parameter $k = 2.9637$ and $\mu = 2504.8$, which yields an average duration of about 40 minutes for a typical session, and which also yields that the probability of having durations above two hours is negligible.

The computation requirement of each job ranges from 315 to 385 Mcycles/s, which amounts to 10% to 15% of a standard server CPU core. The energy requirements of the game sessions are strongly correlated with the computation requirements, and they are randomly generated within the range 70–130 W, with a mean of 100 W. These values are obtained from studies on online gaming requirements (cf. [6, 43, 44]). We provide more information about energy dynamics in Section 6.1.3.

In terms of bandwidth requirements, we consider that it can vary uniformly from 10 to 30 Mb/s, which matches the requirements for video resolutions that range from 720p to 4K [48]. Other game session requirements (memory, delay, CPU) are inline with previous works [6, 9]. For instance, the maximum delay allowed for each game session is a random variable uniformly distributed between 50 and 150 ms, and RAM requirements are also uniformly distributed in the interval from 750 to 850 MB.

With the above numbers, a system working *at full capacity* at all the nodes can allocate on average up to 14 jobs in each far-edge node and a maximum of about 20 jobs at each M1 node. Note that, for the already mentioned dependency on renewable energy, this peak of capacity is likely never reached in the far-edge nodes.

We would like to note that, for the assumed specifications of both nodes and jobs, the system is saturated (i.e., the servers are using all the available resources for active jobs) for $\alpha > 0.6$, while $\alpha < 0.1$ implies generically that all nodes have always room for more jobs and every job is accepted and served.

6.1.3. Energy fluctuation dynamics

Let us explain how the energy availability at the edge nodes and the jobs energy requirements evolve over time.

We focus first on the availability of green energy at the nodes. We consider that the green energy available at each node (and locally generated) changes every 15 minutes. In contrast with our previous work [1], we consider that the energy available presents space-time correlation. We generate random samples of green energy availability from the datasets provided by Elia for wind [39] and solar [40] energy generation. For each node we select an energy profile from a different day of the forecasting dataset (see Figure 3 for a sequential visualization of the profile for seven different days).

Every 15 minutes, each node changes its available green energy following the given statistics (in terms of mean, minimum, and maximum expected value) from the random day profile. The specific value is obtained as a random sample of a PERT distribution [49] characterized by the mean, minimum, and maximum values provided by the energy profile. The PERT distribution, which is highly related to the well-known Beta distribution, is usually considered for modelling and estimating the effect of uncertainty. The total green energy available is then the sum of both wind- and solar-generated resources.

The far-edge nodes are assumed to rely only on the local green energy available (apart from a minimum constant energy that ensures the functioning of the server). If no green energy is available at a certain time, jobs in the far-edge node must be migrated to another node or interrupted. This implies that the capacity of the far-edge nodes varies over time. Actually, in our experiments, the average capacity of the far-edge nodes ranges between 25% and 80% of the nominal capacity (i.e., between 375 W and 1.2 kW out of a nominal peak power of 1.5 kW). On the other hand, the M1 nodes have always access to the same amount of energy (2 kW), irrespective of the amount of green energy, which in our experiments is covered by green sources for up to 1.2 kW, i.e., up to 60% of the power available at an M1 node can be green.

We will consider two cases for the M1 layer: The default case (**green M1**), in which the green energy availability at M1 nodes follows the same statistics as the one for far-edge nodes. The only difference in this case between nodes is that M1 nodes use polluting energy to obtain the remaining amount of energy until 2 kW of power. Hence, they can secure a certain level of reliability in the system at the expense of a higher cost due to the cost of energy. The second case (**brown M1**) is the case in which M1 nodes make only use of the general electricity grid, i.e., they only consume non-renewable energy, while they keep enjoying the constant 2 kW. With these two cases we try to understand the impact of heterogeneity in the access to green energy resources.

Next, we describe the dynamics of the energy/power requirements for the game sessions. We also consider two different cases. The first one is the realistic scenario in which the energy requirements of a particular game session vary at each time slot (**dynamic workload**). This continuous variation is inherent to the nature of gaming. We consider that the power required by a job for the next time slot follows a random walk with standard deviation 5 W. We limit the value of this variable to the maximum and minimum values provided before (70 and 130 W, respectively), since in practice a game has a limit on both maximum and minimum requirements. The second case is the simplified case in which the energy required by a job is randomly picked at the start of the game session but then it remains constant with this initial value throughout the duration of the session (**static workload**). This particular case is an abstraction to the approach in which the jobs are allocated the maximum amount of resources that they will ever need, such that there is no need of monitoring the current demand, but at the same time implies overprovisioning and hence a waste of resources due to the inevitable variation of the real requirements, as the worst-case (maximum) value will not be frequently reached.

6.1.4. Monetary gains and costs

Each job provides a monetary gain R_j that ranges between 0.03\$ and 0.0367\$. The revenue of the job is assumed to be uniformly distributed in this range. The motivation to pay a higher fee is that higher-revenue jobs will have less chances of being interrupted. On the other hand, the migration cost of a job is fixed to 0.0003\$, which is also the deployment cost at the far-edge nodes; the deployment cost for jobs allocated to M1 nodes is considered to be 50% more expensive than that of far-edge nodes, due to the longer distance. In case of a job interruption, the penalty is a monetary cost equal to the revenue previously paid by the user (R_j). This value follows from the fact that this use case is a premium service scenario, for which the user *expects* to receive a great QoS, which would not be possible in case of interruption of an active game session.

With respect to the cost of energy consumption, we assume a price of 0.35\$ per kWh, which is a realistic value in line with current prices in Europe (by first half of 2022). We consider that the energy generated from renewable sources does not incur any monetary cost, since it is locally generated at the edge node and cannot be stored for long term. Hence, only the energy from non-renewable (polluting) sources will incur the cost of 0.35 \$/kWh. We assume that the energy obtained from the general grid is coming entirely from non-renewable sources.[3]

6.1.5. Metrics and algorithms

In our experiments, the main performance metric is the system utility, which is computed on a per time slot basis. The average system utility is proportional to the objective function of our online optimization problem (3), so that it represents the performance of the tested algorithms.

Besides the average utility, we also consider other metrics to shed light on the behavior of the system. For that, we also evaluate the user's QoE by means of comparing the normalized *time played*, which we define as the ratio between the sum of the service offered to all active *accepted* jobs over the total aggregate nominal duration of all (*accepted and rejected*) jobs. This metric provides us with information about the percentage of users that are satisfied with the system.

In order to provide a broader perspective of the functioning of the algorithms, we also provide the average amount

[3]The portion of energy generated from renewable sources varies strongly for different countries and for different periods of the year or of the day. Our simulations are directly applicable under the assumption of mixed generation just modifying the price of the energy based on the percentage of green energy p_g as $0.35(1 - p_g)$ \$/kWh.

of jobs in the system, as well as the amount of rejected, interrupted, and migrated jobs. We omit the study of other typical QoS metrics like jitter or packet losses because in the edge gaming scenario here considered their values are typically small and thus they are less relevant.

The above metrics are computed in terms of average and 95% confidence intervals for 8 different algorithms:

- `Solver` is an algorithm that solves integer linear program (3), evaluated at each time slot over a time horizon of one time slot ($|\mathcal{T}| = 1$). It uses the Matlab *intlinprog* function with a timeout of 40 seconds for each simulated time slot, in order to avoid long lasting experiments. A single experiment showed in what follows can require up to one week to complete notwithstanding the imposed timeout. Due to such huge complexity, we only provide the solution with this algorithm in a subset of the experiments.

- `GREENING` is our proposed heuristic defined in Algorithm 2.

- `PFPJ-1` is derived from [33], which presents a resource-aware allocation and migration algorithm designed for IoT Cloud applications. It clusters servers into highly and lightly loaded subsets, and enforces migrations from highly to lightly loaded servers to enforce load balancing. We have added a power constraint in the original algorithm for a fairer comparison with our scheme.

- `PFPJ-2` is the original placement algorithm defined in [33].

- `GREENING-NoMig` is a simplified version of `GREENING` where we disable the migration function, so that the heuristic becomes very similar to the baseline greedy approach of Algorithm 1, although with energy context information used in the sorting of candidate nodes for job allocation.

- `Random` performs probabilistic placement and does not consider migrations. It considers a random job placement with all nodes having equal probability to be chosen.

- `Free-Green` is a green-energy-aware probabilistic placement that does not consider migrations. In this case, the random job placement assigns probabilities to nodes proportionally to the level of green energy available at the node but yet not assigned to other jobs.

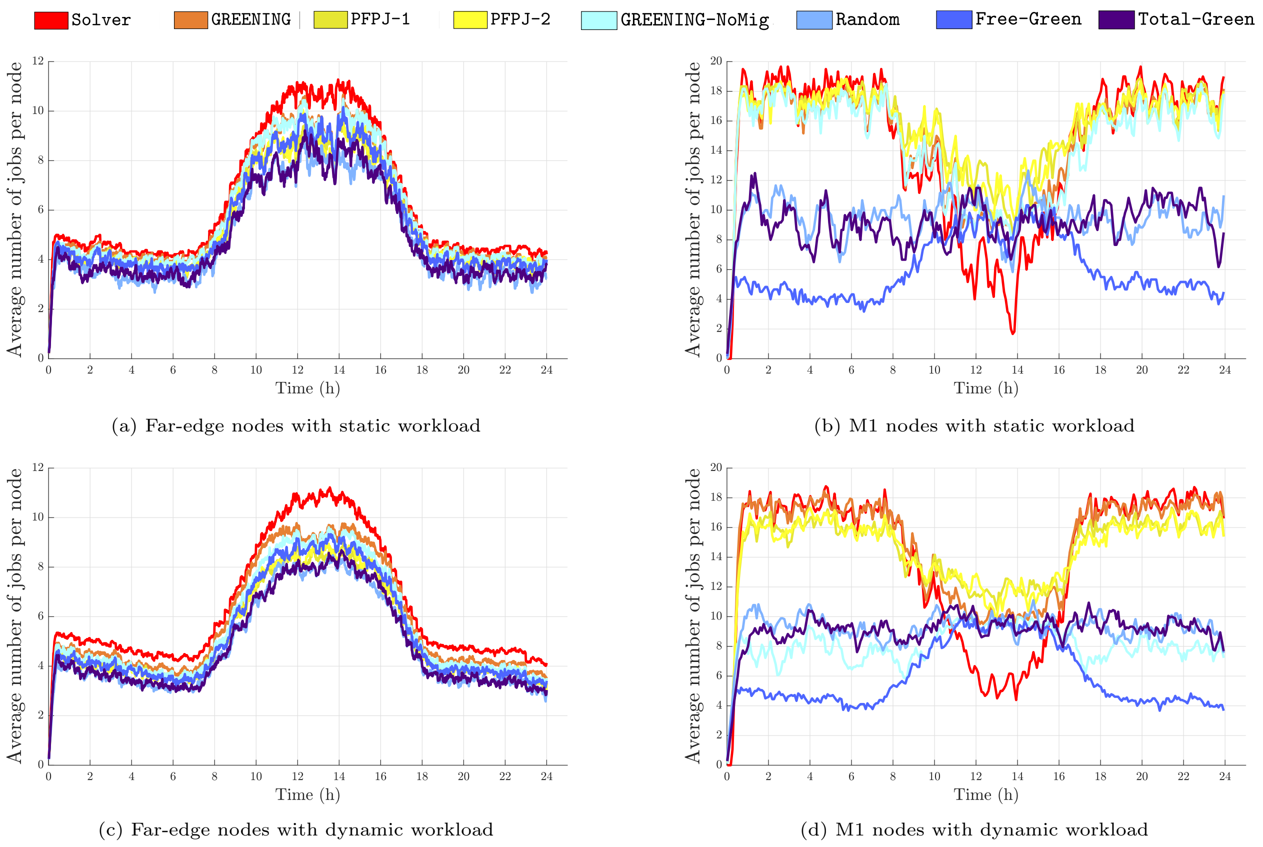
- `Total-Green` is a variant of `Free-Green` in which the random job placement assigns probabilities to nodes proportionally to the total level of green energy available, independently on whether the energy is already in use or not.

To obtain the values reported in this article, `Solver` takes several days on a Dell T640 server with 128 GB of RAM and 40 logical cores with a variable clock rate (but *intlingprog* uses only 1 thread per instance, so we parallelized the number of experiment replicas rather than the single experiment), while all other algorithms need just a few minutes.

6.2. Results

To assess and start comparing the behavior of the 8 algorithms described before, Figure 4 reports the average number of online gaming sessions active at an edge node over time, for a 24-hour period taken at random from the Elia's dataset used in this article. Here we use a baseline network configuration with 9 far-edge nodes, 3 M1 nodes, and a total intensity of arrivals $\lambda = 0.25N$ (expressed in terms of gaming session requests per slot). This load corresponds to a moderately high utilization of edge game resources of about 75% of the total available computing resources. M1 nodes are allowed to use green energy according to its availability, according to the green M1 case described above. Figure 4a shows statistics for far-edge nodes with static workload, while Figure 4b refers to M1 nodes in the same experiment. The figures clearly show a dependency on the availability of green energy, which allows far-edge nodes to host more jobs in the central hours of the day. `Solver` is particularly able to offload jobs to far-edge servers as soon as possible, followed by `GREENING` and `GREENING-NoMig`. `PFPJ-1` and `PFPJ-2` perform similarly, while the other algorithms are less reactive to green energy level changes. From this figure, it is clear that enforcing migrations or not has an impact, although limited. However, the way the algorithms account for the presence of green energy makes a bigger difference. Differences are further exacerbated if we consider the case of dynamic workload in Figures 4c and 4d. In this case, the average of active jobs decreases with all algorithms, which tells that gaming session deviations from the average behavior require more resources, as expected. This effect is particularly detrimental for algorithms that cannot enforce migrations. Indeed, `GREENING-NoMig` drastically reduces the number of active jobs in M1 nodes. Both under static and dynamic workloads, the `Free-Green` algorithm tends to balance the load across all available nodes, so that it is the only algorithm under which the occupancy of M1 nodes increases also in the central hours of the day. `Total-Green` behaves almost as `Random`, because the total level of green energy fluctuates for all nodes following the same daily trend. All other algorithms tend to move jobs to the far-edge when the green energy is more abundant. This also tells that the network is not saturated when the green energy level is higher although the load is quite high. Indeed, consider that, with the parameters of Table 2, a far-edge node can handle up to 15 jobs, on average, while an M1 node can host up to 20 jobs.

To see how the above described behaviors map onto system utility, Figure 5 depicts average results for `GREENING`



(a) Far-edge nodes with static workload

(b) M1 nodes with static workload

(c) Far-edge nodes with dynamic workload

(d) M1 nodes with dynamic workload

Figure 4: Average number of jobs per node for an entire simulated day for the case in which the energy of far-edge nodes is 100% green whereas up to 75% of the energy available at M1 nodes can be green (but in practice only up to $\sim$ 60% in this example), with $N = 12$ nodes, 3 of which are M1 nodes, and $\lambda = 0.25N$.

and the other algorithms based on the trend of Elia's traces for green energy availability as described in Section 6.1.3, for the 4 cases with green or brown M1 nodes and static or dynamic workloads. Here we fix the intensity of job arrivals per node and per slot to $0.25N$, as in Figure 4, and we test different network sizes, up to 48 nodes, although for `Solver` we only report results up to 12 nodes.

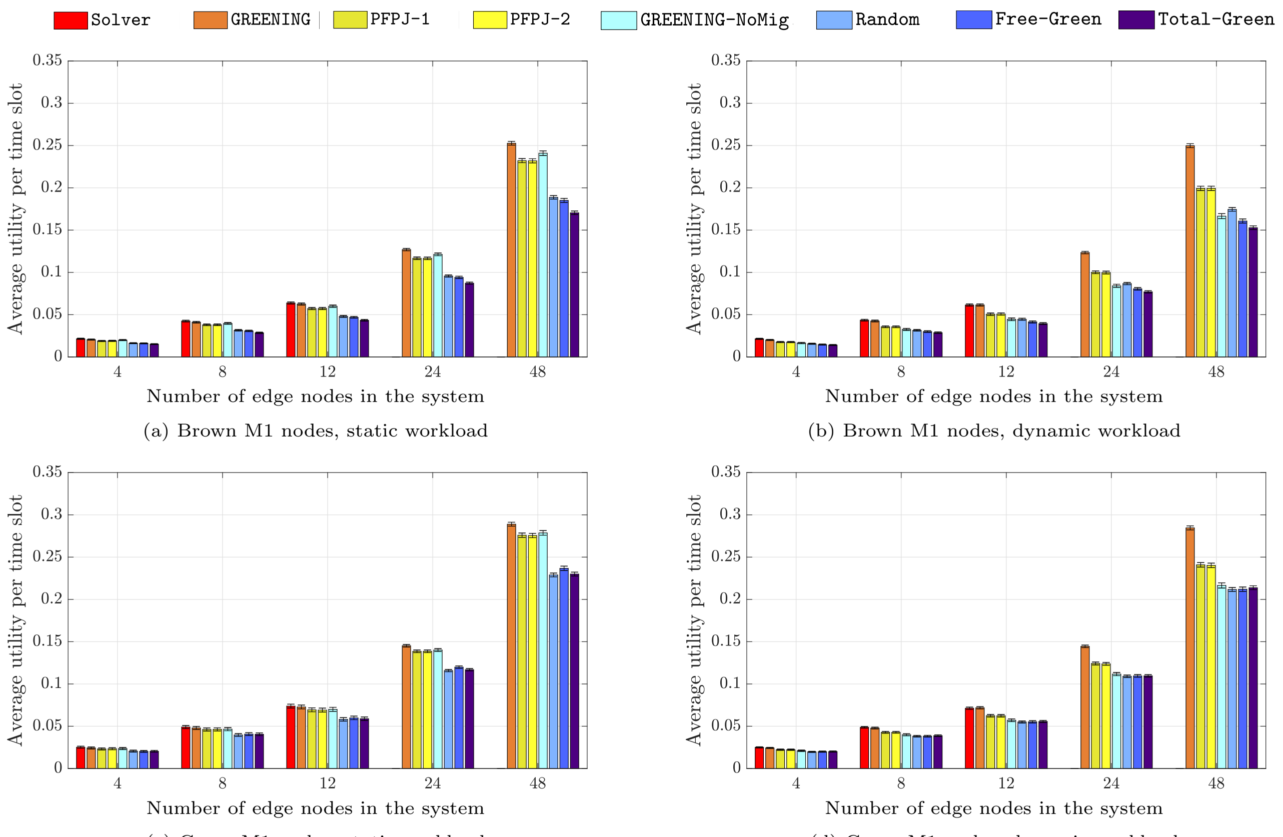
All bar charts in Figure 5 show that the utility increases more or less linearly with the size of the network (note that the intensity of job requests scales linearly as well), although some differences are visible. In particular, while with a tiny network scale and static workloads ($N = 4$ in Figures 5a and 5c) the differences between the 8 algorithms are small, the advantages of `GREENING` become evident as N grows and especially when considering dynamic workloads (see Figures 5b and 5d).

With static workloads, most of the algorithms perform at the same level, with `GREENING` and `Solver` only being slightly superior. This occurs because the revenue deriving from accepting a job is greater than its cost, so eventually all algorithms tend to maximize the amount of accepted jobs. They do that by means of direct placement, with migration only used when strictly necessary.

With dynamic workloads, the importance of timely migration becomes more evident, and performance differences emerge more clearly. For instance, the performance of `GREENING-NoMig`, which is almost as good as `GREENING` under static workloads, here decreases significantly because of the impossibility to perform migrations when the workloads are dynamic. This can cause up to a 35% of utility reduction in the case of dynamic workloads with respect to static workloads with the same average. `PFPJ-1` and `PFPJ-2` suffer dynamic workloads as well because they only enforce migrations from highly loaded to lightly loaded servers. The impairment is less evident, but it becomes substantial as the number of edge nodes increases.

Therefore, Figure 5 shows that well orchestrated migrations are key to achieve good results. In particular, fetching information on the dynamic properties of jobs is key to adapt the optimization on a per-slot-basis, which is what `GREENING` exploits better than the other algorithms because it constantly adapts to changes in green energy levels and game session workloads.

Figure 5 also shows that the impact of having brown vs. green M1 nodes is much higher than the impact of dynamic vs. static workloads, because the use of brown energy entails high costs. Interestingly, `GREENING` is the only algorithm that practically performs the same with static and dynamic workloads and green M1 nodes, and its loss of performance with respect to the case with green M1 nodes and dynamic workload is very close to (or even less than) what observed for `Solver`. We conclude that



(a) Brown M1 nodes, static workload

(b) Brown M1 nodes, dynamic workload

(c) Green M1 nodes, static workload

(d) Green M1 nodes, dynamic workload

Figure 5: Utility comparison as the network size scales up ($\lambda = 0.25N$, far-edge to M1 node ratio equal to 3:1).

GREENING is very robust to the network context, whereas the other heuristics suffer much more.

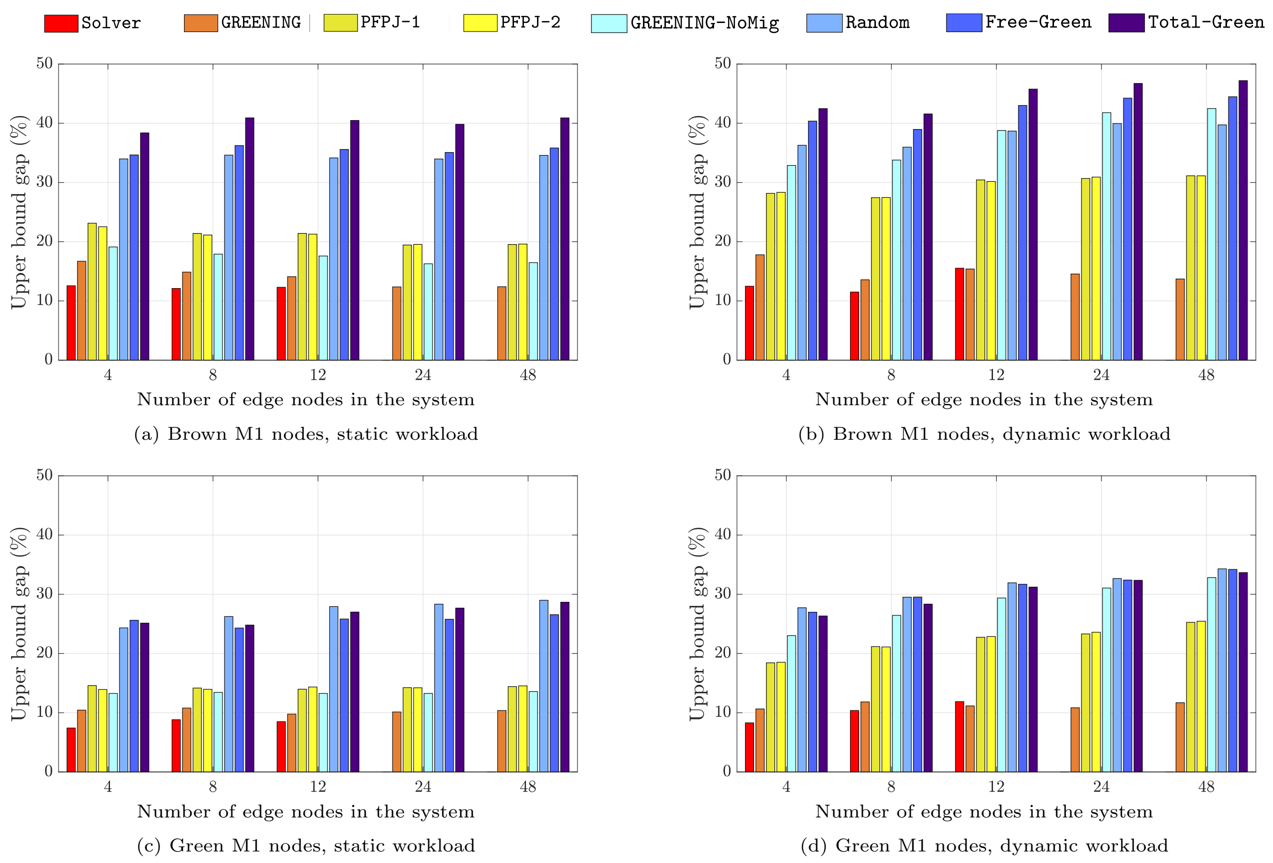
It it important to note that GREENING achieves practically the same utility as Solver in all cases, although with much less complexity. This means that the utility improvement achieved with Solver by offloading more jobs from M1 to far-edge nodes (see Figure 4) has limited importance in most of the cases.

GREENING clearly performs close to Solver, which however is not guaranteed to be optimal because it solves Problem (3) with a very short time horizon (1 time slot only), while the results commented so far are computed for jobs lasting much longer (tens of time slots).[4] Therefore, a question that might arise at this point is: *how far is GREENING from the optimal?* To answer this question, we derive a simple upper bound on the utility. The bound is computed by multiplying the average revenue of a job, minus its deployment cost, times the average number of jobs arriving in a time slot. The above does not account for energy costs, so that we then subtract the cost of the average quantity of energy required by all jobs in a slot, but only for the part that complements the volume of energy available from renewables in the overall system. This bound is optimistic because it assumes that no brown energy is used if there is spare green energy anywhere in the system. This is definitely not the case for brown M1 nodes, and is also an overestimate on the use of green energy for any other node. In addition, the bound neglects the costs of migrations and interruptions.

Figure 6 shows the gap between the utility reported in Figure 5 and the upper bound. The results clearly show that GREENING's distance from the (unfeasible) upper bound is slightly above 10% with brown M1 nodes and at about 10% with green M1 nodes. Solver does only slightly better, while PFPJ-1 and PFPJ-2 pay 5% to 10% more than GREENING and the other heuristics are 3 times less efficient.

As an interesting note, since GREENING and Solver are close to the lower bound, we can argue that our bound must be tight with respect to the optimal. The bound is valid not only for greedy allocations with finite time horizon, but it is valid in general over any time horizon and scheduling of jobs (including for jobs delayed before being deployed), because the bound considers the overall job arrival rate, not just the accepted jobs. Therefore we must conclude that GREENING and Solver are near-optimal with respect to any possible allocation policy, and in particular achieve much better performance than what guaranteed by using any standard greedy algorithm, which cannot guarantee anything better than $1-1/e \simeq 63\%$ of the optimal.

[4]Note that Solver becomes optimal as the cost of migration goes to 0 because in that case the greedy optimization of each time slot becomes optimal, as explained in Section 5.



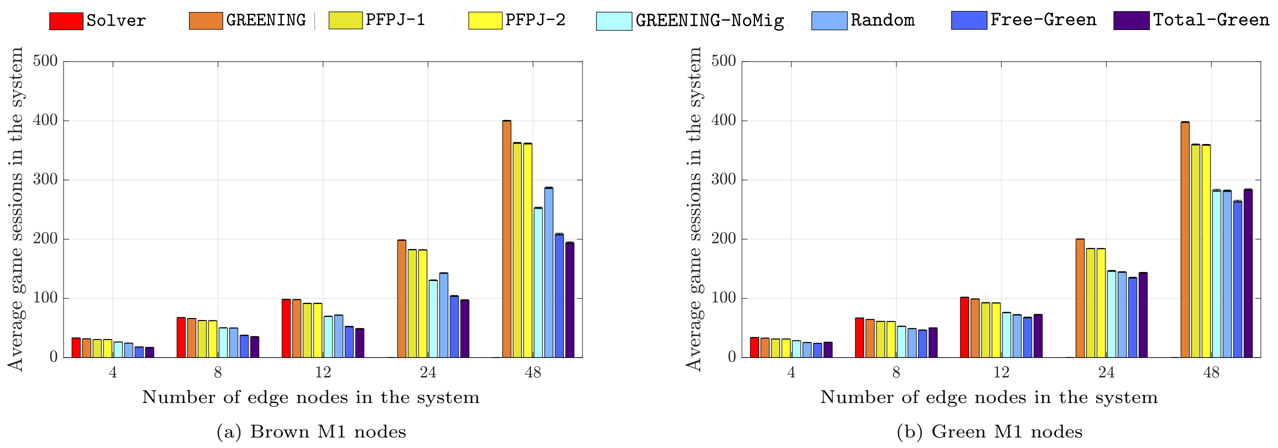
(a) Brown M1 nodes, static workload

(b) Brown M1 nodes, dynamic workload

(c) Green M1 nodes, static workload

(d) Green M1 nodes, dynamic workload

Figure 6: Utility distance from upper bound as the network size scales up ($\lambda = 0.25N$, far-edge to M1 node ratio equal to 3:1).



(a) Brown M1 nodes

(b) Green M1 nodes

Figure 7: Average jobs in the system with dynamic workloads as the network size scales up ($\lambda = 0.25N$, far-edge to M1 node ratio equal to 3:1).

In addition, `GREENING` significantly outperforms state-of-the-art algorithms in the evaluated scenarios.

The above-described results show that static workloads are preferable. However, real workloads are rarely static, and can be used to idealize the service behavior under specific circumstances and with some approximation. For instance, the online game operator might want to allocate resources as the maximum possible rate required by players when there is no information about job dynamics. This is useful and provides users with strict guarantees, but incurs overprovisioning. Therefore, to further analyze the performance of `GREENING` and the other benchmarking algorithms, next we will consider results for the dynamic workload case only, which is more realistic.

The average number of jobs active in the system is depicted in Figure 7 for different cases and as the number of edge nodes increases. `PFPJ-1`, `PFPJ-2` and `GREENING-NoMig` perform almost as `Solver` and `GREENING`, and only `GREENING-NoMig` exhibits a significant degrade of performance. The other algorithms sustain much less jobs,

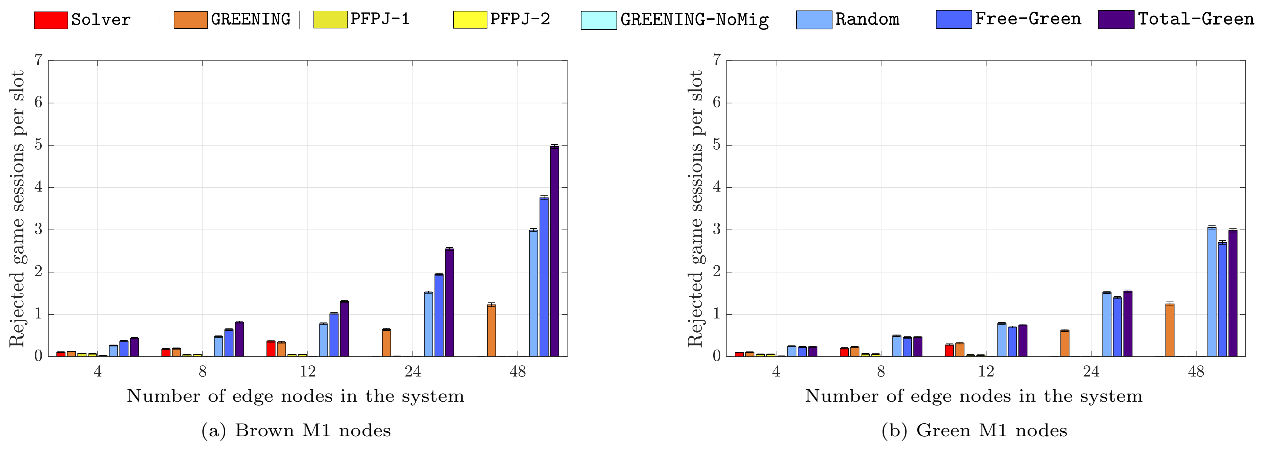


Figure 8: Rejected jobs per time slot with dynamic workload as the network size scales up ($\lambda = 0.25N$, far-edge to M1 node ratio equal to 3:1).

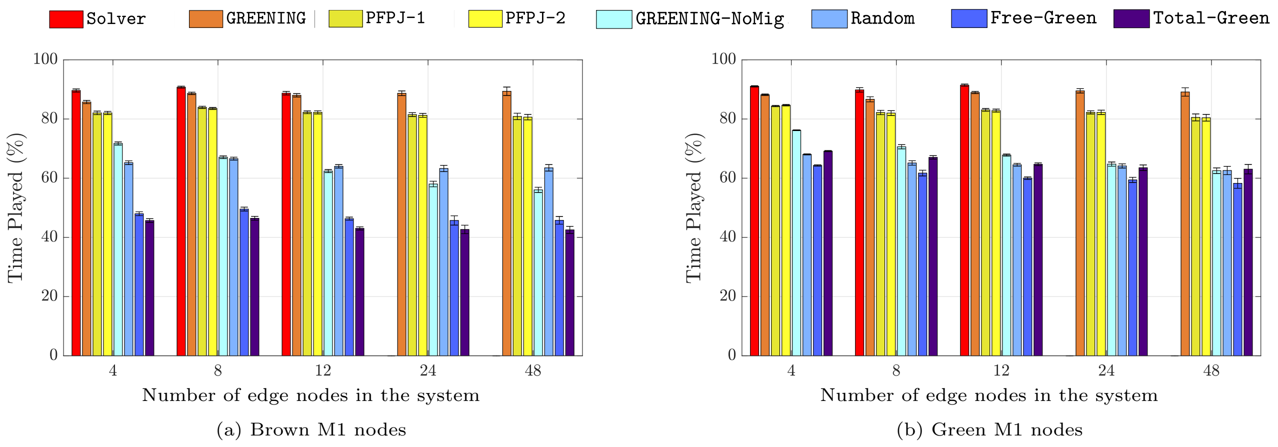


Figure 9: Percentage of game time played with dynamic workloads as the network size scales up ($\lambda = 0.25N$, far-edge to M1 node ratio equal to 3:1).

on average. Therefore, the superiority of `GREENING` with respect to, e.g., `PFPJ-1`, is not due to its capacity to accept more jobs or keeping a higher number of active game sessions. It must reside instead in the ability to use resources better and avoid job interruptions.

To proceed with the analysis of the causes of utility differences between the tested algorithms, we next evaluate job rejections. Figure 8 shows that `Solver` and `GREENING` reject about 10% of online game session requests in all of the cases shown in the figure. For instance, with 12 nodes in total, out of which 3 are (either green or brown) M1 nodes, there are 0.3 rejected jobs per slot out of $\lambda = 0.25 \cdot 12 = 3$ requests per slot. Interestingly, `Solver` and `GREENING` can reject more jobs than `PFPJ-1`, `PFPJ-2` and `GREENING-NoMig`. The other heuristics are much worse. For instance, with brown M1 nodes, the `Total-Green` rejects about 42% of requests (e.g., with 48 nodes, the total arrival rate is 12 requests per slot, out of which about 5 are rejected), and 25% with green M1 nodes (e.g., with 48 nodes, about 3 requests out of 12 are rejected). Therefore, against intuition, high numbers of active jobs do not necessarily mean less rejected jobs. In fact, it is more convenient to reject some jobs rather than interrupt them.

Rejecting more jobs must occur because `Solver` and `GREENING` have on average more jobs allocated and less jobs interrupted, therefore having less space to admit new jobs. To evaluate the correctness of this deduction, let us observe the time played for accepted jobs, normalized to the nominal duration of all jobs, as shown in Figure 9. Indeed, the figure shows that `GREENING` and `Solver` guarantee the highest percentage of time played, i.e., less and shorter job interruptions. `Solver` misses about the 10% of played time, which corresponds to the 10% of rejected jobs visible in Figure 8. Here, `GREENING` misses an additional 2% to 3% of played time with respect to `Solver`, which is therefore due to rare job interruptions in addition to the $\sim$ 10% of rejected jobs. Other heuristics accept more jobs, but then they are forced to abruptly interrupt many more jobs, thus experiencing lower utilities. In particular, probabilistic allocation algorithms reject and interrupt many more jobs, which is due to the fact that they can allocate jobs to nodes that are close to saturation. Specifically, `Total-Green` incurs about 13% to 16% of interruptions, as it yields a time played as low as 42–45% with brown M1 nodes and $\sim 60\%$ with green M1 nodes (and, as noted before, about 25% and 42% of the missing played time is due to rejections for the cases with brown and green M1 nodes, respectively. Similarly, the other random algorithms cause interruptions for about 15% to 20% of the time. `PFPJ-1` and `PFPJ-2`, whose rejection rate is negligible, still have a played time of about 80%, which implies a 20% of losses due to interruptions of jobs. This hints to the fact that considering migrations only when a server is highly loaded, as done in `PFPJ-1` and `PFPJ-2` with different metrics, is more expensive and less effective than enforcing migrations continuously, as done with `GREENING` and even more massively with `Solver`.

To generalize the conclusions drawn so far, we eventually evaluate the impact of the load, by varying λ and the ratio between far-edge and M1 nodes in the system.

The results of Figure 10 were obtained with 4 different levels of load and report the utility per time slot, normalized to the number of nodes. At $\lambda/N = 0.1$ requests per node per time slot, the system is underloaded and all algorithms perform similarly, with `GREENING` being a bit better than the others, for all values of N considered in the figure. The case $\lambda/N = 0.25$ is the one described before in this section, which shows that `GREENING` offers non-negligible gains, especially as the network size increases. The remaining two cases, with $\lambda/N = 0.4$ and $\lambda/N = 0.6$, are cases in which the network is lightly or heavily overloaded, respectively. In those cases, `GREENING` is always the best choice, offering at least 20% more utility than other algorithms with at least 12 edge nodes in the system. It is interesting to note that, as the network saturates, random allocations become competitive with respect to baseline algorithms with migrations, `PFPJ-1` and `PFPJ-2`. This behavior occurs because, differently from `GREENING`, these heuristics do not attempt to minimize energy costs.

Figure 11 compares different far-edge to M1 node ratios, starting with the case 3:1 considered so far in the article, except here $\lambda = 0.3N$. M1 nodes are green and the game sessions exhibit dynamic workloads. The ranges for the number of nodes evaluated in each plot are different as it was not possible to use a common range that satisfied all ratios exactly. The case 3:1 is the one in which the gain of `GREENING` is the least, whereas the gain can become much higher under higher ratios like 6:1 or 9:1. Small ratios are appropriate to model early stage MEC deployment scenarios in 5G networks, due to the cost of deploying MEC hosts and controllers. Instead, much higher ratios are foreseen for future releases of 5G and beyond. Therefore, we can conclude that while `GREENING` can offer decent gain in limited deployment frameworks, its potential would be truly unleashed as the roll-out of 5G and beyond 5G networks progresses.

7. Conclusions

In this article, we have studied the green edge gaming concept and how to maximize the utility by leveraging MEC-like facilities and locally generated green energy. We have formulated a *multi*-constrained integer-linear problem for the one-shot allocation of game sessions to servers, and shown that it is NP-hard in strong sense. The problem is however sub-modular over a single step time-horizon, and practically also over any time-horizon optimization instance, at least if migration costs are limited with respect to energy costs and/or per-job revenues. This fact might encourage the use of greedy heuristics with strict performance guarantees (of the order of $1 - 1/e$). However, we have shown that it is possible to sensibly improve performance by exploiting energy context information and timely migrations. In particular, we have shown that the green energy component is key to drive the optimization of job allocations and migrations. Moreover, a dynamic optimization is needed to account for energy level dynamics in green energy generation as well as in job power absorption. As network size and load increase, and far-edge nodes become largely prevalent in number, our proposed algorithm, `GREENING`, largely outperforms state-of-the-art approaches and achieves near-optimal results with very low complexity. Notably, without the possibility to timely migrate online game sessions, which is a feature of the edge context, the greening of online gaming could not be a viable solution.

References

[1] F. Spinelli, V. Mancuso, A Migration Path Toward Green Edge Gaming, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) (WoWMoM 2022), Belfast, United Kingdom (Great Britain), 2022.

[2] J. Moar, Will cloud gaming change the way we play?, White Paper, Juniper Research Ltd (2020) 1–7.

[3] ETSI, Multi-Access Edge Computing (MEC); Framework and Reference Architecture, Tech. rep., ETSI MEC ISG (jan 2019).

[4] F. Spinelli, V. Mancuso, Toward enabled industrial verticals in 5G: A survey on MEC-based approaches to provisioning and flexibility, IEEE Communications Surveys Tutorials 23 (1) (2021) 596–630. `doi:10.1109/COMST.2020.3037674`.

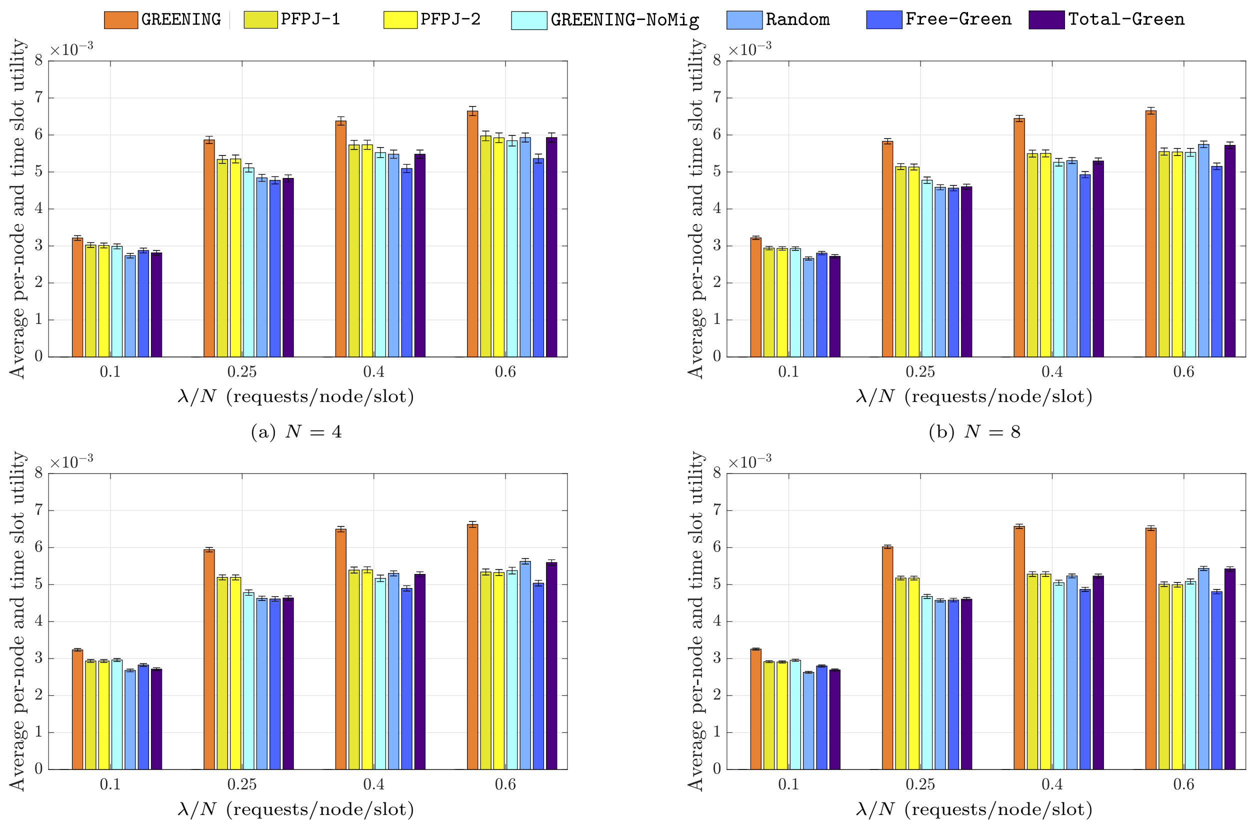
[5] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, D. O. Wu, Improving Cloud Gaming Experience through Mobile Edge Computing, IEEE Wireless Communications 26 (4) (2019) 178–183. `doi:10.1109/MWC.2019.1800440`.

[6] E. Mills, N. Bourassa, L. Rainer, J. Mai, A. Shehabi, N. Mills, Toward Greener Gaming: Estimating National Energy Use and Energy Efficiency Potential, The Computer Games Journal 8 (3) (2019) 157–178. `doi:10.1007/s40869-019-00084-2`.

[7] Q. Wu, G. Y. Li, W. Chen, D. W. K. Ng, R. Schober, An Overview of Sustainable Green 5G Networks, IEEE Wireless Communications 24 (4) (2017) 72–80. `doi:10.1109/MWC.2017.1600343`.

[8] T. Huang, W. Yang, J. Wu, J. Ma, X. Zhang, D. Zhang, A survey on green 6G network: Architecture and technologies, IEEE Access 7 (2019) 175758–175768. `doi:10.1109/ACCESS.2019.2957648`.

[9] Y. Zhang, P. Qu, J. Cihang, W. Zheng, A Cloud Gaming System Based on User-Level Virtualization and Its Resource Schedul-



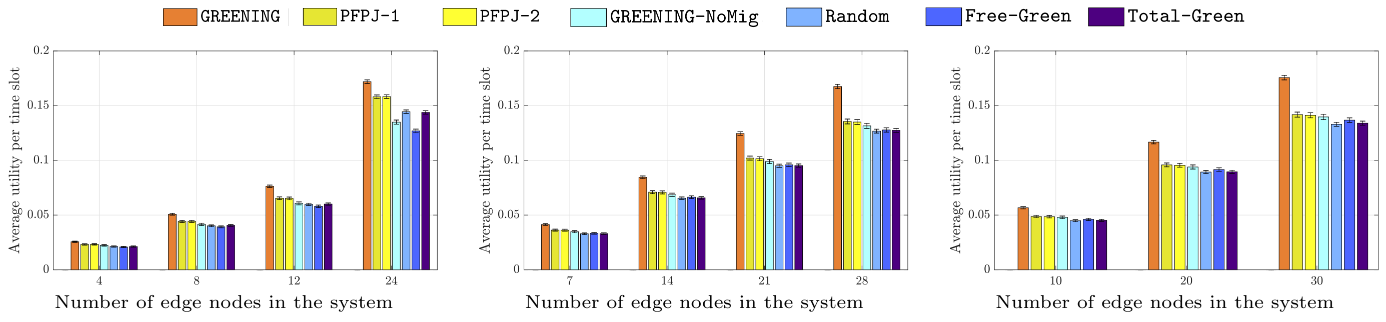
(a) $N = 4$

(b) $N = 8$

(c) $N = 12$

(d) $N = 24$

Figure 10: Utility comparison as the arrival rate λ size scales up for different network sizes, with dynamic workloads and far-edge to M1 node ratio equal to 3:1, and green M1 nodes.



(a) Far-edge to M1 ratio: 3:1

(b) Far-edge to M1 ratio: 6:1

(c) Far-edge to M1 ratio: 9:1

Figure 11: Utility per time slot with green M1 nodes and dynamic workload at $\lambda = 0.3N$ and different far-edge to M1 node ratios.

ing, IEEE Transactions on Parallel and Distributed Systems 27 (5) (2016) 1239–1252. doi:10.1109/TPDS.2015.2433916.

[10] T. Kämäräinen, Y. Shan, M. Siekkinen, A. Ylä-Jääski, Virtual machines vs. containers in cloud gaming systems, in: 2015 International Workshop on Network and Systems Support for Games (NetGames), 2015, pp. 1–6. doi:10.1109/NetGames.2015.7382987.

[11] P. J. Braun, S. Pandi, R.-S. Schmoll, F. H. P. Fitzek, On the study and deployment of mobile edge cloud for tactile Internet using a 5G gaming application, in: 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC), 2017, pp. 154–159. doi:10.1109/CCNC.2017.7983098.

[12] T. Braud, A. Alhilal, P. Hui, Talaria: In-engine synchronisation for seamless migration of mobile edge gaming instances, in: Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 375–381. doi:10.1145/3485983.3494848.

[13] Y. Li, Y. Deng, X. Tang, W. Cai, X. Liu, G. Wang, Cost-Efficient Server Provisioning for Cloud Gaming, ACM Trans. Multimedia Comput. Commun. Appl. 14 (3s) (Jun. 2018). doi:10.1145/3190838.
URL https://doi.org/10.1145/3190838

[14] Y. Deng, Y. Li, X. Tang, W. Cai, Server Allocation for Multiplayer Cloud Gaming, in: Proceedings of the 24th ACM International Conference on Multimedia, MM '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 918–927. doi:10.1145/2964284.2964301.
URL https://doi.org/10.1145/2964284.2964301

[15] D. Wu, Z. Xue, J. He, *iCloudAccess*: Cost-Effective Streaming of Video Games From the Cloud With Low Latency, IEEE Transactions on Circuits and Systems for Video Technology

24 (8) (2014) 1405–1416. doi:10.1109/TCSVT.2014.2302543.

[16] Y. Li, C. Zhao, X. Tang, W. Cai, X. Liu, G. Wang, X. Gong, Towards Minimizing Resource Usage With QoS Guarantee in Cloud Gaming, IEEE Transactions on Parallel and Distributed Systems 32 (2) (2021) 426–440. doi:10.1109/TPDS.2020.3024068.

[17] H. Chen, X. Zhang, Y. Xu, J. Ren, J. Fan, Z. Ma, W. Zhang, T-gaming: A Cost-Efficient Cloud Gaming System at Scale, IEEE Transactions on Parallel and Distributed Systems 30 (12) (2019) 2849–2865. doi:10.1109/TPDS.2019.2922205.

[18] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, C.-H. Hsu, Placing Virtual Machines to Optimize Cloud Gaming Experience, IEEE Transactions on Cloud Computing 3 (1) (2015) 42–53. doi:10.1109/TCC.2014.2338295.

[19] Y. Han, D. Guo, W. Cai, X. Wang, V. Leung, Virtual Machine Placement Optimization in Mobile Cloud Gaming through QoE-Oriented resource competition, IEEE Transactions on Cloud Computing (2020) 1–1doi:10.1109/TCC.2020.3002023.

[20] S.-P. Chuah, C. Yuen, N.-M. Cheung, Cloud gaming: a green solution to massive multiplayer online games, IEEE Wireless Communications 21 (4) (2014) 78–87. doi:10.1109/MWC.2014.6882299.

[21] H. Guan, J. Yao, Z. Qi, R. Wang, Energy-efficient SLA guarantees for virtualized GPU in cloud gaming, IEEE Transactions on Parallel and Distributed Systems 26 (9) (2015) 2434–2443. doi:10.1109/TPDS.2014.2350499.

[22] K. Bilal, A. Erbad, Edge computing for interactive media and video streaming, in: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), 2017, pp. 68–73. doi:10.1109/FMEC.2017.7946410.

[23] L. Lin, X. Liao, H. Jin, P. Li, Computation Offloading Toward Edge Computing, Proceedings of the IEEE 107 (8) (2019) 1584–1607. doi:10.1109/JPROC.2019.2922285.

[24] R. D. Yates, M. Tavan, Y. Hu, D. Raychaudhuri, Timely cloud gaming, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057197.

[25] K. Li, H. Zheng, J. Wu, Migration-based virtual machine placement in cloud systems, in: 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), 2013, pp. 83–90.

[26] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, K. Poularakis, Service Placement and Request Scheduling for Data-Intensive Applications in Edge Clouds, IEEE/ACM Transactions on Networking 29 (2) (2021) 779–792. doi:10.1109/TNET.2020.3048613.

[27] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, L. Tassiulas, Service Placement and Request Routing in MEC Networks With Storage, Computation, and Communication Constraints, IEEE/ACM Transactions on Networking (2020) 1–14.

[28] D. Wang, X. Tian, H. Cui, Z. Liu, Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware MEC network, China Communications 17 (8) (2020) 31–44. doi:10.23919/JCC.2020.08.003.

[29] I. Labriji, F. Meneghello, D. Cecchinato, S. Sesia, E. Perraud, E. C. Strinati, M. Rossi, Mobility Aware and Dynamic Migration of MEC Services for the Internet of Vehicles, IEEE Transactions on Network and Service Management 18 (1) (2021) 570–584. doi:10.1109/TNSM.2021.3052808.

[30] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, K. Li, Distributed Task Migration Optimization in MEC by Extending Multi-Agent Deep Reinforcement Learning Approach, IEEE Transactions on Parallel and Distributed Systems 32 (7) (2021) 1603–1614. doi:10.1109/TPDS.2020.3046737.

[31] R. Li, X. Li, J. Xu, F. Jiang, Z. Jia, D. Shao, L. Pan, X. Liu, Energy-aware decision-making for dynamic task migration in MEC-based unmanned aerial vehicle delivery system, Concurrency and Computation: Practice and Experience n/a (n/a) (2020) e6092. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6092, doi:https://doi.org/10.1002/cpe.6092.

[32] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, K. K. Leung, Dynamic Service Migration in Mobile Edge Computing Based on Markov Decision process, IEEE/ACM Transactions on Networking 27 (3) (2019) 1272–1288. doi:10.1109/TNET.2019.2916577.

[33] G. J. L. Paulraj, S. A. J. Francis, D. Peter, I. J. Jebadurai, Resource-aware virtual machine migration in IoT cloud, Future Generation Computer Systems 85 (2018) 173–183. doi:https://doi.org/10.1016/j.future.2018.03.024.

[34] L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, W. Dai, Energy efficient task allocation and energy scheduling in green energy powered edge computing, Future Generation Computer Systems 95 (2019) 89–99. doi:https://doi.org/10.1016/j.future.2018.12.062.

[35] ITU-T, Consideration on 5G transport network reference architecture and bandwidth requirements, Tech. rep., ITU-T Study Group (feb 2018).

[36] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, A. Y. Zomaya, On Enabling Sustainable Edge Computing with Renewable Energy Resources, IEEE Communications Magazine 56 (5) (2018) 94–101. doi:10.1109/MCOM.2018.1700888.

[37] Y. Mao, Y. Luo, J. Zhang, K. B. Letaief, Energy harvesting small cell networks: feasibility, deployment, and operation, IEEE Communications Magazine 53 (6) (2015) 94–101. doi:10.1109/MCOM.2015.7120023.

[38] D. Renga, M. Meo, Dimensioning Renewable Energy Systems to Power Mobile Networks, IEEE Transactions on Green Communications and Networking 3 (2) (2019) 366–380. doi:10.1109/TGCN.2019.2892200.

[39] Wind power production estimation and forecast on belgian grid (near real-time), White paper, Elia Transmission Belgium SA, https://opendata.elia.be/explore/dataset/ods086/information/, [Accessed: 2022-03-30].

[40] Sun power production estimation and forecast on belgian grid (near real-time), White paper, Elia Transmission Belgium SA, https://opendata.elia.be/explore/dataset/ods087/information/, [Accessed: 2022-03-30].

[41] D. Finkel, M. Claypool, S. Jaffe, T. Nguyen, B. Stephen, Assignment of games to servers in the OnLive cloud game system, in: 2014 13th Annual Workshop on Network and Systems Support for Games, 2014, pp. 1–3.

[42] K.-T. Chen, Y.-C. Chang, H.-J. Hsu, D.-Y. Chen, C.-Y. Huang, C.-H. Hsu, On the Quality of Service of Cloud Gaming Systems, IEEE Transactions on Multimedia 16 (2) (2014) 480–495. doi:10.1109/TMM.2013.2291532.

[43] E. Mills, N. Bourassa, L. Rainer, J. Mai, A. Shehabi, N. Mills, Energy consumption of cloud games, Data from work [6], https://docs.google.com/spreadsheets/d/134WlGZK3tuXFnh3igJz-L3nmpBJzXg3PznGXZagtbm4, [Accessed: 2022-03-30].

[44] E. Mills, N. Bourassa, L. Rainer, J. Mai, A. Shehabi, N. Mills, Supplemental information for the article entitled "Toward Greener Gaming: Estimating National Energy Use and Energy Efficiency Potential", Data from work [6], https://docs.google.com/spreadsheets/d/1ZXLwHuWodc6EsROfgXnOgbDkYnSHB_StEGyA36GiklM, [Accessed: 2022-03-30].

[45] S. Martello, Knapsack Problems: Algorithms and Computer Implementations, Wiley-Interscience series in discrete mathematics and optimiza tion (1990).

[46] M. L. Fisher, G. L. Nemhauser, L. A. Wolsey, An analysis of approximations for maximizing submodular set functions—I, Springer Berlin Heidelberg, Berlin, Heidelberg, 1978, pp. 265–294. doi:10.1007/BF01588971.

[47] NVIDIA RTX Blade Server, Data Sheet, NVIDIA Corporation, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/cloud-gaming-server/geforce-now-rtx-server-gaming-datasheet.pdf, [Accessed: 2022-03-30].

[48] Google Stadia Website, Website, Google Stadia, https://stadia.google.com, [Accessed: 2022-03-30].

[49] C. E. Clark, The PERT model for the distribution of an activity

time, Operations Research 10 (3) (1962) 405–406.