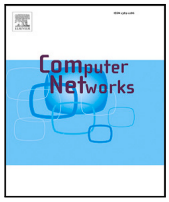
Computer Networks 252 (2024) 110675

Contents lists available at ScienceDirect

Computer Networks

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



LESS-ON: Load-aware edge server shutdown for energy saving in cellular networks

Blas Gómez [a,*], Suzan Bayhan [b], Estefanía Coronado [a,c], José Villalón [a], Antonio Garrido [a]

[a] High-Performance Networks and Architectures, Universidad de Castilla-La Mancha, Albacete, Spain
[b] Faculty of EEMCS, University of Twente, Enschede, The Netherlands
[c] I2CAT Foundation, Barcelona, Spain

ARTICLE INFO

Keywords:
Edge computing
Energy efficiency
5G
Wireless networks
Sustainable communications

ABSTRACT

While advances in wireless networks enable novel services with previously unreachable latency guarantees, edge computing becomes essential for delivering computing resources close to the users and meeting the strict latency requirements. However, addressing the energy footprint of computing resources is crucial amid the pressing sustainability concerns. The energy consumption of idle resources accounts for a significant part of the total energy footprint. While server shutdown during low-demand periods is common in cloud computing, it is challenging to determine which edge servers to shut down and how to route requests due to the stringent latency requirements of the applications. Thus, this work formulates an optimal orchestration policy to minimize the energy consumption of the edge computing infrastructure and presents LESS-ON, a strategy with a polynomial time complexity that reduces the operational energy footprint of edge computing by shutting down edge servers during low-demand periods. In contrast to previous studies, LESS-ON considers the energy requirements associated with routing requests to the designated edge servers. Our numerical evaluation shows that LESS-ON reduces the total consumption by 42% with respect to the common always-on approach during low-demand periods and by 35% over 24 h, all while meeting latency requirements.

1. Introduction

Recent advances in wireless networks supporting low latency, high connection density, and multi-Gbps data rates have the potential to greatly enhance the user experience by enabling MNOs and third parties to offer innovative applications with strict latency requirements that previous generation networks were unable to meet [1]. Examples of such applications include vehicular safety and medical robotics. In this evolving landscape, edge computing is key to building smarter digital infrastructures. By bringing computational resources closer to the user, it enables more responsive applications, improves data processing efficiency, and reduces latency. However, this involves the deployment of numerous edge servers in the Radio Access Network (RAN) with the associated increase in the energy demands of the infrastructure. In a context in which global warming and the energy crisis have raised significant concerns regarding the energy consumption of communication and computing infrastructures [2–5], addressing the energy consumption of edge computing becomes paramount.

The energy consumption of computing systems has attracted significant attention, especially in cloud computing [6–10]. Prior works have shown significant energy savings by shutting down servers during low-demand periods, as a substantial portion of their energy use corresponds to their idle state [11]. Resources in a cloud data center are scaled to meet demand during peak hours, leaving plenty of resources idle in off-peak hours. However, thanks to their centralized nature and their abstraction as a single server, resources can easily be scaled up and down by transparently shutting down servers during off-peak hours. Similarly, edge computing involves the deployment of numerous servers with their capacity scaled to meet peak-hour demand. Yet, unlike cloud computing, server shutdown strategies have received less attention in edge computing. Due to its ultra-low latency requirements and distributed nature, not all edge servers are potential candidates to handle a request uploaded to the RAN by a user connected to a specific BS. To shut an edge server down, other servers must be reachable within the maximum delay of the services. Moreover, those servers must have enough capacity available to host the extra load. Yet, there is limited research on strategies to route requests to other edge servers aiming to shut down idle ones. For this reason, edge servers always stay on and are ready to receive user requests to ensure that the

* Corresponding author.
E-mail address: blas.gomez@uclm.es (B. Gómez).

https://doi.org/10.1016/j.comnet.2024.110675
Received 16 November 2023; Received in revised form 13 June 2024; Accepted 25 July 2024
Available online 27 July 2024

ultra-low latency requirements are fulfilled, especially as the routing of 5G networks with integrated edge deployments rely on these servers for user plane routing [12]. Consequently, a considerable fraction of resources remain idle during off-peak hours. Thus, introducing server shutdown strategies can have a considerable impact in reducing the energy footprint of computing infrastructure as a whole. The distributed nature of edge computing involves the deployment of edge servers across thousands of locations to cover extensive geographical areas, as the proximity between users and computing resources is critical. Thus, addressing the energy consumption of edge computing's idle servers becomes paramount.

Studies such as [13,14] have explored the possibility of shutting down edge servers, but an assessment of its impact on Quality of Service (QoS) and especially application deadline satisfaction has yet to be carried out. Moreover, these studies overlook the energy consumption of routing requests through backhaul links from offline servers to active ones. Thus, to effectively apply server shutdown strategies to the edge, there is a need for a comprehensive strategy that routes user requests to the lowest possible number of servers to minimize the energy consumption of the edge computing infrastructure, considering both edge servers and backhaul links, while satisfying QoS requirements.

To address this, this paper presents LESS-ON, an orchestration framework that reduces the energy consumption of a cellular network's co-located edge computing infrastructure by selectively shutting down edge servers while ensuring that the remaining active servers meet the application's computing demands and delay bounds. LESS-ON shuts down servers by routing requests to the smallest set of servers possible, accounting for the delays that the requests will experience on the path from the source BS toward the selected active edge server and during computation as well as the energy consumption of this transmission. LESS-ON ensures that the maximum delay experienced by a request is below the tolerable delay of that service. LESS-ON also considers the energy consumption caused by the booting of previously inactive edge servers. In summary, our contributions are threefold:

i. To the best of our knowledge, LESS-ON is the first edge server shutdown strategy that considers the energy consumption related to routing requests and optimizes the assignment and routing of computing requests from their source cells to edge servers, taking into account the available capacity of the servers and the backhaul links.
ii. We formulate the optimal orchestration policy for energy consumption minimization, and then, we design an efficient heuristic with polynomial time complexity to solve the formulated problem. We solve the minimization problem using Gurobi optimization software and compare our heuristic's performance with the optimal solution.
iii. We use MintEDGE [15], our in-house edge computing simulator, to evaluate the performance of LESS-ON in terms of energy savings and deadline satisfaction. MintEDGE is released under a permissive MIT license. We leverage real data from an MNO in the Netherlands using two other strategies as a benchmark, namely *Always-on* (where edge servers are constantly active) and *Threshold* (where servers are shut down when their load is below a threshold as presented in [13]). We also study the impact of the number of deployed edge servers to evaluate the effect of various utilization levels in the infrastructure on deadline satisfaction and the energy-saving potential.

The rest of the paper is organized as follows. Section 2 reviews the related work, while Section 3 presents the system model. Section 4 describes the problem formulation and Section 5 introduces our approach to solving it. Section 6 presents and discusses the performance evaluation results. Finally, Section 7 concludes the paper and includes a list of future work.

2. Related work

The sustainability of computing infrastructure has been in the focus of recent research. Prior studies on cloud computing [2–5] highlight the benefits of edge computing in reducing WAN traffic and the role of backhaul transmissions in energy usage. Moreover, shutting down network devices during low-demand periods emerges as a promising energy-saving approach. However, these works also acknowledge that implementing such strategies in the realm of strict latency requirements presents new challenges.

In view of these challenges, a considerable body of literature presents new request routing and resource management strategies to reduce the energy consumption of edge computing architectures. In [16], the authors jointly consider the routing of requests from users to edge servers and the server resource allocation to minimize the energy consumption of the edge servers. However, unlike LESS-ON, their work does not consider shutting down edge servers or the energy consumption of the backhaul. The authors of [17] present another resource allocation strategy to minimize energy consumption. Their strategy jointly considers the energy of computing and communication of Virtual Machines (VMs) running in an edge infrastructure operating under hard delay constraints. However, while LESS-ON focuses on reducing the system's energy consumption by eliminating the idle consumption of unused edge servers, their work does not consider this consumption and never turns edge servers off. In [18], the authors study service placement and resource allocation in an edge environment with different actors with competing interests, such as service providers, application providers, or network providers. However, while their work has a positive impact on energy consumption, they address neither routing nor idle energy consumption.

In [19] a resource allocation strategy that considers clusters of edge servers is presented. In this strategy, the cluster head is in charge of routing requests to the appropriate edge server to minimize energy consumption by leveraging Dynamic Voltage Frequency Scaling (DVFS) CPUs to save energy while maintaining delay levels. However, similar to previous work, edge servers are never shut down. This strategy does not consider the backhaul energy consumption, and the resulting delays are in the order of hundreds of milliseconds, which are too high for many of the ultra-low latency edge use cases [20]. In contrast, LESS-ON targets both the idle energy consumption of edge servers and the energy resulting from routing requests through the backhaul. Moreover, contrary to [19], LESS-ON is hardware-agnostic. Another approach that requires specific hardware is presented in [21]. This strategy assumes that each edge server has multiple CPUs that can be independently powered off based on different thresholds. This method maximizes idle servers and relies on specialized hardware to minimize idle energy consumption. However, while this partially alleviates idle energy consumption, the servers are never completely off; therefore, there is still potential to reduce the energy consumption further. LESS-ON explores this potential by fully shutting them down, eliminating idle energy consumption. In [22], the authors propose a location-based load prediction algorithm that uses historical edge server data. While the authors acknowledge the potential benefits of using this algorithm in server shutdown strategies, they do not explore the benefits mentioned. Instead, they focus on evaluating the prediction strategy employed.

While the previous works present resource allocation and routing strategies aimed at reducing energy consumption, none addresses idle resource consumption. With capacity scaled to meet the demands of peak hours, there is potential for further improvements if this energy consumption is eliminated. To the best of our knowledge, this issue is only addressed by the following two studies. The approach in [13] puts edge servers to sleep when the load drops below 10% threshold. However, it overlooks the energy consumption of the backhaul needed to route incoming requests to active servers, which may outweigh the savings achieved by server shutdown [2]. Furthermore, they do not assess the impact on latency-constrained services. In contrast, LESS-ON

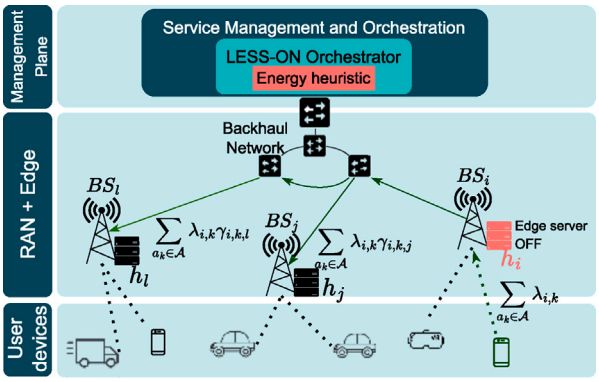


Fig. 1. A cellular network with edge servers. The orchestrator determines how to route the requests to the assigned edge servers and which servers to shut down to minimize the network's energy consumption.

balances the energy saved by server shutdown with the energy used for backhaul routing to active servers. Moreover, LESS-ON considers the strict delay requirements of applications by accounting for the routing and computing time of requests in the new servers before the shutdown decisions. We study the impact of LESS-ON on maximum delay fulfillment. The authors of [14] propose a server shutdown strategy aimed at minimizing costs instead of energy. Their study assumes fixed running costs and shows latencies that are unsuitable for latency-constrained use cases such as vehicular safety. In contrast, our approach focuses specifically on applications with strict delay requirements. Moreover, LESS-ON considers an architecture that places edge servers closer to the user, and our evaluation models energy consumption dynamically.

Shutdown or sleep strategies have been studied in other areas of research, including BSs and cloud servers as shown by the considerable body of literature [23,24]. Within BSs, shutdown strategies for heterogeneous networks have attracted considerable attention [25–28]. The authors in [25] formulate a minimization problem of the energy consumption of the RAN in a heterogeneous network by putting Small Base Stations (SBSs) to sleep or turning them off when their traffic can be served by neighboring SBSs. They solve the optimization problem using two metaheuristics: Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). In [26], the authors also rely on a heterogeneous network to decouple control and data communication. They also put to sleep or turn off SBSs according to a vacation time predicted to minimize the energy consumption of the RAN. The authors of [27] present another method for saving energy by turning off SBSs using graph theory. In [28], the authors consider the same network architecture using BSs and present two algorithms that put the SBSs with the lowest load to sleep during random periods considering that each User Equipment (UE) has more than one SBS in range. Cloud server shutdown strategies have also been explored in the research literature [29,30]. For instance, the authors of [29] present a cloud server shutdown strategy based on thermal management and server utilization. Similarly, the authors of [30] present a Dynamic Round-Robin algorithm to consolidate VMs in the smallest number of servers to shut down the idle ones.

3. System model

This section presents the system model considered in this work and its components, as illustrated in Fig. 1.

Communication network: We consider a 5G RAN comprising a finite set of BSs, denoted by $\mathcal{B} = \{BS_1, \ldots, BS_N\}$. The BSs are interconnected with each other and with the management plane via a backhaul network represented by a given set of links $\mathcal{L}$. We denote the connectivity graph as $\mathcal{G} = \langle \mathcal{B}, \mathcal{L} \rangle$. Each link $\ell_{i,j}$ is characterized by its source and destination BS together with its associated capacity, denoted by $\alpha_{i,j}$. Hence, we represent each link as a three tuple: $\ell_{i,j} = \langle BS_i, BS_j, \alpha_{i,j} \rangle$. We assume that the capacity of the links is equal in both directions. For the sake of simplicity, we assume a fixed routing algorithm, denoting the associated routing matrix by Γ. Any pair of BSs can communicate with each other through multiple hops according to the shortest path in Γ. In this work, we will refer to the BS that a user is connected to as serving BS. This generic representation of the RAN results in a RAN-agnostic approach that facilitates broad applicability across diverse network architectures, offering greater flexibility and future-proofing, as it can be seamlessly adapted to evolving network architectures. This work focuses on the edge infrastructure and its resource management, independent of the underlying specific RAN architecture.

Edge computing infrastructure: The cellular network incorporates a co-located edge server deployment, as shown in Fig. 1. We denote the set of edge servers as $\mathcal{H}$. Without loss of generality, we assume that each edge server is associated with a BS. Because an MNO can choose not to deploy edge servers at every BS, we indicate whether a particular BS_i hosts an edge server with binary variable e_i, where $e_i = 1$ if BS_i hosts an edge server and $e_i = 0$ otherwise. We assume each edge server $h_m \in \mathcal{H}$ can host a finite set $\mathcal{A}$ of computation services. We denote the computing capacity of h_m by C_m^{max} in operations per second.[1] The system also includes a set $\mathcal{U}$ of UEs. Any $u_l \in \mathcal{U}$ can access services on any edge server h_m using its BS and the corresponding backhaul links. The edge server at BS_i is denoted as h_i, and can be represented by the tuple $\langle BS_i, C_i^{max} \rangle$. Edge servers can be shut down and booted as needed for energy-saving purposes. The booting process incurs a setup time, denoted by T_i^s, during which the edge servers cannot handle computing requests. Consequently, the orchestrator must consider T_i^s before seeing h_i as eligible to attend computing requests. In this work, we will refer to the edge server receiving a request as the serving or target edge server. The serving edge server can be hosted on a different BS to the serving BS.

Computing requests: We define the set of all computing services as $\mathcal{A}$, which consists of various tasks such as risk detection in connected vehicles and augmented reality. The computing services are placed in the network edge, and UEs access these services to execute their tasks. Following the state of the art [31,32], we represent each computing request by a four-tuple: $\langle o_k, V_k^{in}, V_k^{out}, T_k^{max} \rangle$, where o_k represents the workload, V_k^{in} indicates the input size in bytes, V_k^{out} denotes the size of the computation output, and T_k^{max} specifies the delay budget of the task. A request made to $a_k \in \mathcal{A}$ results in a computing workload of o_k operations required to complete the task. The specific value of o_k depends on the nature of the service. Assuming that requests for a particular service a_k have an arrival rate at BS_i of $\lambda_{i,k}$, the resulting workload caused by a_k per unit of time is given by $o_k \lambda_{i,k}$. The arrival rate is influenced by factors such as the number of users and service characteristics. An edge server located at BS_i might serve computing requests originating from other BSs, e.g., a BS without an edge server or whose edge server is off. We assume that the computing capacity of h_i is divided among the hosted services proportionally to the number of operations to be executed by each one [33]. If the orchestrator cannot find the free capacity to attend to all the requests, they are rejected in a First In First Out (FIFO) manner. We assume requests are either attended on the edge or rejected and computed locally by the user. We do not consider rerouting them to the cloud as we consider latency-sensitive services that require real-time processing. Each computing request is also identified by the volumes of the input (V_k^{in}) and the computation output (V_k^{out}) data. As requests might be associated with delay-sensitive tasks, we denote the delay budget for a_k by T_k^{max}.

[1] This work focuses on real-time applications that are more CPU-intensive than storage-intensive. For this reason, we have assumed that all services can be hosted on all servers regardless of the storage capabilities.

Table 1
Summary of key notation.

Symbol	Definition
$\mathcal{B}$, N	The set of BSs and the number of BSs
$\mathcal{L}$	The set of links connecting BSs
$\mathcal{G}$	BS connectivity graph
$\ell_{i,j}$	The link connecting BS_i and BS_j
$\alpha_{i,j}$	The capacity of the link connecting BS_i and BS_j
Γ	Routing Matrix
$\mathcal{H}$, h_m	The set of edge servers and a server in $\mathcal{H}$
e_i	Variable marking the presence of an edge server at BS_i
$\mathcal{A}$, a_k	The set of services and a service in $\mathcal{A}$
C_m^{max}	The maximum capacity of h_m in operations per second
O_m	Total workload of h_m
o_k	Workload in operations of a request to a_k
V_k^{in}	Size of the input for a request to a_k
V_k^{out}	Size of the output for a request made to a_k
T_k^{max}	The delay budget for a request made to a_k
$\lambda_{i,k}$	Arrival rate of requests from BS_i to service a_k
$\gamma_{i,k,j}$	Fraction of requests for a_k routed from BS_i to h_j
η_i	Status (on/off) of edge server $h_i \in \mathcal{H}$
$\beta_{k,j}$	Fraction of CPU allocated for service a_k at h_j
$p_{i,j}^{o,p}$	Variable marking if $\ell_{o,p}$ is in the path from BS_i to BS_j
$T_{i,k}^c$	Time required to compute a response to a request
$T_{i,k}^u$	Time to upload a request in the RAN
$T_{i,k}^r$	Time to route requests through the backaul
$T_{i,k}^o$	Time to route back the output of the computation
$T_{i,k}^d$	Time to download the output from the serving BS
E_m^{idle}	Idle energy consumption of h_m
E_m	Energy consumed by operation executed in h_m
$\sigma_{o,p}$	Energy consumed per bit transmitted by $\ell_{o,p} \in \mathcal{L}$
T_m^s	Time necessary to turn on h_m
E_m^{boot}	Energy consumed to boot $h_m \in \mathcal{H}$

The delay budget indicates the time before a request's result must be delivered to the submitting user.[2]

Operation of the edge orchestrator: The edge computing infrastructure is managed by an orchestrator located at the Service Management and Orchestration (SMO) layer, which also encloses the government of the RAN and the transport network.[3] It periodically gathers information from the RAN and the edge servers, including latency and utilization metrics. It is also in charge of managing the capacity of the edge servers and selecting the appropriate one for the deployment, relocation, or termination of services. To optimize energy consumption, the orchestrator can dynamically shut down edge servers as needed. When the edge server h_i associated with BS_i is shut down, computing requests received by BS_i are reassigned to another active edge server h_j. The selection of the appropriate active edge server is based on factors such as link capacities and the available capacity of the active edge servers, as shown in Fig. 2.

At a given time, the orchestrator decides on the configuration of the infrastructure to minimize the energy footprint while meeting the latency constraints of the computing requests and resource constraints of the links and the edge servers. The decision is communicated to the infrastructure so that it can operate according to the planned orchestration in the next time slot.

4. Problem formulation

To ensure the timely completion of computing requests, we use edge server capacities and routing delays to calculate service completion time and assign requests to specific edge servers. Our work aims to minimize the energy footprint by assigning requests to suitable edge servers and potentially shutting down selected servers while meeting delay requirements. This section introduces our system's latency and capacity constraints and presents the energy consumption model. A list of key notations is provided in Table 1.

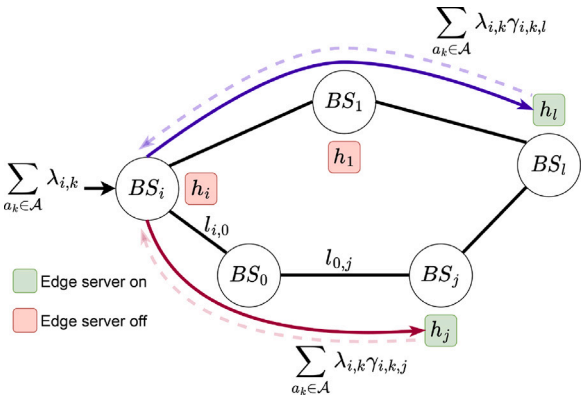


Fig. 2. Requests are forwarded to active edge servers based on link capacities and server locations. The computation results are delivered back to the user through the same path. In this example, since the edge server hosted at BS_i is off, requests are forwarded to either BS_l or to BS_j with active servers.

At a given time, the orchestrator can decide on the following parameters to optimize the QoS or save energy:

- Fraction of requests for a_k received by BS_i to be computed by an edge server h_j, as illustrated in Fig. 2. We denote this fraction by $\gamma_{i,k,j} \in [0, 1]$.
- Status of each edge server denoted by $\eta_i \in \{0, 1\}$, where $\eta_i = 0$ means that h_i is off.
- Fraction of computing capacity allocated for a_k at h_j. We denote this fraction by $\beta_{k,j} \in [0, 1]$.

Computing capacity constraints: These constraints express both the requests' need for computing resources and the capacity limitations of the edge servers. With the said orchestrator settings, if h_j receives a fraction $\gamma_{i,k,j}$ of computing requests for a_k from BS_i (see example in Fig. 2), the total number of operations per second needed to be completed by service a_k at h_j is given by:

$$O_{k,j} = \sum_{BS_i \in \mathcal{B}} \gamma_{i,k,j} o_k \lambda_{i,k} \quad \forall a_k \in \mathcal{A}. \tag{1}$$

Consequently, considering all services, we can calculate the total computing load at h_j as:

$$O_j = \sum_{a_k \in \mathcal{A}} O_{k,j} \quad \forall a_k \in \mathcal{A}. \tag{2}$$

To guarantee that the computing capacity of h_j is not exceeded, the orchestrator must ensure that the total workload assigned to h_j is less than or equal to its capacity. This constraint can be expressed formally as follows:

$$O_j \leq C_j^{max} \quad \forall BS_j \in \mathcal{B}. \tag{3}$$

To serve all incoming requests, h_j assigns a fraction $\beta_{k,j}$ of the total capacity C_j^{max} to service a_k. We assume that $\beta_{k,j}$ is proportional to the total workload of each service with respect to the total capacity C_j^{max} of h_j. Thus, to avoid internal queuing on the CPU, the following constraint must be satisfied:

$$\beta_{k,j} \geq \sum_{BS_i \in \mathcal{B}} \frac{\gamma_{i,k,j} o_k \lambda_{i,k}}{O_j}, \ \forall h_j \in \mathcal{H}, \forall a_k \in \mathcal{A}. \tag{4}$$

Link capacity constraints: To respect the limited capacity of each link $\ell_{i,j}$, the orchestrator must consider the data size of a computing request and the routing path from the serving BS to the target edge server. We define a binary variable $p_{i,j}^{o,p}$ which yields 1 if $\ell_{o,p}$ is part of the shortest

[2] We implicitly assume that the result is delivered to the same user submitting the request. Other patterns, such as one-to-many, can also be considered. However, we focus on this model for the sake of simplicity.

[3] Federation across MNOs is beyond the scope of this work.

path between BS_i and BS_j, and 0 otherwise. For each link $\ell_{o,p}$ with a capacity $\alpha_{o,p}$, the following capacity constraint must be satisfied to avoid long transmission delays:

$$\sum_{a_k \in \mathcal{A}} \sum_{BS_i \in \mathcal{B}} \sum_{BS_j \in \mathcal{B}} V_k^{in} \lambda_{i,k} \gamma_{i,k,j} p_{i,j}^{o,p} \leqslant \alpha_{o,p} \quad \forall \ell_{o,p} \in \mathcal{L}. \tag{5}$$

Delay budget constraints: Each service has a specific deadline T_k^{max} that must be met after a request is submitted. The orchestrator needs to assign resources so the delay experienced is below T_k^{max}. The delay experienced consists of:

(i) time to upload a request and the associated data to the serving BS (T^u);
(ii) time to route this request toward the serving edge server (T^r);
(iii) time required to compute the response (T^c);
(iv) time to route back the output (T^o); and
(v) time to download the output from the serving BS (T^d).

When the serving BS also hosts the serving edge server, T^r is 0. Since the output data is significantly smaller than the input, we assume a constant delay for T^o and T^d and focus on the first three delay components:

- T^u: Given the datarate R_i of the radio link at BS_i that can be obtained with Shannon–Hartley's theorem, $T_{i,k}^u$ is given by:

$$T_{i,k}^u = \frac{V_k^{in}}{R_i} \qquad \text{[seconds]}. \tag{6}$$

- T^r: The time to route a request for a service a_k from the serving BS_i to the target edge server h_j through $\ell_{o,p}$ depends on the link capacity $\alpha_{o,p}$ and the size of the input V_k^{in}. For a request routed from BS_i to BS_j, we can calculate the backhaul latency as the sum of the delays of all the links along the path.[4] More formally, we compute the total delay of this path as follows:

$$T_{i,k,j}^r = \sum_{\ell_{i,j} \in \mathcal{L}} \frac{V_k^{in} p_{i,j}^{o,p}}{\alpha_{o,p}}. \tag{7}$$

Since requests for a_k can take different paths and be executed on different edge servers ($\gamma_{i,k,j} > 0$ for several h_j), the orchestrator needs to ensure that the path with the maximum latency is below the delay budget. We can represent this as follows:

$$T_{i,k}^r = \max_{\ell_{i,j}} T_{i,k,j}^r \quad \forall BS_j \text{ where } \gamma_{i,k,j} \geqslant 0. \tag{8}$$

- T^c: The computing time required to process a request made for service a_k depends on the fraction $\beta_{k,j}$ of computing capacity assigned to that service at edge server h_j and the total server capacity C_j^{max}. Since requests for service a_k can be computed on different edge servers, the computing delay for a_k is given by the maximum delay observed among all the edge servers. Therefore, the execution time is given by:

$$T_{i,k}^c = \max_{h_j} \left(\eta_j \frac{o_k}{\beta_{k,j} C_j^{max}} \right) \text{ where } \gamma_{i,k,j} > 0. \tag{9}$$

Using the maximum delay for each component provides the upper bound of the total delay for a_k, which must adhere to its delay budget T_k^{max}, as expressed by:

$$T_{i,k}^u + T_{i,k}^r + T_{i,k}^c + T_{i,k}^o + T_{i,k}^d \leqslant T_k^{max} \; \forall a_k \in \mathcal{A}, \forall BS_i \in \mathcal{B}. \tag{10}$$

4.1. Energy consumption model

The system's energy consumption can be divided into three components:

(i) energy consumed by edge servers during request execution;
(ii) energy consumed by the backhaul links to route requests to edge servers hosted at other BSs; and
(iii) energy used in the boot process when an edge server is shut down and restarted.

We provide further details of these components below.

Edge servers: Any edge server h_m has a capacity of C_m^{max} operations per second and a baseline energy consumption E_m^{idle} when it is idle. Each operation executed on h_m results in extra energy consumption in addition to E_m^{idle}. We denote this energy per operation as E_m. As mentioned above, each edge server h_m performs a certain number of operations per second O_m, given by (1). Thus, if we assume a linear power model [34], the power consumed by the set of active edge servers is given by:

$$\sum_{h_m \in \mathcal{H}} \eta_m \left(E_m^{idle} + O_m E_m \right). \tag{11}$$

Routing of the computing requests: When the edge server hosted at a particular BS is off, or the BS does not host an edge server, the requests need to be routed to another BS's edge server, resulting in additional energy consumption. The energy overhead of this communication can be calculated according to a hardware-specific parameter σ that denotes energy consumption for transmitting one bit (typically in J/bit).[5] We denote the per-bit transmission energy for a link $\ell_{o,p}$ by $\sigma_{o,p}$. The total data volume traversing a link can be determined by considering both the computing request data and its output as follows:

$$V_{o,p} = \sum_{a_k \in \mathcal{A}} \sum_{BS_i \in \mathcal{B}} \sum_{BS_j \in \mathcal{B}} \left(V_k^{in} + V_k^{out} \right) \lambda_{i,k} \gamma_{i,k,j} p_{i,j}^{o,p} \quad \forall \ell_{o,p} \in \mathcal{L}. \tag{12}$$

Thus, with the amount of data traversing $\ell_{o,p}$ and its $\sigma_{o,p}$ the energy consumed by the set of links $\mathcal{L}$ is given by:

$$\sum_{\ell_{o,p} \in \mathcal{L}} \sigma_{o,p} V_{o,p}. \tag{13}$$

Boot process: When the orchestrator shuts down an edge server, it must ensure that the energy saved during the inactive period outweighs the energy used in the booting process. An inactive edge server h_m takes a certain setup time T_m^s until it is ready to serve requests again. The orchestrator turns the edge servers back on in a timely manner to avoid the rejection of requests during the booting process. During this setup time, the power consumption is assumed to have a constant value P_m^s [35]. Therefore, the total energy consumed during the edge server boot process is given by $E_m^{boot} = T_m^s P_m^s$. The booting energy is consumed only upon state changes and is negligible when the time slot between state changes is big enough. For this reason, we do not include it in the formulation of the problem but instead account for it when deciding on the time period between the orchestrator's decisions.

4.2. Orchestration for energy consumption minimization

Now, let us formally define our optimization problem, which aims to minimize the computing infrastructure's energy consumption without violating the constraints introduced in the preceding sections. Given the decision variables γ, η, and β, we can state the optimal orchestration problem as follows:

$$\min_{\gamma,\eta,\beta} \sum_{h_i \in \mathcal{H}} \eta_i (E_i^{idle} + O_i E_i) + \sum_{\ell_{o,p} \in \mathcal{L}} \sigma_{o,p} V_{o,p} \tag{$\mathcal{P}$1}$$

[4] Requests might experience queuing delays if the capacity constraints are violated. We assume node processing and propagation delays to be negligible.

[5] Since detailed energy consumption models for the network equipment are missing in the literature, we assume the following model considering previous studies [34]. More sophisticated models, such as non-linear relationships between energy consumption and data volume, can be used to model the energy consumption of the backhaul.

subject to:

$$O_j \leq C_j^{max} \qquad \forall BS_j \in \mathcal{B} \quad (14)$$

$$\beta_{k,j} \geq \sum_{BS_i \in \mathcal{B}} \frac{\gamma_{i,k,j} o_k \lambda_{j,k}}{O_j} \forall h_j \in \mathcal{H}, \forall a_k \in \mathcal{A} \quad (15)$$

$$\sum_{a_k \in \mathcal{A}} \sum_{BS_i \in \mathcal{B}} \sum_{BS_j \in \mathcal{B}} (V_k^{in} + V_k^{out}) \lambda_{i,k} \gamma_{i,k,j} p_{i,j}^{o,p} \leq \alpha_{o,p} \forall \ell_{o,p} \in \mathcal{L} \quad (16)$$

$$T_{i,k}^u + T_{i,k}^r + T_{i,k}^c + T_{i,k}^o + T_{i,k}^d \leq T_k^{max} \forall a_k \in \mathcal{A}, \forall BS_i \in \mathcal{B} \quad (17)$$

$$\sum_{a_k \in \mathcal{A}} \beta_{k,j} \leq \eta_j \qquad \forall h_j \in \mathcal{H} \quad (18)$$

$$\sum_{BS_j \in \mathcal{B}} \gamma_{i,k,j} = 1 \qquad \forall a_k \in \mathcal{A}, \forall BS_i \in \mathcal{B} \quad (19)$$

$$\gamma_{i,k,j} \leq \eta_j \qquad \forall a_k \in \mathcal{A}, \forall BS_i \in \mathcal{B}, \forall h_j \in \mathcal{H} \quad (20)$$

$$\gamma_{i,k,j} \in [0,1] \qquad \forall BS_i \in \mathcal{B}, \forall a_k \in \mathcal{A}, \forall BS_j \in \mathcal{B} \quad (21)$$

$$\beta_{k,j} \in [0,1] \qquad \forall a_k \in \mathcal{A}, \forall BS_j \in \mathcal{B} \quad (22)$$

$$\eta_i \in \{0,1\} \qquad \forall h_i \in \mathcal{H}. \quad (23)$$

The objective function in ($\mathcal{P}1$) minimizes the total energy consumption of edge servers and the communication network. Constraints (14), (15), (16) and (17) ensure compliance with the capacity and delay constraints introduced in this section. Constraint (18) indicates that the sum of all the assigned capacities cannot exceed the total capacity. Rejecting all requests would yield optimal energy consumption, as all servers could be shut down. To avoid this, we introduce Constraint (19), which states that all requests must be routed to an edge server, ensuring that all requests are attended. We state that an edge server that is off ($\eta_j = 0$) cannot receive requests with Constraint (20). Constraints (21), (22), and (23) define the domain of the decision variables.

Eq. ($\mathcal{P}1$) is a Mixed-Integer Linear Program (MILP), which is hard to solve optimally in real-time in realistic scenarios. If we assume that $|\mathcal{A}| = 1$ (i.e., there is just one service) and the link capacities are sufficiently large (i.e., $\alpha_{i,j} = \infty \; \forall \ell_{i,j}$), the solution for the formulated problem can solve a Capacitated Facility Location Problem (CFLP). A CFLP is the problem of opening a number of facilities with limited capacity (edge servers) to serve the demand of all clients (BSs) at a minimum cost (energy consumption). In this case, the CFLP problem would involve

(i) determining the subset of edge servers to activate and
(ii) establishing which edge server will attend each request.

Since the CFLP is NP-hard [36], the solution to Problem ($\mathcal{P}1$) is NP-hard, which leads us to devise a lower-complexity algorithm, which we present in the next section.

5. LESS-ON: Load-aware Edge Server ShutdOwN

This section presents our proposed operation flow and a lower complexity heuristic to solve ($\mathcal{P}1$). As illustrated in Fig. 1, the energy-saving heuristic is hosted on the orchestrator, which is located at the SMO layer, which also encloses the government of the RAN and the transport network. This gives the orchestrator a global view of the network, which it uses to gather demand information from the infrastructure. This information is used as input for the energy-saving heuristic, which returns new resource allocation and routing configurations. The orchestrator then applies this new configuration to the infrastructure.

Algorithm 1: LESS-ON energy saving heuristic

Input: λ, O
γ, β, $\eta \leftarrow$ `Alg.2` (λ)
Sort $\mathcal{H}$ according to E^{idle} in descending order
Sort $\mathcal{A}$ according to T^{max} in ascending order
for each h_j **in** $\mathcal{H}$ **do**
  $E^{on} \leftarrow \eta_j(E_j^{idle} + O_j E_j)$
  $E^{off} \leftarrow 0$
  $\eta_j \leftarrow 0$
  for each a_k **in** $\mathcal{A}$ **do**
    for each BS_i **in** $\mathcal{B}$ **do**
      if $\gamma_{i,k,j} > 0$ **then**
        $\mathcal{H}^{can} \leftarrow$ Servers reachable within T^{max} sorted by $\sigma_{i,m}$
        for each h_m **in** $\mathcal{H}^{can}$ **do**
          if $\lambda_{i,k}\gamma_{i,k,j} V_k^{in} \leq \alpha_{i,m}$ **then**
            if $\gamma_{i,k,j} o_k \lambda_{i,k} < C_m^{max} - O_m$ **then**
              $\gamma_{i,k,m} \leftarrow \gamma_{i,k,m} + \gamma_{i,k,j}$
              $\gamma_{i,k,j} \leftarrow 0$
            else
              $\gamma_{i,k,m} \leftarrow \gamma_{i,k,m} + \frac{C_m^{max} - O_m}{\lambda_{i,k} o_k}$
              $\gamma_{i,k,j} \leftarrow \gamma_{i,k,j} - \gamma_{i,k,m}$
            $\alpha_{i,m} \leftarrow \alpha_{i,m} - V_k^{in} \lambda_{i,k} \gamma_{i,k,m}$
          else
            if $\gamma_{i,k,j} o_k \lambda_{i,k} < C_m^{max} - O_m$ **then**
              $\gamma_{i,k,m} \leftarrow \gamma_{i,k,m} + \frac{\lambda_{i,k} V_k^{in} - \alpha_{i,j}}{\alpha_{i,j}}$
              $\gamma_{i,k,j} \leftarrow \gamma_{i,k,j} - \gamma_{i,k,m}$
              $\alpha_{i,m} \leftarrow 0$
            else
              $\gamma_{i,k,m} \leftarrow \gamma_{i,k,m} + \gamma_{i,k,j}$
              $\gamma_{i,k,j} \leftarrow 0$
              $\alpha_{i,m} \leftarrow \alpha_{i,m} - V_k^{in} \lambda_{i,k} \gamma_{i,k,m}$
            $E^{off} \leftarrow E^{off} + E_{i,k,m}^{off}$
            Update O_m and O_j
      if $E^{off} \geq E^{on}$ ***or*** $\sum_{BS_i \in \mathcal{B}} \sum_{a_k \in \mathcal{A}} \gamma_{i,k,j} \neq 0$ **then**
        $\eta_j \leftarrow 1$
        Undo changes
for each h_m **in** $\mathcal{H}$ **do**
  for each a_k **in** $\mathcal{A}$ **do**
    $\beta_{k,j} \leftarrow \sum_{BS_i \in \mathcal{B}} \frac{\gamma_{i,k,j} o_k \lambda_{i,k}}{O_j}$
if any $T^u + T^r + T^c + T^o + T^d \leq T^{max}$ **then**
  γ, β, $\eta \leftarrow$ `Alg.3` $(\gamma, \beta, \eta, \lambda)$
return γ, β, η

5.1. Energy saving heuristic

LESS-ON's heuristic, outlined in Alg. 1, is adapted from the DROP greedy algorithm introduced in [37] for solving CFLP. By reducing the problem to a CFLP, we gain access to a rich pool of well-studied and efficient heuristics. In particular, the DROP-based heuristic provides a good approximation to the optimal solution in polynomial time, achieving a good trade-off between complexity and potential runtime limitations and prioritizing a practical solution that achieves results within a reasonable time frame. Alg. 1 begins with an initialization step described in Alg. 2 to identify the initial values of γ, β, and η. Alg. 2 proceeds as follows. First, all edge servers are turned on ($\eta_i = 1 \; \forall h_i \in \mathcal{H}$) and requests received at each BS_i are routed to the co-located server h_i for the BSs that host one ($e_i = 1$). If the local server does not have enough capacity ($\lambda_{i,k} o_k \geq C_i^{max}$), the remaining requests are routed to the closest BS_j with an edge server. Only a subset of candidate servers denoted by $\mathcal{H}^{can}$, consisting of the BSs reachable within the delay constraint of a_k, is considered. If the closest server, h_j, cannot accommodate all the requests from BS_i, the algorithm routes as many requests as possible to that server and then proceeds to the second-closest server in terms of delay. This process continues until all requests are served or all h_j have been considered. If there is

Algorithm 2: Initialization algorithm

```
   Input: λ
1  η_j ← 1 for each h_j in H
2  for each BS_i in B do
3      for each a_k in A do
4          if e_i is 1 then
5              if λ_{i,k} o_k ≤ C_i^max − O_i then
6                  γ_{i,k,i} ← 1
7                  Continue with the next iteration
8              else
9                  γ_{i,k,i} ← (C_i^max − O_i) / (λ_{i,k} o_k)
10         H^can ← Servers reachable within T^max sorted by T^r_{i,k,m}
11         for each h_j in H^can do
12             if λ_{i,k} V_k^in ≤ α_{i,m} then
13                 if O_{i,k,j} < C_j^max − O_j then
14                     γ_{i,k,j} ← 1 − Σ_{BS_j∈B} γ_{i,k,j}
15                 else
16                     γ_{i,k,i} ← (C_i^max − O_i) / (λ_{i,k} o_k)
17                 α_{i,j} ← α_{i,j} − λ_{i,k} V_k^in
18             else
19                 γ_{i,k,j} ← (λ_{i,k} V_k^in − α_{i,j}) / α_{i,j}
20                 α_{i,j} ← 0
21             λ_{i,k} ← (1 − γ_{i,k,j}) λ_{i,k}
22             Update O_j
23 for each h_m in H do
24     for each a_k in A do
25         β_{k,j} ← Σ_{BS_i∈B} (γ_{i,k,j} o_k λ_{i,k}) / O_j
26 return γ, β, η
```

insufficient capacity to handle all the requests, the remaining ones are rejected in a FIFO manner.

After initialization, Alg. 1 sorts the set of servers $\mathcal{H}$ based on their E^{idle} to prioritize the shutdown of those with higher idle energy consumption. Similarly, the set of services $\mathcal{A}$ is sorted according to their delay budget (T^{max}) to prioritize those services with stricter deadlines. It then assesses the feasibility of shutting down each server (line 4 in Alg. 1). When h_j is shut down, the requests of all a_k received at any BS_i (lines 8 and 9) that were previously served by h_j need to be rerouted to other edge servers (i.e., $\gamma_{i,k,j} > 0$). The algorithm then identifies a subset $\mathcal{H}^{can}$ of candidates to serve the requests previously being served by h_j. $\mathcal{H}^{can}$ includes all the edge servers that can be reached by BS_i within T_k^{max}. $\mathcal{H}^{can}$ is sorted based on the path with the lowest energy consumption indicated by $\sigma_{i,m}$, which is given by:

$$\sigma_{i,m} = \sum_{l_{o,p} \in \mathcal{L}} \sigma_{o,p} p_{i,m}^{o,p} \quad \forall \ell_{o,p} \in \mathcal{L}. \tag{24}$$

For each candidate server h_m (line 12), Alg. 1 checks the free capacity $\alpha_{i,m}$ of the path from BS_i to BS_m (line 13). The path's capacity $\alpha_{i,m}$ is given by the minimum capacity of all links on the path, i.e., $\min_{\ell_{o,p}} \alpha_{o,p}\ \forall \ell_{o,p}$ where $p_{i,m}^{o,p} = 1$. If there is enough capacity in the link, the algorithm checks the server's capacity. The free capacity on the server is given by the difference between its maximum capacity C_m^{max} and the capacity in use O_m (line 14). If both the link and server capacities are sufficient, all requests initially routed to h_j are redirected to h_m. However, if the capacities are insufficient, the algorithm routes as many requests as possible to h_m and attempts to distribute the remaining requests among the other eligible candidates in the next iterations.

The algorithm tracks the energy consumption E^{off} resulting from shutting down h_j and compares it with the energy resulting from keeping it on (E^{on}). In this way, the algorithm ensures that the backhaul energy does not rise above the energy saved by shutting down h_j. If $E^{off} \geq E^{on}$, the server h_j is kept on, and the routing changes are rejected (line 32). In addition, the heuristic checks whether all the requests formerly attended to by h_j are routed to other edge servers ($\sum_{BS_i \in \mathcal{B}} \sum_{a_k \in \mathcal{A}} \gamma_{i,k,j} = 0$). If that is not the case, the changes are also rejected to ensure enough servers are kept on to handle the entire demand.

Algorithm 3: Delay violation handler algorithm

```
   Input: η, γ, β, λ, M
1  c ← 0
2  while Server h_v with deadline violations found and c < M do
3      a_k ← Service causing the violation
4      BS_i ← BS that receives the delayed requests
5      H^can ← Servers reachable within T^max
6      Sort H^can according to C^max − O in ascending order
7      h_m ← First server of H^can
8      if o_k λ_{i,k} γ_{i,k,m} ≥ C_m^max − O_m then
9          h_m ← Closest inactive server to BS_i
10         η_m ← 1
11     γ_{i,k,m} ← γ_{i,k,v}
12     γ_{i,k,v} ← 0
13     c ← c + 1
14 return γ, β, η
```

When the new routing has been calculated, the heuristic calculates β, i.e., it assigns resources to the services at each server. This is done proportionally to the total workload of each server, as indicated in (4). The resulting $\beta_{k,j}$ values are normalized at server level so that $\sum_{a_k \in} \beta_{k,j} = 1\ \forall h_j \in \mathcal{H}$. Finally, the delay constraints are checked (line 38). If violations are found, the heuristic takes corrective action using Alg. 3, which redirects requests from the violating server to other underutilized ones. This helps alleviate the load on the offending server and reduces computing time, which is usually the biggest contributor to the total latency. If this is not enough, the algorithm activates the inactive server with the shortest routing delay to the BS receiving the delayed requests. This process is repeated until all deadline violations are resolved or all servers are turned on.

LESS-ON operates in a time-slotted manner. At the beginning of each time slot, $t \in \{1, 2, \ldots, T\}$, the orchestrator executes the energy optimization heuristic (Alg. 1) using as input the maximum expected load in t, which we denote by λ^t.[6] In this way, the resource allocation is periodically adapted to match network conditions. When a server is shut down, it has to remain inactive for the time necessary to compensate for the energy consumed during the booting process. Consequently, the duration of t is determined by the boot energy E^{boot}.

5.2. Complexity analysis

The time complexity of Alg. 1 depends on the number of edge servers in $\mathcal{H}$, which must be traversed twice (lines 4 and 12), the number of services in $\mathcal{A}$, which is iterated once (line 8), and the number of BSs in $\mathcal{B}$, which is also traversed once (line 9). Thus, the time complexity of Alg. 1 is $\mathcal{O}(|\mathcal{H}|^2 \times |\mathcal{A}| \times |\mathcal{B}|)$. However, in realistic scenarios, the number of edge servers and BSs is significantly bigger than the number of services ($|\mathcal{B}| \gg |\mathcal{A}|$, $|\mathcal{H}| \gg |\mathcal{A}|$) and the number of edge servers is at most equal to the number of BSs ($|\mathcal{B}| \geqslant |\mathcal{H}|$). Hence, the complexity can be simplified as $\mathcal{O}(N^3)$, where N represents the number of BSs. The time complexity of Alg. 2 depends on the number of BSs, N, the number of services in $\mathcal{A}$ and the number of servers in $\mathcal{H}$ and is given by $\mathcal{O}(|\mathcal{H}| \times |\mathcal{A}| \times |\mathcal{B}|)$ which can be simplified to $\mathcal{O}(N^2)$ following the same reasoning for Alg. 1. Finally, for Alg. 3, to avoid infinite iterations when a solution that satisfies all deadlines cannot be found, we implement a configurable limit of M iterations. Therefore, its time complexity is $\mathcal{O}(1)$.

[6] Load prediction models fall outside the scope of this work. We focus on evaluating the energy-saving potential of LESS-ON. Hence, for the purpose of this study, we assume an ideal predictor.

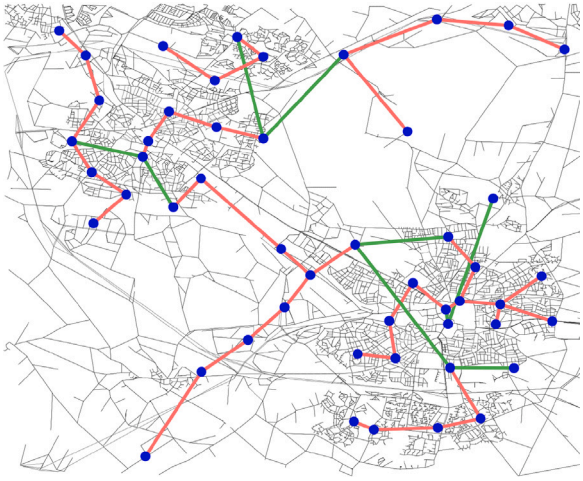


Fig. 3. Map of the simulated scenario, showing the cities of Enschede and Hengelo in the Netherlands, the location of the BSs of a real MNO, the known backhaul links in green and the added ones in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Infrastructure parameters. Reference values from [39,40].

Servers	Type 1	Type 2	Type 3
Max. Capacity (C^{max}, ops/s)	22 151 384	11 260 532	33244766
Idle Consumption (E^{idle})	415 W	222 W	541 W
Max. Consumption (E^{max})	1280 W	696 W	2336 W
Links			
Link Capacity (α)	10 Gbps		
Link energy consumption (σ)	5.9 nJ/bit		

5.3. Practical implementation challenges

In a 5G network that integrates an edge computing deployment, the edge servers have a crucial role in routing user plane traffic. Consequently, server shutdown, by default, would render communication impossible. However, in such deployments, the core network can redirect user plane routing decisions to alternative edge servers [38]. Through control plane interactions with the 5G Core Network (CN), the orchestrator can influence data plane decisions in the RAN. After obtaining the new configuration from the heuristic, the orchestrator shuts down specific edge servers accordingly. However, before the shutdown, the data plane traffic needs to be reconfigured, and its routing rules and functions migrated to the closest edge server (we remind the reader that BSs are connected between them, and they can access each other edge servers). The orchestrator is also responsible for the new configuration of the routing of computing requests toward other edge servers.

Note that another point to consider before shutting down an edge server is the presence of computing tasks already in execution at an edge server that is to be shut down. The server shutdown happens when the server is actually idle, meaning that all computing tasks have been completed. This could impact the actual value of the energy consumption as the server remains on until all tasks have finished, but as we consider computing requests that take fractions of a second to be completed, this impact on the actual value of the energy consumption is negligible.

6. Performance evaluation

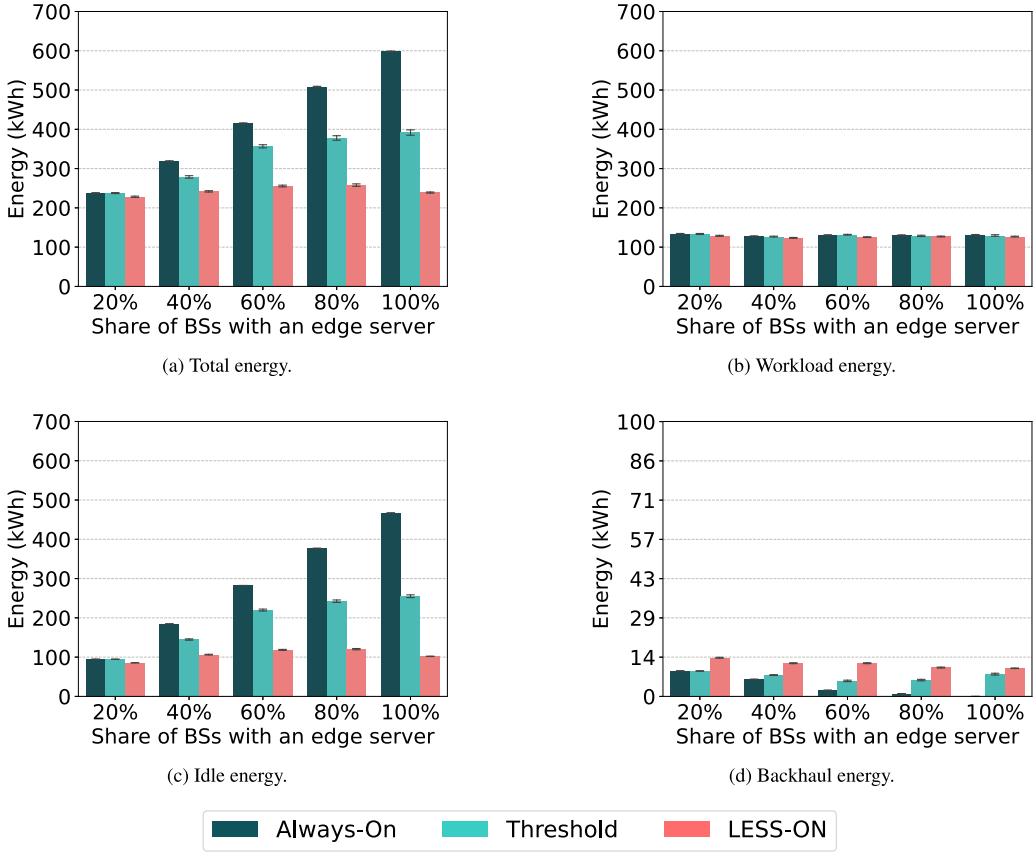
We evaluate the performance of LESS-ON using MintEDGE [15], our in-house Python simulator. By leveraging actual data from the infrastructure of an MNO in the Netherlands, we evaluate the potential of LESS-ON in comparison with two baselines via simulations, and we address the following questions:

(i) How much energy does LESS-ON save compared with two baseline methods, namely *Always-on* (where edge servers are constantly active) and *Threshold*, where servers sleep when their load is below a 10% threshold (set as the optimal value in [13])?
(ii) How does LESS-ON perform in meeting application deadlines, and how does it compare with the other strategies?
(iii) How does the number of deployed edge servers and the utilization levels impact the energy-saving capacity of LESS-ON and the other two strategies for different application requirements?
(iv) How significant is the impact of the backhaul energy consumption on total energy consumption?

6.1. Scenario and parameters

The evaluation is based on real data from an MNO's infrastructure in Enschede and Hengelo (The Netherlands) [45]. The infrastructure consists of 50 BSs, located across the municipalities as shown in Fig. 3. The BSs are connected via the X2 interface with 10 Gbps fiber optics links using Cisco ASR9010 Routers, which consume 5.9 nJ/bit [40]. The wireless backhaul connectivity is partially obtained from [45], shown in green in Fig. 3. To ensure connectivity for all BSs and also represent fiber backhaul links, we introduce additional links shown in red in Fig. 3. Since the MNO can choose not to deploy edge servers at all BSs, we evaluate various server placements, ranging from 20% to 100% of the BSs hosting an edge server, with increments of 20%. This allows us to evaluate the behavior of LESS-ON and the two baselines with different utilization levels in the computing infrastructure. In order to decide which BSs host an edge server, we consider the physical location of the BSs and their connectivity with other BSs. In particular, we define two metrics: the degree centrality (d_d), which takes into account the number of connections to other BSs, and the location-based centrality (d_l), which considers the physical distance to the closest connected neighbors. Because we are looking for a trade-off between them to place servers at the BSs with more connections but at the same time spread them across the geography, we finally calculate the global centrality as $d = d_d/d_l$. Then, the BSs with the biggest d centrality are chosen to host an edge server. We consider three types of servers: Type 1 represents an HP ProLiant DL560 Gen11 with an Intel Xeon Platinum 8490H at 1.90 GHz; Type 2 is an HP ProLiant DL380a Gen11 with an Intel Xeon Platinum 8480+ at 2.0 GHz; and Type 3 is a Fujitsu Server Primenergy CX2560 M7 with an Intel Xeon Gold 6428N at 1.8 GHz. This selection aims to represent the heterogeneous nature of an edge architecture by selecting three types of servers with different capacities and energy efficiencies. Their energy consumption data is obtained from [39], where the three edge server types' full hardware and software configuration can be found. All hardware-specific parameters are summarized in Table 2. The energy per operation, E_m, can be obtained from E^{max} and E_m^{idle} as $E_m = (E_m^{max} - E_m^{idle})/C_m^{max}$. In addition, we assume a T_m^s of 20 s and a time slot of 10 min. We also assume that during those 20 s, the booting power P_m^s is the maximum consumption of the servers.

We evaluate a subset of three services with different latencies and arrival rates from the service scenarios defined by ETSI [20] to represent service diversity. The three selected services, listed in Table 3, are: video analytics (referred to as SVA), augmented reality (referred to as SAR), and connected vehicles (referred to as SCV). Request arrival for all services follows a Poisson distribution with an expected arrival rate λ_k. The user count for each service depends on the area's population and the service's market penetration. We have assumed a 1% market penetration for SAR, 10% for SCV, and 15 users (CCTV cameras) per km^2 for SVA. To represent service diversity and different processing demands, each selected service has a different processing density, i.e., they need different amounts of operations to process a single data unit: SAR represents a high processing density service, SCV is a low processing density service, and SVA falls in between.



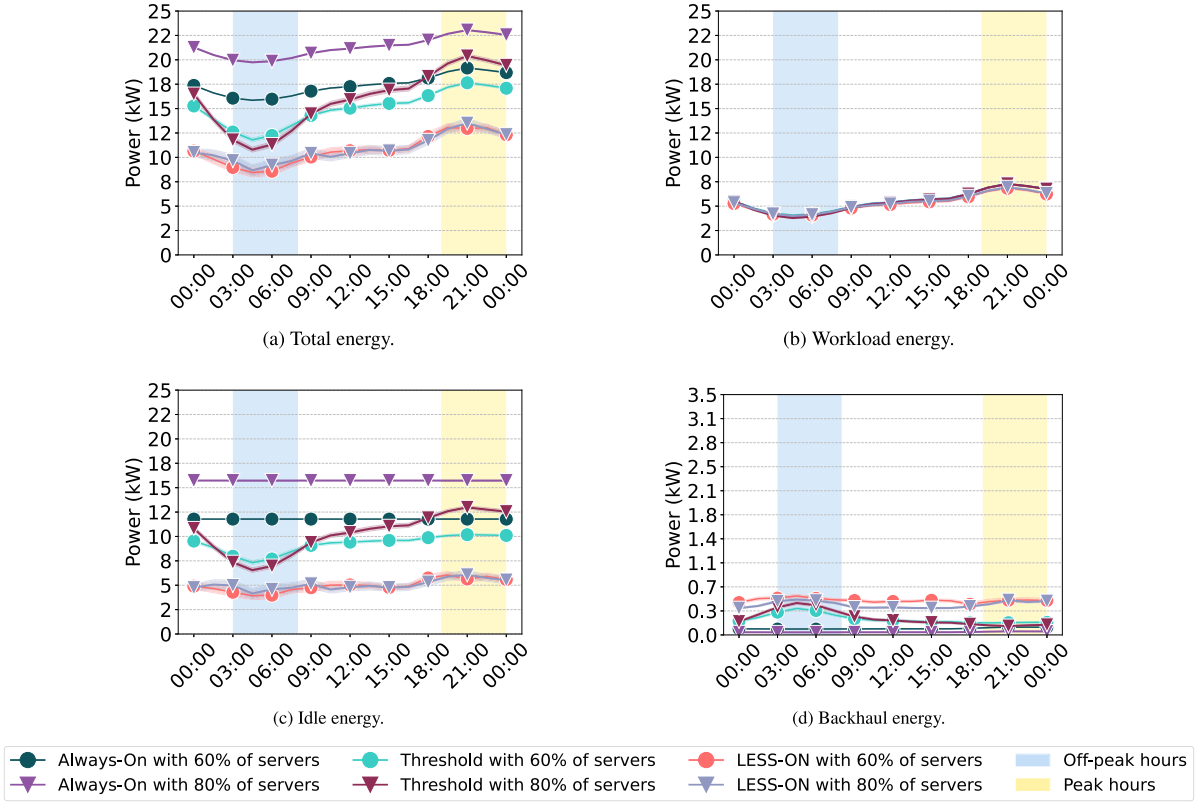
(a) Total energy.

(b) Workload energy.

(c) Idle energy.

(d) Backhaul energy.

Fig. 4. Accumulated total energy consumption, energy consumed by the servers, and energy consumed by the backhaul in kWh after the 24-hour period.



(a) Total energy.

(b) Workload energy.

(c) Idle energy.

(d) Backhaul energy.

Fig. 5. Breakdown of the energy consumption over the 24-hour period for the scenarios with 60 and 80% of BSs with a co-located edge server. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Types of services studied[20,41–44].

Service type	Examples of applications	T_k^{max}	λ_k (req/s)	V_k^{in}	V_k^{out}	o_k (ops)	Max. users
Connected Vehicles (SCV)	Hazard alert, traffic congestion, parking location	5 ms	10	1600 B	100 B	7000	4970
Augmented Reality (SAR)	Show extra information on real-time camera footage	15 ms	0.5	1500 kB	25 kB	50 000	876
Video Analytics (SVA)	Pattern recognition, CCTV	30 ms	6	1500 kB	20 B	30 000	3000

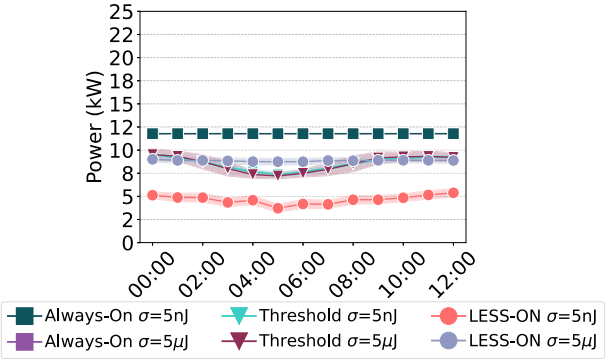


Fig. 6. Idle energy consumption for the three approaches with two different $\sigma_{i,j}$ values: $\sigma_{i,j}$ = 5.9 nJ and $\sigma_{i,j}$ = 5.9 μJ.

We simulate a 24-hour period in which the number of active users varies from 2% to 16% of the maximum number of users following the time-traffic distribution in [42]. After completing the 24-hour simulation period, the user count cycle repeats itself, mirroring the typical weekly pattern. Consequently, extending the simulation duration beyond 24 h would provide no additional insights. Users are generated in a random geographical location and move randomly across the map shown in Fig. 3.

We compare LESS-ON against two baselines:

(i) *Always-On:* In this case, all edge servers are active and requests are routed in a greedy manner in the same way as Alg. 2 to the local edge server or to the closest one for BSs without an edge server. If the closest edge server cannot meet the demand of a BS, it routes as many requests as possible to the closest and the remaining to the second closest.

(ii) *Threshold:* This approach also starts placing services in a greedy manner following Alg. 2. After that, it shuts down the servers whose load is below 10%. Then, it routes the requests from the servers that have been shut down to the closest server in the same greedy manner.

6.2. Results discussion

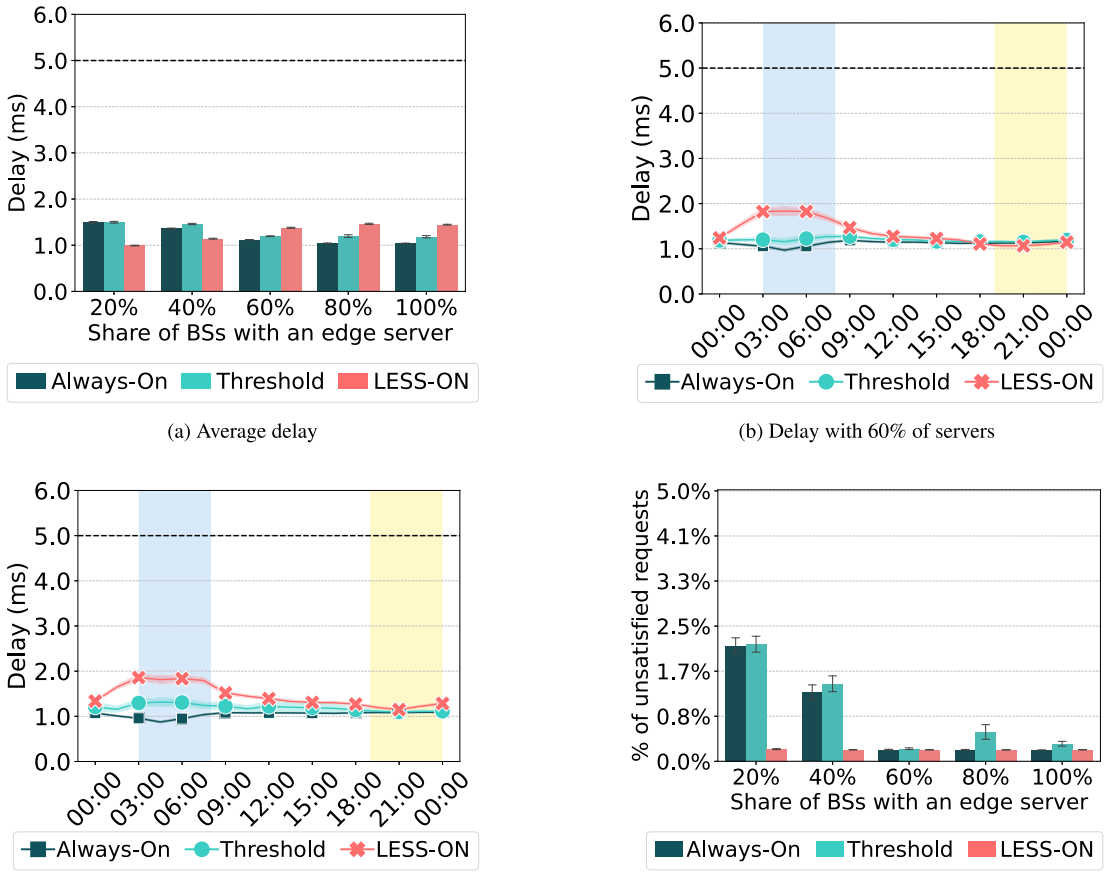
6.2.1. Energy consumption

Let us first assess the energy consumption difference between LESS-ON and the baselines with increasing fractions of edge servers deployed in the network. Fig. 4 shows a breakdown of the accumulated energy consumption over 24 h. As expected, LESS-ON provides the highest benefit under denser edge server deployment scenarios. For example, when all BSs have an edge server (100% on the x-axis), LESS-ON reduces the energy consumption by 60% (with respect to Always-On) and by 39% (with respect to Threshold), as shown in Fig. 4(a). However, with only 20% of the BSs having an edge server, the energy savings decrease to just 4% compared with the other two approaches, as 10 servers do not suffice to attend to all the demand within the delay requirements, and there is not much potential for server shutdown. On average, LESS-ON achieves a 35% reduction in energy consumption compared with Always-On and a 23% reduction compared with Threshold. This reduction is mainly due to the reduction in idle energy consumption, which accounts for 59% of the total energy, as shown in Fig. 4(c). LESS-ON makes more intensive use of the backhaul links, increasing the backhaul energy consumption, as shown in Fig. 4(d). However, since backhaul energy only represents 2% of the total energy, this increment is negligible.

Now, let us take a closer look at the change in energy consumption over a 24-hour period. For this analysis, we select the two most representative deployments, as indicated by the request satisfaction ratio. Looking at Figs. 7(d), 8(d) and 9(d), which show the ratio of unsatisfied requests for all the deployments, we select the scenarios with 60% and 80% as the most appropriate deployments for the demand being evaluated. The behavior over time of the chosen deployments for the three strategies is shown in Fig. 5. Let us focus on peak hours (highlighted in yellow) and off-peak hours (marked blue). In Fig. 5(a), during peak hours (from 19:00 to 00:00), LESS-ON achieves a reduction of 31% with respect to Always-On and a reduction of 25% with respect to Threshold when 60% of the BSs host an edge server. When 80% of the BSs have an edge server, these reductions increase to 43% and 34%, respectively. Shifting our focus to off-peak hours (from 03:00 to 08:00), LESS-ON reduces energy consumption by 46% with respect to Always-On and by 29% with respect to Threshold in the deployment with 60% of servers. In the deployment with 80% of servers, these reductions become 54% and 19%, respectively. The energy savings are primarily due to reduced idle energy consumption, as shown in Fig. 5(c). Threshold only reacts to servers with very low loads, which makes its reduction more prominent during off-peak hours. In contrast, LESS-ON studies the possibility of shutting down every server regardless of its load, which results in energy savings across the whole day. Moreover, LESS-ON shuts down the servers with the highest E^{idle} first. In this scenario, Type 3 servers have both the E^{idle} and the highest consumption per operation E, contributing to a small decrease in the workload energy consumption, as shown in Fig. 5(b). In contrast, Fig. 5(d) shows increased backhaul energy consumption, mainly during off-peak hours. In this period, with more servers inactive, the backhaul links are utilized more intensively to route requests to active servers.

The deployments with 60% and 80% of BSs hosting an edge server exhibit similar trends. The energy consumption in the denser deployment does not increase when using LESS-ON as it keeps the new servers off, maintaining similar idle energy consumption. Always-On has an increment equal to the idle consumption of the new servers, and Threshold experiences this increment in consumption only during peak hours while slightly reducing the energy consumption in comparison with the 60% deployment in off-peak hours.

In scenarios with a more significant backhaul consumption, e.g., links with a higher σ, LESS-ON would strike a balance between server shutdowns and backhaul utilization, as illustrated in Fig. 6, which shows a comparison between the per bit consumption of the backhaul links of the selected infrastructure (σ = 5.9 nJ) and a bigger one (σ = 5.9 μJ), selected to show the behavior of the algorithms in a scenario with a high energy consumption backhaul. We focus on the first half of the day to better show this behavior, as edge servers are shut down during off-peak hours. With the new σ, LESS-ON shuts down fewer edge servers, as making more intensive use of the backhaul outweighs the savings of shutting down edge servers. In this case, Threshold, which does not look at the backhaul energy consumption, keeps shutting down edge servers in the same way as it did with the original σ, leading to an increase of 29.61% in the total energy consumption with respect to LESS-ON.



(a) Average delay

(b) Delay with 60% of servers

(c) Delay with 80% of servers

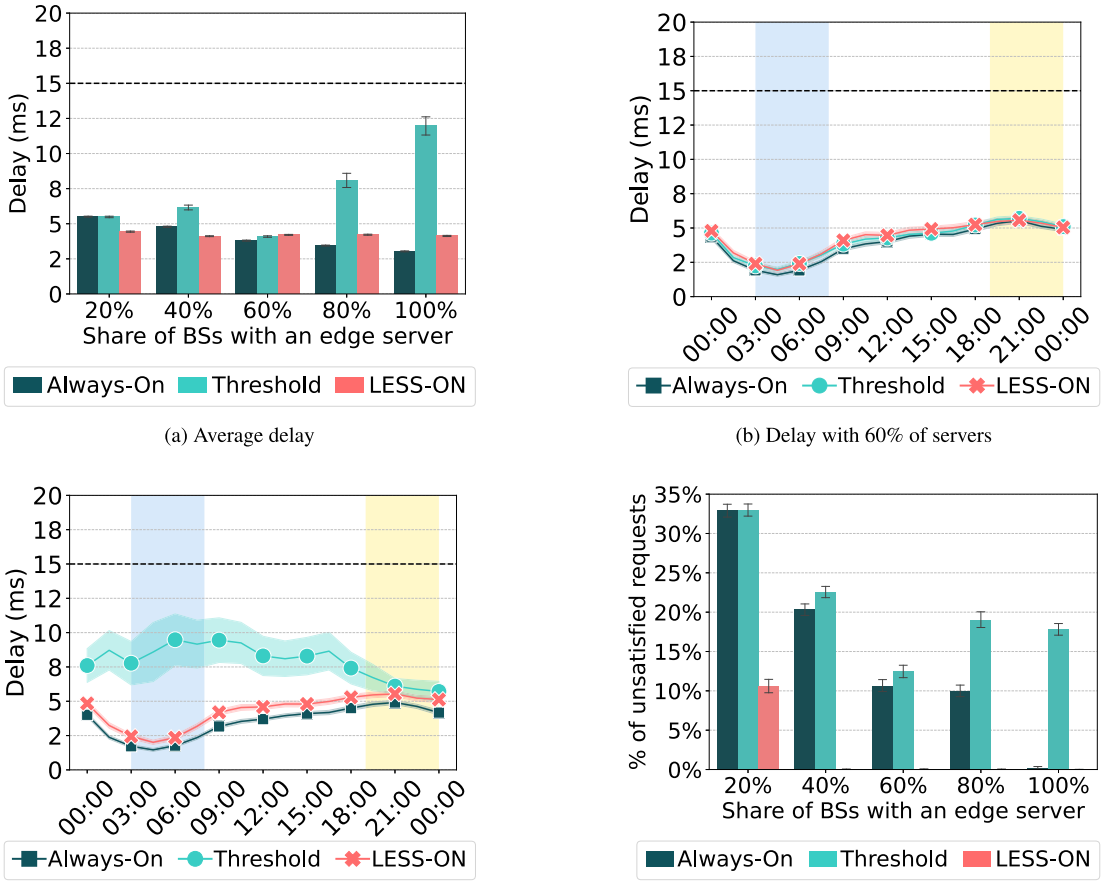
(d) Share of unsatisfied deadlines

Fig. 7. Performance of the *connected vehicles* service with a delay budget of 5 ms: (a) average delay under increasing server density, (b) delay over time for the scenarios of 60%, and (c) 80% of BSs having a co-located edge server, and (d) share of unsatisfied deadlines under increasing server density. Peak and off-peak hours are shown with yellow and blue backgrounds, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.2.2. Delays and deadline satisfaction

Let us assess whether, and to what extent, LESS-ON degrades the delay performance of the requests and consequently the satisfaction ratio. We report the results for each service, namely SVA, SAR, and SCV, in Figs. 7, 8 and 9, respectively.

In Figs. 7(a), 8(a), and 9(a), we observe that all schemes can maintain an average delay below the respective delay budgets. We can distinguish two behaviors, one where computing resources are scarce and another where computing resources are correctly scaled or even overprovisioned. The first case happens in the deployments with 20% and 40% of BSs hosting an edge server. In both of them, LESS-ON's Alg. 3 effectively handles deadline violations when computing resources are saturated. LESS-ON redirects requests from the violating server to other servers that can fulfill the constraints of the request. If it cannot find any, it turns on a new server and redirects the requests there. This allows these requests to have more computing resources available for them, reducing the computing time, which tends to be the biggest contributor to the total delay. Moreover, it reduces the load on the server that caused the violation. In practice, the effect of this is a reduction in the average delay for SCV and SAR, as shown in Figs. 7(a) and 8(a). These two services are prioritized by the resource allocation of LESS-ON as they have tighter delay constraints. In the case of SVA, LESS-ON experiences a slight increase in average delay (Fig. 9(a)) as a result of the order in which LESS-ON assigns resources to requests, leaving the services with more flexible delay budgets for the end of the assignation, giving SVA a lower priority. We can observe the second of the two aforementioned behaviors in the deployments with 60%, 80%, and 100% of BSs hosting an edge server. In this case, LESS-ON experiences a minor increase in average delay for all three services, not only SVA, as shown in Figs. 7(a), 8(a) and 9(a). This is mainly due to the inherent delay in the backhaul when reaching active servers and the fact that it can carry out more shutdowns (since there are more idle resources), increasing also computing time with respect to the other two approaches. Figs. 7(b) and 7(c) illustrate this behavior, where the delay with LESS-ON increases as more servers are shut down during the off-peak hours (03:00 to 08:00, shown in blue). The Threshold approach also shows a similar effect, albeit to a lesser extent due to fewer server shutdowns. Despite this minor increase in average delay in the chosen deployments (60% and 80% of BSs having an edge server), LESS-ON prevents an increase in deadline violations, as shown in Figs. 7(d), 8(d) and 9(d). In contrast, Always-On and Threshold, which prioritize the nearest server without considering its load, lead to a higher percentage of missed deadlines compared with LESS-ON. This increase in the share of missed deadlines for Always-On and Threshold is more pronounced in deployments with limited computing resources (20% and 40% of BSs having an edge server), as shown in Figs. 7(d) and 8(d) for the SCV and SAR services, respectively. Threshold faces significant challenges in meeting the tight delay budget of SAR, resulting in a high number of missed deadlines across all deployments, as shown in Fig. 8(d). While the performance of Always-On initially improves as more servers are deployed, Threshold experiences an increase in missed deadlines for deployments with 80% and 100% of servers. As the number of deployed servers increases, the load is naturally spread across more servers, leaving more servers below the 10% threshold. Threshold then routes the requests to the closest server, regardless of its load, resulting in a higher share of unsatisfied deadlines. As shown



(a) Average delay

(b) Delay with 60% of servers

(c) Delay with 80% of servers

(d) Share of unsatisfied deadlines

Fig. 8. Performance of the *augmented reality* service with a delay budget of 15 ms: (a) average delay under increasing server density, (b) delay over time for the scenarios of 60%, and (c) 80% of BSs having a co-located edge server, and (d) share of unsatisfied deadlines under increasing server density. Peak and off-peak hours are shown with yellow and blue backgrounds, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

in Fig. 9(d), LESS-ON causes an increase in the number of unsatisfied deadlines for SVA. This is a result of the order in which LESS-ON assigns resources, prioritizing the services with the shortest deadline. However, on average, LESS-ON achieves a 95% reduction in unsatisfied deadlines.

6.2.3. Heuristic performance

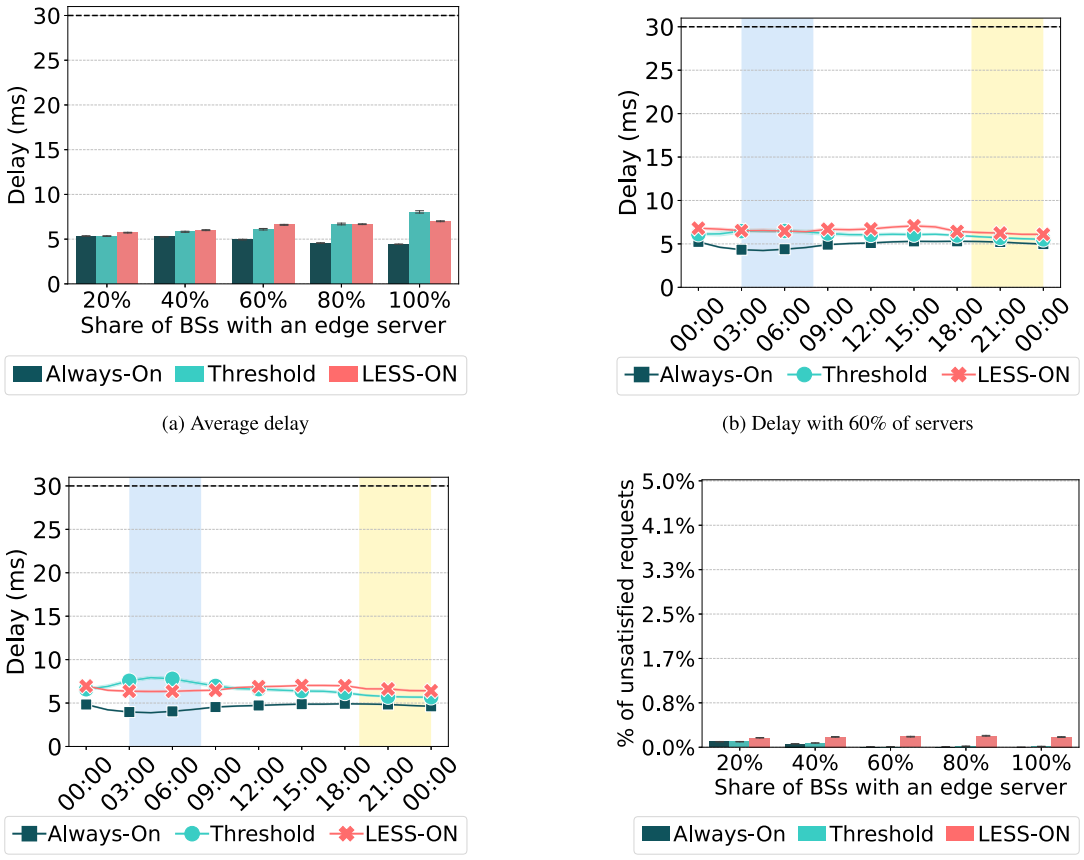
We use Gurobi [46] optimization library for Python to find the solution to ($\mathcal{P}1$). Table 4 lists the execution times for different problem sizes. The elevated execution times required to obtain the solution using Gurobi, which increase exponentially with the size of the network, confirm that solving this problem optimally for real-time decision-making is impractical in realistic scenarios. In all the scenarios where an optimal solution could be obtained within 48 h, the result for ($\mathcal{P}1$) obtained by LESS-ON's heuristic is only 0.04% over the optimal solution given by Gurobi. Given this small difference, the energy used to obtain the optimal solution using Gurobi outweighs the minimal savings achieved compared to the heuristic approach. The average execution time of the heuristic is only 1 s, showing its efficiency. Notably, all the execution times of the heuristic are within a realistic slot duration t.

Table 4
Performance of LESS-ON's heuristic against the optimal solution obtained by Gurobi within a 48-hour limit with different network sizes and 3 offered services ($|\mathcal{A}| = 3$).

BSs	Solution	Time (s)	Power (W)
10	Gurobi	3.043	3341.26
	LESS-ON	0.046	3342.85
15	Gurobi	37.152	4939.47
	LESS-ON	0.161	4941.20
20	Gurobi	936.140	6527.89
	LESS-ON	0.541	6530.21
25	Gurobi	51 891.282	8067.25
	LESS-ON	1.417	8071.20
30	Gurobi	–	–
	LESS-ON	2.947	9614.24

7. Conclusions

In this work, we have introduced LESS-ON, an energy-saving approach with polynomial time complexity for edge computing infrastructures that strategically shuts down idle edge servers during periods of low demand, effectively reducing the energy consumption of the infrastructure while meeting the delay requirements of the applications. By striking a balance between the energy consumed by idle servers and the energy used for backhaul routing, LESS-ON achieves significant energy savings of 35% compared with the commonly implemented always-on approach and 23% with respect to a threshold-based approach while keeping delays within the application's requirements and reducing the share of delay violations. LESS-ON offers a sustainable and cost-effective solution for the efficient deployment of edge computing



(a) Average delay

(b) Delay with 60% of servers

(c) Delay with 80% of servers

(d) Share of unsatisfied deadlines

Fig. 9. Performance of the *video analytics* service with a delay budget of 30 ms: (a) average delay under increasing server density, (b) delay over time for the scenarios of 60%, and (c) 80% of BSs having a co-located edge server, and (d) share of unsatisfied deadlines under increasing server density. Peak and off-peak hours are shown with yellow and blue backgrounds, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

services. Future work will focus on investigating the use of workload prediction models to enable proactive changes to server states and further improve energy efficiency, and the study of the impact of inaccurate predictions on QoS levels and energy consumption. Moreover, we plan to investigate the influence of different boot-up times on energy consumption.

CRediT authorship contribution statement

Blas Gómez: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Formal analysis, Conceptualization. **Suzan Bayhan:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Formal analysis, Conceptualization. **Estefanía Coronado:** Writing – review & editing, Supervision, Resources, Funding acquisition, Formal analysis, Conceptualization. **José Villalón:** Writing – review & editing, Supervision, Resources, Funding acquisition, Conceptualization. **Antonio Garrido:** Supervision, Resources, Funding acquisition.

Declaration of competing interest

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

Data availability

All data generated or analyzed during this study are included in this article.

Acknowledgments

Grants PID2022-142332OA-I00 and PID2021-123627OB-C52 funded by MICIU/AEI/10.13039/501100011033 and the EU, ERDF. This work is also funded by the European Social Fund and UCLM under grant 2019-PREDUCLM-10921, the Government of Castilla-La Mancha under project SBPLY/21/180501/000195 and Universidad de Castilla-La Mancha under project 2023-GRIN-34056. This work is also supported by the EU "NextGenerationEU/PRTR", MCIN, and AEI under project IJC2020-043058-I and the EU's H2020 XGain project (GA No 101060294). The authors from UT acknowledge the support of the Faculty of EEMCS under the research grant EERI: Energy-Efficient and Resilient Internet. Blas Gómez thanks UCLM's Vice-rectorate of Science Policy for the mobility grant.

References

[1] S. Redana, O. Bulakci, C. Mannweiler, L. Gallo, A. Kousaridas, D. Navrátil, A. Tzanakaki, J. Gutiérrez, H. Karl, P. Hasselmeyer, A. Gavras, S. Parker, E. Mutafungwa, 5G PPP Architecture Working Group - View on 5G architecture, 2019, Version 3.0.

[2] B. Ramprasad, A. da Silva Veith, M. Gabel, E. de Lara, Sustainable computing on the edge: A system dynamics perspective, in: Proc. of ACM HotMobile, 2021, http://dx.doi.org/10.1145/3446382.3448607.
[3] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, L. Thamsen, Let's wait awhile: how temporal workload shifting can reduce carbon emissions in the cloud, in: Proc. of ACM Middleware Conference, Québec, Canada, 2021, http://dx.doi.org/10.1145/3464298.3493399.
[4] Z. Cao, X. Zhou, H. Hu, Z. Wang, Y. Wen, Toward a Systematic Survey for Carbon Neutral Data Centers, IEEE Commun. Surv. Tuts. 24 (2) (2022) 895–936, http://dx.doi.org/10.1109/COMST.2022.3161275.
[5] R. Jacob, L. Vanbever, The internet of tomorrow must sleep more and grow old, in: Proc. of HotCarbon, 2022.
[6] U. Wajid, C. Cappiello, P. Plebani, B. Pernici, N. Mehandjiev, M. Vitali, M. Gienger, K. Kavoussanakis, D. Margery, D.G. Perez, P. Sampaio, On Achieving Energy Efficiency and Reducing CO2 Footprint in Cloud Computing, IEEE Trans. Cloud Comput. 4 (2) (2016) 138–151, http://dx.doi.org/10.1109/TCC.2015.2453988.
[7] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, A.V. Vasilakos, Cloud Computing: Survey on Energy Efficiency, ACM Comput. Surv. 47 (2) (2014) 1–36, http://dx.doi.org/10.1145/2656204.
[8] I. Raïs, A.-C. Orgerie, M. Quinson, Impact of Shutdown Techniques for Energy-Efficient Cloud Data centers, in: J. Carretero, J. Garcia-Blas, R.K. Ko, P. Mueller, K. Nakano (Eds.), Proc. Springer ICA3PP, Vol. 10048, Granada, Spain, 2016, http://dx.doi.org/10.1007/978-3-319-49583-5_15.
[9] R. Buyya, A. Beloglazov, J. Abawajy, Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open challenges, in: Proc. of PDPTA, Las Vegas, USA, 2010.
[10] Y.C. Lee, A.Y. Zomaya, Energy efficient utilization of resources in cloud computing systems, J. Supercomput. 60 (2) (2012) 268–280, http://dx.doi.org/10.1007/s11227-010-0421-3.
[11] D. Meisner, B.T. Gold, T.F. Wenisch, PowerNap: Eliminating Server Idle power, in: Proc. of ACM ASPLOS, Washington, DC, USA, 2009, http://dx.doi.org/10.1145/2528521.1508269.
[12] ETSI, MEC 003 - V2.2.1 - Multi-Access Edge Computing (MEC); Framework and Reference Architecture, vol. 1, 2020, pp. 1–21.
[13] S. Wang, X. Zhang, Z. Yan, W. Wenbo, Cooperative Edge Computing With Sleep Control Under Nonuniform Traffic in Mobile Edge Networks, IEEE Internet of Things J. (3) (2019) 4295–4306, http://dx.doi.org/10.1109/JIOT.2018.2875939.
[14] B. Wu, J. Zeng, S. Shao, W. Ni, Y. Tang, New Game-Theoretic Approach to Decentralized Path Selection and Sleep Scheduling for Mobile Edge Computing, IEEE Trans. Wirel. Commun. 21 (8) (2022) 6125–6140, http://dx.doi.org/10.1109/TWC.2022.3146514.
[15] B. Gómez, S. Bayhan, E. Coronado, J. Villalón, A. Garrido, MintEDGE: Multi-tier simulator for energy-aware strategies in edge computing, in: Proc. of the ACM Annual International Conference on Mobile Computing and Networking, MobiCom, Madrid, Spain, 2023, http://dx.doi.org/10.1145/3570361.3615727.
[16] Z. Ali, S. Khaf, Z.H. Abbas, G. Abbas, F. Muhammad, S. Kim, A Deep Learning Approach for Mobility-Aware and Energy-Efficient Resource Allocation in MEC, IEEE Access 8 (2020) 179530–179546, http://dx.doi.org/10.1109/ACCESS.2020.3028240.
[17] T. Dlamini, A.F. Gambín, Adaptive Resource Management for a Virtualized Computing Platform within Edge computing, in: Proc. of IEEE SECON, 2019, http://dx.doi.org/10.1109/SAHCN.2019.8824927.
[18] M. Zakarya, L. Gillam, H. Ali, I. Rahman, K. Salah, R. Khan, O. Rana, R. Buyya, epcAware: A Game-based, Energy, Performance and Cost Efficient Resource Management Technique for Multi-access Edge Computing, IEEE Trans. Serv. Comput. (2020) http://dx.doi.org/10.1109/TSC.2020.3005347.
[19] J. Ahn, J. Lee, S. Park, H.-S. Park, Power Efficient Clustering Scheme for 5G Mobile Edge Computing Environment, Mob. Netw. Appl. 24 (2) (2019) 643–652, http://dx.doi.org/10.1007/s11036-018-1164-2.
[20] ETSI, ETSI GS MEC-IEG 004 V1.1.1 (2015-11) - Mobile-edge computing (MEC); service scenarios, 2015.
[21] A.A. Amer, I.E. Talkhan, T. Ismail, Optimal Power Consumption on Distributed Edge Services Under Non-Uniform Traffic with Dual Threshold Sleep/Active control, in: Proc. of IEEE NILES, Giza, Egypt, 2021, http://dx.doi.org/10.1109/NILES53778.2021.9600496.
[22] C.N.L. Tan, C. Klein, E. Elmroth, Location-aware load prediction in Edge Data Centers, in: Proc. of IEEE FMEC, Valencia, Spain, 2017, http://dx.doi.org/10.1109/FMEC.2017.7946403.
[23] F. Han, S. Zhao, L. Zhang, J. Wu, Survey of Strategies for Switching Off Base Stations in Heterogeneous Networks for Greener 5G Systems, IEEE Access 4 (2016) 4959–4973, http://dx.doi.org/10.1109/ACCESS.2016.2598813.
[24] H. Cheng, B. Liu, W. Lin, Z. Ma, K. Li, C.-H. Hsu, A survey of energy-saving technologies in cloud data centers, J. Supercomput. 77 (11) (2021) 13385–13420, http://dx.doi.org/10.1007/s11227-021-03805-5.
[25] H. Fourati, R. Maaloul, L. Fourati, M. Jmaiel, An Efficient Energy-Saving Scheme Using Genetic Algorithm for 5G Heterogeneous Networks, IEEE Syst. J. 17 (1) (2023) 589–600, http://dx.doi.org/10.1109/JSYST.2022.3166228.
[26] M.J. Daas, M. Jubran, M. Hussein, Energy Management Framework for 5G Ultra-Dense Networks Using Graph Theory, IEEE Access 7 (2019) 175313–175323, http://dx.doi.org/10.1109/ACCESS.2019.2957378.
[27] H. Pervaiz, O. Onireti, A. Mohamed, M. Ali Imran, R. Tafazolli, Q. Ni, Energy-Efficient and Load-Proportional eNodeB for 5G User-Centric Networks: A Multilevel Sleep Strategy Mechanism, IEEE Veh. Technol. Mag. 13 (4) (2018) 51–59, http://dx.doi.org/10.1109/MVT.2018.2871740.
[28] H. Çelebi, Y. Yapıcı, I. Güvenç, H. Schulzrinne, Load-Based On/Off Scheduling for Energy-Efficient Delay-Tolerant 5G Networks, IEEE Trans. Green Commun. Netw. 3 (4) (2019) 955–970, http://dx.doi.org/10.1109/TGCN.2019.2931700.
[29] L. Mao, R. Chen, H. Cheng, W. Lin, B. Liu, J.Z. Wang, A resource scheduling method for cloud data centers based on thermal management, J. Cloud Comput. 12 (1) (2023) 84, http://dx.doi.org/10.1186/s13677-023-00462-2.
[30] C.-C. Lin, P. Liu, J.-J. Wu, Energy-aware virtual machine dynamic provision and scheduling for cloud computing, in: 2011 IEEE 4th International Conference on Cloud Computing, 2011, pp. 736–737, http://dx.doi.org/10.1109/CLOUD.2011.94.
[31] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, R.P. Liu, Energy-efficient admission of delay-sensitive tasks for mobile edge computing, IEEE Trans. Commun. 66 (6) (2018) 2603–2616, http://dx.doi.org/10.1109/TCOMM.2018.2799937.
[32] J. Yan, S. Bi, Y.J. Zhang, M. Tao, Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency, IEEE Trans. Wirel. Commun. 19 (1) (2019) 235–250, http://dx.doi.org/10.1109/TWC.2019.2943563.
[33] P. Wang, Z. Zheng, B. Di, L. Song, HetMEC: latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing, IEEE Trans. Wirel. Commun. 18 (10) (2019) 4942–4956, http://dx.doi.org/10.1109/TWC.2019.2931315.
[34] P. Wiesner, L. Thamsen, LEAF: Simulating large energy-aware fog computing environments, in: Proc. of IEEE ICFEC, Melbourne, Australia, 2021, http://dx.doi.org/10.1109/ICFEC51620.2021.00012.
[35] A. Gandhi, M. Harchol-Balter, I. Adan, Server farms with setup costs, Perform. Eval. 67 (11) (2010) 1123–1138, http://dx.doi.org/10.1016/j.peva.2010.07.004.
[36] J. Krarup, P.M. Pruzan, Selected Families of Discrete Location Problems: Part III, the Plant Location Family, Research Library, Faculty of Business, University of Calgary, 1977.
[37] S.K. Jacobsen, Heuristics for the capacitated plant location model, European J. Oper. Res. 12 (3) (1983) 253–261, http://dx.doi.org/10.1016/0377-2217(83)90195-9.
[38] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, et al., MEC in 5G networks, ETSI White Paper 28 (2018) 1–28.
[39] Standard Performance Evaluation Corporation, SPECpower results, URL https://www.spec.org/power_ssj2008/results/.
[40] Cisco power calculator, 2023, URL http://www.cisco.com/c/en/us/applicat/camp/CiscoPowerCalculator_Layout.html. (Accessed 25 June 2023).
[41] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J.J. Ramos-Munoz, J.M. Lopez-Soler, A Survey on 5G Usage Scenarios and Traffic Models, IEEE Commun. Surv. Tuts. 22 (2) (2020) 905–929, http://dx.doi.org/10.1109/COMST.2020.2971781.
[42] METIS-II mobile and wireless communications enablers for twenty–twenty information society II: Deliverable D2.3 performance evaluation results, 2020, URL https://metis-ii.5g-ppp.eu/wp-content/uploads/deliverables/METIS-II_D2.3_V1.0.pdf. (Accessed 05 June 2024).
[43] C. Sonmez, A. Ozgovde, C. Ersoy, Fuzzy Workload Orchestration for Edge Computing, IEEE Trans. Netw. Serv. Manag. 16 (2) (2019) 769–782, http://dx.doi.org/10.1109/TNSM.2019.2901346.
[44] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, Q. Li, LAVEA: latency-aware video analytics on edge computing platform, in: Proc. of ACM/IEEE SEC, New York, NY, USA, 2017, http://dx.doi.org/10.1145/3132211.3134459.
[45] Antennekaart, 2023, https://antennekaart.nl. (Accessed 15 April 2023).
[46] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual, 2023, URL https://www.gurobi.com.

Blas Gómez is a Ph.D. student at Universidad de Castilla-La Mancha since 2020. In 2019, he completed his M.Sc. in Computer and Networks Engineering from the Polytechnical University of Valencia and his B.Sc. in Computer Science in 2018 from Universidad de Castilla-La Mancha. His current research interests include the delivery of multimedia content over SDWLANs, wireless communications, MEC systems, and AI-driven network management.

Suzan Bayhan is an associate professor at the University of Twente. Previously, she was a senior researcher at TU Berlin, and a postdoctoral researcher at the University of Helsinki, where she is currently a docent in computer science. Suzan received her Ph.D. in computer engineering from Bogazici University in 2012. Her current research interests include resource management and continuum orchestration for sustainable and resilient 5G/6G networks.

Estefanía Coronado is a Senior Researcher at Fundació i2CAT (Spain) and a Juan de la Cierva Senior Researcher at the University of Castilla-La Mancha (Spain). From 2018 to 2020 she was an Expert Researcher at Fondazione Bruno Kessler (Italy). In 2018, she completed her Ph.D. at the University of Castilla-La Mancha (Spain) on multimedia ML-driven distribution over SD-WLANs. She has worked in several H2020 projects, acted as technical task leader, and as PI of a national project. She has published around 50 papers in international journals and conferences. Her current research interests include wireless communications, MEC systems, and AI-driven network management.

José Miguel Villalón is currently an Associate Ph.D. Professor at UCLM. He received an M.Sc. degree in Computer Science and a Ph.D. in Computer Engineering from the University of Castilla-La Mancha, Spain, in 2003 and 2007, respectively. He has been a Visiting Researcher at INRIA, France. His research interests include high-performance networks, wireless networks, QoS and QoE over IEEE 802.11 and WiMAX, multicast transmission, software-defined networking, video distribution, edge computing and error-resilient protocol architectures.

Antonio Garrido received the Ph.D. degree from the University of Valencia, Spain, in 1991. In 1986, he joined the Computer Systema Department at the University of Castilla-La Mancha, where he is currently a Full Professor of Computer Architecture and Technology. His research interests include wireless sensor networks, video compression, and video transmission. From the year 2000, collaborates with the National Agency for Quality Assessment and Accreditation of Spain and several regional quality agencies in Spain.