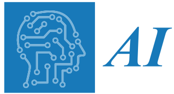 *AI*

Article

An Emulated Dynamic Framework for Evaluating Metaheuristic-Based Load Balancing Techniques in Edge Computing Networks

Daisy Nkele Molokomme [1,*], Adeiza James Onumanyi [2] and Adnan M. Abu-Mahfouz [1,2]

[1] Department of Electrical, Electronic, and Computer Engineering, University of Pretoria, Pretoria 0028, South Africa; a.abumahfouz@ieee.org
[2] Next Generation Enterprises and Institutions, Council for Scientific and Industrial Research (CSIR), Pretoria 0001, South Africa; aonumanyi@csir.co.za
* Correspondence: u11261766@tuks.co.za

Abstract

Edge computing (EC) has emerged as a paradigm to support computation-intensive Internet of Things (IoT) applications by enabling task offloading to nearby servers. Despite its potential, the inherent heterogeneity of edge resources and the dynamic, unpredictable nature of task arrivals present significant challenges for designing and evaluating effective load balancing strategies. Traditional evaluation methods are limited as follows: physical testbeds lack scalability and flexibility, while abstract simulators often oversimplify network behavior, failing to capture realistic system dynamics. To address these limitations, we present an emulated dynamic edge computing framework (EDECF) designed for evaluating load balancing schemes in EC networks. First, we developed dedicated service models for each EC node within the EDECF and implemented them using the common open research emulator (CORE) platform, thereby providing a scalable, flexible, and realistic environment for testing optimization strategies. Second, we introduced a robust fitness function that explicitly models latency, queue stability, and fairness for metaheuristic-based load balancing under dynamic edge conditions. To assess its effectiveness, this function was incorporated and tested using the following methods: the particle swarm optimization, genetic algorithm, differential evolution and simulated annealing-based load balancing algorithms. In addition, baseline methods such as the round robin and shortest queue techniques were also deployed to demonstrate the framework's capacity to facilitate rigorous analysis in heterogeneous and time-varying scenarios. Overall, results are presented to demonstrate EDECF's capability to emulate realistic workloads, capture resource variability at the edge, and support comprehensive evaluation of algorithmic performance across diverse network settings. Thus, this work aims to establish a practical and extensible foundation for researchers and practitioners to design, test, and optimize load balancing strategies in EC environments.

Keywords: computation offloading; edge computing; internet of things (IoT); load balancing; metaheuristics

Academic Editors: Xuan-Qui Pham and Miguel Angel Cazorla

Received: 9 October 2025
Revised: 9 January 2026
Accepted: 5 February 2026
Published: 1 March 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The proliferation of the Internet of Things (IoT) has led to massive data generation and an expanding class of computation-intensive, latency-sensitive applications that demand real-time responsiveness. However, most IoT devices are severely constrained in computing,

storage, and energy, rendering local execution impractical. Traditionally, such data have been offloaded to remote cloud infrastructures for processing and storage, but this approach incurs high latency due to network congestion and long transmission distances. To this end, edge computing (EC) has gained significant attention as a paradigm that brings computation and storage closer to IoT devices [1], thereby reducing response time and improving service quality.

Consequently, a wide range of offloading and load balancing schemes have been proposed to further optimize latency, energy efficiency, and related trade-offs [2–7]. For example, ref. [6] proposed a decentralized ant colony optimization (ACO)-based framework for vehicular networks under high mobility and dynamic connectivity. While effective in that domain, such assumptions diverge from static or heterogeneous resource-constrained IoT (RCIoT) environments, where mobility is limited and resources are scarce. Furthermore, most of these studies remain largely theoretical, nevertheless, some efforts have progressed toward near-realistic deployment solutions for EC systems [8–11]. For instance, ref. [8] introduced the COSMOS framework, an orchestration platform for computation offloading in edge clouds that integrates workload prediction, load balancing, and admission control, validated on a 5G testbed to demonstrate feasibility for latency-sensitive services such as object recognition. Similarly, ref. [9] proposed an artificial bee colony (ABC) algorithm for intelligent task scheduling and offloading in vehicular EC, validated through SUMO and NS-3 simulations, showing significant improvements in execution time and offloading reliability.

Despite this progress, several gaps remain in existing EC solutions. First, most evaluation platforms rely on small-scale physical testbeds or oversimplified simulators that fail to capture queue dynamics, real-time communication delays, and resource heterogeneity, limiting realism and scalability. Second, while metaheuristic optimization techniques have shown promise for handling the nondeterministic polynomial (NP)-hard nature of task offloading and load balancing, many remain theoretical and lack validation in realistic, dynamic environments. Third, although distribution communication protocols, such as Zenoh, offer inherent advantages for decentralized coordination [12], they remain largely unexplored in the context of metaheuristic-driven load balancing. Finally, existing studies often assume abundant computational resources or idealized network conditions [7], which are impractical in heterogeneous resource-constrained IoT deployments.

These limitations highlight the need for a dynamic, queue-aware, and heterogeneous edge computing (EC) framework that incorporates real-time communication to support rigorous evaluation of metaheuristic-based load balancing techniques under realistic deployment conditions. Consequently, we focus on the introduction of an emulated dynamic edge computing framework (EDECF), which enables the realistic evaluation of metaheuristic-based load balancing strategies under dynamic and heterogeneous edge conditions. By integrating the CORE network emulator with the Zenoh data-centric communication protocol, EDECF supports queue-aware task offloading, dynamic node join/leave events, heterogeneous server capacities, and controller-level decision making in a controlled yet near-realistic environment. The specific metaheuristic-based load balancers considered in this study were employed as a representative case study to demonstrate the framework's capabilities, while the framework itself remains algorithm-agnostic and extensible to other metaheuristics and scheduling strategies. This positions EDECF as a bridge between purely simulation-based evaluations and costly physical testbeds for edge computing research. Building on this foundation, our key contributions are as follows:

- We present EDECF, a queue-aware heterogeneous EC framework that provides a near-realistic emulation environment for designing and evaluating EC solutions. Our framework bridges the gap between costly, small-scale physical testbeds and the use

https://doi.org/10.3390/ai7030081

of abstract simulations by enabling scalable, flexible, and dynamic emulations under realistic network conditions.

- We propose a function that explicitly captures latency, queue stability, and fairness for metaheuristic-based load balancing in dynamic edge environments. This function was deployed in different metaheuristic-based load balancing scheme, which leverages Zenoh's brokerless, data-centric communication to manage dynamic workloads and heterogeneous resources. By integrating queue length assignment and Jain's fairness index into the formulation, the balancer demonstrates the ability to support efficient, fair, and stable workload distribution under realistic conditions.
- For realistic deployment, we developed service models for each EC node under the proposed EDECF and implemented them using the common open research emulator (CORE) platform. This emulation-based approach preserves key real-world characteristics such as heterogeneity, communication delays, and dynamic system behavior that are often overlooked in conventional simulations. Furthermore, since the developed services and codes in the emulator can be directly ported to actual computing systems, the framework substantially reduces development and deployment time.
- We evaluated EDECF by benchmarking the proposed metaheuristic-based load balancing scheme against other baseline methods such as the round robin and shortest queue techniques. This comparison enabled a comprehensive assessment of scalability, convergence speed, workload distribution efficiency, and robustness under dynamic operating conditions.

The remainder of this paper is organized as follows. Section 2 reviews related work on EC frameworks, optimization, and communication protocols. Section 3 details the design of the proposed EDECF. Section 4 presents the queue-aware optimization methodology. Section 5 describes the implementation in CORE and the experimental setup. Section 6 discusses the results and performance evaluation. Finally, Section 7 concludes the paper and outlines future research directions.

2. Related Work

Recent studies on EC systems can be broadly classified by their validation approaches as follows: simulation, physical testbeds, and emulation. Each has distinct strengths and limitations. Some of these representative EC studies are summarized in Table 1. The first category comprises simulation-based studies, which often assume homogeneous servers with abundant resources, idealized network conditions, and perfect state information [7,9,13–16]. In these studies, tasks are typically abstracted as mathematical functions, overlooking practical parameters such as construction time, packet sizes, and execution overhead. While such simplifications enable scalability analysis, they limit the realism of the evaluation. For instance, ref. [13] investigated cooperative offloading with divisible tasks, whereas [17] applied a grouped-crossover genetic algorithm (GCGA) for collaborative computing among end devices. Similarly, other works formulated offloading as mixed-integer nonlinear programming (MINLP) problems and adopted metaheuristic solutions such as differential evolution (DE) [14], grey wolf optimization (GWO) with Lévy flight [2,15], and discrete whale optimization (DWOA) [16]. In parallel, software tools like Mininet have been employed for Software-Defined Network (SDN)-enabled EC systems [18], and EmuFog has provided a container-based framework for fog/edge infrastructures [19]. Although valuable for algorithmic exploration, these approaches remain largely unvalidated under emulated or real-life deployment conditions.

The second category comprises physical testbed-based approaches, which provide more practical validation scenarios. For instance, the authors in [8,20] demonstrated scalable EC architectures for smart offloading on 5G testbeds, while IndustEdge [10] and

SAIndust [11] explored edge–cloud collaboration in Industrial IoT environments. Essentially, the attempt in [8] represents one of the few efforts to achieve real-time performance evaluation of EC systems, though its reliance on signal strength for server selection restricts applicability to small-scale settings and excludes optimization-driven load balancing. Despite the realism of physical testbeds, it should be noted that they are costly to deploy, difficult to scale, and often domain-specific. Hardware limitations, vendor dependencies, and restricted configurability further constrain their ability to reproduce large-scale, heterogeneous, and dynamic scenarios.

The third category encompasses emulation-based approaches, which aim to bridge the scalability of simulations with the realism of testbeds. However, to the best of our knowledge, this category has seen little to no exploration for studying edge computing and load balancing techniques. To fill this gap, we introduce EDECF, which is a first framework that integrates the Zenoh distributed protocol with CORE, hence enabling data-centric communication and practical evaluation of metaheuristic-based load balancing under dynamic conditions. EDECF captures system heterogeneity, network-induced delays (e.g., queuing and processing delays), and varying workloads, while preserving scalability and providing near-realistic evaluation capabilities.

The above discussion highlights that existing studies on metaheuristic-based load balancing in edge and fog computing primarily focus on evaluating algorithmic performance using discrete-event simulators or simplified system models, often assuming static network conditions, homogeneous resources, and idealized communication. While these works provide valuable insights into optimization behavior, they offer limited support for studying queue dynamics, network-level interactions, and system disruptions that commonly occur in real deployments. In contrast, the proposed EDECF in this work emphasizes emulation-driven evaluation, hence enabling the execution of unmodified control logic, realistic communication patterns, and dynamic system behavior within a reproducible environment. Unlike prior approaches that combine metaheuristics with simulation outputs, EDECF integrates network emulation (CORE), data-centric middleware (Zenoh), and edge controllers to support realistic experimentation across heterogeneous and dynamic scenarios. This distinction allows EDECF to complement existing simulation-based studies by providing a higher-fidelity evaluation platform for metaheuristic load balancing strategies in edge computing systems.

Table 1. Comparison of different edge computing validation approaches.

Validation Approach	Ref.	Architectural Design		Performance Metrics			
		Homogeneous	Heterogeneous	Latency	Solution Accuracy	Fairness	Scalability
Simulation	[7]	✗	✓	✗	✓	✗	✓
	[9]	✗	✓	✓	✓	✗	✓
	[13]	✗	✓	✓	✗	✗	✗
	[14]	✗	✓	✓	✓	✗	✓
	[15]	✗	✓	✗	✓	✗	✓
Physical Testbed	[8]	✗	✓	✓	✗	✗	✓
	[18]	✗	✓	✗	✗	✗	✓
Emulation	Ours (EDECF)	✓	✓	✓	✓	✓	✓

3. Emulated Dynamic Edge Computing Framework Architecture

In this section, we provide a concise overview of the EDECF's architecture and functional details.

3.1. EDECF System Layout

This subsection presents the overall system layout of the proposed EDECF, as illustrated in Figure 1, by first outlining its key architectural components and their roles in end-to-end task offloading. The EDECF is organized into five main components: (i) a sensor layer comprising resource-constrained IoT (RCIoT) sensors that generate, transmit, and receive task responses; (ii) base stations that provide access connectivity and aggregate sensor tasks before forwarding them toward the edge; (iii) an edge controller deployed in the edge computing center (ECC) that maintains global state (sensor and server information tables) and performs scheduling/load balancing decisions; (iv) a set of heterogeneous edge servers that execute offloaded tasks using finite compute resources and queue-based processing; and (v) a communication component based on Zenoh that enables low-latency publish/subscribe interactions and telemetry exchange among all entities. The following subsections describe each component and its functional responsibilities in supporting queue-aware, reliable task delivery and execution within the emulated edge environment.

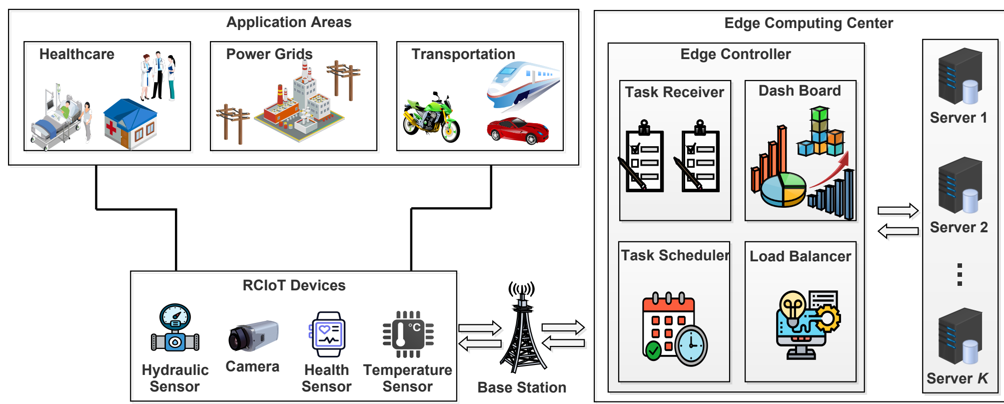

Figure 1. EDECF architecture.

3.1.1. Sensor Layer

The first component of the proposed EDECF (shown in Figure 1) is the sensor layer, which models resource-constrained IoT (RCIoT) sensors deployed in the EC environment. The set of sensors deployed in the EC network are denoted as $\mathcal{N} = \{1, 2, \ldots, N\}$, where each element represents a uniquely identified sensor instance. Each sensor is decomposed into three logical modules: the Task Generator (TG), which samples and aggregates sensor data into structured computational tasks; the Task Sender (TS), which transmits these tasks to the edge computing center (ECC) and ensures reliable delivery; and the Task Response Receiver (TRR), which receives processed results from the ECC and triggers corresponding local actions within the sensor. Each module operates over a dedicated Zenoh key to maintain modularity and avoid interference. To coordinate distributed sensors, the EDECF employs a lightweight synchronization mechanism, in which each sensor subscribes to a globally published time reference.

3.1.2. Base Station

As depicted in Figure 1, the base station (BS) forms the second core component of the EDECF, serving as an intermediary between geographically distributed RCIoT sensors and the ECC. Each sensor is associated with at least one nearby BS via a wireless interface, selected according to physical proximity and signal quality [21]. In alignment with the EDECF's discrete-time operational model, a BS performs periodic task aggregation at each time slot t, collecting tasks from its assigned sensor set $\mathcal{I}_{b_j} \subseteq \mathcal{N}$. The aggregated task batch at the BS b_j is expressed as

$$\mathcal{A}_{b_j}(t) = \bigcup_{n \in \mathcal{I}_{b_j}} \mathcal{I}_n(t), \tag{1}$$

where $\mathcal{I}n(t)$ denotes the set of tasks generated by sensor n at time t, and $\mathcal{A}b_j(t)$ represents the aggregated batch. Once aggregation is complete, each BS forwards $\mathcal{A}_{b_j}(t)$ to the ECC via a dedicated edge router, connected through a wired backbone and also provisioned with Zenoh services. The edge router serves as the network ingress point to the ECC, ensuring low-latency delivery of aggregated tasks. By leveraging Zenoh's publish/subscribe paradigm across both the BS and edge router layers, the EDECF achieves real-time, reliable, and scalable task propagation from sensors to the ECC. A detailed discussion is provided in Section 4.

3.1.3. Edge Controller

The third component of the EDECF is the edge controller, deployed within the ECC. It functions as a centralized coordination entity that manages interactions between multiple RCIoT sensors and a set of edge servers. It maintains a global network view to enable adaptive, data-driven decisions and supports key functions such as task reception, scheduling, load balancing, and dynamic offloading. To ensure load balancing and fairness, the controller can leverage metaheuristics-based load balancing techniques while maintaining two information tables: the sensor information table ($\mathcal{N}$) and the server information table ($\mathcal{M}$). Together, these structures provide real-time visibility into sensor demands and server resources, thus allowing the controller to balance workloads and minimize delay. Communication is built on Zenoh's publish/subscribe paradigm, which enables reliable task propagation and continuous monitoring of system status.

3.1.4. Edge Servers

The fourth component of the EDECF is the edge server, which is responsible for executing computation-intensive tasks offloaded by the edge controller. The set of servers deployed in the ECC are denoted as $\mathcal{M} = \{1, 2, \ldots, M\}$, where each element represents a uniquely identified server instance. To reflect realistic deployment scenarios, each server operates under finite resource constraints, which offers a higher computational capacity than the RCIoT sensors but less than a centralized cloud infrastructure. Similar to other components, the servers rely on the Zenoh publish/subscribe protocol, with distinct keys assigned to each information flow. To ensure fairness and responsiveness, the servers follow a queuing approach (e.g., first come first serve (FCFS)) and support concurrent task execution through multithreading. Each server also periodically reports its resource utilization to the edge controller, hence enabling dynamic scheduling and load balancing. Once processing is complete, the server acknowledges task completion and publishes results back to the originating sensor.

3.1.5. Communication Component of EDECF

The communication component forms a core pillar of the EDECF, to enable low-latency, bidirectional task exchange among RCIoT sensors, the edge controller, and multiple edge servers. It is implemented using the Zenoh protocol, a lightweight middleware that unifies data-in-motion and data-at-rest through a publish/subscribe and query model [12,22]. In the following subsections, we provide an overview of Zenoh and its deployment in the EDECF.

https://doi.org/10.3390/ai7030081

Overview of Zenoh

Zenoh (Zero Overhead Network Protocol) is a brokerless, data-centric middleware designed to unify data in motion, data at rest, and computations [22]. Unlike traditional publish/subscribe systems, Zenoh decouples applications from the underlying transport and provides an abstraction layer where data is addressed using key expressions (string-based identifiers similar to hierarchical key names) [12]. These expressions allow data to be organized into Zenoh keys (topics), which follow a flexible naming hierarchy (e.g., edf/edge/sensor1/cpu_load, edf/edge/sensor2/queue_length). This design enables fine-grained data selection, aggregation, and routing across heterogeneous devices and networks.

Zenoh was chosen for the EDECF because it supports peer-to-peer communication, distributed routing, and adaptive forwarding, making it highly suitable for dynamic environments such as edge computing networks. By eliminating the need for a centralized broker, Zenoh reduces latency, improves scalability, and ensures resilience in the presence of dynamic workloads and node heterogeneity. Furthermore, it has been shown in the literature to perform well as compared to other distributed communication protocols as demonstrated in [12], thus justifying its choice for the EDECF.

Zenoh Deployment in EDECF

Within the EDECF, Zenoh serves as the base communication substrate connecting emulated nodes in the CORE environment (described in the next section). Each node (e.g., edge servers, IoT devices, and controllers) publishes local state variables, such as CPU utilization, queue lengths, task arrival rates, and response times, onto designated Zenoh keys. These keys are defined hierarchically to reflect both the node identity and the resource type, hence ensuring organized and queryable data streams across the emulated network.

For instance, keys such as

```
edf/edge/server1/cpu
edf/edge/server2/queue
edf/iot/sensor1/task_rate
```

were used to expose real-time performance indicators. The subscribing nodes, such as the EDECF controller, dynamically aggregate this information to monitor the global system state and to execute the metaheuristic-based load balancing algorithm. The optimized task assignment decisions are then disseminated back to the relevant edge servers through Zenoh's publish/subscribe channels.

Zenoh was deployed in the EDECF as a distributed runtime service, which is initialized on each emulated CORE node. This ensures that all interactions (publishing and subscribing) occur under conditions that capture network heterogeneity (that is, servers with different resource specifications), queuing and processing delays, and dynamic workload variations. The integration of Zenoh within the EDECF provides a lightweight yet realistic communication layer, thus enabling near-real emulation of EC workloads while preserving scalability.

3.2. EDECF Design Phases

We considered two major phases in the design of the EDECF, namely, the edge communication and edge computation phases, described as follows:

3.3. Edge Communication Phase

This phase establishes reliable user–edge connectivity to support efficient task forwarding to the EC center (ECC). Each sensor follows four sequential workflows: (1) data

acquisition and aggregation, (2) task forwarding, (3) acknowledgment receipt from the edge controller, and (4) response retrieval from the edge server. This sequential workflow design accounts for real-world conditions, such as when sudden shutdowns of ECC components (e.g., controller or servers) occur. In the worst case, a sensor may forward a task and wait for a response without knowing whether the task was successfully received or whether the assigned server failed before completing execution. Hence, detecting the source of such failures is considered in the EDECF.

To implement failure detection, the EDECF employs a lightweight heartbeat signaling approach to periodically verify the active status of sensors and servers. If acknowledgments are not received within a predefined wait time, the link is marked inactive and a retransmission mechanism is triggered. As shown in Figure 2, connectivity between sensor i and edge controller c is first established through a handshake mechanism, which ensures that both are operational before tasks are forwarded. This can be expressed as a binary indicator η:

$$\eta = \begin{cases} 1, & \text{acknowledgment received from edge controller,} \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

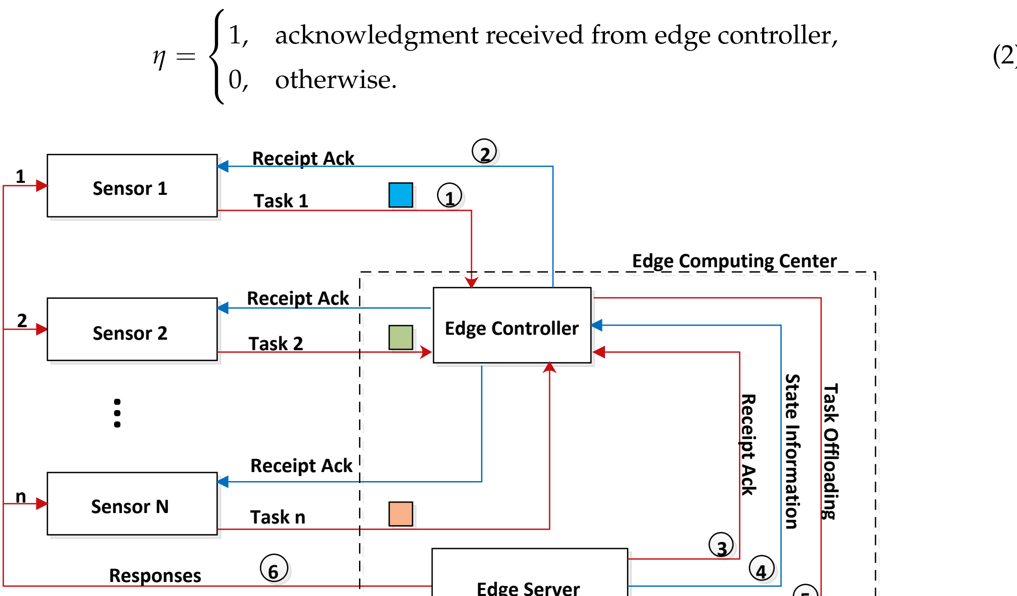

Figure 2. EDECF information flow.

Here, $\eta = 1$ indicates that task k was successfully admitted by the edge controller c, with an acknowledgment returned to the originating user i, while $\eta = 0$ denotes that no acknowledgment was received within the allowed time. Successfully processed tasks are stored in a completed-task table, thus enabling the controller to track execution, guarantee user responses, and prevent duplication. For efficient recording, the controller assigns unique identifiers c_i and d_i to map tasks to their originating sensors in each time slot, consistent with in the approach in [23]. The tasks are admitted into the completion task table only after completion. Once communication is secured, tasks are forwarded for optimal assignment. The EDECF employs a PSO-based optimization strategy that dynamically selects the most suitable edge server while accounting for queue lengths, processing capacities, and transmission delays (Figure 3). This ensures balanced workload distribution, prevents congestion, and achieves low-latency execution.

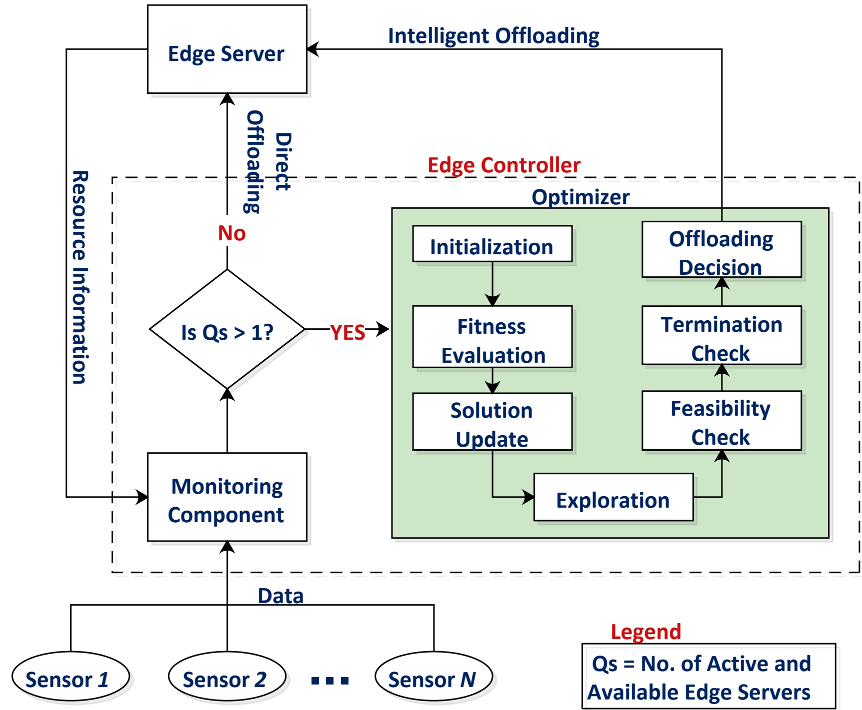

Figure 3. High-level depiction of metaheuristic-enabled load balancing algorithm.

3.4. Edge Computation Phase

To determine the optimal edge server in a dynamic edge environment, we propose an optimization framework (Figure 3) combining direct and intelligent offloading. It considers two ECC phases: the initial phase, with all edge servers at maximum capacity using direct offloading, and the resource depletion phase, where optimization using a load balancing technique is required as resources are intelligently and efficiently allocated to users' tasks.

As shown in Figure 3, the monitoring component collects real-time task data from sensors and server states, maintaining them in the sensor ($\mathcal{N}$) and server ($\mathcal{M}$) information tables. In the EDECF, offloading is enabled only when both tables are sufficiently populated. As shown in Algorithm 1, for the baseline case with a single server ($|\mathcal{M}| = 1$) at the ECC, no optimizer is required. At system startup, assuming the ECC has just initialized and the single server is at full capacity, tasks from sensor i are offloaded directly to the server. Following the FCFS principle, tasks from the first sensor in $\mathcal{N}$ are allocated to the server until it reports resource saturation; otherwise, tasks continue being queued for the same server. In this setup, the system latency experienced by sensors depends on the number of tasks waiting in the queue. The average system latency is composed of the submission time t_i^{sub} and the completion time t_i^{cmp}. Specifically, t_i^{sub} includes the forwarding delay t_i^{forw} (the time for the task from sensor i to reach the controller and receive acknowledgment) and the queuing delay t_i^{que}. The completion time t_i^{cmp} accounts for the processing delay at the allocated server, while the result-return delay is considered negligible. Hence, the latency of task i is defined as

$$\ell_i = t_i^{\text{cmp}} - t_i^{\text{sub}}. \tag{3}$$

In contrast to direct offloading, intelligent offloading leverages a load balancer, such as a metaheuristic optimizer. When the number of servers exceeds one ($|\mathcal{M}| > 1$), tasks in $\mathcal{N}$ are scheduled using the optimizer to identify the most suitable execution point among the available servers. Although the optimization phase introduces additional delay, it is

constrained to remain within the predefined deadline, thereby preserving the real-time requirements of EDECF. To this end, we discuss a metaheuristic-based load balancing algorithm in Section 4.

Algorithm 1 Load Balancing Mechanism in EDECF.

1: **Inputs:** $\mathcal{N}$: The information table of users. $\mathcal{M}$: The information table of servers. P: The population size. $I_{\max}$: The total iteration number.
2: **Outputs:** Mapping $\mathbf{x}^*$: each sensor $\rightarrow$ one edge server ID
3: Extract ordered sensor IDs: $\mathbf{U} \leftarrow \text{keys}(\mathcal{S})$; let $N \leftarrow |\mathbf{U}|$
4: Extract ordered server IDs: $\mathbf{V} \leftarrow \text{sorted}(\text{keys}(\mathcal{E}))$; let $M \leftarrow |\mathbf{V}|$
5: **if** $M = 0$ **then**
6:     **return** (no decision; wait for servers)
7: **else if** $M = 1$ **then**
8:     Assign all sensors to the single server $\mathbf{V}[0]$ and **offload**; **return**
9: **end if**
10: **if** $|\mathcal{N}| \neq 0 \ \land \ |\mathcal{M}| > 1$ **then**
11:     Enable the optimizer in Algorithm 2
12: **end if**

Algorithm 2 PSO-Based Load Balancing for EDECF.

1: **Inputs:** Refer to Algorithm 1
2: **Outputs:** Mapping $\mathbf{x}^*$: each sensor $\rightarrow$ one edge server ID
$**$ **Initialization...** $**$
3: **Initialize a discrete PSO over server ID assignments**
4: **for** p = 1 to P **do**
5:     Generate parameter values $\tilde{\mathbf{x}}^{(p)}$ with random task assignments
6:     Perform load balancing using (5) based on $\tilde{\mathbf{x}}^{(p)}$
7:     Compute fitness θ_p of optimal mapping using (5)
8:     $I \leftarrow I + 1$
9:     **if** $f_{\text{pbest}}^{(p)} \geq f_{\text{gbest}}$ **then**
10:         $f_{\text{gbest}} = f_{\text{pbest}}^{(p)}$
11:     **end if**
12: **end for**
$**$ **Main Loop...** $**$
13: **while** $I = 1$ to $I_{\max}$ **do**
14:     **for** p = 1 to P **do**
15:         Update velocity using (7)
16:         Update position using (8)
17:         Evaluate fitness
18:     **end for**
19:     Apply best mapping $\mathbf{x}_{\text{gbest}}$
20: **end while**
21: **return** $\mathbf{x}_{\text{gbest}}$

3.5. EDECF Implementation in CORE

This subsection describes how the proposed EDECF is realized within the common open research emulator (CORE) to enable realistic, repeatable emulation of edge computing environments. It first provides a brief overview of CORE as the underlying emulation platform, followed by a description of how EDECF components are instantiated as emulated nodes and services. The subsection then details the implementation of the sensor, edge controller, and edge server services, highlighting their interactions, communication mechanisms, and runtime behavior. Together, these descriptions illustrate how the EDECF operationalizes end-to-end task generation, offloading, scheduling, and execution within a controlled emulation framework.

3.5.1. Overview of CORE

CORE is a network emulator that enables the creation of virtual networks consisting of nodes, links, and services within a single Linux host or across distributed environments [24]. Each node in CORE runs as a network namespace, which isolates processes, interfaces, and routing tables, thereby providing a realistic environment for testing networked systems. CORE supports various link types (wired, wireless, point-to-point) and integrates with the Extendable Mobile Ad-hoc Network Emulator (EMANE) for modeling complex wireless and mobile scenarios [25]. A key strength of CORE lies in its flexibility, which allows users to design topologies through a graphical interface or configuration files, while also supporting service scripting for automating node behavior.

3.5.2. EDECF Customized Services

For the proposed EDECF, we used CORE as the foundational tool for building and managing the emulated edge computing environment. The framework instantiated heterogeneous nodes representing IoT sensors, edge servers, and controllers, each configured with the required services and processing scripts. CORE's ability to host custom Python 3.13 services enabled the seamless integration of resource monitoring modules (e.g., CPU usage, queue length) and the metaheuristic-based load balancing algorithm.

Communication among nodes was realized through the integration of Zenoh, which was deployed as a distributed runtime on each emulated node within CORE. This setup allowed the EDECF to capture realistic system behavior, including heterogeneity of resources, queuing and processing delays, and dynamic workloads, while still benefiting from the scalability of software-based emulation. In addition, each CORE element service is developed and described as follows:

Sensor Service

The sensor service flow shown in Figure 4 models the behavior of a sensor within an edge computing network, simulating its data generation, transmission, and interaction with an edge controller and server. Upon initialization, the sensor establishes Zenoh communication channels, sends an online status signal, and awaits acknowledgment to confirm connectivity. Once connected, it continuously measures energy consumption values at predefined intervals, batching and transmitting them as tasks to the edge controller for processing. Each transmission is followed by a waiting phase during which the sensor monitors acknowledgment and response messages to evaluate network reliability and task completion. If acknowledgments or responses are delayed beyond threshold limits, retransmission procedures are triggered to ensure data delivery, with each retry logged for performance tracking. Throughout this process, the system dynamically calculates end-to-end latency, maintains averages, and records retransmission counts, which are periodically plotted and saved for later analysis. The design emphasizes resilience, responsiveness, and fault-tolerant communication under dynamic network conditions, providing a robust simulation framework for evaluating sensor-to-edge interactions, latency behavior, and data reliability in edge computing environments.

Edge Controller Service

The edge controller service depicted in Figure 5 functions as a load balancer, continuously aggregating two live views: (i) pending sensor tasks and (ii) edge server health and capacity. For each sensor task cycle, the load balancer selects a destination server using a capacity- and fairness-aware strategy aimed at reducing queue buildup and keeping latency within bounds, then dispatches the task and monitors completion acknowledgments. Timeouts on either the controller or server side trigger bounded retransmissions at the sen-

sor, hence ensuring eventual delivery without unbounded growth in retries. Throughout the controller execution, the system tracks operational metrics-per-task latency, moving-average latency, total server queue length, and a fairness indicator so that performance and bottlenecks are visible in real time and saved for post-hoc analysis. The result is a modular, fault-tolerant pipeline where sensors, the controller/load balancer, and edge servers cooperate to maintain reliable throughput and stable latency under dynamic conditions.

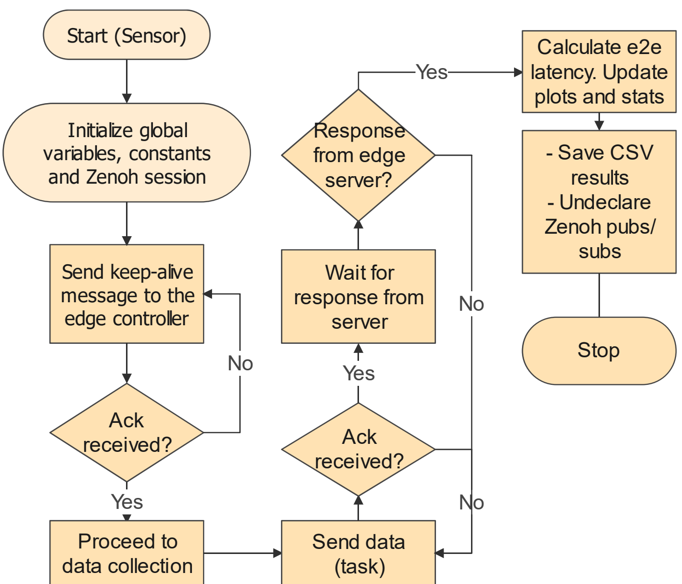

Figure 4. Sensor service process.

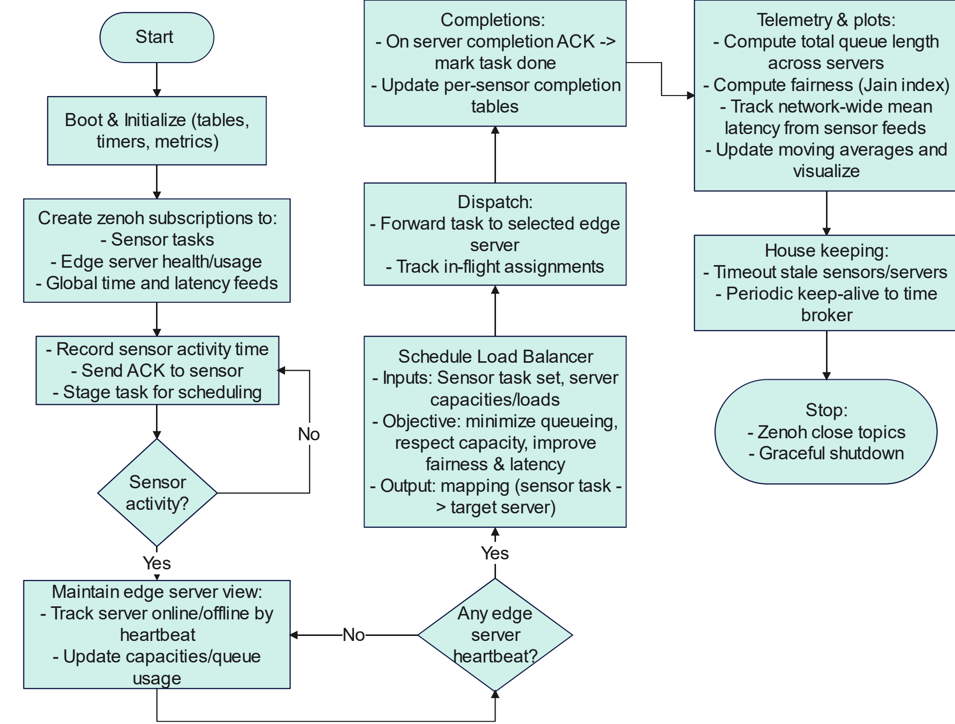

Figure 5. Edge controller service process.

Edge Server Service

In Figure 6, the edge server service was modeled as a dedicated worker node that receives the offloaded tasks from the controller. Operationally, it boots by establishing messaging endpoints, advertises its health through periodic resource-usage updates, and keeps the server alive while idling for tasks. When the controller dispatches a task (already

load-balanced to the correct destination), the server service validates the target identifier, acknowledges receipt back to the controller for reliability tracking, and processes the payload. Upon completion, it sends a task-completion acknowledgment to the controller and returns a processed-result message directly to the originating sensor, thereby closing the feedback loop. Throughout, lightweight counters (queue length, active threads/utilization) feed the telemetry stream so the controller can make informed scheduling decisions, while bounded, event-driven processing and clean teardown keep the node robust under dynamic loads and orderly during shutdown.

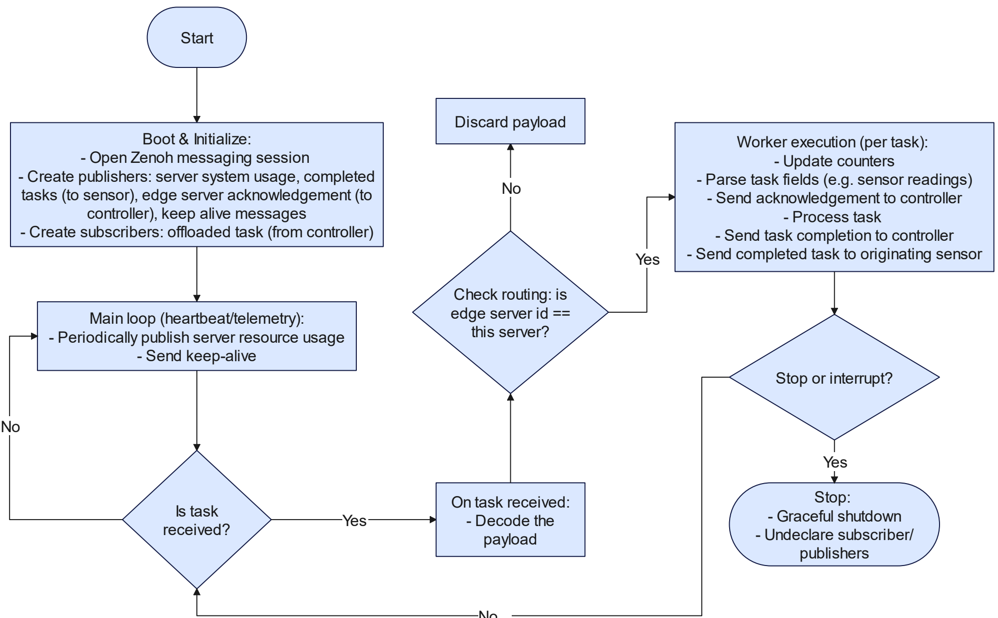

Figure 6. Edge server service process.

4. Metaheuristic-Based Load Balancing for Deployment in EDECF

4.1. Problem Overview and Design Rationale

The EDECF is designed as an algorithm-agnostic platform for evaluating metaheuristic-based load balancing strategies under near real-time, heterogeneous, and dynamic edge computing conditions. Rather than being tailored to a specific optimization technique, the EDECF provides a unified control and evaluation interface that enables different metaheuristic algorithms to observe system state, evaluate candidate task–server mappings, and deploy optimized decisions dynamically.

To demonstrate this capability, multiple representative metaheuristic algorithms were implemented within the EDECF, including particle swarm optimization (PSO), genetic algorithm (GA), differential evolution (DE), and Simulated Annealing (SA). All algorithms operate over the same optimization model, fitness function, and controller feedback mechanisms, thereby ensuring a fair and controlled comparison. The experimental results presented in Section 6 confirm that the EDECF can accommodate diverse classes of metaheuristics without modification to the underlying framework.

While PSO is presented in detail in this section as a representative case study, the framework itself is not dependent on PSO and remains fully independent of the chosen optimization method.

4.2. Mathematical Model

Consider a set of tasks generated by active sensors,

$$\mathcal{N} = \{1, \ldots, N\},$$

and a set of available edge servers,

$$\mathcal{M} = \{1, \ldots, M\}.$$

An edge solution is defined as a task–server assignment vector

$$\mathbf{x} = [x_1, x_2, \ldots, x_N],$$

where each decision variable $x_i \in \mathcal{M}$ denotes the edge server assigned to task i.

Let A_m denote the number of tasks assigned to server m and C_m its maximum number of processing threads. Feasible solutions must satisfy the capacity constraint

$$A_m \leq C_m, \quad \forall m \in \mathcal{M}, \tag{4}$$

where

$$A_m = \sum_{i=1}^{N} \mathbb{1}[x_i = m].$$

Equation (4) enforces the server capacity constraint within the proposed optimization model. Specifically, A_m represents the total number of tasks assigned to edge server m under a candidate solution $\mathbf{x}$, computed as the sum of indicator functions over all tasks. The parameter C_m denotes the maximum number of concurrent processing threads supported by server m. Therefore, Equation (4) ensures that no feasible solution assigns more tasks to a server than it can process simultaneously. Any candidate mapping that violates this constraint is considered infeasible and is discouraged during optimization through the penalty term in the fitness function (cf. Equation (5)). This mechanism allows metaheuristic algorithms to explore the solution space freely while systematically steering the search toward capacity-respecting and practically deployable task–server assignments.

4.3. Optimization Objective and Fitness Function

The objective is to minimize end-to-end task response time while ensuring balanced resource utilization and fairness across heterogeneous edge servers. This is captured by the fitness function

$$\begin{aligned} f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \left(\tau_{x_i} + W_{x_i} + D_i/\mu_{x_i}\right) \\ + \lambda_1 \sum_{m \in \mathcal{M}} \left[\max(0, A_m - C_m)\right]^2 + \lambda_2\big(1 - \mathcal{J}(\mathbf{A})\big), \end{aligned} \tag{5}$$

where τ_{x_i} is the communication delay, W_m is the queue waiting time at server m, μ_m is the service rate, D_i denotes task demand, and $\mathcal{J}(\mathbf{A})$ is the Jain fairness index (JFI),

$$\mathcal{J}(\mathbf{A}) = \frac{\left(\sum_{m=1}^{M} A_m\right)^2}{M \sum_{m=1}^{M} A_m^2}. \tag{6}$$

This formulation is independent of the optimization algorithm and is shared by all metaheuristics evaluated within EDECF. Specifically, it should be noted that Equation (5) defines the objective function used to evaluate the quality of a candidate task–server mapping $\mathbf{x}$. The first term represents the average end-to-end response time experienced by tasks, combining communication delay (τ_{x_i}), queue waiting time (W_{x_i}), and service time (D_i/μ_{x_i}). Minimizing this term directly promotes low-latency task processing.

The second term introduces a quadratic penalty for server overload. When the number of tasks assigned to a server exceeds its processing capacity ($A_m > C_m$), the squared excess

$\left[\max(0, A_m - C_m)\right]^2$ imposes an increasing cost, thereby strongly discouraging infeasible or highly congested assignments while still allowing metaheuristic algorithms to explore the search space.

The third term captures fairness across edge servers using the Jain fairness index defined in Equation (6). Since $\mathcal{J}(\mathbf{A}) \in (0,1]$, the expression $1 - \mathcal{J}(\mathbf{A})$ penalizes imbalanced load distributions. This encourages solutions that distribute tasks more evenly across available servers, preventing persistent hot spots.

The weighting coefficients λ_1 and λ_2 regulate the relative importance of capacity violations and fairness with respect to latency minimization. Together, these components form a multi-objective cost function that balances responsiveness, feasibility, and fairness within a single scalar fitness value, which enables efficient evaluation by diverse metaheuristic algorithms.

Furthermore, Equation (6) quantifies load fairness by measuring how uniformly tasks are distributed across servers. A value of $\mathcal{J}(\mathbf{A}) = 1$ indicates perfect balance, while lower values reflect increasing imbalance. Embedding this index within the fitness function provides an intuitive and computationally efficient mechanism for guiding the optimization toward well-balanced task–server mappings.

4.4. Metaheuristic-Agnostic Optimization Interface in EDECF

Within the EDECF, each metaheuristic interacts with the edge controller through a common interface consisting of: (i) real-time state acquisition (queue lengths, service rates, RTT proxies, CPU utilization, and server availability); (ii) generation of candidate task–server mappings; (iii) fitness evaluation using Equation (5); (iv) algorithm-specific evolution or update rules; and (v) deployment of the selected mapping to live task offloading.

This abstraction enables different metaheuristics, including PSO, GA, DE, and SA, to be integrated and evaluated interchangeably, thereby confirming the algorithm-agnostic design of the EDECF.

4.5. Representative Case Study: PSO-Based Load Balancing

Although the EDECF supports multiple metaheuristics, PSO is presented here as a representative case to illustrate the framework's operation in detail. PSO was selected for the following reasons:

- Low computational overhead: PSO requires fewer control parameters and simpler update rules compared to evolutionary algorithms such as GA and DE, making it suitable for real-time edge controllers.
- Fast convergence: PSO exhibits rapid convergence toward high-quality solutions, which is important in dynamic environments with frequently changing workloads.
- Ease of discretization: PSO can be efficiently adapted to discrete task–server assignment problems using lightweight encoding schemes.
- Wide adoption: PSO is widely used in the edge and cloud scheduling literature, enabling direct comparison with existing studies.

It is emphasized that, in addition to PSO, the algorithms GA, DE, and SA were also implemented and evaluated within EDECF, and their comparative performance is reported in Section 6. The multiple comparative test results provided in the Results Section aims to establish that the framework does not favor a particular optimization technique.

Nevertheless, for clarity purposes, Algorithm 2 implements a discrete particle swarm optimization (PSO) procedure to solve the task–server assignment problem defined in Section 4. In this context, each particle represents a candidate mapping of tasks generated by sensors to available edge servers. The objective of the swarm is to identify an assignment that minimizes the fitness function in Equation (5).

https://doi.org/10.3390/ai7030081

During the *initialization phase*, a swarm of P particles is generated with random task–server assignments. For each particle, the fitness of the corresponding mapping is evaluated, and both the personal best ($\mathbf{x}_{\text{pbest}}$) and global best ($\mathbf{x}_{\text{gbest}}$) solutions are recorded.

The velocity of particle p at iteration $t+1$ is updated as

$$
\begin{aligned}
v_i^{(p)}(t+1) = \omega\, v_i^{(p)}(t) \\
&+ c_1 r_{i,p}^{(1)}(t)\Big(p_i^{(p)} - x_i^{(p)}(t)\Big) \\
&+ c_2 r_{i,p}^{(2)}(t)\Big(g_i - x_i^{(p)}(t)\Big),
\end{aligned}
\tag{7}
$$

where ω is the inertia weight controlling momentum, c_1 and c_2 are the cognitive and social acceleration coefficients, $p_i^{(p)}$ denotes the personal best position of particle p, g_i represents the global best position, and $r_{i,p}^{(1)}(t)$ and $r_{i,p}^{(2)}(t)$ are random values sampled uniformly from $[0, 1]$.

The particle position is then updated using

$$
x_i^{(p)}(t+1) = x_i^{(p)}(t) + v_i^{(p)}(t+1), \tag{8}
$$

after which a discretization step maps the continuous value to a valid edge server index. This process is repeated iteratively until convergence or the maximum iteration limit is reached.

In the *main optimization loop*, particles iteratively update their trajectories in the solution space by adjusting their velocities and positions. The velocity update rule, given in Equation (7), combines three components: (i) an inertia term that preserves the particle's previous motion, (ii) a cognitive component that attracts the particle toward its personal best solution, and (iii) a social component that attracts it toward the global best solution discovered by the swarm. This balance between exploration and exploitation enables efficient search of the combinatorial assignment space.

The updated velocity is then used to compute a new particle position according to Equation (8). Since the task–server assignment problem is discrete, the resulting continuous positions are mapped to valid server identifiers using a discretization operator, ensuring that each task is assigned to exactly one available edge server.

After each update, the fitness of the new assignment is evaluated. If the new solution improves upon the particle's personal best, the personal best is updated accordingly. The global best solution is also updated whenever a particle achieves a better fitness than the current global best.

Once the maximum number of iterations is reached or convergence is observed, the global best mapping $\mathbf{x}_{\text{gbest}}$ is applied by the EDECF controller for live task offloading. This optimized mapping is subsequently monitored, and fallback mechanisms are triggered if server availability or network conditions change.

4.6. Initialization, Selection, and Online Fallback

Initialization, selection, and fallback mechanisms follow the same principles across all evaluated metaheuristics. Although the encoding and evolution rules shown above correspond to PSO, the solution representation, fitness evaluation, constraint handling, and fallback logic are reused unchanged by GA, DE, and SA. This ensures that performance differences observed in the results stem from the optimization strategies themselves rather than from framework-level bias.

https://doi.org/10.3390/ai7030081

4.7. Edge Solution Evolution and Constraint Handling

During optimization, candidate solutions evolve iteratively until convergence or the maximum iteration limit $I_{\max}$ is reached. Constraints are enforced at every iteration: offline servers cannot be assigned tasks, overload conditions are penalized through the fitness function, and retransmission safeguards are applied when acknowledgments are not received within a predefined time window. Through this iterative process, EDECF enables robust, queue-aware task–server mappings under realistic and dynamic edge computing conditions.

4.8. Computational Complexity Analysis

This subsection analyzes the computational complexity of the load balancing strategies evaluated within the proposed EDECF. To analyze the complexity of different approaches, let N denote the number of tasks awaiting offloading, M the number of edge servers, S the population size for population-based metaheuristics, and I the maximum number of iterations.

4.8.1. Baseline Heuristic Methods

Round Robin (RR): The round robin algorithm assigns tasks sequentially without considering system state. Each assignment requires constant time, yielding a complexity of $\mathcal{O}(1)$ per task, $\mathcal{O}(N)$ per scheduling cycle.

Shortest Queue (SQ): The shortest queue heuristic selects the server with the minimum queue length. This requires scanning all available servers, resulting in $\mathcal{O}(M)$ per task, $\mathcal{O}(NM)$ per scheduling cycle.

Although these heuristic approaches are computationally efficient, they lack global optimization capability under heterogeneous and dynamic edge conditions.

4.8.2. Metaheuristic-Based Methods

Metaheuristic algorithms iteratively evaluate candidate task-to-server assignments. The dominant computational cost arises from the fitness evaluation, which considers queue length, processing capacity, and workload distribution across servers.

Particle Swarm Optimization (PSO): In PSO, each particle represents a candidate solution. At each iteration, fitness evaluation across all servers incurs a cost of $\mathcal{O}(NM)$, while velocity and position updates incur $\mathcal{O}(N)$. Therefore, the overall complexity is $\mathcal{O}(I \times S \times NM)$.

Genetic Algorithm (GA): The GA involves selection, crossover, mutation, and fitness evaluation. Since fitness evaluation dominates the computational cost, the overall complexity is $\mathcal{O}(I \times S \times NM)$.

Differential Evolution (DE): DE performs mutation and recombination operations for each candidate solution, followed by fitness evaluation, leading to a complexity of $\mathcal{O}(I \times S \times NM)$.

Simulated Annealing (SA): SA evaluates a single candidate solution per iteration. As a result, its computational complexity is given by $\mathcal{O}(I \times NM)$, which is lower than population-based methods but may require more iterations to achieve convergence.

It should be noted that although metaheuristic-based approaches exhibit higher computational complexity than heuristic baselines, their execution remains feasible within the EDECF due to moderate population sizes, bounded iteration counts, and centralized execution at the edge controller. This analysis highlights the trade-off between optimization quality and computational overhead, and confirms that the EDECF supports systematic evaluation of both lightweight heuristics and more sophisticated metaheuristic load

balancing strategies under realistic emulated edge computing conditions, which will be demonstrated in the Results Section.

5. Baseline Method and Performance Evaluation Metrics and Settings

This section describes the baseline method used and the performance metrics for evaluating the proposed metaheuristic-based load balancing scheme within the EDECF, benchmarked against the round robin and shortest queue baseline methods across diverse experimental scenarios.

5.1. Baseline Methods

To evaluate the effectiveness of the proposed metaheuristic-based load balancing strategy in the EDECF, we compared their performance against the widely adopted baseline round robin (RR) and shortest queue (SQ) techniques [26]. The essence was to demonstrate the feasibility and realistic implementation of the EDECF and its capability for studying load balancing techniques. Essentially, the RR and SQ techniques were selected because of their simplicity, popularity, and ability to evenly distribute tasks across servers, which makes them a natural reference point. By benchmarking against the RR and SQ, we can clearly assess the improvements, if any, achieved by the metaheuristics optimization-based approaches such as the particle swarm optimization (PSO), genetic algorithm (GA), differential evolution (DE) and simulated annealing techniques in terms of their scalability and latency management capabilities.

Specifically, in the RR approach, tasks are assigned sequentially to servers in a cyclic order, ensuring an even distribution of task counts across all servers. Using the information tables of both the sensors $\mathcal{N}$ and the servers $\mathcal{M}$ (defined in Section 3.4), the server assigned to task $i \in \mathcal{N}$ is given by

$$s(i) = \big((i-1) \bmod |\mathcal{M}|\big) + 1, \qquad \forall i \in \mathcal{N}. \tag{9}$$

This formulation guarantees that each server receives either

$$n_m = \left\lfloor \frac{N}{M} \right\rfloor \quad \text{or} \quad \left\lfloor \frac{N}{M} \right\rfloor + 1 \tag{10}$$

tasks, thereby balancing task counts without considering queue states or server capacities.

On the other hand, in the SQ approach, each arriving task is assigned to the edge server that currently exhibits the smallest queue backlog, with the objective of minimizing waiting time by avoiding congested servers. Using the information tables of the sensors $\mathcal{N}$ and the servers $\mathcal{M}$ (defined in Section 3.4), let $q_m(t)$ denote the instantaneous queue length of server $m \in \mathcal{M}$ at decision time t. The server selected for task $i \in \mathcal{N}$ is therefore given by

$$s(i,t) = \arg\min_{m \in \mathcal{M}} q_m(t), \qquad \forall i \in \mathcal{N}. \tag{11}$$

In cases where multiple servers share the same minimum queue length, a deterministic tie-breaking rule is applied by selecting the server with the smallest index, i.e.,

$$s(i,t) = \min\left\{ m \in \mathcal{M} \;\middle|\; q_m(t) = \min_{k \in \mathcal{M}} q_k(t) \right\}. \tag{12}$$

Unlike the RR strategy, the SQ approach explicitly incorporates real-time queue state information into the scheduling decision. While this enables reduced queuing delays under dynamic workloads, it does not explicitly account for heterogeneous server capacities and may lead to imbalanced task allocations when service rates differ significantly.

https://doi.org/10.3390/ai7030081

5.2. Performance Metrics

The different load balancing schemes were evaluated in the EDECF using the following metrics:

5.2.1. Average System Latency

System latency captures the responsiveness of the EDECF by measuring the time elapsed from task submission at the sensor to its completion at the edge server. The latency for task i is defined as

$$\ell_i = t_i^{\text{cmp}} - t_i^{\text{sub}}, \tag{13}$$

where t_i^{sub} and t_i^{cmp} denote the submission and completion times of task i, respectively. Specifically, t_i^{sub} includes the transmission delay until acknowledgment at the edge controller, the queuing delay, and the optimization time. The completion time t_i^{cmp} primarily consists of the processing delay at the allocated server and the return of the result to the originating sensor. Since the return time is negligible, the average system latency across all successfully completed tasks is expressed as

$$\bar{\ell} = \frac{1}{N_c} \sum_{i=1}^{N_c} \ell_i, \tag{14}$$

where N_c denotes the total number of completed tasks.

5.2.2. Retransmission Attempts

This refers to the number of times a task is resent due to failures such as lost acknowledgments, server unavailability, or timeouts, used as a reliability indicator.

$$R_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} r_i, \tag{15}$$

$$R_{\text{rate}} = \frac{\sum_{i=1}^{N} r_i}{T}, \tag{16}$$

$$R_{\text{ratio}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{r_i > 0\} \tag{17}$$

where r_i is the number of re-sends for task i, T is the total observation period, and $\mathbb{1}\{\cdot\}$ is the indicator function.

5.3. Parameter Settings

The set of system and algorithmic parameters used in the experiments are summarized in Table 2. These parameters define the dynamics of task generation, server activity detection, and the search behavior of the metaheuristic-based optimizers. Essentially, Table 2 summarizes the key experimental parameters used to evaluate the proposed EDECF under controlled yet realistic edge computing conditions. The selected values were intended to balance emulation realism, computational feasibility, and reproducibility, while enabling meaningful comparison between baseline and metaheuristic-based load balancing strategies.

The number of sensors and servers reflects a moderate-scale multi-edge deployment that is sufficiently complex to induce contention and queuing effects, yet practical for software-based emulation. Sampling and task-generation intervals were chosen to emulate continuous IoT monitoring workloads. The inactive periods for sensors and servers introduced controlled dynamics and failure scenarios, thus allowing the framework's robustness to be evaluated.

For the metaheuristic-based optimization, the population size and iteration range were varied to study convergence behavior and scalability trade-offs. The inertia weight and acceleration coefficients follow commonly adopted values in the literature to ensure stable convergence and balanced exploration–exploitation dynamics. Overall, these settings ensured a fair, transparent, and representative evaluation of the EDECF and the considered load balancing techniques.

Table 2. Experimental and metaheuristic parameter settings.

Category	Parameter	Value
System Parameters	Number of sensors	5–10
	Number of servers	1–4
	Sampling time	0.01 s
	Server inactive period	2 s
	Sensor inactive period	10 s
	Task arrival pattern	Periodic (fixed interval)
PSO Parameters	Population size	10–100
	Number of iterations	10–100
	Inertia weight (w)	0.5
	Cognitive coefficient (c_1)	1.5
	Social coefficient (c_2)	1.5
	Fitness weight (λ)	0.5
GA Parameters	Population size	10–100
	Number of generations	10–100
	Selection method	Tournament selection
	Crossover probability	0.8
	Mutation probability	0.05
SA Parameters	Initial temperature	1.0
	Cooling schedule	Exponential
	Cooling rate (α)	0.95
	Stopping criterion	Max. iterations/convergence
DE Parameters	Population size	10–100
	Number of generations	10–100
	Mutation factor (F)	0.5
	Crossover rate (CR)	0.9

6. Experimental Results and Discussion

This section presents the results and discussion of the performance evaluation, focusing on the load balancing techniques evaluated in the EDECF across the different metrics discussed in Section 5.2.

6.1. Experimental Environment

The experimental evaluation of the proposed EDECF was conducted using the common open research emulator (CORE) running on Ubuntu Linux. All experiments were executed on a laptop equipped with an Intel Core i7 dual-core processor and 32 GB of RAM. CORE was used to instantiate and manage the emulated edge computing environment, including heterogeneous nodes representing IoT sensors, base stations, an edge controller, and multiple edge servers, each operating within isolated network namespaces.

Although all experiments were executed on a single physical machine, the use of CORE ensures logical isolation of nodes and reproducible network conditions, which makes the results suitable for comparative evaluation of scheduling and load balancing strategies in addition to raw performance benchmarking.

https://doi.org/10.3390/ai7030081

System Emulation Validation

Figure 7 illustrates the emulated deployment of the EDECF within CORE. The architecture comprises multiple sensors and an ECC consisting of key network and compute components, including switches, routers, and heterogeneous edge servers. For illustration purposes, Figure 7 shows a network with 10 sensors and two edge servers; however, the framework is inherently scalable and can be extended to a larger number of components depending on the designer's requirements and the physical resources of the computer where emulation is conducted. Note that all communication messages within EDECF are enabled by Zenoh sessions, which provide efficient, reliable, and data-centric message exchange.

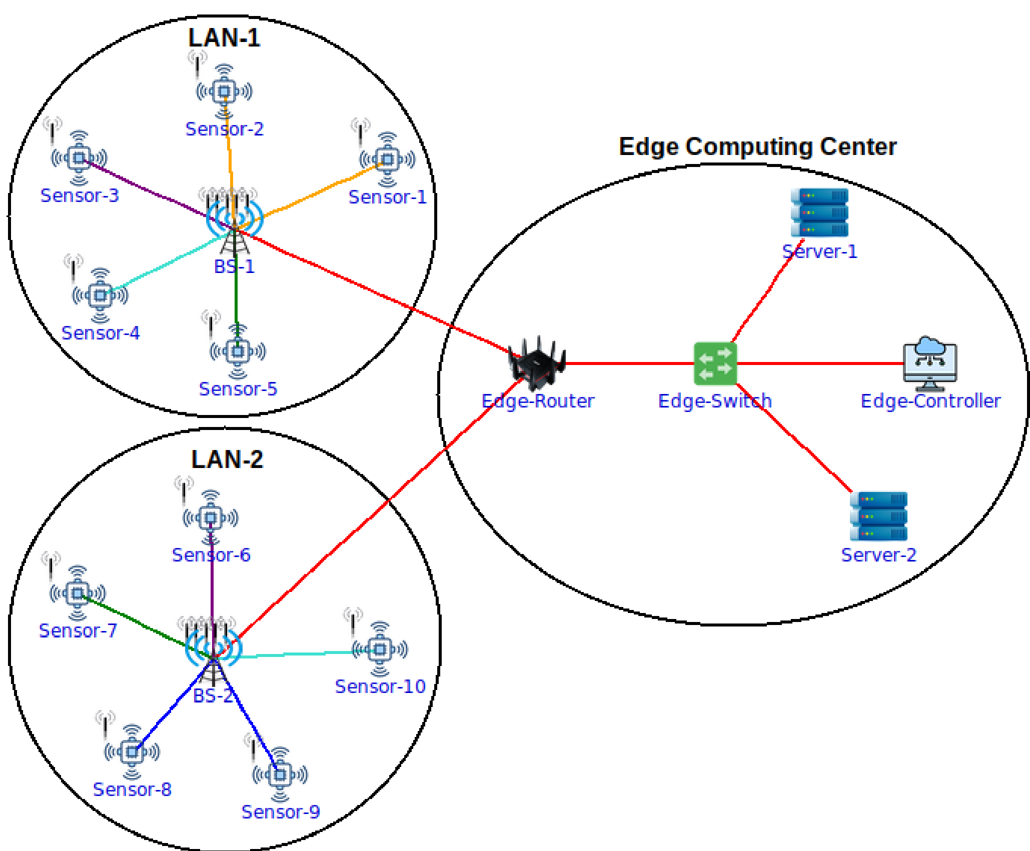

Figure 7. EDECF framework implemented in CORE.

In the emulated topology of Figure 7, two local area networks (LANs) host multiple sensors that periodically generate computation tasks and forward them through their respective base stations (BS-1 and BS-2). These BSs provide wireless aggregation and backhaul connectivity to the ECC. Within the ECC, the edge router routes data coming from the different LANs, while the edge switch provides the high-speed interconnect fabric. The edge controller orchestrates task management, including sensor discovery, acknowledgment handling, and queue-aware scheduling. On the other hand, heterogeneous edge servers (Server-1 and Server-2) execute the assigned tasks, with each server configured to reflect different computational capacities using number of threads.

This setup validates the ability of the EDECF to operate in a multi-LAN, multi-server environment where traffic heterogeneity and resource diversity can be systematically studied. The clear separation of access, routing, and compute layers of the EDECF allows evaluation of queue-aware decision-making, fairness across domains, and responsiveness to failures. By emulating realistic conditions such as variable wireless links, heterogeneous server capacities, and heartbeat-based liveness monitoring. The CORE-based topology

provides a controlled yet flexible testbed. This enables rigorous analysis of key performance metrics, including system latency, fairness, deadline–miss ratio, and retransmission attempts. Overall, the emulation confirms that the EDECF effectively integrates communication, control, and computation in a unified framework, hence supporting the evaluation of load balancing, reliability, and scalability under near-realistic EC conditions.

6.2. EDECF Terminal Emulation Validation

Furthermore, we show actual real-time emulation evidence based on the terminal logs in Figure 8a–f, which demonstrate the dynamic operational states of the EDECF during emulation. These outputs provide evidence of how the system responds to different initialization and runtime conditions across sensors, the edge controller, and edge servers.

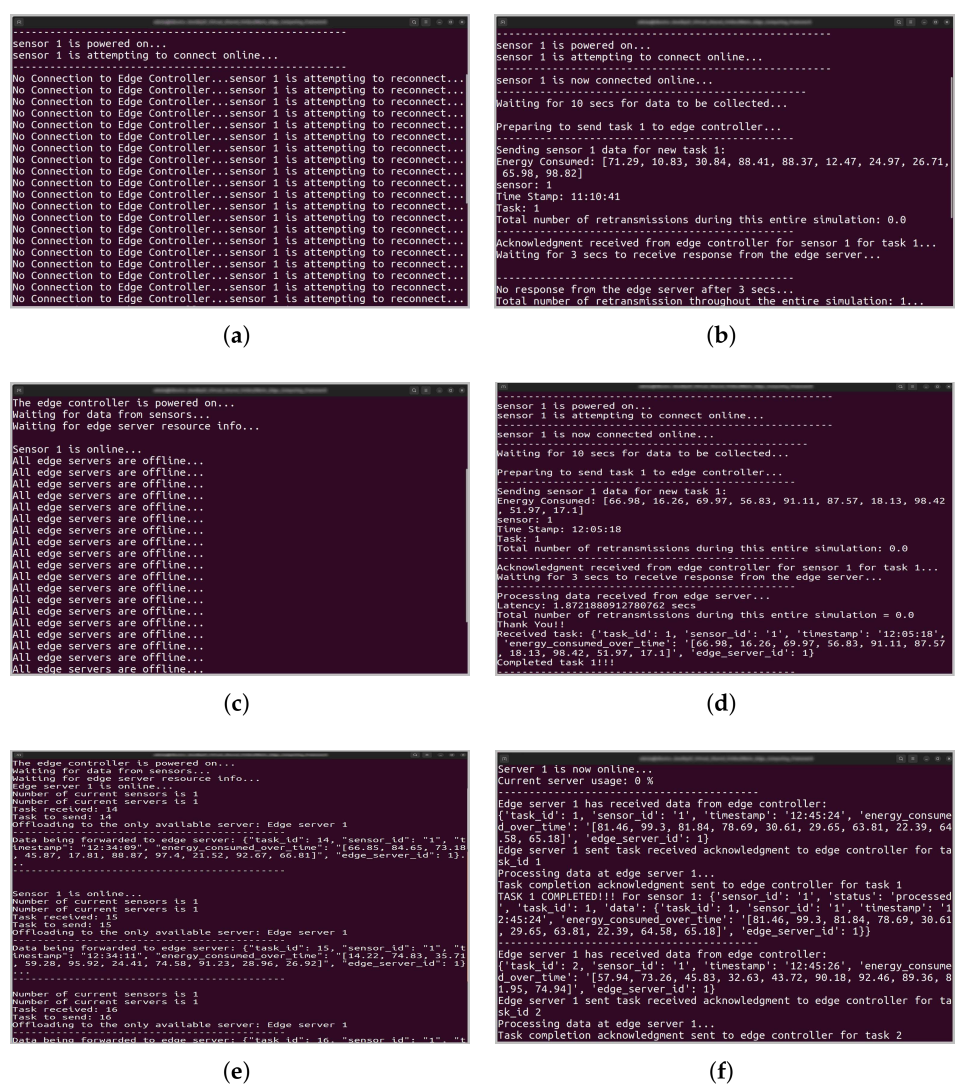

Figure 8. Sample terminal logs of EDECF components under varying operating states. (**a**) Sensor log: controller offline state. (**b**) Sensor log: connectivity established but no server response. (**c**) Edge controller log: sensor online with all servers offline. (**d**) Sensor log: task forwarded and response received. (**e**) Edge controller log: task received from sensor and offloaded to server. (**f**) Edge server log: tasks processed and feedback returned.

Specifically, Figure 8a demonstrates a scenario where the sensor powers on and attempts to establish connectivity but repeatedly retries due to the absence of an active edge controller. This highlights the framework's built-in reconnection and fault-tolerance mechanisms.

Once the controller is powered on, as shown in Figure 8b, the sensor is acknowledged, but since all servers are offline, tasks cannot be forwarded. The logs confirm that the controller continuously monitors server availability. In Figure 8c, when the controller is reachable but the servers are unresponsive, the sensor transmits data, receives acknowledgment from the controller, but fails to get results from the servers. Retransmission attempts are logged, validating the reliability and recovery capability of the EDECF.

When at least one server becomes active, the controller immediately forwards received tasks, logging the current number of connected sensors and servers. This confirms the controller's role in resource discovery, queue management, and scheduling as illustrated in Figure 8d. Finally, in Figure 8e,f, it can be seen that the edge server terminal shows successful task reception, processing, and completion acknowledgments, which confirms end-to-end execution in the EDECF.

6.3. Dynamic Sensor and Edge Controller Participation

In realistic IoT deployments, sensors may intermittently join or leave the network due to mobility, energy constraints, or failures, while the edge controller itself may experience restarts or temporary outages. Another common case arises when IoT sensors are powered on at different time intervals according to end-user requirements. This scenario is also considered in the EDECF; for example, a smartphone that is switched off due to low battery and later turned back on, while other sensors remain continuously active. Such dynamics introduce uncertainty in task forwarding and acknowledgment, directly impacting system stability and reliability. To evaluate robustness, we emulate scenarios where both sensors and the edge controller dynamically join and leave the network as depicted in Figures 9 and 10. The results presented here focus on key performance aspects, including task continuity, acknowledgment success, retransmission attempts, and overall system responsiveness.

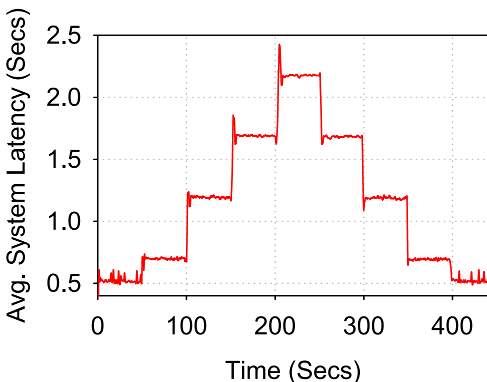

Figure 9. Dynamic sensor connectivity.

6.3.1. Effect of Sensors Joining/Leaving the Network

Figure 9 illustrates the system latency when sensors dynamically join and leave the network while the controller remains in the ON state. The results show a clear step-wise behavior in latency, beginning with a stable baseline of approximately 0.5 s when only a few sensors are active. As additional sensors join the network around 100–200 s, the workload on the system increases, leading to sharp rises in average latency that peak at about 2.3 s. This growth reflects the accumulation of tasks in server queues and the effect of heterogeneous server capacities, where weaker servers become bottlenecks under higher load. Once sensors begin leaving the network after 200 s, the latency decreases in corresponding steps, dropping from around 2.2 s to 1.7 s and then to 1.2 s, before eventually returning to its baseline level of 0.5–0.6 s by 350–400 s. These trends demonstrate the

workload sensitivity of the EDECF, where latency scales with the number of active sensors but also highlight its robustness, as the system remains stable throughout and quickly recovers to low-latency operation once the load diminishes. Overall, the results confirm that the EDECF can effectively manage dynamic sensor participation under an active controller, ensuring task continuity and stable operations despite fluctuating workloads.

6.3.2. Effect of Multiple Sensors Joining and Leaving with Controller Active/Inactive

Figure 10a–c present the impact of dynamic sensor participation combined with abrupt edge controller shutdowns on overall system performance. Two key performance metrics are tracked in real time: the average system latency and the number of retransmission attempts. The latency metric captures the end-to-end delay from task submission to completion, while retransmission attempts record each instance where a sensor resends a task to the edge controller due to the absence of an acknowledgment or response. When multiple sensors join, latency initially rises due to increased queuing and scheduling overhead. During controller outages (observable at 100 s and 225 s), latency temporarily collapses to zero because tasks cannot be scheduled or acknowledged, effectively stopping system progress.

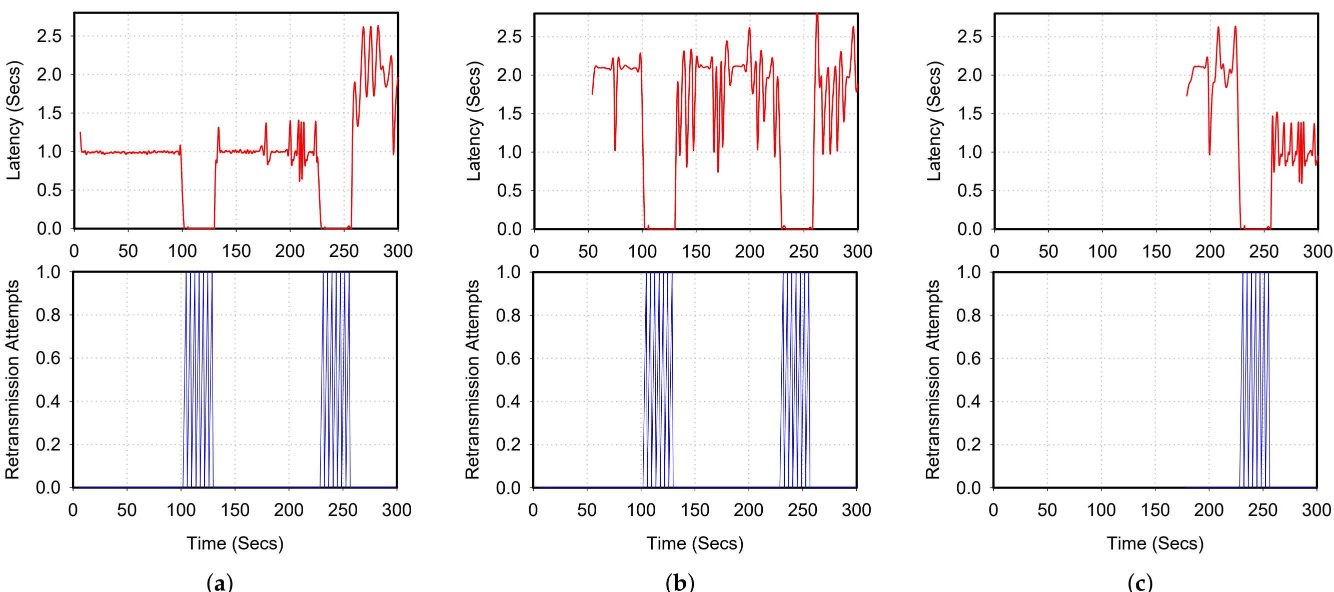

Figure 10. Evaluation of sensor dynamics with controller operational and non-operational States. (**a**) Moderate disturbances: latency and retransmissions around controller down/up. (**b**) High load: large latency swings and dense retransmissions during outages. (**c**) Reduced load: smoother latency with few retransmissions after outage.

Once the controller recovers, latency spikes occur as backlogged tasks are retransmitted and processed simultaneously, leading to congestion before the system stabilizes again. With more sensors active (Figure 10), baseline latency is consistently higher (2.0 s), showing that system load amplifies the impact of controller failures. The bottom sub-figures show sharp bursts of retransmission activity aligned with controller downtime intervals. This indicates that sensors are actively retrying transmissions until acknowledgments are restored, confirming that the retransmission mechanism is functioning as a fault-recovery strategy.

It is evident that both the frequency and intensity of retransmissions increase with the number of active sensors, highlighting the greater recovery burden in multi-sensor deployments. For example, Figure 10a illustrates moderate latency growth and limited retransmissions under a low sensor load. In contrast, Figure 10b exhibits pronounced latency fluctuations and dense retransmission bursts when multiple sensors join simul-

taneously, placing significant stress on the system. Finally, Figure 10c shows that as the number of active sensors decreases after approximately 200 s, the system experiences smoother latency recovery and fewer retransmissions, confirming that resilience improves as workload pressure diminishes. These results validate that the EDECF remains resilient to dynamic participation and controller faults through retransmission-based recovery, but also reveal a trade-off: under heavy load, retransmissions amplify congestion and increase latency variance. This highlights the importance of controller redundancy and load-aware admission control to ensure stable latency performance in large-scale deployments.

6.4. Evaluation of Metaheuristic-Based Load Balancing Techniques in EDECF

Figure 11a–d present the average network latency performance of four metaheuristic-based load balancing algorithms, namely differential evolution (DE), genetic algorithm (GA), particle swarm optimization (PSO), and simulated annealing (SA), evaluated using the proposed EDECF. The experiments systematically investigate the effects of network scale (number of sensors), population size, and number of iterations on latency performance under realistic edge computing conditions.

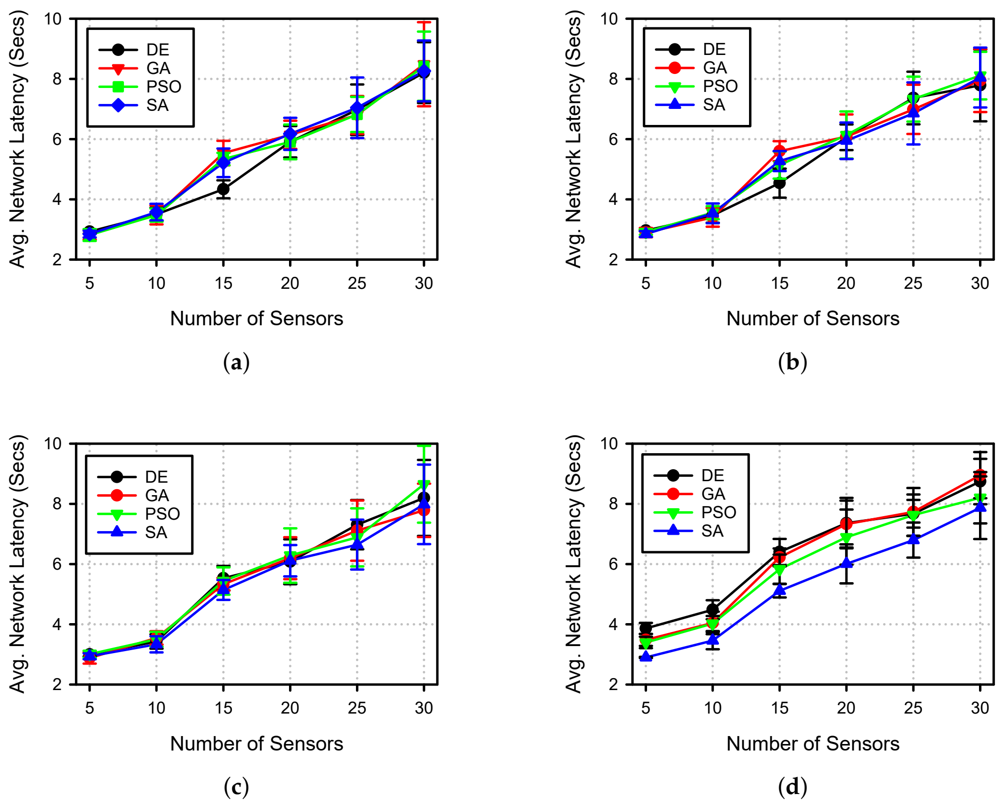

Figure 11. Average network latency performance of DE, GA, PSO, and SA under varying population sizes and iteration counts using the proposed EDECF. (**a**) Population size = 10, Iterations = 10. (**b**) Population size = 100, Iterations = 10. (**c**) Population size = 10, Iterations = 100. (**d**) Population size = 100, Iterations = 100.

6.4.1. Impact of Network Scale

Across all experimental configurations, the average network latency increased monotonically with the number of sensors, ranging from 5 to 30. This trend was consistent across all algorithms and parameter settings and reflected the expected increase in communication overhead, task contention, and scheduling complexity as network density grew. The smooth and consistent scaling behavior observed in all four conditions of Figure 11 indicates that

EDECF accurately captures load-induced performance degradation without introducing instability, thereby validating its suitability for scalable edge network emulation.

While all algorithms exhibited an upward latency trend, their relative performance diverged as network scale increased, thus highlighting differences in their ability to manage growing workloads effectively.

6.4.2. Effect of Population Size at Low Iteration Counts

Figure 11a,b compares the impact of increasing population size from 10 to 100 while keeping the number of iterations fixed at 10. It was seen that a larger population generally improved solution diversity, which led to slight reductions in mean latency and improved stability at moderate sensor counts, particularly for PSO and SA.

In contrast, DE and GA showed comparatively marginal improvements under low iteration budgets, suggesting that their evolutionary operators require sufficient iteration depth to fully exploit the increased population diversity. This highlights that population size alone does not guarantee improved performance unless accompanied by adequate iteration counts.

6.4.3. Effect of Iteration Count at Small Population Sizes

The effect of increasing the number of iterations was illustrated by comparing Figure 11a,c, both using a population size of 10. Increasing the number of iterations from 10 to 100 led to improved convergence behavior and more stable latency trends across all algorithms.

PSO and SA benefited more noticeably from increased iterations, thus indicating faster exploitation of promising solutions and smoother convergence dynamics. In contrast, DE and GA exhibited diminishing returns at higher sensor counts due to limited population diversity, which restricted their ability to escape suboptimal regions of the search space.

6.4.4. Combined Effect of Large Population and High Iteration Count

Figure 11d represented the most computationally intensive configuration, combining a population size of 100 with 100 iterations. This configuration consistently yielded the lowest or near-lowest latency values across most sensor counts, particularly for PSO and SA.

SA demonstrated strong robustness at higher sensor densities, suggesting that its probabilistic acceptance mechanism helps avoid premature convergence in complex, high-load scenarios. PSO also exhibited competitive performance, benefiting from collective learning and information sharing within a large population.

These results highlight the trade-off between optimization quality and computational overhead, a critical consideration in resource-constrained edge environments.

6.4.5. Performance Variability and Framework Implications

The observed increase in latency variance with growing network size reflects the stochastic nature of metaheuristic optimization under dynamic and congested conditions. Importantly, the EDECF captures both mean performance and variability, hence enabling comprehensive evaluation of algorithm robustness rather than relying solely on average metrics.

Overall, the results demonstrate that the EDECF provides a flexible and realistic experimental platform for evaluating metaheuristic-based load balancing strategies and enables systematic analysis of scalability, convergence behavior, and parameter sensitivity in edge computing systems.

https://doi.org/10.3390/ai7030081

6.5. Effect of Server Scaling on Network Latency

Figure 12 illustrates the impact of scaling the number of edge servers alongside an increasing number of sensors on the average network latency. For the single-server configuration, the average latency increased sharply as the number of sensors grew from 5 to 30. This increase was accompanied by a notable rise in performance variability, particularly at higher sensor densities.

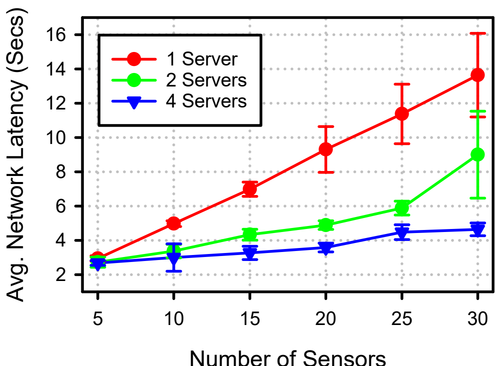

Figure 12. Average network latency as a function of the number of sensors for different numbers of edge servers.

When the number of servers was increased to two, the observed latency growth was significantly reduced across all sensor counts. The latency values remained consistently lower than those of the single-server case, indicating improved load distribution and reduced processing contention at the edge.

The four-server configuration exhibited the lowest average latency across all experimental points. Even at the highest sensor count, the latency increase remained moderate, and the variability was comparatively constrained. This demonstrated that additional edge servers effectively absorbed the increased workload generated by higher sensor densities.

The observed trends indicate that distributing computational load across multiple edge servers mitigates congestion effects and reduces task queuing delays. As the number of servers increased, sensor-generated tasks were processed closer to their sources, leading to reduced communication overhead and improved parallelism in task execution.

From a systems perspective, these results highlight that network latency is strongly influenced not only by sensor density but also by the availability of edge-side computational resources. The results further demonstrate that scaling edge resources proportionally with sensing infrastructure is an effective strategy for maintaining latency stability in dense IoT deployments.

Importantly, these experiments demonstrate that the EDECF supports controlled and systematic scaling of both sensing nodes and edge servers within the same emulated environment. The EDECF enables fine-grained manipulation of infrastructure parameters while preserving realistic communication and processing dynamics, allowing the direct observation of performance trade-offs under varying deployment scenarios.

Overall, the results confirm that the EDECF provides an experimental platform for studying resource scaling strategies in edge computing systems. The framework enables reproducible evaluation of how infrastructure provisioning decisions impact key performance metrics such as latency, thereby supporting informed design and optimization of scalable edge computing architectures.

6.6. Comparative Performance of Metaheuristic and Baseline Techniques

In this sub-section, we present the results of the performance of baselines and metaheuristic-based load balancers evaluated in the EDECF. The comparison focuses on analyzing how performance evolves as the number of servers increases under homogeneity, where servers have equal capacities and heterogeneity, where servers differ in resources and optimization-based placement becomes critical.

6.6.1. Comparison of Load Balancing Techniques Under Homogeneous Server Conditions

Figure 13a–c present the average network latency obtained when comparing different load balancing techniques under homogeneous server configurations, where each server was provisioned with an equal number of worker threads fixed at two per server. The experiments were conducted by scaling both the number of sensors and the number of servers in order to evaluate latency behavior under balanced computational resources.

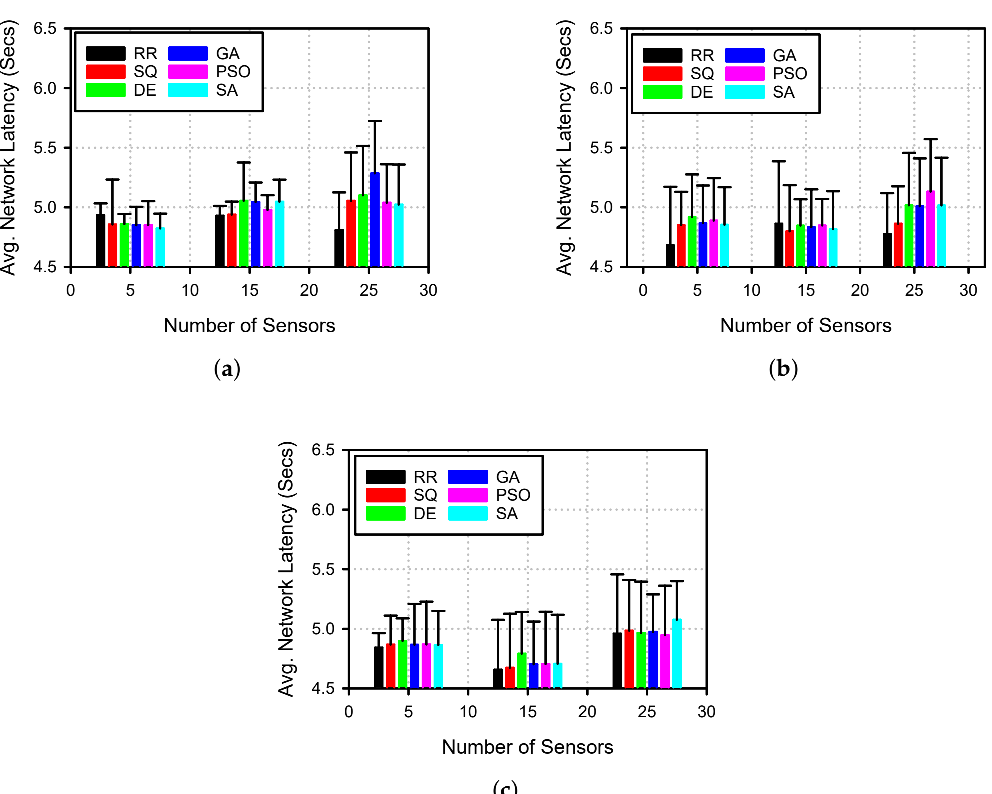

Figure 13. Average network latency comparison of load balancing techniques under homogeneous server configurations with increasing sensor counts. All servers were configured with an equal number of worker threads (two per server). (**a**) Two homogeneous servers (2 worker threads per server). (**b**) Three homogeneous servers (2 worker threads per server). (**c**) Four homogeneous servers (2 worker threads per server).

For the two-server configuration shown in Figure 13a, the average network latency remained relatively stable across increasing sensor counts for all load balancing techniques. Minor variations were observed between algorithms; however, the overall latency differences were small, indicating that under homogeneous conditions with limited server capacity, the choice of load balancing strategy had little impact on performance.

When the number of servers was increased to three, as illustrated in Figure 13b, the observed latency values decreased slightly and exhibited improved stability across sensor counts. The results showed reduced sensitivity to sensor scaling, suggesting that additional homogeneous servers helped distribute the incoming workload more evenly and alleviated processing contention at the edge.

The four-server configuration in Figure 13c consistently yielded the lowest latency values across all sensor counts and load balancing techniques. The latency curves were closely clustered, indicating that when sufficient homogeneous edge resources were available, the system performance became less dependent on the specific load balancing algorithm employed.

The observed trends indicate that under homogeneous server conditions, the dominant factor influencing average network latency was the aggregate processing capacity rather than the specific load balancing heuristic. As servers were scaled proportionally with sensor density, task queues were shortened, parallelism increased, and communication delays were reduced.

From a systems perspective, these results highlight that provisioning adequate and evenly matched edge resources is an effective strategy for maintaining predictable latency performance in dense IoT deployments. While advanced metaheuristic-based load balancing techniques may offer advantages in heterogeneous environments, their relative performance differences diminish when server resources are homogeneous and sufficiently provisioned.

Importantly, these experiments demonstrate the capability of the EDECF to support controlled, reproducible evaluation of load balancing strategies under precisely defined infrastructure conditions. The EDECF enabled simultaneous scaling of sensors and homogeneous edge servers while preserving realistic communication and processing dynamics. This capability allows systematic isolation of infrastructure effects from algorithmic behavior, which is essential for rigorous evaluation of edge computing architectures.

Overall, the results demonstrate that the EDECF provides a flexible and realistic experimental platform for studying the interaction between resource provisioning and load balancing strategies. The framework enables detailed analysis of scalability, performance stability, and algorithm sensitivity under homogeneous edge computing scenarios, thereby supporting informed design decisions for scalable and latency-sensitive edge systems.

6.6.2. Comparison of Load Balancing Techniques Under Heterogeneous Server Conditions

Figure 14a–c present the average network latency obtained when comparing different load balancing techniques under heterogeneous server configurations. In this setting, server heterogeneity was introduced by assigning unequal numbers of worker threads to each server, with two threads for the two-server configuration, three threads for the three-server configuration, and four threads for the four-server configuration. The experiments were conducted by scaling both the number of sensors and the number of servers to evaluate latency behavior under unequal computational capacities.

For the two-server heterogeneous configuration shown in Figure 14a, the average network latency increased noticeably as the number of sensors grew. Greater performance variability was observed across the different load balancing techniques, particularly at higher sensor counts. Metaheuristic-based approaches generally yielded lower latency than simple heuristic methods, indicating their improved ability to adapt to uneven server capacities.

When the number of servers was increased to three, as illustrated in Figure 14b, the observed latency values were reduced and exhibited improved stability across sensor counts. The performance gap between the load balancing techniques narrowed, although differences remained visible, especially under higher load conditions. This behavior suggests that increased processing capacity partially mitigated the impact of server heterogeneity while still allowing algorithmic differences to influence task distribution.

The four-server heterogeneous configuration in Figure 14c consistently produced the lowest latency values across all sensor counts. Latency trends were smoother and more tightly clustered compared to the two- and three-server cases, indicating that increased

https://doi.org/10.3390/ai7030081

computational resources reduced the sensitivity of the system to both sensor scaling and server heterogeneity.

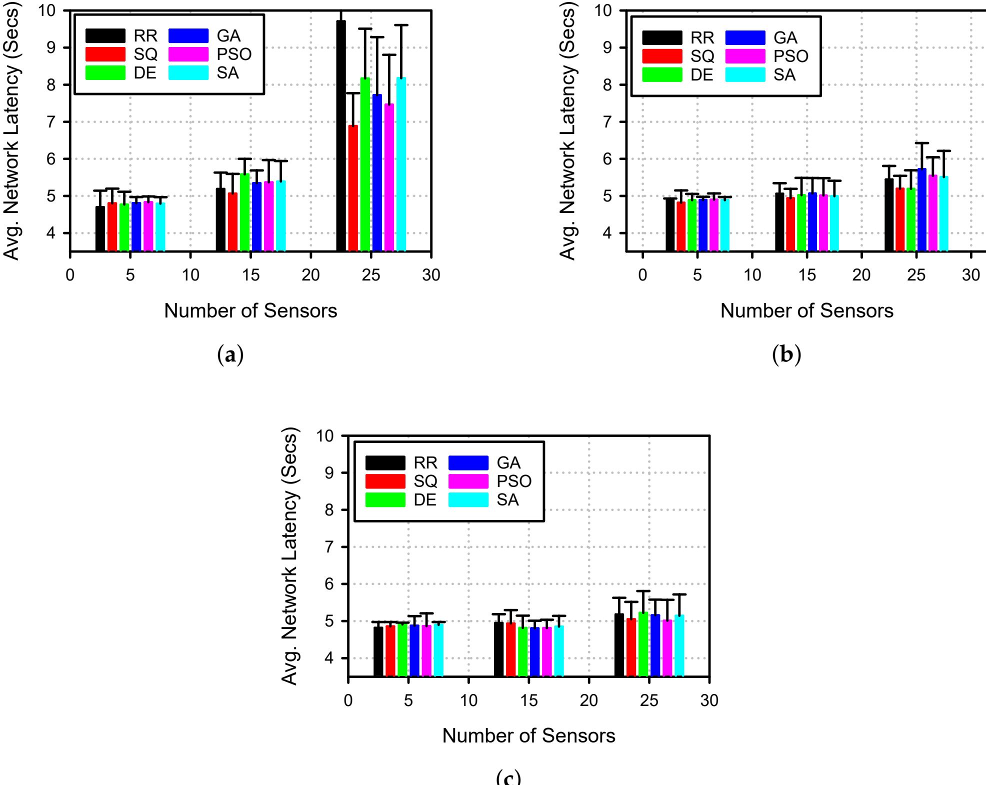

Figure 14. Average network latency comparison of load balancing techniques under heterogeneous server configurations with increasing sensor counts. The heterogeneous condition was defined by unequal server capacities, with two, three, and four worker threads assigned per server for the 2-, 3-, and 4-server configurations, respectively. (**a**) Two heterogeneous servers (2 worker threads per server). (**b**) Three heterogeneous servers (3 worker threads per server). (**c**) Four heterogeneous servers (4 worker threads per server).

The observed results indicate that under heterogeneous server conditions, the effectiveness of load balancing strategies becomes increasingly important, particularly when server capacity is limited. Unequal processing capabilities introduced imbalance in task execution, leading to higher queuing delays when load was not distributed effectively. As server capacity increased, the impact of heterogeneity diminished, and overall latency became more stable.

From a systems perspective, these results highlight that heterogeneous edge environments benefit more strongly from adaptive and intelligence-driven load balancing techniques. While simple heuristics may perform adequately under abundant resources, metaheuristic-based approaches offer improved robustness when computational capacities vary across servers.

Importantly, these experiments demonstrate the capability of the EDECF to support controlled and reproducible evaluation of heterogeneous edge infrastructures. The EDECF enabled precise configuration of server capacities, dynamic scaling of sensors and servers, and realistic emulation of communication and processing behavior. This allowed the direct study of how heterogeneity and resource provisioning interact with load balancing strategies.

Overall, the results demonstrate that the EDECF provides a flexible and realistic experimental platform for investigating the performance of load balancing techniques in heterogeneous edge computing environments. The framework enables systematic

analysis of scalability, algorithm sensitivity, and resource heterogeneity, thereby supporting informed design and optimization of real-world edge computing systems.

7. Limitations and Future Works

This section discusses potential limitations and the measures taken to mitigate them alongside future works.

7.1. Internal Validity

Internal validity concerns whether the observed performance differences are attributable to the evaluated load balancing strategies rather than experimental artifacts. To mitigate this risk, all algorithms, including heuristic and metaheuristic approaches, were evaluated under identical emulated conditions using the same network topology, workload parameters, and system configurations. Furthermore, to account for the stochastic nature of metaheuristic algorithms, experiments were conducted over multiple independent runs, and mean values with error bars were reported. This reduces the likelihood that results are influenced by random initialization or single-run bias.

7.2. External Validity

External validity relates to the generalizability of the findings beyond the specific experimental setup. The experiments were executed on a single physical machine hosting the emulation environment, which may limit direct extrapolation of absolute performance metrics to large-scale real-world deployments. However, the primary objective of this work is not raw performance benchmarking, but rather the validation of the emulated dynamic edge computing framework (EDECF) as a realistic and reproducible evaluation platform. The framework is platform-agnostic and can be deployed on more powerful or distributed server infrastructures without modification. Therefore, while absolute performance values may vary across hardware platforms, the observed comparative trends and framework behavior are expected to generalize.

7.3. Construct Validity

Construct validity concerns whether the selected evaluation metrics adequately capture the intended system properties. In this study, latency and system behavior under dynamic conditions were selected as evaluation metrics, as they are widely used indicators of quality of service in edge computing environments. While additional metrics such as energy consumption or deadline–miss ratio could provide further insights, the chosen metrics sufficiently capture the framework's ability to support queue-aware and dynamic load balancing evaluation.

7.4. Reproducibility

To ensure transparency and reproducibility, the hardware configuration, software environment, and key experimental parameters are explicitly reported in the manuscript. All required installation scripts, configuration files, and experimental artifacts are available upon request from the corresponding author. Public release of these materials is kept under the current name of the platform: "emulated dynamic edge computing framework (EDECF)".

7.5. Model Extensions

In addition, the workload models considered in this study are intentionally designed to support controlled framework validation rather than exhaustive workload realism. While the current evaluation captures heterogeneous server capacities and dynamic sensor behavior, more complex workload characteristics such as bursty task arrivals, task deadlines, and

deadline–miss ratio are not explicitly modeled. These extensions can be naturally incorporated within the sensor generation characteristics of the EDECF, which can extend the study of real-time and deadline-constrained edge applications. Furthermore, energy consumption modeling is identified as part of future work, as it requires hardware-specific power models that are beyond the current emulation scope. Most importantly, their inclusion is left for future work, as the primary objective of this study is to validate the framework's ability to support realistic, queue-aware, and dynamic evaluation of load balancing strategies. At the moment, we note that all possible workload dimensions have not been explored, which naturally leaves room for feasible extensions of the framework in future works.

8. Conclusions

This paper has presented an emulated dynamic edge computing framework (EDECF) as a scalable and realistic emulation platform for the systematic design, implementation, and evaluation of load balancing solutions in edge computing networks. Built on CORE and integrated with Zenoh-enabled communication, the EDECF enables near-realistic experimentation by supporting dynamic sensor populations, scalable multi-LAN and WLAN topologies, and configurable edge server infrastructures under both homogeneous and heterogeneous resource conditions.

Through extensive experimental evaluation, the EDECF has been shown to reliably capture the effects of sensor scaling, server scaling, and resource heterogeneity on average network latency and performance variability. The results demonstrated that network latency increases with sensor density but can be effectively mitigated through proportional scaling of edge servers, highlighting the importance of joint sensing–compute provisioning in edge deployments. Under homogeneous server conditions, performance differences among load balancing strategies were relatively small when sufficient resources were available, indicating that aggregate processing capacity dominated system behavior. In contrast, under heterogeneous server configurations, algorithmic choice plays a more important role, with metaheuristic-based approaches exhibiting improved robustness and stability compared to simple heuristic methods.

The comparative analysis of metaheuristic-based load balancing techniques further showcased the EDECF's algorithm-agnostic nature, thus allowing for different methods to be evaluated, such as the particle swarm optimization, genetic algorithm, differential evolution and simulated annealing techniques. The parameter sensitivity studies conducted using the EDECF revealed that increasing population size and iteration count beyond moderate ranges yielded diminishing returns while increasing computational overhead, which emphasized the need to balance optimization quality with real-time responsiveness in edge environments.

Beyond algorithmic performance, the results collectively demonstrated the EDECF's capability to support controlled, repeatable, and fine-grained experimentation across a wide range of deployment scenarios. The framework enabled isolation of infrastructure effects from algorithmic behavior, captured both mean performance and variability, and supported realistic stress testing under dynamic workloads and scaling conditions.

Overall, the EDECF provides a credible and flexible experimental foundation for evaluating metaheuristic-based load balancing and other edge intelligence solutions. By bridging the gap between abstract simulation and costly hardware testbeds, the EDECF facilitates reproducible performance analysis and informed design decisions for scalable, latency-sensitive edge computing systems.

Author Contributions: Conceptualization, D.N.M. and A.J.O.; funding acquisition, A.M.A.-M.; investigation, D.N.M.; methodology, D.N.M.; project administration, A.J.O. and A.M.A.-M.; supervision,

A.J.O. and A.M.A.-M.; writing—original draft, D.N.M.; writing—review and editing, A.J.O. and A.M.A.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Council for Scientific and Industrial Research (CSIR) under project number 05400 054AT KR6EDNP.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Thai, M.T.; Lin, Y.D.; Lai, Y.C.; Chien, H.T. Workload and capacity optimization for cloud-edge computing systems with vertical and horizontal offloading. *IEEE Trans. Netw. Serv. Manag.* **2019**, *17*, 227–238. [CrossRef]
2. Bi, J.; Yuan, H.; Duanmu, S.; Zhou, M.; Abusorrah, A. Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization. *IEEE Internet Things J.* **2020**, *8*, 3774–3785. [CrossRef]
3. Wang, B.; Cheng, J.; Cao, J.; Wang, C.; Huang, W. Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve SLA satisfaction. *PeerJ Comput. Sci.* **2022**, *8*, e893. [CrossRef]
4. Zhao, L.; Li, T.; Zhang, E.; Lin, Y.; Wan, S.; Hawbani, A.; Guizani, M. Adaptive swarm intelligent offloading based on digital twin-assisted prediction in VEC. *IEEE Trans. Mob. Comput.* **2023**, *23*, 8158–8174. [CrossRef]
5. Hoang, T.H.; Nguyen, C.T.; Do, T.N.; Kaddoum, G. Joint task offloading and radio resource management in stochastic MEC systems. *IEEE Trans. Commun.* **2024**, *72*, 2670–2686. [CrossRef]
6. Feng, J.; Liu, Z.; Wu, C.; Ji, Y. AVE: Autonomous vehicular edge computing framework with ACO-based scheduling. *IEEE Trans. Veh. Technol.* **2017**, *66*, 10660–10675. [CrossRef]
7. Yin, L.; Sun, J.; Zhou, J.; Gu, Z.; Li, K. ECFA: An efficient convergent firefly algorithm for solving task scheduling problems in cloud-edge computing. *IEEE Trans. Serv. Comput.* **2023**, *16*, 3280–3293. [CrossRef]
8. Papathanail, G.; Fotoglou, I.; Demertzis, C.; Pentelas, A.; Sgouromitis, K.; Papadimitriou, P.; Spatharakis, D.; Dimolitsas, I.; Dechouniotis, D.; Papavassiliou, S. COSMOS: An orchestration framework for smart computation offloading in edge clouds. In *Proceedings of the NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020*; IEEE: New York, NY, USA, 2020; pp. 1–6. [CrossRef]
9. de Souza, A.B.; Rego, P.A.L.; Chamola, V.; Carneiro, T.; Rocha, P.H.G.; de Souza, J.N. A bee colony-based algorithm for task offloading in vehicular edge computing. *IEEE Syst. J.* **2023**, *17*, 4165–4176. [CrossRef]
10. Wang, Y.; Yang, S.; Ren, X.; Zhao, P.; Zhao, C.; Yang, X. IndustEdge: A time-sensitive networking enabled edge-cloud collaborative intelligent platform for smart industry. *IEEE Trans. Ind. Inform.* **2021**, *18*, 2386–2398. [CrossRef]
11. Yu, Z.; Chen, Y.; Leng, J.; Ji, H.; Zheng, L.; Zou, Z. SAIndust: A Self-Aware Heterogeneous Computing Framework for Industrial Internet of Things. *IEEE Internet Things J.* **2025**, *12*, 28776–28792. [CrossRef]
12. Zhang, J.; Yu, X.; Ha, S.; Peña Queralta, J.; Westerlund, T. Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ros 2 systems. *J. Intell. Robot. Syst.* **2024**, *110*, 162. [CrossRef]
13. Ning, Z.; Dong, P.; Kong, X.; Xia, F. A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things. *IEEE Internet Things J.* **2018**, *6*, 4804–4814. [CrossRef]
14. Wang, Y.; Tao, X.; Zhang, X.; Zhang, P.; Hou, Y.T. Cooperative task offloading in three-tier mobile computing networks: An ADMM framework. *IEEE Trans. Veh. Technol.* **2019**, *68*, 2763–2776. [CrossRef]
15. Bi, J.; Wang, Z.; Yuan, H.; Zhang, J.; Zhou, M. Cost-Minimized computation offloading and user association in hybrid cloud and edge computing. *IEEE Internet Things J.* **2024**, *11*, 16672–16683. [CrossRef]
16. Nguyen, L.X.; Tun, Y.K.; Dang, T.N.; Park, Y.M.; Han, Z.; Hong, C.S. Dependency tasks offloading and communication resource allocation in collaborative UAVs networks: A meta-heuristic approach. *IEEE Internet Things J.* **2023**, *10*, 9062–9076. [CrossRef]
17. Chen, X.; Mao, Y.; Wang, H.; Xu, Y.; Li, D.; Liu, S.; Zhao, X. Data-driven task offloading method for resource-constrained terminals via unified resource model. *IEEE Internet Things J.* **2023**, *10*, 9703–9715. [CrossRef]
18. Mavromatis, A.; Colman-Meixner, C.; Silva, A.P.; Vasilakos, X.; Nejabati, R.; Simeonidou, D. A software-defined IoT device management framework for edge and cloud computing. *IEEE Internet Things J.* **2019**, *7*, 1718–1735. [CrossRef]

19. Mayer, R.; Graser, L.; Gupta, H.; Saurez, E.; Ramachandran, U. Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In *Proceedings of the 2017 IEEE Fog World Congress (FWC), Santa Clara, CA, USA, 30 October–1 November 2017*; IEEE: New York, NY, USA, 2017; pp. 1–6. [CrossRef]
20. Spatharakis, D.; Dimolitsas, I.; Dechouniotis, D.; Papathanail, G.; Fotoglou, I.; Papadimitriou, P.; Papavassiliou, S. A scalable edge computing architecture enabling smart offloading for location based services. *Pervasive Mob. Comput.* **2020**, *67*, 101217. [CrossRef]
21. Molokomme, D.N.; Onumanyi, A.J.; Abu-Mahfouz, A.M. Hybrid metaheuristic schemes with different configurations and feedback mechanisms for optimal clustering applications. *Clust. Comput.* **2024**, 27, 8865–8887. [CrossRef]
22. Corsaro, A.; Cominardi, L.; Hecart, O.; Baldoni, G.; Avital, J.E.P.; Loudet, J.; Guimares, C.; Ilyin, M.; Bannov, D. Zenoh: Unifying communication, storage and computation from the cloud to the microcontroller. In *Proceedings of the 2023 26th Euromicro Conference on Digital System Design (DSD), Golem, Albania, 6–8 September 2023*; IEEE: New York, NY, USA, 2023; pp. 422–428. [CrossRef]
23. Chen, X.; Wen, H.; Ni, W.; Zhang, S.; Wang, X.; Xu, S.; Pei, Q. Distributed online optimization of edge computing with mixed power supply of renewable energy and smart grid. *IEEE Trans. Commun.* **2021**, *70*, 389–403. [CrossRef]
24. Ahrenholz, J.; Danilov, C.; Henderson, T.R.; Kim, J.H. CORE: A real-time network emulator. In *Proceedings of the MILCOM 2008-2008 IEEE Military Communications Conference, San Diego, CA, USA, 16–19 November 2008*; IEEE: New York, NY, USA, 2008; pp. 1–7. [CrossRef]
25. Baumgärtner, L.; Meuser, T.; Bloessl, B. coreemu-lab: An automated network emulation and evaluation environment. In *Proceedings of the 2021 IEEE Global Humanitarian Technology Conference (GHTC), Seattle, WA, USA, 19–23 October 2021*; IEEE: New York, NY, USA, 2021; pp. 304–311. [CrossRef]
26. Garcia-Carballeira, F.; Calderon, A.; Carretero, J. Enhancing the power of two choices load balancing algorithm using round robin policy. *Clust. Comput.* **2021**, *24*, 611–624. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.