

Congestion-Game-Inspired Server Selection for Edge DNN Inference

Mateus S. Gilbert¹, Nicolas Peugnet², Anne Fladenmuller³, Kinda Khawam⁴,
Marcello L. R. de Campos⁵, Miguel Elias M. Campista⁶, and Marcelo Dias de Amorim⁷

Abstract—Edge computing has emerged as a critical paradigm for delivering deep neural network (DNN)-based services to end users in networks with resource-constrained Internet of things (IoT) devices. This approach provides lower latency and higher reliability than cloud deployment, while mitigating the limitations of running DNNs locally on constrained devices. Deploying multiple DNN-hosting servers further increases service resilience and availability through redundancy. However, selecting the optimal DNN in heterogeneous wireless networks can be challenging, particularly when nodes are highly mobile and frequently disconnected, which is exacerbated if nodes compete for the best DNN and are uncooperative. This presents a scenario well-suited for game-theoretic solutions. Motivated by this, we introduce a decentralised DNN server-selection framework that is resilient to node mobility and network partitions. The framework integrates a non-cooperative congestion-game-inspired selection algorithm with a shared data space via the Matrix protocol, enabling end users to estimate server load within their network partition and select the DNN that best meets their service-quality and latency requirements. Simulation-based evaluation demonstrates that the proposed system outperforms baseline strategies that always select either the server with the highest-performing DNN, the one hosted in the closest server, as well as the best DNN that strictly satisfies the client's time constraints. Using both synthetic and real-world mobility traces, we show that the system is resilient to user mobility and network partitioning, adapting to server overload and dynamic network conditions.

Index Terms—edge computing, mobile wireless networks, deep neural network, congestion games.

I. INTRODUCTION

IN the past decade, solutions that resort to deep neural network (DNN) have become increasingly prevalent in

Manuscript created in June 2026. This work was supported by CNPq (310234/2025-5, 407304/2025-8, 408255/2023-4, 312145/2023-3 and 405940/2022-0), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Funding Code 001 and 88887.954253/2024-00, FAPERJ (E-26/200.438/2026, E-26/200.363/2026 and E-26/210.778/2025), FAPESP (2023/00673-7 and 2023/00811-0), Fundação de Desenvolvimento da Pesquisa (Fundep) Rota 2030, Stellantis, Mobway and Bpifrance IE6 - France 2030 “Telecom 5G”. (Corresponding author: Mateus S. Gilbert)

Mateus S. Gilbert is with the Grupo de Teleinformática e Automação, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, and also LIP6, CNRS, Sorbonne Université, Paris, France. (e-mail: gilbert@gt.a.ufrj.br)

Nicolas Peugnet, Anne Fladenmuller and Marcelo Dias de Amorim are with LIP6, CNRS, Sorbonne Université, Paris, France. (e-mail: nicolas.peugnet@lip6.fr; anne.fladenmuller@lip6.fr; marcelo.amorim@lip6.fr)

Kinda Khawam is with the Université Paris-Saclay, Orsay, France. (e-mail: kinda.khawam@universite-paris-saclay.fr)

Marcello L. R. de Campos is with the Laboratório de Sinais Multimídia e Telecomunicações, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil. (e-mail: campos@smt.ufrj.br)

Miguel Elias M. Campista is with the Grupo de Teleinformática e Automação, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil. (e-mail: miguel@gt.a.ufrj.br)

Internet of things (IoT) and mobile wireless networks, either to support these networks or as part of the services they provide [1, 2]. The effectiveness of a DNN often improves with its size, but this also imposes implementation costs that may overwhelm resource-constrained devices [2]. Hence, developers commonly deploy DNNs in the cloud, enabling users to leverage the cloud's resources [2]. However, these implementations require constant connectivity to cloud infrastructure. In emergency scenarios – such as natural disasters, large-scale accidents, or network outages – access to the core network may become intermittent or unavailable [3]. In such cases, first responders who could benefit from these DNN-powered applications may experience impaired field operations. These scenarios require timely responses, making the data transmission between end users and cloud servers prohibitive for latency-sensitive applications [2, 4].

Edge computing offers an alternative that minimises communication latency by deploying infrastructure capable of running the DNN closer to end users [2], albeit usually more limited than cloud infrastructure. Additionally, multiple nodes hosting similar DNNs can be deployed at the edge to allow end users to switch between servers as connectivity changes, reducing latency and improving service continuity [5]. However, node mobility and heterogeneity can pose a challenge for end users when selecting a server to outsource their inference requests. DNN performance varies across servers because edge nodes have heterogeneous computational resources [6], especially when these nodes host different models. Besides, the load on each server can impact the end user's experience. For instance, if the server hosting the best DNN (e.g., the most accurate) is distant or overloaded, an end user may switch to a server with a less accurate DNN to avoid delayed responses. This motivates the development of server-selection algorithms that take all these factors into account.

Given this scenario, this work addresses the following question: *how can resource-constrained end users, acting as clients of a DNN application, select the server that provides the best service quality under strict latency constraints?* More specifically, we seek a solution that explicitly accounts for the presence and behaviour of other users sharing the network, while being resilient to node mobility and potential network partitioning. The resulting system should ensure high-quality DNN service delivery and strong reliability, particularly in critical situations.

A. Background: Game Theory for Networking Problems

Numerous studies in IoT and mobile wireless networks implement decentralised approaches to mitigate computational complexity, latency, and fault tolerance challenges in resource allocation and service provision for distributed, competing end users [7]–[15]. For instance, Cohen et al. introduce a framework for edge-cloud multi-tier networks where servers autonomously allocate end-user requests using a job-migration mechanism within a hierarchical, tree-structured network topology [14], according to user mobility and request criticality. However, this approach presumes a stable hierarchy, which may not exist in partition-prone networks where server paths change frequently. The problem considered here also involves clients who maintain preference rankings over available servers based on DNN accuracy. This restricts such migration mechanisms, as these clients may decline the service from a server hosting a DNN they perceive as less accurate.

Game theory provides an effective framework for analysing the behaviour of rational and competing agents. For example, Picano et al. present an end-user allocation method for edge servers hosting large language models (LLMs) using matching games [15]. In this approach, users are assigned to servers based on expected response time and deadline length until all users are allocated. In another work, Gao et al. use potential games to model computation offloading in low-Earth-orbit (LEO) satellite networks to minimise end-user overhead [12]. The proposed model accommodates the dynamic environment of these networks, which stems from satellite movement and communication interferences [12]. This adaptability, combined with the lack of direct user interaction, aligns well with the requirements of the server-selection problem addressed in this study, leading us to congestion games previously used in many networking problems, including load balancing, routing, and scheduling [16]–[19].

Building on this literature, we model the server-selection problem as a congestion game, in which servers act as shared resources whose latency and reliability degrade as user load increases. The proposed formulation captures the impact of congestion on the risk of missed inference requests, while naturally favouring servers that offer higher inference accuracy.

B. Proposal: Selecting Servers using Congestion Games

We propose two server-selection algorithms based on a congestion game formulation. Within this framework, resource-constrained end users select the server that minimises a cost function capturing both expected latency and DNN inference accuracy. This enables end users to choose a suitable server that provides a good balance between minimising job losses and maximising DNN service performance. We also provide a first implementation of the proposed selection game by introducing a node-communication policy based on the Matrix protocol [20] to create a persistent shared data space where servers disseminate their current load information. This mechanism ensures that users can make up-to-date decisions reflecting the state of their network partition. The result is a *server-selection system for mobile resource-constrained end*

users running latency-sensitive DNN applications, designed to operate under network partitions.

We evaluate the proposed server-selection system using *Meshsim* (a dynamic wireless mesh network (WMN) emulator [21]) with both synthetic and real-world WMN traces [22], in which DNN-hosting nodes simulate different Raspberry Pis (RPis). We compare the proposed algorithm against (1) a greedy strategy that always selects the best reachable DNN, (2) a risk-averse strategy that always chooses the server with the lowest round-trip time (RTT), (3) a strategy that selects servers randomly, and (4) a greedy strategy that selects the best DNN whose server strictly satisfies the end user's time constraints. We also compare it against running all inference requests locally, which serves as the baseline for all selection strategies. We measure the DNN performance using the ImageNet's image classification dataset [23] to reflect the quality of delivered service.

The results with the synthetic traces show that the proposed congestion-game-based server-selection algorithms achieved the largest improvement in accuracy among the analysed strategies in most cases and are more robust to errors in network state estimation. In particular, the results show that the proposed algorithms remain effective without introducing a tolerance parameter to account for these errors. Finally, when using the real-world traces, in which clients perform local inference when interactions with the selected server fail, clients using the proposed game-based algorithms achieved the best trade-off between accuracy gains and overdue minimisation.

In summary, this paper's contributions are:

- We model the server-selection problem in a mobile wireless network with resource-constrained devices as a congestion game that we name the server-selection game (SSG).
- We provide a theoretical analysis of the convergence and equilibrium characteristics of the proposed SSG.
- We propose two server-selection algorithms for selecting an edge DNN that account for both model performance and end-user latency constraints.
- We propose a persistent shared data space in which servers share their current usage information with all end users in a decentralised manner to support the proposed algorithms
- We present a first version of the proposed SSG-based system, combining it with a shared data space implemented using the Matrix protocol [20], yielding a server-selection system suitable for DNN selection at the edge that is resilient to node mobility and network partitions.

II. DNNs IN DYNAMIC IOT NETWORKS AT THE EDGE

Edge computing is a paradigm that relocates computational resources closer to the end devices [24]. With this approach, resource-rich servers, albeit less powerful than cloud servers, can be deployed to support constrained end users, providing access to DNN models that cannot be stored locally. Still, model choice and placement should account for the inherent computing, storage, energy, and communication limitations of both edge and end users [2].

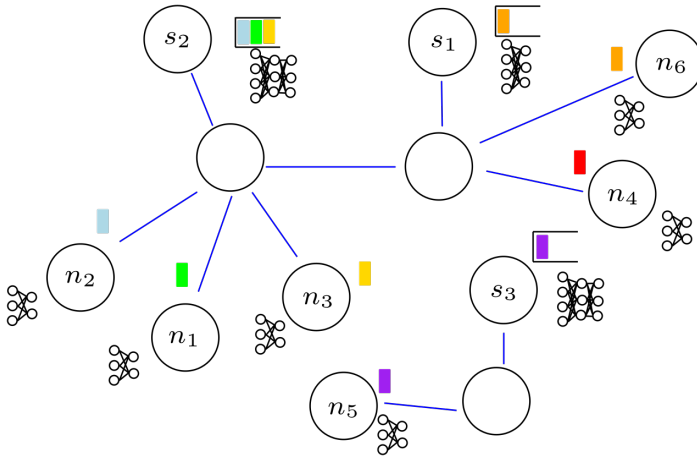


Figure 1. Illustration of the server-selection problem. Blue edges indicate node connectivity. Expected response time increases as more clients (n_i nodes) connect to the same server (s_i). Colours differentiate the inference request from each client. Unlabelled nodes are routers.

The positioning of DNN-hosting nodes can be done strategically to address different challenges. In dense IoT networks with numerous resource-constrained devices requesting the same DNN service, deploying multiple nearby servers enhances redundancy and prevents client overload in a single node. Distributing servers that offer the same DNN application increases the service's availability in networks where end users are mobile, and connections may be unreliable or prone to degradation. These redundancies are helpful even when cloud support is available, as they mitigate service disruptions that can occur during cloud connectivity losses.

A. Edge Heterogeneity and DNN Performance

The inherent node heterogeneity of edge wireless networks means that end users who want to benefit from a DNN application (hereinafter referred to as clients) can find servers offering different levels of accuracy and processing capacity. Node resources can impact DNNs' real-time performance, especially if equipped with hardware not optimised for DNN processing [25]. In these devices, CPU clock speed impacts instruction throughput, RAM size is crucial for accommodating model parameters and activations, and CPU caches can reduce costly memory access. Thus, higher resource availability directly translates into reduced inference latency. Client-server distance and the number of pending requests also impact the expected response time of a client's inference request. Hence, clients should account for the network's changing conditions and the selection behaviour of other clients.

B. Which DNN Server Should Be Selected?

The server-selection problem addressed in this work can be summarised as follows. Resource-constrained clients in a wireless edge network have lightweight DNNs and access to more powerful DNN-hosting servers. Each client seeks to maximise service quality (by maximising DNN accuracy) while meeting its latency constraints. Ideally, these clients want to outsource inference requests to a server; however, they can use their less accurate local DNN if no server satisfies their

time constraints. Due to client mobility and dynamic network conditions, clients may need to switch servers, which in turn impacts server load. Thus, we require a server-selection system that accounts for each client's individual selection and remains functional under node mobility and network partitions.

Figure 1 illustrates the problem at hand, where n_i and s_i represent the clients and servers, respectively, and the DNN sizes represent the model complexity. In particular, the larger DNNs is assumed to be more accurate. Suppose client n_4 prefers s_2 and s_3 to s_1 because the first two have the most accurate DNNs. It cannot select s_3 because it belongs to a different network partition from n_4 . Next, the response time of s_2 may be high for n_4 due to the distance between them and the s_2 's current load, which is serving three other clients. In this scenario, n_4 might prefer s_1 because it is closer and less requested, minimising the risks of overdue inferences. Finally, it can fall back to its local DNN when it loses connectivity to s_1 and s_2 , or when their service times exceed its latency constraints, to preserve service continuity.

Given these characteristics, we formulate the server-selection problem for client n_i as follows. Let d_j and q_j denote, respectively, the processing time and service quality (e.g., the accuracy of the hosted DNN) of server s_j . Let l_j^i represent the round-trip communication latency between client n_i and server s_j , and let τ_i denote the latency constraint of client n_i . Within this context, the objective of n_i is:

$$\max_{s_j} q_j \quad \text{s.t.} \quad d_j + l_j^i < \tau_i. \quad (1)$$

While selecting the best s_j for a single client n_i is a straightforward optimisation task, finding an optimal solution in real time for every client in a dynamic, partition-prone network is complex. This work addresses this challenging problem.

III. DEVISING A SERVER SELECTION STRATEGY

A key aspect of the server-selection problem described in Section II is that clients select a server without cooperating with each other. Game theory provides a solid framework for analysing the decisions of rational, competing agents (players), enabling predictions of how they adapt to competitive scenarios. Within game theory, congestion games [26] present similarities with our server-selection problem.

A congestion game can be defined as follows:

Definition 1. (Adapted from [26]) A congestion game is a tuple $(\mathcal{N}, \mathcal{R}, (\Sigma_i)_{i \in \mathcal{N}}, (d_r)_{r \in \mathcal{R}})$, where $\mathcal{N}$ is the set of players, $\mathcal{R}$ is the set of resources, $\Sigma_i \subseteq 2^{\mathcal{R}}$ denotes the strategy space of player i ¹, and $d_r : \mathbb{N} \rightarrow \mathbb{R}_+$ is the cost function associated with resource r .

A state $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_{|\mathcal{N}|}) \in \Sigma_1 \times \Sigma_2 \times \dots \times \Sigma_{|\mathcal{N}|}$ represents a strategy profile, where each σ_i specifies the set of resources selected by player i . The number of players using resource r at state σ , referred to as the congestion on resource r , is $v_r(\sigma) = |\{i \in \mathcal{N} \mid r \in \sigma_i\}|$. Accordingly, the

¹ $2^{\mathcal{R}}$ is the power set of $\mathcal{R}$, the set of all subsets of $\mathcal{R}$.

cost of resource r at state σ is $d_r(v_r(\sigma))$, and the total cost experienced by player i is

$$\sum_{r \in \sigma_i} d_r(v_r(\sigma)).$$

Given a state σ , an improvement step for player i consists of changing its strategy from σ_i to some $\sigma'_i \in \Sigma_i$ so that its individual cost strictly decreases. In the finite case, congestion games admit at least one pure-strategy Nash equilibrium (PSNE), and any sequence of strict improvement steps is finite and therefore terminates at such an equilibrium [26].

When modelling the server-selection problem as a congestion game, DNN-hosting servers are the resources, and clients are the players. In this setting, the cost of using a server reflects its load-dependent performance, given by inference latency and queueing time. This is consistent with prior work in networking, where congestion games are used to model latency-sensitive resource allocation problems [16, 17, 19]. However, our formulation differs from the standard congestion game in Definition 1. The cost perceived by a client is not solely determined by the resource's congestion level, but also depends on client-specific factors and additional system constraints. As discussed in Section II, the cost must incorporate client-server communication latency, clients' latency constraints, and the servers' service quality. These extensions naturally lead to a player-specific cost structure. In contrast to standard congestion games, player-specific congestion games generally do not admit an exact potential function, and best-response dynamics are not guaranteed to converge to a Nash equilibrium (NE) [27].

A. Total Server Response Delay

Player-specific congestion games are a generalisation of the standard congestion game where each player i has its own cost function d_r^i for each resource r [28]. This individualised cost can be used in load-balancing applications to rank player requests by priority, assigning a higher cost to low-priority requests. Similarly, considering l_j^i as the round-trip time (RTT) between client n_i and server s_j , we define

$$d_j^i(v_r(\sigma)) = d_j(v_r(\sigma)) + l_j^i \quad (2)$$

to be the player-specific cost with which it can estimate the expected response time from every server. We set $l_j^i = \infty$ when n_i and s_j belong to distinct partitions, discouraging n_i from selecting unreachable servers².

B. Avoiding Overloaded and Distant Servers

We incorporate the restriction from Equation 1 into our congestion-game model by proposing a punishment function ϕ_i that asymptotically grows to infinity when $d_j^i \rightarrow \tau_i$. This enables client n_i to adopt a congestion-based analysis of how close d_j^i is to its time restriction. Specifically, we define

$$\phi_i(d_j^i) = \begin{cases} \frac{\tau_i}{\tau_i - d_j^i}, & \text{if } \tau_i > d_j^i, \\ \infty, & \text{otherwise.} \end{cases} \quad (3)$$

²Hereinafter, d_j and d_j^i will sometimes be used in place of $d_j(v_r(\sigma))$ and $d_j^i(v_r(\sigma))$, respectively, to simplify notation.

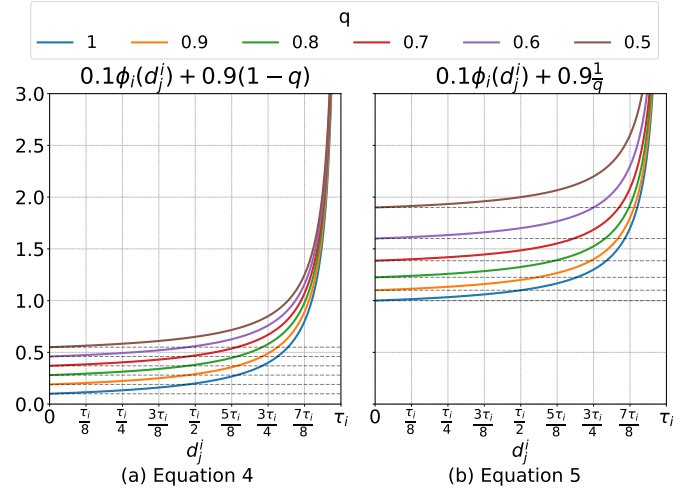


Figure 2. Comparison between the two proposed cost functions and the impact of the service quality q .

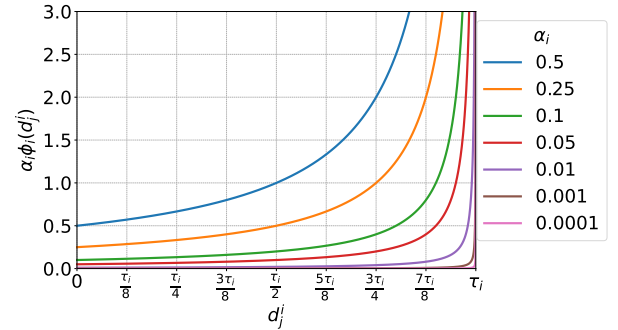


Figure 3. Impact of α_i on Equation 3

This punishment function can encourage clients to adopt a more risk-averse approach, allowing them to switch from s_j to other servers when d_j^i approaches τ_i .

C. Taking DNN Performance into Account

We define a service-quality function $q : S \rightarrow [0, 1]$ that enables the ranking of servers in S based on their DNNs. In our experiments, q corresponds to the inference accuracy of the DNN. We then construct a player-specific cost function related to the objective described in Equation 1 by performing a convex combination of this quality metric and Equation 3, following approaches used in congestion games with mixed objectives [17]. $\alpha_i \in [0, 1]$ denotes the preference of client n_i for minimising the penalty term versus maximising service quality. We define

$$c_j^i(\sigma) = \alpha_i \phi(d_j^i) + (1 - \alpha_i)(1 - q(s_j)), \text{ and} \quad (4)$$

$$\hat{c}_j^i(\sigma) = \alpha_i \phi(d_j^i) + (1 - \alpha_i) \frac{1}{q(s_j)} \quad (5)$$

as two possible cost functions.

Figure 2 shows the differences between Equations 4 and 5, according to the impact of the preference variable α_i . Subfigures 2(a) and 2(b) show that the service quality of each server functions as an offset, reducing the impact of the punishment function in servers with better DNNs. Additionally, we can see that it is harder to transition to worse DNNs when using Equation 5 than with Equation 4.

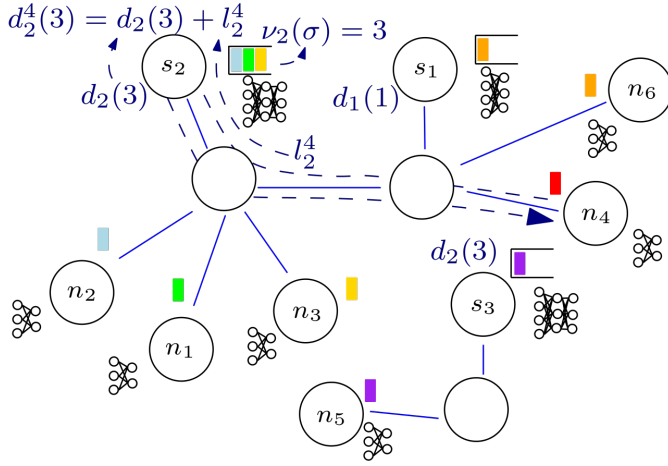


Figure 4. Complete game-theoretic modelling of the server-selection problem, with operationally relevant additions. Client nodes using congestion games and other games derived from the first to select a server to outsource DNN inference or to run it locally.

In Figure 3 we can see that α_i also controls how fast the punishment function tends to ∞ when $d_j^i \rightarrow \tau_i$. In particular, we can see that Equations 4 and 5 are approximately equal to $q(s_j)$ for all $d_j^i < \tau_i$ when $\alpha_i \rightarrow 0$. For infinitesimal α_i , clients using the resulting functions remove only the servers for which $d_j^i \geq \tau_i$ and select the most accurate DNN among them. This is exactly what is described in Equation 1. However, as will be seen in the upcoming experiments, this strategy may perform worse compared to a more risk-averse α_i (i.e. when $\alpha_i \neq 0$).

D. A Server-Selection Game

Figure 4 illustrates our game-theoretical model to solve the server-selection problem in Equation 1, showing how each cost is measured. We name our proposed player-specific congestion game as server-selection game (SSG), defined as follows:

Definition 2. *The SSG is a player-specific congestion game in which the players are resource-constrained client nodes $n_i \in \mathcal{N}$ that must select one server $s_j \in \mathcal{S}$ or itself (local computation) to execute a specific task. In this game, the set of resources $\mathcal{R} = \mathcal{S} \cup \mathcal{N}$, and the strategy space of each player n_i consists of singletons that represent the decisions of n_i selecting a server $s_j \in \mathcal{S}$ or itself. All resources in $\mathcal{R}$ can be ordered using a service-quality function $q : \mathcal{R} \rightarrow [0, 1]$, and $\forall n_i \in \mathcal{N}, \forall s_j \in \mathcal{S}$ we have $q(n_i) < q(s_j)$. Finally, the cost function for using resource s_j is given by c_j^i , as stated in (4) or (5).*

In particular, we can view the filtering approach discussed previously as a special case of Definition 2 with $\alpha_i \approx 0$. Intuitively, using Figure 2, we can see that the transition points (indicated by the dashed horizontal lines) will occur near τ_i when α_i is infinitesimal. Hence, this is equivalent to downgrading to a worse server only when a better option has $d_j^i \geq \tau_i$.

SSG is a finite game because the sets of clients and servers are finite. While the existence of mixed NE is guaranteed, such equilibria are often difficult to interpret and implement in

practical systems [29, Section 2.2]. To address this limitation, we propose heuristic solutions to enable the convergence of the system where self-interested clients operate under incomplete and asymmetric information in the following sections. Additionally, we identify a practically relevant restricted game for which the existence of a pure Nash equilibrium (PNE) is guaranteed. In particular, for this restricted SSG, best-response dynamics are proven to converge to a PNE.

IV. SERVER-SELECTION ALGORITHMS

In our experiments, we assume that clients generate requests according to a Poisson process with rate λ . Similarly, we assume that servers process data according to a Poisson process, with mean service times (denoted T_j) being advertised periodically. In particular, we assume s_j serves one request at a time, as a new DNN inference does not begin until the previous one is completed. This is a reasonable assumption for the resource-constrained scenario studied, as it allows requests to be handled immediately upon arrival, reduces the memory footprint of the DNN application given that batched inferences require more memory, and limits variations in on-device inference latency [25]. Finally, we assume that the mean client-server transmission latencies are constant while client n_i makes a new server choice. These assumptions allow us to model the server's processing behaviour as an M/M/1 queue, allowing us to compute d_j as the expected response time from server s_j (consider $\mu_j = \frac{1}{T_j}$) [30]:

$$d_j(\sigma) = \frac{1}{\mu_j - v_j(\sigma)\lambda}, \quad (6)$$

where $v_j(\sigma)\lambda$ is the arrival rate in server s_j at state σ . This comes from the fact that the superposition of Poisson processes is a Poisson process whose rate is the sum of the rates of the originating processes [31].

These assumptions simplify the client's online analysis of the network state over short time intervals and enable minimal convergence guarantees in the studied dynamic environment. However, they may not capture the complete network state. To this end, we will discuss implementation considerations addressing these limitations in the next section. The upcoming results show that these assumptions were sufficient for clients to efficiently select servers and reduce deadline violations.

A. SSG-derived Algorithm

Whenever a client n_i has a new sample to be processed by a DNN, it selects one $r \in \mathcal{S} \cup \{n_i\}$. In particular, we assume that clients apply informed best-response dynamics, in which each client iteratively selects the strategy that minimises its individual cost (Equations 4 or 5). This selection decision takes into account a set of synchronised information about each server, which is updated whenever a client joins or leaves a server. Specifically, for each $s_j \in \mathcal{S}$, this information is:

- the service time T_j , static throughout the game;
- the service quality $q(s_j)$ of its DNN, also static throughout the game;
- the number of clients served v_j ;
- the current RTT from client n_i to this server (l_j^i).

The collection of these values is stored in a set of tuples $I = \{(s_j, T_j, q(s_j), v_j, l_j^i) \mid s_j \in S\}$. Finally, we denote by $\hat{s}_i$ the current choice of n_i and by c_0 the local inference cost. Algorithm 1 shows how a client n_i selects a server under the SSG (c_j^i represents both Equations (4) and (5) for simplicity).

Algorithm 1 Selection process of clients under the SSG.

Require: $I, c_j^i, c_0, \hat{s}_i, T_i$

```

1:  $C = [c_0], R = [n_i]$   $\triangleright$  Lists of costs and resources
2: for  $(s_j, T_j, q(s_j), v_j, l_j^i) \in I$  do
3:   if  $l_j^i \neq \infty$  then
4:     if  $s_j = \hat{s}_i$  then
5:       compute  $d_j^i$  using  $v_j$   $\triangleright$  Using Eqs. (2) and (6)
6:     else
7:       compute  $d_j^i$  using  $v_j + 1$ 
8:     end if
9:     compute  $c_j^i$  using  $d_j^i, l_j^i$  and  $q(s_j)$ 
10:     $C \leftarrow C \cup [c_j^i], R \leftarrow R \cup [s_j]$ 
11:   end if
12: end for
13:  $\hat{s}'_i \leftarrow \arg \min_{r \in R} C[r]$   $\triangleright$  Pick the smallest punishment
14: if  $\hat{s}'_i \neq \hat{s}_i$  then
15:   Inform  $\hat{s}_i$  that he is disconnecting
16:    $\hat{s}_i \leftarrow \hat{s}'_i$ 
17: end if
```

For $\alpha_i \approx 0$, Algorithm 1 can be simplified by replacing Equation 3 with a simple Boolean check $d_j^i < \tau_i$. We will refer to it as “filter-then-select” (FTS) algorithm to distinguish it, given its selection behaviour discussed previously. Notice that the resulting algorithm is equivalent to strictly following the description in Equation 1. Thus, we will also treat the FTS algorithm as a baseline in the upcoming experiments.

B. The Restricted SSG

As discussed at the end of Section III, SSG is a finite game and therefore admits at least one mixed NE. However, mixed equilibria are difficult to interpret and implement in distributed systems. Moreover, PNE are not always guaranteed to exist, and best-response dynamics may exhibit prolonged or cyclic behaviours. To address these challenges, we restrict each player’s strategy set to two servers. Specifically, we leverage the network topology to pre-select the two servers with the shortest graph distance for each client. This reduction yields a player-specific congestion game with a binary strategy space per player that is proven to possess the finite improvement path (FIP) property [27], ensuring the existence of PNE [32]. Moreover, games with FIP are guaranteed to converge to a PNE via simple best-response dynamics [32], whereby players iteratively and unilaterally switch to their cost-minimising strategy in response to the current state of the game. The two-strategy structure simplifies the dynamics of the best-response process, improving tractability while preserving locality in decision-making, an essential property in dynamic environments such as the one considered in this paper.

Algorithm 2 formalises the decision process of a client node n_i within the *restricted server-selection game* (RSSG). The

client selects a server from a constrained subset $S_k \subset S$ composed of at most k candidate servers. Thus, the client operates on a reduced information set $I_k^i = \{(s_j, T_j, q(s_j), v_j, l_j^i) \mid s_j \in S_k^i\}$, where $S_k^i \subset S$ is the subset assigned to n_i , instead of the full informations set I . In practice, S_k^i is determined by the mesh router that functions as an access point of n_i , which selects the k closest servers in the WMN. When $k = |S|$, the RSSG reverts to the unrestricted SSG. Thus, following the previous discussion, we set $k = 2$ to guarantee the existence of a PNE. Unlike Algorithm 1, node n_i does not include itself in the RSSG. Therefore, we introduce a threshold c_0 , interpreted as a local cost, to allow n_i to remove itself from the RSSG whenever no server can deliver inference results within its required latency constraints.

Algorithm 2 Selection process of clients under the RSSG.

Require: $I_{topk}, c_j^i, c_0, \hat{s}_i, T_i$

```

1:  $C = [], R = []$   $\triangleright$  Lists of costs and resources
2: for  $(s_j, T_j, q(s_j), v_j, l_j^i) \in I_{topk}$  do
3:    $\vdots$   $\triangleright$  Compute  $d_j^i$  and  $c_j^i$  as in Algorithm 1.
12: end for
13:  $\hat{s}'_i \leftarrow \arg \min_{r \in R} C[r]$   $\triangleright$  Pick the smallest punishment
14: if  $\hat{s}'_i \neq \hat{s}_i$  then
15:   Inform  $\hat{s}_i$  that he is disconnecting
16:   if  $C[\hat{s}'_i] > c_0$  then
17:      $\hat{s}_i \leftarrow n_i$   $\triangleright$  Execute inference locally
18:   else
19:      $\hat{s}_i \leftarrow \hat{s}'_i$ 
20:   end if
21: end if
```

C. Selecting α_i

Selecting α_i not only impacts client n_i choices but also those of other clients. A small α_i makes n_i more tolerant of high inference latencies, making it more likely that n_i will select distant servers, which can worsen the experience of clients closer to those servers. The proposed RSSG mitigates this issue, but client choices are still affected by the appropriate choice of α_i . Still, we propose a simple analytical method for selecting a satisfactory α_i . We select an α_i that can make the cost function of the theoretically best DNN, i.e., a server with $q_j = 1$, exceed the local cost c_i^i at a specific latency $\hat{d}_j^i$. Using this rule, we can derive an α_i for Equation 4 as follows. Select $\hat{d}_j^i$ as the latency value after which we want the cost function of the theoretically best server to be capable of exceeding the local cost function c_i^i . Given that ϕ is a monotonically non-decreasing function, this occurs at the point where $c_j^i(\hat{d}_j^i) = c_i^i(0)$, as $\forall x < \hat{d}_j^i \nexists y > 0$ such that $c_j^i(x) > c_i^i(y)$ (see Figure 2(a) in Section III for a visual illustration). In this scenario, we have

$$\begin{aligned}
\alpha_i \phi(\hat{d}_j^i) + (1 - \alpha_i)(1 - 1) &= \alpha_i \frac{\tau_i}{\tau_i - 0} + (1 - \alpha_i)(1 - q_i) \Leftrightarrow \\
&\Leftrightarrow \alpha_i \phi(\hat{d}_j^i) = 1 + \alpha_i q_i - q_i \\
&\Leftrightarrow \alpha_i = \frac{1 - q_i}{\phi(\hat{d}_j^i) - q_i}.
\end{aligned} \tag{7}$$

Following the same steps, we can derive a similar rule for Equation 5, which yields

$$\alpha_i = \frac{q_i^{-1} - 1}{\phi(d_j^i) + q_i^{-1} - 2}. \quad (8)$$

V. NODE COMMUNICATION AND DEPLOYMENT CHOICES

A key aspect to ensure that Algorithms 1 and 2 work properly is to guarantee that information about servers that share the same partition as client n_i is up to date. In this regard, we implement a simple algorithm on the servers to track the number of connected clients and share this information with the network. This algorithm counts the number of open sockets, which close after a client disconnects or a predefined timeout expires. In our experiments, we set this timeout to be 25% larger than the client's sampling period to remove unresponsive clients and ensure that the response time estimates (Equation 6) are consistent. Every server s_j informs the network of its current number of clients (v_j), along with its service time and quality, periodically or whenever v_j changes. These messages are disseminated to all nodes in the network, and each client updates its information about each server.

Every client periodically transmits a small sequence of 5 ICMP Echo Requests for each server $s_j \in S$ ($s_j \in S_k^i$, in RSSG case) to obtain an estimate of l_j^i . We set the interval between consecutive messages in the sequence to 50 ms, and an interval of 2 s between consecutive l_j^i estimations. We take the average of all successful attempts, provided at least half of all attempts succeed; otherwise, we consider the connection unreliable and set $l_j^i = \infty$. We also adopt the following choices to minimise the overhead associated with these l_j^i estimations. When client n_i does not follow the RSSG, n_i starts monitoring server s_j upon receiving an advertising message from s_j , provided it is not already monitoring s_j . Additionally, n_i stops monitoring s_j after 5 consecutive tries that yield $l_j^i = \infty$, after which it assumes that s_j is unreachable. Finally, we further limit this procedure by defining a timeout of 1s for each request. If n_i follows the RSSG, it probes all servers in S_{topk}^i , adapting whenever this set is updated. Equipped with this latency information, each client updates its knowledge in I (or, in the case of Algorithm 2, I_{topk}) and performs its selection.

A. Implementing a Shared Data Space

The success of SSG (and RSSG) requires that the information about the congestion on each server, i.e., the number of clients v_j served by s_j in I (I_{topk}) is correct. However, meeting this requirement in the studied mobile wireless network can be challenging because we need to ensure that:

- updates are delivered with minimal delay and low temporal deviation across clients within the same partition, to enable them to correctly know if the other clients have changed their strategies (i.e., changed servers);
- the approach supports an unknown and dynamic set of clients; and
- information sharing remains operational under network disconnections, operates independently across partitions, and resynchronizes upon reconnection.

Based on these requirements, we propose implementing a *shared data space* in which servers share their current state and clients collect these updates. Specifically, we propose a distributed messaging implementation where servers do not need to know the number of listeners, and routing is handled transparently to the node's communication application. A suitable protocol should adequately meet these requirements.

In this work, we implement the shared data space via a control messaging environment in which servers can broadcast their current state using the `Matrix` protocol. `Matrix` is an open-source standard for decentralised communication with native state synchronisation, which effectively handles dynamic clients and network partitions, with over a million monthly active users and progressively adopted in the European public sector [33]. It is a type of decentralised publish-subscribe (PubSub) communication protocol designed for instant messaging, voice-over-IP, and IoT applications [20] that synchronises JavaScript object notation (JSON) objects across different devices without the need for a centralised broker. We exploit this characteristic to enable clients to receive updates from any reachable server, allowing each partition to function as an isolated network. However, the `Matrix` persistence mechanism delivers all messages published in each partition to all members of the communication channel upon reconnection, which could cause clients to execute the server-selection algorithms with outdated information. Preliminary experiments further showed that synchronising a large number of messages after reconnections can significantly increase message delivery latency, causing received state information to become stale; an issue expected to worsen in larger deployments with high message volumes. To mitigate this impact on the operation of the proposed algorithms, we use the `Matrix`'s time-sampling policy to make clients ignore messages older than 5 s.

Servers share their current state by sending a text message (referred to as an event) in a `Matrix` room (equivalent to a PubSub topic) in a single packet whose size ranges from 1.5 KB to 2.7 KB in our setup. Additionally, the receivers in the room reply to every event using a 500 B packet, following the `Matrix` protocol. As discussed previously, servers transmit their state information periodically or whenever the number of clients they serve changes. We set the periodic updates to happen every 30 s to minimise the impact of the control traffic. Although we identified a non-negligible traffic generation volume and scalability issues as the update volume increases, we select `Matrix` for its resilience to network partitioning and low CPU usage on resource-constrained clients. We discuss the `Matrix`'s traffic generation in more detail in the appendix A.

B. Data Transfer Between Clients and Servers

We implement the server-client communication using the transmission control protocol (TCP). We opt for this approach to ensure the integrity of the images transmitted to the server and to leverage TCP functionalities to verify if the connection between client and server is open. As shown in Figure 5, n_i opens a TCP connection with s_j the first time it selects this server. This connection will remain open until either the client or the server closes it, saving time and communication

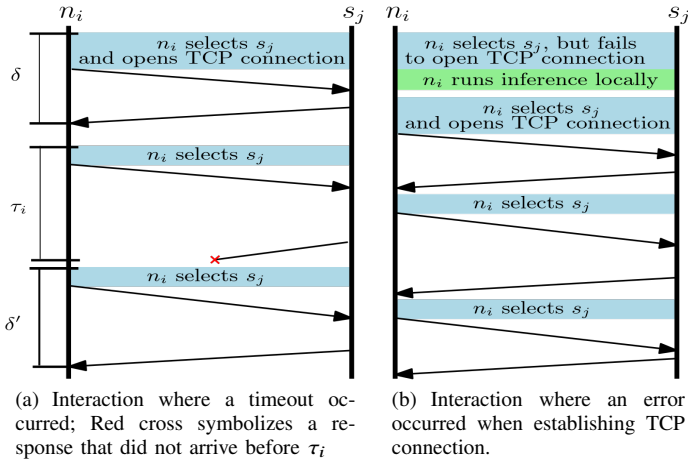


Figure 5. Simplified message diagrams showing the interaction between client and server. Blue rectangles indicate the time interval during which client n_i performs server selection and other operations before sending a new sample. n_i waits for an answer from s_j for τ_i from the moment it generates a new sample, after which the server's response is considered lost.

resources incurred by the TCP three-way handshake. If s_j does not answer n_i before τ_i , the connection will remain open for the next iterations, unless n_i selects another server (Subfigure 5(a)). If n_i fails to open a connection, as in Subfigure 5(b), it executes the inference locally and, if it selects this server s_j again, retries in the next iteration.

C. Determining the Top-K Closest Servers

As discussed in Section IV, clients following the RSSG are presented with a set of $k = 2$ pre-selected servers from which they choose one using a similar procedure as in SSG. This set is constructed from client-server distances, determined by the router serving as the client's access point. In our implementation, the routers retrieve the network graph and compute the shortest path to each server for which it has an active path using Dijkstra's algorithm [34]. They do so by collecting a GraphML file that describes the network graph [35], simulating the procedure implemented on Green Communication's router API [36], which constructs the network topology from information retrieved from the optimized link state routing protocol (OLSR) [37]. Each router sends a message containing its 2 closest servers in the shared data space, which clients connected to this router then retrieve. These messages are sent periodically every minute or whenever the two closest servers change. We opted for this messaging frequency because this information changes less frequently, and to avoid overloading the shared data space. However, new clients arriving between messages that failed to retrieve this information within 20s can ask the "access-point" router directly for it.

VI. EXPERIMENTAL SETUP

We compare the proposed selection algorithms described in Section IV with other simple and uncoordinated selection algorithms with low-telemetry requirements. Specifically, the algorithms are: a greedy algorithm that always tries to maximise service quality, a risk-averse algorithm that always selects the closest server, and a random selection algorithm. All strategies filter out servers for which the RTT is greater

than or equal to τ_i , using the same node communication and probing procedures discussed in Section V. After that, they select the best server that meets their objective: the greedy scheme selects the most accurate DNN, the risk-averse scheme selects the closest server, and the random one selects a server at random. Specifically, the selection strategies derived from these algorithms exploit the Matrix-based shared data space to discover servers within their partition, ignoring information irrelevant to their decision-making process. Finally, in addition to these algorithms, we also use the FTS algorithm as a baseline, since it is equivalent to directly solving the optimisation problem described at the end of Section II. Similar to our proposal, all these selection algorithms share the low telemetry requirements, estimating the distance to servers and network status through the probing mechanism described previously, and have low computational complexity, with all schemes being $O(n)$ (dependent on the number of servers), with the random one being $O(1)$. Specifically, the proposed algorithms share this computational complexity, with the RSSG being more efficient than $O(n)$ due to the restriction on available servers.

In the synthetic traces, we evaluate each algorithm individually using 8 clients, 3 servers and 6 routers. We opted for this client-server configuration because it resembles the real-world trace discussed later in this section and to encourage competition between the clients. We simulate network configurations with heterogeneous nodes simulating RPis, where:

- clients (labelled n_i) are RPi3B+ hosting a MobileNetV3_small [38] for local inference;
- one server (s_1) is a RPi4B hosting a MobileNetV3_large [38];
- and the remaining servers are RPi5, one (s_2) hosting a Swin_t, a type of Swin Transformer [39], the other (s_3) a multi-axis vision transformer (MaxViT) [40].

These models are provided by torchvision [41]. In this context, Table I summarises the main characteristics of this configuration, where the accuracies were measured using the validation set of the ILSVRC2012 ImageNet dataset [23], and inference latency values were measured in preliminary experiments using 224×224 RGB images.

Table I
NODE CONFIGURATION

Node Id.	Acc. [%]	Inf. Latency [ms]
n_i	67.67	162
s_1	75.31	231
s_2	81.10	293
s_3	83.39	622

A. Network Emulation

We perform our experiments using mesh network simulator (Meshsim) [21]. This software emulates dynamic network environments by managing connections between Docker container nodes. Meshsim offers an interactive mode in which the network topology can be modified via the user interface, and an automated mode in which the network dynamics are determined by a series of GraphML files. In particular, in automated mode, the user interface enables observation of dynamic changes in the network.

In our experiments, we construct three directories with GraphML files using known WMN topologies [42] and node mobility. The series represents a 15-minute period during which the network undergoes topology and partition changes, as well as client node movement. We construct three scenarios: (1) topological changes (due to link failures, recoveries, and partition changes) without client mobility, (2) static network topology with mobile clients, and (3) topological changes with client mobility (combining the two previous scenarios). The starting positions of servers and clients were randomly assigned before constructing the traces; however, we maintained the same starting position throughout all experiments to enable a fairer comparison between the selection algorithms. Finally, we set the maximum bandwidth for each link to 5.12 Mbit/s, the default value in Meshsim [21]. In our experiments, each image accounts for a payload of approximately 1.2 Mbits. This means that one image takes at least 230 ms to transfer over each link, making link bandwidth a non-negligible bottleneck in the synthetic traces.

B. Real-World Scenario: Mission Critical Operations

In addition to the handmade scenarios, we analyse the selection algorithms with a real-world WMN trace obtained from a staged operation of the Parisian firefighters [22]. We take the nodes corresponding to routers and the firefighters' smartphones (labelled "backhaul" and "station" in the dataset, respectively). We assign the server role to the nodes corresponding to the crew commanders and drivers, whereas the remaining firefighters are assigned the client role. In this scenario, we have 9 clients and 4 servers, where we treat the crew commander nodes as s_1 in the synthetic traces and the drivers as s_2 and s_3 . We opted to do this because the former group of nodes is more mobile. Overall, the firefighters' trace used in our experiments corresponds to about 20 minutes.

Unlike the synthetic scenarios, we allow client nodes to perform inference locally if the "wait-for-server-inference" timeout occurs. We consider this appropriate to the mission-critical scenario we want to analyse, as inference misses might not be acceptable, even when low. Nonetheless, we still consider these occurrences undesirable, as the client node would expend communication resources on an unsuccessful inference attempt and increase the overdue rate.

C. Evaluation Metrics and Further Considerations

Client nodes randomly iterate over the same predefined sequence of samples from ILSVRC2012 ImageNet validation dataset [23], as the Poisson process discussed in Section IV. This sequence of predefined samples was sorted randomly before performing the experiments. The ILSVRC2012 ImageNet validation dataset is a large image classification dataset with more than 50,000 samples across 1,000 image classes. Because the analysed scenarios were limited in duration (15 minutes for the synthetic and 20 minutes for the firefighter scenarios), clients do not iterate over the entire dataset, and the number of samples each client uses varies. In this regard, for each client, we compute the local model's accuracy over the predefined sequence up to the last

sampled element and compare it with the accuracy obtained using the selection algorithms over the same truncated sequence. Using local accuracy as the baseline, we compute the difference relative to the selection algorithms, where a positive value indicates DNN performance improvements over local inference. We also compute the number of missed inference requests in the synthetic scenarios and the number of overdue responses, *i.e.*, replies arriving later than τ_i after sample generation, in the real-world scenario. Finally, we measure how often clients switch servers to estimate which selection criterion results in fewer changes in client-server distributions.

Clients perform the selection process using τ_i in some scenarios, and a relaxed version $\tau'_i = \tau_i - \epsilon$ to account for the estimation errors discussed in Section IV. We analyse scenarios where ϵ is equal to 1×10^{-9} , 1×10^{-6} , and 1×10^{-3} , where higher ϵ values provide a larger buffer against estimation errors. Additionally, in our experiments, $T_i = 4s$ and $\tau_i = 3s$, meaning that τ_i is $\frac{3}{4}$ of the sampling period.

Following the discussion in Section IV-C, we select α_i using $\hat{d}_j^i \in [\frac{11\tau_i}{16}, \frac{7\tau_i}{8}]$. Thus, using the configuration described in Table I, we have $\alpha_i \in [0.13, 0.07]$ and $\alpha_i \in [0.18, 0.1]$ when using Equations 4 and 5, respectively³. We perform 10 repetitions per parameter combination on the handmade traces, using the ensuing results as reference for the parameters of the firefighters' trace. We compute the weighted average of misses, overdues, changes, and the accuracy gains obtained after concluding all repetitions, using the number of samples generated in each run as the weights. For instance, considering N_k to be the number of samples used by client n_i in the k -th experiment, we consider average miss to be $\text{Miss}(n_i) = \frac{\sum_{k=1}^{10} N_k \text{Miss}_k(n_i)}{\sum_{k=1}^{10} N_k}$. All the code and GraphML files are available at [43].

VII. RESULTS WITH SYNTHETIC TRACES

Under the 15-minute period, each client generates about 230 samples. Figure 6 presents the average results of the best selection strategies for all clients. In it, the topmost subplots show the average accuracy (percentage) differences relative to running inference locally, while the remaining subplots show the average inference failures and server changes across the experiments. Additionally, Table II presents the individual (average) results for each client (in bold, we highlight the best results for each algorithm). Table II also supports one of the main assumptions of this work: if all clients select the best DNN, its server will become overloaded, thereby degrading their user experience. This is particularly evident in scenarios where clients are mobile in a static network, where each client has a communication path to every server. In this scenario, all clients experienced a decrease in accuracy of at least 35% percentage points due to miss rates exceeding 45%. We observe a similar trend when clients adopt the FTS algorithm, though to a lesser extent than with the greedy selection algorithm. This shows the difficulties of solving the objective described in Equation 1 in a dynamic wireless environment with limited and sometimes outdated information. Recalling that FTS can be seen as a particular case of the proposed

³We round the computed α_i values to the second decimal place.

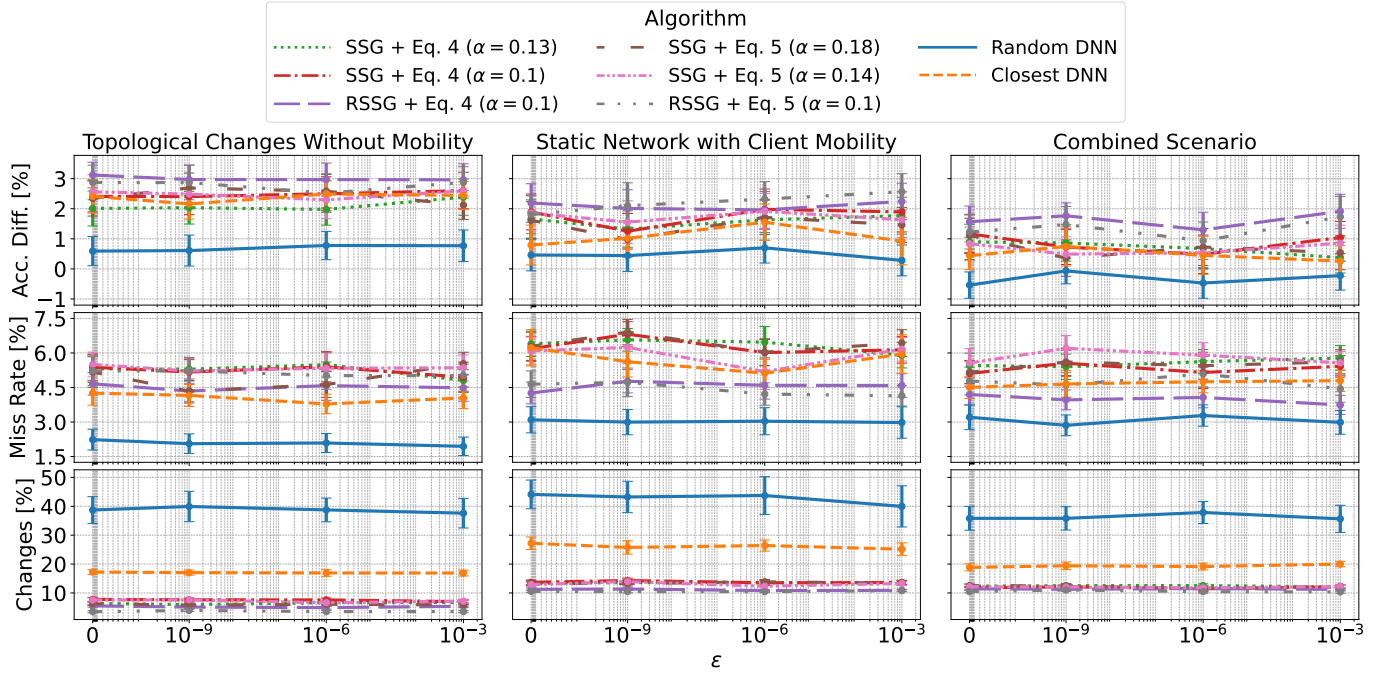


Figure 6. Weighted average of all clients, for best strategies in Table II, except the greedy strategy. x-axis in linear scale from $\epsilon = 0$ to $\epsilon = 10^{-10}$ and logarithmic scale onwards to improve readability. In all scenarios, RSSG obtained the best results. Additionally, we see that both SSG and RSSG have the best results when $\epsilon = 0$, suggesting that the game-based algorithms are more robust to estimation errors. In particular, the FTS algorithm suggests that trying to solve Equation 1 directly can lead to inconsistent performance, as this approach yields the largest discrepancies (as exemplified by the size of the standard error of the mean).

SSG-based algorithm, FTS is a more risk-taking version of Algorithm 1 that will stick to the best DNN available even when the time estimates approach the client's τ_i deadline.

Unlike the greedy and FTS algorithms, the random selection algorithm consistently showed the lowest miss rates. However, these miss rates did not translate into improvements in the DNN task performance, as this algorithm sub-exploits the best DNN available. Similarly, the risk-averse algorithm that selects the closest server also exhibits low miss rates, experiencing higher accuracy improvements. However, the results in Figure 6 indicate that this algorithm does not handle mobility well, particularly when $\epsilon = 0$ (i.e., when the selection algorithm does not account for imperfections in network state estimation). And although some ϵ configurations outperformed some configurations of the proposed unconstrained algorithm, this was not consistent across scenarios: $\epsilon = 10^{-6}$ in the static network with mobility, and $\epsilon = 10^{-9}$ in the combined scenarios. Additionally, as will be seen in the next section, accuracy improvements depend on the quality of the closest server, which can lead to inconsistent gains across clients.

Figure 6 shows that the proposed algorithms consistently delivered the best results when $\epsilon = 0$, achieving the highest accuracy improvements among all strategies under this condition. This suggests that the game-based approach handles inconsistencies in network state estimation better. Also, as shown in Table II, these configurations are close to the best ϵ configurations, and in some cases delivered the best results. Restricting the analysis to SSG and RSSG, the “changes” plot shows small discrepancies between the restricted and unrestricted SSG. However, the unrestricted implementations consistently exhibited higher misses than their restricted counterparts. In the SSG, a client can select a distant server, making

it more prone to missed inference requests. RSSG can mitigate this issue at the cost of implementing mechanisms that gather knowledge of the network topology. For instance, the RSSG with $k = 2$, analysed here for its convergence guarantees, achieved the best accuracy across all scenarios. This suggests that control over the state in which the game converges is desirable to ensure consistent improvements. Still, as will be seen later, SSG can perform better if the application is more tolerant of overdue responses. Finally, Table II and Figure 6 indicate that different values of α_i may perform better under different network conditions and ϵ choices, suggesting that network-aware tuning could further improve the performance of the proposed algorithms. The analytically based α_i proposed in this work provides a practical operating point for the scenarios considered and thus can be regarded as a practical baseline that can be further refined to improve client performance.

The “changes” results in Figure 6 show another advantage of the game-based algorithms: once the game converges, clients will switch between servers only when significant changes occur to the network. In particular, when we focus on this metric, clients using the algorithm that selects the closest DNN (which obtained the best results among baseline strategies) switch between servers twice as often as the ones using the proposed algorithms. The former clients have to open and close TCP connections more frequently, which, although less costly than image transmissions, remains non-negligible on resource-constrained devices.

VIII. RESULTS WITH REAL-WORLD TRACE

Figure 7 brings some of the best results of the baseline strategies (FTS and closest DNN) against the proposed game-based algorithms using $\epsilon = 0$ using the firefighters’ trace

Table II

WEIGHTED AVERAGE OF THE ACCURACY PERCENTAGE DIFFERENCE BETWEEN EACH ALGORITHM AND RUNNING ALL THE INFERENCE LOCALLY. THE SHADES OF GREEN (RED) REPRESENT THE MAGNITUDE OF THE AVERAGE ACCURACY IMPROVEMENT (DETERIORATION), SEEN IN FIGURE 6.

	ϵ	Topological changes without mobility								Static network with client mobility								Combined scenario							
		n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
Random DNN	0	1	3	1	-1	0	0	-1	1	2	3	-1	0	0	-1	-1	1	-1	-1	0	-1	-1	-1	0	0
	10^{-9}	-1	2	1	-1	0	2	0	1	2	2	-1	0	1	-1	-1	1	0	0	0	0	0	0	0	0
	10^{-6}	1	3	2	0	1	0	0	1	3	1	0	1	0	0	0	1	0	-1	0	0	-1	0	-1	-1
	10^{-3}	0	3	2	-1	1	1	0	0	2	2	0	0	0	0	-1	1	1	-1	0	-1	0	0	0	0
Closest DNN	0	0	3	4	1	4	3	1	2	3	3	0	0	1	0	-1	1	1	1	2	1	1	0	-1	0
	10^{-9}	1	4	4	1	3	2	1	1	2	3	0	2	0	1	0	0	2	1	2	0	0	0	0	1
	10^{-6}	1	5	4	2	4	2	1	2	4	4	0	2	1	0	1	2	2	0	1	0	-1	1	0	-1
	10^{-3}	-1	4	3	2	4	4	1	2	2	3	1	1	0	-1	0	0	1	0	0	3	-1	2	-1	-2
SSG + Eq. 4 $\alpha = 0.13$	0	0	5	3	0	3	3	0	0	4	4	1	1	0	0	1	1	1	1	1	1	1	1	-1	1
	10^{-9}	0	6	4	0	3	3	1	-1	3	3	0	1	2	1	-1	1	1	2	1	1	2	-1	1	1
	10^{-6}	-1	6	4	0	3	3	0	0	4	3	1	2	2	0	0	2	0	1	1	1	1	0	-1	1
	10^{-3}	0	6	4	1	2	3	1	0	4	3	1	3	2	0	0	2	0	0	1	1	1	1	-2	1
SSG + Eq. 4 $\alpha = 0.1$	0	1	6	4	1	3	3	0	1	4	4	2	2	0	0	1	2	1	2	1	1	3	-1	2	2
	10^{-9}	-1	7	4	0	4	4	1	0	4	3	-1	1	1	0	0	1	2	1	0	1	0	0	0	2
	10^{-6}	1	6	4	1	4	3	1	0	3	4	1	2	2	1	0	3	1	0	0	1	1	1	0	0
	10^{-3}	0	7	4	2	4	4	0	1	3	3	0	3	2	1	0	3	2	1	1	2	0	1	0	1
RSSG + Eq. 4 $\alpha = 0.1$	0	1	8	4	1	4	3	2	2	4	4	0	3	2	1	1	3	2	2	1	2	2	1	0	3
	10^{-9}	1	7	4	2	4	3	1	2	3	4	2	1	2	1	1	2	3	2	2	3	2	2	0	2
	10^{-6}	1	7	4	2	4	3	0	3	4	3	2	2	1	2	1	1	1	2	2	1	1	2	0	2
	10^{-3}	1	7	4	1	4	3	1	2	3	5	2	2	2	1	1	2	2	3	3	2	2	2	0	2
SSG + Eq. 5 $\alpha = 0.18$	0	0	6	4	0	3	4	1	0	3	5	0	2	1	1	0	2	1	2	2	1	1	2	0	0
	10^{-9}	0	7	4	1	3	5	1	0	2	3	0	2	0	1	-2	1	1	1	0	1	0	1	0	0
	10^{-6}	1	7	4	0	3	3	0	1	3	3	1	3	1	1	0	2	2	1	1	1	0	1	-1	1
	10^{-3}	-1	7	4	0	3	3	1	1	2	4	0	2	2	1	1	1	0	1	1	1	0	1	0	0
SSG + Eq. 5 $\alpha = 0.14$	0	1	6	3	1	4	4	1	1	3	4	1	2	1	1	1	2	1	2	2	1	1	1	0	0
	10^{-9}	0	7	4	0	3	4	2	1	2	2	1	2	1	1	-1	2	1	1	0	0	1	1	0	0
	10^{-6}	0	7	3	1	3	3	0	2	4	4	0	1	2	1	0	2	0	0	1	1	1	1	0	1
	10^{-3}	1	6	4	1	3	3	1	1	3	4	1	2	1	0	1	1	2	1	2	1	1	1	-1	1
RSSG + Eq. 5 $\alpha = 0.1$	0	1	6	5	2	4	3	1	2	2	3	1	2	1	0	1	3	3	2	2	2	1	1	-2	2
	10^{-9}	1	7	4	2	4	3	0	3	3	4	2	2	3	1	0	2	3	1	2	2	2	0	-1	3
	10^{-6}	1	4	4	1	3	4	1	2	3	5	1	3	1	2	0	3	1	1	1	1	1	1	-1	1
	10^{-3}	0	6	4	1	4	3	1	3	4	6	1	2	2	1	1	3	3	1	3	2	1	2	1	2
FTS	10^{-9}	-4	-1	-1	-1	-2	-1	-3	-2	-4	-3	-4	-6	-3	-3	-2	-4	-3	-4	-3	-3	-3	-3	-2	-1
	10^{-9}	-2	-1	-3	-3	-3	-3	-3	-2	-3	0	-3	-4	-1	-3	-4	-3	-4	-3	-4	-4	-2	-3	-2	-3
	10^{-6}	-4	0	0	-3	-1	0	-1	-3	-3	-3	-4	-4	-3	-4	-3	-3	-1	-2	-3	-2	-3	-1	-3	-2
	10^{-3}	-4	0	-2	-2	-3	-2	-1	-2	-4	-2	-3	-2	-4	-1	-3	-2	-3	-3	-3	-3	-3	-2	-4	-3
Best DNN	0	-36	-34	-30	-26	-30	-32	-34	-36	-47	-40	-40	-47	-47	-41	-40	-45	-29	-28	-32	-28	-23	-30	-24	-25
	10^0	-26	-22	-24	-26	-21	-23	-26	-28	-51	-53	-43	-49	-50	-40	-45	-48	-26	-23	-27	-29	-24	-28	-24	-23
	10^{-6}	-33	-32	-27	-32	-28	-27	-33	-34	-43	-40	-36	-38	-40	-35	-37	-41	-28	-24	-27	-31	-25	-29	-28	-28
	10^{-3}	-36	-33	-31	-34	-31	-32	-32	-36	-48	-49	-41	-45	-46	-38	-38	-46	-28	-26	-31	-31	-24	-28	-26	-26

[22]. This scenario closely resembles the combined scenario from Section VII, where clients are mobile and router connections vary over time. In contrast to the synthetic traces, the s_1 servers are also mobile because they correspond to the crew commanders. During the 20-minute period, each client generates approximately 300 samples. As stated in Section VI, we modify the procedure used in the previous experiments to allow clients to perform local inference if a server request does not arrive before τ_i . This modification enhances the resilience of the DNN application to communication failures; however,

it remains undesirable compared to directly selecting local execution, as the client wastes energy on a failed inference request and experiences inference latencies beyond τ_i . To this end, we measure the percentage of overdue inference requests, i.e., inference requests that exceed the desired timeout.

Initially, when we focus on accuracy gains, the FTS algorithm achieved the greatest improvements, followed closely by the proposed algorithms using the unconstrained SSG. As shown by the dark-red line, clients using FTS achieved, on average, an increase of approximately 4.73

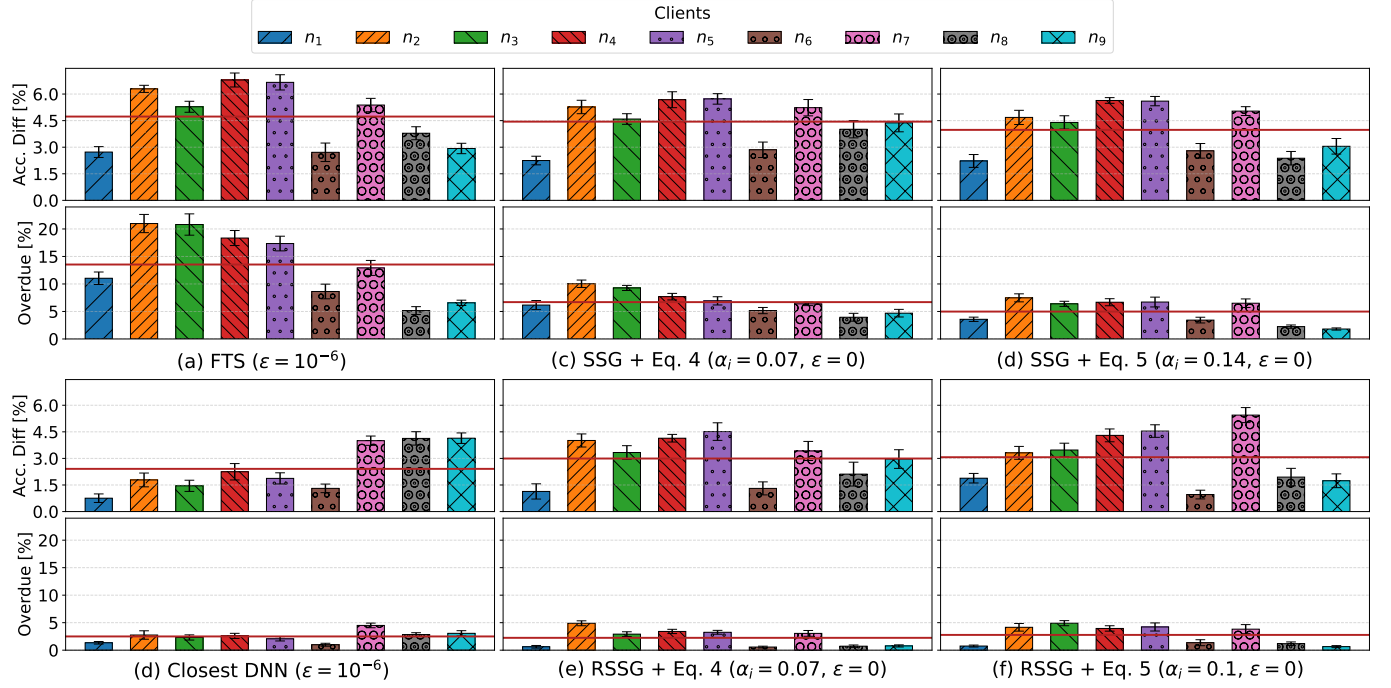


Figure 7. Best results of the baseline strategies that always select the closest server and FTS against the proposed algorithm using $\epsilon = 0$, with the firefighters trace [22]. The red line represents the mean for all clients.

percentage points compared to running inference locally, with 3 clients surpassing a 6-point improvement. However, these improvements come at a high overdue cost, where 13.53% of inference requests were overdue, with two clients experiencing 20% of their requests arriving late. In contrast, clients using the proposed unrestricted SSG and Equation 4 obtained 4.44 percentage-point improvements with 6.70% overdue on average, and clients using SSG and Equation 5 obtained 3.98 percentage-point improvements with 4.98% overdue. This entails an accuracy difference of less than 1% between the proposed selection algorithm and FTS, with at least half the latter's overdue rate.

When we focus on minimising the overdue rate, the proposed algorithms that use RSSG deliver the best trade-off between accuracy gains and the overdue rate. If the algorithm that selects the closest DNN delivers similar results across both metrics, the accuracy results are more uneven across clients. We consider the relative deviation from the mean to quantify this variability. When we compare it with the proposed RSSG strategies, 4 clients adopting the algorithm the closest-DNN algorithm exhibit accuracy improvements that deviate from the mean by at least 65%; in contrast, no client adopting RSSG with Equation 4 exhibit such deviation and only 2 clients present this trend when adopting Equation 5. Accuracy gains depend on the quality of the servers near each client when the client does not adopt the game-based selection algorithm. And even though this is still true in the RSSG to some degree, the game-based selection algorithm enables them to adopt “riskier” selections that maximise their accuracy gains. The result is that both RSSG-based strategies outperform the conservative algorithm in terms of accuracy – 2.99% and 3.07% when using Equations 4 and 5, respectively, against 2.41% when always selecting the closest DNN – with similar

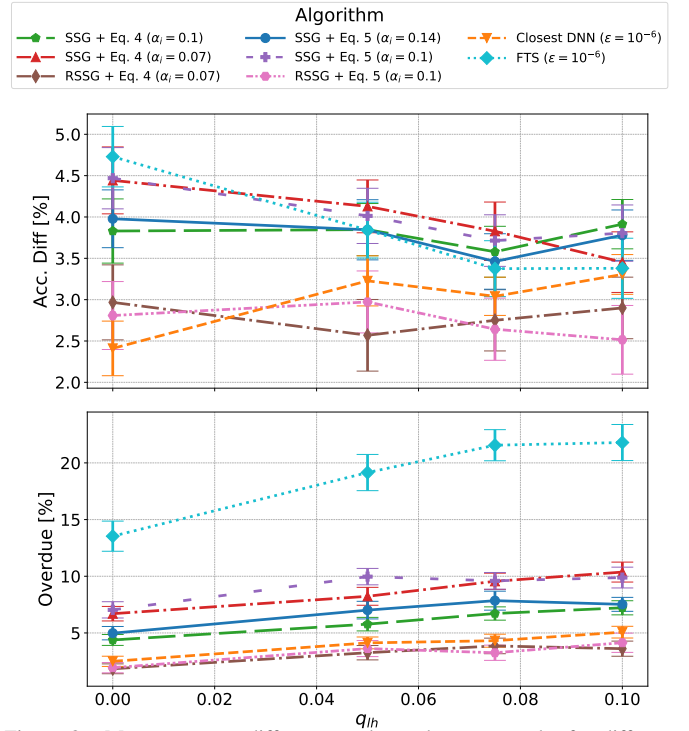


Figure 8. Mean accuracy difference and overdue rate results for different bursty traffic, modelled as an MMP. Clients double their sample generation rate during the “bursty” state, thus generating more inference requests. The higher q_{th} , the more likely clients will enter a “bursty” state. As was the case in Figure 7, the proposed algorithms use $\epsilon = 0$.

overdue rates – 2.25% and 2.78%, respectively, against 2.50%.

Results under bursty traffic

Although we assumed that clients generate inference requests as a stationary Poisson process, real-world mobile wireless networks often exhibit bursty traffic. In such scenarios,

clients may temporarily generate a higher volume of requests, causing the queuing model adopted in the previous formulation to no longer accurately capture the system state [44], leading to further estimation errors. To evaluate this impact, Figure 8 presents the mean accuracy difference and overdue rates of each selection algorithm under different burstiness levels. We model clients' request generation as a two-state Markov-modulated Poisson process (MMPP) [45], in which clients alternate between a low-rate state, using the same sample generation rate as in the previous experiments, and a high-rate state, in which they double that rate. Clients transition from high to low state with a rate $q_{hl} = 0.2$, and from low to high with $q_{lh} \in \{0.05, 0.075, 0.1\}$, yielding a mean sojourn time in the high-rate state equal to 5 s and a mean sojourn time in the low state equal to 20 s, 13.33 s and 10 s, respectively [46]. A larger q_{lh} implies more frequent transitions into the high-rate state, which in turn increases the burstiness.

Figure 8 shows that the performance of the FTS algorithm degrades significantly even at low burstiness levels. If, in Figure 7, this algorithm exhibited the best accuracy improvements, it is surpassed by the proposed unconstrained game-based algorithms already at the lowest burstiness level. Since FTS attempts to solve the server-selection optimisation problem directly, it operates on tight margins, making it more sensitive to estimation errors. The proposed unconstrained SSG algorithms perform better than it because they do not require precise estimates of expected latency. If the expected latency of the best server exceeds that of the client's second option, and the latter's DNN accuracy is deemed acceptable, the client will divert to its second option even if the first server's latency remains below τ_i . Unlike FTS, the strategy that selects the closest DNN is barely affected by clients' behavioural change, showing an improvement of more than 0.5 percentage point over the previous scenario, which did not happen with the other selection algorithms. This outcome is not unexpected because the algorithm that selects the closest DNN does not rely on the queuing model adopted by the other algorithms, as it always selects the server with the lowest RTT. Still, the fact that the unconstrained SSG algorithms outperform this more conservative algorithm, even under inaccurate network-state information, highlights the resilience of the proposed approach. Finally, although RSSG achieved lower accuracy improvements than the algorithm that selects the closest DNN under bursty traffic, it still achieved the lowest overdue rates among all algorithms. This is remarkable, considering that clients using RSSG make their selection based on an inaccurate queue model.

IX. CONCLUSION

This paper addresses the server-selection problem in wireless IoT networks, in which resource-constrained devices seek to outsource the execution of DNN inference to more powerful edge nodes. We present a decentralised server-selection system that is resilient to node mobility and robust against network partitioning, thereby ensuring continuous access to DNN services for clients with limited resources. Recognising structural similarities between this server-selection problem

and congestion games, we propose two server-selection algorithms grounded in game-theoretic principles. Additionally, we present a first implementation using the Matrix protocol to distribute real-time server load information to enable informed, distributed decision-making without the need for centralised coordination or client cooperation. The resulting system enables clients to balance service quality and latency requirements under dynamic network conditions while avoiding overloaded servers. Experiments with nodes emulating various RPi versions indicate that this method outperforms strategies that always select the best DNN available, the best that strictly satisfies the clients' time constraints, or the closest server in terms of RTT. Experiments using real-world mobility traces show that clients using the proposed algorithms achieved the best trade-off between accuracy improvements and overdue rate minimisation, consistently being among the best algorithms in both metrics while being more robust to network-state estimation errors. These findings confirm that the ensuing system meets its objectives by providing robust, decentralised DNN server selection that adapts to node mobility and network partitioning in heterogeneous IoT environments.

APPENDIX

A. Matrix Messaging in More Detail

In addition to ensuring that the *shared data space* works on partition-prone networks, as required by the proposed system and its intended application, it is important to quantify the Matrix protocol cost on resource-constrained devices. To this end, we perform performance-characterisation experiments to estimate the CPU usage, end-to-end latency from when a message is posted until it is delivered to every receiver, and the payload generated by the communication system. The results were collected on static networks with 11 nodes and varying density, with 5 receivers, 3 publishers, and 3 nodes that perform both tasks (i.e., they are receivers and publishers). Finally, we model publisher update generation as a Poisson process with an average sampling period ranging from 7.5 to 30 seconds, and we run each experiment instance for 10 minutes.

Tables III-V provide a summary of the main results, presenting the average of each measurement and its respective standard deviation. First, we notice that the total CPU time is low relative to the overall experiment duration, regardless of the node's role in the communication process. In particular, this characteristic is desirable for the proposed server-selection system, as it suggests that the Matrix does not interfere with in-node computations. Specifically, we use Matrix's Conduit version, which is specifically designed to run on resource-constrained devices such as RPis⁴.

Table III
AVERAGE TOTAL CPU TIME OF THE MATRIX SERVICE IN A
10-MINUTE-LONG OPERATION WITHOUT NETWORK SATURATION.

T [s]	Total CPU Time [s]
7.5	3.14 ± 0.32
10	2.52 ± 0.16
15	1.74 ± 0.19
20	1.38 ± 0.20
30	1.06 ± 0.20

⁴<https://docs.conduit.rs>

Tables IV and V present Matrix's end-to-end delivery latency and payload overhead across varying network densities ($\rho = \frac{2e}{n(n-1)}$, where e and n are the number of edges and nodes, respectively). Specifically, in Table V, Total includes all generated traffic, and Control Overhead accounts for the Matrix intrinsic signalling and network probing policies; and the average payloads measure operational messages shared in the proposed *shared data space* (average service – Avg Service), with average update (Avg Update) isolating messages that actually carried updates from publishers. The results show a trade-off between communication overhead and delivery latency. As expected, Matrix delivers updates to clients faster in denser networks, particularly when senders generate updates at higher rates; however, this network tends to produce higher communication overhead. At the lowest update generation rate, the average update message size is 2.64 KB. Conversely, the sparsest network exhibits an average of 1.57 KB at the highest rate. Because of Matrix's header, update transmission becomes more efficient as the number of updates increases, since a single Matrix message can carry multiple updates. Still, the latency results in Table IV suggest that a high volume of updates in a short period can overwhelm message dissemination via the Matrix protocol, although average latencies increased only mildly in all but the sparsest network. The noticeable increase in standard deviations suggests instances when Matrix struggled to forward multiple messages. In particular, we ran additional preliminary experiments in which Matrix average delivery latency exceeded 20 s in all networks when clients generated updates at $T = 5$ s.

Matrix proved effective in the setup studied in this paper because of the scale of generated updates, thanks to the number of nodes and frequent network disconnections, which further reduced the number of concurrent updates. In our setup, the main requirement was to ensure the communication system was resilient to network partitions and that messages were delivered seamlessly to the SSG-based application upon reconnection. Still, further design choices should be made to ensure its viability on larger networks, which are beyond the scope of this paper.

REFERENCES

- [1] K. Bochie, M. S. Gilbert, L. Gantert, M. S. Barbosa, D. S. Medeiros, and M. E. M. Campista, "A survey on deep learning for challenged networks: Applications and trends," *Journal of Network and Computer Applications*, vol. 194, p. 103213, 2021.
- [2] Y. Wang, C. Yang, S. Lan, L. Zhu, and Y. Zhang, "End-edge-cloud collaborative computing for deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2024.
- [3] C. Kyrkou, P. Kolios, T. Theodorides, and M. Polycarpou, "Machine learning for emergency management: A survey and future outlook," *Proceedings of the IEEE*, vol. 111, no. 1, pp. 19–41, 2022.
- [4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [5] M. S. Elbamby, C. Perfecto, C.-F. Liu, J. Park, S. Samarakoon, X. Chen, and M. Bennis, "Wireless edge computing with latency and reliability guarantees," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1717–1737, 2019.
- [6] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 514–522.
- [7] C. Ju and Q. Shao, "Energy efficiency oriented access point selection for cognitive sensors in internet of things," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, p. 619546, 2015.
- [8] Q.-V. Pham, T. Leanh, N. H. Tran, B. J. Park, and C. S. Hong, "Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach," *IEEE Access*, vol. 6, pp. 75 868–75 885, 2018.
- [9] S. Kumar, A. Goswami, R. Gupta, S. P. Singh, and A. Lay-Ekuakille, "A game-theoretic approach for cost-effective multicast routing in the internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 18 041–18 053, 2022.
- [10] B. Wu, J. Zeng, S. Shao, W. Ni, and Y. Tang, "New game-theoretic approach to decentralized path selection and sleep scheduling for mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 8, pp. 6125–6140, 2022.
- [11] H. Xu, H. Hui, C. Zhou, G. Zeng, and Z. Han, "Cooperative energy trading for hetnets with renewable energy: A dynamic energy trading game," *IEEE Internet of Things Journal*, vol. 11, no. 7, pp. 11 606–11 618, 2023.
- [12] Y. Gao, Z. Ji, K. Zhao, T. De Cola, and W. Li, "Game-based computation offloading and power allocation for leo constellation networks in distributed and dynamic environment," *IEEE Internet of Things Journal*, vol. 11, no. 4, pp. 7040–7058, 2023.
- [13] L. Cai, L. Luo, Z. Li, and Z. Xiong, "Game-based dynamic clustering routing strategy for mobile wireless sensor networks," *IEEE Internet of Things Journal*, vol. 11, no. 15, pp. 26 481–26 491, 2024.
- [14] I. Cohen, A. Calagna, P. Giaccone, and C. F. Chiasserini, "Distributed asynchronous service provisioning in edge-cloud multi-tier networks," *IEEE Transactions on Mobile Computing*, 2025.
- [15] B. Picano, A. Buratto, and L. Badia, "Joint communication and inference user allocation in llm native networks," in *2025 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. IEEE, 2025, pp. 1–6.
- [16] L. Gourves, J. Monnot, S. Moretti, and N. K. Thang, "Congestion games with capacitated resources," *Theory of Computing Systems*, vol. 57, no. 3, pp. 598–616, 2015.
- [17] M. Feldotto, L. Leder, and A. Skopalik, "Congestion games with mixed objectives," *Journal of Combinatorial Optimization*, vol. 36, pp. 1145–1167, 2018.
- [18] N. Bertrand, N. Markey, S. Sadhukhan, and O. Sankur, "Dynamic network congestion games," *arXiv preprint arXiv:2009.13632*, 2020.
- [19] F. Zhang and M. M. Wang, "Stochastic congestion game for load balancing in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 778–790, 2020.
- [20] Matrix.org Foundation C.I.C., "Matrix specification," <https://spec.matrix.org>, 2025, accessed: 2025-07-25; version v1.14.
- [21] Nicolas Peugnet, "Meshsim," <https://gitlab.lip6.fr/ie6/meshsim>, 2025, accessed: 2026-04-10.
- [22] N. Peugnet, A. Fladenmuller, T.-M.-T. Nguyen, B. Kervella, and J. Thourot, "Maintaining Connectivity in Subterranean Emergency Response: A 5G and Wi-Fi Mesh Mobility Dataset," in *NetMob 2025 Book of Abstracts*, Paris, France, Oct. 2025, pp. 37–38.
- [23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [24] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal et al., "Mobile-edge computing introductory technical white paper," *White paper, mobile-edge computing (MEC) industry initiative*, vol. 29, pp. 854–864, 2014.
- [25] T. Wang, T. Shi, X. Liu, J. Wang, B. Liu, Y. Li, and Y. She, "Minimizing latency for multi-dnn inference on resource-limited cpu-only edge devices," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 2239–2248.
- [26] B. Vöcking and R. Aachen, "Congestion games: Optimization in competition," in *ACiD*, 2006, pp. 9–20.
- [27] I. Milchtaich, "Congestion games with player-specific payoff functions," *Games and economic behavior*, vol. 13, no. 1, pp. 111–124, 1996.
- [28] H. Ackermann, P. W. Goldberg, V. S. Mirrokni, H. Röglin, and B. Vöcking, "A unified approach to congestion games and two-sided markets," *Internet Mathematics*, vol. 5, no. 4, pp. 439–457, 2008.
- [29] M. J. Osborne and A. Rubinstein, *A course in game theory*. MIT press, 1994.
- [30] J. Sztrik et al., *Basic queueing theory*. GlobeEdit OmniScriptum GmbH, KG, Saarbrücken, Germany, 2016.
- [31] G. Last and M. Penrose, *Lectures on the Poisson process*. Cambridge University Press, 2017, vol. 7.

Table IV
END-TO-END LATENCIES MEASURE ON NETWORKS WITH 11 NODES WITH DIFFERENT DENSITIES.

T [s]	Hop Distance	Latency [s]			
		$\rho = 0.182$	$\rho = 0.200$	$\rho = 0.236$	$\rho = 0.345$
7.5	1 to 3 hops	7.63 $\pm$ 15.43	2.06 $\pm$ 2.22	1.56 $\pm$ 1.15	1.26 $\pm$ 0.84
	4 to 7 hops	18.63 $\pm$ 29.15	4.36 $\pm$ 2.81	5.24 $\pm$ 4.89	3.57 $\pm$ 1.92
	Above 7 hops	96.34 $\pm$ 110.25	—	—	—
10	1 to 3 hops	2.43 $\pm$ 3.47	1.31 $\pm$ 0.77	1.25 $\pm$ 0.65	1.12 $\pm$ 0.62
	4 to 7 hops	13.00 $\pm$ 20.94	3.50 $\pm$ 1.61	3.53 $\pm$ 1.99	3.13 $\pm$ 1.39
	Above 7 hops	76.32 $\pm$ 80.53	—	—	—
15	1 to 3 hops	1.37 $\pm$ 1.03	1.17 $\pm$ 0.58	1.19 $\pm$ 0.58	1.08 $\pm$ 0.55
	4 to 7 hops	4.35 $\pm$ 2.88	2.89 $\pm$ 0.94	2.79 $\pm$ 0.97	2.40 $\pm$ 0.45
	Above 7 hops	12.87 $\pm$ 13.98	—	—	—
20	1 to 3 hops	1.10 $\pm$ 0.56	1.15 $\pm$ 0.55	1.17 $\pm$ 0.54	1.05 $\pm$ 0.52
	4 to 7 hops	3.48 $\pm$ 1.49	2.79 $\pm$ 0.79	2.47 $\pm$ 0.48	2.40 $\pm$ 0.43
	Above 7 hops	6.83 $\pm$ 3.36	—	—	—
30	1 to 3 hops	1.09 $\pm$ 0.53	1.14 $\pm$ 0.54	1.17 $\pm$ 0.54	1.07 $\pm$ 0.52
	4 to 7 hops	3.03 $\pm$ 0.81	2.61 $\pm$ 0.49	2.47 $\pm$ 0.45	2.41 $\pm$ 0.43
	Above 7 hops	5.45 $\pm$ 1.66	—	—	—

Table V
MATRIX-BASED SHARED DATA SPACE NETWORK TRAFFIC GENERATED AT DIFFERENT NETWORK DENSITIES OVER 10-MINUTE EXPERIMENTS.

T [s]	Metric	Network Traffic Volume			
		$\rho = 0.182$	$\rho = 0.200$	$\rho = 0.236$	$\rho = 0.345$
7.5	Total [MB]	18.54 $\pm$ 0.11	19.16 $\pm$ 0.13	20.22 $\pm$ 0.09	23.26 $\pm$ 0.15
	Avg Service [KB]	1.07 $\pm$ 0.01	1.06 $\pm$ 0.01	1.11 $\pm$ 0.02	1.27 $\pm$ 0.01
	Avg Update [KB]	1.57 $\pm$ 0.02	1.54 $\pm$ 0.02	1.61 $\pm$ 0.03	1.85 $\pm$ 0.02
	Control Overhead [KB]	1065.85 $\pm$ 5.73	1209.75 $\pm$ 46.27	1247.27 $\pm$ 37.16	1286.10 $\pm$ 55.48
10	Total [MB]	18.39 $\pm$ 0.15	18.94 $\pm$ 0.17	20.00 $\pm$ 0.12	22.98 $\pm$ 0.10
	Avg Service [KB]	1.11 $\pm$ 0.00	1.11 $\pm$ 0.01	1.17 $\pm$ 0.02	1.34 $\pm$ 0.02
	Avg Update [KB]	1.68 $\pm$ 0.01	1.66 $\pm$ 0.02	1.76 $\pm$ 0.04	2.04 $\pm$ 0.05
	Control Overhead [KB]	880.05 $\pm$ 41.26	983.52 $\pm$ 48.37	981.73 $\pm$ 54.43	991.07 $\pm$ 45.46
15	Total [MB]	18.18 $\pm$ 0.11	18.65 $\pm$ 0.18	19.72 $\pm$ 0.09	22.70 $\pm$ 0.16
	Avg Service [KB]	1.17 $\pm$ 0.01	1.19 $\pm$ 0.01	1.25 $\pm$ 0.02	1.44 $\pm$ 0.00
	Avg Update [KB]	1.85 $\pm$ 0.03	1.89 $\pm$ 0.04	1.99 $\pm$ 0.05	2.31 $\pm$ 0.02
	Control Overhead [KB]	627.85 $\pm$ 21.20	642.19 $\pm$ 68.12	668.98 $\pm$ 40.80	666.70 $\pm$ 35.10
20	Total [MB]	18.09 $\pm$ 0.12	18.55 $\pm$ 0.14	19.57 $\pm$ 0.12	22.59 $\pm$ 0.14
	Avg Service [KB]	1.19 $\pm$ 0.01	1.23 $\pm$ 0.01	1.31 $\pm$ 0.02	1.49 $\pm$ 0.01
	Avg Update [KB]	1.93 $\pm$ 0.03	1.99 $\pm$ 0.02	2.16 $\pm$ 0.05	2.45 $\pm$ 0.02
	Control Overhead [KB]	529.81 $\pm$ 26.25	519.88 $\pm$ 5.51	477.63 $\pm$ 44.07	538.89 $\pm$ 18.60
30	Total [MB]	17.96 $\pm$ 0.12	18.40 $\pm$ 0.12	19.50 $\pm$ 0.13	22.42 $\pm$ 0.10
	Avg Service [KB]	1.24 $\pm$ 0.01	1.27 $\pm$ 0.01	1.34 $\pm$ 0.00	1.55 $\pm$ 0.01
	Avg Update [KB]	2.08 $\pm$ 0.01	2.12 $\pm$ 0.03	2.24 $\pm$ 0.01	2.64 $\pm$ 0.01
	Control Overhead [KB]	367.85 $\pm$ 21.73	379.32 $\pm$ 36.06	400.88 $\pm$ 22.21	376.98 $\pm$ 9.57

- [32] D. Monderer and L. S. Shapley, "Potential games," *Games and economic behavior*, vol. 14, no. 1, pp. 124–143, 1996.
- [33] Matrix Foundation, "2025 public annual report," <https://matrix.org/foundation/reports/2025%20Public%20Annual%20Report.pdf>, 2025.
- [34] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
- [35] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall, "Graphml progress report structural layer proposal: Structural layer proposal," in *International Symposium on Graph Drawing*. Springer, 2001, pp. 501–512.
- [36] Green Communications, "GreenSoft: Software suite to create an internet of edge on customer hardware," <https://www.green-communications.fr/wp-content/uploads/2025/07/GreenSoft.pdf>, 2025, accessed: 2026-03-10.
- [37] T. H. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626, Oct. 2003. [Online]. Available: <https://www.rfc-editor.org/info/rfc3626>
- [38] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.
- [39] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.
- [40] Z. Tu, H. Talebi, H. Zhang, F. Yang, P. Milanfar, A. Bovik, and Y. Li, "Maxvit: Multi-axis vision transformer," in *European conference on computer vision*. Springer, 2022, pp. 459–479.
- [41] T. maintainers and contributors, "Torchvision: Pytorch's computer vision library," <https://github.com/pytorch/vision>, 2016.
- [42] D. Benyamina, A. Hafid, and M. Gendreau, "Wireless mesh networks design—a survey," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 299–310, 2011.
- [43] M. S. Gilbert, "Synapse-meshsim: Server selection problem," 2025. [Online]. Available: https://gitlab.lip6.fr/ie6/synapse-meshsim/-/tree/server_selection_problem
- [44] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Transactions on networking*, vol. 3, no. 3, pp. 226–244, 2002.
- [45] W. Fischer and K. Meier-Hellstern, "The markov-modulated poisson process (mmp) cookbook," *Performance evaluation*, vol. 18, no. 2, pp. 149–171, 1993.
- [46] H. Heffes and D. Lucantoni, "A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance," *IEEE Journal on selected areas in communications*, vol. 4, no. 6, pp. 856–868, 1986.