Highlights

Effectiveness of Distributed Stateless Network Server Selection under Strict Latency Constraints

Vincenzo Mancuso, Leonardo Badia, Paolo Castagno, Matteo Sereno,
Marco Ajmone Marsan

- Distributed selection of edge vs. cloud servers is viable
- The Nash equilibrium of distributed edge vs. cloud selection is near-optimal
- The Price of Anarchy of distributed edge vs. cloud selection is close to 1
- Experimental validation shows that a distributed allocation converges to the Nash equilibrium in practical contexts

Effectiveness of Distributed Stateless Network Server Selection under Strict Latency Constraints

Vincenzo Mancuso[a,*], Leonardo Badia[b,*], Paolo Castagno[c], Matteo Sereno[c,d], Marco Ajmone Marsan[a]

[a] *IMDEA Networks Institute, Madrid, Spain*
[b] *Dept. of Information Engineering, University of Padova, Padova, Italy*
[c] *Dept. of Computer Science, University of Turin, Turin, Italy*
[d] *Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy*

Abstract

We consider a set of network users (nodes), each generating latency-constrained service requests corresponding to the execution of computational tasks on servers positioned either within a cloud infrastructure or at the network edge. Within this framework, we systematically assess the efficacy of a distributed stateless server selection strategy, strategically performed by individual nodes. Leveraging principles from game theory, our study allows for a comparative analysis between the optimality achieved through globally orchestrated stateless allocation and a decentralized stateless server selection mechanism driven by the self-interested objectives of individual nodes.

Our emphasis on stateless server allocation, rooted in a probabilistic selection framework between edge and cloud servers, stems from prior empirical revelations demonstrating the advantageous outcomes of determining the optimal distribution of edge and cloud tasks based on static network characteristics. Importantly, this determination occurs irrespective of the real-time network state.

The suboptimal nature of the selfish allocation is quantified by the so called "price of anarchy," a metric shown to approximate unity closely. This observation substantiates the justification for a distributed strategic implementation of stateless policies. This elucidation serves as a pivotal guide for crafting algorithms governing server selection, providing a quantitative validation of the efficacy inherent in distributed self-interested approaches.

Keywords: Edge computing, Game theory, Radio access network, Distributed policies, Performance evaluation
2000 MSC: Primary: 68M14; Secondary: 68M20, 91A10

*Please address correspondence to Vincenzo Mancuso or Leonardo Badia
Email addresses: vincenzo.mancuso@imdea.org (Vincenzo Mancuso), badia@dei.unipd.it (Leonardo Badia), paolo.castagno@unito.it (Paolo Castagno), matteo.sereno@unito.it (Matteo Sereno), ajmone@polito.it (Marco Ajmone Marsan)

[1] This is an extended version of [1], published in IEEE MedComNet 2023.

1. Introduction

Future mobile communication networks will provide ubiquitous availability of services based on pervasive storage and computing [1], provided by either resource-abundant computing entities physically located in the network core, or less powerful peripheral servers offering the advantage of proximity to the end user [2].

A fundamental distinction is implied in many studies, and similarly reflected in our investigation, between cloud and edge servers [3, 4, 5]. Under an extreme simplification, cloud computing is generally entangled with longer latency, but also higher capacity than invoking a server located at the edge of the network. Since the ultimate performance experienced by service requests strongly depends on the congestion encountered at the chosen server, under operational network conditions no choice between the two server types dominates the other. Rather, the choices of where to direct individual service requests are interconnected and should be harmonized so as to avoid overloads.

In this paper, we consider a network scenario where users issue service instances that must be completed within a strict deadline. Computing is performed in the network, either at the edge or in the cloud [6].

The network control can exploit multiple criteria to allocate service requests to the more convenient type of server. In [7], we discussed various algorithms, with the objective of investigating how network state information affects the overall management and can be captured through different parameters that lead to optimize network allocation. We performed a comparison between *stateless* and *stateful* policies [8] used to route incoming requests, to either cloud or edge servers. With "stateless" we refer to a policy that relies on network parameters (such as the server capacity and the overall service arrival rate) that are relatively stationary and easy to estimate. Stateless policies do not need to track the status of any client or server. Instead, a "stateful" policy exploits the instantaneous network conditions and tracks the status of servers and/or clients. A key result that goes in favor of a low-complexity implementation of the selection policy is that the performance improvements brought by stateful policies are minimal and also vulnerable to errors in the parameter estimation that can sometimes lead to worse performance with respect to stateless policies [7]. Thus, it might be more convenient to apply simple stateless policies, like the one called `RANDALPH` (randomized alpha) in [7], which translates in the random assignment of a given fraction α of requests to the edge server, while the remaining share $1-\alpha$ is executed in the cloud. The computation of α is centralized, which is proven to be overall efficient in practical contexts.

In [1], we pushed this further by analyzing whether stateless approaches for service request allocation can be made in a distributed fashion, still achieving efficient control, without awareness of the network state that would be expensive to acquire. To do so, we used game theory [9, 10] to solve a distributed selection of a local parameter α_i for each request i, chosen by a strategic agent with only local information driven towards an individualistic objective. This resulted in a game theoretic implementation of the randomized alpha policy, therefore called `GANDALPH` (game theoretic `RANDALPH`). In the same paper we performed a preliminary quantitative comparison of the performance of the centralized policy `RANDALPH` against the distributed policy `GANDALPH`, in particular by considering the globally best assignment of the former and the Nash

equilibrium (NE) achieved by the latter, and to compute the Price of Anarchy (PoA) [11]. We provided analytical justifications about why the PoA is expected to be contained, and we presented results showing that it practically falls within ranges of less than 10%, thereby supporting a fully distributed and scalable implementation of the selection policy.

This paper is an extended version of [1], where we present a much more comprehensive set of results for the comparison of `RANDALPH` against `GANDALPH`, as well as the results of a fictitious play (FP) with two users, which serves to show how the NE can be achieved in practice, with no need to disseminate information on server capacities and on the presence of other users. We have implemented the FP both in an artificial environment using MATLAB, and in a testbed comprising data centers in Italy and Spain.

In a nutshell, the main contributions of this paper are the following.

- We prove that a distributed selection of either edge or cloud servers is viable.
- We show that the NE of edge or cloud server selection is near-optimal.
- We show that the PoA of distributed edge or cloud server selection is close to 1.
- We validate the game theoretic model results through an artificial and over-the-network controlled experimental setups, relying on operational networks.

It is worth noting that in experimental setups the endpoints (i.e., users and servers) are under the experimenter's control, so that the system parameters and workload can be controlled and varied as desired. In the case of artificial experimental setups, we use Matlab simulations in which the behavior of the network connecting end users to servers is controlled by the experimenter. On the contrary, in the case of over-the-network controlled experimental setups, the end users and the servers are connected over the Internet, and are thus subject to load and latency fluctuations deriving from the interactions with whatever traffic shares the same network resources as the experiment data. Experiments executed over the network serve to validate the simple model proposed in this work, and in particular the fact that model assumptions are reasonable.

The rest of this paper is organized as follows. Section 2 addresses related works. Section 3 describes our scenario and how to evaluate a centralized stateless allocation of jobs to edge and cloud servers. We present the distributed allocation modeled as a static game of complete information in Section 4, where we discuss the resulting NE and its properties, which explains why the resulting solution is near-optimal. Section 5 presents numerical results comparing centralized and distributed allocations. Section 6 describes the implemented FP and reports the results obtained over MATLAB as well as in a testbed operated over the Internet. Eventually, conclusions are drawn in Section 7.

2. Related Work

The problem of selecting between cloud and edge servers can also be alternatively termed as *computation offloading* or *request routing*. This issue, or to be more precise, these variations or instances of the problem, have been examined in numerous

studies, including [8, 12, 13, 14, 15, 16]. It's worth emphasizing that this class of problems can be framed within the research stream that compares global (centralized) with individual (decentralized) optimizations [17]. This issue has been investigated across various domains, including communication services, queueing systems, and transport optimization, among others (a non-exhaustive list of contributions includes [18, 19, 20, 21, 22, 23]).

In [3, 24], a taxonomy of different approaches to this problem is presented, according to which a centralized offloading scheme makes decisions about directing traffic to either edge or cloud through a single agent that may or may not use information on the status of the servers, according to which the policy is called either *stateful* or *stateless*.

In [7] we showed that ideal stateful policies, with access to instantaneous information on either of the available servers or both of them, can outperform stateless policies. However, stateful policies are prone to errors, and small uncertainty on the status of the servers dramatically reduces their performance, well below the level that can be achieved by stateless and simple policies. Indeed, the advantage of stateless policies is that they are not affected by state estimate errors by definition. In that work, we only considered centralized policies, whereas here, expanding on [1], we discuss distributed implementations with their specific challenges.

In general, selfish routing leads to performance degradation, and factors like network topology [15] and load [16] play an important role in performance degradation. However, the authors of [25] show via analysis and experiments that the performance loss is minimized if selfish players have little context information, which means that conveying too much path-load context information to service customers is counterproductive when they can make decisions on their own. This justifies why here we focus on stateless routing strategies and do not investigate more complex operational scenarios, although it is known that providing appropriately "curated" and "persuasive" data to routing deciders can improve performance [26, 27].

In all the studies cited previously, including ours, the comparison between the decentralized or selfish approach and the centralized (globally optimized) approach is conducted using the concept of PoA (Price of Anarchy) introduced by [28], which serves as a metric to quantify how much system performance suffers from the lack of regulation.

Contributions that share similar objectives have been investigated in other papers. For example, [29] proposes a scheduling algorithm aimed at minimizing task execution time. In other works, offloading is optimized alongside complementary aspects, such as power allocation [30, 4]. A stateful, distributed, globally optimized solution is presented in [31], while [32] suggests user clustering, enabling a globally optimized solution while addressing scalability issues.

With respect to our previous conference contribution [1], in this extended version we present many more numerical results and design new types of realistic experiments, to show that the performance of the distributed approach of `GANDALPH` comes close to that of the centralized approach of `RANDALPH`. We do this both by computing the NE of `GANDALPH` with game theory approaches, as well as by implementing a fictitious play (FP) that we play either in an artificial MATLAB environment, or in the wild, using two data centers (in Italy and Spain) together with their Internet connections.

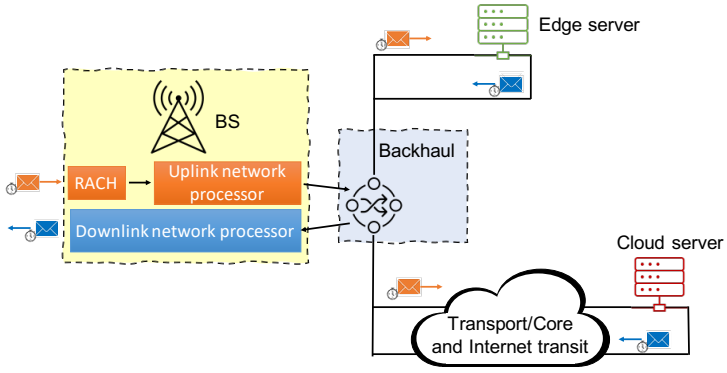


Figure 1: Reference scenario

3. Preliminaries

The structure of the system considered in our investigation is as follows. We take the perspective of a service provider that wants to implement an online computing service for mobile users. The service provider relies on edge and cloud computing premises. We further assume that the network infrastructure offers the tools necessary to instantiate network slices. In other words, we assume that the computing service be implemented through network function virtualization (NFV) and managed through the gears of software-defined networking (SDN).

Following 3GPP standards, we consider that mobile users (UEs) of a network slice are connected through a random access channel (RACH) to a base station (BS) that employs two separate network processors for uplink and downlink messages. The RACH is assumed to use dedicated resources, so it is seen as isolated from other network slices on the same BS. The BS is, in turn, attached to a backhaul (BH), where an edge server is directly connected and can be used for computing tasks requested by the UEs, albeit with limited capacity. Alternatively, tasks can be sent from the BH through the transport network and the Internet to a physically distant but more powerful cloud server [8, 5]. Edge computing resources are potentially convenient for UEs, because they are close to the UEs, although they can cost more than cloud resources. However, the cost incurred by the user is oblivious to the use of edge or cloud servers, since the user pays the operator based on the computing service, not based on where computing happens. Hence, UEs would tend to use the edge, due to its proximity. Since edge resources are typically not sufficient to satisfy the demand of all UEs, some UEs will nonetheless opt for the use of more distant resources at the cloud. Figure 1 illustrates the reference scenario used in this work, with the backhaul represented as a simple logical switch although it could be an optical ring. Transport and core of the cellular network, jointly with the Internet, are symbolically represented as a transit cloud, which may include several high-speed routing devices. Uplink and downlink messages are in different colors because they carry different information.

For simplicity, only one server of each kind (cloud or edge) is considered, but the analysis and the presented expressions can be extended to multiple servers. In particular, to extend the analysis to the case of more than two servers, it is enough to observe

that the workload of UEs splits across servers according to a vector of service selection probabilities whose values sum to one. In the two-server case, such a vector is simply given by $[\alpha_i, 1-\alpha_i]$, being α_i the probability that UE i selects the edge server. This means that we can discuss the two-server case considering just one scalar per UE, and avoid complicating the notation with vectors in this manuscript. Notice also that performance metrics like loss or failure probabilities will be expressed as average values over the loss or failure observed at any of the available servers. Also, our model allows for inserting background traffic whenever needed. This would be necessary whenever a quantitative comparison with practical implementations is performed because the presence of background traffic is unavoidable in any real-world scenario, such as our testbed, and sometimes its impact is significant [33]. However, in the present evaluations, the background traffic does not affect the discussion that we bring forward related to the distributed implementation of the server selection policies.

UEs are assumed to generate service requests with rate λ_u. If these requests are directed to a server whose capacity is saturated, then the request is discarded, and we consider this to be a *loss* event. Moreover, we impose a delay constraint on the execution of the request so that a *failure* can occur both if the request is discarded and if a non-discarded request violates the timeout condition (i.e., if the result of the computation associated with the request does not reach the UE before the timeout expiration). This means that failures happen with probability $\mathsf{P}_{\mathrm{fail}}$ that is greater than the loss probability $\mathsf{P}_{\mathrm{loss}}$ as failures are a broader condition. We also distinguish between $\mathsf{P}_{\mathrm{fail}}^{(\mathrm{E})}$ and $\mathsf{P}_{\mathrm{fail}}^{(\mathrm{C})}$ (or analogously $\mathsf{P}_{\mathrm{loss}}^{(\mathrm{E})}$ and $\mathsf{P}_{\mathrm{loss}}^{(\mathrm{C})}$) as the conditional probability of failures (or losses) of requests that have been directed to the edge or to the cloud server, respectively.

The foundation of our analysis is to characterize all the system components as work-conserving FIFO queues and therefore evaluate the experienced latency as the convolution of the delay terms at each step. If we represent all the delay terms of these individual components through the Laplace-Stjeltjes transform (LST) of their probability distribution function (pdf), the resulting overall delay has an LST obtained by a chain multiplication of all terms.

Let α_i be the probability that a request of UE i is sent to the edge server. Hence, the pdf of the delay of a request that is not lost, has an LST $\widehat{f}_T(s)$ that can be written as

$$\widehat{f}_T(s) = \widehat{f}_{\mathrm{R}}(s)\,\widehat{f}_{\mathrm{L_u}}(s)\,\widehat{f}_{\mathrm{L_d}}(s)\,\widehat{f}_{\mathrm{B_u}}(s)\,\widehat{f}_{\mathrm{B_d}}(s)\,\frac{1}{1-\mathsf{P}_{\mathrm{loss}}} \cdot \left(\alpha_i\left(1-\mathsf{P}_{\mathrm{loss}}^{(\mathrm{E})}\right)\widehat{f}_{\mathrm{E}}(s) + (1-\alpha_i)\left(1-\mathsf{P}_{\mathrm{loss}}^{(\mathrm{C})}\right)\widehat{f}_{\mathrm{T_u}}(s)\,\widehat{f}_{\mathrm{T_d}}(s)\,\widehat{f}_{\mathrm{C}}(s)\right), \quad (1)$$

where subscripts indicate individual sources of delay: the RACH (R), the link between the BS and the network processor ($\mathrm{L_u}$ and $\mathrm{L_d}$ for uplink and downlink, respectively), the BH connection ($\mathrm{B_u}$ and $\mathrm{B_d}$ for uplink and downlink, respectively), the transport network ($\mathrm{T_u}$ and $\mathrm{T_d}$ for uplink and downlink, respectively), and the time spent in the edge or in the cloud server (E and C, respectively). With the above, the overall distribution of latency incurred by requests depends on the load of each system component. Therefore, it is straightforward to use the above formulation either by accounting or not for the intensity of background traffic.

The failure probability P_{fail} of UE i is the sum of the loss probability and the probability that service exceeds the timeout. The former is computed as a weighted average over edge and cloud servers, i.e.,

$$\mathsf{P}_{\text{loss}} = \alpha_i \mathsf{P}_{\text{loss}}^{(\text{E})} + (1-\alpha_i)\mathsf{P}_{\text{loss}}^{(\text{C})}, \tag{2}$$

The probability that service exceeds the timeout is given by $1 - F_T(T_O)$. As a result, the failure probability of UE i has the following expression:

$$\mathsf{P}_{\text{fail}} \;\; = \;\; \mathsf{P}_{\text{loss}} + \big(1-\mathsf{P}_{\text{loss}}\big)\big(1-F_T(T_O)\big) = 1 - F_T(T_O)\big(1-\mathsf{P}_{\text{loss}}\big), \tag{3}$$

where T_O is the timeout value and $F_T(t)$ is the cumulative distribution function of the latency computed as the inverse LST of $\widehat{f}_T(s)/s$ via numerical techniques.[2] Notice that, as described in [7], loss at edge and cloud, and timeout probabilities, depend on the average of α_i across all UEs.

Now, we sketch how these components can be modeled, with a more detailed analysis that can be found in [7]. The RACH model is taken from [33] and considers that user requests experience a delay due to the transmission over the channel, possibly encountering collisions and subsequent backoffs. We take a sufficiently high number of allowed retransmission attempts to prevent the RACH from losing service requests by itself. However, losses can occur due to the capacity of the servers being saturated, as previously discussed. Thus, from [7] we get the following approximate expression for the LST of the sojourn time in the RACH:

$$\widehat{f}_R(s) = \frac{1-e^{-s(T_x+W_x)}}{s(T_x+W_x)} \sum_{i=1}^{k_x} \frac{1-e^{-i}}{e^{i(i-1)/2}} \left(\frac{e^{-sT_x}}{1+\bar{\tau}_B} \right), \tag{4}$$

where T_x, W_x, k_x, and $\bar{\tau}_B$ are RACH parameters corresponding to the maximum time to reply to a RACH request, the maximum time to establish a connection after a RACH exchange, the maximum number of transmission attempts, and the average backoff time in case of collisions. In (4), we consider a traffic-independent expression for the probability that the RACH request is successful in i attempts, which holds true if RACH losses are negligible and we adopt power ramping over multiple transmission attempts.

From the implementation standpoint, the BH consists of individual links and dedicated resources interconnecting the BS, the edge server, and the transport network that bridges to the cloud. The BH can be considered reliable, i.e., not introducing any loss, and with high capacity. Both the edge and the cloud servers are assumed to run on virtual machines (VMs), whose numbers are n_{E} and n_{C}, respectively. Service requests coming from UEs can be allocated to one VM equivalently located in either server, after being queued in a buffer with finite capacity. The maximum number of requests

[2]The division by s in the LST domain corresponds to an integration in the probability domain, so that while by inverting $\widehat{f}_T(s)$ one would obtain a probability density function, by inverting $\widehat{f}_T(s)/s$ we will obtain the corresponding cumulative distribution function.

that can be held (either queued or undergoing service) at the edge or cloud server is k_{E} and k_{C}, respectively. Such number represents the capacity of the queue. Hence, a loss event happens if the number of allocated requests exceeds the queue capacity.

We model the network processor as an M/D/1-PS queue, because time-frequency resources of a base station are shared among all active transmissions, which can occur in parallel. All the intermediate queues (backhaul and transport networks) are more simplistically modeled as M/M/1 queues because of the serial nature of transmission over many wired link technologies [24]. For all these systems, we just repeat this model for both uplink and downlink directions. Finally, the edge and cloud servers are represented as M/M/n_{E}/k_{E} and M/M/n_{C}/k_{C} queues, respectively.[3] The multi-server nature of edge and cloud queues reflects the fact that they are essentially computing stations with multiple processors available for service [6]. This promptly gives us expressions for the LST terms in (1). Additionally, the loss probabilities $P_{\mathrm{loss}}^{(\mathrm{E})}$ and $P_{\mathrm{loss}}^{(\mathrm{C})}$ are computed from the probabilities that the buffer at the queues is full, that is, the queue is occupied by k_{E} or k_{C} requests, respectively.

Note that we model the (uplink and downlink) network processor delays as deterministic because we consider the case of a lightly loaded BS slice. On the contrary, we model computing times at both the edge and cloud servers with exponential distributions because of their intrinsic variability and because of our focus on the latency introduced by those computing elements.

4. Game theoretic model

The idea of `RANDALPH` is to set the same $\alpha_i = \alpha$ for all service requests. This value is computed in a centralized way as a global optimum so that all service requests are randomly directed to either the edge or the cloud server with respective probabilities α and $1-\alpha$, without constant monitoring of the network state. This centralized globally-optimized stateless approach is suitable for a simpler implementation, whose efficiency is still comparable to more complex stateful policies [7].

However, in light of the drawbacks of a centralized computation, we want to explore a game theoretic approach to determine whether the α–values can be efficiently computed through a distributed evaluation by individual UEs seen as selfish (i.e., strategic) agents. With our approach, selfish UEs would only be required to adjust their server selection probability α_i so as to minimize their own failure probability. Adjusting α_i might be implemented with or without explicit notifications sent by servers. We opt for the latter operational style for the following reasons. First, implementing server notifications would require a modifications at *both* servers and clients and the implementation of a message exchange. Second, how, when and what should be reported by servers in order to obtain a robust protocol behavior would require further design, analysis and evaluation, which goes beyond the scope of this paper. Third, UEs could seamlessly infer how to adjust their α_i by observing their failure probability.

[3] Note that this implies an exponential assumption for service times at both the edge and cloud servers. This assumption will be validated through a comparison against the results of an experimental setup where service times are taken to be deterministic.

This results in the game theoretic procedure that we named GANDALPH. The differences with the original centralized global optimization approach are subtle but essential. Firstly, we consider an individual value α_i for each service request i. Since we consider continuous α-values, it does not really matter if i identifies the atomic service request or the UEs as this can be framed as either a fine-grained choice of individual tasks or a probabilistic assignment of all the tasks of the same user.

Moreover, we do not assume any communication exchange between the agents, so they all act independently. In the jargon of game theory, we consider a *static game of complete information* [9, 19], where all the UEs are equally aware of the network parameters but cannot communicate their choices. This sets a difference with other game theoretic approaches based on auctions or bargaining [22, 34], in that our simpler implementation does not require any subsequent interaction. Finally, the choice of each agent is selfish, i.e., just driven toward the minimization of the failure probability $\mathsf{P}_{\mathrm{fail}}$ experienced by that player.

Notably, under this approach all users will still choose the same α for symmetry reasons. However, the rationale of RANDALPH and GANDALPH is different. In the former, α is a value chosen to be the same by a central optimizer so as to achieve the best performance. In the latter, each strategic player chooses α_i so as to optimize a *selfish* goal while at the same time being aware that all other players will do the same [11]. This results in a local optimum from the individual player's perspective, and NEs are found as the points without incentives for unilateral deviation towards another α_i (while the other α_j, $j \neq i$, are kept as they are).

The analysis summarized in the previous section strongly motivates this approach to be sensible. In practice, the failure probability $\mathsf{P}_{\mathrm{fail}}$ that a single player can expect to get depends on α_i through a dependence that, avoiding the analytical intricacies, can be summarized as (2), which uses the total probability on the conditions of choosing the edge or the cloud server. For a selfish player, the best choice of α_i happens where the first derivative of $\mathsf{P}_{\mathrm{fail}}$ is zero, but it is immediate to see that the failure probability of a player is the average failure observed over the two available servers, so that

$$\begin{aligned}\frac{\partial \mathsf{P}_{\mathrm{fail}}}{\partial \alpha_i} &= \frac{\partial}{\partial \alpha_i}\left(\alpha_i \mathsf{P}^{(\mathrm{E})}_{\mathrm{fail}} + (1-\alpha_i)\,\mathsf{P}^{(\mathrm{C})}_{\mathrm{fail}}\right) \\ &= \mathsf{P}^{(\mathrm{E})}_{\mathrm{fail}} - \mathsf{P}^{(\mathrm{C})}_{\mathrm{fail}} + \alpha_i \frac{\partial \mathsf{P}^{(\mathrm{E})}_{\mathrm{fail}}}{\partial \alpha_i} + (1-\alpha_i)\frac{\partial \mathsf{P}^{(\mathrm{C})}_{\mathrm{fail}}}{\partial \alpha_i}. \end{aligned} \tag{5}$$

Therefore, equalizing the failure probability of edge and cloud, e.g., with a load balancing approach, is not necessarily enough to reach a NE. However, if failure probabilities of edge and cloud can be equalized without incurring server saturation, partial derivatives will be close to zero. In this case, (5) tells that the load balancing point will be close to the NE. Hence, the analysis reveals that the NE would lead the system to operate almost like in a centralized load balancing scenario when the system load is low. Under those circumstances, load balancing is indeed near-optimal and very robust [7]. Instead, the operating point would progressively deviate from a balanced assignment as the load increases and the partial derivative terms in (5) become predominant. This must not surprise because, at high load, a load balance policy cannot work well as it would fairly lead all jobs to fail.

In general, the partial derivative of the failure probability must be positive for the edge at any α_i, while at the cloud it must be negative. Thus, the difference $\mathsf{P}_{\text{fail}}^{(\text{E})} - \mathsf{P}_{\text{fail}}^{(\text{C})}$ is a monotonic increasing function of α_i, ranging from a negative value $-\mathsf{P}_{\text{fail}}^{(\text{C})}|_{\alpha_i=0}$ to a positive value $\mathsf{P}_{\text{fail}}^{(\text{E})}|_{\alpha_i=1}$. The weighted sum of the partial derivatives in (5) has a similar behavior, but it goes from a positive to a negative value instead. These considerations support the uniqueness of the NE as a point where the failure probabilities at the cloud and the edge must be different, unless edge and cloud servers have the same characteristics. Besides, the role of the partial derivatives is important to determine which server (edge or cloud) must experience a higher loss rate at the NE.

A key observation from the results of [7] is that P_{fail} is relatively flat around its minimum in practical situations. This implies that: (i) a local search from an individual standpoint is likely to achieve near-optimal values of α_i; (ii) even if α_i is not exactly chosen by each individual user as the optimal value of α for `RANDALPH`, the resulting P_{fail} is still likely to be near-optimal.

Moreover, we are interested in obtaining a small average failure probability, otherwise performance cannot be good and the system operation becomes pointless. Thus, at least one server must be far from saturation, and if one server approaches saturation, it must receive just a little portion of traffic. Hence, all terms of (5) are small or slowly changing with α_i, which explains why P_{fail} is flat around its minimum. In particular, the last term describes the influence of a single request on the failures in the cloud server, which again is sensible to assume to be minimal. The same can hold for the third term but, while we can argue that the edge server can be more sensitive, we remark at the same time that if this is the case, it is also likely that α_i is small, thereby causing the term to be close to 0.

As argued above, the NE of `GANDALPH` is necessarily unique because of the monotonic trend that the failure probability of any selfish user must have if it deviates from the behavior of the other UEs – in particular, it grows if α_i goes to either 0 or 1, meaning that only either the cloud or the edge server is used. To quantitatively juxtapose the approaches, we can compare the two values of α under a centralized-global optimization or distributed-selfish management, i.e., the optimal value chosen by `RANDALPH` and the NE of `GANDALPH`. A further comparison can quantify the impact that a possible difference of these α values has on the failure probability. Thus, we compute the PoA [11, 28] as $\text{PoA} = \mathsf{P}_{\text{fail}}^{\text{Nash}} / \mathsf{P}_{\text{fail}}^{\text{Coordinated}}$ with a clear meaning of the superscripts.

5. Evaluation

We designed a set of simulations and real network experiments to compare coordinated globally-optimal decisions and selfish routing decision strategies in the reference scenario considered in this paper. In particular, we analyze and compare performance figures in terms of failure probability P_{fail}. The difference between `RANDALPH` and `GANDALPH` will be expressed in terms of PoA. We will also show how the two different approaches result in routing strategies that can diverge substantially.

5.1. Coordinated optimum and NE search

The optimal configuration for RANDALPH is the value of edge routing probability α, commonly adopted by all users, that minimizes the failure probability $\mathsf{P}_{\mathrm{fail}}$. Since this optimization problem is not convex in general, as we will show with an example at the beginning of the numerical evaluation subsection, we resort to a discretized search. In practice, we run a brute force search with resolution 10^{-3} on the value of α. The search requires a few hours of computation of a dedicated 3 GHz processor. We do not optimize the search of a coordinated optimum, because the object of the work is not optimization per se, rather the evaluation of the performance and the PoA of a distributed implementation of the server selection [17].

For what concerns the NE, using a brute force search on condition (5) is impractical, since the computation of partial derivatives is prone to numerical errors. It would in any case involve the computation of several values of failure probability for each candidate value of edge routing probability, so as to be able to confidently estimate the partial derivative function. Instead, we resort to design a search algorithm which is similar to the well known binary search. To explain how our algorithm works, consider that $\mathsf{P}_{\mathrm{fail}}$ observed by a selfish user has a single minimum as her routing α_i changes, while the rest of users do not change their strategy. To see that, consider what can occur when the traffic of the selfish user is routed in full to the cloud. The failure probability at the edge, $\mathsf{P}^{(\mathrm{E})}_{\mathrm{fail}}|_{\alpha_i=1}$ can be smaller, equal, or larger than the failure probability at the cloud, $\mathsf{P}^{(\mathrm{C})}_{\mathrm{fail}}|_{\alpha_i=1}$. In the first case, offloading some traffic from cloud to edge is beneficial, although beyond some point the offload can become counterproductive, because $\mathsf{P}^{(\mathrm{E})}_{\mathrm{fail}}$ can only increase while $\mathsf{P}^{(\mathrm{C})}_{\mathrm{fail}}$ can only decrease. The last case is similar, but this time the offload cannot help and the minimum is observed at $\alpha_i = 1$. If instead the failure probabilities are identical, we would need to compare the partial derivatives, to see which failure probability changes faster, and would reach the same conclusions as in the other two cases. In all cases, the value of $\mathsf{P}_{\mathrm{fail}}$ observed by a selfish user has a single absolute minimum value for $\alpha_i \in [0, 1]$.

From the above considerations, we infer that a selfish user would only deviate her strategy toward a given direction, and in so doing, she would greedily find her best routing probability. Moreover, as users have the same characteristics and are rational, in a distributed implementation scenario they would all have the same incentive to move like the selfish users we have taken as reference in the discussion so far. This means that all users would move toward the same direction and iteratively converge to a NE point. In particular, when we have identified the direction that a selfish user would take, we can assume that the entire set of users will move in the same direction.

We therefore implement the following search for the NE configuration. We start by considering a candidate interval for α and take the central value of the interval as the current routing probability of all users. The initial interval can simply be the entire space of strategies, i.e., $[0, 1]$. We then split the interval into three equally sized adjacent sub-intervals and compute the average value of the failure probability of a selfish user under the hypothesis that she can decide to move toward a point in any of the three intervals. We only take a few (3 to 5) evenly spaced samples per sub-interval, to compute averages and identify the smallest of them. This shows in which direction a selfish user would move if starting from the center of the interval. Since, as previously

Table 1: Parameters used in the numerical evaluation

Parameter	Notation	Default value
Number of UEs	n_u	50
Service request rate per UE	λ_u	60 s^{-1}
Number of servers at edge	n_{E}	1
Buffer size at edge	k_{E}	10 requests
Number of servers at cloud	n_{C}	10
Buffer size at cloud	k_{C}	50 requests
Average request service time	μ^{-1}	5 ms
Round trip time from UE to edge		26.955 ms
Round trip time from UE to cloud		52.498 ms
Number of RACH preambles		54
Max number of RACH retransmissions	k_x	10
RACH retransmission timeout		10 ms
Uplink packet size		2000 b
Downlink packet size		4000 b
Uplink radio slice capacity		10 Mb/s
Downlink radio slice capacity		25 Mb/s
Uplink BH slice capacity		20 Mb/s
Downlink BH slice capacity		35 Mb/s
Core network slice capacity		100 Mb/s
Service timeout	T_O	100 ms

noted, the rest of users would follow the target selfish user, we can next repeat the procedure by considering a narrower search interval that corresponds to either (a) the right half of the original interval if the selfish user had an incentive to increase α_i, (b) the left half is the incentive was toward decreasing α_i, or (c) a half-sized interval centered on the current value of α_i otherwise. Using three partially overlapped intervals rather than two separate intervals makes the search robust to numerical errors which might appear when dealing with failure probability values of the order of 10^{-6} or less, and adjusting the routing probability by 10^{-3} or smaller steps. Those errors might otherwise cause the search to remain stuck in the wrong interval, which is relevant for cases in which the failure probability is relatively flat and can be small as required by many commercial applications. The procedure continues until the search interval size becomes smaller than a threshold (10^{-4}, in the numerical results shown in what follows). The middle point of the last identified interval is considered as the NE point, and the associated failure probability is computed.

5.2. Evaluation scenario

To illustrate the behavior of `GANDALPH`, we evaluate its performance in the concrete scenario described next, which allows us to evaluate the impact of several parameters like the system load and the granularity at which UEs contribute to the system load, the distance of the cloud service and its capacity. Notation and default parameters used in the numerical evaluation are indicated in the following text and are summarized in Table 1.

Workload. We consider a radio access network (RAN) slice accessed by n_u UEs, each issuing λ_u requests per second (req/s), on average. In the plots, we use the value of the system load ρ, which is the workload of the system, calculated as the ratio of the aggregate offered traffic $n_u\lambda_u$ divided by the total service capacity of the system $(n_{\mathrm{C}}+n_{\mathrm{E}})\mu$, which in turn depends on the number of VMs in the edge and cloud servers (n_{E} and n_{C}, respectively), and their capacity μ.

Request processing. All requests go through the RACH procedure for which they use 54 orthogonal preambles with a RACH opportunity every millisecond. The maximum number of RACH transmission attempts is $k_x = 10$, enough to guarantee full reliability of the channel, since users adopt a power ramping algorithm to progressively increase the transmission power of their RACH preambles after each failure. Retries are spaced according to random backoff times with average duration 10 ms. With the adopted configuration, the probability that the RACH will cause a request loss is below $3 \cdot 10^{-18}$ and can be therefore neglected. After success on the RACH, requests are sent in packets of 2000 bits and, as discussed in Section 3, served by a network processor at the BS, modeled as an FCFS queue. After that, requests move to another queue representing a BH from which requests are dispatched to either the edge server or to another queue representing the core network segment between the BH and the cloud server. The mentioned queues have infinite buffer space, and the RACH does not cause losses, so that all requests eventually reach either the edge or the cloud server. Downlink transmissions from the edge or cloud server follow the respective inverse path, and are sent in individual packets of 4000 bits.

Edge server. All UEs see the same distance to an edge server, since users are connected to the same BS, hence access the same backhaul. The edge server runs a single VM, modeled as an $M/M/1/k_{\mathrm{E}}$ queue with a finite buffer space ($k_{\mathrm{E}} = 10$ requests – one in service and up to 9 waiting – in the numerical evaluations) with an exponential service at rate $\mu = 200$ requests per second. In other words, a request takes on average 5 ms to be served, thus representing short jobs typical of mobile applications that require network assistance for parsing the local context and making informed decisions, e.g., in assisted driving applications, steering of UAV fleets, and so on [35]. The round-trip-time (RTT) between UEs and the edge server is the same as between UEs and BH, which we set to 24.875 ms, plus an almost negligible extra delay in accessing the edge server from the BH (we use 0.08 ms as the time needed to cross the BH) plus 2 ms to account for internal edge data center latency. The resulting RTT is 26.955 ms. These RTT values are the ones observed in a testbed built to evaluate `RANDALPH` in [7].

Cloud server. The cloud is an $M/M/n_{\mathrm{C}}/k_{\mathrm{C}}$ server with $n_{\mathrm{C}} = 10$ servers (each akin to the edge server) and $k_{\mathrm{C}} = 50$ maximum requests by default (including requests under service), although we also explore other values in specific experiments shown later. The cloud is reachable through the same BH, and the RTT between the BH and the cloud is set to 24.543 ms unless otherwise specified (again, we use the values observed in [7]). We also add 1 ms to account for internal delay in the cloud data center. Therefore, the total RTT between UEs and cloud is $26.955 + 24.543 + 1 = 52.498$ ms, without accounting for queueing and processing at the cloud.

Network slice. The capacity of the RAN slice radio link is 10 Mb/s in uplink and 25 Mb/s in downlink, representing a BS slice dedicated to the considered service. The BH

capacity is 20 and 35 Mb/s in uplink and downlink, respectively. These values consider a service that accesses only a slice of the backhaul resources. The core network slice capacity is 100 Mb/s in both uplink and downlink.

Application timeout. With these system parameters, the performance bottleneck is at the edge and cloud sites, while network elements only cause random delays. This implies that system performance is determined by edge/cloud losses and service timeouts. The value of the latter is set to $T_O = 100$ ms, in line with the requirements of real-time services for autonomous driving, online gaming, and augmented reality, just to mention a few use cases [35, 36].

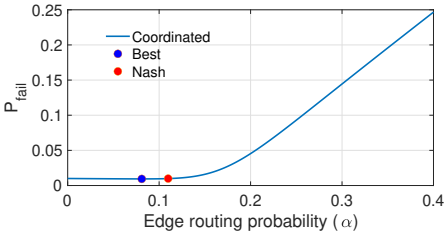
5.3. Numerical evaluation

We evaluate the NE of GANDALPH as a function of the system load ρ, the number of cloud servers n_{C} and the round-trip-time distance of the cloud from the backhaul, as well as the number of UEs in the system. We compare the NE to the optimum α computed by a centralized strategy that minimizes the failure probability. The NE is numerically found as the value of a common edge selection probability $\alpha_i = \alpha_{\text{Nash}}$ for which the individual failure probability does not improve by selfishly deviating from it, according to the procedure described in Section 5.1. Numerical results produced by using MATLAB consist in loss probabilities computed as overflow at servers' queue, with queueing theory formulas. They also consist in probabilities of exceeding the timeout, computed by inverting the LST of RTT. LST expressions take into account the load partition between edge and cloud imposed by the chosen α_i. Losses and timeouts are put together by means of (3), thus yielding the failure probability. Our MATLAB code needed to compute the NE and to search for the optimal coordinated strategy, given a set of network and traffic parameters, has been publicly released on GitHub.[4] Since with a centralized approach or at equilibrium all users adopt the same value of α, and because we only consider GANDALPH at the NE, in what follows we drop the index i from the notation.

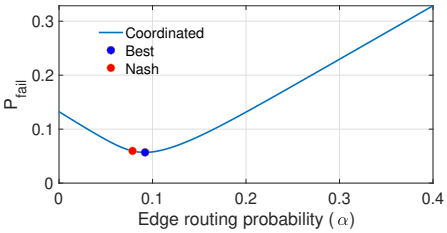
Table 1 summarizes the default parameters used in the numerical evaluation.

We start by showing that the optimization of RANDALPH is not necessarily a convex problem and that multiple local minima can exist. Figure 2 shows the case with default parameters, with three different values for the load. The figure zooms into the interval in which the optimum can be found, which is marked with a blue dot on the curve of P_{fail} vs α. Figure 2a illustrates a case with low/medium load ($\rho = 0.5$), in which the NE, marked with a red dot, is found at a higher value of the edge routing probability than at the optimal point. In this case, the failure probability is practically flat until α grows to saturate the edge server, after which we can observe a linearly increasing failure probability, corresponding to the linear increase in loss at the edge server due to its overload. Different is the case of Figure 2b, with $\rho = 1$, for which we observe that the NE occurs at a lower edge routing probability than at the coordinated optimum, and the region in which the optimum is found is not flat. In both cases, with $\rho = 0.5$ and $\rho = 1$, the optimization is convex (there is a single local minimum). Convexity is, however, not guaranteed, as shown in Figure 2c. In this case, the failure probability is

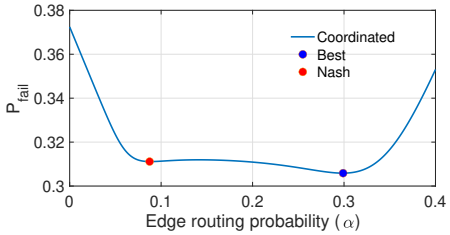
[4] https://github.com/VMancuso/gandaph



(a) At $\rho = 0.5$ the function is convex and the NE is reached at a value of α higher than the optimal



(b) At $\rho = 1$, the function is convex and the NE is reached at a value of α lower than the optimal



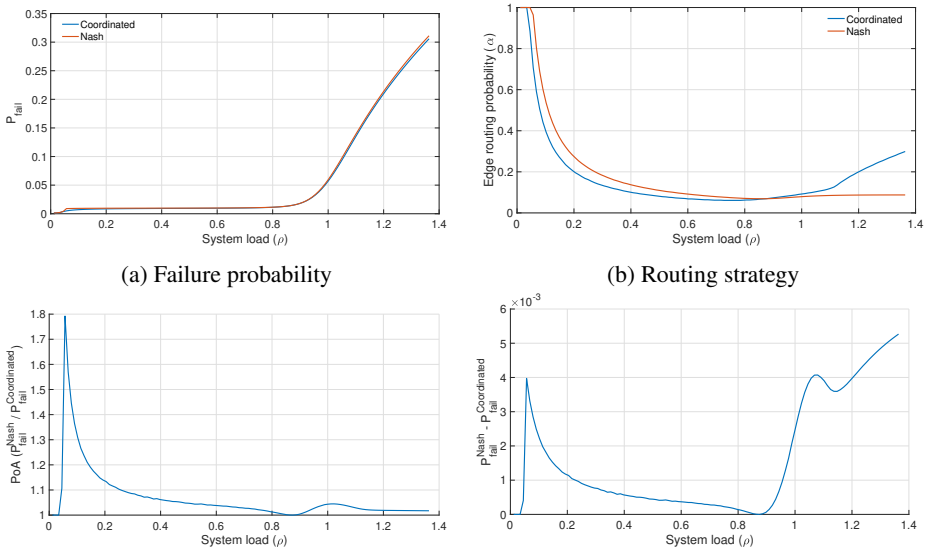
(c) For $\rho > 1$, there is a non-convex dependency of failure probability on edge routing probability ($\rho \simeq 1.36$)

Figure 2: Example of failure probability dependency on edge routing strategy obtained with $n_u = 50$ users and network parameters as in Table 1. "Coordinated" indicates results achieved with RANDALPH. "Best" and "Nash" mark the optimal coordinated strategy and the NE found with GANDALPH, respectively.

somehow flat over a large portion of the interval, which makes the search tedious. Two local minima are clearly visible, and the NE point is close to one of them, although unfortunately not the one giving the absolute minimum. Here, to clearly show non-convexity, we considered a case with very high load. Indeed, the total offered traffic is $n_u \, \lambda_u = 3000$ req/s, while the aggregate service capacity of edge and cloud servers is $(n_C + n_E)/0.005 = 2200$ req/s, i.e., $\rho = 1.36$. This explains why the failure probability observed in the figure is high. In particular, since at least $3000 - 2200 = 800$ req/s cannot be served (without considering timeouts), the bare minimum loss we observe in the system is $800/3000 = 0.267$. Values reported in the figure, just above 0.3, are therefore not surprising.

It is important to notice that differences in failure probabilities at the two minima of Figure 2c and the NE are however quite small. In general, in all the cases described in Figure 2, the failure probability at the NE is very similar to the one at the coordinated optimum.

For a more comprehensive analysis, we next consider a more extensive range of loads that implies also more reasonable failure probabilities through the results illustrated in Figure 3 for coordinated optimization and NE. Results were obtained through the centralized solution (i.e., RANDALPH) or the NE where the edge routing probabil-



(a) Failure probability

(b) Routing strategy

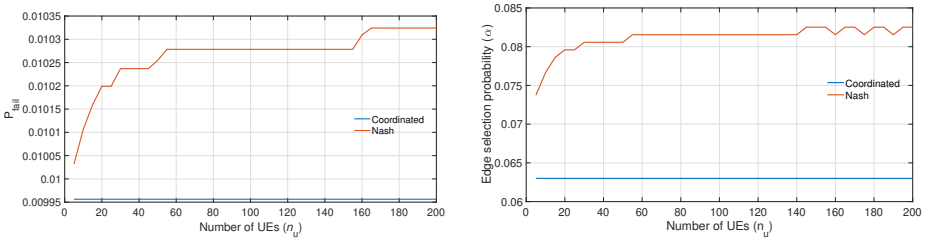
(c) Price of anarchy

(d) Difference between failure probabilities

Figure 3: Performance achieved with optimized RANDALPH (Coordinated) and GANDALPH (Nash) with the parameter values in Table 1 and variable system load (obtained by varying λ_u only).

ity is chosen by each UE (i.e., GANDALPH) under the default system configuration of Table 1, except for λ_u, which varies from 0 to 3000 req/s ($\rho \leq 1.36$).

In particular, Figure 3a reports the failure probability as a function of load. Both coordinated and Nash approaches achieve low and comparable failure probabilities as far as the system load is below 1. In general, we can see that the two selection policies achieve very close results. Figure 3b shows that the slightly worse performance at the NE is due to the fact that the Nash approach leads to higher edge routing probabilities for low loads. This happens because, at low load, the edge server can handle the majority if not the entire offered load, hence selfish users tend to opt more for the edge server. The two curves of α have a similar trend, with just a slight divergence for values of the system load ρ higher than 1, which have limited relevance. Figure 3c illustrates how the PoA (i.e., the ratio of the two curves for $\mathsf{P}_{\mathrm{fail}}$) evolves as the load increases. We can notice a peak of 1.8 at about $\rho = 0.1$, whereas most of the values of ρ yield PoA values which are extremely limited, always below a decrease of efficiency of 10% for $\rho > 0.25$. The high peak of PoA at $\rho = 0.1$ is occurring at very low failure probabilities and so it is not very relevant. Indeed, Figure 3d shows that the maximum difference between failure probabilities at NE and optimum at the PoA peak is below 0.004, which can be considered negligible. More interesting is the fact that the PoA attains its lowest values for a system load value close to 90%, which is a more realistic operation point. This consideration, jointly with the observation that the maximum difference in the considered range is below 0.006, indicates that a distributed coordination, with GANDALPH, promises to be almost as efficient as a centralized coordination,



(a) Failure probability

(b) Routing strategies

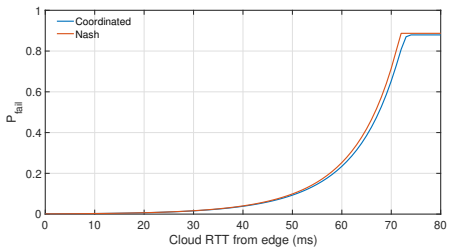
Figure 4: Performance with optimized RANDALPH (Coordinated) vs. GANDALPH (Nash) with a variable number of UEs at fixed aggregate traffic of $n_u\lambda_u = 1500$ req/s, and the default parameter values in Table 1 ($\rho = 0.682$).

achieved through RANDALPH.

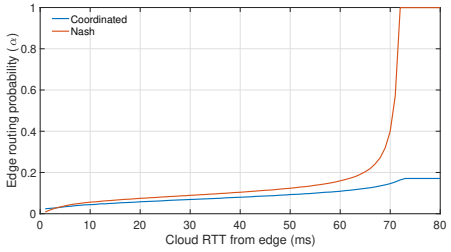
Figure 4 investigates the role of the granularity in the amount of traffic produced by a single UE. In particular, the figure shows results for a fixed total arrival rate of $n_u\lambda_u = 1500$ req/s, i.e., a medium-high system load ($\rho = 0.682$), as the number of UEs that generate the load increases and hence the importance of each user on the system dynamics progressively vanishes. The rest of parameters take their default values. As can be seen in Figure 4a, the "size" of the UE (i.e., $\lambda_u = 1500/n_u$) has little impact on failure performance, which empirically confirms that using α as the probability to route a single service request of a UE does not matter, especially when the number of UEs is high. To explain the result, note that only the aggregate traffic matters for RANDALPH, as also confirmed by the plot of the edge routing probability shown in Figure 4b. Instead, the efficiency of GANDALPH slightly decreases when more UEs are considered (each with a smaller size so that the aggregate traffic is the same), and the edge routing probability slowly increases. This follows from the principle known as the tragedy of the commons [11], i.e., the system efficiency decreases when the role of the individual in the community is less impactful, as selfish behaviors are encouraged. In this case, however, the efficiency loss is just marginal (note the values on the vertical axis). The oscillation in the curve of α at the NE in Figure 4b are due to the fact that we have set the resolution of the search for the optimal α to 10^{-3}.

Figure 5 shows the impact of the distance to the cloud, whose increasing value deteriorates performance, because it causes more failures. Here, we use $n_u = 60$ and $\lambda_u = 250$ req/s, and variable RTT between the cloud and the BH, while the rest of parameters take their default values. Notice that the system load in the plots of Figure 5 is the same as in Figure 4.

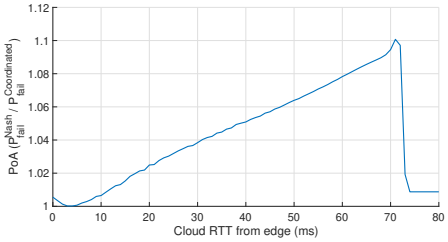
As illustrated in Figure 5a, the distance between optimum and NE keeps increasing when RTT increases, until both curves reach a plateau. Indeed, the inspection of the edge routing probability, through Figure 5b, shows that the selfish allocation of GANDALPH pushes up the edge routing probability α, i.e., the probability that the edge server be selected, up to 100%. This value is reached when requests sent to the cloud are often violating the timeout, which occurs almost certainly as soon as the edge-cloud RTT approaches 73 ms, the RTT between UE and edge being just below 27 ms, and the timeout 100 ms. The distributed assignment GANDALPH is not particularly worse than



(a) Failure probability



(b) Failure probability



(c) Price of anarchy

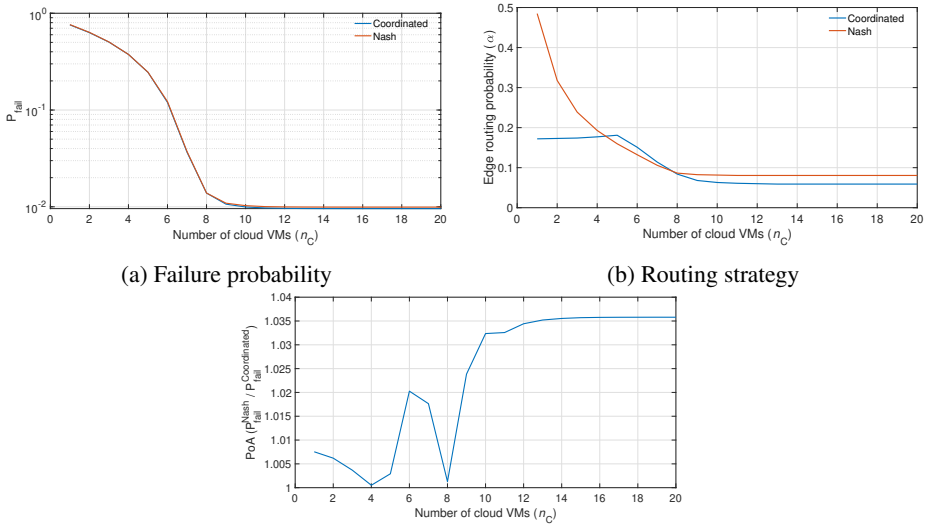
Figure 5: Performance of optimized centralized RANDALPH (Coordinated) and GANDALPH (Nash) vs the distance of the cloud from the edge, with $n_u = 60$ UEs, $\lambda_u = 250$ req/s ($\rho = 0.682$) and other default parameters.

the original centralized RANDALPH, once again keeping the loss of efficiency within 10%, as reported in Figure 5c. That figure also shows that the PoA diminishes at very high delay. This is not surprising because when the RTT is too high, no strategy can lead to good performance. In that case, almost all requests fail, either (i) because they are routed to the cloud and violate the timeout constraint, as done by the coordinated-global optimization approach, or (ii) because of the overflow of the edge queue to which they are routed by the distributed-selfish approach.

Finally, the role of the cloud capacity is explored in Figures 6–7. In all plots, the capacity of the cloud is changed by varying the number of VMs (n_C). Figure 6 considers a fixed offered traffic ($n_u = 60$, $\lambda_u = 250$ req/s, while ρ changes with n_C), whereas Figure 7 considers a fixed number of UEs and system load ($n_u = 60$, $\rho = 0.682$, while λ_u changes with n_C).

In these experiments, the buffer space at the cloud also scales with the number of VMs, with a ratio 5:1, i.e., $k_C = 5\,n_C$. The rest of parameters are as described in Table 1.

The figures show that (i) edge routing probabilities obtained with centralized and distributed approaches are not distant (cf. Figures 6b and 7b), and (ii) the corresponding failure probabilities are even closer (cf. Figures 6a and 7a), with PoA values of the order of a few percent (cf. Figures 6c and 7c). In particular, Figures 6a and 7a show that when the cloud contains fewer servers, the performance of both GANDALPH and RANDALPH deteriorates, and both the coordinated and distributed approaches achieve the same level of quality. They also dictate a more frequent selection of the edge server, save for GANDALPH under fixed aggregate traffic, where a plateau is reached (see Fig-



(a) Failure probability

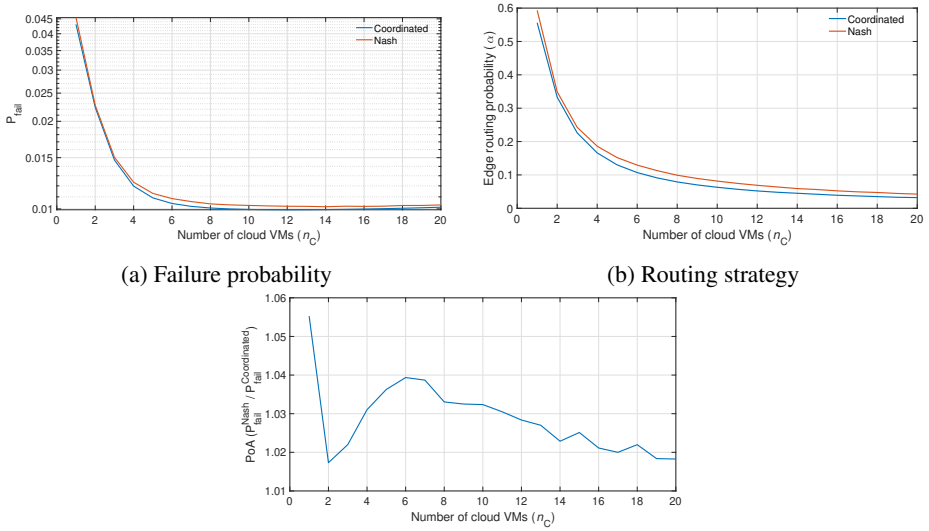
(b) Routing strategy

(c) Price of anarchy

Figure 6: Performance of optimized RANDALPH (Coordinated) and GANDALPH (Nash) with variable cloud capacity (i.e., variable n_C, the number of virtual machines (VMs)), with fixed aggregate traffic of 1500 req/s generated by $n_u = 60$ UEs at $\lambda_u = 250$ req/s. The buffer space of the cloud is $k_C = 5\,n_C$. Other parameters are as in Table 1. Network capacity is less than offered traffic, hence $\rho > 1$, as long as the number of VMs is below 7.

ure 6b). However, the resulting differences in terms of failure probability (Figures 6a and 7a) are minimal.

With fixed request arrival rate ($n_u\,\lambda_u = 1500$ req/s) and variable cloud capacity (cf. Figure 6), the load ρ is below 1 with 7 or more VMs, because each VM contributes with a capacity of 200 services per second (and the edge has one VM). This explains the different behaviors of the curves before and after the point at 7 VMs. The behavior of the algorithms with only one VM at the cloud is interesting. In that case, the capacities of edge and cloud are the same, and the network is largely overloaded (1500 req/s with a total capacity of 400 services per second). Edge and cloud servers are in this case almost equivalent, since whenever a service request reaches a server, it has a high chance of being lost due to a full buffer. However, if the request is accepted, it is very likely to be the last in the buffer and, thus, have in front 9 services (one in progress and 8 in the buffer) at the edge, and 4 services (one in progress and 3 in the buffer) at the cloud, since at $n_C = 1$ we have just one VM. On average this implies 45 ms waiting delay at the edge and 20 at the cloud, plus 5 ms for the request service, plus the RTT from/to the UE. This means 76.955 ms at the edge server ($24.875 + 0.08 + 2 + 45 + 5$) and 75.498 ms at the cloud server ($24.875 + 0.08 + 24.543 + 1 + 20 + 5$). Hence, the cloud server is slightly better, and we see in Figure 6b that its choice probability is slightly higher than 0.5 with the selfish approach. Instead, the best coordinated and globally optimal strategy consists in sending only a small part of the traffic to the edge,



(a) Failure probability

(b) Routing strategy

(c) Price of anarchy

Figure 7: Performance of optimized RANDALPH (Coordinated) and GANDALPH (Nash) with variable cloud capacity (i.e., variable n_{C}, the number of virtual machines (VMs)), with fixed system load $\rho = 0.682$ generated by $n_u = 60$ UEs and variable λ_u. The buffer space of the cloud is $k_{\mathrm{C}} = 5\, n_{\mathrm{C}}$. Other parameters are as in Table 1.

and let the rest be lost. By pursuing the more appealing solution, GANDALPH incurs higher losses, a few percent more than RANDALPH, although this effect is not well visible in Figure 6a due to the adopted log scale.

Conversely, with fixed system load and variable offered traffic (cf. Figure 7), the differences between GANDALPH and RANDALPH are less important. In this case, at the NE point, the edge routing probability is always a bit higher than at the best coordinated operation point found with RANDALPH. This happens because, being $\rho < 1$ and the edge not saturated, any selfish UE sees an incentive in offloading some traffic from cloud to edge, thus sparing some RTT and, in turn, reducing the timeout probability.

6. Fictitious Play Implementation

In this section we introduce a fictitious play (FP) that represents a distributed and iterative implementation of GANDALPH. We used FP to evaluate the realistic case in which players do not have complete information on the systems and the other players. We consider two FP versions. The first consists in a Matlab implementation in which we can control network and server features in full. The second is a live implementation of FP over Docker containers connected through the Internet. In the following, we describe FP, give details on the experimental setup, and compare results obtained with FP with the ones obtained with our model.

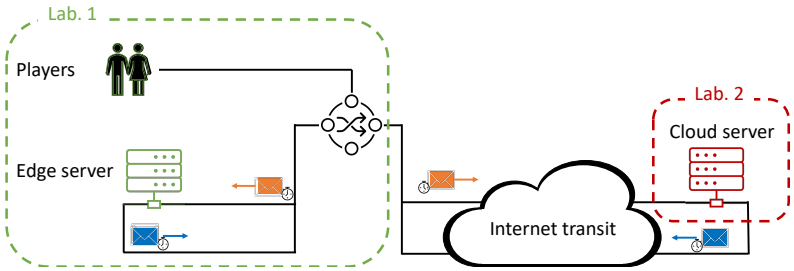


Figure 8: FP experimental setup

6.1. Fictitious play and Nash equilibria

Also known as best response dynamics (BRD), FP is a game theoretic procedure where iterative convergence to a NE is attempted in a distributed fashion [37]. In static games of complete information, agents have full awareness of the game and therefore can be assumed to immediately reach a NE as the outcome of the game. In setups with incomplete or imperfect information, this can be approached through subsequent stepwise convergence. In other words, agents take actions that maximize their expected utilities in the current setup, while at the same time forming beliefs about the future actions of others, also based on local information. While the two terms are equivalent and often used interchangeably, the former action-taking part is more properly referred to as BRD, whereas "fictitious play" usually includes the learning and dissemination aspects as well.

In general, not every NE of a game can be reached through FP, since asymptotic convergence is guaranteed only for correlated equilibria, i.e., working points that are also attractors [38]. However, specific equivalence holds in principle for specific classes of problems such as congestion games (and more in general potential games), to which the problem considered here belongs, and where best response dynamics is actually an efficient way to find the NEs. FP is especially practical in communication networks, for example, in routing or information spreading problems, where distributed procedures are common [39]. In a sense, the Bellman-Ford algorithm can be framed as a BRD, converging to the minimum-cost routing. Also in a similar fashion, FP can be regarded as a distributed learning procedure, for example through reinforcement learning [40].

For the particular problem investigated here, we modified the setup to make it dynamic (as opposed to a one-shot choice of the parameter α) and allowing players to alternate their actions in rounds and wait to change their action profile until their next turn. This way, players do not have complete information about the system and can only observe the effects of the actions on their perceived $\mathsf{P}_{\mathrm{fail}}$. The available actions are to increase, decrease, or keep α_i constant. Players choose their actions according to the $\mathsf{P}_{\mathrm{fail}}$ observed at the end of the previous game rounds. If the previous action decreased $\mathsf{P}_{\mathrm{fail}}$, the player chooses the same action for the upcoming round, otherwise the player changes action. Players do not change their previous action for small oscillation of $\mathsf{P}_{\mathrm{fail}}$, smaller than a threshold Δ.

Table 2: Parameters used in the experimental evaluation

Parameter	Notation	Value
Number of UEs	n_u	2
Service request rate per UE	λ_u	$100 \sim 450\ \mathrm{s}^{-1}$
Number of virtual machines at edge	n_{E}	1
Request capacity at edge	k_{E}	5 requests
Number of virtual machines at cloud	n_{C}	3
Request capacity at cloud	k_{C}	15 requests
Average request service time	μ^{-1}	5 ms
RTT (UE—edge)		2 ms
RTT (UE—cloud)		40 ms
Service timeout	T_O	75 ms
Min step size	$\sigma_{\min}$	0.01
Max step size	$\sigma_{\max}$	0.1
Multiplicative step factor	ψ	1.1
$\mathsf{P}_{\mathrm{fail}}$ threshold	Δ	1% of $\mathsf{P}_{\mathrm{fail}}$ at the previous game stage

6.2. Experimental setup

As shown in Figure 8, the experimental setting is based on a distributed measurement infrastructure spanning two laboratories across Europe, one in Italy and one in Spain. Labs are connected through the public Internet. Cloud and edge servers are deployed in the two laboratories, and each server instance is configurable in the number of available virtual machines, waiting room size, and average service time. There are two players, located in only one of the two labs, so that the server which resides in the same lab plays the role of the edge server, while the other server plays the role of the cloud server. We only implement two players to be able to show the game dynamics in a simple yet effective way.

Servers are implemented using Golang and communicate with players through QUIC; for ease of deployment, servers are compounded within a Docker image that we have publicly released on GitHub.[5] Players are highly parameterized and follow the same design pattern of the servers—hence, they are implemented in Golang as well, and are deployed through Docker images. Players generate a traffic flow whose intensity is parameterized, and the routing of outgoing traffic is probabilistic, according to the selected α.[6] At the beginning of an experiment, players have no record of previous actions, and the prescribed action for the two initial stages is to increase α by a fixed amount. Subsequent decisions are taken according to the effects on $\mathsf{P}_{\mathrm{fail}}$ measured during the previous game stages: every time a player needs to pick an action among those available—namely, stay, increase, or decrease—she compares the past two $\mathsf{P}_{\mathrm{fail}}$ and chooses an action to improve the observed $\mathsf{P}_{\mathrm{fail}}$.

[5] `https://github.com/paolocastagno/gandalf.git`

[6] In the FP, each player has her own α_i at each stage of the interactive game, however, here and in what follows, we omit the index i.

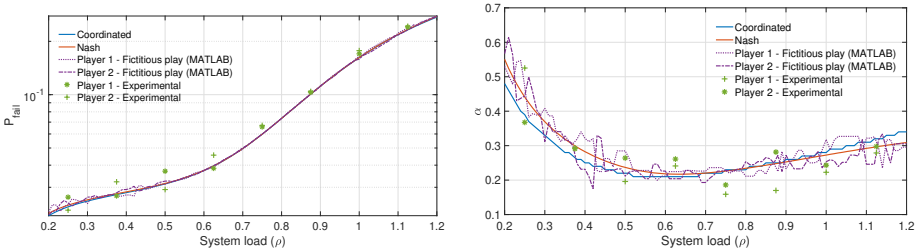
The observed average round trip time between players and edge server, all within the same lab, is 2 ms, while the cloud server is located 39 ms away from the players, in the other lab. Both servers use virtual machines: one at the edge ($n_E = 1$) and three at the cloud ($n_C = 3$). All virtual machines implement a deterministic service time equal to 5 ms, i.e., each of them can serve 200 req/sec. The edge server implements a buffer hosting four requests, hence $k_E = 5$, considering the single virtual machine used. The cloud has three times the buffer space of the edge server, i.e., $k_C = 15$, twelve slots in the waiting room and three virtual machines. Service requests flowing through the players have exponentially distributed inter-arrival times, and by varying the average inter-arrival time it is possible to control the system offered load ρ. Thus, the edge server is an $M/D/1/5$ queue, while the cloud server is an $M/D/3/15$ queue. Table 2 summarizes all the relevant parameters of the experimental setup.

6.3. Comparative performance evaluation of FP, NE and global optimization

Before detailing the experiment's results, we would like to point out similarities and differences differences between the experimental settings in the FP implemented over the Internet, the FP implemented over Matlab and the analytical models for NE and centralized-global optimum:

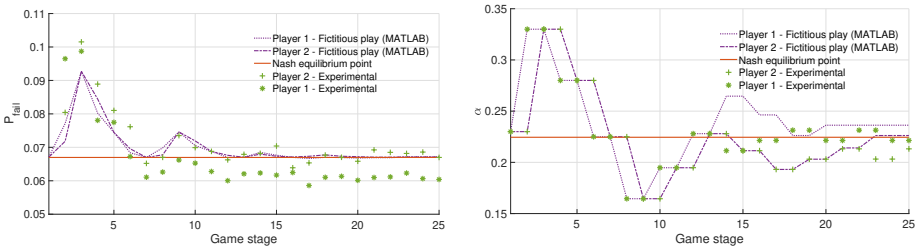
1. Since we cannot control latency in the experimental setup, we measure the round trip times of interest in there, and use them for implementing FP over Matlab and for computing the model.
2. Servers are fully dedicated to players' requests in all cases, although the experimental implementation uses Docker containers over lab workstations, hence they might see some noise due to the host operating system.
3. Similarly, network conditions are not necessarily stable in the experiments over the Internet, in which some uncontrolled background traffic was present.
4. FP in experiments and in Matlab always uses an iterative algorithm in which players alternate their moves, while in the model the users can directly calculate the NE and jump to it.
5. In the experimental FP implementation, the service time is deterministic rather than exponentially distributed like in FP over Matlab and in the model.
6. Since we implement FP with a tolerance Δ on $\mathsf{P}_{\mathrm{fail}}$, the two players might converge to a strategy in which their failure probabilities differ slightly. Therefore, for FP, the PoA is computed as the ratio between the *average* $\mathsf{P}_{\mathrm{fail}}$ observed by the two players and the optimal solution.
7. In experimental settings, testing players' actions requires to dwell in the current setting until a reliable evaluation of $\mathsf{P}_{\mathrm{fail}}$ is made. To speed-up convergence of FP, we implemented an adaptive step that is increased by a factor ψ every time the selected action reduces $\mathsf{P}_{\mathrm{fail}}$ or halved otherwise. Also, to control the oscillation arising from the interleaving choices of the players, the step length is bounded within $[\sigma_{\min}, \sigma_{\max}]$.

The FP implementation has been extensively evaluated, and Figure 9 reports the actions taken by the players at FP convergence and the corresponding $\mathsf{P}_{\mathrm{fail}}$ observed for different traffic loads in both the experimental and the MATLAB settings, compared



(a) Players' observed $\mathsf{P}_{\mathrm{fail}}$ (b) Routing strategy

Figure 9: Players' action profile for GANDALPH (Nash), for the analytical and experimental implementations of FP with the parameter values in Table 2 and variable system load (obtained by varying λ_u only).



(a) Evolution of failure probability (b) Evolution of edge routing strategy

Figure 10: Action profile for GANDALPH (Nash) with two players when implementing FP (solid and dashed are MATLAB and experimental measures respectively) with fixed system load $\rho = 0.75$ generated by $n_u = 2$ UEs. The buffer space of the cloud is $k_{\mathrm{C}} = 15\, n_{\mathrm{C}} = 3$ and in the edge is $k_{\mathrm{E}} = 5\, n_{\mathrm{E}} = 1$. Other parameters are as in Table 2.

to both the NE and the centralized global optimal solution. The iterative approach of FP shows some variability around the expected routing choices; however, such oscillations barely impact the observed failure probabilities. The MATLAB implementation of FP shows some differences in the failure probability for low traffic loads, but this is due to the relatively flat shape of the $\mathsf{P}_{\mathrm{fail}}$ in the surrounding of the NE. The iterative approach easily reaches the plateau, but then struggles to reach the exact equilibrium point due to the minor variations in the values of failure probability in this area. Indeed, small variations of the failure probability–smaller than the threshold Δ–do not cause the algorithm to change its previous decision. On the other hand, the variability of the network conditions and the presence of unrelated background traffic cause the experimental measure to slightly differ from the analytical solution. Also, small differences in the observed $\mathsf{P}_{\mathrm{fail}}$ depend on the reduced variability induced by the deterministic service time used in the experimental measures to model services at the edge and cloud servers, whereas the analytical implementations (i.e., FP over MATLAB, Nash, and centralized global solutions) employ exponentially distributed service times.

Figure 10 shows the evolution of the iterative game for an offered load $\rho = 0.75$. Specifically, Figure 10a reports the observed $\mathsf{P}_{\mathrm{fail}}$, while Figure 10b shows the evolution of the routing actions taken by the two players, i.e., the values of α controlling

the routing of the service requests, which produce the failure probability of the previous figure. In this setup, the initial routing configuration is close to the NE, but due to the algorithm design the early stages take players away from it. However, as soon as players start to evaluate the effects of past choices, they invert the trend and start approaching values close to the NE. Remarkably, already at the twelfth game stage, the observed $\mathsf{P}_{\mathrm{fail}}$ values converge to the desired equilibrium.

Notice that stochastic and dynamic system behaviors exist in reality and in our specific implementation, while our model accounts for a mean-field analysis of performance. Nonetheless, the results in this section have shown that our simple model produces accurate predictions of the system performance.

7. Conclusions

We presented a game theoretic analysis of a randomized policy for edge/cloud server selection to satisfy latency-constrained computing-based service requests.

Our objective was to quantify the efficiency of a distributed-selfish implementation of the policy as compared to a centralized-global optimization of the server selection probabilities. This is useful for system design purposes, as it offers the tools to predict the system behavior of alternative policies in a temporal snapshot, under the worst case operational conditions, with ρ representing the traffic load in the peak hour.

Our results are extremely encouraging, since they show that a selfish allocation by strategic agents, through an algorithm called `GANDALPH`, behaves very similar to the centralized-global optimal policy `RANDALPH`, with values of α that are close, at least when the system is not overloaded, and resulting performance metrics that are even closer, in all cases, for the two approaches.

Furthermore, our experimental validation showed that a distributed allocation converges to the Nash equilibrium in practical contexts, even when using the iterative approach typical of selfish players following best response dynamics (with a "fictitious play").

Although the presented performance analysis is valid for homogeneous settings, and a wider performance evaluation of more heterogeneous servers, users and scenarios in general is left for future work, our results bear significant practical relevance. They prove that a simple distributed server selection policy based on a distributed approach can provide performance quite close to the centralized globally optimum approach. Indeed, while our previous work in [7] proved that algorithms based on global system parameters can be more effective than stateful algorithms due to the inherent staleness of state information, in this paper we go one step further, proving that distributed selfish stateless algorithms perform almost as well as their centralized global optimization counterparts. This provides solid ground for implementations based on simple distributed approaches, which do not require continuous parameter monitoring and reporting or a central controller.

Acknowledgments

This work has been supported by the Project AEON-CPS (TSI-063000-2021-38), funded by the Ministry of Economic Affairs and Digital Transformation and the Eu-

ropean Union NextGeneration-EU in the framework of the Spanish Recovery, Transformation and Resilience Plan and by the RESTART Program, financed by the Italian government with the resources of the Italian Recovery, Transformation and Resilience Plan – Mission 4, Component 2, Investment 1.3, theme 14 "Telecommunications of the future" (PE00000001 - program "RESTART", projects R4R and ITA NTN).

References

[1] V. Mancuso, L. Badia, P. Castagno, M. Sereno, M. Ajmone Marsan, Efficiency of distributed selection of edge or cloud servers under latency constraints, in: Proc. IEEE Mediterranean Communication and Computer Networking Conference (MedComNet), 2023, pp. 158–166.

[2] J. Pan, J. McElhannon, Future edge cloud and edge computing for Internet of things applications, IEEE Internet Things J. 5 (1) (2017) 439–449.

[3] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, P. Mohapatra, Edge cloud offloading algorithms: Issues, methods, and perspectives, ACM Comp. Surv. 52 (1) (2019) 1–23.

[4] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, IEEE Internet Things J. 5 (4) (2017) 2633–2645.

[5] L. Liu, Z. Chang, X. Guo, S. Mao, T. Ristaniemi, Multiobjective optimization for computation offloading in fog computing, IEEE Internet Things J. 5 (1) (2017) 283–294.

[6] R. Gouareb, V. Friderikos, A.-H. Aghvami, Virtual network functions routing and placement for edge cloud latency minimization, IEEE J. Sel. Areas Commun. 36 (10) (2018) 2346–2357.

[7] V. Mancuso, P. Castagno, M. Sereno, M. Ajmone Marsan, Stateful versus stateless selection of edge or cloud servers under latency constraints, in: Proc. IEEE WoWMoM, 2022, pp. 110–119.

[8] G. Panek, I. Fajjari, H. Tarasiuk, A. Bousselmi, T. Toukabri, Application relocation in an edge-enabled 5G system: Use cases, architecture, and challenges, IEEE Commun. Mag. 60 (8) (2022) 28–34.

[9] G. Quer, F. Librino, L. Canzian, L. Badia, M. Zorzi, Inter-network cooperation exploiting game theory and Bayesian networks, IEEE Trans. Commun. 61 (10) (2013) 4310–4321.

[10] J. Moura, D. Hutchison, Game theory for multi-access edge computing: Survey, use cases, and future trends, IEEE Commun. Surveys Tuts. 21 (1) (2018) 260–288.

[11] L. Prospero, R. Costa, L. Badia, Resource sharing in the Internet of Things and selfish behaviors of the agents, IEEE Trans. Circuits Syst. II 68 (12) (2021) 3488–3492.

[12] L. Qiu, Y. R. Yang, Y. Zhang, S. Shenker, On selfish routing in Internet-like environments, in: Proc. ACM SIGCOMM, 2003, p. 151–162.

[13] W. Xu, J. Rexford, MIRO: Multi-path interdomain routing, in: Proc. ACM SIGCOMM, 2006, p. 171–182.

[14] L. Badia, M. Miozzo, M. Rossi, M. Zorzi, Routing schemes in heterogeneous wireless networks based on access advertisement and backward utilities for QoS support, IEEE Commun. Mag. 45 (2) (2007) 67–73.

[15] F. Benita, V. Bilò, B. Monnot, G. Piliouras, C. Vinci, Data-driven models of selfish routing: why price of anarchy does depend on network topology, in: Proc. WINE, Springer, 2020, pp. 252–265.

[16] R. Colini-Baldeschi, R. Cominetti, P. Mertikopoulos, M. Scarsini, When is selfish routing bad? The price of anarchy in light and heavy traffic, Op. Res. 68 (2) (2020) 411–434.

[17] C. H. Bell, S. Stidham, Individual versus social optimization in the allocation of customers to alternative servers, Manag. Sc. 29 (1983) 831–839.

[18] H. Kameda, J. Li, C. Kim, Y. Zhang, Overall Optimal Load Balancing vs. Individually Optimal Load Balancing, Springer London, London, 1997, pp. 35–97.

[19] M. Haviv, T. Roughgarden, The price of anarchy in an exponential multi-server, Oper. Res. Lett. 35 (4) (2007) 421–426.

[20] E. Altman, U. Ayesta, B. Prabhu, Load balancing in processor sharing systems, Telecommun. Syst. 47 (2011) 35–48.

[21] A. Orda, R. Rom, N. Shimkin, Competitive routing in multiuser communication networks, IEEE/ACM Trans. Netw. 1 (5) (1993) 510–521.

[22] A. V. Guglielmi, M. Levorato, L. Badia, A Bayesian game theoretic approach to task offloading in edge and cloud computing, in: Proc. IEEE ICC Wkshps, 2018.

[23] T. Roughgarden, E. Tardos, How bad is selfish routing?, J. ACM 49 (2) (2002) 236–259.

[24] F. Sufyan, A. Banerjee, Computation offloading for distributed mobile edge computing network: A multiobjective approach, IEEE Access 8 (2020) 149915–149930.

[25] S. Scherrer, A. Perrig, S. Schmid, The value of information in selfish routing, in: Proc. SIROCCO, 2020, p. 366–384.

[26] Y. Zeng, Q.-C. He, X. Cai, Targeted Bayesian persuasion in a basic selfish routing game, in: Proc. INFORMS-CSS, Springer, 2022, pp. 47–56.

[27] P. N. Brown, Providing slowdown information to improve selfish routing, in: Proc. GameNets, Springer, 2023, pp. 328–338.

[28] C. H. Papadimitriou, Algorithms, games, and the Internet, in: Proc. ACM STOC, 2001, pp. 749–753.

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

[30] F. Shan, J. Luo, J. Jin, W. Wu, Offloading delay constrained transparent computing tasks with energy-efficient transmission power scheduling in wireless IoT environment, IEEE Internet Things J. 6 (3) (2018) 4411–4422.

[31] I. Cohen, P. Giaccone, C. F. Chiasserini, et al., Distributed asynchronous protocol for service provisioning in the edge-cloud continuum, in: International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2023), IEEE, 2023.

[32] D. Haja, M. Szabo, M. Szalay, A. Nagy, A. Kern, L. Toka, B. Sonkoly, How to orchestrate a distributed openstack, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE, 2018, pp. 293–298.

[33] P. Castagno, V. Mancuso, M. Sereno, M. Ajmone Marsan, A simple model of MTC flows applied to smart factories, IEEE Trans. Mobile Comput. 20 (10) (2020) 2906–2923.

[34] G. Gao, M. Xiao, J. Wu, H. Huang, S. Wang, G. Chen, Auction-based VM allocation for deadline-sensitive tasks in distributed edge cloud, IEEE Trans. Serv. Comput. 14 (6) (2019) 1702–1716.

[35] D. Callegaro, M. Levorato, Optimal edge computing for infrastructure-assisted UAV systems, IEEE Trans. Veh. Technol. 70 (2) (2021) 1782–1792.

[36] G. Premsankar, M. Di Francesco, T. Taleb, Edge computing for the Internet of Things: A case study, IEEE Internet Things J. 5 (2) (2018) 1275–1284.

[37] B. Swenson, S. Kar, On the exponential rate of convergence of fictitious play in potential games, in: Proc. IEEE Allerton Conf., 2017, pp. 275–279.

[38] G. Ostrovski, S. van Strien, Payoff performance of fictitious play, J. Dyn. Games 1 (4) (2014) 621–638.

[39] E. Altman, Bio-inspired paradigms in network engineering games, J. Dyn. Games 1 (1) (2014) 1–15.

[40] F. Chiariotti, L. Badia, Strategic age of information aware interaction over a relay channel, IEEE Trans. Commun. (2024).