Performance of Distributed Multiparty Online Gaming Over Edge Computing Platforms [1]

D. Olliaro[1,2], V. Mancuso[3,4], P. Castagno[5], M. Sereno[5,6], M. Ajmone Marsan[3]
[1] Ca' Foscari University of Venice, Venice, Italy
[2] KTH Royal Institute of Technology, Stockholm, Sweden
[3] IMDEA Networks Institute, Leganes, Spain
[4] University of Palermo, Palermo, Italy
[5] Università di Torino, Torino, Italy
[6] Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy

Abstract

We study the performance of online games played over a platform that implements gaming as a service (GaaS) in a mobile network slice that hosts concatenated virtual network functions (VNFs) at the edge. The distributed gaming architecture is based on edge computing facilities, whose utilization must be carefully planned and managed, so as to satisfy the stringent performance requirements of game applications. The game manager must consider the latency between players and edge server VNFs, the capacity and load of edge servers, and the latency between edge servers used by interacting players. This calls for a careful choice about the allocation of players to edge server VNFs, aiming at extremely low latency in interactions resulting from player's commands. We develop an analytical model, which we validate with experiments in the wild, and show that, under several combinations of system parameters, deploying gaming VNFs at the edge can deliver better performance with respect to cloud gaming, in spite of the complexities arising from the distribution of gaming VNFs over edge servers. Our analytical model provides a useful tool for edge gaming systems performance prediction, thus supporting the management of GaaS applications.

Keywords: Online gaming; Gaming as a Service; Edge computing; Performance model.

1. Introduction

In the early days, video games demanded expensive specialized hardware consoles with advanced graphics processing capabilities. Network connections were primarily for player interactions, limited to those not on the same gaming console. Today, the landscape has shifted dramatically. People now seek to indulge in their favorite games using their smartphones anytime, anywhere. Games have evolved into services delivered via ad hoc virtual network functions (VNFs), following the Gaming as a Service (GaaS) paradigm. Game logic and video processing occur within a VNF housed in a computing infrastructure, typically a cloud data center [1]. Players transmit their game commands to the cloud, where the VNF amalgamates them with others' commands to construct the video scene, which is then encoded and streamed back to players. RTP streams via UDP ports facilitate bidirectional communication. Additionally, the game VNF in the cloud must assess the player's connection quality to determine the appropriate video quality. Consequently, the player's device is tasked solely with collecting commands, transmitting them along with connection feedback to the VNF, and decoding and displaying the received video stream. However, only players with broadband access and low latency to game VNFs can properly experience online gaming. Hence, considering the increasing relevance of mobile gaming, and the evolutionary trend of radio access networks (RANs), we can expect that game VNFs will soon move from cloud to edge, thus increasing bandwidth and reducing latency for network customers [2]. Hence, we propose and study a GaaS framework built on VNFs residing on the network edge.

Processing player's commands with a VNF residing on one of the edge computing facilities, and generating the corresponding video stream at the edge, has obvious advantages in terms of response delay and network resource utilization. However, one of the most relevant issues is the allocation of players to VNFs, as players that connect to VNFs residing in different computing facilities could experience consistency issues. Since we cannot expect that all interacting players will connect to the same VNF, it will be necessary to transfer relevant commands from one VNF to the other. VNFs serving different players will therefore be chained to control the game. However, multimedia rendering and streaming will remain bounded to each player's serving VNF and will not need to be propagated to other VNFs, thus saving network bandwidth and making the most of edge connectivity.

We look at the challenges arising when chaining VNFs to build a distributed online gaming architecture, considering a slice of a cellular network, where VNFs are hosted. When players join from distant locations, the slice management, and operation (MANO) VNF must map players onto VNFs available in the edge network it manages, considering latency between VNFs and players, load, and processing capacity of VNFs, as well as the latency between VNFs that serve interacting players.

Our primary contribution is the development of *the first stochastic model* for the evaluation of edge gaming *success probability*, i.e., the probability that player's commands be in-

corporated in the rendered game within a deadline compatible with the players' reaction time. Additional contributions are: *i)* the comparison of predictions obtained with our stochastic model with results of experiments that validate the simplifying model assumptions; *ii)* the assessment of the impact of the main system parameters on edge gaming success probability, using data derived from measurements of real gaming platforms; *iii)* a comparison of success probabilities achievable with edge or cloud gaming approaches; *iv)* various approaches for optimizing the incremental allocation of incoming groups of concurrent gamers to edge servers, enhancing performance.

2. Related work

The concept of cloud gaming emerged around 2009 and brought to the online gaming domain the streaming technologies that Spotify and Netflix introduced in music and video domains about a decade earlier. However, cloud gaming designers soon realized that the quality of experience (QoE) requirements for online gaming are quite different from those for music and video.

This led the ITU (International Telecommunications Union) Study Group 12, focusing on *Performance, QoS and QoE* [3] to work on the identification of factors that influence online gaming QoE and on approaches to predict, plan, and manage QoE [4].

Building on this initial activity, ITU SG 12 produced three recommendations: G.1032, P.809, and G.1072 [5, 6, 7]. Recommendation G.1032 identifies factors that influence cloud gaming QoE, while Recommendation P.809 identifies approaches for subjective evaluation of gaming QoE. Finally, Recommendation G.1072 describes QoE models for cloud gaming, accounting for the effect of underlying networks.

Our work goes in the direction of Recommendation G.1072, except we develop a probabilistic approach for the estimation of the gaming QoS (which is not present in G.1072).

In [8] the authors define a methodology for the assessment of the user-perceived quality of cloud gaming systems and conduct extensive experiments. Other recent papers report measurement studies of cloud gaming applications [9, 10, 11, 12], and the very recent paper [13] looks at the specific case of gamers connected through a cellular network. These works identify the main network characteristics of cloud gaming and look at achievable QoS metrics, from which they infer the most relevant drivers of user's QoE. Measurements reported in these studies determined the choice of parameters we use later in this paper for the performance analysis of edge gaming.

It is interesting to note that all the above-mentioned works consider *cloud* gaming systems, i.e., systems where game control and graphics engines that generate video streams for gamers reside in the cloud. Instead, Kim *et al.* [14] propose to separate game control, which remains in the cloud, from rendering, that is brought to the edge, hence closer to the end user, with the advantage of reduced latency in video delivery, and lower impact on network resources, especially in the case of ultra high definition video. Extending our conference contribution [15] with the analysis and performance evaluation for groups of simultaneous players and with their optimal allocation to available edge servers, we bring this concept one step forward, allocating both game control and rendering engines on the edge, with additional benefits in terms of latency and resource consumption, at the cost resulting from coordinating edge computing facilities used by interacting gamers.

In [16], gaming is mentioned as a relevant use case for the IETF Reliable and Available Wireless (RAW) architecture. Moreover, in [17] the authors propose an emulation framework to investigate QoE in a real-time video streaming scenario.

Related studies explore architectural propositions for online games, relevant to our work. For instance, [18] proposes a distributed engine for online gaming. The paper [19] discusses edge computing's role in addressing latency and network resource efficiency in online gaming. Unlike our work, these proposals lack analytical insights for game service providers.

Some studies propose architectures for online gaming that prioritize the economical utilization of resources, encompassing bandwidth, computational resources, and energy. For instance, the authors of [20] leverage software migrations to optimize energy consumption, and [21, 22, 23, 24] propose efficient resource allocation for massive multiplayer online gaming with latency constraints. Our work is complementary to resource allocation, because we analyze the compound importance of server characteristics, given the amount of allocated resources.

Demanding QoS prerequisites of massive multiplayer online gaming with strict latency constraints call for optimal and adaptable resource management of both computational and network resources. This particular challenge can be addressed through VNF optimization techniques similar to those proposed, for instance, in [25, 26]. For the specific case of online gaming, we offer a tool that permits to identify VNF requirements and VNF chaining to achieve high QoS.

3. System model and game requirements

In the following, we interchangeably use the terms "game VNF" and "edge server" to indicate a VNF dedicated to online gaming. We as well as use terms "gamer" and 'player" interchangeably.

We consider a mobile network that supports a multiparty distributed gaming application with strict latency requirements. Gaming engines are located on the edge, spread across multiple game VNFs, with the goal of minimizing the delay experienced by end users.

We cast the required gaming architecture onto the standard 3GPP (3rd Generation Partnership Project) architecture for communication and computing.

3.1. Connect-compute architecture

In a 3GPP network there are four main network functions that we need to consider [27]:

- *User Equipment* (UE) is the equipment on top of which the gamer plays. We consider wireless-connected gamers.

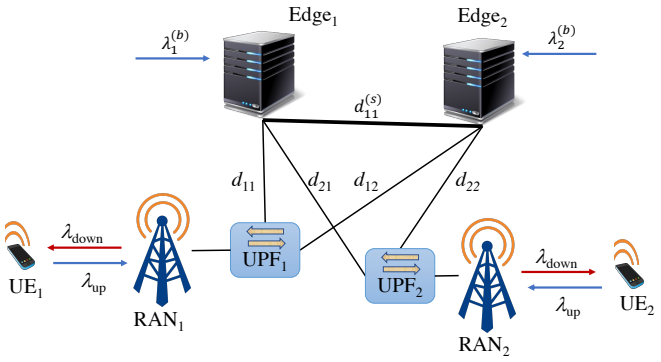


Figure 1: Reference scenario

- *(Radio) Access Network*, or (R)AN is the (wireless) infrastructure that connects user's devices to the core network. It includes several facilities enabling communication. We consider wireless access networks.

- *User Plane Function* (UPF) is a vital element in 3GPP (5G) networks, as it is responsible for managing data traffic and handles tasks such as packet forwarding, traffic routing, and QoS enforcement.

- *Edge* is a computing facility under the control of the cellular network, that brings computation resources closer to gamers, improving response times. Edges are deployed as VNFs accessible through the user plane (via UPFs). Their proximity enhances real-time interaction.

As shown in Fig. 1, the interaction between the mentioned components determines a chaining of VNFs. RAN establishes wireless links for reliable connectivity, while UPFs manage data transmission and routing. Different edge computing facilities can be reached from different UPFs with different delays. Edge facilities could be implemented with today's MEC VNF [28] or with their future extensions and replacements [29], which will act on the user plane and run on virtual machines or containers that can be located at base stations as well as in core and transport hardware of telecommunication operators or in data centers. Access to Edge/MEC VNFs must be supported by the presence of proxy functionalities in the user plane, which are typically implemented in the UPF. In practice, the UPF can be responsible of intercepting game access requests and deciding which Edge/MEC VNF shall be contacted to enter a game platform. As an alternative, game access decisions could be left, at least partially, to the end user. This can be done by including the GaaS platform supported by the network operator in the catalogue exposed by the NEF (Network Exposure Function) VNF in the control plane of a 3GPP network [30]. With NEF, the user will be able to fetch information about the presence of gaming servers and their characteristis. Instead, Edge/MEC VNF interconnection needed to synchronize multiple users accessing different servers must be done at application layer by the edge servers. However, this can be achieved over a plain 3GPP architecture, with servers acting as users connected to UPFs. For simplicity, we omit this passage in the architecture shown in Fig. 1, but one can easily figure out that the link connecting Edge$_1$ and Edge$_2$ in the figure crosses one or more UPFs. Going through UFPs is necessary, and the overlay public IP infrastructure cannot be used, because VNFs typically reside in private networks and use private IP addresses. As an alternative, it could be possible to establish plain IPSec tunnels in between Edge/MEC VNFs, which is equivalent to pass through UPFs but would not guarantee that those application tunnels, not controlled by the network operator, would eventually fully lay on the operator's network, and hence would be longer and slower with high probability.

While in this paper we consider a wireless access of players to gaming VNFs residing in edge computing facilities, it must be noted that our approach can be easily modified to accommodate the case of wired access.

3.2. Application requirements

Latency requirements depend on the level of interaction between gamers. If two gamers are closely interacting (e.g., fighting against the same monster), it is necessary that the latency with which they observe each other's actions (and those of the monster) is as small and as equal as possible.

The most important QoS KPIs (key performance indicators) for online gaming are upstream and downstream data rates available between game server and user, upstream and downstream packet loss probabilities, and round trip latency. According to [9, 10, 11], typical requirements for immersive games are of the order of 1 Mb/s upstream (a small part for gamer's commands, the rest for feedback on network connection KPIs) and between 5 and 50 Mb/s downstream (for game video stream; of course, the downstream data rate depends on the gamer's screen size, and heavily impacts the quality of the video received by the gamer, hence the immersivity of the game). Packet loss probabilities are tolerated up to 10-30% in upstream (the KPI feedback is very redundant) and to a few percent in downstream. Latency requirements are of the order of 100 ms or less, which refer to the whole *game response delay* that includes all latency components due to network, processing, and buffering [8, 14].

4. Analysis and problem formulation

4.1. Reference scenario

The reference scenario for this work is depicted in Fig. 1, and is modeled as shown in Fig. 2. We consider a network with two or more base stations (BSs) where the (R)AN network function is accessed by UEs as a necessary step to access UPFs and two or more game VNFs. To simplify the description of the considered scenario, and given the geographically distributed nature of the service, jointly with the extreme RAN densification process envisioned by 5G and beyond architectures, we assume that within the gaming slice, each BS hosts a single gamer and each UPF serves a single BS. Therefore, we use a single index to refer to a user/BS/UPF chain. This assumption can be easily removed—and was actually removed in the implementation of the model that will be described later in this section.

We assume that communicating between BS i and Edge j incurs three delay components: (i) a fixed delay d_{ij}; (ii) a random

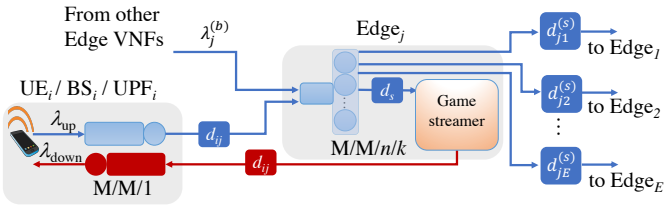


Figure 2: Network model (detail for a generic gamer)

component that accounts for user mobility, signal fluctuations, noise, scheduling, as well as other factors, including retransmissions due to losses; (iii) a variable queueing and processing time, which is modeled by means of an M/M/1 queue in uplink (from BS to Edge) and an M/M/1 queue in downlink (from Edge to BS). The sojourn time in the uplink (respectively downlink) queue associated with BS i and Edge j is a random variable denoted as U_{ij} (respectively D_{ij}).

An online gamer produces a flow of λ_{up} packets per second in uplink, and receives a multimedia stream of λ_{down} packets per second in downlink. For simplicity, we assume that the intensity of streams does not change over time. The stationary assumption is used because it allows to design and plan the service by considering periods of peak utilization.

All uplink packets of a user are delivered by the BS to a game VNF, where they are processed by the edge server to which the user is associated. We model this uplink flow processing with a queue. We will primarily use a M/M/n/k model, $k \geq n$ (and we will present our modeling approach in this case), but we will also compare results against other models (and we will validate our assumptions with experiments). Hence, the game VNF is represented as an ensemble of n processors with a waiting room of $k-n$ packets. These parameters represent how resources are allocated within the network slice dedicated to online gaming. The time T_j spent by an uplink packet in Edge j is a random variable.

When a game VNF receives any instruction from a gamer that requires implementing changes in the downlink stream (e.g., a change of view, a movement that has to be reflected in the multimedia stream, etc.), the change is implemented with a fixed delay d_s, which represents the time needed to start producing the modified stream within the VNF.

4.2. Distributed gaming server model

Since gamers that are closely interacting are not necessarily close in the network, they may interface with different game VNFs and even different physical edge facilities. Thus, when two or more gamers interact, all streams they receive have to be adapted synchronously, with a tolerance of few tens of milliseconds. Hence, commands/moves issued by a gamer must be notified not only to the game VNF that manages the stream of that user but also to all other involved game VNFs. We remark that only commands and not any rendering/multimedia streams belonging to a player will need to reach VNFs serving other players.

We assume that gamers are not aware of the location of all game VNFs to be notified, while the serving game VNF of a gamer is. Therefore, when a command issued by a gamer is served by her Edge j, it is also forwarded to any Edge j' of any other gamer involved in the game scene, with a latency $d_{jj'}^{(s)}$ between serving game VNFs, which we model as a constant. This simplification is possible since edge servers are connected via high-speed networks (typically optical rings).

Guaranteeing a good level of interaction between gamers implies that the latency with which actions of one gamer are visible to other gamers remain below a fixed threshold T_O (for example 100 ms) with sufficiently high probability (e.g., 0.8) and acceptable packet loss rate (a few percent) [8], [10].

4.3. Latency

In the simple case of two users, in the worst case, the close interaction of gamers involves two game VNFs and two BSs (each of which has a separate uplink and downlink processor) in addition to the VNFs located on gamers' terminals. In general, the worst case number of involved entities scales linearly with the number of interacting gamers. The resulting latency depends on communication and computing delays, and there are two groups of delays to evaluate:

- First, the *self-latency*, i.e., the delay with which a gamer i connected to Edge j sees her own actions reflected in her game's multimedia streaming, i.e.,

$$L_{ii}^{(j)} = d_{ij} + U_{ij} + T_j + d_s + D_{ij} + d_{ij}, \qquad (1)$$

  where the fixed latency between gamer i and Edge j is reasonably taken as symmetric in uplink and downlink.

- Second, the *cross-latency*, i.e., the latency between the time a gamer i issues a command and any other gamer i' interacting with i and connected to Edge j' sees the effect of that command on her multimedia stream, i.e.,

$$L_{ii'}^{(j')} = d_{ij} + U_{ij} + \tau + d_{jj'}^{(s)} + T_{j'} + d_s + D_{i'j'} + d_{i'j'}. \qquad (2)$$

  with τ being the random component of the RAN latency.

Let us consider for example the simple case of two gamers (indicated as g_1 and g_2) connected to two game VNFs (indicated as e_1 and e_2, respectively). We need to account for four latency values: the round trip time between gamer g_1 and Edge e_1, the latency of the path $g_1 \to e_1 \to e_2 \to g_2$, the latency of the symmetric path, $g_2 \to e_2 \to e_1 \to g_1$, and the round trip time between gamer g_2 and game VNF e_2.

Assuming that it is possible to model with simple queues the delay introduced by individual system components (communication between gamers and game VNFs, and processing at game VNFs), and assuming independence among latency components, it is possible to compute probability distributions for relevant latency variables. In fact, it is possible to compute convolutions of distributions of additive delays or, equivalently, obtain the LST (Laplace–Stieltjes transform) of the delay pdf (probability distribution function) as the product of LSTs of pdfs of individual delay components.

4.4. Success probability

In addition to latency, if some queues have a finite capacity, we have to consider network losses. Hence, the success of multiparty communication depends on a combination of latency and loss. The success rate is the fraction of game control messages that are not lost and are processed on time to cause an observable change in the game multimedia streaming delivered to gamers in the same game scene, within a maximum latency T_O from message generation.

In our model, queues that can incur loss are the ones representing game VNFs, and the associated loss probability is the probability that the buffer is full, which can be computed with standard queueing theory results.

Thus, once we know the loss probabilities at queues and the pdf of the latency, it is possible to compute the probability that the game interaction be completed within T_O for all gamers in the same game scene.

For the specific case of M/M/1, M/M/n/k and M/D/1-PS (M/D/1 with processor sharing) queues used in this paper, the following results hold:

- The LST of the sojourn time (wait plus service) in an M/M/1 queue with arrival rate λ and service rate μ is

$$\widehat{f}_{\mathrm{M/M/1}}(s) = \frac{\mu-\lambda}{s+\mu-\lambda}. \tag{3}$$

- The LST of the sojourn time in an M/M/n/k queue with arrival rate λ and individual server service rate μ is

$$\widehat{f}_{\mathrm{M/M}/n/k}(s)=\frac{1}{1-p(k)}\,\frac{\mu}{s+\mu}\sum_{j=0}^{k-1} p(j)\left(\frac{n\,\mu}{s+n\,\mu}\right)^{[j-n+1]^+}, \tag{4}$$

  with

$$p(j) = p(0)\,\frac{(\lambda/\mu)^j}{\min(j,n)!\;n^{[j-n]^+}}, \quad j=0,\cdots,k, \tag{5}$$

  where $p(0)$ is found by solving $\sum_j p(j)=1$ and $[x]^+$ indicates the max between 0 and x.

- The LST of the sojourn time in an M/D/1-PS queue with arrival rate λ and service time $1/\mu$ can be computed from (5.16) in [31], i.e.:

$$\widehat{f}_{\mathrm{M/D/1-PS}}(s)=\frac{\left(1-\frac{\lambda}{\mu}\right)(\lambda+s)^2\,e^{-\frac{\lambda+s}{\mu}}}{s^2+\lambda\left(s+(\lambda+s)\left(1-\frac{\lambda}{\mu}\right)\right)e^{-\frac{\lambda+s}{\mu}}}. \tag{6}$$

The random component of the latency due to the RAN activity is modeled through a random variable with limited variability. As suggested by experimental results reported in [32, 33], the RAN latency is close to the average most of the time, although values can vary in a relatively narrow interval. We therefore use a triangular distribution between latency values t_m and t_M, whose LST is as follows:

$$\widehat{f}_{\mathrm{RAN}}(s)=\left(\frac{2}{t_M-t_m}\cdot\frac{e^{-\frac{3t_M+t_m}{4}s}-e^{-\frac{t_M+3t_m}{4}s}}{s}\right)^2. \tag{7}$$

The above expression, with appropriate parameters t_m and t_M will be incorporated into network delays U_{ij} and D_{ij}.

The additive delay due to different path components has an LST equal to the product of the LSTs of the components. Of course, each LST related to processing must be tailored to the game VNF capacity in terms of number of servers, buffer capacity, server speed, and request arrival rate. Similarly, the LST relating to communication hops must be tailored to the link data rate, message size, and load. We omit here the specific expressions of latency LSTs because their derivation is straightforward.

The loss probability relative to arrivals at the game VNF is the probability that an arrival finds all k positions busy (k includes positions in service and in the waiting line). This probability, denoted by $p(k)$, equals the steady state probability of k positions busy, due to the assumption that the arrival process is Poisson, and is expressed as:

$$\pi_{\mathrm{loss}} = p(k) = p(0)\,\frac{(\lambda/\mu)^k}{n!\;n^{j-n}}. \tag{8}$$

We use $F^{(j')}_{L_{ii'}}(t)$ to indicate the CDF (cumulative distribution function) of the cross-latency observed by game commands issued by a gamer i served by Edge j, and whose game control messages have to reach another gamer i' connected to a different Edge j'. $F^{(j)}_{L_{ii}}(t)$ indicates the self-latency. The probability of not exceeding the timeout is therefore simply computed as $F^{(j')}_{L_{ii'}}(T_O)$ and $F^{(j)}_{L_{ii}}(T_O)$, for cross- and self-latency, respectively. Notice that we assume that packet loss is negligible, including it in the RAN behavior. This assumption holds because gamer's transmissions use a modulation and coding scheme at which the bit error rate is very low, hence errors on a few bits can be recovered and the occurrence of error bursts can be obviated by means the HARQ protocol adopted at the BS. The impact of HARQ is accounted for in the model within the variable delay described by $\widehat{f}_{\mathrm{RAN}}(s)$.

Performance results depend on the intensity of flows that are offered to the modeled queues. In particular, apart from the above-mentioned λ_{up} and λ_{down}, we denote by $\lambda^{(b)}_j$ the background traffic of Edge j, generated by other gamers on the same server, playing in other game scenes. Notice that here we have neglected background traffic of communication links between gamers and game VNFs because we have assumed to use a network slice dedicated to a specific online game.

More complex and intertwined is the computation of the traffic offered to game VNFs. If we consider K groups of multiparty gamers that can use E edge servers, we indicate with $\ell^{(k)}$ the size of the gamers group $G^{(k)}$, and with $\ell^{(k)}_j$ the number of gamers of $G^{(k)}$ using server j. Hence, in group $G^{(k)}$, $\ell^{(k)}_j$ gamers are connected to server j and interact with the remaining $\ell^{(k)}-\ell^{(k)}_j$ gamers of the group, connected to other edge servers.

Therefore, the traffic to server j is as follows:

$$\lambda_j = \lambda_{\mathrm{up}}\sum_{k=1}^{K}\left(\ell^{(k)}_j+\alpha\left(\ell^{(k)}-\ell^{(k)}_j\right)\right)+\lambda^{(b)}_j, \tag{9}$$

where α is the fraction of uplink messages that convey game control instructions instead of simple streaming feedback to the game server, hence they have to be forwarded to Edge j even when the gamers are served by another edge server. In the above formula, $\lambda_j^{(b)}$ indicates background traffic at Edge j.

Eventually, the success probability of a gamer i in a multiparty game group k connected to a set of game VNFs can be expressed as the probability that all self and cross latency values originated at that gamer be below the timeout, after having discounted lost messages, which happens with probability $\pi_{\text{loss}}^{(j)}$ at each server j:

$$S_i^{(k)} = \prod_{i' \in G^{(k)}} \left(F_{L_{ii'}}^{(a(i'))}(T_O) \left(1 - \pi_{\text{loss}}^{(a(i'))}\right)\right), \qquad (10)$$

where $a(i')$ indicates the server used by i'. Notice that, in the above formula, i' takes all values in group $G^{(k)}$, including i; hence, the formula accounts for self latency in addition to cross latency.

The success of the multiparty game of group k can be seen as the success of all interacting gamers:

$$S^{(k)} = \prod_{i \in G^{(k)}} S_i^{(k)}. \qquad (11)$$

which represents the fraction of players' actions that are incorporated into the video stream and visualized on gamer screens of all players in the group within the specified timeout T_O (i.e., those that are neither lost nor late).

The overall groups success probability can then be defined in multiple ways, depending on how group performance are aggregated. What ultimately matters is the success probability of individual groups, since these determine the effective quality of experience perceived by gamers. Consequently, in our experimental evaluation that will follow, we compute the overall success probability using different aggregation strategies:

- the average group success probability, which treats all groups equally, i.e.,

$$S_{\text{avg}} = \frac{1}{K} \sum_{k=1}^{K} S^{(k)}, \qquad (12)$$

- the weighted average of group success probability, where weights account for group sizes, i.e.,

$$S_{\text{wavg}} = \frac{\sum_{k=1}^{K} |G^{(k)}| \cdot S^{(k)}}{\sum_{k=1}^{K} |G^{(k)}|}, \qquad (13)$$

  where $|G^{(k)}|$ gives the number of gamers in a given group.

- the product of group success probabilities, capturing the probability that all groups succeed simultaneously, i.e.,

$$S_{\text{prod}} = \prod_{k=1}^{K} S^{(k)}, \qquad (14)$$

- the minimum of the groups success probabilities, which reflects the worst-case performance, i.e.,

$$S_{\text{min}} = \min_{k=1,\ldots,K} S^{(k)}. \qquad (15)$$

In the following sections, we evaluate the performance of gamers using our model as well as experiments in the wild. In Section 5, we start with the case of a single test group, in the presence of background traffic that can describe the activity of other groups of gamers. Afterwards, in Section 6, we will use the model to evaluate the allocation of multiple groups, comparing ideal optimization with low-complexity heuristics.

5. Performance evaluation for a reference group of gamers

In this section we present the results of our analytical model for a given reference group of gamers, and we validate the model through experiments over the Internet.

In the case of experiments, we consider non-exponential distributions for the times between game commands generated by players, and for the VNF service times. In addition, we obviously remove independence assumptions.

The analytical results presented in the following section were derived from experiments using a set of dedicated MATLAB scripts, whose specific structure varies based on the particular experiment framework and the chosen operational parameters. To perform the necessary numerical inversion of the transforms employed within this study, we used the Concentrated Matrix Exponential (CME) method, as detailed in [34]. This choice is motivated by the fact that this method is widely known for its excellent numerical stability and the smoothness of its approximations, offering a significant advantage over other inversion techniques. Our specific implementation uses the MATLAB package made available in [35].

We mostly focus on the simplest scenario to which our work applies: a group of two gamers (g_1 and g_2) that play over a GaaS slice connected to two edge servers (e_1 and e_2). More gamers are present, but they are modeled as background traffic. This scenario is simple, yet includes all the ingredients needed to evaluate the impact of edge server selection, server load, and inter-edge communication overheads in distributed multiparty online gaming. However, we will conclude the section by showing results for larger groups of gamers with more edge options (cf. subsection 5.7).

5.1. Reference scenario

Gamers g_1 and g_2 are modeled in detail, and individual performance metrics are collected for them. Other gamers are collectively modeled through the load they impose on game VNFs, which represents background traffic for the selected pair of gamers whose performance we study. Game VNFs are modeled as M/M/n/k queues with k taking values n (hence no buffering), $2n$, $5n$ (hence the buffer space is as large as the number of servers n or four times as much), or ∞ (hence an M/M/n queue). We assume n=5 at e_1 and n=10 at e_2 for the considered GaaS slice. We also consider an M/D/1-PS queue (where

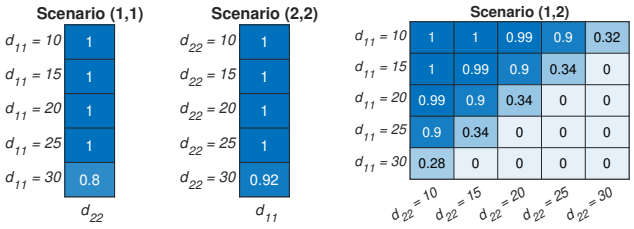


Figure 3: Success probability of a reference group, with two gamers playing together when VNFs are modeled as M/M/n queues and ρ=0.9. Delay d_{ij} in ms.

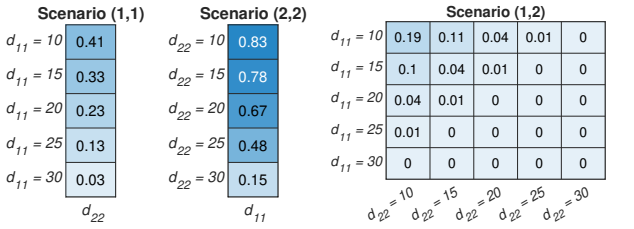


Figure 4: Success probability of a reference group, with two gamers playing together when VNFs are modeled as M/M/n queues and ρ=0.99. Delay d_{ij} in ms.

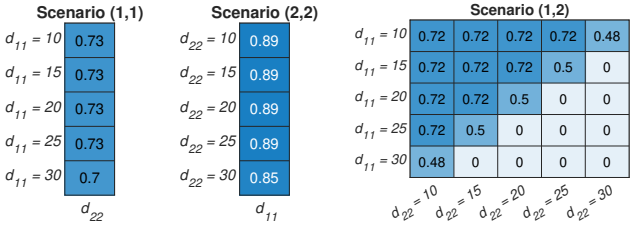


Figure 5: Success probability of a reference group, with two gamers playing together when VNFs are modeled as M/M/n/k queues and ρ=0.9. Delay d_{ij} in ms.

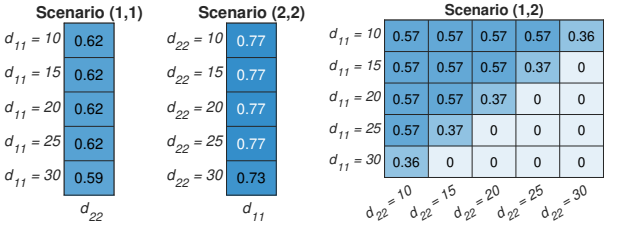


Figure 6: Success probability of a reference group, with two gamers playing together when VNFs are modeled as M/M/n/k queues and ρ=0.99. Delay d_{ij} in ms.

PS stands for Processor Sharing) at the edge servers, to approximate the analysis of realistic systems that implement round-robin execution of tasks (i.e., portions of jobs) rather than sequential processing of whole jobs.

The two gamers generate 1 Mb/s in uplink toward the game VNF to which they are connected, that is one thousand packets per second, of one thousand bits each; 10 of those packets correspond to game commands (1%), the rest is QoS feedback. Game VNFs generate 30 Mb/s in downlink toward each gamer (3 000 packets per second, each of 10 000 bits) to provide gamers with their game video stream. The time to prepare the video stream is assumed to be d_s=10 ms.

The values we use to parametrize our model were selected so as to reflect what was measured over popular online game platforms by independent studies, e.g., [9, 11]. We further assume that a single thread of a single processor on a server is able, on average, to serve a gamer. Thus, we fix μ=1 000 packets/s for the purposes of this numerical evaluation. The uplink and downlink data rates for connecting gamers to their BS through the GaaS slice are set to 10 Mb/s and 50 Mb/s, respectively.

We consider four different scenarios, that correspond to the available options for the association of the two gamers that we monitor in detail to the two edge servers. Scenario (i, j) assumes that g_1 is connected to e_i while g_2 is connected to e_j, with $i, j \in \{1, 2\}$.

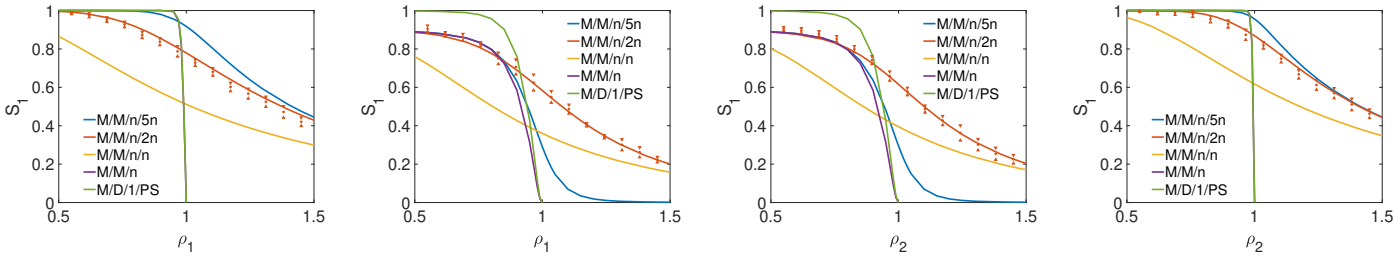
5.2. Impact of User-Edge Latency

We discuss the impact of latency between gamers and VNFs by reporting in Figs. 3 to 6 success probabilities of players–according to the definition given in (10)–and of the group, according to what defined in (11), considering five possible average values for one-way delays, assuming symmetric paths, hence U_{ij}=D_{ij}: 10, 15, 20, 25, and 30 ms, in different conditions of edge server load and buffering. On top of the average delay we consider an additive random value taken from a triangular distribution between t_m=−2 ms and t_M=2 ms, to account for the variable component of the delay incurred in the RAN (whereas the average RAN delay is already counted in U_{ij} and D_{ij}). The timeout is set to T_O=75 ms, while the delay between the two edge servers is 10 ms. Latencies between 50 and 100 ms are normally considered the limit for an enjoyable experience of real time games (see for example [36, 37]). Inter-edge delays depend significantly on the specific network and edge server distribution within the network; values between few ms and some tens of ms can be considered realistic. For a fair comparison of different scenarios, we impose the load of used edge servers to be fixed to either 0.9 or 0.99, which, once we have calculated the traffic due to the two reference gamers g_1 and g_2, is obtained by appropriately tuning the quantity of background traffic corresponding to other users. Notice that we use as definition of the edge server load the ratio $\rho=\frac{\lambda}{n\mu}$, so that as ρ approaches 1, available computing resources saturate.

Figs. 3 and 4 report the success probability for the two gamers with very large (actually, infinite) edge server buffers. Figs. 5 and 6 show similar results, with k=2n, i.e., 5 processors and 5 positions in the buffer at e_1, and 10 processors and 10 positions in the buffer at e_2.

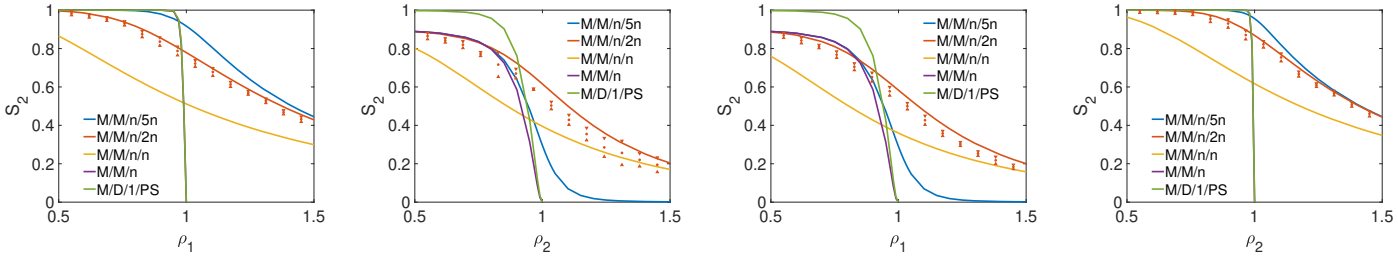
Note that in Scenario (1,1) results are invariant with respect to d_{22}, since e_2 is not used; hence, we only report results for variable d_{11}. Conversely, in Scenario (2,2) results are invariant with respect to d_{11}, because e_1 is not used; hence, we only report results vs. d_{22}. In Scenario (2,1) results are invariant to both d_{11} and d_{22} because g_1 connects to e_2 and g_2 to e_1. Since we set d_{12}=d_{21}=20 ms, results for Scenario (2,1) correspond to those of Scenario (1,2) with d_{11}=d_{22}=20 ms.

We clearly see that performance degrades with higher latency (i.e., higher distance) between gamers and VNFs, and when approaching the edge server capacity. Both effects are quite intuitive. We can also observe that very large buffers are useful



(a) Scenario $(1,1)$ (b) Scenario $(1,2)$ (c) Scenario $(2,1)$ (d) Scenario $(2,2)$

Figure 7: Success probability of gamer g_1 vs load on the associated edge with T_O=75 ms and delays: d_{ij}=20 ms, $\forall\, i,j \in \{1,2\}$ and d_s=10 ms, $d_{12}^{(s)}$=20 ms. Experimental measures with confidence intervals are reported for the M/M/n/2n case



(a) Scenario $(1,1)$ (b) Scenario $(1,2)$ (c) Scenario $(2,1)$ (d) Scenario $(2,2)$

Figure 8: Success probability of gamer g_2 vs load on the associated edge with T_O=75 ms and delays: d_{ij}=20 ms, $\forall\, i,j \in \{1,2\}$ and d_s=10 ms, $d_{12}^{(s)}$=20 ms. Experimental measures with confidence intervals are reported for the M/M/n/2n case

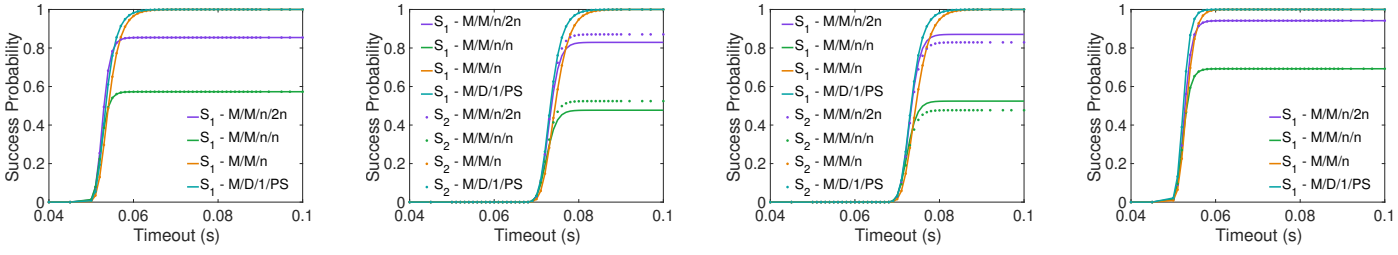
in some cases, not in others. E.g., Scenario $(2,1)$ at 90% load performs consistently better with shorter buffers, and almost all scenarios at 99% load perform better with shorter buffers, except for Scenario $(2,2)$, with d_{11}=10 ms.

These results indicate that choosing a buffer size at game VNFs is an important issue. Serving all incoming requests is not useful when approaching saturation because this risks increasing latency for all requests to the point that they miss their deadline. It would be better to instead drop some requests to ensure that those that are served meet their deadline. Not excessively large buffers might be therefore preferable.

5.3. *Impact of Buffers at the Edge*

The next set of results looks more closely at the effect of buffering in game VNFs. Figs. 7 and 8 plot the success probability of g_1 and g_2 versus the load of the selected edge servers. The distance between gamers and game VNFs is 20 ms for each gamer-VNF pair, while other parameters are like before. Each figure contains five curves that correspond to (i) no buffer, (ii) a number of positions in the buffer equal to the number of servers (k=2n, with n=5 at e_1 and n=10 at e_2), or (iii) 4 times the number of servers (k=5n), and infinite buffer with either (iv) exponential service times, or (v) deterministic service times—with one high-speed processor that, for fairness of comparison, serves requests n times faster than the single processor used with the other types of queues in which n processors are used—and a PS scheduler.

In general, we see that with infinite buffers the success probability drops to zero when ρ=1 because queues reach saturation and delays diverge. On the contrary, the presence of a finite buffer yields success probabilities that in some cases remain at acceptable levels, even in (light) overload. This is because the finite buffer generates losses that keep the workload at a feasible level, thus allowing a fraction of the gamers' interactions to reach their destination within the specified timeout. For example, in Scenario $(2,2)$ at load 1.25 both gamers see success probabilities of the order of 0.6. This implies that the gaming service can survive periods of temporary overload with reduced



(a) Scenario $(1,1)$ (b) Scenario $(1,2)$ (c) Scenario $(2,1)$ (d) Scenario $(2,2)$

Figure 9: Success probability of gamers g_1 and g_2 vs timeout, for variable buffer sizes; d_{ij}=20, $\forall\, i,j \in \{1,2\}$, $d_{12}^{(s)}$=20, d_s=10 and load on edges: ρ_1=ρ_2=0.9. Results for S_2 in scenarios $(1,1)$ and $(2,2)$ overlap those for S_1, therefore the legend for S_2 is omitted.

performance, avoiding blackouts. Indeed, slow variations of the number of gamers or of their activity, hence of the VNF load, can translate into the operating point moving along the curves in Figs. 7 and 8.

It is also interesting to observe that, with the chosen parameters, the allocation of both gamers to the same game VNF yields success probabilities equal to $\sim$1 for low loads, which is not possible with the allocations to different game VNFs (except for the case of the M/D/1-PS model), due to the additional 20 ms necessary to go from one edge server to the other. The reason why the M/D/1-PS model behaves differently is twofold. First of all, in low load the probability of a small number of customers in the queue is high so that the whole processing capacity is divided among few, each receiving a higher share than with a predefined division in n shares. Second, the deterministic service time avoids long service instances that result from sampling the tail of the exponential distribution. This is on the one hand an indication of the fact that using fewer powerful processors yields better performance than using many slower ones, and on the other of the fact that assuming exponential service times leads to pessimistic estimates of the overall system performance.

5.4. Impact of Timeout

The delay with which a gamer visualizes the effect of the actions of gamers that are playing in her game environment is a key QoS metric for online gaming. Curves in Fig. 9 show the success probability of the two gamers in the four possible scenarios with different values for the buffer sizes at the two edge servers (buffer sizes are however proportional to the number of processors at each server). The load of each server is 0.9. Delays from gamers to game VNFs and between game VNFs are like in the previous case.

In Scenario $(1, 2)$ as well in $(2, 1)$ we clearly see that without buffering only about 50% of gamer's commands at most are represented in the video stream within the chosen timeout. Very large (infinite) buffers allow success probabilities to reach one, but only for timeouts of the order of about 85 ms or more (with little benefit deriving from deterministic service times).

In the two scenarios in which a single game VNF is used, timeouts are easily met in the very large buffer cases for values as low as 60 ms. On the contrary, with small or no buffer, the loss at edge servers prevents success probabilities to reach desirable levels, with the possible exception of Scenario $(2, 2)$, provided a success probability around 95% is acceptable.

5.5. Cloud versus Edge Gaming

Fig. 10 presents an intriguing set of results comparing edge gaming with legacy cloud gaming. In the edge scenarios, gamers are connected to a GaaS slice hosting two game VNFs on two different edges (namely, "edge VNFs"). Gamers experience the same distance from edge servers, respectively 10 ms to edge server 1 and 15 ms to edge server 2. We also consider different values of latency between edge servers, from 10 to 35 ms, each case generating a different curve in the figure, as indicated with different labels and colors. In contrast, in the

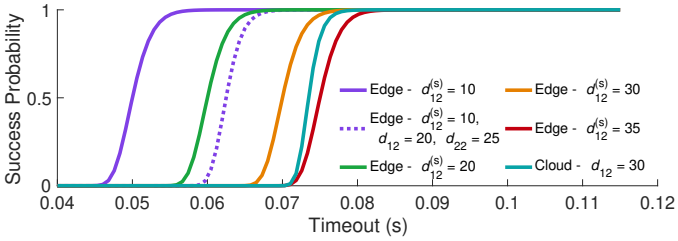


Figure 10: Cloud gaming scenario versus edge gaming scenario with a single reference group, with server load ρ=90%, servers modeled as M/M/n queues with n=20 in the cloud and n=10 in the edge. Each curve corresponds to a different distance between edge servers.

cloud gaming scenario, both gamers connect to the same "cloud VNF", which is 30 ms away from each of them. The load of the server is fixed at 90%, with each edge VNF using 10 processors ($n = 10$). compared to 20 processors ($n = 20$) in the cloud VNF. Both configurations assume infinite buffer capacity, so that the server facilities are modeled through M/M/n queues. Notice that, fixing the load across servers implies that each edge performs half of the work done by the cloud server, but this is a reasonable assumption since edge architectures typically distribute the workload across many less powerful edge servers instead of relying on a few highly powerful cloud servers. This setup naturally leads to a more distributed workload across edge servers for the same population of gamers.

We clearly see that the edge VNFs cannot satisfy timeout values less than 46 ms even in the case in which the two edge servers are closer to one another (i.e., 10+10+15 to go and come back to the game server, 10 for video preparation, 1 for processing at the edge). Similarly, the cloud VNF cannot provide latency lower than 71 ms (i.e., 30+30+10+1 ms). Success probabilities grow rapidly beyond those latency values and reach close to 1 at around 58 ms in the *best* case of the edge VNFs, and 78 ms in the cloud case. These results underline the better performance of various plausible edge scenarios over the cloud setting, despite the absence of inter-edge server communication delays in the cloud scenario.

Edge scenarios in Fig. 10 differ primarily in the fixed delay between edge servers, which varies from 10 ms (purple solid curve), to 35 ms (red solid curve). Additionally, a dotted curve represents an edge scenario where the fixed delay between edge servers is 10 ms, and delays between edge servers and users are increased by 10 ms with respect to the other cases. The figure shows that delays on the gamer-server link have a heavier impact on performance than delays on the inter-edge link. Indeed, the dotted curve obtained by adding 10 ms on the gamer-edge link, with 10 ms between servers, is worse than the green solid curve at 20 ms between edge servers.

Interestingly, the cloud scenario only outperforms the edge scenario when the delay between the two edge servers exceeds the delay between gamers and cloud VNF.

The advantage of the edge architecture is intrinsic in the distribution of computing facilities over the network, whose objective is to reduce the latency between users and computing facilities, and is especially beneficial for latency-constrained services, like the gaming application we are considering.

5.6. Measuring the performance of a group of gamers in the wild

To assess the online gaming performance in real-world edge computing scenarios, we set up a distributed GaaS architecture akin to the one described in Fig. 1. Specifically, we developed a flexible set of instrumented microservices providing the building blocks of our architecture. Three main blocks were defined: *client*, *server*, and *UPF*. The setup architecture is depicted in Fig. 11. The micro-services comprise QUIC endpoints, are coded in Golang, and are containerized using Docker.

The *client* container consists of several elements contributing to setting up an active connection with the game server and also to tracing and measuring packet receptions, losses, and time statistics. The packet generator is responsible for modulating the traffic flow, which subsequently gets marked with the specific type—either control (*Ctrl*) or feedback (*Fb*), respectively. Moreover, for the *Ctrl* flow, the marker adds a timestamp, the client identifier, and a unique identifier for the packet. Then, dedicated *goroutines* (i.e., lightweight Golang threads) handle packet transmissions through QUIC connections; these goroutines are also responsible for collecting data and computing statistics from the incoming packets by matching identifiers, with the help of a list of *pending* control threads (i.e., control packets already sent, whose incorporation in the game stream is not yet completed). Uplink traffic passes through the *UPF* container, emulating the behavior of a tunnel between the player and the game server with a QUIC connection. Specifically, the *UPF* implements a single server with a finite but huge buffer size. At this point, requests are routed toward the game server associated with the player originating the traffic flow, obtaining the network setting of Fig. 11. Packets incoming to the *Edge* containers are either local—that is, incoming from the associated player—or *Inter-edge control* packets coming from the other game server, see the solid and dashed lines in Fig. 11 between the two edge servers. Local packets get to the QUIC endpoint where *Fb* packets get discarded, whereas *Ctrl* packets originate new *Inter-edge control* packets (that are necessary for the remote VNF to generate the video to be delivered to the remote gamer). Also, identifiers and timestamps associated with *Ctrl* packets get inserted in packets carrying the video stream, marking the fact that such frames report the updates related to players' control instructions, as depicted at the bottom of Fig. 11. Similarly, packets incoming from *Inter-edge control* flow get integrated into the downlink stream, with the only difference being that they are associated with a different QUIC endpoint. These timestamps and identifiers are necessary for the computation of the delays with which game commands are incorporated in the video stream observed by players.

With the above, we validated our model by comparing model predictions and experiment measurements with two game VNFs positioned at a distance of about 20 ms. Both gamers experienced similar latency when connecting to their nearest game VNF, also situated about 20 ms away.

Specifically, the game VNFs are 19.30 ms away from each other, with a standard deviation (σ) of 2.23 ms. The two players are 20.02 ms (σ=1.76 ms) and 19.77 ms (σ=2.55 ms) away

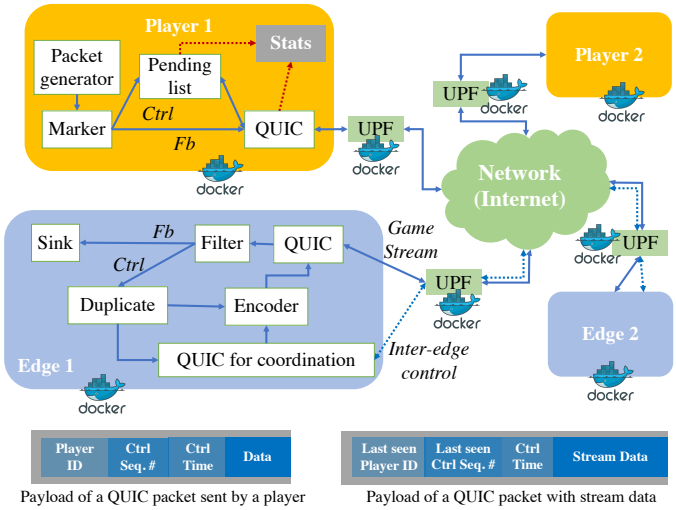


Figure 11: Experimental setup. To enable the computation of statistics, control (Ctrl) and stream packets carry the relevant ID of players and control sequence numbers, in addition to timestamps.

from their associated game VNF, respectively. The time required to implement changes to the video stream in the game VNF is set to d_s=10 ms. Game VNFs have either 5 or 10 processors and the same number of positions in the buffer. Gamers access their game VNF through their associated UPF, requiring constant times equal to 0.1 ms to process uplink packets and 0.2 ms for the downlink ones. Gamers generate an uplink stream of 1 000 control packets per second of 1 000 bytes each, with only one in every one hundred packets delivering commands to the game streamer.

We evaluate each gamer's success probability considering a latency threshold of 75 ms with different background traffic intensity; in our experiments, the background traffic ranges from ρ_i=0.5 to 1.5, with a step size of 0.1. The comparative evaluation of our model with real-world measurements is illustrated in Figs. 7 and 8, with 90% confidence intervals. We can observe a very good match between model predictions and experiments in spite of the many simplifications introduced in the model.

5.7. Performance with larger sets of gamers and servers

So far, we considered the performance of pairs of gamers that can use two edge servers. However, our model allows for the study of scenarios involving multiple edge computing facilities and multiple gamers. Hence, to show the capability of our model, we next report numerical results for scenarios comprising ten gamers and three edges. Each specific scenario is described by a tuple, where the number of elements corresponds to the number of gamers considered in the experiment. Each value in the tuple specifies the particular edge to which each gamer is connected (if the k-th entry of the tuple equals j, this means that gamer k is connected to edge j). We define *communication clusters* as sets of gamers who are playing within the same game environment and are thus actively communicating with each other. For these clusters, we compute the communication success probability according to (11). This success probability can be determined for each individual user, for each pair of users, as well as for the entire communication cluster. The

experimental setting mirrors the one of Section 5.1, except for the fact we are now investigating a setting in which ten users connect on three different available edge servers. We assume all three edges have $n = 10$ cores allocated to the network slice devoted to GaaS, and that fixed delays are as follows.
Given $X = \{i, j \mid (j = 1 \wedge 1 \leq i \leq 3) \vee (j = 2 \wedge 4 \leq i \leq 6) \vee (j = 3 \wedge 7 \leq i \leq 10)\}$ we have delays: $d_{ij} = 15$ ms, $\forall\, i, j \in X$ and $d_{ij} = 20$ ms, $\forall\, i, j \notin X$; $d_{ij}^{(s)} = 20$ ms, $\forall i, j \in \{1, 2, 3\}$, $d_s = 10$ ms. That is, gamers 1, 2, and 3 are located nearest to edge 1; gamers 4, 5, and 6 are nearest to edge 2; while the rest are closest to edge 3.

We present results for three cases. The first one is a scenario in which each gamer is connected to her closest edge. The second is a scenario in which two gamers connect to a farther edge. In the third scenario, four out of ten gamers do not connect to their closest edge. Of course, we expect performance to be best for the first scenario, and worst for the third. The timeout is set to 75 ms. In Figs. 12, 13, 14 we report curves of the success probability versus load (the load values start from the traffic that is generated by the ten users, and then increase because of the addition of background traffic representing additional gamers) for the whole group of 10 gamers, and for two partitions comprising respectively the first and the last five gamers in the group. As usual, we consider different queuing models for the edge operations. We can note, as expected, that in all cases the worst performance is obtained when the edge is not equipped with a waiting line (M/M/n/n model), and that this leads to quite unsatisfactory performance, even at low loads, when we look at the entire group of 10 gamers. We also observe that the allocation of gamers to edges different from the closest one leads to significant performance losses, especially when we look at the entire group of 10 gamers. This is due to the fact that the success probability is obtained as the product of 100 components, each corresponding to the success probability in the communication between one (directed) pair of gamers. Figs. 15 and 16 report the values of the failure probability for the 100 components for scenarios where all gamers connect to their closest edge and with two out of ten gamers connected to their non-closest edge respectively, and for edge models M/D/1-PS and M/M/n/n (that correspond to the best and worst performance in Figs. 12, 13, 14) under load factor 0.5. From those values we can clearly see both the cost of no waiting line and the cost of associating a gamer to a non-closest edge.

Finally, in Fig. 17, we set the load of the three edges to $\rho = 0.9$ and we observe the success probability as a function of the timeout, for the edge model M/M/n. Other queuing disciplines and load factors, not reported here, show similar behaviours. The figure shows that small differences in the timeout can be critical, as the transition between low and high success probability is fast and a few milliseconds can make the difference, which justifies the need to carefully select performance guarantees offered in SLAs and to implement precise gamer allocation strategies. The figure also shows that the larger the communication cluster, the harder it is to achieve high performance levels. In practice, adding more gamers to a communication party is equivalent to decreasing the application timeout, which cannot be reduced below feasible values determined by the network topology.

6. Performance of Concurrent Groups of Gamers

Having assessed the performance of a group of gamers and the validity of our model in the previous section, we now move to the evaluation of multiple concurrent groups of gamers. Since the allocation of each group impacts on the performance of all other groups using the same set of servers, we will use the definitions of success probabilities given in (12) to (15) in order to optimize the allocation of groups as they asynchronously arrive. However, given the high complexity of the resulting optimization process, we will resort to define and evaluate multiple possible heuristics.

6.1. Optimization and heuristics

To evaluate different gamer/edge allocation policies we implemented and compared optimal solutions as well as several heuristics. We consider that groups arrive sequentially, one at a time, and we neglect allocation costs, e.g., we focus on technical performance figures only.

In the optimization, once a new group arrives, *all* existing allocations are reconsidered, looking for a new optimal allocation of all gamers present in the system. This is a case of unconstrained optimization whose objective function is the system success probability. However, since we have multiple possible definitions of system success probability, we will compare the results obtained by maximizing each of the definitions given in (12) to (15).

Note that our unconstrained optimization accounts for migrations of gamers, assuming idealistically that players can be instantaneously reallocated to any of the available servers. Hence, optimization results presented in what follow are a non-necessarily strict upper bounds of achievable performance.

Since the objective functions used for the optimization are convex but non-linear, and depend on binary decisions (allocate each gamer to a server or to another), we have an unconstrained convex binary non-linear problem. This class of problem is NP-Hard, as we have, in essence, a bin-packing problem with some extra additive weights due to inter-edge communications, and a non-linear (but convex) utility. Hence we resort to heuristics and we further only consider the case of *online* algorithms that cannot undo the current allocation of previously arrived groups of gamers. We do that because migrations of gamers from a server to another could cause service discontinuities. Heuristics are *greedy* in the sense that when a new group of gamers arrives, the players in the group are allocated optimally with respect to the chosen metric and subject to the constraint that existing allocations cannot be changed.

The first set of heuristics that we define will use allocation policies that aim to optimize the collective user experience by maximizing one of the specific metrics of group success probability, as defined in Eqs. (12)-(15). These allocation policies can be described as follows:

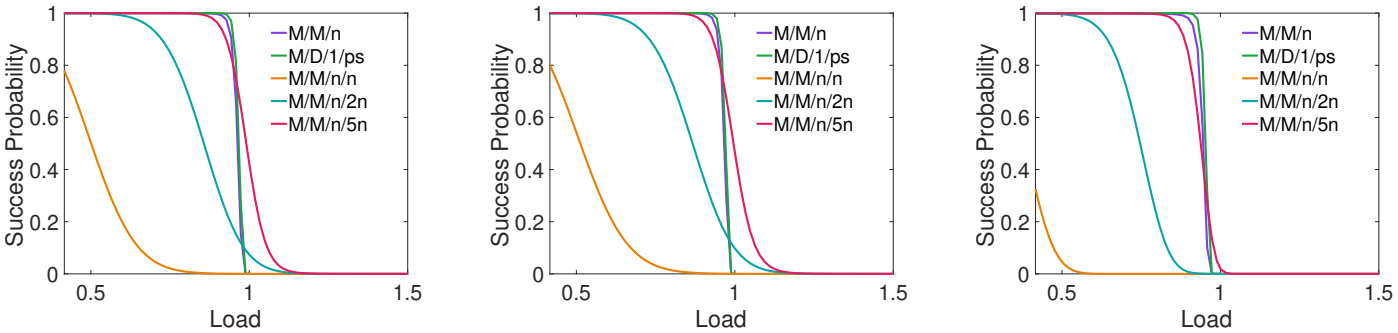


Figure 12: Overall success probability computed as in (11) vs load (on all edges) with T_O=75 ms. Given $X = \{i,j \mid (j = 1 \wedge 1 \leq i \leq 3) \vee (j = 2 \wedge 4 \leq i \leq 6) \vee (j = 3 \wedge 7 \leq i \leq 10)\}$ we have delays: d_{ij}=15, $\forall\, i,j \in X$ and d_{ij}=20, $\forall\, i,j \notin X$; $d_{ij}^{(s)}$=20, $\forall i,j \in \{1,2,3\}$, $d_s = 10$, for scenario $(1,1,1,2,2,2,3,3,3,3)$.

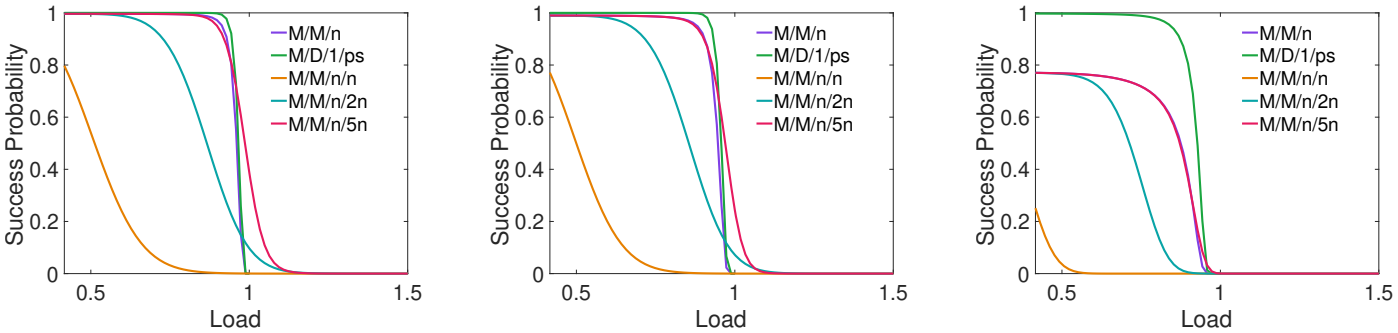


Figure 13: Overall success probability computed as in (11) vs load (on all edges) with T_O=75 ms. Given $X = \{i,j \mid (j = 1 \wedge 1 \leq i \leq 3) \vee (j = 2 \wedge 4 \leq i \leq 6) \vee (j = 3 \wedge 7 \leq i \leq 10)\}$ we have delays: d_{ij}=15, $\forall\, i,j \in X$ and d_{ij}=20, $\forall\, i,j \notin X$; $d_{ij}^{(s)}$=20, $\forall i,j \in \{1,2,3\}$, $d_s = 10$, for scenario $(1,1,1,1,2,2,2,3,3,3)$.

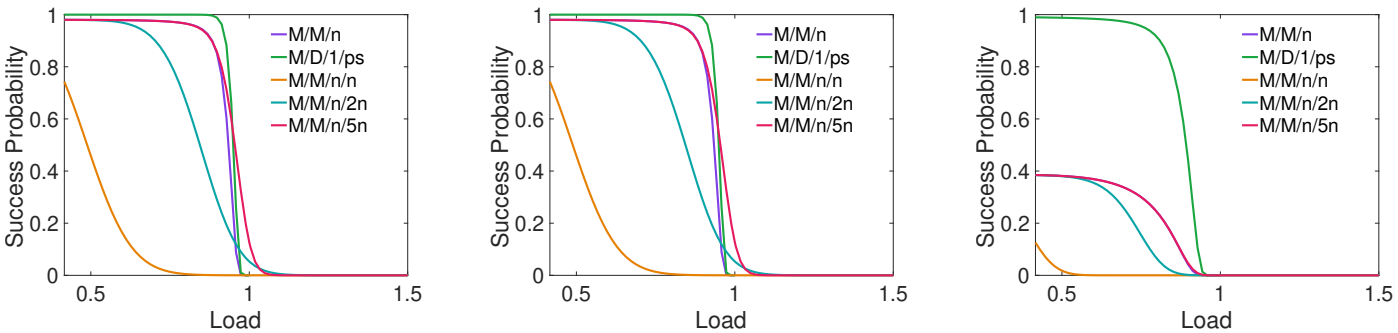


Figure 14: Overall success probability computed as in (11) vs load (on all edges) with T_O=75 ms. Given $X = \{i,j \mid (j = 1 \wedge 1 \leq i \leq 3) \vee (j = 2 \wedge 4 \leq i \leq 6) \vee (j = 3 \wedge 7 \leq i \leq 10)\}$ we have delays: d_{ij}=15, $\forall\, i,j \in X$ and d_{ij}=20, $\forall\, i,j \notin X$; $d_{ij}^{(s)}$=20, $\forall i,j \in \{1,2,3\}$, $d_s = 10$, for scenario $(3,3,1,2,2,2,3,3,1,1)$.

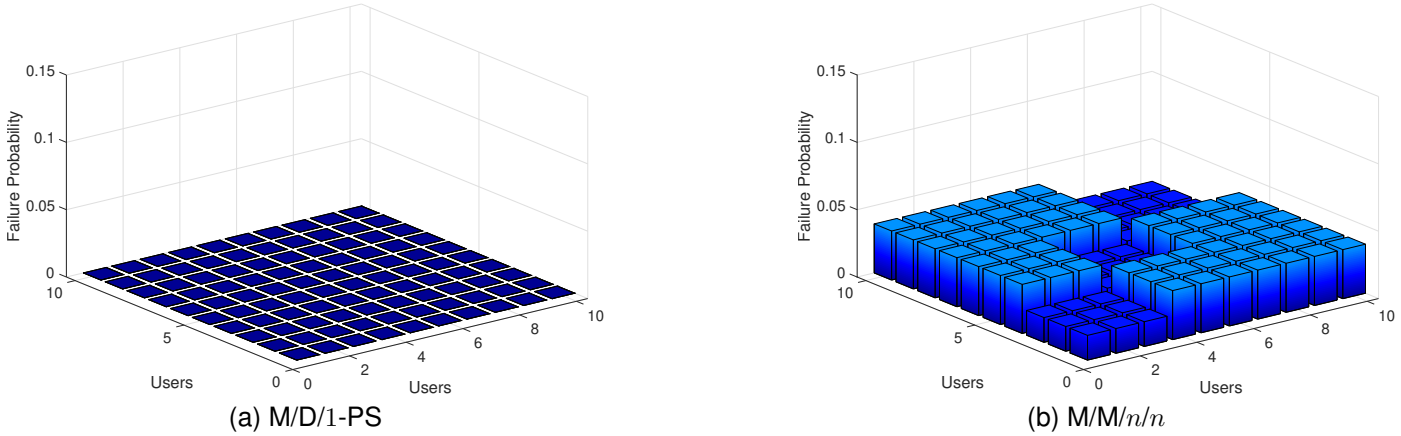


Figure 15: Failure probability computed as the complement to one of the success probability in (11) for each pair of gamers in scenario $(1, 1, 1, 2, 2, 2, 3, 3, 3, 3)$, considering the same setting parameters as in Fig. 12.

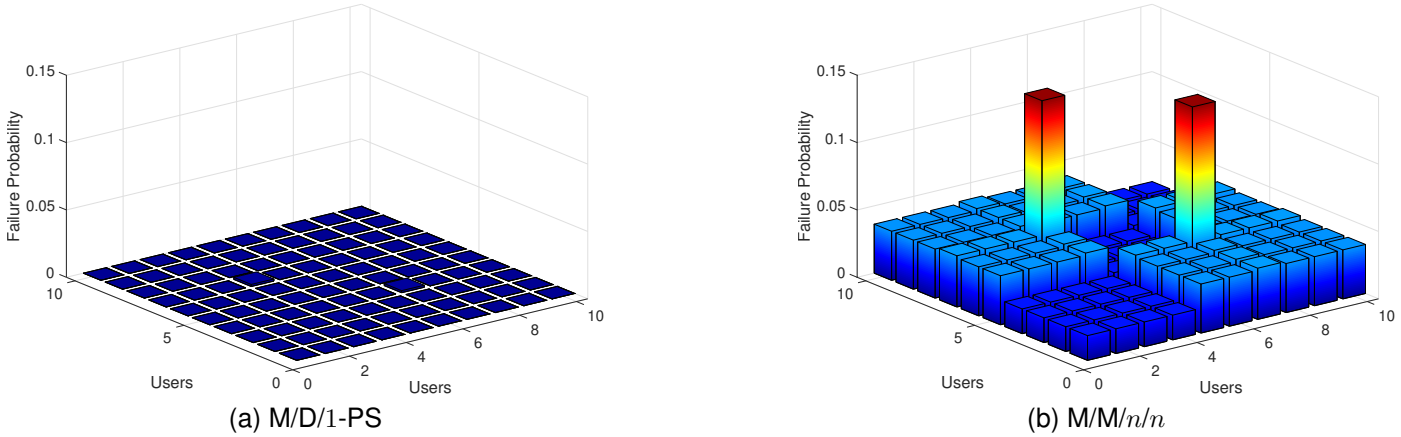


Figure 16: Failure probability computed as the complement to one of the success probability in (11) for each pair of gamers in scenario $(1, 1, 1, 1, 2, 2, 2, 3, 3, 3)$, considering the same setting parameters as in Fig. 14.

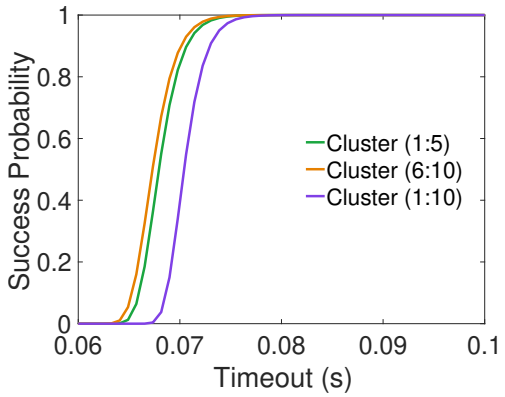


Figure 17: Scenario $(1, 1, 1, 2, 2, 2, 3, 3, 3, 3)$. Overall success probability computed as in (11) vs timeout for cluster of gamers $(1 : 5)$, $(6 : 10)$, $(1 : 10)$; Given $X = \{i, j \mid (j = 1 \wedge 1 \leq i \leq 3) \vee (j = 2 \wedge 4 \leq i \leq 6) \vee (j = 3 \wedge 7 \leq i \leq 10)\}$ we have d_{ij}=15, $\forall\, i, j \in X$ and d_{ij}=20, $\forall\, i, j \notin X$; $d_{ij}^{(s)}$=20, $\forall\, i, j \in \{1, 2, 3\}$, d_s=10 and load on edges: ρ_1=ρ_2=ρ_3=0.9, edges modeled with M/M/n buffers.

- `maxAvg` maximizes the arithmetic mean of success probabilities across all groups, i.e. it maximizes Eq. (12)
- `maxWeightedAvg` maximizes a weighted mean of success probabilities, where each group's weight is proportional to its size, i.e., it maximizes Eq. (13)
- `maxProd` maximizes the product of all group success probabilities, i.e. it maximizes Eq. (14)
- `maxMin` maximizes the success probability of the worst-performing group, i.e. it maximizes Eq. (15).

A fifth policy, `selfish`, adopts a greedy policy approach by only maximizing the success probability for the most recently arrived group. In addition to these, we also evaluate a load-balancing policy, `minLoadStDev`, which seeks to improve system stability by minimizing the standard deviation of the load across all edge servers.

Note that we are interested in online gaming performance and in the gap between optimal allocation and heuristic policies. Instead, we are not interested in the complexity of optimization

Table 1: Comparison of different allocation strategies

Policy	Type	Allocation	Loads				Group Success Probabilities				Overall Success Probability			
			Edge 1	Edge 2	Edge 3	Std. Dev	$G^{(1)}$	$G^{(2)}$	$G^{(3)}$	$G^{(4)}$	S_{avg}	S_{wavg}	S_{prod}	S_{min}
maxAvg	Heur	[3, 3, 3, 1, 2 | 3, 2, 3]	0.26	0.55	0.62	0.144	0.75	0.88			**0.82**	0.80	0.66	0.75
	Opt	[3, 3, 3, 2, 2 | 3, 1, 3]	0.41	0.44	0.62	0.102	0.79	0.88			**0.84**	0.82	0.70	0.79
maxWeightedAvg	Heur	[3, 3, 3, 1, 2 | 3, 2, 3]	0.26	0.55	0.62	0.144	0.75	0.88			0.82	**0.80**	0.66	0.75
	Opt	[3, 3, 3, 2, 2 | 3, 1, 3]	0.41	0.44	0.62	0.102	0.79	0.88			0.84	**0.82**	0.70	0.79
maxProd	Heur	[3, 3, 3, 1, 2 | 3, 2, 3]	0.26	0.55	0.62	0.144	0.75	0.88			0.82	0.80	**0.66**	0.75
	Opt	[3, 3, 3, 2, 2 | 3, 1, 3]	0.41	0.44	0.62	0.102	0.79	0.88			0.84	0.82	**0.70**	0.79
maxMin	Heur	[3, 3, 3, 1, 2 | 3, 2, 3]	0.26	0.55	0.62	0.144	0.75	0.88			0.82	0.80	0.66	**0.75**
	Opt	[3, 3, 3, 2, 3 | 2, 1, 2]	0.41	0.51	0.58	0.069	0.82	0.84			0.83	0.83	0.69	**0.82**
selfish	Heur	[3, 3, 3, 1, 2 | 3, 2, 3]	0.26	0.55	0.62	0.144	0.75	0.88			0.82	0.80	0.66	0.75
minLoadStdDev	Heur	[3, 2, 3, 2, 1 | 3, 3, 2]	0.41	0.51	0.58	0.069	0.68	0.92			0.80	0.77	0.63	0.68
	Opt	[3, 3, 2, 1, 2 | 3, 3, 1]	0.52	0.54	0.53	0.008	0.56	0.85			0.70	0.67	0.48	0.56
maxAvg	Heur	[3, 3, 3, 1, 2 | 3, 2, 3 | 3, 1 | 2, 3]	0.52	0.81	0.89	0.148	0.16	0.48	0.78	0.71	**0.53**	0.43	0.04	0.16
	Opt	[1, 1, 1, 1, 1 | 3, 3, 3 | 2, 3 | 2, 3]	1.70	0.54	0.56	0.484	0.00	0.96	0.95	0.95	**0.71**	0.56	0.00	0.00
maxWeightedAvg	Heur	[3, 3, 3, 1, 2 | 3, 2, 3 | 1, 3 | 2, 3]	0.67	0.81	0.83	0.066	0.17	0.54	0.74	0.74	0.55	**0.45**	0.05	0.17
	Opt	[1, 1, 1, 1, 1 | 3, 3, 3 | 2, 3 | 2, 3]	1.70	0.54	0.56	0.484	0.65	0.16	0.61	0.61	0.51	**0.52**	0.04	0.16
maxProd	Heur	[3, 3, 3, 1, 2 | 3, 2, 3 | 3, 1 | 2, 3]	0.52	0.81	0.89	0.148	0.16	0.48	0.78	0.71	0.53	0.43	**0.04**	0.16
	Opt	[3, 3, 3, 3, 3 | 2, 2, 2 | 1, 2 | 1, 3]	0.81	0.77	0.78	0.013	0.38	0.47	0.62	0.68	0.54	0.49	**0.08**	0.38
maxMin	Heur	[3, 3, 3, 1, 2 | 3, 2, 3 | 2, 1 | 3, 2]	0.52	0.98	0.79	0.170	0.18	0.46	0.64	0.64	0.48	0.40	0.03	**0.18**
	Opt	[3, 3, 3, 1, 3 | 2, 3, 1 | 2, 2 | 2, 2]	0.52	1.04	0.74	0.194	0.38	0.47	0.68	0.59	0.53	0.49	0.07	**0.38**
selfish	Heur	[3, 3, 3, 1, 2 | 3, 2, 3 | 3, 1 | 3, 2]	0.52	0.71	0.95	0.178	0.13	0.46	0.74	0.72	0.51	0.41	0.03	0.13
minLoadStdDev	Heur	[3, 2, 3, 2, 1 | 3, 3, 2 | 2, 1 | 3, 1]	0.92	0.95	0.91	0.018	0.10	0.65	0.55	0.62	0.48	0.40	0.02	0.10
	Opt	[3, 3, 3, 3, 2 | 3, 3, 2 | 1, 2 | 1, 2]	0.81	0.79	0.78	0.009	0.30	0.61	0.61	0.59	0.53	0.47	0.06	0.30

and heuristic allocation policies, so that we do not study how to make optimization and heuristics fast, which would deserve a separate study.

6.2. Performance comparison with multiple groups

We assess optimization and proposed heuristics in two distinct scenarios, at a small and a larger network scale, respectively. In both setups we assume that three edge servers are connected to the GaaS slice, and users with an even ID generate 500 packets per second, while those with an odd ID generate 800 packets per second. Each individual processor on an edge server can process 1000 packets per second. The buffer capacity at each edge is twice its number of servers and the timeout is set, as before, to 75 ms. Network latencies are again defined by a delay matrix. The user-to-edge latency consists of a fixed base delay of 5 ms plus a variable component drawn from a uniform random distribution ranging between 0 and 25 ms. Uplink and downlink delays are symmetric, but to the edge-to-user latency we must add 10 ms delay for the video generation. The latency between any two edge servers is set to 10 ms. For reproducibility, the random number generator used for the choice of variable delays is initialized with the same seed in all experiments, ensuring an identical network topology for all tested policies and for optimization.

6.2.1. Small-Scale Scenario

This setting is designed for a detailed analysis and comparison of heuristics with optimal solutions. It involves four groups of 5, 3, 2 and 2 gamers, respectively. The three edge servers are equipped with 2, 3 and 5 servers. The comprehensive results for this scenario are presented in Table 1, where we show the evolution of metrics for both heuristics and optimal solutions, as groups of gamers arrive at the network gaming slice. Results are divided into two "sub-scenarios" one considering only the first two arriving groups of gamers and the second one involving four groups in total.

When considering only two groups, the success probabilities of the heuristic and optimal solutions show no significant difference, except for the `minLoadStdDev` policy, where the load standard deviation of the optimal solution is smaller by an order of magnitude. Looking at overall success probability, optimal allocations generally provide only marginal improvements across most policies. The `minLoadStdDev` policy, however, stands out: it balances the load across edges remarkably well, especially under the optimal allocation, where it achieves an almost perfectly even distribution (loads: 0.52, 0.54, 0.53). This balance, though, comes at the expense of lower overall success probabilities compared to other policies.

When all four groups are considered, the problem's complexity increases, and differences among policies become more pronounced. The performance gap between heuristic and optimal allocations widens significantly. This highlights how, as system complexity grows, the choice of the gamer allocation policy becomes increasingly critical. It also underscores the inherent trade-offs between maximizing overall performance, ensuring fairness across groups, and maintaining balanced resource utilization.

6.2.2. Larger-Scale Scenario

This setting evaluates the scalability and performance of the heuristics under a heavier load resulting from the presence of a larger number of players connected to the gaming slice. It considers 100 groups, each one composed of 2 gamers (hence 200 players in total). The infrastructure is scaled accordingly, with the three edge servers being now equipped with 20, 30 and 50 processors, respectively. Note that in this case we only report the heuristics performance, since computing optimal allocations would require the exploration of $3^{200} \simeq 2.7 \cdot 10^{95}$ options (some of which can be immediately discarded).

Figure 18 compares the heuristics by plotting their minimum group success probability as the number of groups in the network slice increases. The results clearly highlight the superi-

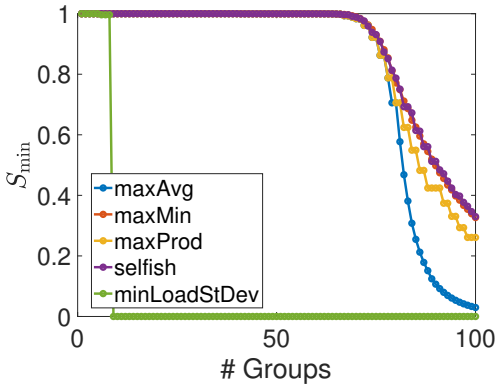


Figure 18: Heuristic Comparison (Success probability from (15))

ority of the selfish policy and of the `maxMin` heuristic, which exhibit almost identical performance. Very poor is instead the performance obtained with the simple load balancing approach of the `minLoadStdDev` heuristic. Similarly poor are the performances of the `maxAvg` ans `maxProd` policies. These results show a tendency to create scenarios where certain user groups suffer from very poor performance. In particular, the load-balancing heuristic (`minLoadStdDev`) focuses solely on edge utilization while completely disregarding group success probabilities, leading to imbalances in user experience despite balanced resource allocation.

The plots in Fig. 19 illustrate the trade-offs associated with different optimization objectives as the system scales. As shown in Fig. 19a the `maxAvg` heuristic consistently achieves the highest average success probability among the considered heuristics, as expected. However, this high average performance comes at the cost of fairness, as reflected by its poor results for the minimum success probability. Conversely, Fig. 19b shows the performance of the `maxMin` heuristic, which of course delivers the best minimum success probability. This reinforces its role as a fairness-oriented policy as it is focused on the worst-case performance. However, this results in a slightly lower average success probability compared to the `maxAvg` heuristic. The heuristic `maxProd` depicted in Fig. 19c, emerges as a balanced compromise. It achieves a significantly better minimum success probability than `maxAvg` while also providing a higher average success probability than `maxMin`.

7. Conclusions

We have proposed an online gaming service deployed on edge computing facilities and studied the performance of the system. Through analysis and experiments, we have highlighted how our proposal is, on the one hand, very promising, while being, on the other hand, also very challenging. Indeed, the downside of the edge option is the need to coordinate gaming engines on different edge facilities and VNFs.

Our analytical results show that, for a number of parameter values that reflect real gaming systems, the edge gaming option can yield lower latency than cloud gaming, provided that the allocation of interacting gamers to edge servers is carefully chosen. It often happens that better performance is achieved by allocating gamers that closely interact on the same edge server, but under some combinations of the system parameters, an allocation of gamers to their closest game VNF can be more effective, in spite of the need for interaction among servers. Our model is also helpful to unveil the dependencies of gamers' performance on server allocations, when multiple groups of gamers share to the same set of servers. In that case, optimization becomes cumbersome, although simple online heuristics can achieve good performance without reshuffling gamers' allocations at each new group arrival. In the case of gamers with wireless internet access who move (for example on a bus or a train) while playing, a relevant issue is the delay due to handovers from one RAN cell to another. The handover events can cause spikes in latency, that disturb the gamer experience, and may require reallocation of the moving gamer to a new edge server. This aspect will be considered in future works.

Our model can provide a very effective tool for managers of GaaS applications since it enables the choice of the best performance option upon the arrival of new groups of gamers. This allows the GaaS slice to operate at its best performance and guarantee the best possible QoE to gamers.

References

[1] European Commission, European Media Industry Outlook report, https://digital-strategy.ec.europa.eu/en/library/european-media-industry-outlook (2023).
[2] J. Clement, Mobile gaming market in the United States - Statistics & Facts, https://www.statista.com/topics/1906/mobile-gaming/ (2024).
[3] ITU, SG12: Performance, QoS and QoE (2022).
[4] S. Möller, S. Schmidt, J. Beyer, Gaming taxonomy: An overview of concepts and evaluation methods for computer gaming QoE, in: 2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX), 2013, pp. 236–241.
[5] ITU, ITU-T Recommendation ITU-T G.1032, Influence factors on gaming quality of experience, Tech. rep., ITU (2017).
[6] ITU, ITU-T Recommendation ITU-T P.809, Subjective evaluation methods for gaming quality, Tech. rep., ITU (2018).
[7] ITU, Recommendation ITU-T G.1072, Opinion model predicting gaming quality of experience for cloud gaming services, Tech. rep., ITU (2020).
[8] K.-T. Chen, Y.-C. Chang, H.-J. Hsu, D.-Y. Chen, C.-Y. Huang, C.-H. Hsu, On the Quality of Service of Cloud Gaming Systems, IEEE Trans. on Multimedia 16 (2014).
[9] A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, D. Giordano, A Network Analysis on Cloud Gaming: Stadia, GeForce Now and PSNow, Network 1 (2021).
[10] O. S. Peñaherrera-Pulla, C. Baena, S. Fortes, E. Baena, R. Barco, Measuring Key Quality Indicators in Cloud Gaming: Framework and Assessment Over Wireless Networks, Sensors 21 (2021).
[11] M. Carrascosa, B. Bellalta, Cloud-gaming: Analysis of Google Stadia traffic, Computer Communications 188 (2022).
[12] P. Graff, X. Marchal, T. Cholez, S. Tuffin, B. Mathieu, O. Festor, An Analysis of Cloud Gaming Platforms Behavior under Different Network Constraints, in: 2021 17th International Conference on Network and Service Management (CNSM), 2021, pp. 551–557.
[13] X. Marchal, P. Graff, J. R. Ky, T. Cholez, S. Tuffin, B. Mathieu, O. Festor, An Analysis of Cloud Gaming Platforms Behaviour Under Synthetic Network Constraints and Real Cellular Networks Conditions, J. Netw. Syst. Manage. 31 (2023).
[14] Y. Kim, Y. Choi, Y. C. Lee, H. Han, S. Kang, E-Render: Enabling UHD-Quality Cloud Gaming Through Edge Rendering, IEEE Access 10 (2022).

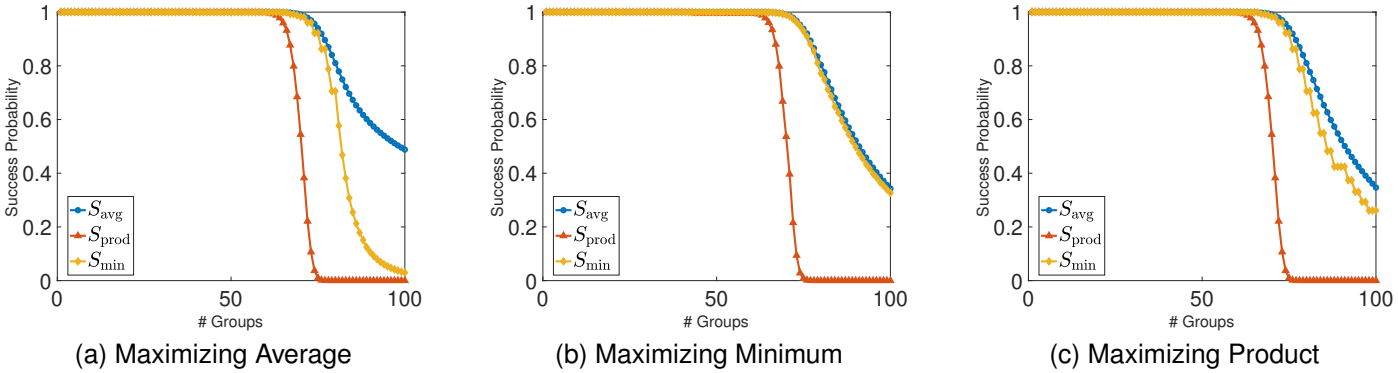


Figure 19: Overall success probability (using different metrics) vs. number of groups for different heuristics.

[15] D. Olliaro, V. Mancuso, P. Castagno, M. Sereno, M. Ajmone Marsan, Gaming on the Edge: Performance Issues of Distributed Online Gaming, in: 2024 IFIP Networking Conference (IFIP Networking), 2024, pp. 259–267.

[16] G. Z. Papadopoulos, F. Theoleyre, P. Thubert, N. Montavont, IETF Reliable and Available Wireless (RAW): Use Cases and Problem Statement, in: Ad-Hoc, Mobile, and Wireless Networks: 19th International Conference on Ad-Hoc Networks and Wireless, ADHOC-NOW 2020, Bari, Italy, October 19–21, 2020, Proceedings, 2020, pp. 303–314.

[17] M. Bacco, P. Cassarà, A. Gotta, M. Puddu, A Simulation Framework for QoE-Aware Real-Time Video Streaming in Multipath Scenarios, in: Ad-Hoc, Mobile, and Wireless Networks: 19th International Conference on Ad-Hoc Networks and Wireless, ADHOC-NOW 2020, Bari, Italy, October 19–21, 2020, Proceedings, 2020, pp. 114–121.

[18] L. De Giovanni, D. Gadia, P. Giaccone, D. Maggiorini, C. E. Palazzi, L. A. Ripamonti, G. Sviridov, Revamping Cloud Gaming With Distributed Engines, IEEE Internet Computing 26 (2022).

[19] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, D. O. Wu, Improving Cloud Gaming Experience through Mobile Edge Computing, IEEE Wireless Communications 26 (2019).

[20] F. Spinelli, A. Bazco-Nogueras, V. Mancuso, Edge Gaming: A Greening Perspective, Computer Communications 192 (2022).

[21] M. Ghobaei-Arani, R. Khorsand, M. Ramezanpour, An autonomous resource provisioning framework for massively multiplayer online games in cloud environment, Journal of Network and Computer Applications 142 (2019).

[22] Y. Gao, L. Wang, Z. Xie, Z. Qi, J. Zhou, Energy- and Quality of Experience-Aware Dynamic Resource Allocation for Massively Multiplayer Online Games in Heterogeneous Cloud Computing Systems, IEEE Trans. on Services Computing 16 (2023).

[23] Y. Liang, J. Ge, S. Zhang, J. Wu, L. Pan, T. Zhang, B. Luo, Interaction-Oriented Service Entity Placement in Edge Computing, IEEE Trans. on Mobile Computing 20 (2021).

[24] Z. Chen, H. Zhu, L. Song, D. He, B. Xia, Wireless Multiplayer Interactive Virtual Reality Game Systems With Edge Computing: Modeling and Optimization, IEEE Trans. on Wireless Communications 21 (2022).

[25] S. Agarwal, F. Malandrino, C. F. Chiasserini, S. De, VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks, IEEE/ACM Trans. on Networking 27 (2019).

[26] G. Einziger, G. Scalosub, C. F. Chiasserini, F. Malandrino, Virtual Service Embedding With Time-Varying Load and Provable Guarantees, IEEE Trans. on Cloud Computing 11 (2023).

[27] E. T. S. Institute, TS 123 501 - V15.3.0 - 5G; System Architecture for the 5G System (3GPP TS 23.501 version 15.3.0 Release 15), Tech. rep., ETSI (2018).

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

[29] A. Asheralieva, D. Niyato, Y. Miyanaga, Efficient dynamic distributed resource slicing in 6g multi-access edge computing networks with online admm and message passing graph neural networks, IEEE Transactions on Mobile Computing 23 (4) (2024) 2614–2638. doi:10.1109/TMC.2023.3262514.

[30] M.-A. Garcia-Martin, M. Gramaglia, P. Serrano, Network automation and data analytics in 3gpp 5g systems, IEEE Network 38 (4) (2024) 182–189. doi:10.1109/MNET.2023.3321524.

[31] T. J. Ott, The Sojourn-Time Distribution in the M/G/1 Queue with Processor Sharing, J. of Appl. Prob. 21 (1984).

[32] Y. Pan, R. Li, C. Xu, The First 5G-LTE Comparative Study in Extreme Mobility, Proc. ACM Meas. Anal. Comput. Syst. 6 (2022).

[33] R. A. K. Fezeu, E. Ramadan, W. Ye, B. Minneci, J. Xie, A. Narayanan, A. Hassan, F. Qian, Z.-L. Zhang, J. Chandrashekar, M. Lee, An In-Depth Measurement Analysis of 5G mmWave PHY Latency and Its Impact on End-to-End Delay, in: Passive and Active Measurement, 2023, pp. 284–312.

[34] G. Horváth, I. Horváth, S. A. Almousa, M. Telek, Numerical inverse laplace transformation using concentrated matrix exponential distributions, Perform. Evaluation 137 (2020).

[35] G. Horvath, A CME-based numerical inverse Laplace transformation method, https://it.mathworks.com/matlabcentral/fileexchange/71511-a-cme-based-numerical-inverse-laplace-transformation-method, MathWorks (2024).

[36] D. Monaco, A. Sacco, D. Spina, F. Strada, A. Bottino, T. Cerquitelli, G. Marchetto, Real-time latency prediction for cloud gaming applications, Computer Networks 264 (2025) 111235. doi:https://doi.org/10.1016/j.comnet.2025.111235.
URL https://www.sciencedirect.com/science/article/pii/S1389128625002038

[37] S. Vlahovic, M. Suznjevic, L. Skorin-Kapov, The impact of network latency on gaming qoe for an fps vr game, in: 2019 Eleventh International Conference on Quality of Multimedia Experience (QoMEX), 2019, pp. 1–3. doi:10.1109/QoMEX.2019.8743193.