An Edge Gaming Overview

Matthew Spilman
Spilm005@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

Cloud gaming is a concept that is already in practice, but has issues that make it less user friendly for the average user. Cloud gaming aims to make gaming more accessible for people who don't have a console or a PC, but has major issues with latency for many users. Edge gaming is an evolution that fixes some of these, but has its own with energy and migrations. If these issues aren't properly managed, it can be a resource drain or not work optimally. Several papers talk about these issues and potential fixes for them. This paper is an overview of some of their research and findings, talking about the main issues of edge gaming and their proposed fixes for them. It goes over energy solutions and an algorithm that manages migrations.

1 Introduction

As of 2020, there were approximately 3.1 billion active video game players in the world [2]. Many of these people have used cloud gaming, but latency problems have made it much less adopted than one would expect. Gaming through the cloud has the potential to make it easier to play than ever, as one wouldn't need to buy an expensive gaming rig or a console. However, the latency problems make it hard to play certain more action based games such as first person shooters or fighting games. These sorts of games need to have split second precision in order to win sometimes. This compounded with packet loss due to distance makes it nearly impossible to play games that need precise inputs. But by applying aspects of edge computing, a new version of cloud gaming can fix the latency and packet loss issues: edge gaming.

This paper presents the material in three main sections. We start off with Section 2 to talk about a bit of needed information. Section 3 is the comparison between cloud and edge, as this allows the improvements and drawbacks to become extremely clear. This also allows the similarities to become clear as well. Section 4 will be the associated energy issues. The energy drawbacks are an expansion of an issue that cloud gaming already faces, but it is increased or at least equal for edge gaming, with the current solution that many are looking to being green energy sources. Section 5 describes migration issues mostly unique to edge gaming. Edge gaming migrations can cause many issues with latency and packet loss but there are algorithms that aim to solve, or at least alleviate some of these issues. There is also a second kind of migration, or rather offloading of tasks that can take place which is also described. Section 6 is wrapping up the paper with a conclusion.

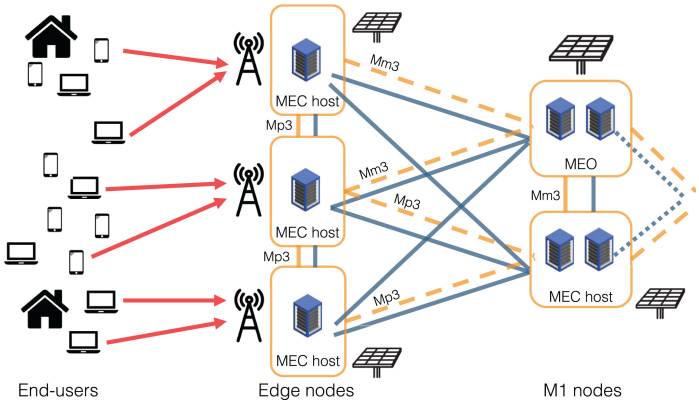

Figure 1. A representation of how edge networks work. The M1 nodes are the central cloud (or a closer node) with the MEC (Multi-node edge computing) hosts being the edge nodes.[7]

2 Background

Cloud gaming is where the cloud is used to help run a video game, and stream it to a device, whether that be a tv, a computer, or even a phone. This allows many people who can't afford a gaming computer or a console to play games they'd normally miss out on due to not being able to buy the expensive console or PC. Cloud gaming has been an idea for a long time, with attempts being made as early as 2005 [1]. Edge gaming is the same concept, but with aspects of edge computing applied to it. Both edge computing and edge gaming work in the same general way. Instead of connecting to the central cloud, you connect to a much closer edge node that runs the game for you or relays it back to the cloud directly. This makes the latency lower due to not being at such a huge distance. Figure 1 shows the setup. Edge nodes essentially work like range extenders for the cloud, making it far easier to connect while it helps make the latency much lower. This helps avoid high latency and lag issues for the user, improving the experience. It has its own issue with energy and migrations that are currently being worked on, and are the biggest obstacle to its performance. Edge gaming has been implemented, but is not common yet.

3 The Comparison between Cloud and Edge

Cloud gaming currently has been implemented by several companies. Google Stadia, Xbox Cloud Gaming, PlayStation Now, Amazon Luna, and GeForce Now all are current examples of cloud gaming [8], but due to the long distances between the user and the cloud the latency is higher than what is optimal. Many of these services have latency regularly over 100 ms. More optimal numbers for gaming would be 60 ms or lower. Partially because of this, it hasn't been as widely adopted by people, but as it improves it can be assumed that it will grow in adoption and be more popular. The other issue is energy, which can become a bottleneck for the performance of the cloud. Edge gaming aims to solve the latency issues by putting edge nodes closer to the user, which reduces not only the latency but also the packet loss they might experience [3]. Edge gaming doesn't solve the energy issue though, and possibly takes even more due to the edge nodes needing power as well. Edge gaming also has one other thing that becomes a much bigger problem: migrations.

Migrations are when your device has to change what it is connected to. As the cloud only has the central cloud, migrations don't happen very frequently. It only happens when you move between the coverage of two clouds. With edge gaming however, it is a much more frequent problem due to migrations between edge nodes. Sometimes, an edge node can't handle all of the load being requested of it. In these cases, some jobs will need to migrate to another edge node, which causes latency spikes and increased packet loss. In certain structures of this concept, multiple edge nodes will be employed in a multi-tiered structure. One edge node would connect to another which could connect to another or the central cloud. If you refer back to Figure 1, you can see M1 nodes, which are a larger node closer to the central cloud. This is what edge gaming would look like. In this hierarchy, the provider can do some jobs on the current node, or relay it back to the cloud or another node [4]. This provides the benefit of being able to push it further back if needed, or merely doing it here without going through the trouble. The main drawback of this is you have to take time to relay the information back and forth which can take time and energy.

4 Energy Problems with Edge Gaming

One of the two biggest issues with edge gaming is the energy costs. The energy costs are just as bad, if not a bit worse than that of cloud gaming. One of the best fixes to this appears to be green energy. By hosting green energy on site, the edge nodes would have a free source of energy, as long as the edge networks provider can pay the upfront costs and any maintenance issues that may come up. Certain edge nodes can only access green energy as well, due to location. These nodes have another issue altogether though, which is since they are limited to green energy only, their capabilities

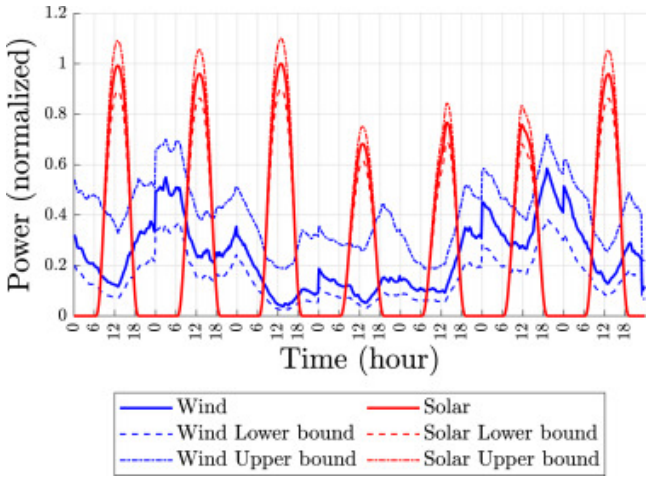

Figure 2. A graph of the green energy available in Belgium. It shows the general ebb and flow of power over days, showing that while green energy is good, there are times where it might bottleneck due to lack of energy.[7]

are limited by the energy they have access to [7]. This can hamper user experience or make it impossible to fill user demand effectively.

Due to the nature of different green energies, the power will always fluctuate. The main two green energies considered here are solar and wind, as those two are some of the easiest to access regardless of area. The above graph taken from Spinelli [7] shows exactly how energy fluctuates. But if we assume the majority of people are asleep at night, this shows that the peak energy production is while everyone is awake, making energy limitations come down to how many panels and wind turbines the edge network's provider can put up. As long as they have a big enough green energy farm the edge network provider can keep it powered for normal use with minimal issues, while having a relatively cheap cost, as the only payment is the initial for the most part. If there is demand when it's night and the wind is calm though, this could fall short of what's needed.

Another option is having access to both the traditional grid, as well as their own personal green energy. This allows them to use the normal grid during busy hours or times when their green energy sources are down at the time to fulfill service without problems. This also makes it so that the provider doesn't have to invest in so much green energy up front to try to keep themselves able to provide service [6]. Solar goes to 0 in the night, and wind fluctuates all the time, but through partial powering from the normal energy grid, it can cover this issue. But even then, there can be issues with bottle-necking due to power. This can limit how many jobs can be done, or will make some unable to be done effectively. Demand can grow to be too high, and will still take too much energy for an edge node to fulfill all the orders by itself. At times like this, jobs need to be carefully managed

to prevent a decline in user quality. This is where algorithms and migration come in.

5 Migration Issues

Migrations are the other issue that can harm user experience related to edge gaming. Due to the bottlenecks that the edge nodes can experience because of power constraints, or when the users move too far from the current node, the user's device might migrate between edge nodes. When a migration happens, there can be many issues with latency and packet loss. There is no way to stop migrations entirely, but through algorithms this issue can be dealt with so that they don't need to happen as often.

By taking in many variables, Spinelli et al. [7] proposes potential algorithms to fix this issue or at least make it far less prevalent. These algorithms are decently complex, but aim to take as many of the variables into account as they can. Variables such as job cost, available energy, proximity, projected users, and current users are all things that are taken into account while trying to figure out which node the device should be connected to.

Spinelli et al. describe 2 algorithms in depth. Figure 4 is a table of their variables. The first one they call the greedy algorithm, which doesn't worry about the future and only worries about what it has right now. Figure 3 is a screenshot of that algorithm. It takes a lot of inputs, which can be referenced in Figure 4, but I will go through them as well. The algorithm takes the network topology which is the physical arrangement of the nodes and connections. It also takes the jobs J with many parameters (such as power at the node or the delay of said job). It outputs a map of which nodes each job should be attached to. The algorithm uses a while loop to figure this out by running through parameters to figure out the optimal spot for user experience. The other algorithm is proposed by them, and Spinelli et al. call it *greening*. This is based on the greedy algorithm, but looks to improve upon it. This algorithm's concept is trying to keep the same amount of quality to the user experience while trying to be more efficient for speed and energy reasons. Some of the major

Algorithm 1 GREEDY Algorithm

1: **Input:** Network topology, $\mathcal{N}$, jobs $\mathcal{J}$ (with parameters $t_j, p_j, s_j, e_j, D_j \ \forall j \in \mathcal{J}$), $T_n, P_n, S_n, G_n, E_n \ \forall n \in \mathcal{N}$, $d_z \ \forall z \in \mathcal{Z}$

2: **Output:** Job-to-node placement map S
3: **Initialize:** $S = \emptyset$; $\mathcal{J}^{\emptyset} = J$; $S^{\emptyset} = \mathcal{J} \times \mathcal{N}$
4: **while** $\exists (j,n) \in S^{\emptyset}$ s.t. $S \cup (j,n)$ satisfies (2c) **do**
5: $(j^\star, n^\star) \leftarrow \arg\max_{(j,n) \in S^{\emptyset}} \Theta(S \cup (j,n))$
6: $S \leftarrow S \cup (j^\star, n^\star)$
7: $\mathcal{J}^{\emptyset} \leftarrow \mathcal{J}^{\emptyset} \setminus j^\star$
8: $S^{\emptyset} = \mathcal{J}^{\emptyset} \times \mathcal{N}$
9: **end while**

Figure 3. An algorithm that manages migrations.[7]

Table 1
Notation used in this work.

Notation	Meaning
C_j	Total Cost of job j
$C_j^{(b)}$	Energy cost (per slot) for job j
$C_j^{(d)}$	Deployment cost for job j
$C_j^{(m)}, C_j^{(p)}$	Migration and interruption costs for job j
$\mathcal{J}, J$	Set of jobs and its size
$\mathcal{N}, N$	Set of nodes (game servers), and its size
R_j	Revenue of job j
$\mathcal{T}$	Set of consecutive time slots
$\mathcal{Z}, Z$	Set of links and its size
T_n	Bandwidth of node n
t_j	Downlink throughput for job j
d_z	Delay incurred on link z
D_j	Maximum delay for job j
G_n, E_n	Green and total power at node n
e_j	Power required by job j
P_n	Computing power at node n
p_j	Computing power for job j
S_n	Memory capacity at node n
s_j	Memory required for job j
$w_{jz} = \{0,1\}$	(Variable) 1 if job j passes through link z
$x_{jn} = \{0,1\}$	(Variable) 1 if node n handles job j

Figure 4. A table of all the variables Spinelli et al.[7] use in their algorithms.[7]

differences are its focus on green energy and its dedicated migration function. Whereas the greedy algorithm places them in the most optimal spot initially, it tends to just leave them. The *greening* algorithm runs anytime there is an energy change in a job or a new job comes in and will move them over to a new node as needed. This makes it so that a job that suddenly spikes in power usage (i.e. a game session was on a menu and moves to actual gameplay) or the node suddenly drops in energy production (i.e. the wind stops blowing) it will move jobs if that is necessary to maintain quality.

As it ends up being approximately 80 lines of code, I won't be showing it here, but here is a description of the general way it works. It takes similar variables as the greedy algorithm in Figure 3, and outputs a node placement map and energy left. Below I'll provide the comparison between the algorithms.

Spinelli et al. compared 8 algorithms in their results, and while the differences were small they weren't insignificant (see Figure 5). They measured the algorithms on utility based on their own equation that takes into account the time taken and the cost. The experiment was ran as a simulation using data from an edge node. They took the data from a 24 hour period of playing games to use for the simulation. *Solver* is an algorithm that uses the Matlab intlinprog function with a 40 second timeout to make sure that it returns a result fast enough. Due to the nature of this though, it only can be

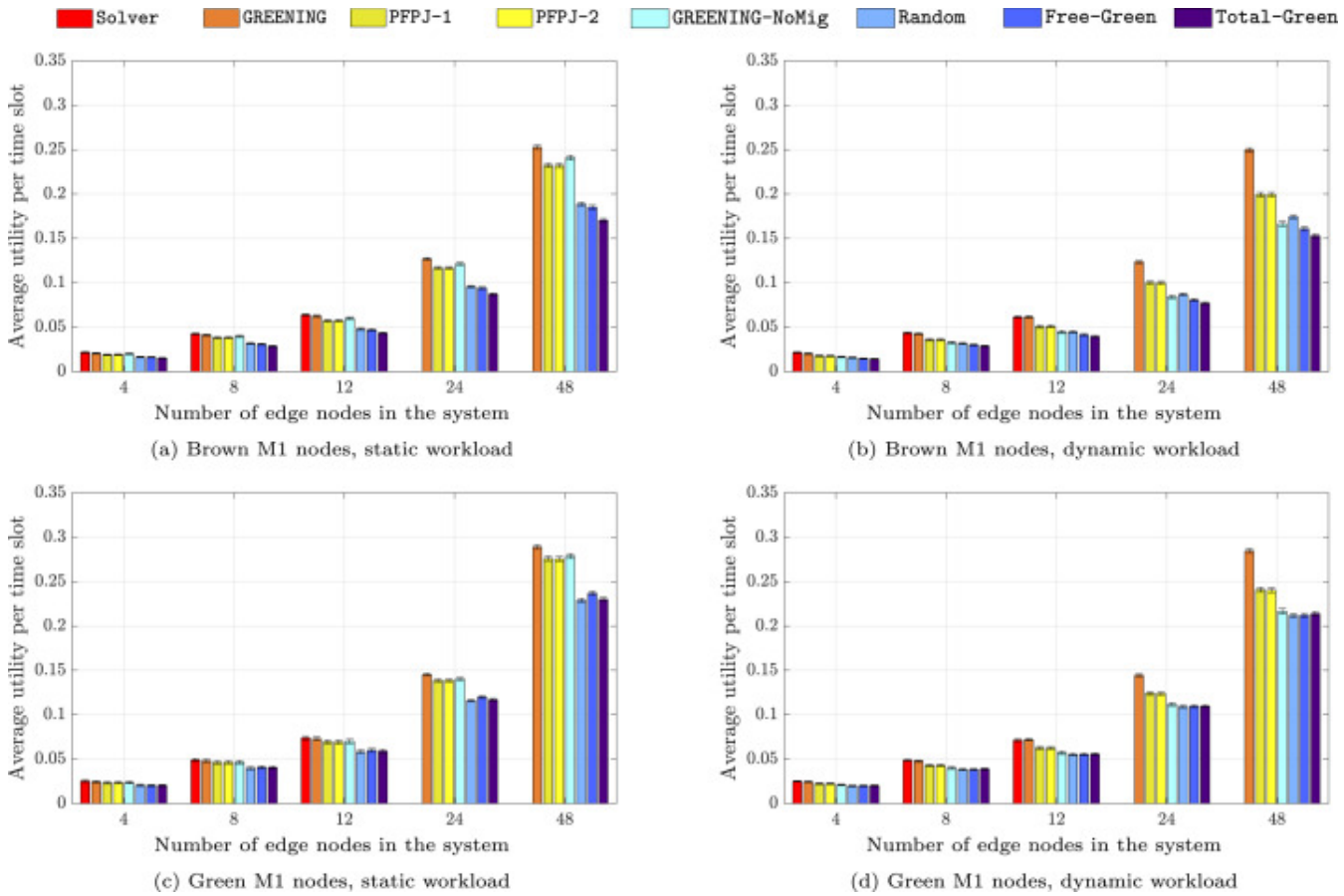

Figure 5. Results comparing many algorithms that manage migrations. It shows their algorithm, *greening*, does make a difference in performance.[7]

included in certain parts of the results. *Greening* is Spinelli et al.'s proposed algorithm. *PFPJ-1* is derived from *PFPJ-2* which is taken from Paulraj et al.[5]. *PFPJ-1* has been altered by Spinelli et al. to have a power constraint to match better with their algorithm. *Greening-NoMig* is the *greening* algorithm simplified with migrations disabled, so that it merely sorts on the current node only. *Random* performs probabilistic placement, with and equal chance for each node. *Free-Green* and *Total-Green* both place based on the amount of green energy available, but *Total-Green* does so independently of whether that energy is in use or not. The graph refers to brown and green, either static or dynamic. Brown means that it only takes from the standard electrical grid, while green means that it relies on green energy as much as it can (but pulls from the electrical grid until it matches the 2 kW standard set so that there was a baseline for measuring using the same overall energy for both tests). Static means it assumes an energy cost for the job and maintains that static energy cost. Dynamic means that it's adjusting the energy cost depending on what the user is doing at that current time. For example, a menu takes less energy than gameplay, and dynamic would adjust for this difference. The x-axis shows how many edge nodes were in the system and the y-axis shows the average utility per time slot. The more edge nodes in each system the better their *greening* algorithm did, as each node had higher utility than the others. The difference only increases the more nodes added to the edge network. Utility is also increased across the board with green energy, and this is due to many of these algorithms focusing on green energy as well as it improving costs through less costly energy. Over time, other algorithms could be made to further improve the utility which could help with both energy concerns and user experience.

Ma et al. is another paper that shows algorithms managing migrations, although a slightly different kind. Their list of variables is in Figure 6. These migrations involve the offloading of tasks and migrating them. Offloading these tasks to a higher node or the cloud can improve the quality by not making the edge node or the cloud work so hard alone. It can split the load of the job between the two, making it work faster. This migration also needs to be as fast as possible though, as if it isn't fast it could cause the same sorts of

Parameter	Definition
$\mathcal{T}$, τ_0	the set of time slots, the length of each time slot
$\mathcal{F}$, $\mathcal{B}$, $\mathcal{N}$	the set of FET, BET, tasks
λ_i^e, λ_i^t	$\in$ [0,1], means the weighting parameters of energy consumption costs and the cloud rental fees, or means the weighting parameters of latency
V	Lyapunov control parameter
$I_i(t)$, $O_i(t)$	the input and output size of the task of device i
$Y_i(t)$	the amount of computing resource required for the task of device i
$\pi_i(t)$	that is 1 if device i has a task to be offloaded, and 0 otherwise
$P_f^{Front}(t)$	the physical resources (in CPU cycles) of FES-EH f
$P_b^{Back}(t)$	the physical resources (in CPU cycles) of BES b
$D_{if}^{Fronttra}(t)$	the upload data rates from device i to its associated FES-EH f
$D_{if}^{Frontrec}(t)$	the download data rates from device i to its associated FES-EH f
$D_{ib}^{Backtra}(t)$	the upload data rates from device i to its associated BES b
$D_{ib}^{Backrec}(t)$	the download data rates from device i to its associated BES b
$D_i^{Cloudtra}(t)$	the upload data rates from device i to cloud server
$D_i^{Cloudrec}(t)$	the download data rates from device i to cloud server
$e_f^{Front}(t)$	the energy consumption in FES-EH f generated by receiving one unit of input data, calculating the data and transmitting the calculated result data back to device i
$e_b^{Back}(t)$	the energy consumption in BES b generated by receiving and calculating one unit of data, and transmitting the calculated result data back to device i
c^{cloud}	the resource rental fees (costs) for receiving one unit of input data, calculating the data and transmitting the result data back to device i
$p_f^{Front}(t)$	the electricity price of FES-EH f
$p_b^{Back}(t)$	the electricity price of BES b
α	the cost coefficient of using green energy
β	the cost coefficient of using grid power energy (fossil energy)
$Q_f(t)$	the energy level of the EH module at time slot t
$r_f(t)$	the amount of harvested green energy in FES-EH f at time slot t

Variable	Definition
$x_{ij}(t)$	the task offloading decision making at time slot t
$E_f^{green}(t)$	the consumed green energy in FES-EH f

Figure 6. A table of the variables used in Ma et al.'s algorithms.[4]

issues as the other type of migration. And when there could be 3 or more layers (with more than one edge node) this could be relayed back farther than just one layer, this needs to be managed by a fast algorithm too. Ma et al.'s paper has an example of this.

Ma's et al.'s algorithm can be seen in Figure 7. Ma. et al. shows two algorithms of varying length and thoroughness. The first one they show is *JTOES* which is in Figure 7. As you can see, it takes an input of the energy level, a Lyapunov control parameter (an optimization technique that needs this parameter), a variable representing whether a device has a task that needs to be offloaded or not, the electricity price of

Algorithm 1: On-line JTOES algorithm for solving $\mathcal{P}1$.

Input: $Q_f(0)$, V, r_f^{max}, $\pi_i(t)$, $P_f^{Front}(t)$, $P_b^{Back}(t)$;
Output: x_{ij}, $E_f^{green}(t)$;
1: **for** each time slot $t \in \{0, 1, 2, \ldots, T-1\}$ **do**
2: Obtain $x_{if}(t)$, $x_{ib}(t)$, x_{ic} and $E_f^{green}(t)$ by solving $\mathcal{P}2$.
3: $Q_f(t+1) = Q_f(t) - E_f^{green}(t) + r_f(t)$.
4: **end for**

Figure 7. One of the algorithms shown in Ma et al.[4]

the FES-EH (Front Edge Server), and the electricity price of the BES (Backend Edge Servers). It outputs whether the task should be offloaded to a further back node or the main cloud, and the consumed energy. This is a short, but more energy efficient way of managing offloading. Ma et al. have a second algorithm, that much like *greening* is quite a bit longer, so I won't include it here. It takes the same inputs as *JTOES*, and outputs things that *JTOES* uses (namely the things on the second line). They compare these to a few other algorithms, with this one outperforming them. This sort of algorithm is complex but is necessary to guarantee the user experience. [4] The difference between Ma et al.'s algorithms and Spinelli et al. is whereas Spinelli et al. is meant to manage jobs from one node to another, Ma et al.'s algorithm is only meant to decide whether something should be pushed back to a bigger node. Ma et al.'s don't move it from one node to an adjacent edge node, only a further back one.

6 Conclusion

Edge gaming is an innovation that will roll out more and more as time goes on, and while there are issues, people are already working on fixes. As it gets better and better, it is a safe assumption that it will grow in popularity, and as it does, it will solve many of cloud gaming's issues. It could change how gaming works, with it making less sense to buy a console or PC and just to stream it to your phone or television. The energy issues and migration issues could be a major issue if they continue, but as edge gaming starts getting implemented this research should be able to mitigate some of these issues. And as it is implemented, many more breakthroughs are sure to happen, through further attempts to streamline the process or improve the hardware. In the future, it is likely that edge gaming will become a major part of the gaming atmosphere, and will only continue to improve as time goes on.

References

[1] 2023. Cloud gaming. https://en.wikipedia.org/wiki/Cloud_gaming
[2] Adam Bankhurst. 2020. Three Billion People Worldwide Now Play Video Games, New Report Shows. ThreeBillionPeopleWorldwideNowPlayVideoGames,

NewReportShows [Online; accessed 14-April-2023].

[3] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. 2019. Computation Offloading Toward Edge Computing. *Proc. IEEE* 107, 8 (2019), 1584–1607. https://doi.org/10.1109/JPROC.2019.2922285

[4] Huirong Ma, Peng Huang, Zhi Zhou, Xiaoxi Zhang, and Xu Chen. 2022. GreenEdge: Joint Green Energy Scheduling and Dynamic Task Offloading in Multi-Tier Edge Computing Systems. *IEEE Transactions on Vehicular Technology* 71, 4 (2022), 4322–4335. https://doi.org/10.1109/TVT.2022.3147027

[5] Getzi Jeba Leelipushpam Paulraj, Sharmila Anand John Francis, J. Dinesh Peter, and Immanuel Johnraja Jebadurai. 2018. Resource-aware virtual machine migration in IoT cloud. *Future Generation Computer Systems* 85 (2018), 173–183. https://doi.org/10.1016/j.future.2018.03.024

[6] Daniela Renga and Michela Meo. 2019. Dimensioning Renewable Energy Systems to Power Mobile Networks. *IEEE Transactions on Green Communications and Networking* 3, 2 (2019), 366–380. https://doi.org/10.1109/TGCN.2019.2892200

[7] Francesco Spinelli, Antonio Bazco-Nogueras, and Vincenzo Mancuso. 2022. Edge Gaming: A Greening Perspective. *Computer Communications* 192 (2022), 89–105. https://doi.org/10.1016/j.comcom.2022.05.022

[8] Wikipedia. 2021. Category:Cloud gaming services — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Category:Cloud_gaming_services [Online; accessed 21-March-2023].