



Edge Gaming: An Overview

Matthew Spilman



Cloud gaming

An idea with its merits, allowing one to play a game through the cloud, making the need for a powerful pc or computer not needed. There are a few examples you may have heard of , such as...

Google Stadia, Xbox Cloud Gaming, GeForce Now, and PlayStation Now.

But cloud gaming has an issue with latency and packet loss, causing some players to have a disadvantage in multiplayer games, or even certain single player games.

But how can we fix the latency issues?





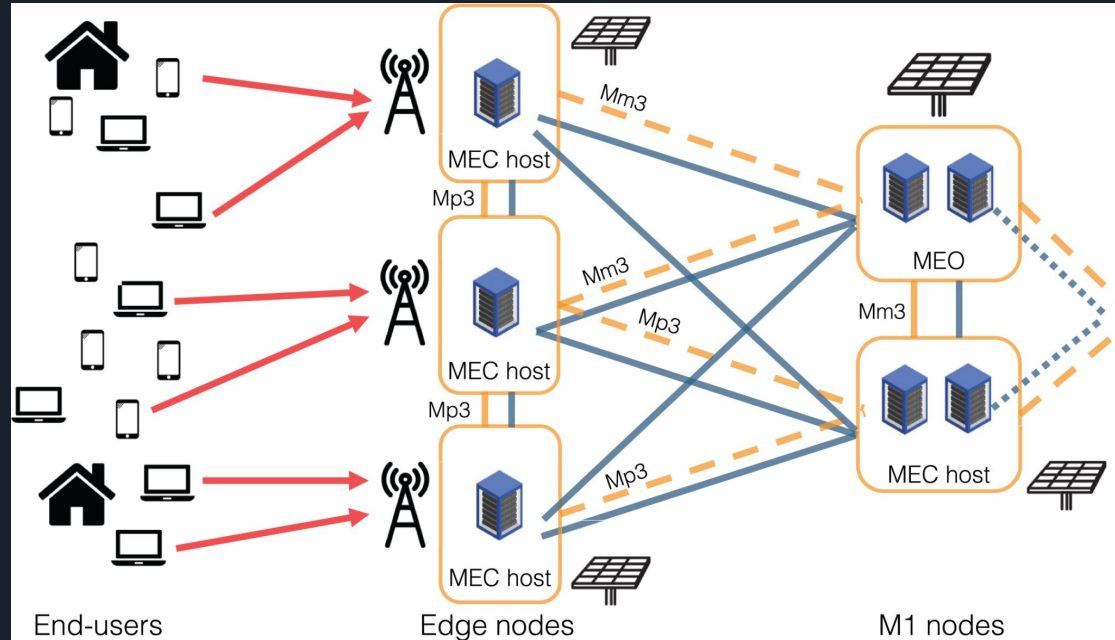
Outline

- What is Edge Gaming?
- Cloud vs Edge
 - What are the key improvements and drawbacks of edge gaming vs cloud gaming?
- Energy Issues
 - Both have energy issues, but what kind of solutions are there that could alleviate or fix those issues?
- Migrations
 - What are they? How can they be fixed?

What is Edge Gaming?

MEC: Multi-access Edge Computing
MEO: MEC Orchestrator
M1 Nodes: Larger than the edge nodes

- Edge gaming is taking aspects of edge computing and applying it to games.
- End users connect to edge nodes rather than the central cloud.





Cloud Gaming vs Edge Gaming

- Gaming requires constant updates, as user input can change any minute.
- Cloud gaming runs it on a central server, then streams it to your device, but the further from the central cloud the more issues you might have.
- Latency and packet loss can cause your inputs to be delayed
- Edge gaming puts nodes closer to the edge, helping limit the latency and packet loss



Cloud Gaming vs. Edge Gaming (Cont.)

When playing on your own PC comes down to your internet connection and the game server.

During cloud gaming, the server is connecting to the game server, and the connection there should be good. Your connection to the server is where latency comes from, and your distance from the server can affect this.

During edge gaming it is much the same as cloud, but distance is far less of an issue.



Energy Issues

- Cloud gaming has energy issues, they are equal or worse with edge gaming
- The best way to deal with this appears to be green energy.

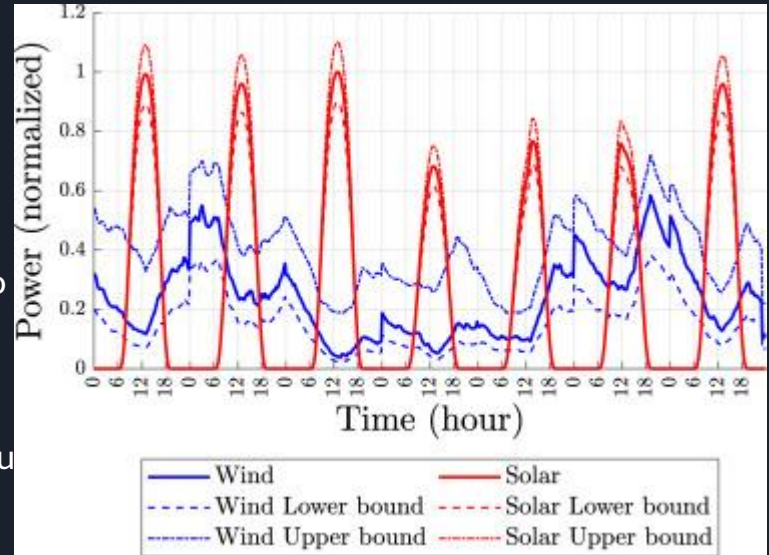
Energy issues cont.

This is a graph of wind and solar energy in Belgium.

It shows the general power being generated by them over multiple days.

You can see how the energy fluctuates, solar going to 0 at night, and wind fluctuating over time.

Using this we can see the issue that green energy would have with edge gaming, as it fluctuates a lot, but could work to help fix some of the issues.





Energy Issues (Cont.)

Green energy helps to alleviate some of the energy problems.

It is free after the initial setup, and can be a much needed boost to the nodes energy.

But even then, some nodes will get too many users, and might struggle to keep up. Some users will need to be moved to a different node in this case.



Migrations

Migrations need to happen to keep each users experience above a certain level. If you never migrated, one node could get too much traffic and be unable to service everyone, or you might be playing a game on the bus and get farther away from the node you were connected to.


When these sorts of things happen, your service needs to move to a new node, but if it's not fast and efficient, it can cause latency to spike and packet loss to occur. So they are managed by algorithms.

Migrations (Cont.)

- Migrations are important to keeping the quality of the service.
- Migrations are managed by algorithms like this.
- This particular algorithm is a greedy algorithm, which means it doesn't worry about the future, it makes the most optimal choice at this very second.

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:  $S = \emptyset$ ;  $\mathcal{J}^\theta = \mathcal{J}$ ;  $S^\theta = \mathcal{J} \times \mathcal{N}$   
4: while  $\exists(j, n) \in S^\theta$  s.t.  $S \cup (j, n)$  satisfies (2c) do  
5:    $(j^*, n^*) \leftarrow \arg \max_{(j, n) \in S^\theta} \Theta(S \cup (j, n))$   
6:    $S \leftarrow S \cup (j^*, n^*)$   
7:    $\mathcal{J}^\theta \leftarrow \mathcal{J}^\theta \setminus j^*$   
8:    $S^\theta = \mathcal{J}^\theta \times \mathcal{N}$   
9: end while
```



1: Input. Takes the Network topology and all the jobs (with the parameters)

Here is all the node information it takes

- bandwidth of the node
- computing power of the node
- memory of the node
- Green energy and total energy of the node
- all nodes that are an element of \mathcal{N} (the full node map)
- delay incurred on link z
- and all links that are part of the full map of links.

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:** $S = \emptyset$; $\mathcal{J}^\theta = \mathcal{J}$; $S^\theta = \mathcal{J} \times \mathcal{N}$

4: **while** $\exists(j, n) \in S^\theta$ s.t. $S \cup (j, n)$ satisfies (2c) **do**

5: $(j^*, n^*) \leftarrow \arg \max_{(j, n) \in S^\theta} \Theta(S \cup (j, n))$

6: $S \leftarrow S \cup (j^*, n^*)$

7: $\mathcal{J}^\theta \leftarrow \mathcal{J}^\theta \setminus j^*$

8: $S^\theta = \mathcal{J}^\theta \times \mathcal{N}$


9: **end while**

The parameters for the jobs

- The downlink bandwidth
- computing power required
- memory required
- Energy required
- maximum delay
- and all of the jobs that are an element of J (the full list of jobs)

Algorithm 1 GREEDY Algorithm

- 1: **Input:** Network topology, \mathcal{N} , jobs J (with parameters $t_j, p_j, s_j, e_j, D_j \forall j \in J, T_n, P_n, S_n, G_n, E_n \forall n \in \mathcal{N}, d_z \forall z \in \mathcal{Z}$)
- 2: **Output:** Job-to-node placement map S
- 3: **Initialize:** $S = \emptyset; \mathcal{J}^\theta = J; \mathcal{S}^\theta = J \times \mathcal{N}$
- 4: **while** $\exists(j, n) \in \mathcal{S}^\theta$ s.t. $S \cup (j, n)$ satisfies (2c) **do**
- 5: $(j^*, n^*) \leftarrow \arg \max_{(j, n) \in \mathcal{S}^\theta} \Theta(S \cup (j, n))$
- 6: $S \leftarrow S \cup (j^*, n^*)$
- 7: $\mathcal{J}^\theta \leftarrow \mathcal{J}^\theta \setminus j^*$
- 8: $\mathcal{S}^\theta = \mathcal{J}^\theta \times \mathcal{N}$
- 9: **end while**



2: Output. It outputs a placement map of where every job should go.

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:** $S = \emptyset$, $\mathcal{J}^\theta = \mathcal{J}$; $S^\theta = \mathcal{J} \times \mathcal{N}$

4: **while** $\exists(j, n) \in S^\theta$ s.t. $S \cup (j, n)$ satisfies (2c) **do**


5: $(j^*, n^*) \leftarrow \arg \max_{(j, n) \in S^\theta} \Theta(S \cup (j, n))$

6: $S \leftarrow S \cup (j^*, n^*)$

7: $\mathcal{J}^\theta \leftarrow \mathcal{J}^\theta \setminus j^*$

8: $S^\theta = \mathcal{J}^\theta \times \mathcal{N}$

9: **end while**



3: Initialization. It initializes a set of variables, circled on the right.

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

4: Start of the while loop. Essentially, while the current job and node are included in the overall list so that the union of S and (j,n) satisfy (2c) which makes sure that the job is following certain rules.

The rules that it must follow are:

- Each job can only be on one node
- The job's placement can't violate the servers capacity in regards to : downlink bandwidth, processing power, available energy, and memory.
- The delay on each job is acceptable

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:** $S = \emptyset$; ~~$\mathcal{J}^\theta = \mathcal{J}$; $S^\theta = \mathcal{J} \times \mathcal{N}$~~

4: **while** $\exists(j, n) \in S^\theta$ s.t. $S \cup (j, n)$ satisfies (2c) **do**

5: ~~$(j^*, n^*) \leftarrow \arg \max_{(j, n) \in S^\theta} \theta(S \cup (j, n))$~~

6: $S \leftarrow S \cup (j^*, n^*)$

7: $\mathcal{J}^\theta \leftarrow \mathcal{J}^\theta \setminus j^*$


8: $S^\theta = \mathcal{J}^\theta \times \mathcal{N}$

9: **end while**

5: (j^*, n^*) are assigned to the maximum argument of $\Theta(S \cup (j, n))$ which essentially is the theoretical requirements for $S \cup (j, n)$. It does this provided that (j, n) exist in S^\emptyset

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



6: S is reassigned to the union of S and (j^*, n^*)

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



7: The variable J^\emptyset is reassigned to $J^\emptyset \setminus j^*$ which removes j^* from J^\emptyset to not count it twice.

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



8: Sets S^\emptyset to $J^\emptyset \times N$ which is the cartesian product (which essentially is a map of all the elements of J^\emptyset paired with N)

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



9: ends the while loop

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



Migrations (cont.)

An algorithm meant to manage migrations are complicated, but are important to fixing the migration issue. Without this, it wouldn't be able to migrate efficiently, which would make the user's service suffer.



Study

The study was conducted by Spinelli et al. and included 8 algorithms. The study was a simulation done in a few minutes (other than the solver algorithm, which took days.) I'll go over each of the algorithms briefly here. They measured their results on the utility which they determined using an optimization algorithm that they developed.



Algorithms

- Greening: Greening is Spinelli et al.'s algorithm, based off of the greedy algorithm from before.
- Greening-NoMig: Greening-NoMig is the same algorithm, but with the dedicated migration feature turned off (which means that instead of finding the best node, it just merely denies it at this one)
- Solver is an algorithm using the uses the Matlab intlinprog function, and needed a timeout to avoid the experiment taking too long
- Random is merely a random placement with equal probability

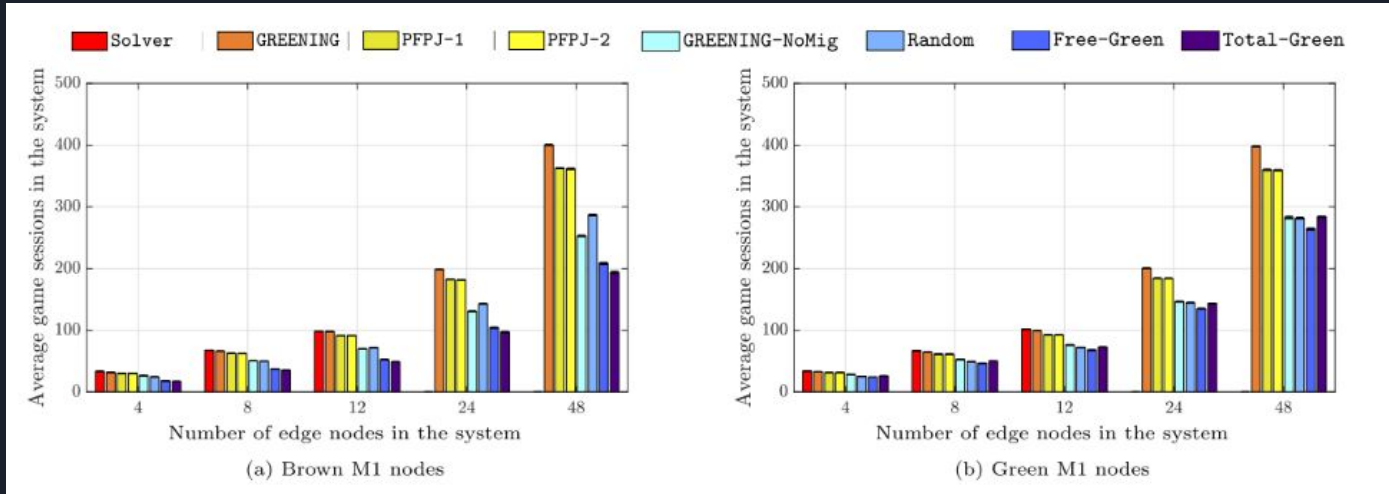


Algorithms (Cont.)

- PFPJ-2: PFPJ-2 is from another paper
- PFPJ-1: PFPJ-1 is the same algorithm but with an energy component to match their greening algorithm better.
- Free-Green places based on the amount of green energy available
- Total-Green places based on the total green energy at a node, regardless if it is available.

Results (Cont.)

This graph shows how many game sessions on average are in the system with each algorithm. Spinelli et al. shows that their greening algorithm can handle the most at once, but not by a huge margin.





Conclusion

Edge gaming isn't in practice currently, but as it rolls out in the future, it will fix the main issue of cloud gaming, allowing more types of games to be played. If it is to be rolled out, these issues need to be addressed, but people are already looking for solutions.



Sources

Images from:

[https://www.logo.wine/logo/Google Stadia](https://www.logo.wine/logo/Google_Stadia)

<https://seeklogo.com/vector-logo/457171/xbox-cloud-gaming>

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>^[1]