Reinforcement Learning Based Fair Edge-User Allocation for Delay-Sensitive Edge Computing Applications

Alaa Eddin Alchalabi

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
Doctorate in Philosophy (Ph.D.) degree
in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Cloud Gaming systems are among the most challenging networked-applications, since they deal with streaming high-quality and bulky video in real-time to players' devices. While all industry solutions today are centralized, we introduce an AI-assisted hybrid networking architecture that, in addition to the central cloud servers, also uses some players' computing resources as additional points of service. We describe the problem, its mathematical formulation, and potential solution strategy.

Edge computing is a promising paradigm that brings servers closer to users, leading to lower latencies and enabling latency-sensitive applications such as cloud gaming, virtual/augmented reality, telepresence, and telecollaboration. Due to the high number of possible edge servers and incoming user requests, the optimum choice of user-server matching has become a difficult challenge, especially in the 5G era where the network can offer very low latencies. In this thesis, we introduce the problem of fair server selection as not only complying with an application's latency threshold but also reducing the variance of the latency among users in the same session. Due to the dynamic and rapidly evolving nature of such an environment and the capacity limitation of the servers, we propose as solution a Reinforcement Learning method in the form of a Quadruple Q-Learning model with action suppression, Q-value normalization, and a reward function that minimizes the variance of the latency. Our evaluations in the context of a cloud gaming application show that, compared to a existing methods, our proposed method not only better meets the application's latency threshold but is also more fair with a reduction of up to 35% in the standard deviation of the latencies while using the geo-distance, and it shows improvements in fairness up to 18.7% compared to existing solutions using the RTT delay especially during resource scarcity. Additionally, the RL solution can act as a heuristic algorithm even when it is not fully trained.

While designing this solution, we also introduced action suppression, Quadruple Q-Learning, and normalization of the Q-values, leading to a more scalable and implementable RL system. We focus on algorithms for distributed applications and especially esports, but the principles we discuss apply to other domains and applications where fairness can be a crucial aspect to be optimized.

Acknowledgements

First and foremost, I would like to express my wholehearted gratitude to Prof. Shervin Shirmohammadi for the tremendous support during the past years. Without your guidance, motivation, patience, and persistent help, this thesis would not have been possible. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. You provided me with the tools that I needed to choose the right direction and successfully complete my thesis. It was an honor being your student during my Master's and PhD studies.

I want to express my appreciation to government of Ontario, the Faculty of Engineering, and the Scholarship Committee of the Office of the Vice-Provost for awarding me with the prestigious Ontario Trillium Scholarship (OTS) which was worth a minumum of $40,000 CAD per year for 4 years of my PhD program. I also want to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for supporting this research under grant number STPGP-50689.

I would like to acknowledge my colleagues from the Discover Lab and MCR lab at uOttawa for their wonderful collaboration. I would particularly like to single out Prof. Abdulmotaleb El Saddik. I want to thank you for all of the opportunities I was given to further my research, in addition, I want to express my appreciation to the energetic and the friendly atmosphere you always maintained at the lab, even during the most stressful moments. Thank you for making the past 4 years full of joy and fun. I'd like to give special thanks to Prof. Hussein Al Osman for always being there and for graciously agreeing to participate in my thesis committee. I also want to acknowledge my gratitude to my collegue, friend, and research partner, Shady Mohammed, who has been standing by my side since the first day I arrived in Canada with his wife, Ayşe Rumeysa Mohammed. Shady and Ayşe Rumeysa, I would not have made it this far without you.

I also want to thank my colleagues Karthigesu Vijayasuganthan and Sorin Stoian from Swarmio Media who provided expertise and insights that greatly assisted this research.

Dearest of all are my parents, Dr. Mamoun Alchalabi and Dr. Zainab Hadal, for their wise counsel and sympathetic ear, and my siblings, Dr. Lujain, Qutaiba, Elaf, Yaman and Mariam, who are behind each step I take and are my greatest source of strength. I want to acknowledge the endless love and support they have been giving through thick and thin.

Last but not least, I want to thank my friends, Rabea Alriffai, Fadi Saqbani, Maher Ghalyoun, Rami Ghamian, Emad Badawi, Mohamad Kannan Idris, Jawad Abou-zraiba, Haya Turki, who provided happy distractions to rest my mind outside of my research making the past 4 years in Canada the best years of my life so far. Thanks for making me feel at home, or more precisely, making Canada, my new home.

Dedication

I want to dedicate my Ph.D. thesis to my biggest supporter,
Bana Husseini . . .

To my martyred uncle, Aiman Alchalabi,
and grandfathers, Youssef Alchalabi and Khalil Hadal,
I know you would have been proud . . .

To all of my extended family members that I haven't met in years;
the ones in Syria and those scattered across the globe . . .

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Edge computing is expected to become the predominant platform for supporting many applications, including latency-sensitive applications such as cloud gaming, virtual/augmented reality, telepresence, and telecollaboration, because edge computing servers are physically closer to the customer than conventional cloud servers and can therefore offer lower network latencies that are within the acceptable thresholds of such applications. The increasing demand on the multi-billion dollars' industries that require lag-less performance, such as the games industry, is drawing researchers' attention to find solutions that are closer to the optimum. In order to satisfy the rising demand of users for better Qualify of Experience (QoE), the interest towards building alternative decentralized networks to minimize the network's delay is quite a need. For instance, in order to host a gaming tournament, the traditional way is to build servers geographically distributed where users, gamers, can directly connect to them if they are located in a given geo-zone. In this state, the measured distance between the server and the players should be reasonable in order to keep the delay in the acceptable range. The type of games that we are referring to here are event-based games where there's no overhead on the network and where all/most of the video processing is done on the end-users' machines. Off-zone users cannot connect to any of the servers because the delay can be above the accepted/tolerable threshold, and hence, the expected QoE will be low. For example, First Person Shooter games require a latency of no more than 80 ms [9], while in Third Person Perspective cloud games every 100 ms of latency results in a 25% decrease in player performance even though such type of games can generally tolerate end-to-end delays of up to 500 ms [29]. Indeed, failure to meet the delay threshold of a game is one of the main reasons game service providers lose subscribers [32–34]. Conventional cloud solutions currently used by such services offer general-purpose computing resources at competitive costs. Their server selection methods are typically concerned with minimizing the costs of

cooling and power, rather than minimizing the user's latency [28]. In order to satisfy the latency thresholds, providers can deploy edge servers in the service areas, reducing latency by bringing the computing server closer to the user [22].

Compared to the cloud, an Edge cloudlet tends to be more limited in terms of its capability, storage and communication resources, and can therefore only be deployed with limited services [95]. To make up for this limitation, typically more Edge cloudlets are deployed in the users geographic area, leading to a high number of possible individual edge servers that could fulfill a user application's latency threshold. This large number of edge servers is exacerbated in the 5G era, because 5G provides very low network latencies, so edge servers that before 5G were just outside a user's latency range are now within that range because of 5G. With such a high population of potential edge servers, selecting the optimum edge server becomes one of the main challenges. Wu et al in [95] argue that designing an algorithm for selecting edge servers is needed because selecting servers for service requests can be a critical problem, especially in case of failures. As we will see in chapter 3, today when a new user joins a system deployed in edge cloudlets, most existing server selection methods connect the user to an edge server that has enough capacity and the lowest latency to that user. But this causes two problems, as we will show in chapter 7: (1) users who join earlier will be privileged with lower latencies while users who join later will experience higher latencies, sometimes violating the latency threshold, and (2) users who are in the same session, for example players who are playing against/with each other in the same gaming session, will experience widely different latencies, leading to unfairness among participants.

1.2 Challenges and Research Problem

The ideal system will potentially and optimally connect an incoming user of an online multiuser application to the application's best cloud/edge server, such that not only the maximum delay threshold of the application is adhered to, but also the delay variations experienced by different users collaborating with each other are minimized, leading to a smoother, fairer, and higher quality group collaboration compared to existing systems. The applications that can benefit from it are those that have geographically distributed servers and enable geographically distributed users to collaborate with each other live; e.g., multiplayer cloud gaming, esports, telepresence, telecollaboration, etc.

Among these applications, esports is one of the most challenging, because of the highly interactive and engaging nature of games. Multiple geographically distributed gamers playing with each other must not only watch the game scene with utmost focus, but also react extremely fast to perform quick actions during the game. The esports platform must provide players with

high visual quality and low delay. While the other said applications such as telecollaboration can also benefit from low delay, they are less sensitive compared to esports. For example, during the COVID-19 pandemic, people used video-conferencing tools such as Zoom, Skype, etc. As such, many people are now aware of the importance of audiovisual quality and low delay in such applications. However, if in these applications the quality temporarily degrades or the delay becomes temporarily high, participants wait for a few moments and then repeat or resume their conversation when the situation improves. While this certainly inconveniences the users, for most cases it is adequate if it happens only occasionally. But in esports players cannot afford to "repeat" or "resume" while playing. A player affected by low quality or high delay may very well lose the game! The same thing can be said about tightly-synchronous applications [85] such as telesurgery or teleoperation, except that esports has a much higher number of simultaneous users, so esports is more challenging. Other applications with a large number of users, such as Netflix or live online concerts, are what we call in multimedia science "presentational", meaning the viewers simply watch and have limited interaction ("play", "pause", "rewind", and maybe text chat), so delays as large as a few seconds are tolerable in those applications. Esport on the other hand is "conversational", meaning each user is a significant source of live input into the game; therefore esports has strict end-to-end delay thresholds, sometimes as low as 80 msec as in First Person Shooter games [9]. As such, we can see that esports is the one of the most challenging of all multiuser applications.

1.2.1 Esports Background

There were 2.5 billion active gamers worldwide in 2019, leading to a global market of US$152.1 Billion [94]. A share of this belongs to esports, which refers to organized multiplayer computer game tournaments, usually played by professional players and/or teams, and watched by a large number of spectators. An esports platform typically provides support for (1) local and/or geographically distributed players to play with each other, and (2) geographically distributed viewers to watch the game. The size of the viewership can be in the millions, with the current record belonging to the 2019 "League of Legends" World Championship at more than 100 million viewers, with a peak of 44 million concurrent viewers [91]. Esports is now easily the largest and most popular market of the entertainment sector [38], and its popularity has increased even more during the COVID-19 pandemic [93]. In the aforementioned League of Legends championships, teams from 13 regions around the world competed, and the matches were broadcast in 16 different languages, with the winning team taking home $834,375 in prize money [91]. This also shows the vast geographic distribution that an esports platform must be able to support live.

CG, the youngest entry into the gaming market, surpassed US$1 billion in 2018 and is projected to grow fast to US$8 billion by 2025 [14]. CG refers to adopting the cloud computing

paradigm to offer Game as a Service. In CG, the game events are captured from the players' devices and transmitted to the cloud, which processes those events, runs the game logic, renders the game scene, and streams the resulting high-quality scene in the form of video to the players. This is different from conventional online gaming, in which the server, after running the game logic, sends the state of the game to all players, and the player devices will then render the game scene and display it. Examples of CG are Sony PlayStation Now, Nvidia GeForce Now, Google Stadia, and the upcoming Amazon Tempo.

Whether it uses CG or conventional online gaming, an esports platform has one major challenge: delay. Players' events are captured and sent to a server/cloud, which then broadcasts either the game state or the video of the rendered scene to the players. All of this has to be done fast enough as to not adversely affect the player's performance and Quality of Experience (QoE). It has been shown that every 100 msec of latency in CG results in about a 25% decrease in player performance for third person perspective games which can generally tolerate end-to-end delays of up to 500 msec [30]. As such, ensuring an esports system's response delay, which consists of network delay + processing delay + playout delay, does not violate the game genre's delay threshold is one of the main challenges, and requires a well-organized resource management system to satisfy both the service provider's and players' requirements [20]. This is even more challenging for multiplayer games, because the real-time, conversational, and highly interactive nature of multiplayer games make them more even sensitive to delay; therefore much research have been done to address this challenge [8, 10, 25].

Usually, the playout delay is assumed to be negligible and does not have a significant impact on players' experience [21]. Also, processing delay depends on the available processing power in the cloud and can be improved by using faster/more CPU, GPU, and memory. It is therefore the network delay which has the most adverse effect in esports systems. To mitigate it, normally a large number of game servers are allocated in geographically distributed datacenters in the cloud or as edge servers. Hence, the server selection strategy becomes crucial. Considering that each game depending on its genre and pace has different delay requirements [31, 60, 76], assigning players to servers becomes a non-trivial challenge [42].

1.3 Research Approach

In this thesis, we propose the idea of AI-assisted hybrid networking for cloud gaming [6] to allow more users to connect without compromising the QoE is to introduce the concept of Edge Nodes (EN). EN is a user's computer that acts as a server in the network and connect other Users (U) to each other. The computing power could be, but is not limited to, PCs, smart cars, tablets, or smart phones. EN nodes are also always connected to at least one of the main servers or

Delegated Edge Nodes (DEN, see chapter 4) to save the state of the game. In case of any failure, the main server will have a recovery point of the latest state, and the users will be assigned to another EN to resume the gaming session.

Also, we also propose a fair server selection method for edge cloudlets using Artificial Intelligence (AI), specifically Reinforcement Learning (RL). We chose AI because the system is highly dynamic due to users joining and leaving at will and due to changes in the servers bandwidth and latency capacities. RL is especially well-suited for highly changing environments due to its notion of user-designed rewards. Building on the idea of AI-assisted hybrid networking for cloud gaming, here we propose a concrete solution to the server selection problem and formulate it as an RL problem. Inspired by works which use RL in networks [16], fast Q-learning with a uniformly bounded variance and large discount factors [43], multi q-table q-learning [63] and local normalization [65], we propose RL models that aim to minimize the variance of latency in user-server matching for edge cloudlets. We evaluate our model using actual data collected from a cloud gaming application, and we show that compared to existing methods which simply choose an available server with the lowest latency to the user, our method not only better meets the latency threshold, but also leads to more fairness; i.e., lower latency variation among users in the same session.

1.4 Broader Impact

To the best of our knowledge, fair matching problems haven't been dealt with as reinforcement learning problems, and in fact, with the attention drawn towards fairness in AI, designing reward functions in RL would be an ideal technique for fair matching problems. We presented 4 different reward functions in order to optimize variance in accordance to our notion of fairness (having a lower variance in matching distance). The application of gaming systems in edge cloud networks was one example of the applications that can be impacted by this methodology, and since the concept of edge computing is recently expanding, it has the potential to redefine how networks services are delivered and it would potentially optimize the computation power to users' end.

The proposed methodology and models are not only tailored for edge cloud server selection problems. The same methodology can be applied to a wide range of similar problems such as point-to-point matching, cardinality, or connectivity constrains problems. A plethora of these problems were highlighted in [27], such as, matching students to schools, ad slots to advertisers, search results to ranked positions, and with their corresponding constrains, such as, students' school choice, ad relevance to users, and the click through rate of the search result. The questionable scalability of RL algorithms can be easily addressed with the use if approximation functions instead of using tabular RL. The use of Neural Network is a perfect solution to handle signifi-

cantly large state and action spaces in RL problems, and that is quite common with the recent advances in deep reinforcement techniques.

Technologically, our work will impact mostly cloud gaming providers such as Google Stadia, Sony PlayStation Now, Nvidia GeForce Now, Amazon Tempo, and Facebook Gaming, although providers of any multiuser real-time collaborative application such as Zoom, WebX, or Microsoft Teams can also benefit. Our work enables these providers to optimize their offerings with a higher quality of user experience than what is possible today: better latency compliance and lower latency variation among users in the same online session. Economically, our work can contribute to the market expansion of such services because it will provide users with lower latencies and a fairer opportunity to participate, which is synonymous with more customers because failure to meet the latency threshold of games is one of the main reasons for subscriber turnover. The technical contributions are twofold: we propose an RL approach for cloud/edge server selection considering the variance of latency, and not just latency as in existing methods, for fairer user experience. We also introduce action suppression, Quadruple Q-Learning (QQL), and new Q-value normalization techniques in RL.

1.5 Intended Contribution

To the best of our knowledge, this is the first work that intends to make the following contributions:

- We propose the idea of AI-assisted hybrid networking for cloud gaming [6] to allow more users to connect without compromising the QoE introducing the concept of Edge Nodes;
- We consider the standard deviation of latency and its change, and not just latency itself as in existing methods, to provide a fairer experience to users;
- We design an RL method to solve the multivariate multi-constraint fair edge server selection problem. As part of the design, we also introduce the following novelties in RL:
  - We introduce Action Suppression in RL, which will reduce the practical difficulties in using RL in large action spaces;
  - We introduce Quadruple Q-Learning (QQL), allowing the system to choose the best action from among 4 different Q-Learning models;
  - We introduce normalization of the Q-values, allowing the system to fairly compare each model's Q-table values that have different scales.

1.6 Research Publications

1.6.1 Journals

- **Alaa Eddin Alchalabi**, Shervin Shirmohammadi, Shady Mohammed, Sorin Stoian, and Karthigesu Vijayasuganthan, "Fair Server Selection in Edge Computing with Q-Value-Normalized Action-Suppressed Quadruple Q-Learning", IEEE Transactions on Artificial Intelligence, August 16, 2021, 9 pages. DOI: https://doi.org/10.1109/TAI.2021.3105087.

- **Alaa Eddin Alchalabi**, Shervin Shirmohammadi, Sorin Stoian, and Karthigesu Vijayasuganthan, "An Edge-User Allocation Solution for Cloud Gaming: Achieving Fairness with Reinforcement Learning", Submitted to the IEEE Internet Computing, 9 pages, August 2021.

- Shady A. Mohammed, Shervin Shirmohammadi, **Alaa Eddin Alchalabi**, "Measurement Prediction of Network Delay with Deep Learning: From Lab to the Real World", Submitted to the IEEE Instrumentation and Measurement Magazine, 6 pages, September 2021.

- Mohammad Alja'Afreh, Hikmat Adhami, **Alaa Eddin Alchalabi**, Mohamed Hoda, Abdulmotaleb El Saddik, "Toward Integrating Software Defined Networks with the Internet of Things: A review", Cluster Computing, 16 pages, Accepted: August 26, 2021.

1.6.2 Conferences

- **Alaa Eddin Alchalabi**, Shervin Shirmohammadi, Shady Mohammed, Qnetwork: AI-Assisted Networking for Hybrid Cloud Gaming. In the 2019 IEEE International Symposium on Measurements Networking (M&N), pages 1–6, Catania, Italy, July 2019.

1.6.3 Patents

- Karthigesu Vijayasuganthan, Sorin Stoian, Shervin Shirmohammadi, Shady Mohammed, **Alaa Eddin Alchalabi**, "Methods and Apparatus for Network Delay and Distance Estimation, Computing Resource Selection, and Related Techniques," U.S. Patent Number US11063881B1, July 13, 2021, Available: https://patents.google.com/patent/US11063881B1/

1.6.4 Datasets

- **Alaa Eddin Alchalabi**, Shervin Shirmohammadi, "CGCSDD: Cloud Gaming Client-Server Delay Dataset", IEEE Dataport, August 3, 2021, DOI: `https://dx.doi.org/10.21227/jr75-0215`.

1.6.5 Technology Transfer

- Transferring the invention of Delay Sensitive Server Selection System (DS4[1]) to Swarmio Media Inc. after signing an exclusive licensing agreement with the University of Ottawa's Technology Transfer Office. Integrating DS4 into Swarmio Matrix™ which runs in parallel with Swarmio Hive™ [2] creating the AI based Latency-Optimized Edge Computing (LEC™) technology [3].Through Swarmio Hive™ & Matrix™, Swarmio Media is disrupting the rapidly scaling eSports ecosystem with both their customer based and B2B telco platforms.
- Contributing to the deployment of arQ™ [4] and QNetwork™[5].

1.7 Thesis Organization

The road map for the rest of this thesis is outlined below:

Chapter 2– Background

The chapter presents background information on cloud computing; it provides an extensive background of the existing paradigms of network computing as well as a very detailed background on Reinforcement Learning techniques.

[1]This work was financially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant number STPGP-506890, by Swarmio Media.

[2]Swarmio Hive™ is an esports platform-as-a-service (PaaS) that helps partners such as Telecoms, Esports Organizations and Enterprises monetize their esports communities. Swarmio Matrix™ is a globally distributed edge computing platform for the gaming industry. It uses AI based latency optimization technology to provide game server hosting solutions for video game developers and esports organizers.

[3]More information about the product can be found at: `https://swarmio.media/`.

[4]arQ™ is a blockchain-powered edge computing platform that makes it fast & simple to develop & deploy new latency optimized edge applications for IoT, Industry 4.0, AR/VR, 4K video delivery and Smart City initiatives.

[5]More information about QNetwork can be found at: `https://www.qnetwork.ai/`.

Chapter 3– Related Works

The chapter presents an extensive review of the existing works in the areas of server selection, gaming systems, edge cloud networks, edge user allocation, RL with Variance, Fairness, and Action Suppression.

Chapter 4– Problem Definition

The chapter presents the proposed formulation of the AI-assisted hybrid networking for cloud gaming and the optimal approach in detail in addition to the optimization formulation. This chapter reuses material from papers [6, 7] with permission.

Chapter 5– Proposed Methodology

The chapter presents the proposed AI solution of the problem with the RL Algorithm in detail and explains its components. We present the RL method to solve the multivariate multi-constraint fair edge server selection problem introducing Action Suppression in RL for large action spaces, Quadruple Q-Learning (QQL) as an ensemble method, and normalization of the Q-values. This chapter reuses material from papers [5, 7] with permission.

Chapter 6– Cloud Gaming Client-Server Delay Dataset (CGCSDD)

The chapter presents our collection methodology and the dataset of client-server Round Trip Time (RTT) network delay measurements of an actual cloud gaming tournament run on the infrastructure of the cloud gaming company Swarmio Media. This chapter reuses material from [4] with permission.

Chapter 7– Performance Evaluation

The chapter presents a simulated case study with preliminary results. In particular, the experiment results show a comparison of the proposed method and other existing anchor methods with respect to the fairness models. This chapter reuses material from papers [5, 7] with permission.

Chapter 8– Conclusion & Future Work

The chapter discusses our future research plan on further improving the current proposed method. In this chapter we explain how the new notion of fairness is affecting the selection in comparison

to state-of-the-art practices. Also, we talk about 2 different sets of experiments using single sessions and grouped sessions in a simulated environment. After all, we list the remaining steps that should be conducted toward completing my research work.

Chapter 2

Background

2.1 Introduction

In this thesis, we are introducing the idea of an AI-powered edge cloud for latency-sensitive real-time applications, and we are simulating gaming systems as an example to those application that will be use this system as a Platform-as-a-Service (PaaS). This concept utilizes technical knowledge from different ares, such as: edge cloud networks, cloud gaming, server selection, edge-user allocation, reinforcement learning, and fairness in AI. Specifically, when we are proposing a fair server selection algorithm for edge cloud gaming, we are integrating a lot of those concepts in one product in order to obtain the best QoE for gaming sessions. To give some background to this thesis, in this chapter, we will be explaining the technical topics and the related concepts we have tackled, so we will lay the ground on some of the definitions and the terminology used throughout thesis. Namely, we will be introducing the concepts and the terminology related to Cloud computing paradigms as well as all its related types and deployment architectures, and then we will talk about the concept of cloud gaming. Followed by we will introduce the various computing paradigms that was invented to tackle the drawbacks of cloud computing such as edge and fog computing. After that we will introduce a general knowledge about AI algorithms, and we will specifically go into reinforcement learning (RL) in detail along with all the recent advancements in the field in addition to the notion of fairness in AI.

2.2 Cloud computing

Cloud computing is a service that offers the users to utilise an unlimited computing resource over the internet at any time. The delivery of this service is on-demand for which customers pay for their amount usage only, and it is flexible so that users can utilize elastic services increasing and

decreasing the resources to match their desired application. Cloud computing offers the use of computing resources, network, storage and memory as-a-service where the customers' usage is metered and measured, and the data-intensive computation is offered over Wide Area Networks (WAN) which can be accessed from anywhere. The cloud uses shared resource pooling such that the resources are pooled to serve multiple customers using a multi-tenant model combining both physical and virtual resources. Those resources can be dynamically reassigned in order to obtain a rapid provisioning of resources based on the customers' demand. The cloud host is expected to have an ample amount of storage, memory and fast processing computing units offering quick data access, and they are also responsible of the cloud management ensuring the continuity of the service and its availability at all times.

2.2.1 Cloud types

The cloud services can come in different categories (layered responsibilities of cloud-based delivery) as shown in figure 2.1, and we will be discussing the three types of cloud computing which are:

- Infrastructure as a Service (IaaS): such as Amazon AWS, MS Azure, OpenStack, DigitalOcean, RackSpace, Google Compute Engine (GCE)
- Platform as a Service (PaaS): such as Google App Engine, Azure Kubernetes Service, Azure Functions, etc.
- Software as a service (SaaS), or Cloud applications : such as Office 365, OneDrive, Google drive, Dropbox, MS Teams, Zoom, etc.

Infrastructure as a Service (IaaS)

In IaaS, the cloud service provider is only responsible for managing the maintaining the required resources such as the availability of the servers, network, virtualization, storage, and memory. Customers will have the access through an API or a dashboard where they can rent and request the infrastructure. Customers are responsible for deploying the operating systems, middleware, apps while the cloud provider only maintains the hardware and the availability of the service.

Platform as a Service (PaaS)

In PaaS, the cloud provider not only provides the hardware but also offers a software-platform for users' application and data. This cloud platform is includes the operating system, middleware, and runtime software that developers relies on for application development and management.

Software as a service (SaaS)

In SaaS, the cloud provider is responsible for delivering the software to users with all the management and the maintenance related to that application. An example to SaaS are most web applications and mobile apps which allows users to use them at anytime. Cloud providers will be responsible not only for the delivery of this service but also maintaining the software updates and fixes. It offers more convenient user experience because it does not require the software to be installed on the users' machines allowing more collaboration and ease of use.

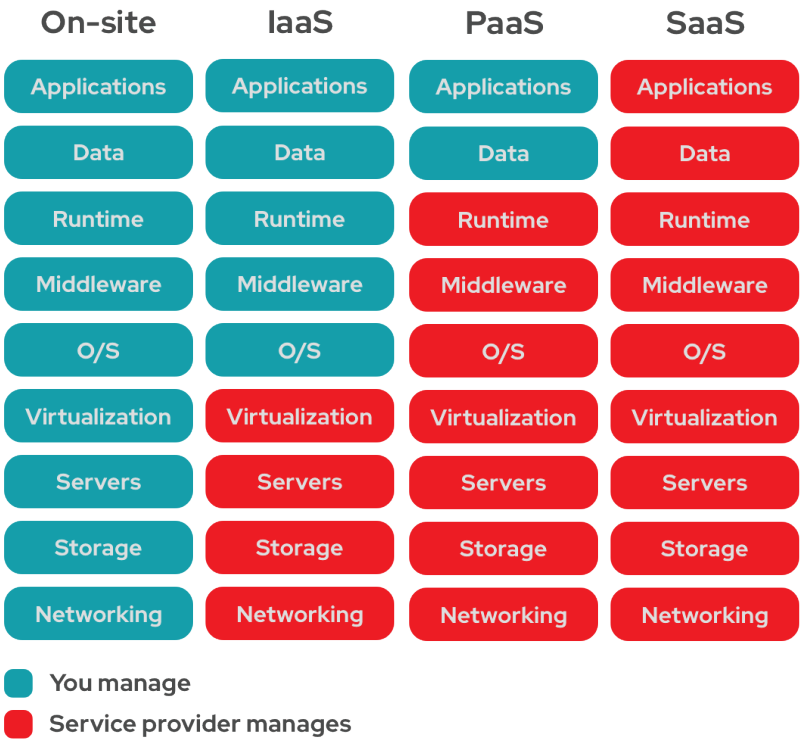


Figure 2.1: On-site, IaaS, PaaS, and SaaS[1]

2.2.2 Cloud Computing Deployment Methods

Cloud customers can choose from different cloud deployment methods and architectures based on the nature of their requirements and need. There are four different cloud deployment methods:

[1]Modified from source `https://www.redhat.com/en/topics/cloud-computing/what-is-iaas`

Public cloud

The public cloud is a cloud service that is offered by the provider over the public internet, and the service can either be free or paid based on the usage limit and the demand. As such, users can only pay for their usage of resources such as the computing power, storage, or network bandwidth.

The public cloud has an advantage over private clouds for companies because it cut costs of maintaining, managing, and purchasing the hardware infrastructure. Additionally, public cloud has a scalability advantage which allows the platform to expand and scale largely.

Private cloud

The private cloud (sometimes refereed to as internal or corporate cloud) is a cloud service that is offered mainly by companies either over the public internet or a private network. It gives businesses the advantage of being self-maintained, internally scalable, and more controllablity and customizablilty. Private clouds enhance security features due to the sensitivity of the data and the internal operations. The company's tech department is solely responsible for the management and maintenance because no other providers have the access to the cloud.

Hybrid cloud

The hybrid cloud is a cloud service that consists of an orchestration of the resources from 2 or more of public clouds, private clouds, and/or on-site infrastructures. The architecture of hybrid clouds makes them very flexible and agile which is essential for growing digital businesses. However, connecting the multiple instances of the hybrid clouds could face compatibility issues between them. Therefore, a hypervisor and a layer of virtualization is usually used in order to abstract the computing, storage and memory resources.

Community cloud

Community cloud is a cloud service that is shared among different organizations and companies that have similar computing concerns, have similar interests, and/or belongs to the same community. This type of cloud is common between inter-governments agreements, ventures, research organizations, and others. The concept is to decentralize the ownership of the infrastructure by multiple organizations which will allow those different organizations to collaborate on joint projects and applications that belongs to the community without relying on a cloud vendor. This deployment solution reduces the cost pressure and integration complexities as well as it covers the security concerns among the participating organizations.

2.2.3 Cloud Gaming

Cloud gaming is a type of video gaming or online gaming that is hosted on remote servers while users interact with a video stream of the game coming from the server. Cloud gaming is sometimes refereed to as gaming-as-a-service or gaming on-demand. In the case of cloud gaming, the game is run on a remote cloud and when users use any input, that input is sent to the cloud, and the interaction is sent back to the user in a shape of a compressed frames of a video stream. This allows users to play computationally expensive games at low computation devices such as mobile devices.

As explained in section 1.2.1, esports ecosystems are expected to be at an increase demand. In order to scale that up, cloud gaming can address some of the scalability and agility complications. Some companies like Google and Microsoft tried to join the race and enter the gaming competition by launching projects like Stadia and xCloud. However, cloud gaming platforms are still new business models that could not prove its practicality for esports due to their higher latency. In order to tackle that, edge computing and fog computing can be prospective solutions to the gaming industry.

2.3 Edge, Fog, and Transparent Computing

The cloud computing architecture allows users to utilize a computing resources ubiquitously which has been regarded as the second generation of network computing. However, the use of a centralized architecture fails to satisfy the service demands of ultra-low delay applications in the Interner of Things (IoT) era. Cloud-native applications may require more authentication time due to the WAN high latency which is impractical for applications that requires low or ultra-low latency such as gaming applications. The second issue is that the traffic capacity of WAN is going to be challenged with the dramatic increase of data generated by devices at the edge of the network. The drawbacks of the cloud paradigm is due to the fact that the architecture was developed to improve the efficiency and the availability of resources in a remote and centralized computing way. As such, there is a larger demand emerged for ultra-low latency, higher-bandwidth, context-aware, and geographically-sparse processing and computing paradigms. In order to address the drawbacks of the cloud, other network computing paradigms was introduced by the industry and academia to offer computation in proximity of end-users such as transparent computing, fog computing, mobile edge computing, and cloudlet. Those paradigms were reviewed in recent survey papers such as [82], and they can be suitable for delay-sensitive as well as decentralized applications (dApps).

Transparent Computing (TC)

In transparent computing the software is decoupled from the hardware as a form of persuasive computing, such that it separates the hardware storage and execution of applications away from the software, which include operating systems. It logically considers and integrates the distributed devices across the network as one system, and the system will assign services according to the capabilities of the device and network status. Those client devices are enabled to fetch services and tasks on -demand from the server side and locally process them in a block-streaming fashion (where only essential parts of the application is fetched instead of the whole software). Such a paradigm will consequently improve the delay of those services as well as the energy consumption significantly.

Mobile Edge Computing (MEC)

In mobile edge computing, currently referred to as multi-access edge computing or edge computing for short, edge servers are deployed in proximity of users usually by telecommunication companies. Edge servers will not only alleviate the computing load over the core network, but also can improve the QoE of users by processing the services at the network edge reducing the latency and involving context-awareness. Along with virtualization and Software-Defined Networks (SDNs), edge computing is expected to be the leading technology in the upcoming 5G era.

Fog Computing

Similar to edge computing, fog computing is a novel architecture that offers data processing on the network edge that support temporal and spatial awareness in addition to mobility and wireless access support. Initially proposed for the context of IoT, fog computing comprises of n-tier architecture that utilises all the computing power of the network devices along the data path in order to serve the end devices. Ideally, this paradigm is used for the instantaneous performance improvement because of the interplay between fog servers (responsible for reducing the service delay) and cloud servers (responsible for high-computation of data analysis).

Cloudlet

Cloudlets refer to small datacenters that are placed closer to users such as in a classroom of or a coffee shop. The motivation behind this paradigm was to improve the performance of the mobile applications that are delay and jitter sensitive. The architecture consists of 3 tiers, namely, cloud,

Pradaigm	Virtualization	Computing location	Research focus
Cloud	VM	Central Cloud and datacenter	Workflow and VM management
TC	MetaOS	End devices and close	Cross-platform and service on-demand devices
MEC	VM and containers	Micro and macro base stations and close devices	Computation offloading and caching
Fog	VM and containers	Network devices along routing path	Computation offloading and caching
Cloudlet	VM	Close cloudlets, e.g., classroom	Computation offloading and VM management

Table 2.1: Similarities and Differences Comparison of Different Computing Paradigm [82]

cloudlets, and mobile users where cloudlets enable servers to respond is a timely fashion and also ensuring the data privacy before releasing the data to the cloud servers.

Similarities and Differences

The aforementioned computing paradigms were introduced to reduce the cost of infrastructure management, reducing the end-to-end delay, and to improve the overall QoE and scaling networks more efficiently. Despite the paradigms having a common goal, they differ in some aspects, as explained in Table 2.1.

2.4 Reinforcement Learning (RL)

In this subsection, we will lay the ground on all the terminologies and the background knowledge used later in this thesis as long as few of the common RL algorithms. It should be mentioned that the definitions and the formulations follows Sutton and Barto in [88].

2.4.1 RL Background

Symbols

Here are the most common symbols used in RL[2].

a action

A advantage

α learning rate, step size

b bandit, baseline estimate

c cost, context (= state)

d difference error

Δ difference

D replay buffer

E eligibility trace

ϵ exploration rate ($0 \sim 1$)

Φ state transition function

g gain

γ discount factor

∇ gradient

H horizon

J expected return

L loss / regret

λ trace decay ($0 \sim 1$)

O conditional observation probabilities

Ω observations

Pr transition distribution

π policy ($state \rightarrow action$)

q quality

r reward

R return

s state

t time

T conditional transition probabilities between states

τ trajectory

θ parameter weight

v value

w weight

y target / prediction

$_*$ optimal

$'$ next / derivative

$\hat{}$ estimation

$||\,||$ vector norm

[2]Referenced from the RL cheat-sheet by Kiara Grouwstra `https://github.com/KiaraGrouwstra/Reinforcement-Learning-Cheat-Sheet` which is a modified version of Francesco Zuppichini's cheat-sheet.

Agent-Environment Interaction

Reinforcement learning (RL) refers to goal-oriented algorithms, which learn how to attain a complex objective/goal or maximize along a dimension over multiple steps. The RL algorithms are penalized when they make the wrong decisions and rewarded when they make the right ones, and that's how they learn.

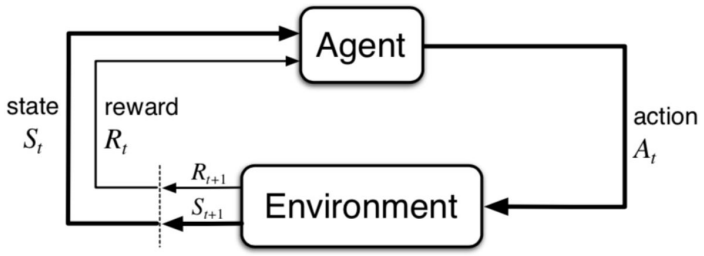


Figure 2.2: RL Agent-Environment Interface

Agents: is the subject that takes actions and interacts with the environment. **Environment** is the world through which the agent moves. The environment takes the agent's action as input and returns the agent's reward and next state as output. **Action:** is the set of all possible moves the agent can make. An action is almost self-explanatory, but it should be noted that agents choose among a list of possible actions. **State:** is a concrete/immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant dynamics.

The agent receives the environment's state or a representation of the environment's state at each time step t, where $S_t \in S$. The RL agent takes an action $A_t \in A(s)$ for that given state. Then as a consequent to the taken action, the agent receives a reward $R_{t+1} \in R \in \mathbb{R}$.

There are general challenges when it comes to RL problems. (1) First of all choosing the right representation of the problem and the states of the environment is a challenge. Since the agents depends on those representations in selecting actions, the choice needs to be not only optimized, but also inclusive. (2) One other challenge is the generalization problem where RL agents sometimes fail to generalize on states and cases it was not trained on. (3) The Temporal Credit Assignment (CAP) is also considered one of the challenges of such systems. In some RL environments where it depends on a delayed reward, it is hard to assign which action lead to that outcome. An action that could have a contribution to a higher cumulative reward should have more credit than an action that has a lower contribution to the final reward. (4) That can also be tackled by finding the optimal balance between the Exploitation and the Exploration parameters where it be optimized to know if the RL agent prefers short term outcomes or long term outcomes by taking random actions at specific probabilities.

Policy

A policy is a strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward. Mathematically, it is a mapping from a state to an action, and it is denoted as $\pi_t(s|a)$ which the probability to select an action $A_t = a$, given a current state, $S_t = s$. Note that the sum of probabilities of all outbound actions from s is equal to 1:

$$\sum_a \pi(s|a) = 1 \tag{2.1}$$

Reward

The reward is the signal that the agent receives from the environment after performing a particular action. The gain (G_t) or total reward is expressed as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_3 + ... = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{2.2}$$

Where γ is the discount factor and H is reward's horizon which can be infinite.

After that we can calculate the regret (L) accordingly which is the difference between the optimal gain and the gain under a policy.

$$L = G^* - G_\pi \tag{2.3}$$

ϵ-Greedy

It is a technique used to overcome the problem of exploration vs. exploitation. It allows the agent to take a random action at a probability $p = \epsilon$, and pick actions according to a policy π at a probability $p = 1 - \epsilon$. This will allow the agent to explore the environment more in case it did not get lucky with the first trials.

Markov Decision Process (MDP)

MDP is mathematical representation of the environment as a complex decision making process where it involves: all the possible actions, all states, a reward, and a transition function T which returns the next state S' if we start at a state S and take the action a. The end-goal of the MDP

is to represent the environment and find the optimal policy that will maximize the long-term expected reward or the discounted sum of rewards. When the system's state depends only on it's previous state and the last action, we say it is **Markovian**. MDP is a 5-tuple (S, A, P, R, γ) where:

$$\begin{aligned}
&\text{Finite set of states:} \\
&\quad s \in S \\
&\text{Finite set of actions:} \\
&\quad a \in A \\
&\text{State transition probabilities:} \\
&\quad p(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} \\
&\text{Expected reward for state-action-next-state:} \\
&\quad r(s',s,a) = \mathbb{E}\Big[R_{t+1}|S_{t+1} = s', S_t = s, A_t = a\Big]
\end{aligned} \tag{2.4}$$

Value Function

The expected long-term return with discount, as opposed to the short-term reward r. $V_\pi(s)$ is defined as the expected long-term return of the current state under policy π. It also an indication of how good to be in a certain state under a given policy.

$$V_\pi(s) = \mathbb{E}\Big[G_t|S_t = s\Big] = \mathbb{E}\Big[\sum_{k=0}^{H} \gamma^k r_{t+k+1}|S_t = s\Big] \tag{2.5}$$

Action-Value Function or Q-function

Q-value is similar to Value, except that it takes an extra parameter, the current action a. $Q_\pi(s,a)$ refers to the long-term return of the current state s, taking action a under policy π. Q-tables contains the q values which maps state-action pairs to rewards.

$$q_\pi(s,a) = \mathbb{E}_\pi\Big[G_t|S_t = s, A_t = a\Big] = \mathbb{E}\Big[\sum_{k=0}^{H} \gamma^k r_{t+k+1}|S_t = s, A_t = a\Big] \tag{2.6}$$

We can realize that the value function is equal to the sum of the q-functions of actions, multiplied by the policy probability of selecting each action. We can also derive the optimal value function $V_*(s)$ in terms of the optimal Q-function $q_*(s,a)$ as follows:

$$V_*(s) = \max_{a \in A(s)} q_{\pi*}(s,a) \tag{2.7}$$

Bellman Equation

The Bellman equation is one of the central elements of RL algorithms. It decomposes the value function into two main parts: the immediate reward in addition to the discounted sum of future values. It simplifies the composition of the value function such that instead of summing over multiple time steps, it is possible to find the optimal solution of any problem by breaking it down to sub-problems recursively.

Expanding both the value and the Q-function we can redefine the value function in terms of the probability of the state-action pair (a, s) resulting is a state s' as follows:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma V_\pi(s') \Big] \tag{2.8}$$

The above equation also expresses the stochasticity of the environment with the sum over the policy probabilities. Similarly, we can do the same for the Q function:

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a) \Big[r + \gamma V_\pi(s') \Big] \tag{2.9}$$

The Bellman equation of optimally

Since the goal of the RL algorithms is to find the optimal policy (or get a policy as close as the optimal policy), Bellman equation helped in redefining the definition of both the optimal value function and the optimal action-state function. For the optimal value function, it will be calculating by finding the action a that outcomes the maximum expected discounted sum of rewards for the next state s'.

$$V_*(s) = \max_a \sum_{s'} p(s', r|s, a) \Big[r + \gamma V_*(s') \Big] \tag{2.10}$$

Similarly, by substitution and using the same approach, the optimal Q-function will be recursively calculated by the following:

$$Q_*(s, a) = \sum_{s'} p(s', r|s, a) \Big[r + \gamma \max_{a'} Q_*(s', a') \Big] \tag{2.11}$$

2.4.2 Common RL Algorithms: Q-learning, DQN

Q-learning

Q-learning is a Temporal Difference (TD) method which allows the agent to learn directly from the experience model-free which means it does not need the environment's dynamics. It substitutes the expected discounted sum rewards G_t with an estimation as the following:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \tag{2.12}$$

The generic implementation of the algorithm is shown below.

Algorithm 1 Q Learning

Initialise $Q(s, a)$ arbitrarily and $Q(terminal - state, \cdot) = 0$
for each episode $\in$ episodes **do**
    while s is not terminal **do**
        Choose a from s using policy derived from Q (e.g., ϵ-greedy)
        Take action a, observer r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$

Deep Q Learning (DQL)

Deep Q Learning, which is created by *DeepMind*, substitutes the $Q-$function with a neural network called *Q-network* to estimate the Q-values. It also collects a batch of some observation in an experience history (or a replay memory) to use them later to train the *Q-network*.

$$\begin{aligned} y &= r_t + \gamma \max_{a \in A} Q(\Phi(s_{t+1}, a); \theta_i^-) \\ \underbrace{\nabla_i J(\theta_i)}_{\text{loss function}} &= \mathbb{E}_{(s,a,r,s') \sim \underbrace{U(D)}_{\text{memory}}} [(\underbrace{y}_{\text{target}} - \underbrace{Q(s, a; \theta_i)}_{\text{prediction}})^2] \end{aligned} \tag{2.13}$$

Where θ are the weights of the *Q-network* and $U(D)$ is the experience replay memory.

The next Algorithm explains DQL in detail.

Algorithm 2 Deep Q Learning

Initialize replay memory D with capacity N
Initialize $Q(s, a)$ arbitrarily
for each episode $\in$ episodes **do**
    Pick a from s by policy from Q (e.g. ϵ-greedy)
    while s is not terminal **do**
        Take action a, observer r, s'
        Store transition (s, a, r, s') in D
        Sample random transitions from D
        $y_i \leftarrow \begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{otherwise} \end{cases}$
        Perform gradient descent step on $(y_j - Q(s_j, a_j; \Theta))^2$
        $s \leftarrow s'$

Chapter 3

Related Works

3.1 Introduction

There is a significant body of academic work that addresses similar insights in an intersection of similar areas. Those areas include the recent advancements in edge cloud networks, cloud gaming, server selection, Edge-User Allocation, and fairness in AI. In this thesis, we are bringing them all together. We are proposing a fair server selection algorithm for edge cloud gaming. We focus on algorithms that will match players and edge servers fairly increasing the average QoE, **but it should be noted that the principles we discuss apply to other domains, and also to any other matching algorithm carrying out one-to-one or one-to-many matching tasks and rules**.

To give substance to this thesis, we will be discussing the different types of approaches for server selection and edge-user matching, as well as the fair RL algorithms that our approach is inspired by. In this chapter, the focus is to address the works incrementally. We will start by discussing general server selection works, then we will discuss gaming related selections and matching criteria. Then, we will talk about the recent works in edge cloud networks followed by recent edge-user allocation works which lack a fairness notion. After that we will discuss similar RL fairness works along with safe RL algorithms and action space reduction. Throughout the discussion we will try to compare our proposed approach in fair server selection/matching in edge cloud gaming to the other works.

3.2 Server Selection

Outside of cloud gaming, there was a decent amount of interest in the server selection problem. Hu et al. [59] formulated a server selection problem for interactive video streaming as a geomet-

ric Euclidean K-median optimization problem to reduce end-to-end delay. Their optimization was purely mathematical formulated as a euclidean k-median problem to reduce end-to-end delay based on network coordinate system, not on delay measurements. Goel in [52] proposed a client-assisted CDN server selection by using a DNS-proxy in the client side that shares load-balancing functionality with CDNs and chooses the CDN with the lowest delay from among resolved CDN servers based on the network performance. Authors in [81] presented a Model Predictive Control-based algorithm to determine routing optimization and server-selection in intelligent SDN-based CDN architecture, optimizing users' response time (delay) and bandwidth satisfaction shift. The work in [102] uses lightweight methods that can determine network topology and select the server for multi-party video conferencing, minimizing the mean end-to-end delay between clients using delay-based clustering. Finally, Wu et al. in [95] combined Genetic algorithm and simulated annealing algorithms for service selection in Mobile Edge Computing to reduce the time delay.

3.3 Server Selection for Gaming Systems

Server selection for gaming systems has a decent share of the conducted research. However, existing server selection methods however consider only delay, and not its variance. For example, Web et al. [92] proposed methods for players to servers allocation under a specified constraint of minimizing overall delay for all players to mirrored servers. The goal of optimal client-to-mirror assignment is optimizing the minimum average client-to-mirror delay considering joining and leaving clients. Also, in [48] Farlow and Trahan proposed player-server matching algorithms to maximize the number of players supported by the system by moving players to different servers during gameplay as new players join or leave, optimizing the overall delay: 1) GDA-1: joins player to region server, then players are ranked in decreasing QoS order. For each of these players, the region server attempts to find a new contact server with available capacity. 2) BUMP-ON-LEAVE (B-O-L): attempts to assign a player to the first server that satisfies QoS. If not, they connect to the lowest delay, then they wait until there's a capacity to be moved to. 3) BUMP-ON-JOIN-OR-LEAVE (B-O-J-O-L): similar to B-O-L just keeps 2 options per players while iterating and moves player to second option if a player is waiting. 4) REASSIGNMENT THRESHOLD (R-T): attempts to reduce the number of times the "bumping" portion of the algorithm needs to be run setting a threshold for the ratio of players without QoS to the total number of players.

Others have tackled the problem of cost in cloud gaming and incorporated the pricing in their selection of cloud providers. In [41, 42], Deng et al. formulated multiplayer cloud gaming problem with the objective of minimizing the rental and bandwidth cost charged by cloud providers. Another work [44] formulates the optimal virtual machine placement problem and proposes an

algorithm capturing the trade-off between response delay and cost. Similarly, [58] tackles the challenge of optimizing cloud gaming experience and introduces an optimization problem to maximize the cloud gaming provider's total profit while achieving an acceptable QoE. In [89], Tian et al. address the same problem with the consideration of adaptive streaming technology, therefore, the work aims to optimize the overall cost by adaptive adjusting the selection of servers, virtual machine allocation, and video bitrate for each user request.

3.4 Edge Cloud Networks

With the recent advances in cloud services and IoT, edge computing has been getting increasingly more and more attention in the last decade [84]. Edge computing has many privileges such as speed, security, cost saving, reliability and scalability, which permits the prompt delivery of new applications and services compatible with the future 5G Internet. In terms of Cisco Global Cloud Index, 75% of people and devices produced data will be stored, processed, analyzed, and synthesized close to or at the edge of the network by 2021 [61]. As a novel paradigm, edge computing also introduces a few new challenges for app vendors and a new research venues in allocation-like problems, for example, edge server-user allocation [55,66–69], edge data caching [74, 96], the problem of joint service placement and request scheduling in edge clouds with shareable and non-shareable resources [57], edge server placement [37], and edge application deployment [24, 40].

Additionally, the recent research trends have addressed a variety of research topics in the area of Edge cloud networks. For example, Cache optimization with the end goal of having shorter delays for edge users was one of the main interests which aimed at reducing amount of network data traffic in edge cloud architectures [22, 23]. A paper by Min et. al [23] considers the challenge of reducing network traffic by evaluating date to be offloaded at the edge cloud intelligence, and then reducing the amount of data transmission. Also, authors in [28] propose a hybrid architecture that leverages the advantages provided by the edge-only and datacenter-only in order to increase the number of users using a voting-base strategy as well as optimize the latency due to the use of edge nodes closer to user-end. Other paper [83] proposes a hybrid method for minimizing service delay in a scenario with two cloudlet servers, controlling Processing Delay through Virtual Machines migration and improving Transmission Delay with Transmission Power Control through a mathematical mode. Connecting back to server selection, Wu et al. in [95] combined a genetic algorithm and simulated annealing algorithms for service selection (selecting the services to be deployed on edge servers) in Mobile Edge Computing systems. Finally, one of our previous work [6] we introduces an AI-assisted hybrid networking architecture for edge cloud gaming where players' computing resources acts as points of service

(edge devices) in addition to central cloud servers.

A short paper by Dimopoulos et al. [45] formulated a distributed recourse allocator that preserves fairness and satisfies low latency workloads in a two-level multi-cloudlet environment that is offloading compatible. Another paper have approached the same task differently. Since edge resources are limited in comparison to the cloud, the algorithms of allocation and scheduling assumes the availability of the resources. Some work [75] incorporated a different notion of fairness in edge computing where they challenge the assumption of the availability of all needed resources. In order to guarantee fairness among the clients (Cloudservers offloading jobs), the consider the task of accounting for priorities of the jobs. They present four schduling techniques, namely: first come first serve, client fair, job-priority fair, and hybrid of the last two.

3.5 Edge User Allocation (EUA)

Recently, the edge user allocation (EUA) problem has been receiving increasingly more attention. In their recent paper, [67] a first attempt to formulate the EUA problem was made by Lai et al. The EUA problem was modeled as a variable sized vector bin packing problem, and then they proposed an approach to maximize the number of allocated users in parallel with minimizing the number of edge servers allocated. Then, the paper investigated a more complicated problem by introducing user satisfaction to the formula characterized by the Quality of Experience (QoE). The main goal was to maximize the overall QoE of all users by allocating edge servers with suitable QoS levels [68]. On the other hand, Peng et al. had another apporach in modeling the EUA problem by considering the mobile edge computing environment and modeling the EUA problem as a revolvable process. Their approach exploits the mobility of users, so they proposed a greedy algorithm based on the mobility of the edge users to find a proper edge user-server allocation [80]. Figure 3.1 shows an example of the EUA problem in a wireless network setting.

However, most of the existing research on EUA has focused on minimizing the delay between users and edge devices. A very similar and interesting work by Z. Xu et al. [97] has introduced distance-awareness in EUA systems. Their problem definition leverages the complexity of wireless transmission in a real-world scenarios such that they have incorporated the correlation between the user-edge distance and the signal strength (affecting the data rate) in their formulation. This has been neglected in previous research because it was assumed that all users will receive the same data rate of they were in the specific coverage area of each edge base station. Since the distance will affect signal strength and eventually affecting the data rate, users being in a far position inside the coverage area would result in unsatisfactory data rates lowering users' overall QoE. The paper took into account the distance between users and edge servers affecting users' data rates and QoE. Then, they utilize that knowledge to propose a design

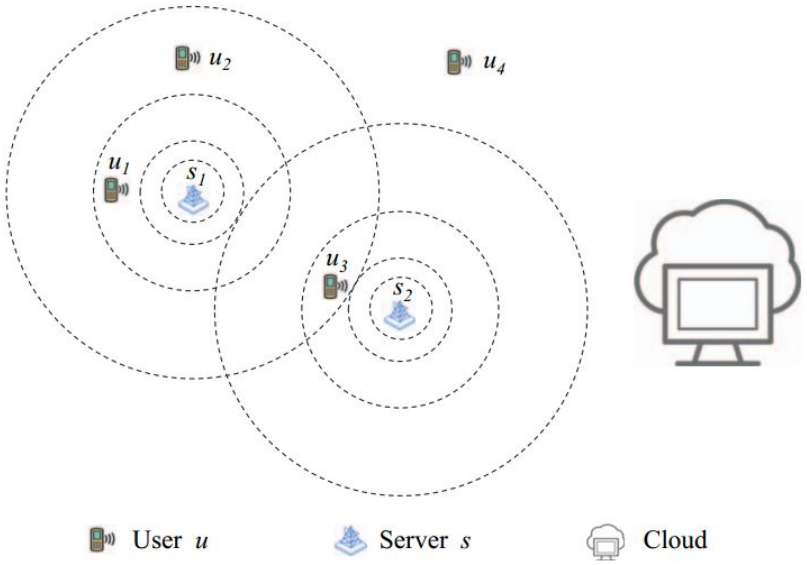


Figure 3.1: An example of distance-aware EUA problem [97]. Users are denoted u_i and edge servers are denoted by s_i

of an optimal approach, named DEUA-O, based on Integer Linear Programming (ILP), and to propose a heuristic approach, named DEUA-H, for finding sub-optimal solutions in large-scale scenarios of the EUA problem. DEUA-H has three main steps: 1) obtaining the set of available edge servers that have enough computing resources to accommodate the user; 2) calculating the ratio of the remaining computing resources to the respective distance for each edge server, and finding the edge server which has the highest ratio; 3) selecting the maximum QoS level the edge server can provide, and allocate the user to that edge server. In their experiments, they compared the performance of DEUA-O and DEUA-H with one baseline approach and two state-of-the-art approaches: 1) Random: This approach randomly allocates users to available edge servers. 2) VSVBP [67] : This approach solves the EUA problem as a variable sized vector bin packing problem, with the aims to maximize the number of allocated users and to minimize the number of edge servers needed. 3) DQoS [68]: This approach solves the EUA problem with the consideration of the correlation between users' QoE and required QoS. It aims to maximize the total QoE of all the users, the same as DEUA-O and DEUA-H.

Despite the interesting results and the intuitions behind the build models, those approaches work perfectly well for wireless transmission networks and their intended applications, but they fail to generalize for solving more complicated applications, such as gaming for example. The algorithms use either an optimization formula to find the optimum solutions or use a few heuristics to find a sub-optimal one to cut the computation cost, however, the algorithms should be

re-run again once a new node gets introduced. As such, they can be costly to retrain every once in a while for a highly changing environment such as cloud gaming. Another reason why these algorithms can not fit our problem is that they lack the notion of fairness in the allocation process such that the first users to be processed will be allocated to the best fit edge server. This phenomenon could create a difference in group gaming sessions where each of the users should have almost a similar QoS and eventually QoE metrics to all others. This could have a huge impact on the QoE once the number of options get exponentially high. In the next section, we will go through that more in depth.

3.6 RL with Variance, Fairness, and Action Suppression

3.6.1 Fairness in AI

As mentioned earlier in the introduction, cloud gaming not only has certain latency thresholds depending on the game's genre, but must also minimize the variance of delay among players in the same gaming session to provide fairness. It is well known that players with higher latencies perform worse than those with lower latencies [101] and could lose the advantage in the game. Therefore, we are incorporating the notion of fairness in the allocation problem.

Recently, there is lot of attention has been drawn recently to design fair algorithms. In [15], the authors discuss a variety of approaches to find fair resource allocation from the literature such as max-min fairness, lexicographic ordering, proportional fairness in addition to some fairness measures. Another example is the interesting fairness in AI work is the paper [13] which studies the problem of finding low-cost fair algorithms for clustering allowing users to define the maximum over- and minimum under-representation of any group in any cluster. Fairness has been one of the research trends in AI, and it captured a variety of topics such as: fairness in classic and contextual bandits where in a pool of applicants, a worse applicant is never favored over a better one [62], fair algorithms for clustering [13, 17, 26, 47, 64], fairness in ranking algorithms with diversity constrains [19, 98], removing Disparate Impact from selection processes [49, 99], fairness in multiwinner voting rules which are used to select a small representative subset of or items from a larger set [18], fairness in decision making algorithms such as deciding whether defendants awaiting trial are too dangerous to be released back into the community. [36,50] to name a few. Our work falls under designing fair matching algorithms, and as previously mentioned, we concentrate on the notion of sub-optimal matching of 2 groups, reducing the variance of their distance function. Our current notion of fairness is to reduce the variance of the latency and, as will be explained further, we use geo-distance between users and edge servers as an indication of this latency.

3.6.2 Safe RL Algorithms

There are RL algorithms that minimize the variance of the reward in addition to the expected reward. Those techniques can commonly be found in Safe Reinforcement Learning, which can be described as the process of learning policies that maximize the rewards in problems whose system performance, reasonableness, and safety constraints are critical during the learning or deployment processes, as explained by Garcia et al. [51], who surveyed and categorized two different approaches of Safe RL: classic discounted finite and infinite horizon with a safety factor, and incorporation of external knowledge or the guidance of a risk metric. In these approaches, the risk concept is related to the inherent stochasticity of the environment. While these work well for their intended application, Safe RL algorithms mainly maximize the final long-term rewards which could avoid the occurrences of larger rewards along the way and do not necessarily circumvent the rare occurrences of large negative outcomes. As such, they are not suitable for our application because in our problem we are trying to reduce the variance of rewards in every step an action is taken which does not necessarily lead to an optimized variance in the long-term reward.

3.6.3 Action-Space Reduction in RL Algorithms

Reducing the action space, which we also use in our RL method, has been present in current research too. The work in [100] proposes the Action Elimination Network (AEN): a system that utilises 2 Neural Networks: one containing an approximation of the Q-function and the other concurrently learning to eliminate actions. This would help overcoming some difficulties in large action spaces in situations such as text generation. The AEN outputs a linear contextual bandit model that eliminates actions with high probability. Inspired by this approach, we have adopted the concept but modified the technique for a tabular case while also emphasizing fairness for matching problems. In contrast to the Action Elimination Network which uses neural networks, we have used a linear vector to indicate if the actions are available, as will be shown in the next chapter. Our vector handles the availability of the actions with the ability to either have the vector values fixed or learned.

3.7 Conclusion

In this chapter, we overviewed different works in the areas on server selection for cloud gaming, edge-user allocation, safe RL, and other fair selection algorithms. We have also discussed what the researchers have proposed and introduced to the problem and how they reasoned their

heuristics. The majority of the studies in fair server selection are defining the problem as an optimization problem with their specific constrains. This makes the solution very impractical for highly changeable environments such as the prospective edge cloud networks. Therefore, a learning approach is needed to address the scalability issue. Other RL methods can differ in their end-goal. Moreover, such algorithms would need some modifications in order to fit the problem and the nature of the environment while none have proposed a generic algorithm for fair matching and formulated it as an RL problem.

Our research goal is to fill these gaps. We have simulated the edge server-user problem as an RL problem and incorporated the notion of fairness in the selection process. We have introduced a few models with a dynamic reward function and ensemble models to perform a live fair action-taking in the environment leveraging all the shortcomings of the literature. Additionally, in the design process of the algorithm the reproduciblity factor was taken into consideration such that the same generic approach can be applied to any similar problems.

We will discuss the details of our proposed algorithm in the rest of this thesis.

Chapter 4

Problem Definition

4.1 Motivation & Inspired Works

Gaming systems are among the most challenging networked-applications, since they deal with streaming high-quality and bulky video in real-time to players' devices (such as in cloud gaming) or they deal with a very latency-sensitive packets' processing, interactive, and engaging nature games (such as the case of online gaming and esports). While all industry solutions today are centralized, in this chapter we introduce an AI-assisted hybrid networking architecture that, in addition to the central servers, also uses some computing resources as additional points of service (network edge devices). We describe the problem, its mathematical formulation, and potential solution strategy. It should be mentioned that the problem formulation explained in this chapter is derived from the business requirements of Swarmio Media's esports ecosystem.

A recently launched project named New Kind of Networking (NKN) [1] has similar objectives to our proposed solution. NKN is a P2P network that is powered by a new public Blockchain that includes a new network protocol and ecosystem. NKN motivates users to share the unused internet connection bandwidth using economic incentives. The network offers an open environment for developers to build decentralized applications (DApps) on a universally accessible, low cost, and robust infrastructure. The similarity lies on the fact that both network requires the users to share the network connections and join to construct a Blockchain of available devices. While registering in QNetwork and registering in NKN are voluntarily, QNetwork differs in utilizing the bandwidth as well as the idle power of edge devices and offers their owners the opportunity to make a profit from it by renting their devices in their idle time. QNetwork will initially target gaming applications, but in the future the architecture is able to extend to real-time/low-delay sensitive applications. Another main difference is that in QNetwork, nodes will be acting as local servers hosting real-time applications, while in NKN the nodes are simply part of a large P2P

enabling developers to leverage the existence of an open-sourced Blockchain to develop DApps.

The rest of this chapter consists of the following sections. Section 4.1 is presenting similar related works which our problem definition was inspired by. Section introduces the system terminology. Section 4.3 discusses the proposed formulation of the optimal approach in detail in addition to the optimization formulation, and finally Section 5.1 illustrates the proposed AI model as a sub-optimal approach for QNetwork.

4.2 System Terminology

Here we start by defining some of the terminologies that will be used in our explanation of the problem and its constrains. Figure 4.1 illustrates a possible distribution of the system components.

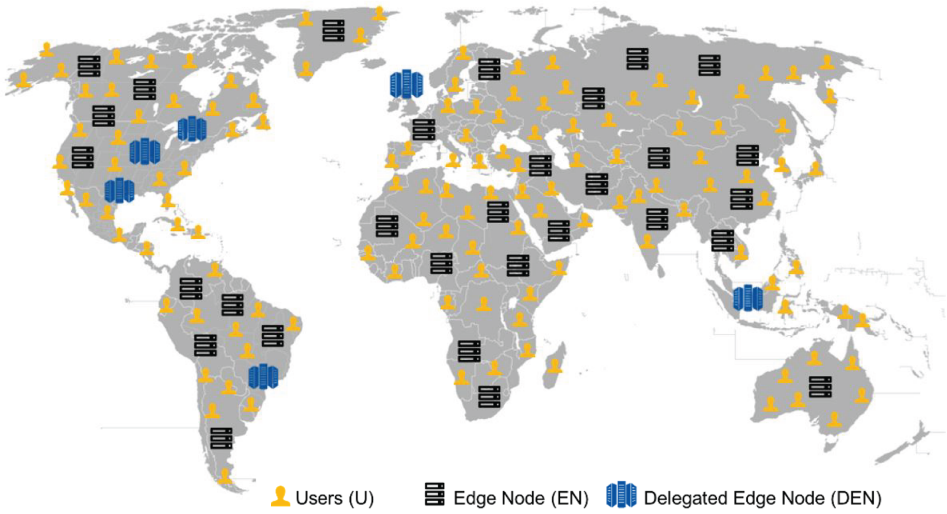


Figure 4.1: DEN, EN, and U Distribution.

4.2.1 Edge Node (EN):

EN is a user computer/server who agreed to let the company rent his machine and act as a server to host tournaments in uncovered areas. This type of user will be ranked by a certain ranking criterion that will be defined in a later stage.

Constraints:

- Each EN must connect to a DEN (see section 4.2.2) to save the session's state and to avoid any loss of data and to ensure failure handling.
- The total number of EN is expected to be in the millions.
- The CPU capability, RAM, GPU, RTT between EN or U and all DENs, and Internet Bandwidth of the EN are known features.
- One EN can support up Y Us and hold only one session (typically could have a maximum of 12 in gaming applications).

4.2.2 Delegated Edge Node (DEN):

DEN is a special EN selected to be an "entry" point where all prospective users should connect to first. A DEN can also be considered as an EN that can handle multiple sessions according to its capabilities, although in our mathematical model we consider each of those sessions as a separate logical DEN, so in our model a DEN handles one session.

Constraints:

- The number of DENs is not fixed, and they will be dynamically changing. To start with, there are 10 different DENs and few are in the following cities: Toronto, Chicago, Dallas, London, Singapore, and Sao Paolo.
- The average delay from each DEN to all others is known.
- A DEN is an EN that can handle one session, and it will be selected by an algorithm to perform the network tasks. However, according to its capabilities and for the sake of simplifying the mathematical optimization, if the DEN/EN can handle more than a session, it will be replicated.

4.2.3 User (U):

U is the client who using an application and tries to connect to one of the company's DENs.

Constraints:

- Each U must initially connect to a DEN and later possibly connect to only one EN at a time to be able to use the application.
- All the users connecting to the same gaming session (same instance of the game) should be connected to the same EN or DEN that is hosting the game.

4.3 Proposed Formulation of Optimal Approach

In this section, we will try to address the connection decision-making problem characterized by choosing the EN users should connect to. Once we have an estimate of the end-to-end delay, bandwidth, CPU, RAM, GPU capabilities, and other features of ENs, we want to optimize the choice of an EN for a new U such that the QoE of users is optimized (delay is at minimum, bandwidth is at highest, etc.).

When a user attempts to connect to a gaming session, the system must check the geo-location and initially assign him to the nearest DEN. Later, the algorithm will assign him to a EN if the user is off the DEN's zone. The system should be intelligent in making a decision based on the rankings of the EN and the new user U's geo-location and delay metrics.

A possible solution is using AI/machine learning to optimize this multi-constrain problem. Additionally, the system should rank EN according to their reliability and machine abilities; bandwidth, CPU, RAM, and GPU. The user will connect to the DEN/EN which has the highest rank, highest availability coefficient and minimum delay which should be less than a certain threshold (e.g 50 msec as a requirement from Swarmio Media).

4.3.1 Delay and Bandwidth:

We will start by explaining some terminologies. We will denote the DEN and each of ENs and Us as nodes in a network diagram as illustrated by figure 4.2. The delay between any DEN and a EN will be denoted by d_{EN}, while the delay between each U and EN/DEN will be denoted by d_{U_i}; where i represent the U number.

In order to ensure the QoE of the users, we have a constrain to limit the end-to-end delay between users and EN/DEN, to be less than 50 msec (as an example of a defined threshold), Then, we add:

$$d_u < 50 \tag{4.1}$$

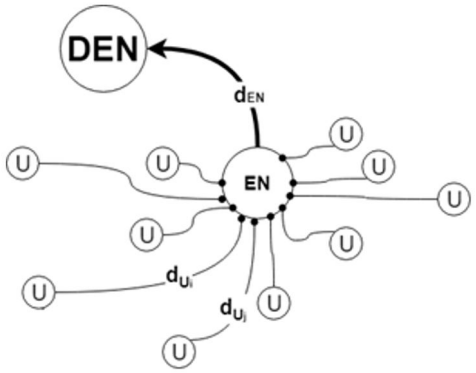


Figure 4.2: Stable Network Diagram.

Then we define the total delay in the network $d_{u_{total}}$ to be the sum of all delays between each U and session admin (EN/DEN).

$$d_{u_{total}} = \sum_{i=1}^{N} d_{u_i}, \text{where N is the total number of U nodes.} \tag{4.2}$$

Then the average delay can be calculated as follows:

$$d_{u_{Avg}} = 1/N * d_{u_{total}} \tag{4.3}$$

To account for the usable bandwidth at the users' side, we denote user u's bandwidth by bw_u, and the EN/DEN's available bandwidth; i.e., how much bandwidth the EN/DEN has left when user u is joining it, as available bw_{EN}. It then follows logically that the usable bandwidth between user u and the EN/DEN is:

$$\text{Useable } bw_{u-EN} = \min(bw_u, bw_{EN}) \tag{4.4}$$

Then we define the ratio between the bandwidth and the delay for each user to be the network metric (X). X will be a heuristic measure that will be later used for performing the selection and the optimization.

$$X_{u-EN} = \frac{\text{Useable } bw_{u-EN}}{d_u} \tag{4.5}$$

The range of X is $[0, \infty)$, so we normalize X to the range $[0 - 1)$ by dividing the X values by the maximum X obtained.

4.3.2 Ranking ENs:

In order to evaluate the network sessions and how good the choice of sessions' connections are, we need to verify the algorithm's choices after it is outputted. We came up with a few different possible evaluation methods that could be utilized for such a purpose. Additionally, those rankings could also take part in determining the payment amounts in later stages and other related recommendations in the network.

1) Ranking using Bandwidth-Delay Ratio (BDR): (How good is the EN connection in the network session?)

This score will rank the ENs in the network according to the performance of each EN in the session. Here, we will use the ratio of the available bandwidth of the EN and the average delay in the session, as shown in formula 4.6. *BDR* reflects the reliability of the EN since it measures the data capacity it is able to send (the higher the better). Thus, it gives an indication about the performance of the EN which is needed for ranking purposes.

$$BDR_{EN} = \frac{BW_{EN}}{d_{u_{Avg}}} \tag{4.6}$$

Where BW_{EN} is the available bandwidth of the EN, and $d_{u_{Avg}}$ is the average delay of all connected users to the same EN

2) Ranking using user feedback (FB): (How good is the user experience during the network session?)

After finishing a session, users will be asked to rate the experience given by the session. The rating can be from 0 to 5 (0 being the worst), and this will be used for future decisions or a possible learning algorithm. This score will be denoted by FB_{EN}.

3) Ranking using EN capabilities (CAP):

Here, we can define levels of available resources and capabilities which we can rank ENs upon. E.g. Level 0 would have a min of 12 GB RAM available, Level 2 would have a min of 8 GB RAM available, etc. This score will be denoted by CAP_{EN}, and will also be normalized to be in [0,1] by dividing the value by the maximum level number.

4) Weighted Ranking score (R_W):

In order to have a more accurate vision to performance of each EN, a weighted score will be given to each EN based on the different rankings. Each score will be multiplied by a weight that can be set initially randomly and then can be modified when the network grows. The condition on the weights is: $\alpha + \beta + \gamma = 1$.

Additionally, since the rankings have different range of values, they should be normalized to a value between the range $(0, 1)$. This is achievable by dividing BDR_{EN} by the maximum obtained value, dividing FB_{EN} by the maximum rating (in our case it is 5), and dividing CAP_{EN} by the number of levels.

$$R_{W_{EN}} = \alpha * BDR_{EN} + \beta * FB_{EN} + \gamma * CAP_{EN} \tag{4.7}$$

5) Ranking using a User-based analysis: (How good is the EN positioned in the network?)

Since we make selections from the EN perspective, an analysis from the U perspective is necessary to evaluate the selection decision. For every U connected to a EN in a given gaming session, we will draw a 50 ms radius circle similar to what we have done previously, but from a U prospective. ENs within the range for each U are ranked based on the delay measured/estimated, then the U will be given the rank of the EN that it is connected to. This rank represents how good/bad the EN-U connection is based on the delay (the lower the better). In figure 4.3, the U is connected to the EN that is ranked 3rd based on the delay (the U is connected to the 3rd lowest EN in terms of end-to-end delay).

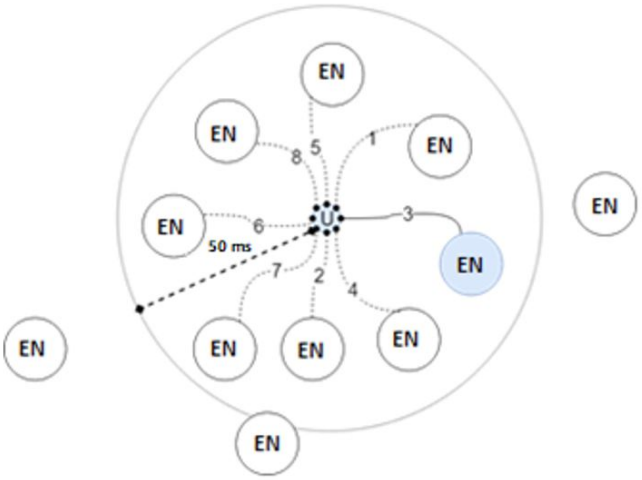


Figure 4.3: Diagram of a U-based ranking with a 50 ms radius circle..

The process is repeated for every U node in a session, and then they are averaged and the rank will be assigned to the EN in a given session. In figure 4.4, similar to what we have explained above, the process is repeated for every U in a session, and then in this example, the EN will have a score of 2.5. This means that the EN is, on average, the 2nd-3rd most preferred EN for the Us in the session. This rank will be updated session after session until the scores reach a point of stability. This score can be used in future connection decisions as well as payment purposes because it indicates the importance of how well the EN is located.

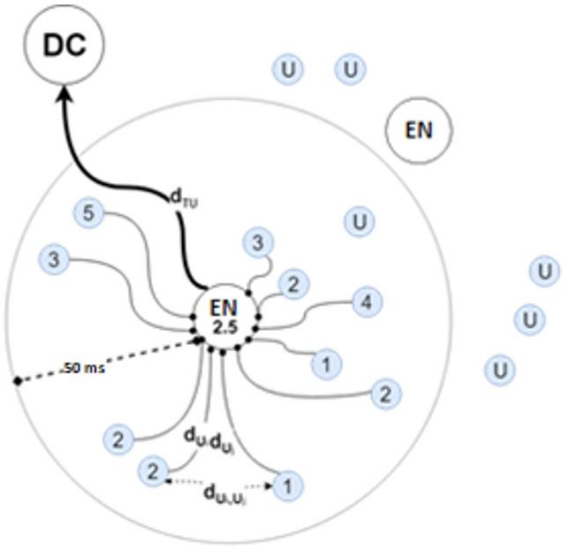


Figure 4.4: Diagram of a U-based ranking for a network session.

6) Nodes' Centrality (Other possible measures):

There are other possible centrality measures that can be applied to the subset of the whole network which is the Blockchain of ENs and DENs. These measures include: between-ness centrality, Eigen-Vector centrality, and PageRank centrality.

4.3.3 Failure Handling:

Here we will be discussing the possible solutions for some cases of EN failure. It is essential that the network handles a failure of some ENs in a decentralized fashion.

DEN Connectivity.

In case of EN session failure, the DEN will have a recovery point of the latest state, and the users will be assigned to another DEN or EN to resume the session at. The assignation of the

EN/DEN will utilize the same algorithm. In order for this to be valid, the ENs should feed the state periodically to the nearest connected DEN.

Limitation. In case that a re-accommodation to resume the session was not possible (given certain limitations, e.g. delay > 50 msec as a requirement from the industry partner, Swarmio Media), an alternative should be provided. In this case, we can proposes turning one of the recently disconnected Us into a EN (rank, resource and latency dependable). Then, the session can be resumed, and the new EN can replace the old EN duties.

4.3.4 Optimization Formulation

Since now we have formulated the problem, we will try to optimize the network based on the defined network metric (which is the usable bandwidth over the delay between Us and EN/DEN). As an example, we have the network shown in figure 4.5 where we have 2 ENs and 15 Users. We suppose that the EN can take up to 12 Us, and we suppose that each of the 2 ENs have 5 Us connected to them, while there are 5 Us that are not connected to a EN/DEN yet (connected by a dotted line).

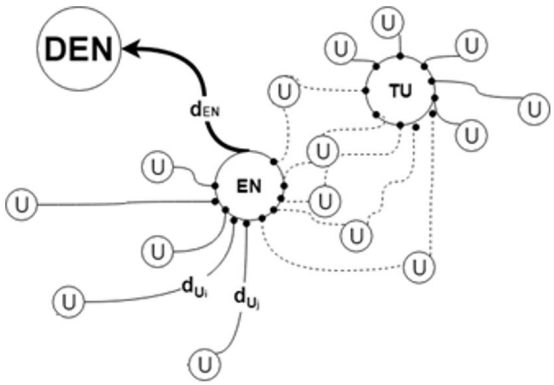


Figure 4.5: Diagram of a network to be optimized.

We assume that we have N users where $U = u_1, u_2, \ldots u_N$, M DENs and ENs (machines hosting sessions) where $D = d_1, d_2, \ldots d_M$, each with the capability to connect to Y number of users (typically $Y = 12$), M gaming sessions where $C = c_1, c_2, \ldots c_M$ where each session contains a subset of users in U connected to one EN/DEN from D, and network matrices X_{ij} for each user $\forall i \in C, \forall j \in D$.

In order to maintain the QoE, we would like to minimize the average delay while maximizing the average bandwidth from all combinations of choices of connecting Us to EN/DENs, while at the same time choosing the highest ranked EN/DENs as much as possible. Hence, our optimization formula becomes:

$$\max : \sum_{i=1}^{N} \sum_{j=1}^{M} X_{ij} + R_{Wj} \tag{4.8}$$

Subject to:

$$d_u < 50, \quad \forall i \in C, \forall j \in D \tag{4.9}$$

$$\cap_{j=1}^{M} c_j = \emptyset \quad \forall j \in C \tag{4.10}$$

The first constraint ensures the end-to-end delay is not larger than 50 msec, while the second constraint specifies that each user can only be connected to one session. The choice of the network connections will be made by the nearest DEN as it will have the authority to optimize the network. Since each EN is connected to a DEN, and each U has to connect to a DEN in order to start the session, using a DEN to make decisions is a fair choice. Whenever a gaming session is about to start, the DEN will be responsible for assigning the Us to the sessions based on their X value.

In other words, this can be illustrated in figure 4.6 as DEN will assign each EN the Us that lie in a 50 ms radius to connect to it. The choice of Us will be according to the X ratio explained earlier (maximum available bandwidth and minimum end-to-end delay).

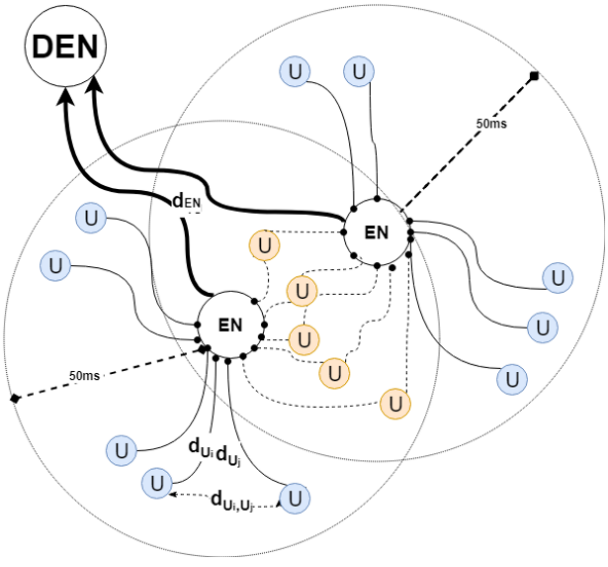


Figure 4.6: Diagram of a network with a 50ms radius circle

4.3.5 Approaches Complexity and Drawbacks

Solving this optimization problems could be computationally expensive. In addition, the solution would only be valid for a static setting: as users join and leave, the optimization problems needs to be solved over and over again. That creates a huge obstacle in adopting the system in highly changeable environments such as the gaming networks and edge clouds in general. Additionally, introducing new constrains to the problem as the environment evolves requires revisiting the formulation, modifying the rationale, resolving the optimization, and obtaining the results and the limits every time. Hence, a more dynamic solution should be adapted to capture insights from the problem rather than resolving the equation again, and AI-based models are the best fit in this context.

As noticed, the high complexity of the optimization is a result of having a large number of parameters to be optimized. Therefore, we decided to use an existing model that has defined the EUA problem and been already established and proven in the literature. The recently publised work models the EUA problem as a Variable Sized Vector Bin Packing (VSVBP) Problem, and it has been considered the state-of-the-art in solving the EUA problem. In the next section, we will cover the formulation of the EUA problem, and we will tie it to the Cloud gaming context.

4.4 EUA in Cloud Gaming Literature - A Motivating Example

An example of latency-sensitive applications that is used in the literature is shown in figure 4.7 by He et. al in [56]. Taking the figure as a cloud gaming network substrate, the ovals represent the game's latency threshold; i.e., players within a server's oval can be supported by that server. Also, each server's CPU power, memory, storage, and bandwidth capabilities are shown below the server, in that order. The figure shows why an EUA solution is far from trivial: it's a multivariate optimization problem because each player has the option of connecting to multiple servers; e.g., player 8 is within the latency threshold of servers 2, 3, or 4. At the same time, each server comes with limited capacity in terms of the said 4 factors. In addition, the system must provide the aforementioned fairness.

To better appreciate EUA's complexity, let's assume there are 4 servers, denoted by s_{1-4} as shown in figure 4.7, and 10 players denoted by u_{1-10}. If we assume that each server can handle all 10 players, theoretically we will have $10{\times}9{\times}8{\times}7 = 5,040$ number of matching possibilities. But servers have limited computing capacity, denoted as a vector $\langle CPUcore, memory, VRAM, bandwidth\rangle$. Similarly, the workload generated by players is denoted by that vector. Naturally, the total player workload assigned to a server should not exceed the server's capacity. For example, in server s_1's

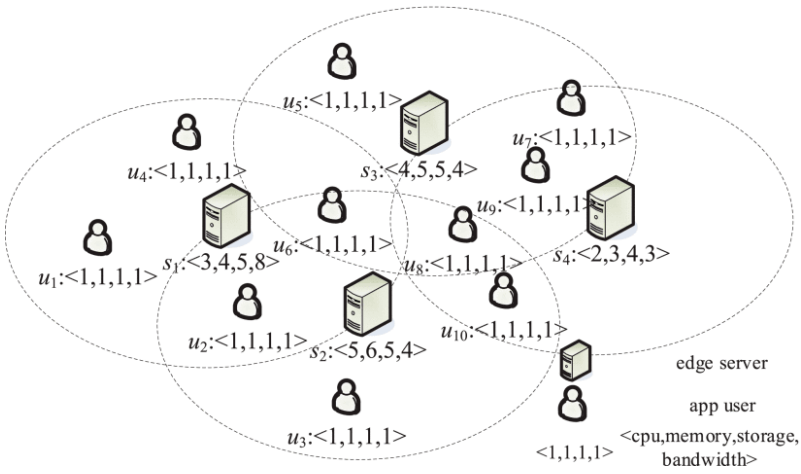


Figure 4.7: An example of the EAU problem and Edge computing deployment. ©IEEE reused with permission [56].

proximity, there is u_1, u_2, u_4, and u_6 generating a total workload $\langle 4, 4, 4, 4\rangle$ exceeding the server's capacity of $\langle 3, 4, 5, 8\rangle$. Thus, the game provider will fail to assign those players to that server. One solution is to assign u_1, u_4 to s_1, and u_2, u_6 to s_2, but it might not be optimal for the rest of the system. Another solution is to assign u_1, u_4, u_6 to s_1, u_2, u_3, u_8, u_{10} to s_2, and u_5, u_7, and u_9 to s_3. This way server s_4 will not be required, lowering the total cost. This solution satisfies both the proximity and capacity constraints, and it uses fewer servers, but does not guarantee fairness.

4.4.1 EUA as a Variable Sized Vector Bin Packing (VSVBP) Problem

To compare our solution with an analytical one, we use the multi-objective optimization problem definition in [67] where the authors defined the EUA problem as a as a Variable Sized Vector Bin Packing (VSVBP) Problem. Let $S = \{s_1, ..., s_i\}$ represent the set of edge servers, and $U = \{u_1, ..., u_j\}$ the set of users. $C_i = \langle C_i^1, ..., C_i^d\rangle$ is a d-dimensional vector representing the remaining capacity of the servers s_i, and $w_j = \langle w_j^1, ..., w_j^d\rangle$ a d-dimensional vector representing the workload by user u_j. Finally, d_{ij} represents the distance between server s_i and user u_j, while $cov(s_i)$ represents the delay requirement by game providers (originally, represents the coverage of server s_i). The problem can be formulated as follows [67]:

$$\text{Maximize} \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} \tag{4.11}$$

$$\text{Minimize E} = \sum_{i=1}^{m} y_i \tag{4.12}$$

subject to:

$$\sum_{j=1}^{n} w_j^k x_{ij} \leq C_i^k y_i, \tag{4.13}$$

$$\forall i \in \{1, ..., m\}; \forall k \in \{1, ..., d\}$$

$$d_{ij} \leq cov(s_i), \tag{4.14}$$

$$\forall i \in \{1, ..., m\}; \forall j \in \{1, ..., n\}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \forall j \in \{1, ..., n\} \tag{4.15}$$

$$y_i \in \{0, 1\}, \forall i \in \{1, ..., m\} \tag{4.16}$$

$$x_{ij} \in \{0, 1\}, \tag{4.17}$$

$$\forall i \in \{1, ..., m\}; \forall j \in \{1, ..., n\}$$

where:

$y_i = 1$ if server s_i is hired.

$x_{ij} = 1$ if u_j is allocated to s_i.

$cov(s_i)$ is the delay requirement by game providers.

Objective (4.11) maximizes the number of users, while (4.12) minimizes the number of servers. The *capacity constraint* is preserved by (4.13) which dictates that user workload must not exceed the servers' remaining capacity. The *proximity constraint* is satisfied by (4.14) which ensures that only users located within a server's coverage are assigned to that server. However, in our case of Cloud Gaming, the proximity constraint is replaced by the delay threshold provided of the game. Finally, constraint (4.15) prevents a user from being allocated to more than one server. The problem is then solved with Lexicographic Goal Programming (LGP) which optimizes multiple objectives by their importance [67], and by their definition, objective (4.11) had a higher importance than objective (4.12).

After going through the problem formulation using the optimal approach, in the next chapter, we will cover our proposed methodology to solve the problem using an RL-based method.

Chapter 5

Proposed Methodology

5.1 Motivation

In order to have a robust and a scalable solution for this problem, AI (and specifically Reinforcement Learning) has an immense potential to be utilized in this context. We are investigating the possibility to apply Deep Reinforcement Learning (DRL) and/or Reinforcement Learning (RL) in this given problem, and we believe that building an AI agent ease the decision-taking task as the network grows. It will keep on getting smarter since it will be dealing with big amounts of data and it will be modifying the future actions to be taken based on the feedback of previous actions taken in previous episodes. The advantage of this approach is that some of the RL algorithms are model-free which means that it doesn't need any prior knowledge of the environment it was applied to. Reinforcement learning (RL) refers to goal-oriented algorithms, which learn how to attain a complex objective/goal or maximize along a dimension over multiple steps. The RL algorithms are penalized when they make the wrong decisions and rewarded when they make the right ones, and that's how they learn.

In the literature, using RL/DRL is a research trend, and it has been showing promising results with similar applications. DRL has shown impressive results in the famous paper [77] where the agent was able to play videogames without any prior knowledge of the games' environments. Another work [87] utilized an DRL agent to optimize routing in a small network.

In this chapter, we will present an RL solution to the multivariate multi-constraint problem of fair Edge server selection for latency-sensitive applications such as cloud gaming. Since the number of edge servers are high, the optimum and fair user-server matching is a difficult problem, and we will demonstrate that RL, specifically Quadruple Q-learning, can solve the problem much better than existing approaches, with a significant reduction in the standard deviation of the user-server latency, leading to more fairness.

5.2 Proposed AI models for Sub-Optimal Approach

Rationale

Consider the simplified gaming subnetwork shown in Figure 4.6, where U, EN, and DEN are the User (player), the Edge Node (a single Edge server), and the Delegated Edge Node, respectively. EN and DEN are part of the gaming service provider's gaming platform: edge/cloud infrastructure that is either rented from the likes of Google, Amazon, Microsoft, etc., or built proprietary such as Sony PlayStation Now's Gaikai platform. DEN is a special EN that users connect to first. The DEN then decides which EN the user should connect to, so the server selection algorithm runs in DEN.

An EN can handle multiple users, but its capacity is finite. The system must ensure that the latency between the user and the EN does not violate the specific game's delay thresholds. For example, if some users are playing together Counter-Strike which has an end-to-end delay threshold of 100 msec, then the one-way delay between any player and its EN cannot be more than 50 msec, giving a radius of 50 msec to each EN shown in Figure 4.6. In addition, the system must ensure that the latency experienced by the players in that session is as close to each other as possible; i.e., the latency variation among players is minimized, in order to offer fairness and a level playing field.

As we can see from the figure, the players highlighted with orange can connect to more than one EN. Now, consider that figure 4.6 shows only 2 ENs; in the real world there are many more ENs that a player can connect to. In fact, cloud gaming systems can be massive. For example, the game EVE Online recorded 6557 players concurrently in the same battle (same session) on October 6, 2020 [35]. It is not unusual for cloud games to support millions of concurrent users [2], though not all in the same session. If we consider also the viewers and not just the players, then the massiveness of the system becomes even larger; for example, the LoL Mid-Season Invitational 2018 had 60 million viewers, 24.6 million of them concurrently [39].

Therefore, a cloud gaming system has multiple options of servers to connect to (ENs) and multiple constraints (latency threshold, EN maximum capacity). This becomes a multivariate optimization problem that is difficult to solve efficiently, especially when adding to it the fact that not only players may join or leave at will but also the provider may dynamically deploy (on-demand renting) more ENs or remove some ENs, making the problem highly dynamic. Considering that, as explained earlier, each game depending on its genre and pace has different latency requirements, assigning players to servers becomes a non-trivial challenge [42].

Since we use a distance function for selection, we also need to define what fairness means in that context. A fair user-server matching policy in this context should match pairs with a distance

close to the global average distance of the previously matched set. In other words, the matching policy's main goal is to have the lowest variance in distance. As we saw in Chapter 3, when matching a new user to a server, existing methods choose the closest server (lowest latency) on a first-come-first-serve basis. But, as we will show in Chapter 7, this leads to unfairness: early users will be privileged with the closest servers while later users will be matched with servers that are typically further, because the closer servers will have run out of capacity by that time. In comparison, we use RL and propose a solution that provides more fairness while also better meeting the latency thresholds.

It should be noted that although the above have been described in the context of cloud gaming, they equally apply to any latency-sensitive multi-user application, such as virtual/augmented reality, telepresence, teleconferencing, and telecollaboration.

5.2.1 Specifics of Our Problem

As we are solving a business case for our industry partner Swarmio Media, it helps to understand how games are played in Swarmio's gaming platform. Players first connect to Swarmio's portal, where they can be teamed together by a matching algorithm or by prior agreement. We can also have the everyone-against-everyone scenario, in which there are no teams. Then, two cases can happen:

1. The first case is when each team of players can be matched to the same edge server, because the delay distributions for the team's players are within threshold for that server. The advantage of allocating a server for the whole team is that communication and update sharing among team members becomes really fast. The game state will be maintained and shared among the edge servers via Swarmio's ultra-low latency platform.

2. The second case is when the distribution of players requires the allocation of those players to different edge servers, irrespective of their team (or in the case of no teams). In this case, the game state will still be maintained and shared among the edge servers via Swarmio's ultra-low latency platform.

For both of the above cases, we note that:

1. All players are in the same gaming session. Our work targets a single gaming session, not multiple parallel sessions. In other words, our algorithm must be applied to each gaming session separately. This makes sense because we are interested in fairness among players who play in the same gaming session. Fairness does not apply to players who are not in the same gaming session, as those players are not playing with/against each other.

2. Latecomers are not allowed. Players must be ready at the beginning of the session. Once the session starts, no one can join anymore. This is due to the nature of games offered by Swarmio, such as Counter Strike, League of Legends, etc. When such games are played in tournament mode, players need to be present from the beginning of the session and cannot join in the middle.

3. For the same reason, players are also not allowed to move among sessions during gameplay.

4. A server can be assigned to one gaming session, not multiple. Currently, Swarmio servers are not shared among multiple parallel gaming sessions: it is decided beforehand which servers are used for which gaming session. However, this manual process is not scalable and must at some point in the future be made automatic.

5.3 The Proposed QNetwork system

The problem described in Chapter 4 is a highly dynamic and rapidly evolving multivariate optimization problem. This suggests the usage of Reinforcement Learning, specifically Q-learning due to its flexibility and dynamic nature as a model-free algorithm. We use Q-learning in our solution, and consequently call our system QNetwork.

To solve the problem, we can design a Q-learning model with a reward that enforces fair edge server selection. However, typical Q-learning models will be selecting the closest possible server even if its capacity has been reached. Hence, we apply a few novel techniques to Q-learning in order to make it applicable to our fair server-selection problem. It should be noted that the said techniques are not limited to fair server selection in edge clouds, and can also be applied to any matching problem where there are 2 sets that need to be matched based on specific constraints.

Using RL terminology, we consider RL states to be the users joining the network, and RL action space to be the available hosts/servers to handle those users. Throughout the thesis, we will synonymously associate ”actions” to ”servers” and ”states” to ”users”, and we alternate in the usage of those terms based on the context.

Next, we will take a detailed look at classic Q-Learning, introducing action suppression, and then how we use it in our system design. After that, we will present our Q-Learning models.

5.3.1 Classical Q-learning as a Reinforcement Learning Model

In this section, we will go in detail in explaining the modified Q-learning algorithm that was used for the server selection problem. We will start with the classical Q-learning implementation

and then grow slowly on how the algorithm was modified in order to fit the context of server selection. We will be justifying the design choices along with explaining the modifications. As previously discussed, Q-learning is one of the most famous reinforcement learning algorithms which is a temporal difference-based (TD), off-policy, and model-free algorithm. Q-learning's main update to the Q-values is in equation 5.1:

$$Q(S,A) \leftarrow Q(S,A) + a[R + \gamma * max_a Q(S',a) - Q(S,A)] \tag{5.1}$$

Each episode, the Q-value is updated according to equation 5.1 by taking the action with the highest value estimate $max_a Q(S',a)$ of the next state-action pair. This update is similar to the update of SARSA (which is another TD algorithm) except for the fact that it is an off-policy with respect to the bootstrap action it takes at the next value estimate of the next state.

Algorithm 3 illustrates main steps of the Q-learning algorithm. $Q(s,a)$ indicates the Q-table entries, S indicates the state space, A indicates the action space, R is the reward, γ is a discount factor, and α is the learning rate.

Algorithm 3 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$ for all $s \in S, a \in A(s)$, arbitrarily.
Set $Q(terminal - state, \cdot) = 0$
for each episode **do**
    Initialize S
    repeat (for each step of the episode):
        Choose A from S using policy derived from Q(e.g., ϵ-greedy)
        Take action A, observe R , S' *(according to a reward function)*
        $Q(S,A) \leftarrow Q(S,A) + a[R + \gamma \cdot max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
    until S is terminal

5.3.2 Action Suppression

Motivation. Since the computing power of edge servers are limited, after some time an edge server could reach its maximum capacity. As a result, some actions/servers are expected to be unavailable at some point of time. Therefore, we must consider the availability of those servers before taking the action and updating their Q-values, because the reward returned by taking any of those unavailable servers could negatively affect choosing that server in the future. Unlike existing action mask methods where the rewards are updated after the actions are sampled, action suppression completely removes those actions from consideration. The technique not only masks servers out of the selections, but also has the ability to "suppress" and choose a certain action even

if the Q-value isn't the highest. That "suppressed" action could be user coded in certain cases where we want to use certain servers at certain times. A similar work on action suppression in [100] introduces the Action Elimination Network (AEN) which outputs a linear contextual bandit model that eliminates actions with high probability. This helps overcome difficulties in massive action spaces in situations such as text generation. Inspired by that approach, we have adopted the concept and modified the technique for a tabular case for fair matching problems. Algorithm 4 illustrates the detailed technique, where our additions are in lines 2, 8, 9, and 10.

Technical implementation. The proposed solution is to suppress certain actions during the learning process when they are no longer available. Similar to the elimination signal used in AEN and instead of using a probability to remove actions, we use an internal record of admissible actions. This could be simply implemented by defining a set of available actions ($A_{available}$) which will keep track of the actions that are possible to be taken in the next iterations. When choosing the action with the highest Q-value, the algorithm will keep track of whether it is available, because if it is not, it will choose the action with the next highest Q-value. In our application, once the capacity of a single server has reached maximum, it will be removed from the available actions set before the next iteration. If at some point one or more users from that unavailable server leave, then the serve becomes available again.

Action suppression's exploration effect. Q-learning could get stuck in sampling certain actions iteratively, so the famous epsilon-greedy solution addresses the exploration vs. exploitation common dilemma [88], in which the algorithm chooses actions with the probability of $1 - \epsilon$. Suppressing actions enforces exploration in Q-learning in certain cases. Since the eliminated action could be one of the most selected actions, it's possible that it will be selected in later iterations with a high probability. As a result, removing the highly chosen action from the available actions will be forcing the algorithm to choose the second best; i.e., letting the algorithm explore other actions that are sub-optimal based on their sum of expected return of rewards. This would fall in between exploration and exploitation; the algorithm is neither taking a random action nor using the optimal action according to its policy, but doing something in between. When the action is removed, the policy will sample an action in an ϵ-greedy fashion; i.e., the next sampled action is not necessarily the second highest Q-value action since it could be a random one with ϵ probability.

It is important to note that the RL agent will be trained offline first in order to get the knowledge about the environment and train the policy parameters to converge closer to the optimal policy. At this phase, the algorithm will not be penalizing users it has not seen before when exploring; it will simply be gaining knowledge about how to match and obtain the fairness in

server to user allocation. However, when deploying the model and training it online, it is possible that the algorithm would penalize a user-server allocation (a.k.a. the action to match them) in order not to take that action again given the current state of the network. That can be mitigated if that state was faced again and the matching happened to be a good matching, and in this case, the "penalty" or the Q-value will be adjusted converging to its true value to reflect the future expected rewards in the future episodes of training.

Action suppression's regularization effect. Dropout for Neural Networks was proposed by Srivastava et al. [86] where the output of random neurons is multiplied by zero to prevent overfitting. Suppressing actions in Q-learning could also be compared to dropout in Neural Networks. Since in dropout certain neurons are dropped and their output is ignored, removing actions is similar to multiplying their Q-value by zero and ignoring their effect on choosing the next action. Generally, that performs a form of regularization in training since it introduces a situation where it breaks the exploitation cycle which might be essential in a cases were actions and state spaces are large.

Future implications of action suppression. Eliminating actions could have a future implication on building a more sophisticated algorithm that could capture more insights from the network. The actions that are eliminated indicate that they are in high demand in certain areas at certain times. That could be a great piece of information to keep and this could be utilized to make recommendations on where to deploy servers. Adding more factors would result in building a smarter agent that learns and predicts the next state of the network. The capacity left in servers could also be employed in their selection in order to avoid any overhead on certain servers while others are available within the acceptable range. Also, it is important to mention that the values of the parameter $A_{available}$ can be both fixed and learned. In our application, we used the fixed capacity of the servers to indicate the availability; however, this value can also be learned to eliminate overloading servers with tasks and requests close to their capacity limit.

5.3.3 Proposed Q-learning Models

As explained earlier, fairness in our system is defined as matching a user to a server close to the global average of distance (given by a specific distance function) among all user-servers in the same gaming session. Therefore, in the implementation of our Q-learning model, we defined a global variable, D, which is the set of distances. This global variable is updated at every iteration, and is reflected in the reward of actions.

Algorithm 4 Q-learning with actions suppression

initialize $Q(s, a)$ for all $s \in S, a \in A(s)$, arbitrarily.
define $A_{available}$ **for all possible** $a \in A(s)$
set $Q(terminal - state, \cdot) = 0$
for each episode **do**
    initialize s
    repeat (for each state of the episode):
        choose highest a for s using policy derived from Q(e.g., ϵ-greedy)
        while $a \notin A_{available}$ **do**
            choose next highest a using the same policy
        take action a, observe r , s' *(according to a reward function)*
        remove a **from** $A_{available}$ **if limit is reached**
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \cdot max_a Q(s', a) - Q(s, a)]$
        $s \leftarrow s'$
    until s is terminal **and** $A_{available}$ **isn't empty**

To choose a reward function in the context of fair selection, we designed 4 different reward functions, resulting in 4 different models plus 2 additional models combining all 4. All of the models have negative rewards as a function of the sum of the distances which we are trying to minimize. Having a negative reward will reinforce the agent to select actions that will minimize the global sum of distances, and that would be reflected on the Q-value of that action. The models are explained next.

Model 1: Distance (d) **as a reward** ($Reward = -1 * d$)**.** Here, d is the single distance of the matched pair according to a distance function (whether it was geo-distance, delay, euclidean distance, etc.). Since we are training the agent to reduce the distance, in this model the reward was the negative value of d. This will train the agent to pick the lowest distance possible which will potentially have the highest Q-value given the state of the network. And example of distance functions that can be used is the geo-distance between an edge node (e) and a user point (u) which can be calculated using their longitude ($long$) and latitude (lat) by:

$$d = \sqrt{(e_{long} - u_{long})^2 + (e_{lat} - u_{lat})^2} \tag{5.2}$$

Model 2: The standard deviation (STDV) of the set of distances (D) **of the matched users-servers as a reward** ($Reward = -1 * stdv(D)$)**.** In this model, the focus is to reduce the global average of the distances by setting the reward to be the negative value of the current standard deviation (STDV) of the distances of the current connections. Since the agent's selection is affecting the average to change, setting the reward to this value will train the agent to make

choices that minimizes the global STDV of the distances, therefore matching users and servers with a distance closer to the average distance.

Model 3: The change in the STDV of the set of distances (D) of the matched users-servers as a reward ($Reward = -1 * \Delta stdv(D)$). In this model, the focus is to reduce the occurrences of the actions that cause a fluctuation in the STDV of the distances. In contrast to the previous models, this model will give a negative reward proportional to the change in the STDV caused by their taken action. In other words, when the agent takes an action, it will be penalized by how much of a change they caused to the STDV of the delays. This would encourage taking more actions that cause less fluctuations in the STDV, and therefore, the action selection would be fair.

Model 4: The absolute value of the change in the STDV of the set of distances (D) of the matched users-servers as a reward ($Reward = -1 * |\Delta stdv(D)|$). similar to model 3, except that it takes the absolute value of the change. Since the change of the STDV could be a negative or a positive change, unifying the change is more meaningful. Additionally, the goal is to minimize the change in the STDV regardless of the sign, therefore, the use of absolute value of the change in the STDV is justified.

Model 5: Quadruple Q-learning (QQL). This model combines the 4 previous models discussed in order to improve the results. Since we had 4 different models operating on the same problem, combining the Q-tables of the models leads to creating a different model which leverages the experiences of all models in reducing the variance of distance. Inspired by [63] and their work on multi-table Q-learning, Model 5 looks at each state in the 4 corresponding Q-tables and picks the action from the table with the highest Q-value. The idea is similar to double Q-learning [54] with two additions: double Q-learning is single objective and uses the same reward model, while our problem is multi objective and selects from multiple reward models. Each of the 4 models will elect an action to be taken for the current state where the agent is at, and QQL will choose the action with the highest Q-value.

Model 6: QQL with Local Min-Max Normalization. The range of Q-values in QQL depends on the combination of the settings used in training because the values could have different scales affected by the number of episodes and reward function they were trained on. Therefore, in order to fairly compare 4 different Q-tables in QQL that have different scales, the values should be normalized. The idea is similar to Local Response Normalization (LRN) introduced in the famous AlexNet by Krizhevsky et al [65]. LRN improved a four-layer CNN's effectiveness

on CIFAR-10 dataset from %13 test error to %11 error. As will be seen in Section V, it also significantly improves our results. Our implementation is shown in figure 5.1.

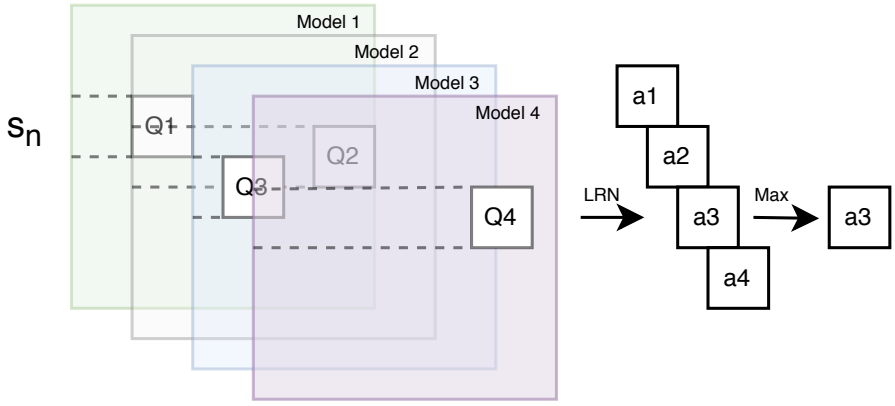


Figure 5.1: Model 6 - Quadruple Q-learning with Normalization

Each row of the Q-table (representing Q-values of a single state) is normalized with the min-max values of that row, as shown in the next equation.

$$X_{norm_{ij}} = \frac{X_{ij} - X_{min_i}}{X_{max_i} - X_{min_i}}, \forall i \in rows, \forall j \in columns \tag{5.3}$$

Extensions to Model 6. We use Model 6 in the experiments in section 7.1. However, in section 7.2, we expand this model with 3 different normalization settings. The first is normalizing the Q-values per state row which considers normalizing each row of the Q-table according to only the values of that row independently without considering the whole table. The second is normalizing the Q-values of the whole table which considers all the values in the table. The third is normalizing per state row first and then normalizing the whole table. We use both Min-Max and Softmax for normalization. The Softmax normalization is shown in the next equation.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}, \forall i \in rows, \forall j \in columns \tag{5.4}$$

Having 3 different settings and 2 different normalization functions results in 6 different normalized ensemble sub-models significantly improving our results, as shown in the Experimental Evaluation section. **Model 6.1** represents Min-Max normalization of Q-values per state row of the 4 ensemble models, **Model 6.2** represents applying Min-Max normalization on the whole Q-table, and **Model 6.3** applies Min-Max normalization per state row and then on the whole table. On the other hand, **Model 6.4** uses Softmax normalization of Q-values per state row, **Model 6.5** applies Softmax normalization on the whole Q-table, and lastly **Model 6.6** applies Softmax normalization per state row and then on the whole table.

Algorithm 5 illustrates the detailed model, where it combines QQL, Action Suppression, and LRN all together.

Algorithm 5 QQL with LRN, Action Suppression, and Fairness

initialize $Q_i(s, a)$. $\forall i \in \{1, 2, 3, 4\}, \forall s \in S, \forall a \in A(s)$, arbitrarily.
define $A_{available}$ for all possible $a \in A(s)$
set $Q_i(terminal - state, \cdot) = 0$
for each episode **do**
    initialize s, and total distances
    repeat (for each step of the episode):
        choose highest a_i for s using policy derived from normalized Q_i(e.g., ϵ-greedy)
        while $a_i \notin A_{available}$ **do**
            choose next highest a_i using the same policy
        update total distances *(based on the distance function for calculating next rewards)*
        take action a_i from table Q_i, observe r , s' *(according to a reward function)*
        remove a from $A_{available}$ if limit is reached
        $Q_i(s, a) \leftarrow Q_i(s, a) + \alpha[r + \gamma \cdot max_a Q_i(s', a) - Q_i(s, a)]$
        Normalize table Q_i with LRN
    until s is terminal and $A_{available}$ isn't empty

5.3.4 Q-table Scalability

Approximating Q-values of new states and new actions to the k-closest neighbours of nodes

Since Q-learning is designed to handle a finite and a pre-defined space of actions and states, there's a concern on scalability and practicality, since the number of incoming users and servers can increase, and a fixed-sized Q-table won't handle the increase. To address this, we built an approximating function that maps new states and new actions to the current Q-table. The approximation function averages the Q-values of the k-nearest neighbours of the new state (user) or the action (server) and then builds a record of the new entry.

Another approach that could be taken for mapping the new states and actions is a learning approach as a regression problem. In this approach the Q-values are learnt through a simple neural network regressor trained on a history of entries which is similar to [100] but emphasizing the notion of fairness. Since it is out of the scope of the current work, we keep it as a future work.

Q-table implementation is based on a dynamic hashmap

To further improve the aforementioned scalability, we also implemented the system's Q-table as dynamic hashmaps, a.k.a. dictionaries or hashtables, which are essentially lookup tables that

allow the fast and efficient lookup, appending, and removal of any instance associated with a given key.

5.4 Conclusion

In this chapter, we attempted to formulate the fair server selection problem as a reinforcement learning problem, and the approach taken was to tailor the reward function in order to reflect the notion of the fairness. We presented 4 different models with 4 different reward functions in order to optimize the latency variance in accordance to our notion of fairness. While designing the above-mentioned solution, we also introduced some novelties in RL, such as action suppression, Quadruple Q-Learning, and normalization of the Q-values, leading to a more scalable and implementable RL system.

The next chapter will introduce the collected dataset that is used later to evaluate the proposed models. The methodology of data collection and the details on the collected fields are also discussed.

Chapter 6

Cloud Gaming Client-Server Delay Dataset (CGCSDD)

6.1 Introduction

As previously discussed, optimum selection of game servers continues to be a topic of much research, as can be seen in [10, 11, 41, 46, 70, 72, 73] to give a few examples. Games have specific player-server network delay thresholds, and failure to meet these thresholds is one of the main reasons game service providers lose subscribers [32–34]. For example, First Person Shooter games require latencies of no more than 80 ms [9], while in Third Person Perspective cloud games every 100 ms of latency results in a 25% decrease in player performance even though such type of games can generally tolerate end-to-end delays of up to 500 ms [29]. As such, cloud gaming systems must meet these delay thresholds while minimizing the provider's costs. To meet the thresholds, providers can deploy edge servers in the service areas, reducing latency by bringing the computing server closer to the player [22]. But the number of edge servers can become large, and this large number of edge servers is exacerbated in the 5G era, because 5G provides very low network latencies, so edge servers that before 5G were just outside a player's latency range are now within that range because of 5G. With such a high population of potential edge servers, selecting the optimum edge server continues to be one of the main challenges.

To facilitate researchers who work on this challenge, in this chapter, we present the Cloud Gaming Client-Server Delay Dataset (CGCSDD)[1]; a dataset of client-server Round Trip Time (RTT) network delay measurements of an actual cloud gaming tournament run on the infrastructure of the cloud gaming company Swarmio Media. The dataset can be used for designing algorithms and tuning models for gaming server selection. To collect the dataset, tournament

[1]The dataset is publicly available at : `https://dx.doi.org/10.21227/jr75-0215`

players were connected to Swarmio servers and delay measurements were taken in real time and actual networking conditions. The dataset consists of two subsets: the main dataset contains network delays between each of 189 players around the world to each of 9 different Swarmio servers. The secondary dataset contains the delays between each of 67 players to each of 11 servers around the world. As an example demonstration, we use the dataset to test and report the results of our player-server fair allocation algorithm in the next chapters.

In the next section, we introduce our collection methodology and the dataset, while in Section 6.3 we present an example of using the dataset to test our server selection algorithm. Finally, section 6.4 gives the information about the availability and copyright information of the dataset.

6.2 Dataset

The dataset consists of two subsets. In the first dataset, data was collected during a series of online gaming tournaments run by Swarmio Media in 2018. In the tournaments, 189 unique players from around the world had access to 9 geographically distributed servers. We also ran a second collection session with 67 players and 11 different servers, with the resulting data available in the secondary dataset. The following subsections present detailed explanations of each dataset.

6.2.1 Main Dataset

For the main dataset, the 189 players and the 9 servers were distributed among 4 different regions: North America, South America, Europe, East Asia. The 9 servers were located in the following cities with their acronyms in the dataset:

1. Santa Clara (nasc),
2. Chicago (nach),
3. Dallas (nada),
4. Toronto (nato),
5. Brazil (sabr),
6. London (uk),
7. Amsterdam (nl),

8. Hong Kong (hk),

9. Singapore (sg).

Each of the 189 players were able to connect to each of the 9 servers. The following data is registered for each player:

1. User Identifier (in the field: user_id)

2. Time of access (in the field: timestamp)

3. Longitude (in the field: longitude)

4. Latitude (in the field: latitude)

5. IP Address (in the field: address)

6. Access Support Network or Internet Service Provider (in the field: asn_org)

Most of the players were located in North America and South America, as illustrated the Figure 6.1.

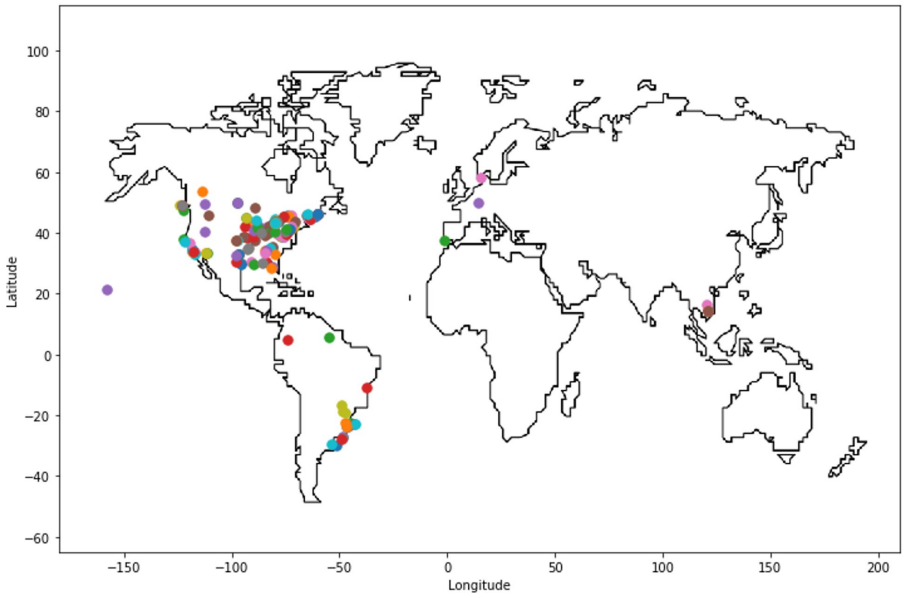


Figure 6.1: The distribution of players in the main dataset.

In the dataset file *main-dataset.json*, every record contains the network delay measurements from a particular player to each of the 9 servers. It should be noted that the URLs and the IP

```
{"https://nach.swarmio.gg:2222": "38.65.238.5",
"https://sg.swarmio.gg:2222":"63.251.127.116",
"https://uk.swarmio.gg:2222":"95.172.86.132",
"https://nato.swarmio.gg:2222":"38.65.236.132",
"https://hk.swarmio.gg:2222":"203.190.121.20",
"https://nada.swarmio.gg:2222":"162.217.97.4",
"https://sabr.swarmio.gg:2222": "177.54.158.99",
"https://nasc.swarmio.gg:2222":"107.6.92.150",
"https://nl.swarmio.gg:2222": "72.26.219.37"}
```


Figure 6.2: The URLs and IP addresses of the servers in the main dataset.

addresses of the servers are provided in a separate file *main-dataset-servers.json*, as shown in Figure 6.2.

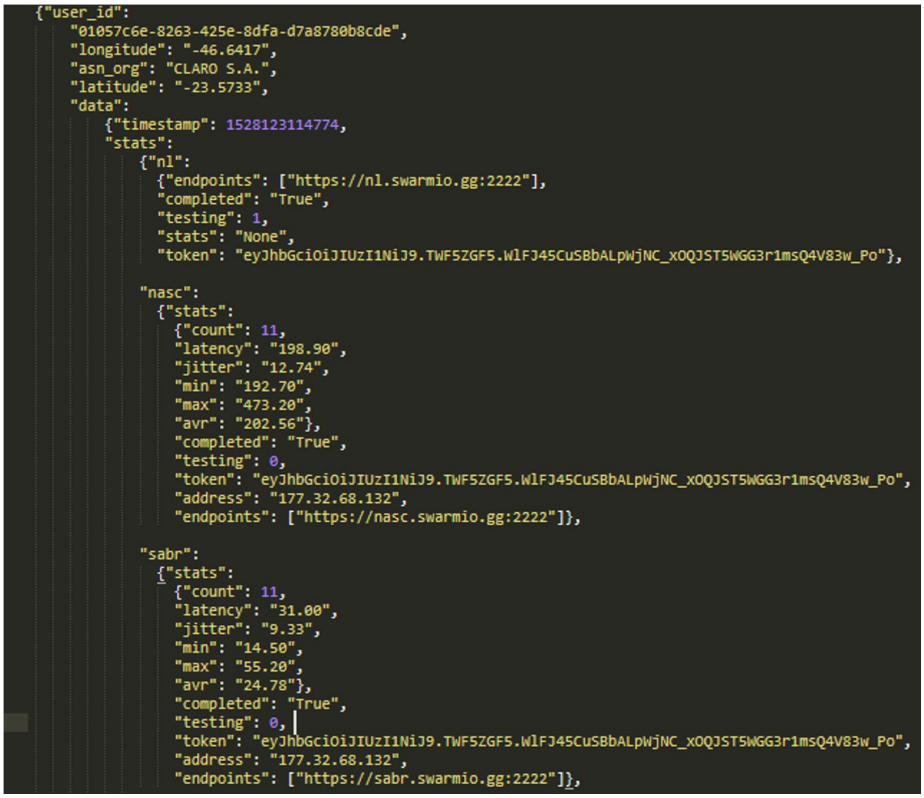
The user ID is a unique 32-character identifier that is generated for each player; for example, 5193b0e1-2412-4338-ac8d-6f519049aa77. The time of access is based on the Unix timestamp which is counted in seconds; for example, 1528484445170. Longitude and latitude are based on the geo-location of the player; for example, "longitude": "121.0409", "latitude": "14.5832". The Access Support Network is the ISP network in which the player is registered, for example Rogers Communications Canada Inc, Philippine Long Distance Telephone Company, AT&T Services Inc., etc.

After registering each player, a background JavaScript script was run in Swarmio's client software to obtain the latency measurements connecting to all of the servers. The script would query Swarmio's portal to retrieve a list of all servers. Then, it would cycle through each server and measure the RTT latency. It would then push the results back to Swarmio's central server for storage.

Each measurement consisted of sending 11 packets from the player to the server, and the following measurements were obtained (all in ms):

1. Median latency/delay (in the field: latency)
2. Delay jitter (in the field: jitter)
3. Minimum obtained delay (in the field: min)
4. Maximum obtained delay (in the field: max)
5. Average obtained delay (in the field: avr)

It should be noted that out of the 9 servers, only the 1st server ("nl") was used for testing the connection, and that can be noted from the field "testing" having the value of "1". Therefore, the value of "stats" for the first server will have no measurements. A sample entry in the main dataset can be seen in Figure 6.3.



```
{"user_id":
    "01057c6e-8263-425e-8dfa-d7a8780b8cde",
    "longitude": "-46.6417",
    "asn_org": "CLARO S.A.",
    "latitude": "-23.5733",
    "data":
        {"timestamp": 1528123114774,
        "stats":
            {"nl":
              {"endpoints": ["https://nl.swarmio.gg:2222"],
              "completed": "True",
              "testing": 1,
              "stats": "None",
              "token": "eyJhbGciOiJIUzI1NiJ9.TWF5ZGF5.WlFJ45CuSBbALpWjNC_xOQJST5WGG3r1msQ4V83w_Po"},

            "nasc":
              {"stats":
                {"count": 11,
                "latency": "198.90",
                "jitter": "12.74",
                "min": "192.70",
                "max": "473.20",
                "avr": "202.56"},
                "completed": "True",
                "testing": 0,
                "token": "eyJhbGciOiJIUzI1NiJ9.TWF5ZGF5.WlFJ45CuSBbALpWjNC_xOQJST5WGG3r1msQ4V83w_Po",
                "address": "177.32.68.132",
                "endpoints": ["https://nasc.swarmio.gg:2222"]},

            "sabr":
              {"stats":
                {"count": 11,
                "latency": "31.00",
                "jitter": "9.33",
                "min": "14.50",
                "max": "55.20",
                "avr": "24.78"},
                "completed": "True",
                "testing": 0,
                "token": "eyJhbGciOiJIUzI1NiJ9.TWF5ZGF5.WlFJ45CuSBbALpWjNC_xOQJST5WGG3r1msQ4V83w_Po",
                "address": "177.32.68.132",
                "endpoints": ["https://sabr.swarmio.gg:2222"]},
```


Figure 6.3: An example of one of the entries in the main dataset

6.2.2 Secondary Dataset

For the secondary dataset, we set up 11 different servers: 1 server owned by Swarmio Media in Toronto and 10 servers using the AWS cloud in the following locations:

1. North Virginia,
2. Ohio,
3. Northern California,
4. Oregon,
5. Montreal,

6. Brazil,
7. Singapore,
8. Mumbai,
9. Sydney, AU
10. Ireland

The same script as the main dataset was run in the Swarmio client software of 67 players. This time, each server sent 8 packets to each player, and only the average delay was recorded and stored.

The secondary dataset consists of the JSON file *secondary-dataset.json*, where the keys are the names of the servers, and the values contain a list of the delays to the 67 players. The players IPs are provided in order in a separate file *secondary-dataset-users.json*. It is also possible to reuse the code that was used to retrieve the measurements in the file *HostsUsersRTT.py* . The IP addresses of the 11 servers can also be accessed in the file *secondary-dataset-servers.json* where the key of the record will have the name of the server; for example "N Virginia", and the value will have the IP address of the server, as shown in Figure 6.4.

In contrast to the main dataset, the secondary dataset contains only the delay between the servers and the players whereas the main dataset has more information such as the geo-location and the ISP. This makes the secondary dataset more suitable for testing and verification due to having a single label with only 2 features (IP addresses and city names), while the main dataset contains more features and measurements suitable for training and inference.

6.3 Availability and Format (CC BY 4.0)

The dataset is protected with the Creative Commons Attribution Licence (CC BY 4.0), which is available for free and can be downloaded from `https://dx.doi.org/10.21227/jr75-0215` as long as it is strictly used for research and non-commercial purposes. At the time of writing this chapter, the size of both datasets together is less than 1 Megabyte.

6.4 Conclusion

While primarily intended for cloud gaming research, the dataset can also be used by researchers who are developing other networking applications and algorithms that incorporates network la-

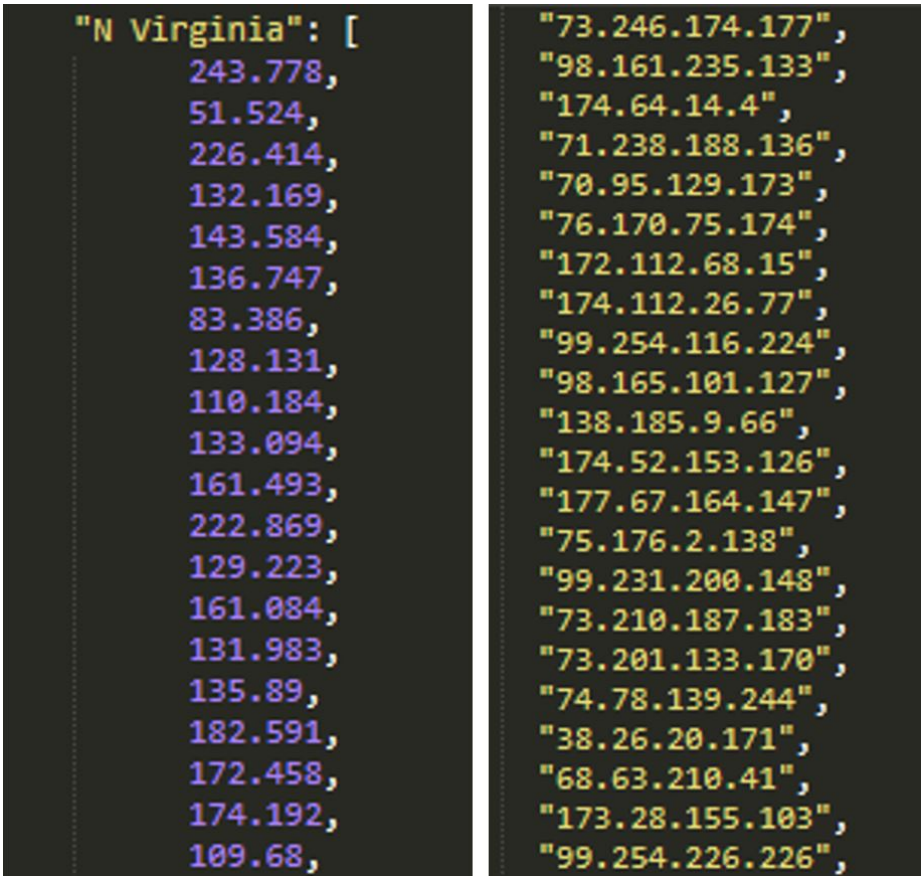


Figure 6.4: An example of one of the entries in the secondary dataset

tency, for example, delay prediction based on region/IP [78], geo-location estimation based on delay or IP address, as well as building algorithms for simulating edge-user allocation.

We presented a dataset of server-player RTT delays in an actual cloud gaming tournament. We hope researchers can use the dataset to improve the performance of cloud gaming server selection systems, as well as other algorithms based on network latency.

The next chapter will discuss our preliminary results in evaluating the proposed models in different settings. The insights extracted from the experiments are also discussed for the sake of improvements and proposing the future works.

Chapter 7

Performance Evaluation

In this chapter, the focus is on evaluating the proposed models in the previous chapters. We solve the EUA problem for cloud gaming using Reinforcement Learning to meet the latency thresholds of games while minimizing the variance of delay thereby achieving fairness among players. The RL solution uses ensembles and normalization to yield efficient results: testing with a real-world cloud gaming dataset shows our solution can outperform the state-of-the-art in fairness, especially when resources become scarce. The solution can also act as a heuristic when under-trained with carefully designed reward functions, allowing the system to quickly converge to a solution. We evaluate the proposed models in 2 different settings: An offline setting using the geo-distance as a distance function, and an online setting using the RTT delay as the distance function.

7.1 Offline Experiments - Geo-Distance as a Distance Function

In a network, latency between two nodes can be indicated by end-to-end delay, round trip time, number of hops, or geo-location distance (longitude and latitude). In this section, we use the latter as an example of a distance function, for two reasons: first, specifically for gaming, it has been shown that one of the important factors when choosing edge servers is their location since location implies the end-user demand [28]. Second, since there are potentially a large number of edge server and users, obtaining the actual end-to-end or round-trip delay between all users (n) and edge servers (m) will result in significant network traffic and has $O(n \times m)$ complexity. This complexity can be significantly reduced by bounding the number of edge servers via geo-location. However, it should be mentioned that our methodology itself can be applied to

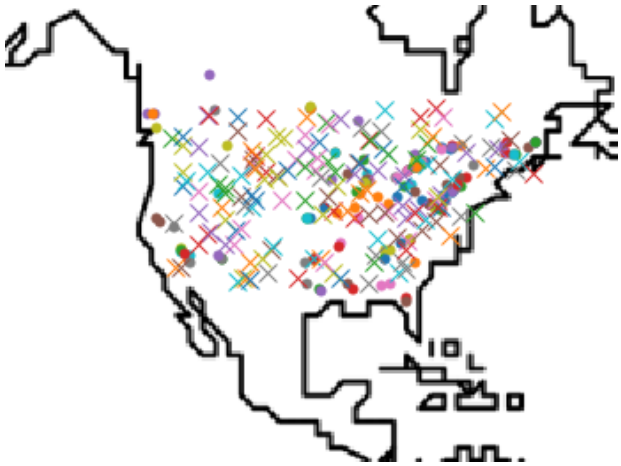


Figure 7.1: The distribution of the user nodes (Dot) with randomized servers (Cross) in North America

any indication of defined distance function such as end-to-end delay/latency, and is not exclusive to geo-location. In future experiments, we will evaluate this approach on different distance functions as well as different ranking/performance metrics.

7.1.1 Experiment Dataset

We used a subset of the main dataset of the CGCSDD which is collected from an actual cloud gaming tournament hosted by Swarmio Inc., in which 181 players participated and were connected to 9 different servers. During the game, we ran a script on each of Swarmio's servers in order to collect each user's delay, jitter, IP address, and location (longitude, latitude). In our simulation, we chose only the 153 North American players and placed the servers also in North America not only because the region has the greatest number of players, but also because evaluating the algorithm in a single region will help us see the effect of fair selection better given the limited number of users. Then we simulated servers randomly within the proximity of the chosen players: 153 players located in North America with 153 simulated servers. Figure 7.1 shows the distribution of the data collected and the simulated edge server nodes. We then ran various experiments, both for single session and grouped session, as explained next.

7.1.2 The Single Session Experiment

The single session scenario represents case 2 of Section 5.2.1; i.e., each player with or without a team assigned to the best server. Such a session can be massive, like the aforementioned EVE Online example. We took the 153 players in North America and assigned them to 153 servers,

which in theory has $153! = 2 \times 10^{269}$ number of matching possibilities. Throughout training, the RL hyper-parameters were adopted based on ranges of similar RL problems and then specified using hyper-parameter optimization through a grid search. As a result, we adopted the following values for the hyper-parameters of all the models and experiments: the learning rate $\alpha = 0.1$, the reward's discount factor $\gamma = 0.6$, the exploration factor $\epsilon = 0.1$, and the training duration $epochs = 100000$.

In order to compare the performance of our models with the state-of-the-art, we built 3 anchor methods representing existing methods which try to minimize latency but not the variance of latency, as well as a conventional heuristic method that specifically aims to reduce the variance of latency. These are explained next.

- **Anchor 1**, which is the most commonly used method in practice, matches the user to the closest server (based on geo-distance) that has capacity left for a new user.
- **Anchor 2** matches the user to the second closest available server. This approach will save the "best" servers to the users matched in later stages of the process.
- **Anchor 3** matches the first half of the users (76 users) to the servers that are in the 50^{th} percentile of distance (i.e. the 75^{th} closest server), while matching the other half of users to the closest servers.
- **Conventional heuristic method** directly aims to reduce the variance and works as follows:
    1. For the first user node, it finds the lowest and highest latencies between this node and any server, and takes the average of those two values as d_{conv}. Then, the node will be connected to any server with latency closest to d_{conv}.
    2. For each of the next nodes, it connects the node to a server with latency closest to the same d_{conv}; i.e., d_{conv} is calculated only once in step 1.

Results

As shown in Table 7.1, among **the Anchor methods**, it is clear that using the 2^{nd} and 75^{th} closest matching, as in Anchor 2 and Anchor 3 respectively, slightly reduces the variance while also increasing the average, with Anchor 2 having the best variance. The results in figure 7.3 highlights the unfairness in existing methods, represented by Anchor 1. The results for Anchor 2 and 3 were similar and are also shown in the figure. We can see that Anchor 2 follows a very similar pattern as Anchor 1, and also suffer from the same shortcoming: it can do a good job for about half the users, but then runs out of low-delay servers and does an exceptionally

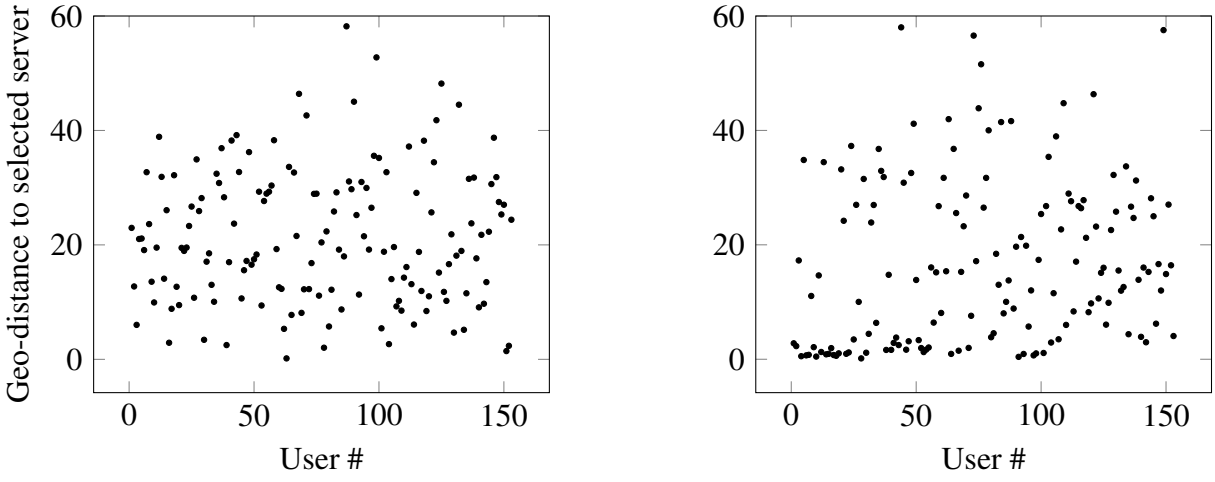
bad job with the remaining users. Anchor 3 experiments with the idea of deliberately not using the lowest-delay servers all at the beginning, but still does not manage to achieve good overall results.

The conventional algorithm's results depend on the first selected node which determines the value d_{conv}. Therefore, the algorithm was run 153 different times, each time starting with a different node. In order to report the results, we have selected 3 cases based on the value d_{conv}: Lowest, Median, and Highest d_{conv}. The results can also be seen in Table 7.1. Figure 7.4 shows the performance results of the conventional algorithm's results in 3 different cases: lowest D, median D, and highest D. In the lowest D case of figure 7.4 (a), the algorithm reduces the variance to the minimum for the first half of the users, but for the second half the values explode due to the unavailability of the low-latency servers. In the median and the highest D cases, the same scenario repeats. The only difference is that instead of the values increasing rapidly, the values become more and more sparse in figure 7.4(b), whereas in figure 7.4(c) the values start very high and then decrease.

Among **our proposed models**, we can see in Table 7.1 that model 6 has the least variance, being a significant 35% better than the best anchor methods, anchor 2. Compared to the conventional method, model 6 still has a better variance than the best of conventional methods, conv (highest), while offering a much better average latency. Figure 7.2 shows the performance of all 6 of our models. We can see that model 6 has the least variance. Since Model 6 is an aggregation of 4 different models and it selects one out of the 4 models every epoch, monitoring which model was used would help in validating the results of those models. The results showed that model 3 was used 139/153 times, while model 2 and model 4 were each used 7/153 times. This verifies that model 3 had the best Q-values in the majority of cases, while using the combination of different models has enhanced the results overall.

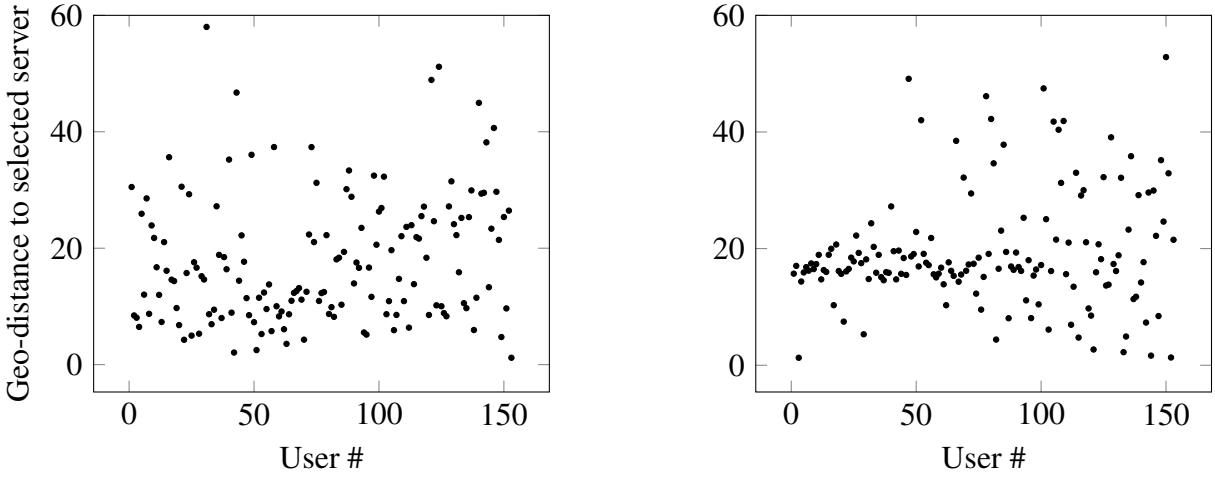
Discussion

The fact that the average latency of the anchor methods is better than that of QNetwork models is frivolous because any latency value less than the game's latency threshold is sufficient for a high-quality gaming experience. What really matters is that (1) no player has a latency more than the game's latency threshold, and (2) the variation of latency among players is minimized as much as viable. As we can see from figure 7.3, the anchor methods actually fail in both. Let us assume that the latency threshold is 50. We can see in figure 7.3(a) that for users connected to the system later in the algorithm, this latency threshold is violated. In comparison, QNetwork meets the latency threshold no matter at which position a user is connected to the system, as shown in figure 7.2(f). Similarly, we can see that the anchor method leads to widely different latency values between users, while this is much better controlled in QNetwork, leading to a



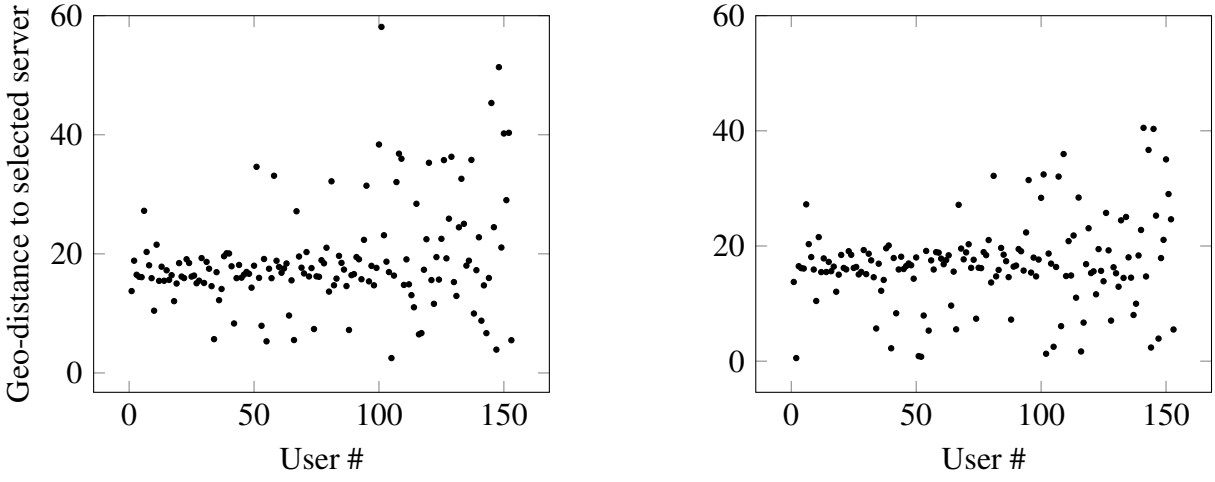
(a) Model 1 Performance

(b) Model 2 Performance



(c) Model 3 Performance

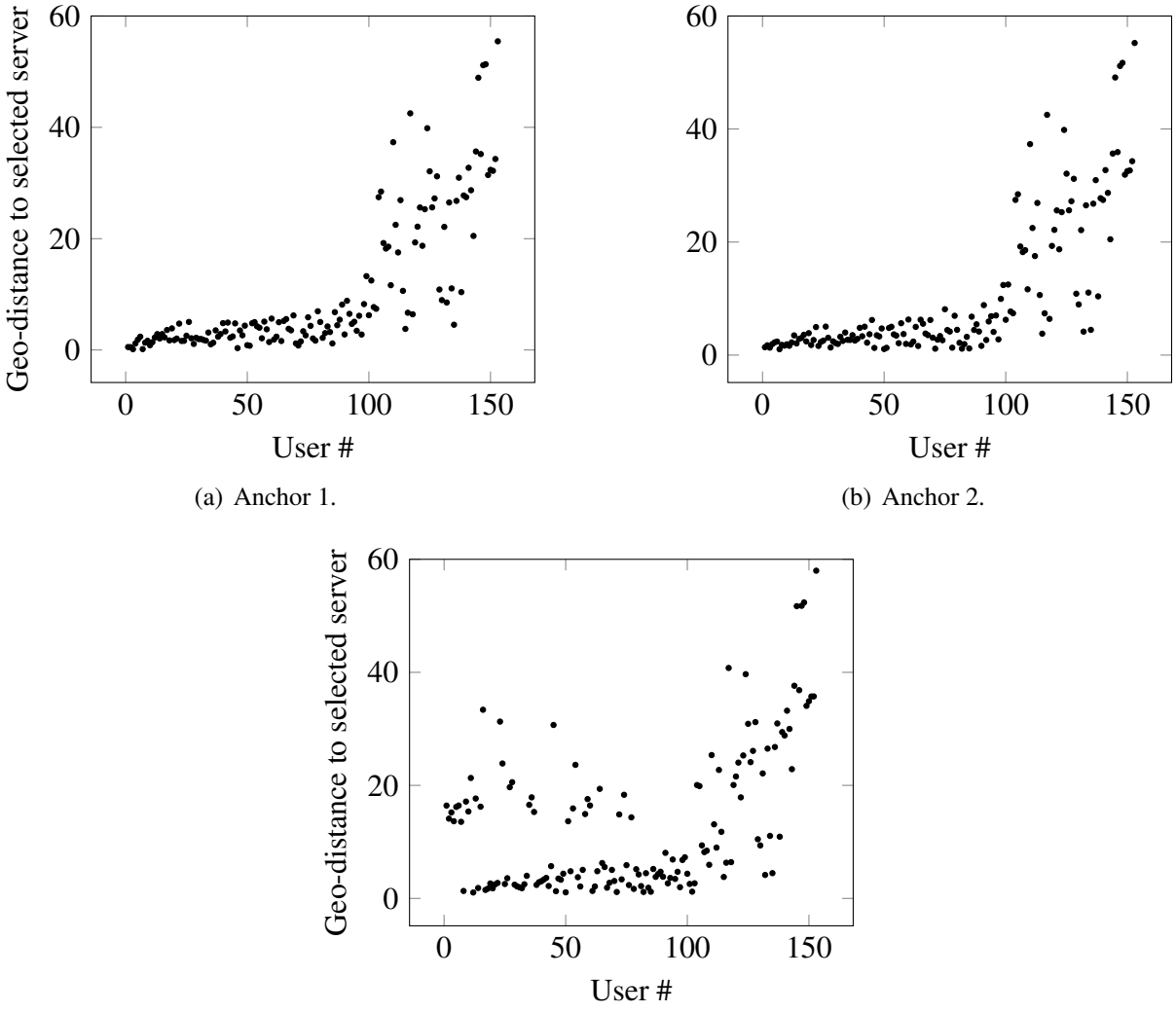
(d) Model 4 Performance



(e) Model 5 Performance

(f) Model 6 Performance

Figure 7.2: Performance of Models 1 to 6 on Single Session Experiments

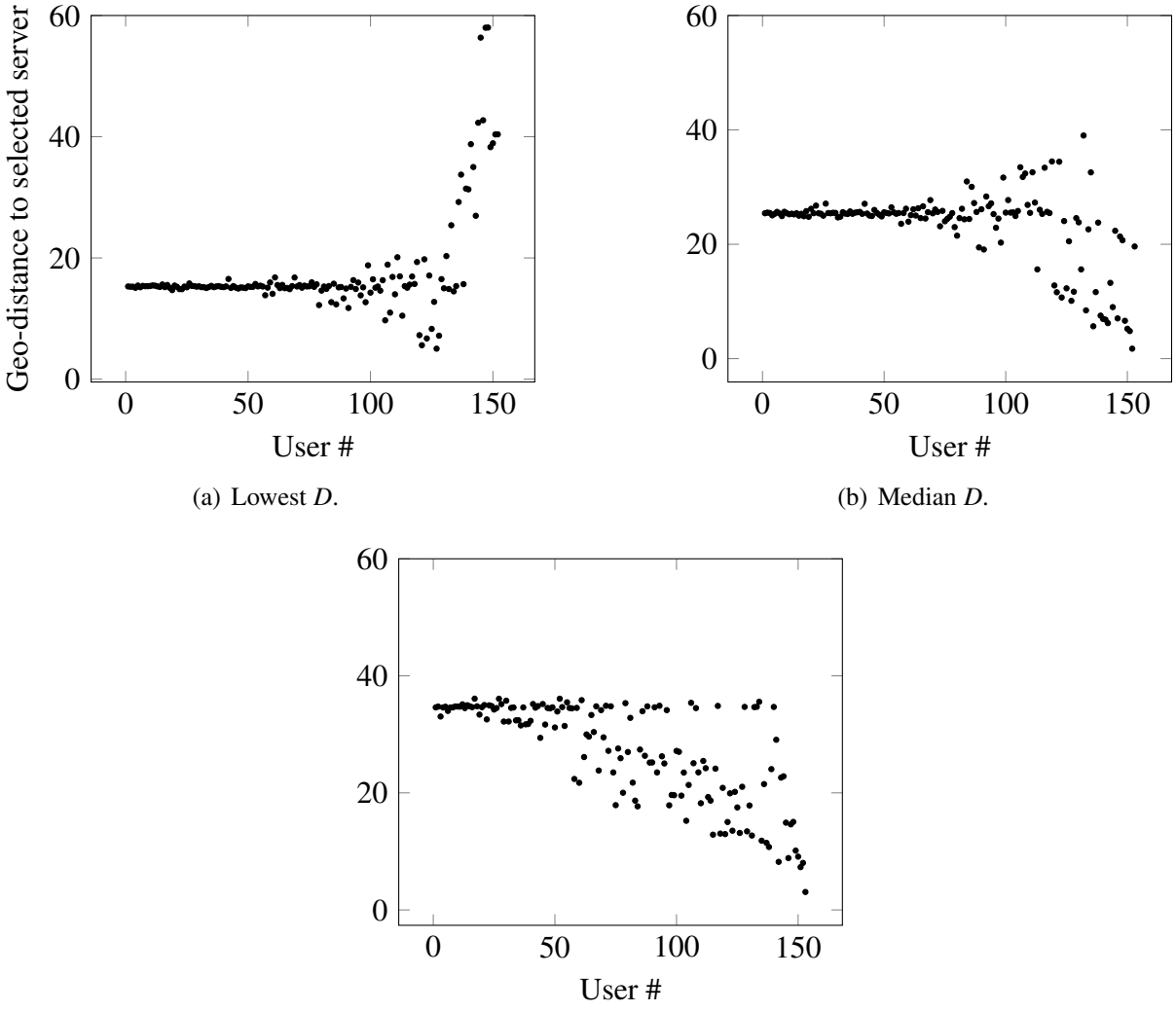


(a) Anchor 1.

(b) Anchor 2.

(c) Anchor 3.

Figure 7.3: Performance of Anchor methods



(a) Lowest D.

(b) Median D.

(c) Highest D.

Figure 7.4: Performance of Conventional Algorithm in 3 different cases

Method	avg	stdv
Anchor 1	**10.63**	12.67
Anchor 2	10.78	12.60
Anchor 3	13.13	12.62
Conv (lowest)	17.89	9.30
Conv (median)	23.38	8.92
Conv (highest)	27.20	8.75
Model 1	16.76	14.33
Model 2	19.22	11.95
Model 3	17.19	10.92
Model 4	19.35	9.83
Model 5	18.94	8.67
Model 6	17.40	**8.22**

Table 7.1: Models latency of single session

fairer gaming experience. The behaviours of these two methods can be explained as follows: Anchor 1 starts by assigning the first users to the available servers with the lowest distance. After a while, the capacity of these servers maxes out, and the system has no choice but to use servers farther away for later users. The conventional algorithm also acts in a similar way: as we can see in figure 7.4, the algorithm successfully reduces the variance for the first half of users, but for the second half the values explode due to the unavailability of closer servers. On the other hand, model 6 starts by assigning users to servers with a distance closer to the eventual average of 17.40; therefore, the variance of the values are lower.

Since Model 6 is an aggregation of 4 different models and it selects one out of the 4 models every epoch, monitoring which model was used would help validate those models. The results showed that model 3 was used 139/153 times, while model 2 and model 4 were each used 7/153 times. That verifies that model 3 had the best Q-values mostly, while using their combination has enhanced the results overall.

7.1.3 The Grouped Session Experiment

To simulate case 1 mentioned in Section 5.2.1; i.e., a team of players connecting to the same server, we also ran experiment for grouped sessions. An example for this is when a “red” team plays the game of Counter Strike against a “blue” team. Here, all players in team red are connected to one edge server while all players in team blue are connected to another edge server. This is only possible if all players in a given team are within the delay threshold of the server they are assigned to. If they are not, the scenario switches to single session.

If a grouped session is feasible, then our method will work as follows:

1. For each edge server, the group delay between that group and the server is calculated. Group delay is simply the average of all player-server delays for players in that group.

2. The delay matrix now consists of delay values between groups and edge servers. In comparison, the delay matrix in the single session scenario consisted of delay values between players and edge servers.

3. Our RL algorithm runs the same as before, using the groups' delay matrix.

We created 15 groups of 10 users each, with the distribution shows in figure 7.5.

Figure 7.5: The distribution of the grouped nodes (Star) with randomized servers (Cross) in North America.

Results

As we can see from Table 7.2, model 6 again outperforms all other models. It should be mentioned that the average and STDV numbers in the table represent individual users. This time, Model 6 selected Model 3 11/15 times, Model 2 3/15 times, and Model 4 1/15 time. Again, Model 3 had the best Q-values in most cases, while using the combination of different models has enhanced the results overall.

The results also confirm our choice of using min-max normalization in Model 6. As mentioned in the main paper, this normalization standardizes the Q-values of the first 4 models' tables between 0-1. By doing this, we would be able to compare the results of the 4 models fairly. The effect of this can be seen as the difference between Model 6 and Model 5, and we proposed it when we noticed that Model 5's results looked very similar to Model 4, since the latter has the lowest scale so its actions were always chosen without normalization. But, after the normalization, Model 6 is able to compare the 4 models fairly.

Model	avg	stdv
Model 1	**16.66**	17.08
Model 2	24.33	14.21
Model 3	23.86	11.06
Model 4	17.95	12.33
Model 5	22.91	8.70
Model 6	19.24	**8.68**

Table 7.2: Models latency of grouped session

7.1.4 Conclusion

In this section, we presented an RL solution to the multivariate multi-constraint problem of fair edge server selection for latency-sensitive applications such as cloud gaming. We demonstrated using real-world data that RL, specifically Quadruple Q-learning, can solve the problem fairer than existing approaches, with up-to a 35% reduction in the standard deviation of the user-server latency. While designing this solution, we also introduced action suppression, Quadruple Q-Learning, and normalization of the Q-values, leading to a more scalable and implementable RL system.

A number of topics can be investigated for future work. The first one is multiple parallel sessions. Currently each server is manually assigned to one gaming session. Automating this process becomes necessary for scalability. Furthermore, it will be interesting to investigate sharing of servers among multiple sessions; i.e., the leftover capacity of a server used in gaming session 1 can be used to run gaming session 2. This will make more efficient use of resources.

The second topic is support for latecomers: new users joining in the middle of a gaming session, or existing users moving from one gaming session to another. While such a business case does not currently apply to Swarmio, as a general cloud gaming case it is interesting.

For the RL part, we plan to adopt a tabular approach which will be more robust to handle infinite action and state spaces. Since the tabular RL can limit the approach's capability to scale for more complex scenarios, we will consider the use of approximation functions and estimators to handle larger action and state spaces. When a new user or a server gets introduced, by using an approximation function or a similarity function, the Q-values would be estimated and predicted which will give the node a quicker time to converge to find its optimal Q-values. Another possible approach is to use more sophisticated architectures such as DQN (Deep Q-Network) which eliminates the dependency on the table to store Q-values and replace that with a Neural Network to estimate the Q-values for the states. As the natural progression to the tabular case, we will investigate the use of such state-of-the-art architectures in our future works.

7.2 Online Experiments - RTT as a Distance Function

In the previous subsection and in our previous work [5], we evaluated our proposed a Reinforcement Learning (RL) solution to the EUA problem for Cloud Gaming systems. Cloud Gaming, a half-a-billion dollar industry in 2020 and expected to be $7.24 billion by 2027 [3] thanks to industry heavyweights such as Google Stadia, NVIDIA GeForce Now, Microsoft xCloud, PlayStation Now, and Amazon Luna, has unique properties that make it a challenging application: not only must the latency experienced by players be within acceptable thresholds, for example no more than 80 ms for First Person Shooter games [9], but also the variance of latency among the players in the same gaming session must be low, because players with higher latencies perform worse than those with lower latencies [12, 101] and could lose the game. Hence, optimal choice of the EUA problem has become a difficult challenge due to the high number of edge servers, especially in the 5G era with ultra low latency providing a larger coverage for a server. In fact it is known that multiuser synchronous collaboration apps, such as multiplayer gaming, suffer noticeably more from latency variance than from latency itself: a collaborative session with a low 10 msec latency that has jitter offers an experience as bad as a session with 200 msec latency but no jitter [79]. Therefore, an added challenge is the "fair" allocation of players to edge severs, such that the variation of latency among players is minimized. We considered these challenges and proposed a system in [5] that solves the EUA by both meeting the latency threshold and providing the lowest latency variation. Our RL solution uses ensembles and normalization to yield efficient results: testing with a real-world cloud gaming dataset shows our solution can outperform the state-of-the-art by up to 18.7% in fairness, especially when resources become scarce. Our solution can also act as a heuristic when under-trained with carefully designed reward functions, allowing the system to quickly converge to a solution.

In this section, we expand our system with the following additions:

- We consider not only delay and its variance, but also the edge server's capacity, and we show that the RL solution works especially well during resource scarcity;
- instead of geolocation, we use actual latency values from a real-world cloud gaming tournament;
- we present new ensemble and normalization techniques that yield better results;
- we show that even with under-training, the RL solution achieves excellent results, allowing the system to reach a heuristic quickly; and
- we evaluate our system against VSVBP [67] as a state-of-the-art EUA analytical solution.

We solve the EUA problem for cloud gaming using Reinforcement Learning to meet the latency thresholds of games while minimizing the variance of delay thereby achieving fairness among players.

7.2.1 Experiment Dataset

We use the secondary dataset in CGCSDD [4], which is collected from an actual cloud gaming tournament run by cloud gaming company Swarmio Inc. The dataset contains the delays between each of 67 players to each of 11 servers in Toronto, North Virginia, Ohio, Northern California, Oregon, Montreal, Brazil, Singapore, Mumbai, Sydney (Australia), and Ireland. The Toronto server was a rendez-vous server handling incoming join requests, so we are left with 10 servers. The dataset contains IP addresses of the players and servers, and we use ip-api.com to extract more information, such as geographical locations.

7.2.2 Experimental Setup

We ran experiments on a Windows machine with Intel Core i9-9900K processor 3.6GHz and 64.0 GB RAM. Our RL models were implemented in Python, while the VSVBP LGP model was solved using IBM's ILOG CPLEX. We compared our method against the following baselines: random assignment, greedy assignment, VSVBP, and a conventional algorithm that reduces STDV. These are described next:

- Random Assignment: Each user is allocated to a random server if that server has sufficient remaining capacity.
- Greedy Assignment: Each user is allocated to a server that has the least delay to that user and sufficient remaining capacity.
- VSVBP [67]: as described earlier.
- Conventional heuristic: Specifically aims to minimize the variance of latency. For the first user, it finds the average delay between this user and all servers in d_{conv}. For the consequent users, it connects the user to a server with a delay closest to this average.

7.2.3 Experiment parameters

Throughout training, hyper-parameters were tuned using optimization through grid search, resulting the following values: the learning rate $\alpha = 0.1$, the reward's discount factor $\gamma = 0.6$, the exploration factor $\epsilon = 0.1$, and the training duration epochs $E = 100000, 1000$.

Server capacity

With 10 servers and 67 users, a server's capacity C had the values:

- $C = 10$, handling 100 users which is more than 67, representing a highly available service.
- $C = 7$, handling 70 users which is almost equal to 67, representing a good availability of the service.
- $C = 5$, handling 50 users which is less than 67, representing a risky availability of service.
- $C = 3$, handling 30 users which is significantly less than 67, representing bad service availability.

Number of episodes

In RL, the number of episodes E determines how long the models should be trained. A single episode represents a single pass through the initial state all the way to a terminal state; i.e., performing a single scenario of matching servers and users until either all users are matched or until there is no capacity left in the servers. We used 2 values:

- $E = 100K$, which is a standard value widely adopted by existing work.
- $E = 1K$, which tests how well the models perform as heuristics with significantly fewer episodes.

7.2.4 Performance metrics

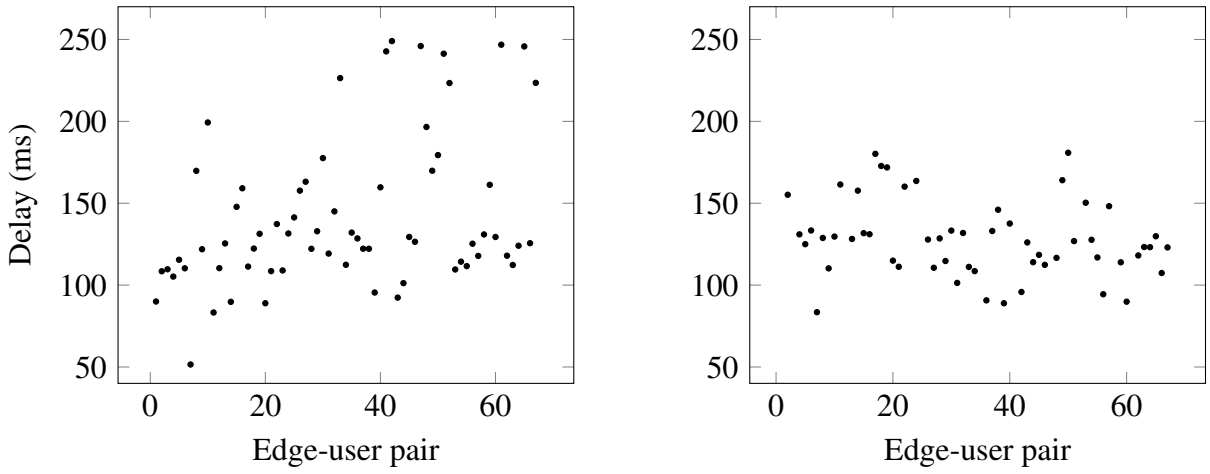
The models are compared against the baselines in terms of:

1. the average delay among all the edge-user allocations;
2. the STDV of the set of delays, representing fairness.

7.2.5 Results and Discussion

We varied one parameter at a time, keeping the others fixed to observe the each parameter's impact. Figure 7.6 illustrates an example of the effect of using Model 6 and its difference with the

greedy approach that is commonly used. It is noticeable that in the greedy approach the distribution of delay between the matched edge-user pairs has an increasing pattern, because allocating the best servers to the early users leaves the later users with whatever server is available, which might not meet the latency threshold anymore. On the other hand, Model 6 matches pairs closer to an eventual global average, providing much better fairness.



(a) Greedy.

(b) Model 6.

Figure 7.6: Model 6 vs. the greedy approach with $C = 10, E = 100K$. The x-axis represents the edge-user pairs, while the y-axis represents the delay between the pairs in ms.

Tables 7.3, 7.4, 7.5, and 7.6 show the results of the experiments on capacities: $C = 10$, $C = 7$, $C = 5$, and $C = 3$, respectfully. The performance with the different capacities is depicted in Figures 7.7, 7.8, 7.9, and 7.10 where sub-figure (a) shows the average delay and (b) shows the STDV of delays.

For $C = 10$, $E = 100K$, the first column in table 7.3 demonstrates that our proposed models, especially models 4 and 5, have slightly lower STDV (-1.57%) compared to greedy and random approaches, but not lower than VSVBP. However, that comes with an expected slight increase in the average delay (+1.6%) compared to the greedy allocation. We can also see that all of our proposed models have better STDV than the baselines. Model 6 in most cases has improved the results which justifies the use of an ensemble. Among Model 6's sub-models, normalizing per state row (6.1 and 6.4) are generally doing better than normalizing the whole table or mixing both techniques.

For $C = 7$, $E = 100K$, the second column in table 7.4 shows how applying fairness starts to be more visible with resource scarcity. It is noticeable that the STDV difference has slightly increased, especially with Model 6.1 showing an improvement of 1.96% in the STDV of the delays.

For $C = 5$, $E = 100K$, the third column in table 7.5 illustrates results while having less resources than the demand. Our models show a better STDV performance of up to 11.9% in Model 6.1 compared to the baseline models.

Lastly, for $C = 3$, $E = 100K$, figure 7.10 crystallizes the need of fairness-awareness in resource-scarce situations. Our models, especially Model 6.5 and 6.6 were able to decrease STDV by as much as 18.7%.

Comparing $E = 100K$ to $E = 1K$ in tables 7.3, 7.4, 7.5, and 7.6, and figures 7.7, 7.8, 7.9, and 7.10, it can be seen that most of the models with $E = 100K$ are not that far from $E = 1K$. Therefore, even with less episodes of training, the models can still outperform baseline approaches in terms of fairness. This can be explained due to carefully designed reward functions acting as heuristics during allocation although the RL agent is not fully trained yet. Since the reward functions produce Q-values for that matching, this reward is very similar to a cost that the RL agent keeps a history of. This would ease the implementation of this system in the real world because in addition to pre-defined system constraints, the reward functions can act in parallel as cost functions helping the agent to reach a stable state faster.

7.2.6 Conclusion

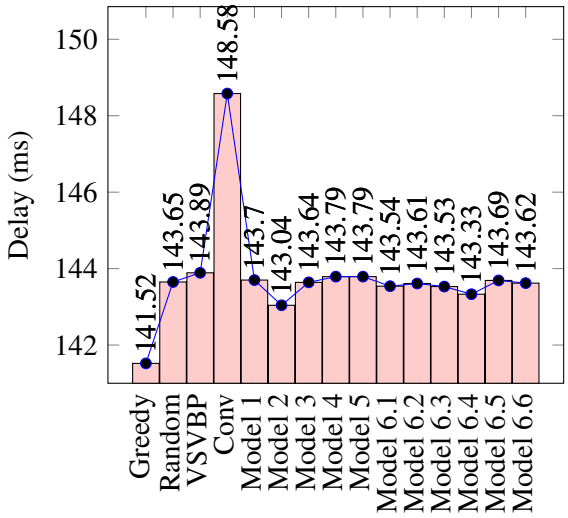
For the multivariate multi-constraint EUA problem, we presented an RL solution that not only considers the delays and the servers' capacities but also the variance of delays. The proposed solution shows improvements in fairness up to 18.7% compared to existing solutions especially during resource scarcity. Additionally, the RL solution can act as a heuristic algorithm even when it is not fully trained.

Model (C=10)	E=100k		E=1k	
	avg	stdv	avg	stdv
Greedy	**141.52**	46.56	**141.52**	46.56
Random	143.65	46.23	143.65	46.23
VSVBP	143.89	**45.64**	143.89	**45.64**
Conv.	148.58	**45.64**	148.58	**45.64**
Model 1	143.70	45.93	143.74	46.05
Model 2	143.04	46.21	143.26	46.50
Model 3	143.64	46.38	144.71	45.87
Model 4	143.79	**45.83**	143.81	46.08
Model 5	143.79	**45.83**	143.81	46.08
Model 6.1	143.54	46.01	143.74	**45.66**
Model 6.2	143.61	46.33	144.73	45.81
Model 6.3	143.53	46.13	143.26	46.61
Model 6.4	143.33	46.12	143.79	46.05
Model 6.5	143.69	46.28	144.27	46.05
Model 6.6	143.62	46.33	144.20	46.14

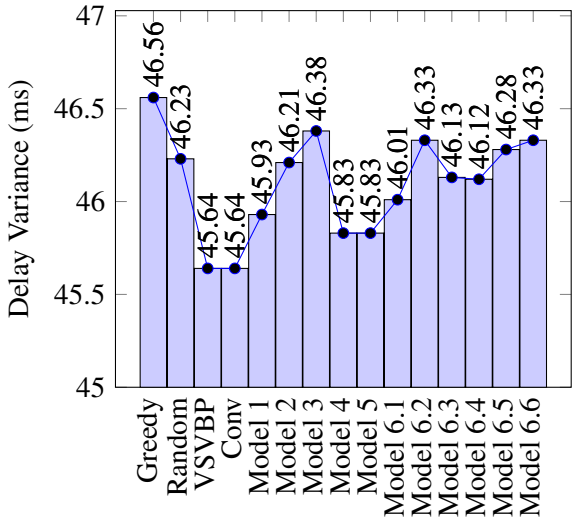
Table 7.3: Models' delay (ms) with C=10

Model (C=7)	E=100k		E=1k	
	avg	stdv	avg	stdv
Greedy	**141.77**	46.59	**141.77**	46.59
Random	145.31	46.28	145.31	46.28
VSVBP	143.60	45.88	143.60	45.88
Conv.	148.17	45.86	148.17	45.86
Model 1	143.44	46.66	143.63	45.78
Model 2	142.94	46.59	143.39	46.90
Model 3	143.85	45.91	144.14	45.68
Model 4	144.62	46.85	144.93	46.18
Model 5	144.62	46.85	144.92	46.04
Model 6.1	143.41	**45.68**	143.98	45.75
Model 6.2	143.85	46.01	144.11	45.67
Model 6.3	143.69	47.02	144.14	46.31
Model 6.4	143.19	45.96	143.75	45.95
Model 6.5	143.85	46.18	144.27	**45.58**
Model 6.6	143.80	46.22	144.09	45.66

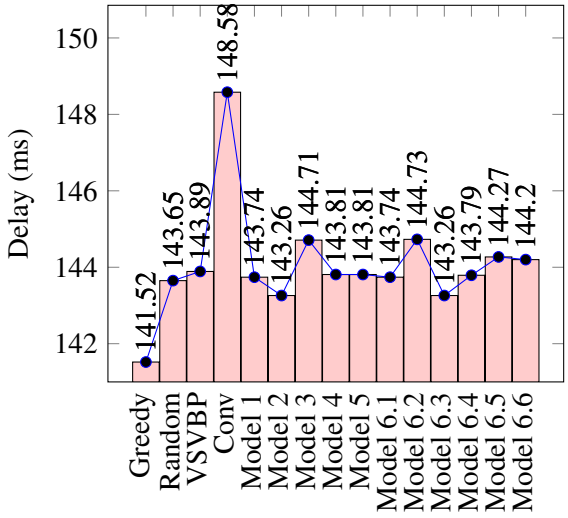
Table 7.4: Models' delay (ms) with C=7



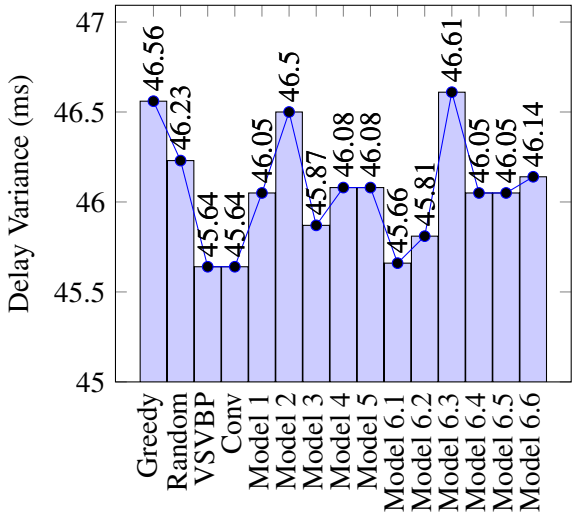
(a) The Average Delay for each model at $E = 100K$



(b) The Standard Deviation of Delays for each model at $E = 100K$

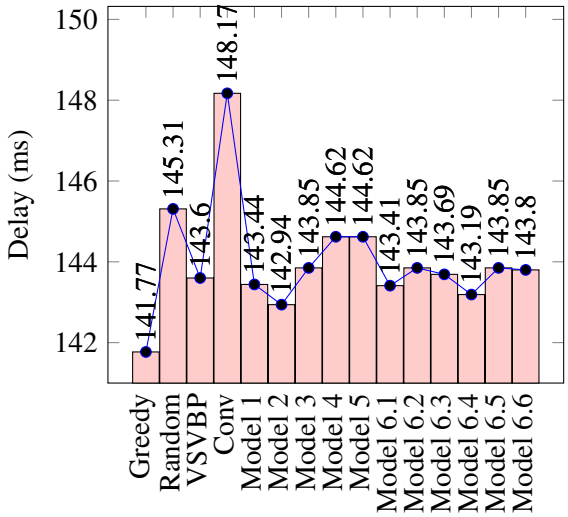


(c) The Average Delay for each model at $E = 1K$

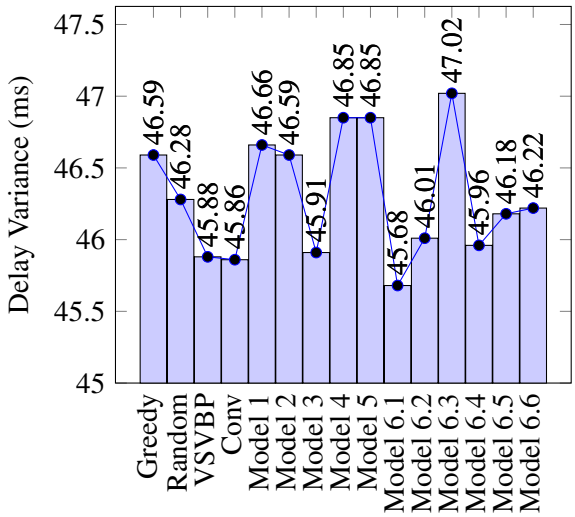


(d) The Standard Deviation of Delays for each model at $E = 1K$

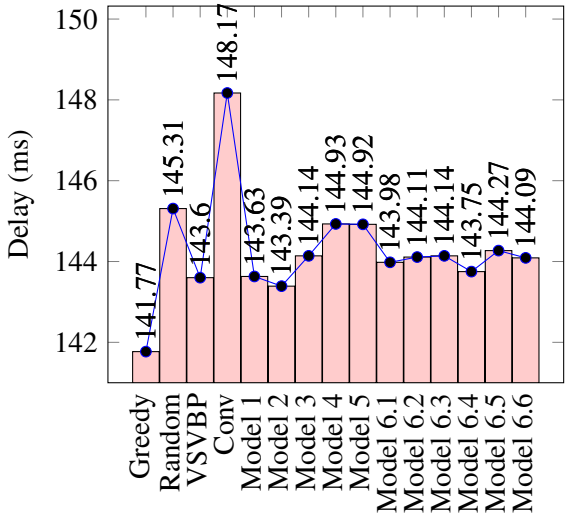
Figure 7.7: The results of using our approach on Capacity (C=10) and Episodes (E = 100K, and 1K). The top left and bottom left graph (in red) represent the average delays of the models, and the top right and bottom right graph (in blue) illustrate the delay variance.



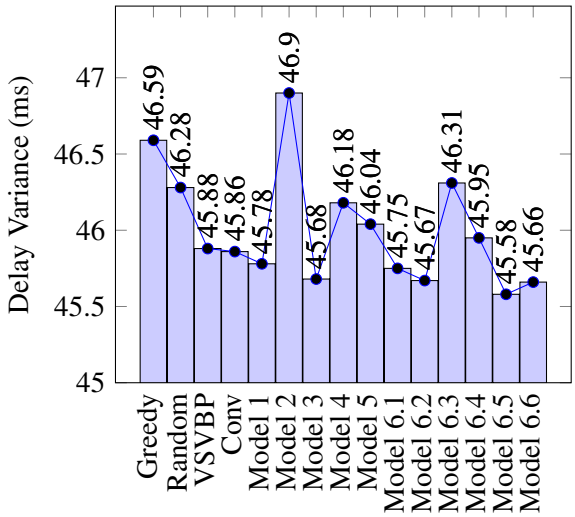
(a) The Average Delay for each model at $E = 100K$



(b) The Standard Deviation of Delays for each model at $E = 100K$



(c) The Average Delay for each model at $E = 1K$



(d) The Standard Deviation of Delays for each model at $E = 1K$

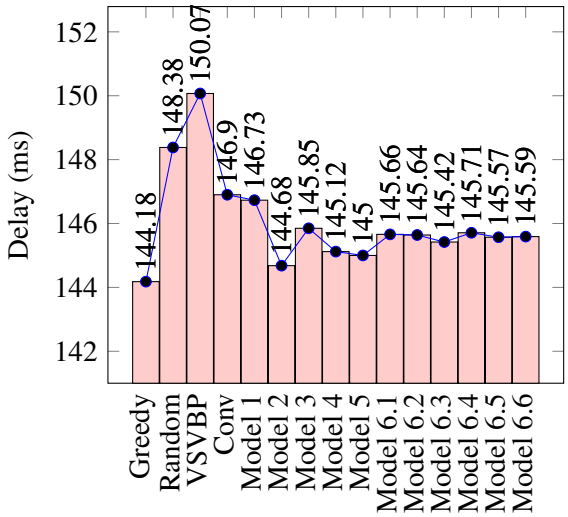
Figure 7.8: The results of using our approach on Capacity (C=7) and Episodes (E = 100K, and 1K). The top left and bottom left graph (in red) represent the average delays of the models, and the top right and bottom right graph (in blue) illustrate the delay variance.

Model (C=5)	E=100k		E=1k	
	avg	stdv	avg	stdv
Greedy	**144.18**	52.26	**144.18**	52.26
Random	148.38	48.77	148.38	48.77
VSVBP	150.07	47.64	150.07	47.64
Conv.	146.90	46.33	146.90	46.33
Model 1	146.73	46.78	145.04	46.40
Model 2	144.68	46.60	144.54	46.47
Model 3	145.85	47.06	146.09	46.37
Model 4	145.12	46.40	145.25	45.94
Model 5	145.00	46.50	145.28	45.96
Model 6.1	145.66	**46.04**	145.41	46.18
Model 6.2	145.64	46.26	146.37	46.39
Model 6.3	145.42	46.28	145.64	46.83
Model 6.4	145.71	46.14	145.40	**45.91**
Model 6.5	145.57	46.27	146.36	46.35
Model 6.6	145.59	46.19	146.47	46.27

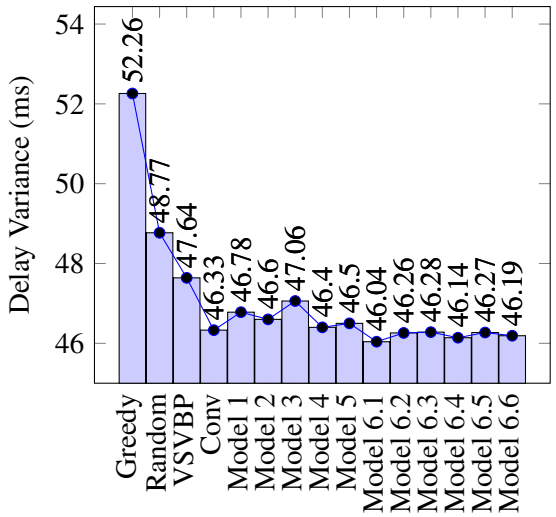
Table 7.5: Models' delay (ms) with C=5

Model (C=3)	E=100k		E=1k	
	avg	stdv	avg	stdv
Greedy	149.90	55.66	149.90	55.66
Random	153.02	48.91	153.02	48.91
VSVBP	155.10	52.93	155.10	52.93
Conv.	145.45	45.86	145.45	45.86
Model 1	148.84	47.28	149.97	46.05
Model 2	149.84	46.08	149.70	45.90
Model 3	148.56	45.67	148.03	45.81
Model 4	148.43	45.78	148.09	45.97
Model 5	148.43	45.78	148.09	45.97
Model 6.1	**148.05**	45.76	148.67	45.65
Model 6.2	149.08	46.40	148.28	45.91
Model 6.3	149.83	46.15	149.02	**45.42**
Model 6.4	149.05	45.57	147.97	45.73
Model 6.5	148.29	**45.25**	149.11	45.65
Model 6.6	148.29	**45.25**	**147.94**	46.06

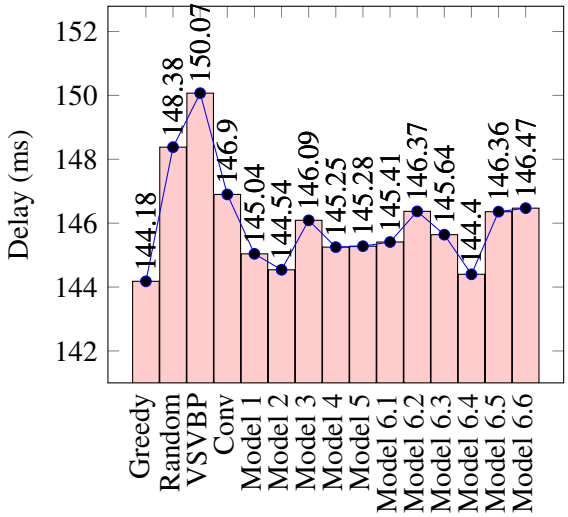
Table 7.6: Models' delay (ms) with C=3



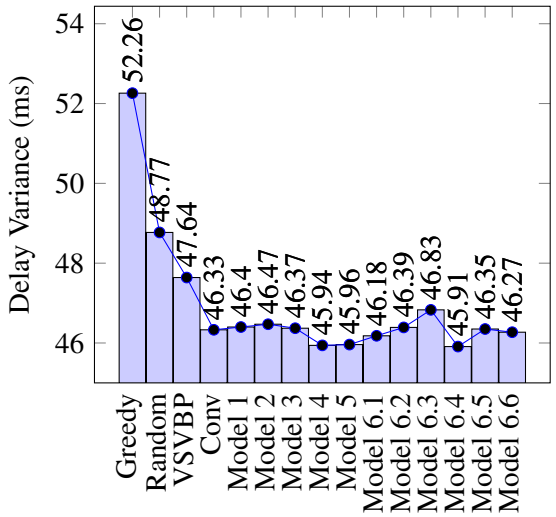
(a) The Average Delay for each model at $E = 100K$



(b) The Standard Deviation of Delays for each model at $E = 100K$

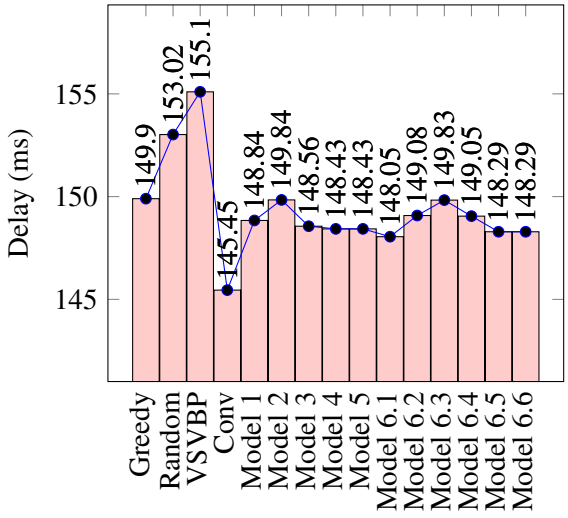


(c) The Average Delay for each model at $E = 1K$

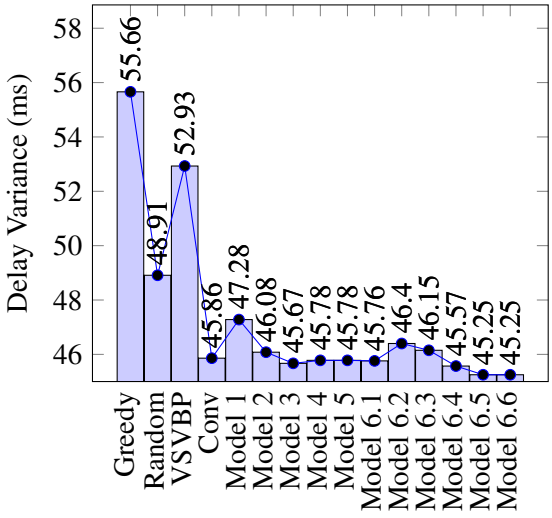


(d) The Standard Deviation of Delays for each model at $E = 1K$

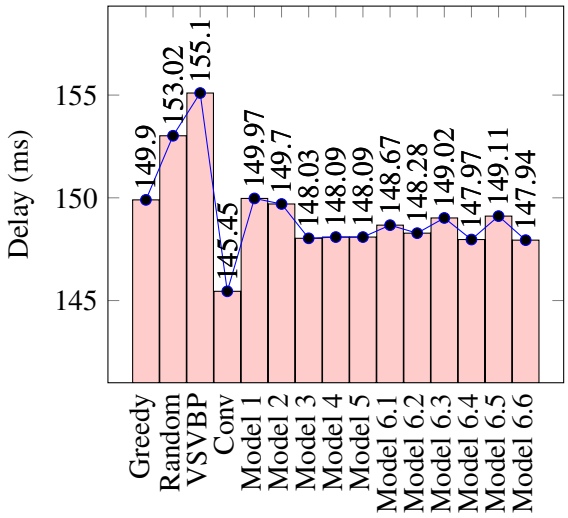
Figure 7.9: The results of using our approach on Capacity (C=5) and Episodes (E = 100K, and 1K). The top left and bottom left graph (in red) represent the average delays of the models, and the top right and bottom right graph (in blue) illustrate the delay variance.



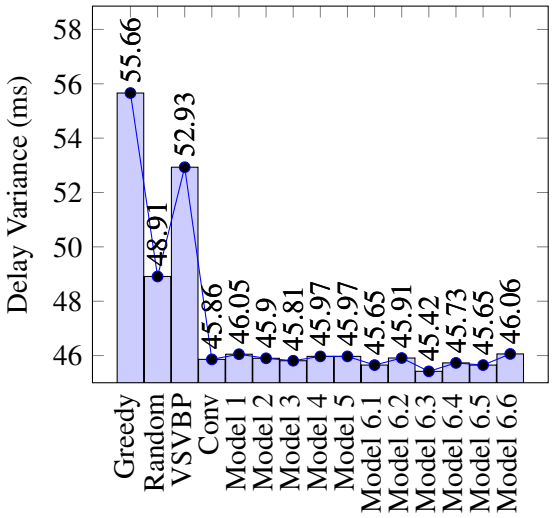
(a) The Average Delay for each model at $E = 100K$



(b) The Standard Deviation of Delays for each model at $E = 100K$



(c) The Average Delay for each model at $E = 1K$



(d) The Standard Deviation of Delays for each model at $E = 1K$

Figure 7.10: The results of using our approach on Capacity (C=3) and Episodes (E = 100K, and 1K). The top left and bottom left graph (in red) represent the average delays of the models, and the top right and bottom right graph (in blue) illustrate the delay variance.

Chapter 8

Conclusion & Future Work

8.1 Conclusion

In this thesis, we presented an RL solution to the multivariate multi-constraint problem of fair edge server selection (a.k.a the EUA problem) for latency-sensitive applications such as cloud gaming. We presented an RL solution that not only considers the delays and the servers' capacities but also the variance of delays. We demonstrated using real-world data that RL, specifically Quadruple Q-learning, can solve the problem fairer than existing approaches, with up-to a 35% reduction in the standard deviation of the user-server latency while using the geo-distance, and it shows improvements in fairness up to 18.7% compared to existing solutions using the RTT delay especially during resource scarcity. Additionally, the RL solution can act as a heuristic algorithm even when it is not fully trained.

While designing this solution, we also introduced action suppression, Quadruple Q-Learning, and normalization of the Q-values, leading to a more scalable and implementable RL system.

For the RL part, we plan to adopt a tabular approach which will be more robust to handle infinite action and state spaces. Since the tabular RL can limit the approach's capability to scale for more complex scenarios, we will consider the use of approximation functions and estimators to handle larger action and state spaces. When a new user or a server gets introduced, by using an approximation function or a similarity function, the Q-values would be estimated and predicted which will give the node a quicker time to converge to find its optimal Q-values. Another possible approach is to use more sophisticated architectures such as DQN (Deep Q-Network) which eliminates the dependency on the table to store Q-values and replace that with a Neural Network to estimate the Q-values for the states. As the natural progression to the tabular case, we will investigate the use of such state-of-the-art architectures in our future works.

8.2 Future Work

After reviewing the promising results of this work, in this section, we outline the remaining work required to complete this research along with a few possible extensions that would improve the system and increase the functionality. This thesis mostly sought to introduce the issue of fair server selection in edge computing and address the problem from a simple reinforcement learning prospective. The approached that was discussed showed a promising results for a selected application that was trained offline. Scaling the application and deploying it in a live environment could possibly need some modifications. Moreover, the approach could be applied to other AI techniques to with more sophisticated state representations. For example, the recent advances in RL and more sophisticated algorithms could scale to applications with more complex states and rewards.

A number of topics can be investigated for future work. The first one is multiple parallel sessions. Currently each server is manually assigned to one gaming session. Automating this process becomes necessary for scalability. Furthermore, it will be interesting to investigate sharing of servers among multiple sessions; i.e., the leftover capacity of a server used in gaming session 1 can be used to run gaming session 2. This will make more efficient use of resources.

The second topic is support for latecomers: new users joining in the middle of a gaming session, or existing users moving from one gaming session to another. While such a business case does not currently apply to Swarmio, as a general cloud gaming case it is interesting. Next, we discuss the possible further extensions and additions to our work.

8.2.1 Deployment on Swarmio's esports live environment

We intend to experiment the deployment of our approach on a live and running environment. Since the system was only trained on a collected data offline, the accuracy and the performance of the system could lead to different observations since the behavior of the environment is unexpected unlike training on a pre-recorded dataset. Since the environment we are dealing with here is the edge network, the distribution of data will differ carrying more variations over time. The system should be resilient to those changing and using the different variations to its advantage in order to produce more efficient outcomes in the future. In our evaluation, we have used servers with capacities ($c = 10, 7, 5, and 3$) for simplicity, while in the live environment we will be experimenting a variable number capacities per servers and we will evaluate the systems on different capacities. In addition to all above, we intend to experiment the use of live end-to-end delay measurements for the environment in addition to the geo-distance.

With the advantage of training on-the-go that RL has, this is easy to overcome. The algorithm

is expected to be elastic to the changes in the data distributions since the training should be continuous and on-going as long as data is following and the network is operating. Leveraging the knowledge that was acquired by the trained model already, transfer learning is a great asset to be used here. It would ease the convergence of the models even if they were trained on a totally different setting. Finally, a fine-tuning stage might be needed in order to fit the new environment better. Similar the closed-loop system, deploying the system live with a pipeline that has an tolerable error which can be measured by how good the selection was done. When the performance of the model degrades after that certain error margin, the system should flag the model and deploy a new training version of the model.

8.2.2 Evaluating on KING dataset

In order to evaluate the developed techniques effectively, we intend to evaluate the approaches on a massive dataset such as the KING dataset. As described in [53], KING has data from 1740 Internet DNS servers and around 100 million pairwise RTTs in total. The evaluation would increase our understanding of the scalablility of our approach over a massive dataset. Such insights can be used in order to propose more effective solutions compromising less sensitive parameters.

8.2.3 Handling dynamic ranges of state space

In cases where the actions and states are massive, the use of approximation functions can be utilized. An example can be the KING dataset with a massive number of nodes where in this case, the states need to be replaced by representations and a higher level features before it gets passed to the RL algorithm. We intend to implement the approach on few of the most recent advances in RL.

Deep Q-network agents (DQN)

After the impressive success by [77, 100] in developing a model to play games from row images, the use of Deep Reinforcement Learning kept rising. The authors were able to develop a deep learning model to learn policies from highly dimensional data using RL. The models are based on CNN where it take the row images as inputs and outputs a value function estimating future rewards. Their work bridges the gap between high-dimensional sensory inputs and actions, resulting in DQN agent capable of learning to excel at challenging tasks.

The approach can be adopted for massive networking applications where the number of possible states and actions of the network are uncountable and cannot be handles by tabular cases.

Having our same set of reward functions and developed techniques, a combination between them would not only improve the results of our problem but also have the potential to solve more complicated problems that has to do with fairness.

Contrastive Unsupervised Representations for RL (CURL)

In the work of Srinivas et al. [71], they introduced contrastive unsupervised representations (CURL) that would work for any general RL algorithm. CURL uses encoders and momentum encoders to introduce those meaningful representations of the environment's observations, which means extracting useful information from the observation and the state of the environment rather than using the whole state in the RL algorithm. They have used the contrastive loss for encoding the observations and extracting high-level features from raw pixels using constrastive learning. Similar to DQN, CURL tries to abstract high-level features and representations from the input data before it is passed to the RL algorithm. The difference here is that the constastive learning approach teaches the encoding network the positive and negative predictions of the states resulting in more meaningful and reliable embeddings. Similarly, we intend to use CURL and apply the same approaches we have developed in order to obtain a more resilient agent that can generalize to more general tasks.

8.2.4 Other possible extensions

The work by [90] introduced the attention mechanism that revolutionized many applications and inspired many other architectures (mainly targeted at the language models). For areas like Natural Language Processing (NLP), the need for model with the ability to "remember" and "relate" words to each other to an unknown number of steps. The attention mechanism filled in the gap where it constructs a heat map of how each term relates to the other term in a corpus of text. this allows each point of data at any point of time to "attent" to any other point. The same concept can be applied to computer networks and especially edge computing. The network should be selective on what to keep out of all the types of data, insights, features, and other measurements in the network. The attention mechanism will scale networking applications and would result in having a remarkable effect on the storage needed for deploying smarter controllers.

8.2.5 Future plan

As future work on the proposed algorithm and system architecture designed to improve the QoE of cloud gaming systems with edge computing, we plan to implement the following steps:

1. Design and determine the detailed duties of the components needed to be updated for the live deployment and testing of the algorithm.

2. Evaluating the system with the different normalization functions used on our dataset and pn the KING dataset.

3. Adapting to dynamic ranges of state space by using DQN and CURL techniques to solve the scalability problem.

4. Revisit the optimization method presented in Chapter 4. This task requires the following steps:

   (a) A new objective function is needed that provides a more realistic model capturing the new requirements of the system. To develop this function, we will propose a preference model that considers the impact of different factors contributing to the server selection decision such as the QoE and ranking.

   (b) Constraint definitions that describe and formulate the system limitations are required. For the optimization problem, this translates into a set of equations and inequalities that we call constraints. These constraints consist of the regions that defines the performance limits of the system.

   (c) A set of decision variables must be defined based on considered and desired outputs. Those variables appear in the objective function and must be adjusted to satisfy the constraints. This can be accomplished by having multiple instances of variable values, leading to a feasible region.

   (d) Evaluate the performance of the primal optimization function relative to the designed models in terms of computation requirements, accuracy, and flexibility.

5. Using 5G network slicing in edge computing (after getting the business requirements from the industry partner, Swarmio) and leveraging the attention mechanism in the action elimination phase.

References

[1] *New Kind of Networking (NKN)*, 2017 (Accessed: 2019-03-28). https://www.nkn.org/.

[2] Most played games ever, ranked by peak concurrent players. Feb 2018.

[3] Cloud gaming market analysis report by type, by device, by gamer type, by region and segment forecasts from 2021 to 2027. *Million In$ights, Report ID: MN17620203*, Mar 2021.

[4] Alaa Eddin Alchalabi and Shervin Shirmohammadi. CGCSDD: Cloud gaming client-server delay dataset. IEEE Dataport, August 2021, URL: https://dx.doi.org/10.21227/jr75-0215.

[5] Alaa Eddin Alchalabi, Shervin Shirmohammadi, Shady Mohammed, Sorin Stoian, and Karthigesu Vijayasuganthan. Fair server selection in edge computing with q-value-normalized action-suppressed quadruple q-learning. *IEEE Transactions on Artificial Intelligence*, page 9 pages, August, 2021.

[6] Alaa Eddin Alchalabi, Shervin Shirmohammadi, and Shady A. Mohammed. Qnetwork: Ai-assisted networking for hybrid cloud gaming. In *2019 IEEE International Symposium on Measurements Networking (M&N)*, pages 1–6, July 2019.

[7] Alaa Eddin Alchalabi, Shervin Shirmohammadi, Sorin Stoian, and Karthigesu Vijayasuganthan. An edge-user allocation solution for cloud gaming: Achieving fairness with reinforcement learning. *Submitted to the IEEE Internet Computing*, page 9 pages, August, 2021.

[8] Maryam Amiri, Hussein Al Osman, Shervin Shirmohammadi, and Maha Abdallah. An sdn controller for delay and jitter reduction in cloud gaming. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, page 1043–1046. Association for Computing Machinery, 2015.

[9] Maryam Amiri, Hussein Al Osman, and Shervin Shirmohammadi. Resource optimization through hierarchical sdn-enabled inter data center network for cloud gaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*, page 166–177, Istanbul, Turkey, 2020. ACM.

[10] Maryam Amiri, Hussein Al Osman, Shervin Shirmohammadi, and Maha Abdallah. Toward delay-efficient game-aware data centers for cloud gaming. *ACM Trans. Multimedia Comput. Commun. Appl.*, 12(5s), September 2016.

[11] Maryam Amiri, Ashkan Sobhani, Hussein Al Osman, and Shervin Shirmohammadi. SDN-Enabled Game-Aware Routing for Cloud Gaming Datacenter Network. *IEEE Access*, 5:18633–18645, Sep 2017.

[12] G. Armitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In *The IEEE International Conference on Networks*, pages 137–141, 2003.

[13] Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Advances in Neural Information Processing Systems*, volume 32, pages 4954–4965. Curran Associates, Inc., 2019.

[14] A. Bhutani and P. Wadhwani. Cloud gaming market size ... market share & forecast 2019–2025. *Global Market Insights, Report ID: GMI2368*, June 2019.

[15] Hamoud S. Bin-Obaid and Theodore B. Trafalis. Fairness in resource allocation: Foundation and applications. In Ilya Bychkov, Valery A. Kalyagin, Panos M. Pardalos, and Oleg Prokopyev, editors, *Network Algorithms, Data Mining, and Applications*, pages 3–18, Cham, 2020. Springer International Publishing.

[16] Timothy X. Brown, Hui Tong, and Satinder P. Singh. Optimizing admission control while ensuring quality of service in multimedia networks via reinforcement learning. In *Advances in Neural Information Processing Systems 11*, pages 982–988. MIT Press, 1999.

[17] Toon Calders and Sicco Verwer. Three naive bayes approaches for discrimination-free classification. *Data Min. Knowl. Discov.*, 21:277–292, 09 2010.

[18] L. Elisa Celis, Lingxiao Huang, and Nisheeth K. Vishnoi. Multiwinner voting with fairness constraints. *preprint arXiv: 1710.10057*, 2017.

[19] L Elisa Celis, Damian Straszak, and Nisheeth K Vishnoi. Ranking with fairness constraints. *preprint arXiv:1704.06840*, 2017.

[20] K. Chen, C. Huang, and C. Hsu. Cloud gaming onward: research opportunities and outlook. In *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–4, 2014.

[21] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, page 1269–1272. Association for Computing Machinery, 2011.

[22] Min Chen, Yixue Hao, Long Hu, M. Shamim Hossain, and Ahmed Ghoneim. Edge-cocaco: Toward joint optimization of computation, caching, and communication on edge cloud. *IEEE Wireless Communications*, 25:21–27, 2018.

[23] Min Chen, Yixue Hao, Kai Lin, Zhiyong Yuan, and Long Hu. Label-less learning for traffic control in an edge network. *IEEE Network*, 32:8–14, 11 2018.

[24] Y. Chen, S. Deng, H. Zhao, Q. He, Y. Li, and H. Gao. Data-intensive application deployment at edge: A deep reinforcement learning approach. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 355–359, 2019.

[25] Yuchi Chen, Jiangchuan Liu, and Yong Cui. Inter-player delay optimization in multiplayer cloud gaming. *IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 702–709, June 2016.

[26] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017.

[27] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvtiskii. Matroids, matchings, and fairness. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 2212–2220. PMLR, 16-18 Apr 2019.

[28] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. A hybrid edge-cloud architecture for reducing on-demand gaming latency. *Multimedia Systems*, 20:503–519, 10 2014.

[29] M. Claypool and D. Finkel. The effects of latency on player performance in cloud-based games. In *2014 13th Annual Workshop on Network and Systems Support for Games*, pages 1–6, 2014.

[30] M. Claypool and D. Finkel. The effects of latency on player performance in cloud-based games. In *13th Annual Workshop on Network and Systems Support for Games*, pages 1–6, 2014.

[31] M. Claypool, D. Finkel, A. Grant, and M. Solano. Thin to win? network performance analysis of the onlive thin client game system. In *11th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, 2012.

[32] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Commun. ACM*, 49(11):40–45, November 2006.

[33] Mark Claypool and Kajal Claypool. Latency can kill: Precision and deadline in online games. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, page 215–222, Phoenix, Arizona, USA, 2010. ACM.

[34] Mark Claypool, Ragnhild Eg, and Kjetil Raaen. The effects of delay on game actions: Moving target selection with a mouse. In *The Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, page 117–123, Austin, Texas, USA, 2016. ACM.

[35] CCP Convict. Two guinness world records titles broken! *EVE news*, Oct 2020.

[36] Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. Algorithmic decision making and the cost of fairness, 2017.

[37] G. Cui, Q. He, X. Xia, F. Chen, H. Jin, and Y. Yang. Robustness-oriented k edge server placement. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 81–90, 2020.

[38] A.M. D'Argenio. Statistically, video games are now the most popular and profitable form of entertainment. *Game Crate*, June 10, 2018.

[39] T. de la Navarre. Biggest esports live events in history. Jun 2020.

[40] S. Deng, Z. Xiang, J. Taheri, K. A. Mohammad, J. Yin, A. Zomaya, and S. Dustdar. Optimal application deployment in resource constrained distributed edges. *IEEE Transactions on Mobile Computing*, pages 1–1, 2020.

[41] Y. Deng, Y. Li, R. Seet, X. Tang, and W. Cai. The server allocation problem for session-based multiplayer cloud gaming. *IEEE Transactions on Multimedia*, 20(5):1233–1245, 2018.

[42] Yunhua Deng, Yusen Li, Xueyan Tang, and Wentong Cai. Server allocation for multi-player cloud gaming. In *The International Conference on Multimedia*, page 918–927, Amsterdam, The Netherlands, 2016. ACM.

[43] Adithya M. Devraj and Sean P. Meyn. Q-learning with uniformly bounded variance: Large discounting is not a barrier to fast learning, 2020.

[44] E. Dhib, K. Boussetta, N. Zangar, and N. Tabbane. Cost-aware virtual machines placement problem under constraints over a distributed cloud infrastructure. In *The International Conference on Communications and Networking*, pages 1–5, 2017.

[45] Stratos Dimopoulos, Chandra Krintz, and Rich Wolski. Towards distributed, fair, deadline-driven resource allocation for cloudlets. In *Proceedings of the 4th Workshop on Middleware for Edge Clouds & Cloudlets*, pages 7–9, 2019.

[46] Hossein Ebrahimi Dinaki and Shervin Shirmohammadi. GPU/QoE-Aware Server Selection Using Metaheuristic Algorithms in Multiplayer Cloud Gaming. In *Annual Workshop on Network and Systems Support for Games*, Sep 2018.

[47] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. pages 214–226, 2012.

[48] S. Farlow and J. L. Trahan. Client-server assignment in massively multiplayer online games. In *Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games*, pages 1–8, 2014.

[49] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. *The SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, 2015.

[50] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. On the (im)possibility of fairness, 2016.

[51] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. 16(1):1437–1480, January 2015.

[52] U. Goel, M. P. Wittie, and M. Steiner. Faster web through client-assisted cdn server selection. In *The International Conference on Computer Communication and Networks*, pages 1–10, 2015.

[53] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop*

on Internet Measurment, IMW '02, page 5–18. Association for Computing Machinery, 2002.

[54] Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[55] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang. A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 31(3):515–529, 2020.

[56] Qiang He, Guangming Cui, Xuyun Zhang, Feifei Chen, Shuiguang Deng, Hai Jin, Yanhui Li, and Yun Yang. A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 31(3):515–529, 2020.

[57] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein. It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 365–375, 2018.

[58] H. Hong, D. Chen, C. Huang, K. Chen, and C. Hsu. Placing virtual machines to optimize cloud gaming experience. *IEEE Transactions on Cloud Computing*, 3(1):42–53, 2015.

[59] Y. Hu, D. Niu, and Z. Li. A geometric approach to server selection for interactive video streaming. *IEEE Transactions on Multimedia*, 18(5):840–851, 2016.

[60] Chun-Ying Huang, Kuan-Ta Chen, De-Yu Chen, Hwai-Jung Hsu, and Cheng-Hsin Hsu. Gaminganywhere: The first open source cloud gaming system. 10(1s), January 2014.

[61] C. V. N. Index. Forecast and methodology, 2016–2021. *White paper, Cisco public*, Vol. 6, 2017.

[62] Matthew Joseph, Michael Kearns, Jamie H Morgenstern, and Aaron Roth. Fairness in learning: Classic and contextual bandits. 29:325–333, 2016.

[63] N. Kantasewi, S. Marukatat, S. Thainimit, and O. Manabu. Multi q-table q-learning. In *International Conference of Information and Communication Technology for Embedded Systems*, pages 1–7, 2019.

[64] Jon M. Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *arXiv*, abs/1609.05807, 2017.

[65] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[66] P Lai, Q He, G Cui, F Chen, M Abdelrazek, J Grundy, J Hosking, and Y Yang. Quality of experience-aware user allocation in edge computing systems: A potential game. In *40th IEEE International Conference on Distributed Computing Systems*, 2020.

[67] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, and Yun Yang. Optimal edge user allocation in edge computing with variable sized vector bin packing. In Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu, editors, *Service-Oriented Computing*, pages 230–245, Cham, 2018. Springer International Publishing.

[68] Phu Lai, Qiang He, Guangming Cui, Xiaoyu Xia, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, and Yun Yang. Edge user allocation with dynamic quality of service. In Sami Yangui, Ismael Bouassida Rodriguez, Khalil Drira, and Zahir Tari, editors, *Service-Oriented Computing*, pages 86–101, Cham, 2019. Springer International Publishing.

[69] Phu Lai, Qiang He, Guangming Cui, Xiaoyu Xia, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, and Yun Yang. Qoe-aware user allocation in edge computing systems with dynamic qos. *Future Generation Computer Systems*, 112:684 – 694, 2020.

[70] Zhuorui Lan, Weiwei Xia, Wenqing Cui, Feng Yan, Fei Shen, Xuzhou Zuo, and Lianfeng Shen. A Hierarchical Game for Joint Wireless and Cloud Resource Allocation in Mobile Edge Computing System. In *2018 10th International Conference on Wireless Communications and Signal Processing, WCSP 2018*, Nov 2018.

[71] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 5639–5650, 2020.

[72] Yusen Li, Yunhua Deng, Xueyan Tang, Wentong Cai, Xiaoguang Liu, and Gang Wang. On server provisioning for cloud gaming. In *Proceedings of the 2017 ACM Multimedia Conference*, pages 492–500. ACM, Oct 2017.

[73] Yusen Li, Yunhua Deng, Xueyan Tang, Wentong Cai, Xiaoguang Liu, and Gang Wang. Cost-efficient server provisioning for cloud gaming. *ACM Transactions on Multimedia Computing, Communications and Applications*, 14(3s), Jun 2018.

[74] Y. Liu, Q. He, D. Zheng, X. Xia, F. Chen, and B. Zhang. Data caching optimization in the edge computing environment. *IEEE Transactions on Services Computing*, pages 1–1, 2020.

[75] A. Madej, N. Wang, N. Athanasopoulos, R. Ranjan, and B. Varghese. Priority-based fair scheduling in edge computing. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 39–48, 2020.

[76] Marc Manzano, Manuel Urueña, Mirko Suznjevic, Eusebi Calle, José Hernández, and Maja Matijasevic. Dissecting the protocol and network traffic of the onlive cloud gaming platform. *Multimedia Systems*, 20:1–20, 03 2014.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[78] Shady A. Mohammed, Shervin Shirmohammadi, and Sa'DI Altamimi. A Multimodal Deep Learning-Based Distributed Network Latency Measurement System. *IEEE Transactions on Instrumentation and Measurement*, 69(5):2487–2494, May 2020.

[79] Kyoung Shin Park and R.V. Kenyon. Effects of network characteristics on human performance in a collaborative virtual environment. In *Proceedings IEEE Virtual Reality (VR'99)*, pages 104–111, Houston, Texas, 1999.

[80] Q. Peng, Y. Xia, Z. Feng, J. Lee, C. Wu, X. Luo, W. Zheng, S. Pang, H. Liu, Y. Qin, and P. Chen. Mobility-aware and migration-enabled online edge user allocation in mobile edge computing. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 91–98, 2019.

[81] F. Qin, Z. Zhao, and H. Zhang. Optimizing routing and server selection in intelligent sdn-based cdn. In *The International Conference on Wireless Communications Signal Processing (WCSP)*, pages 1–5, 2016.

[82] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. *ACM Comput. Surv.*, 52(6), October 2019.

[83] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato. Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control. *IEEE Transactions on Computers*, 66(5):810–819, 2017.

[84] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[85] Shervin Shirmohammadi and Nicolas Georganas. An end-to-end communication architecture for collaborative virtual environments. *Computer Networks*, 35:351–367, 11 2000.

[86] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[87] Giorgio Stampa, Marta Arias, David Sanchez-Charles, Victor Muntes-Mulero, and Albert Cabellos. A deep-reinforcement learning approach for software-defined networking routing optimization. In *arXiv*, 2017.

[88] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[89] H. Tian, D. Wu, J. He, Y. Xu, and M. Chen. On achieving cost-effective adaptive cloud gaming in geo-distributed data centers. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2064–2077, 2015.

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, 2017.

[91] K. Webb. More than 100 million people watched the 'league of legends' world championship, cementing its place as the most popular esport. *Business Insider*, December 18, 2019.

[92] Steven Webb and Sieteng Soh. Adaptive client to mirrored-server assignment for massively multiplayer online games. *The International Society for Optical Engineering*, Jan 2008.

[93] A. Webster. Pro gaming leagues are seeing a huge spike in viewership. *The Verge*, April 27, 2020.

[94] T. Wijman. The global games market will generate $152.1 billion in 2019. *Newzoo*, June 18, 2019.

[95] H. Wu, S. Deng, W. Li, M. Fu, J. Yin, and A. Y. Zomaya. Service selection for composition in mobile edge computing systems. In *IEEE International Conference on Web Services*, pages 355–358, 2018.

[96] Xiaoyu Xia, Feifei Chen, Qiang He, Guangming Cui, Phu Lai, Mohamed Abdelrazek, John Grundy, and Hai Jin. Graph-based optimal data caching in edge computing. In Sami Yangui, Ismael Bouassida Rodriguez, Khalil Drira, and Zahir Tari, editors, *Service-Oriented Computing*, pages 477–493, Cham, 2019. Springer International Publishing.

[97] Z. Xu, G. Zou, X. Xia, Y. Liu, Y. Gan, B. Zhang, and Q. He. Distance-aware edge user allocation with qoe optimization. In *2020 IEEE International Conference on Web Services (ICWS)*, pages 66–74, Los Alamitos, CA, USA, Oct 2020. IEEE Computer Society.

[98] Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, Jun 2017.

[99] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P. Gummadi. Fairness constraints: A flexible approach for fair classification. *Journal of Machine Learning Research*, 20(75):1–42, 2019.

[100] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems 31*, pages 3562–3573, 2018.

[101] Sebastian Zander and Grenville Armitage. Empirically measuring the qos sensitivity of interactive online game players. In *ATNAC*, Dec 2004.

[102] Shuopeng Zhang, Di Niu, Yaochen Hu, and Fangming Liu. Server selection and topology control for multi-party video conferences. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 43–48, Singapore, Singapore, 2014. ACM.