Real-Time Online Matchmaking Queuing

Anthony Chen, Isaac Fu

***Abstract*- In online multiplayer games, latency between machines on the internet heavily affects the user experience. Ideally players seek to join game sessions with low latency to other players. For the majority of games, the matchmaking system is ran at the server-side instead at the clients. It is crucial for these systems to select groups of players with low latency to each other. However other aspects of matchmaking is important like time takes for parties to form. Our project is a case study on a few different matchmaking systems that arranges parties of users from a larger group of players. One matchmaking method is to probe the network of users and select groups that have the overall best connection to each other. Another method involves the concept of Htrae, a latency prediction system for matchmaking based on network coordination systems and geolocation, that will predict for players to join the queue who has better connections.**

I. Introduction

Due to the increase of users in the gaming industry, there has been an increasing demand for more efficient matchmaking algorithms that also generate lower amount of overall ping. Figure 1 shows the increase in the reach that gaming has just in the United States over the past five years.

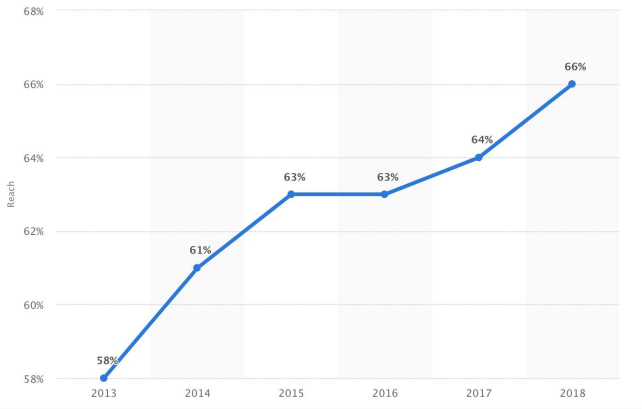


Fig 1. Percentage of people in the United States who have been "reached" by the gaming industry by year

Due to this increased influx of users, matchmaking algorithms can now utilize this influx to create more efficient algorithms that predictionally wait for users to enter the queue, whereas before with less users it was less reliable. As gaming gains more and more reach into U.S. culture and society, there is a larger spread of users across the country. New matchmaking algorithms can use this geolocation data to predict users entering the queue, and create optimized groupings among users using the data.

The main purpose behind this project is to analyze and weigh the consequences of using geolocation as a tool to create more optimized groupings in matchmaking queues. This will be tested with a live queue of users, and our server will create groupings of five within the users. One user will be the **host** of that group, meaning that all of the other users in the group are connecting to that one host. Their connection status will be calculated through pinging. We will be analyzing the efficiency of our matchmaking algorithm using mainly 2 criteria:

- **Time:** The time it takes for the algorithm to successfully create all the groupings within a set amount of users.
- **Lower Overall Ping:** The average amount of ping among the users within the group in respect to the host of the group.

II. Design and Implementation

The matchmaking system is built up using three main classes: a city class, server class, and a user class. The city class and user class are simple, and the server class is a complex system with the matchmaking functions and all of the information. The goal was to imitate where modern companies would handle matchmaking assignments, on the company's servers and not on the client's device.

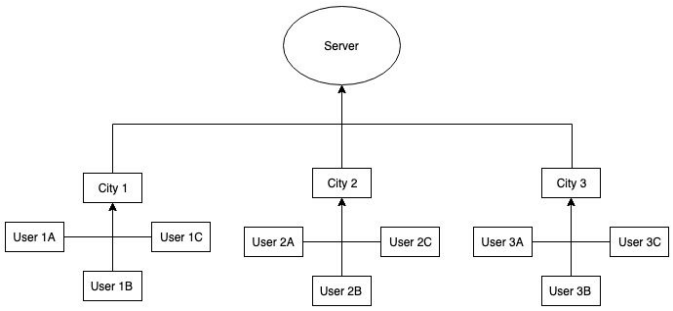


Fig 2. Relationship of objects to each other

A. *Design Elements*

1.) City Object: The city objects are connected to each other as a graph with weighted edges, and mainly just contain users. The class is used to find the distance between users, with the weighted edges being the distance from cities. They serve a key role in the implementation geolocation prediction, as each city will have a statically initialized probability of a new user entering into the queue in their city. This works since cities that have more people, or just a higher gaming population usually will have more users playing.

2.) Server Object: The server objects contain a list of all the users, cities, and the adjacency matrix to calculate distances between the weighted graph of cities (each city will have users in them). The server does most of the work in our system, as it tells users to ping each other, and will be accessed for the distances between cities. It also contains our matchmaking algorithm, which we will discuss below.

3.) User Object: The user objects contains a bandwidth identifier(which is its own personal internet speed), and a vector of hosts that it would want to connect to. The functionality that the user has is mainly pinging other users that our looking to queue, and relaying that information back to the server for matchmake. The **ping** of each user with respect to other users is calculated by the distance between the two users divided by the pinging users own bandwidth value.

4.) *Simulator Object*: It initializes the server and all the cities with the initial conditions(such as desired amount of ping, bandwidth of users, number of cities, number of users) and then adds users into the matchmaking queue. The simulator can then be used to track the users and their statuses. The simulator will also calculate the **time** that each algorithm takes to matchmake all the users through the use of the <chrono> library in C++.

B. *Implementation of Matchmaking Algorithms*

We have implemented 3 matchmaking algorithm and compared their performances. One implementation is a standard queueing system without geolocation, another implementation is simply a random matchmaker, and the final implementation is using a mix of the first algorithm and geolocation.

1.) Ping Decision Matchmaking: The matchmaking algorithm works by having each user creating a list of users that it would want to connect to. This is done through the server having a list of all the users in queue, and telling them to ping each other and find their latency with each other user. This latency would be found through a calculation of the distance from each user(which is found through the edge weights between city nodes, as each user is in a city) with the bandwidth of each user. If this found latency is less than a set desired amount of ping, then the user will be added to its list of hosts it is willing to connect to. This information is then sent to the server, which will now have the acceptable hosts vector for each user. The server then looks through these vectors, keeping track of how many times each user is mentioned. If one user is mentioned four times, then the server will matchmake all of these users together to form a group. The server will then remove all these users from the queue, and update all other users of this information. Effectively, this is a greedy algorithm that might not create the most optimal groupings for a set amount of users; however, with a matchmaking queue that continues to add more players, it will efficiently create adequate groupings.

This algorithm draws ideas from the SMP, Stable Marriage Problem. [3]

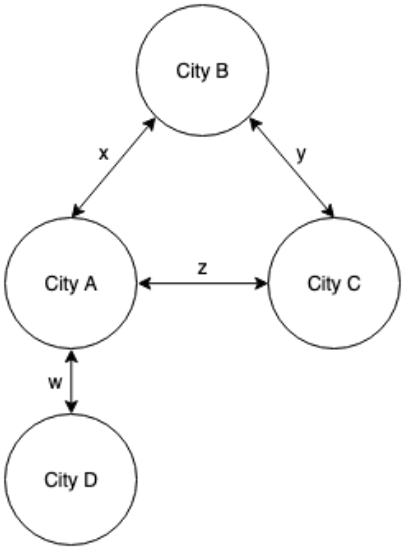
2.) Random Matchmaking: This matchmaking algorithm is simply a random matchmaker. Once the matchmaking queue reaches a certain size, the matchmaker will randomly select 5 people within the queue and create a group out of them. Obviously, this algorithm will not create the most optimal group pairings; however, it will be used as a basis of ping measurement.

3.) Geolocation Matchmaking: The matchmaking algorithm utilizes ideas found in the SMP, while also using geolocation probabilities as a qualifier. Our algorithm creates a list of users/hosts that are desired by 4 or more other users. The algorithm then checks to see if there are at least 4 users that are in the same city as the desired hosts city. If there are, then the algorithm groups them together. If not, then the algorithm will check the probability that another user will enter into the queue at

the desired hosts city. Based on how high that probability is, the algorithm will either wait for a longer or shorter time for a new user to join the queue at the city. In this sense, the algorithm is predicting the next string of users joining the queue. If the algorithm successfully waits and a new user joins at the desired hosts city, then all is well and the algorithm pairs them together. If not, then the algorithm will decide that it has waited too long for a new user, and will group the users together anyways(even though they aren't all in the desired hosts city).

C. Implementation of Simulation

The simulation is manages and controls the matchmaking server. All the information within the server is configured by the simulation. There are three key values in the matchmaking server: the distance matrix of the cities, the list of users, and the lists of cities. The cities and their connections to one another may be visualized as a weighted graph, shown in figure 3. For the server, the graph is encoded into a form of an adjacent matrix, shown in figure 3. Our tests use a simple city layout consisting of four cities that are connected to each other; the orientation of the different cities are displayed in figure 3.



	A	B	C	D
A	0	x	z	w
B	x	0	y	null
C	z	y	0	null
D	w	null	null	0

Fig 3. City Connections as graph(top) and adjacent matrix (bottom)

The simulation adds users into the server where they would be grouped into parties of five. Methods to calculate measurements like the average ping of each party and the time for a user to be placed into a group are carried out after all the users were placed into groups. The simulation records the parties of players and emulate the pinging functions to calculate the average speeds of each party of users.

III. Alternative Methodologies

Throughout our design process, we came across other ways of doing matchmaking; however we deemed ours to give the most freedom in host choosing and group making.

The original idea we had was to have all the users ping each other user and create a distance-vector table. We would implement some grouping algorithm that clusters these graphs into optimal clusters of 5. So, if we had a graph of 50 users, then they would be clustered into 10 optimal groups of 5. We found that there really was no efficient way to cluster graphs into groups. On top of this, it would be hard to use this algorithm in conjunction with a real-time list of users. It would be difficult because the distance-vector table would have to shift and change every time a user is removed or added. The advantages of this methodology would be that it would remove the need for a server to centralize all decisions. The users themselves could divide each other into groups and find optimal host-groups.

Another idea was to create groupings as the users come in. One user would become the designed host for that group, and all future users to join the queue will measure their latency with the designated hosts for each group. If the latency was too high for any of these groups, then they would create their own group. We found that this methodology, although simple in creating groups, would designate hosts ineffectively. These hosts would be assigned too preemptively, and not give the algorithm enough time to properly choose the most optimal host for the group.

IV. Evaluation and Results

Each algorithm focuses on certain characteristics of matchmaking that are evident in the performance of each

matchmaking method. In this section, the evaluation of each method to determine whether the algorithm is well suited for real-world application. The two metrics characterized to represent the performance are average ping connection between each player in the group to the party host and the time to match players together. One disadvantage of our simulation is that it does not account for the connection over time during the game, and focuses on the initial connections between users.

Method	Ping (s)	Time (ms)
Ping-Decision	1.22971	2071
Random	7.05727	2990
Geolocation	1.03173	20178

Fig 4. Average ping and time to matchmake for 100 simulators with 50000 users

A. Ping-Decision Matchmaking

The overall Ping-Decision results in the fastest time in matchmaking. There is little stalling for players to join and mainly takes account for current players within the lobby. This is a greedy algorithm that aggressively forms parties and minimize the size of the lobby and the time to form groups of players, which ultimately causes the time to be very fast. The ping is pretty small since the desired ping that we set to find under was low as well. Figure 5 shows the results and the average ping per group distribution amongst 100 simulations. Our distribution follows a normal distribution. Since the matchmaking is attempted whenever users are added into the server, the overall ping of each one is

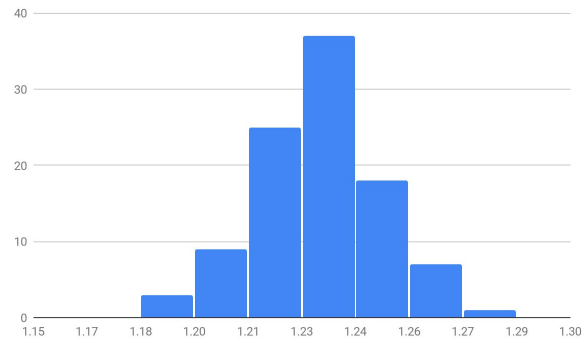


Fig 5. Distribution of ping amongst 100 simulations for Ping-Decision matchmaking algorithm

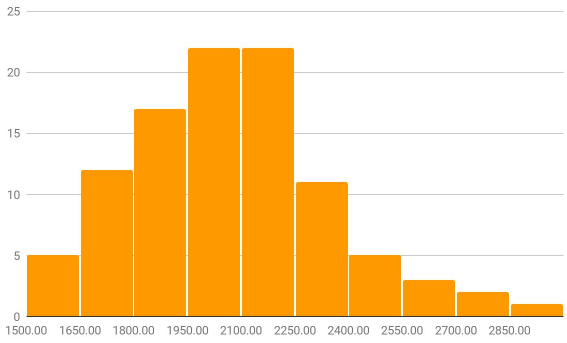


Fig 6. Distribution of Time(in ms) of Ping-Decision matchmaking algorithm

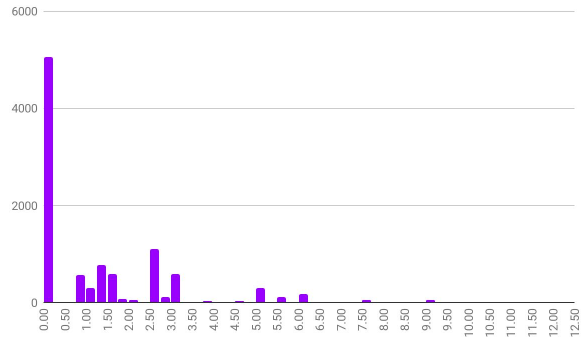


Fig 7.: Distribution of ping among 10,000 parties for one simulation with Ping-Decision Matchmaking algorithm

B. Random Matchmaking

Random Matchmaking algorithm is used as a baseline to determine the effectiveness of other matchmaking algorithms. Random Matchmaking is not as greedy as the Ping-Decision Matchmaking, which results in the slightly slower time to form parties. The selection speed is the largest due to the nature of not having any heuristic method of selecting players into a party, as shown in figure 8.

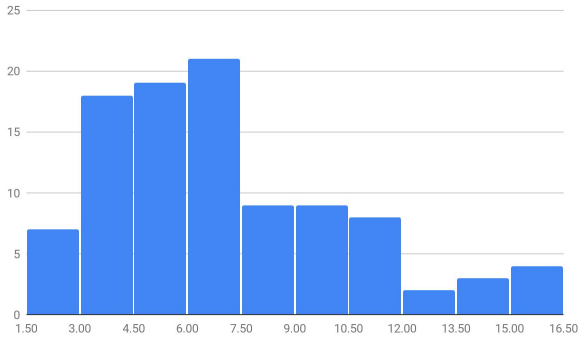


Fig 8. Distribution of ping amongst 100 simulations for Random matchmaking algorithm

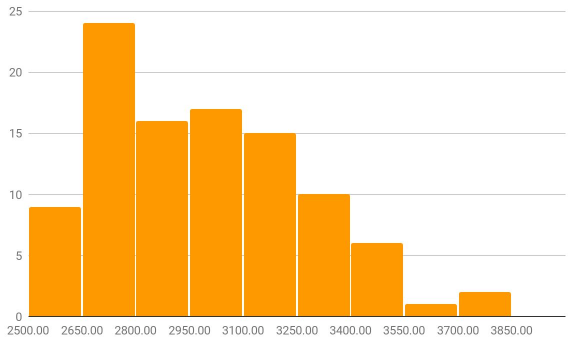


Fig 9. Distribution of Time(in ms) of Random matchmaking algorithm

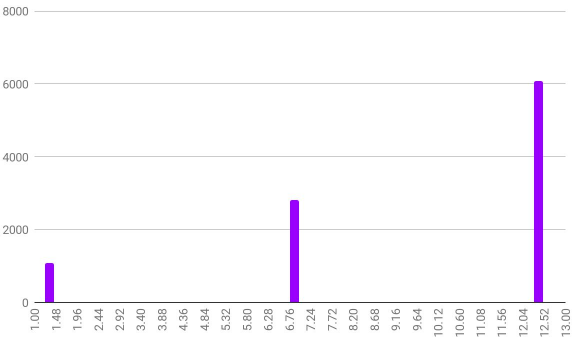


Fig 10. Distribution of ping among 10,000 parties for one simulation with Random Matchmaking algorithm

C. Geolocation Matchmaking

Geolocation matchmaking produces the best average pings between users of a party compared to other algorithms, however the time required to form groups is significantly larger than the others. This is due to the algorithm deciding whether or not to form parties with current players or wait for new users with better connections.

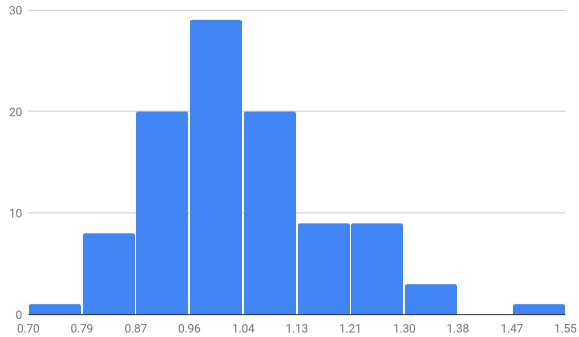


Fig 11. Distribution of ping amongst 100 simulations for Geolocation matchmaking algorithm

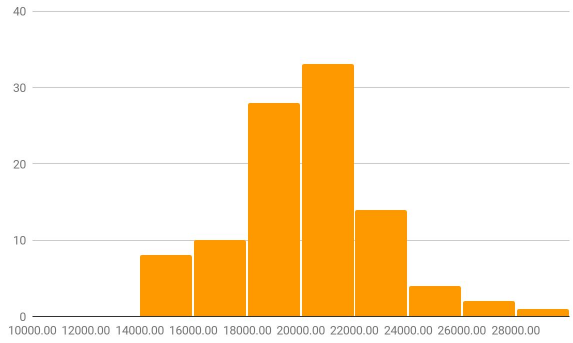


Fig 12. Distribution of Time(in ms) of Geolocation matchmaking algorithm

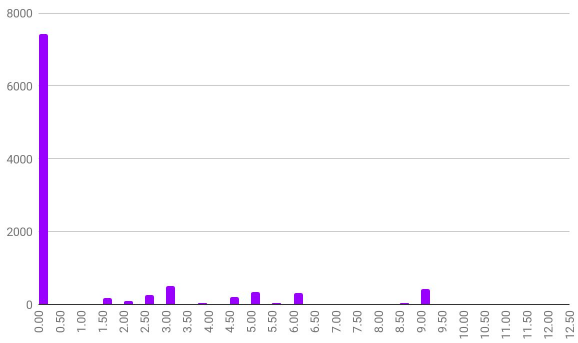


Fig 13. Distribution of ping among 10,000 parties for one simulation with Geolocation Matchmaking algorithm

D. Analysis

From the data, we see that geolocation created groupings of the lowest amount of average ping, quickly seconded by the Ping-Decision matchmaking, and then lastly followed by the random matchmaking. We see that both of our implemented methods heavily improve average ping amongst users when compared to the Random matchmaking. However, the difference in average ping between geolocation and Ping-Decision was more miniscule than originally thought. We hypothesize that this is due to geolocation probabilities indirectly affecting the Ping-Decision algorithm. Since, more users are generally joining at one city, even though the Ping-Decision algorithm doesn't account for this, it naturally will since it groups quickly and aggressively. This means that it is probable that many users will be in the same city, and therefore geologically located closely, when the Ping-Decision algorithm groups users together, even though it doesn't account for that in the calculation.

For time, we see that the Ping-Decision method was the fastest, quickly followed by the Random algorithm, then followed by geolocation. A surprising result was to see the Ping-Decision method run faster than the random algorithm. We hypothesize that this is due to the random algorithm waiting to matchmake to ensure randomness, while the Ping-Decision method doesn't have to wait. The main thing is that the geolocation algorithm is considerably slower than the other algorithms. We hypothesize that this heavy increase in time is simply due to the complexity of the algorithm. Similar to encoding/compression delays that we learned in class, where encoding algorithms created smaller packets at the cost of longer delay, even though the geolocation algorithm creates better groupings, it does this at the cost of the time it takes to run these complex algorithms.

One main cause of concern for the geolocation algorithm is that it is very uneven in the way that it groups users. Fig.13 shows the distribution of groups made in one simulation(10000 groupings). There is a large disparity in the quality of the groupings. There are many groupings which have a very low amount of ping; however, there are other groupings which have ping values of upward to 9, worse than the average for the random matchmaking. We hypothesize that this is due to the geolocation algorithm heavily favoring cities that have a high probability of new users joining. When cities have a low probability of new users joining, users in that city are always matched poorly and get the short end of the stick in terms of ping. Without a way to compensate for this difference, the algorithm unfortunately makes a considerable amount of groupings with very poor average ping, which could be detrimental to the overall experience of the online matchmaking.

The random algorithm generated a very weird set of data for the distributions of ping within one simulation, as shown in Fig. 10. We hypothesize that this is because the random number generator in C++ fails to create enough randomness.

V. Future Work

A future feature that could be added to our matchmaking algorithm would be fine tuning the users that each user decides to ping. In a larger simulation with more cities, it would benefit the efficiency of the algorithm if users only pinged other users that are under a set distance away from them. This would reduce the overhead of pinging and speed up the algorithm, by having users that are all the way across the country not ping each other. In the scope of our project, we didn't implement this feature, since our simulation was small in comparison; however, when dealing with large graphs, it would be necessary to expedite the pinging process.

Another feature that could be implemented is calculating the probability used for geolocation dynamically instead of declared statically. The server would keep track of the city that each user is joining in the queue at and dynamically calculate the probability of another user joining the queue at each city. This would allow the geolocation to adjust to changing user demographics, as certain cities have more players playing at night rather than the day. Since geolocation's efficiency is tied to the probability data, this would increase performance for the geolocation algorithm.

Another feature that can be implemented into our algorithm is a way to compensate for the cities with low amount of influx in users. This would result in lower deviations between groupings and get rid of outliers in the data set, and ultimately create a better matchmaking system.

VI. Conclusion

The rise of the gaming industry has brought a new concern of more efficient matchmaking algorithms that would generate lower amount of overall ping for players connected to one other. There are various characteristics to grouping players like player's geolocation. We designed two different algorithms: one that captures the fundamental concept of Microsoft's Htrae and another that considers ping connection to all other users. We also implemented an algorithm that generates random parties of users to use as a benchmark for other algorithms. The metrics we considered are ping connection and time to form parties.

Although we discovered that the heuristic of waiting to group users within the same geolocation would result in the best ping among players of a party, the processing required places a toll in the performance and time of the program. The ping-awareness algorithm produces groups with slightly larger ping but in a faster time. In future work, we could consider reducing the amount of pinging between all users and dynamically calculating the probability of new users within cities.

Bibliography

- [1]Agarwal, S., & Lorch, J. R. (2009). Matchmaking for online games and other latency-sensitive P2P systems. *ACM SIGCOMM Computer Communication Review, 39*(4), 315. doi:10.1145/1594977.1592605
- [2]Wang, Ning, and Jie Wu. “Latency Minimization Through Optimal User Matchmaking in Multi-Party Online Applications.” *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2018, doi:10.1109/wowmom.2018.8449817.
- [3]Gusfield, D., & Irving, R. W. (1989). *The Stable marriage problem: Structure and algorithms*. Cambridge, MA: The MIT.