Area Mapping in Voluminous Virtual Environments

Jason D. Stock
stock@colostate.edu

Evan J. Steiner
evanjs@colostate.edu

William A. Pickard
wpickard@colostate.edu

Colorado State University
December 7, 2019

1 Introduction

A major drive for many video game developers is to present as realistic simulated environments as possible. Such advancements have been made possible via improvements in software, hardware and game architectures. One popular gaming genre that has recently taken the spotlight is massively multiplayer online (MMO) games, which enable many players, on the scale of hundreds to thousands, to interact with each other within a large and persistent open world. Unlike most other game genres, MMO's require highly specialized servers and infrastructure in order to handle the enormous volume of requests required to run these games.

Various online game architectures, and MMO's alike, require users to offset server dependencies by storing local copies of graphics, textures, and sounds for rendering. In most cases, a user is required to download content to their local machine prior to playing the game. A few examples of this design include Grand Theft Auto V developed by Rockstar in 2013, and Gears of War 4 developed by Microsoft in 2016, with a download size of 65 GB and 120 GB respectively. The quality of many games continues to improve with larger virtual environments and more realistic graphics at the expense of increasing download size. Although download size has yet to cause significant issues, what happens when a game becomes as large as half a terabyte? A simple remedy may be to expand local storage at the client at the expense of additional resources. However, it becomes clear that relying on clients to meet these demands can be detrimental to a business model. Moreover, many clients do not have access to blazing fast internet, which can result in extremely long download times. This presents a dilemma where we must improve the effectiveness of server designs and hardware in order to further advance the state of modern video games.

While network speed is one of the primary issues preventing the gaming world from moving to the cloud, it is not something that can be easily improved from a game development and programming perspective. Rather, there is a reliance on hardware to enhance the bandwidth and reach of our networks, changes that are rapidly approaching as demand for high-speed internet increases. Therefore, it is worthwhile to further consider how distributed systems can be leveraged as we move towards a cloud-based gaming era.

As an alternative to existing architectures we explore how properties of large virtual environments can complement modern server designs, e.g., cloud services. We evaluate issues inherent with extraordinarily large environments, the effectiveness of current dominant approaches, and contributing our own solution to the problem. It is important to recognize that this work is not a fit-all model for every game design. Instead we focus on a specific use case characterized by our problem in the following section.

2 Problem Characterization

Architecture for large scale MMO's, and other online games, cannot simply be solved through a single server design. These designs must handle many requests simultaneously, perform calculations and maintain information as numerous users interact with each other and the virtual world. Furthermore, most software applications have numerous nonfunctional requirements, and online games, specifically, are tied to very stringent latency measures [1]. Modern multiplayer games run on a client-server architecture, and as such, all actions initiated by a player, must run through that server [1]. Because of this, the likelihood of prolonged latency increases, and is easily noticed by a client.

In a game that requires high precision movements and accuracy, introducing latency may lead one client to perceive another in a position different than what they have reported to the server. This may cause the client to misinterpret the position and wrongfully act upon the situation, causing significant disadvantages and frustrations to the player. Even in situations where precision is not as necessary, such as when moving about an environment, high latencies can cause issues. Increased latency causes movements to register in infrequent intervals, resulting in a concept known as lag - which is jarring to the user. This is not ideal in an online game where consistent movements are necessary.

In order to address latency concerns, distributed server architectures for these online games seek to address issues of high congestion and hotspots with seamless interactions between players by partitioning virtual areas across a set of servers [2]. Considering virtual proximity improves network latency by focusing computations on separate servers for players that are close in the virtual world, reducing the amount that different servers need to communicate - section 3 elaborates on this approach with the use of an example.

Many software applications are rapidly moving toward cloud environments, and as such, an opportunity arises for video games to migrate. As mentioned prior, it is important to maintain low latencies for users, but there is now an additional focus on data distribution. With a game as large as half a terabyte, it becomes infeasible for clients to download, and even for individual servers to maintain, requiring server designs to handle an additional constraint. Specifically, servers will need to effectively distributed data, and partitions of large environments in order to distribute load, improve redundancy, and minimize network latency. Servers must also handle many clients which may be playing different games or be in different map sections by distributing load in a manner that takes advantage of virtual proximity. A system that best achieves these goals will create a desirable gaming environment.

3 Dominant Approaches

In online multiplayer games, the dominant architectures for the backend server infrastructure fall into two main categories: fixed zones and instances [3]. In fixed zone architectures, the virtual game world is subdivided spatially. Each spatial subdivision is then assigned to a separate server. While most implementations do allow for multiple subdivisions to be hosted on a single server, the converse is rarely possible, i.e., a single subdivision cannot be shared across multiple servers. An example of a popular and commercially successful game that uses this architecture is EvE Online. In EvE Online, each solar system in a large galaxy is a subdivision and is handled by a server [4]. On the contrary, fixed zone architectures rely on an even spatial distribution of players to maintain balanced server loads, however, real-world scenarios have shown that most regions are empty while 1% are overloaded [5].

Instanced server architectures (sometimes referred to as shards) are used across multiple types of online multiplayer games. Instanced architectures simply replicate the virtual world across multiple servers or server groups. In almost all implementations of instanced server architectures, a key restriction is that once a user has joined a particular instance, they remain within that instance for the duration of the session. For games with short session durations such as first-person games like Fortnite [6], or strategy games like Starcraft 2, load balancing can be reasonably handled by efficient match-making algorithms when users attempt to join a new session. For persistent online worlds, such as World of Warcraft, balancing load across instances is much more difficult because the social component of the games drives users to cluster together to be with friends and experience a more vibrant virtual world.

While both dominant approaches described above are designed to balance server load in different ways, in real-world implementations they still suffer severe imbalances. Moreover, detailed design specifications for how many of these games handle these challenges are proprietary. Making it difficult for fine-grained research. However, our approach seeks to improve on the visible issues with these dominant approaches by implementing both spatial subdivision and dynamic replication based on user load.

4 Methodology

To support the aforementioned problem, we design an architecture for managing a number of isolated clients interacting in virtual environment. Clients will interact by moving randomly within a specific section of the map called a sector. A voluminous environment may be composed of terabytes, or even petabytes of data, with thousands of sectors. However, within our resource constraints, our simulated two-dimensional

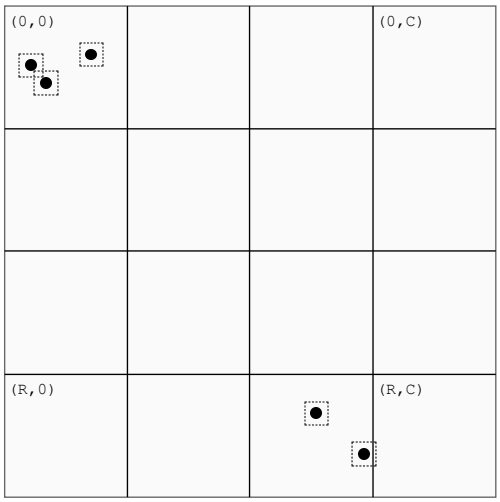


Figure 1: An R by C map divided into sectors showing various connected client locations. Each with their associated window sizes denoted by the surrounding dotted box.

map will have a total logical size of 100 GB's partitioned into 100 sectors, requiring each sector to be equally rendered by 1 GB of data (Figure 1).

This architecture will be broken into four components, including; (a) a client application for moving within the environment, (b) application servers for simulating the render of a client's environment, (c) a control switch for determining which server a client shall establish a connection with, and (d) a distributed file system (DFS) to effectively manage the data for all sectors in the environment (Figure 2).

Early architectural designs revolved around utilizing a custom backend DFS. This was preferable since there was a wide range of customizable features that could be implemented to adhere with how sector data is organized. For example, sectors bordering others may not want to reside on the same machine, or writes may need to be optimized for an environment that is changing. However, the data in this work is unstructured without correlation to other sectors and is infrequently modified. Under this assumption, we turn to use the Hadoop Distributed File System (HDFS) as our backend data store. HDFS has a relaxed write-once-read-many concurrency model that simplifies data coherency and provides high throughput access to data [7]. This allows for a reliable and consistent data store that can be simply integrated into our architecture.

Prior to initializing the application, and before clients are able to join, we populate HDFS with the environments sectors. To reach a map size of 10×10 sectors, a single 1 GB file is used to logically represents each sector of the map. This file will be distributed across a set of 10 worker nodes in 128 MB chunks with a replication factor of three. Replication is viable because we want to limit the cost of reconstructing each chunk of the file, at the expense of greater disk I/O at the data store. When a server requests the data for a given sector, it will retrieve the file with a label for that sector. Therefore, allowing a single 1 GB file stored in HDFS to simulate 100 different sectors. Upon delivery, the server reshapes this file into a $s \times s$ byte array, where $s = \sqrt{1\text{GB}}$. This defines the boundaries of a sector, and thus, the acceptable location for a client along the x, y coordinates.

When a client enters the game at a given sector, it will connect to an application server that has the data for that sector. This is only true under the conditions that other connected clients do not experience performance issues, otherwise a new server is consulted. If there are no server with the sector data stored in memory, an applicable server will request the sector data from HDFS. The control flow is between the servers and control switch, and as such, the switch has an omniscient view of the distributed services.

Each application server is responsible for servicing requests of connected clients by simulating the rendered environment. Each client is managed on a thread-per-connection basis. This design was chosen over utilizing a thread pool since operations could not be executed in batches or queued since minimizing latency per client is a priority. A connected client requests a specific $n \times n$ window around its given location at a rate of 25 packets per second (PPS). This interval is an average found over client packet loads from

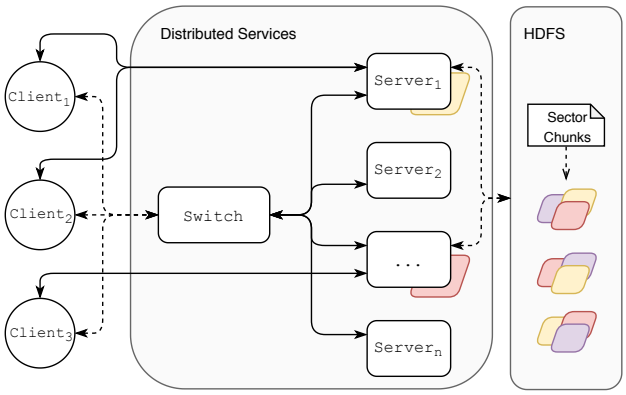


Figure 2: All four components (i.e., the client, control switch, application servers, and HDFS), are shown above, with their respective message flow. Solid lines represent persistent connections, and dotted lines are temporary connection used to send or receive information.

a MMO role playing game traffic analysis study from Chen *et al.* [8].

As a client approaches the boundary of a sector, a protocol is followed to preload the sector to an appropriate server. Upon receiving movement requests from clients, the server will return the window of data from all the shared volatile sectors in that window. If the window overlays two or more sectors, and they are not located on the same server, then the server forwards the request to the server with the sector preloaded. This additional server will send the partial window to the requesting client.

The control switch will receive heartbeats periodically from each application servers. These heartbeats, from each server, indicate the number of clients connected and the current sectors cached. This information will be used by the control switch to decide where to send incoming client join requests. When server processing times begin to increase, the switch may decide to send joining clients to a different server, even if that new server does not have those sectors cached. This approach will require network and disk I/O overhead when bringing new sectors into memory but will balance load as more requests for the cached sectors are made. Furthermore, by scheduling clients in a round robin fashion, with these considerations, we can minimize latency by directing requests away from overloaded servers.

When the application is launched there is a single switch initialized with a configurable number of servers connected. Our experiments detailed in the benchmarks below show utilization of various servers each on different nodes. To further isolate servers, we separately select one of 31 machines for servers such that they differentiate from the 28 machines reserved for clients and one machine for the switch. Once running, servers can join the network, although, we do not elaborate on the effectiveness of horizontal autoscaling in our results. Instead, we explore the limitations of the architecture under defined configurations to understand how distributing load effects performance.

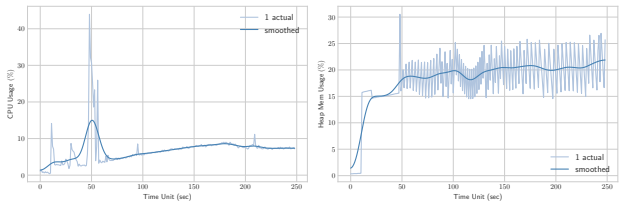
Initial experiments are conducted with a single server to establish base performance metrics. We seek to understand the quantitative limitations for how many sectors and clients can be managed by one JVM. These results will provide insight into when servers should be added to the application to scale effectively. After scaling resources, we extend similar evaluations to contrast how well distributing load is as compared to the base results.

5 Experimental Benchmarks

To analyze the effectiveness of our solution we perform benchmark tests at the client and server with variable configurations, i.e., number of sectors, servers, and clients. We analyze different resources on the server, e.g., system CPU usage, and JVM heap memory usage and thread count. We further evaluate the mean Round Trip Time (RTT) for clients as server load increased. This is similar to latency (one of the most important measurements in online gaming [2]), but only measures the time it takes for a message to be sent between two nodes over the network. While RTT measures the latency in both directions, including the processing time at each node. By measuring RTT we are able to measure and address deficiencies on a given server.

These metrics allow us to analyze how compute and memory intensive our solution was, and what sort of hardware would be required for each server. Figure 3a shows the CPU usage of server at an instance over time when one server is started up with one sector. The spike in usage occurred when the server finished loading the sector from HDFS, and the queued clients join the server simultaneously. Once the large influx of clients has established connections, the CPU usage flattens out. Similar results are seen in Figure 3b with heap memory usage for the same experiment. The usage is a percentage of the maximum heap memory being used - which is set to 7.4 GB. The memory usage increases by 15% or 1 GB after loading the sector into memory. A gradual increase is seen as more response messages are generated on the server.

Figure 4 shows the mean client RTT as more clients join the server. Over numerous trials we observe a maximum number of concurrent clients for a single server is consistently 95 before performance degrades. In our experiments clients request 100×100 windows at a rate of 25 PPS, resulting in a maximum response size of 0.05 MB with metadata from the server. Given



(a) (b)

Figure 3: (a) and (b) show the resource of a single server during an experiment with continuous connecting clients.

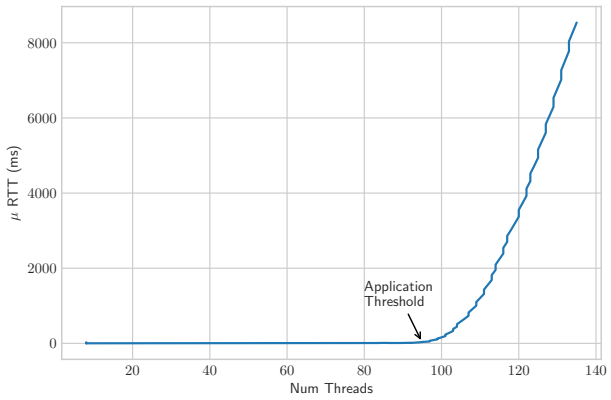
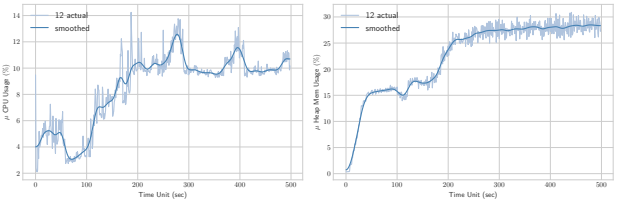


Figure 4: RTT for the initial client while additional clients continue to establish connections with a single server. The *Application Threshold* signifies the number of clients connected before performance degrades due to network I/O.



(a) (b)

Figure 5: (a) and (b) show the mean resources of 12 server during an experiment with clients continuously connecting in 10 different sectors.

ethernet bandwidth speeds of 850 Mbit/s, it is evident that a maximum of 100 MB can be transferred over the network considering TCP, IP, and other overheads [9]. Therefore, each server is bound by network I/O with a comfortable limit of 80 connections.

To contrast scaling the application with our base performance metrics, we evaluate our design with 12 servers and 1000 clients equally distributed among 10 sectors. All clients finish joining within the first 3 minutes of the experiment, and the system is left running to monitor performance. In Figure 5a, the mean CPU usage across all servers is shown for the duration of the experiment. The percentage does not greatly deviate from the base trial with the exception of spikes in two-minute intervals near the end, likely related to garbage collection. The JVM heap memory usage is consistent with an average of 2.25 GB per server (Figure 5b). This is higher than the base trial due to having duplicate sectors hosted on different servers to minimize mean client round trip time.

Server health is of importance when demonstrating

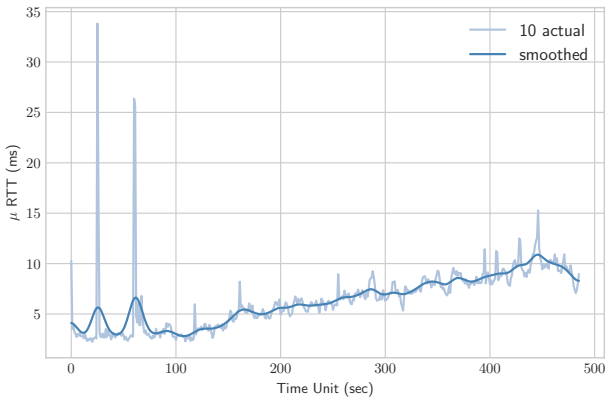


Figure 6: Mean Round Trip Time for the initial clients in each of the 10 sectors while additional clients continue to establish connections.

how load is managed, however, more prevalent results are seen with client RTT. We track the performance from the first client in each of the 10 sectors. This enables monitoring while other clients establish connections with servers. Mean times between all 10 clients range between 2 - 11 ms with a gradual increase before times decrease (Figure 6). Although there are spikes near the beginning of the figure, these can be attributed to the initial loading of sectors. In a real game server scenario, this is where loading screens would be used.

6 Insights

Our original architecture design was created with minimal knowledge of the problem space and the issues we would need to resolve within it. As such, we encountered numerous issues and oversights during our development process, which required us to make modifications and improvements to our original design.

Initial designs included our own DFS for storing and retrieving sector data. This decision was made to provide greater control over our project and allow specific improvements to the file system to better optimize our design. However, after further analysis we determined that a customized solution was not needed (as discussed in section 4). Moving towards HDFS improved our overall application and simplified it's design immensely, allowing us to abstract away the implementation of our DFS which was not the focal point of this work. Thereby bringing more attention to important implementation details in the distributed

services and map design.

Map topology was one of the larger oversights in our original design from which it was nearly absent and led to many issues during our initial implementations. We quickly discovered that the complexity of the sector transition logic comes from the topology of the virtual world. To simplify our concept, we chose to represent our environment in two-dimensions. Many online games simplify the topology of their virtual worlds even further by creating jump points or portals between parts of a larger map [10]. This can simplify the interconnection of sectors to a simple graph. In this implementation, a client was able to transition between two sectors at any point along a sector boundary. In a worst-case scenario, this meant that our solution must work for a client straddling four sectors simultaneously. A feature that is greatly simplified in two-dimensions.

It was evident that client performance was an essential area to monitor prior to experimentation. With this came various challenges, especially when interpreting the results. By reaching this understanding, we were able to tailor our implementation towards client experience and find limitations within the system. One example seen in initial evaluations of RTT brought insight to a defect for which delays were seen in continuous intervals. Extensive testing showed that by buffering the data I/O stream for transporting messages we could reach a more consistent flow of requests/responses. Furthermore, it was seen that network I/O at the server had a significant impact on overall performance. Initial assumptions lead us to believe that the high volume of data stored in memory would be most problematic, and distributing load based on server resource performance (i.e., CPU and memory usage) would be sufficient. However, our testing and analysis showed it is necessary to consider network traffic and throughput in our algorithm when scheduling client connections.

7 Future Space

The problem space surrounding the area of online game development is large today and has the potential to grow massively in the near future. The days of offline games are all but over, but there are continuous advancements in cloud infrastructures that present intriguing possibilities. While multiplayer games have always been server based by necessity, single player games are now more commonly requiring an internet connection to play. To achieve the goal of utilizing modern architectures, such as cloud services, applications and game designs must adapt.

Many game developers hope to create games for the masses, but for many the costs and requirements for hardware and networks are too high. One way to remedy this issue is to stream games to the user base by utilizing cloud infrastructures. Google Stadia [11] is one of the first and most prominent examples of this, seeking to stream games over the internet to users at home. Unfortunately, their work is proprietary, making research of the implementation difficult. Although the time may or may not be right for Google Stadia to break the market, given the stringent network requirements needed to stream games, a growing number of people are gaining access to higher speed internet. It is easy to imagine a future where nearly everyone has internet capable of streaming games to their homes. Furthermore, not every video game environment is large enough to warrant dedicated servers, but state of the art game developers suggest they are not far behind. Microsoft's new Flight Simulator [12] is said to be released in early 2020 and present planet earth with a two-petabyte dataset. This design is still under development, and similarly to Google Stadia, the work remains proprietary.

Another major component that will help drive the need and demand for cloud game services is Virtual Reality (VR). Although today, VR goggles run through powerful and expensive home computers, if the market wants to grow it should investigate pushing its way towards the cloud. Hosting and streaming games would lower the entry barrier dramatically, allowing more individuals to buy VR goggles and use them immediately. Our research shows that a future where games are streamed from the cloud, backed by terabytes of data, and presented in VR may not far away, but there remains a need for further research in the area.

8 Conclusions

The distributed architecture presented in this work has made several significant contributions. The first is the delivery of a voluminous virtual world, potentially petabytes in size, to clients with minimum latency. This was achieved by subdividing very large data files into smaller sectors that could be memory resident across multiple servers. This solution allowed client-server RTT to be within a 10 ms range, which is multiple orders of magnitude faster than if the client were to attempt to access this data directly from the

original DFS. This magnitude of difference is especially significant in the gaming world where small increases in latency can have significant negative impact on gameplay and user satisfaction. A key benefit of this architecture is that at no time is the client itself required to store the data composing the entire virtual environment. Because of this, the size of the virtual world is no longer limited by the hardware of the client, but by the resources available for the backend infrastructure. This result is also significant because it decouples the size of the virtual world from performance. A game developer can now easily increase the size of the virtual world by increasing the storage available on the DFS.

A second major contribution made by this architecture is the dynamic load balancing of the application servers based on the spatial-temporal state of the users, and server resource performance. The methodology utilized a control switch to monitor the load of both (a) users newly joining the system and (b) users as they changed locations within the virtual environment over time. As was shown by experimental data, individual servers will eventually experience a degradation in performance as their load exceeds a certain threshold due to the bounds of network I/O. By using the control switch to trigger replication of highly populated sectors, when performance began to degrade, horizontal scaling of applications servers was achieved. The experimental data verified that as more users joined the environment, the workload distributed across multiple servers, maintaining low client latency.

Combined, this distributed architecture for voluminous virtual environments successfully decouples client hardware, world size, and server resource utilization. In so doing, it provides step towards game developers having the freedom to scale worlds limited only by the capacity of the data store, and scale concurrent user count through efficient and balanced horizontal scaling of distributed services. Furthermore, this architecture helps open the door for future streaming services that require hosting games in a cloud environment.

References

[1] *MMSys '10: Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, New York, NY, USA, 2010. ACM. 423105.

[2] Marios Assiotis and Velin Tzanov. A distributed architecture for mmorpg. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '06, New York, NY, USA, 2006. ACM.

[3] Raluca Diaconu and Joaquín Keller. Kiwano: A scalable distributed infrastructure for virtual worlds. pages 664–667, 07 2013.

[4] Brendan Drain. Eve evolved: Eve online's server model. `www.engadget.com/2008/09/28/eve-evolved-eve-onlines-server-model/`. Accessed: 2019-11-05.

[5] Amir Yahyavi and Bettina Kemme. Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Comput. Surv.*, 46(1):9:1–9:51, 7 2013.

[6] Royal O'Brien. How would you keep 125 million gamers playing smoothly online? epic games shares its fortnite story. `https://aws.amazon.com/blogs/gametech/epic-fortnite-all-in-on-aws-cloud/`. Accessed: 2019-11-05.

[7] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[8] Kuan-Ta Chen, Polly Huang, Chun-Ying Huang, and Chin-Laung Lei. Game traffic analysis: an mmorpg perspective. pages 19–24, 06 2005.

[9] What is the actual maximum throughput on gigabit ethernet? `http://www.gigabit-wireless.com/gigabit-wireless/actual-maximum-throughput-gigabit-ethernet/`. Accessed: 2019-11-05.

[10] Narendra L. How does any mmo games backend work? `https://medium.com/@narengowda/how-does-any-mmo-games-backend-work-df19b44f73a7`. Accessed: 2019-11-05.

[11] Google stadia: A new way to play. `https://store.google.com/product/stadia`. Accessed: 2019-11-05.

[12] Mike Nelson. See the world in microsoft flight simulator. `http://news.xbox.com/en-us/2019/09/30/microsoft-flight-simulator-preview/`, 9 2019. Accessed: 2019-11-05.