Exploiting Google's Edge Network for Massively Multiplayer Online Games

Jared N. Plumb
University of Utah
jared.plumb@utah.edu

Ryan Stutsman
University of Utah
stutsman@cs.utah.edu

***Abstract*—Massively multiplayer online game servers are challenging to build and maintain; they require always-on availability, low-latency, and high predictability. In many ways, these systems could benefit from pushing application logic to the connected clients. However, historically peer-to-peer networks have not worked when applied to this domain. Pushing game logic to the clients complicates development since client end hosts cannot trust their peers and they have unreliably persistent connections, increasing the possibility of cheating or network failure.**

Google's Edge Network changes everything we have concluded about peer-to-peer networks over the past decade. Having access to thousands of servers all over the world solves the solution to problems of availability, data access, link saturation, security, and control. This new system allows for the inclusion of trusted peers in otherwise untrusted node clusters. Developers can explore peer-to-peer algorithms while allowing complete control over their data, code, and players, in their developed virtual worlds. They can offload application game logic to a low-latency scalable system, which provides a cost-effective solution residing closer to the connected clients.

In this paper, we investigate the gains game developers may obtain by exploring emerging edge cloud technology. We demonstrate how massively multiplayer online games benefit from this new system though simulation. We compare area-of-interest latency, which is the latency between client interactions within the virtual world, for known World of Warcraft and Google data centers locations. We show the benefits to area-of-interest latency gained from using Googles Edge Network, while minimally affecting client to server latency. Finally, we present a novel approach to maximize latency reductions for area-of-interest latency by moving players to optimal peering edge servers, thus reducing distance between clients.

I. Introduction

The past decade has shown that although peer-to-peer networks could result in vast improvements for online games, problems such as high jitter, link saturations, security, and reliability prevents the approach from use in practice [1] [2] [3]. Many of these conclusions changed as the Internet has improved, and advancements in hardware and algorithms could result in renewed exploration of the original problems.

Initially our research started by exploring the use of peer-to-peer networks when applied to massively multiplayer online games. We created a simulator to explore our theories about peer-to-peer network use, and found that although some problems such as jitter and reliability showed great promise with modern networks, issues of link saturation and byzantine behavior are still problematic. However, edge networks provide a new method forward. This is where Google's Edge Network [4] provides the solution for what we have been exploring.

Google's Edge Network is a collection of many small servers scattered throughout the world that act as a cache for a central server. These small servers known as **peering edge servers**, provide quick access to data originating from the central server. Clients connect to these peering edge servers and can quickly switch from edge to edge as needed.

Peering edge servers may communicate directly with each other, allowing re-exploration of peer-to-peer networks. This new approach allows us to have direct access to powerful hardware that alleviates the problems originally found in this type of network. Specifically, removing clients from the normal peer-to-peer equation solves link saturations, reliability, and security. We gain the benefits from peer-to-peer networks, while allowing the developers complete control over their virtual worlds and server software.

We explore these changes through simulation and find that the reduction in latency for area-of-interest interactions, which is the latency between client interactions within the virtual world, is significant. Figure 1 demonstrates these interactions showing the area around a player in the virtual world and its collisions with other players.

The effects of poor latency in the online game environment can be problematic and reducing latency and jitter between clients must be a high priority [5] [6]. Our work focuses on these latency requirements to establish if the possibility exists to use edge networks to improve performance for massively multiplayer online games.

We use three test cases to evaluate the performance of using Google's Edge Network: traditional networks, edge networks, and an optimal solution. **Traditional** networks, one where each client connects directly to the central server as shown in Figure 2, is the base case in our experiments. We demonstrate the use of an **edge network**, where each client connects to an edge in the same city as the client as shown in Figure 3, as a vast improvement over the traditional network. These peering edge servers connect to the central server and to each other.

Finally, we use an **optimal solution**, where connections are given an ideal path to each other through peering edge servers as shown in Figure 4, to show a best case scenario for area-of-interest latency. This connection type may be unrealistic, showing an unrealizable lower bound depending on hardware requirements. In this network topology, each client connects to

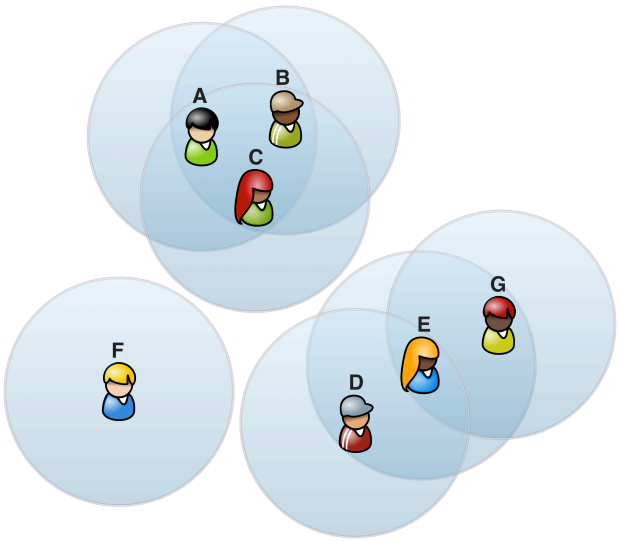

Fig. 1. An example of typical virtual world positioning with area-of-interest interactions with other players. A players immediate area is the area-of-interest for that player. Player A, B, and C are all positioned within each others area-of-interest. Player E has player D and G within the area-of-interest. However, player D and G are not within each others area-of-interest. Player F resides independently of the other players.

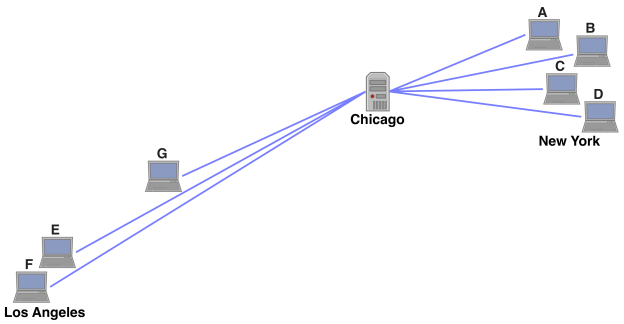

Fig. 2. An example of a traditional client to server topology used by most massively multiplayer online games for the virtual world in Figure 1. All clients connect directly to the central server regardless of their geographic location.

the peering edge server that has the fastest median ping time between the client and another client in the player's area-of-interest. We allow clients to connect to more than one edge server to minimize latency between numerous other clients in the player's area-of-interest. If no other clients are within the player's area-of-interest, the client connects to an edge server in the same city as the client. These peering edge servers follow the same rules as a normal edge network, where each edge server connects to the central server and to each other.

The optimal solution provides additional problems with actual implementation. An undefined re-routing algorithm must determine which peering edge servers a client must connect with before the client encounters collisions within the area-of-interest. Further, link saturation for the client may become an issue in highly congested virtual world space if

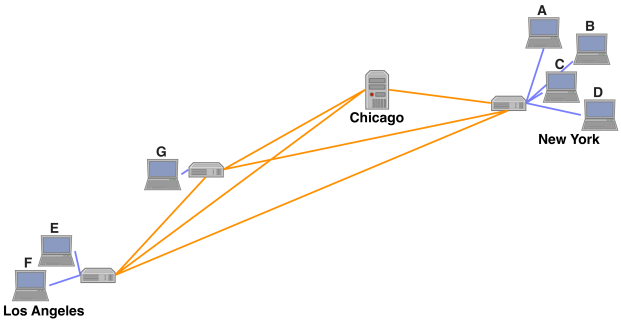

Fig. 3. An example of a typical network topology using peering edge servers for the virtual world in Figure 1. All clients connect to the nearest peering edge server, and all peering edge servers communicate directly with each other and the central server. Geographic location of the clients determines which peering edge server the clients connect through.

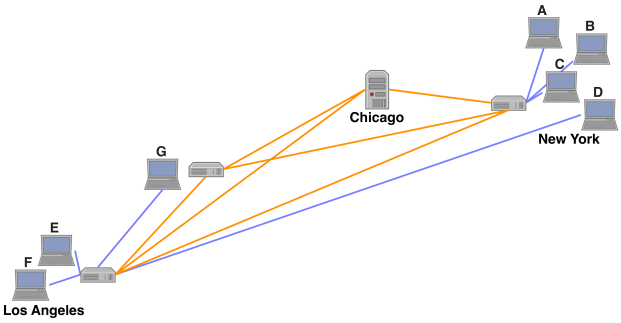

Fig. 4. An example of a typical network topology for peering edge servers with optimized connection for the virtual world in Figure 1. Clients connect to peering edge servers that minimize median ping times between two clients. Clients D and G no longer connect to their nearest peering edge server, and now connect to a peering edge server that minimizes latency between other clients in their area-of-interest.

a client must connect to many peering edge servers and the clients hardware cannot support the traffic load the connections require. However, we only use this metric to evaluate how close our solution is to the optimal, so we are unhindered by these constraints.

We find that edge networks provide potential for significant latency and jitter reduction within massively multiplayer online games. Our work contributes the following to these conclusions:

- A simple evaluation of data center locations in the United States comparing current World of Warcraft data center locations to Google data center locations.
- First consideration of how edge networks can improve client-facing latency in massively multiplayer online games, specifically regarding area-of-interest latency.
- Simulation results that demonstrates the benefits of edge networks with an optimal solution comparison.
- Ideas on how massively multiplayer online games can exploit edge networks to gain lower latency.

II. Edge Networks

Google's Edge Network is a collection of many small servers scattered throughout the world that act as a cache for a central server. The peering edge servers provides a programmable, reliable, and global traffic integrated scalable network [4]. These peering edge servers can communicate with each other and can very quickly switch users from one peering edge server to a different peering edge server as need arises due to server failure or developers necessity.

This network topology provides a similar topology to a peer-to-peer network, but eliminates several of the problems encountered when navigating within peer-to-peer. A peer-to-peer network is a network where each client connects directly to other clients, eliminating the need for a central server. Most peer-to-peer networks still require some sort of connection point where clients can discover other clients, or even utilize partial need of a central server such as in the case of hybrid peer-to-peer networks. Problems of trust, reliability, and hardware limitations cause the avoidance of peer-to-peer networks within massively multiplayer online games [1] [3].

Utilizing peering edge servers allows massively multiplayer games to again explore the benefits of peer-to-peer topologies by allowing trusted nodes within the system that are reliably available and have the hardware capacity for many network connections. Additionally, these nodes give full control to the developers allowing for the benefits of a central server.

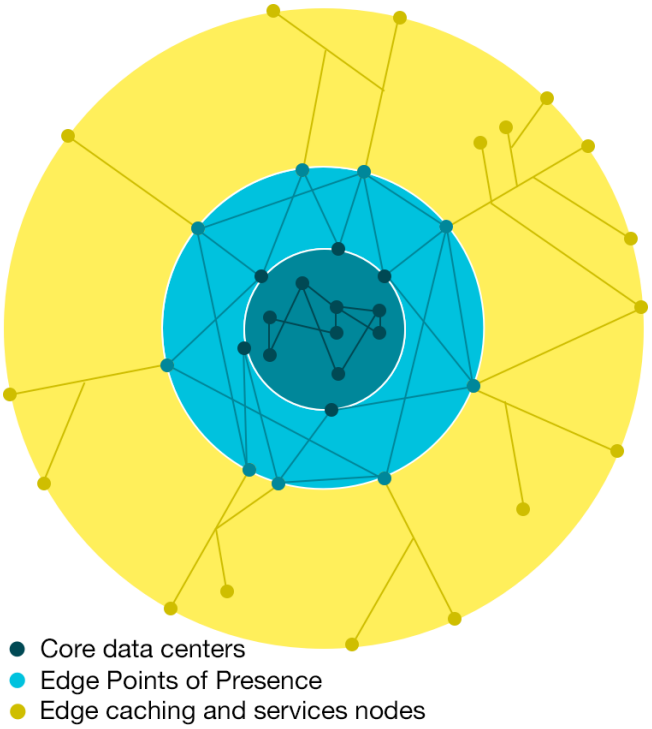

Fig. 5. Google's Edge Network topology diagram from Googles website, demonstrating data centers connecting to edge points of presence, which connect to edge caching and services nodes (peering edge servers) [7].

Figure 5 shows the connection topology used by Google's Edge Network. The inner most circle represents core data centers. These data centers specialize in data replication to all nodes and delivering content to end users that are closest [7]. The middle ring is the edge points of presence. These are global network interconnection points that allow peering edge servers to connect to data centers anywhere in the world. The outer yellow ring is the edge caching and services nodes. These are peering edge servers, or known by Google as *Google Global Cache*. They replicate popular static content for delivery very close to end users.

Google scatters peering edge servers abundantly around the world. Figure 6 shows the locations in the United States. A peering edge server is always close in geographic proximity compared to a traditional network. With a traditional network, a client might need to connect to a server across the country. Yet, Googles Edge Network prevents lengthy connections of that magnitude from happening.

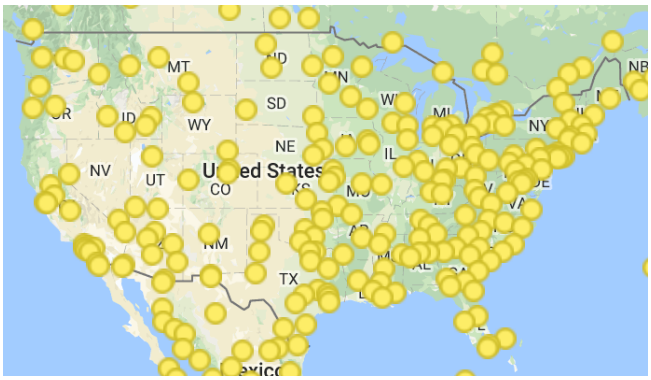

Fig. 6. Map of the United States showing metros where at least one peering edge server exists, obtained from Googles website [7].

Our work focuses on utilizing this outer ring. However, instead of using peering edge servers to cache static content, we explore the benefits of allowing more server control and customizing the behavior of these servers. Allowing the exposure of the programmable functionality of these servers as outlined in previous work [4], we accomplish the changes needed to allow peering edge servers to communicate with each other directly.

III. Traditional Networks

Massively multiplayer online games typically use a traditional network topology consisting of a server and many clients as shown in Figure 2. Although the servers may reside in more complicated topologies, this usually only consist of redundancy protection on the side of the server. Servers reside within deliberately specified geographic locations. As an example, querying the individual server locations using the Battle.net Looking-Glass tool [8], World of Warcraft data centers in the United States are in Los Angeles, California, or Chicago, Illinois. Players choose a location, routing all data to and from one of these data centers depending on the players selection.

To accommodate growing populations of players, companies add additional hardware at these geographic locations, and

assign the virtual worlds maximum populations to account for the limited resources of individual machines. If players are unhappy with the performance of their virtual world, developers encourage players to switch to newer servers, or servers that are geographically closer to the players physical location in the real world. However, players are typically unaware of the proximity of their server selections and often pick servers with locations that are not ideal for gameplay such as the selection of an Eastern Time zone location for a player in New York, but the server itself resides in Los Angeles.

To find the optimal location for a data center in the United States, we took the median city-to-city ping, and adjusted for metropolitan population densities finding both an average and a standard deviation. Figure 7 shows these results for the top ten cities in our data. Both Chicago and Kansas City have very low average ping times and very low standard deviations. These seem to be ideal, however, other factors may affect an ideal data center location such as cost and geographic distribution of customers using internal data. We ignore these other criteria points and focus only on latency reduction for our analysis.

	City	Average	Std Deviation
1	Chicago, IL	30.14	14.85
2	Atlanta, GA	31.19	21.64
3	Secaucus, NJ	32.90	22.18
4	Kansas City, MO	33.45	11.36
5	Detroit, MI	33.52	18.23
6	Manhattan, NY	33.61	24.60
7	New York, NY	34.40	24.52
8	Charlotte, NC	34.80	18.67
9	Pittsburgh, PA	35.67	20.94
10	Cincinnati, OH	35.74	16.76

Fig. 7. Top ten cities with the lowest average latency, in milliseconds, to all cities in our weighted population. The standard deviation shows the variation, in milliseconds, these ping values exhibit within their median ping times.

Using the Battle.net Looking-Glass tool, we found that current World of Warcraft data centers geographically reside in either Los Angeles, California, or Chicago, Illinois [8]. Given our own findings for optimal locations for a data center, Blizzard's choice of Chicago falls into our list of top ten ideal data center locations. Further, this allows us to use Chicago as a reference point when making comparisons within our simulations while keeping a real-world example of a location used by a popular massively multiplayer online game.

Our goal is to maximize the number of clients connected to the virtual world simultaneously on a single server. We show a cumulative distribution function drawn from a distribution based on population density for five thousand clients in a virtual world and their round-trip time (RTT) for latency to the central server if the server is in Chicago or Los Angeles in Figure 8. To better get an assessment of data centers, we also look at Google data center locations to further find an ideal location in the United States.

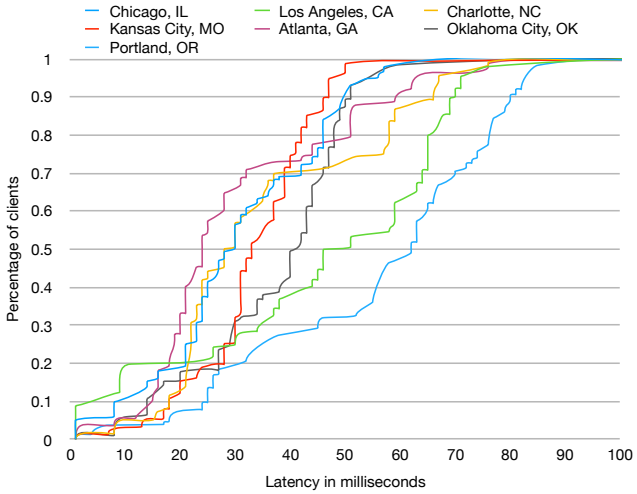

Fig. 8. The latency from World of Warcraft and Google data centers to all other cities in our data.

Google's data centers are more complicated than Blizzards data centers and there are numerous locations beyond the two used by World of Warcraft. Further, a user does not select their server when navigating Google's services such as Google.com or YouTube; they automatically route to the location best selected for their needs. This includes peering edge servers already in use by Google. However, to facilitate our comparison, we select locations in our data that are close to the actual locations used by Google. Google's data centers reside in metropolitan areas that are outside of the top fifty largest locations that we use in our data. However, most are close to larger metropolitan areas, so we adjust their locations to fall within our data [9]. Figure 9 shows where Google's data centers reside, and where we placed their data centers for our simulation.

Google's Location	Simulated Location
Berkeley County, SC	Charlotte, NC
Lenoir, NC	Charlotte, NC
Council Bluffs, IA	Kansas City, MO
Douglas County, GA	Atlanta, GA
Jackson County, AL	Atlanta, GA
Mayes County, IK	Oklahoma City, OK
Montgomery County, TN	Oklahoma City, OK
The Dalles, OR	Portland, OR

Fig. 9. Google data center locations are mapped to simulated locations that are nearest geographically. This may cause a few milliseconds of inaccuracy when simulating these data center locations.

Figure 8 shows the latency for each of the data centers for both Google and World of Warcraft. Since our goal is to pick a single location to best serve the whole United States, we find that Chicago is the ideal location given actual ping data. Both Atlanta and Kansas are strong locations, but Atlanta suffers from the slowest twenty percent of clients, and Kansas suffers with the fastest sixty percent. Overall, we choose Chicago for its constancy in all latency tiers.

IV. Edge Network Solution

For traditional network topologies, we have focused our work on ideal central server locations. In massively multiplayer online games, the virtual world is equally important for optimizing network connections. Figure 1 shows a typical group of interactions that can have profound consequences on optimal network connections. A very important metric used to determine beneficial inter-client interaction of players in the virtual world is **area-of-interest** latency. This is the latency between players in the virtual world when they encounter each other in a meaningful way. Usually, this is a player in the immediate area of another player, which is what we have used in our simulation. In Figure 1, players A, B, and C are all within each other's area-of-interest. Player F is isolated, independent of the behavior of other players. Players D, E, and G have a more complicated interaction where E cares about the other two, while D and G do not need to know about anyone besides E.

To simulate Google's Edge Network, we place a peering edge server in each city within our data. In practice, there are multiple peering edge servers in these cities. We set Chicago, Illinois as the central server location, and assume that each peering edge server can communicate directly. When clients connect to the network, they connect to the peering edge server that is closest geographically, which results in the clients connecting to a peering edge server in the same city.

Although we use a central server in our simulation, the clients never connect directly to this server, and instead always connect to a peering edge server. Because of this, the central server location does not affect area-of-interest latency. Peering edge servers communicate directly, avoiding the central server. However, peering edge servers will still communicate with the central server as needed.

Figure 3 shows a typical network layout for these communications. A client connects to a peering edge server, which connects to another peering edge server, which connects to the desired client in the player's area-of-interest. In some instances, the two clients connect to each other through a single peering edge server reducing the extra hop encountered by most other connections.

As shown in Figure 10, area-of-interest latency is improved significantly with the mean latency improving from 52 milliseconds to 39 milliseconds. Further, the 90% percentile shows improvement form 89 milliseconds to 72 milliseconds, and the 99% percentile shows improvement form 110 milliseconds to 91 milliseconds.

The time taken for the central server to send a message to any client worsens by a couple of milliseconds as shown in Figure 11. This is due to the time taken for a peering edge server to receive the message and forward it to the connected clients. However, we find this loss to be minimal compared to the reductions in area-of-interest latency.

V. Optimal Solution

In creating an optimal solution, we decided to focus on area-of-interest latency at the expense of server to client latency. To accomplish this, we allow each client to connect to many peering edge servers. Ideally, a client only connects to one peering edge server, so a client will move around the network topology as needed. This connection scheme may increase client-side connection states, which might hurt scalability. However, we wanted an area-of-interest comparison that represents the best case, or near best case, as a metric of comparison. Further, we force clients to connect to the minimum number of peering edge servers to help avoid link saturation.

Figure 4 shows a typical case for this network topology when players in the virtual world are positioned such as the players in Figure 1. As players progress into another player's area-of-interest, both clients of these players find a common peering edge server to directly connect. We accomplish this by taking the median ping times from the client's current physical locations in the real world, and find the total lowest estimated

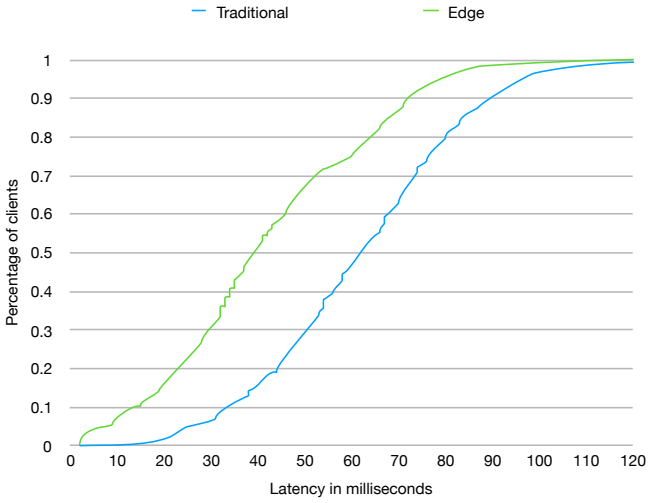

Fig. 10. Area-of-interest latency using Google's Edge Network compared with the latency of a traditional client to server network. Peering edge servers reside within each major metropolitan area.

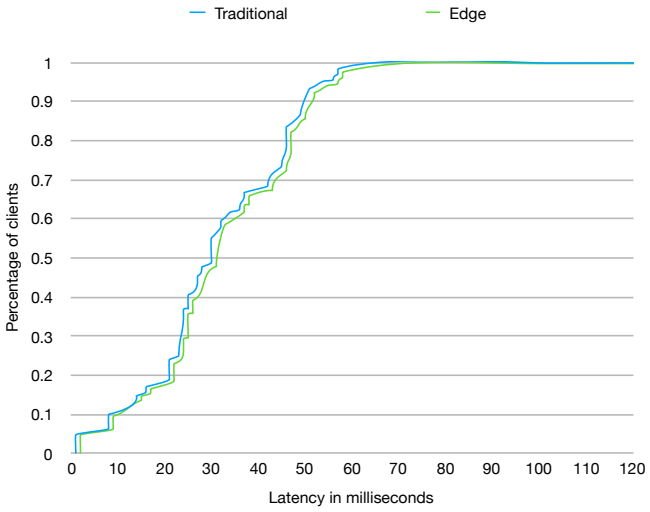

Fig. 11. Latency from the central server to each client for Google's Edge Network compared to a traditional client to server network.

ping between them. Both clients then reconnect to this peering edge server. If additional players are in the area-of-interest, the client will connect to many peering edge servers to connect with all other clients as needed.

As shown in Figure 12 area-of-interest latency improves, especially in the worst cases. The ninetieth percentile shows some of the largest reductions with improvements to latency exceeding ten percent improvements from the previous normal edge network solution. The ten percent of clients with the lowest latency using the previous normal edge network solution see little to no improvement in area-of-interest latency.

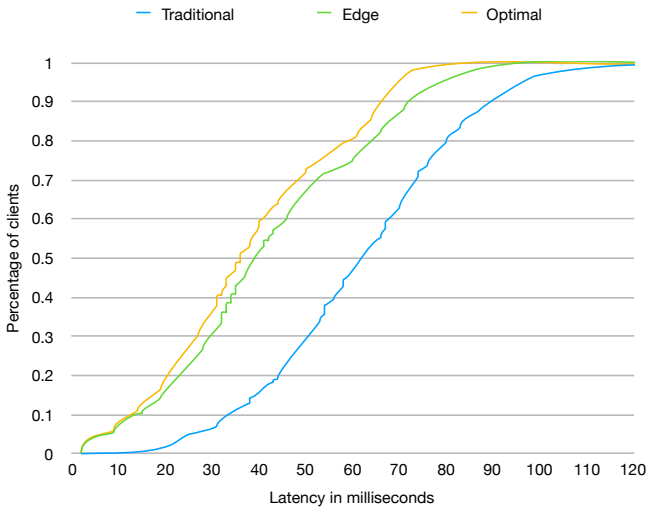

Fig. 12. Area-of-Interest latency including an optimal solution for comparison. Worst-case latency percentages see improvements while best-case latency percentages see little to no change.

Figure 13 shows that server to client latency worsens in every category, especially in the slowest ten percent. This is observed when a client connects to a peering edge server that is geographically distant such as the case in Figure 4, when Client D connects to the peering edge server in Los Angeles while residing within the New York area. However, we can remedy this slowdown by allowing each client to maintain a connection to the nearest peering edge server, thus resulting in the same client to server latency as the previous edge network solution. Adding this additional connection increases the chance of encountering link saturation issues. We can reduce this risk if we limit the number of maximum connections a client will maintain.

Connections to any peering edge server, as shown in Figure 14, also experiences performance problems with the worst several percent seeing latency slower than the traditional network topology. However, the latency from a client to any server improves enormously in the non-optimal edge network solution. Server latency improves if these messages originate from the peering edge servers instead of the central server and every client maintains a connection to the peering edge server within the same city as the clients. We accomplish this by allowing the peering edge servers to maintain replicated game

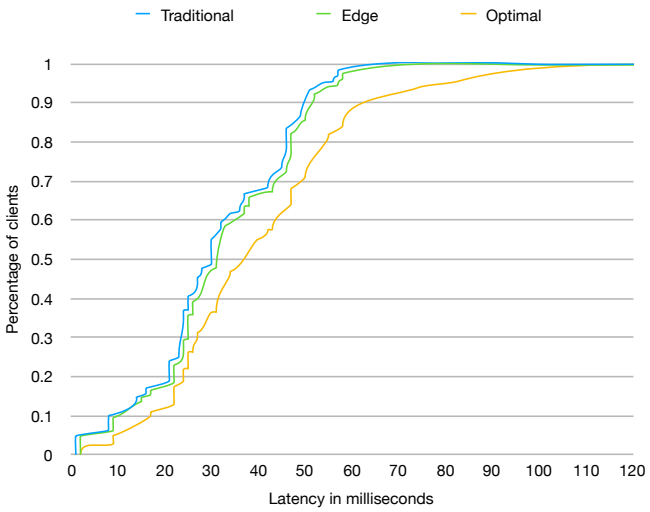

Fig. 13. Client to server latency including an optimal solution for comparison. The optimal solution sees worst latency across the whole graph.

state information that does not need frequent updates from the central server.

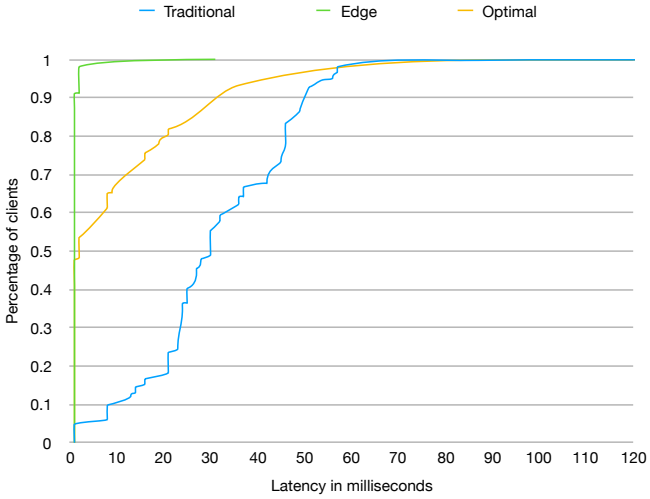

Fig. 14. Latency from a client to any peering edge server. When a client connects to multiple peering edge servers, this shows only the fastest connection times.

To facilitate a more realistic connection topology, we can limit the number of maximum connections a client maintains to peering edge servers. However, as shown in our results of this optimal solution, we feel the optimal solution is a minor improvement over our normal edge solution and believe that the simplicity of that solution is ideal.

VI. Implementation Ideas and Observations

While exploring the benefits of Google's Edge Network and the latency reduction peering edge servers potentially have on massively multiplayer online games, we have formulated several ideas on how to implement this system. We have also made some observations about future work associated with these networks.

A. Message Aggregation Caching

One network development technique we believe benefits from Googles Edge Network is message aggregation. The server can combine multiple game state messages together and send a batch simultaneously at delayed intervals. This technique helps improve performance by reducing the amount of data sent between a client and server or between multiple clients. This form of message optimization benefits massively multiplayer online games, especially when messages are small and frequent [1].

The static caching behavior already present in Googles Edge Network only needs minor modification to adapt to a form of *message aggregation caching*. If we only allow a peering edge server to receive and send updates to the central server every quarter second, we inherently gain a portion of the benefits from our *edge network solution* by allowing players to only go as far as their immediate peering edge server when requesting information.

Peering edge servers always work off a cached version from the server. Local updates push to the server when requesting updates on the quarter second intervals. Using this system peering edge servers connect to the central server, and clients connect to the peering edge server closest geographically. This system can further improve by grouping players in the virtual world with players that connect to the same peering edge server.

If our game design accounts for these message delays, we minimize the negative gameplay experience associated with delayed gaming events. Players may not notice the delays if properly accounted for in the game design [10]. We suggest actions such as spell casting times or longer animations to account for the delays.

B. Game Architecture

As shown in Figure 12, area-of-interest latency between clients is improved greatly when using peering edge servers. However, as shown in Figure 13, client to server latency suffers, especially with the optimized solution. Using this information, developers can gain the most benefit from this system when game logic does not need the servers involvement. Game logic pushed to the peering edge servers will help development see the full potential of these reductions in latency. A more ideal solution may even push some game logic to the clients allowing the peering edge servers and the clients to manage their own states.

Games with a high dependency on the central server may not see benefits from this new system. Increases in latency are minimal, usually less than a couple of milliseconds for messages originating from the server and sent to the client when using peering edge servers. However, if the game has no way of offloading this data, and the central server must control all the game logic, then latency improves by those couple of milliseconds when using a traditional topology already in use by most massively multiplayer online games.

It is possible that a game utilizes both a high central server dependency and still has opportunities to offload to a peering edge server to realize the gains from both setups. In World of Warcraft, a player may spend hours playing mostly alone in the virtual world completing quests. This gameplay may benefit from pushing this content to a peering edge server, but may also benefit from keeping the needed state information on the central server since there is not a lot of interaction with other players. However, the game's player-vs-player battlegrounds, where players battle each other in teams of up to forty, have many area-of-interest interactions. Most of the messages in this system originate from the clients. This portion of the game code is a prime candidate for the latency reductions found by offloading this gameplay to the peering edge servers. This gameplay further benefits from this shift to peering edge servers because the isolation each player-vs-player battleground experience from the rest of the virtual world during the events of the gameplay. This enables an independent game state moved to a different remote server instead of handling the complexities of integrating the entire game itself.

To align this architecture with our research, this system can borrow the gains realized by using a version of our optimized solution. During the duration of a player-vs-player battleground, the central server chooses a peering edge server location that is most ideal to each of the players. After the event concludes, all the clients move back to their desired connection either through another peering edge server, or connecting to the central server itself.

C. Peer-to-Peer Trust

Related work shows the benefits from offloading game code to peers in peer-to-peer networking systems [3]. Over the past decade, these systems have shown great potential but peer-to-peer systems end up abandoned. A major factor in this decision has been trust of peers within the system.

Developers spend major efforts to insure cheating is not present in massively multiplayer online games. Offloading code to clients increases the risk that player data changes by means outside of the control of the developers. Peering edge servers provide a secure server to offload game code and player data without the dangers usually present with such an operation. Byzantine behavior, intentional or unintentional, is ever present with massively multiplayer online games [11]. Malicious clients can cause problems for players within the virtual world by creating denial of service attacks or attempting to hack other clients machines to gain personal information.

Algorithms such as virtual world positioning, when offloaded to the peering edge servers, improve response time between players in area-of-interest interactions. Yet, this new peering edge server network does not compromise the game system and reduces concerns of cheating and byzantine behavior.

An additional benefit of Google's Edge Network is the ability to completely replace comprised code on the peering edge server and perform a full update to the infected machine without bringing down the entire system of peering edge

servers [4]. Clients will quickly connect to a new peering edge server with little to no impact on gameplay.

Overall, this new system of peering edge servers gives new potential to ideas of trust in peer-to-peer style networks. We feel that we can reevaluate peer-to-peer algorithms abandoned do to trust issues.

VII. Methodology

Using WonderNetwork [12], we collected ping data every hour for eight weeks from the beginning of April 2017 to the middle of June 2017. After removing some problematic entries, and focusing on cities within the top fifty metropolitan areas in the United States, we collected 4,457,055 total ping entries for fifty different cities. This data contains thousands of pings between each of these cities. Although the precisions of these pings are to the thousandth of a millisecond, we round the pings to the nearest millisecond.

We have removed all self-pings from the original WonderNetwork data, since their representation from a machine to the same machine is insufficient for inter-city tests. These pings are very small percentages of what an actual ping within the same city will yield. Instead, when we need a ping from a city to the same city, we use data collected from Manhattan, NY to Secaucus, NJ, because these two cities are the closest geographically to each other in our data.

Using data from the 2016 annual estimates for resident populations in the United States [13], we created population distribution representing connection behavior. This started with the fifty largest metropolitan areas, and then we added cities from our ping data to represent these geographic locations. We eventually ended with fifty cities representing the fifty largest metropolitan areas in the Unites States. Figure 15 describes how these metropolitan areas converted to our ping data, and what population was assigned to each city. Multiple cities in the same metropolitan area have populations split evenly from the Census data. We distribute metropolitan areas with no city ping data to the closest cities or metropolitan areas to create an accurate estimate.

We remapped the population of several metropolitan areas to nearby metropolitan areas due to lack of city ping data within the original metropolitan area limits. The population of the *Riverside-San Bernardino-Ontario* metropolitan area is added to the *Los Angeles-Long Beach-Anaheim* metropolitan area. We divide the population of the *Nashville-Davidson–Murfreesboro–Franklin* equally and add their populations to *Memphis* and *Knoxville* metropolitan areas. The population of the *Virginia Beach-Norfolk-Newport News* metropolitan area is added to the *Washington-Arlington-Alexandria* metropolitan area. We take the population of the *Providence-Warwick* metropolitan area and add to the *Boston-Cambridge-Newton* metropolitan area. The population of the *Milwaukee-Waukesha-West Allis* metropolitan area is added to the *Chicago-Naperville-Elgin* metropolitan area. The population of the *Richmond* metropolitan area moves to the *Washington-Arlington-Alexandri* metropolitan area. Finally, the population of the *Birmingham-Hoover* metropolitan area is added to the *Atlanta-Sandy Springs-Roswell* metropolitan area.

When we add a new client to our simulation, we use the populations given to each city to randomly select a city as the destination. We give each city a probability based off their population against the total population. We then add the clients to the virtual world, and these new players start to interact with other players. Initially, we send players in a random direction in the virtual world and their locations may modify when they interact with other players or encounter the edge of the game space. We only analyze the movement game states, since all aspects of play in a massively multiplayer online game primarily use movement [14].

Using this data set, each city-to-city location has an array of several thousand real world pings. To simulate a ping from one location to another, we randomly access an element from the array using a uniform random distribution. When the city-to-city location is the same city for both the source and destination, we use Manhattan, NY to Secaucus, NJ.

We then perform data collection for the three network topologies evaluated. This is the traditional network topology, the edge network topology, and the optimized edge network topology. We leave the virtual world players and positions unchanged between the data collection of the three topologies to gain a more accurate evaluation of how these changes in topology compare. Similarly, we also leave the positions within the physical world unchanged, and reconnect the clients as needed for the new topology data collection.

VIII. Conclusions

We have shown improvements to latency between players in a virtual world using Google's Edge Network. We presented the traditional model used currently, a model that connects clients to their closest peering edge servers, and an optimized solution where clients move to different peering edge servers to reduce latency between players in the virtual world. Of these three, we conclude that clients connecting to their nearest peering edge server results in the simplest implementation while maintaining low latency from the central server and other clients. The non-optimized edge network solution also provides a simple path to numerous suggestions we have presented for game architecture consideration.

The optimized solution presents potential when client to server latency is less important and we isolate game state information to individual peering edge servers for extended portions of gameplay. Although this system appears to be more complicated, game development decisions may vastly simplify those complications further increasing the benefits gained from this system.

We gain the benefits of peer-to-peer network topologies when using Google's Edge Network, while maintaining the security and control required by modern game developers. This work only focuses on the latency requirements due to the importance of this metric [5] [6]. Future work needs to strengthen this model by creating the message system needed to allow gameplay. We assume that Googles system

handles data synchronization between peering edge servers and the central server [4], but further work could optimize this synchronization and utilize the clients as members of this network topology.

Overall, our improved results excite us to the possibilities that peering edge servers provide for future work.

REFERENCES

[1] J. L. Miller and J. Crowcroft, "The near-term feasibility of p2p mmog's," in *Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on*. IEEE, 2010, pp. 1–6.

[2] J. S. Gilmore and H. A. Engelbrecht, "A survey of state persistency in peer-to-peer massively multiplayer online games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 818–834, 2012.

[3] A. Yahyavi and B. Kemme, "Peer-to-peer architectures for massively multiplayer online games: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 9, 2013.

[4] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain *et al.*, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 432–445.

[5] T. Fritsch, H. Ritter, and J. Schiller, "The effect of latency and network limitations on mmorpgs: a field study of everquest2," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005, pp. 1–9.

[6] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[7] "Google edge network," https://peering.google.com/#/infrastructure/.

[8] "Battle.net looking-glass," http://us-looking-glass.battle.net/.

[9] "Google data center locations," https://www.google.com/about/datacenters/inside/locations/index.html.

[10] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?" *Communications of the ACM*, vol. 49, no. 11, pp. 34–38, 2006.

[11] P. Kabus and A. P. Buchmann, "Design of a cheat-resistant p2p online gaming system," in *Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*. ACM, 2007, pp. 113–120.

[12] "WonderNetwork network testing solutions," https://wondernetwork.com/.

[13] "Annual estimates of the resident population: April 1, 2010 to july 1, 2016," U.S. Census Bureau, Population Division, Mar. 2017.

[14] M. Suznjevic and M. Matijasevic, "Player behavior and traffic characterization for mmorpgs: a survey," *Multimedia systems*, vol. 19, no. 3, pp. 199–220, 2013.

	Metropolitan Area	Population	Cities
1	New York-Newark-Jersey City	20,153,634	New York Manhattan Piscataway Secaucus
2	Los Angeles-Long Beach-Anaheim	13,310,447	Los Angeles
3	Chicago-Naperville-Elgin	9,512,999	Chicago
4	Dallas-Fort Worth-Arlington	7,233,323	Dallas
5	Houston-The Woodlands-Sugar Land	6,772,470	Houston
6	Washington-Arlington-Alexandria	6,131,977	Washington Olney
7	Philadelphia-Camden-Wilmington	6,070,500	Philadelphia Wilmington
8	Miami-Fort Lauderdale-West Palm Beach	6,066,387	Miami
9	Atlanta-Sandy Springs-Roswell	5,789,700	Atlanta
10	Boston-Cambridge-Newton	4,794,447	Boston Salem
11	San Francisco-Oakland-Hayward	4,679,166	Fremont San Francisco
12	Phoenix-Mesa-Scottsdale	4,661,537	Phoenix
13	Riverside-San Bernardino-Ontario	4,527,837	*Remapped*
14	Detroit-Warren-Dearborn	4,297,617	Detroit
15	Seattle-Tacoma-Bellevue	3,798,902	Seattle
16	Minneapolis-St. Paul-Bloomington	3,551,036	Minneapolis
17	San Diego-Carlsbad	3,317,749	San Diego
18	Tampa-St. Petersburg-Clearwater	3,032,171	Tampa
19	Denver-Aurora-Lakewood	2,853,077	Denver
20	St. Louis	2,807,002	St Louis
21	Baltimore-Columbia-Towson	2,798,886	Baltimore
22	Charlotte-Concord-Gastonia	2,474,314	Charlotte
23	Orlando-Kissimmee-Sanford	2,441,257	Orlando
24	San Antonio-New Braunfels	2,429,609	San Antonio
25	Portland-Vancouver-Hillsboro	2,424,955	Portland
26	Pittsburgh	2,342,299	Pittsburgh
27	Sacramento-Roseville-Arden-Arcade	2,296,418	Sacramento
28	Cincinnati	2,165,139	Cincinnati
29	Las Vegas-Henderson-Paradise	2,155,664	Las Vegas
30	Kansas City	2,104,509	Kansas City
31	Austin-Round Rock	2,056,405	Austin
32	Cleveland-Elyria	2,055,612	Cleveland
33	Columbus	2,041,520	Columbus
34	Indianapolis-Carmel-Anderson	2,004,230	Indianapolis
35	San Jose-Sunnyvale-Santa Clara	1,978,816	San Jose
36	Nashville-Davidson–Murfreesboro–Franklin	1,865,298	*Remapped*
37	Virginia Beach-Norfolk-Newport News	1,726,907	*Remapped*
38	Providence-Warwick	1,614,750	*Remapped*
39	Milwaukee-Waukesha-West Allis	1,572,482	*Remapped*
40	Jacksonville	1,478,212	Jacksonville
41	Oklahoma City	1,373,211	Oklahoma City
42	Memphis	1,342,842	Memphis
43	Raleigh	1,302,946	Raleigh
44	Louisville/Jefferson County	1,283,430	Louisville
45	Richmond	1,281,708	*Remapped*
46	New Orleans-Metairie	1,268,883	New Orleans
47	Hartford-West Hartford-East Hartford	1,206,836	Cromwell
48	Salt Lake City	1,186,187	Salt Lake City
49	Birmingham-Hoover	1,147,417	*Remapped*
50	Buffalo-Cheektowaga-Niagara Falls	1,132,804	Buffalo

Fig. 15. List of the top fifty metropolitan areas mapped to cities with ping data from WonderNetwork. Each city has an equal portion of the population of the given metropolitan area. We remapped some metropolitan due to lack of gathered ping data for cities within their limits.