Distributed VR: An Analysis of Inter-server Traffic Through a LAN

Mahad Ali and Murat Yuksel
Electrical and Computer Engineering
University of Central Florida, Orlando, Florida 32816
E-mail: ma649596@ucf.edu, murat.yuksel@ucf.edu

***Abstract*—Immersive Virtual Reality (VR) applications demand low network latency, large bandwidth, and substantial computational resources. Despite significant progress in addressing these challenges, creating Distributed VR environments remains complex. Existing VR deployments are predominantly centralized. Extending VR to a distributed setup requires solving scalability challenges of the network support needed for VR servers distributed across a network. In particular, the scale of traffic between distributed VR servers and the interaction of this VR traffic's size with various features of the VR applications are unexplored. In this study, we present and evaluate a distributed multi-server VR environment based on Mozilla's popular open-source platform, Hubs, on a local area network (LAN). By conducting traffic measurements, we evaluate how the network traffic volume to support such distributed VR setups may evolve. Our work assesses the feasibility of creating such distributed VR environments. We find that the inter-server traffic exhibits logarithmic increase with respect to the client count when the clients make human-like movements, pointing to the scalability potential of Distributed VR environments. Additionally, the study lays the foundation for future optimizations, aiming to enhance the distributed VR experience for users.**

I. Introduction

Immersive VR applications offer users rich, interactive environments mirroring the real world. These environments, however, pose formidable challenges in delivering seamless user experiences, primarily due to their stringent network latency requirements to make sure the VR user does not get dizzy [1], large network bandwidth requirements [2] to transfer volumes of rendered 360-view to the VR headsets, and high computational resources for the VR server performing 3D rendering tasks. Addressing these challenges becomes more complex when attempting to create VR-based environments with increased user load and dispersed users. Despite significant advancements in VR technologies, developing distributed VR environments remains an ongoing research endeavor. Traditional client-server architectures, while widely used, may struggle to maintain responsiveness and scalability as the number of concurrent users increases. To overcome these limitations, we explore the potential of distributed multi-server architectures for VR applications in a local environment.

In this study, our primary objective is to enhance VR experiences by harnessing the potential of distributed multi-server setups built upon the widely popular Mozilla Hubs' [3] open-source VR platform. Our proposed system is especially suitable for local or metro area network environments due to the low latency requirements between servers. By embracing this innovative architecture, we aim to significantly improve overall system performance, reduce latency, and create a seamless and immersive user experience within shared virtual spaces. Notably, the distributed multi-server approach offers the advantage of locating servers closer to users and dividing the workload, thereby allowing for hosting more clients and reducing client-server latency compared to the conventional central server approach due to a smaller round trip time between the VR user and the closest VR server. While Hubs is a social VR platform, the approach we mention can be applied to other types of VR environments as well. We focus on optimization in data transmission, ensuring faster and more responsive interactions, enhancing the realism and immersion for users.

The central goal of our research is to provide a technical solution to distributed VR that is tailored towards local area usage. We propose a setup that utilizes concepts that are widely used in global systems, such as the Publisher/Subscriber model, for synchronization in a local area VR environment. We evaluate the effectiveness of this distributed multi-server VR, focusing especially on the server-to-server synchronization and scalability. To achieve this, we conducted experiments measuring traffic patterns and data exchanged between servers as the number of users within a shared virtual room increased.

Our findings present insight into the potential of distributed multi-server architectures for scalable VR experiences. We showcase the advantages of efficient data synchronization mechanisms, which help optimize the system's performance under high user concurrency. The contributions of this research extend beyond a simple evaluation of the proposed architecture. We introduce a novel approach for creating distributed VR environments, providing valuable data and empirical evidence to support the viability of this setup in real-world applications. By offering a foundation for future optimizations, our work seeks to inspire further research and development to enhance the VR experience for users worldwide.

In the following sections, we delve into the setup & methodology used for our experiments, present our measured data, and discuss the implications of our findings. We conclude by highlighting the potential impact of distributed multi-server architectures on the future of scalable and responsive VR applications.

II. Background and Related Work

VR applications demand stringent network latency to ensure a smooth and immersive experience for users. Previous studies have established that the latency requirement for most immersive VR applications is around 20 ms [4]. Conventional threshold inter-frame time for VR headsets is 15 ms, which includes the rendering of frames at the VR server, their transfer to the VR headset, and the headset's processing time. Increased latency can lead to motion sickness, making it crucial to address latency issues effectively.

To tackle these latency requirements, researchers have explored various optimization techniques. For instance, some works have focused on using viewport prediction and leveraging machine learning techniques to pre-render VR viewports [5]. Additionally, [6] conducted a comprehensive measurement study on five widely-used VR worlds, analyzing network measurements, frames per second, and optimizations. The study revealed that most VR environments lack scalability, server-side rendering, and adequate graphics optimization, highlighting the need for further improvements. The VR environments assessed in [6] include Worlds [7], VR Chat [8], AltSpaceVR (now discontinued) [9], Hubs [3], and Rec Room [10]. Furthermore, [6] conducted an in-depth analysis of these environments, identifying common limitations such as static backgrounds, pre-downloaded backgrounds, simplistic avatars, and limited user interaction gestures. Anycast, a technique used to assign users to the closest servers, was found to be underutilized across these platforms.

There have been several previous architectural studies on Distributed VR. DIVE [11] operates on a peer-to-peer (P2P) architecture where synchronization is achieved through a distributed event system. Another approach involving a P2P VR setup was published in [12]. Although this approach aims to reduce the overhead of client-server communication, it ignores the benefits of server-side rendering, poses higher security risks, and faces challenges in terms of scalability. A P2P approach would scale quadratically as the number of users in the VR system increase. Also, server-side rendering has the potential to improve the VR experience by enabling wireless and lightweight VR headsets and reducing latency through faster pre-rendering.

Client-server designs also received notable attention. MASSIVE [13] manages multi-user interactions through the spatial mode of interaction where a client-server architecture is used, and each server manages an area-of-interest, thereby reducing the amount of data being synchronized between the servers. NSPNET [14] focuses on large-scale, networked virtual environments primarily for military simulation. It also employs a distributed client-server architecture, with each server being responsible for a different segment of the virtual environment. These existing works provide insights into different distributed architectures and their application domains, laying a foundation for further exploration in this area. Hybrid architectures utilizing both P2P and client-server components were also considered. [15] proposed a hybrid design to handle large virtual worlds, including military simulations, but without any empirical results in an actual VR environment.

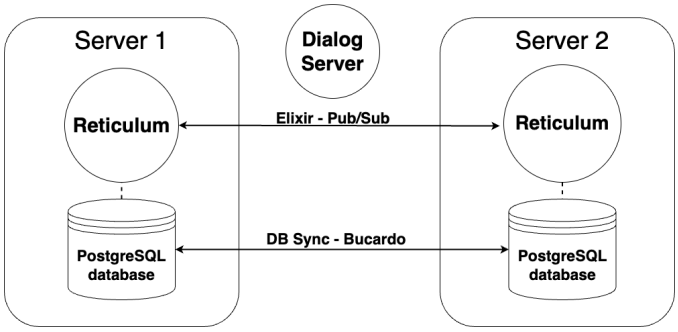

Fig. 1: Overview of Hubs' architecture and the server synchronization components.

Most of the prior studies relevant to distributed VR have focused on creating globally distributed virtual environments. While globally distributed virtual environments remains a complex challenge, there are distributed networking techniques that can help aid in the creation of virtual environments hosted over local- or metro-area network. Despite various attempts to address VR latency, there is a noticeable gap in research evaluating distributed multi-server VR approaches. Our study aims to fill this gap by providing a detailed analysis of network measurements and synchronization-related traffic between servers in a distributed multi-server architecture. By conducting a comprehensive evaluation of a distributed multi-server VR setup, our research seeks to contribute to the growing body of knowledge on VR latency reduction and enhance the overall VR user experience.

III. Distributed VR Architecture

We utilize Mozilla's Hubs, a widely-used open-source VR platform. Whilst Hubs was designed to be a global VR environment, we adapt it to support a locally distributed VR environment. Our choice of Hubs is driven by its wide adoption and open-source nature, which allows for customization and exploration of the underlying architecture.

Hubs is a social VR application that allows users to create rooms, invite other users to join rooms, and interact with other. Hubs employs a web-based client-server architecture, featuring two main server-side components:

- *Reticulum*: The primary Elixir/Erlang server that utilizes the Phoenix framework.
- *Dialog*: A NodeJS server which handles audio data using the mediasoup framework and WebRTC.

On the client side, Hubs utilizes ReactJS (a popular client-side web framework), and a-frame (a JavaScript VR framework built on top of WebGL) [16] to manage graphics processing, enabling users to access the environment through web browsers or VR headsets. Hubs stores persistent state data (e.g., room names, authentication, and avatar information) inside a PostgreSQL database. The communication of real-time audio data between clients and servers employs the WebRTC protocol, utilizing the UDP protocol for transport. Meanwhile, control information, such as user profiles and names, is exchanged via HTTP requests, while the transmission of real-

time avatar coordinates and orientation within the VR world space is achieved through TCP websocket connections.

Hubs has the capability of hosting multiple *Dialog* servers. Hubs also has the capability of hosting multiple *Reticulum* servers. However, one room in Hubs can be hosted by one server only. This bottlenecks the number of users that are able to join a room. A solution to this problem is to implement mechanisms within Hubs' architecture to support multiple clients. Including another server, however, would mean that the servers would need to synchronize information related to clients. While this problem is hard to solve in a globally distributed setup due to network bottlenecks and the stringent latency requirements in VR, we have more options when solving it for environments spanning smaller geographical span such as local and metro-area networks.

To facilitate this transfer of synchronization data, including user orientation information, across servers, we employ the Publisher/Subscriber (Pub/Sub) feature and cluster management capabilities provided by Elixir. Our strategy prioritizes the use of Elixir's built-in functionalities owing to their advantages in terms of low latency, adaptability, and minimal resource consumption. This method ensures that all server hosts are interconnected, maintaining these connections through protocols such as gossip protocols based on UDP or the Erlang Port Mapper Daemon (epmd) [17] based on TCP. For our experiments, we utilize epmd due to its reliability and robustness. The integration of additional server hosts into the system is streamlined, underscoring the scalability of our setup. Instead of depending on external messaging queue systems such as RabbitMQ [18], we leverage Elixir's Pub/Sub and clustering modules. Each message received by a server pertaining to a particular room is sent to all the connected servers that are subscribed to the room.

We also factor in the synchronization of the PostgreSQL database. As we show in our evaluation section later on, the amount of database data that needs to be synchronized is very low when compared to the volume of coordinate/orientation data synchronization. We employ Bucardo [19], a database replication tool, for multi-master PostgreSQL database replication. We choose Bucardo over other approaches since it offers multi-master replication alongside reliable conflict resolution. More popular tools such as PostgreSQL's native support for synchronization do not offer multi-master replication. Using Bucardo ensures that crucial information, including room details, user profiles, and logs, remain consistent and available on both servers.

Since *Dialog*, used by Hubs for audio data transfer, already has the capability for horizontal scaling, we do not perform experiments concerning synchronization of audio data and shift our focus more towards synchronization related to user movement.

Figure 1 provides an overview of our architectural setup, as well as the synchronization techniques employed to ensure seamless communication between the servers and the clients. The figure shows the *Reticulum* server, the PostgreSQL database, and the *Dialog* server. The *Dialog* server is shown separately in the figure since clients connect with the *Dialog* servers directly. The *Dialog* servers facilitate WebRTC connections between clients for audio communication. The *Dialog* server can be run on the same instance as the *Reticulum* server, or it can be run separately. There can be multiple *Dialog* servers for horizontal scalability. The PostgreSQL database is managed by the *Reticulum* server. This comprehensive setup allows us to evaluate the feasibility and effectiveness of a distributed multi-server VR environment, assess the traffic patterns between servers, and identify potential areas for optimization and performance enhancement. The following sections detail our evaluation of the proposed setup, the implications of our findings for the future development of scalable VR experiences, and potential limitations of our work.

```
// Select the avatar using query selector
const entity = document.querySelector('#avatar-rig')

// Set the initial position to (0, 0, 0)
let positionX = 0;
let positionY = 0;
let positionZ = 0;

// Set the movement speed
const movementSpeed = 0.1;

// Function to update the position
function updatePosition() {
  // Update the avatar position
  positionX += movementSpeed;
  positionZ += movementSpeed;

  // Update the position of the entity
  entity.setAttribute('position', {
    x: positionX,
    y: positionY,
    z: positionZ
  });

  // Call recursively before each frame
  requestAnimationFrame(updatePosition);
}
// Call the function to start the movement
updatePosition();
```

Algorithm 1: JavaScript code that automates the movement of a client avatar.

IV. Evaluation

We conducted a comprehensive evaluation of our distributed VR setup, focusing on measuring the traffic between the two servers across different configurations.

A. Experimental Setup

We configured two VR servers that are hosted over the same LAN, and we utilized Wireshark to perform precise traffic measurements. Each server hosted a *Reticulum* instance, a PostgreSQL database, and a webpack server to host the client VR app. One of the servers included a central *Dialog* server. Our evaluation is centered on data synchronization related to movement and orientation, crucial aspects of the VR experience.

To evaluate how the traffic between the two servers scales with increasing client connections, we conducted a series of runs, each spanning 250 seconds. These runs involved varying

numbers of clients connected to the servers. We consider two scenarios, one where the number of clients connected to both the servers increases, and another where the number of clients connected to only one of the servers increases whilst the other server has only one client connected to it. In both cases, the maximum number of clients connected to a server is 12. To ensure consistent conditions, we automated client movement, updating user coordinates each frame with a script. The script ensured that clients moved at a constant speed of 0.1 units per frame. This automation script was executed directly in the browser's console window. The movement was random, such that the clients kept moving in the x and z planes without following any specific pattern. This random movement is the worst-case scenario where the client continuously sends server requests, requiring synchronization. The script used is shown in Algorithm 1. The script runs in the client's browser console. It uses a-frame, an open-source framework that Hubs uses for its VR rendering.

Since users in the real-world do not move like the motion described in the script shown in Algorithm 1, we also perform an experiment where we simulate human-like movement by using the MineRL [20] dataset, which is a dataset comprising of actions that players perform while playing the game Minecraft [21]. Minecraft [21] is a popular sandbox game consisting of a large 3-dimensional environment where users can perform various tasks. MineRL [20] contains user activity data, including user movement, for various scenarios within the Minecraft world. To simulate human-like movement, we utilize data from a small subset of the activities in MineRL to make the users move using a script running in the user's browser. This script is a modified version of the script shown in Algorithm 1. We compare the synchronization data traffic under this scenario with the random movement scenario for up to 12 clients.

Measuring synchronization traffic between the two servers necessitates accounting for the Frames Per Second (FPS) of clients, a metric closely linked to rendering capabilities. Avatar coordinate updates occur frame by frame, requiring server s to transmit coordinates for client c at an FPS f every $1/f$ seconds to relevant servers for synchronization. Inaccurate traffic measurements could arise from FPS variations in different experiments. To ensure consistent FPS rates across clients, we adopted a methodology where all clients accessed the VR environment through individual browser sessions hosted on virtual operating systems running on the same machines throughout the experiment. This approach maintained synchronized client movement and FPS, ensuring consistent parameters for synchronization-related traffic measurements, irrespective of system load or user activity fluctuations.

B. Results

We start by dissecting and analyzing the synchronization traffic between the two *Reticulum* servers, database synchronization, and the incoming traffic to the *Dialog* server. As stated earlier, we mainly focus on traffic synchronization related to movement of the clients within the VR room. As such, all clients in the VR room have their audio muted during the experiments. Since all audio communication happens via the WebRTC protocol and connections are maintained by a mediasoup-based server (Dialog), we see incoming traffic to Dialog, albeit not very significant, despite the clients being muted. This is because Hubs keeps connections between the clients and the servers alive, which requires some bandwidth. For the database related synchronization, the traffic between the VR servers is almost negligible. Figure 2 shows a throughput comparison over 250 seconds, with one client connected to each server (2 clients, total).

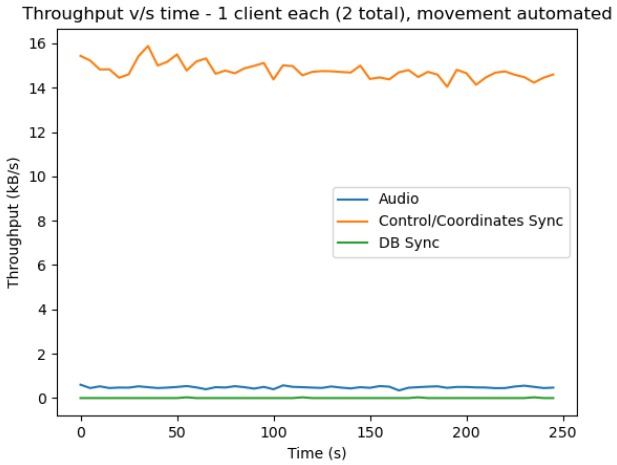

Fig. 2: Dissecting synchronization data traffic between the two servers over 250 seconds, with 1 client connected to each server.

To measure scalability of our distributed VR system, we measure the traffic between the two servers against the number of clients. Our analysis, depicted in Figure 3, reveals a linear trend in server-to-server synchronization throughput between the two servers as the number of clients connected increases. The figure shows two scenarios, one where the number of clients connected to each server is increased, and another where the number of clients connected to only one of the servers is increased (the other server has only one client connected to it at all times). In both of these scenarios the movement is automated using the script shown in Algorithm 1. We observe that the rate at which the throughput increases when the number of clients connected to both the servers is varied is 1.9 times higher than when the number of clients connected to only of the servers is increased. In both of the cases, we observe a linear trend in throughput. This correlation highlights user additions' consistent impact, offering insights into the system's scalability. This trend aligns with the observations made in [6] where a central server architecture was used to show that an increase in users resulted in a linear upsurge in downlink throughput per user.

Whilst the experiments in Figure 3 were performed using a constant speed and movement such that updates to the clients' coordinates were made before the rendering of each frame, we also perform an experiment where we simulate movement that is closer to how real users would act inside a virtual world.

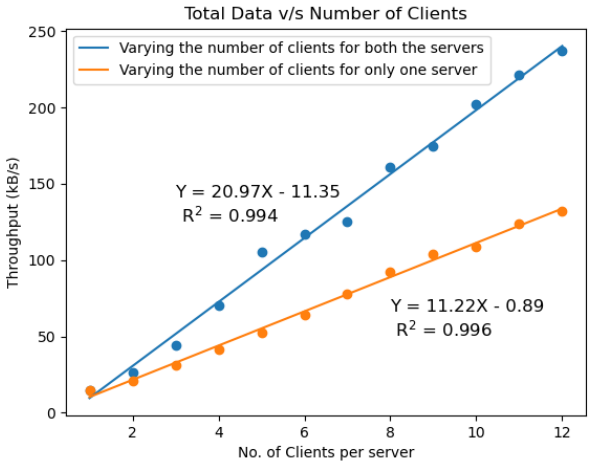

Fig. 3: Average synchronization throughput between the two servers over 250 seconds vs. the number of clients.

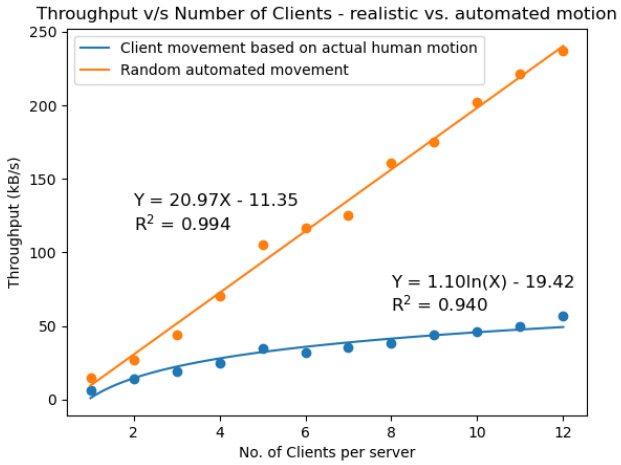

Fig. 4: A comparison of the server-to-server throughput when the client movement mimics real-user movement vs. when the movement is random.

This movement was based on a subset of the activity data from the MineRL [20] dataset. We collect synchronization traffic data under this scenario for up to 12 users connected to each server, and compare the average throughput with the aforementioned random movement scenario. Figure 4 shows this comparison. As can be seen from Figure 4, there is a significant difference in the average throughput between the two scenarios. The movement based on real users follows a sub-linear trend, reminiscent of a logarithmic curve. This is because real users are seldom constantly moving, and usually have movement that is more sporadic and spatially localized. This trend implies that the increase in traffic would reduce as the number of clients connected to the servers increases, as opposed to the linear trend where the addition of clients adds a constant load on the system. This is a promising outcome, as it shows that the multi-server distributed VR system can scale well in terms of inter-server traffic as the number of clients increases.

Comparing the trends in these findings with [6] and observing the similarity between the trends in central and distributed server traffic, it becomes evident that there is minimally added overhead from synchronization processes in the distributed approach. Also, the logarithmic relationship between inter-server traffic data and the number of clients when the clients mimic movement that is closer to that of real users suggests that the system would scale well as more clients are added.

[6] also highlights the potential for optimization strategies such as sending coordinates based on Field-of-View (FoV) and spatial positioning, as opposed to broadcasting to all users. Implementing similar optimizations in our system would mitigate the effect of client numbers on inter-server traffic, thus enhancing scalability. Also, making our system synchronize only the relevant data between the two servers instead of all the data would further curb the logarithmic growth that we observe in Figure 4.

The identified trends show the significance of effective load balancing strategies within our distributed multi-server VR environment. By equitably distributing user traffic across servers, we can ensure consistent performance and responsiveness, especially during periods of increased user activity. Our findings can be used as a foundation to further improve and optimize the performance of the distributed VR system. We briefly look at some of the potential optimizations that can be performed as part of the future work to further improve the multi-server architecture in terms of scalability, load balancing, latency, and facilitating server-side rendering. These advancements hold the potential to elevate the VR experience, offering users seamless interactions within immersive virtual environments.

V. Limitations

While a distributed multi-VR server architecture offers a more scalable approach to a VR world system, it does have some disadvantages compared to the central server approach. A significant drawback is the overhead from server synchronization in distributed setups. Real-time server-side synchronization consumes a significant amount of data, which can impact the overall performance of the system. In contrast, a central server architecture involves data transmission only between clients and the central server, which then broadcasts the data to all clients, potentially reducing the overall data traffic.

Another limitation of our study is that we only tested the setup with two servers. Although we demonstrated that the traffic between servers scales sub-linearly as the number of connected clients increases under scenarios where user movement is realistic, testing the architecture with more servers deployed over a larger area could provide more insights. Conducting a study over a wider area, such as in a MAN area, may present additional challenges such as network latency and potential communication delays between distant servers. It is worth noting that, within real-time applications like Mozilla Hubs and many other VR metaverses, the impact of minor packet loss due to network delays is not markedly detrimental to the overall user experience. This parallel can be drawn

from real-time multiplayer games, wherein marginal packet loss typically has limited repercussions on game play quality.

VI. Future Work

The distributed multi-server architecture we explored in this research demonstrates substantial promise for the evolution of VR metaverse applications. While our experiments highlight the benefits of reduced client-to-server latency due to smaller round trip time and studies scalability of the multi-server distributed VR system, there are still avenues for further exploration and improvement.

Future work in this field could focus on developing more sophisticated prediction algorithms to better anticipate user actions and optimize data transmission between servers. Using techniques such a FoV and selectively synchronizing only the required state between the servers would reduce the traffic between the servers significantly. Additionally, investigating advanced load balancing techniques and dynamic resource allocation strategies could enhance the scalability and fault tolerance of the distributed system.

Moreover, a realm of substantial potential in enhancing the VR experience lies in the domain of server-side rendering. The notion of delegating rendering-related computations to servers, catering to multiple users concurrently, resonates powerfully within the framework of the distributed multi-server architecture. A particularly advantageous aspect of our architecture surfaces here – the proximity of servers to clients results in diminished latency due to reduced round-trip times. This effect is amplified when compared to more centralized setups. The ripple effects are manifold: enriched graphics quality, seamlessly fluid frame rates, and an enhanced, uniform experience spanning an array of user devices.

Our research has broader implications for the VR industry, paving the way for the development of more immersive and responsive virtual experiences. As the demand for realistic and seamless VR environments continues to grow, the distributed multi-server architecture, along with server-side rendering and potential optimizations, offers a compelling solution to address the challenges of user concurrency and performance.

VII. Conclusion

In this paper, we presented a detailed analysis of inter-server traffic in distributed VR environments using Mozilla Hubs. Our results demonstrate that a multi-server VR architecture can effectively scale within a LAN, managing increased user loads with linear and sub-linear synchronization overhead, depending on the complexity of user interactions and the frequency of synchronization events. This suggests that distributed architectures can support large-scale VR applications with multiple servers, provided synchronization mechanisms are optimized.

We highlighted the critical role of server-to-server synchronization in maintaining a consistent and immersive VR experience. By simulating real-world user movements, we provided empirical evidence on the network traffic and synchronization overhead between servers.

Despite the promising results, our experiments were limited to a controlled LAN environment with two servers and 12 clients. Future work will involve expanding the experimental setup to include more servers and diverse network conditions, as well as exploring advanced synchronization techniques to further enhance scalability and efficiency. We acknowledge that the study would benefit from a deeper analysis with more clients and servers.

In conclusion, our research provides valuable insights into the design and optimization of distributed VR systems over LANs. By addressing the challenges of inter-server traffic and synchronization, we contribute to the development of more scalable, responsive, and efficient VR architectures, paving the way for future innovations in distributed virtual reality environments.

Acknowledgment

This work was supported in part by U.S. National Science Foundation award 2346681.

References

[1] U. A. Chattha, U. I. Janjua, F. Anwar, T. M. Madni, M. F. Cheema, and S. I. Janjua, "Motion Sickness in Virtual Reality: An Empirical Evaluation," *IEEE Access*, vol. 8, pp. 130 486–130 499, 2020.
[2] C. Westphal, "Challenges in networking to support augmented reality and virtual reality," *IEEE ICNC*, 2017.
[3] Mozilla, "Mozilla hubs," https://hubs.mozilla.com.
[4] I. Tosic, D. M. Hoffman, and N. Balram, "Effect of latency on simulator sickness in smartphone virtual reality," *Journal of the Society for Information Display*, vol. 27, pp. 561–572, 2021.
[5] T. Xu, B. Han, and F. Qian, "Analyzing Viewport Prediction under Different VR Interactions," in *Proceedings of ACM International Conference on Emerging Networking Experiments And Technologies (CoNEXT)*, 2019, p. 165–171.
[6] R. Cheng, N. Wu, M. Varvello, S. Chen, and B. Han, "Are We Ready for Metaverse? A Measurement Study of Social Virtual Reality Platforms," in *Proceedings of ACM Internet Measurement Conference (IMC)*, 2022, p. 504–518.
[7] Meta, "Horizon worlds," https://www.oculus.com/horizon-worlds.
[8] "VR chat," https://hello.vrchat.com.
[9] Microsoft, "Altspacevr," https://altvr.com/.
[10] "Rec room," https://recroom.com.
[11] C. Carlsson and O. Hagsand, "DIVE A multi-user virtual reality system," in *Proceedings of IEEE Virtual Reality Annual International Symposium*, 1993, pp. 394–400.
[12] J. Keller and G. Simon, "Toward a peer-to-peer shared virtual reality," in *Proceedings IEEE ICDCS Workshops*, 2002, pp. 695–700.
[13] C. Greenhalgh and S. Benford, "MASSIVE: a distributed virtual reality system incorporating spatial trading," in *Proceedings of 15th International Conference on Distributed Computing Systems*, 1995, pp. 27–34.
[14] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz, "Npsnet: A Network Software Architecture For Large Scale Virtual Environments," *Presence*, vol. 3, pp. 265–287, 1994.
[15] C. Anthes, P. Heinzlreiter, and J. Volkert, "An Adaptive Network Architecture for Close-Coupled Collaboration in Distributed Virtual Environments," in *Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry*, 2004, p. 382–385.
[16] K. N. Diego Marcos, Don McCurdy, "A-frame," https://aframe.io.
[17] Erlang, "EPMD," https://www.erlang.org/doc/man/epmd.html.
[18] RabbitMQ, "Rabbitmq," https://www.rabbitmq.com/.
[19] G. S. Mullane, "Bucardo," https://bucardo.org.
[20] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov, "MineRL: A Large-Scale Dataset of Minecraft Demonstrations," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 2019, pp. 2442–2448.
[21] Mojang Studios, "Minecraft," https://www.minecraft.net/.