Scaling VR Video Conferencing

Mallesham Dasari[1], Edward Lu[1], Michael W. Farb[1], Nuno Pereira[2], Ivan Liang[1], Anthony Rowe[1]
[1]Carnegie Mellon University
[2]ISEP/IPP and INESC TEC

Figure 1: VR video conferencing where many users can explore worlds and interact as live videos mapped onto a cube.

Abstract

Virtual Reality (VR) telepresence platforms are being challenged to support live performances, sporting events, and conferences with thousands of users across seamless virtual worlds. Current systems have struggled to meet these demands which has led to high-profile performance events with groups of users isolated in parallel sessions. The core difference in scaling VR environments compared to classic 2D video content delivery comes from the dynamic peer-to-peer spatial dependence on communication. Users have many pair-wise interactions that grow and shrink as they explore spaces.

In this paper, we discuss the challenges of VR scaling and present an architecture that supports hundreds of users with spatial audio and video in a single virtual environment. We leverage the property of *spatial locality* with two key optimizations: (1) a Quality of Service (QoS) scheme to prioritize audio and video traffic based on users' locality, and (2) a resource manager that allocates client connections across multiple servers based on user proximity within the virtual world. Through real-world deployments and extensive evaluations under real and simulated environments, we demonstrate the scalability of our platform while showing improved QoS compared with existing approaches.

1 Introduction

Industrial and enterprise automation, fueled by the pandemic, has contributed to rapid growth in interactive and immersive telepresence applications. Unlike conventional 2D video conferencing systems that flattens user attention equally across a grid of videos, Collaborative Virtual Environments (CVEs) allow users to interact with each other and the environment in 3D with spatial audio. SecondLife [4],AltspaceVR [41], Facebook Spaces [39], VRChat [43], and Mozilla Hubs [23] are just a few examples of popular CVEs that are being used for events ranging from live music performances to virtual conferences. In many cases, these platforms not only work on computers, but also integrate with VR headsets.

Unfortunately, scaling these 3D environments in terms of both world-size and number of simultaneous users presents several new challenges compared to 2D video content delivery. First, these environments support many concurrent flows of both audio and video as opposed to a small subset of "pinned" users or active speakers. Some users may only be heard off in the distance, but they still need to be transmitted across the network. Second, as these worlds scale to larger and larger events, the size and total number of engaged users (as opposed to passive listeners) is increasing. There have been reports of live concert performances (SuperBowl LVI) where artists were broadcast into many parallel environments each of which could only support a few dozen attendees [40,42]. In the simplest case, the increased user load simply saturates the capacity of traditional video servers that are designed to send similar content to most users. Third, connections between users are highly dynamic with conversation groups forming, moving, and merging with the social dynamic of users moving through the space. One might naturally assume peer-to-peer architectures would be the ideal solution, but these struggle to scale beyond a handful of users since they are unable to leverage server-side aggregation and transcoding of streams. We ideally need a solution that provides the efficiency of centralized video servers but can also scale horizontally across multiple servers.

In this paper, we present a VR video conferencing telepresence

platform that enables highly scalable VR audio and video streaming from within a web browser. Using a desktop computer or mobile device, users can navigate through a 3D world using the keyboard and mouse for navigation. Audio and video are captured from the view device and presented to other users in close proximity within the virtual environment. When enabled, video appears streamed as a texture map onto a video cube that replaces their normal avatar as shown in Figure 1. The platform is accessible using mixed reality headsets through reality browsers that render the web content in an immersive VR manner. In order to improve scale, we developed a VR Streaming Quality-of-Service (QoS) system that performs Frustum Video Culling and distance-based QoS link estimation based on a user's location within the virtual world (§3.2). Finally, we provide a resource allocator that operates on the communication graph between users to load balance and optimize user to audio/video server to maintain the correct communication linkages while minimizing setup connection latency (§3.3). It is worth noting that we see VR video streams as a stepping-stone to more advanced streaming-based representations, such as real-time capture or codec avatar systems that would benefit from these same techniques [25, 28].

We implemented our VR chat system using an open source Jitsi [3] video conferencing backend connected to the ARENA XR platform [36]. Our system is cross-platform compatible and works with a wide variety of devices including desktops, laptops, VR Headsets and tablets. We experimentally evaluate our system under diverse environments including real-world deployments (dozens of social and conference related events), trace-driven emulation, and large-scale simulation with synthetic traces. Compared to a 2D content delivery baseline over two the traces, our system requires 15× less upload bandwidth, 4× less download bandwidth, and reduces CPU load by 2× on the server-side. Moreover, for the same bandwidth our system has 46% better quality video with 3× rendering frames per second on the client-side. Finally, our system scales to hundreds of video clients without any disruption in clients' connection as they move around in VR space.

In summary, our key contributions are the following.

- We present a scalable VR video conferencing system for Telepresence, that allows a multi-user 3D video conferencing from within a web browser.
- We introduce a series of techniques to optimize and scale our system to hundreds of users. Specifically, we present a distance based QoS, video frustum culling technique, and a resource provisioning mechanism.
- We implemented our system and hosted several real-world sessions such as poster session, student class presentation, group meetings etc. We experimentally demonstrate the benefits of our system by comparing with existing baselines[1].

2 Background and Related Work

Telepresence, as a sense of being present in a shared virtual environment, has been studied extensively in the past [10, 16, 20, 26, 30]. The idea of telepresence has been around for almost four decades and has been realized in a variety of forms ranging from traditional 2D video conferencing, avatar based communication, 3D reconstruction based immersive conferencing, etc.

2D video telepresence systems: The conventional 2D telepresence systems (e.g., Skype, Facetime, Zoom) are shown to be successful for various forms of online communication. These systems provide mostly a single point view for all the participants in the session by capturing the scene with one or more cameras. This form of communication is fundamentally different from our everyday, face-to-face, in-person conversations, where different people have different viewpoints, and each such view point is different from the other. Several advances have been made to resolve these real-world experiences within the 2D video conferencing systems. Notable such solutions include synchronizing eye contact through optimized camera placement [8, 29, 45, 48], telepresence robots with assisted control [6], situated displays and avatars [18, 19, 31, 46]. More recent work in this space include gaze-preserving multiview telepresenc [33], gaze estimation and its improvement for telepresence [32, 34]. Despite several attempts have been made to enable a true co-presence in these applications, it is shown to be extremely difficult to preserve the natural eye contact, situational awareness and gaze direction from multiple viewpoints in 2D telepresence applications.

3D avatars: Recent solutions introduce virtual avatars [23,39,41,43] that can be controlled by users' body tracking and eye movements. Most popular work in this space is Facebook codec avatars, that are generated by neural networks that are trained on images from specialized capture rigs with arrays of cameras [25]. Further line of work in this space include spatial audio, gaze based facial animation (i.e., animated 3D avatars [22]), personalized avatars, to improve the realism of avatar based communication. Previous works also used a spherical object to map the user's video [24], which requires using a 360° camera.

3D scene capture and reconstruction: To provide novel multiple views from each user, more recent solutions introduced 3D scene capturing and reconstruction from depth sensors. With the availability of commodity depth sensors (e.g., Azure Kinect [1], Intel Lidar [2]), there has been significant interest in enabling immersive telepresence via 3D video delivery. This line of work deploys a series of depth cameras and fuses the overlapping depth regions for accurate surface reconstruction [12, 26, 27, 47] in enabling one-one, one-many and many-many group conversations. Another type of immersive 3D content is created from Photogrammetry type of techniques by placing an array of cameras around participants [7]. However, enabling 3D telepresence applications using these technologies face several challenges in terms of cost, network bandwidth, sensor capabilities and remains an open area of research. VR video conferencing is another form of immersive 3D telepresence application, where the shared virtual experiences are created by connecting users in a static 360-degree environment with 2D video streams on top of Web based VR frameworks [13–15, 37].

Video streaming optimizations: There has been extensive prior work on improving the experience for regular video streaming with efficient network resource provisioning. Much of the previous work focuses on improving the adaptive bitrate algorithms by better predicting the available throughput [17, 44]. More recently, streaming 360-degree videos is becoming popular to enable immersive VR applications. To overcome the bandwidth challenges, recent studies use viewport prediction to stream 360-degree videos, where the video is partitioned spatially into *tiles* and only the tiles in the user's predicted viewport are streamed to the client [35, 38].

3 System Design

To capture the spatial properties of users in virtual environments, we composite a video-based avatar by texture mapping live video on one or more surfaces of a 3D geometry. An example video avatar in our system is shown Figure 2. The 3D video avatar element allows seeing other users from multiple angles while still capturing the direction of their gaze. We have experimented with other forms of projection schemes (e.g., 2D projection on a cylinder instead of cube) and find that video cubes are more appealing. The key innovation is that the user is visible from all sides, but the front side of the object is highlighted to show the direction the user is facing. All other sides are darkened making it possible to identify the user

[1]We opensourced the source code for our system and several people in the community have already started using it. We have masked the link to code to preserve double blind submission. We will update the link to source code here upon paper acceptance.

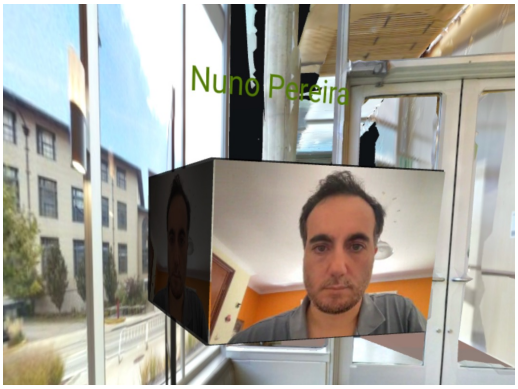


Figure 2: Example of how video can be texture mapped on avatars such that they are visible from multiple directions. Note that the front-facing face is lighter compared to the sides, providing a cue to where the user is facing. In the example, the user is inside a 3D scanned model of a real venue.

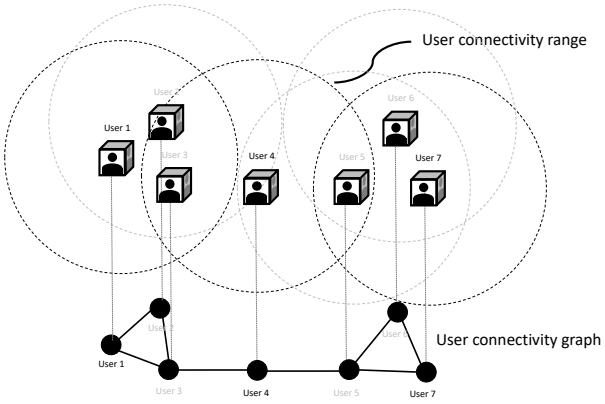


Figure 3: A connectivity graph can be constructed based on the distance between users. The graph is constructed dynamically in real-time as the users move around.

but makes it more difficult to read more subtle social cues (like lip movement etc) from a distance. We believe that a video cube is better to support a single camera facing the user, which does not map well to a sphere as used in previous work [24].

Users move through the 3D environment with mouse movements, keyboard arrow and WASD keys for physical keyboard devices, and with touchscreen swipes, long press, and accelerometer rotations for mobile devices and VR headsets. In this way, users can alter their perspective to pan, rotate, tilt and travel through the environment. By default, all movement height is set slightly above the ground at roughly the same height a user 'sees' while walking along the ground.

3.1 System Model

Figure 3 shows a graph representation of a typical VR session where each user has a radius that defines their connectivity within the 3D environment. Graph edges are undirected and weighted based on distance (i.e. conversation between closer users is more important than distant users). Figure 4 depicts our system design. A set of servers (total S) are managed by a resource allocator that assigns client subgraphs to servers. Clients (total U clients) send control messages to request AV streams of other users with desired QoS given their pose and connectivity in the 3D environment. Audio volume and quality are based on distance as described in Section 3.2. Similarly,

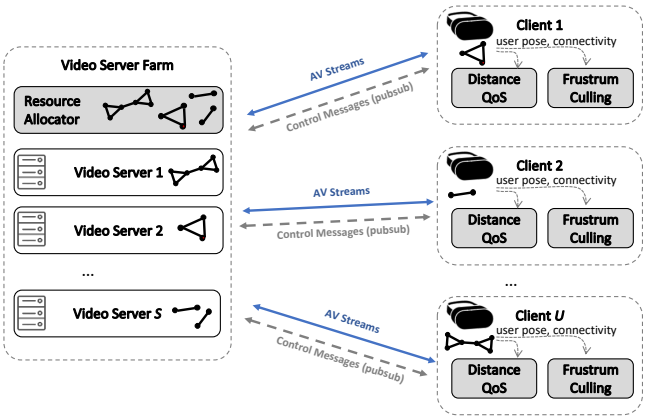


Figure 4: End-to-end system design with three system components: 1) Distance-based QoS, 2) Frustrum culling, 3) Resource allocator.

video quality is based on distance with a maximum range, and also subject to frustum culling at each client. Users who are far away are not streaming their video to each other, thus not connecting in the same session graph. We define a VR session/scene to have U users in total with S servers available to handle audio/video streaming sessions. Each server is capable of handling M client connections. Conversations between any two users are only successful if the edge connecting the two nodes exists on the same server. This implies that for two users to communicate, they need to be connected to at least one shared server. Finally, N denotes how many servers a user can associate with. In practice, N is typically 1, and rarely will be more than 3 as the overhead for clients to manage multiple server connections is often quite high. A user might want to connect to multiple servers in cases when they leave one conversation group and enter another. In these cases, a user can set up two sessions in parallel to avoid a loss in connection during a handover. This also means they can be in multiple conversation groups simultaneously that could be hosted on independent servers.

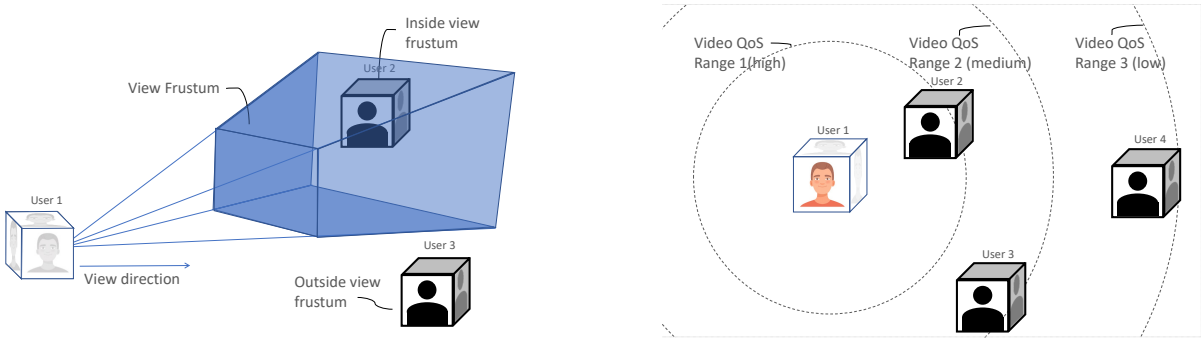
3.2 VR Streaming Quality-of-Service

In this section, we detail VR streaming Quality-of-Service (QoS) features implemented to improve the scalability of the system: (i) frustum video culling and (ii) a spatially-aware QoS mechanism.

3.2.1 Frustum Video Culling

To tackle the challenges of building a scalable VR video conferencing system, we developed filtering techniques that take advantage of the 3D environment, where interactions have similarities to real-world interactions. We do this in two ways: (1) users only need to receive video from nearby users, within their field of view, and (2) downgrade video from distant users while improving the video quality of nearby users in the field of view.

View frustum is a fundamental computer graphics technique to determine the region of space in the 3D environment that appears in the user's field of view, with extensive software and hardware support in modern graphics pipelines. The view frustum has often been used to reduce the complexity of rendering by avoiding out of view computations [9, 11]. We determine which users are in the field of view of each user and dynamically manage their video streams, reducing the audio/video Selective Forwarding Units (SFUs) load. Figure 5a illustrates the idea implemented by our dynamic frustum culling video streaming management, where the user (User 1) only has another user (User 2) in its field of view and thus does not need to receive video streams from other users outside the frustum (User 3). This technique is a major enabler of scalable VR telepresence.



(a) View Frustum-based video control

(b) Distance-based QoS

Figure 5: Video QoS control mechanisms. Each user sends video control messages to (a) request video streams of users in their view and (b) request video quality based on distance and available resolutions.

3.2.2 Distance-based Quality-of-Service

In traditional video conferencing solutions, interactions are "flat," in the sense that all users interact as if they are all very close to each other. In our VR telepresence environment, much like the real world, users can form circles of interaction, where some are closer than others. A user's sound volume, video quality, and dimensions can reflect this. Figure 5b illustrates our distance-based audio/video quality management principle. User 1 is close to User 2, so the video/audio quality between the two is high. User 4, however, is distant from User 1, meaning that the audio/video quality with User 1 can be low, as they will be occupying a small portion of each other's field of view.

3.3 Resource Provisioning

A typical video server has a limited capacity in terms of how many users it can service based on a combination of network and computational resources. In standard video conferencing applications, resource allocation is relatively straightforward since people enter and leave a single conversation medium where each downstream link feed is sized uniformly. In VR, the problem is challenging because groups of users can have conversations that slowly bleed into and/or merge with other groups of users dynamically. With large enough virtual worlds, there is a need to allocate different conversation clusters across multiple servers. A good allocation strategy should try to cluster all users within range on the same set of servers while minimizing the impact of connection disruptions and handover as people move from one area to another. In this section, we describe a technique for allocating groups of users to servers based on their distance-based connectivity graph. We formulate the allocation task as a minimal k-cut balanced graph partitioning problem with the goal of minimizing the total cut edges not covered by a subgraph as described in the next section.

The resource allocator will need to provide solutions under three main conditions: (1) the baseline case where a single server is sufficient, (2) the scenario when multiple servers are required to handle all users, and (3) the overload cases where there are not enough servers to handle the client requests.

3.3.1 Single Server

The simplest case for our resource allocator is when a single server can handle all sessions. The server can host multiple different VR environments, as long as all users can be assigned to a single server. As an optimization, the allocator might distribute subgraphs across servers to balance load and more easily accommodate new users, as discussed in Section 3.3.4.

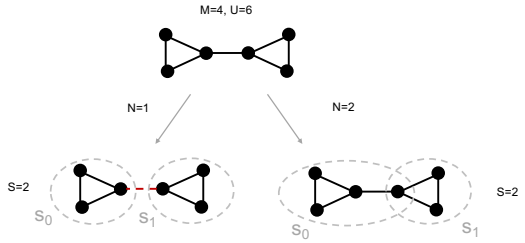


Figure 6: Graph partitioning example for $N = 1$ and $N = 2$.

3.3.2 Multiple Servers

When the number of clients U exceeds the maximum capacity of a single server S, the system needs to load balance clients across multiple servers. Remember that users can only communicate if the nodes and the edge between them is allocated to the same server. This resource management problem can be modeled as a minimal k-cut graph partitioning problem. The cost metric should try to balance the number of nodes on each server while minimizing any cut edges not covered by any subgraph (i.e. users that are near each other but can't communicate). Figure 6 illustrates an example where there are 6 total users that need to be allocated on 2 servers that each support up to 4 users each. We see two possible graph partition solutions depending on N, the total number of connections a single client can make. Considering $N = 1$, the 6-user graph can be partitioned into two disjoint subgraphs, each with 3 users. This will result in a single user from each of the subgraphs that are not able to communicate with the other, but are within range. If instead users are allowed to connect to two servers ($N = 2$), another possible solution is to create a subgraph with 4 users and another with 3 users, where one of the users is in both subgraphs. This reduces users' perceived connectivity breakage at the cost of complexity to manage multiple server connections. Note that, in the general case, where we do not have a predetermined number of subgraphs, this problem is known as NP-hard. Our resource allocator uses several heuristics (including heuristics to predetermine the number of subgraphs) that simplify the problem and approximate the optimal solution.

3.3.3 Overloaded Servers

In the case when there is no feasible mapping of users to servers that covers all edges or there simply isn't enough server capacity for all users ($M * S < U$), some user connections will be dropped.The minimal k-cut graph partitioning heuristic will naturally tend to select strong (higher weight / more closely connected) subgraphs and be biased towards dropping the more distant links nodes with the weakest edges.Alternative approaches to scaling video conferencing sessions

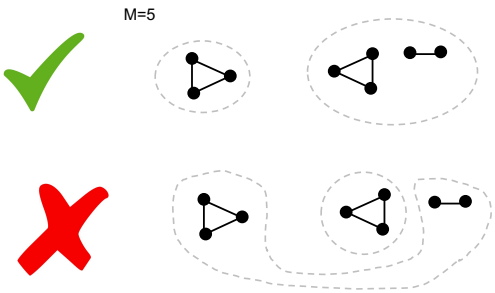


Figure 7: Proximity-based server allocation. Second solution is less ideal since the two clusters on the right are more likely to merge based on proximity.

include decreasing overall QoS through coding and compression or sharing multiplexed streams between servers in the back-end. These approaches are less applicable in VR environments where multiple audio/video channels can not easily be mixed on the server since each user adjusts volumes differently based on their distance from other speakers. In practice, each server can support full duplex (everyone speaking) group sizes of over 50 users. It is reasonable to assume larger clusters would not be fully connected and hence could spread across multiple servers. It is also quite common to find situations where a small number of users are speaking to a large group. Audio for directed half-duplex broadcast can be optimized since the total number of active streams can be reduced. For video, distance-based QoS will naturally reduce bandwidth since there is a limit to the number of close users (nearby enough to be actively streaming high-rate video) that are within anyone's active field of view.

The final and most practical approach to coping with overloaded users is to hand them off to a broadcast channel that could be hosted on an auxiliary server. This broadcast server takes all active sound channels and mixes them into a single output channel so that users can at least hear ongoing conversations even if they can't transmit their own sound and video. A single (or small subset) of active video streams can also be broadcast. It is comparatively easy to scale one-way voice streams out to thousands of users as a fallback support option. This is ideal for the case of a speaker or band performing to tens of thousands of users in the audience. It is possible to swap users in and out based on participation in an event from an active server to the broadcast fall-back server.

3.3.4 Allocator Optimization

The characteristics of our VR telepresence solution, such as the spatial nature of the environment allow for some particular optimizations which introduce additional constraints on the resource allocator.

Spatially Pack Disjoint Sessions: Subgraphs naturally tend to capture the spatial relationship between groups of users. For this reason, it is more likely that a subgraph would need to merge with another nearby subgraph as compared to one that is far away in terms of virtual distance. To reduce the number of connections that need to be migrated during these join/merge operations, nearby subgraphs and users are allocated to the same server (as possible), in anticipation of join/merge operations. Figure 7 portrays two possible graph allocation solutions. The dotted lines depict that the subgraphs are allocated to the same server. The second solution (lower in the figure) is not based on the environment's spatial properties and will lead to more connection migrations in the likely event that the subgraphs to the right merge.

Leveraging Link Quality: When the allocator has the freedom to map users into several subgraphs (users maintain more than one connection, $N > 1$) this choice can be biased based on the network quality of various nodes. For example, it's likely better to request multiple connections from clients that have larger bandwidth network connections. The resource allocator collects and uses quality metrics to prioritize which users could participate in multiple sessions.

Minimize Multiple Client Sessions: It is typically advantageous to reduce the number of clients that are part of multiple subgraphs. Users associated with more than one server introduce complexity in join/teardown and require additional overhead to maintain multiple client sessions.

4 Implementation

Here, we describe the implementation used to build our system, its optimizations, and the framework used to test and measure those optimizations for 3D videoconferencing.

4.1 System

3D System. Our lab has built a 3D web-based user-programmable collaboration system. Our system can be operated from browsers in desktops, laptops, VR/AR Headsets, AR/VR tablets, and command-line. A client visitor to a 3D scene in our system will see other users rendered in 3D on the same scene. User movement and pose are relayed over a MQTT publish-subscribe message bus. Our entire system is available to all as open source software.

Server Setup. We separated our test implementation into 2 servers to more easily isolate and measure load on our Video Server running Jitsi Videobridge in the first case. The second server is a web server hosting client 3D JavaScript code, authentication, MQTT publish-subscribe messaging, and database services. Each server has 20x Intel Core i9-9820X CPU @ 3.30GHz with 64GB RAM available.

Automated Browsers. In our evaluation of these optimizations, we use the Selenium WebDriver v4.2 browser automation software, writing test scripts in Python. Our headless test browser is Google Chrome v100.0, and each browser we automate is using a default window size (w=800 x h=600). The source video streams we use have a variety of sizes 480p, 720p, 1080p, but are capped at 480p upload artificially for consistency of measurement. Video streams are downloaded to each client at a consistent resolution of 480p for the default case, but according to Table 1 for distance QoS.

Replaying Event Traces. When automating a browser client of our system, we send Selenium some URL parameters to set the initial position of each user. Further, we can use trace logs of past events, and replicate in real-time the movement MQTT messages of users to study how the performance of our bandwidth optimization schemes would have behaved for that event.

4.2 Optimizations

Frustum Culling. Here we use the native measurement of our system's client frustum from the JavaScript Three.js library, and for any remote client avatar who's center-point does not appear in the local client's frustum, we disable the video stream for that user (Figure 5a).

Distance Freeze. In this scheme we allow the user to configure the maximum A/V distance to stream (default 20m). The video and audio streams will be disabled for all remote client avatars beyond this distance.

Distance QoS. To create an efficient and appropriate limit on video resolution downloaded from remote clients to the local client we compute the actual video height the user can view in pixels (Figure 5b). The actual remote client video height, R^h, expressed in pixels, may be calculated by the following equation.

$$R^h = W^h * R^{h^m} / 2 * D^m * \tan(fov * 0.5)$$

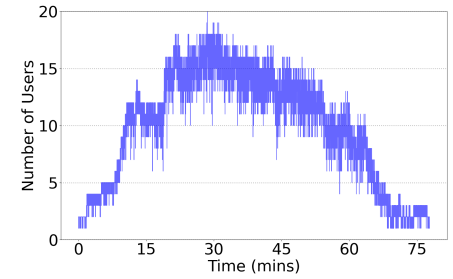
Where, W^h is window height in pixels (600 pixels in our experiment, the headless Chrome default), R^{h^m} is the remote avatar's video

Distance QoS Resolution Constraint Matrix			
Actual Video Height (p)	Video Constraint (p)	Frame Rate Constraint (fps)	Camera Definition
0-45	180	5	—
45-90	180	15	—
90-180	180	30	Thumbnail
180-360	360	30	—
360-480	480	30	SD (standard definition)
480-720	720	30	HD (high definition)
720-1080	1080	30	Full HD/2K
1080-1440	1440	30	—
1440-1800	1800	30	—
1800+	2160	30	Ultra HD/4K

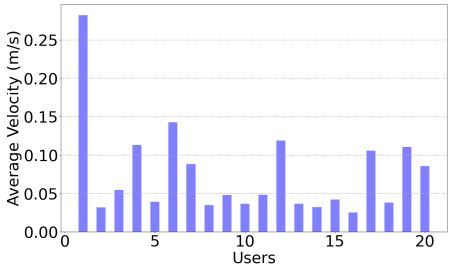
Table 1: Stepped resolution constraints used for video bandwidth allocation in distance-based QoS.

Test Scenarios		Performance		
System	Distance (m)	Upload (Mbps)	Download (Mbps)	CPU Load (%)
Default	N/A	152	13.8	23.27
D	10	26.3	6.8	12.9
	20	40.6	5.7	15.0
	30	57.6	7.1	17.2
F	N/A	29.6	7.39	13.7
D+F	10	17.2	4.6	11.6
	20	18.6	5.2	12.5
	30	21.6	5.6	12.7

Table 2: Performance of the four alternatives (Default, D, F, D+F) for the example poster session at a NSF workshop with a video freezing threshold of different viewing distances.



(a) Number of users in the scene over time



(b) Average velocity of each user

Figure 8: Distribution of users in NSF poster session: a) number of users entered in the scene over the time, b) the average velocity of each user exploring the VR space.

height in 3D meters (0.4m constant), D^m is the distance from the client's POV to each remote client's 3d video in meters, and fov is the Field of View in radians (1.396 rad, 80° Euler constant).

We then create a stepped table of resolution limits (shown in Table 1) for the maximum resolution constraint to allow our client Jitsi Meet library to download for each remote client based on what can be rendered given the distance and window resolution. Each local client checks once per second whether any remote client has changed to a new video constraint tier based on distance changes and updates the allocation request to be higher or lower for that remote client if such a change has occurred.

5 Evaluation

We experimentally evaluate our system both in terms of end-to-end streaming performance and scalability. We compare the performance with a variety of baselines under different scenarios. Our goal is to answer the following key questions: 1) how much network bandwidth (both upload and download) can we save with our optimizations? 2) what are the performance benefits in terms of client-side quality of experience? 3) what is the limit in terms of users when scaling to multiple servers?

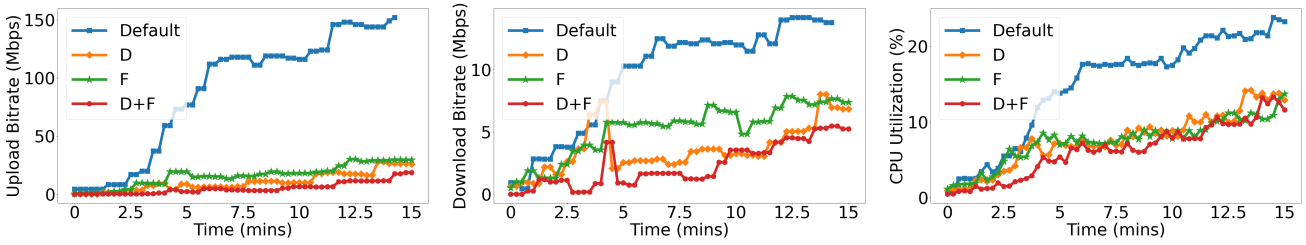
5.1 Evaluation Methodology

Over the past 10 months, we have hosted a variety of sessions on our VR conferencing platform, including poster sessions, team meetings, student class presentations, etc. To experimentally evaluate and quantify our system, we conduct two types of experiments: 1) a trace-driven emulation, where we replay the user movement that is collected in one of our sessions under different types of optimization strategies, 2) a large-scale experiment with synthetically generated trace with 100s of users to evaluate the scalability with multiple servers. Next, we describe the methodology to evaluate our system.

Traces: We use two traces to evaluate our system. One is a multi-university poster session held at a NSF workshop. The NSF trace has about 20 users in total exploring the VR space for about one hour. The second trace is generated synthetically to represent a social mixture style session. We generate the synthetic trace using a popular technique called Brownian motion [21], where the users are free to walk around with random motion. We process the two traces to have the users' 6-DoF pose (translation and rotation). We use the pose data from the traces to emulate user movement and adapt the video quality by replaying the trace for each of the optimization strategies. The user distribution for the NSF poster session is shown in Figure 8. A total of 20 users join and leave slowly within a duration of 75 minutes. The average velocity and maximum distance traveled by the users throughout the session is 5cm/s and 150m respectively. Note that the minimum average velocity is 3cm/s and maximum average velocity is 23cm/s for a given user, and the overall maximum velocity among all users is observed at 46cm/s. The users are also given the freedom of moving around freely, and so we observed a wide variety of motion patterns during the session. This covers a range of motion patterns such as the speed and trajectory for different users.

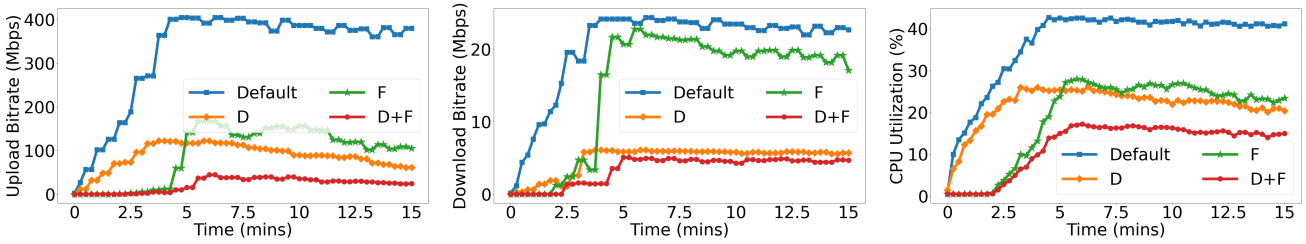
Experimental setup: We host our VR telepresence (Jitsi [3]) server on a Linux machine as described in §4. For emulating the video clients in the case of trace-driven experiments, we use Selenium [5] to launch web clients programmatically. We launch the clients on multiple AWS compute instances each with 192 vCPUs. We input a fake video and audio to a Selenium driver to simulate a real-time web camera feed and stream video to and from the server. In the interest of time, the experiments are run for only 15 minutes of duration for each trace. We have also experienced our platform on a variety of devices ranging from Desktop, Laptops, Tablets, and VR headsets (e.g., Oculus Quest 2)[2]. During the experiments, we do not limit the network conditions to avoid any influence of poor network performance. We evaluate all the alternatives with a variety of resolutions, and present results of 480p resolution for brevity.

[2]Note that for Oculus headset, we receive video feeds from other clients, but do not stream video from the headset because of the lack of front-facing camera



(a) Upload Video Bitrate (b) Download Video Bitrate (c) Server CPU Utilization

Figure 9: Performance comparison of four alternatives (Default, D, F, D+F) for a poster session at a NSF workshop. The timeline also indicates the number of users entered in the scene over the time (at 15^{th} minute the session has 20 users in the scene).



(a) Upload Video Bitrate (b) Download Video Bitrate (c) Server CPU Utilization

Figure 10: Performance comparison of four alternatives (Default, D, F, D+F) for a synthetically generated trace with Brownian random motion as an example of social mixture event. The timeline shows that the users are quickly added to the scene within 5 minutes of the session with a total of 30 users from 5^{th} minute onwards.

More details are given in our Implementation Section 4.

Metrics: We measure the performance using the following metrics: 1) upload bandwidth which is the total outgoing bitrate for the video bridge, 2) download bandwidth which is the total incoming bitrate for the video bridge, 3) server CPU utilization (busy state of all cores together), 3) client-side connection quality as defined in Jitsi [3], 4) rendering frames per second (FPS) on the client. Additionally, we report the number of supported clients when we scale to multiple servers.

Optimization strategies: We compare our system with the following alternatives:

- **Default:** This is a default system scenario that has no optimizations. In this case, the server streams all client videos to everyone with a default constraint of 480p.
- **Distance-based QoS (D):** This system has only distance based QoS. We experiment with different distances: {10, 20, 30}m for freezing the video streams and report the results for each case.
- **Frustum Culling (F):** This system has only frustum culling enabled.
- **D+F:** This is our system with both frustum culling and distance based QoS both enabled.

5.2 Performance Results

Upload and Download bandwidth: Figure 9a-b shows the upload and download bandwidth at the video server bridge under an NSF poster session trace. The plot shows the bandwidth needed as the users are added to the VR scene (within a timeline of 15 minutes). As shown our system with **D** and **F** both enabled, consumes significantly less bandwidth compared to the default system. At the 15^{th} minute, there are 20 users in the scene, for which the upload and download for the default system are 150 Mbps and 14 Mbps respectively. Our system with **D** and **F** enabled together reduces this bandwidth by 22× and 3× for upload and download respectively.

Similar performance trends can be observed for the social-mixture style synthetic trace as shown in Figure 10. In this scenario, the users entered the scene quickly within two minutes of the session and bandwidth stays constant for the rest of the session at 30. As before, our system consumes much lower upload and download bandwidth compared to the default system. The key performance difference with the synthetic trace from an example NSF poster session is that the distance based QoS is not playing a big role because the users are much closer than in the NSF trace. Unless we specify the distance threshold to very low (e.g., less than 2m), we did not see much change in the performance difference. Under both traces, the frustum culling has the most benefits, and combined with distance based QoS, our system has significant benefits.

CPU load: The server-side CPU load is shown in Figures 9c and 10c for the NSF and Synthetic traces respectively. The CPU load is computed as an average busy status of all the cores on the server (in our system 20 cores). The CPU load goes up to 25% and 42% under NSF and synthetic traces for the default system. In our system, the CPU load is well under 10% for both traces when **D** and **F** are both enabled.

Impact of distance threshold: The distance threshold used in our distance based QoS optimization plays a critical role in resource consumption on the server-side. For example, a low distance threshold leads to lower bandwidth requirement and has less CPU load, but prohibits the nearby clients from communicating with each other. On the other hand, a larger threshold leads to high resource utilization, but all the clients stream videos to all the other clients. Table 2 shows the impact of the distance threshold (with {10, 20, 30}m) on upload and download bandwidth as well as the CPU for 20 clients at a NSF workshop example poster session. As shown, the bandwidth and CPU utilization increases as we increase the distance threshold value. The threshold can be adjusted to different values for different applications to achieve optimal server-side performance and client-side experience (e.g., a social mixture kind of application should have high threshold value, whereas a poster session kind of application can have it under 10m).

In summary, the above savings in both load the network and compute resources allows our system to scale to significantly more users on a single server compared to the default system with no culling and distance based adaptation.

Client-side performance: For a good quality of experience for clients, it is important to preserve high video quality as well as

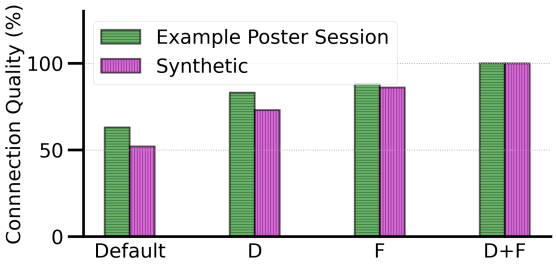


Figure 11: Client-side connection quality under 60 video clients with multiple simultaneous replays of NSF example poster session and synthetic trace.

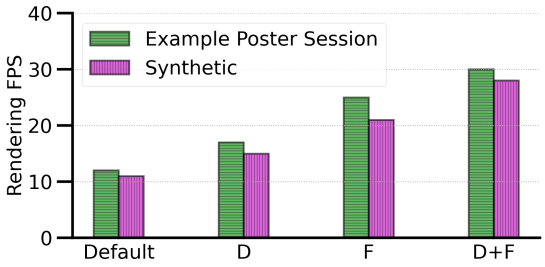


Figure 12: Client-side rendering FPS for 60 clients with multiple simultaneous replays of NSF example poster session and synthetic trace.

the temporal smoothness (i.e., frames displayed per second (FPS)). We measure two types of metrics for this purpose: 1) connection quality, which is solely influenced by the network and the server-side load, 2) rendered FPS, which is mainly influenced by the amount of rendering load on the client. For example, poor clients such as tablets or headsets have a very low compute capacity and cannot tolerate many clients in the browser to display at line speed (i.e., 30FPS). As a stress test, we conduct this experiment by deliberately replaying the traces multiple times simultaneously from different AWS instances to create a load of 60 clients.

Figure 11 shows the connection quality under the two traces for all four scenarios. The system by default drops down to 50% for both traces whereas our system is not at all affected. The primary reason for connection quality drop in case of the default system is the heavy load on the server (both network and compute). In our experiments, we observe a 70% CPU load when there are 60 clients on the server. Using **D** and **F**, our system significantly cuts down the network and compute load on the server and improves the video resolution as well as the streaming FPS, and hence the overall connection quality. Similarly, Figure 12 shows the rendered FPS when there are 60 clients in the scene. The rendered FPS goes down below 10 FPS for the default system for both traces because it is extremely compute intensive to render 60 video cubes in the browser. Our system avoids this by not rendering out-of-sight as well as farther away video clients.

5.3 Scaling to Multiple Servers

We evaluate scalability of our system in terms of the number of supported clients with respect to multiple servers. As mentioned in section 3.3, the server is limited to support only a few tens of clients and once it reaches its maximum load in terms of CPU and network, it has to either drop the client connections or all more connections but reduce the quality of other connections. In this section, we

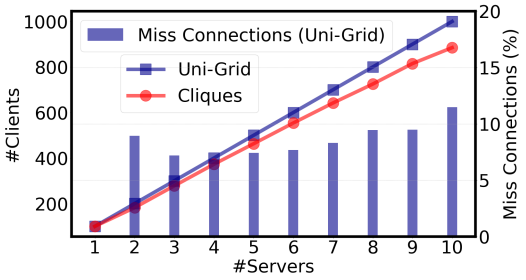


Figure 13: Scalability with the increase in number of servers.

evaluate the maximum number of supported clients provided they all get the best connection quality.

In our experiments (with 480p video resolution), our server on the default system (i.e., with no **D+F** optimizations) supports up to 60 clients without affecting the quality of clients' connection. However, our system can support up to 100 clients on a single server because of the reduced resource consumption with **D+F**. In theory, we could extrapolate this number linearly with more servers and estimate the scaling performance of our system. However when the number of clients exceeds the capacity of a single server, the single session should be scaled to multiple servers with some overlapping users streaming to multiple servers to serve all users in their locality. The design choices here either accommodate parallel connections on multiple servers for overlapping clients with a certain bandwidth overhead or miss the connections among overlapping clients for bandwidth efficiency. In the following, we compare the two approaches. We evaluate the scaling performance of resource provisioning in our system by comparing with a baseline solution: Uniform grid based geographic boundaries as used by many cloud gaming solutions today, where the VR space/scene is divided into grids statically and each grid is served by a separate server. While this design is relatively simple compared to our resource scaling mechanism, it hinders the clients that overlap in the neighboring grids from communication and the clients often miss connections.

Figure 13 shows the scalability of our system compared to the above uniform fixed grid approach under a synthetic trace when scaling from 100 to 1000 clients. The uniform grid approach has better scaling ability with the increase in the number of servers, however, shows a significant increase in miss connections as we increase the number of grids and clients. On the other hand, single session/server can only support 60 clients. Our system bridges the gap between the two by introducing the redundancy of connections from multiple servers for overlapping clients in the cliques. Because of the redundancy, our system can scale to 886 clients while the uniform grid approach can go up to 1000 clients, however, the key advantage with our approach is that there are no missed connections and all the clients are served. The uniform grid approach has up to 11% miss connections for overlapping clients.

6 Conclusion and Future Work

We have presented a VR video conferencing telepresence system that scales to hundreds of users in a single virtual environment with spatial audio and video. We introduced two optimizations to reduce the resource consumption on the server— distance based QoS technique and frustum video culling. In addition, we presented resource provisioning mechanism to scale our system to many clients while providing high quality of experience i.e., connection quality as well as the rendering frames per second. Through the experimental evaluation and several real-world deployments, we demonstrated that our system reduces upload and download bandwidth, and CPU load significantly on the server-side. We also open sourced our platform code for the community to further explore this line of work.

Our current VR conferencing is realized by compositing a video-based avatar with texture mapping to video cubes. While this allows us to view the users from multiple viewpoints, it still lacks the true immersive 6-DoF content where the users can see through occlusions. We are currently working on extending our system to capture scene using depth sensors and reconstruct 3D scene via point clouds and meshes. In the future, we envision our system to support fully immersive 3D video conferencing using other forms of volumetric capture such as depth sensors.

ACKNOWLEDGEMENT

This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] Azure Kinect DK. https://azure.microsoft.com/en-us/services/kinect-dk/. Online. Accessed: May 2022.

[2] Intel RealSense LiDAR Camera L515. https://www.intelrealsense.com/lidar-camera-l515/. Online. Accessed: May 2022.

[3] Jitsi Videobridge. https://github.com/jitsi/jitsi-videobridge. Online. Accessed: May 2022.

[4] Second Life. https://secondlife.com/. Online. Accessed: May 2021.

[5] WebDriver. https://www.selenium.dev/documentation/webdriver/. Online. Accessed: May 2022.

[6] S. Alers, D. Bloembergen, M. Bügler, D. Hennes, and K. Tuyls. Mitro: an augmented mobile telepresence robot with assisted control. In *AAMAS*, pp. 1475–1476, 2012.

[7] M. Broxton, J. Flynn, R. Overbeck, D. Erickson, P. Hedman, M. Duvall, J. Dourgarian, J. Busch, M. Whalen, and P. Debevec. Immersive light field video with a layered mesh representation. *ACM Transactions on Graphics (TOG)*, 39(4):86–1, 2020.

[8] M. Chen. Leveraging the asymmetric sensitivity of eye contact for videoconference. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 49–56, 2002.

[9] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, Oct. 1976. doi: 10.1145/360349.360354

[10] J. V. Draper, D. B. Kaber, and J. M. Usher. Telepresence. *Human factors*, 40(3):354–375, 1998.

[11] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics*, pp. 124–133, 1980.

[12] M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, et al. blue-c: a spatially immersive display and 3d video portal for telepresence. *ACM Transactions on Graphics (TOG)*, 22(3):819–827, 2003.

[13] S. Gunkel, M. Prins, H. Stokking, and O. Niamut. *WebVR meets WebRTC: Towards 360-degree social VR experiences*. IEEE, 2017.

[14] S. N. Gunkel, M. Prins, H. Stokking, and O. Niamut. Social vr platform: Building 360-degree shared vr spaces. In *Adjunct Publication of the 2017 ACM International Conference on Interactive Experiences for TV and Online Video*, pp. 83–84, 2017.

[15] S. N. Gunkel, H. M. Stokking, M. J. Prins, N. van der Stap, F. B. t. Haar, and O. A. Niamut. Virtual reality conferencing: Multi-user immersive vr experiences on the web. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pp. 498–501, 2018.

[16] R. Held. Telepresence. *The Journal of the Acoustical Society of America*, 92(4):2458–2458, 1992.

[17] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *SIGCOMM*, 44(4):187–198, 2015.

[18] A. Jones, M. Lang, G. Fyffe, X. Yu, J. Busch, I. McDowall, M. Bolas, and P. Debevec. Achieving eye contact in a one-to-many 3d video teleconferencing system. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

[19] N. P. Jouppi. First steps towards mutually-immersive mobile telepresence. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 354–363, 2002.

[20] R. Kachach, S. Morcuende, D. Gonzalez-Morin, P. Perez-Garcia, E. Gonzalez-Sosa, F. Pereira, and A. Villegas. The owl: Immersive telepresence communication for hybrid conferences. In *SMAR-Adjunct*, pp. 451–452. IEEE, 2021.

[21] I. Karatzas and S. E. Shreve. Brownian motion. In *Brownian motion and stochastic calculus*, pp. 47–127. Springer, 1998.

[22] A. Kreskowski, S. Beck, and B. Froehlich. Output-sensitive avatar representations for immersive telepresence. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2020. doi: 10.1109/TVCG.2020.3037360

[23] D. A. Le, B. Maclntyre, and J. Outlaw. Enhancing the experience of virtual conferences in social virtual environments. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 485–494, 2020. doi: 10.1109/VRW50115.2020.00101

[24] Z. Li, T. Teo, L. Chan, G. Lee, M. Adcock, M. Billinghurst, and H. Koike. Omniglobevr: A collaborative 360-degree communication system for vr. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, p. 615–625. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3357236.3395429

[25] S. Ma, T. Simon, J. Saragih, D. Wang, Y. Li, F. De La Torre, and Y. Sheikh. Pixel codec avatars. In *CVPR*, pp. 64–73, 2021.

[26] A. Maimone and H. Fuchs. Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 137–146. IEEE, 2011.

[27] W. Matusik and H. Pfister. 3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *ACM Transactions on Graphics (TOG)*, 23(3):814–824, 2004.

[28] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, M. Elgharib, P. Fua, H.-P. Seidel, H. Rhodin, G. Pons-Moll, and C. Theobalt. Xnect: Real-time multi-person 3d motion capture with a single rgb camera. *Acm Transactions On Graphics (TOG)*, 39(4):82–1, 2020.

[29] D. Nguyen and J. Canny. Multiview: spatially faithful group video conferencing. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 799–808, 2005.

[30] T. Ogawa, K. Kiyokawa, and H. Takemura. A hybrid image-based and model-based telepresence system using two-pass video projection onto a 3d scene model. In *ISMAR*, pp. 202–203. IEEE, 2005.

[31] O. Oyekoya, W. Steptoe, and A. Steed. Sphereavatar: A situated display to represent a remote collaborator. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 2551–2560, 2012.

[32] Y. Pan and K. Mitchell. Improving vip viewer gaze estimation and engagement using adaptive dynamic anamorphosis. *International Journal of Human-Computer Studies*, 147:102563, 2021.

[33] Y. Pan and A. Steed. A gaze-preserving situated multiview telepresence system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2173–2176, 2014.

[34] Y. Pan and A. Steed. Effects of 3d perspective on head gaze estimation with a multiview autostereoscopic display. *International Journal of Human-Computer Studies*, 86:138–148, 2016.

[35] S. Park, A. Bhattacharya, Z. Yang, M. Dasari, S. R. Das, and D. Samaras. Advancing user quality of experience in 360-degree video streaming. In *2019 IFIP Networking Conference*, pp. 1–9, May 2019. doi: 10.23919/IFIPNetworking.2019.8816847

[36] N. Pereira, A. Rowe, M. W. Farb, I. Liang, E. Lu, and E. Riebling. Arena: The augmented reality edge networking architecture. In *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 479–488. IEEE, 2021.

[37] M. J. Prins, S. N. Gunkel, H. M. Stokking, and O. A. Niamut. Togethervr: A framework for photorealistic shared media experiences in 360-degree vr. *SMPTE Motion Imaging Journal*, 127(7):39–44, 2018.

[38] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *MobiCom*, pp. 99–114. ACM, 2018.

[39] Facebook Inc. Facebook Spaces. https://www.facebook.com/spaces. Online. Accessed: May 2021.

[40] Hamish Hector. Meta's Foo Fighters Super Bowl VR concert failed

in the most basic ways. https://www.techradar.com/news/metas-foo-fighters-super-bowl-vr-concert-failed-in-the-most-basic-ways. Online. Accessed: May 2021.

[41] Microsoft Inc. AltspaceVR. https://altvr.com/. Online. Accessed: May 2021.

[42] Scott Hayden. Foo Fighters to Play Concert in VR for Free After the Super Bowl. https://www.roadtovr.com/foo-fighters-quest-2-concert-super-bowl/. Online. Accessed: May 2021.

[43] VRChat Inc. VRChat. https://hello.vrchat.com/. Online. Accessed: May 2021.

[44] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *INFOCOM*, pp. 1–9. IEEE, 2016.

[45] W. Steptoe, R. Wolff, A. Murgia, E. Guimaraes, J. Rae, P. Sharkey, D. Roberts, and A. Steed. Eye-tracking for avatar eye-gaze and interactional analysis in immersive collaborative virtual environments. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, pp. 197–200, 2008.

[46] T. Tanikawa, Y. Suzuki, K. Hirota, and M. Hirose. Real world video avatar: real-time and real-size transmission and presentation of human figure. In *Proceedings of the 2005 international conference on Augmented tele-existence*, pp. 112–118, 2005.

[47] H. Towles, W.-C. Chen, R. Yang, S.-U. Kum, H. F. N. Kelshikar, J. Mulligan, K. Daniilidis, H. Fuchs, C. C. Hill, N. K. J. Mulligan, et al. 3d tele-collaboration over internet2. In *In: International Workshop on Immersive Telepresence, Juan Les Pins*. Citeseer, 2002.

[48] R. Vertegaal, I. Weevers, C. Sohn, and C. Cheung. Gaze-2: conveying eye contact in group video conferencing using eye-controlled camera direction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 521–528, 2003.