Quality of Service Impact on Edge Physics Simulations for VR

Sebastian Friston[1], Elias Griffith[2], David Swapp[1], Caleb Irondi[2], Fred Jjunju[2], Ryan Ward[2], Alan Marshall[2] and Anthony Steed[1]

Abstract—Mobile HMDs must sacrifice compute performance to achieve ergonomic and power requirements for extended use. Consequently, applications must either reduce rendering and simulation complexity - along with the richness of the experience - or offload complexity to a server. Within the context of edge-computing, a popular way to do this is through render streaming. Render streaming has been demonstrated for desktops and consoles. It has also been explored for HMDs. However, the latency requirements of head tracking make this application much more challenging. While mobile GPUs are not yet as capable as their desktop counterparts, we note that they are becoming more powerful and efficient. With the hard requirements of VR, it is worth continuing to investigate what schemes could optimally balance load, latency and quality. We propose an alternative we call edge-physics: streaming at the scene-graph level from a simulation running on edge-resources, analogous to cluster rendering. Scene streaming is not only straightforward, but compute and bandwidth efficient. The most demanding loops run locally. Jobs that hit the power-wall of mobile CPUs are off-loaded, while improving GPUs are leveraged, maximising compute utilisation. In this paper we create a prototypical implementation and evaluate its potential in terms of fidelity, bandwidth and performance. We show that an effective system which maintains high consistencies on typical edge-links can be easily built, but that some traditional concepts are not applicable, and a better understanding of the perception of motion is required to evaluate such a system comprehensively.

Index Terms—virtual reality, streaming, edge-computing

1 Introduction

Mobile Head Mounted Displays (HMDs) endeavour to support high quality VR, but ultimately must trade-off power for ergonomics, accessibility and battery life. Consequently, developers face more constraints in delivering rich experiences on these platforms than on traditional ones such as desktops. A potential solution is streaming: treating the HMD as a thin client where significant computation is off-loaded to the cloud. Modern examples include game-streaming services such as Google Stadia, but similar solutions have existed for several years in different vertical markets (e.g. SGI's Visual Area Networking, or Microsoft's Azure Remote Rendering).

Typically, the cloud service renders images that are streamed to a client as video. For HMDs, such systems must fit bandwidth constraints while guaranteeing viewpoint response-times of less than 16 ms. This is challenging even without network transport delay, as video encoding and decoding are expensive operations. It has been questioned whether even next-generation technologies can reconcile these constraints [23].

It is worth considering then, what other schemes may support an optimal balance of power, latency and Quality of Experience (QoE). Scene-graph streaming has been an approach to Distributed Virtual Environments (DVEs) since their inception [54], but clients usually require complex prediction and synchronisation to overcome poor network Quality of Service (QoS) [3]. This is because they are designed to operate over wide area networks, where delays can reach 10s-100s of milliseconds. These functions are complex, and such clients would not be considered 'thin'.

In this paper we re-evaluate scene-graph streaming within the context of edge-computing. In edge-computing, a high-quality connection brings resources close to a device, allowing a powerful remote processor to respond more quickly than a weak local one. This high QoS creates the potential for alternative ways to build DVEs. Whereas traditional DVEs and games require complex local clients to overcome poor QoS, an edge-physics client takes on a role analogous to a cluster node, turning the HMD back into a thin client.

- [1]*Department of Computer Science, University College London.*
- [2]*Department of Electrical Engineering and Electronics, University of Liverpool.*

Digital Object Identifier:10.1109/TVCG.2021.3067757

Unlike configurations such as render streaming, scene-graph streaming allows keeping loops with the tightest latency requirements, such as head tracking, local to the device. Intensive jobs that stress the power budget of mobile CPUs are off-loaded. Mobile GPUs, however, are increasingly power efficient and can be leveraged to maximise total compute resources for the best experience.

In this paper, we implement a framework that explores what we term *edge physics*. Complex simulations typical of high-end games are offloaded to an edge-computing resource, and the results streamed as scene graph changes. The types of computation considered are designed to be more complex than a mobile HMD could reasonably support. We use our framework to explore practical tradeoffs in this configuration, and evaluate it in terms of fidelity, bandwidth and performance.

In many ways edge physics is analogous to cluster rendering, but with important differences. A typical edge physics configuration will use heterogenous platforms, have clients autonomously handle user input, and will connect over a shared and more highly variable network (running over, e.g., WiFi6 or 5G) than would be available to a cluster. We find resulting QoS fluctuations undermine traditional latency compensation through prediction, but that overall performance suggests this configuration is very promising.

2 Related Works

2.1 Streaming Virtual Reality

Many recent works consider streaming pre-recorded or dynamically rendered VR through 360 video. Pohl et al [42] described an end-to-end system using current-generation technology, and compared compression techniques and commerical offerings. One of the latest is Shi et al's [47], which streamed subsets of panoramas as FFMPEG encoded video from edge-nodes over LTE or WiFi.

To reduce bandwidth, modern systems are usually view-dependent. The viewer's gaze is used to send a subset of the scene, or to optimise sampling of the scene, such that bandwidth is dedicated to what is most salient. For example, Zhou et al [55] used non-traditional projections to vary information density. Ozcinar et al [39] used visual attention maps to control tile delivery. This does however create a dependency between server response time, visual quality and speed. If the server does not respond quickly enough to viewpoint changes, the user will see visual artefacts. Authors have tried to compensate for this in different ways. For example, Rossi et al [45] used gaze path prediction to load an optimal set of tiles, while Shi et al [47] used overscanning. The trade-off becomes between bandwidth and uncertainty. Video

underlies a number of commerical streaming applications, such as NVidia Shield [35], Valve's SteamLink [52] and Google Stadia [8].

However, video is far from the only representation to be considered for streaming VR. Lamboray et al [24] were one of the first to suggest streaming point-video. Recently, Park et al [40] applied 2D techniques such as view-dependent transmission and tiling to voxel streaming. Guéziec et al [9] presented a framework for streaming with VRML. Olbrich & Pralle [37] streamed virtual realty 'movies' at the triangle level. Hladky et al [16] and Mueller et al [33] streamed shaded primitives. Lin et al [28] repurposed JPEG compression for 3D geometry, including support for lossy compression. Hnidek [17] presented a 'semi-reliable' protocol for real-time 3D data, which selectively re-transmitted lost packets based on information content, without delaying others.

Another approach is command streaming, which streams instructions at a level comparable to the graphics API. WireGL [19] streamed OpenGL commands to virtualise a distributed graphics architecture. PolyStream uses command streaming to support cloud based 3D content. The motivation is not to reduce bandwidth, but the cost of supporting the GPUs necessary to deliver video, which is a primary challenge for the cloud gaming industry [43].

At the scene graph level, streaming overlaps with traditional DVEs. In fact, cluster rendering was one of the original use-cases for DVEs, where they provided transparent APIs for scaling systems beyond a single workstation [53] [15] [29]. This led to distribtued scene-graph libraries such as blue-c [34] and Wolverine [4], and concepts such as the scene-graph-as-a-bus [54], that underlie DVEs as collaborative systems.

2.2 Prediction

At the scene-graph level, prediction is used for both bandwidth reduction (through dead reckoning) and latency compensation. Singhal & Cheriton [49] were one of the first to use position history with adaptive convergence in remote rendering. Lau & Lee [25] modelled the error budget of a predictor and classified four error sources: model mismatch, noise, quantization and time precision, which authors have addressed in different ways.

Kim & Kim [22] demonstrated improved accuracy using a Kalman filter. LaViola [26] proposed Double Exponential Smoothing (DESP) which used linear regression. Stakem & AlRegib [50] proposed Exponential Smoothing (ES), which combined numerical differencing (backwards) with Eulers method (forwards). Both DESP and ES show similar performance to a Kalman filter, but require parameter tuning. Hanawa & Yonekura [12] [13] [14] used a Taylor Series expansion running on the server in order to achieve smaller sampling intervals and so higher precision. Aggarwal et al [1] proposed synchronising clocks to improve accuracy, while Tumanov et al [51] proposed proactive lag compensation based on the estimated latency to a client. These are just a few examples of the many works addressing predictor error.

Almost all these examples use some variant of Euler's method or regression per-dimension. This can work, but makes certain assumptions. Further, the return of increasing complexity is limited. For example, Singhal & Cheriton [49] and Lau & Lee [25] demonstrate sensitivity to unmodelled terms, however Hanawa & Yonekura [12] showed better results with lower polynomial-order models. Meng et al [31] noted the context sensitivity of this and proposed a hybrid system that would switch order dynamically.

2.3 Smart Clients

For internet-based applications, the performance of independent, per-object predictors alone is typically too low because real applications have many highly non-linear inputs, such as collision responses and stochastic user input. Internet-based DVEs typically use contextual and domain knowledge to compensate for large latencies. For example, Cronin et al [5] proposed trailing state synchronisation, which ran multiple parallel simulations for immediate context switching. This provided similar features to *rollback*, often used in online fighting games [44]. Li et al [27] and Shi et al [48] proposed using potential fields to predict player motion. Ohlenburg [36] extended dead-reckoning to include collision responses. Bernier [3] describes the latency compensation in Half Life. Clients ran local simulations while mirroring user input to a remote simulation that integrated input from all players. Local simulations provided instantaneous feedback between receving authoritative updates from the server. This is typical of modern DVEs and games which must remain responsive over the high latencies of the public internet. Following the dead reckoning principle: the more duplication between client and server, the longer they can go without communicating, but the more power the client must expend. This reveals a trade-off: power vs. distance.

2.4 Edge Computing

Edge computing is a concept in distributed computing which brings resources close to the client. As Hu et al [18] demonstrate, edge computing can benefit CPU intensive applications as powerful remote processors can respond faster than weak local ones if complemented by a sufficiently fast connection.

One complementary technology generating interest is 5G. Lai et al [23] assert that high quality mobile VR is delay limited. The QoS sufficient for desktop games is insufficient for VR, which requires much lower latencies. (e.g. 16 ms for head tracking). The capabilities of 5G may meet these requirements, and authors are already examining its potential for streamed VR [10] [38] [6]. The state-of-the-art is Furion [23], which splits a virtual scene between local and remote renderers. Furion's authors are more pessimistic than most about streaming video alone however, and show through careful analyses that the QoS requirements for VR will saturate even next-generation networks.

Therefore, when we consider the potential of next-generation networks for streamed VR, it is still pertinent to ask: what should be streamed?

3 Edge Physics

Edge physics supports high quality VR on low-power devices by streaming at the scene-graph level. Both client and server have a logical representation of the scene state. The client also has a visual representation. The server runs the simulation & logic and sends state updates. The client is responsible for updating the camera and rendering. A diagram of the process is shown in Figure 1. Edge physics has a number of advantages: the protocol is simpler than for command streaming. There is no compression or decompression delay as with video. Bandwidth is superior to video streaming for most use-cases. The visual quality is always pixel-perfect, and the same stream can broadcast to heterogenous clients. Edge physics exchanges the same variables as common DVEs and games, but relies on the edge-configuration's network QoS rather than a local simulation to maintain QoE (as in, e.g. [3]).

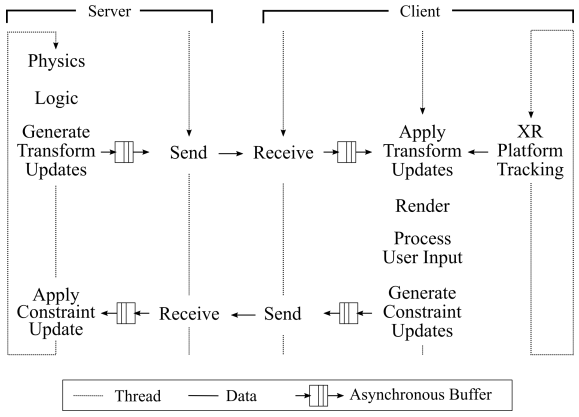

Fig. 1: Logical flow diagram for prototypical implementation

Scene-streaming may provide a near optimal balance of computing power. Mobile CPUs have a low per-core 'power wall' due to thermal constraints. This will most disadvantage single-threaded tasks such as physics simulation. Despite this there are still improvements needed to support next-gen applications [11]. Mobile CPUs are also more likely

to be loaded with compute intensive tasks such as inside-out tracking. Mobile GPUs however are becoming more powerful, driven in part by machine vision and deep neural network applications (e.g. DeepSense [20]). Beyond efficiency, we can also exploit the sum of the computing power available across two machines. We demonstrate a scene complex enough that a desktop PC cannot maintain VR frame-rates on it own. However, coupling it to an Oculus Quest for visualisation allows it to be viewed comfortably.

More importantly for VR, edge physics offers a good distribution of latency. Sensitivity to latency depends on modality. Head tracking latency thresholds are on the order of 16 ms or less [21]. Higher level perceptions are more task dependent, and can be more tolerant. For example Morice et al [32] found in a ball bouncing task that perception-action coupling was changed between 30 ms to 90 ms, but only at 50 ms did users change control modes, and only at 70 ms did they perceive a change. In edge physics, head tracking responses are computed locally, and maintain visual quality regardless of speed. Only logical cues are subject to link delays, and though delayed, the state is always consistent.

Edge physics could be a low-overhead way to build co-located collaborative VR. Edge physics is not an alternative to a smart client however. The compute distribution relies on the visualiser being a dumb terminal, which is enabled only by a good local connection as part of the edge-computing configuration.

Figure 2 compares the distribution of responsibilities between different architectures. The distances between the Simulation, Render and Display stages have different sensitivities. The cloud icon size is indicative of these delays. The long-range links in a traditional DVE are likely to have large latencies over the internet, while edge-nodes will be lower, on the order of building-scale latencies.

For a typical DVE which operates over long distances, some simulation code (the 'smart client') must be duplicated. In Figure 2 we indicate that similar simulation code often runs on both client and server, so for N clients, there are N+1 copies of the code. The application also requires a conflict resolution protocol. Render Streaming and Edge Physics are concerned with the local configuration. Neither precludes additional, traditional long distance synchronisation.

For example, a nearby render streaming node may responsible for simulation and rendering for a client, while also being networked to a central server to support multi-player gaming. An edge physics node may also have a more complex upstream synchronisation scheme to a centrally managed simulution.

In this paper however we focus on the downstream connection, between edge server and one or more clients. This is analogus to cluster rendering. However, our implementation is built to be complementary to the existing scene-graph API and to support heterogenous clients, rather than building an application around a distributed API. Further, cluster rendering is usually performed on a dedicated wired network. The shared, typically wireless, networks of variable architecture used for edge-computing have qualitatively different properties.

To explore the potential of edge physics, we construct a prototype implementation and evaluate it in terms of consistency and performance, as well as investigate the effect of design decisions such as prediction.

4 Implementation

4.1 Overview and Architecture

We implemented a proof-of-concept in Unity 2019.2.6. Unity processes acted as both the server and client(s). A deskop PC was used as a server. Two additional PCs and an Oculus Quest (an Android mobile platform) acted as clients (Figure 3). The server only has to run the physics loop in real-time, so can be less powerful than a typical VR desktop.

The scene was duplicated in each process, with physics simulations disabled on the clients and rendering disabled on the server. Objects were assigned GUIDs at design time. Static geometry was shared ahead of time in the application binary. The server had control of all dynamic objects. For example, the 1600 balls in Subway (Section 5.1). The server would compute their dynamics, collision responses, etc, and transmit their resulting state at points in time to the clients.

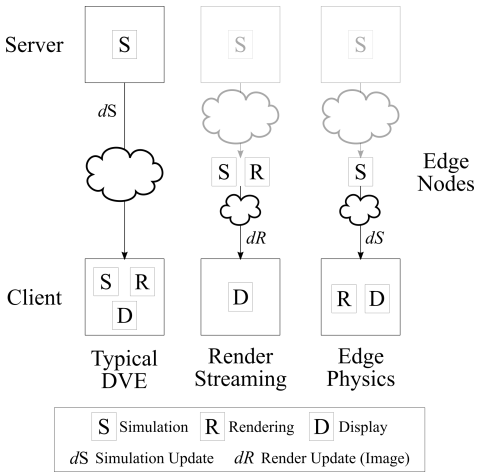

Fig. 2: Comparison of process distribution between a typical DVE, render steaming and edge physics.

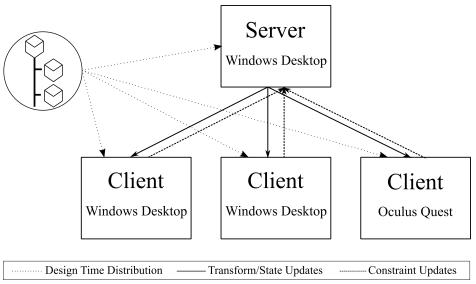

Fig. 3: Message flow diagram of edge-physics network

This is sufficient for visualisation, but effective VR requires the world to be responsive. Clients had control of the viewpoint and any avatar geometry (such as hand models). Grasping was supported through spring constraints between hand controllers and objects [46]. Three constraints were created per controller to facilitate torque. Spring-constraints are good mechanisms, because forces are integrated locally making them less sensitive to QoS than force-reflection. They support L1-L3 collaboration (multiple users affecting the same object at the same time [30]), and most algorithms such as haptics are based on Hooke's Law in any case.

4.2 Messaging

The whole system defined only two types of message: a spring constraint update and object state update. These were 48 and 52 bytes respectively (Table 1). Messages were packed into UDP datagrams, and generated and consumed using blitting. UDP datagrams were limited to 508 bytes in order to minimise fragmentation and reduce packet loss. Our networking code required a further 8 byte overhead, meaning 9 state updates could be sent per datagram (though they were packed dynamically).

Unity uses a component-based programming model. Seperate objects were created for handling state and constraint messages. Consequently, message types were not mixed within datagrams. Our networking code routed datagrams to specific objects using the scene graph [54]. The interpreted type depended on the destination.

Messages were generated in the physics loop at a fixed interval and complete datagrams transmitted on a separate thread via a lock-free queue. The timestamp member was used as instrumentation for our evaluation, and to discard out-of-order packets which would otherwise introduce spatial jitter. The timestamp was the server's time to 100 ns. High-precision time APIs are now available on modern desktop OSs, allowing the system clock to be used. Low-power clients may not have high-precision time APIs, but they only need to compare timestamps.

State Update		Constraint Update	
Type	Property	Type	Property
Int32	Id	Int32	Id
Int64	Timestamp	Int64	Timestamp
Vector3	Position	Vector3	LocalAnchorPosition
Vector3	Velocity	Vector3	RemoteAnchorPosition
Quaternion	Rotation	Int32	IsConnected
		Float	SpringCoefficient
		Float	SpringDamper

Table 1: Message definitions

Whether a powerful remote processor can outperform a weaker local one depends on the speed of the connection to it. Our implementation therefore optimised for latency above all else. No blocking calls were used in the main-thread, logic & control-flow such as type interpretation was embedded in the routing, and strong typing was used to facilitate processing through direct memory copies.

4.3 Prediction

In pre-trials we captured server data from our two evaluation environments (Section 5.1) and evaluated various predictors. We found position-history based Euler extrapolation had the highest accuracy without interaction, and hold-last-sample with, over delays up to 100 ms. Consistent with previous works, higher-order predictors ended up introducing more error than they compensated. Accordingly, we implemented three prediction modes: Hold-Last-Sample (H) and single-order Euler where states were extrapolated from time of message receipt (L) or transmission (G). The behaviour of the three modes is shown in Figure 4.

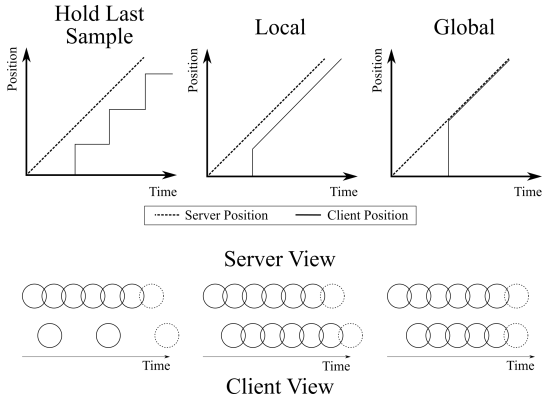

Fig. 4: Diagram of expected predictor behaviour. In Hold Last Sample, the client shows snapshots of the server state, delayed by the transport latency. In Local mode, the state includes a velocity term, so motion is extrapolated between updates, but the state is still delayed by the transport latency. In Global mode, the clocks are synchronised and so the client can compensate for transport delay when extrapolating the state of the server.

4.4 Example

As an example of a typical configuration, we show a subway scene (Figure 6) with 1600 objects falling through the environment. The physics simulation alone takes 12 ms per frame. A VR-capable desktop PC can show it at 50 fps, bound by the CPU. Using edge physics, we explore this environment on an Oculus Quest connected via WiFi. The desktop runs the physics and transmits updates at 50 Hz. The Quest maintains its native 72 Hz frame rate, using local prediction to upsample the physics. The bandwidth was 47 Mbps and the latency 50 ms on average between a message being generated and parsed. This is the one-way delay for the physics updates only; recall the head and hand tracking loops run locally, and at 72 Hz are less than 14 ms.

Profile	Latency (ms)	Jitter (ms)	Packet Loss (%)
1	0	0	0
2	10	1	0.1
3	50	5	1
4	100	10	2
5	250	30	5

Table 2: Quality of Service Profiles

5 Evaluation

We evaluate the potential of edge physics by testing our system with different configurations under different QoS. Our primary measure was consistency. Simulations of rich environments will include many collisions and other effects that result in highly dynamic object trajectories. Inconsistencies will be revealed in the differences between these trajectories. The server has the true state, so consistency was measured as Eucliden distance between object positions at fixed wall-clock times. There is nothing special about how positions are handled, so measures should generalise to other continous parameters. Consistency is important because the fidelity of the local view determines how effectively a user can perceive and manipulate the world.

Additionally, we measure system metrics such as latency and framerate, in order to understand the source of inconsistency and to validate our apparatus works as expected.

Currently there are no absolute thresholds to define success. We would consider an edge-physics system successful if it avoids introducing any artefacts not present in an equivalent native system. It is reasonble to expect additional inconsistencies may be tolerable however, depending on the nature of any motion artefacts introduced. In this initial work though, we aim only to understand what these artefacts may be, and where they come from.

5.1 Environments

We use two prototypical environments. The first (Room, Figure 5) is a room-scale scene with interactive objects. Interaction between two users and the environment was recorded and played back for each trial. Expert users were recorded ensuring L3 interaction was captured, such as handover and collaborative lifting of stacked objects. Note that while one or more users may introduce constraints, these are always evaluated at the server, so the consistency measure is always the same: between the server and one client. The second (Subway, Section 4.4, Figure 6) consisted of a large number of objects falling through a complex world. This environment had no interaction, so all motion should be purely physics-based. Therefore only one client was connected for the trials in Subway.

5.2 Apparatus

Three computers were connected via our building's 1G Ethernet network. A parallel network synchronised their clocks to within a microsecond using PTP. One was nominated as the server, which also hosted a software network emulator. Edge physics traffic was routed through the emulator, then over the building's network to the clients. The emulator inserted delays and dropped packets to reduce the QoS according to one of five profiles (Table 2). The emulator operated at the UDP datagram level. Delay per datagram was given by $Latency + Jitter \cdot \mathcal{N}(0,1)$. Packet Loss was the uniform probablility of a datagram being discarded.

Our intent was to use this apparatus to measure the impact of QoS on our primary measure, consistency. We first profiled our system to ensure it was operating as expected (Section 6.1). Then the QoS was degraded to examine in what ways consistency was undermined (Section 6.2). Based on the results from these tests, we introduced additional conditions to evaluate the effects of Update Rate (Section 6.3). Finally we measure the computational overhead of our implementation, and make some observations about bandwidth consumption (Sections 7 & 8).

Fig. 5: Room environment

Fig. 6: Subway environment

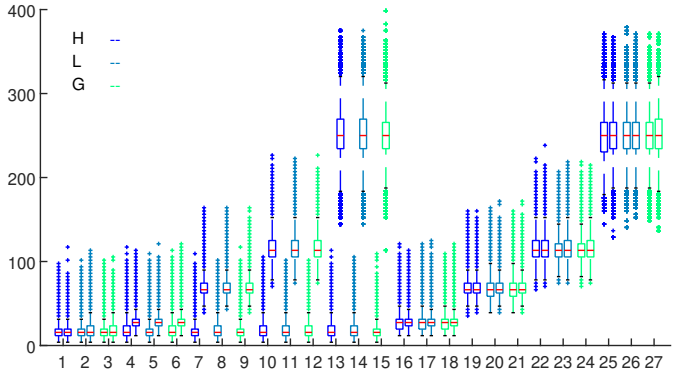

Fig. 7: Message latencies (in ms) for Room QoS trials for both clients, for each experimental condition. For clarity, predictor conditions (H,L,G) are also distinguished by colour. Larger copies of all plots are available in the supplementary materials.

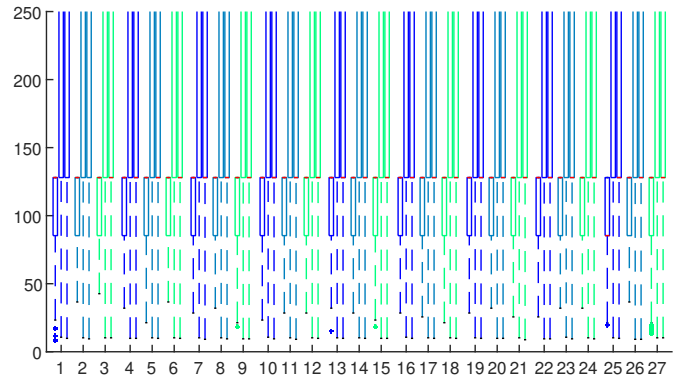

Fig. 8: Box-plot of FPS samples taken during the Room QoS trials for each condition. Samples were taken each frame. Each (per condition) triple corresponds to the server (left) and clients (two rightmost). Additional colour coding of predictor modes as in Figure 7.

6 Network Quality

Our first objective was to determine how consistency degrades with QoS, and whether prediction can ammeliorate it. Unlike in tightly controlled networks, it is expected that different edge clients may have different QoS connections to the same server, and could therefore have quite different experiences. Futher unlike, e.g. clusters, users interact through their client, using it to generate constraints, and clients are deliberately not synchronised in order to support heterogenity. We suspect therefore even though there is one server with the ground truth, the different experiences of two clients could still affect the global simulation. To test this we include both symmetric and asymmetric network profiles. In total, we defined 27 conditions: 9 QoS pairs with 3 predictor modes each (Table 3). We run Room under conditions 1-27 with both clients, and Subway under conditions 1-15 with client 2 only. We recorded approximately 90 and 15 seconds of data for Room and Subway respectively, per trial. All metrics - consistency, latency, framerate - were captured for each trial.

6.1 Latency Breakdown

We use the Tracepoints method [7] to measure the latency between generating and processing an edge physics message. This method tracks messages through the system, taking high precision timestamps as they go using the PTP-synchronised system clocks.

During each trial, the latency of each recieved message was recorded. Figure 7 plots the latencies for the Room QoS conditions. The boxes show the IQ range and the crosses the outliers. Each pair of plots per-condition corresponds to the two clients. For clarity, conditions are also distinguished by colour based on their prediction mode. Larger copies of all plots are available in the supplementary materials.

We see that the latencies for each client closely follow the condition's QoS profiles. For the asymmetrical conditions, 4-15, Client 2 (right) shows an increase in latency with QoS profile, while Client 1's (left) latency remains stable. The change in prediction mode has no effect, as expected. In the symmetrical conditions, 1-3 & 16-27, measurements are consistent between clients. This is expected as the client load is low, so there is less sensitivity to client compute power. The measurements do include the time to receive updates on the main thread however, so it would not be unexpected if they deviated on mismatched clients.

The baseline latency - measured for Clients with QoS profile 1 - was an average of 15 ms. The Tracepoints method allows measuring the latency of intervening stages. To evaluate the sources of latency we measure three additional QoS profiles, with latency, but no jitter or packet loss (Table 4). From this we can see the majority of the delay is waiting at the client for the next frame. For performance reasons messages are processed in the physics loop, which runs at 15 ms intervals. It may be expected based on this, that frame-quantisation could hide network latency. As the 5 ms profile (middle column, Table 4) shows however, this is not the case. Quantisation is not deterministic so any network delay affects the probability of being quantised to the subsequent frame.

We also record the FPS of each node (Figure 8). The first of each triple is the server, and the other two the clients. We see the frame-rate is consistent across all conditions, as expected. We see that the clients maintain VR frame-rates (first quartile above 120 fps). We also see that client frame-rates are higher than the server's, suggesting that clients could not run the application alone, as the more powerful server itself cannot reach the same frame-rate. The outliers show a number of transient drops, likely due to unexpected system load spikes and operations such as garbage collection. As clients are asynchronous however they maintain their frame-rate and tracking latency irrespective of server interruptions.

These latency and frame-rate measures confirm our apparatus is operating as expected.

Condition	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Client 1 Profile	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5
Client 2 Profile	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	2	2	2	3	3	3	4	4	4	5	5	5
Predictor	H	L	G	H	L	G	H	L	G	H	L	G	H	L	G	H	L	G	H	L	G	H	L	G	H	L	G

Table 3: Network Quality Experimental Conditions, by Client Profile (see Table 1) and Predictor ([H]old-Last-Sample, [L]ocal and [G]lobal)

Stage	0 ms			5 ms			10 ms		
	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
Create Message	0.24	0.12	8.62	0.25	0.13	8.70	0.25	0.13	8.82
Transmit Message	0.14	0.00	8.62	0.14	0.00	9.16	0.14	0.00	10.72
Receive Message	0.72	0.10	42.20	5.76	0.00	28.97	10.92	1.20	39.26
Process Message	16.84	4.44	108.45	15.65	4.43	105.90	15.94	4.40	90.50

Table 4: System delays (in ms) for 3 emulated latencies. (Jitter and packet loss are always zero.)

6.2 Consistency

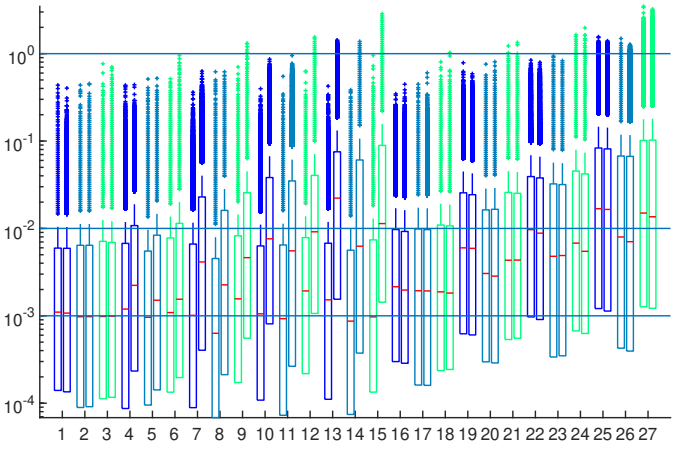

Fig. 9: Position consistency (in m) for the Room QoS conditions. Additional colour coding of predictor modes as in Figure 7. Guides at the mm, cm and m scale.

Figure 9 shows the consistency measures for Room under each condition. We see that for conditions with profiles 1 & 2, consistency is almost always mm to cm scale, but with outliers of 1 m to 2 m. Conditions with profiles 3 & 4 maintain a mean consistency below one cm, though the 25th percentile is between one cm and one m. Only conditions with profile 5 have a mean error above one cm. Even with pre-recorded avatars there is an effect of asymmetrical QoS. Dynamics that could be changed by this include more extreme violation of spring constraints, for example. Consistency tends to decrease regularly with decreasing QoS, however in absolute terms the differences are on the order of centimeters at most. Conditions are distinguished mostly by their outliers, so we examine these in detail.

6.2.1 Motion Thresholds

Any parallel system will have some inconsistency between physics and rendering due to sampling quantisation. The faster an object moves the larger this inconsistency will be. Figure 11 shows the error as a function of velocity for one client. As expected, the predictors skew the relationship, replacing delay error with prediction error to one degree or another. Contrary to expecations, the largest absolute errors occur at the lowest speeds. As speeds are the true speeds at the server at measurement time, we suspect this is due to prediction error following collision responses.

6.2.2 Lifetime of an Object

All conditions have outliers up to 1 m or so. To understand what these mean to the user we isolate single objects and investigate their

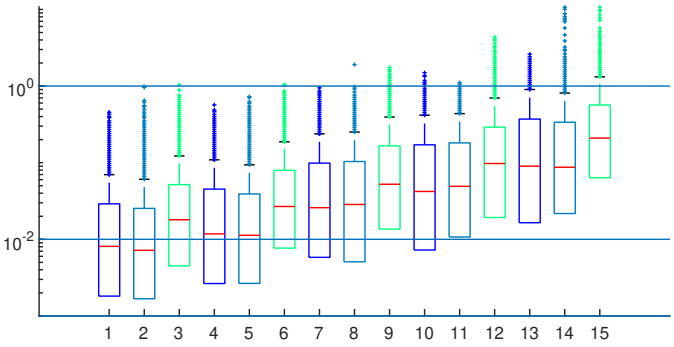

Fig. 10: Position consistency (in m) for the Subway QoS conditions. Additional colour coding of predictor modes as in Figure 7. Guides at the mm, cm and m scale. Only conditions 1-15, for Client 2, are shown - Subway had no interaction and so no asymmetrical conditions.

trajectories. Figure 12 shows such a trajectory. We see there are two types of error. In the first an object follows the server trajectory, but delayed. In the second an object leaves the trajectory. To quantify these we split them into M and Q errors - representating tangential and phase error, respectively. M is defined as the Euclidean distance to the closest point on the true trajectory. Q is the distance along the true trajectory from the closest point to the true point. Both errors are measured in metres. This is to make them comparable as for slow objects, small distance errors can manifest as large time-offsets.

Figure 14 shows the MQ error distribution for the Room trials. As expected, we see that misprediction error is the source of M error, as errors with large components in the vertical axes correspond almost exlusively to prediction modes Local and Global. We also see that Global time has worse performance than Local (a broader distribution in both axes). On first glance this is unexpected, as the clocks are tightly synchronised. If predictors have high error however, integrating over larger time periods that include transport delay, as done in Global mode, would integrate these errors as well.

While Q errors are not by definition harmless, they will be less noticable so long as objects do not stutter, as the simulation will appear consistent over time. Figure 13 shows the M errors for Client 2. Almost all are due to prediction error. Though it is possible for no-prediction conditions to have M errors as well, if the local trajectory skips a curved section due to infrequent updates.

We repeated all measurements for the Subway environment (see Figure 10 for overall consistency). We see the same susceptibilities to prediction error as Room, and higher sensitivity to network conditions. On first glance this is surprising: with no interaction, prediction should be more beneficial in this environment. However, while motion is regular, it is highly non-linear due to bounces and collisions. Further, with more objects we must send more packets, representing proportionally more moving objects, increasing sensitivity to jitter and packet loss.

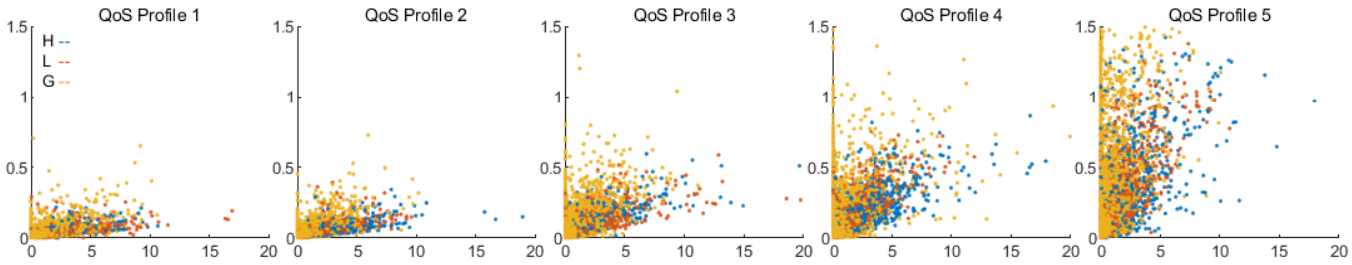

Fig. 11: Error (y-axis) (in m) as a function of velocity (x-axis) (in m/s) for all errors at Client 2 in the Room QoS trials, shown by QoS profile (left to right) and predictor mode.

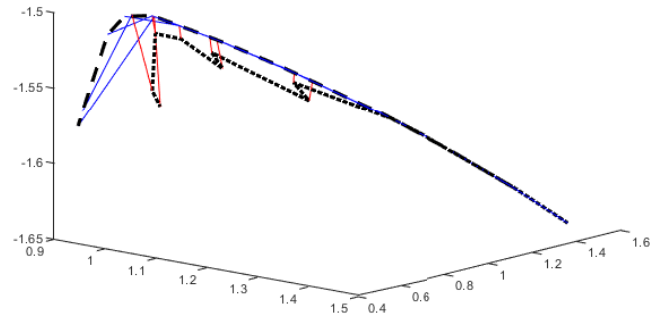

Fig. 12: Server (dashed) and client (dotted) trajectories for an object using local prediction (L), illustrated with errors tangential to (red) and along (blue) the path.

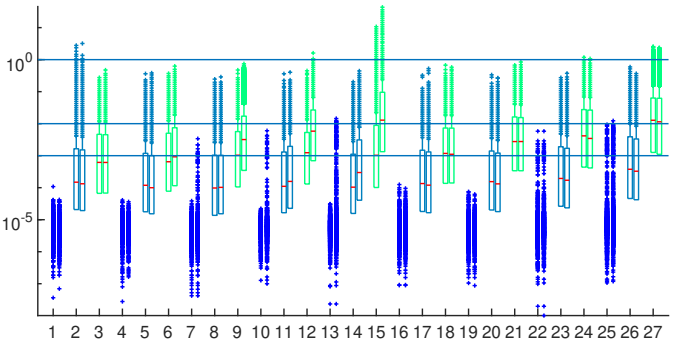

Fig. 13: M errors (in m) for the Room QoS conditions. Additional colour coding of predictor modes as in Figure 7. Guides at the mm, cm and m scale.

Finally, we fit a linear model to see if error is predictable. Some independent variables were significant, but with R^2 values of 15 to 25 % the model was not deemed to be useful. Further details are available in the appendix.

6.3 Update Rate

We observe that many outliers are due to prediction error, and that without prediction, the major source of M-error is skipped curves due to infrequent updates. Based on this, we suspect that reductions in update rate may have the greatest potential to introduce artefacts. Accordingly we ran further trials to isolate the effects of update rate.

We define 12 new conditions, with baseline network QoS but varying update rates (Table 5). Figure 15 shows the measured periods for the 12 conditions. The period is defined as the interval between two successive message receipts for an object. As can be seen, the effective update rate is never below the physics rate of the server, as this quantises transmissions. As expected, prediction has no effect as the overheads are trivial.

Figure 16 shows the consistency for each condition. This shows an interesting pattern, in that the position errors for condition 7 are lower than for condition 4, even though condition 7 has a lower update rate (33 Hz vs 50 Hz). This suggests it is more important the update rate be a multiple of the physics rate (fixed at 15 ms during the trials), than high. We also see that while update rate does affect consistency, it does so trivially, with a 4x reduction in update rate resulting in only 1-2 mm additional error on average (condition 10 vs. 1), and no obvious change in outliers.

6.4 Jitter

A weakeness in our evaluation is that we do not measure spatial jitter. Jitter can be more salient than latency [41], but we are not aware of an objective measure. Though, we can make an approximation using acceleration. Figure 17 compares the histograms of accelerations for all moving objects at the Server and Client 2 in the Room QoS trials. If a distribution skews right compared to the server, it means the system is rendering higher accelerations than are present in the true motion. If it skews left it indidicates the rendered accelerations are smaller, as if there was low-pass filtering of velocity. As can be seen, the client distributions skew right compared to the server, indicating that the system is distorting the dynamics in a way that could appear as jitter. Additionally, the Hold-Last-Sample mode skews right compared to the other predictor modes. This is expected as objects will jump more between updates (see Figure 4). Predictor modes also affect the dynamics of the server. This is also as expected, as Room is interactive.

We can quantify this distortion by measuring the skewness of the client distribution compared to the server. This is a very coarse approximation, which cannot be used to quantify jitter, or how QoE is affected, but for interests sake we show the measures for Room in Figure 18. The measures are generally consistent with the overall coherency (Figure 9). Predictor mode H shows lowest distortion; though the absolute skew is highest for H in Figure 17, the relative skew between the client and server is smaller than for L or G.

7 Bandwidth

There are limited message types so bandwidth can be easily estimated. Messages are packed into UDP datagrams. Each UDP datagram is limited to 508 bytes to minimise fragmentation. There is a further overhead of 8 bytes for our networking code. Figure 19 shows the bandwidth requirements as a function of object count. Modern video streaming solutions have bandwidths of between 3 and 144 mbps depending on latency and quality [23] [47]. These guides are shown on Figure 19.

Condition	1	2	3	4	5	6	7	8	9	10	11	12
Client 1 & 2 Profile	1	1	1	1	1	1	1	1	1	1	1	1
Update Period	10	10	10	20	20	20	30	30	30	40	40	40
Predictor	H	L	G	H	L	G	H	L	G	H	L	G

Table 5: Update Rate Experimental Conditions, by Client Profile, Update Period (ms) and Predictor (Hold-Last-Sample, Local and Global)

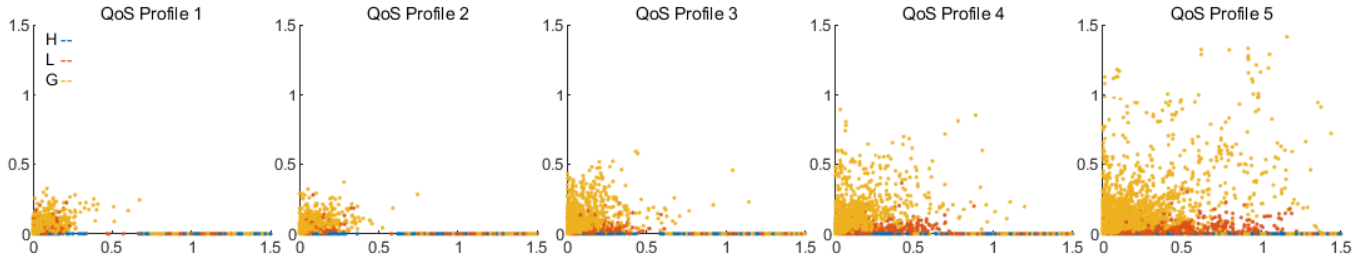

Fig. 14: M (y-axis) and Q (x-axis) components of each error in the Room QoS trials, by QoS profile and predictor mode.

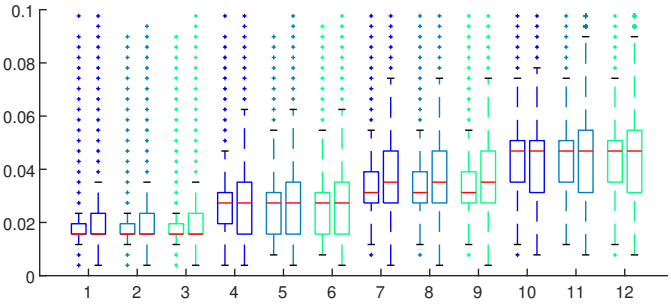

Fig. 15: Measured update periods (in s) of Room for each Update Rate condition in Table 5. Additional colour coding of predictor modes as in Figure 7.

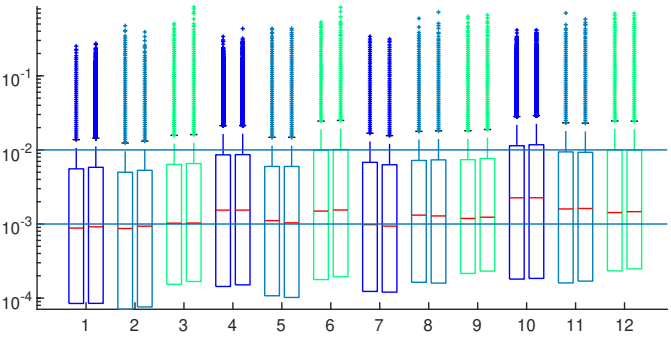

Fig. 16: Position errors (in m) of Room for each Update Rate condition in Table 5. Additional colour coding of predictor modes as in Figure 7.

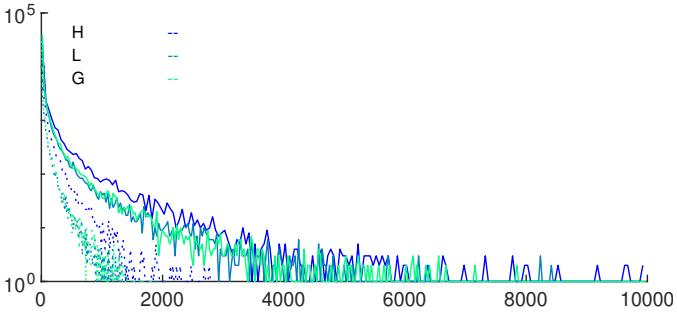

Fig. 17: Histogram bin counts of accelerations (in m/s^2) measured at the Server (dotted) and Client 2 (solid) in the Room trials for conditions 4,5,6 (Table 3), by predictor mode.

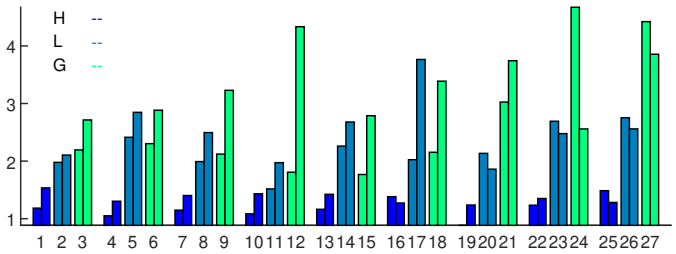

Fig. 18: Ratio (unitless) of acceleration histogram skew between the Server and Client 2, for all conditions in the Room QoS trials.

8 Performance

Table 6 shows the execution time overhead for both scenarios. Only time on the main thread is considered. Overhead is trivial, consisting mainly of memory accesses. Message transmission is the most expensive operation, because messages must be partitioned into sets of UDP datagrams. These could be preallocated if desired. As the total number of objects decreases, the relative overhead of handling the datagrams increases, but in absolute terms it remains very low.

9 Discussion

9.1 Prediction

The effects of prediction mode on consistency reinforces the need to test on real networks, as we find the opposite results from our pre-trials. Generally, our system is hindered more than helped by prediction. Previous works have shown the benefits of higher-order predictors can be outweighed by their noise [12], but we show this occurs even for low-order predictors when subject to highly non-linear conditions in real systems: namely, stochastic update-rates and user input.

The timescales over which simple predictors work are those which the user is unlikely to notice anything, so long as a high message rate is maintained. Recall that the interaction considered here is hand/object interaction - head tracking is supported by the fast local loop. Edge-computing configurations should support reliably high update rates, but real systems will need benchmarking to confirm how consistent an environment they can provide.

Low-order predictors do not have the accuracy necessary for interactive worlds. This was shown by Global mode having poorer performance than Local. This suggests that the predictors are error prone, and integrating over longer periods introduces more error. Local prediction was less suspectible to this, but rarely offered any improvement. It may then be better not to attempt to optimise consistency, but instead use local prediction to negate salient motions, such as jerks, introduced by transient interruptions.

	Execution Times (ms)			
	Subway		Room	
Method	Per Frame	Per Object	Per Frame	Per Object
Receive	1.1133	0.0007	0.0947	0.0018
Transmission	4.3412	0.0027	0.1551	0.0029
Processing	0.3971	0.0002	0.0265	0.0005
Update	1.4771	0.0009	0.0741	0.0014
Predictive Update	1.7435	0.0010	0.1011	0.0019

Table 6: Execution times of the edge physics implementation by method

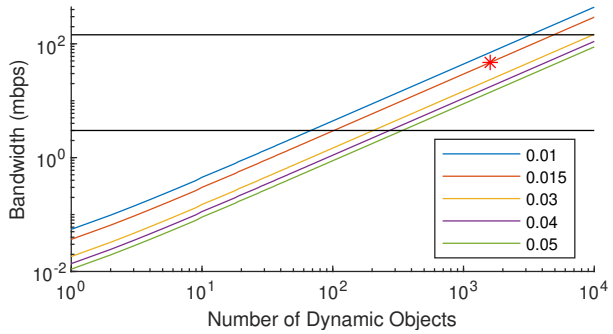

Fig. 19: Bandwidth (mpbs) as a function of object count, with guidelines at the upper (144) and lower (3) bounds for render streaming with H264 for VR, and our most complex scenario indicated (*).

9.2 Performance

Our implementation has very low overhead, and the most expensive operation (transmission) takes place on the server. Compared to video, the bandwidth of our most complex scene is one order of magnitude lower than an H264 stream of equivalent quality, and we do not require CPU or GPU resources to do encoding or decoding. Our implementation transmits all dynamic objects each frame. This is necessary with prediction on as UDP is unreliable, and any lost updates may cause prediction error to accumulate. With prediction off (or a fall-off in place) the server would only need to transmit moving objects, reducing bandwidth further. Our system is optimised for latency and the predominant source of latency is frame-quantisation, even with artificial network latencies of 10 ms. Irrespective of compute power, this potential for low latency interaction with a global simulation could make edge-physics attractive for co-located collaborative VR compared to traditional DVE technologies.

9.3 Consistency and Limitations

The biggest gap in our evaluation is the effect on spatial jitter. Even if error is low, it can be salient if it varies in a way that introduces jerk. We are not aware of an objective measure of spatial jitter perception however. Our acceleration approximation suggests there is dynamic distortion, but not what the impact is. Work on jitter in the 3DUI literature (e.g. [2]) could provide grounds to build such evaluations in the future.

Our evaluation showed a clear difference between tangential and phase errors as a consequence of prediction mode. In any multi-threaded system there will be some phase error in object motion due to time quantisation differences and cross-thread communication delays. However, it is not so simple as phase error is acceptable, and non-phase isn't. Our evaluation showed that the largest errors occured at relatively low speeds. If these were interpenetrations as suspected, they may be highly salient. Our QoS profiles included some far below what would be employed in an edge-computing configuration. While the 'good' profiles overwhelmingly had sub-mm precision, every profile had outliers due to transient latencies. It is not clear how salient large phase errors may be, and any phase error will become salient if it changes sufficiently quickly. Our MQ breakdown is a start, but to fully evaluate any scene-synchronisation scheme will require developing more comprehensive, perhaps perceptually based, motion metrics.

Our implementation was latency-optimised so our consistency results would generalise to a real system, but feature-wise it is only a proof-of-concept. While we considered rigid-bodies, the same approach could transfer any continous parameter, such as the vertex positions of deformable meshes, animations or other scene graph parameters (e.g. material settings). The major theoretical limitation is bandwidth, as the more dimensions transferred per object, the fewer dynamic objects that can be supported at an equivalent bandwidth. Practically, what can be streamed will be dependent on what the engine makes available programatically. It is likely a real implementation would not have fixed messages but rather a way to address arbitrary properties. This would have higher overhead than we show in Section 9.2 however.

10 Conclusion

In this paper we consider streaming VR for mobile headsets. The necessary trade-off of compute power for ergonomics means developers must reduce the richness of the experience on mobile HMDs, or offload complexity to a server. Render streaming has been a popular approach on desktops and consoles. HMDs however have far tighter latency requirements than these. While popular, render streaming is far from the only streaming approach in the literature. Instead, we propose re-evaluating scene streaming - turning the mobile HMD into a thin client analogous to a cluster render node. Such a client would keep the most latency sensitive loops local to the device, always show a pixel-perfect image, and maximise use of local compute resources.

Traditionally scene-graph synchronisation has been used for geographically distributed VEs, with accurate prediction and synchronsation schemes compensating for poor QoS. In contrast, edge physics relies on link quality to reduce the computational load of the client and offer the best load distribution for new headsets with power-walled CPUs and increasingly powerful GPUs.

Scene-synchronisation for DVEs developed from cluster rendering. We suspect that treating a mobile headset as a rendering node in an edge-computing configuration has the potential to facilitate high quality VR on low power devices. While similar problems however, edge-computing is not the same as cluster rendering. Edge clients are heterogenous, with more autonomy than render nodes, running over a network beyond the developer's control, and so cannot be tightly synchronised into a single system like a cluster. The potential of a mobile headset to act as a dumb terminal, and how it will perform as one, is unclear.

To investigate this, we implemented and tested a prototype system in a number of conditions. We identified two forms of error that should be general to remote visualisation of real-time multi-object simulations (e.g. games, physics simulations, collaborative VR, etc) and demonstrated asymmetrical sensitivites of each to configuration and QoS. We showed our system could maintain very low absolute error with the QoSs expected in edge-computing. The biggest hurdle for edge physics is to handle transient drops in update rates in a transparent way, as low-cost predictors are inaccurate over timescales beyond a few 10s of milliseconds.

Based on our results so far, compared to video streaming edge physics has orders of magnitude less computational overhead, an order of magnitude less bandwidth for typically sized scenes, and half as much latency. The scene-graph representation has limited parameters, simplifying the protocol and making it easy to create a highly optimised implementation. We validate our implementation demonstrating the advantages of edge computing. An Oculus Quest explores a scene too complex for even a VR-desktop to render alone, but coupling both wih edge-physics allows it to be seen comfortably at native frame-rates.

As we consider the progress of standalone HMDs in the context of next generation communication technologies, it is still necessary to ask what is best to stream. Our results begin to examine the potential of scene-streaming for this. Additionally, our results apply to co-located collaborative VR, which may be more easily supported with something like edge physics than a traditional DVE.

References

[1] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04 Network and system support for games - SIGCOMM 2004 Workshops*, p. 161. ACM Press, New York, USA, 2004. doi: 10.1145/1016540.1016559

[2] A. U. Batmaz and W. Stuerzlinger. Effects of 3D Rotational Jitter and Selection Methods on 3D Pointing Tasks. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 1687–1692. IEEE, 3 2019. doi: 10.1109/VR.2019.8798038

[3] Y. W. Bernier. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. Technical report, Valve, 2001.

[4] F. Chardavoine, S. Ageneau, and B. Ozell. Wolverine: A Distributed Scene-Graph Library. *Presence: Teleoperators and Virtual Environments*, 14(1):20–30, 2005. doi: 10.1162/1054746053890297

[5] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools and Applications*, 23(1):7–30, 2004. doi: 10.1023/B:MTAP.0000026839.31028.9f

[6] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler. Toward Low-Latency and Ultra-Reliable Virtual Reality. *IEEE Network*, 32(2):78–84, 2018. doi: 10.1109/MNET.2018.1700268

[7] S. Friston, E. Griffith, D. Swapp, A. Marshall, and A. Steed. Profiling Distributed Virtual Environments by Tracing Causality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 238–245. IEEE, 3 2018. doi: 10.1109/VR.2018.8446135

[8] Google. *Stadia*, 2020 (Accessed May 2020). `https://stadia.google.com/`.

[9] A. Guéziec, G. Taubin, B. Horn, and F. Lazarus. A Framework for Streaming Geometry in VRML. *IEEE Computer Graphics and Applications*, 19(2):68–78, 1999. doi: 10.1109/38.749125

[10] L. Gupta, R. Jain, and H. A. Chan. Mobile Edge Computing – An Important Ingredient of 5G Networks, 2016.

[11] M. Halpern, Y. Zhu, and V. J. Reddi. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. *Proceedings - International Symposium on High-Performance Computer Architecture*, 2016-April:64–76, 2016. doi: 10.1109/HPCA.2016.7446054

[12] D. Hanawa and T. Yonekura. On the error modeling of dead reckoned data in a distributed virtual environment. *Advanced Modeling and Optimization*, 7(1):85–98, 2005. doi: 10.1109/CW.2005.69

[13] D. Hanawa and T. Yonekura. A Proposal of Dead Reckoning Protocol in Distributed Virtual Environment based on the Taylor Expansion. In *2006 International Conference on Cyberworlds*, pp. 107–114. IEEE, 2006. doi: 10.1109/CW.2006.10

[14] D. Hanawa and T. Yonekura. Improvement on the accuracy of the polynomial form extrapolation model in distributed virtual environment. In *Visual Computer*, vol. 23, pp. 369–379, 2007. doi: 10.1007/s00371-007-0109-8

[15] G. Hesina, D. Schmalstieg, A. Furhmann, and W. Purgathofer. Distributed Open Inventor. In *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '99*, pp. 74–81. ACM Press, New York, New York, USA, 1999. doi: 10.1145/323663.323675

[16] J. Hladky, H.-P. Seidel, and M. Steinberger. The camera offset space. *ACM Transactions on Graphics*, 38(6):1–14, 11 2019. doi: 10.1145/3355089.3356530

[17] J. Hnidek. Network protocols for applications of shared virtual reality. *19th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG 2011 - In Co-operation with EUROGRAPHICS, Full Papers Proceedings*, pp. 31–38, 2011.

[18] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan. Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys 2016*. ACM Press, 2016. doi: 10.1145/2967360.2967369

[19] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A scalable graphics system for clusters. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, pp. 129–140, 2001.

[20] L. N. Huynh, R. K. Balan, and Y. Lee. DeepSense: A GPU-based deep convolutional neural network framework on commodity mobile devices. In *WearSys 2016 - Proceedings of the 2016 Workshop on Wearable Systems and Applications, co-located with MobiSys 2016*, pp. 25–30, 2016. doi: 10.1145/2935643.2935650

[21] J. Jerald and M. Whitton. Relating Scene-Motion Thresholds to Latency Thresholds for Head-Mounted Displays. In *Proceedings of the 2009 IEEE Virtual Reality Conference*, pp. 211–218. IEEE, 3 2009. doi: 10.1109/VR.2009.4811025

[22] H. G. Kim and S. W. Kim. An improvement of dead reckoning algorithm using Kalman filter for minimizing network traffic of 3D on-line games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3768 LNCS:676–687, 2005. doi: 10.1007/11582267

[23] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee. Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices. *IEEE Transactions on Mobile Computing*, 19(7):1586–1602, 7 2020. doi: 10.1109/TMC.2019.2913364

[24] E. Lamboray, S. Würmlin, and M. Gross. Real-time streaming of point-based 3D video. In *Proceedings - Virtual Reality Annual International Symposium*, pp. 91–98, 2004. doi: 10.1109/VR.2004.1310060

[25] R. W. Lau and K. Lee. On error bound estimation for motion prediction. In *Proceedings - IEEE Virtual Reality*, pp. 171–178. IEEE, 2010. doi: 10.1109/VR.2010.5444795

[26] J. J. LaViola. Double exponential smoothing: An alternative to Kalman filter-based predictive tracking. In *Proceedings of the Workshop on Virtual Environments, EGVE'03*, pp. 199–206, 2003. doi: 10.1145/769953-769976

[27] S. Li, C. Chen, and L. Li. A new method for path prediction in network games. *Computers in Entertainment*, 5(4):1–12, 2008. doi: 10.1145/1324198.1324206

[28] N. H. Lin, T. H. Huang, and B. Y. Chen. 3D Model streaming based on JPEG 2000. *IEEE Transactions on Consumer Electronics*, 53(1):182–190, 2007. doi: 10.1109/TCE.2007.339523

[29] B. MacIntyre and S. K. Feiner. A distributed 3D graphics library. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, pp. 361–370, 1998. doi: 10.1145/280814.280935

[30] D. Margery, B. Arnaldi, and N. Plouzeau. A General Framework for Cooperative Manipulation in Virtual Environments. In *Proceedings of the Eurographics Workshop in Vienna, Austria, May 31-June 1, 1999*, Eurographics, pp. 169–178. Springer Vienna, Vienna, 1999. doi: 10.1007/978-3-7091-6805-9

[31] D. Meng, Y. P. Yao, and F. Yao. An enhanced dead reckoning algorithm with hybrid extrapolation models (AisaSim 2016). *International Journal of Modeling, Simulation, and Scientific Computing*, 8(2):1–14, 2017. doi: 10.1142/S1793962317500271

[32] A. H. P. Morice, I. A. Siegler, and B. G. Bardy. Action-perception patterns in virtual ball bouncing: Combating system latency and tracking functional validity. *Journal of Neuroscience Methods*, 169(1):255–266, 2008. doi: 10.1016/j.jneumeth.2007.11.020

[33] J. H. Mueller, P. Voglreiter, M. Dokter, T. Neff, M. Makar, M. Steinberger, and D. Schmalstieg. Shading atlas streaming. *ACM Transactions on Graphics*, 37(6):1–16, 1 2019. doi: 10.1145/3272127.3275087

[34] M. Naef, E. Lamboray, O. Staadt, and M. Gross. The blue-c distributed scene graph. In *Proceedings of the 2003 IEEE Virtual Reality Conference*, pp. 275–276. IEEE Comput. Soc, 2003. doi: 10.1109/VR.2003.1191157

[35] NVidia. *NVidia Shield*, 2020 (Accessed May 2020). `https://www.nvidia.com/en-us/shield/`.

[36] J. Ohlenburg. Improving collision detection in distributed virtual environments by adaptive collision prediction tracking. In *Proceedings - Virtual Reality Annual International Symposium*, pp. 83–90, 2004. doi: 10.1109/VR.2004.1310059

[37] S. Olbrich and H. Pralle. Virtual reality movies-real-time streaming of 3D objects. *Computer Networks*, 31(21):2215–2225, 1999. doi: 10.1016/S1389-1286(99)00097-3

[38] J. Orlosky, K. Kiyokawa, and H. Takemura. Virtual and Augmented Reality on the 5G Highway. *Journal of Information Processing*, 25(0):133–141, 2017. doi: 10.2197/ipsjjip.25.133

[39] C. Ozcinar, J. Cabrera, and A. Smolic. Visual Attention-Aware Omnidirectional Video Streaming Using Optimal Tiles for Virtual Reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):217–230, 2019. doi: 10.1109/JETCAS.2019.2895096

[40] J. Park, P. A. Chou, and J. N. Hwang. Volumetric Media Streaming for Augmented Reality. In *2018 IEEE Global Communications Conference, GLOBECOM 2018 - Proceedings*, pp. 1–6. IEEE, 2018. doi: 10.1109/GLOCOM.2018.8647537

[41] K. Park and R. Kenyon. Effects of network characteristics on human performance in a collaborative virtual environment. *Proceedings of the 1999 IEEE Virtual Reality Conference*, 1999.

[42] D. Pohl, S. Nickels, R. Nalla, and O. Grau. High quality, low latency in-home streaming of multimedia applications for mobile devices. *2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014*, 2:687–694, 2014. doi: 10.15439/2014F42

[43] PolyStream. *About Us*, 2020 (Accessed May 2020). `https://polystream.com/about-us/`.

[44] R. Pusch. Explaining how fighting games use delay-based and rollback netcode. *Ars Technica*, 2019.

[45] S. Rossi and L. Toni. Navigation-aware adaptive streaming strategies for omnidirectional video. *2017 IEEE 19th International Workshop on Multimedia Signal Processing, MMSP 2017*, 2017-Janua:1–6, 2017. doi: 10.1109/MMSP.2017.8122230

[46] G. Sankaranarayanan and B. Hannaford. Virtual Coupling Schemes for Position Coherency in Networked Haptic Environments. In *The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, pp. 853–858. IEEE, 2006. doi: 10.1109/BIOROB.2006.1639197

[47] S. Shi, V. Gupta, M. Hwang, and R. Jana. Mobile VR on edge cloud. In *Proceedings of the 10th ACM Multimedia Systems Conference*, pp. 222–231. ACM, New York, NY, USA, 6 2019. doi: 10.1145/3304109.3306217

[48] X. B. Shi, X. Wang, J. Bi, F. Liu, D. Yang, and X. Y. Liu. A DR algorithm based on artificial potential field method. *Multimedia Tools and Applications*, 45(1-3):247–261, 2009. doi: 10.1007/s11042-009-0296-6

[49] S. K. Singhal and D. R. Cheriton. Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality. *Presence: Teleoperators and Virtual Environments*, 4:169–193, 1995.

[50] F. Stakem and G. AlRegib. An adaptive approach to exponential smoothing for CVE state prediction. In *IMMERSCOM '09: Proceedings of the 2nd International Conference on Immersive Telecommunications*, pp. 1–6. ACM Press, 2009. doi: 10.4108/immerscom.2009.17

[51] A. Tumanov, R. Allison, and W. Stuerzlinger. Variability-Aware Latency Amelioration in Distributed Environments. In *Proceedings of the 2007 IEEE Virtual Reality Conference*, pp. 123–130. IEEE, 2007. doi: 10.1109/VR.2007.352472

[52] Valve. *Steam Link*, 2020 (Accessed May 2020). `https://store.steampowered.com/steamlink/about/`.

[53] G. Voß, J. Behr, D. Reiners, and M. Roth. A multi-thread safe foundation for scene graphs and its extension to clusters. In *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pp. 33–37, 2002. doi: 10.1145/569673.569679

[54] B. Zeleznik, L. Holden, M. Capps, H. Abrams, and T. Miller. Scene-Graph-As-Bus: Collaboration between Heterogeneous Stand-alone 3-D Graphical Applications. *Computer Graphics Forum*, 19(3):91–98, 2000. doi: 10.1111/1467-8659.00401

[55] C. Zhou, Z. Li, and Y. Liu. A measurement study of oculus 360 degree video streaming. In *Proceedings of the 8th ACM Multimedia Systems Conference, MMSys 2017*, pp. 27–37, 2017. doi: 10.1145/3083187.3083190

A Linear Model (Appendix)

Previous works attempted to model expected error. We fit a linear model to our data to see if we can make such predictions. Each predictor had its own model fitted.

The model was defined[1] as

$$error \sim -1 + speed + latency : speed + jitter : speed + packetloss : speed$$

based on prior observations and pre-tests. We use the absolute error as this is what previous works used.

Table 7 shows the statistically significant coefficients for each condition.

The low R^2 values show the model is not of much use and so not generally interesting. However there are some things worth remarking on.

Sensitivity to latency decreases with reference time scope, as expected. Packetloss is insignificant; likely because we maintain high update rates in general. The coefficients for jitter are negative. This may seem erroneous, but our emulator uses the exact definition of variance so can reduce latency as well as increasing it.

When we change the definition of latency from the network condition to 'experienced latency', the packet loss does become significant. The R^2 values are not greatly affected, so the model is no more use. However it does suggest there is potential to use the value for control purposes (e.g. as per Meng et al [31]).

	Predictor		
Coefficient	H	L	G
Speed	0.002	0.005	0.007
Speed:Jitter	-3.195	-2.411	-2.971
Speed:Packetloss			
Speed:Latency	0.511	0.413	0.384
R^2	0.162	0.254	0.168

Table 7: Linear regression coefficients for all error measures for each prediction condition

[1] in Wilkinson notation