Towards Efficient Edge Cloud Augmentation for Virtual Reality MMOGs

Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri
WINLAB, Rutgers University, NJ, U.S.A. Email: {wuyang, jiachen, yyzhang, ray}@winlab.rutgers.edu

ABSTRACT

With the popularity of Massively Multiplayer Online Games (MMOGs) and Virtual Reality (VR) technologies, VR-MMOGs are developing quickly, demanding ever faster gaming interactions and image rendering. In this paper, we identify three main challenges of VR-MMOGs: (1)a stringent latency requirement for frequent local view change responses, (2) a high bandwidth requirement for constant refreshing, and (3)a large scale requirement for a large number of simultaneous players. Understanding that a cloud-centric gaming architecture may struggle to deliver the latency/bandwidth requirements, the game development community is attempting to leverage edge cloud computing. However, one problem remains unsolved: how to distribute the work among the user device, the edge clouds, and the center cloud to meet all three requirements especially when users are mobile.

In this paper, we propose a hybrid gaming architecture that achieves clever work distribution. It places local view change updates on edge clouds for immediate responses, frame rendering on edge clouds for high bandwidth, and global game state updates on the center cloud for user scalability. In addition, we propose an efficient service placement algorithm based on a Markov decision process. This algorithm dynamically places a user's gaming service on edge clouds while the user moves through different access points. It also co-places multiple users to facilitate game world sharing and reduce the overall migration overhead. We derive optimal solutions and devise efficient heuristic approaches. We also study different algorithm implementations to speed up the runtime. Through detailed simulation studies, we validate our placement algorithms and also show that our architecture has the potential to meet all three requirements of VR-MMOGs.

Research supported by the National Science Foundation, Future Internet Architecture - Next Phase (FIA-NP) grant CNS-1345295.

SEC '17, October 12–14, 2017, San Jose / Silicon Valley, CA, USA

ACM ISBN 978-1-4503-5087-7/17/10…$15.00
https://doi.org/10.1145/3132211.3134463

CCS CONCEPTS

• **Networks → Network architectures**;

ACM Reference Format:
Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri. 2017. Towards Efficient Edge Cloud Augmentation for Virtual Reality MMOGs. In *Proceedings of SEC '17, San Jose / Silicon Valley, CA, USA, October 12–14, 2017,* 14 pages.
https://doi.org/10.1145/3132211.3134463

KEYWORDS

edge computing, network architecture, gaming architecture, virtual reality MMOGs

1 INTRODUCTION

The rapid rise of Massively Multiplayer Online Games (MMOGs) calls for gaming platforms that support ultra low latency and intensive 3D world rendering [1]. With emerging Virtual Reality (VR) technologies (*e.g.*, HTC Vive, Oculus, Google Cardboard), VR based MMOGs, *i.e.*, VR-MMOGs, are looming on the horizon, demanding even faster gaming interaction and image rendering. In addition, VR-MMOGs place a new set of requirements on the underlying system design due to a union of VR and MMOGs: *1*) the need to simultaneously render two images with different perspectives for both eyes, *2*) the need to support wider angles of visual field (120° compared to 60° in normal games), *3*) the need to provide ultra-low latency that prevents people from having motion sickness (<30ms compared to 100–1000ms in normal games), and *4*) the need to render with a higher refresh rate (60–120 frames per second (fps), compared to 24–60fps in normal games).

Meanwhile, to enable players with "thin" mobile devices (*e.g.*, smartphones, pads, TVs) to enjoy high-quality gaming, game creators have proposed and developed a cloud gaming paradigm (also known as gaming on-demand [1]) that delivers games to players from the cloud. GeForce NOW (for Nvidia Shield clients [2]), PlayStation NOW [3], GamingAnywhere [4], as the industrial pioneers of cloud gaming, are

drawing substantial numbers of players from traditional gaming to this cloud-based paradigm [5]. These cloud gaming services perform game logic computation on cloud servers and stream encoded views over the Internet to apps on heterogeneous mobile devices. Cloud gaming allows users to access games anywhere via mobile devices without periodically upgrading their hardware to satisfy the ever increasing hardware demands. It also significantly reduces the energy consumption incurred by heavyweight rendering tasks on mobile devices. Finally, high resolution frames that are cumbersome, if not impossible, to generate locally can be streamed from the cloud.

While cloud gaming can provide a multitude of benefits to players, its service providers face a set of serious challenges to ensure quality of service (QoS). The first and foremost challenge stems from the *latency* between cloud servers and players. Responses that are not fast enough in VR gaming can result in dissatisfying game experiences and contribute to player motion sickness. According to [6], ~20ms is an acceptable end-to-end latency for such applications. A latency of 50ms can still support responsive services but with noticeable lagging. Unfortunately, the average network latency in today's Internet between the Amazon EC2 cloud and mobile devices is more than 80ms [1], which exceeds the tolerable latency level even without performing any computations.

The second challenge is the high *bandwidth* demand of MMOGs – they generally require a bandwidth of 100Mbps to stream VR games with 1080p resolution at 60 fps [7], while the wireless Internet bandwidth available to a mobile device is 2Mbps [8]. Network jitters cause decreased refreshing rates and increased packet delays, both of which worsen user experience. Moreover, users with mobile devices are more inclined to move around compared to those connected to fixed hosts, and some may even play while sitting in cars or trains. The disconnections caused by changing network access points can also lead to deteriorated gaming performance.

Edge cloud computing [8–10] moves cloud services closer to the users. It has the potential to bring down unacceptable network delays and to provide high downlink bandwidth while taking advantage of high performance computing resources. VR-MMOGs can leverage edge cloud computing to meet their QoS requirements [11], but simply moving all the gaming tasks to the edge makes it harder for players to share games across the network since it is difficult to synchronize users' profiles and game worlds among widely distributed edge clouds.

To address the challenges, we take a closer look at the game flows in VR-MMOGs and discover that player-initiated events can generally be classified into two categories based on the tolerance levels of response latency. The response to the user's local *view change events* (which has effect only on his/her screen, *e.g.*, mouse movements, map scrolls, selection of a game object without changing it) has much more stringent timeliness requirements compared to the response to the *game events* (which involves global game state updates, *e.g.*, updated scores, bleeding on shot targets). In VR-MMOGs, view change events occur much more frequently than in non-VR-MMOGs because the orientation of the VR device constantly changes and requires immediate (~20ms) feedback on the screen. In game events, on the other hand, players can tolerate more than 100ms latency, and in some games this value can be as large as 1 second [6].

Based on the fundamental differences between view change and game events, we believe that they should be treated differently in order to provide the best VR-MMOG user experience. In this paper, we propose EC+ , an architecture for Edge Cloud augmented VR-MMOGs. EC+ exploits edge clouds in view change event rendering to satisfy the ultra low delay requirement. This rendering on edge clouds can also provide a higher resolution and refresh rate compared to rendering on mobile devices because edge clouds have more computation power. For game events, on the other hand, EC+ uses a central cloud to manage global game and game logic. This provides wide coverage with minimal overhead in maintaining game state consistency.

In addition to proposing the EC+ architecture, we also devise an efficient algorithm that selects an edge cloud for each player in order to handle player mobility and dynamic edge cloud workload. Modeled upon a Markov decision process (MDP), the proposed algorithm periodically makes edge cloud placement decisions, taking into consideration the overall QoS (the latency and bandwidth between client and edge, *and* between edge and game server), mutual impact among players (*e.g.*, edge load, game world sharing), and player mobility patterns. To ensure feasibility, we come up with an approach that reduces the algorithm complexity in both storage and execution time. We also design a mechanism to ensure seamless handoff when a gaming service migrates from one edge cloud to another.

We summarize our contributions below:

- A study of the new requirements of VR-MMOGs and explanation of how client-centric and cloud gaming fall short in fulfilling these requirements (§3);
- A hybrid architecture design that leverages both edge and central clouds to satisfy the latency and throughput requirements of VR-MMOGs (§4);
- A general edge cloud placement algorithm which maximizes game performance for a large number of players with different bandwidth, latencies, edge loads, and game world sharing scenarios (§5); and
- A comprehensive evaluation using both synthetic and real-world topologies to quantify the benefit of the proposed architecture and algorithm (§6).

2 RELATED WORK

We first present the background of (VR-)MMOGs and then review the existing solutions that could potentially support VR-MMOGS, namely, cloud centric gaming and edge cloud assisted gaming.

2.1 Massively Multiplayer Online Gaming Meets Virtual Reality

Virtual reality (VR) has been supported by multiple industrial products like PlayStation.VR [3], HTC Vive [12], Oculus [6], Google Cardboard [13]. VR devices extract players' sensory (*e.g.*, eyes and ears) information and accordingly "hijack" the natural stimulation with the artificial stimulation from a virtual world generator [14]. VR technology remarkably hands a highly immersive experience with substantial depth perceptions. Playing MMOGs through VR devices is the natural next step [15]. In fact, the VR versions of several popular MMOGs have been developed, *e.g.* World of Warcraft, Minecraft Multiplayer, and Grand Theft Auto V Online [16].

As much as VR-MMOGs generate excitement in the gaming community, it also poses the unprecedented demands and the challenges, especially with respect to providing ultra low latency and a high refresh rate, on the underlying system design. This paper aims to design an architecture that is carefully tuned to satisfy these demands.

2.2 Supporting Gaming through Cloud

Many cloud gaming solutions [2–4, 17, 18] have been proposed to reduce the computation and/or storage requirements on game terminals. These solutions can be broadly classified into two categories: file-streaming games and video-streaming games. In file-streaming gaming (*i.e.*, progressive downloading), a small portion of the game is initially downloaded to a user device. While this portion runs, the rest of game can be downloaded and installed in parallel [17, 18]. While it is true that file-streaming games can reduce the game boot time and the storage required on game devices, it still requires devices to process game logic and perform 3D rendering. Therefore, it is difficult to support VR-MMOGs on mobile devices like Google cardboard.

Video-streaming games, on the other hand, place all the processing in a cloud, including user profile management, game update calculation, game frame rendering and encoding, *etc.* The cloud then streams the encoded frames to players over the Internet [2–4]. This mechanism enables players to enjoy high-quality games even on the devices with limited computing and power resources (*e.g.*, smartphones, pads, TVs) – the game terminals merely need to decode frames just like watching a Youtube video. With the advent of GPU grids [19], game processing has become more efficient than purely using CPU-based clouds.

Many studies have been conducted to further improve the user experience of video-streaming games. In [20] video games are classified into CPU-consuming and memory-consuming types to increase the resource utilization in a cloud. Lee *et al.* [21] use High Efficiency Video Coding (HEVC) to reduce the bandwidth requirement by 59% without compromising video quality. Solutions in [22, 23] reduce the response time by predicting possible game updates and rendering speculative frames ahead of time.

The main disadvantage of cloud gaming, however, involving both file-streaming and video-streaming gaming, is the need to transmit a large amount of data, either a game itself or game frames, through the core network. Due to the massively multiplexing nature of the Internet design, the available bandwidth and latency between a cloud and a player may change dramatically over time [24]. This often leads to jitters, lags, frame drops or low-quality frames (glitches) in the middle of a game, and resulting in a poor gaming experience, especially for video-streaming games [25].

2.3 Edge Cloud Computing

Edge cloud (or fog computing [8–10]) moves computing and storage closer to clients, promising to deliver shorter latency and higher bandwidth. It can benefit applications which require high bandwidth, low latency but without large-scale aggregation, *e.g.*, preprocessing of surveillance camera data [26], image classification [27], smart traffic light control [9], *etc.* Edge cloud computing also has the potential to better serve (VR-)MMOGs, if carefully designed to solve the challenge of large-scale aggregation (*i.e.*, game state synchronization among all players).

Work in [28–30] proposes peer-to-peer (P2P) MMOGs wherein the delegate of the players (game consoles or edge cloud servers) form a P2P network to synchronize gaming states shared among the players directly. This distributed architecture incurs a large amount of synchronization overhead and may potentially limit the number of concurrent players in a game. To address these challenges, the authors in [31] propose to move the rendering process from a cloud to idle desktops which are close to clients. Indeed, this technique can reduce the network latency by 20% and reduce the network traffic volume by 90%. Nevertheless, any user events, including those only need local updates, are dispatched to the center cloud altogether. This solution performs well for non-VR games because there aren't many local update events in these games. However, VR-MMOGs have a significantly larger number of such events, which makes this solution ill-suited.

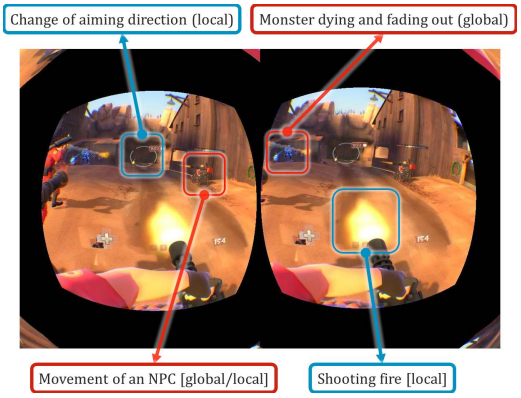


Figure 1: A game view usually contains both view change updates local to a player (*e.g.*, change of look direction, immediate feedback on action like firing) and game world updates synchronized among all players (*e.g.*, monsters' dying, non-player characters' actions).

2.4 Service Migration Among Clouds

To satisfy specific application requirements, tasks have to be initially placed on assigned machines of a cloud [32–34]. Later, the tasks may be migrated (reassigned) to underutilized machines to meet particular optimization targets. Concerning where to migrate, distinct migration strategies have been proposed on the basis of expected optimal targets. Lim *et al.* [35] propose a performance aware migration schema in respond to dynamic server workloads. Ghribi *et al.* [36] investigate an energy efficient scheduling to achieve significant energy savings. With respect to how to migrate, Douglis [37] comes up with a process migration schema that moves a process from a source machine to a destination machine, which encounters the difficulty of separating a process from its operating system. Clark *et al.* [38] design a live virtual machine (VM) migration mechanism that effectively overcomes this barrier. Yet, a core cost of VM migration is a short downtime during which an application is compulsively paused. The downtime changes among different applications, ranging from several milliseconds to several seconds [39]. To reduce the downtime, Jin *et al.* [40] investigate a memory compression approach and Ha *et al.* [41] study a pipelining processing of VM migration .

3 A CLOSER LOOK AT VR-MMOGS

A VR-MMOG is essentially a large-scale event driven system. Even though each VR-MMOG may have unique and complicated game logic, they do have similar game events and share an identical underlying game flow. In this section, we first study the new features and challenges of VR-MMOGs and then present several existing MMOG models to help understand why they fail to satisfy these new requirements.

3.1 View Change Events *vs.* Game Events

Any game flow begins with a particular user event. When playing a game, a player can trigger a user event through external devices including mouse, keyboard, VR headset, *etc.* Clicking a mouse at a certain point in a game world, pressing a particular key, or changing the orientation of head (while wearing a VR headset), for example, each entails a user event. However, players have different delay expectations on different kinds of events, and therefore a game architecture should treat these events differently.

We realize that there are two fundamental types of user events, namely, (local) view change events and (global) game events. The first type of user events– view change events – only causes transient changes to a user's perspective, but leaving his/her game world intact. For instance, a user event of clicking a mouse at the location (153, 85) might be interpreted as selecting a troop from a player's army. The chosen soldiers will be highlighted on the player's screen, but this event is invisible to other players.

The second type of user events– game events – not only causes changes to a player's perspective, but also cause permanent updates to a player's game world, which we refer to as game events. In (VR-)MMOGs, such updates should be synchronized among all the players who can see this game event. For example, the same mouse click at (153, 85) might be interpreted as "player A punches player B" or "player A collects 100 golds from the ground", both causing changes to a game world and deemed as a game event. Since a same user event can be interpreted differently based on a particular game logic and a particular game world, MMOGs usually include a module to distinguish the type of user events before sending them all the way to a game server.

As shown in Fig. 1, all user events, no matter which type, will eventually be reflected on a user interface. However, players do have different expectations on the feedback delay. According to [6], players have different tolerance levels for game events, ranging from 100 milliseconds (*e.g.*, first person shooting games) to 1 second (*e.g.*, real-time strategy games). A number of studies have been conducted to reduce the game event response latency by optimizing server scheduling [20], improving rendering algorithm and hardware [19] and optimizing network dissemination [42].

Compared to game events, we find it counterintuitive that players expect much shorter feedback delays for view change events. Immediate local view updates lead to a smooth game control and a seamless user experience. Here, the tolerable latency varies from tens of milliseconds (*e.g.*, orientation changes in 3D games) to a couple of hundred milliseconds (*e.g.*, keystrokes). This latency is much more critical to video-streaming games since a renderer resides much farther away in a cloud rather than in a local GPU.

Table 1: Comparison between event types in VR-MMOGs.

	View change	Game
Tolerable latency (ms)	20	100
Event size (bytes)	180	90
Frequency (events/sec)	95	5

The high frequency of such events of VR-MMOGs makes matters even worse. We compare the features between view change events and game events in Table 1 based on several studies on games [6, 43–45]. With constant orientation changes in a VR-MMOG (view change events), such changes usually require the feedback delay of less than 20ms if possible [6], to ensure a pleasant user experience. Players would experience dizziness when the latency increases beyond 50ms. This ultra-low latency makes it almost infeasible for a central clouds to support VR video-streaming games as the average latency between Amazon EC2 and clients is already above 80ms. The amount of orientation change events is another challenge as games usually try to get accelerometer and gyroscope readings more than 50 times/second, some high-end devices like HTC VIVE can even reach 100. This frequency is much higher compared to view change events. Game events, estimated by game actions per minute (APM), is usually around 50 and maximized at 300 with proficient players [45].

Therefore, we believe that view change events should be treated as "first-class citizens" and a gaming architecture should be carefully designed to provide a better support for these events in VR-MMOGs.

3.2 Overview of Existing MMOG Architectures

We study the underlying communication and computation models of the existing MMOG architectures to understand why these solutions fall short in supporting VR-MMOGs. We classify them into two categories: traditional client-centric gaming and cloud-centric gaming, based on where rendering happens. While file-streaming gaming also gets support from a cloud, its game model is almost equal to client-centric games as it requires a local powerful game consoles for rendering. The flowcharts of the two game models are shown in Fig. 2a and 2b.

3.2.1 *Traditional Client-Centric Gaming.*

A traditional gaming architecture performs most of the tasks on the client side except the synchronization of a shared game world. As shown in Fig. 2a, a game loop starts with a capture of a user event (*e.g.*, mouse click at (153, 85)). This event is firstly sent to a local game event calculator (step $S1$) that detects whether it is a game event. In most games, a game event also triggers a view change event (*e.g.*, shooting fire in Fig. 1), and therefore, a view change rendering request is sent to a render (step $S2'$). If the event is a game event (*e.g.*, player A punches B), this game event will be forwarded to a game server (step $S2$).

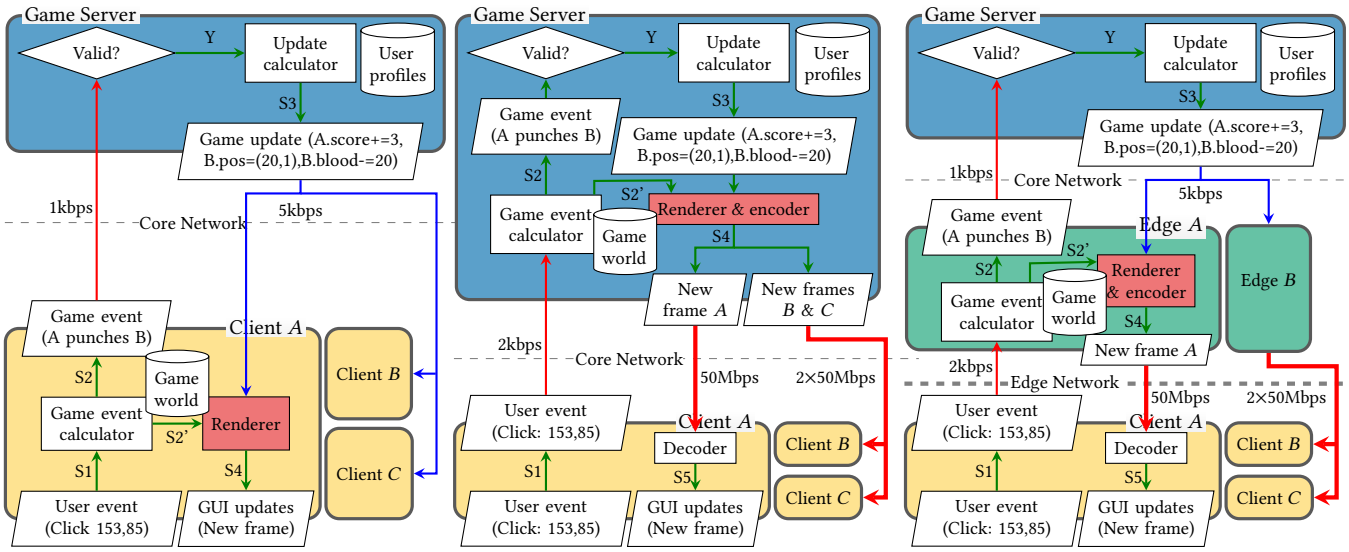
Table 2: Refresh rate (fps) comparison between mobile devices and desktop machines in different games .

Game	Resolution	Refresh rate (fps)	
		Desktop	Mobile
StarCraft II	1024×768	380	55
	1280×1024	136	13
	1920×1080	119	5
GTA V	1024×768	168	10.8
	1280×1024	161	<5
	1920×1080	136	<5

Once receiving the game event at the server, a validation module checks the validity of the event according to a game logic (*e.g.*, if A is allowed to punch B, if A is close enough to reach B, etc). It will discard suspiciously cheating events (possibly generated by a game bot) and stale events (caused by network delays). A update calculator at the server then computes the "consequences" (game updates, *e.g.*, A gets 3 points, B retreats by 1 step and looses 20 blood) of each valid game event and accordingly updates users' profiles. A user profile usually consists of individual state information of a game character (*e.g.*, the current location, the experience point and the skill level). To avoid transmitting too many small game updates, the server usually accumulates game updates for a small period of time and sends all the updates in the period as a batch (step $S3$). The length of the period is usually determined by the smallest frame interval among all players (*e.g.*, 33ms for 30fps clients). When all players share a same synchronized game world, the same updates should reach all of them, and therefore they can be sent via multicast or broadcast to improve the efficiency in dissemination.

A renderer on the client side usually renders a game world periodically (30-60 frames per second) to reflect outputs of view change events (from $S2$) and game updates (from $S3$). In most games, a renderer may generate frames even without any updates to reflect, for example, variations of luminous intensity, flowing of water and/or moving of non-player characters (NPC). Rendering projects geometry, viewpoint, texture, lighting, and shading upon 3D skeletal objects in a game world, and finally outputs a game frame in a game interface like screens or VR devices (step $S4$).

This architecture usually requires a game client to have abundant CPU, GPU and RAM resources since rendering a frame involves a large number of matrix multiplication and floating point operations. Thus, it is not friendly for players who prefer to enjoy games with their mobile devices. For example, an average desktop GPU like AMD Radeon HD



(a) Traditional gaming **(b) Video streaming gaming** **(c) Edge cloud augmented gaming**

Figure 2: Comparison of Different MMOG Architectures (red line: unicast, blue line: multicast).

7970M, can reach 380, 136 and 119 fps for StarCraft II at resolutions 1024x768, 1280x1024 and 1920x1080 respectively. However, the refresh rate on Intel HD Graphics Cherry Trail (a GPU adopted on Microsoft Surface 3 tablets) can only deliver 55, 13 and 5 fps with the same resolution (see Table 2). It means that, if a player tries to play this game on a Microsoft Surface 3 tablet, he/she can only choose the resolution of 1024x768 or below. A same restriction can be found in most popular MMOGs like GTA V, Minecraft, *etc.* Yet, to enjoy an immersive VR gaming experience, players should not be constrained by desktop machines and cables.

3.2.2 *Cloud-Centric Video Streaming Gaming.*

Cloud-centric video streaming gaming [2–4] significantly reduces the resource requirement on user devices. All rendering tasks will be executed in a central cloud as shown in Figure 2b. Specifically, a client device merely sends user events directly to a cloud, and receives subsequent updated frames. This video streaming gaming architecture promises to enable players to enjoy MMOGs on mobile devices, while several important challenges must be addressed to support VR-MMOGs by this architecture.

Firstly, it is demanding for this architecture to satisfy the ultra low latency requirement of VR-MMOGs. In particular, a user may desire a view change rendering to be completed within 20ms [6]. With latency of 50ms, VR games can still respond to a user's input, but with noticeable lagging, which may lead to an undesirable user experience. Unfortunately, the average network delay between Amazon EC2 and a mobile device is around 80ms [1], which is much longer than the preferred latency of view change events.

Secondly, it is also challenging for this architecture to support a high refreshing rate. VR-MMOGs generally require bandwidth of 50Mbps to stream a video with a 1080p resolution at 60 fps, while the available bandwidth in the wireless Internet to a mobile device is merely about 2Mbps [8]. In the traditional gaming architecture, a game server only needs to send game updates to clients, which can be multicasted to minimize the network traffic. In this architecture, however, a game server needs to send distinct rendered frames to each individual client. As a result, unicast is required in this scenario. Everything considered, network bandwidth needed in this architecture increases significantly compared to the traditional gaming architecture.

4 EC+: A VR-MMOG ARCHITECTURE AUGMENTED BY EDGE CLOUDS

In the previous section, we discuss that traditional client-centric gaming places heavy-weight rendering tasks on the game client which essentially prevents users from playing VR-MMOGs on mobile devices. Video-streaming gaming intends to facilitate mobile devices but eventually fails to do so due to slow responses and poor video quality. It is thus desirable to leverage a third computing platform that has sufficient resources while incurring short network latency from and to the clients. We believe edge cloud computing is a good candidate for the following reasons: *1*) it has enough computation power as the servers in edge clouds usually have GPUs that are desktop-level or better, and *2*) it is located in the access network that is close to the users.

Based on this understanding, we propose a new architecture, EC+, that cleverly distributes the work among the central cloud and edge clouds. It has the following salient features: *1*) engaging edge clouds in managing view change updating and rendering to achieve low latency and high refreshing rate, *2*) engaging the central cloud in managing game state updating to support a large number of players and minimize the overhead required to maintain consistent game states, and *3*) handling user mobility and edge-cloud workload imbalance by performing dynamic gaming service migration to provide continued performance.

4.1 Flow of Gaming in EC+

Below, we discuss the game flow in the EC+ architecture step by step, which is also shown in Figure 2c:

- *User event forwarding on the client* ($S1$): VR game devices capture all user inputs and send them to a designated edge cloud. We will discuss how to choose and dynamically change the associated edge cloud for a user in § 5.
- *Local view updating and rendering on the edge cloud* ($S2$, $S2'$): When the edge cloud receives a user event, it passes the event to a "game event calculator", which performs two tasks in parallel: (1) calculating local view updates, and (2) determining whether a global game event is represented. After task (1) is completed, the local view update request is passed to a "renderer and encoder" which will then perform image rendering and encoding. After task (2) is completed and a global game event is needed, then the game event is further passed on to a central cloud.
- *Global game updating on the central cloud* ($S3$): The game server behaves similarly to that in traditional gaming. When the game server on the central cloud receives the game event, it calculates the updates that are caused by this event, updates user profiles accordingly, and then generates one or more update requests to all the players who are involved in this game event. It then sends these requests to the edge clouds that these players are currently connected to. Similar to traditional games, the server here can also take advantage of multicast in game update dissemination.
- *Game world change rendering on the edge cloud* ($S4$): The edge cloud performs all the rendering, including local view change rendering, game world change rendering, and background view refreshing rendering. The rendering performance is critical to the overall performance of EC+. We can leverage techniques such as the one proposed in [46] that involves a scalable parallel rendering framework to simultaneously render for multiple players who share a same game world, which can greatly reduce the overall rendering latency.

In summary, the proposed game flow has the following advantages. Firstly, bypassing the center cloud when dealing with view change events (in step $S2'$) can greatly shorten their response latencies, making it possible to have immediate local view updates. Secondly, by rendering frames on edge clouds (in step $S4$), we can harness their low latency and high bandwidth. Thirdly, the core network traffic can be largely reduced due to the adoption of edge clouds and the possibility of multicasting game updates to users.

4.2 Edge Cloud Migration

When we try to place a player's gaming service (including all the components resided in an edge cloud) onto edge clouds, selecting a suitable edge cloud becomes an important issue. After an initial edge cloud selection, we also need to consider the need of dynamically migrating the services to other edge clouds as the workloads and user locations change. Specifically, we note that service migration becomes necessary when the player moves around while playing games and/or the workload at each edge cloud changes over time.

In our framework, we consider the migration problem by partitioning continuous time into discrete time slots with equal length (say, 2 minutes). With the time partition, we can simultaneously make the optimal migration decisions, upon offline snapshots of the network/server states, for overall clients in the network. In respond to dynamic network/server states, any online solution, nevertheless, introduces the significant computation overhead in highly frequent decision making procedures. We have developed an efficient algorithm based on Markov Decision Process (MDP) to select and migrate a player's edge service, which we will discuss in detail in § 5. However, unlike many of the service migration solutions which assumes an ignorable service transition time, we acknowledge that it is impossible to migrate an edge service from one edge to another instantly given the size of a VR game world. Therefore, we propose a mechanism to ensure a new edge cloud is activated when a player connects to the new one.

To ensure a smooth transition between two edge clouds, the key is the ability to render frames correctly for a player connected to the destination edge cloud. A frame rendering process consists of a series of matrices operations on a game world matrix, a view/perspective matrix as well as a projection matrix, where the game world matrix represents a collection of 3D game models with the particular spatial relations, the view/perspective matrix transfers the relative positions of 3D game models to fit a particular view perspective, and the projection matrix converts 3D positions of game models into the homogeneous screen space. Service migration in EC+ mainly involves migrating a player's game world as the other matrices are ignorable in size and reproduced easily.

Here, we discuss the migration events of interest within a time slot starting from τ. We assume that a mobile user gets the service from an edge cloud e at τ.

- EC+ starts to make the migration decisions for all clients in the network at the time τ.
- At the time $\tau + \Delta_1$, EC+ finishes the computations of the decision making and determines to migrate the service of this mobile user to the edge cloud e'. e' will get a notification so that it subscribes to the multicast group of this game and start receiving all the game updates. The edge cloud e starts to send a snapshot of the game world at the time $\tau + \Delta_1$.
- At the time $\tau + \Delta_2$, e' successfully receives the game world snapshot (taken at $\tau + \Delta_1$) and start to merge the game updates received since $\tau + \Delta_1$.
- At the time $\tau + \Delta_3$, e' finishes the merging, and it now has the latest game world that is exactly same with the one kept in e. Meanwhile, the e' continues to receive the game updates and keeps the game world up-to-date.
- At the end of the time slot τ, this mobile user connects to e' and successfully gets the gaming service from this new edge cloud. The previous edge cloud e will release all the gaming resources if there is no other client connects to it.

With this mechanism, we can seamlessly complete service migration in EC+ without any service down-times as long as the time slot is larger than Δ_3 for all the migrations.

5 EDGE CLOUD SELECTION ON USER MOBILITY

We devise an algorithm to efficiently determine where to place and migrate an edge cloud service in the presence of dynamic network states and server workload states, and user mobility (initial edge cloud selection can also be generalized as a migration operation). In EC+, we model our placement/migration algorithm as a Markov decision process (MDP) [47] since a placement/migration decision is only affected by a current state and user mobility. We realize that several MDP-based selection approaches have been studied in the literature [11, 48–51], but we notice that VR-MMOGs impose new challenges, namely, the changing network status over time, the mutual impact among players, and the existence of an extra entity (central server) in the communication. In this section, we present our modified edge selection algorithm.

5.1 Modelling edge selection problem using MDP

In our selection algorithm, we consider a total of M edge clouds, and N access points through which mobile users connect to the Internet. As we discussed in §4, we partition continuous time into discrete time slots with equal length. At a time slot τ, a mobile user connects to an access point $n_\tau \in [1, N]$ and receives a gaming service from an edge cloud $m_\tau \in [1, M]$. We define this as a state S_τ=$m_\tau n_\tau$. A player may move and connect to a new access point at the end of a time slot. Due to the user mobility and changing workloads on the edges, we may need to migrate the gaming service to a proper edge to satisfy the user's QoS requirement. To achieve this, an action a_τ upon the state, migrates the service from the edge cloud m_τ to $m_{\tau+1}$. The action a_τ is represented by the location of a possible edge cloud $m_{\tau+1}$, thereby $a_\tau \in [1, M]$. The new edge cloud at $m_{\tau+1}$ is anticipated to have the minimal network cost by considering the player's any possible locations ($n_{\tau+1}$) in the next time slot. Note that while we are calculating MDP at the time τ, we assume that the migration happens at $\tau + 1$, as we described in the previous section. As a result, at the time slot $\tau+1$, the system may enter a transit state: $S_{\tau+1}$=$m_{\tau+1}n_{\tau+1}$, with the transition probability $p(S_\tau, a_\tau, S_{\tau+1})$. We assume that the transition probability is given as the known parameter of our algorithm as there are many studies on mobility prediction including [52–54] and our earlier work [55] that calculates the probabilities of user movements based upon the aggregated network-level statistics.

To determine the destination edge cloud $m_{\tau+1}$, a cost function $C(S_\tau, a_\tau, S_{\tau+1})$ is defined to measure the overall network transmission cost as well as the migration cost from a state S_τ to a state $S_{\tau+1}$, when we take an action a_τ^π. We detail this cost function in §5.2. Our objective is to find an optimal action (a_τ^π) for each user in each time slot that minimizes long-term cost. The long-term cost function is given by

$$V(S_0) = \sum_{\tau=0}^{\infty} \gamma^\tau \cdot \sum_{S_{\tau+1}=1}^{M\times N} p(S_\tau, a_\tau^\pi, S_{\tau+1}) \cdot C(S_\tau, a_\tau^\pi, S_{\tau+1}), \quad (1)$$

where $\gamma \in [0, 1)$ is a discount factor that controls the impact of future states on the long-term cost counted from the current state. We convert the cumulative sum of the long-term cost given by Equation 1 into a recursive definition:

$$V^*(S_\tau) = \min_{a_\tau} \{p(S_\tau, a_\tau, S_{\tau+1}) \cdot [C(S_\tau, a_\tau, S_{\tau+1}) + \gamma \cdot V^*(S_{\tau+1})]\}, \quad (2)$$

It is well known that the optimal action $a_\tau^\pi = m_{\tau+1} \in [1, M]$ for each state S_τ can be obtained by Bellman's value iteration [47] which iteratively update the equation 2 until the value of $V^*(S_\tau)$ is converged.

5.2 Game-specific Cost Function

While modeling the placement algorithm, we try to minimize the "cost" of actions to provide the best game experience. We believe that the cost function should take different features into consideration, including latency, bandwidth, *etc.*. Here, in order to propose a general framework that can satisfy all

kinds of VR-MMOGs, we do not mandate the application requirements on different features. Instead, we assume the game provider can get their cost function based on the studies in [56–59] and their policies.

In this section, we list a set of features that we have in mind. They come in two categories: transition cost and transmission cost. The transition cost is the cost incurred when we migrate an edge service. As we already have a mechanism to avoid application down-times, this cost is curtailed to the bandwidth cost (*i.e.*, the size of the game world). The transmission cost is the cost to the communication between an edge cloud and a player. This transmission cost can be further categorized into two sub-types: cost without mutual impact and cost with mutual impact. The cost without mutual impact is merely measured by network latency, bandwidth and server load, while the cost with mutual impact is additionally measured by a count of game world sharing (since a migration decision for one player can meanwhile affect the decision of other players who are sharing one game world). Importantly, we highlight the cost with mutual impact which intends to co-place multiple users in one edge cloud to facilitate game world sharing and to reduce the overall migration overhead.

5.3 Optimal Joint Migration Decisions

Many earlier MDP based migration approaches calculate an individual migration decision for each user, assuming a user's migration decision have little impact on others. However, when to consider the co-placement, the assumption fails to be hold. To this point, we have to consider all possible combinations of migration decisions at each step and find the optimal joint migration solution.

Assume the total number of the migration decisions we need to jointly consider at each step is K, which is also the number of users in the system, and denote each decision as $d_k, k \in [0, K)$. We then redefine a state as $S_{global}(t) = \{S_{d_1}(t), S_{d_2}(t), ..., S_{d_{K-1}}(t)\}$, and a joint action as $a_{global}(t) = \{a_{d_1}(t), a_{d_2}(t), ..., a_{d_{K-1}}(t)\}$. The new reward function is the sum of the reward function of each individual decision. Finally, we solve Eqn. 1 to compute the optimal joint migration action $a^*_{global}(t)$.

Though this approach provides a globally optimal migration decision, the time complexity to search for the optimal joint solution is much higher than that of treating each migration decision independently. Specifically, the time complexity of the latter is $O(M^3N^2)$, while the time complexity of the global solution is $O((M^3N^2)^K)$. This cost is prohibitively high, preventing us from finding the optimal joint solution in real-time.

5.4 Heuristic Joint Migration Decisions – Highest Migrate Probability First

When to calculate the optimal migration decision for each player, we hold an assumption that all other players remain connecting to their current edge clouds and therefore, the whole edge cloud serving conditions does not change. Only under this assumption, the optimal migration decision can keep being optimal. Yet, we fail to hold this assumption if we consider a collection of migration decisions for multiple players. To be close to the assumption, we can order the migration probability of all players and preferentially calculate the optimal migration decisions for the players with higher migration likelihoods. By doing so, after making a migration decision, we argue that the latter migration decisions are more likely to have players connected to the current edge cloud. To estimate the migration likelihoods, we use the overall cost function value subtracting the migration cost. We argue that this heuristic approach with the time complexity of $O(kM^3N^2)$ can minimize the global migration cost.

5.5 Runtime Optimization to Reduce Execution Time

We discover a few characteristics of the MDP calculation in this edge placement problem, which can be explored to optimize the runtime. The first characteristic we find is

$$\forall a_\tau, m_{\tau+1} : p(m_\tau n_\tau, a_\tau, m_{\tau+1} n_{\tau+1}) = 0, where\ a_\tau \neq m_{\tau+1}.$$

It indicates that a migration action is deterministic towards next state. Thus, we can simplify the state transition probability from $p(m_\tau n_\tau, a_\tau, m_{\tau+1} n_{\tau+1})$ to $p(m_\tau n_\tau, m_{\tau+1} n_\tau)$, also simplify the cost function from $C(m_\tau n_\tau, a_\tau, m_{\tau+1} n_{\tau+1})$ to $C(m_\tau n_\tau, m_{\tau+1} n_{\tau+1})$, Accordingly, we can reduce the space complexity of p and C from $O(M^3N^2)$ to $O(M^2N^2)$.

The second characteristic we discover is

$$\forall m_\tau, m'_\tau, m_{\tau+1}, m'_{\tau+1} : p(m_\tau n_\tau, m_{\tau+1} n_{\tau+1}) = p(m'_\tau n_\tau, m'_{\tau+1} n_{\tau+1}).$$

It demonstrates that the state transition probability merely relates to linked access points, but not to server placements . Thus, we can simplify the transition probability from $p(m_\tau n_\tau, m_{\tau+1} n_\tau)$ to $p(n_\tau, n_{\tau+1})$ and accordingly reduce the space complexity of p from $O(M^2N^2)$ to $O(N^2)$.

The third characteristic we discover is

$$\forall n_\tau, n'_\tau : C(m_\tau n_\tau, m_{\tau+1} n_{\tau+1}) = C(m_\tau n'_\tau, m_{\tau+1} n_{\tau+1}).$$

It implies that the cost function merely associates with the connected access point $n_{\tau+1}$ at the time slot $\tau + 1$. Thus, we can simplify the cost function from $C(m_\tau n_\tau, m_{\tau+1} n_{\tau+1})$ to $C(m_\tau, m_{\tau+1} n_{\tau+1})$ and accordingly reduce the space complexity of C from $O(M^2N^2)$ to $O(M^2N)$.

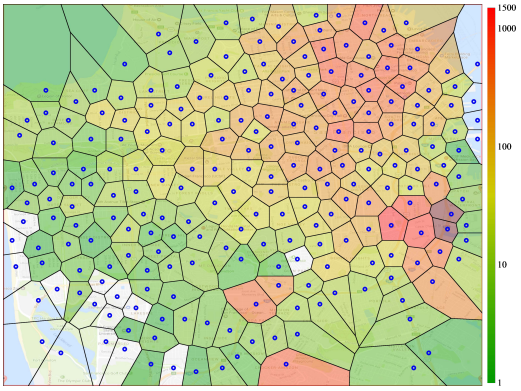


Figure 3: Access points with corresponding effective ranges and heat ($\int Conn(t)dt$).

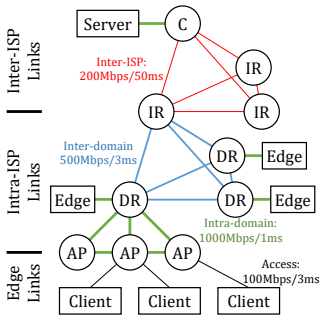
By jointly considering the above 3 propositions, we can simplify Equation 1 to

$$V(m_0 n_0) = \sum_{\tau=0}^{\infty} \gamma^{\tau} \cdot \sum_{n_{\tau+1}=1}^{N} p(n_\tau, n_{\tau+1}) \cdot C(m_\tau, m_{\tau+1}^{\pi} n_{\tau+1}), \quad (3)$$

where $m_{\tau+1}^{\pi}$ is our decision at the time slot τ which takes effect at the time slot $\tau+1$. Therefore, we can reduce the total space complexity from $O(M^3N^2)$ to $O(M^2N+N^2)$, and reduce the time complexity of each MDP iteration from $O(M^3N^2)$ to $O(M^2N^2)$. Since the computation of Equation 3 can be converted into the vector multiplication of $p(n_\tau, *)\cdot[C(m_\tau, a_\tau *)+\gamma V(a_\tau, *)]$, we can further reduce the execution time using parallel computing (multi-threading, and GPU). We evaluate the performance improvement of our proposed optimizations in §6.

5.6 Further optimizations

Besides the aforementioned optimizations of MDP applied in VR-MMOG migration, we also consider the following optimizations: *1*) Every player moves in a regular activity range and may never, if not impossible, link to a portion of remote access points. Accordingly, the probability table is sparse. We can therefore compress this table as well as the cost table to further reduce the space and time complexity. *2*) In many cases, players can only link to several nearby edge clouds due to the stringent latency and bandwidth requirements. We can identify and exclude remote edge clouds that fail to satisfy the requirements from the possible migration destinations. We can then remove the states associated with the migration destinations and eliminate the calculation of the cost and utility table with respect to the states. Since proposed edge placement algorithm is a framework that is generally applicable to all (VR-)MMOGs, we leave the application-specific optimizations as our future work.



Link	B/w (Mbps)	Latency (ms)
C↔IR	200	50
IR↔IR	200	50
IR↔DR	500	3
DR↔DR	500	3
DR↔AP	1000	1
AP↔AP	1000	3
C↔Server	1000	2
DR↔Edge	1000	1
AP↔Client	100	3

Figure 4: 3-layered network topology and corresponding bandwidth and latency.

6 EVALUATION

We conduct a detailed simulation-based evaluation of the proposed EC+ architecture as well as the MDP based service placement/migration algorithm. We summarize the simulation results in this section.

6.1 Comparison of EC+ with Other Gaming Architectures

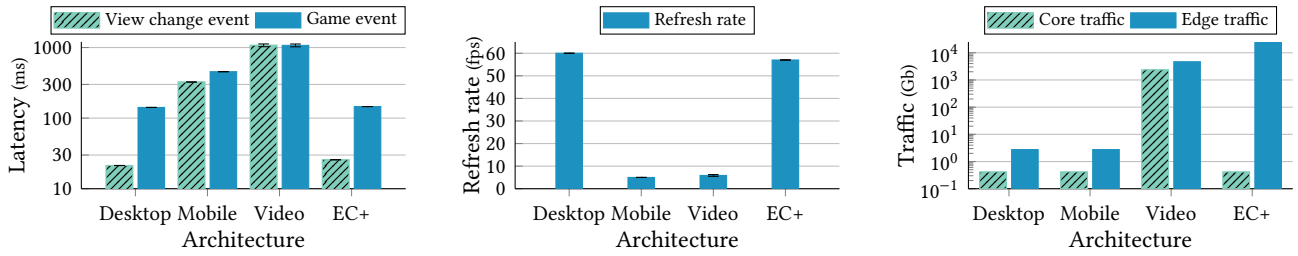
We use detailed simulation studies to compare our EC+ architecture with the traditional client-centric gaming architecture and the cloud-centric video streaming architecture.

6.1.1 Simulation Setup. We first present our simulation set up for the comparison study.

Network Topology: We use the San Francisco AP map developed in our earlier work [55] as the network topology. We estimate each AP's coverage using Voronoi cells (see Fig. 3). We further assign the APs to different domains to represent more realistic network topologies. Specifically, we build a 3-level hierarchical topology as shown in Fig. 4.

In simulations, games players connect to the Internet through APs, and edge clouds are assumed co-located with the domain routers. The central cloud is placed at C (see Fig. 4). We carefully choose bandwidth and latency parameters for different links in the network. For example, we assume in the intra-domain network, nodes are connected through gigabit switches with millisecond level latencies. The actual capacity for inter-ISP connections is usually much higher, but since the core network is multiplexed with other traffic, and the ISPs might not be directly linked to each other, we choose a bandwidth of 200Mbps (shared bandwidth) and a latency of 50ms (due to the number of hops between two ISPs). As such, we summarize the chosen bandwidth and latency values for different types of links in Fig. 4.

Player Location Trace: To model mobile game players, we use the San Francisco cab trace that we adopted in our earlier work [60]. The trace contains the locations of more than 500 cabs between 2008-05-17 and 2008-06-10. We observe



(a) Event latency (95% CI) **(b) Refresh rate (95% CI)** **(c) Aggregate network traffic**

Figure 5: Result of game simulation without mobility in different architectures: traditional gaming with powerful GPU (Desktop), traditional gaming with mobile devices (Mobile), videostream gaming (Video), and EC+.

noticeable daily mobility patterns in the trace. In our study, we pick the data sets on May 31, 2008, Saturday, as our trace. Since the simulated APs are mainly located in the central San Francisco area (see map in Fig. 3), we focus on the 66 cabs that traveled in that area. We assume these 66 players are playing a same MMOG game. Fig. 3 shows the "heat" of each AP. The hotness of AP a is calculated as $H(a) = \sum_{u \in U} t_u(a)$, where $t_u(a)$ is the total time a user u is associated with a.

Game trace: We consider a 1-hour synthetic game trace, taking the parameters from the study in [43]. Each player's user events arrive with Poisson distribution (λ between 9.5 and 15), a portion of which are randomly selected as game events. Each player's action per minute is between 30 and 300. In total, we have 18,686,459 user events (average: 78.642 UEs per second per user), and 287,567 (1.53%) game events (average: 72.618 actions per minute per user). Size of user events: Poisson distribution (λ=40); Size of updates (in traditional and edge): Poisson distribution (λ=130) [43]; Size of frames (in video stream and edge): (60Mbps / 60fps) = ~1Mb/f. The games are refreshing at 60fps.

Metrics: We use metrics including event latency (time between when the event happens and the clients see the related update) for both user and game events, core and edge network traffic, and frame rate.

6.1.2 Comparison Results with Stationary Players.

To study the fundamental difference among the architectures, we compare them assuming the players are stationary. Specifically, we take the locations of each user at 0:30, 1:30, ..., and 23:30 on 5/31 and create 24 different traces. We evaluate each architecture using these traces and report the results in Fig. 5.

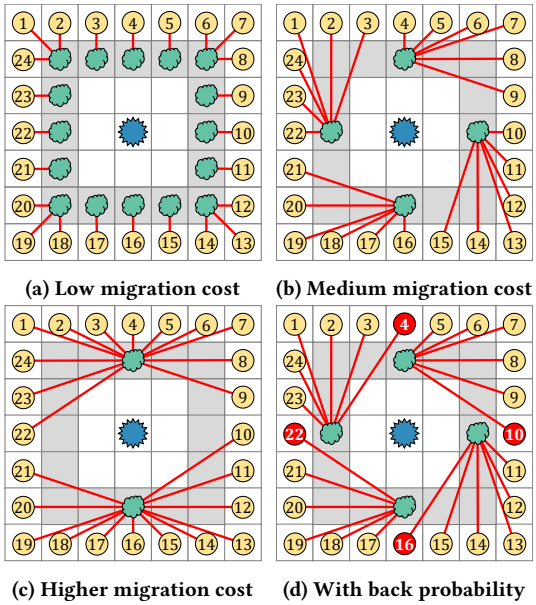
Traditional client-centric gaming fares well when users are equipped with desktops with powerful GPUs (with an average rendering latency pf 10ms). In this case, the view change response latency is rather low, ~20ms. The aggregate network traffic is also low (~400Mb) since only small game update packets are exchanged between the server and the client, with multicast support. Finally, it can obtain a refreshing rate of 60fps. When users with mobile devices (GPUs can render 5 frames per second) try to adopt traditional gaming, the rendering performance degrades significantly. The average view change rendering latency is 300ms, and the average refreshing rate is just around 5 fps.

Client-centric video stream gaming faces performance bottleneck in the core network, since it has to unicast frames to each client (which consumes more than 1Tb core network traffic). The frame drop rate is quite high, and therefore the actual frame rate at the client side is only around 7 fps. To alleviate the frequent frame drops, we adopt forward error correction (FEC) so that each frame contains all the events that arrive before the frame is rendered. However, even with this technique, the rendering latency is still high (>1s for both view changes, and game events) since the events can only be delivered by the next frame that is successfully delivered. We note that the performance can be even worse in the real world due to the multiplexing on the back haul links.

EC+ can provide event update latency (< 30ms for view change events and around 100ms for game events) and refresh rate (of 56fps) similar to traditional gaming (desktop), since the renderers in edge clouds are powerful and are close enough to the clients. With such a low update latency on the non-game events, our architecture can have good support on VR applications where users need immediate feedback for the non-game events (*e.g.*, look into another direction). While our solution does consume more traffic (>10Tb) in the edge network compared to the traditional gaming, we argue that it is feasible and stable since the ISPs usually have full control of the edge network to ensure QoS.

6.2 Validation of Edge Placement

To validate our edge selection solution, we conduct a set of small-scale simulations with synthetic topology and a small number of users.



(a) Low migration cost (b) Medium migration cost

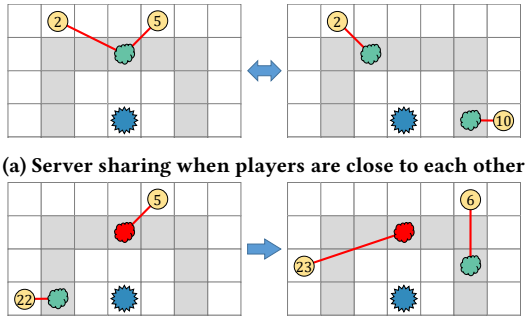
(c) Higher migration cost (d) With back probability

Figure 6: Edge placement decision in single user case.

6.2.1 Simulation Setup. We consider a 7×7 grid (shown in Fig. 6a), where each of the 24 outermost grid cells represents an AP and the users that are connected to this AP. We consider all the edge clouds are resided in the gray grid cells next to the outermost circle, and the central cloud is resided in the central grid. We assume a link's latency is proportional to the distance between the two endpoints, while its bandwidth is reversely proportional to the distance.

6.2.2 Validation for single player scenarios. We first validate our algorithm with a single player and varying the migration cost. We note that the migration cost varies from game to game, dependent on the game world size.

We first consider a simple mobility pattern where the player moves around clock-wise from one grid cell to the next (along with the outermost circle). Fig. 6a shows that when the migration cost is small, the placement algorithm always tries to place the service on the nearest edge cloud whenever the play moves. In this example, the player changes the location 24 times, and the corresponding service changes the location 16 times. When the migration cost increases, the algorithm decides to migrate less frequently (see Fig. 6b). For example, when the migration cost is around twice the current transmission cost – in this case the transmission latency is doubling the frame size in a time slot – the service location changes four times when the player changes location 24 times. When we further increase the migration cost, there are only two service locations for a total of 24 location changes for the player (see Fig. 6c). Finally, when the migration cost becomes too large (greater than four times of the current



(a) Server sharing when players are close to each other

(b) Reuse of existing edge (red) to reduce migration cost even when the original client (at 6) is going to move away

Figure 7: Mutual impact in multiplayer scenario.

transmission cost), the algorithm does not migrate at all (not shown in the figure).

We next consider a different player mobility pattern: at each time slot, the player moves to the next grid cell in the clockwise direction with a 60% probability, moves to the next grid cell in the counter clockwise direction with a 10% probability, and stay in his/her current cell with 30% probability. This new mobility pattern leads to different migration decisions (shown in Fig. 6d). Here, we use the same migration cost as in Fig. 6b. The results show that with the probability of the player moving backward, the placement algorithm becomes more conservative. It still has 4 different service locations, but the location change occurs one-time slot later than that in Fig. 6b.

In the considered single player scenarios, the algorithm outcomes match our expectation. We thus validate its correctness in the single player case.

6.2.3 Validation for Multi-Player Scenarios. Next, we validate our algorithm when we consider two players in the same 7×7 grid topology.

Here we vary the distance between the two players. Fig. 7a shows that, when the two players are close to each other (when they are in cells 2 and 5 respectively), our algorithm decides to place them on the same edge cloud to take advantage of shared game world. In Fig. 7b, when considering where to migrate to, our algorithm tries to migrate a player to an edge cloud that is hosting other players to ruse game worlds and reduce the migration cost. In this example, even though the original player (in cell 6) is about to leave the red edge cloud, our algorithm still migrates the player in cell 23 to the red edge cloud.

6.2.4 Optimal vs. Heuristic Placement with Multiple Players. Next, we compare the optimal placement solution *vs.* the heuristic placement solution when we have two players in the 7×7 grid topology. In each time slot, the player moves to the next grid cell clockwise, stay in the same grid

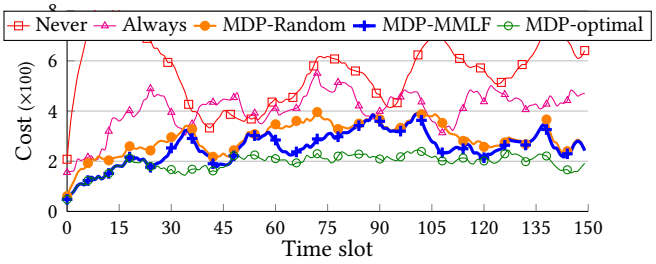


Figure 8: Result of service migration with different strategies.

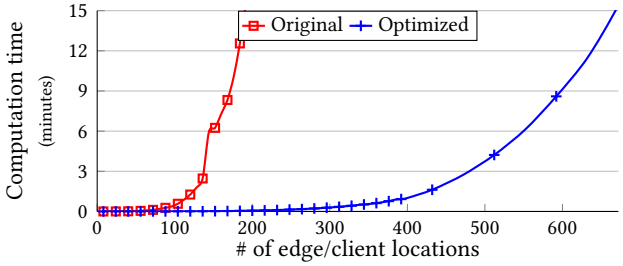


Figure 9: Result computation time.

cell, and moves to the next grid cell counter clockwise, each with 1/3 probability respectively. Fig. 8 reports the resulting migration cost of the following schemes: (1) MDP-optimal, the global optimal placement, (2) MDP-MMLF, the heuristic MDP placement scheme in which we place the user with the maximum migration likelihood first, (3) MDP-random, the heuristic MDP placement scheme in which we randomly sort the players, (4) Always, an always-migrate scheme, and (5) Never, a never-migrate scheme.

Among these five schemes, MDP-optimal gives the best performance. It also consumes the most memory and CPU resources. Given the 24×24 client locations and 16×16 edge cloud locations, it takes more than 1 minute to compute the migration decisions for two players. When we have a large number of players, the computation cost will be prohibitively costly. We also observe that all three MDP based solutions give much better performance compared to naive always-migrate or never-migrate schemes. Finally, we find that our proposed MDP-MMLF performs closer to the optimal solution than MDP-random.

6.2.5 Evaluating the Runtime Overhead. Finally, we measure the run-time overhead of the three versions of MDP-MMLF implementation: (1) original, (2) optimized. Our hardware platform consists of an Intel Core i7-4790 CPU with the clock rate of 3.60GHz [61], running Ubuntu 14.04.

In our experiments, we consider 1 user and vary the number of edge clouds and client locations. Figure 9 shows the computation times for all three implementations. The results show that the execution time of the original implementation increases fast with the number of edge/client locations (so does its memory consumption). The optimized implementation has a much lower execution time with the same edge/client numbers. We believe this version can be used for scenarios with many more users since in the real network, there are usually ~10 possible edge clouds that a user can use at any time.

7 CONCLUSION

In this paper, we highlight the main challenges of VR-MMOGs and propose two solutions. The EC+ architecture seamlessly distributes the required processing across user devices, edge clouds, and the center cloud to achieve ultra low-latency responses, frequent refreshing, and a large number of concurrent players. To complement our architecture, our game service placement algorithm maximizes gaming performance for all players by dynamically placing their services on those edge clouds that lead to the best performance. Finally, we have conducted detailed simulation studies to evaluate our edge-cloud assisted gaming architecture and dynamic service placement algorithm. Our results indicate that the proposed approach serves as a viable solution for supporting VR-MMOGs.

REFERENCES

[1] S. Choy *et al.*, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *NetGames*, 2012.
[2] NVIDIA, "Geforce NOW," https://www.nvidia.co.uk/shield/games/.
[3] SONY, "PlayStation NOW," https://www.playstation.com/en-us/explore/playstationnow/.
[4] C.-Y. Huang *et al.*, "GamingAnywhere: The First Open Source Cloud Gaming System," *TOMM*, p. 10, 2014.
[5] R. Shea *et al.*, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.
[6] J. Carmack, "John Carmack's Delivers Some Home Truths on Latency," http://oculusrift-blog.com/john-carmacks-message-of-latency.
[7] Nvidia, "Geforce NOW System Requirements," https://shield.nvidia.com/support/geforce-now/system-requirements/2.
[8] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, 2009.
[9] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *MCC*, 2012.
[10] T. Taleb *et al.*, "Follow me cloud: interworking federated clouds and distributed mobile networks," *IEEE Network*, pp. 12–19, 2013.
[11] W. Zhang *et al.*, "Segue: Quality of service aware edge cloud service migration," in *CloudCom*, 2016.
[12] HTC, "Vive's immersive room-scale technology," https://www.vive.com/us/.
[13] Google, "Google Cardboard brings immersive experiences to everyone in a simple and affordable way." https://vr.google.com/cardboard/get-cardboard/.
[14] S. M. LaValle, *VIRTUAL REALITY*. Cambridge University Press, 2015.
[15] Lookingglass, "Virtual reality mmorpg," https://lookingglass.services/virtual-reality-mmorpg/.
[16] J. Lee, "7 Games You Can Mod to Add VR Support Right Now," http://www.makeuseof.com/tag/mods-provide-vr-support/.
[17] Kalydo, "Kalydo," https://en.wikipedia.org/wiki/Kalydo/.

[18] Spawnapp, "Spawnapp," http://spawnapp.com//.
[19] NVIDIA, "Nvidia grid," http://www.nvidia.com/object/nvidia-grid.html.
[20] Y. Zhang *et al.*, "A cloud gaming system based on user-level virtualization and its resource scheduling," *TPDS*, pp. 1239–1252, 2016.
[21] A. Banitalebi-Dehkordi *et al.*, "Compression of high dynamic range video using the hevc and h. 264/avc standards," in *QShine*, 2014.
[22] K. Lee *et al.*, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," in *MobiSys*, 2015.
[23] J. R. Lange *et al.*, "Experiences with client-based speculative remote display." in *USENIX Annual Technical Conference*, 2008, pp. 419–432.
[24] F. M. Bartucca *et al.*, "Detecting network instability," Jul. 12 2005, uS Patent 6,918,067.
[25] P. Quax *et al.*, "Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game," in *NetGames*, 2004.
[26] K. Hong *et al.*, "Mobile fog: A programming model for large-scale applications on the internet of things," in *MCC*, 2013.
[27] A. R. Elias *et al.*, "Where's the bear?: Automating wildlife image processing using iot and edge cloud systems," in *IoTDI*, 2017.
[28] A. Bharambe *et al.*, "Colyseus: A Distributed Architecture for Online Multiplayer Games," in *NSDI*, 2006.
[29] N. E. Baughman *et al.*, "Cheat-proof playout for centralized and distributed online games," in *INFOCOM*, 2001.
[30] S. Choy *et al.*, "A hybrid edge-cloud architecture for reducing on-demand gaming latency," *Multimedia Systems*, pp. 503–519, 2014.
[31] Y. Lin *et al.*, "Cloudfog: Leveraging fog to extend cloud gaming for thin-client mmog with high quality of experience," *TPDS*, pp. 431–445, 2016.
[32] H.-J. Hong *et al.*, "Qoe-aware virtual machine placement for cloud games," in *NetGames*, 2013.
[33] M. Steiner *et al.*, "Network-aware service placement in a distributed cloud environment," *SIGCOMM CCR*, pp. 73–74, 2012.
[34] J. T. Piao *et al.*, "A network-aware virtual machine placement and migration approach in cloud computing," in *GCC*, 2010.
[35] S.-H. Lim *et al.*, "Migration, assignment, and scheduling of jobs in virtualized environment," *Migration*, vol. 40, p. 45, 2011.
[36] C. Ghribi *et al.*, "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms," in *CCGrid*, 2013.
[37] F. Douglis *et al.*, "Transparent process migration: Design alternatives and the sprite implementation," *Software: Practice and Experience*, vol. 21, no. 8, pp. 757–785, 1991.
[38] C. Clark *et al.*, "Live migration of virtual machines," in *NSDI*, 2005.
[39] W. Voorsluys *et al.*, "Cost of virtual machine live migration in clouds: A performance evaluation," in *CLOUD*, 2009.
[40] H. Jin *et al.*, "Live virtual machine migration with adaptive, memory compression," in *CLUSTER*, 2009.
[41] K. Ha *et al.*, "Adaptive vm handoff across cloudlets," Technical Report CMU-CS-15-113, CMU School of Computer Science, Tech. Rep., 2015.
[42] J. Chen *et al.*, "G-COPSS: A Content Centric Communication Infrastructure for Gaming," in *ICDCS*, 2012.
[43] M. Claypool *et al.*, "Thin to win?: network performance analysis of the onlive thin client game system," in *NetGames*, 2012.
[44] ——, "Latency and player actions in online games," *Communications of the ACM*, pp. 40–45, 2006.
[45] Wikipedia, "Actions per minute," https://en.wikipedia.org/wiki/Actions_per_minute.
[46] C. F. Perez-Monte *et al.*, "Modelling frame losses in a parallel alternate frame rendering system with a computational best-effort scheme," *Computers & Graphics*, vol. 60, pp. 76–82, 2016.
[47] D. L. Poole *et al.*, *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.
[48] Q. Zhang *et al.*, "Dynamic service placement in geographically distributed clouds," *JSAC*, pp. 762–772, 2013.
[49] S. Wang *et al.*, "Dynamic service placement for mobile micro-clouds with predicted future costs," *TPDS*, 2016.
[50] A. Ksentini *et al.*, "A markov decision process-based service migration procedure for follow me cloud," in *ICC*, 2014.
[51] R. Urgaonkar *et al.*, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, pp. 205–228, 2015.
[52] P. Deshpande *et al.*, "Predictive methods for improved vehicular wifi access," in *MobiSys*, 2009.
[53] A. J. Nicholson *et al.*, "Breadcrumbs: forecasting mobile connectivity," in *MobiCom*, 2008.
[54] V. A. Siris *et al.*, "Enhancing mobile data offloading with mobility prediction and prefetching," *SIGMOBILE CCR*, pp. 22–29, 2013.
[55] F. Zhang *et al.*, "Edgebuffer: Caching and prefetching content at the edge in the mobilityfirst future internet architecture," in *WoWMoM*, 2015.
[56] H. Tian *et al.*, "On achieving cost-effective adaptive cloud gaming in geo-distributed data centers," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 2064–2077, 2015.
[57] S. Grizan *et al.*, "djay: enabling high-density multi-tenancy for cloud gaming servers with dynamic cost-benefit gpu load balancing," in *SoCC*, 2015.
[58] D. Wu *et al.*, "icloudaccess: Cost-effective streaming of video games from the cloud with low latency," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1405–1416, 2014.
[59] T. N. B. Duong *et al.*, "Qos-aware revenue-cost optimization for latency-sensitive services in iaas clouds," in *DS-RT*, 2012.
[60] M. Piorkowski *et al.*, "CRAWDAD dataset epfl/mobility (v. 2009-02-24)," Downloaded from http://crawdad.org/epfl/mobility/20090224, Feb. 2009.
[61] D. Raychaudhuri *et al.*, "Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols," in *WCNC*, 2005.