Peer-Assisted Texture Streaming in Metaverses

Ke Liang, Roger Zimmermann, Wei Tsang Ooi
School of Computing
National University of Singapore
Singapore, 117417
{liangke, rogerz, ooiwt}@comp.nus.edu.sg

ABSTRACT

User-extensible metaverses need an effective way to disseminate massive and dynamic 3D contents (*i.e.*, meshes, textures, animations, etc.) to end users, and at the same time maintain a low consumption of server bandwidth. Peer-to-peer (or peer-assisted) technologies have been widely considered as a desirable complementary solution to efficaciously offload servers in large-scale media streaming applications. However, due to both the bandwidth constraints of heterogeneous users and unpredictable access patterns of latency-sensitive 3D contents, it is very challenging to cut the server bandwidth cost in metaverses. In this paper, we propose a peer-assisted texture streaming architecture to minimize the server bandwidth consumption without degrading the end-user satisfaction. We propose a game-theoretic peer selection strategy which achieves a good trade-off between performance and complexity. Our algorithm is light-weight, and can efficiently utilize the bandwidth of users in a fully decentralized manner by enabling each peer (*i.e.*, user) to quickly select its content providers which can satisfy the requests of the peer within the latency constraint of the content. We evaluate our algorithm through an extensive comparison study based on simulations using realistic data (*i.e.*, avatar mobility traces and textures) collected from Second Life. The simulation results show that the proposed algorithm can effectively reduce the server bandwidth consumption without degrading the user experience.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; C.4 [**Performance of Systems**]: Performance attributes

General Terms

Algorithms, Design, Theory

Keywords

Peer-assisted streaming, game-theoretic algorithm

*Area Chairs: Surender Chandra.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MM'11, November 28–December 1, 2011, Scottsdale, Arizona, USA.
Copyright 2011 ACM 978-1-4503-0616-4/11/11 ...$10.00.

1. INTRODUCTION

Recent years have witnessed a significant growth in *metaverses*, defined as online 3D immersive networked virtual worlds, where each user, represented by an *avatar*, can socialize with other users and interact with user-generated 3D contents. Unlike massively multiplayer online games (MMOGs), where users install the whole static 3D contents in advance via DVDs or patches downloaded offline from the Internet, the metaverses need to stream interactive 3D contents to users live over the Internet, according to their visibility or interests. Specifically, when avatars move around in a metaverse, their immediate environment will be downloaded dynamically from the metaverse server to the corresponding users. Considering the fact that a significant portion (over 50%) of the server bandwidth is consumed by transmission of the *textures*[1] of 3D contents in metaverses [22], we focus on texture streaming in this study and refer to 3D contents as textures throughout the paper.

Most commercial metaverses (*e.g.*, Second Life [27] and ActiveWorlds [2]) are currently deployed using a client/server (C/S) scheme, where centralized servers are used to maintain the states of metaverses and online users, and to distribute 3D contents to users. Throughout the paper, we use the term content, texture, and 3D content interchangeably. Since users do not store the entire virtual world, but download the contents from dedicated servers when needed, the server bandwidth costs can be huge when the amount of contents and the number of online users increases. For example, according to the economic report from Second Life [28] in the third quarter of 2010, the user hours (*i.e.*, the sum of the combined length of all user sessions) per month is roughly 35 million hours. From Tables 2 and 3 in Section 5, we can observe that the average amount of texture data sent from the servers per user hour is 263 MB. Therefore, roughly 9 petabytes (PB) of texture data will be delivered from the servers to users monthly. As a consequence, one of the most important and formidable challenges facing the metaverse providers is to reduce the server bandwidth consumption.

A natural way to offload servers is to utilize bandwidth and caches of online users using peer-to-peer (P2P) technologies, *i.e.*, enable online users to serve one another. We use the term peer, avatar and user interchangeably thereafter. With the assistance of *peers*, the server bandwidth cost can be reduced, and the scalability and affordability of the system can be improved. More specifically, a peer can download the required contents from other peers via sending

[1]A texture is simply a bitmap image that is used to provide surface coloring for a 3D model.

requests to them, which may send back the corresponding contents if they have available bandwidth. In order not to degrade the user satisfaction of metaverses, all the required contents need to be downloaded within a certain *latency constraint* measured in seconds. If a peer fails to download the required textures from other peers within the latency constraint, it will send the requests directly to dedicated servers, thus consume bandwidth on the server side. Since the server is not replaced but complemented by the peers, we refer to such systems as peer-assisted systems. To minimize the server bandwidth cost, peers need to select appropriate peers as their content *providers*, respecting the available bandwidth of peers and latency constraints of the contents.

Peer selection strategies, which aim to reduce the server bandwidth consumption, have been extensively studied in peer-assisted media streaming systems, including live [17,19] and video-on-demand (VoD) [16,30] P2P streaming. Unfortunately, current peer selection strategies for live or VoD P2P streaming systems cannot be easily applied to 3D content delivery, due to the unpredictable and non-linear request patterns of 3D content.

More specifically, in live P2P streaming systems, every peer has the same request pattern and it does not usually request the contents that precede the playback timestamp (assuming it has been stored in its cache if seeking is allowed). In VoD P2P streaming systems, every peer can (mostly) predict the contents it needs in the near future. Therefore, the media contents in both live and VoD streaming systems are viewed as one-dimensional (*i.e.*, time) data files. In texture streaming systems, however, the contents are viewed as data files that are stored in a multi-dimensional space (*i.e.*, locations). Peers with different download and upload capacities download the contents on demand according to their unpredictable behaviors *per se*, such as walking, running and flying in metaverses. Moreover, the number of potential content providers for each peer in live or VoD P2P streaming systems is large, and thus the sets of content providers of peers are less overlapping. However, in texture streaming systems, the potential content providers of a peer (*e.g.*, an avatar) are mostly its neighbors in the virtual world. Hence the number of potential content providers is small and the sets of content providers of peers are highly overlapping. This makes the design of peer-assisted texture streaming much more challenging compared with live or VoD P2P streaming systems. Prior research on peer selection strategies in peer-assisted texture streaming systems has paid little attention to the issue of optimally utilizing the bandwidth of heterogeneous peers while respecting the latency constraints of textures.

In this paper, we design a decentralized peer selection strategy for peer-assisted texture streaming with heterogeneous peers. The proposed peer selection strategy has a rapid convergence rate, and it can minimize the server bandwidth consumption while respecting the latency constraints of textures. It is important to point out that we seek to reduce the server bandwidth consumption via optimally utilizing the available network capacities of peers, hence the focus of this paper is not on incentive mechanisms [18] that aim to increase the available network resources.

Note that a practical peer selection strategy should be performed in a decentralized manner. We believe the peer selection strategy in such peer-assisted texture streaming systems can be characterized in a game-theoretic setting, and thus inherit its decentralized and practical nature. Specifically, requests sent by peers can be viewed as players in a game, where each is associated with a weight which is the size of the requested texture. Each player in the game is selfish and will try to minimize her own cost. In other words, peers will send their requests to underloaded providers. We formulate the server bandwidth minimization problem as an asymmetric congestion game, and we show that its equivalent problem is to minimize the *potential* of the game.

The rest of the paper is organized as follows. We model the peer-assisted texture streaming system and formulate the server bandwidth minimization problem in Section 2. We describe our game-theoretic peer selection algorithm in Section 3. The convergence time of the proposed algorithm is analyzed in Section 4. Section 5 details the evaluation of the proposed peer selection algorithm. The simulation results show that the proposed peer-assisted system can achieve a much lower server bandwidth consumption in comparison with existing algorithms. Finally, we discuss related work and conclude the paper in Sections 6 and 7, respectively.

2. SYSTEM MODEL

The most popular metaverse today is Second Life [27], where the virtual world consists of non-overlapping and independent regions. Each region is managed by a server that maintains the states of all objects and avatars within the region. To simply the discussion, we restrict our focus to a single region in Second Life. Once a peer logs in, it contacts the server which ensures the persistency of the region by continually sending relevant updates of objects and avatars to the peer through the network. In addition, the server determines the relevant subset of data that is required by each peer according to its location and visibility.

Each peer can freely roam around within the region and interact with other peers and objects within its *area-of-interest* (AoI), which is defined as a circular space around the peer with a given radius. The peers in the AoI of a peer are called AoI neighbors. Each peer needs to download the textures within its AoI that are not stored in its *cache* as it is moving in the region. After receiving the demanded textures, it will store them in its cache, from where a view is rendered according to the spatial information (*e.g.*, location and orientation) of the objects that the textures apply to. The size of the cache is determined by the metaverse provider or peers themselves. We assume that the default cache size of each peer is 128 MB (default cache size of users in Second Life). A simple but effective cache replacement algorithm is used in our setting: the cached textures that are far way from the avatar will be replaced with the newly downloaded textures when the cache is full.

To cut the server bandwidth cost, every peer is allowed to download the required textures from other peers by sending requests to others. If a peer receives requests from other peers, it is regarded as a content *provider* of those peers, and we assume every peer can be a provider to others. These requests are processed by content providers in a FIFO (First In First Out) manner. To maintain a good user satisfaction, all the requested textures must be delivered within a certain time limit, defined as the *latency constraint* of textures. If a peer fails to obtain the requested textures from its content providers within the latency constraint (due to the upload constraints of its content providers), it will send the requests to the server, thus consuming server bandwidth.

2.1 Notations and Definitions

Let P and T denote the set of online peers and the set of textures in a region, respectively. The size of texture $i \in T$, denoted by w_i, is measured in units of kilobytes (*i.e.*, KB). Let r_i^j denote the request sent by peer j for texture i, and the *weight* of the request r_i^j be the size of the requested texture (*i.e.*, w_i). The latency constraint for each texture is denoted by d, measured in seconds. Without loss of generality, we assume online peers are heterogeneous and have different upload capacities. Let u_i denote the upload capacity of peer i, measured in kilobytes per second, *i.e.*, KB/s. It is critical to note that the upload capacity is assumed to be the only bottleneck in the system, since the bottleneck caused by the constrained download capacities of peers can be eliminated by reducing the number of requests accordingly.

Let $R_i = \cup_{j \in P}\{r_i^j\}$ be the set of requests sent by all peers for texture i. Since every texture is unique, we have $R_i \cap R_j = \emptyset$ for any two textures i and j. In order to avoid receiving duplicate textures which would significantly increase bandwidth overhead and server bandwidth cost, every request is sent to only one of the peers. In other words, every request has only one provider. Similarly, let $S_i \subseteq P$ be the set of peers that have texture i in their caches. The information of S_i can be piggybacked into the updates sent by the server through the network. The *load* of peer i, denoted by l_i, is defined as the total weights of requests sent to peer i. The *height* of request r_k^i (sent by peer i) at peer j, denoted by $h_k^{i,j}$ (since every request is unique and has only one provider), is defined as the load of peer j directly after request r_k^i arrives at peer j. The notations that will be used throughout this paper are summarized in Table 1.

2.2 Problem Formulation

Since the amount of data that can be delivered by peer i within the latency constraint is $d \cdot u_i$, the server will disseminate the rest of the amount of data, if any, to the corresponding peers that sent their requests to peer i. Therefore, the server will send $\max\{l_i - d \cdot u_i, 0\}$ of data due to the inadequate upload capacity of peer i. Our goal is to minimize the bandwidth consumption at the server while respecting the upload capacities of heterogeneous peers, which can be formally stated as follows:

$$(\mathbf{F1}) \min \quad \sum_{i \in P} \max\{\sum_{R_k \neq \emptyset} \sum_{i \in S_k} x_k^{j,i} w_k - d \cdot u_i, 0\} \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in S_k} x_k^{j,i} = 1, \ \forall r_k^j \in R_k \tag{2}$$

$$x_k^{j,i} \in \{0,1\}, \ \forall r_k^j \in R_k. \tag{3}$$

Term	Definition
P	Set of online peers.
T	Set of textures.
u_i	Upload capacity of peer $i \in P$.
w_i	Size of texture $i \in T$.
r_k^i	Request sent by peer i for texture k.
R_k	Set of requests for texture k. $R_k = \cup_{i \in P}\{r_k^i\}$
S_k	Set of peers that have texture k in their caches.
l_i	Load of peer $i \in P$.
$h_k^{i,j}$	Height of request r_k^i at its provider, peer j.
d	Latency constraint of textures.

Table 1: List of notations and their definitions.

As shown in Eq. 3, $x_k^{j,i}$ is an indicator variable indicating whether the request r_k^j is sent to peer i. Specifically, if r_k^j is sent to peer i, $x_k^{j,i} = 1$; otherwise $x_k^{j,i} = 0$. Eq. 2 shows that every request has only one provider. The load of peer i is

$$l_i = \sum_{R_k \neq \emptyset} \sum_{i \in S_k} x_k^{j,i} w_k. \tag{4}$$

Note that the equivalent problem of minimizing the bandwidth consumption at the server is to maximize the total amount of data that is transmitted by peers within the latency constraint of textures (*i.e.*, d seconds), which can be formulated as follows:

$$(\mathbf{F2}) \ \max \quad \sum_{j \in P} \sum_{k \in T} w_k x_k^{j,i} \tag{5}$$

$$\text{s.t.} \quad \sum_{i \in S_k} \sum_{k \in T} w_k x_k^{j,i} \leq d \cdot u_i, \ \forall r_k^j \in R_k, \tag{6}$$

$$x_k^{j,i} \in \{0,1\}, \ \forall r_k^j \in R_k. \tag{7}$$

It can be observed that the alternative formulation (*i.e.*, **F2**) is a multidimensional knapsack problem [24], which is NP-complete [13] in general. Many approaches have been proposed to solve it, such as LP-relaxation [25], the primal-dual method [6] and dynamic programming [26].

We would like to point out that centralized approaches are not desirable for the design of peer selection strategies in peer-assisted texture streaming systems since a global connectivity information and the request information is required to obtain a feasible solution. Therefore, the computation to solve **F1** (Eq. 1) or **F2** (Eq. 5) will be invoked upon churn (*i.e.*, peer's leaving or joining) and avatar mobility that significantly changes the set of requests. As the number of peers increases, the overhead of communication and processing is not tolerable in practice.

Although there exist decentralized [23] and online [6] [9] algorithms that compute approximate solutions within polylogarithmic communication rounds, they are based on the assumption that each peer has the same capacity and is aware of global information, *i.e.*, it knows about loads and upload capacities of peers, and the connectivity information between any two peers. However, we consider the scenario where peers have different upload capacities and each requested texture needs to be downloaded within the latency constraint. As a result, existing centralized or decentralized algorithms which aiming to solve Eq. 5 in the formulation **F2** are not suitable for peer-assisted texture streaming. By combining Eq. 1 and Eq. 4, **F1** can be simplified to

$$\min \quad \sum_{i \in P} \max\{l_i - d \cdot u_i, 0\}, \tag{8}$$

where $\max\{l_i - d \cdot u_i, 0\}$ is the amount of bandwidth consumed by the server due to the inadequate upload capacity of peer i. Therefore, we transform **F1** into an asymmetric load balancing problem: how to distribute the loads on *overloaded* peers ($l_i > d \cdot u_i$) to *underloaded* peers ($l_i < d \cdot u_i$).

The objective of this paper is to design a light-weight peer selection strategy that can be implemented in a decentralized and online fashion. To this end, we model the peer selection strategy as a *congestion game* and thus inherit its practical and decentralized nature, which has been successfully used to model load balancing problems in P2P networks as well as many other real world applications due to its conceptual simplicity.

3. SERVER BANDWIDTH MINIMIZATION

We first describe the concept of congestion games and Nash equilibrium. Then we present a peer selection strategy based on an asymmetric weighted congestion game to minimize server bandwidth cost in a decentralized fashion.

3.1 Congestion Games

The classical congestion games have been investigated for many years. A congestion game, denoted by G, can be defined as a tuple $(N, M, (A_i)_{i\in N}, (f_e)_{e\in M})$, where $N = \{1, ..., n\}$ denotes the set of *players* and $M = \{1, ..., m\}$ denotes a set of *facilities*. Each player $i \in N$ is assigned a finite set of strategies A_i and a cost function f_e is associated with facility $e \in M$.

To play the game, each player i selects a strategy $a_i \subseteq A_i$, where $A_i \subseteq M$ is the strategy set of player i. The strategy profile, denoted by $\mathbf{a} = (a_i)_{i\in N}$, is defined as a vector of strategies selected by all the players. Similarly we use the notation $\mathcal{A} = \times_{i\in N} A_i$ to denote the set of all possible strategy profiles. A congestion game is *symmetric* if all the players have the same strategy set, *i.e.*, $\forall i, j \in N, A_i = A_j$; otherwise it is *asymmetric*. A congestion game is *weighted* if each player i is specified a weight w_i. The cost of player i for the strategy profile $\mathbf{a}$ is given by $c_i(\mathbf{a}) = f_{a_i}(\mathbf{a}, w_i)$.

The goal of each player in congestion games is to minimize her own cost without trying to optimize the global situation. That is, all players will try to lower their own cost by changing their strategies individually. A *pure Nash equilibrium* is defined as a steady state in which no players have an incentive to change their strategies.

Definition 1 (Pure Nash equilibrium [21]). *A strategy profile $\mathbf{a} \in \mathcal{A}$ is said to be a pure Nash equilibrium of G if for all players $i \in N$ and each alternate strategy $a_i' \in A_i$,*

$$c_i(a_i, \mathbf{a}_{-i}) \leq c_i(a_i', \mathbf{a}_{-i}), \tag{9}$$

where $\mathbf{a}_{-i} = (a_j)_{j\in N\setminus\{i\}}$ denotes the list of strategies of the strategy profile $\mathbf{a}$ for all players except i.

Congestion games have a fundamental property that a pure Nash equilibrium always exists [11]. To analyze convergence properties of congestion games (*i.e.*, the time from any state to a pure Nash equilibrium), we introduce the definition of potential function for congestion games.

Definition 2 (Potential function [10]). *A function $\Phi : \mathcal{A} \to R$ is a potential for game G if $\forall \mathbf{a} \in \mathcal{A}$, $\forall a_i, a_j \in A_i$,*

$$c_i(a_i, \mathbf{a}_{-i}) \leq c_i(a_j, \mathbf{a}_{-i}) \Rightarrow \Phi(a_i, \mathbf{a}_{-i}) \leq \Phi(a_j, \mathbf{a}_{-i}). \tag{10}$$

Therefore, $\mathbf{a}^*$ is a Nash equilibrium if and only if

$$\mathbf{a}^* = \arg\min_{\mathbf{a}\in\mathcal{A}} \Phi(\mathbf{a}). \tag{11}$$

An ϵ-Nash equilibrium is an approximate Nash equilibrium, which is defined as a state in which no player can reduce her cost by a multiplicative factor of less than $1 - \epsilon$ by changing her strategy.

3.2 Minimizing Server Bandwidth Cost

We show that the server bandwidth minimization problem can be formulated as an asymmetric weighted congestion game, denoted by G, which is defined as

$$G = (N, M, (A_i)_{i\in N}, (f_e)_{e\in M}), \tag{12}$$

where $N = \cup_{k\in W} R_k$ and $M = \cup_{k\in W} S_k$ correspond to the set of requests and peers (*i.e.*, providers) in our scenario, respectively. Each request $i \in N$ has a weight w_i (*i.e.*, the size of the requested texture), and each peer $i \in M$ has an upload capacity u_i. The strategy set of request i, denoted by A_i, is the set of peers M_i that have stored the requested texture in their caches. Since every request is sent to only one of the peers at a time, G is a *singleton* congestion game, *i.e.*, $\forall i \in N$, $a_i \in A_i$.

The strategy profile of G, denoted by $\mathbf{a} = (a_i)_{i\in N}$ corresponds to the peer selection profile, and $\mathcal{A} = \times_{i\in N} A_i$ corresponds to the set of all possible peer selection profiles. The cost of request i given a peer selection profile $\mathbf{a}$ is defined as

$$c_i(\mathbf{a}) = \min\{\max\{h_i - d \cdot u_j, 0\}, w_i\}, \tag{13}$$

where $j = a_i$ is the peer selected by request i and h_i is the *height* of request i at peer j. Clearly, for each peer $j \in M$, the sum of the cost of requests that select peer j is

$$\sum_{a_i=j} c_i(\mathbf{a}) = \sum_{a_i=j} \min\{\max\{h_i - d \cdot u_j, 0\}, w_i\} \tag{14}$$

$$= \max\{l_j - d \cdot u_j, 0\}, \quad \forall a_i \in \mathbf{a}. \tag{15}$$

Recall that $\max\{l_j - d \cdot u_j, 0\}$ is the amount of data that the server will disseminate to the corresponding peers due to the upload capacity limitation of peer j. We define the potential function of G is as the sum of the cost of all requests:

$$\Phi(\mathbf{a}) = \sum_{j\in M} \sum_{a_i=j} c_i(\mathbf{a}) = \sum_{j\in M} \max\{l_j - d \cdot u_j, 0\}. \tag{16}$$

Therefore, the problem of minimizing the server bandwidth cost can be solved by minimizing the potential function of the corresponding congestion game. The peer selection profile, *i.e.*, $\mathbf{a}$, has a significant impact on the server bandwidth cost, which is minimized if $\mathbf{a}$ is a pure Nash equilibrium (*i.e.*, $\mathbf{a} = \mathbf{a}^*$ as shown in Eq. 11).

3.3 Peer Selection Strategy

Based on the established connection between the congestion game G (see Eq. 12) and the server bandwidth cost minimization problem, an optimal peer selection strategy can be thought of as an optimal strategy profile, *i.e.*, a pure Nash equilibrium of G. It has been shown that finding a pure Nash equilibrium in a congestion game is PLS-complete [11], hence the number of changes of the strategy profile required from one state to any pure Nash equilibrium is exponentially large. Therefore, our goal is to quickly reach an *approximate* Nash equilibrium from any state by enabling each peer to change its strategies for its requests according to a *bounded jump rule* (see lines 11 and 14 in Algorithm 1), which will be discussed later in this section.

It is worth to point out that the offline peer selection strategies are undesirable in practice due to both the additional delay they incur to find appropriate content providers for all peers, and the inefficient utilization of peer bandwidth (since the peer bandwidth cannot be used to delivery texture data during the process of finding a good peer selection strategy). Furthermore, metaverses are highly dynamic environments such that churn, avatar mobility, and time-varying peer capacities will significantly degrade the feasibility and efficiency of offline peer selection strategies.

We introduce a light-weight peer selection strategy, shown in Algorithm 1, which reduces the server bandwidth cost by

enabling each peer to repeatedly *update* its strategy for each texture that it requests. More specifically, all requests are sent by peers in a repeated fashion, *i.e.*, the strategy of each request can be changed if (1) the current content provider of the request is overloaded, and (2) the corresponding peer finds an underloaded provider within the latency constraint.

Algorithm 1: Peer selection for each peer $i \in P$.

1 Let t be the local time of peer i;
2 Let t_k be the time when r_k^i is generated;
3 Let a_i^k be the provider of the request r_k^i;
4 **foreach** $r_k^i \in R_k$ **do**
5 Contact a peer j from $S_k(d)$ u.a.r. ;
6 $a_i^k \leftarrow j$;
7 Send r_k^i to peer a_i^k;
8 Let h_k^i be height of r_k^i at peer j;
9 **if** $h_k^i \leq d \cdot u_j$ **then**
10 Stop contacting other peers;
11 **while** $t \leq t_k + d$ *and* $h_k^i - (d - t + t_k)u_{a_i^k} \geq w_k$ **do**
12 Contact a peer j' from $S_k(d - t + t_k)$ u.a.r. ;
13 Let $l_{j'}$ be the load of peer j';
14 **if** $l_{j'} \leq (d - t + t_k)u_{j'}$ **then**
15 $a_i^k \leftarrow j'$;
16 Send r_k^i to peer a_i^k;
17 **else**
18 Go to line 11;
19 Let h_k^i be height of r_k^i at peer j';
20 **if** $h_k^i < w_k + (d - t + t_k) \cdot u'_j$ **then**
21 Stop contacting other peers;
22 Send the request for texture k directly to the server;

Our approach is an online algorithm such that each peer performs the peer selection process and the texture transmission process in parallel. Each request is sent to a peer once it is generated (line 7 in Algorithm 1). Each peer i can change its strategy for each request in a repeated fashion if the bounded jump rule is satisfied. Also, our algorithm is light-weight such that each peer i needs to contact u.a.r. (*i.e.*, uniformly at random) only one other peer in $S_k(d')$ for each request r_k^i with latency constraint d' at one time, where $S_k(d') = \{i \in S_k : l_i < d'u_i\}$. The information of S_k (the set of peers that have texture k in their caches) is maintained at the server and can be piggybacked onto the update packets in practical implementations. The current latency constraint (*i.e.*, $d' = d - t + t_k$) of the request is calculated at the requesting peer, where t and t_k are the current time and the time when the request is generated, respectively. It is worth noting that every peer in S_k has the same probability being selected, although some peers have failed to satisfy the bounded jump rule in previous rounds. Because both requests and caches of peers are highly dynamic due to the avatar mobility.

The key idea of the bounded jump rule is to restrict the change of strategies such that the potential of the system decreases monotonically. Since all requests are processed by their content providers in a FIFO fashion, the peers that have the requests which can be processed within their latency constraints will not consider changing their current strategies (see lines 9 and 20 in Algorithm 1). From the perspective of congestion games, those requests (*i.e.*, players in games) have no incentives to change their strategies since their costs cannot be reduced (for pure Nash equilibrium) or not significantly reduced (for ϵ-Nash equilibrium).

More formally, for a given request r_k^i (sent by peer i for texture k) at peer a_i^k with a weight w_k and a height $h_k^i(t)$, by Eq. 13, the cost of r_k^i at time t is

$$c_{r_k^i}(\mathbf{a}(t)) = \min\{\max\{h_k^i(t) - (d - t + t_k)u_{a_i^k}, 0\}, w_k\},$$

where $\mathbf{a}(t)$ is the strategy profile of all peers at t, and $(d-t+t_k)$ is the latency constraint of texture k at t. This implies

$$c_{r_k^i}(\mathbf{a}(t)) = w_k \iff h_k^i(t) - (d - t + t_k)u_{a_i^k} \geq w_k. \quad (17)$$

Therefore, a peer will consider to change its strategy for its requests only when the costs of requests equal their weights, otherwise it will stop contacting other peers (line 21 in Algorithm 1). If the requesting peer of request r_k^i (*i.e.*, peer i) finds another peer j' at t (line 12 in Algorithm 1) during the repeated update process (the loop from line 11 to line 21 in Algorithm 1) such that $l_{j'}(t) \leq (d - t + t_k)u_{j'}$, it will send the request r_k^i to peer j'. It is worth noting that multiple peers may find peer j' and send requests to peer j' at the same time. In the worst case, only the first request can be processed by peer j' since peers process requests in a FIFO manner. Finally, the request will be sent to the server if $c_{r_k^i}(\mathbf{a}(t)) = w_k$ for any $t \in [t_k, t_k + d]$. It should be noted that no requesting peer will change its selection mid-way during a download, since it will stop contacting other content providers if it can download the request texture from the current content provider (lines 9 and 20 in Algorithm 1).

With the bounded jump rule, it is easy to observe that the potential function of game G is monotonically decreasing such that for a given set of requests and $\forall t_1, t_2$, we have

$$t_1 \leq t_2 \implies \Phi(\mathbf{a}(t_1)) \geq \Phi(\mathbf{a}(t_2)). \quad (18)$$

Furthermore, our algorithm applies to a fully distributed and concurrent setting such that all peers can change their strategies for their requests at the same time without a centralized coordination.

4. ANALYSIS

Recall that the proposed peer selection strategy follows principles and mechanisms of congestion games, and thus it is carried out in a repeated fashion (see Algorithm 1). More specifically, every peer i will repeatedly contact a provider j in S_k for each texture k that it requests (*i.e.*, r_k^i), and it will change its strategy for the request (*i.e.*, a_i^k) if the bounded jump rule (see lines 11 and 14 in Algorithm 1) is satisfied. In addition, peers perform this process individually without centralized coordination.

We formulate the problem of minimizing the server bandwidth cost as a congestion game, and we define the potential of the congestion game as the server bandwidth cost (see Eq. 16), which is minimized when a Nash equilibrium (a steady state) is reached. With the proposed bounded jump rule, a steady state is a ϵ-Nash equilibrium of the congestion game. Therefore, a critical question to investigate is *how fast does the system reach a ϵ-Nash equilibrium from any state?* In what follows, we seek to investigate the upper bound of the convergence time of our peer selection strategy.

Recall that the potential function of the game G with the bounded jump rule is monotonically decreasing in the presence of concurrent changes of peers' strategies (peers update their content providers (*i.e.*, peers in M) individually), because peers only consider to change the strategies of the requests that are sent to overloaded peers. More specifically, peer i will change the content provider of request r_k^i, if and only if $h_k^{i,j} - (d - t + t_k)u_j \geq w_k$ (ee Eq. 17 and Line 11 in Algorithm 1), where t and t_k are the local time and the time when the request is generated, respectively. The equivalent condition is $h_k^{i,j} + (t - t_k)u_j - w_k \geq d \cdot u_j$. Therefore, we can analyze the convergence rate of the proposed peer selection strategy via analyzing the convergence rate of the corresponding congestion game.

THEOREM 1. *Given a game $G = (N, M, (A_i)_{i\in N}, (f_e)_{e\in M})$ that satisfies the bounded jump rule, where $N = \cup_{k\in W} R_k$ and $M = \cup_{k\in W} S_k$ is the set of requests and content providers, respectively. Then a ϵ-Nash equilibrium can be reached within $O(\log n)$ rounds, where n is the number of online peers.*

PROOF. Let $\Phi(t)$ be the potential of the game at time t. According to Eq. 16, we have $\Phi(t) = \sum_{k\in R(t)} w_k$. Similarly, let $S(t) = \cup_{k\in R(t)} S_k(d)$ be the set of underloaded peers at time t, where $S_k(d) = \{i \in S_k : l_i < d \cdot u_i\}$ denotes the set of underloaded peers that have texture k in their caches. Let $n = |P|$ and $m_t = |S(t)|$ be the number of online peers and underloaded content providers at time t, respectively.

Note that a pure equilibrium can be reached at time t if (1) $\Phi(t) = 0$ (all requests can be delivered by peers within the latency constraint), or (2) $m_t = 0$ (no more underloaded content providers). Therefore, we introduce a non-increasing function $\Psi(t) = m_t\Phi(t)$ to analyze the convergence rate of proposed peer selection strategy.

At time $t+1$, let $\mathbf{x} = \{x_1, x_2, ..., x_{m_t}\}$ be a vector, where x_i denote the total size of requests that migrate to peer $i \in S(t)$. Obviously, $\Phi(t) = \sum_{i=1}^{m_t} x_i$. By applying Cauchy-Schwarz inequality to $\mathrm{E}[\Psi(t+1)]$, which is the expected value of $\Psi(t+1)$, we have the following:

$$\begin{aligned}\mathrm{E}[\Psi(t+1)] &= \mathrm{E}[m_{t+1}\Phi(t+1)] \\ &\leq \left(\mathrm{E}[m_{t+1}^2]\mathrm{E}[\Phi^2(t+1)]\right)^{\frac{1}{2}}. \end{aligned} \quad (19)$$

Noting that the square root function $f : x \to \sqrt{x}$ and the function $g : x \to x^2$ are convex functions, we apply Jensen's inequality to Eq. 19 and get

$$\begin{aligned}&\mathrm{E}[\Psi(t+1)] \leq \mathrm{E}[m_{t+1}]\mathrm{E}[\Phi(t+1)] \\ &= \sum_{m=1}^{m_t} m \Pr(m_{t+1} = m)\mathrm{E}[\Phi(t+1)|m_{t+1} = m], \end{aligned} \quad (20)$$

where m is the number of underloaded content providers in $S(t+1)$ at time $t+1$. The expected value of the potential function given m is

$$\begin{aligned}\mathrm{E}[\Phi(t+1)|m_{t+1} = m] &= \Phi(t) - \frac{\binom{m-1}{m_t-1}}{\binom{m}{m_t}} \sum_{i=1}^{m_t} x_i \\ &= \left(1 - \frac{m}{m_t}\right)\Phi(t). \end{aligned} \quad (21)$$

Combining Eq. 21 and Eq. 20, we have the following:

$$\mathrm{E}[\Psi(t+1)] \leq \sum_{m=1}^{m_t} m \left(1 - \frac{m}{m_t}\right)\Phi(t)\Pr(m_{t+1} = m) \quad (22)$$

$$\leq \frac{m_t}{4}\Phi(t)\sum_{m=1}^{m_t}\Pr(m_{t+1} = m) = \frac{1}{4}\Psi(t). \quad (23)$$

Let τ be the number of rounds that a ϵ-Nash equilibrium (*i.e.*, $\Psi(t+\tau) = O(1)$) is reached from time t. By Lemma 2 (see right column), we have

$$\mathrm{E}[\tau|\Psi(t)] \leq \log\left(\frac{\Psi(t)}{\Psi(t+\tau)}\right) = \log\left(\Psi(t)\right). \quad (24)$$

Note that the maximum value of of $\Psi(t)$ is $|T|n^2$ ($|T|$ is the total number of textures), which occurs in the case when every peer requests all the textures in the region. Therefore, $\mathrm{E}[\tau] \leq 2\log(|T|n)$. By Markov's inequality, we have that

$$\Pr(\tau \geq 40\log(|T|n)) \leq \frac{\mathrm{E}[\tau]}{40\log(|T|n)} = 0.05. \quad (25)$$

This implies that $\Pr(\tau \leq 40\log(|T|n)) \geq 0.95$. As a result, with a high probability, a ϵ-Nash equilibrium can be reached from any state within $O(\log n)$ rounds. □

LEMMA 2. *Let $X_1, X_2, ...$ denote a sequence of non-negative random variables and assume that for all $i \geq 0$*

$$\mathrm{E}[X_i|X_{i-1} = x_{i-1}] \leq \alpha \cdot x_{i-1}$$

for some constant $\alpha \in (0, 1)$. Furthermore, fix some constant $x^* \in (0, x_0]$ and let τ be the random variable that describes the smallest t such that $X_t \leq x^*$. Then,

$$\mathrm{E}[\tau|X_0 = x_0] \leq \frac{2}{\log(1/\alpha)} \cdot \log\left(\frac{x_0}{x^*}\right). \quad (26)$$

PROOF. The complete proof is shown in [12]. □

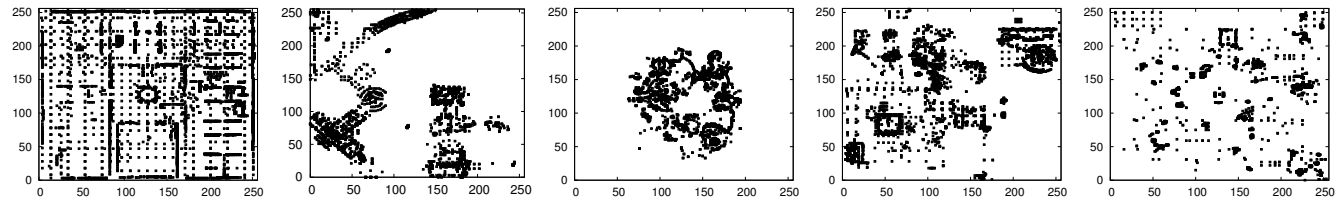
Theorem 1 indicates that the proposed peer selection strategy can rapidly reach a ϵ-Nash equilibrium within $O(\log n)$ communication rounds from any state, where n is the total number of online peers in the region.

5. EVALUATION

We now evaluate our peer-assisted texture streaming system using a comprehensive simulation program with input from realistic traces collected from the most popular metaverse application - Second Life. We choose Second Life to conduct our evaluation, as the Second Life Viewer[2] and OpenMetaverse[3] are open source and provide functionalities to collect the texture information and location information of avatars from Second Life servers.

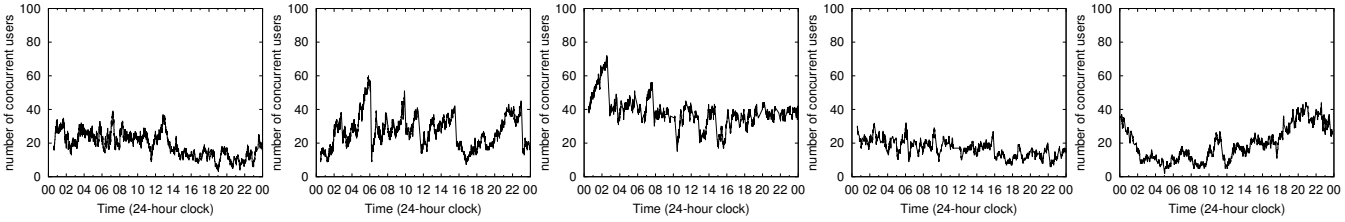
For comparisons, we have also simulated the following three peer selection algorithms using the same implementation settings with the proposed algorithm: (1) C/S scheme, where all the requests are solely served by the centralized server; (2) FLoD [15], where each peer can send its requests to AoI neighbors; if AoI neighbors fail to respond, it will send the requests to the server; (3) BAPS [7], which allocates the upload capacities of peers into channels and each peer will select the provider that in the currently connected channels;

[2] `http://wiki.secondlife.com/wiki/Source_downloads`
[3] `http://openmetaverse.org/`



(a) Freebies (b) Waterhead (c) Orientation Island (d) Mauve (e) Japan Resort

Figure 1: 2D positions of textures in different regions, each of which is 256×256 meters.

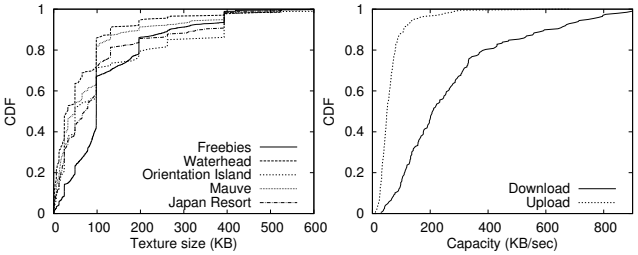


(a) Freebies (b) Waterhead (c) Orientation Island (d) Mauve (e) Japan Resort

Figure 2: Number of concurrent users in different regions during a day.

Region	Number/Size of Textures	User Hours
Freebies	2940 / 0.71 GB	201.1
Waterhead	407 / 0.21 GB	580.1
Orientation Island	87 / 0.56 GB	604.7
Mauve	621 / 0.57 GB	270.7
Japan Resort	551 / 0.41 GB	364.5

Table 2: Total number and size of textures and user hours of different regions during a day.



(a) Texture size (b) Peer bandwidth

Figure 3: Distribution of (a) textures sizes, and (b) peer bandwidth.

if the provider is overloaded, it will pick a peer from AoI neighbors as the provider; if that fails, it will randomly select a peer beyond of the AoI; eventually the request will go to the server if no appropriate provider is found.

5.1 Data Collection and Simulation Setup

We collect the realistic texture information of 5 regions (*i.e.*, Freebies, Waterhead, Orientation Island, Mauve, and Japan Resort) from Second Life. The 2D positions of textures, and the distributions of texture sizes of different regions are shown in Figures 1 and 3(a), respectively. The total size of textures and user hours (*i.e.*, the sum of the combined length of all user sessions) during a day in these regions are summarized in Table 2. Before logging into each region, we clear the cache of our client to store all the textures after we log in. After logging into each region, we move our avatar within the region step by step. Specifically, we stop the avatar at each step and wait for all textures within the AoI to be downloaded and rendered before moving to the next step until the whole region is covered and all textures within the region have been downloaded. When logged into the regions with cleared cache, a regular client requires 10 to 20 seconds to download all the textures within the avatar's AoI. We choose 5 and 10 seconds as two latency constraints of textures attempting to improve the user experience.

It is worth noting that we only collect the textures of objects instead of avatars, since the positions of textures on avatars may change over time due to the mobility and behavior of avatars. Since a texture can be applied to multiple objects that exist in different locations, textures can have multiple positions in our simulations.

We set up a discrete-event simulation using the one-day avatar mobility traces of the 5 regions from Second Life. Each collected mobility trace is a time series with a sampling rate of 10 seconds. The unique ID of each avatar and its trajectory are used to reproduce the movement of avatars. As shown in Figure 2, the number of concurrent users changes over time. We assume that every peer joins the system with a cleared cache, hence peers will request all the textures in their AoIs immediately after they join the system. As avatars move around in the virtual world, they will request the textures within their AoI that are not stored in their caches. The default cache size is set to 128 MB for each peer. We use a simple but efficient cache replacement algorithm which evicts textures that are farther away from the avatar with newly downloaded textures.

To evaluate the proposed algorithm in a heterogeneous environment, every peer is assigned an upload capacity and a download capacity randomly selected from a realistic data set collected from dslreports.com [5], as shown in Figure 3(b). Note that peers (especially peers with wireless access) may have bandwidth (both upload and download) fluctuations, we assume that each peer can utilize 80% of their assigned

capacities most of the time for downloading or sending textures. It is worth to point out that the bandwidth distribution of peers has no impact on the convergence rate of the proposed peer selection strategy, as shown in Theorem 1. The AoI radius of each peer is set to be 64 m, which is the default value in Second Life.

5.2 Server Bandwidth Consumption

Recall that our main objective is to reduce the server bandwidth consumption via utilizing the network capacities of participating peers, hence the main performance metric in our simulation is the average server bandwidth consumption in one day. Three different algorithms are evaluated using collected avatar mobility traces during a day. We sum up the total amount of textures required and total amount of texture data that is delivered by the server.

Table 3 shows the efficiency of different schemes in terms of server bandwidth consumption when the latency constraint of all textures is set to 5 seconds, *i.e.*, all the required textures should be delivered (from peers or the server) to the destinations within 5 seconds. We see that all the algorithms except the C/S scheme can reduce the server bandwidth consumption in all the regions via utilizing the bandwidth of participating peers. It can be observed that the proposed peer selection strategy outperforms the other alternatives by a substantial margin in any region.

Table 4 shows the server bandwidth consumption when the latency constraint of all textures is set to 10 seconds. Since relaxing the latency constraint of textures increases the number of requests that peers can satisfy, the server bandwidth consumption is reduced significantly. It is worth noting that relaxing the latency constraint of textures inevitably degrades the user satisfaction.

In the case when the number of textures is large (*e.g.*, the Freebies region with 2940 textures), the number of requests is large. Therefore the server bandwidth consumption is high due to the inadequate capacities of peers which cannot satisfy all the demand. Nevertheless, the proposed algorithm can save 23% and 39% of server bandwidth consumed by the C/S scheme in the Freebies region when the latency constraints of textures are 5 seconds and 10 seconds, respectively. In the case when the number of textures is relatively small (*i.e.*, less than 1000), peer-assisted schemes can save a large amount of server bandwidth. Moreover, we see in Tables 3 and 4 that the proposed peer selection strategy significantly outperforms existing algorithms since it can utilize the peer capacities more efficiently than FLoD and BAPS.

5.3 Server Request Ratio

Another performance metric we use is the server request ratio, which is defined as the percentage of requests in the region that are successfully served by the server. Intuitively, a higher server request ratio results in both a higher server bandwidth consumption and a higher server load. Figures 4 and 5 show the distribution of server request ratio of different algorithms when the latency constraints of textures are 5 seconds and 10 seconds, respectively.

We observe in Tables 5 and 6 that our peer selection strategy achieves a lower expected server request ratio than other existing algorithms in all regions and under all implementation settings. This is because the requests with the proposed peer selection strategy can be sent to underloaded peers rapidly within the latency constraint of textures, so that the peers' bandwidth is efficiently utilized. Therefore, the server load and bandwidth consumption can be reduced (since fewer requests will be processed). Benefitting from this, the system scalability can be improved.

Region	FloD	BAPS	Proposed
Freebies	76.3 %	66.3 %	**50.5** %
Waterhead	25.1 %	20.1 %	**10.2** %
Orientation Island	20.1 %	15.9 %	**7.5** %
Mauve	47.4 %	42.1 %	**31.8** %
Japan Resort	45.4 %	42.7 %	**30.2** %

Table 5: Expected server request ratio (smaller is better) in different regions, when the latency constraint of textures is 5 seconds.

Region	FloD	BAPS	Proposed
Freebies	66.3 %	49.6 %	**35.8** %
Waterhead	13.4 %	10.2 %	**6.6** %
Orientation Island	12.6 %	8.3 %	**5.7** %
Mauve	37.4 %	30.1 %	**20.1** %
Japan Resort	35.6 %	32.8 %	**19.4** %

Table 6: Expected server request ratio (smaller is better) in different regions, when the latency constraint of textures is 10 seconds.

Region	Communication Overhead	
	$d = 5$ Seconds	$d = 10$ Seconds
Freebies	2.36 KB/s	2.16 KB/s
Waterhead	0.26 KB/s	0.12 KB/s
Orientation Island	0.08 KB/s	0.06 KB/s
Mauve	1.02 KB/s	0.86 KB/s
Japan Resort	1.04 KB/s	0.88 KB/s

Table 7: Averaged communication overhead per peer under different latency constraints (*i.e.*, d) of textures in different regions.

5.4 Overhead

Since the protocol overhead incurred is naturally an important metric for every distributed algorithm, we investigate the communication overhead incurred by the proposed peer selection strategy in the last group of simulations. We compute the average overhead per peer as shown in Table 7. It is easy to observe that the communication overhead of our algorithm is very low, with a consumption of less than 3 KB/s per peer in all regions. This is because each peer contacts only one peer for each request at each round, and the sizes of request packets and their corresponding acknowledge packets are usually small (less than 100 bytes per packet). Considering that the requests are processed in a FIFO fashion at each peer, we believe that the processing overhead is reasonable in practice.

6. RELATED WORK

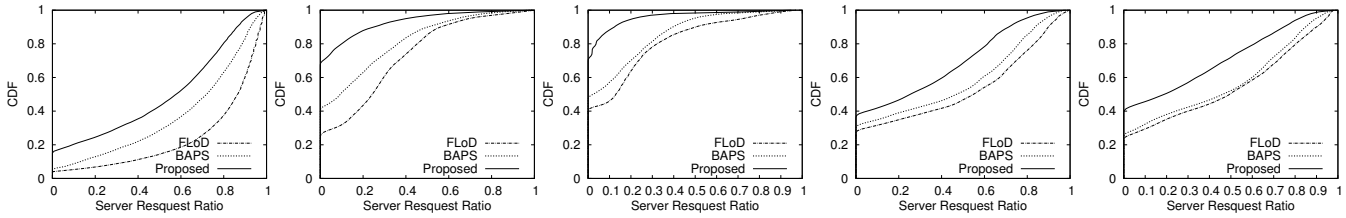
Peer-to-peer or peer-assisted 3D streaming systems for networked virtual environments have been studied extensively. They allow peers to store the 3D contents they receives in caches, and share their network resources and caches to serve others. Varvello *et al.* [29] proposed a structured P2P architecture (implemented on the top of Kad [20]) where all the 3D objects are stored in the participating peers instead of centralized servers. Peers need to query their required objects in Kad, and wait for the data before rendering

Region	Server Bandwidth Consumption (GB)				Fraction of Saved Bandwidth		
	C/S	FLoD	BAPS	Proposed	FLoD	BAPS	Proposed
Freebies	215.6	201.1	183.5	166.1	6.7 %	14.8 %	**23.0** %
Waterhead	17.5	8.4	6.6	2.8	52.2 %	62.2 %	**84.1** %
Orientation Island	20.3	11.1	10.5	5.4	45.2 %	48.1 %	**73.4** %
Mauve	21.6	17.6	16.4	13.7	18.3 %	23.8 %	**36.4** %
Japan Resort	32.6	25.4	24.7	19.4	22.1 %	24.2 %	**40.5** %

Table 3: Server bandwidth consumption (smaller is better) and the fraction of saved bandwidth (larger is better) of different algorithms during a day, when the latency constraint of textures is set to 5 seconds.

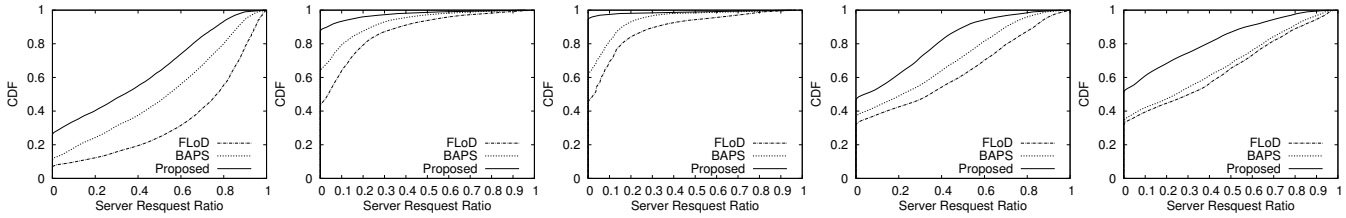
Region	Server Bandwidth Consumption (GB)				Fraction of Saved Bandwidth		
	C/S	FLoD	BAPS	Proposed	FLoD	BAPS	Proposed
Freebies	215.6	189.9	161.0	131.4	11.9 %	25.3 %	**39.0** %
Waterhead	17.5	5.1	3.0	1.3	70.9 %	82.8 %	**92.2** %
Orientation Island	20.3	7.7	6.5	3.2	61.9 %	68.2 %	**84.0** %
Mauve	21.6	14.9	12.5	9.3	30.5 %	41.8 %	**56.5** %
Japan Resort	32.6	21.3	20.3	12.6	34.6 %	37.9 %	**61.3** %

Table 4: Server bandwidth consumption (smaller is better) and the fraction of saved bandwidth (larger is better) of different algorithms during a day, when the latency constraint of textures is set to 10 seconds.



(a) Freebies (b) Waterhead (c) Orientation Island (d) Mauve (e) Japan Resort

Figure 4: Distribution of server request ratio (smaller is better) in different regions during a day, when the latency constraint of textures is set to 5 seconds.



(a) Freebies (b) Waterhead (c) Orientation Island (d) Mauve (e) Japan Resort

Figure 5: Distribution of server request ratio (smaller is better) in different regions during a day, when the latency constraint of textures is set to 10 seconds.

those objects in their virtual environments. This pure P2P architecture suffers from low consistency (due to the query delay) and persistency (due to the avatars' mobility).

Hu *et al.* [15] proposed FLoD, which is the first framework for peer-assisted 3D streaming in metaverses. FLoD enables each peer to send its requests to a randomly chosen AoI neighbor, thus reduce the server bandwidth cost. If the AoI neighbors fail to respond, the peer will send the requests to the server. To minimize the server bandwidth cost in FLoD respecting the bandwidth constraints of peers, Chien *et al.* [7] proposed a bandwidth aware peer selection algorithm, named BAPS, that improves the utilization of peer bandwidth via bandwidth reservation.

Our objective is to minimize the server bandwidth cost by enabling peers to rapidly select underloaded content providers in a decentralized manner. The key idea is to balance the loads of peers. A number of algorithms have been proposed for dynamic load balancing, which is modeled as a weighted congestion game with selfish and non-cooperative users. A steady state in this context is a pure Nash equilibrium, in which no user is willing to change her/his strategy. Since the idea of using a potential function to measure the closeness to a Nash equilibrium was introduced by Even-Dar *et al.* [10], a number of protocols (*e.g.*, [8] [14]) have been proposed to balance the loads sequentially, *i.e.*, the migration events take place one at a time, and costs on users are updated immediately. More recently, concurrent protocols (*e.g.*, [4] [3] [1]) for load balancing have been proposed to improve the convergence time by allowing users to change their strategies concurrently. We adopt the concept of these load balancing techniques by applying it to peer selection strategy in peer-assisted 3D streaming systems.

Our system distinguishes itself from all existing systems in the following two important aspects. First, we consider the latency requirements of 3D contents, and design a light-weight peer selection strategy to minimize the server bandwidth respecting the latency constraints of contents and the bandwidth constraints of peers. Second, the peer' bandwidth is optimally utilized in a decentralized manner.

7. CONCLUSION

In this paper we have introduced a peer-assisted texture streaming system for metaverses that minimizes the server bandwidth cost without degrading the end-user satisfaction. We formulate the problem of server bandwidth minimization as a congestion game, and use the concept of congestion games to design a peer selection strategy. Our algorithm is light-weight, online, and can efficiently utilize the bandwidth of heterogeneous peers in a decentralized manner by enabling each peer to repeatedly update its content providers independently and concurrently. We evaluate our algorithm through an extensive comparison study based on simulations using realistic texture information and avatar mobility traces collected from Second Life. As shown by our simulation results, the proposed algorithm can effectively reduce the server bandwidth cost and increase the scalability of metaverses. Encouraged by our results from this paper, we are currently working on a real-world deployment of our proposal as an extension to Second Life.

Acknowledgment

This research has been funded in part by A*Star SERC PSF grant 082 101 0028.

8. REFERENCES

[1] H. Ackermann, S. Fischer, M. Hoefer, and M. Schöngens. Distributed Algorithms for QoS Load Balancing. In *SPAA '09: 21^{st} Annual Symposium on Parallelism in Algorithms and Architectures*, pages 197–203, 2009.

[2] ActiveWorlds. Online: http://www.activeworlds.com.

[3] B. Awerbuch, Y. Azar, and R. Khandekar. Fast Load Balancing via Bounded Best Response. In *SODA '08: 19^{th} Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 314–322, 2008.

[4] P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, Z. Hu, and R. Martin. Distributed Selfish Load Balancing. In *SODA '06: 17^{th} Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 354–363, 2006.

[5] Broadband Reports and Speed Test Statistics. Online: http://www.dslreports.com/archive.

[6] N. Buchbinder and J. Naor. The Design of Competitive Online Algorithms via a Primal-Dual Approach. *Foundations and Trends in Theoretical Computer Science*, 3:93–263, 2009.

[7] C.-H. Chien, S.-Y. Hu, and J.-R. Jiang. Bandwidth-Aware Peer-to-Peer 3D Streaming. In *NETGAMES '09: 8^{th} Annual Workshop on Network and Systems Support for Games*, 2009.

[8] S. Chien and A. Sinclair. Convergence to Approximate Nash Equilibria in Congestion Games. In *SODA '07: 18^{th} Annual ACM-SIAM Symposium on Discrete algorithms*, pages 169–178, 2007.

[9] Y. Emek, M. M. Halldórsson, Y. Mansour, B. Patt-Shamir, J. Radhakrishnan, and D. Rawitz. Online Set Packing and Competitive Scheduling of Multi-Part Tasks. In *PODC '10: 29^{th} ACM Symposium on Principles of Distributed Computing*, pages 440–449, 2010.

[10] E. Even-Dar and Y. Mansour. Fast Convergence of Selfish Rerouting. In *SODA '05: 16^{th} Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 772–781, 2005.

[11] A. Fabrikant, C. Papadimitriou, and K. Talwar. The Complexity of Pure Nash Equilibria. In *STOC '04: 36^{th} Annual ACM Symposium on Theory of Computing*, pages 604–612, 2004.

[12] Fischer, Simon and Olbrich, Lars and Vöcking, Berthold. Approximating Wardrop Equilibria with Finitely Many Agents. *Distributed Computing*, 21:129–139, 2008.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[14] P. W. Goldberg. Bounds for the Convergence Rate of Randomized Local Search in a Multiplayer Load-balancing Game. In *PODC '04: 23^{rd} Annual ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2004.

[15] S.-Y. Hu, T.-H. Huang, S.-C. Chang, W.-L. Sung, J.-R. Jiang, and B.-Y. Chen. FLoD: A Framework for Peer-to-Peer 3D Streaming. In *INFOCOM '08*, pages 1373 –1381, 2008.

[16] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P-VOD System. In *SIGCOMM '08: ACM SIGCOMM Conference on Data Communication*, 2008.

[17] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang. Performance Bounds for Peer-Assisted Live Streaming. In *SIGMETRICS '08: International Conference on Measurement and Modeling of Computer Systems*, pages 313–324, 2008.

[18] R. T. B. Ma, S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau. A Game Theoretic Approach to Provide Incentive and Service Differentiation in P2P Networks. In *SIGMETRICS '04: International Conference on Measurement and Modeling of Computer Systems*, pages 189–198, 2004.

[19] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-drIven MEsh-Based Streaming. In *INFOCOM '07: 26^{th} IEEE International Conference on Computer Communications*, pages 1415 –1423, 2007.

[20] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *IPTPS '01: 1^{st} International Workshop on Peer-to-Peer Systems*, pages 53–65, 2002.

[21] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[22] I. A. Oliver, A. H. Miller, and C. Allison. Virtual Worlds, Real Traffic: Interaction and Adaptation. In *MMSys '10: 1^{st} Annual ACM SIGMM Conference on Multimedia Systems*, 2010.

[23] A. Panconesi and M. Sozio. Fast Distributed Scheduling via Primal-Dual. In *SPAA '08: 20^{th} Annual Symposium on Parallelism in Algorithms and Architectures*, pages 229–235, 2008.

[24] J. Puchinger, G. R. Raidl, and U. Pferschy. The Multidimensional Knapsack Problem: Structure and Algorithms. *INFORMS Journal on Computing*, 2010.

[25] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., 1986.

[26] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[27] Second Life. Online: http://secondlife.com.

[28] The Second Life Economy. Online: http://blogs.secondlife.com/tags/quarterly_economic_report.

[29] M. Varvello, C. Diot, and E. Biersack. P2P Second Life: Experimental Validation Using Kad. In *INFOCOM '09*, pages 1161–1169, 2009.

[30] J. Wang, C. Huang, and J. Li. On ISP-friendly Rate Allocation for Peer-assisted VoD. In *MM '08: 16^{th} ACM International Conference on Multimedia*, pages 279–288, 2008.