Bandwidth-Aware Peer-to-Peer 3D Streaming

Chien-Hao Chien, Shun-Yun Hu, Jehn-Ruey Jiang

Department of Computer Science and Information Engineering
National Central University, Taiwan, R.O.C.

ABSTRACT

Peer-to-Peer (P2P) streaming support for 3D content (i.e., P2P 3D streaming) has recently been proposed to provide more affordable and cost-effective content streaming in virtual environment (VE) applications. However, the generally limited client upload bandwidth requires maximal bandwidth utilization for effective streaming. This paper proposes Bandwidth-Aware Peer Selection (BAPS), a peer selection strategy that improves the bandwidth utilization for 3D streaming. BAPS avoids request contentions and peer overloading as object and user densities increase, thus improving both bandwidth utilization and system scalability. We compare BAPS with strategies that select from only peers within the area of interest (AOI) as data sources and without bandwidth capacity considerations. Our evaluation shows that BAPS achieves better performance in general and maintains a stable minimal quality of service (QoS) for streaming, which is important for commercial applications.

Categories and Subject Descriptors

I.3.2 [**Computer Graphics**]: Graphics Systems–Distributed / network graphics

Keywords

peer-to-peer, networked virtual environment, 3D streaming, peer selection, bandwidth-aware

1. INTRODUCTION

Networked Virtual Environments (NVEs) are computer simulations that combine networked communications and 3D graphics techniques to provide immersive and responsive virtual interactions. The early military simulator SIMNET [14], and the recent Massively Multiplayer Online Games (MMOGs), such as *World of Warcraft* (WoW)[1] or *Second Life* (SL)[2], are all well-known examples of networked virtual environments. Rendering a scene in a virtual environment (VE) requires a combination of various 3D objects (e.g., mesh models and textures), currently often obtained from a full installation via a DVD or a network download. However, downloading and installing the entire content to local storage might take a long time. To address this issue, the technique of *3D Streaming* has been utilized.

3D Streaming [1, 4, 5, 11] refers to the real-time and continuous transmission of 3D content through networks. Users only need to download the data for rendering a given scene before navigation, without having to wait for the entire content download to complete. For example, *Second Life* delivers terabytes of user-generated content with their 3D streaming technology [7]. 3D streaming is similar to media streaming [10] where users are allowed to immediately use the data when the data is only partially received. Before transmission, the data also needs to be fragmented into many pieces, so that users will be able to see the content progressively. However, differences exist between 3D streaming and media streaming. For instance, media streaming transmits data pieces according to the time sequence of the video, so the transmission sequence of the pieces is fixed. On the other hand, 3D streaming transmits data based on 3D objects viewable to the user. So different viewing angles or distances produce individually different transmission sequences.

If 3D streaming is supported by a client-server (C/S) architecture, then all content is provided by a central server. When a user requests new data, the request is sent directly to the central server, waiting for the server's processing and response. However, C/S architectures cannot scale easily with user size, because the bandwidth or computing resources of any given server is often fixed, whereas the number of users might increase with user activities, and may in some cases overload the server's capacity (e.g., a *flash crowd* could gather for a concert). When concurrent requests from users exceed the server's capacity, the efficiency and quality of service (QoS) of the system will degrade. To address this problem, peer-to-peer (P2P) architectures [3, 7, 12] have been proposed to support 3D streaming. P2P networks are characterized by the design where every user plays the role of both the provider and the requester: each user shares data with other users on the network. In other words, because users can get what they need from other users, the data source is not limited to the server. With this architecture, the total network bandwidth or computing resources will increase as user size scales, improving the system's scalability.

[1] http://www.worldofwarcraft.com

[2] http://www.secondlife.com

Recent proposals in P2P-based 3D streaming (e.g., FLoD [7]) show the benefits of using the P2P architecture to reduce server load. However, request contention and overloading can occur for some nodes [15], so some users' bandwidth cannot be properly used when the data sources are limited to only a few nearby neighbors in the virtual space.

This motivates us to propose a Bandwidth-Aware Peer Selection (BAPS) method that avoids request contention and overloading. Unlike existing P2P 3D streaming schemes, BAPS allows users to send requests to neighbors within the AOI (area of interest) as well as to other users. Therefore, more data sources become available. Furthermore, BAPS adopts bandwidth reservation and the *Tic-for-Tac* concept from BitTorrent [6] to 1) ensure a stable level of quality of service (QoS); 2) improve bandwidth utilization, and 3) achieve higher scalability. We verify BAPS with simulations and compare its performance with related P2P 3D streaming schemes to show its advantages.

The rest of the paper is organized as follows. Section 2 defines our problem and describes the expected results; Section 3 proposes our peer selection and piece selection methods; Section 4 presents the results of the experiments; and conclusions will be given in Section 5.

2. PROBLEM FORMULATION

We formulate our user scenario based on *Second Life*, as it is a more generalized model for virtual worlds among current MMOGs. There are two types of network transmissions in *Second Life*: *states* and *content* data. States provide information about what surrounds the current user, such as the positions of other users and the placements or status of objects (e.g., how filled is a glass of water, or how much money an item costs). When a user is aware of the surrounding objects through the states, requests of the 3D objects (i.e., the content data) can then be sent to the server. States therefore consist of smaller packets and require higher responsiveness and security. On the other hand, according to Huang et al. [9], 61% to 88% of the network traffic in *Second Life* is for textures, which is a type of 3D content. Therefore, to reduce server loading and maximize the number of concurrent users, the sensible priority is to reduce bandwidth usage due to the transmission of 3D objects. This paper is based on a former study of FLoD [7], which is a framework that addresses issues of 3D streaming in a P2P network. In FLoD, there are two sources for content download: AOI neighbors and the server. AOI neighbors refer to the other users who are within a given user's visibility, or *Area of Interest* (AOI). As AOIs may intersect, users can find other users with similar object interests to form a neighbor group, which can then serve as another source for content download. When a user wants to download certain objects in the AOI, the user can send a request to another user who owns the object from the AOI neighbor group. If there are no AOI neighbors to request, requests are *then* sent to the server. However, as there is a limit on concurrent servable requests, a download might be delayed due to the inability to send or respond to requests. We identify three problems in FLoD's basic strategy:

1. Non-server content sources are limited to current AOI neighbors. However, other non-AOI users who have been in the same area may still possess the objects of interest. The download thus can be inefficient due to the insufficiency in content sources.

2. Random peer selection can cause bandwidth waste and request jam. It is found that a naive random peer selection causes users of different upload bandwidth to carry the same loading [15], which means that the users of low upload bandwidth can be in a request jam (i.e., receiving requests beyond their capacities), whereas users with high bandwidth capacity are idling. The upload bandwidth thus is not efficiently allocated.

3. Unstable connections exist between peers. As connections are created and broken very dynamically according to user movements, the response time of the content requests cannot be estimated accurately. The quality of streaming thus would suffer and cannot be consistently guaranteed.

In summary, it is found that getting content from only the AOI neighbors limits the download sources, and random selection causes unbalanced workload and request contention. Sung et al. [15] propose to solve the above problems by reserving a list of past AOI neighbors and using a multi-level AOI for the request areas. However, the work neglects the difference in upload bandwidth between users and does not assure that the source nodes have sufficient bandwidth to provide for download. The P2P network thus may not transmit most effectively. Besides, when the user density is high, even with multi-level AOI requests, users within a single area might still receive excessive data requests. Therefore, we propose an improved mechanism with the following objectives:

1. Develop a bandwidth allocation method to reduce the waiting time to fulfill download requests and improve the streaming quality.

2. Construct *peer lists* to provide additional data sources, and adopt a *Bandwidth-Aware Peer Selection* (BAPS) strategy, so that peer selection is not limited to only AOI neighbors.

3. Introduce the *Tit-for-Tat* in BitTorrent [6] to provide prioritized download for users with larger upload bandwidth, so that they are able to provide more data to other users sooner, reducing the server's loading for better scalability.

3. DESIGN OF BAPS

3.1 Assumptions and Basic Ideas

Based on the ideas of FLoD [7] and the work of Sung et al. [15], we divide a virtual world into several *scenes* with fixed sizes. Each scene consists of a *scene description* with the number of objects and each object's placement and size. When a user logins the virtual world, he/she will be informed of the scene descriptions according to the user's AOI coverage. In this work we assume that there is a *gateway server* to perform this task (i.e., the server notifies each user the necessary scene descriptions) in order to focus on the P2P aspect of content exchange. Note that in the original FLoD design, scene descriptions are provided by a P2P overlay as well. However, as scene descriptions are often much smaller than the actual content, we assume that they have negligible contribution to bandwidth and may be performed in either a client-server or P2P fashion. The required 3D objects are then downloaded according to the scene descriptions. When new objects are found, a list of required objects is formulated by a *Piece Selection* procedure to determine the content pieces to be downloaded and their priorities. Once a piece is selected, the *Peer Selection* procedure will find
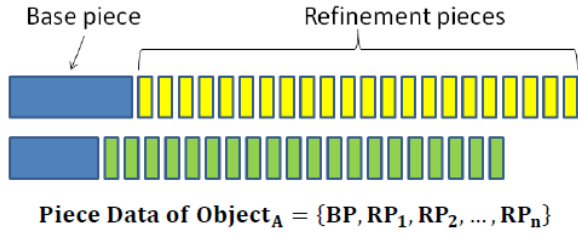
Piece Data of Object$_A$ = {BP, RP$_1$, RP$_2$, ..., RP$_n$}

Figure 1: Data Structure of 3D Streaming Object

an appropriate user to request. We describe the procedures below:

1) Piece Selection mainly deals with the transmission priority of piece request. We first assume that objects in the VE are fragmented into one or several pieces with Level of Detail (LoD) techniques [13]. As shown in Figure 1, every object is divided into a base piece (BP), and many refinement pieces (RP), where successive pieces depend on the previous ones, so every 3D object can be transmitted as a streaming content. When a user has downloaded a base piece, the object can be seen in its rough outline; and if the download continues with getting more refinement pieces, then the object will appear to be more refined. According to the nature of virtual world, Piece Selection prioritizes the download order with two considerations: (1) visual contribution of the piece and (2) object proximity in the virtual world.

As 3D objects are displayed progressively, BP has to be downloaded before showing RP_1, and R_{i-1} is required before R_i. So, download priority depends on the piece number, where pieces with smaller numbers are downloaded first.

$$Pr(Piece_i) = w(Piece_i)/dist(Object_A)$$

We thus define the priority Pr of $Piece_i$ of $Object_A$ as above, where $w(Piece_i)$ indicates the visual contribution (or weight) of the piece, and $dist(Object_A)$ represents the virtual distance between $Object_A$ and the user. Higher Pr values indicate higher priorities. When a user has to transmit several pieces at the same time, this value would help to decide which data piece has the higher priority.

2) Peer Selection decides to whom a request is sent. In P2P 3D streaming, in order to provide a good navigation, the system should focus on the timeliness of piece download. To reduce waiting time, we propose to use *bandwidth reservation* to allow requests be served as soon as they are received. We realize that if content sources are limited to AOI neighbors, insufficient downloadable sources and request jam may result. To solve these problems, we propose *Bandwidth-Aware Peer Selection* (BAPS), which includes *Bandwidth Allocation* and *Multi-Source Selection*, to help users allocating bandwidth properly and finding appropriate content providers. Similar to other media streaming, 3D streaming should assure a stable streaming quality, which can be achieved by avoiding excessive waiting time on piece requests. Considering that in residential networks, bandwidth resources are often limited and asymmetric for upload and download, we thus lower the allowable requests for users with lower bandwidth in order to avoid jamming in the requests. For users with higher bandwidth, we should also allocate the bandwidth efficiently to avoid resource idling.

By using pre-allocation, the upload bandwidth of a user is divided into several *connection channels* of identical sizes. When a requester asks a peer for content, a connection channel is created first, where the provider will reserve some upload bandwidth, so that requests will not be delayed due to request jam. If enough bandwidth cannot be allocated, the requester is denied connection and may need to look elsewhere for other peers. As object pieces are linearly dependent, when a user makes requests to some peers, it means that their AOIs have been overlapped at some time (past or present) and include the same objects. The provider thus is likely to own other content pieces needed by the requester. By reserving bandwidth at the provider, the request flow can be kept smooth and continuous (i.e., similar to a pipeline), which increases its efficiency.

3.2 BAPS algorithm

We now describe BAPS in more details according to the main stages in P2P 3D streaming [8]:

1) Object Discovery: Before performing the piece and peer selections, our first task is to identify which objects are within the user's view and thus are relevant to download. This is achieved by reading the scene descriptions obtained from the central server. As object discovery is not our main focus, we assume that such a simple method would suffice. If dynamic object creation / deletion / update were to be supported, then a more sophisticated mechanism can be used.

2) Source Discovery: To improve the source insufficiency due to requesting from only AOI neighbors, a *Peer List* is also included in the scene description, which describes the users who have ever downloaded the scene. The Peer List is constructed by the *gateway server*, which logs the users who request for each scene descriptions. So when other users request the scene descriptions, the server can randomly select some users who have requested the same scene description previously into the Peer List. Users can learn from the list about other users who have ever accessed the scene. This way, the potential sources for peer selection is increased.

3) State Exchange: This step includes *connection request* and *state exchange*. After knowing who are the content sources, we then need to know which peers own the required data pieces by exchanging some simple states [15]. We adopt the proactive push-based distribution of such meta states on piece availability (as proposed in [2,16]) to save the time on state exchange, and a passive pull-based method for the later content exchange (which tolerates more latency). In order to inform existing connected peers on the availability of pieces, we specify the state exchange peers as both the AOI neighbors and peers with established connected channels. When a user comes to a new scene which it has no knowledge of, it would connect with some randomly selected peers from the Peer List and AOI neighbors. Otherwise, the user will only connect with peers who are known to possess the desired content. When the size of known peers is lower than a pre-defined value, the server is asked to provide a new Peer List.

4) Content Exchange: This is the main stage where piece selection and peer selection are performed. We determine the request rp_i for piece p_i according to the Piece Selection policy as follows. If set A represents all the owners of p_i among known users; set B represents the known owners of p_i in the connected channels; and set C is the owners of p_i in

Table 1: Simulation parameters

World dimension (units)	1000x1000
Cell size (units)	100x100
AOI-radius (units)	100
Time-steps (10 steps = 1 sec)	1500
Object Size (KB)	100 - 300
Piece Size (KB)	5
Percentage of Base Piece	10%
Server Download/Upload Limit (KB)	1000/1000

Table 2: user bandwidth distribution

download (KB/sec)	upload (KB/sec)	Node Fraction
96	10	0.05
187	30	0.45
375	100	0.40
1250	625	0.10

AOI neighbors. As users in set B have reserved bandwidth for incoming requests, when a piece request is sent to any peer in set B, it will immediately be served without being delayed due to too many requests. We choose from set B a content provider who has not yet reached the maximum supportable requests. If no provider is found in set B, then we pick a source from set C. When no appropriate provider is found either in set B or set C, we examine whether we meet the Server Request Condition [7]. If so, we send a request to the server. In case we fail to find anyone to request, that means the total system resource is insufficient, and we would randomly select a few users in set A to ask for new connections.

When a provider receives a piece request, it should immediately decide whether to serve the request. In the original FLoD, the piece request procedure uses the *first come first serve* model, or FCFS. The benefit of this model is an equal opportunity for all requesters. Thus, the requested loading is evenly distributed. However, the FCFS model offers the same opportunity for users with high upload and users with low upload bandwidth, making the high capacity users to have idle bandwidth resource while unable to distribute content to others (called a *content bottleneck* [10]). To improve such scenario, when a requester asks for a connection channel, if the provider is fully loaded, connection preference is given to peers with higher upload bandwidth, using a *Tit for Tat* policy [6]. When connections are fully loaded, those with lower transmissions will be suspended, so that high capacity peers can obtain content faster and serve sooner.

4. EVALUATION

This section describes the evaluation of BAPS via simulations. We first present the simulation environment and methods, followed by the metrics used and result analysis. Our experiment is based on the FLoD architecture and procedures [7] (see Table 1 for the simulation parameters). At the initialization phase, all 3D objects are placed randomly in the VE, and the object sizes are between 100KB to 300KB. During the experiment, several nodes are created to represent users of the virtual world. Similar to the objects, their placements are random. At the beginning, all nodes remain at the initial positions until an initial set of AOI objects are all downloaded. This assures that users are equipped with some content to exchange with others before their movements. This way we can focus on the steady state behaviors during data distribution. The user bandwidth allocation is set as in [2] (see Table 2) to simulate a real environment, so that we may examine the performance of different algorithms under a realistic bandwidth distribution.

4.1 Metrics

The following metrics are used for performance evaluation:

Server Request Ratio (SRR): Users request data pieces from both peers and the server. When a user is unable to find a peer provider and that the Server Request Condition is matched, the user can request from the server. The proportion of pieces (in data size) obtained from the server is described in percentage as *Server Request Ratio* (SRR). Lower values of SRR mean lower loading for the server and better system scalability.

Fill Ratio: To evaluate the visual quality of a scene, a simple quantity is to measure the ratio between successfully downloaded content and the total interested (i.e., within AOI) content (in sizes). Higher ratio means the user has downloaded a more complete 3D scene and thus will have better visual quality.

Request Latency: The delay between sending a piece request to acquiring the piece is important, as it evaluates the efficiency of request serving and shows whether there is a request jam.

4.2 Performance Analysis

In the following simulations, we use two simulation setups to evaluate bandwidth utilization and the system's scalability: (1) Fixed user size and movement paths to maintain the same available bandwidth, while adjusting object quantities to evaluate bandwidth utilization; (2) Fixed object placements and quantities with varying user size to evaluate the system's scalability. Finally, we evaluate the streaming quality of with time-series in how fill ratio changes.

Bandwidth utilization: In a P2P network, data pieces are obtained from both server and peers. When users cannot find appropriate sources in the P2P network, the request is shifted to server. So if the SRR is high, then the P2P network may be under-utilized. By maintaining the same user size (100 peers) and movement paths while varying the object size (from 100 up to 500), we test the P2P network's performance under different workloads. We randomly place M objects in the VE, whose sizes are $D_i | i = 1, 2, 3..., M$. The total content size thus is: $\sum_{i=1}^{M} D_i$, which is also the maximum downloadable volume for a user. The average downloadable volume in AOI is shown as $AOIArea/WorldArea * \sum_{i=1}^{M} D_i$. For example, when there are 300 objects with an average size of 200KB, there will be a total of 60 MB of content, and the average content size in an AOI is 1.88MB.

As shown in Figure 2(a), when the content sources are extended from AOI neighbors to also include Peer List neighbors, with our Peer Selection strategy and the same number of objects, server's loading reduces 59.8% on average. For example, when there are 200 objects, the SRR is 14.6% in FLoD but only 8.6% in BAPS, which is about a 55.9% reduction on the server loading; when there are 500 objects, the reduction rate becomes 66.2%.

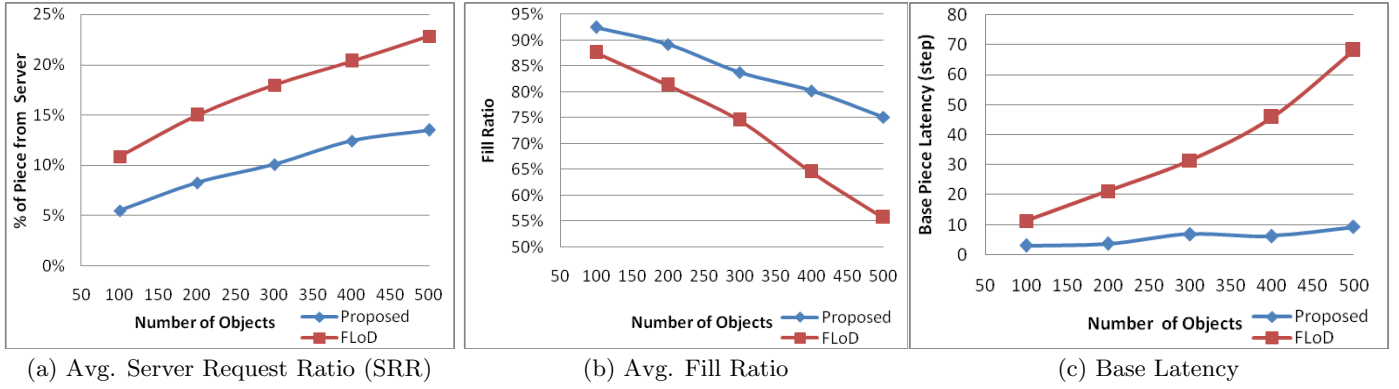The server loading is efficiently reduced due to having

(a) Avg. Server Request Ratio (SRR) (b) Avg. Fill Ratio (c) Base Latency

Figure 2: Bandwidth utilization evaluation under varying object sizes (100 peers)

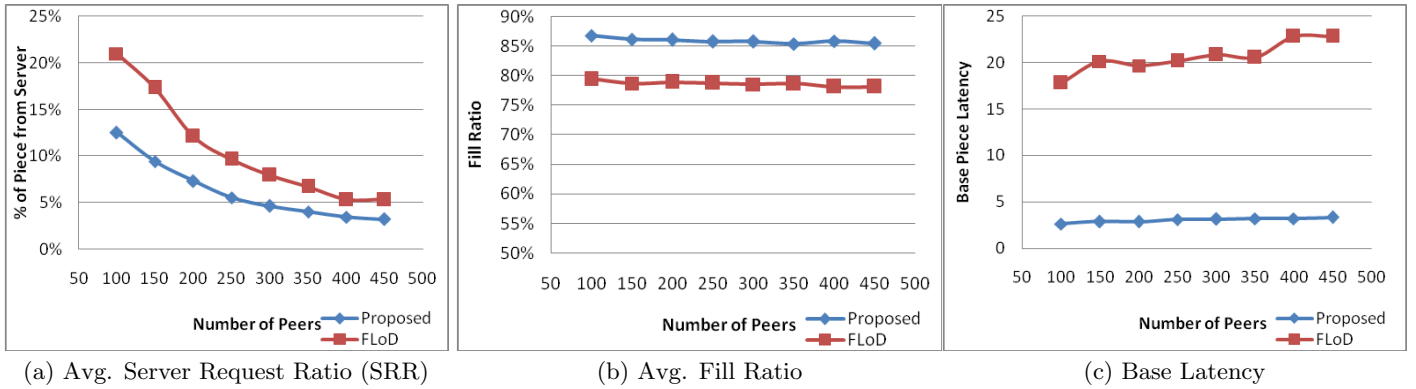(a) Avg. Server Request Ratio (SRR) (b) Avg. Fill Ratio (c) Base Latency

Figure 3: System scalability evaluation under varying peer sizes (300 objects)
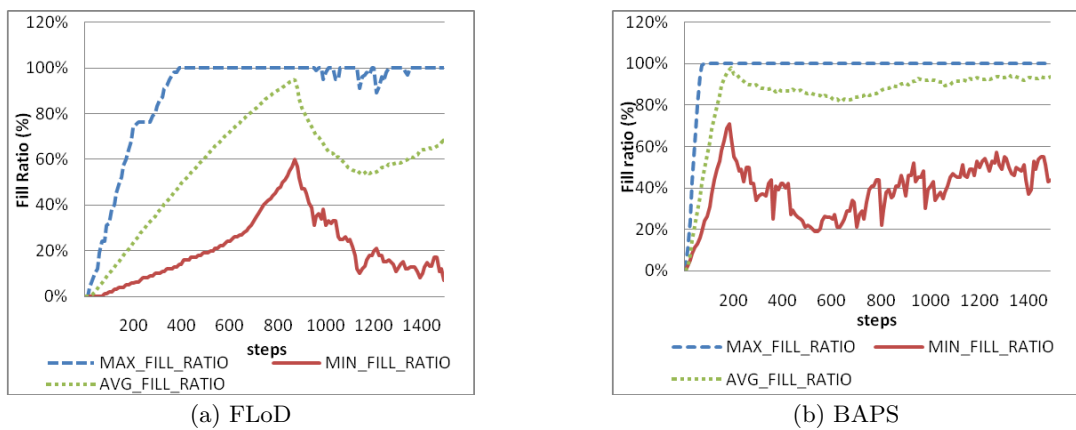

(a) FLoD (b) BAPS

Figure 4: Fill Ratio Time-series (100 peers, 300 objects)

multiple sources and the improved peer selection. However, we also need to know the bandwidth overhead of using Peer Lists. It is found that Peer List transmission takes 4.3% in the entire transmission, and the proportion increases as the number of objects increases. This shows that its cost is acceptable, but also indicates that as the object quantity grows, AOI neighbors will not provide sufficiently, so requests for Peer Lists would increase.

As for the fill ratio, Figure 2(b) shows that the fill ratio reduction in FLoD is higher than BAPS. When there are 100 objects, the fill ratio in FLoD is 87.5%, and when the object size increases to 500, the fill ratio is reduced to 55.7% with a reduction rate of 31.8%. So as object density increases, bandwidth becomes insufficient and the fill ratio suffers. As for BAPS, the fill ratio is reduced from 92.5% to 75.1%, which is 17% less compared to FLoD. Consequently, given identical bandwidth, BAPS is able to provide better utilization with a lower reduction in fill ratio.

Figure 2(c) shows the latency for getting the base piece (i.e., the *base latency* [7]), where the base piece is 10% of object size. We note that as object size increases, it takes more time to get the base piece. The latency curve of FLoD, however, grows noticeably faster than BAPS. It is likely due to the limited content sources and random peer selection in FLoD, as requests are distributed evenly to all peers, even those with low bandwidth. With BAPS, not only base piece download is prioritized, bandwidth is also reserved for base piece requests to ensure that users can obtain them faster.

System Scalability: We simulate different number of peers to evaluate the scalability of the system, while fixing the object size at 300. When users are unable to obtain content from peers, requests are shifted to the server. So we need to observe whether server loading increases as the number of user scales. Figure 3(a) shows the percentage of data obtained from the server in FLoD and BAPS. When user size increases, FLoD and BAPS both reduces server loading. But for BAPS the server request ratio is even lower. Figure 3(b) shows the average fill ratio after 1500 time-steps. The more limited content sources may produce the lower fill ratio for FLoD. Figure 3(c) shows the comparisons in base latency. We can easily observe the difference in performance from Figure 3(b) and Figure 3(c). One observation is that both FLoD and BAPS can effectively relive server loads and maintain relatively stable performances as user scales. However, BAPS in general achieves better performance, as connection channels are created to provide better guarantee on request latency.

Streaming Quality: Figure 4(a) and Figure 4(b) show the time-series of fill ratio under 100 peers and 300 objects. We see that FLoD takes more time to stabilize, and when users begin to move, the fill ratio decreases more significantly than in BAPS. In these figures, we show the maximum / average / minimum fill ratios achieved among all users. A fast increase in maximum fill ratio indicates the effective use of the Tit-for-Tat policy. It preferentially makes the users connect with peers of higher upload speed and suspend the non-performing connection channels. The minimum fill ratio indicates the percentage of peers with the worst streaming quality. An important observation from Figure 4(b) is that by reducing request jam, a more stable streaming is achieved even in the worst case scenario. For commercial providers, such minimal QoS guarantee can be important to ensure a basic level of user satisfaction.

5. CONCLUSION

FLoD demonstrates the possibility of progressive 3D content streaming in a P2P network. In this paper, we point out some improvements that can be made to FLoD's original design, by proposing bandwidth allocation channels that reduce the request latency, and a Bandwidth-Aware Peer Selection (BAPS) that improves the source limitation due to getting data from only AOI neighbors. The experiments show that with limited AOI neighbors as content sources, when object density increases, source insufficiency and request jam can occur. Such insufficiency, however, can be solved with BAPS and the adoption of Peer Lists. To make high capacity users contribute more, we use BitTorrent's Tit-for-Tat peer selection strategy to determine which peers can form connection channels. Future improvements to BAPS include using a more distributed approach (instead of relying on server) to maintain the Peer List, as this will bring workloads to server that would increase with user size and create a scalability bottleneck in the system.

6. REFERENCES

[1] A. Andre, S. Saito, and M. Nakajima. Adaptive 3d content for multi-platform on-line games. In *Proc. Cyberworlds*, 2007.

[2] A. Bharambe, C. Herley, and V. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanisms. In *Proc. INFOCOM*, 2006.

[3] J. Botev et al. The hyperverse: concepts for a federated and torrent-based '3d web'. *IJAMC*, 2(4):331–350, 2008.

[4] W. Cheng et al. An analytical model for progressive mesh streaming. In *Proc. ACM Multimedia*, 2007.

[5] C.-H. Chu, Y.-H. Chan, and P. Wu. 3d streaming based on multi-lod models for networked collaborative design. *Computers in Industry*, 59(9):863–872, 2008.

[6] B. Cohen. Incentives build robustness in bittorrent. In *Proc. Economics of Peer-to-Peer Systems*, 2003.

[7] S.-Y. Hu et al. Flod: A framework for peer-to-peer 3d streaming. In *Proc. INFOCOM*, 2008.

[8] S.-Y. Hu, J.-R. Jiang, and B.-Y. Chen. Peer-to-peer 3d streaming. *IEEE Internet Computing*, (to appear).

[9] H. Liang et al. Texture in second life: Measurement and analysis. In *Proc. P2P-NVE*, 2008.

[10] N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *Proc. INFOCOM*, pages 1415–1423, 2007.

[11] S. Mondet et al. Streaming of plants in distributed virtual environments. In *Proc. Multimedia*, 2008.

[12] J. Royan, P. Gioia, R. Cavagna, and C. Bouville. Network-based visualization of 3d landscapes and city models. *IEEE CG&A*, 27(6):70–79, 2007.

[13] D. Schmalstieg and G. Schaufler. Adaptive 3d content for multi-platform on-line games. In *Proc. Virtual Reality Annual International Symposium*, 1997.

[14] S. Singhal and M. Zyda. *Networked Virtual Environments*. ACM Press, 1999.

[15] W. L. Sung, S. Y. Hu, and J. R. Jiang. Selection strategies for peer-to-peer 3d streaming. In *Proc. NOSSDAV*, 2008.

[16] X. Susu et al. Coolstreaming: Design, theory, and practice. multimedia. *IEEE TMM*, 9(8), 2007.