# Bandwidth-Aware Peer-to-Peer 3D Streaming


## Chien-Hao Chien, Shun-Yun Hu, Jehn-Ruey Jiang* and Chuan-Wei Cheng

Department of Computer Science and Information Engineering,
National Central University,
Taiwan, ROC
E-mail: chienhao1@gmail.com
E-mail: syhu@csie.ncu.edu.tw
E-mail: jrjiang@csie.ncu.edu.tw
E-mail: 985202091@cc.ncu.edu.tw
*Corresponding author



**Abstract:** Peer-to-Peer streaming support for 3D content (i.e., P2P 3D streaming) has recently been proposed to provide affordable and real-time virtual environment (VE) content delivery. 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 enhances FLoD (Flowing Level-of-Details), a peer-to-peer 3D streaming scheme, by avoiding request contention and peer overloading as object and user densities increase, thus improving both bandwidth utilization and system scalability. We compare BAPS with FLoD's strategies that select from only peers within the area of interest (AOI) as data sources and do not consider bandwidth capacity. 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.



**Keywords:** P2P; peer-to-peer; virtual environments; 3D streaming; peer selection.



**Biographical notes:** Chien-Hao Chien received his Master Degree in Computer Science in 2009 from National Central University, Taiwan. His research interests include BitTorrent and peer-to-peer systems.

Shun-Yun Hu is currently a postdoctoral fellow at Academia Sinica, Taiwan. He received his PhD in Computer Science in 2009 from National Central University, Taiwan. His main research interests are networked virtual environments and peer-to-peer systems. He started the SourceForge project VAST (http://vast.sourceforge.net) in 2005, to provide open source libraries for creating scalable peer-to-peer-based virtual environments.

Jehn-Ruey Jiang received his PhD in Computer Science from National Tsing-Hua University, Taiwan, in 1995. He is currently a professor in the Department of Computer Science and Information Engineering,

National Central University, Taiwan, and co-leads Adaptive Computing and Networking (ACN) Laboratory. He was Guest Editor of International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC) and Journal of Information Science and Engineering (JISE). His research interests include distributed algorithms, peer-to-peer networked virtual environments, mobile ad hoc networks and wireless sensor networks.

Chuan-Wei Cheng is currently a Master student in the Department of Computer Science and Information Engineering at National Central University, Taiwan. His current research interest is peer-to-peer systems.

## 1 Introduction

Networked Virtual Environments (NVEs) are computer simulations that combine networked communications and 3D graphics techniques to provide immersive and responsive virtual interactions. Users in an NVE can assume virtual representatives (or *avatars*) to interact with each other concurrently. The early military simulator SIMNET (Singhal and Zyda, 1999), 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 (Mondet et al., 2008; Cheng et al., 2007; Andre et al., 2007; Chu et al., 2008) 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 download to complete. For example, *Second Life* delivers terabytes of user-generated content with their 3D streaming technology (Hu et al., 2008). 3D streaming is similar to media streaming (Magharei and Rejaie, 2007), where users can 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, video 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, different viewing angles or distances thus would 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 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 (Hu et al., 2008; Royan et al., 2007; Botev et al., 2008) have been proposed to support 3D streaming. P2P networks are characterized by the design where every user (or peer) 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, total amount of network bandwidth or computing resources will increase as user size scales, improving the system's scalability.

Recent proposals in P2P-based 3D streaming (e.g., Hu et al., 2008; Royan et al., 2007; Botev et al., 2008) show the benefits of using the P2P architecture to reduce server load. However, request contention and overloading can occur for some nodes (Sung et al., 2008), 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 on the basis of FLoD (Flowing Level-of-Details) (Hu et al., 2008, 2010), a peer-to-peer 3D streaming scheme, to avoid request contention and overloading. BAPS was first reported in (Chien et al., 2009). 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 *Tit-for-Tat* concept from BitTorrent[3] (BT) (Cohen, 2003) 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 the original FLoD to show its advantages.

The rest of the paper is organized as follows. Section 2 describes some related work. Section 3 defines our problem and describes the expected results; Section 4 proposes our peer selection and piece selection methods; Section 5 presents the results of the experiments; and conclusions will be given in Section 6.

## 2 Related Work

In this section, we introduce some related work, including 3D streaming, P2P data sharing and P2P 3D streaming.

### *2.1 3D Streaming*

With the prevalence of 3D applications, the number of 3D objects accessible via networks gradually increases. 3D objects are currently distributed by two strategies: *pre-installation* and *3d streaming*. Below we introduce both of the strategies.

The pre-installation strategy is widely used in MMOGs. In such a strategy, objects are obtained from a full pre-installation via a DVD or a network download.

However, downloading and installing the entire content to local storage might take a long time. To take the most popular MMOG, World of Warcraft (WoW), for an example, downloading all its 3D objects of 8GB takes about 10 hours with a network link of 2Mbps download bandwidth.

Unlike pre-installation, 3D Streaming (Mondet et al., 2008; Cheng et al., 2007; Andre et al., 2007; Chu et al., 2008) allows users to download only a small portion of 3D data for rendering a given scene before navigation. For example, Second Life delivers terabytes of user-generated content with 3D streaming technology (Hu et al., 2008). In general, 3D streaming can be achieved by two schemes: *object streaming* and *scene streaming*, which are introduced below.

In object streaming, the data of a 3D object are fragmented into one *base piece* and many *refinement pieces* before transmission. After downloading the base piece, the user can immediately have a coarse view of the object. As time goes by, refinement pieces are downloaded one by one and the user can have finer and finer views of the object. The *progressive meshes* scheme (Hoppe, 1997) is one of the most famous object streaming schemes. In the scheme, 3D objects are represented by numerous triangular polygons. The more polygons that represent an object, the finer the view of the object. Certainly, the finest representation of a 3D object has the largest number of polygonal meshes, consuming most bandwidth to transmit. In order to effectively transmit 3D object mesh data, Hoppe proposed an *edge collapse* procedure to reduce the number of lines or meshes of 3D objects by merging adjacent 3D points. By repeatedly executing the edge collapse procedure on an object, we can derive a very coarse version of the object, which has only a small number of meshes. And by recording the series of the collapsed edges and merged points, which are called *refinements data*, we can retain all meshes by recovering the original edges and points. Users can download the coarsest version of meshes as the base piece of the 3D object for fast startup display, while gradually downloading the series of refinements to have finer rendering of the object.

QSplat (Rusinkiewicz and Levoy, 2001) is another scheme of object streaming. It uses a multi-level tree of bounding spheres to represent objects to handle the visibility culling, level-of-detail control and rendering. Each node in the tree structure is called a *splat*. It represents a spherical range and records the sphere center position, sphere radius, and normal vectors and colors of objects contained in the sphere. Spheres in a layer can be surrounded by another larger sphere of the upper layer, while a leaf node of the bottom layer corresponds to a vertex of the original mesh. The rendering of an object is achieved by traversing the tree in a breadth first manner from the root with some culling rules. In the hierarchical structure of the tree, if some node is not visible for the viewer, it and its child nodes can be culled to increase rendering speed. When deeper nodes are rendered, the rendering quality becomes better. When all leaf nodes are visited, the complete details of the object are rendered. In this way, QSplat can serve as a 3D object streaming scheme.

### *2.2 P2P Data Sharing*

Data sharing is one of the most important applications in peer-to-peer networks. Below we introduce the well-known peer-to-peer file (or data) sharing protocol, BitTorrent (BT, Cohen, 2003), from which BAPS borrows ideas. In BT protocol[4],

a user (or peer) can share a file by first creating a small file called a *torrent* containing metadata about the shared file and about the *tracker*, the computer that coordinates the file distribution. Users that want to download the file must first obtain a torrent file for it, and then connect to the specified tracker. The tracker records all users downloading the file and tells them where to download the pieces of the file. Users can then contact with each other to exchange the list of downloaded pieces, and then send requests to download the desired pieces.

BT utilizes some mechanisms, such as *Tit-for-Tat* (TFT) and *Local-Rarest-First* (LRF) to improve file-sharing efficacy. To avoid overwhelming a peer, BT lays a limit on the maximum number (5 by default) of uploading connections. Each peer $p$ will check periodically (every 10 seconds by default) if a connecting peer $q$ that is downloading data from $p$ provides a corresponding bandwidth for $p$ to download data. If so, $p$ is *unchoking* with respect to $q$ and continues providing data to $q$. If not so, $p$ becomes choking with respect to $q$ and does not send data to $q$ (while holding $q$ in its waiting queue). This is the so-called TFT mechanism. On the other hand, a peer always sends request to first download the rarest piece among all connecting peers to keep the completeness of the file. This is the so-called LRF scheme. To make the newly joining peer can fast obtain the first file piece, a peer adopt the *Optimistic Unchoking Policy* to periodically (every 30 seconds by default) to arbitrarily pick up an extra peer to provide it data.

### *2.3 P2P 3D Streaming*

In this subsection, we introduce FLoD, on which the proposed scheme BAPS is based. FLoD is a peer-to-peer 3D streaming scheme that allows users to download 3D scene data from other users and to navigate the scene without a full download of the scene. As shown in Figure 1, FLoD divides the whole virtual world into fix-sized, indexed regions, where small squares and the star stand for avatars, '+' symbols stand for object centers, and the circle stands for the AOI of the avatar represented by the star. Each region has an associated *scene description* recording the properties, such as the IDs, positions, orientations, scales and piece information, of all objects within it. When a user enters the virtual world, it first acquires the scene descriptions of all regions that are covered by its AOI. The descriptions may be obtained from the server or from the user's *AOI neighbors*, users whose avatars are within the user's AOI. Afterwards, the user applies the *piece selection* strategy to determine the download priorities of pieces of objects within the AOI according to the object's importance, visibility, dependency, and distance to the user. To make users navigate the scene as soon as possible, base pieces always have the highest priorities.

A user in FLoD always tries to download object pieces from AOI neighbors first. The user sends queries to all AOI neighbors and sends a request to an arbitrary neighbor that replies positively for downloading a desired piece. If the downloading request is not granted, the user sends a request to another positively responding neighbor. If no neighbor responds positively, the user then sends a request to the server for downloading the piece. The query-reply-request-download handshaking is simple and can avoid redundancy of piece downloading. However, the queries consume a lot of bandwidth. Furthermore, the user needs to wait for the replies to the queries and the responses to the requests, which consumes much

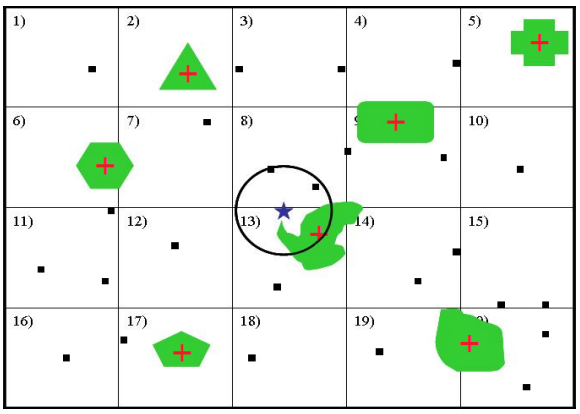



**Figure 1** Schematic of a Virtual Environment Divided into Fixed-Sized Cells

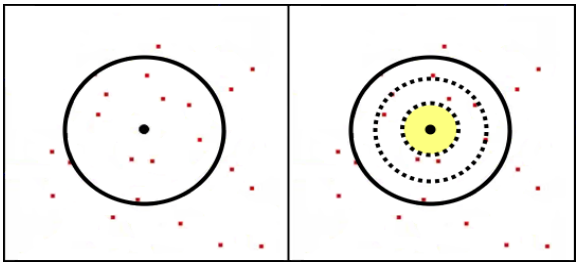


**Figure 2** Original Request Area of FLoD (left) and Multi-level Request (right)

time. The paper in (Sung et al., 2008) proposed that peers proactively exchange with AOI neighbors the list of available pieces. The exchange of available-piece lists is performed incrementally, and thus can be performed efficiently. To be more precise, a user sends to a newly connected AOI neighbor the whole list of available pieces, and it just sends out the list of additional pieces that it obtains afterwards. The paper also proposed that users send requests for downloading desired pieces in a multi-level manner. As shown in Figure 2, instead of sending requests to AOI neighbors within the whole AOI, the user first sends requests to AOI neighbors in the most inner circle of AOI. If the requests are not granted, then the user sends requests to the second inner circle of AOI, and so on. In this way, it is unlikely that a user gets requests from different users at the same time. Hence, the probability of request contention becomes lower and the download requests can be granted faster.

## 3 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 (Liang et al., 2008), 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 (Hu et al., 2008, 2010), 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. 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 causes 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 (Sung et al., 2008), users of low upload bandwidth can thus be in a request jam (i.e., receiving requests beyond capacities), whereas users with high upload bandwidth are idling. 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., 2008) propose to solve the above problems by reserving a list of past AOI neighbors and using a multilevel 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:

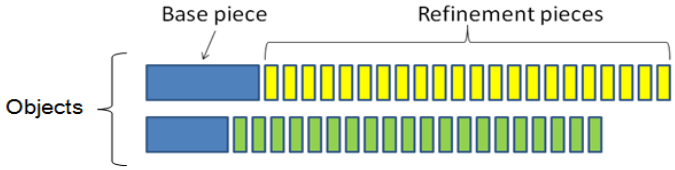



**Figure 3** Data Structure of 3D Streaming Object

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 (Cohen, 2003) 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.

## 4 Design of BAPS

### *4.1 Assumptions and Basic Ideas*

Based on the ideas of FLoD (Hu et al., 2008, 2010) and the work of (Sung et al., 2008), 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 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 (Schmalstieg and Schaufler, 1997). As shown in Figure 3, 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, then the object will appear 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 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. Each channel is constantly monitored to make sure that it is occupied. If channels sit idle for a system-defined time, the channel is recycled for other users. 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.

### *4.2 BAPS Algorithm*

We now describe BAPS in more details according to the main stages in P2P 3D streaming (Hu et al., 2010):

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. A new Peer List is obtained each time a scene is entered; however, for a given scene the list is not updated by the user unless the sources are too few. AOI neighbors, on the other hand, is updated continuously from the P2P overlay, and may reflect the sources more timely.

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 (Sung et al., 2008). We adopt the proactive push-based distribution of such meta states on piece availability (as proposed in Bharambe et al., 2006; Xie et al., 2007) 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 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 (Hu et al., 2008). 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.

**Table 1** Simulation Parameters

| | |
|---|---|
| World dimension (units) | $1000 \times 1000$ |
| Cell size (units) | $100 \times 100$ |
| 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 |

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* Magharei and Rejaie, 2007). To improve such scenario, when a requester asks for a connection channel, if the provider is fully loaded, connection preference is given to peers who have contributed more content, using a *Tit-for-Tat* policy (Cohen, 2003). By Tit-for-Tat, we mean that if a remote peer has provided adequate responses (i.e., upload) to the provider recently, then preference will be given to this remote peer to establish a connection channel. Note that the contribution is not accumulated but rather recent contribution within a certain time period, so both the remote peer's capacity and content availability are considered in such an approach. When connections are fully loaded, those with lower transmissions will be suspended, so that higher contributing peers (which may more likely be high capacity peers) can obtain content faster and serve sooner.

## 5 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 (Hu et al., 2008), where users, represented as nodes on a 2D plane, are simulated to move for a certain number of *time-steps* under a clustering movement pattern (i.e., nodes would move near certain *hotspots* with a high probability, Hu et al., 2008). One experiment case is run 10 times for obtaining average simulation results. Each node needs to obtain the scene description of the cells that its AOI covers, before

requesting the objects located in each cell. Please 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, different number of nodes are created (e.g., from 50 to 500) 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 99% of the initial AOI objects are downloaded. This assures that users are equipped with some content to exchange with others before their movements start. We can thus focus on the steady state behaviors during data distribution. The user bandwidth allocation is set as in (Bharambe et al., 2006) (see Table 2) to simulate a real environment, so that we may examine the performance of different algorithms under a realistic bandwidth distribution. Note that for the comparisons below, the same bandwidth limits are applied for both the original FLoD scheme and BAPS for evaluation.

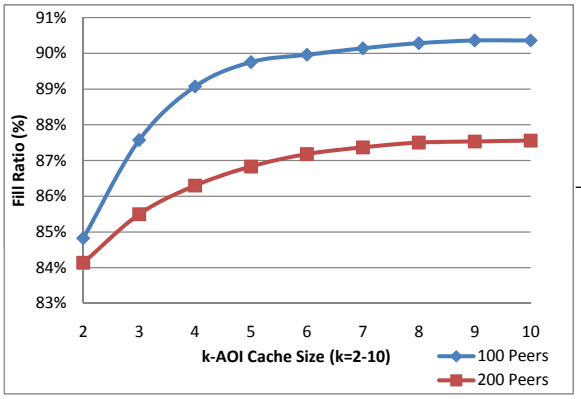
To simplify the simulation and focus on the effects caused by different numbers of peers and objects, we assume peers have an unlimited cache size. The assumption is justified by the FLoD designers' observation that cache sizes over a certain critical value would not affect the actual performance much, as long as a minimal amount of cache is provided. We have observed BAPS has the similar property. For example, Figures 4 and 5 show that BAPS has similar fill ratios and base piece latencies (two major performance metrics which will be defined in the next subsection) when peers have a cache size that can accommodate over 8 times the number of objects expected to appear in the AOI for the cases of 100 and 200 peers with 300 objects. A peer is said to have a $k$-AOI cache size if its cache size equals to $k * (TotalObjectSize/WorldArea) * AOIArea$. Therefore, when peers have an over 8-AOI (i.e., 14.67 MB) cache size, the system performance is pretty good for the above-mentioned cases. A peer stores all object data into cache for efficient object display and quick response to object download request. When cache is full, objects are replaced in the *farthest-object-first* manner. To further simplify our comparisons, there is no direct comparison made with another FLoD enhancement (Sung et al., 2008). The main reason is that of the two main improvements in (Sung et al., 2008), the idea to actively exchange content availability information is already incorporated in BAPS; while the benefits of multi-level AOI requests is not significant as compared with random request in the original FLoD (see Sung et al., 2008). We thus note that our work complements the work of Sung et al. to provide additional enhancements.

### *5.1 Metrics*

The following metrics are used for performance evaluation:

**Server Request Ratio (SRR):** Data pieces are requested from both peers and the server, but when a peer provider cannot be found and 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 SRR means 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




| Peers | AOI-sized | STD* |
|---|---|---|
| 100 | 2 | 2.16 |
| | 3 | 2.72 |
| | 4 | 3.31 |
| | 5 | 3.24 |
| | 6 | 3.25 |
| | 7 | 1.87 |
| | 8 | 3.04 |
| | 9 | 3.01 |
| | 10 | 3.01 |
| 200 | 2 | 3.14 |
| | 3 | 1.92 |
| | 4 | 2.21 |
| | 5 | 3.3 |
| | 6 | 2.44 |
| | 7 | 2.38 |
| | 8 | 2.12 |
| | 9 | 2.23 |
| | 10 | 2.33 |

* STD: stands for standard deviation

**Figure 4** Average Fill Ratio for Different Cache Sizes (BAPS)

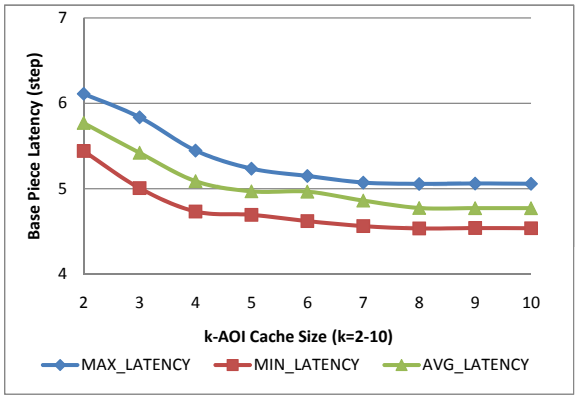
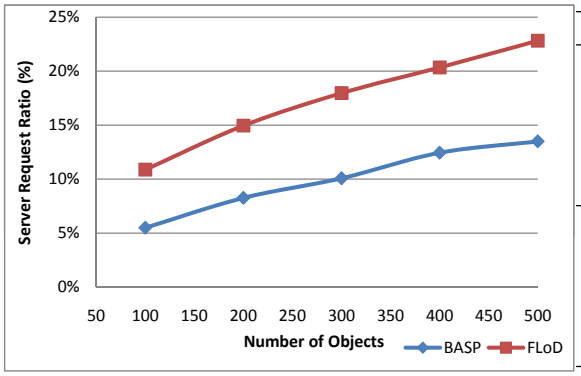



**Figure 5** Average Base Latency for Different Cache Sizes (BAPS)

(i.e., within AOI) content (in size). A higher ratio means a more sophisticated 3D scene and thus a better visual quality.

**Base-Piece Fill Ratio:** This metric is defined to be the ratio of the number of *displayable objects* over the total number of objects within AOI, where displayable objects stand for the objects whose base pieces have been downloaded. A higher ratio means a scene displaying more objects, which in turn leads to a better coarse view of the scene.




| Methods | Objects | Max% | Min% | STD |
|---|---|---|---|---|
| BAPS | 100 | 79.01 | 0 | 8.59 |
| | 200 | 76.0 | 1.0 | 9.91 |
| | 300 | 75.04 | 2.02 | 10.92 |
| | 400 | 74.01 | 4.06 | 12.92 |
| | 500 | 73.04 | 5.0 | 13.13 |
| FLoD | 100 | 100 | 0 | 13.13 |
| | 200 | 100 | 0 | 14.89 |
| | 300 | 100 | 3.01 | 14.89 |
| | 400 | 100 | 6.1 | 15.32 |
| | 500 | 100 | 6.1 | 15.66 |

**Figure 6** Average Server Request Ratio (SRR) for Different Numbers of Objects

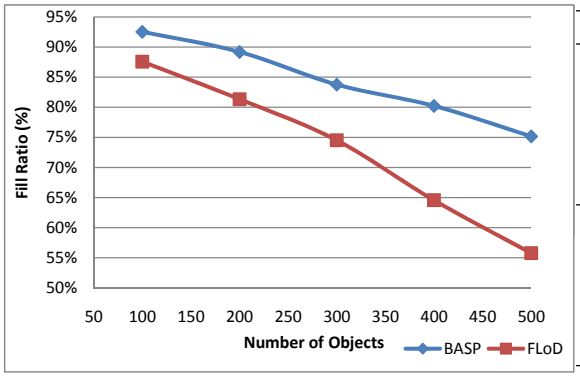
**Request Latency:** The delay between sending a piece request to acquiring the piece shows the efficiency of request serving and whether there is a request jam. Here we focus on the latency for the first (base) piece (i.e., *base latency*), as it shows how quickly a user may start navigation in a scene. Note that while the *completion latency* (Hu et al., 2008) (i.e., when an object is fully downloaded) is also important, but because we simulate constantly moving nodes, the completion latency may not be measured for all objects.

### *5.2 Performance Analysis*

In the following simulations, we use two 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 sizes to evaluate the system's scalability. Finally, we evaluate the streaming quality with time-series in how fill ratio changes.

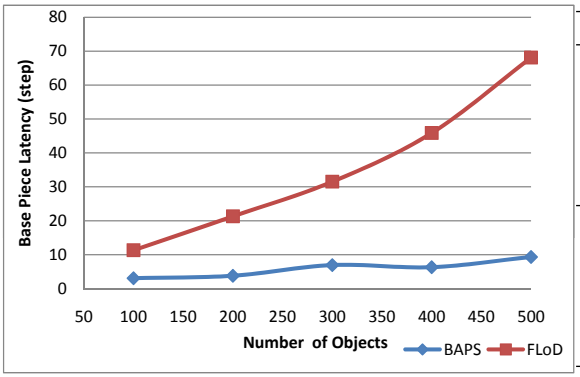
Bandwidth utilization: In a P2P network, data pieces are obtained from both server and peers. When users cannot find appropriate sources in the P2P network, the request is shifted to server. So if the SRR is high, then the P2P network may be under-utilized. By maintaining the same user size (100 peers) and movement paths while varying the object size (from 100 up to 500), we test the performance under different workloads. $M$ objects are randomly placed in the VE, whose sizes are $D_i|i = 1, 2, 3, \ldots, M$. 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 300 objects with an average size of 200KB exist, there will be a total of 60 MB of content, and the average AOI content size is 1.88MB.

As shown in Figure 6, when the content sources are extended from AOI neighbors to Peer List neighbors, under our strategy and the same number of objects, the 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 server loading; when there are 500 objects, the reduction rate becomes 66.2%.




| Methods | Objects | STD |
|---|---|---|
| BAPS | 100 | 12.31 |
| | 200 | 15.58 |
| | 300 | 17.29 |
| | 400 | 18.47 |
| | 500 | 19.36 |
| FLoD | 100 | 20.08 |
| | 200 | 24.63 |
| | 300 | 26.17 |
| | 400 | 24.94 |
| | 500 | 23.92 |

**Figure 7** Average Fill Ratio for Different Numbers of Objects



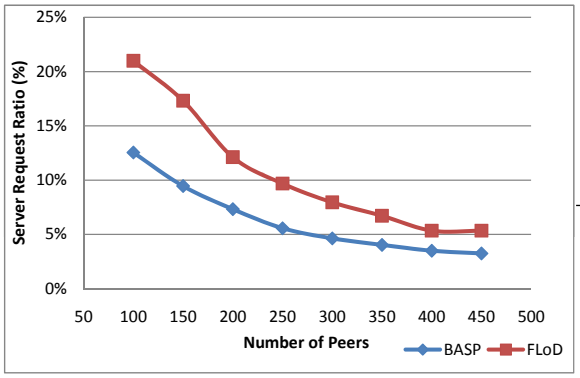

| Methods | Objects | Max | Min | STD |
|---|---|---|---|---|
| BAPS | 100 | 14.35 | 1 | 0.8 |
| | 200 | 11.58 | 1 | 0.92 |
| | 300 | 11.73 | 1 | 1.31 |
| | 400 | 21.22 | 1 | 2.26 |
| | 500 | 21.78 | 1 | 2.75 |
| FLoD | 100 | 160.2 | 1 | 26.59 |
| | 200 | 311.54 | 1 | 75.05 |
| | 300 | 440.85 | 1 | 121.95 |
| | 400 | 593.11 | 1 | 179.07 |
| | 500 | 740.38 | 1 | 235.69 |

**Figure 8** Base Latency for Different Numbers of Objects

The server loading is efficiently reduced with 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 Lists take 4.3% in the entire transmission, and the proportion increases with the number of objects. This shows that its cost is acceptable, but also indicates that as the object quantity grows, AOI neighbors will not provide sufficiently, and requests for Peer Lists would increase.

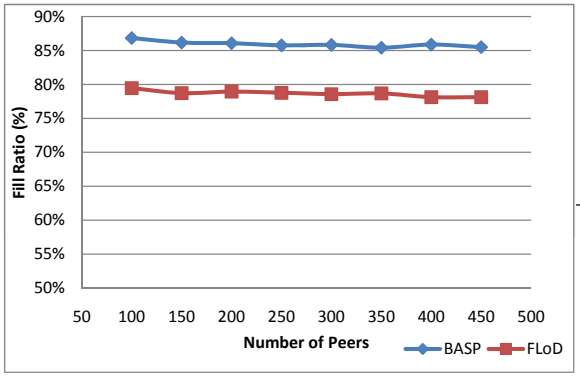
As for the fill ratio, Figure 7 shows that the fill ratio reduction in FLoD is higher than BAPS. When there are 100 objects, the fill ratio in FLoD is 87.5%, and when the object size increases to 500, the fill ratio is reduced to 55.7% with a reduction rate of 31.8%. So as object density increases, bandwidth 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 8 shows the latency for getting the base piece (i.e., *the base latency*, Hu et al., 2008), 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




| Methods | Objects | Max | Min | STD |
|---|---|---|---|---|
| BAPS | 100 | 80.08 | 3.06 | 12.4 |
| | 150 | 78.08 | 3.03 | 10.96 |
| | 200 | 79.04 | 2.01 | 9.77 |
| | 250 | 65.05 | 1.05 | 7.57 |
| | 300 | 64.06 | 1.03 | 7.23 |
| | 350 | 73.08 | 0.02 | 7.47 |
| | 400 | 100 | 0.04 | 8.97 |
| | 450 | 100 | 0.02 | 8.86 |
| FLoD | 100 | 79.02 | 0 | 16.97 |
| | 150 | 81.05 | 0 | 14.46 |
| | 200 | 79.09 | 0 | 12.07 |
| | 250 | 66.1 | 0 | 9.62 |
| | 300 | 75.01 | 0 | 9.65 |
| | 350 | 76.1 | 0 | 8.96 |
| | 400 | 85.02 | 0 | 8.83 |
| | 450 | 85.02 | 0 | 8.83 |

**Figure 9** Average Server Request Ratio (SRR) for Different Numbers of Peers

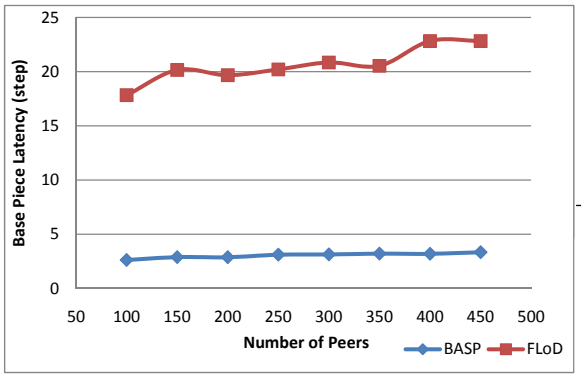



| Methods | Objects | STD |
|---|---|---|
| BAPS | 100 | 14.89 |
| | 150 | 15.5 |
| | 200 | 16.24 |
| | 250 | 16.33 |
| | 300 | 16.68 |
| | 350 | 16.96 |
| | 400 | 17.04 |
| | 450 | 16.95 |
| FLoD | 100 | 23.22 |
| | 150 | 24.69 |
| | 200 | 25.94 |
| | 250 | 26.22 |
| | 300 | 26.86 |
| | 350 | 27.24 |
| | 400 | 28.04 |
| | 450 | 28.04 |

**Figure 10** Average Fill Ratio for Different Numbers of Peers

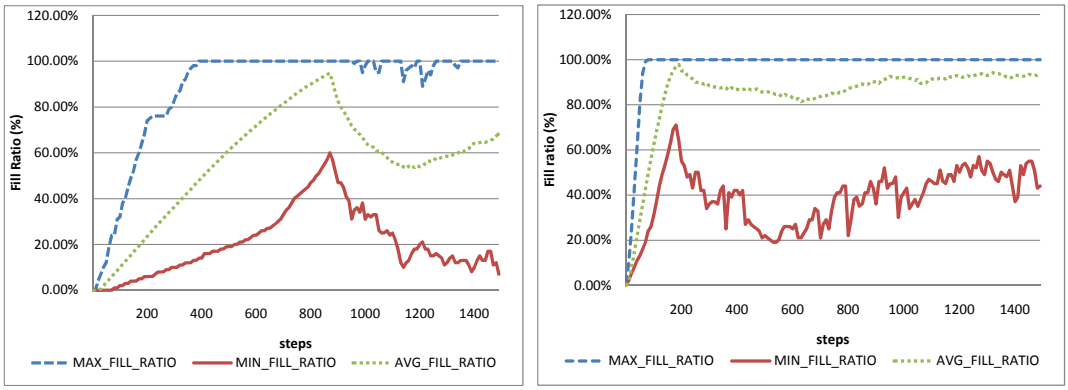
to all peers, even those with low bandwidth. With BAPS, not only base piece download is prioritized, bandwidth is also reserved for base piece requests to ensure that users can obtain them faster.

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 9 shows the percentage of data obtained from the server in FLoD and BAPS. When




| Methods | Objects | Max | Min | STD |
|---|---|---|---|---|
| BAPS | 100 | 11.28 | 1 | 0.94 |
| | 150 | 21.28 | 1 | 1.85 |
| | 200 | 20.94 | 1 | 1.8 |
| | 250 | 20.95 | 1 | 1.84 |
| | 300 | 21.58 | 1 | 1.89 |
| | 350 | 31.2 | 1 | 2.97 |
| | 400 | 30.55 | 1 | 2.92 |
| | 450 | 30 | 1 | 2.88 |
| FLoD | 100 | 340 | 1 | 86.22 |
| | 150 | 401.31 | 1 | 107.64 |
| | 200 | 421.08 | 1 | 114.29 |
| | 250 | 418.47 | 1 | 113.63 |
| | 300 | 440.1 | 1 | 121.42 |
| | 350 | 435.32 | 1 | 120.78 |
| | 400 | 455.692 | 1 | 127.43 |
| | 450 | 455.69 | 1 | 127.43 |

**Figure 11** Average Base Latency for Different Numbers of Peers




(a) Fill Ratio Time-Series for FLoD (b) Fill Ratio Time-Series for BAPS

**Figure 12** Fill Ratio Time-Series (fixed at 100 peers)

user size increases, FLoD and BAPS both reduces server loading. But for BAPS the server request ratio is even lower. Figure 10 shows the average fill ratio after 1500 time-steps. The more limited content sources produce the lower fill ratio for FLoD. Figure 11 shows the comparisons in base latency. We can easily observe the difference in performance from Figure 10 and Figure 11. 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 12(a) and Figure 12(b) show the time-series of fill ratio under 100 peers and 300 objects. Here we see the general trend of the fill

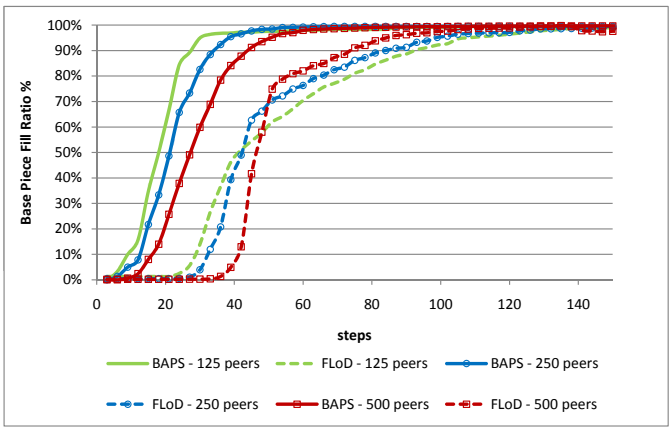



**Figure 13** Base Piece Fill Ratio Time-series with 250 Objects

ratio first increases for some period, then decreases as peers start to move after the overall fill ratio has reached 99%. This is to ensure that peers have enough initial content to share. The fill ratio restores with time, because when peers obtain more pieces, sharing also becomes more effective. 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 connects with peers with more contribution and suspends non-performing channels. The minimum fill ratio indicates the worst streaming quality a user may experience. An important observation from Figure 12(b) is that by reducing request jam, a more stable streaming is achieved even for worse case scenarios. For commercial providers, such minimal QoS guarantee can be important to ensure a basic level of user satisfaction.

Figure 13 and Figure 14 show the time-series of base-piece fill ratio under the cases of 125, 250 and 500 peers, and the cases of 250 and 500 objects. Note that we just depict the first 150 steps in the figures because all the curves reach 100% for almost all the cases after 150 steps. We see that more objects or more peers lead to worse ratios. This is because in the first 100 steps, many pieces are still provided by the server, so more time is needed to serve more peers with more objects. We can also see that BAPS has better base-piece fill ratios than FLoD. This means BAPS can render more base pieces of the scene than FLoD. Therefore, users will feel that newly-encountered objects are rendered faster in BAPS than in FLoD.

## 6 Conclusion

FLoD demonstrates the possibility of progressive 3D content streaming in a P2P network. In this paper, we propose a Bandwidth-Aware Peer Selection (BAPS) method that reduces the request latency by having bandwidth allocation channels and more content sources beyond AOI neighbors. We perform simulation

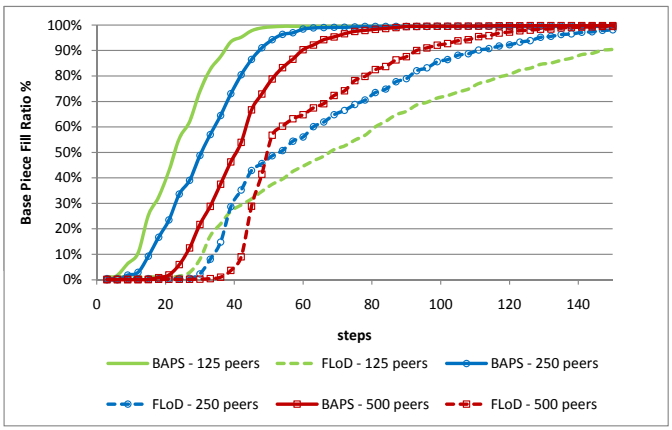



**Figure 14** Base Piece Fill Ratio Time-series with 500 Objects

experiments to evaluate BAPS and FLoD in terms of the server request ratio, fill ratio, base-piece fill ratio, and request latency. The simulation results show that when content sources are limited, source insufficiency and request jam can occur. Such insufficiency is mitigated by BAPS with the adoption of peer lists provided by the server. To make high capacity users contribute more, we use BitTorrent's Tit-for-Tat peer selection strategy to determine the peers to form connection channels. Future improvements to BAPS include a more distributed approach to maintain the Peer List (instead of relying on server), as maintaining Peer List can be a potential bottleneck for the server. We would also like to evaluate our strategies using more realistic user-traces or behaviors. Empirical comparisons of BAPS and real systems (e.g., Second Life) are also considered as future work. The comparisons are challenging since BAPS needs accurate end-to-end available bandwidth determination for implementation.

## References

A. Andre, S. Saito, and M. Nakajima. (2007) 'Adaptive 3d content for multiplatform on-line games', *Proceedings of the 2007 International Conference on Cyberworlds*, Washington, USA, pp.194–201.

A. Bharambe, C. Herley, and V. Padmanabhan. (2006) 'Analyzing and improving a bittorrent networks performance mechanisms', *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, pp.1–12.

J. Botev, A. Höhfeld, H. Schloss, I. Scholtes, and M. Esch. (2008) 'The hyperverse: concepts for a federated and torrent-based "3d web"', *International Journal of Advanced Media and Communication (IJAMC)*, Vol. 2, No. 4, pp.331–350.

W. Cheng and W.T. Ooi and S. Mondet and R. Grigoras and G. Morin. (2007) 'An analytical model for progressive mesh streaming', *Proceedings of the 15th international conference on Multimedia (ACM Multimedia)*, New York, USA, pp.737–746.

C.-H. Chien and S.-Y. Hu and J.-R. Jiang (2009) 'Bandwidth-Aware Peer-to-Peer 3D Streaming', *Proceedings of Network and Systems Support for Games (NetGames 2009)*, Paris, pp.1–6.

C.-H. Chu, Y.-H. Chan, and P. Wu. (2008) '3d streaming based on multilod models for networked collaborative design', *Computers in Industry*, Vol. 59, No. 9, pp.863–872.

B. Cohen. (2003) 'Incentives build robustness in bittorrent', *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems.*

S.-Y. Hu and T.-H. Huangy and S.-C. Chang and W.-L. Sung and J.-R. Jiang and B.-Y. Chen. (2008) 'Flod: A framework for peer-to-peer 3d streaming', *Proceedings of the 27th Conference on Computer Communications (INFOCOM)*, Phoenix, AZ, pp.1373–1381.

S.-Y. Hu and J.-R. Jiang and B.-Y. Chen. (2010) 'Peer-to-peer 3d streaming', *IEEE Internet Computing*, Vol. 14, No. 2, pp.54–61.

H. Hoppe. (1997) 'View-dependent refinement of progressive meshes', *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, New York, USA, pp.189–198.

H. Liang and M. Motani and W.T. Ooi. (2008) 'Texture in second life: Measurement and analysis', *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, Washington, USA, pp.823–828.

N. Magharei and R. Rejaie. (2007) 'Prime: Peer-to-peer receiver-driven mesh-based streaming', *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, Anchorage, Alaska, pp.1415–1423.

S. Mondet and W. Cheng and G. Morin and R. Grigoras and F. Boudon and W.T. Ooi. (2008) 'Streaming of plants in distributed virtual environments', *Proceedings of the 16th ACM international conference on Multimedia (ACM Multimedia)*, New York, USA, pp.1–10.

J. Royan, P. Gioia, R. Cavagna, and C. Bouville. (2007) 'Network-based visualization of 3d landscapes and city models', *IEEE Computer Graphics and Applications (IEEE CG&A)*, Vol. 27, No. 6, pp.70–79.

S. Rusinkiewicz and M. Levoy. (2001) 'Streaming QSplat: a viewer for networked visualization of large, dense models', *Proceedings of the 2001 symposium on Interactive 3D graphics*, New York, USA, pp.63–68.

D. Schmalstieg and G. Schaufler. (1997) 'Smooth levels of detail', *Proceedings of the 1997 Virtual Reality Annual International Symposium (VRAIS '97)*, Washington, USA, pp.12–19.

S. Singhal and M. Zyda. (1999) *Networked Virtual Environments: Design and Implementation. ACM Press.*

W. L. Sung, S. Y. Hu, and J. R. Jiang. (2008) 'Selection strategies for peer-to-peer 3d streaming', *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, New York, USA, pp.15–20.

S. Xie and B. Li and G.Y. Keung and X. Zhang. (2007) 'Coolstreaming: Design, theory, and practice. multimedia', *IEEE Transactions on Multimedia (IEEE TMM)*, Vol. 9, No. 8, pp.1661–1671.

## Note

[1]http://www.worldofwarcraft.com
[2]http://www.secondlife.com
[3]http://bittorrent.com
[4]http://en.wikipedia.org/wiki/BitTorrent_(protocol)