VMesh: Distributed Segment Storage for Peer-to-Peer Interactive Video Streaming

W.-P. Ken Yiu, Xing Jin, S.-H. Gary Chan

***Abstract*—Provisioning random access functions in peer-to-peer on-demand video streaming is challenging, due to not only the asynchronous user interactivity but also the unpredictability of group dynamics. In this paper, we propose VMesh, a distributed peer-to-peer video-on-demand (VoD) streaming scheme which efficiently supports random seeking functionality. In VMesh, videos are divided into segments and stored at peers' local storage in a distributed manner. An overlay mesh is built upon peers to support random forward/backward seek, pause and restart during playback. Our scheme takes advantage of the large aggregate storage capacity of peers to improve the segment supply so as to support efficient interactive commands in a scalable manner. Unlike previous work based on "cache-and-relay" mechanism, in our scheme, user interactivity such as random seeking performed by a peer does not break the connections between it and its children, and hence our scheme achieves better playback continuity. Through simulation, we show that our system achieves low startup and seeking latency under random user interactivity and peer join/leave which is a crucial requirement in an interactive VoD system.**

***Index Terms*—Peer-to-peer, media streaming, random seeking, distributed storage, locality-aware, popularity-based, distributed consensus.**

I. Introduction

With the penetration of broadband Internet access into households, there has been an increasing interest in media streaming services. Video-on-demand (VoD) is one of such services where movies are delivered to desktops of distributed users with low delay and free interactivity (in terms of pause, jump forward/backward, etc.). However, providing VoD with traditional client-server architecture where each client is allocated a dedicated stream from the server is not scalable to large number of clients. This is mainly due to heavy server load and limited network bandwidth at the server side. Though IP multicast may be used as a scalable solution for media streaming, providing such services worldwide is still challenging due to the lack of widely deployed multicast-capable networks and dedicated proxy servers [1], [2]. Recently, peer-to-peer (P2P) technologies have been proven as a scalable solution to many applications, e.g., multicasting and file sharing among distributed users [3], [4]. In this paper, we propose a novel P2P technique to provide interactive VoD service.

In P2P systems, cooperative peers self-organize into an overlay network via unicast tunnels.[1] Each peer (called overlay node) in the overlay network acts as an application-layer proxy, caching and relaying data for other peers. In addition, by sharing their resources such as storage and network bandwidth, the storage and streaming capacity of the whole system is greatly amplified as compared with traditional client-server architecture. Recent research shows that it is feasible to support large-scale media streaming in the Internet using P2P approach [5]–[13]. P2P approach has been shown to be feasible for on-demand media streaming, unfortunately, its support to user interactivity is still a challenge because a user may jump forward, backward, pause and resume its playback anytime. This means that the parents (or "suppliers") of a user may need to be changed quite frequently. Therefore, an efficient algorithm for switching to appropriate parents is needed to support user interactivity.

Though P2P file swarming systems like BitTorrent may be used to download the whole media objects before playback, this introduces long startup delay [4]. Though there has been much work on providing on-demand video service in P2P networks (i.e., low startup delay) [5], [8], [10]–[14], only few tackle *user interactivity* issue which we focus in this paper. The previous work uses "cache-and-relay" paradigm, in which a peer caches what it has recently played out for a period of time before discarding it, and uses the cached content to serve others. As opposed to the previous work, our scheme uses static local *storage* instead of sliding window *buffering* to help handle user interactivity efficiently and to reduce the complexity. In "cache-and-relay" systems, peers rely on the cached content of their parents. If a parent jumps to another playpoint in the video, it starts to receive media data which is not interested by its children. As a result, all its children would be abandoned since the cache of their parent can no longer supply useful data to them. The situation becomes severe if the system uses tree-based overlay for streaming because the descendants of the children are also affected. The advantage of our scheme is that any interactive action (e.g. random seeking) of a peer does not stop its children from continuing to receive its stored data. The peers, on the other hand, connect to new parents for each segment. The main difference between our scheme and the previous schemes is that, during normal playback, the time for a peer to switch its parents is predictable. Therefore, the peer can start caching the next segment when it nearly finishes playing out the current

Manuscript was received on March 1, 2007; revised on August 1, 2007. This work was supported, in part, by the Research Grant Council of the Hong Kong Special Administrative Region (HKUST611107) and Hong Kong Innovation and Technology Commission (GHP/045/05).

The authors are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: {kenyiu, csvenus, gchan}@cse.ust.hk).

Digital Object Identifier 10.1109/JSAC.2007.071210.

[1]In this paper, we use "client", "user" and "peer" interchangeably.

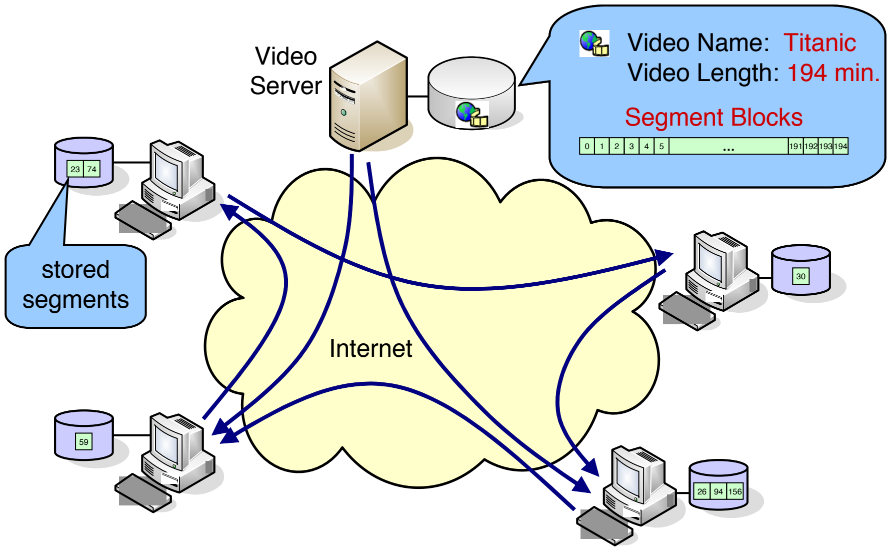


Fig. 1. An overview of VMesh architecture. Each VMesh peer stores some segments for serving others.

segment. This is also an advantage of our scheme over the previous schemes.

Most of the existing work on P2P-based media streaming systems have made an implicit assumption that a user who joins a streaming session would play the media from the beginning to the end. However, as suggested in [9], based on large amount of user viewing logs, most users performed random seeking (i.e., jumping) frequently. From the statistics, we also found that the jump distances are usually small. This is reasonable because users would usually skip boring scenes by jumping a bit forward or review some exciting scenes by jumping a bit backward. It would be beneficial if the system allows the users to jump to close scenes efficiently.

Based on the above observations, we propose a novel P2P architecture called *VMesh* to support interactive VoD service over the Internet. VMesh utilizes the large aggregate storage capacity of peers to amplify the supply of video segments to achieve user scalability. Figure 1 depicts the architecture of VMesh system. In VMesh, videos are divided into smaller segments (identified by segment IDs) and distributed to peers over the network. An overlay mesh is built among the peers to support playback and interactive functionalities. Each peer stores a number of video segments at its local storage such as hard disk, depending on its storage capacity. It keeps a list of the peers who have the previous and the next video segments. Following the list, its children can quickly find the peers with the next requested segments. Furthermore, a peer also keeps a list of peers storing the same segment for load balancing purpose. If a node is loaded, it redirects some of its children to other peers on its list. In order to provide failure-tolerant streaming service, a client connects to multiple parents who have stored the segment of interest so that they stream the video ***in parallel*** and ***collaboratively***. The parents could be searched in the network using a distributed hash table (DHT) system such as Chord or Pastry with the key comprising video ID and segment ID [15], [16].

Some people may argue that DHT is unstable under peer dynamics. Recently, there have been works on improving DHT's efficiency and resilience [17], [18] and how DHT systems could be applied in practical systems [19], [20]. Our scheme assumes using an efficient and resilient DHT system as a subsystem, how to improve the efficiency of searching results from DHT is out of the paper scope. Our scheme does not pose any limitation on which DHT systems to be used. Apart from that, network address translation (NAT) is another major concern on the feasibility of P2P streaming systems. Since there are many end-hosts currently situated behind NAT routers, it is difficult to connect them from outside and stream media to them. This issue can be addressed by applying hole punching technique proposed in [21]. Once a host behind NAT joins the system, it registers both of its private and public endpoints (i.e., the two addresses used by the end-host inside and outside of the NAT network) at a rendezvous server (RS) which is dedicated for storing peers' endpoints. When a peer A wants to connect to another peer B behind NAT, A can obtain B's endpoints from the RS and try to connect to the private and public endpoints of B. As mentioned in [21], this method works for most of the NAT routers available nowadays. To get rid of the centralized RS, one may also make the registered entries distributed over a DHT. The routing tables in DHT nodes then contain both private and public endpoints of a node behind NAT, and the DHT nodes imitate the RS by forwarding both endpoints to A and B.

In this paper, we propose VMesh architecture to provision VoD service in a pure P2P network. In VMesh, we propose a locality-aware segment location algorithm for improving the streaming efficiency and reducing server stress. In addition, we also propose a popularity-based segment storage scheme for better load-balancing and relieving server stress. Our extensive simulation shows that VMesh has the following desirable properties:

- Scalability — the system is simple and scalable to large number of users with low server bandwidth requirement. It is completely decentralized, without the need of a server to organize overlay nodes.
- Efficiency — users can start playing the media with low

delay as it is based on streaming without file downloading.
- Failure-resilience — the system is robust to peer dynamics, node and link failures to offer continuous streaming.
- Interactivity support — users can interact with the media at any time.

This paper is organized as follows. We first briefly mention the related work in Section II. We then give a detailed description of our system and the popularity-based storage scheme in Section III and Section IV. Next, we present our experimental results in Section V. Finally, we conclude the paper in Section VI.

II. Related Work

Research on providing on-demand media streaming using IP multicast, such as patching [1], [2], periodic broadcasting [22], [23], stream merging [24], etc., has been done. Those works take advantage of the efficient data dissemination using IP multicast and repeat the media in multicast channels. Nevertheless, over the past decade, the deployment of IP multicast-capable networks has remained very limited. This leads researchers to tackle the problem in the application layer. Based on this, VMesh is a pure P2P architecture and does not assume the existence of IP multicast.

Recently, researchers use peer-to-peer (P2P) approach to provide VoD services. P2Cast [25] uses a P2P approach to cooperatively stream video using patching techniques. Similar to traditional patching schemes, while a new peer receives a base stream from an overlay tree formed by the peers in the same session, it also requests another peer from the same session as its patch server for sending it the patch stream. However, hosts in P2P networks usually do not have enough bandwidth to serve a client, the single parent approach used by P2Cast is unable to cope with the bandwidth limitation and fluctuation problem and the parent departure problem in P2P networks. Moreover, overlay tree's bandwidth is guaranteed to be monotonically decreasing from the root to the leaves, streaming quality becomes worse at the leaf nodes. CoopNet [14] delivers multiple description codes (MDC) of a video stream over multiple overlay trees to solve the departure problem. Each peer joins multiple trees, and hence has more than one parent. If one of the parents departs, the peer can still receive and decode other descriptions from the remaining parents with a little quality degradation. PROMISE [26] peers also receive stream from several parents. However, each of the parents is required to have the full copy of the requested media, which is rarely available in practice. P2VoD [13] organizes each video session tree into layers. Peer departure is efficiently handled by finding another parent only in the upper layer without the involvement of the source. With each peer having a fixed bound on the amount of cached most recent stream, each new client can quickly join the system by either joining the lowest layer of the tree or creating a new layer in the tree. However, P2VoD assumes that all clients play the stream from the beginning which is not true in reality. It also does not provide any efficient mechanism for client to random seek another playpoint in the video which is an essential functionality in a VoD service. Cui *et al.* developed a temporal dependency model among end-hosts [11]. Based on this model, a media distribution tree is found by the central server to minimize the overall transmission cost. Nevertheless, every time a peer leaves or seeks another position in the video, the server needs to recompute the whole tree. In our design of VMesh, we consider provisioning the important function - random seeking, by distributed storage and search schemes. Each peer in VMesh stores a few segments of the media data which are independent of what it is playing, hence, user interactivity of the peers does not affect its children in the delivery network. In [10], a hybrid approach is introduced. Upon a tree overlay, a gossip-based data exchange mechanism is added to withstand the unreliability of peers. However, all the parent-children relationships and gossip partners are assigned by a central server. Therefore, the central server needs to keep track of all the users. This centralized approach is not preferred in P2P systems because a large amount of control traffic would congest the network near by the central server. PROP [27] also uses distributed storage approach for providing VoD service. But, the scheme relies on proxy servers and global information for cache replacement. In our system, we use distributed algorithms for segment location, storage and replacement. We do not assume any availability of global information. In addition, we also build an overlay mesh called VMesh to minimize the searching overhead.

For searching good parents in a P2P network, Zhou and Liu proposed to build an overlay AVL tree for efficient searching [28]. Though searching parents is achieved using a distributed AVL tree, the network locations of peers are not taken into consideration when assigning parent-children relationship and gossip partnership. Careless assignment may lead to transmission of the stream to and fro some bottleneck links many times, increasing the link stress of the network. Also, transmission from distant parents may experience higher probability of packet loss. Chi and Zhang proposed buffer-assisted search (BAS) in [29] which builds a binary search tree based on peers' buffered content range. As peers' buffer coverage may overlap, many peers who have content overlapped by others can be dismissed from the search tree. Therefore, the scheme keeps the search tree size small and hence the searching process becomes quick. Again, the scheme may also assign distant parents to a new client which leads to inefficient data delivery. In VMesh, we use distributed hash table (DHT) for searching parents. With each peer registers its own stored segments on the DHT [15], a new client can perform DHT search to find out which peers are storing the required content. In addition, the network locations of peers are embedded in the search key so as to allow the DHT search to find out close peers for efficient streaming.

Data scheduling among multiple parents is another important issue in P2P streaming system. Zhang *et al.* proposed a deadline-oriented greedy algorithm for assigning packet delivery from multiple parents [6]. In [30], the problem is modeled as a classical min-cost network flow problem and proposed a distributed scheduling algorithm to maximize the system throughput. The algorithm solves a linear program (LP) which maximizes the cost in terms of packet rarity and emergency, with a set of constraints on available inbound, outbound and end-to-end bandwidth. In this paper, we concentrate on segment storage and location, and provisioning random seeking functionality. This topic is out of the paper scope. In this paper, we simply apply round-robin scheduling among parents. However, our system can be adapted to apply any other advanced techniques for data scheduling.

In [31], we proposed the use of distributed storage to allow efficient support of user interactivity in P2P VoD system. In the paper, we extend the work by adding a distributed algorithm to balance the supply and demand of video segments under non-uniform segment popularity distribution. The algorithm makes popular segments to be duplicated more and match their demands, so as to relieve requirements on the server bandwidth. Besides, since the distributions of popular segments are denser, it is more likely that a client could find close parents for those segments, and hence receive video of better quality. We also evaluate the performance of the proposed algorithm through simulation.

III. Scheme Description

In this section, we first give an overview of VMesh (Section III-A), followed by our locality-aware segment location scheme for locating distributed stored segments (Section III-B). Then, we introduce our scheme on mesh construction (Section III-C) and its feedback-based maintenance (Section III-D). Finally, we discuss how to choose the size of segment (Section III-E) and how to extend our work in order to support fast-forward operation (III-F).

A. VMesh Overview

A video server stores the videos for user access. Each video is divided into N segments, each of them are identifiable by its video ID and segment ID. For example, the segment is 5 minutes long with a video bit rate of 1 Mbps, therefore, it is of size 36 MBytes. Depending on the capacity of its local storage, each peer stores a number of segments randomly chosen from the N segments of the video. These peers are referred as *storage peers*. In this way, there are multiple copies of each video segment in the network. When a client wants to play a segment, it first looks for the supplying peers of that segment in the P2P network, then sends requests to those peers for the service. Those supplying peers with enough outgoing bandwidth would serve the requesting peer. If there is no supplying peer, the requesting peer requests the media server for the target segment as the last resort.

Since video segments are distributed among peers, VMesh utilizes *distributed hash table* (DHT) to locate these segments. DHT is a structured overlay constructed among peers. It works like a traditional hash table, that is, given a hashed key, it returns the corresponding object or its location. The difference is that the table entries are not located in the same place but distributed among the peers in the network. With a proper routing mechanism, the DHT supports primitive functions such as: *put(id, object)*, *get(id)* and *delete(id)*, where *id* is the object's identifier, as in a traditional hash table. In VMesh, peers use a DHT built among the peers to bootstrap a new video streaming session. A new client searches for its first segment of interest in the DHT network and starts playing the video when the requested data arrives and fills up its buffer. A

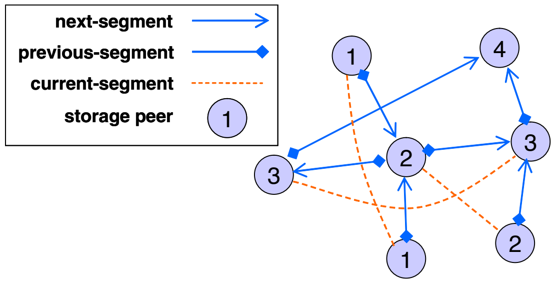


Fig. 2. In VMesh, storage peers hold various video segments and at the same time, they keep a list of peers who possess the next / previous / current segments for random seeking and load balancing purposes.

client continues to request the next segment when the current one is nearly finished. At the mean time, a peer uses its spare bandwidth to randomly download and store video segment(s) in its local storage. The segment would then be used to stream to another peer of interest. Each stored segment should be registered in the DHT network by the storage peer, so as to allow other peers to locate the segment. This can be done using *put(id, object)*.

It is feasible for a client to continuously search for each segment using *get(id)* in the DHT network. However, it is likely that, a client who is accessing the current segment also wants to access the next segment, i.e., their access probabilities are highly correlated. Therefore, it is beneficial to cache the links to the next segment. In VMesh, peers also form an overlay mesh among themselves in order to save messaging overhead for the DHT search and shorten the segment location latency. Each peer keeps a list of pointers (i.e., the IP addresses) pointing to some peers which store the next video segment and the previous video segment. By using the pointer list, clients could request the current parents for the locations of next required segment. This does not require the client to go through the DHT query process. Besides, for the load balancing issue, each storage peer also keeps a list of pointers to some peers which are storing the same segment. When a storage peer is overloaded by its children, it can redirect some of them to other storage peers which are also able to serve them.

Figure 2 illustrates the idea of VMesh. In the figure, a circle represents a storage peer and the number inside represents the ID of the segment it holds. A storage peer holds a few video segments and keeps a list of peers who have the next, previous, or current segments for seek and load balancing purposes. The pointers are used to redirect users to the appropriate peers storing the next requested segments. In case of joining or jumping, a VoD client in our system searches for its parents using DHT and gets the stream from those parents. In the traditional "cache-and-relay" paradigm proposed in the previous work, a VoD client relies on the content resides in its parents' buffers [11], [13]. If a parent jumps to another position in the video, the peer needs to search for a new parent again. In contrast, the segment stored by a peer in VMesh would not be changed by any user interactive actions.

All peers register the search keys of their stored video segments in the DHT. A joining client performs the following procedures:

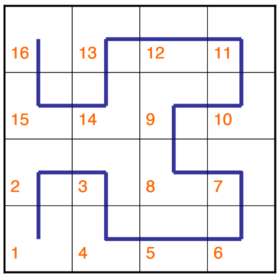


Fig. 3. An example of Hilbert curve mapping from two-dimensional space to one-dimensional space.

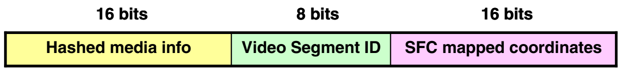
1) It searches for the first interested segment in the DHT network.
2) Upon receiving the list of peers from the DHT node, it contacts the returned list of peers to request for the segment. Close peers are preferred for transmission efficiency.[2]
3) Those contacted peers with enough upload bandwidth become the client's parents for sending the requested segment. Multiple parents are employed for fault tolerance purpose. Among the multiple parents, the client has to decide which blocks to be delivered by each parent. There are various scheduling algorithms to assign packets with multiple parents [6], [26], [30]. We consider simple round-robin scheduling in this paper.
4) When the client nearly finishes playing the current segment, it requests from its current parents their lists of peers holding the next segment. It then contacts the peers in the returned list for continuous playback.
5) If the client wants to jump to another video position, it uses either the mesh pointers or the DHT network to find the target segment according to Algorithm 1. The objective of the algorithm is to reduce the number of overlay hops for the search process. Let L_{seek} be the seeking distance (in minutes) from the current playpoint position, S the segment size, and M the size of DHT search key space. If the new position is not far away from the current one, it simply follows its forward/backward pointers in the video mesh to contact the new parents. Otherwise, it triggers another DHT search for the segment corresponding to the new position.

Algorithm 1 SeekSegment
```
1: if $(L_{seek}/S) \leq \log M$ then
2:     Follow mesh links to find the target segment;
3: else
4:     Search the target segment using DHT;
5: end if
```

B. Locality-Aware Segment Location

The mutual network distance between parent and children is a crucial factor for efficient streaming. If the parent-child relationships are casually formed, much network resources would be wasted due to inefficient routing and increase in link stress. To address this problem, VMesh takes the network locations of the peers into consideration while locating supplying peers for a client.

[2]Close parents can be selected from the list by measuring their round-trip times using simple `ping`.



16 bits	8 bits	16 bits
Hashed media info	Video Segment ID	SFC mapped coordinates

Fig. 4. The 40-bit DHT search key consists of three parts: hashed media information, video segment ID and space filling curve mapped network coordinates of the peer.

The network location of a peer may be obtained using a network coordinate system such as GNP and Vivaldi [32], [33]. Such system gathers ping measurements among peers and landmarks, and returns multi-dimensional coordinates in an Euclidean space. In order to search for parents with close network locations, we put this locality information into the DHT search keys of the segments registered by the peers. Since most DHTs in structured overlays use a one-dimensional space for keys as opposed to the multi-dimensional coordinates, we need a mapping from the multi-dimensional coordinate space $\Re^d$ to the one-dimensional DHT search key space $\Re$. We apply a space filling curve (SFC), such as Hilbert curve, for such mapping because SFC is *locality preserving* (i.e., if two points in the multi-dimensional space are close, the distance between their corresponding mapped one-dimensional points is also short.) [34]. Figure 3 shows how Hilbert curve maps a two-dimensional space for a 4x4 2-D coordinates to a one-dimensional space labelled from 1 to 16.

With the mapped coordinates, each peer constructs its 40-bit DHT key consisting of media information and its segment ID as well.[3] As shown in Figure 4, the DHT key is constructed by combining the fields in the order of importance. The most important field is the media information, followed by the video segment ID, and then the SFC-mapped coordinates. The media information is hashed to obtain a *video ID*, so that each video can be assigned to an ID uniformly distributed in the key space for the purpose of balancing DHT load among peers. Each peer registers its own key for its stored video segment(s) in the DHT. At the same time, it searches for its parents using DHT search keys constructed by segment ID and *its own mapped coordinates*. Most DHTs can be modified to reply queries with multiple peers whose keys are *numerically closest to the search key*. Since the peer's own mapped coordinates are used to construct the search key, multiple parents closest to the requesting peer are returned. The peer can then connect to them and request for the segment.

Figure 5 illustrates how a new client receives the requested video stream in VMesh. Firstly, it searches for the segment of its interest using DHT. For system fault tolerance of DHT, DHT nodes may duplicate its table entries to the k neighboring nodes in the key space. Then, the request is forwarded to the k closest neighboring nodes. Based on our design of DHT search key, the k closest available storage peers may reply and begin to serve the requesting client.

[3]The length of the DHT key can be extended to accommodate larger systems. Currently, the 16-bit hashed media information can already accommodate up to about 65,000 media objects in the system.

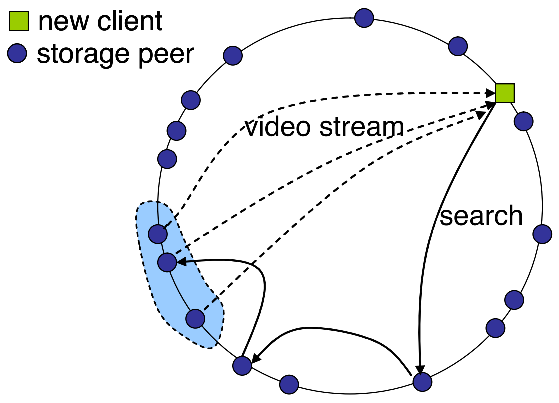


Fig. 5. A new client has to search for the first interested video segment to bootstrap a new video streaming session.

C. Video Mesh Construction

In our system, it is possible for a client to continuously search for the next required segment using the DHT network, as proposed in [27]. However, a client who is playing the current segment is likely to play the subsequent segment too. That is, the access probabilities of two adjacent segments should not be independent but highly correlated. Therefore, to further reduce the latency as well as the routing message overhead introduced by the DHT search process, our system additionally builds an overlay mesh among the peers, which links up the contiguous media segments with bi-directional pointers. That is, each storage peer keeps an IP address list of the storage peers who is holding the previous and the next segments. By following those pointers, a client does not always need to search for every segment over the DHT network during its playback.

Upon entering the system and DHT search, a new client may start viewing its video. Using its residual bandwidth, it also downloads some random segments for storage. After the video segment is completely downloaded, the client registers its segment(s) in the DHT. As shown in Figure 2, a peer needs to keep three lists of peer pointers (i.e., peers' IP addresses): *next-segment-list*, *previous-segment-list* and *current-segment-list*. The lists could be easily obtained by searching for the previous / next / current segment IDs from other peers using the DHT built among them. The pointers in the lists are used to redirect a peer's children to some other peers during playback or for load-balancing purpose. In the case of normal playback, when a child nearly consumes the whole segment, it requests for some pointers in the *next-segment-list* from its current parents, so that it can get the subsequent segments from other peers. In case of random seeking a favorite scene in a movie, users may jump back and forth in the video. Short-distance jumps (say within 10 minutes), can be satisfied by both the *next-segment-list* and *previous-segment-list* from its current parents. The *current-segment-list* is used when the parent's load is too heavy. Requesting peers are redirected to other peers who keep the same video segment. As a result, these lists can reduce control messaging overhead by avoiding DHT searching for new parents by the client each time a segment is nearly used up. Instead, the storage peers search only once and keep the list.

D. Feedback-based Maintenance

Keeping all the pointers in the lists refreshed by frequent updates is not economical. We eliminate this kind of messaging overhead by employing a feedback-based mechanism for maintaining the pointers in the lists. Children of peers are responsible for checking the validity of the pointers sent by their parents. If the percentage of invalid or failed pointers is greater than a certain threshold t, the child reports the situation to its parent. Upon receiving such failure report, the parent needs to update the pointers in its list by searching for the corresponding segment using DHT. The advantages of this passive updating mechanism are twofold: 1) The storage peers need not keep track of all pointers in their lists, which may be very costly. 2) Some failed pointers may become valid again because the failure may be transient.

Each list only contains at most k pointers. A peer does not need to keep all the qualified parents (i.e., the peers who store the segments of interest) in its lists. Due to our design of the DHT search key, the lists should contain the qualified parents whose locations are close to the client. Therefore, children of a peer are likely redirected to close parents during playback or jumping.

E. Variation of Segment Size

In this proposed scheme, we do not fix the size of each segment. In fact, having very small segments and requiring each peer to switch parents very often would possibly introduce more control overhead on messaging. However, we consider that each peer has only a limited storage space for caching video data. For example, we cannot assume that a low-profile set-top box (STB) is capable of storing a whole movie in its local storage. Based on this limited storage consideration, it is a trade off between the requirement on the peers' local storage size and the messaging overhead caused by switching parents.

Therefore, it is an art to choose the most suitable segment size in the system. If we consider deploying the client software on a small low-profile STB, it is not preferable to put a requirement that the STB has much storage space for caching video data, which definitely would increase the STB's cost. At the other extreme, the performance of the whole system would become much better if each peer could store the whole video (or a very large segment) in its local storage, and its children are no longer required to switch parents. Moreover, it is absolutely possible for a peer to store more than one segment if it has more spare storage space.

To conclude, segment size should be chosen properly based on these considerations. In our simulation, we just set our segment size to be 5 minutes long of a video stream with bit rate of 1 Mbps, which is around 36 megabytes. It is reasonable for both set-top boxes and personal computers to have such a storage space.

F. Support of Fast-Forward Operation

There are a number of approaches to extend our system for supporting fast-forward operation. In this section, we discuss how to support a double-speed (2X) fast-forward (FF) operation. For other speeds, e.g., 4X, 8X, etc., the mechanism is the same.

One may consider to perform fast-forward operation solely by playing out the video stream at a faster speed by the client peer. That is, for a 2X FF operation, the client can either display video frames at a double frame rate or play only one frame (and skips one frame) for every two frames. However, since this approach consumes the bit stream at a double speed, the peer is also required to download the video stream at a double speed, which would definitely increase the download bandwidth required for a peer. Also, the transmission of skipped frames costs 100% overhead (and the overhead is even more for higher speed FF operation). Thus, this approach is not suitable for supporting FF operation in our system.

Therefore, we propose two approaches to support FF in VMesh:

- *Encode-on-demand at peers*: In this approach, when a peer performs a FF operation, it requests its parents to encode and deliver a video stream of the segment at a faster frame rate dynamically. This approach requires not only processing power from the peers for dynamic encoding, but also the synchronization of all encoding parameters. This would also complicate the client software implementation. Moreover, the client peer needs to switch its parents more frequently while it performs FF operation. For example, a peer plays a 5-minute segment at a double speed, it needs to switch to new parents after 2.5 minutes.
- *Distribution of pre-encoded frame-skipped version*: In this approach, the source provides encoded streams of the original version as well as the frame-skipped versions for various speeds. The frame-skipped versions are then distributed to the peers in the same way as the original. The speed of a version can be embedded into the media information part of the DHT key, so as to allow clients who perform FF operations to search the frame-skipped versions on-demand. This approach requires the source to pre-encode all versions, and the peers to store segments of versions other than the original. Therefore, some peer storage resources are contributed to store various speed versions.

Both of these two approaches are able to support FF operation. Applying which approach is a trade off between the overhead on peers' computation and the system storage.

IV. Popularity-based Segment Storage

Given a streaming media, the popularity of the segments are different if user interactivity like jumping is allowed. For instance, users who are viewing sports events usually jump to view the scoring moments. This makes the access rate of the segments non-uniform. Therefore, if storage nodes uniformly pick random segments to store, some nodes would have heavier load than others since some segments are accessed more frequently. Intuitively, the load-balancing in the system can be improved if the supply of each segment matches the demand of that segment. In this section, we first model the popularities of segments by considering user interactivity (Section IV-A). Then, we propose a mechanism to estimate the segment popularities in a distributed manner (Section IV-B). Based on the estimated popularities, we describe our distributed algorithm to determine which segments to be cached by the storage peers and how to perform cache replacement to adapt the supply of segments which meets the changing demand (Section IV-C).

A. Segment Popularities

In [27], segment popularities (i.e., the access probabilities) are assumed to follow a Zipf distribution, like the distribution of web objects' popularities in the Internet. If all the segments are ranked in the descending order of their popularities, the popularity of the ith segment, p_i, is expressed as

$$p_i = \frac{1/i^{\alpha}}{\sum_{n=1}^{N} 1/n^{\alpha}}$$

where N is the total number of segments, and α is a constant. In our study, we link up the user interactivity and the segment popularities, since the popularity of a segment depends on how frequent the segment is accessed by the users. We model the user interactivity as follows: A user starts to watch a video at a segment randomly selected. The user stays in normal playback mode for a random time period with an exponential distribution with mean T_{seq} seconds, then jumps to a random segment and returns to normal playback mode again. When the user finishes the last segment of the video, it loops back to the first segment. The process continues until the user leaves the system. The time period for the user staying in the system follows an exponential distribution with mean T_{life}.

For calculating segment popularities, we approximate the model as a discrete-time system. The continuous stream is divided into N segments, each of which is Δt seconds long. Let π_i be the state that a user is accessing segment i. Therefore, the time for a user staying in the system is discretized to $\lceil T_{life}/\Delta t \rceil$. With probability p_{seq} $(= e^{-\Delta t/T_{seq}})$, the user plays the media normally and sequentially access segment $(i+1)$. Thus, the average number of consecutive segments the user access in sequence is $1/(1-p_{seq})$ (for this geometric distribution). At the segment i, with probability $q_{i,j}$, the user jumps to another segment, j, such that

$$\sum_{i=1}^{N} q_{i,j} = 1 - p_{seq}.$$

The user interactivity model is a discrete-time Markov chain. We can see that the one-step transition probability $p_{i,j}$ from π_i to π_j for any i and j, $1 \leq i,j \leq N$,

$$p_{i,j} = \begin{cases} q_{i,j} + p_{seq} & \text{if } j = ((i+1) \mod N) + 1, \\ q_{i,j} & \text{otherwise.} \end{cases} \tag{1}$$

Given that a user starts with a randomly chosen segment, the initial access probabilities $p_i^{(0)}$ of segments are evenly distributed, i.e.,

$$p_i^{(0)} = \frac{1}{N}, \forall i \in N.$$

At discrete time slot n, the access probabilities $p_i^{(n)}$ change to some other values given by,

$$p_i^{(n)} = p_i^{(0)} \mathbf{P}^n$$

Suppose at the end of each time slot, the user still stays in the system with probability p_{stay} $(= e^{-\Delta t/T_{life}})$, the time period that the user stays in the system is geometrically distributed. The average time period is $1/(1 - p_{stay})$. Thus, the expected access probability for a segment i is,

$$\begin{aligned} p_i &= \sum_{m=0}^{\infty} p_i^{(m)} Pr[\text{the user stays in the system at time } m] \\ &= \sum_{m=0}^{\infty} p_i^{(m)} p_{stay}^m (1 - p_{stay}) \qquad (2) \end{aligned}$$

From the equations above, given the user interactivity (i.e., the one-step transition probabilities), we can calculate the popularity (i.e., the access probability) of each segment. Obviously, if all the transition probabilities are equal, all the final access probabilities are also equal. However, users usually jump to some positions where they can watch something they want, this makes the transition probability distribution non-uniform. Though we realize this fact, it is hard to obtain the one-step transition probabilities for calculating the popularities of the segments. In addition, the transition probabilities may change over time, we need an adaptive mechanism for estimating the popularities of the segments in a distributed manner.

B. Popularity Estimation by Distributed Consensus

In order to store segments according to their popularities, the first requirement is to obtain their popularities over the P2P network. To achieve this, all segment popularities can be gathered at a central server, and all nodes can reference to the server. However, this creates a single point of failure and heavy network load at the server. Therefore, we apply in our scheme a distributed averaging algorithm to obtain those popularity information in a decentralized manner.

Distributed consensus problem (or distributed averaging problem) has been studied in the control and sensing literature for obtaining the global average of all the values distributed over the network [35]. In order to calculate global averages, one can flood the whole network with all the values, or use synchronized message propagation over a structured overlay network (e.g. a spanning tree). These are natural methods for the network nodes to come up with a common value, but the former has very large messaging complexity, and the latter requires a structured overlay network and synchronized messaging among nodes. A practical and implementable algorithm was proposed in [36] for calculating averages over distributed nodes in a communication network. Also, it is proved that the algorithm allows the nodes to converge to a common value (within 1% of the average) under very general asynchronous timing assumptions. Therefore, we use the algorithm proposed in [36] in our system for popularity estimation.

In [36], each network node connects to k random neighbors and only exchange messages with those neighbors locally. Suppose node i has a local static value z_i, and the nodes wish to obtain the average value of all z_i over the network. Then, each node maintains a dynamic variable x_i (called *state*) which is initialized to z_i. Each node periodically communicates with its neighbors as the followings (as shown in Figure 6):

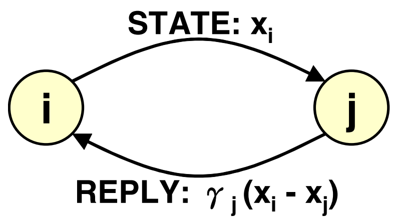


Fig. 6. The message passing in distributed averaging algorithm.

1) Node i sends a STATE message containing x_i's value to node j.
2) Upon the receipt of the STATE message, node j updates its local variable x_j to $x_j + \gamma_j(x_i - x_j)$ where γ_j $(0 < \gamma_j < 1)$ is a local stepsize parameter[4]. Also, it sends back node i a REPLY message containing $\gamma_j(x_i - x_j)$.
3) Node i receives the REPLY message and updates its local variable x_i to $x_i - \gamma_j(x_i - x_j)$.

The idea behind the algorithm is to conserve the sum of all the variables, and the pair of state variables become closer at each update. The algorithm is simple and it is proved that all x_i will converge to the average. In order to cope with peer dynamics, each node i also maintains an additional variable δ_{ij} associated with each neighbor j, which accumulates all the changes made due to that neighbor. Whenever node i detects the departure of its neighbor j, it can subtract δ_{ij} from x_i to conserve the total sum. The algorithm also works if z_i's value is time-varying. For any change Δz on z_i, the node also updates its state x_i by the same amount Δz. This ensures that the sum of all states are conserved.

We apply the above-mentioned algorithm to count the total number of accesses of each segment. Each peer in VMesh maintains a state a_i for each segment i, indicating its access to segment i. If the peer is downloading segment i, it sets $a_i = 1$, otherwise, it sets $a_i = 0$. In order to obtain the distribution of all segment popularities, a peer runs the averaging algorithm on those N states and maintains a set of variables b_i, $1 \le i \le N$, where N is the total number of segments in the video. Each b_i represents the global average number of current access for segment i. Each peer then computes the estimated popularity $\hat{p_i}$, $0 \le \hat{p_i} \le 1$ for segment i from its own set of averages as

$$\hat{p_i} = \frac{b_i}{\sum_{k=1}^{N} b_k},$$

and thus,

$$\sum_{i=1}^{N} \hat{p_i} = 1.$$

Besides the demand of the segments, we also need to know the supply of each segment so as to provide a better load-balancing between storage peers. By using the averaging algorithm, we also obtain the information of the segment supplies. Each peer maintains a vector $\vec{c} = [c_1 \ c_2 \ \ldots \ c_N]$, where c_i is set to 1 if the peer stores segment i, otherwise, c_i is set to zero. Each peer runs the averaging algorithm and obtains the average number of copies d_i for segment i in the

[4]The proposed algorithm still works even if nodes have different values of γ_j. Hence, the value of γ_j does not need to be synchronized among nodes.

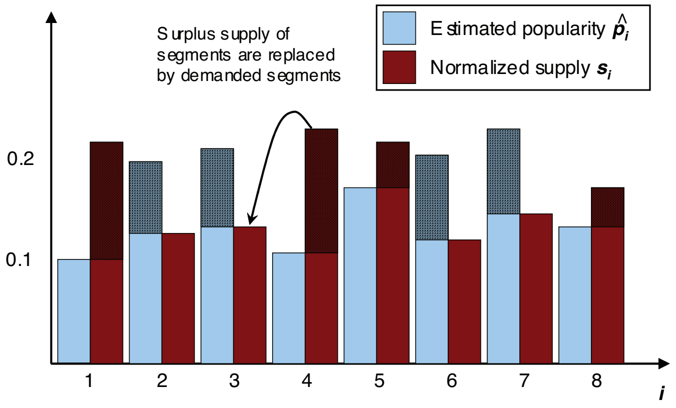


Fig. 7. Stored segment replacement probability is calculated using the difference between $\hat{p_i}$ and s_i ($N = 8$).

network. Similarly, each peer can then compute the normalized supplies s_i, $0 \le s_i \le 1$ for segment i from its own set of d_i's as

$$s_i = \frac{d_i}{\sum_{k=1}^{N} d_k},$$

such that,

$$\sum_{i=1}^{N} s_i = 1.$$

Based on those estimated segment popularities $\hat{p_i}$ and normalized supplies s_i, $1 \le i \le N$, each peer determines which segments to be cached or replaced by which segments. We will give the details on how to store segments according to the changing supplies and demands in the following section.

C. Adaptive Segment Caching

When a new peer joins the system, it requests for the segment popularities $\hat{p_i}$ and normalized supplies s_i, $1 \le i \le N$, from one of its neighbors. The peer then chooses the segments to cache with probability P_i^C to choose segment i,

$$P_i^C = \begin{cases} \frac{\hat{p_i} - s_i}{\sum_{\{k \mid \hat{p_k} > s_k\}} (\hat{p_k} - s_k)} & \text{if } \hat{p_i} > s_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Equation (3) captures the discrepancy between the supply and demand. If the demand is greater than the supply for segment i (i.e., $\hat{p_i} > s_i$), there is a need to cache more copies of that segment. P_i^C is the probability proportional to the need of segment i among the demanding segments. Hence, the larger the difference between supply and demand of a segment, the larger the probability for a new peer to cache that segment. After downloading the chosen segments, the peer sets those c_i's for the chosen segments to 1. The averaging algorithm will then propagate and update the corresponding d_i's throughout the network.

Periodically, each storage peer checks the distribution of $\hat{p_i}$ and s_i to see if there is any discrepancy between them. We define the discrepancy degree D as the mean square error (MSE) between the supplies and demands of all the N segments. That is,

$$D = \frac{\sum_{k=1}^{N} (\hat{p_k} - s_k)^2}{N} \tag{4}$$

If there is a large discrepancy (i.e., D is greater than a threshold thr_D), the peer determines whether it needs to replace its stored segments or not. For each of its stored segments, the peer decides whether or not to replace segment i with probability P_i^R,

$$P_i^R = \begin{cases} \frac{s_i - \hat{p_i}}{s_i} & \text{if } \hat{p_i} < s_i, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

If the peer decides to replace segment i, it then chooses to download another segment j to replace segment i with probability P_j^S,

$$P_j^S = \begin{cases} \frac{\hat{p_j} - s_j}{\sum_{\{k \mid \hat{p_k} > s_k\}} (\hat{p_k} - s_k)} & \text{if } \hat{p_j} > s_j, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Since we only replace segment i by the segments whose demand is greater than its supply, the probabilities to choose segments with supplies exceed demands are zero. For segment j with its demand exceeds its supply in Equation (6), the numerator represents the amount by which its demand exceeds its supply. And, the denominator is the sum of all the amounts for those segments. Hence, the probability to choose segment j for replacement is proportional to the amount by which its demand exceeds its supply among those demanded segments. Again, the peer sets those c_i's for the chosen segments to 1 after downloading them and let the averaging algorithm propagate and update the corresponding d_i's throughout the network.

V. Performance Evaluation

We have evaluated VMesh using packet-level event-driven simulation. We also studied the performance of embedding locality information into the DHT search key. We used Chord implementation for our DHT module [37]. For the distributed consensus protocol, each peer randomly connects to 5 peers and sends state messages to them periodically with the period of 2 seconds.[5] We set the local stepsize parameter γ to 0.5 for all peers. Each peer checks for the discrepancy between supply and demand periodically with the period of 10 minutes, and the threshold thr_D is set to 0.001. In Section V-A, we first describe how we model our network and user behaviors. We also introduce the performance metrics we used to evaluate the system performance. Finally, we present our experimental results in Section V-B.

In order to compare our scheme with traditional "cache-and-relay" approach, we implemented P2VoD system [13]. In P2VoD, peers are divided into generations according to their playpoints. Peers in the uppermost generations directly connected to the source and peers in an upper generation delivers stream to peers in its lower generation. Each peer caches the streaming data for a period and streams the cached data to its children. However, the system restricts its peers playing the video from the beginning. Therefore, we modify the system by allowing the peers to search for their parents starting from the root. In addition, the system also does not

[5]In order to further reduce message overhead for distributed consensus protocol, we use periodic messaging instead of the original message-on-update approach.

support random seeking, hence, we add this functionality by simply leaving and re-joining the peer to the system. In case of parent failure, a child would request data from the siblings of its parent. The cache size of each peer is also set to the size of one segment.

A. Data and Network Models

In our simulations, the length and bit rate of a movie are L seconds and R Mbps, respectively. We set $L = 7200$ and $R = 1$ in our simulations. Each segment is 5 minutes long and of size about 36 MBytes. Each peer owns local storage space with size of one video segment, thus, a peer can at most store one segment locally. Each peer is randomly attached to a router node. When a peer joins the system, it requests the beginning of the video. Meanwhile, each peer is assigned an uploading capacity of at most $2R$ Mbps, i.e., a peer can at most upload two streams to its neighbors. Group size, i.e. the total number of peers in a measurement session, varies from 100 to 3200. The underlying network topology is generated using GT-ITM [38]. The whole network consists of 4080 routers and more than 20000 links.

In order to further study the performance of our locality-aware segment location scheme, we model the link loss of our network topology according to the typical settings in [39]. A fraction f of the links were classified as "good" links and the rest as "bad". The loss rate for good links is picked uniformly at random in the $0 - 1\%$ range and that for bad links is picked in the $5 - 10\%$ range. Once each link is assigned a loss rate, packet loss events at each link follows a Bernoulli loss process, i.e., each packet traversing a link is dropped with a fixed probability determined by the loss rate of the link. In our simulation, we set f to 0.95.

One way to evaluate the performance of a streaming system under user behavior is to run the simulation based on real user traces. However, it is difficult to gather enough data for various settings of user behavior to perform such trace-driven experiments. Therefore, we generate users and their activities according to [40]. The user arrival process follows a Poisson distribution, i.e.,

$$p(x;\lambda t) = \frac{(\lambda t)^x}{x!}e^{-\lambda t}, x = 0, 1, 2, \ldots, t \geq 0,$$

with average inter-arrival time τ $(= 1/\lambda)$ seconds. The duration of each user session is exponential with mean T seconds. As the usual case in the Internet, we model all user departures as ungraceful leaving, i.e., the users leave the system without informing their neighbors nor clearing their DHT entries. This also provides a pessimistic evaluation on our system. Thus, the system performance would be better if there are users, on the other hand, leave the system gracefully. According to the statistics in [9], each user performs 6 to 7 random seeking in its session on average. We hence model each user to jump with exponential inter-jump period of one-seventh of T. A user may jump forward or backward and the jump distance is modeled as a Pareto distribution, i.e.,

$$p(x;k,x_m) = k\frac{x_m^k}{x^{k+1}}, x \geq x_m,$$

where x_m is a location parameter to control the lower bound of the jumping distance, and k is a shape parameter to control

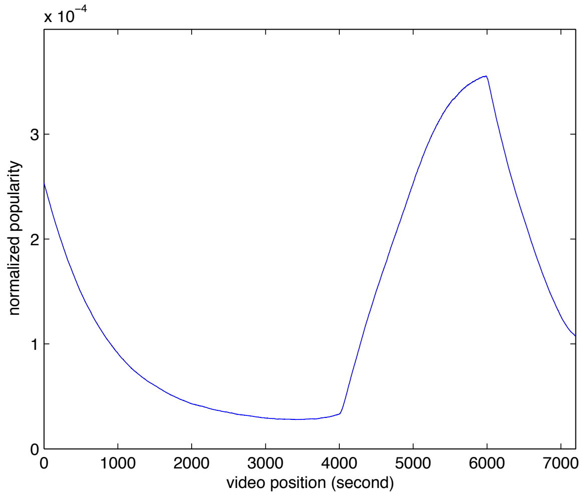


Fig. 8. Simulated popularity distribution of a video ($L = 7200, R = 1, T = L, \tau = 9.0$), with the average number of concurrent users $N = T/\tau = 800$. Given the same user interactivity model, our experiments show that the distributions are similar for other settings of T and τ.

the shape of the distribution (Higher value of k means the density shifts to x_m.). The probability distribution is bounded by the two ends of the video, and is normalized accordingly. In order to provide a non-uniform popularity distribution, we partition the video into three regions with various parameter settings for jumping:

- Region 1: The region ranges from the 0th second to the 2500th second. Inside the region, $x_m = 4000$ and $k = 100$.
- Region 2: The region ranges from the 4000th second to the 6000th second. Inside the region, $x_m = 10$ and $k = 500$.
- Region 3: The rest of the video belongs to this region, in which $x_m = 1000$ and $k = 1$.

Figure 8 shows the popularity distribution of a video we obtained after simulating the user behavior according to our model for 24 hours of simulation time.

We evaluate our system using the following performance metrics:

- **Server stress** – the outgoing bandwidth required at the media server to support the whole system. In our results shown below, we normalized the server stress by the bit rate of the video stream to indicate the number of streams required. The lower the server stress, the more scalable the system is.
- **Continuity index** – the percentage of media data received and played out at the client side by the playback deadline of the media data. The higher the continuity index, the higher the video quality received is.
- **Startup latency** – the time period starting from the instant that a client joins the system to the instant that the client starts playing out the video after buffering a few seconds of media data[6].
- **Jumping latency** – the time period starting from the

[6]The buffer size does affect the smoothness of the playback. In our simulation, we set it to be 3 seconds.

instant that a client issues a jump command to the instant that the client starts playing out at the seek position after buffering a few seconds of media data. Low jumping latency means that the system responds quickly to user jump commands.

B. Experimental Results

In order to study the performance of our schemes on segment location and distributed storage, we tested three variants of VMesh. The first version is the pure VMesh in which peers choose parents randomly regardless their network locations (i.e., without locality information in DHT search key). This scheme only applies random selection for segment storage. The scheme is denoted as **VMesh(RAND)**. The second one applies locality-aware segment location, and is denoted as **VMesh(LA)**. This scheme also performs random segment selection for local storage. The last one applies both locality-aware segment location and popularity-based segment storage schemes, and is denoted as **VMesh(LA-POP)**.

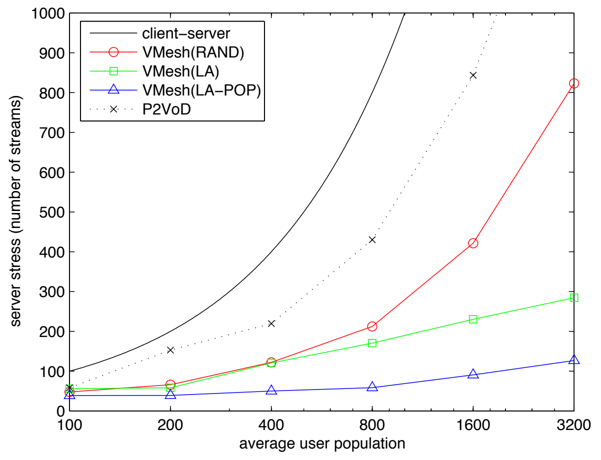
1) Server Stress Reduction: To illustrate the performance on saving server resources and bandwidth, we first present the effect on the server stress in terms of numbers of media streams as the average user population increases. Since our simulations start with no client and add clients into the system one by one according our user activity model, we change the parameter $\tau(= 1/\lambda)$ to adjust the average size of user population according to Little's law (i.e., $N = \lambda T$ where N is the average number of users in the system.). Figure 9(a) shows the server stress against the average user population. As the average user population increases exponentially, the server stresses for VMesh systems increase very slowly. In general, VMesh is highly scalable for large user population. Obviously, VMesh(LA) achieves lower server stress than VMesh(RAND). This is because a peer in VMesh(RAND) may request segments from some distant parents, which may experience higher packet loss rates and higher delays. The peer then requests the missing packets from the media server as the last resort. Therefore, this puts more streaming load on the server. On the other hand, locality-aware segment location scheme searches for close peers to become a client's parents, and those close peers can provide efficient streaming in terms of error rate and delay. Hence, VMesh(LA) can reduce server stress, in this case, by 20 - 60%. In addition to VMesh(LA), VMesh(LA-POP) matches supply and demand of each segment in the system, and thus relieves the server from handling requests for segments whose demands exceed their supplies. Since the peers keep joining and leaving the network, the computed averages using distributed consensus protocol also fluctuate. However, as the peers replace their stored segments according to their computed distribution, we found that the computed segment distribution is within 10% of the real one after around ten rounds of message exchanges. VMesh(LA-POP) can further reduce the server stress by 20 - 65% when the user population is large. Note that, VMesh(LA-POP) can save server resources only when the user population is large enough for the scheme to distribute enough segments among peers and match their supplies and demands.

For P2VoD, the server stress increases linearly with the average user population. The system organizes its peers in a tree-like structure, with upper-level peers watching the latter part of the video while lower-level peers watching the former part. Therefore, only a small portion of upper-level peers can contribute and stream the latter part of the video. However, according to our popularity distribution, the system fails to balance the supply and demand of video data. As a result, most peers directly request data from the server and this makes the server stress so high.

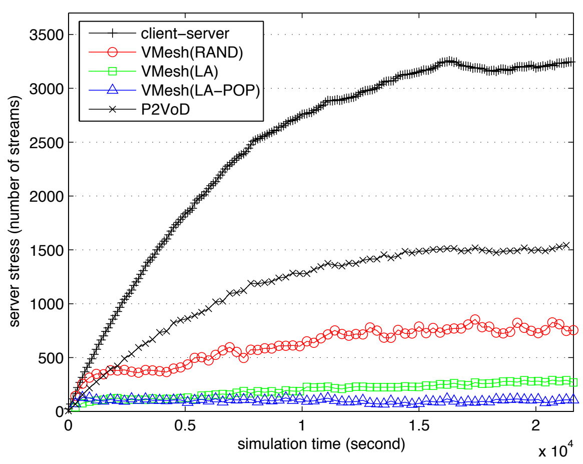
In Figure 9(b), we show how scalable our VMesh schemes are in terms of server stress compared to traditional client-server approach. The figure shows the server stress against simulation time when the average user population is set to 3200. In traditional client-server system, each new client requires a dedicated stream from the server to the client. Therefore, its server stress increases linearly with the user population, and hence the system is not scalable. On the other hand, VMesh maintains a relatively constant server stress at a low level (around 100 to 800). This is because the peers themselves also become data suppliers and serve other peers. One important point to note is that, in the previous work which applies "cache-and-relay" paradigm, when a peer jumps to another playpoint of the video, all its children need to search for new parents. This situation becomes severe if the system uses a tree-based overlay for delivering media data, because all its descendants would suffer from buffer shortage if new parents could not be found before all the buffered content is drained. In this case, clients experience discontinuous playback and need to wait for the buffer to be filled up again. One solution to this may be requesting data from the media server, but this definitely increases server stress. Therefore, those systems are not suitable for providing interactive VoD services. Unlike the previous work, VMesh peers store segments statically. A peer's random seeking does not stop its children from continuing to receive data from its storage. Therefore, VMesh can keep its server stress at a low level under dynamic peer join/leave and user interactivity.

2) High Playback Continuity: Figure 10(a) plots the average continuity index against the average user population. All three VMesh schemes can achieve high continuity under dynamic peer join/leave in a lossy network. Since VMesh employs multiple parents, if one of the parents leaves the system ungracefully, the child can request the remaining parents to share the load of the departed parent temporarily. This helps keep the continuity stable and high even under dynamic peer join/leave. Additionally, VMesh(LA) and VMesh(LA-POP) achieve better continuity than VMesh(RAND) by around 5% and 8%, respectively. This is because our locality-aware segment location scheme assists the peers to search for close parents, whose data delivery may experience less packet losses. Also, in case of packet loss, close parents are quick enough to retransmit lost packets before the playback deadlines. VMesh(LA) and VMesh(LA-POP) have similar playback continuity, as the popularity-based segment storage scheme used in VMesh(LA-POP) only helps in load-balancing among storage peers, and hence reduces the server stress. It does not help improve the playback continuity very much as expected.

For P2VoD, since the peers are organized in a tree-like structure, as other tree protocols, data loss accumulates downwards

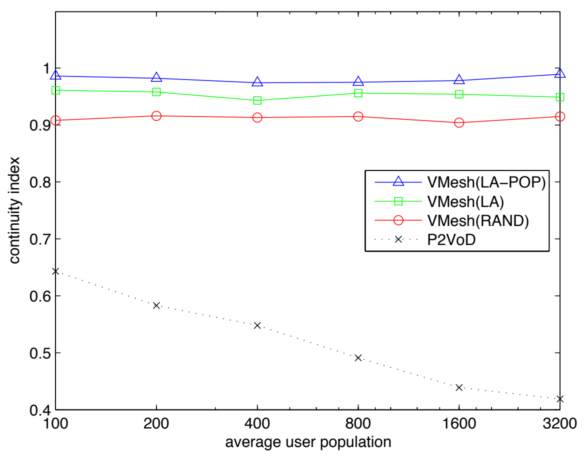


(a) Server stress against average user population.

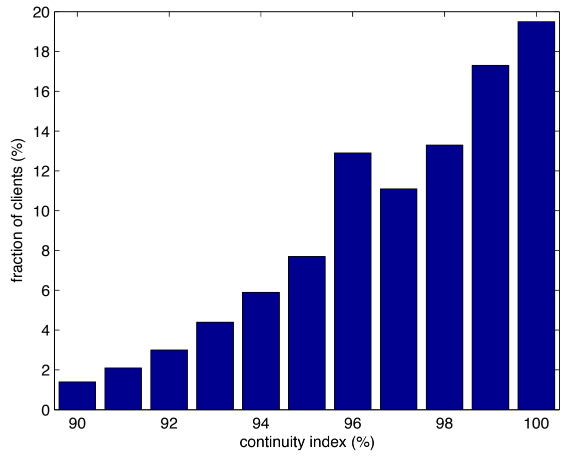


(b) Server stress against simulation time (in second). The user population grows from zero to 3200.

Fig. 9. Server stress under dynamic peer join/leave in a lossy network.

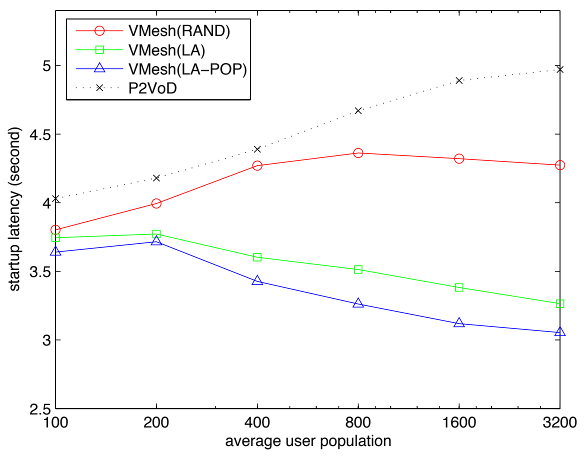


(a) Continuity index against average user population.

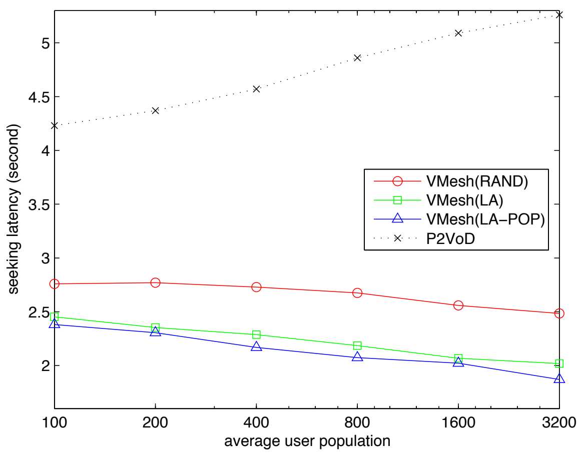


(b) Distribution of continuity index among users in VMesh(LA-POP). (User population = 3200.)

Fig. 10. Playback continuity under dynamic peer join/leave in a lossy network.



(a) Startup latency against average user population.



(b) Seeking latency against average user population.

Fig. 11. Startup and seeking latency.

the tree. The high loss rates at the lower-level peers lead to very low playback continuity. In addition, since a peer has only one parent, dynamic peer joins and leaves cause frequent parent failures. This makes the peers miss playback deadlines very often when searching for new parents.

Figure 10(b) shows the distribution of playback continuity among VMesh(LA-POP) users when the average user population is 3200. As shown in the figure, over 95% of the users receive playback continuity greater than 90%. For streaming video, a small number of lost frames can be skipped without user noticing the difference. To recover original media data, techniques such as forward error correction (FEC) could be applied on the transmitted packets. Also, enlarging the clients' buffer size can help improve the continuity, but this may make the startup and jumping latency higher. This is a common tradeoff between the two metrics in media streaming systems.

3) Low Startup and Seeking Latency: Startup latency includes two parts: 1) the segment location latency, and 2) the buffering latency. The first part depends on the diameter of the network. If the end-to-end latency between nodes is low, the segment location latency will also be low since the search involves several hops of routing for DHT search. The average end-to-end latency among all the nodes in our network is around 200 milliseconds. The second part depends on the size of buffer to be filled up before playback and the end-to-end bandwidth. In our simulations, we set the buffer size to be 3 seconds of media data, and we limit the end-to-end bandwidth to the streaming bit rate. Figure 11(a) shows the startup latency for various average user population. As there are more users in the system, it is more likely for users in VMesh(LA) and VMesh(LA-POP) to find close parents and hence fill up their buffers more quickly. One disadvantage of using DHT for searching parents is that the searching latency is more or less the same regardless of the user population size. This is why we build another overlay mesh among peers to speed up random seeking latency. However, for playback startup, we need to apply DHT search for our locality-aware segment location.

Similarly, seeking latency includes two parts: 1) the segment location latency, and 2) the buffering latency. However, with the overlay links to the peers who store the next and the previous segments, seeking latency can be lower than the startup latency, because a peer could follow the mesh links to locate the targeted segments. The parents located using this method are also close to the peer since the mesh links are built using locality-aware segment location at the beginning. Therefore, this does not affect the performance on data delivery and server stress. Figure 11(b) plots the seeking latency against the average user population. Again, as there are more users in the system, it is more likely for users in VMesh(LA) and VMesh(LA-POP) to find close parents and hence fill up their buffers more quickly.

For P2VoD, since a new peer or a seeking peer needs to search its parent from the source downwards the tree overlay, the number of searching hops should be in $O(\log N)$ where N is the size of the tree overlay. However, due to limited outgoing bandwidth, the located parent may not be able to stream to the requesting client. Hence, the requesting peer needs to contact other peers again until it finds a peer with spare bandwidth.

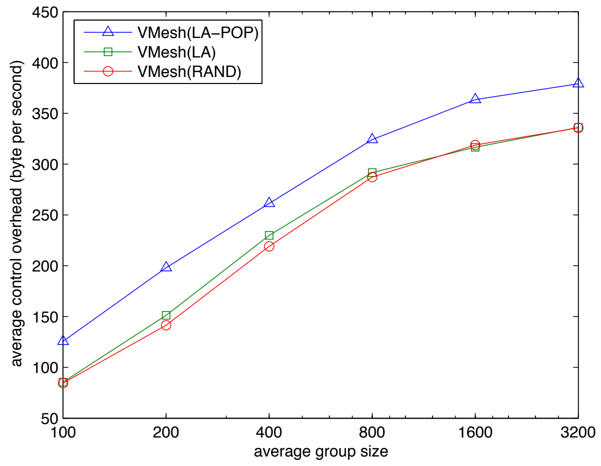


Fig. 12. Control traffic overhead against average user population.

VMesh does not share the same problem because a peer contacts a number of parents and starts downloading data in parallel.

4) Low Control Overhead: In our simulations, we count all VMesh control messages as control traffic overhead. In VMesh, joining, data scheduling, mesh construction, etc. requires peers to exchange control messages. For VMesh(LA-POP), distributed consensus requires exchange of STATE and REPLY messages. As shown in Figure 12, VMesh consumes very low control overhead and is scalable when the user population increases, ranging from 100 to 400 bytes per second. For a 1 Mbps video stream, the control traffic is only 0.1% to 0.3%. VMesh(RAND) and VMesh(LA) have similar control traffic, because the only difference between them is the value in the DHT search key. In VMesh(LA-POP), distributed consensus algorithm adds extra control messages, and hence increases the control traffic overhead by around 40 bytes per second.

VI. Conclusions

Providing interactive VoD service in P2P network is challenging, not only due to the asynchronous user access pattern but also the unpredictability of group dynamics and user interactivity. In this paper, we propose a novel architecture called VMesh to support interactive VoD service in P2P networks. VMesh utilizes the large storage capacity of peers to amplify the supply of videos so as to easily support the large demand in a scalable manner. In VMesh, videos are divided into smaller segments and stored in peers distributed over the Internet. A video mesh is built upon peers to support playback and jumping forward/backward during playback. A peer, who has a video segment stored in its local storage, connects to the peers who have the same, the previous and the next video segments. As a result, its children can be redirected to the peers who have the required segments. In order to provide failure tolerant streaming service, a client in the system connects to multiple parents who have stored and are able to stream the requested video segments in parallel and collaboratively. The parents could be searched in the P2P network via distributed hash table (DHT) technique using the key comprised of video ID and segment ID.

Our results show that our locality-aware segment location algorithm substantially reduce the server stress by allowing clients to find close and good quality parents. Our popularity-based segment storage scheme also helps reduce the server stress and improve the playback continuity by matching the supplies and demands of segments over the network. Unlike the previous work, in which a peer depends on what resides in the buffers of its parents, if the parents jump to another position in the video, the peer needs to search a new parent again. In VMesh, parent activities do not affect the children unless the parent shutdowns the service. Through simulation, we show that our system performs well under peer dynamics. We show that the system achieves high playback continuity under random member join/leave. In addition, the system achieves very low startup and seeking latency which is crucial to the performance of an interactive VoD system.

REFERENCES

[1] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services," in *Proceedings of the 6th ACM International Conference on Multimedia (MM)*, Bristol, England, Sept. 1998.

[2] L. Gao, D. Towsley, and J. Kurose, "Efficient Schemes for Broadcasting Popular Videos," in *Proceedings of the 8th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Cambridge, UK, July 1998.

[3] Y. Chu, S. Rao, and H. Zhang, "A Case for End System Multicast," in *Proceedings of ACM SIGMETRICS*, Santa Clara, CA, USA, June 2000.

[4] [Online]. Available: http://www.bittorrent.com

[5] H. Chi, Q. Zhang, J. Jia, and X. Shen, "Efficient search and scheduling in P2P-based media-on-demand streaming service," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 25, no. 1, pp. 119–130, Jan. 2007.

[6] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A Data-driven Overlay Network for Live Media Streaming," in *Proceedings of IEEE INFOCOM*, Miami, FL, USA, Mar. 2005.

[7] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," in *Proceedings of ACM SIGCOMM*, Portland, OR, USA, Aug. 2004.

[8] A. Sharma, A. Bestavros, and I. Matta, "dPAM: A Distributed Prefetching Protocol for Scalable Asynchronous Multicast in P2P Systems," in *Proceedings of IEEE INFOCOM*, Miami, FL, USA, Mar. 2005.

[9] C. Zheng, G. Shen, and S. Li, "Distributed Prefetching Scheme for Random Seek Support in Peer-to-Peer Streaming Applications," in *Proceedings of ACM Multimedia Conference, Workshop on Advances in Peer-to-Peer Multimedia Streaming*, Singapore, Nov. 2005.

[10] M. Zhou and J. Liu, "A Hybrid Overlay Network for Video-on-Demand," in *Proceedings of IEEE International Conference on Communications (ICC)*, Seoul, Korea, May 2005.

[11] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 22, no. 1, pp. 91–106, Jan. 2004.

[12] M. Guo, M. H. Ammar, and E. W. Zegura, "Cooperative Patching: A Client based P2P Architecture for Supporting Continuous Live Video Streaming," in *Proceedings of the 13th IEEE International Conference on Computer Communications and Networks (ICCCN)*, Chicago, IL, USA, Oct. 2004.

[13] T. T. Do, K. A. Hua, and M. A. Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," in *Proceedings of IEEE International Conference on Communications (ICC)*, Paris, France, June 2004.

[14] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proceedings of the 12th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, USA, May 2002.

[15] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001, pp. 329–350.

[17] H. Zhang, A. Goel, and R. Govindan, "Improving lookup latency in distributed hash table systems using random sampling," *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 5, pp. 1121–1134, Oct. 2005.

[18] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *1st International Workshop on Peer-to-peer Systems (IPTPS)*, Mar. 2002.

[19] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A public DHT service and its uses," in *Proceedings of ACM SIGCOMM*, Aug. 2005, pp. 73–84.

[20] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein, "A case study in building layered DHT applications," in *Proceedings of ACM SIGCOMM*, Aug. 2005, pp. 97–108.

[21] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proceedings of the 2005 USENIX Annual Technical Conference (USENIX '05)*, Anaheim, California, April 2005.

[22] A. Hu, "Video-on-Demand Broadcasting Protocols: A Comprehensive Study," in *Proceedings of IEEE INFOCOM*, Anchorage, AK, USA, Apr. 2001.

[23] Y. Guo, L. Gao, D. Towsley, and S. Sen, "Seamless Workload Adaptive Broadcast," in *Proceedings of IEEE International Packetvideo Workshop*, Pittsburgh, PA, USA, Apr. 2002.

[24] D. Eager, M. Vernon, and J. Zahorjan, "Bandwidth Skimming: A Technique for Cost-effective Video-on-Demand," in *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, Jan. 2000.

[25] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-peer Patching Scheme for VoD Service," in *Proceedings of the 12th ACM International World Wide Web Conference (WWW)*, Budapest, Hungary, May 2003.

[26] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "PROMISE: Peer-to-Peer Media Streaming Using CollectCast," in *Proceedings of the 11th ACM International Conference on Multimedia (MM)*, Berkeley, CA, USA, Nov. 2003, pp. 45–54.

[27] L. Guo, S. Chen, and X. Zhang, "Design and Evaluation of a Scalable and Reliable P2P Assisted Proxy for On-Demand Streaming Media Delivery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 5, pp. 669–682, May 2006.

[28] M. Zhou and J. Liu, "Tree-Assisted Gossiping for Overlay Video Distribution," *Kluwer Multimedia Tools and Applications*, vol. 29, no. 3, pp. 211-232, June 2006.

[29] H. Chi and Q. Zhang, "Efficient search in P2P-based video-on-demand streaming service," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, July 2006, pp. 565–568.

[30] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "On the Optimal Scheduling for Media Streaming in Data-driven Overlay Networks," in *Proceedings of IEEE GLOBECOM*, San Francisco, CA, USA, Nov. 2006.

[31] W.-P. K. Yiu, X. Jin, and S.-H. G. Chan, "Distributed storage to support user interactivity in peer-to-peer video streaming," in *Proceedings of IEEE International Conference on Communications (ICC)*, Istanbul, Turkey, June 2006.

[32] T. S. E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," in *Proceedings of IEEE INFOCOM*, New York, NY, USA, June 2002.

[33] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *Proceedings of ACM SIGCOMM*, Portland, OR, USA, Aug. 2004.

[34] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier, "Space Filling Curves and Their Use in Geometric Data Structures," *Theoretical Computer Science*, vol. 181, no. 1, pp. 3–15, July 1997.

[35] A. Fax and R. M. Murray, "Information Flow and Cooperative Control of Vehicle Formations," *IEEE Transactions on Automatic Control*, vol. 49, pp. 1465–1476, Sept. 2004.

[36] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray, "Asynchronous Distributed Averaging on Communication Networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 512-520, June 2007.

[37] "The Chord/DHash project." [Online]. Available: http://pdos.csail.mit.edu/chord/#downloads

[38] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 1996.

[39] V. N. Padmanabhan, L. Qiu, and H. J. Wang, "Server-based Inference of Internet Link Lossiness," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003.

[40] S. Jin and A. Bestavros, "GISMO: Generator of Streaming Media Objects and Workloads," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 3, pp. 2–10, Dec. 2001.

W.-P. Ken Yiu (S'03) received the B.Eng. and M.Phil. degrees in computer science from the Hong Kong University of Science and Technology (HKUST), Kowloon, in 2002 and 2004, respectively. He is currently working towards the Ph.D. degree at the Department of Computer Science and Engineering, HKUST.

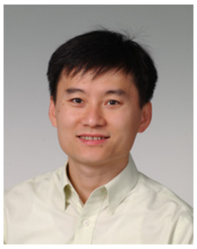
His research interests include computer networks, peer-to-peer systems, multimedia networking, and network security.

Mr. Yiu was awarded the Academic Achievement Medal from HKUST in 2002, and the Sir Edward Youde Memorial Fellowship from Sir Edward Youde Memorial Fund in 2005 and 2006. In 2007, he received the Professor Samuel Chanson Best Teaching Assistant Award at HKUST, and the Hong Kong Telecom Institute of Information Technology (HKTIIT) Post-Graduate Excellence Scholarship.

Xing Jin (S'04) received the B.Eng. degree in computer science and technology from Tsinghua University, Beijing, China, in 2002. He is currently working towards the Ph.D. degree at the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology, Kowloon.

His research interests include overlay multicast with applications and QoS issues, Internet topology inference, end-to-end measurements, and peer-to-peer streaming.

Mr. Jin was awarded the Microsoft Research Fellowship in 2005. He is a junior editor of the *Journal of Multimedia* since 2006.



S.-H. Gary Chan (S'89-M'98-SM'03) received the B.S.E. degree (Highest Honor) in electrical engineering from Princeton University, Princeton, NJ, in 1993, with certificates in applied and computational mathematics, engineering physics, and engineering and management systems, and the M.S.E. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1999, respectively, with a minor in business administration.

He is currently an Associate Professor with the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology, Kowloon, and an Adjunct Researcher with Microsoft Research Asia, Beijing. He was a Visiting Assistant Professor in Networking with the Department of Computer Science, University of California, Davis, CA, from 1998 to 1999. During 1992-93, he was a Research Intern at the NEC Research Institute, Princeton, NJ. His research interests include multimedia networking, peer-to-peer technologies and streaming, and wireless communication networks.

Dr. Chan is a member of Tau Beta Pi, Sigma Xi, and Phi Beta Kappa. He was a William and Leila Fellow at Stanford University during 1993-94. At Princeton University, he was the recipient of the Charles Ira Young Memorial Tablet and Medal, and the POEM Newport Award of Excellence in 1993. He served as a Vice-Chair of IEEE COMSOC Multimedia Communications Technical Committee (MMTC) from 2003 to 2006. He is a Guest Editor for the *IEEE Communication Magazine* (Special Issues on Peer-to-Peer Multimedia Streaming), 2007 and Springer *Multimedia Tools and Applications* (Special Issue on Advances in Consumer Communications and Networking), 2007. He is Co-Chair of the Multimedia Symposium for IEEE ICC (2007). He was the Co-Chair for the workshop on Advances in Peer-to-Peer Multimedia Streaming for the ACM Multimedia Conference (2005), and the Multimedia Symposia for IEEE GLOBECOM (2006) and IEEE ICC (2005).