Distributed Storage to Support User Interactivity in Peer-to-Peer Video Streaming

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

Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Email: {kenyiu, csvenus, gchan}@cs.ust.hk

Abstract— **Providing random access function in peer-to-peer on-demand video streaming is a challenging task, 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 in peers in a distributed manner. An overlay mesh is built upon peers to support jumping forward/backward, pause and restart during playback. Our scheme utilizes the large total storage capacity of peers to improve the segment supply so as to support interactive commands in a scalable manner. Through simulation, we show that our system outperforms a recent work, P2VoD. VMesh also has low segment missing rate under random member join/leave. In addition, the system achieves low joining and seeking latencies which are crucial requirements in an interactive VoD system.**

I. Introduction

With the popularity of broadband Internet access, there has been 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 model where each client has a dedicated stream from a VoD 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. Despite many proposals on scalable media streaming using IP multicast, providing such services 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 provided scalable solutions 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 themselves into overlay networks via unicast tunnels.[1] Each peer (called overlay node) in an 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 capacity of the whole system is greatly increased as compared to 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].

Though P2P approach has been shown to be quite successful for on-demand media streaming, its support to user interactivity is still challenging due to the following reasons:

- *Request asynchrony* – User requests come at different times, so different users are receiving different segments of the media. Therefore, simply multicasting the stream using ALM does not fulfill all the user requests.
- *Peer dynamics* – Users in the system may join or leave at any time, or even suddenly fail. Hence, there is a need for an efficient mechanism which is resilient to dynamic peer-to-peer environment.
- *Unpredictable interactivity* – Users may jump forward, backward, pause and resume the video at any time. This means that the parents (or "suppliers") of a user may need to be changed quite frequently.

Though P2P file swarming systems like BitTorrent may be used to download the whole media objects before playback, this introduces long startup delay for playback [4]. Though there has been much work on providing on-demand video service in P2P networks (i.e., low startup delay) [7]–[11], [13], few tackle the important problem — *user interactivity* — which we focus in this paper. As opposed to previous work, ours also uses *storage* instead of sliding window *buffering* to reduce the complexity.

We propose a novel P2P scheme called *VMesh* to support interactive VoD service. VMesh utilizes the large total storage capacity of peers to improve the supply of video segments so as to support the large interactive demand in a scalable manner. In VMesh, videos are divided into smaller segments (identified by segment IDs) and stored in peers distributed over the network. An overlay mesh is built upon peers to support playback and interactive functionalities during playback. A peer may store one or more video segments in its local storage. 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 who have the next requested segments. Furthermore,

This work was supported, in part, by the Direct Allocation Grant in HKUST (DAG05/06.EG10) and by the Research Grant Council in Hong Kong (HKUST6156/03E).

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

a peer also keeps a list of peers who store 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 and hence can stream the video ***in parallel*** and ***collaboratively***. The parents could be searched for in the P2P network using distributed hash table (DHT) with the key comprising video ID and segment ID [14]–[16].

VMesh has the following desirable properties:

- Scalability — the system is 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 could start playing the media with low delay (without downloading the whole movie).
- Failure resilience — the system is robust to 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 give a detail description of our scheme in Section II. Next, we present our experimental results in Section III. Finally, we conclude the paper in Section IV.

II. Scheme Description

In this section, we first describe VMesh, followed by our scheme on distributed storage of video. Finally, we introduce our random storage overlay mesh mechanism and its feedback-based maintenance.

A. VMesh Description

A video server stores a collection of videos for user access. Videos are divided into multiple segments, say 5 minutes per segment. For example, if the video bit rate is 1 Mbps, each segment is of size only 36 MBytes, which is usually available from local storage of most peers. Since the video segments are distributed among peers, VMesh makes use of DHT built among the users to bootstrap a new video streaming session. A peer randomly stores video segment(s) in its local storage. The segment would be used to stream to another peer of interest.

An overlay mesh called *video mesh* (VMesh) is built among the peers. A peer keeps pointers (i.e., the IP addresses) pointing to the peers which store the next video segment, the previous video segment or the same video segment as what the peer stores (this is for redirection to achieve load-balancing).

Figure 1 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 / current segments for random seeking and load balancing purposes. The overlay links are used to redirect users to the appropriate parents who store the next requested segments. In case of joining or jumping, a VoD client in our system searches its parents using DHT and gets the stream from those parents. In the traditional "cache-and-relay" paradigm

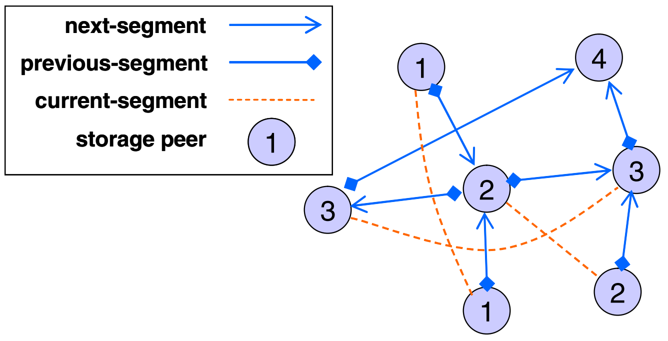


Fig. 1. In VMesh, storing peers hold various video segments and at the same time, they keep a list of peers who possess the next / previous / current segments for random seeking and load balancing purposes.

proposed in previous work, a VoD client relies on the content 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 contrast, each storage peer in VMesh would not change the video segments it stores and shares during its lifetime, and hence its activities (i.e., jumping) do not affect its children.

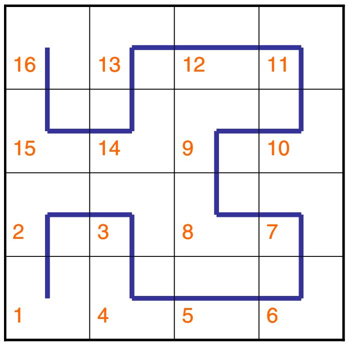
All peers register the search keys of their stored video segments in the DHT built among all the users. A new joining client performs the following procedure:

1) It searches for the segment of its interest using the DHT.
2) It contacts the returned list of peers and requests for the segment.
3) The contacted peers, if they have enough user capacity and bandwidth, become the client's parents for sending the requested segment. (Close parents are preferred for efficiency.[2]) Multiple parents are used for fault tolerance purpose. There are many scheduling algorithms for packet assignment with multiple parents [5], [12]. We consider simple round-robin scheduling in this paper.
4) When the client nearly consumes its current segment, it requests from its current parents for their lists of peers holding the next segment. It then contacts the peers in the returned list for continuous streaming.
5) If the client wants to jump to another video position which is not far away from the current one, it can simply follow its forward/backward pointers in the video mesh to contact the new parents. On the other hand, if the new position is too far away, it triggers another DHT search for the segment corresponding to the new position.

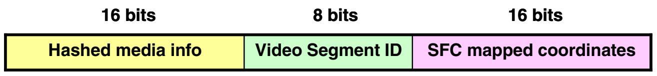
B. Distributed Video Storage using DHT

Network locations of the parents are quite important for efficient data dissemination. If the parent-child relationships are casually decided, there may be triangular routing between peers, wasting much network resources and increasing link stress. To address this, VMesh takes the network locations of the peers in the system into consideration.

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



(a) An example of Hilbert curve mapping from two-dimensional space to one-dimensional space.



(b) 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.

Fig. 2. The Hilbert curve mapping is used for mapping 2-D network coordinates of peers to 1-D numbers and the mapped numbers are put into the DHT search keys.

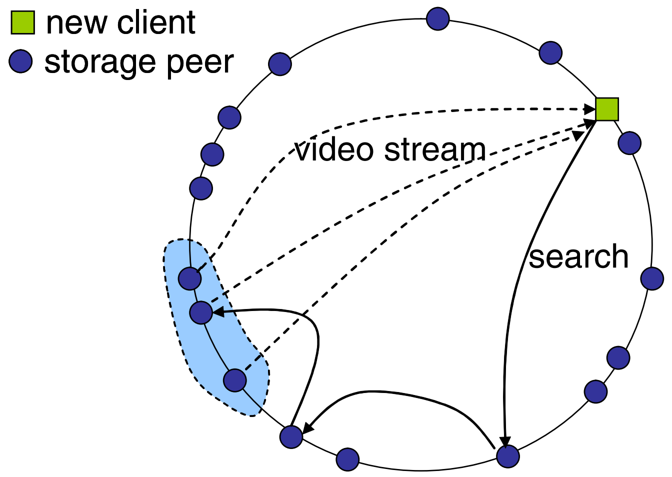


Fig. 3. Searching for the first video segment to kick off a new video streaming session.

The network location of a peer can be obtained using a network coordinate system like GNP and Vivaldi [17], [18]. 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 keys of the segments registered by the peers. Since most DHTs in structured overlays use a one-dimensional space for keys while coordinates are multi-dimensional, we need a mapping from the multi-dimensional coordinate space $\Re^d$ to the one-dimensional DHT key space $\Re$. We apply 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, their corresponding mapped one-dimensional distance is also close) [19]. Figure 2(a) shows how Hilbert curve maps a two-dimensional space to a one-dimensional space.

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 2(b), 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 for the purpose of balancing DHT load among peers. Each peer registers its own key for its stored video segment 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 in the search key, multiple parents closest to the requesting peer are returned. The peer can then connect to them as its parents and request for the stream.

Figure 3 illustrates how a new client receives the requested video stream in VMesh. Firstly, it searches for the first segment of the requested video stream using DHT. Then, based on our design of DHT search key, the closest available storage peers reply and begin to serve the requesting client.

C. Video Mesh and its Feedback-based Maintenance

Upon entering the system, 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 1, 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 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 parents during playback 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. Upon receiving the request, some pointers are sent to the child peer so as to allow it to get the next segment from other parents. In case of random seeking a favorite scene in a movie, users usually jump forward and backward 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. These lists can also reduce control messaging overhead by avoiding DHT searching for new parents by the client each time a segment

[3]The length of the DHT key can be extended to accommodate larger systems. The 16-bit hashed media information can already accommodate to about 65000 media objects in the system.

is nearly used up. Instead, the storage peers search only once and keep the list.

Though searching in DHT is not an expensive operation, keeping all the pointers in the lists current requires frequent updates which may be uneconomical. 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 again. 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.

III. Performance Evaluation

We evaluate the performance of VMesh with various parameter settings, and also compare it with other overlay on-demand streaming systems, in particular, P2VoD [11]. P2VoD organizes nodes into multi-level clusters according to their joining time, and the data stream is forwarded along the overlay tree built among the peers. Each host receives data from a parent in its upper cluster and forwards it to its children in its lower cluster. A new node tries to join the lowest cluster or forms a new lowest cluster. If it fails to find an available parent from the tree and the server has enough bandwidth, it directly connects to the server. In our experiment, we use Smallest Delay Selection for P2VoD's parent selection process. And, we set the system parameter $K = 6$ and the cache size MB to be 5 minutes of video in the system (K is the maximum number of clients allowed in the first generation of each video session). For details of the protocol, readers may refer to [11].

We build VMesh on top of a public Chord implementation [20]. In our simulations, the length and bit rate of a movie are 120 minutes and 1 Mbps, respectively. Each segment is 5 minutes long and of size about 36 MBytes. Each peer stores one video segment at its local storage. The peers participating in the system follow a Poisson arrival, each being randomly attached to a router node and request the video from the beginning. Group size, i.e. the total number of peers in a measurement session, varies from 200 to 12800. The underlying network topology is created using GT-ITM [21]. The whole network consists of 4080 routers and about 20000 links. In case a new client is unable to find available parent(s), the client is rejected to be served.

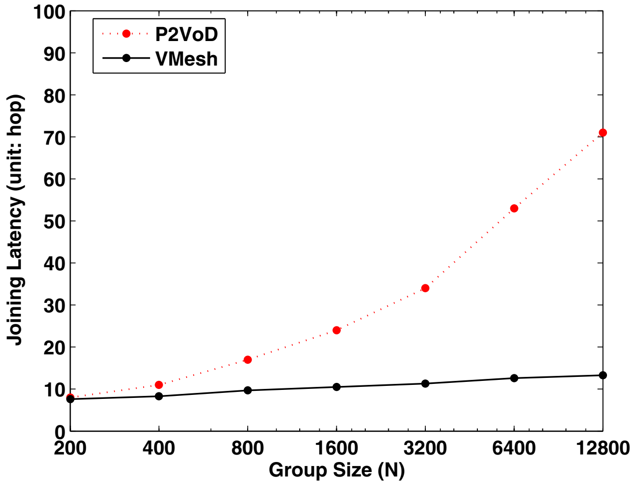


Fig. 4. Average joining latency.

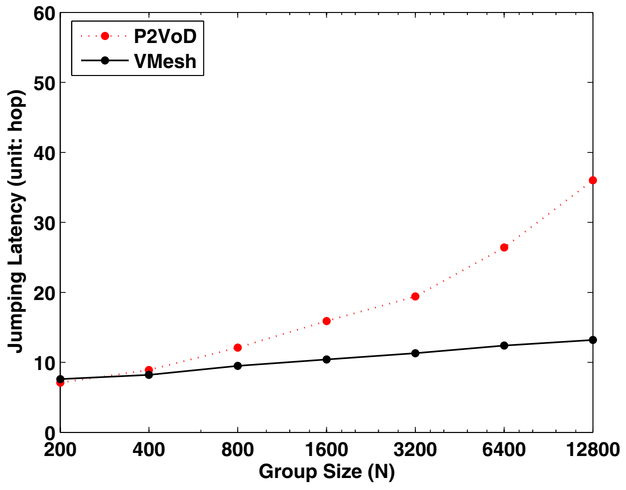


Fig. 5. Average jumping latency.

Figure 4 shows the joining latency in terms of hop count in P2VoD and VMesh. Note that we have observed negligible rejection rate in both systems with our settings. We hence do not discuss rejection rates in the following. From the figure, the join latency in P2VoD increases almost linearly with the group size, while that in VMesh only increases in logarithmic scale. In P2VoD, a new node first sequentially searches the second lowest cluster to find a parent with enough bandwidth. If that fails, it forms a new cluster and sequentially searches the lowest cluster to find a parent with enough bandwidth. If that fails again, it tries to connect to the server directly. With more nodes in the system, a new node usually needs to search more nodes for an appropriate parent. On the other hand, in VMesh, a new node can quickly locate nodes with the first several segments through DHT routing. The searching time via DHT is shown to be $O(\log N)$ where N is the number of nodes in the system. The latency is hence significantly reduced.

Figure 5 compares the jumping (forward or backward) latency in the two systems. As shown in the figure, the jumping latency in P2VoD increases much more quickly than that in VMesh. P2VoD has not been specially optimized for quick jumping. If a user initiates a jumping request, the node needs to sequentially search its upper or lower clusters. In the case of a large group of users, the number of clusters is very large and the search cost is high. In VMesh, however, a jumping request and a joining request are almost the same. In either case, the node searches for a certain segment through the DHT network. For a short-distance jump request, a client could request from its current parents for peers who have the previous/next segment. The searching time can be further improved.

Figure 6 shows the segment missing rate (SMR) in a dynamic network. A data segment is considered missing if it is not available at a node till the playback time, and the SMR for the whole system is the average ratio of the missed segments at all the participating nodes in the session [8]. As the figure shows, VMesh achieves much smaller SMR (below 5%) than P2VoD (around 14%). With error resilient coding techniques such as interleaving, such a loss rate can be efficiently masked [22]. Since P2VoD uses a single overlay tree for data delivery, the error rates for the nodes are increasing down the tree. Each P2VoD node maintains a large number of peer information, including its parent, children and siblings in the same cluster. If a node loses some segments because of background traffic, it needs to request for retransmission from its parent. In VMesh, however, we adopt a different approach to address this problem. Using multiple parents, while one of the parents fails to deliver its assigned packets, the remaining ones could be immediately requested to recover the missing packets. In addition, with DHT searching, multiple parents that store the desired segment can be found in one search. In case of data loss from some parents, the others can be quickly contacted for recovery. Furthermore, each storing peer maintains a few pointers to peers storing the same segment. This can also help its children quickly locate a new qualified parents in case of node failure.

IV. Conclusions and Discussions

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 scheme 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

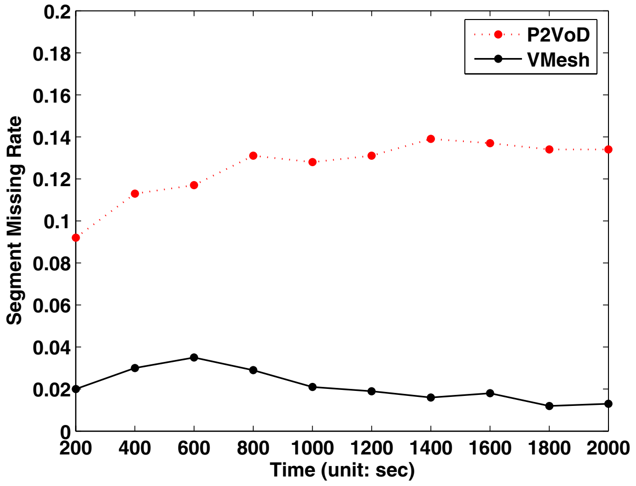


Fig. 6. Average segment missing rate.

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.

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 outperforms a recent research work, P2VoD. We show that the system has low segment missing rate under random member join/leave and random injection of background network traffic, which means that the video quality is good in the presence of group dynamics and bandwidth fluctuation. In addition, the system achieves very low joining and seeking latency which is crucial to the performance of an interactive VoD system.

Adding pointers to peers which are storing video segments far away from the current playback position can help a user seek a far away position more efficiently, instead of searching using DHT. In our future work, we will investigate how to add other pointers in the mesh and the effect of adding them. Also, in our current implementation, storage peers randomly choose video segments to download. Since a popular segment is more in demand than the others, it is better to balance the supply and demand in a strategical manner. The performance may be improved if storage peers could choose to download a video segment according to its popularity. However, how to obtain the popularity of each segment in a video and how to update it dynamically are very challenging in P2P VoD system. We will also look for solutions for this in the future. Implementing the system and testing it in PlanetLab (a worldwide testbed for Internet deployment) is also our future work.

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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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, jan 2004.

[10] 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.

[11] 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, jun 2004.

[12] 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.

[13] 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.

[14] 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, 2003.

[15] 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.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001.

[17] 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.

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

[19] 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.

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

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

[22] Y. Wang, S. Wenger, J. Wen, and A. G. Katsaggelos, "Error resilient video coding techniques," *IEEE Signal Processing Magazine special issue on Multimedia Communications over Networks*, vol. 17, no. 4, pp. 61–82, July 2000.