A DHT-Aided Chunk-Driven Overlay for Scalable and Efficient Peer-to-Peer Live Streaming

Haiying Shen, Lianyu Zhao and Ze Li
Department of Electrical and Computer Engineering
Clemson University Clemson, SC 29634
{shenh, lianyuz, zel}@clemson.edu

Jin Li
Microsoft Research
Redmond, WA 98052
jinl@microsoft.com

***Abstract*—Internet-based video streaming applications is becoming more and more popular and attract millions of online viewers every day. The incredible growth of viewers, dynamics of participants, and high video quality-of-service (QoS) requirement pose scalability, availability and low-latency challenges to peer-to-peer (P2P) live video streaming systems. Tree-based systems have low-delay but are vulnerable to churn, while mesh-based systems are churn-resilient but suffers high delay and overhead. Both systems cannot make full utilization of the bandwidth in the system. To tackle the challenges, we propose a DHT-aided Chunk-driven Overlay (DCO). It introduces a scalable DHT ring structure into a mesh-based overlay to efficiently manage video stream sharing. DCO includes a hierarchical DHT-based infrastructure and a chunk sharing algorithm. Aided by DHT, DCO guarantees stream chunk availability. In this way, DCO flexibly takes full advantage of available bandwidth in the system and at the same time provides high scalability and low-latency. Simulation results show the superiority of DCO compared with mesh-based and tree-based systems.**

Keywords: DHT; P2P; Live streaming; Chunk-driven.

I. Introduction

Internet-based video streaming applications is becoming more and more popular and attract millions of online viewers every day [1]. The number of unique viewers of online video increased 5.2% year-over-year, from 137.4 million unique viewers in January 2009 to 142.7 million in January 2010 [2]. Take YouTube as an example, 120.5 million viewers watched videos on YouTube in the month of August of 2009 [3], and the number is expected to rise to at least one billion viewers worldwide in 2013 [4]. Live streaming applications provide live broadcasting streams from live channels such as TV and live events. For instance, YouTube's live streaming of Ireland U2 Concert performance was watched by 10 million people [5]. Recently, peer-to-peer (P2P) techniques have attracted significant interests for live video broadcasting over the Internet due to its high scalability. In a P2P live video streaming system, a streaming media server generates a series of chunks, each of which is a small video stream fragment containing the media contents of a certain length. The peers watching the same video program form an overlay for video stream sharing between each other in the form of chunks. The P2P paradigm dramatically reduces the bandwidth burden on the centralized content provider and generates more available bandwidth as the number of viewers increase. Typical P2P video streaming applications include PPLive [6], Joost [7], SopCast [8], UUSee [9], ESM [10] and CoolStreaming [11]. As an example, UUSee simultaneously sustains 500 live stream channels [9] and routinely serves millions of users [12] each day. As users spend more and more time watching videos online, they are becoming increasingly unsatisfied with the quality-of-service (QoS) (i.e., image freezes and poor resolution) [13, 14]. The incredible growth of viewers, dynamism of participants, and high-QoS requirement pose *scalability, availability and low-latency* challenges to the widespread adoption of the applications.

- *Scalability*. The performance of a system will not degrade or will even improve as the number of users grows to a large scale.
- *Availability*. The live streaming service with acceptable streaming quality is always available to users under all network conditions, including node dynamics (i.e., churn).
- *Low-latency*. High quality video streaming with stringent real-time performance demand requires that video stream is transferred under timing and bandwidth constraints.

However, the capability of existing P2P live streaming systems [11, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29] is insufficient to tackle these challenges. Most current systems construct their overlays into either tree-based structure or mesh-based structure.

In a tree-based structure [15, 16, 17, 18, 19, 20, 21], all peers are arranged in a tree and the source distributes the live stream from the tree root. The internal nodes receive media pieces from parents and relay them to child nodes. This solution can rapidly deliver contents if the tree is stable and the links between nodes have sufficient bandwidth. However, it has a number of inherent limitations: *(i)* It is vulnerable to churn. If a parent leaves or fails, its children cannot receive live stream in time before the tree is repaired. *(ii)* Fixed tree structure and stream flow direction makes it difficult to flexibly take full advantage of bandwidth. *(iii)* Since the performance is limited by the minimum throughput among the upstream connections, an inappropriately constructed tree may result in inefficient bandwidth utilization and long delay.

In a mesh-based structure [11, 13, 22, 23, 24, 25, 26, 27, 28, 29], each node maintains a list of their overlay neighbors. Each node maintains a buffer map which summarizes the chunks that it currently has cached and makes them available for sharing. Nodes periodically exchange and compare the buffer maps with their neighbors and retrieve video data accordingly. Specifically, peers share chunks through either pull-based [11, 22, 23, 24, 25, 26, 27, 28] approaches or push-based [29, 13] approaches. In the former, each peer sends requests to its neighbors to constantly seek for new chunks; while in the latter, a peer pushes chunks to its neighbors once there is available upload bandwidth. A mesh-based structure is naturally resilient to churn without a strict topology, but it also has a number of limitations. *(i)* A peer may suffer long latency if its desired chunks are not available in its neighbors. *(ii)* Periodical information exchange generates high overhead and consumes much bandwidth. *(iii)* Push-based approaches may cause a node to receive many identical chunk, producing extra overhead.

Several recently proposed works [23, 24, 30, 31] seek to combine mesh and tree structures to enhance their individual performance. However, the nodes in the mesh still need to exchange information periodically and the tree structure needs to be maintained and frequently adjusted. These hybrid methods cannot overcome the inherent problems in both structures.

Distributed Hash Table (DHT) [32, 33] is a structure for managing information and is deployed in many file sharing applications. It is well-known for high scalability, reliability and self-organizing. In order to solve the drawbacks of current live streaming solutions and tackle the aforementioned three challenges, we propose a DHT-aided Chunk-driven Overlay (DCO). DCO includes a hierarchical DHT-based infrastructure and a chunk sharing algorithm. The DHT structure is formed by stable nodes to help efficiently locate optimal chunk providers among all available candidates in the system in order to guarantee content availability and low-latency. Basically, nodes report their available chunks to the DHT and also resorts to the DHT for locating a chunk provider for its requested chunk. The DHT always returns a chunk provider with sufficient bandwidth. In this way, DCO flexibly takes full advantage of available bandwidth in the system and at the same time provides high scalability, availability and low-latency.

Tree-based systems rigidly determine node locations in the tree. Thus it is unable to flexibly adapt to continuous changes in bandwidth status and churn. DCO is churn-resilient since a node can always find a chunk provider with sufficient bandwidth even in churn. Mesh-based systems consume high bandwidth for frequent message exchanges and cannot guarantee chunk availability due to local neighbor search. Unlike the mesh-based systems, DCO does not need frequent message exchanges while ensuring chunk availability due to system-wide search with the aid of DHT. As far as we know, this is the first work that leverages DHT to increase scalability and availability and reduce latency in P2P live streaming systems.

The rest of this paper is organized as follows. Section II gives a brief overview of the existing approaches for P2P live streaming. Section III presents the DCO system in detail. Section IV presents simulation results of DCO in comparison with other approaches. Section V concludes the paper with remarks on our plans for the future work.

II. Related Works

P2P live video streaming systems can be mainly classified into two categories: tree-based and mesh-based [34]. The early P2P streaming solutions are single-tree based, such as multicast overlay [16, 15], Narada [17] and ZIGZAG [18]. They feature a single multicast tree with the server at the root position. The single-tree approach suffers from sub-optimal performance of throughput and is vulnerable to churn. Later works including CoopNet [19], SplitStream [20] and THAG [21] employ multiple description coding (MDC) to divide media contents into multiple sub-streams, which are delivered through multiple multicast trees. Although they are more robust to churn, they generate high maintenance cost and involve complex protocols.

A mesh-based system such as CoolStreaming [11], AnySee [35] and Chainsaw [22] constructs a mesh out of the overlay nodes and swarms media contents by interchanging chunks with neighbors. CliqueStream [25] builds a live streaming network on top of eQuus [36], which is a clustered locality-aware P2P overlay. To improve resource utilization between peers, a number of packet scheduling algorithms have been proposed, such as AQCS [26], RUPF [27] and DP/LU [13], and their performances are compared extensively in [13, 28].

A number of recently proposed works [23, 24, 30, 31] combine tree and mesh structures to construct hybrid overlays. PRIME [23] is a two-phase mesh-based live P2P streaming system. It builds a tree with nodes being located in different levels according to their distances in hops to the server. In the first phase, data segments of a chunk are rapidly transmitted from the server in the top-down manner along the tree. The second phase is swarming content delivery, in which peers pull their data segments of the chunk from their neighbors in the same level. Chunkyspread [24] forms multiple trees over a mesh. The server multi-casts video stream to its neighbors, each is the root of each tree. Each node periodically checks to find overloaded parent and recommends a set of candidate nodes in the tree to its parent for the replacement. In [30, 31], Wang *et al.* proposed a two-tier structure. In the first tier, stable nodes constitute a tree-based backbone, which pushes most data downwards. All transient (and stable) nodes form a mesh overlay in order to enhance the resilience to churn and provide high utilization of bandwidth among overlay nodes. To reconcile mesh pull and tree push methods, each node collects the chunks pushed to it and uses a pointer to indicate the latest chunk obtained through the push method. The missing chunks, which precede the pointer's chunk, are filled by the pull method.

The heterogeneous nature of a P2P network has been studied, and peers with different outgoing bandwidth are treated differently in some designs. Banerjee *et al.* [37] used supernodes or dedicated proxies to provide efficient data distribution

services to a set of end-hosts. In the method proposed in [38], peers with larger outgoing bandwidth adaptively move closer to the source to reduce the mesh delay of the whole system. This method takes some time to reach the optimum state because every node only has knowledge of its neighbors. Yeung and Kwok [39] proposed to assign more parent nodes to peers with large outgoing bandwidth in order to ensure they receive stream in node dynamism.

III. The System Design

A. Background

1) P2P Video Streaming System: In a P2P video streaming system, live channel sources are broadcasting various media programs. A server in the live streaming network slices the media stream in a live channel into small chunks, each contains media contents of a certain length. The chunks are then delivered to the users who are watching the channel in the network. The users watching the same channel constitute an overlay for chunk sharing between each other. This process of chunk production and delivery is shown in Figure 1. Each chunk is named uniquely in the format of channel name plus its generation timestamp. For example, if a chunk has a session length of one second, the name of the chunk of NBC channel beginning at 01:30:01 on January 1st, 2009 is labeled as *NBC20090101013001*. The naming mechanism ensures that every chunk name is unique. Chunks are constantly generated from channel servers at a certain streaming rate. Every node watching the channel keeps a playing buffer which contains a certain number of chunks whose timestamps are within a short time window. These chunks are called *active chunks*. The time window steadily moves forward as new chunks are received for streaming playback.

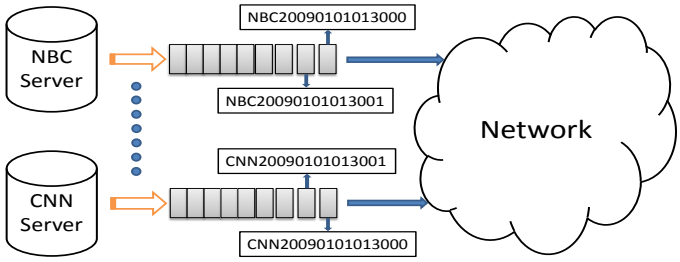

Fig. 1. The process of chunk production and delivery.

2) The DHT Systems: DHT systems allow for file sharing on a large scale without being controlled by a centralized organization. In a DHT, every node maintains a routing table, in which the number of entries amounts to $\log n$, where n is the number of nodes in the system. Each node and file is assigned with a unique ID which is the consistent hash value [40] of its IP address or file name, respectively. A file is stored in the node whose ID equals or immediately succeeds the file's ID. We call this node the *owner* of the file or the file's ID. A DHT system provides two main functions: *Insert(ID,object)* and *Lookup(ID)* to store a file to the ID's owner and to retrieve the file. The message of the functions is forwarded based on the DHT routing algorithm. The number of hops in a routing in the worst case is $\log n$. DHT systems have a self-maintenance mechanism to deal with churn including node joins, departures and failures for structure maintenance. We use Chord DHT [32] in this work, although any other DHT system can be adopted for this work. As shown in Figure 2, in Chord, all nodes constitute a virtual ring in the network. Each node N_i has a predecessor *predecessor*(N_i) and a successor *successor*(N_i).

B. DHT-Aided Chunk-Driven Overlay

By taking advantage of the file storage and lookup functions of DHTs, we build a DHT-aided chunk-driven overlay for scalable and efficient chunk sharing in P2P live streaming systems. DCO builds a hierarchical DHT-based infrastructure as shown in Figre 2, where stable nodes form a Chord DHT in the upper tier and other nodes connect to the DHT nodes in the lower tier. DCO organizes all nodes' chunk information elegantly in the DHT so that a node can always find providers with sufficient outgoing bandwidth in a short time. An active chunk in a node has a chunk index indicating its name, its owner node, its owner's buffer map, available bandwidth and so on. The index of each active chunk in a node is regarded as a file for storage through *Insert(ID,index)* and lookup through *Lookup(ID)* in the DHT. Using the name of a missing chunk as a file name, a node can always find a chunk provider from the DHT. While the node is watching the channel and receiving chunks from its provider, it continuously reports its buffered chunks to the DHT and provides the chunks to other nodes upon receiving their requests.

1) Hierarchical DHT-based Infrastructure: The importance of stable peers in the overlay is recognized and they are given priorities in the peer selection process [31, 41, 42, 43, 44]. Our design selects stable nodes to form a DHT ring structure for high chunk availability and QoS. Specifically, a small number of stable nodes watching the same channel are selected to form the DHT ring structure and to manage indices of active chunks. We call these nodes *coordinators*. There are two main reasons for choosing partial stable nodes rather than all nodes to form the DHT structure. First, the number of active chunks in a live stream channel at a time is limited. As each node's time window of a fixed length steadily moves forward, new chunks are created and outdated chunks are discarded, hence the total number of active chunks in the system at a time does not vary greatly. The second reason is stability. While watching live videos, every node in the network consults the coordinators for chunk indices and reports its chunks to the coordinators for chunk sharing. Therefore, the stability of the coordinators is critical to the availability of chunks and the quality of the live stream.

a) Stable Node Identification: Previous works in [44, 31, 43, 42] have proposed methods to identify stable nodes. It is indicated that the longer a node stays in the overlay, the longer it would stay in the future [44]. We adopt the method in [42] to calculate the probability that a node will stay long

in the network. We call it longevity probability. We consider two factors that are proved effective in identifying stable nodes [42]: (1) streaming quality, which is the *buffering level* defined as the number of consecutive blocks in the playback buffer starting from the current playback position, and (2) joining time, which is the time of a day when the node joins in the network. The longevity probability $p_l(t)$ that a node will stay in the network after time t can be calculated by the Cox proportional hazards model [45].

$$p_l(t) = 1 - h_0(t)\exp(\beta^T \boldsymbol{z}) \quad (1)$$

where $h_0(t)$ is a non-negative baseline hazard function whose value is chosen so that $p_l(t) \leq 1$; $\boldsymbol{z} = (z_1, \ldots, z_p)$ is a covariate vector containing the aforementioned two influential factors affecting the lifetime of a node; $\beta = (\beta_1, \ldots, \beta_p)$ is a column vector of coefficients corresponding to the covariates values in $\boldsymbol{z}$. Based on $p_l(t)$, stable nodes can be identified.

b) Infrastructure Construction and Maintenance: *Node Join.* In current P2P live streaming applications such as UUSee, the server keeps track of tens to hundreds of nodes in each channel and provides up to 50 nodes to a newly joined node so that it can join in the overlay of the channel. In DCO, the server (that also functions as a coordinator) provides one coordinator N_i to each newly joined node N_j in a round-robin manner in order to achieve load balance between coordinators. Then, N_j connects to N_i and becomes N_i's *client* in the lower tier. N_j requests chunks from the DHT and reports chunks to the DHT via N_i. DCO aims to minimize the DHT network size in order to minimize its maintenance overhead and avoid overloading any coordinator upon many chunk requests. Driven by this goal, the network size of DHT in DCO is not fixed. Rather, it adapts to the actual load in the system. Specifically, when a node in the upper tier is overloaded, one of its stable clients in the lower tier joins in the DHT to release its load. For example, N_j periodically calculates its longevity probability. If the probability exceeds a pre-defined threshold, N_j reports to N_i. If N_i is overloaded, it acts as a bootstrap node and makes N_j as its successor or predecessor in the DHT. The process of node joins in the DHT is the same as that in general DHTs. Then, N_j becomes an coordinator and can directly communicates with the coordinators without relying on N_i. Regarding chunk indices as files in DHTs, N_j receives its responsible chunk indices from N_i based on the DHT file assignment policy, and will handle all requests for these chunks. Thus, partial load in N_i is transferred to N_j.

Node Departure. A node may leave a channel either gracefully by informing its neighbors or abruptly without any notice. When N_i gracefully leaves the network, it notifies nodes that are currently pulling chunks from it, so that they can request for new chunk providers from the DHT. Meanwhile, it informs the coordinators to which it has reported its chunks before, so that the coordinators remove N_i from their index tables. If N_i is a coordinator, it needs to conduct three more operations. (1) It notifies its clients, and recommends *successor(N_i)* and *predecessor(N_i)* to each group of half of its clients respectively for new connections. (2) It transfers its chunk indices to its successor and predecessor according to the DHT file assignment policy. That is, the chunk indices are stored in their new owners after N_i leaves. Thus, the chunk availability is guaranteed since the requests for the chunks will automatically be forwarded to their new owners according to the DHT routing algorithm. (3) It performs the standard leaving process in Chord by notifying its successor and predecessor, so that relevant DHT nodes are aware of N_i's departure.

Node Failure. When node N_i fails or abruptly departs, node N_j pulling chunks from N_i will notice a timeout failure in fetching chunks. Then, N_j will inform the chunk's coordinator about the failure of N_i and meanwhile receive a new chunk provider. The coordinator removes N_i from its index table. If N_i is a coordinator in the DHT, its client N_k will also notice its failure or departure upon communication failure. Then, N_k contacts the server for a new coordinator to connect to. The *stabilization* operation in DHTs helps to maintain the DHT infrastructure due to node joins, departures and failures. In stabilization, each node periodically probes its neighbors and update them if they are outdated.

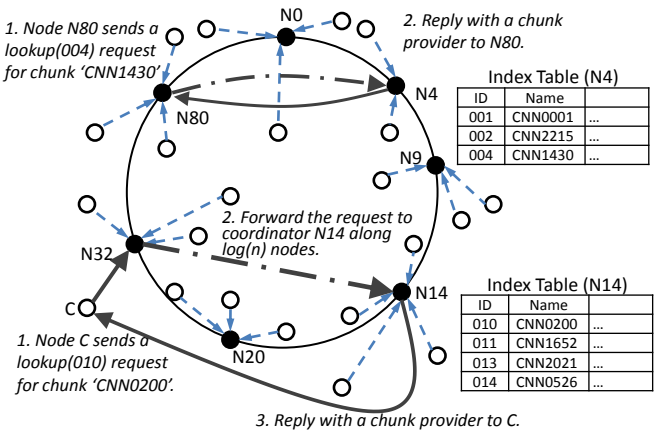

Fig. 2. Chunk sharing in the DHT-aided chunk-driven overlay.

2) Efficient Chunk Sharing: In DCO, the coordinators function as index servers by collecting chunk indices in order to facilitate chunk discovery. Each coordinator maintains an index table where each entry holds the indices of a chunk. Figure 3 illustrates an example of an index table in a node. Each chunk has an ID that is the consistent hash value of its name. A chunk index includes the chunk's ID, name (e.g., *CNN0240*), the IP address of its holder node (e.g., *192.168.0.2*), the chunk owner's buffer map and available bandwidth.

Figure 2 shows an example of the DCO infrastructure along with index tables in coordinators $N4$ and $N14$. When a video server generates a new chunk or a node receives a new chunk from another node, it stores the index of the new chunk in the DHT. Specifically, it generates the ID of the chunk by applying the consistent hash function to the chunk's name.

Algorithm 1: Pseudo-code for the chunk sharing algorithm.

Every node N:

if N *needs to buffer the next chunk* **then**
  // contact the coordinator
  Generate the chunk ID and send *Lookup(ID)* query
    // contact the chunk provider
  if *receive the response from the coordinator* **then**
    Send a request to chunk provider
  end
  if *receive the chunk* **then**
    Register to the coordinator as a chunk provider
  end
end

if N *receives a chunk requester* **then**
  if *have idle bandwidth* **then**
    Send the requested chunk to the requester
  end
end

if N *is a coordinator & it receives a message* **then**
  if N *is the destination of the message* **then**
    if *the message is a Lookup(ID) request* **then**
      Send a chunk provider with sufficient bandwidth to the requester
    end
    if *the message is a Insert(ID,index) message for a chunk* **then**
      Add the index of the chunk to its index table
    end
  end
  else
    Forward the message to the next hop
  end
end

if N *wants to join in the system* **then**
  Contact the server and obtain a coordinator
  Build a connection with the coordinator
end

if N *wants to leave the system* **then**
  Notify nodes that are receiving chunks from N
  Notify the coordinators of its chunks
  if N *is a coordinator* **then**
    Recommend its clients to its successor and predecessor
    Transfer the chunk indices in its index table to successor and predecessor
    Execute standard leaving process in DHT
  end
end

It then sends the chunk's index to the DHT by the function *Insert(ID,index)*. By the DHT routing algorithm, the index will be forwarded to the coordinator which is the owner of the ID. The coordinator adds the chunk's index to the corresponding entry in its index table. As a result, the indices of a specific chunk of different nodes in the system gather in the same coordinator, which facilitates the chunk discovery. In Figure 2, different providers' chunk indices of the chunk with ID=001 and name=CNN0001 are in the first entry. Since $N4$ is the owner of IDs 001, 002 and 004, it stores the chunk indices of chunks with these IDs. Similarly, the chunk indices for chunks 010, 011, 013 and 014 are in coordinator N14. Key distribution based on the consistent hash function in DHT leads to comparatively balanced key distribution, i.e., $\log n$ imbalance [32]. Hence, the chunk indices are distributed among coordinators in comparative balance. If a coordinator is overloaded due to the number of lookup inquiries from peers, new coordinators can always be added to the DHT to release its load.

The Index table in a coordinator

ID	Name	IP address	Buffer map	Bandwidth
00001234	CNN0021	192.168.0.2	...	...
		...	...	...
00001237	CNN0240	123.83.2.4	...	...
		...	...	...
00001238	CNN9343	195.163.2.1	...	...
		...	...	...
00001240	CNN2034	100.94.3.24	...	...
		...	...	...

Fig. 3. An example of an index table.

When a node needs chunks to play for a certain time, it consults the DHT. Specifically, it calculates the ID of the chunk by applying the consistent hash function on the chunk's name. Then, it sends *Lookup(ID)* request to the DHT. Through the DHT routing algorithm, the request will be forwarded to the owner of the ID, i.e., the coordinator of the chunk. For example, in Figure 2, node C requests chunk CNN0200. It generates the ID of the chunk, 010, and then asks $N32$ to send out request *Lookup(010)*. This request is forwarded to the coordinator $N14$, the owner of ID 010. Then, $N14$ responds to C a chunk provider with sufficient available bandwidth for the chunk transmission. A coordinator can directly send *Lookup(ID)* requests to DHT for chunks. As the figure shows, the coordinator $N80$ sends *Lookup(ID)* request to coordinator $N4$. The coordinator processes the request and sends the information of chunk provider to the requester.

After receiving the information of a provider from the coordinator, the requester sends a chunk request to the provider. Once a requester receives new chunks, it reports the chunks to the DHT using *Insert(ID,index)* for chunk sharing. When a coordinator gracefully leaves, it transfers its chunk indices to its predecessor or successor based on the DHT file assignment policy. Then, the requests for the chunks will be forwarded

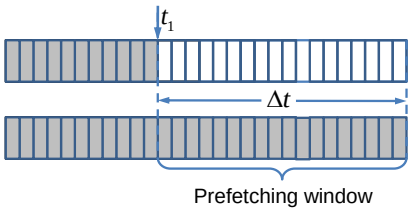

Fig. 4. Prefetching mechanism.

to the departed node's predecessor or successor accordingly based on the DHT routing algorithm. If a coordinator fails or abruptly leaves, the requests for the chunks whose indices were originally in the coordinator will be forwarded to a new coordinator, and at the same time, new chunk indices of the requested chunk will also be reported to the new coordinator. In conclusion, unlike tree-based systems, DCO can guarantee chunk availability even in churn. Also, the DHT is able to collect the active chunks of all nodes in the network and offer requested chunks to the participants all the time. This system-wide chunk search significantly enhances chunk availability of mesh-based methods, which only allow nodes to search chunks within their local neighbors. Algorithm 1 shows the pseudo-code for the chunk sharing process.

3) Prefetching Mechanism: We use the prefetching mechanism [46, 47] to enhance the QoS of live streaming. In DCO, the prefetching mechanism helps to reduce the possibility of disruption in playback caused by the unavailability of chunks. Thus, when a node turns to a new chunk provider when its current chunk provider fails, the playback process will not be affected. Recall that a node's message needs to take $\log n$ hops in the DHT overlay before arriving at the coordinator. The prefetching mechanism also helps to offset this delay. Figure 4 shows the prefetching window of a node, and t_1 indicates the chunk that is being played. The prefetching window size, Δt, should be sufficient to offset the delay caused by the $\log n$ hop routing and chunk provider switch. In UUSee, each chunk represents 1/3 of a second of video content, and the typical Δt of its prefetching window is 20s [12], hence the number of chunks in the buffer is usually 60. Users usually have different lags in the video playback, we consider the largest lag among the users, which is typically on the order of minutes [48]. If the lag difference is 10 minutes, the number of chunks of the same channel in the network at a time is approximately $60 + \frac{10 \times 60}{1/3} = 1860$. Assume one chunk provider is in charge of each chunk, then 1860 chunk providers are needed in the DHT. Considering that the typical delay in today's broadband Internet connection is below 0.1s [49], the longest delay in the DHT is $0.1 \times \log_2 1860 \approx 1.09s \ll 20s$. The result shows that the $\log n$ hop delay in DHT can be easily covered by the current prefetching mechanism.

We propose an adaptive prefetching window mechanism. First, the system pre-defines the size of the prefetching window based on the fetching delay. Then, each node adjusts the size based on two factors: (1) Its download bandwidth denoted by b. (2) The probability of the chunk fetching failure p_f it has experienced. Nodes with lower download bandwidth and more chunk fetching failures need a larger prefetching window. Therefore, each node calculates its prefetching window size W_{pf} by:

$$W_{pf} = \frac{W \times B}{b \times (1 - p_f)}, \tag{2}$$

where W is a predefined prefetching window size, and B is the average bandwidth in the network. Therefore, nodes periodically adjust the size of their prefetching window dynamically in order to guarantee high QoS while minimizing window size for low overhead.

IV. Performance Evaluation

To measure the performance of DCO, we developed our simulator based on P2PSim [50] and compared DCO's performance with pull-based, push-based, and tree-based methods. The number of nodes in the network was set to 512 unless otherwise specified. To make results comparable, all nodes form a DHT in DCO. We regard the neighbors in a node's successor list in DCO as the node's neighbors. In the pull-based and push-based mesh overlays, every node is randomly connected with its neighbors. In these overlays and DCO, the number of neighbors per node was varied from 8 to 64 with a step increment of 8. The tree-based method is constructed such that each node has the same out-degree, which is $\frac{1}{8}$ of the number of neighbors in the other three methods. Nodes in the pull/push-based methods exchange buffer maps with their neighbors every second. In the push-based method, every node sends missing chunks to their neighbors regardless whether they have received chunks from others; while in the pull-based method, every node sends a request to each of its neighbors asking for its missing chunk in a round robin manner until it receives the chunk. In the tree-based method, the chunks are pushed top-down from the server. In the simulation, a chunk is a video fragment that can be played for one second. Since today's online video is approximately 300kbps, the size of a video chunk was set to 300kb and was generated from server node every second for 100 seconds unless otherwise specified. We set both the upload and download bandwidths of the server to 4000kbps, and those of all other nodes to 600kbps. When a node is overloaded, it will queue its chunks in its buffer and will not perform any chunk transmission until it has sufficient bandwidth. We evaluate our proposed approach by examining the following performance metrics.

(1) *Mesh delay*: The time interval from the time when a chunk is generated at the server until it reaches all nodes in the overlay. This metric measures the stream dissemination latency of a live streaming system.
(2) *Fill ratio*: The ratio of nodes holding a chunk at a certain time. This metric indicates the spreading rate of video chunks.
(3) *Extra overhead*: The number of communication messages other than video chunks. In the push and pull methods, it includes messages for buffer map exchanges and sub-

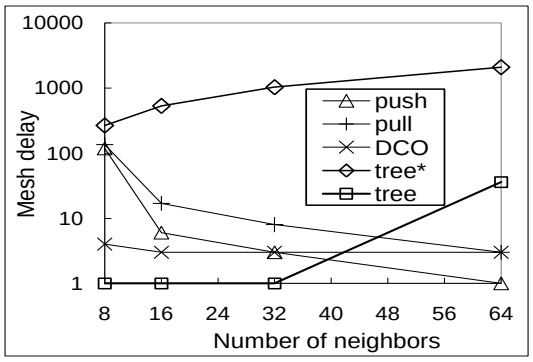

Fig. 5. Mesh delay vs. the number of neighbors.

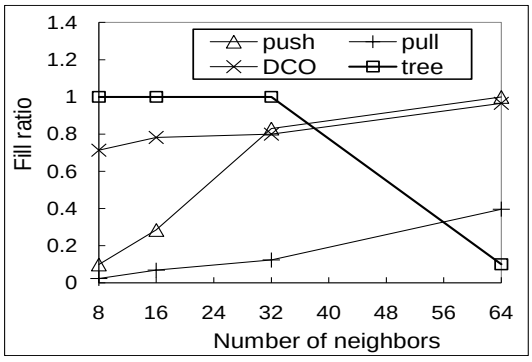

Fig. 6. Fill ratio vs. the number of neighbors.

sequent requests. In DCO, it includes messages between nodes and both coordinators and chunk owners. The tree-based method does not generate any extra overhead due to its top-down push method. One message forwarding operation is regarded as one unit of extra overhead. This metric reflects the cost, scalability and efficiency of a live streaming system.

(4) *Percentage of received chunks*: The number of chunks successfully received by all the recipients over the total number of chunks. It shows the churn resilience performance of a live streaming system.

A. Performance of Latency

Figure 5 shows the mesh delay versus the number of neighbors per node. Here "tree*" denotes a tree-based method in which the number of children each node has is the same as the number of neighbors in the other three methods, while "tree" denotes our experiment setting for the tree method in which the number of children of each node is $\frac{1}{8}$ of the number of neighbors in the other three methods. As the result shows, the mesh delays in the push and pull methods are very high when the number of neighbors is small. The mesh delay of DCO stably remains in a very low level all the time, even though DCO has $\log n$ hops delay in routing. The reason is that DCO can almost guarantee the availability of a chunk due to its system-wide search, while nodes in push/pull may not always get desired chunks due to their local search among neighbors. A chunk request in DCO is always answered with a chunk provider. However, in the push and pull methods, a peer may take a relatively longer time to find a neighbor with a requested chunk, especially when the size of the neighbor list is small. We also observe that the pull method generates higher mesh delay than the push method. This is due to the fact that nodes in the pull method need to pull their neighbors one by one and wait for their responses, which takes a longer time than directly accepting chunks from neighbors in the push method.

In the tree-based method, when the number of children is set to the same number of neighbors as other methods, the performance of tree is degraded significantly. This is because the parent nodes need a long time to be able to push a specific chunk to a large number of children. However, when the number of children is set to $\frac{1}{8}$ of the number of neighbors of other methods, tree can achieve the best mesh delay when the number of children is less than $\frac{32}{8}$. This is because when the number of children is larger, the bandwidth limit constrains the spreading of chunks from parents to children. Therefore, to make the results comparable, in the following experiments, we set the number of children of the tree method to $\frac{1}{8}$ of other methods, and set the default number of children to 3.

B. Performance of Availability

By measuring the fill ratio at a certain moment, we test the speed that chunks are made available to nodes in different approaches. We evaluate the fill ratio versus the number of neighbors per node and the elapsed time, respectively. In the experiment, a server generates 100 chunks. Figure 6 shows the measured fill ratios two seconds after a chunk is generated and the number of neighbors. Clearly, the performance of DCO is most stable when the number of neighbors is under 32, and beyond that the push method has nearly the same fill ratio as DCO. The fill ratio of the push method grows sharply when the number of neighbors increases from 8 to 32. This is because when a node has many neighbors, the push method is able to push the chunks to every node in the network in a few steps, functioning as flooding. When a node has fewer neighbors, the push method needs more time to spread a chunk. The pull method always shows the worst performance. This is because a node has to pull from its neighbors for a chunk and then wait for the response. If the neighbor does not have the chunk, the requester needs to pull from another neighbor. Thus, chunk spreading takes a long time. For the tree-based method, when the number of children is less than $\frac{32}{8}$, its fill ratio reaches nearly 100%, which is higher than the other three methods. Without buffer map exchanges and $\log n$ hop request routing, the tree-based method directly pushes a chunk to nodes along the tree. Thus, it generates a higher fill ratio than others. However, when the number of children is larger than $\frac{32}{8}$, the fill ratio drops dramatically. This is because when the number of children is small, a node can rapidly deliver a chunk to its children with low bandwidth constraint. However, when the number of children is large, because of the bandwidth

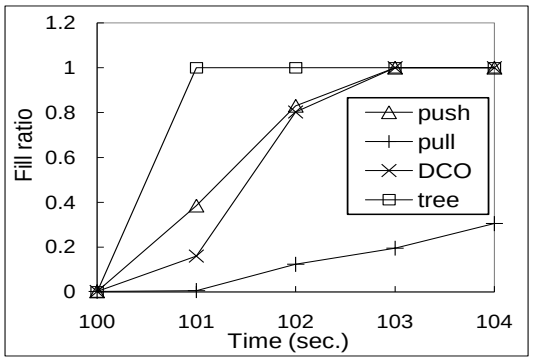

Fig. 7. Fill ratio vs. time.

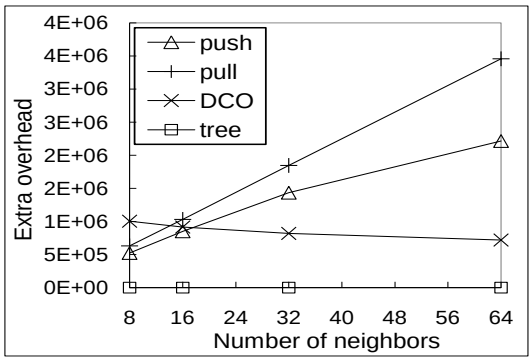

Fig. 8. Extra overhead vs. the number of neighbors.

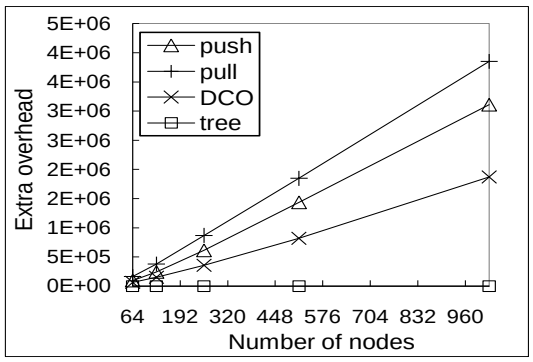

Fig. 9. Extra overhead vs. the number of nodes.

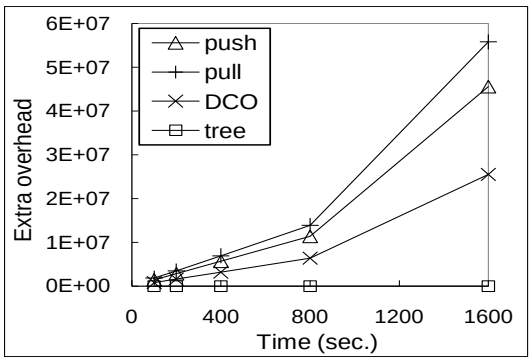

Fig. 10. Extra overhead vs. time.

constraint, the pushing process is significantly slowed down.

Figure 7 shows that the fill ratio increases as time elapses. Since it took the server 100 seconds to create all 100 chunks, we measured the fill ratio every second from the time instant of 100 seconds. In the tree-based method, when the number of children is set to the default value 3, it achieves the fastest chunk spreading speed for the same reason in Figure 6. The push method and DCO show better performance. It takes a chunk only three seconds to swarm the entire network after its generation. One interesting observation is that at the 101^{st} second, the fill ratio of push is better than DCO. This is because the $\log n$ routing hops in the DHT overlay make the initial chunk disseminating speed of DCO slower. At the 102^{nd} second, DCO is able to catch up with the push method, because there are more and more providers of the chunk as time goes on, which helps to achieve faster chunk disseminating speed by effectively utilizing more nodes' bandwidth. Due to the same reason in Figure 6, the pull method's fill ratio is significantly lower than others. It confirms the low speed of the pull method in spreading chunks.

C. Performance of Scalability and Overhead

In this experiment, we measured the total extra overhead for all nodes in the system to receive 100 newly generated chunks. Figure 8, 9 and 10 show the extra overhead as a function of the number of neighbors per node, the number of participants and elapsed time, respectively. As Figure 8 shows, the tree-based method can achieve zero extra cost, because tree-based method only pushes in a top-down manner without redundant traffic. For the other three methods, when the number of neighbors per node is 8, the push and pull methods perform better than DCO; as the number increases to 16, the three methods behave almost the same; as the number increases more, DCO presents the best performance. Moreover, the extra overhead of the push and pull methods mount up when there are more neighbors per node, while that of DCO decreases. When the number of neighbors climbs to 64, the overhead of DCO is almost one third of the push method and one fifth of the pull method. This result tells that DCO works better when there are more neighbors. This is because when a node has more neighbors, it needs to exchange messages with more nodes in the pull and push methods, but it needs less hops for forwarding its messages in DCO.

Figure 9 illustrates the relationship between extra overhead and the number of participants in the network, which is a measurement of scalability. The number of neighbors was set to 32. We can observe that the extra overhead of each method increases linearly as the number of nodes grows. The tree-based method again produces 0 extra overhead since its top-down chunk dissemination is the most efficient. DCO generates less extra overhead than the push method, which produces less extra overhead than pull. The reason is that the chunk lookup mechanism in DCO can always provide a valid provider to the requester, and it does not need frequent buffer map exchanges as in push and pull methods. This result also

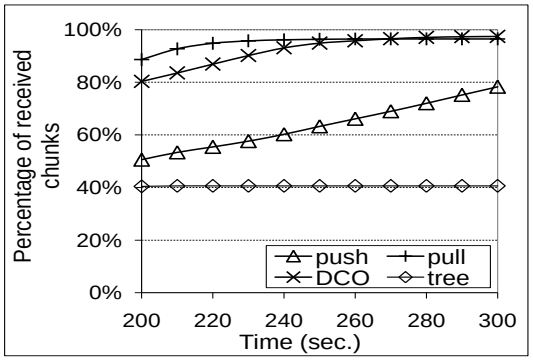

Fig. 11. Percent of received chunks vs. time.

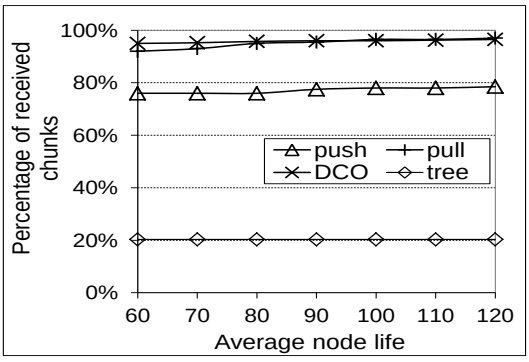

Fig. 12. Percent of received chunks vs. node life.

shows that DCO is more scalable than push and pull methods.

Figure 10 shows the extra overhead as time elapses. The four methods exhibit similar behaviors as in Figure 9. The result further verifies that apart from tree, which does not incur extra overhead, DCO produces the lowest overhead, showing its high efficiency. The reason why DCO always generates less extra overhead than other methods is that it can guarantee a node to receive its requested chunks while other methods cannot. In addition, DCO can provide better parent candidates. The periodic exchanges of buffer maps between neighbors generate significantly high extra overhead in the push and pull methods. The fact that the push method is better than the pull method is because the pull method needs one more step of request after buffer map exchange.

D. The Impact of Churn

In this experiment, we set the number of chunks to disseminate to 200, and allow up to 300 seconds for nodes to retrieve the chunks. Also, the node life span is set to an exponential distribution [32] with mean ranging from 60s to 120s, and the join interval of nodes is set to the same distribution. Therefore, nodes are constantly leaving and joining the network, and the network scale remains relatively stable.

Figure 11 shows the percentage of received chunks given chunk dissemination time ranging from 200s to 300s with increment of 10s in each step and node life span equal to 60s. It can be observed that DCO achieves comparable performance with the pull-based method. Nodes in DCO actively request missing chunks from the chunk providers, which enables them to obtain chunks in time. It can also be seen that DCO has a little lower performance compared with the pull method at the beginning. This is because the chunk spreading speed of DCO is first slowed down by the $\log n$ delay in DHT overlay routing, but this is soon remedied because as the number of chunk holders increases, the chunks can be disseminated much faster. The push-based method has slower speed in receiving chunks than DCO and the pull method. This is because nodes in the push-based method passively receive chunks from neighbors. Some chunks may not be sent quickly by certain nodes. The tree-based method has the worst performance because churn will break its topology, and a great number of nodes will not be able to receive chunks from their parents.

Figure 12 shows the relationship between the percentage of received chunks and the expected life of each node as the average node life ranges from 60s to 120s with 10s increment in each step. DCO and the pull method gain a higher percentage than the push-based method. This is because the primary goal of push is to distribute fresh chunks so that some nodes may depart before they have sent all the chunks to their neighbors. The tree-based method is not resilient to churn due to the reason in Figure 11.

The above experimental results of churn-resilience on the percentage of received chunks show that DCO achieves comparable churn-resilience as the pull-based method. Push-based method is also resilient to churn, while the fragile structure of the tree-based method makes itself vulnerable to churn.

V. Conclusion and Future Work

In this paper, we propose a DHT-aided chunk-driven overlay for P2P live streaming, targeting higher scalability, availability and low latency. The design consists a hierarchical DHT-based infrastructure and a chunk sharing algorithm. The hierarchical DHT-based infrastructure offers high scalability. The chunk sharing algorithm provides service for chunk index collection and discovery, which guarantees high availability. As a result, the overlay can provide high quality video streaming. DCO is superior over tree-based systems in dealing with churn, and mesh-based systems (pull and push) in bandwidth consumption and latency. More importantly, it can flexibly take full advantage of system bandwidth by dynamically matching chunk requesters and providers. The experimental results show that DCO improves the performance of the mesh-based systems and tree-based systems, in term of scalability, availability, latency and overhead.

In the future, the performance under churn and the efficiency of the overlay will be investigated from various aspects. Furthermore, an optimal peer selection algorithm will be developed by considering actual situations in the real world.

Acknowledgements

This research was supported in part by U.S. NSF grants CNS-1025652, CNS-1025649, and CNS-0917056, Microsoft

Corporation PO # 8300751, and Sandia National Laboratories, 10002282. We would like to thank Linlin Yang and Shuangyang Yang for their valuable help on this work.

REFERENCES

[1] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, 2007.

[2] "Total Viewers Of Online Video Increased 5 Percent Year-Over-Year," http://blog.nielsen.com/nielsenwire/online_mobile.

[3] "YouTube Still the King of Online Videos," http://www.searchenginejournal.com/.

[4] "1 Billion Online Video Viewers Served by 2013," http://www.straightupsearch.com/archives/2008/05/.

[5] "10 million people see U2 Concert on YouTube," http://mashable.com/2009/10/29/u2-youtube-10-million/.

[6] "PPLive," http://www.pplive.com.

[7] "Joost," http://www.joost.com.

[8] "SopCast," http://www.sopcast.com.

[9] "UUSee," http://www.uusee.com.

[10] Y.-H. Chu, S. G.Rao, and H. Zhang, "A Case For End System Multicast," in *Proceedings of ACM SIGMETRICS*, 2000.

[11] X. Zhang, J. Liu, B. Li, and T. P. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," in *Proc. of INFOCOM*, 2005.

[12] C. Wu and B. Li, "Exploring large-scale peer-to-peer live streaming topologies," *ACM Transactions on Multimedia Computing*, vol. 4, no. 3, 2008.

[13] F. Picconi and L. Massoulie, "Is there a future for mesh-based live video streaming?" in *Proc. of P2P*, 2008.

[14] J. Wang, C. Huang, and J. Li, "On ISP-Friendly Rate Allocation for Peer-Assisted VoD," in *Proc. of ACM Multimedia*, 2008.

[15] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. of SIGCOMM'02*, Pittsburgh, PA, USA, 2002.

[16] Y. Chu, A. Ganjam, T. Ng, S. Rao, K. Sripanidkulchai, J. Zhang, and H. Zhang., "Early experience with an internet broadcast system based on overlay multicast," in *Proc. of USENIX*, 2004.

[17] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proc. of ACM SIGMETRICS*, 2000.

[18] D. Tran, K. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *Proc. of INFOCOM*, 2003.

[19] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributed streaming media content using cooperative networking," in *Proc. of ACM NOSSDAV*, 2002.

[20] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *Proc. of SOSP*, 2003.

[21] R. Tian, Q. Zhang, Z. Xiang, Y. Xiong, X. Li, and W. Zhu, "Robust and efficient path diversity in application-layer multicast for video streaming," *IEEE TCSVT*, 2005.

[22] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: eliminating trees from overlay multicast," in *Proc. of IPTPS*, 2005.

[23] N. Magharei and R. Rejaie, "PRIME: peer-to-peer receiver-driven mesh-based streaming," in *Proc. of INFOCOM*, 2007.

[24] J. Venkataraman and P. Francis, "Chunkyspread: multi-tree unstructured peer-to-peer multicast," in *Proc. of IPTPS*, 2006.

[25] S. Asaduzzaman, Y. Qiao, and G. Bochmann, "CliqueStream: an efficient and fault-resilient live streaming network on a clustered peer-to-peer overlay," in *Proc. of P2P*, 2008.

[26] Y. Guo, C. Liang, and Y. Liu, "Adaptive queue-based chunk scheduling for P2P live streaming," in *Proc. of IFIP Networking*, 2008.

[27] L. Massoulie, A. Twig, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proc. of IEEE INFOCOM*, 2007.

[28] C. Liang, Y. Guo, and Y. Liu, "Is random scheduling sufficient in P2P video streaming?" in *Proc. of ICDCS*, 2008.

[29] A. Silva, E. Leonardi, M. Mellia, and M. Meo, "A bandwidth-aware scheduling strategy for P2P-TV systems," in *Proc. of P2P*, 2008.

[30] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A hybrid tree/mesh overlay for application-layer live video multicast," in *Proc. of IEEE ICDCS*, 2007, p. 49.

[31] F. Wang, J. Liu, and Y. Xiong, "Stable Peers: Existence, Importance, and Application in Peer-to-Peer Live Video Streaming," in *Proc. of INFOCOM*, 2008, pp. 1364–1372.

[32] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. 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.

[33] A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proc. of IFIP/ACM Middleware*, 2001, pp. 329–350.

[34] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and challenges of peer-to-peer internet video broadcast," in *Proc. of Special Issue on Recent Advances in Distributed Multimedia Communications*, 2007.

[35] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "AnySee: Peer-to-Peer Live Streaming," in *Proc. of IEEE INFOCOM*, 2006.

[36] T. Locher, S. Schmid, and R. Wattenhofer, "eQuus: a provably robust and locality-aware peer-to-peer system," in *Proc. of P2P*, 2006.

[37] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proc. of IEEE INFOCOM*, 2003.

[38] D. Ren, Y. H. Li, and S. G. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *Proc. of IEEE INFOCOM*, 2008.

[39] M. K. Yeung and Y. Kwok, "Game theoretic peer selection for resilient peer-to-peer media streaming systems," in *Proc. of ICDCS*, 2008.

[40] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and P. R., "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in *Proc. of STOC*, 1997, pp. 654–663.

[41] F. Wang, J. Liu, and Y. Xiong, "Stable peers: existence, importance, and application in peer-to-peer live video streaming," in *Proc. of IEEE INFOCOM*, 2008.

[42] Z. Liu, C. Wu, B. Li, and S. Zhao, "Distilling superior peers in large-scale P2P streaming systems," in *Proc. of IEEE INFOCOM*, 2009.

[43] F. Wang, Y. Xiong, and J. Liu, "mTreebone: a hybrid tree/mesh overlay for application-layer live video multicast," in *Proc. of ICDCS*, 2007.

[44] M. Bishop, S. Rao, and K. Sripanidkulchai, "Considering priority in overlay multicast protocols under heterogeneous environments," in *Proc. of IEEE INFOCOM*, 2006.

[45] D. R. Cox, "Regression models and life-tables," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 34, no. 2, pp. 187–220, 1972.

[46] X. Cheng and J. Liu, "Nettube: Exploring social networks for peer-to-peer short video sharing," in *Proc. of INFOCOM*, 2009.

[47] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *Proc. of SIGCOMM*, 2007.

[48] E. Setton, J. Noh, and B. Girod, "Low latency video streaming over peer-to-peer networks," in *Proc. of ICME*, 2006.

[49] "What is a computer ping test?" http://compnetworking.about.com/od/.../pingtest.htm.

[50] p2psim, http://pdos.csail.mit.edu/p2psim/.