This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

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

Haiying Shen*, *Member, IEEE,* Ze Li, *Student Member, IEEE,* Jin Li, *Fellow, IEEE*

**Abstract**—Internet-based video streaming applications are becoming more and more popular, attracting 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 suffer from high delay and overhead. Also, both structures 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 two-layer hierarchical DHT-based infrastructure, a chunk sharing algorithm and a video provider selection algorithm. It selects stable nodes to form a scalable DHT-based infrastructure. The nodes in the DHT serve as distributed matchmakers between video providers and requesters. In order to motivate stable nodes to serve as the DHT nodes, we introduce an incentive mechanism based on the game theory. Aided by DHT, DCO guarantees stream chunk availability and assigns to a chunk requester a provider among all available providers in the system so that stream chunks are transmitted along a dynamic tree with top-down decreasing node bandwidth. In this way, DCO takes full advantage of available bandwidth in the system and, at the same time, provides high scalability and low latency. Experimental results show the superior performance of DCO compared with mesh-based and tree-based systems, and the effectiveness of the incentive mechanism and provider selection algorithm.

**Index Terms**—DHT; P2P live streaming; Chunk-driven.

## 1 Introduction

Internet-based video streaming applications are becoming more and more popular and attract millions of online viewers every day [1]. The number of unique online video viewers increased 5.2% year-over-year, from 137.4 million 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 in 2009 [3], and that number is expected to rise to at least one billion viewers worldwide by 2013 [4]. Live streaming applications provide broadcasting streams from live channels such as TV and live events. Recently, peer-to-peer (P2P) techniques have attracted significant interest for live video broadcasting over the Internet. 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 media content of a certain length. The peers watching the same video program form an overlay for sharing the chunks of a video stream between each other. The P2P paradigm dramatically reduces the bandwidth burden on the centralized media server and generates more available bandwidth as the number of viewers increases. Typical P2P video streaming applications include PPLive [5], UUSee [6], ESM [7], and CoolStreaming [8]. As an example, UUSee simultaneously sustains 500 live stream channels and routinely serves millions of users each day [9]. 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) [10, 11]. The incredible growth of viewers, dynamics 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 and 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 demands requires that video streams are transferred under time and bandwidth constraints.

However, the capability of existing P2P live streaming systems is insufficient to tackle these challenges. Most current systems construct their overlays into either tree-based structures or mesh-based structures.

**Existing Methods.** In a tree-based structure [12–21], 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 the live stream before the tree is repaired. *(ii)* The fixed tree structure and stream flow direction make it difficult to easily 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 [8, 10, 22–31], each node maintains a list of their overlay neighbors. Each node also maintains a buffer map which summarizes the chunks that it currently has cached 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 approaches or push-based approaches. In the former, each peer sends requests to its neighbors to constantly look for new chunks; in the latter, a peer pushes chunks to its

- * *Corresponding Author. Email: shenh@clemson.edu.*
- *H. Shen and Z. Li are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. J. Li is with the Microsoft Research, Redmond, WA 98052.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

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)* Periodic information exchange generates high overhead and consumes a lot of bandwidth. *(iii)* Push-based approaches may cause a node to receive many identical chunks, producing extra overhead.

Several recently proposed works [32–37] 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.

**Our Proposed Method.** Distributed hash table (DHT) [38] is a structure for managing data 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 is suitable for channels with large-scale users. It includes a two-layer hierarchical DHT-based infrastructure, a chunk sharing algorithm, and a provider selection algorithm.

- A two-layer hierarchical DHT-based infrastructure. The infrastructure has two layers. In the upper layer, stable nodes form into a DHT, and in the lower layer, normal nodes connect to DHT nodes, which serve as deputy to the normal nodes in mapping chunk requesters and providers to guarantee content availability and low latency. A game theory based incentive mechanism motivates stable nodes to serve as the DHT nodes.
- A chunk sharing algorithm. Using the DHT functions indexed by chunk IDs, each node initially reports its available chunks to the DHT and sends chunk requests to the DHT. The DHT maps the providers and requests of the same chunk to the same coordinator DHT node, which locates an chunk provider for each requester, and the requester then stays connected with the provider for subsequent requests, so that stream chunks are transmitted along a dynamic tree with top-down decreasing node bandwidth.
- A provider selection algorithm. In order to build a dynamic tree with top-down decreasing node bandwidth for fluent stream flow, a coordinator DHT node considers total bandwidth, available bandwidth and buffering capacity in assigning chunk providers to requesters. A decentralized selection algorithm is also proposed to enable chunk providers and requesters to automatically matched up by themselves without relying on coordinators.

With the three components, DCO flexibly takes full advantage of available bandwidth in the system and provides high scalability, availability and low latency. As far as we know, this is the first work that leverages DHT to increase scalability and availability and reduces latency in P2P live streaming systems.

The rest of this paper is organized as follows. Section 2 gives a brief overview of the existing approaches for P2P live streaming systems. Sections 3 and 4 present the background of this work and the design of the DCO system in detail. Section 5 presents simulation results of DCO in comparison with other approaches. Section 6 summarizes the paper.

## 2 Related Work

P2P live streaming systems fall into three categories: tree-based [12–21], mesh-based [8, 10, 22–31] and hybrid structure, which combines both structures [32–37, 39].

**Tree-based structure.** The early P2P streaming solutions are single-tree based, such as Overcast [40], ESM [41], multicast overlay [13, 12], Narada [15], and ZIGZAG [16]. They feature a single multicast tree with the server at the root position. Each user joins in the tree at a certain level, and it receives the video from its parent and forwards the received video to its children. However, the single-tree approach suffers from sub-optimal performance of throughput and is vulnerable to churn. To deal with this problem, multi-tree based approaches, including CoopNet [17], SplitStream [18] and Bullet [42], have been proposed. In multi-tree streaming, the server divides the stream into multiple sub-streams. Each peer joins all trees to retrieve sub-streams. Within each tree, the corresponding sub-stream flows down level by level from the source server to all the leaf nodes. Thag *et al.* and Yin *et al.* [19, 21] employ multiple description coding to divide media contents into multiple sub-streams, which are delivered through multiple multicast trees. However, these mechanisms generate high maintenance cost and involve complex protocols. Tree-based systems rigidly determine node locations in the tree; thus, they are unable to flexibly adapt to continuous changes in bandwidth status and churn. Unlike the tree-based system that builds a fixed tree, DCO forms a dynamic tree in a distributed manner and ensures that each parent has sufficient bandwidth to transfer a stream to its children. DCO is also churn-resilient since a node can quickly find a new chunk provider when its current provider fails.

**Mesh-based structure.** A mesh-based system constructs a mesh out of the overlay nodes and swarms media contents by interchanging chunks with neighbors. At any given time, a peer has multiple neighbors, and can download/upload video from/to them. If a peer's neighbor leaves, the peer can download video from other neighbors. Examples of mesh-based systems include PPLive [5], UUSee [6], CoolStreaming [8], AnySee [23] and Chainsaw [24]. CliqueStream [22] builds a live streaming network on top of eQuus [25], which is a clustered locality-aware P2P overlay. Mol *et al.* [36] presented extensions to BitTorrent for supporting live video streaming. Carra *et al.* [29] studied the fundamental properties of stream-based content distribution services. To improve resource utilization between peers, a number of packet scheduling algorithms have been proposed, such as AQCS [26], RUPF [27] and DP/LU [10]. Their performances are compared extensively in [10, 43]. 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 to ensure chunk availability due to system-wide search using DHT.

**Hybrid structure.** A number of recently proposed approaches [32–37] combine tree and mesh structures to construct hybrid overlays. PRIME [32] is a two-phase mesh-based live P2P streaming system. It builds a tree with nodes 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 data segments of the chunk from their neighbors in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

the same level. Chunkyspread [33] forms multiple trees over a mesh. The server multi-casts a video stream to its neighbors, each of which is the root of each tree. Each node periodically checks whether its parent is overloaded and recommends a set of candidate nodes in the tree to its parent for replacement. Wang *et al.* proposed a two-tier structure [34, 35]. 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 a high utilization of bandwidth among overlay nodes. However, the hybrid structure still inherits the drawbacks of the tree-based and mesh-based system.

**Leverage heterogeneous bandwidth and stable nodes.** 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.* [44] used supernodes or dedicated proxies to provide efficient data distribution services to a set of end-hosts. In the method proposed in [45], peers with larger outgoing bandwidth adaptively move closer to the source to reduce the mesh delay of the entire system. Yeung and Kwok [46] proposed assigning more parent nodes to peers with large outgoing bandwidth in order to ensure they receive stream even in node dynamism. Contracts [47] provides contribution incentives in P2P live streaming systems. It rewards the globally effective node contributions by placing those nodes close to the server. Liu [37] derived the minimum delay bounds for P2P live streaming systems. He further proposed a snow-ball streaming algorithm to approach the minimum delay bound by exploiting bandwidth heterogeneity among peers. Wang *et al.* [35] indicated that stable nodes are important in P2P live video streaming systems using traces from PPLive and analytical models. Kumar [30] developed a stochastic fluid model that exposes the fundamental characteristics of a P2P streaming system, including the peers' real time demand for content, peer churn, peers with heterogeneous upload capacity, limited infrastructure capacity, and peer buffering and playback delay. Built upon these previous works, DCO forms nodes into a tree with top-down decreasing node bandwidth for fluent stream flow. DCO also forms stable nodes to a DHT to help all nodes in the system to efficiently retrieve chunks, achieving reliable video sharing.

## 3 Background

**P2P Live Video Streaming Systems.** In a P2P live video streaming system, live channel sources broadcast various media programs. A server in the live streaming network slices the media stream in a live channel into small chunks, each containing media contents of a certain length. The chunks are then delivered to the users in the channel. 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 the channel name plus its generation timestamp. For example, if a chunk has a session length of one second, the name of the chunk from the NBC channel beginning at 01:30:01 on January 1st, 2009 is labeled as *NBC20090101013001*. 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.

**DHT Systems.** 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 or file is assigned a unique ID that is the consistent hash value [48] 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 messages of the functions are 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 [38] in this work, although any other DHT system could also be adopted. 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(*$N_i$*)*.

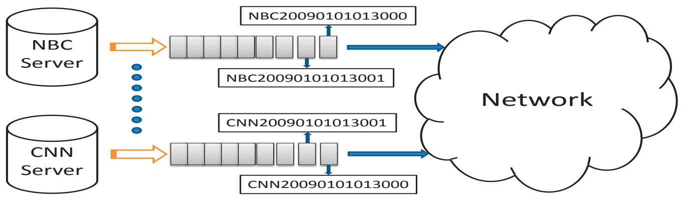


*Fig. 1:* The process of chunk production and delivery.

## 4 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 has a two-layer hierarchical DHT-based infrastructure as shown in Figure 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 the nodes' chunk information elegantly in the DHT so that a node can always find providers with sufficient outgoing bandwidth in a short time period. An active chunk in a node has an 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, *Insert(ID,index)*, and lookup, *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. Because the provider is also watching the same channel, it may continue to provide subsequent chunks as long as it is alive. Thus, the requester directly pulls the provider for subsequent chunks. 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. As each node receives chunks from a provider and offers chunks to others, thus the chunks spread out in the network in the fashion of a tree. In the following, we present the three components of DCO.

### 4.1 Two-layer Hierarchical DHT-based Infrastructure

DCO selects stable nodes to form a DHT ring structure for high chunk availability and QoS. We call these distributed nodes *coordinators*. There are two main reasons for choosing only the 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 any given time stays fairly constant. Second, coordinators must be stable. 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. Section 7 in the supplementary file presents the stable node selection method and incentive mechanism to encourage nodes to be coordinators.

#### 4.1.1 *Infrastructure Construction and Maintenance*

**Node Join.** In current P2P live streaming systems, the server keeps track of tens to hundreds of nodes in each channel. In DCO, the server 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 the DHT in DCO is not fixed, and 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. If a node in the upper tier is always lightly loaded for a certain period of time, it becomes a node in the lower tier. 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 $N_j$ its successor or predecessor in the DHT. The process of node joining in the DHT is the same as that in general DHTs. Then, $N_j$ becomes a coordinator and can directly communicate with other 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, builds its index table accordingly, and will handle all requests for chunks in its index table. Thus, partial load in $N_i$ is transferred to $N_j$. Similarly, if $N_i$ is always lightly loaded, it leaves the DHT and becomes a client.

**Node Departure.** A node may leave a channel either gracefully, by informing its neighbors, or abruptly, without any notice. As we will explain later, the chunks are transmitted along a tree in DCO. When $N_i$ gracefully leaves the network, it notifies nodes that are currently pulling chunks from it about its parent in the tree, so that they can directly connect to the parent for subsequent chunks. Meanwhile, it informs the coordinators to which it has previously reported its chunks, then the coordinators remove $N_i$ from their index tables. If $N_i$ is a coordinator, it needs to conduct three more operations. (1) It partitions its clients to two groups, and recommends *successor($N_i$)* and *predecessor($N_i$)* to each group for them to connect to. (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.

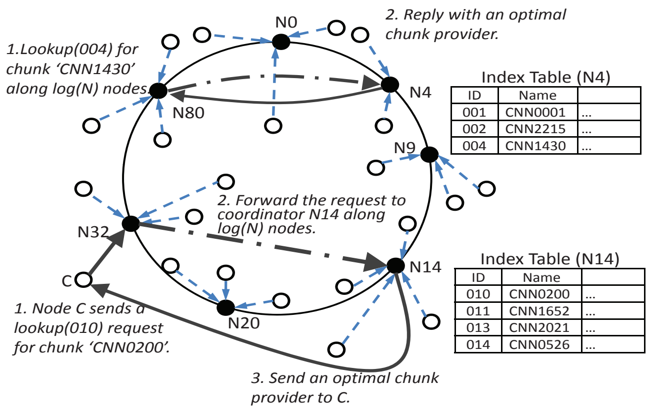

1.Lookup(004) for chunk 'CNN1430' along log(N) nodes.
N0
2. Reply with an optimal chunk provider.
N4
N80
Index Table (N4)
ID | Name
001 | CNN0001 | ...
002 | CNN2215 | ...
004 | CNN1430 | ...
N9
2. Forward the request to coordinator N14 along log(N) nodes.
N32
N14
C
Index Table (N14)
ID | Name
010 | CNN0200 | ...
011 | CNN1652 | ...
013 | CNN2021 | ...
014 | CNN0526 | ...
1. Node C sends a lookup(010) request for chunk 'CNN0200'.
N20
3. Send an optimal chunk provider to C.

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

**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 updates them if they are outdated.

### 4.2 Efficient Chunk Sharing

**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 2 shows an example of the DCO infrastructure along with index tables in coordinators $N4$ and $N14$. 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.

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. 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. The chunk index distribution based on the consistent hash function in the DHT leads to a comparatively balanced distribution, i.e., $\log N$ imbalance [38]. 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.

When a node needs chunks to play for a certain time and does not have any connection with a chunk provider, it consults

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

the DHT. Specifically, it calculates the ID of the chunk by applying the consistent hash function to the chunk's name. Then, it sends a *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. It 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 $C$ with an optimal chunk provider. The algorithm of the provider selection is presented in Section 4.3. A coordinator can directly send *Lookup(ID)* requests to the DHT for chunks. As the figure shows, the coordinator $N80$ sends a *Lookup(ID)* request to another coordinator $N4$. The coordinator processes the request and sends the information of a chunk provider to the requester.

After receiving the information of a provider from the coordinator, the requester sends a chunk request along with its buffer map to the provider. Then, in addition to the requested chunk, the provider also sends the requester its active chunks missed by the requester. The requester directly turns to this provider later on for its missing chunks. Thus, after the first chunk querying, the chunk transfer between the provider and the requester follows a direct pull-based style. If the provider cannot offer the chunk or the requester detects the provider's failure or departure through a timeout chunk request, the requester can always turn to the DHT for a new provider of the chunk. Therefore, chunk availability is guaranteed. 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 to the departed node's predecessor or successor 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.

**Tree Formation.** In DCO, the process of chunk spreading dynamically forms a tree structure. In Figure 3, we can see how a chunk is transmitted in a tree manner. The three layers in the figure follow a top-down time sequence. In the highest layer (earliest time), node $N80$ contacts coordinator $N4$. $N4$ responds with chunk provider $N14$, from which $N80$ pulls for the chunk. Later on, in the second layer, node $N32$ also requests the chunk, and coordinator $N4$ recommends $N80$ to $N32$ as the chunk provider. Then, $N32$ pulls the chunk from $N80$. After that, nodes $N20$ and $N0$ in the third layer request the chunk. If $N32$ is the recommended node, then it provides the chunk to $N0$ and $N20$. Consequently, as shown in the dotted arrows, the dissemination process of the chunk is in the fashion of a tree.

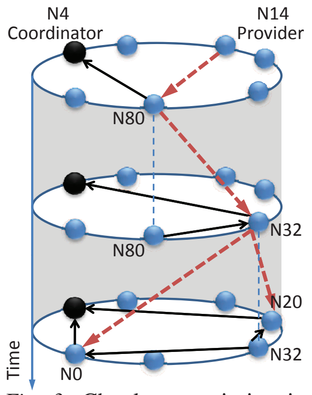


*Fig. 3:* Chunk transmission in a tree manner.

**Index Table Entry Size.** After a new chunk is produced by a server, it is spread to more and more nodes until every node has the chunk. Thus, at the beginning, a chunk is highly shared. As time goes on, the frequency that a chunk is shared decreases. It is a resource waste if a coordinator maintains all chunk indices. We set a threshold for the number of chunk indices in an entry to $\delta n$ $(0 < \delta < 1)$. Our experiment results showed that $20\%$ is an optimized number for $\delta$ that keeps high performance and reduces unnecessary messages. Every coordinator sets up a *flag* for each entry in its index table. When an entry is first created, the *flag* is set to zero. It turns to one when the number of chunk indices in the entry exceeds the threshold. Recall that a chunk's name indicates its timestamp and a chunk is discarded when it is out of a node's time window. If the coordinator receives a new chunk index, it replaces an old one with the new one. Therefore, the chunk indices in index tables are almost up-to-date. Also, the bandwidth of the reporters of the new index can be utilized so that the nodes for the old chunk index already connected with several requesters will not be overloaded. Since every one contacts the server when joining in the overlay, the server can keep a counter to record the number of nodes $n$ in the channel.

## 4.3 Chunk Provider Selection

It was indicated that there are around 50,000 concurrent peers at any time in the streaming overlay watching a popular channel such as CCTV1 [9]. This means a chunk needs to be distributed among so many peers in time for high QoS. In this section, we will discuss two ways for the nodes to select chunk providers: a centralized way and a decentralized way. Section 8 in the supplementary file introduces a prefetching mechanism in DCO.

### *4.3.1 Centralized Chunk Provider Selection*

In the centralized chunk provider selection algorithm, the chunk distribution speed depends on how a coordinator selects a provider for a chunk requester from the provider list, so that providers' bandwidth can be optimally used. That is, the selection method will not produce overloaded providers or idle providers. In order to choose the optimal chunk provider for each requester, a coordinator considers the following factors:

1) Total bandwidth. Recall that chunk dissemination in DCO is essentially in a tree manner. Arranging nodes with higher total bandwidth closer to the server allows chunks to be rapidly transmitted along the tree [45]. Thus, we aim to dynamically form a tree with top-down decreasing bandwidth.
2) Available bandwidth. A node with greater available bandwidth is preferred, since it can quickly and successfully provide chunks.
3) Buffering level. A node with a higher buffering level can provide the requester more consecutive blocks in the playback buffer, reducing subsequent chunk requests.

DCO divides the node total bandwidth into different levels, denoted by $L^{TB}$. To consider total bandwidth, a coordinator should choose providers whose $L^{TB} = min\{L^{TB} | L^{TB} > L^{TB}_r, \forall L^{TB}\}$, where $L^{TB}_r$ is the requester's $L^{TB}$.

Recall that a node $i$ reports its available chunk to a coordinator with its available bandwidth, denoted by $b_i$. To estimate a chunk provider's current available bandwidth, we need to consider the number of requesters it has been assigned for this chunk, denoted by $m_i$. The chunk provider may also offer other chunks. To take into account this factor, we use the elapsed time from the reporting time to the current time, denoted by $t$, to approximately estimate its bandwidth

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

consumption during $t$. We use $\frac{b_i-\gamma m_i}{t_i}$ to estimate provider $i$'s available bandwidth, where $\gamma$ is the average bandwidth consumed by continuously offering a chunk to a requester.

When a node requests a chunk provider from a coordinator, the coordinator first selects a set of chunk providers $s$ that have higher $L^{TB}$ than the requester. We use $\frac{L_i^{TB}-L_r^{TB}}{L_r^{TB}}$ to consider the level closeness of the requester and chunk provider $i$. We introduce the *recommendation degree* (denoted by $R_i$) to show the degree that a chunk provider $i$ should be recommended to the requester considering both total and available bandwidths. The coordinator calculates the $R_i$ of each provider $i$ in the node set $s$.

$$R_i = \begin{cases} \frac{b_i-\gamma m_i}{t_i} \times \frac{L_r^{TB}}{L_i^{TB}-L_r^{TB}}, \ i \in s \\ 0, \ otherwise. \end{cases} \tag{1}$$

The coordinator chooses a subset of nodes from $s$ that have the highest $R_i$, and then selects the node with the highest buffering level. The final selected node is the optimal chunk provider for the requester.

### 4.3.2 Decentralized Chunk Provider Selection

We propose a decentralized chunk provider selection algorithm, in which the chunk scheduling does not depend on the coordinators. Rather, the chunk providers and requesters are autonomously matched up by themselves in a decentralized manner. In the network, a number of nodes may ask the DHT for the same providers' information at the same time. There are several providers in a chunk index entry with different available bandwidths. How can a node select optimal providers efficiently in a distributed manner? Based on non-cooperative game, we design a decentralized provider selection strategy for requesters. The game players are a number of requesters asking for the same chunk and competing for the outgoing bandwidths of providers in a chunk index entry. If more than two nodes are assigned to a provider, the provider prefers to select a requester with larger outgoing bandwidth in order to increase the utilization of the transferred chunk. If the requesting is successful, the requester will receive the chunk from the provider and gain benefits. If a request is not responded by the provider, the requester gains no benefit. The basic strategy of our provider selection strategy is to select a provider with enough bandwidth while generating low probability that the requester's request will be rejected by the chosen provider. Specifically, a coordinator periodically sends all available providers to all requesters along with the information of available outgoing bandwidth of the providers. After a requester $i$ receives the information of providers and requesters from a coordinator, it chooses the provider satisfying $L^{TB} = min\{L^{TB} | L^{TB} > L_r^{TB}, \forall L^{TB}\}$, where $L_r^{TB}$ is the requester's $L^{TB}$. If the $L_r^{TB}$ is evenly distributed, the requesters will be evenly distributed to different providers. Therefore, a selected provider is unlikely to be overloaded and a requester's request is unlikely to be rejected and its chosen provider should have the bandwidth to serve it. In the high-level view of the entire system, all nodes form into different layers and the bandwidth of nodes decreases from the top to the bottom. This structure enables a video stream to flow fluently in the top-down manner.

## 5 Performance Evaluation

We developed our simulator based on P2PSim [49] and compared DCO with the pull-based, push-based, and tree-based methods. A DHT needs $\log N$ hops on average for message routing and $\log^2 N$ messages for handling a node join or departure. Higher $N$ leads to more delay and cost for the operations. Thus, we made $N$ the maximum by forming all nodes into a DHT in DCO to compare the worst performance of DCO with other methods. A higher-layer DHT formed by a subset of the nodes would lead to better performance than our presented performance of DCO. The default chunk provider selection algorithm was the centralized algorithm. In the push-based method, every node sends missing chunks to their neighbors regardless of whether they have received chunks from others; 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.

The number of nodes in the network was set to 512 unless otherwise specified. In the pull-based and push-based mesh overlays, every node was randomly connected with 8 to 64 peers with an increment of 8. In the tree-based method, in order to ensure fluent top-down stream flow, we set the number of children per node to $\frac{1}{8}$ of the number of neighbors per node in the other three methods. The default number of children per node in the tree-based method was set to 3, and the default number of neighbors per node in other methods was set to 48. Nodes in the pull/push-based method exchange buffer maps with their neighbors every second.

In the simulation, a chunk was 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 the 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 was overloaded, it would queue its chunks in its buffer and would not perform any chunk transmission until it had sufficient bandwidth.

In the experiments with churn, the node life span was set to an exponential distribution [50] with a mean ranging from 60s to 120s, and the join interval of nodes was set to the same distribution. Therefore, nodes are constantly leaving and joining the network, and the network scale remains relatively stable. We run each experiments 5 times and report the average as the final experimental results. We examine the following performance metrics.

(1) *Mesh delay*: The interval from the time when a chunk is generated at the server to the time it reaches all nodes. We report the average mesh delay of all chunks.

(2) *Fill ratio*: The ratio of nodes holding a chunk at a certain time. Unless otherwise specified, we report the average fill ratio of all chunks.

(3) *Extra overhead*: The number of communication messages other than video chunks after all chunks reach all nodes. In the push and pull methods, it includes messages for buffer map exchanges and subsequent requests. In DCO, it includes messages between nodes and coordinators, chunk owners and the server. 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.

(4) *Percent of received chunks*: The number of chunks successfully received by all nodes from peers rather than the server over the total number of chunks in churn.

Section 9 in the supplementary file presents additional experimental results for the incentives for being cooperative

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

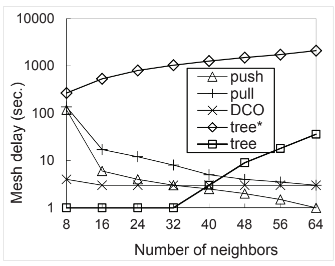


*Fig. 4:* Mesh delay vs. number of neighbors.

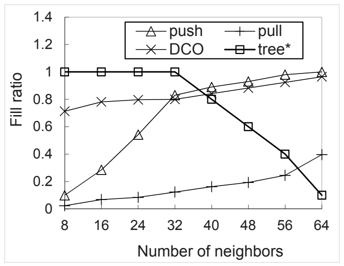


*Fig. 5:* Fill ratio vs. number of neighbors.

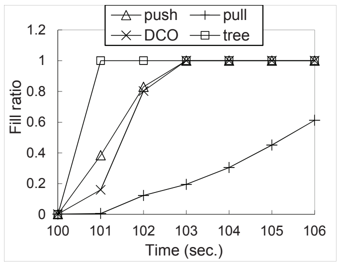


*Fig. 6:* Fill ratio vs. time.

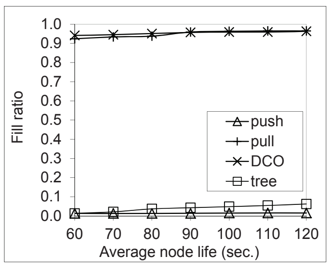


*Fig. 7:* Fill ratio vs. average node life.

coordinators and chunk provider selection method, as well as the performance of different methods on real-world PlanetLab.

## 5.1 Performance of Latency

Figure 4 shows the mesh delay versus the number of neighbors per node. Here "tree*" denotes a tree-based method in which the number of children of each node is the same as those in the other three methods, while "tree" denotes the tree-based method with our experiment setting. 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 with its $\log N$ hop routing delay. This is because that DCO can guarantee the availability of a chunk due to its system-wide search, while nodes in push/pull may not always receive 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 because the 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 the tree method 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, the tree method 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 in the tree method to $\frac{1}{8}$ of other methods.

## 5.2 Performance of Availability

Figure 5 shows the measured fill ratios two seconds after a chunk is generated versus the number of neighbors per node. Clearly, the performance of DCO is the most stable. The fill ratio of the push method grows sharply when the number of neighbors increases from 8 to 32, and beyond that it generates nearly the same fill ratio as DCO. This is because when a node has fewer neighbors, the push method needs more time to spread a chunk; but when a node has many neighbors, the push method can push the chunks to every node in the network in only a few steps, functioning as flooding. The pull method always shows the worst performance. A node has to pull from each of its neighbors for a chunk and then waits for the response. If the neighbor does not have the chunk, the requester needs to pull from another neighbor. For the tree 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 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 $\frac{32}{8}$, its fill ratio drops dramatically. This is because when the number of children is large, the bandwidth constraints significantly slow down the pushing process.

Recall the server creates 100 chunks in total. Figure 6 shows the fill ratio of the $100^{th}$ chunk every second starting from the time of 100 seconds. The figure shows that the fill ratio increases as time elapses. In the tree method, when the number of children is set to the default value 3, it achieves the fastest chunk spreading speed for the same reason as in Figure 5. 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 dissemination 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 dissemination speed by effectively utilizing more nodes' bandwidth. For the same reason as Figure 5, the pull method's fill ratio is significantly lower than others. It confirms the low speed of the pull method in spreading chunks.

The next experiment measures the fill ratio in churn. Figure 7 shows the fill ratio two seconds after a chunk is generated versus the average node lifetime in seconds. We can see that DCO and pull generate over 90% fill ratio which stays nearly constant. However, the tree and push methods produce less than 10% fill ratio and the ratio increases as the average node lifetime increases. In DCO, a chunk requester can always find a chunk provider as long as there exists at least one chunk provider. Thus, DCO provides high chunk availability. In the pull method, a node keeps pulling its neighbors, which enables a node always to receive its requested chunks. In the tree and push methods, a node passively receives chunks or information about available chunks. Shorter node lifetimes make many nodes unable to receive chunks, leading to a significantly lower fill ratio. We also observe that the fill ratios in all methods grow as the average node lifetime increases. This is because the located chunk providers have a lower probability of leaving before or while serving the requesters when the average node lifetime is longer. Also, in the tree and push methods, longer lifetimes make more nodes push chunks to other nodes before leaving, resulting in a faster increase in fill ratio than DCO

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

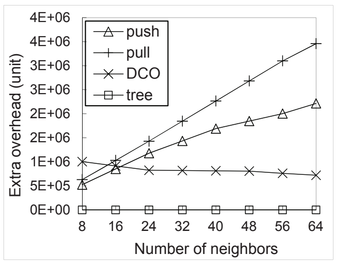


*Fig. 8:* Extra overhead vs. number of neighbors.

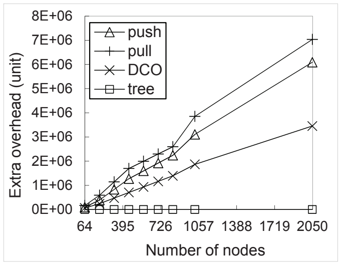


*Fig. 9:* Extra overhead vs. number of nodes.

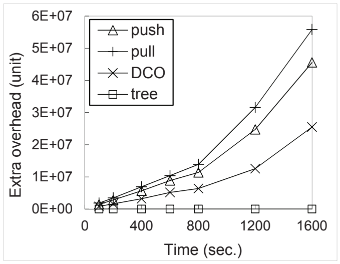


*Fig. 10:* Extra overhead vs. time.

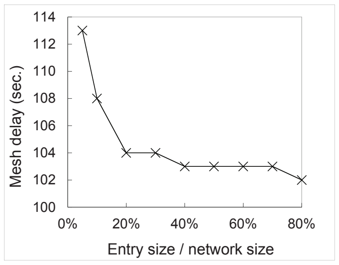


*Fig. 11:* Mesh delay vs. ratio of entry size.

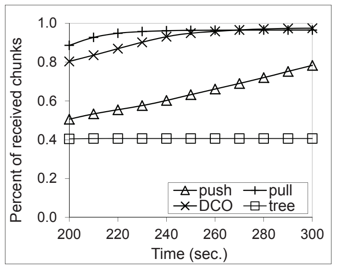


*Fig. 12:* Percent of received chunks vs. time in churn.

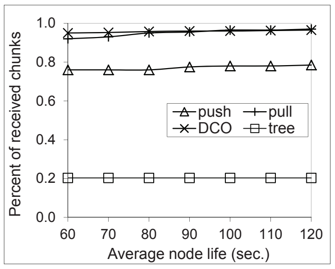


*Fig. 13:* Percent of received chunks vs. node life in churn.

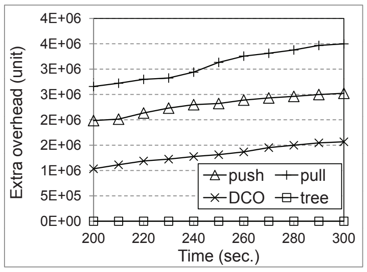


*Fig. 14:* Extra overhead vs. time in churn.

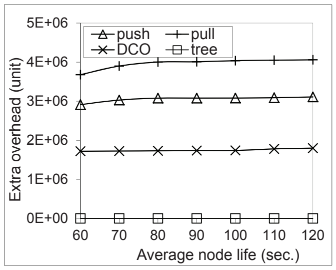


*Fig. 15:* Extra overhead vs. node life in churn.

and the pull method.

## 5.3 Performance of Scalability and Overhead

Figure 8 shows the extra overhead as a function of the number of neighbors per node. We see that the tree method can achieve zero extra cost, because it 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 shows 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 provider queries in DCO.

Figure 9 illustrates the relationship between extra overhead and the number of nodes in the network. The number of neighbors per node was set to 32. We observe that the extra overhead of each method increases linearly as the number of nodes grows. The tree method again produces no 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 the push and pull methods. This result shows that DCO is more scalable than the 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, indicating its high efficiency. The reason is because DCO can guarantee that a node will 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 push method generates lower extra overhead than the pull method because the pull method needs one more request step after a buffer map exchange.

## 5.4 The Impact of Chunk Indices

We measure how the number of chunk indices in one index table entry (i.e., entry size) affects the mesh delay. Through this experiment, the optimal number of chunk indices in a chunk's index entry can be found. Figure 11 shows the mesh delay versus the ratio of entry size over network size in DCO. When the ratio is five percent, its mesh delay is high. As the ratio increases, the mesh delay decreases sharply until the ratio is 20 percent. After reaching 20 percent, the mesh delay stays at roughly the same level and does not exhibit any more abrupt changes. In DCO, the coordinators offer chunk providers to requesters. Therefore, as the number of available chunk providers to be shared grows, the chunk requests can be disseminated to more nodes, hence the bandwidth utilization of nodes increases and mesh delay decreases. The result indicates that 20 percent of the number of all participants is the optimal size for one index table entry with efficiency consideration.

## 5.5 The Impact of Churn

Since the pull and push methods are mesh-based methods, they are naturally resilient to churn. We disable the stabilization in DCO in order to compare its worst case performance with others to show its superiority. There is also no tree structure update in the tree method. We set the total number of chunks to 200, and allowed up to 300 seconds for nodes to retrieve the chunks. Figure 12 shows the percentage of received chunks from the time instant of 200s to 300s with an increment of 10s in each step and a 60s average node life span. It can be observed that DCO achieves comparable performance with the pull approach. 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

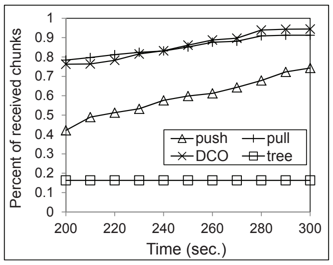


*Fig. 16:* Percent of received chunks vs. time with node failures.

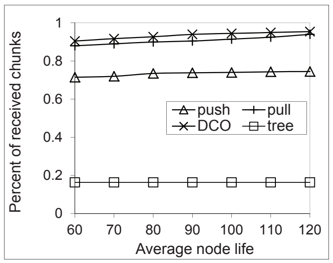


*Fig. 17:* Percent of received chunks vs. node life with node failures.

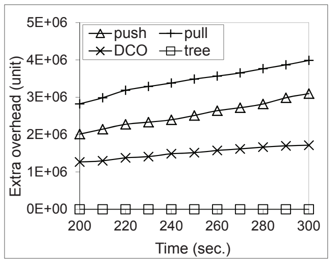


*Fig. 18:* Extra overhead vs. time with node failures.

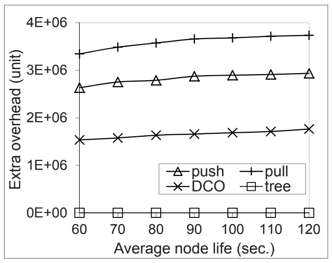


*Fig. 19:* Extra overhead vs. node life with node failures.

DCO is initially slowed down by the $\log N$ delay in DHT overlay routing; the slow-down is soon remedied because the chunks are disseminated faster as the number of chunk holders increases. The push method is slower in receiving chunks than DCO and the pull method. This is because nodes in the push-based method passively receive chunks from neighbors, and some chunks may not be sent quickly by certain nodes. The tree 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 13 shows the relationship between the percentage of received chunks and the average node life. DCO and the pull method gain a higher percentage than the push method, in which some nodes may depart before they send chunks to their neighbors. The tree method is not resilient to churn due to the same reason as in Figure 12.

Figure 14 shows that the extra overhead grows linearly as time goes on. The tree method produces no extra overhead, because the tree structure ensures that chunks are always disseminated from the server to the leaf nodes without other messages. DCO performs better than the push and pull methods because it can always locate the proper node for data transmission, which can reduce the massive communication overhead among each node and its neighbors. Here, the extra overhead increases over time because nodes keep requesting chunks during the time, which constantly generates extra overhead to DCO and the pull and push methods. Figure 15 shows the extra overhead as a function of node lifetime. It exhibits similar results as Figure 14, in which the tree method incurs no overhead due to its efficient top-down topology. DCO still exhibits better performance than the push and pull methods. The cost of all four methods stays relatively stable, because there is no maintenance cost for churn, so their extra message transmission costs are not sensitive to the length of node life. The above experimental results of churn-resilience on the percentage of received chunks show that DCO achieves churn-resilience performance comparable to that of the pull method. Further, the results on extra overhead show that DCO also features low overhead under churn.

### 5.6 The Impact of Node Failures

We then test the impact of node failures on chunk delivery performance and overhead. In this experiment, we changed the node departures to node failures (or abrupt departures) in the previous experiments in churn. Figure 16 shows the percentage of received chunks over time. We see that pull and DCO have the highest percentage of received chunks, which are followed by push, and tree generates the smallest percent of received chunks. The reasons are the same as in Figure 12. Figure 17 shows the percentage of received chunks of different systems versus the average node life. As the figure shows, DCO and pull still have the highest percentage of received chunks, and tree has the lowest percentage of received chunks. The reasons are the same as in Figure 13.

Comparing Figure 16 and Figure 12, and Figure 17 and Figure 13, we can see that the systems with node failures have much lower percentages of received chunks than the systems with graceful node departures. The main reason is that a node failure is detected only when a provider-requester connection timeout occurs, while in graceful node departures, a leaving node notifies nodes that are currently pulling chunks from it about its parent in the tree, so that the nodes can directly connect to the parent for subsequent chunks. The chunk retrieval delay in node failures leads to a lower percentage of received chunks.

Figure 18 shows the extra overhead of different systems versus time. We see that the extra overhead follows pull>push>DCO>tree. The reason remains the same as in Figure 14. Figure 19 shows the extra overhead of different systems versus the average node life. We see that pull has the highest overhead, which is followed by push and then DCO. Tree generates the least extra overhead. The reason is the same as in Figure 15.

Comparing Figure 18 and Figure 14, and Figure 19 and Figure 15, we see that the systems with failures produce much less overheads than the systems with graceful node departures. This is because a graceful node departure generates more communication between nodes than a node failure. A gracefully leaving node needs to notify other nodes about its departure, while a node failure is detected by the connection timeout. However, the less overhead of node failures than node graceful departures is at the cost of lower percentage of received chunks. In a conclusion, DCO can achieve high percent of received chunks with small extra overhead comparing to push, pull and tree.

## 6 Conclusion

In this paper, we propose a DHT-aided chunk-driven overlay for P2P live streaming that targets higher scalability, better availability and low latency. The design has three main components: a two-layer hierarchical DHT-based infrastructure, a chunk sharing algorithm, and a video provider selection 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. The provider selection algorithm enables full utilization of system bandwidth. As a result, the overlay can provide high quality video streaming. We use cooperative game theory to analyze the incentives that should be provided to stable nodes to encourage cooperative behaviors in the DHT-based infrastructure. We also propose a centralized and simplified decentralized provider selection algorithm. DCO

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

is superior to tree-based systems in dealing with churn and mesh-based systems 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 (pull and push) and tree-based systems, in term of scalability, availability, latency and overhead. The experimental results also confirm the importance of providing incentives to encourage nodes to serve as coordinators in the DHT-based infrastructure and the importance of selecting chunk providers with sufficient bandwidth in chunk delivery. In our future, we will study how to adopt the mechanisms that enhance the churn-resilience of tree-based methods into DCO.

## Acknowledgements

This research was supported in part by U.S. NSF grants OCI-1064230, CNS-1049947, CNS-1156875, CNS-0917056 and CNS-1057530, CNS-1025652, CNS-0938189, CSR-2008826, CSR-2008827, Microsoft Research Faculty Fellowship 8300751, and U.S. DoE's Oak Ridge National Laboratory including the Extreme Scale Systems Center located at ORNL and DoD 4000111689. An early version of this work was presented in the Proceedings of ICPP'10 [51].

## References

[1] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A measurement study of a large-scale P2P IPTV system. *TMM*, 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] PPLive. http://www.pplive.com.

[6] UUSee. http://www.uusee.com.

[7] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, 2000.

[8] 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.

[9] C. Wu and B. Li. Exploring large-scale peer-to-peer live streaming topologies. *TOMCCAP*, 4(3), 2008.

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

[11] J. Wang, C. Huang, and J. Li. On ISP-friendly rate allocation for peer-assisted VoD. In *Proc. of ACM Multimedia*, 2008.

[12] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of SIGCOMM*, 2002.

[13] 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.

[14] T. T. Do, K. A. Hua, and M. A. Tantaoui. P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. In *Proc. of ICC*, 2004.

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

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

[17] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanid-kulchai. Distributed streaming media content using cooperative networking. In *Proc. of ACM NOSSDAV*, 2002.

[18] 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.

[19] 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.

[20] J. Liu and M. Zhou. Tree-assisted gossiping for overlay video distribution. *Multimedia Tools and Applications*, 2006.

[21] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. In *Proc. of MM*, 2009.

[22] 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.

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

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

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

[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 INFOCOM*, 2007.

[28] W. P. K. Yiu, X. Jin, and S. H. G. Chan. VMesh: Distributed segment storage for peer-to-peer interactive video streaming. *IEEE J-SAC*, 2007.

[29] D. Carra, L. Cigno, and E.W. Biersack. Graph based analysis of mesh overlay streaming systems. *J-SAC*, 2007.

[30] R. Kumar, Y. Liu, and K. Ross. Stochastic fluid theory for P2P streaming systems. In *Proc. of INFOCOM 2007*, 2007.

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

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

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

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

[35] 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.

[36] J. Mol, A. Bakker, J. Pouwelse, D. Epema, and H. Sips. The design and deployment of a bittorrent live video streaming solution. In *Proc. of ICM*, 2009.

[37] Y. Liu. Delay bounds of chunk-based peer-to-peer video streaming. *TON*, 2010.

[38] 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. *TON*, 2003.

[39] M. Zhou and J. Liu. A hybrid overlay network for video-on demand. In *Proc. of ICC*, 2005.

[40] J. Jannotti, D. Gifford, K. Johnson, and M. Kaashoek. Overcast: Reliable multicasting with an overlay network. In *Proc. of OSDI*, 2000.

[41] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *J-SAC*, 2002.

[42] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemincation using an overlay mesh. In *Proc. of SOSP*, 2003.

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

[44] 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 INFOCOM*, 2003.

[45] 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.

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

[47] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe. Contracts: Practical contribution incentives for P2P live streaming. In *Proc. of NSDI*, 2010.

[48] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of STOC*, 1997.

[49] p2psim. http://pdos.csail.mit.edu/p2psim/.

[50] C. Vassilakis and I. Stavrakakis. Minimizing node churn in peer-to-peer streaming. *Computer Communications*, 2010.

[51] H. Shen, L. Zhao, Z. Li, and J. Li. A dht-aided chunk-driven overlay for scalable and efficient peer-to-peer live streaming. In *Proc. of ICPP*, 2010.

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

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

[54] D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society*, 34(2):187–220, 1972.

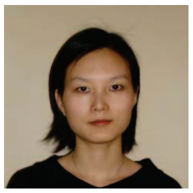
[55] M. J. Osborne and A. Rubinstein. *A course in game theory*. The MIT Press, July 1994.

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

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

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



**Haiying Shen** Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the Department of Electrical and Computer Engineering at Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.

**Ze Li** Ze Li received the BS degree in Electronics and Information Engineering from Huazhong University of Science and Technology, China, in 2007. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include distributed networks, with an emphasis on peer-to-peer and content delivery networks. He is a student member of IEEE.

**Jin Li** Jin Li is currently a Principal Researcher managing the Multimedia Communication and Sto9rage team at Microsoft Research, (Redmond, WA). He received his Ph.D. from Tsinghua University (Beijing, China) in 1994. After working at USC and Sharp Labs of America, he joined Microsoft Research in 1999. Dr. Li has published in top conferences and journals in a wide area, cover audio/image/video compression, multimedia streaming, VoIP and video conferencing, P2P networking, distributed storage system with erasure coding and deduplication, high performance storage system design. His invention has been integrated into many Microsoft products, such as Microsoft Office Communicator/Lync, Live Messenger, Live Mesh, Windows 7, Windows 8, etc.. He holds 45 issued US patents. He is the lead TPC Chair for ICME 2011.