InstantLeap: An Architecture for Fast Neighbor Discovery in Large-Scale P2P VoD Streaming

Xuanjia Qiu[1], Wei Huang[1], Chuan Wu[1], Francis C.M. Lau[1], Xiaola Lin[2]
[1]Department of Computer Science, The University of Hong Kong, Hong Kong,{xjqiu,whuang,cwu,fcmlau}@cs.hku.hk
[2]Department of Computer Science, Sun Yat-Sen University, P. R. China, linxl@mail.sysu.edu.cn

Abstract

In large-scale peer-to-peer (P2P) Video-on-Demand (VoD) streaming applications, a fundamental challenge is to quickly locate new supplying peers whenever a VCR command is issued, in order to achieve smooth viewing experiences. For many existing commercial systems which use tracker servers for neighbor discovery, the increasing scale of P2P VoD systems has overloaded the dedicated servers to the point where they cannot accurately identify the suppliers with the desired content and bandwidth. To avoid overloading the servers and achieve instant neighbor discovery over the self-organizing P2P overlay, we design a novel method of organizing peers watching a video. The method features a light-weight indexing architecture to support efficient streaming and fast neighbor discovery at the same time. *InstantLeap* separates the neighbors at each peer into a streaming neighbor list and a shortcut neighbor list, for streaming and neighbor discovery respectively, which are maintained loosely but effectively based on random neighbor list exchanges. Our analysis shows that *InstantLeap* achieves an $O(1)$ neighbor discovery efficiency upon any playback "leap" across the media stream in streaming overlays of any size, and low messaging costs for overlay maintenance upon peer join, departure, and VCR operations. We also verify our design with large-scale simulation studies of dynamic P2P VoD systems based on real-world settings.

Index Terms

P2P streaming, video-on-demand, fast neighbor discovery, indexing overlay

I. Introduction

Peer-to-peer (P2P) Video-on-Demand (VoD) streaming has been successfully deployed over the Internet [1]–[3], providing an abundance of online videos to hundreds of thousands of users. Unlike the traditional client-server approach which tends to incur enormous server costs [4], P2P VoD applications allow peers watching the same video to exchange available media blocks among themselves, and can therefore dramatically alleviate the server load [5]–[7].

Most state-of-the-art P2P VoD streaming applications adopt mesh-pull based P2P protocols. Each application typically consists of multiple mesh overlays, each of which connects the peers watching the same video. The peers in an overlay request available video blocks in their local buffers from each other, based on the exchanged buffer availability bitmaps (*i.e.*, buffer maps). The buffer at each peer represents a sliding window of the video stream, containing the block it is currently playing (referred to as its *playback position* hereinafter) and a number of blocks the peer has just watched or is about to watch in the near future.

When compared to P2P live streaming which has more mature applications in deployment, P2P VoD streaming presents a fundamental technical challenge to the designers: Given a certain video, the users (peers) could be watching different parts of the video, and may issue VCR commands (*e.g.*, pause, random seek) at will to "leap" to new playback positions (referred to as the *target playback positions*), which leads to a polarization of the buffered and needed contents among the peers and consequently the necessity to frequently search for new supplying peers. In order to provide the user with a smooth playback experience, the *re-buffering delay* from the time when a VCR command is issued to the time when playback resumes needs to be minimized, to a level comparable to that of VCR operations with a videocassette recorder, or to the channel switching delay when watching TV. In existing P2P VoD systems, this re-buffering delay can be as long as 18 seconds on average [8], which certainly needs improving in the future.

The *re-buffering delay* in P2P VoD streaming is typically composed of the time to search for new supplying peers and the time for downloading and buffering new blocks from them. While existing research work by others have investigated reducing the block downloading delay using such methods as optimized block scheduling algorithms [9], this paper aims at minimizing the neighbor discovery delay which in most situations is the dominant component in the re-buffering delay. In particular, we propose the design of a novel overlay architecture, with which given any target playback position, a set of new supplying peers can be found as fast as possible; these supplying peers (1) have the available blocks around the target playback position and (2) can aggregately provide the *leaping peer* (*i.e.*, the peer that is switching to a new playback position) with sufficient upload bandwidth to match the required streaming rate.

For neighbor discovery upon playback "leaps", existing commercial P2P VoD systems have largely resorted to tracker servers which keep track of the block availability at all the peers, and the peers would query the server for available serving peers. The disadvantages of such a centralized mechanism are: first, the tracker server can easily become a bottleneck if peers join, depart, and issue VCR commands frequently; second, in order to return a set of serving peers that can provide sufficient upload bandwidth to a requesting peer, the load on each individual peer needs to be periodically reported to the tracker server as well, which may further overload the server.

A few recent P2P VoD proposals construct different overlay structures for neighbor discovery in a decentralized fashion, using DHTs (Distributed Hash Tables) [10]–[12], dynamic skip list [13], and ring-based overlays [14]. These schemes typically involve $\log(N)$ complexity (where N is the total number of peers) for each lookup of supplying peers with the required blocks, without taking their upload bandwidth sufficiency into consideration.

In this paper, we propose *InstantLeap*, a new architecture of organizing peers watching the same video in a P2P VoD application. The overlay structure is simple but efficient, supporting both effective streaming and *instant* neighbor lookup due to playback "leaps". In *InstantLeap*, peers are grouped according to their playback locality. Each peer strategically maintains connections to a number of peers with similar playback progress, as well as some selected peers watching different parts of the video. The neighbors with similar playback progress provide efficient streaming of media blocks the peer is currently watching and are referred to as *streaming neighbors*; the neighbors watching other parts of the video facilitate fast discovery of new supplying peers with sufficient aggregate upload bandwidth to serve the leaping peer, which are named as *shortcut neighbors*. The list of shortcut neighbors is maintained loosely but effectively based on random neighbor list exchanges, which effectively pushes the neighbor discovery delay to the shortest possible.

The highlights of our overlay architecture design are as follows.

▷ We show an $O(1)$ efficiency for discovering new supplying neighbors upon peer joins and playback leaps across the video stream. In most cases, the identified supplying peers in the target playback position can provide sufficient aggregate upload bandwidths and are ready to transmit media blocks immediately after the new connections are established.

▷ We show an $O(m)$ overlay maintenance overhead for dealing with peer dynamics, including joins, failures/departures, and any playback leaps upon various VCR operations, where m is the number of peer groups. We show that such an increment of control overhead in a typical mesh-pull based P2P VoD protocol is negligible, as compared to the overhead in the basic protocol.

▷ InstantLeap can be seamlessly integrated into the mesh-pull protocol employed in prevailing P2P VoD systems, as a simple add-on component for overlay management, which is implemented based on random exchanges of neighbor information to build shortcuts among peers watching different parts of a video.

The remainder of this paper is organized as follows. Sec. II discusses related work on P2P VoD streaming and indexing overlay design. In Sec. III, we present our network model and the architecture of *InstantLeap* overlay. In Sec. IV , we discuss the detailed protocols in *InstantLeap*. In Sec. V, we analyze the performance and overhead of the protocols. We evaluate *InstantLeap* by extensive simulations and comparisons against existing schemes in Sec. VI, and conclude the paper in Sec. VII.

II. Related Work

Riding on the prevailing success of P2P file sharing and live streaming applications, a number of studies have been proposed to take the advantage of the P2P paradigm to provide VoD service. Most existing P2P VoD streaming protocols fall into two main categories: one of them is tree-push based protocols which organize peers into multiple multicast trees, and distribute the media streams by pushing data from the tree root downward [15]–[18]. However, such tree-push based protocols suffer from their low resilience to volatile peer dynamics. The other category includes the mesh-pull based protocols, such as BitOS [19], BASS [20], Toast [5], and Redcarpet [21], which are based on block "swarming" [22] over mesh overlay topologies. PPLive [1], UUSee [2], and PPStream [3] are early successful deployments of such mesh-pull based P2P VoD service.

In contrast to P2P live streaming, many problems related to peer asynchrony and VCR operations in P2P VoD streaming are more challenging [10], [13], [14], [23], [24]. One of the fundamental challenges is to minimize the re-buffering delay after a VCR command is issued.

Some studies propose to use prefetching to minimize the re-buffering delay [10], [23], which may require placing anchors throughout the media stream. VOVO [24] explores the association rules between two segments—the one a user is playing and another one the user might jump to—and prefetches the potential destination segments. In general, prefetching takes up extra bandwidth which may otherwise be used to improve streaming quality, which may not be accurate due to the difficulty in predicting a future segment to watch. Other efforts on minimizing the re-buffering delay focus on the improvement of the block selection and source selection algorithms [9].

For new supplying peer discovery without prefetching, existing commercial P2P VoD systems [1]–[3] have largely resorted to tracker servers. In P2P file sharing applications, a peer only reports to a tracker server when it joins or departs. In responding to a request for neighbors, the tracker server provides a random subset of peers in the system to the requesting peer, which may work well since each peer is interested in the entire file. The case in P2P streaming applications is quite different: to ensure a low streaming delay, tracker servers have to receive more frequent updates from all peers in the system on their current segment availability, in order to provide a refined list of neighbors to the requesting peers. CollectCast [25] and oStream [18] require peers to report their playback progress to the tracker server periodically. Kangaroo [9] designs an algorithm to estimate the playback progress of peers, which may introduce inaccuracy. In addition, the protocol does not guarantee that the peers found may have sufficient upload bandwidth to serve the leaping peer, which is one of the focuses in our design.

To alleviate the load on tracker servers, DHT (Distributed Hash Table) has been adopted in a number of recent P2P VoD proposals [10]–[12] to map block locations to peers. In general, each DHT lookup takes $log(N)$ steps, where N is the number of peers in the system, and DHT updates are required whenever the cached blocks are changed at the peers as their playback progresses.

A few recent proposals suggest different overlay structures to implement neighbor lookup without the complexity and cost of constructing a DHT. Wang *et al.* [13] utilize a dynamic skip list (DSL) to construct a P2P VoD overlay, where all the peers are connected sequentially according to their playback progress at the base layer of the skip list, and each peer may also randomly connect to a few non-adjacent peers in the higher layers. A $\log(N)$ complexity is shown for each lookup over the skip list. Chi *et al.* [26] suggest the use of an AVL tree for peer indexing, which can achieve a search efficiency sublinear to the number of peers. Cheng *et al.* [14] propose a ring-assisted overlay management scheme, where each peer maintains a set of concentric rings with power-law radius and places neighbors on the rings based on how similar their cached contents are. This overlay structure promises to achieve an $O(\log(T/w))$ lookup complexity (T and w are the video size and the buffer size, respectively), but a rigorous proof is missing.

RanSub [27] is one of the earliest work to achieve load balancing among network nodes via random peer selections. We also make use of the general idea of biased random neighbor selection at peers. Nevertheless, RanSub provides a general scalable mechanism for delivering state about a random subset of global participants in large-scale networks, while *InstantLeap* makes use of random peer selection to implement fast neighbor discovery and load balancing at VoD peers at the same time.

In this paper, we aim to design an efficient overlay architecture and associated protocols which can achieve $O(1)$ lookup efficiency upon any playback leaps, and which can provide sufficient streaming bandwidth for any new playback position with high probability.

III. Architecture

Unlike live streaming where a peer's playback position can only move continuously forward, VoD streaming allows the users to freely change their playback position to any random point in the video stream. Such random *playback leaps* necessitates quick discovery of new supplying peers which are able to provide video blocks at the target playback position and have the upload bandwidth to actually serve these blocks. To achieve the fastest possible neighbor discovery, *InstantLeap* presents a simple but efficient P2P overlay architecture, corresponding to the streaming of one video to N peers.

A. Peer grouping with playback locality

We partition the video stream into m consecutive segments along the time axis. Each segment consists of multiple media blocks. A peer is marked as a member of group i if its current playback position falls into the i^{th} segment. Peers in the same group (*e.g.*, group i), and those in the two adjacent groups (*e.g.*, groups $i-1$, $i+1$), may have overlapping buffer contents and are thus potential supplying peers for one another.

Each peer in group i maintains two neighbor lists: the first list (referred to as the *streaming neighbor list*) contains a subset of peers within the same group i as well as those in the two adjacent groups $i-1$ and $i+1$; the second list (referred to as the *shortcut neighbor list*) includes peers that are not in group i nor the two groups adjacent to group i, the connections to which serve as shortcuts to reach other parts of the video stream upon playback leaps. A conceptual model of the overlay design is illustrated in Fig. 1 (A). The discovery of peers to be maintained in the neighbor lists is based on a random exchange protocol, to be discussed in Sec. IV.

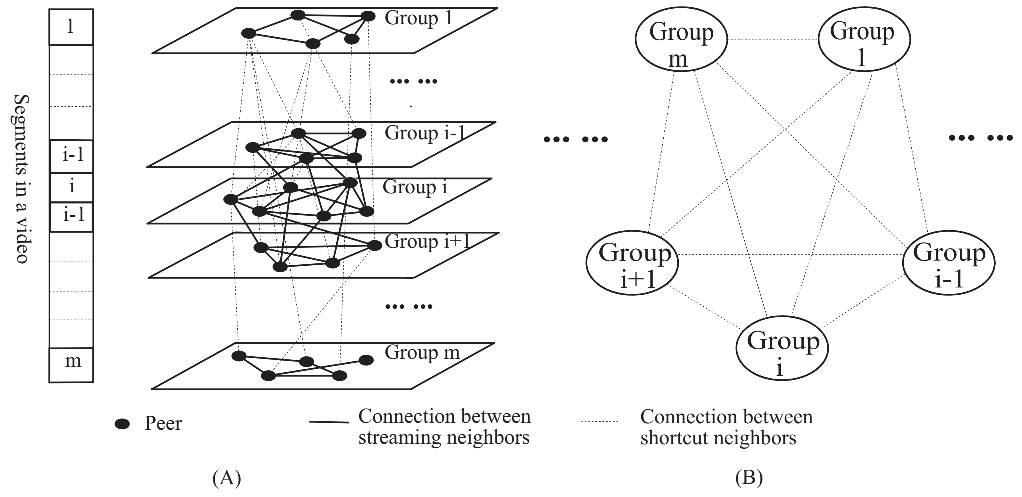

Fig. 1. P2P VoD overlay architecture in *InstantLeap*.

The two neighbor lists at a peer facilitate efficient streaming from neighbors in the same group or adjacent groups, and meanwhile enable fast discovery of new supplying peers in other destination groups whenever a playback leap occurs, by following connections to the shortcut neighbors. By maintaining neighbors in a random subset of all groups at each peer, we decouple the complexity of inter-group neighbor discovery from the total number of peers in the overlay, N; the complexity is reduced to at most a simple function of the number of segments in the video, m. What is more, in the following, we show that the complexity of neighbor discovery can be independent of m as well.

B. Inter-group connectivity

We can represent each group of peers in Fig. 1 (A) by one graph node and merge all the connections across groups into one; the condensed overlay graph is shown in Fig. 1 (B). In practical large-scale P2P VoD applications, a streaming overlay of a video can be populated by thousands of peers or more, with a large number of peers in each group. The node degrees in the condensed overlay graph in Fig. 1 (B) can be much larger than the number of inter-group connections at a single peer. Therefore, given a reasonable number of neighbors at each peer, the condensed overlay graph can turn out to be a dense graph, or even a complete graph. Hence the number of hops between any two nodes in the condensed graph would tend to be small, *i.e.*, $O(1)$ with high probability. For a peer currently at segment i who initiates a playback leap to a non-adjacent segment j, the complexity of finding a neighbor in the destination group j is proportional to the number of hops between the nodes concerned in the condensed graph,

TABLE I
NOTATIONS IN *InstantLeap* PROTOCOLS

N	The total number of peers in the overlay
p_i	A peer in the overlay, $i = 1, \ldots, N$
m	The total number of groups
$G(k)$	The set of peers in Group k, $k \in \{1, \cdots, m\}$
$U(p_i)$	Upload capacity of peer p_i
$N_u(p_i)$	The set of neighbors p_i is uploading to
$N_{sc}(p_i)$	The shortcut neighbor list at p_i
$N_{st}(p_i)$	The streaming neighbor list at p_i
$R(p_i)$	Streaming bandwidth requirement at p_i
β_1	Minimum fraction of groups a peer's shortcut neighbors span
β_2	Maximum fraction of groups a peer's shortcut neighbors span
T	Upper limit on the number of times a peer exchanges neighbor lists with others, in order to discover a peer in a destination group
L	Upper limit on the number of times a peer exchanges neighbor lists with others, in order to obtain more shortcut neighbors in one group

i.e., $O(1)$ with high probability. After a neighbor in the destination group is located, the leaping peer can acquire more streaming neighbors that can aggregately provide sufficient bandwidth for its media streaming, by exchanging neighbor lists with the established neighbor in the group.

We present the detailed protocols for realizing the architecture in the Sec. IV, and extensive analysis of the protocols in Sec. V, which shows that when each peer has a reasonable number of shortcut neighbors (*i.e.*, $O(m)$), an $O(1)$ complexity for discovering supplying peers in any new playback position with sufficient upload bandwidth can be achieved with high probability.

IV. INSTANTLEAP: PROTOCOLS

InstantLeap assumes a P2P VoD streaming framework similar to that of most state-of-the-art mesh-pull based P2P streaming protocols (*e.g.*, CoolStreaming [28], UUSee [2], PPLive [8]): New peers are introduced into a streaming overlay by a bootstrapping tracker server; they then stream the video by retrieving needed blocks from neighbors based on exchanges of buffer maps, and may request neighbor lists from each other to learn about more peers in the overlay.

One of the original features of *InstantLeap* is the construction of streaming and shortcut neighbor lists based on simple exchanges of known neighbors among the peers. These lists facilitate efficient streaming and fast neighbor discovery upon playback leaps with low additional protocol overhead. In this section, we first discuss the core neighbor list construction protocols in *InstantLeap* and then present the detailed procedures to handle peer join, departure, and playback leap.

A. Neighbor List Construction

Based on random neighbor list exchanges between peers, the core procedures of constructing the streaming neighbor list and the shortcut neighbor list at a peer, respectively, are given in **Algorithm 1**. The notations used in **Algorithm 1** and those hereinafter are summarized in Table I.

In *InstantLeap*, the number of neighbors in the streaming neighbor list of a peer is a constant, *e.g.*, 30–50, as typically used in prevailing P2P streaming protocols [1]–[3]. The number of groups the peer has shortcut neighbors in is in the range of $[\beta_1 m, \beta_2 m]$, where $0 \leq \beta_1 < \beta_2 \leq 1$: our *ConstructShortcutNeighborList* procedure would stop finding neighbors in new groups when the shortcut neighbors the peer has obtained span $\beta_2 m$ different groups; this procedure will be restarted when the number of groups has fallen under $\beta_1 m$.

In our design, a peer may maintain multiple shortcut neighbors from the same group, for two purposes: (1) to maximally guarantee a sufficient aggregate streaming bandwidth when its playback "leaps" to the destination group,

Algorithm 1 Basic Procedures at Peer p_i in *InstantLeap*

```
1: procedure ConstructStreamNeighborList()
2:     while my streaming neighbor list $N_{st}(p_i)$ is not full
3:         randomly select a peer $p_j \in N_{st}(p_i)$
4:         obtain the streaming neighbor list from $p_j$, i.e., $N_{st}(p_j)$
5:         merge $N_{st}(p_j)$ into my streaming neighbor list $N_{st}(p_i)$
6:     end if
7: end procedure
1: procedure ConstructShortcutNeighborList()
2:     while the number of groups my current shortcut neighbors belong to is smaller than $\beta_2 m$
3:         randomly select a peer $p_j$ from my streaming or shortcut neighbor list $N_{st}(p_i) \cup N_{sc}(p_i)$
4:         obtain the shortcut neighbor list from $p_j$, i.e., $N_{sc}(p_j)$
5:         merge $N_{sc}(p_j)$ into my shortcut neighbor list $N_{sc}(p_i)$
6:     end while
7:     for each group $k$ I have shortcut neighbor(s) in
8:       if the SNM Condition has not been satisfied and the number of times for neighbor list requests has not exceeded $L$
9:         randomly select a peer $p_j$ from my shortcut neighbor list that belongs to group $k$
10:        obtain the streaming neighbor list from $p_j$, i.e., $N_{st}(p_j)$
11:        merge $N_{st}(p_j)$ into my shortcut neighbor list $N_{sc}(p_i)$
12:      end if
13:    end for
14: end procedure
```

such that the peer can immediately start playback with satisfactory smoothness; (2) as our shortcut neighbor lists are maintained loosely (*i.e.*, with a low update frequency in cases of peer departures and playback leaps to incur low maintenance overhead), such a redundancy in shortcut neighbors maintained for each group provides *robustness* in fast neighbor discovery, *i.e.*, with high probability at least one neighbor can be successfully identified that still caches the segment corresponding to the destination group.

The number of shortcut neighbors from the same group that a peer maintains is decided by: the streaming quality the peer aims to obtain, the available upload bandwidth of peers from the group, and the upper limit L on the number of neighbor list exchanges a peer can carry out. In particular, peer p_i decides whether it should obtain more shortcut neighbors from group k by examining the following *Shortcut Neighbor Maintenance Condition* (*SNM Condition*):

$$\sum_{p_j \in N_{sc}(p_i) \cap G(k)} \frac{U(p_j)}{1 + |N_u(p_j)|} \geq R(p_i). \tag{1}$$

The left side of inequality (1) represents an estimate of the aggregate upload bandwidth peer p_i can obtain from its current shortcut neighbors if its playback “leaps” to group k, where $G(k)$ represents the set of peers in group k, $U(p_j)$ denotes the upload capacity at peer p_j, and $N_u(p_j)$ is the set of neighbors p_j is concurrently uploading to. Here we estimate the available supplying bandwidth from p_j to p_i by evenly dividing the upload capacity of p_j among all its receivers. Information on the upload capacity and the number of concurrent uploading peers can all be exchanged among peers with their neighbor list exchanges and status probing at low frequencies. $R(p_i)$ denotes the desired streaming bandwidth at peer p_i. For constant rate video streams, the streaming bandwidth is equivalent to the streaming rate of the video; in cases that layered coding or multiple description coding is employed [29], $R(p_i)$ can be set to different levels according to the number of layers/descriptions the peer may retrieve. Using the SNM condition, peer p_i wishes to maximally guarantee a sufficient streaming bandwidth when its playback “leaps” to a new group, within the limit of L neighbor exchanges.

By limiting the number of groups and the number of shortcut neighbors per group, we aim to achieve a good tradeoff between neighbor discovery efficiency and maintenance overhead. Further discussions and appropriate

Algorithm 2 Join Procedure at Peer p_i in *InstantLeap*

```
1: procedure PEERJOIN($PlayPosition$)
2:     Calculate group ID $k$ corresponding to $PlayPosition$
3:     Request a few neighbors from the tracker server and add them into $N_{st}(p_i)$ and $N_{sc}(p_i)$ according to their group membership
4:     while my streaming neighbor list is empty, i.e., $N_{st}(p_i) == \varnothing$, and the number of times for neighbor list requests has not exceeded $T$
5:         Randomly select a peer $p_j$ from my shortcut neighbor list $N_{sc}(p_i)$
6:         Obtain the shortcut neighbor list $N_{sc}(p_j)$ from peer $p_j$
7:         Merge peers in $N_{sc}(p_j)$ into my streaming neighbor list $N_{st}(p_i)$ or shortcut neighbor list $N_{sc}(p_i)$
8:     end while
9:     if no neighbor with group ID $k$ has been discovered
10:        Request a neighbor with group ID $k$ from a tracker server
11:    end if
12:    ConstructStreamNeighborList()
13:    ConstructShortcutNeighborList()
14: end procedure
```

values of these parameters will be presented in Sec. V.

Main protocols in *InstantLeap* can be implemented based on the procedures in **Algorithm 1**.

B. Peer Join

With *InstantLeap*, we seek to minimize the load on the tracker servers. When a peer first joins a streaming overlay, the number of existing peers assigned to it can be as small as one or a few. These bootstrapping peers are added to the new peer's streaming or shortcut neighbor lists, according to their group membership. Following the procedure of *PeerJoin* in **Algorithm 2**, if none of the assigned neighbors is in the group where the peer's desired playback position falls into, the new peer discovers a peer in the destination group by exchanging neighbor lists with the few known neighbors. Then the new peer executes *UpdateStreamNeighborList()* to obtain more neighbors with similar playback progress for video streaming, and *UpdateShortcutNeighborList()* to establish shortcuts to segments across the entire video stream, as described in **Algorithm 1**.

C. Continuous playback

When a peer watches the video continuously, its group membership changes whenever it moves on to play the next video segment. The peer notifies all its neighbors in the two lists with its new group membership. Nevertheless, there would be little changes to its streaming neighbor list, when most of its streaming neighbors are pursuing a continuous playback and they can continue serve media blocks to each other. In addition, most of its shortcut neighbors remain intact as well, except the few that have switched to the streaming neighbor list. Only when the number of groups its shortcut neighbors span falls below $\beta_1 m$ will the *ConstructShortcutNeighborList* procedure be invoked again.

D. Playback Leap

When there is a playback leap due to a VCR operation, the procedure of *PlaybackLeap* in **Algorithm 3** is executed. In the procedure, a peer p_i looks for neighbors in the destination group by exchanging neighbor list with its existing neighbors. When it still fails to discover such a neighbor peer after T exchanges of neighbor lists, it queries the tracker server as the last resort. T is a protocol parameter, which represents the maximum number of times a peer exchanges neighbor lists with others, in its attempt to discover a supplying peer in the destination group, before it resorts to the tracker server. We discuss an appropriate choice of T based on our analysis in Sec. V.

After connecting to discovered peers in the destination group, the peer starts retrieving video blocks right away, since the shortcut neighbors maintained at each peer maximally guarantee a sufficient streaming bandwidth to a

Algorithm 3 Playback Leap Procedure at Peer p_i in *InstantLeap*

```
1: procedure PlaybackLeap($DestPosition$)
2:     Calculate group ID $k$ corresponding to $DestPosition$
3:     while there is no peer with group ID $k$ on my shortcut neighbor list $N_{sc}(p_i)$ and the number of times for neighbor list requests has not exceeded $T$
4:         Randomly select a peer $p_j$ from my streaming neighbor list $N_{st}(p_i)$ or shortcut neighbor list $N_{sc}(p_i)$
5:         Obtain the shortcut neighbor list $N_{sc}(p_j)$ from peer $p_j$
6:         Merge peers in $N_{sc}(p_j)$ into my shortcut neighbor list $N_{sc}(p_i)$
7:     end while
8:     if no neighbor with group ID $k$ has been discovered
9:         Request a neighbor with group ID $k$ from a tracker server
10:    end if
11:    Clear my streaming neighbor list $N_{st}(p_i)$
12:    Add the discovered neighbors with group ID $k$ into my streaming neighbor list $N_{st}(p_i)$
13:    Start streaming from the current streaming neighbors
14:    ConstructStreamNeighborList()
15:    ConstructShortcutNeighborList()
16: end procedure
```

new requesting peer, based on the procedure of *ConstructShortcutNeighborList*. Meanwhile, the peer executes *ConstructStreamNeighborList* to obtain more new streaming neighbors, and *ConstructShortcutNeighborList* to update its shortcut neighbor list.

The leaping peer p_i may reside in the streaming or shortcut neighbor lists of other peers. When peer p_i has changed its group membership, those peers can still keep peer p_i in their respective streaming or shortcut neighbor lists, as long as p_i still caches the segment corresponding to its previous group. In this way, a peer which has cached multiple segments in its local buffer can effectively serve peers in multiple groups, leading to maximal utilization of peer cache and bandwidth resources.

E. Peer departure and failure

When a peer p_i leaves a P2P VoD overlay gracefully, it will notify its streaming neighbors and shortcut neighbors, which remove the peer from their neighbor lists and may invoke *ConstructStreamNeighborList* or *ConstructShortcutNeighborList* to rebuild the respective list when necessary.

When a peer p_i unexpectedly fails, its information may still be cached at the streaming neighbor lists or shortcut neighbor lists of some other peers. Such outdated information will be purged out of the streaming neighbor list of a peer when it notices the connection to the streaming neighbor p_i has been shut down, and can be removed from the shortcut neighbor list of a peer when it tries to connect to peer p_i upon playback leaps. The redundancy provided by *ConstructShortcutNeighborList*, in that typically more than one shortcut neighbor from the same group is maintained in a peer's shortcut neighbor list, guarantees there may still be available neighbors to resort to in case of such lagged updates.

V. Analysis

Although *InstantLeap* protocols appear to be simple add-ons to the existing typical mesh-based streaming protocols, we show in the following that an $O(1)$ neighbor discovery efficiency can be achieved upon any playback leap. We also carefully analyze the overlay maintenance overhead upon various peer operations in *InstantLeap*, and explain the tradeoff between neighbor discovery robustness and maintenance overhead.

A. Neighbor discovery efficiency

We first show that a peer can obtain shortcut neighbors across $O(m)$ groups by only a small number of neighbor list exchanges. In our analysis, we focus on the simplified case for each peer to obtain one shortcut neighbor from a group. To acquire more neighbors from the same group, we know at most a constant number L of neighbor list exchanges are involved, based on **Algorithm 1**.

Let s_i^t denote the average number of groups the shortcut neighbor list at peer i spans (referred to as "group span" hereinafter) after t times of random neighbor list exchanges. We first prove a lemma.

Lemma 1. *Let peer p_i and peer p_j be two randomly selected peers from all N peers in the streaming overlay, with an initial group span of the shortcut neighbor list of $s_i^0 = a$ and $s_j^0 = b$, respectively. The average group span of p_i's shortcut neighbor list after p_i exchanges neighbor list with p_j is $s_i^1 = s_j^1 = a + b - \frac{a \times b}{m}$.*

Proof: The average group span of the merged shortcut neighbor list is the sum of the group span of the shortcut neighbor list of p_i (*i.e.*, a) and that of the shortcut neighbor list of p_j (*i.e.*, b), minus the expected number of overlapping groups which is $\frac{a}{m} \times \frac{b}{m} \times m$. ∎

Based on Lemma 1, we have the following theorem.

Theorem 2. *Assuming $s_i^0 \in [K, (1+\alpha)K]$, where $K \geq 1$ and $0 \leq \alpha \ll 1$, for any peer p_i in the streaming overlay. After t times of shortcut neighbor list exchanges with randomly selected other peers in the overlay, the average group span of the shortcut neighbor list of p_i is $s_i^t \geq m[1 - e^{-\frac{2^t \times K}{m}}]$.*

Proof: Given that each peer's shortcut neighbor list has a similar group span, based on Lemma 1, the average group span of shortcut neighbor list at peer p_i after one exchange for neighbors with another random peer is $s_i^1 = 2s_i^0 - \frac{(s_i^0)^2}{m}$.

After $t+1$ times of exchanges between p_i and other peers, we can derive that

$$s_i^{t+1} = 2s_i^t - \frac{(s_i^t)^2}{m}, (t \geq 0),$$

Let $Q(t) = s_i^t - m$. We have

$$Q(t+1) = s_i^{t+1} - m = 2s_i^t - \frac{(s_i^t)^2}{m} - m = -\frac{1}{m}[s_i^t - m]^2 = -\frac{1}{m}[Q(t)]^2$$

We then have $Q(t) = -\frac{[Q(0)]^{2^t}}{m^{2^t - 1}} = -m(1 - \frac{s_i^0}{m})^{2^t}$, and thus we derive the expression for s_i^t:

$$s_i^t = Q(t) + m = m[1 - (1 - \frac{s_i^0}{m})^{2^t}]$$

Since $1 - \frac{s_i^0}{m} \leq e^{-\frac{s_i^0}{m}}$ with $0 \leq s_i^0 \leq m$, we have

$$s_i^t \geq m[1 - e^{-\frac{2^t \times s_i^0}{m}}] \geq m[1 - e^{-\frac{2^t \times K}{m}}]$$

∎

Corollary 3. *Assuming initially $s_i^0 = 1$ for any peer i in the streaming overlay, after $log(m)$ times of shortcut neighbor list exchanges with randomly selected other peers, the average group span of the shortcut neighbor list of peer i is $s_i^{\log m} \geq (1 - \frac{1}{e})m \approx 0.63m$.*

This corollary tells us that even in the extreme cases that each peer is assigned with only one neighbor initially, after a small number ($\log(m)$) of neighbor list exchanges, the peer can obtain shortcut neighbors covering more than half (0.63) of all the groups.

$\log(m)$ is generally a very small value. Based on the corollary, we know that a peer will have a shortcut neighbor list spanning no less than $(1 - \frac{1}{e})m$ groups after a few exchanges after joining the overlay. As a side note, in our protocol design discussed in Sec. IV-A, parameter β_2 can be set to a value of $\frac{2}{3}$ considering this effect. In addition, we can set the maximum number of neighbor list exchanges in order to build the shortcut neighbor list, T, to be $\log(m) \times \frac{1}{(1-\frac{1}{e})} = \frac{e}{e-1}\log(m)$, representing the expected number of neighbor list exchanges in order to obtain a neighbor in each group.

In this case, when a peer, which has finished the joining procedure, initiates a playback leap, the probability that it already has a neighbor belonging to its destination group is at least $1 - \frac{1}{e}$. If there is no such a neighbor, according to our protocol in **Algorithm 3**, the peer will exchange neighbor lists with its current neighbors. The probability it can successfully obtain a shortcut neighbor within a specific destination group after v times of neighbor list exchanges is at least $(1-\frac{1}{e})(\frac{1}{e})^v$. Therefore, we can derive the following theorem on the expected number of neighbor list exchanges a peer needs upon a playback leap, in order to discover a shortcut neighbor in the destination group.

Theorem 4. *The expected number of neighbor list exchanges, for a peer which has finished its joining procedure, to find a shortcut neighbor to a destination group upon any playback leap, is $O(1)$.*

Proof: Consider any peer p_i in the overlay who makes a playback leap to destination group k. Let q denote the average probability that a peer's shortcut neighbor list includes a peer in the destination group. This probability equals the ratio of the average size of the shortcut neighbor list at a peer over the total number of groups (*i.e.*, m). In the case that p_i's shortcut neighbor list does not contain such a neighbor, the probability that p_i can obtain such a peer by one neighbor list exchange with another peer p_j, randomly selected from the overlay, is q, *i.e.*, the probability that p_j has a neighbor in group k. Therefore, the probability that a peer successfully obtains a neighbor in the destination group after v times of random neighbor list exchanges is $(1-q)^v \times q$, and the expected number of exchanges is $\sum_{v=0}^{\infty} v \times (1-q)^v \times q = \frac{1-q}{q}$.

Considering that the peers involved in the exchanges are not new joiners (*i.e.*, they have all finished their joining procedures), we have $q \geq 1-\frac{1}{e}$, and the expected number of exchanges is $\frac{1-q}{q} \leq \frac{1}{e-1} \approx 0.58$. Therefore, in general, the expected number of neighbor list exchanges upon any playback leap is $O(1)$. ■

In our analysis, we have assumed that all neighbor list exchanges occur between two peers randomly selected from the overlay. The random exchange in *InstantLeap* between a peer and one of its randomly selected neighbors represents the best possible approximation to the expected randomness in a practical P2P VoD system.

B. Overlay Maintenance Overhead

The overhead in *InstantLeap* protocols is due mainly to the exchange, construction and maintenance of neighbor lists in case of various peer operations, including peer joins, departures and various VCR operations. We next analyze such overlay maintenance overhead.

1) Peer Join: As shown in Corollary 3, a new joining peer to the overlay carries out at most $\log(m)+(1-\frac{1}{e})mL$ neighbor list exchanges in order to construct its shortcut neighbor list, in which $\log(m)$ represents the number of times it requests neighbor lists to obtain at least one neighbor in each of $(1-\frac{1}{e})m$ groups, and $(1-\frac{1}{e})mL$ is the upper bound for further neighbor list exchanges in order to acquire a few more neighbors in each group to satisfy the SNM condition.

For streaming neighbor list construction, a new joining peer carries out an expected number of $\frac{e}{e-1}\log(m)$ neighbor list exchanges in order to obtain at least one neighbor in the group it belongs to, and then a small number of exchanges to obtain more streaming neighbors, that is much smaller than the total number of streaming neighbors (*i.e.*, 30–50) a peer can maintain.

2) Continuous Playback: When a peer is continuously playing the video stream, the overlay maintenance overhead includes sending updates of its group membership to neighbors when it crosses group boundaries, and updating its neighbor lists to purge outdated neighbor information and adding new neighbors. Such overhead is generally much smaller than that for new neighbor list construction upon peer joins.

3) Playback Leap: After a peer's playback "leaps" to a new segment and it is connected to at least one neighbor in the destination group k, the peer reconstructs its streaming neighbor list and updates its shortcut neighbor list. The overhead in streaming neighbor list construction involves a small number of neighbor list exchanges with its known neighbors in group k (much smaller than 30–50) to obtain sufficient streaming neighbors. The update to its shortcut neighbor list is little as well, as the peer may only need to take the neighbors in group k out of its shortcut neighbor list and place a few of its previous streaming neighbors into the shortcut neighbor list instead.

4) Peer Departure/Failure: In cases of peer departures and failures, the overlay maintenance overhead is proportional to the size of the leaving peer's streaming and shortcut neighbor lists at $O(m)$, unless *ConstructStreamNeighborList* or *ConstructShortcutNeighborList* is invoked to rebuild the respective list at a neighbor.

C. Tradeoff between Neighbor Discovery Robustness and Overlay Maintenance Overhead

As discussed in Sec. IV-A, for each group a peer's shortcut neighbors span, it may maintain more than one shortcut neighbors, in order to provide robustness into the neighbor discovery process in practical systems with possibly outdated neighbor group membership information. We now analytically show the tradeoff between such robustness and neighbor list maintenance overhead in this case.

Let g denote the probability that a neighbor p_j peer p_i obtains for destination group k by neighbor list exchanges can no longer serve p_i the segment corresponding to group k, *e.g.*, in cases of p_j's playback leap and buffer replacement, or p_j's failure from the system. The probability that peer p_i has at least one valid neighbor in group k by maintaining l shortcut neighbors for the group is $1 - g^l$.

To maintain more than one shortcut neighbors for a group, there are additional L neighbor list exchanges (at most) at each peer to acquire more neighbors in the group upon peer join based on **Algorithm 1**, and $O(l)$ more overhead for shortcut neighbor list maintenance in cases of peer failure, departure, and playback leaps, when the number of shortcut neighbors maintained for the group is l.

As outdated neighbor information is mainly caused by unexpected failures of peers and segment replacement in peers' buffers, in a system with large buffer sizes (thus lower chances of segment replacement) and a low probability of unexpected failures (*e.g.*, wired networks), we may choose to maintain fewer shortcut neighbors per group at the peers; otherwise, more neighbors per group is more desirable. In this way, a best tradeoff between neighbor discovery robustness and overlay maintenance overhead can be achieved in systems of different scenarios.

VI. Performance Evaluation

We present evaluations of *InstantLeap* based on a P2P simulator we have developed. The simulator is implemented using Java, featuring a multi-threaded high-performance architecture, with supports for multiple event-driven time-outs. All peer dynamics, including playback leaps, joins and departures, are simulated with events scheduled at their respective times. With careful optimizations, our simulator can simulate large-scale P2P systems with $10,000$ or more simultaneous peers, distinguishing itself from representative existing P2P simulators [30] which may support $3,000$ peers at most.

In our evaluations, the streaming rate of videos via the overlay is 450 Kbps. The upload bandwidth at the peers ranges between 300–$10,000$ Kbps, following a Pareto distribution with shape parameter $K = 2$, which corresponds to a resource index ($RI = \frac{\text{average upload capacity per peer}}{\text{streaming rate}}$) of 1.3. Peers' lifetime follows an exponential distribution with an expected length of 30 minutes. Peers join the overlay following a Poisson arrival model, whose inter-arrival times follow an exponential distribution. The expectation of the inter-arrival times differs across the experiments where we tried different overlay sizes, in order to keep the total number of online peers at a similar level over time in each experiment. The interval between two playback leaps at each peer follows an exponential distribution with an expected length of 200 seconds. We experiment with videos of different lengths, varying from 40 minutes to 200 minutes. Each peer's buffer can cache media segments up to three minutes of playback. The number of groups (m) thus ranges from 40 to 200, accordingly. By default, $\beta_1 = \frac{1}{3}$, $\beta_2 = \frac{2}{3}$, and $T = 10$. Other than neighbor list exchanges upon different peer operations such as playback leap, we also implement periodical exchanges of neighbor lists among peers to update neighbor status during a peer's continuous playback: streaming neighbor lists are exchanged among streaming neighbors every 5 seconds; the default interval for shortcut neighbor list exchanges and shortcut neighbor status probing among peers is 60 seconds. These parameters are carefully selected to be consistent with the measurement results in some existing representative P2P VoD systems [8], [23].

For streaming blocks scheduling in our simulations, peers exchange buffer maps every 5 seconds. A peer requests blocks needed from its multiple neighbors, following a hybrid deadline-rarest-first block selection policy, as is done in [5].

A. Performance of Neighbor Discovery

Fig. 2 and Fig. 3 show the average number of neighbor list exchanges upon peer joins and playback leaps, respectively, in overlays of different sizes and for videos of different lengths. These numbers translate into the delay for neighbor discovery, when the protocols are implemented in practice. From both figures, we observe that the numbers of exchanges are fewer than four, and little change with the increase of the overlay size. This clearly confirms that *InstantLeap* achieves a constant neighbor discovery performance of $O(1)$, independent of the number of peers in the overlay. When a peer joins or its playback leaps, it can get a ready collection of streaming neighbors in the destination group from its neighbors or its own shortcut neighbor list. This provides users with short re-buffering delay and satisfactory viewing experience.

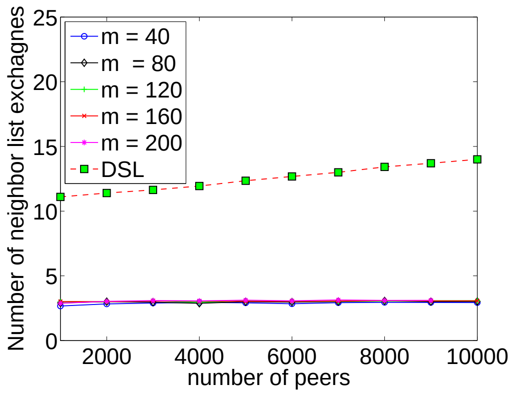

Fig. 2. Average number of neighbor list exchanges upon peer joins.

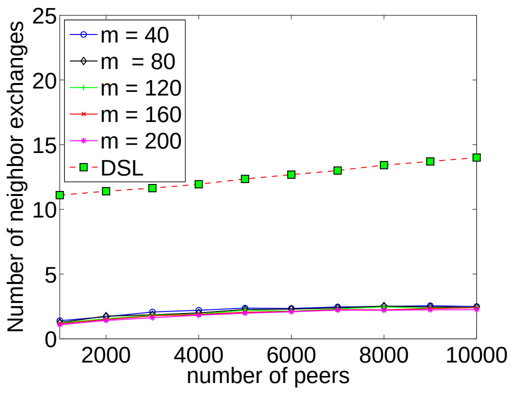

Fig. 3. Average number of neighbor list exchanges upon playback leaps.

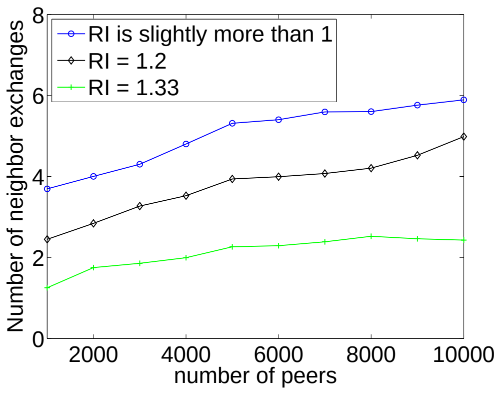

Fig. 4. Average number of neighbor list exchanges upon playback leaps with different resource indices.

Fig. 4 further shows the average number of neighbor list exchanges under different resource indices in overlays of different sizes (with $m = 60$). Streaming neighbor discovery is faster when the upload bandwidth in an overlay is more abundant. We observe that even when the upload capacity supply barely meets the demand (RI is around 1), a peer can still find sufficient suppliers with an average of 6 times of neighbor list exchanges in large overlays of up to 10000 peers.

In addition, we have also implemented the Dynamic Skip List (DSL) algorithm [13] for comparison purpose. In Fig. 2 and 3, the numbers corresponding to the DSL case represent the average numbers of search steps to discover a supplying peers in a DSL. We observe that *InstantLeap* generally requires just a few steps for neighbor discovery as compared to DSL, while DSL apparently requires many more steps with the increase of the overlay size.

B. Overlay Maintenance Overhead

In this subsection, we show that although *InstantLeap* maintains $O(m)$ shortcut neighbors and a constant number of streaming neighbors at each peer, the additional maintenance traffic incurred is indeed negligible, as compared to other control overhead and the streaming traffic in a typical mesh-based P2P VoD system.

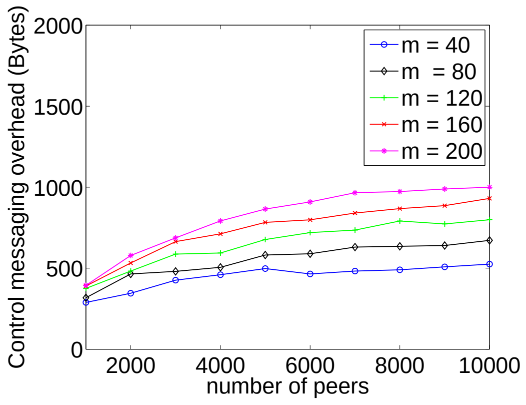

Fig. 5. Maintenance overhead upon peer joins.

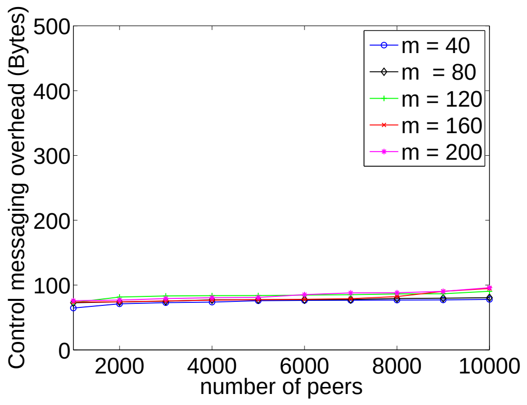

Fig. 6. Maintenance overhead upon playback leaps.

1) Overhead at peers: Fig. 5 and Fig. 6 plot the control messaging overhead for overlay maintenance upon peer joins and playback leaps, which depends on the number of control messages and the sizes of the neighbor list exchange messages. We have observed that upon peer joins, the total number of message bytes to build both the streaming neighbor list and the shortcut neighbor list is no more than a mere $1,000$ bytes. A comparison between Fig.5 and Fig. 6 shows that control messages incurred by a playback leap is much fewer than those caused by a peer join, as a leaping peer only needs to rebuild its streaming neighbor list while updating its shortcut neighbor list slightly.

Besides messaging overhead caused by a peer join or leap, Fig. 7 shows the messaging overhead incurred by periodical exchanges of neighbor lists and status probing among the peers. We observe that such messaging overhead is lower than 2.5 Kbps, equivalent to 0.5% of the streaming rate.

Comparing the overhead of *InstantLeap* to that of an existing protocol, such as DSL [13] and the ring-assisted solution [14], we find by comparing our results with those given in their respective papers, that those protocols generally incur larger messaging overhead in cases of peer joins and leaps, while *InstantLeap* requires more control message exchanges for maintenance of the overlay during playback.

2) Load on the tracker server: *InstantLeap* minimized the requests to tracker servers, which are only used as the last resort when a peer fails to identify a streaming or shortcut neighbor in the destination group after T times of neighbor list exchanges. In our implementation, the tracker server maintains information about 100 peers only in all the overlays of different sizes. Fig. 8 shows that the control messaging overhead on the tracker server is less than 100 Kbps, which is very low indeed.

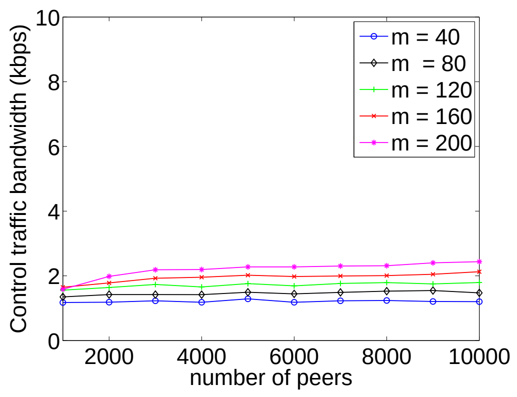

Fig. 7. Regular overlay maintenance overhead.

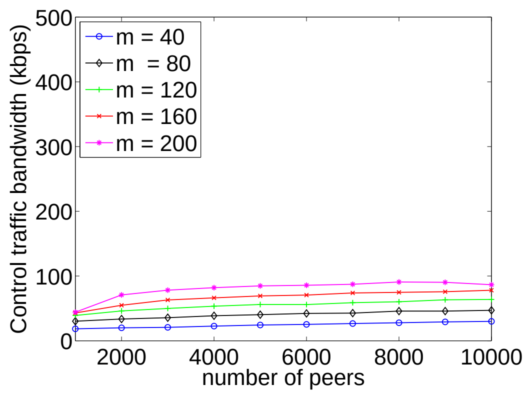

Fig. 8. Load on the tracker server.

C. *Streaming quality*

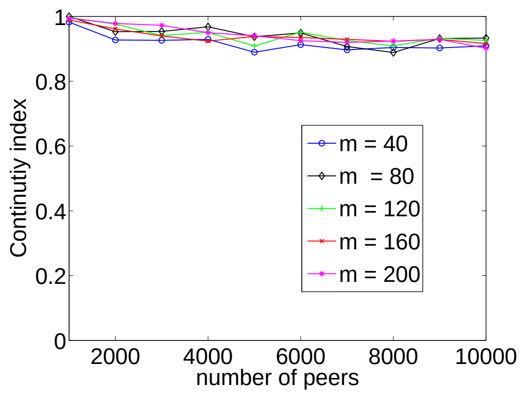

Fig. 9. Average streaming quality at peers.

We evaluate the streaming quality at the peers in *InstantLeap* by calculating a continuity index, which represents the percentage of blocks a peer receives before their respective playback deadlines during playback. In Fig. 9 we observe that the average continuity index is always larger than 0.9, showing that the new neighbors a peer connects to upon its playback leaps can provide a sufficient streaming bandwidth most of the times.

We are also interested to explore whether the overlay can adapt to severe peer dynamics. In this experiment, we have a large portion of peers leap to other playback positions simultaneously in an overlay of $10,000$ peers while watching a 60-minute video. In Fig. 10, we see that if 10% of all the peers leap concurrently, there is little impact on the average streaming quality of the peers, as most of them can find new neighbors very quickly. In the case that 50% of all the peers leap simultaneously, the average continuity index drops to 0.8, but picks up quickly again in less than 10 seconds.

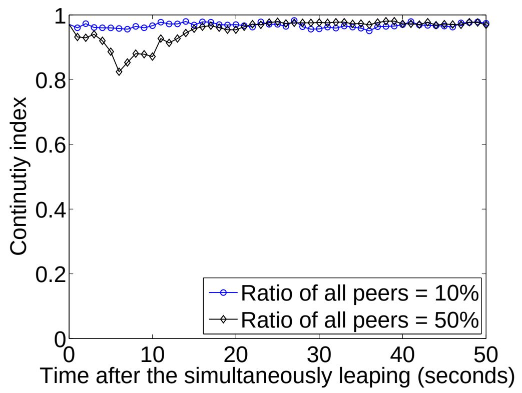

Fig. 10. Average streaming quality upon severe peer dynamics.

D. *Load Balance among Peers*

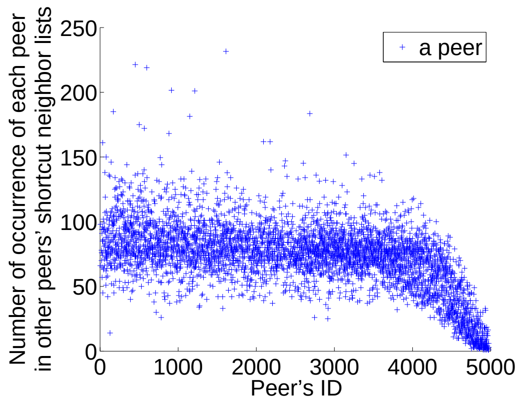

Fig. 11. Number of occurrence of each peer in other peers' shortchut neighbor lists (sorted by peer ID).

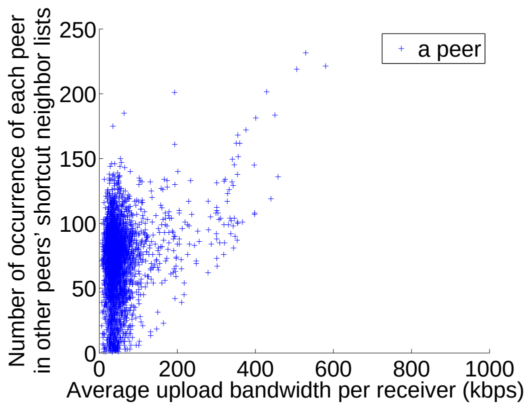

Fig. 12. Number of occurrence of each peer in other peers' shortchut neighbor lists (sorted by the peer's average upload bandwidth per receiver).

We next investigate the load on each peer to serve as shortcut neighbors for peers in other groups. In this experiment, we stream over an overlay of $5,000$ peers watching a 80-minute video, and count the number of occurrence of each peer in other peers' shortcut neighbor lists in the entire system. In Fig. 11, we observe that most peers serve as shortcut neighbors at a similar number of other peers. As peers with larger IDs join the overlay later, we also observe from Fig. 11 that peers with longer online time tend to be known by more other peers.

In Fig. 12, we plot the results again, against the average bandwidth that each peer can upload to each of its receivers. We find that if a peer can allocate more upload bandwidth to each of its streaming neighbors, it tends to be known by more peers. This helps the leaping peers to find new supplying peers with sufficient bandwidth more quickly. We also observe that the average bandwidth per upload link at the majority of peers lies within a

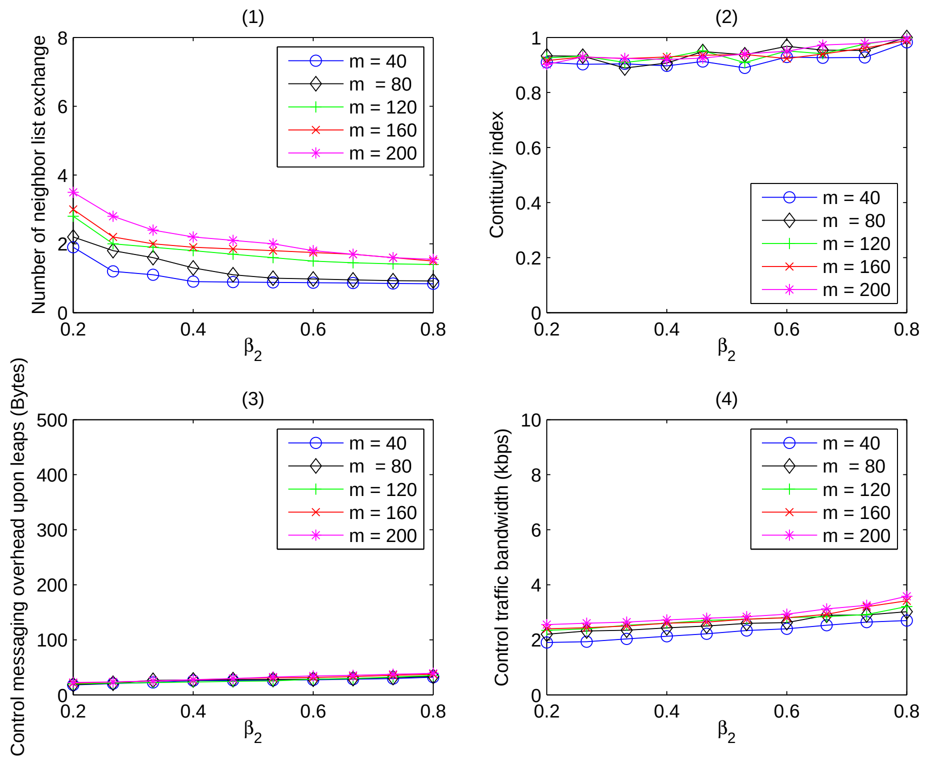

Fig. 13. Impact of different values of β_2.

relatively small range, which is due to the effective load balancing using the SNM condition: Based on SNM, peers tend to select potential shortcut neighbors from those that are currently uploading at high bandwidth to each of their streaming neighbors. In this way, a high-capacity peer will have a higher chance to be selected as a shortcut neighbor and subsequently to serve as a streaming neighbor; with the increase of streaming neighbors, its average upload bandwidth per link decreases, and thus its chance to be further selected as shortcut and streaming neighbors decreases, resulting in the relatively similar bandwidth per upload link at the majority of the peers.

E. *Impact of parameters*

1) Impact of the number of groups maintained: In *InstantLeap*, we impose a range on the number of groups spanned by shortcut neighbors at each peer, *i.e.*, $[\beta_1 m, \beta_2 m]$. We now investigate whether the number of groups affects the performance and overhead in *InstantLeap*. Fig. 13 plots the neighbor discovery performance and overlay maintenance overhead for different values of β_2, respectively. In all these experiments, we set $\beta_1 = 0.5\beta_2$, and the size of the overlay is $10,000$. We observe that the neighbor discovery performance becomes better when peers have shortcut neighbors in more groups (with the increase of β_2), which is at the cost of increased overlay maintenance overhead. A closer look reveals that the optimal value of β_2 is achieved at around 0.6, where there is a good balance between the performance and the overhead. This also explains our choice of using $\beta_2 = \frac{2}{3}$ by default in all of the experiments.

2) Impact of the number of shortcut neighbors maintained per group: We have analyzed the tradeoff between neighbor discovery performance and overlay maintenance overhead when a peer keeps different numbers of shortcut neighbors for each group in Sec. V-C. We verify our analysis by experiment: We stream over an overlay of 10000 peers watching a 40-minute video. There are 40 groups in this overlay and each group contains 250 peers on average. In Fig. 14, the number of shortcut neighbors shown is the maximum number of shortcut neighbors a peer can maintain per group. SNM conditions are applied for dynamic shortcut neighbor maintenance. We observe that increasing the maximum number of shortcut neighbors per group reduces the re-buffering delay upon playback leaps and improves the average streaming quality, with slight sacrifice in terms of messaging overhead.

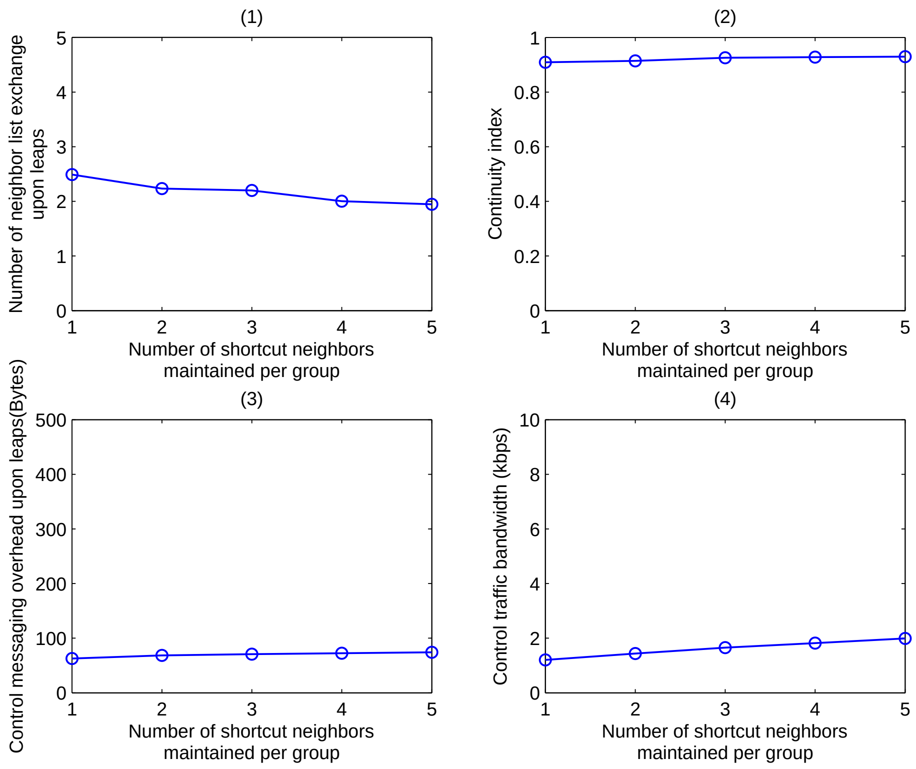

Fig. 14. Impact of the number of shortcut neighbors in each group

VII. Concluding Remarks

This paper proposes *InstantLeap*, a scalable light-weight indexing architecture to achieve efficient streaming and fast neighbor discovery for P2P VoD streaming applications. The highlight of *InstantLeap* is its neighbor discovery method which has a performance of $O(1)$ upon any playback leap with low overlay maintenance overhead. After each playback leap, the identified supplying peers can provide sufficient aggregate streaming bandwidth to the leaping peer with high probability, and would be ready to start transmitting media blocks immediately after the new connections are established.

InstantLeap can be implemented on top of the basic framework of prevailing mesh-pull based P2P VoD protocols, by adding the simple function of random neighbor list exchanges to maintain shortcut neighbors. The seemingly simple protocol achieves unexpectedly good neighbor discovery performance, thus providing good support for various VCR operations, which is validated by both theoretical analysis and extensive simulations with large-scale overlays and intense peer dynamics.

In the future, we may extend the current framework to solve the problem of bandwidth imbalance across the groups. This problem more likely occurs in overlays of small size. During shortcut neighbor list construction, peers can estimate the bandwidth situation of each group, so that a peer belonging to a group with sufficient bandwidth can help distribute the blocks belonging to a group with limited bandwidth.

References

[1] *PPLive*, http://www.pplive.com/.
[2] *UUSee*, http://www.uusee.com/.
[3] *PPStream*, http://www.ppstream.com.
[4] C. Huang, J. Li, and K. W. Ross, “Can Internet Video-on-Demand Be Profitable?” in *Proceedings of ACM SIGCOMM*, August 2007.
[5] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai, “Improving VoD Server Efficiency with BitTorrent,” in *Proceedings of ACM Multimedia*, September 2007.
[6] C. Huang, J. Li, and K. Ross, “Peer-assisted VoD: Making Internet Video Distribution Cheap,” in *Proceedings of the Sixth International Workshop on Peer-to-Peer Systems (IPTPS 2007)*, Feburary 2007.
[7] Y.-F. Chen, Y. Huang, R. Jana, H. Jiang, M. Rabinovich, J. Rahe, B. Wei, and Z. Xiao, “Towards Capacity and Profit Optimization of Video-on-Demand Services in a Peer-assisted IPTV Platform,” *Multimedia Systems*, vol. 15, pp. 19 – 32, February 2009.
[8] Y. Huang, T. Z. J. Fu, D.-M. Chiu, J. C. S. Lui, and C. Huang, “Challenges, Design and Analysis of a Large-Scale P2P-VoD System,” in *Proceedings of ACM SIGCOMM*, August 2008.

[9] X. Yang, M. Gjoka, P. Chhabra, A. Markopoulou, and P. Rodriguez, “Kangaroo: Video Seeking in P2P Systems,” in *Proceedings of The Eighth International Workshop on Peer-to-Peer Systems (IPTPS 2009)*, April 2009.

[10] N. Vratonjic, P. Gupta, N. Knezevic, D. Kostic, and A. Rowstron, “Enabling DVD-like Features in P2P Video-on-Demand Systems,” in *Proceedings of the SIGCOMM Peer-to-Peer Streaming and IP-TV Workshop*, August 2007.

[11] Z. Yin and H. Jin, “DHT Based Collaborative Multimedia Streaming and Caching Service,” in *Proceedings of the IEEE International Region 10 Conference (TENCON 2005)*, November 2005.

[12] W.-P. Yiu, X. Jin, and S.-H. Chan, “VMesh: Distributed Segment Storage for Peer-to-Peer Interactive Video Streaming,” *IEEE Journal on Selected Areas in Communications, Special Issue on Advances in Peer-to-Peer Streaming Systems*, vol. 25, no. 9, pp. 1717–1731, December 2007.

[13] D. Wang and J. Liu, “A Dynamic Skip List-Based Overlay for On-Demand Media Streaming with VCR Interactions,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 4, pp. 503 – 514, April 2008.

[14] B. Cheng, H. Jin, and X. Liao, “Supporting VCR Functions in P2P VoD Services Using Ring-Assisted Overlays,” in *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, June 2007.

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

[16] T. Do, K. A. Hua, and M. Tantaoui, “P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment,” in *Proceedings of the IEEE International Conference on Communications (ICC 2004)*, june 2004.

[17] Y. Guo, K. Suh, J. Kurose, and D. Towsley, “A Peer-to-Peer On-Demand Streaming Service and its Performance Evaluation,” in *Proceedings of The IEEE International Conference on Multimedia and Expo 2003(ICME 2003)*, June 2003.

[18] Y. Cui, B. Li, and K. Nahrstedt, “oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks,” *IEEE Journal on Selected Areas in Communications, Special Issue on Recent Advances in Service Overlay Networks*, vol. 22, no. 1, pp. 99–106, January 2004.

[19] A. Vlavianos, M. Iliofotou, and M. Faloutsos, “BiToS: Enhancing BitTorrent for Supporting Streaming Applications,” in *Proceedings of The Ninth IEEE Global Internet Symposium*, April 2006.

[20] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, “BASS: BitTorrent Assisted Streaming System for Video-on-Demand,” in *Proceedings of the 7th IEEE Workshop on Multimedia Signal Processing (MMSP)*, October 2005.

[21] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. R. Rodriguez, “Is High-Quality VOD Feasible Using P2P Swarming?” in *Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, May 2007.

[22] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, P. Rodriguez, “Exploring VoD in P2P Swarming Systems,” in *Proceedings of the 26th Annual IEEE Conference on Computer Communications Mini Symposiums (INFOCOM 2007 Mini Symposium)*, May 2007.

[23] B. Cheng, X. Liu, Z. Zhang and H. Jin, “A Measurement Study of a Peer-to-Peer Video-on-Demand System,” in *Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS 2007)*, February 2007.

[24] Y. He and Y. Liu, “VOVO: VCR-Oriented Video-on-Demand in Large-Scale Peer-to-Peer Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 528–539, April 2009.

[25] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, “PROMISE: peer-to-peer media streaming using CollectCast,” in *MULTIMEDIA ’03: Proceedings of the eleventh ACM international conference on Multimedia*. ACM Press, 2003, pp. 45–54.

[26] H. Chi, Q. Zhang, J. Jia, and X. Shen, “Efficient Search and Scheduling in P2P-based Media-on-Demand Streaming Service,” *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 119–130, January 2007.

[27] D. Kostic, A. Rodriguez, J. Albrecht, B. Abhijeet, and A. Vahdat, “Using Random Subsets to Build Scalable Network Services,” in *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[28] X. Zhang, J. Liu, B. Li and T.P. Yum, “CoolStreaming/DONet: A Data-Driven Overlay Network for Live Media Streaming,” in *Proceedings of IEEE INFOCOM*, March 2005.

[29] V. K. Goyal, “Multiple Description Coding: Compression Meets the Network,” *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 74–93, September 2001.

[30] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers, “The State of Peer-to-Peer Simulators and Simulations,” *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 95 – 98, April 2007.