InstantLeap: Fast Neighbor Discovery in P2P VoD Streaming

Xuanjia Qiu[1] Chuan Wu[2] Xiaola Lin[1] Francis C.M. Lau[2]

[1]Department of Computer Science
Sun Yat-Sen University
P. R. China
is03qxj@mail2.sysu.edu.cn, linxl@mail.sysu.edu.cn

[2]Department of Computer Science
The University of Hong Kong
Hong Kong
{cwu,fcmlau}@cs.hku.hk

ABSTRACT

A fundamental challenge in peer-to-peer (P2P) Video-on-Demand (VoD) streaming is to quickly locate new supplying peers whenever a VCR command is issued, in order to achieve smooth viewing experiences. For most existing commercial systems which resort to tracking servers for such neighbor discovery, the increasing scale of P2P VoD systems has brought heavy load onto the dedicated servers. 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 the same video, that constitutes a light-weighted indexing structure to support efficient streaming and fast neighbor discovery at the same time. *InstantLeap* achieves an $O(1)$ neighbor discovery efficiency upon any playback "leaps" across the media stream in streaming overlays of any sizes, with a low messaging cost for the overlay maintenance. We support our design with rigorous analysis and extensive simulations.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed Applications*

General Terms

Algorithms, Design

Keywords

Peer-to-peer network, video-on-demand, indexing overlay, neighbor discovery

1. INTRODUCTION

Peer-to-peer (P2P) Video-on-Demand (VoD) streaming has been successfully deployed over the Internet [1, 2, 3], providing thousands of videos to hundreds of thousands of users. The state-of-the-art P2P VoD applications are based on the design philosophy of allowing peers watching the same video to exchange available media blocks among themselves, in order to alleviate the server load [7]. As 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 same video, the users (peers) could be watching different parts of the stream, and may issue VCR commands at will to "jump" to new playback positions, leading to fundamentally lower levels of content overlap among the peers than those in live streaming; this necessitates frequent search for new supplying peers, and such neighbor discovery has to be carried out as fast as possible in order to guarantee smooth playback.

For neighbor discovery upon such "jumps", existing commercial P2P VoD systems have largely resorted to tracking servers which keep track of the block availability at all the peers, and the peers would query the server for available serving peers. Such a tracking server however can easily become a bottleneck when peers join, depart, and issue VCR commands frequently.

To map block locations to peers, DHT (distributed hash table) has been adopted in a number of recent P2P VoD proposals [10, 12, 13]. 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 they progress in the playback.

It is possible using different overlay structures to implement neighbor lookup without the complexity and cost of constructing a DHT. Wang *et al.* [11] utilize a dynamic skip list 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.* [6] suggest the use of an AVL tree for peer indexing, which can achieve a search efficiency sublinear to the number of peers. Recently, Cheng *et al.* [5] propose a ring-assisted overlay management scheme, where each peer maintains a set of concentric rings with different radii 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.

In this paper, we propose *InstantLeap*, a new method of organizing peers watching the same video in a P2P VoD application. The resulting overlay structure is simple but efficient, supporting both effective streaming and *instant* neighbor lookup in the face of any playback "leaps". Peers are grouped according to their playback locality. Each peer strategically connects to a number of peers with similar playback progress, as well as some other selected peers watching different parts of the video. The original highlights of our overlay design are as follows.

NOSSDAV'09, June 3–5, 2009, Williamsburg, Virginia, USA.

- ▷ We show an O(1) neighbor discovery efficiency upon peer joins and playback leaps across the video stream.
- ▷ We show an $O(m)$ overlay maintenance overhead for managing peer dynamics, including failures, departures, and any playback leaps, where m is the number of groups.
- ▷ *InstantLeap* can be implemented based on the mesh-pull protocol employed in prevailing P2P VoD systems, with the simple add-on of 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. In Sec. 2, we present our network model and the design rationale. In Sec. 3, we discuss the detailed protocols in *InstantLeap* and analyze the performance. We evaluate *InstantLeap* by extensive simulations and comparisons against existing schemes in Sec. 4, and conclude the paper in Sec. 5.

2. DESIGN RATIONALE

A typical mesh-based P2P VoD streaming application consists of multiple mesh overlays, each of which connects the peers watching the same video. The peers in the same overlay exchange information about available video blocks in their local buffers. The buffer at each peer represents a sliding window of the video stream, containing the block it is currently playing (referred to as its *playing position* hereinafter) and a number of blocks the peer has just watched or retrieved to be played in the near future.

Unlike live streaming where a peer's playing position can only move continuously forward, VoD streaming allows the users to freely change their playing positions to any random point in the video stream (referred to as playback *leaps* hereinafter). Such random playback leaps give rise to the need to quickly discover new supplying peers, which can provide video blocks at the new playing position. To achieve the fastest possible neighbor discovery, our design of a P2P VoD overlay, corresponding to the streaming of one video to N peers, has the following features.

2.1 Peer grouping with playback locality

We partition the video stream into m consecutive segments along the time axis. The size of each segment approximates the size of the local buffer at each peer. A peer is marked as a member of group i if its current playing 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 and in the two adjacent groups $i-1$ and $i+1$ for downloading and exchanging of video blocks; the second list (referred to as the *shortcut neighbor list*) includes peers that are not in group i or 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. 3.

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

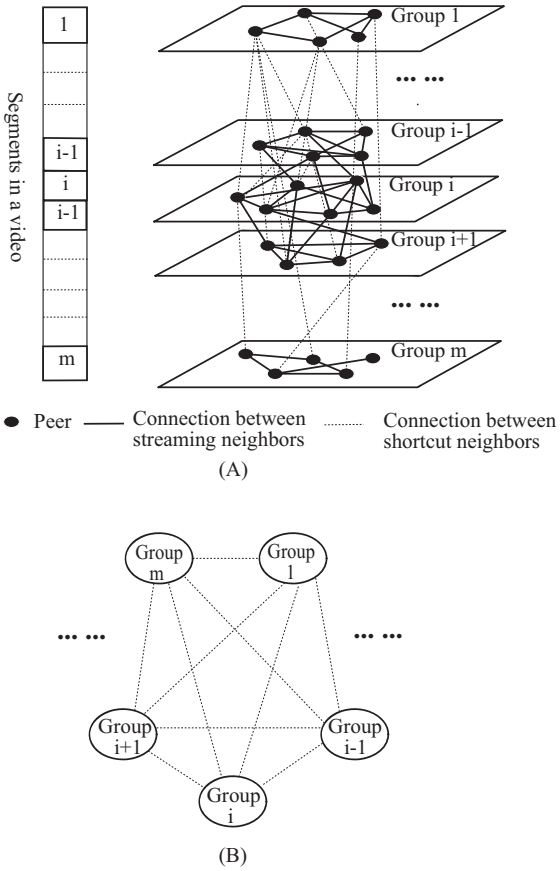

Figure 1: P2P VoD overlay design in InstantLeap.

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 is independent of m.

2.2 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, *i.e.*, $O(1)$ with high probability.

The following section presents the detailed protocols for realizing the above design. We also give a rigorous analysis showing 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 playing position can be achieved with high probability.

3. INSTANTLEAP: PROTOCOLS AND ANALYSIS

InstantLeap assumes an underlying framework similar to that of most of the state-of-the-art mesh-based P2P streaming protocols (*e.g.,* CoolStreaming [14], UUSee [2], PPLive [8]): new peers are introduced into a streaming overlay by a bootstrapping tracking server; they then stream the video by retrieving needed blocks from neighbors based on exchanges of buffer maps, and may exchange neighbor lists among each other to learn more peers in the overlay.

The original highlight of *InstantLeap* is the construction of shortcut neighbor lists based on simple exchanges of known neighbors among the peers with some strategies. These lists facilitate fast neighbor discovery upon playback leaps with low additional protocol overhead.

3.1 Protocols

Using random neighbor list exchanges between peers as a basic function, the main procedures in *InstantLeap* include the following three: *lookup* of a peer in the destination group of a playback leap, *construction of the streaming neighbor list* at a peer, and *construction of the shortcut neighbor list* at a peer. The highlights of these three procedures are given in **Algorithm 1**. Main protocols in *InstantLeap* can be implemented using these procedures, as follows.

Initial neighbor lists construction upon peer joins

With *InstantLeap*, we seek to minimize the load on the tracking 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. If none of the assigned neighbors is in the group where the peer's desired playing position falls into, the new peer executes the procedure *lookup* to discover a peer in the destination group by exchanging neighbor lists with the few known neighbors. Then the new peer executes *ConstructStreamNeighborList* to obtain more neighbors with similar playback progress for video streaming, and *ConstructShortcutNeighborList* to establish shortcuts to segments across the entire video stream.

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, 2, 3]. The size of the shortcut neighbor list 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 expanding the shortcut neighbor list of a peer when the peer has obtained $\beta_2 m$ neighbors in $\beta_2 m$ different groups; the peer will restart this procedure when its number of shortcut neighbors has fallen under $\beta_1 m$.

The case of continuous playback

When a peer watches the video continuously, its group membership changes when it moves on to play the next video segment. The peer updates 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 as well.

The case of playback leaps

When there is a playback leap due to a VCR operation, the *lookup* procedure is executed, similar to the case when a peer first joins the overlay. When the peer still fails to discover a peer in the destination group after T exchanges of neighbor lists, it queries the tracking 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 tracking server. The value of T is to be decided in practice, considering the tradeoff between the maximum allowed delay for neighbor discovery and the aggregate load on a tracking server.

After connecting to a peer in the destination group, the peer executes *ConstructStreamNeighborList* to obtain more new streaming neighbors, and starts streaming by retrieving video blocks from these neighbors. Meanwhile, it notifies all its existing neighbors in the two lists of the changes of its group membership.

Algorithm 1 Main Procedures at each peer in *InstantLeap*

procedure GETNEIGHBORLIST($listType$,$peerId$)
  if $listType$ is shortcut neighbor list
    obtain the shortcut neighbor list from $peerId$
    merge obtained peers into my shortcut neighbor list
  end if
  if $listType$ is streaming neighbor list
    obtain the streaming neighbor list from $peerId$
    merge obtained peers into my streaming neighbor list
  endif
end procedure

procedure CONSTRUCTSTREAMNEIGHBORLIST
  while my streaming neighbor list is not full
    select a random peer with $peerId$ from my streaming neighbor list
    GetNeighborList($streamNeighborList$, $peerId$)
  end while
end procedure

procedure CONSTRUCTSHORTCUTNEIGHBORLIST
  while the size of my shortcut neighbor list is smaller than $\beta_2 m$
    select a random peer with $peerId$ from my streaming or shortcut neighbor list
    GetNeighborList($shortcutNeighborList$, $peerId$)
  end while
end procedure

procedure LOOKUP($PlayPosition$)
  calculate $groupId$ of the destination group corresponding to $PlayPosition$
  while there is no peer in my streaming or shortcut neighbor list with $groupId$ and the times of neighbor list exchanges has not exceeded T
    select a random peer with $peerId$ in my streaming or shortcut neighbor list
    GetNeighborList($shortcutNeighborList$, $peerId$)
  end while
  if no neighbor with $groupId$ has been discovered
    request a neighbor with $groupId$ from tracking server
  end if
  if I am not evoking the procedure as a new join peer
    update all neighbors in my streaming and shortcut neighbor lists with my new group membership
    clear my streaming neighbor list
  end if
  add the discovered neighbor into streaming neighbor list
end procedure

Neighbor list maintenance upon neighbor dynamics

When a peer detects the failure or departure of a neighbor, it may simply remove the neighbor from its corresponding neighbor list. When a peer receives update of group membership from a streaming neighbor which can no longer serve itself, it will transfer the neighbor to its shortcut neighbor list, if it previously has no shortcut neighbor for the updated group. When group membership update from a shortcut neighbor arrives, the peer will keep the neighbor and update its group membership, if there has been no shortcut neighbor for the group.

Refinement

Based on the random neighbor list exchanges, a peer may get to know multiple peers in a group after several neighbor list exchanges. In *InstantLeap*, only one shortcut neighbor for each group is to be maintained in the shortcut neighbor list at a peer, in order to minimize the buffering overhead. Since a shortcut neighbor may well become a streaming neighbor in the near future if the peer's playing position changes to the respective segment, high-quality peers are favored and maintained as shortcut neighbors, *e.g.*, those with larger bandwidth, better stability, or more available blocks in the buffer.

3.2 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 complexity can be achieved upon any playback leaps with low protocol overhead.

We first show that a peer can obtain shortcut neighbors across $O(m)$ groups by only a small number of neighbor list exchanges with other peers. Let s_i^t denote the average size of the shortcut neighbor list at peer i after t times of random neighbor list exchanges. We first prove a lemma.

LEMMA 1. *Let peer i and peer j be two randomly selected peers from all N peers in the streaming overlay, with an initial size of the shortcut neighbor list of $s_i^0 = a$ and $s_j^0 = b$, respectively. The average size of their shortcut neighbor lists after one exchange of the shortcut neighbor list between peer i and j is $s_i^1 = s_j^1 = a + b - \frac{a \times b}{m}$.*

PROOF. The average size of the merged shortcut neighbor list is the sum of the number of groups peer i's shortcut neighbors belong to (*i.e.*, a) and the number of groups peer j's shortcut neighbors belong to (*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 i in the streaming overlay. After t times of shortcut neighbor list exchanges with randomly selected other peers in the overlay, the average size of the shortcut neighbor list at peer i is $s_i^t \geq m[1 - e^{-\frac{2^t \times K}{m}}]$.*

PROOF. Given each peer has a similar number of initial shortcut neighbors, based on Lemma 1, the average number of shortcut neighbors at peer i after one exchange with another random peer is $s_i^1 = 2 * s_i^0 - \frac{(s_i^0)^2}{m}$.

After $t+1$ times of exchanges between peer i and other peers, we have $s_i^{t+1} = 2 * s_i^t - \frac{(s_i^t)^2}{m}$, for $t \geq 0$.

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

$$\begin{aligned} Q(t+1) &= s_i^{t+1} - m = 2 * s_i^t - \frac{(s_i^t)^2}{m} - m \\ &= -\frac{1}{m}[s_i^t - m]^2 = -\frac{1}{m}[Q(t)]^2 \end{aligned}$$

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

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 size of the shortcut neighbor list at 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 case 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 of size no less than $(1 - \frac{1}{e})m$ after a few exchanges after joining the overlay. As a side note, in our implementation of the protocol handling peer joins, as discussed in Sec. 3.1, parameter β_2 is set to a value of $\frac{2}{3}$ by considering this effect. 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 Sec. 3.1, 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 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 to 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 i in the overlay who makes a playback leap to destination group d. Let p 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 peer i's shortcut neighbor list does not contain such a neighbor, the probability that peer i can obtain such a peer by one neighbor list exchange with another peer j, randomly selected from the overlay, is p, *i.e.*, the probability that peer j has a neighbor in group d. Therefore, the probability that a peer successfully obtains a neighbor in the destination group after v times of random neighbor list exchanges is $(1-p)^v \times p$, and the expected number of exchanges is $\sum_{v=0}^{\infty} v \times (1-p)^v \times p = \frac{1-p}{p}$.

Considering that the peers involved in the exchanges are not new joiners (*i.e.*, they have all finished their joining procedures), we have $p \geq 1 - \frac{1}{e}$, and the expected number of exchanges is $\frac{1-p}{p} \leq \frac{1}{e-1} \approx 0.58$. 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 exchanges in *InstantLeap* between a peer and one of its randomly selected neighbors represent the best possible approximation to the expected randomness in a practical P2P VoD system.

The overhead in *InstantLeap* protocols is due mainly to the exchange, construction and maintenance of neighbor lists in case of various peer operations, including continuous playback, playback leaps, joins and departures. Since we maintain at each peer a streaming neighbor list of at most a constant size and a shortcut neighbor list with size $O(m)$, the overhead involved in each operation is at most $O(m)$. We will show that the protocol overhead is indeed negligible as compared to the streaming rate in our simulations.

4. 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 timeouts. 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 [9] which may support 3000 peers at most.

We have implemented *InstantLeap* protocols in our simulator, and also implemented the dynamic skip list (DSL) algorithm [11] for comparison purpose. In our evaluations, the streaming rate of the video distributed in the overlay is 450 Kbps. The upload bandwidth at the peers ranges between $300 - 600$ Kbps. 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 we have carried out using different overlay sizes, in order to keep the total number of online peers at a similar level overtime 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. The peer buffer has a length of 1 minute. The number of groups (m) thus ranges from 40 to 200, accordingly. These parameters are carefully selected to be consistent with the measurement results in the existing representative P2P VoD systems [4, 8]. In addition, unless stated otherwise, the parameters in our protocols are $\beta_1 = \frac{1}{3}$, $\beta_2 = \frac{2}{3}$, and $T = 10$.

4.1 Performance of neighbor discovery

Figure 2 and 3 show the average numbers of neighbor list exchanges upon peer joins and playback leaps, respectively, in overlays of different sizes and with videos of different lengths. These numbers would translate into the delay for neighbor discovery in case of peer joins and playback leaps, when the protocols are implemented in practice. We observe that the numbers of exchanges in both cases are all less than 6, and there is almost no change with the increase of the overlay size. This clearly confirms that *InstantLeap* achieves a constant neighbor discovery performance, independent of the number of peers in the overlay. Although this number is larger than what is derived in Theorem 4 (where we assume ideal random exchanges, which may not be feasible in practical systems), it would already be quite satisfactory in practice.

In the case of DSL (dynamic skip list) algorithm, the numbers in Figure 2 and 3 represent the average numbers of messages needed to discover a supplying peer. We can see that DSL in general requires many more messages (hence longer delay) for neighbor discovery, and their number increases with the increase of overlay size as well.

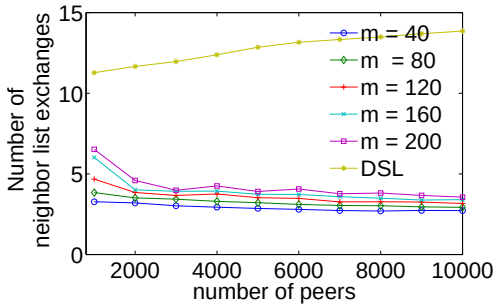

Figure 2: Average number of neighbor list exchanges upon peer joins.

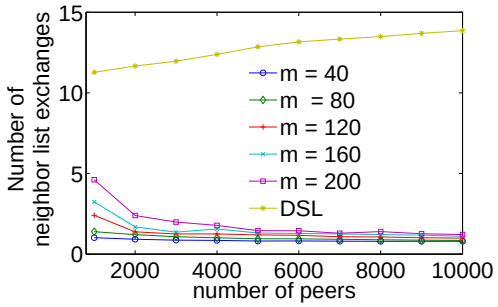

Figure 3: Average number of neighbor list exchanges upon playback leaps.

4.2 Overhead for overlay maintenance

Figure 4 and 5 plot the control messaging overhead for overlay maintenance upon peer leaps and departures, which is derived as a function of the number of control messages, typical control message sizes, and typical delays among Internet hosts. We observe a control messaging bandwidth that is no more than 2% of the video streaming rate of 450 Kbps, which is also largely independent of the overlay size. Therefore, we can conclude that by sacrificing just a little of the bandwidth for the fast neighbor discovery upon VCR operations, the users' experience can be improved significantly.

4.3 Impact of the number of shortcut neighbors

In *InstantLeap*, we impose a range on the number of shortcut neighbors at each peer, *i.e.*, $[\beta_1 m, \beta_2 m]$. We now investigate whether the number of shortcut neighbors at peers affects the performance and overhead in *InstantLeap*. Figure 6 and 7 plot 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 performance becomes better when peers have more shortcut neighbors (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 $\beta_2 = \frac{2}{3}$ in all of our previous evaluations.

5. CONCLUDING REMARKS

This paper proposes *InstantLeap*, a scalable light-weighted indexing structure to achieve efficient streaming and fast neighbor

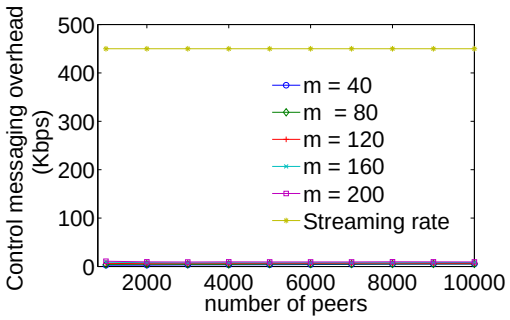

Figure 4: Maintenance overhead upon playback leaps.

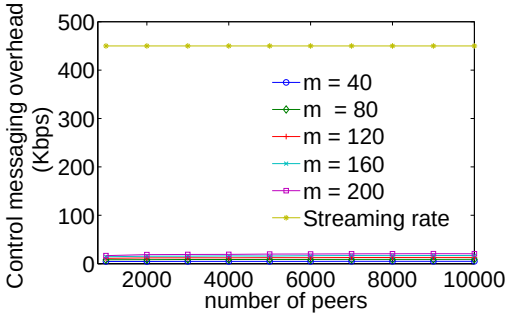

Figure 5: Maintenance overhead upon peer departures.

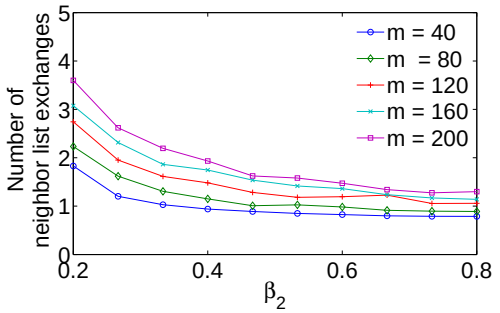

Figure 6: Average number of neighbor list exchanges upon playback leaps.

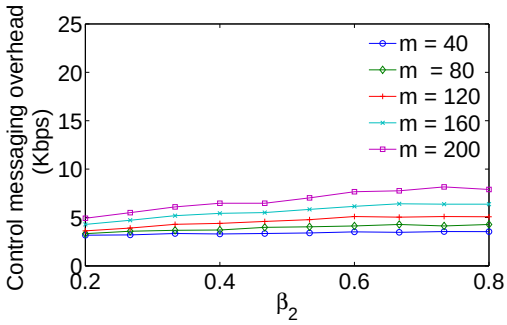

Figure 7: Maintenance overhead upon playback leaps.

discovery for P2P VoD streaming applications. The distinctive feature of *InstantLeap* is its neighbor discovery method, which has a performance of $O(1)$ upon any playback leap with low overhead. *InstantLeap* can be implemented on top of the basic framework of prevailing mesh-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, which is validated by both theoretical analysis and simulations with large-scale overlays and intense peer dynamics. As ongoing work, we are extending our design to address neighbor discovery across multiple concurrent streaming overlays, as in practical P2P VoD applications.

6. REFERENCES

[1] *PPLive*, http://www.pplive.com/.

[2] *UUSee*, http://www.uusee.com/.

[3] *PPStream*, http://www.ppstream.com/.

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

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

[6] 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*, 25(1):119–130, January 2007.

[7] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai. Improving VoD Server Efficiency with BitTorrent. In *Proc. of ACM Multimedia 2007*, September 2007.

[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 *Proc. of ACM SIGCOMM*, August 2008.

[9] 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*, 37(2):95 – 98, April 2007.

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

[11] 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*, 19(4):503 – 514, April 2008.

[12] Z. Yin and H. Jin. DHT Based Collaborative Multimedia Streaming and Caching Service. In *Proc. of the IEEE International Region 10 Conference (TENCON 2005)*, November 2005.

[13] 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*, 25(9):1717 – 1731, December 2007.

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