# RINDY: A Ring Based Overlay Network for Peer-to-Peer On-Demand Streaming*

Bin Cheng, Hai Jin, and Xiaofei Liao

Cluster and Grid Computing Lab
Huazhong University of Science and Technology, Wuhan, 430074, China
{showersky, hjin, xfliao}@hust.edu.cn

**Abstract.** Using peer-to-peer overlay network to provide video-on-demand service has been a promising solution due to its potential high scalability and low deployment cost. However, it remains a great challenge to construct an efficient overlay network for peer-to-peer video-on-demand systems owing to their inherent dynamicity caused by frequent VCR operations or joining/leaving operations. In this paper, we propose a *ring based overlay network* to handle this problem, called *RINDY*, in which each peer maintains a *gossip-ring* to explore appropriate data suppliers and several *skip-rings* with power law radius to assist the quick relocation of VCR operations. Our simulation results show that RINDY achieves better load balance in the control overhead than tree based overlay. As compared with the traditional client/server model, it saves more server bandwidth and achieves lower start-up latency when lots of users watch a same video simultaneously.

## 1 Introduction

With the exponential expansion of Internet resource and users, Video-on-Demand has become one of the most attractive services over Internet [15]. In order to provide large scale on-demand streaming services, most of existing systems deploy content distribution networks (***CDN***s) [5][16] and distributed proxies [6][11] to extent their capacities. Unfortunately, the bandwidth and I/O capacity of servers inevitably become their performance bottleneck as the number of online user increases constantly.

Peer-to-peer multimedia streaming systems [1][2] have become popular both in academic and industry. Compared with the traditional client/server model, they can sufficiently utilize the capacity of end nodes to improve their scalability. There have been a number of successful research projects on peer-to-peer live streaming, such as CoolStreaming [21], ESM [4], SplitStream [3], AnySee [18]. But there are few peer-to-peer on-demand systems due to the following difficulties. *First*, for peer-to-peer on-demand streaming systems, peer nodes have more obvious ***dynamicity***. Beyond joining or leaving, users can perform all kinds of VCR operations at will, such as PAUSE, REWIND and FORWARD, which lead to frequent changing of topology. *Second*, each peer has ***heterogeneous*** capacities, such as different inbound/outbound

* This paper is supported by National Science Foundation of China under grant 60433040, and CNGI projects under grant CNGI-04-12-2A and CNGI-04-12-1D.

J. Ma et al. (Eds.): UIC 2006, LNCS 4159, pp. 1048 – 1058, 2006.

bandwidth, memory size. In this case, it still is a problem to determine where and when to fetch an expected data segment from multiple suppliers with non-uniform bandwidth and data availability, under the limitation of the playback deadline. *Third*, it is also very difficult to achieve high ***reliability*** and guarantee the quality of streaming under the condition that peer nodes join, leave or seek frequently.

In order to solve these problems, we propose a novel ***ring based overlay*** network for peer-to-peer on-demand streaming service, called ***RINDY***, in which each peer keeps a set of concentric rings to implement efficient membership management and fast relocation of VCR operations under a low control overhead. It maintains a *gossip-ring* to explore appropriate data suppliers and several *skip-rings* with power law radius to assist quick relocation of VCR operations. Compared with tree-based overlay, it is more reliable because it only needs to maintain a loosely consistent ring overlay and can tolerate the failure of many peer nodes.

The rest of this paper is organized as follows. An overview of RINDY system architecture is presented in section 2. In section 3, we give an insight of ring based overlay, including membership management and overlay maintenance. The performance of RINDY is evaluated by simulation in section 4. Related work is discussed in section 5. Finally, we end this paper with some conclusions and future work.

## 2 Overview of RINDY

As shown in Figure 1, the infrastructure of RINDY is consisted of four components: *tracker server, source server, peer,* and *web portal*. For the tracker server, it is a well-known Rendezvous Point (RP) to help each newly joining peer bootstrap. We deploy a backup tracker server. There are many source servers distributed in different ISP networks and organized in a logical ring. When an incoming peer logins our overlay network, the tracker server will allocate a near source server for it. Peer is the most

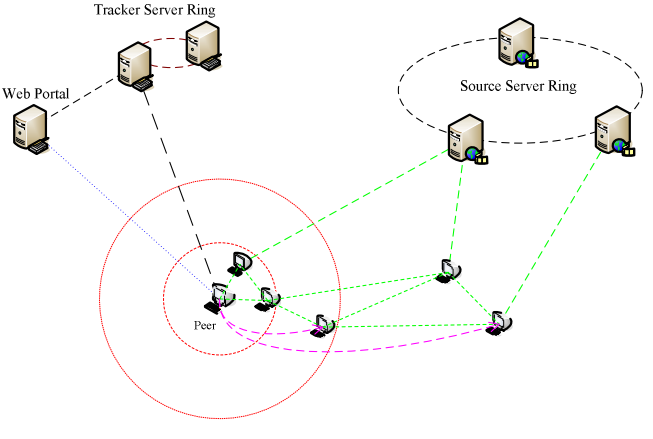


**Fig. 1.** Architecture of RINDY System, including Tracker Server, Source Server, Peer and Web Portal

complicated component in our peer-to-peer on-demand streaming system. For any peer, it can not only request data packets from its source server, but also exchange data packets with other peers. Each peer caches the recent watched media data and announces its buffer map to all neighbors periodically. At the same time, it schedules data requests to fetch its required media data packets to support playback continuously. The web portal is the entry of the system, providing the latest channel list to show the number of online peers and the rank of all movies.

## 3 Ring-Based Overlay

In this section, we focus on the discussion on the construction and maintenance of ring based overlay, including neighbor distribution, ring based gossip protocol, and relocation algorithm for new neighbors.

### 3.1 Overlay Construction

***A. Ring***

In RINDY, each peer node keeps track of a small and partial view of the total overlay and organizes all neighbors into concentric, non-overlapping logical rings according to their relative distance. The radius of each ring is represented with $a2^i$, where $a$ represents the length of buffer window ($a = w$) and $i$ is the layer number ($0< i <= B$, $B$ is the maximum layer number). The $i$th ring denotes the zone between the inner radius ($r_{i-1} = a2^{i-1}$) and the outer radius ($r_i = a2^i$), where $i >= 1$. The distance between any two peers can be calculated with their playing positions, i.e., given any two peers $i$ and $j$ and their playing positions $cur_i$ and $cur_j$, respectively, the distance from peer $j$ to peer $i$ ($d_j$) equals $cur_j - cur_i$. If $d_j$ is negative, the playing position of peer $j$ is behind that of peer $i$ and we call peer $j$ a *back-neighbor* of peer $i$. Otherwise if $d_j$ is positive, peer $j$ is a *front-neighbor* of peer $i$. The distance remains unchanged until some VCR operations occur. For all neighbors of peer $i$, their locations are up to the distances from them to peer $i$. If the distance of neighbor $j$ meets the inequation $a2^{k-1} < |d_j| <= a2^k$, peer $i$ will place neighbor $j$ in the $\boldsymbol{k}$th ring.

***B. Neighbor Distribution over Ring***

For any peer, a large $q$ will help it get larger member list, increasing its knowledge about the total overlay network and the locality of lookup operations. But at the same time, a large $q$ also entails more memory and more bandwidth and causes worse network traffic. A reasonable neighbor distribution always brings more efficient performance for VCR relocation and member discovery. For peer-to-peer video-on-demand systems, nodes that are temporal diversity instead of clustered together can forward a query to a wider region.

In RINDY, we define two kinds of rings, gossip-ring and skip-ring, and provide different neighbors distribution rules for them respectively. For the innermost ring ($i = 1$), it collects some peer nodes with close playing positions and overlapped buffer windows with peer $i$. We call this ring *gossip-ring*. For all outer rings ($i>= 2$), they sample some remote peer nodes to help peer $i$ look up new neighbors after seeking. These rings are mainly used to improve the locality of lookup operations and decrease the

load of tracker server. We call these rings *skip-rings*. In the gossip-ring, each peer node keeps *m* ring members by the received gossip message, called near-neighbors. In each skip-ring, a peer node keeps track of the *k* primary ring members and *l* secondary ring members which serve as *backup candidates* for primary ring members. All of these primary remote members are called far-neighbors. When any far-neighbor leaves, a new far-neighbor will be selected from *l* back candidates quickly.

Figure 2 illustrates the member distribution of peer *i*. Node A is a near-neighbor in the gossip-ring and node B, D, E, G and H are far-neighbors in the skip-rings. Node C is the backup candidate of far-neighbor B. Peer *i* can propagate gossip messages through its near-neighbors and explore new members by receiving gossip messages. For any peer, to find good neighbors as its partners is critical to continuous playback. The join and departure messages need to announce to other peers in the gossip-ring. So we try to maintain as many near-neighbors as possible at the permission of overhead. According to the analysis in [9][10][19], log(*n*) is enough to make most of near peers receive join and departure events. The ring based overlay only requires loose consistent. For skip-rings, they function as the bridge to look up new proper neighbors and good partners. It is enough to ensure that there is one connection in each skip-ring. Considering users may seek back or forward, we remain two primary far-neighbors pointing to the front and back of current position respectively. Here we configure $k = l = 2$.

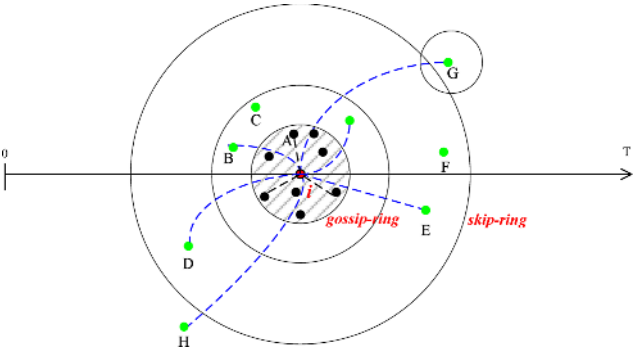


**Fig. 2.** Neighbor distribution of peer node i, node A in the gossip-ring and node B, C, D, E, F, G, H in the skip-ring

### 3.2 Overlay Maintenance

***A. Discovery of New Members***

In RINDY, each peer can leave the overlay network or seek new positions so that some peers may lose lots of neighbors due to the departure of neighbors, which results in the damage of the normal ring structure. In previous work, there are many projects importing a gossip protocol to discover new members, such as CoolStreaming [21], AnySee [18]. The principle of traditional gossip protocol is simple, that is, a node sends a newly generated message to a set of randomly selected nodes and these nodes do similarly in the next round. Thus contents will spread throughout the network in an epidemic fashion. The random choice of gossip targets achieves resilience to random

failures and enables decentralized operations. However, a great trouble for gossiping based streaming is how to minimize delay and avoid the date outage of messages. It is difficult for peer-to-peer on-demand streaming to handle this problem because the playing position of each peer is moving constantly and even changing unexpectedly. In order to address this problem, we design a *temporal and spatial restraint* gossip protocol based on ring structure to explore new members and maintain stable distribution of neighbors in all rings.

In RINDY, each peer sends a gossip message to announce its existence and its buffer windows status periodically. At the same time, it updates its member table according to the member information carried in the received gossip messages. The gossip message format is shown in Figure 3, where *GUID*, *Peer Address*, *near-neighbor* and *Cur* represent the global identifier, the network address, neighbor number and the playback offset of source peer, respectively. *TTL* is the remaining hop number of this message. *Max, Min, Want* and *Avail* denote the snapshot of moving buffer together. All gossip messages are divided into two kinds, ANNOUNCE and FORWARD. The difference between them is that ANNOUNCE messages carry the buffer snapshot while FORWARD messages not. Because the main purpose of gossip is to help online peers to discover more data suppliers with close playback offset, we restrict all gossip messages spreading in the range of gossip-ring to decrease the overhead of gossip propagation. For a peer, it sends a ANNOUNCE message to all of its near-neighbors in each gossip round, notifying its position and buffer map. After its near-neighbors receiving this message, they update their mCache and randomly choose one near-neighbor with the playback offset $t_{\text{forward}}$ from their near-neighbors, where $t_{\text{forward}}$ meets the relation $t_{\text{forward}} - m <= t_{\text{source}} <= t_{\text{forward}} + m$ and $t_{\text{source}}$ is the current playing position of source peer of this gossip message.

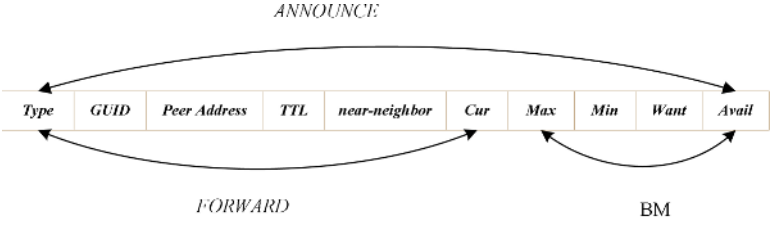


**Fig. 3.** Format of gossip message

### *B. Lookup of New Position*

The lookup operations based on skip-rings mainly occur in the following two cases. First, when users perform VCR operations lookup operations will be triggered to find some new neighbors close to the destination. Second, when a peer finds that there are not enough neighbors in a ring with the simultaneous leave of all primary far-neighbors, it also executes a lookup procedure to find more backup candidates for this ring. As Figure 4 illustrates, when peer $i$ with the playing position $p$ wants to seek to the destination $d$ it first judges whether the destination $d$ is located on its gossip-ring or not. If $|d - p|$ is less than or equal to $a$, that means the destination $d$ is in its gossip-ring, it will execute a local search to find out enough near-neighbors and far-neighbors for peer $i$. If

$|d - p|$ is more than $a$, it will find out a neighbor as its next hop from its rings closest to the destination $d$ and then forward this query to the next hop neighbor. When the next hop neighbor receives this query it will execute the same procedure and the procedure will be iterated until this request is forwarded to a peer that is in the same gossip-ring with the new peer $i$. As Figure 4 shows, if $d$ is in the 3rd ring of peer $i$ and $P_1$ is the neighbor closest to $d$, the query $q$ will forward to the neighbor $P_1$ at first. Then $P_1$ will forward this query to its neighbor $P_2$ in its 2nd ring closest to $d$. Finally, when $P_2$ finds out that the destination $d$ is in its gossip ring it will add all neighbors in its gossip ring into the result set and return it to the source peer $i$, following the forwarding path.

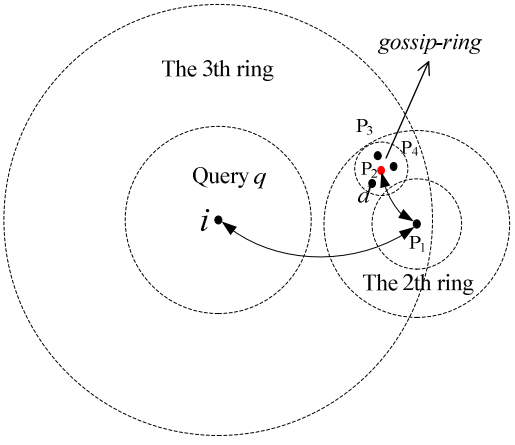


**Fig. 4.** Procedure of lookup operation for peer $i$

## 4 Performance Evaluation

In this section, we investigate the performance of RINDY in server bandwidth, control overhead and start-up latency by simulation. At the same time, we also give some comparisons between RINDY and other on-demand systems.

### 4.1 Simulation Setup

We use GT-ITM topology generator [20] to create a topology of 1000 peer nodes based on transit-stub model. The network consists of 3 transit domains, each with 5 transit nodes and a transit node is connected to 6 stub domains, each with 12 stub nodes. In this set of experiment, peers can be located on any stub nodes in the topology. We randomly choose 1000 stub nodes as peer clients and place the source server on a transit node. The bandwidth between two transit nodes, a transit node and a stub node, two stub nodes are 100Mbps, 10Mbps, and 4Mbps, respectively. In addition, we choose a movie with 60 KB/s streaming rate and 60 minutes content as our testing stream. We modify the source code of NICE simulator (http://www.cs.umd.edu/~ suman/research/myns) to implement a simulator of RINDY, in which each peer remains a ring structure instead of a layer structure. Some important parameters for our ring based overlay network are presented as Table 1.

**Table 1.** Some configuration parameters

| Parameters | Value and Meaning |
|---|---|
| $w$ | *300 seconds, buffer window size* |
| $B$ | *5, maximal layer number* |
| $R$ | *60KB/s, average streaming rate* |
| $\alpha$ | *300, radius coefficient of rings* |
| $t$ | *30 seconds, gossip period* |
| $m$ | *12, near-neighbor number in each gossip-ring* |
| $k$ | *2, far-neighbor number in each skip-ring* |
| $l$ | *3, back candidates in each skip-ring* |
| $L$ | *60 Minutes, length of a sample movie* |

### 4.2 Simulation Result

***A. Control Overhead***

We first evaluate the control overhead for overlay construction and maintenance in our ring based overlay network. Here the control overhead is represented by the number of processed messages. In order to investigate the overhead of ring based gossip, we record the average message number of each peer in a gossip period when there are no any peer failed and any VCR operation occurred, then we calculate the average message number of all peers with different scales. The result is illustrated in Figure 5. We can see that the average messages number increases very slowly after the total peer number of the same channel reaches 400. When the total peer number exceeds 1000, the average message number basically remains about 100, in a restricted constant range. Since the gossip period $t$ is 30 seconds, each peer just processes about three or four messages in a second. Compared to the streaming overhead, the control overhead is very low and can be ignored.

Figure 6 shows the average message number of each peer when 10 percent peers join, leave RINDY network or perform VCR operations randomly. When the total peer number reach a specified scale, about 300 nodes, the average control overhead to accommodate the overlay changing effected by peer joining, leaving or seeking basically keeps a constant. That illustrates that the control overhead induced by peer joining, leaving or seeking does not increase with the scale of peer nodes.

Figure 7 shows the distribution of lookup request message processed by each peers in ring based overlay and tree based overlay when the total peer number is 600 and 10 percent peers perform VCR operations randomly. We can see that, in the ring based overlay, most peer nodes process 0~8 messages and a few peer nodes reach 12 messages but none of them exceeds 20 messages. However, in the tree based overlay, most peer nodes process 0~3 messages but the lookup message number of some peer nodes reaches 30 or 40 and the root node processes 60 messages. Compared with the tree based overlay, our ring based overlay gets better load balance among peers during the relocation of VCR operations. To the best of our knowledge, there are two main reasons for this result. First, in our ring based overlay network, each peer has an individual neighbor distribution and its lookup operations begin with its local rings. However, for the tree based overlay network, all of

lookup operations travel from the root node firstly. The root node and the peer node of upper layer always process more lookup request messages, which results the unbalance of control overhead among peer nodes. Second, for each peer, its skip rings construct a quick index based relative playback offset, just like a binary index, which decreases the length of lookup path.

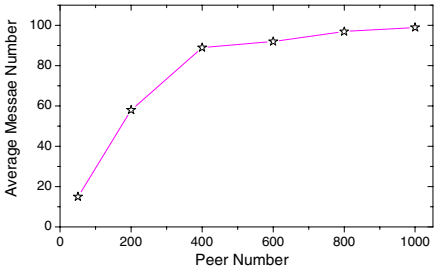


**Fig. 5.** Message cost of stable status

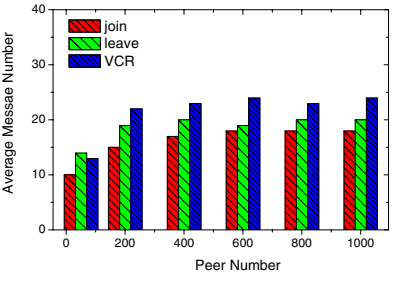


**Fig. 6.** Message cost of joining/leaving and VCR

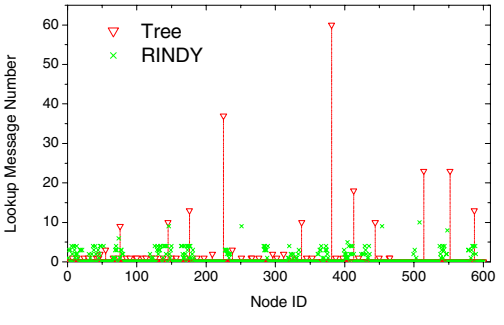


**Fig. 7.** Lookup message distribution when 10% peers randomly seek

### *B. Server Bandwidth Consumption*

Since the main purpose to deploy peer-to-peer on-demand streaming is to alleviate the bandwidth bottleneck, we compare the server bandwidth consumption between traditional C/S model and our RINDY. Figure 8 reports their bandwidth cost with the increasing of peer client number. When the peer client number arrives at 200, the traditional solution nearly consumes all bandwidth of the central server, about 89Mbps. But for our RINDY, the increase of peer client number has no obvious effect to the server bandwidth consumption. The main reason is that peers can exchange their data packets efficiently. In our testing case, the server bandwidth has been the bottleneck when the peer client number reaches about 200, while RINDY just uses 6~8Mbps, 10 percent of C/S model. RINDY has great potential for bandwidth saving.

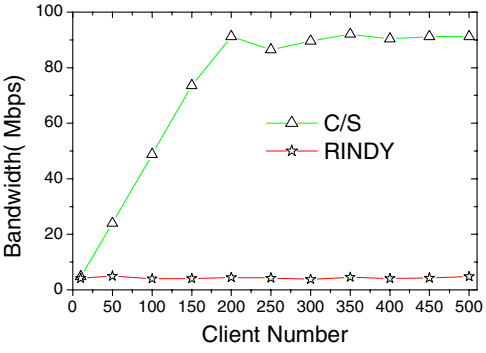


**Fig. 8.** Comparison of server bandwidth between C/S model and RINDY

### *C. Start-up Latency*

The start-up latency is the time that users must wait before playback since users start a channel. Users always expect shorter waiting time, however, it is not easy for most peer-to-peer systems to find initial corresponding data supplier quickly, which always take longer time than expected. In addition, it must try to avoid fetch data from the central server. Otherwise, the central server becomes the bottleneck easily. In RINDY, we use a scheme to solve this problem. We make each peer buffer the initial one minute stream packets and the SDP packet need by RTSP protocol. For a newly incoming peer, it retrieves the initial data from any neighbors. Figure 9 gives a comparison of start-up latency between C/S model and RINDY. The start-up latency of RINDY has little change with the increasing of peer client number, keeping about 30 seconds. But the start-up latency of C/S model increases quickly with the system scale. When the server is overloaded, the start-up latency will make users unacceptable.

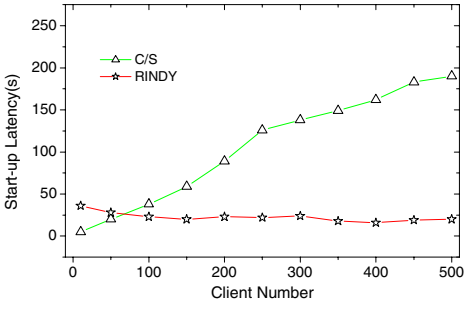


**Fig. 9.** Comparison of start-up latency between C/S model and RINDY

## 5 Related Work

There are mainly following three methods to provide peer-to-peer on-demand streaming systems. The first method is to construct *tree based overlay* network, such as P2Cast [12], P2VoD [8], oStream [7], DirectStream [13]. In these systems, peers on

the upper layer always play an important role in the whole overlay network. Their departure will lead to the lower layer network fluctuation. Moreover, each peer has only one data supplier, which will cause inefficient utilization of available bandwidth in a heterogeneous and highly dynamic network environment.

Another method is to deploy *mesh based overlay* to organize all joined peers, such as PROMISE [14]. Although they can fetch packets from many data suppliers [17], they meet another problem, that is, how to relocate the destination efficiently to support random VCR operations. Because of the shortage of structure, mesh based overlay always consume longer time to find appropriate close partners, therefore they hardly meet the real-time requirements of VoD services and always induce an unacceptable waiting time after seeking. Combing the advantages of tree based overlay and mesh based overlay, TAG [22] presents a *hybrid overlay*, called Tree Assistant Gossip. It uses a balance binary tree to index all joined peer nodes and search many partners from the tree when a peer joins. It integrates the advantages of tree overlay and mesh overlay to avoid the problems above. However, the maintenance algorithm of the distributed AVL tree is complicated. In addition, there is only one tree structure in the global overlay and each lookup operation begins from the root node, so the control overhead among peer clients is unbalanced seriously.

## 6 Conclusions and Future Work

In this paper, we present a novel ring based overlay network for peer-to-peer on-demand streaming, called ***RINDY***, which maintains a gossip-ring to explore appropriate data suppliers and several skip-rings with power law radius to assist quick relocation of VCR operations. We investigate the control overhead of RINDY by simulation experiments. The results illustrate that RINDY achieves better load balance and faster relocation of VCR operations compared with tree based overlay. We also compare the bandwidth consumption and start-up latency between the traditional C/S model and RINDY. The results are so exciting that we believe that RINDY has great potential in bandwidth saving and QoS guarantee. In future we plan to study a distributed cache scheme to improve the performance of peer-to-peer on-demand streaming further.

## References

1. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris, Resilient overlay network, In Proceedings of ACM SOSP'01, Banff, Canada, 2001.
2. S. Banerjee, B. Bhattacharjee, and A. Srinivasan, Resilient multicast using overlays, In Proceedings of ACM SIGMETRICS'03, Jun. 2003.
3. M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, SplitStream: High-Bandwidth Content Distribution in Cooperative Environments, In Proceedings of ACM SOSP'03, Oct. 2003.
4. Y. H. Chu, S. G. Rao, and H. Zhang, A Case for End System Multicast, In Proceedings of ACM SIGMETRICS, Jun. 2000.

5. L. Cherkasova and J. Lee, FastReplica: Efficient Large File Distribution within Content Delivery Networks, In Proceedings of 4th USENIX Symposium on Internet Technologies and Systems, Mar. 2003.
6. S. Chen, B. Shen, S. Wee, and X. Zhang, Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery, In Proceedings of ACM NOSSDAV'03, Jun. 2003.
7. Y. Cui, B. Li, and K. Nahrstedt, oStream: asynchronous streaming multicast, IEEE J. Select Areas in Communication, Jan. 2004.
8. T. Do, K. A. Hua, and M. Tantaoui, P2VoD: providing fault tolerant video-on-demand streaming in peer-to-peer environment, In Proceedings of ICC'04, 2004.
9. P. Eugster, R. Guerraoui, A. M. Kermarrec, and L. Massoulie, From epidemics to distributed computing, IEEE Computer Magazine, 2004.
10. J. Ganesh, A. M. Kermarrec, and L. Massoulie, Peer-to-peer membership management for gossip-based protocols, IEEE Transaction on Computer, 52(2), Feb. 2003.
11. L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang, PROP: a scalable and reliable P2P assisted proxy streaming system, In Proceedings of ICDCS'04, Mar. 2004.
12. Y. Guo, K. Suh, J. Kurose, and D. Towsley, P2Cast: peer-to-peer patching scheme for VoD service, In Proceedings of WWW'03, May 2003.
13. Y. Guo, K. Suh, J. Kurose, and D. Towsley, A peer-to-peer on-demand streaming service and its performance evaluation, In Proceedings of ICME'03, 2003.
14. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, Promise: Peer-to-Peer Media Streaming Using Collectcast, In Proceedings of ACM Multimedia, 2003.
15. M. Hefeeda and B. Bhargava, On-demand media streaming over the Internet, In Proceedings of FTDCS'03, May 2003.
16. S. S. Kien A. Hua, and Y. Cai, Patching: A Multicast Technique for True Video-on-Demand Services, In Proceedings of ACM Multimedia, Sep. 1998.
17. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh, In Proceedings of ACM SOSP, Oct. 2003.
18. X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, AnySee: Peer-to-Peer Live Streaming, In Proceedings of IEEE INFOCOM'06, Apr. 2006.
19. B. Wong, A. Slivkins, and E. G. Sirer, Meridian: A Lightweight Network Location Service without Virtual Coordinates, In Proceedings ACM SIGCOMM'05, Aug. 2005.
20. E. Zegura, K. Calvert, and S. Bhattacharjee, How to model an internetwork, In Proceedings of IEEE INFOCOM, Mar. 1996.
21. X. Zhang, J. Liu, and B. Li, CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming, In Proceedings of IEEE INFOCOM'05, Mar. 2005.
22. M. Zhou and J. Liu, Tree-Assisted Gossiping for Overlay Video Distribution, Technical Report, 2005.