This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Supporting VCR Functions in P2P VoD Services Using Ring-Assisted Overlays

Bin Cheng, Hai Jin, Xiaofei Liao
Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China

***Abstract*—Peer-to-Peer (P2P) networks have been shown to be a promising approach to providing large-scale Video-on-Demand (VoD) services over the Internet for their potential scalability. However, how to efficiently support VCR functions for VoD services in such networks remains a major challenge. In this paper we propose a novel ring-assisted overlay topology, called RINDY, to address this issue. In RINDY, a peer can implement fast relocation of random seeks by maintaining some near neighbors and remote neighbors in a set of concentric rings with power law radii. We explore several key problems in RINDY including ring organization, neighbor placement over rings and neighbor lookup for random seeks. We evaluate its performance through simulations and compare it with some existing approaches. The experimental results show that RINDY outperforms previous schemes in terms of control overhead, latency and quality of streaming, especially under frequent VCR operations.**

Keywords: Video-on-Demand; ring-assisted overlay; peer-to-peer

I. INTRODUCTION

With the widespread penetration of broadband access, Video-on-Demand (VoD) services over the Internet have been one of the most popular Internet applications. The traditional client-server architecture can hardly cope with the ever-growing client population due to its limited service capacity. An alternative method is to deploy dedicated content delivery networks (CDN) [4], in which video data are duplicated on replica servers and user requests are directed to these servers strategically. This approach however suffers from prohibitive cost and complicated management. Recently, overlay networks [7][8] have been shown to be a promising approach to offering high scalability media streaming services. In this kind of systems, nodes help to relay received data to other peer nodes, thereby removing the bottleneck of central servers and further providing better fault tolerance.

While P2P live streaming appears to be evolving towards a mature stage (e.g., [8] [13]), to date few VoD systems have been successfully deployed using overlay networks. This is because, first, VoD systems are supposed to offer greater control over their video streams with more VCR operations, such as resume and random seek. According to the work of Zhang et al [14], the average number of seeks in a session can reaches 5; second, these VCR operations introduce more dynamics to overlay structures and thus make overlay maintenance more challenging.

A variety of overlay construction algorithms have been proposed to address these problems. A typical approach is using tree-based overlays. The drawbacks of this approach are its complicated maintenance, high degree of load imbalance and poor fault tolerance. For example, any bandwidth fluctuation or failure of the upper nodes may cause buffer underflow on a large number of downstream nodes. For on-demand streaming services, frequent topology changes caused by random seeks further aggravate these problems. In contrast to the tree-based overlays, mesh-based overlay networks avoid these problems by introducing a simple random gossip algorithm. For instance, DONet [13] uses this protocol to implement a scalable live streaming system successfully. Although very helpful for nodes to identify peers in a live streaming system, this gossip algorithm is not efficient for P2P VoD systems where nodes usually have different playing offsets across a wide range.

To this end, we propose a novel ring-assisted overlay management scheme, called RINDY. In RINDY, each peer maintains a set of concentric rings with power law radii [10] and places all neighbors on these rings according to their relative distances. More specifically, near neighbors with overlapped buffer windows with the current node are placed on the innermost ring as backup data suppliers; while some randomly sampled remote neighbors are placed on the outer rings as routers to nodes with playing positions specified by VCR operations. Under this scheme, a peer only needs $O(\log (T/w))$ hops (where T is the total time length of the video stream and w is the buffer window size of peers) to identify and connect to a new group of peers close to the destination playing positions when a random seek occurs. RINDY has the following features: first, it inherits good reliability from the random gossip protocol in mesh-based overlays; second, peers look for neighbors only from local rings, thereby avoiding the load imbalance problem of tree-based overlays; third, the remote neighbors provide a bridge to efficiently find new desired neighbors for random seeks.

The remainder of this paper is organized as follows. In Section II, we document some related work; Section III introduces some basic concepts of our ring-assisted overlays; Section IV presents the design of ring-assisted overlays and discusses some key problems in details; Section V evaluates its

This work was supported by China National Natural Science Foundation (NSFC) under grant No.60433040, 60642010 and by CNGI 2005 Projects under grant No.CNGI-04-12-2A, CNGI-04-12-1D.

1-4244-0353-7/07/$25.00 ©2007 IEEE

performance through simulation experiments; and Section VI concludes the paper.

II. Related Work

Many tree-based overlays are proposed to provide p2p VoD services, such as P2Cast [5], oStream [1], DirectStream [6]. They build an application-layer multicast tree to provide basic stream and search for appropriate patching streams for later coming peers from the root peer. Similar to the tree-based schemes, P2VoD [2] organizes nodes into multi-level clusters according to their joining times. A new node tries to join the lowest cluster or forms a new lowest cluster. For these two kinds of overlays, the major problem is that the one-to-one data delivery model is not enough efficient under a heterogeneous network environment. Additionally, it is not easy to maintain a consistent tree or layer structure in a highly dynamic overlay network.

Mesh-based overlays have been used in pcVoD [11], PROMISE [7]. Although they improve bandwidth utilization by fetching data from multiple data suppliers, they have the problem of how to efficiently locate destination nodes to support VCR operations in VoD services, such as random seeks. A pure mesh-based overlay may lead to an unacceptable latency for random seeks.

Beside these two solutions above, DSL [9] provides a dynamic skip list overlay. It tries to maintain a global skip list with multi-layers and executes a top-down search procedure for each random seek, which always causes load imbalance of control overhead among peers. In contrast, in RINDY each peer performs lookup operations from its local rings and thus avoids this problem.

III. Prelimilaries

In RINDY, a video is divided into a series of timeslots and each timeslot encapsulates the data content of one second. Each peer maintains a sliding buffer window to manage the most recently received timeslots. The buffer window can contain at most w timeslots. A peer can possibly exchange data with other peers only if their buffering windows are overlapped.

In RINDY, the overlay is managed through three layers, as shown in Figure 1. The first layer is the member cache, called *mCache*, which keeps the records of some peers watching the current video. It can be seen as a large table, which is initialized by the peer list fetched from the bootstrap server (named tracker) and is updated by received gossip messages. The second layer is the *neighbor list* which forms a ring-assisted control overlay. There is a persistent TCP connection between any two neighbors, over which gossip messages are disseminated. The third layer is the *partner list*, which constructs the data overlay of RINDY. The timeslots are only exchanged between partners.

To accommodate the dynamics of overlay topology and improve the quality of streaming, we use a dynamic selection mechanism to promote better candidates to the upper layer. A peer first selects some nearest peers from mCache as its neighbors according to their distance measured by the difference between their playing positions. Then, it selects some neighbors from the neighbor list into its partner list, based on their buffer map proximity that can be calculated by the timeslot availability of their buffer windows. If a partner has not been exchanging any data packets for a predefined time interval, it will be removed from the partner list and a promotion procedure to find new partners is triggered; likewise, if a peer record of mCache has not been updated for a predefined time interval, it will be discarded from mCache.

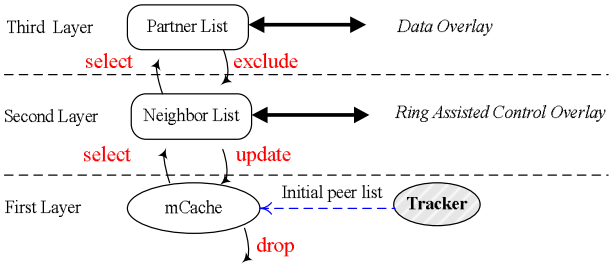

Figure 1. Overlay organization of RINDY

IV. Ring-Assisted Overlay

In this section we present the design of RINDY, particularly its support for random seeks.

A. Structure of Ring

In RINDY, each peer keeps track of a small and partial view of the total overlay and organizes all its neighbors into a series of concentric, non-overlapping logical rings according to their relative distances. The radius of the ith ring is $w*2^i$, where w is the length of buffer window. We can see that the ith ring covers the area between the two circles with radii $w*2^{i-1}$ and $w*2^i$. The distance between any two peers is calculated from their current playing positions. For instance, the distance from peer j to peer i is $d_j = cur_j - cur_i$. If d_j is negative, then the playing position of peer j is behind of that of peer i and we call peer j a *back-neighbor* of peer i; otherwise we call peer j a *front-neighbor* of peer i. Note that the relative distance between two peers will remain unchanged if no VCR operations are performed by them. With these definitions, it is easy to see that a neighbor j's ring number with respect to peer i is the floor of $log\ (|d|/w)$.

Given a certain limit of neighbor pool size maintained by a peer, the distribution of the neighbors over rings has important influence on the performance of VCR operations and member discovery. On the one hand, a peer needs many neighbors with close playing positions so as to have plenty of choices of finding data suppliers; on the other hand, a neighbor list that is temporally diversified can help forward a VCR query to a wide region more efficiently. So we introduce two types of rings, *gossip-ring* and *skip-ring,* to organize all neighbors. A peer's innermost ring (whose ring number is zero) is mainly responsible for collecting some near neighbors with close playing positions and overlapped buffer windows. We call this ring a *gossip-ring*. For all outer rings, a peer samples some *far-neighbors* to help forward the queries of seeks. These rings are mainly used to improve the speed of lookup operations and reduce the load of the tracker server. We call these rings *skip-rings*. For the gossip-ring, each peer keeps track of at most ***m***

neighbors, called *near-neighbors*; for each skip-ring, a peer tries to maintain at most ***k*** *front far-neighbors* and ***k*** *back far-neighbors*.

Figure 2 presents an example of neighbor distribution over rings for peer P. Neighbor A is a near-neighbor in the gossip-ring and neighbors B, C, D, E, F and G are far-neighbors in the skip-rings. Neighbors C, E, G are its front far-neighbors and neighbors B, D, F are its back far-neighbors. Peer P can propagate gossip messages through its near neighbors; it can also identify new members from the received gossip messages. To find good partners from its near neighbors, peer P should maintain as many near-neighbors as possible under the constraint of control overhead. Additionally, all of neighbors in its skip-rings serve as bridges to look up potentially good partners when it performs a random seek. Because the seek operation can be forward or backward, it is necessary to ensure that there are at least two connections in each skip-ring, one connecting to a front far-neighbor and the other to a back far-neighbor.

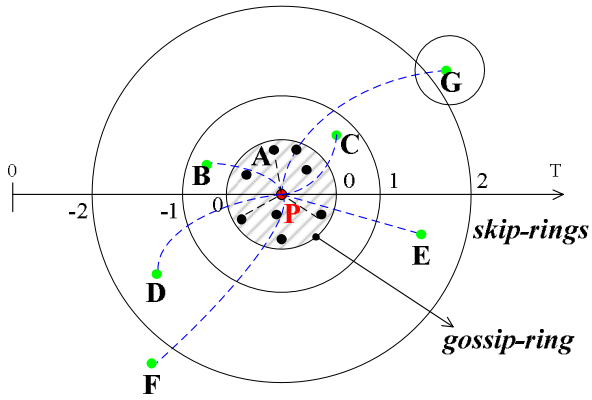

Figure 2. Neighbor distribution over rings for peer P

B. Joining of New Peer

We assume that a well-known rendezvous point, called *tracker*, is in place assisting the joining of new peers. While for most of existing schemes the tracker indexes all joined peers, for RINDY the tracker only indexes a small fraction of participated peers for each video with a lower overhead. In RINDY, the tracker is virtually a peer with a fixed position 0, only keeping track of recent near-neighbors and far-neighbors. When a new peer joins, it first contacts the tracker and the tracker picks up some near-neighbors and far-neighbors from its local rings as an initial peer list of the newly-joined peer. Then the tracker tries to promote the new peer as a new near-neighbor. In order to help new peers quickly construct its own ring set, the tracker needs to provide enough initial peers for them. Therefore, the tracker always keeps more neighbors for each ring compared with normal peers.

C. Gossiping over Rings

In RINDY, each peer periodically sends gossip messages to declare its existence and its buffer window status and updates its mCache by received gossip messages. The content of a gossip message includes the fields *GUID, Peer_Address, number_neighbor, cur, TTL, max, min, end* and *Avail*, where *GUID*, *Peer_Address*, *number_neighbor* and *cur* represent the global identifier, the network address, number of neighbors, and the current playing position, respectively; *TTL* is the remaining hop count of this message; *max, min, end* and *Avail* denote a snapshot of the moving buffer window. We design two types of gossip messages, *ANNOUNCE* and *FORWARD*. The difference between them is that *ANNOUNCE* messages carry the buffer snapshot while *FORWARD* messages do not. Since any two peers can possibly share data only if their distance is less than the length of buffer windows, it is unnecessary for a peer to send gossip messages to the peers out of its gossip-ring. Based on this, we design a *scoped gossip* protocol over rings to help a peer find more neighbor candidates. In our algorithm each peer periodically sends an *ANNOUNCE* gossip message with an initial *TTL* to all of near neighbors. Then these near neighbors select one random near-neighbor from their neighborhoods, change the message type from *ANNOUNCE* to *FORWARD*, decrease its TTL and finally forward this message to the selected neighbor. This procedure is iterated until the TTL becomes zero or the selected neighbor is the source of this message.

Beside gossip messages, each peer also sends EXCHANGE messages to all its far-neighbors periodically. This kind of message can be used to exchange the near-neighbor list with its far-neighbors to get some backup far-neighbors; it can also be used to detect the failure of some far-neighbors. When a far-neighbor in some skip-ring goes away, a peer can select a new far-neighbor for this skip-ring from those backup far-neighbors.

Since peers can randomly jump to any playing position or depart the system, the existing entries in mCache may become outdated. Here we define a time-to-live for each entry in mCache. If an mCache item has not been updated by gossip messages for a threshold time interval, it will be removed from the mCache.

D. Lookup over Rings

The lookup operations based on skip-rings mainly occur in the following two cases. First, when a peer performs a VCR operation, a lookup procedure will be triggered to find some new neighbors close to the target position. Second, when a peer finds out that there are not enough neighbors in a certain skip-ring due to the failure or departure of far-neighbors, it also executes a lookup procedure to find more backup neighbors for that ring.

We design two types of messages, REQUEST and REPLY, to efficiently locate new neighbors required by VCR operations. The content of messages include the fields *type*, *dest, src_peer*, *routing_path*, and *result_set*, where *type* represents the type of message, such as REQUEST or REPLY; *dest* is the target position specified by VCR operations; *src_peer* denotes the source address of the message; *routing_path* is a list of peers traversed by the request message; and the field *result_set* is used to store the result of query. Suppose that a peer seeks to the target position *d*. It first checks whether *d* is within its gossip-ring. If *d* is in its gossip-ring, it performs a local search to find enough near-neighbors and far-neighbors, otherwise it finds the closest neighbor to *d* as its

next hop, pushes its address into the *routing_path* and *result_set* fields of the query message, and finally forwards this query to that next-hop neighbor. Upon receipt of this query, the next-hop neighbor executes the same procedure, which will be iterated until this request arrives at some peer whose gossip-ring covers *d*. The final peer adds all of its near-neighbors into the *result_set* field of the query message, changes the type of this message to REPLY, and finally returns the message to the source peer along the routing path. When the source peer receives the REPLY message, it adds all members of *result_set* into its mCache and then changes its current playing position to *d*.

Figure 3 presents a diagram to illustrate the lookup procedure. Peer P wishes to jump to the target position *d*. It first checks whether *d* is in its gossip-ring. In this example, *d* is in the 2nd ring and P_1 is the currently closest neighbor to *d*. So peer P sends a query to P_1. When P_1 receives this query, it knows that the destination *d* is in its first skip-ring and P_2 is the closest neighbor to *d*, so it adds its address into the *routing_path* and *result_set* fields and forwards this query to P_2. Finally, when P_2 receives this query it finds out that the *d* is in its gossip ring; P_2 then adds all near-neighbors of its gossip ring (i.e., P_3, P_4) into the *result_set* field and returns the message to peer P along the path contained in the *routing_path* field in the message.

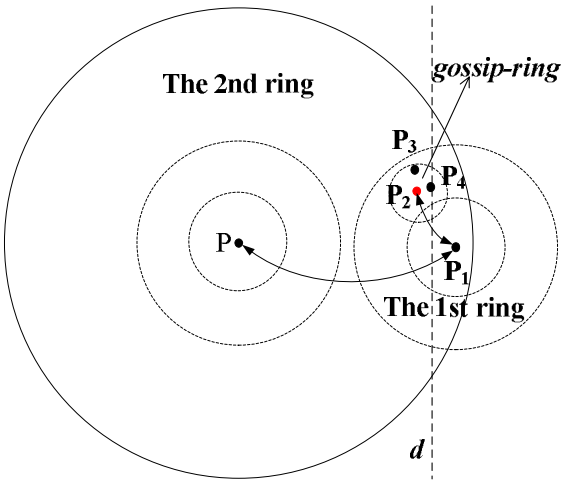

Figure 3. Procedure of a lookup operation

V. PERFORMANCE EVALUATION

We use GT-ITM [12] topology generator to create a topology of 1000 peer nodes based on the 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 experiments, 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 that stores all of media contents on a transit node. The bandwidth settings between two transit nodes, a transit node and a stub node are 100Mbps and 10Mbps, respectively. The out-bound bandwidths of stub nodes are heterogeneous, including 4Mbps, 1Mbps, and 512Kbps. In addition, we choose a movie with 60 KB/s streaming rate and 60 minutes content as our testing stream. Some other important parameters are given in Table I.

TABLE I. PARAMETERS LIST

Parameter	Value and description
w	300 seconds, buffer window size
B	5, maximum ring number
TTL	5, maximum hop number for gossip messages
t	30 seconds, gossip period
m	15, number of near-neighbors in each gossip-ring
k	1, number of front or back far-neighbors in each skip-ring
n	10, number of partners
T_p	3 minutes, checking period of partners in partner list
T_m	5 minutes, checking period of members in mCache

A. Control Overhead

We first investigate the control overhead of overlay construction and maintenance in our scheme. Here the control overhead is measured by the number of processed messages. We record the average number of processed messages of each peer in a gossip period in the absence of peers failure and VCR operations, and then we calculate the average number of messages of all peers. From Figure 4, we can see that the overhead increases very slowly after the total number of peers reaches 400. When the total number of peers exceeds 1000, the control overhead remains around 150. Since the gossip period is 30 seconds, each peer only needs to process about 5 messages in a second. Compared with the high-bandwidth streaming traffic, this overhead is negligible. Figure 5 shows the average number of messages per peer when 10 percent of peers join, leave the network and perform VCR operations randomly. When the total number of peers reaches about 400, the average control overhead to accommodate the overlay changes brought by peer joining, leaving or seeking keeps at a constant level. This indicates that the control overhead caused by peer dynamics is independent of the size of network. That is mainly because for each peer the number of rings is fixed, up to the time length of the current movie (60 minutes). In this case, the number of messages can be constrained by the maximal number of near-neighbor (m), the maximal number of far-neighbor (k) and the *TTL*.

Figure 6 compares the control overheads of normal gossip protocol [3] and our scoped gossip protocol over rings under a stable status of the overlay. The gap shows a reduction of control overhead of 15-66% by our protocol, which is an encouraging improvement over the traditional mesh-based p2p VoD systems. That is mainly because our scoped gossiping algorithm can efficiently restrict the spreading range of gossip messages of each peer in their gossip-rings. Therefore, many inefficient gossip messages are discarded and only those close peers can receive them.

B. Server Stress

We study the effect of data sharing among partners in RINDY by examining the stress of source server under different arrival rates and buffer window sizes. The server stress is measured by the number of streams supported by the source server. First, we explore the effect of arrival rate with a

fixed buffer window size (w = 300s). We start 1,000 peer nodes with varying arrival rates to play 30 minutes; the simulation is run for two hours. Figure 7 plots the server stresses for arrival rates varying from 0.001 to 100. We can see that the server stress is very low when the arrival rate is in the range from 0.1 to 10. When the arrival rate is less than 0.1 we find that a lower arrival rate leads to a higher server stress. This is because that the sparse peer arrivals make peers' buffer windows overlap very little, so most media sessions cannot be established directly between peers. The server stress almost remains unchanged for arrival rates between 0.1 and 10, showing a balance between the effect of buffer sharing and the requirement of client population. When the arrival rate is beyond 10, the server stress rises noticeably with the increase of arrival rate. We find that a flash crowd occurs in this case and many peers request a lot of same timeslots, which increases the load of the source server.

Figure 8 shows the server stress for varying window sizes. Intuitively, a large buffer size decreases the server stress. However, from Figure 7, we find that it actually brings little benefit for a relatively high arrival rate of 1; this is because the dense arrival of peers has already caused significant overlapping of peers' buffers, making a large buffer size unnecessary. When the arrival rate is far lower, the increase of buffer window size significantly improves the performance. This phenomenon suggests that it is unnecessary for popular files to assign a large buffer window, while it is useful for system to assign a large buffer window for cold files.

C. RINDY vs. P2VoD

Both RINDY and P2VoD (see the introduction in Section II) cache recently watched media data in local memory to relay to other peers, so they are similar with respect to buffer management. We compare the performance of RINDY and P2VoD in terms of server stress, quality of streaming, latency and load balance. In our experiments, P2VoD uses Smallest Delay Selection as its parent selection scheme. Additionally, the maximum number of clients allowed in the first generation of each session is 8 and the buffer window size is 300 seconds.

1) Server Stress

Figure 9 shows the source server stress caused by RINDY and P2VoD with different numbers of nodes. The arrival rate of client is 1 per second. Note that the server stress of RINDY remains at 6~7 streams when the number of nodes increases. In contrast, for P2VoD, the server stress increases almost linearly (from 8 to 23). This is mainly because that in P2VoD, each peer receives data from only one parent; and consequently, a peer's residual bandwidth will be wasted if it cannot support one more child. When a newly arriving client fails to find a peer capable of supporting a full stream, it has to create a new session from the source server, even though there may exist some peers whose aggregate bandwidth is greater than that of a

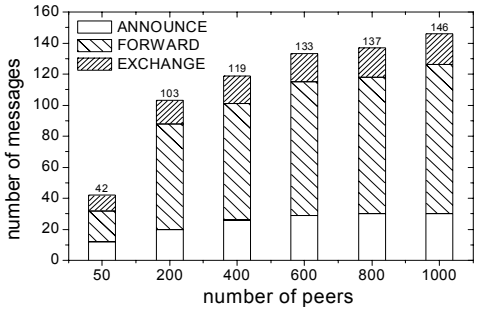

Fig. 4 Control overhead under a stable status

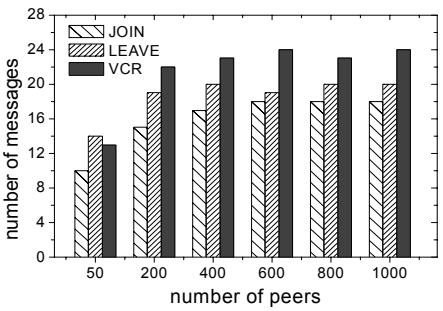

Fig. 5 Control overhead under a dynamic status

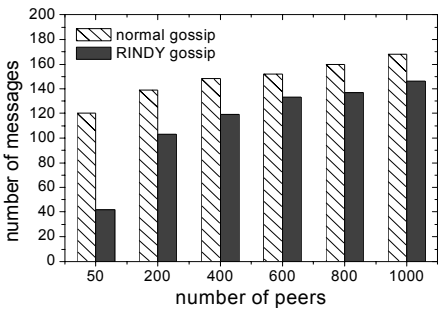

Fig. 6 normal gossip vs. scoped gossip

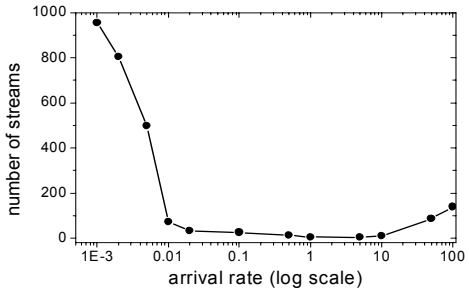

Fig. 7 Server stress for varying arrival rates

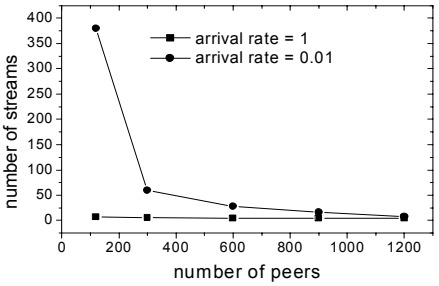

Fig. 8 Server stress for varying buffer window sizes

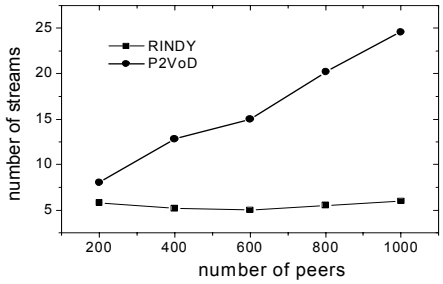

Fig. 9 Server stresses (RINDY vs. P2VoD)

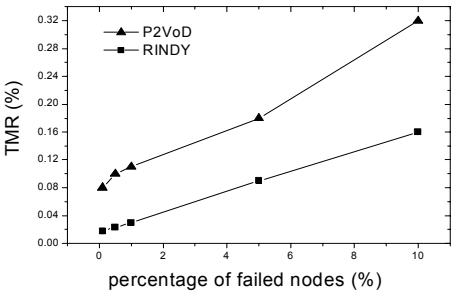

Fig. 10 Quality of streaming (RINDY vs. P2VoD)

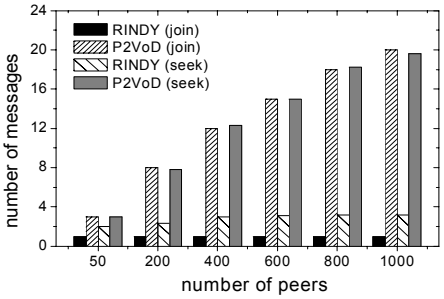

Fig. 11 Control overhead for joining and random seeking

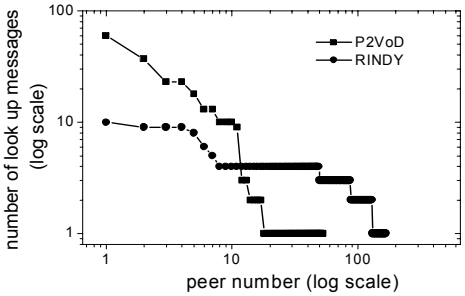

Fig. 12 Distributions of lookup messages

full stream.

2) Quality of Streaming

We compare the reliability of RINDY and P2VoD by "shutting down" some peers. Initially we start 1000 peers with an arrival rate of 2. After all peers have started and played for a while, we randomly stop some joined peers at a speed of 2 peers per minute, and then calculate the average timeslot missing rate (TMR) for the remained peers. TMR is measured by the number of missed timeslots divided by the total number of timeslots. We repeat this experiment ten times. Figure 10 presents the result of timeslot missing rate with different percentages of node failure. We can see that RINDY has a lower TMR than P2VoD for the same percentage of node failure. That means RINDY achieves better reliability by using gossip protocol over rings and retrieving data packets from multiple partners.

3) Latency

For VoD service, latency is one of the important metrics. We mainly consider the average latency of joining and random seeking in terms of average number of message hops. Figure 11 reports the results of RINDY and P2VoD. In RINDY, a new peer only needs to send one request to the tracker server to join the overlay network. In P2VoD, a new peer has to contact many overlay members to find an appropriate parent; furthermore, the new peer always starts to play from the beginning of current movie and its request will be forwarded to the lowest generation. As a result a peer in P2VoD takes more time to join the overlay and the joining latency increases with the number of peers. Figure 11 also presents the average number of message hops of random seeking in RINDY and P2VoD. In RINDY, the average number of message hops for random seeking is about 3 hops and does not increase with the number of peers. For P2VoD, this number is close to that of RINDY when the number of peers is very small (e.g., less than 100). But in the presence of a large number (e.g., 1000) of nodes, the average number of hops reaches 18, some 5 times higher than that of RINDY. The main reason is that RINDY constructs a local ring set based on playing position for each peer to forward its requests. In contrast, a peer in P2VoD always searches from the root node to find a new parent in a way similar to the joining procedure. From the results shown in Figure 11, RINDY significantly decreases the latency of random seeking operations as compared with P2VoD.

4) Load Balance

Figure 12 shows the distribution of lookup request messages processed by each peers in RINDY and P2VoD when the total number of peers is 600 and 10% of peers perform VCR operations randomly. We can see that, in RINDY, most peers process 0~8 messages and a few peers process up to 10 messages, but none of them process more than 20 messages. In P2VoD, most peers process 0~3 messages but the number of lookup messages of some peer nodes reaches 30 or 40 and the root node processes up to 60 messages. Compared with tree-based overlays as in P2VoD, our ring-assisted overlays obtain better load balance among peers during VCR operations. There are two main reasons behind this result. First, in our ring-based overlay network, each peer has its own neighbor organization and its lookup operations begin with its local rings. In contrast, for the tree-based overlay network, all lookup operations start searching from the root node, and so the root node and the peers at upper layers need to process more lookup requests, which obviously results in load imbalance among peer nodes. Second, a peer in RINDY has a set of skip rings which help to significantly decrease the length of lookup path.

VI. CONCLUSIONS

We propose a novel ring-assisted overlay network for peer-to-peer Video-on-Demand services, called RINDY. In RINDY, each peer maintains some near neighbors and remote neighbors in a set of concentric rings with power-law radii, efficiently supporting VCR functions in p2p VoD services, especially random seeks. Additionally, its maintenance algorithm is easier to implement compared with tree-based overlay networks. We also investigate the control overheads of RINDY by simulation experiments. The experimental results show that RINDY has a low control overhead on gossiping and lookup operations, and is superior to a previous scheme in terms of latency, quality of streaming and load balance.

REFERENCES

[1] Y. Cui, B. C. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," *Journal on Selected Areas in Communications*, vol. 22, no. 1, 2004.

[2] 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*, Jun. 2004.

[3] J. Ganesh, A. M. Kermarrec, and L. Massoulie, "Peer-to-peer memebership management for gossip-based protocols," IEEE Transaction on Computer, 52(2), Feb. 2003.

[4] 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*, Tokyo, Mar. 2004.

[5] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: peer-to-peer patching scheme for VoD service," In *Proceedings of WWW'03*, Budapest, Hungary, May 2003.

[6] 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,* Jul. 2003.

[7] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, Promise: Peer-to-Peer Media Streaming Using Collectcast, Proceedings of ACM Multimedia, Nov. 2003.

[8] X. F. Liao, H. Jin, Y. H. Liu, Lionel M. Ni, and D. F. Deng, "AnySee: Peer-to-Peer Live Streaming Service," In *Proceedings of IEEE INFOCOM'06*, Apr. 2006.

[9] D. Wang and J. Liu, "Peer-to-Peer Asynchronous Video Streaming using Skip List," In *Proceedings of ICME'06*, Canada, Jul. 2006.

[10] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A Lightweight Network Location Service without Virtual Coordinates," In *Proceedings ACM SIGCOMM'05*, Aug. 2005.

[11] L. H. Ying and A. Basu, "pcVOD: Internet Peer-to-Peer Video-On-Demand with Storage Caching on Peers," In *Proceedings of the Eleventh International Conference on Distributed Multimedia Systems DMS'05*, Canada, 2005.

[12] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," In *Proceedings of IEEE INFOCOMM'96*, Mar. 1996.

[13] X. Zhang, J. Liu, B. Li, and T. S. P. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," In *Proceedings of IEEE INFOCOM'05*, Mar. 2005.

[14] C.X. Zheng, G.B. Shen, and S.P. Li, "Distributed prefetching scheme for random seek support in peer-to-peer streaming applications," In *Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming(P2PMS'05)*, Hilton, Singapore, 2005.