Supporting VCR-like Operations in Derivative Tree-Based P2P Streaming Systems

Tianyin Xu*, Jianzhong Chen*, Wenzhong Li*, Sanglu Lu*, Yang Guo†, Mounir Hamdi‡
*State Key Lab. for Novel Software and Technology, Nanjing University, Nanjing 210093, China
†Corporate Research, Thomson Inc., Princeton, NJ 08540, USA
‡Hong Kong University of Science and Technology, Kowloon, Hong Kong

Abstract— **Supporting user interactivity in peer-to-peer streaming systems is challenging. VCR-like operations, such as random seek, pause, fast forward and rewind, require timely P2P overlay topology adjustment and appropriate bandwidth resource re-allocation. If not handled properly, the dynamics caused by user interactivity may severely deteriorate users' perceived video quality, e.g., longer start-up delay, frequent playback freezing, or blackout altogether. In this paper, we propose a derivative tree-based overlay management scheme to support user interactivity in P2P streaming system. Derivative tree takes advantage of well organized buffer overlapping to support asynchronous user requests while brings high resilience to the impact of VCR-like operations. A session discovery service is introduced to quickly locate parent peer. We show that the overhead of VCR-like operations in derivative-tree based scheme is $O(log(N))$, where N is the number of sessions. Simulation experiments further demonstrate the efficiency of the proposed scheme.**

I. Introduction

Providing media streaming service over the Internet has become immensely popular in recent years, with the proliferation of emerging applications including Internet TV, online video, distance education, etc. Peer-to-peer (P2P) technology has been proved to be an effective and resilient approach for media streaming in dynamic and heterogeneous network environments [1][2][3][4][5]. In P2P media streaming systems, peers actively cache media contents and further relay them to other peers that are expecting these contents. Compared to traditional server-based solutions or content delivery networks (CDNs), the cache-and-relay mechanism can effectively alleviate server workload, save server bandwidth consumed and thus bring high system resilience and scalability.

P2P live streaming, a typical media streaming service designed for all peers receiving streamed video at the same playback point, has been extensively studied and widely deployed in the past ten years [6][7][1][2]. Recently, researchers show great interests in P2P *interactive* streaming, a new genre of P2P streaming service that supports not only asynchronous user requests but also VCR-like user interactivity including random seek, pause, fast forward and rewind [4][5].

Supporting interactive streaming in P2P system is a challenging task. On one hand, asynchronous user requests result in caching different parts of video files by the peers, which affects the efficiency of coordinating content delivery. On the other hand, frequent VCR-like interactive operations lead to peer churn, i.e., peers dynamically leave their current positions and "jump" to a new playback position. Besides, frequent VCR-like operations cause long latency and high network overhead for re-establishing streaming services. The tree-based [8][9] approaches organize nodes into a multicast tree for delivering media streaming, with the media source as the root of the tree. The gossip-based approaches [10][11] maintain a series of supplier candidates in each node. Nodes periodically exchange data availability information to update their candidate lists. However, none of these work overcomes the high network overhead caused by VCR-like operations.

In this paper, we propose *DTStream*, a peer-to-peer media streaming system supporting VCR-like interactive operations. Based on the observation that users' viewing quality is seriously affected by their suppliers' frequently leaving current positions for VCR-like operations, our objective is to devise a novel P2P scheme to eliminate the impact of user interactivity. In the proposed scheme, we explore the key design issues including node organization scheme, media content discovery, and user interactivity support. Specifically we made the following contributions:

1) We introduce *Derivative Tree* (DT), a novel distributed overlay structure with inherent ability to organize dynamic and asynchronous peers while bring high resilience to peer churn. The derivative tree has advantages such as well structured, controllable start-up delay, less memory consumption and quick service construction. Especially, it effectively reduces the impact of dynamic node join and departure with low network overhead.
2) We propose VCR-like operation implementation in the P2P streaming system. Using an efficient session discovery service assisted by DHT, the overhead of VCR-like operations is proved to be in $log(N)$.
3) The efficiency of the proposed scheme is confirmed using extensive simulations. The result shows that DTStream outperforms the existing representative systems by reducing the VCR impact over 50%.

The rest of the paper is organized as follows. Section II overviews related work. Section III introduces the system model. Section IV describes the derivative tree and its buffer management. The VCR-like operations are presented in Section V. We evaluate the system performance by simulations in Section VI. Section VII concludes the paper and poses the future work.

II. Related Work

Nowadays, using P2P technology to support interactive video streaming attracts great research interests. There has been a lot of work on providing continuous P2P on-demand streaming, aiming at supporting VCR-like operations [9-14].

Most of existing work organizes peers into structured (e.g. *tree*) or unstructured (e.g. *gossip*) overlays to provide streaming service and resource relocation when user interactions occur. P2VoD [9] defines *generation* to organize nodes according to their joining time, where nodes in the same generation store the same segments in their playback buffers. However, although P2VoD supports asynchronous requests effectively, it does not perform well under frequent user interactions. VMesh [12] and BBTU [13] use dedicated storage in each node to store segments across the P2P system. The user interactivity is supported by continuously searching required segments in the system. DSL [14] organizes all the nodes into a dynamic skip list overlay. However, DSL does not make full use of playback buffers as a node only chooses its left neighbor as its supplier. RINDY [10] designs a gossip-based ring-assisted overlay to implement fast relocation of random seeks. In RINDY, each node maintains some near and remote neighbors in a set of rings according to their relative distances.

However, little attention has been paid to the network overhead caused by frequent VCR-like operations. In this paper, we introduce an efficient P2P system for media streaming service with low cost VCR-like interactive operations.

III. System Overview

The DTStream system mainly consists of three components: *media servers*, *sessions* and a *SessionCircle*, which is illustrated in Fig. 1.

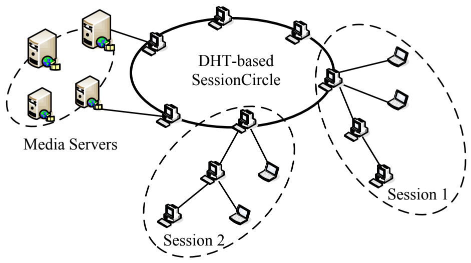

Fig. 1. The system model of DTStream

Media servers simply provide media streaming service. All video streams are initiated from the media servers, delivered through the distributed P2P overlay, and finally reached to end users. We assume media servers do not provide any VCR functions, thus VCR-like operations are supported by the coordination of peers in the overlay scheme. Peers with similar playback positions are organized as sessions. Nodes in a session form a tree-like structure, which is called derivative tree. Each session is composed of a set of nodes whose playback position differences are within a threshold. If a node performs VCR-like operations, i.e., jumping to a new playing position beyond the threshold, it will join another session or initiate a new session in the system.

TABLE I
Notations

Not.	Description	Not.	Description
n	Number of nodes	m	Number of nodes in a tree
N	Number of sessions	s_x	Split line position of x
α	Overlapping ratio	p_x	Playback position of x
B	Playback buffer length	h_x	Buffer head of x
K	Upload bandwidth capacity	t_x	Buffer tail of x
H	Height of a derivative tree	d_x	Pre-fetched length of x

The SessionCircle connects the root nodes of all sessions to form a ring structure. Distributed Hashing Table (DHT) is used in the ring for efficient information searching. Based on DHT, a session discovery service (SDS) is implemented for quick resource location and service re-establishing.

Similar to [6][9][10], a video stream is divided into segments of uniform length and each segment is equivalent to one unit of playback time (usually 1 second). In DTStream, we assume the playback rate is at constant bit-rate. Each node maintains a size-limited sliding buffer window to cache the most recently received segments, which could be supplied to other subsequent peers.

IV. Derivative Tree-Based Overlay Scheme

In this section, we introduce the derivative tree-based scheme for organizing peers in a session to support media streaming service. The notations used in this paper are summarized in Table I.

A. Derivative Tree

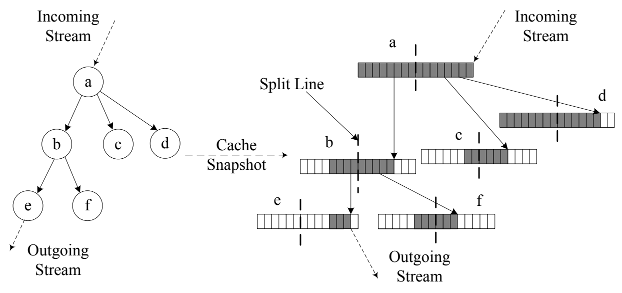

Fig. 2. A derivative tree and its buffer snapshot with $\alpha = \frac{1}{2}$

A derivative tree is a data delivery tree with streaming data entering the root node and then delivering to all nodes in the tree. It looks like an overlay multicast tree, but it has special buffer management strategies and construction rules. Unlike overlay multicast tree where similar contents are cached by each node, derivative tree employs an overlap cache design: only parts of buffer contents of the nodes are similar, other parts are different. Assume each node has the same playback buffer length B units of streaming segments (Note that the model can be easily extended to the conditions where B is different). For each node, its playback buffer is split into two parts by a *split line*: $\alpha \times B$ and $(1 - \alpha) \times B$, where α is a tunable parameter called *overlapping ratio*, indicating the degree of two nodes' buffer overlapped. Fig. 2 shows a derivative tree with $\alpha = \frac{1}{2}$. A node x is a child of node y only if its playback point p_x is within y's buffer, which guarantees x can obtain playing segments from y directly. Assume x is y's child and the split line of y is s_y. If $p_x < s_y$, x is called a *left child* of y; if $p_x \geq s_y$, it is a *right child*. The following rule is applied in constructing a derivative tree: for each node in the tree, if it is root or a left child, it may have a number of right children and no more than one left child; if it is a right child, it cannot have children any more. Based on this rule, each left child can be viewed as the root of a sub-tree derived from the original tree, which is the reason we use the word "derivative". If a node has no child, it is called an *external node* (EN), otherwise it is called an *internal node* (IN). According to the above rule, a right child's buffer is completely overlapped with its parent, yet a left child only has $\alpha \times B$ overlapped. Thus, the derivative tree supports asynchronous user request: a descendant's playback position is different from its ancestor's.

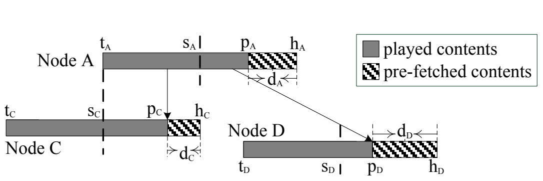

Fig. 3. Buffer contents in a part of derivative tree

When a child node starts getting streaming contents from its parent node, the relative distance of the playback positions between the two nodes will remain unchangeable if no VCR-like operation is performed. We use p_x to depict the playback position of node x, and d_x denote the distance between p_x and the buffer head h_x (pre-fetched contents). Fig. 3 shows a part of derivative tree nodes and their buffers. The following theorem gives the properties of derivative tree.

Theorem 1: Assume node C/D is a left/right child of node A, as shown in Fig. 3. Suppose the height of the tree is H, the number of nodes in the tree is m, and the upload bandwidth capacity of each node is K (at most K children for each node). We have

(1) The playback position of C and D satisfies:

$t_A \leq p_C < s_A$, $s_A \leq p_D \leq h_A$;

(2) The number of nodes in the tree satisfies:

$H + 1 \leq m \leq K \times H + 1$

(3) In the overlapped caching scheme, data range of a session is $[h_{root} - (1 + (1-\alpha) \times H) \times B),\ h_{root}]$, and the length of caching content is $(1 + (1-\alpha) \times H) \times B$.

The proof of the theorem is straightforward and is omitted due to page limit.

Derivative tree has the following main features: Firstly, it supports asynchronous user requests by using an overlapping buffer design. Secondly, it is well structured and controllable. We can trade off the network latency and message overhead by adjusting the overlapping ratio and the height of the tree. Thirdly, it provides low cost VCR-like operations and quick streaming service reconstruction, as discussed in the later sections.

B. Node Join

The join operation is simple. When a new node x joins a derivative tree, it first contacts the root node r, comparing its request point p_x with the root's split line s_r. If $s_r \leq p_x \leq h_r$ and r has plenty of bandwidth, x will be assigned as a right child of the root. If $t_r \leq p_x < s_r$ and r has no left child, x will be the left child of r. Otherwise, x will trigger a join operation to the sub-tree from the left child of r. The process is repeated recursively. If none of the above situation is satisfied, the join operation is failed.

The following theorem shows the start-up delay for join operations.

Theorem 2: The join operation yields $O(\frac{H+1}{2} + C)$ hops in average, where $C = \frac{1}{1-\alpha}$.

Proof: Consider a node x joins a derivative tree with height H. Let $P(H)$ be the probability that x become a right child of the root. According to Theorem 1, the caching content length of the session is $(1 + (1-\alpha) \times H) \times B$. Assume the request position of x is uniformly distributed in the range. Only when $s_r \leq p_x \leq h_r$, it will become a right child of the root. The probability is

$$P(H) = \frac{(1-\alpha) \times B}{(1 + (1-\alpha) \times H) \times B} = \frac{1}{H + C}$$

With probability of $1 - P(H)$ the node x will join a sub-tree of height $H - 1$ recursively. The average hops can be calculated by the following recurrence equation

$$\begin{aligned} T(H) &= P(H) \times 1 + (1 - P(H)) \times (1 + T(H-1)) \\ &= 1 + \frac{H + C - 1}{H + C} \times T(H-1). \end{aligned}$$

Solution to the recurrence equation yields

$$\begin{aligned} T(H) &= \frac{(H + 2C)(H+1)}{2(H+C)} \\ &= \frac{H+1}{2} + \frac{C(H+1)}{2(H+C)} \\ &\leq \frac{H+1}{2} + C. \end{aligned}$$

As C is a constant, the average hops is $O(\frac{H+1}{2} + C)$. ■

According to the theorem, the average start-up delay of join operation is $O(\frac{H+1}{2} + C)$. By choosing H and α carefully, the start-up delay can be restricted in a reasonable range.

Centralized directory can be used to facilitate join operations. An efficient implementation is maintaining a *left offspring table* (LOT) in the root node, which contains <IP address, sub-range> pairs of all left children in the tree. When a new node joins, it lookups the table and locates the destination node. The following theorem shows join operations in constant hops.

Theorem 3: Using LOT implementation, the join operation is in $O(2)$ hops.

Proof: If a node x joins as a right child of the root, it only takes 1 hop. If not, the node lookups the LOT in the root node and locates the target offspring, and then joins as its child. It needs at most 2 hops for x to join the tree. So the join operation is in $O(2)$ hops. ■

C. Node Departure

Interactive streaming service causes much more frequent node departure than live streaming, which affects the viewing quality of the nodes in the downstream. In this session, we will show the efficiency of the DT-based overlay scheme in reducing node departure affects. Assume node x is leaving, the departure operation is handled according to the following cases:

(1) If x is an external node, its departure will not affect the tree structure and other nodes.

(2) If x is an internal node with right children, its departure will leave a "vacant" node in the tree. In order to keep the derivative tree structure, a right child of x with the closest playback position is chosen to fill the vacant.

(3) If x is an internal node without any right child, its departure will affect all its downstream nodes. In this case, the sub-tree rooted from its left child also departs the original session and forms a new session in the system.

The following theorem shows the service recovery delay caused by departure operations.

Theorem 4: The service recovery delay caused by departure operations is in $O(H)$ hops.

Proof: Consider the above 3 cases of node departure:

In case (1), the node departure overhead is zero;

In case (2), service recovery needs 1 hop to promote a right child to its parent's position;

In case (3), the sub-tree rooted from x's left child is affected. The maximum service recovery delay equals to the hops for the media server delivers stream data to the furthest node in the sub-tree, which is the height of the sub-tree.

So in the worst case, the service recovery delay is $O(H)$ hops. ■

According to the theorem, the service recovery delay of departure operations is $O(H)$. However, the probability that case (3) happens is quite small. In most condition, the service recovery delay is 0 or 1, which is favorable to VCR-like operations. The influence of case (3) can be eliminated by introducing a "graceful" departure: x keeps providing service until its left child establishes connection to the media server, which will not affect other downstream nodes in the sub-tree.

V. VCR-Like Operations

This section introduces providing VCR-like operations in the DTStream system. A session discovery service is used for fast resource location, based on which VCR-like operations are discussed.

A. Session Discovery Service

Session discovery service (SDS) is used to locate sessions with the requested media files and their playback positions. When a peer generates a request, it locates the proper session using SDS and then joins the session. SDS is constructed based on the SessionCircle using DHT technique. Notice that each node in the SessionCircle is the root of a session.

We use Chord [15] as a representative DHT protocol in DTStream. DHT implements fast discovery for static data object, while in this scenario we need SDS for dynamic data range: a node can join a session as long as its request point is within the data range of that session. We design *Distributed Index* (DI) on each SessionCircle node to implement SDS. All the DIs together form a global index for a video. Owing to the good properties of DHT, even in large scale, the maintenance of DI is efficient and scalable with balanced load.

The DI construction process is described as follows. A video is hashed as a groupID and mapped to a node, called the Rendezvous Point (RP). To join the SessionCircle, a new node sends a JOIN message to a successor node closer to the RP. Upon receiving the JOIN message, the successor checks whether it is an index node of the groupID. If so, the node just adds the sender into its local index because there already exits a path from the RP to itself. Otherwise, the node sets itself as an index node and initiates its local index.

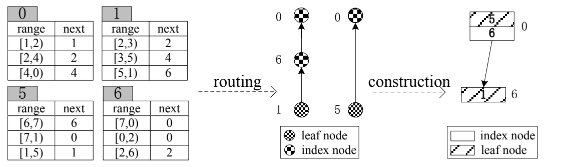

Fig. 4. Construction of distributed index based on Chord DHT

Fig. 4 presents an example of DI construction. Suppose ID space is 8, the hashed groupID of movie f is 7 and position 7 is vacant. Node 0 is the successor of identifier 7 so it is the RP of f. Initially, the SessionCircle only contains node 0 and 6. After that, node 1 and 5 join the SessionCircle in turn. When node 1 comes, it checks its routing table, finding 7 belongs to the interval $[5, 1)$, and then sends JOIN message to the first node, node 6, in $[5, 1)$. Receiving the JOIN message, node 6 sets itself as an index node of groupID 7 and adds node 1 into its local index. When node 5 comes, it finds node 0 in its routing table is just the RP of groupID 7. Then node 5 sends a JOIN message to node 0 and joins as a leaf node.

Assisted with DI, service location is simple. To search a session, a node first contacts an arbitrary index node with the groupID, then the index node performs a breath-first searching. If a session is found to satisfy the join request, it will send notification to the requester.

Theorem 5: It takes at most $O(logN)$ hops to locate the desired session using session discovery service, where N is the number of sessions.

Proof: According to the properties of DHT, the length of the searching path is at most $O(logN)$. Thus searching a session in the P2P streaming system needs $O(logN)$ hops. ■

B. Discussion on VCR-Like Operations

Typical VCR-like operations include random seek, pause, fast forward and rewind. As reported in [16][17], fast forward and rewind occupy less than 1% in the workload while the majority of user interactions are random seeks (48%) and pauses (51%). Most VCR-like operations can be implemented by the jump process: the combination of leaving the current position and then re-joining with a new playback position. A random seek jumps only once, and a fast forward or rewind generally consists of a series of jump process [14]. Thus, we focus on the jump process in this section.

When a VCR-like operation emerges on a node x, x generates a jump request to the root node. When a root node r receives a jump request from its descendant, it handles the request according to the following cases: (1) jump in the local session; (2) jump to another session; (3) jump nowhere. In the first case, the root node checks its local LOT and finds a new position for x. In the second case, node r searches a new session in the SessionCircle and then x departs and joins the other session. In the third case, since no node can provide service for x, a new session is initiated.

The following theorem discusses the performance of jump process.

Theorem 6: The jump process is in $O(logN)$ hops.

Proof: As introduced in the previous paragraph, the join operation needs $O(2)$ hops, the departure operation needs $O(1)$ hops, and searching in the SessionCircle needs $O(logN)$ hops. Consider the three cases of jump process mentioned above:

In case (1), node x performs a departure and a join operation in the local session, which needs $O(1)$+$O(2)$=$O(3)$ hops;

In case (2), node x performs a departure operation, followed by a search operation, and then a join operation to the new session, which is $O(1)$+$O(logN)$+$O(2)$=$O(logN)$ in total;

In case (3), node x departs the current session, and after a searching failure, initiates a new session, which needs $O(1)$+$O(logN)$=$O(logN)$ hops.

Thus, the jump process is in $O(logN)$ hops. ∎

VI. Performance Evaluation

In this section, we evaluate the performance of DTStream via simulation, and compare it with some existing solutions.

A. Simulation Settings

The simulation is built on top of a topology of 5000 peer nodes based on the transit-stub model generated by GT-ITM [18]. The streaming rate is 256 Kbps (most video stream over the Internet today is encoded in the 200-400 Kbps range [4]) and the length of the media stream, denoted as L, is 60 minutes. The default size of the playback buffer, denoted as B, is 120MB, i.e., each peer can cache 120 second recent stream. The default overlapping ratio α of DT-based scheme is $\frac{1}{2}$. The default number of right child an internal node can supply, denoted as K, is 6. We discuss the impact of different α, B and K in the experiments. The arrival of peers follows the Poisson Process with $\lambda = 5$. We also compare some simulation results with RINDY [10], as the two systems use similar buffer management without setting dedicated storage for pre-fetching.

B. Simulation Results

1) ***Server Stress:*** Server stress is measured by the number of sessions, i.e., the number of streams supported by the media server. We discuss the impact of server stress for varying B and K, by letting 5000 nodes join the system. As is shown in Fig. 5, when the buffer size is small, increasing buffer size makes server stress decrease quickly. For example, when B/L ratio increases from 1% to 3%, the server stress drops 39.5% for K=2, 42.2% for K=6 and 33.4% for K=11. However, while B/L ratio exceeds 3%, the server stress trends to stable, increasing buffer size will not improve server stress. The reason is explained as follows. A small buffer size leads to low coverage of media contents in the whole system. At this time, increasing buffer size effectively alleviates server stress. When the coverage of the media contents is dense enough, increasing buffer size is unnecessary. The parameter K also makes impact on server stress. If each node only supplies one right child (K=2), the server stress is much higher than K=6 and 11. Large K can significantly decrease server stress, but imposes heavy workload on internal nodes.

2) ***Control Overhead:*** The control overhead is measured by the number of processed control messages. We compare the control overhead of overlay maintenance in DTStream with that of RINDY, which has 3 types of control messages: *ANNOUNCE*, *FORWARD*, *EXCHANGE*. Fig. 6 shows the average number of control messages per peer under a dynamic environment where each peer performs VCR-like operations with the probability 0.1. We can see that the control overhead of DTStream is much smaller than RINDY: only about 50% of RINDY in the node scale from 200 to 1400 peers. The main reason is that RINDY uses gossip-based protocol to maintain peer membership. Such maintenance entirely relies on message exchanging and leads to higher overhead. In DTStream, peers are grouped by the DT-based overlay to disseminate media streaming. The derivative tree is well-structured and easy to maintain with only a few message exchanging. Furthermore, in DTstream, when a node leaves its current position, it needn't gossip its departure to "remote" peers, as is done in RINDY.

3) ***VCR Impact:*** DTStream can significantly reduce the VCR impact of dynamic node departure due to frequent user interactions, and thus provides smooth playback. We first measure the VCR impact by average number of impacted nodes per VCR-like operation. Here, *impacted nodes* refer to nodes that must relocate media sources when their upstream nodes perform VCR-like operations. Fig. 7 reports the VCR impact of DTStream and RINDY for varying node scale from 500 to 5000 with VCR probability being 0.1. Fig. 8 shows the VCR impact for varying buffer size, B and parameter K of DTStream, and compared with RINDY in the scale of 5000 nodes. From these results, we can see that DTStream is much less influenced by VCR-like operations compared to RINDY. Besides, larger K results in lower VCR impact for the higher probability of mounting right children.

Fig. 9 measures the average server stress increased due to VCR-like operations. It shows that the server stress increased in RINDY is much higher than DTStream. The server stress of RINDY is insensitive to the number of peers. However, server stress increment in DTStream is dropped in a larger system. When the number of peers reaches to 5000, the server stress increment of RINDY is more than 3 time of DTStream.

4) ***Discussion on Overlapping Ratio:*** In derivative tree-based overlay, a node buffer is split into two parts by the

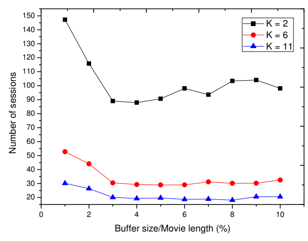

Fig. 5. Server stress for varying B

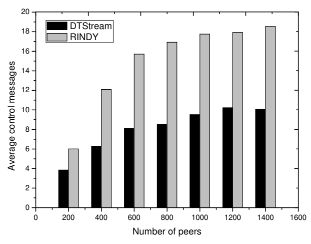

Fig. 6. Control overhead

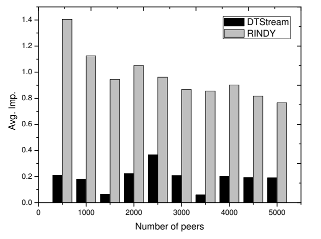

Fig. 7. VCR imp. for varying scale

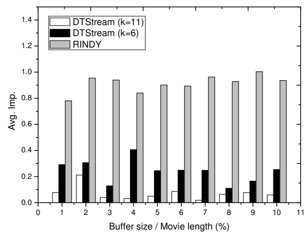

Fig. 8. VCR imp. for varying B

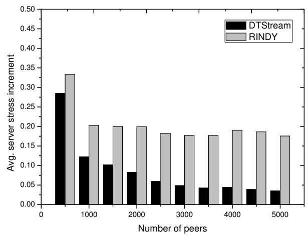

Fig. 9. VCR imp. to server stress

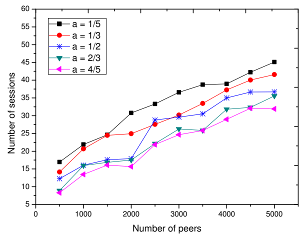

Fig. 10. Server stress for varying α

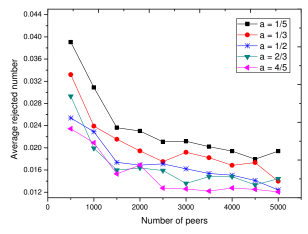

Fig. 11. Reject rate for varying α

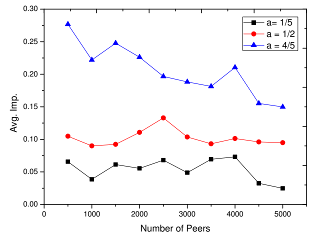

Fig. 12. VCR Imp. for varying α

split line according to the overlapping ratio α. How to select appropriate α is important to the system performance. Fig. 10 and Fig. 11 present the server stress and service reject rate for different α in varying node scale. *Service reject rate* refers to the average number of requests rejected by all sessions due to resource constraints. Fig. 12 shows the VCR impact for different α. We can see that smaller α will decrease VCR impact while imposing higher server stress and service reject rate. The main reason is that a smaller α means a less overlapped buffer, which will increase the proportion of right children in the tree. The more right children, the less VCR impact on these nodes. However, it is also easier for internal nodes to exhaust their uplink bandwidth, which causes higher service reject rate and server stress.

VII. Conclusions and Future Work

In this paper, we propose DTStream, an interactive streaming system to provide continuous media streaming with VCR-like user interactivity in large-scale P2P network. We design a derivative tree-based overlay scheme to organize dynamic and asynchronous peers while bring such advantages as well structured, controllable start-up delay, less memory consumption and quick service construction. By using an efficient session discovery service, we discuss VCR-like operations and their implementation in the P2P streaming system. The cost of VCR-like operations is proved to be in $log(N)$. The efficiency of the proposed scheme is confirmed by extensive simulations.

We plan to improve our work in the following aspects. First, we expect to design a pre-fetching scheme in our system to utilize the playback buffer more efficiently. Second, server complementary streaming [11] can be integrated in our system to shorten startup delay and smooth playback disruption. Third, we consider providing QoS guarantee in DTStream and finally, user behavior should also be considered to better support user interactivity.

Acknowledgment

This work is partially supported by the National High-Tech Research and Development Program of China (863) under Grant No. 2006AA01Z199; the National Natural Science Foundation of China under Grant No. 60803111, 90718031, 60573106, 60721002; and the National Basic Research Program of China (973) under Grant No. 2002CB312002.

References

[1] "PPLive", http://www.pplive.com/.
[2] "Joost", http://www.joost.com/.
[3] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," In *Proc. of ACM SIGCOMM'04*, Aug. 2004.
[4] C. Huang, J. Li, K. W. Ross, "Can Internet Video-on-Demand be Profitable?" In *Proc. of ACM SIGCOMM'07*, Aug. 2007.
[5] Y. Huang, T. Z. J. Fu, D. M. Chiu, J. C. S. Liu, C. Huang, "Challenges, Design and Analysis of a Large-scale P2P-VoD System," In *Proc. of ACM SIGCOMM'08*, Aug. 2008.
[6] X. Zhang, J. Liu, B. Li, and T. Yum, "CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming," In *Proc. of IEEE INFOCOM'05*, Mar. 2005.
[7] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D.Deng, "AnySee: Peer-to-Peer Live Streaming," In *Proc. of IEEE INFOCOM'06*, Apr. 2006.
[8] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," In *IEEE Journal on Selected Areas in Communications*, 22(1):91-106, Jan. 2004.
[9] T. T. Do, K. A. Hua, and M. A. Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," In *Proc. of IEEE ICC'04*, Jun. 2004.
[10] B. Cheng, H. Jin, X. Liao, "Supporting VCR Functions in P2P VoD Services Using Ring-Assisted Overlays," In *Proc. of IEEE ICC'07*, Jun. 2007.
[11] Y. Guo, S. Mathur, K. Ramaswamy, S. Yu, and B. Patel, "PONDER: Performance Aware P2P Video-on-Demand Service," In *Proc. of IEEE GLOBECOM'07*, Nov. 2007.
[12] W. P. K. Yiu, X, Jin, and S. H. G. Chan, "VMesh: Distributed Segment Storage for Peer-to-Peer Interactive Video Streaming," In *IEEE Journal on Selected Areas in Communications*, 25(9):1717-1731, Dec. 2007.
[13] C. Xu, G. M. Muntean, E. Fallon, and A. Hanley, "A Balanced Tree-based Strategy for Unstructured Media Distribution in P2P Networks," In *Proc. of IEEE ICC'08*, Jun. 2008.
[14] D. Wang and J. Liu, "A Dynamic Skip List-based Overlay for On-Demand Media Streaming with VCR Interactions," In *IEEE Transaction on Parallel and Distributed Systems*, 19(4):503-514, Apr. 2008.
[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," In *Proc. of ACM SIGCOMM'01*, Aug. 2001.
[16] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "DISC: Dynamic Interleaved Segment Caching for Interactive Streaming," In *Proc. of IEEE ICDCS'05*, Jun. 2005.
[17] C. Costa, I. Cunha, A. Borges, C. Ramos, M. Rocha, J. Almeida, and B. Ribeiro-Neto, "Analyzing Client Interactivity in Streaming Media," In *Proc. of the International World Wide Web Conference*, May. 2004.
[18] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," In *Proc. of IEEE INFOCOM'96*, Mar. 1996.