Journal of Network and Computer Applications 33 (2010) 353–362

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A vEB-tree-based architecture for interactive video on demand services in peer-to-peer networks

Chung-Nan Lee *, Yung-Cheng Kao [1], Ming-Te Tsai [1]

Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, Republic of China

ARTICLE INFO

Article history:
Received 17 September 2009
Received in revised form
5 February 2010
Accepted 25 March 2010

Keywords:
vEB tree
Peer-to-peer
Interactive operations
Video on demand

ABSTRACT

To provide interactive operations such as random seeking for peer-to-peer on demand video streaming is a challenge. In this paper, a vEB-tree (*van Emde Boas tree*)-based architecture is proposed for interactive VoD services in peer-to-peer networks. The proposed architecture divides a video into many segments that are distributed among participating peers. In this architecture, it includes a vEB-tree-based topology, searching procedures for demand segments, and a segment distribution scheme. It not only efficiently provides interactive operations but also reduces control messages. Additionally, each peer stores segments based on the proposed segment distribution scheme to reduce server stress. Experimental results demonstrate that the proposed architecture outperforms other competing architectures in terms of jump latency, server stress, and cost of maintaining searching topology.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

Nowadays, watching TV on the Internet is becoming more and more popular. Thus many video-on-demand (VoD) architectures have been proposed to allow users to view live streaming in real time, or to download video segments into a diversity of devices for viewing at any time. The traditional VoD service is provided by the client–server architecture that is easy to be implemented, but when the number of users increases, the server would be unable to support the heavy burden. Because of bandwidth limitation of the server, scalability of the whole system is also constrained. The peer-to-peer (P2P) architecture has been proved to be a good solution for this problem.

The P2P network has become an important paradigm to build scalable and distributed application, and the P2P traffic has accounted for a significant part of Internet traffic (Ars technical, online). The major strategy of P2P technique is to use the cumulative bandwidth and computing capability of all network participants, rather than traditional centralized resources, where a relatively low number of servers are available. This method could reduce the dependence on a server, and alleviate the server load; for this reason, one would achieve the objective of scalability. Different from general P2P file-sharing systems (BitTorrent, online; eMule, online; Foxy, online), P2P streaming systems require more stringent resources for real-time video data transmission (Do et al., 2004; Guo et al., 2003; Hefeeda et al., 2003; Liu et al., 2007; Small et al., 2007).

At the same time, when many people watch videos, they may not want to watch them from the beginning to the end. Hence providing friendly interactive actions becomes a necessary trend, such as fast forward, slow forward, jump, pause, etc., but it is really a challenge since providing interactive actions would strongly increase burden on the whole system. In using the P2P architecture, there are more conditions needed to be considered; joining and leaving of peers would influence other related peers; for instance, when a peer leaves, the quality of videos of its children may be affected, and these children have to consider how to seek a new parent. Moreover, how to efficiently respond for interactive actions also needs to be considered. In this paper, a vEB-tree-based (Van Emde Boas et al., 1976) architecture is proposed for supporting interactive VoD services in P2P networks to solve these problems. The major contributions of this paper are as follows: (i) to use the vEB-tree topology to reduce the search cost of interactive operations, (ii) to reduce the maintenance information of peers for the topology construction, and (iii) to balance demand and supply to reduce the server stress.

The remaining part of this paper is organized as follows. The related work is introduced in Section 2. Section 3 describes the proposed architecture. Simulation results are presented in Section 4. Conclusions are drawn in Section 5.

2. Related work

More and more approaches are being proposed to support video on demand services using the P2P transmission architecture.

* Corresponding author. Tel.: +886 07 5254335; fax: +886 07 5254301.
E-mail addresses: cnlee@mail.cse.nsysu.edu.tw (C.-N. Lee), m9034617@student.nsysu.edu.tw (Y.-C. Kao), Chineder@gmail.com (M.-T. Tsai).
[1] Tel.: +886 07 5254335; fax: +886 07 5254301.

1084-8045/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2010.03.021

The random-neighbor topology such as Coolstreaming (Zhang et al., 2005c) and GridMedia (Zhang et al., 2005b) spreads data to randomly chosen neighbors using either "push" or "pull" technique. Although these techniques are robust to dynamic behaviors of peers, peers' bandwidth utilization efficiency is not a global optimum as it is based only on peers' local information. Moreover, the random-neighbor architecture suffers long startup delay and control message overhead among peers. Chunkyspread (Venkatramen et al., 2006) focuses on the simplicity of adopting unstructured networks and the efficiency of topology construction.

A tree-based topology, such as NICE (Banerjee et al., 2002) and ZIGZAG (Tran et al., 2004), uses hierarchical clustering to relieve control overhead. Sioutas et al. (2008) propose a novel decentralized approach for web services (WSs) discovery. Peers that hold WS information are efficiently located by utilizing the balanced distributed tree (BDT) data indexing structure for peer-to-peer networks. P2Cast (Guo et al., 2003) is a tree-like P2P architecture that uses traditional patching schemes. When a new peer joins the system, it would receive the base stream and the patch stream from other peers in the same session. However, the streaming quality monotonically decreases from the root peer to the leaf peers, because the single source approach has some challenges like the parent departure problem and bandwidth limitation. In PROMISE (Hefeeda et al., 2003), peers receive streams from multiple parents, and consider the benefit of each segment from different paths to avoid too many segments routed from a few number of paths. However, it needs to probe the real world network all the time, which is rarely available in practice. Wu et al. (2009) propose a system in which peers receive multiple forward error coding (FEC) packets from different parents to protect delivery content. P2VoD (Do et al., 2004) is used to handle the departure of peers without passing through the source by finding new parents in the upper layer. A peer joining the system is quick by either joining the lowest layer of the tree or creating a new layer in the tree. Outreach (Small et al., 2007) is a tree-like P2P architecture that focuses on maximizing the available peers' upload bandwidth so as to minimize the loading at the server, and consequently maximize scalability. SBO (Liu et al., 2007) employs a strategy for distributing the optimization task among peers. Each joining peer is randomly assigned a color; peers with the same color would do the same task, and each peer is connected only to peers with a different color. In addition, DeMSI (Yim and Buyya, 2006) proposes a peer selection strategy to avoid network congestion and a re-scheduling algorithm to improve smoothness of the aggregated streaming rate experienced by users. However, all the above works do not support the random-seeking operation that is required for an interactive VoD service.

Recently, some works have supported interactive operations in a multimedia steaming system. Comodin (Fortino et al., 2007) provides interactive operations for e-learning and e-entertainment services on the Internet, which enables sharing and cooperative controlling by an explicitly formed group of users for remote media playback. Wong et al. (2007) use two fundamental operations, patching and caching for multicast streaming algorithms, but not all routers support multicast in the real network. PONDER (Guo et al., 2008), according to the interactivity requirements, supports interactive actions by dynamically adjusting the downloading priority. The major strategy employed in BitTorrent is tit-for-tat. This strategy could encourage collaborations to speed up downloading, but the startup delay is still too long. VMesh (Yiu et al., 2007) divides videos into segments, and peers store a few segments of the media data independent of what they are playing. Each peer maintains the list of parent peers that hold the previous segment and next segment based on the DHT (distribution hash table; Zhang et al., 2005a; Stoica et al., 2003; Maymounkov et al., 2002; Rhea et al., 2005; Chawathe et al., 2005). If the jump operation is requested, the list or the DHT search is used to search the parent peer that holds the target segment for viewing the new video position. When users request the jump operations, it needs a lot of search cost to find the peers that can provide the target segment. Therefore, the jump operations are not efficiently supported since the search cost via DHT is $O(\log_2 N)$, where N is the number of peers streaming a particular video in the system and thus the search cost increases with the increasing number of peers. In order to tackle this problem, Jagadish et al. (2005) and Xu et al. (2008) propose balance tree based system architectures. Each peer keeps a list of links to logical parent peers, logical child peers, adjacent peers, and some peers in the same level. Although, the balance tree architecture reduces the search cost, it still introduces long jump latency and large cost of maintaining a lot of neighbors.

In the previous works for supporting interactive operations in P2P networks, if they reduce the jump latency, they must increase the cost of maintaining neighbors significantly. In fact increasing the cost of maintaining neighbors would increase the extra control messages and thus waste network bandwidth. In this paper, the vEB-based architecture is proposed to reduce the jump latency, and not to increase the cost of maintaining neighbors significantly. The main characteristic of a vEB tree to outperform other tree-based P2P architectures is because the search cost of the former is $\log_2 \log_2 S$, which is lower than $\log_2 S$ required by the latter, where S is the number of segments in a video. Additionally, a segment distribution scheme is also designed to balance supply and demand of segments to reduce server stress.

3. Proposed architecture

Most of the existing works on P2P-based media streaming assume that all users who join the streaming session would play the media continuously from the beginning. In fact, most users perform interactive operations frequently, like jump. Users usually review some exciting scenes by jumping backward, or skip boring frames by jumping forward. It is important that a P2P-based architecture can allow users to jump to the target segment efficiently and each peer maintains as less of other peers' information as possible. Therefore the proposed architecture includes vEB-tree-based topology, searching procedures of demand segments, and a segment distribution scheme to make interactive operations more efficient and use peers' upload bandwidth more adequate.

3.1. Overview of the proposed architecture

In the proposed architecture, videos are divided into some smaller segments that are identified by the video ID and the segment ID. Peers store some extra segments at their local storage for serving other peers, and each video segment has multiple copies in the P2P network. Each peer has storage and playback buffers. The advantage of separating storage buffer from playback buffer is that any interactive action of a parent peer does not prevent its children from continuing to receive the parent peer's stored data, because each parent peer does not discard the cached segments even if it jumps to a new playpoint of the video. When a peer wants to play a segment, it firstly searches for the supplying peers that have the required segment in the P2P network, and then sends requests to some of those peers for the service. Those peers with enough bandwidth and ability would serve the requesting peer. In the proposed architecture, an independent topology is constructed for streaming a video. Peers watching the same video would form a network topology, and thus there are many independent topologies for streaming different videos. This approach is reasonable and is widely used, such as in BBTU (Muntean et al., 2008) and VMesh (Yiu et al., 2007), as peers

watching only the same video are willing and responsible to cache segments of this video. As well, in this approach, a video watched by more peers would be more duplicated by peers to balance the relation of supply and demand.

The proposed vEB-tree-based architecture is presented in Fig. 1 and the video is divided into sixteen segments, where a circle with a number X stands for these peers caching the segment X. Each peer in the vEB tree holds "links" to its root, peers with tree relations, and peers with sequence relations. The cost of the search in vEB tree is bounded by $O(\log_2 \log_2 S)$, where S is the number of segments in a video. Apart from the playing buffer used for video streaming, the proposed architecture additionally involves extra local storage to each peer. Videos are divided into uniform segments. For the failure-tolerance purpose, a peer may simultaneously connect to multiple parents that have cached the segment of interest so that they can stream the video in parallel and collaboratively.

In the following, the joining and jumping procedures of a newly arriving peer are described as shown in.

Step one: On joining the system, a new peer randomly connects to some peers and then to find root peers for streaming segment 1 through the help of the vEB tree topology. For example, as shown in , the new peer first randomly connects to peers that cache segment 5. Next, the vEB tree topology would help the new peer find root peers that cache segment 1 for streaming.

Step two: At the same time, the new peer also utilizes its remaining bandwidth to download in advance segment 6 and caches this segment in its local storage. The video segment is not cached randomly; it is determined by the proposed segment distribution scheme. After the video segment is completely downloaded, it will be cached and used to serve others until the peer leaves the system.

Step three: When VCR-like operations occur, for instance requesting segment 8, new parent peers caching the requested segment need to be located; during this procedure, the vEB tree helps the peer to jump to the new playpoint of the video quickly.

Details about how the vEB tree helps the peer to search peers caching the requested segment is described in the following sections.

3.2. Symbol definition

As listed in Table 1, suppose there are N peers and S segments in a video. H_X is the level of peers that stores the segment X (the Xth segment of a video) in the vEB tree. *MaxChildren*(X) is the maximum number of tree-logical children of peers that store segment X; *MAX*(a,b) returns the larger value between a and b, and *Radical* (i,k) returns the kth root of i.

3.3. Overview of the vEB-tree-based topology

In the proposed architecture, a video is divided into S segments, which are distributed among peers over the P2P network. According to extra segments stored by each peer for serving others, we propose a topology that includes tree-logical relation and sequence relation by modifying the vEB tree.

The tree-logical relation is illustrated in Fig. 2. The upper bound number of tree-logical children of a peer and the total number of nodes in the vEB tree in each level are different. The former is calculated by the equation

$$MaxChildren(X) = MAX(Radical(S, 2^{H(X)}), 2) \quad (1)$$

Peers in each level contain links that point to its tree-logical parent and tree-logical children. The upper bound number of tree-logical children of peers in level 1 is equal to $MAX(Radical(S,2^1), 2)$, the upper bound number of tree-logical children of peers in level 2 is equal to $MAX(Radical(S,2^2), 2)$, and so on. Note that the upper bound number of possible tree-logical children (the value of *MaxChildren* (X)) is always larger than or equal to 2. As shown in Fig. 2, the number in a circle denotes the segment number in a video, and a video is divided into 16 segments ($S=16$). Peers that store segment 1 (the 1st segment in a video) located in level 1 of the vEB tree have $MAX(\sqrt{16},2)=4$ tree-logical children. Peers that store segment 2 located in level 2 of the vEB tree have

Table 1
List of notations.

Notation	Description
N	Total number of peers streaming a particular video
S	Number of segments in a video
$H(X)$	Level of peers that store the Xth segment in the vEB tree
Segment i	ith segment of a video
MaxChildren (X)	Returns the maximum number of tree-logical children of peers that store the Xth segment
Radical (i,k)	Returns the kth root of i
MAX(a,b)	Returns the larger value between a and b

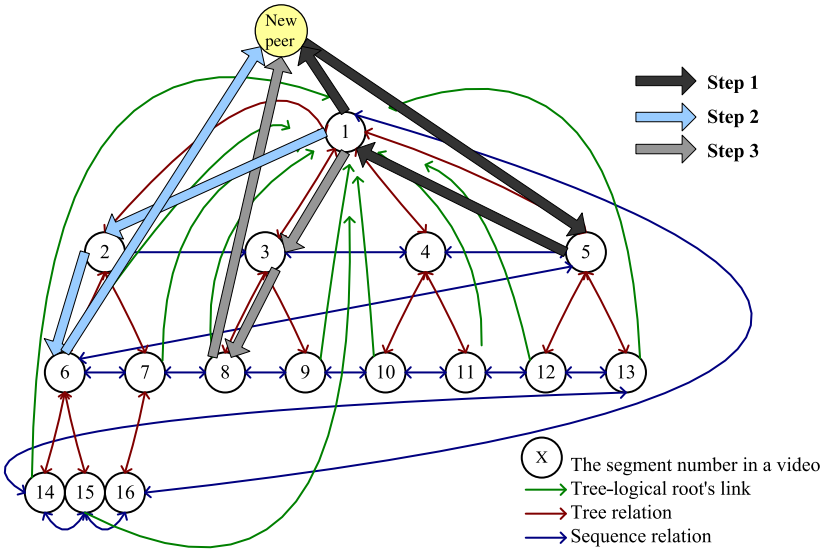

Fig. 1. Overview of the proposed architecture.

$MAX\left(\sqrt{\sqrt{16}},2\right)=2$ tree-logical children. Peers that store segment 3 located in level 2 of the vEB tree have $MAX\left(\sqrt{\sqrt{16}},2\right)=2$ tree-logical children. Peers in level 3 have $MAX\left(\sqrt{\sqrt{\sqrt{16}}},2\right)=2$ tree-logical children, and peers in level 4 have $MAX\left(\sqrt{\sqrt{\sqrt{\sqrt{16}}}},2\right)=2$ tree-logical children. For more efficient searching, each peer also maintains an extra link that points to the root peer of the vEB tree, which is called as the tree-logical root's link as indicated by green lines in Fig. 2.

Because most jump actions occur frequently to locate the closer playpoint, links to peers that hold the previous or next segment would be very useful. For normal playback, the links to peers that hold the next segment are very beneficial. Thus, the second linking relation is the sequence relation. As shown in Fig. 3, each peer has links to peers that hold the previous and the next segment. In addition, peers that hold the last segments and peers that hold the 1st segments have inter-connection.

In the proposed topology, based on the segment number stored by a peer, a peer selects a number of peers to be its neighbors and maintains links to these neighbors. More specifically, as the example shown in Fig. 4, when the video is divided into 16 segments, a peer storing segment 6 maintains links to peers storing the previous segment and the next segment, i.e., segments 5 and 7. In addition, the peer also needs to maintain links to root peer, tree-logical parent, and tree-logical children, i.e., peers storing segments 1, 2, 14, and 15, respectively.

The average number of maintenance neighbors is the number of all relation links divided by the number of segments in a video, which is calculated as follows:

$$\begin{cases} ((S-1)\times 2+S\times 2)/S & \text{if } S\le 5 \\ ((S-1)\times 2+S\times 2+S-5)/S & \text{if } S>5 \end{cases}$$

where $(S-1)\times 2$ is the link number between tree-logical parents and children in the vEB tree, $(S\times 2)$ is the link number of sequence relation among peers, and $(S-5)$ is the link number of peers ($>$ level 2) to the tree-logical root. Note that peers of level 2 do not require the additional link to the tree-logical root since they already have tree-logical parent's link to the root. Fig. 5 presents the average number of neighbors that a peer needs to maintain. It shows the average number of maintenance neighbors by counting all relation links, and then dividing by the number of segments in a video. In the vEB-tree-based topology, if a video is divided into

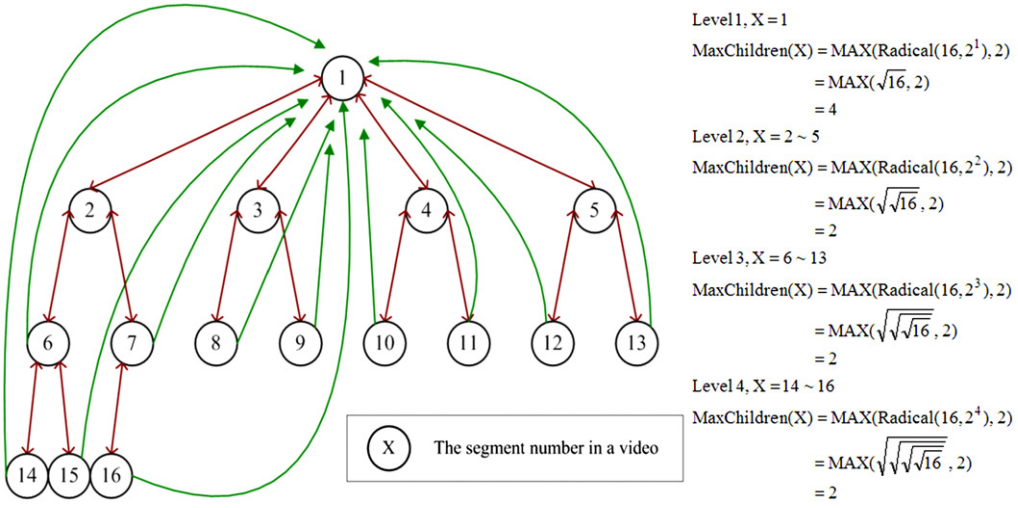

Fig. 2. Tree-logical relation based on the vEB-tree structure with $S=16$.

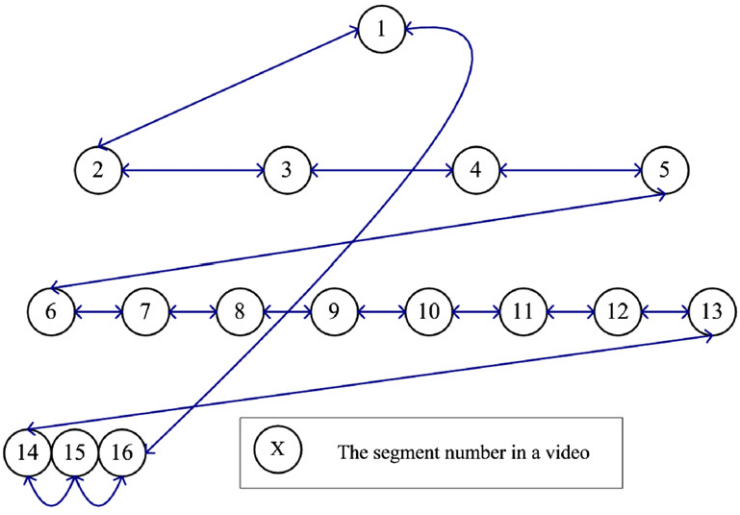

Fig. 3. Sequence relation among peers.

more segments, a peer needs to maintain more neighbors. When the number of segments in a video is 300, the number of maintaining neighbors is almost close to 5.

3.4. Searching procedure of demand segment

In order to search the demand segment promptly, we build the *last visiting segment table*. The *last visiting segment table* records the shortest path between any two segments by modifying the *Floyd algorithm* (Floyd, online). The procedure of building the table is presented in Table 2, where *Distance*[*i*,*j*] denotes the distance from segment *i* to *j*. If segment *i* can connect to segment *j* directly, *Distance*[*i*,*j*] is assigned as 1. The initial value of *Distance*[*i*,*j*] is infinity (∞). *TheLastVisitingSegmentTable*[*i*,*j*] denotes the last visiting segment from segment *i* to *j*.

In fact, the *last visiting segment table* needs to be created only once for each video, and any value in the table is never changed. Each peer would get the table of a video before it prepares to view the video. When a peer needs to search the demand segment, it can use the table to find the shortest path following the procedure in Table 3, where N_{Start} is the number of the segment currently accessed by the requesting peer, and $N_{Destination}$ is the number of the segment demanded by the requesting peer. The objective is to find the shortest path from segment N_{Start} to $N_{Destination}$.

For example, as illustrated in Fig. 6, a video is divided into 16 segments and the last visiting segment table is built as shown in Fig. 7, where the vertical headline in the table represents the start segment number, and the horizontal headline in the table represents the destination segment number. The content in the table represents the last visiting segment number in the shortest path for the given start and destination segment numbers. For example, while a peer wants to jump from segment 4 to 14, it would look for *TheLastVisitingSegmentTable*[4,14]=segment 13, which is the last visiting segment. Then, it would look for *TheLastVisitingSegmentTable*[4,13]=segment 5, which means that while jumping from segment 4 to 13, the last visiting segment is segment 5. Repeating the above operations, in the last step, looking for *TheLastVisitingSegmentTable*[4,5]=4, which means that while jumping from segment 4 to 5, the last visiting segment is segment 4, and it also indicates that segment 4 can directly connect to segment 5. Thus, the result of *TheShortestPath*[] stores 14, 13, 5, and 4. Finally, by inversing the result of *TheShortestPath*[], the shortest path found from segment 4 to 14 is segment 4 → segment 5 → segment 13 → segment 14.

In summary, when a peer caches segment 4 or currently connects to parent peers caching segment 4, it would like to view segment 14. As shown in Fig. 6, it has or can get IP address of parent peers caching segment 5 and thus it first forwards the request to some parent peers caching segment 5. Next, as parent

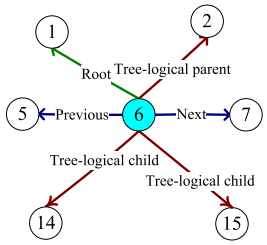

Fig. 4. A peer stores segment 6 and its neighbors in the proposed topology with $S=16$.

Table 2
Procedure for building the last visiting segment table.

Input: *Distance*[*i*,*j*] ($1 \le i \le S$, $1 \le j \le S$)
Output: *TheLastVisitingSegmentTable*[*i*,*j*] ($1 \le i \le S$, $1 \le j \le S$)
Loop *k* from 1 to ***S***
  Loop *i* from 1 to ***S***
    Loop *j* from 1 to ***S***
      IF(*Distance*[*i*,*k*]+*Distance*[*k*,*j*] ≤ *Distance*[*i*,*j*])
        Distance[*i*,*j*]=*Distance*[*i*,*k*]+*Distance*[*k*,*j*]
        TheLastVisitingSegmentTable [*i*,*j*]=segment *k*
      End IF
    EndLoop
  EndLoop
EndLoop

Table 3
Procedure for searching the shortest path.

Input: *TheLastVisitingSegmentTable*[*i*,*j*] ($1 \le i \le S$, $1 \le j \le S$)
  $N_{Destination}$
  N_{Start}
Output: Inverse the result of *TheShortestPath*[] to get the shortest path from segment N_{Start} to $N_{Destination}$
$k=0$
IF($N_{Destination}=N_{Start}+1$ or $N_{Destination}=N_{Start}-1$ or $N_{Destination}=N_{Start}$)
  TheShortestPath[*k*]=$N_{Destination}$
  TheShortestPath[*k*+1]=N_{Start}
ELSE
  TheShortestPath[*k*]=$N_{Destination}$
  WHILE($N_{Destination} \neq N_{Start}$)
    $N_{Destination}$=*TheLastVisitingSegmentTable*[N_{Start}][$N_{Destination}$]
    k=*k*+1
    TheShortestPath[*k*]=$N_{Destination}$
  EndLoop
EndIF

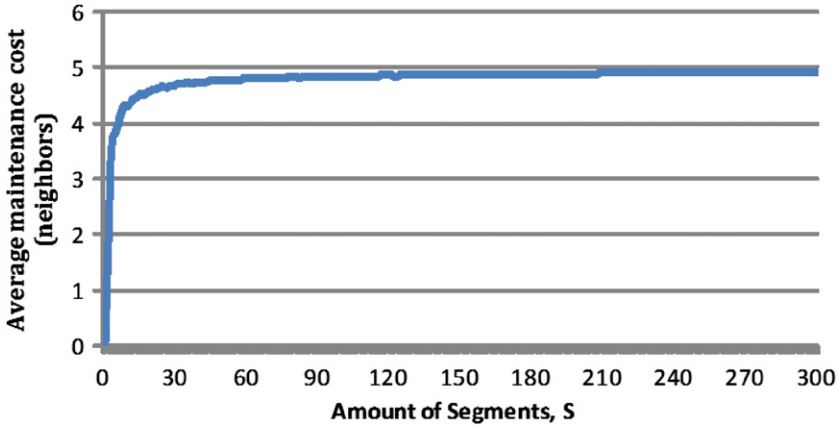

Fig. 5. Average maintenance cost (neighbors) against the number of segments in a video.

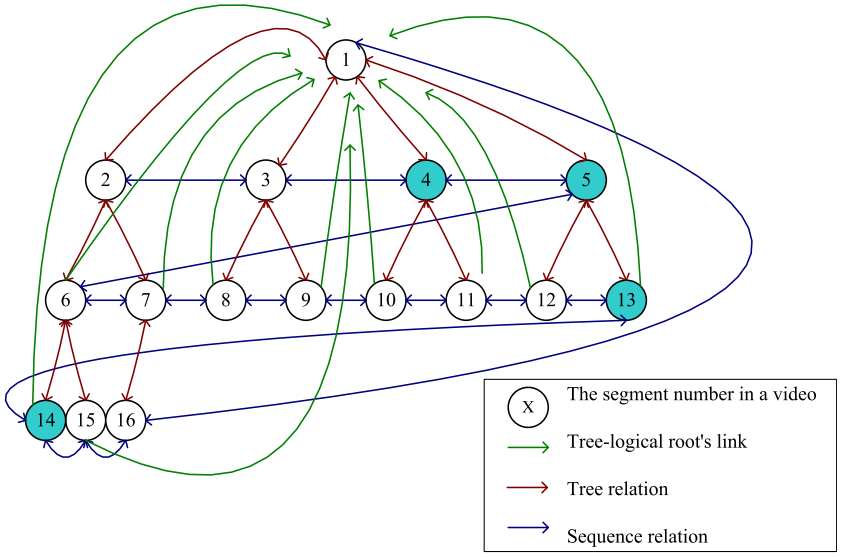

Fig. 6. Proposed topology with $S=16$.

		Destination Segment Number															
		To 1	To 2	To 3	To 4	To 5	To 6	To 7	To 8	To 9	To 10	To 11	To 12	To 13	To 14	To 15	To 16
Origin Segment Number	From 1	1	1	1	1	1	5	16	3	3	4	4	5	5	13	16	1
	From 2	2	2	2	3	6	2	2	7	3	9	4	5	14	6	6	7
	From 3	3	3	3	3	4	2	8	3	3	9	4	11	5	6	16	1
	From 4	4	3	4	4	4	5	16	3	10	4	4	11	5	13	16	1
	From 5	5	6	4	5	5	5	6	7	10	4	12	5	5	13	6	1
	From 6	6	6	5	5	6	6	6	7	8	4	12	5	14	6	6	15
	From 7	7	7	8	1	6	7	7	7	8	9	4	5	14	6	16	7
	From 8	8	7	8	3	1	7	8	8	8	9	10	5	5	6	16	7
	From 9	9	3	9	10	1	5	8	9	9	9	10	11	5	13	16	1
	From 10	10	1	9	10	4	5	16	9	10	10	10	11	12	13	16	1
	From 11	11	1	4	11	12	5	16	3	10	11	11	11	12	13	16	1
	From 12	12	1	1	11	12	5	16	3	3	11	12	12	12	13	16	1
	From 13	13	1	1	5	13	14	16	3	3	11	12	13	13	13	14	1
	From 14	14	6	1	1	13	14	6	7	3	4	4	13	14	14	14	15
	From 15	15	6	1	1	6	15	16	7	3	4	4	5	14	15	15	15
	From 16	16	7	1	1	1	15	16	7	8	4	4	5	5	15	16	16

Fig. 7. Last visiting segment table with $S=16$.

peers caching segment 5 have IP address of parent peers caching segment 13, the former forwards the request to the latter. Then, parent peers caching segment 13 forward the request to the destination peers caching segment 14 for serving the request. Each peer according to stored segment needs to maintain IP address of tree-logical parents, tree-logical children, the root, and peers with sequence relations as well as hold the fixed-size last visiting segment table for the purpose of searching parent peers caching a certain video segment.

3.5. Segment distribution scheme

A peer's storage segment is first determined when the peer joins the system, and is kept until the peer leaves. When a peer needs certain video segment, it has to find peers that can provide the demand segment. If this peer cannot obtain the demand segment from other peers, it can request the demand segment only from the server and thus to increase the server's load. In fact, some more popular segments are accessed more frequently if interactive operations are allowed. Therefore, if peers store segments randomly, some peers may store segments that are requested frequently and have heavier load than others. In contrast, if a balance of demand and supply of each segment can be achieved, it makes the upload bandwidth of peers used more efficient, and reduces server loading. Therefore, a segment distribution scheme is proposed to reduce server loading and make the architecture more scalable. Based on the distribution scheme, the server needs to determine which peer caches which segment for serving other peers.

For each video, the server maintains a request vector $A=[a_1, a_2, a_3, \ldots, a_{S-1}, a_S]$, where a_i and S, respectively, represent the requested number of segment i to the server (initial value is 1) and the total number of segments in a video. In other words, the

server records only those requests not served by peers. When the number of requests for a certain segment to the server increases, it indicates that the supply of the segment is less than the demand, and thus new joining peers should have higher probability to store the segment. We define the probability of a newly joining peer storing segment i, θ_i, as follows:

$$\theta_i = \frac{a_i}{\sum_{k=1}^{S} a_k} \quad (2)$$

VMesh (Yiu et al., 2007) achieves the objective by exchanging segments' information among peers to estimate the supply and demand relation, which requires huge control messages and complicated distribution algorithm. The proposed scheme can balance the supply and demand segments without additional control overhead. This is because the server itself records these requests, which cannot be served by peers, without requiring any additional control messages to finish it.

4. Performance evaluation

4.1. Simulation model

The performance is evaluated using ns-2 (NS-2, online). There is a media server, and each video in the media server is of 2-h length, 0.5 Mbps, and is divided into S segments. The transit-stub model (Zegura et al., 1996), which has a 2-level hierarchical topology, is used in the simulations. The GT-ITM (Zegura et al., 1996) is utilized to generate the core network of 1100 routers, where there is one transit domain with 11 routers. Each transit domain router has 11 stub domains with an average of 9 routers. A video server and 1000 peers are randomly attached to stub-domain routers through a 100 Mbps duplex link. To simulate the effect of cross traffic, each peer builds an additional UDP traffic flow connection to other peers in the simulation. The traffic flow follows an on–off exponential distribution that is assigned a bursty time of 500 ms and an idle time of 500 ms. The packet size is 1024 bytes and the transmission rate is 200 kbps. Each peer is assigned an uploading capacity of 1 Mbps, and maintains two segments, one for playing and the other for serving other peers. Peers' inter-arrival time follows an exponential distribution with a mean of 10 s. The service time of participating peers is exponentially distributed with a mean of 7200 s. The performance of the proposed architecture is compared with Baton (Jagadish et al., 2005), BBTU (Muntean et al., 2008), and VMesh (Yiu et al., 2007). Additionally, VMesh uses the distribution algorithm presented in Mehyar et al. (2007) for segment popularity estimation by exchanging information among peers. In the simulation, VMesh applies popularity-base segment storage schemes to make 50 neighbor peers exchange information with each other for estimating the popularities of segments.

4.2. Popularity model

According to the statistics in Zheng et al. (2005), each user randomly seeks 6–7 times during its session on average. Based on VMesh (Yiu et al., 2007), the popularity distribution of a video is modeled as shown in Fig. 8.

4.3. Jump latency

Jump latency is in terms of hop count or delay time for jumping from the current accessing segment to the new-requested segment. Low jump latency means responding more quickly to user jumping commands.

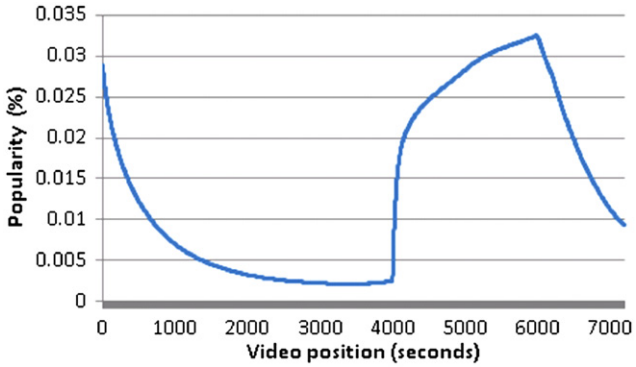

Fig. 8. Popularity model of a video with length = 7200 s.

Table 4
Time complexity of jump latency.

Architecture	Time complexity
Baton	$O(\log_2 S)$
BBTU	$O(\log_2 S)$
VMesh	$O(\log_2 N)$
Proposed architecture	$O(\log_2 \log_2 S)$

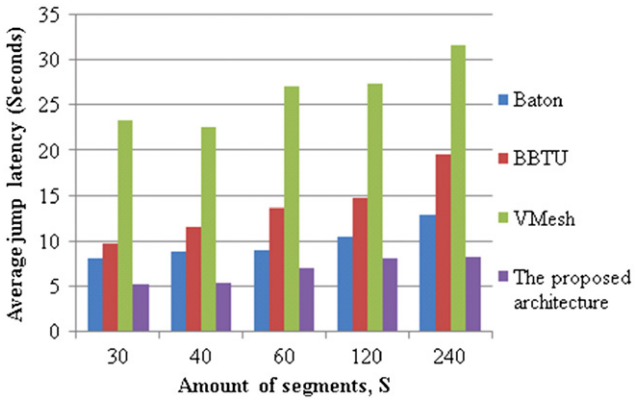

Fig. 9. Jump latency (seconds) against the number of segments in a video with the popularity model.

Comparisons of time complexity of jump latency for the proposed architecture, Baton, BBTU, and VMesh are listed in Table 4, where S is the total segments in a video and N the total number of peers watching the same video. In Baton, peers not only link to parent peers and child peers but also to peers in the same level of the tree; thus, the search cost of Baton is $O(\log_2 S)$. Because BBTU's search method always starts with the root, the search cost is $O(\log_2 S)$ plus one hop for connecting to the root peer. As a part of peers in VMesh use DHT search to find the target segments, the search cost in VMesh is $O(\log_2 N)$. The proposed architecture uses the modified vEB-tree structure, whose search cost is $O(\log_2 \log_2 S)$. Therefore, the proposed architecture is more efficient than competing architectures for jumping operations. The jump latency of the proposed architecture grows up slightly with increasing number of segments in a video.

Fig. 9 indicates the average search time for a jump operation when segments are accessed based on the popularity model in Section 4.2. It is observed that the proposed architecture can reduce the search time by about 22–38%, 45–57%, and 70–77% compared with Baton, BBTU, and VMesh, respectively. Fig. 10

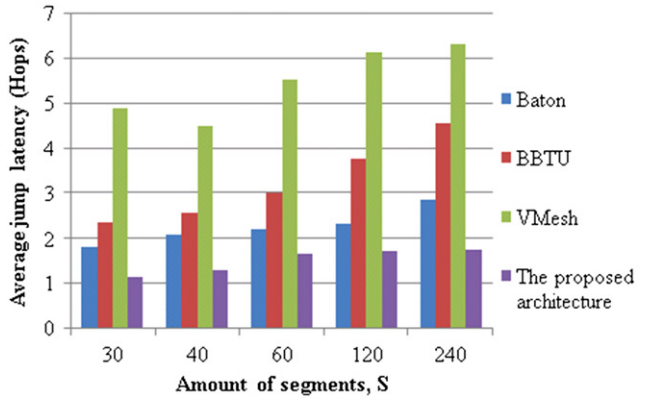

Fig. 10. Jump latency (hops) against the number of segments with the popularity model.

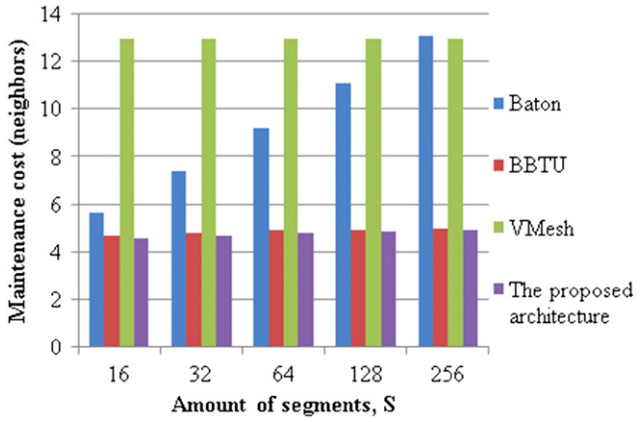

Fig. 11. Average number of maintaining neighbors against the number of segments.

Table 5
Complexity of maintaining neighbors.

Architecture	Maintenance complexity
Baton	$O(\log_2 S)$
BBTU	$O(1)$
VMesh	$O(\log_2 N)$
Proposed architecture	$O(1)$

reveals the average travelled hop number for a jump operation when segments are accessed based on the popularity model in Section 4.2. It demonstrates that the proposed architecture can find out the peers that store the demand segment through about 23–38% fewer hops than Baton, 44–64% fewer than BBTU, and 69–76% fewer than VMesh. Note that although the search cost in VMesh is $O(\log_2 N)$, if a video is divided into more segments, it also increases the search cost. If a video is divided into more segments, the same jumping distance needs to be travelled in more hops.

4.4. Cost of maintaining neighbors

The cost of maintaining neighbors is the number of neighbors needed to be linked by a peer. Clearly, the more the number of maintaining neighbors, the lesser the search cost. However, it requires heavy loading to keep the newest and correct neighbors' information all the time. The proposed architecture can achieve the most efficient search by maintaining the fewest neighbors compared with competing architectures as listed in Table 5.

In Baton, the number of maintaining neighbors grows up as the number of segments in a video increases. There are at most 2^{d-1} nodes in level d (assume root peer in level 1) of a binary tree, and Baton would record peers in the same level based on Jagadish et al. (2005). The maintenance cost of Baton is $O(\log_2 S)$. BBTU's maintenance cost is $O(1)$, a constant, and almost the same as that of the proposed architecture, but the proposed architecture can achieve more efficient searching as described in Section 4.3. In VMesh, each peer not only needs to maintain peers that store the previous segment, the next segment, and the same segment, but also needs to hold a DHT table (Zhang et al., 2005a; Stoica et al., 2003; Maymounkov et al., 2002; Rhea et al., 2005; Chawathe et al., 2005). The DHT table records $O(\log_2 N)$ entries, where N is the total number of peers streaming a particular video.

Fig. 11 shows the search cost under different number of segments in a video. The maintenance cost of Baton grows up with the number of segments. BBTU and the proposed architecture perform best and keep almost close to 5 regardless of the number of segments.

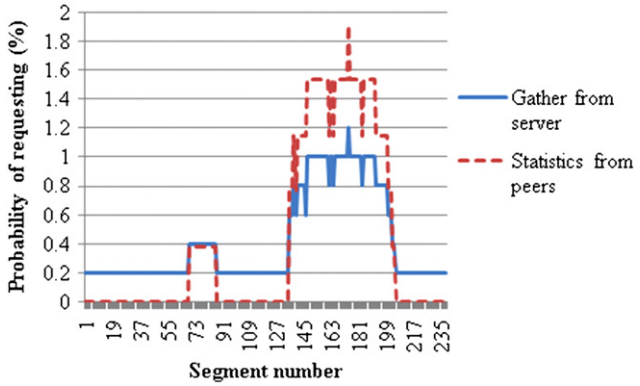

Fig. 12. Probability of requesting each segment when the server receives 500 requests.

4.5. Estimation accuracy of the proposed segment distribution scheme

In this section, the accuracy of the proposed segment distribution scheme is estimated. The probability of requesting each segment is estimated from these requests to the server. Suppose that the length of each segment is 30 s, and a video is divided into 240 segments. Fig. 12 shows the probability of requesting each segment when the server has received 500 requests from peers, the solid and blue line is the probability of requesting each segment estimated from the server, and the dotted and red line is the real probability of requesting each segment by peers. Since these requests to the server indicate that the supply of certain segment is less than the demand, it is necessary to make newly joining peers to cache the segment to eliminate the gap. Figs. 13–15, respectively, show the probability of requesting each segment when the server has received 1000, 1500, and 2000 requests. When the server receives more requests, the requesting probabilities of segments estimated by the server is closer to the real value obtained from all peers. As shown in Fig. 16, when the number of requests to the server increases, the probability of requesting each segment estimated by the server would be more accurate. In fact, when the server receives more than 1000 requests that cannot be served by other peers, it can obtain more than 90% accuracy for estimating the probability of requesting each segment.

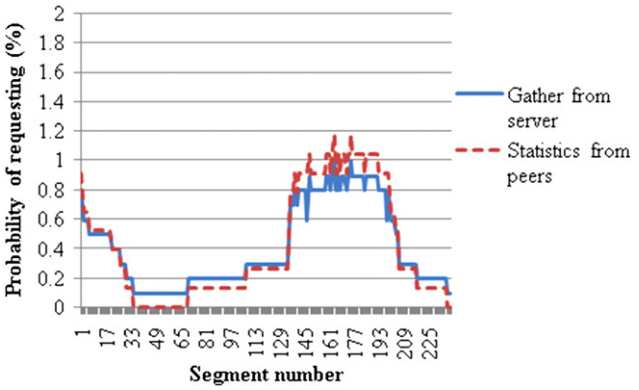

Fig. 13. Probability of requesting each segment when the server receives 1000 requests.

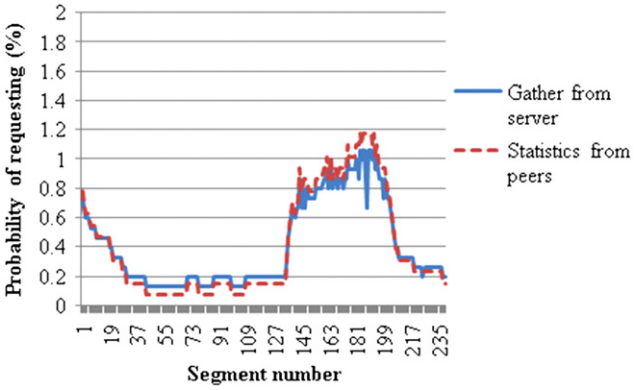

Fig. 14. Probability of requesting each segment when the server receives 1500 requests.

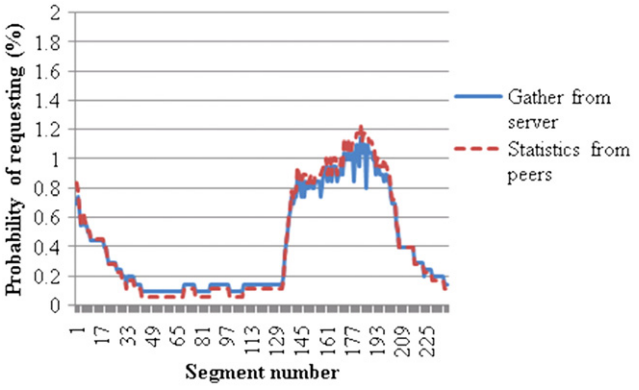

Fig. 15. Probability of requesting each segment when the server receives 2000 requests.

4.6. Server stress

Server stress is evaluated by calculating the number of peers served by the media server. If a peer cannot be served by other peers, it would request the server for the service. Restated, if peers' upload bandwidth can be used more efficiently, it would decrease server stress. The lower the server stress, the more scalable the P2P architecture. Each segment has different popularity. If more popular segments are stored by more peers, it would balance the demand and the supply of segments, and reduce server stress. Thus, how segments are distributed to peers would determine server stress. Fig. 17 presents server stress over simulation time. The proposed architecture can reduce server stress by about 63%, 63%, and 49% compared

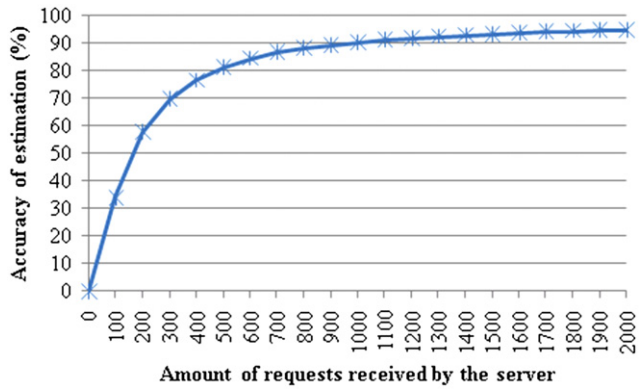

Fig. 16. Accuracy of estimating the probability of requesting each segment by the proposed scheme.

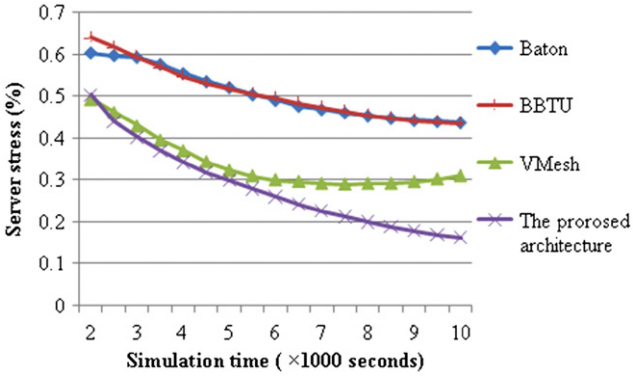

Fig. 17. Server stress over simulation time. The user population grows from 0 to 1000.

with Baton, BBTU, and VMesh, respectively. In Baton, peers randomly store segments. In BBTU, initially, newly joining peers store segments based on the breadth first searching algorithm in the binary tree until all segments have been stored, and then newly joining peers would randomly store segments. Although, in this approach, all segments would be stored in the system, the intrinsic distribution scheme of BBTU is still random. Hence, the server stress of Baton and BBTU are close after 3000 s. In VMesh, server stress is lower than those of BBTU and Baton, but peers in VMesh must connect to 50 neighbors to exchange segments' popularity information, and thus it needs huge cost to determine which peer should store which segment. The proposed segment distribution scheme records only a little information when peers request the server, but this is better for balancing the supply and the demand of segments to reduce server stress.

4.7. Discussion

Simulation results show that the proposed architecture substantially reduces jump latency using the vEB-tree topology to decrease search time for a certain segment. The proposed segment distribution scheme also helps reduce server stress by balancing supply and demand of segments in the P2P network. For these systems, where a peer depends on what is buffered by its parent peers, if parent peers randomly seek another playpoint in the video, the child peer requires to search a new parent again, which interrupts its playback. Managing playing buffer to provide interactive operations needs a quite complex scheme. Therefore, if a P2P system would like to

support interactive operations, using an additional local storage rather than playing buffer would reduce the complexity.

In addition, for a topology, the larger the number of maintaining neighbors, the lesser the search time. However, it is a heavy loading to update the newest and correct neighbors' information all the time. Thus, a good topology must be able to achieve the most efficient search by maintaining the fewest neighbors, which simultaneously has both low search time complexity and low maintenance cost.

How to apply the proposed approach to the existing systems is described as follows. First, each peer must additionally prepare a local storage to cache a segment according to the segment distribution scheme implemented by the server. Next, each peer based on stored segment needs to maintain IP address of tree-logical parents, tree-logical children, the root, and peers with sequence relations as well as to hold the fixed-size last visiting segment table. When a peer would like to view another segment, based on this table, it forwards the request through peers to search parent peers caching the requested segment. The existing systems need to utilize only these rules to build a vEB-tree-based topology to be able to use the proposed approach.

5. Conclusion

We have proposed a vEB-tree-based architecture for supporting interactive VoD services in P2P networks to reduce jump latency, and to decrease the number of maintaining neighbor peers. We also proposed a segment distribution scheme to reduce server stress. Simulation results show that the proposed architecture can achieve less jump latency, less server stress, and less cost for maintaining neighbors compared with Baton, BBTU, and VMesh. In terms of jump latency, the proposed architecture can reduce delay time by about 22–38%, 45–57%, and 70–77% compared with Baton, BBTU, and VMesh, respectively. In terms of server stress, the proposed architecture can reduce server load by about 63%, 63%, and 49% compared with Baton, BBTU, and VMesh, respectively. Therefore, the proposed architecture can have better performance and scalability for supporting interactive VoD operations in a P2P environment.

Acknowledgements

This work was supported by the National Science Council and Ministry of Economic Affairs In Taiwan, Republic of China, under grant NSC 95-2221-E-110-052-MY2 and MOEA 98-EC-17-A-02-S2-0114.

References

Ars technica. [online], 〈http://arstechnica.com/web/news/2009/02/internet-traffic-report- p2p-porn-down-games-and-flash-up.ars〉.
Banerjee S, Bhattacharjee B, Kommareddy C. Scalable application layer multicast. In: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, Pennsylvania, USA. p. 205–17.
BitTorrent. [online];2001. 〈http://www.bittorrent.com/〉.
Chawathe Y, Ramabhadran S, Ratnasamy S, LaMarca A, Shenker S, Hellerstein J. A case study in building layered DHT applications. In: Proceedings of the ACM SIGCOMM conference 2005, Philadelphia, Pennsylvania, USA. p. 97–108.
Do TT, Hua KA, Tantaoui MA. P2VoD: providing fault tolerant video-on-demand streaming in peer-to-peer environment. In: Proceedings of the IEEE international conference on communications 2004, Paris, France, vol. 3. p. 1466–72.
eMule. [online];2002. 〈http://www.emule-project.net/〉.
Floyd. [online];1962. 〈http://en.wikipedia.org/wiki/Floyd_algorithm〉.
Fortino G, Russo W, Mastroianni C, Palau CE, Esteve M. CDN-supported collaborative media streaming control. IEEE Multimedia Magazine 2007;14:6–12.
Foxy. [online];2006. 〈http://download.gofoxy.net/〉.
Guo Y, Suh K, Kurose J, Towsley D. P2Cast: peer-to-peer patching scheme for VoD service. In: Proceedings of the ACM international world wide web Conference 2003, Budapest, Hungary. p. 301–9.
Guo Y, Yu S, Liu H, Mathur S, Ramaswamy K. Supporting VCR operation in a mesh-based P2P VoD system. In: Proceedings of the IEEE consumer communications and networking conference 2008, Las Vegas, Nevada, USA. p. 452–7.
Hefeeda M, Habib A, Botev B, Xu D, Bhargava B. PROMISE: peer-to-peer media streaming using CollectCast. In: Proceedings of the ACM international conference on multimedia 2003, Berkeley, CA, USA. p. 45–54.
Jagadish HV, Ooi BC, Vu QH BATON: A Balanced tree structure for peer-to-peer networks. In: Proceedings of the international conference on very large data bases 2005, Trondheim, Norway. p. 661–72.
Liu Y, Xiao L, Ni LM. Building a scalable bipartite P2P overlay network. IEEE Transaction on Parallel and Distributed Systems 2007;18:1291–6.
Maymounkov P, Mazieres D, Kademlia: A peer-to-peer information system based on the XOR metric. In: Proceedings of the international workshop on peer-to-peer systems 2002, Massachusetts, USA. p. 53–65.
Mehyar M, Spanos D, Pongsajapan J, Low SH, Murray RM. Asynchronous distributed averaging on communication networks. IEEE/ACM Transactions on Networking 2007;15:512–9.
Muntean CX, Fallon GM, Hanley E, ICC A. A Balanced Tree-Based Strategy for unstructured media distribution in P2P networks. In: Proceedings of the IEEE international conference on communications 2008, Beijing, China. p. 1797–801.
NS-2. [online];1989. 〈http://www.isi.edu/nsnam/ns/〉.
Rhea S, Godfrey B, Karp B, Kubiatowicz J, Ratnasamy S, Shenker S, et al. Public DHT service and its uses. In: Proceedings of ACM SIGCOMM conference 2005, Philadelphia, Pennsylvania, USA. p. 73–84.
Sioutas S, Sakkopoulos E, Drossos L, Sirmakessis S. Balanced distributed web service lookup system. Journal of Network and Computer Applications 2008;31:114–49.
Small T, Li B, Liang B. Outreach: peer-to-peer topology construction towards minimized server bandwidth costs. IEEE Journal on Selected Areas in Communication 2007;25:11–35.
Stoica I, Morris R, Nowell L, Karger R, Kaashoek M, Dabek F, et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking 2003;11:16–7.
Tran D, Hua K, Do TA. Peer-to-peer architecture for media streaming. IEEE Journal on Selected Areas in Communications 2004;22:113–21.
Van Emde Boas P, Kaas R, Zijlstra E. Design and implementation of an efficient priority queue. Theory of Computing Systems 1976;10:29–99.
Venkatramen V, Yoshida K, Francis P. Chunkyspread: heterogeneous unstructured tree-based peer-to-peer multicast. In: Proceedings of the IEEE international conference on network protocols 2006, California, USA, p. 2–11.
Wong YW, Lee YB, Li OK, Chan SH. Supporting interactive video-on-demand with adaptive multicast streaming. IEEE Transactions on Circuits and Systems for Video Technology 2007;17:114–29.
Wu PJ, Hwang JN, Lee CN, Gau CC, Kao HH. Eliminating packet loss accumulation in peer-to-peer streaming systems. IEEE Transactions on Circuits and Systems for Video Technology 2009;19:1715–66.
Xu C, Muntean GM, Fallon E, Hanley AA. Balanced tree-based strategy for unstructured media distribution in P2P networks. In: Proceedings of the IEEE international conference on communications 2008, Beijing, China. p. 1797–801.
Yim AKW, Buyya R. Decentralized media streaming infrastructure (DeMSI): an adaptive and high-performance peer-to-peer content delivery network. Journal of Systems Architecture 2006;52:736–7.
Yiu WP, Jin X, Chan SH. VMesh: distributed segment storage for peer-to-peer interactive video streaming. IEEE Journal on Selected Areas in Communication 2007;25:1715–7.
Zegura EW, Calvert KL, Bhattacharjee S. How to model an Internetwork. In: Proceedings of IEEE INFOCOM 1996, San Francisco, CA, USA, vol. 2. p. 594–8.
Zhang H, Goel A, Govindan R. Improving lookup latency in distributed hash table systems using random sampling. IEEE/ACM Transactions on Networking 2005a;13:1114–21.
Zhang M, Tang Y, Zhao L, Luo JG, Yang SQ. GridMedia: a multi-sender based peer-to-peer multicast system for video streaming. In: Proceedings of the IEEE international conference on multimedia and expo 2005b, Amsterdam, Netherlands. p. 614–7.
Zhang X, Liu J, Li B, Yum TP. CoolStreaming/DONet: a data driven overlay network for efficient live media streaming. In: Proceedings of IEEE INFOCOM conference 2005c, Miami, USA. p. 2102–11.
Zheng C, Shen G, Li S. 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 2005, Hilton, Singapore. p. 29–38.