A Balanced Tree-based Strategy for Unstructured Media Distribution in P2P Networks

Changqiao Xu[1, 2, 3]
[1]Software Research Centre
Athlone Institute of Technology
Athlone, Ireland
[2]Institute of Software
[3]Graduate University
Chinese Academy of Sciences
Beijing, China

Gabriel-Miro Muntean
Performance Engineering Laboratory
School of Electronic Engineering
RINCE, Dublin City University
Dublin, Ireland

Enda Fallon, Austin Hanley
Software Research Centre
Athlone Institute of Technology
Athlone, Ireland

Abstract—**Most research on P2P multimedia streaming assumes that users access video content sequentially and passively. Unlike P2P live streaming in which the peers start playback from the current point of streaming when they join the streaming session, in P2P video-on-demand streaming VCR-like operations such as forward, backward, and random-seek have to be supported. Providing this level of interactive streaming service in a P2P environment is a significant challenge. This paper proposes a Balanced Binary Tree-based strategy for Unstructured video-on-demand distribution in P2P networks (BBTU). BBTU assumes videos can be divided into several segments which can be fetched from different peers. BBTU involves two steps: 1) balance binary tree construction based on a prefetching algorithm in order to support interactivity; 2) unstructured video dissemination over network based on gossip protocol, which is the overlay for video distribution. Analysis and simulation show how BBTU is an efficient interactive streaming solution in P2P environment.**

Keywords: P2P video-on-demand; interactivity; balanced binary tree; unstructured network

I. INTRODUCTION

Peer-to-peer (P2P)-based approaches for multimedia streaming services have been studied extensively recently [1-12]. The research work on P2P streaming can be classified into P2P live [1-3] and P2P Video-on-Demand (VoD) [4-7]. In most of these research papers, the authors assume that the peers start playback from the beginning or the current point of streaming when they join the streaming session, and the peers will keep on watching until they leave or fail the session. This supposition ignores the users' interactivity.

Based on a large number of observations on true VoD [13, 14] it was concluded that: 1) most multimedia objects are visited partially; 2) Many VCR-like operations such as forward, backward, random-seek occur, as they are very popular when people watch multimedia programs. Providing this level of interactive streaming service in a P2P environment is a significant challenge. In P2P live streaming, all of the peers have a "close" playback time; a peer can find its streaming suppliers easily when it joins the streaming session. However, P2P video-on-demand streaming allows the peers to request streaming asynchronously with different offsets, so it is difficult to find suitable streaming suppliers. Although media files can be downloaded in advance by P2P file technology such as BitTorrent and played it on-demand afterwards, this introduces long startup delays for playback.

"Cache-and-relay" mechanism [8] is used in P2P streaming which caches the played content and relays it to other peers which request the same streaming later. Furthermore, the mechanism is extended by caching only the played portion to prefetch some portions in the future using additional bandwidth (besides storage) [4, 9]. The prefetching strategy can grant peers the ability to overcome the bursty packet loss, the departure of source peer, and to smoothen the playing experience.

Most research use prefetching strategy to assist solving the interactivity problem for multimedia streaming. Researchers proposed a hierarchical prefetching scheme for popular or sub-popular segments in [5], which is based on the assumption that the popular or sub-popular segments will be visited often with the random-seek. However, this scheme's main challenge is collecting the user viewing logs in a distributed P2P system. In VMesh [6], video segments are prefetched and stored in nodes over a Chord network. Each node maintains previous/next-segment-lists based on distributed hash tables (DHT) and uses the lists to support short jumps. If the jump is too far away, a DHT search is triggered for the segment corresponding to the new position. VMesh does not use "cache-and-relay" mechanism, so the video segments in playback buffer are discarded and not made full use of. DHT is efficient for exact queries but is not well suited for range queries since hashing destroys the ordering of data.

This paper proposes a Balanced Binary Tree-based strategy for Unstructured media distribution in P2P networks (BBTU) along with novel algorithms to address VoD VCR operations-related issues and to increase their efficiency. BBTU uses a balanced binary tree structure of additional prefetching buffers at the nodes. Video segments are prefetched and stored in the nodes' prefetching buffers along the tree in a distributed manner, which will support interactivity. Furthermore, by using the buffer overlapping mechanism and gossip protocol, an unstructured network is established to distribute streaming between nodes. The paper presents and discusses BBTU's algorithms and evaluates the performance of the proposed solution.

This work was supported by National Basic Research Program of China (973) under grant No.2007CB307100, 2007CB307106

II. BBTU SOLUTION

BBTU has a hybrid architecture (see figure 1), which integrates two networks: a balanced binary tree and an unstructured network. The balanced binary tree is based on BATON [10] which is balanced and adaptive to dynamic network. Each node in BATON maintains "links" to its parent, children, and adjacent nodes (in-order traversal). The cost of the search in BATON is bounded by $O(\log N)$.

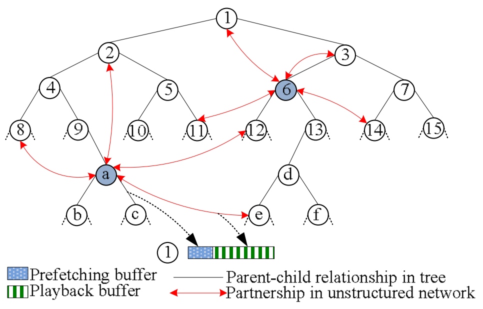
Prefetching buffer
Playback buffer
Parent-child relationship in tree
Partnership in unstructured network

Fig. 1 The architecture of BBTU

Apart from the playback buffer used during video streaming, BBTU involves the addition of an extra prefetching buffer to each node. Videos are divided into uniform segments. Upon entering the system, a new client may start viewing on demand a video stream. Using its residual bandwidth, it also downloads in advance some video segments and stores them in its prefetching buffer. The video segments are not selected at random; they are chosen by the prefetching scheme, which orders the video segments in the adjacent nodes' prefetching buffers according to the playback time along the tree. After the video segments are completely downloaded, they will be maintained during the node's lifetime.

In BBTU, the unstructured network is the routine overlay for video distribution, which is established based on buffer overlapping mechanism and gossip protocol. Each node has a data structure, called Gossip Partners Link Table (GPLT). Each item of node X's GPLT points to the node whose playback buffer overlaps with X's playback buffer. Each node maintains and updates its GPLT independently.

Under normal conditions, each node gets the multimedia streaming from its gossip partners. When VCR-like operations occur, its original gossip partners can not supply the streaming anymore for the playback position being changed, so the new partners need to be searched, during its course, the balanced binary tree assists the node to jump to the new playing scene quickly. The adjacent nodes in the tree will provide the node with the multimedia streaming segments from their prefetching buffers. After re-establishing new partners, the streaming suppliers are switched to the new partners again.

BBTU has the following main features: firstly, it integrates balanced binary tree's high search efficiency merit which decreases the delay for the jump operations; secondly, it inherits good reliability from the random gossip protocol in unstructured overlays and thirdly, the video segments in prefetching buffer are relatively stable in nodes' lifetime, which achieves the high success rate for random-seek and smoothen the multimedia streaming playing experience.

A. Prefetching Scheme

TABLE I
NOTATIONS USED IN THE PREFETCHING ALGORITHM

Notations	Descriptions
S	The VoD source media server
T	The balanced binary tree
P	The requested video for playing
Len	Length of P (the total seconds time for playback)
R	The root node of T
X	Node of T
d	Size of prefetching unit (the number of segments)
$Level(X)$	The level of X in T
$Prebuf(X)$	Node X's prefetching buffer
$Preunit(X)$	The sequence number of prefetching unit in $Prebuf(X)$
$Parent(X)$	Node X's parent node in the T

As the table I shows, P is the requested video stream. It is supposed P is encoded at a CBR rate and divided into equal segments (one segment is 1 sec. for playback). If d successive segments are set as one prefetching unit, then P has $L = \lceil Len/d \rceil$ prefetching units which are numbered from 1 to L sequentially. The prefetching buffer of node X is defined as $Prebuf(X)$ with size of d video segments. In the tree, a range of values managed by a node is greater than ranges of values managed by its left child while less than the range of values managed by the right child.

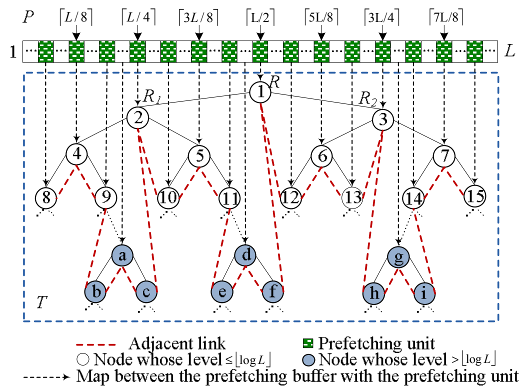
P
⌈L/8⌉
⌈L/4⌉
⌈3L/8⌉
⌈L/2⌉
⌈5L/8⌉
⌈3L/4⌉
⌈7L/8⌉
1
L
R
R1
R2
T
Adjacent link
Prefetching unit
Node whose level ≤ ⌊log L⌋
Node whose level > ⌊log L⌋
Map between the prefetching buffer with the prefetching unit

Fig. 2 Prefetching scheme of BBTU

The proposed prefetching scheme is illustrated in figure 2. As the figure shows, at level 0 of T, the mapping relationship between the prefetching unit corresponding to P's middle position is set to $Prebuf(R)$, namely $Preunit(R) = \lceil L/2 \rceil$. At level 1, presuming the left child and right child of R are named R_1 and R_2, P is divided into two equal subsections: L_1 and L_2. Assuming $Mid(L_1)$, $Mid(L_2)$ are the prefetching unit sequence numbers corresponding to the middle position of L_1 and L_2 respectively, then $Preunit(R_1)=Mid(L_1)$ and $Preunit(R_2)= Mid(L_2)$ are set. At level 2 of T, L_1, L_2 are divided into two equal subsections L_{11}, L_{12} and L_{21}, L_{22} respectively; the mapping relationships between the prefetching unit corresponding to middle position of L_{11}, L_{12} and L_{21}, L_{22} are set with the left and right node's prefetching buffer of R_1 and R_2 respectively. The above operations are repeated for each of the tree levels until P cannot be divided anymore (the subsection length is less than one prefetching unit); then the prefetching unit's sequence number is set to equal its parent node's. For node *X*, presuming *Level*(*R*)=0, from the settings and notations presented above, the following equations can be deduced:

(1)If *Level*(*X*) =0, then

$$Preunit(X) = \lceil Len/2d \rceil \tag{1}$$

(2) If $0 < Level(X) \le \lfloor \log(Len/d) \rfloor$, then there are two cases:

① If *X* is left child, then

$$Preunit(X) = Preunit(Parent(X)) - \left\lceil \frac{Len}{2^{Level(X)+1}d} \right\rceil \tag{2}$$

② If *X* is right child, then

$$Preunit(X) = Preunit(Parent(X)) + \left\lceil \frac{Len}{2^{Level(X)+1}d} \right\rceil \tag{3}$$

(3) If $Level(X) > \lfloor \log(Len/d) \rfloor$, then

$$Preunit(X) = Preunit(Parent(X)) \tag{4}$$

For conditions (1), (2) and (3), *X* needs prefetching of the specific prefetching unit from *S*; for condition (4), *X* gets the prefetching unit from its *Parent*(*X*) or one of its adjacent nodes whose prefetching unit sequence number equals *Preunit*(*X*). (4) avoids to download all the prefetching units from *S*, alleviating the load of *S*. (4) also keeps a list of peers who store the same prefetching unit, which achieves load balancing.

BATON is adaptive to dynamic network, for node join, departure and failure. If a node's position in *T* is changed, it should check whether the prefetching unit in its prefetching buffer satisfies the equations (1)-(4); if not it should update the prefetching unit.

B. Unstructured Video Dissemination Network

In P2P environment, for any node *X* and *Y*, assuming they request the same streaming, the video segments aggregated in *X* and *Y*'s playback buffer are defined as *C*(*X*), *C*(*Y*) respectively, if $C(X) \cap C(Y) \ne \phi$. This means they are "close" enough in playback time, their playback buffer overlaps, so node *X* can supply the streaming service to *Y*. Consequently *Y* does not need to get the streaming from *S*, which can reduce the load of *S*. Note that as long as no VCR operations occur and the network bandwidth is sufficient, this buffer overlapping relationship is unchanged. The buffer is divided into three parts at every node *X*: the prefetching buffer which has been discussed above, receiving buffer named *Recvbuf*(*X*) with size of rb_x which gets streaming from other nodes, and sending buffer named *Sendbuf*(*X*) with size of sb_x which caches the played segments and supplies them to other nodes upon request. The receiving buffer and sending buffer form the playback buffer.

Figure 3 depicts a snapshot of the buffers at nodes *X* and *Y* respectively at time *t*. Supposing t_x and t_y are the currently played segments for nodes *X* and *Y* respectively. The minimum serial number of the video segment in *Sendbuf*(*X*) is t_x-sb_x and the maximum serial number in *Recvbuf*(*X*) is t_x+rb_x. A similar situation it can be deduced for node *Y*.

Assuming *C*(Sb_x), *C*(Rb_x), *C*(Rb_y) are the video segments set in *Sendbuf*(*X*), *Recvbuf*(*X*), *Recvbuf*(*Y*), if $(C(Sb_x) \cup C(Rb_x)) \cap C(Rb_y) \ne \phi$ then *X* can be the streaming supplier for node *Y* and satisfies equation (5):

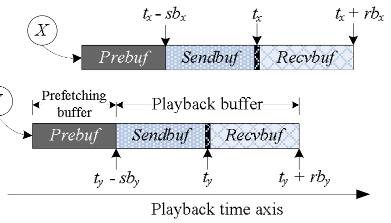

Fig. 3 Node Buffer status at time *t*

$$\begin{cases} t_x - sb_x < t_y + rb_y \\ t_x + rb_x > t_y \end{cases} \tag{5}$$

Similarly *Y* can be a streaming supplier of *X* if it satisfies equation (6):

$$\begin{cases} t_y - sb_y < t_x + rb_x \\ t_y + rb_y > t_x \end{cases} \tag{6}$$

Combining (5) and (6), equation (7) is obtained:

$$t_x - rb_y < t_y < t_x + rb_x \tag{7}$$

Supposing *X* and *Y* join the system at time $t_{x\text{-}join}$, $t_{y\text{-}join}$ respectively, with $t_{x\text{-}join}=t-t_x$, $t_{y\text{-}join}=t-t_y$, (7) results:

$$t_{x-join} - rb_x < t_{y-join} < t_{x-join} + rb_y \tag{8}$$

It is presumed above that the node joins the system at the beginning of the streaming. If node *X* joins with a playback offset o_i, it can be considered that node *X* joins the system on the virtual time $t_{x\text{-}join} - o_i$. *S* records *X*'s joining time; if it joins with a playback offset then the actual joining time becomes a virtual joining time and $t_{x\text{-}join}$ is replaced with $t_{x\text{-}join} - o_i$. *S* sorts all of the nodes by their joining time in ascending order and links all the active nodes into a timing list named QJtime.

Gossip protocols [1, 15] enable random data dissemination with no support from a regular overlay structure. In a typical gossip process, a node randomly selects a subset of target nodes to deliver recently available data segments, and meanwhile, receives segments pushed from these nodes. The random choice of targets achieves resilience to random failures and enables decentralized operations.

Supposing the size of GPLT is *k*, the partners search algorithm for node *X* is as follows:

1) Traverse the timing list QJtime, starting from the node *i* whose joining time is greater than $t_{x\text{-}join}$ - rb_x;

2) Test condition (8) for each encountered node. If the node satisfies (8) and it is not the gossip partner of *X*, then it is added into a set N;

3) If number of nodes in N is more than *K* (invariable parameter) or the visited node violates condition (8), then traversing stops;

4) *k* nodes are selected randomly from N and their IP address and port information is added into *X*'s GPLT.

When GPLT has been established, *X* should send message to its partners periodically, monitoring whether its partners are in normal status. When it discovers any partners such as *Y* leaving, failing accidentally or not satisfying (8), *X* deletes *Y* from its GPLT and triggers partner search algorithm to get a new node to replace *Y*.

When the unstructured media distribution network has been established, Push or Pull [12] mechanisms (not detailed here) can be used to transfer multimedia between nodes.

C. Interactivity Supporting Procedure

When interactivity occurs for node *X* during its playback, the target video segment corresponding to the new position can be calculated roughly by the player interface. Assuming it is *g*, the corresponding prefetching unit is $\lceil g/d \rceil$, which is denoted with *M*. *X* maintains a queue called Que with size of *K* (invariable parameter), each member in Que keeps a pointer (i.e., the IP address) pointing to a node. The interactivity supporting procedure is as follows:

1) *X* empties GPLT, partners updating progress is triggered;

2) *T* is traversed begin with *R*, *M* is compared with *Preunit*(*R*), if equals, then *R* is the target node, *R* is regarded as node *J*, jumps to 4), otherwise, executes next step;

3) *T* is traversed until node *J* with *Preunit*(*J*)=*M*. If *J* cannot be found until node *H* with $Level(H) > \lfloor \log(Len/d) \rfloor$ is visited, failure is returned;

4) Assuming *Ln* is the tail of Que, *Cn* is the current visited node, *J* is put into Que and the right adjacent link of *J* begins to be traversed. If *Preunit*(*Cn*) = *Preunit*(*Ln*)+1, *Cn* is put into Que. If *Preunit*(*Cn*) = *Preunit*(*Ln*) and the present usable bandwidth of *Cn* is more than *Ln*'s, then *Ln* is replaced by *Cn*. If *K*' (default value is 5) numbers of successive visited nodes have the same prefetching unit, then *Cn* switches to *R*, searches from *R* to locate *Y* with *Preunit* (*Y*)= *Preunit*(*Ln*)+1, then continues to traverse *Y*'s right adjacent node. The traversing progress repeats until finding the node whose prefetching unit equals *Preunit*(*Ln*)+1 fails or Que is full;

5) The head of Que begins to send its video segments in its prefetching buffer beginning with the segment whose serial number is *g*. The next nodes in Que send all segments in theirs prefetching buffer to *X* in order. This streaming transmission sequence is of the continuous video segments, beginning with *g* for the new position. The node which finishes sending its prefetching video segments is deleted from Que;

6) Assuming the time for *X*'s jump is $t_{jump,}$ then *X*'s joining time in QJtime is replaced by t_{jump} - *g* and QJtime reorders.

7) When *X* finishes re-establishing its new partners, Que is notified to stop sending streaming. Then, the streaming suppliers are switched to *X*'s new partners. Otherwise, if there are only *K*'' (invariable parameter) numbers of nodes in Que, then the right adjacent node of *Ln* is regarded as *J* and executes the step 4) again, then new nodes are put into Que and Que continues to send streaming to *X*, this operation repeats until *X* finishes re-establishing. At worst, if *X* can not re-establish its new partners successfully, Que will supplies *X* with the continuous multimedia streaming, which ensures the success and stability for random-seek.

As figure 4 shows, presuming node 5 jumps during playback, the video segment corresponding to the new target playback position is *g* which can be calculated approximately from the player interface, After searching in *T*, presuming *g* falls into node 6's prefetching buffer. Node 5 empties its partners in its GPLT which includes node 1, 2, e, 12 and

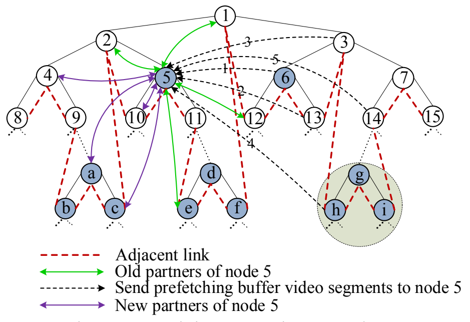

Fig. 4 Interactivity supporting procedure

triggers partners updating process by partners search algorithm. During this process, node 6 and nodes 13, 3, h, 14 (which are the traversal results after executing step 4) send its prefetching buffer's video segments to node 5 until node 5 has established its new partnership nodes (which are 4, a, 10, c). Then the new partner nodes will supply the streaming to node 5.

III. PERFORMANCE EVALUATION

A. Simulation Settings

The source media server has ten videos for streaming; each is encoded at 256 Kbps and has two hours length. A segment length for playback is 1 second; the playback buffer can accommodate 120 segments and is divided into receiving buffer and sending buffer equally. The default size of a prefetching unit is 30 segments. The size of GPLT is 6. The performance of BBTU is evaluated with various parameter settings, and also is compared with VMesh. VMesh is built on top of a public Chord implementation [16]; the length and bit rate of the video and each prefetching unit's length are set same as BBTU's. The topology is generated using the GT-ITM package [17], which consists of ten transit domains, each with twelve transit nodes. Any transit node is then connected to six stub domains, each with nine stub nodes. The total number of nodes is thus 6,600. Assuming routing between two nodes in the network follows the shortest path. The initial bandwidth assigned to the links is as follows: 1.5 Mbps between two stub nodes, 5 Mbps between a stub node and a transit node, and 10 Mbps between two transit nodes. Cross traffic will be injected during the experiments to emulate dynamic network conditions. The peers participating in the system follow a Poisson arrival.

B. Performance and Comparison

Jump Latency. The jump latency is in terms of hop count for locating the node, the segment corresponding to the new position falls into this node. Figure 5 presents results comparison when different parameters are used in which the size of prefetching unit *d* and number of nodes in system are set to different values. This result shows how BBTU is more efficient than VMesh. In VMesh, the cost for search via DHT is *O*(log*N*). *N* is the number of nodes in system and the cost increases with the total number of nodes. Though the cost for search in BATON is *O*(log*N*), taking BBTU's prefetching

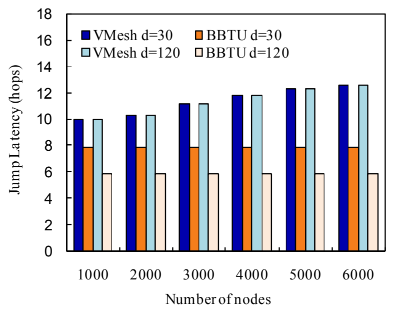

Fig. 5 Jump latency

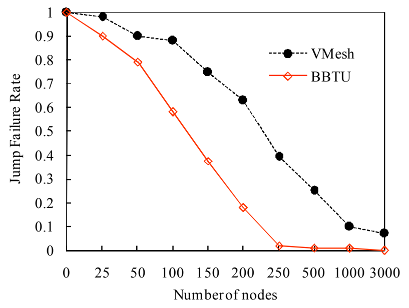

Fig.6 Jump failure

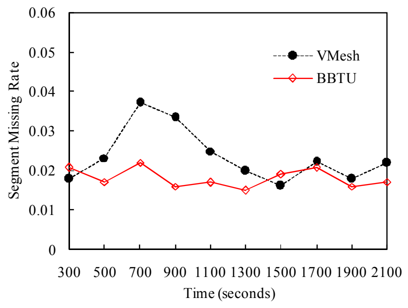

Fig. 7 Average segment missing rate

scheme into account, the hops for searching a target node whose prefetching buffer includes any playing position segments are less than $O(\log\lceil Len/d \rceil)$. The cost decreases with the increasing of prefetching unit size.

Jump Failure. Jump Failure Rate (JFR) is compared between BBTU and VMesh when random-seek occurs. It is considered that a jump fails if a node cannot locate the target node with the first number of segments corresponding to the new position during its jump. JFR is set as the average ratio of the total nodes which fail jumps to the total nodes for requesting jumps during playback. It reflects the capability to support random-seek function. As figure 6 shows, the JFR of the two systems decreases with increasing of number of nodes. However the decrease rate for VMesh is generally lower than that of BBTU, it is obvious when not dealing with large scale system. In VMesh, peers download segments randomly for storage when they join the system, however, in BBTU, peers fetch segments by the prefetching algorithm, which enables distributing the whole segments regularly over nodes. In general, $\lceil Len/d \rceil$ numbers of nodes' prefetching buffers can share the whole video segments of the requested video.

Streaming Quality. Continuous and smooth playback is important for P2P streaming applications. The Segment Missing Rate (SMR) is adopted as the criterion for evaluating streaming quality. A data segment is considered missing if it is not available at a node till the play-out time, and the SMR for the whole system is the average ratio of the missed segments at all the participating nodes during the simulation time. Figure 7 shows SMR in a dynamic network, VMesh fluctuates at a greater rate than BBTU. In VMesh, when a node consumes its current segment, it gets its next segment for playing from a new parent node by searching its next-segment-list or DHT search, so the node's suppliers and route path change frequently with the playback time axis; in BBTU, when a node has established its gossip partners, the partnership is relatively stable, the multiple partners will continually provide the streaming, which achieves better stability than VMesh. From a video decoding point of view, the low and stable SMR achieves higher streaming quality.

IV. CONCLUSION

This paper proposed a novel Balanced Binary Tree Unstructured media distribution strategy for P2P VoD streaming (BBTU). It constructs a balanced binary tree based on a prefetching algorithm and establishes an unstructured video dissemination network based on the buffer overlapping mechanism and gossip protocol. Simulation-based testing and consequent comparison-based result analysis show how BBTU is a highly efficient interactive streaming service solution in P2P environment.

REFERENCES

[1] X. Zhang, J. Liu, and B. Li, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," Proc. IEEE INFOCOM'05, March 2005.
[2] X. Liao, H. Jin, and Y. Liu, "AnySee: Peer-to-Peer Live Streaming," Proc. IEEE INFOCOM'06, April 2006.
[3] K. Sripanidkulchai, A. Ganjam, and B. Maggs, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," Proc. ACM SIGCOMM'04, August 2004.
[4] C. Huang, J. Li, and K. W. Ross, "Can Internet Video-on-Demand be Profitable?" Proc. ACM SIGCOMM'07, August 2007.
[5] C. Zheng, G. Shen, and S. Li, "Distributed prefetching scheme for random seek support in peer-to-peer streaming applications," Proc. ACM workshop on Advances in peer-to-peer multimedia streaming (P2PMMS'05), November 2005.
[6] W.-P. K. Yiu, X. Jin, and S.-H. G. Chan, "Distributed storage to support user interactivity in peer-to-peer video streaming," Proc. IEEE ICC '06, June 2006.
[7] B. Cheng, H. Jin, and X. Liao, "Supporting VCR functions in p2p vod services using ring-assisted overlays," Proc. IEEE ICC'07, June 2007.
[8] S. Jin, A. Bestavros, "A Cache-and-relay streaming media delivery for asynchronous clients," Proc. Int. Workshop on Networked Group Communication (NGC'02). Boston, Massachusetts, USA, October, 2002.
[9] A. Sharma, A. Bestavros, and I. Matta, "dPAM: a distributed prefetching protocol for scalable asynchronous multicast in P2P Systems," Proc. IEEE INFOCOM'05, March 2005.
[10] H. V. Jagadish, B. C. Ooi and Q. H. Vu, "BATON: A balanced tree structure for peer-to-peer networks," Proc. Int. Conference on Very Large Data Bases (VLDB'05), August 2005
[11] Y. Cui, B. Li, K. Nahrstedt, "oStream: asynchronous streaming multicast in application-layer overlay networks," IEEE Journal on Selected Areas in Communications, vol. 22, no. 1, Jan. 2004, pp. 91 - 106.
[12] M. Zhang, J.-G. Luo, and L. Zhao, "A peer-to-peer network for live media streaming using a push-pull approach," the 13th annual ACM int. conference on Multimedia, Hilton, Singapore, 2005.
[13] J. Padhye, J. Kurose, "An empirical study of client interactions with a continuous-media courseware server," The ACM Int. Workshop on Network and Operating Systems Support for Design Audio and Video (NOSSDAV'98), Cambridge, UK, 1998.
[14] M. Chesire, A. Wolman, G. Voelker, "Measurement and analysis of a streaming media workload," Proc. The USENIX Symp. Internet Technologies and Systems, Boston, Massachusetts, 200l.
[15] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie, "From epidemics to distributed computing," IEEE Computer Magazine, 2004.
[16] "The Chord/DHash project." [Online]. Available: http://pdos.csail.mit.edu/chord/#downloads.
[17] E. Zegura, K. Calvert, S. Bhattacharjee, "How to model an internetwork," Proc. IEEE INFOCOMM'96, March 1996.