SkipStream: A Clustered Skip Graph Based On-demand Streaming Scheme over Ubiquitous Environments

Qifeng Yu, Tianyin Xu, Baoliu Ye, Sanglu Lu and Daoxu Chen
State Key Laboratory for Novel Software and Technology
Department of Computer Science and Technology, Nanjing University, China
{yuqifeng, flydutchman}@dislab.nju.edu.cn, {yebl, sanglu, cdx}@nju.edu.cn

Abstract

Providing continuous on-demand streaming services with VCR functionality over ubiquitous environments is challenging due to the stringent QoS requirements of streaming service as well as the dynamic characteristics of both underlying network and user behavior. In this paper, we propose SkipStream, a skip graph based Peer-to-Peer (P2P) on-demand streaming scheme with VCR support to address the above challenges. In the design of SkipStream, we first group users into a set of disjoint clusters in accordance with their playback offset and further organize the resulted clusters into a skip graph based overlay network. In addition, we present a distributed on-demand streaming scheduling mechanism to minimize the impact of VCR operations and balance system load among nodes adaptively. The average search latency of SkipStream is $O(log(N))$ where N is the number of disjoint clusters. We also evaluate the performance of SkipStream via extensive simulations. Experimental results show that SkipStream outperforms early skip list based scheme DSL by reducing the search latency 20%-60% in average case and over 50% in worst case.

1. Introduction

With the increasing improvement of network availability and computing performance, on-demand streaming services with VCR functionality such as Video-on-Demand (VoD) systems over ubiquitous environments become more and more attractive in recent years. On-demand streaming allows users to actively select/play the media stream asynchronously and issue different VCR operations (such as Pause, Fast Forward/Backward, Random Jump, etc.) freely. The asynchrony of user requests as well as the freeness of user behaviors makes it challenging to design and implement such kind of streaming systems over ubiquitous environments. The streaming system has to serve each asynchronous request independently with low startup latency and response VCR requests immediately without affecting the continuity of streaming service. Furthermore, the dynamics and the heterogeneity of ubiquitous environments impose additional difficulty in guaranteeing the QoS of on-demand streaming service. The streaming system must deal with the inconstant change of network topologies, the biased capabilities of heterogeneous devices and the unpredictability of user behaviors.

Many proposals have been presented to address the above mentioned challenges in the past years. Although early proxy-based solutions such as Content Delivery Networks (CDNs) are easy to be deployed over the Internet, the overall throughput of the system is constrained by the available bandwidth on the clustered servers. Thus, this kind of approach still lacks of scalability. Recent years research shows that P2P computing is a low-cost and effective means for aggregating available resource among nodes and supporting cooperative content distribution over the Internet. In addition to requesting for data from some nodes, each node within a P2P system could also serve other nodes with the received data. P2P based on-demand streaming service has attracted great research interests in the last decade[2-6][10-11]. It has been deemed as a promising approach to address the scalability problem. However, the self-organized nature of P2P system makes it difficult to maintain the connectivity of P2P overlay network. Therefore, the reliability becomes one of the key problems to be addressed for any P2P based streaming systems. It is essential to regulate the streaming overlay network effectively so as to provide efficient on-demand streaming service with VCR functionality in the context of ubiquitous environments.

Inspired by the topology characteristics of *skip graph*, in this paper we propose *SkipStream*, a novel P2P based on-demand streaming scheme with VCR support over ubiquitous environments. In the design of SkipStream, we assume each user has a fixed size of buffer to cache the recently received media segments. SkipStream groups users into a

0190-3918/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPP.2009.57

set of disjoint clusters according to their playback offset, and then organizes the resulted clusters into a hierarchical overlay network named *clustered skip graph*. When a new streaming request or a VCR operation is issued, SkipStream tries to redirect the request to a node holding the requested segment by performing the search operation over the clustered skip graph. This enables us to reduce the load on streaming server significantly. We explore the key issues associated with SkipStream and evaluate its performance via simulations. The main contributions of this paper are as follows:

1) We present clustered skip graph (CSG), a novel structure to construct the failure-resilient P2P streaming overlay network and facilitate the on-demand streaming scheduling over ubiquitous environments. By grouping users into a set of disjoint clusters in accordance with their playback offset and further organizing the clusters into a skip graph based overlay network, the proposed clustered skip graph could constrain the average media segment search latency to be $O(log(N))$ where N is the number of disjoint clusters.

2) We present a distributed on-demand streaming scheduling mechanism over the clustered skip graph. With some stream redirection policies, the impact of VCR operations is minimized and load balance among nodes is achieved.

3) We evaluate the performance of SkipStream via extensive simulations. Simulation results show that SkipStream could effectively improve the search efficiency with low control overhead. It outperforms skip list based scheme DSL [9] by reducing the search latency 20%-60% in average case and over 50% in worst case.

The remainder of this paper is organized as follows. Section 2 overviews some related work. Section 3 describes the system architecture of SkipStream. Section 4 presents the details of SkipStream. Section 5 shows the performance evaluation results of SkipStream. Finally, we conclude the work of this paper in Section 6.

2. Related Work

There has been a lot of work on providing on-demand streaming services using P2P technology. Early tree based proposals such as oStream [2], P2Cast [4], DirectStream [5] and CoopStreaming [10] try to build an application layer tree in which parent nodes relay the media segments to their descendants. P2VoD [3] brings the concept of *generation*, where peers in the same generation store the same segment in the prefix of their buffers. However, the single parent approach in these systems doesn't work well in real situations over ubiquitous environments, since a single peer usually cannot serve a client due to the limited bandwidth between them. Although PROMISE [6] could present multiple suppliers for a client, it cannot efficiently support VCR operations. In order to support VCR operations, Mesh-based approaches such as VMesh [11] employ dedicated storage in each node to store video segments. However, it is difficult to make a common consensus on determining the segment size among heterogeneous node over ubiquitous environments.

W. Pugh shows that *skip list* [8] is a well-designed data structure that could be used as a probabilistic alternative of balanced trees. It provides the full functionality of a balanced tree in a distributed system. Wang *et. al* present a skip list based scheme named DSL [9] to support on-demanding streaming services with VCR functionality over P2P network. To control the network dynamics, DSL maintains a global skip list with the playback offset as the indexing key. With multi-layer links, a client can relocate suppliers of expected data segments within the expected time to be logarithmic to the client population. However, skip lists have serious worst-case performance. Besides, The top-layer node has to handle much more control messages. Its failure will split the whole topology into two parts. Thus, it is vulnerable and not failure-resilient. Recently, we witnessed that skip graph [1] was in essence an extension of skip list and was a good candidate for designing fault-tolerant on-demand streaming systems. In addition, it can support queries based on key ordering. However, its fault-tolerant property comes at the cost of letting each node maintains a large number of pointers to others. Moreover, the top-down search routine in skip graph does not make use of the neighboring information efficiently. Motivated by the above observations, we present a novel skip graph based on-demand streaming scheme and investigate the key challenges involved.

3. SkipStream: An Overview

3.1. Preliminary of Skip Graph

Before illustrating SkipStream, we first give a brief introduction about skip graph. A skip graph has a basic linked list which contains all the nodes. The nodes are sorted in the basic linked list by indexing keys. Each node x has a membership vector $m(x)$. In [1], $m(x)$ is taken as a random word over some fixed alphabets. Fig.1 shows a simple skip graph using random bit strings as membership vectors. Let $SPL(m(x), m(y))$ represent the length of same prefix that $m(x)$ and $m(y)$ have. If there has no such node z between x and y in the basic linked list that $SPL(m(x), m(z)) \geq SPL(m(x), m(y))$, then node x and node y are called far-neighbors. There's a link between two far-neighbor nodes. These links build up multiple layers of linked lists in the skip graph. Every linked list is labeled by some finite word W, and a node x is in the list labeled by W if and only if W is a prefix of $m(x)$. The length of word W infers which level the corresponding list stays. Linked lists

in higher levels have longer length of the labeled words.

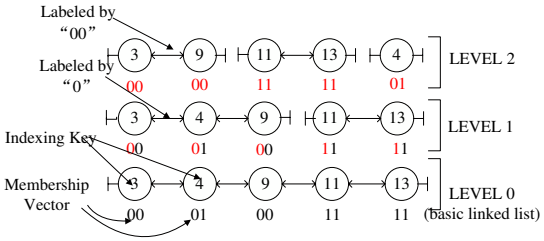

Figure 1. A skip graph for five nodes

A top-down search algorithm is presented in [1]. For a given indexing key, it can find the node whose indexing key equals the given one, if it exists, or the address of the node storing the largest key less than (or the smallest key greater than) the search key is returned. The search operation in the top-down search algorithm is executed from high level linked lists to low level ones. As each node in the skip graph only maintains the links to its far-neighbors, the algorithm is performed in a distributed way. In Fig.1, for node 13 to search node 3 using the top-down routine, the query path is $13 \rightarrow 11 \rightarrow 9 \rightarrow 4 \rightarrow 3$. The levels of links used in the search process are monotonically decreasing. In this case the link information has not been efficiently used, where node 9 can reach node 3 directly.

3.2. System Architecture of SkipStream

In SkipStream system, a media stream is divided into a series of segments. Each segment contains the data content of a fixed time interval (for example, one second). Each client maintains a sliding window to buffer recently received media segments. The media server holds all media files and delivers media segments to a user directly only if the segments the user requested could not be served by other nodes in the system. Users issue service requests asynchronously with different initial playback offset and are allowed to perform VCR operations such as pause, resume, random seek, fast forward/backward freely. The notations used in this paper are summarized in Table 1.

SkipStream employs a novel overlay structure called CSG (clustered skip graph), where users are group into disjoint clusters. Each cluster is made up of users with similar playback point. In this paper we call such a cluster a *virtual node* (VN). All VNs are structured in a variant of skip graph. Each VN has a unique indexing key, which is an integer related to the width of a VN and the start point of the users in it. All VNs have the same width, which limits the length of the users' playback point range in a single VN. The width of each VN is set to less than half of sliding window length of a user, which makes the users in the same

Not.	Description
N	Number of VNs
B	Minimum buffer size of client
n	Number of nodes
m	Number of delegate nodes in a VN
w	Width of VN(s)
$K(X)$	Key of VN X
T	Length of the movie(s)
p_x	Playback position of node x
L	Maximum number of VNs (T/w)
d_x	The deadline of the transmission for node x
$LF(X)$	Closest left far-neighbor of X
$RF(X)$	Closest right far-neighbor of X

Table 1. Notations

VN potential data parents and children. A user's start point equals to its streaming service startup time if it playbacks from the beginning (which can be calculated as the current time minus the playback offset). A user falls into the VN whose key equals the user's start point divided by the width of the VN. For example, if the start point of a user is 6705 (seconds since the first node joins the system) and the width of a virtual node is 20 (seconds), the user belongs to the VN whose key is $6705/20 = 350$. To create links between VNs, the membership vector of a VN is made by the inverse bit string of the key. Let $K(X)$ denotes the key of VN X. If two existing VNs (X and Y) are linked as far-neighbors to each other, and k is length of same prefix that $m(X)$ and $m(Y)$ have, we can get that $(K(X) - K(Y)) \ mod \ 2^k = 0$ and $(K(X) - K(Y)) \ mod \ 2^{k+1} \neq 0$. Let $SBL(a, b)$ be the maximum integer k that $(a - b) \ mod \ 2^k = 0$, (a, b are integers and $a \neq b$). Hence, we can define far-neighbors in CSG as follows:

Definition 1 *Let $K(X) < K(Y)$, X and Y are far-neighbors, if and only if there is no virtual node Z that $K(X) < K(Z) < K(Y)$ and $SBL(K(X), K(Z)) \geq SBL(K(X), K(Y))$.*

Fig.2 shows a simple clustered skip graph based system.

As it's not efficient to have all the nodes maintain the far-neighbor information, users within a VN are divided into two categories, delegate nodes and normal ones. Delegate nodes represent the VN and maintain the information of far-neighbors of the VN, while normal ones have the knowledge of the existence of all the delegate nodes in the VN. A far-neighbor X's information denotes the set of delegate nodes in X. If a delegate node's control overhead reaches a threshold TH or it has bad playback continuity, it picks a node that is not a delegate one in the same VN to replace itself and sends the new set of delegate nodes to all far-neighbors. Each node also maintain the information

(buffered media segment range) of some other nodes in the same VN. A user can get media segment delivered from some of the nodes with smaller start point in the same VN, and may find suppliers in the closest left far-neighbor.

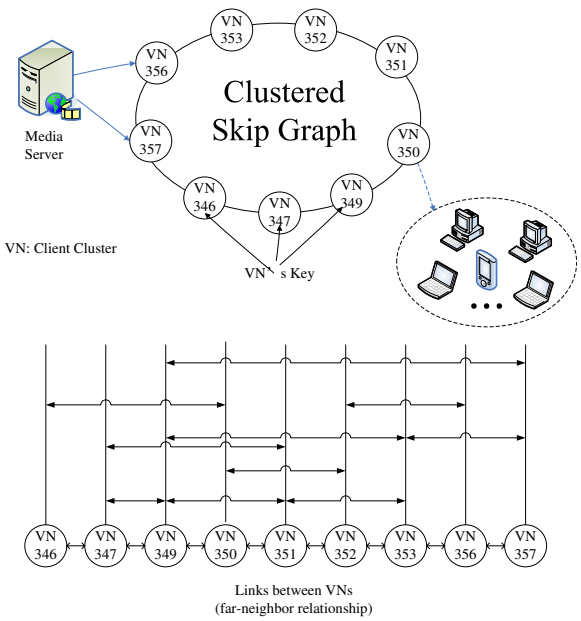

Figure 2. An overview of CSG based streaming system

4. Overlay Construction, Maintenance, Data Scheduling and VCR Operations

In this section we show the details of SkipStream including the construction and maintenance of the CSG overlay, the data scheduling scheme for transmission and the support of VCR operations.

4.1. Basic Operations Over CSG

There are three basic operations for overlay construction and maintenance: VN search, node joining and node departure. A VN search operation is performed to find the VN with the search key, if it exists, or the one with the largest key less than (or the smallest key greater than) the search key. Node joining (departure) operation is to add (remove) a user node to (from) the CSG. All overlay construction and maintenance can be done by these three basic operations.

4.1.1 VN Search Operation

Algorithm 1 gives the pseudo-code of search operation at a delegate node which receives a VN search request. The status information a delegate node maintains is shown in Table 2. In a search operation, every requested delegate node sends the search request to one of its far-neighbors, whose key is closest to the search key, until it gets to the destination. For a user node already in the CSG to start a VN search operation, it can run Algorithm 1 directly if it is a delegate node or it can send a search request to a delegate node in the same VN if it is a normal one. For a new coming client, since the media server records the peers that are getting video segments from it, it can first contact the server, which determines the key of VN that the new client belongs to and issues a VN search request to a node in its client record, who's playing the same video with the closest playback position.

Not.	Description
lflist	Sorted list of left far-neighbors
rflist	Sorted list of right far-neighbor
uplist	Upstream node list
downlist	Downstream node list
lm[]	Array of nodes in the same virtual node
dele[]	Array of delegate nodes

Table 2. Node status information

The search operation here is more efficient compared to the top-down routine in [1], since each hop in Algorithm 1 goes closer to the destination on average. The top-down search routine is well discussed in [1], and it takes expected $O(log(n))$ messages and $O(log(n))$ time, where n is the size of the skip graph. Hence, the search operation in CSG takes expected $O(log(N))$ messages and $O(log(N))$ time, where N is the number of VNs which is bounded by L.

Algorithm 1 Searching Algorithm at Requested Node v

```
1: Upon receiving message {SearchRequest, startNode, KEY};
2: if v.key == KEY then
3:     Send {Found, v.dele} to startNode;
4:     Exit();
5: end if
6: if v.key < KEY then
7:     if v.rflist is empty or v.rflist.first().key > KEY then
8:         Send {NotFound, v.dele} to startNode;
9:         Exit();
10:    end if
11: end if
12: if v.key > KEY then
13:    if v.lflist is empty then
14:        Send {NotFound, v.dele} to startNode;
15:        Exit();
16:    end if
17:    if v.lflist.first().key < KEY then
18:        Send {NotFound, v.lflist.first()} to startNode;
19:        Exit();
20:    end if
21: end if
22: Get far-neighbor fn whose key is closest to KEY;
23: Send {SearchRequest, startNode, KEY} to a delegate node in fn;
```

4.1.2 Node Join

For a new coming client, after the VN search process, if the VN it belongs to does not exist, a new VN is created and inserted into the clustered skip graph. In the insertion operation, the far-neighbors closest to the new VN, which can be known after the search operation, are first notified of the insertion. Algorithm 2 shows the pseudo code for VN insertion when the new VN is right to the processing node (node being notified of the insertion). If the new VN is at the left side of the processing node, the process is just symmetric to it. When a VN receives the notification of a new VN's insertion, it checks if the new VN is a new far-neighbor. Assume the new VN is right to the notified one. If the new VN is a new far-neighbor, the processing VN refreshes its right far-neighbor list and sends a far-neighbor message to the new VN. The processing VN looks up its left far-neighbor list and checks whether there is one that needs be notified. If the VN that the joining client belongs to exists and there are m delegate nodes already, the node only notices the delegate nodes of its joining. If there are less than m nodes in the VN, the joining node becomes a new delegate node, and notices the change of the delegate node set to the far-neighbors of that VN.

Algorithm 2 Joining Algorithm at Processing Node v

```
1: Upon receiving message {Insert, startNode, KEY};
2: if v.key < KEY then
3:    if ∃ VN X in rflist, K(X) <KEY and SBL(v.key, K(X)) ≥
   SBL(v.key, KEY) then
4:       Add VN (startNode) to rflist;
5:       Send {NotFound, v.dele} to startNode;
6:    end if
7:    for each VN X in rflist that K(X) > KEY
8:       if SBL(v.key, K(X)) ≤ SBL(v.key, KEY) then
9:          Delete X in rflist;
10:      end if
11:   Broadcast the change of far neighbors to other delegate nodes;
12:   if ∃ VN Y in lflist, SBL(K(Y), KEY) ≥ SBL(v.key, KEY)
13:   then
14:      Send {Insert, startNode, KEY} to a delegate node in Y;
15:   end if
16: end if
```

4.1.3 Node Departure

If the leaving node is the only node in the VN it belongs to, a VN deletion operation is performed. Algorithm 3 shows the deletion operation performed at the leaving node. The leaving node in this situation notices its far-neighbors of its departure. Together with the leaving notice, a list of VNs that will be new far-neighbors of the noticed one is piggybacked. If the leaving node is a non-delegate node, it just notices the delegate nodes about its leaving. If the leaving node is a delegate node, but isn't the only node in the VN, it randomly picks a non-delegate node to substitute itself and notices the change to all the far-neighbors of the VN. The new picked delegate node broadcasts the change to the nodes in the same VN.

Algorithm 3 Deletion Algorithm at Leaving Node v

```
Require: nlist(X): list of new far-neighbors for X after deletion;
1: for each far-neighbor X in rflist
2:    for each far-neighbor Y in lflist
3:       if ∃ far-neighbor Z in rflist and lflist that K(X) < K(Z) <
   K(Y) and SBL(K(X), K(Z)) ≥ SBL(K(X), K(Y)) then
4:          Add Y to nlist(X);
5:          Add X to nlist(Y);
6:       end if
7:    end for
8: end for
9: for each far-neighbor X in rflist
10:   Send {Delete, v.key, nlist(X)} to a delegate node in X;
11: end for
12: for each far-neighbor Y in lflist
13:   Send {Delete, v.key, nlist(Y)} to a delegate node in Y;
14: end for
```

4.2. Theoretical Analysis

4.2.1 Search Efficiency

Lemma 1 *Let V_i $(i = 1, \cdots, n)$ be a sequence of VNs that $K(V_i) < K(V_{i+1})$ $(i = 1, \cdots, n-1)$ (or $K(V_i) > K(Vi+1)$ for all $i = 1, \cdots, n-1$), V_i and V_{i+1} are far-neighbors $(i = 1, \cdots, n-1)$, for any i, j, $1 \le i < j \le n-1$, V_i and V_{j+1} are not far-neighbors, and $D = |K(V_n) - K(V_1)|$. Then n is no more than $\sqrt{3 \times D + 1}$.*

The proof is omitted due to space limit.

Theorem 2 *In a clustered skip graph constructed by VNs, let x be the joining node, v the first node redirected to, and $D = |x.key - v.key|$. The search operation using Algorithm 1 needs no more than $\sqrt{3 \times D + 1}$ hops.*

Proof: A VN search path can be transformed to the precondition of Lemma 1 so the conclusion can be derived.

The number of VNs in the clustered skip graph is bounded by L, so is the distance of any two VNs. L ranges from dozens to hundreds in practice, i.e., even in an extreme case, the search latency is also well bounded.

There is a special case that for any VN v in the clustered skip graph, if VN u is next to v in the basic linked list, we will have $|v.key - u.key| = 1$, i.e., along the basic linked list, the keys of VNs are monotonically increased by 1. We call such clustered skip graph the **perfect skip graph**.

Lemma 3 *In a perfect skip graph, VN X and VN Y are far-neighbors if and only if there is an integer k, $k \ge 0$ and $|K(X) - K(Y)| = 2^k$.*

Proof: It can be easily derived from the definition of far-neighbors.

Theorem 4 *When using the IP layer hops during the search process to represent the search latency. In a perfect skip graph, a search operation only needs $log(N) + 1$ hops at most, where N is the number of VNs.*

Proof: From lemma 3, we can see that the distance between the processing node and the joining node will decrease at least half of the size, so most $log(N) + 1$ hops is needed.

4.2.2 VN Insertion and Deletion Cost

Theorem 5 *The insertion of a new VN costs $O(log(L))$ time and $O(m \times log(L))$ messages on average. The deletion of a VN costs $O(m \times log(L))$ messages and $O(1)$ time.*

The proof is omitted due to the space limit.

4.3. Data Transmission

4.3.1 Stream Suppliers

For a new coming client, after the VN search process is done, there may be nodes in the closest left far-neighbor that buffer the media segment the new client expects. Potential suppliers may also exist in the same VN the new client belongs to. The address information of such nodes can be obtained from the delegate nodes of the VNs in O(1) time after the VN search. The total stream suppliers search latency will be $O(logN)$ on average since the average VN search time is $O(logN)$. If a node cannot get node suppliers or it cannot get enough downloading speed from the node suppliers, it asks the server for media data transmission.

4.3.2 Stream Scheduling

We propose a stream redirection scheme to maintain stream load balance and to minimize the impact of VCR operations.

When user a requests data from another user s, it sends a message to s denoting the data wanted and the deadline of transmission (d_a). If s cannot finish the transmission before d_a, the requested data does not exist in s or the upload bandwidth from s to a is less than that a expects (other load metrics can be applied here). s finds a set of nodes in its upstream node list, downstream node list and local member array, which have the data a wants. s redirects a to these nodes by sending the set to a. a picks some of them for suppliers (locality can be achieved by selecting suppliers with short round-trip times). By the request redirection, the stream load is balanced among nodes with different upload abilities.

For a newly joined node, after it begins to get data content from potential suppliers, the suppliers redirect some nodes in their downstream node list to the new joined node when they can get data delivered from the newly joined one. The redirection scheme makes that the new joined node can participate relaying as soon as possible. Together with that a user does not clear the buffer when performing a VCR operation such as random seek and still delivers data to its children, the impact to other nodes' playback continuities caused by VCR operations is minimized.

4.4. VCR Operations

4.4.1 Pause and Resume

When a pause operation is performed, the user simply stops playing, but it still stays in the VN and receives data and forwards to its children before the buffer is full of data that is ahead of the stop point. By then, the user can temporarily leave the overlay but still servers as a supplier. It continues to play at the original point and rejoins the overlay when the client resumes. It appears just the same as the user is playing a video on a local disk.

4.4.2 Random Seek

When a random seek is performed, if the client is still in the VN after a short jump, no additional overlay maintenance is needed. Otherwise, the client simply leaves and rejoins the overlay after a search operation. But it keeps the data in buffer and still delivers the data which makes less impact to the playback continuities of its children.

4.4.3 Fast Forward/Backward

When a client performs a fast-forward operation, assume the playback rate is x times of the original and the length of video played during the fast-forward operation is t seconds. Then, the VN the client belongs to changes every $w/(x-1)$ seconds in t/x seconds. Since it is changing rapidly, until the fast-forward operation stops, the client is added to the VN it should be, i.e., a join operation is performed. A leave operation is performed when the client leaves the original VN. Before every change of the VN the client belongs to, it first gets the address information of nodes in the new VN and sends data requests to them.

5. Performance Evaluation

In this section, we evaluate the performance of SkipStream via simulation. The simulation is built on top of a topology of 5000 peer nodes based on the transit-stub model generated by GT-ITM [10]. The length of the movie, denoted as l, is 60 minutes and the bit rate is 400 Kbps (most video stream over the Internet today is encoded in the

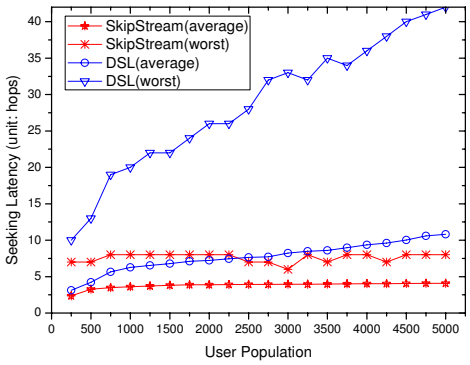

Figure 3. Search latency against varying user population

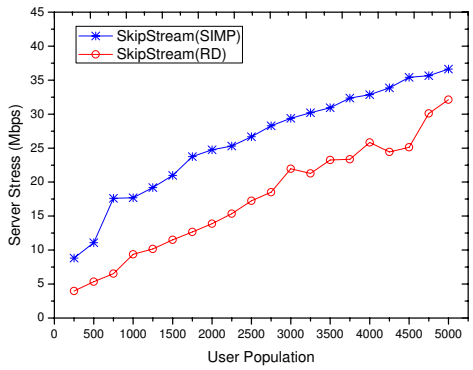

Figure 4. Server Stress against varying user population

200-400 Kbps range [7]). The minimum size of the sliding window buffer, denoted as B, is 2MB, i.e., each peer can at least cache 40 second recent stream. The buffer size of the users in the simulation randomly ranges from 2MB to 4MB (40s-80s) and the upload bandwidth of a user randomly ranges from 280Kbps to 520Kbps (70%-130% to the playback bit rate). The width of a VN, denoted as w, is set to 20s. $L = l/w = 180$. The number of delegate nodes in a VN, denoted as m, is set to 2. The arrival of peers follows the Poisson Process with $\lambda = 3$. We also compare some simulation results with DSL [9].

5.1. Search Latency

We use the hops over the IP layer to estimate the search latency. Fig.3 shows that the average number of hops for search operations in SkipStream is much smaller than that in DSL. It is reduced by 20%-60%. In SkipStream, the average number of hops for search does not increase with the user population after it reaches about 4, which coincides with the discussion in Section 4. While in DSL, it is logarithmic to the user population. The number of hops for search operations in the worst case in SkipStream is bounded by 8, and is even smaller than the average number in DSL when the user population goes over 2500. Since skip lists have bad worst case performance, the number of hops in the worst case in DSL reaches as high as 40. It means users in DSL will have longer start up latency as well as longer seeking latency which is not well bounded. Hence, SkipStream is much more scalable from the user satisfaction point of view.

5.2. Server Stress

Fig.4 shows that, even in an extreme situation that the upload bandwidth of each user randomly ranges from 280Kbps to 520Kbps (70% - 130% to the play bit rate), the server stress for SkipStream increases very slowly. It only cost about 2% of the bandwidth of that in a client-server architecture. Obviously, SkipStream with stream redirection (SkipStream(RD)) reduces more server stress. This is because the stream redirection scheme uses the upload abilities of different kinds of users more efficiently. Hence, SkipStream (RD) reduces more server stress by 10% - 50%.

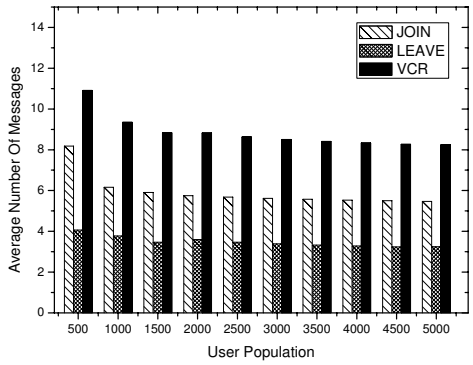

Figure 5. Control messages

5.3. Control Overhead

In our simulation, the control overhead is counted under the environment where each node performs 7 random seeks on average in its session time according to the statistics in [13]. Fig.5 shows that the average numbers of messages for node join, leave and VCR operations do not increase as the average user population increases. In contrast, the numbers converge to small values because the percentage of normal nodes other than delegate nodes increases, which means average control overhead needed for each operation decreases. The control overhead for join is higher than that for leave, mainly due to that the supplier acquiring mes-

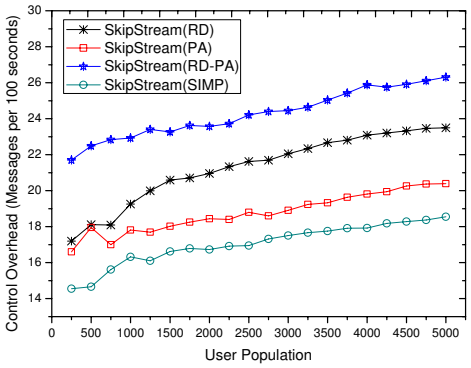

Figure 6. Control overhead

sages are counted for join. Fig.6 records all SkipStream control messages as control traffic overhead, such as messages for searching, node joining, node departure, stream redirection, periodical information exchanging, etc. SkipStream consumes very low control overhead ranging from 14 to 27 messages a node per 100 seconds which is negligible compared to the stream bandwidth costs. SkipStream (PA) represents SkipStream with parallel searching, which makes the search latency smaller in practice. SkipStream (RD-PA) represent SkipStream with both parallel searching and stream redirection, which only increases the control overhead by about 8 messages every node per 100 seconds. It is worthwhile due to the server stress and search latency reduction, which is more important for system scalability.

6. Conclusions

In this paper, we propose SkipStream, a skip graph based P2P on-demand streaming system to provide scalable and reliable streaming services with VCR functionality over ubiquitous environments. SkipStream mainly focuses on the scalability and reliability problems related to providing on-demand streaming with VCR support. By grouping users into a set of disjoint clusters in accordance with their playback offset and further organize the resulted clusters into a clustered skip graph, we first provide a hierarchical overlay network to facilitate the streaming cooperation among heterogeneous nodes. Following that we present a distributed on-demand streaming scheduling mechanism over the clustered skip graph to achieve load balance and minimize the impact of VCR opereations. The search latency of SkipStream on average is $O(log(N))$ where N is the number of disjoint clusters. The search efficiency and low control overhead of SkipStream are shown by our experiment results.

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. 90718031, 60721002; the National Basic Research Program of China (973) under Grant No. 2009CB320705.

References

[1] J. Aspnes and G. Shah. Skip graphs. In *Proc. of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, Baltimore, MD, USA, Jan 2003.

[2] Y. Cui, B. Li, and K. Nahrstedt. oStream: Asynchronous streaming multicast in application layer overlay networks. In *IEEE JSAC*, volume 22, pages 91–106, Jan 2004.

[3] 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 2004*, Paris, France, Jun 2004.

[4] Y. Guo, K. Suh, J. Kurose, and D. Towsley. P2Cast: Peer-to-peer patching scheme for vod service. In *Proc. of the 12th ACM International World Wide Web Conference (WWW'03)*, Budapest, Hungary, May 2003.

[5] Y. Guo, K. Suh, J. Kurose, and D. Towsley. A peer-to-peer on-demand streaming service and its performance evaluation. In *Proc. of the ICME 2003*, Baltimore, USA, Jul 2003.

[6] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using collectcast. In *Proc. of 11th ACM International Conference on Multimedia (MM'03)*, Berkeley, CA, USA, Nov 2003.

[7] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *Proc. of ACM SIGCOMM'07*, Kyoto, Japan, Aug 2007.

[8] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Communications of ACM*, volume 33, pages 668–676, Jun 1990.

[9] 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 (TPDS)*, volume 19, pages 503–514, Apr 2008.

[10] B. Ye, M. Guo, and J. Zhou. CoopStream: A cooperative cache based streaming schedule scheme for on-demand media services on overlay networks. In *Proc. of ICPP 2006*, Columbus, OH, USA, Aug 2006.

[11] W. P. K. Yiu, X. Jin, and S. H. G. Chan. VMesh: Distributed segment storage for peer-to-peer interactive video streaming. In *IEEE JSAC*, volume 25, pages 1717–1731, Dec 2007.

[12] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM'96*, San Francisco, CA, USA, May 1996.

[13] C. Zheng, G. Shen, and S. Li. Distributed prefetching scheme for random seek support in peer-to-peer streaming applications. In *Proc. of the ACM Workshop on Advances in Peer-to-Peer Multimedia Streaming (P2PMMS'05)*, Hilton, Singapore, Nov 2005.