J. Parallel Distrib. Comput. 70 (2010) 537–546

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

On the source switching problem of Peer-to-Peer streaming

Zhenhua Li [a], Jiannong Cao [b,*], Guihai Chen [c], Yan Liu [b]

[a] Department of Computer Science and Technology, Peking University, China
[b] Internet and Mobile Computing Lab, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
[c] Department of Computer Science and Engineering, Shanghai Jiaotong University, China

ARTICLE INFO

Article history:
Received 24 May 2009
Received in revised form
9 September 2009
Accepted 18 January 2010
Available online 2 February 2010

Keywords:
Peer-to-Peer
Multimedia streaming
Source switching

ABSTRACT

Peer-to-Peer(P2P) streaming has been proved a popular and efficient paradigm of Internet media streaming. In some applications, such as an Internet video distance education system, there are multiple media sources which work alternately. A fundamental problem in designing such kind of P2P streaming system is how to achieve fast source switching so that the startup delay of the new source can be minimized. In this paper, we propose an efficient solution to this problem. We model the source switch process, formulate it into an optimization problem and derive its theoretical optimal solution. Then we propose a practical greedy algorithm, named *fast source switch algorithm*, which approximates the optimal solution by properly interleaving the data delivery of different media sources. The algorithm can adapt to the dynamics and heterogeneity of real Internet environments. We have carried out extensive simulations on various real-trace P2P overlay topologies to demonstrate the effectiveness of our model and algorithm. The simulation results show that our proposed algorithm outperforms the normal source switch algorithm by reducing the source switch time by 20%–30% without bringing extra communication overhead. The reduction in source switching time is more obvious as the network scale increases.

1. Introduction

1.1. P2P streaming

The application level Peer-to-Peer(P2P) streaming has been proved a popular and efficient decentralized paradigm of Internet media streaming in recent years, e.g. P2P voice systems like Skype [23] and UUCall [26], and P2P video systems like Joost [9], PPLive [18], PPStream [19], UUSee [27] and so on.

Existing P2P streaming systems can be generally classified into three categories: tree-based, gossip-based, and the hybrid. Tree-based systems, e.g. Bayeux [34], SCRIBE [6], NICE [4], and ZIGZAG [24], organize nodes into a multicast tree. The root of the tree is the media source and data segments are always delivered from parent to sons. Tree-based method can minimize redundancy of data delivery and ensure full coverage of data dissemination, but cannot well adapt to network dynamics because even the failure of a single node will partition the tree to a forest. Besides, in the multicast tree all leaf nodes are just consumers and contribute little to other nodes. Taking these into consideration, SplitStream [5], CoopNet [15] and Chunkyspread [28] use multiple trees to increase the resilience and balance the load. However, multiple trees bring much higher maintenance overhead.

Gossip-based systems, also referred to as mesh-based systems, have been proved to be effective and resilient especially in dynamic and heterogeneous network environments. In a typical gossip algorithm [8], every node maintains a limited number of neighbors and sends a newly generated or received data segment to a random subset of its neighbors. The random choice of data forwarding targets achieves high resilience to random failures and enables distributed operations. However, direct use of gossip for streaming is ineffective because its random push may cause significant data redundancy. As a result, existing gossip-based P2P streaming systems, e.g. CoolStreaming [32], PeerStreaming [10], PALS [20], AnySee [11] and PRIME [14], adopt a smart pull-based gossip algorithm: every node periodically exchanges data availability information with its neighbors and then retrieves required data segments from a subset of its neighbors.

Recently, a hybrid architecture for P2P streaming has been proposed to integrate the merits of multicast tree and gossip mesh, e.g. Climber [16], mTreebone [29] and CliqueStream [3]. In these systems, usually the stable nodes are organized into one or several multicast tree(s), while the unstable nodes form a gossip mesh. Nevertheless, there has been no practical P2P streaming system which utilizes the hybrid method till now.

* Corresponding author.
E-mail addresses: lzh@net.pku.edu.cn (Z. Li), csjcao@comp.polyu.edu.hk (J. Cao), gchen@nju.edu.cn (G. Chen), csyliu@comp.polyu.edu.hk (Y. Liu).

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.01.005

1.2. Motivation

In this paper, we focus on the study of gossip-based P2P streaming systems which have multiple sources that generate and disseminate data segments to other nodes. For a multiple-source system, the sources may work in parallel or in order (or says alternately). Here are two application examples:

- *Parallel sources*: In an Internet TV system, there usually exist several parallel streaming sources (i.e. the TV channels) working simultaneously to disseminate their TV contents.
- *Alternate sources*: In a video distance education system all system members lying in different places can become the streaming source, but there is usually only one source (i.e. the speaker) at a time, so the sources work alternately.

So far, almost all the previous works focus on the research of multiple-source P2P streaming systems with parallel sources. However, during our system design practice which aims at upgrading an Internet video distance education system from the Client/Server architecture (the Human-Centered Multimedia E-Learning System [13] has been developed and in use) to Peer-to-Peer architecture, we are motivated to consider this special but useful kind of P2P streaming system in which multiple sources exist and work alternately.

In a P2P streaming system with alternate sources, a fundamental problem is how to make the source switch process fast, that is to say, how to minimize the startup delay of the new source. On the contrary, the source switch problem is usually trivial for a P2P streaming system with parallel sources, because in this scenario the solution is straightforward. In a word, P2P streaming systems with alternate and parallel sources have different requirements on their source switch process and then need different solutions to their source switch problem. Here are two examples indicating their different requirements and solutions:

- *Parallel sources*: Suppose you are watching TV Channel 1 with a P2P TV tool and suddenly you want to switch to TV Channel 2. Just on receiving your switch command, the data scheduler of the P2P TV software will at once stop receiving and displaying any data from Channel 1. And then the data scheduler spends all the available bandwidth in receiving data from Channel 2, because Channel 1 has been meaningless to you when you send the switch command.
- *Alternate sources*: Suppose you are attending a video distance education class. Since every class member has his chance to become the speaker, a source switch process often happens between a teacher and his assistant, a teacher and a student, a teacher and another teacher, and so on. In this situation, the P2P data scheduler cannot start displaying the new source until it has finished displaying the old source, so the solution to the source switch problem is not straightforward any more.

Fig. 1 demonstrates a source switch process between alternate sources S_1 and S_2. It is composed of three phases. (a) At first the old source S_1 was streaming its contents and every node was receiving and playing the data segments of S_1. (b) Then S_1 stopped streaming and the new source S_2 started streaming. Both the data segments of S_1 and S_2 were being disseminated amongst all the non-source nodes. (c) Finally every node had finished the whole playback of S_1, and only the data segments of S_2 were being disseminated in the system. Obviously, the source switch problem is essentially how to minimize the duration of phase (b). More specifically, we need to design a proper source switch algorithm for every node to minimize its playback start time (or says the startup delay) of S_2, on condition that a node can start its playback of S_2 only when 1) it has finished the whole playback of S_1, and 2) it has gathered sufficient data segments of S_2.

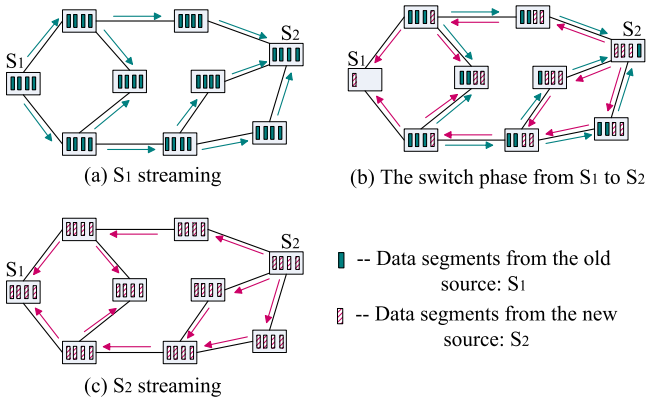

Fig. 1. A source switch process from the old source S_1 to the new source S_2.

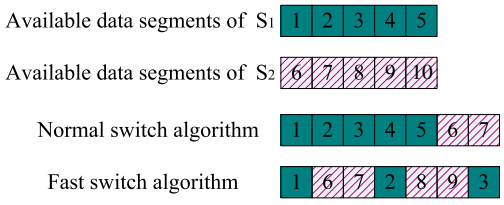

Fig. 2. A comparison of our fast switch algorithm and the normal switch algorithm.

1.3. Our work

To solve the above-mentioned source switching problem, we first model the source switch process by capturing its essential features. Then, we formulate the source switching problem into an optimization problem, and derive the theoretical optimal solution to this optimization problem. To address the more complicated constrains in real Internet environments, we propose a practical greedy algorithm, named *fast switch algorithm*, that approximates the optimal solution by properly interleaving the data delivery of the old source and the new source. This algorithm is embedded into the architecture of a P2P streaming client software as a *bridge module* between the data scheduler and rate controller. It is triggered and executed by every node independently and it relies on only local computation, which does not need any extra communication. Therefore, this *bridge module* does not lead to any negative impact on the performance in application layer.

We have done comprehensive simulations on various real-trace overlay topologies, scaling from 100 to 10 000 nodes, to demonstrate the effectiveness of our algorithm. The simulation results show that our proposed fast switch algorithm outperforms the *normal switch algorithm* by reducing the source switch time by 20%–30% without bringing extra communication overhead, and the reduction in source switching time is more obvious as the network scale increases. The normal switch algorithm is a straightforward and greedy data scheduling method which can be brought to mind when one tries to handle the source switch process. It does not interleave the data delivery of the old source and the new source. Instead, it always gives priority to the data delivery of the old source. The example in Fig. 2 shows the difference between the two algorithms. The current node can receive 7 data segments per scheduling period but there exist 10 available data segments, 5 segments from S_1 and 5 segments from S_2. Each algorithm arranges the order of data delivery according to its own computation of the data priorities, but we would see their different results in the remaining parts of this paper.

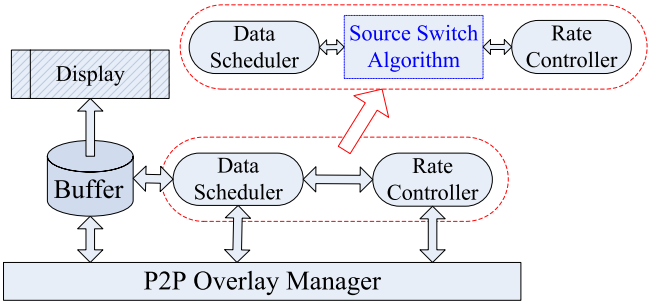

Fig. 3. In the architecture of P2P streaming client software, the source switch algorithm is embedded as a *bridge module* between the data scheduler and rate controller.

In summary, the contribution of this paper lies in four aspects:

(1) To the best of our knowledge, we are the first to clarify the existence and significance of the source switching problem of P2P streaming. In particular, we distinguish alternate sources from parallel sources in multi-source P2P streaming systems.
(2) To solve the source switching problem, we model the source switch process, formulate it into an optimization problem, and derive its theoretical optimal solution.
(3) We propose a practical greedy algorithm that approximates the optimal solution by properly interleaving the data delivery of the old source and the new source, so that this algorithm can adapt to the dynamics and heterogeneity of real Internet environments.
(4) We demonstrate the effectiveness of our proposed algorithm through comprehensive simulations on various real-trace overlay topologies.

The rest of this paper is organized as follows. Section 2 introduces the system background. Section 3 models the source switch process. Section 4 presents our proposed fast source switch algorithm and we evaluate its performance by simulation in Section 5. Section 6 overviews the related work. Finally, we conclude the paper in Section 7.

2. System background

A P2P streaming system is usually composed of two kinds of nodes: (1) one or several rendezvous servers, and (2) a lot of client nodes. A rendezvous server maintains a partial list of live client nodes and its role is to help the client nodes join the system or update the neighbor table. The real streaming media data is only exchanged among the client nodes by using the P2P streaming client software.

From Fig. 3 we can see that the architecture of P2P streaming client software usually consists of the following key modules:

- *P2P Overlay Manager* behaves as the interface between the local node and overlay network. It maintains and updates the neighbor table.
- *Data Scheduler* gets the information about available data segments of neighboring nodes and arranges where and how to retrieve required data.
- *Rate Controller* monitors and estimates the receiving rates from each neighboring node and the sending rates to each neighboring node. It collects rate information for the data scheduler and gets feedback from the data scheduler.

In the architecture of P2P streaming client software, our proposed source switch algorithm is embedded as a *bridge module* between the data scheduler and rate controller. When a node detects a source switch process, it activates the source switch module, which coordinates the data scheduler and rate controller so as to accelerate the switch process. And when the source switch process finishes, the source switch module is set to inactive and behaves like a bypass channel between the data scheduler and rate controller.

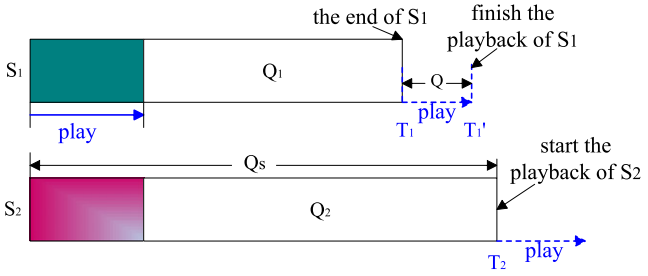

Fig. 4. The time sequence graph corresponding to our model of the source switch process.

Table 1
Model parameters.

Param	Description
S_1	The old source.
S_2	The new source.
Q	The stream from S_1 is played when every Q new consecutive data segments of S_1 have been gathered. If the number of new consecutive data segments is not enough ($\leq Q$), the playback would be stopped at once.
Q_1	The number of undelivered data segments of S_1.
Q_s	The number of required data segments of S_2 to start the playback of S_2.
Q_2	The number of undelivered data segments of S_2 to start the playback of S_2. Initially $Q_2 = Q_s$.
p	The number of data segments being played per second.
I	Total inbound rate of the local node. The rate is measured by the number of data segments per second. I is a constant.
I_1	The inbound rate allocated to receive data segments of S_1. I_1 is dynamically configured.
I_2	The inbound rate allocated to receive data segments of S_2. I_2 is dynamically configured.
T_1	The expected time to receive all the undelivered data segments of S_1.
T_1'	The expected time to finish the playback of S_1.
T_2	The expected time to receive the first Q_s data segments of S_2.

3. Model the source switch process

Since the P2P streaming system we consider is fully distributed, a node does not know the source switch process until it discovers data segments of a new source in its neighbors, that is to say, the source switch algorithm assumes no pre-arranged knowledge on the ordering of the sources' sessions. When a node discovers the new source it activates its source switch algorithm and re-executes the switch algorithm per scheduling period until the node finishes the whole playback of the old source. We assume there exists a mechanism for synchronizing the old source S_1 and the new source S_2, so that S_2 knows when S_1 stops generating streaming data. For example, S_1 can send a unicast message that contains the *id* of S_1's ending segment to S_2 after S_1 sends its ending segment. When receiving this message, S_2 adds the *id* of S_1's ending segment into S_2's data segments to notify the other nodes.

The parameters used in modeling the source switch process are shown in Table 1. Considering the mathematical symbols' obscurity, we use Fig. 4 to visualize these parameters. The stream from S_1 is played when every Q new consecutive data segments of S_1 have been gathered, but the stream from S_2 cannot be started to play until the first Q_s data segments of S_2 have been gathered. The constraints on Q and Q_s come from the basic requirements of P2P live streaming to reduce playback jitter. In a practical P2P streaming system, Q_s is generally configured much bigger than Q to guarantee a *smooth startup* of the new source. Because the playback of S_2 must follow the playback of S_1, $T_2 \geq T_1'$ is a necessary

condition. The total inbound rate I is a constant for the local node and I is divided into I_1 and I_2 to receive data segments of S_1 and S_2 respectively. I_1 and I_2 are dynamically configured by the source switch algorithm.

The problem of fast source switching can be formulated into the following optimization problem:

Minimize T_2
subject to the following conditions:

$$\begin{cases} I = I_1 + I_2; \\ T_1 = \dfrac{Q_1}{I_1}; \\ T_1' = T_1 + \dfrac{Q}{p}; \\ T_2 = \dfrac{Q_2}{I_2}; \\ T_2 \geq T_1'. \end{cases}$$

The conditions can be rewritten as

$$\begin{cases} T_1' = \dfrac{Q_1}{I_1} + \dfrac{Q}{p}; \\ T_2 = \dfrac{Q_2}{I - I_1}; \\ T_2 \geq T_1'. \end{cases}$$

So we get the following inequality

$$\frac{Q_2}{I - I_1} \geq \frac{Q_1}{I_1} + \frac{Q}{p}; \tag{1}$$

which can be reformulated as

$$I_1^2 + \left(\frac{p(Q_1 + Q_2)}{Q} - I\right) I_1 - \frac{pIQ_1}{Q} \geq 0. \tag{2}$$

Solving the above inequality, we have the following

$$I_1 \geq r_1 \quad \text{or} \quad I_1 \leq r_1'; \tag{3}$$

$$r_1 = \frac{I - \frac{p(Q_1+Q_2)}{Q} + \sqrt{(\frac{p(Q_1+Q_2)}{Q} - I)^2 + \frac{4pIQ_1}{Q}}}{2} \tag{4}$$

$$r_1' = \frac{I - \frac{p(Q_1+Q_2)}{Q} - \sqrt{(\frac{p(Q_1+Q_2)}{Q} - I)^2 + \frac{4pIQ_1}{Q}}}{2}. \tag{5}$$

Clearly $r_1' < 0$ and thus $I_1 \leq r_1'$ is not a feasible region. $I_1 \geq r_1$ is the only solution. Therefore, in order to minimize T_2 we let $I_1 = r_1$ and $I_2 = r_2 = I - r_1$, which is the theoretical optimal solution to the optimization problem.

4. Fast source switch algorithm

The ideal condition for achieving the theoretical optimal solution does not always hold because the real Internet environments usually involve more complicated constraints. For example, $I_1 + I_2$ may be less than I when the total incoming data rate are not enough to fill up the inbound rate of local node. This may be caused by $I_1 \leq O_1$ or $I_2 \leq O_2$ or both (the meaning of O_1, O_2 will be explained in the next paragraph). Therefore, we need a practical source switch algorithm that can approximate the optimal solution in dynamic and heterogeneous environments.

Fig. 5 demonstrates the local working environment of a node. The local node has neighbors N_1, N_2, N_3, N_4 with outbound rate o_1, o_2, o_3, o_4 respectively. Suppose O_1 is the total available outbound rate for the data delivery of S_1 and O_2 is the total available outbound rate for the data delivery of S_2, then the optimization problem in Section 3 is changed to:

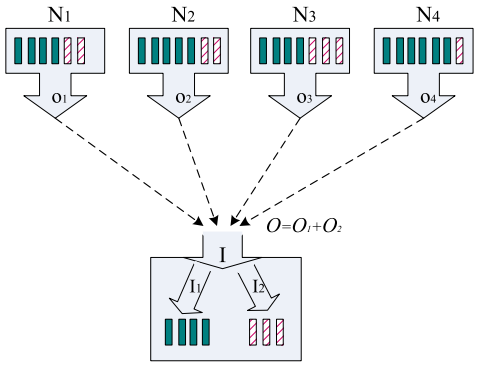

Fig. 5. The local working environment of a node.

Minimize T_2
subject to the following conditions:

$$\begin{cases} I_1 + I_2 \leq I; \\ I_1 \leq O_1; \\ I_2 \leq O_2; \\ T_1 = \dfrac{Q_1}{I_1}; \\ T_1' = T_1 + \dfrac{Q}{p}; \\ T_2 = \dfrac{Q_2}{I_2}; \\ T_2 \geq T_1'. \end{cases}$$

Under the above conditions, the solution $I_1 = r_1, I_2 = r_2$ we get in Section 3 can only hold when $r_1 \leq O_1$ and $r_2 \leq O_2$. r_1 is defined in the Eq. (4) and $r_2 = I - r_1$. Therefore, when $r_1 > O_1$ or $r_2 > O_2$ our target is to maximize the inbound throughput of the local node. Then the solutions become:

- Case 1: when $r_1 \leq O_1$ and $r_2 \leq O_2$,
  Then $I_1 = r_1, I_2 = r_2$;
- Case 2: when $r_1 \leq O_1$ and $r_2 > O_2$,
  Then $I_1 = \min(O_1, I - O_2), I_2 = O_2$;
- Case 3: when $r_1 > O_1$ and $r_2 \leq O_2$,
  Then $I_1 = O_1, I_2 = \min(O_2, I - O_1)$;
- Case 4: when $r_1 > O_1$ and $r_2 > O_2$,
  Then $I_1 = O_1, I_2 = O_2$.

Now the critical problem is how to compute O_1 and O_2, more exactly, to compute the two sets $\mathbb{O}_1$ and $\mathbb{O}_2$, where $O_1 = |\mathbb{O}_1|$ and $O_2 = |\mathbb{O}_2|$. Note that we measure the outbound rate O_1, O_2 with the number of data segments here (each data segment has the same size). $\mathbb{O}_1, \mathbb{O}_2$ are recomputed for every scheduling period. Data segments in $\mathbb{O}_1$ are in descending order of their priorities and $\mathbb{O}_2$ is alike. Required parameters for our algorithm are listed in Table 2.

Taking both the urgency and rarity of each data segment into consideration, a data segment D_i's requesting priority is computed through Eqs. (6) to (10).

$$R_i = \max\{R_{i_1}, R_{i_2}, \ldots, R_{i_{n_i}}\} \tag{6}$$

$$t_i = \frac{id_i - id_{play}}{p} - \frac{1}{R_i} \tag{7}$$

then

$$urgency_i = \frac{1}{t_i}. \tag{8}$$

Segment i's rarity is the probability it will be replaced in all its suppliers' buffers, which we think is more reasonable than the traditional computation $rarity_i = \frac{1}{n_i}$.

$$rarity_i = \left(\frac{p_{i_1}}{B}\right) \times \left(\frac{p_{i_2}}{B}\right) \times \cdots \times \left(\frac{p_{i_{n_i}}}{B}\right). \tag{9}$$

Table 2
Parameters for our algorithm.

Param	Description
τ	Data scheduling period.
id_i	The *id* of data segment D_i.
n_i	The number of neighbors that can supply the data segment D_i.
R_{i_j}	The receiving rate of data segment D_i from the jth neighbor.
R_i	The maximum receiving rate of data segment D_i.
id_{play}	The *id* of the data segment being played at this moment.
id_{end}	The *id* of the ending data segment of S_1.
id_{begin}	The *id* of the beginning data segment of S_2. We set $id_{begin} = id_{end} + 1$.
t_i	The expected deadline left time of data segment D_i.
B	Buffer size, i.e. the number of data segments Buffer can accommodate.
p_{i_j}	Data segment D_i's position in the jth neighbor's buffer. The replacement strategy of Buffer is FIFO, and the position is the distance from the tail of Buffer.
$urgency_i$	The urgency of data segment D_i, i.e. the probability of D_i to miss its deadline.
$rarity_i$	The rarity of data segment D_i, i.e. the probability that D_i will be replaced in all its suppliers' buffers.
$priority_i$	The requesting priority of data segment D_i. It takes both urgency and rarity into consideration.

And finally,

$$priority_i = \max\{urgency_i, rarity_i\} \tag{10}$$

because according to our observation, $urgency_i$ and $rarity_i$ have the same importance in deciding $priority_i$. $priority_i$ cannot be a weighted mean of the two metrics. Either a big $urgency_i$ or a big $rarity_i$ can easily lead to the loss of data D_i.

Having got each data segment's priority, our proposed fast source switch algorithm is able to compute $\mathbb{O}_1, \mathbb{O}_2$ and then arrange the data retrieval process, see Algorithm 1. The data segments are sorted in the descending order of their priorities. Usually the data segments of S_1 and S_2 are mixed in this order. Suppose the order is like $D_1, D_2, D_3, \ldots, D_m$. For a segment D_i, there may exist several neighbors who can supply it, and usually the neighbor who can send it earliest will become D_i's supplier. But here we encounter a conflict problem where two segments choose the same supplier, so one of them needs to wait or choose another supplier. The problem is: how to choose a proper supplier for every data segment so that the number of segments missing deadlines or being replaced can be the minimal? In fact, even a simple special case of this problem is NP-hard (known as the *Parallel machine scheduling problem* [7]), so we use a greedy algorithm trying to get high-priority segments as early as possible. In this algorithm, the scheduler makes greedy efforts to minimize the expected receiving time t_{min} of every data segment. For a data segment D_i, the scheduler checks all its suppliers to find a proper supplier which can send D_i earliest.

After getting $\mathbb{O}_1$ and $\mathbb{O}_2$, the computation of I_1 and I_2 follows one of the four cases described formerly. And the data retrieval is straightforward.

Algorithm 1 Fast Source Switch Algorithm

Input:
Data segments $D_1, D_2, D_3, \ldots, D_m$, in descending order of priority;
Supplier set for each segment: $S_1, S_2, S_3, \ldots, S_m$;
Sending rate of node j: R(j);
Queuing time of node j: $\tau(j)$, initially $\tau(j) = 0$;

Step 1: Computing $\mathbb{O}_1$ and $\mathbb{O}_2$
for $i = 1$ to m **do**
  set segment D_i's earliest receiving time $t_{min} = \infty$;
  suppose S_i contains k suppliers $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$;
  for $j = 1$ to k **do**
    compute the expected transfer time of D_i from S_{i_j}: $t_{trans} = \frac{1}{R(S_{i_j})}$;
    if $t_{trans} + \tau(S_{i_j}) < t_{min}$ **and** $t_{trans} + \tau(S_{i_j}) < \tau$ **then**
      $t_{min} \leftarrow t_{trans} + \tau(S_{i_j})$; $supplier_i \leftarrow S_{i_j}$;
    end if
  end for
  if $supplier_i \neq null$ **then**
    $\tau(supplier_i) \leftarrow t_{min}$;
    add D_i to its corresponding set $\mathbb{O}_1$ or $\mathbb{O}_2$;
  end if
end for

Step 2: Arranging Data Retrieval
compute I_1 and I_2 according to $\mathbb{O}_1, \mathbb{O}_2, r_1$ and r_2;
retrieve the first I_1 data segments of $\mathbb{O}_1$;
retrieve the first I_2 data segments of $\mathbb{O}_2$;

5. Performance evaluation

5.1. Simulation methodology

To evaluate the performance of our proposed source switch algorithm, we perform simulations on 30 real-trace unstructured P2P overlay topologies whose data was collected from Dec. 2000 to Jun. 2001 on dss.clip2.com (this DSS trace is a dataset which has been widely used to analyze the performance of P2P systems [2,1]). The data contains each node's ID, IP, host name, port, ping time, speed and so on, but we just use the ID, IP and ping time information. The trace topologies we used scale from 100 to 10 000 nodes. They represent typical unstructured P2P topologies which have a relatively small average node degree, e.g. the Gnutella overlay network. Most topologies show a loose power-law distribution with high deviation.

Since the average node degree of the P2P overlay topologies we used is too small for media streaming, we add random edges into each overlay to increase the connected neighbors in average. According to our simulation experience, every node holds $M = 5$ neighbors is usually a good practical choice. The default streaming rate is 300 Kbps (this is the data rate most Internet streaming websites offer for normal video quality) and each data segment contains 30 Kb, so the playback rate $p = \frac{300\text{ Kb}}{30\text{ Kb}} = 10$. Each node maintains a Buffer of 600 data segments, i.e. one minute of stream. We randomly arrange inbound rate (from 300 Kbps to 1 Mbps) to each node and let the average inbound rate be 450 Kbps, i.e. $I \in [10, 33]$ and $I = 15$ in average. The arrangement of outbound rate is alike. An exception is that the source node has zero inbound rate and much larger outbound rate, which is consistent with the common situation. The data scheduling period $\tau = 1.0$ s.

For each simulation, we first let the system run for a sufficient period of time to enter its stable phase (i.e. most nodes can continuously playback the stream), and then stop S_1 from generating new data segments and meanwhile choose a new source S_2 to generate new data segments. Any node that can provide sufficient outbound rate has chance to be the source. Therefore, in all the following paragraphs the simulation time "0" means the time when S_1 stops and S_2 starts. The stream from S_1 is played once $Q = 10$ consecutive data segments of S_1 have been gathered. The total number of required data segments of S_2 to start the playback of S_2 is $Q_s = 50$, i.e. 5 s of the streaming media.

We compare the performances of our fast switch algorithm with the normal switch algorithm. The normal switch algorithm works as follows: for a node n when its neighbors can supply data segments of both S_1 and S_2, node n would retrieve data segments of S_1 in priority. If n still has available inbound rate after retrieving

data segments of S_1, it would allocate the remaining inbound rate to retrieve data segments of S_2.

5.2. Metrics

We mainly use the following three metrics to evaluate the performance of our fast switch algorithm:

(1) *Average preparing time of* S_2 (= *Average switch time*) means the average time for all nodes to prepare sufficient data segments of S_2 to start the playback of S_2.
(2) *Reduction ratio* means the reduction ratio of average source switch time by using the fast switch algorithm compared with using the normal switch algorithm.
(3) *Communication overhead:* For every scheduling period each node exchanges buffer information with its neighbors. Communication overhead is defined as the ratio of communication cost for buffer information exchange over the real communication cost for data segments transfer.

We also measure some supplementary metrics which can help to understand the source switch process. The supplementary metrics include:

- *Undelivered ratio of* S_1 ($=\frac{Q_1}{Q_0}$) means the ratio of the undelivered data segments of S_1 currently (Q_1) to the undelivered data segments of S_1 at time "0" (Q_0).
- *Delivered ratio of* S_2 ($=\frac{Q_s-Q_2}{Q_s}$) means the ratio of the delivered data segments of S_2 (Q_s-Q_2) to the total required data segments of S_2 to start the playback of S_2 (Q_s).
- *Unfinished number of* S_1 means how many nodes have not finished gathering the undelivered data segments of S_1.
- *Prepared number of* S_2 means how many nodes have prepared sufficient data segments of S_2 to start the playback of S_2.
- *Average finishing time of* S_1 ($=T_1'$) means the average time for all nodes to finish the playback of S_1.

5.3. Simulation results in static environments

We first track the undelivered ratio of S_1 and delivered ratio of S_2 of our fast switch algorithm and the normal switch algorithm in a static network environment with 1000 nodes. From Fig. 6 we can see that the normal switch algorithm gathers the undelivered data segments of S_1 more quickly than the fast switch algorithm but prepares sufficient data segments to start the playback of S_2 more slowly. Fig. 7 records the track of unfinished number of S_1 and prepared number of S_2. By using the normal switch algorithm, the last node finishes S_1 at time 15 but prepares S_2 at time 24. Note that the last node that finishes S_1 is usually different from the last node that prepares S_2. Meanwhile, by using the fast switch algorithm, the last node finishes S_1 and prepares S_2 both at time 18. So we can find the fast switch algorithm brings on a "compromise" between the speeds of gathering data segments of S_1 and S_2, and thus makes the whole source switch process faster.

We further examine the average finishing time of S_1 and average preparing time of S_2 of overlay networks with different sizes, ranging from 100 to 8000, working in static network environments. The bar graph in Fig. 8 illustrates the results. For each size there are 4 bars corresponding to (from left to right): (1) the average finishing time of S_1 by using the normal switch algorithm; (2) the average finishing time of S_1 by using the fast switch algorithm; (3) the average preparing time of S_2 by using the fast switch algorithm; (4) the average preparing time of S_2 by using the normal switch algorithm. The 4 bars of each size indicates that the fast switch algorithm splits the difference between the average finishing time of S_1 and preparing time of S_2 of the normal switch algorithm, and thus makes the startup delay of the new source shorter.

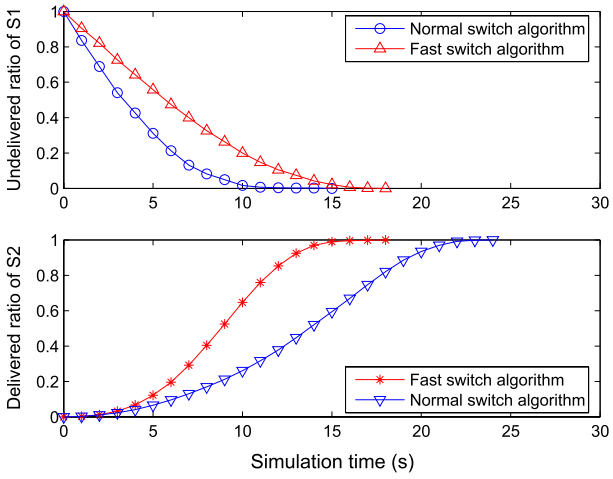

Fig. 6. Ratio track in a static network with 1000 nodes.

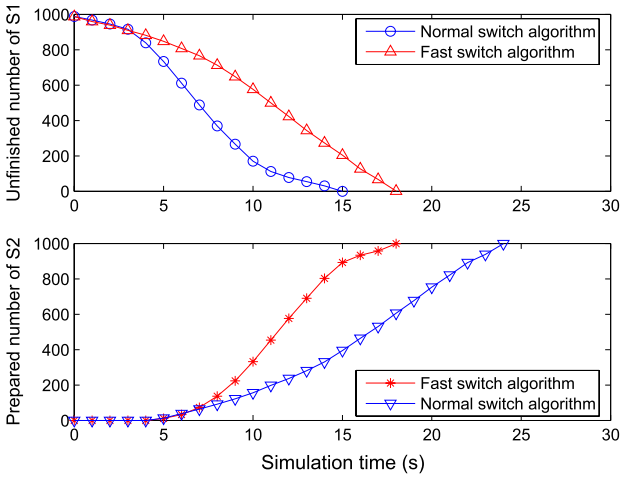

Fig. 7. Number track in a static network with 1000 nodes.

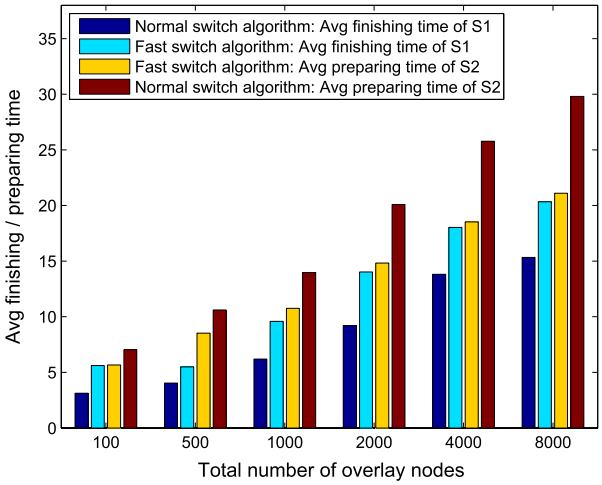

Fig. 8. Avg finishing time of S_1 and preparing time of S_2 in static environments.

To illustrate the effect more clearly, the average switch time and its reduction by using the fast switch algorithm are shown in Fig. 9. We can see the reduction ratio lies between 0.2 and 0.3, and it tends to increase when the network scale expands.

Besides, we measure the communication overhead of the two algorithms in overlay networks with different sizes. The buffer can accommodate $B = 600$ data segments, so we use 600 bits to record the data availability. "1" indicates that the corresponding

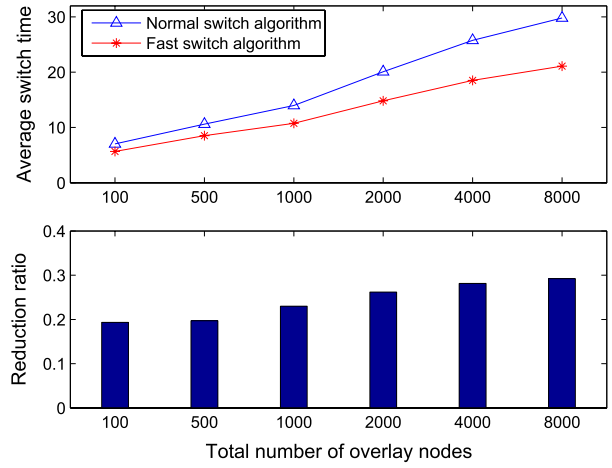

Fig. 9. Avg switch time and its reduction ratio in static environments.

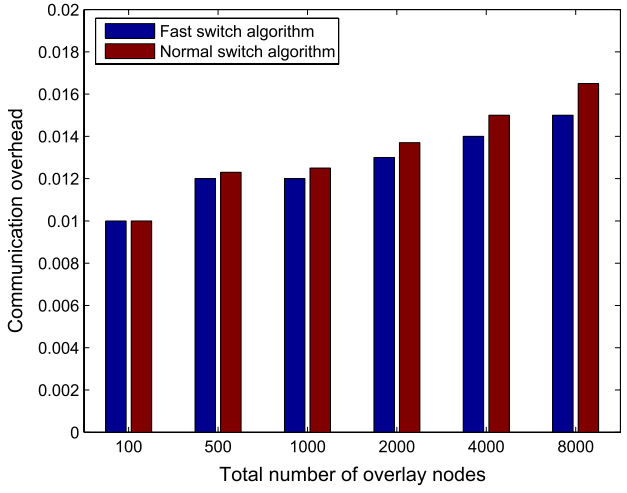

Fig. 10. Communication overhead in static environments.

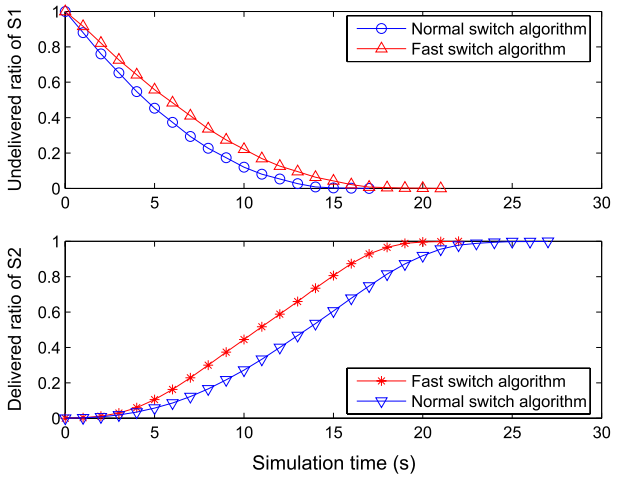

Fig. 11. Ratio track in a dynamic network with 1000 nodes.

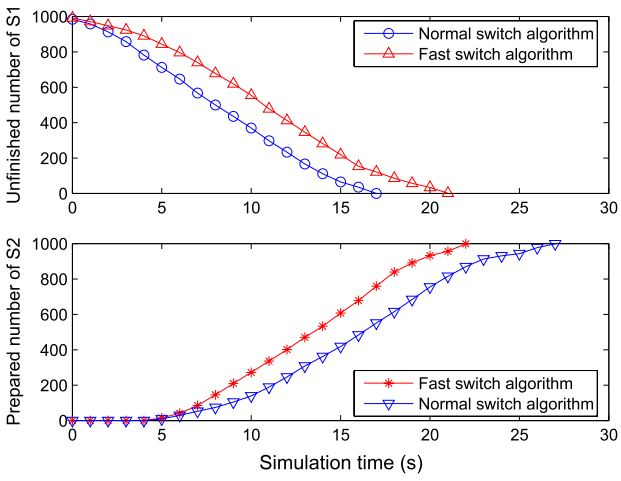

Fig. 12. Number track in a dynamic network with 1000 nodes.

data segment is available while "0" indicates that the data segment is unavailable. The id of the first segment in the buffer is indicated by 20 bits because the source will disseminate at most $10 \times 3600 \times 24 = 864\,000 \in (2^{19}, 2^{20})$ data segments per day (one hour is 3600 s, and one day is 24 h). Therefore, getting the buffer information of one neighbor takes 620 bits' communication cost in total. Every data segment contains 30 Kb data of streaming. If every node can get $p = 10$ required data segments from its neighbors per second, i.e. the data delivery rate just matches the media play rate, then the communication overhead is about $\frac{620\times M}{30\times 1024\times 10} = \frac{5}{495} \approx 1\%$. But in the real applications, like simulation results shown in Fig. 10, the communication overhead is a little larger than 1% because data delivery rate of most nodes may not exactly catch the media display rate. The communication overhead of the fast switch algorithm is a bit lower than that of the normal switch algorithm because the fast switch algorithm actually increases the bandwidth utilization.

5.4. Simulation results in dynamic environments

To simulate a dynamic network environment, we randomly choose 5% old nodes leave and 5% new nodes join per scheduling period. We choose the number "5%" because it represents a typical high churn rate in our simulations. When the churn rate is larger than 5%, in the simulations we observe the overlay network has difficulty in repairing itself and thus has a possibility of network partition. When the churn rate is much smaller than 5%, the simulation results in dynamic environments much resemble those in static environments, indicating little difference. A new joined node does not need to retrieve all the disseminated data segments from each source, and it just requests the data segments being played or will be played by its neighbors. That is to say, a new joined node starts its media playback by following its neighbors' current steps.

In general, simulation results in dynamic environments are consistent with the results in static environments. Fig. 11 tracks the undelivered ratio of S_1 and delivered ratio of S_2 of our fast switch algorithm and the normal switch algorithm in a dynamic network environment with 1000 nodes. Fig. 12 tracks the unfinished number of S_1 and prepared number of S_2. By using the normal switch algorithm, the last node finishes S_1 at time 17 but prepares S_2 at time 27. By using the fast switch algorithm, the last node finishes S_1 at time 21 and prepares S_2 at time 22. The above-mentioned two figures show that in dynamic environments the fast switch algorithm still accelerates the source switch process.

We also examine the average finishing time of S_1 and average preparing time of S_2 of overlay networks with different sizes working in dynamic network environments. The results in Fig. 13 indicate that in dynamic environments the fast switch algorithm also splits the difference between the average finishing time of S_1 and preparing time of S_2 of the normal switch algorithm, and thus makes the startup delay of the new source shorter. From Fig. 14 we can see the reduction ratio in dynamic environments lies between 0.18 and 0.28, which is similar to the reduction ratio in static environments.

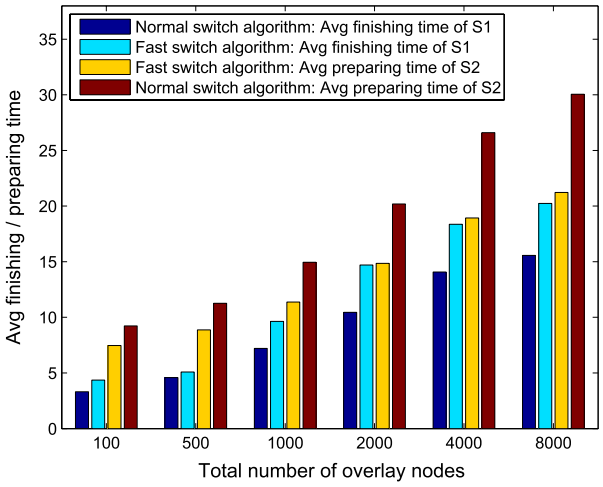

Fig. 13. Avg finishing time of S_1 and preparing time of S_2 in dynamic environments.

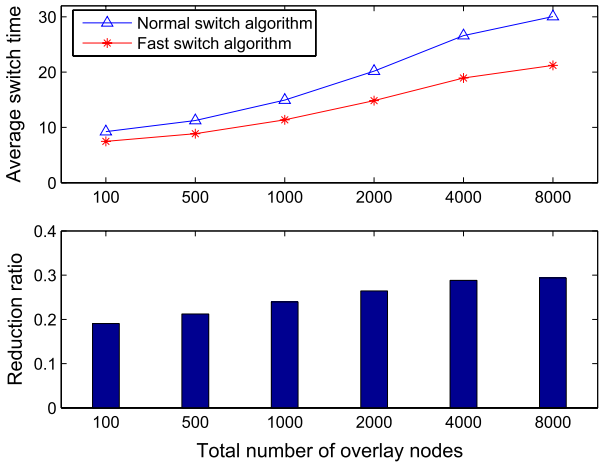

Fig. 14. Avg switch time and its reduction ratio in dynamic environments.

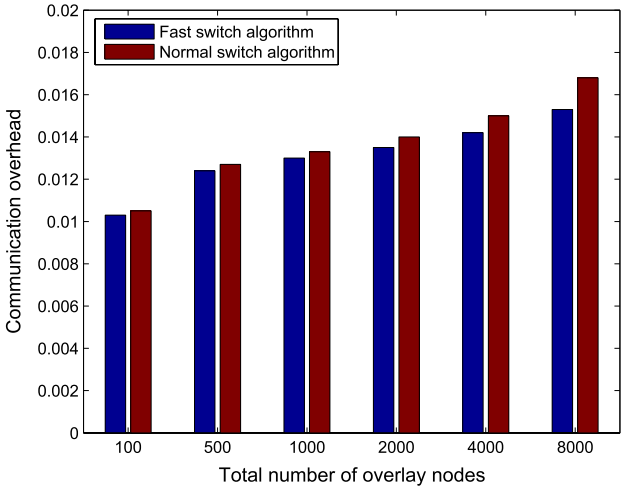

Fig. 15. Communication overhead in dynamic environments.

Finally, the communication overhead of the two algorithms is depicted in Fig. 15, which is still consistent with the communication overhead in static environments.

6. Related work

In general, two basic performance metrics, bandwidth and delay, must be carefully considered when we design a network system. It is also the case when designing a P2P streaming system. In this section we overview the existing methods that aim at reducing the *delay* of P2P streaming, including the source-to-end delay, end-to-end delay, client startup delay, data buffering delay and the source switch delay.

- *Data scheduling method:* The data scheduling algorithm is a kernel function in mesh-based P2P streaming systems. The CoolStreaming [32] system designs a "rarest-first" data scheduling policy to assign data segments which own fewer suppliers with higher priorities to retrieve. This policy can reduce the probability of waiting for a rare data segment and thus lessen the source-to-end delay. Xu et al. [31] consider the problem of media data assignment for a multi-supplier P2P streaming session. Given a requesting peer and a set of supplying peers with heterogeneous outbound rates, their algorithm, named OTS_{p2p}, computes optimal media data assignments for P2P streaming sessions to achieve minimum data buffering delay and thus to reduce the client startup delay. But OTS_{p2p} has very strict assumptions that can hardly hold in a real P2P streaming system. The fast source switch algorithm we proposed in this paper belongs to the data scheduling method.
- *Pull–push method:* Zhang et al. [33] observe that *pure-pull* method in P2P streaming brings tremendous latency and thus propose a hybrid *push–pull* system called GridMedia. They classify the streaming packets into pulling packets and pushing packets. A pulling packet is delivered by a neighbor only when the packet is requested, while a pushing packet is relayed by a neighbor as soon as it is received. The main goal of GridMedia is to reduce the source-to-end delay with the cost of extra "push" communication overhead. The hybrid push and pull scheme was also adopted by the new CoolStreaming system [12] (As a contrast, the old CoolStreaming system [32] utilizes a pure-pull data scheduling scheme). In the new CoolStreaming system, when a node subscribes to a sub-stream by connecting to one of its partners via a single request (pull), the requested partner, i.e., the parent node, will continue pushing all data blocks in need of the sub-stream to the requested node.
- *Inter-overlay optimization:* The P2P live streaming system AnySee [11] employs an "inter-overlay optimization" mechanism to reduce the source-to-end delay. In the AnySee system, each node maintains one active streaming path set and one backup streaming path set. Initially all streaming paths are managed by the overlay manager. When the number of backup streaming paths is less than a threshold, the inter-overlay optimization algorithm is called to find appropriate streaming paths in the global P2P network with the help of the mesh-based overlay. When one active streaming path is cut off due to its poor QoS or peer's leaving, a new streaming path is selected from the backup set.
- *Reducing the minimum longest path:* Ren et al. study how to optimize the source-to-end delay while meeting a certain streaming rate requirement from the fact that the peer delay in a mesh depends on its longest path through its parents to the source [22]. They formulate the minimum delay mesh problem which is NP-hard and propose a distributed algorithm which makes continuous improvement on delay until some minimum delay is reached.
- *Reducing the topology mismatch:* The construction of the peer overlay in existing P2P systems has not considered the underlying physical network topology and can cause serious topology mismatch between the P2P overlay network and the physical network. The topology mismatch problem brings great link stress (unnecessary traffic) in the Internet infrastructure and greatly degrades the system performance. Tu et al. address this problem and propose a locality-aware P2P overlay construction method called Nearcast [25], which builds an efficient overlay multicast tree by letting each peer node choose physically closer nodes as its logical children.
- *Enhancing the AS awareness:* Through intensive Internet measurements based on Border Gateway Protocol (BGP) routing

tables and other dynamic information, Ren et al. observe that the peer selections in the Skype system do not take Autonomous System (AS) topology into consideration and thus many relay peer selections are suboptimal. Motivated by their measurements and analysis, they propose an AS-aware peer-relay protocol called ASAP [21], which can efficiently avoid or mitigate the inter-AS latency, so as to reduce the end-to-end delay of VoIP streaming. Besides, the P4P platform [30] provides portable plugins for P2P systems to detect the AS and ISP information. Its effect has been checked in Liveswarms [17], a BitTorrent-based P2P live streaming system.

7. Conclusion

This paper studied the source switching problem in P2P streaming systems and proposed an efficient solution to minimize the delay in switching between alternate sources. We model the source switch process, formulate it into an optimization problem, and derive its theoretical optimal solution. Considering the dynamics and heterogeneity of real Internet environments, we propose a practical greedy algorithm that approximates the optimal solution by properly interleaving the data delivery of the old source and the new source. Comprehensive simulations on various real-traced overlay topologies confirm the effectiveness of our fast source switch algorithm.

Acknowledgments

The work is partly supported by China NSF grants (60873051, 60721002, 60825205), China 973 project (2006CB303000), and Hong Kong RGC under the CERG grant PolyU 5103/06E.

References

[1] L. Adamic, R. Lukose, A. Puniyani, B. Huberman, Search in power-law networks, The American Physical Society 64 (2001) 46135–46143.
[2] E. Adar, B. Huberman, Free riding on gnutella, 2000.
[3] S. Asaduzzaman, Y. Qiao, G. Bochmann, CliqueStream: An efficient and fault-resilient live streaming network on a clustered peer-to-peer overlay, in: Proceedings of the Eighth International Conference on Peer-to-Peer Computing, P2P'08, 2008, pp. 269–278.
[4] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable application layer multicast, in: Proceedings of the 2002 SIGCOMM Conference, vol. 32, ACM Press, New York, NY, USA, 2002, pp. 205–217.
[5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, A. Singh, SplitStream: High-bandwidth multicast in cooperative environments, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP, 2003, pp. 298–313.
[6] M. Castro, P. Druschel, A. Kermarrec, A. Rowstron, Scribe: A large-scale and decentralized application-level multicast infrastructure, IEEE Journal on Selected Areas in Communications 20 (8) (2002) 1489–1499.
[7] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, USA, 1990.
[8] A. Ganesh, A. Kermarrec, L. Massoulie, Peer-to-peer membership management for gossip-based protocols, IEEE Transactions on Computers 52 (2) (2003) 139–149.
[9] Joost: www.joost.com.
[10] J. Li, PeerStreaming: An on-demand peer-to-peer media streaming solution based on a receiver-driven streaming protocol, in: IEEE 7th Workshop on Multimedia Signal Processing, 2005, 2005, pp. 1–4.
[11] X. Liao, H. Jin, Y. Liu, L. Ni, D. Deng, AnySee: Peer-to-peer live streaming, in: Proceedings of the 25th IEEE International Conference on Computer Communications, IEEE INFOCOM, 2006, pp. 1–10.
[12] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, X. Zhang, Inside the new coolstreaming: Principles, measurements and performance implications, in: Proceedings of the 27th IEEE International Conference on Computer Communications, IEEE INFOCOM, 2008, pp. 14–17.
[13] Y. Liu, H. Yung, Human-centered multimedia E-learning system for real-time interactive distance education, in: Proceedings of the IEEE International Conference on Multimedia and Exposition, ICME, 2007, pp. 2042–2045.
[14] N. Magharei, R. Rejaie, PRIME: Peer-to-peer receiver-driven mesh-based streaming, in: Proceedings of the 26th IEEE International Conference on Computer Communications, IEEE INFOCOM, 2007.
[15] V. Padmanabhan, H. Wang, P. Chou, Resilient peer-to-peer streaming, in: Proceedings of the Eleventh IEEE International Conference on Network Protocols, ICNP, 2003, pp. 16–27.
[16] K. Park, S. Pack, T. Kwon, Climber: An incentive-based resilient peer-to-peer system for live streaming services, in: Proceedings of the International Workshop on Peer-to-Peer Systems, IPTPS, 2006.
[17] M. Piatek, C. Dixon, A. Krishnamurthy, T. Anderson, LiveSwarms: Adapting BitTorrent for end host multicast, tech. rep., Technical Report UW-CSE-06-11-01, 2006.
[18] PPLive: www.pplive.com.
[19] PPStream: www.ppstream.com.
[20] R. Rejaie, A. Ortega, PALS: Peer-to-peer adaptive layered streaming, in: Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV, 2003, pp. 153–161.
[21] S. Ren, L. Guo, X. Zhang, ASAP: An AS-aware peer-relay protocol for high quality VoIP, in: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, ICDCS, IEEE Computer Society Washington, DC, USA, 2006, pp. 70–79.
[22] D. Ren, Y. Li, S. Chan, On reducing mesh delay for peer-to-peer live streaming, in: Proceedings of the 27th IEEE International Conference on Computer Communications, IEEE INFOCOM, 2008, pp. 1058–1066.
[23] Skype: www.skype.com.
[24] D. Tran, K. Hua, T. Do, ZIGZAG: An efficient peer-to-peer scheme for media streaming, in: Proceedings of the 22th IEEE International Conference on Computer Communications, IEEE INFOCOM, 2003, pp. 1283–1292.
[25] X. Tu, H. Jin, X. Liao, J. Cao, Nearcast: A locality-aware P2P live streaming approach for distance education, ACM Transactions on Internet Technology 8 (2) (2008) 1–23.
[26] UUCall: www.uucall.com.
[27] UUSee: www.uusee.com.
[28] V. Venkataraman, P. Francis, J. Calandrino, Chunkyspread: Multi-tree unstructured peer-to-peer multicast, in: Proceedings of the International Workshop on Peer-to-Peer Systems, IPTPS, 2006.
[29] F. Wang, Y. Xiong, J. Liu, B. Burnaby, C. Beijing, mTreebone: A hybrid tree/mesh overlay for application-layer live video multicast, in: Proceedings of the 27nd International Conference on Distributed Computing Systems, ICDCS, 2007.
[30] H. Xie, Y. Yang, A. Krishnamurthy, Y. Liu, A. Silberschatz, P4P: Provider portal for applications, in: Proceedings of the ACM SIGCOMM conference, 2008.
[31] D. Xu, M. Hefeeda, S. Hambrusch, B. Bhargava, On peer-to-peer media streaming, in: Proceedings of the 22nd International Conference on Distributed Computing Systems, ICDCS, 2002, pp. 363–371.
[32] X. Zhang, J. Liu, B. Li, T. Yum, CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming, in: Proceedings of the 24th IEEE International Conference on Computer Communications, IEEE INFOCOM, 2005, pp. 2102–2111.
[33] M. Zhang, J. Luo, L. Zhao, S. Yang, A peer-to-peer network for live media streaming using a push–pull approach, in: Proceedings of the 13th Annual ACM International Conference on Multimedia, 2005, pp. 287–290.
[34] S. Zhuang, B. Zhao, A. Joseph, R. Katz, J. Kubiatowicz, Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination, ACM Press, New York, NY, USA, 2001.

Zhenhua Li received the B.Sc. and the M.Sc. degree in Computer Science from Nanjing University, Nanjing, China. He is currently pursuing his Ph.D. degree in the department of computer science and technology, Peking University, Beijing, China. His research interests include peer-to-peer systems, social network systems, cloud computing and so on. He has published one book and over 10 technical papers in the above areas. He is a member of China Computer Federation.

Jiannong Cao received the B.Sc. degree in computer science from Nanjing University, Nanjing, China, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, Pullman, WA, USA.

He is currently a professor in the Department of Computing at Hong Kong Polytechnic University, Hung Hom, Hong Kong. He is also the director of the Internet and Mobile Computing Lab in the department. Before joining Hong Kong Polytechnic University, he was on the faculty of computer science at James Cook University and University of Adelaide in Australia, and City University of Hong Kong. His research interests include mobile and pervasive computing, computer networking, parallel and distributed computing, and fault tolerance. He has published over 250 technical papers in the above areas. His recent research has been focused on wireless networks and mobile and pervasive computing systems, developing test-bed, protocols, middleware and applications.

Dr. Cao is a senior member of China Computer Federation, a senior member of the IEEE, IEEE Computer Society, and the IEEE Communication Society, and a member of ACM. He is the Coordinator of the Technical Committee on Distributed Computing (TPDC) of IEEE Computer Society in Asia. He is also a member of the IEEE Technical Committee on Distributed Processing, IEEE Technical Committee on Parallel Processing and IEEE Technical Committee on Fault Tolerant Computing. He has served as an associate editor and a member of editorial boards of

several international journals, a reviewer for international journals/conference proceedings, and also as a chair and member of organizing/program committees for many international conferences.

Guihai Chen obtained his B.S. degree from Nanjing University, M. Engineering from Southeast University, and Ph.D. from the University of Hong Kong. He visited the Kyushu Institute of Technology, Japan in 1998 as a research fellow, and the University of Queensland, Australia in 2000 as a visiting professor. From September 2001 to August 2003, he was a visiting professor at Wayne State University. He is now a distinguished professor in the department of computer science and engineering, Shanghai Jiaotong University, China. Prof. Chen has published more than 150 papers in peer-reviewed journals and refereed conference proceedings in the areas of wireless sensor networks, high-performance computer architecture, peer-to-peer computing and performance evaluation. He has also served on technical program committees of numerous international conferences. He is a member of the IEEE Computer Society.

Yan Liu obtained her B.Eng. degree in Electronic Engineering from Southeast University and a M.Sc. in Business School from Nanjing University, China. She obtained her Ph.D. degree in Computer Science from Columbia University, USA. She is currently an assistant professor in the Department of Computing at Hong Kong Polytechnic University. Her research interests include multimedia understanding, video transmission, machine learning and pattern recognition. Her recent research has been focused on how to develop machine learning methods in multimedia data management, more specifically in the context of feature selection in video classification and semantic video retrieval.