Media Streaming on P2P Networks with Bio-inspired Cache Replacement Algorithm

Masahiro Sasabe[1], Naoki Wakamiya[1], Masayuki Murata[2], and Hideo Miyahara[1]

[1] Graduate School of Information Science and Technology, Osaka University
Toyonaka, Osaka 560-8531, Japan
{m-sasabe, wakamiya, miyahara}@ist.osaka-u.ac.jp
[2] Cybermedia Center, Osaka University
Toyonaka, Osaka 560-0043, Japan
murata@cmc.osaka-u.ac.jp

Abstract. With the growth of computing power and the proliferation of broadband access to the Internet, the use of media streaming has become widely diffused. By using the P2P communication architecture, media streaming can be expected to smoothly react to changes in network conditions and user demands for media streams. To achieve continuous and scalable media streaming, we proposed scalable media search and retrieval methods in our previous work. However, through several simulation experiments, we have shown that an LRU (Least Recently Used) cache replacement algorithm cannot provide users with continuous media play-out for unpopular media streams. In this paper, inspired by biological systems, we propose a new algorithm that considers the balance between supply and demand for media streams. Through several simulation experiments, it has been shown that our proposed algorithm could improve the completeness of media play-out compared with LRU.

1 Introduction

With the growth of computing power and the proliferation of broadband access to the Internet, such as ADSL and FTTH, the use of media streaming has become widely diffused. A user receives a media stream from an original media server through the Internet and plays it out on his/her client system as it gradually arrives. However, on the current Internet, the major transport mechanism is still only the best effort service, which offers no guarantee on bandwidth, delay, and packet loss probability. Consequently, such a media streaming system cannot provide users with media streams in a continuous way. As a result, the perceived quality of media streams played out at the client system cannot satisfy the user's demand because he or she experiences freezes, flickers, and long pauses.

The proxy mechanism widely used in WWW systems offers low-delay delivery of data by means of a "proxy server," which is located near clients. The proxy server deposits multimedia data that have passed through it in its local buffer, called the "cached buffer." Then it provides cached data to users on demand in

place of the original content server. By applying the proxy mechanism to streaming services, we believe that high-quality and low-delay streaming services can be accomplished without introducing extra load on the system [1]. However, the media servers and proxy servers are statically located in the network. Users distant from those servers are still forced to retrieve a media stream over a long and unreliable connection to a server. If user demands on media and their locations in the network are known in advance, those servers can be deployed in appropriate locations. However, they cannot flexibly react to dynamic system changes, such as user movements and user demands for media streams. Furthermore, as the number of users increases, load concentration at the servers is unavoidable.

Peer-to-Peer (P2P) is a new network paradigm designed to solve these problems. In a P2P network, hosts, called peers, directly communicate with each other and exchange information without the mediation of servers. One typical example of P2P applications is a file-sharing system, such as Napster and Gnutella. Napster is one of the hybrid P2P applications. A consumer peer finds a desired file by sending an inquiry to a server that maintains the file information of peers. On the other hand, Gnutella is one of the pure P2P applications. Since there is no server, a consumer peer broadcasts a query message over the network to find a file. If a peer successfully finds a file, it retrieves the file directly from a peer holding the file. Thus, concentration of load on a specific point of the network can be avoided if files are well distributed in a P2P network. In addition, by selecting a peer nearby from a set of file holders, a peer can retrieve a file faster than a conventional client-server based file sharing.

By using the P2P communication technique, media streaming can be expected to flexibly react to network conditions. There have been several research works on P2P media streaming [2-7]. Most of them have constructed an application-level multicast tree whose root is an original media server while the peers are intermediate nodes and leaves. Their schemes were designed for use in live broadcasting. Thus, they are effective when user demands are simultaneous and concentrated on a specific media stream. However, when demands arise intermittently and peers request a variety of media streams, as in on-demand media streaming services, an efficient distribution tree cannot be constructed. Furthermore, the root of the tree, that is, a media server, is a single point of failure because such systems are based on the client-server architecture.

We have proposed several methods for on-demand media streaming on pure P2P networks where there is no server [8]. There are several issues to resolve in accomplishing effective media streaming on pure P2P networks. Scalability is the most important among them. Since there is no server that manages information on peer and media locations, a peer has to find the desired media stream by itself by emitting a query message into the network. Other peers in the network reply to the query with a response message and relay the query to the neighboring peers. Flooding, in which a peer relays a query to every neighboring peer, is a powerful scheme for finding a desired media stream in a P2P network. However, it has been pointed out that the flooding scheme lacks scalability because the number of queries that a peer receives significantly increases with the growth in

the number of peers [9]. Especially when a media stream is divided into blocks for efficient use of network bandwidth and cache buffer [10-13], a block-by-block search by flooding queries apparently introduces much load on the network and causes congestion. To tackle this problem, we proposed two scalable per-block search methods. Taking into account the temporal order of reference to video blocks, a peer sends a query message for a group of consecutive blocks. The range of search is dynamically regulated based on the preceding search result using two different algorithms.

Another problem in P2P streaming is continuity of media play-out. In media streaming services, continuous media play-out is the most important factor for users. To accomplish the continuity of media play-out, we have to consider a deadline of retrieval for each block. To retrieve a block by its corresponding play-out time, we have proposed methods to determine an appropriate provider peer (i.e., a peer having a cached block) from search results by taking into account the network conditions, such as the available bandwidth and the transfer delay. By retrieving a block as fast as possible, the remaining time can be used to retrieve the succeeding blocks from distant peers.

Through several simulation experiments, we have shown that our mechanisms can accomplish continuous media play-out for popular media streams without introducing extra load on the system. However, we also have found that the completeness of media play-out deteriorates as the media popularity decreases. The main reason for the deterioration is the cache replacement algorithm. In our media streaming system, a peer stores retrieved media data into its cache buffer. If there is no room to store the media data, the peer has to perform a replacement on cached media data with the newly retrieved media data. Although LRU is a simple and widely used cache replacement algorithm, it has been proved to fail in continuous media play-out [8]. The reason is that popular media streams are cached excessively while unpopular media streams eventually disappear from the network. To improve the continuity of media play-out, in this paper, we propose an effective cache replacement algorithm that considers the supply and demand for media streams. Since there is no server, a peer has to conjecture the behavior of other peers by itself. We expect that each peer can assign an appropriate media stream to be replaced based on its local information and, as a result, that each media stream can be cached according to its corresponding popularity in the network. Thus, our proposed media streaming is one of distributed systems constructed by peers.

In biology, social insects, such as ants, also construct a distributed system [14]. In spite of the simplicity of their individuals, the social insect society presents a highly structured organization. As a result, the organization can accomplish complex tasks that in some cases far exceed the individual capacities of a single insect. It has been pointed out that the study of social insect societies' behaviors and their self-organizing capacities is interesting for computer scientists because it provides models of distributed organization that are useful to solve difficult optimization and distributed control problems. Therefore, we can expect that a bio-inspired mechanism would be applicable to our media-streaming

system by regarding the peer as insect. In particular, a recently proposed model of division of labor in a colony of primitively eusocial wasps, based on a simple reinforcement of response thresholds, can be transformed into a decentralized adaptive algorithm of task allocation [15].

In the model of the division of labor, the ratio of individuals that perform a task is adjusted in a fully-distributed and self-organizing manner. The demand to perform a task increases as time passes and decreases when it is performed. The probability that an individual i performs a task is given by the demand, i.e., stimulus s, and the response threshold θ_i as $\frac{s^2}{s^2+\theta_i^2}$, for example. When individual i performs the task, the threshold to the task is decreased and thus this individual tends to devote itself to the task. After performing the task several times, it becomes a specialist of the task. Otherwise, the threshold is increased. Through threshold adaptation without direct interactions among individuals, the ratio of individuals that perform a specific task is eventually adjusted to some appropriate level. As a result, there would be two distinct groups that show different behaviors toward the task, i.e., one performing the task and the other hesitating to perform the task. When individuals performing the task are withdrawn, the associated demand increases and so does the intensity of the stimulus. Eventually, the stimulus reaches the response thresholds of the individuals of the other group, i.e., those that are not specialized for that task. Some of these individuals are stimulated to perform the task, their thresholds decrease, and finally they become specialized for the task. Finally, the ratio of individuals with regard to task allocations reaches the appropriate level again.

By regarding the replacement of media streams as a task, we propose a bio-inspired cache replacement algorithm based on the division of labor and task allocation. We employ the ratio of supply to demand for a media stream as stimulus. Since each peer relays the query and response messages to its neighboring peers, it can passively obtain information on supply and demand without introducing extra signaling traffic on the system. It estimates the demand for a media stream from the number of queries for the media stream received from other peers and the supply for a media stream from the number of them included in the response messages. Then, based on the stimulus, it assigns a media stream to be replaced in a probabilistic way. The selected media stream is replaced in a block-by-block basis from the end of the media stream. Since the threshold of the victim is decreased, a media stream tends to be discarded often and sequentially once it is chosen as a victim. Although a deterministic approach can also be applicable to this problem, it can not perform effectively without complex parameter settings. On the other hand, our proposed cache replacement algorithm is parameter-insensitive since each peer dynamically changes the response threshold in accordance with the information obtained from the network environment.

Through several simulation experiments, we evaluated the proposed cache replacement algorithm in terms of the completeness of media play-out and the insensitivity to parameter setting.

The rest of the paper is organized as follows. In Section 2, we discuss our media streaming on P2P networks. We give an overview of our streaming system on P2P networks, that is, per-group based search and retrieval of media streams in Subsection 2.1. Then, we introduce search and retrieval methods to accomplish scalable and continuous media streaming in Subsections 2.2 and 2.3. Furthermore, we propose a bio-inspired cache replacement algorithm in Subsection 2.4. Next, in Section 3, we evaluate our proposed cache replacement algorithm through several simulation experiments. Finally, we conclude the paper and describe future works in Section 4.

2 Media Streaming on P2P Networks

A peer participating in our system first joins a logical P2P network for the media streaming. For efficient use of network bandwidth and cache buffer, a media stream is divided into blocks. A peer searches, retrieves, and stores a media stream in a block-by-block basis. In this section, we introduce scalable search methods to find desired blocks and algorithms to determine an optimum provider peer from the search results. Finally, a bio-inspired cache replacement algorithm that takes into account the balance between supply and demand for media streams is given.

2.1 Per-group Based Block Search

In our system, a peer retrieves a media stream and plays it out in a block-by-block basis. However, a block-by-block search apparently increases the number of queries that are transferred on the network and causes network congestion. To tackle this problem, taking into account the temporal order of reference in a media stream, our mechanism employs a per-group search to accomplish a scalable media search based on the number of peers.

A peer sends out a query message for every N consecutive blocks, called a round. Figure 1 illustrates an example of $N = 4$. P_A, P_B, P_C, and P_D, which indicate peers within the range of the propagation of query messages. Numbers in parentheses next to peers stand for identifiers of the blocks that a peer has. At time $T_s(1)$, a query message for blocks from 1 to 4 is sent out from P to the closest peer P_A. Since P_A has the second block out of four requested blocks, it returns a response message. It also relays the query to the neighboring peers P_B and P_C. P_B also replies with a response message to P. Since P_C does not have any of the four blocks, it only relays the query to P_D. Finally, P_D sends back a response message. P determines a provider peer for each block in the round from the search results obtained by the query. It takes two RTT (Round Trip Time) from the beginning of the search to the start of reception of the first block of the round. To accomplish continuous media play-out, P sends a query for the next round at a time that is $2RTT_{worst}$ earlier than the start time of the next round. RTT_{worst} is the RTT to the most distant peer among the peers that returned response messages in the current round.

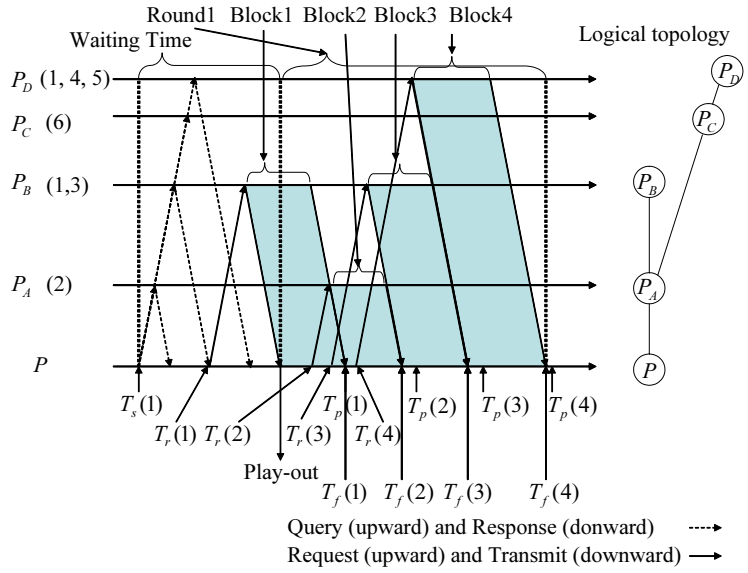

Fig. 1. Example of scheduling for search and retrieval

2.2 Block Search Methods Based on Search Results

Since each peer retrieves a media stream sequentially from the beginning to the end, we can expect that a peer that sent back a responses message for the current round has some blocks of the next round. In our mechanisms, a peer tries flooding at the first round. However, in the following rounds, it searches blocks in a scalable way based on the search results of the previous round.

A query message consists of a query identifier, a media identifier, and a pair of block identifiers to specify the range of blocks needed, i.e., $(1, N)$, a time stamp, and TTL (Time To Live). A peer that has any blocks in the specified range sends back a response message. A response message reaches the querying peer through the same path, but in the reversed direction, that the query message traversed. The response message contains a list of all cached blocks, TTL values stored in the received query, and sum of the time stamp in the query and processing time of the query. Each entry of the block list consists of a media identifier, a block number, and block size. If TTL is zero, the query message is discarded. Otherwise, after decreasing the TTL by one, the query message is relayed to neighboring peers except for the one from which it received the query. In the case of Gnutella, a fixed TTL of seven is used. By regulating TTL, the load of finding a file can be reduced. We have called the flooding scheme with a fixed TTL of 7, which is used in Gnutella, full flooding, and that with a limited TTL based on the search results, limited flooding.

In limited flooding, for the kth round, a peer obtains a set R of peers based on response messages obtained at round $k-1$. A peer in R is expected to have

at least one of the blocks from $kN+1$ to $(k+1)N$. Since time has passed from the search at round $k-1$, some blocks listed in a response message may be already replaced by other blocks. Assuming that a peer is watching a media stream without interactions such as rewinding, pausing, and fast-forwarding, and that the cache buffer is filled with blocks, the number of blocks removed can be estimated by dividing the elapsed time from the generation of the response message by one block time B_t. We should note here that we do not take into account blocks cached after a response message is generated. In limited flooding, TTL is set to that of the most distant peer among the peers in R.

To attain an even more efficient search, we have also proposed another search scheme called selective search. The purpose of flooding schemes is to find peers that do not have any blocks of the current round but do have some blocks of the next round. Flooding also finds peers that have newly joined our system. However, in flooding, the number of queries relayed on the network exponentially increases according to the TTL and the number of neighboring peers. As a simple example, when a query is given a TTL whose value is H and a peer knows D other peers, the total number of query messages relayed on the network becomes $\sum_{i=1}^{H}(D-1)^i = O((D-1)^{H+1})$. Therefore, when a sufficient number of peers are expected to have blocks in the next round, it is effective for a peer to directly send queries to those peers. We call this selective search.

By considering the pros and cons of full flooding, limited flooding, and selective search, we have proposed efficient methods based on combining them.

FL method
The FL method is a combination of full flooding and limited flooding. For blocks of the next round, a peer conducts (1) limited flooding if the conjectured contents of cache buffers of peers in R satisfy every block of the next round, or (2) full flooding if one or more blocks cannot be found in the conjectured cache contents of peers in R.

FLS method
The FLS method is a combination of full flooding, limited flooding, and selective search. For the next round's blocks, a peer conducts (1) selective search if the conjectured contents of cache buffers of peers in R contain every block of the next round, (2) limited flooding if any one of the next round's blocks cannot be found in the conjectured cache contents of peers in R, or, finally, (3) full flooding if none of the provider peers it knows is expected to have any block of the next round, i.e., $R = \phi$.

2.3 Block Retrieval Methods Considering the Continuity of Media Play-out

The peer sends a request message for the first block of a media stream as soon as it receives a response message from a peer that has the block. Then, it plays it out immediately when the reception of the block starts. Consequently, the

deadlines for the retrieval of succeeding blocks $j \geq 2$, $T_p(j)$ are determined as follows:

$$T_p(j) = T_p(1) + (j-1)B_t, \tag{1}$$

where $T_p(1)$ corresponds to the time that the peer finishes playing out the first block and B_t stands for one block time.

We do not wait for the completion of the reception of the preceding block before issuing a request for the next block because it introduces an extra delay of at least one RTT, and the cumulative delay affects the timeliness and continuity of media play-out. In our block retrieval mechanism, the peer sends a request message for block j at $T_r(j)$. As a result, the peer can start to receive block j just after finishing the retrieval of block $j-1$, as shown in Fig. 1. Equation (1) guarantees that the completion time of a block retrieval is earlier than that of the block play-out. Furthermore, the retrieval of the next block starts after the completion of the retrieval of the previous block. As a result, our block retrieval mechanism can maintain the continuity of media play-out. By observing the way that the response message is received in regard to the query message, the peer estimates the available bandwidth and the transfer delay from the provider peer. The estimates are updated through reception of media data. For more precise estimation, we can use any other measurement tool as long as it does not disturb media streaming. Every time the peer receives a response message, it derives the estimated completion time of retrieval of block j, that is $T_f(j)$, from the block size and the estimated bandwidth and delay, for each block for which it has not sent a request message yet. Then, it determines an appropriate peer in accordance with deadline $T_p(j)$ and calculates time $T_r(j)$ at which it sends a request.

In the provider determination algorithm, the peer calculates set S_j, a set of peers having block j. Next, based on the estimation of available bandwidth and transfer delay, it derives set $S_j^{'}$, a set of peers from which it can retrieve block j by deadline $T_p(j)$, from S_j. If $S_j^{'} = \phi$, the peer waits for the arrival of the next response message. However, it gives up retrieving and playing block j when the deadline $T_p(j)$ passes without finding any appropriate peer. To achieve continuous media play-out, it is desirable to shorten the block retrieval time. The SF (Select Fastest) method selects a peer whose estimated completion time is the smallest among those in $S_j^{'}$. By retrieving block j as fast as possible, the remainder of $T_p(j) - T_f(j)$ can be used to retrieve the succeeding blocks from distant peers. On the other hand, an unexpected cache miss introduces extra delay on the client system. The SR (Select Reliable) method selects the peer with the lowest possibility of block disappearance among those in $S_j^{'}$. As a result, this suppresses block disappearance before a request for block j arrives at the provider peer.

2.4 Bio-inspired Cache Replacement Algorithm

Since the cache buffer size is limited, there may be no room to store a newly retrieved block into the cache. Although LRU is a simple and widely used scheme,

it has been shown that LRU cannot accomplish continuous media play-out under a heterogeneous media popularity [8]. This is because popular media streams are cached excessively while unpopular media streams eventually disappear from the P2P network.

In this paper, to solve this problem, we propose a bio-inspired cache replacement algorithm that considers the balance between supply and demand for media streams. Since there is no server in a pure P2P network, a peer has to conjecture the behavior of other peers by itself. It is disadvantageous for the peer to aggressively collect information about supply and demand by communicating with other peers, since this brings extra load on the system and deteriorates the system scalability. Therefore, in our scheme, a peer estimates the supply and demand based on locally available and passively obtained information. This information consists of search results it obtained and messages it relayed. We expect that each peer can determine an appropriate media stream to be replaced based on its local information, and, as a result, each media stream can be cached according to the media popularity in the network. Thus, our proposed media streaming is a distributed system constructed by peers. In biology, social insects, such as ants, also form a decentralized system. Furthermore, it has been pointed out that social insects provide us with a powerful metaphor for creating decentralized systems of simple interacting [14]. In particular, a recently proposed model of division of labor in a colony of primitively eusocial wasps, based on a simple reinforcement of response thresholds, can be transformed into a decentralized adaptive algorithm of task allocation [15].

In the model of the division of labor, the ratio of individuals that perform a task is adjusted in a fully-distributed and self-organizing manner. The demand to perform a task increases as time passes and decreases when it is performed. The probability that an individual i performs a task is given by the demand, i.e., stimulus s, and the response threshold θ_i as $\frac{s^2}{s^2+\theta_i^2}$, for example. When the individual i performs the task, the threshold to the task is decreased and thus it tends to devote itself to the task. After performing the task several times, it becomes a specialist of the task. Otherwise, the threshold is increased. Through threshold adaptation without direct interactions among individuals, the ratio of individuals that perform a specific task is eventually adjusted to some appropriate level. As a result, there are two distinct groups that show different behaviors toward the task, i.e., one performing the task and the other hesitating to perform the task. When individuals performing the task are withdrawn, the associated demand increases and so does the intensity of the stimulus. Eventually, the stimulus reaches the response thresholds of the individuals in the other group, i.e., those not specialized for that task. Some individuals are stimulated to perform the task, their thresholds decrease, and finally they become specialized for the task. Finally, the ratio of individuals with regard to task allocations reaches the appropriate level again.

By regarding the replacement of media streams as a task, we propose a cache replacement algorithm based on the division of labor and task allocation model. Compared with LRU, our proposed cache replacement algorithm can flexibly

adapt to the temporal changes of supply and demand for media streams. The proposed algorithm is organized by following two steps.

Step1 Estimate the supply and demand for media streams per round. A peer calculates supply $S(i)$ and demand $D(i)$ for media stream i from search results it received and query and response messages it relayed at the previous round. $S(i)$ is the number of fully cached stream i in response messages that it received and relayed. Here, to avoid the overlap of calculation, $S(i)$ includes only the response messages for media stream i as search results. $D(i)$ is the number of the query messages for media stream i, which the peer emitted by itself. To adapt to the temporal changes of supply and demand for media streams, we use the moving average as follows.

$$\overline{S(i)} = w_s S(i) + (1 - w_s)\overline{S(i)}, \ (0 \leq w_s \leq 1) \tag{2}$$

$$\overline{D(i)} = w_d D(i) + (1 - w_d)\overline{D(i)}, \ (0 \leq w_d \leq 1) \tag{3}$$

Step2 Determine a candidate media stream for replacement. Based on the "division of labor and task allocation", we define probability $P_r(i)$ of replacement of media stream i as follows:

$$P_r(i) = \frac{s^2(i)}{s^2(i) + \theta^2(i)}, \tag{4}$$

where $s(i)$ indicates the ratio of supply to demand for media stream i after replacement of media stream i, that is, $\max\left(\frac{\overline{S(i)}-1}{\overline{D(i)}}, 0\right)$. $s(i)$ means how excessively media stream i exists in the network if it was replaced. Therefore, by discarding a media stream whose $s(i)$ is large, we can expect that the supply and demand becomes the same among streams in a P2P network. A peer does not discard the media stream that it is currently watching. To shorten the waiting time for media play-out, it is better if mostly the former part of a media stream exists. Therefore, a peer replaces a media stream in a block-by-block basis from the end of the media stream. As in [16], the threshold of a media stream that a peer works on is decreased. As a result, a media stream tends to be discarded often and sequentially once it is chosen as a victim.

$$\theta(j) = \begin{cases} \theta(j) - \xi & \text{if } j = i \\ \theta(j) + \varphi & \text{if } j \neq i \end{cases} \tag{5}$$

By sequentially replacing blocks of the same media stream, fragmentation of media streams can be avoided. Since our mechanism replaces a media stream from the end of the media stream, the increase in the fragmentation leads to the disappearance of the latter part of media streams from the network. We expect that controlling the threshold is one way to solve this problem.

3 Simulation Evaluation

We have already evaluated our proposed search and retrieval methods and have shown that the FLS method can accomplish continuous media play-out with a

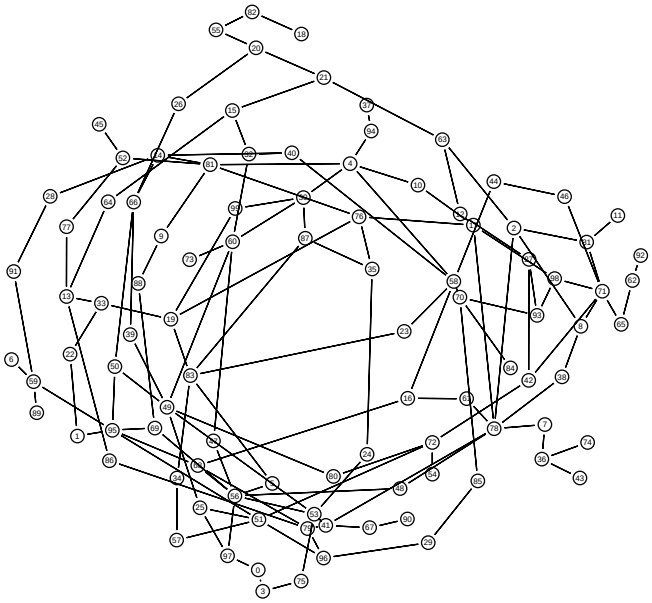

Fig. 2. Random network with 100 peers

smaller amount of search traffic compared with full flooding in [8]. In this section, we conduct simulation experiments to evaluate our proposed cache replacement algorithm in terms of the completeness of media play-out and insensitivity to parameter setting.

3.1 Simulation Model

We use P2P logical networks with 100 peers randomly generated by the Waxman algorithm [17] with parameters $\alpha = 0.15$ and $\beta = 0.3$. An example of generated networks is shown in Fig. 2. The round trip time between two contiguous peers is also determined by the Waxman algorithm and ranges from 10 ms to 660 ms. To investigate the ideal characteristics of our proposed mechanisms, the available bandwidth between two arbitrary peers does not change during a simulation experiment. The bandwidth is given at random between 500 kbps and 600 kbps, which is larger than the media coding rate of CBR 500 kbps.

At first, all 100 peers participate in the system, but no peer watches a media stream. One peer begins to request a media stream at a randomly determined time. The inter-arrival time between two successive requests for the first media stream follows an exponential distribution whose average is 20 minutes. Forty media streams of 60-minute length are available. Media streams are numbered from 1 (the most popular) to 40 (the least popular), where popularity follows a Zipf-like distribution with $\alpha = 1.0$. Therefore, media stream 1 is forty times more popular than media stream 40. Each peer watches a media stream without

interactions such as rewinding, pausing, and fast-forwarding. When a peer finishes watching a media stream, it becomes idle during the waiting time, which also follows an exponential distribution whose average is 20 minutes.

A media stream is divided into blocks of 10-sec duration and amounts to 625 KBytes. Each peer sends a query message for a succession of six blocks, i.e., $N = 6$, and retrieves blocks. Obtained blocks are deposited into a cache buffer of 675 MB, which corresponds to three media streams. In the first run of the simulation, each peer stores three whole media streams in its cache buffer. The population of each media stream in the network also follows a Zipf-like distribution whose parameter α is 1.0. We set the parameters of moving average, w_s and w_d, to 0.9, respectively. Based on the values used in [14], we set the parameters of the cache replacement algorithm as follows: $\xi = 10$, $\varphi = 1$, and the initial value of $\theta(i)$ is set to 500 and $\theta(i)$ changes between 1 and 1000. To prevent the initial condition of the cache buffer from influencing system performance, we only use the results after the initially cached blocks are completely replaced with newly retrieved blocks for all peers.

Since there is almost no difference in simulation results among the six combinations of search methods and block retrieval methods in our experiments, we only show the results of the combination of the FLS and SF methods. We show the average values of 60 set of simulations in the following figures. We define the waiting time as the time between the emission of the first query message for the media stream and the beginning of reception of the first block. Through simulation experiments, we observe that, independent of method, the waiting time decreases as the popularity increases. However, independent of popularity, all media streams successfully found can be played out within 2.6 sec.

3.2 Evaluation of Completeness of Media Play-out

To evaluate the completeness of media play-out, we define completeness as the ratio of the number of retrieved blocks in time to the number of blocks in a media stream. Figures 3 and 4 illustrate the completeness with a 95% confidence interval of each media stream. The horizontal axis indicates the media popularity that decreases with the growth of the media identifier. Comparing Fig. 3 with Fig. 4, we find that our proposed algorithm can reduce the decrease in completeness to the deterioration of the media popularity. As a result, for unpopular media streams, the completeness of the proposed algorithm is higher than that of LRU by 0.2 at most. On the other hand, our proposed algorithm slightly deteriorates the completeness for popular media streams compared with the performance of LRU.

Since a media stream is selected based on the Zipf-like distribution, the completeness of popular media streams is more important than that of unpopular media streams, in terms of the total degree of user satisfaction. Here, we define weighted completeness, which means the completeness weighted with the media popularity, as follows.

$$W(i) = C(i) \times \frac{1}{i}, \tag{6}$$

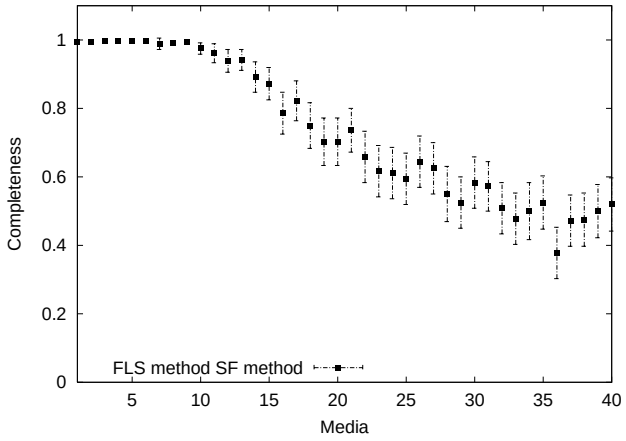

Fig. 3. Completeness (LRU)

Table 1. Degree of user satisfaction

	FLS method
LRU	3.870
Proposed algorithm	3.855

where $C(i)$ is completeness of media stream i. Furthermore, we define the user satisfaction as $\sum_i W(i)$. Table 1 shows the user satisfaction of Figs. 3 and 4. As shown in Tab. 1, there is almost no difference between LRU and the proposed algorithm. Therefore, we can conclude that our proposed algorithm can accomplish high completeness even for unpopular media streams without deteriorating the total of the degree of user satisfaction.

3.3 Evaluation of Insensitivity to Parameter Setting

Parameter setting is a common problem in network control mechanisms. It has been pointed out that it is difficult to select an appropriate parameter statically. To solve this problem, the division of labor and task allocation dynamically changes the response threshold in accordance with the information obtained from the network environment. As a result, it can flexibly adapt to diverse network environments without a specific parameter setting. Figure 5 illustrates the completeness of the proposed cache replacement algorithm with normalized parameters: $\xi = 0.01$, $\varphi = 0.001$. The initial value of $\theta(i)$ is set to 0.5, and $\theta(i)$ changes between 0.001 to 1. Furthermore, to make the weight of $s(i)$ and $\theta(i)$ in Eq. (4) uniform, $s(i)$ is normalized by dividing by $\sum_i s(i)$. Comparing Fig. 4 with Fig. 5, we find that there is almost no difference between them. Therefore,

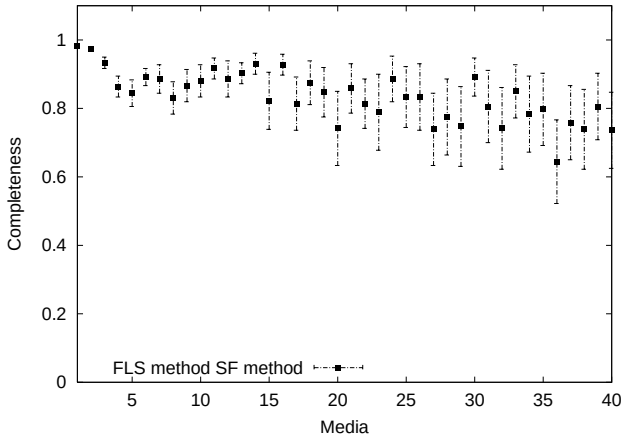

Fig. 4. Completeness (proposed cache replacement algorithm)

we can conclude that our proposed cache replacement algorithm is insensitive to parameter setting.

4 Conclusions

In this paper, to accomplish scalable and continuous media streaming on P2P networks, we introduced two scalable search methods and two algorithms for block retrieval and proposed a bio-inspired cache replacement algorithm that considers the balance between supply and demand for media streams. Through several simulation experiments, we have shown that our proposed cache replacement algorithm can accomplish continuous media play-out independent of media popularity. Furthermore, our simulation results show that proposed cache replacement algorithm is not sensitive to parameter setting.

As future research work, we should evaluate our proposed mechanisms in more realistic situations where network conditions dynamically change and peers randomly join and leave the system.

Acknowledgements

This research was supported in part by "The 21st Century Center of Excellence Program" and Special Coordination Funds for promoting Science and Technology of the Ministry of Education, Culture, Sports, Science and Technology of Japan, and the Telecommunication Advancement Organization of Japan.

References

1. M. Sasabe, Y. Taniguchi, N. Wakamiya, M. Murata, and H. Miyahara, "Proxy caching mechanisms with quality adjustment for video streaming services," *IE-*

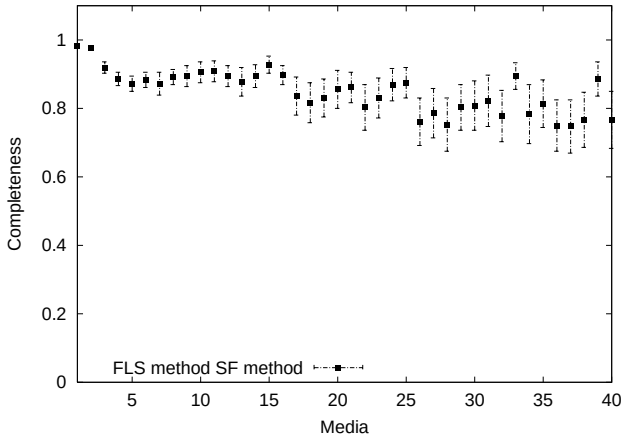

Fig. 5. Completeness (proposed cache replacement algorithm with normalized parameters)

ICE Transactions on Communications Special Issue on Content Delivery Networks, vol. E86-B, pp. 1849–1858, June 2003.
2. AllCast. available at `http://www.allcast.com`.
3. vTrails. available at `http://www.vtrails.com`.
4. Share Cast. available at `http://www.scast.tv`.
5. D. A. Tran, K. A. Hua, and T. T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE INFOCOM2003*, (San Francisco), Mar. 2003.
6. D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *Proceedings of ICDCS2002*, vol. 1, (Vienna), pp. 363–371, July 2002.
7. V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," *Microsoft Research Technical Report MSR-TR-2003-11*, Mar. 2003.
8. M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara, "Scalable and continuous media streaming on peer-to-peer networks," in *Proceedings of P2P2003*, (Linkoping), pp. 92–99, Sept. 2003.
9. R. Schollmeier and G. Schollmeier, "Why peer-to-peer (P2P) does scale: An analysis of P2P traffic patterns," in *Proceedings of P2P2002*, (Linkoping), Sept. 2002.
10. K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of the 10th International WWW Conference*, pp. 36–44, 2001.
11. J. Shudong, B. Azer, and I. Arun, "Accelerating internet streaming media delivery using network-aware partial caching," *Technical Report BUCS-TR-2001-023*, October 2001.
12. B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal proxy cache allocation for efficient streaming media distribution," in *Proceedings of IEEE INFOCOM 2002*, (New York), June 2002.
13. W. Jeon and K. Nahrstedt, "Peer-to-peer multimedia streaming and caching service," in *Proceedings of ICME2002*, (Lausanne), Aug. 2002.
14. E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

15. E. Bonabeau, A. Sobkowski, G. Theraulaz, and J.-L. Deneubourg, "Adaptive task allocation inspired by a model of division of labor in social insects," in *Proceedings of BCEC1997*, (Skovde), pp. 36–45, 1997.
16. M. Campos, E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg, "Dynamic scheduling and division of labor in social insects," *Adaptive Behavior*, vol. 8, no. 2, pp. 83–96, 2000.
17. B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1617–1622, Dec. 1988.