EUROPEAN TRANSACTIONS ON TELECOMMUNICATIONS
Euro. Trans. Telecomms. 2004; **15**:549–558
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/ett.1014

Special Issue on P2P Networking and P2P Services

Effective methods for scalable and continuous media streaming on peer-to-peer networks

Masahiro Sasabe*, Naoki Wakamiya, Masayuki Murata and Hideo Miyahara

Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan

SUMMARY

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 peer-to-peer (P2P) communication architecture, media streaming can be expected to smoothly react to changes in network conditions and user demands for media streams. In this paper, to achieve continuous and scalable media streaming, we introduce our scalable media search and in-time retrieval methods. Through several simulation experiments, we show that our methods can accomplish continuous media play-out for popular media streams without introducing extra load on the system. However, we also show that an LRU cache replacement algorithm cannot provide users with continuous media play-out for unpopular media streams. To tackle this problem, we take inspiration from biological systems to propose a new cache replacement algorithm that considers the balance between supply and demand for media streams. We demonstrate that our proposed algorithm can improve the continuity of media play-out compared with LRU. Furthermore, we find that the proposed algorithm can adapt to changes in the popularity of various media. Copyright © 2004 AEI.

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 progressively arrives. However, with the current internet, the major transport mechanism is still only the best effort service, which offers no guarantees of bandwidth, delay and packet loss probability. Consequently, such a media streaming system cannot provide users with media streams in a dependably continuous way.

By using the peer-to-peer (P2P) communication technique, media streaming can be expected to flexibly react to network conditions. There have been several research works on P2P media streaming [1, 7, 10, 11, 14]. Most of these 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, can be regarded as a critical point of failure because such systems are based on the client–server architecture.

In Reference [8], we proposed scalable search and in-time media retrieval methods for on-demand media streaming on pure P2P networks. In our system, every peer participating in a service watches a media stream and deposits it in its local cache buffer. A media stream is divided into blocks for efficient use of network bandwidth and cache buffer [6, 12]. By retrieving blocks from other

*Correspondence to: Masahiro Sasabe, Advanced Network Environment Division, Cybermedia Center, Osaka University, 1-32 Machikaneyama, Toyonaka, Osaka 560-0043, Japan. E-mail: m-sasabe@cmc.osaka-u.ac.jp

Contract/grant sponsors: Ministry of Education, Culture, Sports, Science and Technology of Japan; Telecommunication Advancement Organization of Japan.

Received 14 July 2004
Revised 22 July 2004
Accepted 22 July 2004

peers in time, a peer can watch a desired media stream. Since there is no server that manages information on peer and media locations, a peer has to find each block constituting a desired media stream 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. If a peer successfully finds a block cached in other peers, it retrieves it from one of them and deposits it in its local cache buffer. If there is no room to store the newly retrieved block, a peer has to perform replacement on cached blocks with it.

There are several issues to resolve in accomplishing effective media streaming over pure P2P networks. Scalability is the most important among them. Flooding, in which a peer relays a query to every neighboring peer, is a powerful scheme for finding a desired media stream. However, it has been pointed out that the flooding lacks scalability because the number of queries that a peer receives significantly increases with the growth in the number of peers [9]. In particular, a block-by-block search by flooding apparently introduces much load on the network and causes congestion. To tackle this problem, we proposed two scalable block search methods. Taking into account the temporal order of reference to media blocks, a peer sends a query message for a group of consecutive blocks. Then, the peer performs adaptive block search by regulating the search range based on the preceding search result.

Since continuous media play-out is the most important factor for users in media streaming services, we have to consider a deadline of retrieval for each block. To retrieve a block by its corresponding play-out time, we 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.

In this paper, we first evaluate the effectiveness of the above-mentioned methods through several simulation experiments. The results show that our methods can accomplish continuous media play-out for popular media streams without introducing extra load on the system. However, we also point out that the continuity of media play-out deteriorates as the media popularity decreases. The reason is that popular media streams are cached excessively while unpopular media streams eventually disappear from the network. Although LRU is a simple and widely used cache replacement algorithm, it fails in continuous media play-out.

To improve the continuity of media play-out, in this paper we consider an effective cache replacement algorithm that takes into account the supply and demand for media streams. Since there is no server, a peer has to make conjectures about the behavior of other peers by itself. A peer estimates the supply and demand from P2P messages that it relays and receives from a flooding-based media search. Then a peer determines a media to discard to make room for a newly retrieved block. Furthermore, a peer also adapts to changes in the supply and demand of media streams. For this purpose, we propose a novel caching algorithm based on the response threshold model of division of labor and task allocation in social insects [3].

In biology, social insects, such as ants, also construct a distributed system [2]. In spite of the simplicity of their individuals, the insect society presents a highly structured organization. It has been pointed out that social insects provide us with a powerful metaphor for creating decentralized systems of simple interacting [2]. 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 [3]. By regarding the replacement of media streams as a task, we propose a fully distributed and autonomous cache replacement algorithm that can adapt to changes in environments, i.e. the supply-to-demand, without parameter tunings. Through several simulation experiments, we evaluate the algorithm in terms of the continuity of media play-out and adaptability to changes in media popularity.

The rest of the paper is organized as follows. In Section [2], we give an overview of our streaming system on P2P networks, describe our per-group based search and retrieval methods, and conduct preliminary simulation experiments. After describing the problem of proposed methods, we propose and evaluate a supply-demand based cache replacement algorithm in Section 3. Finally, we conclude the paper and describe future works in Section 4.

2. SEARCH AND RETRIEVAL METHODS FOR MEDIA STREAMING ON P2P NETWORKS

A peer participating in our system first joins a logical P2P network for the media streaming. Then, a peer sends a query message to find a media stream that it wants to watch. We especially call this query as a media request. 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 on a block-by-block

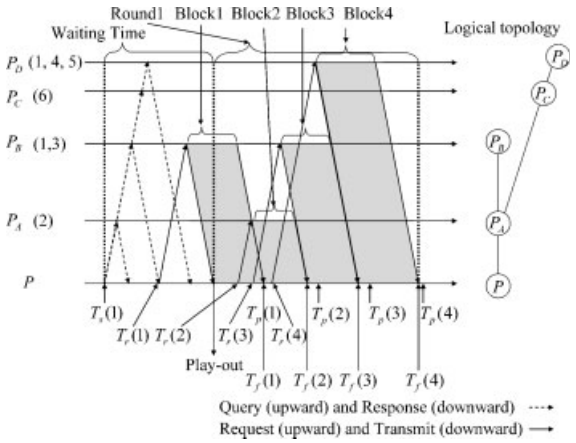

Figure 1. Example of per-group search and retrieval.

basis. In this section, we describe our scalable search methods to find desired blocks and algorithms to determine an appropriate provider peer from the search results.

2.1. Per-group based block search and retrieval

In our system, a peer retrieves a media stream and plays it out in a block-by-block manner. 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 method employs a per-group search to accomplish scalable media search.

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 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 1 to 4 is sent out from P to the closest peer P_A. The query is relayed among peers. Since P_A, P_B and P_D have one or more blocks out of four requested blocks, they return response messages. P determines a provider peer for each block in the round from the search results obtained by the query. It takes two round trip time (RTT) periods 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 $2\mathrm{RTT}_{\mathrm{worst}}$ earlier than the start time of the next round. $\mathrm{RTT}_{\mathrm{worst}}$ is the RTT to the most distant peer among the peers that returned response messages in the current round.

2.2. Scalable block search

Since each peer retrieves a media stream sequentially from the beginning to the end, we can expect that a peer that sent back a response message for the current round has some blocks of the next round. In our methods, 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 time to live (TTL). 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, which 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 this flooding scheme with a fixed TTL of seven '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 $k-1$. R is a set of peers expected to have at least one of the blocks belonging to round k. Since time has passed from the search at round $k-1$, some blocks listed in the response message may have already been 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, we can estimate the number of removed blocks 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 also proposed another search scheme. 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 [9]. 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, there are efficient methods based on combining them.

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 cache contents 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 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 cache contents 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 for continuous media play-out

The peer sends a request message for the first block of a media stream just after receiving a response message from a peer that has the block, because it cannot predict whether any better peer exists at that time. In addition, it is essential for a low-delay and effective media streaming service to begin the media presentation as quickly as possible. Thus, in our method, the peer plays out the first block immediately when its reception starts. Of course, we can also defer the play-out in order to buffer a certain number of blocks in preparation for unexpected delays.

The deadlines for retrieval of succeeding blocks $j \geqslant 2$ are determined as follows:

$$T_p(j) = T_p(1) + (j-1)B_t \qquad (1)$$

where $T_p(1)$ corresponds to the time that the peer finishes playing out the first block.

Although block retrieval should follow a play-out order, the order of request messages does not. We do not wait for completion of reception of the preceding block before issuing a request for the next block because this introduces an extra delay of at least one round-trip, and the cumulative delay affects the timeliness and continuity of media play-out. Instead, the peer sends a request message for block j at $T_r(j)$, which will be given by Equation (3), so that it can start receiving block j just after finishing the retrieval of block $j-1$, as shown in Figure 1. As a result, our block retrieval method can maintain the continuity of media play-out.

The peer estimates the available bandwidth and the transfer delay from the provider peer by using existing measurement tools. For example, by using the inline network measurement technique [5], those estimates can be obtained through exchanging query and response messages without introducing any measurement traffic. Furthermore, the estimates are updated through reception of media data. Every time the peer receives a response message, it derives the estimated completion time of the retrieval of block j, that is $T_f(j)$, from the block size and the estimated bandwidth and delay, for each block to which it has not yet sent a request message. 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. The detailed algorithm to determine the provider peer is given below.

Step 1: Set j to r, which is the maximum block number among blocks that have already been requested.

Step 2: Calculate set S, a set of peers having block j. If $S = \phi$, that is, there is no candidate provider, set $T_f(j) \leftarrow T_p(j)$, $j \leftarrow j+1$ and repeat Step 2 for the next block. Otherwise, proceed to Step 3.

Step 3: Derive set S', a set of peers from which a peer can retrieve block j by deadline $T_p(j)$, from S. Time required to retrieve block j from provider peer i becomes the sum of round trip time $R(i)$ to peer i and the transfer time of block j obtained by dividing block size $B(j)$ by available bandwidth $A(i)$ from peer i. For each peer i in S, the estimated completion time of the retrieval of block j from peer i is derived as $\max(T_f(j-1), T_{\text{now}} + R(i)) + (B(j)/A(i))$, considering the case that the retrieval of block $j-1$ lasts more than $R(i)$ and the request for block j is deferred. Here, T_{now} is the time when this algorithm is performed. If the estimated completion time is earlier than $T_p(j)$, the peer is put in S'. If $S' = \phi$, set $T_f(j) \leftarrow T_p(j)$, $j \leftarrow j+1$ and go back to Step 2.

Step 4: Determine provider peer $P(j)$ of block j from S'. We propose the following two alternative methods for determining the provider peer.

SF (select fastest) method selects a peer whose estimated completion time is smallest among peers in S'. By retrieving block j as fast as possible, the remainder $T_p(j) - T_f(j)$ can be used to

retrieve the succeeding blocks from distant peers or peers with insufficient bandwidth.

SR (select reliable) method selects a peer with the lowest possibility of block disappearance among those in S'. Since the capacity of a cache buffer is limited, block j may be replaced by another block before a request for block j arrives at the provider peer. The list of block identifiers in a response message is in ascending order of referenced time. Thus, a block located closer to the head of the list is likely to be removed in the near future. In SR method, in order to perform reliable retrieval, we consider the peer with a buffer in which block j has the largest number among those of peers in S'.

Step 5: Derive estimated completion time of retrieval $T_f(j)$ and time $T_r(j)$ to send a request message for block j as follows.

$$T_f(j) = \max(T_f(j-1),\ T_{\text{now}} + R(P(j))) + \frac{B(j)}{A(P(j))} \tag{2}$$

$$T_r(j) = T_f(j) - R(P(j)) - \frac{B(j)}{A(P(j))} \tag{3}$$

Step 6: If $j = kN$, finish and wait for receiving the next response message. Here, k is the round number. Otherwise, set $j \leftarrow j + 1$ and go back to Step 2.

A peer emits a request message for block j to peer $P(j)$ at $T_r(j)$ and sets r to j. On receiving the request, peer $P(j)$ initiates block transmission. If it replaces block j with another block since it returned a response message, it informs the peer of a cache miss. When a cache miss occurs, the peer determines another provider peer based on the above algorithm. However, if it has already requested any block after j, it gives up retrieving block j in order to keep the media play-out in order.

After receiving block j, the peer replaces $T_f(j)$ with the actual completion time. In the algorithm, the estimated completion time of retrieval of block j depends on that of block $j-1$, as in Equation (2). Therefore, if the actual completion time $T_f(j)$ of the retrieval of block j changes because of changes of network conditions or estimation errors, the peer applies the algorithm and determines provider peers for succeeding blocks. Our proposed algorithm stated above depends on the accuracy of estimation. One of possible solutions to inaccurate estimates is to introduce some reserved time in Equation (2). In addition, deferment of the play-out also contributes to absorb estimation errors.

2.4. Simulation experiments

We conducted simulation experiments to evaluate the basic characteristics of our proposed methods in terms of the amount of search traffic and the continuity of media play-out.

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

At first, none of the 100 peers watch any media stream. Then, peers randomly begin to request a media stream one by one. The inter-arrival time between two successive media requests for the first media stream among clients follows an exponential distribution whose average is 20 min. Forty media streams of 60 min length are available. Media streams are numbered from 1 (most popular) to 40 (least popular), where the various levels of popularity follow a Zipf-like distribution with $\alpha = 1.0$. Therefore, media stream 1 is 40 times more popular than media stream 40. Each peer watches a media stream without such interactions as rewinding, pausing or fast-forwarding. When a peer finishes watching a media stream, it becomes idle during the waiting time, which also follows a exponential distribution whose average is 20 min. A media

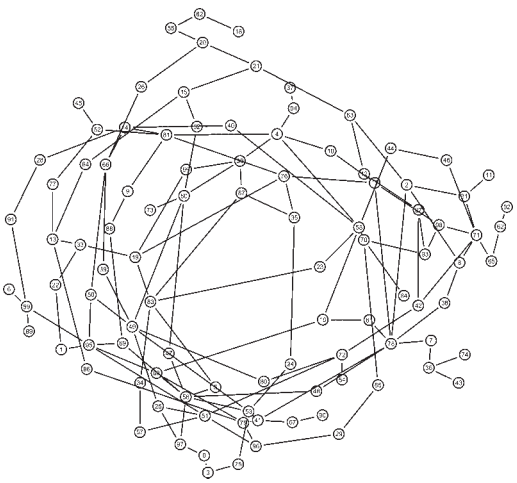

Figure 2. Random network with 100 peers.

stream is divided into blocks of 10 s duration and 625 kbytes. Each peer sends a query message for a succession of six blocks, i.e. $N = 6$, and retrieves blocks. Blocks obtained are deposited into a cache buffer of 675 MB, which corresponds to three media streams. At the beginning of each simulation experiment, each peer stores three whole media streams in its cache buffer. The initial population of each media stream in the network also follows a Zipf-like distribution whose parameter α is 1.0. 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.

We consider six combinations of three search methods, i.e. full flooding only, FL, and FLS, and two retrieval methods, i.e. SF and SR. We conducted 90 set of simulations for each of six methods and show average values in the following figures.

2.4.1. Evaluation of scalability of search methods. First, we evaluate search methods from the viewpoint of the scalability in terms of the number of queries. Figure 3 illustrates transitions of the average number of queries that a peer receives. As shown in Figure 3, the FL method only slightly reduces the number of queries compared with full flooding. This is because the average number of relays in limited flooding is relatively large in our simulation experiments, independent of the block retrieval method. Since TTL is determined in accordance with the previous search results, the number of relays chosen for limited flooding immediately after full flooding tends to remain large. On the other hand, selective search can considerably reduce the number of queries.

2.4.2. Evaluation of continuity of media play-out. We defined 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. Although not shown in figures, we observed that, independent of the combination of methods, the waiting time decreases as the popularity increases and, independent of popularity, all media streams that are successfully found can be played out within 3.5 s. This is small enough from a viewpoint of service accessibility [15].

Figure 4(a) and (b) illustrate the completeness with 95% confidence interval of each media stream after 20 000 media requests occur. To evaluate the continuity of media play-out, we define the completeness as the ratio of the number of retrieved blocks in time to the number of blocks in a media stream. As shown in Figure 4(a) and (b), independent of method, media streams from 1 to 10 are played out almost continuously from the beginning to the end. On the other hand, as media popularity decreases, the completeness also deteriorates. In our experiments, most of the blocks that cannot be retrieved in time are blocks that have already been replaced by blocks of more popular streams. In spite of the less number of query messages, FLS method can accomplish equivalent completeness compared with other two methods. Comparing Figure 4(a) and (b), we find that there is little difference between SF and SR. This is because the remaining time is not used effectively and unexpected cache miss hardly occurs in our experiments.

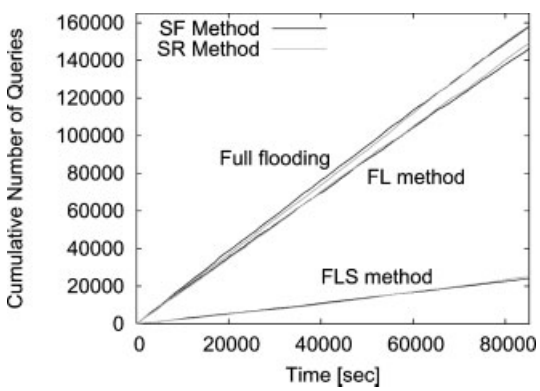

Figure 3. Cumulative number of queries.

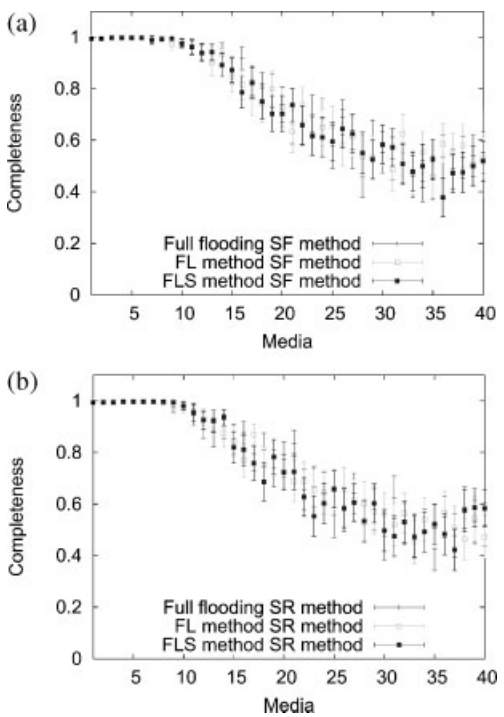

Figure 4. Completeness. (a) SF method. (b) SR method.

3. SUPPLY-DEMAND-BASED CACHE REPLACEMENT ALGORITHM

Although LRU is a simple and widely used scheme, simulation results showed that LRU cannot accomplish continuous media play-out under the condition of heterogeneous media popularity. This is because popular media streams are cached excessively while unpopular media streams eventually disappear from the P2P network.

In this section, 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 make conjectures about the behavior of other peers by itself. It is important to avoid the situation where a peer aggressively collecting information on 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 them based on locally available and passively obtained information, i.e. search results it obtained and P2P messages it relayed. Then, each peer autonomously determines a media stream to replace so that the supply and demand is well balanced according to the media popularity in the network. For this purpose, we use the response threshold model [3].

In the response threshold 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 individual i performs a task is given by the demand, i.e. stimulus s, and response threshold θ_i as $s^2/(s^2+\theta_i^2)$, for example. When individual i performs the task, θ_i is decreased and thus it tends to devote itself to the task. Otherwise, θ_i is increased. After performing the task several times, it becomes a specialist in the task. 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, they form 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. 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 allocated to the task reaches the appropriate level again.

3.1. Description of algorithm

By regarding the replacement of media streams as a task, we propose a cache replacement algorithm based on the response threshold model. In the cache replacement, a task corresponds to discarding a block of a media stream. However, per-block based decision consumes much computational power and memory. In addition, it leads to fragmentation of cached streams, and a cache becomes a miscellany of a variety of independent blocks of media streams. Thus, we define a stimulus as the ratio of supply to demand for a media stream. By introducing the response threshold model, a peer continuously replaces blocks of the same stream with newly retrieved blocks once a stream is chosen as a victim, i.e. a media stream to be replaced. As a result, fragmentation of media streams can be avoided. Each peer discards blocks based on the following algorithm when there is no room in the cache buffer to store a newly retrieved block.

Step 1: Estimate the supply and demand for media streams per round. For a set of cached media streams M, a peer calculates supply $S(i)$ and demand $D(i)$ for media stream $i \in M$ from search results it received and messages it relayed at the previous round. $S(i)$ is the ratio of total number of blocks for media stream i in received and relayed response messages to the number of blocks in media stream i. Here, to avoid overestimation, only response messages received are taken into account for $S(i)$ when a peer watches stream i. $D(i)$ is the number of query messages for media stream i, which the peer emitted and relayed.

Step 2: Determine a media stream to replace. Based on the 'division of labor and task allocation', we define ratio $P_r(i)$ that media stream i is replaced as follows:

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

where $s(i)$ is derived as $\max(S(i) - 1)/D(i), 0$, which indicates the ratio of supply to demand for media stream i after the replacement. $s(i)$ means how excessively media stream i exists in the network after it is discarded. $l(i)$ is the ratio of the number of locally cached blocks to the number of blocks in media stream i. $l(i)$ is used to restrain the replacement of a fully or well-cached stream. Among cached streams except for the stream being watched, for example stream m, a victim is chosen

with probability $(P_r(i)/\sum_{i\in M-m} P_r(i))$. Then, a peer discards blocks from the head or the tail of the stream at random. As in Reference [4], thresholds are regulated using Equation (5). Thus, media i is to be discarded more often once it is chosen as a victim.

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

Inspired by biological systems, we can accomplish fully distributed but globally well-balanced cache replacement. Furthermore, our proposed algorithm is insensitive to parameter settings since it adaptively changes the response threshold in accordance with the obtained information from the network. With slight modification of equations of the response threshold model, we can apply our proposed algorithm to other caching problems in distributed file sharing systems.

3.2. Simulation experiments

Since, it was shown that the FLS method can accomplish continuous media play-out with a smaller amount of search traffic compared with full flooding, we only show the results of the combination of the FLS and SF methods in this section. We conducted simulation experiments to evaluate our proposed cache replacement algorithm in terms of the continuity of media play-out and adaptability to changes in media popularity. The simulation model and scenario are the same as those used in Subsection 2.4. Based on the values used in Reference [2], we set the parameters as follows: $\xi = 0.01$ and $\varphi = 0.001$. $\theta(i)$ is initially set to 0.5, but it dynamically changes between 0.001 and 1. $s(i)$ is normalized by dividing by $\sum_i s(i)$. We show the average values of 40 sets of simulations in the following figures.

3.2.1. Evaluation of continuity of media play-out. Figure 5 compares the completeness with a 95% confidence interval of each media stream among LRU and the proposed approach. We find that our proposed algorithm can improve the completeness of unpopular media streams without affecting the popular streams. We conducted several experiments and verified that this improvement could be attained under a variety of conditions.

3.2.2. Evaluation of adaptability to changes in media popularity. We changed the popularity of each media stream over time based on a model used in Reference [13]. In the model, the media popularity changes every L media requests. Another well-correlated Zipf-like distribution with the same parameter ($\alpha = 1.0$) is used for the change. The correlation between two consecutive Zipf-like distributions is modeled by using a parameter n that can be any integer between 1 and the number of media streams, i.e. 40. First, the most popular media stream in the current Zipf-like distribution becomes the r_1th popular in the next Zipf-like distribution, where r_1 is randomly chosen between 1 and n. Then, the second popular media stream in the current distribution becomes the r_2th popular in the next distribution, where r_2 is randomly chosen between 1 and $\min(40, n+1)$, except that r_1 is not allowed. Thus, as time passes, initially popular media streams become less popular while initially unpopular media streams become more popular. We set $n = 5$ in the experiments and the demand changes every L media requests.

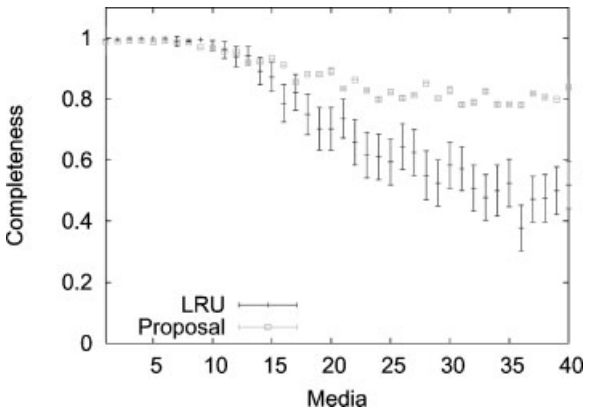

Figure 5. Completeness (LRU vs. proposal).

Figure 6 illustrates the transition of the completeness of the proposed algorithm. To clarify the transition, we show the completeness at instants when 5000, 10 000, 15 000 and 20 000 media requests occur. To evaluate the adaptability to the speed of popularity change, we set L to 200, 500 and 1000. As shown in Figure 6, in the case of $L = 200$ where the popularity changes fast, the completeness of initially unpopular media streams, identified by a large number, becomes higher than that of initially popular media streams with a smaller number as time passes and demand changes. On the other hand, in the case of $L = 1000$, where the popularity changes rather more slowly, the completeness of media streams with a small number is kept higher than that of media streams with a large number. Thus, we can conclude that our proposed algorithm can adapt to changes in media popularity. To further improve completeness, we can assume a repository or a peer with a larger cache buffer that posses media streams statically or for a longer duration of time.

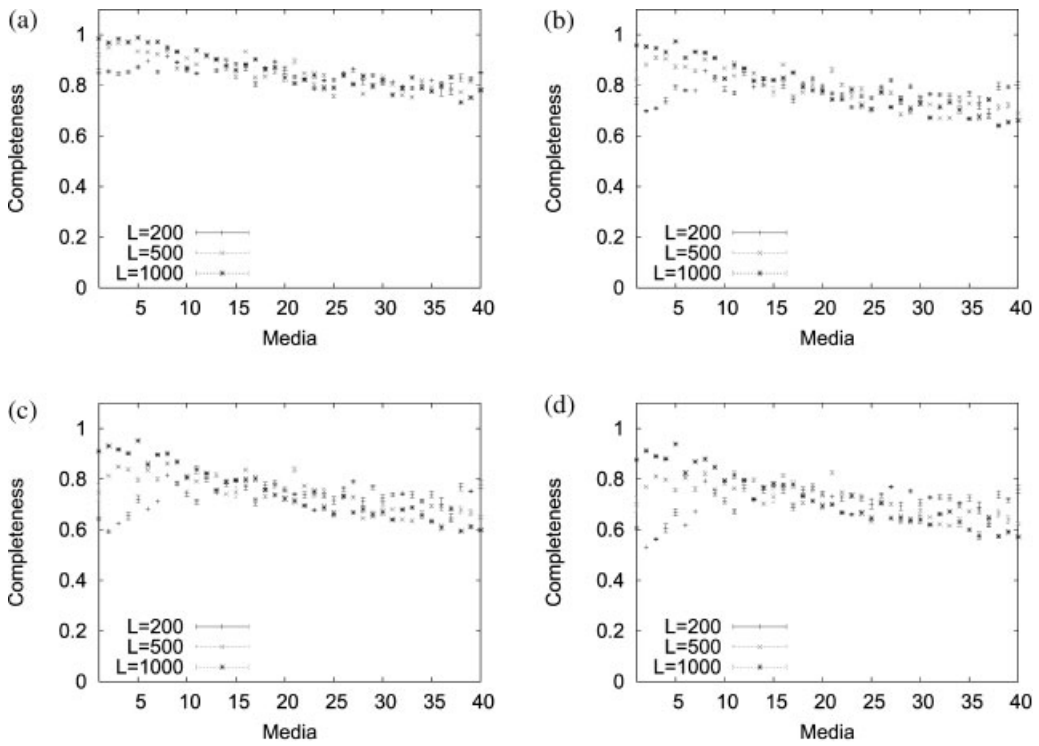

Figure 6. Completeness with changes in media popularity ($\xi = 0.01,\ \varphi = 0.001$). (a) Number of media requests: 5000. (b) Number of media requests: 10 000. (c) Number of media requests: 15 000. (d) Number of media requests: 20 000.

We conducted several simulation experiments by changing ξ and φ in Equation (5), which are associated with the degree of adherence to a specific victim in cache replacement. Although not shown in figures, we find there is almost no difference among different ξ and φ. Thus, we do not need to give careful consideration to a problem of parameter setting as in other algorithms that need several critical parameters to be carefully determined in advance. Furthermore, the proposed algorithm flexibly adapts to a variety of network environment without any parameter adjustment.

4. CONCLUSIONS

In this paper, we proposed two scalable search methods and two block retrieval methods for use in scalable and continuous media streaming on P2P networks. Through several simulation experiments, we showed that the FLS method can provide users with continuous media play-out without introducing extra load on the system. However, we also found that the continuity of media play-out deteriorates for unpopular media streams. To tackle this problem, inspired by biological systems, we further proposed an effective cache replacement algorithm that considers the supply and demand for media streams. Simulation experiments showed that our proposed algorithm can improve the continuity of media play-out compared with LRU. In addition, the proposed algorithm can adapt to changes in media popularity. As future research work, we should evaluate our proposed methods in more realistic situations where network conditions dynamically change and a peer randomly joins and leaves our system. We also plan to implement our mechanisms on a real system to verify the practicality of our proposal. An actual system has characteristics different from our assumptions or models in this paper. For example, peers are heterogeneous in terms of the capacity of cache buffer and access link. We expect that our proposed scheme can provide heterogeneous peers with continuous video streaming services. Furthermore, we will investigate the accuracy of estimations and how it affects the performance of our proposal.

Although, we consider that our proposal can adapt to estimation errors to some extent, we improve the algorithms taking into account real environments.

ACKNOWLEDGEMENTS

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

REFERENCES

1. AllCast. Available at http://www.allcast.com.
2. Bonabeau E, Dorigo M, Theraulaz G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press: UK, 1999.
3. Bonabeau E, Sobkowski A, Theraulaz G, Deneubourg J-L. Adaptive task allocation inspired by a model of division of labor in social insects. In *Proceedings of BCEC1997*, Skovde, 1997; pp. 36–45.
4. Campos M, Bonabeau E, Theraulaz G, Deneubourg J-L. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior* 2000; **8**(2):83–96.
5. Man C, Hasegawa G, Murata M. Available bandwidth measurement via TCP connection. In *Proceedings of IFIP/IEEE MMNS 2004* (E2EMON Workshop), San Diego, October 2004; pp. 38–44.
6. Jeon W, Nahrstedt K. Peer-to-peer multimedia streaming and caching service. In *Proceedings of ICME 2002*, Lausanne, August 2002.
7. Padmanabhan VN, Wang HJ, Chou PA. Resilient peer-to-peer streaming. *Microsoft Research Technical Report MSR-TR-2003-11*, March 2003.
8. Sasabe M, Wakamiya N, Murata M, Miyahara H. Scalable and continuous media streaming on peer-to-peer networks. In *Proceedings of P2P 2003*, Linköping, September 2003; pp. 92–99.
9. Schollmeier R, Schollmeier G. Why peer-to-peer (P2P) does scale: an analysis of P2P traffic patterns. In *Proceedings of P2P2002, Linköping, September 2002.*
10. Share Cast. Available at http://www.scast.tv.
11. Tran DA, Hua KA, Do T. ZIGZAG: an efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM2003*, San Francisco, March 2003.
12. Wang B, Sen S, Adler M, Towsley D. Optimal proxy cache allocation for efficient streaming media distribution. In *Proceedings of IEEE INFOCOM 2002*, New York, June 2002.
13. Wu K-L, Yu PS, Wolf JL. Segment-based proxy caching of multimedia streams. In *Proceedings of the 10th International WWW Conference*, New York, 2001; pp. 36–44.
14. Xu D, Hefeeda M, Hambrusch S, Bhargava B. On peer-to-peer media streaming. In *Proceedings of ICDCS2002*, Vol. 1, Vienna, July 2002; pp. 363–371.
15. Zona Reaserch, Inc. The economic impacts of unacceptable web site download speeds. Available at http://www.webperf.net/info/wp_downloadspeed.pdf.

AUTHORS' BIOGRAPHIES

Masahiro Sasabe received his B.E. and M.E. degrees from Osaka University, Japan, in 2001 and 2003 respectively. From 2003 to 2004, he was a Ph.D. student of Graduate School of Information Science and Technology, Osaka University and a 21COE-JSPS research fellow. Since July 2004, he is an assistant professor of Cybermedia Center, Osaka University. His research interest includes QoS architecture for real-time and interactive video distribution system. He is a member of IEICE.

Naoki Wakamiya received his M.E. and D.E. degrees from Osaka University, Japan, in 1994 and 1996 respectively. He was a research associate of Graduate School of Engineering Science, Osaka University from 1996 to 1997, a research associate of Educational Center for Information Processing from 1997 to 1999, an assistant professor of Graduate School of Engineering Science from 1999 to 2002. Since 2002, he is an associate professor of Graduate School of Information Science and Technology, Osaka University. His research interests include QoS architecture for distributed multimedia communications, wireless sensor networks and mobile ad hoc networks. He is a member of IEICE, IPSJ, ACM and IEEE.

Masayuki Murata received his M.E. and D.E. degrees from Osaka University, Japan, in 1984 and 1988 respectively. In 1984, he joined Tokyo Research Laboratory, IBM Japan, as a researcher. He was an assistant professor of Computation Center, Osaka University from 1987 to 1989, an assistant professor of Faculty of Engineering Science from 1989 to 1992, an associate professor of Graduate School of Engineering Science from 1992 to 1999. From 1999, he has been a professor of Osaka University. He moved to Graduate School of Information Science and Technology in 2004. He has more than 300 papers of international and domestic journals and conferences. His research interests include computer communication networks, performance modeling and evaluation. He is a member of IEEE, ACM, The internet Society, IEICE and IPSJ.

Hideo Miyahara received his M.E. and D.E. degrees from Osaka University, Japan in 1969 and 1973 respectively. From 1973 to 1980, he was an assistant professor of Faculty of Engineering, Kyoto University, Japan. He was an associate professor of Faculty of Engineering Science, Osaka University from 1980 to 1986, a professor of computation center from 1986 to 1989, a professor of Faculty of Engineering Science from 1989 to 2003, a director of Computation Center from 1995 to 1998, a dean of Faculty of Engineering Science from 1998 to 2000, a dean of Graduate School of Information Science and Technology from 2002 to 2003. From 1983 to 1984, he was a visiting scientist at IBM Thomas J. Watson Research Center. From 2003, he is a president of Osaka University. His research interests include performance evaluation of computer communication networks and multimedia systems. He is a fellow of IPSJ, IEICE and IEEE.

