Adaptive Media Streaming on P2P Networks

Masahiro Sasabe, Naoki Wakamiya, and Masayuki Murata
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
Email:{m-sasabe,wakamiya,murata}@ist.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 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 Least Recently Used (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.**

I. Introduction

With the growth of computing power and the proliferation of broadband access to the Internet, such as Asymmetric Digital Subscriber Line (ADSL) and Fiber To The Home (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 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]–[6]. 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 [7], 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 [8], [9]. By retrieving blocks from other 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 [10]. 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.

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 have also found that the continuity of media play-out deteriorates as the media popularity

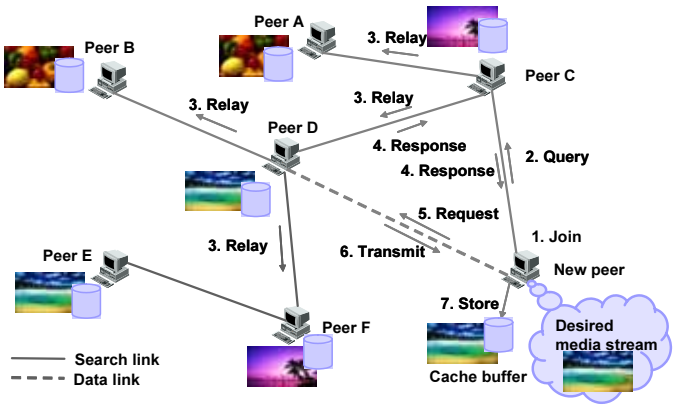


Fig. 1. Overview of our media streaming on pure P2P networks

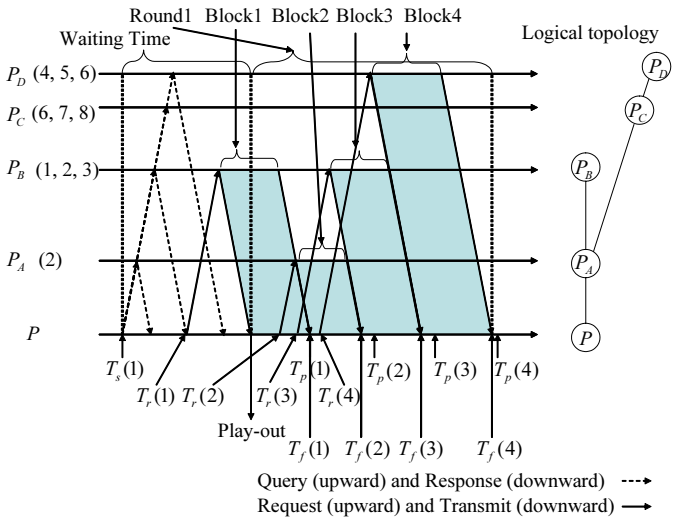


Fig. 2. Example of per-group search and retrieval

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 [11].

In biology, social insects, such as ants, also construct a distributed system [12]. 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 [12]. 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 [11]. By regarding the replacement of media streams as a task, we propose a fully distributed and autonomous cache replacement algorithm which can adapt to changes in environments, i.e., the supply-to-demand. Our proposed algorithm is also insensitive to parameter settings since it adaptively changes the response threshold taking into account the obtained information from the network. Through several simulation experiments, we evaluate the algorithm in terms of continuity of media play-out, adaptability to changes in media popularity, and sensitivity to parameter settings.

The rest of the paper is organized as follows. In Section II, we give an overview of our streaming system on P2P networks, describe our per-group based search and retrieval methods, and propose a supply-demand-based cache replacement algorithm. Then, in Section III, we evaluate our proposed cache replacement algorithm through several simulation experiments. Finally, we conclude the paper and describe future works in Section IV.

II. Mechanisms for Media Streaming on P2P Networks

Figure 1 illustrates our media streaming system on pure P2P networks. 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 basis. A peer participating in our system first joins a logical P2P network for the media streaming. Then, a peer sends a series of query messages to find blocks constituting the media stream that it wants to watch. We call the first query a media request. A peer retrieves blocks from other peers and plays them out. In this section, we describe our scalable search methods to find desired blocks and algorithms to determine the optimum provider peer from the search results. Finally, a cache replacement algorithm that takes into account the balance between supply and demand for media streams is given.

A. 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 2 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 block 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 $2RTT_{worst}$ 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.

B. Adaptive 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 responses 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, 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 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 [10]. 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, called the FLS method. 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., $R = \phi$.

C. 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 $j \geq 2$ 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.

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 $T_r(j)$ so that it can start receiving block j just after finishing the retrieval of block $j-1$, as shown in Fig. 2. 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 [13], 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.

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 in 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. In simulation experiments for this paper, we employed the SF method.

A peer emits a request message for block j to provider peer $P(j)$ at $T_r(j)$. On receiving the request, $P(j)$ initiates block transmission. If it replaced 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 $j-1$. 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 again 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 to the derivation of $T_f(j)$. In addition, deferment of the play-out also contributes to absorb estimation errors.

D. Supply-Demand-based Cache Replacement Algorithm

Although LRU is a simple and widely used scheme, it has been shown that LRU cannot accomplish continuous media play-out under heterogeneous media popularity [7]. 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 [11].

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 demands to perform a task increases as time passes and decreases as it becomes accomplished as $s(t+1) = s(t) + \delta - \alpha N_{act}/N$, where δ and α are parameters, N_{act} is the number of individuals performing a task among N individuals. The probability $P(X_i = 0 \rightarrow X_i = 1)$ 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. The probability $P(X_i = 1 \rightarrow X_i = 0)$ is given by a constant p. When the individual i performs the task, the threshold to the task is decreased as $\theta_i = \theta_i - \xi$, and thus it tends to devote itself to the task. Otherwise, the threshold is increased as $\theta_i = \theta_i + \varphi$. 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 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. Consequently, the ratio of individuals allocated to the task again reaches the appropriate level.

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 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.

Step1 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.

Step2 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:

$$P_r(i) \quad = \quad \frac{s^2(i)}{s^2(i) + \theta^2(i) + l^2(i)}, \qquad (2)$$

where $s(i)$ is derived as $\max\left(\frac{S(i)-1}{D(i)}, 0\right)$, 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. In our previous work, we found that continuous media play-out could not be sufficiently accomplished without $l(i)$ due to the fragmentation of cached streams. $l(i)$ is used to restrain the replacement of a fully or well-cached stream. Among cached streams except for the stream being watched, e.g., stream m, a victim is chosen with probability $\frac{P_r(i)}{\sum_{i \in M-m} P_r(i)}$. Then, a peer discards blocks from the head or the tail of the stream at random. As in [14], thresholds are regulated using Eq.(3). Thus, media i is to be discarded more often once it is chosen as a victim.

$$\forall j \in M,\ \theta(j) = \begin{cases} \theta(j) - \xi & \text{if } j = i \\ \theta(j) + \varphi & \text{if } j \neq i \end{cases} \qquad (3)$$

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.

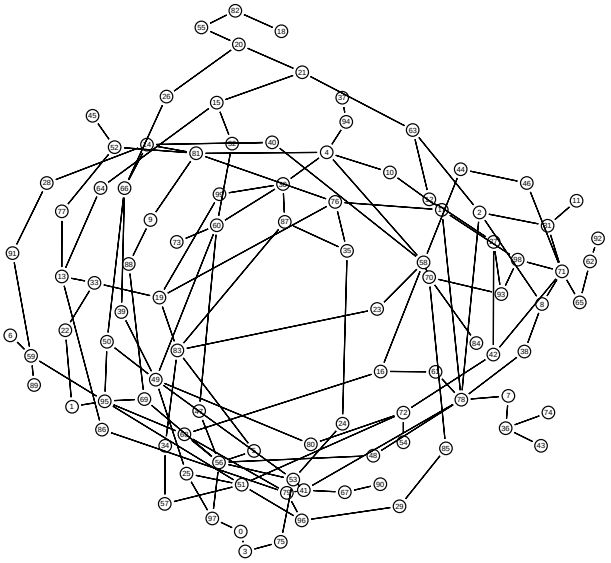


Fig. 3. Random network with 100 peers

III. Simulation Experiments

We conducted simulation experiments to evaluate our proposed cache replacement algorithm in terms of continuity of media play-out, adaptability to changes in media popularity, and sensitivity to parameter settings.

A. Simulation Model

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 Fig. 3. 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 methods, the available bandwidth between two arbitrary peers does not change during a simulation experiment and is given at random between 500 kbps 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 minutes. Forty media streams of 60 minutes 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 forty 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 minutes. A media stream is divided into blocks of 10-sec 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

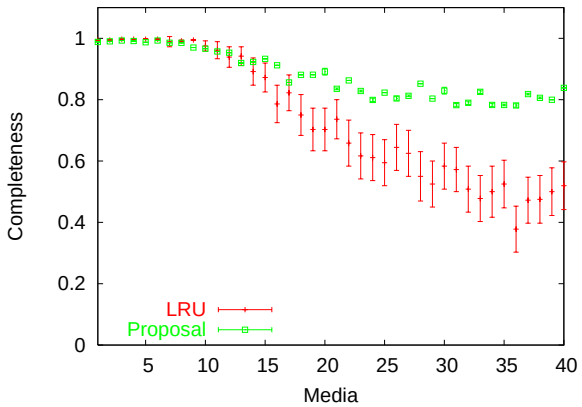


Fig. 4. Completeness (LRU vs. Proposal)

buffer of 675 MB, which corresponds to three media streams. Our algorithm can be effectively applied to other environments, since the replacement of cached blocks with newly retrieved blocks is always needed as long as most peers have a cache buffer that is insufficient to deposit all of the available 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. Based on the values used in [12], we set the parameters of the cache replacement algorithm 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 $\sum_i s(i)$. 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 show the average values of 40 sets of simulations in the following figures.

B. Evaluation of Continuity of Media Play-out

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. Although not shown in the figures, we observed that the waiting time decreases as the popularity increases. However, independent of popularity, all media streams successfully found can be played out within 1.7 sec. This is small enough from a viewpoint of service accessibility [15].

To evaluate the continuity 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. Figure 4 depicts the completeness with a 95 % confidence interval of each media stream after 20000 media requests. We find that our proposed algorithm can improve the completeness of unpopular media streams without affecting popular streams. As time passes, the completeness for unpopular media streams slightly decreases even with our algorithm. To 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. We conducted several experiments and verified that this improvement could be attained under variety of conditions.

C. Evaluation of Adaptability to Changes in Media Popularity

We changed the popularity of each media stream over time based on a model used in [16]. 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.

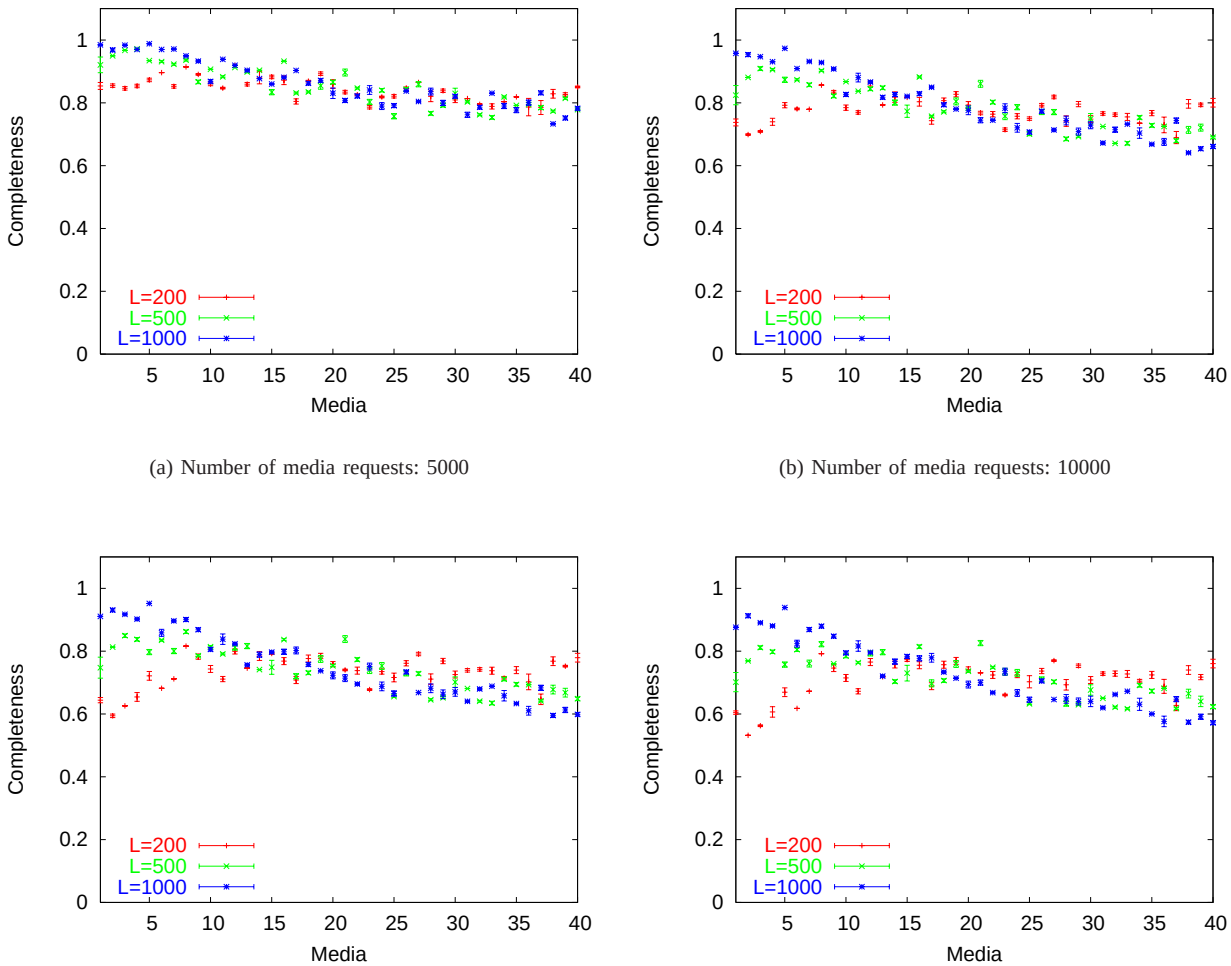
Figure 5 illustrates the transition of the completeness of the proposed algorithm. To clarify the transition, we show the completeness at instants when 5000, 10000, 15000, and 20000 media requests occur. To evaluate the adaptability to the speed of popularity change, we set L to 200, 500, and 1000. As shown in Fig. 5, 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.

D. Evaluation of Sensitivity to Parameter Settings

Our cache replacement algorithm has a set of parameters, $\theta(i)$, ξ, and φ. $\theta(i)$ is dynamically adjusted by Eq.(3). We conducted several simulation experiments by changing ξ and φ in Eq. 3, which are associated with the degree of adherence to a specific victim in cache replacement. As Figs. 5 through 7 illustrate, there is almost no difference among different ξ and φ. It follows that the proposed algorithm is insensitive to parameter settings. Thus, we do not need to give careful consideration to the 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 environments without any parameter adjustment.

IV. Conclusions

In this paper, we discussed effective methods for continuous media streaming on pure P2P networks. First, we introduced



(a) Number of media requests: 5000

(b) Number of media requests: 10000

(c) Number of media requests: 15000

(d) Number of media requests: 20000

Fig. 5. Completeness with changes in media popularity ($\xi = 0.01$, $\varphi = 0.001$)

adaptive block search methods and a block retrieval method for continuous media play-out. Next, inspired by biological systems, we proposed a supply-demand-based cache replacement algorithm for P2P media streaming. Through several simulation experiments, we showed that our proposed mechanism can accomplish continuous media play-out independent of media popularity. Furthermore, we also demonstrated that our proposed algorithm could adapt to changes in media popularity.

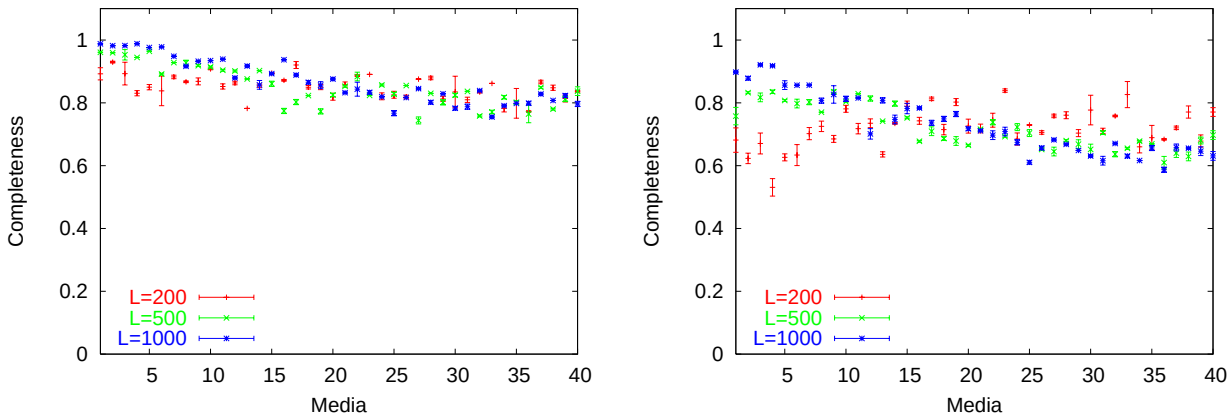
As future research work, we should evaluate our proposed methods in more realistic situations where network conditions dynamically change and a peer randomly joins and leaves our system. We also plan to implement our mechanisms on a real system to verify the practicality of our proposal. An actual system have 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.

ACKNOWLEDGMENTS

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

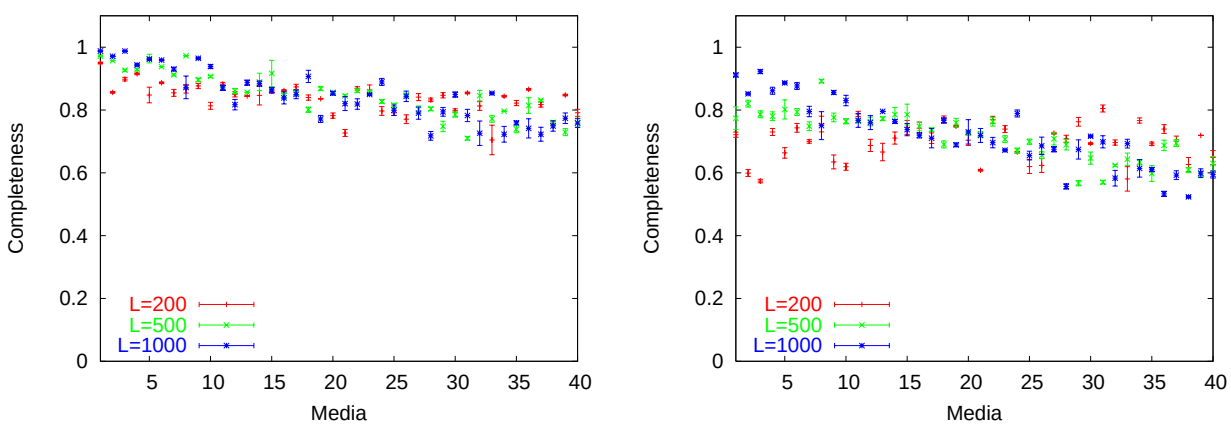
[1] AllCast, available at http://www.allcast.com.



(a) Number of media requests: 5000

(b) Number of media requests: 20000

Fig. 6. Completeness with changes in media popularity ($\xi = 0.1$, $\varphi = 0.01$)



(a) Number of media requests: 5000

(b) Number of media requests: 20000

Fig. 7. Completeness with changes in media popularity ($\xi = 0.001$, $\varphi = 0.0001$)

[2] 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.

[3] Share Cast, available at http://www.scast.tv.

[4] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE INFOCOM2003*, San Francisco, Mar. 2003.

[5] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *Proceedings of ICDCS2002*, vol. 1, Vienna, July 2002, pp. 363–371.

[6] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "PROMISE: Peer-to-Peer media streaming using CollectCast," in *Proceedings of ACM Multimedia 2003*, Berkeley, Nov. 2003, pp. 45–54.

[7] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara, "Scalable and continuous media streaming on Peer-to-Peer networks," in *Proceedings of P2P 2003*, Linköping, Sept. 2003, pp. 92–99.

[8] W. Jeon and K. Nahrstedt, "Peer-to-peer multimedia streaming and caching service," in *Proceedings of ICME2002*, Lausanne, Aug. 2002.

[9] 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.

[10] R. Schollmeier and G. Schollmeier, "Why peer-to-peer (P2P) does scale: An analysis of P2P traffic patterns," in *Proceedings of P2P2002*, Linköping, Sept. 2002.

[11] 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, 1997, pp. 36–45.

[12] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[13] Cao Man, Go Hasegawa and Masayuki Murata, "Available bandwidth measurement via tcp connection," *IFIP/IEEE MMNS 2004.*, in press, Oct. 2004.

[14] 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.

[15] Zona Reaserch Inc., "The economic impacts of unacceptable web site download speeds," available at http://www.webperf.net/info/wp-downloadspeed.pdf.

[16] 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*, 2001, pp. 36–44.