# Scalable, Adaptive, and Robust Media Streaming on Peer-to-Peer Networks

Masahiro Sasabe    Naoki Wakamiya    Masayuki Murata

Cybermedia Center
Osaka University
1-32 Machikaneyama, Toyonaka,
Osaka 560-0043, Japan
m-sasabe@cmc.osaka-u.ac.jp

Graduate School of Information Science and Technology
Osaka University
1-5 Yamadaoka, Suita,
Osaka 565-0871, Japan
wakamiya@ist.osaka-u.ac.jp    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. In this paper, we propose a Peer-to-Peer (P2P) media streaming system which can provide a large number of users with continuous media streaming services while dynamically adapting to changes in media popularity, peer leaves, and changes in network conditions. We show the effectiveness of proposed methods through several simulation experiments.

## *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.

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], [2]. 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 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 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 (called a provider peer). 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 provider peer nearby from a set of file holders, a peer can retrieve a file faster than a conventional client-server based file sharing.

Considering the fact that the client-server architecture lacks the scalability and adaptability, there have been several research works on media streaming over P2P networks [3]–[6]. Most of these were designed for use in live broadcasting. They have constructed an application-level multicast tree whose root is an original media server while the peers are intermediate nodes and leaves [3]–[5]. A media stream emitted from a server is distributed over a tree. Each intermediate peer receives media data from its parent peer, makes copies of data, and forwards them to its child peers. 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. On the other hand, the other works such as PROMISE [6] are similar to our proposed system, where a peer finds a provider peer for the whole or a part of a desired media stream by itself, retrieves it from the provider, and deposits it in its local buffer to supply to other peers. However, those works did not take into account variations in the popularity among multiple media streams.

In [7], we proposed scalable search and in-time media retrieval methods for on-demand media streaming on pure P2P networks. In our system, a media stream is divided into blocks for efficient use of network bandwidth and cache buffer [8], [9]. By retrieving blocks from appropriate provider 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 performs replacement on cached blocks with it.

Through several simulation experiments, we have shown that our systems can accomplish continuous media play-out for popular media streams without introducing extra load on the system. However, we have also found that the completeness 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 Least Recently Used (LRU) is a simple and widely used cache replacement algorithm, it fails in continuous media play-out.

To improve the completeness 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 stream 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 [10].

In biology, social insects, such as ants, also construct a distributed system [11]. 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 [11]. 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 [10]. 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 the completeness of media play-out, adaptability to changes in media popularity, and sensitivity to parameter settings.

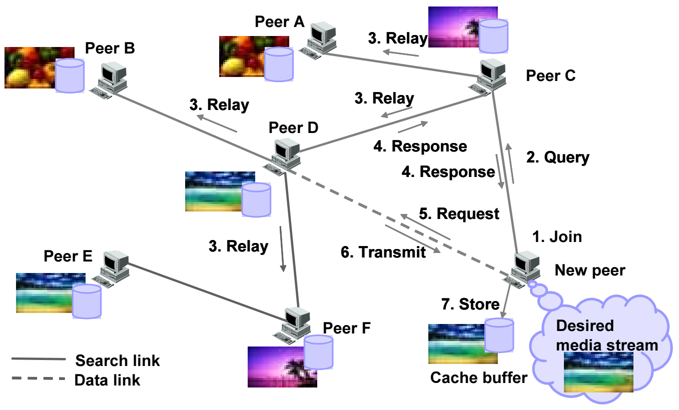


Fig. 1. Overview of our media streaming on pure P2P networks

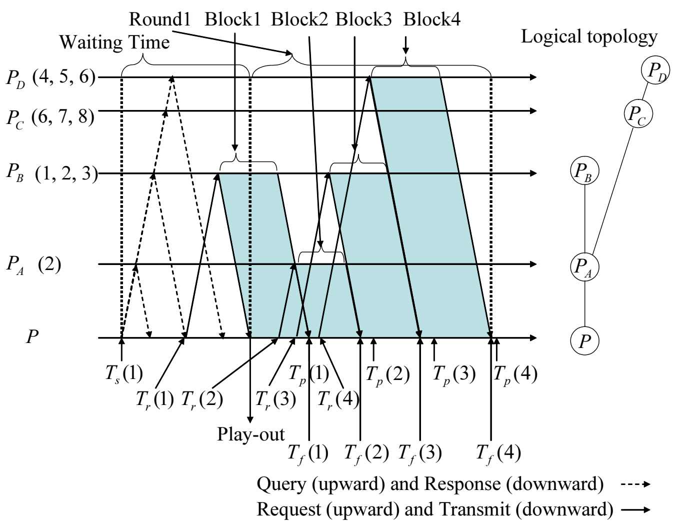
Furthermore, in an actual situation, media streaming fails since peers participating a service occasionally leave a P2P network due to user's interactions or system failures. Network conditions including the available bandwidth and Round Trip Time (RTT) also change dynamically. In this paper, we newly propose a block retrieval method which dynamically switches provider peers based on the estimation of the available bandwidth and RTT. In addition, we extend our block search method to prepare for peer leaves. Through several simulation experiments, we evaluate our proposed methods in terms of the completeness of media play-out under unstable system conditions.

The rest of the paper is organized as follows. In the next section, we first give an overview of our streaming system on P2P networks, then propose an adaptive and robust block search method, an adaptive block retrieval method, and a supply-demand-based cache replacement algorithm. Next, we evaluate our proposed methods through several simulation experiments. Finally, we conclude the paper and describe future works.

## *ADAPTIVE MEDIA STREAMING ON P2P NETWORKS*

### Overview of Proposed System

Figure 1 illustrates our media streaming system on pure 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. In our system, a peer retrieves a media stream and plays it out in a block-by-block manner. To find a block, a peer emits a query message to a P2P network. A query message is diffused over a P2P network by being copied and relayed by peers as shown as 'Relay' in Fig. 1. To avoid flooding a network with query messages, taking into account the temporal order of reference in a media stream, our method employs a per-group search scheme. 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



Round1 Block1 Block2 Block3 Block4
Waiting Time
Logical topology
$P_D$ (4, 5, 6)
$P_C$ (6, 7, 8)
$P_B$ (1, 2, 3)
$P_A$ (2)
$P$
$T_s(1)$
$T_r(1)$ $T_r(2)$
$T_r(3)$ $T_r(4)$
$T_p(1)$ $T_p(2)$ $T_p(3)$ $T_p(4)$
Play-out
$T_f(1)$ $T_f(2)$ $T_f(3)$ $T_f(4)$
Query (upward) and Response (downward)
Request (upward) and Transmit (downward)

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

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, that is, *flooding*. Since $P_A$, $P_B$, and $P_D$ have one or more block out of four requested blocks, they return response messages.

To accomplish continuous media play-out, $P$ sends a query for the next round at 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. Range of the next search is determined based on the search results of the current round as described later.

From search results obtained by the query, $P$ determines a provider peer for each block in the round, from which a block can be retrieved in time. Then, a peer retrieves blocks from provider peers as illustrated in Fig. 2. If the in-time block retrieval is expected to fail due to changes in network conditions, a peer switches provider peers. We will propose a block retrieval method adaptive to system changes.

If there is no room for depositing a newly retrieved block in its cache buffer, $P$ determines a media stream of which cached blocks are replaced with the new block, taking into account the balance between the supply and demand for media streams in the network. The detail of the cache replacement is given later.

## Adaptive and Robust Block Search Method

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 of the requested media stream, 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 $k$th 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 [12]. 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 is an efficient method 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., $R = \phi$.

If any of provider peers leaves a P2P network, a peer loses a chance to find and retrieve the corresponding block. To improve the robustness, we introduce parameter $x$, which defines the number of provider peers to be found in the current round to move to the selective search in the next round. For example, by setting $x$ to two, a peer moves to the selective search when it finds two provider peers and it can prepare an alternative provider peer.

## Block Retrieval Method

We first introduce our in-time block retrieval method [7], then newly propose a block retrieval method adaptive to system changes.

### In-time Block Retrieval Method

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 completeness 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 completeness 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.

### Adaptive Block Retrieval Method

In an actual situation, system conditions dynamically change. The in-time block retrieval fails if the available bandwidth decreases due to congestions. A peer cannot find a provider peer since search results becomes unreliable due to peer failures or leaves. To tackle this problem, we propose an adaptive block retrieval method that appropriately switches provider peers based on the estimated available bandwidth and RTT. As far as the following condition holds, a peer can retrieve block $j$ in time.

$$\frac{V(t) + r(j,t)}{\frac{B_s}{B_t}} \geq \frac{r(j,t)}{\Delta_{bw} A(i,t)}, \tag{2}$$

where $A(i,t)$ is the available bandwidth from peer $i$ at time $t$, $V(t)$ is the remaining buffer size at $t$, $r(j,t)$ is the remaining size of block $j$ being retrieved at $t$. $B_s$ and $B_t$ are block size and block time, respectively. $\Delta_{bw}$ takes into account the degree of accuracy of bandwidth estimation. When peer $i$ leaves a P2P network, the available bandwidth is considered to be zero.

When Eq.(2) cannot be satisfied, the peer tries to find an alternative provider peer $i'$ from which it can retrieve the remainder of block $j$ in time. Peer $i'$ must satisfy the following condition:

$$\frac{r(j,t)}{A(i,t)} \geq \min_{i' \in S_j - i} \left( \frac{r(j,t)}{A(i',t)} + 2R(i',t) \right), \quad (3)$$

where $S_j$ is the set of provider peers of block $j$ obtained from the search results and $R(i',t)$ is the RTT from peer $i'$ at time $t$. If the peer can find $i'$ then it retrieves the remainder of block $j$ from $i'$. Otherwise, it gives up retrieving block $j$.

## 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 [10].

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 $\delta$ and $\alpha$ 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 $\theta_i$ 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:

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

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

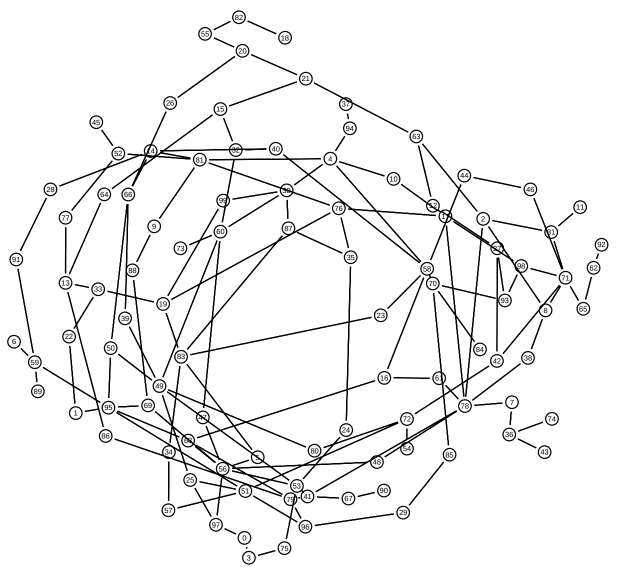


Fig. 3. Random network with 100 peers

$\frac{P_r(i)}{\sum_{i \in M-m} P_r(i)}$. Then, a peer discards blocks from the head or the tail of the stream at random. As in [14], thresholds are regulated using Eq.(5). Thus, media $i$ is to be discarded more often once it is chosen as a victim.

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

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.

## *SIMULATION EXPERIMENTS*

### 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.

Forty media streams of 60 minutes length were available. The media coding rate was set to CBR 500 kbps. A media stream was divided into blocks of 10-sec. Thus, one block amounts to 625 KBytes. Media streams were numbered from 1 (most popular) to 40 (least popular), where the various levels of popularity followed a Zipf-like distribution with $\alpha = 1.0$. Therefore, media stream 1 was forty times more popular than media stream 40.

At first, none of the 100 peers watched any media stream. Then, peers randomly began to request a media stream one by one. The inter-arrival time between two successive media requests for the first media stream among clients followed an exponential distribution whose average was 20 minutes. Each peer watched a media stream without such interactions as rewinding, pausing, or fast-forwarding. Each peer sent a query message for a succession of six blocks, i.e., $N = 6$. Blocks obtained were deposited into a cache buffer of 675 MB, which corresponds to three media streams. When a peer finished watching a media stream, it became idle during the waiting time, which also followed an exponential distribution whose average was 20 minutes.

At the beginning of each simulation experiment, each peer stored three whole media streams in its cache buffer. The initial population of each media stream in the network also followed the Zipf-like distribution with $\alpha = 1.0$. Based on the values used in [11], we set the parameters of the cache replacement algorithm as follows: $\xi = 0.01$ and $\varphi = 0.001$. $\theta(i)$ was initially set to 0.5, but it dynamically changed between 0.001 and 1. $s(i)$ was normalized by dividing by $\sum_i s(i)$. To prevent the initial condition of the cache buffer from influencing system performance, we only used the results after the initially cached blocks were completely replaced with newly retrieved blocks.

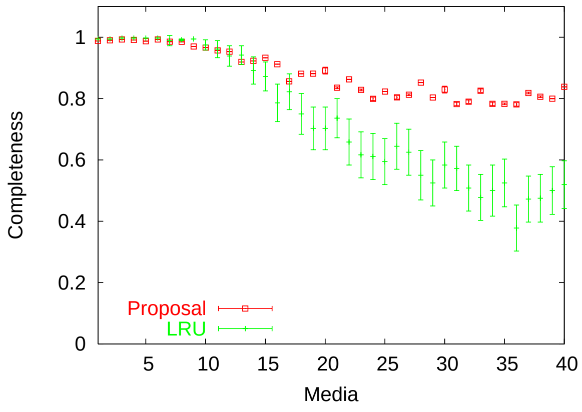


Fig. 4. Completeness (LRU vs. Proposal)

### Evaluation in Stable P2P Networks

We conducted simulation experiments to evaluate our proposed cache replacement algorithm in terms of the completeness of media play-out, adaptability to changes in media popularity, and sensitivity to parameter settings. In this scenario, we considered the stable P2P network where the available bandwidth and RTT did not change and we set $x$ to 1 and employed the in-time block retrieval method. We set the RTT from 10 ms to 660 ms and the available bandwidth from 500 kbps to 600 kbps between two arbitrary peers. We show the average values of 40 sets of simulations in the following figures.

#### Evaluation of Continuous 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 increased. 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].

We define the 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.

## Evaluation of Adaptability to Changes in 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 $r_1$th 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 $r_2$th 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.

## Evaluation of Sensitivity to Parameter Settings

Our cache replacement algorithm has a set of parameters, $\theta(i)$, $\xi$, and $\varphi$. $\theta(i)$ is dynamically adjusted by Eq.(5). We conducted several simulation experiments by changing $\xi$ and $\varphi$ in Eq. 5, which are associated with the degree of adherence to a specific victim in cache replacement. Although not shown in figures, we found that there was almost no difference among different $\xi$ and $\varphi$. 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 changes in media popularity without any parameter adjustment.

## Evaluation in Unstable P2P Networks

We conducted simulation experiments to evaluate the proposed improvements in terms of the completeness of media play-out under unstable P2P networks. We show the average values of 20 sets of simulations in the following figures.

## Evaluation with Changes in Network Condition

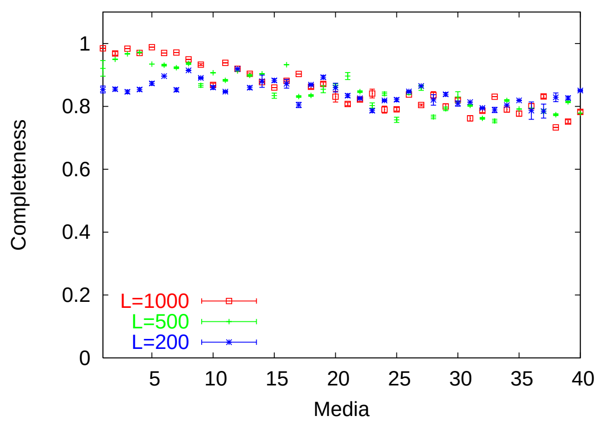
We randomly changed RTT from 10 ms to 660 ms and the available bandwidth from 450 kbps and 550 kbps between two arbitrary peers at one-second intervals. A peer estimated the available bandwidth and RTT at one-second intervals with the degree of accuracy of 0.975. We set $\Delta_{bw}$ to 0.95 which was less than the estimation accuracy. Since peers did not leave in this scenario, we set $x$ to 1.

Figure 6 depicts the completeness of the in-time block retrieval method and the adaptive block retrieval method. For comparison purposes, results in a stable environment are also shown. As shown in this figure, the completeness of the in-time block retrieval method is less than 0.5 even for the most popular media stream under unstable network conditions. This is because peers keep retrieving blocks even when the available bandwidth decreases and they cannot finish the retrieval in time. In contrast, the adaptive block retrieval method can improve the completeness by appropriately changing a provider peer. However, it is still lower than in a stable environment.

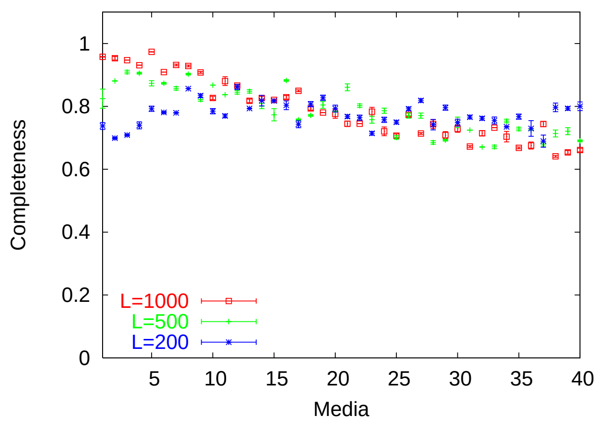
For a peer effectively to switch provider peers, it needs to have at least one alternative peer $i'$ which satisfies Eq.(6).

$$\left\{ i' \in S_j - i \,\middle|\, \frac{V(t) + r(j,t)}{\frac{B_s}{B_t}} \geq \frac{r(j,t)}{A(i',t)} + 2R(i',t) \right\} \tag{6}$$

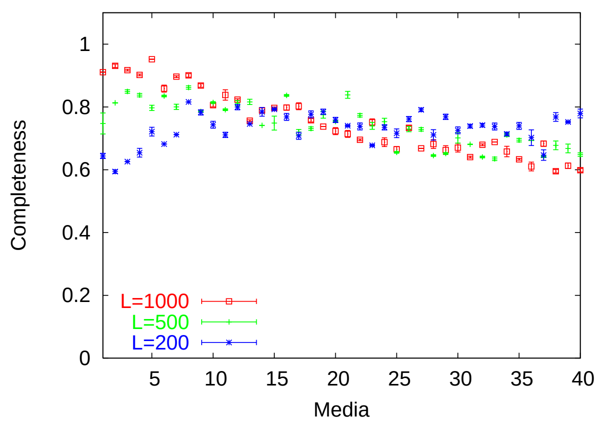
The number of alternative peers depends on the number of media streams against the capacity of the whole P2P network. Figure 7 illustrates the results of the case with 10 media streams. As shown in the figure, the completeness of the adaptive block retrieval method is more than 0.9 independently of the media popularity. Although not shown in figures, we conducted additional experiments with the various number of media streams and found that the completeness of more than 0.9 could be attained for all media popularity when the number of media streams was less than 12 against the network capacity of $100 \times 3 = 300$ media streams.



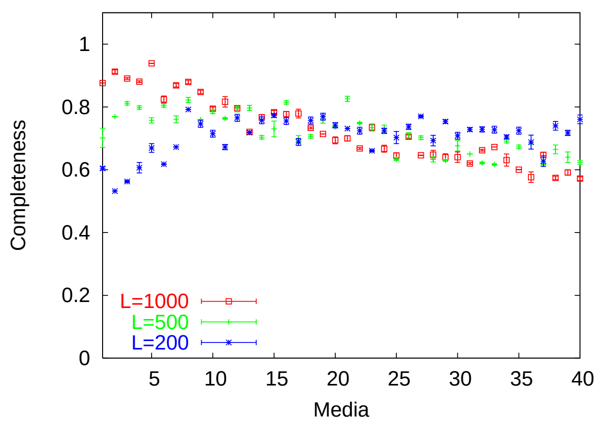
(a) Number of media requests: 5000



(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 ($\xi = 0.01$, $\varphi = 0.001$)

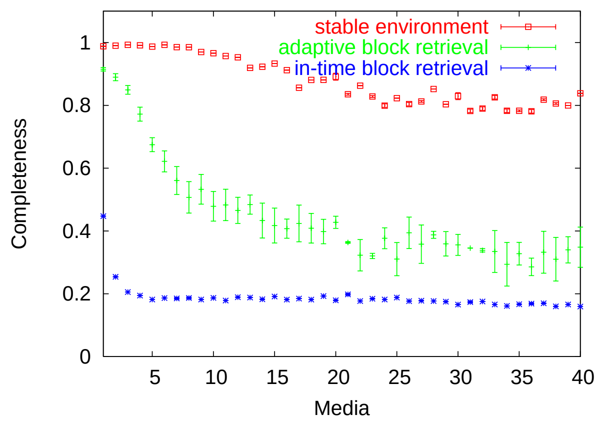


Fig. 6. Completeness (40 streams)

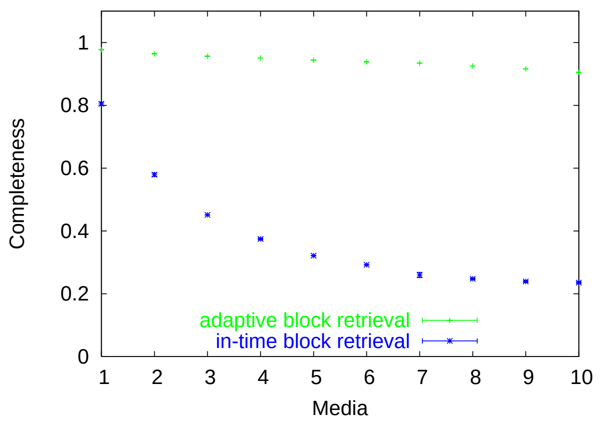


Fig. 7. Completeness (10 streams)

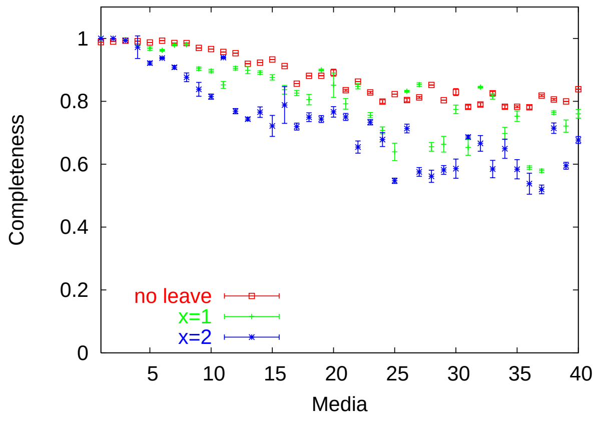


Fig. 8. Completeness with peer leaves

### Evaluation with Leaves of Provider Peers

To evaluate the robustness to peer leaves, we next considered the following scenario. First, one designated peer was randomly chosen. Then, we removed peers while the designated peer was retrieving the first media stream. Peers to be removed were randomly chosen among peers that deposited blocks of the media stream which the designated peer was interested in. The inter arrival time between two successive removals followed an exponential distribution whose average was 10 minutes. To investigate the impact of peer leaves, there was no reconstruction or recovery of the P2P network when peers left and links were broken and the available bandwidth and RTT did not change in this scenario.

Figure 8 shows that the completeness of $x = 1$ is higher than that of $x = 2$, on the contrary to our expectation. With $x = 2$, a peer conducts the limited flooding more often than the case with $x = 1$. In the limited flooding, query messages are diffused over a P2P network by being relayed by peers. Thus, the possibility that a peer can find an appropriate provider peer decreases due to breaks of the network caused by disappearance of peers. On the other hand, in the selective search, query messages are directly sent to provider peers without mediations of other peers. As a result, even though only 6 $\%$ peers left the network in Fig. 8, the completeness of the case of $x = 1$ becomes superior to that of $x = 2$. Thus, we can conclude that the selective search is more robust to peer leaves than the limited flooding from simulation results.

## *CONCLUSIONS*

In this paper, we proposed an adaptive and robust block search method, an adaptive block retrieval method, and a supply-demand-based cache replacement algorithm inspired by biological systems. Through several simulation experiments, we showed that our proposed cache replacement algorithm could accomplish continuous media play-out independent of media popularity and adapt to changes in media popularity. Furthermore, we demonstrated that the adaptive block retrieval method could improve the completeness compared to the in-time block retrieval method. In addition, we found that the selective search was more efficient than flooding methods when peers failed and left.

As a future research work, we plan to implement our proposed methods on a real system to verify the practicality and evaluate the applicability.

## *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] M. Sasabe, Y. Taniguchi, N. Wakamiya, M. Murata, and H. Miyahara, “Proxy caching mechanisms with quality adjustment for video streaming services,” *IEICE Transactions on Communications Special Issue on Content Delivery Networks*, vol. E86-B, no. 6, pp. 1849–1858, June 2003.

[2] J. Liu and J. Xu, “Proxy caching for media streaming over the internet,” *IEEE Communications Magazine*, vol. 42, no. 8, pp. 88–94, Aug. 2004.

[3] AllCast, available at http://www.allcast.com (last access: 2005/01/13).

[4] 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.

[5] D. A. Tran, K. A. Hua, and T. T. Do, “A peer-to-peer architecture for media streaming,” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 121–133, Jan. 2004.

[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] 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.

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

[12] 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.

[13] C. Man, G. Hasegawa, and M. Murata, “Available bandwidth measurement via TCP connection,” in *Proceedings of IFIP/IEEE MMNS 2004 E2EMON Workshop*, San Diego, Oct. 2004, pp. 38–44.

[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 (last access: 2005/01/13).

[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.

## *THE AUTHOR*

**Masahiro Sasabe** is an Assistant Professor of Cybermedia Center, Osaka University, Japan. Previously he received the B.E. and M.E. degrees from Osaka University, 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. His research interest includes QoS architecture for real-time and interactive video distribution system. He is a member of IEICE.

**Naoki Wakamiya** is an Associate Professor of Graduate School of Information Science and Technology, Osaka University, Japan. Previously he received the M.E. and D.E. from Osaka University, 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. 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** is a Professor of Graduate School of Information Science and Technology, Osaka University, Japan. Previously he received the M.E. and D.E. degrees from Osaka University, 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. He has more than three hundred 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.