Scalable and Continuous Media Streaming on Peer-to-Peer Networks

Masahiro Sasabe Naoki Wakamiya Masayuki Murata Hideo Miyahara
Graduate School of Information Science and Technology, Osaka University, Japan
{m-sasabe, wakamiya, murata, miyahara}@ist.osaka-u.ac.jp

Abstract

With the growth of computing power and the proliferation of broadband access to the Internet, media streaming has widely diffused. Although the proxy caching technique is one method to accomplish effective media streaming, it cannot adapt to the variations of user locations and diverse user demands. By using the P2P communication architecture, media streaming can be expected to smoothly react to network conditions and changes in user demands for media-streams. In this paper, we propose efficient methods to achieve continuous and scalable media streaming system. In our mechanisms, a media stream is divided into blocks for efficient use of network bandwidth and storage space. We propose two scalable search methods and two algorithms to determine an optimum provider peer from search results. Through several simulation experiments, we show that the FLS method can perform continuous media play-out while reducing the amount of search traffic to 1/6 compared with full flooding.

1. Introduction

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

The proxy mechanism widely used in WWW systems offers low-delay and reliable delivery of data by means of a "proxy server." The proxy server deposits multimedia data that have passed through it in its local buffer, called the "cached buffer," and 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 expect that high-quality and low-delay streaming services can be accomplished without introducing extra load on the system. However, the current proxy mechanism cannot adapt to the variations of user locations and diverse user demands. In addition, it has been pointed out that the server–client model lacks scalability and stability. All information is concentrated in a few designated servers that are statically located at various points in the network, and they have to process all requests coming in.

Peer-to-peer (P2P) is a new network paradigm to solve these problems. In a P2P network, peers, entities that constitute the P2P network, 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, where a consumer peer directly communicates with a provider peer to obtain a file. Since there is no server, over-concentration of traffic can be avoided.

By using the P2P communication technique, media streaming can be expected to flexibly react to network conditions and changes in user demands for media streams. The P2P network is dynamically constructed by instances of joining and leaving the network by peers. A consumer peer searches a desired media stream by itself and retrieves it from an appropriate provider peer. At this time, the consumer peer can become a provider peer of the media stream for other peers if the media stream is cached there. 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 function as intermediate nodes and leaves. This architecture is effective when user demands are simultaneous and concentrated on a specific media stream, as in live-media streaming services. However, when demands arise intermittently and peers request a variety of media streams, as in current P2P services, an efficient distribution tree cannot be composed.

In this paper, we discuss methods for providing media

streaming with QoS considerations in a scalable way on pure P2P networks, that is, there is no server. By taking into account the network conditions and the timeliness of data arrival, a peer finds a set of peers having a desired media stream and then retrieves the media stream from the most appropriate peer. First, we introduce segmentation of media streams for efficient use of storage space and bandwidth. Next, we propose two scalable methods to find a desired media block. Finally, two algorithms to determine an optimum provider peer from the search results are proposed. Through several simulation experiments, we compare several combinations of those methods and algorithms, in terms of the amount of search traffic and the continuity of media play-out.

The rest of the paper is organized as follows. In Section 2, we give an overview of our streaming system on P2P networks and propose several methods to accomplish continuous and scalable media streaming. Next, in Section 3, we evaluate our proposed methods through several simulation experiments. Finally, we conclude the paper and describe future works in Section 4.

2. Media streaming on P2P networks

A peer participating in our system first joins in a logical P2P network for the media streaming. Members of the P2P logical network are peers that are being served. Some of them may be watching media streams while the others are not. Each peer maintains a part or the whole of one or more media streams that it has watched or is watching. In our system, a media stream is divided into blocks for efficient use of network bandwidth and storage space. Since there is no server that manages meta information such as locations of peers and media data, a peer retrieves and stores a media stream in a block-by-block basis [7-10]. Since the query messages propagates in the P2P network in an exhaustive way, we propose two scalable methods for searching a desired block in Subsection 2.3. A peer that has the corresponding media stream sends a response message about cached data. The new peer determines an appropriate peer for retrieving the media stream on the basis of responses, retrieves it from the peer, and plays it out. We will propose two algorithms for this purpose in Subsection 2.4. The new peer repeats the same procedure until it successfully receives and perceives the entire media stream. Thus, each peer plays the roles of both consumer and provider. Messages and media streams are transferred over TCP and UDP sessions, respectively.

2.1. Segmentation of media stream

For the efficient use of storage space and the retrieval of necessary parts of a media stream, it is effective to divide a media stream into blocks [7-10].

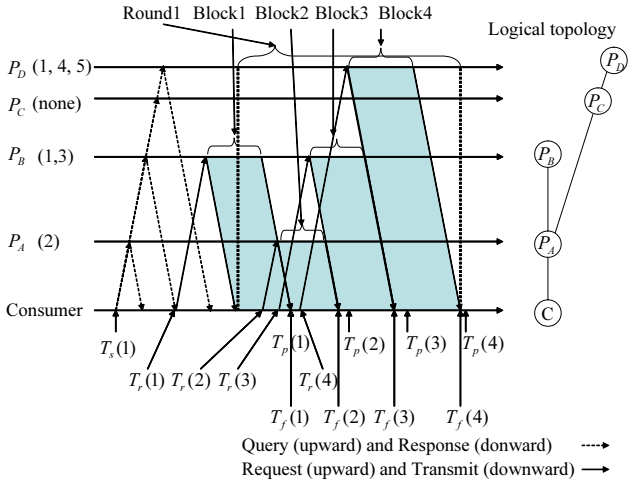


Figure 1. Basic behavior of our mechanism

A "block" is a processing unit that can be encoded and decoded by itself. An example of a block unit is a multiple of the GoP (Group of Pictures) of MPEG-2. A GoP can be coded and decoded independent of the other GoP, when the closed-GoP algorithm is employed. Since dividing a two-hour media stream into one-second blocks apparently makes it difficult to maintain cache buffer, to have a multiple of GoP is reasonable for a MPEG-2 media stream. However, a longer block introduces the possibility that the network condition drastically changes while retrieving a block and, as a result, the continuous media play-out cannot be attained. The number of blocks of a media stream affects the system scalability in terms of the amount of search traffic. In our experiments in Section 3, we use a block of 10 seconds. The details of appropriate block size are discussed in Subsection 2.3.

2.2. Basic behavior of proposed system

A peer finds and obtains a desired block by itself without the mediation of servers. Flooding is a powerful means of finding media data in an ideal network where bandwidth is unlimited and propagation delay is negligible. However, a block-by-block exhaustive search by flooding apparently consumes much bandwidth as the number of peers increases and lacks scalability. Therefore, taking into account the temporal order of reference in a media stream, our mechanism employs two methods to efficiently and effectively search and retrieve blocks without deteriorating the scalability of media streaming services.

The first method is based on a per-group search. A consumer peer periodically sends out a query message for N consecutive blocks. Figure 1 illustrates an example of $N = 4$. The symbols on the horizontal axis in the figure are for further use in the next subsection. P_A, P_B, P_C, and

P_D indicate candidates of provider peers that are within the range of the propagation of query messages. Numbers in parentheses besides peers stand for identifiers of the blocks that a peer has. At time $T_s(1)$, a query message for blocks from 1 to 4 is sent out from a consumer peer to the closest peer P_A. Since P_A has the second block out of four requested blocks, it returns a response message. The response message contains a list of identifiers of the blocks it has. It also relays the query to the next neighboring peer P_B. P_B also responses and relays the query. Since P_C does not have any of the four blocks, it only relays the query. Finally, P_D sends back a response message.

The consumer peer first waits for a response for block 1. However, if no response for block 1 arrives until $T_s(1) + 4$, the consumer peer gives up watching the media stream. Since it takes twice the response time to start the media play-out as shown in Fig. 1 and we have the so-called eight-second rule, we consider a time-out value of four seconds as appropriate. On receiving the response, the consumer peer immediately sends out a request message to retrieve the first block from the provider peer P_B for faster media play-out. By observing the way that the response message is received in regard to the query message, the consumer estimates the available bandwidth and the transfer delay from the provider peer. The estimates are updated through reception of media data. For more precise estimation, we can use any other measurement tools as long as they do not disturb media streaming. From estimations, the consumer peer predicts the completion time of retrieval of the first block. Since the received block is immediately decoded and displayed, deadlines of retrieval of all succeeding blocks are determined at this time. For each of blocks 2 through 4, the consumer chooses an appropriate peer to retrieve the block in time on the basis of received response messages and deadline. To efficiently utilize the bandwidth and avoid congestion, the retrieval of a block is scheduled to start immediately after the preceding block is completely retrieved. We define the period of the retrieval of blocks from 1 to N as round 1. A query message for the next N blocks in round 2 is also scheduled appropriately so that the peer can receive enough responses and the retrieval of block $N + 1$ starts at the desired instant. For the detailed scheduling algorithm, refer to the following subsections.

Our per-group search spoils the freshness of responses. Since a provider peer is also a customer of the media streaming service, it may watch a media stream at the same time, and the contents of its cache buffer may change. The limited capacity of a cache buffer raises the possibility that the required block, which is listed in the response, disappears when the consumer peer decides to retrieve the block from the peer and a request message arrives at the peer. To solve this problem, the consumer peer takes into account the probability of such disappearance in selecting the provider peer from which to retrieve the block. For this purpose, we employ LRU (Least Recently Used) as a cache replacement algorithm, and the response message takes the form of a list of all cached blocks in ascending order of referenced time.

The second method contributing to scalability is the suppression of message exchanges. In a P2P framework, a peer relays a query message to all of the logically neighboring peers that it knows. A response message is reversely relayed backward on the same path that the corresponding query message traversed. The number of relays is limited by the same mechanism as in the IP protocol. When a query is sent out, it is given a TTL (Time To Live) designation, which specifies the maximum number of relays. When an intermediate peer relays a query to neighboring peers, it decreases the TTL by one. If a peer receives a query with TTL equal to zero, it ceases to relay the query.

This flooding with a large TTL costs much in the number of message exchanges and the bandwidth consumed, although a peer can find many peers that have some of the required blocks. When a query is given a TTL whose value is H and a peer knows D other peers, the number of query messages relayed becomes $\sum_{i=1}^{H}(D-1)^i = O((D-1)^{H+1})$. Each participant regularly sends queries to find and retrieve media data.

Our second method decreases bandwidth consumption and improves scalability. In the first search, where a newly participating peer tries to find as many candidate providers as possible, full flooding is conducted where a query message is given a TTL whose value is H. For example, the default value of Gnutella, i.e., 7, is used as TTL. For the succeeding searches, the relay limit is decreased to $H' < H$ so that the peer can receive a sufficient number of response messages without wasting bandwidth. This is called limited flooding. Furthermore, we consider a selective search where query messages are directly sent from the peer to a selected set of candidate peers in unicast sessions. Detailed discussions on how we combine these three types of searches will be given in the next subsection.

2.3. Block search mechanism and algorithm

A new peer first tries full flooding by sending query messages to neighboring peers. A query message consists of a query identifier, a media identifier, a pair of block identifiers to specify the range of blocks needed, e.g., $(1, N)$, a time stamp, and TTL. The query identifier and the media identifier are each uniquely numbered.

When an intermediate peer receives the query, it first refers to a table to avoid making a loop of query relays. The forwarding table is composed of pairs of a query identifier and a peer identifier from which the peer received the query. If the query identifier of the received query mes-

sage is already recorded in the table, the query is discarded. A peer that has any of blocks in the specified range sends back a response message by relaying backward on the same path that the corresponding query message traversed. The response message contains a list of all cached blocks in ascending order of referenced time, the TTL in the query, and sum of the timestamp 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. Then, the query message is relayed to neighboring peers after decreasing the TTL by one if TTL is not zero. When there are two or more neighboring peers, the peer makes copies of the query message and sends them to neighboring peers.

While retrieving the first N blocks, the new peer searches the next N blocks. As mentioned in the previous subsection, full flooding costs much in terms of the number of messages exchanged and the bandwidth consumed. In order to efficiently gather sufficient information about desired blocks without introducing extra load on the network, it is effective to restrict the number of peers to be searched by carefully choosing TTL on the basis of the previous search results. In limited flooding, TTL is determined so that all peers that are expected to have any of blocks to retrieve in the next round are within the range of search. As mentioned in Subsection 2.2, the contents of a cache change as time passes. Since a peer does not have any way to know the contents of another peer's cache at the time it determines TTL, it conjectures the transition of the contents from an obtained response. Assuming that a peer is watching a media stream without interactions such as rewinding, pausing, and fast-forwarding, and that the cache buffer is filled with blocks, the number of blocks removed can be estimated by dividing the elapsed time from the arrival of the response message by one block time B_t. For example, B_t is equal to 0.5 sec when a block corresponds to a GoP of 15 frames and the media is played out at 30 frames per second. The peer conjectures the contents of all peers that returned response messages and obtains a set R of peers, which are expected to have at least one of blocks from $N+1$ to $2N$. We should note here that we do not take into account blocks cached after a response message is generated, since we cannot predict which block of what media stream will be retrieved and cached without up-to-date knowledge of a distant peer's behavior. It is also risky to rely on a block that has not existed but is expected to exist.

To further reduce the amount of search traffic, we propose a selective search method. The purpose of the flooding scheme is to find potential peers that did not respond to the previous query but newly obtains blocks of interest. Flooding also finds peers that have newly joined the service. However, the sufficient number of peers is already known, and they are expected to have blocks in the next round. Accordingly, it is less efficient to use flooding to find only a few new candidate peers while introducing a high load on the network. In such a case, it is useful to directly send queries to known peers to confirm the existence of desired blocks.

We propose two scalable search methods by combining full flooding, limited flooding, and selective search.

FL method

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

FLS method

The FLS method is a combination of full flooding, limited flooding, and selective search. For the next round's blocks, a peer conducts (1) selective search if the conjectured contents of cache buffers of peers in R contain all of the next round's blocks, (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., $R = \phi$.

Here, we examine the scalability of each method in terms of the amount of search traffic V that a peer induces per media stream. V is defined as the total number of query messages that are relayed and generated. First, in the case of the full flooding scheme, V becomes

$$V_F = \frac{M}{N}(D-1)^{H+1}, \quad (1)$$

where D is the average number of neighboring peers, H is a default value of TTL, and M is the number of blocks in a media stream. Next, in the case of the FL method, the amount of search traffic V becomes

$$V_{FL} = \left(\frac{M}{N} - L\right)(D-1)^{H+1} + L(D-1)^{H'+1} \quad (2)$$

H' is the average value of TTL in the case of the limited flooding and $L < \frac{M}{N}$ is the number of times that limited flooding is chosen. Finally, in the case of the FLS method, the amount of search traffic V becomes

$$V_{FLS} = \left(\frac{M}{N} - L - Q\right)(D-1)^{H+1} + L(D-1)^{H'+1} + Q\overline{|R|}. \quad (3)$$

$\overline{|R|}$ is the average number of peers in R and $Q < \frac{M}{N}$ is the number of times that the selective query is chosen. Since the first search is the full flooding, $L + Q < \frac{M}{N}$.

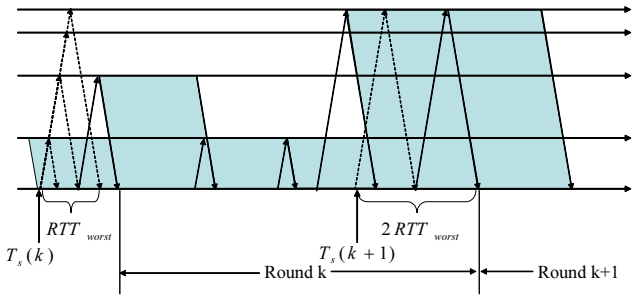


Figure 2. Search start time ($N = 4$)

In the case of the full flooding scheme, the amount of search traffic does not changes regardless of the popularity of a media stream. On the other hand, the performance of each proposed method is influenced by the media popularity. For a popular media stream, the amount of search traffic of each proposed method becomes the following:

$$V_{FL} = (D-1)^{H+1} + \left(\frac{M}{N} - 1\right)(D-1)^{H'+1}, \quad (4)$$

$$V_{FLS} = (D-1)^{H+1} + \left(\frac{M}{N} - 1\right)\overline{|R|}. \quad (5)$$

For an unpopular media stream, the amount of search traffic of the FL method is equal to that of the FLS method as follows:

$$V_{FL} = V_{FLS} = \left(\frac{M}{N}\right)(D-1)^{H+1} = V_F. \quad (6)$$

Independent of methods, the amount of search traffic is proportional to the number of blocks in a media stream, M, and is inversely proportional to the number of blocks in a round, N. Thus, to have $N = M$ is the most effective to reduce the amount of search traffic since a peer conducts search only once. When the block size is small, M becomes large for a long media stream. If N is set at M, information available in retrieving Mth block becomes out-of-date and of no use because cached blocks listed in response messages have been replaced with other blocks. Determination of appropriate block size and group size in accordance with, for example, the number of peers, the media size, and the network conditions, remains as a future research work.

To accomplish continuous media play-out, it is indispensable for the peer to emit a query while considering response time. It takes one round trip time to receive a response message from a peer. It also takes one round trip time before the beginning of reception of a block after a peer sends a request message. Thus, to efficiently utilize the bandwidth without causing congestion by starting the reception of the first block of the next round, a query message for the next N blocks should be issued two round trip times earlier than the estimated completion time of receiving the last block of the current round (Fig. 2). Taking into account the worst case that the first block of the next round is found only in a cache buffer of the most distant peer, it is necessary to consider RTT required by the most distant peer among those peers at which a query message is expected to arrive. Thus, the time to issue a query message for the next round $k+1$ is given as $T_s(k+1) = T_p(kN) - 2RTT_{worst}$, where RTT_{worst} is the RTT, which is estimated by observing the way that response messages are received or by measurement tools, to the most distant peer among peers which returned response messages within kth round. The peer gives up trying to retrieve blocks whose corresponding request messages have not been emitted in the current round k at $T_s(k+1)$.

2.4. Block retrieval mechanism and algorithm

The new peer sends a request message for the first block of a media stream as soon as it receives a response message from a peer that has the block without waiting for other responses. In some cases, for example, when the available bandwidth to the closest peer is far smaller than that to the next closest peer, it is worth waiting for other response messages to find a better peer. However, the new peer cannot predict whether any better peers exist or not when it receives the first response message for the first block. In addition, it is indispensable for a low-delay and suitable media streaming service to begin the media presentation as fast as possible. Thus, in our mechanism, the new peer retrieves the first block from the peer that first answers and plays it out immediately when the its reception starts. Of course, we can defer the play-out to buffer some blocks preparing for unexpected delays.

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

$$T_p(j) = T_p(1) + (j-1)B_t, \quad (7)$$

where $T_p(1)$ corresponds to the time that the peer finishes retrieving the first block and B_t stands for the duration of playing out one block. As far as the retrieval of block j is completed before $T_p(j)$, QoS in terms of the continuity of media play-out can be guaranteed.

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. In our block retrieval mechanism, a request message for a block is sent out early enough for the block retrieval to finish in time without causing congestion and to efficiently utilize bandwidth as shown in Fig. 1. Every time a peer receives a response message, the instant that it emits a request message and the peer from which it receives a block are determined. The detailed algorithm is given below.

Provider peer determination algorithm

Notation

- r: Maximum block number among blocks that have already been requested.
- S: Set of peers having block j.
- $T_f(j)$: Estimated completion time of retrieval of block j.
- $T_p(j)$: Deadline for retrieval of block j
- S': Set of peers from which a peer can retrieve block j by the deadline $T_p(j)$.
- $R(i)$: Round-trip times to peer i.
- $B(j)$: Size of block j.
- $A(i)$: Available bandwidth from peer i.
- T_{now}: Time when this algorithm is performed.
- $P(j)$: Provider peer for block j.
- $T_r(j)$: Time to request block j.
- k: Round number.

Step 1 Set j to r.

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 the round trip times $R(i)$ to peer i and the transfer time of block j obtained by dividing the block size $B(j)$ by the 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 $\max(T_f(j-1), T_{now} + R(i)) + \frac{B(j)}{A(i)}$, considering the case that the retrieval of block $j-1$ lasts more than $R(i)$ and the request for block j is deferred. If the estimated completion time is smaller 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

Select 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

Select 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 the estimated completion time of retrieval $T_f(j)$ and the time $T_r(j)$ to send a request message for block j as follows.

$$T_f(j) = \max(T_f(j-1), T_{now} + R(P(j))) + \frac{B(j)}{A(P(j))} \quad (8)$$

$$T_r(j) = T_f(j) - R(P(j)) - \frac{B(j)}{A(P(j))} \quad (9)$$

Step 6 If $j = kN$, finish the algorithm and wait for reception of the next response message. 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 Eq. (8). Therefore, if the actual completion time $T_f(j)$ of the retrieval of block j changes, the peer applies the algorithm and determines provider peers.

3. Simulation experiments

In this section, we conduct simulation experiments to evaluate our proposed methods in terms of the amount of search traffic and the continuity of media play-out.

3.1. Simulation model

We use a P2P logical network with 100 peers, which is randomly generated by the Waxman algorithm [11] whose parameters α, β are 0.15, 0.3, respectively. 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 mechanisms, 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.

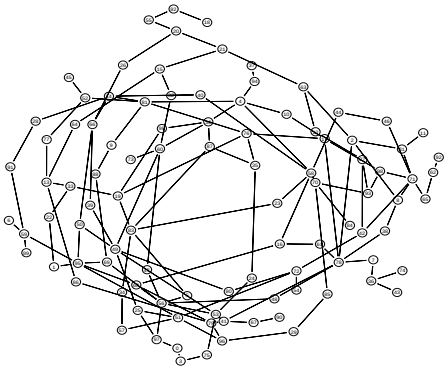


Figure 3. Random network with 100 peers

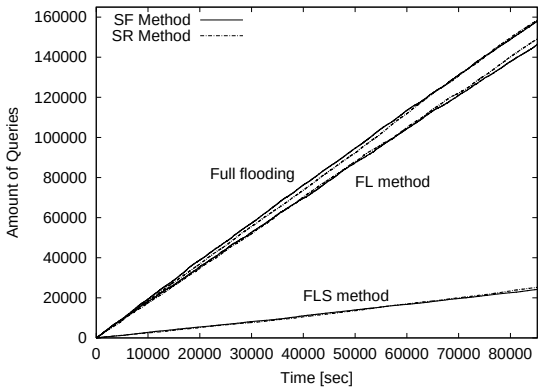


Figure 4. Number of queries

At first, all hundred peers participated in the system, but nobody watched media. One peer begins to request a media stream at a randomly determined time. The inter-arrival time between two successive requests for the first media stream follows the exponential distribution whose average is 20 minutes. Forty media streams of 60 minute length are available. Media streams are numbered from 0 (the most popular) to 39 (the least popular), whose popularity follows a Zipf-like distribution with $\alpha = 1.0$. Each peer watches a media stream without such interactions as rewinding, pausing, or fast-forwarding. Thus, LRU algorithm used for cache replacement becomes identical to FIFO. When a peer finishes watching a media stream, it becomes idle for the waiting time, which also follows the exponential distribution whose average is 20 minutes. A media stream is divided into blocks of 10-sec duration and amounts to 625 KBytes. Each peer sends a query message for a succession of six blocks, i.e., $N = 6$, and retrieves blocks. Blocks obtained are deposited into a cache buffer of 675 MB, which corresponds to three media streams. In the first time of the simulation, each peer stores three whole media streams in its cache buffer. The population of each media stream in the network also follows a Zipf-like distribution whose parameter α is 1.0. 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 propose six possible combinations of search methods, i.e., full flooding only, FL, and FLS, and two block retrieve methods, i.e., SF and SR. We conducted 90 set of simulations for each of six methods and show averaged values in the following figures.

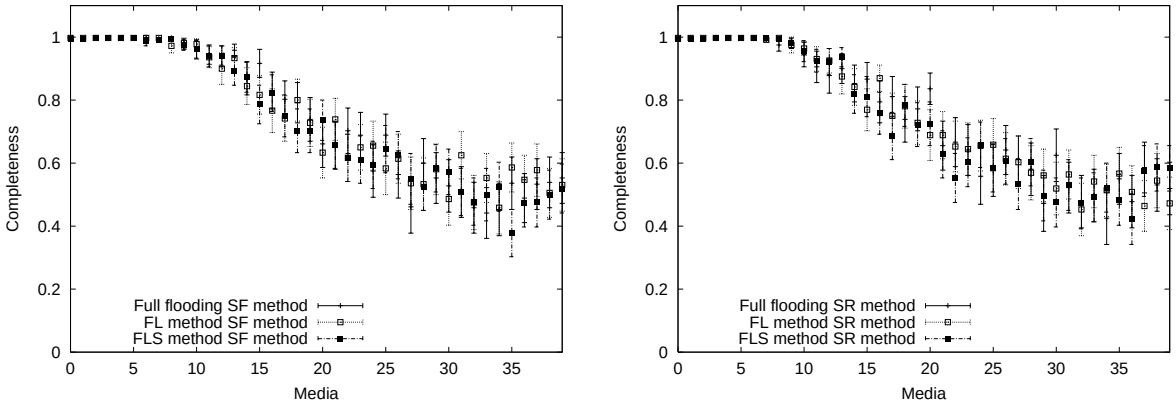
3.2. Evaluation of scalability of search mechanism

First, we evaluate the scalability of our P2P streaming system in terms of the number of queries. Figure 4 illustrates transitions of the average number of queries that a peer receives during the simulation. As shown in Fig. 4, the FL method can slightly reduce the number of queries compared with full flooding. This is because the average number of relays in limited flooding, TTL H', is relatively large in our simulation experiments, independent of block retrieval method. Limited flooding restricts the number of relays to reduce the overhead of searches. 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. The FL method tries full flooding in the first round. Thus, the number of queries cannot be effectively reduced with the FL method. On the other hand, selective search can considerably reduce the number of queries.

3.3. Evaluation of continuity of media play-out

First, we define the waiting time as the time between the emission of the first query message for the media stream and the beginning of reception of the first block. Through simulation experiments, we observe that, independent of method, the waiting time decreases as the popularity increases, and, independent of popularity, all media streams successfully found can be played out within 3.5 sec.

Figures 5(a) and 5(b) illustrate the completeness with 95 % confidence interval of each media stream. 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 Figs. 5(a) and 5(b), independent of method, media streams from 0 to 9 are played out almost continuously from the beginning to the end. On the other hand, as the media popularity decreases, the completeness also deteriorates. Especially in the FLS methods, where query messages are directly sent to a set of peers that are expected to have desired blocks, the completeness is lower than that of the other methods by 0.3 at most. In our experiments, most of the blocks that cannot be retrieved in time are blocks that have already been replaced by blocks of other more popular streams. Since the selective search inquires of the less number of peers cached blocks than that of the other two methods, it is difficult to follow the changes in cached blocks in



(a) SF method

(b) SR method

Figure 5. Completeness

the network. Comparing Figs. 5(a) and 5(b), we find that there is little difference between SF and SR. The reasons are that the remaining time is not used effectively and unexpected cache miss hardly occurs in our experiments.

4. Conclusions

In this paper, we proposed two scalable search methods and two algorithms for block retrieval in scalable and continuous media streaming on P2P networks. Through several simulation experiments, we have shown that the FLS method can provide users with continuous media play-out without introducing extra load on the system.

Several issues still remain to be solved. It was shown that the completeness of unpopular media streams is not high enough. We are now considering an effective cache replacement algorithm that increases the possibility that blocks of unpopular media streams can be found. One candidate is to take into account demand/supply ratio that can be estimated from messages received and relayed. To buffer some of initial blocks and defer the media play-out also contribute to the completeness. We should further consider the trade-off among several QoS requirements such as the completeness, the continuity, and the low-latency of media play-out. For example, at the sacrifice of the continuity, the completeness can be improved. The search mechanism can devote much time to finding blocks by being tolerant with freezes and deferring media play-out. Furthermore, we should evaluate proposed methods in more realistic situations where network conditions dynamically change toward the implementation.

Acknowledgement

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

References

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

[2] vTrails. available at `http://www.vtrails.com`.

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

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

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

[6] 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.

[7] K.-L. Wu, P. S. Yu, and J. L. Wolf, “Segment-based proxy caching of multimedia streams,” in *Proceedings of the 10th International WWW Conference*, pp. 36–44, 2001.

[8] J. Shudong, B. Azer, and I. Arun, “Accelerating internet streaming media delivery using network-aware partial caching,” *Technical Report BUCS-TR-2001-023*, October 2001.

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

[11] B. M. Waxman, “Routing of multipoint connections,” *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1617–1622, Dec. 1988.