Master's Thesis

Title

Design, Implementation and Evaluation of Continuous and Scalable P2P Media Streaming System

Supervisor

Professor Masayuki Murata

Author

Go Yoshida

February 15th, 2005

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Design, Implementation and Evaluation of Continuous and Scalable P2P Media Streaming System

Go Yoshida

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. Peer-to-Peer (P2P) is a new network paradigm designed to solve the problems in the traditional client–server based architecture. 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.

We propose effective 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. 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 showed our proposed methods could accomplish continuous media play-out independent of media popularity.

To verify the practicality of our system, we first design the proposed P2P media streaming system. Then, based on the design, we implement a java-based system on a real environment. Furthermore, we conduct several experiments to evaluate the effectiveness and usefulness of the proposed system. Through evaluations, it is shown that our proposed P2P media streaming system can provide users with a continuous media play-out.

Keywords

P2P
media streaming
continuous media play-out
scalability
streaming caching

Contents

List of Figures

List of Tables

1 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, 4, 5, 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, 4, 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], effective methods for on-demand media streaming on pure P2P networks is proposed. In the 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.

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.

To improve the completeness of media play-out, we also proposed 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 [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, a fully distributed and autonomous cache replacement algorithm, which can adapt to changes in environments, i.e., the supply-to-demand, is proposed. The 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, it is shown that our proposed methods could accomplish continuous media play-out independent of media popularity. In this thesis, to verify the practicality and effectiveness, we design, implementation, and evaluation our proposed system. We first introduce the design of the proposed P2P media streaming system. Then, we implement a java-based system based on the design. Finally, we conduct several experiments to evaluate the effectiveness and usefulness of the proposed system. Through evaluations, it is shown that our proposed P2P media streaming system can provide users with a continuous media play-out.

The rest of this thesis is organized as follows. In section 2, we describe our proposed system for on-demand media streaming on pure P2P networks. In section 3, we explain the design of the P2P media streaming system. Then, section 4 gives several experiments to evaluate our implemented system. Finally, we conclude this thesis and describe future works in section 5.

2 Continuous and Scalable P2P Media Streaming System

2.1 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 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

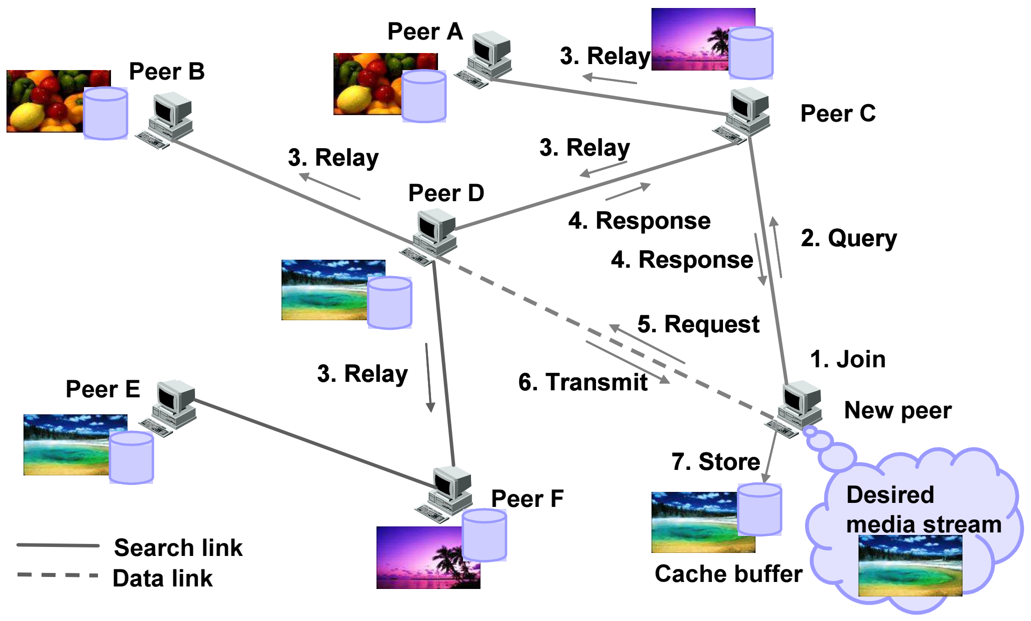

Figure 1: Overview of our media streaming on pure P2P networks

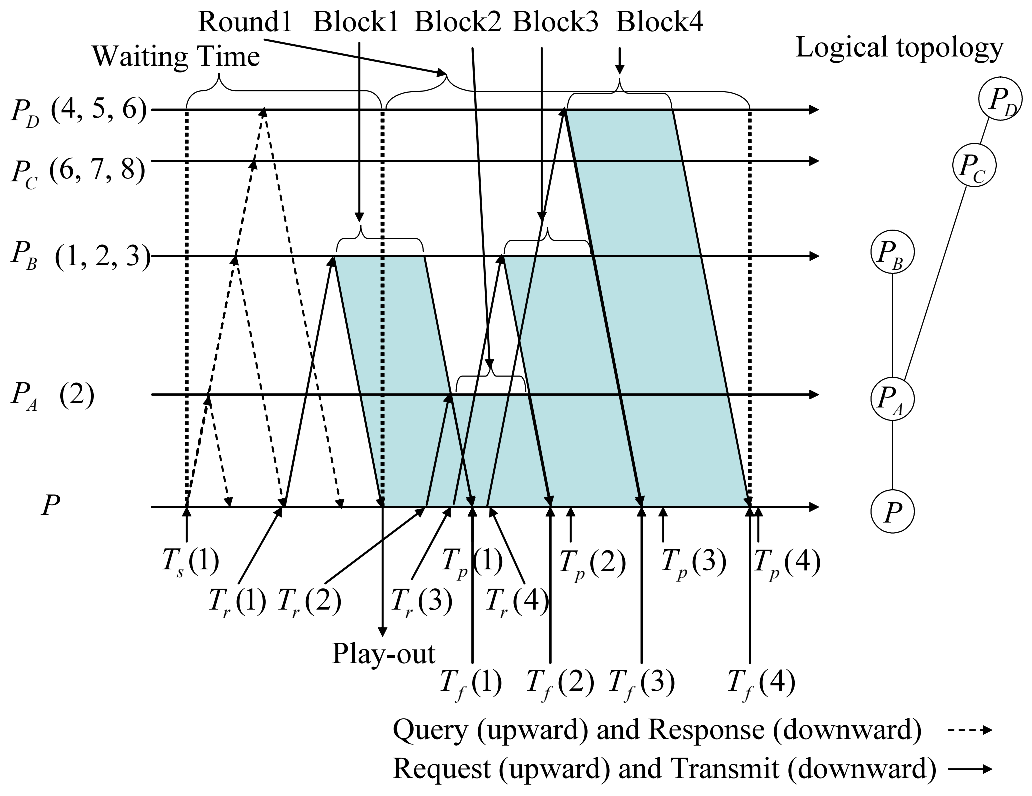

Figure 2: Example of per-group search and retrieval

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.

2.2 Scalable Block Search

Since each peer retrieves a media stream sequentially from the beginning to the end, we can expect that a peer that sent back a 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 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 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., $R = \phi$.

2.3 Block Retrieval for Continuous Media Play-out

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

The deadlines for retrieval of succeeding blocks $j \geq 2$ 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 $T_r(j)$, which will be given by Equation (3), so that it can start receiving block j just after finishing the retrieval of block $j-1$, as shown in 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. The detailed algorithm to determine the provider peer is given below.

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

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

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

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

SF (Select Fastest) Method selects a peer whose estimated completion time is smallest among peers in S'. By retrieving block j as fast as possible, the remainder $T_p(j) - T_f(j)$ can be used to retrieve the succeeding blocks from distant peers or peers with insufficient bandwidth.

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

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

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

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

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

A peer emits a request message for block j to peer $P(j)$ at $T_r(j)$ and sets r to j. On receiving the request, peer $P(j)$ initiates block transmission. If it 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 $j-1$, as in Eq. (2). Therefore, if the actual completion time $T_f(j)$ of the retrieval of block j changes because of changes of network conditions or estimation errors, the peer applies the algorithm and determines provider peers for succeeding blocks. Our proposed algorithm stated above depends on the accuracy of estimation. One of possible solutions to inaccurate estimates is to introduce some reserved time in Eq. (2). In addition, deferment of the play-out also contributes to absorb estimation errors.

2.4 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 [14]. 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 $P(X_i = 0 \to X_i = 1)$ 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 $P(X_i = 1 \to X_i = 0)$ 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 over-estimation, 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:

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

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 $\frac{P_r(i)}{\sum_{i \in M-m} P_r(i)}$. Then, a peer discards blocks from the head or the tail of the stream at random. As in [15], 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.

3 Design of Continuous and Scalable P2P Media Streaming System

In this subsection, we describe the design of our system. Figure 3 illustrates the modules that constitute our P2P media streaming system. Each arrow, dashed arrow, chain arrow, and box corresponds to data flow, message flow, signal, and module, respectively. The media streaming is controlled through a RTSP (Real Time Streaming Protocol) [16]/TCP session between the player and the manager, the query/response messages are transferred over TCP sessions. Since our media streaming system is based on the existing P2P file sharing system, the media data are also transferred over TCP sessions.

A peer participating in our media streaming service first initiates *Manager* module which controls all functions constituting our system, such as, *Participation*, *Block Search*, *Block Retrieval*, and *Block Replacement*. Then, the *Manager* tries to get a candidate list of neighbors. We assume the list can be obtained from a bootstrap server that manages and stores IP addresses of peers joining the streaming service. The *Manager* makes TCP connections with several peers in the obtained list through the *Participation* module. The connected peers become neighbors of the peer and the *Manager* adds them into the *Neighbor Table*. After setting up the TCP connections with neighbors, the user on the peer can start watching its desired media stream. The user sends a request for the desired media stream to the *Manager* through *Player*. We assume that the *Player* is a

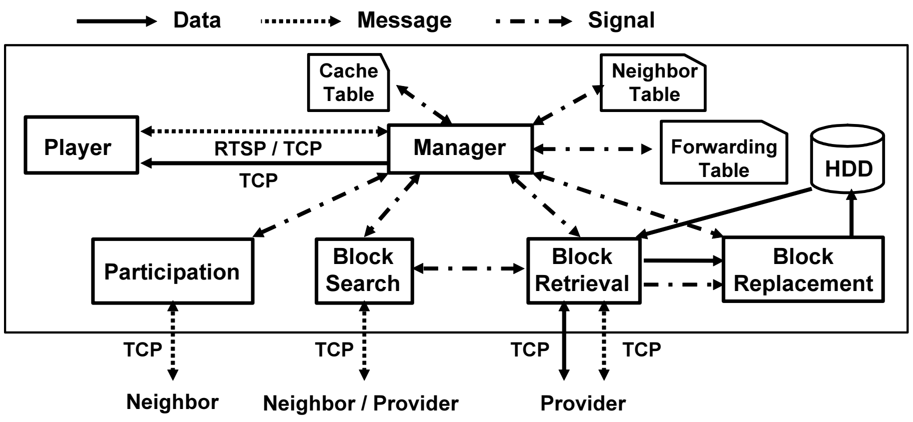

Figure 3: Modules constituting system of our proposed P2P media streaming

typical media player that is compatible with RTSP, such as QuickTime Player [17] and RealOne Player [18]. The *Player* establishes a RTSP session with the *Manager* and sends an RTSP PLAY message only once at the beginning of the session specifying Range. The *Manager* derives the desired blocks based on the specified time range and the block time. Then, the *Manager* sends a query message through the *Block Search* to its neighbors or candidate peers as providers in accordance with the block search method described in subsection 2.2. At the same time, the *Manager* updates its *Forwarding Table* by adding a new entry consisting of the query identifier and the peer identifier to which the query is sent. On receiving the query message, the *Block Search* looks up the *Cache Table* through the *Manager.* If there is the desired block in the cache buffer, the *Block Search* sends a response message back to the peer originating the query message. In the case of flooding based search, the *Block Search* also forwards the query message to the neighbors until the TTL in the query message is greater than zero. On receiving the response message, the *Block Search* looks for the corresponding entry in the *Forwarding Table.* If the response message is not originated by the peer, the *Block Search* forwards it to the peer corresponding to the entry listed in the *Forwarding Table.* Otherwise, it calls the *Block Retrieval* with the information consisting of response message, available bandwidth, and RTT. The *Block Retrieval* determines an appropriate provider peer in accordance with the block retrieval methods described in subsection 2.3. On receiving the block, the *Block Retrieval* passes it to both the *Block Replacement* and the *Manager.* If there is not enough room to store the newly retrieved block in the cache buffer, the *Block Replacement* replaces it with cached blocks by using the cache replacement algorithm in subsection 2.4 and updates the *Cache Table.* At the same time, the *Player* plays out the block obtained from the *Manager.*

3.1 Estimation of Available Bandwidth

In our system, the proposed block retrieval methods rely on the estimation of available bandwidth and RTT from the provider peer. RTT can be estimated by using the time stamp in the query/response message. On the contrary, the estimation of the available bandwidth generally needs much more probe traffic and time than that of RTT. In [19], they propose a new mechanism, called ImTCP (Inline measurement TCP), that estimates the available bandwidth by exploiting data packets transmitted in a TCP connection. Our proposed system is based on the existing P2P file sharing system such as Gnutella, the block is transferred over a TCP connection. By applying the ImTCP to our system, the peer can estimate the available bandwidth without introducing extra load on the network.

We briefly explain the mechanism of ImTCP. ImTCP is based on the probe rate method [20]. Measurement packets are sent by a sender host to a receiver host that immediately transmits the packets back to the sender host. The sender host changes the interval of sending measurement packets. Then, it estimates the available bandwidth of the path based on the arrival intervals of the echoed packets. ImTCP uses ACK packets as the packets sent from a receiver to a sender. During every measurement, ImTCP searches for the available bandwidth only within a given search range. The search range is a range of bandwidth that is expected to include the current available bandwidth. By introducing the search range, ImTCP can avoid sending probe packets at an extremely high rate. Through simulation experiments, it was shown that ImTCP could also keep the number of probe packets for the measurement quite small.

4 Experimental Evaluation of Implemented System

We implement a prototype of our system in Java, that consists of *Manager*, *Participation*, and *Block Retrieval* modules based on the design described the previous section. In this section, we conducted experiments to evaluate the implemented system.

Figure 4 illustrates the configuration for our experimental system. Peer A is connected to peer B and peer C through router A and router B, respectively. Each router can control the available bandwidth of the links to which it connects by using NIST Net [21] that is a network emulator. In the following experiments, a peer uses the bandwidth configured by the NIST Net when it determines a provider peer based on the algorithm described in subsection 2.3. Four media streams of 3 minutes length are available. The media coding rate is set to CBR 500 kbps. A media stream is divided into blocks of 10 sec. Thus, one block amounts to 625 KBytes. Each peer has a cache buffer of 22.5 MB, which corresponds to two media streams. The number of blocks in a round is set to 6.

4.1 Evaluation of Basic Behavior

In this subsection, we conduct the following scenario to evaluate the basic behavior of the block retrieval methods. The available bandwidths between peer A to peer B and between peer A and peer C are set to 700 kbps and 600 kbps, respectively. At first, peer A obtains the search result of peer B, then determines an appropriate provider peer for each block of round 1 as shown in Tab. 1. Note that the search result is locally given at the time scheduled at the start of the experiment because our prototype does not include the *Block*

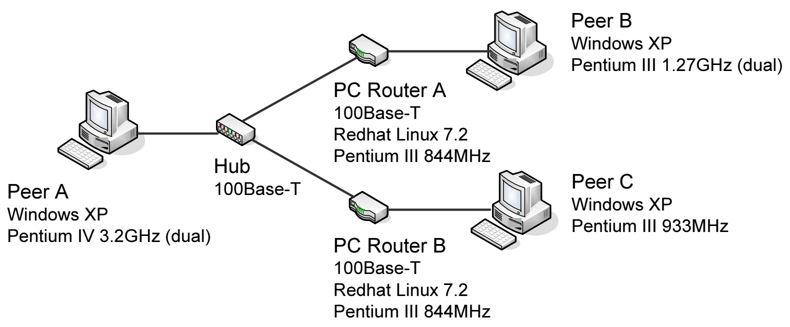

Figure 4: Experimental Environment

Search module. In the table, the block identifier is denoted by X_Y, which means Yth block of media stream X. $x \searrow y$ means blocks from x to y in order of the priority of the block replacement. Block identifiers emphasized by the bold type indicate that those blocks will be retrieved by peer A in the case of SF/SR method. Then, peer A also obtains the search result of peer B and recalculates the provider peer for each block from A_2 to A_6 as shown in Tab. 2. In this case, independent of block retrieval methods, peer A retrieves all blocks from peer B at about 7-sec intervals. In this experiments, blocks of peer B are renewed at 10-sec intervals, which is equal to block time B_t. On the other hand, those of peer C does not change during the experiments, that mean peer C pauses watching a media stream. As a result, peer A obtains the search result of peer B for round 2 as shown in Tab. 3, which corresponds to the 45.45 sec $(= 45.5 - 0.05)$ later condition of Tab. 1. Then, peer A also obtains the search result of peer C, and the provider peer for each block is determined as shown in Tab. 4. In the case of SF method, peer A selects peer B as the provider peer for all blocks because it tries to retrieve a block as fast as possible. On the other hand, in the case of SR method, taking into account the possibility of block disappearance, peer A selects peer C as the provider for each block except for block A_7. This is because it obtains the search result of peer B earlier than that of peer C. In similar fashion, peer A obtains search results at round 3 and determines an appropriate provider peer for each block as shown in Tabs. 5 and 6.

Figures 5 through 8 illustrate the experimental results in the above scenario. Figures 5 and 6 depict T_r, T_f, and T_p for each block in the case of SF and SR method, respectively. As shown in these figures, SF and SR methods can retrieve all blocks in time because the available bandwidths from both provider peers exceed the media coding rate during the experiments. Furthermore, we find that SF method finishes retrieving the last block, i.e. 18, 42.1 sec earlier than the deadline $T_p(18)$. On the contrary, in the case of SR method, the remainder time is 34.2 sec. Figures 7 and 8 depict the reception rate of peer A, which is measured by tcpdump [22] in the case of SF and SR method, respectively. We observe that both block retrieval methods can retrieve all blocks in accordance with the above scenario. Furthermore, we find that the reception rate is slightly lower than the available bandwidth due to the performance limitation of TCP Reno. We expect that this gap can be reduced by using the ImTCP [19] instead of the TCP Reno.

4.2 Evaluation with Changes in Network Condition

In subsection 4.1, we showed both SF method and SR method could accomplish retrieving all blocks in time under an ideal situation where the available bandwidths did not change during the experiments. However, in an actual situation, the available bandwidth changes during the streaming session due to the network congestion caused by background traffic. In such a case, we expect that SF method, which retrieves blocks as fast as possible, is more effective than SR method.

In this subsection, we conduct the following scenario to evaluate the effectiveness of SF method. At the start of the experiments, the available bandwidth between peer A and peer B and between peer A and peer C are set to 800 kbps and 600 kbps, respectively. In this scenario, the block lists that peer B and peer C have do not change during the experiments. The search results obtained at rounds 1, 2, and 3 are shown in Tabs. 7 through 9. At round 1, independent of block retrieval methods, peer A retrieves block A_1 from peer B because it obtains the search result of peer B earlier than that of peer C. For the following blocks from A_2 to A_6, in the case of SF method, peer A selects peer B as the provider to retrieve them as faster as possible. On the other hand, in the case of SR method, it retrieves them from peer C that is more reliable than peer B. Then, at round 2, in the both methods, peer A retrieves block A_7 from peer B because of the same reason at round 1. Here, we depress the available bandwidth between peer A and peer B from 800 kbps to 300 kbps at 65 sec, that is, peer A detects the bandwidth degradation during the retrieval of block A_{10}. Therefore, in the case of SF method, peer A retrieves blocks between A_8 and A_{10} from peer B, then switches the provider peer to peer C whose available bandwidth is 600 kbps. On the contrary, in the case of SR method, peer A keeps to retrieve the succeeding blocks from peer C. At round 3, independent of block retrieval methods, peer A tries to retrieve all blocks from peer B that is the only provider for them. Here, we assume that the available bandwidth between peer A and peer B recovers to 800 kbps at 150 sec, that corresponds to the time when peer A is retrieving block A_{15}. In the case of SF method, peer A can retrieves all blocks in time by using the remainder time accumulated at the previous rounds even when the available bandwidth is far below the media coding rate of 500 kbps. On the contrary, in the case of SR method, peer A

fails to retrieve block A_{15} in time due to the lack of the available bandwidth.

Figures 9 through 12 illustrate the experimental results in the above scenario. Figures 9 and 10 depict T_r, T_f, and T_p for each block in the case of SF and SR methods, respectively. As shown in Fig. 9, peer A can retrieve all blocks in time in the case of SF method. On the other hand, in the case of SR method, it fails to retrieve blocks from A_{15} to A_{18} as shown in Fig. 10. Peer A cannot find any provider peer for block A_{15} until the deadline $T_p(15)$ comes. Although the available bandwidth between peer A and peer B recovers to 800 kbps at 150 sec, peer A cannot detect it because there is no data transmission from peer B after finishing the retrieval of block A_{14}. As a result, it also fails to retrieve the succeeding blocks. This implies the underlying problem in our proposed system, that is, a peer cannot detect the changes in network condition when there is no message/data transmission. To tackle this problem, we can introduce an additional search when a peer cannot find any provider peer during a round.

4.3 Evaluation of Reliability

In subsection 4.1, we found SF method can retrieve blocks faster than SR method under an ideal situation where the blocks that the provider peer has change during the experiments at 10-sec intervals, which is equal to one block time. However, in an actual situation, this assumption is not certainly approved because the provider peer may also retrieves its desired blocks at higher rate than the media coding rate by using SF/SR method. In such a situation, we expect that SR method, which considers the possibility of block disappearance, is more effective than SF method.

In this subsection, we conduct the following scenario to evaluate the effectiveness of SR method. The available bandwidth between peer A and peer B and between peer A and peer C are set to 600 kbps and 560 kbps, respectively. Blocks of peer B are renewed at 6-sec intervals, which is faster than one block time of 10 sec. On the other hand, those of peer C does not change during the experiment. The search results obtained at round 1, 2, and 3 are shown in Tabs. 10 through 12. At round 1, independent of the block retrieval methods, peer A can retrieve all blocks in time. However, at round 2, in the case of SF method, a cache miss occurs to block A_{11} because peer B updates its block list at 6-sec intervals while peer A retrieves blocks at 580 kbps, which corresponds to 8.6-sec intervals. When the cache miss occurs, peer A recalculates the provider peer for block A_{11}. Consequently, it retrieves block A_{11} from another provider peer, that is, peer C. On the other hand, in the case of SR method, peer A can retrieve blocks without any cache misses since it selects peer C, that permanently keeps blocks in its cache buffer, as the provider for all blocks. Similarly, in round 3, SF method fails to retrieve block A_{14} from peer B while SR method does not.

Figures 13 through 16 illustrate the experimental results in the above scenario. Figures 13 and 14 depict T_r, T_f, and T_p for each block in the case of SF and SR method, respectively. On the contrary to our expectation, these figures show that SF method can retrieve blocks faster than SR method. In the case of SF method, there are 4 cache misses to blocks A_{11}, A_{12}, A_{14}, and A_{16} as shown in Fig. 15. When a cache miss occurs, peer A obtains a latest block list from the current provider, i.e. peer B. Then, it recalculates the provider peer and retrieves the block from a new provider peer. Consequently, it takes at least the sum of one-way delay from the current provider peer and RTT from the new provider peer. However, Fig. 15 shows that there are almost no degradation of the reception rate when cache misses occur. As mentioned above, the actual reception rate is slightly lower than the available bandwidth in our experiments, thus, the estimated completion time of the block retrieval always becomes earlier than the actual completion time of the block retrieval. Since peer A sends a request to the provider peer at the calculated time based on the estimated completion time, it notices the cache miss before finishing the retrieval of the previous block. As a result, it can retrieve the following block without almost no delay despite of the cache miss. Even if the reception rate is equal to the available bandwidth, we expect that SF method is more effective than SR method because the time needed to switch the provider peer is less than 1 sec in our experimental environment.

One possible solution to reduce the probability of cache miss is following. A peer can conjecture the behavior of the provider peer taking into account the difference between search results of two successive rounds, that is, it can estimate the change interval of the block list of the provider peer. By considering the estimated interval when the peer determines a provider peer, we can reduce the probability of cache misses.

Table 1: Search Results of Peer B at Round 1 (Basic Behavior)

Block Retrieval Method		SF Method	SR Method
Peer Identifier		Peer B	Peer B
Priority of Block Replacement	1-3	$B_{16} \searrow B_{18}$	$B_{16} \searrow B_{18}$
	4-9	$\boldsymbol{A_1 \searrow A_6}$	$\boldsymbol{A_1 \searrow A_6}$
	10-18	$A_7 \searrow A_{15}$	$A_7 \searrow A_{15}$
	19-36	$C_1 \searrow C_{18}$	$C_1 \searrow C_{18}$

Table 2: Search Results of Peer B and Peer C at Round 1 (Basic Behavior)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-3	$B_{16} \searrow B_{18}$	$A_1 \searrow A_3$	$B_{16} \searrow B_{18}$	$A_1 \searrow A_3$
	4-9	$\boldsymbol{A_1 \searrow A_6}$	$A_4 \searrow A_9$	$\boldsymbol{A_1 \searrow A_6}$	$A_4 \searrow A_9$
	10-18	$A_7 \searrow A_{15}$	$A_{10} \searrow A_{18}$	$A_7 \searrow A_{15}$	$A_{10} \searrow A_{18}$
	19-36	$C_1 \searrow C_{18}$	$B_1 \searrow B_{18}$	$C_1 \searrow C_{18}$	$B_1 \searrow B_{18}$

Table 3: Search Results of Peer B at Round 2 (Basic Behavior)

Block Retrieval Method		SF Method	SR Method
Peer Identifier		Peer B	Peer B
Priority of Block Replacement	1-5	$A_2 \searrow A_6$	$A_2 \searrow A_6$
	6-11	$\boldsymbol{A_7 \searrow A_{12}}$	$\boldsymbol{A_7 \searrow A_{12}}$
	12-14	$A_{13} \searrow A_{15}$	$A_{13} \searrow A_{15}$
	15-32	$C_1 \searrow C_{18}$	$C_1 \searrow C_{18}$
	33-36	$D_1 \searrow D_4$	$D_1 \searrow D_4$

Table 4: Search Results of Peer B and Peer C at Round 2 (Basic Behavior)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-5	$A_2 \searrow A_6$	$A_1 \searrow A_5$	$A_2 \searrow A_6$	$A_1 \searrow A_5$
	6	$\boldsymbol{A_7}$	A_6	$\boldsymbol{A_7}$	A_6
	7	$\boldsymbol{A_8}$	A_7	A_8	A_7
	8-11	$\boldsymbol{A_9 \searrow A_{12}}$	$A_8 \searrow A_{11}$	$A_9 \searrow A_{12}$	$\boldsymbol{A_8 \searrow A_{11}}$
	12	A_{13}	A_{12}	A_{13}	$\boldsymbol{A_{12}}$
	13-14	$A_{14} \searrow A_{15}$	$A_{13} \searrow A_{14}$	$A_{14} \searrow A_{15}$	$A_{13} \searrow A_{14}$
	15-18	$C_1 \searrow C_4$	$A_{15} \searrow A_{18}$	$C_1 \searrow C_4$	$A_{15} \searrow A_{18}$
	19-32	$C_5 \searrow C_{18}$	$B_1 \searrow B_{14}$	$C_5 \searrow C_{18}$	$B_1 \searrow B_{14}$
	33-36	$D_1 \searrow D_4$	$B_{15} \searrow B_{18}$	$D_1 \searrow D_4$	$B_{15} \searrow B_{18}$

Table 5: Search Results of Peer B at Round 3 (Basic Behavior)

Block Retrieval Method		SF Method	SR Method
Peer Identifier		Peer B	Peer B
Priority of Block Replacement	1-7	$A_6 \searrow A_{12}$	$A_6 \searrow A_{12}$
	8-10	$\boldsymbol{A_{13} \searrow A_{15}}$	$\boldsymbol{A_{13} \searrow A_{15}}$
	11-28	$C_1 \searrow C_{18}$	$C_1 \searrow C_{18}$
	27-36	$D_1 \searrow D_8$	$D_1 \searrow D_8$

Table 6: Search Results of Peer B and Peer C at Round 3 (Basic Behavior)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-7	$A_6 \searrow A_{12}$	$A_1 \searrow A_7$	$A_6 \searrow A_{12}$	$A_1 \searrow A_7$
	8	$\boldsymbol{A_{13}}$	A_8	$\boldsymbol{A_{13}}$	A_8
	9-10	$\boldsymbol{A_{14} \searrow A_{15}}$	$A_9 \searrow A_{10}$	$A_{14} \searrow A_{15}$	$A_9 \searrow A_{10}$
	11-13	$C_1 \searrow C_3$	$A_{11} \searrow A_{13}$	$C_1 \searrow C_3$	$A_{11} \searrow A_{13}$
	14-15	$C_4 \searrow C_5$	$A_{14} \searrow A_{15}$	$C_4 \searrow C_5$	$\boldsymbol{A_{14} \searrow A_{15}}$
	16-18	$C_6 \searrow C_8$	$A_{16} \searrow A_{18}$	$C_6 \searrow C_8$	$\boldsymbol{A_{16} \searrow A_{18}}$
	19-28	$C_9 \searrow C_{18}$	$B_1 \searrow B_{10}$	$C_9 \searrow C_{18}$	$B_1 \searrow B_{10}$
	29-36	$D_1 \searrow D_8$	$B_{11} \searrow B_{18}$	$D_1 \searrow D_8$	$B_{11} \searrow B_{18}$

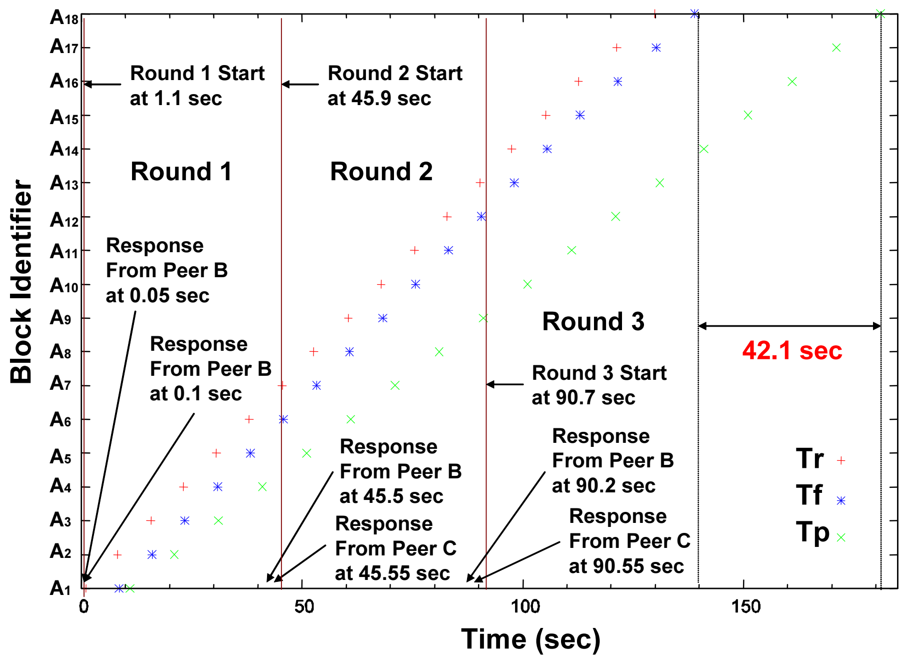

Figure 5: Flow of Block Retrieval (Basic Behavior, SF)

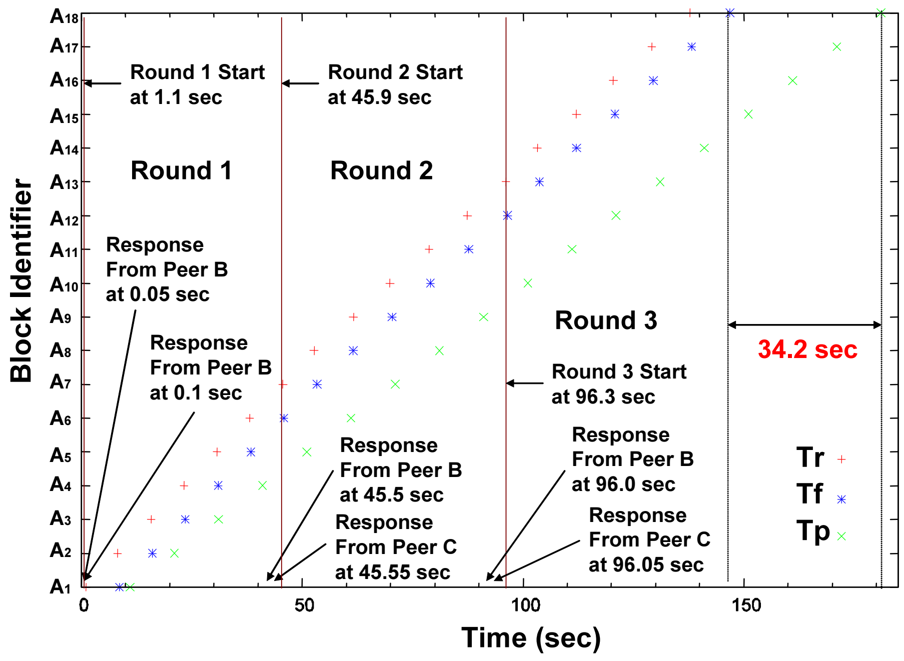

Figure 6: Flow of Block Retrieval (Basic Behavior, SR)

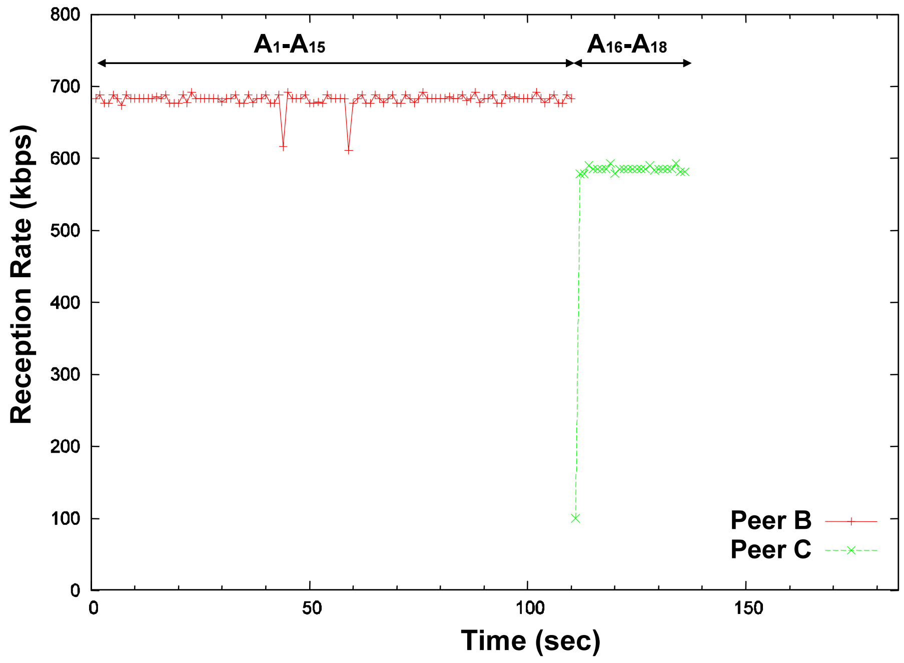

Figure 7: Reception Rate (Basic Behavior, SF)

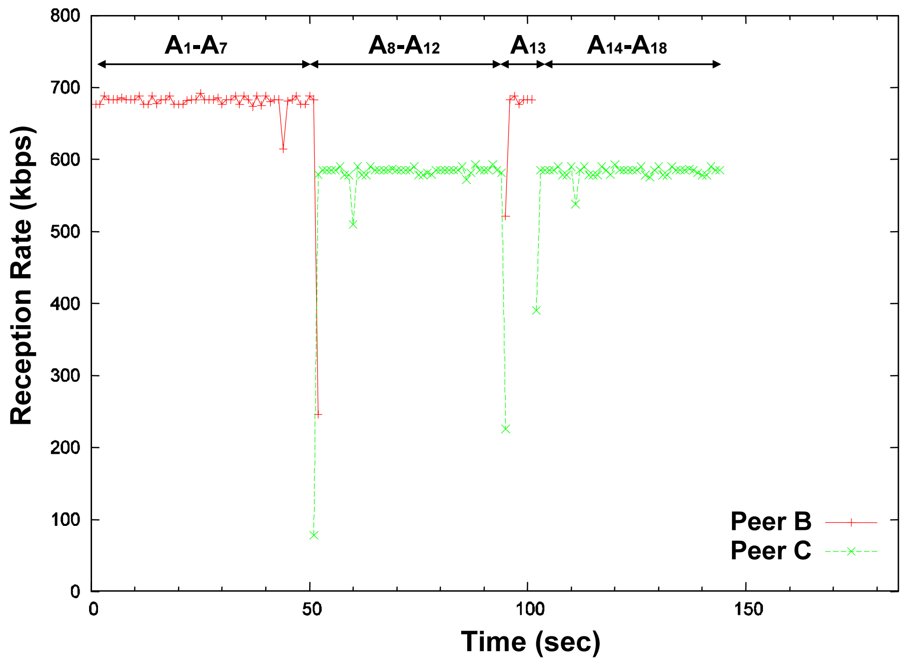

Figure 8: Reception Rate (Basic Behavior, SR)

Table 7: Search Results of Peer B and Peer C at Round 1 (Changes in Network Condition)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1	$\boldsymbol{A_1}$	B_1	$\boldsymbol{A_1}$	B_1
	2-6	$\boldsymbol{A_2 \searrow A_6}$	$B_2 \searrow B_6$	$A_2 \searrow A_6$	$B_2 \searrow B_6$
	7-18	$A_7 \searrow A_{18}$	$B_7 \searrow B_{18}$	$A_7 \searrow A_{18}$	$B_7 \searrow B_{18}$
	19	B_1	A_1	B_1	A_1
	20-24	$B_2 \searrow B_6$	$A_2 \searrow A_6$	$B_2 \searrow B_6$	$\boldsymbol{A_2 \searrow A_6}$
	25-30	$B_7 \searrow B_{12}$	$A_7 \searrow A_{12}$	$B_7 \searrow B_{12}$	$A_7 \searrow A_{12}$
	31-36	$B_3 \searrow B_{18}$	$C_1 \searrow C_6$	$B_3 \searrow B_{18}$	$C_1 \searrow C_6$

Table 8: Search Results of Peer B and Peer C at Round 2 (Changes in Network Condition)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-6	$A_1 \searrow A_6$	$B_1 \searrow B_6$	$A_1 \searrow A_6$	$B_1 \searrow B_6$
	7	$\boldsymbol{A_7}$	B_7	$\boldsymbol{A_7}$	B_7
	8-10	$\boldsymbol{A_8 \searrow A_{10}}$	$B_8 \searrow B_{10}$	$A_8 \searrow A_{10}$	$B_8 \searrow B_{10}$
	11-12	$A_{11} \searrow A_{12}$	$B_{11} \searrow B_{12}$	$A_{11} \searrow A_{12}$	$B_{11} \searrow B_{12}$
	13-18	$A_{13} \searrow A_{18}$	$B_{13} \searrow B_{18}$	$A_{13} \searrow A_{18}$	$B_{13} \searrow B_{18}$
	19-24	$B_1 \searrow B_6$	$A_1 \searrow A_6$	$B_1 \searrow B_6$	$A_1 \searrow A_6$
	25	B_7	A_7	B_7	A_7
	26-28	$B_8 \searrow B_{10}$	$A_8 \searrow A_{10}$	$B_8 \searrow B_{10}$	$\boldsymbol{A_8 \searrow A_{10}}$
	29-30	$B_{11} \searrow B_{12}$	$\boldsymbol{A_{11} \searrow A_{12}}$	$B_{11} \searrow B_{12}$	$\boldsymbol{A_{11} \searrow A_{12}}$
	31-36	$B_{13} \searrow B_{18}$	$C_1 \searrow C_6$	$B_{13} \searrow B_{18}$	$C_1 \searrow C_6$

Table 9: Search Results of Peer B and Peer C at Round 3 (Changes in Network Condition)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-12	$A_1 \searrow A_{12}$	$B_1 \searrow B_{12}$	$A_1 \searrow A_{12}$	$B_1 \searrow B_{12}$
	13-18	$\boldsymbol{A_{13} \searrow A_{18}}$	$B_{13} \searrow B_{18}$	$\boldsymbol{A_{13} \searrow A_{18}}$	$B_{13} \searrow B_{18}$
	19-30	$B_1 \searrow B_{12}$	$A_1 \searrow A_{12}$	$B_1 \searrow B_{12}$	$A_1 \searrow A_{12}$
	31-36	$B_{13} \searrow B_{18}$	$C_1 \searrow C_6$	$B_{13} \searrow B_{18}$	$C_1 \searrow C_6$

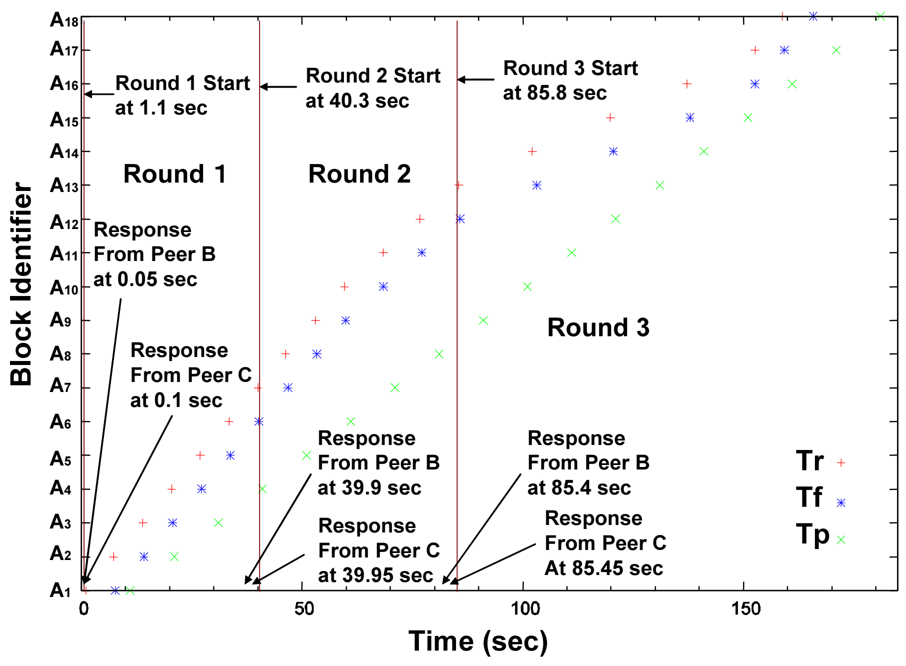

Figure 9: Flow of Block Retrieval (Changes in Network Condition, SF)

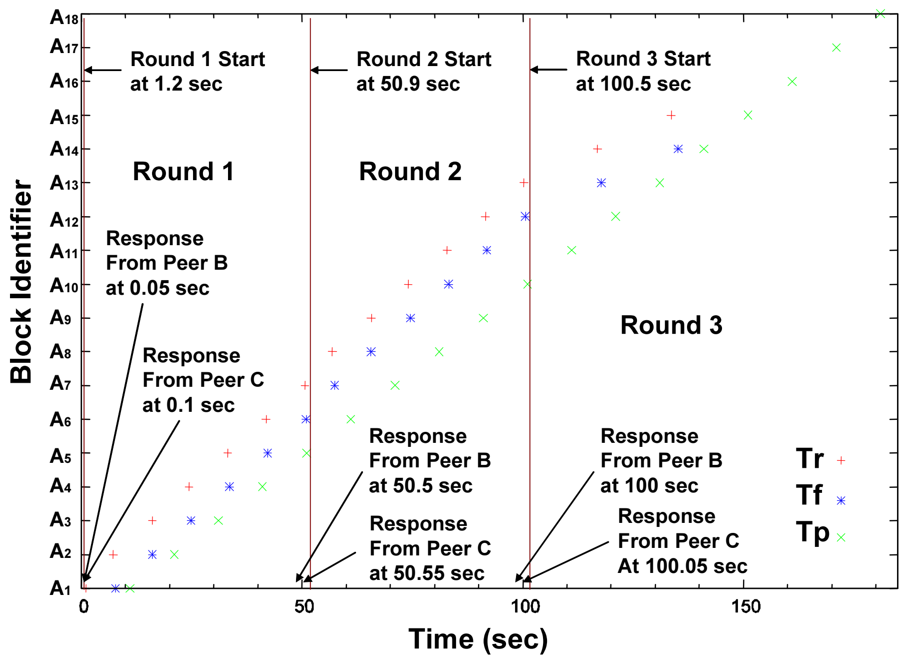

Figure 10: Flow of Block Retrieval (Changes in Network Condition, SR)

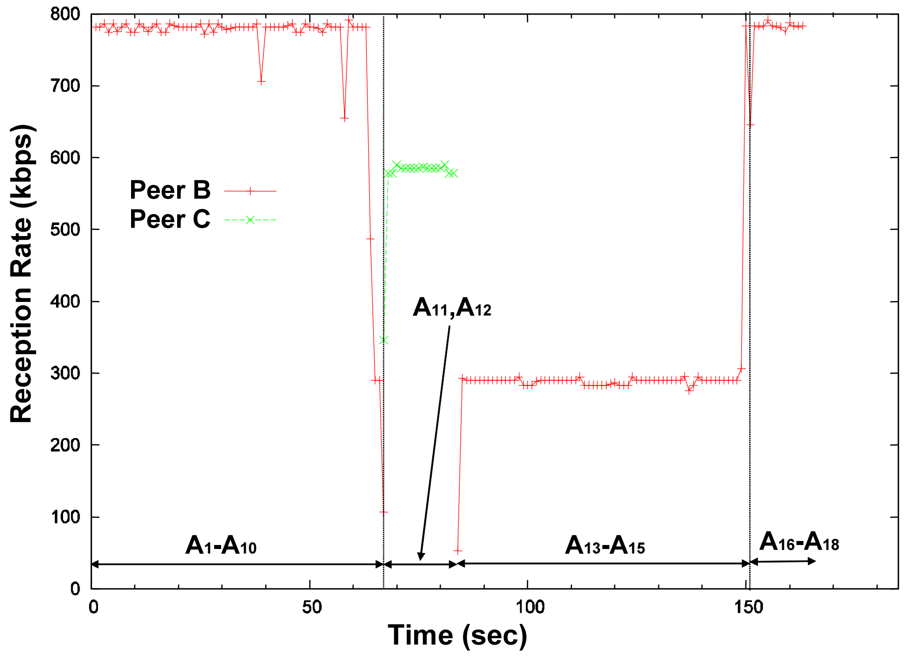

Figure 11: Reception Rate (Changes in Network Condition, SF)

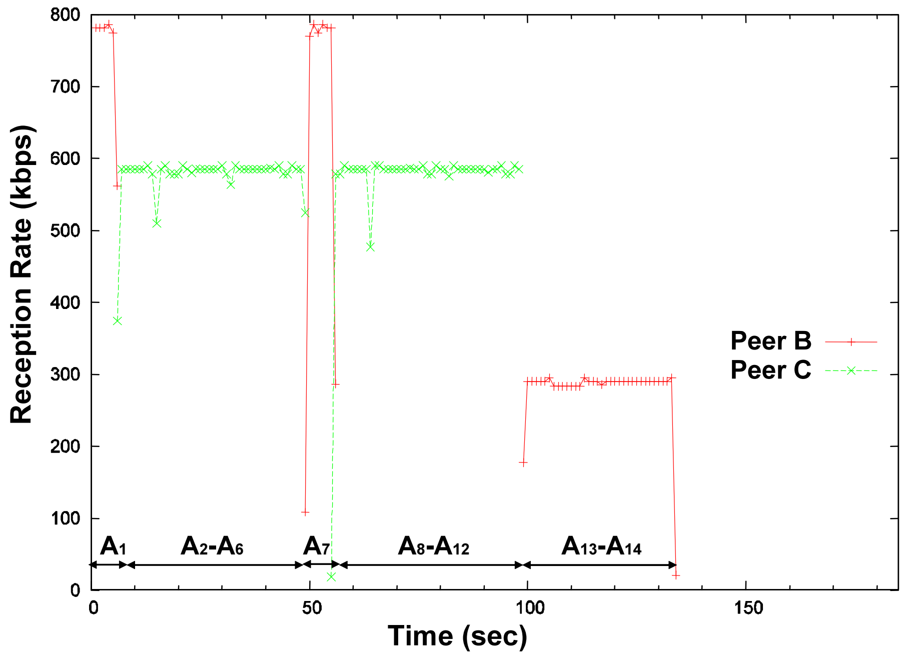

Figure 12: Reception Rate (Changes in Network Condition, SR)

Table 10: Search Results of Peer B and Peer C at Round 1 (Reliability)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-4	$B_{15} \searrow B_{18}$	$A_1 \searrow A_4$	$B_{15} \searrow B_{18}$	$A_1 \searrow A_4$
	5-10	$\boldsymbol{A_1 \searrow A_6}$	$A_5 \searrow A_{10}$	$\boldsymbol{A_1 \searrow A_6}$	$A_5 \searrow A_{10}$
	11-18	$A_7 \searrow A_{14}$	$A_{11} \searrow A_{18}$	$A_7 \searrow A_{14}$	$A_{11} \searrow A_{18}$
	19-22	$A_{15} \searrow A_{18}$	$B_1 \searrow B_4$	$A_{15} \searrow A_{18}$	$B_1 \searrow B_4$
	23-36	$C_1 \searrow C_{14}$	$B_5 \searrow B_{18}$	$C_1 \searrow C_{14}$	$B_5 \searrow B_{18}$

Table 11: Search Results of Peer B and Peer C at Round 2 (Reliability)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-2	$A_5 \searrow A_6$	$A_1 \searrow A_2$	$A_5 \searrow A_6$	$A_1 \searrow A_2$
	3	$\boldsymbol{A_7}$	A_3	$\boldsymbol{A_7}$	A_3
	4-5	$\boldsymbol{A_8 \searrow A_9}$	$A_4 \searrow A_5$	$A_8 \searrow A_9$	$A_4 \searrow A_5$
	6-7	$\boldsymbol{A_{10} \searrow A_{11}}$	$A_6 \searrow A_7$	$A_{10} \searrow A_{11}$	$A_6 \searrow A_7$
	8	$\boldsymbol{A_{12}}$	A_8	A_{12}	$\boldsymbol{A_8}$
	9-12	$A_{13} \searrow A_{16}$	$A_9 \searrow A_{12}$	$A_{13} \searrow A_{16}$	$\boldsymbol{A_9 \searrow A_{12}}$
	13-14	$A_{17} \searrow A_{18}$	$A_{13} \searrow A_{14}$	$A_{17} \searrow A_{18}$	$A_{13} \searrow A_{14}$
	15-18	$C_1 \searrow C_4$	$A_{15} \searrow A_{18}$	$C_1 \searrow C_4$	$A_{15} \searrow A_{18}$
	19-32	$C_5 \searrow C_{18}$	$B_1 \searrow B_{14}$	$C_5 \searrow C_{18}$	$B_1 \searrow B_{14}$
	33-36	$D_1 \searrow D_4$	$B_{15} \searrow B_{18}$	$D_1 \searrow D_4$	$B_{15} \searrow B_{18}$

Table 12: Search Results of Peer B and Peer C at Round 3 (Reliability)

Block Retrieval Method		SF Method		SR Method	
Peer Identifier		Peer B	Peer C	Peer B	Peer C
Priority of Block Replacement	1-5	$\boldsymbol{A_{14} \searrow A_{18}}$	$A_1 \searrow A_5$	$A_{14} \searrow A_{18}$	$A_1 \searrow A_5$
	6-12	$C_1 \searrow C_7$	$A_6 \searrow A_{12}$	$C_1 \searrow C_7$	$A_6 \searrow A_{12}$
	13	C_8	$\boldsymbol{A_{13}}$	C_8	$\boldsymbol{A_{13}}$
	14-18	$C_9 \searrow C_{13}$	$A_{14} \searrow A_{18}$	$C_9 \searrow C_{13}$	$\boldsymbol{A_{14} \searrow A_{18}}$
	19-23	$C_{14} \searrow C_{18}$	$B_1 \searrow B_5$	$C_{14} \searrow C_{18}$	$B_1 \searrow B_5$
	24-36	$D_1 \searrow D_{13}$	$B_6 \searrow B_{18}$	$D_1 \searrow D_{13}$	$B_6 \searrow B_{18}$

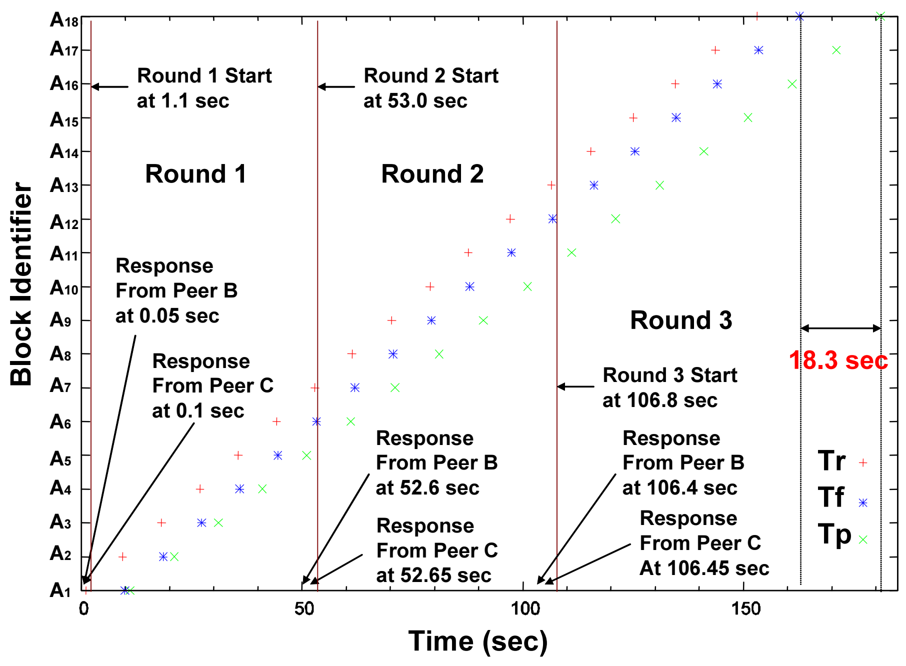

Figure 13: Flow of Block Retrieval (Reliability, SF)

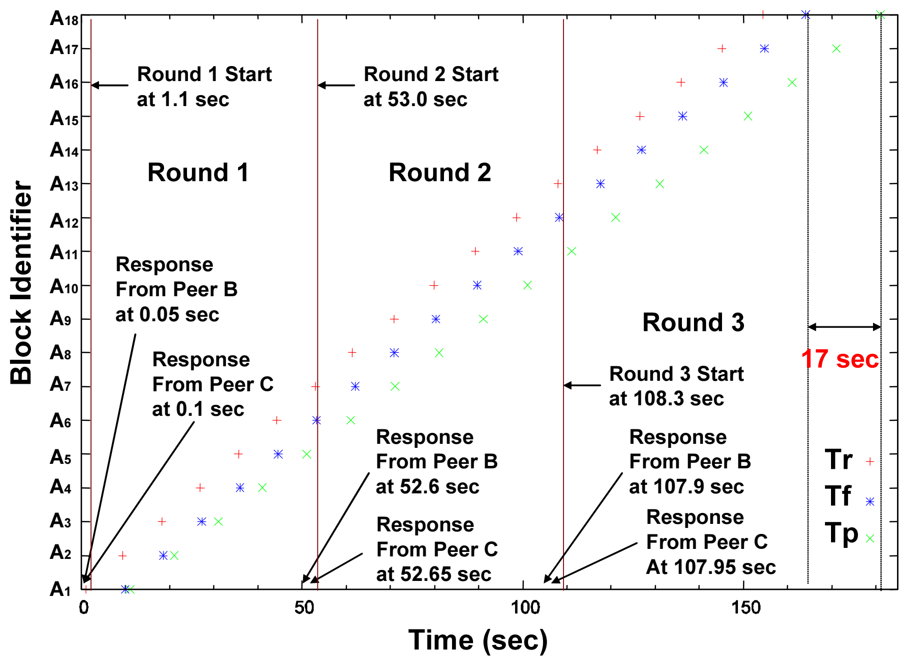

Figure 14: Flow of Block Retrieval (Reliability, SR)

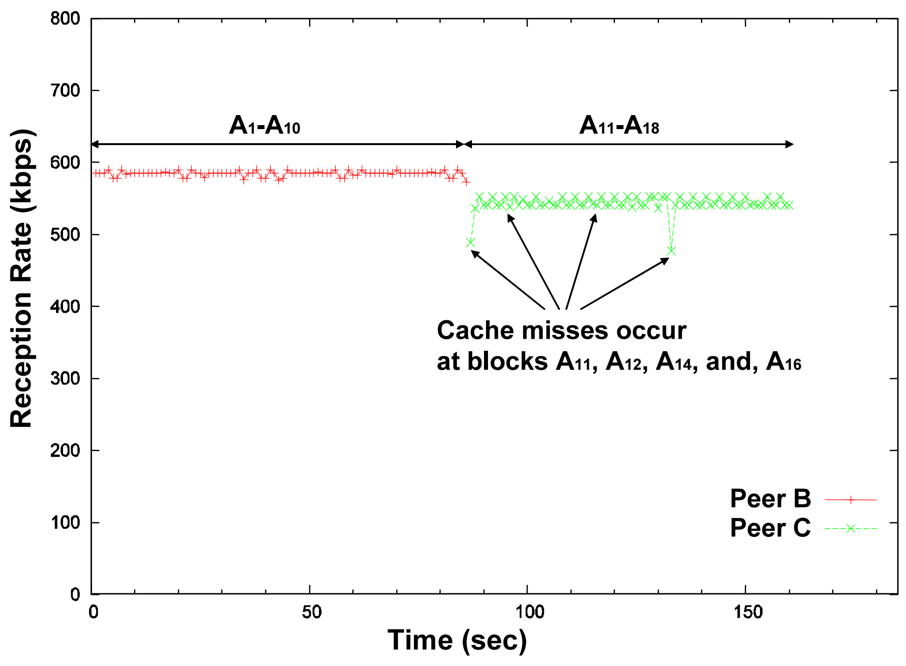

Figure 15: Reception Rate (Reliability, SF)

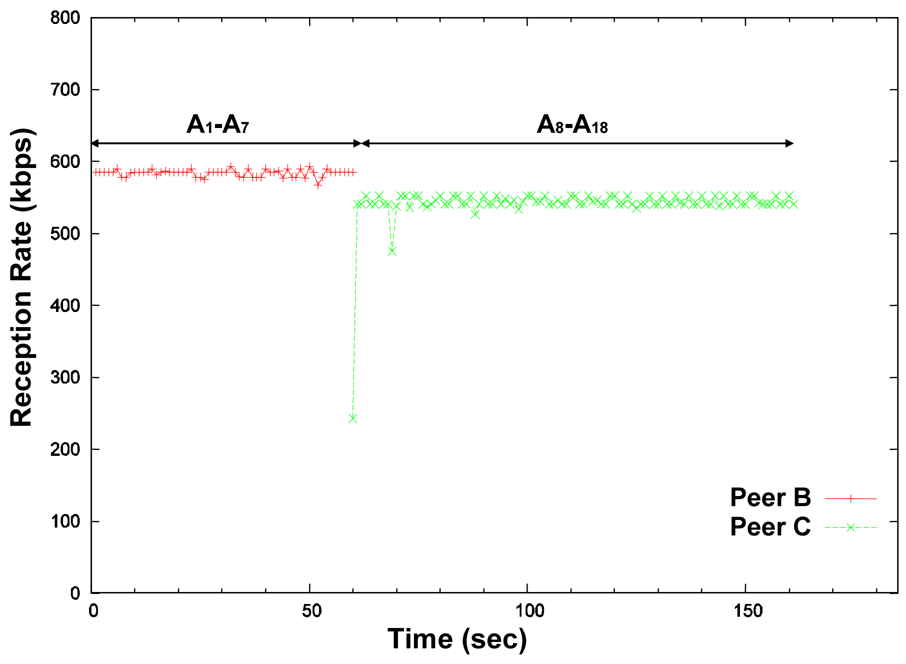

Figure 16: Reception Rate (Reliability, SR)

5 Conclusion

In this thesis, we first designed our P2P media streaming system considering the compatibility with the existing media player and protocol. Then, we implemented a java-based prototype of our system on a real environment. Through several experimental evaluations, it was shown that our P2P media streaming system can provide users with a continuous media streaming service. In particular, SF method is more superior to SR method in various situations.

As future research works, we have to implement the rest of the designed system and evaluate the implemented system in large-scale networks. We expect that a network emulator developed by our research group can be used to the evaluation. Furthermore, we would also need to improve the proposed methods taking into account the experimental results.

Acknowledgements

I would like to express my appreciation to my supervisor, Professor Masayuki Murata of Osaka University, who advised and led me about not only the daily research but also many various aspects of the lifestyle through three years.

I am most grateful to Associate Professor Naoki Wakamiya of Osaka University, for his kindness, pointed advice, and support. I acknowledge for his appropriate guidance for my research.

I appreciate Research Associate Masahiro Sasabe of Osaka University, who supported my research overall and encouraged me to achieve this thesis at every moment. I could accomplish this thesis due to his help.

I extend my deepest appreciation to Associate Professor of Hiroyuki Ohsaki, Associate Professor Go Hasegawa, Research Associate Shin'ichi Arakawa, and Research Associate Ichinoshin Maki of Osaka University for all kinds of their support.

Finally, I thank many friends and colleagues in the Department of Infomation Networking Graduate School of Information Science and Technology of Osaka University and my family for their assist and advice.

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, pp. 1849–1858, June 2003.

[2] J. Liu and J. Xu, "Proxy caching for media streaming over the internet," *IEEE Communications Magazine*, vol. 42, pp. 88–94, Aug. 2004.

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

[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, 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), pp. 45–54, Nov. 2003.

[7] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara, "Effective methods for scalable and continuous media streaming on peer-to-peer networks," *European Transactions on Telecommunications*, vol. 15, pp. 549–558, Nov. 2004.

[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), pp. 36–45, 1997.

[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, “ImTCP: TCP with an inline measurement mechanism for available bandwidth,” submitted to *the Internet Measurement Conference IMC-2004*, Oct. 2004.

[14] 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), pp. 92–99, Sept. 2003.

[15] 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.

[16] H. Schulzrinne, A. Rao, and R. Lanphier, “Real Time Streaming Protocol, (RTSP,),” *Internet Request for Comments 2326*, April 1998.

[17] “QuickTime Player.” available at `http://www.apple.com/quicktime/`.

[18] “RealOne Player.” available at `http://www.real.com/`.

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

[20] M. Jain and C. Dovrolis, “Pathload: A measurement tool for end-to-end available bandwidth,” in *Proceedings of Passive and Active Measurement Workshop (PAM’02)*, (Fort Collins), Mar. 2002.

[21] “NIST Net.” available at `http://www-x.antd.nist.gov/nistnet/`.

[22] “The protocol packet capture and dumper program - tcpdump.” available at `http://www.tcpdump.org/`.