Master's Thesis

Title

Design, Implementation, and Evaluation of Proxy Caching Mechanisms for Video Streaming Services

Supervisor

Professor Masayuki Murata

Author

Yoshiaki Taniguchi

February 13th, 2004

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Design, Implementation, and Evaluation of Proxy Caching Mechanisms
for Video Streaming Services

Yoshiaki Taniguchi

Abstract

The proxy mechanism widely used in WWW systems offers low-delay and scalable delivery of data by means of a "proxy server." By applying a proxy mechanism to video streaming systems, high-quality and low-delay video distribution can be accomplished without imposing extra load on the system.

We have proposed proxy caching mechanisms to accomplish the high-quality and highly interactive video streaming services. In our proposed mechanisms, a video stream is divided into blocks for efficient use of the cache buffer and the bandwidth. A proxy retrieves a block from a server, deposits it in its local cache buffer, and provides a requesting client with the block in time. It maintains the cache with limited capacity by replacing unnecessary cached blocks with a newly retrieved block. The proxy cache server also prefetches video blocks that are expected to be required in the near future in order to avoid cache misses. In addition, the quality of cached video data can be adapted appropriately in the proxy to cope with the client-to-client heterogeneity, in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality. Furthermore, we have proposed cooperative proxy caching mechanisms where proxies communicate with each other, and retrieve missing video data from an appropriate server by taking into account transfer delay and offerable quality.

In this thesis, to verify the practicality of our mechanisms, we design and implement them on a real system for MPEG-4 (Moving Picture Experts Group) video streaming services, and conduct experiments. We employ off-the-shelf and common applications for the server and client systems. Through evaluations conducted from several performance aspects, it is shown that our proxy caching system can provide users with a continuous and

high-quality video streaming service in a heterogeneous environment. In the experiment, a proxy retrieves a video block of a desired level of quality from an appropriately chosen server taking into account network conditions and cache conditions. All blocks arrive at a requesting client in time. Furthermore, the practicality of our design and usefulness of our mechanisms are also verified.

Keywords

video streaming service, proxy caching, quality adjustment, MPEG-4

Contents

List of Figures

List of Tables

1 Introduction

With the increase in computing power and the proliferation of the Internet, video streaming services have become widely deployed. Through these services, the client receives a video stream from a video server over the Internet and plays it as it gradually arrives. However, on the current Internet, only the best-effort service, where there is no guarantee on bandwidth, delay, or packet loss probability, is still the major transport mechanism. Consequently, video streaming services cannot provide clients with continuous or reliable video streams. As a result, the perceived quality of video streams played at the client cannot satisfy expectations, and freezes, flickers, and long pauses are experienced. Furthermore, most of today's Internet streaming services lack scalability against increased clients since they have been constructed on a client-server architecture. A video server must be able to handle a large number of clients simultaneously. It injects a considerable amount of video traffic along the path to distant clients, and the network becomes seriously congested.

The proxy mechanism widely used in WWW systems offers low-delay and scalable delivery of data by means of a "proxy server." The proxy server caches multimedia data that have passed through it in its local buffer, called the "cache buffer," and it then provides the cached data to users on demand. Furthermore, in today's Internet, a cooperative proxy caching technique is also used to share cached data among proxies [1]. By applying these proxy mechanisms to video streaming systems, high-quality and low-delay video distribution can be accomplished without imposing an extra load on the system.

However, existing caching architectures are not intended for video streaming services and cannot be directly applied to video delivery. Once a user starts to watch a video stream, the service provider should guarantee continuity of the video playback. Each frame or block of the stream must arrive at a client before a user watches the part. Furthermore, clients are heterogeneous, in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality. Since receiving and decoding video streams put large loads on both of the networks and end-systems, the affordable video rate and quality also differ from client-to-client. Several papers [2-7] describe proxy caching mechanisms for video streaming services. However, they do not consider client-to-client

heterogeneity or lack the scalability and adaptability to rate and quality variations due to layered video coding algorithm.

In our previous research work [8], we proposed proxy caching mechanisms to accomplish high-quality and low-delay video streaming service in a heterogeneous environment. In our proposed mechanisms, a video stream is divided into blocks so that the cache buffer and the bandwidth can be used efficiently. A proxy retrieves a block from the server, deposits it in its local cache buffer, and provides the requesting client with the block in time. It maintains the cache with a limited capacity by replacing unnecessary cached blocks with a newly retrieved block. The proxy cache server also prefetches video blocks that are expected to be required in the near future to avoid cache misses. The proxy server also adjusts the quality of a cached or retrieved video block to the appropriate level through video filters to handle client-to-client heterogeneity. We also implemented the mechanisms on a real system. The system consisted of a video server, a proxy, client programs, and was built for delivery of an MPEG-2 video stream. Through simulations and experiments, it was shown that our proposed mechanisms can provide users with a low-latency and high-quality video streaming service in a heterogeneous environment. However, we put some assumptions on client and server systems. For example, a client sent requests on a block-by-block basis and a server was capable of adjusting the quality of a video stream. They were not the norm in today's Internet video streaming services. Thus, the practicality and adaptability of our systems to existing video streaming services were not verified. In [9, 10], we considered the cooperation among proxies to provide further effective and high-quality video streaming service. We proposed cooperative proxy caching mechanisms and showed their effectiveness through simulation experiments. However, several assumptions were made to evaluate the ideal performance of our system. For example, it was assumed that a server and proxies could send video data at an arbitrary rate. However, in most of today's Internet streaming services, a server application usually sends a video stream on a frame-by-frame basis at the rate of the frame rate. Furthermore, The practicality and adaptability of our mechanisms to existing video streaming services were also not verified.

In this thesis, first, we design and implement our proxy caching mechanisms [8] for MPEG-4 video streaming services on a real system which is constructed on off-the-shelf and prevailing products. We employ the Darwin Streaming Server [11] as the server ap-

plication and RealOne Player [12] and QuickTime Player [13] as the client applications. There are problems in implementing our mechanisms that consider some ideal conditions and environments, since we use off-the-shelf and prevailing products. We design the system to implement our mechanisms for MPEG-4 video streaming service, and implement it on a real system. Our implemented system is designed to conform to the standard protocols of video streaming systems. RTSP/TCP sessions are used to control video streaming and RTP/UDP to deliver video data. We introduce a TFRC (TCP Friendly Rate Control) mechanism to the system for video streaming to share the bandwidth fairly with conventional TCP sessions. We use a frame dropping filter to adapt the rate of video streams to the bandwidth available to service. Through evaluations from several performance points of view, it is shown that our proxy caching system can dynamically adjust the quality of video streams to suit network conditions, while providing users with a continuous and high-quality service. Furthermore, it is shown that our proxy caching system can reduce the amount of traffic between a video server and a proxy in the limited cache buffer.

Then, being based on the system implemented in the above, we further design and implement a video streaming system with cooperative proxy cache servers for efficient and effective control. Our implemented system employs the standard protocols for communications among proxies. RTSP/TCP and RTP/UDP are used for video sessions among proxies. RTSP/TCP session are also used to share the information of cached blocks. We conduct several experiments on our implemented system to verify the practicality of our design and usefulness of our mechanisms. Through evaluations, it is shown that our cooperative proxy caching system can provide users with a continuous and high-quality video distribution in accordance with the network condition.

The rest of this paper is organized as follows. In section 2, we describe protocols, mechanisms and architecture used in MPEG-4 video streaming services. Then we introduce our proxy caching mechanisms with video quality adjustment capability [8] and cooperative proxy caching mechanisms [9, 10]. In section 3, we explain how the proxy caching mechanisms are implemented for an MPEG-4 video streaming service. We conduct several experiments to evaluate our system. In section 4, we verify our cooperative caching mechanisms through implementation and experiments. Finally, we conclude this thesis and describe future works in section 5.

2 Video Streaming Service and Proxy Caching Mechanisms

In this section, we first briefly explain video coding, architecture, and protocols of video streaming services. Then, we introduce our proxy caching mechanisms [8] and cooperative proxy caching mechanisms [9, 10].

2.1 Video Streaming Service

In this subsection, we describe an MPEG-4 video coding technique [14], and an MPEG-4 video streaming specification, ISMA 1.0 (Internet Streaming Media Alliance) [15], which is standard and widely used in the world. Then we introduce architecture and mechanisms of MPEG-4 video streaming service.

2.1.1 MPEG-4 Video Coding Technique

An MPEG-4 video stream (VOS: Visual Object Sequence) consists of one or more visual objects (VO). Each VO sequence consists of several layer streams (VOL: Video Object Layer). A layer video stream consists of three types of VOPs as shown in Fig. 1. VOP is the basic unit of image data and is equivalent to the frame or picture of MPEG-1 and MPEG-2. I-VOP is a self-contained intra-coded picture and it is coded using information only from itself. P-VOP is predictively coded using motion compensation algorithm, referring to the previously coded VOP. B-VOP is a bi-directionally predictive-coded VOP using the differences between both the previous and next VOPs. An I-VOP is directly (or

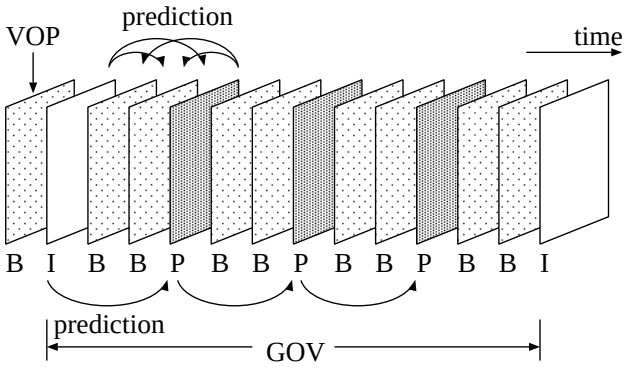

Figure 1: An example of MPEG-4 video structure

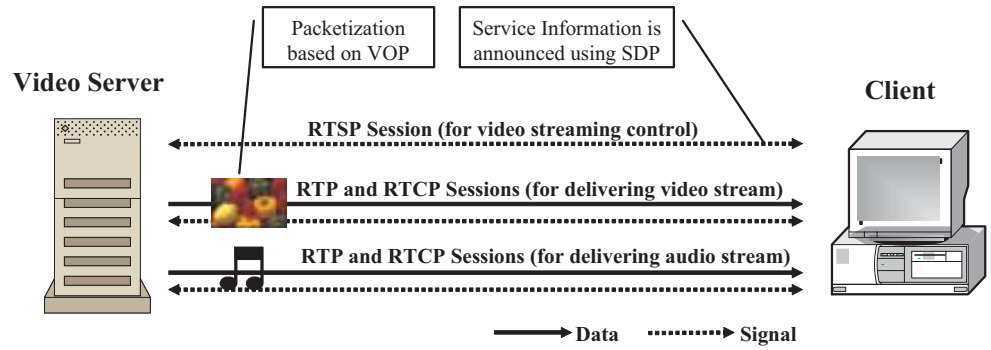

Figure 2: MPEG-4 video streaming service

indirectly) referred to by all following P-VOPs (and B-VOPs) until another frame is coded as an I-VOP. It is recommended to have I-VOPs regularly in a MPEG-4 video stream since the efficiency of motion compensation-based compression decreases as the distance from a referring frame to a referred picture becomes longer. In addition, by inserting I-VOPs as "refreshing points," we can achieve error resilience in the video transfer. Since an entire frame can be completely reconstructed from a successfully received I-VOP, error propagation can be interrupted when video quality is degraded in the preceding VOPs due to accidental packet losses.

A sequence of VOPs beginning from an I-VOP is called GOV (Group Of VOP) and defined by two parameters: the number of P-VOPs between two I-VOPs and the number of B-VOPs between two P-VOPs. In an example of Fig. 1, they are three and two, respectively. Using the GOV structure is highly recommended for regularity of video traffic, error resilience and accessibility to video contents. However, for achieving a high-quality video stream, many encoders select the VOP type, considering characteristics of a frame when it encodes a video stream. Thus, proxy caching mechanisms depend on GOV structure lacks domain of applicability. In this thesis, we do not have such assumptions for the structure of VOP.

2.1.2 MPEG-4 Video Streaming Specification

ISMA 1.0 [15] is a standard specification of MPEG-4 video streaming. In this specification, as illustrated in Fig. 2, RTSP [16] is used to control a video streaming service. A client can

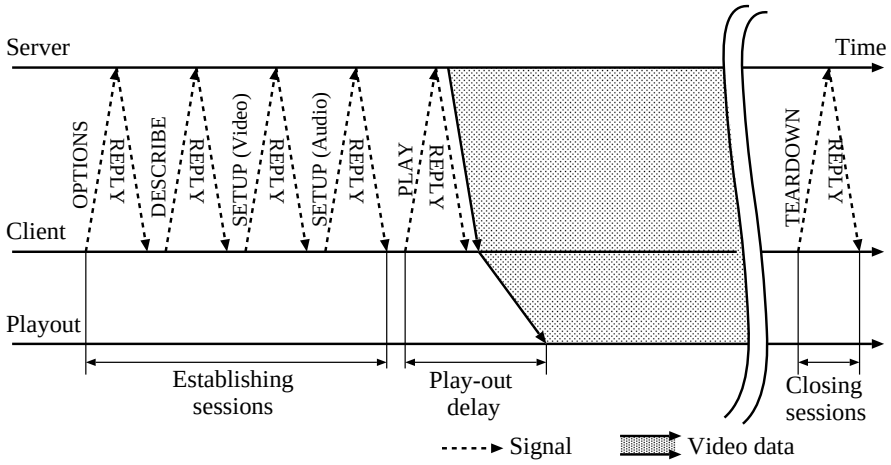

Figure 3: Basic behavior of MPEG-4 video streaming service

initiate a session, request a video stream, and wind forward, rewind, pause, and re-start the video play-out. An MPEG-4 video stream consists of a video stream and an audio stream, and they are separately delivered using RTP [17] and RTCP (RTP Control Protocol) sessions. Furthermore, for a video stream, packetization based on VOP is performed [18]. Information of video stream, for example bit rate, length and so on, are announced using SDP[19].

There are two standards described by ISMA 1.0: Profile 0 and Profile 1. Profile 0 is designed for narrowband and a mobile environment, such as PDAs or cell phones. The maximum bit rate of video stream is 64 kbps. Profile 1 is designed for broadband networks assuming high-performance client system such as video set-top boxes and personal computers. In Profile 1, the bit rate of video stream is limited to 1.5 Mbps. Profile 1 is a superset of Profile 0. In this thesis, we consider profile 1 for implementation of the system.

2.1.3 MPEG-4 Video Streaming Service

Figure 3 illustrates the basic behavior of MPEG-4 video streaming services, which we consider in the thesis. First, a client begins by establishing a video session with the video server using RTSP messages, such as OPTIONS, DESCRIBE, and SETUP. The SETUP message is sent twice for video/audio streams. An OPTIONS message is used to negotiate communication options available in the session. A DESCRIBE message is used for media

initialization. A client can obtain the information of a video stream from SDP descriptions of a DESCRIBE REPLY message. A SETUP message is used for transport parameter initialization. Information about port numbers, transport protocol and session identifier are exchanged and negotiated among a server and a client. For each of the coupled video and audio streams, a pair of RTP and RTCP sessions between a video server and a client is established by exchanging SETUP and SETUP REPLY messages. The order of RTSP messages is different among client applications. For example, RealOne Player sends messages in the order of OPTIONS, DESCRIBE and SETUP messages. On the contrary, QuickTime Player sends a DESCRIBE message first and then a SETUP message. It does not employ an OPTIONS message. In this thesis, the order of RealOne Player is used in figures and explanations. However, our system itself does not rely on any specific order of RTSP messages and can be applied to any kind of video streaming systems, as far as they conform to ISMA 1.0 specifications.

When a video session between a video server and a client is established, the client requests delivery of a video stream by sending an RTSP PLAY message. A PLAY message can have a Range field in its header with which a client can specify the desired range of video stream. A PLAY message without a Range header indicates that a client wants to receive a video stream from the beginning to the end.

On receiving a request for a video stream from a client, the video server begins providing the client with the video stream on two pairs of RTP and RTCP sessions. A client receives the video stream from a video server, and first deposits it in a so-called play-out buffer. Then, after some period of time, it gradually reads it out from the buffer and plays it. By deferring the play-out as illustrated in Fig.3, the client can prepare for unexpected delay in delivery of the video stream due to network congestion.

When a client leaves the service, it sends an RTSP TEARDOWN message to the video server and closes the sessions.

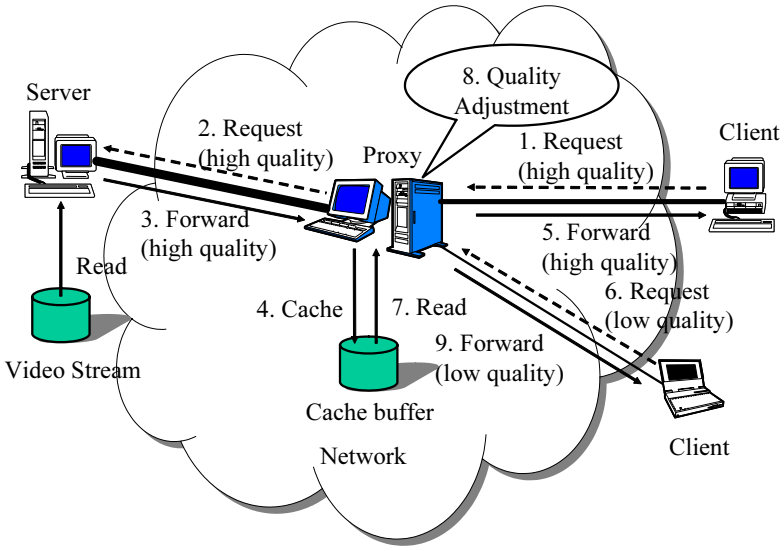

Figure 4: Video streaming system with proxy cache server

2.2 Proxy Caching Mechanisms with Video-Quality Adjustment Capability

In this subsection we introduce our proxy caching mechanisms [8]. In the proposed system illustrated in Fig. 4, a video stream is divided into N blocks for efficient use of a cache buffer and the bandwidth. Each block has the same length in time. A proxy cache server is assumed to be able to adjust the quality of cached or retrieved video blocks to requests through video filters or transcoders [20]. The system determines the quality of a block to retrieve from a video server, replaces cached blocks with a newly retrieved block, and prefetches blocks prior to requests.

In the following subsections, we first explain an overview of mechanisms. Then we introduce the details of mechanisms for the block retrieval, prefetching and replacement mechanisms. To simplify the discussion, we consider the case where clients concentrate on a single video stream. Our mechanism is, however, applicable to cases where the video-streaming system is providing multiple video streams to individual users.

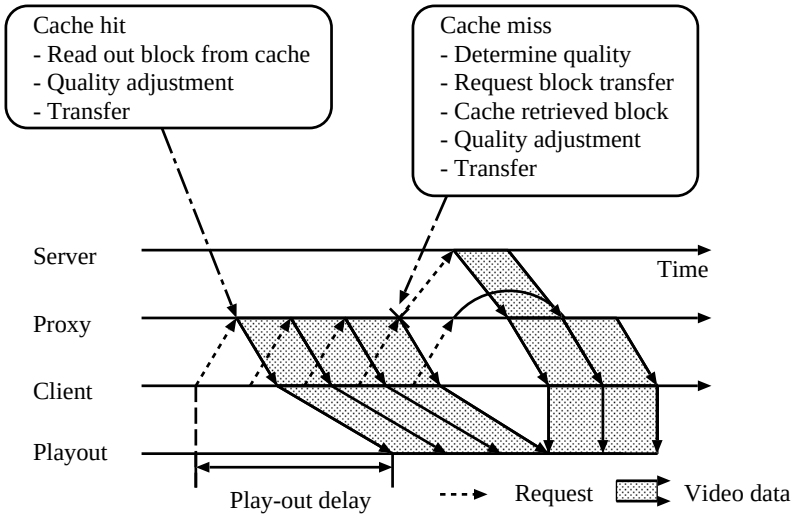

Figure 5: Basic behavior of our proxy caching system

Table 1: Structure of cache table

block	size	quality	quality of receiving block
1	a	A	B
2	b	C	0
3	0	0	D
⋮	⋮	⋮	⋮

2.2.1 Overview of Mechanisms

Figure 5 illustrates the basic behavior of our mechanisms. A client periodically requests a designated proxy to send a block. Each request expresses the desired level of quality of the block, which is determined based on the client-system performance, user's preferences on the perceived video quality, and the available bandwidth specified by an underlying protocol or the link capacity.

The proxy maintains a table, called a "cache table," for each of the video streams and possesses information on cached blocks. As shown in Table 1, each entry of a cache table includes the block number, the size and quality of the cached block and the quality of

blocks under transmission. The size and quality become zero, if the block is not cached or not being transmitted.

On receiving a request, a proxy compares it to a corresponding entry in the table. If the quality of a cached block can satisfy the request, i.e., a cache hit, the proxy reads out the cached block, adjusts the level of the quality to the request, and transmits it to the client. Video-quality adjustment is performed by QoS filtering techniques such as frame dropping, low-pass, and re-quantization filters [20]. In some cases, a block being received can satisfy the request even if the cache cannot satisfy. To avoid introducing extra delay in retrieving a block, the proxy waits for the completion of the reception and provides it to the client. Otherwise, when a cache misses the request, the proxy retrieves a block of the appropriately determined quality from the server on a session established to the server for the client. Then, the newly obtained block is stored in the cache. If there is not enough room, one or more cached blocks are replaced with the new block. Cached blocks are useful to reduce the response time, but further reduction can be expected if the proxy retrieves and prepares blocks in advance using the residual bandwidth.

2.2.2 Block Retrieval Mechanism

When a cache cannot supply a client with a block of the requested quality, a proxy should retrieve the block. The quality of a block that a proxy can retrieve from a server in time is determined in accordance with the available bandwidth between the server and the proxy. Thus, when the path between the server and the proxy is congested, the proxy cannot satisfy the client's demand even if it retrieves the block from the server. We introduce a parameter α which is given as:

$$\alpha = \frac{\max(Q_{sp}(i,j), Q_{cache}(j))}{Q_{pc}(i,j)}. \tag{1}$$

α is the ratio of the quality that the proxy can provide to the request. j $(1 \leq j \leq N)$ is the block number that client i requires. $Q_{sp}(i,j)$ stands for the quality of block j that can be transferred from the server to the proxy within a block time. The block time is given by dividing the number of frames in a block by the frame rate. For example, if the block corresponds to 30 frames and it is played out at 30 frames per second, one block-time is equal to one second. By multiplying the available bandwidth by the block-time, the proxy

obtains the size feasible for block j. Then, assuming that the quality can be determined from the size [21], $Q_{sp}(i,j)$ is derived. The quality affordable on the path between the proxy and the client is expressed as $Q_{pc}(i,j)$ and is regarded as the client i's request on block j. The quality of a cached block j, $Q_{cache}(j)$, is obtained from a corresponding entry of the cache table. In this subsection, since we consider a case of the cache miss, $Q_{cache}(j) < Q_{pc}(i,j)$ holds.

When $\alpha \geq 1$, that is, the proxy, can provide the client with the block of the desired quality, we have three alternatives of determining the quality of block j to retrieve for the client i, $Q_{req}(i,j)$.

method1: A possible greedy way is to request the server to send block j of as high quality as possible. This strategy seems reasonable because cached blocks can satisfy most of the future requests and the probability of cache misses becomes small. Then, the request $Q_{req}(i,j)$ becomes

$$Q_{req}(i,j) = Q_{sp}(i,j). \tag{2}$$

method2: When the available bandwidth between the server and the proxy is extremely larger than that between the proxy and the client, method1 cannot accomplish the effective use of bandwidth and cache buffer. Thus, we propose an alternative which determines the quality $Q_{req}(i,j)$ based on a prediction of demands on block j, as follows:

$$Q_{req}(i,j) = \min(\max_{k \in S, 0 \leq l \leq j} Q_{pc}(k,l), Q_{sp}(i,j)), \tag{3}$$

where S is a set of clients which are going to require block j in the future. The client i is also in S.

method3: To accomplish a further efficient use of the cache, it is possible to request block j of the same quality that the client requests, as follows:

$$Q_{req}(i,j) = Q_{pc}(i,j). \tag{4}$$

With this strategy, the number of cached blocks increases and the probability of cache misses is expected to be suppressed as far as future requests can be satisfied with them.

In some cases, both the cached and retrieved blocks cannot meet the demand ($\alpha < 1$). One way is to request a server to send a block of the desired quality, but it may cause an

undesirable delay. The other is for a client to be tolerant of the quality degradation and accept a block of lower quality than was requested. We introduce another parameter β to tackle this problem. β is defined as the ratio of the acceptable quality to the demand, and it expresses the client's insistence on the video quality. Clients with β close to one want to receive blocks in accordance with the request at the risk of undesirable transfer delay. On the other hand, those who value timeliness and interactivity of applications will choose β close to zero.

First, we consider the case that the quality of a cached block can satisfy the client, but is still lower than the request ($\beta \leq \frac{Q_{cache}(j)}{Q_{pc}(i,j)} \leq \alpha < 1$). In such a case, in order to effectively reuse the cached block, the proxy only sends the cached block to the client regardless of the quality of the block that the server can provide, $Q_{sp}(i,j)$. When the quality of the cached block is not high enough ($\frac{Q_{cache}(j)}{Q_{pc}(i,j)} < \beta \leq \frac{Q_{sp}(i,j)}{Q_{pc}(i,j)} = \alpha < 1$), the proxy requests the server to send block j whose quality is equal to $Q_{sp}(i,j)$ as:

$$Q_{req}(i,j) = Q_{sp}(i,j). \tag{5}$$

Finally, if the proxy cannot provide the client with a block of the satisfactory quality ($\alpha < \beta$), it requests the server to send the block of the minimum quality which is expected not to cause a cache miss, that is,

$$Q_{req}(i,j) = \beta \cdot Q_{pc}(i,j). \tag{6}$$

The proxy requests the server to send block j of the quality $Q_{req}(i,j)$. The corresponding entry for client i, $Q_{rec}(i,j)$ for the quality of block j being received in the cache table is set to $Q_{req}(i,j)$. When the block reception is finished, $Q_{cache}(j)$ is set to $Q_{rec}(i,j)$.

2.2.3 Block Prefetching Mechanism

To reduce the possibility of cache misses and avoid the delay in obtaining missing blocks from a server, a proxy prefetches blocks that clients are going to require in the future. After checking the cache table for block j being requested by client i, a proxy compares the minimum requirement $\beta \cdot Q_{pc}(i,j)$ to the quality of cached blocks $Q_{cache}(k)$ and that of receiving blocks $Q_{rec}(i,k)$ (for $\forall i,\ j+1 \leq k \leq j+P \leq N$). Here, P is the size of a sliding window called a prefetching window, which determines the range of examination

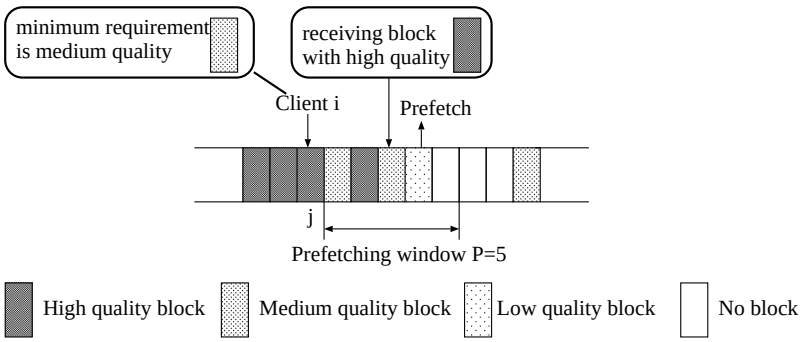

Figure 6: An example of prefetching

for prefetching. If there exists any block whose quality is lower than the minimum, a block retrieval mechanism is triggered. The mechanism is the same as one explained in subsection 2.2.2 except that the available bandwidth to prefetching is the remainder of bandwidth between the server and the proxy. Prefetch requests have a lower priority than requests for retrieving cache-missed blocks at the server in order not to disturb urgent block-retrieval.

Figure 6 illustrates an example of prefetching. Client i requests block j, and its minimum requirement is "medium" quality. Block $j+3$ is receiving from a server with "high" quality. In this case, the proxy prefetches block $j+4$.

2.2.4 Cache Replacement Mechanism

Since a cache has a limited capacity, the replacement of cached blocks should be considered to accomplish the effective use of a storage. When the quality of a newly retrieved block is lower than that of a cached block, the new block is not to be cached. Otherwise, a proxy first removes a cached block of the lower quality if exists. Then, it tries to deposit the new block in the cache. If there is not sufficient room to store the new block, some cached blocks must be removed. We propose a replacement algorithm with consideration of size, quality and reusability of blocks.

First, the proxy assigns priority to cached blocks. Blocks requested by clients at the moment of the replacement have the highest priority and are never removed from the cache. The block residing at the beginning of the stream is also assigned the highest priority to

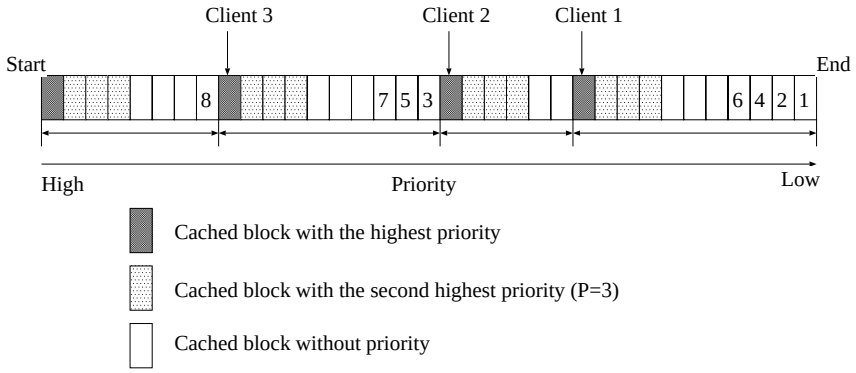

Figure 7: Priorities of cached blocks against replacement

provide potential clients with a low-latency service. The second most important blocks are those in the prefetching windows following the most important blocks. The other blocks have no priority.

Then, block candidates for replacement are chosen one by one until the sufficient capacity becomes available. In Fig. 7, we show an example of victim selection. A cached block, which is located at the end of the longest succession of unprioritized blocks, is regarded as the least important and becomes the first victim, as indicated by "1" in the figure. Among successions of the same length, one closer to the end of the stream has a lower priority.

The proxy first tries the quality adjustment to decrease the size of the victim if it is larger than the new block. Since it is meaningless to hold a block whose quality is smaller than $Q_{req}(i, j)$ determined by the method chosen in the block retrieval mechanism, no further adjustment is performed and the victim is removed from the cache. The proxy repeatedly chooses the next victim and applies the same techniques until the capacity for the new block becomes sufficient. When all unprioritized blocks are removed but it is still insufficient, the proxy gives up storing the new block.

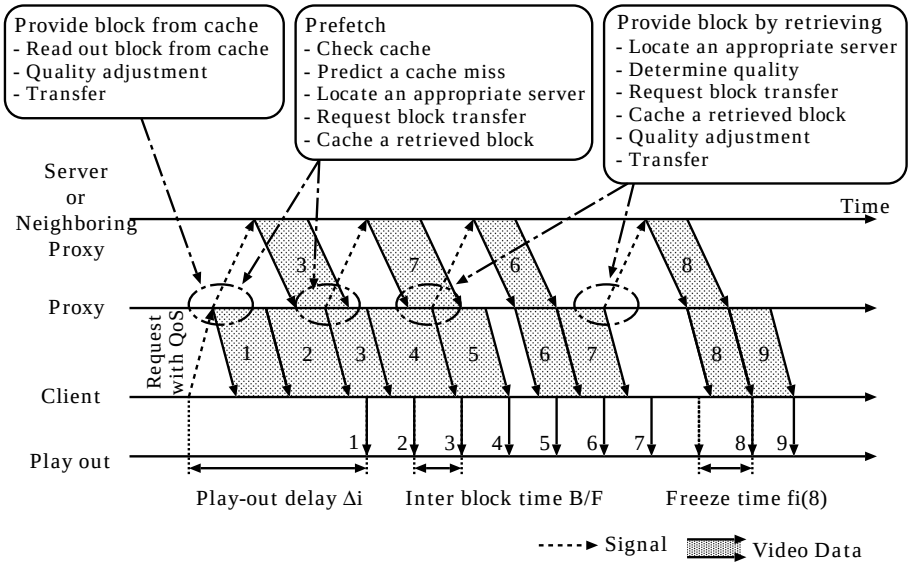

Figure 8: Basic behavior of our cooperative proxy caching system

2.3 Cooperative Proxy Caching Mechanisms

In this subsection we introduce our cooperative proxy caching mechanisms [9, 10]. The video streaming system with cooperative proxy cache servers is constructed based on the system described in subsection 2.2. It consists of an originating video server, cooperative proxies, and heterogeneous clients. A proxy cache server retrieves video blocks from a distant origin server or neighboring proxies on behalf of clients, deposits them in its local cache buffer, and adapts the quality of blocks to the user's demands. In addition, proxy cache servers communicate with each other. Servers exchange blocks over a video session established among them. We should note here that to reduce loads on networks and servers, there is only one session between any pair of servers regardless of the number of clients.

In the following subsections, we describe several mechanisms for the video streaming system where proxies cooperate to provide users with low-latency and high-quality services under a heterogeneous environment.

2.3.1 Overview of Mechanisms

Figure 8 illustrates how servers and client communicate with each other in our video streaming system. The video streaming service is initiated by a request message issued by a client to a designated proxy cache server. The message contains information about preferable (upper-constraint) and tolerable (lower-constraint) levels of video quality which are determined in accordance with the available bandwidth, end-system performance and user's preferences. The client is allowed to dynamically change QoS requirements during a video session to reflect changes of those constraints.

On receiving the first request message, the proxy cache server begins to provide a requested video stream to the client. The proxy adopts the fastest way that can provide the client with a block of higher level of quality. The proxy can (i) read out and send a cached block, (ii) use a block being received, (iii) wait for the preceding request for the same block to be served, or (iv) newly retrieve a block from the other server. If the proxy has a block of the same number and its quality can satisfy the request, the proxy may consider that using the cached block is the best way. It applies the video quality adjustment to the block as necessary and transfers the block to the client over a video session established between them. If the block being received has satisfactory quality, the proxy may decide to send the block under transmission to the client. If there is a preceding request for the same block of higher quality, the proxy can wait for the request to be served. In some cases the proxy determines to retrieve the block. It first identifies an appropriate server among candidates, then determines the quality of the block to retrieve, and sends a request message to the server. A retrieved block is cached and sent to the client after the video quality adjustment is applied as necessary. When the proxy cache server finishes sending out the block to the requesting client, it moves to the next block and repeats similar procedures. While providing clients with video blocks, the proxy cache server predicts and prepares for a future potential cache miss by prefetching a block from the other server. A prefetching request is processed without disturbing normal block retrievals. Details of the prefetching mechanisms will be given in subsection 2.3.4.

The cooperative proxy caching mechanisms consists of a block provisioning mechanism, block prefetching mechanism and cache replacement mechanism. These mechanisms are

based on proxy caching mechanisms, which are a block retrieval mechanism, block prefetching mechanism and cache replacement mechanism, described in subsection 2.2.2, 2.2.3, and 2.2.4, respectively. In addition, we introduce a remote table to maintain information of cached blocks at other servers. Detail of these mechanisms will be given in the following subsections.

To avoid or absorb unexpected block transmission delays, a client i first defers playing out a received video block for a predetermined waiting time, denoted as Δ_i in the thesis as shown in Fig. 8. Then, it begins to decode and display received blocks one after another, at regular intervals of B/F where B corresponds to the number of frames in a block and F stands for the frame rate. Once a user begins to watch a video, the video streaming system must provide the client with video blocks in a continuous and timely manner. However, in some cases, a block does not arrive in time and the client is forced to pause. Time required for the client i to wait for the arrival of necessary block j is called *freeze time*, $f_i(j)$ in the thesis. If the block j arrives in time, freeze time $f_i(j)$ becomes zero. A cache miss is one of the reasons that continuous and in-time delivery of video block is made difficult. Even if all blocks of a high level of quality are cached, a proxy cache server cannot guarantee the in-time delivery due to unexpected and uncontrollable network congestion.

2.3.2 Cache Table and Remote Table

In providing a client with a block, a proxy cache server can send a cached block, apply a receiving block, make use of the preceding request and retrieve a block. Among them, a proxy chooses the best way in terms of the offerable quality and the block transfer time. For this purpose, servers communicate with each other and maintain the up-to-date information, such as the quality of offerable blocks, round-trip time, and available bandwidth. To predict the transfer time as accurately as possible, the proxy is assumed to be able to estimate the block size, propagation delay and throughput. Information related to the network condition among the proxy and clients is also required.

The proxy server k has two tables. Information on the locally cached blocks is maintained in one, referred to as the cache table, while the other, the remote table, is for information the caching of blocks at other servers. The cache tables maintain information on the cached and non-cached blocks of a video stream. They are slightly different from

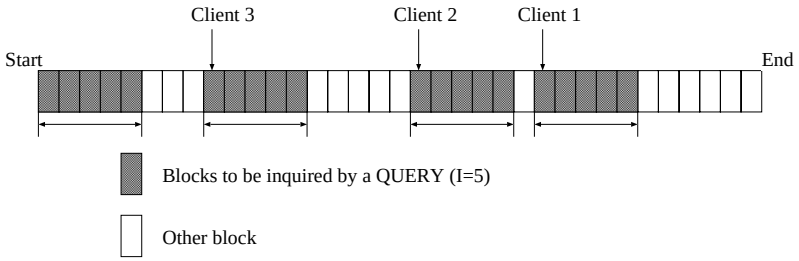

Figure 9: Inquiry blocks by sending QUERY

the cache table in subsection 2.2. Each entry in the cache table corresponds to one block of the stream and consists of a block number j, the quality of the block $q_k(j)$ in the cache, and a list of markers $M_k(j)$ for the cached block j. Larger values of $q_k(j)$ indicate higher levels of video quality, while a zero means that the block j is not cached in the local buffer. After a block has been received, the corresponding $q_k(j)$ is replaced by the quality of the newly retrieved block. The list of markers $M_k(j)$ is a set of identifiers of proxy servers. The remote table in a given proxy holds information on the other servers. Each entry in the remote table corresponds to the name of a server (e.g., server s, $s = 0$ indicates the originating server). It includes the estimated one-way propagation delay $d^{SP}_{s,k}$ and the estimated available bandwidth $r^{SP}_{s,k}$ for the connection, along with the list of blocks available on that server and their respective quality $q^r_s(j)$. The table is built and maintained by exchanging *QUERY* and *REPLY* messages.

When a proxy receives a first request from a new client, it sends a *QUERY* message to a multicast address which is shared by all of the servers. The proxy sets a TTL for the *QUERY* message and thus limits the scope of neighboring servers that will respond to the query as in SOCCER [6]. The *QUERY* message informs the other servers of the video stream which the proxy is requesting and a list of the blocks it expects to require in the near future. On receiving the query, the proxy server s searches its local cache for the blocks which are listed in the *QUERY* message, and sets marks $M_s(j)$ for the blocks in its cache table. After a block has been marked, it leaves the scope of block replacement. Those blocks not being declared in the message are unmarked for the proxy. The server s then returns a *REPLY* message which carries a timestamp and the list of $q^r_s(j)$ values for those blocks it holds which were in the list included with the query. As illustrated in

Fig. 9, the blocks indicated in the inquiries about quality are those which are currently requested by clients and subsequent I blocks. The window of inquiry I is defined as a way of restricting the range of blocks to include in a single inquiry and preventing all blocks from being locked. I blocks from the beginning of the stream are also listed in the $QUERY$ message to prepare for new clients that will request the stream in the future [7].

The interval at which a proxy sends $QUERY$ messages must be neither too short nor too long. The proxy issues a $QUERY$ message when the end of the prefetching window, which is explained in subsection 2.3.4, of any client finds an entry $q_s^r(i)$ which is zero. In addition, we also introduce a timer to force a refreshing of the remote tables. Timing reaches expiry every $(I - P - 1)B/F$ where P is the size of the prefetching window.

2.3.3 Block Provisioning Mechanism

A proxy adopts the fastest way that can provide the client with a block of higher level of quality. Assume that the proxy k is trying to provide the client i with the block j at time t. The quality of the block j must be higher than $q_i(j)$ and as high as $Q_i(j)$, which are determined by the QoS requirements specified by the latest request message. A deadline $T_i(j)$ that the client i should finish receiving the block j is determined as

$$T_i(j) = T_i + \Delta_i + (j-1)\frac{B}{F} - \delta_i + D_i(j-1), \tag{7}$$

where T_i indicates an instant when the client i issues the first request message, Δ_i is the initial waiting time, and B and F stand for the number of frames in a block and the frame rate, respectively. δ_i is introduced to absorb unexpected delay jitters and estimation errors. Even if we carefully design control mechanisms and a streaming system is successfully built, we cannot avoid freezes. After a freeze, the client decodes and displays the succeeding blocks at regular intervals as shown in Fig. 8. $D_i(j-1)$ is accumulated freeze time and is given as $D_i(j-1) = \sum_{l=1}^{j-1} f_i(l)$. We consider four kinds of offerable quality to provide the client with a low-delay and high-quality block distribution.

case1: The offerable quality of the block j to the client i with the block j cached at the proxy k's buffer, $c_{k,i}^{PC}(j)$, is derived as

$$c_{k,i}^{PC}(j) = \min(q_k(j), \overline{c}_{k,i}^{PC}(j)), \tag{8}$$

where

$$\overline{c}_{k,i}^{PC}(j) = \max(q|t + d_{k,i}^{PC}(t) + \frac{a_j(q)}{r_{k,i}^{PC}(t)} \leq T_i(j)). \tag{9}$$

$q_k(j)$ stands for quality of the block j cached at the proxy k's buffer. $a_j(q)$ is a function that gives the size of block j of quality q. If $j = 0$ or $q = 0$, $a_j(q)$ gives zero. Thus, $c_{k,i}^{PC}(j)$ becomes zero if a corresponding block is not cached at t. $d_{k,i}^{PC}(t)$ and $r_{k,i}^{PC}(t)$ are estimates of available bandwidth and one-way propagation delay between the proxy k and the client i at t. Thus, $d_{k,i}^{PC}(t) + \frac{a_j(q)}{r_{k,i}^{PC}(t)}$ provides estimated time required for the client i to receive a whole of the block j of quality q via a video session whose available bandwidth is $r_{k,i}^{PC}(t)$ and propagation delay is $d_{k,i}^{PC}(t)$.

case2: If the block j is being received, the proxy can use it. The quality offerable is derived as

$$b_{k,i}^{PC}(j) = \max_{\forall s, s \neq k}(\min(q_{s,k}^{SP}, \overline{b}_{k,i}^{PC}(j))), \tag{10}$$

where

$$\overline{b}_{k,i}^{PC}(j) = \max(q|t + d_{k,i}^{PC}(t) + \max(\frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}, \frac{a_j(q)}{r_{k,i}^{PC}(t)}) \leq Ti(j)). \tag{11}$$

In the above equations, s indicates the server from which the proxy is receiving the block j. $q_{s,k}^{SP}(t)$ is the quality of block being received at t. $a_{s,k}^{SP}(t)$ stands for the amount that has been received. $d_{s,k}^{SP}(t)$ and $r_{s,k}^{SP}(t)$ correspond to estimates of available bandwidth and one-way propagation delay between the server s and the proxy k at t.

case3: A proxy keeps track of requests sent out for block retrieval. $\mathcal{W}_{s,k}^{SP}(t) = \{W_1, W_2, \ldots, W_w\}$ is a list of requests sent from the proxy k to the server s before t. The quality of the block W_n, $q_{s,k}^{w}(n)$, is also maintained in a list. A pair of the block number and the quality is added to the lists when the proxy sends a request for block retrieval and is removed from the lists when the proxy begins receiving the block. When the mth request is for the block j, that is, $W_m = j$, the offerable quality $w_{k,i}^{PC}(j)$ becomes

$$w_{k,i}^{PC}(j) = \max_{\forall s, s \neq k}(\min(q_{s,k}^{w}(m), \overline{w}_{k,i}^{PC}(j))), \tag{12}$$

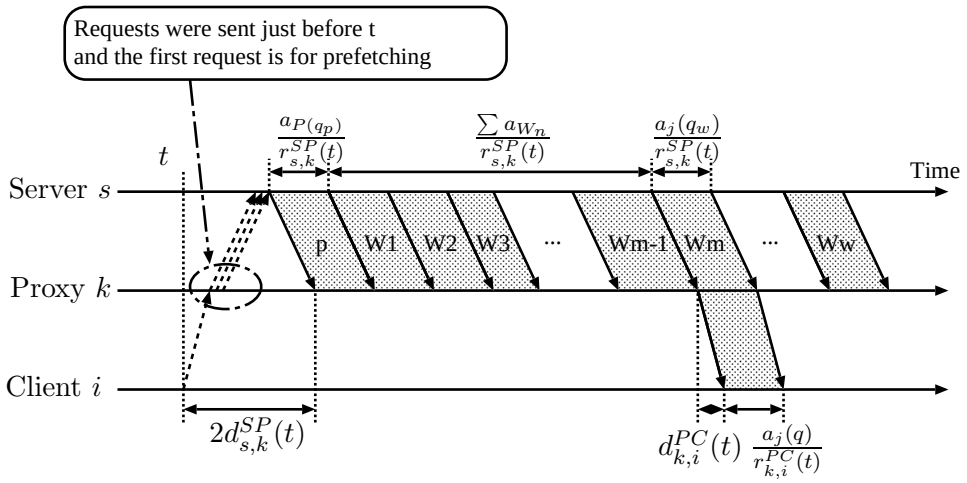

Figure 10: Worst-case delay introduced in waiting for the preceding request

where

$$\overline{w}^{PC}_{k,i} = \max(q|t + 2d^{SP}_{s,k}(t) + d^{PC}_{k,i}(t) + \frac{\sum_{n=1}^{m-1} a_{W_n}(q^w_{s,k}(n)) + a_{p_{s,k}(t)}(q^p_{s,k}(t))}{r^{SP}_{s,k}(t)} + \max(\frac{a_j(q^w_{s,k}(m))}{r^{SP}_{s,k}(t)}) \leq T_i(j)). \quad (13)$$

$p_{s,k}(t)$ indicates the block number to be prefetched from the server s and $q^p_{s,k}(t)$ is the quality requested for the block $p_{s,k}(t)$. If no prefetching request is waiting, both $p_{s,k}(t)$ and $q^p_{s,k}(t)$ are zero. In our system, only one prefetching request is permitted by a server per proxy and prefetching is preempted by normal block retrievals. Details of a prefetching mechanism will be given in subsection 2.3.4. Equation 13 considers the worst case when no block is under transmission and the preceding requests from 1 to $m-1$ and the prefetching request will be served prior to the block W_m, as illustrated in Fig. 10. If the proxy is receiving a block at t, $\overline{w}^{PC}_{k,i}$ is derived using the following equations.

$$\overline{w}^{PC}_{k,i} = \max(q|t + \max(2d^{SP}_{s,k}(t), \frac{a_j(q^{SP}_{s,k}(t)) - a^{SP}_{s,k}(t)}{r^{SP}_{s,k}(t)}) + d^{PC}_{k,i}(t) + \frac{S^{SP}_{s,k}(t)}{r^{SP}_{s,k}(t)} + \max(\frac{a_j(q^w_{s,k}(m))}{r^{SP}_{s,k}(t)}, \frac{a_j(q)}{r^{PC}_{k,i}(t)}) \leq T_i(j)), \quad (14)$$

where

$$S_{s,k}^{SP}(t) = \begin{cases} \sum_{n=1}^{m-1} aw_n(q_{s,k}^w(n)) + a_{p_{s,k}(t)}(q_{s,k}^p(t)), & \text{if } 2d_{s,k}^{SP}(t) > \dfrac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} \\ \sum_{n=1}^{m-1} aw_n(q_{s,k}^w(n)), & \text{otherwise} \end{cases}. \quad (15)$$

case4: Finally, the offerable quality $s_{k,i}^{PC}(j)$ by retrieving the block j from the server s is derived by the following equations when there is no block under transmission,

$$s_{k,i}^{PC} = \max_{\forall s, s \neq k}(\min(q_s^r(j), \overline{s}_{k,i}^{PC}(j))), \quad (16)$$

where

$$\begin{aligned} \overline{s}_{k,i}^{PC} &= \max(q | t + 2d_{s,k}^{SP}(t) + d_{k,i}^{PC}(t) + \frac{\sum_{n=1}^{w} aw_n(q_{s,k}^w(n)) + a_{p_{s,k}(t)}(q_{s,k}^p(t))}{r_{s,k}^{SP}(t)} \\ &\quad + \frac{a_j(q)}{\min(r_{s,k}^{SP}(t), r_{k,i}^{PC}(t))} \leq T_i(j)). \end{aligned} \quad (17)$$

Here, $q_s^r(j)$ corresponds to the quality of the block j cached at the server s. On the contrary, if the proxy k is receiving a block from the server s,

$$\begin{aligned} \overline{s}_{k,i}^{PC} &= \max(q | t + \max(2d_{s,k}^{SP}(t), \frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}) + d_{k,i}^{PC}(t) + \frac{U_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} \\ &\quad + \frac{a_j(q)}{\min(r_{s,k}^{SP}(t), r_{k,i}^{PC}(t))} \leq T_i(j)), \end{aligned} \quad (18)$$

where

$$U_{s,k}^{SP}(t) = \begin{cases} \sum_{n=1}^{w} aw_n(q_{s,k}^w(n)) + a_{p_{s,k}(t)}(q_{s,k}^p(t)), & \text{if } 2d_{s,k}^{SP}(t) > \dfrac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} \\ \sum_{n=1}^{w} aw_n(q_{s,k}^w(n)), & \text{otherwise} \end{cases}. \quad (19)$$

The fastest and finest way is chosen as far as the offerable quality is higher than the minimum request $q_i(j)$ and is lower than the maximum request $Q_i(j)$. If none of $c_{k,i}^{PC}(j)$, $b_{k,i}^{PC}(j)$, $w_{k,i}^{PC}(j)$, and $s_{k,i}^{PC}(j)$ cannot satisfy the request $q_i(j)$, the proxy chooses the fastest way.

2.3.4 Block Prefetching Mechanism

In order to achieve further effective control, a proxy prefetches blocks in accordance with client requests. While supplying the client i with the block j, the proxy investigates the

cache table for blocks $j+1$ to $j+P$ to find a potential cache miss. The parameter P determines the range of the prefetching window. When the proxy finds the block with quality is lower than $q_i(j)$ and there is no request waiting to be served, it attempts to prefetch the block of finer quality. If there are two or more unsatisfactory blocks, the one closest to the block j is chosen.

The prefetch requests are treated at the server in a different way from those for retrieving cache-missed blocks. The origin server and proxy servers maintain pairs of prioritized queues per video session. The queue given a higher priority is for usual block retrieval, and requests are handled in a first-come-first-served discipline. The other queue for prefetching has only one room and the request waiting in the queue is overwritten with a new one. The prefetch request in the queue is served only when there is no request in the high-priority queue. The prefetch request is considered obsolete and canceled when the server receives the request for the same block with the higher quality requirement.

The proxy decides to prefetch a block if reception of the block is expected to be completed in time. The expected time $t^p_{s,k}(t)$ when the proxy finishes prefetching is derived as

$$t^p_{s,k}(t) = t + 2d^{SP}_{s,k}(t) + \frac{a_{p_{s,k}(t)}(q^p_{s,k}(t))}{r^{SP}_{s,k}}, \tag{20}$$

where $p_{s,k}(t)$ and $q^p_{s,k}(t)$ stand for the block number and the requested quality, respectively. This equation means that the proxy tries prefetching only when no preceding request is waiting to be served. If the derived time $t^p_{s,k}(t)$ is faster than the time $T^p_i(p_{s,k}(t))$ which is derived as,

$$T^p_i(p_{s,k}(t)) = T_i + \Delta_i + (p_{s,k}(t)-1)\frac{B}{F} - \delta_i + D_i(j-1) - d^{PC}_{k,i}(t), \tag{21}$$

the proxy sends a request to the server s.

Information about a prefetching request is kept at a proxy as a pair of the block number $p_{s,k}(t)$ and the quality $q^p_{s,k}(t)$ and is overwritten by a new prefetching and canceled when a corresponding block reception begins or a normal block retrieval is requested for the same block of higher quality. The quality $q^p_{s,k}(t)$ is determined on the basis of the QoS requirement as $\beta Q_i(j)$ where $0 < \beta \leq 1$. If we set β to a small value, we can expect to prepare enough number of blocks but the quality of blocks provided to clients becomes

low. On the other hand, with a large β, there is little chance to successfully prefetch blocks but a high-quality video stream can be provided with prefetched blocks.

2.3.5 Cache Replacement Mechanism

The proxy makes a list of blocks to be discarded on the basis of its cache table. Those blocks which are located in windows of inquiry and marked by the other proxy servers are placed beyond the scope of replacement. Reference to the blocks about which the inquiry was made is expected in the near future, i.e., in IB/F seconds. The marked blocks should not be dropped or degraded in order to maintain consistency of the remote tables that the other proxies hold. In addition, retention of the first I blocks is also considered important as a way of hiding the initial delay [7]. The rest of the blocks are all candidates for replacement. Of these, the block which is closest to the end of the stream is the first to be a victim, i.e., to be assigned a reduced level of quality or discarded. The proxy first tries lowering the quality of the victim as a way of shrinking the block size and making a room for the new block. The quality adjustment is limited to the maximum among the QoS requests issued by the clients watching blocks behind the victim. If the adjustment is not sufficiently effective, the victim is discarded from the cache buffer. If this is still insufficient, the proxy moves on to the next victim.

When all of the candidate victims have been dropped and there is still not enough room, the proxy identifies the least important of the protected blocks. The last P blocks of the successive marked blocks are only for use in prefetching and may thus be considered less important since the prefetching is given a lower priority by the servers, as was explained in subsection 2.3.4. The proxy starts the process of trying the degradation of quality and dropping blocks at the block closest to the end of a video stream. Finally, if all of the attempts described above fail, the proxy gives up on storing the new block.

3 Design, Implementation, and Evaluation of Proxy Caching Mechanisms with Video-Quality Adjustment Capability

We introduced our proxy caching mechanisms with video quality adjustment capability in subsection 2.2. In this section, we first design the MPEG-4 video streaming system that adopts our proposed mechanisms. Since we use off-the-shelf and prevailing products, there are problems in implementing our mechanisms that consider some ideal conditions and environments. We conduct several experiments on our implemented system to verify the practicality of our design and usefulness of our mechanisms.

3.1 Design and Implementation of System

In this subsection, we describe the system we implemented. Figure 11 illustrates the modules that constitute our video streaming system. The system consists of a video server, a proxy cache server and several clients. Each arrow, dashed arrow, and box corresponds to data flow, signal, and module, respectively. The modules constituting proxy are generated for each client. We employ the Darwin Streaming Server as the server application, and RealOne Player and QuickTime Player as the client applications. The video stream is coded using the MPEG-4 video coding algorithm. As explained in subsection 2.1, the video streaming is controlled through RTSP/TCP sessions, the video and audio are each transferred over a dedicated RTP/UDP session, and the quality of service is monitored over RTCP/UDP sessions. There are two sets of sessions for the client. The first is established between the originating video server and proxy to retrieve uncached blocks. The other is between the proxy and client.

Our mechanisms described in subsection 2.2 considers some ideal conditions and environments. Since we use off-the-shelf and prevailing products, there are problems in implementing our mechanisms. In our implemented system, a client application does not perform the block-by-block request which is described in subsection 2.1. Instead, it sends an RTSP PLAY message only once at the beginning of the session specifying Range. Furthermore, a server application cannot adjust the quality of a video stream, and cannot tell whether a request is for prefetching or for usual retrieval. Thus, we embed a request for prefetching in a preceding retrieval request using a Range field. In the following sub-

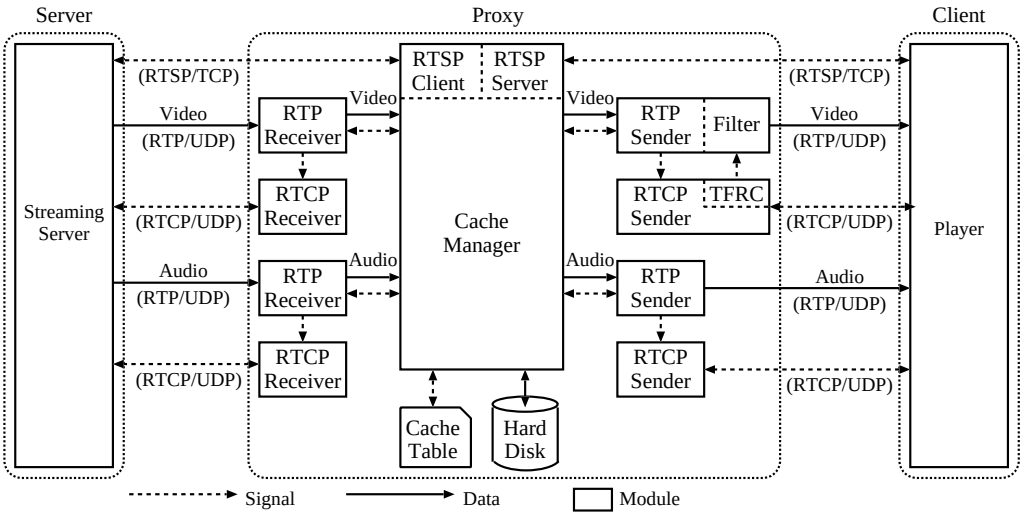

Figure 11: Modules constituting system of our proxy caching system

sections, we explain how the system is designed to implement the mechanisms for the MPEG-4 video streaming service.

3.1.1 Overview of Implementation

Figure 12 illustrates the basic behavior of our system. In our implemented system, each block corresponds to a sequence of VOPs of an MPEG-4 stream. A block consists of a video block and an audio block, and they are separately stored. The number of VOPs in a block is determined by taking into account the overheads introduced in maintaining the cache and transferring the data block-by-block. The strategy to determine the block size is beyond the scope of this thesis. However, we empirically used 300 in our empiric implementation. Since the MPEG-4 video stream we used in the experiments is coded at 30 frames per second, a block corresponds to 10 seconds of video. Segmentation based on VOP was reasonable since packetization is also performed on a VOP basis, as described in subsection 2.1. Furthermore, we could use the Range field of the RTSP PLAY message to specify a block, e.g., `Range 20-30`, because we could easily specify the time that the block corresponded to.

In our system, since the client applications do not request the video stream on a block-by-block basis, the proxy translates a request for a whole video stream issued at the beginning of the session into requests for blocks and examines the table for each block.

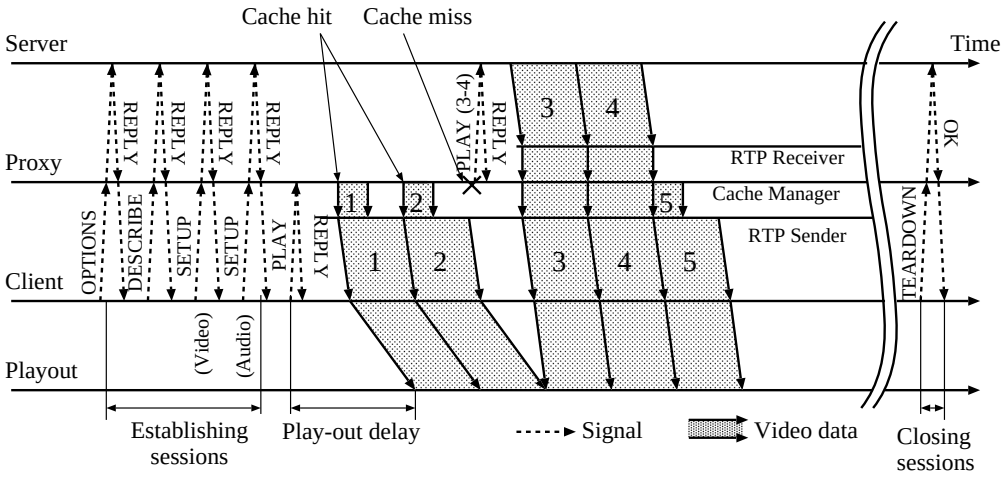

Figure 12: Basic behavior of our proxy caching system

Since the client applications we used in our implementation do not have any functionality to inform the server of desired quality of a video stream to receive and perceive, the proxy considers that the available bandwidth it estimates determines the quality desirable to the client. The server application also does not have a capability to adjust the quality of a video stream to demands, and the proxy cache server only retrieves a block of the highest or original quality and deposits it in a cache buffer. Therefore, if a block is cached, it always satisfies the request.

First, a client begins by establishing connections for video and audio streams with the proxy server by sending a series of RTSP OPTIONS, DESCRIBE, and SETUP messages. These RTSP messages are received by the *Cache Manager* through an *RTSP Server* module (in Fig. 11). The proxy server relays those messages to the video server. Then, connections between the video server and the proxy server are also established at this stage. On receiving SETUP REPLY messages, the client requests delivery of the video stream by sending an RTSP PLAY message.

A proxy maintains information about cached blocks in the *Cache Table*. Each entry in the table contains a block number, the size of the cached block, and the flag. The size is set at zero when the block is not cached. The flag is used to indicate that the block is being transmitted. On receiving a request for a video stream from a client through the *RTSP Server*, the *Cache Manager* begins providing the client with blocks. *Cache*

Manager divides the request on a block basis and examines the table for each block. If the requested block is cached, the *Cache Manager* reads it out and passes it to the *RTP Sender*. The *RTP Sender* packetizes the block and sends the packet to the client. The quality of video blocks is adjusted to fit the bandwidth on the path to the client by the *Filter*. The bandwidth is estimated by the *TFRC* (TCP Friendly Rate Control) module using feedback information collected by exchanging RTCP messages. When a connection between the video server and the proxy server is not used for the predetermined timeout duration, the video server terminates the connection according to RTSP specification. In our system, the proxy server maintains the connection for future use by sending an RTSP OPTIONS message after a 90-second idle period.

When a block is not cached in the local cache buffer, the *Cache Manager* retrieves the missing block by sending an RTSP PLAY message to the video server. Details of block retrieval will be given in subsection 3.1.4. On receiving a block from the video server through the *RTP Receiver*, the *Cache Manager* sets its flag to on to indicate that the block is being transmitted, and it relays the block to the *RTP Sender* VOP by VOP. When reception is completed, the flag is cancelled and the *Cache Manager* deposits the block in its local cache buffer. However if the retrieved block is damaged by packet loss, the *Cache Manager* does not deposit it. If there is not enough room to store the newly retrieved block, the *Cache Manager* replaces one or more less important blocks in the cache buffer with the new block. Details of cache replacement will be also given in subsection 3.1.4.

A client receives blocks from a proxy and first deposits them in a so-called play-out buffer. Then, after some period of time, it gradually reads blocks out from the buffer and plays them. By deferring the play-out as illustrated in Fig. 12, the client can prepare for an unexpected delay in delivery of blocks due to network congestion or cache misses.

When a proxy server receives an RTSP TEARDOWN message from a client, the proxy server relays the message to the video server, and closes the sessions.

3.1.2 Rate Control with TFRC

TFRC is a congestion-control mechanism that enables a non-TCP session to behave in a TCP-friendly manner [22]. The TFRC sender estimates the throughput of a TCP session

sharing the same path using following equation.

$$X = \frac{s}{R\sqrt{\frac{2bp}{3}} + t_{RTO}(3\sqrt{\frac{3bp}{8}})p(1+32p^2)}$$

X is the transmit rate in bytes/second. s is the packet size in bytes. R is the round-trip time in seconds. p is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted. t_{RTO} is the TCP retransmission timeout value in seconds. b is the number of packets acknowledged by a single TCP acknowledgement.

In the system we implemented, those information are obtained by exchanging RTCP messages between the *RTCP Sender* of the proxy cache server and the client application. A client reports the cumulative number of packets lost and the highest sequence number received to a proxy. From those information, the proxy obtains the packet loss probability. RTT is calculated from the time that the proxy receives LSR and DLSR fields of a RTCP Receiver Report message and the time that the proxy receives the message. By applying the exponentially weighted moving average functions, the smoothed values are derived for both.

Although the TFRC requires a client to send feedback messages at least once per RTT, the client application employed in the experiments issues RTCP Receiver Report messages every three to six seconds. According to RTCP specifications, the sender can trigger feedback by sending an RTCP Sender Report to the receiver, but it ignores this. Thus, to make a client frequently report reception conditions, we have to modify the client application. In the current system, we employed off-the-shelf and common applications for the video server and clients so that we could verify the practicality and applicability of the proxy cache system we propose. Problems inherent in public applications remain for future research.

3.1.3 Video Quality Adjustment

The quality of the block sent to a client is adjusted so that resulting video rate fits the available bandwidth estimated by the TFRC. We employed a frame dropping filter [20] as a quality adjustment mechanism. The frame dropping filter adjusts the video quality

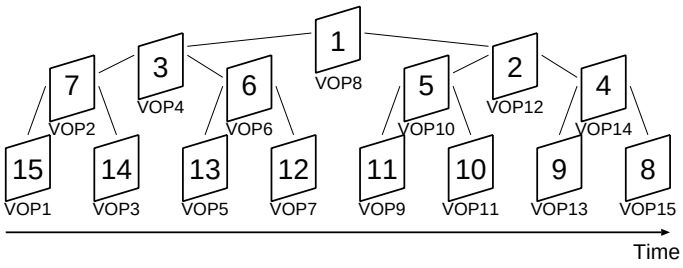

Figure 13: Frame discarding order

to the desired level by discarding frames. The smoothness of video play-out deteriorates, but it is simpler and faster than other filters such as low-pass and requantization.

We should take into account the interdependency of video frames in discarding frames. For example, discarding an I-VOP affects P-VOPs and B-VOPs that directly or indirectly refer to the I-VOP in coding and decoding processes. Thus, unlike other filters [23], we cannot do packet-by-packet or VOP-by-VOP adaptation. The frame-dropping filter is applied to a series of VOPs of one second. The *Filter* first buffers, e.g., 15 VOPs in our system, where the video frame rate is 15 fps. Then, the order for discarding is determined. To produce a well-balanced discard, we prepared a binary tree of VOPs. The VOP at the center of the group, i.e., the eighth VOP in the example, became the root of the tree and was given the lowest priority. Children of the eighth VOP were the fourth and twelfth VOPs, and respectively became the second and third candidates for frame dropping. Fig. 13 illustrates the resulting order for discarding assigned to VOPs. The order itself does not take into account VOP types. Then, considering inter-VOP relationships, we first discard B-VOPs from ones that have the lowest priority until the amount of video data fits the bandwidth. If it is insufficient to discard all B-VOPs to attain the desired rate, we move to P-VOPs. Although we could further discard I-VOPs, they have been kept in the current implementation for the sake of smooth video play-out without long pauses.

Figure 14 shows bit rate variation of filtered video streams generated aiming at 200, 500, 800 kbps from the original VBR video stream, whose average rate is 1000 kbps.

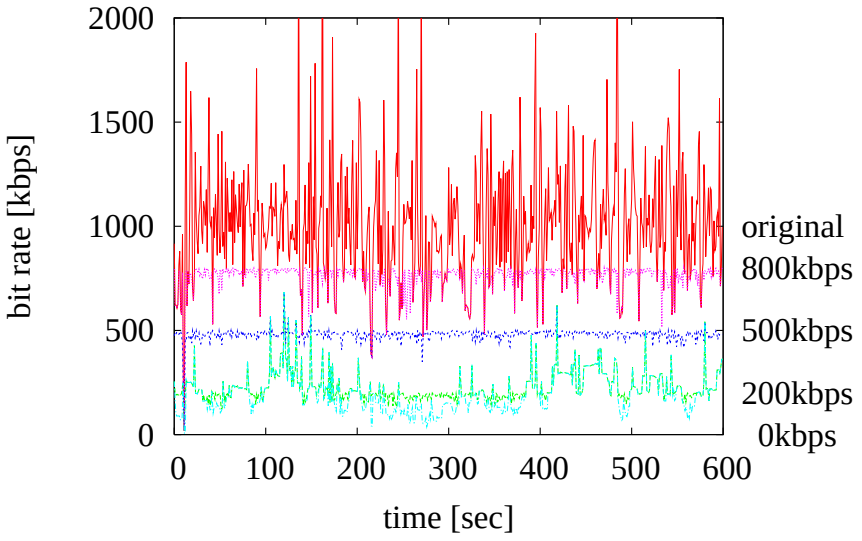

Figure 14: Adjusted video rate by frame dropping filter

3.1.4 Proxy Caching Mechanisms

Block Retrieval Mechanism: When a block is not cached in the local cache buffer, the *Cache Manager* retrieves the missing block by sending an RTSP PLAY message to the video server. In our implemented system, there is only one quality, i.e., the highest among available qualities, in blocks retrieved from the server, since a server application cannot adjust the quality of video stream. Therefore, the *Cache Manager* retrieves the missing block with the highest quality.

In addition, to use bandwidth efficiently, and prepare for potential cache misses, it also requests the video server to send succeeding blocks that are not cached and requested, by using the Range field of the RTSP PLAY message. Blocks 3 and 4 in Fig. 12 have been retrieved from the video server by sending one RTSP PLAY message. Although we have to use an SMPTE, NPT, or UTC timestamp to specify the range, there are block numbers beside the PLAY message in Fig. 12 to allow for easier understanding.

Block Prefetching Mechanism: To reduce the possibility of cache misses and avoid the delay in obtaining missing blocks from a server, a proxy prefetches blocks that clients are going to require in the future. In our implemented system, since the server, in the case the Darwin Streaming Server, cannot tell whether a request is for prefetching or for usual

retrieval, the proxy only sends prefetching request when a cache hit occurs and there is no block being transmitted.

In the case of a cache hit, the *Cache Manager* examines the *Cache Table* in succeeding P blocks. As long as blocks are cached, the *Cache Manager* sequentially reads them out and sends them to the *RTP Sender*. If there exists any block which is not cached in succeeding P blocks, the *Cache Manager* prefetches the missing block by sending an RTSP PLAY message to the video server. The *Cache Manager* also prefetches succeeding blocks that are not cached, such as in the case of a cache miss.

Cache Replacement Mechanism: When a proxy cache server retrieves a block and finds the cache is full, it discards one or more less important blocks to make room for the new block. In our implemented system, there is only one quality in the blocks retrieved from the server, since a server application cannot adjust the quality of video stream. In addition, since a client does not declare the desired level of quality, the proxy cache server cannot predict the tolerable or minimum level of quality worth keeping in the cache buffer. Thus, once the victim is chosen, it is immediately removed from the cache without applying video quality adjustment.

3.2 Experimental Evaluation

In this section, we conduct experiments to evaluate our system. In the experiments, we use a 10-minute-long video stream coded by an MPEG-4 VBR coding algorithm at the coding rate of 1 Mbps. Video data of 320×240 pixels and 30 fps and audio data of 96 Kbps are multiplexed into the MPEG-4 stream. Therefore the size of the video stream is about 84 Mbytes. A block corresponds to 300 VOPs, i.e., 10 seconds. Thus, the stream consists of 60 blocks. A video server always has the whole video blocks. A client watches a video from the beginning to the end without interactions such as rewinding, pausing and fast-forwarding.

3.2.1 Evaluation of Rate Control with Video Quality Adjustment

Figure 15 illustrates the configuration for our experimental system to evaluate the availability of rate control with video quality adjustment. A proxy is directly connected to

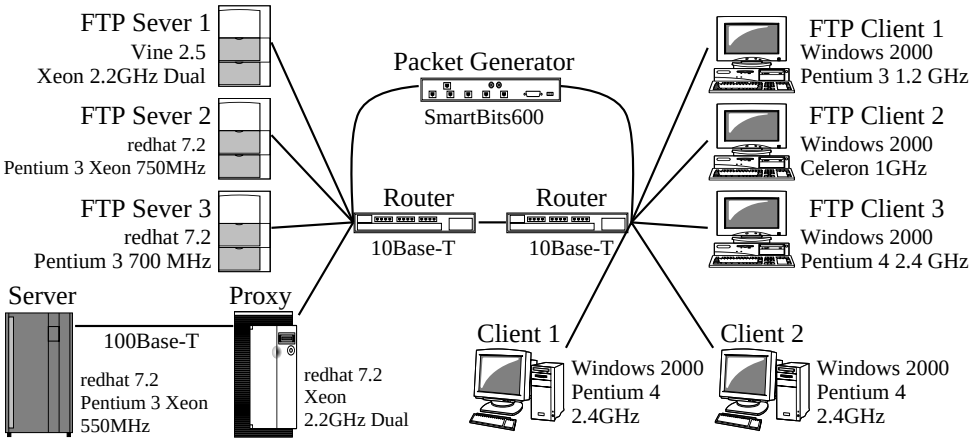

Figure 15: Configuration of experimental system

a video server. Two video clients are connected to the proxy through two routers. The video sessions compete for the bandwidth of the link between two routers with three FTP sessions and a UDP flow generated by a packet generator. The proxy has no blocks and a cache buffer capacity is limited to 50 Mbytes. The prefetching window size P is set to 5. The video client 1 issues an OPTIONS message at time zero, and the video client 2 issues it at 150 seconds. Two clients watch the same video stream. FTP sessions start transferring files at 300 seconds and stop at 450 seconds. The packet generator always generates UDP packets at the rate of 8 Mbps. For purposes of comparison, we also conducted experimental evaluations of the traditional method where the proxy does not adjust video quality.

Figures 16, 17 and 18 illustrate variations in reception rates observed at each client with tcpdump [24]. As Fig. 17 shows, the reception rate changes in accordance with network conditions. During congestion, the average throughput of TCP sessions is 277 kbps with our system. On the contrary, since the traditional system keeps sending video traffic at the coding rate, TCP session are disturbed and, the attained throughput is only 37 kbps. To conclude, by introducing the TFRC algorithm and a video-quality adjustment mechanism, our video streaming system behaves in a friendly manner with the TCP.

However, as observed in Fig. 17, there are large rate variations in video sessions. The average throughput of the video sessions during the competitive period is higher than that of TCP sessions. The major reason for this is that the control interval of adaptation is three

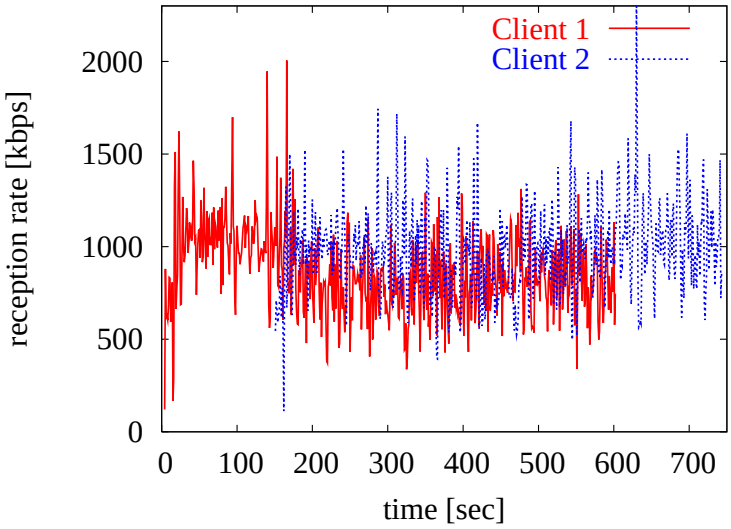

Figure 16: Reception rate variation with traditional method

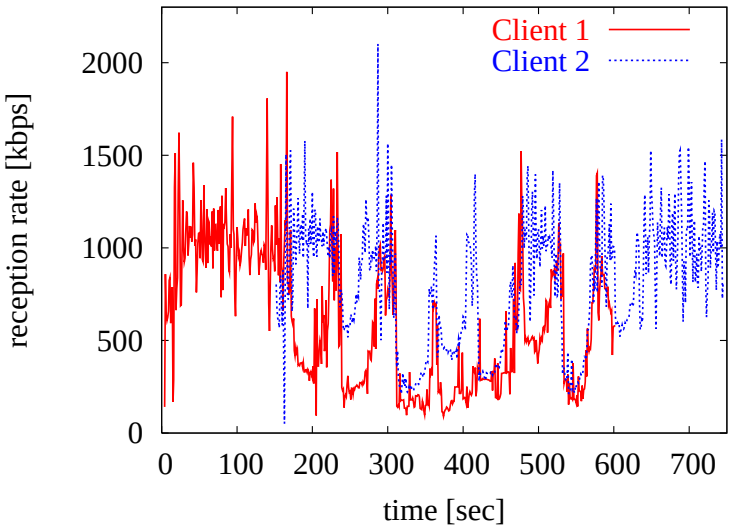

Figure 17: Reception rate variation with quality adjustment

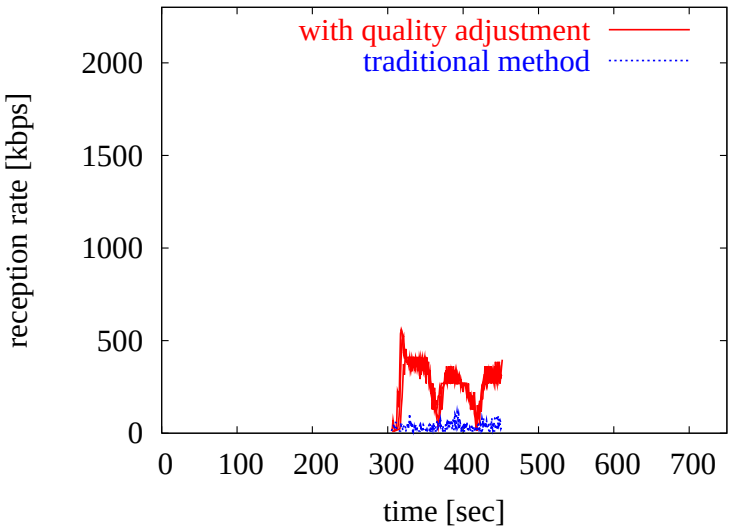

Figure 18: Reception rate variation on FTP sessions

to six seconds, which is considerably longer than that of the TCP, which reacts to network conditions in order of RTT. TCP sessions are sensitive to congestion and they suppress the number of packets to inject into the network when they occasionally observe packet losses. Video sessions, on the other hand, do not notice sudden and instantaneous packet losses due to the long control interval. By increasing the frequency that a client reports feedback information, such discrepancies are expected to be eliminated. Another reason is that the experimental system is relatively small. As a result, only a slight change during a session directly and greatly affects the other sessions. Then, synchronized behaviors are observed in Fig. 17. We plan to conduct experiments within a larger network environment where a large number of sessions co-exist.

Figures 19 and 20 show RTT and packet loss probability calculated from information in RTCP Receiver Report messages. In the traditional system, the proxy persists in sending video data at the coding rate during congestion, and many packets are delayed or lost at routers. The packet delay may cause freezes at play-out due to underflow of play-out buffer. Furthermore, the client application abandons playing out a VOP seriously damaged by packet loss. The number of packets that constitute a VOP is proportional to the size of VOP. Thus, the probability that a single packet loss affects the whole VOP is higher for I-VOP than for P-VOP, and further for P-VOP than for B-VOP. In addition,

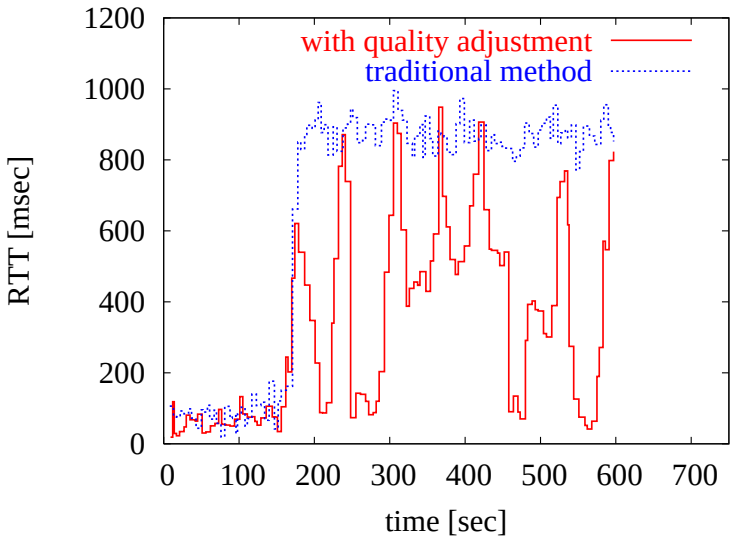

Figure 19: RTT variations at client 1

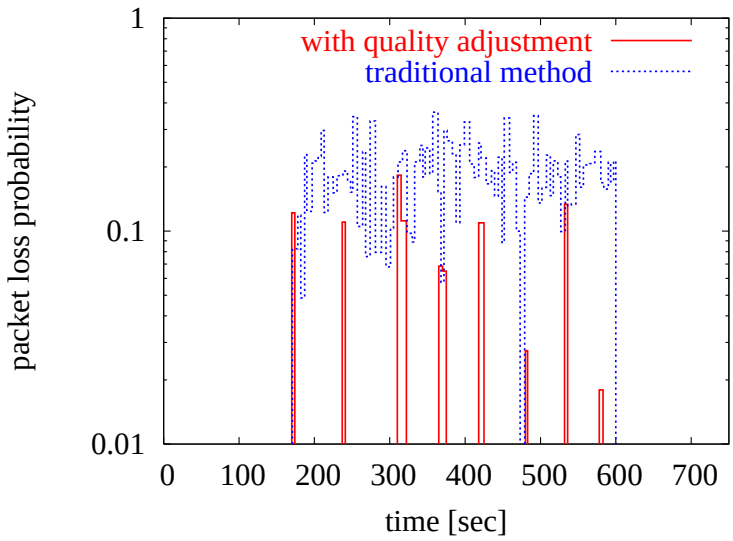

Figure 20: Packet loss probability variations at client 1

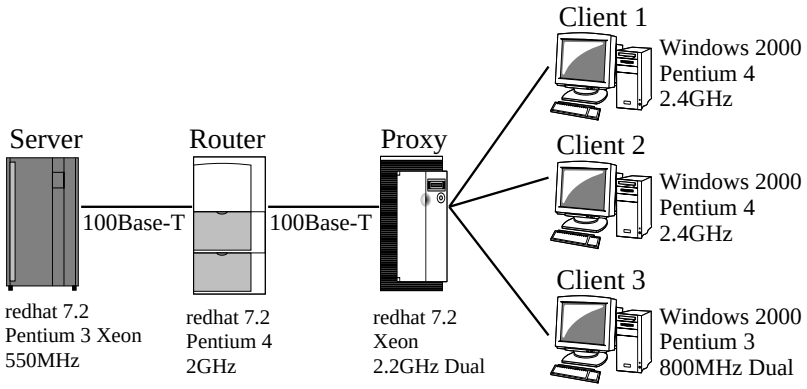

Figure 21: Configuration of experimental system

the influence of packet loss on I-VOP and P-VOP propagates to the succeeding VOPs that directly or indirectly refer to the damaged VOP. As a result, having suffered from packet losses, the user observes frequent freezes and long pauses during congestion. During experiment, 9712 VOPs are played out with our system, but only 9133 VOPs are played out with the traditional system at client 1. Therefore the perceived video quality is higher and smoother with our system than with the traditional system owing to the intentional frame discarding, although the amount of received video data in the traditional system is larger than that in our system.

3.2.2 Evaluation of Proxy Caching Mechanisms

Figure 21 illustrates the configuration for our experimental system to evaluate proxy caching mechanisms. A proxy is connected to a video server through a router. Three clients are directly connected to the proxy. In order to control the delay, NISTNet [25], a network emulator, is used at the router. The one-way delay between the video server and the proxy is set to 125 msec. Clients 1 through 3 issue an OPTIONS messages at time 0, 350, 700, respectively. Three clients watch the same video stream. The proxy has no block at first. Using this configuration, we evaluate the availability of caching mechanisms. For this purpose, we do not introduce rate control with quality adjustment at the proxy. For purposes of comparison, we also conducted experimental evaluations of cases where the proxy has no cache buffer, that is, when clients always receive video blocks from the server.

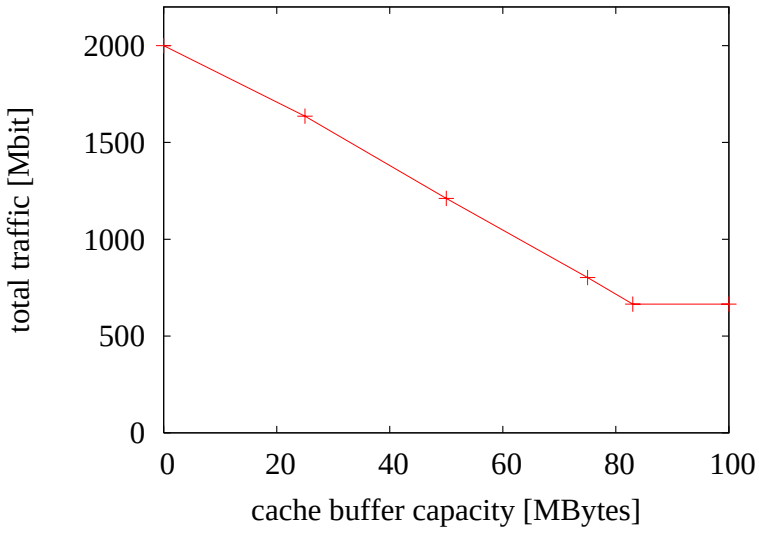

Figure 22: Total ammount of traffic between the server and the proxy (P=0)

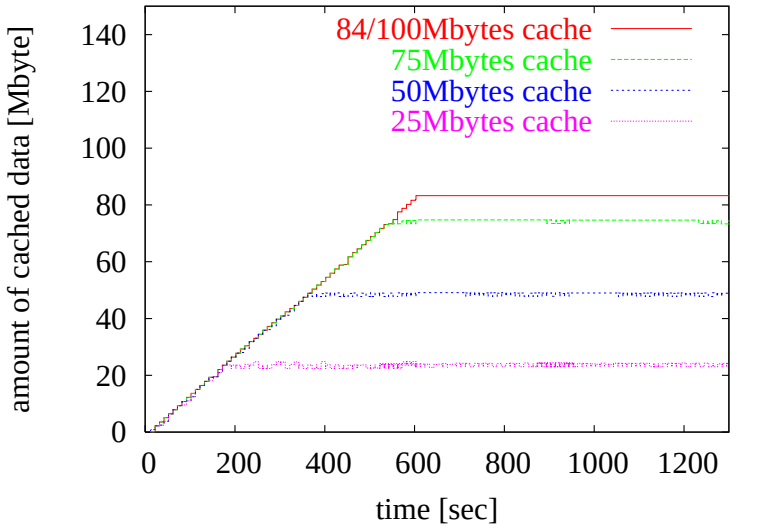

Figure 23: Amount of cached data (P=0)

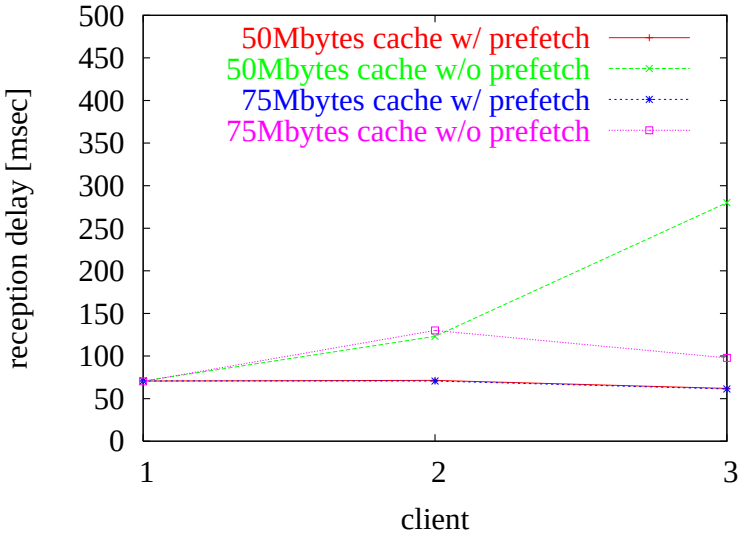

Figure 24: Reception delay

Figure 22 shows the total amount of traffic between the video server and the proxy during experiments, and Fig. 23 shows the amount of cached data. Prefetching window size P is set to zero, i.e., no prefetching. In Fig. 22, 0 Mbyte of the cache buffer capacity means the proxy has no cache buffer. As the buffer capacity increases, the total amount of traffic between the server and the proxy decreases. When the buffer capacity exceeds 84 Mbytes, i.e., the size of the whole video stream, the total amount of traffic does not change any more. In addition, the amount of cached blocks is kept within the limitation on buffer capacity as Fig. 23 shows. Consequently, it is shown that the proxy can provide clients with blocks from its local cache buffer by replacing less important blocks with newly retrieved blocks.

We define the reception delay d_{rec} as follows,

$$d_{rec} = \frac{1}{BN} \sum_{i=1}^{BN} (T_{recv}(i,j) - T_{first}(j) - T_{stamp}(i)) \tag{22}$$

where, B corresponds to the number of VOPs in a block and N stands total number of blocks in a stream. $T_{recv}(i,j)$ is a time that the client j receives the VOP i. $T_{first}(j)$ is a time that the client j receives the first VOP. Finally, $T_{stamp}(i)$ is a timestamp of VOP i, i.e., $T_{stamp} = i/F$, where F corresponds to the frame rate. Thus, the reception delay is the sum of differences between the expected arrival time of a VOP and the actual arrival

time at a client. The long reception delay d_{rec} may cause freezes due to underflow of the play-out buffer. Figure 24 shows the average of reception delay during video session at each client while prefetching window P is set to 0 or 5. Since there is no cached block in the proxy at first, reception delay of client 1 is the same whether the proxy conducts prefetching or not. However, for client 2 and 3, the reception delay without prefetching is larger than that with prefetching, since there is the delay in obtaining missing blocks from a server. Specifically, when the buffer capacity is 50 Mbytes, the reception delay on client 3 with a non-prefetching proxy is 280 msec. When client 3, the last client among three, joined the service, some parts of a video stream had been swept out from a cache buffer due to the limited capacity. As a result, the number of blocks missing in a cache buffer is larger than the other two clients. When a proxy does not have a capability of prefetching, it has to retrieve all missing blocks from a video server when they are requested by a client. It introduces delay. Consequently, the reception delay increases. By introducing the prefetching mechanism, we can avoid freezes in video play-out.

4 Design, Implementation, and Evaluation of Cooperative Proxy Caching Mechanisms

In this section, we design a video streaming system with cooperative proxy cache servers based on the system implemented in the preceding section. Then, we conduct several experiments on our implemented system to verify the practicality of our design and usefulness of our mechanisms.

4.1 Design and Implementation of System

In this subsection, we describe the system we implemented. Our mechanisms as described in subsection 2.3 also consider some ideal conditions and environments. In the case of cooperative caching, other problems, such as frame-by-frame basis data transferring at a server, arise in addition to those solved in the preceding section. In the following subsections, we explain how the system is design to implement the mechanisms for the MPEG-4 video streaming service by expanding the system described in the preceding section.

4.1.1 Overview of Implementation

Figure 25 illustrates the processes that constitute our video streaming system. The *Proxy Process* in Fig. 25 is composed of the same modules of proxy in Fig. 11. It is generated for each client. We additionally introduce a *Cooperation Process* for each proxy to communicate with neighboring proxy cache server. The *Cooperative Process* also estimates one-way propagation delay and available bandwidth among proxies. Usually, a server application sends a video stream in a frame-by-frame basis at the rate of the frame rate. For example, a video stream of 30 fps is always sent at 30 fps by a server. Thus, the required bandwidth for a video session is equal to the coding rate of the video stream. It follows that a server cannot fully utilize the bandwidth if the available bandwidth is larger than the coding rate. To efficiently use the available bandwidth as our algorithm expects, we establish a set of RTSP, RTP and RTCP sessions between a video server and a proxy for each client, although we assumed that there was only one session between them in our proposal.

Figure 26 illustrates modules that constitute *Cooperation Process*. Some modules are

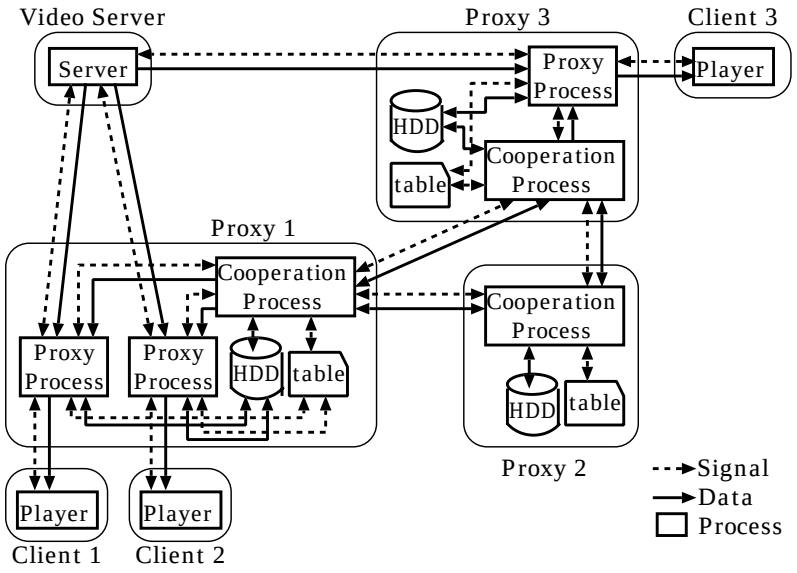

Figure 25: Outline of constitute system

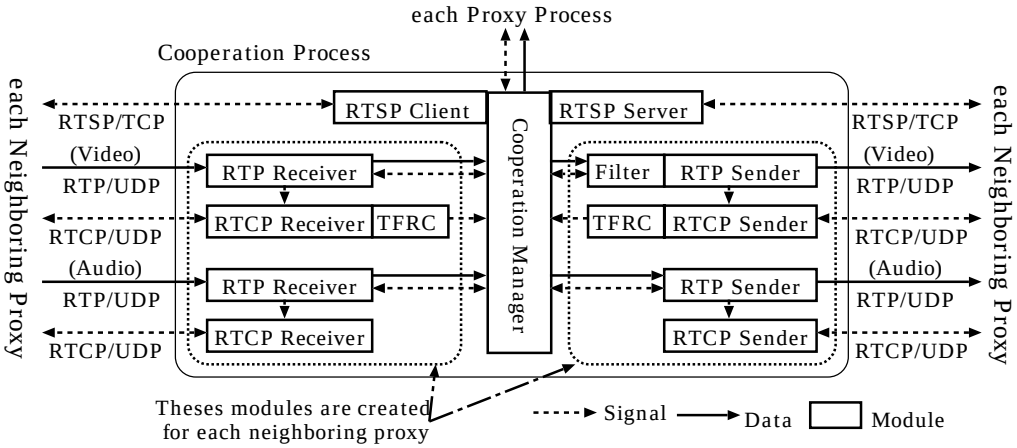

Figure 26: Modules constituting cooperation process

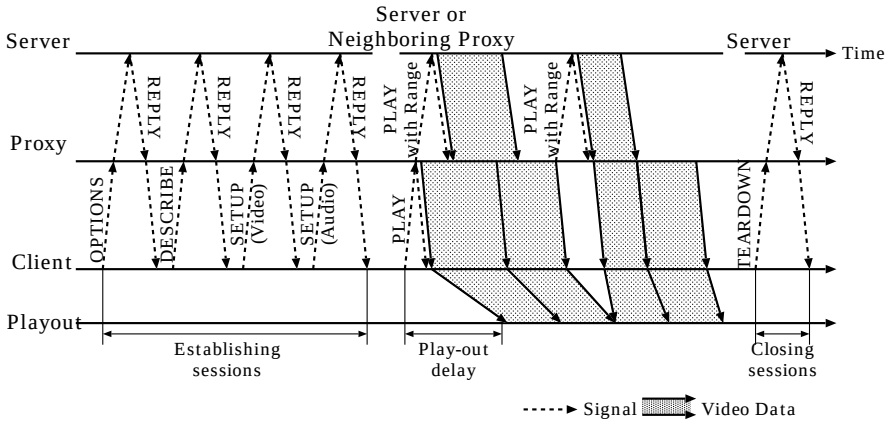

Figure 27: Basic behavior of our cooperative proxy caching system

similar to those in Fig. 11. The *Cooperation Manager* retrieves a block from neighboring proxy server thorough *RTP Receiver* and provides a block to neighboring proxy server through *RTP Sender*. A set of *RTP Sender*, *RTCP Sender*, *RTP Receiver* and *RTCP Receiver* is created for each neighboring proxy as illustrated in Fig. 26. *Cooperation Manager* also communicates with a neighboring proxy server through *RTSP Server* and *RTSP Client* using RTSP.

The basic behavior of our system is illustrated in Fig. 27. It is almost the same as Fig. 12 except that blocks are retrieved from not only the video server but neighboring proxy cache servers. Since the client applications cannot declare preferable (upper-constraint) and tolerable (lower-constraint) levels of video quality, they are considered ready to receive and perceive a video stream of an arbitrary quality.

First, a client begins by establishing connections for audio and video streams with the proxy server by sending a series of RTSP OPTIONS, DESCRIBE, and SETUP messages. These RTSP messages are relayed to the video server by *Proxy Process*. Thus, connections between the video server and the proxy server are also established at this stage. Then, the client requests delivery of the video stream by sending an RTSP PLAY message.

A proxy maintains the cache table and the remote table in the *Table*. A proxy adopts the fastest way that can provide the client with a block of higher level of quality. Details of block provisioning will be given in subsection 4.1.4.

To retrieve a block from a neighboring proxy, *Proxy Process* requests retrieving the

block to the *Cooperation Manager* in a *Cooperation Process*. The *Cooperation Manager* sends an RTSP PLAY message to the candidate server thorough the *RTSP Client*, retrieves the block through the *RTP Receiver*, and relays the block to the *Proxy Process* VOP by VOP. When reception is completed, *Proxy Process* deposits the block in its local cache buffer. If there is not enough room to store the newly retrieved block, the *Proxy Process* replaces one or more less important blocks in the cache buffer with the new block. Details of cache replacement will be also given in subsection 4.1.4.

A client receives blocks from a proxy and first deposits them in a so-called play-out buffer. Then after some period of time, it gradually reads blocks out from the buffer and plays them.

4.1.2 Sharing Information among Proxies using RTSP

In the proposed system described in subsection 2.3, a server and proxies exchange *QUERY* and *REPLY* messages each other to share information of cached data. In an actual system, an originating video server always has the whole video stream. Therefore, the proxy cache server sends *QUERY* messages only to neighboring proxy cache servers to obtain information of their caches. In our system, *QUERY* and *REPLY* messages are realized by RTSP GET_PARAMETER and RTSP REPLY messages, respectively.

4.1.3 Bandwidth Estimation by TFRC

In the system we implemented, we employ a TFRC mechanism to estimate available bandwidth between the server and proxy, proxy and client, and among proxies as described in subsection 3.1.2. RTT and packet-loss probability are obtained by exchanging RTCP messages. One-way propagation delay is estimated from RTT.

For other proxies and clients, a proxy can calculate RTT from a RTCP Receiver Report message and its reception time as described in subsection 3.1.2. However, since a proxy is a receiver of RTP packets on a session to a server, it do not receive any RTCP Receiver Report message. Thus, a proxy estimates RTT from an NTP timestamp field of a RTCP Sender Report message sent by a server and the time when it receives the message.

According to the RTCP specification, an NTP timestamp field has the wallclock time, whose absolute value is the same among a server and proxies. A server can use the elapsed

time from the session establishment instead of the wallclock time, and a proxy can also estimate RTT from its value. However, a server, the Darwin Streaming Server version 4.1.1, does not conform to the specification in the use of an NTP timestamp field. An NTP timestamp has an integer part and zero fractional part. In the experiments, to tackle the problem without ication to the server program, we use a constant value for RTT between the server and proxy.

4.1.4 Cooperative Proxy Caching Mechanisms

Block Provisioning Mechanism: A proxy adopts the fastest way that can provide a client with a block of higher level of quality in time. In our implemented system, a server, the Darwin Streaming Server, sends a video block frame-by-frame at the frame rate of a video stream. In our system, for an originating video server k', the value $\frac{a_j(q)}{r^{SP}_{s,k'}(t)}$ in equations in subsection 2.3.3 is replaced with B/F, where B corresponds to the number of frames in a block and F is the frame rate.

A proxy retrieves a block by sending an RTSP PLAY message. We used a Range field and a Bandwidth field of a PLAY message to specify a block and its quality.

Block Prefetching Mechanism: In order to achieve further effective control, a proxy prefetches blocks in accordance with client requests. Our prefetching mechanism assumes that prefetch requests are treated at an origin server and proxy servers in a different way from those for retrieving cache-missed block. To prefetch a block from another proxy, a proxy sends a RTSP PLAY message. It has an additional field, that is, a Prefetch field. In our system, since a server, Darwin Streaming Server, cannot understand the new field, the proxy does not send a prefetching request for the server.

Cache Replacement Mechanism: Since a client application does not declare the desired level of quality, a proxy cache server only discards a cached block once it is chosen as a victim.

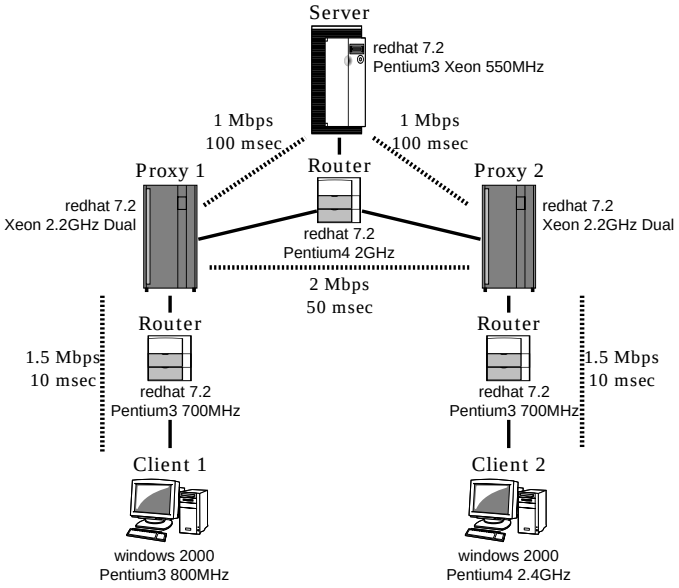

Figure 28: Configuration of experimental system

4.2 Experimental Evaluation

Figure 28 illustrates the configuration for our experimental system to evaluate our cooperative proxy caching mechanisms. Two proxies and a video server are connected through a router. There are two video clients in the system. Each video client is connected to neighboring proxy through a router. In order to control the delay and link capacity, NISTNet [25] is used at routers. The one-way propagation delay and link capacity are set as shown Fig. 28. In the experiments, we use a video stream of 200 seconds. The video stream is coded using the MPEG-4 VBR coding algorithm at the coding rate of 1 Mbps. The frame rate is 30 fps. A block corresponds to 300 VOPs, i.e., 10 seconds. Thus, the stream consists of 20 blocks, $b_1, b_2, \cdots, b_{20}$. The window of inquiry I is set to 5. The play-out delay Δ_i, and δ_i to absorb delay jitters and prediction errors are set to 3 seconds and 2 seconds, respectively.

Initial conditions of cache tables are shown in Tables 2 and 3. Proxy 1 first has blocks b_1 through b_3 of the coding rate of 1000 kbps, b_4 through b_6 of 700 kbps, and b_{11} through b_{20} of 1000 kbps as shown in Table 2. Proxy 2 has blocks b_1 through b_6 of 1000 kbps as shown in Table 3. Client 1 first issues an OPTIONS message at time zero, and client 2 issues it at 200 seconds. Two clients watch the same video stream from the beginning

Table 2: Cache table on proxy 1 (t=0)

block	bit rate [kbps]
$b_1 - b_3$	1000
$b_4 - b_6$	700
$b_7 - b_{10}$	0
$b_{11} - b_{20}$	1000

Table 3: Cache table on proxy 2 (t=0)

block	bit rate [kbps]
$b_1 - b_6$	1000
$b_7 - b_{20}$	0

to the end without interactions such as rewinding, pausing, and fast-forwarding. After 260 seconds, the link capacity between proxy 1 and proxy 2 is changed to 700 kbps. Using this configuration, we evaluate the capability of the block provisioning mechanism against changes in network conditions and cached blocks on the neighboring proxy. For this purpose, in the experiment, we do not consider other mechanisms such as block prefetching and cache replacement. We set a cache buffer capacity to 30 Mbytes, i.e., more than the size of the whole video stream, and the prefetching window to 0.

Figures 29 and 30 illustrate variations in reception rates observed at proxy 1 and client 1 with tcpdump [24], respectively. First, proxy 1 provides client 1 with cached blocks b_1 through b_3, since they have the highest quality and it is the fastest way. By providing the client with cached blocks, the proxy can afford to retrieve blocks of higher quality from proxy 2 for blocks b_4 through b_6. Since both servers do not have blocks b_7 through b_{10}, proxy 1 retrieves them from the video server. However, for 40 seconds blocks, it takes 50 seconds, since the link capacity between the server and proxy 2 is smaller than the video rate. Furthermore, a server sends additional VOPs beginning with the preceding I-VOP, if the specified range starts with a P or B-VOP. It increases the block size and introduces additional delay. However, owing to those cached blocks, all blocks arrives at

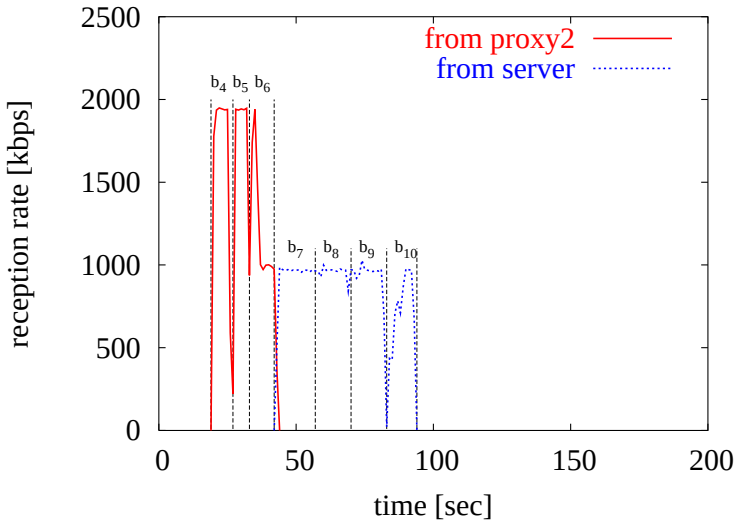

Figure 29: Reception rate variation at proxy 1

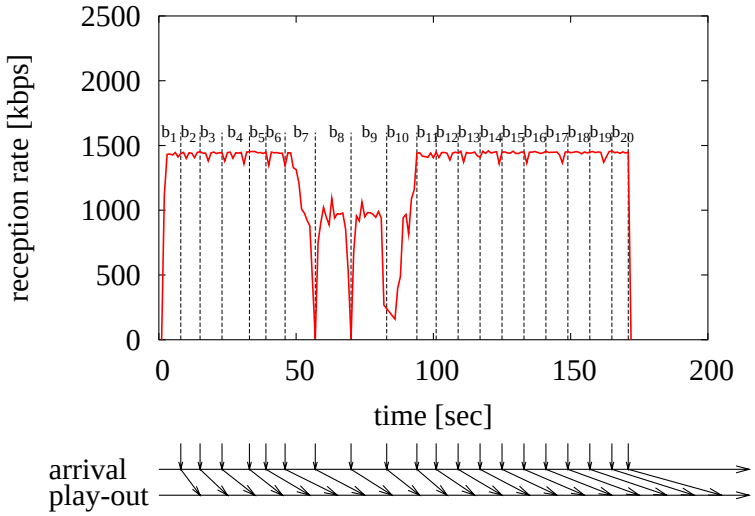

Figure 30: Reception rate variation at client 1

Table 4: Cache table on proxy 1 (t=200)

block	bit rate [kbps]
b_1 – b_{20}	1000

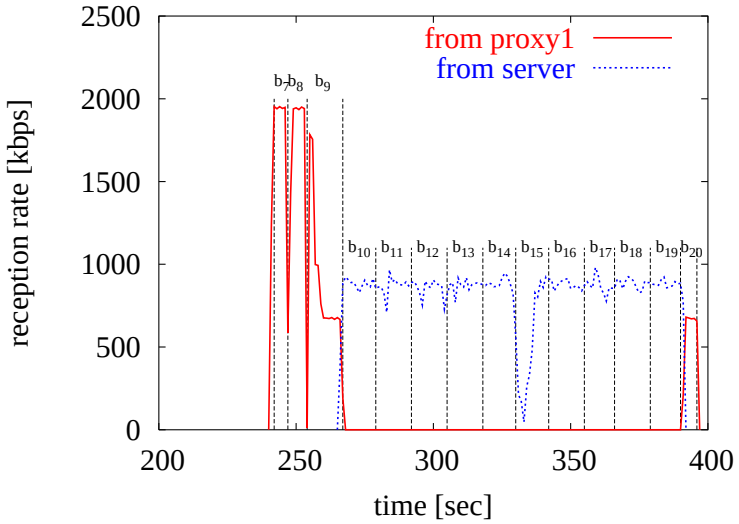

Figure 31: Reception rate variation at proxy 2

client 1 in time. In the bottom part of Fig. 30, instants of block arrivals and those of play-out at client 1 are indicated. In this experiments, the client application first caches a received video block and defers its play-out by three seconds. As Fig. 30 illustrates, a user can watch a video without freezes. On retrieving blocks, proxy 1 caches them in its local cache buffer. As a result, the cache table becomes as Table 4 shows.

Figures 31 and 32 illustrate variations in reception rates observed at proxy 2 and client 2, respectively. Proxy 2 first provides client 2 with cached blocks b_1 through b_6. For uncached blocks b_7 through b_{20}, proxy 2 tries retrieving high quality blocks from proxy 1. At 260 seconds, the capacity of the link between proxy 1 and proxy 2 is reduced to 700 kbps. The estimations of the one-way propagation delay and the available bandwidth to proxy 1 reflect the change at proxy 2. Proxy 2 moves to the video server from b_9, since the video server can provide the highest quality blocks in the fastest way. However, delays are gradually introduced in retrieving blocks from the video server due to the insufficient

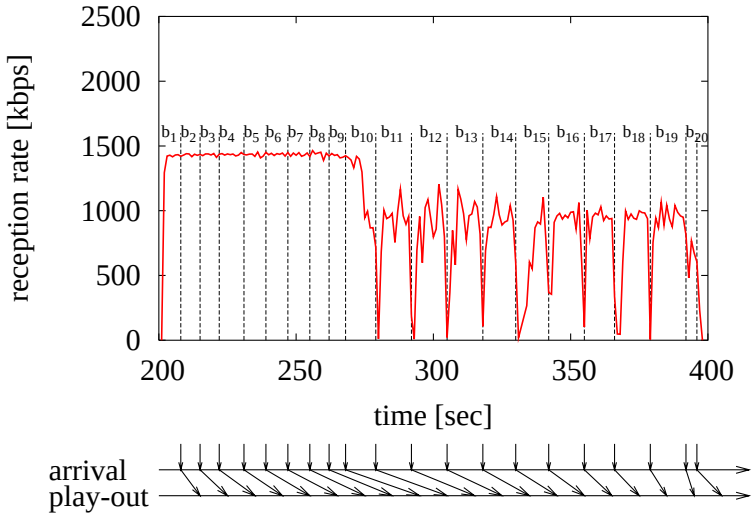

Figure 32: Reception rate variation at client 2

Table 5: Cache table on proxy 2 (t=400)

block	bit rate [kbps]
b_1 – b_{19}	1000
b_{20}	276

link capacity. At 392 seconds, proxy 2 again moves to proxy 1 to retrieve the block b_{20}. Taking into account the time needed in block transmission from proxy 1 to proxy 2 and further to client 2, the quality of block b_{20} to request to proxy 1 is intentionally reduced to 276 kbps. On receiving the request, proxy 1 applies the video quality adjustment to block b_{20} and transfers the modified block to proxy 2. Proxy 2 caches the block and provides it to client 2. As Fig. 32 illustrates, all blocks are successfully provided to client 2 through the above mentioned controls. Finally, the cache table of proxy 2 becomes as Table 5.

In the experiment, not only single proxy server could successfully provide its client with a video stream in time, but also two proxies cooperated to accomplish a continuous video-play out by offering a cached block and the capability of video quality adjustment.

5 Conclusion

In this thesis, we designed and implemented our proxy caching mechanisms on a real system for MPEG-4 video streaming service. Since we employ off-the-shelf and common applications on server and clients, we needed to tackle several problems in proposals that considered some ideal conditions and environment. Through evaluations from several performance aspects, it was shown that our proxy caching system can provide users with a continuous and high-quality video streaming service in a heterogeneous environment. Furthermore, the practicality of our design and usefulness of our mechanisms were also verified. Our mechanisms can be applied to any existing video streaming systems as far as they conform to the specifications.

However, some research issues still remain. There is room for improvement, for example, in terms of the protection of blocks against replacement, which leads to a problem of scalability. It is necessary to conduct additional experiments, e.g., in a larger network environment, with other filtering mechanisms, and with other server and client applications. We would also need to take into account user interactions such as pauses, fast forwarding and rewinding.

Acknowledgements

The author would like to express his gratitude to his supervisor, Professor Masayuki Murata of Osaka University, for his extensive help and continuous support through his studies and preparation of this thesis.

The author is most grateful to Associate Professor Naoki Wakamiya of Osaka University, for his appropriate guidance, hearty encouragement, and invaluable firsthand advice. All works of this thesis would not have been possible without his support.

The author is also indebted to Mr. Atsushi Ueoka of Hitachi Corporation, for his direct support and help. The implementation of this thesis would not have been possible without him. The author is also indebted to Mr. Fumio Noda of Hitachi Corporation who gave him helpful advice.

Thanks are also due to President Hideo Miyahara and Professor Shinji Shimojo of Osaka University; Associate Professor Masashi Sugano of Osaka Prefecture College of Nursing; Associate Professors Kenichi Baba, Hiroyuki Ohsaki and Go Hasegawa of Osaka University; Research Associate Shingo Ata of Osaka City University; and Research Assistants Shin'ichi Arakawa and Ichinoshin Maki of Osaka University, who gave him helpful comments and feedback.

The author heartily thanks his many friends and colleagues in the Department of Information Networking, Graduate School of Information Science and Technology, Osaka University, for their generous, enlightening, and valuable suggestions and advice.

Finally, the author is deeply grateful to his parents. They have always stood behind him and supported him continuously.

References

[1] "Squid." available at `http://www.squid-cache.org/`.

[2] R. Rejaie and J. Kangasharju, "Mocha: A quality adaptive multimedia proxy cache for internet streaming," in *Proceedings of NOSSDAV 2001*, pp. 3–10, June 2001.

[3] K. 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, May 2001.

[4] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal proxy cache allocation for efficient streaming media distribution," in *Proceedings of IEEE INFOCOM 2002*, June 2002.

[5] M. Reisslein, F. Hartanto, and K. W. Ross, "Interactive video streaming with proxy servers," *Information Sciences: An International Journal*, Dec. 2001.

[6] M. Hofmann, T. S. E. Ng, K. Guo, S. Paul, and H. Zhang, "Caching techniques for streaming multimedia over the internet," *Technical Report BL011345-990409-04TM*, Apr. 1999.

[7] S. Sen, J. Rexford, and D. F. Towsley, "Proxy prefix caching for multimedia streams," in *Proceedings of IEEE INFOCOM'99*, pp. 1310–1319, Mar. 1999.

[8] 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.

[9] N. Wakamiya, M. Murata, and H. Miyahara, "Video streaming systems with cooperative caching mechanisms," in *Proceedings of SPIE International Symposium ITCom 2002*, pp. 305–314, July 2002.

[10] N. Wakamiya, M. Murata, and H. Miyahara, "On proxy-caching mechanisms for cooperative video streaming in heterogeneous environment," in *Proceedings of IFIP/IEEE*

International Conference on Management of Multimedia Networks and Services 2002, pp. 127–139, Oct. 2002.

[11] "Darwin Streaming Server." available at `http://developer.apple.com/darwin/`.

[12] "RealOne Player." available at `http://www.real.com/`.

[13] "QuickTime Player." available at `http://www.apple.com/quicktime/`.

[14] ISO/IEC 14496-2, "Information technology - coding of audio-visual objects - part2: Visual," *International Standard*, Dec. 1999.

[15] Internet Streaming Media Alliance, "Internet streaming media alliance implementation specification version 1.0," Aug. 2001.

[16] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," *Internet Request for Comments 2326*, Apr. 1998.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," *Internet Request for Comments 1889*, Jan. 1996.

[18] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata, "RTP payload format for MPEG-4 audio/visual streams," *Internet Request for Comments 3016*, Nov. 2000.

[19] M. Handley and V. Jacobson, "SDP: Session description protocol," *Internet Request for Comments 2327*, Apr. 1998.

[20] N. Yeadon, F. Gracía, D. Hutchinson, and D. Shepherd, "Filters: Qos support mechanisms for multipeer communications," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1245–1262, Sept. 1996.

[21] K. Fukuda, N. Wakamiya, M. Murata, and H. Miyahara, "QoS mapping between user's preference and bandwidth control for video transport," in *Proceedings of IFIP IWQoS '97*, pp. 291–302, May 1997.

[22] M. Handley, S. Floyed, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol specification," *Internet Request for Comments 3448*, Jan. 2003.

[23] T. Yamada, N. Wakamiya, M. Murata, and H. Miyahara, "Implementation and evaluation of video-quality adjustment for heterogeneous video multicast," in *Proceedings of The 8th Asia-Pacific Conference on Communications (APCC2002)*, pp. 454–457, Sept. 2002.

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

[25] "NIST Net." available at `http://snad.ncsl.nist.gov/itg/nistnet/`.