Implementation and Evaluation of Cooperative Proxy Caching Mechanisms for Video Streaming Services

Yoshiaki Taniguchi, Naoki Wakamiya, and Masayuki Murata

Graduate School of Information Science and Technology,
Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan

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 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, proxies communicate with each other, retrieve a missing video data from an appropriate server by taking into account transfer delay and offerable quality. 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. In this paper, to verify the practicality of our mechanisms, we implemented them on a real system for MPEG-4 video streaming services, and conducted experiments. Through evaluations, it was shown that our proxy caching system can provide users with a continuous and high-quality video distribution in accordance with network condition.

Keywords: Quality Adjustment, Video Streaming Service, Proxy Caching, QoS, MPEG-4

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.[2–5] In addition, the quality of cached video data can be adapted appropriately in the proxy when clients are heterogeneous, in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality.[6]

Further author information: (Send correspondence to Y.T.)
Y.T.: E-mail: y-tanigu@ist.osaka-u.ac.jp, Telephone: +81 6 6879 4542
N.W.: E-mail: wakamiya@ist.osaka-u.ac.jp, Telephone: +81 6 6879 4541
M.M.: E-mail: murata@ist.osaka-u.ac.jp, Telephone: +81 6 6879 4540

In our previous research work,[7] 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 high-quality and low-delay video streaming service in a heterogeneous environment. Furthermore, based on our proposal, we designed and implemented our proxy caching mechanisms for MPEG-4 video streaming services on a real system which is constructed on off-the-shelf and prevailing products.[8] Through experiments, the practicality and adaptability of our mechanisms to existing video streaming services were also verified.

In addition, to provide further effective and high-quality video streaming service, we considered the cooperation among proxies.[9,10] 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. Furthermore, the practicality and adaptability of our mechanisms to existing video streaming services were also not verified yet. In this paper, we implemented our cooperative proxy caching mechanisms[9,10] on a real system for MPEG-4 video streaming services, and conducted experiments. We employ off-the-shelf and common applications for the server and client systems. Our implemented system is designed to conform to the standard protocols of video streaming systems. Through evaluations, it was shown that our proxy caching system can provide users with a continuous and high-quality video distribution in accordance with network condition. Furthermore, the practicality of our design and usefulness of our mechanisms were also verified.

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

2. COOPERATIVE PROXY CACHING MECHANISMS FOR VIDEO STREAMING SERVICES

In this section we briefly introduce our cooperative proxy caching mechanisms. For further detailed descriptions, refer to our previous works.[9,10]

2.1. Overview of Mechanisms

Our system consists of an originating video server, cooperative proxies, and heterogeneous clients. Video data are transferred over a video session established among them.

Figure 1 illustrates how servers and client communicate with each other in our video streaming system. A video stream is divided into blocks so that the cache buffer and the bandwidth can be used efficiently. 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. When the proxy cache server finishes sending out the block to the requesting client, it moves to the next block and repeats similar procedures.

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 this paper as shown in Fig. 1. Then, it begins to decode

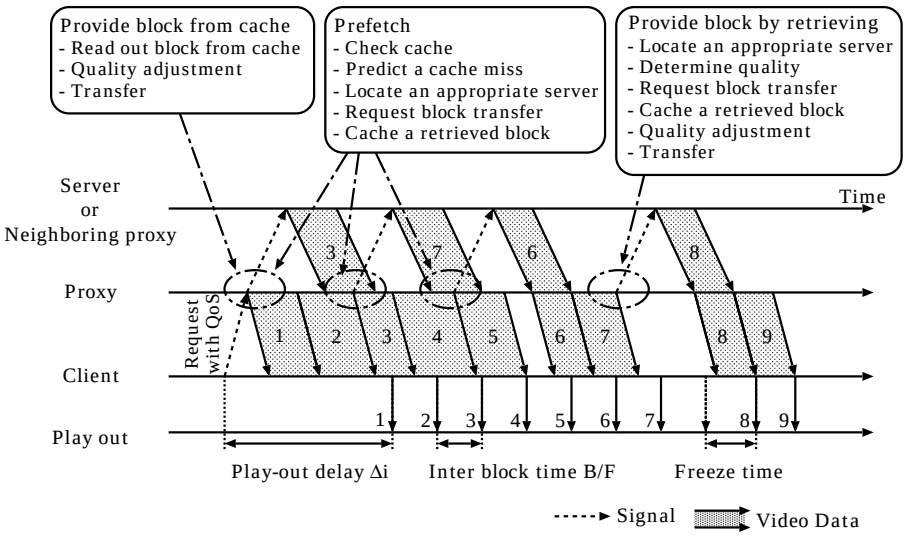

Figure 1. Basic behavior of our cooperative proxy caching mechanism

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 to wait for the arrival of necessary block is called *freeze time*. 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.

2.2. 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, or 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, one-way propagation delay, 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 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 on cached blocks at other servers. The cache tables maintain information on the cached and non-cached blocks of a video stream. Each entry in the cache table corresponds to one block of the stream and consists of a block number, the quality of the block in the cache, and a list of markers for the cached block. 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. It includes the estimated one-way propagation delay and the estimated available bandwidth for the connection, along with the list of blocks available on that server and their respective quality. 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. 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. 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. On receiving the *QUERY* message, the proxy server searches its local cache for the blocks which are listed in the message, and sets marks 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 then returns a *REPLY* message which carries a timestamp and the list of qualities for those blocks it holds which were in the list included with the *QUERY*.

The proxy issues a *QUERY* message when the block after $P-1$ reaches the end of window of inquiry. Here, P is the size of the prefetching window, which is explained in subsection 2.4. In addition, we also introduce a timer to force a refreshing of the remote tables. Timing reaches expire every $(I-P-1)B/F$.

2.3. Block Provisioning Mechanism

A 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.

For each of them, the offerable quality is derived considering one-way propagation delay, available bandwidth, condition of cached blocks on the neighboring proxy, and the deadline of providing the requested block. The fastest and finest way is chosen as far as the offerable quality is higher than the minimum request and is lower than the maximum request. If none cannot satisfy the request, the proxy chooses the fastest way.

2.4. Block Prefetching Mechanism

In order to achieve further effective control, a proxy prefetches blocks in accordance with client requests. While supplying the client 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 the request 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 proxy decides to prefetch the block if reception of the block is expected to be completed in time.

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.

2.5. 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. 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

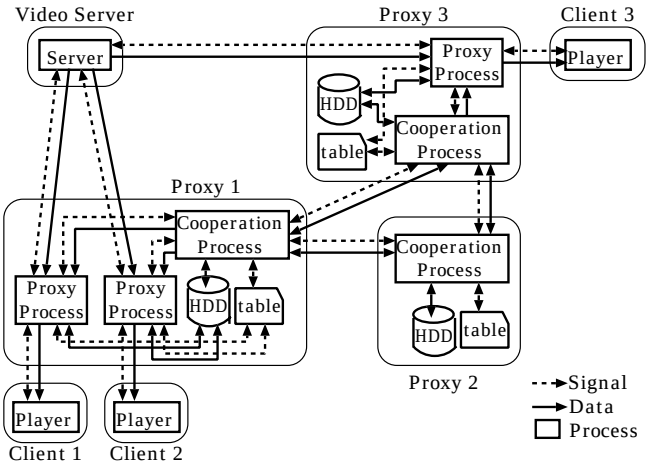

Figure 2. Outline of implemented system

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 2.4. 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. IMPLEMENTATION OF PROPOSED MECHANISMS

Our mechanisms as described in section 2 consider some ideal conditions and environment. In this section, we explain how the system is design to implement the mechanisms for the MPEG-4 video streaming service.

3.1. Overview of Implementation

We employ the Darwin Streaming Server[11] as the server application, and RealOne Player[12] and QuickTime Player[13] as the client applications. The video stream is coded using the MPEG-4 video coding algorithm. The video streaming is controlled through RTSP/TCP sessions. Each of video and audio is transferred over a dedicated RTP/UDP session, and the quality of service is monitored over RTCP/UDP sessions.

Figure 2 illustrates the processes that constitute our video streaming system. The dashed arrow and the solid arrow corresponds to signal and data flow, respectively. The *Proxy Process*, which is based on our previous work,[8] is generated for each client and provides client with video block in appropriate way. We introduce a *Cooperation Process* for each proxy to communicate with neighboring proxy cache server. The *Cooperation Process* also estimates one-way propagation delay and available bandwidth among proxies, and retrieves blocks from the neighboring proxy cache server as necessary. 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.

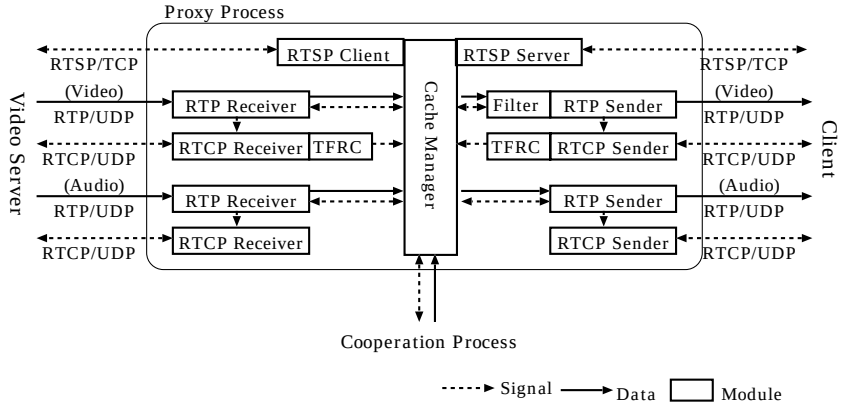

Figure 3. Modules constituting proxy process

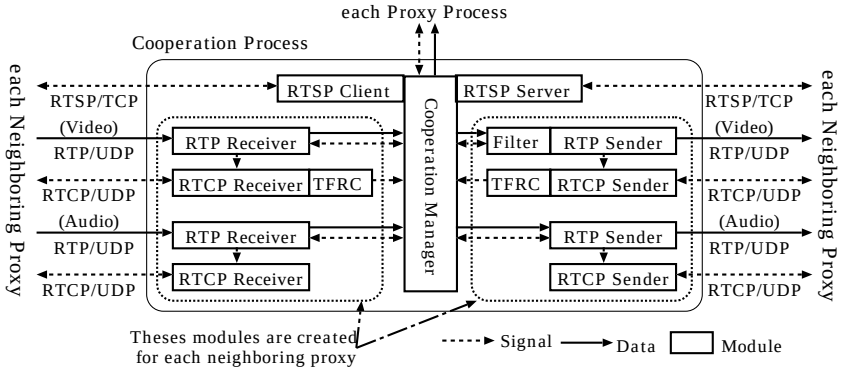

Figure 4. Modules constituting cooperation process

Details of *Proxy Process* and *Cooperation Process* are illustrated in Fig. 3 and Fig. 4, respectively. In Fig. 3, the *Cache Manager* retrieves a block from the server through *RTP Receiver* or the *Cooperation Process*, and provides a block to the client through *RTP Sender*. In Fig. 4, 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. *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. 5. 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. We empirically used 300 as the block size 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. 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.

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 received by *Cache Manager* through an *RTSP Server* module (Fig. 3) and relayed to the video server through the *RTSP Client*. Thus, connections between the video server and the proxy server are also established at this stage. On receiving

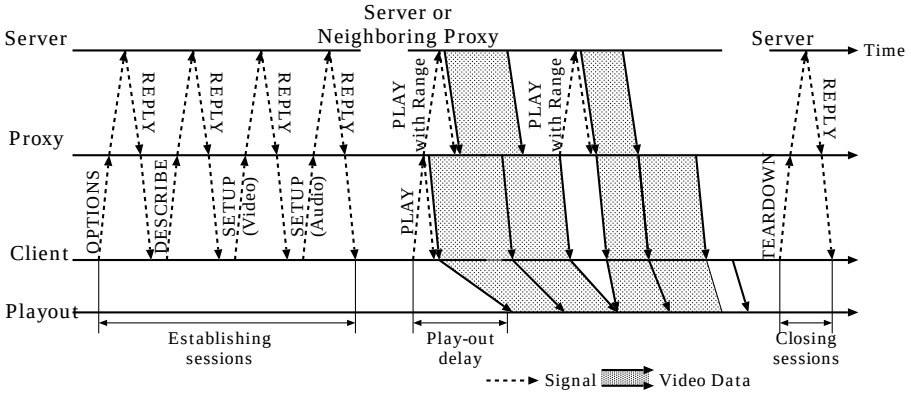

Figure 5. Basic behavior of implemented system

SETUP REPLY messages, the client requests delivery of the video stream by sending an RTSP PLAY message. Here, 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.

A proxy maintains the cache table and the remote table in the *Table*. The *Cache Manager* adopts the fastest way that can provide the client with a block of higher level of quality. When the *Cache Manager* adopts to provide the cached block, 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 necessary by the *Filter*.

When the *Cache Manager* adopts to retrieve the block from a neighboring proxy, it 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 *Cache Manager* in the *Proxy Process* VOP by VOP. When reception is completed, the *Cache Manager* deposits the block in its local cache buffer. 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. When the *Cache Manager* adopts to retrieve the block from the video server, it retrieves the block by sending an RTSP PLAY message to the video server through the *RTSP Client* in the *Proxy Process*.

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. 5, 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.

3.2. Video Quality Adjustment

We use the video coding rate as the video quality, and use a Range field and a Bandwidth field of a PLAY message to specify a block and its quality. We employed a frame dropping filter[14] as a quality adjustment mechanism. The frame dropping filter adjusts the video quality 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,[15] 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

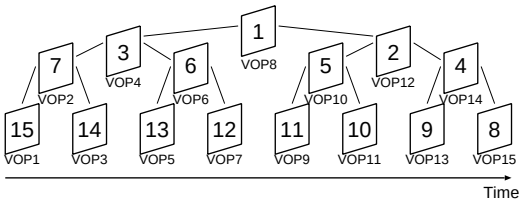

Figure 6. Frame discarding order

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. Figure 6 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.

3.3. Sharing Information among Proxies

In the proposed system described in section 2, 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.

3.4. Bandwidth Estimation by TFRC

In the system we implemented, we employ a TFRC[16] mechanism to estimate available bandwidth between the server and proxy, proxy and client, and among proxies. TFRC is a congestion-control mechanism that enables a non-TCP session to behave in a TCP-friendly manner.[16] 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. A receiver 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. 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. However, since a proxy is a receiver of RTP packets on a session to a server, it does 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 wall-clock 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 wall-clock time, and a proxy can also estimate RTT from its value. However, a server, the Darwin

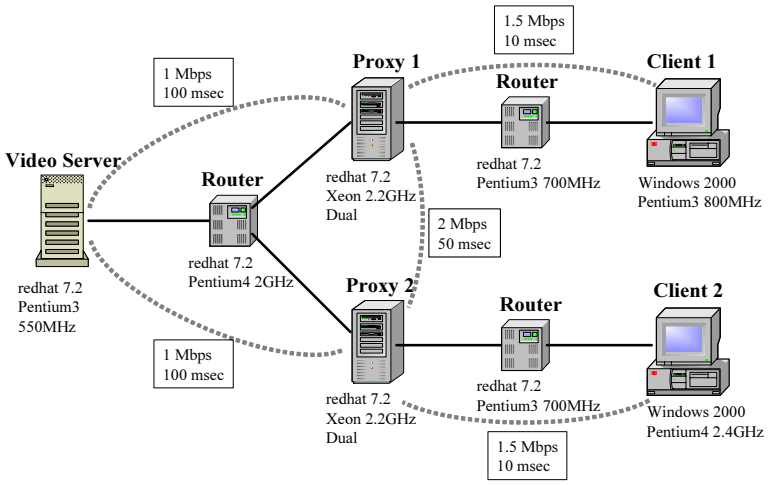

Figure 7. Configuration of experimental system

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 modification to the server program, we use a constant value for RTT between the server and proxy.

3.5. 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, we consider that it takes B/F seconds for delivery of a block, where B corresponds to the number of frames in a block and F is the frame rate.

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.

4. EVALUATION

Figure 7 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[17] is used at routers. The one-way propagation delay and link capacity are set as shown Fig. 7. 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.

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

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

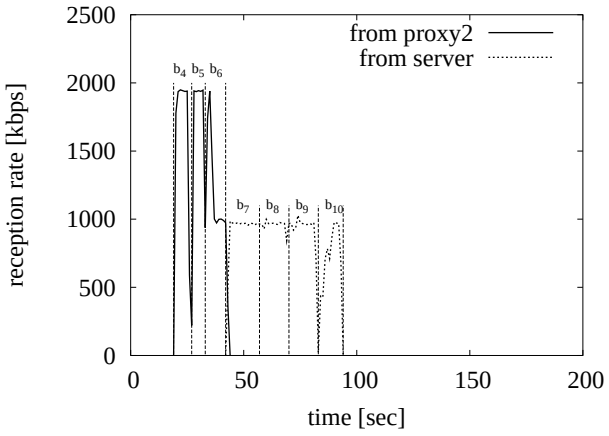

Figure 8. Reception rate variation at proxy 1

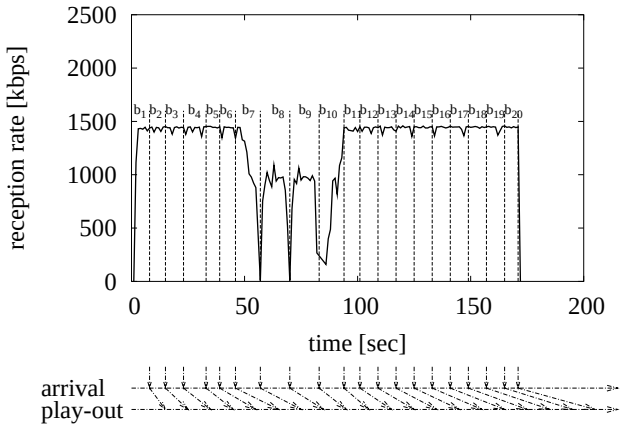

Figure 9. Reception rate variation at client 1

Initial conditions of cache tables are shown in Tables 1 and 2. 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 1. Proxy 2 has blocks b_1 through b_6 of 1000 kbps as shown in Table 2. 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 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 8 and 9 illustrate variations in reception rates observed at proxy 1 and client 1 with tcpdump,[18] 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 client 1 in time. In the bottom part of Fig. 9, 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. 9 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 3 shows.

Figures 10 and 11 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

Table 3. Cache table on proxy 1 (t=200)

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

Table 4. Cache table on proxy 2 (t=400)

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

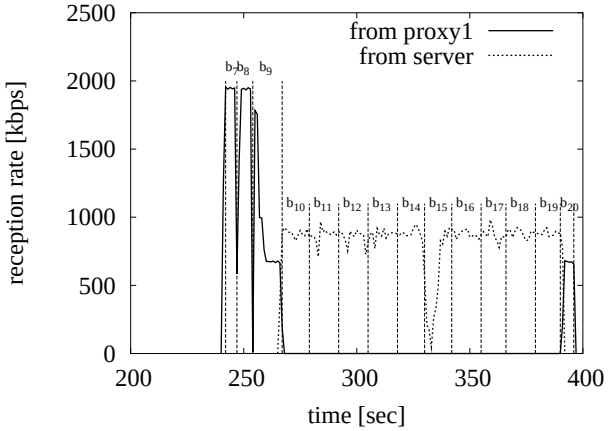

Figure 10. Reception rate variation at proxy 2

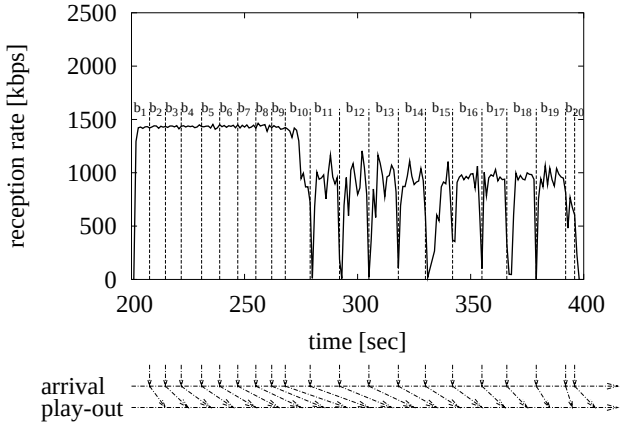

Figure 11. Reception rate variation at client 2

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 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. 11 illustrates, all blocks are successfully provided to client 2 through the above mentioned controls. Finally, the cache table of proxy 2 becomes as Table 4.

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 paper, 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, it was shown that our proxy caching system can provide users with a continuous and high-quality video streaming service. 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.

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," *IEEE Transactions on Multimedia* **6**, pp. 366–374, Apr. 2004.
5. M. Reisslein, F. Hartanto, and K. W. Ross, "Interactive video streaming with proxy servers," *Information Sciences: An International Journal* , Dec. 2001.
6. B. Shen, S.-J. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Transactions on Multimedia* **6**, pp. 375–386, Apr. 2004.
7. 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* **E86-B**, pp. 1849–1858, June 2003.
8. Y. Taniguchi, A. Ueoka, N. Wakamiya, M. Murata, and F. Noda, "Implementation and evaluation of proxy caching system for MPEG-4 video streaming with quality adjustment mechanism," in *Proceedings of The 5th AEARU Workshop on Web Technology*, pp. 27–34, Oct. 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. N. Yeadon, F. Gracía, D. Hutchinson, and D. Shepherd, "Filters: Qos support mechanisms for multipeer communications," *IEEE Journal on Selected Areas in Communications* **14**, pp. 1245–1262, Sept. 1996.
15. 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.
16. M. Handley, S. Floyed, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol specification," *Internet Request for Comments 3448* , Jan. 2003.
17. "NIST Net." available at `http://snad.ncsl.nist.gov/itg/nistnet/`.
18. "The protocol packet capture and dumper program - tcpdump." available at `http://www.tcpdump.org/`.