Implementation and Evaluation of Proxy Caching System for MPEG-4 Video Streaming with Quality Adjustment Mechanism

Yoshiaki Taniguchi[1], Atsushi Ueoka[2], Naoki Wakamiya[1], Masayuki Murata[1], and Fumio Noda[2]

[1] Department of Information Networking,
Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8351, Japan
Tel: +81-6-6850-6588, Fax: +81-6-6850-6589
{y-tanigu, wakamiya, murata}@ist.osaka-u.ac.jp

[2] Systems Development Laboratory, Hitachi Ltd.
1099 Ohzenji, Asao-ku, Kawasaki, Kanagawa, 215-0013 Japan
{at-ueoka, noda}@sdl.hitachi.co.jp

Abstract

With the increase in computing power and the proliferation of the Internet, video streaming services have become widely deployed. In this paper, we propose, design, implement, and evaluate a proxy caching system for MPEG-4 video streaming services. With our system, high-quality, low-delay, and scalable video distribution can be accomplished. Through evaluations conducted from several performance aspects, we proved that our proxy caching system can provide users with a continuous and high-quality video streaming service in accordance with network condition.

Keywords: Quality Adjustment, Video Streaming Service, Proxy Caching, 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. By applying this proxy mechanism to video streaming systems, high-quality and low-delay video distribution can be accomplished without imposing extra load on the system [1, 2, 3, 4]. 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.

In this paper, we propose and design a proxy caching system for MPEG-4 video streaming services to attain high-quality, continuous, and scalable video distribution. In our system, 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

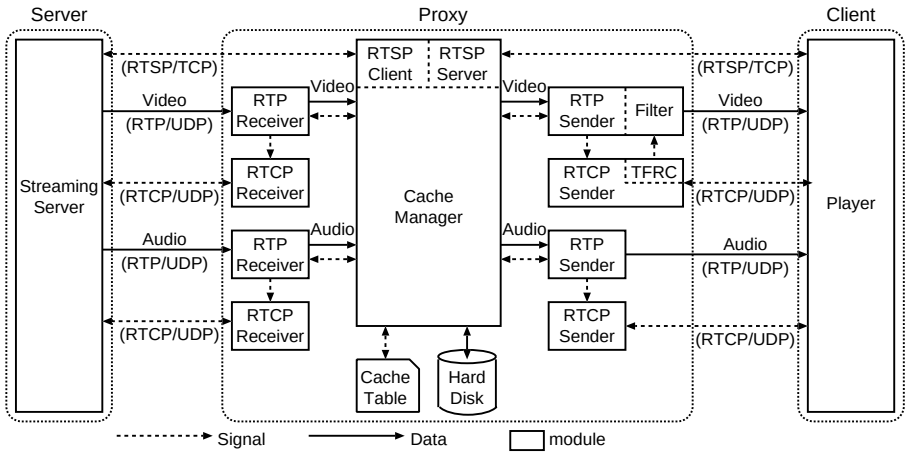


Figure 1: Modules constituting system

video block to the appropriate level through video filters to handle client-to-client heterogeneity without involving the original video server.

We implemented our proxy caching system for MPEG-4 video streaming services on a real current system. We employed off-the-shelf and common applications for the server program and the client program. Our proxy caching system can be applied to environments in that RTSP/TCP was used to control video streaming and RTP/UDP to deliver them. We introduced a TFRC (TCP Friendly Rate Control) mechanism to the system for video streaming to share the bandwidth fairly with conventional TCP sessions. We used 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, we proved that our proxy caching system could dynamically adjust the quality of video streams to suit network conditions while providing users with a continuous and high-quality service.

The rest of this paper is organized as follows. Section 2 describes our proxy caching system and explains how it is implemented. Section 3 discusses several experiments we did to evaluate our system. Section 4 is the conclusion.

2 Proxy Caching System with Video Quality Adjustment

In this section, we describe the system we implemented. Fig.1 illustrates the modules that constitute our video streaming system. The video streaming was controlled through RTSP (Real Time Streaming Protocol) [5] / TCP sessions. There were two sets of sessions for the client. The first was established between the originating video server and proxy to retrieve uncached blocks. The other was between the proxy and client. Each of video and audio was transferred over a dedicated RTP (Realtime Transport Protocol) [6] / UDP session. The quality of service was monitored over RTCP (RTP Control Protocol) / UDP sessions. The video stream was coded using the MPEG-4 video coding algorithm, and it was compliant with ISMA 1.0 [7]

2.1 Basic Behavior

Figure 2 illustrates the basic behavior of our system. In the proxy cache server, a video stream is divided into blocks so that the cache buffer and the bandwidth can be efficiently used. Each block corresponds to a sequence of VOPs (Video Object Planes) of MPEG-4. 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 by maintaining the cache and transferring data block-by-block. The strategy used to determine the block size is beyond the scope of this paper. We used 300 in our empiric implementation. Since an MPEG-4 video stream is coded at 30 frames per second, a block corresponds to ten seconds of video. Segmentation based on VOP was reasonable since packetization based on this is recommended in RFC3016 [8]. Furthermore, we could use the range field of the RTSP PLAY message to

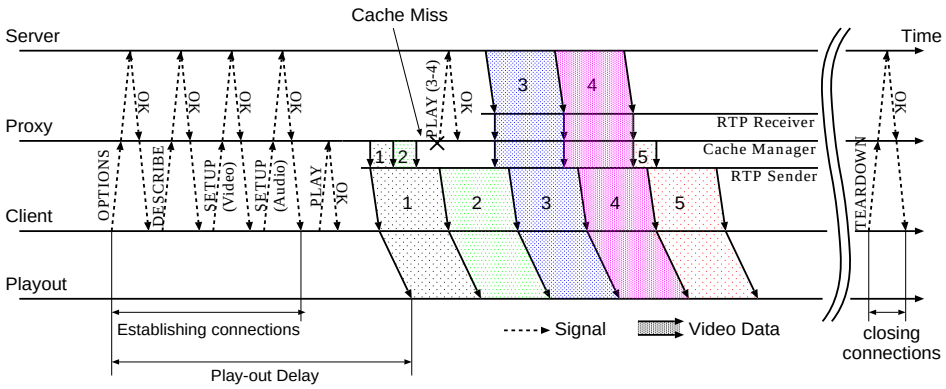


Figure 2: Basic behavior of our system

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/video streams with the proxy server using a series of RTSP OPTIONS, DESCRIBE, and SETUP messages. An OPTIONS message is used to request communication options. A DESCRIBE message is used for media initialization and a SETUP message is used for transport parameter initialization. These RTSP messages are received by the *Cache Manager* through an *RTSP Server* module (in Fig.1). The proxy server relays RTSP OPTIONS, DESCRIBE, and SETUP messages to the video server. 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 information about cached blocks in the *Cache Table*. Each entry in the table contains a block identifier, 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. It first examines the table. As long as blocks of the requested stream are cached, the *Cache Manager* sequentially reads them out and sends them to the client through the *RTP Sender*. 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 regularly sending an RTSP OPTIONS message after 90 seconds 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. To use bandwidth efficiently, and prepare for potential cache misses, it also requests the video server to send succeeding blocks that are not cached, by using the range field of the RTSP PLAY message. Blocks 3 and 4 in Fig.2 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 identifiers beside the PLAY message in Fig.2 to allow for easier understanding.

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. 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.

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.2, the client can prepare for unexpected delay in deliverly 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.

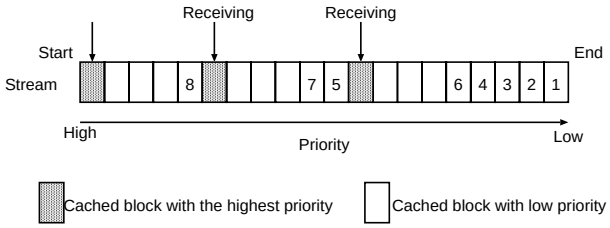


Figure 3: Priority of cached blocks

2.2 Cache Replacement

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. First, the *Cache Manager* selects the video stream with the lowest priority from cached streams using the LRU (Least Recently Used) algorithm. It then assigns priority to blocks in the selected stream using the following algorithm. Blocks being received from the video server have the highest priority. The block at the beginning of the stream is also assigned the highest priority to provide potential clients with a low-latency service. Of the others, those closer to the end of a longer succession of cached blocks are given lower priorities. Finally, blocks candidate for replacement are chosen one by one until sufficient capacity becomes available.

Figure 3 has an example of victim selection. In this example, the block located at the end of the stream is in the longest succession. Therefore, the block is given the lowest priority and becomes the "1"st victim. Among successions of the same length, the one closer to the end of the stream has lower priority.

2.3 Rate Control with TFRC

TFRC is a congestion control mechanism that enables a non-TCP session to behave in a TCP-friendly manner [9]. The TFRC sender estimates the throughput of a TCP session sharing the same path in accordance with network condition, expressed in terms of loss event rate, RTT, and packet size.

In the system we implemented, this information is 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 RTSP 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 proxy cache system we propose. Problems inherent in public applications remains for future research.

2.4 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 [10] 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 re-quantization.

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/decoding processes. Thus, unlike other filters [11], 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

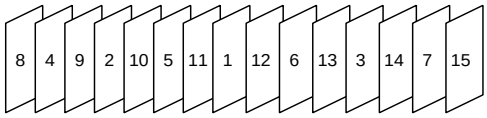


Figure 4: Frame discarding order

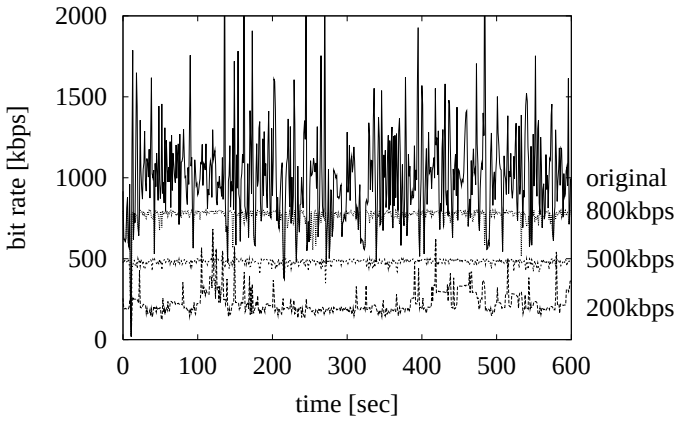


Figure 5: Adjusted video rate by frame dropping filter

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.4 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 5 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.

3 Evaluation

Figure 6 has the configuration for our experimental system. A proxy is directly connected to a video server. One video client is connected to the proxy through two routers. The video session competes for the bandwidth of the link with two routers with three FTP sessions and a UDP flow generated by a packet generator. In the experiment, the video client issues the OPTIONS message at time zero. FTP sessions start transferring files at 180 seconds and stop at 420 seconds. The packet generator always generates UDP packets at the rate of 8.5 Mbps. Using this configuration, we evaluated the capability of adjusting video quality dynamically against changes in network conditions. For this purpose, the proxy as well as the server had the whole video stream in a cache buffer of 200 MBytes. A ten-minutes-long video stream was coded by an MPEG-4 CBR coding algorithm targeted at rate of 1 Mbps. Video data of 320×240 pixels and 30 fps and audio data of 96 Kbps were multiplexed into the MPEG-4 stream. A block corresponds to 300 VOPs, i.e, 10 seconds. Thus, the stream consists of 60 blocks. For purposes of comparison, we also conducted experimental evaluations of the traditional method where the proxy does not adjust video quality.

Figures 7(a) and (b) have variations in reception rates observed at each client with `tcpdump`. As Fig.7(a) shows, the video reception rate is regulated in accordance with network conditions. During congestion, the average throughput of TCP sessions is 185 kbps with our system. On the contrary, since the traditional system keeps sending video traffic at the coding rate, TCP session are disturbed

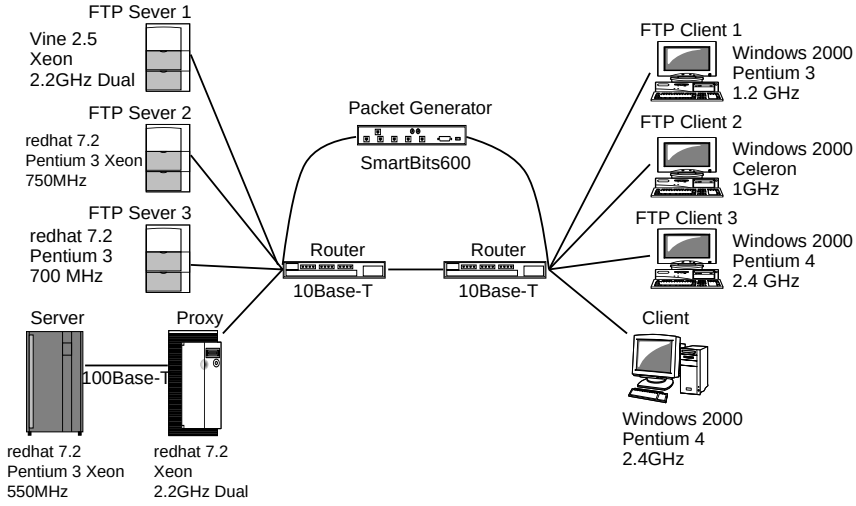
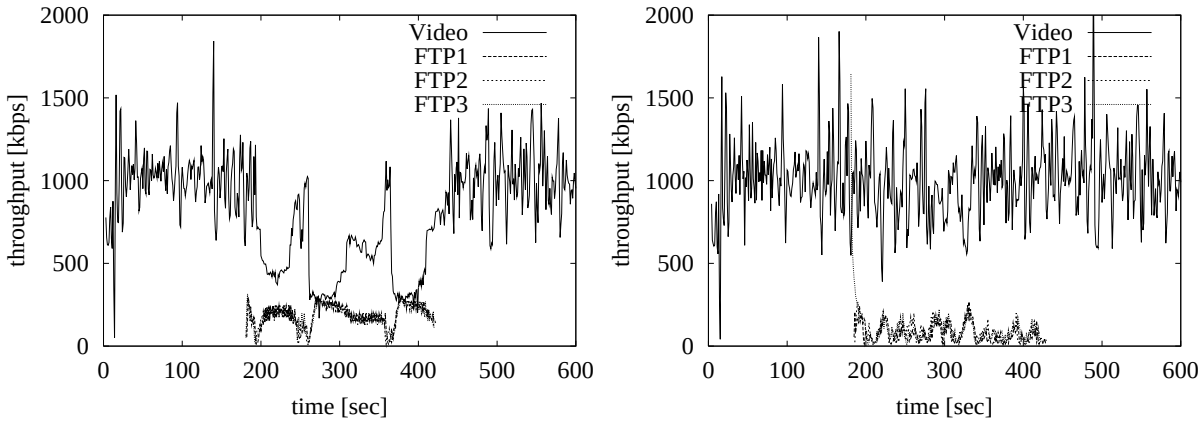


Figure 6: Configuration of experimental system



(a) With quality adjustment

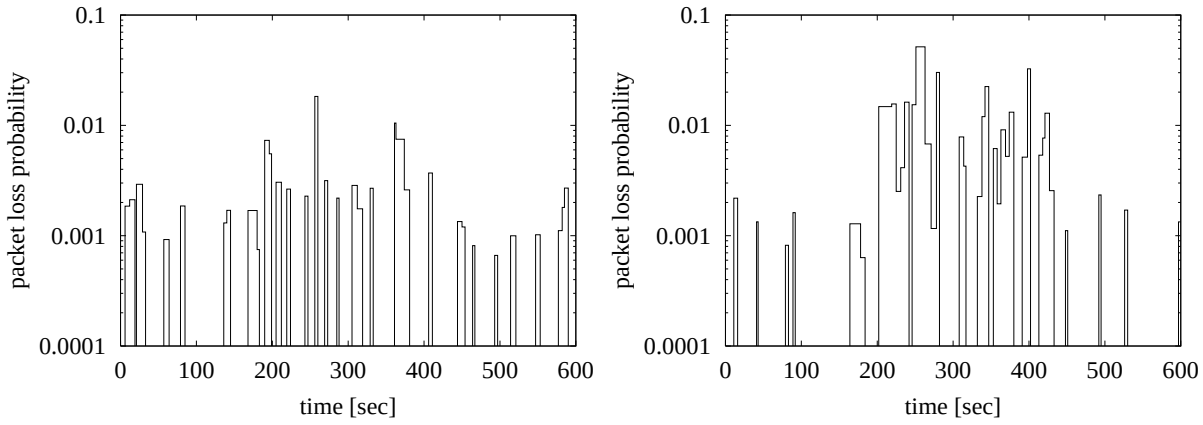
(b) Traditional method

Figure 7: Reception rate variations

and, the attained throughput is only 92 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.7(a), there are large rate variations in video session and the average throughput of the video session 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 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.7(a). We plan to conduct experiments within a larger network environment where a large number of sessions co-exist.

Figures 8 (a) and (b) show packet loss probability calculated from information in RTCP Receiver Report messages. The high and bursty packet loss in the traditional system leads to degradation of video quality perceived by the client. We evaluate the video quality in terms of jerkiness, blockiness, blurriness, and noise using the VideoQoS2 by GENISTA Corporation. Jerkiness corresponds to the smoothness of video play-out. Blockiness is related to degradation in quality caused by loss of data



(a) With quality adjustment (b) Traditional method

Figure 8: Packet loss probability variations

Table 1: Summary of video-quality evaluation

		Blockiness	Blurriness	Noise	Jerkiness
With quality adjustment	average	31	10	83	44
	variance	142	17	139	1290
Traditional method	average	31	10	79	65
	variance	263	128	2810	2410

and a high compression ratio that leads to block-shaped regions or mosaic in the image. Blurriness corresponds to the fineness of the outline of objects in an image. Finally, noise in VideoQoS2 expresses the degree of noise observed around the edge or in the smooth regions of the image. Due to space limitations, there is only a summary of the results in Table 1. Independent of measurements, a larger value indicates lower quality. These values were calculated within a congested period from 180 to 420 seconds. As the table reveals, except for jerkiness, video quality does not differ much between systems. Variations in traditional system are larger than those in our system due to the frequent packet loss. It is obvious that video play-out with our system is smoother than with the traditional system, although the VOPs are intentionally discarded by the frame dropping filter. In the traditional system, the proxy persists in sending video data at the coding rate during congestion, and many packets are lost at routers. 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, 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, being suffered from packet losses, the user observes frequent freezes and long pauses during congestion, and a jerkiness increases in the traditional system. The perceived video quality is higher 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.

4 Conclusion

In this paper, we proposed, designed, implemented, and evaluated a proxy caching system for MPEG-4 video streaming services. Due to space limitations, some experimental results including a case that involved cache replacement had to be omitted from the paper, but the fundamental experiments, revealed that our proxy caching system can dynamically adjust the quality of video streams to prevailing network conditions while providing users with a continuous and high-quality video streaming service.

In future research work, we plan to conduct additional experiments, e.g., within 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] R. Rejaie and J. Kangasharju, "Mocha: A quality adaptive multimedia proxy cache for internet streaming," in *Proceedings of NOSSDAV 2001*, June 2001.

[2] 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.

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

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

[5] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," *Internet Request for Comments 2326*, Apr. 1998.

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

[7] Internet Streaming Media Alliance, "Internet Streaming Media Alliance Implementation Specification Version 1.0," Aug. 2001.

[8] 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.

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

[10] 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.

[11] 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.