PAPER

Proxy Caching System for MPEG-4 Video Streaming with Quality Adjustment Mechanism

Yoshiaki TANIGUCHI[†], **Naoki WAKAMIYA**[†], *and* **Masayuki MURATA**[†], *Members*

SUMMARY With the growth of computing power and the proliferation of the Internet, video streaming services 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, the high-quality, low-delay, and scalable video distribution can be accomplished. In our system, 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 proxy server adjusts the quality of cached or retrieved video block appropriately, because clients are heterogeneous, in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality. 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 a heterogeneous environment.

key words: *video streaming service, proxy caching, quality adjustment, 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 [1-5]. 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 [6-8]. These papers describe proxy caching mechanisms for video streaming services. However, they do not consider the client-to-client heterogeneity, lack the scalability and adaptability to rate and quality variations, or assume specially designed server/client applications which are not available now.

In our previous research work [9], we proposed proxy caching mechanisms to accomplish high-quality and continuous video streaming service in a heterogenious environments. 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 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. However, in our proposal, 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. These assumptions do not hold in today's Internet video streaming services.

The major objective of this paper is to verify the practicaling and adaptabling of our systems to existing video streaming services. For this purpose, based on our previous proposal, we newly propose and design a proxy caching system, which requires no special server/client applications, for MPEG-4 video stream-

Manuscript received June 25, 2004.
Manuscript revised June 25, 2004.
Final manuscript received June 25, 2004.

[†]The authors are with Department of Information Networking, Graduate School of Information Science and Technology, Osaka University, Suita-shi, Osaka 565-0871, Japan.

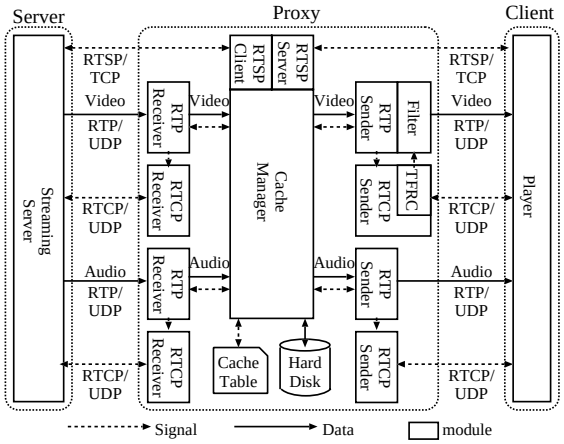


Fig. 1 Modules constituting system

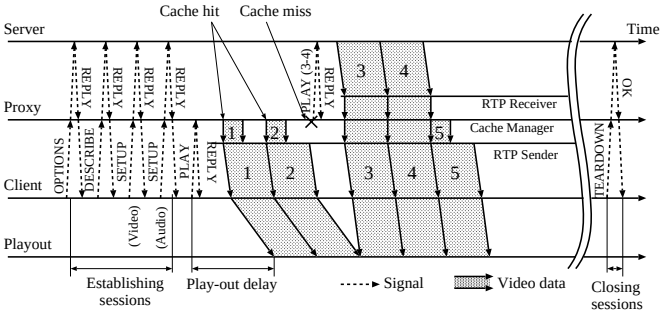


Fig. 2 Basic behavior of our system

ing services to attain high-quality, continuous, and scalable video distribution. We employed off-the-shelf and common applications for the server program and the client program to implement our system. 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) [10] 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. Furthermore, it was shown our proxy caching system can reduce the traffic between a video server and a proxy in the limited cache buffer.

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 to verify the practicality of our system. Section 4 concludes the paper and describes some future works.

2. Proxy Caching System with Video Quality Adjustment

Figure 1 illustrates the modules that constitute our video streaming system. We employed the *Darwin Streaming Server* [11] as the server application, and *RealOne Player* [12] and *QuickTime Player* [13] as the client applications. The video streaming was controlled through RTSP/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/UDP session. The quality of service was monitored over RTCP/UDP sessions. The video stream was coded using the MPEG-4 video coding algorithm, and it was compliant with ISMA 1.0 [14].

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 [15]. 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/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. 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.

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. The request is divided into blocks, and *Cache Manager* examines the table every interval of the block. If the requested block is cached, i.e., cache hit, the *Cache Manager* reads it out and sends it to the *RTP Sender*. The *RTP Sender* packetizes the block and send the packet to the client on time. 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 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. However if the retrieved block is damaged by packet loss, the *Cache Manager* doesn't 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.

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 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.

2.2 Block Retrieval Mechanism

When a requested block is not cached in the local cache buffer, the *Cache Manager* should retrieve the block. In our proposal [9], a proxy appropriately determines the way to provide a client with a requested block, taking into account the quality $Q_{pc}(i,j)$ of block j that can be sent to client i in time, the quality $Q_{sp}(i,j)$ of block j that can be retrieved from the server in time, and the quality $Q_{cache}(j)$ of cached block j. We proposed three methods to determine the quality $Q_{req}(i,j)$ of block j to reqest to the server. For example, $Q_{req}(i,j)$ is detemined such that the retrieved block j can satisfy expected demands on the block in the near future as $Q_{req}(i,j) = \min(\max_{k\in\mathcal{S},0\le l\le j} Q_{pc}(k,l), Q_{sp}(i,j))$ where $\mathcal{S}$ is a set of clients which are going to require block j in the future.

However, since we adopt an off-the-shelf application for the video streaming server, it cannot adjust the quality of video block. Therefore, in our system, the *Cache Manager* always retrieves the missing block with the highest quality, i.e., the quality with which the video stream was coded, from the video server. Of course, when we have a video server capable of the quality adjustment, our proposed scheme can attain the more efficient and effective control.

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 [10]. The TFRC sender estimates the throughput of a TCP session sharing the same path using following equation.

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

where 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. The estimated throughput obtained by Eq. (1) is regarded as the available bandwidth, which is taken into account in determining the quality of a block to retrieve and send.

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 [16] 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 [17]. Adopting layered or multiple-description coding is also helpful to treat the client-to-client heterogeneity. However, no commercially or freely available client application can decode and display a media stream with multiple layers or multiple descriptions.

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 [18], 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. Figure 3 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.

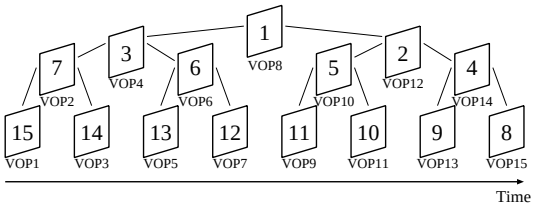


Fig. 3 Frame discarding order

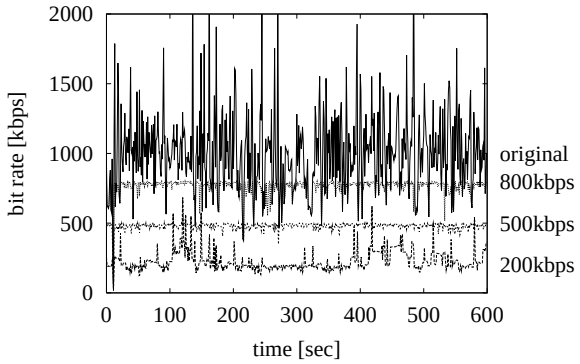


Fig. 4 Adjusted video rate by frame dropping filter

Figure 4 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.

2.5 Block Prefetching

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 a case of a cache hit, the *Cache Manager* examines the *Cache Table* in succeeding P blocks. Here, P is the size of a sliding window called a prefetching window, which determines the range of examination for prefetching. 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 like a case of a cache miss.

2.6 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. In our system,

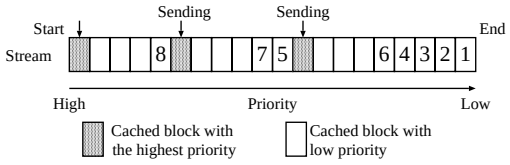


Fig. 5 Priority of cached blocks

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 although our proposal first tries the quality adjustment to decrease the size of the victim [9].

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 sent to a client 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 [19]. 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 5 illustrates 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.

3. 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.

3.1 Evaluation of Rate Control with Video Quality Adjustment

Figure 6 illustrates the configuration for our experimental system to evaluate the availability of rate control with video quality adjustment. A proxy is directly connected to 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.

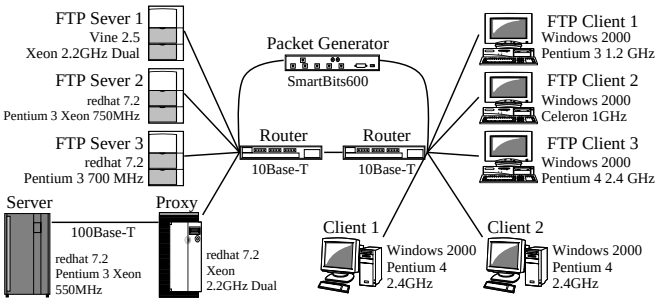


Fig. 6 Configuration of experimental system

Figures 7, 8, and 9 illustrate variations in reception rates observed at each client with `tcpdump`. As Fig. 8 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 as shown in Fig. 7, TCP session are disturbed and, the attained throughput is only 37 kbps. As a result, the friendliness is 1.44 in our system and 23.1 in traditional system, where the friendliness is given by dividing the average throughput of video sessions by that of TCP. 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. 8, 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 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

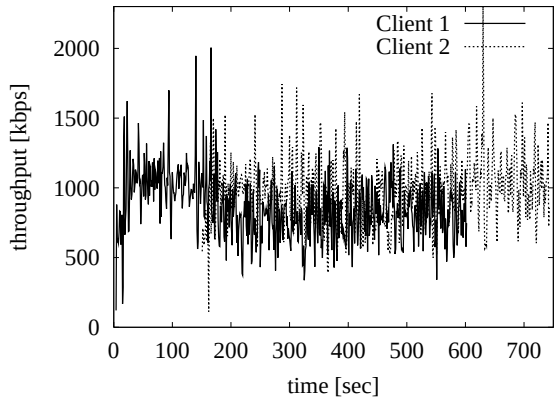


Fig. 7 Reception rate variation with traditional method

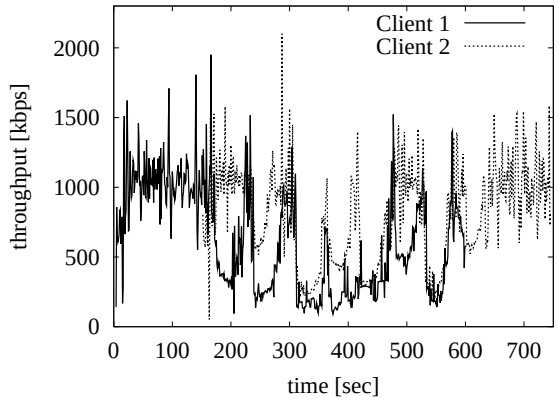


Fig. 8 Reception rate variation with quality adjustment

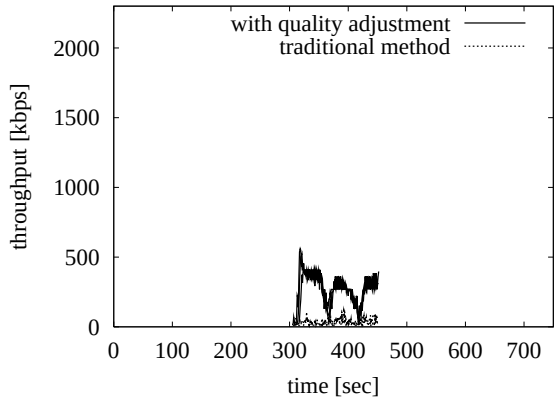


Fig. 9 Reception rate variation on FTP sessions

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. 8 and 9. We plan to conduct experiments within a larger network environment where a large number of sessions co-exist.

Figures 10 and 11 show RTT and packet loss probability calculated from information in RTCP Receiver Report messages. In the traditional system, the proxy

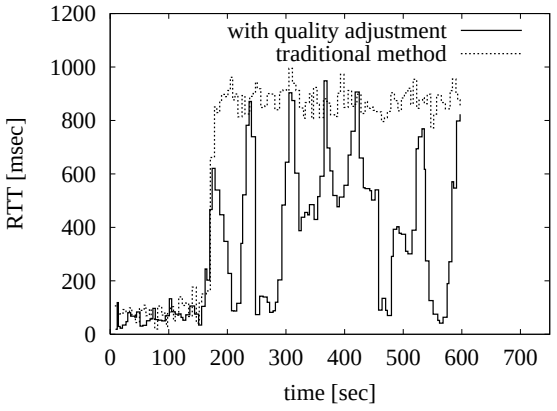


Fig. 10 RTT variations at client 1

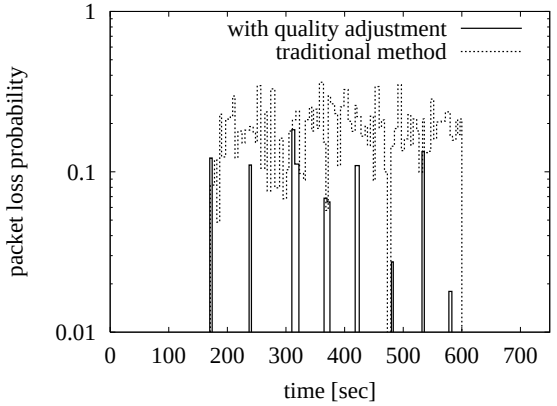


Fig. 11 Packet loss probability variations at client 1

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, 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.

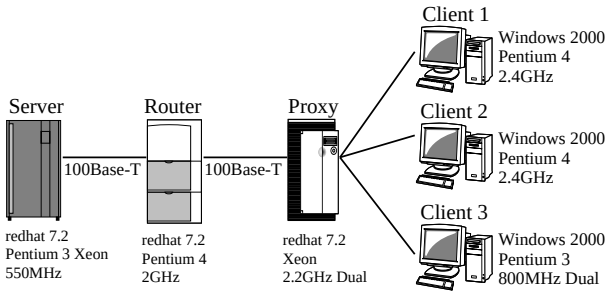


Fig. 12 Configuration of experimental system

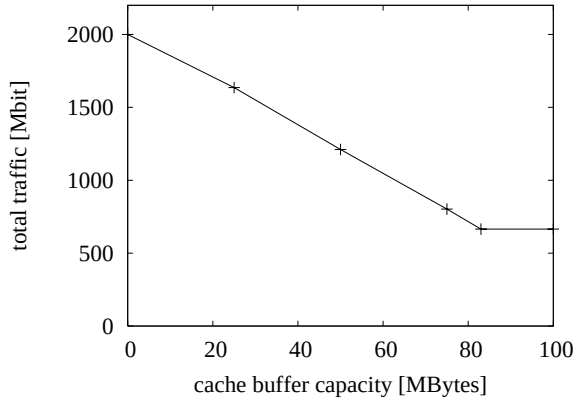


Fig. 13 Total traffic between the server and the proxy (P=0)

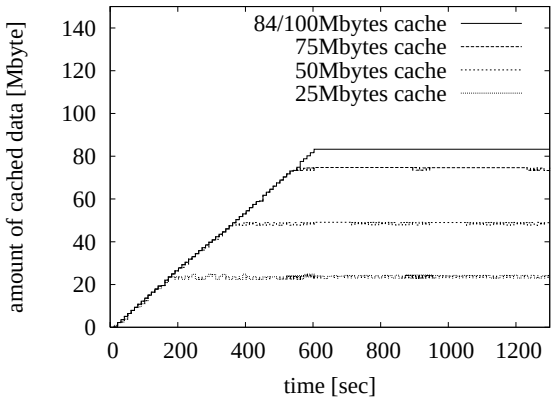


Fig. 14 Amount of cached data (P=0)

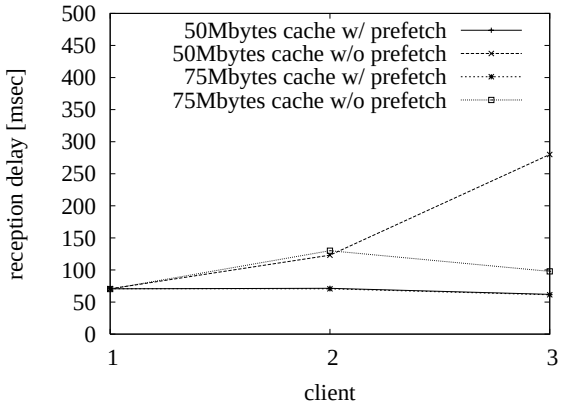


Fig. 15 Reception delay

3.2 Evaluation of Caching Mechanisms

Figure 12 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`, 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 13 shows the total amount of traffic between the video server and the proxy during experiments, and Fig. 14 shows the amount of cached data. Prefetching window size P is set to zero, i.e., no prefetching. In Fig. 13, 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. 14 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 $d_{rec}(j)$ as follows,

$$d_{rec}(j) = \frac{1}{BN}\sum_{i=1}^{BN}(T_{recv}(i,j) - T_{first}(j) - T_{stamp}(i)), \quad (2)$$

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 15 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.

In this experiment, since we consider a small and underloaded network, the reception delay is small enough without the capability of prefetching. We expect that the delay exceeds the initial buffering of three-seconds video data. By introducing the prefetching mechanism and a larger value of P, user becomes free from annoying freezes.

4. Conclusion and Future Work

In this paper, we proposed, designed, implemented, and evaluated a proxy caching system for MPEG-4 video streaming services. We employed off-the-shelf and common applications for the server program and the client program to verify the practicality of our proposed system. Through experiments, it was shown that our proxy caching system could dynamically adjust the quality of video streams to prevailing network conditions while providing users with a continuous and high-quality video streaming service. Furthermore, for the limited cache buffer, our proxy caching system could reduce the traffic between a video server and a proxy.

In future research works, 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] J. Liu and J. Xu, "Proxy caching for media steaming over the internet," *IEEE Communication Magazine*, pp. 88–94, Aug. 2004.

[2] R. Rejaie and J. Kangasharju, "Mocha: A quality adaptive multimedia proxy cache for internet streaming," in *Proceedings of NOSSDAV 2001*, 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," vol. 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*, vol. 6, pp. 375–386, Apr. 2004.

[7] P. Schojer, L. Böszörmenyi, H. Hellwagner, B. Penz, and S. Podlipnig, "Architecture of a quality based intelligent proxy (QBIX) for mpeg-4 videos," in *Proceedings of WWW2003*, pp. 394–402, May 2003.

[8] M. Zink, J. Schmitt, and C. Griwodz, "Layer-enabled video streaming: A proxy's perspective," *IEEE Communication Magazine*, pp. 96–103, Aug. 2004.

[9] 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.

[10] M. Handley, S. Floyed, J. Padhye, and J. Widmer, "Tcp friendly rate control (TFRC): Protocol specification," *Internet Request for Comments 3448*, Jan. 2003.

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

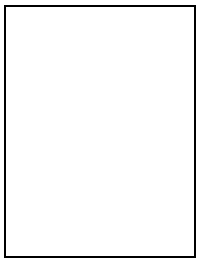
[15] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata, "RTP payload format for MPEG-4 audio/visual streams," *Internet Request for Comments 3016*, Nov. 2000.

[16] 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.

[17] H. Akamine, K. Nakada, N. Wakamiya, M. Murata, and H. Miyahara, "Implementation and evaluation of video filtering mechanisms for real-time multicast," in *Technical Report of IEICE (NS2001-50)*, pp. 13–18, June 2001.

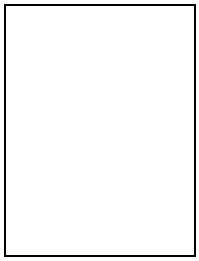
[18] T. Yamada, N. Wakamiya, M. Murata, and H. Miyahara, "Implementation and evaluation of video-quality adjustment for heterogeneous video multicast," in *Proceedings of The 8th Asia-Pacific Conference on Communications (APCC2002)*, pp. 454–457, Sept. 2002.

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

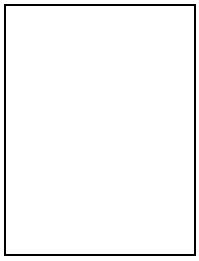


Yoshiaki TANIGUCHI received the B.E. and M.E. degrees in Informatics and Mathmatical Science from Osaka University, Japan, in 2002 and 2004, respectively. He is currently a doctoral course student at the Graduate School of Information Science and Technology, Osaka University, Japan. His research work is in

the area of QoS architecture for real-time and interactive video distribution system. He is a member of IEICE.



Naoki WAKAMIYA received M.E. and Ph.D. from Osaka University in 1994 and 1996, respectively. He was a research associate of graduate school of engineering science, Osaka University from April 1996 to March 1997, a research associate of Educational Center for Information Processing from April 1997 to March 1999, an assistant professor of Graduate School of Engineering Science from April 1999 to March 2002. Since April 2002, He is an associate professor of Graduate School of Information Science and Technology, Osaka University. His research interests include QoS architecture for distributed multimedia communications. He is a member of IEICE, IPSJ, ACM, and IEEE.



Masayuki MURATA received the M.E. and D.E. degrees in Information and Computer Sciences from Osaka University, Japan, in 1984 and 1988, respectively. In April 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. From 1992 to 1999, he was an Associate Professor in the Graduate School of Engineering Science, Osaka University, and from April 1999, he has been a Professor of Osaka University. He moved to Graduate School of Information Science and Technology, Osaka University in April 2004. He has more than three hundred papers of international and domestic journals and conferences. His research interests include computer communication networks, performance modeling and evaluation. He is a member of IEEE, ACM, The Internet Society, IEICE and IPSJ.