Manuscript received May 16, 2007; revised Aug. 18, 2007

A Proxy Caching System for MPEG-4 Video Streaming with a Quality Adaptation Mechanism

YOSHIAKI TANIGUCHI NAOKI WAKAMIYA MASAYUKI MURATA
Graduate School of Information Science and Technology
Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871
JAPAN
y-tanigu@ist.osaka-u.ac.jp http://www.anarg.jp/~y-tanigu/

Abstract: - In this paper, we propose, design, implement, and evaluate a proxy caching system for MPEG-4 video streaming services for heterogeneous users. 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 one. It also prefetches video blocks that are expected to be required in the near future. In addition, the proxy server adjusts the quality of block appropriately, because clients are heterogeneous, in terms of the available bandwidth, end-system performance, and so on. Through evaluations, we proved that our proxy caching system could provide users with a continuous and high-quality video streaming service in a heterogeneous environment.

Key-Words: - video streaming service, proxy caching, quality adaptation, MPEG-4, prototype, evaluation

1 Introduction

With the increase in computing power and the proliferation of the Internet, video streaming services have become widely deployed. Through these services, a 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. 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, which have diverse requirements on a received video stream for their heterogeneity.

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 deposits 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 [6-9]. There have been proposals for 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 widely available.

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 using off-the-shelf servers and clients. On the contrary to other proposals, our system does not need any modifications in server/client applications or systems as far as they conform to streaming standards. 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 requesting clients with blocks in time. It maintains the cache with a limited capacity by replacing unnecessary cached blocks with a newly retrieved block. A

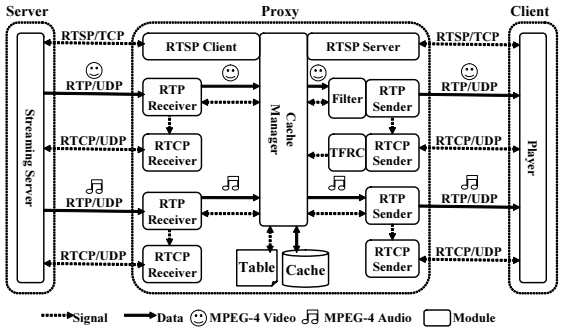


Fig. 1: Modules constituting system

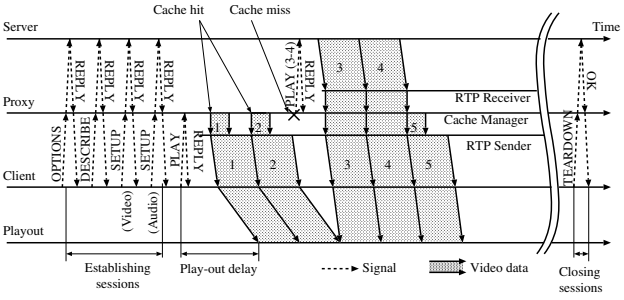


Fig. 2: Basic behavior of our proxy

proxy cache server prefetches video blocks that are expected to be required in the near future to avoid cache misses. It also adjusts the quality of a cached or retrieved video block to the appropriate level through video filters to handle client-to-client heterogeneity without involving the original video server.

We built a prototype of 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 and client programs to implement our system. Through evaluations from several performance points of view, we proved that our proxy caching system could dynamically adjust the quality of video streams to fit to 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 work.

2 Proxy Caching System with Video Quality Adaptation

Figure 1 illustrates modules that constitute our proxy cache server. Video streaming is controlled through RTSP/TCP sessions. There are two sets of sessions for each client. The first is established between an originating video server and a proxy to retrieve uncached blocks. The other is between the proxy and the client. Each of video and audio stream is transferred over a dedicated RTP/UDP session. The condition of streaming is monitored over RTCP/UDP sessions. A proxy server has a cache to deposit video data and a filter to adapt the quality of video to the TCP-friendly rate [10]. A video stream is coded using an MPEG-4 video coding algorithm, and it is compliant with ISMA 1.0 [11]. In this paper, we employed the *Darwin Streaming Server* [12] as a server application, and *RealOne Player* [13] and *QuickTime Player* [14] as client applications. However, other server or client applications being compliant with standard [11] with no or small modification. If other coding algorithm, e.g., MPEG-2, is used, the filtering module is only needed to be changed, which adapts the video rate by manipulating the video data, as far as server and client applications employ a set of the standard protocols, i.e., RTP/UDP, RTCP/UDP, and RTSP/TCP.

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 overhead introduced in 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. The proxy server relays these RTSP 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 sends the packet to the client on time. The quality of video blocks is adapted to fit the bandwidth on the path to the client by the *Filter*. Our proxy cache server adjusts the video rate to the bandwidth estimated by the TFRC module to share the bandwidth among video sessions and conventional data sessions in a friendly and fair manner.

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. Block identifiers are indicated beside the PLAY message in Fig. 2 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, a 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. Since we adopt an off-the-shelf application for the video streaming server, it cannot adjust the quality of video blocks. 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.

2.3 Rate Control with TFRC

TFRC enables a non-TCP session to behave in a TCP-friendly manner. The TFRC sender estimates the throughput of a TCP session sharing the same path using following equation.

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

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 acknowledgment.

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 an 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.

To derive the TCP-friendly rate, the TFRC requires a client to send feedback messages at least once per RTT. It means that a client application has to issue RTCP Receiver Report messages at least once per RTT. According to the RTCP specifications, a sender can trigger feedback by sending an RTSP Sender Report to a receiver. However, widely available client applications such as used in the experiments in this paper ignore this and issue RTCP Receiver Report messages every three to six seconds by their own timing. To verify the practicality and applicability of our proxy cache system, we used the client applications as they are, without any modification. As a result, we observed large variation in the reception rate as will be shown in section 3.1. Problems inherent in public applications remains for future research.

2.4 Video Quality Adaptation

We employed a frame dropping filter as a quality adaptation mechanism. The frame dropping filter adapts 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. 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, we

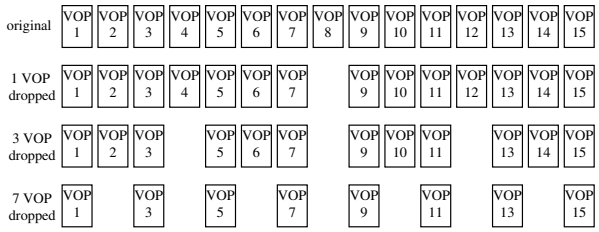


Fig. 3: Video structure after frame dropping

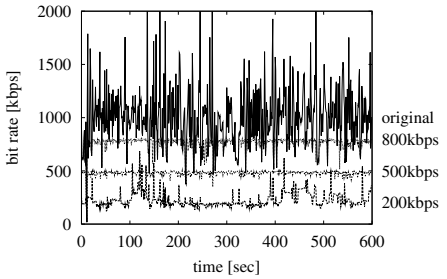


Fig. 4: Adapted video rate by frame dropping filter

cannot do packet-by-packet or VOP-by-VOP adaptation. Therefore, in our proxy cache server, 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 keep the smoothness of video play-out preferably, we propose an algorithm to decide the frame dropping order. We first prepare a binary tree of VOPs and discard frames in a well-balanced manner. 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 shows the resulting sequences of VOPs after frame dropping. As shown Fig. 3, this algorithm makes the number of VOPs among groups divided by gaps the same to have smooth video play-out. However, 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 discarding all B-VOPs is insufficient 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 4 shows bit rate variation of filtered video streams generated aiming at 200, 500, 800 kbps from

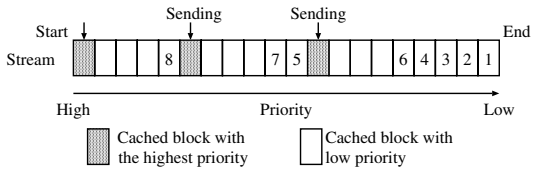


Fig. 5: Priority of cached blocks

the original VBR video stream whose average rate is 1000 kbps. Although our filtering algorithm is simple, resultant video rates are close to target rates with small fluctuations. We can replace ours with any other better algorithm as far as an off-the-shelf client application can decode a manipulated video stream.

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.

2.6 Cache Replacement

With our cache replacement algorithm, first, the *Cache Manager* selects a video stream with the lowest priority from cached streams using an LRU algorithm. It then assigns priorities 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. Among 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.

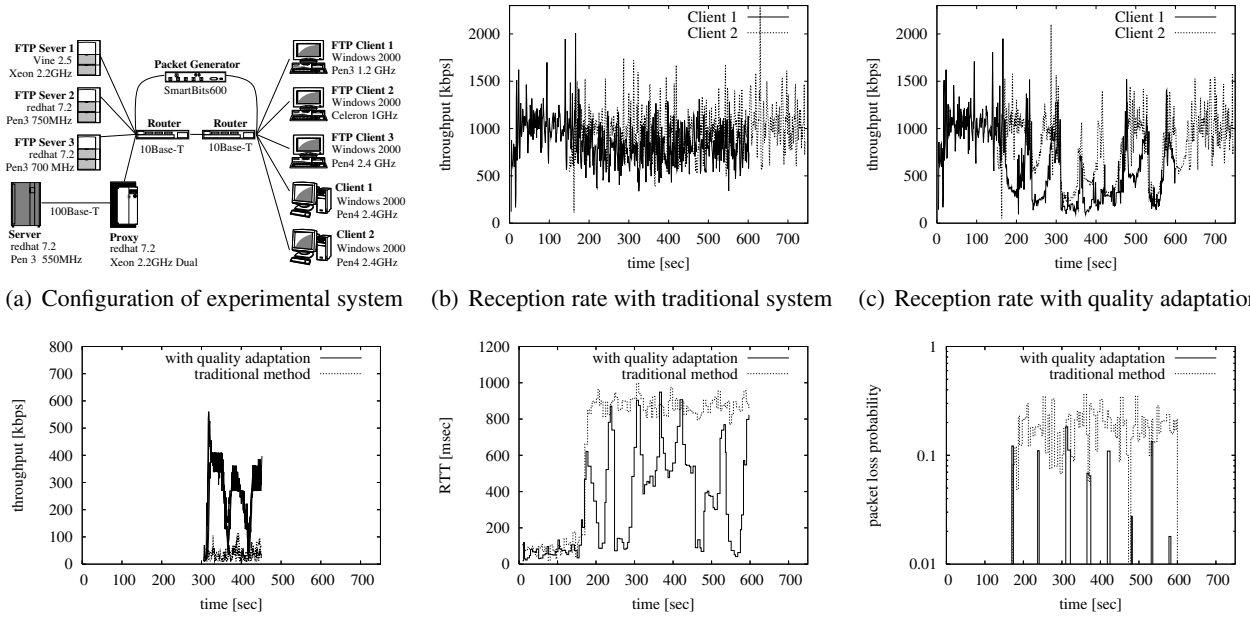
3 Experimental Evaluation

In this section, we show results of experiments on a prototype. In the experiments, we use 10 and 30 minutes long video streams by coding animation, scenery, action, fantasy, computer graphic, and sports movies using 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 an MPEG-4 stream. The maximum and minimum bit rate are about 2 Mbps and 400 Kbps, respectively. An example of rate variation is illustrated in Fig. 4 as "original". The size of the video stream is about 84 Mbytes for 10 minute stream and 248 Mbytes for 30 minute stream, respectively. A block corresponds to 300 VOPs, i.e., 10 seconds. Thus, the stream consists of 60 or 180 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.

There have been proposals for proxy caching mechanisms for video streaming service [1-9]. 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 widely available. There is no suitable work or implementation to compare with our system. Therefore, we do not show comparison results with other research work in following sections.

3.1 Evaluation of Rate Control with Video Quality Adaptation

Figure 6 (a) illustrates the configuration for our experimental system. A proxy is directly connected to a video server. Two video clients are connected to the proxy through two routers. 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. Video client 1 issues an OPTIONS message at time zero, and 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



(a) Configuration of experimental system (b) Reception rate with traditional system (c) Reception rate with quality adaptation

(d) Reception rate on FTP sessions (e) RTT at client 1 (f) Packet loss probability at client 1

Fig. 6: Evaluation of rate control

rate of 8 Mbps. For purposes of comparison, we also conducted experiments using a proxy without the capability of quality adaptation, which is called the traditional system hereafter.

Figures 6 (b), (c), and (d) illustrate variations in reception rates observed at each client with *tcpdump*. As Fig. 6 (c) 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. 6 (b), 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.

However, as observed in Fig. 6 (c), there are large rate variations in video sessions. The average throughput of video sessions during the competitive period is higher than that of TCP sessions. 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 intervals. The major reason for this is that the control interval of adaptation is three to six seconds due to the problem of the client application as described in section 2.3. The interval is considerably longer than that of TCP, which reacts to network conditions in an order of RTT. By increasing the frequency that a client reports feedback information by modifying the client applications, 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. 6 (c) and (d).

Figures 6 (e) and (f) show RTT and packet loss probability calculated from information in RTCP Receiver Report messages. In the traditional system, the proxy persists in sending video data at the coding rate during congestion, and many packets are delayed or lost at routers. The packet delay may cause freezes at play-out due to underflow of play-out buffer. Furthermore, the client application abandons playing out a VOP which is seriously damaged by a packet loss. The influnce of a packet loss propagates to the other VOPs when I-VOP or P-VOP is damaged. During the experiment, 9712 VOPs were played out with our system, but only 9133 VOPs were 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 re-

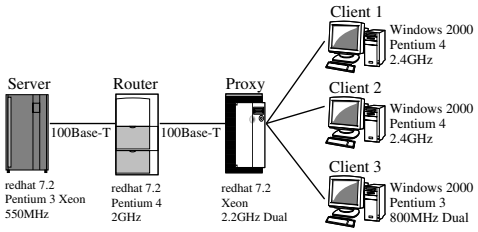


Fig. 7: Configuration of experimental system to evaluate caching mechanisms

ceived video data in the traditional system is larger than that in our system.

3.2 Evaluation of Caching Mechanisms

Figure 7 illustrates the configuration for our experimental system to evaluate our 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* [16], 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 message at time 0, 350, and 700, respectively. Three clients watch the same video stream. The proxy has no block at first. We do not introduce rate control with quality adaptation at the proxy. For purposes of comparisons, we also conducted experimental evaluations of cases where the proxy has no cache buffer, that is, when clients always received video blocks from the server.

Figure 8 (a) shows the total amount of traffic between the video server and the proxy during experiments, and Fig. 8 (b) shows the amount of cached data. Prefetching window size P is set to zero, i.e., no prefetching. 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 of buffer capacity as Fig. 8 (b) 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 of client j, d_j, as follows,

$$d_j = \frac{1}{N}\sum_{i=1}^{N}(T_j(i) - T_j(1) - i/F), \qquad (2)$$

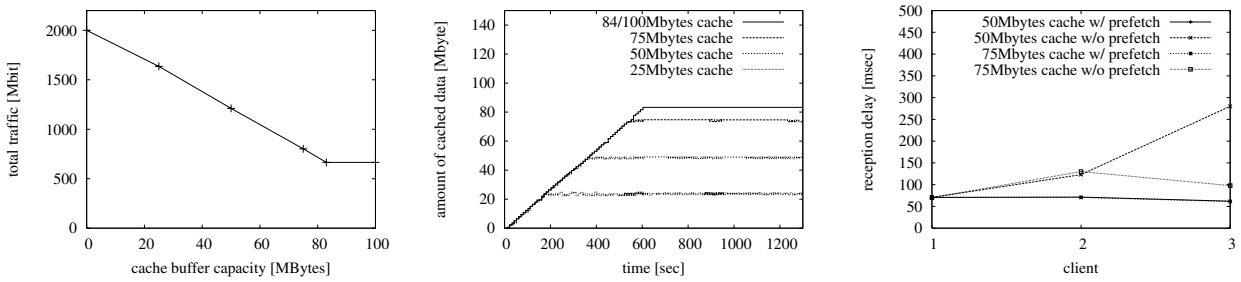
where, N corresponds to the number of VOPs in a stream, and F corresponds to the frame rate. $T_j(i)$ is the time that client j receives the VOP i. Thus, the reception delay d_j is the sum of differences between the expected arrival time of a VOP and the actual arrival time at a client. Figure 8 (c) shows the average of reception delay during a 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, the 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 the 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 in a larger network. However, by introducing the prefetching mechanism and a larger value of P, user becomes free from annoying freezes.

3.3 Evaluation with Many Clients

Finally, we conducted experiments on a system with many clients. In our experimental configuration, a proxy with 200 MBytes cache is directly connected with a video server and five clients. 10 client applications are running on each client and request the same video stream of 30 minutes. The client applications issue an OPTIONS message at random and begin watching the movie. The proxy has no block at first. The load on the proxy in terms of the memory and CPU usage is measured by using *vmstat* every one second, while increasing the number of client applications. Although not shown in figures, we verified the smoothness of video play-out on a monitor for a case of many clients.

Figure 9 shows the average memory usage and av-



(a) Total traffic between server and proxy (b) Amount of cached data (c) Reception delay

Fig. 8: Evaluation of caching mechanisms

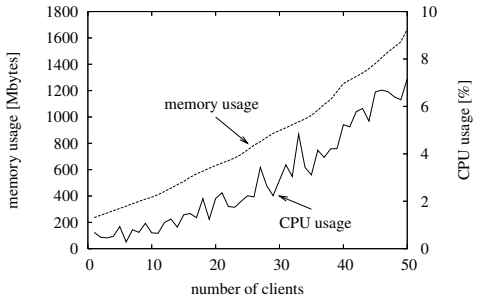


Fig. 9: Average memory usage and average CPU usage for the number of clients

erage CPU usage for the number of clients. As shown in Fig. 9, the memory usage is almost in proportional to the number of clients, because a process of proxy caching modules shown in Fig. 1 is invoked for each client on the proxy cache server. The CPU usage also linearly increases with the number of clients, but it is less than 8% for 50 clients.

In conclusion, although we confirmed that our system can offer heterogeneous services to more than 50 clients, we also observed that the load on the proxy cache server is proportional to the number of clients in the current implementation. By using the latest equipment and applications and optimizing of the system, we can expect that a proxy cache server can accommodate more clients. However, to have the higher scalability, we need to improve our system, whereas we adopted rather general approaches and mechanisms in the current implementation.

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 adapt the quality of video streams to 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. We also verified that our system could provide 50 clients with smooth video streaming.

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 [17].

Acknowledgments

The authors would like to thank Mr. Atsushi Ueoka and Mr. Fumio Noda of Systems Development Laboratory, Hitachi Ltd. for their help, encouragement and suggestions of this work.

References:

[1] Liu J. and Xu J. Proxy Caching for Media Steaming Over the Internet. *IEEE Communication Magazine*, Aug. 2004, pp. 88–94.

[2] Wang B., Sen S., Adler M., and Towsley D. Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution. *IEEE Transactions on Multimedia*, vol. 6(2), Apr. 2004, pp. 366–374.

[3] Song J. Segment-based Proxy Caching for Distributed Cooperative Media Content Servers. *ACM SIGOPS Operating Systems Review*, vol. 39(1), Jan. 2005, pp. 22–33.

[4] Guo L., Chen S., Xiao Z., and Zhang X. DISC: Dynamic Interleaved Segment Caching for Interactive Streaming Accesses. In *Proceedings of ICDCS 2005*, Jun. 2005.

[5] Ip A.T.S., Liu J., and Lui J.C.S. COPACC: An Architecture of Cooperative Proxy-Client Caching System for On-Demand Mediea Streaming. *IEEE Transaction on Parallell and Distributed Systems*, vol. 18(1), Jan. 2007, pp. 70–83.

[6] Shen B., Lee S.J., and Basu S. Caching Strategies in Transcoding-enabled Proxy Systems for Streaming Media Distribution Networks. *IEEE Transactions on Multimedia*, vol. 6(2), Apr. 2004, pp. 375–386.

[7] Zink M., Schmitt J., and Griwodz C. Layer-enabled Video Streaming: A Proxy's Perspective. *IEEE Communication Magazine*, Aug. 2004, pp. 96–103.

[8] Lin C.L., Lee H.H., Chan C.L., and Wang J.S. Cooperative Proxy Framework for Layered Video Streaming. In *Proceedings of GLOBECOM 2005*, vol. 1, Nov. 2005.

[9] He C.H. and Ko R.S. A QoS-Aware Transcoding System Using Composite Multimedia Document and Component Merge Queue. In *Proceedings of the 10th WSEAS International Conference on Communications*, Jul. 2006.

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

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

[12] Darwin Streaming Server. Available at `http://developer.apple.com/darwin/`.

[13] RealOne Player. Available at `http://www.real.com/`.

[14] QuickTime Player. Available at `http://www.apple.com/quicktime/`.

[15] Kikuchi Y., Nomura T., Fukunaga S., Matsui Y., and Kimata H. RTP Payload Format for MPEG-4 Audio/Visual Streams. *Internet Request for Comments 3016*, Nov. 2000.

[16] NIST Net. Available at `http://snad.ncsl.nist.gov/itg/nistnet/`.

[17] Psannis K.E. and Hadjinicolaou M.G. MPEG-based Interactive Video Streaming: A Review. In *Proceedings of the WSEAS Multiconference*, Oct. 2003.

Authors:

Yoshiaki Taniguchi received his BE and ME Degrees in Information and Computer Science from Osaka University in 2002 and 2004, respectively. He is currently a doctoral course student and a research assistant at Osaka University. His research interests include wireless sensor networks and video streaming systems. He is a member of IEICE and IEEE.

Naoki Wakamiya received his ME and DE Degrees from Osaka University, Japan, in 1994 and 1996 respectively. He was a Research Associate from 1996 and an Assistant Professor at the Graduate School of Engineering Science of Osaka University from 1999 to 2002. Since 2002, he has been an Associate Professor at the Graduate School of Information Science and Technology, Osaka University. His research interests include wireless sensor networks, mobile ad hoc networks, and overlay networks. He is a member of IEICE, IPSJ, ACM and IEEE.

Masayuki Murata received his ME and DE Degrees from Osaka University, Japan, in 1984 and 1988 respectively. In 1984, he joined the Tokyo Research Laboratory, IBM Japan, as a Researcher. He was an Assistant Professor from 1987 and an Associate Professor at Osaka University from 1992 to 1999. Since 1999, he has been a Professor at Osaka University, and he is now with Graduate School of Information Science and Technology, Osaka University. He has more than 500 papers in international and domestic journals and conferences. His research interests include computer communication networks, performance modeling and evaluation. He is a member of IEEE, ACM, and IEICE.