

Quality-Aware Cooperative Proxy Caching for Video Streaming Services

Yoshiaki Taniguchi, Naoki Wakamiya, Masayuki Murata
Graduate School of Information Science and Technology, Osaka University, Japan
Email: {y-tanigu, wakamiya, murata}@ist.osaka-u.ac.jp

Abstract—By applying a proxy mechanism widely used in WWW systems to video streaming systems, low-delay and high-quality video distribution can be accomplished without imposing extra load on the system. The video streaming system proposed in this paper consists of a video server and multiple proxy servers. In our mechanism, proxies communicate with each other and retrieve missing video data from an appropriate server by taking into account transfer delay and offerable quality. In addition, the quality of cached video data is adapted appropriately at a proxy to cope with the client-to-client heterogeneity in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality. Through simulation experiments, it is shown that our proposed mechanism can provide users with low-delay and high-quality video streaming services. Furthermore, to verify our mechanism, we implement a real system for MPEG-4 video streaming services and show that our proxy caching system can provide users with a continuous video distribution under dynamically changing network conditions.

Index Terms—video streaming service, cooperative proxy caching, quality adjustment, MPEG-4, implementation

I. INTRODUCTION

With the increase in computing power and the proliferation of the Internet, video streaming services have become widely deployed. However, since only best effort service is still predominantly used in the current Internet, video streaming services cannot provide clients with guaranteed continuous or reliable video streams. Furthermore, most of today's Internet streaming services lack scalability against the number of clients since they have been constructed using a client-server architecture, e.g. YouTube, Google Video, GyaO, and so on.

Proxy mechanisms widely used in WWW systems offer low-delay and scalable delivery of data by means of a "proxy server". A proxy server caches multimedia data that has passed through 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, low-delay and high-quality video distribution can be accomplished without imposing extra load on the system [1-7]. In addition, the quality of cached video data can be adapted appropriately at a proxy to support heterogeneous clients in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality [8-15]. Furthermore, coop-

eration among proxies can provide further effective and high quality video streaming services [16-18].

In our previous research work [19, 20], we proposed proxy caching mechanisms where 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 block to the requesting client in time. It maintains a cache with a limited capacity by replacing unnecessary cached blocks with a newly retrieved block. The proxy also prefetches video blocks that are expected to be required in the near future to avoid cache misses and adjusts the quality of a cached or retrieved video block to the appropriate level through video filters to handle client-to-client heterogeneity.

In this paper, to provide further effective and high quality video streaming service, we extend our previous work by considering the cooperation among proxies. The new mechanism consists of three parts: block provisioning, block prefetching, and cache replacement. The main benefits of our proxy cooperation mechanism include reducing the perceived network latency and achieving a higher degree of content availability by cooperative caching. Through simulation experiments, it is shown that our proposed mechanism can provide users with low-delay and high-quality video streaming services. In addition, to verify the practicality and adaptability of our proposal to existing video streaming services, we implement our cooperative proxy caching mechanism on a real system for MPEG-4 video streaming services extending our previous research work [20]. We employ off-the-shelf and common applications for server and client systems. Our implemented system is designed to conform to the standard protocols. Through experimental evaluations, it is shown that our proxy caching system can provide users with a continuous video distribution under dynamically changing network conditions.

The rest of this paper is organized as follows. In Section II, we show related work. Then, in Section III, we propose the cooperative proxy caching mechanism for video streaming services, and we evaluate our mechanism through simulation experiments in Section IV. In Section V, we describe the implementation of the proposed mechanism on a real system and conduct several experiments. Finally, we conclude this paper in Section VI.

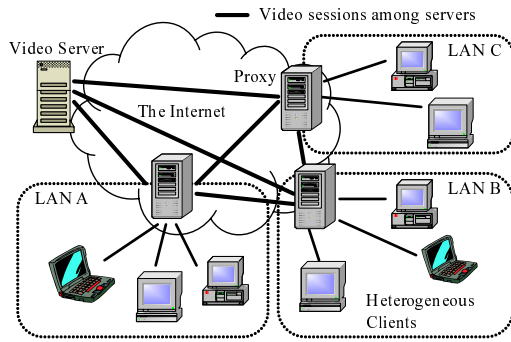


Figure 1. Cooperative video streaming system

II. RELATED WORK

By applying proxy mechanism to video streaming systems, low-delay and high-quality video distribution can be accomplished without imposing extra load on the system. There have been many proposals of proxy caching mechanisms for video streaming services. Typically, they divide a video stream into blocks to use the cache buffer and the bandwidth efficiently [4-7]. Division of a video stream into blocks and caching of initial block, i.e. *prefetch caching* [2], is efficient to allow the client to watch the video immediately. [3] considers an exponential division of a video stream into blocks to reduce the number of blocks. Blocks at the beginning are more important, and they are made smaller in the scheme. However, retrieving cost of postfix block or later blocks is not negligible if the video stream is popular. We use equal-sized blocks in this paper.

In addition, many papers consider quality adaptation of cached video block at a proxy to support heterogeneous clients in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality [11-15]. For example, [8, 15, 16] consider layered video coding algorithm. However, the number of layers is limited due to a coding algorithm employed and, as a result, the layered encoding based system lacks the scalability and adaptability to rate and quality variations.

Cooperation among proxies or clients can provide further effective and high quality video streaming services [16-18]. We also consider cooperation among proxies in this paper.

Most of research work use simulation-based evaluation, or assume specially designed server/client applications which are not widely available [19, 21]. On the other hand, in this paper, we build a prototype of our proxy caching system for MPEG-4 video streaming services on a real current system. We employ off-the-shelf and common applications for the server and client programs. Our system can be applied to any existing video streaming system as far as they conform to the specifications [24].

III. COOPERATIVE PROXY CACHING MECHANISM

In this section, we describe our cooperative caching mechanism.

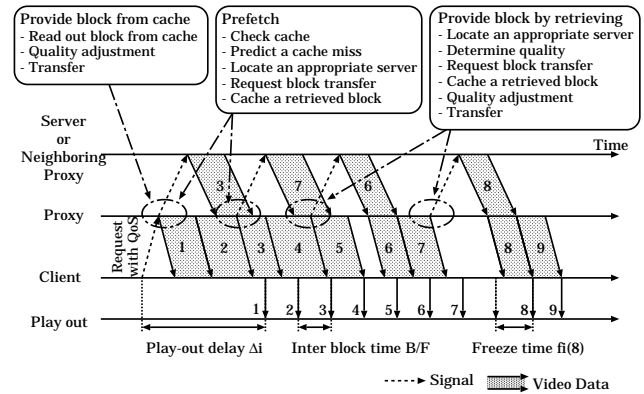


Figure 2. Basic behavior of our cooperative proxy caching system

A. Overview of the System

The video streaming system we consider in this paper is illustrated in Fig. 1. Our system consists of a single video server, cooperative proxies, and heterogeneous clients. Hereafter we refer to the video server and proxies as “servers”. Clients are heterogeneous in regard to their available bandwidth, propagation delays, end-system performance, and user preferences on the video quality.

A video stream is divided into L blocks, each consisting of B frames and assume a constant frame rate of F . A proxy retrieves video blocks from the distant video server or neighboring proxies on behalf of clients, deposits them in its local cache buffer, and adapts the quality of the blocks to the user’s demands. Video quality adjustment is performed by QoS filtering techniques such as frame dropping, low-pass, and re-quantization filters [22]. In addition, proxies communicate with each other and exchange blocks over a video session established among them, and maintain information of cached blocks at the other servers. We should note here that, to reduce the load on the network and servers, there is only a single session established between any pair of servers regardless of the number of clients.

Figure 2 illustrates how servers and client communicate with each other. The numbers in the figure correspond to block identifiers, which are in the order of their playout. The video streaming service is initiated by a request message issued by a client i to a designated proxy. The message contains information about preferable level of video quality Q_i (upper-constraint) and tolerable level of video quality q_i (lower-constraint) which are determined in accordance with the available bandwidth, end-system performance, and user preferences. A client is allowed to dynamically change QoS requirements during a video session to reflect changes of those constraints. A client which want to minimize quality variation may set same level for Q_i and q_i .

On receiving the first request message, the proxy begins to provide a requested video stream to the client in a block-by-block fashion. The proxy can 1) read out and send a cached block, 2) use a block being received, 3) wait for the preceding request for the same block to be

served, or 4) newly retrieve a block from another server. The proxy adopts the fastest way that can provide the client with a block of high level of quality among the four ways.

For example, if the proxy has a block of the same number and its quality can satisfy the request, the proxy may consider that using the cached block is the best way. It applies the video quality adjustment to the block as necessary and transfers it to the client over a video session established between them. If a block being received has satisfactory quality, the proxy may decide to send the block under transmission to the client. If there is a preceding request for the same block of higher quality, the proxy can wait for the request to be served. In some cases the proxy determines to retrieve the block. It first identifies an appropriate server among servers, then determines the quality of the block to retrieve, and sends a request message to the server. A retrieved block is cached and sent to the client after the video quality adjustment is applied as necessary. Details of block provisioning mechanism will appear in Section III-B. When the proxy finishes sending out the block, it moves to the next block and repeats the same procedure.

While providing clients with video blocks, the proxy predicts and prepares for a future potential cache miss by prefetching a block from other servers. A prefetching request is processed without disturbing normal block retrievals. Details of the prefetching mechanism will be given in Section III-C. In addition, if there is not enough space to store the newly retrieved block, a proxy replaces unnecessary cached block with a new block. Details of cache replacement mechanism will be shown in Section III-D.

To handle unexpected block transmission delays, client i first defers playing out a received video block for a predetermined waiting time Δ_i as shown in Fig. 2. Then, it begins to decode and display received blocks one after another, at regular intervals of B/F . In some cases such as a cache miss, a block does not arrive in time and the client is forced to pause. The time required for client i to wait until the arrival of block j is called *freeze time* and denoted as $f_i(j) \geq 0$ in this paper.

In the following sections, we describe the block provisioning mechanism, the block prefetching mechanism, and the cache replacement mechanism for the video streaming system to provide users with low-delay and high-quality services under a heterogeneous environment.

B. Block Provisioning

In providing a client with a block, a proxy chooses the best way among the four described in Section III-A in accordance with the offerable quality and the block transfer delay. For this purpose, servers communicate with each other and maintain the up-to-date information on other servers, such as the quality of offerable blocks, round-trip time, and available bandwidth in two tables. Information on locally cached blocks is maintained in the *cache table*, while the *remote table* is for information on

cached blocks at other servers. To predict the transfer time as accurately as possible, a proxy is assumed to be able to estimate the block size, propagation delay, and available bandwidth. Information related to the network condition among a proxy and its clients is also required.

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

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

where T_i indicates the instant when client i issues the first request message. δ_i is introduced to absorb unexpected delay jitters and estimation errors. $D_i(j-1) = \sum_{l=1}^{j-1} f_i(l)$ is the accumulated freeze time. A proxy estimates the four offerable quality of providing a block to a client as follows. After estimation, the proxy takes the best choice.

1) *Estimation of offerable quality in case of successful cache hit*: The first case is that the desired block j already exist in the cache buffer of proxy k . The offerable quality $c_{k,i}^{PC}(j)$ of block j to client i by using block j cached at the proxy k 's buffer is derived as

$$c_{k,i}^{PC}(j) = \min(q_k(j), \bar{c}_{k,i}^{PC}(j)), \quad (2)$$

where

$$\bar{c}_{k,i}^{PC}(j) = \max(q|t + d_{k,i}^{PC}(t) + \frac{a_j(q)}{r_{k,i}^{PC}(t)} \leq T_i(j)). \quad (3)$$

$q_k(j)$ stands for the quality of block j cached at the proxy k 's buffer. $a_j(q)$ is a function indicating the size of block j of quality q and is defined as 0, if $j = 0$ or $q = 0$. This function depends on the employed codec and we will provide an example later in Section IV. $d_{k,i}^{PC}(t)$ and $r_{k,i}^{PC}(t)$ are estimates of one-way propagation delay and available bandwidth from proxy k to client i at t , respectively. Thus, $d_{k,i}^{PC}(t) + \frac{a_j(q)}{r_{k,i}^{PC}(t)}$ provides the estimated time required for client i to receive the whole of block j of quality q via a video session whose available bandwidth is $r_{k,i}^{PC}(t)$ and propagation delay is $d_{k,i}^{PC}(t)$.

2) *Estimation of offerable quality in case of cache miss with block download in progress*: The second case is that the block j is not available in the cache at proxy k and it is currently being retrieved from another server s . The quality offerable is derived as

$$b_{k,i}^{PC}(j) = \max_{\forall s, s \neq k} (\min(q_{s,k}^{SP}, \bar{b}_{k,i}^{PC}(j))), \quad (4)$$

where

$$\begin{aligned} \bar{b}_{k,i}^{PC}(j) = & \max(q|t + d_{k,i}^{PC}(t) \\ & + \max(\frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}, \frac{a_j(q)}{r_{k,i}^{PC}(t)}) \leq T_i(j)). \end{aligned} \quad (5)$$

In the above equations, s indicates the server from which the proxy is receiving block j . $q_{s,k}^{SP}(t)$ is the quality

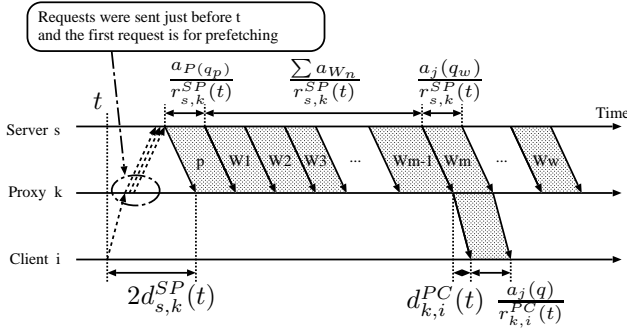


Figure 3. Worst-case delay due to waiting for the preceding request

of block being received at t . $a_{s,k}^{SP}(t)$ stands for the amount that has already been received. $d_{s,k}^{SP}(t)$ and $r_{s,k}^{SP}(t)$ correspond to estimates of one-way propagation delay and available bandwidth from server s to proxy k at t .

3) *Estimation of offerable quality in case of cache miss with waiting for pending block*: The third case is that the block j is not available in the cache at proxy k , but it already submitted a request for j to another server s . A proxy keeps track of requests that it sent out for block retrieval. $\mathcal{W}_{s,k}^{SP}(t) = \{W_1, W_2, \dots, W_w\}$ is a list of requests that had been sent from proxy k to server s before t . The quality $q_{s,k}^w(n)$ of block W_n is also maintained in a list. A pair of the block number and the quality is added to the lists when the proxy sends a request for block retrieval and is removed from the lists when the proxy begins receiving the block. The offerable quality $w_{k,i}^{PC}(j)$ for the m -th request, which is block j , i.e. $W_m = j$, can be estimated as

$$w_{k,i}^{PC}(j) = \max_{\forall s, s \neq k} (\min(q_{s,k}^w(m), \bar{w}_{k,i}^{PC}(j))), \quad (6)$$

where

$$\begin{aligned} \bar{w}_{k,i}^{PC} &= \max(q|t + 2d_{s,k}^{SP}(t) + d_{k,i}^{PC}(t) \\ &+ \frac{\sum_{n=1}^{m-1} a_{W_n}(q_{s,k}^w(n)) + a_{p_{s,k}(t)}(q_{s,k}^p(t))}{r_{s,k}^{SP}(t)} \\ &+ \max(\frac{a_j(q_{s,k}^w(m))}{r_{s,k}^{SP}(t)}) \leq T_i(j)). \end{aligned} \quad (7)$$

$p_{s,k}(t)$ indicates the block number to be prefetched from server s and $q_{s,k}^p(t)$ is its quality. If no prefetching request is waiting for server s , both $p_{s,k}(t)$ and $q_{s,k}^p(t)$ are zero. In our system, only one prefetching request is permitted by a server per proxy and prefetching is preempted by normal block retrievals. Details of the prefetching mechanism will be given in Section III-C. Equation (7) considers the worst case when no block is under transmission and the preceding requests from 1 to $m-1$ and the prefetching request will be served prior to block W_m , as illustrated in Fig. 3. If the proxy is receiving a block at t , $\bar{w}_{k,i}^{PC}$ is

derived using the following equations.

$$\begin{aligned} \bar{w}_{k,i}^{PC} &= \max(q|t \\ &+ \max(2d_{s,k}^{SP}(t), \frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}) \\ &+ d_{k,i}^{PC}(t) + \frac{S_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} \\ &+ \max(\frac{a_j(q_{s,k}^w(m))}{r_{s,k}^{SP}(t)}, \frac{a_j(q)}{r_{k,i}^{PC}(t)}) \leq T_i(j)), \end{aligned} \quad (8)$$

where

$$S_{s,k}^{SP}(t) = \begin{cases} \sum_{n=1}^{m-1} a_{W_n}(q_{s,k}^w(n)) + a_{p_{s,k}(t)}(q_{s,k}^p(t)), \\ \text{if } 2d_{s,k}^{SP}(t) > \frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}. \\ \sum_{n=1}^{m-1} a_{W_n}(q_{s,k}^w(n)), \text{ otherwise} \end{cases} \quad (9)$$

4) *Estimation of offerable quality in case of cache miss with new block request*: The final case is that the block j is not available in the cache at proxy k and a new request must be sent to another server s . The offerable quality $s_{k,i}^{PC}(j)$ is derived by the following equations when there is no block under transmission,

$$s_{k,i}^{PC} = \max_{\forall s, s \neq k} (\min(q_s^r(j), \bar{s}_{k,i}^{PC}(j))), \quad (10)$$

where

$$\begin{aligned} \bar{s}_{k,i}^{PC} &= \max(q|t + 2d_{s,k}^{SP}(t) + d_{k,i}^{PC}(t) \\ &+ \frac{\sum_{n=1}^w a_{W_n}(q_{s,k}^w(n)) + a_{p_{s,k}(t)}(q_{s,k}^p(t))}{r_{s,k}^{SP}(t)} \\ &+ \frac{a_j(q)}{\min(r_{s,k}^{SP}(t), r_{k,i}^{PC}(t))} \leq T_i(j)). \end{aligned} \quad (11)$$

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

$$\begin{aligned} \bar{s}_{k,i}^{PC} &= \max(q|t + \max(2d_{s,k}^{SP}(t), \frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}) \\ &+ d_{k,i}^{PC}(t) + \frac{U_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} \\ &+ \frac{a_j(q)}{\min(r_{s,k}^{SP}(t), r_{k,i}^{PC}(t))} \leq T_i(j)), \end{aligned} \quad (12)$$

where

$$U_{s,k}^{SP}(t) = \begin{cases} \sum_{n=1}^w a_{W_n}(q_{s,k}^w(n)) + a_{p_{s,k}(t)}(q_{s,k}^p(t)), \\ \text{if } 2d_{s,k}^{SP}(t) > \frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}. \\ \sum_{n=1}^w a_{W_n}(q_{s,k}^w(n)), \text{ otherwise} \end{cases} \quad (13)$$

5) *Selection of provisioning way among the four*: The fastest and best way is chosen among the four as far as the offerable quality is above $q_i(j)$ and below $Q_i(j)$. If none of $c_{k,i}^{PC}(j)$, $b_{k,i}^{PC}(j)$, $w_{k,i}^{PC}(j)$, and $s_{k,i}^{PC}(j)$ can satisfy request $q_i(j)$, the proxy chooses the fastest way.

C. Block Prefetching

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

Prefetch requests are treated at a server in a different way from requests for retrieving cache-missed blocks. The video server and proxies maintain a pair of prioritized queues per video session. The queue for usual block retrieval is given a higher priority and requests are handled in a first-come-first-served discipline. On the other hand, the queue for prefetching has a limited length of 1 and a waiting request is always overwritten by a new one. A prefetch request in the queue is served only when there is no request in the high-priority queue. A prefetch request is considered obsolete and removed when a server receives a request for the same block with higher quality.

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

$$t_{s,k}^p(t) = t + 2d_{s,k}^{SP}(t) + \frac{a_{p_{s,k}(t)}(q_{s,k}^p(t))}{r_{s,k}^{SP}}, \quad (14)$$

where $p_{s,k}(t)$ and $q_{s,k}^p(t)$ stand for the block number and the requested quality, respectively. This means that the proxy tries prefetching only when no preceding request is waiting to be served. If the derived time $t_{s,k}^p(t)$ is earlier than

$$T_i^p(p_{s,k}(t)) = T_i + \Delta_i + (p_{s,k}(t) - 1) \frac{B}{F} - \delta_i + D_i(j-1) - d_{k,i}^{PC}(t), \quad (15)$$

the proxy sends a request to server s .

Quality $q_{s,k}^p(t)$ is determined on the basis of the QoS requirement as $\beta Q_i(j)$ where $0 < \beta \leq 1$. If we set β to a small value, we can expect to prefetch a large number of blocks, but their quality becomes low. On the other hand, with a large β , there is little chance to successfully prefetch blocks in time, but a high-quality video stream can be provided with prefetched blocks. Information about a prefetching request is kept at a requesting proxy as a pair of block number $p_{s,k}(t)$ and quality $q_{s,k}^p(t)$ and is overwritten by a new prefetch and canceled when a block reception begins or a normal block retrieval is requested for the same block of higher quality.

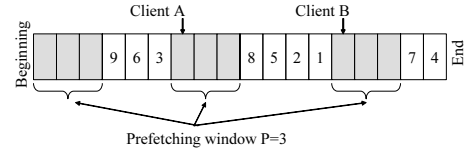


Figure 4. Sample order of block replacement

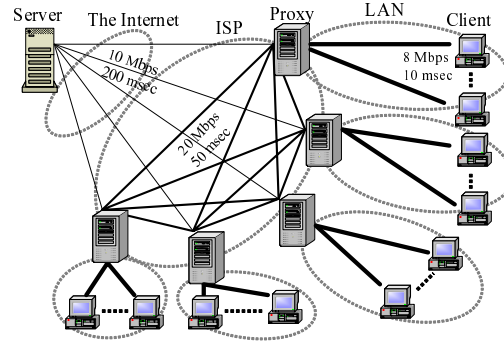


Figure 5. Configuration of simulation

D. Cache Replacement

When the available space of a cache buffer becomes insufficient to deposit a newly received block, a proxy first makes a list of blocks to be discarded. Those blocks which are located in prefetching windows are likely to be used earlier, and thus they will not be discarded. The first P blocks of a video stream are also considered important to suppress the initial delay. The rest of blocks are all candidates for replacement. The block m closest to the end of the longest run, i.e. a succession of non-prioritized blocks, becomes the first candidate. If the m -th block is cached, a proxy first tries degrading its quality to shrink the block size. The quality of the candidate should be larger than $\max_{1 \leq j \leq m-1} \max_{i \in S_{m-1}} Q_i(j)$ to prepare for future requests. Here, S_{m-1} is a set of clients which is watching any of blocks 1 through $m-1$. If the quality degradation is still insufficient, the proxy discards the candidate and moves to the next candidate at the end of the longest run. When all candidates, i.e. non-prioritized blocks, are dropped, but there is not enough room yet, the proxy gives up storing the new block. Figure 4 illustrates an example of an order of candidates.

There have been many proposals of cache replacement mechanisms. On the contrary of [2, 3], our mechanism set priority not only initial block but also blocks succeeding requested blocks for preparing future reference. Other cache replacement strategies may also be applied to our system with no or small modifications.

IV. SIMULATION EXPERIMENTS

In this section, we discuss the performance of our proposal with results we obtained through simulation experiments.

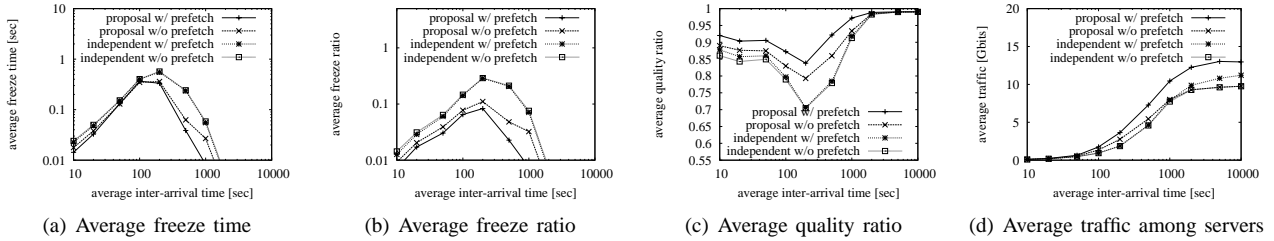


Figure 6. Simulation evaluations against average inter-arrival time τ

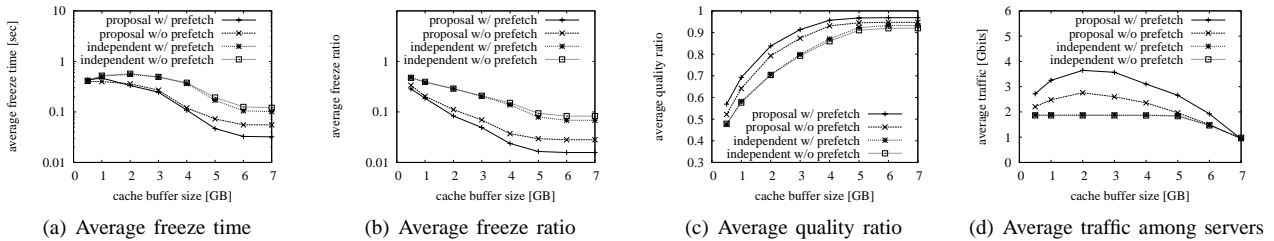


Figure 7. Simulation evaluations against cache buffer size

A. Simulation Configurations

The network we employed in the simulation is shown in Fig. 5. A video server is behind a wide-area network or the Internet. It communicates with five proxies through 10 Mbps sessions which are established over long-haul links with propagation delays 200 msec. Proxies are located at the boundary of an ISP network and are connected with each other via 20 Mbps sessions which are established over intra-network broadband links of 50 msec delay. A proxy establishes a 8 Mbps fixed-bandwidth session with each of its clients. The one-way propagation delay of each session is 10 msec. Although the available bandwidth and one-way delay fluctuate greatly under realistic conditions, we have employed static values so that we can clearly observe the basic behavior of our mechanism. A client starts requesting video blocks at randomly determined times. The inter-arrival time of the video requests issued by clients in each LAN follows an exponential distribution with average τ sec.

The video stream has a duration of 5400 sec and is played back at $F = 30$ fps. The stream is divided into 1 sec CBR blocks, that is, $L = 5400$ and $B = 30$. There are ten levels of quality from $q_{min} = 1$ to $q_{max} = 10$, and they are mapped to block sizes from $a_{min} = 1$ to $a_{max} = 10$ Mbits. The relationship between quality and block size is usually described as convex downward [23]. In this paper, we use $\forall j a_j(q) = \frac{a_{max}q_{min}}{q_{max}+q_{min}-q} \frac{B}{F}$ Mbits. Thus, the whole video stream amounts to 675 MB to 6.75 GB depending on the quality. A proxy can adjust quality of cached or retrieved blocks to lower quality. Each proxy is equipped with a limited cache buffer. Initially, all cache buffers are empty and only the video server has all blocks of the highest quality. The prefetching window size P is set at 10 blocks.

On client side, the buffering times and time to absorb delay jitters and prediction errors are set at $\Delta_i = 4$ and

$\delta_i = 2$ sec, respectively. The tolerable video quality $q_i(j)$ is fixed at 1. However, the preferable quality $Q_i(j)$ varies from 1 to 10, is initially set to 5, i.e. $Q_i(1) = 5$. Then, a client randomly determines the quality requirement on a block-by-block basis. In our experiments, the quality level $Q_i(j)$ is increased or decreased with a probability of 5% after each block is sent. This is introduced as a way of imitating the dynamic changes in quality requirements according to system conditions and the user preferences. The ratio of the quality of prefetched blocks to that of requests is determined as $\beta = 1.0$.

For comparison purposes, we conduct simulations of a system with four different schemes. One is referred as “independent w/o prefetch”. In this approach proxies always retrieve the missing or unsatisfactory block from the originating video server without prefetching. “independent w/ prefetch” corresponds to the case where independent proxies are coupled with the prefetching mechanism. The schemes “proposal w/o prefetch” and “proposal w/ prefetch” indicate the corresponding cases where the proxies cooperate.

We compare the performance in terms of the average freeze time, the average freeze ratio, the average quality ratio, and the average traffic among servers per user. The average freeze time is derived as $\sum_{i=1}^n \sum_{j=1}^L f_i(j) / Ln$ where n is the number of clients. The average freeze ratio is defined as the ratio of number of freezes to the number of all blocks L per user. The quality ratio is defined as the ratio of quality of provided block to the requested preferable quality $Q_i(j)$. Simulations finish after 100τ sec, i.e. approximately 500 clients arrive in the system. All results are averaged over 5 simulation runs.

B. Simulation Results

First, we evaluate the effects of the inter-arrival time τ . The cache buffer size is fixed at 2 GB. Figure 6

shows the average freeze time, the average freeze ratio, the average quality ratio, and average amount of traffic among servers against the average inter-arrival time of clients. As shown in Fig. 6, our proposed mechanism can provide users with video distribution of lower delay and fewer freezes, achieving the same or higher quality ratio at the cost of a slight increase in traffic due to inter-proxy communication. Furthermore, the prefetching mechanism contributes to achieving low-delay and high-quality video distribution.

For all schemes, if the inter-arrival time is less than 200 sec, the average freeze time and the average freeze ratio increase with the inter-arrival time and the quality ratio decreases. This is because shorter inter-arrival times can achieve a higher probability of utilizing cached blocks and lower probability of retrieving new blocks from other servers. On the other hand, if the inter-arrival time is larger than 200 sec, as the inter-arrival time increases, the average freeze time and the average freeze ratio decrease and the quality ratio increases. This is because a proxy can use more bandwidth for each client, if the number of clients is small as shown in Fig. 6 (d).

Next, we evaluate the effects of the cache buffer size. The average inter-arrival time τ is fixed at 200 sec. Figure 7 shows the average freeze time, the average freeze ratio, the average quality ratio, and the average amount of traffic among servers against cache buffer size. As shown in the figures, our proposed mechanism can provide users with video distribution of lower delay and fewer freezes, achieving higher quality ratio.

For all schemes, the average freeze time and the average freeze ratio decrease, the average quality ratio increases, and the average traffic among servers decreases with the cache buffer size as shown in Fig. 7. This is because a larger cache buffer size can achieve higher probability of cache hit and that of finding blocks in other proxies' cache buffers. However, in this experiment, increasing the cache buffer size more than 6 GB has no impact on the average freeze time, the average freeze ratio and the average quality ratio.

V. IMPLEMENTATION AND EVALUATION OF PROPOSED MECHANISM

In this section, we describe our implementation of proposed mechanism on a real system, and evaluate our mechanism through practical experiments. The system we implemented is based on our previous work [20] and video streaming is controlled through RTSP/TCP sessions. Each of the video and audio streams is transferred over a dedicated RTP/UDP session and the condition of streaming is monitored over RTCP/UDP sessions. A video stream is coded using MPEG-4, and it is compliant with ISMA 1.0 [24]. In this paper, we use the *Darwin Streaming Server* as a server application, and *RealOne Player* and *QuickTime Player* as client applications. However, other server or client applications being compliant with the standard can be incorporated with no or only small modification.

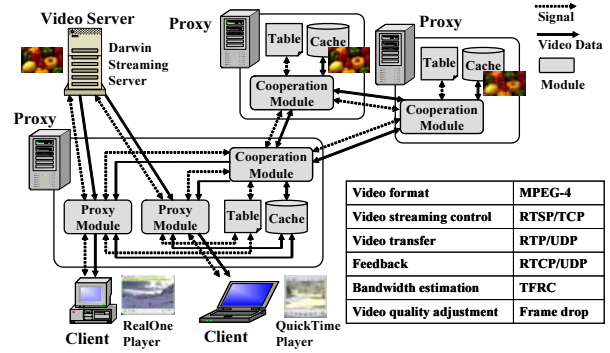


Figure 8. Overview of the implemented system

A. Overview of the Implemented System

Figure 8 illustrates the modules that constitute our video streaming system. Each dotted arrow and solid arrow corresponds to signal and data flow, respectively. A *Proxy Module* is generated for each client and provides a client with video blocks. A proxy has a *Cache* to deposit video data, and maintains the cache table and the remote table in the *Table*. We introduce a *Cooperation Module* for each proxy to communicate with neighboring proxies.

For its simplicity and speed, these modules have a frame dropping filter to adapt the quality of video. It adjusts the video quality to the desired level by discarding frames in a well-balanced way. To know the available bandwidths, i.e. r^{SP} and r^{PC} , among the server, the proxies, and clients, they have the capability to estimate TCP-friendly [25] rates. Proxies estimate the throughput of a TCP session sharing the same path using control information obtained by exchanging RTCP messages. One-way propagation delays, i.e. d^{SP} and d^{PC} , among the server, the proxies, and clients are also estimated by exchanging RTCP messages.

In our implemented system, each block corresponds to a sequence of VOPs (Video Object Planes) of an MPEG-4 stream. A block consists of a video block and an audio block, and they are separately stored in the *Cache*. We empirically use $B = 300$ VOPs as the block size in our implementation. Since our MPEG-4 video stream is coded at $F = 30$ frames per second, a single block corresponds to 10 sec. We use the video coding rate to indicate its quality, and the Range and Bandwidth field of an RTSP PLAY message to specify the block and its quality.

The basic behavior of our system is as follows. First, a client establishes connections for audio and video streams with a proxy by sending a series of RTSP OPTIONS, DESCRIBE, and SETUP messages. These RTSP messages are received by the *Proxy Module* and relayed to the video server. Thus, connections between the video server and the proxy are also established at this stage. On receiving a SETUP REPLY message, the client requests delivery of the video stream by sending an RTSP PLAY message. Here, since the used client applications cannot declare an acceptable range of video quality levels, they are considered ready to receive and perceive a video

stream at any quality, i.e. $q = 0$, $Q = \infty$.

The *Proxy Module* adopts the fastest way so that it can provide a client with a block of higher level of quality. When the *Proxy Module* provides a cached block, it reads it from *Cache* and sends it to the client. The quality of the video block is adjusted if necessary. When the *Proxy Module* retrieves a block from a neighboring proxy, it sends a request to the *Cooperation Module*. The *Cooperation Module* sends an RTSP PLAY message to the proxy, retrieves the block, and relays the block to the *Proxy Module*. When the reception is completed, the *Proxy Module* deposits the block in the *Cache*. If there is not enough room to store the newly retrieved block, the *Proxy Module* replaces the new block with less important blocks in the cache buffer. When the *Proxy Module* retrieves the block from the video server, it sends an RTSP PLAY message to the video server.

A client receives blocks from a proxy and first deposits them in the so-called play-out buffer. Then, it gradually reads blocks out from the buffer and plays them. When a proxy receives an RTSP TEARDOWN message from a client, the proxy relays the message to the video server, and closes the sessions.

B. Information Sharing Among Proxies

In order to maintain a remote table, a proxy issues an RTSP GET_PARAMETER message to other proxies when the end of the prefetching window of any client reaches an entry which is zero, i.e. an uncached block. An RTSP GET_PARAMETER message includes a list of blocks required in the near future. Blocks in the list are those which are currently requested by clients and its subsequent I blocks. These I blocks from the beginning of the stream are also listed in the message to prepare for new clients that will request the stream in the future. In addition, we also introduce a timer to force a refreshing of the remote table. The timer expires every $(I - P - 1)B/F$ and a proxy sends an RTSP GET_PARAMETER message to other proxies.

On receiving an RTSP GET_PARAMETER message, a proxy first examines its cache table about blocks listed in the message. It then returns an RTSP REPLY message which contains the list of pairs of a cached block and its quality.

C. Cooperative Proxy Caching

In order to run an implementation of our cooperative proxy caching mechanism as described in Section III on an actual system, we made the following small modifications.

1) *Block Provisioning*: A proxy adopts the fastest way that can provide a client with a block of higher quality in time. Since a server sends a video block frame-by-frame at the frame rate of a video stream in our implemented system, the value $a_j(q)/r_{s,k'}^{SP}(t)$ in the equations in Section III-B is replaced with B/F for the originating video server k' .

2) *Block Prefetching*: In our implemented system, a proxy sends an RTSP PLAY message with an additional field to prefetch a block from another proxy, that is the Prefetch field. Since the video server cannot process this new field, a proxy only sends a prefetching request to other proxies and not to the video server.

3) *Cache Replacement*: Since the client application does not declare its desired level of quality, a proxy only discards a cached block once it is chosen as a candidate.

D. Experimental Configuration

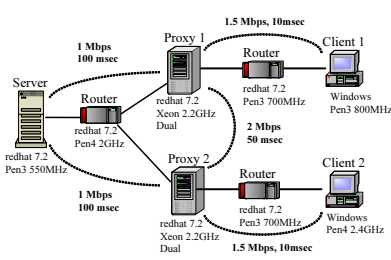
We already evaluated our whole caching mechanism in previous simulation section. Furthermore, the effectiveness of the caching mechanism such as block prefetching and cache replacement as well as the scalability against the number of clients in our implemented system have been already verified in the case of a single proxy in [20]. In this paper, we evaluate the block provisioning mechanism when proxies cooperate with each other, to evaluate feasibility and effectiveness of cooperation among proxies on a real current system.

Figure 9 (a) illustrates the configuration of our experimental system. For the sake of clarity, we limit the experimental settings to only two proxies and a video server connected through a router. There are two video clients in the system and each is connected to a neighboring proxy through a router. In order to control the delay and link capacity, *NISTNet* is used at the routers. The one-way propagation delay and link capacity are set as shown in Fig. 9 (a). In the experiments, we use an MPEG-4 video stream of 200 sec encoded at a rate of 1 Mbps and a frame rate of 30 fps. A block corresponds to 300 VOPs, i.e. 10 sec. Thus, the stream consists of $L = 20$ blocks, b_1, b_2, \dots, b_{20} . Initially, proxies already have some blocks as shown in Fig. 9 (b). The window of inquiry is set to $I = 5$.

The experimental scenario is as follows. Client 1 first issues an RTSP OPTIONS message at time 0, and client 2 issues it at 200 sec. Both clients watch the same video stream from the beginning to the end without interactions such as rewinding, pausing, and fast-forwarding. After 260 sec, the link capacity between proxy 1 and proxy 2 is reduced from 2 Mbps to 700 kbps. Using this configuration, we evaluate the capability of the block provisioning mechanism against changes in network conditions and cached blocks on the neighboring proxy. For this purpose, we do not consider other mechanisms such as block prefetching and cache replacement in this experiment. We set the cache buffer capacity to 30 MB, i.e. larger than the size of the whole video stream, and the prefetching window to $P = 0$.

E. Experimental Results

Figures 9 (c) and (d) illustrate variations in reception rates observed at proxy 1 and client 1 with *tcpdump*, respectively. First, proxy 1 provides client 1 with cached blocks b_1 to b_3 , since they are available fastest and have



(a) Configuration of experimental system

Cache table on proxy 1 ($t=0$)

block	bit rate [kbps]
b1-b3	1000
b4-b6	700
b7-b10	0
b11-b20	1000

Cache table on proxy 2 ($t=0$)

block	bit rate [kbps]
b1-b6	1000
b7-b20	0

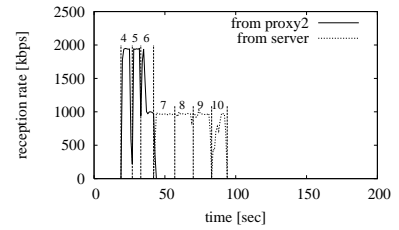
Cache table on proxy 1 ($t=200$)

block	bit rate [kbps]
b1-b20	1000

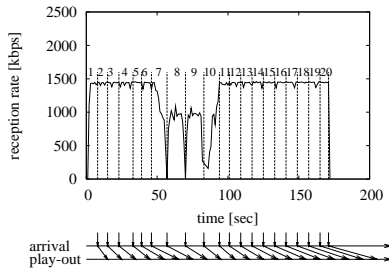
Cache table on proxy 2 ($t=400$)

block	bit rate [kbps]
b1-b19	1000
b20	276

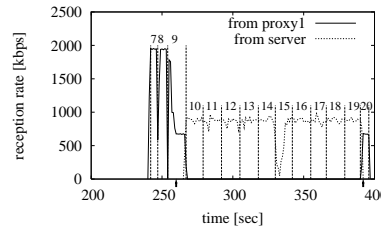
(b) Cache tables



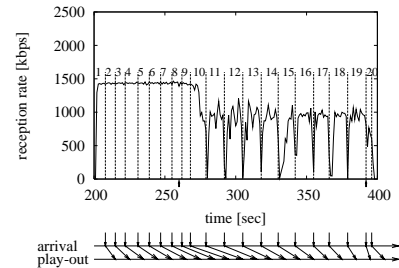
(c) Reception rate at proxy 1



(d) Reception rate at client 1



(e) Reception rate at proxy 2



(f) Reception rate at client 2

Figure 9. Experimental evaluation of block provisioning mechanism

the highest quality. While sending cached blocks, the proxy can afford to retrieve blocks of higher quality from proxy 2 for blocks b_4 to b_6 . Then, proxy 1 retrieves b_7 to b_{10} from the video server. For these 40 sec of blocks, it takes 50 sec of transmission time, because the link capacity between the video server and proxy 2 is smaller than the video rate. Furthermore, the video server sends additional VOPs beginning with the preceding I-VOP, if the specified range starts with a P or B-VOP. This increases the block size and introduces additional delay. However, owing to those cached blocks, proxy 1 has enough time to retrieve them and provide all blocks to client 1 for smooth playback.

In the bottom part of Fig. 9 (d), instants of block arrivals and those of play-out at client 1 are indicated. In these experiments, the client application first caches a received video block and defers its play-out by 3 sec. As Fig. 9 (d) illustrates, a user can watch the video without freezes. As a result of block retrieval, the cache table of proxy 1 changes as shown in Fig. 9 (b).

Figures 9 (e) and (f) illustrate variations in reception rates observed at proxy 2 and client 2, respectively. Proxy 2 first provides client 2 with cached blocks b_1 to b_6 . For uncached blocks b_7 to b_{20} , proxy 2 tries retrieving high-quality blocks from proxy 1. At 260 sec, the capacity of the link between proxy 1 and proxy 2 is reduced to 700 kbps. Consequently, proxy 2 contacts the video server from b_9 , since the video server can provide the highest quality blocks in the fastest way. However, delays are gradually introduced in retrieving blocks from the video server due to the insufficient link capacity.

At 392 sec, proxy 2 again contacts proxy 1 to retrieve the block b_{20} . Taking into account the time needed in block transmission from proxy 1 to proxy 2 and that to

client 2, the quality of block b_{20} to request to proxy 1 is intentionally reduced to 276 kbps. On receiving the request, proxy 1 applies the video quality adjustment to block b_{20} and transfers the modified block to proxy 2. Proxy 2 caches the block and provides it to client 2. As Fig. 9 (f) illustrates, all blocks are successfully provided to client 2 through the above mentioned control. Finally, the cache table of proxy 2 becomes as in Fig. 9 (b).

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

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed an effective video streaming mechanism where proxies cooperate with each other. Simulation results show that our proposed mechanism can provide users with low-delay and high-quality video streaming services in heterogeneous environments. In addition, we designed and implemented our proxy caching mechanism on a real system for an MPEG-4 video streaming service employing off-the-shelf and common applications. Through evaluations, it was shown that our proxy caching system can provide users with a continuous video streaming service under dynamically changing network conditions.

As future research work, it is necessary to conduct additional experiments, e.g., in a larger network environment, with other filtering mechanisms, and with other server and client applications. We also intend to take into account user interactions such as pauses, fast forwarding, and rewinding.

ACKNOWLEDGMENTS

The authors would like to thank Kenji Leibnitz and anonymous reviewers for their help, suggestions, and helpful comments for this work.

REFERENCES

- [1] J. Liu and J. Xu, "Proxy caching for media steaming over the Internet," *IEEE Communication Magazine*, pp. 88–94, Aug. 2004.
- [2] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proceedings of IEEE INFOCOM 1999*, Mar. 1999, pp. 1310–1319.
- [3] K. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of WWW 2001*, May 2001, pp. 36–44.
- [4] J. Song, "Segment-based proxy caching for distributed cooperative media content servers," *ACM SIGOPS Operating Systems Review*, vol. 39, no. 1, pp. 22–33, Jan. 2005.
- [5] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "DISC: dynamic interleaved segment caching for interactive streaming accesses," in *Proceedings of ICDCS 2005*, June 2005, pp. 763–772.
- [6] C.-L. Chan, S.-Y. Huang, and J.-S. Wang, "Performance analysis of proxy caching for VOD services with heterogeneous clients," *IEEE Transactions on Communications*, vol. 55, no. 11, pp. 2142–2151, Nov. 2007.
- [7] M. H. Kabir, G. C. Shoja, and E. G. Manning, "On-demand segmentation and proxy buffer provisioning for scalable and interactive video streaming scheme," in *Proceedings of the 8th IEEE Workshop on Multimedia Signal Processing*, 2006, pp. 65–70.
- [8] R. Rejaie and J. Kangasharju, "Mocha: a quality adaptive multimedia proxy cache for Internet streaming," in *Proceedings of NOSSDAV 2001*, June 2001.
- [9] K.-C. Chang and T.-F. Chen, "Efficient segment-based video transcoding proxy for mobile multimedia services," *Journal of Systems Architecture*, vol. 53, no. 11, Nov. 2007.
- [10] J. Liu, J. Xu, and X. Chu, "Fine-grained scalable video caching for heterogeneous clients," *IEEE Transactions on Multimedia*, vol. 8, no. 5, pp. 1011–1020, Oct. 2006.
- [11] W. hsiu Ma and D. H. C. Du, "Design a progressive video caching policy for video proxy servers," *IEEE Transactions on Multimedia*, vol. 6, no. 4, pp. 599–610, Aug. 2004.
- [12] Z. Miao and A. Ortega, "Scalable proxy caching of video under storage constraints," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1315–1327, Sept. 2002.
- [13] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 622–636, June 2002.
- [14] S. H. G. Chan and F. Tobagi, "Distributed servers architecture for networked video services," *IEEE/ACM Transactions on Networking*, vol. 9, no. 2, pp. 125–136, Apr. 2001.
- [15] M. G. Podolsky, S. McCanne, and M. Vetterli, "Soft ARQ for layered streaming media," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Techonology*, vol. 27, no. 1–2, pp. 81–97, Feb. 2002.
- [16] C.-L. Lin, H.-H. Lee, C.-L. Chan, and J.-S. Wang, "Cooperative proxy framework for layered video streaming," in *Proceedings of GLOBECOM 2005*, vol. 1, Nov. 2005.
- [17] A. T. S. Ip, J. Liu, and J. C.-S. Lui, "COPACC: An architecture of cooperative proxy-client caching system for on-demand media streaming," *IEEE Transaction on Parallel and Distributed Systems*, vol. 18, no. 1, pp. 70–83, Jan. 2007.
- [18] E. Kusmierek, Y. Dong, and D. H. C. Du, "Loopback: Exploiting collaborative caches for large-scale streaming," *IEEE Transactions on Multimedia*, vol. 8, no. 2, pp. 233–242, Apr. 2006.
- [19] 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*, vol. E86-B, no. 6, pp. 1849–1858, June 2003.
- [20] Y. Taniguchi, N. Wakamiya, and M. Murata, "A proxy caching system for MPEG-4 video streaming with a quality adaptation mechanism," *WSEAS Transactions on Communications*, vol. 6, no. 10, pp. 824–832, Oct. 2007.
- [21] H. Guo, G. Shen, Z. Wang, and S. Li, "Optimized streaming media proxy and its applications," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 265–281, Jan. 2007.
- [22] N. Yeadon, F. Gracia, D. Hutchinson, and D. Shepherd, "Filters: QoS support mechanisms for multipeer communications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1245–1262, Sept. 1996.
- [23] K. Fukuda, N. Wakamiya, M. Murata, and H. Miyahara, "QoS mapping between user's preference and bandwidth control for video transport," in *Proceedings of IWQoS'97*, May 1997, pp. 291–302.
- [24] Internet Streaming Media Alliance, "Internet streaming media alliance implementation specification version 1.0," Aug. 2001.
- [25] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," *Internet Request for Comments 3448*, Jan. 2003.

Yoshiaki Taniguchi received his B.E., M.E., and Ph.D. degrees from Osaka University, Japan, in 2002, 2004, and 2008, respectively. Since Oct. 2008, he has been an Assistant Professor at the Cybermedia Center, Osaka University. His research interests include wireless sensor networks, wireless mesh networks, and video streaming systems. He is a member of IEICE and IEEE.

Naoki Wakamiya received his M.E. and Ph.D. 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 M.E. and D.E. 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, where he is now with the Graduate School of Information Science and Technology. 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.