Scalable Proxy Caching of Video under Storage Constraints

Zhourong Miao *Student Member, IEEE*,

Antonio Ortega *Senior Member, IEEE*

Integrated Media Systems Center, Signal and Image Processing Institute,
Department of Electrical Engineering-Systems,
University of Southern California, Los Angeles, CA 90089.
{zmiao, ortega}@sipi.usc.edu

IEEE Journal on Selected Areas in Communications special issue on
"Internet Proxy Services"

Corresponding author: Zhourong Miao

Phone: 213-740-2318. Fax: 213-740-4651. Email: zmiao@sipi.usc.edu

Manuscript received April 2001; revised January 2002.

This work has been funded in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center and by the National Science Foundation under grant MIP-9804959.

Abstract

Proxy caching has been used to speed up web browsing and reduce networking costs. In this paper we study the extension of proxy caching techniques to streaming video applications. A trivial extension consists of storing complete video sequences in the cache. However this may not be applicable in situations where the video objects are very large and proxy cache space is limited. We will show that the approaches proposed in this paper (referred to as *selective caching*), where only a few frames are cached, can also contribute to significant improvements in the overall performance. In particular we will discuss two network environments for streaming video, namely, Quality-of-Service (QoS) networks and best-effort networks (Internet). For QoS networks, the video caching goal is to reduce the network bandwidth costs; for best-effort networks, the goal is to increase the robustness of continuous playback against poor network conditions (such as congestion, delay and loss). Two different selective caching algorithms (SCQ and SCB) are proposed, one for each network scenario, to increase the relevant overall performance metric in each cases, while requiring only a fraction of the video stream to be cached. The main contribution of our work is to provide algorithms that are efficient even when the buffer memory available at the client is limited. These algorithms are also scalable so that when changes in the environment occur it is possible, with low complexity, to modify the allocation of cache space to different video sequences.

Keywords: proxy, caching, selective caching, streaming video, QoS networks, best-effort networks

I. Introduction

Interest in proxy based caching has increased with the growth in Internet traffic and the initial research in this area (e.g., within the Harvest project [1]) has quickly led to the development of commercial products and continuing research activity (e.g., [2], [3], [4], [5], [6], [7]). The great majority of recent research and development on proxy caching has focused on techniques that can handle generic web objects; among the "cacheable" objects no distinction is made between, say, an HTML text file and a JPEG image. As real-time streaming video is becoming a significant proportion of network traffic and, given the large data volumes involved and its variable-bit-rate (VBR) nature, even a few popular video applications can result in potential network congestion problems. The congestion can cause not only packet loss, but also packet delays, which degrade the video playback quality dramatically (since the packets that arrive after their playback time are useless).

In this paper we focus on the caching problem specifically for streaming video objects. Some recent work has proposed that having caching strategies which are specific for particular types of objects can help improving the overall performance. In particular, for some objects, it is possible to perform "partial caching", where only part of the objects are stored on the proxy, as opposed to the "complete caching" where the objects are stored completely. An example can be found in "soft caching" for images ([8], [9], [10]). Approaches for partial caching for video have also been proposed in [11], [12]. In this paper, we study a "selective caching" strategy [13], which selects only *certain parts* of the video to be cached. We will show that the strategy to use depends on the specific network environment; and we focus on two representative scenarios, namely networks with Quality-of-Service (QoS) and best-effort networks. We provide a selective caching method for each of these scenarios.

Our proposed caching methods are *frame-wise* selection algorithms, i.e., the smallest caching unit we consider is one frame of the video (each frame may contain different number of bytes). Since there can be frequent changes on caching parameters (e.g., the popularity of the video objects), it is desirable to enable the proxy to be *scalable*, i.e., to be able to easily increase/reduce the portion of video being cached while still maintaining good performance. This scalability is inherent in our proposed frame-wise selective caching methods, by which the proxy can simply add/drop the selected frames as the environment changes.

A. System architecture

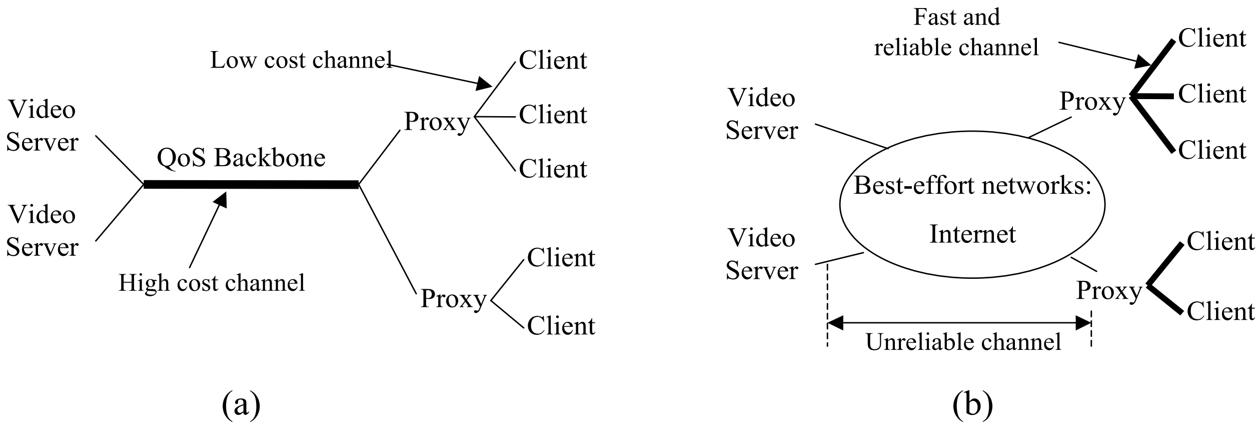

Fig. 1. System architecture. The proxies are set close the clients, and are connected to the video server via either a QoS (a) or best-effort network (b).

The server-proxy-client model used in this paper is shown in Fig. 1. The clients are attached to the proxy and all their requests for videos go through the proxy. The proxy allocates a certain cache space for each video sequence. Upon the client's request, if the frames are cached, the proxy will send them from its cache to the client directly; otherwise, the proxy will retrieve the frames from the video server. For a video streaming session, usually there is an end-to-end playback delay d, which is the interval between the time when the first packet is sent and the time when the first frame is displayed. For a continuous playback, the transmission of any given frame cannot exceed the delay d. We are also interested in other parameters for the design of selective caching, such as the server-proxy and proxy-client channel characteristics; the cache space (H) allocated on the proxy for a particular video; and client buffer size (B_c). We consider two network cases in the following and will propose different strategies for selective caching for each of them.

Case 1: Proxy caching in QoS networks (see Fig. 1a). The bandwidth on the server-proxy channel can be reserved, and the cost of reservation is proportional to the reserved bandwidth. The goal of caching in this case is to reduce the amount of bandwidth C that has to be reserved on the server-proxy backbone channel (therefore reducing the network cost), while minimizing the required client buffer size B_c to achieve that reduction, given a limited cache space H.

Case 2: Proxy caching in best-effort networks without QoS on the server-proxy channel, as for example in the current Internet (see Fig. 1b). The delivery of data over these channels is vulnerable to congestion, delay and loss, which are harmful for real-time streaming video delivery. The caching goal for this case is to provide more robustness[1] for continuous playback against poor network conditions on the proxy-server channel.

To improve the streaming performance given that the proxy-client channel is fast and reliable, one could consider using the proxy as an external buffer for each client, i.e., such that a client with minimal buffer resources can store some of the incoming video data at the proxy. In this case the proxy would need to allocate separate storage resources to each client for each streaming session (even when some sessions may be accessing the same video object). Thus, as a proxy may serve a large number of clients simultaneously, this scenario may require the availability of high performance caching resources at the proxy, including not only memory but also output bandwidth. Both these resources have to increase as

[1] See Section IV for detailed definition of robustness.

the number of clients is increased, since all the clients active at any given time will be simultaneously using the proxy for secondary storage.

Thus, in this paper, we focus on a less resource-intensive caching scenario where caching storage is assigned to *each video object*, rather than to each client. The assumption here is that the data stored for each video object changes only when the popularity of the video object changes and that the storage is shared by all clients accessing a given video object. Therefore the requirement of the cache storage space is reduced. In additional, this approach reduces the amount of data that the proxy has to provide to the clients (since only certain video frames need to be served from the proxy) and also requires substantially less real time storage management (since only when the popularity of an object changes is the storage devoted to it modified.)

In both cases, the proxy-client channel delivers the data originated from both server and proxy. In our model we assume that the main network bottleneck (in terms of both cost and reliability) is the server-proxy channel, and therefore the proxy-client link is assumed to have "left-over" bandwidth even when video transmission is ongoing. One example of such a scenario would be accessing a relatively low bandwidth video stream through a DSL/cable link. Another example would that where proxy and client are co-located within the same local area network [11]. The algorithms proposed here are good approximations for the case when there is a substantial amount of "left-over" bandwidth, in which we can assume that the delay in delivering a frame from the proxy to the client is very small (even when transmission of other frames from the server is taking place simultaneously.) This will enable us to assume that, since they can delivered in a very short time, frames stored at the proxy consume practically no client buffer memory. Therefore, to simplify the analysis, we exclude them from client buffer consumption in the rest of this paper. For the scenario when when the "left-over" bandwidth on proxy-client channel is not large enough to ignore the delay between proxy and client, our buffer analysis can still be used as an approximation for the proposed selective caching algorithms. In the extreme case where there is no "left-over" bandwidth, and therefore the client cannot receive data simultaneously from the proxy and the server, our proposed will reduce to the prefix caching proposed in [12], which will provide the optimal solution.

We will show that the performance of some "partial" video caching strategies may be limited by client buffer size constraints. For example, it is obvious that in QoS networks caching any part of the video can reduce the server-proxy bandwidth C, since less data has to be retrieved from the server directly. However, the bandwidth may be reduced at the expense of increasing in the required client buffer size B_c, because the frames that are stored at the server (not cached at the proxy) will have to be buffered at the client until they are displayed. We will show that the increment in B_c varies among different selective caching strategies which lead to the same bandwidth reduction. Similar constraints also exist in the case of video caching for best-effort networks. Therefore, there are trade-offs between the reduction of C (or improvement of U) and the memory size B_c to achieve it. Our goal is to *find proper selective caching methods such that the best performance is achieved under a constraint on B_c*. Thus one of the main differences between our work and other proposed caching algorithms (see below) is that we will consider the storage constraints at *both* client and proxy.

In summary, the main assumptions we make in our work, which also explain how it differs from other proposed caching algorithms (see below), are (i) that memory at the client is constrained, (ii) that cost considerations preclude using the proxy cache to provide dynamic "additional memory" for each client buffer, and (iii) that there is some "left-over" bandwidth in the proxy-client link. We will propose two approaches for selective caching in each of the above cases, namely, *Selective Caching for QoS networks* (SCQ) and *Selective Caching for Best-effort networks* (SCB).

B. Related work

Proxy caching for video has been explored in [12], [11], [14] under network conditions similar to those in Case 1 (QoS networks in Fig. 1a). Prefix caching, proposed by Sen *et al.* [12], is a special form of selective video caching, which involves caching only a group of consecutive frames at the beginning of the video sequence to smooth and reduce the bit-rate of a VBR video. We will show that our proposed SCQ algorithm, compared to prefix caching, requires less client buffer B_c while achieving the same bandwidth reduction (Case 1), and that SCB can improve robustness more than prefix caching (Case 2). Note that when B_c is large, SCQ/SCB can reduce to prefix caching (see Sections III and IV-B).

Wang *et al.* propose a "video staging" algorithm in [11], which prefetches to the proxy a portion of bits from the video frames whose size is larger than a pre-determined "cut-off" rate, to reduce the bandwidth on the server-proxy channel. Therefore some frames are separated into two parts: one is cached on the proxy and the other remains on the server. By contrast, our proposed algorithms are *frame-wise* caching schemes so that a frame is either not available at the proxy or it is stored in its entirety. One advantage of frame-wise caching is that those frames available in the proxy can actually be played by the client (this would not be possible with partially cached frames) in the event where congestion prevents any data from being delivered from the server for some period of time. Another advantage of frame-wise caching scheme is that the proxy can easily add (or drop) more frames when cache space increases upon the changes of caching condition (such as network status, video object popularities, cache space, etc.), by using a caching table created off-line (proposed in Section IV-D). As an example, with a staging approach, if the popularity of a video increases the proxy will need to increase the percentage of data in *all* frames (sending a request to the server to achieve this). Instead, with frame-wise caching only a few complete frames need to be requested from the server.

Ma *et al.* [14] also study a frame-wise video caching problem slightly different from that in Case 1, where selective caching is performed but the algorithm attempts to select groups of consecutive frames rather than isolated frames, in order to reduce the complexity of proxy management. In our work, by using a caching table, the proposed SCQ/SCB algorithms can select isolated frames (during iterations) to maximally improve the overall performance without increase the complexity for proxy online operations. Another major difference between our proposed work and other works in [14], [12], [11] is that we provide a caching strategy (SCB) for video delivery over best-effort networks, which are the most popular nowadays but are not explicitly considered in those works.

Both proposed SCQ/SCB algorithms consider non-layered coded video streams, while caching for layered (scalable) video can be found in [15], [16]. Rejaie *et al.* [15] propose a video caching algorithm for scalable video, which co-operates with the congestion control mechanism (for best-effort networks in Case 2) proposed in [17]. This work studies the caching replacement mechanism and cache resource allocation problem, according to the popularity of video objects, e.g., more layers of the video with higher popularity will be cached, and vice versa. Therefore the overall streaming performance can be improved (e.g., less network congestion, better playback quality). Kangasharju *et al.* formulate the caching problem for layered video differently, aiming to maximize the overall revenue for the service providers [16]. Tewari *et al.* [18] and Reisslein *et al.* [19] study cache replacement of streaming media (in non-layered format) to improve the cache hit ratio and therefore the streaming quality. Our work can be complementary to [18], [19], as we are focusing on the problem of selecting which part of the video should be cached, after the cache space for this particular video has been allocated.

The rest of this paper is organized as follows. Section II gives the background and definitions. Section III addresses the video caching problem for QoS networks and proposes the SCQ algorithm, while the SCB algorithm is proposed in Section IV to solve the caching problem for best-effort networks. The experimental results are shown in Section V. Finally, Section VI concludes the paper.

II. Basic definitions

Most standard video codecs (e.g., [20], [21]) produce VBR data after compression, which leads to high data burstiness. Usually there is a small start-up playback delay d (here we define it as the duration between sending out the first packet and playing the first frame) for most existing streaming video services to allow the client buffer to store a few beginning frames before playback starts. This delay is useful to (i) smooth the burstiness of VBR video data [22]; and (ii) to provide robustness against packet delay resulting from poor network conditions (in best-effort networks), thus playback from the client buffer is possible even when frames are being delayed. For this reason, we always cache this beginning portion of video data, which is referred as the "required initial buffering segment" (I_{req}), such that the client can start to playback with a smaller start-up delay. When we can assign more cache space than I_{req} for a given video, we can choose between continuing to cache the immediately following frames (as would be done in a prefix caching technique), or instead selecting other intermediate frames to be cached. In this paper, we consider the latter option, i.e., selective caching rather than prefix caching.

Assume there are N frames in a video V, denoted as $F(i), i = [1, 2, ..., N]$; each frame $F(i)$ has a size of $R(i)$ bytes, and a constant playback duration of T_F seconds (e.g., $T_F = 1/30$ *second*). We discretize the time axis t with intervals of T_F. The total size of the video is $R_{total} = \sum_{i=1}^{N} R(i)$. We define a "caching indicator sequence", $\mathcal{A} = [a(1), a(2), ..., a(N)]$, to indicate whether the i^{th} frame $F(i)$ is cached or not:

$$a(i) = \begin{cases} 0 & \text{if frame } F(i) \text{ is not cached} \\ 1 & \text{if frame } F(i) \text{ is cached} \end{cases} \tag{1}$$

A sequence $\mathcal{A}$ uniquely defines which part of video is cached, and can represent a selective caching scheme. We denote $\mathcal{A}_\phi$ (or simply ϕ) as the zero sequence, i.e., the sequence where no frames are cached so that all $a(i) = 0$. A possible $\mathcal{A}$ must satisfy the cache space constraint

$$\sum_{i=1}^{N} R(i)a(i) \leq H, \quad a(i) \in \mathcal{A} \tag{2}$$

We also denote $\mathcal{A}^1$ as the indicator sequence where only I_{req} is cached. H is pre-determined for each video object.

III. Video caching in QoS networks

A. Problem formulation

An example of a QoS network is shown in Fig. 1a. We assume that a CBR bandwidth is reserved on the server-proxy backbone, and the cost of reservation is proportional to that bandwidth. Therefore we always reserve only the minimum required bandwidth for video delivery in a particular streaming session. Define C_r as the *required bandwidth* of a CBR channel to deliver the video V on time for real-time playback without jitter, given a finite start-up delay d. Due to the data burstiness of VBR video, C_r is usually higher than the *average video data rate*, R_{avg}, to avoid decoder buffer underflow. Feng *et al.* [23] proposed a general method to find such C_r for pre-coded video without caching (where C_r is defined as *critical bandwidth*) in [23]. We will show that C_r changes after some frames are cached, and is a function of caching indicator $\mathcal{A}$. One of our objectives in this case is to choose $\mathcal{A}$ to minimize $C_r(\mathcal{A})$ by caching selected frames, so that the bandwidth reservation cost on server-proxy channel can be reduced.

However, while there may exist many possible $\mathcal{A}$ which lead to the same maximum reduction on $C_r(\mathcal{A})$, they may require different amount of client buffer size B_c to achieve that reduction, as shown in the analysis below. Considering both bandwidth and buffer size, we formulate the video caching problem for QoS networks as follows.

Problem formulation 1: *Given a limited proxy cache space H for a video sequence ($H < R_{total}$), a pre-encoded video stream V with N frames and a fixed delay d, among all possible $\mathcal{A}$ satisfying (2), find $\mathcal{A}^*$ which maximally reduces $C_r(\mathcal{A})$ after caching while requiring a minimum B_c to achieve that bandwidth reduction.*

B. Analysis on bandwidth reduction

To calculate $C_r(\mathcal{A})$, here we will extend the solution in [23] with minor modifications. We will only refer to the results from [23]; readers can refer to [23], [24], [25] for more details. Assume the transmission starts at time $t_0 = -d$, and playback starts at $t = 0$, the start-up delay is d ($d \geq 0$). Frame $F(i)$ is scheduled to be displayed at time $t = i$. Define $S_{\mathcal{A}}(t)$ as the *cumulative frame rate* at time t, for a given $\mathcal{A}$,

$$S_{\mathcal{A}}(t) = \sum_{i=0}^{t}(1 - a(i))R(i). \tag{3}$$

Define $C(t)$ as the server-proxy channel bandwidth at time t. A feasible channel rate function $C(t)$ to ensure a continuous playback must meet the following constraint:

$$\sum_{i=-d}^{t} C(i) \geq S_{\mathcal{A}}(t), \quad \text{for all} \quad t \in [1, N]. \tag{4}$$

$S_{\mathcal{A}}(t)$ and $\sum_{i=-d}^{t} C(i)$ can been thought of as the video data *consumption curve* at the client and the data *supply curve* from the channel, respectively. Since cached frames do not consume the server-proxy bandwidth and they can be fetched when needed (right before decoding) from the proxy, then the cached frames can be excluded from the client consumption curve in (3). Eq. (4) means that the supply curve should be greater than the consumption curve in order to avoid client buffer underflow (see Fig. 2). For a CBR channel, the bandwidth cannot exceed the constant allocated rate, so that

$$C(t) \leq C_r(\mathcal{A}), \quad \text{for all} \quad t \in [1, N]. \tag{5}$$

Define the *slope function* of a video sequence to be $L_{\mathcal{A}}(t)$,

$$L_{\mathcal{A}}(t) = S_{\mathcal{A}}(t)/t. \tag{6}$$

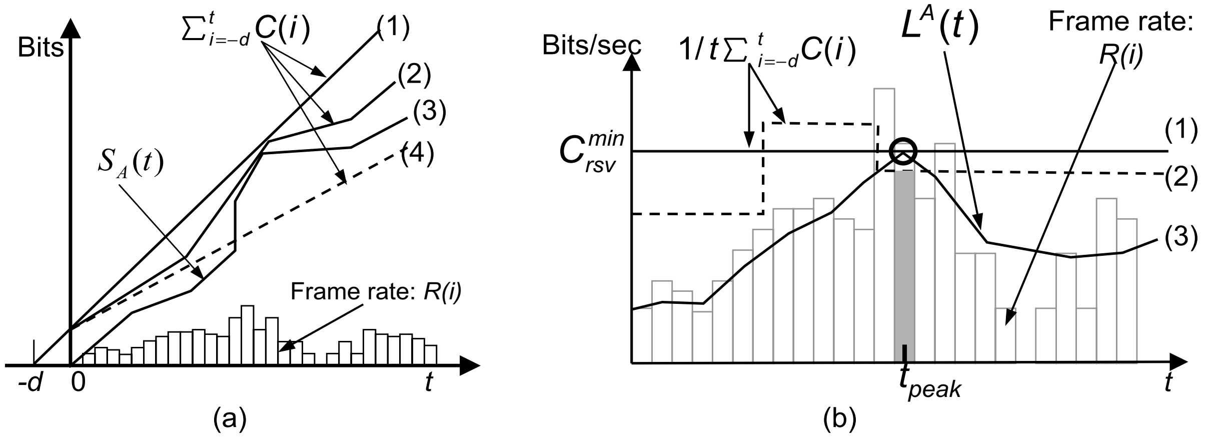

Fig. 2. Cumulative rate and slope functions. (a): Cumulative frame/channel rate. Curve (1), (2) and (4) are cumulative channel rate functions, $\sum_i C(i)$. Among them only (1) and (4) represent CBR channels. Curve (3) is the cumulative frame rate, $S_{\mathcal{A}}(t)$. Curve (1) and (2) are feasible channel rate, while (4) is not. (b): Slope function $L_{\mathcal{A}}(t)$ is drawn in curve (3). The minimum CBR channel bandwidth that has to be reserved is $C_r = \max\{L_{\mathcal{A}}(t)\}$.

Fig. 2 shows an example of $S_{\mathcal{A}}(t)$ and $L_{\mathcal{A}}(t)$. In fact, $L_{\mathcal{A}}(t)$ represents the lower bound of a feasible $C(t)$, and C_r can be obtained as (see [23], [24], [25] for a proof)

$$C_r(\mathcal{A}) = \max_{1 \le t \le N} \{L_{\mathcal{A}}(t)\}, \quad t \in [1, N], \tag{7}$$

i.e., the minimum bandwidth $C_r(\mathcal{A})$ that has to be reserved on a CBR channel is the maximum value of the video slope function $L_{\mathcal{A}}(t)$, which is reached at time t_{peak}, referred to as the *consumption peak time*,

$$t_{peak} = \arg \max_{1 \le t \le N} \{L_{\mathcal{A}}(t)\}. \tag{8}$$

Suppose we want to cache one more frame $F(k)$ after some frames have already been cached. Let $\mathcal{A}$ and $\mathcal{A}_k$ be the caching indicator sequences before and after caching $F(k)$, respectively. The following proposition shows the change in bandwidth after caching $F(k)$.

Proposition *1: For a given $F(k)$ and $\mathcal{A}$, $\max\{L_{\mathcal{A}_k}(t)\} = \max\{L_{\mathcal{A}}(t)\} - \Delta l$, where*

$$\Delta l = \begin{cases} R(k)/t_{peak}, & k \in [1, t_{peak}], \\ 0, & \textit{otherwise.} \end{cases} \tag{9}$$

See Appendix for a proof. An illustration is shown in Fig. 3. This means that $\max\{L_{\mathcal{A}_k}(t)\}$ can be reduced if and only if $k \in [1, t_{peak}]$. Therefore we should cache a frame *before* t_{peak} to reduce C_r, and the reduction Δl depends only on the size of the cached frame but not on its position, i.e.,

$$C_r(\mathcal{A}_k) = C_r(\mathcal{A}) - R(k)/t_{peak}, \text{ if } k \in [1, t_{peak}]. \tag{10}$$

The above conclusion indicates that caching successive frames from the beginning of the video sequence is one of the caching schemes that can reduce C_r maximally. As an example, "prefix caching", which selects the group of beginning frames to be cached until the cache space is full, is a good approach to reduce the bandwidth requirements [12].

C. Client buffer analysis

Recall that we need to select frames to be cached to (i) *maximally reduce* $C_r(\mathcal{A})$, and (ii) *require the minimum* B_c *to achieve that reduction.* Selecting different frames to be cached within the period of $[1, t_{peak}]$ leads to different requirements of B_c. This can be explained by the following buffer analysis. Define a byte-level buffer trace function, $B_{\mathcal{A}}(t)$, as the amount of video data (in number of bytes) in the client buffer during playback (given a cache sequence $\mathcal{A}$), which can be written as (11) (see [23]),

$$B_{\mathcal{A}}(t) = \sum_{i=-d}^{t} C(i) - S_{\mathcal{A}}(t), \tag{11}$$

where $C(i) \le C_r(\mathcal{A})$ in (7). The required client buffer size, $B_{max}(\mathcal{A})$, is the maximum value of $B_{\mathcal{A}}(t)$. Thus we need to minimize $B_{max}(\mathcal{A})$ in order to reduce the required client buffer size to achieve the reduction of $C_r(\mathcal{A})$ stated in Proposition 1. We also denote t_{max} as the *buffer peak time*, when $B_{\mathcal{A}}(t)$ reaches to $B_{max}(\mathcal{A})$,

$$t_{max} = \arg\max_{1 \le t \le N} \{B_{\mathcal{A}}(t)\}. \tag{12}$$

We first examine the change from $B_{\mathcal{A}}(t)$ to $B_{\mathcal{A}_k}(t)$ for the period of $1 \le t \le t_{peak}$, after caching one frame $F(k)$ (note that $1 \le k \le t_{peak}$ according to Proposition 1).

Because the channel bandwidth is constant, see (4) and (5), we have $C(t) = C_r(\mathcal{A})$ for $1 \le t \le t_{peak}$ (so that $C(t) = C_r(\mathcal{A}_k)$ after caching $F(k)$). From (11) and (10) we have

$$B_{\mathcal{A}_k}(t) = C_r(\mathcal{A}_k)t - S_{\mathcal{A}_k}(t)$$

$$= \quad C_r(\mathcal{A})t - R(k)\frac{t}{t_{peak}} - S_{\mathcal{A}_k}(t), \tag{13}$$

where $1 \leq t \leq t_{peak}$. Note that after caching $F(k)$, $S_{\mathcal{A}_k}(t) = S_{\mathcal{A}}(t)$ for $1 \leq t < k$; and $S_{\mathcal{A}_k}(t) = S_{\mathcal{A}}(t) - R(k)$ for $k \leq t \leq t_{peak}$. Thus we have

$$B_{\mathcal{A}_k}(t) = \begin{cases} B_{\mathcal{A}}(t) - R(k)\frac{t}{t_{peak}}, & 1 \leq t < k, \\ B_{\mathcal{A}}(t) - R(k)\frac{t}{t_{peak}} + R(k) & k \leq t \leq t_{peak}. \end{cases} \tag{14}$$

Eq. (14) means that after caching frame $F(k)$, the new buffer trace *decreases* before k (due to the reduction of C_r), and *increases* during $[k, t_{peak}]$ (due to the removal of $R(k)$ from the consumption curve)[2]. It also indicates that at any particular time $t \in [1, t_{peak}]$, the amount of increment/decrement of $B_{\mathcal{A}_k}(t)$ depends only on $R(k)$. More specifically, if $t_{max} < t_{peak}$, we can reduce $B_{max}(\mathcal{A}_k)$ *only* if we cache frame $F(k)$ after t_{max}, and we will have

$$B_{max}(\mathcal{A}_k) = B_{max}(\mathcal{A}) - \Delta b, \qquad \text{if } t_{max} < k \leq t_{peak}, \tag{15}$$

$$\text{where} \qquad \Delta b = \frac{t_{max}}{t_{peak}} R(k). \tag{16}$$

The change of $B_{\mathcal{A}_k}(t)$ for the remaining period $t_{peak} \leq t \leq N$ may not be easily expressed in a closed form. However, based on the results from [23], [24], [25] we can find that the *maximum* buffer occupancy may increase (or at least not decrease) for the period of $t_{peak} < t \leq N$. This is because $B_{\mathcal{A}_k}(t_{peak}) = B_{\mathcal{A}}(t_{peak}) = 0$, and the only difference is that a smaller bandwidth ($C_{\mathcal{A}_k}(i)$) is available to transmit the video data for $t > t_{peak}$, after caching frame $F(k)$ (see [23] for details). Proposition 1 and (14) assume that t_{peak} or t_{max} do not change after $F(k)$ is cached. However those results still hold when t_{peak} or t_{max} changes, except that the absolute values of Δl and Δb will become smaller, which would not affect the conclusions in next section.

D. Proposed SCQ algorithm

The results in (10) and (15) lead to the following conclusion for caching one frame $F(k)$: *one can cache the frame at t_{peak}, $F(t_{peak})$, in order to minimize $C_r(\mathcal{A})$ and $B_{max}(\mathcal{A})$.* The reasons are:

[2] Eq. (14) is obtained as the delay of transmitting $F(k)$ (over the proxy-client channel) is small, and the delay is ignored to simplify the analysis. The frames after $F(k)$ are received earlier than they would have if caching is not used.

1. From Proposition 1 we have to cache a frame before t_{peak} to reduce C_r.

2. From (14) and (15), if B_{max} is reached before t_{peak} (i.e., $t_{max} < t_{peak}$), caching the frame $F(t_{peak})$ can also reduce B_{max}. More specifically, in this case, caching any frame within the period of $(t_{max}, t_{peak}]$ has the *same* effect in terms of reducing both B_{max} and C_r. Similarly, caching any frame of a given size within the period of $[1, t_{max}]$ has the *same* effect on reducing C_r while *increasing* B_{max}. See Fig. 4 for an illustration. Therefore, simply selecting frame $F(t_{peak})$ is guaranteed to reduce B_{max} and C_r, without requiring to compute t_{max}.

3. If B_{max} is reached *after* t_{peak} (i.e., $t_{max} > t_{peak}$), caching any frames before t_{peak} can not reduce (or may increase) B_{max}. However, from (14), we can see that caching frame $F(t_{peak})$ has the effect of reducing $B_{\mathcal{A}}(t)$ for the longest duration of $[1, t_{peak}]$. Also as more frames are cached, t_{peak} tends to increase monotonically. Once $t_{max} < t_{peak}$, we will have the above situation 2, and the new B_{max} is already reduced if we keep on caching frame $F(t_{peak})$.

The above analysis shows that the changes on $C_r(\mathcal{A})$ and $B_{max}(\mathcal{A})$ depend *only* on the cached frame size (i.e., the absolute values of Δl and Δb *depend only* on $R(k)$), not the exact position of the cached frame, as long as that frame falls in the range of $[1, t_{peak}]$.

For a VBR video, the frames around t_{peak} may have different sizes, the search for the optimal selection of frames to be cached can be complex when cache space H is large. Therefore we propose an heuristic approach, SCQ, for selective video caching in QoS networks, which selects frames iteratively. During each iteration, SCQ computes the $L_{\mathcal{A}}(t)$ and locates t_{peak} using (8), then selects the frame at t_{peak} (i.e., $F(t_{peak})$) to be cached. This process is iterated until the cache space is full. The detailed procedure is summarized in the following steps.

Step 1. Initialization. Set $n = 1$ (n is the iteration index). Cache the required initial segment I_{req} and set $\mathcal{A}^1$ correspondingly (see Section II). Let $\mathcal{A}^n$ be the cache indicator after the n^{th} iteration. Set $H \Leftarrow H - I_{req}$.

Step 2: Find $t^n_{peak} = \arg\max_t\{L_{\mathcal{A}^n}(t)\}$ for the n^{th} iteration.

Step 3: Cache frame $F(t^n_{peak})$, set $a(t^n_{peak}) = 1$; set $H \Leftarrow H - R(t^n_{peak})$.

Step 4: If there is no cache space left ($H \leq 0$), procedure ends. Otherwise, set $n \Leftarrow n + 1$, go to Step 2 (start next iteration).

We will obtain the solution $\mathcal{A}^* = \mathcal{A}^n$ at the last iteration. Note that usually $t_{max} > t_{peak}$ during the initial iterations. Therefore increasing of B_{max} can not be avoided since we have to cache a frame before t_{peak} to reduce bandwidth. B_{max} starts to decrease once $t_{max} < t_{peak}$ after more frames are cached, and SCQ tries to keep that initial increment as small as possible. See the results in Section V.

IV. Video caching in best-effort networks

An example of video caching in best-effort networks is shown in Fig. 1b. The server-proxy channel bandwidth may have variations due to network congestion or other poor conditions. These variations can cause dramatic degradation on continuous video playback quality, since packets that arrive too late are considered to be lost. Thus it is useful for the client to buffer a certain number of frames before and during playback, in order to increase the likelihood that frames are available for playback in the decoder buffer during the periods of packet lost (delay). The more frames are buffered at a given time, the more *robustness* there will be against the packet delay.

However, the frames in VBR video have different sizes, which means that the number of buffered frames in the client's buffer may not be *constant* during playback. Periods during which the number of frames in the decoder buffer becomes low are referred to as the *risky periods* (less robust). Our caching goal here is to improve the robustness by increasing the number of frames in the client buffer during risky periods.

As we are focusing on an off-line caching algorithm that only has the knowledge of the video sequence, we do not make any assumptions about the actual bandwidth variations while deriving the caching algorithm. The risky periods of the video sequence can be located before transmission by analyzing the decoder buffer contents during a "virtual" playback, where a constant server-proxy bandwidth C (close to the average video bit-rate) is applied. Note that the network congestion may happen any time during a real-time session. However the congestion is more likely to cause quality degradation when the client buffer contains fewer frames, i.e., during the risky periods identified in our analysis. In the simulation, we transmitted video (with partial caching) using a server-proxy channel with bandwidth variations to verify the effectiveness of the buffer analysis and the caching algorithm.

We define a frame-level *buffer trace* function, $B^f(t)$, which indicates the number of *frames available at the client* during the playback. A measurement of the robustness of a video stream U can be defined as

$$U = \min_t \{B^f(t)\}, \tag{17}$$

i.e., the minimum value (referred to as a *trough*) of the buffer trace in number of frames. Risky period is the time when a trough occurs, i.e., $t_r = \arg\min_t\{B^f(t)\}$. There might be many risky periods/frames in one session. The larger $B^f(t)$, the more robust this video stream will be around time t.

The robustness metric U defined in (17) corresponds to using a *MaxMin* criterion (as we will try to maximize U by caching frames, see below). Obviously there are alternative ways to define robustness such as, for instance, the average number of frames in the buffer (referred to as a *MaxAverage* criterion):

$$U_a = \frac{1}{N}\sum_{t=1}^{N} B^f(t). \tag{18}$$

Each of these two measures of robustness, U or U_a, leads us to different algorithms to select which frames should be cached. For most scenarios in this paper, we use the MaxMin criterion for robustness, and we will use the MaxAverage criterion only to break the tie among multiple choices that all improve U in the same way (see Section IV-C).

Note that $B^f(t)$ (measured in number of frames) is used to calculate robustness; while $B(t)$ (measured in number of bits, see (11)) is used to determine the occupancy of the client physical buffer during playback. It should be emphasized that *both* the frames in the client's physical buffer and the cached frames at the proxy are counted as the *available* frames for the client. In other words, cached frames can increase $B^f(t)$ (and therefore the robustness), while not occupying client physical buffer, i.e., they do not increase $B(t)$[3].

A. Problem formulation

In this case the decoder buffer size B_c is the bottleneck in improving U. For example, a straightforward way to improve U is to cache the "earlier" frames from the beginning of video sequence. Thus the client can retrieve the "later" non-cached frames from the server, while it is displaying the cached frames retrieved from proxy. At the time when those "later"

[3] Again, this is true because the cached frames can be fetched quickly from proxy right before their playback time, such that they can be excluded from the consumption of client physical buffer, see explanation in Section-I.

frames that are not cached start to be displayed, many of them are already buffered at the client, and therefore U is improved. However, this method could soon fill up the client buffer since the frames retrieved from the server are not played until all the cached frames (scheduled to be displayed at earlier time) are displayed. Thus the server may have to slow down the transmission speed when the client buffer is full, which waste the bandwidth to further increase the robustness. Therefore a proper selection scheme should be designed under the constraint a limited memory buffer B_c. We formulate the caching problem as follows (assume B_{max} is known).

Problem formulation 2: *Given a limited cache space ($H < R_{total}$) on the proxy, a pre-encoded video stream V with N frames and a fixed delay d, among all possible $\mathcal{A}$ satisfying (2), find the cache indicator sequence $\mathcal{A}^*$ such that the robustness $U = \min_t\{B^f_{\mathcal{A}^*}(t)\}$ is maximized, without exceeding the maximum client buffer size B_{max}.*

B. Analysis on buffer trace after caching

The byte-level buffer trace function $B(t)$ can be computed from (11). The frame-level buffer trace function $B^f(t)$ can be obtained by simulating the transmission frame by frame, assuming that the nominal channel rate C is provided. An example is shown in Fig. 5.

We first study the case of caching a single frame. For a given $\mathcal{A}$, and available channel bandwidth C, if the client buffer size (B_{max}) is large enough, i.e., $B_{max} \geq \max_t\{B_{\mathcal{A}}(t)\}$, then based on (11) we have

$$\begin{aligned} B_{\mathcal{A}}(t) &= C(t+d) - S_{\mathcal{A}}(t) \\ &= C(t+d) - \left(\sum_{i=1}^{t} R_i - \sum_{i=1}^{t} a(i)R(i)\right) \\ &= B_\phi(t) + \sum_{i=1}^{t} a(i)R(i), \qquad (19) \end{aligned}$$

where $B_\phi(t)$ is the buffer trace (in bits) where no frame is cached ($\mathcal{A} = \phi$). From (19) it can be seen that *caching one frame $F(k)$ increases the buffer trace $B_{\mathcal{A}}(t)$ for the duration $t \in [k, N]$ by the cached frame size $R(k)$*. This is because, after being cached, frame $F(k)$ will be fetched from the proxy rather than from the server, thus all the non-cached frames later than $F(k)$ are "shifted" to an earlier transmission time (because frame $F(k)$ is skipped for transmission) from server. Those later frames (after $F(k)$) will stay in the client buffer for a longer period and thus increase $B_{\mathcal{A}}(t)$ (and corresponding $B^f_{\mathcal{A}}(t)$), by caching $F(k)$.

A simple example of the "raise" in $B_{\mathcal{A}}(t)$ after caching $F(k)$ is shown in Fig. 6. However, (19) does not hold if there is a tight buffer size limitation, i.e., $B_{max} < \max_t\{B_{\mathcal{A}}(t)\}$. If the decoder buffer is full, then the server and proxy have to reduce the transmission speed, which means $C(i)$ is smaller than C, otherwise packets will be discarded due to client buffer overflow.

Proposition *2: If there exists a t_{max} (defined in (12)) such that $B_\phi(t_{max}) = B_{max}$ (where B_{max} is the known maximum buffer size), then caching one frame $F(t_i)$ which is located before t_{max} ($t_i < t_{max}$) can only increase the buffer trace by approximately $R(t_i)$ between time $\hat{t}_i$ and t_{max}, where $\hat{t}_i$ is the transmission time of frame $F(t_i)$, and $\hat{t}_i < t_i < t_{max}$.*

$$B_{\mathcal{A}}(t) = \begin{cases} B_\phi(t) & \text{if } t \le \hat{t}_i \ , \\ B_\phi(t) + R(t_i) & \text{if } \hat{t}_i < t \le t'_{max} \ , \\ B_{max} & \text{if } t'_{max} < t \le t_{max} \ , \\ B_\phi(t) & \text{if } t_{max} < t \le N. \end{cases} \tag{20}$$

See the Appendix for a proof. In short, before $B_{\mathcal{A}}(t)$ reaches B_{max} (i.e., $t_i < t < t'_{max}$), it is "lifted" according to (19), and as a result $B_{\mathcal{A}}(t)$ will reach B_{max} at an "earlier" time t'_{max} (where $t'_{max} < t_{max}$). Then the proxy and/or server have to reduce transmission speed after time t'_{max} when the buffer is full, until it is drained to accept further data. This reduction cancels out the "extra" frames cumulated in the buffer, so that the remaining buffer trace for $t > t_{max}$ is the same as if no frames are cached. See Fig. 7 for an illustration.

Caching multiple frames is similar to the single frame case. When additional frames are cached, the buffer trace $B(t)$ is "raised" consequently, and may hit the maximum bound B_{max} at more points. Therefore, we conclude that *each cached frame can only increase the buffer trace between its scheduled transmission time $\hat{t}_i$ and next nearest t_{max}, if it exists, where $B(t_{max}) = B_{max}$.*

C. Proposed SCB algorithm

Based on the above conclusions, we now propose the Selective Caching for Best-effort networks (SCB), which iteratively selects one frame that is located within the range of

$[t_{max}, t_r]$, where t_{max} is the closest buffer peak time before t_r. An example of SCB is shown in Fig. 8, where troughs occurs at t_1 and t_4 before caching. According to Proposition 2, caching frames before the buffer peak time t_2 will not lift $B^f(t)$ after t_2. So after caching frames before before t_1 to increase $B^f(t_1)$, we should select frames between t_2 and t_4 to increase $B^f(t_4)$ (see Fig. 8(b)), and therefore improve U defined in (17). The details of SCB algorithm are summarized in the following steps.

Step 1. Initialization. Same as Step 1 in SCQ algorithm. See Section III-D.

Step 2. In the n^{th} iteration, find the most *risky* period $t_r^n = \arg\min_t\{B_n^f(t)\}$, e.g., time t_1 in Fig. 8a. If there are multiple t_r^n, choose the first one (the MaxMin criteria is applied first, and MaxAverage is applied to break the tie choices if needed).

Step 3. Find the *nearest* buffer peak time t_{max}^n before the chosen t_r^n (e.g., t_2 for risky time t_4 in Fig. 8b). If no maximum peak exists before t_r^n, set $t_{max}^n = 0$ (e.g., $t = 0$ for t_1 in Fig. 8a). Note that t_{max}^n is obtained from the byte-level buffer trace $B_{\mathcal{A}^n}(t)$.

Step 4. Select *one* frame $F(c^n)$ which is right after t_{max}^n (obtained in Step 3) to be cached, set the $a(c^n) = 1$. Update the trace $B_{\mathcal{A}^n}^f(t)$ and set $H \Leftarrow H - R(c^n)$. If $H \leq 0$, there is not enough space left on proxy, procedure ends. Otherwise, set $n \Leftarrow n + 1$, go to Step 2.

In each iteration, we first locate the t_r for U, thus to increase U is the same as to increase $B^f(t_r)$. From Proposition 2, we know that in order to increase $B^f(t_r)$, we have to cache the frames *after* the nearest previous buffer peak time.[4] Since there might be multiple choices for selecting, e.g., caching any frames between t_{max} and t_r can increase $B^f(t_r)$, the MaxAverage criteria for robustness requires to select the frame furthest away from t_r but after the nearest previous peak, t_{max}. This provides the largest increase in average robustness U_a.

D. Caching table

Both SCQ/SCB algorithms select *additional* frames when there is more cache space available, while the frames selected earlier still remain being cached. Therefore we can perform

[4] Our goal is to increase the number of available frames in the buffer during the playback, which is measured by $B^f(t)$. The exact value of $B^f(t)$ may have a slightly different shape from $B(t)$, due the variable size of frames in VBR video. However, the increment in $B(t)$ should also lead to the increment in $B^f(t)$ when there is more data in the client buffer. Therefore the results in (19) and (20) can also be applied to approximate the increment in $B^f(t)$ after caching a frame $F(k)$.

the SCQ/SCB algorithms for the complete video sequence to determine the *order* of all frames to selected for caching (this can be achieved by setting the cache space equal to the size of the video), and store the results into a *cache table*. When the available cache space increases or decreases, then the proxy can add or remove frames that need to be cached according to the cache table, without re-computing the selection procedure. Thus, the caching scalability and low complexity (for online operations of the proxy) can be achieved, by using such a cache table.

V. Experimental results

Experimental results on the SCQ algorithm described in Section III-D are shown in Fig. 9. A video clip (part of movie *Star Wars* in MPEG-1 [26]) with 10,000 consecutive frames is used for simulation. The original video has the average rate (R_{avg}) of 5,516 KBits/sec, (peak frame rate is 9,812 KBits/sec). The total size of video clip (R_{total}) is 280.6 MBytes. Fig. 9a shows that the proposed SCQ algorithm can reduce the server-proxy bandwidth almost the same as that of the prefix caching algorithm. Fig. 9b shows that as expected, the SCQ algorithm requires a much smaller client buffer size to achieve the same bandwidth reduction as prefix caching.

In the experiments of SCB algorithm, the video clip having 10,000 frames is encoded with MPEG-2 under rate control[5] similar to [27]. The original video data has an average rate (R_{avg}) of 2,112 KBits/sec, with peak data rate 2,400 KBytes/sec. The average frame size is 88 KBits. The average channel bandwidth, C, is also 2,112 KBits/sec. The client buffer size (B_c) is 512 KBytes. Fig. 10a shows the robustness $U = \min_t\{B^f(t)\}$ with respect to the percentage of video being cached. Fig. 10b shows the average number of frames in the buffer during the playback, or $U_a = \frac{1}{N}\sum_{t=1}^{N} B^f(t)$. Note that as defined in Section IV-A, the selective caching uses the MaxMin robustness criterion first rather than the MaxAverage criterion to eliminate the worst case first. When only a portion of the video is cached, the SCB outperforms prefix caching in both cases (for U and U_a).

Fig. 11 shows the simulation results to verify the effectiveness of our definition of robustness

[5] The reason to apply rate control is to avoid large variance of video data rate to avoid potential network congestion in a best-effort network.

(U and U_a). We simulate the delivery of streaming video when it is partially cached. The cached part of the video is sent to client from the proxy with no loss. The non-cached frames are retrieved from server (via proxy), starting from the beginning of the playback session, and the data is buffered at client until it is displayed. The client physical buffer size is 512KB.

We use a binary erasure channel (BEC) [28] to model the server-proxy channel. A packet (we use a fixed packet size of 512 bytes) is lost with probability ϵ, and arrives to the client correctly with probability $1 - \epsilon$, where ϵ is the channel packet loss rate. All lost packets are recovered by retransmission. In this simulation all frames *have to be played*, thus if a frame arrives too late to the client, due to delay or retransmission, the previous frame is "frozen" on the screen until it arrives. We refer to the period during which a frame is "frozen" as the "jitter duration" T_J. The fraction of T_J/T_V (where T_V is the total video playback time) is used to measure the continuity of the playback. $T_J/T_V = 0$ means that the playback has no jitter; a larger T_J/T_V indicates that more jitter happens during the playback. Thus a smaller T_J/T_V indicates more robustness for a continuous playback. The experiment uses 1000 realizations, the packet loss is performed randomly with the given channel error ϵ. We can see from Fig. 11a shows that when a larger proportion of the video is cached, the robustness is increased and the jitter duration is reduced. Fig. 11b shows the jitter duration with different packet loss rate. In both cases, the proposed SCB algorithm leads to smaller jitter duration than prefix caching since it select frames to be cached to maximize the robustness.

In the presence of network congestion, the congestion duration can last over some unpredictable time-scale. During the congestion all packets are delayed, and the bandwidth drops to zero. Fig. 11c shows the results when both the channel congestion and the congestion duration (denoted by d_c) occurs randomly. d_c follows an exponential distribution with mean of m_c. We test the robustness with two cache schemes with different value of m_c. Fig. 11c shows that in both schemes, the jitter duration increases when m_c becomes large. This is true as jitter is more likely to happen when network congestion becomes severe (last longer). As expected, the proposed SCB algorithm outperforms prefix caching algorithms with different m_c. The results showed in Fig. 11 verify the effectiveness of our definition of robustness criteria developed for SCB algorithm.

VI. Conclusions

In this paper, two novel approaches for proxy caching of video are presented for both the QoS networks and best-effort networks (e.g., the Internet). The video caching performance is measured differently in these network environments, i.e., for QoS networks, the metric of interest is the network bandwidth cost; while the robustness of continuous playback against poor network conditions is more important in best-effort networks. Therefore the caching algorithms should be designed accordingly. We also emphasized that some resources, such as client decoder buffer size and limited proxy cache space, are also critical for the design of video caching algorithms. We proposed two caching algorithms, SCQ and SCB, for QoS and best-effort networks, respectively. SCQ can reduce the network cost of bandwidth reservation near optimally and requires a small client buffer size to achieve it; SCB can increase the playback robustness while not violating the client buffer size budget. Both SCQ and SCB algorithms provide good scalability for proxy space adjustment, and low complexity for proxy online operations. The proxy can easily reduce/increase the cache space for a video object while still maintain the good performance provided by these algorithms.

Both SCQ and SCB algorithms are designed for caching a single video object with a pre-allocated cache space. In the situation that the total cache space is limited, to determine how much cache space allocated to each video to maximize the overall performance can be an interesting resource allocation problem (e.g., [4], [19], [16], [5], [18].). The proposed caching algorithms (SCQ and SCB) in this paper are independent of any other cache space allocation mechanisms, and can be used in conjunction with them. The cache table described in Section IV-D can be used to find the trade-offs between the cache space and the caching performance for each individual video sequence.

Appendix

Proof of Proposition 1.

Proof: From (3) and (6) we get

$$L_{\mathcal{A}_k}(t) = \begin{cases} L_{\mathcal{A}}(t), & 1 \leq t < k \\ L_{\mathcal{A}}(t) - R(k)/t, & k \leq t \leq N \end{cases} \tag{21}$$

Recall the definition of t_{peak} in (8). If $k \leq t_{peak}$, from the second case in (21), we have $\max\{L_{\mathcal{A}_k}(t)\} < \max\{L_{\mathcal{A}}(t)\}$, therefore caching a frame $F(k)$ before t_{peak} can reduce the required bandwidth C_r. The reduction is $\Delta l = R(k)/t_{peak}$ (note that this assumes t_{peak} remains the same after caching $F(k)$; however the Proposition still holds if t_{peak} changes except that the absolute value of Δl is smaller than $R(k)/t_{peak}$). Obviously if $k > t_{peak}$, caching that frame cannot reduce the maximum of $L_{\mathcal{A}}(t)$ which occurs before k. ■

Proof of Proposition 2.

Proof: From (19), we know that without the physical buffer size constraint, $B_{\mathcal{A}}(t)$ would exceed B_{max}. Thus to prevent the buffer overflow, the server and/or proxy has to reduce the transmission speed ($C(i)$) at (or before) t_{max}. However we assume the reduction of $C(i)$ is as small as possible[6] so that client buffer is always kept full during the period around t_{max}[7]. Therefore, at time t_{max}, $B_{\mathcal{A}}(t)$ also reaches the maximum, $B_{\mathcal{A}}(t_{max}) = B_{max}$.

Because $R(t_i) > 0$ and $t_{max} \geq \hat{t_i}$, it is easy to see that there must exist $t'_{max} < t_{max}$ such that $B_{\mathcal{A}}(t'_{max}) = B_{max}$. This means $B_{\mathcal{A}}(t)$ reaches B_{max} at an earlier time t'_{max} due to the increase of $R(t_i)$.

For $\hat{t_i} \leq t < t'_{max}$, we know from (19),

$$B_{\mathcal{A}}(t) = B_{\phi}(t) + R(\hat{t_i}), \tag{22}$$

For $t'_{max} < t \leq t_{max}$, $B_{\mathcal{A}}(t)$ remains full at B_{max} (as the above assumption). We also have $B_{\mathcal{A}}(t_{max}) = B_{\phi}(t_{max}) = B_{max}$ For $t > t_{max}$, from (11) we know that

$$B_{\mathcal{A}}(t_{max}+1) = \sum_{i=1}^{t_{max}+1} C(i) - \sum_{i=1}^{t_{max}+1} R(i)$$

[6] This can be easily achieved by sending feedback from client to proxy indicating the buffer fullness during the transmission, so that the proxy/server can adjust the $C(i)$ accordingly.

[7] To always keep the buffer as full as possible is to maximize the robustness, by storing more frames in the buffer.

$$
\begin{aligned}
&= \Big(\sum_{i=1}^{t_{max}} C(i) - \sum_{i=1}^{t_{max}} R(i)\Big) + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{\mathcal{A}}(t_{max}) + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{max} + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{\phi}(t_{max}) + C(t_{max}+1) - R(t_{max}+1) \\
&= B_{\phi}(t_{max}+1)
\end{aligned}
$$

Applying this recursively for all $t_{max} < t \leq N$, we can get

$$B_{\mathcal{A}}(t) = B_{\phi}(t), \quad \text{where} \quad t_{max} < t \leq N. \tag{23}$$

Finally, for $t \leq \hat{t_i}$ ($\hat{t_i}$ is the transmission time of frame $F(t_i)$), obviously there is no change for the buffer trace, $B_{\mathcal{A}}(t) = B_{\phi}(t)$. Combining these results we get (20). ■

References

[1] A. Chankhunthod, P. B. Danzig, C. Neerdals, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," in *USENIX Tech. Conf.*, 1996.

[2] G. Abdulla S. Williams, M. Abrams, and S. Patel, "Removal policies in network caches for world-wide web documents," in *Proc. of ACM SIGCOMM'96*, Stanford, CA, Aug. 1996.

[3] J. Shim, P. Scheuermann, and R. Vingrale, "A case for delay-conscious caching of web documents," in *Proc. Intl. WWW Conf*, Santa Clara, CA, Apr. 1997.

[4] J. Shim, P. Scheuermann, and R. Vingrale, "Proxy cache algorithms: design, implementation, and performance," *IEEE Trans. on Knowledge and Data Engineering*, vol. 11, no. 4, pp. 549–562, July-Aug 1999.

[5] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Trans. on Networking*, vol. 8, no. 2, pp. 158–170, April 2000.

[6] A. Mahanti, C. Williamson, and D. Eager, "Traffic analysis of a web proxy caching hierarchy," *IEEE Network*, vol. 14, no. 3, pp. 16–23, May-June 2000.

[7] J. Wang, "A survey of web caching schemes for the internet," in *ACM Computer Communication Review*, Oct. 1999, vol. 29(5), pp. 36–46.

[8] A. Ortega, F. Carignano, S. Ayer, and M. Vetterli., "Soft caching: Web cache management for images," in *IEEE Signal Processing Society Workshop on Multimedia*, Princeton, NJ, June 1997.

[9] X. Yang and K. Ramchandran, "An optimal and efficient soft caching algorithm for network image retrieval," in *Proc. of ICIP*, Chicago, IL, Oct. 1998.

[10] J. Kangasharju, Y. Kwon, A. Ortega, X. Yang, and K. Ramchandran, "Implementation of optimized cache replenishment algorithms in a soft caching system," in *IEEE Signal Processing Society Workshop on Multimedia*, CA, Dec. 1998.

[11] Z.-L. Zhang, Y. Wang, D. H. C. Du, and D. Su, "Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks," *IEEE Trans. on Networking*, vol. 8, no. 4, pp. 429–442, Aug. 2000.

[12] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE Infocom. 99*, New York, USA, March 1999.

[13] Z. Miao and A. Ortega, "Proxy caching for efficient video services over the internet," in *9th International Packet Video Workshop (PVW '99)*, New York, April 1999.

[14] W.-H. Ma and H.C. Du, "Reducing bandwidth requirement for delivering video over wide area networks with proxy server," in *Proc. International Conference on Multimedia and Expo.*, 2000, vol. 2, pp. 991 –994.

[15] R. Rejaie, H. Yu, M. Handely, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet," in *Proc. of IEEE Infocom'2000*, Tel-Aviv, Israel, March 2000.

[16] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," in *Proc. of IEEE Infocom*, Anchorage, AK, USA, 2001.

[17] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet," in *Proc of ACM SIGCOMM '99*, Cambridge, MA., Sept. 1999.

[18] R. Tewari, H. Vin, A. Dan, and D. Sitaram, "Resource-based caching for web servers," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[19] M. Reisslein, F. Hartanto, and K. W. Ross, "Interactive video streaming with proxy servers," in *Proc. of First International Workshop on Intelligent Multimedia Computing and Networking (IMMCN)*, Atlantic City, NJ, Feb. 2000.

[20] J. Mitchell, W. Pennebaker, C. Fogg, and D. LeGall, *MPEG Video Compression Standard*, Chapman and Hall, 1996.

[21] "ISO/IEC JTC1/SC29/WG11, MPEG-4 version 2 visual working draft rev. 3.0, N2202," March 1998.

[22] J. Salehi, Z. Zhang, J. Kurose, and D. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing," in *IEEE/ACM Trans. Networking*, September 1998.

[23] Wu chi Feng, F. Jahanian, and S. Sechrest, "An optimal bandwidth strategy for the delivery of compressed prerecorded video," *ACM/Springer-Verlag Multimedia Systems Journal*, vol. 5, no. 5, Sept. 1997.

[24] J. Rexford and D. Towsley, "Smoothing variable-bit-rate video in an internetwork," *IEEE/ACM Transactions on Networking*, April 1999.

[25] S. Sen, J. Dey, J. Kurose, J.Stankovic, and D. Towsley, "Cbr transmission of vbr stored video," in *SPIE Symposium on Voice Video and Data Communications*, Nov. 1997.

[26] M. W. Garrett, *Contributions Toward Real-Time Services on Packet Switched Networks*, Ph.D. thesis, Dept. of Electrical Eng., Columbia Univ., 1993.

[27] A. Ortega, "Optimal rate allocation under multiple rate constraints," in *Data Compression Conference*, Snowbird, Utah, Mar. 1996.

[28] T. Cover and J. Thomas, *Elements of Information Theory*, John Wiley & Sons, 1991.

List of Figures

Author Biographies

Zhourong Miao received the B.S. degree in Electrical Engineering from Shanghai Jiao Tong University, Shanghai, China, in 1996 and M.S. degree in Electrical Engineering from the University of Southern California, Los Angeles, CA, in 1999. He is currently pursuing Ph.D degree in the Electrical Engineering-Systems department at the University of Southern California. He worked in IBM T. J. Waston Research Center and Microsoft Research Center as Intern in the summer of 1999 and 2000, respectively. He is currently a Research Assistant in the University of Southern California. His research interests includes multimedia compression and processing, multimedia communication and streaming, proxy caching for streaming video and multimedia networking.

Antonio Ortega was born in Madrid, Spain, in 1965. He received the Telecommunications Engineering degree from the Universidad Politecnica de Madrid (UPM), Madrid, Spain in 1989 and the Ph.D. in Electrical Engineering from Columbia University, New York, NY in 1994. He was a research assistant at the Image Processing Group at UPM (1990). At Columbia he was a graduate research assistant at the Center for Telecommunications Research (1991-94) and was supported by a scholarship from the Fulbright commission and the Ministry of Education of Spain.

In September 1994 he joined the Electrical Engineering-Systems department at the University of Southern California, where he is currently an Associate Professor. At USC he is also a member of the Integrated Media Systems Center (IMSC), an NSF Engineering Research Center, and the Signal and Image Processing Institute. In 1995 he received the NSF Faculty Early Career Development (CAREER) award. He is a senior member of IEEE, and a member of SPIE and ACM. He has been a member of the IEEE Signal Processing Society Multimedia Signal Processing (MMSP) technical committee and is currently a member of the Image and Multidimensional Signal Processing (IMDSP) technical committee. He was the technical program co-chair for the 1998 Workshop on Multimedia Signal Processing and is the technical program co-chair of ICME 2002. He is also a representative of the IEEE Signal

Processing Society to the ICME Steering Committee. He has been an Associate Editor for the IEEE Transactions on Image Processing (1996-2000) and is currently an Associate Editor for the IEEE Signal Processing Letters. In 1997 he was the recipient of the USC School of Engineering Northrop-Grumman Junior Research Award. He received the 1997 IEEE Communications Society Leonard G. Abraham Prize Paper Award for a paper co-authored with C.-Y. Hsu and A. R. Reibman which appeared in the IEEE Journal on Selected Areas in Communications in Aug. 1997. He has also received the IEEE Signal Processing Society 1999 Magazine Award for a review paper co-authored with Kannan Ramchandran that appeared in November 1998.

His research interests are in the areas of image and video compression and communications. They include topics such as adaptive methods for image/video coding, joint source-channel coding for robust video transmission, rate control and video transmission over packet wired or wireless networks. He has over 100 publications on these topics in international conferences and journals.

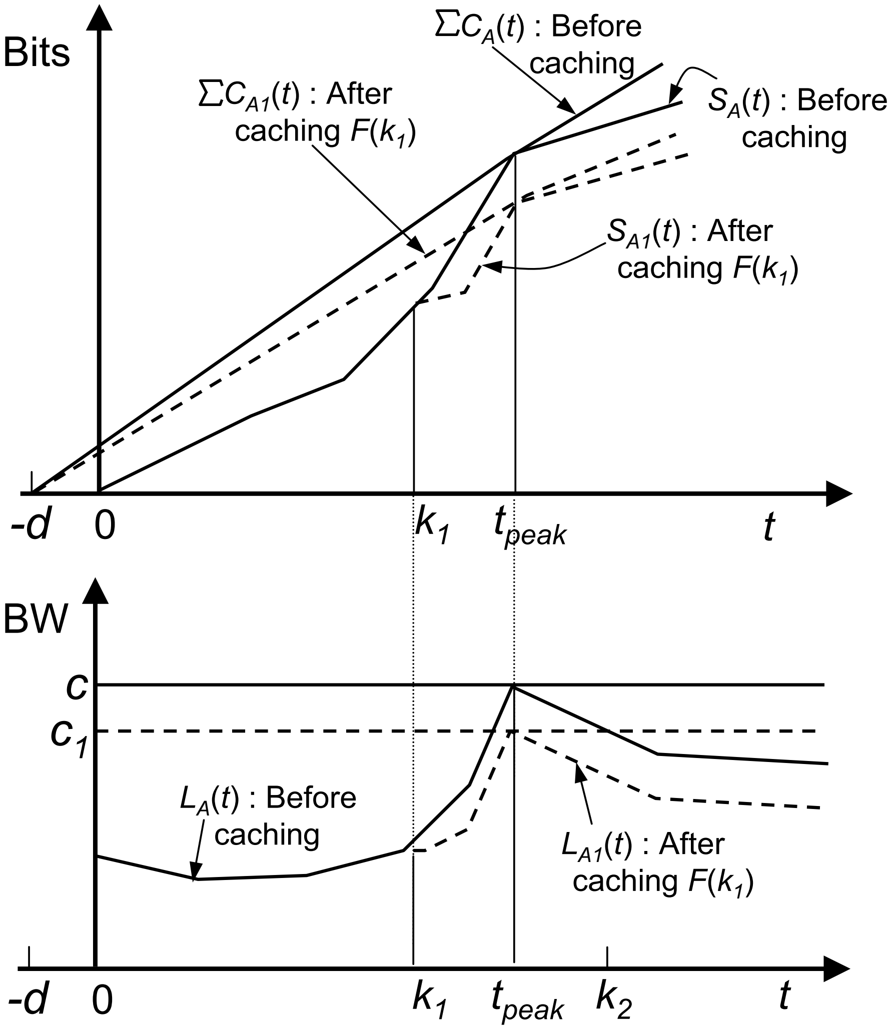

Fig. 3. Illustration of Proposition 1. The lower figure shows the slope functions before and after caching one frame $F(k_1)$, represented by $L_{\mathcal{A}}(t)$ (solid line) and $L_{\mathcal{A}_1}(t)$ (dashed line), respectively. The upper figure is the corresponding cumulative channel/frame rate functions. After caching an "earlier" frame before t_{peak}, i.e., $k_1 \leq t_{peak}$, $\max\{L_{\mathcal{A}}(t)\}$ reduces from c to c_1. Obviously, caching a "later" frame $F(k_2)$, i.e., $k_2 > t_{peak}$, can not reduce $\max\{L_{\mathcal{A}}(t)\}$, which occurs at t_{peak}.

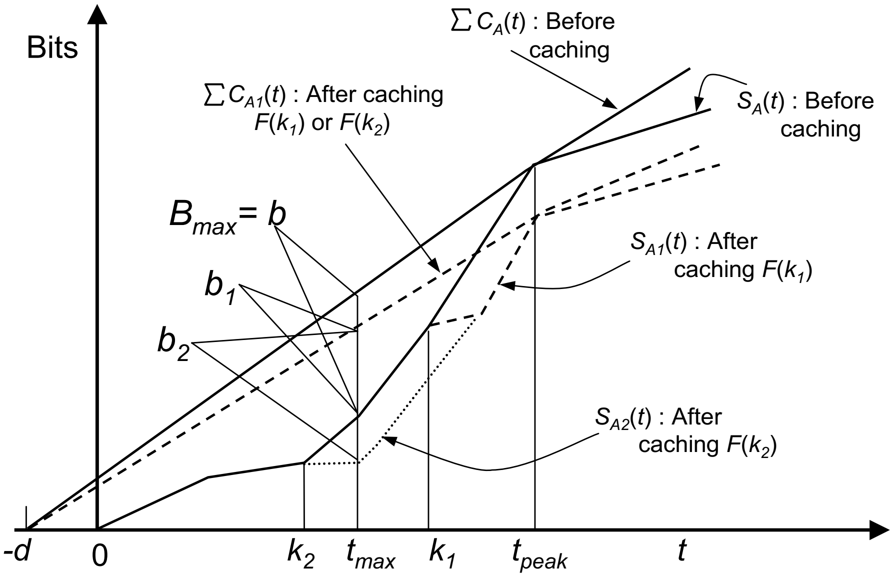
Bits
∑ $C_A(t)$: Before caching
$S_A(t)$: Before caching
∑ $C_{A1}(t)$: After caching $F(k_1)$ or $F(k_2)$
B_{max}= b
b_1
b_2
$S_{A1}(t)$: After caching $F(k_1)$
$S_{A2}(t)$: After caching $F(k_2)$
-d
0
k_2
t_{max}
k_1
t_{peak}
t

Fig. 4. Illustration of caching one frame before or after t_{max}. $S_{\mathcal{A}_1}(t)$ and $S_{\mathcal{A}_2}(t)$ are the cumulative frame rate functions after caching $F(k_1)$ and $F(k_2)$, (drawn in dashed and dotted lines) respectively. Note that only *one* frame is selected in each case. If $R(k_1) = R(k_2)$ and $k_2 < t_{max} < k_1 < t_{peak}$, Proposition 1 shows that selecting $F(k_1)$ or $F(k_2)$ leads to the same reduction on bandwidth C_r. However, the corresponding changes in B_{max} are different: caching $F(k_1)$ reduces B_{max} from b to b_1 ($\Delta b = b_1 - b < 0$); while caching $F(k_2)$ increases B_{max} from b to b_2 ($\Delta b = b_2 - b > 0$). Therefore caching a frame between t_{max} and t_{peak} (e.g., $F(k_2)$) can reduce B_{max} while keeping the same reduction on C_r.

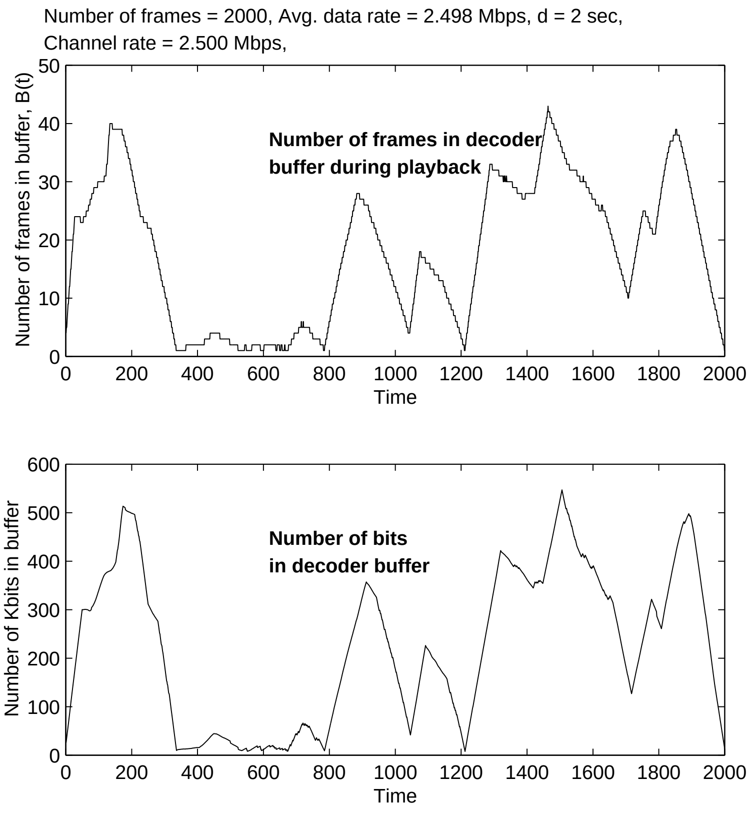

Fig. 5. Trace of client buffer size in number of frames and bits.

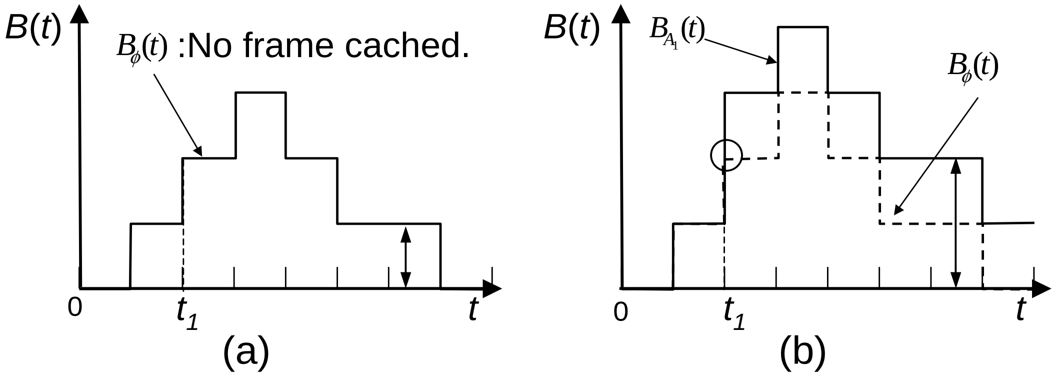

Fig. 6. (a): $B_\phi(t)$, no frame is cached. (b): $B_{\mathcal{A}_1}(t)$, frame $F(t_1)$ is cached. These figures illustrated that by caching one frame $F(t_1)$, the buffer trace can be "lifted" for $t \geq t_1$, when there is no buffer size limitation.

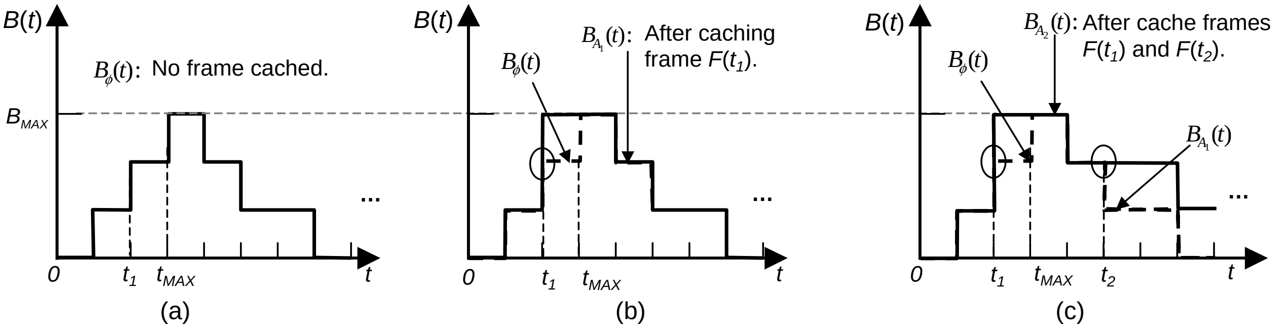

Fig. 7. The client buffer size limitation is B_{MAX}. (a): $B_\phi(t)$, no frame is cached. (b): $B_{\mathcal{A}_1}(t)$, after $F(t_1)$ is cached. (c): $B_{\mathcal{A}_2}(t)$, after both $F(t_1)$ and $F(t_2)$ are cached. With the buffer size limitation, the buffer trace after caching frames will not follow that in Fig. 6. (b) shows that caching frame $F(t_1)$ only increase $B_{\mathcal{A}_1}(t)$ between $t_1 \leq t \leq t_{max}$. (c) shows that caching frame $F(t_2)$ can lift buffer trace for all $t \geq t_2$ because there is no maxima point after t_2.

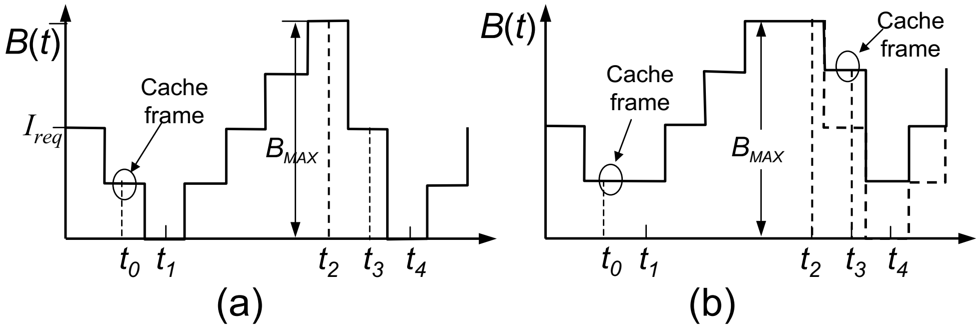

Fig. 8. (a): Trace where only I_{req} is cached and troughs occur at time t_1 and t_4. (b): After SCQ caching. First select frames before t_1, but t_4 still remains the same, due to the maximum peak at t_2, drawn in dotted line. Next select frames within $[t_2, t_4]$ to increase robustness for t_4.

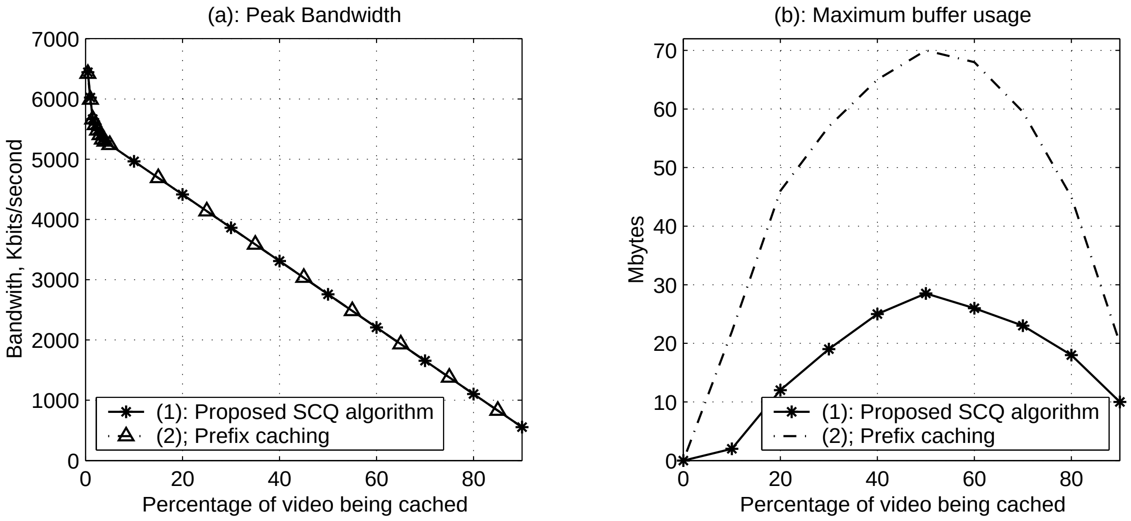

Fig. 9. (a) The bandwidth ($C_r(\mathcal{A})$, see (7)) that has to be reserved, v.s. the percentage of the video been cached. (a): Both the proposed SCQ and prefix caching can reduce $C_r(\mathcal{A})$ similarly as more portion of the video has been cached. (b): The maximum buffer size, B_{max}, required at the client to achieve the caching performance in (a).

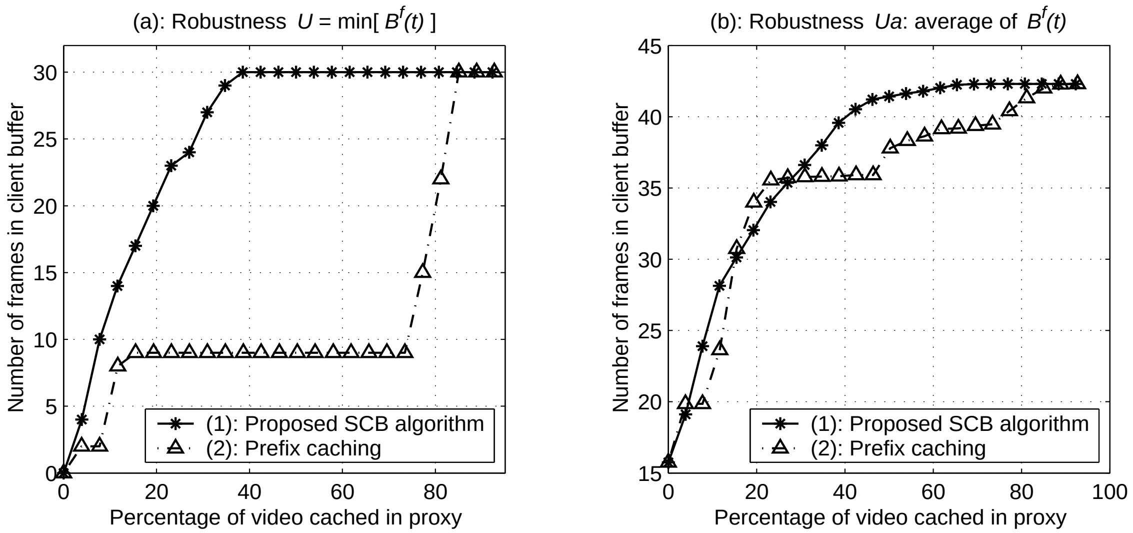

Fig. 10. Robustness vs the percentage of the video been cached, using SCB and prefix caching methods. (a) Robustness U defined in (17). (b) Robustness U_a defined in (18).

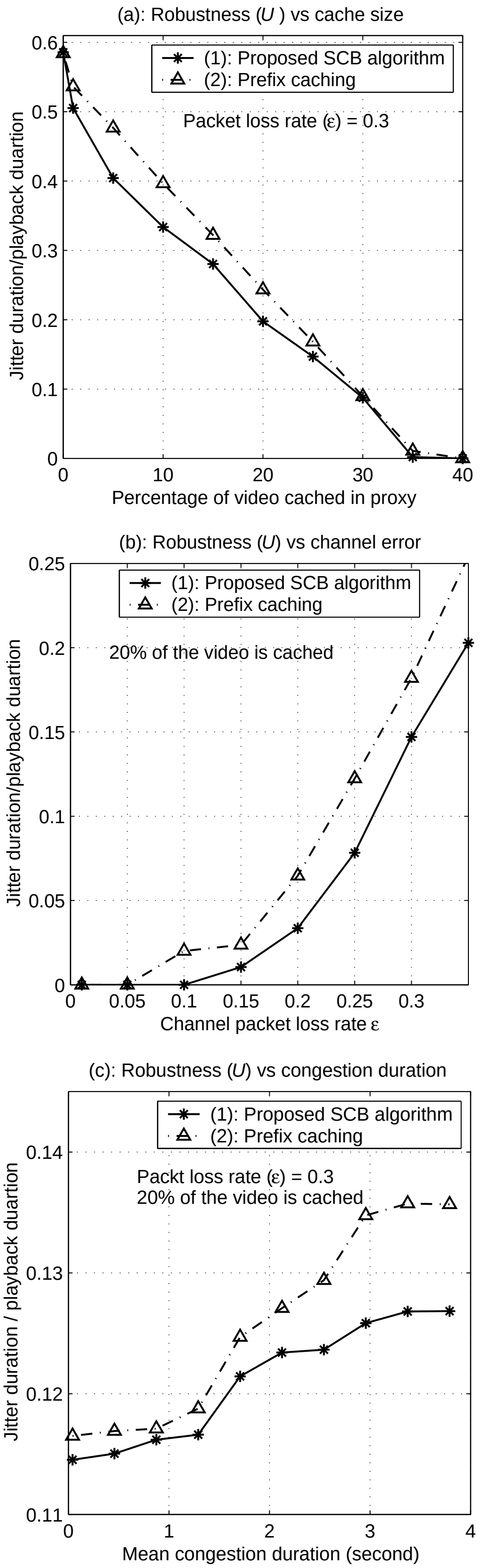

Fig. 11. Robustness verification, with 1000 realizations used in each case. (a): T_J/T_V vs percentage of the video being cached. (b): T_J/T_V vs channel error ϵ, when 20% of the video is cached. (c): T_J/T_V vs average channel congestion duration d_c.