An Interactive Video Delivery and Caching System Using Video Summarization

Sung-Ju Lee, Wei-Ying Ma, and Bo Shen

Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304-1126
{sjlee, boshen}@hpl.hp.com; wyma@microsoft.com

Abstract

With the advance of high-speed network technologies, the availability and popularity of streaming media content over the Internet has grown rapidly in recent years. The delivery and caching of streaming media must be handled in a different fashion than that of traditional non-streaming objects such as HTML or image files, because of its distinct characteristics and user viewing patterns. We propose a novel scheme that provides users with the video summary (a number of key-frame images) before they download the file, and options for them to select the starting playback position. We introduce the content analysis service to achieve these functionalities. The video content analysis performs shot boundary detection, key-frame selection, and face detection and tracking. The results of the processing are a segmented video sequence and an XML-based meta-data describing the video content. We also design a caching system that utilizes our video abstraction and summarization technique. Our integrated video delivery and caching system combines content-aware segmentation, prefix caching, prefetching, and cooperative caching. We describe how our scheme can be applied in three proposed caching architectures.

Keywords: streaming media delivery, web caching, content distribution networks

1 Introduction

With the popularity growth of the Internet and the wide availability of high-speed network access, an increasing number of streaming media objects are being distributed over the Internet. Compared with just a few years ago, larger files are found on the web today because of improved video resolution and longer video length. Class lectures, news reports, sports highlights, movie trailers, commercial ads, and personal home videos are just a few examples of videos on the web.

Conventional video streaming systems use a linear playback scheme that forces users to download from the beginning of a video. Users often need to download and view at least a portion of the video to decide if the content is what they expected. Even during the playback, users may wish to skip some of the parts and jump directly to a specific scene to save time or network bandwidth. Although fast-forward and rewind functionalities are provided by some streaming media servers, long delays and processing are experienced. Moreover, without the knowledge of what is contained in the video and the precise position of a video shot boundary, users usually fail to promptly locate the desired scenes they wish to watch. These problems and limitations have considerably affected user viewing experience and wasted network resources. A recent study [2] reports that almost half of the video requests stopped during near the beginning of the playback. This result indicates that an improved video delivery scheme needs to be developed to manage the user video browsing behavior.

Web caching has been used to accelerate the delivery of web objects such as HTML files and images. Streaming video objects differ from these web objects in several ways. First, the size of video files is usually larger than non-streaming files by orders of magnitude. Storing the entire video file in a single proxy cache is therefore inefficient or even impossible. Second, video objects are mostly static contents with the WORM (Write Once Read Many) property. Hence, the cache consistency and coherency are not important issues in video caching. Moreover, user video access behavior and streaming media workload show different characteristics than those of non-streaming objects, as reported in [2] and [6]. Because of the special characteristic of streaming videos, a more suitable caching system that is different from traditional proxy caching systems must be developed.

We propose an integrated interactive video delivery and caching system that provides users with a better viewing environment. The main goal of our work is to design a video system that supports and utilizes the automatic video analysis and summarization technique. The video analysis performed in our system includes shot boundary detection and key-frame selection. When a client requests a video, our system first shows the key-frames that summarize the video to the user, instead of streaming the video from the beginning. Users can quickly browse through the summary images to decide if they want to download any portion of the video. This key-frame provision helps the users avoid wasting time and network bandwidth on the video they are not interested in. Our system also enables users to easily select and jump to a scene. When a user decides to download the video after viewing the summary, the user can choose the starting playback position by selecting a key-frame. The clients can watch the video immediately from the segment they select. This feature is similar to chapters and "jump to a scene" features of a DVD player. The key difference is that on DVD this information is created by content providers. In today's Internet, the content providers rarely offer this information, and hence we designed a system that makes video analysis and interactive video delivery a value-added service to content providers and end users.

A video caching system is designed to take advantage of the high-level information extracted from the video analysis. We utilize several well known caching techniques such as prefix caching, prefetching, and cooperative caching with the assumption that the video is segmented and spread across multiple proxies for cache sharing and cooperative caching. Segmentation facilitates the distribution and balance of server load, and allows the usage of a fine grain replacement algorithm. Prefix caching [23] is a scheme that stores only the beginning of the video to minimize storage. We apply this technique for each video segment to reduce the start-up latency. Prefetching [15] is performed at the client cache of our system to also reduce the delay. We propose three cooperative proxy architectures and illustrate the operation of video delivery and caching in each of the architectures.

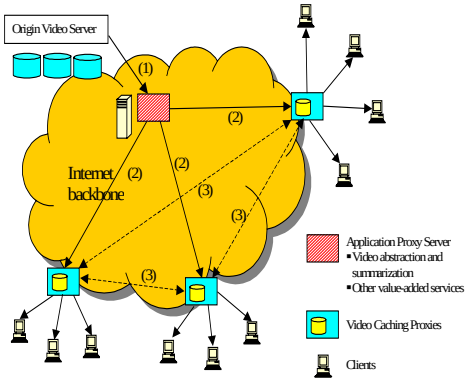
The remainder of this paper is organized as follows. Section 2 provides the overview of our system architecture. The video analysis technique is presented in Section 3, followed by video delivery and caching description in Section 4. We report the current status in Section 5, and conclude the paper in Section 6.

2 System Architecture Overview

Figure 1 depicts the overview of our proposed video delivery and caching system. Video is analyzed by an application proxy server, which can be installed by content providers as a gateway in front of their content servers or deployed by a third-party service provider that has the authorization to process the video on behalf of the content providers. In either approach, we assume a trust relationship between the application proxy server and the origin server, so the video is processed and value-added before it is distributed on the Internet.

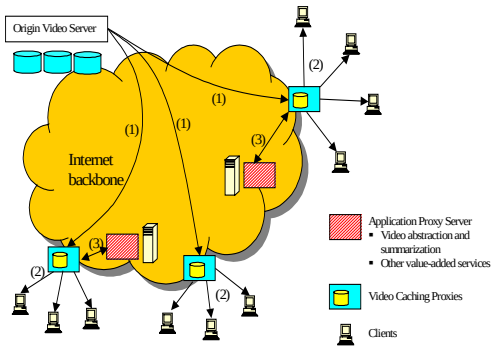
This architecture matches well with the business model of content distribution networks, with the video summarization and interactive delivery being provided as a value-added service to their customers, i.e., content providers. Note that this service is an off-line procedure performed before the video is replicated and distributed to the caching proxies at the edges of networks.

[11] proposed another new layer of Internet infrastructure, called a content services network, this is built around content distribution networks. This layer consists of a network of cooperating application proxy servers that provide computational resources and value-added services to content providers or end users. The system interaction described in Figure 1 corresponds to the pre-distribution service performed by a content services network on behalf of content providers as described in [11].



(1)	Video is transferred from the origin server to the application proxy server that analyzes video content and performs value-added services. This application proxy server may be installed (and owned) by content providers as a gateway, or provided by a third-party content services provider that has the authorization to process the content on behalf of content providers.
(2)	The segmented video and the associated meta-data resulted from the services of application proxy server are distributed to video caching proxies at the edge of the Internet. This may be a service provided by content distribution networks.
(3)	Video is served from a caching proxy close to the client. In case of a local cache miss, it may communicate with other cooperative caches for a possible cache hit.

Figure 1: The system overview. The application proxy server is located near the origin server and performs content service on behalf of a content provider.



(1)	Assuming the video is served from the origin server, the caching proxies, installed within local area networks by ISPs or corporations, will cache the video after it has been requested by a local client.
(2)	Because of the processing delay, the first client requesting the video does not receive the value-added video delivery services.
(3)	While the video is being cached, the caching proxies can send it to a cooperative application proxy server for video summarization and other value-added services. The subsequent client requests for the same video will receive the benefit of the services.

Figure 2: The system overview. The application proxy servers are located near the clients and perform content service on behalf of end users.

Figure 2 shows the overview of another possible deployment scenario, where the application proxy server performs the video summarization on behalf of the clients. The application proxy server only processes the video that has been requested by a local user. After the video is downloaded, the caching proxy sends it to the application proxy server for video summarization and other content services. The video summarization could also be conducted while the video is being downloaded since our video analysis algorithm does not require the video to be fully downloaded in advance. Note that to perform summarization of the whole video, the caching proxy must continue to download the entire video even if the first user that requested the video object halts in the middle of the session. The limitation of this deployment scenario is that because of the processing delay, the first user is not able to receive the value-added services provided by the application proxy server, and the video is served in the form the origin server provides. This problem is similar to that of proxy caching where the client that first requests an object does not receive the benefic of caching.

The system interaction described in Figure 2 corresponds to the post-distribution service performed by a content services network on behalf of end users as described in [11].

3 Content Analysis Services

The video content analysis performs shot boundary detection, key-frame selection, and face detection and tracking. The results of the processing are a segmented video sequence and an XML-based meta-data describing the video content. In this section, we discuss the video parsing techniques used in our application proxy server. In our study, we assume the streaming video files are of MPEG-1 or MPEG-2 format. Since the principle encoding schemes for RealNetworks Media, Microsoft Windows Media, and Apple QuickTime are proprietary information and not released to the public, our algorithm has not been applied to those media formats. For a more detailed description of our video parsing technique, readers are referred to [12].

3.1 Shot Boundary Detection and Key-frame Selection

In order to partition a long video sequence into smaller and meaningful components, we apply shot boundary detection to identify the discontinuities between different shots. Each shot

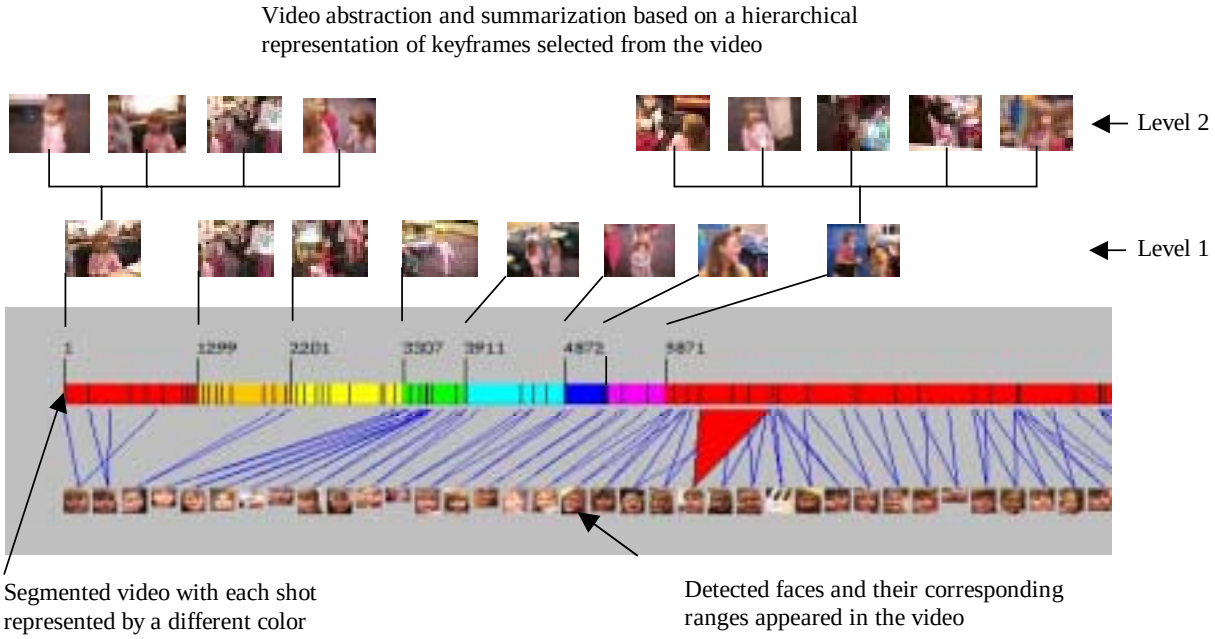


Figure 3: The results of video content analysis. The video is segmented into shots that can be managed separately. The key-frames from each shot are selected and clustered to form a hierarchical representation that is used to assist video delivery and caching.

corresponds to a sequence of frames recorded contiguously and represents a continuous action in time and space. We have developed a very efficient and robust shot boundary detection scheme that operates directly on MPEG compressed data at real-time processing speed [12]. The scheme first looks for a potential shot boundary within a group-of-pictures (GOP) by comparing the difference between two consecutive *I*-frames based on their DCT (Discrete Cosine Transform) information. A typical MPEG video sequence may have a structure of *IBBPBBPBB* (first GOP) *IBBPBBPBB* (second GOP), … When a potential shot boundary is detected, a further examination is applied to all the *B*- and *P*-frames in-between to identify the exact location of a shot boundary. This examination only requires partial decompression of the video to retrieve the macro-block modes. Note that *P*-frame uses its preceding *I*- or *P*-frame as a reference to perform motion compensation, and B-frame uses both preceding and succeeding *I*- or *P*-frames as a reference. By checking the degree of such references from their macro-block modes, we can locate the precise shot boundary.

The video is summarized through the selection of key-frames that represent the content of each video shot. In our system, key-frame selection is performed along with the video segmentation process. The algorithm selects the first frame in each shot that passes the image blur test as a key-frame. It finds a new key-frame by continuously searching for the next *I*-frame that is sufficiently different from the previous key-frame. The *I*-frames are compared with one another using the color information extracted from the frames. The required degree of difference for key-frame selection can be adjusted in the algorithm to control the approximate number of key-frames generated. Note that to be selected as a key-frame, each key-frame must satisfy a certain requirement to ensure it is not blurred [13]. Key-frames from each shot are clustered to form a hierarchical representation that provides multiple levels of granulations for video browsing and navigation [30]. Figure 3 shows the results of video content analysis.

3.2 Face Detection and Tracking

People are often the most important objects in a video. Hence, the knowledge of who is in the video and in which part of the video they appear can make video delivery and client navigation more effective and efficient. We have incorporated face detection [22] and tracking techniques into our video parser. Face detection is applied on every *I*-frame and the location of detected faces is matched with the face locations currently under tracking. If a match is identified, the new location of a tracked face is updated and the detected face is labeled as the same person as the tracked one. Whenever there is an uncertainty for identifying a newly detected face or there is no match for a current tracked face, the face tracking is applied. The face tracking uses the motion information contained in the macro-blocks of *P*- and *B*-frames to predict face locations from the previous *I*-frame to the next *I*-frame. Because face detection algorithm only detects frontal faces in scattered *I*-frames, face tracking is essential to identify the correspondence between these detected faces and associate them to the same person within a continuous video shot. The tracking stops when the projected face region leaves a scene or a shot boundary is detected.

Face detection is also used to assist the selection of key-frames. When a new face is detected, the corresponding frame is selected as a key-frame. In addition, for a set of similar frames, face information is used to help select the best key-frame. That is, a frame that contains larger and better-positioned frontal faces with more people appearing inside will replace the previously selected key-frame if their contents are similar.

3.3 Meta-data for Video Content Description

The results of video content analysis are stored in a meta-data whose format is shown in Figure 4. This meta-data uses XML-based language to describe the temporary structure and

summarization of video. In this example, the video has been segmented into 13 shots, with each shot represented by a key-frame URL in the first-level cluster denoted by <kf_cluster level = "1">. The range of each video shot is represented by *begin* and *end* tags. The seventh video shot, represented by kf-0000685.jpg, has more than a key-frame because it contains rapid scene changes. Those key-frames are put into the second-level cluster denoted by <kf_cluster, level = "1.7"> for further detailed browsing if necessary. The meta-data also contains the information about the coordinates of detected faces and their corresponding ranges in the video. The URLs for the video and a possible logo to be inserted are also included in the representation.

This meta-data serves two purposes. First, upon the client's request, the server sends this meta-data along with key-frame images to the client. The media player uses this information to render the video so that the user can browse the summary in advance of downloading the video file. In most scenarios, the summary contains enough information for the user to decide whether to download. If users find the summary (key-frames) uninteresting or do not see the expected content, they can decide not to download the file. In the case of using other existing applications, users must initiate the streaming session in order to view the content. When users do not appreciate the content and decide to stop in the middle of a session, bandwidth and power are wasted just to decide whether users want to access the video file. A study in [2] showed that nearly 45% of video file requests stopped the session during the first 5% of the video playback period. Based on this user access behavior, we believe providing users with the video summary can conserve network resources. Second, using the information provided in the meta-data, the client could communicate with the server to control how the video is streamed to the client. When the clients browse the summary (key-frames), they can select any particular frame that interests most. By clicking the specific key-frame, the client can view the video starting from the segment represented by the selected key-frame. Therefore, users can watch the interesting parts of the video without having to stream it from the beginning of the entire video. Note that each key-frame (or face) in the meta-data represents a video segment that can be played back independently and immediately. This requires a special handling on the server to ensure that the beginning of each video segment corresponds to a breakable point in the compressed video data. For MPEG-1 and MPEG-2 videos, this point is the beginning of an *I*-frame, as it does not depend on any previous frames to decode it.

4 Video Delivery and Caching

4.1 Goals and Design Overview

Streaming media have different characteristics than non-streaming objects. Video files are much larger and their contents do not change. User access behavior and workload of video files and streaming media are shown to be different from those of non-streaming objects [2], [6]. Based on these characteristics of streaming video files, we take an approach that is different from traditional proxy caching systems. The main goal of our work is to design an integrated video delivery and caching system that supports and utilizes the content analysis and summarization technique introduced in Section 3. The key technical components of our system, which we discuss below, are segmentation, prefix caching, prefetching, and cooperative caching.

```
<video structure>
   <kf_cluster level="1">
      <img src="kf-0000132.jpg" begin="132" end="227"/>
      <img src="kf-0000228.jpg" begin="228" end="336"/>
      <img src="kf-0000354.jpg">
         <face coords="179,69,37,41">
            <playback begin="337" end="385"/>
         </face>
         <playback begin="337" end="385"/>
      </img>
      <img src="kf-0000386.jpg" begin="386" end="458"/>
      <img src="kf-0000459.jpg" begin="459" end="576"/>
      <img src="kf-0000577.jpg" begin="577" end="684"/>
      <img src="kf-0000685.jpg" begin="685" end="764"/>
      <img src="kf-0000765.jpg" begin="765" end="800"/>
      <img src="kf-0000801.jpg" begin="801" end="852"/>
      <img src="kf-0000885.jpg" begin="853" end="894"/>
      <img src="kf-0000912.jpg" begin="895" end="950"/>
      <img src="kf-0000951.jpg" begin="951" end="992"/>
      <img src="kf-0000993.jpg" begin="993" end="1008"/>
   </kf_cluster>

   <kf_cluster level="1.7">
      <img src="kf-0000717.jpg" begin="717" end="764"/>
      <img src="kf-0000762.jpg" begin="762" end="764"/>
   </kf_cluster>

   <video src="rtsp://www.hpl.hp.com/test.mpg"/>
   <logo src="hp-logo.jpg" coords="6,6"/>
<video structure/>
```

Figure 4: An XML-based meta-data describing the information about the video.

4.1.1 Content-aware Segmentation

A video segmentation module divides the video into segments based on the result of shot boundary detection. Each segment corresponds to a continuously recorded sequence of frames and we can manage each segment separately. The segments of a video are spread across multiple proxies for cache sharing and cooperative caching. Note that in contrast with the simple equal-sized segmentation technique used in [1] and [10], the size of segments may vary in our system. It is desirable however, to have constraints on segment lengths, as having too many or too few segments may cause difficulties in object management.

Segmenting the streaming media files has several advantages. By placing segments on several proxy caches, we can better distribute the load. In addition, spreading the video into several caches allows us to perform intelligent caching. Instead of taking an "all or nothing" approach where either the video file is stored in its entirety or not cached at all, we cache the streaming objects by segments, i.e., the cache replacement granularity is the segment. Therefore, at some instances, only popular parts of the video may be present at the caches of the network edges. We discuss the caching algorithm we use in Section 4.5.

4.1.2 Prefix Caching

Storing the entire video file into proxies may be inefficient as video files can be very large. This is particularly true if the cached streaming media object is not popular. Prefix caching [23] stores only the first few frames of the popular video files. When a client requests a video stream and the prefix is cached, the proxy delivers the prefix to the client while it requests the remainder of the video to the origin server. By caching the (large enough) prefix, the start-up latency perceived by the user can be reduced. The challenge could be in determining the optimal prefix size. Items such as roundtrip delay, server-to-proxy latency, video specific parameters (e.g., size, bit rate, etc.), and retransmission rate of lost packets

must be considered in calculating the appropriate prefix length [23].

As our system has multiple segments for a video object, we adopt prefix caching for each cached segments. When users click one of the summary frames provided by our content analysis and summarization technique, our system delivers the specific segment immediately to the client (and not from the beginning of the video file, unless the user requested the first segment). Hence, we store the prefix of each segment to benefit from prefix caching in our system.

4.1.3 Prefetching

Prefetching [4], [15], [31] is another scheme to reduce the user perceived latency. Prefetching can increase the caching efficiency by predicting the client request pattern and caching the objects in advance of user requests. The sacrifice however, is the increase in traffic. Studies report other potential costs of prefetching. [26] shows that prefetching is beneficial only when traffic is lightly loaded and the prediction algorithm is very effective. [7] reports that prefetching affects the queueing behavior and increases network burstiness. Prefetching should be performed with extra cautions in streaming media delivery systems as they are dealing with much larger objects.

We use prefetching at the client cache instead of the proxy cache. Prefetching scheme on the client side further reduces the startup latency. Since the user is browsing the video based on the meta-data already downloaded, this interaction provides a window of opportunity for the client device to initiate a prefetching based on the availability of the resources such as bandwidth and storage in the client cache. The user's browsing behavior provides more clues to help the prediction algorithm to be more effective. We propose the following prefetching algorithm that decides when and which portion of the video is downloaded while browsing. As shown in Figure 5, if the focus window of the user's browser stays at certain section for more than t seconds, the first n bytes of each video segment within the window are prefetched.

The duration t is proportional to the number of key frames showing in the browser window. In addition, the parameter pair (t, n) is designed to satisfy certain constraints posed by the resources limitation. For example, if the connection between the client and proxy is congested, the algorithm can choose to increase t and decrease n in order to reduce the prefetching frequency and payload, respectively.

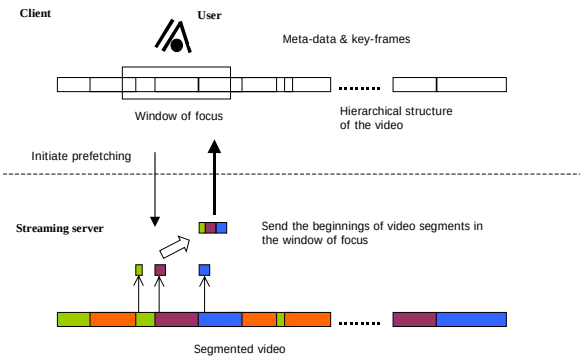


Figure 5: The video prefetching scheme.

4.1.4 Cooperative Caching

Cooperative caching [17], [27], [29] is widely studied and deployed in many content delivery systems, starting from Harvest [5] and Squid [24]. A number of proxies cooperate and share the cache with one another to improve the cache hit rate. When a proxy receives a request to an object that is missing from its cache, it checks whether the object is cached on other proxies within the cooperative network. The most well known schemes are broadcast probing [5], [24], [28], directory service based [8], [9], [21], and hash-partitioned namespace approach [25].

As our system divides the streaming media object into segments, it is natural that our system spreads them across several proxies and uses a web cache sharing technique. Our cooperative caching scheme needs an agent that determines and has the knowledge of which segment is cached on which proxy. The segment assignment policy is important as each segment has different sizes. Note that there are no duplicate segments cached on different proxies within our cooperative caching network.

Our video delivery system can be implemented in various cache sharing architectures. They are centralized, distributed, and hybrid cooperative caching, and none uses a hierarchical cache structure. Each proposed scheme is examined in the following three sections.

4.2 Video Delivery with Centralized Cooperative Caching

In the centralized cooperative caching scheme, a *master cache* coordinates the interactions between the client and the cache array. For each video object, the master cache stores the key-frame and the prefix of each segment. In addition, the master cache assigns and knows the location of each segment, i.e., the information regarding which cache keeps which segment. The request from the client is redirected to the master cache. Upon a request, the master cache provides the key-frames to the client. When the client selects a certain segment, the master cache transmits the proper prefix to the client (and hence reduce the startup delay) while instructing the appropriate proxy, which cached the requested segment, to deliver the segment. The video stream is always delivered to the user through the master cache.

For the illustration of this scheme, we use the following example that is also used in the description of the other two schemes. As shown in Figure 6, the video object is segmented into three sections and each segment is represented by a key-frame. The first segment is cached in the proxy cache C1, the second in C2, and the third in C3. The key-frames and prefixes are stored in the master cache along with the location information of each segment.

Consider a client requests a video and browses through the key-frames. Subsequently, the client requests to watch from the second segment, and continuously views through the third segment towards the end of the video.

The interactions involved in the above scenario are explained as follows:

1. The client requests a video.
2. If the requested video is not in a cache, the master cache sends a request to the origin server. The video object is segmented and summarized in key-frames by the content analysis service (this process is performed by the application proxy server, as described in Section 2). The

master cache hosts the key-frames and prefixes and distributes the segments to appropriate proxy caches. It then sends the video summary to the client. Note that if the requested video is already in the cache, the master cache simply provides the video summary to the client.

3. After browsing the summary, the client selects the segment #2 as the starting playback position.
4. The master cache sends the prefix of segment #2 to the client and requests the rest of the segment from C2.
5. C2 sends segment #2 to the master cache.
6. The master cache forwards segment #2 to the client.
7. Right after the completion of delivering segment #2, the master cache sends the prefix of segment #3 to the client and requests the rest of the segment from C3, similar to step 4.
8. C3 sends the segment #3 to the master cache.
9. The master cache forwards the segment #3 to the client.

Note that the length of prefix stored by the master cache must be large enough to ensure the client does not experience delays between viewing the prefix and the rest of the segment. The segment assignment and distribution must be handled with care by the master cache. If the number of segmented portions is smaller than the number of proxies, segment distribution to overloaded caches should be avoided. On the other hand, if there are more video segments than the caches, intelligent assignment must be performed. For example, consider there are four segments to be spread across three proxy caches, and the second cache is the least loaded. Instead of assigning the segment #1 to cache #1, segment #2 to cache #2, segment #3 to cache #3, and segment #4 to cache #2, we can distribute the segment #1 to cache #1, segments #2 and #3 to cache #2, and segment #4 to cache #3. As users that watch segment #2 are likely to access segment #3 next as they are continuous, this assignment policy can reduce coordination processing and the control traffic between the master cache and proxies.

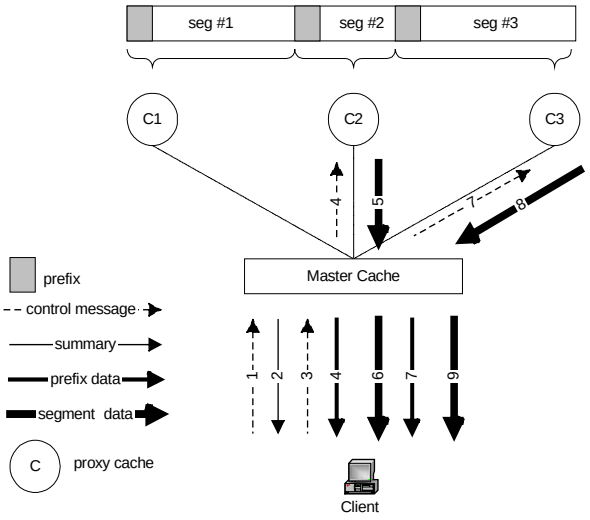


Figure 6: A video delivery example in centralized cooperative caching.

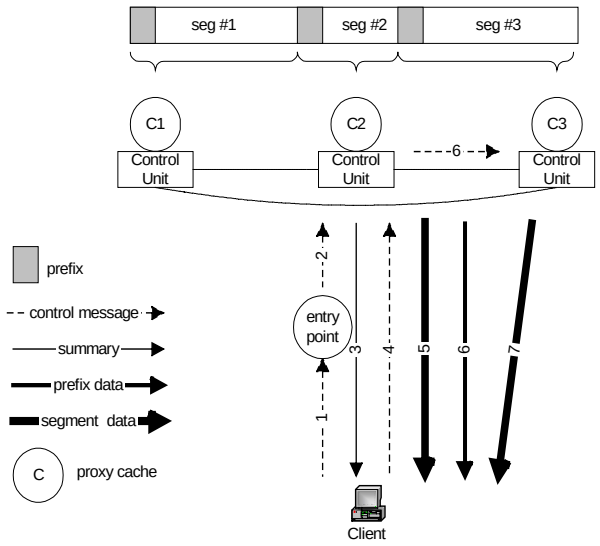


Figure 7: A video delivery example in distributed cooperative caching.

The centralized cooperative cache scheme is simple and easy to implement and manage. No communication is needed between proxy caches as the master cache holds all necessary information. Since every object and stream goes through the master cache, however, it can easily be overloaded and congested. In addition, the master cache creates the single point of failure problem.

4.3 Video Delivery with Distributed Cooperative Caching

A master cache does not exist in the distributed architecture. Instead, each proxy cache has a control unit that has the functionality of the master cache. In this architecture, the summary key-frames and the prefix of each segment are replicated and cached in each proxy. Each cache also holds the location information of all the segments. The proxies must communicate with each other to exchange this information. Schemes such as summary cache [8] or cache digest [21] can be used for this purpose.

As shown in Figure 7, the client's initial request is redirected to the *entry point*. The entry point serves as a gateway to the client and it could be as simple as a forwarder. The basic function of the entry point is to select a proxy that coordinates the caching and delivery of a video object for each request. The election algorithm can be a round robin mechanism, for example. When this proxy receives a request to the segment that it does not hold, it sends the prefix of the requested segment to the client and forwards or redirects the request to the appropriate proxy.

Let us study the systematic process of this architecture using the same scenario as in Section 4.2.

1. The client requests a video.
2. The entry point receives the request and forwards it to the selected proxy (C2 in this example).
3. If the C2 does not have the requested video, it sends a request to the origin server. The video object is segmented and summarized in key-frames by the content

analysis service. C2 distributes the segments to appropriate proxy caches and sends the key-frames and prefixes of the video to all the proxies. It then delivers the video summary to the client. Note that if the requested video is already in the cache, C2 simply provides the video summary to the client.

4. After browsing the summary, the client selects segment #2 as the starting playback position.
5. As C2 has cached segment #2, it delivers the entire segment to the client.
6. Right after the completion of delivering segment #2, C2 sends the prefix of segment #3 to the client and sends a request to C3 to deliver the rest of the segment to the client.
7. C3 sends segment #3 directly to the client.

In this architecture, all possible problems of having a master cache are eliminated. The cost is the added complexity for each proxy. Streaming server must be implemented in each control unit, as each proxy streams the video content to the client. Each proxy cache cannot hold as many video segments as in the centralized architecture since the storage is also used for caching the prefixes and key-frames. In addition, session handoff scheme must be implemented among the proxy caches. A simple entry point is also needed to handle the initial request from the client.

4.4 Video Delivery with Hybrid Cooperative Caching

In the centralized cooperative caching, the bulk of the video content is delivered through the master cache. This easily overloads the master cache. In the distributed scheme, each proxy needs added capability and coordination can be complex. In this section, we propose a hybrid architecture where a master cache is used, but its role is reduced to only controlling the interactions and delivering the key-frames and prefixes. The delivery of the video segments to the client is done directly by the proxy caches.

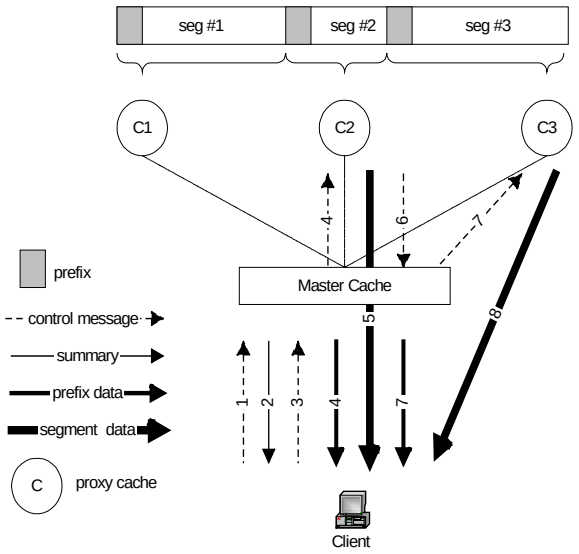


Figure 8: A video delivery example in hybrid cooperative caching.

Figure 8 shows the interactions using the example scenario.

1. The client requests a video.
2. If the requested video is not in a cache, the master cache sends a request to the origin server. The video object is segmented and summarized in key-frames by the content analysis service. The master cache hosts the key-frames and prefixes and distributes the segments to appropriate proxy caches. It then sends the video summary to the client. Note that if the requested video is already in the cache, the master cache simply provides the video summary to the client.
3. After browsing the summary, the client selects segment #2 as the starting playback position.
4. The master cache sends the prefix of segment #2 to the client and sends a request to C2 to deliver the rest of the segment to the client.
5. C2 sends segment #2 directly to the client.
6. Towards the end of sending of segment #2, C2 informs the master cache with the status.
7. The master cache sends the prefix of segment #3 to the client and sends a request to C3 to deliver the rest of the segment to the client, similar to step 4.
8. C3 sends segment #3 directly to the client.

Since the bulk of the video is delivered directly from the proxy caches to the client, the master cache is less prone to overload, congestion, and failure. The master cache is mainly in charge of the control message exchanges. The connection handoff however, must be resolved as the clients request is sent to the master cache but the data delivery is from the proxy caches. Depending on the communication protocol used in the video access, the master cache should be able to hand the data delivery duty to the proxy caches. For example, if the communication is UDP based, the master cache can forward the client address with the request (e.g., in steps 4 and 7). On the other hand, if the client is involved in a TCP session, we must resolve TCP handoff between the master cache and the proxy caches.

4.5 Cache Replacement Granularity and Policy

As segments of a video are spread across multiple proxies, granularity of our cache replacement scheme is segments. If certain segments of a video are popular, only popular segments are cached. This partial video caching differs from the "all or nothing" approach where the object is either cached in its entirety or not cached at all. A "cache hit" is measured in segments. Using the example scenario from Sections 4.2 to 4.4, when the client selects the segment #2 as the starting playback position and continues to view through segment #3, cache hits are recorded for both segments #2 and #3.

Recent studies [2], [6] indicate that client requests to video files show strong temporal locality. A cache replacement algorithm such as LRU-*k* [14] can exploit temporal locality [1]. When the cache is full and a new item must be inserted, LRU-*k* replaces an object whose *k*-th previous request is the least recent. Hence, the LRU scheme is LRU-*k* with $k = 1$. Each proxy cache of our system runs this LRU-*k* scheme as the cache replacement algorithm.

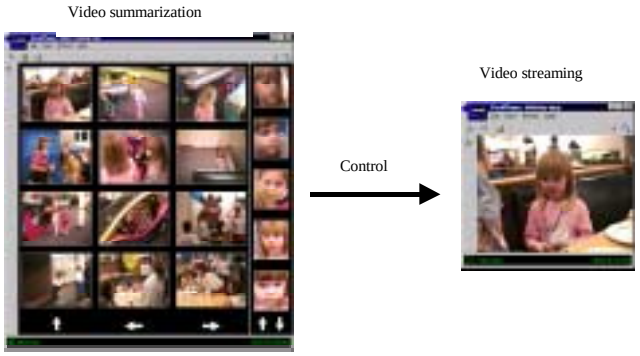
The cache replacement takes place at two locations in our system: the master cache (or control units) and proxy caches. When a video segment is replaced by a proxy cache, the prefix of that segment remains in the master cache (in the centralized or hybrid architectures) or all the control units of proxy caches (in the distributed architecture). If there is a request to the replaced segment, the prefix can be sent to the client while the rest of the segment is requested to the origin server. When all the segments of a video are removed from caches, the key-frames and prefixes can also be deleted. Note that the prefix should not be removed before the actual segment is replaced. When the prefix of the segment does not exist, our system considers that the segment is not cached and hence it will not be accessed.

There are also three levels of replacement: key-frame level, prefix level and segment level. A prefix is always physically stored along with a key-frame. On the other hand, a prefix is logically associated with a segment since it is the initial part of the segment. Since key-frames and prefixes consume less resource, we assign higher priorities to them. In the caching replacement algorithm, object of higher priority is less likely to be replaced. Therefore, a weighted LRU-*k* scheme can be applied considering the different priorities we assign to each level.

5 Current Status

We implemented a prototype system based on the RealNetworks software development kit (SDK) [18]. Several RealSystem plug-ins are implemented to give the end users the control on the playback of a video stream. On the server side, a file-format plug-in is implemented to provide the RealServer the ability to interactively deliver a MPEG video. On the client side, a rendering plug-in is implemented to enable the RealPlayer to render a MPEG video interactively. The file-format and rendering plug-ins use the XML-based meta-data to communicate with each other in the video streaming process. This system allows users to quickly browse through the entire video content by viewing key-frames and faces. Each key-frame and face represents a video segment that can be played back individually according to the user interests. Figure 9 (a) shows the snapshot of this system on a desktop PC. Note that two RealPlayer windows launch when the user clicks on a video that is processed and value-added by our application proxy server: one window shows the hierarchical video summary and the other window plays the video.

To apply our technique to video streaming on a handheld computer with a small form factor, we implemented a video player using the Microsoft Foundation Class (MFC) [16] on the Windows CE platform. The player connects to the video server that is extended from the Apache web server. The player sends an HTTP request to the server with a special header specifying the starting point of the requested video. The server performs *fseek* or switches between different video segment files before packetizing the data. The server has the information about the byte location associated with the beginning of each video segment or where the individual segment is stored. Figure 9 (b) shows a picture of the interactive video delivery to a handheld computer (HP Jornada 720) with a wireless LAN connection. Because the screen is small, only two key-frames are shown each instance. When the user clicks on a particular key-frame, the left window is used to play the video and the right window continues to be used to browse the summary.



(a) Interactive video delivery to a desktop PC.

(b) Interactive video delivery to a handheld computer.

Figure 9: A prototype system is implemented on (a) the SDK provided by RealNetworks System and (b) MFC on Windows CE platform.

6 Conclusions

We presented an integrated interactive video delivery and caching system that utilizes the content analysis services to provide better user interfaces and conserve network resources. The content analysis service performs shot boundary detection, key-frame selection, and face detection and tracking. When a client requests a streaming video file, our system initially provides the video summary to the user. The user quickly browses these summary images to decide whether to download the video. If the user selects to download, the user can also choose which part of the video to download. We believe our summary provision offers better video viewing environment to the users, and we expect user access behaviors to change.

Our video system uses content-aware video segmentation, prefix caching, prefetching, and cooperative caching techniques to enable users to view any part of the cached video without long delays. By segmenting the video, we are able to use a fine grain cache replacement algorithm that exploits temporal locality and multi-level caching. We described the operation of video delivery and caching mechanism on three proposed cooperative caching architectures: centralized, distributed, and hybrid. Our system is designed to utilize the content analysis service that is currently applied to videos of format MPEG-1 and MPEG-2. We believe that our system can also be used for other streaming formats. We can still perform segmentation, prefix caching, and cooperative caching to those video formats, even though we cannot apply content analysis and provide video summary to the users because of proprietary issues.

We currently have a content analysis service prototype implementation on the SDK provided by RealNetworks and MFC on the Windows CE environment. It operates on our desktop PCs and wireless mobile handheld computers. We are continuing our work in several ways. Simulation is in progress to evaluate the performance on three system architectures, various replacement algorithms, and the optimal *k* size in LRU-*k*. Based on the lessons we learn from simulation, we plan to implement our integrated video delivery and caching system on a real test-bed.

References

[1] S. Acharya and B. Smith, "MiddleMan: A Video Caching Proxy Server," *Proceedings of the NOSSDAV 2000*, Chapel Hill, NC, June 2000.

[2] S. Acharya, B. Smith, and P. Parnes, "Characterizing User Access To Videos On the World Wide Web," *Proceedings of the SPIE/ACM MMCN 2000*, San Jose, CA, January 2000, pp. 130-141.

[3] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and Implementation of a Caching System for Streaming Media over the Internet," *Proceedings of the IEEE RTAS 2000*, Washington, DC, June 2000, pp. 111-121.

[4] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, "A Study of Integrated Prefetching and Caching Strategies," *Proceedings of the ACM SIGMETRICS'95*, Ottawa, Canada, May 1995, pp. 188-197.

[5] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A Hierarchical Internet Object Cache," *Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996, pp. 153-163.

[6] M. Chesire, A. Wolman, G. M. Voelker, and H. M Levy, "Measurement and Analysis of a Streaming-Media Workload," *Proceedings of the USITS'01*, San Francisco, CA, March 2001.

[7] M. Crovella and P. Barford, "The Network Effects of Prefetching," *Proceedings of the IEEE INFOCOM'98*, San Francisco, CA, March 1998, pp. 1232-1239.

[8] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *Proceedings of the ACM SIGCOMM'98*, Vancouver, Canada, September 1998, pp. 254-265.

[9] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches," *Proceedings of the IEEE HotOS-VI*, Cape Cod, MA, May 1997, pp. 93-98.

[10] M. Hofmann, T. S. E. Ng, K. Guo, S. Paul, and H. Zhang, "Caching Techniques for Streaming Multimedia over the Internet," *Bell Laboratories Technical Report*, BL011345-990409-04TM, April 1999.

[11] W. Y. Ma, B. Shen, and J. T. Brassil, "Content Services Networks: The Architecture and Protocol," *Proceedings of the WCW'01*, Boston, MA, June 2001.

[12] W. Y. Ma and H. J. Zhang, "An Indexing and Browsing System for Home Video," *Proceedings of the EUSIPCO 2000*, Tampere, Finland, September 2000, pp. 131-134.

[13] X. Marichal, W. Y. Ma and H. J. Zhang, "Blur Determination in the Compressed Domain Using DCT Information," *Proceedings of the IEEE ICIP'99*, Kobe, Japan, October 1999, pp. 386-390.

[14] E. O'Neil, P. O'Neil, and G. Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering," *Proceedings of the ACM SIGMOD'93*, Washington, DC, May 1993, pp. 297-306.

[15] V. N. Padmanabhan and J. C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, July 1996, pp. 22-36.

[16] J. Prosise, *Programming Windows with MFC*, 2nd edition, Microsoft Press, May 1999.

[17] M. Rabinovich, J. Chase, and S. Gadde, "Not All Hits are Created Equal: Cooperative Proxy Caching over a Wide-area Network," *Computer Networks and ISDN Systems*, vol. 30, no. 22-23, November 1998, pp. 2253-2259.

[18] RealNetworks Software Development Kit, http://www.realnetworks.com/devzone/tools/.

[19] R. Rejaie, M. Handley, H. Yu, and D. Estrin, "Proxy Caching Mechanism for Multimedia Playback Streams in the Internet," *Proceedings of the WCW'99*, San Diego, CA, April 1999.

[20] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet," *Proceedings of the IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000, pp. 980-989.

[21] A. Rousskov and D. Wessels, "Cache Digest," *Proceedings of the WCW'98*, Manchester, England, June 1998.

[22] H. A. Rowley, S. Baluja and T. Kanade, "Neural Network-based Face Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, January 1998, pp. 23-38.

[23] S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," *Proceedings of the IEEE INFOCOM'99*, New York, NY, March 1999, pp. 1310-1319.

[24] Squid Internet Object Cache, http://squid.nlanr.net.

[25] V. Valloppillil and K. W. Ross, "Cache Array Routing Protocol v1.0," *IETF Internet Draft*, draft-vinod-carp-v1-03.txt, February 1998.

[26] Z. Wang and J. Crowcroft, "Prefetching in World Wide Web," *Proceedings of the IEEE Global Internet'96*, London, UK, November 1996, pp. 28-32.

[27] Z. Wang and J. Crowcroft, "Cachemesh: A Distributed Cache System for World Wide Web," *Proceedings of the WCW'97*, Boulder, Colorado, June 1997.

[28] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP) version 2," *IETF Request For Comments 2186*, September 1997.

[29] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching," *Proceedings of the ACM SOSP'99*, Charleston, SC, December 1999, pp. 16-31.

[30] H. J. Zhang and W. Y. Ma, "Structured and Content-based Video Browsing," *Proceedings of the 32nd IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 1998.

[31] 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/ACM Transactions on Networking*, vol. 8, no. 4, August 2000, pp. 429-442.