

# Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet

Reza Rejaie<sup>1</sup>, Haobo Yu<sup>1</sup>, Mark Handley<sup>2</sup>, Deborah Estrin<sup>1</sup>

<sup>1</sup>USC/ISI

<sup>2</sup> ACIRI at ICSI

reza, haoboy, estrin@isi.edu    mjh@aciri.org

*Abstract*—The Internet has witnessed a rapid growth in deployment of Web-based streaming applications during recent years. In these applications, server should be able to perform end-to-end congestion control and quality adaptation to match the delivered stream quality to the average bandwidth available. The delivered quality is limited by the bottleneck bandwidth on the path to the client.

This paper proposes a proxy caching mechanism for layered-encoded multimedia streams in the Internet to maximize the delivered quality of popular streams to interested clients. The main challenge is to replay a quality-variable cached stream while performing quality adaptation effectively in response to the variations in available bandwidth. We present a pre-fetching mechanism to support higher quality cached streams during subsequent playbacks and improve the quality of the cached stream with its popularity. We exploit inherent properties of multimedia streams to extend the semantics of popularity and capture both level of interest among clients and usefulness of a layer in the cache. We devise a fine-grain replacement algorithm suited for layered-encoded streams. Our simulation results show that the interaction between the replacement algorithm and pre-fetching mechanism converges the state of the cache to an efficient state such that the quality of a cached stream is proportional to its popularity, and the variations in quality of a cached stream are inversely proportional to its popularity. This implies that after serving several requests for a stream, the proxy can effectively hide low bandwidth paths to the original server from interested clients.

*Keywords*—Proxy Caching Mechanism, Quality Adaptive Video Playback, Layered Transmission, Internet

## I. INTRODUCTION

The explosive increase in commercial usage of the Internet has resulted in a rapid growth in demand for audio and video streaming. This trend is expected to continue and a larger portion of Internet traffic will consist of multimedia streams (i.e., audio and video) in the future. Most current applications involve web-based audio and video playback [8], [11] where a multimedia server maintains a large number of multimedia streams and pipelines a stream to a client upon request. The quality (i.e., bandwidth) of the delivered stream in a client-server approach is limited to the

bottleneck bandwidth between the server and the client. If the server is behind a bottleneck, a client with high bandwidth local connectivity to the network may still receive low quality streams due to congested links somewhere between the point of network attachment and the server. Clearly, if the client has only low-bandwidth connectivity to the network (i.e., the bottleneck is the last hop), the delivered quality can not be improved. However, clients with high bandwidth connectivity expect to receive high quality streams.

This paper explores an adaptive solution to this problem using multimedia proxy caching [16]. A proxy server is a small server that resides close to a group of clients. The required amount of storage (i.e., cache space) for a proxy is proportional to the number of local clients and it is substantially lower than the original server. Requested streams are always delivered through the proxy, thus it is able to intercept and cache them. However, the cached streams and their corresponding qualities are adaptively adjusted based on both popularity of each stream among clients and available bandwidth between the proxy and interested clients. The proxy server can significantly increase the delivered quality of popular streams to high bandwidth clients despite the presence of a bottleneck on the path to the original server. Due to low storage and processing requirements for a proxy, any institution can easily deploy a proxy server to improve their perceived quality.

A primary challenge for multimedia proxy caching in the Internet is the need to operate within the context of congestion control. Because of the shared nature of the Internet all end-systems, including streaming applications, are expected to perform end-to-end congestion control to keep the network utilization high while limiting overload and improving inter-protocol fairness [6]. Since a dominant portion of today's Internet traffic consists of TCP-based flows such as HTTP, TCP-friendly congestion control [12] is preferred.

Performing congestion control results in random and potentially wide variations in transmission rate. To maximize the delivered quality to clients while obeying congestion controlled rate limits, streaming applications should be quality adaptive over the Internet— that is, they should

match the quality of the delivered stream with the average available bandwidth on the path. Thus the quality of cached streams will depend on the available bandwidth to the first client that retrieved the stream. Once the stream is cached, the proxy can replay it from the cache for subsequent requests but it still needs to perform congestion control and quality adaptation based on the state of the connection between the proxy and the client. This connection is likely to exhibit different characteristics (e.g. different average bandwidth, changes in background traffic) from previous sessions. Thus, variations in the quality of the cached stream are not correlated with the required changes in playback quality due to quality adaptation during the new session.

We have developed an end-to-end client/server architecture for playback of quality adaptive multimedia streams in a congestion controlled fashion over the Internet [13]. We have adopted a layered approach to quality adaptation using hierarchical encoding [21]. With hierarchical encoding, each stream is split into a base layer that contains the most essential low quality information, and higher layers that provide optional enhancement information. Thus the more layers are played back, the better the quality becomes. The quality adaptation mechanism adjusts the quality by adding and dropping layers as the available bandwidth changes.

Layered organization of the stream provides an opportunity to adjust the quality of the cached stream in a demand-driven fashion. To allow fine-grain adjustment of quality, each layer of the encoded stream is divided into equal-size pieces called *segments*. Thus the proxy can pre-fetch those segments that are required by the quality adaptation mechanism and are missing in the cache. If available bandwidth between the proxy and a client can support a stream with a higher quality, higher layers are gradually pre-fetched from the server to improve the quality. Thus the quality of the cached stream is adjusted with its popularity.

Rapid increase in the volume of multimedia traffic in the Internet justifies the need for multimedia proxy caching. Our multimedia proxy caching mechanism has other advantages for both high and low-bandwidth (e.g. dial-up) clients:

- Supporting low-latency VCR-functions for clients.
- Supporting asynchronous access.
- Minimizing startup latency.
- Reducing load on the server and the network.

As bandwidth and sever capacity increase, the demand and ability to support streaming applications are expected to develop rapidly. Realtime streams have several inherent properties that can be exploited in the design of effective multimedia proxy caching mechanisms.

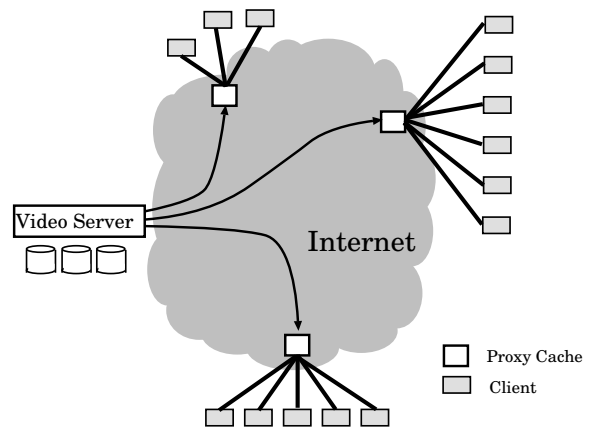


Fig. 1. The end-to-end server/client/proxy architecture

- Because of large size and duration, the entire object need not be sent at once. Instead, the server can pipeline the data to the client through the network.
- Multimedia streams are able to change their size by adjusting their quality.

#### A. The Proxy-based Architecture

Figure 1 shows an extension of an end-to-end client/server architecture that includes proxy servers. This architecture is still end-to-end, i.e., proxy servers are end systems and do not require any support from the network. All streams are layered encoded and stored at the server's archive. Here we assume *linear-layered* encoding where all layers have the same bandwidth just for the sake of simplicity, but the architecture and the caching scheme can be extended to other layered-encoding bandwidth distributions. Traffic is always routed through a corresponding proxy server that is associated with a group of clients in its vicinity. Thus the proxy is able to intercept each stream and cache it. All playback streams between the original server and the client or between the proxy server and the client must perform congestion control and quality adaptation. This implies that not only the original server, but also the proxy server, must be able to support congestion control and quality adaptation. We do not make any assumption about the inter-cache architecture. Our work is compatible with the various inter-cache architectures.

#### B. Contribution of this Paper

This paper presents a novel multimedia proxy caching mechanism for layered encoded multimedia streams. We describe a pre-fetching scheme that enables the proxy to perform quality adaptation more effectively while improving the quality of the cached stream in a demand-driven fashion during subsequent playbacks from the cache. We also exploit inherent properties of multimedia streams and

streaming applications to devise a fine-grain replacement algorithm suited for multimedia proxy caching. We show that interaction between our replacement algorithm and pre-fetching results in the state of the cache converging to an efficient state such that the quality of a cached stream is proportional to its popularity, and the variations in quality of a cached stream are inversely proportional to its popularity. This implies that after serving several requests for a stream, the proxy can effectively hide low bandwidth paths to the original server from interested clients. Thus the delivered quality of a popular stream is not limited to the available bandwidth from the original server. Instead, the quality of each stream is determined by its popularity and the average bandwidth between the proxy and interested clients. Our simulation results demonstrate that our mechanism effectively achieves its goals.

The rest of this paper is organized as follows: In section II we present the delivery procedures on a cache-hit and cache-miss scenarios in order to demonstrate role of proxy in each scenario. We also discuss the pre-fetching mechanism during the cache hit scenario to cope with variations in quality and address some of the related challenges and trade-offs. Different aspects of our replacement algorithm including fine and coarse-grain replacement pattern as well as popularity function are described in section III. We present our simulation environment and some of our simulation results in section IV. Section V summarizes some of the related works on multimedia caching. Finally, section VI concludes the paper and presents our future directions.

## II. DELIVERY PROCEDURE

Clients always send their requests for a particular stream to their corresponding proxy. When a proxy receives a request, it looks up the cache for availability of the requested stream. The rest of delivery procedure varies for a cache miss or a cache hit. We describe each scenario separately in the next two subsections.

### A. Relaying on a Cache Miss

If the requested stream is missing from the cache, the request is relayed to the original server or neighbor caches, depending on inter-cache architecture. The original server plays back the stream to the client through the proxy. The proxy relays data packets toward the client and the ACK packets in the reverse direction. Thus the proxy remains virtually transparent from the end systems' point of view while it is able to intercept and cache each packet. The server performs end-to-end congestion control and quality adaptation based on the state of the session between the server and the client. The quality of the delivered stream

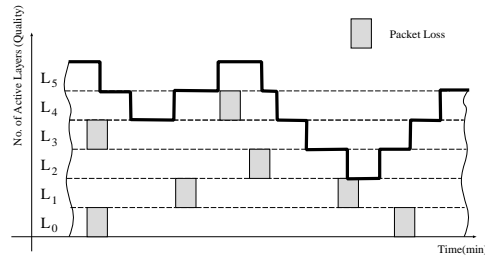


Fig. 2. A sample quality adaptive stream in the cache

is limited to the average bandwidth between the server and the client. Thus on a cache miss scenario, the client does not observe any benefit (e.g. improvement in quality or lower startup latency) from the presence of the proxy cache.

The proxy always caches a missing stream during its first playback. If cache space is exhausted, the replacement algorithm flushes a sufficient number of segments from the cache to make room for the new stream. Details of the replacement algorithm are discussed in section III.

Since the original server performs quality adaptation, a cached stream has variable quality after its first playback. Furthermore, there might be occasional missing packets that have been lost and were not repaired during the first session because they have missed their playout times. Figure 2 shows variations in quality as well as missing packets for a portion of a sample cached stream. To perform quality adaptation effectively during subsequent playbacks from the cache, the proxy may smooth out the variations of the cached stream and repair the losses by pro-actively pre-fetching the missing segments during idle hours. Alternatively, the proxy may per-fetch missing segments on a demand-driven fashion while it serves subsequent requests for the cached stream. We adopted the latter approach assuming the future access pattern is not predictable. We plan to investigate pro-active pre-fetching as part of our future work.

### B. Pre-fetching on a Cache Hit

On a cache hit, the proxy acts as a server and starts playing back the requested stream from the cache. As a result the client observes shorter startup latency. The proxy must still perform congestion control and quality adaptation. However the connection between the proxy and the client might have different characteristics. Moreover, there is no reason to expect that the variations in quality of the cached stream would be correlated with the variations in quality of the delivered stream (those that are due to quality adaptation by the proxy). This implies that the quality adaptation mechanism may be able to send some segments that do not exist in the cache. To maximize the delivered

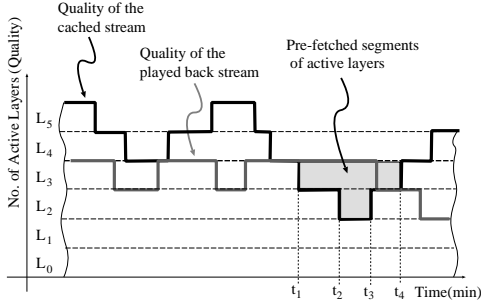


Fig. 3. Delivery of lower bandwidth stream from the cache

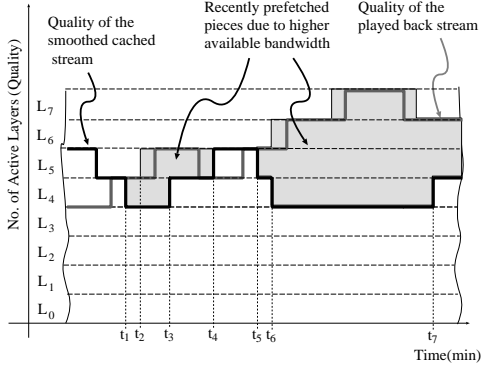


Fig. 4. Delivery of higher bandwidth stream from the cache

quality to the client, the proxy should pre-fetch the missing segments that are required by the quality adaptation mechanism from the server ahead of time. During each interval of the session two scenarios are possible:

1.  $Playback_{AvgBw} \leq Stored_{AvgBw}$
2.  $Playback_{AvgBw} > Stored_{AvgBw}$

where  $Playback_{avgBw}$  and  $Stored_{avgBw}$  denote average bandwidth of the playback session and the cached stream respectively. Note that during a complete session, the proxy may sequentially experience both of the above scenarios. Figure 3 depicts a scenario where the average quality of the delivered stream is lower than the cached stream. However, there are segments that are required by the quality adaptation mechanism but are missing from the cache. For example, segments of layer 2 within the interval of  $[t_2, t_3]$  and segments of layer 3 within the interval of  $[t_1, t_4]$  are required by the quality adaptation mechanism but are not available in the cache. The second scenario is shown in Figure 4 where the available bandwidth between the proxy and the client is sufficiently high to deliver a higher quality stream than the cached stream. Thus the proxy not only needs to pre-fetch missing segments of the lower layers, it may occasionally pre-fetch higher layers as soon as quality adaptation indicates possibility of adding a new layer in the future. All the pre-fetched segments during a session are cached in both scenarios.

### C. Challenges and Trade-offs

During playback of a cached stream, the proxy needs to maintain two unsynchronized connections, (i) the connection between the server and the proxy for pre-fetching and (ii) the connection between the proxy and the client for delivery of the stream, where the proxy performs quality adaptation. Pre-fetching a segment from the server will take at least one RTT between the server and the proxy. Thus the proxy must predict a missing segment that may be required by the quality adaptation mechanism in the future and send a pre-fetching request. Since the quality adaptation mechanism adjusts the number of active layers with the random changes in available bandwidth, the time for the upcoming adjustment (i.e. adding or dropping of a layer) is not known a priori. This implies that the proxy is facing a trade-off, the earlier the proxy requests for pre-fetching of a missing segment, the less accurate the prediction would be, however the higher is the chance of receiving the requested segment in time.

The rate of pre-fetching requests from the proxy to the server depends on the available data in the cache as compared with available bandwidth between the proxy and interested client. However pre-fetched segments are delivered in a congestion controlled fashion from the server to the proxy. Thus if the requested rate for pre-fetching exceeds the available bandwidth between the server and the proxy, the server should deliver the requested segments based on their priority otherwise the pre-fetching stream will fall behind the playback stream and pre-fetched segments may miss their playout deadline.

The pre-fetching mechanism that addresses this problem is illustrated in figure 5. The proxy maintains a playout time for each active client. At playout time  $t_p$ , the proxy examines the interval of  $[t_p + T, t_p + T + \delta]$ , called the pre-fetching window of the cached stream, and identifies all the missing segments within the pre-fetching window. The missing segments include any lost segment or the segments of current active layers within the pre-fetching window that have not been played back. Furthermore, if the quality adaptation mechanism is close to adding a new layer, any missing segment of the new layer within the pre-fetching window is included in the pre-fetching request as well. This mechanism enables the proxy to improve the quality during a playback if there is sufficient pre-fetching bandwidth available. Then the proxy sends a single pre-fetching request to the server that refers to a batch of missing segments in the current pre-fetching window.

To loosely synchronize the pre-fetching stream with the playback stream, the pre-fetching window should slide as

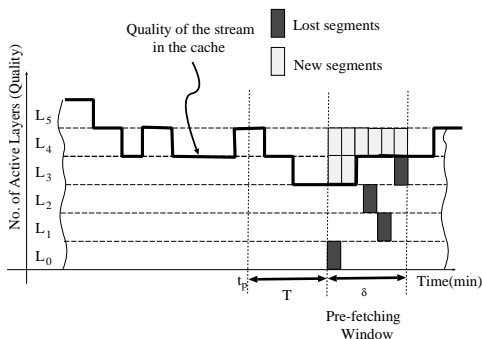


Fig. 5. pre-fetching mechanism

fast as the playout point. Thus after  $\delta$  second, the proxy examines the next pre-fetching window and sends a new pre-fetching request to the server.

When the server receives a pre-fetching request, it starts sending the requested segments based on their priorities. Thus it first sends all the requested segments of layer 0, then requested segments of layer 1, etc. A new pre-fetching request preempts the previous one. If the server receives a new pre-fetching request before delivery of all the requested segments in the previous request, it simply ignores the old request and starts delivery of the segments in the new request based on their priorities. The preempting mechanism results in pre-fetching high priority segments while still limiting the pre-fetching rate to the congestion controlled rate limit. Furthermore, it causes the pre-fetching and the playback to proceed with the same rate. Notice that the average improvement in quality of a cached stream after each playback is determined by average pre-fetching bandwidth. Thus it may take several playbacks until the quality of the cached stream reaches the maximum quality that can be viewed by a high bandwidth client.

To avoid multiple pre-fetching sessions between a server and a proxy, all the pre-fetching sessions must be multiplexed into a single congestion controlled flow. This implies that as the number of pre-fetching sessions increases, the allocated pre-fetching bandwidth share for each session decreases. The pre-fetched segments are always cached even if they arrive after their playout times.

### III. REPLACEMENT ALGORITHM

This section addresses different aspects of our replacement algorithm. The replacement algorithm must be designed based on the expected functionality from the cache. We assume that it is generally preferred to have a complete stream in the cache and adjust its quality based on its popularity. Thus the chief goal of the caching mechanism is to converge the state of the cache to an efficient state after

several playbacks. We define that the state of the cache is efficient if the following conditions are met:

- The *average quality* of any cached stream is directly proportional to its popularity. Furthermore, the average quality of the stream must converge to the average bandwidth across the most recent playback for interested clients.
- The *variations in quality* of any cached stream is inversely proportional to its popularity.

We first describe coarse and fine-grain replacement patterns that are suited to layered-encoded stream. Then we extend semantics of a “hit” from Web caching and define a simple popularity function that captures level of interest among users who interactively perform VCR-functionalities.

#### A. Replacement Pattern

Web objects do not have a sub-structure. Consequently each object is usually treated as an atomic object, i.e. a client requests and receives the entire object. Thus most of the replacement algorithms for Web caching make a binary decision for replacement of web objects, i.e. the least popular object is flushed in its entirety. However, layered encoded streams are structured into separate layers. Furthermore, each layer is divided into the same number of segments with a unique segment ID. This organization allows us not only to make a *multi-valued* replacement decision but also to perform replacement with different granularities. The stream popularity primarily affects the quality of a cached stream and then its status of residency in the cache. As the popularity of a cached stream decreases, its quality and consequently its size is reduced in several steps before it is completely flushed out.

Figure 6 depicts the replacement pattern for segments within a single cached stream. The coarse-grain replacement is achieved by dropping the highest layer, called the *victim layer*, from the cache. However, to maximize the efficiency of the cache and avoid fragmentation of the cache space, the victim layer is dropped with the granularity of a segment.

It is generally preferred to cache a contiguous portion from the beginning of a layer to absorb startup latency and minimize the variations in quality[19]. Thus once a victim layer is identified, its cached segments are flushed from the end towards the beginning in a demand-driven fashion. If flushing all segments of the victim layer does not accommodate sufficient space for a new stream, the proxy identifies a new victim layer and repeats the fine-grain replacement process.

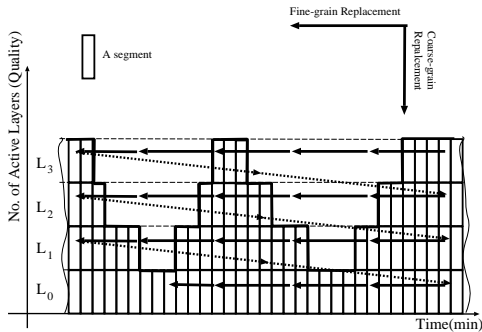


Fig. 6. Replacement priority within a cached stream

## B. Popularity Function

We initially used number of hits (i.e. requests) for a cached resident stream during an interval, called *popularity window*, as a metric to measure its popularity. Most of the current Web caching schemes assign a binary value to a hit, i.e. 0 for lack of interest and 1 for each request. This model perfectly suits the atomic nature of interest in web objects, i.e. the client is either interested in the entire object or is not interested at all. However, in the context of streaming applications, the client can interact with the server and perform VCR-functionalities (i.e. Stop, Fast forward, Rewind, Play). Intuitively, the popularity of each stream should reflect the level of interest that is observed through this interaction. We assume that the total duration of playback for each stream indicates the level of interest in that stream. For example if a client only watches half of one stream, its level of interest is half of a client who watches the entire session. This approach can also include weighted duration of fast forward or rewind with proper weighting.

Based on this observation we extend the semantics of a hit and define the term *weighted hit (whit)* which is defined as follows<sup>1</sup>:

$$whit = \frac{PlaybackTime}{StreamLength}, \quad 0 \leq whit \leq 1$$

where *PlaybackTime* and *StreamLength* denote total playback time of a session and length of the entire stream, respectively. Both *PlaybackTime* and *StreamLength* have dimension of time (i.e. measured in second).

Notice that adding and dropping layers by the quality adaptation mechanism results in a different *PlaybackTime* for various active layers in a session and consequently affects the value of a cached *layer*. For ex-

<sup>1</sup>The term “weighted hit” have been used in caching literature [4] to extend the definition of hit in the context of Web caches. However we have introduced a new definition for this term in the context of proxy caching for multimedia streams.

ample, even if all layers of a stream are available in the cache and the client watches the entire stream, the quality adaptation mechanism may only send Layer 0, 1 and 2 for 100%, 80%, 50% respectively, of the play back time. Since the available bandwidth directly controls the number of active layers, the longer a layer is played back for interested clients during recent sessions, the higher is the probability of using that layer in future sessions. To capture the value of a layer, the server calculates the value of weighted hits on a per-layer basis for each session. The total playback time for each layer is recorded and used to calculate the *whit* for that layer at the end of the session. The cumulative value of *whit* during a recent window is used as a popularity index of a layer of a cached stream. The popularity of each layer is recalculated at the end of a session as follows:

$$P = \sum_{x=t-\Delta}^t whit(x)$$

where  $P$  and  $\Delta$  denote popularity and the width of the popularity window respectively. Applying the definition of popularity on a per-layer basis is in fact compatible with our proposed fine-grain replacement mechanism because layered decoding guarantees that popularity of different layers of each stream monotonically decreases with the layer number<sup>2</sup>. Thus a victim layer is always the highest layer of one of the cached stream. Notice that length of a cached stream does not affect its popularity because replacement is performed at the granularity of a segment instead of a layer.

The proxy maintains a popularity table such as Table I. Each table entry consists of stream name, layer number, popularity of the layer and a locking flag. Once the cache space is exhausted, the proxy flushes the last segments of the least popular layer (e.g.  $L_1$  of “Amistad”) until sufficient space becomes available. Popularity of this layer could be low due to lack of interest among clients, or lack of sufficient bandwidth to play this layer for interested clients, or both<sup>3</sup>. Except for the base layer of each stream, all segments of other layers can be flushed out if they are the least popular layer. The first few segments of the base layer for each stream are kept in the cache for a long period to hide the startup latency of possible future requests.

<sup>2</sup>The decoding constraint requires that to decode a segment of layer  $i$ , corresponding segments for all the lower layers must be available

<sup>3</sup>Note that these three scenarios can be easily recognized from the distribution of popularity values among layers. Close popularity values imply lack of clients interest whereas widely variable popularity values imply lack of available bandwidth to play the less popular layers.

$P$	Lock	Stream name	Layer no.
5.85	1	<i>Titanic</i>	$L_0$
4.92	1	<i>Titanic</i>	$L_1$
4.76	0	<i>Amistad</i>	$L_0$
3.70	1	<i>Titanic</i>	$L_2$
3.50	0	<i>Contact</i>	$L_0$
3.33	0	<i>Apollo 13</i>	$L_0$
2.30	0	<i>Titanic</i>	$L_3$
1.28	0	<i>Amistad</i>	$L_1$

TABLE I  
SAMPLE OF A POPULARITY TABLE

### C. Locking Mechanism

The replacement for a web object is an atomic operation. However, in the context of multimedia streams, replacement is a timely process that proceeds gradually as the session continues. This causes the potential for thrashing where the tail of the highest layer of the cached stream (e.g.  $L_3$ ) is flushed to make room for the initial segments of a higher layer (e.g.  $L_4$ ) of the same stream during a playback session with higher bandwidth. To avoid this, while a particular stream is played back from the cache, its layers are locked in the cache and can not be replaced. In practice, each layer is locked as soon as it is played back for the first time and remains locked until the end of the session. Thus if a layer has never been played during the session then it does not need to be locked. At the end of the session, the weighted hit of each layer is calculated and consequently the popularity value of each layer is updated in the popularity table. Then all the locked layers of that stream are unlocked.

## IV. SIMULATION

We evaluate our multimedia caching mechanism via simulation using the *ns* [1] simulator. It provides web server, client and proxy cache modules, and allows us to quickly evaluate our design under various scenarios. We use RAP [15] as underlying congestion control mechanism along with layered quality adaptation [14] as transport protocol for multimedia streams. Our simulation does not include any error control mechanism.

As we discussed earlier, our caching mechanism has two major design goals: pre-fetching during playback to enhance cached stream quality, and fine-grain cache replacement algorithm to adjust stream quality based on per-layer popularity. In this paper, we are merely interested in qualitatively evaluating the correctness of our mechanism to verify whether and how our mechanism satisfies its design

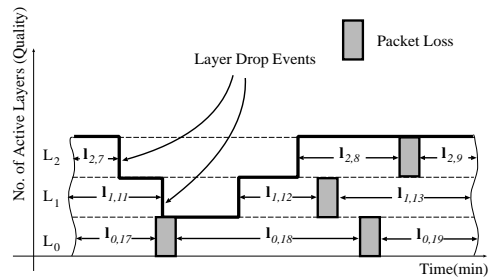


Fig. 7. Average quality and drop events for a cached stream

goals. It remains as future work to examine the *performance* of our caching mechanism such as the byte hit ratio, or the convergence speed of our replacement algorithm under realistic background traffic. The performance analysis will require careful examination under wide range of parameter settings.

.0.a Evaluation Metrics. To examine the correctness of our mechanism, we choose the following two metrics that collectively represent the resulting quality of a layer of a cached stream:

- *Completeness* measures the percentage of a stream residing in the cache. This metric allows us to trace the evolution of a cached stream after each playback as a result of replacement and pre-fetching. Because our replacement algorithm is layer-based, we define the completeness on a per-layer basis. We define the completeness of a layer as the ratio of its size in cache to its “official length”:

$$Cp(s, l) = \frac{\sum_{\forall i \in Chunks(l)} L_{l,i}}{RL_l} \quad (1)$$

We define a *chunk* as a continuous group of segments of a single layer of a cached stream, and denote the set of all chunks of layer  $l$  as  $Chunks(l)$ .  $L_{l,i}$  is the length of the  $i$ th cached chunk in layer  $l$ , and  $RL_l$  is the “official” length of the layer. Obviously the value of completeness always falls within  $[0,1]$ . For example, when a stream sits at its server, each of its layer has completeness value of 1.

- *Continuity* measures the level of smoothing of a cached stream. Completeness alone does not capture the level of smoothing for a cached layer. It does not represent the continuity of segments in a layer. Thus we define continuity on a per-layer basis. The continuity of a cached stream  $s$  is defined as the average the number of bytes between two consecutive layer break.

$$Ct(s, l) = \frac{\sum_{\forall i \in Chunks(l)} L_{l,i}}{LayerBreak + 1} \quad (2)$$

A layer break occurs when there is a missing segment in a layer. A missing segment may be due to either quality

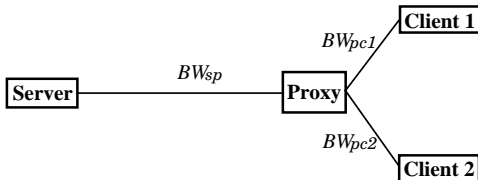


Fig. 8. Simulation topology

adaptation dropping a layer, or a packet loss. Figure 7 illustrates this for a portion of a cached stream. Note that layer-drop and packet loss are fundamentally two different phenomena. However pre-fetching copes with both of them similarly in a priority-based fashion. In the absence of an error control mechanism, our results represent worst case scenarios.

.0.b Simulation Setup. We have performed two sets of simulations. The first set focuses on evaluating the ability of the pre-fetching algorithm to gradually improve the quality of a cached stream if there is unlimited cache space. The second set focuses on evaluating the interaction between pre-fetching and replacement algorithm to converge state of the cache to the optimal state as defined in Section III.

For both sets of simulations, we used the same simple network topology depicted in figure 8.  $BW_{sp}$  denotes the average available bandwidth between server and proxy whereas  $BW_{pc1}$  and  $BW_{pc2}$  are bandwidths between proxy and the two clients, respectively. This simple topology is not chosen arbitrarily. As we will discuss later, our sequential request pattern enables us to group many clients whose connections to the proxy have similar bandwidth, and use a single client to represent them. For any client, one may construct two interesting scenarios from this simple topology:

- Scenario I:  $BW_{sp} < BW_{pc1}$ , the server-proxy connection is the bottleneck.
- Scenario II:  $BW_{sp} \geq BW_{pc1}$ , the proxy-clients connections are the bottleneck.

We are particularly interested in scenario I because this is the ideal case for the proxy to gradually improve the quality of cached streams. In scenario II, however, the quality of the cached stream will always be higher than what the client can afford. Thus there is simply no incentive for the proxy to improve the quality. Assuming multiple clients, there is an interesting case of mixing Scenario I and II by having  $BW_{sp} < BW_{pc1}$  and  $BW_{sp} \geq BW_{pc2}$ . Different client bandwidth will affect the resulting quality of a cached stream, as we will discuss later in this section.

There are other parameters controlling our simulations, such as cache capacity, segment size, layer consumption rate (the rate at which receiver consumes data in each

layer)<sup>4</sup>, number of layers, number of streams and stream length. To limit number of parameters, we let all streams have the same segment size of 1KB, layer consumption rate of 2.5KB/s, and 8 layers. Changing these parameters will not qualitatively change our results as long as they are changed proportionally. In all of our simulations the server-proxy link is shared by 10 RAP and 10 long-lived TCP(*i.e.* FTP) flows. One of the RAP flows is used for delivery of streams from the server to the cache and the other 19 flows simulate a fairly realistic background traffic. Since RAP is TCP-friendly, each flow obtains an even share of bandwidth on average. Furthermore, dynamics of the background traffic results in bandwidth changes that triggers adding and dropping of layers. To allow gradual improvement in quality, bandwidth of the server-proxy link is set to  $(20 \times 56 \text{Kbps} = 1.12 \text{ Mbps})$ . Thus bandwidth share of server-proxy connection is 56 Kbps on average(*i.e.*  $BW_{sp} = 56 \text{ Kbps}$ ) which is only enough for 2.8 layers.

Without statistical knowledge about the size distribution of multimedia streams in the Internet, we choose stream duration from a uniform distribution between 1 minute and 10 minutes (the size in byte can be obtained by multiplying the stream length, number of layers and layer consumption rate). Any stream longer than 10 minutes, can be viewed as combinations of several shorter streams with the same popularity. Further details about setting these parameters are specific to each set of simulations, and are discussed later.

In order to generate a request sequence, we need to know two factors: first, stream popularity, *i.e.*, how many requests are issued for each stream; and second, temporal distribution of the requests for different streams. We choose two simple models for each of the two factors.

First, we assume that the popularity distribution of these stream conforms to the same distribution that was observed in web page requests, *i.e.*, Zipf's distribution [3]. We then generate the number of requests for each stream using the properties of Zipf's law<sup>5</sup>: given the number of total requests  $R$  and total number of streams  $N$ , we let the  $m$ th popular stream have  $\frac{1}{m} \Omega R$  requests, where  $\Omega = \sum_{i=1}^N \frac{1}{i}$ .

It is non-trivial to generate a temporal request sequence where the number of requests to each stream conforms to the Zipf's law [2]. Requests are uniformly distributed in

<sup>4</sup>We assume linear layered encoding, *i.e.*, every layer has the same consumption rate.

<sup>5</sup>Although many web traces conform to Zipf's law, many other web proxy traces do not follow the exact Zipf's law. Instead, they exhibit *Zipf-like behavior*, as described in [3]. Here, we use Zipf's law for simplicity.

this sequence with the interval large enough so that no two requests overlap. Meanwhile the number of requests conforms to the above Zipf’s distribution. The subtle difference between a conforming request sequence and a non-conforming sequence may lead to different cache performance, such as hit ratios. Since performance analysis is not a primary concern, we are able to simplify the request sequence generation by assuming that different requests are served sequentially by the proxy, *i.e.*, the proxy transmits at most one multimedia stream at any time. As a result, we exclude the situation of simultaneous playbacks of multiple cached streams. The only added complexity in these situations is that two or more streams compete for the pre-fetching bandwidth between the server and the proxy. We consider this as a secondary effect, and ignoring it does not affect the correctness of the evaluation of our algorithms. Avoiding this situation reduces the number of variables that affect the replacement and helps us to understand the simulation results because we are then able to assess the effect of more important parameters. One issue which still remains is how to distribute requests between two clients who have different bandwidth to the proxy. We will discuss it later.

## A. Results

### A.1 Pre-fetching

This simulation is intended to demonstrate that the pre-fetching algorithm results in gradual improvement in quality of cached streams when there is a bottleneck between the server and the proxy (*i.e.* scenario I). To disable the cache replacement mechanism, we set cache size to infinity and use only one stream and one client. As discussed before, when  $BW_{sp} \gg BW_{pc1}$ , the quality of the cached stream will immediately reach the maximum quality that can be afforded by the client, and leave little room for gradual quality improvement. Therefore we choose average  $BW_{sp}$  to be 56Kbps<sup>6</sup>, and  $BW_{pc1}$  to be 1.5Mbps. Stream size is set to 5 minutes. The simulation ran for 125 minutes, containing 15 completed requests.

Figure 9 shows the evolution of completeness and continuity of every layer of the cached stream. Every point in the graph represents the status of a particular layer after one playback. We use time for the *x*-axis instead of request number to be comparable with other scenarios. Time is more appropriate for the following scenarios where various streams have different length. Since continuity inver-

sly depends on the number of layer break, we plot continuity in a log-scale. It takes about 4 requests for the 4 lowest layers to achieve maximum quality. The higher is a layer, the more requests it takes to improve the layer’s quality.

The convergence speed is not constant, rather, it happens in a thresholded fashion. For every layer there are several requests that greatly enhance its quality, while other requests just marginally improve its quality. This can be explained by the layer dependence during pre-fetching: a higher layer is only pre-fetched when corresponding data of all lower layers are available. Thus, the higher is a layer, the longer it takes to obtain a share of pre-fetching bandwidth and receive missing segments.

Currently we only repair for packet losses by pre-fetching. We expect that the convergence speed in continuity will be much faster if the transport layer provide error control mechanisms, which we do not have right now.

### A.2 Replacement Algorithm

This set of simulations is intended to examine the interaction between pre-fetching and replacement mechanisms to converge the state of the cache to an optimal state after serving a series of requests for multiple streams.

The resulting quality due to cache replacement depends on two factors: stream popularity and the bandwidth between interested client and the proxy. If we assume  $BW_{sp} \gg BW_{pc1}$ , the first factor will be dominant. It is also inadequate to only examine the other extreme case, *i.e.*,  $BW_{sp} \geq BW_{pc1}$ , where the client bandwidth becomes the major factor and overshadows popularity. As a solution, we studied both extreme cases and an additional intermediate case using two clients with different bandwidth to the proxy. The bandwidth share of the RAP connection between the server and the proxy is 56Kbps on average. The bandwidth between client 2 and the proxy is 56Kbps, sufficient for 2.8 layers. The bandwidth between client 1 and the proxy is 1.5Mbps that is sufficient for delivery of all 8 layers. By distributing the number of requests between the high bandwidth client 1 and the low bandwidth client 2, we are able to go from one extreme ( $BW_{sp} < BW_{pc1}$ ) to the other ( $BW_{sp} \geq BW_{pc2}$ ).

We choose 10 streams with uniformly distributed duration ranging from 1 minute to 10 minutes. Their popularity decreases with their index, *i.e.*, stream 0 is the most popular one. In order to show the effect of cache replacement, we set the cache size to be half the total size of all 10 streams. This cache size is chosen heuristically: we want a moderate number of replacements, but not so many as to cause frequent oscillations in quality. In all of these simulations, we fix the popularity window of our replacement algorithm to  $\infty$ , because we expect that in reality the popu-

<sup>6</sup>That means the total bandwidth between the server and the proxy is (20\*56Kbps) that is shared by 10 RAP and 10 TCP flows. Share of the RAP connection between the server and the proxy is 56Kbps on average

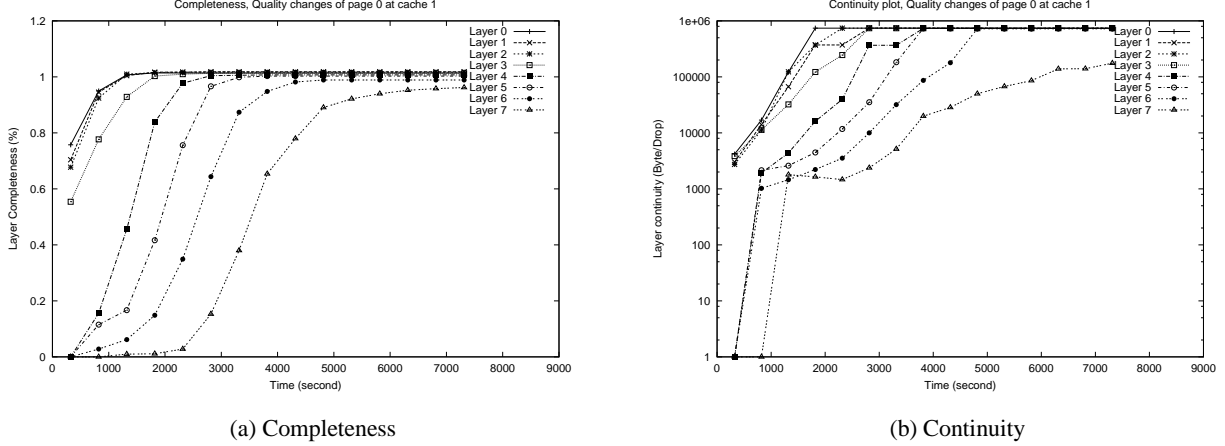


Fig. 9. Quality improvement due to pre-fetching.

larity window should be much larger than the time-scale in our simulations. We leave it as future work to investigate the impact of the popularity window.

We have conducted other simulations to show that our results do not critically depend on the selected simulation parameters. In particular, we studied the impact of cache size and the number of streams on the resulting quality evolution of cached streams. Our results suggested that the conclusions presented below are still applicable when those parameters are changed. Due to space limitation, we refer readers to [17] for further details.

**A.2.a Effect of Popularity.** In order to emphasize the influence of stream popularity on cache replacement, we should reduce the influence of client bandwidth to the minimum. We achieved this by distributing 95% of all requests to client 1, who has 1.5Mbps to the proxy, and only 5% to client 2. The simulation ran for about 44 (virtual) hours, containing 310 requests. Here we only show the first half (i.e. first 22 hours) in these graphs; the rest shows the same trend and is omitted for clarity.

Figures 10 and 11 show the quality change of the most popular stream and the least popular stream, respectively. The most popular stream is eventually able to keep all of its layers in the cache after 20 requests. Compared to the most popular stream, the least popular stream is not able to keep adequate quality in the cache (it was almost completely flushed out at one time), and all of its layers experienced many drops. Its quality keeps oscillating because it was accessed only occasionally and had to be flushed out to make room for other streams.

To summarize, popular streams are likely to have more layers in the cache, and these layers tend to have higher quality both in terms of completeness and layer continuity.

However, when cache space is limited, the highest layers of popular streams may still be flushed out to make room for unpopular streams which are currently being played back.

**A.2.b Effect of Client Bandwidth.** This simulation examines the effect of the bandwidth of the interested clients on the quality of the cached streams. Because we accumulate hit count on a per-layer basis, if most clients who have requested the stream can not afford higher layers, even if the stream is extremely popular, only the lower layers should be kept in the cache. In order to examine the effect of popularity, we assigned 95% of all requests to the low bandwidth client 2 and only 5% of all requests to the high bandwidth client 1.

Figure 12 shows the quality change of the most popular stream, and Figure 13 shows that of the least popular stream. Comparing Figure 12 to Figure 10 reveals that when clients have limited bandwidth, the maximum quality of the popular cached stream drops significantly. In the previous case, when the majority of requests come through a 1.5Mbps connection, the cached stream could keep all 8 layers in cache. However in this case, the cached stream keeps only 4 layers as the majority of requests come through a 56Kbps connection, and the highest 3 layers exhibit oscillating behavior because they were accessed less frequently by the high bandwidth client.

One interesting phenomenon in Figure 12 is that the highest 4 layers of the most popular stream finally converged to maximum. This suggests that if there is no replacement and the simulation ran long enough, Quality of cached streams converges to average bandwidth among interested clients. Furthermore pre-fetching will eventually

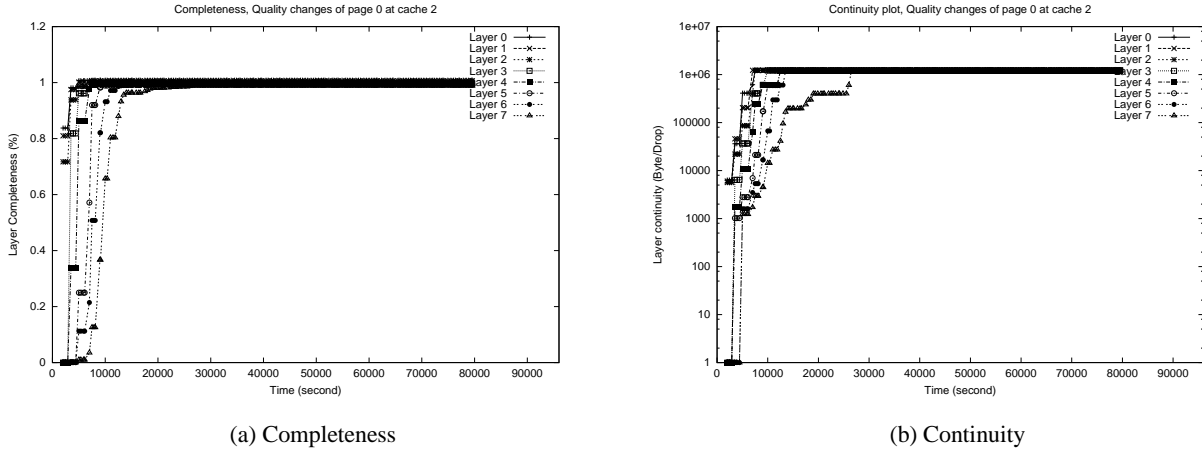


Fig. 10. Effect of popularity in cache replacement: the most popular stream.

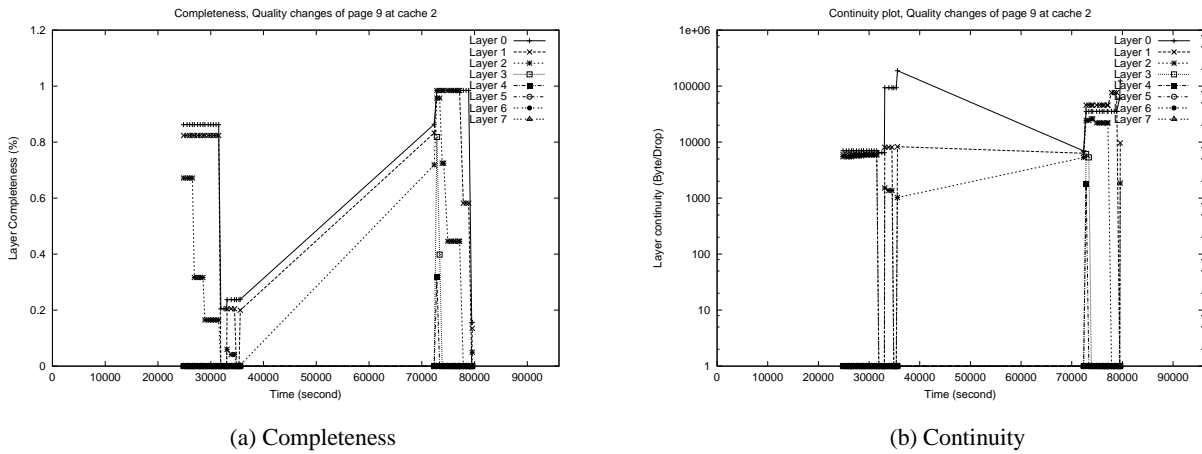


Fig. 11. Effect of popularity in cache replacement: the least popular stream.

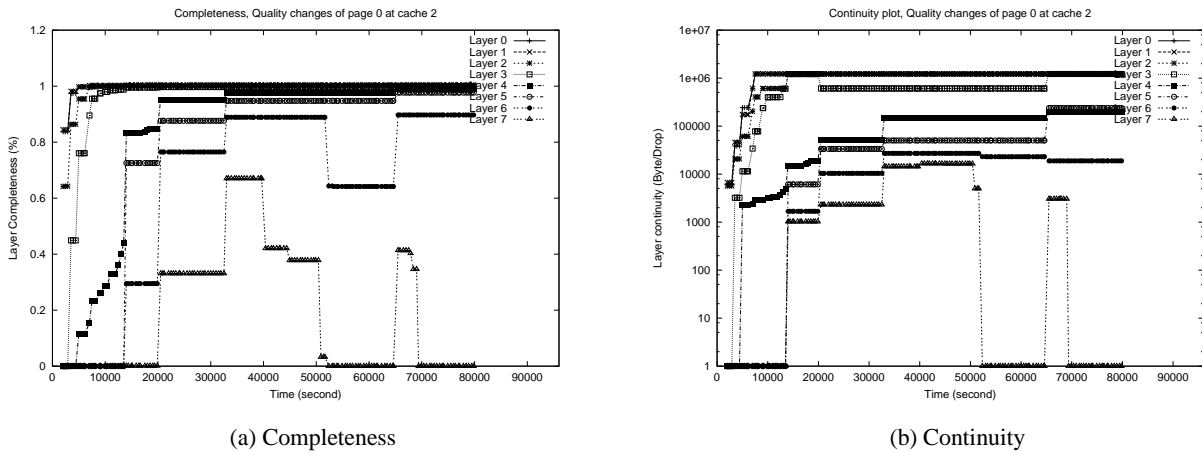


Fig. 12. Effect of client bandwidth in cache replacement: the most popular stream.

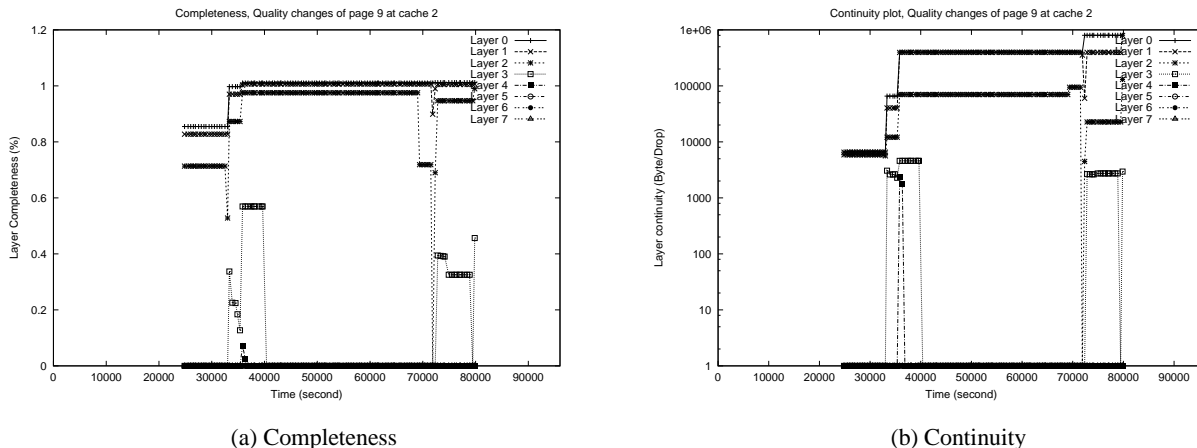


Fig. 13. Effect of client bandwidth in cache replacement: the least popular stream.

fill every lost segment in a stream, hence the continuity of every layer will be maximized.

The least popular stream in Figure 13 is able to keep 2 layers in the cache most of the time. This is due to the fact that the least popular stream caches large number of layer whenever it is played back for the high bandwidth clients. Thus its lower layer are more protected and have a higher chance of staying in the cache in compare to the previous case(Figure 11)

**A.2.c Mixing Together.** Having examined the above two extreme cases, this simulation examines an intermediate case where requests are evenly distributed between the low bandwidth client and the high bandwidth client. Figures 14 and 15 show the quality change of the most popular stream and the least popular stream, respectively.

Comparing Figure 14 to Figures 10 and 12, we found that the average quality of the popular stream is higher than the low bandwidth client case, but lower than the high bandwidth client case. A similar situation is observed for the least popular stream in Figure 15 in compare with Figures 11 and 13. This illustrates the essence of what our cache replacement algorithm is intended to achieve, *i.e.*, converge the resulting quality of cached stream to the average quality that has been accessed by clients.

Notice that the quality convergence here is closer to the high bandwidth case(Figure 10) than the low bandwidth case (Figure 12). This implies that the impact of client bandwidth limitation may be less than that of stream popularity.

## V. RELATED WORK

Memory caching for multimedia streams has been extensively studied in the context of multimedia servers [5],

[10]. The idea is to reduce disk access by grouping requests and retrieving a single stream for the entire group. In an abstract view, the problem is to design an object management mechanism that minimizes the migration of objects among different levels of a hierarchical storage system, *i.e.* tertiary and disk, disk and memory [7]. Most of these studies have focused on resource management issues. Note that there is an analogy between (tertiary, disk, memory) and (server, proxy, client). Object migrations among different levels of a hierarchical storage system are somewhat similar to object migrations among server, proxy, and client. However, there is a fundamental difference between these two problems. The available bandwidth between levels of a hierarchical storage system is fixed and known a priori. In contrast, available bandwidth between server and proxy or proxy and client randomly changes with time. As a result applying the proposed resource management schemes across the network becomes problematic. Once these entities are scattered across a best-effort network, all connection must be congestion-controlled. This issue has not been addressed previously.

Video streams exhibit burstiness due to encoding algorithm and variations within and between frames. The variation poses a problem to both bandwidth requirement and playback buffer management. In order to smooth the burstiness during transmission, a number of schemes have been proposed. One scheme [22] pre-fetches and stores portions of a stream in proxy servers, and later uses them to smooth the stream during playback. Another scheme [19] stores prefixes of multimedia streams in proxy caches. It is able to improve startup latency in addition to performing smoothing.

Our work is complementary to work on smoothing. Smoothing does not address congestion control of multi-

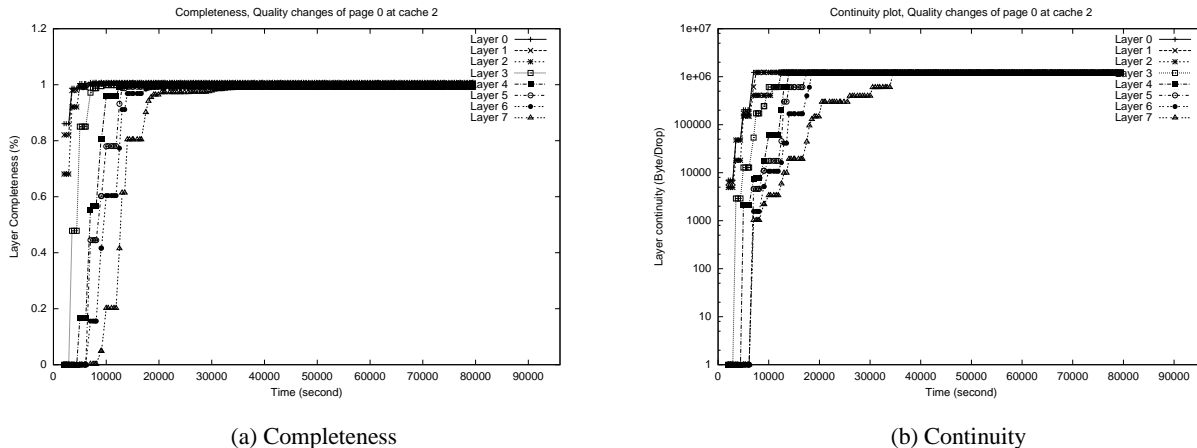


Fig. 14. General case of cache replacement: the most popular stream.

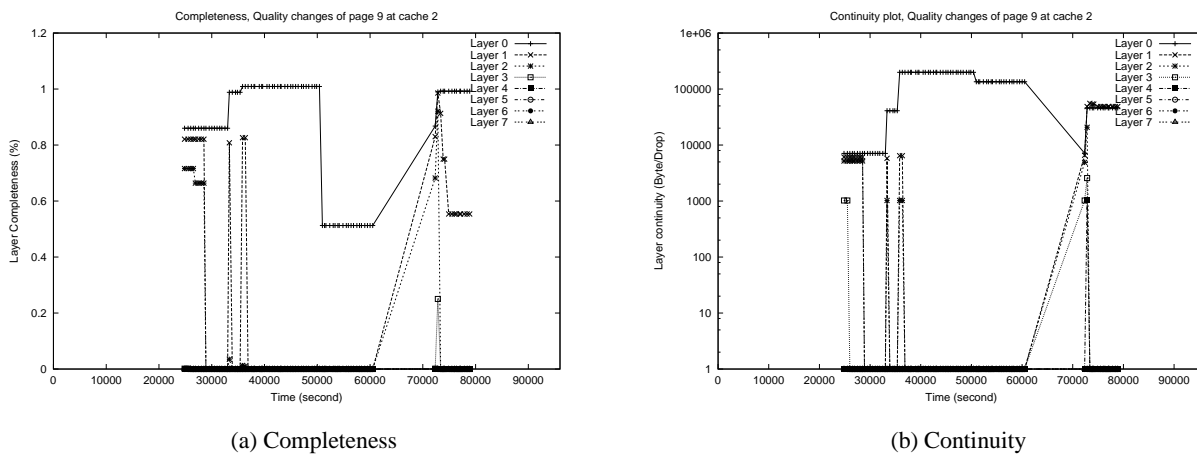


Fig. 15. General case of cache replacement: the least popular stream.

media streams in the Internet. We focus on the design of a proxy caching mechanism which is aware of congestion control mechanisms used in the transmission of multimedia streams. Cached streams with our mechanism can also be used to perform smoothing to facilitate client-side playback buffer management.

There are numerous works on proxy cache replacement algorithms [4], [?], [?], [?], [?]. They evaluate their algorithms using existing web proxy traces. However, the behavior of these algorithms for an access pattern with a significant number of requests to huge multimedia streams has not been studied. To our knowledge, there is only one work that addresses the influence of multimedia streams on cache replacement algorithms [20]. They consider the impact of resource requirements (i.e. bandwidth and space) on cache replacement algorithms. Our work complements

this effort in that we provide a more accurate estimation of bandwidth and size requirements through the exposure of congestion control mechanisms. Thus their algorithms may easily be applied to our architecture. It remains as future work to examine all these replacement algorithms with our proposed scheme.

## VI. CONCLUSIONS AND FUTURE WORK

This paper discussed a multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet. Our goal is to improve the delivered quality of a popular stream despite presence of a bottleneck between a server and interested clients. We assumed a layered approach to quality adaptation and sketched an end-to-end proxy-based architecture. Performing quality adaptation results in variable quality streams cached at a proxy. As

a result, performing quality adaptation during subsequent playback from the cache could be problematic because there is no correlation between the variations in quality of the cached stream and required quality for the new session.

Our layered approach to quality adaptation provides a perfect opportunity to cope with variations in quality of cached streams in a demand-driven fashion by pre-fetching required segments that are missing in the cache. We have also exploited inherent properties of multimedia streams and devised a fine-grain replacement algorithm. Simulation-based evaluation of our mechanism reveals that interaction between the replacement and pre-fetching mechanism converges the state of the cache to the efficient state. Thus the quality of popular cache streams is only limited by the available bandwidth between the proxy and its clients. Furthermore, the quality of any cached stream is directly determined by its popularity.

As part of our future work, we plan to conduct more simulations and examine the effect of other background traffic and simultaneous access to cached streams as well as VCR-functions on replacement dynamics, and bandwidth sharing among concurrent pre-fetching session. We would also like to explore caching aspects of our replacement algorithm, i.e. reduction in the load on the network and the server.

## REFERENCES

- [1] Sandeep Bajaj et al. Virtual InterNetwork Testbed: Status and research agenda. Technical Report 98-678, University of Southern California, July 1998.
- [2] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS*, pages 151–160, June 1998.
- [3] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. On the implications of Zipf’s Law for Web caching. In *Proceedings of The Third International WWW Caching Workshop*, 1998.
- [4] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 193–206, December 1997.
- [5] A. Dan and D. Sitaram. Multimedia caching strategies for heterogeneous application and server environments. *Multimedia Tools and Applications*, 4:279–312, 1997.
- [6] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *Under submission*, February 1998.
- [7] S. Ghandeharizadeh and C. Shahabi. On multimedia repositories, personal computers, and hierarchical storage systems. In *Proceedings of ACM Multimedia Conference*, 1994.
- [8] Microsoft Inc. Netshow service, streaming media for business.
- [9] S. Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the Annual ACM Symposium on the Theory of Computing*, March 1997.
- [10] M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In *Proceedings of the 4th International Conference on Database Systems for Advanced Applications*, April 1995.
- [11] Progressive Networks. Http versus realaudio client-server streaming. <http://www.realaudio.com/help/content/http-vs-ra.html>.
- [12] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. *ACM SIGCOMM*, September 1998.
- [13] R. Rejaie, M. Handley, and D. Estrin. Architectural considerations for playback of quality adaptive video over the. Technical Report 98-686, USC-CS, November 1998.
- [14] R. Rejaie, M. Handley, and D. Estrin. Quality adaptation for congestion controlled playback video over the internet. *Proceedings ACM SIGCOMM*, September 1999.
- [15] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. *Proceedings IEEE Infocom*, March 1999.
- [16] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy caching mechanism for multimedia playback streams in the internet. *The 4th International Web Caching Workshop*, March 1999.
- [17] Reza Rejaie, Haobo Yu, Mark Handley, and Deborah Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet. Technical Report 99-XXX, USC-CS, 1999.
- [18] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. Technical Report RN/98/13, UCL-CS, 1998.
- [19] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of the IEEE Infocom*, 1999.
- [20] R. Tewari, H. Vin, A. Dan, and D. Sitaram. Resource based caching for web servers. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, San Jose, 1998.
- [21] M. Vishwanath and P. Chou. An efficient algorithm for hierarchical compression of video. *Proc IEEE Intl. Conf. Image Processing*, November 1994.
- [22] Y. Wang, Z.-L. Zhang, D. Du, and D. Su. A network conscious approach to end-to-end video delivery over wide area networks using proxy servers. In *Proceedings of the IEEE Infocom*, April 1998.
- [23] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM SIGCOMM*, pages 293–305, 1996.
- [24] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Proceedings of the Sixth International WWW conference*, April 1997.