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

Reza Rejaie[1], Haobo Yu[1], Mark Handley[2], Deborah Estrin[1]
[1]USC/ISI [2] 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 available bandwidth. 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 causes the state of the cache to converge 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 Internet multimedia streaming applications require that a server maintains a large number of multimedia streams and pipeline a stream to a client upon request. The quality (*i.e.,* bandwidth) of the delivered stream in such an approach is limited to the bottleneck bandwidth between the server and the client. If the server is behind a bottleneck link, a client with high bandwidth local connectivity to the network may receive low quality streams even though it has paid for a high bandwidth local link.

This paper explores an adaptive solution to this problem using multimedia proxy caching [1]. A proxy cache resides close to a group of clients. Requested streams are always delivered from the original servers through the proxy to clients, thus the proxy is able to intercept and cache these streams. The proxy can significantly increase the delivered quality of popular streams to high bandwidth clients despite the presence of a bottleneck on its path to the original server.[1] Compared to other quality-enhancing approaches, such as mirror servers, proxy caches are more adaptive and have lower storage and processing requirements. Thus they are more cost-effective and can be widely deployed.

A primary challenge for multimedia proxy caching in the Internet is the requirement 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 [2]. Since a dominant portion of today's Internet traffic consists of TCP-based flows, such as HTTP, TCP-friendly congestion control [3] is prefered.

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 for subsequent requests but it still needs to perform congestion control and qual-

This work was supported by DARPA under contract No. DABT63-95-C0095 and DABT63-96-C-0054 as part of SPT and VINT projects

[1] Certainly, if a 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, even in this case a proxy cache is still able to significantly reduce startup delay, facilitate VCR-functionalities, and reduce load on the server and network. Note that these benefits are also applicable to high bandwidth clients.

ity 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 changes in available bandwidth, from previous sessions. Thus, quality variations of the cached stream are not correlated with the quality variations required by quality adaptation during the new playback.

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 [4]. TCP-friendly congestion control is performed using Rate Adaptation Protocol(RAP)[5]. Hierarchical encoding [6] is used to provide a layered approach to quality adaptation. With hierarchical encoding, each stream is split into a base layer that contains the most essential low quality information, and higher layers provide optional quality enhancement information.[2]

Layered organization provides an opportunity for proxy caches to adjust the quality of a cached stream in a demand-driven fashion. To allow fine-grain adjustment of quality, each layer of the encoded stream is divided into equal-sized pieces called *segments*. The proxy then prefetches the segments that are required by the quality adaptation mechanism and are missing in the cache. If the available bandwidth between the proxy and a client can support a stream with higher quality, higher layers are then gradually prefetched from the server. Thus the quality of the cached stream is adjusted based on its popularity. Main contributions of this paper are novel prefetching and fine-grain cache replacement algorithms for multimedia proxy caching. The interaction of the two algorithms causes the state of the cache to converge to an *efficient state* such that the quality of a cached stream is proportional to its popularity, and its quality variations are inversely proportional to its popularity.

Fig. 1 shows an extension of our end-to-end client/server architecture to include proxy caches. All streams are layered encoded and stored at the server's archive. Traffic is always routed through a corresponding proxy cache that is associated with a group of clients in its vicinity, thus a proxy is able to intercept each stream and cache it. We also assume that all streams between servers and proxies, and between proxies and clients, must perform congestion control and quality adaptation. We do not make any assumption about the inter-cache architecture that addresses control message exchange and request forwarding. Our caching mechanisms are orthogonal to the design of such architectures.

[2]Throughout this paper, we will assume *linear-layered* encoding where all layers have the same bandwidth for the sake of clarity. However, our architecture and caching mechanisms can be extended to other layered-encoding bandwidth distributions.

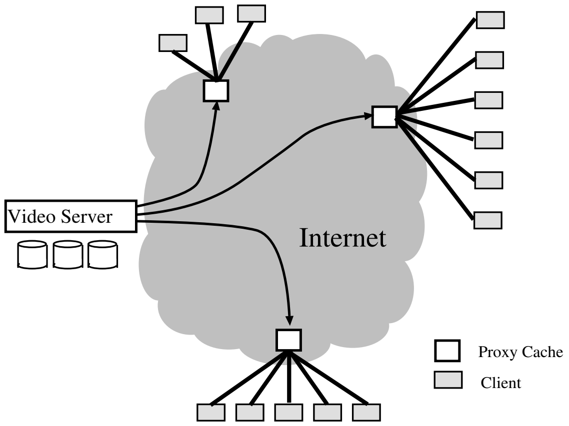


Fig. 1. The end-to-end server/proxy/client architecture

The rest of this paper is organized as follows: In Section II we present the delivery procedures on cache-miss and cache-hit scenarios. We will also discuss our prefetching algorithm during the cache hit scenario to cope with variations in quality. In Section III, we discuss different aspects of our replacement algorithm including replacement patterns and popularity function. We present our simulation environment and some of our simulation results in Section IV. Section V summarizes some of the related works and finally Section VI concludes the paper and addresses 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 checks the availability of the requested stream. The rest of the delivery procedure varies for cache miss or a cache hit. Next we describe each scenario separately.

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 the inter-cache architecture. For simplicity we assume that the stream is played back from the original server to the proxy via a RAP connection. The proxy then relays data packets toward the client through a separate RAP connection. Thus the server is able to perform end-to-end congestion control and quality adaptation based on the state of the session between the server and the proxy. The quality of the delivered stream is limited by the average bandwidth between the server and the proxy. In the cache miss scenario, the client does not observe any benefit (*e.g.,* improvement in quality) 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

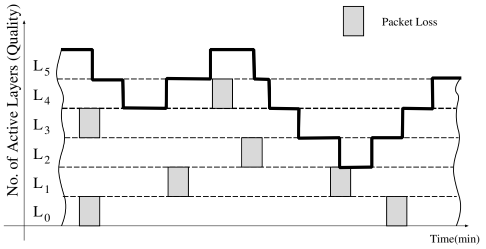


Fig. 2. A sample quality adaptive stream in the cache

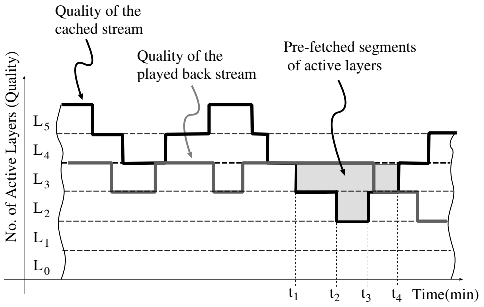


Fig. 3. Delivery of lower bandwidth stream from the cache

cache to make room for the new stream. Details of the replacement algorithm are discussed in section III. After the first playback, there might be occasional missing packets in the cached stream that were lost but not repaired. Fig. 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 proactively prefetching the missing segments during idle hours. Alternatively, the proxy may prefetch missing segments on a demand-driven fashion while it serves subsequent requests for the cached stream. We adopted the latter approach because the future request pattern may not be predictable. We plan to investigate proactive prefetching as part of our future work.

B. Prefetching on a Cache Hit

On a cache hit, the proxy acts as a server and starts playing back the requested stream. As a result the client observes shorter startup latency. The proxy must still perform congestion control and quality adaptation. As discussed before, the quality variations in the cached stream may not match those required by the quality adaptation mechanism at the proxy. This means that the quality adaptation mechanism may attempt to send some segments that do not exist in the cache. To maximize the delivered quality to the client, the proxy should prefetch these missing segments from the server ahead of time.

During each interval of the session two scenarios are possible:

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

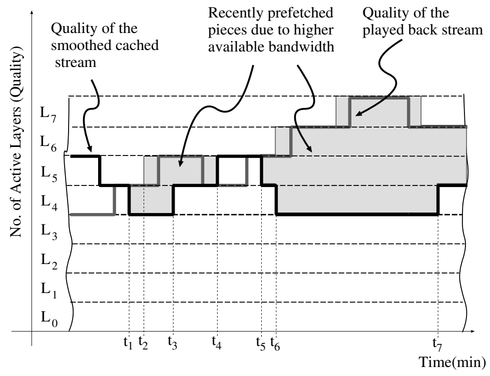


Fig. 4. Delivery of higher bandwidth stream from the cache

where $Playback_{avgBw}$ and $Stored_{avgBw}$ denote average bandwidths 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. Fig. 3 depicts the first scenario. Even if $Playback_{AvgBw} \leq Stored_{AvgBw}$, there may still exist segments that are required by quality adaptation but are missing from the cache. 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 such examples. The second scenario is shown in Fig. 4. In this case, the proxy not only needs to prefetch missing segments of lower layers, it also needs to occasionally prefetch higher layers as soon as quality adaptation indicates the possibility of adding a new layer in the future. All the prefetched 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 prefetching and (ii) the connection between the proxy and the client for delivery of the stream. Prefetching 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 quality adaptation in the future. Since quality adaptation adjusts the number of active layers according to random changes in available bandwidth, the time for the upcoming adjustment is not known *a priori*. Thus there exists a tradeoff: the earlier the proxy prefetches a missing segment, the less accurate is the prediction, but the higher is the chance of receiving the prefetched segment in time. Prefetched segments are delivered in a congestion controlled fashion from the server to the proxy. Therefore, the server should deliver the requested segments based on their priority, otherwise the prefetching stream will fall behind the playback stream and become useless. Note that the prefetched segments are always cached even if they arrive after their playout times.

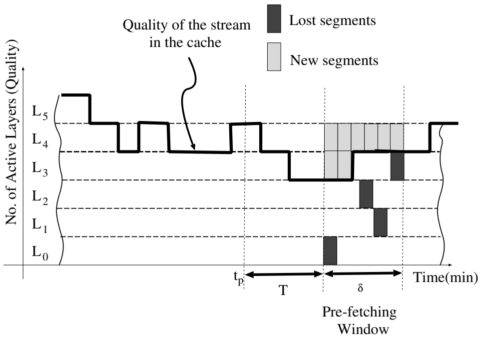


Fig. 5. Prefetching mechanism

We propose a prefetching algorithm that addresses the above problem, as illustrated in Fig. 5. The proxy maintains a playout time for each active client. At playout time t_p, the proxy examines the interval $[t_p + T, t_p + T + \delta]$, which is called the *prefetching window* of the stream, and identifies all missing segments within this window. These segments are missing due to either packet losses or layer drops in the previous playbacks. Furthermore, if the quality adaptation mechanism is going to add a new layer[3], all missing segments of the new layer within the prefetching window are also included. The proxy then sends a single request to the server to prefetch all these missing segments in a batch. To loosely synchronize the prefetching stream with the playback stream, the prefetching window should slide as fast as the playout point. Thus after δ second, the proxy examines the next prefetching window and sends a new prefetching request to the server.

When the server receives a prefetching request, it starts to send the requested segments based on their priorities (*i.e.,* layer numbers). For example, it first sends all the requested segments of layer 0, then those of layer 1, and so on. A new prefetching request preempts the previous one. If the server receives a new prefetching request before delivery of all the requested segments in the previous request, it ignores the old request and starts to deliver segments in the new request. This preempting mechanism causes the prefetching and the playback to proceed with the same rate. Notice that the average quality improvement of a cached stream after each playback is determined by the average prefetching bandwidth. Thus it may take several playbacks for the stream's quality to reach the maximum that the clients can receive (assuming no replacement).

To avoid multiple prefetching sessions between a server and a proxy, all the prefetching sessions must be multiplexed into a single congestion controlled flow. This implies that as the number of prefetching sessions increases, the allocated prefetching bandwidth share for each session decreases.

[3]see [7] for details on conditions for adding a new layer

III. Replacement Algorithm

The chief goal of our caching mechanism is to converge the cache state to an efficient state. The state of the cache is efficient if the following conditions are met:

- The *average quality* of each cached stream is directly proportional to its popularity. Furthermore, the average quality of the stream must converge to the average bandwidth across most recent playbacks.
- The *quality variations* of each cached stream are inversely proportional to its popularity.

In this section we present a replacement algorithm that helps to achieve this goal. We start by describing coarse and fine-grain replacement patterns for layered-encoded stream. Then we extend the traditional semantics of a "hit" and define a simple popularity function that captures the level of interest among users who interactively perform VCR-functionalities.

A. Replacement Pattern

Ordinary Web pages do not have a sub-structure. Consequently each page is considered atomic, *i.e.,* a client requests and receives the entire page. Thus most existing replacement algorithms for Web caching make a binary replacement decision, *i.e.,* pages are cached and flushed in their entirety. Layered encoded streams are structured into separate layers, and each layer is further divided into segments with a unique ID. This organization allows us not only to make *multi-valued* replacement decisions but also to perform replacement with different granularities. As the popularity of a cached stream decreases, its quality—and consequently its size—is gradually reduced before it is completely flushed out.

Fig. 6 depicts the replacement pattern of a cached stream. The coarse-grain replacement is achieved by dropping the entire layer with the lowest popularity, 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 prefered to cache a contiguous portion from the beginning of a layer to absorb startup latency [8] and minimize the variations in quality. Thus once a victim layer is identified, its cached segments are flushed from the end. If flushing all segments of the victim layer does not provide sufficient space, the proxy then identifies a new victim layer and repeats this process.[4]

[4]Note that this segment-based replacement may result in thrashing. For example, the tail of the victim layer (e.g. L_3) is flushed to make room for a higher layer (e.g. L_4) of the same stream when available bandwidth suddenly increases. To avoid this, while a particular stream is played back from the cache, its active layers are locked in the cache and can not be replaced during the playback.

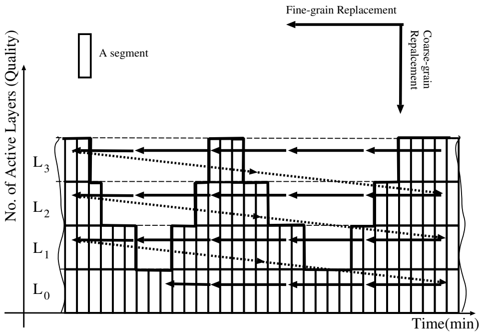


Fig. 6. Replacement priority within a cached stream

B. Popularity Function

Most of the current Web caching schemes assign a binary value to a hit, *i.e.,* 1 for each request. This model perfectly suits the atomic nature of interest in web pages. 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 playback time of 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 stream. 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[5]:

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

where $PlaybackTime$ and $StreamLength$ denote total playback time of a session and length of the entire stream, respectively. Both $PlaybackTime$ and $StreamLength$ are measured by time (*e.g.,* second). Notice that adding and dropping layers by quality adaptation results in different $PlaybackTime$s in a session and consequently affects the popularity of a cached layer. For example, even if all layers of a stream are available in the cache and the client watches the entire stream, quality adaptation may only send Layer 0, 1 and 2 for 100%, 80%, 50%, of the playback time respectively.

The proxy calculates weighted hits on a per-layer basis for each playback. 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 (called the *popularity window*) is used as the popularity index of the layer. The popularity of each layer is recalculated at the end of a session as follows:

$$P = \sum_{x=t-\Delta}^{t} whit(x) \tag{2}$$

where P and Δ 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 encoding guarantees that popularity of different layers in the same stream monotonically decrease with the layer number[6]. Thus a victim layer is always the highest in-cache layer of one of the cached streams. Notice that the length of a layer does not affect its popularity, because $whit$ is normalized by length.

P	Lock	Stream name	Layer no.
5.85	1	$Titanic$	L_0
4.92	1	$Titanic$	L_1
4.76	0	$Amistad$	L_0
3.33	0	$Apollo$ 13	L_0
2.30	0	$Titanic$	L_2
1.28	0	$Amistad$	L_1

TABLE I
SAMPLE OF A POPULARITY TABLE

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. In order to hide startup latency, the first few segments of the base layer for each cached stream are kept in the cache for a long period even though the popularity of this layer becomes low.

IV. SIMULATION

We evaluate our multimedia caching mechanism via simulation using the *ns* [10] simulator. We use RAP [5] (for congestion control) along with layered quality adaptation [7] as transport protocol for multimedia streams. Note that we did not include error control mechanisms.

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

[5]The term "weighted hit" has been used in caching literature [9] to take into account of page sizes. Here we extended the definition of this term into the context of multimedia stream caching.

[6]The encoding constraint requires that to decode a segment of layer i, corresponding segments for all lower layers must be available.

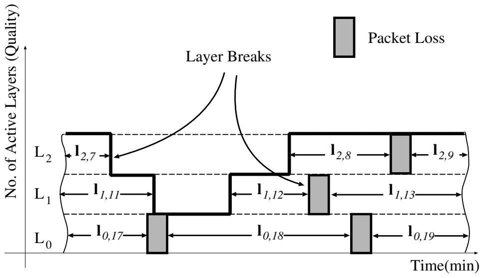


Fig. 7. Average quality and layer breaks for a cached stream

and how this mechanism satisfies these design goals. It remains as future work to examine the *performance* of this mechanism, such as the byte hit ratio and the convergence of our replacement algorithm, under realistic background traffic.

A. Evaluation Metrics

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

- *Completeness* measures the percentage of a stream residing in the cache. This metric allows us to trace the quality evolution of a cached stream after each playback. Because our replacement algorithm is layer-based, we define the completeness on a per-layer basis. The completeness of layer l in cached stream s is defined as the ratio of the layer's size in cache to its "official size":

$$Cp(s,l) = \frac{\sum_{\forall i \in Chunks(l)} L_{l,i}}{RL_l} \quad (3)$$

Here 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 (in terms of segments) of the ith 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]. If every byte of a stream is cached, each of its layers has completeness value of 1.

- *Continuity* measures the level of smoothing of a cached stream. Completeness alone does not capture this, because it does not reflect the number of "holes" in a cached stream. Continuity is also defined on a per-layer basis. The continuity of layer l in cached stream s is defined as the average number of bytes between two consecutive layer breaks, *i.e.,* average chunk size:

$$Ct(s,l) = \frac{\sum_{\forall i \in Chunks(l)} L_{l,i}}{LayerBreak + 1} \quad (4)$$

A layer break occurs when there is a missing segment in a layer. It may be due to either quality adaptation dropping a layer or a packet loss (Fig. 7). Including error control in the transport protocol will speed up the prefetching process to fill these missing segments. In the absence of error control, our results shown here represent the worst case scenarios.

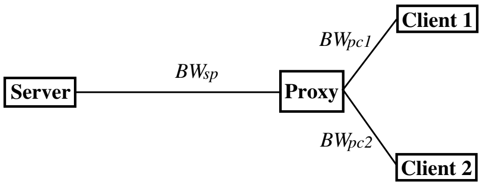


Fig. 8. Simulation topology

B. Simulation Setup

We have conducted two sets of simulations. The first focuses on evaluating the prefetching algorithm, and the second on the replacement algorithm. They both use a simple network topology shown in Fig. 8. BW_{sp} denotes the average available bandwidth between server and proxy whereas BW_{pc1} and BW_{pc2} are physical link bandwidths between proxy and two clients, respectively. One may construct two interesting scenarios from this simple topology:

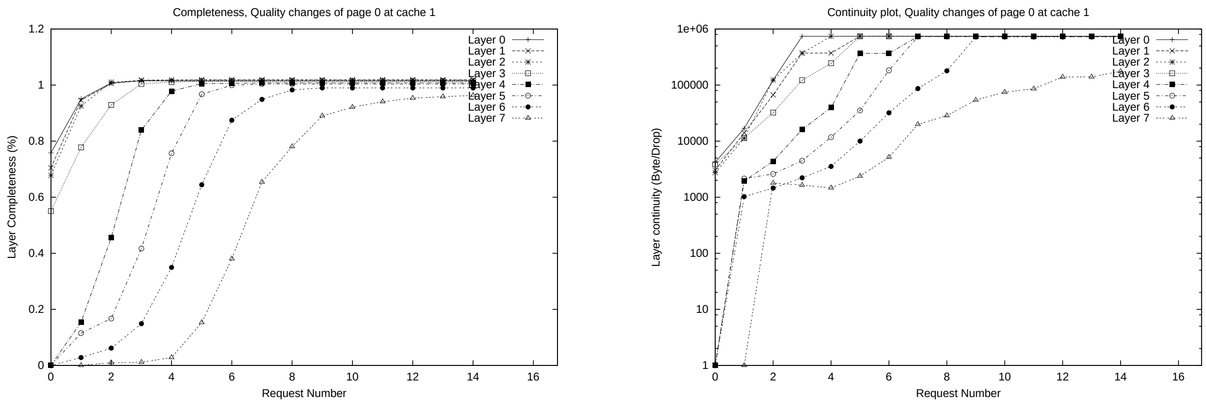
- Scenario I: $BW_{sp} < BW_{pc1}$, the server-proxy connection is the bottleneck.
- Scenario II: $BW_{sp} \geq BW_{pc1}$, the proxy-client connection are the bottleneck.

We are particularly interested in scenario I because in scenario II the quality of cached streams will always be higher than what the client can afford and leave no room for the proxy to gradually improve the quality. When there are multiple clients, it is interesting to mix 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 cached streams, as we will discuss later in this section.

There are other parameters controlling our simulations, such as cache capacity, segment size, etc. To limit the number of parameters, we let all streams have 8 layers, the same segment size of 1KB, and layer consumption rate of 2.5KB/s. Changing these parameters will not qualitatively change our results as long as they are changed proportionally.

In all simulations the server-proxy link is shared by 10 RAP and 10 long-lived TCP flows (carrying FTP traffic). One of the RAP flows is used to deliver multimedia streams from server to cache; the other 19 flows present background traffic, whose dynamics results in available bandwidth changes that will trigger adding and dropping of layers. Bandwidth of the server-proxy link is set to 1.12Mbps (20*56Kbps). Since RAP is TCP-friendly, each flow obtains an even share of bandwidth on average, thus the average bandwidth of the server-proxy connection (BW_{sp}) is 56 Kbps which can afford 2.8 layers.

In order to generate a request sequence, we need to know two factors: number of requests for each stream (*i.e.,*



(a) Completeness (b) Continuity

Fig. 9. Quality improvement due to prefetching.

stream popularity), and temporal distribution of these requests. First, we assume that the stream popularity conforms to the Zipf's law, which was observed in web page requests [11].[7] Given the number of total requests R and total number of streams N, we let the mth popular stream have $\frac{1}{m}\Omega R$ requests, where $\Omega = \sum_{i=1}^{N} \frac{1}{i}$. Second, it is non-trivial to generate a request sequence that exhibits temporal locality while the stream popularity follows the Zipf's law [12]. A request sequence that lacks temporal locality may lead to different cache performance, *e.g.,* hit ratio. Since performance analysis is not our primary concerns, 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. This excludes the situations of simultaneous playbacks from the cache. However, the only added complexity in these situations is that two or more streams compete for the prefetching bandwidth between the server and the proxy; ignoring this does not affect the evaluation of correctness of our algorithms. The distribution of requests between the two clients that have different bandwidth to the proxy varies in simulations as we will discuss it next.

C. Results

C.1 Prefetching

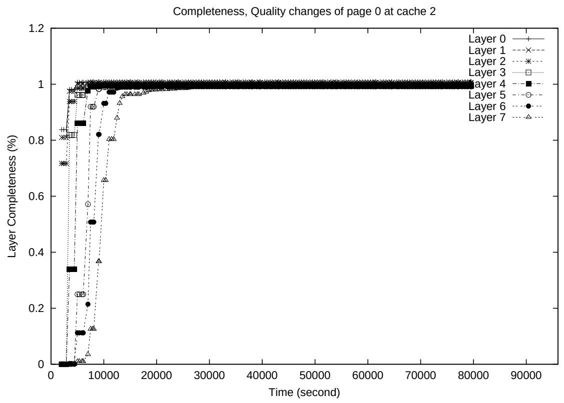
This simulation is intended to demonstrate that the prefetching algorithm results in gradual improvement in quality of cached streams. To disable the cache replacement mechanisms, we set cache size to infinity and use only a single client and a single stream. The stream size is set to 5 minutes. As discussed above, we choose scenario I where BW_{sp} is 56Kbps and BW_{pc1} is 1.5Mbps. The simulation ran for 125 minutes, containing 15 completed requests.

Fig. 9 shows the evolution of completeness and continuity of every layer of the cached stream. Each point in the figure represents the status of a particular layer after one playback. Since continuity inversely depends on the number of layer break, we plot it 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 playbacks it takes to improve the layer's quality. The convergence speed is not constant, rather, it exhibits a thresholded pattern. For each layer, there are several requests that greatly enhanced its quality, while other requests just marginally improve its quality. This can be explained by the layer dependence during prefetching: a higher layer is only prefetched when corresponding data of all lower layers are available. Currently we only repair for packet losses by prefetching. We expect that the convergence of continuity will be much faster if an error control mechanism is provided by the transport protocol.

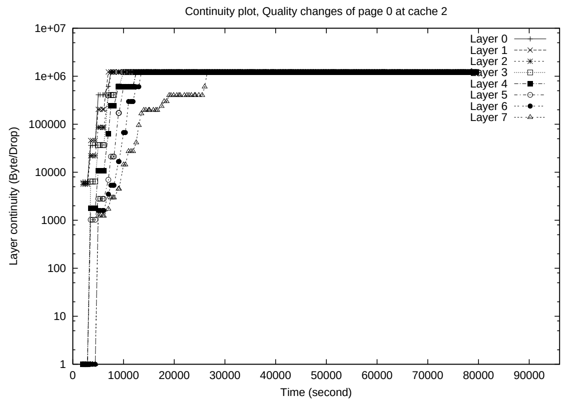
C.2 Replacement Algorithm

This set of simulations is intended to examine that the state of the cache gradually converges to an efficient state as result of the interaction between prefetching and replacement algorithms. The resulting quality due to cache replacement depends on two factors: stream popularity and the bandwidth between the requesting client and the proxy. If we assume $BW_{sp} \gg BW_{pc1}$, stream popularity will be dominant. In the other extreme, $BW_{sp} \leq BW_{pc1}$, the client bandwidth overshadows stream popularity. To examine the impact of different parameters, we studied both extremes and an intermediate case using two clients with different bandwidths to the proxy. BW_{sp}, BW_{pc1} and BW_{pc2} are set to 56Kbps, 1.5Mbps and 56Kbps respectively. By distributing the number of requests between

[7] There exist many web traces that do not exactly follow Zipf's law, instead, they exhibit *Zipf-like behavior* [11]. Here, we use Zipf's law for simplicity.

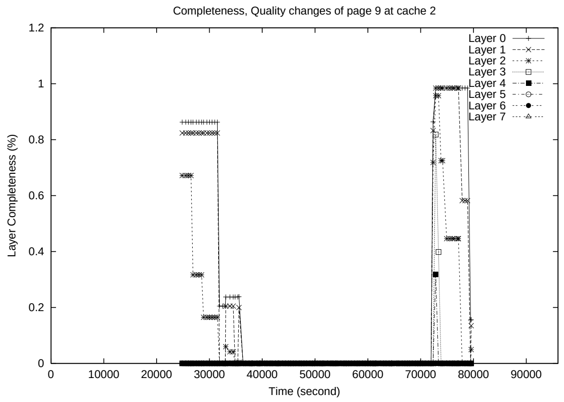


(a) Completeness

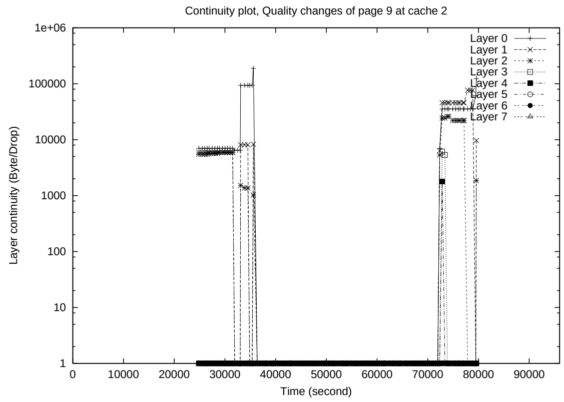


(b) Continuity

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



(a) Completeness



(b) Continuity

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

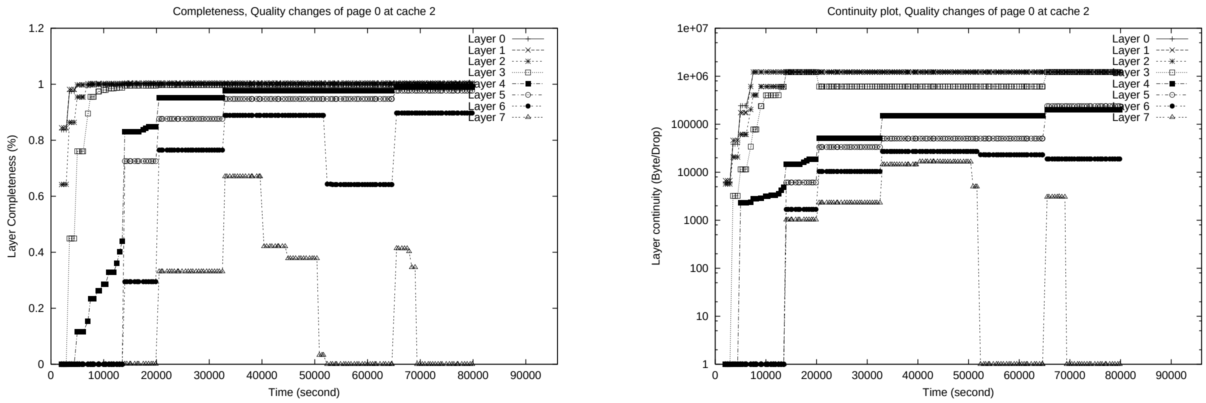
client 1 and client 2, we are able to go from one extreme ($BW_{sp} \ll BW_{pc1}$) to the other ($BW_{sp} \geq BW_{pc2}$).

Without statistical knowledge about the size distribution of real Internet multimedia streams, we choose 10 streams with lengths uniformly distributed between 1 and 10 minutes(the size in byte can be obtained by multiplying the stream length, number of layers and layer consumption rate). Their popularity decreases with their index, *i.e.,* stream 0 is the most popular one. Streams longer than 10 minutes can be viewed as combinations of several shorter streams with the same popularity.

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. 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 background traffic (using a statistically realistic bursty Web traffic model [13]), cache size and the number of streams on the resulting quality evolution. The results suggested that the conclusions presented below are still applicable when those parameters are changed. Due to space limitation, we refer readers to [14] for more details.[8]

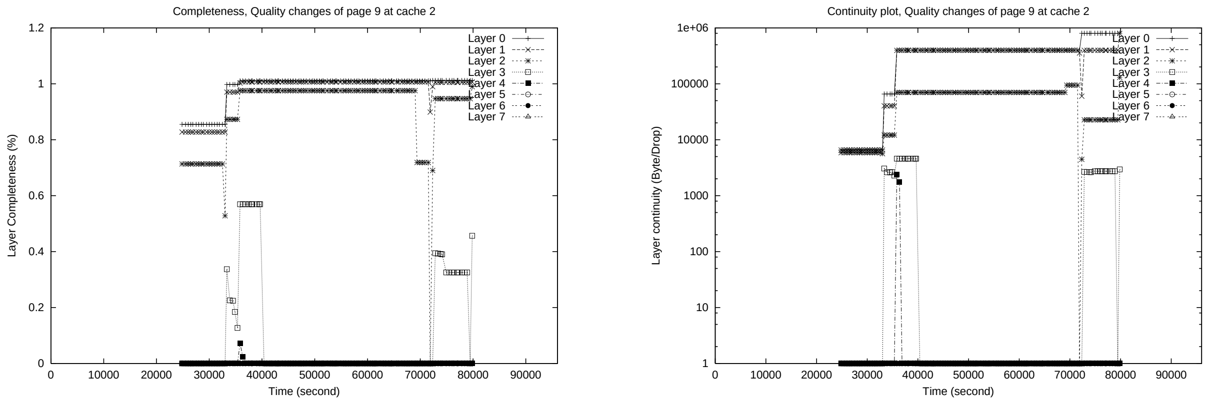
C.2.a **Effect of Popularity.** In order to emphasize the influence of stream popularity on cache replacement, we should reduce the influence of client bandwidth to the minimum. We achieved this by distributing 95% of all requests to the high bandwidth client 1 and only 5% to client 2. The simulation ran for 44 (virtual) hours and contains 310 requests. We only show the first half (*i.e.,* first 22 hours); the rest exhibits the same trend and is omitted for clarity. Note that time is used as x-axis in these figures. With replacement, a stream's quality not only changes with its own requests but also with requests to other streams. It is easier to represent this relationship using time as x-axis.

[8]In all of these simulations, we maintain an infinite popularity window, because we expect that in reality the popularity 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.



(a) Completeness (b) Continuity

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



(a) Completeness (b) Continuity

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

Figs. 10 and 11 show the quality change of the most popular stream 0 and the least popular stream 9, respectively. Stream 0 is eventually able to keep all of its layers in the cache after 20 requests. As a comparison, stream 9 is not able to keep adequate quality in the cache, and it was even completely flushed out during the interval [36000s, 72000s]. These figures showed qualities of popular streams are gradually improved. They are likely to have more layers in cache, and these layers tend to have higher quality both in terms of completeness and continuity. In other words, our algorithms successfully accomplished their goals.

C.2.b **Effect of Client-side Bandwidth.** Since we calculate popularity on a per-layer basis, if most clients who requested a stream have limited bandwidth, only the lower layers of the stream should be kept in cache. In order to examine this effect of client-side bandwidth on replacement, we assigned 95% of all requests to the low bandwidth client 2 and 5% to the high bandwidth client 1. Fig. 12 shows the quality change of the most popular stream, and Fig. 13 shows that of the least popular stream.

Comparing Fig. 12 with Fig. 10 reveals that when clients have limited bandwidth, the maximum quality of the popular stream drops significantly. In the previous case where most requests came through a 1.5Mbps link, the most popular stream 0 could keep all 8 layers in the cache. However, in this case it is able to cache only 4 layers as most requests come from a 56Kbps link. To the contrary, now the least popular stream 9 has improved quality and is able to keep the 2 lowest layers in cache most of the time. This is because the higher layers of more popular streams were accessed less frequently, thus the lower layers of the less popular streams became relatively more popular and are able to stay in cache.

C.2.c **Mixing Together.** Having examined the above two extreme cases, we now examine an intermediate case where requests are evenly distributed between the low bandwidth client and the high bandwidth client. Figs. 14 and 15 show the quality change of the most popular stream 0 and the least popular stream 9, respectively. Comparing Fig. 14 to Figs. 10 and 12, we found that the average quality of the popular stream is higher than that in the low bandwidth client case, but lower than that in the high bandwidth client case. A similar situation is observed for the least popular stream. This is exactly what our cache replacement algorithm is intended to achieve, *i.e.,* converging the resulting quality of cached streams to the average quality that has been accessed by clients. Notice that the quality convergence here is closer to the high bandwidth case (Fig. 10) than the low bandwidth case (Fig. 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 [15], [16]. The idea is to reduce disk access by grouping requests and retrieving a single stream for the entire group. In other words, these approaches tried to minimize the object migration among different levels of a hierarchical storage system, *i.e.,* tertiary, disk and memory. There is an analogy between memory caching and proxy caching. In a sense, object migration in a hierarchical storage system is similar to stream transmission among server, proxy and client. However, there is a fundamental difference between them: the available bandwidth between two levels of a hierarchical storage system is fixed and known *a priori*, whereas the available bandwidths among server, proxy and client randomly change with time. As a result, one can not simply apply these memory caching schemes for proxy caching of Internet streams.

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 [17] prefetches and stores portions of a stream in proxies, and later uses them to smooth its playback. Another scheme [8] stores prefixes of multimedia streams in proxy caches to improve startup latency as well as perform smoothing. Our work is complementary to work on smoothing. Smoothing does not address congestion control of Internet multimedia streams while our algorithms are aware of and utilize underlying congestion control mechanism.

There are numerous works on proxy cache replacement algorithms [9], [18]. It is not clear how they will behave when a request sequence contains a significant number of requests to huge multimedia streams. To our knowledge, there is only one work that addresses the influence of multimedia streams on cache replacement algorithms [19]. 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.

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 quality variations of cached streams in a demand-driven fashion by prefetching 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 prefetching mechanism causes the state of the cache to converge to an efficient state. In such a state, the quality of every cached stream is directly determined by its popularity, and its upper limit is defined by the average available bandwidth between the proxy and its clients. As part of our future work, we plan to conduct more simulations and examine the effect of popularity window, simultaneous access to cached streams, and bandwidth sharing among concurrent pre-fetching sessions as well as VCR-functions.

References

[1] R. Rejaie, M. Handley, H. Yu, and D. Estrin, “Proxy caching mechanism for multimedia playback streams in the Internet,” in *Fourth International WWW Caching Workshop*, Mar. 1999.

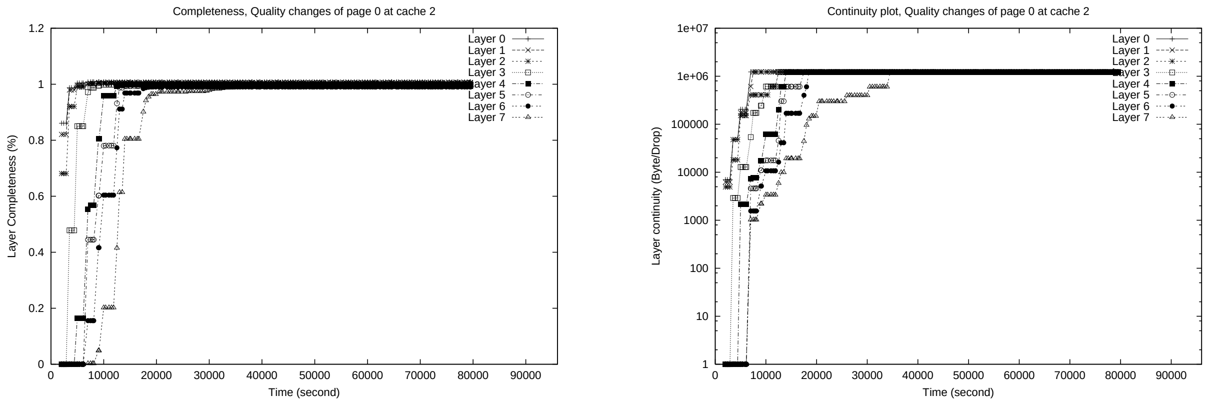
[2] S. Floyd and K. Fall, “Promoting the use of end-to-end congestion control in the Internet,” *Under submission*, Feb. 1998.

[3] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: a simple model and its empirical validation,” *Proc. ACM SIGCOMM*, Sept. 1998.

[4] R. Rejaie, M. Handley, and D. Estrin, “Architectural considerations for playback of quality adaptive video over the Internet,” Tech. Rep. 98-686, USC-CS, Nov. 1998.

[5] R. Rejaie, M. Handley, and D. Estrin, “RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet,” *Proc. IEEE Infocom*, Mar. 1999.

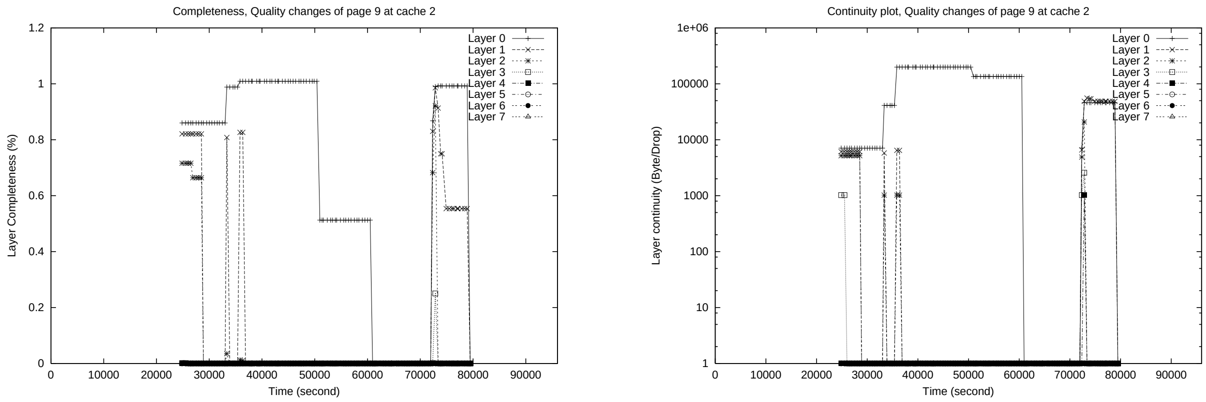
[6] M. Vishwanath and P. Chou, “An efficient algorithm for hierarchical compression of video,” *Proc. IEEE International Conference in Image Processing*, Nov. 1994.



(a) Completeness

(b) Continuity

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



(a) Completeness

(b) Continuity

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

[7] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled playback video over the Internet," *To appear Proc. ACM SIGCOMM*, Sept. 1999.

[8] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE Infocom*, 1999.

[9] P. Cao and S. Irani, "Cost-aware www proxy caching algorithms," in *Proc. USENIX Symposium on Internet Technologies and Systems*, Dec. 1997, pp. 193–206.

[10] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, S. Shenker, K. Varadhan, H. Yu, Y. Xu, and D. Zappala, "Virtual InterNetwork Testbed: Status and research agenda," Tech. Rep. 98-678, University of Southern California, July 1998.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the implications of Zipf's Law for Web caching," in *Proc. IEEE Infocom*, 1999.

[12] P. Barford and M. Crovella, "Generating representative web workloads for network and server peformance evaluation," in *Proc. ACM SIGMETRICS*, June 1998, pp. 151–160.

[13] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *To apprear Proc. ACM SIGCOMM*, Sept. 1999.

[14] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet," Tech. Rep. 99-708, USC-CS, May 1999.

[15] A. Dan and D. Sitaram, "Multimedia caching strategies for heterogeneous application and server environments," *Multimedia Tools and Applications*, vol. 4, pp. 279–312, 1997.

[16] M. Kamath, K. Ramamritham, and D. Towsley, "Continuous media sharing in multimedia database systems," in *Proc. 4th International Conference on Database Systems for Advanced Applications*, Apr. 1995.

[17] 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 *Proc. IEEE Infocom*, Apr. 1998.

[18] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, "Removal policies in network caches for world-wide web documents," in *Proc. ACM SIGCOMM*, 1996, pp. 293–305.

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