Proxy Caching for Quality Adaptive Multimedia Streams in the Internet: A Performance Perspective

Reza Rejaie
AT&T Labs-Research
reza@research.att.com

Haobo Yu
USC/ISI
haoboy@isi.edu

Abstract—Multimedia proxy caching (MCaching) presents a cost-effective solution to support large scale access to high quality multimedia streams over the Internet. However, introducing the notion of “quality” of cached streams adds a new dimension to the evaluation space and complicates the problem. This paper proposes a comprehensive framework for the evaluation of multimedia proxy caching mechanisms. We identify the fundamental tradeoff between stream quality improvement and caching efficiency across the evaluation space. We then introduce two sets of performance metrics in this space to effectively capture various aspects of quality evolution and cache efficiency at both the per-stream level and the aggregate level.

We have applied this framework to conduct simulation-based evaluation of multimedia caching mechanisms. At the aggregate level, our results show that MCaching efficiently utilizes cache space and adaptively maximizes overall performance along both dimensions of the evaluation space. At the per-stream level, we provided insights in interactions among quality evolution, prefetching and caching efficiency, and congestion control. Our results reveal that prefetching and caching efficiency are directly determined by the difference between cached quality and deliverable quality.

I. INTRODUCTION

The explosive growth of commercial usage in the Internet has resulted in a rapidly increasing 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 pipelines a stream to a client, i.e., the client is playing back the available portion of stream while the rest of it is being delivered.

There are two major obstacles in supporting high quality streaming applications at a large scale over the Internet:

• High Bandwidth Flows: High quality multimedia streams usually consume high bandwidth. For example, an MPEG-2 stream may require as much as 4 Mbps bandwidth and may last as long as two hours. This raises scalability concerns in terms of both network load and server load.

• Variability of Available Bandwidth: Today’s Internet is best-effort. The quality (i.e., bandwidth) of a stream delivered to a client is limited by the bottleneck bandwidth along the path to the server. Thus a client with a high bandwidth local access may receive low quality streams due to a remote bottleneck, even though it has paid a premium for a high bandwidth local link.

Multimedia proxy caching (MCaching) [1] is an adaptive solution that addresses both problems simultaneously. By caching popular streams with appropriate quality at a proxy close to the interested clients, subsequent requests for these streams can be replayed directly from the cache. Similar to Web caching, MCaching can significantly reduce the load on the network and the server, thereby improving scalability. Furthermore, the quality of the stream delivered from the proxy to the client is only limited by the available bandwidth between the proxy and the client (i.e., the last hop(s))\(^1\). Caching popular streams close to interested clients also significantly reduces startup delay and facilitates interactive VCR-functionalities. Compared with other quality-enhancing approaches (e.g., mirror servers), proxy caches are more cost-effective and can be widely deployed.

MCaching introduces a new dimension of caching performance that does not exist in current Web caching schemes, namely the quality of cached streams. MCaching not only manages to keep popular streams in the cache, but also tries to maintain appropriate quality for each cached stream. Its goal is to maximize both the per-stream quality and the aggregate performance of the cache at the same time.

MCaching works as follows. All the requests and responses are “routed” through the proxy, as they are in traditional Web caches. On a cache miss, the request is forwarded to the original server (or a neighbor cache depending on the inter-cache architecture). The stream is played back from the server to the cache, and simultaneously relayed from the cache to the client. The quality of the cached stream after the initial playback is thus limited by the bottleneck between the server and the cache. On a cache hit, the stream is played back directly from the cache. If the client has sufficient bandwidth to afford higher quality than what is available in the cache, the missing higher quality portion is incrementally prefetched from the server in a demand-driven fashion. Consequently, the more high-bandwidth clients request a stream, the better its quality becomes in the cache. If a cached stream (or its high-quality portion) is not frequently used, the replacement algorithm gradually flushes it out to make room for new streams or higher quality portions of other streams. Thus MCaching introduces partial prefetching and fine-grain replacement, as opposed to atomic replacement used in traditional Web caching.

Stream quality adds a new dimension into the evaluation space of Web caching mechanisms. Fig. 1 illustrates the hypothetical overall performance of three caching mechanisms across the evaluation space. The dotted lines show the expected trajectories for these mechanisms when cache size increases. Notice that there is a tradeoff between maximizing quality of cached streams and maximizing the overall cache performance (e.g., byte hit ratio), because changing the quality of a cached stream directly affects its size. If the cache always maintains low-quality versions of streams, as in Low Quality Caching, the byte hit ratio can be high because larger number of streams reside in the cache. However maximum deliverable stream quality is consistently low due to the lack of any quality improvement mechanism. On the other hand, High Quality Caching always stores the highest quality for every requested stream, even if the client does not have sufficient bandwidth to receive such high quality. It maximizes the cached quality at the cost of lower utilization of cache storage space, and consequently a lower byte hit ratio. MCaching tries to match the quality of cached streams with the deliverable quality, i.e., the quality that most clients can afford. Thereby MCaching uses cache storage space more efficiently and achieves both high byte hit ratio and high cached stream quality.

This paper presents a comprehensive framework for performance evaluation of multimedia proxy caching mechanisms. Because the notion of stream quality does not exist in traditional Web caching, the existing cache performance metrics (e.g., byte hit ratio) are not sufficient to fully evaluate multimedia caching mechanisms. We show that quality improvement and cache efficiency are orthogonal in terms of their goals, hence the performance of multimedia caching should be examined collectively along both dimensions. Our framework proposes two classes of metrics for both two dimensions of the evaluation space. The

---

1 Certain, 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 the cache is still able to reduce the load on the network and the server.

We would also like to thank Deborah Estrin and Jennifer Rexford for their insightful comments during the development of this paper. Haobo Yu is supported by the Defense Advanced Research Projects Agency (DARPA) through the VINT project at USC/ISI under DARPA grant DABT63-96-C-0054.
caching efficiency metrics capture the cache performance in reducing network load and server load, whereas the quality metrics measure the cache performance in terms of improvement in stream quality. Furthermore, each class contains metrics at the aggregate and per-stream levels for evaluations at different granularities.

We have conducted simulation-based evaluations using our framework to compare MCaching with two other strawman multimedia caching mechanisms. Our results verified the fundamental tradeoff between quality improvement and caching efficiency. At the aggregate level, we found that compared to the two strawman approaches, MCaching adaptively maximizes both caching efficiency and quality improvement across the evaluation space by means of partial prefetching and fine-grain replacement. At the per-stream level, we observed interesting interactions among quality evolution, prefetching and caching efficiency, and congestion control. Our results reveal that prefetching and caching efficiency are directly determined by the difference between cached quality and deliverable quality. Our results also verified that MCaching converges the cached stream quality to the deliverable quality, which is the exact reason why MCaching results in higher aggregate cache efficiency.

The rest of this paper is organized as follows: Section II provides an overview of the design of MCaching. Section III presents our evaluation methodology for multimedia caching mechanisms, including two sets of metrics and two strawman approaches. We then explain our simulation strategy in Section IV. In Sections V and VI we describe our simulation results for aggregate and per-stream level evaluations. We review related work in Section VII. Section VIII concludes the paper and addresses our future directions.

II. MULTIMEDIA PROXY CACHING (MCACHING): AN OVERVIEW

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].

Performing congestion control results in unpredictable 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 [3] over the Internet—that is, they should match the quality of the delivered stream with the average available bandwidth on the path. Once a stream is cached, the cache can replay it for subsequent requests but it still needs to perform congestion control and quality adaptation based on the state of its connection to the client. The proxy-client connection is likely to exhibit different characteristics, e.g., different changes in available bandwidth, from previous connections. Thus, some of the required segments by quality adaptation might be missing from the cache, and should be prefetched.

MCaching assumes the existence of an end-to-end architecture for playback of quality adaptive multimedia streams in a congestion controlled fashion over the Internet (such as that in [4]). TCP-friendly congestion control is performed using Rate Adaptation Protocol (RAP) [5]. The quality adaptation module adjusts the quality of the played back stream. 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. MCaching further assumes that all the streams are linear-layered encoded where all layers have the same constant bandwidth for the sake of clarity. However, this architecture (and MCaching) can be extended to other layered-encoding bandwidth distributions. 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.

MCaching assumes that all streams among servers and proxies, and among proxies and clients, must perform congestion control and quality adaptation. It does not make any assumption about the inter-cache architecture which determines control message exchange and request forwarding.

A. Delivery Procedure

- Relaying on a Cache Miss: On a cache miss, the request is forwarded to the original server or a neighbor cache depending on the inter-cache architecture. For simplicity we assume that the stream is played back from the original server to the cache via a congestion controlled connection. The cache then relays data packets to the client through a separate congestion controlled connection. The quality of the delivered stream is limited by the average bandwidth between the server and the cache. Thus the client does not observe any benefit (e.g., quality improvement) from the presence of the cache.

- 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 quality adaptation during the new session. This means that the cache may require to send some segments that it does not have. To improve the delivered quality, the cache should prefetch these missing segments from the server before their playout times, i.e., the deadline when they must be delivered for the client to play back. Fig. 2 illustrates the difference between played back and cached quality that is filled with prefetched segments. Because of unpredictable changes in quality, some of the prefetched segments may not be used.

As outlined above, MCaching has two major components: 1) prefetching and 2) replacement. We will discuss these two mechanisms in more details for the rest of this section.

B. Prefetching

During the playback of a cached stream, the cache needs to maintain two unsynchronized connections: (i) that between the server and the proxy for prefetching, and (ii) that between the proxy and the client for delivery of the stream. The proxy must predict a missing segment that may be required by quality adaptation in the future and prefetch it before its playout time. 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.

To better meet the playout deadlines, prefetching should loosely follow the playback session; otherwise the prefetching stream may fall behind and become useless. Towards that goal, a sliding-window mechanism was devised for prefetching [1]. The proxy examines a window of time in the near future and sends an ordered list of all the missing segments in that window based on their priority (i.e., layer number). These segments may be missing due to packet losses, layer drops in previous playbacks, or replacement. Furthermore, if quality adaptation decides to add a new layer, all missing segments of the new layer within the prefetching window are also requested (e.g., $L_0$ in Fig. 2).

To ensure in-time delivery of required segments, the prefetching window should slide as fast as the playout point. Thus the proxy periodically slides the prefetching window and sends a new prefetching request to the server. The server delivers requested segments via a congestion controlled connection to the proxy 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. To loosely synchronize the prefetching stream with the playback stream, a new prefetching request preempts the previous one. If the server receives a new prefetching request before finishing delivery of segments in the previous request, it ignores the old request and starts to deliver segments in the new request. 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).

C. Replacement Algorithm

Due to the lack of sub-structure in ordinary Web documents, most of the existing replacement algorithms for Web caching are atomic. They make binary replacement decisions, i.e., pages are cached or flushed out in their entirety. Layered encoded streams are naturally structured into separate layers, and each layer is further divided into equal-size segments. MCaching exploits this structure to perform fine-grain replacement, which allows fine-grain adjustment of the quality of cached streams, reduces fragmentation of the cache space and improves the cache efficiency. (See Appendix B for further discussion about replacement granularity.)

The proxy maintains popularity of individual layers of each stream.

To maximize the performance of the cache, the proxy always flushes segments of the least popular layer, called the victim layer. The victim layer is always the top layer of a cached stream. It is generally preferred to cache a contiguous portion from the beginning of a layer to have minimum variations in quality and reduce startup latency [7]. Thus segments of the victim layer are flushed from the end toward the beginning. Fig. 3 depicts the replacement pattern within a single cached stream. 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. In order to hide startup latency, the first few segments of the base layer for each cached stream may be kept in cache for a long period even though its popularity becomes low. While one can devise other replacement patterns to optimize other aspects of MCaching performance, no other pattern seems to maximize the quality and minimize the load on the server simultaneously.

III. EVALUATION METHODOLOGY

The quality adaptive nature of multimedia streams brings two dimensions to the performance evaluation of multimedia caching:

1. Quality improvement describes how the quality of individual cached streams is improved, and,
2. Caching efficiency describes how effectively the cache reduces network load and server load.

In addition, the performance of a multimedia cache can be evaluated at two levels:

- Aggregate level: where we study the overall performance of the cache in response to a sequence of requests.
- Per-stream level: where we investigate the behavior of the cache with respect to individual streams.

Caching efficiency exists in traditional Web caching, and it has been measured at the aggregate level. Quality improvement is specific to multimedia caching, and it can be measured at both per-stream and aggregate levels. In the following subsections, we present our evaluation metrics for these two levels as well as two strawman mechanisms to compare with MCaching.

In our evaluations, we only refer to the quality of cached streams instead of the delivered quality to clients. The maximum deliverable quality only depends on available client bandwidth, and is a constant target during a single experiment (see Section IV). We then examine the evolution of the cached quality to see how effectively it matches with the target deliverable quality.

A. Evaluation Metrics

We present our aggregate and per-stream level metrics separately. Note that at each level, we need metrics to measure both caching effi-

---

$^{23}$ See [3] for details on conditions for adding a new layer.

$^{24}$ Note that this segment-based replacement may result in thrashing. To avoid this, while a particular stream is played back from the cache, its active layers are locked in the cache and cannot be replaced during the playback.
efficiency and quality improvement. Table I classifies all our metrics based on their granularity and corresponding dimension.

### A.1 Per-Stream Metrics

Several metrics are required to capture various aspects of the caching mechanism at the per-stream level. We first describe quality-related metrics, then the metrics for caching efficiency.

#### A.1.a Stream Quality

There is no well-accepted metric for measuring perceptual quality of a multimedia stream. Instead, we present three metrics that collectively capture the pattern of changes in number of layers (i.e., layer add and drop) for a layered encoded stream in the cache. Notice that the preceptual effect of layer add and drop is an encoding-specific issue. These metrics enable us to quantitatively track the evolution of a layered encoded stream in the cache. Thus for any given encoding, one could analyze the effect of this evolution on perceptual quality.

- **Completeness** is defined on a per-layer basis and measures the percentage of the layer resident in the cache. This metric allows us to trace the quality evolution for a cached layer. The completeness of layer $l$ of a cached stream $s$ after an arbitrary request $r$ is defined as the ratio of the layer size in cache to its “official length”\(^6\):  
  \[
  CP(r, s, l) = \frac{1}{C_l} \sum_{C(r, s, l)} L_{r,i}
  \]
  where we define a *chunk* as a continuous group of segments in a single layer of a cached stream, and $C(r, s, l)$ is then defined as the set of all chunks of layer $l$ of stream $s$ after request $r$\(^7\). $L_{r,i}$ is the length (in terms of segments) of the $i$th cached chunk in layer $l$. Obviously the value of completeness always falls within $[0,1]$. Notice that $r$ is not restricted to requests for $s$. Thus if $s$ is not in the cache at the time of $r$, this implies $CP(r, s, l) = 0$.

- **Continuity** is defined on a per-layer basis and reflects the average chunk size for a layer. Completeness alone does not reflect the number of “holes” in a cached layer. For a given value of completeness, the higher the value of continuity, the lower the number of holes (or chunks). The continuity of layer $l$ of a cached stream $s$ after an arbitrary request $r$ is defined as the average size of all chunks in the layer normalized by the layer size: \(^6\)
  \[
  CO(r, s, l) = \frac{1}{C_l} \text{mean}(C(r, s, l))
  \]
  Note that continuity may exhibit rapid changes due to the random nature of packet loss and layer add and drop.

- **Variation** is defined on a per-layer basis and shows the standard deviation among chunk sizes within a layer. Continuity alone does not provide any information about distribution of chunk sizes within a layer. For a given completeness and continuity, the lower the variation, the more uniform are the chunk sizes within that layer. The variation of layer $l$ of a cached stream after the $r$-th request is defined as:
  \[
  SM(r, s, l) = \text{stddev}(C(r, s, l))
  \]

#### A.1.b Caching Efficiency

We use two metrics to represent the caching efficiency of MCaching to improve the quality of a single cached stream.

- **Layer Hit Ratio** measures the efficiency of caching mechanism in maintaining segments of each layer. It is in fact a per-layer byte hit ratio. We define the layer hit ratio of layer $l$ in cached stream $s$ during request $r$ as:
  \[
  LR(r, s, l) = \frac{B_s(r, s, l)}{B(r, s, l)}
  \]
  where $B_s(r, s, l)$ is the number of bytes delivered from the cache, and $B(r, s, l)$ is the total amount of bytes delivered. When the cache has all the segments of layer $l$ that the client wants during a request, layer hit ratio of layer $l$ is 100%.

- **Prefetching Efficiency** measures the effectiveness of the prefetching mechanism in delivery of higher layers that are missing from the cache. A prefetched segment might not be played back due to incorrect prediction or late arrival. Either of the two cases reduces the efficiency of the prefetching mechanism. We define the prefetching efficiency of layer $l$ in cached stream $s$ after request $r$ as the portion of prefetched segments of $l$ that have been played back during the session of $r$:
  \[
  PE(r, s, l) = \frac{P_s(r, s, l)}{P(r, s, l)}
  \]
  where $P(r, s, l)$ is the number of total prefetched bytes, among which $P_s(r, s, l)$ bytes arrived in time and were delivered to the requesting client.

For each of the above two metrics, we define its per-stream version as its average across all layers in the stream and all cache hits of the stream in an experiment. Per-stream layer hit ratio captures the overall effectiveness of the caching mechanism to maintain the cached quality of a stream at the deliverable quality during an experiment. Per-stream prefetching efficiency measures the overall effectiveness of prefetching.

### A.2 Aggregate Metrics

The following three aggregate metrics are intended to measure the overall behavior of a caching mechanism with respect to quality improvement and caching efficiency.

- **Average Cached Quality** measures the quality of cached streams averaged over time and across all streams. Let the set of all multimedia streams be $S$ and the set of all requests during the simulation be $R$. Define the average quality of the entire cache to be the average of per-layer completeness across every stream, every layer and every request during an experiment. We only use completeness to measure average quality because averaging other metrics for quality does not provide much meaningful information. This metric represents the average quality of all cached streams during every request in an experiment.

- **Byte Hit Ratio** is used to measure the overall efficiency of a proxy cache in reducing network traffic. It is defined as the percentage of bytes delivered from the cache among the total bytes delivered to the client during an experiment. The difference between per-stream layer hit ratio and byte hit ratio is that per-stream layer hit ratio is only measured during cache hits but byte hit ratio takes into account both cache

<table>
<thead>
<tr>
<th>Per-Stream Level</th>
<th>Aggregate Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>Completeness, continuity, variation</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Layer hit ratio, prefetching efficiency</td>
</tr>
</tbody>
</table>

Table I: Evaluation metrics.
hits and misses. Hence per-stream layer hit ratio does not reflect the overall efficiency of the entire cache.

- **Network Load** is the total amount of traffic from the server to the cache during an entire experiment. It represents how effectively the cache responds to client requests without introducing additional load to the network and the server.

### B. Strawman Mechanisms

We are not aware of any multimedia proxy caching mechanism that addresses quality improvement or incorporates congestion control and quality adaptation. Given that MCaching is tightly coupled with congestion control and quality adaptation, comparison with other schemes without these two components would be unfair and meaningless. Therefore, for the sake of comparison we present two other variations of MCaching as follows:

- **No Prefetching** (NOP): This scheme is similar to MCaching, but without prefetching and with atomic replacement. Hence the quality of a cached stream is limited by the bottleneck bandwidth during the playback from the server and is never improved. Streams are cached and flushed in their entirety instead of on a per-layer basis.

- **Off-line Prefetching** (OFP): This scheme is similar to MCaching, but with atomic replacement and atomic improvement. After the first request, all the missing segments of the stream are fetched from the server via a TCP connection. Thus the proxy has the highest quality version of the stream that resides at the server. Streams are cached and flushed in their entirety instead of on a per-layer basis.

Each of these strawman mechanisms attempts to maximize the performance along only one dimension of the design space. NOP attempts to achieve higher byte hit ratio by caching a larger number of streams at the cost of lower quality. In contrast, OFP provides maximum quality regardless of available bandwidth to the clients, which determines the deliverable quality.

### IV. SIMULATION SETUP

We evaluate the multimedia caching mechanisms with simulation using ns-2 [9]. We use RAP [5] (for congestion control) along with layered quality adaptation [3] as transport protocol for multimedia streams. Our simulations do not include any error control mechanism, i.e., there is no mechanism to repair packet losses (e.g., retransmission or FEC). However, priority-based prefetching in MCaching is able to fill the holes caused by packet losses. Adding an error control mechanism will certainly speed up the quality improvement process. In the absence of error control, our results represent the worst case scenarios.

There are two important factors in designing our simulation scenarios: 1) request sequence, and 2) the bandwidth distribution among clients and the location of the bottleneck link. We address these issues next.

#### A. Request Sequence

Without any knowledge about access patterns of Internet multimedia streams, we generate request sequences using a customized version of the SURGE Web workload generator [10]. Three factors are needed to generate a request sequence: the number of requests for each stream (i.e., stream popularity), request ordering and request interval distribution. We first assume that stream popularity conforms to the Zipf’s law, which has been observed in various Web traces [11]. 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 \( \Omega = \sum_{i=1}^{N} \frac{1}{i} \). We then generate request ordering so that the stack distance of the requests exhibits log-normal distribution with empirical parameters [10]. Finally, although there are empirical request interval distributions, it is difficult to apply them to multimedia stream caching because Web pages are usually much smaller than multimedia streams. Thus using the same distribution is likely to result in a proliferation of traffic. In the absence of empirical data about access patterns of Internet multimedia streams, we chose a uniform distribution from 300 seconds to 400 seconds as our request interval model. One consequence is that most requests are sequential as seen by the cache.

#### B. Network Scenario

Throughout our simulation, we use a simple network topology (Fig. 4). \( BW_{sp} \) denotes the link bandwidth between the server and the proxy, whereas \( BW_{pe1} \) and \( BW_{pe2} \) are link bandwidths between proxy and two clients, respectively. When only one client is active, say client 1, one may construct two interesting scenarios from this simple topology:

- **Scenario I**: \( BW_{sp} < BW_{pe1} \), the server-proxy connection is the bottleneck.
- **Scenario II**: \( BW_{sp} \geq BW_{pe1} \), the proxy-client connection is the bottleneck.

In scenario II, because the bottleneck is at the client side, the cached stream quality is usually higher than the deliverable quality and this leaves no need for further quality improvement. To observe the overall effect of quality improvement and caching efficiency, we mainly explore scenario I in our simulations.

We focus on the effect of client bandwidth heterogeneity which is the key factor that determines the quality of cached streams. In our simulations, we set \( BW_{sp} \), \( BW_{pe1} \) and \( BW_{pe2} \) to 56Kbps (~1.2 layers), 1.5Mbps and 128Kbps (~2.7 layers) respectively. By changing the distribution of requests between the high bandwidth client and the low bandwidth client, we are able to continuously traverse from one extreme—where all clients are low-bandwidth—to the other—where all clients have high bandwidth connectivity. We tune the request distribution by the ratio of requests from the low bandwidth client. For example, a request ratio of 5% means that only 5% of the requests come from the low bandwidth client and the rest are issued by the high bandwidth client. This ratio essentially provides a tuning knob for the average available client bandwidth seen by the cache. Therefore in the following section, we will use “client bandwidth ratio” and “average client bandwidth” interchangeably.

We do not impose background traffic on the server-proxy link in any of our simulations. We have conducted simulations using a self-similar Web traffic model [12], which proved to be too expensive in terms of execution time for our simulations with many requests to large multimedia streams. In addition, results of our medium-sized simulations exhibited the same qualitative trends with and without background traffic (see Appendix C for details).

There are other parameters that affect our simulations, such as segment size, layer consumption rate (the rate at which receiver consumes data in each layer), number of layers, number of streams and stream length. To focus on main variables, we limit the number of parameters and assume all streams have 6 layers, the segment size is 1KB, and

---

1The use of RAP may overlap with the RAP connection from the cache to the server during a cache miss.

2Many Web traces do not exactly follow the Zipf’s law, instead they exhibit Zipf-like behavior [11]. For simplicity we use Zipf’s law in this paper.

3This does not invalidate our final conclusions. The only effect of serving concurrent requests from the cache is that multiple prefetching sessions compete for the server-proxy bandwidth. This results in slightly slower quality improvement. See Appendix C for details.
Hence our results can be scaled with respect to stream length.

V. AGGREGATE PERFORMANCE EVALUATIONS

In this section we present aggregate performance evaluations for MCaching and the two strawman mechanisms (NOP and OFFP). Our data set consists of 200 streams. The stream popularities follow the Zipf’s law. The most popular stream has 400 requests. Without statistical knowledge about the size distribution of real Internet multimedia streams, we set the stream lengths to be uniformly distributed between 30 seconds and 3 minutes (the size in bytes can be obtained by multiplying the stream length, the number of layers and the layer consumption rate)\(^1\).

Two major parameters that affect the aggregate behavior of a caching mechanism are: cache size and client bandwidth. Cache size changes the caching efficiency whereas client bandwidth controls the quality of cached streams. Thus, in our simulations we vary the cache size to be 0.025, 0.05, 0.1, 0.2 and 0.4 times of the data set size (i.e., the size of all streams), and vary the ratio of requests from the low bandwidth client between 0% and 100%.

A. Byte Hit Ratio

Fig. 5(a) shows byte hit ratio as a function of cache size when client bandwidth ratio is 0.05. The following discussion applies to all other client bandwidth ratios that we have tested. We observe that MCaching always exhibits higher byte hit ratio than both NOP and OFFP. The smaller the cache size, the larger is the advantage of MCaching. The culprit is the atomic replacement in OFFP and NOP. Fine-grain replacement is more efficient in utilizing cache space than atomic replacement, as shown in Appendix B. When cache storage is abundant (e.g., 0.4 times the data set size), there is less contention for cache space and cache utilization is a less significant factor. Hence OFFP is able to achieve relatively high byte hit ratio. For smaller cache sizes, however, contention for cache space becomes strong and cache utilization becomes dominant. Thus OFFP experiences more cache misses and its byte hit ratio is low. Notice that the smallest cache size in this simulation (2.5% of total data set) is not an unreasonable number compared with the sizes of most current Web caches\(^2\).

Fig. 5(b) shows byte hit ratio as a function of client bandwidth when cache size is 0.025. Again, the following discussion applies to other cache sizes as well. When most requests come from the high bandwidth client, OFFP’s byte hit ratio increases by 44%. Because OFFP prefetches all missing layers of a stream after a cache miss, when client bandwidth is low, higher prefetched layers unnecessarily occupy cache space. This results in low utilization of cache space, which leads to a low byte hit ratio. In contrast, MCaching adaptively adjusts its behavior based on client bandwidth and only prefetches the required segments. This partial prefetching results in more efficient utilization of cache space and is able to achieve high byte hit ratio regardless of the client bandwidth. Client bandwidth has no effect on the byte hit ratio of NOP, because in our simulations the server-proxy connection is the bottleneck, NOP always results in a cached quality that is lower than the deliverable quality and does not waste cache space.

B. Average Cached Quality

Fig. 6(a) depicts the average cached quality as a function of cache size and client bandwidth ratio. We use a 3D graph because this function is not homogeneous along either dimensions. When cache size is large, OFFP results in higher average cached quality than MCaching and NOP. But for smaller cache sizes (e.g., 0.025 times the data set size), MCaching has higher average cached quality than OFFP. This is again the result of more efficient cache utilization of MCaching due to its fine-grain replacement. The average cached quality of NOP is always lower than the other two, which shows the quality improvement effect of prefetching in MCaching and OFFP.

\(^1\) The number of publicly indexable Web pages in the world is estimated to be about 800 million, containing about 15TB [13]. This is orders of magnitudes larger than the sizes of most current Web caches. When large multimedia streams become common, it is reasonable to expect this number to grow even further.

\(^2\) Longer streams can be viewed as combinations of several shorter streams with the same popularity. Hence our results can be scaled with respect to stream length.
Client bandwidth has no effect on average cached qualities of OFFP and NOP because they do not adjust quality based on the bandwidth of the interested clients. MCaching adjusts the average cached quality according to the available client bandwidth\(^7\).

### C. Network Load

Fig. 6(b) shows the network load between the stream server and the proxy cache as a function of cache size in log scale. The following conclusion remains the same for different values of client bandwidth ratios that we have tested. In general, the network overhead of OFFP is an order of magnitude higher than MCaching. This is the joint effect of blind prefetching of the entire requested streams and the atomic replacement. The network load of MCaching is about 2-5 times higher than NOP, which is a reasonable cost for its 2-4 times improvement in the average cached quality over NOP.

### D. Summary

Fig. 7(a) summarizes our previous simulation results with respect to the overall performance in the evaluation space, which uses average cached quality and byte hit ratio as two dimensions (as same as Fig. 1). Notice that both cache size and client bandwidth change across these results. Fig. 7(a) verifies our hypothesis that there is a fundamental tradeoff between cached stream quality and byte hit ratio. The higher the average quality of cached streams, the smaller the number of cached streams and the lower the byte hit ratio.

Each of the three mechanisms exploits this tradeoff differently. OFFP is able to achieve both high average cached quality and high byte hit ratio when cache storage is abundant, i.e., 0.4 times the data set size. But for smaller cache sizes, OFFP results in more cache misses and exhibits both low byte hit ratio and low average cached quality. In contrast, NOP always results in high byte hit ratio (>65%) at the cost of significantly lower average cached quality (<1). MCaching exploits the tradeoff more effectively to improve the performance along both dimensions. It achieves adequate average cached quality while keeping byte hit ratio always above 70%. Although both MCaching and OFFP achieve high quality, MCaching can effectively reduce network load because of its efficient utilization of cache space. This observation is verified by Fig. 7(b), which maps the same set of simulation results over a different angle using quality and network load as two dimensions. This figure clearly demonstrates that MCaching achieves similar quality as OFFP at significantly lower network load. NOP has lower network load than MCaching, but its quality is also reduced by a similar factor.

Our aggregate performance evaluation presents an evidence that performance evaluations of multimedia proxy caching mechanisms should be conducted with respect to various dimensions of the evaluation space. Furthermore, any performance assessment should be drawn from results along various dimensions collectively.

In the next section we focus on the evolution of quality of cached streams as well as caching efficiency at the per-stream level. Given the static nature of OFFP and NOP in quality adjustment, we will only examine MCaching in order to demonstrate the dynamics of quality adjustment due to fine-grain replacement and partial prefetching.

### VI. Per-Stream Performance Evaluations

In this section, we will first illustrate the “micro-level” quality evolution on a per-layer basis. Then based on these observations, we will discuss the impact of stream popularity and client bandwidth on the resulting quality of cached streams.

In order to closely track the quality evolution of every cached stream on a per-layer basis, our data set consists of only 10 streams for our per-stream performance evaluation. Stream popularities follow the Zipf’s law. The most popular stream has 100 requests. Similar to the aggregate evaluations, stream lengths are uniformly distributed between 30 seconds and 3 minutes. Because we have already discussed the impact of cache size on the aggregate behavior, we set the cache size to half of the size of our data set to observe moderate replacement.

#### A. Per-Layer Quality Evolution

To relate quality evolution to caching and prefetching efficiencies, we focus on the quality adjustment of a single cached stream. Fig. 8 depicts the evolution of per-layer quality (completeness, continuity and variation) for the most popular stream during a sequence of requests where 95% of the requests are from the low bandwidth client. Arrival time of requests for the most popular stream from the high and low bandwidth clients are shown at the top of each graph as long and short ticks, respectively. Fig. 8 also shows per-layer layer hit ratio and prefetching efficiency to demonstrate their correlation with quality evolutions.

After the first request, the cached stream quality is low (70% of the base layer, 30% of the second layer and 10% of the third layer) due to the bottleneck link between the server and the cache. Since both clients can afford higher quality streams, as more requests arrive, prefetching gradually brings in higher layers. The quality improvement occurs on a layer-by-layer basis, i.e., the quality of lower layers are improved before any higher layers. This is most clearly shown in the variation plot. Once a layer is initially fetched on a cache miss, the variation first increases because of random “holes” caused by layer drop or packet loss. Once a layer is mostly resident in the cache, its completeness reaches close to 100%, then prefetching starts to fill the holes. This monotonically reduces the variation until it reaches zero. Thus, the peak of variation for any layer occurs always before that of all the higher layers.

Although the low-bandwidth client can only afford about 2.7 layers, the higher layers are occasionally prefetched into the cache upon the arrival of a request from the high bandwidth client. Requests from the high bandwidth client arrive roughly at times 8000s, 41000s, 55000s, 80500s, 83000s, 90000s and are most clearly visible as spikes in the per-layer prefetching efficiency plot.
The lower layers are only prefetched during a few initial requests, then they reach maximum quality and stay in the cache because of frequent access to these layers. Consequently, their layer hit ratios quickly reach 100% and remain at the maximum level. In contrast, higher layers are only needed to serve a request from the high bandwidth client which is responsible for only 5% of all requests. Therefore, their popularities (hence status of residency in the cache) are small and heavily dependent on the temporal distribution of requests. For example, most of the requests from the high bandwidth client are far apart, thus the top layer usually does not stay in the cache. The only exception is arrival of three requests from the high bandwidth client within the interval [80000s, 90000s] that prefetched most of the top layer in the cache and keep it in the cache for a short period.

The prefetching efficiency plot also shows that prefetching efficiency decreases as the layer becomes more complete, i.e., prefetching is most efficient in “adding new layers” but not in “filling holes”. For example, as layer 4 is filled up, its prefetching efficiency drops from 60% (at time 8000s) to 14% (at time 55000s). This is because of the rate adaptation enforced by the congestion control mechanism, RAP. Thus, it is easier to transfer a continuous stream of data than a burst of prefetched segments in a short period of time. This behavior can be improved by fine-tuning of the prefetching mechanism to spread a big burst of required segments over time within a certain time window. We could also use a bigger prefetching window to smooth out the variations in prefetching bandwidth.

The layer hit ratio plot depicts how well the cache satisfies requests for individual layers without going to the server. While it quickly reaches 100% for the first three layers, it is very bursty for higher layers. Note that layer hit ratio does not capture the amount of data sent during a request. If a layer has only a few segments in the cache and these segments are accidentally the segments that are needed by quality adaptation, it may not reach 100% and remain at the maximum level. In contrast, higher layers require few segments and the layer hit ratio quickly reaches 100% and remains at the maximum level.

Second, the impact of client bandwidth on quality is as important as the effect of random packet loss and layer drop are more visible in their unpopular streams in the continuity plot. Because unpopular streams have few requests and lower chances for adding layers and filling holes, their popularities (hence status of residency in the cache) are small and heavily dependent on the temporal distribution of requests.
stream popularity. For example, when most requests come from the high bandwidth client who can receive all layers, the completeness of the most popular stream (i.e., stream 0) is 5.8, which is almost 6 times higher than that of the least popular stream, 0.98. However, when most requests come from the low bandwidth client who can afford only 2.7 layers, the completeness (3.08) of the most popular stream is almost the same as that (2.52) of the least popular stream.

We can explain this phenomenon as follows. When client bandwidth is high, most layers of the popular streams are likely to remain in the cache. This leaves less space for unpopular streams, which consequently results in more flushing and lower quality. When client bandwidth is low, none of the streams is likely to keep higher layers in the cache. This provides more space for the lower layers of the less popular streams and results in higher per-stream quality for those streams. In fact, in these simulations the cache size is half of the data set size and each stream has 6 layers. When client bandwidth is high, the 6 layers of the 5 most popular streams occupy most of the cache space and leave no room for other less popular streams. When client bandwidth is low (2.7 layers in our simulations), all streams are able to keep about half of their layers in the cache, which are all the required layers.

C. Per-Stream Caching Efficiency

We have shown earlier that per-stream caching efficiency is closely correlated with per-stream quality. In this section, we study the effect of the quality of cached streams on per-stream caching efficiency using per-stream layer hit ratio and prefetching efficiency as evaluation metrics. More specifically, we examine the effect of stream popularity and client bandwidth, which have significant impact on the stream quality. Fig. 10 shows the impact of these two parameters on per-stream layer hit ratio and prefetching efficiency.

There are two cases when layer hit ratio reaches 100%: 1) for the most popular streams regardless of client bandwidth, and 2) for all streams when most requests come from the low bandwidth client. In both scenarios, most of the required layers are able to remain in the cache, which results in high layer hit ratio. As the client bandwidth increases, or the stream popularity decreases, layer hit ratio decreases because there is larger difference between cached quality and maximum deliverable quality, and more bytes need to be prefetched from the server. The lowest layer hit ratio is 60%, which is obtained for the most unpopular stream with the highest client bandwidth.

There exists an almost complementary relation between layer hit ratio and prefetching efficiency. When layer hit ratio is high, most bytes are delivered from the cache, and prefetching is rarely needed. Prefetching efficiency is low because prefetching only occurs in a bursty fashion to fill some remaining random holes. Prefetching efficiency increases when layer hit ratio is lower, and becomes the highest for streams with mid-range popularities. For example, when client bandwidth is high, the highest, the prefetching efficiency of the 4th popular stream can be as high as 95%. These streams are not popular enough to keep all layers always in the cache, but are still popular enough so that their lower layers remain in the cache. Thus they only need to prefetch a few higher layers, which can be prefetched in time without overloading the server-cache bottleneck.

Unpopular streams are exceptions to this complementary behavior between prefetching efficiency and layer hit ratio. They always have the lowest prefetching efficiency and the lowest layer hit ratio. There are different reasons for this behavior depending on client bandwidth. When the client bandwidth is high, quality of unpopular streams is likely to be low with frequent variations. If prefetching occurs, it is more likely that prefetching has to fill the holes in lower layers first before it adds higher layers. Thus prefetching should smooth out these random variations and bring in the missing required segments. This results in a relatively rapid variations in prefetching bandwidth requirement. As we discussed in Section VI-A, in such cases prefetching does not perform efficiently due to the bandwidth regulation by the congestion control protocol. When client bandwidth is low, unpopular streams are able to remain in the cache and prefetching is only needed to fill random holes. Prefetching achieves low efficiency in such cases as we discussed above.

---

14 Notice prefetching efficiency of the most popular stream is sometimes 100%. This is an artifact because we count prefetching efficiency as 100% when there is no prefetching at all.
D. Summary

Micro-level examination of per-layer quality evolutions reveals that MCaching improves quality on a layer-by-layer basis starting from the lowest layer. Quality evolutions are closely correlated with prefetching efficiency. In particular, prefetching of continuous pieces of data can be performed more efficiently than sporadically prefetching in a congestion controlled fashion.

By exploring the effect of popularity and client bandwidth on the cached stream quality, we have found:

- Lower layers that are more frequently requested remain in the cache, whereas higher layers that are less frequently accessed are prefetched and flushed in a demand-driven fashion.
- As the popularity of the stream increases, its average cached quality becomes closer to the maximum deliverable quality which is determined by available client bandwidth.

Evaluations of per-stream caching efficiency reveal that the difference between cached stream quality and the target deliverable quality determines both per-stream caching efficiency and prefetching efficiency.

VII. Related Work

There have been many comprehensive studies of Web cache performance. Among them, work in [11] presented a thorough study of the implication of the Zipf’s law on cache performance, and derived asymptotic hit ratio based on this observation. Work in [10] presented a comprehensive Web workload model. We adopted their model in our request sequence generator.

Using prefetching to improve Web cache performance has been discussed before [14], [15]. They evaluated strategies to predict future requests. Sometimes prefetching may increase source burstiness which has negative impact on the network. Work in [16] proposed techniques to reduce this effect by rate-limit the prefetching requests. These prefetching mechanisms operate at the per-request level while our prefetching works inside the substructure of a single stream, hence these schemes complement each other.

There is also significant previous work on Web proxy cache replacement algorithms [17], [18]. The behavior of these replacement algorithms for traditional Web caches are well understood. However, their behaviors for access patterns with a significant number of requests to large multimedia streams, especially those with variable quality, have not been studied. This is partially due to the absence of any proposal for multimedia Web caching mechanisms.

Work in [19] addresses the implications of resource requirement (i.e., bandwidth and space) for multimedia streams on cache replacement algorithms. Another scheme [7] stores prefixes of multimedia streams in proxy caches to improve startup latency and provides an opportunity for smoothing. Our work complements these efforts in that we provide a comprehensive study of multimedia caching mechanisms focusing on quality improvement and cache efficiency.

VIII. Conclusion

This paper presented an initial attempt to bridge the gap between the evaluations of traditional Web caching and multimedia proxy caching. Multimedia proxy caching introduces cached stream quality as a new dimension of cache performance evaluation space, which is not captured by existing cache performance metrics, e.g., byte hit ratio. Our main contribution is a comprehensive framework for performance evaluation of multimedia caching mechanisms. In this framework, we proposed evaluation metrics for both stream quality and caching efficiency at both aggregate level and per-stream level. Using simulations, we identified the fundamental tradeoff between stream quality improvement and caching efficiency. We showed that compared with alternative approaches, MCaching effectively exploits this tradeoff by adaptively changing the cached stream quality to match the deliverable quality, therefore maximizing both overall stream quality and caching efficiency. Our simulations also revealed interesting dynamic interactions among evolution of cached stream quality, prefetching efficiency and congestion control. Because of these interactions, per-stream caching efficiency and prefetching efficiency are directly determined by the difference between cached stream quality and deliverable quality.

Multimedia proxy caching is still a new area of research. We plan to extend our work in several directions. First, we will leverage the previous studies on Web cache prefetching and replacement algorithms and examine the possibility and impact of incorporating the compatible mechanisms in multimedia caching. Second, given the large number of parameters and the dependencies among them, simulation-based study seems to be inadequate for deep understanding of the dynamics of replacements and its interactions with prefetching. Therefore, we plan to devise an appropriate analytical model that captures key aspects of the problem and helps us to better understand the effect of various parameters on cached stream quality and caching efficiency. Finally, we plan to gather real Internet multimedia stream access traces and study their differences from Web page access patterns. These trace data will also facilitate real-world evaluation of the MCaching mechanism when its implementation becomes publicly available.

REFERENCES


APPENDIX

I. Popularity Function

To do replacement, the proxy should keep track of the popularity for each cached stream, i.e., the level of client interests in the stream. We assume that the total playback time of each stream indicates the level of client interest. For example, if a client only watches half of one stream, its popularity interest is half of a client who watches the entire stream. Based on this observation we extend the semantics of a hit and define
the term weighted hit (whit) as follows\textsuperscript{15}:

\[ \text{whit} = \frac{\text{PlaybackTime}}{\text{StreamLength}} \quad 0 \leq \text{whit} \leq 1 \tag{6} \]

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). While the level of client interests does not affect per-layer popularity, adding and dropping layers by quality adaptation results in different PlaybackTimes for different layers in a session and consequently results in a different popularity for the cached layers. 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\% and 50\% of the playback time, respectively.

To capture both level of client interest and usefulness of individual layers in the cache determined by layer add and drop, the proxy calculates whit 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:

\[ P = \sum_{x=t-\Delta}^{t} \text{whit}(x) \tag{7} \]

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 compatible with the fine-grain replacement algorithm, because layered encoding guarantees that popularity of different layers in the same stream monotonically decrease with the layer number\textsuperscript{16}. 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.

II. REPLACEMENT GRANULARITY

In this section, we first show that compared with atomic replacement, fine-grain replacement improves cache efficiency, then discuss the tradeoff between replacement granularity and book-keeping overhead.

Denote the \( k \)-th segment of the \( j \)-th layer of the \( i \)-th stream to be \( S(i, j, k) \). Let the cache size be finite and able to hold \( C \) segments. Let \( C_e \) denote the set of segments remaining in the cache when atomic replacement is used. We define its asymptotic byte hit ratio as:

\[ B_e(C) = \sum_{k \in C_e} P(k)Z \tag{8} \]

where \( P(k) \) is the popularity of the corresponding layer for segment \( k \), and \( Z \) is the segment size. Similarly, let \( C_f \) denote the set of segments remaining in cache when fine-grain replacement is used. We define its asymptotic byte hit ratio as:

\[ B_f(C) = \sum_{k \in C_f} P(k)Z \tag{9} \]

Notice that because the atomic replacement algorithm suffers more from fragmentation than the fine-grain algorithm, we have: \( |C_f| \geq |C_e| \).

In order to compare Eqs. 8 and 9, we derive the following properties:

Property 1: The popularity of every segment \( S(i, j, k) \in C_e \setminus C_f \) is equal or less than that of any segment \( S(i', j', k') \in C_f \setminus C_e \).

This property is proved by contradiction. By its design, the fine-grain replacement algorithm keeps in cache the layers (hence segments) with the highest per-layer popularity. The existence of \( S(i, j, k) \) violates this fact.

\[ B_f(C) \geq B_e(C) \tag{10} \]

From Property 1, we have that \( \forall S(i, j, k) \in C_e \setminus C_f, \forall S(i', j', k') \in C_f \setminus C_e, P(S(i, j, k)) \leq P(S(i', j', k')) \). Therefore:

\[ B_e(C) = \sum_{k \in C_e} P(k)Z \leq \sum_{k \in C_f \setminus C_e} P(k)Z \leq \sum_{k \in C_f} P(k)Z = B_f(C) \]

This reasoning can be pushed to the extreme to favor replacement based on per-segment popularity. However, there exists a tradeoff between replacement granularity and book-keeping overhead. Maintaining popularity at a finer granularity may result in a higher cache efficiency, but it also requires a higher book-keeping overhead. Given that multimedia streams usually consist of very large number of segments, we believe that per-layer popularity presents the best tradeoff. Some back-of-envelope calculation helps illustrate this point. Let the cache size be 10GB. Each stream has average size 2MB, 5 layers and segment size 1KB. Every popularity number takes 6 bytes since it is a floating point number. The per-segment popularity table for the cache is therefore 60MB. Consider that the popularity table is sorted and updated on every access, this number is quite large for most caches. Repeating this calculation for per-layer popularity gives us a popularity table size of 150KB, which is a much easier to fit in the main memory.

III. IMPACT OF BURSTY TRAFFIC

In this section we use per-stream completeness to illustrate the influence of bursty background traffic. The network topology in this simulation is identical to that described in Section IV. The data set consists...
of 10 streams, and the stream shown is the most popular stream. 95% of all requests are from the low bandwidth client. The Web traffic was generated using methods in [12].

Fig. 11 shows the results. We observe that both simulations show a qualitatively identical trend: when the client bandwidth is low, the higher layers of the most popular stream can be flushed out. The difference between the two is that the stream with bursty traffic seems to have more variable quality, and as a result the higher layers experience more fluctuation than the stream without bursty traffic. The reason is exactly the highly bursty nature of the self-similar background traffic. Because of this burstiness, it is difficult to understand the results using average available bandwidth when we are using bursty background traffic. However, a single bottleneck link makes this task much easier. This is another reason that we chose not to use bursty background traffic, in addition to its extraordinarily long execution time.

IV. EFFECT OF CONCURRENT REQUESTS

In this section we examine the impact of concurrent requests on quality improvement of cached streams. The network topology in this simulation is identical to that described in Section IV. The data set consists of 10 streams, and the stream shown is the most popular stream. 95% of all requests are from the low bandwidth client. We set the layer bandwidth to 2.5KB/s, therefore the bottleneck between the cache and the server can afford 2.8 layers. 80% of one request overlaps with a subsequent request.

Fig. 12 shows the requests. We observe that the results with and without concurrent requests are qualitatively similar. There are two subtle differences. First, with concurrent requests, it takes longer for layer 4 and 5 to achieve maximum quality. This is because of the lower available bandwidth for prefetching on the cache-server bottleneck link. However, the highest layer 5 experiences less fluctuation and higher quality. This is because other cached streams have lower quality due to limited bandwidth, hence leave more room for this layer.