Multimedia Systems (2003)
Digital Object Identifier (DOI) 10.1007/s00530-003-0109-0

Multimedia Systems

Network-aware partial caching for Internet streaming media*

Shudong Jin[1], **Azer Bestavros**[1], **Arun Iyengar**[2]

[1] Computer Science Department, Boston University, Boston, MA 02115, USA
[2] IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA

Abstract. The delivery of multimedia over the Internet is affected by adverse network conditions such as high packet loss rate and long delay. This paper aims at mitigating such effects by leveraging client-side caching proxies. We present a novel cache architecture and associated cache management algorithms that turn edge caches into accelerators of streaming media delivery. This architecture allows partial caching of media objects and joint delivery from caches and origin servers. Most importantly, the caching algorithms are both *network-aware* and *stream-aware*; they take into account the popularity of streaming media objects, their bit rate requirements, and the available bandwidth between clients and servers. Using Internet bandwidth models derived from proxy cache logs and measured over real Internet paths, we have conducted extensive simulations to evaluate the performance of various cache management algorithms. Our experiments demonstrate that network-aware caching algorithms can significantly reduce startup delay and improve stream quality. Our experiments also show that partial caching is particularly effective when bandwidth variability is not very high.

Key words: Web caching – Streaming media – Network measurement – Partial caching

1 Introduction

The increasing popularity of multimedia content on the Internet has stimulated the emergence of streaming media applications. Access to streaming media requires a high and stable transmission rate. To meet such requirements, customers and ISPs typically upgrade their connections to the Internet, e.g., by going to higher-bandwidth services. While necessary, this upgrade of the "last mile" bandwidth does not translate to improved quality of service for streaming media access. Specifically, for such upgrades to yield the desired effects, Internet and streaming media servers must be able to handle the increased demand. To that end, several issues need to be addressed.

First, it is not clear whether the physical bandwidth of the Internet can match the growth pace of the demand for bandwidth. Internet resources are shared by a large number of connections; an individual customer can only expect to get a portion that reflects his/her "fair share" of the physical available bandwidth. This is the result of requirements that network transport protocols for streaming media be "TCP-friendly" [12,24,30]. Thus, the bandwidth available to an individual connection is likely to be limited by the unpredictable sharing of Internet resources. Second, the Internet exhibits extreme diversity in terms of end-to-end packet loss rate and round-trip delay. This diversity is made worse by the bursty nature of these metrics as evidenced by findings in a number of studies on Internet traffic characteristics [11,22,13]. Finally, even if the Internet transport would meet its end of the bargain, streaming media servers may themselves become significant bottlenecks, especially at times of high usage.

Combined, these factors suggest the importance of efficient and robust streaming delivery mechanisms that go beyond the point-to-point, server-to-client delivery of streaming media content. Caching is a good example of such mechanisms. By placing streaming media objects closer to customers, network bandwidth requirements are reduced, user-perceived quality is improved, and demand on servers is decreased. However, efficient streaming media caching algorithms must consider several factors. They must consider the characteristics of streaming media access workloads such as the skewed popularity of streaming media objects and the heterogeneity of bit rate requirements. They must be network-aware, i.e., they should consider network conditions such as packet loss rate and end-to-end delay.

Paper contributions and overview: This paper proposes a novel technique that turns edge caches into accelerators of streaming media delivery. Partial or whole streaming media objects are placed in caches closer to clients to accelerate access and improve stream quality. The cache management algorithms we propose are both *stream-aware* and *network-aware*; they account not only for the popularity of streaming media objects but also for the bit rate requirements of these objects

* This research was supported in part by NSF (awards ANI-9986397 and ANI-0095988) and by IBM. Part of this work was done while the first author was at IBM Research in 2001.
Correspondence to: S. Jin (e-mail: jins@cs.bu.edu)

as well as network condition such as the available bandwidth between servers, clients, and caches. Using synthetically generated access workloads, our simulations show that the proposed algorithms can efficiently utilize cache space to reduce streaming application startup delay and improve stream quality. Our simulations are unique because they rely on models reflecting Internet bandwidth characteristics we observed in real proxy cache logs and measured over real Internet paths.

In the remainder of this paper, we first revisit related work in Sect. 2. Then we formalize the cache problem and propose our service acceleration algorithms. Section 4 describes the methodology of our performance evaluation experiments, and Sect. 5 presents results from our simulations. We end in Sect. 6 with conclusions and directions for future research.

2 Related work

Caching techniques have been widely used for traditional Web content such as HTML pages and image files [1,6,9, 18]. A difference between Web caching and streaming media caching is that Web caches can store nearly everything. This strategy is too expensive for large streaming media objects. Williams et al. [31] evaluated different Web caching policies. Wooster and Abrams [32] considered document retrieval delay in replacement algorithms. Cao and Irani [7] proposed cost-aware replacement algorithms, capitalizing on the variable-size variable-cost property of Web objects. Jin and Bestavros [15] proposed caching algorithms that capture long-term object popularity as well as the variable-size variable-cost nature of Web objects.

The emergence of streaming media applications on the Internet has resulted in an increased interest in effective streaming media delivery techniques. Recent studies have focused on streaming media workload characterization [2,4,5,10,20] and synthesis [16] as well as caching techniques [3,17,25,28, 29].

Several studies have considered the implications of workload characteristics on the performance of streaming media caching and prefetching techniques. Acharya et al. characterized streaming objects [2] and user access patterns [4]. Their work revealed several observations including the highly variable object sizes, the skewed popularity of objects, and the existence of temporal locality. Chesire et al. [10] analyzed a client-based streaming media workload and found that most streaming objects are small and that a small percentage of all requests are responsible for almost half of the total bytes served. They found that requests during periods of peak loads exhibit a high degree of temporal locality due to an object popularity profile that fits a Zipf-like distribution. Almeida el al. [5] analyzed workloads from two media servers used for educational purposes. They studied request arrival patterns, skewed object popularity, and user interactivities, extending results obtained earlier by Padhye and Kurose [20] on user interactivities with a media server.

Several studies have proposed caching schemes for streaming media objects. Due to the large size of media objects, effective use of cache space becomes more important. Aiming to reduce network bandwidth requirements, Wang et al. [29] proposed that proxy servers cache part of a stream with high bit rate. Their scheme is called video staging, which they combined with work-ahead smoothing techniques [26]. Sen et al. [28] proposed that proxies cache the initial frames of multimedia streams and use work-ahead smoothing. Prefix caching requires only small cache space to effectively reduce startup delay. Miao et al. [17] proposed selective caching to maximize the robustness of video streams against network congestion. Rejaie et al. [25] considered layered-encoded multimedia streams. They proposed a proxy caching mechanism to increase the delivered quality of popular streams. Acharya et al. [3] proposed the MiddleMan cooperative caching techniques, which utilize the aggregate storage of client machines. Reisslein et al. [23] developed and evaluated a caching strategy that explicitly tracks client request patterns and achieves higher hit ratios. Paknikar et al. [21] described a media caching framework that uses RTSP [27] in communication. They evaluated different cache replacement policies. Chan and Tobagi [8] studied various caching schemes, all employing circular buffers and partial caching. They analyzed the tradeoffs between disk storage space and network bandwidth. In the context of caching layered video, a recent work [33] investigated how layered video transmissions can be TCP-friendly. To the best of our knowledge, none of these schemes has considered measuring network bandwidth for caching algorithms, and none has used bandwidth models derived from real Internet measurements in their performance evaluation methodologies.

3 Caches as accelerators: proposed algorithms

In this section, we describe an architecture that uses caches to accelerate streaming media access. Then we formalize the cache management problem and propose a number of algorithms accordingly.

3.1 Architecture of streaming media delivery

We consider an Internet streaming media delivery architecture consisting of: (1) caches deployed at the edge of the Internet; (2) streaming media objects that are replicated either entirely or partially on these caches; and (3) clients whose requests are satisfied by servers and caches, possibly partially by servers and partially by caches. Figure 1 illustrates such an architecture.

We first examine how streaming media accesses may proceed in the absence of caches. A client may request any objects available from an origin server. To do so, the client measures the bandwidth between the server and itself. If the bandwidth is abundant, the client can play the stream immediately (it may still need to buffer a few initial frames of the stream in order to tolerate network jitters). If the bandwidth is not high enough to support immediate and continuous play of the stream at an acceptable quality, e.g., the object playback rate is 400 Kbps but the bandwidth is only 200 Kbps, two choices are possible: (1) it introduces a startup delay, during which it prefetches a prefix of the stream, before continuously playing the stream, or (2) it negotiates with the server and degrades the stream quality. For the previous example, the client can retrieve half of a layer-encoded object.

Now we turn to streaming media access in the presence of caches deployed closer to the client. Rather than relying solely

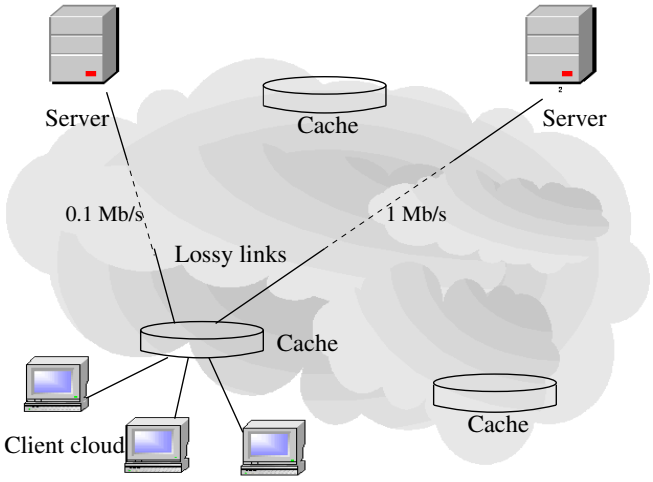


Fig. 1. Using caches to accelerate Internet streaming access. A client request is served partially by a proxy cache and partially by a server. Caches can eliminate bandwidth bottlenecks between clients and servers

on the origin server, a client could retrieve an entire or partial stream from a neighboring cache with higher bandwidth. It does so by measuring the bandwidth from the server and the bandwidth from the cache and then determining whether it is possible for both the server and the cache to jointly support immediate and continuous play of the stream. For the previous example, if half of the object has been cached, then the client can immediately and continuously play out the object; while the client is playing out the object from the cache, the other half is prefetched from the server. Generally, with caches, clients are less affected by the limited and variable bandwidth from the server.

3.2 Formalization of cache management problem

As we hinted earlier, "network awareness" is an important aspect of the streaming media caching techniques we propose in this paper. To appreciate this, consider the paths from a cache to two origin servers shown in Fig. 1, whereby one path has a bandwidth of 1 Mbps while the other can only support streaming at a 0.1 Mbps rate. Intuitively, it is more valuable to cache objects available from the second origin server.

Before we formalize the cache management problem, we make several assumptions (we relax some of them later in the paper). First, we assume that the bandwidth of a specific path is constant over some appropriate timescale. In a real setting, and as we explained earlier, the bandwidth achievable over a given Internet path may vary significantly with time. In Sect. 3.5, we relax this assumption and show how our algorithms can be modified to handle bandwidth variability. Second, we assume that the objective of the system is to minimize average startup delay. We define the startup delay to be the total delay perceived by the client before the playout of an object (at some acceptable quality) can begin. In Sect. 3.6, we also consider other objectives. Third, we assume that streaming media objects are encoded using a constant bit rate (CBR) technique. For variable bit rate (VBR) objects, we assume the use of the optimal smoothing technique [26] to reduce the burstiness of transmission rate. It has been shown that a relatively small buffer is sufficient to smoothen out VBR, essentially close to CBR. Hence this portion of buffer space is ignored in our analysis. Finally, we assume abundant bandwidth on the client side. Also, we assume that clients behind a caching proxy (a client cloud) are homogeneous.

Let N be the number of media objects available for access. For any such object i, we denote by T_i the object's duration in seconds, by r_i the object's CBR encoding in Mbps, by λ_i the arrival rate of requests for that object, and by b_i the bandwidth between the cache and the original server storing that object. The notation y^+ means that $y^+ = y$ if $y > 0$ and 0 otherwise.

Let C denote the total capacity of the cache, and let x_i denote the size of the cached part of object i. Upon requesting object i, the playout of that object must be delayed by $[T_i r_i - T_i b_i - x_i]^+/b_i$. Notice that $T_i r_i$ reflects the overall size of the requested object and that $T_i b_i$ reflects the size of the portion of the object that can be streamed during playout. Note it is assumed that bottleneck bandwidth is not at the client.

The optimization problem is thus to find a set of values $\{x_i, 1 \le i \le N\}$ that would minimize the average startup delay of all streaming media accesses, that is, to minimize

$$\frac{1}{\sum_{i=1}^{N} \lambda_i} \sum_{i=1}^{N} \lambda_i [T_i r_i - T_i b_i - x_i]^+ / b_i$$

subject to the constraint $\sum_{i=1}^{N} x_i \le C$, $x_i \ge 0$. Notice that the constant factor $1/\sum_{i=1}^{N} \lambda_i$ can be ignored.

3.3 An optimal solution for populating caches

We derive the optimal solution under static conditions. By static conditions, we mean the cache content is static, i.e., no replacement is necessary. By optimal solution, we mean caching decisions are made with prior knowledge of request arrival rates. We obtain the optimal solution by solving the above optimization problem.

First, for an object i, if $r_i \le b_i$ (i.e., the bandwidth is higher than the object's bit rate), then there is no need to cache that object (i.e., $x_i = 0$).

Now we consider all other objects. Let I denote the set of objects whose bit rate is higher than the bandwidth. The above optimization problem is equivalent to minimizing:

$$\sum_{i \in I} \lambda_i (T_i r_i - T_i b_i - x_i)/b_i,$$

subject to the constraint $\sum_{i \in I} x_i \le C, 0 \le x_i \le (r_i - b_i)T_i$. Notice that we restrict x_i to be less than or equal to $(r_i - b_i)T_i$ since a larger x_i does not yield more delay reduction.

The above minimization is equivalent to maximizing $\sum_{i \in I} \lambda_i x_i / b_i$. This is a fractional knapsack problem with has the following optimal solution: the caching algorithm chooses those objects with the highest λ_i/b_i ratios and caches them up to $(r_i - b_i)T_i$ until the cache is used up.

The above analysis assumes that b_i does not change with time and is independent of x_i. There is a subtle problem. A larger x_i results in lower bandwidth requirements on the path to the server since the client needs to retrieve a smaller portion

during T_i; by contrast, decreasing x_i results in higher bandwidth requirements. Therefore, there may be self-induced congestion on shared paths to the servers. However, this factor is less critical for two reasons. First, the servers satisfy requests from many clients and proxies across the Internet. Here we are considering a single proxy cache, where the change in b_i due to x_i is limited. For example, if a server has 100 connections on average, but there are only a few connections from the proxy we are considering, then x_i will not affect the available bandwidth (fair share) very much. Second, our algorithms are adaptive to changes in b_i. If at some time a value of x_i is decided upon, but subsequent probes indicate a larger b_i than expected, then our algorithm will automatically decrease x_i. We should also point out that an algorithm working in such an adaptive fashion may not be optimal. In order to optimize the caching algorithm while considering the dependence of b_i on x_i, the caching proxy needs full knowledge about the load on the server and in the network. To have such knowledge is often difficult and very costly.

3.4 Dealing with unknown request rates through replacement

In practice, caching algorithms have no prior knowledge of request arrival rates. Thus, the optimal solution derived in the previous section is not practical. To approximate the optimal solution, we propose a cache replacement algorithm. Our cache replacement algorithm estimates the request arrival rate λ_i of each object by recording the number or frequency of requests to each object, which we denote by F_i. Our cache replacement algorithm works as follows.

As before, if the bit rate of an object is lower than the measured bandwidth to the server, i.e., if $r_i \leq b_i$, then the object is not cached. Otherwise, we define the *utility* of object i as the ratio F_i/b_i. The cache replacement algorithm always caches those objects with the highest utility value. The size of the cached part of object i is up to $(r_i - b_i)T_i$.

Such a replacement algorithm can be implemented with a heap that uses the utility value as the key. Note that on each access to an object, the object's utility value is increased. Therefore, the replacement algorithm may evict other objects. The processing overhead for heap operations is $\mathrm{O}(\log n)$, where n is the number of objects in the cache.

3.5 Dealing with bandwidth variability through overprovisioning

In our exposition so far, we assumed that the bandwidth of a path is a constant. In realistic settings, the bandwidth of an end-to-end path may change over time, possibly drastically and unpredictably. Without prior knowledge of bandwidth variability, it is impossible to derive an optimal solution for caching. Hence we use a heuristic modification of our caching algorithm. With such a modification, we do not have to assume, for example, that bandwidth during prefetching is the same as that during playback.

The basic idea behind our heuristic is to make partial caching decisions based on a more conservative estimation (i.e., underestimate) of bandwidth. Such a conservative estimate of bandwidth would result in caching more than the minimum $T_i(r_i - b_i)$ needed for object i. To understand why we need to do so, we observe that when bandwidth varies significantly, if we only cache $T_i(r_i - b_i)$, then it is very possible that this portion of object i will not be enough to hide the access delay. But how much of the object would be enough to cache? Intuitively, such a determination should depend on the amount of bandwidth variability. If bandwidth does not vary much, then caching $T_i(r_i - b_i)$ is close to optimal. If bandwidth varies, then more conservative caching decisions are warranted; the larger the variations, the larger the portion of the object to be cached.

In the extreme case, the most conservative heuristic would yield a caching algorithm that chooses those objects with the highest λ_i/b_i ratios and would cache them up to r_iT_i (i.e., it would cache whole objects) until the cache is used up. We use the term *integral* caching to refer to techniques that restrict cached content to be of complete objects. Thus, integral caching disallows partial caching. With integral caching, the cache can be used up quickly since it would accommodate fewer objects. As we show in Sect. 5, such an approach is only advantageous when bandwidth variability is extremely high.

3.6 Addressing other caching objectives

So far we have assumed that the primary objective of caching is to reduce delay. However, caching may be desirable for other objectives. We consider the following application as an example. Each streaming media object has an associated *value*.[1] When a client requests *immediate* service of a streaming media object, the cache decides whether the server and the cache can jointly support it. There is an added value if the object is played. Here, the objective is to maximize the *revenue* of the cache. We formalize the problem as follows. Let V_i denote the value of the i-th object. We need to find the set of objects I to maximize $\sum_{i\in I}\lambda_i V_i$, subject to the constraint $\sum_{i\in I}[T_ir_i - T_ib_i]^+ \leq C$. Note that we need to partially cache $[T_ir_i - T_ib_i]^+$ of the i-th object to provide immediate service to the requests.

We solve this problem as follows. As before, those objects with a bit rate lower than bandwidth need not be cached. The problem is thus to decide on a set of other objects to be partially cached. This is a knapsack problem that is NP-hard. A simple but suboptimal greedy solution is caching those objects with the highest $\frac{\lambda_i V_i}{T_ir_i - T_ib_i}$ ratio.

3.7 Implementation issues

The techniques discussed thus far assume that it is possible for a cache to measure the bandwidth from the origin server. Two approaches are possible: active measurement and passive measurement. Using active measurement approaches, end systems (clients, servers, or caches) send a few probing packets and then estimate bandwidth based on observed packet loss and/or delay measurements [14]. For example, for TCP-friendly streaming media transports, the available bandwidth from the server should be close to TCP's throughput, which

[1] This value may reflect the relative importance of the object in a multimedia setting, e.g., audio vs. video.

is inversely proportional to the square root of packet loss rate and round-trip time [19]. Both packet loss rate and round-trip time could be measured using end-to-end approaches. Clearly, active measurement approaches incur some overhead. Using passive measurement approaches, end systems estimate bandwidth by observing the throughput of past connections. Such approaches do not introduce additional network overhead but may not be accurate as bandwidth may change drastically over time. Either way, the estimation of bandwidth can be easily combined into the caching algorithms, and the modification to the caching decision need not be updated very frequently.

The techniques discussed thus far assume that it is possible for a client to be jointly served by both the origin server and the cache. Clearly, some coordination is needed to ensure that the content served from the cache and the origin server are complementary. One approach to ensure this is to restrict caching to object prefixes rather than any interval of an object. When a client starts playing the prefix of an object fetched from the cache, the remainder of the object is prefetched from the corresponding server. Certainly there are many implementation details such as partial object (prefixes or fine-grain segments) maintenance, prefetching decisions, and partial object service protocols.

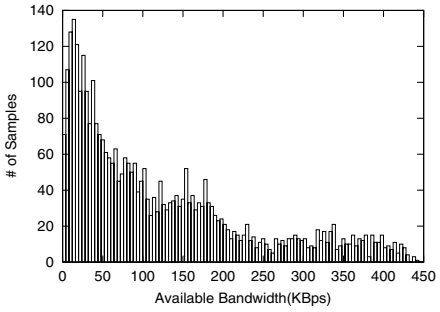
4 Evaluation methodology

This section describes the methodology used in our simulations. We first present results of analyses of NLANR proxy cache [18] logs as well as results from measurement experiments we conducted on representative Internet paths to get realistic bandwidth models (base bandwidth distributions and variability characteristics) for use in our simulations. Next, we describe how synthetic streaming media access workloads were generated to drive our simulations. Finally, we describe the performance metrics used to compare the performance of various algorithms.

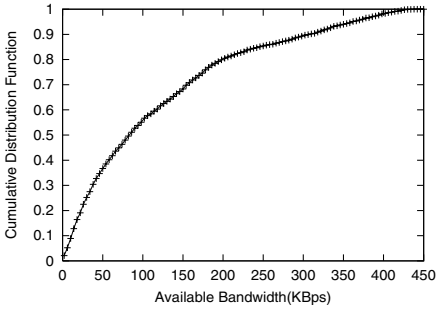
4.1 Network bandwidth modeling

For our performance evaluation simulation experiments, we needed to adopt a model of the base bandwidth over various paths and see how such base bandwidth may vary over time. We derived such models using two methods: (1) analysis of proxy cache logs and (2) measurement of observed bandwidth over a set of real Internet paths. We are not claiming that our measurement and analysis capture all aspects of Internet bandwidth. However, they can help us understand the heterogeneity and variability of Internet bandwidth, and experiments based on them can help us understand how they impact the effectiveness of various caching algorithms.

We obtained bandwidth statistics by analyzing the NLANR proxy cache logs. We used a 9-d log of site UC during 12–20 April 2001. This site has a popular client (a low-level proxy). We observed those missed requests for objects larger than 200 KB. A bandwidth sample is obtained by dividing the size of an object by the connection duration. We used requests for large objects since the long duration of HTTP connections results in more accurate measurement of bandwidth. We used the missed requests so that the objects were served by the original servers and not by the NLANR proxy cache.



a Histogram



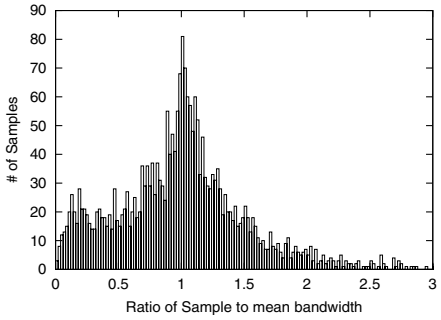
b Cumulative distribution

Fig. 2a,b. Internet bandwidth distribution observed in NLANR cache logs. The histogram in **a** is generated by showing the number of samples in each 4-Kbps slot. Each sample is obtained by computing the throughput of a HTTP connection. The cumulative distribution in **b** is derived from the histogram

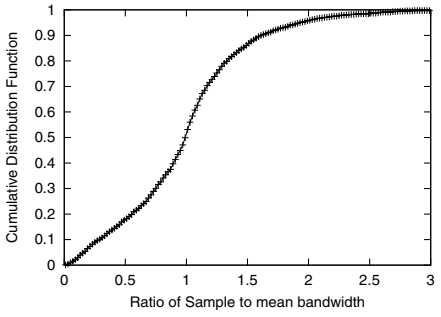
Figure 2a shows a histogram of the bandwidth values observed from analysis of the log. It shows that the bandwidth of various paths varies drastically. The possible causes are (1) the packet loss rate and delay vary significantly from one server to another, and (2) server loads can be quiet different, i.e., congestion at the server. Figure 2b shows the cumulative distribution of bandwidth. We found that 37% of the requests have bandwidth lower than 50 Kbps, and 56% have bandwidth lower than 100 Kbps. We also note that a large portion of the requests yielded much higher bandwidth. This heterogeneity of bandwidth suggests that caching algorithms could benefit from differentiating between objects from origin servers with widely different bandwidth.

We also observed the bandwidth variability for requests to the same server (i.e., using the same Internet path) over different times. To do this, we first computed the average bandwidth of each path. Then we took the ratio of the bandwidth samples to the average value. Figure 3a shows the distribution of this sample-to-mean ratio. It indicates that the bandwidth of a single path may vary significantly. Figure 3b shows the cumulative distribution function. While the sample-to-mean ratio can be very large, the CDF plot indicates that in about 70% of the cases, the sample bandwidth is 0.5–1.5 times the mean. One question is how such variation affects the performance of caching algorithms.

It is important to note that the analysis of NLANR logs cannot give us a realistic model for bandwidth variability, rather, it is likely to magnify bandwidth variations. It provides a fairly pessimistic (i.e., bursty) model of bandwidth variability compared to what one would observe for a typical transaction. In



a Histogram



b Cumulative distribution

Fig. 3a,b. Variation of bandwidth observed in the NLANR cache logs. The average bandwidth of each path is computed. Then we take the ratio of the samples to the average value. Panel **a** shows the histogram of this ratio. Panel **b** shows the cumulative distribution function of the ratio

particular, Fig. 3a is over an unrestricted time scale, i.e., the requests were not restricted to a particular time of day. Specifically, variability that results from diurnal traffic patterns (over time scales of many hours) may indeed be much larger than variability over shorter time scales. Other factors that contribute to this include our inability to separate variability due to network conditions as opposed to NLANR proxy caching load.

To obtain more realistic bandwidth variability models, we measured the bandwidth of real Internet paths over long periods. We repeatedly downloaded large files from Web servers around the world. Each download takes 5 to 30 s. We carefully scheduled the downloads to avoid overlapping them on the client machine (IP address: 128.197.12.3). Figure 4 shows the bandwidth evolution of three such paths spanning over 30 to 45 h (starting from 2 pm, 15 October 2001). We have also generated sample-to-mean ratio histograms for three of these paths, also shown in Fig. 4. The following observations were made. (1) The magnitude of bandwidth variability depends largely on the paths. For instance, the INRIA server appears to have much lower variability than the other two servers. (2) All paths have much lower variability than those obtained through analysis of the NLANR cache logs. This comparison was done by computing the coefficient of variation for the histogram plots in Fig. 4 and contrasting it to the coefficient of variation obtained from Fig. 3.

Table 1. Characteristics of the synthetic workload

Number of objects	5,000
Object popularity	Zipf-like
Number of requests	100,000
Request arrival process	Poisson, mean IAT ≈ 30 s
Object size	Lognormal, $\mu = 3.85$, $\sigma = 0.56$
Object bit rate	48 Kbps
Total storage	790 GB
Cache size	$4 \sim 128$ GB
Bandwidth distribution	NLANR logs
Bandwidth variation	NLANR logs and measurement

4.2 Synthetic workload generation

We used the GISMO toolset [16] to generate a synthetic workload for the purpose of driving our simulation experiments. Table 1 lists the characteristics of this workload.

The workloads used in our simulations consisted of requests to $N = 5000$ unique streaming media objects whose popularity follows a Zipf-like distribution [34]. With Zipf-like distributions, the relative popularity of an object is proportional to $r^{-\alpha}$, where r is the rank of the object's popularity. The probability that the i-th ranked object is accessed is $r^{-\alpha} / \sum_{j=1}^{N} j^{-\alpha}$. The default value for α is 0.73; we also present results with a range of values.

Each workload (for a single simulation run) consisted of 100,000 requests, spanning over one month. Request arrivals were generated using a Poisson process, i.e., the requests arrive independently with mean interarrival time (IAT) close to 30 s. The duration (in minutes) of the streaming media objects follow a Lognormal distribution with parameter $\mu = 3.85$ and $\sigma = 0.56$. The average duration of the objects is about 79 K frames, or about 55 min since we assume 24 frames per second. The bit rate of the objects is 48 Kbps. The total unique object size is 790 GB.

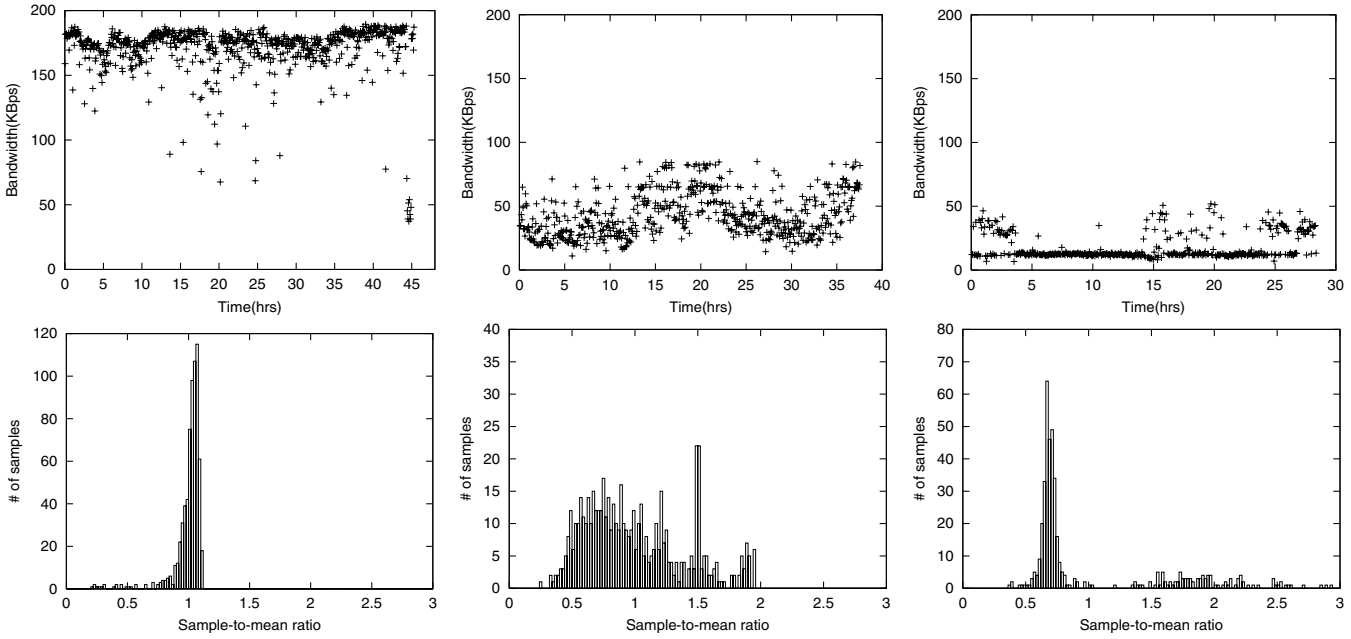
Although we have changed the numeric values shown in Table 1 and generated workloads of different characteristics, we found that the relative performance of the various algorithms is fairly similar to that observed using the base values.

In our simulation experiments, we varied the cache size from 4 GB, about 0.5% of the total unique object size, to 128 GB, about 16.9% of the total unique object size. The bandwidth between the cache and the servers follows the sample distribution from the NLANR logs (Fig. 2). When we study the impact of bandwidth variability, we generate bandwidth instances varying according to the models depicted in Figs. 3 and 4.

4.3 Performance metrics

In our experiments, we considered a number of performance metrics, each reflecting a different objective of caching. We discuss these metrics below.

- Caching algorithms may aim at reducing backbone traffic. To capture the effectiveness of a caching algorithm in reducing backbone traffic, we define the *traffic reduction ratio* as the fraction of the total bytes served by a cache. Traffic reduction ratio does not necessarily reflect



a INRIA, France (138.96.64.17) **b** Taiwan (140.114.71.23) **c** Hong Kong (143.89.40.4)

Fig. 4a–c. Bandwidth variation of real paths from Boston University (IP address: 128.197.12.3) to three servers. Top plots show the sample bandwidth in time series. One sample was taken every 4 min. Bottom plots show the histogram of the sample-to-mean ratio

user-perceived quality. To capture this, we introduce three additional metrics.

- When bandwidth (assisted by cache or not) is not enough to support the immediate playout of a stream, the client may choose to wait for service. During this waiting period, the client prefetches a prefix before continuously playing the stream. In this case, we are interested in the average waiting time, which we call the *average delay*.
- When bandwidth (assisted by cache or not) is not enough to support a full stream, the client may choose to downgrade the quality of the stream to play it out immediately. In this case, we are interested in the *average stream quality*, which we define as the percentage of the full stream that yields an immediate playout. For example, if a layer-encoded object has four layers with equal size but only three layers can be supported, then the quality is 0.75.

Different caching algorithms have different objectives. A given algorithm may optimize one performance metric but sacrifice others. For example, one algorithm (e.g., LFU) caches objects based on their access frequency only, not based on the network bandwidth. It aims at improving hit ratios and reducing traffic but not the average delay or stream quality. Therefore, we need to compare several performance metrics so that we can understand the tradeoffs of different algorithms.

5 Simulation results

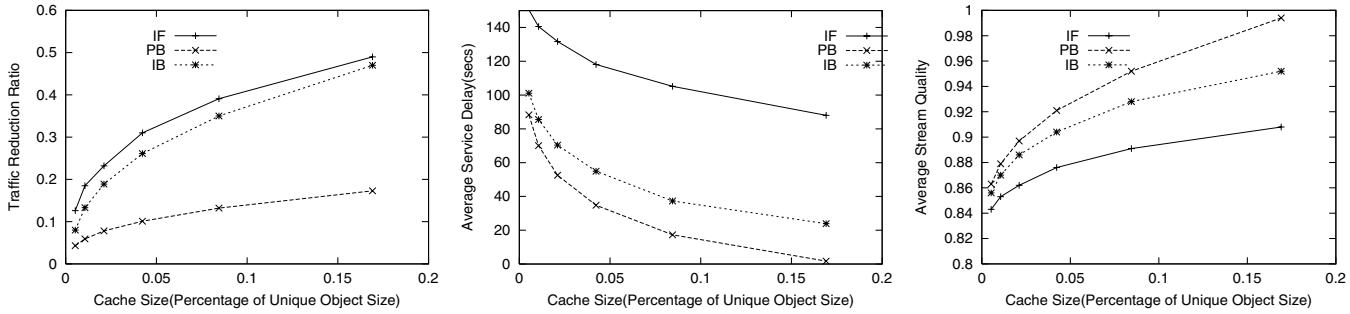
This section presents results from our simulation experiments in which we compared the performance of various caching algorithms and heuristics presented in this paper. Specifically, we study how bandwidth variability affects the performance of caching algorithms. Each result is obtained by averaging ten runs of the simulated system.

5.1 Performance of replacement algorithms

We conducted the first set of simulations to compare the performance of three cache replacement algorithms. The first algorithm caches those objects with the highest request arrival rates and only allows whole objects to be cached. We call this *integral frequency-based caching*, or IF caching for short. The second algorithm is the one described in Sect. 3.3. It caches those objects from origin servers without abundant bandwidth for streaming, i.e., preference is given to those with a higher λ_i/b_i ratio. Also, it allows partial caching. We call this *partial bandwidth-based caching*, or PB caching for short. The third algorithm is the one described in Sect. 3.5. It caches those objects with the highest λ_i/b_i ratio, but does not allow partial caching. We call this *integral bandwidth-based caching*, or IB caching for short. These three algorithms estimate object access frequency and network bandwidth progressively and make eviction decisions when the cache is used up.

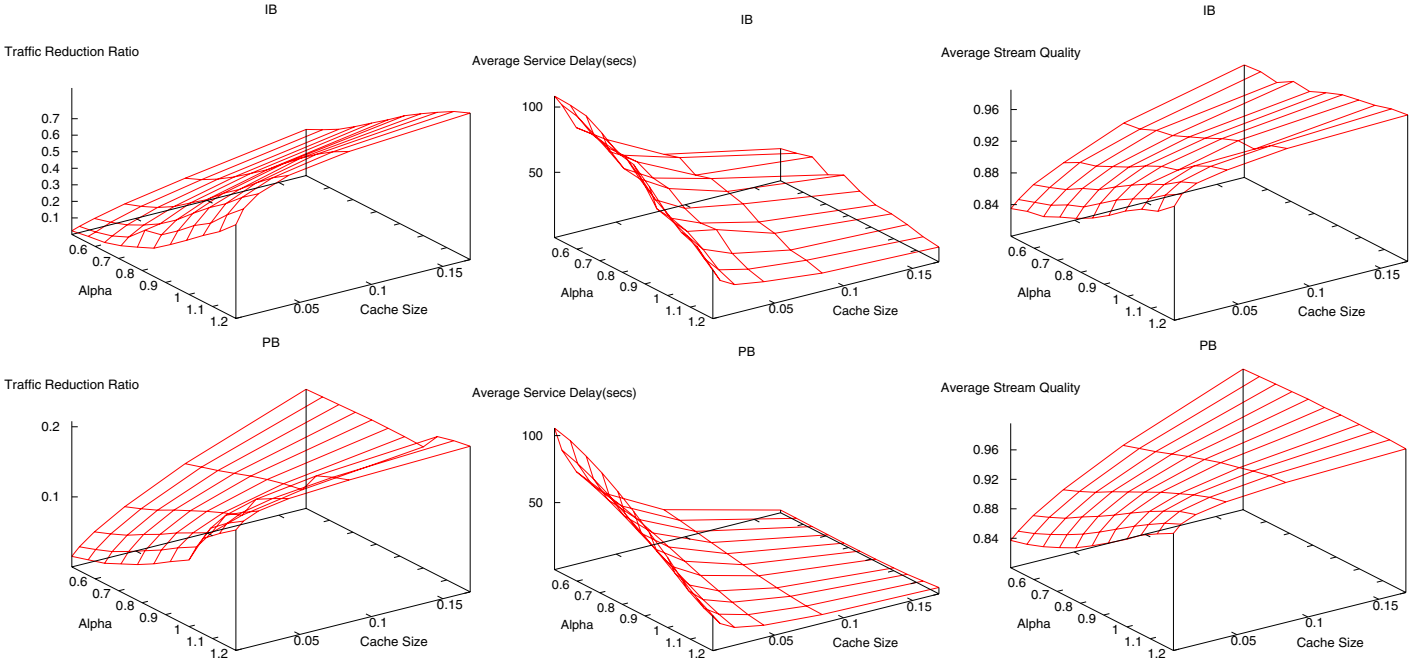
Figure 5 shows the results we obtained from our simulation experiments. For each run of the simulation program, we first warm up the cache using the first half of the workload and then compute the performance metrics from the second half. For this set of simulations, we assumed that the bandwidth of a path does not vary over time.

As depicted in Fig. 5a, IF caching achieves the highest backbone traffic reduction, while PB caching achieved the least such reduction. This is expected since PB caching does not cache whole objects even if the objects are very hot. Alternatively, Fig. 5b,c shows that PB caching achieves the lowest average delay and the highest average quality, whereas IF caching achieves the worst results for these metrics. Even when cache size is relatively high, the inferiority of IF caching is still obvious. The reason is that it results in caching hot objects even when there is abundant bandwidth for streaming



a Traffic reduction ratio **b** Average delay **c** Average stream quality

Fig. 5a–c. Comparison of IF, PB, and IB cache replacement algorithms under constant bandwidth assumption



a Traffic reduction ratio **b** Average delay **c** Average stream quality

Fig. 6a–c. Effect of Zipf-like popularity distribution parameter α

such objects from origin servers, thus limiting its ability to effectively use the cache.

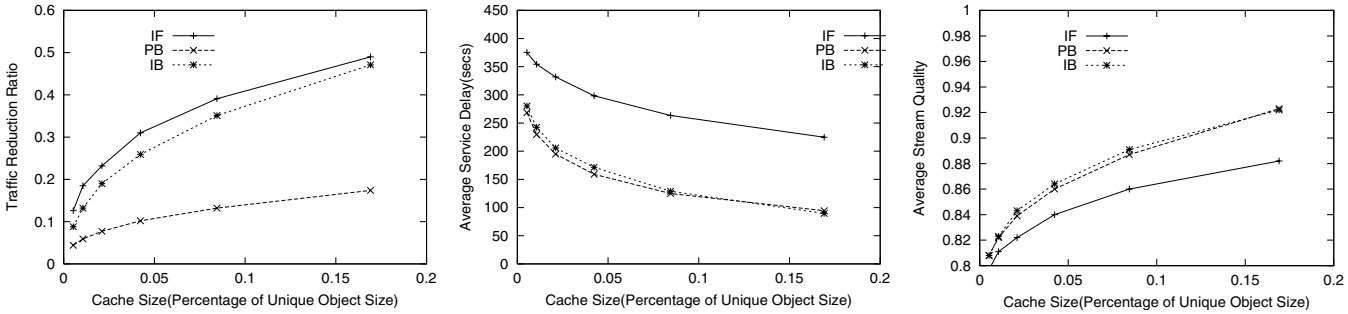
Figure 5a shows that IB caching yields performance metrics that lie in between those of the other two algorithms. IB caching achieves high traffic reduction ratios and is reasonably close to PB caching in terms of average delay and stream quality. An additional advantage of IB caching is simplicity: it caches whole objects, making it unnecessary to coordinate joint service by origin servers and caches, whereas PB caching requires a client to download a stream from the cache and the origin server in parallel.

5.2 Impact of temporal locality of reference

We conducted a second set of simulations to study the effect of the skewness, i.e., parameter α of the Zipf-like distribution governing streaming media object popularity. This skew is a measure of temporal locality of reference. When α increases, temporal locality in the workload is intensified. In simulations, we varied α from 0.5 to 1.2. Figure 6 shows the results of these simulations with respect to the various performance metrics. Only the results of IB caching and PB caching are presented. In general, intensifying temporal locality results in performance gains for both algorithms. Moreover, the relative performance of the algorithms does not seem to change: IB caching obtains much higher traffic reduction ratios, whereas PB caching achieves moderately better average delay and average stream quality.

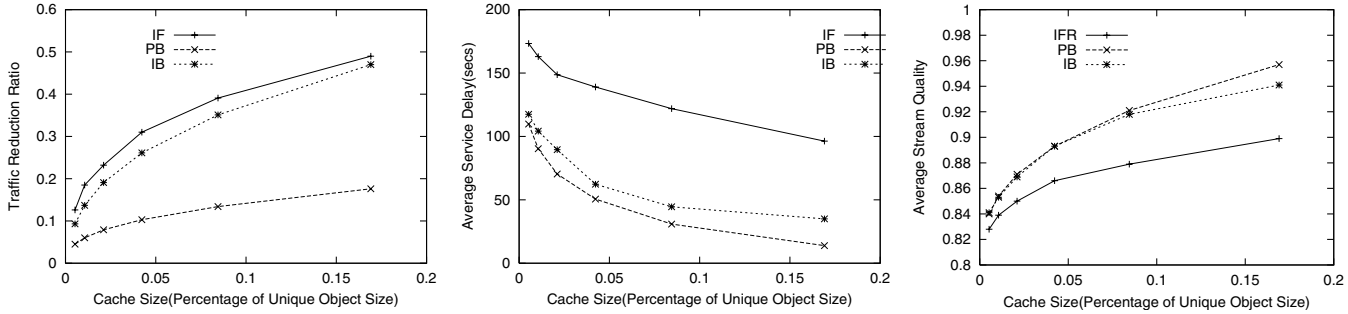
5.3 Impact of bandwidth variability

We conducted a third set of simulations to study the impact of bandwidth variability on the performance of the three caching algorithms under consideration. In these simulations, we allowed the bandwidth of a path to change over time. We generated such variations as follows: each path has an average bandwidth that follows the distribution in Fig. 2, but an instance of the bandwidth is obtained by multiplying that bandwidth by a random ratio that follows the distribution in Fig. 3. The results from this set of simulations are shown in Fig. 7.



a Traffic reduction ratio **b** Average delay **c** Average stream quality

Fig. 7a–c. Comparison of different caching algorithms under variable bandwidth assumption. The variation is obtained from cache logs



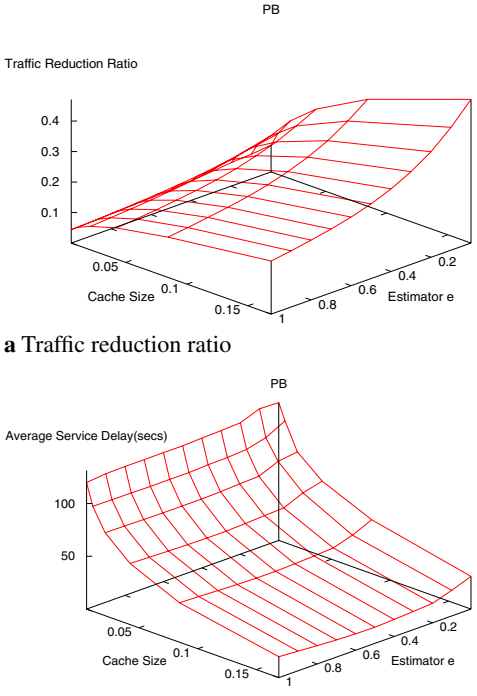
a Traffic reduction ratio **b** Average delay **c** Average stream quality

Fig. 8a–c. Comparison of different caching algorithms under variable bandwidth assumption. The variation is measured from real Internet paths

Comparing Fig. 7a with Fig. 5a shows no noticeable difference in traffic reduction ratio for all three algorithms. However, the other two performance metrics (average delay and average stream quality) exhibit major differences. First, bandwidth variability results in increased delay and degraded stream quality for all three algorithms. When bandwidth varies drastically over time, partial caching becomes less effective in accelerating access and improving stream quality. High bandwidth variability makes it difficult to choose the right objects and the right fraction of each object to cache. Second, IB caching is no worse than PB caching. This is because the optimality of PB caching depends on the constant bandwidth assumption. When bandwidth is insufficient due to variability, clients see higher delays or lowered quality. On the contrary, IB caching makes conservative caching decisions and caches whole objects with the highest λ_i/b_i ratio. That is, it caches those objects with high access frequency and low bandwidth for streaming.

As we discussed before, the bandwidth observed from NLANR cache logs appears to have higher variability than real Internet path measurements we performed. To that end, we conducted a fourth set of simulations using the lower variability modeled by the distribution in Fig. 4. The results of these simulations are shown in Fig. 8. We observe that with this more realistic setting, PB caching outperforms the other integral algorithms (IF and IB) in reducing delay and improving stream quality. These results suggest that the choice of partial vs. integral caching should indeed depend on the level of bandwidth variability.

As has been noted repeatedly, IB caching is the algorithm that is most tolerant to bandwidth variability, whereas PB caching is the most susceptible to bandwidth variability. Thus it is interesting to find out if a hybrid algorithm could bridge these two extremes. To do so, we conducted a fifth set of simulations in which the average bandwidth from the origin server was underestimated by multiplying it by a



a Traffic reduction ratio

b Average delay

Fig. 9a,b. Effect of partial caching based on bandwidth estimation

constant e. In Fig. 9, we plot the traffic reduction ratio and the average delay as e changes between 0 and 1. This gives us the performance of a spectrum of partial bandwidth-based caching algorithms that range from IB caching (when $e = 0$) to PB caching (when $e = 1$). Figure 9 shows that IB caching is always better in reducing network traffic but that choosing a moderate value of e results in slightly lower average delay.

6 Conclusion

In this paper, we proposed a cache architecture and associated algorithms that turn Internet edge caches into accelerators of streaming media delivery. Our algorithms allow partial caching of streaming media objects and joint delivery of content from caches and origin servers. More importantly, they are both *network-aware* and *stream-aware* in that they optimize cache occupancy decisions based on knowledge of available bandwidth from an origin server as well as streaming media object properties such as object popularity and encoding bit rate characteristics.

The performance evaluation experiments we performed using realistic streaming media access workloads augmented with a model of Internet path bandwidth measurements demonstrate the effectiveness of caching mechanisms that take into consideration network bandwidth information. Our simulation experiments have shown that bandwidth variability may impact the effectiveness of partial caching in reducing startup delay and stream quality. However, they also show that simple overprovisioning heuristics work reasonably well, even in the presence of high bandwidth variability.

Our ongoing and future work will proceed on a number of fronts. First, as evident from our findings, accurate measurement of network bandwidth and jitter is key to efficient streaming media delivery techniques. We are in the process of augmenting GISMO [16] with realistic models of Internet path bandwidth and bandwidth variability distributions. Second, we are investigating the possibility of combining our partial caching mechanisms with other streaming content delivery techniques such as patching and batching techniques at caching proxies. Finally, given the importance of real-time bandwidth measurement techniques, we are considering approaches that integrate active bandwidth measurement techniques [14] into proxy caches. This would allow us to prototype the acceleration architecture proposed in this paper using off-the-shelf caching proxies.

References

1. Abrams M, Standridge CR, Abdulla G (1995) Caching proxies: limitations and potentials. In: Proceedings of the WWW conference, Darmstadt, Germany, December 1995
2. Acharya S, Smith B (1998) An experiment to characterize videos stored on the Web. In: Proceedings of S&T/SPIE conference on multimedia computing and networking (MMCN), San Jose, January 1998
3. Acharya S, Smith B (2000) MiddleMan: A video caching proxy server. In: Proceedings of the 10th international workshop on network and operating system support for digital audio and video (NOSSDAV), Chapel Hill, NC, June 2000
4. Acharya S, Smith B, Parns P (2000) Characterizing user access to video on the World Wide Web. In: Proceedings of the S&T/SPIE conference on multimedia computing and networking (MMCN), San Jose, January 2000
5. Almeida J, Krueger J, Eager D, Vernon M (2001) Analysis of educational media server workloads. In: Proceedings of the 11th international workshop on network and operating system support for digital audio and video (NOSSDAV), Port Jefferson, NY, June 2001
6. Bestavros A, Carter R, Crovella M, Cunha C, Heddaya A, Mirdad S (1995) Application level document caching in the Internet. In: Proceedings of SDNE: the 2nd international workshop on services in distributed and networked environments, Whistler, BC, Canada, June 1995
7. Cao P, Irani S (1997) Cost-aware WWW proxy caching algorithms. In: Proceedings of the USENIX symposium on Internet technologies and systems (USITS), Monterey, CA, December 1997
8. Chan SHG, Tobagi F (2001) Distributed servers architecture for networked video services. IEEE/ACM Trans Netw 9(2):125–136
9. Chankhunthod A, Danzig P, Neerdaels C, Schwartz MF, Worrell KJ (1996) A hierarchical Internet object cache. In: Proceedings of USENIX, San Diego, January 1996
10. Chesire M, Wolman A, Voelker G, Levy H (2001) Measurement and analysis of a streaming workload. In: Proceedings of the USENIX symposium on Internet technologies and systems (USITS), San Francisco, March 2001
11. Crovella M, Bestavros A (1996) Self-similarity in World Wide Web traffic: evidence and possible causes. In: Proceedings of ACM SIGMETRICS, Philadelphia, May 1996
12. Floyd S, Fall K (1999) Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Trans Netw 7(4):458–472
13. Glossglauser M, Bolot J (1996) On the relevance of long-range dependence in network traffic. In: Proceedings of ACM SIGCOMM, Stanford, August 1996
14. Harfoush K, Bestavros A, Byers J (2001) Measuring bottleneck bandwidth of targeted path segments. Technical Report BUCS-TR-2001-016, Computer Science Department, Boston University, July 2001
15. Jin S, Bestavros A (2000) Popularity-aware greedydual-size Web proxy caching algorithm. In: Proceedings of IEEE ICDCS, Taipei, Taiwan, ROC, April 2000
16. Jin S, Bestavros A (2001) GISMO: Generator of streaming media ojects and workloads. ACM SIGMETRICS Perform Eval Rev 29(3):2–10
17. Miao Z, Ortega A (1999) Proxy caching for efficient video services over the Internet. In: Proceedings of PVW, Columbia University, New York, April 1999
18. National Laboratory for Applied Network Research. http://ircache.nlanr.net/
19. Padhye J, Firoiu V, Towsley D, Kurose J (1998) Modeling TCP throughput: a simple model and its empirical validation. In: Proceedings of ACM SIGCOMM, Vancouver, BC, Canada, August 1998
20. Padhye J, Kurose J (1998) An empirical study of client interactions with a continuous-media courseware server. In: Proceedings of the 8th international workshop on network and operating system support for digital audio and video (NOSSDAV), Cambridge, UK, June 1998
21. Paknikar S, Kankanhalli M, Ramakrishnan KR, Srinivasan SH, Ngoh LH (2000) A caching and streaming framework for multimedia. In: Proceedings of ACM MULTIMEDIA, Los Angeles, October 2000

22. Paxson V (1994) Wide-area traffic: the failure of Poisson modeling. In: Proceedings of ACM SIGCOMM, London, August 1994
23. Reisslein M, Hartanto F, Ross KW (2000) Interactive video streaming with proxy servers. In: Proceedings of the 1st international workshop on intelligent multimedia computing and networking (IS&T/SPIE conference on multimedia computing and networking (MMCN)), San Jose, February 2000
24. Rejaie R, Handley M, Estrin D (1999) RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In: Proceedings of IEEE INFOCOM, New York, April 1999
25. Rejaie R, Yu H, Handley M, Estrin D (2000) Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet. In: Proceedings of IEEE INFOCOM, Tel-Aviv, March 2000
26. Salehi JD, Zhang ZL, Kurose JF, Towsley D (1996) Supporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing. In: Proceedings of ACM SIGMETRICS, Philadelphia, May 1996
27. Schulzrinne H, Rao A, Lanphier R (1998) Real Time Streaming Protocol(RTSP), RFC 2326, The Internet Society, April 1998
28. Sen S, Rexford J, Towsley D (1999) Proxy prefix caching for multimedia streams. In: Proceedings of IEEE INFOCOM, New York, April 1999
29. Wang Y, Zhang ZL, Du DH, Su D (1998) A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers. In: Proceedings of IEEE INFOCOM, San Francisco, March 1998
30. Widmer J, Denda R, Mauve M (2001) A survey on tcp-friendly congestion control. IEEE Netw, 15(3):28–37
31. Williams S, Abrams M, Standridge CR, Abdulla G, Fox EA (1996) Removal policies in network caches for World-Wide Web documents. In: Proceedings of ACM SIGCOMM, Stanford, CA, August 1996
32. Wooster R, Abrams M (1997) Proxy caching that estimates page load delays. In: Proceedings of the WWW conference, Santa Clara, CA, April 1997
33. Zink M, Griwodz C, Schmitt J, Steinmetz R (2002) Exploiting the fair share to smoothly transport layered encoded video int o proxy caches. In: Proceedings of the S&T/SPIE conference on multimedia computing and networking (MMCN), San Jose, January 2002
34. Zipf GK (1929) Relative frequency as a determinant of phonetic change. Reprinted from Harvard Studies in Classical Philology, XL