A Caching and Streaming Framework for Multimedia

Shantanu Paknikar
Wipro Technologies
Electronic City, Bangalore 560001
+91-80-5588613
shantanu.paknikar@wipro.com

Mohan Kankanhalli
School of Computing
National University of Singapore
Kent Ridge, Singapore
(65) 874-6597
mohan@comp.nus.edu.sg

K.R.Ramakrishnan
Department of Electrical Engineering
Indian Institute of Science, Bangalore
+91-80-3092441
krr@ee.iisc.ernet.in

ABSTRACT

In this paper, we explore the convergence of the *caching* and *streaming* technologies for Internet multimedia. The paper describes a design for a streaming and caching architecture to be deployed on broadband networks. The basis of the work is the proposed Internet standard, Real Time Streaming Protocol (RTSP), likely to be the *de-facto* standard for web-based A/V caching and streaming, in the near future. The proxies are all managed by an 'Intelligent Agent' or 'Broker' - this has been designed as an *enhanced RTSP proxy server* that maintains the state information that is so essential in streaming of media data. In addition, all the caching algorithms run on the broker. Having an intelligent agent or broker ensures that additional 'simple' caching servers can be easily embedded into the network. However, RTSP does not have the right model for doing broker based streaming/caching architecture. The work reported here is an attempt to contribute towards that end.

Keywords

Caching, Streaming, Proxies, Broker, Layered coding, Replacement Policy, Hit Ratio, Quality Hit Ratio.

1. INTRODUCTION

High-speed local area networks (LANs) are now being widely deployed all over the world. Users on such LANs usually access the Internet (the web) through a proxy server, which also caches, or stores a copy of, popular objects on the local disk(s). The advantages of web caching have been discussed in a number of papers, these have been listed in [1]. If a media file is being retrieved, the download delay can be minimized by means of the *streaming* paradigm, in which *a media file is played out while it is being received over the network*. For an introduction to streaming , refer to [1].

We envisage the future of the World-Wide-Web as one involving a large number of streaming transfers of A/V content. Most such transfers would take place with the streaming data passing through one or more of the transparent proxy servers caching the streaming data as it passes through. Effective Web caching techniques will be critical for the successful deployment of streaming multimedia services over the World-Wide-Web. This should be obvious, because of the huge latencies involved, and the requirements of real time play out. However, existing web proxy caching systems have been designed for web pages (HTML documents). Such systems need to be modified for the retrieval of streaming A/V data.

In this paper, we describe the design of a media caching framework in which the local broadband network has a number of co-operating caching servers for A/V content, all managed by an 'Intelligent Agent' or 'Broker'. We believe that such a system will use the proposed Internet Standard, Real Time Streaming Protocol (RTSP) in place of the Hypertext Transfer Protocol (HTTP), to implement the streaming of the media data. Thus, not only the server and clients, but the caching proxy servers too, will all 'talk' RTSP. Information on RTSP is available in [2]. This paper describes a framework for the working of such a distributed caching and streaming system. Earlier web caching concepts and algorithms have been modified for the proposed framework. Several novel issues have been identified, and new approaches to tackle them have been proposed.

2. PROPOSED ARCHITECTURE[1]

The architecture consists of a high-speed local network, which contains a number of caching proxy servers. One of these functions as the broker or central controlling agent and the others act as sibling caching proxies. The broker is an *enhanced RTSP proxy server*, and performs the standard caching functions as well as handling the streaming issues. All client requests are transparently routed through the broker. The broker maintains the state information that is so important in all the RTSP sessions.

Having an intelligent agent or broker ensures that we can embed additional 'simple' caching servers into the network. Another new feature of the proposed architecture is that a server will be allowed to source the content into the caches ahead of any client requests. Thus, the source may not be on-line all the time but its *content* can always be assumed available and hosted in one or more of the caching proxy servers. Besides being used for the obvious purpose of load balancing, the sibling caching proxy servers are individual RTSP proxies in their own right. This enables the broker to transfer control to them efficiently whenever needed. For example, in case of a sibling 'hit', the

[1] This architecture has been earlier described in [7]. The work reported here is a collaborative effort with the author of [7], and is to be incorporated into that framework.

broker can send an RTSP REDIRECT control command to the client with a pointer to the appropriate sibling caching proxy, which has the clip. The streaming would then take place in an RTSP session from the sibling caching proxy to the client.

As a representative case of caching of multimedia objects, we consider video object caching. We also show how our system can be optimized for the caching of scalably encoded or *layered* video objects. Such objects will have a 'base' layer containing essential information, and one or more 'enhanced' layers containing higher level information. Information on layered coding is available in [3] and [10]. We use RTSP as the basis of our work here as we believe that it is likely to be the de-facto standard for web-based A/V streaming, in the future. However, RTSP does not have the right model for doing broker based streaming/caching architecture. Our work is an attempt to contribute towards that end. Additionally, our work is probably applicable to the manipulation of non-live video clips only.

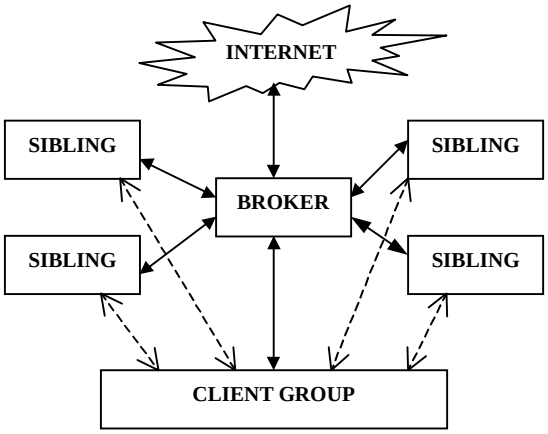


Figure 1. The proposed caching and streaming architecture. The initial interaction is always through the broker. The broker may transfer control to the siblings. The dashed lines indicate that a regular RTSP session can exist between a sibling and a member of the client group.

3. PROPOSALS AND DISCUSSION

3.1 The Auxiliary Cache

As described by the architecture in section 2, the media caching system will be distributed over the broker and it's siblings. The main cache will be a single logical entity physically spread over the disks of the broker and it's siblings. The broker will need to carry out the indexing and retrieval of the information distributed across the proxy servers in an efficient manner. To aid the broker in this task, we could use a data structure containing only the *information* related to the cached objects. We call this the 'auxiliary cache', and it is located on the broker (in memory). The idea is adapted from [6]. The auxiliary cache aids the broker in indexing the entries, and allows it to locate the required A/V clip easily. Its use ensures that the broker needs to access the disk caches only for the actual data writing tasks.

3.2 Replacement Policies

We use the term 'weight' of a clip to describe its relative importance as compared to the other clips. Thus, the higher the weight, the lower is the probability of the clip being replaced. In this work, we compare several existing replacement policies. The standard replacement policies used in caching are the Least Recently Used (LRU) and the Least Frequently Used (LFU) policies. In LRU, the weight is inversely proportional to the time since the last access of the object. In LFU, the weight is directly proportional to the number of times the object is accessed. We also use a policy based on the size of the objects (SIZE), in which the weight is proportional to the size of the clip raised to some exponent. The value of the exponent determines whether the policy is biased towards smaller or larger objects, as explained in the last portion of this section. In addition, we have used a policy called LFU_admin, which is the LFU policy combined with an admission control policy. [Section 3.4] Finally, we experiment with a hybrid policy (HYBRID) proposed in [9], and compare the results with the standard policies. In the hybrid policy, the weight *w* is computed as

$$w = F^f S^s R^r$$

where *F* is the number of times the clip is accessed (frequency), *S* is the size of the clip (size) and *R* is the time since the last access for the clip (recency). The three exponents f, r, and s are chosen by trial and error. The value for f should be a positive number, meaning that more frequently accessed objects are more likely to be cached. The value of r should be a negative number, such that more recent objects (i.e. those with a smaller value of R) are more likely to be cached. The value of s can be either positive or negative. A positive value would favor caching large objects over small ones. If recency is determined to be more important than frequency, the absolute value of the exponent r should be greater than that of the exponent f.

3.3 New Replacement Policies

The framework proposed takes advantage of clips that are scalably encoded into layers to provide a better Quality-of-Service (QOS) to the clients. The typical number of layers could be between five and ten. Layers higher up (upper enhancement layers) in general, will contain less important information as compared to the lower layers, including the base layer. For the replacement policy, we propose a new *layered* approach: The system caches clip layers as separate objects. These are sorted according to their weights, as computed below:

$$w_l = w(\frac{1}{n})$$

where w_l is the weight of the layer, *w* is the weight of the clip, and *n* is the number of the layer (for the base layer, *n* = 1). The layers are then successively deleted in decreasing order of their weights until enough space is created for the incoming object. Thus, the highest (least significant) layer of the lowest weight object is deleted first. *This policy ensures that the caching system is very robust to transients in the workload*, because a

clip will be fully cached only if there are at least as many requests for it as there are layers. In addition, clips will be deleted from the cache gradually rather than at once. This will increase the probability of at least the base layer of clips being present on the caching system.

3.4 Admission Control

The idea of having admission control is that a clip should be cached only provided it can offset the loss of the clip(s) it replaces. This will make the caching scheme less sensitive to transients in the workload. The outcome of the admission control is either positive or negative. We use the algorithm proposed in [6].

Let

W_i be the weight of the incoming clip
S_i be the size of the incoming clip
W_l be the weight of the least weight clip present in the cache
S_l be the size of the least weight clip

The algorithm is given below.

```
while(Wi > Wl){
  if (Si < Sl)
    return TRUE;
  else{
    Wl = Wl + next_least_weight_clip;
    Sl = Sl + sizeof(next_least_weight_clip);
  }
}
return FALSE;
```

Thus, the result is positive only if both conditions Wl < Wi and Sl > Si, are satisfied. If only the first is satisfied, we find the next lowest weight clip on that caching proxy, and increment Sl by the size of the next least_weight_clip. Then we check the condition above again. We exit with a negative result as soon as Wl < Wi becomes false.

In our experiments, we combine this admission control policy with the LFU policy, the resultant policy is denoted as *LFU_adm.* The results as compared to the other policies are shown in section 5.1.

3.5 A novel Performance Metric

The system uses the standard performance metrics of 'Hit Ratio' and 'Byte Hit Ratio', defined in several papers [1]. In addition, we now propose a new performance metric, which we call the *Quality hit Ratio*.

3.5.1 Quality Hit Ratio

The proposed proxy caching system is designed to handle layered video clips in addition to standard non-layered ones. Therefore, we cannot do with performance metrics as simple as Hit Ratio and Byte Hit Ratio, because they do not reflect on the *quality* of the video clip that is being returned to the client. In general, the more the number of layers of a clip present on the cache, the better is the quality. Whenever a request for an object is received, the broker converts it to a request for the necessary layer(s) and forwards the request to the source. Whenever a new object (layer) comes into the cache, older objects must be purged from it. We have proposed in section 3.3 that the purging of unnecessary objects be done on a layer by layer basis. Thus, at equilibrium, the system is made up of a number of stored objects, which may or may not have all of their layers present. A cache 'hit' for such a system would occur whenever at least one layer of the requested object is present on the cache.

The system performance will then be reflected not only by a high Hit Ratio; but also by the *quality* of the objects being returned to the clients; that is the quality of the hits. A *high quality hit* implies that almost all layers of that object are present, because a larger number of layers ensure better quality for users. Thus, we state that:

"Not only should any such system maximize the probability of a Hit, but it should also simultaneously maximize the probability of the Hit being of as high quality as possible".

We therefore introduce the twin concepts of *quality hit* and *quality hit ratio.* We define the *quality hit* as a number between 0 and 1 indicating how many out of the total number of layers of that object, are present on the cache. A quality hit of 1 implies that all layers of that object are present. Correspondingly, a quality hit of 0 implies a cache 'miss'. (That is, not even the base layer is present).

Ensuring that at least the base layers of the most popular objects are stored on the system maximizes the hit ratio. Now, if as many layers of the cached objects as possible are stored, the quality of these hits is maximized. Finally, we define the *quality hit ratio* as

$$Q_h = \frac{\sum_{i=1}^{H} \frac{n_i}{L}}{H}$$

where H is the total number of hits, L is the total number of layers for each object, and n_i is the number of layers present for object i, $n_i \leq L$.

3.6 Simultaneous Request Policy

Standard caching proxies deal with small[2] objects. For any object, the session between the proxy and the client lasts for a very short time. Thus, staggered requests for the same object can be easily dealt with. By staggered requests we mean that a new client requests an object while that is being served to another client from the source. In this case, the proxy can either ask the client to wait until the present object is cached, or start a separate session with the source for the new client.

However, the above approach will be highly inefficient when the proxy is streaming A/V content to the client. This is because each session in this case will last for a much longer time, and the cost associated with setting up a new connection to the source

[2] 'Standard' web objects include html files, small inline images, etc, which have sizes of the order of a few KB.

would be quite high. We propose a new approach, which we call the *simultaneous request policy*, for the case of clients requesting the same clip at staggered intervals. In particular, the proposed scheme takes advantage of the bandwidth mismatch between the local network and the external link to give each successive client (after the first) a slightly better Quality of Service. For popular A/V clips, if the frequency of access is high, the probability of having a simultaneous request increases. Some data on how frequently popular audio clips were accessed from a streaming audio server on the IISc campus is available in table 1. (section 4.2)

The theoretical analysis from this point onwards uses some common notations. These notations are described below.

- ***Csb*** Maximum bit rate / channel capacity possible between the source and broker.
- ***Bsb*** Bandwidth available between Source and Broker for the present session (possibly) reserved through RSVP.
- ***Cbc*** Maximum bit rate available between broker and client
- ***Bbc*** Bandwidth being used to transfer the video clips from the broker to the client on the local network.

Henceforth, we consider the case of the multimedia objects being video clips. The analysis can be easily extended to audio clips too. The notations used with respect to video clips are:

FILE_SIZE	Size of the video clip.
TOTAL_TIME	Total time taken to download the whole video.
T	Time delay between 1st request and 2nd
AV_VIDEO	Amount of video data available (cached) at the broker after time T.
Dsb	network Delay on the source-broker link

The bandwidth mismatch is such that the bandwidth between the broker and the client B_{bc} could be greater than or less than the bandwidth between the broker and the source, B_{sb}. Thus
$B_{bc} = k(B_{sb})$,
where ***k*** is any positive real number. For example, for campus networks and corporate intranets, ***k > 1*** as the local network is much faster than the external link[3].

Let the first client be denoted by 'A' and the second by 'B'. Client A sends a request to the broker for a clip. The result is a cache 'miss'. The broker retrieves the clip from the source and streams it to the client. After some time T that is less than the play-out time of the clip, client B requests the broker for the same clip. This is what we call a simultaneous request.

We provide an analysis for the following three cases:
$k > 1, \quad k \approx 1, \quad k < 1.$

Case One: k > 1

This situation arises in cases like IISc, where the local network is a high-speed one (In this case, a campus-wide FDDI) which can support streaming multimedia applications. However, the external link is much slower.

Whenever a client request at the Broker results in a 'miss', the broker retrieves the video from the source (Over the slow external link). It streams the clip being retrieved to the client, simultaneously copying the data as it passes through. Now, since the speed of the external Source-Broker link is limited to Bsb, the bit rate at which the broker provides the video to the client is also equal to ***Bsb***[4].

That is, ***Bbc = Bsb***. Thus, the frame-rate available to the client A is that which can be supported by the external (slow) link, ***Bsb***, in spite of the fact that ***Bbc*** can support a much higher frame-rate.

However, the actual capacity ***Cbc*** of the broker-client link will be much higher than ***Bsb***, and can therefore support a much higher frame rate. The outline below illustrates how this can be effectively utilized.

Ordinarily, the broker would stream the video to client B at the same rate as A, i.e. ***Bbc = Bsb***. However, as shown below, provided ***Cbc*** is sufficiently greater than ***Bsb***, we need not be constrained by the above limiting rate ***Bsb***. In fact, if ***T***, the delay between the second request and the first is large, then we can pump data to the client B at a much higher bitrate than ***Bsb***. This is because a large portion of the video will have already been cached at the broker in that time period. ***Cbc >> Bsb*** is invariably true because the local network will usually be much faster than the external one.

That is, we can now have ***Bbc= k*Bsb***, where ***k > 1***. This higher bit rate from broker to client could be used in the following ways:

- Quality improvement - The broker could retrieve, from the source, any missing layer(s) of the current clip. It could then combine the layers and stream the resulting sum_clip off to the client B, resulting in B getting a better quality.
- Addition of media - The broker could add a media stream to the presentation currently being streamed to client B.
- Interpolation - The broker could carry out video interpolation to increase the frame rate. We could then take advantage of the higher bit-rate available on the broker - client link and provide a higher frame rate to client B. Whether schemes for interpolation are feasible (in terms of computational burden and improvement in quality) remains to be seen. This is a matter of research and a further discussion is beyond the scope of our work.

An expression for k has been derived below.

The value of k will depend on the time delay T between the first (client A) request and the second (client B); and correspondingly

[3] The external link is shared by all the campus users; therefore, each user usually gets an average bandwidth (in IISc) of about 20 or 30 kbits/s.

[4] The fastest speed on a computer network is equal to the speed of the slowest link

the amount of the video that has been cached at the broker. Of course, once the video is fully cached at the broker, we will not have the external Bsb constraint.

In time T, the amount of the video that has been cached at the broker is given by

$$AVAIL\_VIDEO = B_{sb} * T$$

In the time, it takes to send AVAIL_VIDEO amount of data over the broker-client link; the additional data cached on the broker is given by

$$AVAIL\_VIDEO1 = \frac{AVAIL\_VIDEO}{k} \quad \textbf{(1)}$$

Similarly, while AVAIL_VIDEO1 amount of data is sent over the broker-client link, the amount of data cached on the broker is given by

$$AVAIL\_VIDEO2 = \frac{AVAIL\_VIDEO}{k^2}$$

Carrying on in this manner, we can obtain the expression

$$AVAIL\_VIDEO(1 + \frac{1}{k} + \frac{1}{k^2} + \frac{1}{k^3} + ...) = FILE\_SIZE \quad \textbf{(2)}$$

Thus,

$$\frac{1}{(1 - \frac{1}{k})} = \frac{FILE\_SIZE}{AVAIL\_VIDEO}$$

Simplifying,

$$k = \frac{FILE\_SIZE}{(FILE\_SIZE - AVAIL\_VIDEO)} \quad \textbf{(3)}$$

In practice, *k* must be less than the RHS above, because

- The sum above is an infinite sum
- The precise subsequent amounts of video data required after sending AVAIL_VIDEO over the broker-client link may not be available.

Hence, we conclude that

$$k < \frac{FILE\_SIZE}{(FILE\_SIZE - AVAIL\_VIDEO)} \quad \textbf{(4)}$$

A simple example will illustrate the above: Suppose *FILE_SIZE* = 15 MB and, at time *T AVAIL_VIDEO* = 8 MB.

From equation **4** above, $k < 2.1428571$. Suppose we take $k = 2$, i.e. ***Bbc=2*Bsb***.

In this case, after time *T*, 8 MB of the video is available at the broker cache. This is streamed to the client at a rate ***Bbc=2*Bsb***. In this time, a further 4 MB of the video arrives over the source-broker link. Thus the 'chunks' of video that are cached at the broker and streamed to the client B are of sizes 8, 4, 2, 1 respectively. Here, substituting in equation 2 will give us
8(1 + 1/2 +1/4 + 1/8) = 15.

But, if we had taken $k = 2.143$, then the chunks of video that would have to be cached at the broker would be of sizes 8, 3.733, 1.742, 0.813, 0.379, 0.177, 0.083, and so on. Substituting in 1 would give us
8(1 + 1/2.143 + 1/4.592 +....) = 15.

In general, using ***k = 2*** instead of ***k = 2.143*** would allow for necessary 'breathing space' at the broker. That is, even with some network delay ***Dsb*** on the source-broker link, the available video data cached at the broker is always at least the amount required for the broker to stream to the client.

If the clips being retrieved are not in layered form, then we propose that one of the following two options may be used:

- The broker could, on receipt of a request, *generate* (either on it's own or by invoking another application) a low-bit-rate version of the clip and send it to the broker.
- Real-time retrieval will not be possible for the first client (maybe the first few clients), and the broker will retrieve the clip in the usual HTTP manner. However, once the clip is cached at the broker's system, the clip could be 'taken over' by the broker and could be streamed in real-time to the clients, on the high-speed local network.

Case Two: k < 1

In this case, the external link is faster. This situation will occur when the source and broker are on the same high-speed backbone network. However, a client (e.g. a mobile client connecting through a wireless link) may have a much lower bit rate available.

In this case, the broker can retrieve all the layers from the source, but send only the base layer to the client. In fact, even if the original clip is not in layered form, the broker can actually generate a 'base layer' on the fly - by splitting the input bit stream[5]. For example, the input bit stream could be an MPEG clip (at 1.5 mbits/s). However, the broker to client link may be able to support a much lower bit-rate (say, 512 kbits/s). In this case, the broker can separate the MPEG clip into three new streams: one containing only the I-frames, another containing the P-frames, and the third containing the B-frames. The 'I stream' could then be sent to the client. If necessary, the I-frames could also be subsampled in time. We analyzed fifteen MPEG clips, which were split up into their constituent I, P and B frames using the utility in [13]. The I-frames roughly contributed to about 20% - 30% of the total size of the file. There were usually 2-3 I frames displayed per second. Simple calculations show that the bit-rate required to send only the I-frames is reduced by a factor of 4 to 5. The splitting is not computationally expensive, as there is no transcoding involved, only parsing and restuffing of the bitstream.

If the clips being retrieved are not in layered form, then we propose that the broker could 'generate' (either on it's own or by invoking another application) a low-bit-rate version of the clip being requested. This could then be sent it to the client.

[5] Splitter utilities are available [13].

3.7 Analysis for the Case of More than One Broker[6]

Thus far, we have considered a single-broker-based architecture. However, it is possible that we will have more than one broker in the local network. That is, a few of the other caching proxy servers also act as brokers. We consider here the case of autonomous, co-operating brokers.

Suppose we have three caching proxies serving a client user group as shown in figure 2.

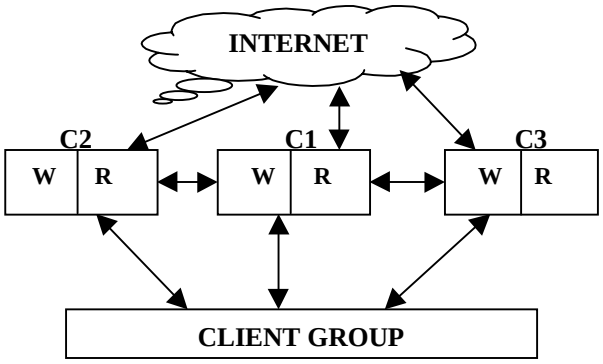


Figure 2. The case of autonomous, co-operating brokers

We consider the case where we have three brokers, each running on one caching proxy. In addition, each broker can have the other two as its siblings. We group the clients together into the 'client group', from where the user requests originate. The clients can contact any of the three brokers for a clip.

Each broker's cache is first divided into two sections, called R and W, of sizes Sr and Sw respectively. So long as the condition *Sr + Sw <= CACHE_SIZE* is satisfied, the exact values of Sr and Sw are not important. In fact, as the system, is 'tuned' for optimum hit ratio and Quality hit ratio, the sizes of the two sections will vary dynamically.

We assume that the local network is a high-speed one, such that the communication overhead between the caching proxies is not significant.

Initially, the broker caches of C1, C2 and C3 are all empty.

Whenever any of the three brokers receives a request, it first checks its local cache, and then queries the other two for the clip. If the clip is found, and it is coherent, we have a 'system cache hit'. Else, we have a 'system cache miss'. The clip is then retrieved from the source, and temporarily cached on the local disk. In case of a 'system cache hit', the clip is streamed back to the client if it is a 'local hit', that is it is on the local disk. Else, it is a 'sibling hit' and a REDIRECT response is sent to the client, with the address of the neighboring caching proxy.

Next, the broker must decide on which Caching Proxy the clip must be stored. Because of our initial assumption, the clips could be uniformly distributed over the three Caching Proxies. Thus, the broker now selects one of C1, C2, or C3 with equal probability. The clip is then cached in the *random* or ***R*** section of the selected Caching Proxy.

Since some amount of replication may be allowed (this will depend on the extent to which our initial assumption is true); the broker also checks to see if the present clip's weight exceeds a certain threshold. If it does, then the clip is also cached locally, in the *weight* or ***W*** section, according to the following rule. If the clip is in the local *R* section, it is *moved* to the *W* section (we don't need two copies of the same clip on the same local cache). If it is in a remote *R* section, then a *copy* of the clip is stored in the local *W* section.

The other two brokers will also receive and process client requests in a similar manner. The *R* section of each broker's cache is filled up with clips coming from any of the three brokers. The *W* section is filled only by the local broker.

At any time, the following will be true:

1. The *R* sections of the three brokers do not have any clips in common.
2. The *R* and *W* sections of the same (local) broker cache do not have any clips in common.
3. The *R* and *W*, and *W* and *W* sections of different brokers may have some clips in common. The number of such clips, and hence the sizes of the two sections, will vary depending on the value of the weight threshold.

The analysis could easily be extended to more than three brokers. The inter-broker communication could be implemented using the Internet Cache Protocol, ICP.

3.8 Multicast Issues

Our caching and streaming problem offers a few possibilities for Multicast to be used. First, on a 'busy' broker, the broker could perhaps wait for a short while to accumulate a few identical requests before sending out the video - this can be done through multicast. This would lead to a 'near media-on-demand' type of application.

The other example is the updating of the layers of a video clip by the servers to broker. The servers could be multicasting their content on a fixed intervals without having to wait for an update request from a broker. This way 'missing' content can be cached by the broker automatically, by subscribing to the corresponding multicast group.

[6] This section has been included only for the sake of completeness, and is speculative. The motive is to highlight a possible direction for future work.

3.9 Stream Control Issues

In media-on-demand type of applications, it is necessary to design efficient mechanisms to implement stream controls ('Trick' Modes) such as Fast-Forward, Rewind, etc. In the unicast case, the server will be serving some number of simultaneous streams, and so can provide stream control.

In multicast situations, stream control has to be implemented at the clients, for obvious reasons: the server will be sending only one stream to the group address, from where copies will be sent to individual subscribers to that group. Hence, the chance of *FF / REW / PAUSE* requests clashing is high (in case stream control is implemented at the server).

However, in this situation of the broker being an intermediate proxy cache, stream control need not be implemented at the server nor at the client - the broker can do the job. The present RTSP draft does not describe mechanisms for implementing stream control, and this remains a topic for future work.

4. IMPLEMENTATION DETAILS

The caching algorithms have been simulated using C. The simulation is driven by a 'trace file', a log file of an actual proxy cache showing the access pattern of requests for some period. What we really need is a trace file showing the RTSP requests, however such a trace file is not yet available anywhere. Instead, we use trace files of HTTP requests taken from two proxy servers in IISc. A justification for using these traces is given in the next section. The client arrival distribution is modeled using a Poisson arrival process with an inter-arrival time of $1/\lambda$. The sizes of the clips are taken from the data in the http trace file, scaled up by some factor N.

All the experiments are performed in some 'window size.' This is essentially how far back into the trace files to look. For a particular cache size, after the cache is full, the algorithms will stabilize (in terms of hit ratio or any other performance metric) after some time. For the cache sizes and clip sizes that we have used in our experiments, we have found that a window size of 30,000 to 40,000 requests is enough to study the cache behavior. We carried out experiments with several different trace files. For the purpose of discussion, we consider here only one of those trace files. The results observed were consistent for all the trace files.

4.1 Justification of the traces used

Our simulation is a trace-driven simulation. That is, instead of assuming some user access pattern and generating the input requests artificially, we use a trace file of previous accesses to the cache. This is the standard practice in web caching experiments. For media clips, trace files for proxy servers serving only streaming media clips are not yet available anywhere[7]. Given this situation, we use HTTP traces as the inputs to our simulation. We believe that to some extent, the HTTP traces will reflect user access patterns for media clips too[8]. A justification for this is presented below.

The broker receives requests for A/V clips from a number of clients. For our problem, these A/V clips will invariably be embedded objects in web pages. Even if they are independent presentations, the user will invariably be 'led' to them through a link from some web page. The justifications are:

- RTSP has been designed for the web and is a proposed *Internet* standard.
- It is likely that web pages of the near future might have tags, which point to an RTSP URL instead of an HTTP URL.
- RTSP will not replace HTTP totally. In fact, they will *coexist*. HTTP servers will continue to serve web pages as always. If the web page has a link to a media resource, the client request for that resource is transferred by the HTTP server to the RTSP (media) server. From that point onwards, the client interacts with the RTSP server.

We therefore believe that an overwhelming majority of requests to RTSP servers will be a result of 'transfer of control' from HTTP servers, and a negligible number of requests will be direct requests. Thus, there will be a 'mapping' of HTTP requests to RTSP requests. Thus, if a web page is very popular, then it is quite likely that a clip(s) to which there is a link on that page is also very popular. This is valid considering the three points mentioned above.

As far as bandwidth constraints are concerned, we have mentioned earlier that the caching and streaming framework is to be implemented on a broadband network. Efficient low bit rate coding schemes can produce A/V clips at data rates of about 16 kbits/s for audio and 30 – 35 kbits/s for video. (These figures are for the case of files encoded in the realaudio or realvideo format, and have been mentioned as an example case). Such data rates can be easily supported by most high speed LANs, for example a 100 mbits/s fast ethernet LAN. In addition, in the architecture considered, it is definitely possible to have an external link of 2 mbits/s i.e. the broker – source link. (IISc already has one such link). In such circumstances, we conjecture that the bandwidth constraints will not skew the access patterns very far from the http ones. This is mainly because the accesses to these a/v objects will be through http objects (as explained earlier). The basic assumption is that the requests, whether for http objects or for A/V, follow an approximate zipf distribution – a few objects are requested very often, and a large number of objects are requested very few times. Video-on-demand literature such as [15] supports this view. The same is also illustrated by the graph in figure 4 that shows the popularity of audio clips on an audio server on the IISc campus.

The above means that the request *distributions* for RTSP, that is the user access patterns should be similar to those for HTTP. The differences will be in the mean object size, the standard deviation, and the minimum and maximum object size. This

[7] To the best of our knowledge

[8] For our problem

suggests that we could use *similar* user access patterns for RTSP as for HTTP, but with a different mean, standard deviation, and minimum and maximum object sizes. Generating the access patterns has the difficulty that we still do not know what will be the request distribution. In the absence of such data, it appears that using the HTTP patterns is the best option available.

An important point is that the only condition for the newly proposed caching algorithms (in section 5.3) to work well is that the accesses follow a zipf – like distribution. It is well known that HTTP trace files follow a zipf-like distribution [8]. This can also be seen from figure 3.

4.2 Analysis of the input trace file

It is important to know the nature of the trace data used for the simulation. For example, we would like to know what is the distribution of requests with respect to their size. We first plot the user access patterns as a rank v/s frequency diagram (the zipf distribution) where the most popular (frequently accessed) document has rank 1. The frequency versus rank plot for the one of the http trace files under consideration is shown below, in fig 3.

Similar plots have been obtained for other http trace files[9]. The plots confirm the general observations that a few objects are accessed extremely often, while a large number of objects are accessed relatively infrequently**.**

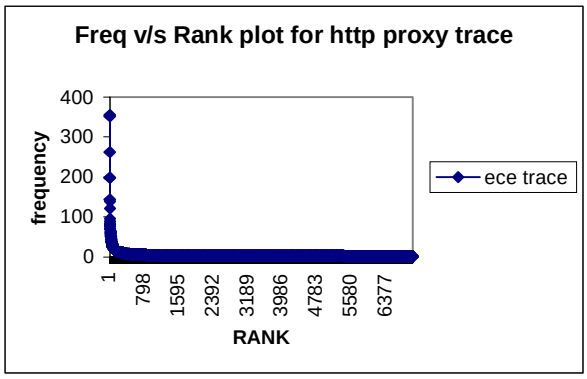


Figure 3. Frequency v/s rank plot for the http proxy trace

The graph in figure 3 shows the frequency v/s rank plot of log files obtained from a streaming audio server on the IISc campus. The similarities between these access patterns and the earlier ones are obvious. Again, we see that a few audio clips are extremely popular and frequently accessed, and a large number of audio clips are accessed less often. This plot has been included mainly to illustrate the popularity patterns for audio clips. However, it is not a trace file for a proxy server, and this is the reason that the slope of the curve is not as sharp as in the previous case.[10]

[9] These have not been included because of space constraints.

[10] The only condition for the newly proposed caching algorithms (in section 4.6) to work well is that the accesses follow a zipf – like distribution. Therefore, this trace file will give similar results as those obtained in section 4.6. . However, since we need *proxy traces*, we use the other trace files.

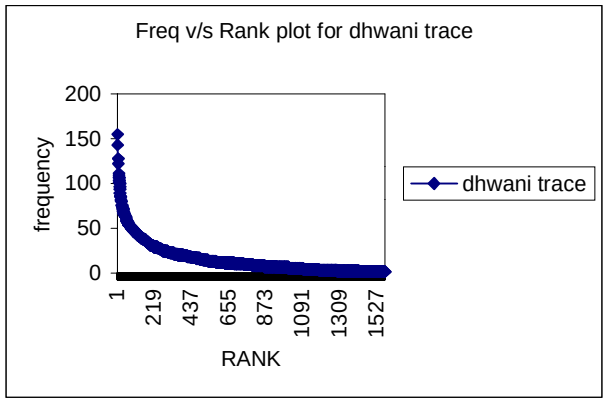


Figure 4. Frequency v/s rank plot for the audio server trace

The table below shows how frequently requests were made for the 5 most popular objects, in a period of 3-4 days. (See section 3.6)

Table 1. Popularity of Audio clips

AUDIO CLIP	NUMBER OF ACCESSES
Taal2.rm	155
Humdil3.rm	143
Taal1.rm	128
Sm_breath.rm	128
Humdil11.rm	122

The following statistics were determined for the trace file under consideration.

Table 1. Statistics for the trace

TRACE	HTTP PROXY TRACE
NO OF REQUESTS	40000
MEAN OBJ SIZE (BYTES)	7567
STANDARD DEVIATION	62854
MIN OBJ SIZE (BYTES)	17
MAX OBJ SIZE (BYTES)	10,393,212

Since size is one of the parameters of the modified caching policies, we also plot the size distribution of the requests for the HTTP traces – this is a plot of the popularity of objects v/s their size. The plot for the trace file under consideration is shown below.

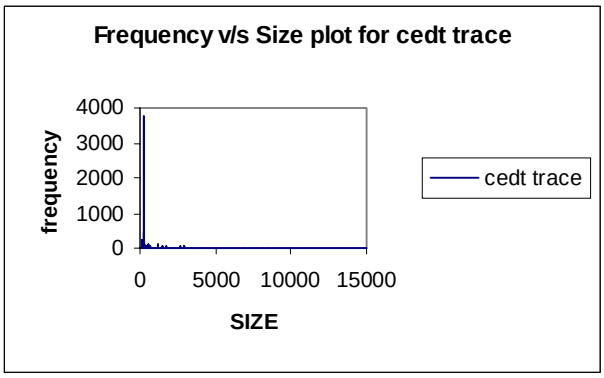


Figure 5. Frequency v/s size plot for the http proxy trace

Interestingly, the user access patterns indicate that smaller objects are more frequently accessed. This confirms the observation made in [14]. This size dependence of requests is a feature of http objects, and this is the reason why a simple replacement policy based on SIZE works well. However, such a policy may not work for media clips, as the request distribution with respect to size may not be the same as for http objects.

5. RESULTS

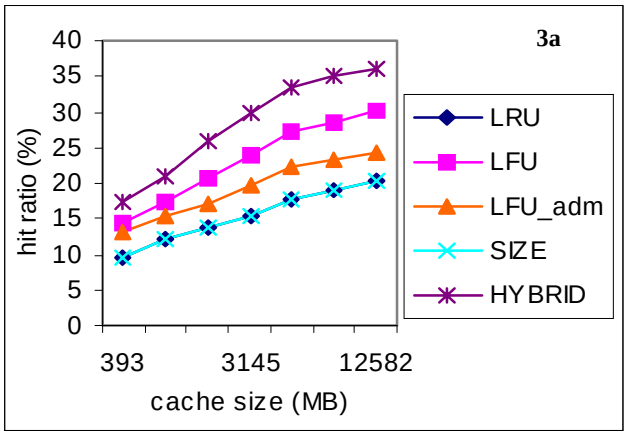
5.1 Modified Replacement Policies

We compare replacement policies on the basis of their hit ratio and byte hit ratio. In all cases except LFU_adm, the incoming object is always cached. This is to take into account the recency factor, as the incoming object is the most recent.

It is interesting to note that the LFU policy outperforms the LRU policy in all the tests, suggesting that for proxy caching, frequency is of greater significance than recency. Also, the importance of the clip size cannot be discounted. Possibly, the best algorithm would be one that would take into account all three factors. The HYBRID policy serves this purpose. This policy outperforms the others as shown in the plots in figure 6.

Our experiments show that the optimum values of the parameters of the HYBRID policy are r = 0, f = 2 and s = -1.5. Thus, the r exponent must be set to 0 for best results. We thus conclude that frequency and size are major factors that determine which objects should be purged from the cache. The incoming object is always cached, and it is here that the temporal locality or recency of requests comes into play. Thus, the object that is admitted into the cache is the incoming one (since there is a high probability that it will be requested again soon). However, the object(s) that is (are) purged from the cache are those with the least weight as determined by the respective replacement policy. Another observation is that the byte hit ratio is usually smaller than the standard hit ratio. We speculate that this is so because the most profitably cached objects are small ones[11]. We also see that the

[11] The frequency versus size plot in figure 5 confirms this for HTTP traces. Whether this is true for RTSP traces remains to be seen.



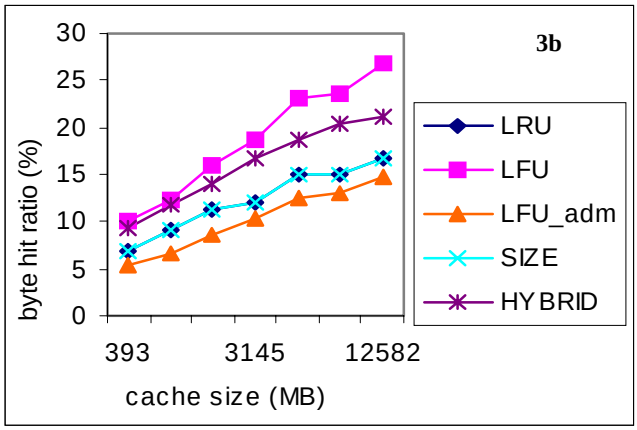


Figure 6. Performance comparison of modified replacement policies with a) hit ratio and b) byte hit ratio plotted against cache size

admission control does not improve the performance. The possible reason is this: the incoming object is the most recent object. If the admission control result is negative, this object is prevented from coming into the cache. Thus, if the traces exhibit the property of recency, then the system performance could deteriorate. Since this is the case in our experiments, it is likely that the traces have some degree of recency.

An important observation from the results is if hit ratio is the performance metric to be maximized, then the HYBRID policy is the best, as shown in the first plot. However, as can be seen from the second plot, the byte-hit ratio is maximized for the LFU policy.

5.2 Layered Replacement Policies

We carry out experiments to determine the *Quality Hit Ratio* (Section 3.5.1) for three different approaches as shown below: Object-In-Object-Out (OIOO), Object-In-Layer-Out (OILO), and Layer-In-Layer-Out (LILO).

Table 2. Layered Replacement Policies

Policy	Incoming Object	Outgoing Object
OIOO	FULL CLIP	FULL CLIP
OILO	FULL CLIP	LAYER
LILO	LAYER	LAYER

The three policies above are combined with the earlier ones to give new policies. The OIOO policy is equivalent to the observations and results in the earlier section.

In general, whenever the broker in our layered caching system receives a request for a video clip, either a full or partial HIT (or a full or partial MISS) may occur. Here, a 'full' hit implies that all layers are present. In case of a partial HIT, the broker translates the request into one for the 'missing' layer(s) and forwards it up to the source. Now, in a non-layered caching system, the objects stored are complete clips. However, in our type of a layered video caching system, the objects stored by the broker are *layers* of the clips, not the clips themselves.

This point is important because, in the first case, for a given cache size, the system's maximum capacity will be to store some ***N*** objects. However, in the second case, for the *same* cache size, the system's maximum capacity will be to store ***k*N*** objects, where *k* is the number of layers for each object. Thus, we would expect that for a layered video caching system, the hit ratio for a given cache size would be higher, but the quality hit ratio would be lower. (A 'non–layered' caching system would have a quality-hit ratio of 1).

The implications of the new and simple concept of 'Quality hit ratio' that we have proposed, are worthy of notice. Assuming that clips are available in layered form, a simple paradigm shift in the manner of their retrieval, storage, and removal from the caching system results in the following distinct benefits.

Firstly, retrieving one layer at each object request results in an implicit form of admission control. Initially, only one layer is retrieved, the other layers are successively retrieved provided the clip is requested often enough. The layered retrieval increases the probability of providing streaming access to the clip, since the individual layers will require a lower bit-rate as compared with the full clip. Another point to be noted is that retrieving one layer at a time instead of the whole clip results in better load balancing of the external ***B****sb* link – a larger number of requests can simultaneously handled.

Secondly, the hit ratio goes up. This is because, it is possible to tune the system to result in a Quality hit ratio of close to 100% while showing a significant improvement in hit ratio. Again, this is possible mainly because of the fact that some clips are much more likely to be accessed than other ones. It would also be possible to select a 'popularity' threshold above which clips are cached at full quality (all layers) and below which the qualitydegrades with decreasing clip popularity. (Fewer and fewer layers of the clip are stored). In our simulated system, no such 'threshold' is selected – the system dynamically purges the least significant layers of the least weight objects until enough space is created for the incoming layer.

We compare the performance of three layered-video caching schemes: Layer-In-In-Layer-Out (LILO), Object-In-Layer-Out (OILO) and Object-In-Object-Out (OIOO). This comparison is done based on three different performance metrics including the newly proposed Quality hit ratio, for 4 replacement policies, for two traces, all to try to answer the following questions: What sort of a performance in terms of Quality hit ratio can we expect from general video caching system of this sort? Which policy gives the best overall performance?

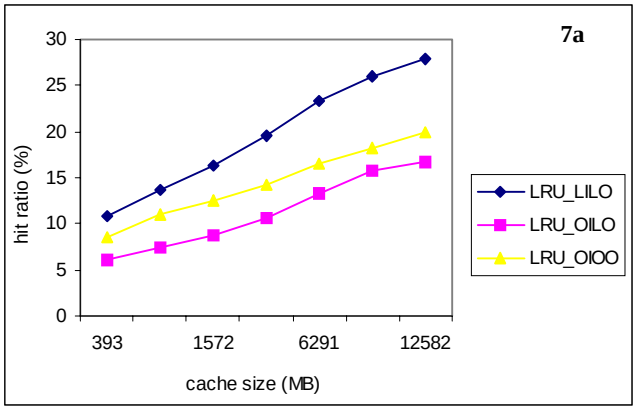
The plots in figure 7 show the results obtained with the LRU policy. We see that for the hit ratio and byte hit ratio, the LILO scheme outperforms the other two, while having a mean Quality hit ratio of 89%. Future work would involve minimizing this tradeoff between lowering of the Quality hit ratio and raising of the hit ratio and byte hit ratio.

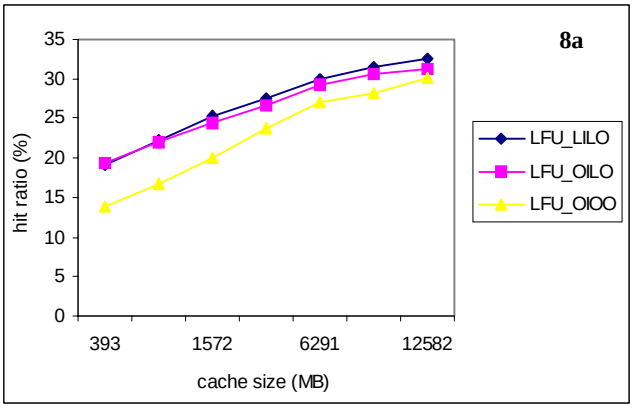
The plots in figure 8 show the results for the LFU policy. As far as the hit ratio is concerned, the two layered schemes give a much better performance than the OIOO non-layered one. A comparison of the two layered schemes indicates that the hit ratios are roughly the same for both. However, the byte hit ratio in case of LILO is much more than that in case of OILO. In addition, the Quality hit ratio is much higher in case of OILO than in case of LILO. This suggests that a choice between the two layered schemes can be made in the following way. For the LFU policy, to maximize the byte hit ratio, we must use the LILO policy whereas to maximize the Quality hit ratio, we must use the OILO policy.

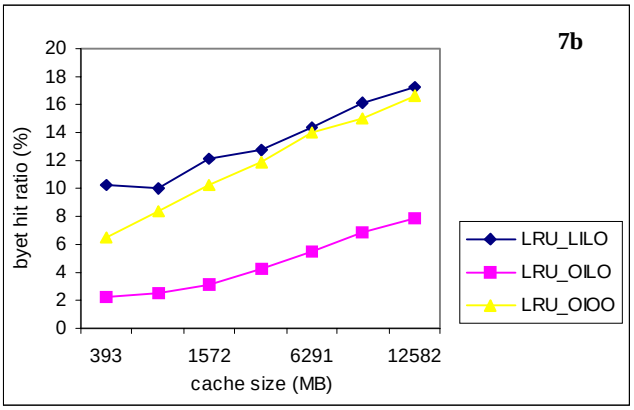
The plots in figure 9 show the results obtained for the HYBRID policy. The general behavior is similar to the previous two plots.

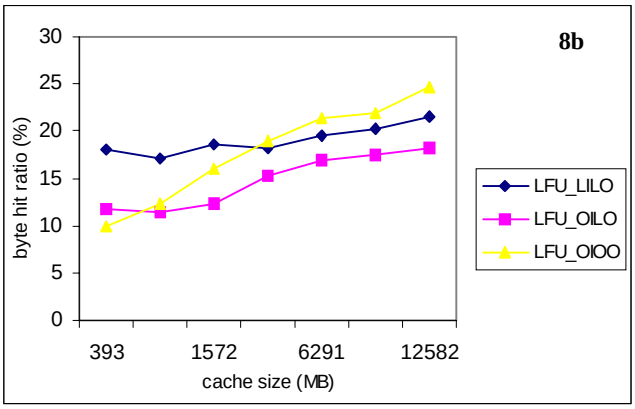
Across the three sets of plots, the highest hit ratio is 38.8%, for the HYBRID_LILO scheme. The highest byte hit ratio is 24.6% for the LFU_OIOO scheme. Between the two layered schemes, the highest byte hit ratio is 21.6% for the LFU_LILO scheme. All of the above observations are for a CACHE SIZE of 12582, the maximum in this set of observations.

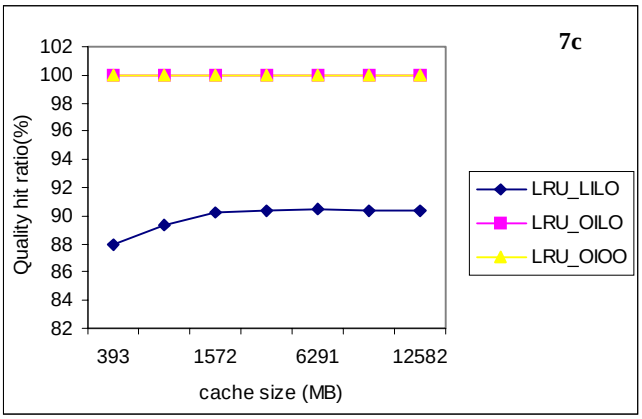
If only hit ratio was the criterion, then HYBRID_OILO appears to be the best scheme overall, with hit ratios almost the same as LILO and with a Quality hit ratio of almost 100%. However, if byte hit ratio is the criterion, then the LFU_OIOO scheme is the best.











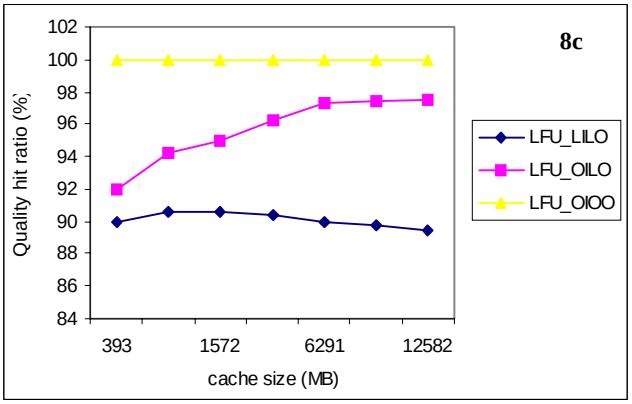


Figure 7. The plots above show the results obtained for the LRU based layered caching schemes, for the metrics hit ratio (7a), byte hit ratio (7b) and quality hit ratio (7c).

Figure 8. The plots above show the results obtained for the LFU based layered caching schemes, for the metrics hit ratio (8a), byte hit ratio (8b) and quality hit ratio (8c).

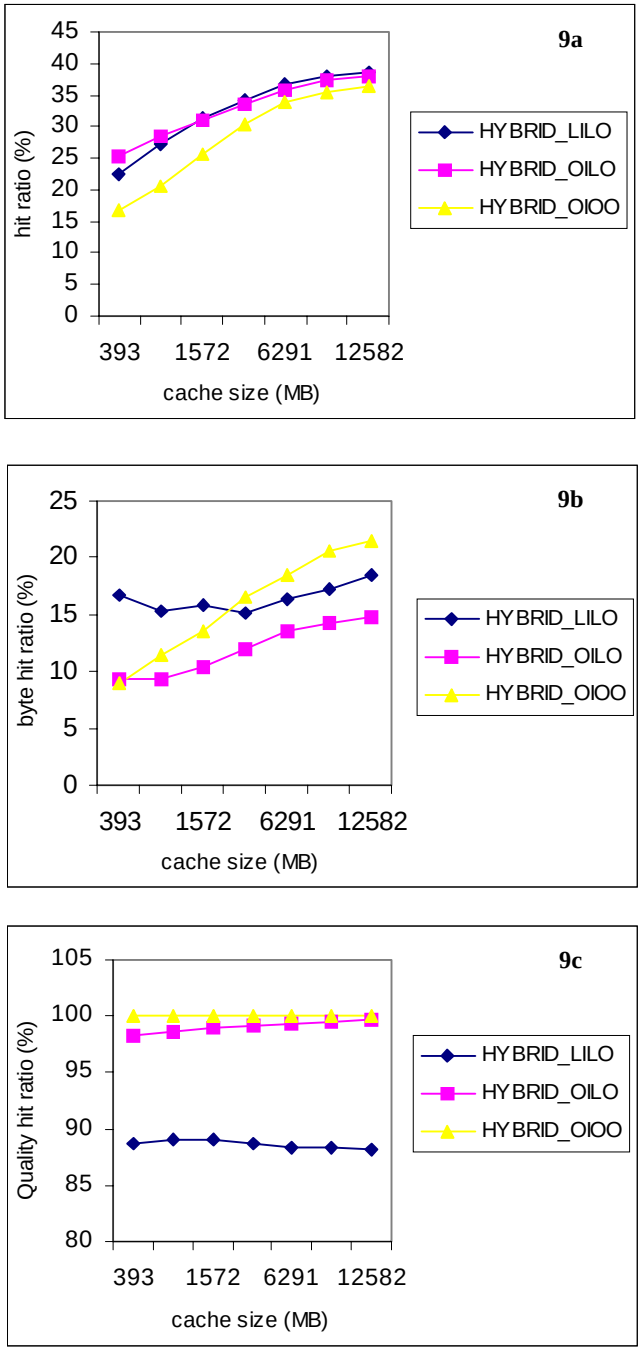


Figure 9. The plots above show the results obtained for the HYBRID based layered caching schemes, for the metrics hit ratio (9a), byte hit ratio (9b) and quality hit ratio (9c).

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a new caching and streaming framework for multimedia objects. The framework has a broker-based architecture and uses the proposed Internet Standard Real Time Streaming Protocol (RTSP). The problem and the area of research itself being a novel one, we hope that the work reported in here proves to be of great value in the near future. It is hoped that the proposed design of a broker based caching and streaming architecture for multimedia will serve as a generic framework. Among the important conclusions drawn through the experiments is that a hybrid caching policy based on frequency, size and recency usually gives the best results. We have also seen that the novel layered replacement approach proposed gives better results than standard object-based replacement schemes. The novel performance metric, Quality Hit Ratio, appears to be adequate as a measure for the evaluation of the new layered caching policies. It appears that caching and streaming are henceforth going to be the major factors influencing the successful deployment of Internet Multimedia systems. However, the convergence of these two technologies is a recent development and has thrown up a number of new challenges. Our work here addresses some of the issues. However, many remain unresolved, and thus open a large number of areas for future work.

The design of the caching and streaming architecture is already in place. The system performance has also been verified through simulation. Therefore, the next logical step in this direction is the implementation of the proposed architecture in an actual network scenario. Also, the integration of the proposed framework into a multicast scenario is extremely important, as multicast is one of the essential technologies of Internet Multimedia. A 'layered multicast' approach needs to be investigated. Stream control issues, particularly related to layered video and RTSP, must be looked into. It is important to add functionality to RTSP to implement 'trick modes' such as FF, REW, etc.

As the Quality hit ratio goes closer and closer to 1, the hit ratio and byte hit ratio drop. This tradeoff between the raising of the Quality hit ratio and lowering of the hit ratio and byte hit ratio needs to be minimized.

Since we are dealing with streaming media, a thorough investigation into QOS related issues, is necessary. This problem involves 3 party QOS negotiation with three cases possible. One, the broker simply 'relays' the requested QOS parameters upwards to the source. Two, the broker 'maps' the requested QOS parameters to a new set based on the external resources available. Three, the broker acts as the source itself (this is in case of a cache 'hit') and responds to the request.

Among other issues that require further research, is the generation of the 'sum_clip'. The broker must be able to combine the layers already present with the incoming layer(s) without introducing additional latency. The layered encoding scheme being used will have to provide for this. Secondly, an investigation into the feasibility and advantages, if any, of video

interpolation and transcoding schemes is also required. These are possible additional functions of the broker. Thirdly, a performance comparison between a single broker-based architecture and a multiple broker-based architecture is needed. Finally, if possible, a trace file of accesses to a *media server* (audio or video) should be used instead of the HTTP file. This will make the results of the trace driven simulation much more realistic.

7. REFERENCES

[1] S. Paknikar. "A Caching and Streaming Framework for Multimedia". *Masters thesis*, IISc, Bangalore, December 1999.

[2] H. Schulzrinne, et al. "Real Time Streaming Protocol (RTSP)." *Proposed Internet Standard*, RFC 2326, April 1998.

[3] S. McCanne. "Scalable Compression and Transmission of Internet Multicast Video." *Ph.D. Thesis*, University of California, Berkeley, December 1996

[4] M. Abrams, et al. "Caching Proxies: Limitations and Potentials." *Fourth International World Wide Web Conference*, Boston, 1995.

[5] S. Williams, et al. "Removal Policies in Network Caches for World Wide Web Documents." Proceedings of the ACM SIGCOMM, 1996.

[6] Aggarwal, et al. "On Caching Policies for Web Objects." *IBM research report* RC 20619 11/08/96

[7] L.H.Ngoh, et al. "A Multicast Connection Management Solution for Wired and Wireless ATM Networks", *IEEE Communications Magazine*, Vol. 35, No. 11, pp. 52-59, Nov 1997.

[8] P. Cao, et al. "Web Caching and Zipf-like Distributions: Evidence and Implications." *IEEE Infocom*, 1999.

[9] D. Wessels. "Intelligent Caching of World-Wide-Web Objects." *MS Thesis*, University of Colorado, 1995.

[10] A. Swan, S. Mccane and L. Rowe, "Layered Transmission and Caching for the Multicast Session Directory Service", Proc. Sixth ACM Multimedia Conference, Bristol, Sep 1998.

[11] Reza Rejaie, Mark Handley, Haobo Yu, Deborah Estrin, "Proxy Caching Mechanism for Multimedia Playback Streams in the Internet." 4th International WWW Caching Workshop, Mar, 1999.

[12] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled playback video over the Internet", Proc ACM SIGCOMMM, Sept. 1999.

[13] Phillip Lougher. "Mpeg Util – an Analysis and edit program for MPEG-1 video streams". http://www.comp.lancs.ac.uk/computing/users/phillip/mpegUtil.htm

[14] A. Bestavros, et al. "Application - Level Document Caching in the Internet.", Proceedings of the International workshop on Services in Distributed and Networked Environments, June 1995.

[15] S. Carter and D. long. "Improving Bandwidth Efficiency of Video-On-Demand Servers". Computer Networks 31 (1999) 111-123