# Further Developments of a Dynamic Distributed Video Proxy-Cache System


## Claudiu Cobârzan

claudiu@cs.ubbcluj.ro
"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science

## László Böszörményi

laszlo@itec.uni-klu.ac.at
University of Klagenfurt, Department of Information Technology




**Institute of Information Technology**
**University Klagenfurt**
**Technical Report No TR/ITEC/06/2.02**
**August 2006**

**Abstract**

We investigate the dynamics of a distributed video proxy-cache system that is able to adapt the number of running nodes depending on conditions like client request patterns, network load etc. Since we've already examined the **split** operation used to expand the system by adding new nodes, we explore in detail two operations (**hibernate** and **shut down**) used to reduce the number of active nodes. Also, several scenarios for object (video data) movement and replication between participants are taken into consideration. Further more, we study the system's efficiency by measuring the byte hit rate under different test scenarios.

# Contents

# 1 Introduction

With the rising amount of multimedia data transferred in the Internet, multimedia (especially video) proxy-caches tend to become common. Numerous caching strategies and proxy-caching systems for video data have been proposed in recent years: prefix caching [1], caching of a prefix and selected frames [2], caching of hotspot segments [3], popularity based prefix caching [4], segment based prefix caching [5], variable sized chunk caching [6]. Other authors considered quality based video caching [7], [8] or cooperative caching schemes [9].

Our approach while having similarities with the work in [9] is aimed at providing offensive adaptation inside a LAN, in a way that fits the description in [10].

In previous work (see [11]) we have proposed a dynamic distributed video proxy-cache system that is able to adapt the number of running nodes inside a LAN, depending on conditions like client request patterns, network load etc. We have identified two situations that should trigger the spawning of a new proxy-cache:

1. *the running cache(s) is/are under storage constraints and all the cached objects are approximatively equally useful.* Performing cache replacement in such a situation has the potential of discarding valuable objects and might not be desirable. Instead, it might be a good idea to start a new proxy-cache somewhere inside the LAN and take advantage of it's storage capacity thus delaying the moment when replacement is required.

2. *the running proxy-cache(s) is under load constraints: due to insufficient CPU/ memory/bandwidth client requests are discarded at a rate that has reached a certain threshold.* In such a situation adding a new proxy-cache to the system increases the amount of available resources. Even if the bandwidth is the limiting factor, starting a new proxy-cache in the LAN might prove beneficial if we also split the client requests between the running nodes depending on their location. By doing so we obtain a number of smaller client clusters, each cluster serviced by its own proxy-cache. If the client clusters have no common elements (the clients in one cluster are serviced exclusively by their proxy-cache), than having a new running proxy-cache doesn't decrease the available bandwidth, but in fact it virtually increases it, once the desired data is cached.

This paper presents further developments in the dynamic proxy-caching scheme introduced in [11] and extends the measurements regarding its efficiency using byte hit ratio as metrics. We've concentrated on the cache replacement strategy's influence on the byte hit rate and mainly on how various factors considered within a utility based cache replacement strategy affect it.

# 2 System Description and Enhancements

Although inside a LAN the physical resources (CPU power, memory, bandwidth) are usually abundant, having a single video proxy-cache that services all the clients might not suffice due to various constraints (CPU, memory, storage). This might happen when high quality video data has to be simultaneously streamed to multiple clients, or when heterogeneous clients that require transcoding operations have to be serviced. In those cases we propose to spawn a new video proxy-cache inside the LAN. The newly created proxy-cache might be used to immediately

hold some of the already cached objects that are extremely popular with the client population (see 2.1) or might be used to service client requests redirected from several of its siblings.

We consider that the distributed video proxy-cache system has a primary node $p$ which is defined in a static way and that each player, browser etc. launched from within the LAN knows its address. This primary node is used by default by all the clients in the LAN, as long as it can handle the request volume and it has enough resources. Once multiple nodes are active within the caching system as a result of a split operation (see [11]), the primary node $p$ will forward some of the requests it receives towards its siblings. A client that receives data from a node $k$ ($k \neq p$), meaning that his request has been redirected by node $p$ to node $k$, will add node $k$ to the pool of caching nodes it uses.

The network topology considered is similar to the examples in Figure 1 and Figure 2.

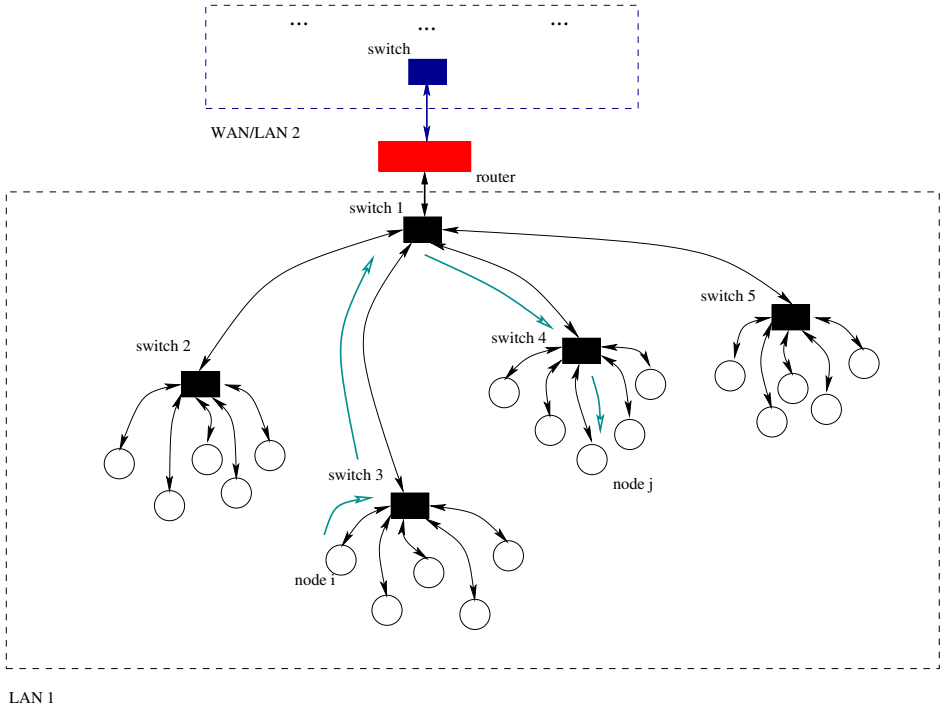



Figure 1: Network Topology (1)

In Figure 1 a number of (4) switches (labeled switch2, switch3, switch4, switch5) are interconnected using another switch (switch1) which also ensures the connection with the outside world (Internet, another LAN etc.) through a router. A request generated by *node i* for an object residing in the cache on *node j* has to travel through a number of three switches (switch3, switch1 and switch4) across four network segments (*node i* $\rightarrow$ switch3, switch3 $\rightarrow$ switch1, switch1 $\rightarrow$ switch4, switch4 $\rightarrow$ *node j*).

When compared with Figure 1, the LAN topology in Figure 2 has an additional switch (switch6). This additional switch is plugged into one of the switches (switch4) directly con-

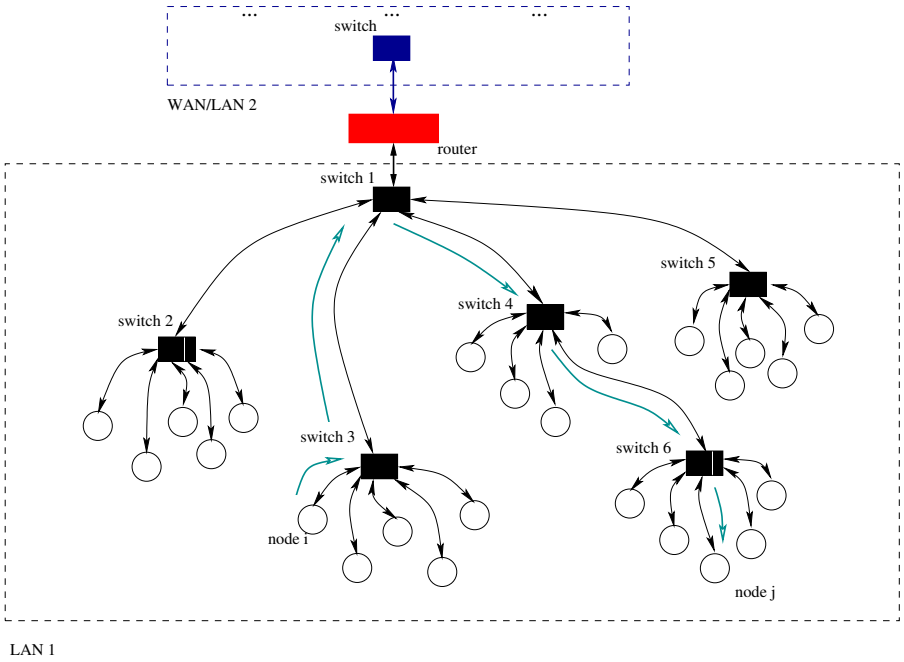



Figure 2: Network Topology (2)

nected to the "main" switch (switch1). In this case, considering that *node j* is connected to switch6, a request from *node i* for one of the objects hosted by *node j* will traverse four switches (switch3, switch1, switch4, switch6) and five network segments (*node i* → switch3, switch3 → switch1, switch1 → switch4, switch4 → switch6, switch6 → *node j*).

It can be seen that in the two cases depicted in Figure 1 and Figure 2 the requests are sharing some of the path's switches and segments.

Within the system, a cached object is defined as follows:

$$\begin{aligned} obj = &(size(obj), duration(obj), bitRate(obj), \\ &qualityValue(obj), TLA, HC, COST) \end{aligned} \tag{1}$$

We've decided to hold information on the size, duration, encoding bit rate and quality value of the video. The quality value (a real number between 0 and 1) is the measure of the object's quality (based on characteristics of the video object like resolution, color information etc.) to different clients. It is a relative value that shows the degree in which the cached object matches the desired quality of a certain class of users. A $qualityValue$ equal or close to 1 indicates that the object has exactly or almost the desired quality. A high quality video object doesn't necessarily has the $qualityValue$ close to 1. For example if almost all the users have limited display capabilities, say 640x480, a video object encoded at 1280x1024 should have a $qualityValue$ closer to 0 than to 1, because transcoding is necessary in order to deliver the object to the requesting clients.

The following notations are used:

- $N$ - the set of nodes in the LAN ($N = \{N_1, N_2, ..., N_n\}$, $n = |N|$);
- $K$ - the set of nodes acting as proxy-caches ($K \subseteq N$).

We also use three vectors to hold additional data. Each vector has the number of elements equal with the number of nodes in the LAN (the $i^{th}$ element in each vector corresponds to the $i^{th}$ node in the LAN):

- *TLA (Time of Last Access)* is a vector whose $i^{th}$ component represents the last point in time when *obj* has been requested from node $i$

  $TLA_j = \{timeLastAccess(obj_{j,k})_{Ni}\}$, where $1 \leq i \leq n$ ($n$ - the number of nodes in the LAN), $1 \leq k \leq q$ ($q$ - the number of objects cached at node $K_j$), $1 \leq j \leq i$

  (e.g. $timeLastAccess(obj_{1,3})_{N_2}$ represents the moment the $3^{rd}$ object cached at node $K_1$ has been last requested from the node $N_2$);

- *HC (Hit Count)* represents a vector which retains the number of times *obj* has been requested from within the LAN

  $HC_j = \{hitCount(obj_{j,k})_{Ni}\}$, where $1 \leq i \leq n$ ($n$ - the number of nodes in the LAN), $1 \leq k \leq q$ ($q$ - the number of objects cached at node $K_j$), $1 \leq j \leq i$

  (e.g. $hitCount(c_{1,3})_{N_2}$ represents the number of times the $3^{rd}$ object cached at node $K_1$ has been requested from node $N_2$);

- *COST* is a vector whose components represent the cost of streaming *obj* from the current node to any other local node

  $COST_j = \{cost(obj_{j,k})_{Ni}\}$, where $1 \leq i \leq n$ ($n$ - the number of nodes in the LAN), $1 \leq k \leq q$ ($q$ - the number of objects cached at node $K_j$), $1 \leq j \leq i$

  ( e.g. $cost(obj_{1,3})_{N_2}$ represents the cost of streaming the $3^{rd}$ object in the cache at node $K_1$ to the node $N_2$. It is computed as follows:

$$cost(obj_{1,3})_{N_2} = \frac{\alpha_{K_1,N_2}}{bitRate(obj_{1,3})} duration(obj_{1,3}) \tag{2}$$

  which reads "the cost of streaming the $3^{rd}$ object from the cache at node $K_1$ to the node $N_2$ equals the amount of bandwidth needed to stream that particular object" - $\alpha_{K_1,N_2}$ denotes the amount of bandwidth available between node $K_1$ and node $N_2$).

By using the three vectors ($TLA$, $HC$ and $COST$) we are able to compute the *utility value* of each cached object for any client from within the LAN with the formula in [11]:

$u : LC_j X N \rightarrow \mathbf{R}$, $LC_j$ - the content of the cache at node $K_j$, $1 \leq j \leq i$

$$\begin{aligned} u(obj, N_i) = coef_1 \times size(obj) + coef_2 \times \frac{1}{timeLastAccess(obj)_{N_i}} + \\ coef_3 \times hitCount(obj)_{N_i} + coef_4 \times qualityValue(obj) \end{aligned} \tag{3}$$

where $obj \in LC_j$, $\quad coef_1, coef_2, coef_3, coef_4 \in [0,1]$ and $coef_1 + coef_2 + coef_3 + coef_4 = 1$ (the utility of *obj* is a weighted average of the different characteristics of the cached video object).

This utility value together with the corresponding cost values of streaming the objects to all the nodes in the LAN are always considered when decisions have to be made regarding the cached objects. Such decisions include:

- replicate the object on a different node in the distributed video proxy-cache;
- move the object on a different node;
- discard the object.

## 2.1 Data Replication and Data Partitioning

The first operation, **replicate the object on a different node**, is performed only when the object is popular enough and has been frequently requested. This means that the total number of requests for the object surpasses a threshold $\zeta$ and the rate for the last $n_0$ requests has been greater than $\gamma$. The new location is chosen so that the $COST$ vector for the object at the new location is minimum (the sum of all the elements in the vector is minimum).

Data replication should also be considered after a new video proxy-cache is spawned due to load constraints, as having popular data distributed on multiple nodes would help to quickly "cool" the hotspots in the system.

The second operation, **move the object on a different node**, might be considered when the local cache is under storage constraints (and the thresholds $\zeta$ and $\gamma$ have not been reached) but the data selected to be discarded is valuable enough for some distant node(s). In this case, the new location is also chosen so that the $COST$ vector for the object at the new location is minimum.

Another possibility would be to move an object from its current location, if it is requested from only one (different) node (or a small number of nodes) and one of the thresholds $\zeta$ or $\gamma$ has been reached. Figure 3 (a) illustrates such a case:

- the clients of *node i* are requesting object *o2*, which is cached locally, and object *o4*, which is cached on *node j* (we consider that *node i*, *node* $k \in K$, $K$ the set of active proxy-caches);
- at the same time, the clients of *node j* are requesting object *o3*, which is cached locally, and object *o1*, which is cached on *node i*.

The objects *o1*, respectively *o4* are not useful in the local cache since they are not requested by the clients using node $i$, respectively node $j$ as their primary proxy. More than this, their current location generates additional traffic inside the LAN as the objects have to be repeatedly streamed towards frequently requesting clients residing on other nodes.

In such a situation, it makes sense to move object *o4* from *node j* to *node i* and object *o1* from *node i* to *node j* and thus save internal bandwidth for future requests.

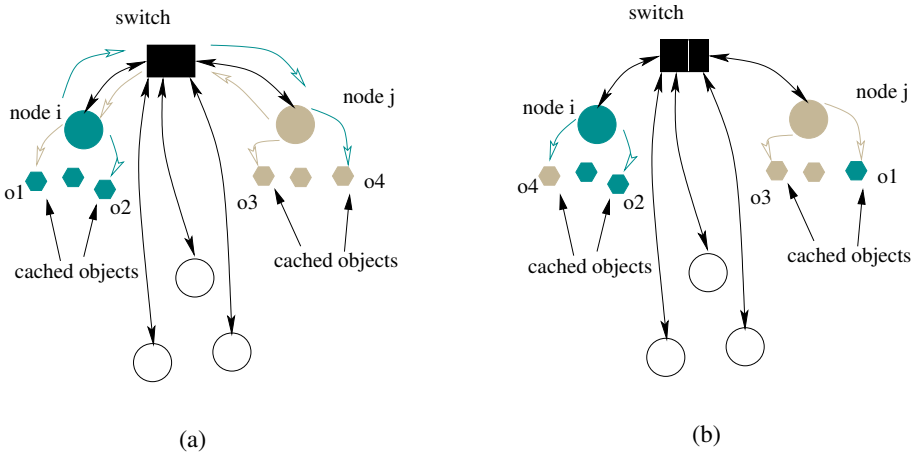



Figure 3: Object move between nodes

When deciding to move or replicate an object from one node to another, the hit rate and frequency for all the nodes that have ever requested the object in question should be considered (the data held in the $TLA$ and $HC$ vectors).

One possible criterion when selecting the node(s) to host moved or replicated data is to consider the node(s) for which the sum of the elements from the $COST$ vector is minimal (and the utility value is maximal).

As an argument let us consider another example (Figure 4). In case (a) two nodes *node i* and *node j* are frequently requesting an object (say *obj*) cached on *node k*. The clients on *node k* are also interested in the object, but on a smaller scale.

Let's compute the total cost of serving the object *obj* to *node i*, *node j* and *node k*, considering that all the network segments have the same bandwidth $\alpha$:

$$\begin{aligned} TotalCost &= cost(obj)_{node\ i} + cost(obj)_{node\ j} + cost(obj)_{node\ k} \\ &= 11\frac{\alpha_{node\ k,node\ i}}{bitRate(obj)}duration(obj) + 10\frac{\alpha_{node\ k,node\ j}}{bitRate(obj)}duration(obj) \\ &\quad + 3\frac{\alpha_{node\ k,node\ k}}{bitRate(obj)}duration(obj) \\ &= 11\frac{4\alpha}{bitRate(obj)}duration(obj) + 10\frac{4\alpha}{bitRate(obj)}duration(obj) + 0 \\ &= \frac{84\alpha}{bitRate(obj)}duration(obj) \end{aligned} \tag{4}$$

Consider now that *node k* comes under storage constraints and it has to discard some of the objects it holds. Since the object *obj* is not that popular (only 3 hits) it might be a candidate for replacement/move. The possible candidates for a move are *node i* and *node j*. If *node i* is under storage constraints too, only *node j* remains eligible to hold the data (Figure 4 (b)). Considering that the request rate remains constant in time, we can compute the total cost in terms of used internal bandwidth in a situation similar to that from Figure 4 (a):

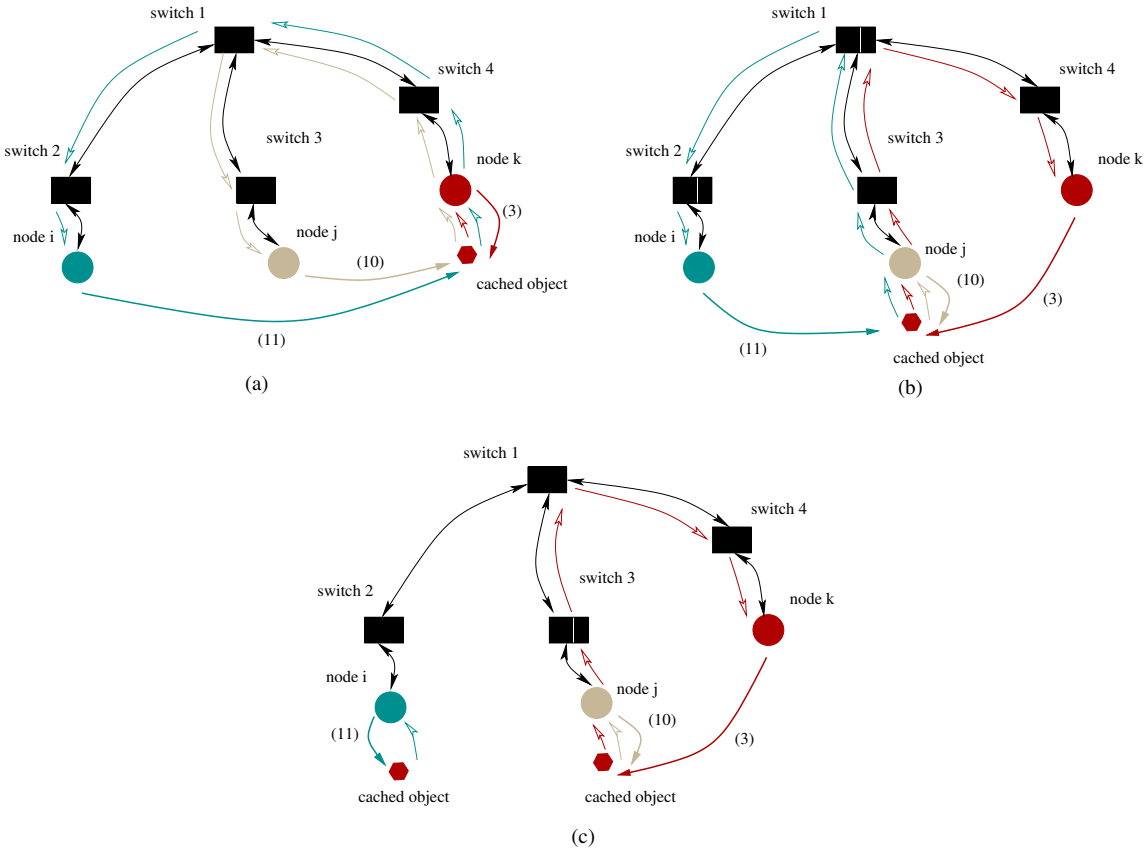



Figure 4: Object move and object replication between nodes

$$
\begin{aligned}
TotalCost &= cost(obj)_{node\ i} + cost(obj)_{node\ j} + cost(obj)_{node\ k} \\
&= 11\frac{\alpha_{node\ j,node\ i}}{bitRate(obj)}duration(obj) + 10\frac{\alpha_{node\ j,node\ j}}{bitRate(obj)}duration(obj) \\
&\quad + 3\frac{\alpha_{node\ j,node\ k}}{bitRate(obj)}duration(obj) \\
&= 11\frac{4\alpha}{bitRate(obj)}duration(obj) + 0 + 3\frac{4\alpha}{bitRate(obj)}duration(obj) \\
&= \frac{56\alpha}{bitRate(obj)}duration(obj)
\end{aligned}
\tag{5}
$$

In Figure 4 (c) we illustrate another case in which the object on *node k* is replicated on both *node i* and *node j* before being discarded on *node k*. Considering again a constant request rate, we can compute the total cost in terms of used internal bandwidth in a situation similar to those from both Figure 4 (a) and Figure 4 (b):

$$
\begin{aligned}
TotalCost &= cost(obj)_{nodei} + cost(obj)_{nodej} + cost(obj)_{nodek} \\
&= 11\frac{\alpha_{node\ i,node\ i}}{bitRate(obj)}duration(obj) + 10\frac{\alpha_{node\ j,node\ j}}{bitRate(obj)}duration(obj) \\
&\quad + 3\frac{\alpha_{node\ j,node\ k}}{bitRate(obj)}duration(obj) \\
&= 0 + 0 + 3\frac{4\alpha}{bitRate(obj)}duration(obj) \\
&= \frac{12\alpha}{bitRate(obj)}duration(obj)
\end{aligned}
\tag{6}
$$

It can be seen that there is a significant difference between case (a) and case (b) in terms of COST, under identical request rates. The difference is even greater when comparing the results in cases (a) and (b) with the result in case (c).

It is important to note that in the case of data replication, or data migration, the utility vector associated with the object being replicated or moved remains the same, and only the cost vector ($COST$) changes as it has to be recomputed considering the new object location. In order to do this, it is considered that the information on the LAN topology is available.

The third operation, **discard the object**, is performed when the local cache is under storage constraints and there is no node participating in the federate cache that can host the object (this means that before discarding an object, the system checks to see whether it can't be moved to one of the nodes in the system).

A node decides if it can host an object from one of its siblings, depending on its remaining amount of free space. For example, a number of storage thresholds could be considered. Depending on the current amount of stored data and on the thresholds, a new object could be (1) accepted unconditionally, (2) accepted if it is valuable enough or (3) not accepted at all.

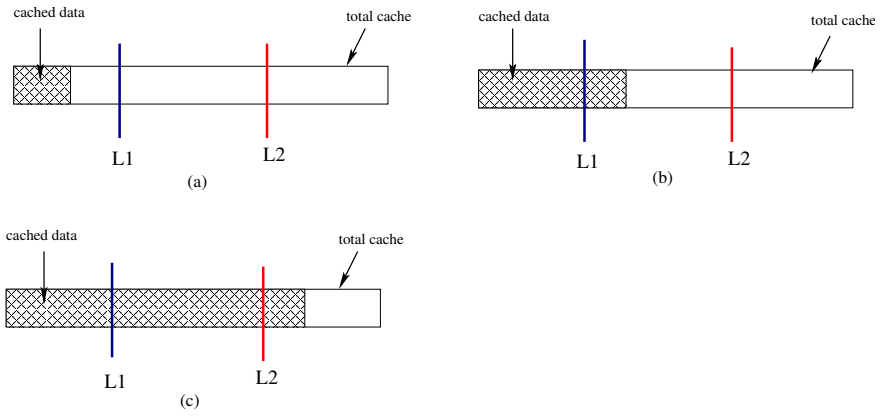



Figure 5: Defined thresholds for a local cache

Figure 5 shows a local cache with two thresholds L1 and L2:

- in the first case (a), the amount of cached data doesn't surpass L1, so if the node is

contacted to store an object from a sibling, it should accept it regardless of the object's utility to the local cache;

- if the amount of cache data is greater than L1 but smaller than L2 (case (b)), "hosting" objects should be considered only if they are useful for the local cache (e.g. the objects being moved are as useful as the local ones located between the beginning of the cache and L1);
- in the case the amount of cached data is over the second threshold L2 (case (c)), accepting new objects from siblings should be done only under very special circumstances (e.g. highly valuable object for the local cache and there are old, useless objects that can be discarded if cache replacement becomes necessary in the near future due to storage constraints).

The siblings have to agree on the criterion that takes precedence in the case of cache replacement: utility or cost. They also have to decide the relation between utility and cost: the optimal case would be when holding only low cost objects with high utility. In fact, one of the system tasks could be to make the necessary operations so that the utility value of the cached objects is maximal and the costs for delivering them to (all potential) clients is kept to a minimum. The same conditions should be considered when making cache replacement.

## 2.2 Hibernate and Shutdown Operations

In [11] we've concentrated on the **split** operation needed to increase the number of active nodes in the federate cache and only mentioned the **hibernate** and **shut down** operations. They are used to reduce the number of active nodes in the distributed video proxy-cache during periods with low activity by either suspending or stopping their activity. A node in hibernation has reduced functionality: it is not allowed to contact any origin server in order to retrieve new content, but it can serve data from its local cache. If a node is shut down, it can't provide any type of service to clients. Following we present some additional information related to these operations.

In order to use them in an efficient manner, a **ranking system** between the running nodes which make-up the federate proxy-cache should be introduced. The ranking system is designed to reward the siblings that serve the largest amount of data (when compared with the cache size) to clients inside the LAN, while also considering the time when the sibling has served its last requesting client:

$$rank(K_i) = \frac{TSD(K_i)}{CS(K_i) * (CT - LRT)}$$

where:

- $TSD(K_i)$ = the total amount of data that has been served by the proxy-cache $K_i$;
- $CS(K_i)$ = the maximal amount of data that can be stored by $K_i$;
- $CT$ = the current time;
- $LRT$ = the last moment in time a client request has been serviced ($CT > LRT$).

In this way the most active siblings should have the largest rankings. In case of equal ranks the sibling caching the largest amount of data should take precedence.

We assume that each proxy-cache knows the ranking of its siblings (e.g. the rankings are exchanged periodically when the siblings are sending digests of their cache content).

### Hibernation

A *hibernate* operation should be performed in two situations:

- the overall rank of the siblings (e.g. computed as a mean from all the rank values) is constantly decreasing over a specified time interval (e.g. 1 hour, 1 day, etc.) and a specified threshold is reached;
- the rank of any proxy-cache part of the distributed system becomes smaller than a specified threshold.

In the first case, decreasing overall rank, the proxy-cache with the smallest rank should be selected and put in hibernation (that is, it doesn't forward requests towards origin servers in order to retrieve new content but can still serve cached data).

The ranking of the siblings in hibernation should be continued, so that nodes in this state could be reactivated if needed. Reactivation might be considered when the overall rank of the siblings is constantly increasing and surpasses a superior limit and/or a split operation is required.

### Shutdown

It may happen that the rank of the siblings continues to drop in spite of putting some of them into hibernation. This could indicate that the request volume on the system is decreasing. Again, if a certain limit is reached, some of the hibernating proxies could be shut down (e.g. the ones with the smallest rank).

Shutting down a proxy-cache raises questions regarding the stored data:

- should it be transferred to a running/hibernating sibling?
- should it be discarded?

Most probably, an analysis of the stored data (the $TLA$, $HC$ and $COST$ vectors together with the utility values), should be performed in order to asses the utility of the objects and only the most useful objects should be transferred while the rest should be discarded. The amount of data that could be transferred should be limited by the amount of free space still available in the running proxy-caches for caching data from siblings, so that the caches are not filled with uninteresting data. A hibernating proxy is shut down because it has the smallest rank, so the data it holds has not been requested very often and/or for a long time.

Both hibernation and shutdown can be viewed as operations which ensure the conservation of computing resources inside the LAN: if a smaller number of proxies can handle the current load, than put some of the less used nodes in hibernation or shut them down. Shutting down a proxy-cache does not mean shutting down the computer that runs the process, but destroying the process-cache process. In this way the newly freed computer resources (CPU, memory and storage space) can be used by other processes.

### 2.3 Client Behavior Considerations

Usually when dealing with a video object a client is confronted with one of the following situations:

1. it can access the object only from its beginning towards its end (no jump are permitted);
2. random access inside the object is permitted (jumps are possible).

Following we consider only the first case, which is quite common when asking for a video from a remote source. In such a situation, the user usually has to wait for the local buffer to be filled before starting the play back. Since it is common for clients to cancel the play back of a video object after a short period of time (either because he had to wait too much, or because he doesn't like the video) we define a virtual *cancellation point* ($C_{point}$) as well as the probability that the client cancels playing back the video with regard to this virtual point.

If we denote $P^v_{cancel}(f)$ = the probability that the client cancels the play back of the video object $v$ when reaching the frame $f$, then:

$$max(P^v_{cancel}(f)) = P^v_{cancel}(C_{point})$$

The question is *how* and *where* to define such a cancellation point?

In order to determine the exact point, two approaches are possible:

- *static*: set the cancellation point after a fix number of GoPs/frames/seconds (e.g. $C_{point} = 7$, where 7 can be the $7^{th}$ GoP/frame/second);
- *dynamic*: keep a history with the cancellation points for each particular object and compute some kind of mean.

It should be expected for the cancellation point to vary during the period in which a video object resides inside the cache, depending on the video object's popularity (hit count/hit rate) but also depending on its size and quality. Because of this, the static and dynamic approaches for setting the cancellation point should be combined: when a new object is inserted into the cache, an arbitrary cancellation point should be set (say $C_{point} = 10$ seconds), then after the object receives a minimum number of hits, the dynamic approach should be used.

Having such a cancellation point could impact the way video objects are cached: no new object (object received from a remote source) should be cached unless the requesting client receives data pass the cancellation point.

Another possible use for this point would be when counting the hits for a cached object: unless the client that has requested the video watches it pass the cancellation point, the hit count for the object is not increased. If cache replacement is done considering the utility values of the cached objects, and if the hit count has a significant importance when computing those values (see Equation (3)), then this would help discarding objects that are not very popular in a quick and efficient manner.

It is often the case that the client looses interest towards the end of the video (especially in the case of long videos), gets bored and cancels the play back. In order to model this, a second *cancellation point* could be set in a similar way with the first (either static, or dynamic). This second cancellation point could be used when performing cache replacement: if an object is considered for replacement, one possibility would be to first discard the chunk from the second cancellation point to the end of the object, and only then discard other parts (see Figure 6).

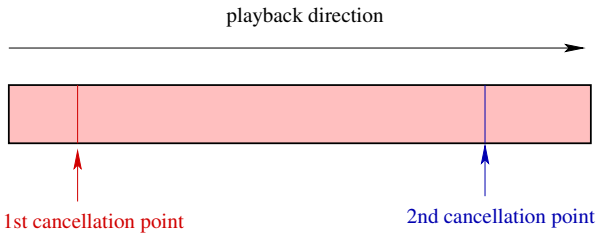



Figure 6: Cancellation points inside a video object

The data between the two cancellation points can be seen as the video part that is most likely to be watched by the potential clients.

# 3 Measurements and Test Setup

We've performed a number of measurements regarding the efficiency of the proposed video caching system using byte hit ratio (the amount of data that has been serviced from within the cache when compared with the total amount of serviced data) as metrics. Due to the high demands in terms of bandwidth of most streaming applications a highly efficient video proxy-cache is most desirable.

The actual measurements were done under some idealized assumptions, as: clients play back a requested object from the beginning to the end of the stream (no interruptions/cancels during the play back) and there are no bandwidth or other hardware resources limitations. Also, no communication delay or errors were considered. The only limitation is the size of the cache in the participating nodes. Those assumptions (which make a quasi-optimal test setup) are not realistic when thinking of a real-world scenario but consist a good comparison basis for such scenarios.

The data we use (see Section 4 for details regarding the logs characteristics) consist of the object id, the object size and the moment the request is received in the component responsible for the proxy-cache's admission control. Because of the assumptions we made, virtually no request will be discarded: in the case the size of the object surpasses the proxy-cache's capacity,

we still consider its delivery but without trying to cache it and we only increment a counter for the number of object cache add failures.

During the tests we've considered a segmented representation of the video objects, similar to the one presented in [5]. Under this representation, the size (measured in Kilobytes) of the $i^{th}$ segment equals $2^{i-1}$, $i \geq 1$ and segment *0* contains block *0*. Also, we've considered that when an object is first requested, it's first $k$ segments are cached. If an object considered for cache replacement has at most $k$ segments left in the cache, it is fully discarded while if it has more than $k$ segments, only the last segment is discarded under the considered cache replacement strategy. No *cancellation points* were set.

When performing the tests two values, $k = 11$ and $k = 15$, were considered. They are equivalent with caching 1 MB respectively 16 MB of data when an object is first requested. In addition to those two values, we've also decided to emulate a "web-like" behavior for the proxy-cache: the objects are cached and discarded as if they would have only one segment, completely disregarding their size.

We've considered a utility based cache replacement strategy (if cache replacement is required, the objects having the smallest utility values are discarded - the data replication or data partitioning features were disabled, so no object movement within the federate cache has been considered). The utility value of each object *obj* is computed using the formula from Equation (3).

Table 1 contains the coefficient's values for which tests were performed while considering the utility based cache replacement strategy.

| *Cache replacement strategy ID* | $coef_1$ *value* | $coef_2$ *value* | $coef_3$ *value* | $coef_4$ *value* |
|---|---|---|---|---|
| CRS1 | 0.50 | 0 | 0.50 | 0 |
| CRS2 | 0.80 | 0 | 0.20 | 0 |

Table 1: Coefficient values considered when computing the utility of an object

When considering CRS1 and CRS2, the *quality* value for all objects was set to 1, meaning that no transcoding capabilities are considered: the objects are available and requested only in the original quality. Those two cache replacement strategies consider only size and hit rate: in CRS1 the two characteristics are considered equal, while in CRS2 the size of the object is considered to be 4 times more important than the hit rate (CRS2 might be deployed in situations when the external bandwidth is scarce).

**The number of active proxy-caches**

We repeat the mentioned test sequence considering that there are 1, 2, 3, 4 and 5 active proxy-caches inside the LAN. When multiple active proxy-caches are considered (2, 3, 4 and 5), we assume that the split operation (similar to the one described in [11]) - which is needed to obtain a new running proxy-cache - has been successfully completed prior to the start of the evaluation session.

**The amount of available cache space**

The tests were run under the consideration that the amount of cache space available ranges from 1% to 10% of the total amount of transferred data during an evaluation session. In practice this means running the test 10 times, each time with a different cache limit.

# 4 Data used for the simulation

The data used for the simulation has been generated using WebTraff [12], a synthetic web traffic generator. We've opted for this tool because of the assumptions we've made. It provides the basic data we need: file id, size and request arrival time. The characteristics of the 4 traces used are presented in Table 2.

| *Trace ID* | *No. of requests* | *No. of objects* | *Zipf slope* | *One-Timers (% of total obj.)* |
|---|---|---|---|---|
| 1 | 1000 | 300 | 0.3 | 70 |
| 2 | 1000 | 300 | 0.3 | 30 |
| 3 | 1000 | 300 | 0.75 | 70 |
| 4 | 1000 | 300 | 0.75 | 30 |

Table 2: Characteristics of the artificial trace logs

The total size of the objects was approximately 3GB and the slope for the Pareto tail of the document size distribution was 1.2 (the document sizes tend to be clustered with many small documents and a few large ones). No correlation between the size of the objects and their popularity was considered (the object size and the object popularity are independent characteristics).

Our intent was to test the effectiveness of the distributed proxy-cache system under both lightly skewed (Zipf $\alpha = 0.3$) and more severe skewed (Zipf $\alpha = 0.75$) object popularity distributions and with varying amount of one-timers (ob jects requested only once) (30% for traces 2 and 4 vs. 70% in traces 1 and 3).

# 5 Results and Discussion

The impact of the number of one timers on the byte hit ratio can be observed for the considered cache replacement strategies when comparing subfigure (a) with subfigure (b) for a Zipf slope of $\alpha = 0.3$ and subfigure (c) with subfigure (d) for a Zipf slope of $\alpha = 0.75$ on all figures from Figure 7 to Figure 12.

If subfigure (a) and (c), respectively subfigure (b) and (d) are compared, we can observe the impact of the objects popularity under different proportions of one-timers from the total amount of requests (70% for the first comparison and 30% for the second).

Surprisingly enough, larger values for byte hit ratio are achieved when the number of one-timers is greater (70%, when compared with 30% in our tests). If the number of one-timers is large, than a small number of objects will ensure most of the hit rate and cache replacement

operations discarding objects that might be useful in the future, will be performed rarely. When the number of one-timers is small, the byte hit ratio drops as cache replacement operations are more frequent and objects that might be requested later on are discarded.

Regarding the change of the byte hit rate values with the number of active proxy-caches inside the system, as one would expect, the byte hit ratio increases with the number of active proxy-caches, and with the size of the cache, but the benefits tend to diminish as the number of active proxy-caches increases (the increase in byte hit rate is greater when moving from 1 to 2 or from 2 to 3 active proxies than it is when moving from 3 to 4 or from 4 to 5 active proxies). This can be seen in all the measurement figures and seems to indicate that a relative small number of active proxy-caches are sufficient to ensure maximal benefits in terms of byte hit rate.

When comparing the byte hit rate values with the cache size, we observed that in cases when finer granularity is considered for adding/removing segments (Figures 7, 8, 9 and 10) the values for byte hit rate are stabilizing at some particular point past which no further increase is observed. The exact point when this happens depends on cache replacement strategy as well as on the popularity of the objects and the number of one-timers.

It can be observed (independently from the cache replacement strategy employed) that the shape of the curve describing the byte hit rate changes from a log like curve (e.g. Figure 7) to an intermediary one (something between logarithmic and linear in Figure 9) and finally to a nearly linear one (e.g. Figure 11) as the granularity for the segment add/remove operations gets coarser (from 1MB for Figures 7, 8 to 16MB for Figures 9, 10 and "web-like" caching in Figures 11, 12). Of course, the exact granularity for which this is noticeable may vary with the size distribution of the objects.

Also the values for the byte hit rate are decreasing as the granularity of the initially saved part increases (Figure 7, Figure 9, Figure 11 when CRS1 is considered and Figure 8, Figure 10, Figure 12 when considering CRS2). This is an expected result, as the amount of replaced data in cases when the granularity is coarse (e.g. "web-like") might be greater than the amount of free space needed to host a new segment and the discarded object might be requested again in the future (a more fine granularity used when discarding segments from the beginning of the video better favors the byte hit rate than a coarser one).

When Figure 7 (cache replacement strategy set to CRS1, initial segment 1MB) and Figure 8 (CRS2), respectively Figure 11 (CRS1, initial segment 16MB) and Figure 12 (CRS2) are compared we can see that the obtained values for the byte hit rate are almost identical. This indicates that regardless of the object size distribution, the influence of the hit rate is much greater than the influence of the size distribution when it comes to computing the byte hit ratio.

Note: The amount of cache size in all the measurement figures refers to individual nodes. When multiple proxy-caches are used, the values on the $x$ axis, say 3, refer to the fact that the size of each proxy-cache in the system equals that particular percentage from the total amount of transferred data. The size of an aggregate proxy-cache can be computed by multiplying the number of active proxy-caches with the values on the $x$ axis.

# 6 Conclusion and Future Work

It seems that a relative small number of proxy-caches (5 for the test data we've considered) with reasonable amount of storing capacity are able to ensure a high byte hit ratio. This is desirable especially in situations when the external bandwidth is a scarce or expensive resource.

The exact values depend on the cache replacement strategy considered as well as on the granularity used when video data segment insertion and discarding is performed inside the system. A finer granularity ensures higher values for the byte hit ratio, but one should be careful as choosing a too fine granularity can defeat the purpose of caching:

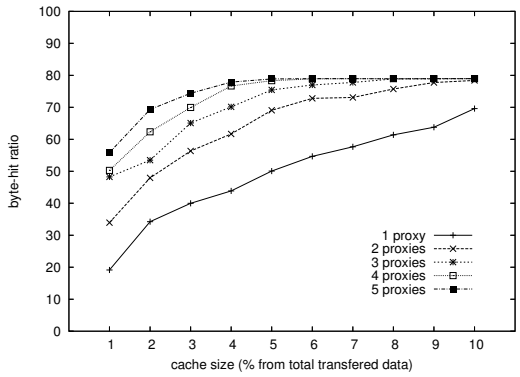
- when an object is requested and only insignificant portions of it are cached, so until sufficient data accumulated within the cache, future requests for the same object will have to be serviced with data retrieved mainly from origin servers; this might be a problem when a different segmentation formula from that in [5] is considered;
- when cache replacement is performed, it may take a lot of extra operations until a useless object is completely discarded from the cache thus adding extra latency.

We intend to investigate the way the byte hit ratio varies in the case different granularities are considered for adding segments of an object (coarser granularity) and removing segments of the same object (finer granularity).

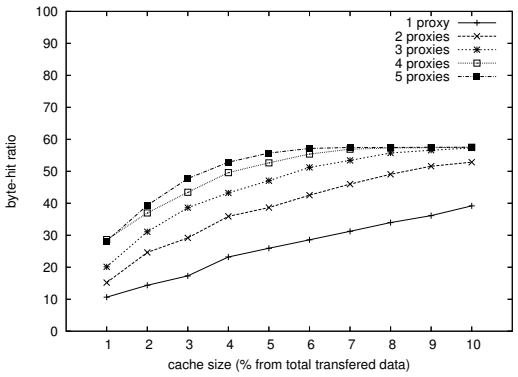
The next step is to consider the latency induced by the inter siblings communications and study how data replication and data partitioning between participants may help achieve improved response time.

Also some additional simulation experiments using different user request patterns, proxy-cache storage capacities and object size distributions should be performed in order to determine the thresholds used by the ranking system.

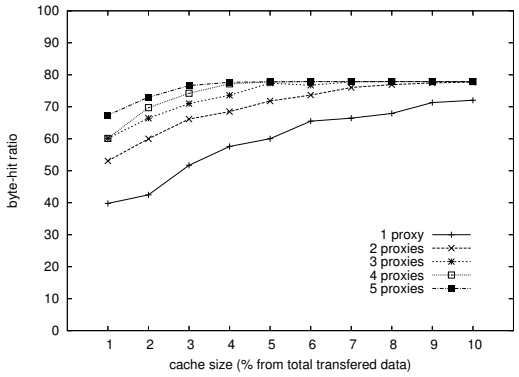
# A Appendix



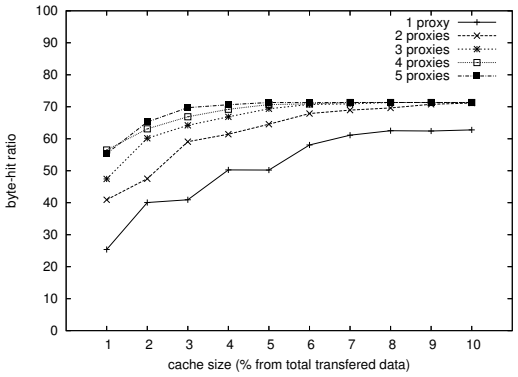

(a) Trace 1 ($\alpha = 0.3$, one-timers 70%)




(b) Trace 2 ($\alpha = 0.3$, one-timers 30%)

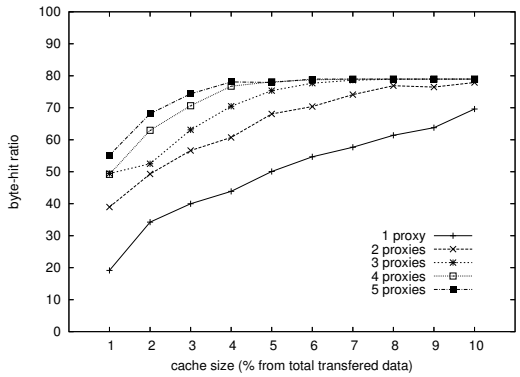



(c) Trace 3 ($\alpha = 0.75$, one-timers 70%)

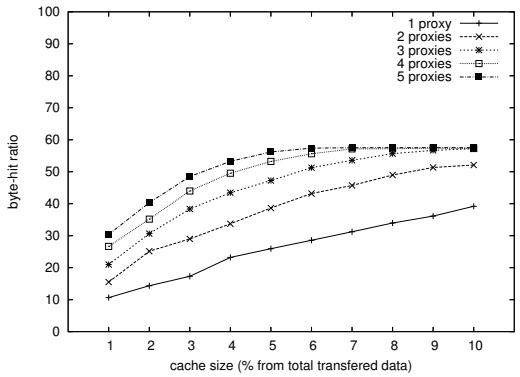



(d) Trace 4 ($\alpha = 0.75$, one-timers 30%)

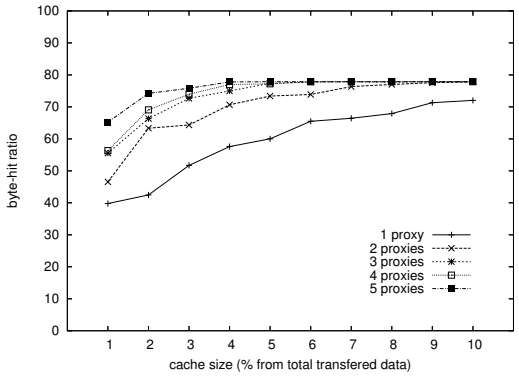
Figure 7: Byte hit ratio values for different traces (CRS1, initial segment 1MB)



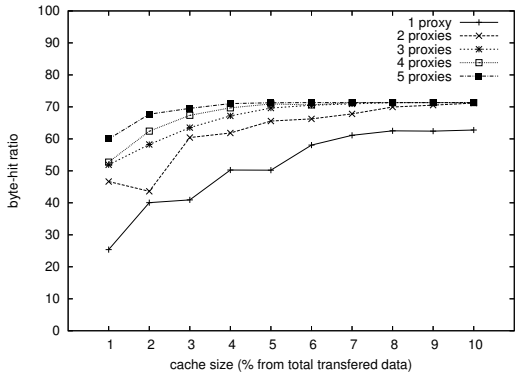

(a) Trace 1 ($\alpha = 0.3$, one-timers 70%)




(b) Trace 2 ($\alpha = 0.3$, one-timers 30%)

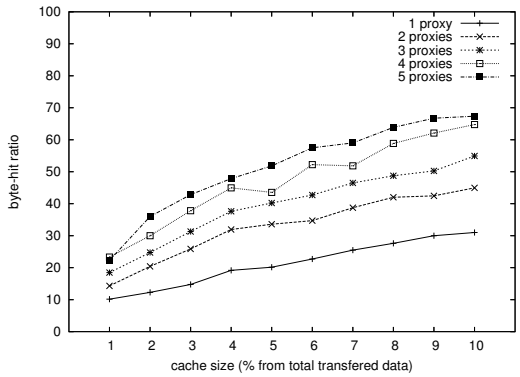



(c) Trace 3 ($\alpha = 0.75$, one-timers 70%)

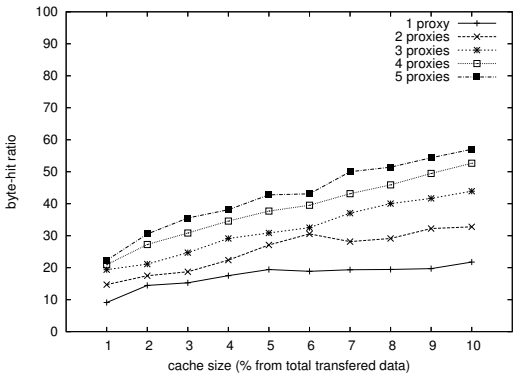



(d) Trace 4 ($\alpha = 0.75$, one-timers 30%)

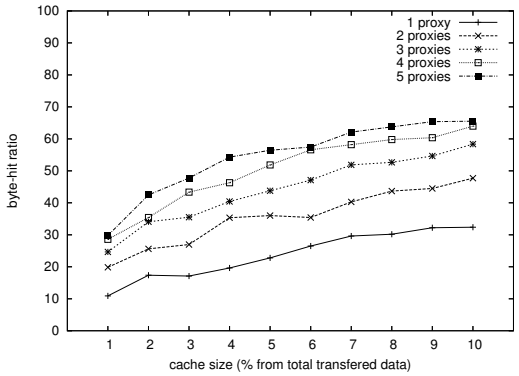
Figure 8: Byte hit ratio values for different traces (CRS2, initial segment 1MB)



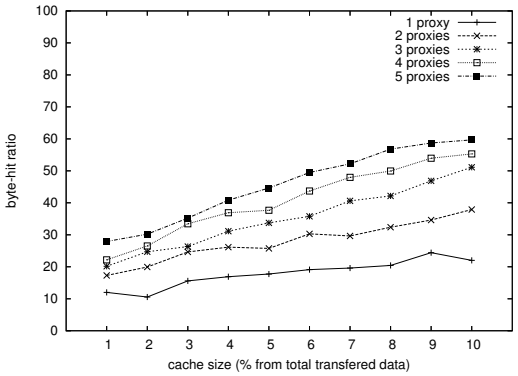

(a) Trace 1 ($\alpha = 0.3$, one-timers 70%)




(b) Trace 2 ($\alpha = 0.3$, one-timers 30%)

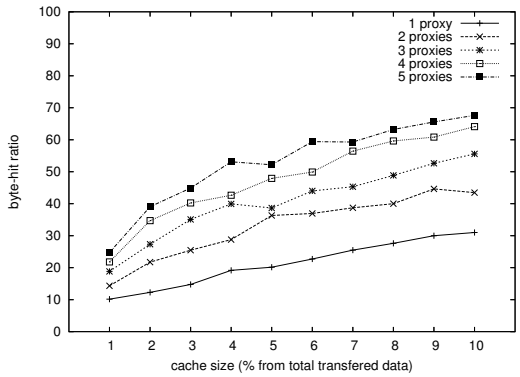



(c) Trace 3 ($\alpha = 0.75$, one-timers 70%)

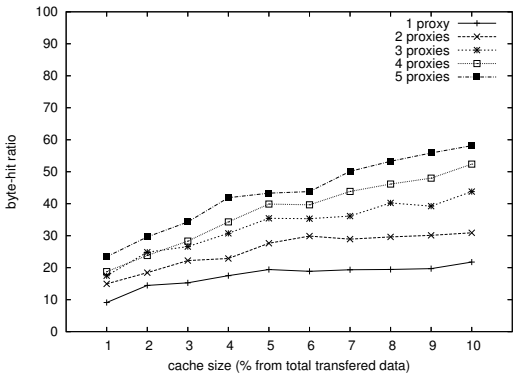



(d) Trace 4 ($\alpha = 0.75$, one-timers 30%)

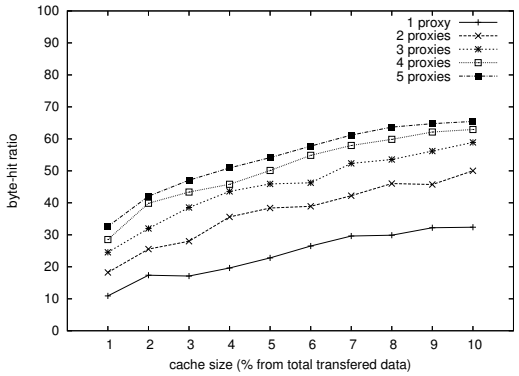
Figure 9: Byte hit ratio values for different traces (CRS1, initial segment 16MB)



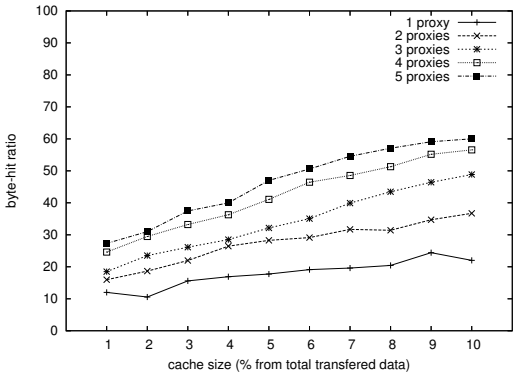

(a) Trace 1 ($\alpha = 0.3$, one-timers 70%)




(b) Trace 2 ($\alpha = 0.3$, one-timers 30%)

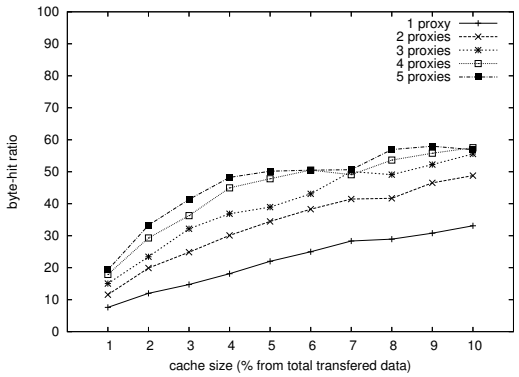



(c) Trace 3 ($\alpha = 0.75$, one-timers 70%)

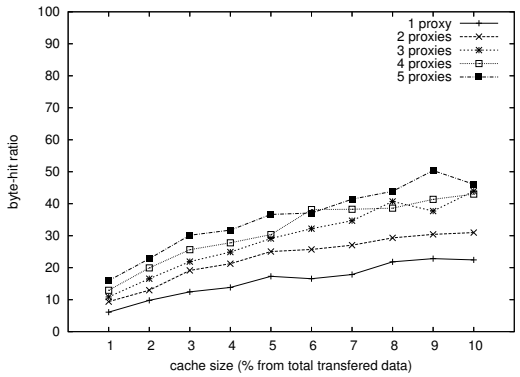



(d) Trace 4 ($\alpha = 0.75$, one-timers 30%)

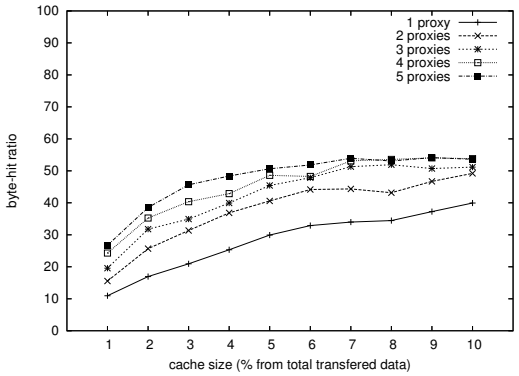
Figure 10: Byte hit ratio values for different traces (CRS2, initial segment 16MB)



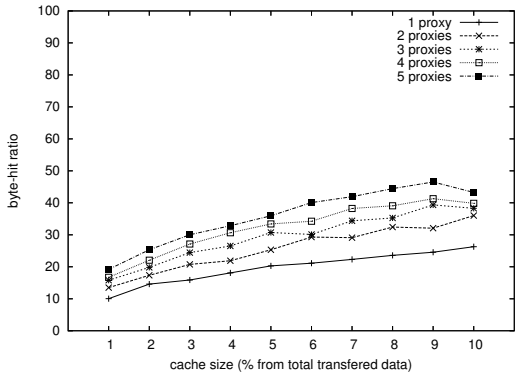

(a) Trace 1 ($\alpha = 0.3$, one-timers 70%)




(b) Trace 2 ($\alpha = 0.3$, one-timers 30%)

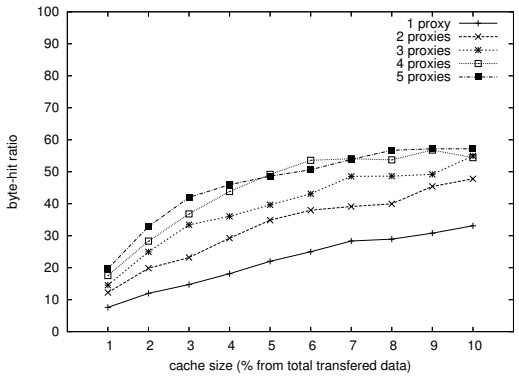



(c) Trace 3 ($\alpha = 0.75$, one-timers 70%)

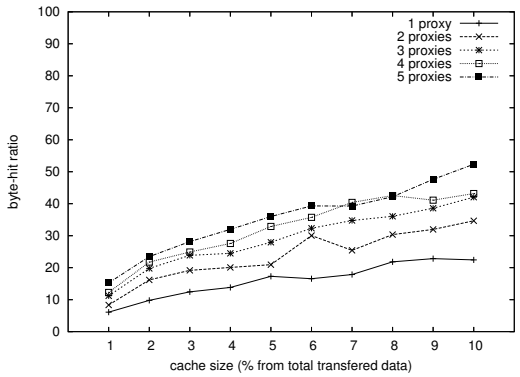



(d) Trace 4 ($\alpha = 0.75$, one-timers 30%)

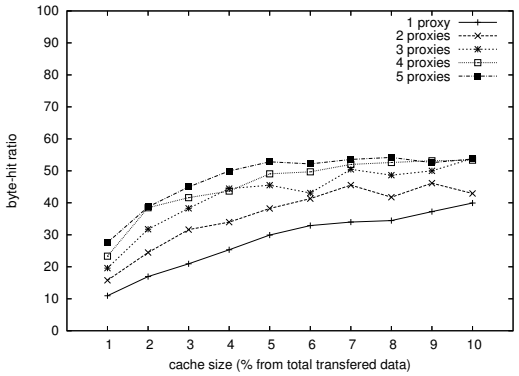
Figure 11: Byte hit ratio values for different traces (CRS1, "web-like" caching)



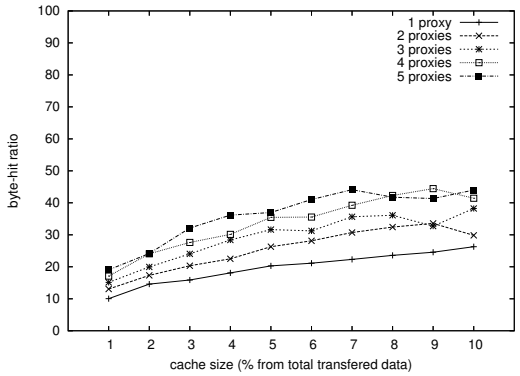

(a) Trace 1 ($\alpha = 0.3$, one-timers 70%)




(b) Trace 2 ($\alpha = 0.3$, one-timers 30%)




(c) Trace 3 ($\alpha = 0.75$, one-timers 70%)




(d) Trace 4 ($\alpha = 0.75$, one-timers 30%)

Figure 12: Byte hit ratio values for different traces (CRS2, "web-like" caching)

## References


[1] Subhabrata Sen, Jennifer Rexford, and Donald F. Towsley. Proxy prefix caching for multimedia streams. In *IEEE INFOCOM*, pages 1310–1319. IEEE Computer Society, 1999.

[2] Wei-Hsiu Ma and David Hung-Chang Du. Reducing bandwidth requirement for delivering video over wide area networks with proxy server. In *IEEE International Conference on Multimedia and Expo (II)*, pages 991–994. IEEE Computer Society, 2000.

[3] H. Fabmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L.H. Hsu. Proxy servers for scalable interactive video support. *Computer*, 34(9):54–60, 2001.

[4] Seong-Ho Park, Eun-Ji Lim, and Ki-Dong Chung. Popularity-based partial caching for vod systems using a proxy server. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01)*. IEEE Computer Society, 2001.

[5] Kun-Lung Wu, Philip S. Yu, and Joel L. Wolf. Segment-based proxy caching of multimedia streams. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 36–44. ACM Press, 2001.

[6] Elias Balafoutis, Antonis Panagakis, Nikolaos Laoutaris, and Ioannis Stavrakakis. The impact of replacement granularity on video caching. In *IFIP Networking 2002*, volume 2345 of *Lecture Notes in Computer Science*. Springer, 2002.

[7] Jussi Kangasharju, Felix Hartanto, Martin Reisslein, and Keith W. Ross. Distributing layered encoded video through caches. In *IEEE INFOCOM*, pages 1791–1800. IEEE Computer Society, 2001.

[8] Stefan Podlipnig and László Böszörményi. Replacement strategies for quality based video caching. In *IEEE International Conference on Multimedia and Expo (ICME), Vol. 2*, pages 49–53. IEEE Computer Society, 2002.

[9] Soam Acharya and Brian Smith. Middleman: A video caching proxy server. In *Proceedings of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video*, 2002.

[10] Roland Tusch, László Böszörményi, Balázs Goldschmidt, Hermann Hellwagner, and Peter Schojer. Offensive and defensive adaptation in distributed multimedia systems. *Computer Science and Information Systems*, 1(1):49–77, 2004.

[11] Claudiu Cobârzan. Dynamic proxy-cache multiplication inside LANs. In *Euro-Par 2005*, volume 3648 of *Lecture Notes in Computer Science*, pages 890–900. Springer, 2005.

[12] Nayden Markatchev and Carey Williamson. Webtraff: A gui for web proxy cache workload modeling and analysis. In *MASCOTS '02: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*. IEEE Computer Society, 2002.

**Institute of Information Technology**
**University Klagenfurt**
**Universitaetsstr. 65-67**
**A-9020 Klagenfurt**
**Austria**

http://www-itec.uni-klu.ac.at


**University Klagenfurt**