Improving Performance of Dynamic Caching in Streaming Proxies

Edison Ishikawa[1,2] and Claudio L. Amorim[1]

[1]Systems Engineering Program - COPPE
Federal University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
{edsoni,amorim}@cos.ufrj.br

[2]Computer and Systems Engineering Department
Military Institute of Engineering
Rio de Janeiro, RJ, Brazil
ishikawa@ime.eb.br

Abstract

Performance studies of dynamic caching schemes for streaming media assume that ring buffers coupled with a FIFO replacement policy offer an efficient caching mechanism to implement streaming proxies in Content Distribution Networks (CDN). However, in this work we show that the ring buffer structure can be extended in a way that improves significantly streaming proxy performance. More specifically, we introduce a novel dynamic caching mechanism, namely the Cooperative Dynamic Caching (CDC) that enables ring buffers to perform streaming operations on video segments such as splitting and chaining while implementing near optimal cache replacement policies. Using the ns-2 simulator, we implemented and compared performance of a CDC-based Video-on-Demand (VoD) proxy against that of streaming VoD proxies with conventional dynamic caching schemes for different parameter values of cache size and interarrival time rates between requests. Our experimental results across several performance metrics such as client blocking rate, backbone traffic rate, peak bandwidth , and average startup latency, indicate that CDC outperforms significantly existing schemes for implementing dynamic caching in streaming proxies

Key Words—multimedia streaming, video proxy , cache management

1. INTRODUCTION

The growing demand for streaming media applications over the Internet, such as distance learning and videoconference faces the inadequate delivery of streaming media of current Internet service providers. In particular, both low resolution and high startup latency of video streams that clients experiment prevent the widespread use in a cost-effective manner of video-on-demand (VoD) services to large audiences. Although increasing the availability of broadband access to the users mitigates the so–called last-mile problem, in practice backbone network bandwidth limits the traffic a VoD server can generate and yet guarantee scalable performance. Therefore, reducing backbone traffic rate is essential. As a result, several existing proposals advocate the use of Content Distribution Networks (CDN) to deliver high-quality videos to geographically distributed clients in a scalable manner.

Figure 1 illustrates the basic CDN infrastructure, in which a distribution network connects the video server to several streaming proxies placed at the network edges near to the clients. Streaming proxies not only unload the server but also add two advantages for continuous media delivery. First, they improve the quality of service (QoS) of streaming media applications since

proxies can store in advance streaming media segments, thus hiding the network's jitter while allowing faster access to them by connected users. Second, streaming proxies can implement efficient caching systems to reduce the demand on the video server, thus alleviating the traffic in the distribution network's backbone.

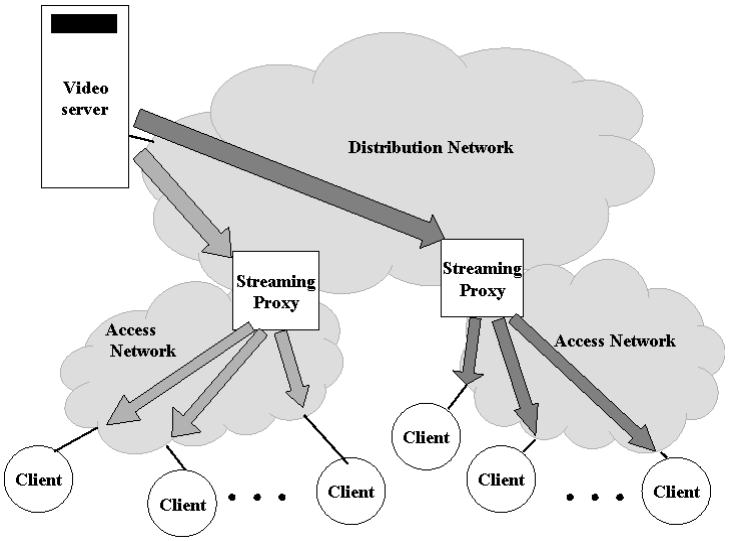

Figure 1. Components of a Content Distribution Network

Recently, several works in the literature [5,14,13,2,18] have proposed VoD proxies with dynamic caching to efficiently deal with video objects. In contrast with traditional static web caching of text files and static images, dynamic caching allows a streaming proxy to explore temporal relationships between streaming segments to further increase cache hit. Also, these studies assume ring buffers as the basic mechanism to implement dynamic caching schemes to handle intra-stream temporal relationships in a simple and efficient way. In such schemes, the proxy cache serves a video request by allocating a ring buffer of a certain size to receive continuously the video stream from the server before forwarding it to the client, until the video ends.

Complementary to previous works, we propose an efficient management of ring buffers using the available memory space in the proxy to guarantee QoS over entire streaming objects. More specifically, we introduce the Cooperative Dynamic Caching (CDC), a new scheme that enables ring buffers to perform operations on streaming segments such as splitting and chaining while implementing near optimal cache replacement policies. Using the ns-2 simulator, we compared performance of CDC-based VoD proxy against that of VoD proxies that use dynamic caching in a conventional manner, for different cache sizes and interarrival time rates. Our experimental results across several performance metrics such as client blocking rate, backbone traffic rate, peak bandwidth, and average startup latency, indicate that CDC outperforms significantly existing dynamic caching schemes.

Our main contributions are:

- We introduce the Cooperative Dynamic Caching (CDC), a new mechanism for improving scalable performance of streaming proxies.
- We propose and evaluate a near optimal cache replacement policy for CDC.
- We show that CDC can easily support VCR-like interactive operations.
- We demonstrate that the dynamic caching problem CDC introduces is NP-complete and propose a near optimal cache replacement policy.
- Through detailed simulations using the ns-2 simulator, we show that CDC outperforms significantly existing dynamic caching schemes for implementing VoD proxies.

This paper is organized as follows. In the next section, we examine the ring buffer scheme for cache management of streaming proxies. Section 3 describes in detail the CDC design. In Section 4, we develop a new heuristic for replacement of video segments in CDC. In Section 5, we present simulated results of a performance comparison of CDC against that of typical dynamic caching implementations. In Section 6, we present related works. Finally, in section 7 we conclude.

2. DYNAMIC CACHING

For simplicity, let us consider a situation where a video streaming proxy uses ring buffers for implementing a dynamic caching system and that users issue requests to the same video. Assume a Variable Bit Rate (VBR) video stream and that to each new request a ring buffer with an average size of b time units is allocated, which correspond to a few seconds of video stream. The basic operations of a ring buffer are illustrated in Figure 2. In Figure 2a, the ring buffer X receives continuously the video stream from the server and sends it to the client, until the video ends. A ring buffer operates like a moving window over the video stream where each segment that moves outside of the window's frame size becomes older. Old segments are useful when the ring buffer becomes full and new space needs to be allocated. In this case, the oldest segment is automatically made available for a FIFO replacement policy. It is important to notice that the cache system must keep video segments cached as long as they remain inside of ring buffers, only after a video segment moves outside of the buffer's space that the cache replacement policy can afford to discard it.

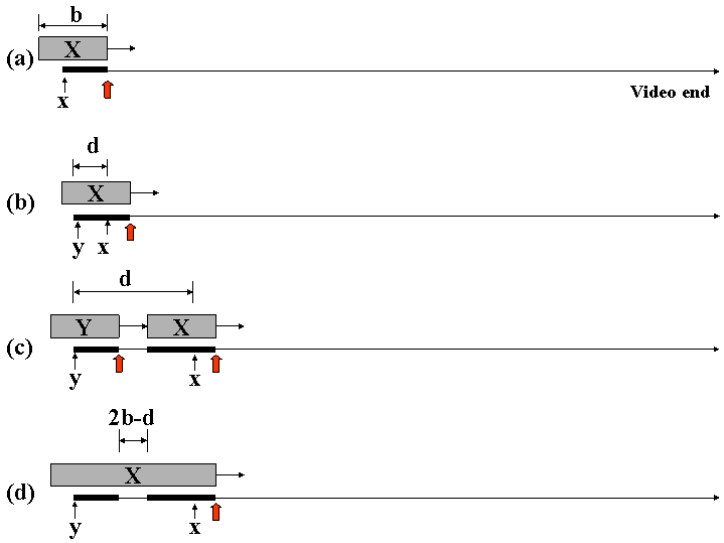

Figure 2. Ring buffer operations

Figure 2 illustrates ring buffer operations for two video requests x and y. When request x arrives at the proxy, a new ring buffer X is instantiated to receive the video streaming from the server before forwarding it to the client (Figure 2a). Suppose that y arrives d time units later, where d is defined as the temporal distance between the two consecutive requests. If d is smaller than the buffer size b then the video segment that y requested is likely to be in the ring buffer and thus y can be served from X (Figure 2b). Otherwise, if d > b, a new ring buffer Y will be created (Figure 2c) and assigned to serve y. So far, ring buffers offer a simple mechanism for dynamic caching of streaming media. Nevertheless, there is some room to improve performance of such a dynamic caching scheme. More specifically, our dynamic caching design is derived from the following observations on sharing characteristics of video segments and the way ring buffers are used in current dynamic caching systems.

First, an alternative to deal with situation in figure 2c is illustrated in figure 2d. The idea is to extend the size of X by (d-b) units while filling the extra buffer with a video patch [6] from either

the cache or the server depending on whether the patch is found in the cache or not. Note that the resulting video streaming has no interruption and it is equivalent to merge video objects that X and Y contain into a single one that instead requires a larger ring buffer of size equal to 2b + (d-b).

Clearly, a potential drawback is the cost of the large buffer we required if it has to be reserved for the entire video playback. Fortunately, this is not the case, as we will show shortly.

Second, caching video prefixes is essential for improving proxy's performance [11] since video playback often starts at the beginning of the video. Furthermore, caching video prefixes prevents the server from sending patches to the users and thus can save both network and server resources. Therefore, it is worth to keep video prefixes stored in the cache even if they move outside of ring buffers.

Also, existing designs of dynamic caches assume that the memory space allocated to a ring buffer cannot be released until the associate video stream ends. The problem is that depending on the temporal distance distribution of request arrivals, a proxy can run out of memory quickly and the proxy will not admit new users since it can not instantiate new ring buffers, which ultimately can affect severely proxy's scalability. The solution we propose for better utilization of cache space is to enable split and chain operations dynamically on ring buffers in order to create new ones of variable size accordingly to the demand on the video segments. For instance, if a ring buffer can be split in two ring buffers each of which with a minimum size, as shown in Figure 2c, the memory space of size (d-b) that is released can be used to admit new requests. In general, we want to be able to split long ring buffers to obtain enough memory either to join two existing ring buffers contiguously or to increase the size of an existing ring buffer, depending on the video popularity. Next, we describe a novel structure for dynamic caching of streaming media that supports such management operations of ring buffers whose size can vary dynamically.

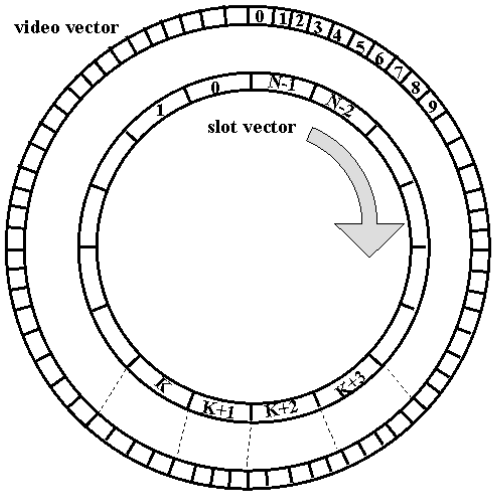

Figure 3. Relationship between Slot and Video vectors

3. COOPERATIVE DYNAMIC CACHING

The Cooperative Dynamic Caching mechanism we propose maintains two mapping structures for each new video streaming stored in the cache system: the video vector and the slot vector. These two structures are implemented as circular vectors that work in combination, as shown in Figure 3. The video vector represents the sequentially numbered video units that compose the entire video stream, but they contain only the video's metadata that CDC requires to access and manage its cache. The slot vector consists of fixed time slots so that the slot collection represents the video duration.

For mapping video units to the cache, each unit of video vector contains also a timestamp of the corresponding video segment that is stored in the cache. Specifically, when a video segment

moves into the cache, CDC associates to it a video vector unit that contains an appropriate pointer to the cache address. Also, while a new video unit moves into the ring buffer CDC changes the segment's caching priority from 0 (absent) to 2 (present and reserved) and when it moves outside of the ring buffer its priority is changed to 1 (present and not reserved).

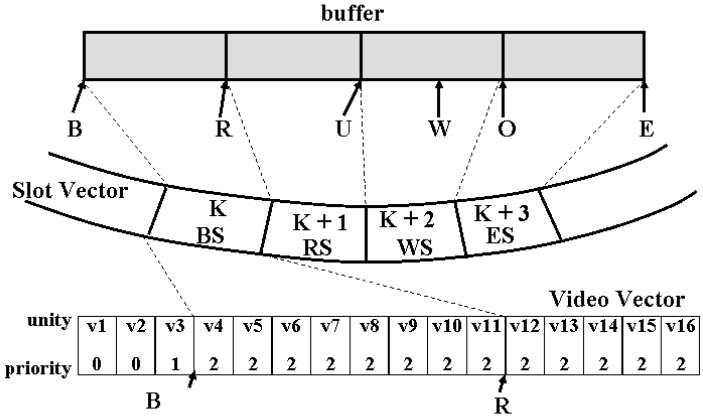

Figure 4. Relationships between ring buffer, slot buffer, slot vector, and video vector.

CDC uses the slot vector to reserve cache space to each new admitted user, as follows. When an user requests a video playback, CDC reserves in advance some slot units of the slot vector, which we called the slot buffer. The slot buffer implements the ring buffer abstraction associated with the user's play out buffer. Figure 4 illustrates a ring buffer mapped onto the four slots k through k+3 of the slot vector. These four slots, namely BS (begin slot), RS (read slot), WS (write slot), and ES (end slot), map directly the associate areas of the ring buffer to the video units. The priority field of each video unit is set and controlled by CDC's replacement policy. Slots that do not belong to any slot buffer are set to OS (Out of Slot buffer).

As the video plays back, the slot vector rotates clockwise synchronously along the video vector causing new units of video to enter and old ones to leave the slot vector. Figure 4 illustrates the ring buffer structure, which uses six pointers to control the video playback, where pointers R and W indicate the next place for reading from and writing to the ring buffer, respectively. The remainder control pointers, namely B, U, O, and E, control the limits of each of the four areas within the ring buffer, which correspond to the slot types BS, RS, WS, and ES, respectively. The control pointers simply update the video unit status whenever the slot's limits they control are crossed.

In Figure 4, the BS slot had its associate video units set to priority 2 (present and reserved), so they cannot be discarded. Once a video unit crosses the begin pointer B, its priority is changed to 1, which indicates that it can be evicted from the cache, if it is necessary. Similarly, the overflow pointer O controls the WS slot's limits sooner a video streaming starts feeding the WS slot. In this case, if the writing pointer W crosses O, then a warning signal is generated to inform CDC of a risk of ring buffer's overflow. An overflow signal generates a warning message that will inform the video streaming source to decrease its transmission speed. In the same way, the underflow pointer U controls the RS slot of the ring buffer so that whenever the writing pointer W crosses U a warning is generated to the CDC proxy, which will look for another video source. Most importantly, such a mechanism of warning signals enables a streaming proxy to deal efficiently with network jitter and thus to guarantee QoS. Specifically, warning signals allow a streaming proxy to sense and react to network delay variation and thus anticipate its effects by interactively asking the server for sending video segments either quickly or slowly, accordingly.

3.1. Locating Slot Buffers and Slot Limits

The design of streaming proxies based on CDC requires it dynamically locates a slot buffer that maps to a given video, and also to identify the slot's limits. This identification depends on the selected slot type, which in turn corresponds to one of the five pointers: B, R, U, O, and E.

CDC uses timestamps to locate precisely video segments within slot buffers. When the first request for a certain video arrives at the proxy, CDC creates the video's data structure, which is marked with timestamp ST (start time) and sets the video's initial state. Suppose that a new client arrives at time CT (Current Time) and asks for the video unit whose timestamp is UTS (Unit TimeStamps). So we have to find the RS slot whose R pointer points to the video unit with timestamp UTS. First, we calculate the video position by subtracting CT from ST and adding the initial time UTS the new client requested. Next, we determine the RS slot dividing the previous result by the slot time length (SL) after which we apply the NS (number of slots) module operation. This operation accounts for the number of complete turns the slot vector performed since time ST. Thus RS calculation is given by:

$$RS= [(CT-ST+UTS)/SL]\%NS. \quad (1)$$

Once the RS slot has been determined, few simple calculations are required to find the remainder slots of the slot buffer. Let PT (Period of Time) be the time spent for a complete rotation of the slot vector, i.e., the video's duration. Given ST, CT, PT and the RS slot number, given by (1), the number of cycles NC can be calculated by the expression:

$$NC = [CT – ST – (RS * SL)]/ PT. \quad (2)$$

To determine slot RS' limits, i.e., the slot' initial time (SIT) and final time (SFT) we simply compute:

$$SIT = CT – ST – NC * PT – RS * SL \quad (3)$$

and

$$SFT = SIT – SL. \quad (4)$$

Both SIT and SFT will return time values relative to the time the video started. Since video units can be accessed using either sequential numbers or timestamps, the CDC proxy can delimit the video unit that each of ES, WS, RS and BS slots controls using the appropriate SST and SFT expressions.

3.2. Streaming operations in CDC

We describe briefly how CDC implements basic streaming operations such as chaining [12] and splitting. In addition to the four slot vector types (BS, RS, WS, and ES), CDC introduces the Link Slot (LS), a new slot type that enables implementation of chaining by extending the basic slot buffer mechanism. Figure 5.b shows four LS joining slot buffers A and B. Also, B' slots WS and ES and A' slot BS changed their types to LS. The reason is that LS' limits are not related to ring buffer control pointers, i.e., when a video unit crosses LS's limits it requires no change of video unit's priority nor it generates any warning message. In Figure 5.b, slot buffers B and A have the BS, WS, and ES slots to control the content of the chained slot buffers. For simplicity, we consider only one user per slot buffer and a single video.

Chaining

Consider clients A and B in Figure 5a where they access two distinct streams of the video, each of which feeds its associate slot buffer. In this case, CDC uses some LS slots, namely K-3 to K to chaining the two streams and eliminating one, as shown in Figure 5b. Due to chaining, WS and ES of slot buffer B and BS of slot buffer A became redundant and thus CDC changed their types to LS. Also if WS, ES slots, the four LS slots, and BS contents are not cached, CDC will request them to the video server, which will reply with corresponding video patches.

Splitting

Whenever the proxy cache runs out of memory space, it can eliminate some LS slots to re-allocate memory space to support the new video stream as shown in Figure 5c, where original K-5, K-4 and K+1 slot types were restored to their original types.

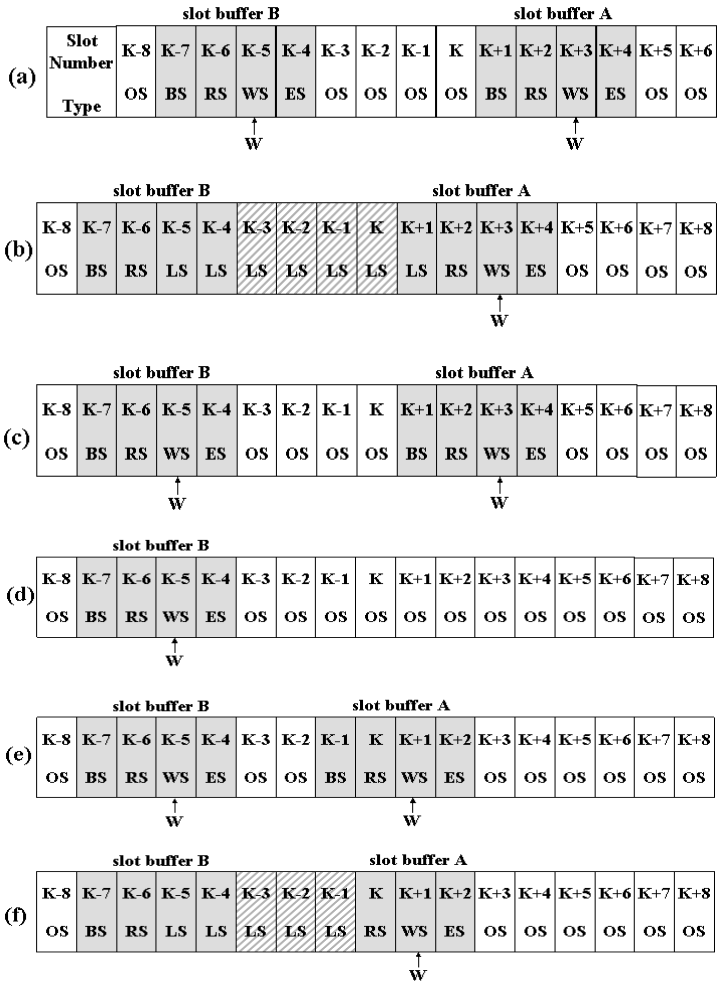

Figure 5. CDC extended operations to ring buffers: normal (a), chaining (b), splitting (c), pause (d), resume (e) and chaining(f).

Support to VCR Operations

To implement stop and pause operations, CDC can simply remove the unused slot buffer from the slot vector, reset the type of the slots to OS, and release corresponding video units. This is illustrated in Figure 5d where user A issued a pause operation and thus slot buffer A was removed.

Other discrete-type VCR operations like their continuous VCR counterparts allow users to jump forwards and backwards to certain points in the video streams but in a way that avoids unnecessary continuous display of the video segments while searching for the point the user requested. CDC implements these operations by enabling users to ask for specific segments of video objects. For example, client A asked for a resume operation as shown in Figure 5-e, which caused CDC to allocate slot buffer A. After, CDC chained (Figures 5f) slot buffers B and A, assuming that there is enough space in the cache. In addition, a patch might be necessary to reload WS, ES plus the seven LS slots because of the chaining operation that CDC performed.

So far, we showed that CDC offers an easy way to handle true VoD functionalities. Most importantly, CDC can further improve scalable performance of streaming proxies, as we will show in the next section.

4. CACHE REPLACEMENT POLICY

For simplicity, we again assume a single video and a cache system that supports a certain amount of slot buffers plus link slots. Figure 6a shows the initial state of the slot vector, in which all slots are initially free. Next, the first request arrives and a snapshot of the slot vector (Figure 6b) is taken after 3-slot time units. As we can see in the Figure, CDC reserved four slots to create a new slot buffer for receiving the video streaming, which flows through a regular channel (represented as a black arrow) that the proxy established with the video server.

Figure 6c presents the slot vector after 3 slot time units where the video prefix remained in cache with high priority even being outside of a slot buffer. In Figure 6-d, another slot buffer was created to serve the second request, however, CDC re-used the video prefix held in cache to attend this request instead of establishing another regular video channel with the server. In this case, CDC simply changed the slot buffer of the video prefix to work as link slots while maintaining their actual video contents, besides mapping the link slots to the new slot buffer. As a result, CDC saved network bandwidth, i.e., one regular server channel.

Figure 6-i illustrates a situation where a new request found the cache full. CDC has three possible options: 1) to discard the request because the proxy's resources are completely busy; 2) to discard the prefix so that its reserved slots can be re-used to create a new slot buffer; and 3) to release link slots and use them for creating another slot buffer. CDC rarely discards any request (first option) since the other two options are often available. This high-availability property of CDC is that distinguishes it from existing dynamic caching schemes that discard requests once all ring buffers are busy.

CDC takes the second option as last resource because if it discards the prefix then a new regular server's channel must be established for transmitting the entire video streaming from the beginning to attend incoming requests.

Therefore, the third option is most effective, so CDC often takes it. In practice, however, there are many alternative selections of link slots depending on the video request distribution. For simplicity, we show only two of them in Figures 6(j1) and 6(j2). In Figure 6(j1), CDC released six consecutive link slots to create a new slot buffer plus one link slot. In Figure 6(j2), CDC released a group of three link slots plus another group of two link slots to create the slot buffer plus one link slot. Given that the number of alternatives is combinatorial in number of existing sequences of link slots, our next step is to examine such allocation problem in depth.

A LS sequence is characterized by two metrics: size and temporal distance, both measured as an integer number of slots. The size of a LS sequence corresponds to the number of slots that the sequence contains whereas the distance is the number of slots that the sequence has to traverse until the video ends. The LS sequence's cost is the number of extra regular channels necessary to feed the slot buffers whose link slot chain was released plus the duration of each extra regular channel, where the duration is the distance of the link slot sequence.

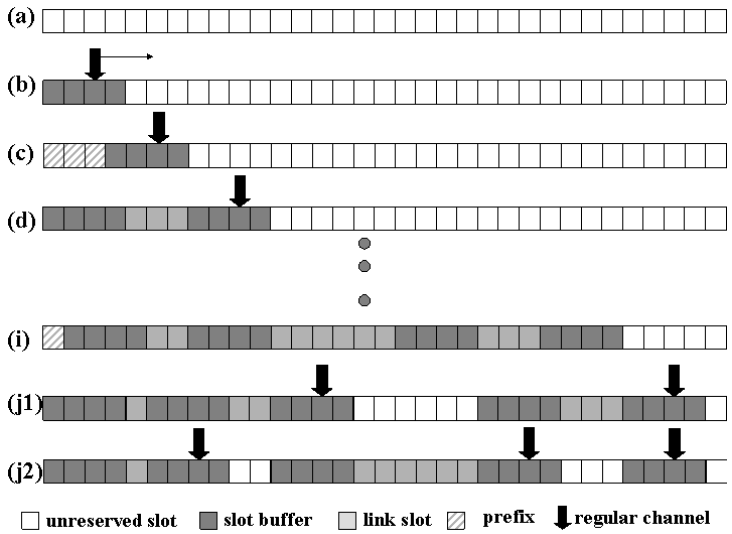

Figure 6. Snapshots of a slot vector

In option 6(j1), we exchanged 6 link slots with temporal distance of 24 units for a single regular channel (RC) that will send the last 24 units. Hence, if we want to save network bandwidth, we should consider that the cost of establishing the RC is 24 time slots. Option 6(j2) traded 3 link slots with temporal distance of 14 units and 2 link slots with temporal distance of 30, for two regular channels plus a total cost of 44 units. Note that CDC needs four slots to create one slot buffer more one link slot to chain the created slot buffer, thus option 6(j1) is preferable to 6(j2). Also note that the cost in time units when we use a slot unit from the cache memory is the same as the cost of using a regular channel for the same amount of time. Also, is better to use one slot instead of two to replace a regular channel.

As this simple example demonstrates, Link slot allocation is an optimization problem that can be formulated better, as follows.

Problem instance: collection of link slot (LS) sequences, number of slots that need to be released, and cost of releasing each link slot.

Question: is there any combination of LS sequences that satisfy simultaneously the three conditions:

1. Is the size of selected LS sequences greater or equal to the number of slots that have to be released?
2. Is the total cost of selected LS sequences smaller than a given value C?
3. Is the number of selected LS sequences the smallest?

In order to prove that this problem is NP-complete we can transform the original problem into its dual, as follows. Suppose that LS sequences correspond to LS buckets that have to be filled up, where each bucket size is measured in number of slots. Also, slots to be released correspond to tokens of size equal to a single slot. So the dual problem instance can be stated as:

Dual problem instance: a set S of buckets, t tokens and cost k. All the units are in temporal distance d, which also is measured in number of slots.

Dual question: how to distribute all the tokens to the smallest possible number of buckets such that the sum of the buckets that received at least one token has minimum cost k. More precisely, the problem can be reinstated, as follows:

Problem: Find S´ subset of S that has at least one token subject to two restrictions: a) the cardinality c of S´ is minimum; b) the total distance of S' buckets is equal to a given value D.

Question: Is there a subset S´ of S such that its cardinality does not exceed c and the sum of bucket distances does not exceed D?

The problem as formulated above, it is clearly the well-known NP-complete problem of Capacity Assignment [16]. Therefore, an optimal solution is too expensive to work under strict time constraints as when implementing a dynamic caching replacement policy. Therefore, we describe next a greedy policy to find victim LS sequences we developed.

4.1. Replacement Algorithm for Link Slots.

Let t be the number of slots we need to allocate new slot buffers and link slots. Let s be the size of a LS sequence and d its distance. Let us define the cost of a LS sequence as a function F(d, t, s) equal to the ratio d/minimum{t,s}. Our goal is to minimize F. Intuitively, a minimum value for F means that we want a LS sequence near to the end of the video that can be exchanged for a regular channel that will be occupied over a small period of time. Also, a LS sequence is little worth if its cost d is substantially greater than the number of necessary slots since it reduces the relative value of the link slot, even for large d because F selects the minimum value between t and s. In this manner, F avoids selecting a costly link slot with an expensive regular channel in place of a low-cost regular channel when it needs to exchange a link slot with small size t. Therefore, F will choose LS sequences that have minimum cost until all the necessary slots are released. It can be shown that the complexity of an algorithm for computing the minimum F of all LS sequences, which is related to inserting and removing LS sequences in increasing order in a priority queue, is O(logN) [15]. Figure 7 shows this algorithm.

```
getFreeSlots( t ) {
        freeSlots=0;
        while ( freeSlots  <  t ) {
                linkSlotSequence=queue.getFirst( );
                freeSlot = freeSlot + linkSlotSequnce.size;
                freeSlot( linkSlotSequence );
        }
}
```

Figure 7. The LS replacement Algorithm

4.2. Prefix Allocation Policy.

Slot buffers are allocated dynamically as new requests arrive. For popular videos, new requests arrive in a way that either CDC does not need new link slots or the LS sequence it requires is short, and thus the video prefix will be automatically stored in the cache. Moreover, the prefix length depends on its popularity, thus if a video is very popular, slot buffers are chaining together as new requests arrive. Also, new requests either do not need link slots to chain slot buffers or they create very short LS sequences. So the chance of releasing link slots will occur only in the final part of the video exhibition.

Note that CDC chooses an LS sequence that causes a minimum increment of the backbone traffic rate. Such a sequence is one closest to the end of the video stream which can release as many link slots as needed by CDC replacement policy. Among the candidate sequences, CDC selects the one with the smallest LS sequence' size, which represents the time interval between two slot buffers that will require two regular video streams to be fed. In this way, CDC can reduce the prefix size in a smart manner (see Figure 6(j1)).

Upon receiving a request for an unpopular video, CDC will allocate a slot buffer to it, and because the prefix is not in the cache, CDC will ask the video server to send the video stream from the beginning. The question is whether or not CDC should maintain low-demand prefixes in cache. Note that even for such a video, a second request may arrive after a small time interval and if the prefix is held in cache CDC could chain the two requests without requesting a patch to the server. Therefore, CDC replacement policy keeps any prefix in cache as long as possible.

CDC does not have an explicit policy to cache prefixes, although its cache management scheme automatically caches the (near) optimal prefix size and the (near) optimal prefix working set. Better, CDC cache management scheme dynamically adjusts the (near) optimal prefix size and (near) optimal prefix size accordingly with video popularity changes. We proof that our management scheme is (near) optimal showing that CDC cache management scheme acts in the same way as gradient descendent based algorithm, but in a dynamical manner.

CDC unit of cache management is the slot. Slot buffers are allocated dynamically as new requests arrive. If a video is very popular, the slot buffers are allocated and chained without using LS to concatenate with others slot buffers until the video end. In this case, all the video is automatically cached, i.e., the prefix has the video length. If a video is not so popular, the slot buffers are chained with link slots. If the proxy has enough memory, all the video is cached. But if the system experiments memory shortage, CDC management scheme chooses one or more LS sequence to be the victim. As gradient descendent algorithm, CDC chooses the LS sequence that will cause the minimum increment in the backbone rate, but also consider the LS size, which in other words is the interval between two slot buffers that will need two regular streams to be feed. The minimum increment in the backbone rate is given by the LS sequence which is the closest to the video end and can leave many LS slots as needed by the system. This way, CDC cache management scheme reduces the prefix in a smart manner. Also, suppose a new video that initially is unpopular. In this case, when a request arrives CDC will allocate a slot buffer, and because the prefix is not in the cache, the video server will have to send the video stream from the beginning. But unpopular videos may receive two requests with a small interval. To chain these request without the need for a patch, the initial content, the prefix, may not be discarded if the proxy has enough memory.

5. EXPERIMENTAL EVALUATION

Using the NS-2 simulator [3] we evaluated the effectiveness of CDC for delivering video streams on demand. Our simulations used the first-hour trace of the Oliver Stone's motion picture Platoon, with MPEG2 wide-screen format and an average throughput rate of 8 Mbps. We assume that the

distribution network backbone supports up to 1000 video streams between the server and the CDC proxy, which can deliver a single video stream to each connected client. Also, each client has an ADSL-type link with 10 Mbps downlink for MPEG2 video transmission with DVD quality, and 64 kb/s uplink for interaction with the server. We simulated one-hour interval with interarrival rates modeled using a Poisson process [17]. For simulation of multiple videos, clients choose videos according to a Zipf distribution [1]. Cache size ranges from 10% to 100% of the total video asset stored at the server.

To evaluate the impact of CDC on streaming proxy performance we simulated three dynamic caching schemes, namely LS-, LS, and LS+. The LS- scheme implements dynamic caching without link slots, so it works like a conventional scheme, in which a different ring buffer is allocated to each new video stream but allows also two overlapping video segments at a temporal distance no greater than the buffer size to merge by joining their associate ring buffers into a single one. The LS scheme like LS- does not use link slots but allows also concatenation of two non-overlapping video segments by chaining its associate ring buffers to form a larger one. LS+ simulates the CDC policy that uses link slots to allow chaining and splitting of video segments dynamically, according to the CDC replacement policy. We compare performance of the three schemes across several performance metrics, including backbone traffic rate, client blocking rate, and average startup latency that clients experiment.

Initially, we ran the simulations to determine the slot size and the pre-prefix size. Next, we ran three series of simulations. First, we compare performance of the three schemes with a single video and next with 100 videos and Zipf parameter fixed at 0.271. Finally, we simulated CDC LS+ with different Zipf parameters and 100 videos. All simulations were executed 30 times to achieve at least an interval confidence of 95%.

5.1. Slot size considerations

We simulate the CDC with slots of 4, 8, 16, 32 and 64 seconds with one video. We stressed the simulations considering a cache size with only 10% of the total video size and 1000 clients arriving in a hour. As most research work in this area we assumed that the jitter is zero [13], i.e, we eliminate jitter considerations to establishing link slot size. Of course, greater jitter needs greater slot buffers, but the slot buffer size is not done in terms of slot size, as we will see next, but in terms of slot quantity. For instance, for a greater jitter the slot buffer can be augmented with two slots in the WS sector of the slot buffer, but these considerations are outside of the scope of this work.

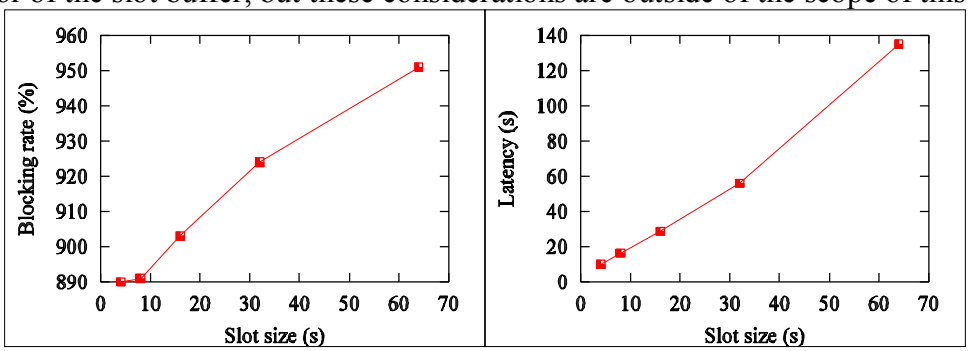

Figure 8. (a) Block rate x Slot size and (b) Latency x Slot size

As we can see in the Figure 8-a, the blocking rate for slot size of 4 seconds and 8 seconds almost remain constant and after 16 seconds it starts to grow almost linearly. This fact can be more understandable if we compare it with traditional virtual memory paginations scheme. Suppose that requests are words in a page memory and the interval between requests are the offset of one word to another. Clearly most of the page content is not used. As page size grows, the probability that a word in this page will be requested grows and hence we waste a lot of memory space keeping a

huge amount of large pages with few requests, i.e., using few words in the page. On contrary, as page size decrease, the probability that a word in this page, within an application with this behavior, reduce and hence we need to keep in memory fewer pages with smaller size. Also, as slot size grows, the latency grows linearly (Figure 8-b). In fact, slot act as a batching policy where clients are grouped to be served. With greater slots, one can wait more time to close the group and consequently the latency is greater.

We choose slots of 8 seconds because the block rate until 8 seconds slot size almost stays constant and an average latency of 16 seconds is not a great penalty to the user. Besides, with smaller slots, consider a slot of 2 seconds for instance with an average of 2 video units per second or 3 video units in 2 seconds, usually found in MPEG2 streams, we have a slot managing a small quantity of video units which is a penalty for CDC system that has to handle more meta information about the video and its state.

5.2. Prefix size considerations

In subsection 4.2, we defined that CDC needs an initial default prefix size in order to chain slot buffers without a patch stream. This default value was determined after detailed simulations that showed that with LS 32 slots sequence we can chain most of the requests with hundred videos and Zipf parameter 0.271, under stressing conditions, i.e, in the prime time (Figure 9). If the second request arrives before 32 slots times, and the proxy do not experiments memory shortage, a new slot buffer is allocated and chained with the previous one with LS slots, and because the prefix is in the cache, the LS slots does not need to be feed with a patch from the server. But if the request arrives after 32 slots and the proxy request memory space, this prefix will be discarded. Also, if the proxy has a severe shortage of memory, the initial part of the prefix (the pre-prefix), which is before the first slot buffer and has the maximum size of the 32 slots, is discarded, from the bigger to the smaller. Bigger pre-prefix means unpopular and smaller more popular videos. Obviously, the pre-prefix is discarded only if CDC has no more LS slots to be the victim of the replacement algorithm. For simplicity, this threshold, maximum of 32 slots, apply to any LS slots sequence.

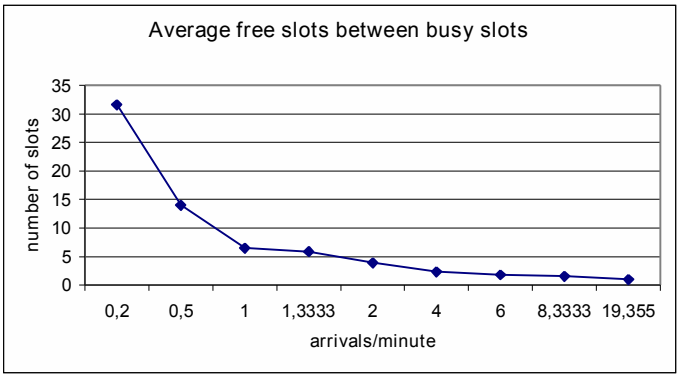

Figure 9. Distance between requests in number of slots

5.3. Experiments with a Single Video

Figures 10 to 13 show the behavior of the three schemes under different cache sizes and inter-arrival times. For small cache sizes below 30%, the LS scheme clearly traded low backbone traffic rates and peak bandwidth rate for high client blocking rates as shown in Figures 10, 11 and 12. A comparison of client blocking rates (Figure 10(a) and (c)) and backbone traffic rates (Figure 11(a) and (c)) reveal that LS- and LS+ perform closely, except for small cache sizes and large inter-arrival times (above 100 seconds), where LS+ scheme performed best. Notably, LS+ performed

significantly better than the other schemes under unfavorable conditions, i.e., for small cache size and unpopular videos. The average latency startup remains low (Figure 13), between 15 and 25 seconds. Next, we further analyze CVC LS+ behavior at such conditions.

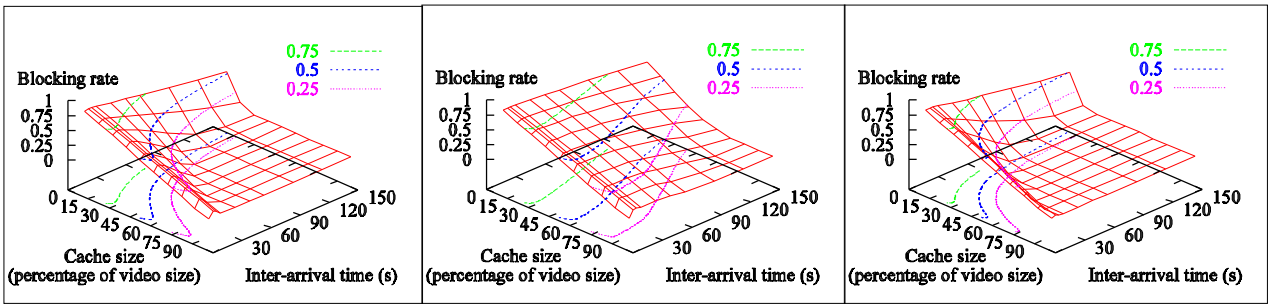

Figure 10. (Single Video) Blocking rate : (a) LS- (b) LS, and (c) LS+

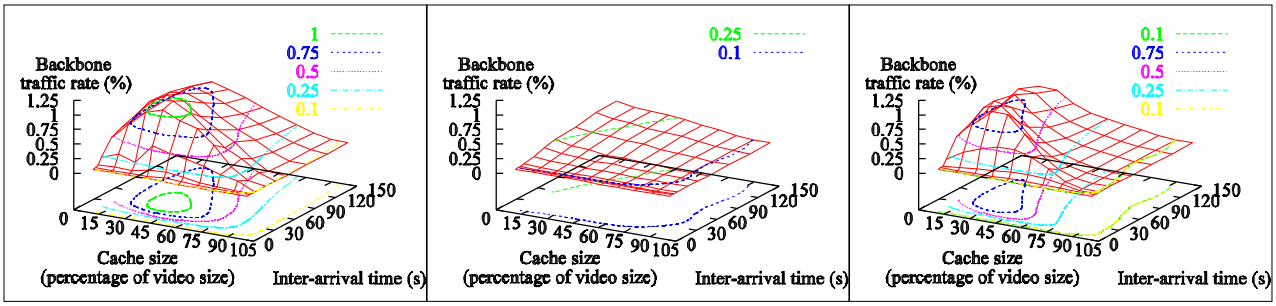

Figure11. (Single Video) Backbone traffic rate: (a) LS-, (b) LS, and (c) LS+

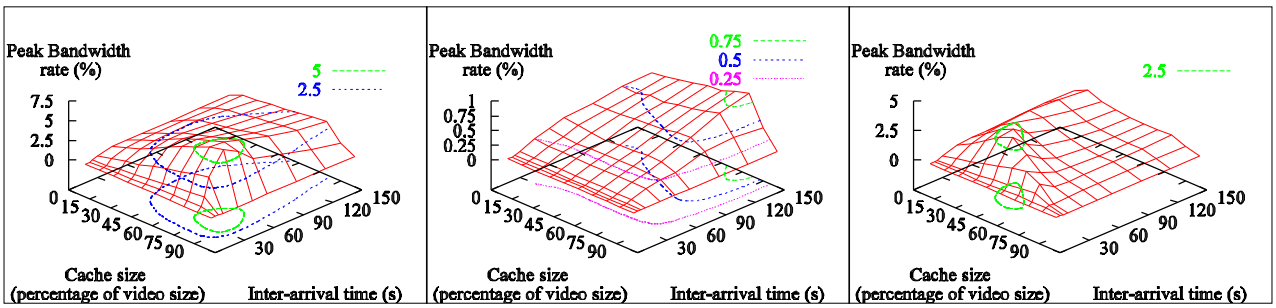

Figure 12. (Single Video) Peak bandwidth traffic rate: (a) LS-, (b) LS, and (c) LS+

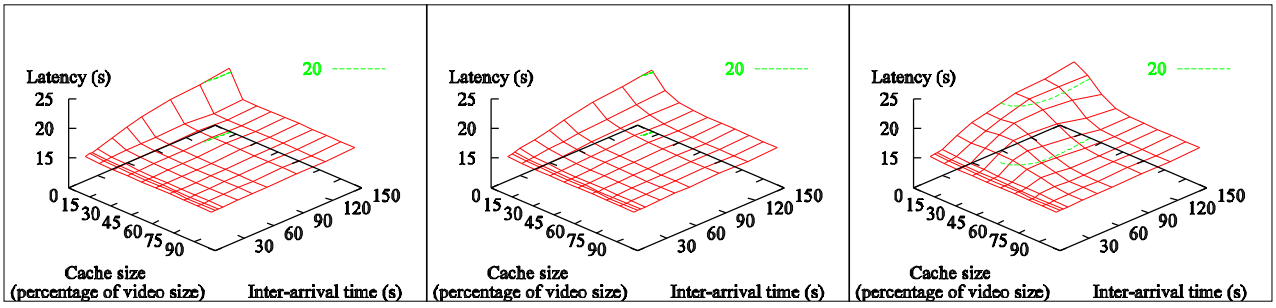

Figure 13. (Single Video) Average startup latency: (a) LS-, (b) LS, and (c) LS+

5.3.1. Client Blocking Rate for 20% Cache Size

Figures 14 (a) and 15 (a) show the results we obtained for the three schemes under restraints conditions. In Figure 14 (a), LS+ blocking rate is significantly better than that of LS and LS-, for all inter-arrival time rates and for a cache size of 20% of total video asset. Also, all three schemes present high blocking rates for inter-arrival rates below 10s, since the cache becomes quickly full due to the relatively small amount of collapsed buffers that can be allocated, which prevents admission of new clients as well as creation of new link slots. Figure 15 (a) shows client blocking rates for inter-arrival time of 150 s, confirming that LS+ also outperforms both LS- and LS for larger cache sizes up to 30% of video size at which the blocking rate drops to zero, except for LS. This result can be explained by the different ways LS+ ,LS , and LS- deal with link slots and the video prefix. While LS- does not use link slots, LS must hold the contents of an allocated link slot in cache until the video ends whereas the LS+ replacement policy is able to discard link slots in order to allocate new collapsed buffers. Also, LS- and LS use video prefixes with constant size whereas LS+ varies prefix size according to the availability of cache space, because it can afford to release link slots. Thus, on releasing link slots that contain video prefixes, LS+ in fact reduces video prefix size dynamically, which leads to more effective use of the streaming cache, which in turn reduces the blocking rate. As the cache size increases, the blocking rates are reduced significantly because more collapsed buffers can be allocated. Similarly, as the inter-arrival time rate grows the client blocking rate diminishes since fewer clients in average arrive within the same time interval.

Also, LS shows higher blocking rates than LS+ even for large cache sizes. The reason is that the LS version creates link slots that cannot be released once they are reserved and thus the cache cannot reuse the associate slots for creating new collapsed buffers. Therefore, new requests had to be discarded, which increased the client-blocking rate. In contrast, LS+ curves show that the client's blocking rate is significantly reduced as the inter-arrival time between requests increases, because LS+ could release link slots for establishing new collapsed buffers.

5.3.2. Backbone Traffic Rate for Interarrival Time Rate=150 s

Figure 14(b) presents backbone traffic rates for a cache size of 20% of video size. For low inter-arrival time (below 10 seconds) all three policies produced low and equivalent backbone traffic rates. The reason is that at this average inter-arrival time rate the system does not use link slots and the three policies performed similarly. After 10 s, LS presented low backbone traffic rate. The reason is that LS spent cache memory chaining slot buffers which in turn allowed the use of less backbone resources despite of attending fewer clients as LS's high client blocking rate (Figure 14 (a)) indicates.

Between 10 s and 60 s, LS- backbone traffic rate was slightly lower than that of LS+ (Figure 14(b)). This occurred because within this time interval almost all slot buffers were chained. The performance difference comes from the fact that LS+ discarded the prefix to allocate more slot buffers, which increased the backbone traffic rate while lowering the client blocking rate, causing LS+ performance being superior to that of LS- and LS.

For interarrival time rates larger than 60 s, LS+ started to use link slots more frequently and hence less regular channels, and thus performed better than LS-.

Figure 15(b) shows that LS+ outperformed LS- for interarrival time rate of 150 s and for practically all cache sizes. For small caches (10% or less) and large caches (90% or above) the resulting performance for the three policies are equivalent because there is no room for improving cache management.

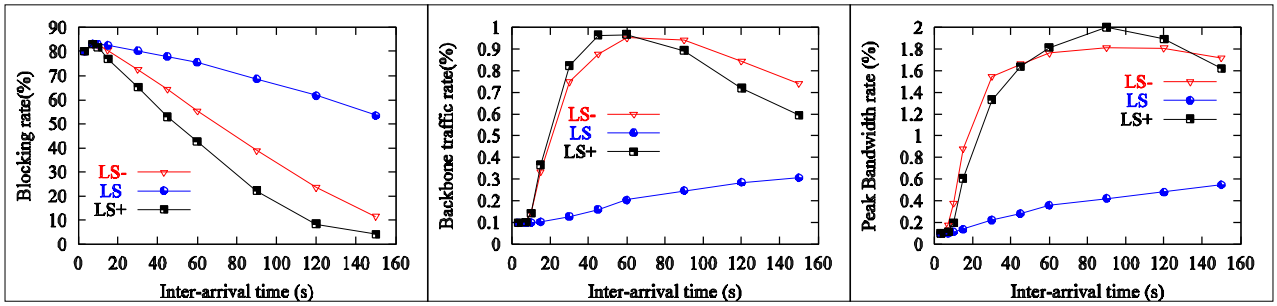

Figure14. Single Video, cache size=20%: (a) Blocking rate, (b) Backbone traffic rate, and (c) Peak bandwidth

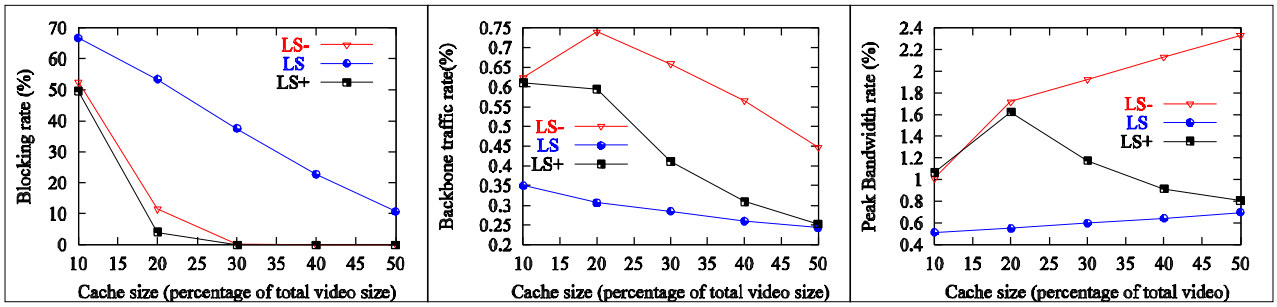

Figure 15. Single Video, interarrival time=150 s: (a) Blocking rate, (b) Backbone traffic rate, and (c) Peak bandwidth

5.3.3 Peak Bandwidth Requirement

In addition to the average backbone bandwidth required to support the backbone traffic rate, it is important to measure the peak backbone bandwidth that each of the three schemes demanded to the Content Distribution Network.

Figure 14(c) shows that there is little difference between the three policies for inter-arrival time rates less than 10 s. Above this rate, LS produced lower peak backbone bandwidth because it attended fewer clients and it did not need patches. Note that LS+ requests patches whenever it replaces link slot sequences by regular video channels from the server. These patches correspond to the missing parts that are located between the regular channel and the actual video content in the cache. Thus, between 10s and 45 s, LS+ replaced link slots to allocate more slots buffers, and above 45 s, with fewer clients arriving at the system, it did not need to release link slots and hence it did not use patch channels frequently

Figure 15(c) shows that the maximum amount of channels that LS- required was on average higher than that of LS and LS+, since LS- had no link slots to replace server's channels. In LS, the peak backbone bandwidth was reduced drastically, because initial requests could be chained until the cache was full, at which all of the remaining requests were discarded. Despite of using the lowest backbone bandwidth, LS's high blocking rate turned it less effective. Last, LS+ produced a peak bandwidth significantly lower than that of LS-, because LS+ used link slots to replace server's video channels. Therefore, LS+ offered a compromise solution that is superior to that of LS-: lower blocking rate, comparable average backbone traffic rate, and lower peak backbone bandwidth.

5.3.4 Average Start-up Latency

Figure 16 show the graphics for latency, and as we might expect, the three schemes produced low average startup latency between 16 and 21 seconds. The reason is that the three schemes tend to maintain the video prefix in cache, and so their average startup latencies were similar.

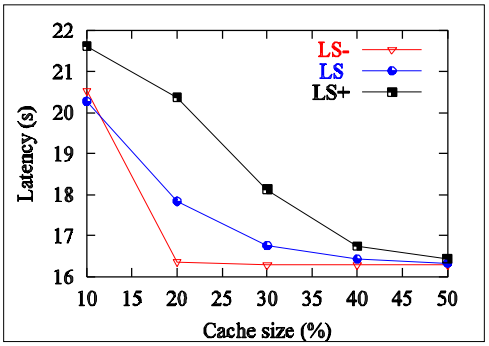

Figure 16. Average startup latency for a Single Video, inter-arrival time=150 s.

5.4. Experiments with 100 videos

We evaluate performance of the three CDC schemes under stressing conditions, i.e, we simulated 1000 client arrivals within one hour interval. Clients selected videos from a collection of 100 videos according to the Zipf distribution with parameter=0.271 [1]. Figures 17 and 18 (a) through (d) show client blocking rates, backbone traffic rates, peak bandwidth and startup latency, respectively for each dynamic caching scheme. Note that the cache size is a percentage of the total amount of videos stored at the server, so there is no direct relationship with the case of a single video. For instance, the 20% cache size is shared among the 100 videos.

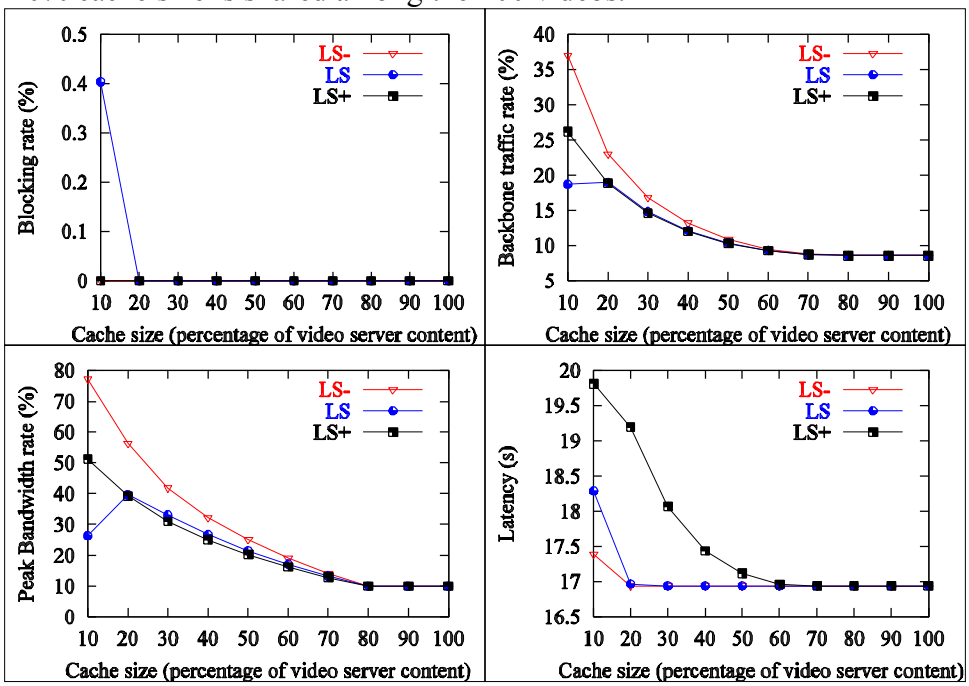

Figure 17. 100 Videos and inter-arrival time of 3.1 seconds: (a-up left) Blocking rate, (b-up right) Backbone traffic rate, (c-down left) peak bandwidth and (d-down right) startup latency.

Figure 17 shows that LS+ performance offered lower blocking rates than that of LS and less backbone traffic rates than that of LS-, for a very small cache size (10%) and at a rush time with an average inter-arrival time of 3.1 seconds. For greater cache size, with this arrival rate, LS+ and LS have the same performance because the system does not experiment shortage of resources. LS+ is effective under shortage of resources. This observation is confirmed for inter-arrival time of 2.5

seconds (Figure 18), where the cache memory space shortage increases, due to the arrival of more clients to the system. In this case, LS+ is better than LS with cache sizes below 20%

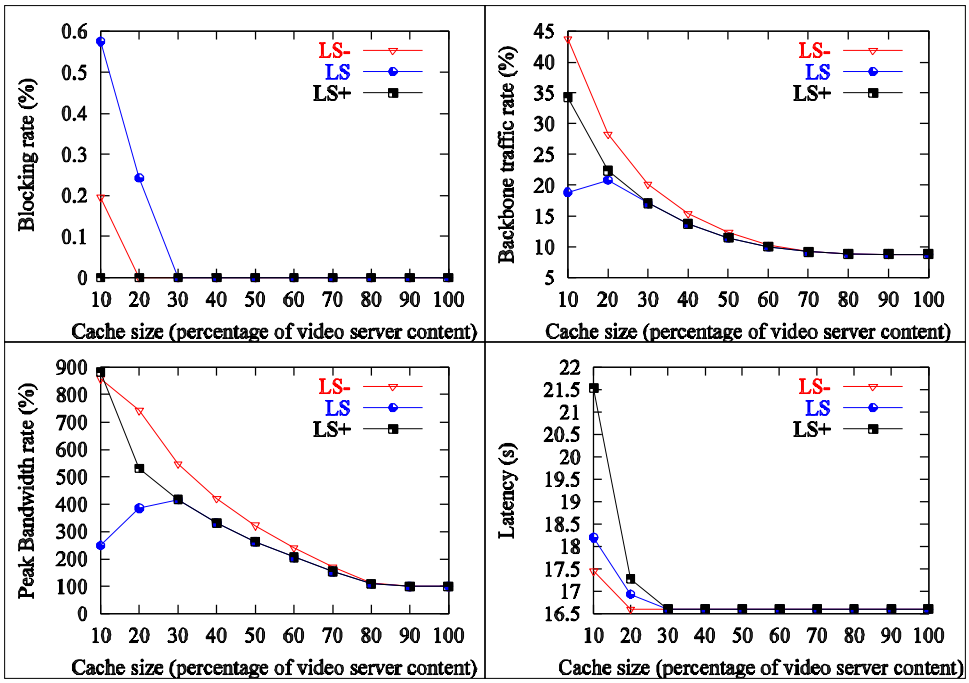

Figure 18. 100 Videos and inter-arrival time of 2.5 seconds: (a-up left) Blocking rate, (b-up right) Backbone traffic rate, (c-down left) peak bandwidth and (d-down right) startup latency.

The start-up latencies for the three policies were low due to the same reason explained previously for the case of a single video.

Results of sensitivity analysis of LS+ performance for two other Zipf parameters are presented in Figure 19(a) and 1b(b). Specifically, they show LS+ with Zipf parameter 0, which has higher skew distribution, and LS+ with Zipf parameter 1, which corresponds to a uniform distribution of video requests to the server. Not surprisingly, CDC LS+ obtained the best performance for the higher skew distribution.

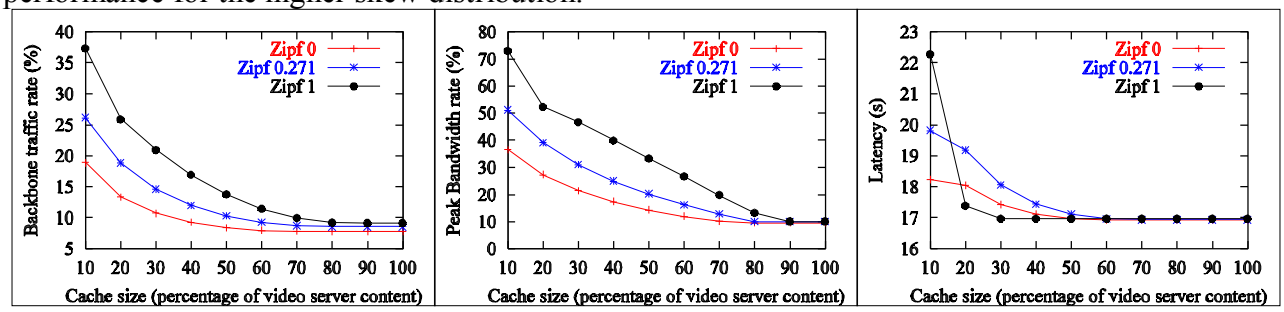

Figure 19. 100 Video and 1000 clients: (a) Backbone traffic rate, (b) peak bandwidth and (c) average startup latency.

6. RELATED WORKS

Chan and Tobagi [5] showed that distributed streaming proxies that support ring buffers of variable size can implement VoD services at low cost. Verscheure [14] proposed ring buffers combined with scalable streaming techniques to guarantee QoS. Also, ring buffers are combined with video prefixes stored previously in the cache to improve proxy performance. Venkatramani [13] extended

the work in [14] and showed analytically how to determine the optimal working set of prefix assets as well as the optimal prefix size to improve performance of caching systems in streaming proxies. Bommaiah's work [2] used ring buffers similarly to that proposed in [13,14] and supported the idea of increasing the size of ring buffer while a video plays back so that the same ring buffer can serve other requests. Also, a best effort service is assumed, since the ring buffer is allocated at a proxy only if memory space is available, otherwise the client will be served from the local disk, using a static caching replacement policy. Chen [18] proposed a more drastic replacement policy when there is no available memory and the proxy has to allocate a new ring buffer for a highly popular video. In this case, a busy ring buffer isl released and its associate client is left without QoS. Furthermore, this work assumes that clients have enough bandwidth to receive multiple video streams simultaneously. So far, these works use dynamic cache algorithms that resemble the LS policy whereas LS- is a special case of LS in which the maximum size of a sequence of link slots is zero. In contrast to previous works CDC introduces the LS+ policy.

CDC replacement algorithm for video segments resembles GreedyDual [19, 7]. In fact, like GreedyDual, our replacement algorithm uses information about both memory space and cost to transmit a video stream until the video ends. Young [20] shows that the GreedyDual algorithm is online optimal in terms of its competitive ratio (cost/size) for static web caching.

The behavior of CDC algorithm simulates the Gradient-descent prefix allocation algorithm as proposed by Venkatramani [13] that determines the optimal working set of prefix asset and optimal prefix size. However, this algorithm has to be ran periodically to determine the prefixes that need to be stored in the cache [14]. In contrast, CDC with LS+ adapts dynamically the video prefix size and the working set of prefixes. In [8] we proof that our management scheme is near optimal since CDC performs similarly to the gradient descendent-based algorithm, but dynamically. Although ring buffers are usually allocated in primary memory, CDC may be managed as a virtual memory scheme [10]. For instance, only contents under RS slots need to be stored in primary memory, and the contents of other slot types may be paginated to secondary memory. In a practical implementation of CDC this fact must be considered, since secondary memory space is larger and less costly than main memory.

7. CONCLUDING REMARKS

In this work, we introduced and evaluated the Collapsed Cooperative Video Cache (CDC), a dynamic caching mechanism that enables ring buffers to perform streaming operations on video segments such as splitting and chaining, while implementing near optimal cache replacement policy.

Our experimental results showed that CDC outperformed existing schemes across several performance metrics such as client blocking rate, backbone traffic rate, peak bandwidth, and average startup latency for different parameter values of cache size and interarrival time rates Therefore, our main conclusion is that CDC is an effective option to build highly scalable streaming proxies in general, and for designing scalable VoD systems in particular.

Research efforts are underway to explore multiple CDC proxies with LS+ as well as studying performance impact of using VCR-like interactive services on scalable VoD systems.

8. ACKNOWLEDGMENTS

The authors would like to thank Brazilian Research Agencies: Finep, Capes, and CNPq, for supporting this research work. Also we are grateful for the helpful assistance of Leonardo Bidese de Pinho from the Parallel Computing Laboratory for his support on running time consuming simulations in a Beowulf cluster.

9. REFERENCES

[1] A.S. Dan and S.P. Shahabuddin, "Scheduling policies for an on-demand video server with batching", IBM Research Report RC 19381, 1994.

[2] E. Bommaiah, K. Guo, M. Hofmann and S. Paul, "Design and Implementation of Caching System for Streaming Media over the Internet", IEEE Real Time Tech. and App. Symposium, Washington, DC, May, 2000.

[3] L. Breslau, et al. (2000) Advances in Network Simulation, IEEE Computer, 33(5), p.p. 59-67, May.

[4] Y. Chae et al., "Silo, Rainbow, and Caching Token: Schemes for Scalable, Fault Tolerant Stream Caching", IEEE Journal on Sel. Areas in Communication , 2002.

[5] S.-H. G. Chan and F. Tobagi, "Distributed Servers Architecture for Networked Video Services", IEEE/ACM Trans. on Networking, vol. 9, No. 2., April 2001.

[6] K. A. Hua, Y. Drops and S. Sheu, "Patching: The multicast technique goes true video-on-demand services", ACM Multimedia´98, Bristol, England, September, 1998.

[7] S. Jin and A. Bestavros, "GreedyDual* Web Caching Algorithm: exploiting the two sources of temporal locality in Web request streams", Computer Communications 24 (2): 174-183 (2001).

[8] E. Ishikawa and C.L. Amorim, "Near Optimal Video Cache Management in Streaming Proxies", Federal University of Rio de Janeiro, Tech. Report ES-610/03, August 2003.

[9] S. Chen, B.Shen, S.Wee and X. Zhang, "Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery", NOSSDAV´03, Monterrey, California, USA, June 2003.

[10] J. Winnograd, S.J. Morganstein and R.Hreman, "Simulation studies of a virtual memory, time shared, demand paging algorithm", proceedings of the Third ACM symposium on Operating Systems Principles, pp.149-155, October 1971.

[11] S. Sen, J. Rexford and D. Towsley, "Proxy prefix caching for multimedia stream", IEEE Infocom´99, New York,1999.

[12] S. Sheu, K.A. Hua and W. Tavanapong, "Chaining: The generalized batching technique for video-on-demand systems", IEEE Intl Conference on Multimedia Computing and Systems, Ottawa, Canada, 1997.

[13] C. Venkatramani, O. Verscheure, P. Frossard and K.-W. Lee, "Optimal Proxy Management for Multimedia Streaming in Content Distribution Networks", NOSSDAV´02, May 12-14, 2002, Miami, Florida, USA.

[14] Verscheure, O., Venkatramini, C., Frossard, P. and Amini, L., Joint server scheduling and proxy caching for video delivery, Computer Communications 25, 2002, pp 413-423.

[15] T.H. Comer, C.E. Leiserson and R.L. Rivest, "Introduction to Algorithms", pp 149-151, McGraw-Hill and MIT Press, 1990.

[16] M. R. Garey and D.S. Johnson, "Computer and Intractability, A Guide to the Theory of NP-Completeness", p. 227, W. H. Freeman and Company, USA, 1997.

[17] E. Veloso, V. Almeida, W. Meira Jr., A. Bestavros, and S. Jin, "A Hierarchical Characterization of a Live Streaming Media Workload", In Proc of ACM SIGCOMM Internet Measurement Workshop 2002, Marseille, França, November, 2002.

[18] S. Chen, B. Shen, Y. Yan, and S. Basu, "Shared Running Buffer Based Proxy Caching of Streaming Sessions", Technical Report HPL-2003-47, Mobile and Media Systems Laboratories, HP Laboratories Palo Alto, March 2003.

[19] P. Cao and S. Irani, "Costa-aware WWW proxy caching algorithms", Proceedings of the 1997 USENIX Symposium in Internet Technology and Systems, December 1997.

[20] N.E. Young, "On-Line File Caching", in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 82--86, January 1998.