Smoothing and Prefetching Video from Distributed Servers

Spiridon Bakiras and Victor O.K. Li
The University of Hong Kong
Department of Electrical & Electronic Engineering
Pokfulam Road
Hong Kong
email: {sbakiras,vli}@eee.hku.hk

Abstract

Video prefetching has been proposed recently for the transmission of variable-bit-rate (VBR) video over a packet-switched network. The objective of these protocols is to prefetch future frames to be stored at the customer's set-top box (STB) in periods of low link utilization. Experimental results have shown that video prefetching is very effective and it achieves much higher network utilization (i.e. larger number of simultaneous connections) than the traditional video smoothing schemes. Video prefetching, however, can only be efficiently implemented when there is one centralized server that serves the different customers over a common link. In a distributed environment there is a large degradation in its performance. In this paper we introduce a new scheme that utilizes smoothing along with prefetching, to overcome the problem of distributed prefetching. We will show that our scheme performs almost as well as the centralized prefetching protocol even though it is implemented in a distributed environment.

1. Introduction

The development of high-speed networks and the tremendous progress made on compression techniques have made applications like Video-on-Demand (VoD) a reality. A VoD system aims to provide video rental services to the customer, with the advantage that the customer does not have to leave home. In a true VoD system, the user will be able to select any movie from a video server and view it at any time. In addition, it will allow the user to perform any VCR-like function such as pause, fast forward, jump forward [2].

For a VoD system to be cost effective, many issues have to be considered. One of the most important issues when designing a VoD network is the allocation of bandwidth to the various video sequences. Video is typically compressed with the Motion Picture Experts Group (MPEG) format. Since MPEG video traffic is very bursty and exhibits large variability in the output bit-rate, researchers have tried to find effective methods to reduce this burstiness. Most of the research reported in the literature focuses on the use of a buffer in the customer's set-top box (STB) to smooth the video traffic and allow for constant-bit-rate (CBR) transmission [3, 7, 8]. In [7] an optimal smoothing algorithm is given which can reduce the peak rate by 75-87%, using a 1MByte buffer at the STB. The optimality refers to the reduction of the rate variability of the transmission schedule. The basic idea is to run the smoothing algorithm on every stored video, and get an optimal transmission schedule (i.e. different CBR transmission rates at different times) which is then used for the transmission of the video. This scheme can provide deterministic Quality of Service (QoS) guarantees (in terms of loss probability) if the bandwidth is allocated to satisfy the largest rate of the transmission schedule. In [8] the optimal smoothing algorithm is considered together with statistical multiplexing. The authors use the same algorithm, and they investigate its performance when some small loss is allowed. This scheme will provide only statistical QoS guarantees, but it increases dramatically the system utilization.

Recently, some work has also been done based on the idea of prefetching [4, 5]. These protocols use variable-bit-rate (VBR) transmission and can only provide statistical QoS guarantees. In [4] a protocol called Join-the-Shortest-Queue (JSQ) prefetching is presented which has many advantages compared to the other transmission schemes in the literature. It achieves very high network utilization, facilitates user interactions, and has minimal start-up latency. Their experimental results showed that JSQ prefetching had a loss probability several orders of magnitude smaller than optimal smoothing [8] for the same buffer size and network utilization. The main idea is to put a buffer in the customer's STB which can be used to prefetch future frames when the transmission link is under utilized. The frames are

prefetched in a way that all the ongoing connections have similar number of prefetched frames so as to minimize the loss probability. This large performance gap between optimal smoothing and JSQ prefetching is due to the different usage of the STB buffer [4]. In a smoothing scheme the buffer is used to reduce the peak rate and the rate variability of the video sequence, and in some time periods it will be almost empty. The idea of video prefetching, though, is to keep the STB buffers as full as possible at all times, so that in periods where the aggregate bit-rate exceeds the link capacity the user will not experience playback starvation, as there will be some frames buffered at the STB.

JSQ prefetching, however, has a major drawback: it can only be implemented when there is one centralized server which serves different users over a common link, since the prefetching algorithm has to know the exact frame sizes from all ongoing connections in advance. In a real system, though, the different movies may be distributed over several servers and, in addtion, there may be more than one VoD service provider who operate in the same area. In [5] a decentralized version is introduced which allows prefetching when there are many distributed servers in the system. Each server keeps a send window which is the number of frames that it is allowed to send in one frame period. The value of the send window is increased when the sent frames are acknowledged by the user, and it is set to one when frames are dropped. When a frame is dropped, it will be retransmitted from the server if the corresponding client has one or more frames buffered at the STB. This scheme works well compared to other VBR schemes, but its loss probability is more than two orders of magnitude bigger than JSQ prefetching for the same buffer size and link utilization, since prefetching is performed for each connection independently of the others. In addition, since there is no coordination between the different servers, this protocol will result in unnecessary retransmission of frames which will increase the traffic load of the backbone network.

In this paper, we introduce a VBR transmission scheme that utilizes smoothing along with prefetching to reduce the bandwidth requirements of MPEG traffic. The idea is to first smooth the MPEG traces over a few group of pictures (GOPs) and then use a central controller to coordinate the transmission of frames from all the servers. This coordination is possible since the traffic from each connection (video) is constant for a period of time equal to the number of smoothed GOPs (one GOP is normally 12 or 15 frames). It should be noted that smoothing is not performed in order to reduce the peak rate of the video sequence, as in the typical video smoothing schemes. In our protocol we smooth each video sequence so as to keep the bit-rate of each connection constant for a small period of time. In order to evaluate the effectiveness of our scheme, we used 10 real MPEG-1 traces [6] with different contents (e.g. music, movie, news) for our simulations. We compared our proposed scheme with JSQ prefetching and the results indicate that our scheme performs almost as well as JSQ prefetching even though it is implemented in a system with distributed servers.

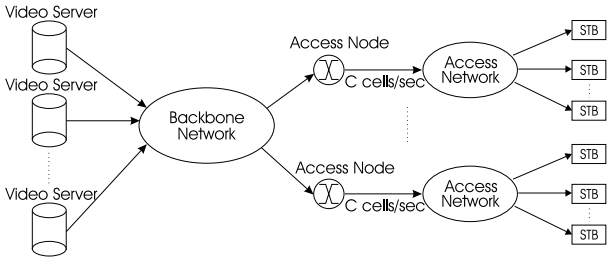

Figure 1. The VoD network architecture.

The rest of the paper is organized as follows. In Section 2 we describe the VoD network architecture that is considered throughout this paper, while in Section 3 we introduce our proposed scheme. Section 4 presents our numerical results, and Section 5 concludes our work.

2. The VoD network architecture

Let us consider the VoD network architecture of Fig. 1. It consists of the video servers, the backbone network, the access nodes, the access networks, and the users' STBs with the prefetch buffers. All the users are connected to the backbone network through the access network which may, for example, be the existing cable TV network, the telephone network or a wireless network. Multiple access networks will be connected to the backbone network. In order to maximize the number of users in the system, we should find a way to maximize the number of users served simultaneously in each access network (assuming of course that the video servers and the backbone network can support all of them). Therefore, in this paper we will focus on one such access network. We assume that the backbone network is an ATM network, and that the maximum capacity that can be accommodated by the access network is C cells/sec. Our scheme, however, is applicable to any type of packet-switched network. We can, therefore, consider that there is one common transmission link with capacity C, connecting all users to the access node.

When a user initiates a request for a particular movie, the access node will decide whether to accept the request or not. We are currently working on the development of a call admission control algorithm for video prefetching, but in this paper we will not perform admission control. The simulation experiments will be planned to achieve a 90% utilization on the common transmission link. If the request is accepted, the access node will forward the request to the

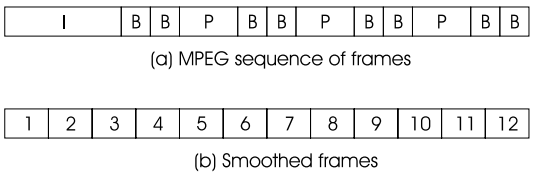

Figure 2. Smoothing of one GOP at the video server.

server, and if the server can accommodate the request the connection will be established. At this point, the server will start retrieving the MPEG frames from the disks. These frames will be sent to the access node through the ATM network, and then the access node will forward them to the user's STB for display. We assume that the access node is bufferless, that is, all cells that exceed the link capacity C are dropped.

Before going any further, we should describe briefly the structure and the types of frames of an MPEG sequence. There are three types of frames generated by an MPEG encoder: intraframes (I), predictive frames (P), and bi-directional frames (B) [1]. The I-frames are coded independently of other frames, and for that reason they are used for random access. The P-frames are coded with respect to a previous I/P-frame, so in general they are smaller than I-frames. Finally, the B-frames are coded with respect to a previous and a future I/P-frame. B-frames are usually much smaller than I or P-frames. A number of frames, typically 12 or 15, are grouped together to form a group of pictures (GOP). The structure of the GOP has a regular pattern, for example, $IBBPBBPBBPBB$. The GOP is defined by two parameters: the number of frames, q, and the number of B-frames between two consecutive I/P-frames, $l-1$. In the above example, $q=12$ and $l=3$.

In our scheme we smooth the traffic from k GOPs before sending it to the ATM network. So in every $k \cdot q$ consecutive frame periods, the same amount of data, which is equal to the average of the $k \cdot q$ frames, is sent from the server to the ATM network. This is illustrated in Fig. 2, for the case where $k=1$, $q=12$ and $l=3$. From this point on when we say *frame*, we will refer to a smoothed frame and not to a whole frame of the MPEG sequence. Since we smooth $k \cdot q$ MPEG frames at each time, we should always have some frames buffered at the STB in order to avoid playback starvation. The I-frame of each GOP, for example, will be sent in parts during a few frame periods, and at the point of its display a few frames will be removed from the STB buffer. For this reason, we also need to preload the buffer with a few frames prior to the beginning of the playback (start-up latency). In order to guarantee no playback starvation, we should always keep $k \cdot q$ frames in the STB buffer. However, the structure of the GOP allows us to keep less frames buffered at the STB. This is possible since every I or P-frame is followed by $l-1$ B-frames which are very small and may be included in less than one frame. So when the B-frames are displayed, the level of the buffer is increased rather than decreased, since video data is arriving at the buffer faster than it is being retrieved (the reverse occurs when an I-frame is displayed).

3. The proposed scheme

We will now present in detail our proposed prefetching protocol. It is based on the idea of a central controller that coordinates the transmission of frames from the different servers. The access node (Fig. 1) will be the location of the central controller, since all the video servers are connected to it through the backbone network. This coordination is possible because of the smoothing that results in a constant level of traffic from each connection for a few frame periods. We will now introduce the following variables associated with each video connection.

- k: number of GOPs to be smoothed.
- f: frame rate (e.g. 24 frames/sec).
- q: GOP size.
- m_i: minimum number of frames that should be buffered at the STB at all times, in order to avoid playback starvation for customer i (as explained in the previous Section).
- p_i: index showing which frame of the smoothed GOP is currently being transmitted to customer i. It takes the values 1, 2 ,. . ., $k \cdot q$.
- b_i: current buffer level for customer i.
- B_i: maximum buffer size for customer i. For simplicity we assume that all customers have the same buffer size.
- s_i: number of frames buffered for customer i.
- a_i: number of prefetched frames for customer i. It is equal to $\max\{s_i - m_i, 0\}$.
- r_i: maximum number of frames that can be sent in the following frame period to customer i.
- l_i: number of frames to be sent in the following frame period to customer i.
- t_i: size of the frame transmitted to customer i in the current frame period.

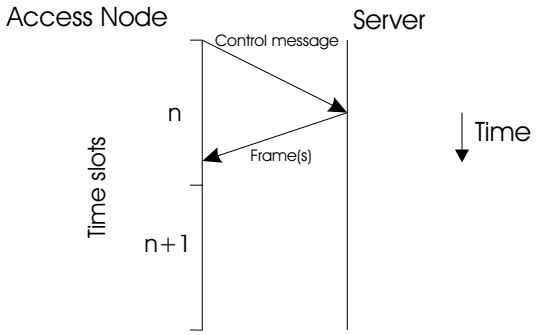

Figure 3. Time slot synchronization between a server and the central controller.

Let us call a *time slot* the time period corresponding to one frame which is equal to $1/f$ seconds. During this time slot, the central controller will transmit the frames from the different servers to the clients, until the link capacity C is reached. If some frames can not be transmitted in the current time slot, they will be discarded. A discarded frame will be retransmitted from the server if the corresponding connection has $a_i \geq 1$, which means that this connection has some frames buffered at the STB.

Since the frames will be transmitted from the controller to the clients according to the controller's own discrete time slots, there must be some kind of synchronization between all servers and the controller. We will assume that the round trip propagation and processing delay between any server and the access node is less than a time slot (i.e. $1/f$ seconds). The controller will coordinate the transmission of frames from all the video servers by sending control messages to them. The control messages will indicate which frames should be sent for each connection. All the control messages will be sent in the beginning of the current time slot, say n (Fig. 3) and, therefore, the frames from all connections will arrive prior to the beginning of time slot $n+1$. They will, then, be transmitted to the clients within the duration of time slot $n+1$. This will allow the controller to know the exact frame sizes to be transmitted in slot $n+2$ (because of the GOP smoothing), and based on this information it will send the appropriate control messages at the beginning of slot $n+1$. Note, however, that this delay bound does not have to be tight. Even if some frames arrive after the beginning of time slot $n+1$, and the controller will not have the exact frame sizes of the frames to be transmitted in slot $n+1$, the prefetching algorithm can use the corresponding information from the previous transmission (slot n) as an estimate. Due to the high correlation between the frame sizes of consecutive GOPs, the impact of this approximation on the performance of our scheme will be insignificant.

Time slot:	n	n+1	n+2	n+3	n+4	·····						
Video 1	1	2	3	4	5	6	7	8	9	10	11	12
Video 2	7	8	9	10	11	12	1	2	3	4	5	6
Video 3	10	11	12	1	2	3	4	5	6	7	8	9
Video 4	12	1	2	3	4	5	6	7	8	9	10	11
Video 5	4	5	6	7	8	9	10	11	12	1	2	3

Figure 4. Transmission schedule without prefetching.

3.1. The prefetching algorithm

Let us consider the transmission schedule of five videos shown in Fig. 4 where we assume that we smooth one GOP ($k = 1$) with $q = 12$, prior to the transmission. This corresponds to the frames that would be transmitted from the access node to the customers if no coordination took place. The numbers in each box are the p_i's for the different connections. Therefore, in time slot n, the first frame of the smoothed GOP would be transmitted to customer 1, the seventh frame to customer 2, and so on. By the beginning of slot n the controller will already know the frame sizes for all five connections, and it will use this information to coordinate the transmission of frames from the video servers for the following time slot.

Since we want all connections to have similar number of prefetched frames, the controller will try to prefetch frames from all connections in order of ascending values of a_i (as in the JSQ prefetching protocol). Let us assume for simplicity that all connections in Fig. 4 have $a_i = 0$. The controller will first try to prefetch one frame from connection 1. Since the transmission will take place in time slot $n+1$, the maximum number of frames that can be sent for connection 1 is 11. In the general case, the maximum number of frames that can be sent for one connection is $r_i = \max\{k \cdot q - p_i, 1\}$, since these are the only frames for which we know their exact size. Note that in the case where $p_i = k \cdot q$ the server will only be allowed to send one frame, since the size of the next $k \cdot q$ cells is not known until the first one of them arrives at the controller. Let us call W the estimate of the amount of traffic that will be sent to the access node in time slot $n+1$ (initially $W = 0$). The number of frames to be sent for each connection is set initially to $l_i = 0$. There are two factors that can limit the number of frames to be sent: the maximum buffer size B_i and the link capacity C (in cells/frame period). The controller will check whether the following two conditions hold:

$$b_i + (l_i + 1) \cdot t_i \leq B_i \quad (1)$$

$$W + t_i \leq C \quad (2)$$

Equation (1) tries to avoid buffer overflow while equation (2) checks whether the additional frame will keep the value of the estimate of the total traffic in time slot $n+1$ below the link capacity. In equation (1), we do not consider the MPEG frame that will be removed from the buffer during time slot $n+1$, since it is not possible for the controller to have this information at the time when the prefetching algorithm is executed. We assume, however, that the controller knows the buffer level b_i for each connection (this can be done with control messages from the STBs). If both conditions hold for connection 1, the controller will update the values of l_1, a_1, r_1, and W, and it will continue with connection 2. The algorithm will stop when there is no connection with $r_i > 0$ that satisfies both (1) and (2). When the algorithm terminates, the controller will check whether there is any connection with $a_i = 0$ for which the algorithm returned the value $l_i = 0$. If such a connection is found, l_i will be set to one, since the next frame has to be transmitted in order to meet its deadline. When the values of all l_i's have been updated, the controller will send the control messages to the corresponding servers at the beginning of time slot n. The control messages will be sent only for those connections with $l_i > 0$. The complete prefetching algorithm is presented in Fig. 5.

```
procedure prefetch()
begin
  W := 0;
  for all connections i do
  begin
    r_i := max{k · q − p_i, 1};
    l_i := 0;
    a'_i := a_i;
    e_i :=false;
  end;
  while exists i with e_i=false do
  begin
    find connection i with minimum a'_i;
    if {(1)=true and (2)=true and r_i > 0} do
    begin
      l_i := l_i + 1;
      a'_i := a'_i + 1;
      r_i := r_i − 1;
      W := W + t_i;
    end;
    else
      e_i :=true;
  end;
end.
```

Figure 5. The prefetching algorithm.

3.2. Transmission of frames

The frames are transmitted from the access node to the clients based on a non-preemptive priority scheme. Connections with smaller values of a_i have priority over those with bigger values of a_i. The non-preemptive scheme will allow all the frames from one connection to be transmitted, once it has started the transmission. The value of a_i will be updated by the controller as soon as it receives the frames from all connections, through the following equation

$$a_i = \max\{a_i + f_i - 1, 0\} \tag{3}$$

where f_i is the number of frames that were successfully transmitted to customer i. Since all the frames will arrive by the beginning of the time slot, the controller can easily calculate, based on the priorities and the frame sizes, which frames will be transmitted in the current time slot.

As we mentioned earlier, the value of W used in the prefetching algorithm is an estimate of the total traffic at the time slot where prefetching will occur. However, many connections will probably have frames from different GOPs being transmitted (i.e. the first frame of the next GOP), so the actual amount of traffic in this time slot might be higher or lower than W. If it is higher, then some frames will have to be dropped since W will probably be very close to the link capacity. The transmission order of our protocol will try to drop frames from those connections that have $a_i \geq 1$. If a frame from such connection is dropped, the controller will indicate to the corresponding server, through the next control message, to retransmit the frame.

To conclude the presentation of our protocol, we should describe briefly the operation of the server. Each server remains idle until it receives a control message. When it receives the control message, it will transmit the indicated frame(s). In order to avoid buffer overflow, the server will check the following condition before sending the first frame of a smoothed GOP (i.e. the frame with $p_i = 1$).

$$b_i + t_i \leq B_i \tag{4}$$

This is necessary, since the value of t_i that was used in equation (1) was from the previous smoothed GOP, and its current value could be much higher. If condition (4) holds it will send the frame, otherwise it will remain idle. The buffer level for each connection can be included in the control message, as we have assumed that the controller has this information.

4. Numerical results

In order to evaluate the effectiveness of our scheme, we used 10 real MPEG-1 traces that were downloaded from [6]. They covered a wide variety of contents, including movies, news, talk shows, sports, music, and cartoons. All the traces were captured at $f = 24$ fps and the GOP parameters were $q = 12$ and $l = 3$. The total number of frames for each

Sequence	Mean (bits)	Peak/Mean	m_i (frames) $k=1$	$k=3$
Asterix	22,348	6.6	8	15
ATP Tennis	21,890	8.7	9	18
Mr Bean	17,647	13	8	13
James Bond: Goldfinger	24,308	10.1	6	11
Jurassic Park	13,078	9.1	9	14
Mtv	19,780	12.7	9	15
News	15,358	12.4	10	18
Race	30,749	6.6	6	13
Soccer	25,110	7.6	7	16
Talk show	14,537	7.3	7	12

Table 1. Characteristics of the MPEG-1 compressed video sequences.

trace was 40,000 which is approximately 30 minutes in duration. We assumed that the common link of Fig. 1 is a typical ATM link with capacity 45 Mbps. However, in all of the calculations we used ATM cells instead of bits, so the total available bandwidth was assumed to be $C = 4422$ cells/frame period. Finally, we assumed that all 48 bytes of the ATM payload are used for the transmission of the MPEG data. In Table 1 we have summarized some characteristics of the different MPEG traces. The value of m_i (which is the start-up latency) for each video was easily obtained by simulation.

In each experiment we used a mixture of traces that resulted in a 90% network utilization, that is, the summation of the individual average bit-rates was approximately 90% of the link capacity (45Mbps). This mixture consisted of 7 *Asterix* traces, 8 *Tennis* traces, 10 *Mr Bean* traces, 7 *James Bond* traces, 13 *Jurassic Park* traces, 8 *Mtv* traces, 11 *News* traces, 5 *Race* traces, 7 *Soccer* traces, and 12 *Talk Show* traces, for a total of 88 connections. The experiments were performed as follows. For each connection we chose a random starting point in the movie (the beginning of a GOP), and we started by transmitting one frame from each connection, with all connections having $b_i = 0$. From the next time slot the prefetching algorithm was used to coordinate the transmissions until the end of the experiment. When a connection displayed the last MPEG frame of the movie, the same movie started again from a new random starting point (with $b_i = 0$). But from this point on, we used appropriate wrap around such that each connection displayed exactly 40,000 frames. We simulated 1×10^8 frame periods for several buffer sizes, ranging from 256 to 896 KBytes in increments of 128 KBytes. Before running the experiments we stored the random starting points for all connections, for the whole duration of the experiment, in a file so as to ensure a fair comparison between the different schemes. Finally, we counted the time slots that experienced losses after an initial period of 10,000 frames (to allow the buffers to fill up).

In Fig. 6 we have plotted the loss probability as a function of the buffer size, for the JSQ prefetching protocol and our proposed scheme. We simulated two versions of our protocol for the cases of $k = 1$ and $k = 3$. It is clear that the performance of our protocol is very similar to the performance of the JSQ prefetching protocol. For $k = 1$, the loss probability of our scheme is only one order of magnitude bigger than the loss probability of the JSQ prefetching protocol. In fact, our protocol has the same loss probability as the JSQ prefetching protocol, with a buffer size increment of only 128 KBytes. JSQ prefetching had zero loss for a buffer size of 796 KBytes while, for $k = 1$, our scheme had zero loss for a buffer size of 896 KBytes (in Fig. 6, however, we used the value of 1×10^{-10} to represent zero). The decentralized prefetching protocol described in [5] had a loss probability of more than two orders of magnitude bigger than the JSQ prefetching protocol and, in addition, its loss probability decreased very slowly with increasing buffer size. The performance of our scheme for $k = 3$, is slightly worse than the one for $k = 1$. One would expect that smoothing three GOPs would result in a better performance, since the prefetching algorithm would be more efficient as the traffic is constant for longer periods of time. However, the disadvantage of using $k = 3$ is that we need a larger buffer size to hold the m_i frames (Table 1) at all times, so the performance does not improve compared to the case where $k = 1$. It should be noted that in our experiments we simulated a system which does not support user interactions (e.g. jump forward, fast forward). These interactions will degrade the performance of the system, since all the prefetched frames of the user issuing an interaction request will have to be discarded. We are currently developing an analytical model to investigate the performance of distributed video prefetching in the environment of an inter-

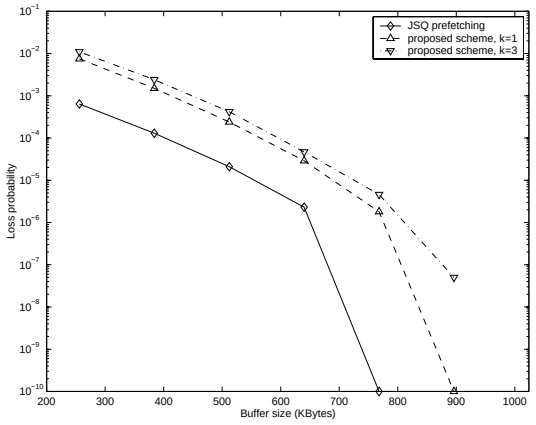

Figure 6. Loss probability for different buffer sizes.

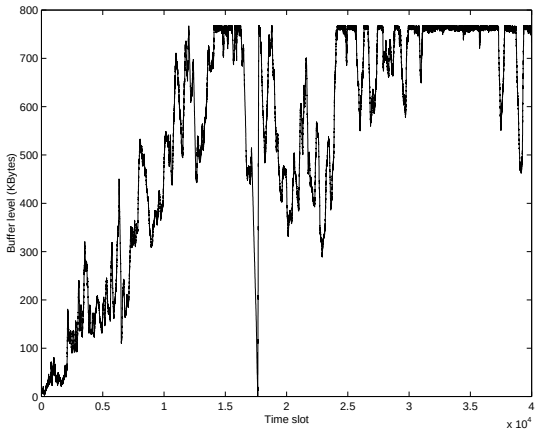

Figure 7. Buffer level for one *Jurassic Park* connection with $B_i = 768$ KBytes ($k = 1$).

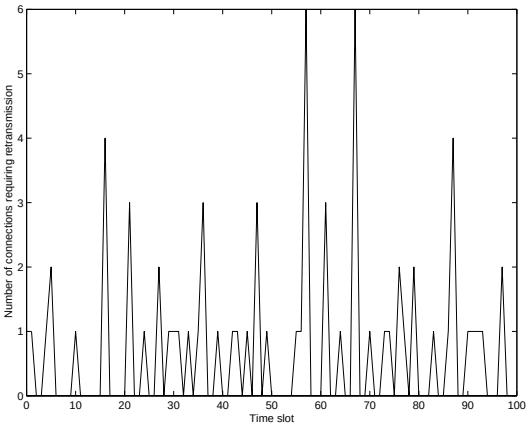

Figure 8. Number of connections requiring retransmission of frames ($k = 1$).

active VoD system.

In Fig. 7 the buffer level for one *Jurassic Park* connection is depicted for the case of $k = 1$, and for a buffer size of 768 KBytes. It shows the buffer level for this connection for a period of 40,000 time slots, starting from the beginning of the experiment. Each downward spike in this figure is an indication of heavy traffic which results in the decrease of the buffer level, since no frames are sent to the particular connection for a few time slots. Since we start the experiment with all the buffers being empty, it takes many time slots until the buffer level reaches its maximum size. When the last MPEG frame for that connection is displayed (around time slot 17,000) the buffer level drops to zero, and the connection starts all over again. In this case, however, the buffer level reaches its maximum size almost instantaneously as this connection has $a_i = 0$ (while the other connections have $a_i \gg 0$) and is given priority by the prefetching algorithm. As a result, even though our scheme requires a start-up latency of around 10 frames, the actual latency will only be a couple of frames. We can, therefore, claim that our proposed protocol has all the advantages of the JSQ prefetching protocol: it achieves very high network utilization (90%), it facilitates user interactions (e.g. temporal jumps), and it has minimal start-up latency.

Finally, in Fig. 8 we have plotted the number of connections that will retransmit some frames as a result of excess traffic at the access node. These retransmissions are caused by the incorrect estimation of W in equation (2). It is obvious that these retransmissions are very rare, and only a few connections (out of the total 88) have their frames dropped at any instant. The decentralized protocol proposed in [5] offers no coordination between the different servers, and it will lead to a lot more retransmissions, thus increasing the traffic load of the backbone network. In addition, the send window used in [5] has a minimum value of one which means that all servers send at least one frame at each frame period. But at such high network utilization (90%) this will certainly cause many retransmissions. In our scheme, however, no frames will be transmitted for those connections that the prefetching algorithm returned the value $l_i = 0$.

5. Conclusions

We have presented a new scheme for the effective transmission of MPEG-compressed video traffic over ATM networks, for a VoD system with distributed video servers. It is based on the idea of prefetching that was originally proposed in [4,5], and was shown to have a great performance improvement in terms of network utilization compared to

traditional video smoothing schemes. Our motivation was the fact that these protocols are efficient only when there is one centralized server in the system that serves all the clients over a common transmission link. In a distributed environment, there is a large degradation in the performance of the prefetching protocols. In our scheme, we used a central controller to coordinate the transmission of future frames from distributed video servers. To make this possible, the MPEG traffic is first smoothed over a number of GOPs before entering the network. Therefore, the central controller is able to coordinate future transmissions, as the traffic will be constant over a number of frame periods.

We compared our scheme with the centralized JSQ prefetching protocol, and the experimental results showed that our scheme had very similar performance. The loss probability of our protocol was one order of magnitude bigger than JSQ prefetching for the same buffer size and network utilization. In addition, with only a small buffer size increment, we could get the same loss probability as the JSQ prefetching protocol. We have shown that our protocol offers the same advantages as JSQ prefetching: high network utilization, minimal start-up latency, and immediate response to user interactions. However, our scheme has the advantage that it is implemented in a system with distributed video servers. This will allow the distribution of different movies to several servers, and also allow multiple VoD service providers to share the same network.

Acknowledgements

This research is supported in part by the University of Hong Kong Area of Fundamentals in Information Technology, and by the State Scholarships Foundation of Greece.

References

[1] B. G. Haskell, A. Puri, and A. N. Netravali. *Digital video: an introduction to MPEG-2*. Chapman & Hall, 1997.

[2] V. O. K. Li and W. J. Liao. Distributed multimedia systems. *Proc. of the IEEE*, 85(7):1063–1108, July 1997.

[3] J. M. McManus and K. W. Ross. Video-on-demand over ATM: constant-rate transmission and transport. *IEEE J. Select. Areas Commun.*, 14(6):1087–1098, August 1996.

[4] M. Reisslein and K. W. Ross. Join-the-shortest-queue prefetching protocol for VBR video on demand. In *Proc. IEEE International Conference on Network Protocols (ICNP)*, pages 63–72, Atlanta, GA, October 1997.

[5] M. Reisslein, K. W. Ross, and V. Verillotte. A decentralized prefetching protocol for VBR video on demand. In *Proc. 3rd European Conference on Multimedia Applications, Services and Techniques (ECMAST)*, pages 388–401, Berlin, Germany, May 1997.

[6] O. Rose. Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems. In *Proc. 20th Annual Conference on Local Computer Networks*, pages 397–406, 1995. The MPEG traces were downloaded via anonymous ftp from ftp-info3.informatik.uni-wuerzburg.de in the /pub/MPEG directory.

[7] J. D. Salehi, Z. L. Zhang, J. F. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. In *Proc. ACM SIGMETRICS*, pages 222–231, Philadelphia, PA, May 1996.

[8] Z. L. Zhang, J. F. Kurose, J. D. Salehi, and D. Towsley. Smoothing, statistical multiplexing and call admission control for stored video. *IEEE J. Select. Areas Commun.*, 15(6):1148–1066, August 1997.