# A Transparent Rate Adaptation Algorithm for Streaming Video over the Internet

L. S. Lam, Jack Y. B. Lee, S. C. Liew, and W. Wang
*Department of Information Engineering*
*The Chinese University of Hong Kong*
*Shatin, N.T., Hong Kong, China*
*{lslam2, yblee, soung, wwang2}@ie.cuhk.edu.hk*

## ABSTRACT

*The lack of end-to-end quality of service support in the current Internet has caused significant difficulties to ensuring playback continuity in video streaming applications. This study addresses this challenge by investigating a new adaptation algorithm to adjust the bit-rate of video data in response to the network bandwidth available to improve playback continuity. Unlike previous works, the proposed algorithm is transparent to the video client, requires no parameter tuning, and yet can outperform existing algorithms. This paper presents this algorithm, evaluates and compares its performance with the best algorithm currently available using extensive trace-driven simulations.*

## 1. INTRODUCTION

The lack of end-to-end quality-of-service (QoS) support in today's Internet has caused significant difficulties to the deployment of video streaming services such as video broadcasting and video-on-demand. In particular, when the network becomes congested, significant packet losses will arise, leading to corrupted or even dropped video frames.

Given QoS support is unlikely to be widely available in the near future, researchers have resorted to another approach to tackle this problem. Specifically, a number of pioneering researchers have investigated algorithms to adapt the video bit-rate to the network bandwidth available [1-5]. For example, when the network becomes congested, the sender will reduce the bit-rate of the encoded video to alleviate the congestion. Clearly, reducing the bit-rate will also degrade the visual quality. Nevertheless, reducing the video bit-rate in a controlled manner at the sender will result in far better visual quality than attempting to recover from data loss at the receiver.

To perform video adaptation we must tackle two fundamental challenges. First, the sender must be able to dynamically control or convert the video bit-rate to the desired value. This can be accomplished by means of scalable video coding [6] and transcoding [7-9]. Second, an adaptation algorithm is needed to estimate the network bandwidth available, and subsequently determine the bit-rate to be used for converting and transmitting the video stream. This study focuses on the second challenge, i.e., design of the rate adaptation algorithm.

This problem has recently been studied by a number of researchers, including the studies by Rejaie, et al. [4] and Assuncao and Ghanbari [5] which adopted UDP as the network transport; and the studies by Cuetos and Ross [1], Cuetos, et al. [2], and Jacobs and Eleftheriadis [3] which adopted TCP as the network transport.

A common property of these adaptation algorithms is the existence of a configurable operating parameter [1-2], which is typically used in the feedback loop of the algorithms. Not surprisingly, as will be illustrated in Section 5, the choice of this operating parameter will significantly affect the performance of the rate adaptation algorithm. Unfortunately, to optimize this parameter for the best performance will require a priori knowledge of the network bandwidth available over the entire duration of the video session. This is clearly not possible in practice and thus poses significant difficulties to deploying these rate adaptation algorithms.

In this study, we address this issue by presenting a new rate adaptation algorithm that does not have configurable parameter at all. In other words, no prior knowledge of the available network bandwidth is needed nor required to run the rate adaptation algorithm. Our results show that compared to the existing algorithms, the presented algorithm can achieve comparable or even better performance and does so without the need to tweak any operating parameters.

## 2. SYSTEM MODEL

In this work we consider a video streaming system that streams pre-encoded video data using TCP as the network transport to the receiver for playback. Despite the common notion that TCP is unsuitable for video streaming

for its aggressive congestion control and full reliability, it does possess a number of appealing features.

First, TCP is intrinsically TCP-friendly and thus fairness with other TCP traffics is automatically guaranteed. Second, using TCP the sender can stream video using say the standard HTTP protocol to the client. As most, if not all, video players in the market supports HTTP-based video streaming and playback, compatibility is greatly enhanced. Third, for security reasons, many company and ISP blocks UDP traffic at their gateways, thus making UDP-based video streaming impossible. By contrast, TCP/HTTP streaming can pass through firewalls in the same way as web traffic. Finally, to perform bandwidth estimation the sender will need some form of feedbacks from the client. Thus with UDP transport the client will need to be modified to send explicit feedbacks to the sender to enable bandwidth estimation and subsequently rate adaptation to be performed. By contrast, TCP with its built-in flow control already can provide implicit feedbacks to the sender and thus no modification to the client is necessary. Again this will greatly enhance the compatibility of the rate-adaptation algorithm to the existing video player software.

Nevertheless, the rate-adaptation algorithm presented in this study can also be applied to UDP-based video streaming with appropriate support from the client's player software (e.g. sending explicit feedbacks).

Figure 1 shows the key components in the video streaming system. Assuming the video data are encoded at a constant bit rate of $r_{max}$ bps. The rate controller can convert the encoded video to any bit-rate between $r_{max}$ and $r_{min}$ (e.g., using scalable video coding [6] or transcoding [7-9]). Note that there is a lower limit $r_{min}$ on the achievable video bit-rate to model, for example, the bit-rate of the base layer in FGS encoded video [6] or the lowest achievable bit rate in transcoding [7-9].

In practice, even with a transcoder the video bit-rate may not be changed at arbitrary time due to the structure of the coding algorithm (e.g. group of pictures, etc.). Thus in the system model we assume video transcoding is performed in discrete video segments of fixed playback duration, denoted by $M$ seconds. The rate controller will then determine the target bit-rate for the next video segment based on estimation of the client's buffer occupancy. We denote the average bit rate for the $k^{th}$ video segment by $r_k$.

The transcoded video segments are then transmitted to the client using TCP. Note that the server does not limit the transmission rate here and simply sends the transcoded video data as fast as TCP allows. This ensures that available network bandwidth is fully utilized.

At the receiver, many existing video players will prefetch a certain amount of video data before starting playback to absorb the inevitable bandwidth fluctuations.

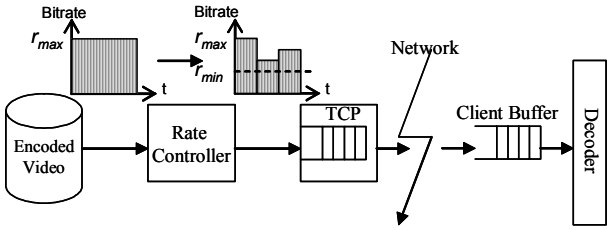


Figure 1: Block diagram of the system model.

We denote the playback duration of the prefetched video data by $B_p$ seconds. Depending on the specific player software, $B_p$ can be a fixed value known to the server, or it can be configurable by the users. If it is the latter case and the existing player software does not report this value to the server, the server will simply assume the worst case of no prefetch, i.e. $B_p = 0$ sec, in performing rate adaptation. Our results show that the performance difference is insignificant (c.f. Section 5-A).

## 3. CLIENT BUFFER OCCUPANCY AND NETWORK BANDWIDTH ESTIMATION

The objective of the rate adaptation algorithm is to prevent playback starvation caused by client buffer underflow. To prevent buffer underflow, the server will need to estimate the available network bandwidth as well as the client buffer occupancy, in terms of second's worth of video data.

Specifically, we make two assumptions on the receiver and the server. First, we assume that the client will not decode and playback a video frame until it is completely received. Thus if a frame arrives late missing the playback schedule, then the player will pause playback until the whole frame is received. We call the period of time when the playback is stalled due to late frame arrival *underflow time*. Second, we assume that the total size of the buffer in between the server application and the network (e.g., including the buffer inside the socket library and TCP) is a known constant, denoted by $Z$.

Estimation of the client buffer occupancy is then performed every time the server completes submitting a video frame to the network transport for delivery. For example, if the common socket library is used then this is equivalent to completing all `send()` function calls for the video frame.

Let $t_i$ be the completion time of submitting video frame $i$ for transmission, and let $f_i$ be the index of the oldest frame (i.e. with the smallest index number) that has not yet been completely received by the client at time $t_i$. Now as the server will submit data for transmission as fast as the transport allows, we can assume that the intermediate buffer at the server is always full, i.e., there are $Z$ bytes of data accumulated awaiting for transmission. Thus we can estimate $f_i$ from

$$f_i = \max\ n \ \text{ s.t. } \sum_{k=n}^{i} s_k \geq Z \qquad (1)$$

where $s_i$ is the size of frame $i$.

Similarly, after frame $i$+1 is submitted for transmission, we can compute $f_{i+1}$ using (1). Now if $f_{i+1} > f_i$, then we know that frame $f_i$ to frame $f_{i+1}-1$ must have arrived at the client during the time from $t_i$ to $t_{i+1}$. Assuming in this short interval the frames arrive at the client at a constant rate. Then we can estimate the arrival time of frame $k$, denoted by $T_k$, from

$$T_k = t_i + \frac{k+1-f_i}{f_{i+1}-f_i}\left(t_{i+1}-t_i\right) \quad k \in \left[f_i, f_{i+1}-1\right] \qquad (2)$$

Note that we ignored in (2) network and processing delay in receiving ACKs from the client. Our simulations show that this does not have significant impact on the algorithm's performance.

Knowing the arrival time of each video frame, we can then proceed to estimate the client buffer occupancy. Let $B_i$ (in seconds of video data) be the client buffer occupancy when frame $i$ arrives at the client and $G$ be the frame rate of the video. Then we can estimate the client buffer occupancy $B_i$ according to the following rules:

- Case 1 - $i \leq B_p \times G$

In this case the frame $i$ belongs to the initial prefetch part of the video, i.e., the player has not yet started decoding the received video data. Thus the buffer occupancy is equal to the duration of video data received:

$$B_i = i/G \qquad (3)$$

- Case 2 - $i > B_p \times G$

In this case, the way to estimate $B_i$ depends on whether or not the frame $i$ has arrived before all the data in the client buffer is consumed as illustrated in Figure 2.

If $(T_{i-1}+B_{i-1})-T_i \geq 0$, that means frame $i$ has arrived before the client buffer becomes empty, then $B_i$ is estimated as:

$$B_i = \left(B_{i-1}+T_{i-1}\right)-T_i+1/G \qquad (4)$$

Otherwise, if $(T_{i-1}+B_{i-1})-T_i \geq 0$, that means the client buffer has been empty for a period of time before frame $i$ arrived, then $B_i$ is simply equal to the time value of a frame, i.e.:

$$B_i = 1/G \qquad (5)$$

From the above derivation, we can estimate $B_i$ when frame $i$ has just arrived at the client. However, since the video bit rate of a segment has to be determined when all the data of the previous segment has been submitted into the server buffer, some frames of the previous segment are still in the server buffer. Therefore, to predict the client buffer occupancy after all the data of the previous segment has arrived at the client, we need to predict the arrival times of the frames in the server buffer.

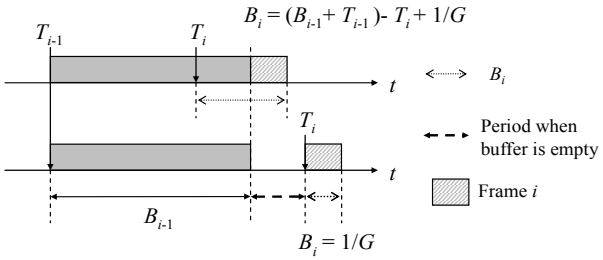


Figure 2: Two ways to estimate $B_i$ when $i > B_p \times G$.

Let $n_i$ be the index of the last frame of segment $i$, we have to predict $B_{n_i}$ at time $t_{n_i}$ while frame $f_{n_i}$ to frame $n_i$ are still in the server buffer and then use the predicted $B_{n_i}$ to perform adaptation of segment $i$+1.

Assuming the remaining data in the server buffer at time $t_{n_i}$ will arrive at the client at a constant rate of $D_{i+1}$', which is also the estimated TCP throughput for sending the segment $i$+1, the arrival times of the remaining frames are estimated as follows:

$$T_k = t_{n_i} + \frac{1}{D_{i+1}'}\sum_{j=f_{n_i}}^{k} F_j(t_{n_i}) \quad \forall k \in \left[f_{n_i}, n_i\right] \qquad (6)$$

where $F_i(t)$ is the remaining amount of data of frame $i$ at time $t$. With $T_k, k \in [f_{n_i}, n_i]$, we can estimate $B_{n_i}$.

To estimate $D_{i+1}$', we simply take the rate at which segment $i$ was submitted into the server buffer as the estimated value, i.e.,

$$D_{i+1}' = \sum_{k=m_i}^{n_i} s_k \Big/ \left(t_{n_i} - t_{m_i}\right) \qquad (7)$$

where $m_i$ is the index of the first frame of segment $i$. This is because the rate at which data are submitted into the server buffer is equal to the rate at which data leave the server buffer.

# 4. RATE ADAPTATION

Armed with a mean to estimate the client buffer occupancy and network bandwidth, the next challenge is to devise an adaptation algorithm to control the video bit-rate to prevent client buffer underflow.

## A. Segment-based Rate Control

As video data are transcoded and transmitted in fixed-duration segments, the server must determine the target bit-rate before converting a video segment for transmission. The server determines the target bit-rate based on two factors, namely the estimated client buffer occupancy and the estimated network bandwidth available which could be estimated using techniques described in Section 3.

Suppose segment $i$ has just been submitted to the server buffer, with the estimated $D_{i+1}'$ and $B_{n_i}$, we can predict the client buffer occupancy after transmitting the segment $i$+1 to the client, i.e. $B_{n_{i+1}}$, from:

$$B_{n_{i+1}} = B_{n_i} + M - \frac{Mr_{i+1}}{D_{i+1}'} \quad (8)$$

where the last term is the predicted time taken to send the whole $i$+1th segment to the client. By rearranging (8), we obtain:

$$r_{i+1} = \left[1 - \frac{B_{n_{i+1}} - B_{n_i}}{M}\right] D_{i+1}' \quad (9)$$

From (9), we can relate the video bit-rate $r_{i+1}$ with the estimated client buffer occupancy (represented by $B_{n_{i+1}}$). Our goal is to adjust the video bit-rate to maintain the client buffer occupancy to above a given threshold denoted by $B_T$ such that short-term bandwidth variations can be absorbed. In practice, $B_T$=$B_p$ when $B_p$ is known, otherwise it is set to 5 seconds.

Specifically, if $B_{n_{i+1}} < B_T$ then it implies the client buffer occupancy is below the threshold. Hence the server will reduce the video bit-rate to raise the buffer occupancy to $B_T$ by substituting $B_{n_{i+1}} = B_T$ in (9) to obtain:

$$r_{i+1} = \left[1 - \frac{B_T - B_{n_i}}{M}\right] D_{i+1}' \quad (10)$$

Otherwise if $B_{n_i} \geq B_T$, then it implies the client buffer occupancy is above the threshold. In this case the server will simply maintain the current client buffer occupancy by setting $B_{n_{i+1}} = B_{n_i}$ in (9) to obtain $r_{i+1}$. This is a conservative strategy to reduce the possibility of buffer underflow. Thus, we have

$$r_{i+1} = D_{i+1}' \quad (11)$$

Finally, the server checks and limits the computed video bit rate to the feasible range [$r_{min}$, $r_{max}$] by

$$r_{i+1} = \min\left\{r_{\max}, \max\left\{r_{\min}, r_{i+1}\right\}\right\} \quad (12)$$

Note that in contrast to previous works [1-3], this adaptation algorithm has no control parameter that requires either offline or online optimization. This has practical significance as optimizing the control parameters in the existing algorithms [1-3] requires *a priori* knowledge of the available network bandwidth over the entire duration of the video session, clearly impossible in practice.

### B. Preemptive Rate Control

In our experiments, we found that the available network bandwidth can occasionally drops drastically to a very low value. These sudden bandwidth drops do not appear to be predictable and thus can result in client video playback starvation.

The fundamental problem is that the adaptation algorithm is executed only when a new video segment is to be transmitted. Thus if bandwidth drops significantly, then the transmission of the current video segment will stall. The adaptation algorithm cannot react in this case as the current video segment has not yet been completely transmitted. Meanwhile the client will continue consuming video data for playback and thus may eventually runs into buffer underflow.

To tackle this problem, we propose a *preemptive scheduling* technique to shorten the time at which the adaptation algorithm can react to changing network conditions. Instead of waiting for a video segment to be completely submitted into the server buffer, the scheduler will timeout after $Mr_{i+1}/D_{i+1}'$ seconds, which is the expected time required to submit the $i$+1th video segment into the server buffer, even if not all video data have been submitted. In this case, any data not yet submitted for transmission will be discarded and the remaining video segment transcoded again according to the new estimates on client buffer occupancy and available network bandwidth.

Note that preemptive rate control requires the video transcoder to be able to adjust the video bit rate in between a video segment. The implementation will be highly dependent on the video compression employed and further study is required to identify the constraints and tradeoffs of this requirement.

## 5. PERFORMANCE EVALUATION

In this section, we use trace-driven simulation written in ns-2 [10] to evaluate the performance of the proposed adaptation algorithm (denoted by AVS) and compare it with the current state-of-the-art algorithm proposed by Cuetos and Ross [1-2] (denoted by CR).

Figure 3 depicts the simulated network topology. We use the common NewReno TCP [11-12] as the transport protocol to deliver the video data to the client. Cross traffic is generated from a packet trace file obtained from Bell Labs [13-14][1]. The trace file captured 107 hours of network traffic passing through a firewall. We divide the 107-hour trace file into 107 1-hour trace files and run a simulation for each 1-hour trace file to evaluate the algorithms' performance under different cross traffics.

Both the streaming traffic and the cross traffic share a link of $R$ Mbps as shown in Figure 3. For each simulation, we adjust $R$ such that the network has just sufficient

[1] Network traces used in the simulations belongs to NLANR project sponsored by the National Science Foundation and its ANIR division under Cooperative Agreement No. ANI-9807479, and the National Laboratory for Applied Network Research.

bandwidth to stream the video $R = r_{\max} + \bar{c}$, where $\bar{c}$ is the average data rate of the cross traffic. We summarize the system settings in Table 1.

We use two performance metrics, namely underflow ratio and bandwidth utilization, to evaluate the algorithms' performance. Underflow ratio is defined as the ratio of underflow duration (i.e. the duration of time that playback starvation occurs) to the video length. Bandwidth utilization is defined as:

$$Utilization = \sum_{i=0}^{N} s_i \Big/ \int_0^{\max\{P+L,S\}} v(t)dt \qquad (13)$$

where $N$ is the total number of frames, $P$ is the initial prefetch delay, $L$ is the movie length, $S$ is the total time taken to stream the video and $v(t)$ is the TCP throughput at time $t$. The value of bandwidth utilization is in the range of [0,1]. This metric measures how well an algorithm utilizes the available network bandwidth.

## A. Sensitivity to prefetch duration

The proposed rate adaptation algorithm makes use of knowledge of the client's initial prefetch duration in estimating the client buffer occupancy. However if this is not known then it simply assumes no prefetch is performed.

To investigate the performance impact of such knowledge we run two sets of simulations for all 107 traffic traces, one set with the prefetch duration known to the server and the other set simply assuming no prefetch. In both cases the client has a prefetch duration of 5 seconds.

Table 2 shows the underflow ratio and bandwidth averaged over all 107 traces for the two cases. In both cases the differences are insignificant and thus implying that the proposed rate adaptation algorithm is insensitive to the knowledge of the prefetch duration. Therefore in practice we can simply assume no prefetch if the prefetch duration is not known.

## B. Effectiveness of preemptive rate control

To investigate how much performance gains can be obtained from preemptive rate control, we run two sets of simulations for all 107 traffic traces, one with segment-based rate control and the other with preemptive rate control.

In all 107 traces, preemptive rate control achieves lower underflow ratios compared to segment-based rate control. On average, the underflow ratio is reduced by 20% when preemptive rate control is used. Nevertheless preemptive rate control does require more complex transcoders and thus further investigation is needed to quantify the gains and the tradeoffs.

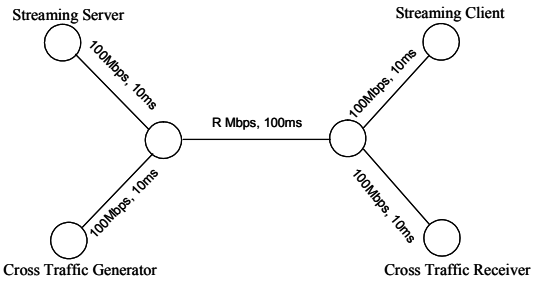


Figure 3: The network topology in the simulation.

Table 1: System settings for simulations.

| Parameter | Symbol | Value |
|---|---|---|
| Prefetch duration | $B_p$ | 5 seconds |
| Video segment length | $M$ | 1 seconds |
| Original video bit-rate | $r_{max}$ | 1.1 Mbps |
| Lowest video bit-rate | $r_{min}$ | 200 kbps |
| Video Length | | 3000 seconds |
| TCP MSS | | 1500 bytes |

Table 2: Effect of knowledge of the prefetch duration.

| Prefetch Duration | Known | Unknown | Difference |
|---|---|---|---|
| Bandwidth Utilization | 0.9999 | 0.9998 | ~0.01% |
| Underflow Ratio | 0.056335 | 0.055502 | ~1.48% |

## C. Comparison with the CR algorithm

In this section, we compare the proposed rate adaptation algorithm (the AVS algorithm) with the current state-of-the-art algorithm proposed by Cuetos and Ross [1-2] (the CR algorithm).

In the CR algorithm, there is a control parameter $\alpha$ ($0 \le \alpha \le 1$) that can substantially affect the performance. To find the optimal value for $\alpha$ it is necessary to know the network bandwidth availability over the entire duration of the video session. This is clearly not possible in practice and the authors did not explain how to adjust the parameter in practice.

Thus to obtain performance results for the CR algorithm we run 2,000 simulations with the control parameter $\alpha$ varied from 0 to 1 with a step size of 0.0005. We found that the optimal value for $\alpha$ depends heavily on the particular traffic trace chosen, and can range from 0 to 0.7 over the 107 traces.

As the optimal $\alpha$ is not known a priori, we compare CR with AVS by computing the proportion of the 2,000 simulation runs that result in higher underflow ratio than the AVS algorithm, which does not need any parameter tuning. The results are summarized in Figure 4, which also plots the bandwidth utilization ratio, defined as (bandwidth utilization of AVS)/(average bandwidth utilization of CR over all $\alpha$ values).

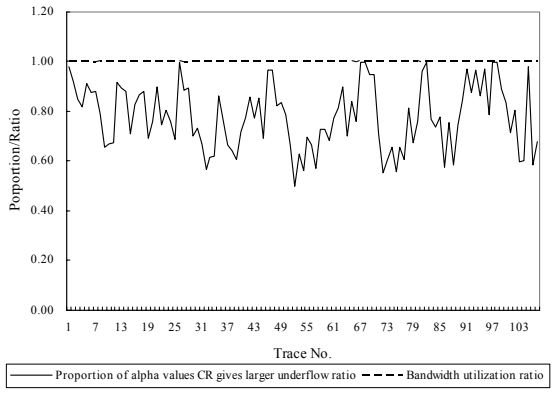


Figure 4: Comparison of underflow ratios and bandwidth utilization of CR and AVS for different traces.

The results in Figure 4 show that the proposed AVS algorithm outperforms CR in more than half of the simulation runs with different $\alpha$ values. Averaging over all 107 traces, the proposed AVS algorithm can achieve lower underflow ratio than the CR algorithm for 77% of the $\alpha$ values. This shows that in practice, the proposed AVS algorithm is likely to perform better and yet does not require any a priori knowledge of the network bandwidth available nor tuning of any control parameter.

Despite the reduction in the underflow ratio, the proposed AVS algorithm can still make efficient use of the network bandwidth, and achieving bandwidth utilization similar to that of the CR algorithm.

## 6. CONCLUSIONS

In this study we presented a new rate adaptation algorithm for video streaming over the Internet. The algorithm has two unique features to maximize its compatibility with existing video player software. First, we show that the rate adaptation algorithm can be applied to streaming video over TCP/HTTP, which is compatible with most of the existing video player software. Second, the rate adaptation algorithm performs network bandwidth and client buffer occupancy estimations using only local information. Thus explicit feedbacks from the client is not needed and hence existing video player software can be supported. More importantly, unlike previous approaches the proposed algorithm does not need any parameter tuning to operate nor requires *a prior* knowledge of the network bandwidth available to perform well, thus simplifying the deployment of the adaptation algorithm in practice. Our results show that the proposed algorithm can outperform existing algorithm and yet still achieve efficient bandwidth utilization.

## ACKNOWLEDGEMENTS

This research is funded in part by an Earmarked Grant (CUHK4229/00E) from the HKSAR Research Grant Council and in part by the Area of Excellence Scheme, established under the University Grants Council of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99).

## REFERENCES

[1] P. de Cuetos and K.W. Ross, "Adaptive Rate Control for Streaming Stored Fine-Grained Scalable Video," *Proc. NOSSDAV*, May 2002, pp.3-12.

[2] P. de Cuetos, P. Guillotel, K.W. Ross and D. Thoreau, "Implementation of Adaptive Streaming Of Stored MPEG-4 FGS Video Over TCP," *Proc. IEEE Multimedia and Expo, 2002*, pp.405-408.

[3] S. Jacobs and A. Eleftheriadis, "Streaming Video using Dynamic Rate Shaping and TCP Congestion Control," *Journal of Visual Communication and Image Representation*, Vol. 9, No. 3, 1998, pp.211-222.

[4] R. Rejaie, M. Handley and D. Estrin, "Architectural considerations for playback of quality adaptive video over the Internet," *Technical Report 98-686*, USC-CS, Nov. 1998.

[5] P.A.A. Assuncao and M. Ghanbari, "Congestion control of video traffic with transcoders," *Proc. IEEE Int. Conf. Communications*, Vol. 1, June 1997, pp. 523-527.

[6] W. Li, "Overview of Fine Granularity Scalability in MPEG-4 Video Standard," *IEEE Trans. Circuits and Systems for Video Tech*. Vol. 11, No. 3, March 2001, pp.301-317.

[7] P. A. A. Assunção and G. Mohammed, "A Frequency-Domain Video Transcoder for Dynamic Bit-Rate Reduction of MPEG-2 Bit Streams," *IEEE Trans. Circuits and Systems for Video Tech.,* Vol. 8, No. 8, Dec. 1998, pp.923-967.

[8] B. K. Natarajan and B. Vasudev, "A Fast Approximate Algorithm for Scaling down Digital Images in the DCT Domain," *Proc. IEEE Int. Conf. Image Processing*, Vol. 2, Oct. 1995, pp.241-243.

[9] H. Sun, W. Kwok and J.W. Zdepski, "Architecture for MPEG compressed bitstream scaling," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 6, No. 2, April 1996, pp.191-199.

[10] The network simulator – ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/

[11] S. Floyd and V. Paxson, "Difficulties in Simulating the Internet," *IEEE/ACM Trans. Networking*, Vol. 9, No. 4, August 2001, pp. 392-403.

[12] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 2582, April 1999.

[13] NLANR Measurement and Network Analysis Group. [Online]. Available: http://pma.nlanr.net/Traces/long/bell1.html

[14] Bell Labs Internet Traffic Research. [Online]. Available: http://cm.bell-labs.com/cm/ms/departments/sia/InternetTraffic/