Distortion-Optimized Scheduling of Packet Video for Internet Streaming with TCP

Antonios Argyriou
Philips Research, Eindhoven, The Netherlands

***Abstract*—In this paper, we propose a scheduling algorithm for improving the performance of packetized video streaming with the transmission control protocol (TCP). The scheduling algorithm estimates the effect of TCP-induced losses on the expected decoder distortion, and makes distortion-optimized scheduling decisions for future transmissions of media packets. The novel characteristic of our approach is that the scheduling algorithm calculates the probability of decoding failure for a specific media packet, by considering the effect of TCP internal mechanisms both at the sender and the receiver. Our objective is to maximize the quality at the decoder for a given startup delay and playback buffer size. Experiments show that decoded video quality is significantly improved when compared with TCP streaming systems that ignore the dynamic behavior of the protocol.**

I. Introduction

The explosive growth of the video content available on the Internet, continues to drive the need for improved media streaming services. However, the transmission control protocol (TCP) [1], that is responsible for the bulk of the existing elastic Internet traffic, is not the protocol of choice for media streaming applications. The main reasons are the rapid throughput fluctuations and the reliability mechanism that introduces additional delays [2]. Therefore, the general belief is that the transport protocol that is best suited for media streaming is UDP, on top of which several application specific algorithms can be implemented [3]. However, the absence of congestion control from UDP, can cause performance deterioration for TCP-based applications if wide-scale deployment takes place on the Internet [4]. As a result, the IETF transport area working group has devoted considerable effort to the standardization of the datagram congestion control protocol (DCCP) [5], that intends to provide congestion control for unreliable data delivery.

Even though the lack of consistent congestion control across the media applications that use UDP might be the primary concern of IETF, application designers avoid UDP for different reasons. The widely deployed network address translation (NAT) routers and firewalls, are often configured to prevent connectionless UDP traffic. Therefore, Internet media streaming applications have to use TCP as a fallback transport protocol. However, when this happens, the establishment of a new TCP connection introduces an additional delay, making thus the user-perceived delay even more noticeable. To compensate for these two closely related problems, media streaming systems frequently use TCP from the start of the session. The result is that much of the Internet video streaming traffic is carried over TCP [6]. For example the popular Skype VoIP application favors the use UDP for media packets, but when UDP hole-punching in a firewall fails, it falls back to TCP [7], [8]. This is the case in most corporate networks where symmetric NAT is used. Other media streaming technologies like the Helix media server, a project initiated by RealNetworks [9], the Apple [10] and Microsoft Windows media streaming servers [11], are examples of systems that have to use frequently TCP. From this preliminary discussion one realizes that enhancing the performance of video streaming with TCP with realistic assumptions, can have significant practical value.

Therefore, our goal is to design a streaming system that offers improved video quality at the decoder, for the most fundamental unicast media streaming scenario: A group of video packets that belong to a media presentation must be transmitted over an IP network with TCP, for a given startup delay and playback buffer size. In addition, we set several restricting requirements in order to allow easy implementation and deployment. We seek to design a video streaming system with TCP that: a) Does not require the slightest modification to the TCP protocol and b) it must be implemented at the application layer without requiring information from the lower protocol layers that cannot be provided trough the standard TCP socket API.

We believe that in order to improve video quality at the decoder, and at the same time meet the aforementioned requirements, a promising solution is to employ a *distortion-aware packet scheduling* algorithm at the streaming server. This means that the server must select for transmission the packets that have highest probability of being decoded on-time and also have the highest contribution to the reduction of the decoder distortion. The aforementioned principle was thoroughly described and formalized in [12]. Many papers extended the use of that framework in different contexts. In one of these papers, that is related more to our work, the authors assume a variable bitrate channel model that approximates TCP throughput variations [13]. By following this approach, it is possible to estimate the variations of the playback buffer occupancy over time, and devise therefore transmission schedules that contribute to a reduction of the playback buffer overflow/underflow events.

While the main concept behind the proposed scheduling algorithm is also based on distortion-optimized stream-

This work was performed before the author joined Philips Research.

ing [12], our work differs in one fundamental aspect. Our algorithm implements TCP-aware distortion optimized (TcpDiO) scheduling, since it is engineered to work when the actual TCP is used. To achieve improved performance in this system, the short-term dynamic behavior of TCP is taken into account when calculating the probability of decoding failure for specific video packets. Consequently, the system does not rely on the use of channel models for estimating the available bandwidth and packet loss behavior. The reason we followed this approach is because both TCP throughput and delay fluctuations are so abrupt, that it is very difficult in practice to estimate the future available TCP rate. Eventually, the improved short-term estimates of TCP's behavior, allow for optimal adjustment of the media scheduling window and reduction of buffering requirements at the receiver side.

The rest of this paper is organized as follows. Initially, in section II we provide an overview of TCP and describe the protocol features that are of interest in this work. An overview of the main principles behind the proposed scheduling and streaming algorithms is presented in section III. Section IV describes the distortion metric that is used by the scheduling algorithm, while the probability of decoding failure for individual video packets is calculated in section V. Details of the overall streaming system are described in section VI and section VII presents our experimental results. A thorough analysis of related works in the area of video streaming with TCP can be found in section VIII. Finally, section IX presents our conclusions.

II. TCP Background and Performance Issues

A brief description of the basic TCP algorithms is necessary before we delve into the details of the proposed streaming system. TCP is a connection-oriented protocol that offers reliable delivery for an unstructured byte sequence [1]. The protocol splits the incoming bytes into data chunks, called segments, and assigns a unique sequence number to each of them. To control the flow of segments into the network, TCP maintains a parameter called congestion window. The congestion window indicates the number of segments that can be outstanding in the network. The details of the algorithm that control the evolution of this parameter can be found in [14]. Regarding data transmission, TCP uses a self-clocking algorithm, where each new acknowledgement arrival at the sender, triggers the transmission of a new data segment if the congestion window allows it. Segment losses are identified from the gaps in the sequence numbers of the received segments. Until a missing segment is retransmitted successfully, the TCP receiver sends acknowledgements for the last in-order received segment. When the sender receives three duplicate acknowledgments, it retransmits the lost segment immediately (fast retransmit algorithm), and reduces by half the congestion window (fast recovery algorithm) [14]. When no duplicate acknowledgments arrive, and the timer expires for a specific segment, the sender retransmits it and sets the congestion window to one. The result of packet losses is that the value of the congestion window fluctuates which makes both the throughput and delay of individual packets to fluctuate. Furthermore, even with a single packet loss, the TCP receiver buffer can block waiting to receive the lost packet, since unordered delivery is not supported. When the sender identifies a loss through three duplicate acknowledgements or a timeout, and retransmits the missing segment, usually many segments will be blocked at the TCP receiver buffer.

III. System Overview

Before we analyze the main concepts behind the overall streaming system, and in particular the proposed scheduling algorithm, we define here the necessary notation and highlight some basic concepts. Let us denote the group of video packets that their decoding deadline has not elapsed, and are therefore eligible for transmission as N, while the packets that have already been transmitted as M. Assume also that the order in which packets were produced by the encoder is expressed in the following form: $\{l_1, l_2, l_3, ..., l_{M+N}\}$. The decoding dependencies of these packets are expressed in the form of $l_1 \prec l_2$ when packet l_1 is needed for the successful decoding of packet l_2. We assume that these dependencies are available in the form of a directed acyclic graph that is stored in a metafile that is produced during encoding [12]. On the successful transmission of a video packet l we assume that the reduction of the decoder distortion is Δd_l, similar to [12]. Finally, the estimate of the decoding failure probability (DFP) for packet l, at time instant t and under transmission schedule π is denoted $\varepsilon(l, \pi, t)$.

In this paper, we define a schedule to be a transmission order of the N video packets eligible for transmission. The initial order of video packets that the encoder produced, is the one that is provided as input to the scheduler. Therefore, the task of the scheduler is to intelligently rearrange the transmission order of the eligible packets, and derive a different schedule π_k that minimizes the DFP for the most important packets. To this aim, we develop an analytical model that calculates on-the-fly the DFP for each video packet, and subsequently an estimate of the decoder distortion. However, the size of the playback buffer B at the client, is the one constraint that must be met by the scheduler.

A. Packet to Segment Assignment

To facilitate the calculation of the DFP, we designed our streaming algorithm so that data transmission is distinguished at the application and transport layers. This means that at the application layer, a separate feedback mechanism for acknowledging video packets is used. This is necessary since acknowledgments regarding the delivery of TCP segments does not necessarily guarantee that the corresponding video packet was delivered on-time for decoding. The client reports with feedback messages (RTCP receiver reports [15]), the sequence number of the last correctly decoded video packet and the measured packet loss rate. Fig. 1 illustrates this concept, where the group of video packets that have already been sent

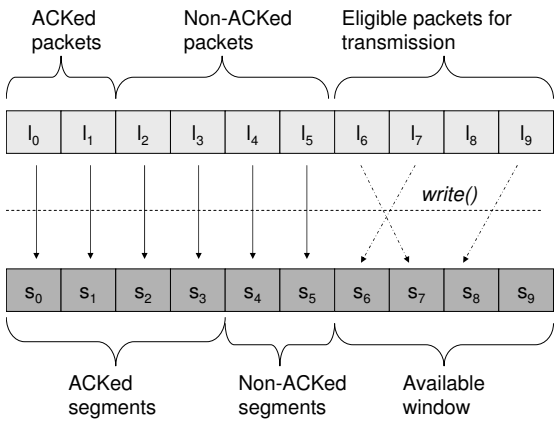

Fig. 1. Assignment of video packets (l) to TCP segments (s) through the socket API *write*() call.

and acknowledged are the l_0, l_1, while packets $l_2, ..., l_5$ have been sent but not acknowledged. On the other hand, segments $s_0, ..., s_3$ have been acknowledged by TCP while s_4 and s_5 they have not. Now the next group of packets that can be scheduled are the $l_6, ..., l_{11}$, but TCP only allows four segments to be sent in the current transmission opportunity. The goal is to transmit the packets that are the most important as soon as possible. In order to make an optimal decision, we estimate the effect of different transmission schedules on the distortion of the decoded video.

B. Streaming Packet Video with TCP

Apart from the scheduling algorithm, the rest of the functionality of the streaming server is simple. The server packetizes the encoded video frames into RTP packets according to the network maximum segment size (MSS), and commits these packets through the socket API to TCP. Each video frame does not have to be contained entirely in a single TCP segment. Since TCP perceives data as an unstructured sequence of bytes, RTP is essentially used for preserving boundaries between different video packets. The server is also responsible for padding each video packet so that its size is equal to MSS and can fit in exactly one TCP segment. The scheduler also creates an initial transmission schedule for the media packets, and during the session it re-schedules them according to the selected startup delay, the probability of decoding failure, and the contribution of each packet to the decoder distortion. On the client side, the application reads the data from the TCP socket when they become available and places the RTP packets in the decoder buffer. The decoding and playback commences after the initial startup delay. Re-buffering of a few video packets is used when there is a buffer underflow event.

IV. Distortion Metric

For making the optimal scheduling decisions, the scheduler needs a distortion metric in order to evaluate fast and accurately the different options. In order to express the distortion for a group of video packets, recall that the successful decoding of a single packet l will decrease distortion by Δd_l. Therefore, for estimating the overall *distortion decrease* under schedule π_k, all the packets being scheduled must be considered. This distortion decrease can be written:

$$\Delta D(\pi_k) = \sum_{l \in N} \Delta d_l [1 - \varepsilon(l, \pi_k, t)] \tag{1}$$

The objective is to create a schedule that maximizes the decrease in the observed decoder distortion according to 1. Formally this can be written:

$$\pi_k = arg \max \Delta D$$

$$\text{s.t. } enum(\pi_k(t)) \leq \min[awnd(t), B] \tag{2}$$

The last constraint guarantees that the number of packets scheduled for transmission in the current instant t, is not greater than the allowed window by TCP or the client playback buffer size constraint B.

In the next two sections, we will first calculate analytically the decoding failure probability ε, and then we will present an algorithm that can derive in real-time valid transmission schedules.

V. Decoding Failure Probability

Our goal in this section to calculate the decoding failure probability for each video packet l that is eligible for transmission or has already been transmitted. We explained earlier that all the packet that are needed for the successfully decoding l, must be received on-time. Therefore, the probability that *all* the video packets that are needed for the successful decoding of packet l, are decoded correctly is:

$$g(l) = \prod_{l' \in N, l' \prec l} (1 - \varepsilon(l', \pi_k, t)) \prod_{l' \in M, l' \prec l} (1 - \varepsilon(l', \pi_{k'}, t)) \tag{3}$$

The first product term expresses the probability that all video packets that are currently considered for scheduling/transmission under policy π_k will be successfully decoded. The second product term expresses the same probability but for a group of M packets that have already been transmitted under a previous schedule $\pi_{k'}$.

Now we have to calculate the probability that the packet l itself is received on-time for decoding. Let us denote as $t_{Si}(l)$ the time instant that video packet l is sent by TCP, with the subscript i denoting the transmission attempt. Now the DFP for video packet l, will depend on the specific transmission pattern that the initial TCP transmission (at t_{S0}) and future re-transmissions will create. Recall that the proposed application layer scheduler can only decide once for the transmission of an application packet, which makes the first transmission $t_{S0}(l)$ very crucial[1]. Therefore, if a packet has a playback deadline at time instant t_D, while the forward tip time or delay is FTT, then the DFP estimate at time instant t is equal to:

$$\varepsilon(l, \pi_k, t) = 1 - P\{FTT < t_D(l) - t_S(l)\} g(l) \tag{4}$$

[1]Essentially $t_{S0}(l)$, is the result of an application layer schedule π_k.

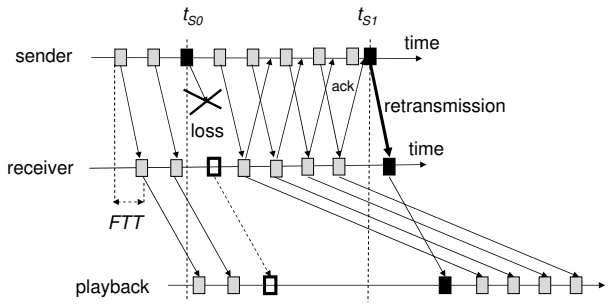

Fig. 2. Unordered delivery of TCP segments results in blocking at the receiver buffer.

The first term is the probability that the corresponding TCP segment for packet l arrives on-time, while the second term was defined in (3). For the initial transmission of any packet the result we derived in (4) will become:

$$\varepsilon_0(l,t) = 1 - P\{FTT < t_D(l) - t_{S0}(l)\}g(l) \quad (5)$$

The previous equation will provide the initial estimate of the DFP for any transmitted video packet.

A. Acknowledged TCP Segments

We explained earlier that successful delivery of a TCP segment does not translate to the on-time delivery of the corresponding video packet. To avoid inconsistencies, application-layer feedback is used for conveying information regarding the correct decoding of video packets. In most cases, the DFP estimate for a video packet with acknowledged the corresponding segment, will be given by (4). However, if the acknowledgment for the TCP segment arrived before t_D, then the sender can safely assume that the video packet was received on-time. Therefore, the DFP for the video packets that correspond to acknowledged TCP segments, is

$$\varepsilon_{ACKed}(l,t) = \begin{cases} \varepsilon_0(l,t) & \text{if } t_{ACK} > t_D \\ 1 - g(l) & \text{if } t_{ACK} \leq t_D \end{cases} \quad (6)$$

Note that the first term is not just a approximation that is made for estimating DFP, but it is the correct estimate. The second term is a way to reduce the computational overhead when the fate of specific TCP segments is available.

B. Retransmissions and Unordered Delivery

The most important step is to calculate how DFP is affected not only for lost and retransmitted segments, but also for the segments that are blocked at the TCP receiver buffer. We will try to illustrate the modeling process, by using Fig. 2. In this figure a single segment is lost. When TCP receives the third duplicate acknowledgment, it will retransmit the lost packet instantly at time t_{S1}. The DFP of all the packets that were sent after the initial transmission t_{S0}, will depend on the retransmission of the lost segment, since its arrival will release the blocked segments at the TCP receiver buffer. In Fig. 2 four segments are blocked at the receiver. Note that in case a packet is loss identified with three duplicate acknowledgements, the TCP sender will retransmit the missing segment the instant it receives the third duplicate ACK (fast retransmission algorithm). However, in case of a timeout, the time instant that the segment will be retransmitted, will include the duration of the timeout which is equal to the retransmission timer (RTO) maintained by TCP.

To formalize our observations, we can write for the probability of on-time delivery for a specific packet l, that was sent after the presumably lost packet m, (i.e. $t_{S0}(m) < t_{S0}(l)$):

$$\rho_{TD}(l) = P\{FTT < t_D(l) - t_{TD}(m) | m \text{ was lost}\} \quad (7)$$

In this equation t_{TD} is the time instant that the TCP sender received the third duplicate ACK. In case of a timeout, the packet will be retransmitted when the retransmission timer (RTO) expires for this packet, making thus the probability for on-time delivery:

$$\rho_{TO}(l) = P\{FTT < t_D(l) - (t_S(m) + RTO) | m \text{ lost}\} \quad (8)$$

These two equations are very important since they capture TCP's inability to deliver to the application segments that are received out-of-order. So the DFP will become in case of segment losses:

$$\varepsilon_{loss}(l,t) = \begin{cases} 1 - g(l,\pi_k,t)\rho_{TD} & \text{if } TD \\ 1 - g(l,\pi_k,t)\rho_{TO} & \text{if } TO \end{cases} \quad (9)$$

In practice, (4), (6), and (9), are used for updating the DFP of individual packets dynamically.

VI. TcpDio Scheduling and Streaming Algorithm

In this section, we present in detail the scheduling and streaming algorithms as well as the implementation details. The pseudo-code is presented in Fig. 3 and highlights the main operations executed by our system. Before the streaming session starts, the scheduler cannot derive an optimal transmission schedule, since the DFP is not known. Therefore, it creates an initial transmission schedule π_0 according to the importance Δd_l of all the packets, and is limited only by the client buffer size B. We discovered that in practice this approach results in reduced number of packet rescheduling events.

The $update\_dfp()$ procedure continuously updates the DFP estimate for transmitted packets by using the equations that we developed in section V. The mean of the forward trip time (FTT) is estimated by application layer feedback from the client, since TCP calculates only the RTT. In order to smooth out the mean value of the FTT, we adopt a low-pass filter that is similar with TCP [14]:

$$FTT' \longleftarrow \frac{7}{8}FTT + \frac{1}{8}FTT_m \quad (10)$$

FTT_m is the last actual measurement that was calculated based on receiver feedback.

The actual schedule is derived by the $tcpdio\_schedule()$ procedure. The parameter K indicates the fact that a subset of the N eligible packets are considered for re-scheduling throughout the session, due to the finite size of the decoder buffer. In order to

```
tcpdio_shedule_init()
    - Create initial schedule π_0
    - Setup TCP connections
    - Calculate t_D for each video packet
update_dfp(M)
    Update FTT estimate
    for all l ∈ M do
      Compute g(l) according to (3)
      Compute ε(l, π'_k, t) according to (4)
      if TO indication for pkt m then
        Compute ε(l,t) according to (8), (9)
      end if
      if Triple duplicate event for pkt m then
        Compute ε(l,t) according to (7), (9)
      end if
    end for
tcpdio_shedule(t,K)
    S_K = parse_dag(l_{M+1}, ..., l_K)
    for all k ∈ S_K do
      for all l ∈ (l_{M+1}, ..., l_K) do
        Compute ε_0(l,t) according to (3), (5)
      end for
      calculate ΔD(π_k)
    end for
    π* = arg max ΔD_k, k ∈ S_K
```

Fig. 3. Pseudo-code for the algorithms used by the TcpDiO streaming system.

prune the search space for possible schedules, we use the information in the directed acyclic graph metafile. For example when there is the strict decoding order between two video packets expressed in the form $l_1 \prec l_2$, then the order $\{l_2, l_1\}$ is rejected by the *parse_dag*() procedure. Therefore, the set S_K contains only the valid transmission schedules. Furthermore, the fact that no retransmissions are employed at the application layer also contributes to reducing the search space. Even though a more sophisticated algorithm could be used, for now we apply a greedy search algorithm within a few groups of pictures (GOP).

A. Timeout Indications

One of the practical problems that has to be solved, is the early identification of timeout events, since the sender cannot identify it until it happens. In order to deal with this problem, the sender measures the rate of acknowledgements from the receiver. A similar method with the FTT calculation is used for filtering the inter-arrival times of acknowledgements, but more importance is given to the last measurement sample. In practice, when the estimate of the inter-arrival time between acknowledgments is doubled when compared to the last sample, then an imminent timeout is assumed, and the DFP is updated accordingly.

B. Linux Implementation

We implemented the streaming server and the scheduling algorithm in Linux. The Linux system provides several facilities that allow practical implementation of our algorithm. The most important one is that it has internal data structures that keep track of low-level parameters of the active TCP connections. The streaming application that creates the TCP socket at the server, requests the current TCP parameters after each group of packets is scheduled and sent to TCP. This is done by calling the *getsockopt*() system call with the TCP_INFO option. This call fills a memory structure with information described in *struct tcp_info*. This approach allows the application to obtain low-level information regarding the TCP connection, without affecting the protocol operation. However, since this Linux data structure does not contain the RTO value for every packet, we use the RTT to calculate it similar to the TCP [14].

VII. Experimental Evaluation

The experimental setup includes the streaming server, a middlebox, and the client, all different Linux machines. We used the Linux network emulator [16] at the middlebox. The link capacity and average delay are set to 1Mbps and 30ms respectively. Shortly after the start of the session, we initiated another 700Kbps CBR/UDP flow that resulted into severe network congestion that generated packet losses and delay fluctuations. This CBR flow was active between the time instants of 2 and 6 seconds. The QCIF sequence Claire was encoded at 30 fps with the H.264/AVC JM12.2 software [17]. We used two B frames between two successive P frames (i.e. IBBPBBP...), while we set the GOP size to 64 frames. The QP was set equal to 21 in order to get constant video quality. With this frame size and QP, the encoded P frames correspond to three RTP packets. Since the duration of the sequence was short (300 frames), the initial startup delay at the playback buffer was fixed at 100 ms. In the case of an underflow, re-buffering is performed until four frames are received. We compared our system with an adhoc method for streaming with TCP that prioritizes P frames with earlier deadline in case of reduced channel rate, while B frames are dropped [18], [19].

A. Results

We present the throughput of a TCP connection in Fig. 4(a), while Fig. 4(b) presents the sequence number trace at the TCP sender. During this TCP connection the sender transmitted the video stream continuously to the receiver. TCP timeout (TO) events happened three times during this session as it can be seen in Fig. 4(b). The ratio of blocked versus lost packets during these three TO events is as follows: 7/12, 3/8, 4/6. This result is indicative of the TCP behavior that suffers from more lost than blocked packets when the congestion event is aggressive. The results for the video quality at the decoder are present next.

Fig. 5 depicts the corresponding instantaneous PSNR at the decoder for both the *adhoc* method and TcpDiO

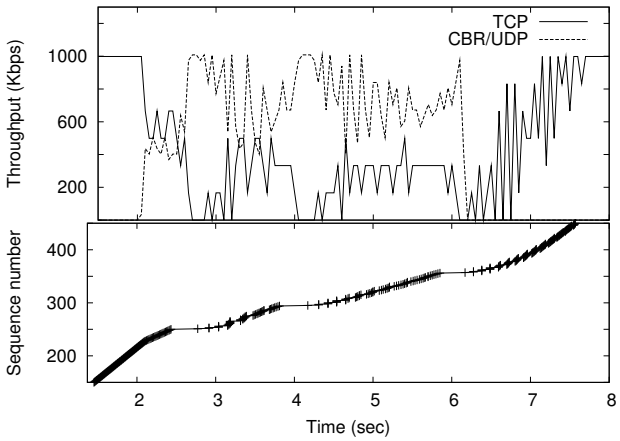

Fig. 4. Throughput and sequence number traces vs. time for the TCP connection.

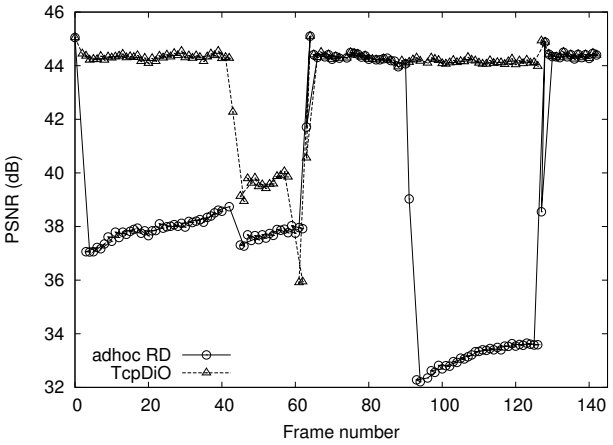

Fig. 5. PSNR at the decoder vs. the frame number.

with an optimization window of $K = 40$ packets. The performance improvement is in the range of 6dB for the period between the decoding of two I frames. The reason behind this improvement is that with TcpDiO, P frames are scheduled even earlier for transmission when compared to the *adhoc* method, and therefore only the loss of one P frame takes place around frame 42. However, with *adhoc* streaming two more P frames are lost earlier, that lead to faster PSNR reduction. Therefore, the key advantage of our system is that enables early identification of the segment loss at the TCP layer, and then aggressively schedules the most important P frames. With a slightly higher value for the optimization window even this P frame was received on-time with TcpDiO. The problem becomes even worse after the second I frame with *id* equal to 64 is received, since congestion becomes more severe. One re-buffering event must take place for both systems at frame 80, which means that both systems rendered the correct frames. In this case the TcpDiO scheduler prioritized the most important of the P frames and took advantage of this re-buffering event. However, with the *adhoc* method, two B frames are received during this event which means that they "steal" a spot from actually important frames. The result is that two more P frames around number 92 are lost, and this loss propagates to subsequent frames. One solution would be to execute further re-buffering.

In Fig. 6, we present results for the average number of decoding failure events that were traced during the simulations for both systems. A decoding failure is actually an event that is more insightful than the corresponding buffer underflow events. The reason is that it is defined as the event where the decoder does not have the correct reference frame for decoding on-time the received frames of lower priority. This means that more than one frames are not decoded in case of such an event and the result was to observe several "freezed frames" at the renderer. The same streaming experiment that we described before, was performed 20 times for different values of the startup delay. Furthermore, two different values of the scheduling window K were tested in order to highlight more clearly the effect of different levels of dependencies across video packets.

Two observations can be done based on the results Fig. 6. First, the number of decoding failures is considerably lower for the proposed system when the scheduling window K is equal to 40. For the higher value of K equal to 80, the results are even better for TcpDiO due the increased scheduling and prioritization options that the scheduler has. Key characteristic of our algorithm is that it is able to minimize such events because it is designed precisely for this purpose. With the proposed method, when a buffer underflow event happens, and a frame misses the prescribed playback deadline, this has a minimum impact on the decoding of future frames unlike the *adhoc* method. We also observed that the *adhoc* algorithm must increase buffering nearly three orders of magnitude in order to reach similar performance with TcpDiO and compensate for the lack of optimal scheduling.

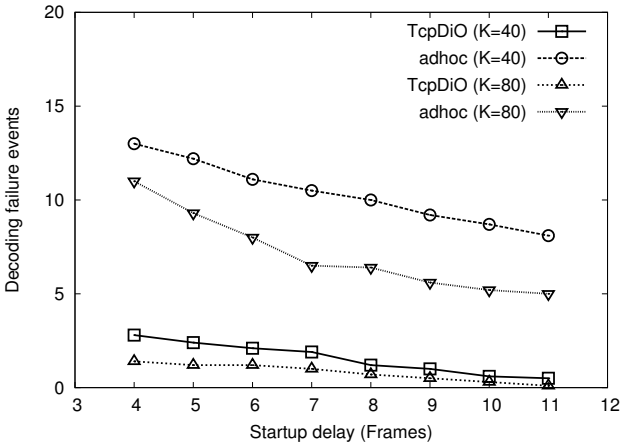

Fig. 6. Average PSNR vs. scheduling window for different GOP structures

VIII. Related Works on Media Streaming with TCP

During the last few years the research community has proposed a number optimization techniques for improving video streaming with TCP. We review the most noteworthy of these mechanisms next. One of the first mechanisms is Time-lined TCP [20], where TCP video streaming is realized by allowing the operating system to control the transmission of data that have strict deadlines. A similar approach to the previous one, can be found in [21]. In that work, the main task of the modified TCP receiver is to deliver out-of-order packets to the application regardless of when they arrive. In a receiver oriented approach [22], the authors evaluate multimedia streaming with TCP, and conclude that buffering at the client can handle the retransmission delays and the congestion control induced throughput variations of TCP. The authors in [23] formally express the minimum playback at the client, by modeling the average throughput of an TCP connection. Another interesting mechanism for TCP-based streaming can be found in [24]. The main idea that the authors had was to provide an approximate CBR channel to the streaming flow that is using TCP, through prioritization over other flows at the last mile connection of the receiver. In most recent works [18], [19], the authors propose techniques for streaming with TCP, that selectively drop frames at the sender, in order to match the available bandwidth. In our recent work [2], we have suggested the use of TCP for real-time video encoding and streaming. However, we did not consider packet scheduling at the sender but only the effect of lost packets in conjunction with error concealment at the decoder. Probably the work that is mostly related to this paper, can be found in [13] and it was analyzed earlier in this paper.

IX. Conclusions

In this paper, we developed an application-layer technique that can improve media streaming performance with TCP while avoiding modifications to the protocol itself. The solution we present is a TCP-aware and distortion-optimized (TcpDiO) scheduling algorithm for media packets. The proposed scheduling algorithm estimates the effect of the loss of each media packet on the decoder distortion, before this packet is passed to the TCP layer. Unlike existing rate-distortion optimized streaming mechanisms, the proposed scheduling algorithm derives the loss probability for a media packet, by considering the actual dynamic operation of the TCP algorithms both at the sender and the receiver. The accurate estimation of the duration and the side-effects of sudden TCP throughput disruptions by the proposed models, allows for optimal adjustment of scheduling window and minimized buffer requirements on the receiver side. Our experiments showed that decoder quality is significantly improved when compared with adhoc methods for TCP video streaming, for a given buffer size and playback delay.

References

[1] J. Postel, "Transmission control protocol," RFC 793, 1981.

[2] A. Argyriou, "Real-time and rate-distortion optimized video streaming with TCP," *Elsevier Signal Processing: Image Communication*, vol. 22, no. 4, pp. 374–388, April 2007.

[3] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video Processing and Communications*. Prentice Hall, 2002.

[4] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, August 1999.

[5] E. Kohler *et al.*, "Datagram congestion control protocol (DCCP)," draft-ietf-dccp-spec-06.txt, February 2004.

[6] L. Guo, E. Tan, S. Chen, Z. Xiao, O. Spatscheck, and X. Zhang, "Delving into internet streaming media delivery: A quality and resource utilization perspective," in *Internet Measurement Conference*, 2006.

[7] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs)," RFC 3489, March 2003.

[8] http://www.skype.com.

[9] "Real media," http://www.realnetworks.com.

[10] "Quicktime," http://www.apple.com.

[11] "Windows media," http://www.microsoft.com.

[12] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *Microsoft Research Technical Report MSR-TR-2001-35*, 2001.

[13] A. Sehgal, O. Verscheure, and P. Frossard, "Distortion-buffer optimized TCP video streaming," in *IEEE International Conference on Image Processing (ICIP)*, 2004, pp. 2083–2086.

[14] R. Stevens, *TCP/IP Illustrated Volume 1*. Addison-Wesley, 1994.

[15] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC 3550, July 2003.

[16] http://linux-net.osdl.org/index.php/Netem.

[17] "JVT reference software," Available from http://bs.hhi.de/ suehring/tml/download/.

[18] C.-F. Wong, W.-L. Fung, C.-F. J. Tang, and S.-H. G. Chan, "TCP streaming for low-delay wireless video," in *Second International Woekshop on Quality of Service in Heterogeneous Wired/Wireless Networks*, 2005.

[19] E. Gurses, G. B. Akar, and N. Akar, "A simple and effective mechanism for stored video streaming with TCP transport and server-side adaptive frame discard," *Computer Networks*, vol. 48, no. 4, pp. 489–501, July 2005.

[20] B. Mukherjee and T. Brecht, "Time-lined TCP for the TCP-friendly delivery of streaming media," in *ICNP*, 2000.

[21] S. Liang and D. Cheriton, "TCP-RTM: Using TCP for Real Time Applications," in *ICNP*, 2002.

[22] C. Krasic, K. Li, and J. Walpole, "The case for streaming multimedia with TCP," in *Workshop on Interactive Distributed Multimedia Sytems (IDMS)*, 2001.

[23] T. Kim and M. H. Ammar, "Receiver buffer requirement for video streaming over TCP," *SPIE*, vol. 6077, no. 607718, 2006.

[24] P. Mehra and A. Zakhor, "TCP-based video streaming using receiver-driven bandwidth sharing," in *International Packet Video Workshop*, 2003.