Video Packet Selection and Scheduling for Multipath Streaming

Dan Jurca, *Student Member, IEEE*, and Pascal Frossard, *Senior Member, IEEE*

***Abstract*—This paper addresses the problem of choosing the best streaming policy for distortion optimal multipath video delivery, under network bandwidth and playback delay constraints. The streaming policy consists in a joint selection of the network path and of the video packets to be transmitted, along with their sending time. A simple streaming model is introduced, which takes into account the video packet importance, and the dependencies between packets. A careful timing analysis allows to compute the quality perceived by the receiver for a constrained playback delay, as a function of the streaming policy. We derive an optimization problem based on a video abstraction model, under the assumption that the server knows, or can predict accurately the state of the network. A detailed analysis of constrained multipath streaming systems provides helpful insights to design an efficient branch and bound algorithm that finds the optimal streaming strategy. This solution allows to bound the performance of any scheduling strategy, but the complexity of the algorithm becomes rapidly intractable. We therefore propose a fast heuristic-based algorithm, built on load-balancing principles. It allows to reach close to optimal performance with a polynomial time complexity. The algorithm is then adapted to live streaming scenarios, where the server has only a partial knowledge of the packet stream, and the channel bandwidth. Extensive simulations show that the proposed algorithm only induces a negligible distortion penalty compared to the optimal strategy, even when the optimization horizon is limited, or the rate estimation is not perfect. Simulation results also demonstrate that the proposed scheduling solution performs better than common scheduling algorithms, and therefore represents a very efficient low-complexity multipath streaming algorithm, for both stored and live video services.**

***Index Terms*—Branch and bound, load balancing, multipath streaming , packet scheduling.**

I. Introduction

DESPITE the development of novel network infrastructures and constantly increasing bandwidth, Internet media streaming applications still suffer from limited and highly varying bandwidth, and from packet loss. Multipath video streaming has recently been proposed as a solution to overcome packet network limitations. It allows to increase the streaming bandwidth by balancing the load over multiple (disjoint) network paths between the media server and the clients, or on different interfaces in mobile environments. It also provides means to limit packet loss effects when combined with error resilient streaming strategies [1]. The efficiency of multipath video streaming is however tied to the packet transmission strategy, whose objective is to offer an optimal quality of service in delay-constrained video applications.

This work addresses the problem of video packet streaming in multipath network scenarios, under playback delay and buffer constraints. It aims at efficiently distributing the video information on the available network paths, while judiciously trading off playback delay and distortion at the receiver. This paper considers the selection of inter-dependent video packets to be transmitted (or equivalently the adaptive coding of the video sequence), and their scheduling on the available network paths, in order to minimize the distortion experienced by the end-user. The complex distortion optimization problem is *a priori* NP-complete, and no method can solve it in polynomial time [2]. With the help of heuristics from constrained multipath streaming scenarios, we propose a polynomial complexity algorithm for efficient video scheduling in practical scenarios.

Assuming a simple streaming model, which captures the unequal importance of video packets and their dependencies, we build on [3] and propose a detailed analysis of timing constraints imposed by delay sensitive streaming applications. This analysis allows us to identify sets of valid, or feasible transmission policies, which compete for the distortion optimized multipath streaming solution. The optimal strategy is computed based on a branch and bound algorithm [4] adapted to the multipath streaming problem. Even if this method greatly reduces the complexity of the computations compared to a full search over the policy space, there is no guarantee that it performs in polynomial time for every instance of the problem. Hence, we propose a heuristic-based approach to the optimization problem, which leads to a polynomial time algorithm, based on load-balancing techniques. This fast scheduling algorithm is finally adapted with sliding window mechanisms, to the case of real-time streaming, where the server has only a partial knowledge about the packet stream. Simulation results demonstrate close to optimal performances of the fast scheduling solution, for a large variety of network scenarios. Compared to state-of-the-art algorithms, it offers smaller quality variations on dynamic bandwidth channels, and preserves a minimal quality level by improved scheduling. Interestingly enough, the performance of the real-time scheduling algorithm stays quite consistent, even for small video prefetching windows, and for limited accuracy in the channel bandwidth prediction. This extends the validity of our algorithm to multipath live streaming systems with stringent delay constraints, and simple bandwidth prediction methods.

Manuscript received March 25, 2005; revised April 20, 2006. This work was supported by the Swiss National Science Foundation under Grant PP-002-68737. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Wenjun (Kevin) Zeng.

The authors are with the Signal Processing Institute, Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland (e-mail: dan.jurca@epfl.ch; pascal.frossard@epfl.ch).

Color versions of Figs. 3 and 5–12 are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2006.888017

The main contributions of this paper are threefold. First, we study video packet scheduling in an efficient streaming scenario, taking into account possible buffer constraints in each intermediate network nodes. Since congestion is the main cause of loss, it certainly becomes primordial to respect the buffer constraints in network nodes, in order to design efficient streaming systems. Second, we propose an optimal solution for the distortion optimization problem, which takes into account the nonstationary nature of the video sequence, the packet dependencies introduced by the encoding algorithm, and the network status. This optimal solution allows to bound the performance of scheduling algorithms. Finally, we present a novel polynomial time algorithm that provides performances similar to the optimal streaming strategy. This algorithm is eventually adapted to real-time scenarios, with more restrictive delays, and to cases where the accuracy in the prediction of the channel status is reduced.

This paper is organized as follows. A description of the related work in multipath streaming is briefly presented in Section II. Section III describes our multipath streaming model and introduces the notation used in the distortion optimization problem. The packet scheduling problem is analyzed in detail in Section IV. Based on this timing analysis, we propose in Section V both optimal and fast heuristic-based algorithms to solve the distortion optimization problem. Simulation results are presented in Section VI, and Section VII concludes the paper.

II. Related Work

Multipath video streaming has recently drawn the attention of the scientific community, due to its interesting properties for media applications [5]. Among the main benefits of using multiple paths between a media server and a client we can enumerate: 1) the reduction in correlation between packet losses; 2) increased throughput; and 3) ability to adjust to variations of congestion patterns on different parts of the network.

Ongoing research efforts are directed towards solving numerous problems that arise in multipath streaming scenarios [6], in order to optimize the quality of service of media applications. An initial experimental approach on path diversity is provided in [7]. The authors select the optimal pair of servers containing complementary video descriptions for each client while accounting for jointness and disjointness of paths and their lengths. The authors of [8] present a path diversity system with forward error correction (FEC) for packet-switched networks, while multipath streaming solutions for wireless networks are proposed in [9].

None of these works specifically addresses server-based packet scheduling in multipath scenarios, in order to adapt to dynamic bandwidth variations. Packet scheduling solutions have, however, been proposed in client-server architectures with a single channel: the authors of [10] solve an optimization-scheduling problem specific to wireless networks, using a partially observable Markov Decision Process (MDP).

In this paper, we propose a packet scheduling mechanism that works for multipath video streaming scenarios, and is fast enough to be implemented in practical scenarios. Our approach to multipath streaming is different than the previous work, since we search for optimal server-driven transmission policies for sets of sequential video packets, given the network scenario and client requirements. We do not only take advantage of the increased aggregated bandwidth of multiple network paths, but we also use the different paths to reduce the playback delay experienced by the client.

Other server-driven strategies have been proposed to adapt to channel-rate fluctuations. Frame-discard strategies have been proposed in [11] and [12]. These works address a network scenario consisting of a single path between the server and the client. When the available bandwidth is not sufficient, the streaming server finds the frames that can be discarded, in order to limit the degradation of the video quality. Branch and bound strategies have been recently proposed in [13]. The authors extend the work of [14] by providing faster algorithms for the rate distortion optimization problem. In our approach we go one step further, by considering packet dropping in a multipath environment. Our scheduling approach not only attempts to solve the problem of packet discard, but also to determine the transmission path, so that the video quality at the receiver is maximized.

The closest existing solution to the problem considered in this paper is represented by the work developed in [15]. The multipath EDPF algorithm solves the packet scheduling problem by computing the earliest delivery time for each packet, on each of the paths. By sending each packet on the path that ensures the earliest delivery at the client, the authors minimize the packet reordering cost. Later, the same authors improve their algorithm with a selective frame discard strategy that drops less important frames in case the channel bandwidth is smaller than the encoded video rate [16]. While it is, in essence, similar to our approach, the novelty introduced here resides in the handling of the video packets. We take into account a more generic encoding format (e.g., scalable coding), with improved granularity in media packets, but also more complex packet dependencies, which are important in the case of prefetch windows that are larger than one single frame. It generally allows to guarantee smoother quality variations at the decoder. Finally, our framework is also able to consider buffer limitations in overlay network nodes, and thus to prevent loss due to congestion. It will be shown that the buffer constraints may have a nonnegligible impact on the scheduling strategy in multipath scenarios.

III. Multipath Video Streaming

A. General Framework

We consider the simple multipath network topology represented in Fig. 1. The client C requests a media stream from a streaming server S, which transmits the requested bitstream via two disjoint paths. Each network path consists in two segments connected through an intermediate node that simply forwards, after a possible buffering delay, incoming packets from the first segment, towards the client on the second segment. The intermediate nodes, simply called nodes in the remaining of the paper, represent network streaming proxies, or edge servers for example. The streaming server is connected to the channels through buffer interfaces, which are modelled as first-in first-out (FIFO) queues. Thus, the channels drain the packets from the

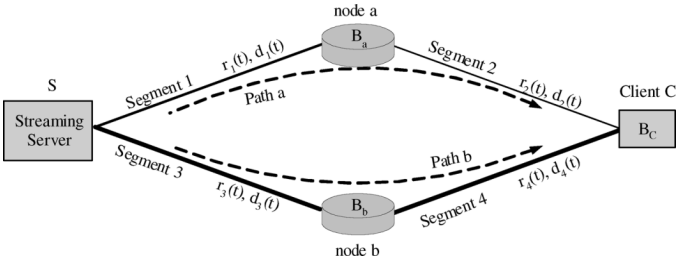

Fig. 1. Multipath streaming scenario. The client accesses the streaming server simultaneously through two different paths, each one composed of two segments with intermediate buffers.

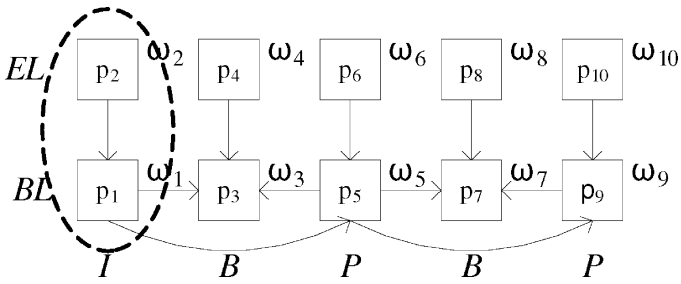

Fig. 2. Directed acyclic dependency graph representation for a typical MPEG layered-encoded video sequence (one network packet per layer, with IPBPB format).

buffers, in the same order in which the server places them into the buffers. The network channels between the server and the client are represented as variable bandwidth, lossless links. The variable nature of the bandwidth implies that the rate at which the channels drain data placed in the server's buffers, changes as a function of time. At the other end, the client waits for an initial playback delay Δ after its request for a stream. It then starts decoding the media stream, and plays it continuously. Such scenarios can be easily imagined in the context of content distribution networks, wireless video transmissions via several interfaces, or even peer-to-peer applications. The video is encoded into multiple layers adding up to a very good quality, and the available aggregated rate between the server and any client represents the share of the total link bandwidth allocated to, or reserved by the streaming application.

During the streaming session, the server selects a subset of the pre-encoded media packets to communicate to the client, taking into account the available bandwidth on the different network paths, and buffer fullness in the nodes or at the receiver. The segment bandwidth, latency and intermediate buffer fullness can be estimated at the server, or reported by various methods that are outside the scope of this paper (e.g., as in [17]). The work presented in this paper rather addresses the selection of the packets that should be communicated to the client, as well as the network path they need to follow. It actually does not even require an exact knowledge of the channel bandwidth, but accurate network information obviously increases the performance of the streaming system. Finally, the network topology could present several disjoint paths, and several nodes on each path. However, for the sake of clarity, we consider in the problem formulation only the two-path scenario presented in Fig. 1. The extension to scenarios with a larger number of paths, is straightforward.

B. Streaming Model and Notations

In the multipath streaming topology represented in Fig. 1, each network segment i is characterized by an instantaneous rate $r_i(t)$ and an instantaneous latency $d_i(t)$. The rate $r_i(t)$ is the total bandwidth allocated to the streaming application on segment i at time instant t. Equivalently, we denote the cumulative rate on segment i, up to time instant t, by $R_i(t) = \int_0^t r_i(u)du$. Additionally, the streaming server assumes that no packet is lost on the network segments, except those induced by late arrivals or buffer overflows, and that the order of the packets is not changed between two successive nodes. These assumptions are quite realistic in most of today's wired streaming networks.

The intermediate nodes $\{a, b\}$ have buffers of capacity B_a and respectively B_b, which is available for the streaming session. The client has a playback buffer of capacity B_c. We first assume that all segment rates and latencies along with intermediate buffer capacities are accurately predicted by the server at all time instants, possibly with feedback of the overlay nodes. We will eventually relax that assumption to consider real-time streaming scenarios.

The video sequence is encoded into a bitstream using a scalable (layered) video encoder. The bitstream is then fragmented into network packets under the general rule stating that 1) each network packet contains data relative to at most one video frame and 2) an encoded video frame can be fragmented into several network packets. Let $P = \{p_1, p_2, \ldots, p_N\}$ be the chronologically ordered sequence of N network packets, after fragmentation of the encoded bitstream. Each network packet p_n is characterized by its size s_n in bytes, and its decoding timestamp t_n^d. From the client viewpoint, all the video packets are not equivalently valuable, due to the nonstationary nature of the video information. Therefore, each network packet can be characterized by a weight ω_n, which represents the reduction in the distortion perceived by the client, in the case where packet p_n is successfully decoded. We refer to a successfully decoded packet as a network packet that is received and correctly decoded by the client before its decoding deadline.

Additionally, in most video encoding schemes, packets generally have dependencies between them. In other words, the successful decoding of one packet p_n is contingent on the successful decoding of some other packets, called *ancestors* of p_n. The successful decoding of one packet may depend on the correct decoding of several ancestors, and we denote by A_n, the set of ancestors of packet p_n. Such dependencies can be represented by a directed acyclic dependency graph [14], as shown in Fig. 2. The nodes in the graph represent the network packets and are characterized by their individual weights, and directed edges represent dependencies between packets and their ancestors.

We denote by $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$ the transmission policy adopted by the streaming server. The policy π_n used for packet p_n consists in a couple a variables $[q_n, t_n^s]$ that respectively represent the action q_n chosen for packet p_n, and its sending time t_n^s. It completely characterizes the server behavior with respect to packet p_n under the general policy vector π. In the multipath network scenario presented above, the server can decide to send packet p_n on paths a or b, or simply to drop the packet without

sending it. Therefore, the action imposed on packet p_n can be written as

$$q_n = \begin{cases} a, & \text{if packet } p_n \text{ is sent on path } a \\ b & \text{if packet } p_n \text{ is sent on path } b \\ 0, & \text{if packet } p_n \text{ is dropped.} \end{cases}$$

Let Π be the set of all the feasible policies π. In our streaming model, a packet is decoded by the receiver only if its arrival time, t_n^c, is smaller than its decoding deadline, i.e., if $t_n^c \leq t_n^d + \Delta$ where t_n^d represents the decoding timestamp of packet p_n, and Δ is the playback delay at client. We assume here, without loss of generality, that the client request has been sent at time $t = 0$, and that the decoding timestamp of the first packet p_1 is set to 0. The processing time at the receiver is further neglected. Remember that packets are sent sequentially on a path, and that the streaming strategy aims at avoiding buffer overflows that would result in packet loss. Under these assumptions, and taking into account packet dependencies, the successful decoding of a packet p_n under the streaming strategy $\pi \in \Pi$, can be represented by the binary variable $\varphi_n(\pi)$, where $\varphi_n(\pi)$ is equal to 1 if the packet arrives on time at the decoder, and if all its ancestors have been successfully decoded. We further take into account the difference between frame order in the bitstream and the decoding order of the frames at the client. This impacts, for example, the scheduling of a B frame that is placed in the bitstream after the future P frame it depends on. In other words, we can write

$$\varphi_n(\pi) = \begin{cases} 1, & \text{if } \begin{cases} q_n \neq 0 \\ t_n^c \leq t_n^d + \Delta \\ \varphi_m(\pi) = 1, \forall p_m \in A_n \\ \text{at time } t_n^d + \Delta \end{cases} \\ 0, & \text{otherwise.} \end{cases}$$

The overall benefit Ω of the streaming strategy $\pi \in \Pi$, which is equivalent to the quality perceived by the receiver, can now simply be expressed as the sum of the weights ω_n of all successfully decoded packets. We assume that packets whose $\varphi_n(\pi) \neq 1$ are simply discarded at the client, hence the overall benefit can be written as $\Omega(\pi) = \sum_{\forall n:\varphi_n(\pi)=1} \omega_n$.

C. Distortion Optimization Problem

Given the abstraction model of the encoded video bitstream, the distortion optimization problem consists in an efficient selection of the subset of video packets to be transmitted, jointly with their streaming policy. We assume a server-driven scenario in which the server is aware of, or can estimate the network conditions (i.e., $r_i(t)$ and $d_i(t)$), at each time instant. The server then only schedules for transmission packets that can arrive at the client before their decoding deadline. The streaming server considers that the transmission links are lossless, and that packet loss only happens due to buffer overflow, or late arrival.

The distortion optimization problem can be stated as follows: *Given* P, the packetized bitstream of an encoded video sequence, Δ, the maximum playback delay imposed by the client, and the network state, *find* the optimal transmission policy $\pi^* \in \Pi$ that maximizes the overall quality measure Ω. The optimization problem translates into finding $\pi^* \in \Pi$ s.t.

$$\Omega(\pi^*) = \max_{\pi \in \Pi} \sum_{\forall n:\varphi_n(\pi)=1} \omega_n.$$

The optimization problem can be easily reduced to the more general case of optimal scheduling problems. This family of problems proves to be NP-complete [2] and an optimal algorithm that solves them in polynomial time does not exist. Hence, we still propose in this paper an optimal algorithm that efficiently finds the distortion minimal streaming strategy for long video sequences, to be used as a benchmark for faster, suboptimal methods. We then design a heuristic-based algorithm that provides close to optimal performance, but in polynomial time, and we eventually apply it to real-time streaming scenarios.

IV. Packet Scheduling Analysis

A. Unlimited Buffer Nodes

This section proposes an in-depth analysis of the scheduling of packets in the streaming model described above, and computes the parameters necessary to solve the distortion optimization problem. Our approach represents a segment-by-segment analysis of the network behavior, including intermediate nodes buffers. This approach is a first step towards a more comprehensive analysis of network behavior related to the specificities of video streaming applications. In general, the particular characteristics of media packet streams, like timing issues or unequal importance of data, prevent the application of general end-to-end analysis like [18] in such scenarios.

We consider first the case where buffering space in the network nodes and the client is not constrained, i.e., $B_a = B_b = B_c = \infty$. The server has the knowledge of N video packets, where N can be the total number of network packets of the video stream (in the case of stored video), or simply the number of packets contained in the prefetch window in real-time streaming. The server is able to transmit network packets simultaneously on the two network paths. Under the assumption of unlimited buffer space, the server can send packets on each of the paths at the maximum rates of the first segments ($r_1(t)$ for path a or $r_3(t)$ for path b—see Fig. 1).

Under a given policy π, the sending time t_n^s of each packet p_n can thus be easily computed. Suppose that p_n is sent on path a (i.e., $q_n = 1$). Let $S_n^a(\pi) = \sum_{m<n, q_m=1} s_m$ represent the cumulative size of all the packets that need to be sent on path a before p_n, under the policy π. Under the assumption that the available bandwidth is fully utilized by the streaming application, t_n^s is the shortest time t at which the cumulative rate $R_1(t)$ is larger than S_n^a

$$t_n^s(\pi) = \arg\min_t \left| R_1(t) - S_n^a(\pi) \right|. \tag{1}$$

In other words, the packet p_n can only be sent when all the previous packets scheduled on the same path have been transmitted. It will then arrive at the client after a certain

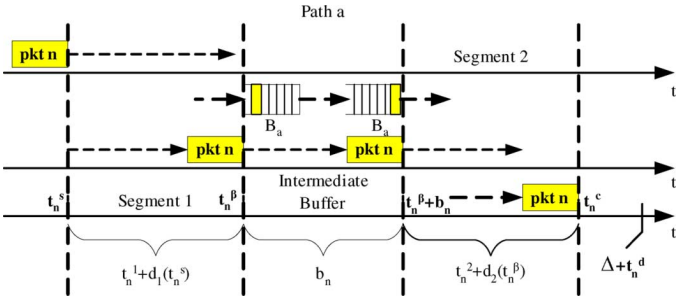

Fig. 3. Time diagram for packet p_n sent on path a.

delay, caused by the transmission delays (t_n^1 and t_n^2) on the two segments that compose path a, the latencies introduced by the two links ($d_1(t)$ and $d_2(t)$) and the queuing time at the node b_n. Therefore, the time instant at which packet p_n enters the node buffer can be expressed as $t_n^\beta = t_n^s + t_n^1 + d_1(t_n^s)$. The arrival time of packet p_n at the client, can be written as $t_n^c = t_n^\beta + b_n + t_n^2 + d_2(t_n^\beta)$. The timing representation of the transmission of packet p_n is provided in Fig. 3.

The transmission delays t_n^1 and t_n^2 represent the time needed to send packet p_n, at the bandwidth available on path a. They have to verify $R_1(t_n^s + t_n^1) - R_1(t_n^s) = R_2(t_n^\beta + t_n^2 + b_n) - R_2(t_n^\beta + b_n) = s_n$, and can be computed similarly to (1). The queuing time b_n corresponds to the time needed to transmit the $B(t_n^\beta)$ bits present in the buffer, at time t_n^β when packet p_n enters the buffer. The buffer fullness can be computed recursively as $B(t_n^\beta) = \max[B(t_{n-1}^\beta) + s_{n-1} - R_2(t_n^\beta) + R_2(t_{n-1}^\beta), 0]$.

Therefore, the queuing time can be computed such that it satisfies $R_2(t_n^\beta + b_n) - R_2(t_n^\beta) = B(t_n^\beta)$. Note that, even if the previous development only considers the path a, the extension of the analysis to the packets transmitted over path b is straightforward. The arrival time of packet p_n, t_n^c is thus fully determined. The minimal playback delay $D(\pi)$ induced by the transmission policy π can finally be expressed as

$$D(\pi) = \max_{1\le n\le N}(D_n(\pi)) = \max_{1\le n\le N}(t_n^c - t_n^d)$$

where $D_n(\pi)$ is the playback delay imposed by the streaming process up to packet p_n under the transmission policy π. Interestingly, the playback delay is a nondecreasing function of the packet number n. That property, expressed in Lemma 1, will be advantageously used in the scheduling optimization problem.

Lemma 1: Given that the streaming server sends the N network packets in parallel on two paths, and that on each path the packets are sent sequentially, the playback delay $D_n(\pi)$ under the given policy vector π is a nondecreasing function of n.

Proof: [Sketch] Observe that $D_n(\pi)$ can be expressed as a recursive function of n:

$$D_n(\pi) = \max(D_{n-1}(\pi), t_n^c - t_n^d). \tag{2}$$

Hence, $D_i(\pi) \le D_n(\pi), \forall n, \forall i$ such that $0 \le i \le n \le N$, with $D_0(\pi) = 0$ and $D(\pi) = D_N(\pi)$. ■

Let us finally define the cumulative quality $\Omega(\pi)$, resulting from the streaming policy π. In a perfect transmission where the set of packets P is entirely transmitted, the quality is denoted by $\Omega_0(\pi) = \sum_{n=1}^N \omega_n$. Due to delay or bandwidth constraints, the server may decide to drop some packets from P. In this case, we iteratively compute the cumulative quality, $\Omega_n(\pi)$, which is decremented each time a packet is dropped. It can be written as

$$\Omega_n(\pi) = \begin{cases} \Omega_{n-1}(\pi), & \text{if } \varphi_n(\pi) = 1 \\ \Omega_{n-1}(\pi) - \omega_n, & \text{otherwise} \end{cases} \tag{3}$$

with $\Omega(\pi) = \Omega_N(\pi)$. While (3) does not explicit the influence of other packets that have packet p_n as their ancestor, the status $\varphi_n(\pi)$ of packet p_n directly affects the status of all packets dependent on p_n.

Lemma 2: Ω_n is a nonincreasing function of the packet number n.

Proof: [Sketch] Observe that ω_n is by definition a non negative value. Hence, $\Omega_n \le \Omega_i, \forall n \le N, \forall i \le n$. ■

The two properties expressed in Lemmas 1 and 2 are used later in the derivation of efficient search algorithms for the optimal scheduling policy.

B. Constrained Buffer Nodes

A similar timing analysis can be performed in the case where the buffering space in the intermediate nodes on each path is limited to B_a and B_b respectively. The buffer capacities in the intermediate nodes may significantly influence the optimal packet scheduling strategy in multipath streaming scenarios. In contrary to single path scenario, the overall packet scheduling is not necessarily sequential any more, which allows to use buffers as a form of staging step. Buffers allows for smoothing bandwidth fluctuations between successive path segments, when delay constraints permit it.

We reasonably assume that the buffering space is larger than any video packet in P. B_a and B_b represent the buffer sizes allocated by the intermediate nodes to the streaming process and they are known by the server. The server estimates the buffer fullness based on its knowledge about the network bandwidth, or with help of feedbacks from intermediate overlay nodes. It tries to avoid buffer overflows by adapting the sending time of each packet to the buffer fullness. Note that it may no longer use the full available bandwidth, without risking loss of packets.

The streaming policy has to take into account these new constraints. In particular, if packet p_n has to be transmitted on path a under policy π, its sending time t_n^s is such that there is enough buffer space available when it reaches the intermediate node. Additionally, the packet p_n can only be sent when all the previous packets on the same path have been transmitted. Using the same notation as defined above, t_n^s becomes the smallest value that simultaneously verifies the following conditions:

$$\begin{cases} R_1(t_n^s) \ge S_n^a(\pi) \\ t_n^s + t_n^1 + d_1(t_n^s) \ge \tau_n \end{cases} \tag{4}$$

where τ_n represents the earliest time at which there is enough space in the intermediate buffer to receive packet p_n, when the buffer is drained at a rate $r_2(t)$. Equivalently, τ_n can be computed recursively, since it verifies the inequality $B_a - (B(t_{n-1}^\beta) + s_{n-1} - R_2(\tau_n) + R_2(t_{n-1}^\beta)) \ge s_n$. We can also define the maximum buffer occupancy during the whole

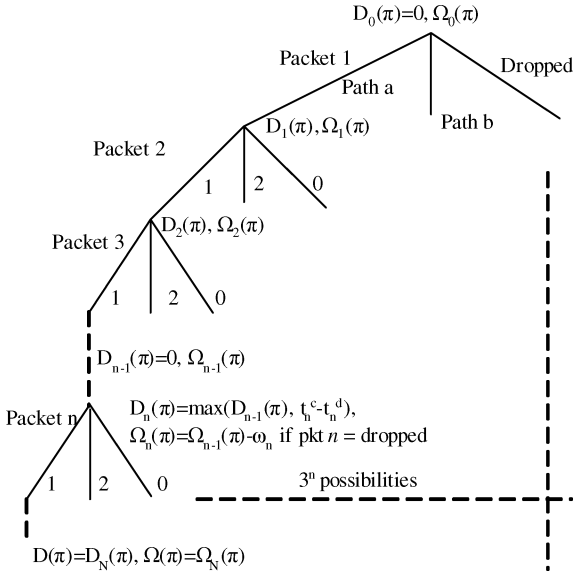

Fig. 4. Depth first B&B algorithm.

streaming process as $B_a^{\max}(\pi) = \max_{1 \le i \le N}(B(t_i^{\beta})) \le B_a$. The timing analysis on path b follows immediately. The strategy π is thus completely defined, and we can compute $D(\pi)$ and $\Omega(\pi)$ similarly to the case with unlimited buffers. A similar reasoning can be applied in order to prevent buffer overflow at the client, in the case where the client also has a limited storage space.

V. Distortion Optimized Streaming

A. Optimal Solution: Depth-First Branch & Bound (B&B)

Since the sending and arrival times for each packet p_n can be computed for a given transmission policy π (see Section IV), we can now search for the optimal packet scheduling π^* that maximizes the client video quality given an imposed playback delay. We first present an efficient algorithm that finds the optimal transmission policy vector π^* for a given encoded video sequence, network topology and playback delay. While being too complex to implement in practice, the algorithm is used as a performance benchmark for the development of sub-optimal, faster scheduling methods. The novelty of the algorithm resides in the use of B&B methods [19] in a multipath video-streaming framework[1], and on adapting pruning rules to the specific characteristics of this scenario. The pruning rules make the algorithm much faster than a brute search but still do not guarantee polynomial execution times on all streaming scenarios. The optimization problem still has a combinatorial complexity.

The scheduling of N packets on two available paths can be organized as a decision tree of depth N (Fig. 4). At each stage n in the tree, packet p_n can be sent on path a, on path b, or can be dropped. Hence, at depth N, the decision tree will contain 3^N leaves, according to the number of scheduling possibilities of the N packets on the two paths. At each stage n in the tree we can compute $D_n(\pi)$, the minimum playback delay and $\Omega_n(\pi)$, the cumulative video quality measure, for a partial scheduling up to packet p_n, according to the recursive (2) and (3), presented in Section IV. This computation can be done for each one of the valid scheduling policies, for the first n packets. As mentioned in Section IV-A, $D_n(\pi)$ and $\Omega_n(\pi)$ are nondecreasing, and respectively nonincreasing functions in n. These two functions are used to establish a fast search on the decision tree for the optimal transmission policy vector π^*. A depth-first search is performed on the decision tree, starting with an initial policy vector π that satisfies the delay constraint $D(\pi) \le \Delta$, where Δ is the playback delay imposed by the client. The policy π becomes our initial optimal policy π^* with $\Omega^* = \Omega(\pi^*)$. The initial policy is computed using a simple Earliest Delivery Path First (EDPF) algorithm with a complexity of $O(N)$, similar to [15]. The EDPF algorithm schedules frames in a FIFO order. Packets belonging to a given frame are scheduled according to their importance ω_n, on the path that guarantees the earliest arrival time at the client. If a packet cannot be successfully scheduled, it is dropped without transmission, along with all his children packets, to avoid waste of network resources.

Since an EDPF strategy is often sub-optimal in a multipath scenario, we start searching the decision tree for better transmission policies, with $\Omega > \Omega^*$. We start with the leftmost transmission policy represented on the tree (equivalent to sending all packets on path a) and move through the decision tree towards right. For each new policy π', we compute $D_n(\pi')$ and $\Omega_n(\pi')$ successively for $n = 1 \ldots N$. At any packet p_n for which $D_n(\pi') > \Delta$ or $\Omega_n(\pi') \le \Omega^*$, the computation of $D_n(\pi')$ is stopped, and the decision tree is pruned for all policies that have the same scheduling up to packet p_n (i.e., the policies $\{\pi\}$ s.t. $\pi_i = \pi'_i, \forall i, 1 \le i \le n$). If $D_N(\pi') \le \Delta$ and $\Omega(\pi') \ge \Omega^*$, the policy π' becomes the new optimal policy π^* and $\Omega^* = \Omega(\pi')$. The operation is repeated until the set of all feasible policies Π represented on the decision tree has been covered. When the search is complete, the optimal policy π^* maximizes the video quality at the receiver and respects the playback delay constraints.

The B&B method provides an efficient way of computing the optimal transmission policy vector π^*. The speed of the method depends on the pruning efficiency, which in turn depends on the quality of the initial policy. However, the method is not scalable with N, since it cannot compute the optimal solution in polynomial time. The worst case complexity of the method remains $O(3^N)$. The extension of the algorithm to more paths follows easily. In the general case of K independent network paths between the streaming server and the client, the complexity grows to $O((K+1)^N)$.

B. Heuristic Solution: Load-Balancing Algorithm (LBA)

Since the B&B algorithm may be too complex in practice, this subsection now presents a heuristic approach, which finds a close-to-optimal solution in polynomial time. The algorithm is inspired from load-balancing techniques, which proved to be very effective in solving problems of task scheduling in multiprocessor systems [20]. In short, the algorithm performs a greedy scheduling of the most valuable packets first. Less valuable packets are scheduled only if the network capacity permits, and only if they do not lead to the loss of a more

[1]While B&B techniques have been used for years by the optimization community, they have only recently been employed in a streaming scenario [3], [13].

valuable packet already scheduled (due to subsequent late arrivals at the client).

First, the N network packets are arranged in descending order of their weight. Hence, we obtain a new representation of the encoded bitstream, $P' = \{p'_1, p'_2, \ldots, p'_N\}$, such that $\omega_1(p'_1) \geq \omega_2(p'_2) \geq \cdots \geq \omega_N(p'_N)$. Then, similarly to the EDPF algorithm, a greedy algorithm (see Algorithm 1), schedules the N ordered packets on the two network paths, while additionally taking care of the packet interdependencies. Algorithm 1 presents the sketch of the complete algorithm, where, for the sake of clarity, we redefine the action imposed on packet p'_n as:

$$q'_n = \begin{cases} a, & \text{if packet } p'_n \text{ is sent on } path\ a \\ b, & \text{if packet } p'_n \text{ is sent on } path\ b \\ 0, & \text{if packet } p'_n \text{ is dropped without sending} \\ \infty, & \text{if packet } p'_n \text{ is not scheduled yet.} \end{cases}$$

Algorithm 1 LBA for finding π

Require: $P, \omega_n, s_n, 1 \leq n \leq N$
Ensure: Suboptimal transmission policy vector π;
1: **Initialization**: Create P': arrange packets in order of weight ω_n;
$n := 1$;
2: **while** $n \leq N$ **do**
3: **if** Packet p'_n s.t. $q'_n = \infty$ **then**
4: invoke $\mathrm{Schedule\_Packet}(n)$;
5: **end if**
6: $n := n + 1$;
7: **end while**
8: **Procedure:** $\mathrm{Schedule\_Packet}(n)$
9: **for all** packets p'_k in A_n s.t. $q'_k = \infty$ **do**
10: invoke $\mathrm{Schedule\_Packet}(k)$;
11: **end for**
12: invoke $\mathrm{do\_Schedule}(n)$;
13: **Procedure:** $\mathrm{do\_Schedule}(n)$
14: **if** $\exists$ packet $p'_k \in A_n$ s.t. $q'_k = 0$ **then**
15: $q'_n = 0$;
16: return;
17: **else**
18: attempt the insertion of packet p'_n on path a and on path b, ordered according to the decoding deadlines, without compromising the decoding of any other scheduled packet;
19: **if** $t^c_n(\text{path } a), t^c_n(\text{path } b) \leq t^d_n + \Delta$ **then**
20: choose the path with shorter t^c_n;
21: set q'_n accordingly;
22: **else**
23: **if** $t^c_n(\text{path } a), t^c_n(\text{path } b) > t^d_n + \Delta$ **then**
24: $q'_n = 0$;
25: **else**
26: schedule packet p'_n on the path with $t^c_n \leq t^d_n + \Delta$;
27: set q'_n accordingly;
28: **end if**
29: **end if**
30: **end if**

To decide which action to take on each packet p'_n, the algorithm first attempts to schedule all ancestors that have not been scheduled yet. If one of them cannot be scheduled, then the algorithm automatically drops the packet p'_n. This ensures that our algorithm does not waste network resources on transmitting network packets that cannot be correctly decoded at the receiver.

All packets marked to be scheduled on a given path, are reordered according to their decoding deadlines before transmission. When a new packet is inserted, it triggers a new packet ordering. If a packet p'_n can be scheduled on both network paths without interfering with the packets already scheduled, the algorithm will choose the path that offers the shortest arrival time for packet p'_n. If packet p'_n can only be scheduled on one path, the algorithm will insert the packet on that path. Otherwise packet p'_n cannot be scheduled on any of the two paths, without interfering with the already scheduled packets, and the algorithm will drop packet p'_n without transmitting it. Hence, the algorithm prevents that the transmission of one packet forces the loss of a more important packet previously scheduled, because of late arrival at the client. Note that in the case where the value of each network packet is directly proportional to the size of the packet, the algorithm offers a real load-balancing solution for the two network paths.

Algorithm 1 performs an initial ordering of the N packets in the new set P'. Any common sorting algorithm that works with complexity $O(N \log N)$ can be employed. Afterwards, for each packet p'_n that must be scheduled, the algorithm requires a search among the packets already scheduled on each of the paths, in order to insert the new packet according to its decoding deadline. The operation requires $O(N)$ computations and is repeated N times, for each packet in P'. The complexity of the proposed algorithm is thus $O(N^2)$. For the more general case of K disjoint paths between the server and the client, the algorithm requires the computation of arrival times on all the paths, for all scheduled packets. The insertion of one packet therefore requires $O(KN)$ operations, and is performed for all N packets. The total complexity of Algorithm 1 grows linearly with the number of network paths, being of $O(KN^2)$. In conclusion, the proposed heuristic algorithm has a complexity that grows linearly with the number of network paths K, and quadratic with the number of video packets N. However, it generally leads to suboptimal strategies due to the greedy optimization strategy. The extensive simulations presented in the next section show that the performance are nevertheless very close to optimal. The combination of efficiency and low complexity makes Algorithm 1 a suitable solution for fast multipath packet scheduling, especially beneficial in real-time video streaming.

C. Real-Time Streaming: Sliding Window Approach

We now relax the assumptions of full knowledge of media packets and channel bandwidths, and we present the adaptation of the hereabove algorithms to the case of live streaming. In this case, the server does not anymore have the knowledge of the complete video sequence. Instead it receives the network packets directly from an encoder. The server may buffer live streams for δ seconds, in order to increase the scheduling efficiency. It has therefore a limited horizon, which we call the prefetching time δ. In other words, the prefetching time, or prefetching window, refers to the look-ahead window employed by the server. At any given time t, the server is therefore aware

only of the network packets $\{p_n\}$ with decoding time-stamps $t_n^d \leq t + \delta$.

We assume that $N(t)$ is the number of packets that are available at the server at time t, and that $P(t) = \{p_1, p_2, \ldots p_{N(t)}\}$ now represents the set of these packets ordered according to their decoding deadlines. $N(t)$ is equal to the number of packets containing data from the video sequence up to time $t+\delta$, minus the packets that were already transmitted to the client in the time interval $[0, t]$. Note that we use the terms of prefetching and sliding window interchangeably, as referring to the same concept.

The previously defined B&B and LBA methods are now applied on the set $P(t)$ in order to compute a transmission policy vector π for the $N(t)$ packets under consideration at time t. Neglecting the computation time, even for the B&B method, we can start transmitting the packets on the two paths according to the policy π, at time t. Let T be the time interval between two successive video frames, and without loss of generality, let t and δ be multiples of T. Hence, $t + \delta = kT$. At time t, the server can send packets that contain data from the encoded video sequence up to frame k. At time $t + T$, the packets containing data from frame $k + 1$ will be available at the server. At this time, the server will stop the transmission process of all packets from the previous sliding window that have not been sent yet, and add them to the new sliding window, along with the new packets from frame $k+1$. B&B and the LBA methods are then applied on the new sliding window. The implementation of our algorithms on top of a sliding window mechanism adapts the scheduling to new packets, as soon as they are available at the server.

It is worth mentioning, that in the case of real-time video streaming, Algorithm 1 is equivalent to a sequential greedy packet scheduling algorithm that considers first the most important packets in the sliding window, while for a sliding window of just one frame, our LBA method in essence reduces to the EDPF algorithm, enhanced with a packet discard strategy [16].

Interestingly, the LBA algorithm has the same behavior even in the case when the exact weights of each packet, w_n, are not known. It suffices to know only the relative ordering of the video packets according to their weight, along with the packet dependencies. While computing online the exact weight of each packet might be difficult (especially in realtime streaming scenarios), the relative ordering of the packets can be easily performed, since it is generally accepted that an I frame packet is more important than a P or a B frame packet, and a base-layer packet is more important than an enhancement layer packet. At the same time, the packet dependencies are known from the encoding and packetization processes.

These observations emphasize the low complexity of our proposal. We argue that, due to its low complexity, the LBA algorithm can be implemented at a real-time streaming server. The LBA algorithm presents a complexity that depends on the number of frames scheduled (N) and the size of the sliding window. Its complexity, C, varies according to $C = 2(\delta/frame\_rate)^2(N - \delta/frame\_rate)$. Along with any simple bandwidth prediction mechanism able to estimate the bandwidth for the duration of the sliding window, it provides a valuable algorithm for any practical multipath streaming scenario. We demonstrate the good performance of the live streaming algorithm in Section VI, where it is compared to long horizon-scheduling mechanisms.

VI. Simulation Results

A. Simulation Setup

This section now presents and discusses the performance of the proposed scheduling algorithms, and compares the heuristic-based solution to the optimal performance bound, in both stored video scenarios and live streaming services. Video sequences are compressed with an MPEG4-FGS [21] encoder, at 30 fps with various GOP structures. We use two different CIF sequences, *foreman* and *news*, encoded in one base layer BL, and one or two enhancement layers (EL1 and EL2). Each encoded frame is split into network packets, one for each encoded layer. We set the weights ω_n of the packets as a function of their relative importance to the encoded bitstream (depending on the type of encoded frame, I, P, or B, and on the encoded layer they represent, BL, EL1, or EL2), as illustrated in Fig. 2.

We simulate network scenarios containing two and three disjoint paths between the server and the client. We conduct experiments for segment bandwidths which vary in time, for the theoretical case when the server knows them in advance, or when it predicts them based on past values. We experiment stored or live streaming scenarios, with a limited prefetch window. Finally, we consider unlimited client buffers, and negligible network latencies (i.e., $d_i(t) = 0, \forall i, \forall t$). We compare the performance of the proposed algorithms to the one of EDPF [15]. We also compare to a simple Round Robin algorithm, which greedily schedules video packets in a FIFO order, according to the available bandwidth on each of the paths. Finally, we also test our algorithm in scenarios with packet loss, in order to evaluate its behavior in very adverse conditions.

B. Stored Streaming Scenarios

The proposed algorithms are first compared in the case of stored video scenarios, where the whole sequence is available at the streaming server, before running the scheduling algorithms. The two sequences are encoded into a BL of 300 *kbps* and 450 *kbps* respectively, and one EL of 550 *kbps*. Due to the high complexity of the B&B algorithm, which computes the performance upper-bound, we use a GOP of six frames, with one B frame between P frames. In a first approximation, we choose the following packets weights: $\omega_i = 5$, for an I frame base-layer packet, $\omega_i = 4$, for the base layer of the first P frame, $\omega_i = 3$, for the base layer of the second P frame, $\omega_i = 2$, for the base layer of B frames, and $\omega_i = 1$, for enhancement layer packets.

Fig. 5 presents the video rate trace at the decoder, when the server schedules the network packets according to the optimal B&B method, the LBA algorithm, the EDPF algorithm [15], and Round Robin. The segment bandwidths are set to $r_1 = 300\,\text{kbps}$, $r_2 = 500\,\text{kbps}$, $r_3 = 400\,\text{kbps}$, and $r_4 = 100\,\text{kbps}$, the intermediate buffers are unlimited, and the maximum playback delay imposed by the client is set to $\Delta = 150\,\text{ms}$.

It can be observed that, while the proposed LBA algorithm manages to successfully schedule almost the same number of

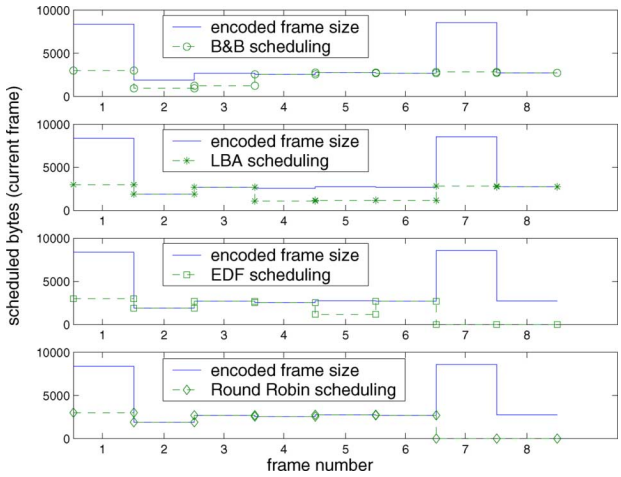

Fig. 5. Packet scheduling obtained by the B&B, LBA, EDPF, and simple Round Robin algorithms for an IBPBPBIB frame sequence (*foreman_cif* sequence).

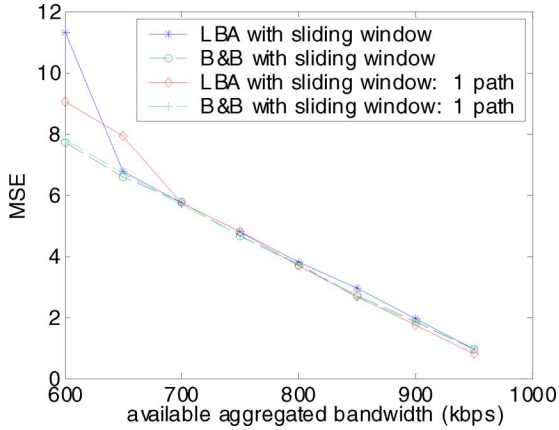

Fig. 6. Mse values between the original encoded sequence and the scheduled one (100 frames).

packets as the optimal B&B solution, the simple EDPF algorithm and the Round Robin method have clearly worse performance since they mostly drop the end of the sequence. This is due to the fact that the proposed LBA algorithm makes sure that the most important packets (the packets from the base layer starting with the *I* frames, then *P* and *B* frames) can be scheduled, and only afterwards adds the enhancement layer packets, if the network rate permits it. On the contrary, the EDPF or Round Robin algorithms schedule as much as possible from any frame, without taking into account future frames. In this way, entire GOPs could be lost, because packets of the *I* frames cannot meet the decoding deadline at the client.

C. Streaming With Limited Look-Ahead

The proposed solutions are now compared in the case of live video streaming, where the server knowledge is limited to the packets within the prefetching window. The prefetching window is set to three frames (i.e., $\delta = 100$ ms), the maximal playback delay is $\Delta = 100$ ms, and the bandwidths of the four network segments are constant in time.

The algorithms are compared in terms of the mean square error (MSE) perceived at the receiver. Fig. 6 presents the distortion due to the network bandwidth constraints, computed between the original encoded video sequence and the sequence available to the client. The MSE values obtained by the real-time B&B and LBA scheduling algorithms on two paths (with equal rates) are compared to the ones obtained by using a single network path with equivalent aggregated bandwidth. The decoder in this case implements a simple error concealment strategy based on previous frame repetition. Both schemes perform quite similarly when the aggregate bandwidth becomes large. We observe that, while the multipath scenario does not require a large bandwidth network path, there is virtually no loss in video quality when using two parallel network paths, instead of a single high-bandwidth channel. This proves the efficiency of the proposed algorithms, relatively to the distortion lower-bound provided by the single-channel scenario. Obviously, multipath streaming is useful when there is no single high bandwidth channel available, which is used here only to assess the scheduling policy performance. Note that the EDPF algorithm is voluntarily omitted here due to the high MSE values it induces when it fails to schedule entire frames or GOPs.

TABLE I
ALGORITHM COMPARISON WITH SLIDING WINDOW.

r_1	r_2	r_3	r_4	B&B	LBA	B&B SW	LBA SW
200	300	400	400	75.8%	65.5%	70.4%	65.5%
300	300	100	200	50.6%	47%	44.9%	47%
300	300	200	200	64%	60.8%	60.6%	60.8%
250	300	200	300	57.6%	51.4%	5.1%	51.5%
300	300	250	300	71%	60.8%	69.7%	60.8%

The algorithms are also compared in terms of the proportion of transmitted information, for different network conditions, in Table I. The values represent the percentage of successfully decoded data at the client, out of the full stream. Interestingly enough, the real-time LBA algorithm has a similar performance to the case of stored video scenarios. The sliding window, even with low prefetch time, does not significantly affect the behavior of the scheduling algorithm. This property, along with the low complexity of the algorithm, shows that LBA represents a valid solution to the multipath packet-scheduling problem, in the case of live streaming.

We now analyze the influence of the Sliding Window size on the LBA packet scheduling process. As seen before, in the case of constant link rates, the packet scheduling process is barely influenced by the size of the sliding window. However, it is not the case if we allow the link rates to vary in time. We tested the performance of the LBA algorithm with various sizes for the sliding window. We use the *foreman_cif* sequence (the first 100 frames) and variable network rates on small time scales (hundreds of milliseconds). We omit the results of the B&B algorithm due to the intractability of the computations for larger window sizes, and those of the EDPF scheduling, since it does not take into account the sliding window size.

We present the MSE results in function of the size of the sliding window, for various network rate sets of different aggregated average bandwidths (Fig. 7). We can observe that, for small sliding windows, the LBA algorithm behavior is close to the one of the EDPF algorithm, which may lose entire GOPs.

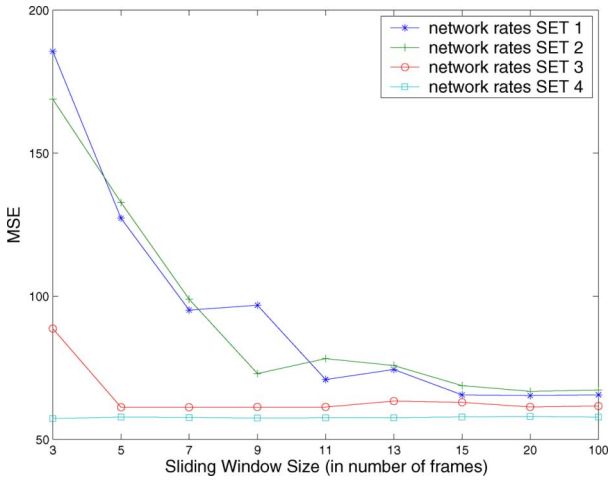

Fig. 7. Mse values for different network rate sets as a function of sliding window size.

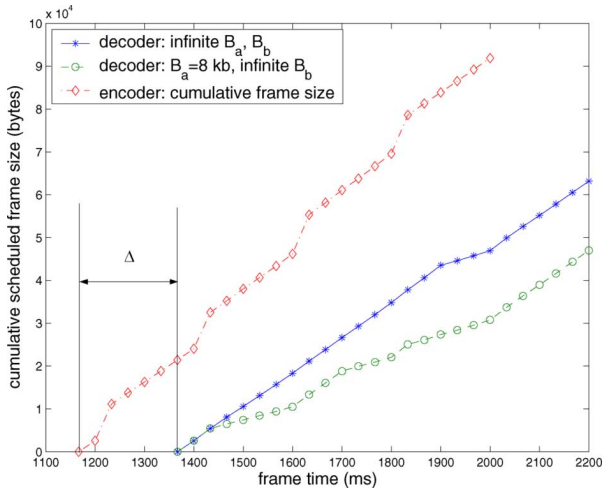

Fig. 9. Encoded video frame rate (cumulative) and decoded video frame rates (cumulative) in the case of infinite and constrained intermediate buffers.

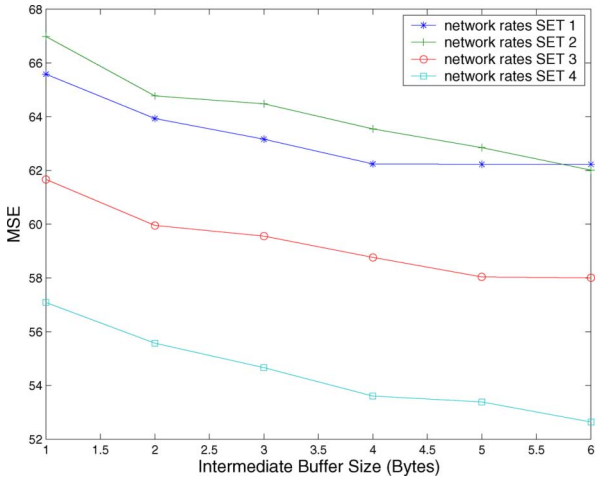

Fig. 8. Mse values for different network rate sets as a function of intermediate buffer size.

Results are improving once the sliding window increases, since the LBA algorithm has more flexibility in scheduling the video packets. Finally, given a reasonable sized window ($\delta = 0.5$ s), the results of the LBA are comparable to the case where the entire sequence is known before scheduling.

We now further investigate the effect of the size of intermediate buffers on scheduling performance. For the same network rate sets as before, we vary the size of intermediate node buffers (B_a and B_b). We observe that, for the same network rates, bigger intermediate buffers allow for the scheduling of more video packets, with improved smoothing of the rate variations; the difference being noticeable in terms of MSE (Fig. 8).

Finally, we study the effect of the intermediate buffer size on the packet load balancing on the two network paths. We compare the scheduling process on the two network paths in the case where the intermediate nodes have infinite or limited buffer space. Fig. 9 presents the cumulative encoded frame rate of the total bitstream and the successfully scheduled bitstream rate in the case of infinite intermediate buffers, compared to the case when the buffer of node a is limited to 8 kB. We can observe the difference in the total scheduled rate for the two scenarios.

Similarly, we observe major differences in the packet scheduling on the two paths between the two scenarios. A small buffer size on the first network path will render it unusable for a considerable period of time. This shortage is partially compensated by sending the base layer packets on the second link during the specific period. However, the effects on the total received bitstream are noticeable. The scheduling of the bitstream in the case of unlimited intermediate buffers is therefore smoother.

D. Streaming With Link Rate Estimation and Channel Losses

Next, we release the assumption of a perfect channel knowledge, and we test our proposed scheduling algorithm in the case where the server estimates the channel availability, and the transmission process suffers losses on the network links. We program our simulation scenario in ns-2 [22], where we simulate ten background flows for each link. These flows are generated according to the On/Off Exponential distribution, with average rates between 100 and 300 kbps. The available instantaneous rate for our streaming application is considered to be the difference between the total link bandwidth and the aggregated instantaneous rate of the background traffic. While the exact shape of the background traffic is not important for our work, the On/Off exponential distribution of background traffic leaves a constant average available rate for our application, with instantaneous rate variations that can be larger than 100% (please refer to [23] for other types of traffic). At the same time, we generate packet losses on each of the network paths, according to an i.i.d. process with probabilities equivalent to packet loss rates between 1% and 3%.

Next, the server implements a simple bandwidth estimation algorithm, based on an auto-regressive model. It estimates the bandwidth for each time window of size T_p, as follows. The available rate r_{k+1} of a segment in the next time interval $k+1$ is given by $r_{k+1} = \gamma(\sum_{j=1}^{k-1} r_j/(k-1)) + (1-\gamma)r_k$, where γ is the prediction coefficient. While the instantaneous rate variations of the channel can happen on very small time scales (of

tens to hundreds of milliseconds), the fastest estimation mechanisms [17] provide accurate results on time intervals of the size of a few round-trip times (e.g., at least 1 s or more). In the simulations, we set $T_p = 1\,\mathrm{s}$. Note finally that exact rate prediction is not crucial for the proposed algorithms, even if accurate prediction can only improve the performance.

We test the LBA protocol in the case when the server uses three disjoint paths for transmission, and the video is scalably encoded into one BL and two ELs. We use a GOP of 31 frames, with 15 P frames between I frames, and one B frame between P frames. The two enhancement layers are created by splitting the FGS enhancement layer created by the MPEG-4 FGS encoder. We split the bitplanes such that the two layers have similar average rate, similar to [24]. We set the rates to 300 kbps for the BL, and 260 kbps, respectively, for the two ELs. The packet weights are set in a similar manner as in the previous experiments.

We schedule the first 100 frames of *foreman_cif*, and we compare the results obtained by our algorithm and the EDPF algorithm [15], [16], in the case where the server knows the rates in advance and there is no channel loss, with the case when it predicts the rates based on the auto-regressive model presented above, and the transmission process suffers from path losses. We set the average rates on the three network paths to 280, 200, and 170 kbps, and the packet loss probabilities to 1%, 3%, and 2% respectively.

The maximum playback imposed by the client is $D = 200\ \mathrm{ms}$. For the computation of the scheduling policy based on predicted rates, we however use a more conservative delay of $D_1 = 150\ \mathrm{ms}$, in order to cope with big shifts in link rates and avoid the drop of important packets [25]. The scheduling results are presented in Figs. 10 and 11. We observe that in the case of LBA, the performance degradation compared with the optimal case, when all rates are known, is negligible. While, in the optimal case, the algorithm correctly schedules 201 packets, out of 300, representing 67% of the total stream, in the case of prediction, it manages to schedule 186 packets, representing 62%. While no frame is lost due to frame dropping or late packet arrivals at the client, we observe a limited number of lost frames due to the loss of BL packets on the transmission process. Simple rate prediction, combined with conservative playback delay settings, offers performance in terms of client video quality that is comparable to the case where rates are perfectly known at the server.

We observe that EDPF tends to schedule entire frames and drop less important frames in favor of more important ones. On the other hand, the LBA algorithm prefers to schedule the most important video layers first, and only then schedules packets belonging to the enhancement layers, if the network bandwidth permits it. Due to the fact that LBA can handle scalable video streams, we also observe that it is more robust to channel loss than EDPF. LBA loses an entire frame only if a BL packet is lost due to channel errors.

In the context of simple error-concealment methods at the client (e.g., frame replacement), the LBA scheduling provides a smoother quality of the received video (7.2 MSE points compared to 22.4 MSE points in the case of EDPF). At the same time, due to the variable size of the frames, EDPF is more vulnerable to network rate variations and prediction errors than LBA.

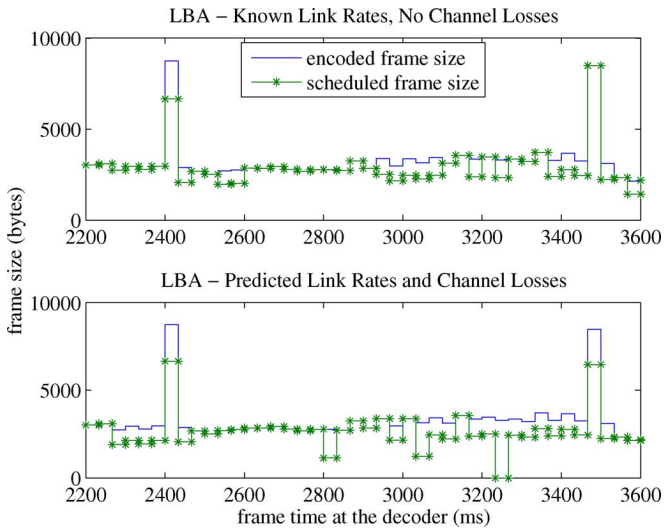

Fig. 10. LBA performance on three network paths with predicted parameters and channel losses.

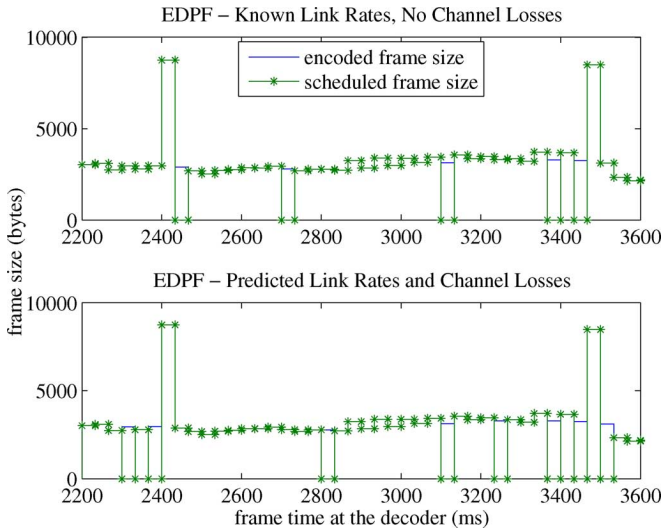

Fig. 11. EDPF performance on three network paths with predicted parameters and channel losses.

Note that packet loss can be mitigated by use of error resilient mechanisms (e.g., FEC or packet retransmissions [26]), and we only present results in lossy scenarios to evaluate the performance of the schemes in limit conditions. The design of a scheduling strategy adapted to lossy environments is, however, outside the scope of the present work.

E. Complexity Considerations

Finally, we analyze the complexity of the proposed algorithms and we try to derive a good trade-off for our LBA method, between complexity and performance, as a function of the size of the sliding window. While the B&B algorithm has a prohibitive exponential complexity as a function of the size of the sliding window, the EDPF and the Round Robin algorithms are very simple, their complexity being linear in terms of the number of total scheduled frames, and independent

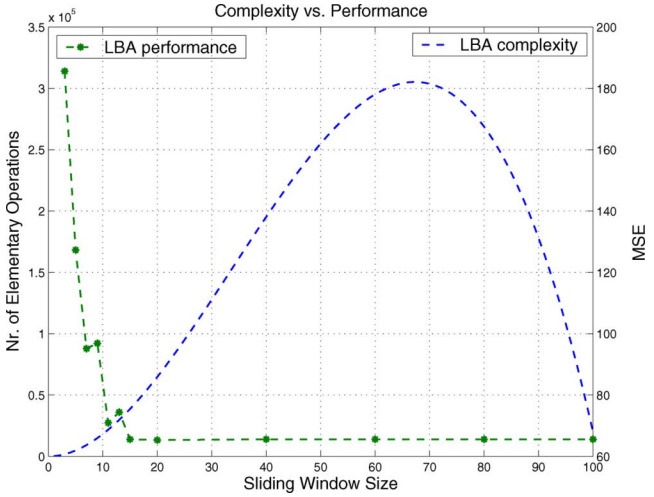

Fig. 12. LBA performance versus complexity (100 frames, average aggregated bandwidth of 450 kbps).

of the size of the sliding window. The complexity of our algorithm lies between the two bounds. It takes more operations than the simple EDPF scheduling, but it is still polynomial in complexity and can be performed in real time. At the same time, it is similar in complexity to the EDPF algorithm with the selective frame-discard enhancement [15].

Fig. 12 presents the performance of the LBA algorithm for different sizes of the sliding window. We superimpose the complexity curve with the performance curve in order to find the operational sliding window size as a function of the two values. We observe that for low values of the sliding window size, the performance of the LBA algorithm matches the one of the scenario when all frames are known in advance. At the same time, the complexity of the algorithm remains low. Low complexity and good performance, even for small sliding window sizes that allow to maintain low end-to-end delays, make the LBA a suitable candidate for real-time packet scheduling in multimedia streaming.

VII. Conclusions

This work addresses the problem of the joint selection and scheduling of video packets on a network topology that offers multiple paths between the streaming server and the media client. We use an encoded video abstraction model that factors in the variable importance of video packets, as well as their interdependencies. A formal analysis of packet transmission timing leads to the derivation of efficient algorithms to find the transmission policy that maximizes the video quality at the client. We propose fast, polynomial time algorithms that still offer close to optimal solutions, in the case of stored videos, and real-time streaming. Simulation results in both scenarios prove that our proposed heuristic-based solution performs well in terms of final video quality, and is moreover suitable for the case of real-time streaming under strict delay constraints. Due to its comprehensive modelling of video streams and its resiliency to imprecise bandwidth estimation, the proposed low complexity solution provides an interesting solution for video streaming in multipath infrastructures.

Acknowledgment

The authors thank Dr C. De Vleeschouwer for the helpful discussions and his comments on this work.

References

[1] T. Nguyen and A. Zakhor, "Distributed video streaming with forward error correction," in *Proc. Packet Video Workshop*, Pittsburgh, PA, Apr. 2002.
[2] M. R. Garey and D. S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, V. Klee, Ed., 23rd ed. New York: W. H. Freeman, 2002.
[3] D. Jurca and P. Frossard, "Distortion optimized multipath video streaming," in *Proc. Packet Video Workshop*, Irvine, CA, Dec. 2004.
[4] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, 2nd ed. Berlin, Germany: Springer-Verlag, 2000.
[5] L. Golubchik, J. Lui, T. Tung, A. Chow, and W. Lee, "Multi-path continuous media streaming: What are the benefits?," *ACM J. Perform. Eval.*, vol. 49, no. 1-4, pp. 429–449, Sep. 2002.
[6] J. G. Apostolopoulos and M. D. Trott, "Path diversity for enhanced media streaming," *IEEE Commun. Mag.*, vol. 42, no. 8, pp. 80–87, Aug. 2004.
[7] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee, "On multiple description streaming with content delivery networks," in *Proc. IEEE INFOCOM*, Jun. 23–27, 2002, vol. 3, pp. 1736–1745.
[8] T. Nguyen, P. Mehra, and A. Zakhor, "Path diversity and bandwidth allocation for multimedia streaming," in *Proc. IEEE Int. Conf. Multimedia and Expo (ICME)*, Jul. 6–9, 2003, vol. 1, pp. 1–4.
[9] H. Man and Y. Li, "A multipath video delivery scheme over diffserv wireless lans," in *Proc. SPIE-IS&T Electron. Imag.*, 2004, pp. 1148–1158.
[10] D. Tian, X. Li, G. Al-Regib, Y. Altunbasak, and J. Jackson, "Optimal packet scheduling for wireless streaming with error-prone feedback," in *Proc. IEEE WCNC*, Mar. 21–25, 2004, vol. 2, pp. 1287–1292.
[11] Z.-L. Zhang, S. Nelakuditi, R. Aggrawal, and R. P. Tsang, "Efficient selective frame discard algorithms for stored video delivery across resource constrained networks," *Real Time Imag.*, vol. 7, pp. 255–273, 2001.
[12] J. Huang, C. Krasic, and J. Walpole, "Adaptive live video streaming by priority drop," in *Proc. Packet Video Workshop*, Nantes, France, Apr. 2003.
[13] M. Roder, J. Cardinal, and R. Hamzaoui, "Branch and bound algorithms for rate-distortion optimized media streaming," *IEEE Trans. Multimedia*, vol. 8, no. 1, pp. 170–178, Feb. 2006.
[14] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *IEEE Trans. Multimedia*, vol. 8, no. 2, pp. 390–404, Apr. 2006.
[15] K. Chebrolu and R. Rao, "Bandwidth aggregation for real-time applications in heterogeneous wireless networks," *IEEE Trans. Mobile Comput.*, vol. 5, no. 4, pp. 388–403, Apr. 2006.
[16] K. Chebrolu and R. R. Rao, "Selective frame discard for interactive video," in *Proc. IEEE ICC*, 2004, pp. 4097–4102.
[17] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "PathChirp: Efficient available bandwidth estimation for network paths," in *Proc. Passive and Active Measurement Workshop*, La Jolla, CA, Apr. 2003.
[18] F. Kelly, "Notes on effective bandwidths," in *Stochastic Networks: Theory and Applications*, F. P. Kelly, S. Zachary, and I. Zeidins, Eds. Oxford, U.K.: Oxford Univ. Press, 1996, pp. 141–168.
[19] J. Jonsson and K. G. Shin, "A parametrized branch and bound strategy for scheduling precedence-constrained tasks on a multiprocessor system," in *Proc. IEEE ICPP*, Aug. 11–25, 1997, pp. 158–165.
[20] F. A. Samadzadeh and G. E. Hedrick, "Near-optimal multiprocessor scheduling," in *Proc. ACM Computer Science Conf.*, Apr. 1992, pp. 477–484.
[21] H. M. Radha, M. van der Schaar, and Y. Chen, "The MPEG-4 fine-grained scalable video coding method for multimedia streaming over ip," *IEEE Trans. Multimedia*, vol. 3, no. 1, pp. 53–68, Mar. 2001.
[22] [Online]. Available: http://www.isi.edu/nsnam/ns/
[23] D. Jurca and P. Frossard, Packet Selection and Scheduling for Multipath Video Streaming EPFL, 2004, Tech. Rep. TR-ITS-2004.025.
[24] P. de Cuetos, M. Reisslein, and K. W. Ross, Evaluating the Streaming of FGS-Encoded Video with Rate-Distortion Traces Institut Eurecom, Sophia Antipolis, France, 2003, Tech. Rep. RR-03-078.
[25] D. Jurca and P. Frossard, "Media streaming with conservative delay on variable rate channels," in *Proc. IEEE ICME*, Toronto, Canada, Jul. 2006.
[26] D. Tian, Y.-C. Lee, G. AlRegib, and Y. Altunbasak, "Packetized media streaming over multiple wireless channels," in *Proc. IEEE Int. Conf. Communications (ICC)*, Paris, France, 2004.

Dan Jurca received the B.Sc. degree from the "Politehnica" University of Timsoara, Timsoara, Romania, in 2002. In 2003, he graduated from the School of Computer and Communication Sciences, Ecole Politechnique Federale de Lausanne (EPFL), Lausanne, Switzerland, where he is currently pursuing the Ph.D. degree.

His research interests focus on adaptive rich media streaming (forward error correction, multipath media transmissions, and streaming in wireless environments).

Pascal Frossard (S'96–M'01–SM'04) received the M.S. and Ph.D. degrees, both in electrical engineering, from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 1997 and 2000, respectively.

From 1998 to 2000, he was with the Signal Processing Laboratory, EPFL, as a Research and Teaching Assistant under a grant from Hewlett-Packard. Between 2001 and 2003, he was a member of the Research Staff at the IBM T. J. Watson Research Center, Yorktown Heights, NY, where he worked on media compression and streaming technologies. Since 2003, he has been an Assistant Professor at EPFL, supported by the Swiss National Science Foundation. His research interests include image representation and coding, nonlinear representations, visual information analysis, joint source and channel coding, multimedia communications, and multimedia content distribution.

Dr. Frossard has been the General Chair of IEEE ICME 2002 (Lausanne, Switzerland) and a member of the organizing or technical program committees of numerous conferences. He has served as Guest Editor of special issues on Streaming Media (IEEE TRANSACTIONS ON MULTIMEDIA), on Media and Communication Applications on General Purpose Processors: Hardware and Software Issues (*Journal of VLSI SPSS*), and on Image and Video Coding Beyond Standards (*Journal of Signal Processing*). He is an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA (2004–present) and of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY (2006–present). He served as a member of the Editorial Board of the *EURASIP Journal of Signal Processing* (2003–2005) and, since 2004, has served as Vice Chair of the IEEE Multimedia Communications Technical Committee, as a member of the IEEE Multimedia Signal Processing Technical Committee, and of the IEEE Multimedia Systems and Applications Technical Committee. He received the Swiss National Science Foundation Professorship Award in 2003 and the IBM Faculty Award in 2005.