A Bandwidth Dependent Window-Based Smoothing Algorithm for Wireless Video Streaming in UMTS Networks

Pietro Camarda*, Domenico Striccoli† and Roberta Laraspata

* Politecnico di Bari - DEE, Via Orabona 4, 70125, Bari – ITALY
† Politecnico di Bari - DIASS, Viale del Turismo 8, 74100, Taranto – ITALY

Abstract — Variable Bit Rate (VBR) video transmission over UMTS networks is assuming an ever growing importance. To face the problem of VBR data transmission over wireless channels, work-ahead smoothing techniques can be fruitfully adopted. In this work, a sliding window smoothing algorithm, particularly suitable for real-time interactive multimedia transmission of VBR streams is proposed. It smoothes video streams over partially overlapped time windows, taking into account the fluctuating available bandwidth profile, typical of the wireless link, and the feedback on the real available buffer size periodically coming from client terminals, with the aim to minimize losses. Numerical results demonstrate the effectiveness of the proposed algorithm if compared with the sliding version of Minimum Variability Bandwidth Allocation (MVBA) algorithm, already known by literature, in different contexts.

I. INTRODUCTION

The technological revolution and the significant progress in wireless telecommunication networks have brought a development of innovative services, whose principal peculiarities are their flexibility and capacity to satisfy the mobility requirements peculiar to the ever growing number of wireless terminal users.

The Universal Mobile Communication System (UMTS) standard [1] has been explicitly developed for providing various types of services with relatively high bandwidth requirements, exploiting the packetized data transmission, typical of Internet. Specifically, the UMTS system provides a large variety of information services, with different Quality of Service (QoS) guarantees, both in real-time and non real-time contexts, and with relatively high bit rates. The principal UMTS aim is to provide to mobile users the same wide range of applications actually used in wired networks.

One of the most interesting services implemented in the UMTS systems in the last years is the multimedia transmission of contents (pictures, video, video conference, TV programs, etc.) with high QoS requirements, that is, high bit rates and reduced delays and jitters [2].

The wireless multimedia streaming is actually an important challenge. The main factors that influence the correct delivery of audio and video contents over terminals are the highly fluctuating conditions of wireless links and the limited amount of buffering on mobile terminals. UMTS systems should guarantee continuous and lossless data delivery despite of the highly variable bandwidth conditions of wireless channel and the relatively high data bit rates.

In Fig. 1 the wireless streaming architecture is illustrated. The streaming server sends packetized multimedia data to the mobile user through the IP network, composed by the public network (Internet), the mobile operator wired network and the wireless link.

![Fig. 1. The wireless streaming architecture.](image1)

In this way, the streaming server knows the number of packets arrived to the client, stored in the buffer or decoded for playout. These information represent an important feedback on the dynamic evolution of the video transmission, that can be exploited to improve the quality of the provided service at server side. In fact, the streaming server could regularize the bit rate of transmitted data according to the client characteristics like the buffer size and the type of compressed data.

1-4244-0981-0/07/$25.00 ©2007 IEEE.
The wireless link illustrated in Fig. 1, that delivers data to the wireless network udders, is considered as the bottleneck of the entire transmission system. In fact, the wireless channel is often subject to bandwidth conditions highly fluctuating in time and frequency. To give a more precise idea of this, in Fig. 2 a wireless channel bandwidth fluctuation, represented as a function of time and frequency, is illustrated.

The problem of bandwidth fluctuation is particularly emphasized if transmitted data present a bit rate highly variable in time. This is often the case of Variable Bit Rate (VBR) video streams, compressed through MPEG coding algorithms [5].

It is known that a MPEG compressed video is characterized by a high variability of its bit rate, both in the short and long time period [6]. The variable nature of compressed stream bit rate is essentially due to the different type of coded video frames, the coding technique and the quality degree of compressed streams [5]. As an example, in Fig. 3 a piece of the “Silence of the Lambs” film, of length 90,000 video frames, MPEG-4 coded, is represented.

As explained in [7], the wireless channel bandwidth fluctuations, together with the VBR stream nature, could bring to a higher error probability at transmission side because of the wireless channel interferences. They are essentially due to the signal attenuation from transmitter to receiver, but also to signal multiple paths and/or total interruptions (the “shadowing-fading” phenomenon) due to physical obstacles. A Constant Bit Rate (CBR) stream transmission or a bandwidth assignment equal to the stream peak rate would not solve the problem. In fact, the first solution would unavoidably bring to a consistent video quality worsening, since a constant bit rate is imposed to the source to transmit a VBR stream. The second solution would bring to a consistent waste of bandwidth resources in the almost totality of the frame times, where the stream bit rate is consistently lower than its peak, as can be noted from Fig. 3.

A useful solution could be regularize the stream bit rate at transmission side to avoid channel congestions, reducing the high bit rate variability of the VBR streams and at the same time guaranteeing continuous and lossless playback at receiving side without a stream quality degradation [8]. The work-ahead smoothing algorithms perform this operation [9][10]. Lots of smoothing techniques have already been presented in literature. Their common principle is the strong reduction of the bursty bit rate behaviour and the peak rate of VBR streams. Starting from the original unsmoothed video data, the smoothing algorithms generate a transmission plan that transmits, whenever possible, video data ahead of playback time. The resulting schedule will be piecewise Constant Bit Rate (CBR). The entity of each CBR segment mainly depends on the original unsmoothed data to be scheduled for transmission and the client buffer size; generally, the larger the buffer size, the smaller the CBR values and the longer their duration. At receiving side, the buffer in the UMTS terminal stores the smoothed data, and the original unsmoothed video stream laves the buffer for decoding and playing. The scheduled transmission profile is built in such a way to avoid receiving buffer overflows and underflows along the entire stream duration, to guarantee a continuous playback at the client side without frame losses and video quality degradation.

In this work, we present a novel smoothing algorithm, to be implemented in UMTS systems, that tries reduce all the problems addressed in this sections. As will be more clear in the sequel, it is thought for real-time interactive multimedia applications. It is a “sliding-window” smoothing, that is, its schedule is calculated over sliding time windows partially overlapped. Exploiting the feedback information on the residual buffer fill level carried to the streaming server by RTCP, the proposed algorithm reduces bit losses by dynamically modifying, in real time, the transmission plan accordingly, and taking also into account the limited available bandwidth in the wireless link.

II. SMOOTHING PRINCIPLES

As described in [10], a VBR video stream is composed by N video frames, each of them of size $d_i$ bytes $(1 \leq i \leq N)$. On the server side, the stream data are scheduled according to the particular smoothing algorithm. At the client side, the smoothed video data
enter the buffer and the original unsmoothed video frame sequence leaves it for decoding and playout. Let us now consider the client buffer model in the \( k \)th discrete time slot, assumed as the basic time unit. A discrete time slot, or frame time, is supposed to be the time interval in which a video frame is transmitted (1/25 s for PAL). To guarantee a feasible transmission, the cumulative input data to the receiving buffer at \( k \)th discrete time, \( S(k) \), should arrive quickly enough to avoid buffer underflow.

The buffer underflow and overflow curves are respectively:

\[
D(k) = \sum_{i=1}^{k} d_i ; B(k) = b + \sum_{i=1}^{k} d_i = D(k) + b.
\] (1)

So it has to be:

\[
D(k) \leq S(k) = \sum_{i=1}^{k} s_i \leq B(k)
\] (2)

where \( s(i) \) represents the smoothed stream bit rate in the \( i \)th discrete time slot. The smoothed stream transmission plan will result in a number of CBR segments, and the correspondent smoothed transmission plan is given by a monotonically increasing and piecewise linear path that lies between the \( D(k) \) and \( B(k) \) curves, as illustrated in Fig. 4.

![Fig. 4. An example of a smoothed stream transmission plan.](image)

Different types of smoothing algorithms have been implemented [11][12][13][14]. They are applied to stored video traffic, where all source video data are all known a priori and can be optimally scheduled “off-line”. Nevertheless, there is a growing number of VBR live interactive video applications (videoconferences, video news, interactive streaming, etc.) in UMTS networks that requires “on-line” smoothing algorithms, to reduce bit rate variability on-the-fly during stream transmission [15]. The main limitation of the on-line smoothing algorithms is that they have a reduced a priori knowledge of frame sizes in short consecutive temporal observation windows; the smoothed scheduling optimality is reduced as consequence. Nevertheless, online smoothing of VBR video streams is still effective to reduce peak rate and rate variability in the temporal window of interest, at the same time performing a on-the-fly computation of smoothing transmission plans. As described in [15], a further optimization of smoothed transmission plan can be obtained by applying the smoothing algorithm on time windows of \( N \) frames sliding by \( \alpha \) frames. The overlapping degree among consecutive smoothing windows will thus be of \( N - \alpha \) frames. Smoothing algorithm is thus applied on \( N \) frames, composed by \( N - \alpha \) smoothed frames calculated in the previous window, plus \( \alpha \) new unsmoothed frames. Results show that smoothing performance improve for higher overlapping degrees \( (N - \alpha) \).

Regarding the impact of available bandwidth resources on smoothing, several works can be found in literature. An available bandwidth dependent smoothing, the Network Constrained Smoothing (NCS) algorithm, is considered in [16]. It takes into account available bandwidth constraints and schedules the single video stream over a server-side transmission. This simple technique considers future network traffic knowledge to derive available bandwidth. The multimedia data are then divided into equal-sized intervals in which a smoothed CBR segment is scheduled.

Another example of bandwidth dependent smoothing algorithm can be found in [17]. In this work, network calculus is exploited to optimize the client buffer size, playback delay and look-ahead delay in such a way to generate a lossless video stream schedule respecting particular traffic envelopes, i.e., curves representing the maximum traffic that can be sent to the network. Smoothing developed in [17] can be applied in combination with other existing smoothing algorithms like the one illustrated in [9] to further minimize other metric, like number of bandwidth changes or rate variability.

A Rate Constrained Bandwidth Smoothing (RCBS) is presented in [18] for interactive video streams delivery. It minimizes the amount of buffering needed by smoothing when a maximum constant rate constraint is given, simply by prefetching video data only when the rate constraint is violated, and leaving the original unsmoothed data unchanged when they maintain under the bandwidth constraint. In this way, if compared wit the classical smoothing techniques previously illustrated (CBA, MCBA, MVBA, etc.), the buffering needed for smoothing is greatly reduced and client buffers can store much more data for VCR functionalities (stop, pause, rewind and examine operations).

In [19] a smoothing scheme is proposed, based on the sliding window smoothing algorithm proposed in [15]. It dynamically adapts the slide length to smooth bursty traffic, minimizing occupied bandwidth and computational cost.

Another approach is suggested in [20], where a Monotonic Decreasing Rate (MDR) scheduler is implemented. It allows only monotonic decreasing rate allocations, reducing bandwidth requirements and greatly simplifying the admission test computational complexity, necessary to establish if a new video stream can be admitted to a system with limited bandwidth resources.
large number of simulations test the algorithm performance.

The dynamic bandwidth allocation issue for RCBR smoothed streams is tackled in [21]. The purpose of this work is twofold. Firstly, a source traffic prediction method is adopted. It is able to predict with sufficient accuracy bandwidth level changes of smoothed video traffic. Secondly, bandwidth prediction is used to decide in advance both channel rate and duration.

RCBR algorithm is considered also in [22], where a network testbed is set up to analyze RCBR smoothing performance. RCBR scheme is chosen because it simplifies buffering and scheduling requirements in network switches for VBR streams. RCBR scheme is compared with traditional CBR schemes in this testbed, testifying significant improvements in terms traffic data loss.

In this work, a novel smoothing algorithm, particularly suitable for real-time video transmission over UMTS networks, is proposed and analyzed. It exploits the basic ideas of the sliding window on-line smoothing illustrated in [15], where video data are smoothed over temporal windows of length $N$ frames, sliding by $\alpha$ frames (with $\alpha \leq N$). For this reason, $\alpha$ is called “slide length”. The transmission schedule is computed every $\alpha$ frames, on a $N$ frame window composed by $N-\alpha$ frames which are part of the previous smoothing window, and $\alpha$ new unsmoothed frames. This algorithm is called SLiDing window-Buffer Dependent Smoothing Algorithm (SLD-BDSA). It is an “on-line” algorithm that acts in real time, since the several UMTS applications require a on-the-fly schedule computation, during stream running. The main novelty of this algorithm is that it takes into account the feedback information on the real buffer fill level, periodically coming from the client terminal. This kind of approach can be very useful for UMTS interactive applications, where the user can perform a variety of “external” actions that cannot be known at server side, but can modify the buffer fill level independently of the received smoothed data. This feedback information can be exploited to modify on-the-fly the stream schedule at transmission side, in such a way to avoid bit losses due to buffer overflows and underflows.

The proposed algorithm takes also into account available bandwidth information to further reduce bit losses. As known by [15] smaller $\alpha$ reduce the smoothed stream peak rate and could thus help to reduce bit losses due to a bandwidth drop. Moreover, lower $\alpha$ values mean a higher computational algorithm overhead. Nevertheless, this could be not sufficient to avoid losses. A modified schedule taking into account available bandwidth profile could help in this sense. In fact, SLD-BDSA reduces its smoothed bit rate whenever available bandwidth falls down. Let us note that the available bandwidth profile could be known in advance in the same temporal observation window where scheduled data will be transmitted through statistical predictive techniques of bandwidth estimation [23]. These techniques predict the bandwidth behaviour, especially for wireless noisy channels, that is the case of UMTS applications. Available bandwidth for the single video stream could also be deterministically assigned a priori. Nevertheless, the aim of this work is a smoothing algorithm that takes into account the available bandwidth information. The way this information is derived is not the purpose of this work. It is thus assumed the available bandwidth profile known a priori.

The rest of the paper is structured as follows. In Section III the SLD-BDSA is presented and analyzed, with particular reference to the off-line and on-line contexts. In Section IV the SLD-BDSA is compared with the existing sliding version of MVBA algorithm as presented in [15], that does not take into account the buffer feedback nor the available bandwidth information. Finally, in Section V some conclusion on the proposed algorithm are provided.

III. THE SLD-BDSA PRESENTATION

In this section, the SLD-BDSA algorithm is presented and illustrated. It is a online server side algorithm, where data are smoothed on overlapping windows of length $N$ video frames, sliding by $\alpha$ frames. Generally, $N$ and $\alpha$ are assigned and cannot change dynamically during stream running. The smoothing window size $N$ should be chosen of few hundreds of frame times (a frame time corresponds to 1/25 s), to preserve a relatively small delay between server and client. Let us note in fact that smoothing can be performed once $N$ unsmoothed data are available to server. The correspondent delay will thus be exactly of $N$ frame times. Regarding $\alpha$, as previously explained it should be chosen as a good compromise between a small smoothed peak rate and a reduced computational overhead. As pointed out in [15], $\alpha = N/2$ is generally a good choice.

SLD-BDSA smooths the VBR video stream taking into account both the feedback information on the buffer fill level and the available bandwidth profile. Both these parameters depend only on the “external” environment and not on the smoothed schedule. As specified in [4], the real free buffer level is periodically provided by RTCP packets, let us suppose each $\Delta_t$ seconds, to the streaming server as a multiple of 64 kB. Let us suppose that the available bandwidth does not influence the schedule. For each sliding window of $N$ video frames, SLD-BDSA generates a transmission plan that minimizes both the scheduled peak rate and rate variability, always respecting the $D(k)$ and $B(k)$ bounds as defined by (1) and (2). Nevertheless, since the free buffer information varies in time, the $B(k)$ curve is modified as follows:

$$B(k) = D(k) + b(k)$$  \hspace{1cm} (3)

where $b(k)$ is a function of discrete time $k$ that assumes a constant value each $\Delta_t$ seconds. Let us note in fact that the free buffer information holds for the entire periodicity interval, until it is refreshed by the new one.
coming from client. We also assume that the new buffer information as generated by the client is immediately available at server side, so that smoothing can be continuously performed without interruptions.

The algorithm acts as follows. Let us suppose to know the stream frame size $d_k$ and the buffer variation profile $b(k)$ in a generic time interval $[k_1, k_2]$ (in frame times). The maximum bit rate $c_{\text{max}}$ is calculated as the maximum bit rate without overflowing the client buffer. Similarly, $c_{\text{min}}$ is the minimum bit rate calculated without emptying the buffer, so that:

$$c_{\text{max}} = \min_{k_1 \leq k \leq k_2} \left\{ \frac{B(k) - (D(k_i) + q)}{k - k_1} \right\}$$  \hspace{1cm} (4a)

The latest time instant where $c_{\text{max}}$ is reached over $[k_1, k_2]$ is:

$$k_g = \max_{k_1 \leq k \leq k_2} \left\{ k \frac{B(k) - (D(k_i) + q)}{k - k_1} = c_{\text{max}} \right\}$$  \hspace{1cm} (4b)

Similarly:

$$c_{\text{min}} = \max_{k_1 \leq k \leq k_2} \left\{ \frac{D(k) - (D(k_i) + q)}{k - k_1} \right\}$$  \hspace{1cm} (4c)

and the latest time instant where $c_{\text{min}}$ is reached over $[k_1, k_2]$ is:

$$k_d = \max_{k_1 \leq k \leq k_2} \left\{ k \frac{D(k) - (D(k_i) + q)}{k - k_1} = c_{\text{min}} \right\}.$$  \hspace{1cm} (4d)

$q$ is the initial buffer level in $k_1$ (in bytes). The transmission plan is feasible only if $c_{\text{max}} \geq c_{\text{min}}$ in $[k_1, k_2]$. In this case, the scheduled bit rate is set as $s(k) = c_{\text{max}} \forall k_1 \leq k \leq k_2$.

Given a temporal window of N frames, for example $[n, (n + N - 1)]$, SLD-BDSA calculates $c_{\text{max}}$, $c_{\text{min}}$, $k_g$, and $k_d$ as defined in (4a)-(4d), in each discrete time $k$, starting from the first frame time ($k = n$). Besides:

$$\begin{align*}
\Delta c^{(s)}_{\text{max}} &= \frac{B(k) - (D(k_i) + q)}{k - k_1} \\
\Delta c^{(s)}_{\text{min}} &= \frac{D(k) - (D(k_i) + q)}{k - k_1}
\end{align*}$$  \hspace{1cm} (5)

where $\Delta c^{(s)}_{\text{max}}$ and $\Delta c^{(s)}_{\text{min}}$ are the maximum and minimum feasible bit rates in $k$, respectively.

If there is $\bar{k} < n + N - 1$ such that $\Delta c^{(s)}_{\text{max}} > c_{\text{max}}$, surely a buffer underflow occurs in $\bar{k}$; the scheduled bit rate will thus be $c_{\text{max}}$ in $[k_1, \bar{k} - 1]$, then setting $k_1 = \bar{k} - 1$.

Similarly, if $\Delta c^{(s)}_{\text{min}} > c_{\text{min}}$, a buffer overflow occurs. The scheduled bit rate will be $c_{\text{min}}$ in $[k_1, \bar{k} - 1]$, and $k_1 = \bar{k} - 1$.

This procedure originates the “smoothest” transmission plan among all the feasible transmission plans in $[n, (n + N - 1)]$. It means that it has the minimum rate variability and the minimum peak rate. The analytical proof of this can be found in [9].

The so built transmission plan takes into account the real buffer fill level information through (3), but SLD-BDSA is able to consider also available bandwidth fluctuations. To respect bandwidth limitations, SLD-BDSA implements a control over the scheduled bit rate $s(k)$, that must satisfy the following three conditions in the $k^{th}$ frame time $(k \in [n, (n + N - 1)])$:

$$S(k) = S(k-1) + s(k) \leq B(k)$$  \hspace{1cm} (6a)

$$S(k) = S(k-1) + s(k) \geq D(k)$$  \hspace{1cm} (6b)

$$s(k) \leq w(k)$$  \hspace{1cm} (6c)

with the initial condition $S(0) = 0$, exploited only for the first time window $(n = 1)$. $w(k)$ is the available bandwidth information at the discrete frame time $k$. The (6) mean that the SLD-BDSA schedule faces available bandwidth drops, at the same time trying to avoid buffer overflows and underflows.

The logic followed by SLD-BDSA is based on the following main steps. Let us suppose SLD-BDSA acting in the generic temporal window of $N$ frames. If no bandwidth constraints are present, SLD-BDSA behaves exactly like MVBA, optimizing the transmission plan in terms of peak rate and rate variability. If available bandwidth is lower than needed by MVBA schedule, SLD-BDSA reduces the bit rate, adapting it to the available bandwidth profile. In Fig. 5 an example of this situation is represented, in a temporal observation window of 3,000 video frames.

The total amount of scheduled bits exceeding the available bandwidth profile, clearly visible in Fig. 5 as the grey areas, is redistributed by SLD-BDSA along the entire transmission plan, in such a way to respect bandwidth bounds. So in correspondence of the grey areas the scheduled bit rate must decrease to respect available bandwidth profile and in the rest of the schedule it must increase to compensate the bit rate reduction due to bandwidth constraints.

The total amount of scheduled bits exceeding the available bandwidth profile, that would be lost because of bandwidth limitation, is firstly stored in a “lost bits” variable. Then, SLD-BDSA redistributes them by increasing the scheduled bit rate of $\Delta$ bits wherever
available bandwidth allows it, that is, for each k respecting the condition: \( s(k) + \Delta \leq w(k) \).

This way of operation guarantees (6c), but not (6a) and (6b). SLD-BDSA must thus prevent also buffer underflows and overflows. For this reason, SLD-BDSA acts as follows. It first searches frame times in which buffer overflows or underflows occur. Let us note that, after the bit redistribution procedure due to the bandwidth limitation, the first critical condition found by SLD-BDSA, if any, is surely a buffer underflow. In fact, the MVBA schedule does never present buffer overflows; thus the bit rate decrease due to bandwidth limitations can only bring to buffer underflows. If a buffer underflow condition is found, the scheduled bit rate is increased by adding \( n + \alpha \) bits in all the time period in which \( S(k) \) does not respect the (6b), but always verifying the (6c). This amount of bits is taken from the “lost bits” variable; if they are not sufficient to totally avoid the buffer underflow, the “lost bits” variable will become null and losses will still occur. Let us note that this procedure of increasing the bit rate to avoid a buffer underflow, can also bring to a buffer overflow in other time intervals of the schedule.

If a buffer overflow occurs, the scheduled bit rate is decreased by reducing the bit rate of \( \Delta_{vw} \) in a time interval that includes the entire overflow time period. Since this operation is a bit rate decrease, again the “lost bits” variable increases and its content can be redistributed to compensate bit losses due to buffer underflows.

This suggests that this procedure can be iteratively repeated since the “lost bits” variable reaches its minimum, or becomes null. Bit losses will be given by the sum of the “lost bits” variable and the losses occurred for buffer underflows in \( n, (n + N - 1) \), after the iterative “lost_bits” redistribution previously explained.

The smoothing window is then temporally shifted by \( \alpha \) frames. This means that SLD-BDSA will act now in \( [n + \alpha, (n + N - 1 + \alpha)] \), with \( N - \alpha \) frames already smoothed in the previous step, plus \( \alpha \) new unsmoothed frames; the first smoothed \( \alpha \) frames of the previous window are instead ready for transmission. SLD-BDSA will exploit the same procedures previously exposed (formulas from (3) to (6)) in this new time window. All the information on the residual buffer fill level \( b(n + \alpha) \), the cumulative schedule \( S(n + \alpha) \) and the “lost_bits” variable, as presented at the end of the previous time window are stored for optimization of \( S(k) \) in the new time window. In fact, \( b(n + \alpha) \) and \( S(n + \alpha) \) are necessary to built \( S(k) \) in the new time window, while the “lost_bits” variable is utilized to further reduce losses in the new time window. Let us remember in fact that this variable represents bits that cannot be redistributed in the previous time window because of bandwidth and/or buffer bounds. Nevertheless, this bits could be effectively redistributed by SLD-BDSA along the time interval of the new \( \alpha \) unsmoothed frames, minimizing losses for bandwidth drops or limited available buffer capacity. Furthermore, the new \( \alpha \) unsmoothed frames falling into the new smoothing window could help to reduce losses for buffer underflows eventually occurring in the first smoothed \( N - \alpha \) frames of the same window. These considerations suggest us that higher values of the slide length \( \alpha \) increase the possibility of a “lost_bits” redistribution and reduce also the SLD-BDSA computational complexity. Nevertheless, higher \( \alpha \) values make the smoothing worse as regards the smoothed stream peak rate and rate variability [15]. And vice versa.

In each case SLD-BDSA minimizes the bits that would be lost because of bandwidth drops and buffer underflows and overflows. They are in fact utilized to increase the lower bit rates that cause buffer underflows through an iterative procedure.

The logic previously explained until a new buffer information is available to the server; then, the SLD-BDSA is applied again. SLD-BDSA modifies the schedule on-the-fly accordingly, during the stream running, and applying the sliding window smoothing in the \( \Delta_{bs} \), in which the last available buffer information holds. This is performed as follows. At the beginning of the video stream, SLD-BDSA calculates the stream schedule on temporal windows of length \( N \) (video frames) sliding length of \( \alpha \) frames, and considering the receiving buffer as empty. When the first buffer information \( a_{rew}(\Delta_{bs}) \) arrives to the server at frame time \( \Delta_{bs} \), it is exploited to calculate the new upper bound \( B(k) \) through (1). The SLD-BDSA schedule is calculated through (2) starting from \( \Delta_{bs} \), always sliding the N frame windows by \( \alpha \) frames, until \( 2\Delta_{bs} \). And so on. Fig. 6 gives an idea of the SLD-BDSA implementation. The algorithm is formally presented through its pseudo-code.
In this Section, we test SLD-BDSA effectiveness by comparing it with the sliding window version of MVBA smoothing algorithm already known by literature [15], that we call for simplicity SLD-MVBA (SLD-MVBA). Simulations are performed taking into account available bandwidth and buffer size variations, together with the real buffer size information, for different kind of VBR video streams. We choose MVBA as meter of comparison, because is the most effective in reducing the scheduled peak rate and the rate variability of a VBR video stream, both in off-line and on-line contexts.

To better explain the main advantages of SLD-BDSA if compared with the SLD-MVBA, let us observe Fig. 7.

Fig. 6. Pseudo-code for the SLD-BDSA.

Lines from 1 to 4 define the variables at the stream beginning, like the initial buffer level and the time interval \( \Delta t \) of buffer information. In each N frame time window, whose length is defined in line 6, the upper and lower bounds are calculated taking into account the new buffer information coming to the server from client each \( \Delta t \) frame times, and stored by the server (lines 11 and 12) to calculate the new upper bound accordingly; this operation is performed in line 14. Calling formally “BDSA” the SLD-BDSA algorithm applied in the single N-frame time window, the relative schedule is calculated in line 16. As previously explained, it depends on unsmoothed video data and available bandwidth, set in line 7, and obviously on the upper and lower bounds calculated in lines 9 and 14 taking into account the real buffer fill level. The time window is then shifted by \( \alpha \) frames, repeating the BDSA in the following N frame window.

Let us point out that SLD-BDSA must save the “lost bit” value, that is indispensible to perform smoothing in the following time window. In fact, the “lost bit” variable could be effectively reduced in the next N-frame time window, where more bits could be allocated in the time interval of the new \( \alpha \) unsmoothed frames, reducing losses for bandwidth limitation and/or buffer overflow. Finally, the iterative procedure is repeated until the stream end, as testified by the cycle set in lines 5 and 19.

IV. SLD-BDSA PERFORMANCE

In this Section, we test SLD-BDSA effectiveness by comparing it with the sliding window version of MVBA smoothing algorithm already known by literature [15], that we call for simplicity SLD-MVBA (SLD-MVBA). Simulations are performed taking into account available bandwidth and buffer size variations, for different kind of VBR video streams. We choose MVBA as meter of comparison, because is the most effective in reducing the scheduled peak rate and the rate variability of a VBR video stream, both in off-line and on-line contexts.

To better explain the main advantages of SLD-BDSA if compared with the SLD-MVBA, let us observe Fig. 7.
confirm this, let us observe Fig. 7b. In this figure, the upper bound sometimes presents abrupt changes, due to the new available buffer information coming from the client that brings to a recalculation of the buffer overflow profile. Analyzing Fig. 7b, it can be observed that SLD-MVBA schedule crosses the upper bound towards the 1500th frame time, thus provoking a buffer overflow at receiving side. SLD-BDSA schedule remains always confined between the upper and lower bounds; no losses occur. Thus, SLD-MVBA performs generally worse than SLD-BDSA, since the first does not take into account the feedback buffer information coming from the client. Let us note that losses have been evaluated in terms of lost bits, disregarding as first approximation the different importance of lost bits belonging to different frame types (I, P or B) in compressed streams. Even if this could be not so fair, it can be a good meter of comparison to compare SLD-BDSA and SLD-MVBA performance.

Now let us analyze SLD-BDSA and SLD-MVBA performance in presence of available bandwidth limitations. This situation can happen very often in the context of cellular UMTS networks, where data transmission occur in the open space and are subject to noise and interferences. Fig. 8 represents this comparison for a piece of the “Silence of the Lambs” video stream, MPEG-4 coded with high quality, and of total length 10,000 video frames. The available buffer information has been simulated as a profile randomly varying between 128 kB and 1 MB. To further stress the system, two bandwidth drops have been simulated. There is an initial bandwidth drop at the beginning of the stream, lasting for about 100 video frames. The second bandwidth drop is longer; it occurs between the 7,000th and 8,500th frame times. Both SLD-MVBA and SLD-BDSA have been performed over time windows of length \( N = 100 \) video frames (corresponding to 4 seconds), sliding by \( \alpha = 50 \) video frames (2 seconds).

In this case, the algorithm can smooth video frames in a limited time window of 100 video frames, that is exactly the duration of the bandwidth drop; for this reason, losses are unavoidable. The slide length of 2 s can only partially help to reduce losses, rescheduling frames in the next 2 s, where available bandwidth rises. Let us note also that SLD-BDSA losses occur because of buffer underflows at receiving side. A very little percentage of bit losses for SLD-BDSA is due to the abruptly changing upper bound

![Fig. 8. An example of SLD-BDSA and SLD-MVBA schedules with a bandwidth drop at the end of the schedule.](image)

As can be noted from Fig. 8, SLD-BDSA follows the bandwidth profile, and compensates the lower scheduled bit rate, if compared with SLD-MVBA, by increasing it immediately when available bandwidth rises again. This bit rate increase, that represents the difference between the two schedules where available bandwidth is high, cannot be appreciated over a large time interval, like the one represented in Fig. 8. Anyway, it is important to note that SLD-BDSA schedule is always under the available bandwidth profile, while SLD-MVBA crosses it in more than one time interval, generating bit losses at transmission side due to insufficient available bandwidth. Losses for SLD-BDSA are essentially due to buffer underflows, occurring because of the available bandwidth reduction. In fact, whenever a bandwidth drop occurs, SLD-BDSA is forced to schedule less bits than needed for lossless decoding at receiving side. Furthermore, the limited “visibility” of a single N frames time window implies the limited SLD-BDSA capability to increase scheduled bit rate in advance, to compensate this bit rate reduction for limited available bandwidth. And relatively low scheduled bit rates could be not sufficient at receiving side to decode the video stream without losses or playback interruptions. Anyway, we expect that SLD-BDSA bit losses are significantly less than the SLD-MVBA ones.

This assert can be confirmed by observing Fig. 9.

![Fig. 9. Lost bit distribution for SLD-BDSA and SLD-MVBA algorithms.](image)

In this figure, the bit losses distribution is depicted for the SLD-BDSA and SLD-MVBA algorithms, in the same simulation scenario illustrated in Fig. 8. Losses for SLD-BDSA are essentially concentrated in the first 100 frames of the scheduled transmission. This is due to the position of the first bandwidth drop. In fact, since the available bandwidth drastically lowers at the stream beginning, SLD-BDSA can not perform a work-ahead schedule to avoid in advance bit losses for bandwidth limitation, and it is forced to redistribute bits that would be lost only after the drop. Furthermore, let us remember that the SLD-BDSA is an on-line smoothing algorithm, performed over limited time windows of N video frames. In this case, the algorithm can smooth video frames in a time window of 100 video frames, that is exactly the duration of the bandwidth drop; for this reason, losses are unavoidable. The slide length of 2 s can only partially help to reduce losses, rescheduling frames in the next 2 s, where available bandwidth raises. Let us note also that SLD-BDSA losses occur because of buffer underflows at receiving side. A very little percentage of bit losses for SLD-BDSA is due to the abruptly changing upper bound
profile, because of a reduced available buffer size information coming from client (as can be noted, for example, in Fig. 7a after the 2,000th frame). In this case, if the cumulative amount of scheduled bits $S(k)$ exceed the upper bound in its changing point, losses will occur for some frame times because SLD-BDSA is unable to prevent the drastic upper bound reduction and change its schedule in advance accordingly.

SLD-MVBA losses are instead distributed along the entire stream duration. They are essentially due to three factors:

- **Losses for bandwidth drops.** This kind of losses occurs at transmission side, and it is due to the SLD-MVBA inability to take into account the available bandwidth profile during stream running;

- **Losses for buffer underflows.** This kind of losses occurs at receiving side because the bandwidth drop allows the transmission of less bits than scheduled by SLD-MVBA, that are not sufficient for a continuous and lossless frame decoding at receiving side. The cause of this loss is exactly the same of the SLD-BDSA algorithm previously explained;

- **Losses for buffer overflows.** This kind of losses, observable at receiving side, occurs because SLD-MVBA is not able to take into account the feedback buffer information periodically arriving to the server from client. SLD-MVBA schedule could sometimes exceed the upper bound whenever the real buffer size available at the receiver is reduced because of client actions. This is exactly the case illustrated in Fig. 7b.

The total losses, in percentage, experimented in the scenario simulated in Fig. 8 and Fig. 9 have been of 0,18% for SLD-BDSA and 0,41% for the SLD-MVBA. The difference of bit losses between the two algorithms obviously depend on the number, position and duration of bandwidth drops, and the available buffer size profile utilized for simulation.

In Fig. 10 the percentage of lost bits for different values of the slide length is reported for both SLD-BDSA and SLD-MVBA algorithms. The video stream utilized for this simulation is a piece of 10,000 video frames of the “Simpsons” cartoon, MPEG-4 coded with high quality. The percentage of lost bits has been calculated for time windows of 400 frames (16 s) and simulating a bandwidth drop lasting approximately 2,000 video frames, like the one represented in Fig. 8, but starting from the 2,000th video frame, that is, towards the beginning of video transmission.

As can be noted from Fig. 10, losses for SLD-BDSA are always smaller than the SLD-BDSA ones for all the experimented $\alpha$ values. This result confirms the SLD-BDSA effectiveness in reducing losses for bandwidth limitation and available buffer abrupt variations if compared with SLD-MVBA, whose bit losses are essentially due to its impossibility to adapt the scheduled bit rate to bandwidth drops and feedback buffer information.

As expected, losses increase with $\alpha$ for both SLD-BDSA and SLD-MVBA algorithms. In fact, from higher slide lengths derives a reduced superimposition among consecutive time windows, with consequent higher smoothed peak rates. Let us note that for $\alpha = N$ we obtain the “non-sliding” version of the two algorithms, obtained by smoothing data in consecutive, hopping windows. Fig. 10 shows that losses would be higher than the sliding window BDSA and MVBA, thus demonstrating the inefficacy of the hopping-window approach.

Fig. 11 shows the percentage of lost bits for both SD-BDSA and SLD-MVBA algorithms, evaluated for different values of the smoothing window size $N$.

The considered video stream is a portion (of length 10,000 video frames) of the “Star Wars” film, MPEG-4 coded with high quality. The smoothing window size has been increased by 100 video frames each time, starting from $N = 100$ frames (4 s) until $N = 500$ frames (20 s), and with a constant value of the sliding length ($\alpha = 50$). It derives that the sliding length, as percentage of the window size, decreases for increasing values of $N$. Also in this case, bit losses experimented
with SLD-BDSA are always smaller than the SLD-MVBA ones. In both cases, losses decrease for increasing values of $N$. This is obvious, since for greater $N$ the superimposition degree among consecutive smoothing windows increases. It derives that the new $\alpha$ unsmoothed video frames can be more effectively scheduled to reduce relative peak rates, reducing losses at transmission side for bandwidth limitation for SLD-MVBA, and at receiving side for buffer underflow for SLD-BDSA.

V. CONCLUSIONS

In this paper a new smoothing algorithm, SLD-BDSA, has been proposed, that regulates the transmitted bit rate of video streams transmitted over UMTS networks, taking into account available bandwidth constraints and the feedback information on available buffer periodically coming from the client. The SLD-BDSA exploits the sliding window smoothing advantages, typically utilized in on-line smoothing context, to reduce the scheduled bit rate and rate variability by recalculating the smoothed schedule on partially overlapping windows, at the expense of an increased computational overhead. SLD-BDSA tries to preserve the same advantages of the Sliding Window version of MVBA smoothing algorithm (conventionally called in this paper “SLD-MVBA”), very effective in reducing smoothed stream peak rate and rate variability in on-line video streaming contexts, but it represents a great innovation respect to the former, due to its capability to take into account the real available buffer space information and available bandwidth on the communication channel. Thanks to a simple and efficient bit redistribution along the smoothing window, SLD-BDSA is able to minimize frame losses. Its great flexibility of implementation in several situations makes it particularly suitable to be implemented in real-time video streaming contexts for UMTS networks, where critical available bandwidth conditions often occur.

REFERENCES


