Optimal Smoothing Schedules for Real-Time Streams*

EXTENDED ABSTRACT

Yishay Mansour — mansour@math.tau.ac.il
Boaz Patt-Shamir — boaz@eng.tau.ac.il
Ofer Lapid — ofer@eng.tau.ac.il

Dept. of Computer Science   Dept. of Electrical Engineering
Tel-Aviv University
Tel-Aviv 69978
Israel

Abstract

We consider the problem of *smoothing* real-time streams (such as video streams), where the goal is to reproduce a variable-bandwidth stream remotely, while minimizing bandwidth cost, space overhead, and playback delay. We focus on *lossy* schedules, where some bytes may be dropped due to limited bandwidth or space. We present the following results. First, we determine the optimal tradeoff between buffer space, queuing delay, and link bandwidth for lossy smoothing schedules . Specifically, this means that if one of these parameters is under our control, we can precisely calculate the optimal value which minimizes data loss while avoiding resource wastage. The tradeoff is accomplished by a simple generic algorithm, that allows one some freedom in choosing which data to discard. This algorithm is very easy to implement both at the server and at the client, and it enjoys the nice property that only the server decides which data to discard, and the client needs only to reconstruct the stream.

In a second set of results we study the case where different parts of the data have different importance, modeled by assigning a real "weight" to each byte in the stream. For this setting we use competitive analysis, i.e., we compare the weight delivered by on-line algorithms to the weight of an optimal off-line schedule using the same resources. We prove that a natural greedy algorithm is 4-competitive. We also prove a lower bound of 1.25 on the competitive ratio of *any* deterministic on-line algorithm. Finally, we give a few experimental results which show that smoothing is extremely effective in practice, and that the greedy algorithm performs very well in the weighted case.

*Research supported in part by Israel Ministry of Science.

1 Introduction

The essence of real-time communication is to reproduce a given data stream in a remote location; one of the main difficulties in doing it is that often, the bit rate of a real-time stream varies with time, while communication bandwidth is usually allocated in advanced, and hence it is basically fixed. This fundamental conflict between variable bandwidth requirement and constant bandwidth supply has many possible solutions:

- Degradation of service by "truncating" the stream to the link rate [6], or alternatively, reserving bandwidth for the peak rate, resulting in link under-utilization [12].
- Statistical multiplexing [11], relying on an assumed statistical independence of the bit rates of multiple streams;
- More complex traffic descriptors such VBR [17], allowing for a more refined specification of bursts;
- Renegotiation protocols [8] which facilitate dynamic bandwidth allocation; and
- *Smoothing,* which is the focus of this paper.

In smoothing, the basic idea is to trade bandwidth for space and latency. Specifically, the bytes of the input stream, instead of being submitted directly to the communication link, are first stored in a *server's buffer*; the server submits bytes stored in its buffer to the link when it deems appropriate, subject to the link rate constraint. Bytes arriving at the other side of the link are first stored in a *client's buffer,* which delivers them to the playout device after a reconstruction action. The added flexibility provided by the buffers is used to create a smoother traffic on the communication link, thus reducing the peak bandwidth requirement of the stream. Indeed, this technique is used on many levels, including, for example, MPEG encoders and decoders [2, 3]. The drawbacks of this method are the additional memory space required, and the added latency. In many practical cases, however, one can significantly reduce the peak bandwidth using only a relatively modest amount of space without unbearable delay.

The basic problem in smoothing is determining what is the link rate, buffer sizes, the playout delay, and, of course, what is the right algorithm to use. Much is known about the off-line case, where the entire stream is known ahead of time, thus allowing for preprocessing. The off-line scenario is well justified in some cases, e.g., stored video. In many other settings, such as live broadcasts, one would like to smooth an unknown stream in an on-line fashion.

Our results. In this paper we consider a basic variant of the smoothing problem, where the link rate is constant, and loss is allowed. Our first result is as follows. Denote the total queuing delay (i.e., end-to-end delay minus the link propagation delay) by D, the buffer size (of the client and the server) by B, and the link rate by R. We show that the minimal number of bytes is lost when the relation $B = R \cdot D$ holds. To understand the applicability of this result, consider the following scenario. Suppose that two of the three parameters (space, delay and link rate) are given. Then by applying our result one can trivially calculate the best choice for the third parameter. This calculation does not depend on any statistical assumptions: it holds deterministically. In a series of additional results, we show that setting the free parameter such that $B \neq RD$ may adversely affect system performance in the sense that we may have either unnecessary loss of data or plain resource wastage.

The algorithm which achieves the minimal data loss is *generic*, in the sense that when some bytes must be dropped, it is not important which bytes are discarded (provided they are available at the server). This intentional under-specification is very useful for practical encoding schemes, were the quality of the output does not degrade linearly with the number of bytes lost; in other words, the server is free to discard what seems to be the least important data. On the other hand, the client's algorithm does not involve such difficult decisions at all; the client's main burden is allocating the buffer space and reconstructing the stream.

We then consider the more general case, where different parts of the data have different importance. We model this case by assigning a positive real number called *weight* to each byte. The goal of a schedule in this generalized model is to maximize the *benefit*, defined to be the sum of the weights of bytes delivered on time. For this setting, we use competitive analysis, i.e., we compare on-line algorithms to optimal off-line schedules. Competitive analysis is attractive because it avoids any statistical assumptions, and gives a performance bound that holds for any input sequence (see [4] for an introduction to competitive analysis). Specifically, we define the *competitive ratio* of an on-line algorithm to be the least upper bound, over all input sequences, of the benefit obtained by an optimal (off-line) schedule, divided by the benefit obtained by the on-line algorithm. For this setting, we have the following results. We present a natural greedy on-line algorithm for smoothing and prove an upper bound of 4 and a lower bound of $2 - \frac{2}{B}$ on its competitive ratio. In other words, we show that for any input stream, the optimal off-line schedule delivers at most 4 times the weight delivered by our algorithm, and that in some cases, the greedy algorithm delivers only a little more than half of the weight deliverable by an optimal off-line algorithm. We conclude our competitive analysis with a general lower bound of 1.25 on the competitive ratio of deterministic on-line algorithm.

We close this paper with a set of experimental results that demonstrate the viability of smoothing: using MPEG clips found on the Internet, we show that modest buffer space can provide significant improvement in the number of dropped frames, and that the greedy algorithm usually have excellent performance in practice.

The rest of this paper is organized as follows. In Section 3 we describe the underlying model in detail. In Section 4 we present the generic algorithm and prove the basic connection between delay, buffer space, and link rate. In Section 5 we consider the model where each byte may have a different weight. Finally, in Section 6, we present some performance results for weighted and unweighted lossy smoothing.

2 Related Work on Smoothing

A considerable body of research was devoted to minimizing the number and cost of changes for lossless smoothing. Rexford and Towsley [14] give a thorough study of lossless smoothing over an internetwork. In this setting, the output stream need not be identical to the input stream. Salehi *et al.* [15] present an off-line lossless smoothing algorithm which is optimal in terms of rate variability. Rexford *et al.* [13] extend the optimal off-line algorithm of [15] to the on-line case by applying the off-line algorithm piecewise, using a sliding window technique. This idea is refined in [5] by dynamically changing the sliding window size as a function of the current burstiness of the input stream. Zhao *et al.* [20] consider the effect of the initial delay in lossless schedules, and give an efficient algorithm that finds the optimal initial delay, after which there is no reduction in the peak bandwidth. The significance of the the initial delay is also the motivation for the work of Sen *et al.* [16], where they investigate the use of a proxy server to cache a prefix of popular streams. This idea cannot be applied to on-line streams. Jiang and Kleinrock [10] give an optimal off-line algorithm for the problem of minimizing the total cost of changes of a lossless smoothing schedule. Their only assumption is that the cost of a change is local, in the sense that the overall cost is the sum of the individual change costs.

Another research direction is to study the inherent properties of VBR traffic, and video in particular. For example, Wrege *et al.* [18] present a deterministic model for VBR traffic, and study the earliest-deadline-first scheduling policy in this context. Feng and Rexford [7] present an experimental study of various smoothing algorithms. Duffield *et al.* [6] propose an algorithm for controlling the MPEG encoding so as to get smooth streams. Ni *at al.* [9] present a formula similar to ours for the connection between delay, buffer size, and link rate. However, their model is different: all the input stream is available in advance, so different bytes may have different queuing delays. In their model, they claim that the delay-bandwidth product is a lower bound on the buffer size. Zhang *et al.* [19] study the case where the buffer size, playout delay and the link rate are given, and the question is how to minimize the number of *frames* discarded, under the assumption that only whole frames can be discarded. They give an optimal algorithm, and a few heuristics which appear to be quite efficient for motion-JPEG streams.

3 Problem Statement

3.1 Informal Description

Intuitively, the system we consider consists of a source, a server buffer, a communication link, a client buffer, and a sink (see Figure 1). During each time step (we consider a slotted time model), the following events happen.

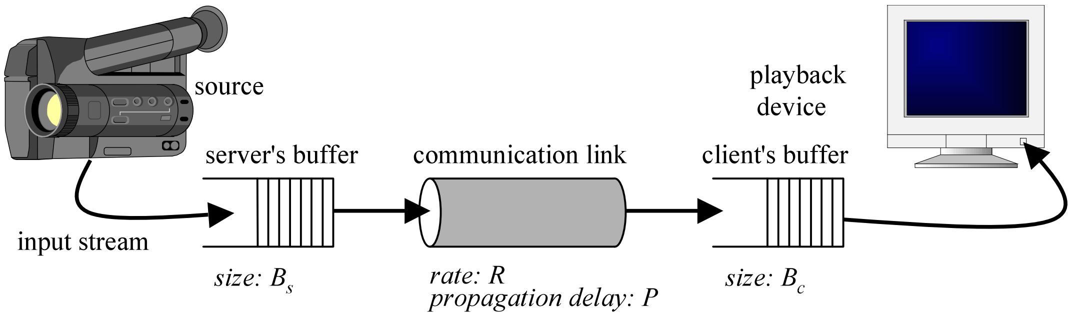

Figure 1: A schematic representation of a smoothing system. The link is lossless, and has constant propagation delay.

- A *source* generates a set of bytes called a *frame*, which is inserted to a *server's buffer*.
- The *server* submits some of the bytes currently stored in its buffer to a *link*.
- The *link* delivers some of the bytes it holds to the *client's buffer*. The link delay is assumed to be fixed.
- The *client* plays out some of the bytes stored in its buffer by delivering them to a *playout device*.
- In addition, any subset of the bytes currently stored at the server's buffer or the client's buffer may be *dropped* at the discretion of the algorithm. The link is assumed to be lossless.

Bytes may be dropped due to the following restrictions.

Server Overflow: The server's buffer has limited space, and the link has a limited rate which bounds the rate at which the server's buffer can be drained. If the number of bytes generated at a time is larger than the current size of free space in the buffer, some of bytes (perhaps stored at the server) must be discarded.

Client Overflow: Usually, there is a limit on the rate in which the client's buffer is drained (this is always the case in real-time systems—see below). If the number of bytes arriving at the client is larger than its current number of free slots, then some bytes must be dropped.

Early Drop: The algorithm may drop bytes arbitrarily, possibly to avoid dropping bytes later.

The goal of a real-time smoothing schedule is to reconstruct the input stream generated at the source as closely as possible at the playout device. To this end, we assume that bytes are somehow tagged by their arrival time (i.e., their frame number). For our analysis, we also assume that link propagation delay is constant.

The performance measures of interest for a smoothing system are the following.

- Playout Latency: How long does it take for a byte since it is generated at the source until it is played out.
- Buffer Requirement: How much buffer space do the server and client need.
- Link Rate: What is the maximal link rate used by the smoothing schedule.
- Fidelity: How close is the reconstructed stream to the original input stream.

Note that while playout latency, buffer space and link rate are well defined and easy to measure, fidelity is harder to quantify. In particular, if the schedule is *lossy*, (i.e., some of the bytes are dropped), the perceived fidelity seems subjective [19], and depends on the actual stream and encoding scheme employed. In full generality, one may have a cost function assigning a "fidelity index" for the pair of full input/output streams. A simpler alternative is to have a *local* cost function assigning a (potentially different) cost to each byte, and the global fidelity index is the sum (or the average) of the cost of lost bytes. A special case of a local cost function is assigning a unit cost to each byte, thus simply measuring how many bytes are lost by the system.

3.2 Formal Model and Notation

The following definition formalizes the notion of an input stream. Intuitively, it is defined by specifying for each byte its exact time of arrival in the system. For example, a frame of a video stream is just the set of all bytes whose arrival time is the time the frame is generated. Using bytes as the basic entity will allow us to break and re-assemble frames later.

Definition 3.1 *An* input stream *is a set* $\mathcal{B}$ *of* bytes *and an* arrival time function $\mathsf{AT} : \mathcal{B} \to \mathbb{N}$.

(We use $\mathbb{N}$ to denote the set of natural numbers including 0.) Next, we define the notion of a schedule using similar a formalism. Note that in the definition below, times may also be infinite due to the possibility of dropping bytes.

Definition 3.2 *A* smoothing schedule *for a given input stream is defined by the following additional functions, all mapping bytes to* $\mathbb{N} \cup \{\infty\}$.

- send time *function, denoted* ST.
- receive time *function, denoted* RT.
- departure time *function, denoted* DT.
- drop time *function, denoted* ZT.

For example, a byte b is generated by the source at time $\mathsf{AT}(b)$, waits in the server's buffer until time $\mathsf{ST}(b)$, when it enters the link. At time $\mathsf{RT}(b)$ byte b is received at the client's buffer, where it waits until it is played out at time $\mathsf{DT}(b)$. For this byte we have infinite drop time, i.e., $\mathsf{ZT}(b) = \infty$. Another byte

b' may be generated at the source at time $\mathsf{AT}(b')$, and dropped from the server's buffer at time $\mathsf{ZT}(b')$; for this byte we have $\mathsf{ST}(b') = \mathsf{RT}(b') = \mathsf{DT}(b') = \infty$. Naturally, the functions defined above must obey certain restrictions if they describe a schedule as we intend. We skip the formal details here.

The following additional derived notations are intuitive and useful.

Notation 3.3 *Let a smoothing schedule be given.*

- $\mathsf{A}(t)$ *is the set of bytes arriving at time t, also called the* frame generated at time t.
- $\mathsf{S}(t)$ *is the set of bytes transmitted at time t.*
- $\mathsf{R}(t)$ *is the set of bytes delivered at time t.*
- $\mathsf{D}(t)$ *is the set of bytes played out at time t, also called the* frame played at time t.
- $\mathsf{Z}(t)$ *is the set of bytes dropped at time t.*
- $\mathsf{B}_s(t)$ *is the set of bytes stored at the server at time t.*
- $\mathsf{B}_c(t)$ *is the set of bytes stored at the client at time t.*
- *The* sojourn time *for a byte b is* $\mathsf{DT}(b) - \mathsf{AT}(b)$.
- *The* link propagation delay *for a byte b with finite sojourn time is* $\mathsf{RT}(b) - \mathsf{ST}(b)$.
- *The* queuing delay *for a byte b with finite sojourn time is its sojourn time minus its propagation delay.*

Finally, we define the performance metrics for a given schedule. Note that our measures are defined in hindsight, given the full schedule.

Definition 3.4 *Let a smoothing schedule be given.*

- *The* server buffer requirement *is the least upper bound on* $|\mathsf{B}_s(t)|$.
- *Similarly, the* client buffer requirement *is the least upper on* $|\mathsf{B}_c(t)|$.
- *The* link rate requirement *is the least upper bound on* $|\mathsf{R}(t)|$.

Thus far, we have defined a model for transmitting a general input stream with buffers. Our last step is to specialize the model to the case of *real-time* streams.

Definition 3.5 *A smoothing schedule is said to be a* real-time smoothing schedule *if the sojourn time is the same for all bytes with finite sojourn time.*

In this paper we are interested in real-time schedules for lossless links with constant propagation delay. We denote the propagation delay by P. Note that for real time schedules with constant propagation delay, all non-dropped bytes must have the same queuing delay. We denote this common queuing delay by D.

We close this section with a definition of a local cost function, which is a simplified representation of the fact that not all bytes in a stream have equal importance.

Definition 3.6 *A* local weight function w *for a given input stream $\mathcal{B}$ is a function which assigns a real number $w(b)$ to each byte $b \in \mathcal{B}$. The* benefit *of a real-time schedule S for $\mathcal{B}$ is the sum of weights of all bytes with finite sojourn time.*

Note that the benefit induced by the local weight function which assigns 1 to all bytes is simply the number of bytes delivered by the schedule.

4 Generic Algorithm

In this section we consider schedules whose goal is to maximize the *number* of bytes played out by the client. The algorithm is defined by means of the following quantities: maximum line rate R, queuing delay D, and buffer space $B = \min(\mathsf{B}_s, \mathsf{B}_c)$. Our main result in this section (Theorem 4.5) is that when given any two of these quantities, the optimal choice for the third is determined by the identity

$$B = D \cdot R \tag{1}$$

The buffer space needed at the client and the server is equal to B: making only one of the buffers bigger does not help, as we shall show later.

We remark that this result is the first proof that relates the three quantities in the case of lossy on-line scheduling algorithms.

4.1 Server's Algorithm

The server's job is extremely simple: whenever the server's buffer is non-empty, its contents is transmitted, in FIFO order, to the client at the maximal possible rate. Whenever a new frame is input into the server's buffer, the size of the buffer is checked to ensure that the total number of bytes in the server's buffer does not exceed B. If the server's occupancy becomes $B + Z$ for some $Z > 0$, then *an arbitrary set of Z bytes is discarded from the buffer*. We leave the way information is discarded unspecified at this point.

Formally, to describe our schedule, let $\mathsf{S}'(t)$, $\mathsf{B}'_s(t)$ and $\mathsf{Z}'(t)$ denote the set of bytes sent, stored or dropped, respectively, at time t. (Primed quantities refer to the generic algorithm.) The algorithm is that at time t:

$$|\mathsf{S}'(t)| \quad = \quad \min\Big(R\,,\,|\mathsf{B}'_s(t-1)| + |\mathsf{A}(t)|\Big) \tag{2}$$

$$|\mathsf{Z}'(t)| \quad = \quad \max\Big(0\,,\,|\mathsf{B}'_s(t-1)| + |\mathsf{A}(t)| - |\mathsf{S}'(t)| - B\Big) \tag{3}$$

The actual identity of the bytes dropped is unrestricted, so long as they are available at the server at time t:

$$\mathsf{Z}'(t) \subseteq \mathsf{B}'_s(t-1) \cup \mathsf{A}(t) - \mathsf{S}'(t)\,.$$

For the set of transmitted bytes, however, we require that these are the bytes with the smallest arrival times among all bytes which were not dropped (i.e., the server buffer maintains FIFO ordering).

4.2 Client's Algorithm

The client's algorithm is even simpler: when the first byte b_0 arrives at the client's buffer, a timer is set to D time units. When the timer expires, all bytes whose arrival time is $\mathsf{AT}(b_0)$ (i.e, the bytes of the first frame) currently stored in the client's buffer are played out; thereafter, a frame is displayed every time unit. Formally, we have

$$\mathsf{D}'(t) \quad = \quad \left\{b \,:\, \mathsf{AT}(b) = t - P - D \ \text{ and } \ \mathsf{RT}'(b) \le t\right\} ,$$

where D' and RT' are the times bytes are played and received, respectively, at the client.

4.3 Analysis

We now prove that the generic algorithm above loses the least number of bytes among all algorithms with buffer space B and link rate R. To show optimality, we first state the simple fact that the server transmits as many bytes as possible, subject to the server buffer and link rate constraints. This fact follows from the greedy nature of the server's algorithm.

Lemma 4.1 *Let $\mathsf{S}'(t)$ denote the set of bytes transmitted by the generic algorithm at time t for a given input stream, and let $\hat{\mathsf{S}}(t)$ denote the set of bytes transmitted by any schedule with server buffer size B and link rate R . Then for all t, $\sum_{i=0}^{t} |\mathsf{S}'(i)| \ge \sum_{i=0}^{t} |\hat{\mathsf{S}}(i)|$.*

Since by Lemma 4.1 the server is pushing as many bytes as possible to the link, it is sufficient to show that all bytes which arrive at the client are played out. We prove this fact by showing that there is no client overflow, and that no byte arrives at the client too late.

We first bound the space requirement of the server, and the time any byte spends in the server's buffer.

Lemma 4.2 *The server's buffer requirement under the generic algorithm is B. Moreover, no byte is submitted to the link more than B/R time units after its arrival.*

Proof: The server's buffer requirement is obvious from the algorithm: at any time t, the number of bytes dropped, according to Eq. (3) is exactly the minimal number which ensures that $\mathsf{B}_s(t) \le B$. For the second part of the lemma, note that by the FIFO order of transmission, and since by Eq. (2) the server transmits at maximum rate whenever it is not empty, a byte which arrives at time t is submitted to the link by time $t + \mathsf{B}_s(t)/R$, or else it was dropped by that time. ∎

A direct implication of Lemma 4.2 is that no byte arrives at the client too late (recall that P is the link's constant propagation delay).

Lemma 4.3 *Let t be any time step. For all bytes $b \in \mathsf{R}(t)$, $\mathsf{AT}(b) \ge t - (P + B/R)$. Conversely, for any byte b: if $\mathsf{AT}(b) = t$ and b is not dropped by the server, then $t + P \le \mathsf{RT}(b) \le t + P + B/R$.*

Proof: Follows immediately from Lemma 4.2, and the assumption that there is a constant propagation delay of P time units over the link, i.e., $\mathsf{R}(t) = \mathsf{S}(t - P)$. ∎

Lemma 4.3 shows that no byte b has to be discarded by the client due to missing its deadline, i.e., there is no *underflow*. Next, we prove that there is no overflow at the client buffer.

Lemma 4.4 *The buffer space requirement of the client under the generic algorithm is B.*

Proof: Consider any time step t. Note that all bytes stored in the client's buffer at time t were input to the system after time $t - P - B/R$: bytes with smaller arrival time have already been played out by the client's algorithm, or have been discarded by the server by Lemma 4.3. Also, since we assume that all bytes have fixed propagation delay P over the link, we have that all bytes stored at the client's buffer at time t were input no later than time $t - P$. Formally, for all $b \in \mathsf{B}_c(t)$, we have that $t - P - \frac{B}{R} \le \mathsf{AT}(b) \le t - P$. This, using Lemma 4.3, means that all bytes stored at the client at time t were delivered by the link in the time interval $[t - B/R\,,\, t]$. Therefore, since the link delivers at most R bytes at each time step, we conclude that $|\mathsf{B}_c(t)| \le B$. ∎

Thus we have the following result, which is a direct consequence of the lemmas above.

Theorem 4.5 *Let $\mathsf{Z}'(t)$ denote the set of bytes dropped by the generic algorithm at time t for a given input stream, and let $\hat{\mathsf{Z}}(t)$ denote the set of bytes dropped by any real-time schedule with server buffer size B and link rate R. Then for all t, $\sum_{i=0}^{t} |\mathsf{Z}'(i)| \le \sum_{i=0}^{t} |\hat{\mathsf{Z}}(i)|$.*

4.4 Negative Results

In the full paper we study the quantitative effect of setting one of the three parameters (buffer size, link rate and queuing delay) to a non-optimal value. The result is invariably either resource wastage or unnecessary information loss: the question is how bad can it get. We only state the results here.

Theorem 4.6 *For any given B and R, there exist an arbitrarily long input stream such that any schedule with server buffer of size B, link of rate R, and queuing delay $D < B/R$ loses, on average, at least $R\left(1 - \frac{DR}{B}\right)$ bytes per frame. On the other hand, there exists a smoothing schedule with buffer size B, link rate R and delay $D = B/R$ that does not lose any byte for that sequence.*

Theorem 4.7 *For any given B and R, there exist an arbitrarily long input stream such that any schedule with server buffer of size B, link of rate R, and queuing delay $D > B/R$ loses, on average, at least $R\left(1 - \frac{2B+PR}{DR+PR+B}\right)$ bytes per frame. On the other hand, there exists a smoothing schedule with buffer size B, link rate R and delay $D = B/R$ that does not lose any byte for that sequence.*

Theorem 4.8 *For any given D and R, there exist an arbitrarily long input stream such that any schedule with playout delay D, link of rate R, and buffers such that $\min(\mathsf{B}_s, \mathsf{B}_c) < DR$ must lose, on average, at least $R\left(1 - \frac{B}{DR}\right)$ bytes per frame. On the other hand, there exists a smoothing schedule with playout delay D, link rate R and buffer size $B = DR$, that does not lose any byte for that sequence.*

Theorem 4.9 *For any given D and B, and $R < B/D$, the input sequence where all the frames are of size B/D must lose, on* *average, at least $R\left(\frac{B}{D}-1\right)$ bytes per frame. On the other hand, there exists a smoothing schedule with delay D, buffer size B and link rate $R = B/D$ that does not lose any byte for that sequence.*

5 Competitive Analysis for General Local Cost Functions

In this section we generalize our treatment to general local cost functions, i.e., we address the case that different bytes have different weights, and the benefit of a real-time schedule is the sum of the weights of the bytes played.

Our main tool here is competitive analysis. In competitive analysis, one compares the performance of an on-line algorithm to the performance of an off-line algorithm. In our case, the competitive analysis is done as follows. For each input stream $\mathcal{B}$, we compute the benefit of the on-line algorithm, denoted by $online(\mathcal{B})$, to the benefit of an optimal off-line algorithm, denoted by $opt(\mathcal{B})$. An optimal off-line algorithm knows all the input stream in advance, and based on the entire stream it finds an optimal schedule. The competitive ratio for an input stream $\mathcal{B}$ is the ratio of the two benefits. The competitive ratio of an on-line algorithm is the worst-case competitive ratio over all input steams. An on-line algorithm is called c-competitive, if for any input sequence $\mathcal{B}$, we have that $opt(\mathcal{B})/online(\mathcal{B}) \leq c$.

In this section, we present the following results. First, we consider a natural greedy algorithm, and prove that it is 4-competitive. We also show that the greedy algorithm cannot be better than $2 - \frac{2}{B}$-competitive. (In Section 6 we provide evidence that show that the performance of the Greedy algorithm in practice is much better.) We then prove a lower bound of 1.25 on the competitive ratio of any deterministic on-line algorithm. We require the on-line scheduler to adhere to FIFO scheduling: it is clear that there exists an optimal off-line schedule which maintains FIFO order; also, this is a very natural restriction when discussing real-time smoothing.

For our algorithm, we choose a buffer size such that $B = DR$. Theorem 4.5, and Lemmas 4.3 and 4.4, guarantee that if $B = DR$ then no overflow or underflow will occur at the client buffer. This allows us to restrict attention to the server buffer and the bytes dropped from it, since any other byte will reach the client and played back on time.

5.1 The Greedy Algorithm

When we a have general local cost function, it seems natural to drop bytes with low weight in exchange for bytes with high weights, which gives rise to the following rule of greedy schedules:

> *Each time an overflow occurs, the B top-weight bytes among all available bytes are kept at the server's buffer.*

We resolve ties as follows: if the arrival times of equal-weight bytes are different, we keep older bytes; if both the weights and arrival times are equal, we choose arbitrarily.

We now analyze the competitive ratio of the greedy algorithm. First we prove an upper bound of 4, and then a lower bound of nearly 2.

Theorem 5.1 *The competitive ratio of the greedy algorithm is at most* 4.

The following concept is central in the proof of Theorem 5.1.

Definition 5.1 *For any time interval I, let $V(I)$ denote the set of the B highest weight bytes among all bytes arriving in I. If fewer than B bytes arrive in I, then $V(I)$ consists of all of the arriving bytes.*

We start with a lemma that relates the benefit of the greedy algorithm to the weight of the bytes in $V(I)$ for intervals of length D. The following notation is useful.

Notation 5.2 *For any set of bytes A, let $w(A) \stackrel{\text{def}}{=} \sum_{b\in A} w(b)$.*

Lemma 5.2 *In the greedy algorithm, for any interval $I = [t, t+D-1]$,*

$$\sum_{i=0}^{D-1} w(\mathsf{S}(t+i)) \geq w(V(I)) - w(\mathsf{B}_s(t+D)) .$$

Proof: Define the set Y to consist of the B highest-weight bytes from

$$\mathsf{B}_s(t) \cup \bigcup_{i=0}^{D-1} \mathsf{A}(t+i) ,$$

i.e., Y is the set of the B "heaviest" bytes among all bytes that are either in the buffer at the start of I, or arrive during I (ties are resolved in the same way they are resolved by the greedy algorithm). We claim that all bytes in Y are either sent during I or stored in the buffer by the end of I. Formally:

$$Y \subseteq \mathsf{B}_s(t+D) \cup \bigcup_{i=0}^{D-1} \mathsf{S}(t+i) . \tag{4}$$

For suppose that Eq. (4) is false. Then there exists $t' \in I$ such that $Y \cap \mathsf{Z}(t') \neq \emptyset$, i.e., a byte from Y is dropped at time t'. Let t_0 be the first such time, and b a byte from Y that was dropped at time t_0. By the rule of the greedy algorithm, we have that $w(b) < w(b')$ for all $b' \in \mathsf{B}_s(t_0)$, which means that there are B bytes with weight higher than $w(b)$. Since $\mathsf{B}_s(t_0) \subseteq \mathsf{B}_s(t) \cup \bigcup_{i=0}^{D-1} \mathsf{S}(t+i)$, this is a contradiction to the definition of Y. We can therefore use Eq. (4) to conclude that

$$\sum_{i=0}^{D-1} w(\mathsf{S}(t+i)) \; + w(\mathsf{B}_s(t+D) \geq \; w(Y) ,$$

and the result follows, since $w(Y) \geq w(V(I))$ by definition. ∎

Lemma 5.3 *In the greedy algorithm, for any time step t,*

$$w(\mathsf{B}_s(t)) \leq \sum_{i=0}^{D-1} w(\mathsf{S}(t+i)) .$$

Proof: Let X_{t+i} be the set of bytes transmitted in the time interval $[t, t+i]$ and the oldest $B - iR$ bytes in $\mathsf{B}_s(t+i)$. Note that $X_t = \mathsf{B}_s(t)$ and $X_{t+D-1} = \cup_{i=0}^{D-1}\mathsf{S}(t+i)$.

We claim that for all $0 \le i \le D-1$, $w(X_{t+i}) \le w(X_{t+i+1})$. To see that, consider at time $t+i+1$. First a set of bytes $\mathsf{A}(t+i+1)$ arrive. Then some of the bytes in $\mathsf{B}_s(t+i)$ may be dropped, but by definition of the greedy algorithm, each byte dropped has smaller weight than all remaining bytes in $\mathsf{B}_s(t+i+1)$. Therefore, this can only increase the weight of X_{t+i}. Secondly, the first R bytes in $\mathsf{B}_s(t+i)$ are transmitted, but the definition of X_{t_i+1} makes sure that this does not affect the set of bytes we are considering. It follows from transitivity that

$$w(\mathsf{B}_s(t)) = w(X_t) \le w(X_{t+D-1}) = \sum_{i=0}^{D-1} w(\mathsf{S}(t+i)) . \quad \blacksquare$$

Combining Lemmas 5.2 and 5.3 we have the following corollary.

Lemma 5.4 *In the greedy algorithm, for any interval $I = [t, t+D-1]$,*

$$2\sum_{t\in I} w(\mathsf{S}(t)) \ + w(\mathsf{B}_s(t+D)) - w(\mathsf{B}_s(t)) \ge \ w(V(I)) .$$

On the other hand, the following simple observation relates the optimal benefit to the weight of $V(I)$ for any D-length interval I.

Lemma 5.5 *Let I be any time interval of $\ell \le D$ steps. Then the total weight of bytes arriving at I that are not dropped by any B-space algorithm is at most $2w(V(I))$.*

Proof: Obviously, in any interval of ℓ time steps, no B-space algorithm can accept (i.e., not drop) more than $B + \ell R$ bytes (that's the "leaky bucket" nature of the buffer). Since the bytes in $V(I)$ have the greatest weight among the bytes arriving in I, it follows that no B-space algorithm can accept total weight of more than $(1 + \frac{\ell R}{B})w(V(I))$. The result follows since by assumption, $\ell \le D$ and hence $\ell R \le B$. $\blacksquare$

Using Lemmas 5.5 and 5.4 we can now prove the theorem.

Proof of Theorem 5.1: Let T denote the last time step of the greedy schedule. Without loss of generality, we may assume that $\mathsf{B}_s(T+1) = \emptyset$. Divide time into intervals $\{I_j\}_j$ of length D (the last interval may be shorter than D). Summing over all intervals I_j we get from Lemma 5.4 that

$$\begin{aligned}\sum_j w(V(I_j)) \ &\le \ w(\mathsf{B}_s(T+1)) - w(\mathsf{B}_s(0)) + 2\sum_{t=0}^{T} w(\mathsf{S}(t)) \\ &= \ 2\sum_{t=0}^{T} w(\mathsf{S}(t)) , \end{aligned} \tag{5}$$

by our assumption that $w(\mathsf{B}_s(0)) = w(B_s(T+1)) = 0$. Eq. (5) means that the total benefit of the greedy algorithm is at least the sum of the $w(V(I_j))$. On the other hand, using Lemma 5.5 and summing over all intervals, we get that the total benefit of the optimal algorithm is at most $2\sum_j w(V(I_j))$. Combining with Eq. (5) completes the proof. $\blacksquare$

Next we show that 2 is a lower bound on the competitive ratio of the greedy algorithm.

Theorem 5.6 *There is an input stream $\mathcal{B}$ on which the weight of the optimal schedule is at least $2 - \epsilon$ times the weight of the greedy scheduling, for $\epsilon \ge 2/B$.*

Proof: Assume that the link rate $R = 1$, and consider the following input stream start at time 0.

- At time 0, B bytes arrive, with weight 1 each, i.e., $\mathsf{A}(i) = 0$ and $w(i) = 1$ for $1 \le i \le B$.
- In each of the next B time steps, a single byte of value L arrives, i.e., $\mathsf{A}(i+B) = i$ and $w(i+B) = L$ for $1 \le i \le B$.
- Finally, at time $B+1$, B bytes of value L each arrive, i.e., $\mathsf{A}(i+2B) = B+1$ and $w(i+2B) = L$ for $1 \le i \le B$.

The greedy algorithm accepts the first $2B+1$ bytes at steps $0, \ldots, B$, since there are empty slots in the buffer. Thus, after step B, the buffer contains B bytes of weight L each, and hence, at step $B+1$, B bytes of weight L are dropped by the greedy algorithm. The overall benefit of the greedy algorithm is $1 \cdot B + L \cdot (B+1)$. The optimal (off-line) algorithm drops all weight-1 bytes, and can therefore send all the bytes of weight L, resulting in total benefit of $1 + 2LB$. For $L = B(B-1)$ we get,

$$\frac{1+2LB}{B+L(B+1)} = 2 - \frac{2B^2-1}{B^3} > 2 - \frac{2}{B} ,$$

and the theorem follows. $\blacksquare$

5.2 A Lower Bound for Deterministic On-line Algorithms

It turns out that no deterministic on-line algorithm can be very close to optimal, as we show next. Our proof uses the assumption that on-line algorithms must send the bytes in FIFO order.

Theorem 5.7 *For any deterministic on-line algorithm $\mathbf{A}$, there exists an input sequence $\mathcal{B}$ such that the total weight of an optimal schedule of $\mathcal{B}$ is at least $5/4$ times the total weight of the on-line algorithm $\mathbf{A}$, i.e., $opt(\mathcal{B})/\mathbf{A}(\mathcal{B}) \ge 1.25$.*

Proof: Assume that the link rate is $R = 1$. We consider two scenarios. In both scenarios, the arrival sequence starts at time $t = 0$ with B bytes of value 1, i.e., $\mathcal{B}(i) = 0$ and $w(i) = 1$ for $1 \le i \le B$. In each of the next time steps a single byte of value L arrives, i.e., $\mathcal{B}(i+B) = i$ and $w(i+B) = L$. This continues until we reach time B or until $\mathbf{A}$ sends an L value byte (which means that $\mathbf{A}$ has dropped all the remaining bytes of value 1 from the buffer). Let t_1 be that time.

In the first scenario, the input stream ends at time t_1. The benefit of $\mathbf{A}$ is $1 \cdot t_1 + L \cdot (B - t_1)$, while the off-line benefit is $1 \cdot B + L \cdot (B - t_1)$.

In the second scenario, at time $t_1 + 1$ a burst of B bytes of value L arrive. The off-line benefit in this case is $L(B + t_1)$, while the benefit of $\mathbf{A}$ is $1 \cdot t_1 + L \cdot B$.

For $L = 2$ the ratio is the minimum of $\frac{1+2(1-z)}{z+2(1-z)}$ and $\frac{2(1+z)}{2+z}$, where $z = t_1/B$. Optimizing, we get that for $z = 2/3$ the ratio is $5/4$. $\blacksquare$

6 Experimental Results

In this section we present some typical numbers for video smoothing for clips found on the Internet. We also evaluate the performance of two on-line algorithms in the case of a cost-based model.

6.1 Typical Numbers

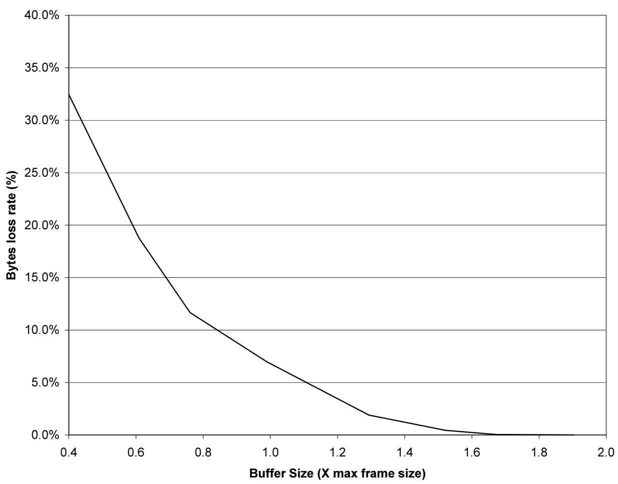

Figure 2: Data loss rate as a function of the size of the buffer. The link rate is 7% more than the average rate.

We have experimented with a few types of clips. In Figure 2, the results of smoothing a typical newscast clip (obtained from CNN archive [1]). As can be seen, a very small buffer (less than twice the largest frame size, or alternatively 8 times the size of an average frame) suffices to avoid losses altogether, with peak link rate only 7% more than the average rate. But if the buffer size is too small, the loss is substantial.

An illustration of the way our algorithm works is provided in Figure 3, where the buffer occupancy at the client and the server may be seen. Note that the client never overflows, while the server has to truncate the excess flow from time to time.

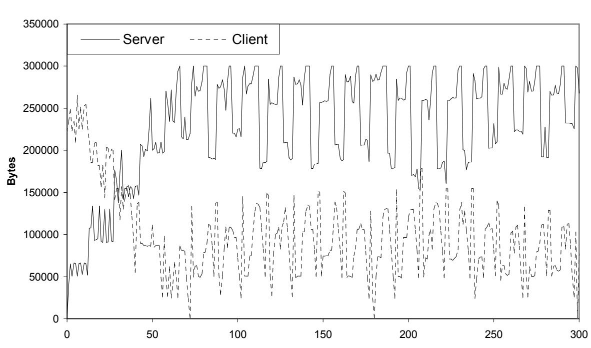

Figure 3: Buffer occupancy at the server and the client with 300Kbytes buffers. Note the symmetric trends.

6.2 Comparison in the cost model

In another set of experiments, we tested the *cost* lost in our simple model: given an MPEG stream, we assigned costs to bytes based on the type of frame they are a part of. Specifically, we assigned costs of $12 : 8 : 1$ for $I : P : B$ frames, respectively. We compared two simple algorithms. In the *FIFO* algorithm (a.k.a. *tail drop* algorithm), if an overflow occurs at time i, then bytes from frame i are discarded (intuitively, all overflow is from the "tail" of the server's buffer). The second algorithm we test is the *greedy algorithm*, in which whenever an overflow occurs, the cheapest bytes are discarded, regardless of their location in the server's buffer or in the current input frame. For comparison purposes, we also plot the optimal loss possible (off-line) for the given buffer size and link rates.

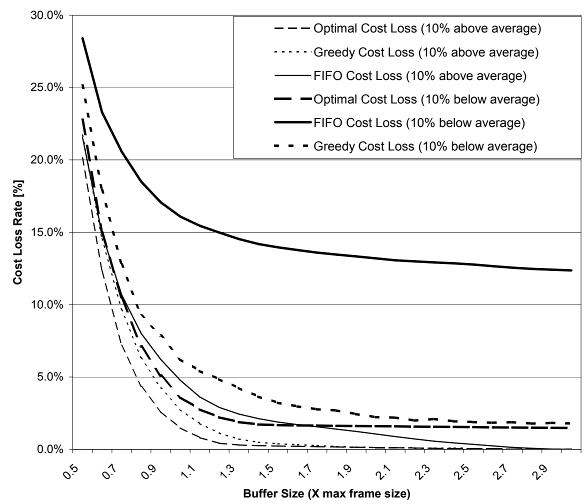

Figure 4: Fraction of lost cost in a high quality clip of the FIFO greedy algorithms under two link rates.

Clearly, both on-line algorithms perform the same if no (or very little) loss occurs. In Figure 4 we see a comparison of the total cost the algorithms lost due to overflow, for a high-quality input stream. As expected, the greedy algorithm outperforms the FIFO algorithm in all cases, and the overall loss is never too high, in terms of cost. Note that link rate 10% below average implies that at least 10% of the information is lost, but the cost lost may be significantly lower (as is the case for the Greedy algorithm and the optimal schedule). Unfortunately for the FIFO algorithm, in MPEG streams costly bytes come in large bursts, and hence the cost lost by FIFO is more than 10%.

In Figure 5, we look more closely at the overflow of the greedy and the FIFO algorithm. We can see that even though the number of bytes discarded is equal, the FIFO algorithm loses many "expensive" bytes.

In Figure 6, we plot the benefit of Greedy and FIFO relative to the optimal benefit as we vary the link rate. It can be seen that the Greedy algorithm manages to salvage most of the benefit even when the rate drops much below the average rate. One way to view this is that using the greedy algorithm, one can trade link capacity for additional processing in the server.

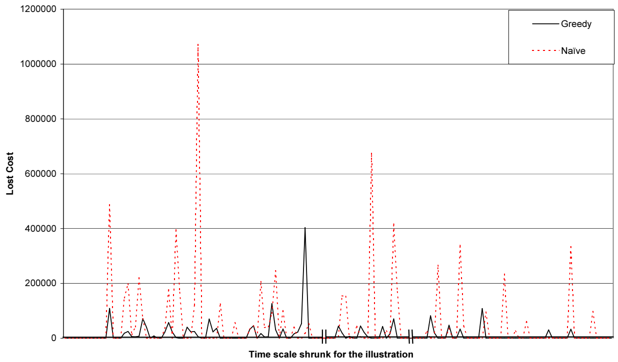

Figure 5: Three segments of overflow traces. In each segment, the number of *bytes* lost is the same for both the greedy algorithm and the FIFO algorithm, but the greedy algorithm loses much less in terms of cost.

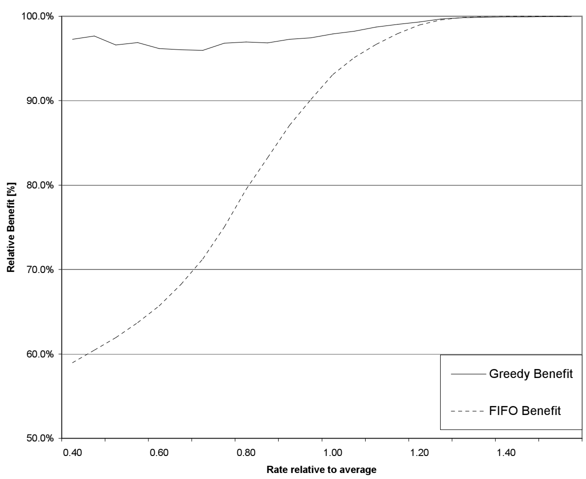

Figure 6: Benefit of FIFO and Greedy relative to optimal under varying link rate.

References

[1] `www.nmis.org/NewsInteractive/CNN/Newsroom`.

[2] MPEG-1 standard (ISO/IEC 11172), 1992.

[3] MPEG-2 standard (ISO/IEC DIS 13818), 1994.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[5] R.-I. Chang, M.-C. Chen, J.-M. Ho, and M.-T. Ko. An effective and efficient traffic smoothing scheme for delivery of online VBR media streams. In *Proceedings of IEEE INFOCOM*, 1999.

[6] N. G. Duffield, K. K. Ramakrishnan, and A. R. Reibman. SAVE: An algorithm for smoothed adaptive video over explicit rate networks. *IEEE/ACM Transactions on Networking*, 6(6):717–728, 1998.

[7] W. Feng and J. Rexford. Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video. *IEEE Trans. on Multimedia*, Sept. 1999. To appear.

[8] M. Grosslauser, S. Keshav, and D. N. C. Tse. RCBR: A simple and efficient service for multiple time-scale traffic. *IEEE/ACM Transactions on Networking*, 5(6):741–755, Dec. 1997.

[9] T. Y. J. Ni and D. Tsang. A CBR transport technique for MPEG-2 video-on-demand connections over ATM networks. In *Proc. IEEE ICC 96*, pages 1391–1395, June 1996.

[10] Z. Jiang and L. Kleinrock. A general optimal smoothing video algorithm. In *Proc. IEEE INFOCOM*, Mar. 1999.

[11] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley Publishing Co., 1997.

[12] S. S. Lam, S. Chow, and D. K. Y. Yau. An algorithm for lossless smoothing of MPEG video. In *Proc. ACM SIGCOMM*, London, England, 1994.

[13] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley. Online smoothing of live, variable-bit-rate video. In *Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 249–257, May 1997.

[14] J. Rexford and D. Towsley. Smoothing variable-bit-rate video in an internetwork. *IEEE/ACM Transactions on Networking*, pages 202–215, Apr. 1999.

[15] J. Salehi, Z. Zhang, J. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. *IEEE/ACM Transactions on Networking*, 6(4):397–410, Aug. 1998.

[16] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proc. IEEE INFOCOM*, Mar. 1999.

[17] The ATM Forum Technical Committee. Traffic management specification version 4.0, Apr. 1996. Available from `www.atmforum.com`.

[18] D. E. Wrege, W. Knightly, Zhang, and J. Liebeherr. Deterministic delay bounds for VBR video in packet-switching networks: fundamental limits and practical trade-offs. *IEEE/ACM Transactions on Networking*, 4(3):352–362, June 1996.

[19] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R. P. Tsang. Efficient selective frame discard algorithms for stored video delivery across resource constrained networks. In *Proc. IEEE INFOCOM*, Mar. 1999.

[20] W. Zhao, T. Seth, M. Kim, and M. Willebeek-LeMair. Optimal bandwidth/delay tradeoff for feasible-region-based scalable multimedia scheduling. In *Proc. IEEE INFOCOM 98*, 1998.