

A Moving Average Predictor for Playout Delay Control in VoIP

Víctor M. Ramos R.^{1*}, Chadi Barakat², and Eitan Altman²

¹ University of Nice-Sophia Antipolis, France. Victor.Ramos@ieee.org

² INRIA-Sophia Antipolis, France. {cbarakat,altman}@sophia.inria.fr

Abstract. Audio applications are now widely used in the Internet. Such applications require receiver playout buffers to smooth network delay variations and to reconstruct the periodic form of the transmitted packets. Packets arriving after their playout deadline are considered late and are not played out. Existing algorithms used in the Internet operate by adaptively adjusting the playout delay from talkspurt to talkspurt. There is an important tradeoff between loss percentage and average playout delay. Current algorithms fail to obtain a particular loss percentage. Controlling this parameter is a key characteristic for any playout adaptation algorithm. This paper presents a Moving Average algorithm for playout delay adaptation with tunable loss percentage. We show with trace-based simulations that, in most of the cases, our algorithm performs better than those implemented in popular audio tools, and this is for the range of loss rates of interest in interactive audio applications.

1 Introduction

Delay, jitter, and packet loss in packet-switched wide-area networks are the main factors impacting audio quality of interactive multimedia applications. Today's Internet still operates in a best-effort basis, and thus, the impact of jitter, delay, and packet loss must be alleviated by employing end-to-end control mechanisms.

Audio applications such as NeVoT [1] and Rat [2] generate packets spaced at regular time intervals. The traffic generated by an audio source is divided into periods of activity, called *talkspurts*, and periods of silence. Silence periods are periods where no audio packets are transmitted.

Audio packets encounter variable delay while crossing the Internet, which is mainly due to the variable queuing time in routers. This delay variability modifies the periodic form of the transmitted audio stream. To playout the received stream, an application must reduce or eliminate this delay variability, by buffering the received packets and playing them out after a certain deadline. Packets arriving after their corresponding deadline are considered late and are not played out. If the playout delay is increased, the probability that a packet will arrive before its scheduled playout time also increases. This reduces the number of packets artificially dropped in the playout buffer. However, very long playout delays have a negative impact on the interactivity of an audio session. Obviously, there exists a trade-off between delay and loss due to late packets. For

* Víctor Ramos is also with the Universidad Autónoma Metropolitana, Mexico.

interactive audio, packet delays up to 400 ms and loss rates up to 5% are considered adequate [3].

We focus in this paper on the tradeoff between loss and delay for playout delay control algorithms. Using measurements of packet end-to-end delay of audio sessions done with NeVoT, we present and validate a Moving Average (MA) algorithm that adjusts the playout delay at the beginning of each talkspurt. To prove the efficiency of our algorithm, we compare it with earlier work done by Ramjee et al. [4]. We present two versions of our algorithm: an offline algorithm and an online one. First, we explain the offline MA algorithm, which serves as a reference for our work. Then, we show how an online hybrid algorithm can be implemented by combining the ideas proposed by Ramjee et al. with the moving average algorithm we propose.

Moving average estimators have been used in different fields of research. For example, in [5] Zhang et al. show that moving average estimation is a good approach to predict end-to-end delay for TCP connections. We use a moving average technique not for predicting end-to-end delay, but rather for predicting an optimal value of the playout delay per-talkspurt in an audio session.

One characteristic that most of the playout delay adaptation algorithms lack is the ability to fix the loss percentage to some a priori value. By changing a measure of variability, the algorithms proposed by Ramjee et al. can achieve different loss percentages. However, there is no explicit relationship between the measure of variability that we can adapt in these algorithms and the average loss percentage. The average loss percentage can change from one audio session to another, even if this parameter is kept unchanged. Here lies the main contribution of our work. The moving average algorithm we propose adjusts the playout delay from talkspurt to talkspurt, given a desired target of average loss percentage p . Our algorithm ensures that the average loss percentage we obtain during the session is close, if not equal, to the target value. At the same time, and in most of the cases, our algorithm realizes this target with a smaller average playout delay than the one we need to obtain the same average loss percentage with the algorithms proposed by Ramjee et al. For practical loss percentages, we validate our algorithm and those of Ramjee et al. using real packet audio traces. By using collected audio traces we can compare the algorithms under the same network conditions.

The remainder of this paper is as follows. Section 2 provides some additional background and describes the packet delay traces used for validation, as well as the notation we use throughout the paper. In Sect. 3, we describe some related work on playout delay algorithms. Section 4 describes the performance measures we consider to validate our algorithm and to compare it with the algorithms of Ramjee et al. We present in Sect. 5 our moving average algorithm. Section 6 describes, based on results from Sect. 5, that it is better to predict a function of the delay rather than the optimal delay itself. Section 7 compares the performance of an online hybrid implementation of the moving average algorithm with the Ramjee et al. algorithms. Finally, Sect. 8 concludes the paper.

2 Some background

Receivers use a playout buffer to smooth the stream of audio packets. This smoothing is done by delaying the playout of packets to compensate for variable network delay.

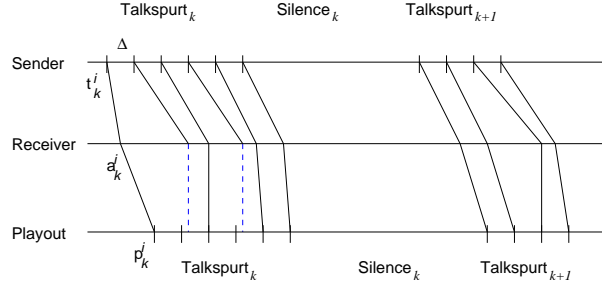


Fig. 1. The timings between the transmission, reception and playout of packets.

The playout delay can be either constant through the whole session, or can be adjusted between talkspurts. Moreover, in a recent work [6], it has been shown that by using a technique called *packet scaling*, it is possible to change the playout delay from packet to packet while keeping the resulting distortion within tolerable levels. In this paper we only focus on the per-talkspurt playout delay adaptation approach.

Figure 1 shows the different stages incurred in an audio session. The i -th packet of talkspurt k is sent at time t_k^i , it arrives at the receiver at time a_k^i , and is held in the smoothing receiver's playout buffer until time p_k^i , when it is played out. Inside a talkspurt, packets are equally spaced at the sender by time intervals of length Δ seconds.

By delaying the playout of packets and dropping those that arrive after their deadline, we are able to reconstruct the original periodic form of the stream. This adaptation results in a regenerated stream at having stretched or compressed silence periods compared to the original stream. These changes are not noticeable by the human ear if they are kept within tolerable small levels.

In Fig. 1, a dropped packet due to a late arrival is represented by a dashed line. A packet is artificially dropped if it arrives after its scheduled deadline p_k^i . The loss percentage can be reduced by increasing the amount of time that packets stay in the playout buffer. An efficient playout adaptation algorithm must take into account the trade-off between loss and delay in order to keep both parameters as low as possible.

Table 1. Definition of variables.

Param.	Meaning
L	The total number of packets arriving at the receiver during a session.
N	The total number of talkspurts in a session.
N_k	The number of packets in talkspurt k .
t_k^i	The time at which the i -th packet of talkspurt k is generated at the sender.
a_k^i	The time at which the i -th packet of talkspurt k is received.
d_k^i	The variable portion of the end-to-end delay of the i -th packet in talkspurt k . $d_k^i = a_k^i - t_k^i - \min_{\substack{1 \leq k \leq N \\ 1 \leq i \leq N_k}} (a_k^i - t_k^i).$
p_k^i	The time at which packet i of talkspurt k is played out.

Table 2. Description of the traces.

Trace	Sender	Receiver	Start time	Length [s]	Talkspurts	Packets
1	UMass	GMD Fokus	08:41pm 6/27/95	1348	818	56979
2	UMAss	GMD Fokus	09:58am 7/21/95	1323	406	24490
3	UMAss	GMD Fokus	11:05am 7/21/95	1040	536	37640
4	INRIA	UMass	09:20pm 8/26/93	580	252	27814
5	UCI	INRIA	09:00pm 9/18/93	1091	540	52836
6	UMass	Osaka University	00:35am 9/24/93	649	299	23293

Throughout the paper, we use the notation described in Table 1. For the validation of our algorithm, we consider the packet traces generated with the NeVoT audio tool that are described in [7]. NeVoT is an audio terminal program used to establish audio conversations in the Internet. The traces we use contain the sender and receiver timestamps of transmitted packets that are needed for the implementation of any playout adaptation algorithm. In these traces, one 160 byte audio packet is generated approximately every 20 ms when there is speech activity. A description of the traces (reproduced from [7]) is depicted in Table 2.

A typical sample of packet end-to-end delays is shown in Fig. 2. A packet is represented by a diamond and talkspurt boundaries by dashed rectangles. The x -axis represents the time elapsed at the receiver since the beginning of the audio session. Only the variable portion of the end-to-end delay (d_k^i) is represented on the y -axis of Fig. 2. To this end, the constant component of the end-to-end delay (mostly caused by the propagation delay) is removed by subtracting from packet delays their minimum over all the corresponding trace. By considering the variable portion of the end-to-end delay, synchronization between sender and receiver clocks can be avoided³ [7].

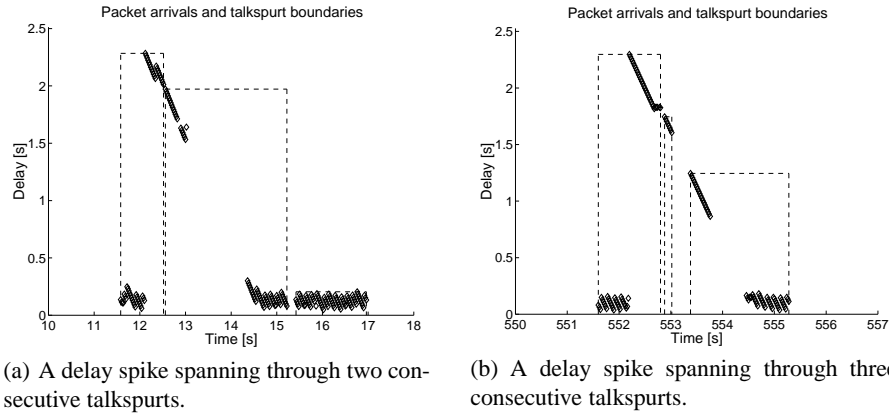


Fig. 2. Delay spikes in end-to-end delay measurements.

³ Later, when comparing the performance of different algorithms, all graphics consider the variable portion of end-to-end delays.

We observe in Fig. 2 the presence of delay spikes. This phenomenon in end-to-end delay has been previously reported in the literature [4, 8]. Delay spikes represent a serious problem for audio applications since they affect the performance of playout delay adaptation algorithms. A *delay spike* is defined as a sudden large increase in the end-to-end delay followed by a series of packets arriving almost simultaneously, leading to the completion of the spike [4].

Delay spikes can be contained within a single talkspurt or can span over several talkspurts. Figure 2(a) shows a delay spike spanning through two consecutive talkspurts. Figure 2(b) shows a delay spike spanning over three talkspurts. Since the playout delay is generally changed between talkspurts, a playout algorithm behaves better when delay spikes span over more than one talkspurt. Only in this way, a playout algorithm can react adequately to the spike by setting the playout delay according to the experienced delay. If the spike vanishes before the end of a talkspurt, the playout algorithm will not have enough time to set the playout time accordingly.

In the next section, we briefly describe the algorithms proposed by Ramjee et al. Playout delay is adapted from talkspurt to talkspurt based on past statistics of the delay process. The playout delay of the first packet of each talkspurt is the basetime of the deadlines for subsequent packets in the same talkspurt. This principle is the basis for most of the existing playout adaptation algorithms [4, 7, 9, 10].

3 Related work

Extensive research work has been done in the area of adaptive playout mechanisms [4, 6, 7, 9–12]. In [4], Ramjee et al. propose four algorithms for playout delay. All the four algorithms proposed in [4] compute an estimate of the average end-to-end delay and a measure of variability of delay similarly to the computation of round-trip-time estimates by TCP for the retransmission timer. We denote them as \hat{d}_k^i and \hat{v}_k^i respectively. These statistics are used to set the playout time for a talkspurt.

The algorithms that perform better in [4] are 1 and 4. We rename these algorithms as *A* and *B* respectively, and refer to them as such throughout the rest of the paper.

To calculate \hat{d}_k^i and \hat{v}_k^i , the packet's sender and receiver timestamps, t_k^i and a_k^i , are read from a trace file. Both algorithms differ only in the way they calculate \hat{d}_k^i and \hat{v}_k^i . Algorithm *A* computes these statistics as follows:

$$\hat{d}_k^i = \alpha \hat{d}_k^{i-1} + (1 - \alpha) d_k^i, \quad \text{and} \quad \hat{v}_k^i = \alpha \hat{v}_k^{i-1} + (1 - \alpha) |\hat{d}_k^i - d_k^i|,$$

where $d_k^i = a_k^i - t_k^i$, and α has the default value of 0.998002.

Algorithm *B* is described in [4] but we also sketch it in Fig. 3 for completeness. It operates in two modes: normal mode and spike (or impulse) mode. In normal mode, it behaves like Algorithm *A* but with different coefficients. When the difference between two consecutive delay values exceeds a given threshold, algorithm *B* triggers the spike mode. During this mode, the variable *var* is updated with an exponentially decreasing value that adjusts to the slope of the spike. The end of a delay spike is detected when *var* reaches an enough small value, and the algorithm returns to normal mode.

```

1.  $d_k^i = a_k^i - t_k^i$ ;
2. if (mode == NORMAL) {
3.   if ( $|d_k^i - d_k^{i-1}| > 2|\hat{v}_k^i| + 800$ ) {
4.     var = 0; /* Detected beginning of spike */
5.     mode = SPIKE;
6.   }
7. }
8. else {
9.   var = var/2 +  $|(2d_k^i - d_k^{i-1} - d_k^{i-2})/8|$ ;
10.  if (var  $\leq 63$ ) {
11.    mode = NORMAL; /* End of spike */
12.     $d_k^{i-2} = d_k^{i-1}$ ;
13.     $d_k^{i-1} = d_k^i$ ;
14.    return;
15.  }
16. }
17. if (mode == NORMAL)
18.    $\hat{d}_k^i = 0.125d_k^i + 0.875\hat{d}_k^{i-1}$ ;
19. else
20.    $\hat{d}_k^i = \hat{d}_k^{i-1} + d_k^i - d_k^{i-1}$ ;
21.  $\hat{v}_k^i = 0.125|d_k^i - \hat{d}_k^i| + 0.875\hat{v}_k^{i-1}$ ;
22.  $d_k^{i-2} = d_k^{i-1}$ ;
23.  $d_k^{i-1} = d_k^i$ ;
24. return;

```

Fig. 3. Algorithm *B*

Once \hat{d}_k^i and \hat{v}_k^i are computed, the playout time of the i -th packet of talkspurt k is set by both algorithms as follows:

$$p_k^i = \begin{cases} t_k^i + \hat{d}_k^i + \beta\hat{v}_k^i, & \text{for } i = 1. \\ p_k^1 + (t_k^i - t_k^1), & \text{for } 1 < i \leq N_k. \end{cases} \quad (1)$$

These values are computed for each packet but the playout time is changed only at the beginning of a talkspurt. By varying β one is able to achieve different loss probabilities and different average playout delays. In [4], β is set equal to the constant value of 4. Larger values of β allow to obtain lower loss percentages due to late arrivals but at the cost of a longer average playout delay.

Algorithm *A* is slow in detecting delay spikes, but it maintains a good average playout delay over an audio session. Algorithm *B* reacts faster to delay spikes, but it underestimates the playout delay at the end of the spike [7].

To compare our moving average algorithm with algorithms *A* and *B*, we use the performance measures defined in [7]. For clarity of the presentation, we redefine these measures in Sect. 4.

4 Performance measures

To assess the performance of a playout adaptation algorithm, we focus on the total number of packets that are played out during an audio session, as well as on the experienced average end-to-end delay. Suppose we are given a packet audio trace with the sender and receiver timestamps of audio packets. Let p_k^i , N , L , N_k , t_k^i , and a_k^i be defined as in Table 1. As in [7], we define r_k^i to be a variable indicating if packet i of talkspurt k is played out or not. So, r_k^i is defined as:

$$r_k^i = \begin{cases} 0, & \text{if } p_k^i < a_k^i. \\ 1, & \text{otherwise.} \end{cases}$$

The total number of packets, T , played out in an audio session is thus given by:

$$T = \sum_{k=1}^N \sum_{i=1}^{N_k} r_k^i. \quad (2)$$

The average playout delay, D_{avg} , is equal to :

$$D_{avg} = \frac{1}{T} \sum_{k=1}^N \sum_{i=1}^{N_k} r_k^i [p_k^i - t_k^i]. \quad (3)$$

Finally, the loss percentage, l , is equal to :

$$l = \frac{L - T}{L} \times 100. \quad (4)$$

5 Moving Average prediction

Algorithms A and B are good in maintaining a low overall average playout delay and reacting to delay spikes. However, they lack a parameter allowing to have a direct control on the overall loss percentage during an audio session. It would be desirable to come up with an algorithm that sets the playout delay in a way to get a loss percentage p , given a trace of N talkspurts and L packets. By varying the parameter β in algorithms A and B , it is possible to obtain different loss percentages, but one is unable to have any particular control on this parameter. We describe in this section our moving average predictor (MA) for playout delay that takes as input the desired loss percentage per-session, p , and a packet delay trace. It returns a set of estimated playout delay values leading to an average loss percentage close, if not equal to the desired value p .

5.1 The Model

Let D_k be the optimal playout delay at the beginning of talkspurt k , and let p be the desired average loss percentage per-session. We mean by *optimal playout delay* the playout delay that makes the number of losses per talkspurt the closest to $p \times N_k$, N_k being the number of audio packets received during the k -th talkspurt. By controlling the loss percentage per-talkspurt to p , we are sure that the overall loss percentage during the

whole audio session is also close to p . We compute D_k as follows, let d_k^j be the variable portion of the end-to-end delay of the j -th packet in talkspurt k . For each talkspurt, $1 \leq k \leq N$, we sort in ascending order the packet end-to-end delay values to obtain N new ordered sets $\{d_{k_{sort}}^j\}$, with $1 \leq j \leq N_k$. We set the optimal playout delay of the k -th talkspurt to the following value:

$$D_k = d_{k_{sort}}^i, \quad i \leq N_k, \quad (5)$$

with $i = \text{round}((1-p)N_k)$. Thus, if $d_k^i \leq D_k$, the i -th packet of talkspurt k is played out, otherwise the packet is dropped due to a late arrival.

We present now our moving average predictor. Consider that we have a set of optimal delay values in the past $\{D_k, D_{k-1}, D_{k-2}, \dots\}$, and that we want to predict the value of D_{k+1} . The predicted value of D_{k+1} is denoted by \hat{D}_{k+1} , and is taken as a weighted average of the last M values of the process $\{D_k\}$. Thus,

$$\hat{D}_{k+1} = \sum_{l=1}^M a_l D_{k-l+1}. \quad (6)$$

The coefficients a_l in (6) must be chosen in a way that minimizes the mean square error between \hat{D}_k and D_k , i.e. $\mathbb{E}[(D_k - \hat{D}_k)^2]$. The desired coefficients are the solution of the set of the so-called normal equations [13]:

$$\sum_{m=0}^{M-1} a_{m+1} r_D(m-l) = r_D(l+1), \quad l = 0, 1, \dots, M-1. \quad (7)$$

In (7), $r_D = \mathbb{E}[D_k D_{k+l}]$ is the lag- l autocorrelation function of the process $\{D_k\}$. The exact form of the autocorrelation function is unknown, but it can be estimated using the past values of the process $\{D_k\}$. Suppose we have K values in the past, we can thus write

$$r_D(r) \simeq \frac{1}{K-|r|} \sum_{k=1}^{K-|r|} D_k D_{k+|r|}, \quad r = 0, \pm 1, \pm 2, \dots, \pm(K-1). \quad (8)$$

The set of normal equations (7) can be solved using single matrix-vector operations. For large values of M (say $M > 100$), the well known Levinson-Durbin algorithm may be preferred [13].

M is called the *model's order*. Figure 4 illustrates how M is calculated. For a given packet trace, starting with $M = 1$, we compute all the values of $\{\hat{D}_k\}$ and estimate $\mathbb{E}[(D_k - \hat{D}_k)^2]$. Then, we increase M and we repeat the process. The model's order is taken equal to the lowest value of M preceding an increase in the mean square error. For example, for trace 1 the algorithm chooses M equal to 19, and for trace 3 it chooses M equal to 8. There exist different methods for selecting the model's order e.g. Double Sided t -test, Minimum Description Length, and Final Prediction Error; the reader is referred to [14].

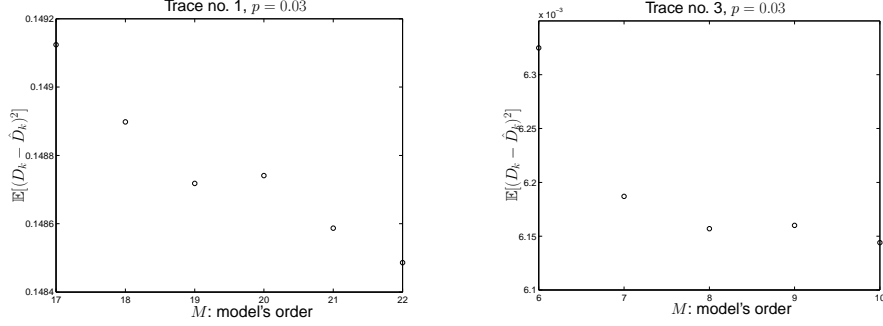


Fig. 4. Model order selection for the MA algorithm.

5.2 The moving average algorithm

We describe now our moving average algorithm for playout delay. The algorithm takes as input a packet delay trace with sender and receiver timestamps, and looks for $\{\hat{D}_k\}$, the estimates of the optimal playout delays $\{D_k\}$. For each past talkspurt, the individual end-to-end delay values are sorted, and \hat{D}_k is computed as in (6). \hat{D}_k is calculated for each talkspurt as a weighted average of the last M talkspurts, for $k = M + 1, M + 2, \dots, N$. Later, when evaluating the average playout delay and the loss percentage per-session, we discard in the computation the first M talkspurts.

Figure 5 depicts a pseudo-code version of the MA algorithm. The `getOptDelay()` function takes as input the whole set of end-to-end delay values, \mathbf{d} , and the desired loss percentage per-session p . Then, it applies (5) to return a set of optimal per-talkspurt delay values $\{D_k\}$. The first `for` loop solves the normal equations for a_l to compute \hat{D}_{k+1} for each talkspurt, then it calculates $\mathbb{E}[(D_k - \hat{D}_k)^2]$ for different values of the model's order m , and holds the result in the vector \overline{mse} . Next, `getOrder(\overline{mse})` is called to find the model's order, M , by choosing the lowest value of M preceding an increase in \overline{mse} . Then, we compute the coefficients a_l with the value of M just found.

The last `for` loop computes \hat{D}_{k+1} for each talkspurt and the playout times p_k^i . The playout time of the i -th packet of talkspurt k is set as follows:

$$p_k^i = \begin{cases} t_k^1 + \hat{D}_k, & \text{for } i = 1 \\ p_k^1 + (t_k^i - t_k^1), & \text{for } 1 < i \leq N_k. \end{cases} \quad (9)$$

The moving average algorithm requires the knowledge of the characteristics of the process $\{D_k\}$. Assuming the process $\{D_k\}$ is stationary during all the audio session, the best performance of this algorithm is obtained when it is run offline or after a large number of samples is collected.

5.3 The problem with low p

As our simulation results will show, the MA algorithm described in Sect. 5.2 deviates from the desired loss percentage p . See in (5) how the value of D_k depends on the talkspurt size N_k , and on the desired loss percentage p . For a given value of p , (5)

```

1.  $D_k \leftarrow \text{getOptDelay}(\mathbf{d}, p);$ 
2.  $R \leftarrow \text{autocorr}(D_k, N);$ 
3.
4. for  $m = 1$  to  $N$  {
5.     /* Get the weights */
6.      $\mathbf{a} = \text{solve}(R, m);$ 
7.     /* Compute  $\hat{D}_{k+1}$  for each talkspurt */
8.      $\hat{D}_{k+1} = \sum_{l=1}^m a_l D_{k-l+1};$ 
9.     /* Update the mse vector for this value of  $m$  */
10.     $\text{mse}(m) = \mathbb{E}[(D_k - \hat{D}_k)^2];$ 
11. }
12.
13.  $M = \text{getOrder}(\overrightarrow{\text{mse}});$ 
14.  $\mathbf{a} = \text{solve}(R, M);$ 
15.
16. for  $k = M$  to  $N - 1$  {
17.      $\hat{D}_{k+1} = \sum_{l=1}^M a_l D_{k-l+1};$ 
18.      $p_{k+1}^1 = t_{k+1}^1 + \hat{D}_{k+1};$ 
19.     20. for  $j = 2$  to  $N_{k+1}$ 
21.          $p_{k+1}^j = p_{k+1}^1 + t_{k+1}^j - t_{k+1}^1;$ 
22. }

```

Fig. 5. The MA algorithm.

returns the delay value closest to $p \times N_k$. Thus, when computing \hat{D}_k , there will be a deviation of the overall perceived loss percentage from the one we desire. The highest deviation is for very low values of p . The algorithm leads to a loss percentage longer than the desired one. To deal with this deviation, for the range $0.005 \leq p \leq 0.02$, we allow our MA algorithm to slightly increase the playout delay by the following offset:

$$\Delta_{\hat{D}_k} = f(p) \sqrt{\mathbb{E}[(\hat{D}_k - D_k)^2]}. \quad (10)$$

In this way, the playout delay is increased as a function of the measured mean square error between D_k and \hat{D}_k , and as a function of p . Since the deviation of the measured loss percentage increases for small values of p , we set f to $f(p) = -\delta \times (\frac{p}{p_{\max}} - 1)$, where δ is a constant controlling how much we increase the playout delay as a function of the square root of $\mathbb{E}[(D_k - \hat{D}_k)^2]$. So, as p increases in the range $(0, p_{\max}]$, \hat{D}_k converges to its original form (6), and if p decreases a longer offset is used. We set $p_{\max} = 0.02$ and $\delta = 0.5$. This allows to reduce considerably the deviation of the measured loss percentage from p , without impacting much the delay.

The following lines must be added to the pseudo-code shown in Fig. 5 between lines 17 and 18:

$$\text{if } p \leq 0.02 \\ \hat{D}_{k+1} \leftarrow \hat{D}_{k+1} + \Delta_{\hat{D}_k}.$$

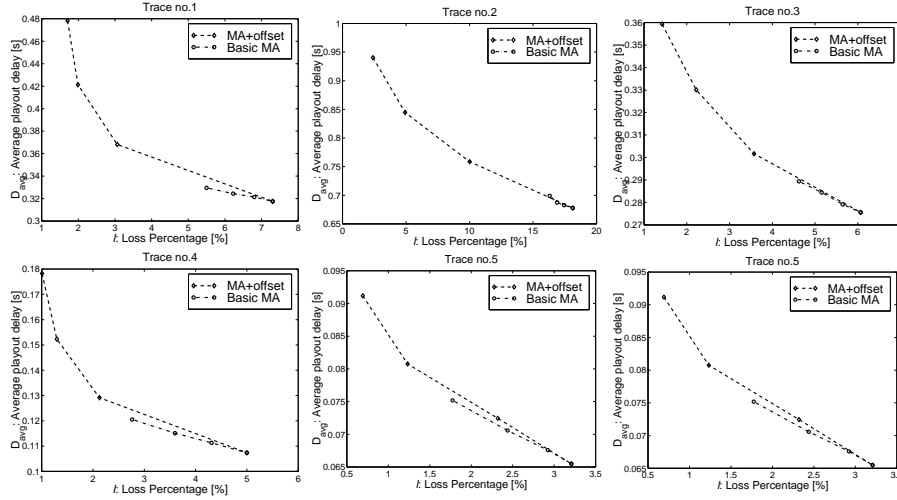


Fig. 6. Performance of the MA algorithm before and after adding the offset to \hat{D}_k for $p \in [0.005, 0.02]$.

To see the gain obtained when applying (10), we plot in Fig. 6, for $p \in [0.005, 0.02]$ the performance of the original MA algorithm before and after this change. The x -axis represents the total measured loss percentage due to late losses, l , and the y -axis plots the average playout delay, D_{avg} , for discrete values of p from 0.005 to $p_{\text{max}} = 0.02$, with p increasing 0.005 at each step. We call this new algorithm *MA+offset*, and we refer to it as such during the rest of the paper.

The deviation of the loss percentage for the *MA+offset* algorithm is much lower, while keeping the average playout delay within reasonable values. The gain we get compared to the basic MA algorithm is very clear. For trace 1, the *MA+offset* algorithm reaches loss percentages of 1.7% compared to 5.4% in the basic MA algorithm. The *MA+offset* algorithm is beneficial for all traces, and the gain is higher for trace 2 and trace 6, where the deviation of the desired loss percentage is now much lower compared to the original case.

Section 5.4 compares the performance of our *MA+offset* algorithm with algorithms *A* and *B*.

5.4 Performance comparison

To evaluate each of the three algorithms, we use a simulator that reads in an input file containing the sender and receiver timestamps of each packet of an audio session. Then, each algorithm is executed, and we use expressions (3) and (4) to compute the average playout delay and the loss percentage during an audio session.

As pointed out above, algorithms *A* and *B* are unable to get a particular target of loss percentage p . Thus, to obtain different loss percentages in Algorithms *A* and *B* we vary β in (1) from 1 to 20; larger values of β allow to get lower loss percentages, at the

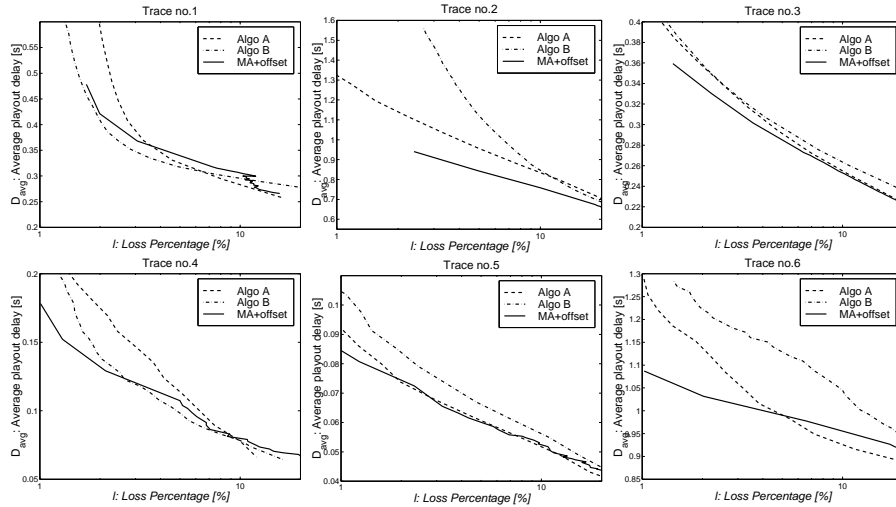


Fig. 7. Performance comparison of algorithms A , B , and the MA+offset algorithm.

expense of a higher average playout delay. We compare the performance of algorithms A and B with our MA+offset algorithm for $0.005 \leq p \leq 0.2$. Loss percentages smaller than 5% are rather desirable for interactive audio applications [3].

Figure 7 plots the corresponding results for each trace. The x -axis represents the perceived loss percentage per-session l , and the y -axis represents the average playout delay D_{avg} . Each execution of an algorithm gives a single point in the graphic. The plots in Fig. 7 were obtained by connecting the discrete points returned by each approach.

In trace 1, we see how for loss percentages greater than 5%, the performance of the three algorithms is quite similar, with algorithms A and B having a slightly better performance than the MA+offset algorithm for loss percentages between 5% and 11%. For loss percentages lower than 5%, the performance of algorithms B and MA+offset remains similar but they outperform algorithm A with a maximum gain on average playout delay of about 40% of the MA+offset algorithm compared to algorithm A . Trace 2 is the only multicast session and has a large network loss percentage of about 50%, it has also long inactivity periods of up to 2 minutes. The MA+offset algorithm clearly outperforms algorithms A and B for the whole range of loss percentage and average playout delay, with a maximum gain on playout delay of about 200% compared to algorithm B . In trace 3, algorithms A and B remain close to the MA+offset algorithm, with the MA+offset algorithm giving better performance for the whole range of loss percentage and average playout delay. For traces 4 to 6, algorithms A and MA+offset remain close in both, loss percentage and average playout delay, outperforming algorithm B . For loss percentages lower than 3%, the MA+offset algorithm performs better than algorithms A and B . This difference in performance is clearly seen in trace 6, where the MA+offset algorithm shows a considerable gain over algorithms A and B .

Deviations of loss percentage persist in the MA+offset algorithm. The highest deviations in Fig. 6 are for traces 2 and 6. Both are the shortest traces, they suffer from

high variations on end-to-end delay, and high network loss due to congestion. Thus, the autocorrelation function does not have useful information about the process $\{D_k\}$, and consequently the estimated values $\{\hat{D}_k\}$ are inaccurate. Section 6 describes a transformation that can be done to reduce the deviations of the MA+offset algorithm.

6 Bias and Transformation

Our scheme is designed with the main objective to control the loss percentage to a certain value p , while minimizing the average playout delay. Here is the strength of our scheme compared to other schemes in the literature, where we do not have a direct control on the loss percentage. Our control on this parameter has been done till now by controlling the optimal playout delay per talkspurt D_k . But, the relationship between the playout delay and the loss percentage may not be linear. This may cause in a deviation of the perceived loss percentage from the desired one. Technically said, our predictor is unbiased with respect to D_k , however it may be biased with respect to the loss percentage per talkspurt. We illustrate this bias by the following analysis.

Our moving average predictor of D_k ensures that $\mathbb{E}[D_k] = \mathbb{E}[\hat{D}_k]$. Let $d_k^i, 1 \leq i \leq N_k$, be the delay of the i -th packet of talkspurt k . The way we define D_k also ensures that $\mathbb{E}[1\{d_k^i > D_k\}] = p$, with $1\{A\}$ being the indicator function. But, the average loss percentage we experience during the audio conversation is not $\mathbb{E}[1\{d_k^i > D_k\}]$, but rather $\mathbb{E}[1\{d_k^i > \hat{D}_k\}]$. We explain next why this experienced average loss percentage can be different from p , when the relationship between loss and delay is non-linear.

Let $F(x)$ be the complementary CDF of packet end-to-end delay, i.e., $F(x) = \mathbb{P}\{d_k^i > x\}$. It is easy to see that for $x = \mathbb{E}[D_k]$, $F(x)$ is equal to p , since $\mathbb{P}\{d_k^i > D_k\} = \mathbb{E}[1\{d_k^i > D_k\}]$ is equal to p by definition.

The average packet loss percentage we obtain with our scheme is equal to $\hat{p} = \mathbb{E}[1\{d_k^i > \hat{D}_k\}]$. We condition on the value of D_k . This leads to

$$\hat{p} = \mathbb{E}[1\{d_k^i > \hat{D}_k\}] = \mathbb{E}\left[\mathbb{E}[1\{d_k^i > \hat{D}_k\} \mid D_k]\right] = \mathbb{E}\left[F(\mathbb{E}[D_k] + \hat{D}_k - D_k)\right].$$

The last equality results from the fact that

$$\mathbb{E}[1\{d_k^i > D_k + y\}] = \mathbb{E}\left[\mathbb{E}[1\{d_k^i > D_k + y\} \mid D_k]\right] = F(\mathbb{E}[D_k] + y). \quad (11)$$

The proof (11) is as follows. The objective is to compute the loss probability of a packet when the playout delay in a talkspurt is set y units far from the optimal playout delay D_k . In other words, we want to compute the loss probability of a packet, when the playout delay is set y units far from the playout delay that results in a packet loss percentage equal to p . But, the playout delay that results in a packet loss percentage equal to p (if we only look at the packet and not at the talkspurt to which the packet belongs) is simply $F(\mathbb{E}[D_k])$. Hence, the problem is equivalent to computing the loss probability of a random packet when the playout delay for this packet is set y units far from $\mathbb{E}[D_k]$, which is equal to $F(\mathbb{E}[D_k] + y)$.

Let ϵ_k be the prediction error for talkspurt k , i.e., $\epsilon_k = \hat{D}_k - D_k$. We write $\hat{p} = \mathbb{E}[F(\mathbb{E}[D_k] + \epsilon_k)]$. The bias of our predictor can be seen from this expression, when the function $F(x)$ is non-linear. $F(x)$ relates the packet loss probability of a packet to the playout delay. For example, if $F(x)$ is a convex function, we have by Jensen's inequality $\hat{p} > F(\mathbb{E}[D_k] + \mathbb{E}[\epsilon_k]) = F(\mathbb{E}[D_k]) = p$.

To correct this bias, some transformation of the process D_k can be used. Define $X_k = G(D_k)$. The prediction will be done on the process X_k instead of D_k , using a Moving Average predictor, i.e., $\hat{X}_{k+1} = \sum_{l=1}^M a_l X_{k-l+1}$. Once the estimate of X_k , denoted by \hat{X}_k is obtained, we set the playout delay to $G^{-1}(\hat{X}_k)$. The average loss percentage becomes equal to $\hat{p} = \mathbb{E}[F(\mathbb{E}[D_k] + G^{-1}(\hat{X}_k) - G^{-1}(X_k))]$.

The function $G(x)$ must compensate for the non-linearity of the function $F(x)$. It must transform the error in setting the playout delay, so as to make \hat{p} equal to p . Unfortunately, it is very difficult to find the expression of $G(x)$. Some approximations can be used. We give an example of a transformation that we use in this paper. Our measurements show that the function $F(x)$ is convex, and close to exponential. We consider then as transformation the exponential function, with a decay coefficient α , that is, we take $G(x) = e^{-\alpha x}$. Hence, we predict $X_k = e^{-\alpha D_k}$ instead of predicting D_k .

6.1 Performance comparison

We apply the exponential transformation $G(x)$ to the process \hat{D}_k in our MA algorithm. The MA algorithm remains the same, but we predict now the process $X_k = e^{-\alpha D_k}$ rather than directly predicting D_k . We call this new algorithm *transformed MA* algorithm.

When testing the transformed MA algorithm we found no significant differences for $10 < \alpha \leq 20$. For $\alpha < 10$ the performance degrades very slowly with decreasing α . We thus set empirically the value of α to 10, and we use it when comparing with algorithms A and B in the next section.

To further improve the performance of the transformed MA algorithm, when transforming back \hat{D}_k from \hat{X}_k , we implement the procedure described in Sec. 5.3 to reduce the deviations for small values of p . We call this variant *transformed MA+offset*.

Figure 8 compares the performance of our two MA+offset algorithms with algorithms A and B . In the subsequent figures, the transformed MA+offset is denoted as MA+transf+offset. Observe how the transformation applied on D_k considerably improves the performance of the MA algorithm. For trace 1, the transformed MA+offset algorithm clearly outperforms algorithms A , B , and the original MA algorithm with a gain of up to 50% for the whole range of loss percentage and average playout delay. For trace 2, the transformed MA+offset algorithm does not reach loss percentages lower than 5%. This is still due to the high jitter and network loss present in trace 2 which does not provide the autocorrelation function with useful information about the process D_k . However, the transformed MA+offset algorithm largely outperforms algorithms A and B for other values of p . For traces 3 to 5, we see clearly the benefit of applying the transformation $G(x)$. The transformed MA+offset algorithm outperforms all the other algorithms with a maximum gain on average playout delay of up to 80% in trace 4

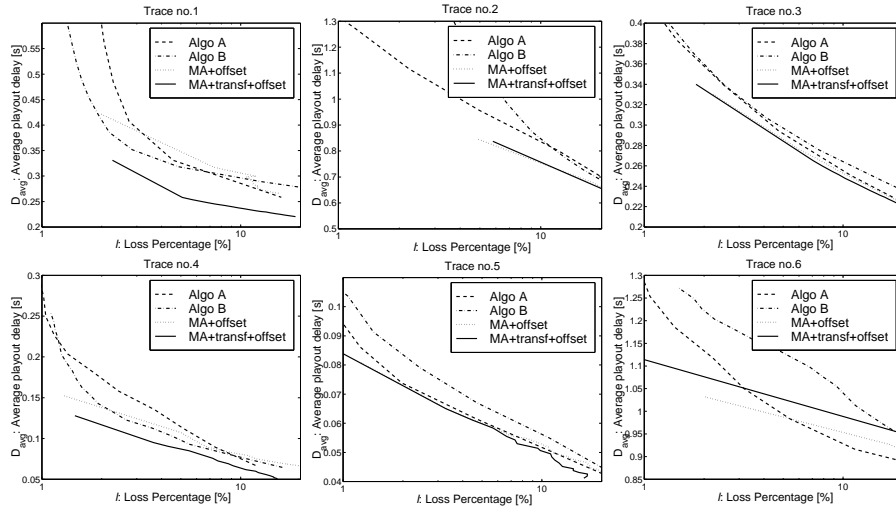


Fig. 8. Performance comparison of algorithms A , B , and the transformed MA+offset algorithm.

for low loss percentages, compared to algorithms A and B . We notice an interesting behavior of the transformed MA algorithm in trace 6. This is the only trace for which the MA+offset algorithm behaves better than the transformed MA+offset version. Like trace 2, trace 6 also has a high end-to-end delay and a high network loss percentage; besides, trace 6 is one of the shortest sessions (in number of talkspurts). Thus, the MA+offset algorithm should be preferred when there are long congestion periods in the network and very high jitter.

We conclude that a moving average scheme is an attractive approach for playout delay control. The algorithms studied till now are offline algorithms. Section 7 presents an online hybrid algorithm combining algorithm B and the transformed MA+offset algorithm which gives a very good performance for most of the scenarios.

7 A hybrid algorithm for playout delay

Moving average estimation has revealed to be an interesting approach for playout delay control. The transformed MA+offset algorithm described in the previous section gives in general better performance than any of the other algorithms we studied. This algorithm was run offline and the entire trace was used to compute the characteristics of $\{D_k\}$. We look now for an online version of the transformed MA+offset algorithm. During our simulations, the maximum model's order was never greater than 23. This means that we do not need a large number of samples to compute a good moving average estimation. We propose in this section a combination of the transformed MA+offset algorithm and algorithm B , that we call *hybrid algorithm*.

The idea is quite simple and is sketched as pseudo-code in Fig. 9. During the first talkspurts of an audio session, say MAXTKSP talkspurts, algorithm B is executed with $\beta = 4$. At the same time, we collect samples of $\{D_k\}$, we transform them to $\{X_k\}$, and

we keep them in memory to be used later to compute the model's order and predictor coefficients. Then, starting from talkspurt $\text{MAXTKSP}+1$, the transformed MA+offset algorithm is executed and playout times are computed. The autocorrelation function is updated at each new talkspurt to account for the MAXTKSP most recent values of X_k . Since finding the model's order, M , is an exhaustive operation its value is computed only once and it is kept during the whole session. MAXTKSP is set equal to 100 for all the traces. The transformation applied to D_k is $X_k = e^{-\alpha D_k}$ and is denoted as $G(D_k)$.

```

1. During the first MAXTKSP talkspurts {
2.   Execute Algo B;
3.   Compute playout times  $p_k^i$ ;
4.   Collect statistics about  $D_k$ ;
5. }
6.  $X_k \leftarrow G(D_k)$ ;
7.  $R \leftarrow \text{autocorr}(X_k)$ ;
8.  $M = \text{findorder}()$ ;
9. /* For each talkspurt from  $k=1$  to MAXTKSP */
10.  $\hat{X}_k = \sum_{l=1}^M a_l X_{k-l+1}$ ;
11.  $\hat{D}_k = G^{-1}(\hat{X}_k)$ ;
12.
13. for  $k=\text{MAXTKSP}$  to  $N-1$  {
14.    $\hat{X}_{k+1} = \sum_{l=1}^M a_l X_{k-l+1}$ ;
15.    $\hat{D}_{k+1} = G^{-1}(\hat{X}_{k+1})$ ;
16.   if  $p \leq 0.02$ 
17.      $\hat{D}_{k+1} \leftarrow \hat{D}_{k+1} + (0.5 - 25p)\sqrt{\mathbb{E}[(D_k - \hat{D}_k)^2]}$ ;
18.
19.    $p_{k+1}^1 = t_{k+1}^1 + \hat{D}_{k+1}$ ;
20.   for  $j = 2$  to  $N_{k+1}$ 
21.      $p_{k+1}^j = p_{k+1}^1 + t_{k+1}^j - t_{k+1}^1$ ;
22.
23.   /* We recompute the autocorrelation function */
24.   /* for the MAXTKSP most recent values of  $X_k$  */
25.    $R \leftarrow \text{autocorr}(X_k)$ ;
26. }
```

Fig. 9. The hybrid online algorithm for playout delay.

7.1 Performance comparison

Figure 10 compares the performance of the hybrid algorithm with algorithms A and B . In trace 1, the hybrid algorithm outperforms algorithms A and B for almost all values of p . We observe an overall gain on playout delay of about 25% of the hybrid algorithm compared to algorithms A and B . We note again in trace 2 how the the hybrid algorithm does not reach loss percentages lower than 5%. In fact, since the number of D_k samples

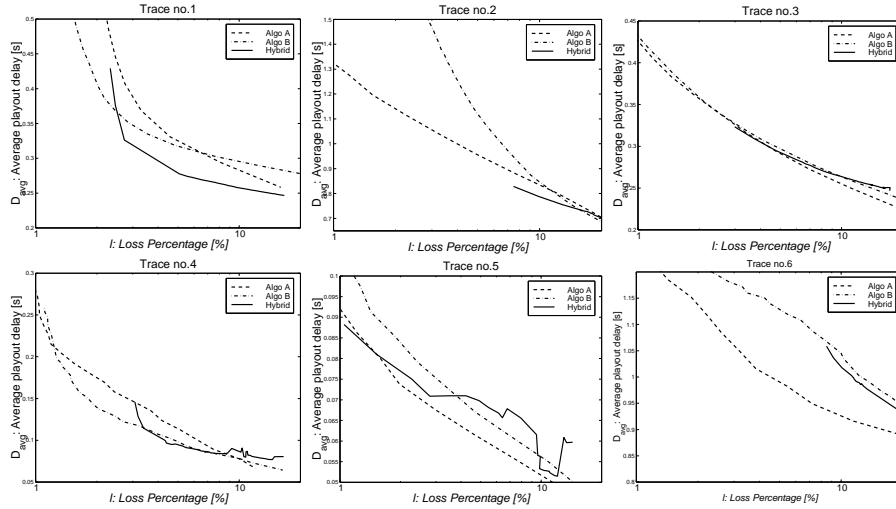


Fig. 10. Performance comparison of algorithms A , B , and the hybrid online algorithm.

used to compute the autocorrelation functions is now small, the error introduced in \hat{X}_k , and consequently in \hat{D}_k , is larger in the hybrid algorithm than in the offline one. For traces 3 to 5, the performance of the hybrid algorithm and algorithms A and B is very similar, with the hybrid algorithm performing better than algorithm A for trace 4, and better than algorithm B for trace 5 in the loss range of interest ($p \leq 0.05$). Trace 6 has the highest session end-to-end delay and high network loss percentages (due to congestion), leading to a behavior similar to that of trace 2.

7.2 Delay spikes

Algorithm B detects delay spikes; when a delay spike occurs, the algorithm switches to spike mode and follows the spike. When the end of the spike is detected, the algorithm switches back to normal mode. We executed the MA+offset algorithm employing the spike detection approach of Algorithm B . When comparing the performance with no spike detection we found no significant differences. The MA+offset algorithm computes the autocorrelation function of the process $\{D_k\}$ to solve the system of normal equations and to calculate $\{\hat{D}_{k+1}\}$. D_k is the optimal per-talkspurt playout delay. When delay spikes occur, the autocorrelation functions of $\{D_k\}$ account for them by definition.

8 Conclusions

This paper describes a moving average algorithm that adaptively adjusts the playout delay at the beginning of talkspurts. To evaluate the performance of our algorithm, we compare it with existing schemes implemented in the NeVoT audio tool. Several variants of our moving average algorithm are studied. For small values of p , there is some

deviation of the perceived loss percentage, and this deviation increases as p decreases. The MA+offset and the transformed MA+offset algorithms are proposed to reduce the deviation of the desired loss percentage. These variants allow to obtain a considerable gain compared to the original version while, at the same time, keeping the average playout delay within tolerable levels.

The strength of our scheme lies in the fact that we are able to tune the loss percentage p to a given desired value. When directly predicting the optimal playout delay, the desired loss percentage deviates from the desired one because the relation between the average playout delay and loss rate is not linear. We demonstrate that, by applying a transformation on D_k , the bias on the loss percentage can be reduced. Based on our measurements, we approximate this transformation by a negative exponential function. A mixture of algorithm B and our transformed MA+offset algorithm proves to be efficient in the loss percentages of interest. We call this algorithm *hybrid algorithm*.

Moving average estimation has revealed to be an efficient method for playout delay control. When network jitter and network loss are very high, as in traces 2 and 6, the MA algorithm do not reach loss percentages lower than 5%. Very high jitter decreases the correlation of the process D_k , leading to an inaccurate MA estimation.

Our algorithm predicts the optimal playout delay per-talkspurt, or a function of it, using the past history of the process. To reconstruct the periodic form of the stream of packets, the playout delay of packet in a talkspurt is based on the playout time of the first packet in the talkspurt. An interesting recent approach [6, 15] shows that it is possible to adapt the playout delay at each packet arrival, leading to a better performance than in a talkspurt basis. We are working on an extension of our MA approach that predicts the playout delay per-packet, allowing to change the playout delay during a talkspurt, and we expect our scheme to give better performance.

Acknowledgments

We thank the CONACyT and the Universidad Autónoma Metropolitana at Mexico for supporting this work.

The authors also thank Sue Moon for making publicly available the traces we used in our simulations.

References

1. Schulzrinne, H.: Voice communication across the Internet: a network voice terminal. Technical report, University of Massachusetts, Amherst (1992)
2. Sasse, A.S., Hardman, V.: Multi-way multicast speech for multimedia conferencing over heterogeneous shared packet networks. RAT-robust audio tool. Technical report, EPSRC Project #GRIK72780 (February)
3. Jayant, N.: Effects of packet loss on waveform coded speech. In: Proceedings of the International Conference on Computer Communications. (1980) 275–280
4. Ramjee, R., Kurose, J., Towsley, D., Schulzrinne, H.: Adaptive playout mechanisms for packetized audio applications in wide-area networks. In: Proceedings of the IEEE Infocom. (1994) 680–688

5. Zhang, Y., Duffield, N., Paxson, V., Shenker, S.: On the constancy of Internet path properties. In: Internet Measurement Workshop (IMW), Marseille, France (2002)
6. Liang, Y.J., Farber, N., Girod, B.: Adaptive playout scheduling and loss concealment for voice communications over IP networks. *IEEE Transactions on Multimedia* (2001)
7. Moon, S.B., Kurose, J., Towsley, D.: Packet audio playout delay adjustment: Performance bounds and algorithms. *ACM/Springer Multimedia Systems* **6** (1998) 17–28
8. Bolot, J.: End-to-end packet delay and loss behavior in the Internet. In: Proceedings of the ACM SIGCOMM. (1993) 289–298
9. Kansar, A., Karandikar, A.: Jitter-free audio playout over best effort packet networks. In: ATM Forum International Symposium, New Delhi, India (2001)
10. Pinto, J., Christensen, K.J.: An algorithm for playout of packet voice based on adaptive adjustment of talkspurt silence periods. In: Proceedings of the IEEE Conference on Local Computer Networks. (1999) 224–231
11. Farber, N., Liang, Y., Girod, B., Prabhakar, B.: Adaptive playout and TCP window control for voice over IP in best-effort networks. Technical report, Stanford University, Information Systems Laboratory, Department of Electrical Engineering (2001)
12. Liu, F., Kim, J.W., Kuo, C.J.: Adaptive delay concealment for Internet voice applications with packet-based time-scale modification. In: Proceedings of the International Conference on Acoustics Speech and Signal Processing ICASSP, Salt Lake City (2001)
13. Proakis, J.G., Manolakis, D.G.: *Digital Signal Processing: Principles, algorithms, and applications*. Prentice-Hall Inc. (1996)
14. Kleinbaum, D.G., Kupper, L.L., Muller, K.E.: *Applied Regression Analysis and Other Multivariable Methods*. PWS-Kent, Boston (1988)
15. Liang, Y.J., Farber, N., Girod, B.: Adaptive playout scheduling using time-scale modification in packet voice communications. In: Proceedings of the International Conference on Acoustics Speech and Signal Processing ICASSP. Volume 3. (2001) 1445–1448