Playout Delay Prediction in VoIP Applications: Linear versus Nonlinear Time Series Models

José B. Araújo Jr.¹ and Guilherme A. Barreto² *

¹, ² - Federal University of Ceará - Department of Teleinformatics Engineering
Av. Mister Hull, S/N - Campus of Pici - Center of Technology
CP 6007, CEP 60455-970, Fortaleza, Ceará, Brazil

Abstract. Voice over IP (VoIP) applications require a buffer at the receiver to minimize the packet loss due to late arrival. Several algorithms are available in the literature whose goal is to predict an optimal playout buffer delay. Classic algorithms differentiate themselves from the novel ones basically due to the lack of learning mechanisms. This paper proposes two new formulations of learning algorithms, the first one is based on the linear autoregressive model, while the second one is based on the MLP network. The obtained results indicate that the proposed algorithms present better overall performance than the classic ones.

1 Introduction

Voice over IP (VoIP) technology is becoming an important paradigm in today’s portfolio of multimedia applications over the internet [1]. This is happening in such a very fast pace that it has drawn wide interest among both research and commercial communities alike. However, the Internet was not originally designed to replace the circuit switched networks that traditionally carry voice traffic over the public switched telephone network. To move through the Internet, user’s continuous speech must be converted to IP packets. As a consequence, the statistical nature of data traffic and the dynamic routing techniques employed in packet-switched networks results in a varying network delay (jitter) experienced by IP packets, which can considerably degrade the quality of the service.

Technically speaking, jitter is the measure of the variability over time of the latency across a network. A widely used solution to alleviate the effects of jitter is to buffer the received audio packets before playing them out in the correct temporal order they were generated [2]. The playout of received audio packets from this buffer is postponed by a certain amount of time, to allow subsequent longer delayed packets to arrive at the receiver ahead of their scheduled playout times. The packets that still do not arrive within their delayed playout schedules are considered lost and are discarded.

The playout delay (or, more accurately, end-to-end application-to-application delay) is defined to be the difference between the playout time at the receiver and the generation time at the sender. If the playout delay in jitter buffer is increased then less packet are lost due to late arrival, but more delay is added to the voice call. A reduction in the playout delay turns out in less delay but more packet loss. The playout delay of the voice packets needs to be continuously

*The authors thanks CAPES for the financial support.
adapted in order to maintain an acceptable compromise between late packet loss and tolerable additional delay over the entire duration of the voice call.

The most commonly implemented solution for playout delay adaptation is suited for use in silence suppressed speech transmission scenarios, where the playout delay is set for individual talkspurts. Using an estimate of the network delay of upcoming voice packets, the playout delay is varied only at the beginning of a new talkspurt resulting in either compression or expansion of silent periods while the temporal structure of packets within a talkspurt is maintained intact.

Standard algorithms for playout delay prediction are based on simple descriptive statistics of the studied phenomenon, such as mean and standard deviation of the end-to-end delay during previous talkspurts. In this paper, we propose a novel formulation for the prediction of the playout delay for individual talkspurts and evaluate it using two time series models. The first one is based on the linear autoregressive (AR) model, while the second one is a nonlinear AR model implemented via an MLP network. The performances of the proposed models are compared with standard playout delay prediction algorithms.

Adapted from [3], the following notation will be used throughout this paper to describe the packet-audio stream. Figure 1 helps understanding the timing information of audio packets in a talkspurt.

- $M$: number of talkspurts in a given trace\(^1\).
- $t^i_k$: sender timestamp of the $i$-th packet in the $k$-th talkspurt.
- $a^i_k$: receiver timestamp of the $i$-th packet in the $k$-th talkspurt.
- $n_k$: number of packets in the $k$-th talkspurt. Here, we only consider those packets actually received at the receiver.
- $d^i_k$: delay between the generation of the $i$-th packet of the $k$-th talkspurt at the sender and its reception at the receiver, namely $d^i_k = a^i_k - t^i_k$.
- $\hat{d}$: smallest network delay in a trace, i.e. $\hat{d} = \min_{1 \leq k \leq M, 1 \leq i \leq n_k} (d^i_k)$.
- $d^i_k$: normalized delay of the $i$-th packet of the $k$-th talkspurt, i.e. $d^i_k = \hat{d} - d^i_k$. This normalization is required to compensate for the asynchrony between the sender and receiver clocks.
- $d^{(i)}_k$: $i$-th smallest normalized delay in the $k$-th talkspurt.
- $v^i_k$: delay variation from the 1st to the $i$-th packet of the $k$-th talkspurt.
- $\hat{e}_k$: estimated excess delay for the $k$-th talkspurt. It is the amount of delay imposed by the buffer to the $k$-th talkspurt.
- $\hat{p}_k$: predicted playout delay for the $k$-th talkspurt. It is the total elapsed time between the emission of the $k$-th talkspurt and its execution at the receiver.
- $p^i_k$: time when the receiver plays out the $i$-th of the $k$-th talkspurt.
- alp: average lost packet rate in a trace.

The remainder of the paper is organized as follows. From Section 2 to 4 we present three classic algorithms for the prediction of the playout delay. In Section 5 we introduce the proposed approach based on AR time series models.

\(^1\)Roughly speaking, a trace is a time series containing the actual network delays experienced by each packet.
The results of the performance evaluation of the classic and proposed algorithms is carried out in Section 6. The paper is concluded in Section 7.

2 Algorithm 1: Optimal Algorithm for a Single Talkspurt

This algorithm is actually a non-causal method to compute the optimal playout delay. It is non-causal because the playout delay is computed after the arrival of all packets of the $k$-th talkspurt. Thus, it is useful only as a reference for comparing the performances of other prediction algorithms.

Algorithm 1 allows the user to assess a posteriori which would have been the playout delay to be applied to the $k$-th talkspurt in order to ensure the loss of $n_k - i$ packets. So, the optimal playout delay is computed as

$$\hat{p}_k = \hat{d}_k^{(i)} \quad \text{where} \quad i = n_k(1 - \alpha p).$$

Once the value of $\hat{p}_k$ is computed, it is used to estimate the value of $\hat{e}_k$ as follows

$$\hat{e}_k = \hat{p}_k - d_{k}^{f},$$

where $d_{k}^{f}$ is the network delay of the 1st packet of the $k$-th talkspurt to arrive at the receiver. Then, the $\hat{e}_k$ is used as the excess delay for the $(k+1)$-th talkspurt.

3 Algorithm 2: Temporal Smoothing of Network Delays

This algorithm, proposed by [4], predicts the playout delay of the $k$-th talkspurt (i.e. $\hat{p}_k$) through the linear combination of the estimated values of the network delays of all packets in a trace and their corresponding estimated standard deviations. First, the network delays are estimated by the following recursive equations:

$$\hat{d}_k^{(i)} = \alpha \hat{d}_k^{(i-1)} + (1 - \alpha) d_k^{(i)}, \quad 2 \leq i \leq n_k, 1 \leq k \leq M,$$
and
\[ \hat{d}^1_k = \alpha \hat{d}^n_k + (1 - \alpha) d^1_k, \quad 1 \leq k \leq M, \] (4)

where \( \hat{d}^1_k \) is the estimated network delay of the \( i \)-th packet of the \( k \)-th talkspurt, \( \hat{d}^n_k \) is the estimated delay for the last packet of the \( k \)-th talkspurt and \( \alpha = 0.998002 \) is a constant weight responsible for the exponentially decaying memory of the algorithm. Second, the estimated delay variation for the \( i \)-th packet of the \( k \)-th talkspurt is computed as
\[ \hat{v}^i_k = \alpha \hat{v}^{i-1}_k + (1 - \alpha) |\hat{d}^i_k - d^i_k|, \quad 1 \leq i \leq n_k, 1 \leq k \leq M. \] (5)

Finally, the predicted playout delay for the \( k \)-th talkspurt is given by
\[ \hat{p}_k = \hat{d}^n_k - 1 + \beta \hat{v}^{n-1}_k, \quad 1 \leq k \leq M, \] (6)
where the constant \( \beta \) is usually set to 4. Eq. (2) is also used to compute the excess delay \( \hat{e}_k \).

4 Algorithm 3: Histogram-based Method

This algorithm was proposed by [5]. It functions by storing the network delays of \( w \) packets and building a histogram from them (Figure 2). In this paper we use \( w = 150 \). Once defined an acceptable \( alp \) (e.g. 0.05) for the problem, \( \hat{p}_k \) is computed as the 100(1 - \( alp \))-th percentile\(^2\) of the packet delay distribution. Equation (2) is also used to compute the excess delay \( \hat{e}_k \).

---

\(^2\)The percentile of a distribution is a number \( z \) such that a percentage \( p \) of the population values are less than or equal to \( z \). For example, the 75th percentile is a value \( (z) \) such that 75% of the values of the variable fall below that value.
5 Prediction via Times Series Models

A common feature of the algorithms to be described in this section is the use of learning or adaptive strategies for predicting the playout delay. By learning we roughly mean the ability of an algorithm to change its parameters according to the network conditions so that a better estimation of the playout delay is expected to be provided.

5.1 Algorithm 4: Proposed Model 1

It is commonly assumed that the \( \hat{p}_k \) can be decomposed as

\[
\hat{p}_k = J \mu(d_k) + L \sigma(d_k),
\]  

(7)

where \( J \) and \( L \) are, respectively, the weights associated with the mean \( (\mu(d_k)) \) and standard deviation \( (\sigma(d_k)) \) of the network delays for the \( k \)-th talkspurt. The values of \( \mu(d_k) \) and \( \sigma(d_k) \) are computed over fixed-length blocks of packets.

Let us further assume that the dynamics of \( \hat{p}_k \) can be modeled by a linear autoregressive (AR) model of order \( n \). Thus, we have

\[
\hat{p}_k = \theta_1 \hat{p}_{k-1} + \theta_2 \hat{p}_{k-2} + \cdots + \theta_n \hat{p}_{k-n},
\]  

(8)

where \( n \) denotes the length of the sliding window that includes the \( n \) most recent talkspurts previous to the current one.

If we substitute the definition in Equation (7) into Equation (8) we can write

\[
\hat{p}_k = \theta_1 \mu(d_{k-1}) + \theta_2 \mu(d_{k-2}) + \cdots + \theta_n \mu(d_{k-n}) + \theta_1 \sigma(d_{k-1}) + \theta_2 \sigma(d_{k-2}) + \cdots + \theta_n \sigma(d_{k-n}).
\]  

(9)

Since \( \theta_i J_i \) and \( \theta_i L_i \), \( 1 \leq i \leq n \), are also constants, we can rewrite Equation (9) as

\[
\hat{p}_k = \theta_1^\mu \mu(d_{k-1}) + \theta_1^\sigma \sigma(d_{k-1}) + \theta_2^\mu \mu(d_{k-2}) + \theta_2^\sigma \sigma(d_{k-2}) + \cdots + \theta_n^\mu \mu(d_{k-n}) + \theta_n^\sigma \sigma(d_{k-n}).
\]  

(10)

We use the standard least-squares (LS) method for estimating the parameters of the model in Eq. (10). For this, considering a trace with \( M \) talkspurts and a sliding window of length \( n \), we can write down Eq. (10) in the matrix form as follows

\[
p = X \theta,
\]  

(11)

where

\[
X = \begin{bmatrix}
\mu(d_n) & \sigma(d_n) & \cdots & \mu(d_1) & \sigma(d_1) \\
\mu(d_{n+1}) & \sigma(d_{n+1}) & \cdots & \mu(d_2) & \sigma(d_2) \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\mu(d_{M-2}) & \sigma(d_{M-2}) & \cdots & \mu(d_{M-n-1}) & \sigma(d_{M-n-1}) \\
\mu(d_{M-1}) & \sigma(d_{M-1}) & \cdots & \mu(d_{M-n}) & \sigma(d_{M-n})
\end{bmatrix},
\]  

(12)
and

\[ \mathbf{\theta} = [\theta_1^\mu \theta_1^\sigma \ldots \theta_n^\mu \theta_n^\sigma]^T \quad \text{and} \quad \mathbf{p} = [p_{n+1} p_{n+2} \ldots p_{M-1} p_M]^T, \]  

(13)

where the superscript \( T \) denotes the transpose vector. The LS estimate of \( \mathbf{\theta} \) is then given by

\[ \hat{\mathbf{\theta}} = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{p}. \]  

(14)

For predicting \( \hat{p}_k \) we insert the estimated parameter vector \( \hat{\mathbf{\theta}} \) into Eq. (10) and run this equation. As before, Equation (2) is used to compute the excess delay \( \hat{e}_k \). In this paper we set \( n = 2 \) for this algorithm.

### 5.2 Algorithm 5: Proposed Model 2

In this algorithm we assume that dynamics of the playout delay is modeled by a nonlinear AR (NAR) model of order \( n \). Thus, we have

\[ \hat{p}_k = F(\hat{p}_{k-1}, \hat{p}_{k-2}, \ldots, \hat{p}_{k-n}), \]  

(15)

where \( F : \mathbb{R}^n \rightarrow \mathbb{R} \) is a nonlinear mapping. In this paper we use the multi-layer Perceptron (MLP) network to approximate the mapping \( F(\cdot) \). Thus, using the MLP and substituting Eq. (7) into the right-hand side of Eq. (15), it is straightforward to show that this equation can be re-written as

\[ \hat{p}_k = G(\mu(d_{k-1}), \sigma(d_{k-1}), \mu(d_{k-2}), \sigma(d_{k-2}), \ldots, \mu(d_{k-n}), \sigma(d_{k-n})), \]  

(16)

where \( G : \mathbb{R}^{2n} \rightarrow \mathbb{R} \) is also an unknown nonlinear mapping. We approximate the nonlinear mapping \( G(\cdot) \) through an MLP network with \( 2n+1 \) inputs (including bias), one hidden layer with \( Q \) neurons, and one output neuron (see Figure 3). The hidden and output neurons use hyperbolic tangent activation functions. Weights and biases are randomly initialized in the range \([-0.5, 0.5]\) and adjusted through the standard gradient descent backpropagation algorithm with learning rate set to 0.05. For this algorithm, the memory order is set to \( n = 5 \). As shown in Figure 3, the output of the MLP provides an estimate of the playout delay for the \( k \)-th talkspurt, i.e. \( O^{mlp}(k) = \hat{p}_k \). As always, Eq. (2) is used to compute the excess delay \( \hat{e}_k \).

It is clear that compared with Algorithm 4, which is linear, Algorithm 5 is nonlinear. Besides this major difference between them, there are more two important ones. First, Algorithm 4 is trained in batch mode, while Algorithm 5 is trained in a pattern-by-pattern mode. Second, there is no training phase for Algorithm 5 in the usual neural network sense. This algorithm is used as an adaptive filter, i.e. there is no freezing of the weights and biases after a training period. In other words, once the MLP network is initialized, its output starts to predict the playout delay and its weights (and biases) are allowed to change in response to every input vector.

The proposed MLP-based NAR model for predicting the playout delay is similar to the neural network algorithm proposed in [6], differing from it basically in the definition of the neural network output. While the former requires only
Fig. 3: MLP network topology used for predicting the playout delay.

one output since it predicts \( \hat{p}_k \) directly, the latter requires two outputs, one for predicting the network delay (\( \mu(d_k) \)) for the \( k \)-th talkspurt and other for predicting its standard deviation (\( \sigma(d_k) \)). These outputs are then combined as in Eq.(7) to predict \( \hat{p}_k \).

5.3 Algorithms 6 and 7: Elman and Jordan Recurrent Networks

Algorithms 6 and 7 are similar to Algorithm 5. The main differences among them are concerned with the input vector that each one processes. More specifically, for the \( k \)-th talkspurt, the input vector of the Algorithm 5 is defined as

\[
\mathbf{x}(k) = [x_0(k) \ x_1(k) \ x_2(k) \ \cdots \ x_{2n-1}(k) \ x_{2n}(k)]^T
\]

(17)

Hence, the corresponding input vectors for the Elman and Jordan recurrent network are defined, respectively, as follows

\[
\mathbf{x}^e(k) = [\mathbf{x}(k) \ | \ y_1(k-1) \ y_2(k-1) \ \cdots \ y_Q(k-1)]^T
\]

(18)

\[
\mathbf{x}^j(k) = [\mathbf{x}(k) \ | \ x^c(k)]^T
\]

(19)

where \( y_i(k-1), i = 1, 2, \ldots, Q \), are the previous outputs of the hidden neurons of the Elman network, and \( x^c(k) = \gamma x^c(k-1) + O_{\text{Jordan}}(k-1) \) is the context unit of the Jordan network, with \( 0 < \gamma < 1 \) as the memory parameter and \( O_{\text{Jordan}}(k-1) \) as the previous output of the Jordan network. The Elman and Jordan recurrent networks are implemented with the same topology and training parameters as the MLP network of Algorithm 5.

6 Simulation Results

The performance evaluation of the seven algorithms previously described is carried out using the same six traces used by [3]. Their summary statistics, shown
Table 1: Summary statistics of the studied traces.

<table>
<thead>
<tr>
<th>Trace</th>
<th>Network delay (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

in Table 1, indicate the presence of delay spikes in the time series, making the prediction task very challenging. Two metrics are used to quantify the performance of the algorithms, namely: (i) the average percentage of lost packets \((alp)\) and (ii) the average total end-to-end delay \((ted)\). They are chosen because the computation of the optimal playout delay is basically a trade-off between them. A low ted for a voice audio stream is desirable to the end user. However, a lower ted typically results in more lost packets due to late arrival. A decrease in the ted, therefore, typically causes an increase in the number of average lost packets. By the same token, a low alp is desirable to the end user since the conference quality of service may be affected if the vocoder cannot compensate accordingly. To obtain a lower alp, however, there is usually an increase in the ted.

In the computation of the alp we considered only the packets lost due to late arrival. We ignore packets dropped by the network due to congestion at routers and assume that they are compensated for by the vocoder at the receiver. As pointed out previously, the Algorithm 1 is used only as a base line for performance comparison since it can not be used in practice (real-time prediction). The results for this algorithm were obtained for a pre-specified alp of 5\% (i.e. \(alp=0.05\)). For all the evaluated neural network models, the number of hidden neurons is set to \(Q = 5\). No normalization of the input data is carried out, since it is an on-line application. The memory orders for the Algorithms 4 and 5 are set to \(n = 2\) and \(n = 5\), respectively. Algorithms 6 and 7 (recurrent networks) use the same memory order of the Algorithm 5 (MLP).

The results for all the algorithms and all the traces are shown in Table 2. Some interesting conclusions can be drawn from this table. First, the best performance in average for the six traces is achieved by the Algorithm 4. Second, despite the fact that the neural network models (Algorithms 5, 6 and 7) did not achieved very good alp values when compared to the other algorithms, if we consider the compromise between a low alp and a low ted the neural network results are very good ones. Finally, considering only the Algorithms 5, 6 and 7, their results are very similar in terms of alp and ted, so there is no special advantage in using recurrent networks for this problem.

To emphasize the importance of jointly minimizing the alp and the ted, we build a table with two different performance rankings (see Table 3), averaged for the six traces. The first ranking considers only the performance evaluation
Table 2: Results for the six traces in terms of the TED and ALP metrics.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Trace 1</th>
<th>Trace 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ted (%)</td>
<td>alp (%)</td>
</tr>
<tr>
<td>1</td>
<td>333.01</td>
<td>5.00</td>
</tr>
<tr>
<td>2</td>
<td>519.46</td>
<td>2.06</td>
</tr>
<tr>
<td>3</td>
<td>406.04</td>
<td>6.61</td>
</tr>
<tr>
<td>4</td>
<td>328.03</td>
<td>1.81</td>
</tr>
<tr>
<td>5</td>
<td>362.52</td>
<td>2.64</td>
</tr>
<tr>
<td>6</td>
<td>362.13</td>
<td>2.74</td>
</tr>
<tr>
<td>7</td>
<td>363.72</td>
<td>2.60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Trace 3</th>
<th>Trace 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ted (%)</td>
<td>alp (%)</td>
</tr>
<tr>
<td>1</td>
<td>259.76</td>
<td>5.00</td>
</tr>
<tr>
<td>2</td>
<td>361.23</td>
<td>1.61</td>
</tr>
<tr>
<td>3</td>
<td>278.43</td>
<td>8.07</td>
</tr>
<tr>
<td>4</td>
<td>254.14</td>
<td>2.86</td>
</tr>
<tr>
<td>5</td>
<td>294.33</td>
<td>4.42</td>
</tr>
<tr>
<td>6</td>
<td>294.51</td>
<td>4.46</td>
</tr>
<tr>
<td>7</td>
<td>295.05</td>
<td>4.45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Trace 5</th>
<th>Trace 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ted (%)</td>
<td>alp (%)</td>
</tr>
<tr>
<td>1</td>
<td>48.63</td>
<td>5.00</td>
</tr>
<tr>
<td>2</td>
<td>80.17</td>
<td>1.47</td>
</tr>
<tr>
<td>3</td>
<td>57.14</td>
<td>3.37</td>
</tr>
<tr>
<td>4</td>
<td>77.53</td>
<td>0.92</td>
</tr>
<tr>
<td>5</td>
<td>78.35</td>
<td>0.99</td>
</tr>
<tr>
<td>6</td>
<td>78.32</td>
<td>0.99</td>
</tr>
<tr>
<td>7</td>
<td>78.93</td>
<td>0.94</td>
</tr>
</tbody>
</table>

in terms of alp values. In this case, the Algorithm 2 is the best one. The second ranking considers the performance evaluation in terms of both alp and ted values. In this case, the best performance is achieved by the Algorithm 4, followed closely by the Algorithms 5, 6 and 7.

The good performance of the Algorithm 4 can be explained by the fact that it is trained in batch mode, while all the neural network models are trained as adaptive (online) filters. We also trained the MLP in batch mode, but the obtained results were inferior to those produced by the online-trained MLP. The Algorithm 4 can also be trained on-line through the LMS learning rule. We tested this alternative, but the results were very poor, confirming the results

Table 3: Ranking of the performance of the studied algorithms.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>alp</td>
<td>4 2 5 6 7 3</td>
</tr>
<tr>
<td>alp+ted</td>
<td>4 5 6 7 3 2</td>
</tr>
</tbody>
</table>
of previous studies (e.g., see [7]). The online training mode seems to work for the MLP due to the presence of the derivative of the sigmoidal activation function in the generalized Delta rule used to update the weights. This derivative makes the adaptive filter less sensitive to sudden changes in the input signals as commonly occurs in VoIP applications in the form of delay spikes. Finally, it is worth mentioning a potential drawback of the Algorithm 4. The matrix inversion required by Eq. (14) can lead to numerical problems. In practice this can be solved through the use of Tikhonov regularization [8, ch.9]. This approach requires a regularization parameter which can be determined by cross-validation.

7 Conclusions

We introduced two novel approaches for the prediction of the playout delay for individual talkspurts. The first one was based on the linear AR model, while the second one was a nonlinear AR model implemented via an MLP network. The obtained results indicated that the proposed algorithms present better overall performance than the classic nonadaptive ones. We are currently trying to improve the performance of the Algorithm 5 (MLP) when trained in batch-mode. Other learning algorithms, such as Hidden Markov Models [9] and Support-Vector Machine [10], are also being evaluated.

References