Flexible Playout Adaptation for Low Delay AAC RTP Communication

Jochen Issing*, Stefan Reuschl†, Falko Dressler*, Nikolaus Färber†
*Dept. Computer Science 7, Computer Networks and Communication Systems, Friedrich-Alexander-University, Erlangen
†Multimedia Transport Group, Fraunhofer Institute for Integrated Circuits, Erlangen

***Abstract*—We present an integrated approach for flexible playout adaptation for high quality audio transmission over impaired network connections. The key concept of our framework is a continuous measurement of the transmission delay, the delay variation, and packet loss. Based on these measurements, the adaptive playout control employs audio time stretching using audio concealment and frame dropping techniques to keep the low delay requirements. In the literature, playout adaptation techniques have mainly been considered for voice over IP, using silence periods between talkspurts, or for high quality audio transmission over dedicated network links. To the best of our knowledge, our playout algorithm is the first achieving low delay high quality audio streaming over impaired network connections for both music and speech. We used a significant number of network traces to estimate the variation of the network quality in DSL, WLAN, UMTS and GPRS links and to update the parameters of our playout adaptation technique. Experimental results clearly indicate that our system provides very high accuracy for the desired accepted late loss rate and achieves a fast playout adaptation, even for rapidly changing network conditions.**

***Index Terms*—RTP, Playout Adaptation, Low Delay, Advanced Audio Coding, Audio Communication Engine**

I. Introduction

Starting with the migration of the public switched telephone network (PSTN) to voice over IP (VoIP), the audio bandwidth has been continuously increased. While PSTN is limited to narrow band (300 Hz to 3400 Hz) speech codecs, voice over IP is dominated by wide band (30 Hz to 7000 Hz) audio codecs like G.722.2 [1]. Full band (22 kHz) audio codecs find application in enterprise level communication systems and are on their way to be integrated into standard VoIP clients as well. With full band audio communication, the conversation is not only more immediate, but the participants are able to include signals like music or environmental sounds during the conversation as well.

While many professional conference systems require a dedicated network link, consumer oriented or mobile systems often have to compensate network impairments like delay jitter, bandwidth limitation and packet loss. To maintain low delay during the whole conversation, several playout adaptation schemes have been developed for VoIP (see Section II). For high quality audio communication, however, these schemes are either too aggressive or are based on adaptation during silence periods, which do not appear in music in general.

In this paper, we therefore present a new flexible playout adaptation, which is applicable for continuous and high quality audio communication and can be parameterized by window size and accepted late loss.

The Audio Communication Engine (ACE) [2], a low delay audio communication system from Fraunhofer IIS with rate adaptation support [3], is used as the basis for our implementations. It supports flexible playout adaptation using AAC-ELD (Advanced Audio Coding-Enhanced Low Delay) [4][5], not only to maintain low bit rate, low delay, and quality robustness. The ACE also exploits the structure of AAC and its excellent error concealment to provide fast and efficient playout adaptation. ACE playout adaptation and flexible playout adaptation are used as synonyms and both refer to the playout adaptation techniques introduced in this paper.

The paper is structured as follows: existing playout adaption schemes are introduced and discussed in section II. Section III provides some general information about the network trace files, which are used to evaluate the introduced mechanisms. The AAC fundamentals with importance for playout adaptation are presented in section IV and the flexible playout adaptation techniques are described and evaluated in detail in section V.

II. Related Work

First promising playout adaptation techniques have been introduced in 1994 [6]. This paper compares four different algorithms of playout adaptation for speech communication. The spike-like structure of packet delays in the Internet has already been discovered. As shown by measurements [7], this structure can still be found. Three of the four described Algorithms are based on weighted moving averages, similar to the one defined in the RTP specification [8], but with different tunings towards increasing/decreasing delays and spike detection. The other algorithm has been taken from a tool called NeVoT [9] and estimates the delay by calculating the minimum delay of the last talk spurt.

A follow-up study by Moon et al. [10] proposed three algorithms for playout adaptation, of which two are based on linear filters as well. One algorithm follows a different approach for delay jitter estimation: When a talk spurt starts, the algorithm calculates a percentile point in a distribution function (the index in a preallocated array) for the last w packets, where w denotes the number of packets in a sliding time window. The algorithm detects spikes and proceeds as follows: once a spike is detected, it stops collecting packet delays and follows the spike until it detects the end of a spike.

978-1-4244-8953-4/11/$26.00 © 2011 IEEE

Another approach for jitter estimation has been proposed in [2]. It calculates the mean and variance of packet delay over a sliding window and estimates the network jitter using the variance and an empirically chosen factor to calculate the confidence interval.

Scaling audio and speech signals without modifying the pitch is always a difficult matter and, therefore, most of the playout adaptation algorithms schedule only the first frame of a talk spurt and maintain fixed playout delay until the end of the talk spurt. A study by Liang et al. [11] proposed audio scaling based on a tailored WSOLA (Waveform Similarity Overlap-Add) algorithm, which searches for a similar segment using a template audio frame in the time domain. If a similar frame is found, the two audio frames are cross-faded without algorithmic delay by a symmetric window.

In contrast to voice communication, for which most of the previously mentioned methods have been created, music signals are much more continuous in nature than speech. While talk spurts and silence periods can be used to schedule the playout of each talk spurt individually for speech, scaling music signals is much more critical and must be accomplished with caution to limit time stretching artifacts. To support continuous audio signals, the flexible playout adaptation therefore uses a redesigned time stretching technique, which exploits the structure of AAC and is described in detail in the following sections.

III. Network Traces

To evaluate the mechanisms introduced in this paper towards stability and functionality as well as to compare them to existing techniques, a set of more than 500 network trace files has been collected. The traces cover different network scenarios (e.g. WLAN over DSL, GPRS, UMTS, LAN) and have been recorded using a custom network tracer.

The nettracer simulates real RTP/UDP streams using pseudo audio payloads with random data of the specific size and framing interval, corresponding to AAC-ELD with $128\,{}^{\mathrm{kbit}}/_{\mathrm{s}}$ and to HE-AAC(v2) with $24\,{}^{\mathrm{kbit}}/_{\mathrm{s}}$ at $4800\,\mathrm{Hz}$. Both codecs are configured to use constant bit-rate (CBR) audio streams. The constant rate coder is regarded as the default audio coder [12] and assures low delay over e.g. fixed rate channels and MPLS connections.

For AAC-ELD, a frame length of 512 samples is used, which results in a framing interval of $10.7\,\mathrm{ms}$ and a payload size of $171\,\mathrm{B}$. HE-AAC(v2) is simulated with a frame length of 2048 samples, a framing interval of $42.7\,\mathrm{ms}$, and a payload size of $149\,\mathrm{B}$.

The nettracer client establishes an RTP/UDP connection to the nettracer server, which is located at Fraunhofer IIS. The server collects all relevant data, e.g. the network route, network configuration, round trip time (rtt), etc. into one trace file per configuration run. The client starts with the first codec configuration (e.g. AAC-ELD) and switches to the next configuration after $120\,\mathrm{s}$. This process is repeated until the user cancels the application. In the resulting trace file, the sequence number, RTP timestamp and arrival time is recorded for each RTP packet. The measurements have been conducted by colleagues from Fraunhofer IIS from different locations including both stationary and mobile network connections. The set of traces comprises very lossy (about $40\,\%$) as well as loss-free channels with different characteristics towards jitter and available bit-rate. All presented techniques have been analyzed with other network simulators and have been tested in real network scenarios in which they have shown similar behavior, as with the network trace files.

IV. AAC Fundamentals

In the following, we briefly introduce AAC fundamentals of importance to playout adaptation. We first describe the internal structure of AAC and then continue with time scaling techniques. The fundamentals are however not only specific to AAC, but to transform-based audio codecs in general. Thus, if the codec uses overlap-add and the audio codec's concealment provides comparable audio quality as the concealment technique described here, the flexible playout adaptation can be applied as is.

A. AAC Framing

AAC is, like many audio codecs, a transform-based codec [13][14][15]. To cancel aliasing, each frame of audio samples (access unit) is overlapped partly by each of its neighbors using a window function, e.g. a sine window. Hence, AAC provides an implicit cross fade between adjacent access units as shown in Figure 1.

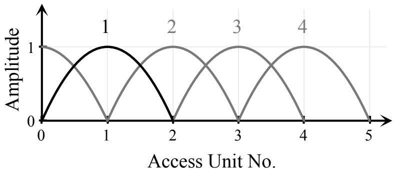

Fig. 1. Overlap add of adjacent access units in AAC

This implicit cross fade permits arbitrary access unit concatenation without clippings at the cost of aliasing artifacts. These artifacts, however, are not critical in general if the dropping rate is kept low.

B. AAC Concealment

For audio stretching, the ACE exploits the excellent concealment technique of AAC. Figure 2 shows a simplified example of how AAC decoders can handle audio concealment. In this example, the AAC decoder duplicates the spectrum of the last audio frame shown in graph (a). The audio spectrum is flattened afterwards to suppress the tonal parts of the signal as shown in graph (b), and attenuated to fade out for longer concealment periods as shown in graph (c). The phase of the audio signal is additionally randomized in its algebraic sign to make the concealed frame more noisy. Thus, AAC concealment could be addressed as shaped noise concealment with attenuation. It supports low delay, because the audio frame can be concealed immediately, and requires low additional complexity compared to other concealment methods, e.g. as introduced in [16].

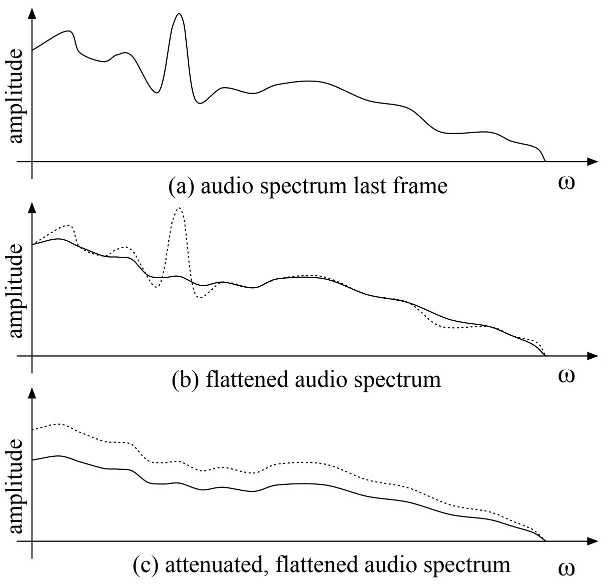

Fig. 2. AAC concealment example

V. ACE Playout Adaptation

Common mechanisms for playout adaptation are based on the a priori playout scheduling of each audio frame. Our flexible playout adaptation, however, increases playout delay implicitly where late loss occurs and becomes active only when the playout delay may be reduced according to the jitter estimation result. The playout adaptation combines separate techniques, starting with the estimation of packet delay jitter, which is introduced in section V-A. The estimation result is then used to control the buffer size. The buffer is stretched and shrunken using mechanisms introduced in V-B. The adaptation process is then further improved using loss to drop conversion in section V-C and surplus dependent dropping in section V-D.

A. ACE Jitter Estimation

The ACE observes the incoming packet delay jitter continuously, to determine the maximum amount of packets to be buffered, which are necessary to compensate the current network jitter.

1) Absolute Delay vs. Interarrival Delay: To describe absolute and interarrival delay, the following quantities are defined:

- s_i: send time of packet with index i, which is conveyed from sender to receiver using the RTP timestamp.
- r_i: the reception time of packet with index i, measured on the receiver side, e.g. using the system clock.
- p_i: playout time of packet with index i, e.g. the receiver's system time at playout.

As shown in Figure 3, each send time s_i represents the offset from a reference send time s_0. The reference receive time r_i represents the elapsed time since r_0. Thus, the absolute values of s_i and r_i depend on the reference times. The delay variation, however, which is used for our jitter estimation does not depend on the reference times.

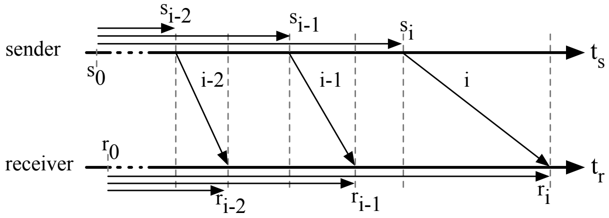

Fig. 3. Absolute packet delay example

The absolute delay is calculated as

$$d_i = r_i - s_i. \tag{1}$$

In contrast to the absolute delay, the interarrival delay is based on the send and receive intervals from packet i to packet $i+1$ on both the sender and the receiver side. Figure 4 shows the sender time intervals Δs_i and Δr_i. Using (1), the time intervals can be expressed as

$$\Delta s_i = s_i - s_{i-1} \tag{2}$$

$$\Delta r_i = r_i - r_{i-1} \tag{3}$$

and the interarrival delay as

$$d'_i = \Delta r_i - \Delta s_i \; . \tag{4}$$

The absolute delay can be derived from the interarrival delay, as shown in Figure 4 ($\Delta s_i + d_i = d_{i-1} + \Delta r_i$):

$$d_i = \Delta r_i - \Delta s_i + d_{i-1}. \tag{5}$$

Note that d_0 must be chosen in advance. It reflects the delay offset of the first packet and is propagated through all subsequent delay measurements. As the jitter estimation assesses the delay variation only and neglects any arbitrary offset, d_0 can be set to 0 for simplicity.

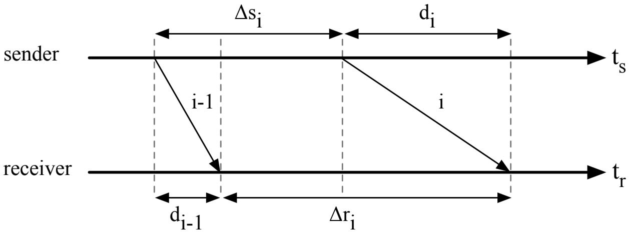

Fig. 4. Absolute delay vs interarrival delay

2) Jitter Estimation Algorithm: Using Equation (5), the absolute delay for each packet is calculated recursively. The network jitter is then expressed by the dispersion of the packet delays. Starting with delay values of a network trace, as recorded using a custom network tracer, the absolute delay is calculated using Equation (5) for each packet. The set of delay values D and the normalized delay values $\hat{D}$ are further described as

$$\begin{aligned} D &= \{d_i\}_{i=0}^{M} & (6)\\ \hat{D} &= \left\{\hat{d}_i\right\}_{i=0}^{M} & (7)\\ \hat{d}_i &= d_i - \min(D) & (8)\end{aligned}$$

where M is the total number of packets of the trace.

Some of our traces show heavy clock drift, because they are recorded on different machines over the network. The virtual playout adaptation of the percentile based method, as described later, is error prone to such clock drift. Thus, before further processing, the clock drift is compensated as described below. Network packet delay is randomly distributed with a common minimum baseline in general. The slope of the baseline indicates the clock drift. To pick well matching anchors for the linear regression, each trace is split into chunks using the following equations:

$$\begin{aligned} P &= \left\lfloor \frac{M}{C} \right\rfloor & (9)\\ v &= \{0, 1, \dots P\} & (10)\\ c_v &= \left\{\hat{d}_j\right\}_{j=vC}^{(v+1)C} & (11)\end{aligned}$$

where P is the number of chunks of size C and c_v is the chunk of index v. A chunk size of 1000 is used throughout the paper, which covers around 11 seconds of audio. The minimum delay value of each chunk is then used as an anchor point and is calculated as

$$c'_v = \min_{0 \leq j < C} (\hat{d}_j^v) \tag{12}$$

where $\hat{d}_j^v$ denotes all normalized delay values in chunk c_v. The clock drift of the trace is now detected by linear regression using all minimum delays c'_v. The prototype function for linear regression is represented by

$$f(x) = a \cdot x + b. \tag{13}$$

The factor a is estimated as $\hat{a}$ and the offset b estimated as $\hat{b}$ using standard regression:

$$\begin{aligned} \hat{a} &= \frac{\sum c'_v \cdot v - \frac{1}{P} \sum c'_v \sum v}{\sum v^2 \frac{1}{P} \left(\sum v\right)^2}, & (14)\\ \hat{b} &= \bar{c}'_v - \hat{a}\bar{v}, & (15)\end{aligned}$$

where P corresponds to the number of minimum values c'_v. Clock drift is then compensated from the normalized delay values using $\hat{f}(x)$, which uses the estimated values in equation 13 instead of a and b. The result is defined as

$$\hat{d}'_i = \hat{d}_i - \hat{f}(i). \tag{16}$$

After that, the actual percentile based algorithm is applied. The normalized and compensated delay values are traversed using the windows W:

$$W = \left\{ \left\{ \hat{d}'_m \right\}_{m=l}^{l+N} \right\}_{l=0}^{M-N}, \tag{17}$$

where N denotes the window size. Each window in W' contains the sorted delay values of the windows W

$$W' = \left\{ \hat{d}'_m \middle| \, \hat{d}'_m < \hat{d}'_{m+1} \right\}. \tag{18}$$

The playout delay necessary to receive a given percentage of packets in time is expressed as the percentile p. It can be read from the sorted window W' by calculating the index of the percentile packet using

$$u = \lfloor p \cdot N \rfloor. \tag{19}$$

(19) must be extended by a phase value, which propagates the error of the index calculation to the next window:

$$\begin{aligned} \phi_l &= p \cdot N - u_l & (20)\\ u_l &= \lfloor p \cdot N + \phi_{l-1} \rfloor. & (21)\end{aligned}$$

The estimated jitter is now calculated as

$$\hat{j} = W'[u_l] - W'[0]. \tag{22}$$

3) Algorithm Comparison: To compare our percentile method to the different playout adaptation algorithms, a virtual playout scheduling algorithm is defined for the percentile based jitter estimation. To test the best theoretical performance of the algorithm, no restrictions are made to time shrinking and stretching for the evaluation, thus, the playout schedule base is re-defined in every window W to the minimum delay in the window. The buffer time is set to the difference between the entry with index u_l and the first entry in window W'. This results in the following playout scheduling:

$$\begin{aligned} p_l &= \hat{j} + min(W) & (23)\\ &= W'[u_l] - W'[0] + min(W) & (24)\\ &= W'[u_l] & (25)\end{aligned}$$

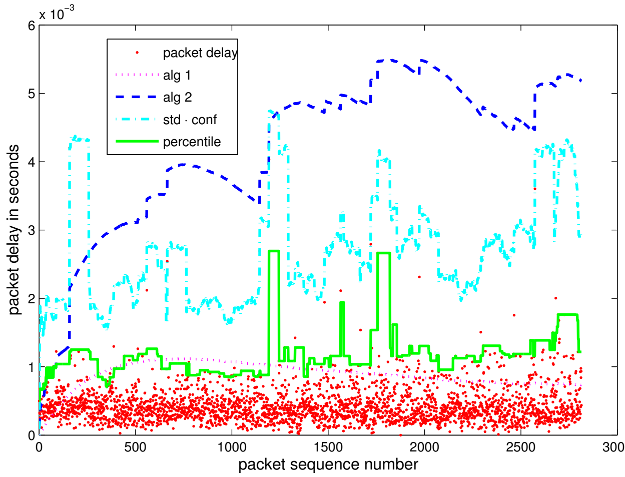

Fig. 5. Overview of jitter estimation algorithms (window size: 100 packets)

In Figure 5, algorithms 1 and 2 from [6], the variance based method ($std \cdot conf$) from [3], and a 99 % percentile based algorithm are plotted over the normalized packet delay. Algorithms 1 and 2 follow delay variations very slowly, which is a result of the weighted moving average method used. The variance based method follows changes in packet delays very fast, but overestimates packet delay jitter. The percentile method, on the other hand, follows the packet delay almost instantly and is therefore used for network jitter estimation by the ACE. We used a window size of 100 packets for the percentile based method. Details of the effect of the window size will be discussed later in this section. The method takes delay spikes into account for jitter estimation as well, in contrast to the algorithm introduced in [10].

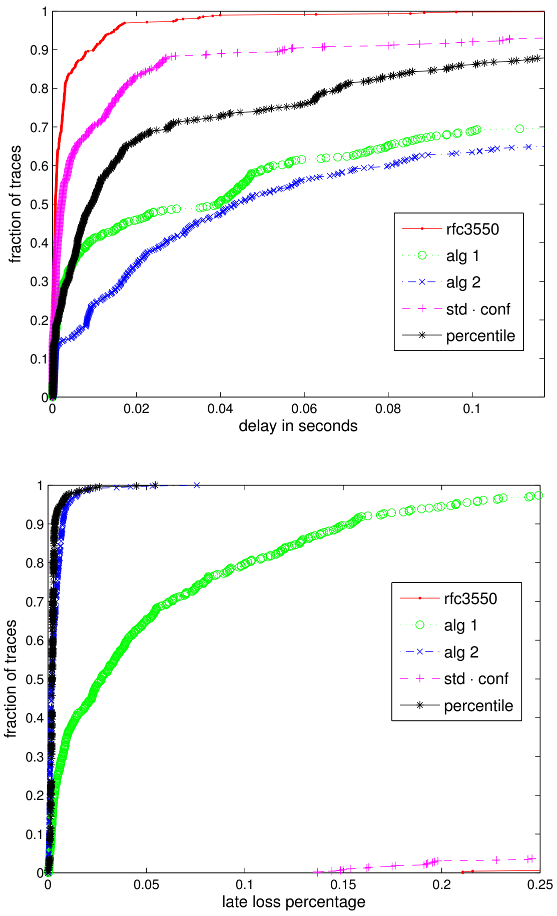

Fig. 6. CDF of delay and late loss

To further analyze the performance of the different algorithms, Figure 6 shows the empirical cumulative distribution functions of the mean delay and the late loss rate for 492 different captured network traces. The mean delay is defined as follows:

$$\bar{d_p} = \frac{1}{M-N}\sum_{k=N}^{M} p_k - r_k. \tag{26}$$

The graph annotated with *rfc3550* shows the jitter estimation as it is carried in RTCP Receiver Reports as specified in [8]. *Algorithm 1* and *algorithm 2* have been taken from [10], $stdv \cdot conf$ is the older *ACE variance* method introduced in [2] and *percentile* represents a (99.8 %) percentile based jitter estimation as described earlier in this section. Regarding the delay, only *rfc3550* and $std \cdot conf$ perform better than the percentile method. Taking a look at the late loss rate, however, the two techniques suffer from very high late loss rates - they appear both at the bottom of the lower graph and provide unacceptable loss rates. *Algorithm 2* provides a late loss close to the late loss rate of the percentile method, however at a higher delay. Our *percentile* method introduced in this paper shows not only very low delay, it is also customizable towards the accepted late loss and the window size of the jitter estimation.

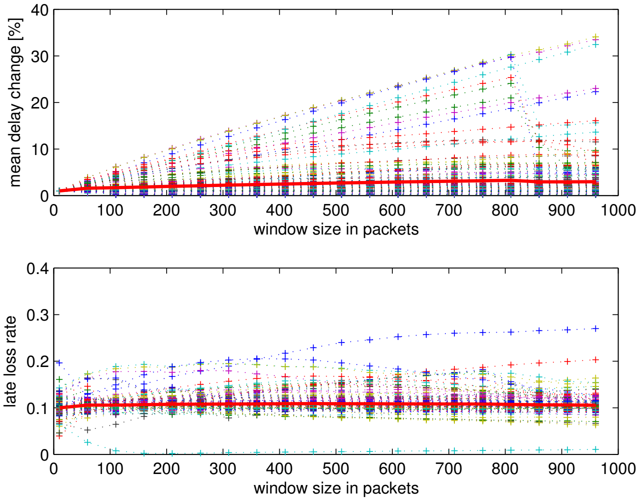

Fig. 7. Window size versus mean delay and late loss from 213 traces

The effects of the window size of the percentile based jitter estimation on mean delay and late loss are not obvious from the algorithm alone and require further investigation. Figure 7 shows the relative change in the mean delay and the jitter estimation window size for 213 network traces. We limited the number of traces for the sake of clarity. The change in mean delay, shown on the Y-axis, is always calculated relative to the mean delay of the smallest window size. This allows the comparison of many different traces within one graph. Window sizes vary from 10 to 1000 packets in steps of 50 packets. The resulting mean delay of each trace is plotted as a plus. The mean over all traces is shown by a thick solid red line. Although the mean delay is very stable, some traces exist with increasing mean delay caused by increasing windows. This effect is originated in the windowing itself: Small windows adapt very quickly to small changes in packet delay without adding much playout delay. Larger windows tend to require more playout delay, especially where large delay spikes occur, but they provide better results for the desired late loss rate.

For some traces, the packet delay shows periodic intervals of high and low jitter. This can be caused for example by interfering wireless devices. The late loss results of such traces converge to the desired late loss rate as soon as the window size reaches the period of the high and low jitter intervals.

Two traces in figure 7 show particularly bad late loss results, reaching up to 30 % late loss instead of the desired 10 %. These traces show varying long term slopes of network jitter, which can not be predicted reliably by any shown algorithm here.

The correct window size depends on the desired adaptation speed and audio quality of the playout adaptation, as well as possible periodic network delay effects. Small windows (e.g. 20 to 500) induce many audio scaling events, which degrade subjective audio quality. This is not acceptable for high quality audio communication. For the ACE, window sizes of 800 to 1000 have proved useful, as they grant good late loss results for most network conditions and react sufficiently to changes in packet delay jitter.

B. ACE Time Scaling

The playout adaptation control of the ACE works separately for stretching and shrinking. In spite of stretching the audio signal pro-actively before a potential delay burst, the ACE keeps the overall delay as low as possible and thus does not maintain any surplus of packets in its buffer. Instead, it uses late loss for re-buffering implicitly whenever a buffer under run occurs. If the packet buffer of the receiver carries more packets than required by the current jitter estimation, time shrinking is accomplished. Both mechanisms are described in the following.

1) ACE Time Stretching: Figure 8 depicts an example: Frame 4 is received too late and has to be concealed due to packet buffer under run. The frame is received nevertheless before frame 5. The ACE then decodes frame 4 next, as if no concealment occurred and thus stretches the audio signal by one frame.

ACE time stretching therefore relies on the quality of the AAC error concealment and that the quality degradation due to playing out late audio frames is negligible if a buffer under run and thus concealment occurs.

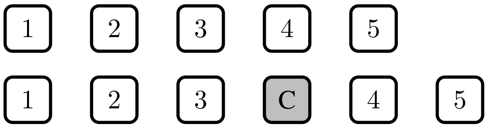

Fig. 8. Access unit decoding order for time stretching

2) ACE Time Shrinking: To reduce the buffer size, the ACE exploits the implicit cross fade of AAC as already explained in Section V. By simply discarding one access unit, the buffer fullness is reduced by one frame length, e.g. $21.3\,\mathrm{ms}$ for 1024 samples at $48\,000\,\mathrm{Hz}$ sample rate. Figure 9 shows the two access unit sequences before and after dropping takes effect.

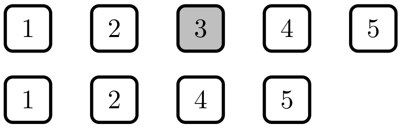

Fig. 9. Access unit decoding order for time shrinking

The amount of dropping is controlled in the ACE receiver by calculating a specific dropping rate. The receiver also decides which access units are dropped and can base this decision on signal properties, like the amount of energy in the frame or other more sophisticated correlations. For the sake of simplicity, we start with a constant dropping rate here. The associated dropping distance ($1/drop\_rate$) specifies the adjacent number of audio frames, after which a drop takes place. The dropping control, however, provides some additional features, which will be explained in the following.

C. Loss to Drop Conversion (LDC)

Frames that are lost on the network are concealed using AAC concealment in general. In lossy networks with varying network delay, however, the audio quality can be strongly impaired by frequent drops due to playout adaptation and concealments due to network loss. Instead of concealing the lost packets, they are interpreted as drops and the total number of concealments is reduced depending on the drop rate. Algorithm 1 is used to convert packet loss to drops.

Algorithm 1 Dropping Instead of Concealment

```
if $pkt\_cnt = drop\_dist$ then
    if $dropped\_early$ then
        $dropped\_early \leftarrow 0$
        $pkt\_cnt \leftarrow 0$
    else
        $drop()$
        $pkt\_cnt \leftarrow 0$
    end if
end if
$pkt\_cnt \leftarrow pkt\_cnt + 1$
if $loss$ then
    if $dropped\_early$ then
        $conceal()$
    else
        $drop()$
        $dropped\_early \leftarrow 1$
    end if
end if
```

The concealment reduction due to loss to drop conversion heavily depends on the network loss. Thus only some traces show significant improvements. To avoid additional concealments, all lost packets can be re-interpreted as drops as long as the buffer contains too many packets according to the jitter estimation.

Figure 10 shows the difference in the number of concealments and mean delay of 212 network traces. All other traces do not show loss and thus have been discarded for this plot. The result is charted in a violin plot, which is a combination of a box plot and a kernel density plot. The width of the shape indicates the density of the traces, like a vertical density plot. The box inside the shape covers the second and third quartiles in solid black and a solid grey dot marks the samples' median.

The mean delay reduction Δd_{ldc} is defined as the relative change in mean delay over all traces between active (d_{ldc}) and inactive (d_{nrm}) loss to drop conversion:

$$\Delta d_{ldc} = \frac{d_{nrm} - d_{ldc}}{d_{nrm}}. \tag{27}$$

The resulting Δd_{ldc} is shown on the left and indicates a maximum mean delay reduction of $80\,\%$. The first and third quartile are close to the median at the bottom of the graph. This confirms the assumption that the improvements occur for certain network conditions only. The cost in terms of late loss Δl_{ldc} as shown in the right plot remains below $2\,\%$. Δl_{ldc} is defined as follows:

$$\Delta l_{ldc} = l_{ldc} - l_{nrm}, \tag{28}$$

where l_{nrm} denotes the loss rate without loss to drop conversion and l_{ldc} the loss rate with loss to drop conversion for each trace. The shape of the violin plot, however, is much more narrow compared to the delay reduction and thus indicates low late loss costs.

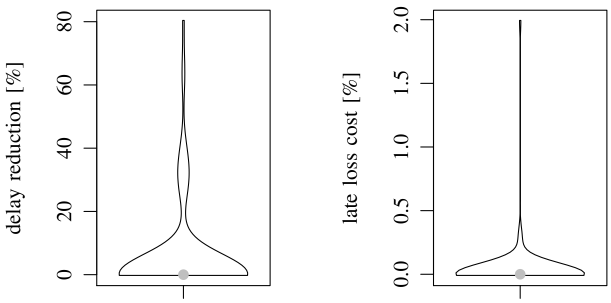

Fig. 10. Loss to drop conversion results (212 real network traces)

To ensure that the late loss to drop conversion never costs late loss without reducing the delay, we need to compare the change in delay to the additional cost due to late loss. Therefore, we introduce another quantity R_{ldc}, expressing the relative delay reduction per concealment for each trace:

$$R_{ldc} = \frac{\Delta d_{ldc}}{\Delta l_{ldc}}. \tag{29}$$

Figure 11 shows the delay reduction per late loss for all traces where late loss is increased by the algorithm. The violin plot contains positive values only and hence proves this assumption. The reduction of delay is up to 1700 ms per concealed audio frame.

D. Surplus Dependent Dropping

Buffer Surplus Dependent Dropping (SDD) can be applied, if delay jitter changes rapidly over time. Figure 12 shows the principles of buffer SDD. The key concept of SDD is to change the dropping rate on the fly, depending on the surplus of the packet buffer. The surplus s of the packet buffer represents the difference between buffered time t_b and jitter estimation $\hat{j}$ using Equation (22):

$$s = t_b - \hat{j} \tag{30}$$

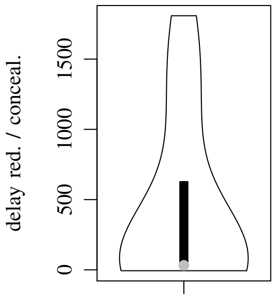

Fig. 11. Loss to drop conversion results (212 real network traces)

The dropping rate δ_{max} indicates the upper limit of dropping, δ_{min} the lower limit. SDD is active between a minimum surplus s_{min} and a maximum surplus s_{max}. The dropping rate is calculated as

$$\begin{aligned} \delta &= \begin{cases} \delta_{min} & \text{if } s \leq s_{min} \\ \beta(s - s_{min}) + \delta_{min} & \text{if } s_{min} < s < s_{max} \\ \delta_{max} & \text{if } s \geq s_{max} \end{cases} && (31) \\ \beta &= \frac{\delta_{max} - \delta_{min}}{s_{max} - s_{min}} && (32) \end{aligned}$$

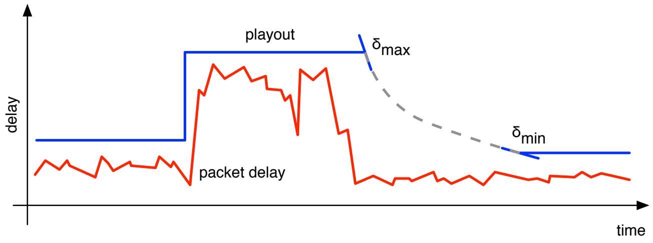

Fig. 12. Buffer Surplus Dependent Dropping

Using SDD, the receiver can catch up faster to heavily decreased jitter while still preserving smooth adaptation for small s. Figure 13 shows the mean delay reduction Δd_{sdd} for SDD ($\delta_{max} = 50\%$, $\delta_{min} = 1\%$, $s_{max} = 100\,ms$ and $s_{min} = 0\,ms$) compared to a fixed dropping rate ($\delta = 1\%$) in a violin plot on the left side. Δd_{sdd} indicates the relative delay reduction between the mean delay using standard adaptation d_{nrm} and SDD d_{sdd} for each trace:

$$\Delta d_{sdd} = \frac{d_{nrm} - d_{sdd}}{d_{nrm}}. \tag{33}$$

The Δd_{sdd} median is at $20\,\%$ and $75\,\%$ of the traces lies between $12\,\%$ and $28\,\%$, thus covering a significant number of traces. The delay reduction reaches up to $88\,\%$ for single traces. The cost in late loss Δl_{sdd} shown on the right hand does not reach $7\,\%$. Δl_{sdd} is defined as the difference between the late loss rate with SDD l_{sdd} and the late loss rate without SDD l_{nrm}:

$$\Delta l_{sdd} = l_{sdd} - l_{nrm}. \tag{34}$$

$75\,\%$ of the traces lie clearly below $1\,\%$. The graphs are sampled from 48 out of 212 network traces, which are selected for a network loss rate below $10\,\%$ and a maximum normalized delay above $100\,\mathrm{ms}$. Traces with maximum normalized delay below $100\,\mathrm{ms}$ showed very little difference in both delay and number of concealed audio frames.

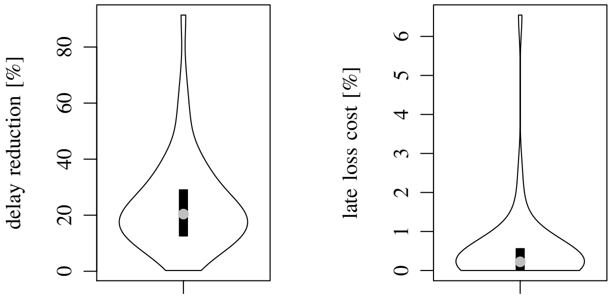

Fig. 13. SDD results taken from 48 out of 212 real network traces

To show that SDD never causes late loss without reducing the mean delay, we express the relative delay reduction per concealment for each trace R_{sdd}:

$$R_{sdd} = \frac{\Delta d_{sdd}}{\Delta l_{sdd}}. \tag{35}$$

Figure 14 depicts the resulting R_{sdd} over all traces where late loss occurs. The violin plot shows only positive values, thus confirming our hypothesis. The mean is close to $100\,\mathrm{ms}$ per concealment, which shows that the delay reduction with SDD strongly depends on the changes of the stationary jitter during the network connection. For some traces, however, the delay reduction per concealment reaches a significant gain of $4000\,\mathrm{ms}$ per late loss.

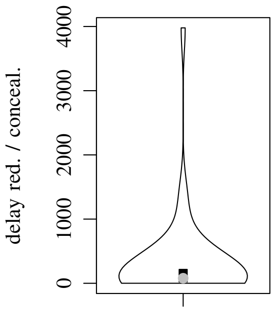

Fig. 14. SDD delay gain per concealment

VI. Conclusion

Adaptive playout is an efficient mechanism to combat network delay variations, also known as jitter. In this paper, we introduced a parametric playout adaptation for high quality audio transmission, which we designed for network connections with varying delay. The framework is based on continuously monitoring network jitter using a percentile based jitter estimator and full band audio time stretching techniques. The percentile based jitter estimator has been compared to existing jitter estimation algorithms using a significant number of network traces and reduces the mean delay by about $20\,\%$ to comparable algorithms. The evaluation of the window size and accepted loss rate towards average delay and measured late loss showed robust results with different window sizes without affecting both parameters. The measured loss rate follows the accepted loss rate closely for most traces.

The time scaling mechanisms exploit the structure of the underlying audio codec like overlap-add and shaped noise concealment with particular low complexity. A mechanism to convert network loss to frame drops improved the audio quality considerably and showed a gain in delay reduction of up to $1700\,\mathrm{ms}$ per additional concealment percentage point. Surplus Dependent Dropping has been introduced to follow long term delay changes quickly and thus reducing the mean delay significantly reaching up to $4000\,\mathrm{ms}$ per additional concealment percentage point.

Therefore, our flexible adaptive playout can significantly increase the reliability of the service under difficult network conditions and is therefore an essential component for high quality audio communication over IP networks.

Acknowledgment

The authors especially would like to thank Sebastian Vogel for the development and deployment of the nettracer application.

References

[1] ITU-T, “G.722.2: Wideband coding of speech at around 16 kbit/s using Adaptive Multi-Rate Wideband (AMR-WB),” Telecommunication Standardization Sector, July 2003.

[2] J. Issing, N. Färber, and M. Lutzky, “Adaptive Playout for VoIP Based on the Enhanced Low Delay AAC Audio Codec,” in *124th AES Convention, Amsterdam*. Audio Engineering Society, May 2008.

[3] J. Issing, S. Reuschl, N. Färber, and R. German, “RTCP based Bit-Rate Adaptation for AAC Audio Communication,” in *Proceedings of NEM Summit 2009, St.Malo*, 2009, p. 6.

[4] M. Schnell, R. Geiger, J. Herre, M. Jander, M. Multrus, M. Schmidt, and G. Schuller, “Enhanced Mpeg-4 Low Delay AAC - Low Bitrate High Quality Communication,” in *122th AES Convention, Munich*. Audio Engineering Society, May 2007.

[5] “ISO/IEC 14496-3 – MPEG-4 Standard: Information technology – Coding of audiovisual objects – Part 3: Audio,” International Organization for Standardization, Geneva, Switzerland International Electrotechnical Commission, 2008.

[6] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, “Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks,” in *INFOCOM ’94. Networking for Global Communications., 13th Proceedings IEEE, Toronto*, June 1994, pp. 680–688.

[7] A. P. Markopoulou, F. A. Tobagi, and M. J. Karam, “Assessing the Quality of Voice Communications over Internet Backbones,” *IEEE/ACM Transactions On Networking*, vol. 11, pp. 747–760, 2003.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” RFC 3550 (Standard), jul 2003, updated by RFC 5506. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[9] H. Schulzrinne, “Voice communication across the Internet: a network voice terminal,” University of Massachusetts, Tech. Rep. UM-CS-1992-050, 1992.

[10] S. B. Moon, J. Kurose, and D. Towsley, “Packet audio playout delay adjustment: performance bounds and algorithms,” *Multimedia Systems*, vol. 6, no. 1, pp. 17–28, 1998.

[11] Y. J. Liang, N. Faerber, and B. Girod, “Adaptive playout scheduling and loss concealment for voice communication over IP networks,” *IEEE Transactions on Multimedia*, vol. 5, pp. 532–543, 2003.

[12] J. Herre, “Temporal Noise Shaping, Quantization and Coding Methods in Perceptual Audio Coding: A Tutorial Introduction,” in *Proceedings of the 17th International AES Conference on High Quality Audio Coding*, Signa, Italy, September 1999.

[13] “ISO/IEC 13818-3 – MPEG: Information technology – generic coding of moving pictures and associated audio – Part 3: Audio,” International Organization for Standardization, Geneva, Switzerland International Electrotechnical Commission, 1994.

[14] K. Brandenburg and G. Stoll, “ISO-MPEG-1 Audio: A Generic Standard for Coding of High Quality Digital Audio,” in *Collected Papers on Digital Audio Bit-Rate Reduction*. Audio Engineering Society, May 1996.

[15] K. Tsutsui, H. Suzuki, O. Shimoyoshi, M. Sonohara, K. Akagiri, and R. M. Heddle, “ATRAC: Adaptive Transform Acoustic Coding for MiniDisc,” in *93rd AES Convention, San Francisco*. Audio Engineering Society, October 1992.

[16] C. Perkins, *RTP - Audio and Video for the Internet*. Addison-Wesley, 2003.