Enhanced Adaptive Playout Scheduling and Loss Concealment Techniques for Voice over IP Networks

Dinei Florencio and Li-Wei He
Microsoft Research
One Microsoft Way, Redmond, WA 98052
Email: {dinei, lhe} @microsoft.com

***Abstract*— Voice over IP (VoIP) is already of commercial quality when traversing corporate, or other high-quality networks. Nevertheless, to make the final bridge to toll quality over standard internet connections, a few problems remains to be solved. These include packet loss, jitter control, and clock drift compensation. Previous research in adaptive playout mechanisms has significantly contributed towards that end. In this paper, we present a novel technique that enhances previous adaptive playout mechanisms, and virtually eliminate late losses. Additionally, enhanced stretching and loss concealment algorithms are also presented. These work in harmony with the proposed technique to alleviate network jitter and any clock drift.**

I. Introduction

Transmitting real time digitized speech over a best-effort packet networks (such as the Internet), is appealing for a number of reasons, including the wide availability and low cost of such networks. For that reason, Voice over IP (VoIP) has become very popular. Nevertheless, to achieve the high quality standards users expect from commercial solutions, a number of issues inherent to the network need to be addressed. Three issues may particularly affect call quality: packet loss, delay jitter, and clock drift.

Packet loss is inherent to the "best effort" characteristics of these networks, and it becomes more pronounced in the Wifi [1] and other lower quality connections. Traditional forward error correction (FEC) techniques would introduce significant extra delay, and thus are not appropriate for real-time communication. So, most VoIP systems use none or simpler FEC techniques (e.g., packet repetition), and rely aggressively on loss concealment techniques. Delay, and delay jitter is another problem. If a packet has not been received by the time it has to be played out, the decoder will have to interrupt the speech stream. This is generally referred to as "late loss". The old solution was to buffer enough packets to make sure the probability of a late loss is small, but this requires a large buffer, and therefore a long delay. More recent solutions use an adaptive playout strategy [2], to keep a small buffer size, while reducing late losses. But current adaptive playout technology still incur in late losses [3]–[5]. One of the key contributions of this paper is to propose a new buffer management algorithm that essentially eliminates late losses. Finally, a third problem afflicting VoIP is related to the clock mismatch between the sender and receiver causing accumulation/exhaustion of the receiver's audio buffer.

The effect of packet losses can be mitigated by forward error correction or error-resilient audio coding techniques. The occurrence of late loss is traditionally minimized by introducing a *jitter buffer*, which stores packets and provides them to a decoder at more regular time intervals. If the size of the jitter buffer were at least as long the difference between the smallest and largest possible delays, the late loss would be eliminated entirely. On the other hand, the jitter buffer also introduces an extra delay, which is undesirable for real-time conversations. So, in practice, the buffer size is set to some length which is a compromise between late loss and delay. In general, the size of this buffer can only be adjusted during silence periods (e.g., between talk spurts). The size of the buffer can be optimized using, e.g., the algorithms in [6]. If the receiver clock is slower than the sender's, the accumulated receiver buffer is also adjusted during silence periods. If the receiver clock is faster, playout gaps often have to be introduced which degrade the speech quality.

By using speech time-scale modification techniques, the jitter buffer length can be adapted during speech utterances, reducing the average delay, without incurring in as many late losses. The technique introduced by Liang et. al. [2] is the most widely employed of such techniques (see, e.g., [3]–[5]). Their algorithm also handles the packet loss concealment due to either *network loss* or *late loss*. The perceived speech quality is significantly improved as a result.

In this paper, we describe a set of algorithms to conceal packet losses, alleviate packet delay jitter ("de-jitter"), and handle clock mismatch between sender and receiver in a unified framework. Although our simulations are based on G722.1 [7], the algorithms are audio codec independent and can be implemented on the receiver alone. In some aspects, our algorithms are similar to those described in [2], but they have two important improvements: 1) our playout algorithm introduces a smaller buffer delay and completely eliminates "late losses", and 2) we utilize specialized stretching algorithms for unvoiced and transitional speech which alleviates the metallic artifacts often introduced when stretching speech.

II. Improved Playout Algorithm

In adaptive playout, the buffer size is minimized by allowing each packet to be stretched and/or compressed. In existing techniques [2]–[5], the past statistics of receiving time is used to assign each packet a *scheduled playout time*. The current packet is then stretched or compressed in order to allow the next packet to be played at the scheduled time. A 1-packet delay is introduced, in order to wait for a packet to be received (or declared lost), before deciding on how to play the current packet (i.e., stretched or not). This is illustrated in Figure 1.

One of the main differences between our solution and the one proposed in [2] is that theirs is "packet-based", while ours is "buffer-based". More precisely, we will not decide to stretch or compress a packet when we receive it; instead, we simply append it to the buffer. We decide to decode, stretch, and compress only when the audio playout device needs a frame. Furthermore, the decision is based only on the size of the jitter buffer.

The algorithm needs to set a minimum buffer size B_{min} and a maximum buffer size B_{max}. We use a fixed value of 10 ms as minimum size. The maximum size controls the tradeoff between the probability of needing to stretch and the audio delay. It is set to 500 ms initially, and then adjusted depending on whether a stretch is performed or not during the playout procedure. The audio buffer will converge to a size that will cause around one stretch out of every 20

```
1   receive packet i;
2   estimate and set the playout time for packet i + 1;
3   calculate the desired length of packet i, L_i;
4   if L_i − L_0 ¿ expansion threshold
5       stretch packet i;
6   else if L_i − L_0 ¡ - compression threshold
7       compress packet;
8   else
9       keep packet i without modification;
10  end if
11  output packet i with actual length L_i;
12  update the playout time of packet i + 1;
```

Fig. 1. Adaptive playout algorithm proposed in [2]

```
1   if B + F < B_min then
2       if undecoded list is not empty then
3           Decode a packet
4           S = 0
5           if there is gap (packet loss) then
6               Conceal Loss
7           endif
8       endif
9       if B + F < B_min then
10          if S > S_max then
11              Output Comfort Noise
12          else
13              Stretch by B_min − (B + F)
14              S+ = Bmin − (B + F)
15          endif
16      endif
17  else if B + F > B_max then
18      Compress by 1 pitch period
19  endif
20  if there is stretch then
21      B_max+ = F
22  else
23      B_max− = F/20
24  endif
```

Fig. 2. Proposed playout algorithm, which is invoked when the audio device needs an audio frame

frames pulled. If the jitter is big, so will the buffer. But the number of stretches (hence the speech quality) should be roughly constant. The buffer should converge to a smaller size if the jitter is reduced later.

In particular, note that if, say, packet n arrives later than scheduled, we would keep stretching the signal existing in the buffer until that packet arrives, or it is declared "lost". This is an important improvement over [2], where the packet would be declared as "late loss". The algorithm proposed in [2] sets a time limit for receiving packet n, as soon as the prior packet (i.e., $n-1$) is received. If packet n is not received by that time, a late loss is declared, and the algorithm goes into "loss concealment" mode. Figure 2 presents our algorithm for declaring a packet as lost. As it can be seen in that figure, only after a subsequent packet is decoded, we would go into a "loss concealment" mode. This essentially eliminates "late losses".

III. Improved Speech Time-Scale Modification

A core element of our algorithms is time-scale modification of speech. The stretch/compress operation can be performed on a frame basis, and small variations of compression ratio from the desired ratio can be compensated at the next frame.

Suppose we have, 20ms of speech that has not yet being played, and we have a target size of stretching that to 40 ms. The first step on our stretching procedure is similar to most stretching algorithms in the literature, and is based on SOLA [10]. We select a smaller segment from the 20 ms (i.e., a *template*), and look for a similar segment in the recent history of the signal (possibly in the part of the signal that has already been played). This search is done by a normalized cross correlation measure, and the range is limited to the range compatible with pitch. For example, on a 16 KHz sampled speech, pitch period is typically between 40 and 250 samples. Additionally, we further limit the low end of the pitch period range to 80 samples instead of 40. Thus, for pitch periods below 80, we will end up finding the maximum autocorrelation at twice the pitch. The artifacts from that are mostly null, while a half-pitch would produce much more noticeable artifacts.

We look for the peak of the normalized cross correlation, and classify the segment as voiced or unvoiced. We currently use a hard threshold of .65. Each kind of segment: voiced or unvoiced, is stretched (or compressed) with a different algorithm. Cross correlation values between .40 and .95 may indicate a segment is partially voiced. We experimented with a third mode (mixed), but the quality gain did not seem to justify the additional computational complexity, and we decided to drop the mixed mode.

A. Voiced Segments

For segments classified as voiced, a windowed overlap-add (SOLA) approach is used. This is mostly as published in the literature, except we introduce a few enhancements. First, we changed the location of the segment to be used as reference, which we place at alternating points, instead of always at the beginning or end of the window. Second, we change the window size (which we may make similar to last pitch period). And third, we change the position of the template, which we place such that the mid-point of the transition window is located at a low-energy point of the waveform. If more stretching is desired, the process is repeated as needed.

We first estimate how many times we need to stretch the segment (k), based on a pitch estimate p. Each iteration will compress or stretch the signal by one pitch period, so a good estimate is $k = |M - N|/p$. We then uniformly distribute the templates over the segment to be stretched. If past history of the signal is available, the match is searched for in the region before the template. If no past history is available, we may look for a match before or after, depending on which one has more data available.

B. Unvoiced Segments

This is one of our important contributions. For unvoiced segments, we do not want to introduce any periodicity in the signal. Repeating past signal (as in the voiced case) would introduce such periodicity, and produce artifacts. Instead, we replace the signal with a different signal with the same power spectrum. More precisely, we do that by computing the Fourier transform of the recent past signal, introducing a random rotation of the phase of the FFT coefficients, and computing the inverse FFT. Additionally, to enhance the fit of the spectral analysis, we use an asymmetric window [12], [13]. This produces a signal with the same spectrum, but no correlation with the original segment. Longer signals can be obtained by zero-padding the signal

before computing the FFT. We then take the beginning and end of the original signal apart, and insert this new signal in the middle, with windowing and overlapping to smooth the transitions. Note that by doing this we kept the beginning and end of the segment the same, as needed to avoid artifacts on the transition to the already played speech and next frame, respectively. We also scale down the inserted noise (reducing the energy by 36%) to minimize any artifacts. Another key difference is the overlapping/smoothing window. Because the two overlapping signals are supposedly independent (instead of correlated as in the voiced case), the windows w_a and w_b should be such that $(w_a[n])^2 + (w_b[n])^2 = 1$, instead of $w_a[n] + w_b[n] = 1$, as in the voiced case. To satisfy this constraint, we use a sine window for the overlap of unvoiced frames and a Hann window for the voiced segments.

The method proposed above eliminates the artifacts produced by periodization, since no periodic extension is introduced. Nevertheless, it may spread the spectrum of a transient over a much longer period of time, sometimes producing another kind of artifact (even if not as objectionable as the periodic one). To solve this problem, we modified the algorithm to produce a smaller FFT, with more localized spectral information: We set the size of the FFT to a pre-defined length (e.g., 128), and compute how many overlapping segments (v) will be needed to obtain the desired final size (M), not counting the first and last half-segments (i.e., $v = (M * 2/FFT\_SIZE) - 1$). Then we spread the center of of these v segments over the available speech segment. If we spread these uniformly, all segments are stretched (or compressed) equally. But, ideally, some parts (e.g., lower energy parts) should be stretched more, to help reduce artifacts. To achieve that, we introduce a non-uniform selection of the stretching points. First, we select initial points for the analysis windows, s[i], as points spread FFT/2 samples apart, defining a number of signal segments. Note that these would not imply in any stretching. Then we recursively split one of the segment (adding new stretch points), until achieving v points. Note that each additional point will imply extra 64 samples. In order to preserve more of the original signal, we want as many starting points as possible to be FFT/2 samples apart. We begin by placing these starting points FFT/2 samples apart. We then insert as many new points in between existing points as needed, one at a time. We insert the new points in the lowest energy segments. For segments of different lengths, we weight the average energy of the segment by the square root of the segment size, to favor splitting longer segments. In the final distribution, many points will still be FFT/2 apart. These segments (more likely the high energy segments), do not need to be modified.

IV. Improved Loss Concealment

As mentioned before, packet losses may occur. Figure 2 shows the procedure we use to declare a packet as lost (i.e., to stop waiting for the packet). Once a packet is declared lost, it can trigger either "loss concealment" or a "comfort noise" mode.

A. Comfort Noise Mode

A comfort noise mode is entered when a packet is not received for a length of time superior to a pre-set threshold S_{max} (in our case 200 ms). This is interpreted as either the end of a talk spurt or a loss of connection. In either case, the receiver will (gradually) mute the current signal and go to a comfort noise mode. This comfort noise is a common feature in many communications systems, and has the objective of trying to simulate the same noise level present when the connection was active, but there was no speech.

We keep a number (e.g., 3) of "silence frames". Whenever a new frame is received, we compute the overall energy of the frame E, and compare to the stored energy of the current silence frames, E_1, E_2, and E_3. If the current frame has lower energy than any of the three, we then replace the highest energy of the three with the current frame. Besides the energy of the frame, we also store the magnitude of the FFT coefficients of the frames, to be used in synthesizing a "comfort noise" frame (to be described next paragraph). Finally, to force a periodic renewal of the silence frames, and also avoid an atypical low energy frame to stay there forever, we include a time-out mechanism. If a particular frame is in the buffer for over 15 seconds, we increase its nominal energy E_i (but not the magnitude of the stored FFT coefficients). This will increase the likelihood that the frame will eventually be replaced. The E_i is doubled every 15 seconds, and a small amount of an energy (equals 1% of the current frame) is added, to handle the cases where $E_i = 0$.

Finally, when a comfort noise frame is needed, we use the information in the silence frames to generate one. More precisely, we compute the average magnitude of the three stored silence frames, add a random phase, and compute the inverse FFT. This signal is then overlapped / added to the signal in the buffer, by the transition (sine) window.

B. Loss Concealment Mode

As described in connection with Figure 2, if a subsequent frame is received, and one or more intermediate frames are still missing, we may declare a packet as lost, and enter into a "loss concealment mode". Loss concealment may be generic or specific to a codec. Many codecs already have loss concealment algorithms specified as part of the codec, and in that case one may simply do the concealment by using those. In other cases, the prescribed loss concealment may not exist or be sub-optimal and other methods are used. This is often the case, since most loss concealment algorithms have the constraint of preserving the (fixed) length of the frame, which, in our case, is irrelevant.

We will first describe a loss concealment mode designed for G.711 (PCM) coded speech, but which may also be appropriate for use with many other codecs. First, we note that the concealment mode will only be entered when we have received at least one subsequent frame. For that reason, besides the signal still remaining in the "current buffer", we have also some (non-contiguous) segment of the signal, which we call "future buffer". The lost segment corresponds to the missing samples between theses two buffers, which were never received.

We start by deciding the number of samples we want to insert (K). In principle, K could be simply the number of samples corresponding to the lost frame(s), but we do a slightly more elaborate computation. First, due to our approach to buffer stretching (as opposed to frame-based, as in [2]), we may have already stretched the last frame received. In that case, we will subtract that from the lost samples. Also, to allow enough space for the appropriate transition window, we need to stretch an additional Ov samples. Finally, if too many frames were lost, the (synthetic) transition is not likely to be a natural-sounding one. For that reason, and to reduce the artifacts, we limit the number of frames to be replaced to two frames. Note that, if necessary, the signal may later be further stretched at some other point by the jitter control algorithm.

The next step is to compute a desired (target) size for the current and future buffers, D_C and D_F, based on the current size of the buffers, E_C and E_F. The simplest method again would be to split the $K + Ov$ samples to be created equally between the two, i.e., to make:

$D_F = E_F + (K + Ov)/2$. We then proceed to stretch the future buffer by the desired amount. Since the stretching process may produce slightly more (or less) than the desired samples, we then proceed to estimate the necessary length of the current buffer as $D_C = E_F + E_C + Ov + K - A_F$, i.e., the desired total length, plus the required overlap (for the overlap/add process), minus the actual size of the future buffer after stretching, A_F. We then proceed to stretch the current buffer by the desired amount, and finally, do the overlap/add to mix (fade) the current and future buffers. A few minor enhancements on choosing the target (desired) size for stretching the future buffer. As we mentioned before, a simple decision would be to distributed the required samples equally between the buffers. Instead, we introduce three main modifications. First, if the future segment is too small (e.g., do not contain at least two pitch periods), we do not stretch it at all. This is because stretching a voiced segment without having at least two pitch periods would almost certainly introduce noticeable artifacts. Second, the way we split the stretching between the current and future frames is a function of the energy of each frame. More specifically, we stretch each buffer in direct inverse proportion to its energy. We do that because in general, stretching a low energy signal close to a high-energy signal will mask the artifacts. Finally, we set lower and upper thresholds for stretching. If one of the buffer signals would be stretched by less than 10% of the desired new samples, we stretch only the other buffer, and leave that one as is.

Finally, we now present the details of the final overlap/add between (stretched) current and future buffers. The overlap/add procedure is similar to others in the literature, with two important differences: first, we only align the buffers before overlap if both are voiced. And, second, we use different windows in that case then for the standard overlap. For the same reasons described in Section III-B, if both frames are voiced, we use a Hann window, and align the segments before the overlap/add. If not (i.e., at least one of the frames is unvoiced), we use a Sine window, and do not align the segments.

V. Implementation and Results

The proposed algorithm was implemented, and compared to [2], and to a few commercial solutions. Initial tests involved 7 subjects. The packet loss and delay were simulated based on 10 traces obtained from real VoIP sessions. For traces with low packet loss rate (e.g., below 1%), performance of all algorithms was similar. As the packet loss rate increases, the performance differences become more significant, with clear advantage of the proposed algorithm. A second pairwise test compared the proposed algorithm to the alternative that best performed on the first test. For the second test, a total of 11 subjects rated a total of 299 sentence pairs. In 126 instances, the user indicated no preference. From the remaining 173 pairs, users preferred the proposed algorithm 62% of the time. For traces with error rates above 10%, the preference rate was 100%.

VI. Conclusions and Future Work

In this paper we presented a new algorithm for adaptive addressing the three main problems inherent to VoIP. The proposed algorithm is based on stretching/compressing the contents of the buffer, thus essentially eliminating "late losses". This is a significant improvement over the existing frame or packet based technologies, which still incur in late losses. Furthermore, we introduced enhancements to the traditional techniques used to stretch/compress speech, which significantly reduced artifacts. A recent empirical study [11] compared several VoIP solutions (Skype, Google Talk, and MSN Messenger), concluding that MSN messenger buffer management performs better. Although that paper independently corroborate the quality results, it seems to ignore the fact that efficient buffer management is dynamic.

We are currently doing further quality evaluation on the algorithm based on CrowdMOS [18], [19]. Of particular interest, is the case where the speech signal is of poor quality, e.g., when acquired with a far field microphone, as well as cases where spatial sound information is acquired through sound source localization [14], [15], and conveyed on the remote side by use of spatialization techniques [20].

References

[1] A. Conceicao, J. Li, D. Florencio, and F. Kon, "Is IEEE 802.11 ready for VoIP?," *in Proc. IEEE Int. Workshop on Multimedia Signal Proc. (MMSP)*, Oct. 2006.

[2] Y. Liang, N. Farber, and B.Girod, "Adaptive playout scheduling and loss concealment for voice communication over IP networks," *IEEE Transactions on Multimedia*, April 2001.

[3] S. Chi and B. F. Womack, "QoS-based optimal adaptive playout-buffer scheduling using the packet arrival distribution," *Proc. of IEEE MILCOM*, pp. 15, Oct. 2009.

[4] H. Li, G. Zhang, and W. Kleijn, Adaptive playout scheduling for VoIP using the K-Erlang distribution," *Proc. of EUSIPCO*, Aug. 2010.

[5] J. Aragao Jr., and G. Barreto, "Novel approaches for online playout delay prediction in VoIP applications using time series models," *Computers & Electrical Engineering*, Volume 36, Issue 3, pp.536-544, May 2010.

[6] R. Ramjee, J. Kurose, and D. Towsley, "Adaptive playout mechanisms for packetized audio applications in wide-area networks," *in Proc. of INFOCOM*, vol. 2, pp.680-688, Jun 1994.

[7] International Telecommunication Union ITU, "Coding at 24 and 32kbits/s for hands-free operation in systems with low frame loss", ITU-T recommendation G.722.1, Sept. 1999.

[8] D.J. Goodman, G.B. Lockhart, O. J. Wasem, and W-C.Wong, "Waveform substitution techniques for recovering missing speech segments in packet voice communications," *IEEE Trans. on Acoustics, Speech and Sig. Proc.*, Vol ASSP-34, n. 6, pp.1440-1448, Dec 1986.

[9] D. Florencio and R. Schafer, "A nonexpansive pyramidal morphological image coder," *in Proc. IEEE Int. Conf. Image Processing (ICIP)*, vol. 1, pp.331-335, 1994.

[10] W. Verhelst and M. Roelands, "An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech," *in Proc. of IEEE Int. Conference on Acoustics, Speech, and Signal Proc. (ICASSP)*, vol. II, pp.554-557, Apr. 1993.

[11] C. Wu, K. Chen, C. Huang, and C. Lei, "An empirical evaluation of VoIP playout Buffer dimensioning in Skype, Google Talk, and MSN messenger," *in Proc. of NOSSSDAV*, 2009.

[12] D. Florencio, "On the use of asymmetric windows for reducing the time delay in real-time spectral analysis," *in Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Proc. (ICASSP)*, pp.3261-3264, 1991.

[13] D. Florencio, "Investigating the use of asymmetric windows in CELP coders," *in Proc. IEEE Int. Conf. Acoustics, Speech, Signal Proc. (ICASSP)*, pp.427-430, Apr. 1993.

[14] C. Zhang, Z. Zhang, and D. Florencio, "Maximum likelihood sound source localization for multiple directional microphones," *in Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Proc. (ICASSP)*, 2007.

[15] Y. Rui, D. Florencio, W. Lam, and J. Su, "Sound source localization for circular arrays of directional microphones," *in Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Proc. (ICASSP)*, 2005.

[16] C. Zhang, D. Florencio, D. Ba, and Z. Zhang, "Maximum likelihood sound source localization and beamforming for directional microphone arrays in distributed meetings," *IEEE Trans. Multimedia*, vol. 10, no. 3, pp.538-548, Apr. 2008.

[17] F. Ribeiro, C. Zhang, D. Florncio, and D. Ba, "Using reverberation to improve range and elevation discrimination for small array sound source localization," *IEEE Trans. on Audio, Speech and Lang. Proc.*, vol. 18, no. 7, pp.1781-1792, Sept. 2010.

[18] F. Ribeiro, D. Florencio, C. Zhang, and M. Seltzer, "crowdMOS Standalone Tools," available at http://www.crowdmos.org/download/, 2011.

[19] F. Ribeiro, D. Florencio, C. Zhang, and M. Seltzer, "crowdMOS: an approach for crowdsourcing mean opinion score studies," *in Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Proc. (ICASSP)*, 2011.

[20] M.-S. Song, C. Zhang, D. Florencio, and H.-G. Kang, "Personal 3D audio system with loudspeakers," *in Proc. of Hot3D (ICME)*, 2010.