Multi-Channel Noise/Echo Reduction in PulseAudio on Embedded Linux

Karl FREIBERGER and **Stefan HUBER** and **Peter MEERWALD**
bct electronic GesmbH
Saalachstraße 88
A-5020 Salzburg, Austria
{k.freiberger, s.huber, p.meerwald}@bct-electronic.com

Abstract

Ambient noise and acoustic echo reduction are indispensable signal processing steps in a hands-free audio communication system. Taking the signals from multiple microphones into account can help to more effectively reduce disturbing noise and echo. This paper outlines the design and implementation of a multi-channel noise reduction and echo cancellation module integrated in the PulseAudio sound system. We discuss requirements, trade-offs and results obtained from an embedded Linux platform.

Keywords

Acoustic echo cancellation, noise reduction, hands-free telephony, beamforming, performance

1 Introduction

At bct electronic, we develop a speakerphone for hands-free Voice over Internet Protocol (VoIP) telephony and intercom. On our communication device, we run a custom embedded Linux system created with OpenBricks[1]. The device is designed for desktop or wall-mount use, has a 7" touch-screen and is powered by a TI OMAP3 processor (DM3730). Two independent hardware audio codecs enable hands-free communication as well as hand-set or headset use at the same time in order to support flexible intercom and VoIP scenarios.

Speech quality is a very important criterion for us. Therefore, our device is equipped with a 4-channel array of digital, omnidirectional MEMS microphones[2]. This allows to reduce noise without distorting the desired speech signal [Souden et al., 2010]. However, elaborate digital signal processing (DSP) is required to achieve good speech quality in challenging acoustic environments with high levels of ambient noise.

Several open-source software components are available in our application area: SIP stacks (Linphone[3], Sophia SIP[4]), audio compression codecs (G722, Opus[5]), sound servers (JACK [Davis, 2003], PulseAudio[6]), DSP primitives for resampling and preprocessing (Speex[7]), to give a few examples. Open-source SIP software has gained support for single-channel acoustic echo and noise reduction (AENR) recently. However, we are not aware of an open-source framework for multi-channel audio communication and AENR.

In section 2 we describe the acoustic setting and the related challenges in AENR. The basic principles behind common methods are explained. In section 3 we motivate the use of PulseAudio as a sound server and integrating component of our software architecture and outline the design and implementation of a multi-channel AENR plug-in module. While we cannot release the DSP code at this point, several improvements to PulseAudio have been made available to enable multi-channel audio processing on embedded Linux platforms. Section 4 outlines algorithms for multichannel AENR. We have prototyped the algorithms in MATLAB and Octave on the PC, transcribed the code to C/C++, and successively adapted and optimized the code to target the ARMv7 platform. Runtime performance analysis and optimization techniques are discussed in section 5. The test setup and experimental results are detailed in section 6. Finally, Section 7 summarizes results and outlines further work.

2 Acoustic Echo and Noise Reduction

The acoustic front-end of a basic speakerphone comprises a microphone for picking up the near-end speaker (NES) and a loudspeaker for play-

[1] http://www.openbricks.org
[2] http://mobiledevdesign.com/tutorials/mems-microphones
[3] http://www.linphone.org
[4] http://sofia-sip.sourceforge.net
[5] http://www.opus-codec.org
[6] http://www.pulseaudio.org
[7] http://www.speex.org

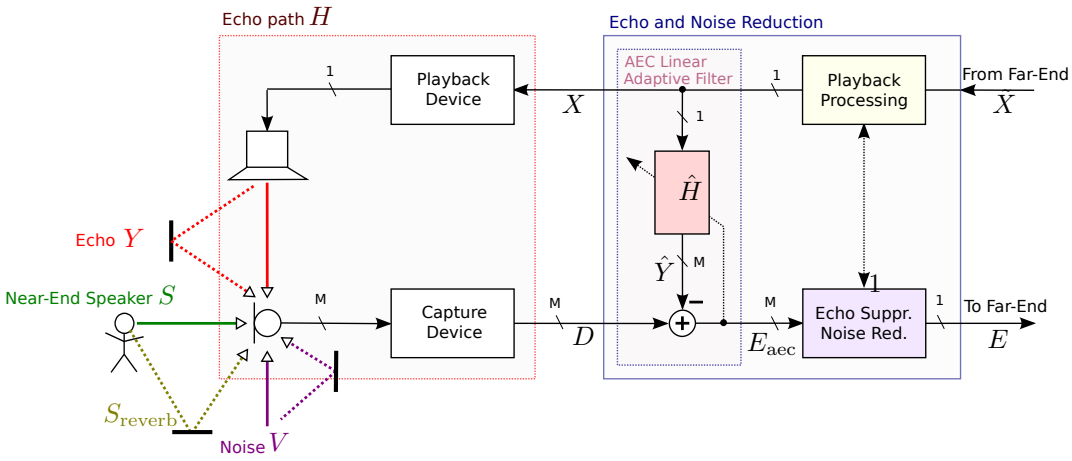

Figure 1: Near-end acoustic setting and general AENR system for one loudspeaker and M microphone channels. For $M > 1$ the echo suppression & noise reduction module may include beamforming. The dashed, colored lines indicate room reflections.

ing back the far-end speaker (FES), see Fig. 1. In practice, the captured microphone signal D does not only contain the desired NES signal S but also undesired components that degrade speech intelligibility, namely room reverberation S_{reverb}, the so-called *echo signal* Y and an additive noise signal V:

$$D = S + S_{\text{reverb}} + Y + V \tag{1}$$

Here, S_{reverb}, Y and V are mutually uncorrelated, S_{reverb} is correlated (only) with S and Y is correlated only with the playback signal X, containing the FES. V denotes all other unwanted parts neither correlated with S nor X. The challenge is to remove or at least reduce the undesired components without (too much) distortion of S.

2.1 Acoustic Echo

The echo signal can be written as $Y = H\{X\}$, where $H\{\cdot\}$ denotes the echo path system consisting of playback device, loudspeaker, room, microphone and capture device. The term "echo signal" stems from the fact that Y is contained in D and, thereby, a delayed and filtered version of the FES signal X is sent back to the far-end. It follows that if the near-end device has an insufficient echo-reduction system, an echo becomes obvious on the far-end. The larger the delay of the echo, the more irritating is the echo of a given level, cf. [Hänsler and Schmidt, 2004]. The overall delay of the echo signal consists of delays due to capture and playback, the acoustic path, the speech codec and VoIP transmission. Because of the limited physical size of a speakerphone, the loudspeaker is located close to the microphone. The level of the echo might hence be several times higher than that of the NES. This makes high quality echo cancellation and/or suppression indispensable.

The terms cancellation and suppression — they are subsumed under the term *reduction* in this paper — shall not be confused: The idea behind echo cancellation is to find an estimate $\hat{Y}$ of the echo and subtract it from the microphone signal, i.e., $E_{\text{aec}} = D - \hat{Y}$, with $\hat{Y} = \hat{H}\{X\}$. By inserting D from Eq. (1), one can see that Y can be fully removed without distorting S if Y equals $\hat{Y}$. Most practical systems use a linear adaptive filter with finite impulse response (FIR) to identify and model the echo path H. Nonlinear models exist, but are in less widespread use due to their higher complexity and slower convergence.

In practice, there are several reasons why the adaptive filter does not fully cancel the echo and a residual echo (RE) Y_{res} remains in E_{aec}: The adaptive FIR filter (i) does not model the nonlinearity of the loudspeaker or a potential clipping of the echo signal, (ii) is too short to model the echo path impulse response $h(t)$, (iii) is too slow to follow changes of the echo path, and (iv) does not fully converge or even diverge due to double talk. As a consequence, E_{aec} is usually further processed by a RE suppression postfilter. The principle of suppression is to apply a real gain factor $G(l, f)$ to the input of the suppression filter. Because echo suppression is typically performed in the frequency domain or subbands of a filterbank, the indices l and f are introduced to indicate the time-

and frequency-dependence, respectively. If D is directly plugged into a suppression filter, we have $E_{\text{suppr}}(l, f) = D(l, f) \cdot G(l, f)$. Looking at Eq. (1), we see that suppression of echo or noise goes along with suppression of the NES S. Because Y and S do typically not fully overlap in the time-frequency plane, duplex communication is possible at least to some extend.

2.2 Ambient Noise

In our application, the NES shall be able to move freely around the device and still be picked up flawlessly, even when being several meters away from the microphone and having a low level. Therefore, the microphone must be very sensitive and/or highly amplified. As a consequence, we face high levels of ambient noise, e.g., fan noise in an office, traffic noise, as well as the acoustic echo described above. Reverberation and the self-noise of the microphone must also be taken into account.

In the single microphone case, noise reduction (NR) is based on the suppression principle. To compute the suppression filter $G_{\text{noise}}(l, f)$, the power spectral density (PSD) of the noise must be estimated. This can be done in speaking pauses, i.e., when $S = 0$ is detected by voice activity detection. Today, more advanced statistical methods are typically used [Hänsler and Schmidt, 2004]. These allow for updating the noise estimator even in times when both V and S are active. Still, single channel NR delivers best results if the noise is stationary, i.e., the noise PSD does not change much over time. Otherwise, the PSD estimation is likely to be inaccurate, which may cause unnatural artifacts in the residual noise and speech. Typically, strong single channel noise reduction comes at the cost of speech distortion. However, it is theoretically possible to perform single channel NR without speech distortion [Huang and Benesty, 2012].

By using more than one microphone, we can not only exploit time-frequency information but also spatial information. This allows for improved NR, which is discussed in section 4. At this point we note that the cancellation principle can also be applied to NR if a reference of the noise signal is available. In section 4 we explain how a so called blocking matrix can provide a noise reference in adaptive beamforming.

3 Echo Cancelling in PulseAudio

Over the last years, several widely-used desktop Linux distributions adopted PulseAudio [Poettering, 2010] as the default sound system. More recently, PulseAudio became an option to enable software audio routing and mixing in embedded Linux handheld devices [Sarha, 2009], competing with AudioFlinger on Android. An alternative sound server, JACK [Davis, 2003; Phillips, 2006], is predominantly used for professional, low-latency audio production.

PulseAudio is the software layer that controls the audio hardware exposed via the ALSA interface by the Linux kernel. Towards the application layer, PulseAudio offers to connect multiple audio streams to the actual hardware, providing services such as mixing, per-application volume controls, sample format conversion, resampling, et cetera. This allows concurrent use of the audio resources and matches the requirements of the application layer. An important service for hands-free telecommunication systems is acoustic echo and noise reduction (AENR). Since version 1.0, PulseAudio furnishes an echo cancellation framework as a pluggable module. In PA's terms, the echo cancellation (EC) module subsumes AENR. The actual AENR implementations (AENRI) are provided by the Speex library and Andre Adrian's code[8]. With version 2.0, the WebRTC[9] AENRI was introduced and became PulseAudio's default.

The decisive advantage of the sound server architecture is that the responsibility for AENR can be separated from the VoIP application, permitting reuse of the AENR resources by multiple software components and saving duplicate development effort. Furthermore, hardware constraints are hidden from the application: While the audio hardware may only handle interleaved stereo samples in 16-bit signed integers with 48 KHz, the application is actually interested in a mono audio stream represented by single-precision floating-point data sampled at 16 KHz.

So far, the PulseAudio echo-cancellation framework was limited to a symmetric number of channels entering and leaving the AENRI, typically a mono audio stream. However, in an audio setup with an array of microphones, a multi-channel audio stream is processed by the AENRI and generally reduced to mono output, see Fig. 2. The AENRI signal processing pipeline may choose to incorporate sample rate adaption as well, leading to an additional asymmetry of sample data entering and exiting the

[8] `http://www.andreadrian.de/intercom/`

[9] `http://www.webrtc.org`

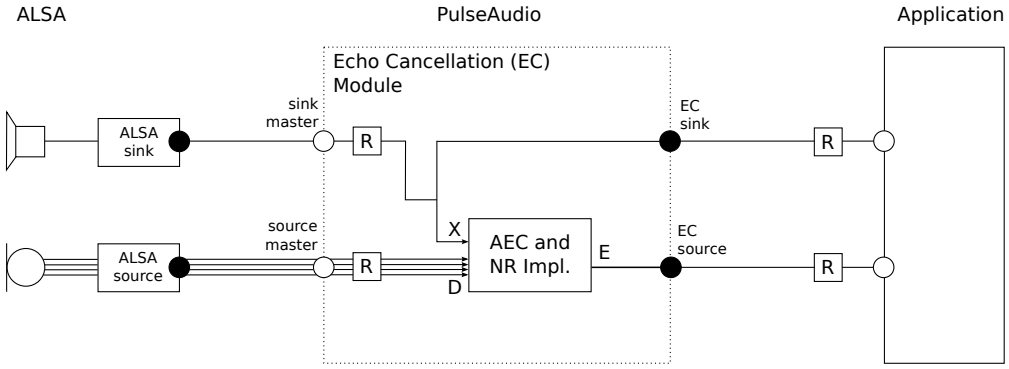

Figure 2: Overview of the PulseAudio sound system providing acoustic echo and noise reduction (AENR) service to an application (with 4 microphone channels).

EC module. A number of patches addressing this issue and related limitations have been submitted during the PulseAudio version 4.0 development cycle.

Fig. 2 shows the PulseAudio sound server in between the ALSA sink/source and the application. Instead of directly connecting to the ALSA sink/source, the application binds to the EC sink/source. Note that the EC module specifies its internal audio sample format and rate, hence resampling stages (denoted by R) may become necessary. Resampling, in PulseAudio's terms, includes sample format conversion, channel remapping, and sample rate conversion as necessary. The modular sound server design brings great flexibility, but efficient implementation of the resampling stages becomes paramount, especially if microphones, AENRI and application layer depend on different sample specifications.

4 Multi-Channel Audio Processing

A multi-channel noise reduction system optimal in the minimum mean square error sense can be factorized in a linearly constrained minimum variance (LCMV) beamformer followed by a single channel postfilter [Wolff and Buck, 2010]. The postfilter is essentially a noise suppressor as explained in chapter 2. Echo suppression can be efficiently combined with noise suppression [Gustafsson et al., 2002].

A beamformer is a spatial filter, i.e., a beam is steered towards a target direction, whereas other directions are suppressed. The basic operation behind linear beamforming is to filter-and-sum the M input signals, i.e., the output F of a filter-and-sum beamformer (FSB) $\boldsymbol{W}$ is

$$F(l,f) = \sum_{m=0}^{M-1} W_m(l,f) D_m(l,f) \qquad (2)$$

where m is the microphone index and $W_m(l,f)$ is the filter weight for the m-th microphone.

A fixed beamformer (FBF) uses fixed weights $\boldsymbol{W}$, that can be precomputed, whereas an adaptive beamformer adapts the weights $W_m(l,f)$ in dependence of the current noise field. The most basic FBF is the delay-sum beamformer (DSB), where $\boldsymbol{W}$ implements pure, frequency independent time delays. The idea is to time-align signals from the target direction. Signals from other directions are to some extent out of phase and cancel partially because of the summation. The DSB exhibits a broad mainlobe of the beampattern at low frequencies and a very narrow mainlobe at high frequencies, i.e., at low frequencies it cannot reduce much noise, whereas at high frequencies little deviation from the target direction causes strong attenuation, leading to a low-pass filtered sound in practical conditions with steering errors. Using filter optimization strategies, better low-end suppression and a wider mainlobe at high frequencies can be achieved [Tashev, 2009]. A FBF can however only be optimal for a certain, given noise-field.

Adaptive beamformers can adapt to changing noise fields and can hence achieve more noise reduction. Still, it is possible to set linear constraints, like distortion-less operation towards the target direction. It can be shown that an adaptive LCMV beamformer can be implemented in the Generalized Sidelobe Canceller (GSC) form that transforms the constrained optimization in an unconstrained one [Souden et al., 2010]. Though formally the same, the GSC has advantages in the implementation and provides an intuitive access to the adaptive beamforming problem, cf. Fig. 3.

The noisy M-channel input is processed by

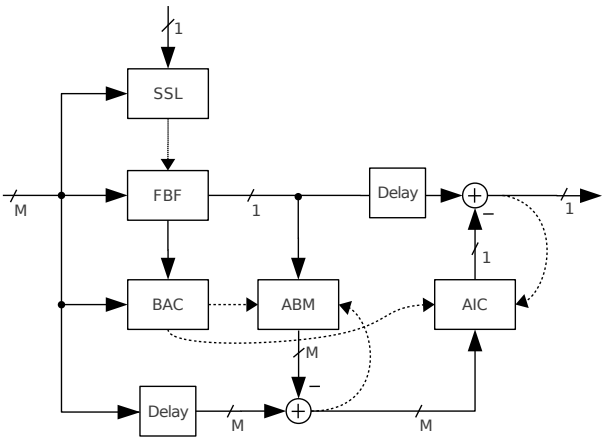

Figure 3: Structure of a Generalized Sidelobe Canceler (GSC) beamformer.

an FBF that keeps the distortion-less constraint towards the target direction. The output of the FBF is further enhanced by subtracting the output of an adaptive interference canceller (AIC). The AIC should be fed with noise-only signals. To this end, the adaptive blocking matrix (ABM) subtracts the target from the noisy microphone signals. The purpose of the beamformer adaption control (BAC) is to guarantee that the AIC is adapted in times of noise-only, whereas the ABM should only be adapted in times of high SNR. The delays are necessary to ensure causality. The FBF needs the target direction as a control input. If the target direction cannot be set to a fixed value, a sound source localization (SSL) algorithm can be used to track the source of interest. SSL is typically based on estimating the direction-dependent time delay of arrival between the individual microphones. In [Souden et al., 2010] a formulation of the GSC is stated, that does not require knowledge of the target direction or the microphone locations, but only the source and noise statistics. This shows the strong link between adaptive beamforming and linear blind source separation.

Our current multichannel AENR system contains a self-steered adaptive beamformer and a postfilter. The latter performs combined echo and noise suppression. A dedicated AEC module has also been developed, but is not yet implemented in C. Combining an AEC with adaptive beamforming promises synergy effects [Herbordt and Kellermann, 2002], i.e., the beamformer can assist the AEC during adaption. Once the AEC is adapted, the beamformer can focus on reducing interfering noise. All processing steps can be done in the frequency domain (FD). To transform a time domain signal block to FD and back, we use the forward and inverse Fast Fourier Transform (FFT), respectively.

```
float v = *(src++) * (1 << 15);
  // load 4 floats from src, increment pointer
  vld1.32      {q0}, [%[src]]!
  // scale by q1 (= 32767)
  vmul.f32     q0, q0, q1

*dst++ = CLAMP(lrintf(v), -0x8000 , 0x7FFF );
  // convert float to 16:16 fixed-point
  vcvt.s32.f32 q0, q0, #16
  // shift right, round, narrow to 16 bit
  // with saturation
  vqrshrn.s32  d0, q0, #16
  // store 4 int16 values, increment pointer
  vst1.16      {d0}, [%[dst]]!
```

Listing 1: Using ARM NEON to convert float to 16-bit integer samples with saturation.

5 Targeting an embedded ARMv7 Cortex-A8 platform

To realize the actual VoIP/intercom application, we build upon the Linux kernel and ALSA for hardware handling and the PulseAudio sound server. Here, we focus on the intermediate component. In order to integrate hardware, AENRI and application, PulseAudio must mediate sample format, sample rate and number of channels at substantial runtime costs. Besides the AENRI, sample rate adaptation is expensive.

The ARMv7 processor architecture is quite power-efficient, yet offers significantly less computational resources than current desktop computers. Especially the Cortex-A8 has weak single-precision floating-point (FP) performance (i.e., most FP instructions take multiple cycles) and requires SIMD-type instructions named NEON [Anderson, 2011] for best performance. Later CPU designs (e.g., Cortex-A9/A15) have improved FP units and performance is less dependent on NEON optimizations. Algorithms in fixed-point arithmetic are more tedious to develop and often have less numeric precision. Hence, we decided to implement all audio signal processing in floating-point arithmetic. On the OMAP3 processor, single-precision FP NEON operations are often executed in a single cycle and are not necessarily slower than equivalent fixed-point/integer instructions.

Resampling is provided by the Speex library

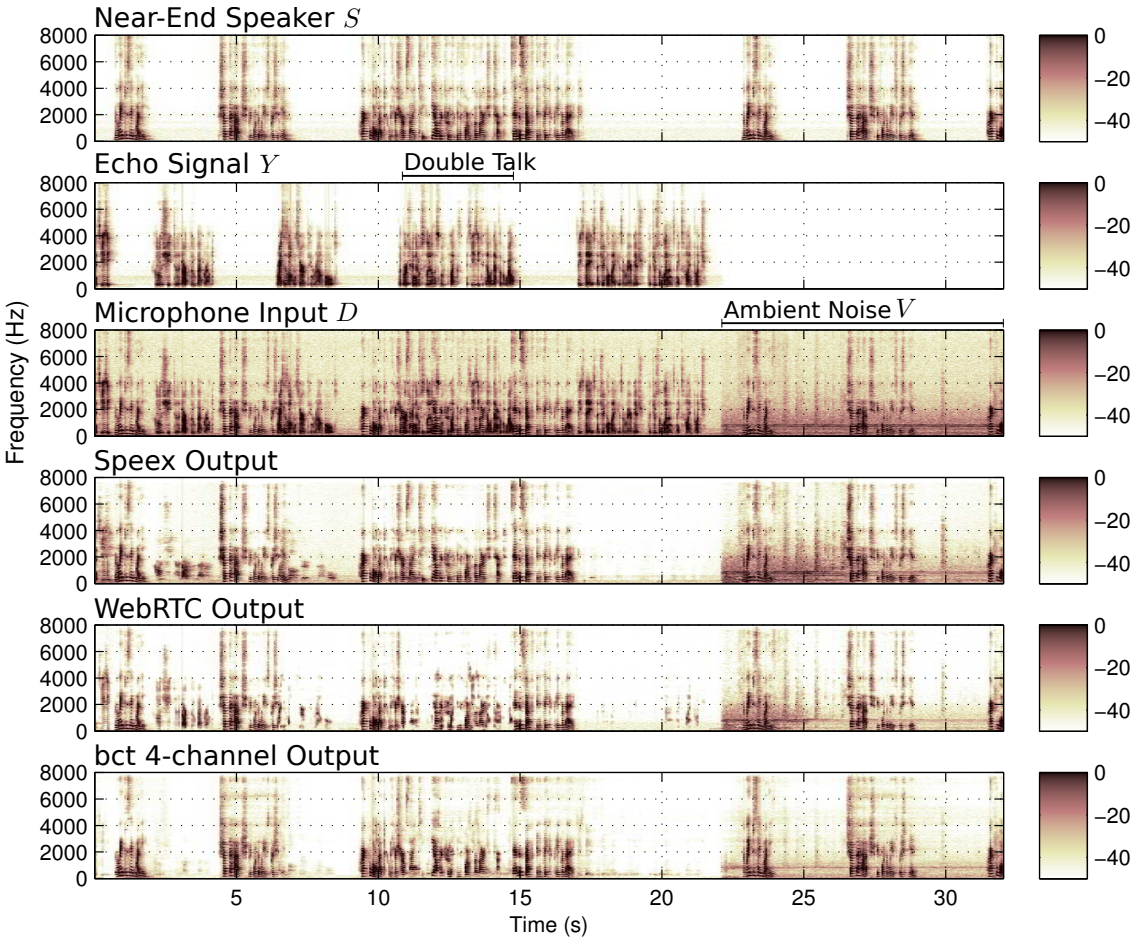

Figure 4: Spectrograms of the test audio signal (top three plots) at 16 KHz and the corresponding output signals (bottom three plots).

for which an ARM NEON patch is available[10]. On the target CPU, the FP implementation is more efficient than fixed-point. Typically, the AENR will be implemented in the frequency domain (FD). To this end, the libav project[11] provides a fast ARM NEON FFT-implementation[12] with a public interface. Listing 1 illustrates how ARM NEON instructions can be used to exploit data parallelism. For the float to 16-bit integer sample conversion operation shown, a speedup of 11× is achieved primarily due to the implicit saturation.

The overall runtime requirements of PulseAudio on the target platform depend on the signal-processing implementation, but to a large part also on the audio latency requirements (set to 50 ms). We observe approximately 25 % CPU load due to PulseAudio providing 4-channel AENR at 16 KHz. Profiling has been performed using the Linux `perf` tool.

6 Test Setup and Experimental Results

Assessing AENR systems is a broad and controversial topic. In our experience, metrics that access speech quality [Loizou, 2011] are often not well suited to describe the behavior and artifacts that occur in complex, real world scenarios. In this work, we rely on spectrogram plots to make an exemplary comparison of different algorithms in a complex scenario with double-talk and noise. We do however believe that listening tests are crucial and need to complement any numerical results.

In order to benchmark the different pluggable AENRIs, PulseAudio's `echo-cancel-test` program is used: it reads raw audio data from a play (denoted signal X) and record (signal D) file and outputs the processed audio data (signal E). All experiments have been performed

[10] `http://blog.gmane.org/gmane.comp.audio.compression.speex.devel/month=20110901`

[11] `http://libav.org`

[12] See `http://pmeerw.net/blog/programming/arm_fft.html` for an informal comparison.

at a sample rate of 16 KHz with PulseAudio 3.0 on a Linux operating system. The GNU compiler in version 4.6 has been invoked with the options `-O2 -ffast-math`. The flags `-march=core2` and `-march=armv7 -mfpu=neon -mfloat-abi=softfp` were used for the x86 64-bit and ARM 32-bit target, respectively.

6.1 Audio Quality

The spectrogram plots in Fig. 4 depict the audio energy in different frequency bands over time (32 seconds; horizontal axis). The audio signals[13] shown are near-end speaker (S), echo signal (Y), microphone input (D) and the output of three AENRIs (Speex, WebRTC, BCT4CH). The Adrian AEC, turned out to not be competitive and completely diverged during double-talk. Therefore, we chose to not devote space to it in our plots.

S and Y are obtained by convolution of speech signals with measured impulse responses H_S and H_X of our device/microphone array in a medium-sized office room. In Fig. 4, only the first channel $m = 0$ (farthest from the loudspeaker) is shown. This channel is also used as an input for the single channel AENRIs Speex and WebRTC. The Cartesian coordinates of the location of microphone m are $\vec{p}_m = [0, p_{m,y}, 0]$, with $p_0 = -0.12$, $p_1 = -0.03$, $p_2 = 0.03$, $p_3 = 0.12$. For measuring H_S, a loudspeaker was placed at $\vec{p}_S \approx [0.5, 0, 0.25]$. We used the exponential sweep method to compute the impulse responses [Holters et al., 2009]. H_X is obtained with the integrated loudspeaker having its acoustic center at $\vec{p}_X \approx [0, 0.1, 0.1]$. In Fig. 4 clearly discernible, alternating speech segments including a period of double talk starting after about 11 seconds can be seen. Before second 22 a recording of the "quiet" office room has been added. After second 22, a broadband ambient noise signal – a recording of a ventilator, placed at $\vec{p}_v \approx [-1.5, 0.3, 0.5]$ – is added to S to compare the noise reduction capabilities of the tested AENRIs. The added noise recordings include the self-noise of the microphones.

Observing the outputs, the echo signal is only partially attenuated in the Speex and WebRTC results during the adaptation (learning) period in the beginning. BCT4CH however delivers echo reduction right from the start and provides good double talk performance. Once adapted, Speex delivers very good double talk performance. This can probably be attributed to its advanced AEC learning rate adjustment[Valin, 2007]. WebRTC, on the other hand, suppresses large portions of the high frequency content of S. Furthermore, WebRTC retains audible echo, see e.g. second 20–22. In other, practical situations WebRTC might however still be preferred to the Speex AENRI, because it employs a more rigorous echo suppression and loss/gain control, which works as a safety guard if nonlinearities or sudden changes of the echo path occur and AEC fails. As outlined in Section 4 BCT4CH does currently not contain an actual AEC module. Knowing this, our good echo reduction performance is even more remarkable. It stems from the superb interference suppression capability of our adaptive beamformer and our high quality postfilter.

Taking a look at the ambient noise scenario at second 22–32 in Fig. 4, all methods are able to reduce noise, however Speex and WebRTC require some time to initially adapt to the new noise characteristics. This clearly show the benefit of the microphone array processing that is less dependent on a stationary noise PSD estimate.

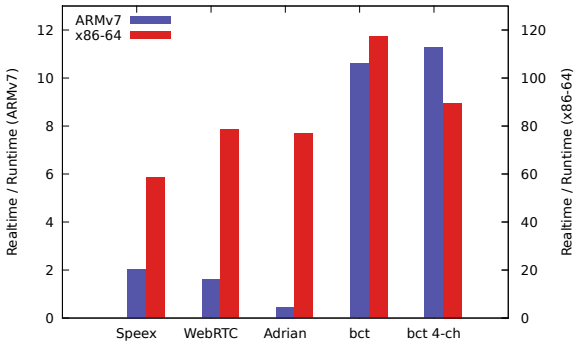

Figure 5: Comparing realtime vs. runtime of several AEC plugins on x86-64 and ARMv7 (higher results are better).

6.2 Runtime

In Fig. 5 we compare the runtime of different AENRIs on an ARMv7 Cortex-A8 platform (TI OMAP3 processor, DM3730, clocked at 800 MHz) and a x86-64 platform (Intel i7-870 clocked at 3 GHz) relative to realtime. Not surprisingly, the embedded platform turns out to be more than 10 times slower than the PC platform. BCT and BCT4CH refer to a single-channel and multi-channel implementation developed by bct electronic. The BCT and BCT4CH code has been optimized and implemented using

[13] Available at `http://bct-electronic.com/lac13/`.

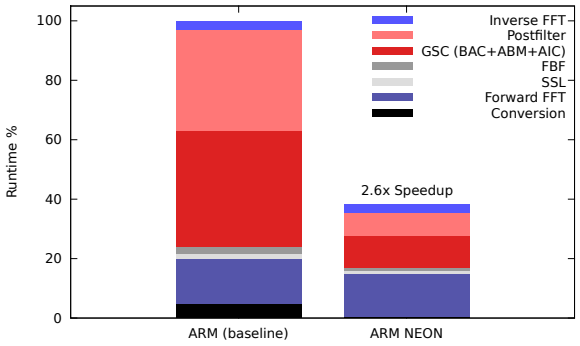

Figure 6: Runtime breakdown and ARM NEON optimization result of the BCT4CH implementation.

the ARM NEON instruction set; they consume approximately 10 % CPU. The other ARMv7 AENRIs lacking optimization compare less favorable with the Intel platform.

Fig. 6 breaks down the runtime of the BCT4CH AENRI according to the processing structure outlined in Section 4. Straightforward optimization of the C/C++ code yields an overall speedup of 2.6×. The runtime contribution in % of the total ARM execution time can be observed: postfilter and GSC are the most expensive execution blocks. The performance of the FFT is not improved as baseline and optimized code both depend on the external libav FFT implementation.

7 Conclusions

We have presented first results of a multi-channel noise/echo reduction solution built on top of PulseAudio and motivated the design decisions. The work has resulted in a number of improvements in the PulseAudio echo cancellation and signal-processing framework, which have been contributed during the version 3.0/4.0 development cycle and should facilitate future embedded Linux audio solutions. Further work includes optimizing code for audio stream mixing, more efficient resampling methods, and the implementation of an efficient AEC in the multi-channel processing pipeline.

References

M. Anderson. 2011. ARM NEON instruction set and why you should care. In *Embedded Linux Conf.'11*, San Francisco, CA, USA.

P. Davis. 2003. The JACK audio connection kit. In *Proc. Linux Audio Conference, LAC'03*, Karlsruhe, Germany.

S. Gustafsson, R. Martin, P. Jax, and P. Vary. 2002. A psychoacoustic approach to combined acoustic echo cancellation and noise reduction. *IEEE Trans. on Speech and Audio Processing*, 10(5):245–256.

E. Hänsler and G. Schmidt. 2004. *Acoustic Echo and Noise Control: A Practical Approach*. Wiley, New York.

W. Herbordt and W. Kellermann. 2002. Frequency-domain integration of acoustic echo cancellation and a generalized sidelobe canceller with improved robustness. *Europ. Trans. on Telecomm.*, 13(2):123–132.

M. Holters, T. Corbach, and U. Zölzer. 2009. Impulse response measurement techniques and their applicability in the real world. In *Proc. 12th Int. Conference on Digital Audio Effects, DAFx'09*, Como, Italy.

Y.A. Huang and J. Benesty. 2012. A multi-frame approach to the frequency-domain single-channel noise reduction problem. *IEEE Trans. on Audio, Speech & Language Processing*, 20(4):1256–1269.

P. Loizou, 2011. *Speech quality assessment*, volume 346, pages 623–654. Springer Verlagq.

D. Phillips. 2006. Knowing Jack. *Linux Magazine*, (67).

Lennart Poettering. 2010. Pro audio is easy, consumer audio is hard. In *Proc. Linux Audio Conference, LAC'10*, Utrecht, Netherlands.

J. Sarha. 2009. Practical experiences from using PulseAudio in embedded handheld device. In *Linux Plumbers Conf.: Audio Miniconf.*, Portland, OR, USA.

M. Souden, J. Benesty, and S. Affes. 2010. On optimal frequency-domain multichannel linear filtering for noise reduction. *IEEE Trans. on Audio, Speech & Language Processing*, 18(2):260–276.

Ivan Tashev. 2009. *Sound Capture and Processing*. Wiley.

J.-M. Valin. 2007. On adjusting the learning rate in frequency domain echo cancellation with double-talk. *IEEE Trans. on Audio, Speech & Language Processing*, 15(3):1030–1034.

T. Wolff and M. Buck. 2010. A generalized view on microphone array postfilters. In *Proc. Int. Workshop on Acoustic Echo and Noise Control, IWAENC'10*, Tel Aviv, Israel.