Whitepaper
Audio Improvements in Android OS

ecom Digital Products and Services

www.ecom-ex.com

Author: Rajan Lal | Marco Kuhn

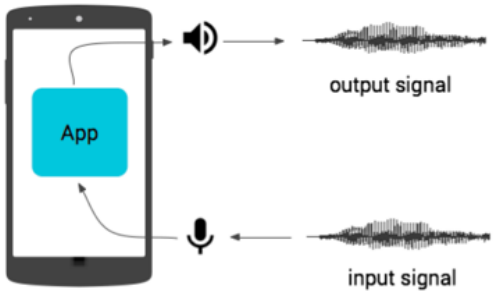
Content

What is Audio Latency?

Latency refers to a short period of delay (usually measured in milliseconds) between when an audio signal enters a system and when it emerges. Potential contributors to latency in an audio system include analog-to-digital conversion, buffering, digital signal processing, transmission time, digital-to-analog conversion and the speed of sound in the transmission medium.

Latency is an important system performance metric. While many kinds of audio latency metrics exist, one useful and well-understood metric is round-trip latency, defined as the time it takes for an audio signal to enter the input of a mobile device, be processed by an app running on the application processor, and exit the output.

Diagram below describes the same:



Why Low Latency is important?

Low audio latency is important for all VOIP applications. Mission critical communication has strict performance requirements for the push-to-talk (PTT) voice calls. The group's calls should be of course established as soon as possible, but the most important thing is that floor control is ‚instant' and mouth-to-ear voice delay is minimal during mission critical PTT calls.

In addition to MCPTT there are other mission critical applications which would need the audio latency to be minimal.

Audio API : Native layer vs JAVA layer

Android SDK provides many classes and API's for the 3rd party developers on the Java layer for audio recording and audio play-out. Audio API's available in 3rd party android SDK provides huge flexibilities to the applications. But this flexibility comes with the cost of latency. The Java layer methods has to call native layer API's which adds on to the processing time.

These Java layer SDK API's are good for general purpose applications which does not have very high latency requirements. When it comes to VOIP or MCC applications, such SDK API's comes with a high cost in terms of time.

Android has thus provided native API's as part of 3rd party NDK (Native Development Kit) which can be used for applications looking for better performance and low latency. Android has also support for OpenSL ES.

OpenSL ES (Open Sound Library for Embedded System) is open source cross platform library for audio. This library is designed especially for the better audio performance by the application on embedded devices. OpenSL ES for Android is an Android-specific implementation of the OpenSL ES API specification from the Khronos Group.

Improvement with AAUDIO

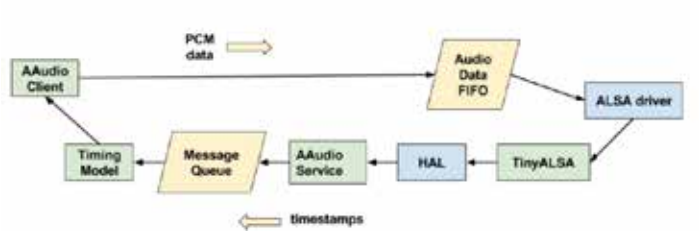
AAudio is a new Android C API introduced in the Android O release. It is designed for high-performance audio applications that require low latency. Apps communicate with AAudio by reading and writing data to streams using the native android API's available in Android NDK library. AAudio is a new native C API that provides an alternative to Open SL ES. It uses a Builder design pattern to create audio streams. AAudio provides a low-latency data path. It supports 2 audio data flow modes:

- EXLUSIVE mode
- SHARED mode

In EXCLUSIVE mode, the feature allows client application code to write directly into a memory mapped buffer that is shared with the ALSA driver. In SHARED mode, the MMAP buffer is used by a mixer running in the AudioServer. In EXCLUSIVE mode, the latency is significantly less because the data bypasses the mixer.

In EXCLUSIVE mode, the service requests the MMAP buffer from the HAL and manages the resources. The MMAP buffer is running in NOIRQ mode, so there are no shared read/write counters to manage access to the buffer. Instead, the client maintains a timing model of the hardware and predicts when the buffer will be read.

In the diagram below, we can see the Pulse-code modulation (PCM) data flowing down through the MMAP FIFO into the ALSA driver. Timestamps are periodically requested by the AAudio service and then passed up to the client's timing model through an atomic message queue.



In SHARED mode, a timing model is also used, but it lives in the AAudioService. For audio capture, a similar model is used, but the PCM data flows in the opposite direction.

Audio subsystem changes

AAudio requires an additional data path at the audio front end of the audio subsystem so it can operate in parallel with the original AudioFlinger path. That legacy path is used for all other system sounds and application sounds. This functionality could be provided by a software mixer in a DSP or a hardware mixer in the SOC. Devices which supports low latency has the hardware feature defined:

android.hardware.audio.low_latency

Application using AAUDIO

3rd party application developers should consider using the open source Oboe library. Oboe is a C++ wrapper that provides an API that closely resembles AAudio. It calls AAudio when it is available, and falls back to OpenSL ES if AAudio is not available. NDK API developers guide provides complete guide on using aaudio API's:

https://developer.android.com/ndk/guides/audio/aaudio/aaudio

Comparison of Audio Latency on ECOM Saiph device

We used 2 sample applications each using native library (Open SL and AAudio) to measure the audio latency on Saiph device. Player data call back method is used to estimate the latency.
Device and audio parameters used

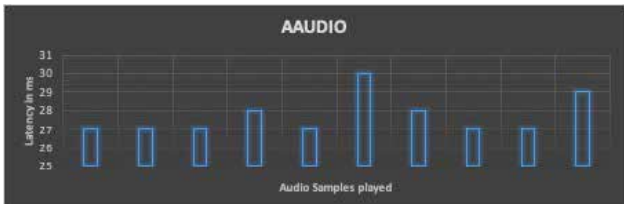
- Model: Ex-Handy 10 ROW
- Android OS Version: 8.1.0
- Build number: H10ROW.EC.01.01.025.00
- Sample Rate: 48000 Hz
- Buffer Size: 240

Using OpenSL ES Android Library

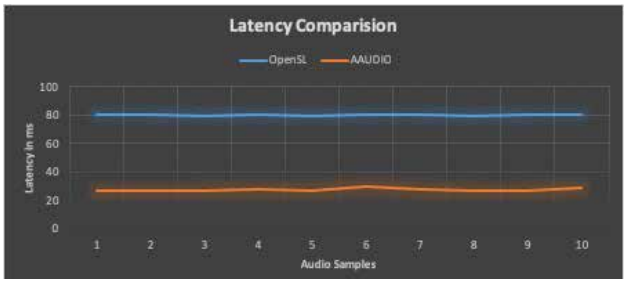
Bar Graph on Audio sample played vs the time
(Calculated Latency ~80ms)

Using AAudio Android Library

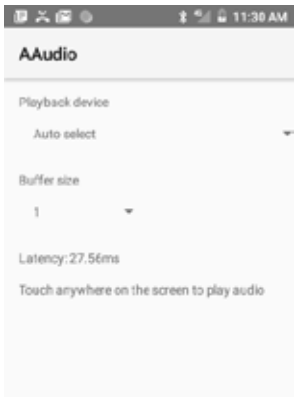
Bar Graph on Audio sample played vs the time
(Calculated Latency ~28ms)



Latency Comparison chart



Ex-Handy 10 images using the test applications



AAUDIO test Application

OpenSL test Application

Sample Applications

Here are link for some of the sample applications using Android™ Oreo and the native audio API's:

https://github.com/google/oboe/tree/master/samples/hello-oboe

https://github.com/googlesamples/android-ndk/tree/master/native-audio

https://github.com/googlesamples/android-audio-high-performance/tree/master/aaudio

Your automation, our passion.

Explosion Protection

- Intrinsic Safety Barriers
- Signal Conditioners
- FieldConnex® Fieldbus
- Remote I/O Systems
- Electrical Ex Equipment
- Purge and Pressurization
- Industrial HMI
- Mobile Computing and Communications
- HART Interface Solutions
- Surge Protection
- Wireless Solutions
- Level Measurement

Industrial Sensors

- Proximity Sensors
- Photoelectric Sensors
- Industrial Vision
- Ultrasonic Sensors
- Rotary Encoders
- Positioning Systems
- Inclination and Acceleration Sensors
- Fieldbus Modules
- AS-Interface
- Identification Systems
- Displays and Signal Processing
- Connectivity

Pepperl+Fuchs Quality
Download our latest policy here:

www.pepperl-fuchs.com/quality

PEPPERL+FUCHS

www.pepperl-fuchs.com

Author: Rajan Lal | Marco Kuhn / 05|2019