iOS Concept for SoniControl 2.0

SoniControl[1] is a novel technology for the recognition and masking of acoustic tracking information. The technology helps end-users to protect their privacy. Technologies like Google Nearby and Silverpush build upon ultrasonic sounds to exchange information. More and more of our devices communicate via this inaudible communication channel. Every device with a microphone and a speaker is able to send and receive ultrasonic information. The user is usually not aware of this inaudible and hidden data transfer. To overcome this gap SoniControl detects ultrasonic activity, notifies the user and blocks the information on demand. Thereby, we want to raise the awareness for this novel technology.

The project SoniControl is funded by Netidee (www.netidee.at) and developed at the Media Computing Group of the Institute of Creative Media Technologies at Sankt Pölten University of Applied Sciences (mc.fhstp.ac.at). You can find all published results and resources for the SoniControl project on our website: sonicontrol.fhstp.ac.at.

funded by:

Table of Contents

[1] https://sonicontrol.fhstp.ac.at/

1. License

This document is released under the CC BY-SA 3.0[2] license.

2. Motivation

We are currently evaluating the possibility to port our Android app SoniControl[3] to iOS. Indeed, many people contacted us in the past and asked if an iOS version of SoniControl is available. There is a large demand from the iOS user community and we would like to provide them with a solution. This document aims at answering the following questions:

- Is it technically possible to create an iOS version of SoniControl?
- Would an iOS version of SoniControl be allowed on the Apple App Store?

3. Architecture

The general architecture of SoniControl is described in the documentation available on our website. It is especially recommended reading the Developer Documentation[4] for implementation details. In order to port SoniControl to iOS, most of the components will need to be written again in a language natively supported by the platform (e.g. Swift), or using a cross platform solution. Most library currently used are Java based and will have to be replaced with an iOS equivalent. The core functionality of SoniControl is, however, based on the Superpowered cross platform library and should only require limited modifications. Here are the main components of the current SoniControl (Android) app and how to get started porting them to iOS:

3.1. Detection algorithm

The core functionality of SoniControl, the detector, is written in C++ and based on the Superpowered library[5]. In order to reuse this code, we will need an Objective-C++ file (a file that can mix Objective-C and C++ code simply by changing the extension to .mm). This kind

[2] https://creativecommons.org/licenses/by-sa/3.0/
[3] https://play.google.com/store/apps/details?id=at.ac.fhstp.sonicontrol
[4] https://sonicontrol.fhstp.ac.at/wp-content/uploads/2018/03/sonicontrol_developer_doc_and-architecture_public.pdf
[5] https://superpowered.com/

of file is used in the code samples from Superpowered[6]. It would then also be possible to interact with this core component from an app written in Swift, using a bridge header.

3.2. Concurrency (Threads)

In SoniControl, we submit tasks to a thread pool to, for instance, start the detector in the background. This archive[7] leads through ways to asynchronously execute code on iOS. It is explicitly recommended not to use threads directly, but instead the Grand Central Dispatch (GCD) or operation queues. This is similar to the Android equivalent, so porting the app to iOS should not require fundamental changes to the concurrency handling, only an adaptation to language specific requirements.

3.3. Jammer

The Jammer actively disturbs and blocks ultrasonic communication around the device by playing white noise at the frequencies used to communicate. It mostly requires playing audio, which should be little development effort using either the AVAudioPlayer[8] or the Audio Queue Services[9].

3.4. Noise generator

The noise generator produces the white noise to be used by the Jammer in order to block ultrasonic communication. It is based on a Java FFT library. As iOS natively includes an FFT library as part of Accelerate[10], the porting to iOS should not present a major issue. Another option would be to use Superpowered to implement this component.

3.5. Signal recognition

After the signal is detected, we analyze it to recognize the technology (most likely) used and inform the user about it. This is based on Java FFT and Windowing libraries. As iOS natively includes Digital Signal Processing libraries as part of Accelerate, the porting to iOS should not present a major issue. Another option would be to use Superpowered to implement this component.

[6] https://github.com/superpoweredSDK/Low-Latency-Android-iOS-Linux-Windows-tvOS-macOS-Interactive-Audio-Platform/blob/master/Examples_iOS/SuperpoweredFrequencyDomain/SuperpoweredFrequencyDomain/ViewController.mm

[7] https://developer.apple.com/library/archive/documentation/General/Conceptual/ConcurrencyProgrammingGuide/ConcurrencyandApplicationDesign/ConcurrencyandApplicationDesign.html#//apple_ref/doc/uid/TP40008091-CH100-SW1

[8] https://developer.apple.com/documentation/avfoundation/avaudioplayer

[9] https://developer.apple.com/documentation/audiotoolbox/audio_queue_services

[10] https://developer.apple.com/documentation/accelerate

3.6. Spectrogram visualization

After the signal is detected, we compute the corresponding spectrum for the frequencies above 17kHz and show it as a spectrogram to the user. This is based on Java FFT, Windowing, and graphics libraries. As iOS natively includes Digital Signal Processing (DSP) as part of Accelerate and 2D rendering via its Core Graphics framework, porting the spectrogram visualization to iOS should not present a major issue. Another option would be to use Superpowered to implement the DSP part of this component.

3.7. Sonification

After the signal is detected, we offer the user to replay it at a lower frequency (currently one octave lower, so dividing the frequency by two). This is based on the Superpowered library and should work with minor modifications (to import the file as Objective-C++).

3.8. Map visualization

We offer a map visualization of the firewall rules saved by the user. It shows one marker per rule, color coded as follows: green for "ignore", orange for "ask next time", and red for "block". When several markers are located too close to each other to be distinguishable, we group them in a blue circle with the number of grouped detections. This map is based on the Java OSMDroid library. Most equivalent options for iOS do not seem to be maintained (e.g. route-me[11] or MapKit[12]) but LibOSMScout[13] might be an option, even though it is intended to be used offline. The data about the rules are stored in a JSON file which can be read natively by Objective C and Swift, the code only needs to be ported to the language chosen.

3.9. GPSTracker Geocoder

When we show the detection history or rules list, we replace the coordinates (place of detection) with a user friendly location description, like the corresponding address if there is one. This feature is called (reverse) geocoding and is available natively in the iOS framework.

3.10. Sharing detections

Users get the option to share their detections with the community anonymously. This will help inform our users and the public audience about the spread of applications and companies using ultrasonic communication. On SoniControl's app side, this means we need a way to make REST calls. On Android we used the Retrofit library to turn the HTTP requests into objects methods, this is optional. On iOS, we could either use an equivalent library, like Alamofire[14], or send bare HTTP requests like recommended in the documentation to upload

[11] https://github.com/route-me/route-me
[12] https://github.com/mattrajca/MapKit
[13] https://github.com/Framstag/libosmscout
[14] https://github.com/Alamofire/Alamofire

data[15] : "*use a URLSession instance to create a URLSessionUploadTask instance. The upload task uses a URLRequest instance that details how the upload is to be performed.*"

3.11. Writing WAV file

When the user decides to share a detection, a short audio sample of the signal (everything under 17kHz is removed) will be written to a WAV file and uploaded to the SoniControl server. No user ID is stored, and thus the data is anonymous. This is based on Java FFT and Windowing libraries. As iOS natively includes Digital Signal Processing libraries as part of Accelerate, the porting to iOS should not present a major issue. Another option would be to use Superpowered to implement this component.

3.12. Background audio capturing

A key requirement in order to port SoniControl to iOS is the possibility to let the firewall run in the background. On Android, such a feature is achieved by using a foreground service, keeping the audio processing alive and signalling it to the user via an infinite notification. On iOS, to keep running in the background, apps must request a **background mode permission** corresponding to what they are doing (e.g. audio, voip, …). However, apps requesting this permission are often rejected during app review, and it is unclear if our use case would be accepted. This section is a summary of our main findings.

3.12.1. Official documentation

The current documentation on background modes[16] does not specify which use cases are allowed or not. The review guidelines[17] do not specify which use cases are allowed or not. In general, it is hard to find up-to-date documentation on background audio: Many people link to documentation that is not available anymore. In this archive[18], (audio) "recording apps" are described as a valid use case for the audio background mode. In this archive[19], the audio background mode is described as:

- "The app plays audible content in the background."
- "Additionally, all watchOS 4 apps can record audio in the background. You must start recording in the foreground, but recording continues when the app transitions to the background."

SoniControl does not **play** content in the background. It starts **recording** in the foreground on user action, and continues in the background.

[15]https://developer.apple.com/documentation/foundation/url_loading_system/uploading_data_to_a_website
[16]https://developer.apple.com/documentation/bundleresources/information_property_list/uibackgroundmodes
[17]https://developer.apple.com/app-store/review/guidelines/
[18]https://developer.apple.com/library/archive/documentation/Audio/Conceptual/AudioSessionProgrammingGuide/AudioGuidelinesByAppType/AudioGuidelinesByAppType.html
[19]https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/iPhoneOSKeys.html#//apple_ref/doc/uid/TP40009252-SW22

3.12.2. Other apps using background audio

Given that the documentation does not clearly state which usages are allowed, we decided to look for existing apps using audio in the background. For instance, the Voice Memo app from Apple does work in the background (and shows a red status bar to alert the user about the microphone usage). Here are some other examples:

- In July 2018, recording audio in the background seemed to be a valid use case according to a standard user citing an older documentation (whose link is broken): https://forums.developer.apple.com/message/323260#323260
- In 2016, an app got rejected for doing audio processing in the background trying to detect something (Shazam like): https://stackoverflow.com/questions/35948665/ios-audio-background-mode
  Especially, one person wrote that the Apple reviewer said: recording is not an appropriate use of Background Mode.

A commonly cited rejection reason is:

"2.16 Details
Your app declares support for audio in the UIBackgroundModes key in your Info.plist but did not include features that require persistent audio.
Next Steps
The audio key is intended for use by applications that provide audible content to the user while in the background, such as music player or streaming audio applications. Please revise your app to provide audible content to the user while the app is in the background or remove the "audio" setting from the UIBackgroundModes key."

Note: in SoniControl the case is that we play *inaudible* content in the background for the protection of the user. It remains an open question how this will be evaluated and seen by the reviewers.

3.12.3. Apple Developer Forums

We tried to clarify the situation by asking on the Apple Developer Forums[20] if our use case could be accepted on the App Store, and got the answer that: "*Only App Review can give you a definitive answer to questions as to what will or won't be allowed on the store.*"

On the positive side: "*At a technical level, the approach you're suggesting (starting a record session in the foreground and continuing it in the background) should work just fine.*"

What will not work on iOS is the restart of recording from the background (happens after we blocked a signal, we then restart scanning automatically, which is not possible since iOS 12.4). We will have to redesign our workflow to start recording again in the foreground, e.g.:

- notifying the user that blocking is over and scanning should be restarted
- or continuously recording and discarding input while jamming to avoid having to restart recording after blocking

[20] https://forums.developer.apple.com/message/386388#386388

4. Conclusion

4.1. Is it technically possible to create an iOS version of SoniControl?

It seems technically feasible to implement the core functionality of SoniControl on iOS: a scanner detecting ultrasonic communication and offering to block them. What cannot work and will be redesigned is the restart of recording from the background (happens after we blocked a signal, we then restart scanning automatically, which is not possible since iOS 12.4. See the previous section on Apple Developer Forums for workarounds and the relevant forum thread[21]).

4.2. Would an iOS version of SoniControl be allowed on the Apple App Store?

Only the app review team can answer this question. It is clear that most of the functionality is going to pass the review without problems (e.g. scanning in the foreground, blocking in the foreground). What might be rejected during review is the permission to keep recording audio in the background. The critical part is that the Audio Background Mode definition changed over time and that the current documentation does not specify anymore which use cases are allowed or not. We thus cannot conclusively state without uncertainty that SoniControl would be allowed on the Apple App Store.

4.3. Potential limitations and solutions

If Apple would refuse our app to use the audio background mode, we would have to limit the usage of SoniControl to an on-demand scanner running only in the foreground. This might limit its ability to detect ultrasonic communication. Indeed, SoniControl first needs to build a model of the ambient noise (background model) before being able to detect anomalies. If the user suspects ultrasonic communication to be used in a place, the ambient noise most likely will include the ultrasound signal, making it look “normal” to our detector. In this situation, one should try to start scanning in an environment assumed to be exempt of ultrasound.

We also realized that on iOS (tested on iOS 13.1.1), recording does not block the microphone for other apps. It is possible to record a memo while e.g. display a spectrogram of the current noise. That means our “soft” blocking option cannot be used on iOS and users will have to actively block signals if they want to be protected.

As mentioned, another component that we will have to redesign is the transition from blocking to scanning as we cannot restart recording from the background. For instance, we could:

- notify the user that blocking is over and scanning should be restarted
- record continuously to avoid having to restart recording, and discard input while blocking

[21] https://forums.developer.apple.com/message/386388#386388