Microphone capture in the web browser

STTS Södermalms talteknologiservice
Folkungagatan 122 2tr, 116 30 Stockholm
http://stts.se

May 14, 2020

1 Introduction

In this document, we describe the background for some demo applications using the web audio API for microphone recording in the browser. Different methods for recording in the browser and sending the resulting audio to a server are listed below.

There is audio processing built into the browser, and there is a JavaScript web audio API (but the audio processing itself is probably not implemented in JavaScript).

There are different ways to capture audio in the browser, and different ways to transporting the audio to a server.

This document is accompanied by some example applications at github.com/stts-se/tillstudpub. These consist of small client-server examples, run in the browser, from wich the audio is sent to a server and saved to disk. All examples work in recent versions of Chrome, but may not work in other browsers.

Precompiled versions for Linux and Windows are found at github.com/stts-se/tillstudpub/releases.

1. `promptrec` presents the user with text to read aloud. The recordings are saved on the server.

2. `audio_streaming` streams microphone audio from the browser to a server. There are two competing methods, one using `ScriptProcessorNode`, the other using `AudioWorklet`, a more recent approch.

3. `webrtc_demo` is a sketchy example of using the WebRTC protocol for streaming audio to a server. Not a fully functioning application, and based on an example of the Pion WebRTC implementation. See pion.ly and github.com/pion.

2 Recording a complete file before sending to the server

One way to do microphone recording in a web browser is using the JavaScript Web Audio API to create an audio file in the browser. You can obtain an audio stream from the microphone using the `getUserMedia()` function. Using a `MediaRecorder`, the audio can be saved as a `Blob`, "a file-like object of immutable, raw data" [9]. When the recording is done, the `Blob` can be added to an HTML5 audio element, that can be played using the web browser.

The audio file can be sent to a HTTP server (typically as a base64 encoded string), and saved on the server.

The audio files might be of different encodings, depending on the browser, so you need to pass his information on to the server (unless you convert to a common format in the browser, but this may be non-trivial).

This method works when the user wants to record something, and only when done send it to the server. This could, for example, be a recording tool, where you read aloud and record a manuscript sentence presented in the browser, such as the current `promprec` demo.

However, this method is not useful for streaming audio to the server.

Pros

- The audio format is known
- A complete audio file is created before sending it over the network
- No built-in packet/data loss (compared to streaming) [TODO???]

Cons

- The audio format cannot be changed (it is controlled by the browser)
- Not for streaming

3 Streaming using ScriptProcessorNode

The ScriptProcessorNode[2] was introduced to meet developers' need to write custom code for audio processing in the Web Audio API. For performance, most processing components of the Web Audio API are run in a separate thread, but the ScriptProcessorNode is run in the main thread, which can cause delays. It has since been deprecated. More information on the motivation behind the move to AudioWorklet can be found in [4].

In the example application, the audio data are converted from floats to 16 bit integers in the browser.

Pros

- Supported by most browsers
- ...

Cons

- Can have performance and quality issues
- Deprecated

4 Streaming using AudioWorklet

The AudioWorklet was developed to handle some design flaws in the ScriptProcessorNode. The first implementation of the AudioWorklet was released 2018 for Chrome [4].

Unfortunately, the AudioWorklet is yet not fully supported by Firefox, but according to the documentation, it should work with Firefox upcoming version 76. We have it tested and working with Google Chrome (v81) and Opera (v68).

In the example application, the audio data are converted from floats to 16 bit integers in the browser.

Pros

- Processing run in a separate thread
- Less latency than ScriptProcessorNode

Cons

- Not supported by all browsers (see [3])
- Packet/data loss can occur [TODO???]

5 Streaming over a websocket

A websocket can be thought of as a bi-directional HTTP connection. Usually, when a client calls an HTTP server, there is a request from the client, and a response from the server, and that's it. Subsequent calls must establish new connections to the server.

A websocket, on the other hand, is an HTTP connection that may stay open for any period of time, and on which both the client and the server may send data. Since a websocket is an upgraded HTTP connection, is has much of the same characteristics, such as guarantees against package loss, etc.

After a websocket connection has been established, the client and the server may send text or binary data to each other in any form.

A difference between a websocket and a normal HTTP connection, is that there is not a well defined protocol for interaction between client and server (such as GET and POST, etc). You have to create your own protocol.

6 Streaming using WebRTC

WebRTC (Web Real-Time Communication) [6] is a peer-to-peer method for streaming audio and video. In other words, you can use it to stream audio and video directly between web-browsers (as long as there is some way of the browsers to find each other, for example using a STUN server).

WebRTC is build on UDP (rather than HTTP). The WebRCT uses the Opus codec for audio (and V8 for video).

Since human cognition is more forgiving to missing samples than to latency, as little lag as possible is more important than being sure that the original sound wave is complete and intact — packet loss is tolerated. The original sound wave might not be possible to reproduce on the other end. Since speech is full of redundant information and human cognition is built for interpreting a noisy signal, this may not be a big issue for understanding speech sent over a bad network using WebRTC.

Under the hood, WebRTC takes care of things like echo-canceling and noise reduction.

WebRTC also includes a DataChannel API, that can be used for transporting data without packet loss. However, a websocket may do the job equally good, if you do not need peer-to-peer capabilities.

Pros

- Supported by most browsers [7]

Cons

- Risk of packet loss
- Little risk of latency

7 Browser settings for audio and streaming

There seems to be some settings in the browser(s), that are difficult to control. Examples:

Sample rate Can be retrived but not changed

Channel count We can set this value, but are not sure how it's used. The wav library currently used on the server side only supports mono.

Audio encoding

AudioWorklet/ScriptProcessorNode For streaming, a browser's recording is by definition 32bit PCM. We have chosen to convert to 16bit PCM before sending to server.

WebRTC OPUS

Non-streaming The browser controls the encoding, but the value can be retreived.

References

[1] *High Performance Browser Networking*, Ilya Grigorik, O'Reilly, 2013

[2] https://developer.mozilla.org/en-US/docs/Web/API/ScriptProcessorNode

[3] https://developer.mozilla.org/en-US/docs/Web/API/AudioWorklet

[4] *AudioWorklet: The future of web audio*, Hongchan Choi, Google Chrome, ICMC, 2018, https://hoch.io/media/icmc-2018-choi-audioworklet.pdf

[5] Boris Smus, *Web Audio API*, O'Reilly, 2013

[6] https://webrtc.org/

[7] https://en.wikipedia.org/wiki/WebRTC#Support

[8] *Investigation into low latency live video streaming performance of WebRTC*, Jakob Tidestrï¿½m, In degree project computer science and engineering, second cycle, Stockholm, Sweden, 2019, http://www.diva-portal.se/smash/get/diva2:1304486/FULLTEXT01.pdf

[9] https://developer.mozilla.org/en-US/docs/Web/API/Blob