Streaming live content to web audio API

Raphaël Goldwaser
France Television
7, esplanade Henri de France
75015 Paris, France
raphael.goldwaser@francetv.fr

Emmanuel Freard
Vi-live
26 Boulevard Jules Ferry
75011 Paris, France
e.freard@vi-live.fr

ABSTRACT
Web Audio API helps to manage sound through a web browser. In most cases, the input is a sound file, fully loaded from a server. Stored in the cache of the browser, it is then transformed using **Web Audio API**.

But we can also want to work with segments of a file. For instance, when streaming live data, Web have to deal with a dataset of undetermined length.

Categories and Subject Descriptors
H.5.1: [Multimedia Information Systems] Audio input/output

General Terms
Experimentation, Standardization

Keywords
Live streaming, audio streaming, real-time processing

1. LIVE CONCERTS
During a live concert, our solution allows the customer to choose his listening position. One can then decide to stand in the crowd, but also to hear the concert from the singer's standpoint, from the bass player, or from backstage. With a headset, the client will enjoy a binaural listening, hence creating a feeling of space. The various instrumental tracks will be sent to a multitrack flow in case s/he wishes want to work on instruments in separatly.

2. TECHNICAL CONSTRAINTS
The system require a live streaming system "from one to many" that can be multi-flow. It also requires a sounds system that allows audio-digital analysis on the client side. The user should though be able to use our technology without having to set up any third-party application, and from a vast array of devices (desktop PC, smartphone, tablet). Web browsers and common web technologies are then an optimal solution, in order to achieve maximum availability and stability of time.

3. STATE OF THE ART
Adobe Flash is historical and the most widely used solution to collect and to analyze live audio streams in a browser. We nonetheless did not retain this technology because it was not available on most mobile and tablet devices. Moreover, this technology is proprietary and it does not match the standards of the web.

Audio analysis, server based solutions like Sox, GStreamer, FFMPEG have not been retained either. Since each session is potentially different (it depends on the interactions of the user); hence it requires its own flow and analysis. Moreover, the server resources that will be needed in the process are hard to estimate with such a technology. Finally, this solution requires much more server resources than a standard flow diffusion.

4. TECHNOLOGICAL CHOICES

4.1 Audio processing
Web Audio API is a native browser API and a web standard.

4.2 Streaming

4.2.1 Streaming format
Before beginning to expose what was done in this demonstration, first we would like to explain the limits to adaptive streaming formats.

For this project, we needed to use 'adaptive streaming format' in order to offer the best quality experience to the end user. Adaptive streaming (also know as ABR streaming), is today's standard to deliver video to the end user by detecting a user's Bandwidth/CPU capacity in real time and adjusting the quality of the video stream accordingly (figure 1) :

[Figure 1 : Adaptative streaming]

The current ABR formats used in the industry are :

- Smooth Streaming (Microsoft)
- HTTP Live Streaming (Apple)
- HTTP Dynamic Streaming (Adobe)
- MPEG DASH (ISO) formats.

In order to stick with our open source philosophy, we didn't want to use proprietary formats. MPEG DASH seems the promising streaming format for the coming years and therefore our preferred choice for this demo. The various reasons we used DASH are [1]:

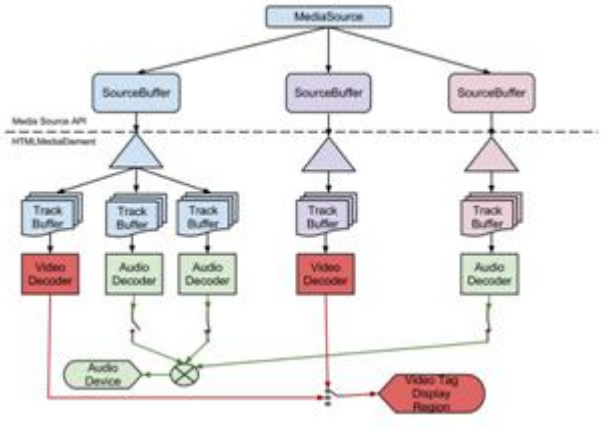
- DASH is a streaming standard based on ISO based media File Standard (ISO/IEC 23009-1:2014), this means no need for proprietary packager: FLV, ISMV, MOV...
- DASH is Codec agnostic, so we can use h.264 or h.265 in the coming years
- DASH is Protocol agnostic: UDP, HTTP, RTMP, multicast are all supported
- DASH offers the possibility of multiple representations (video, audio and metadata)

4.2.2 Packager

We needed to be fully independent to package any demo DASH stream and again we wanted to use Open Source tools for this project. The open source project GPAC was the perfect solution to package the MPEG DASH stream. Indeed, MP4Box[2] can be used to generate content conformant to the MPEG-DASH specification, aka ISO/IEC 23009-1 (available in ISO Publicly Available Standards[3]).

4.2.3 Player

Once we chose the streaming format, we had to find a player solution that would be compatible with the WebAudioAPI. So, we needed to be compatible with W3C standards. We knew that the use of Media Source Extensions would be necessary to avoid using a plugin in our demo application. MSE provides a programmatic interface to the HTML video tag so developpers could build robust streaming applications using only HTML and Javascript. There is no more need for Flash/Silverligh plugins.



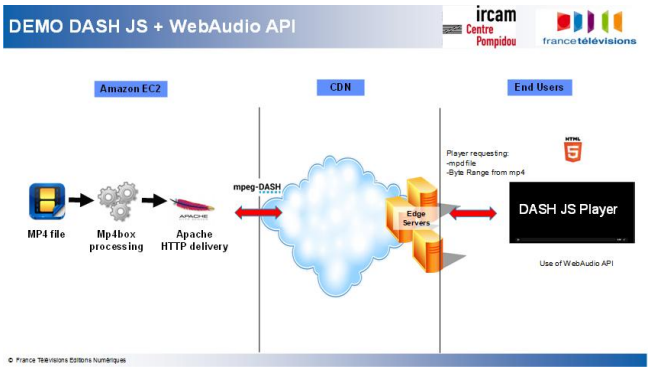
[Figure 2 : How work Media Source API from [4]]

DASH JS[5] is the only DASH player project that answers to our needs. The Dash.js is an initiative to establish a production quality framework for building video and audio players that play back MPEG-DASH content using client-side JavaScript libraries leveraging the Media Source Extensions API set as defined by the W3C. The core objectives of this project are to build an open source Javascript library to playback DASH Streams.

Once the source code forked from github, we could modify the player to plug DASH JS with the WebAudio API.

4.2.4 Delivery

The delivery workflow is quite simple as we used Amazon EC2 in order to store and deliver our content for the demo. Here's the workflow (Figure 3) :



[Figure 3 : Delivery workflow]

- Packaging[6] :

Single Track:

```
MP4Box -dash 10000 -frag 1000 -rap -single-segment INPUT_FILE.mp4
```

- HTTP Delivery :

In order to use DASH JS, we need to enable CORS:

```
Header add Access-Control-Allow-Origin "*"
Header add Access-Control-Allow-Headers "origin, x-requested-with, content-type, range, accept"
Header add Access-Control-Allow-Methods "PUT, GET, POST, DELETE, OPTIONS"
Header set Accept-Ranges bytes
```

5. EXPERIMENTATION

We focused on various ways of sending data to a Web Audio API in order to collect and manage the audio buffer.

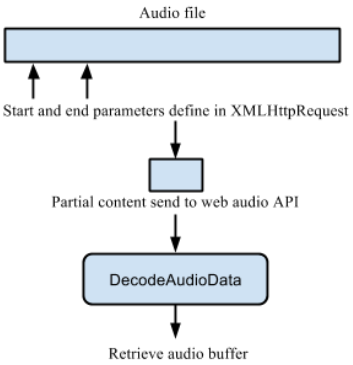
5.1 Web Audio API only

5.1.1 Device

We first tried to test a simple data collection device that sends an XMLHttpRequest without resorting to any streaming system

An audio file is stored on a server and we make an XMLHttpRequest. We define an interval that locates the beginning and the end of a specific section in the audio file we want to analyze.

These data are in turn passed on to the Web Audio API, using a decodeAudioData (Figure 4).



[Figure 4 : Explanatory diagram of device 5.1.1]

5.1.2 Results

It turns out that this solution works in certain instances only. When collecting raw data (e.g. a .WAV file), the decoding function works and allows the analysis of audio segments, even though the whole file has not been properly loaded.

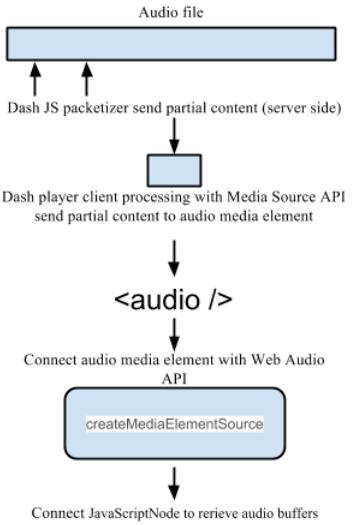
But audio files containing compressed data (MP3, AAC, etc.) cannot but loaded in such a way. Web audio API can only decode theses types of files if the file is fully loaded on the server.

5.2 Web Audio API and Media Source API

5.2.1 Device

This API is used by the audio/video reader DashJS. The methods used by the Media Source API allow to create a data buffer but also to add data to it « appendBuffer » while loading. The buffer is injected to an element in the DOM, the <audio> tag (Figure 5).

As seen previously, the most complex aspect when loading data continuously lies in the decoding. We then had to determine if Media Source API had this ability or if the decoding had to be carried out once the data are injected in the HTML5 tag.



[Figure 5 : Explanatory diagram of device 5.2.1]

5.2.2 Results

Connecting the <audio> of the DOM seems to be the only possible way to collect continuously audio data while treating it live at the same time. We do that using Web Audio API.

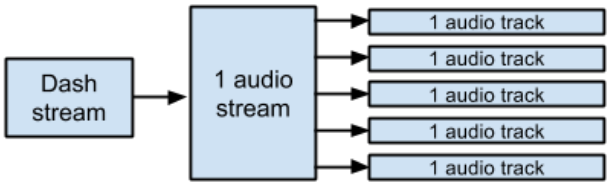
Once the connection has been established between the Web Audio API and the audio tag in the browser, it becomes possible to collect all the data collected in the <audio> tag. Any type of treatment becomes possible then.
Proof of concept can be downloaded here :
https://github.com/nums/DashJs-WebAudioAPI
This solution has limitations nonetheless.
- We have to use a JavaScriptNode, which leads to an added CPU to the browser.
- The sound restitution is not based on the initial data. It must decode the <audio> tag, which creates another layer.
- We inherit the specificities of the <audio> tag
These aspects have been addressed more extensively here [7]

Other experimentation presents a similar project that goes into binaural treatment in depth [11].

5.3 MultiTrack

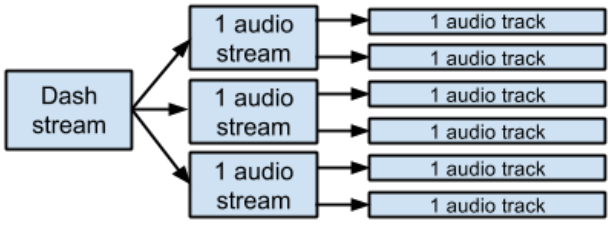
In our use case, we have consider multitrack flow.

It is possible to encapsulate multiple audio tracks in the same stream, like the stereo or Dolby (Figure 5).



[Figure 5 : 1 dolby audio stream]

Dash also offers the ability to stream multiple audio tracks. In this way we are not forced to make a merge of all the flow of the concert but to broadcast each stream separately and synchronized them on the client side (Figure 6).



[Figure 6 : 3 stereo audio stream]

5.3.1 Limitations

The Dash Js player does not support multi track [8].

We investigate Media Source API : the “addTrack” method can not simultaneously play multiple tracks but only switch between different tracks. This function allows you to select the audio language for example.

The "addSourceBuffer" method is also limited [9]:

“*A single SourceBuffer with 1 audio track and/or 1 video track. Two SourceBuffers with one handling a single audio track and the other handling a single video track.”*

5.3.2 Implementation

Starting from this experiment (at 5.2), it is possible to realize a multi-stream streaming by duplicating the audio track for each loading and decoding data processes. Synchronization can be made using loading events : for each fragment, we wait until each track has loaded his packet of data. The last one gives the start.

6. CONCLUSION AND FUTURE WORKS

After we discussed the issue directly with the developers of Web Audio API [7], we were pleased to learn that the working group has decided to improve the decoding function of the Web Audio API [10]. It will therefore be possible in the future to send compressed data directly to Web Audio API which will enhance our solution.

7. REFERENCES

[1] http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/MPEG-DASH-Industry-Forum-Releases-Implementation-Guidelines-90527.aspx

[2] http://gpac.wp.mines-telecom.fr/mp4box/dash/

[3] http://standards.iso.org/ittf/PubliclyAvailableStandards/

[4] https://dvcs.w3.org/hg/html-media/raw-file/tip/media-source/media-source.html

[5] https://github.com/Dash-Industry-Forum/dash.js/

[6] http://gpac.wp.mines-telecom.fr/2011/02/02/mp4box-fragmentation-segmentation-splitting-and-interleaving/

[7] https://github.com/WebAudio/web-audio-api/issues/337

[8] https://github.com/Dash-Industry-Forum/dash.js/issues/16

[9] https://dvcs.w3.org/hg/html-media/raw-file/tip/media-source/media-source.html#methods

[10] https://github.com/WebAudio/web-audio-api/issues/371

[11] http://www.bbc.co.uk/rd/blog/2014/10/under-milk-wood-in-headphone-surround-sound