RAPPORT

28/2015
ISBN 978-82-7492-319-5
ISSN 1890-5226

# MEDIASYNC REPORT 2015

## Evaluating timed playback of HTML5 Media

Forfatter: Njål T. Borch, Ingar M. Arntzen

**PROSJEKTNAVN: MediaScape** Prosjektnr.: 531

**OPPDRAGSGIVER(E):** Oppdragsgivers ref.:

|  |  |  |
|---|---|---|
| Dokumentnr.: 28/2015 | Dokumenttype: Rapport | Status: Open |
| ISBN: **978-82-7492-319-5** | ISSN: **1890-5226** | Ant. Sider: 24 |
| Prosjektleder: Njål T. Borch |  | Dato: 18.12.2015 |

**FORFATTER (E):** Njål T. Borch, Ingar M. Arntzen

**TITTEL: MediaSync Report 2015: Evaluating timed playback of HTML5 Media**

Resymé / Summary:
In this report we provide an extensive analysis of timing aspects of HTML5 Media, across a variety of browsers, operating systems and media formats. Particularly we investigate how playback compares to the progression of the local clock and how players respond to time-shifting and adjustments in playback-rate.

Additionally, we use the MediaSync JS library to enforce correctly timed playback for HTML5 media, and indicate the effects this has on user experience. MediaSync is developed based on results from the above analysis. MediaSync aims to provide a best effort solution that works across a variety of media formats, operating systems and browser types, and does not make optimizations for specific permutations..

Emneord: media synchronization, timing, timed playback, HTML5 Media

Noter:

**UTGIVER: Norut, P.O. BOX 6434, N-9294 Tromsø, Norway**

# Innhold

## Innhold

## 1 INTRODUCTION

In a world where audio and video content can be viewed on more than one device at a time, correctly timed (synchronized) media playback is crucial for the user experience. For example, if sound is presented by one device and video by another, timing differences are typically required to be less than 30ms in order to provide lip sync. If the difference is any bigger, humans will detect that lips do not move together with the sound. For multi-device radio playback in the same room, echoless (<10ms) audio playback may be required.

As an approach to precisely timed multi-device playback in Web browsers, the W3C Multi-device Timing Community Group [1] have recently proposed the Timing Object [2] as basis for precisely timed behavior (e.g. playback) in Web browsers, covering both single-device and multi-device scenarios. The basic idea is that timed components (e.g. media players) take direction from a Timing Object. If multiple, independent components take direction from a shared Timing Object, their behavior (playback) will be precisely coordinated in time (synchronized). This is also true in the multi-device scenario, as the Timing Object may be connected to online timing resources. This enables precisely timed behaviour in Web browsers across the world. An example video of what precisely timed multi-device video playback may look like is available in [5].

The introduction of the Timing Object thus reduces the challenge of multi-device playback for media content. Support for locally controlled timed playback in HTMLMediaElements [3] will be sufficient. In this report we document current abilities and limitations for timed playback of audio and video in Web browsers. In particular, we document precision and reliability of *currentTime*, *seek* and variable *playbackrate*, across a range of media formats, Web browsers and operating systems. Based on these findings we have developed mediaSync [4], a JavaScript library for timed playback of HTML5 audio and video. By synchronizing audio and video using mediaSync we are able to document the precision for timed playback in our implementation, as well as indicate expected user experiences.

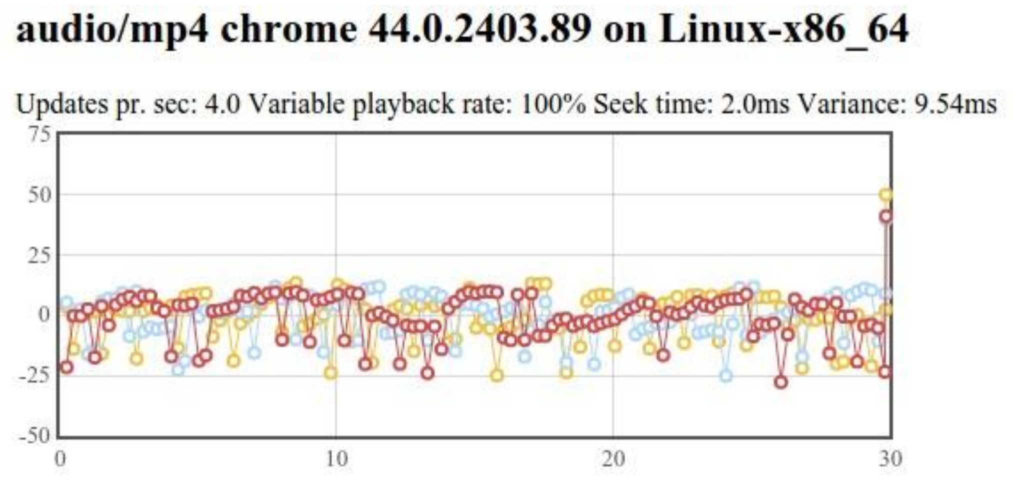
The overall results show that there are large inconsistencies between browsers, operating systems, browser versions and even codecs. In particular, lacking or errors in variable playbackRate is the largest sole problem in this area. It is also clear that the media element's lack of understanding of a moving external clock makes buffering difficult, as a play command will seek to the given position, then start fetching and decoding data, without compensating for the time between the play command and actual playback. This leads to less graceful starts, as the media playback will always be late and require possibly large adjustments.

## 2 EXPERIMENTS

We have performed two different experiments for this report. The first analyses the media elements to see exactly how they report progress (currentTime) and how they respond to controls (seek, playbackRate). The second experiment analyses external control of the media elements by our MediaSync library, hence giving an indication of the end user experience during timed playback (synchronization).

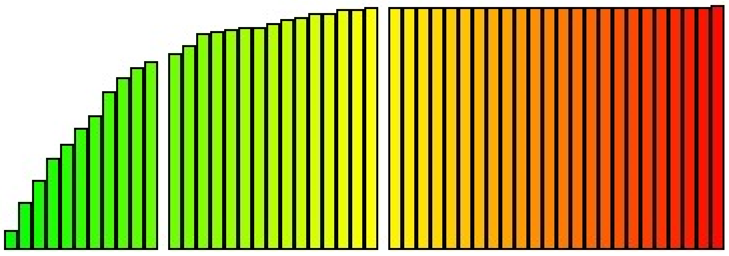
**Experiment 1; Reporting**

We have created a simple test that measures and plots how the currentTime property is reported. The value of currentTime is sampled as a reaction to timeupdate events. We expect media elements to report a linear progress, preferably with a consistent playback speed of 1 - hence ideally creating a flat, horizontal line in our plots. In this report, the values on the Y axis have been normalized to the mean value of each plot, thus any buffering or delays in startup are not visible.



This plot shows three different runs of Chrome 44, playing MP4 audio on 64bit Linux. It shows that the currentTime property does not describe a perfectly linear progress, but it is within about +15 to -25 ms off the mean values. Different players also have a degree of player drift, meaning that the values of currentTime and the system clock do not increase at exactly the same rate. The effect of this is that periodic adjustments are needed also during playback. If not, audio devices will soon create echo, and never improve. Player drift is likely due to media decoders (we have observed different drift on different media codecs on a single device), resource limitations or a number of other factors. However, the experiments performed for this report only last 30 seconds, so player drift may not be clearly visible.

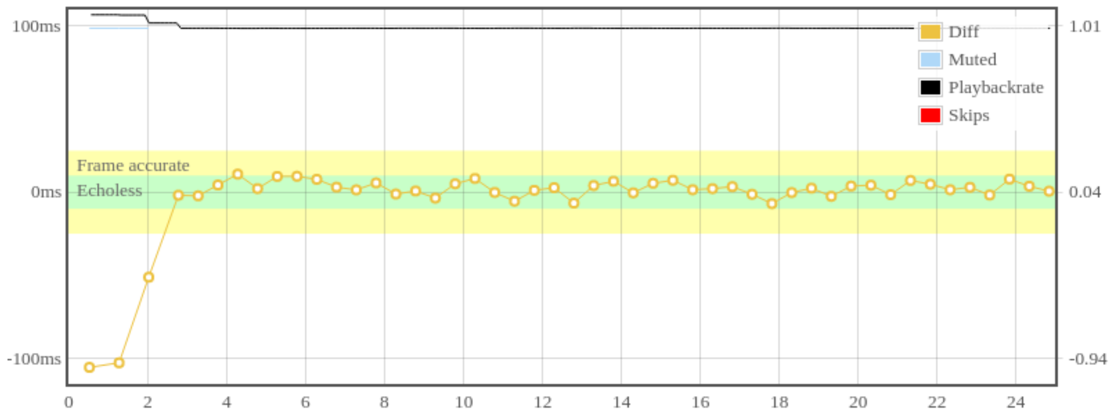
In order to visualize how many of the samples are close to the mean, we've created some cumulative bar charts, showing the aggregate number of measurements within a given distance. The first bar is % of measurements within 0ms of the mean, the second bar is % of measurements within 1ms, and so on. Perfect reporting of currentTime will lead to a chart with all bars at 100%, meaning that all measurements are exactly the mean value. We have also added some gaps for readability. The first gap is after 10ms, which is generally thought to give echo-less playback of audio. The second gap is at 25ms, which is within most definitions of lip sync, or frame accuracy. This gives us a simple profile on how accurate browsers report their position. Note that averaging measurements might improve on these reports, this is just analysis of raw data from the media elements.



**Experiment 2; User experience**

In order to synchronize multiple devices, we need the media elements to slave after an ideal playback clock. This ideal clock is supplied by the timing object, in our experiments implemented by the Motion Corporation in the form of Shared Motions.

In order for media playback to be simultaneous on multiple devices, we have created a Javascript MediaSync library [4] that makes HTML5 media elements take direction from a timing object/ shared motion. The basic algorithm is to compare (periodically) the currentTime of the media element with the ideal playback clock defined by the timing object. If supported, the MediaSync library will use variable playback rate to do fine adjustments to playback position. If this is not supported, or if the player is too far off, it will seek by updating the currentTime property. Ideally, the media elements would immediately start playing from the new offset, but this is of course not possible due to data transmissions, buffers, decoding stacks and so on. Setting the currentTime property may be disruptive and time consuming for the media element.



**The experience plots show how the currentTime compares to the ideal playback position, while playback is controlled by the MediaSync library. The green band is +- 10ms, thus showing when playback is believed to be echoless, the yellow band is +-25ms, showing when playback is believed to be within lip sync. To cover different definitions for lip sync, we've opted for 25ms in our experiments to cover most. The black line on top references the Y axis values on the right, and describes the playback rate. Any seek operations will be noted by red circles.**

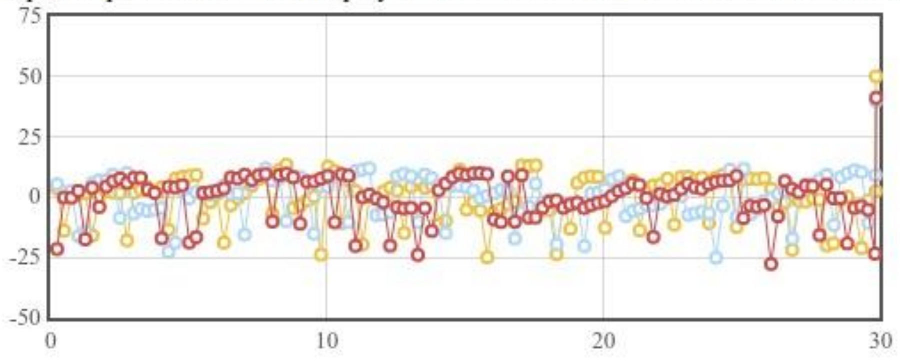
The optimal user experience would be a horizontal line (yellow dots) on top of the 0ms line, with no seeks (red circles) and a playback rate of very close to 1.0 (black line).

## 3 OPERATING SYSTEMS

The largest difference in playback behavior is typically between browsers, but there are some interesting observations about different operating systems too. This could be due to differences in the media stacks, precision/resolution of clock, or a number of other reasons. Not all tests are run on the same machines either, which also affects the results.  Here we have tested the Chrome browser on a variety of operating systems.

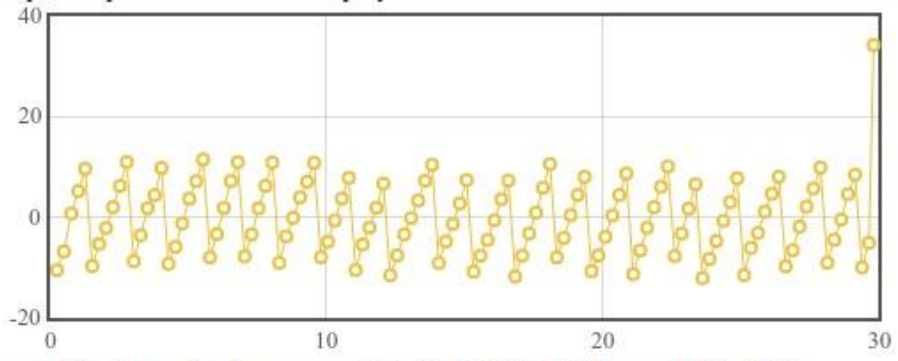
**audio/mp4 chrome 44.0.2403.89 on Linux-x86_64**

Updates pr. sec: 4.0 Variable playback rate: 100% Seek time: 2.0ms Variance: 9.54ms



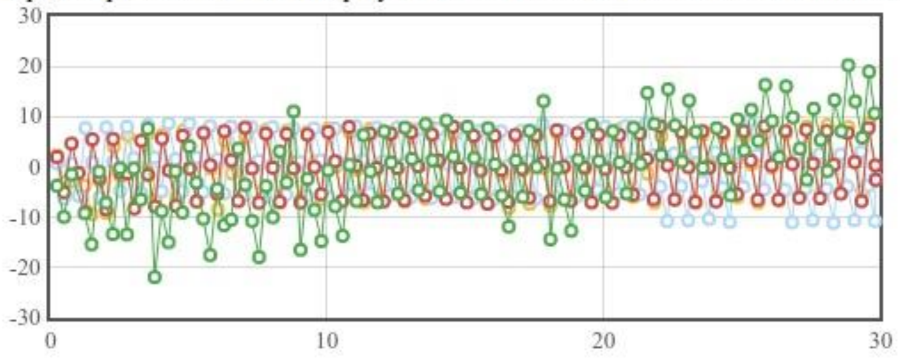
**audio/mp4 chrome 44.0.2403.157 on MacIntel**

Updates pr. sec: 4.0 Variable playback rate: 100% Seek time: 2.0ms Variance: 7.46ms



**audio/mp4 chrome 43.0.2357.124 on Win32**

Updates pr. sec: 3.9 Variable playback rate: 100% Seek time: 3.0ms Variance: 8.94ms



We have tested the Chrome browser running on Linux, Windows 8 and 10 as well as OSX. The browser reports currentTime mostly within +-25ms, but in particular on OSX there is a visible pattern.

The profile bar charts for Chrome look like this:

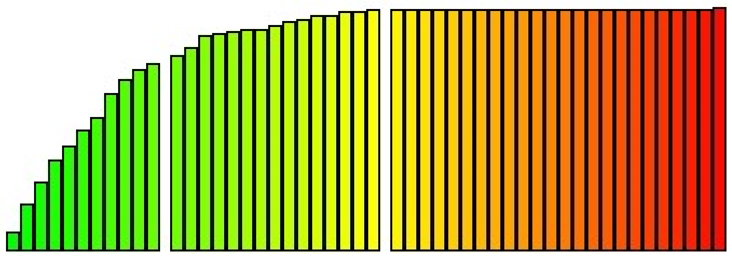


*Illustration 1: Chrome on Linux*

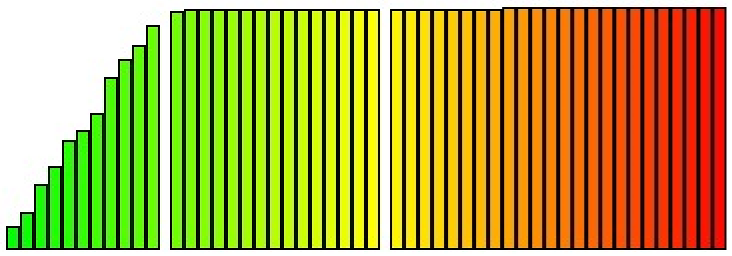


*Illustration 2: Chrome on OSX*

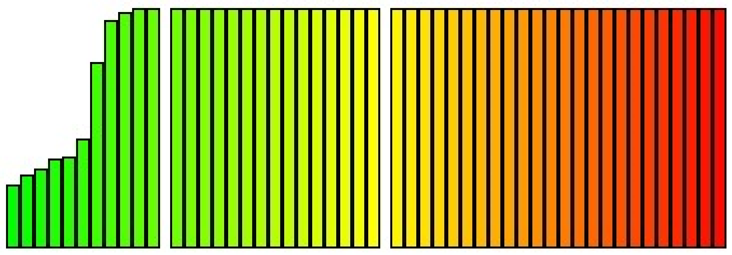


*Illustration 3: Chrome on Windows 10*

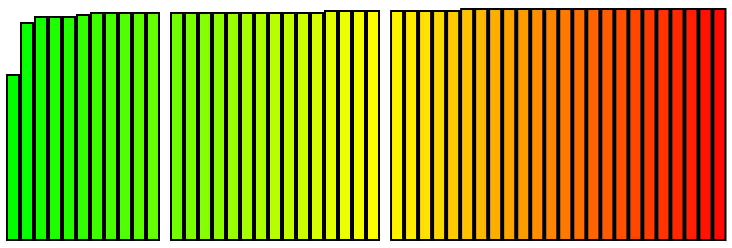


*Illustration 4: Chrome Dev (47) on Android*

As we can see, Chrome does experience some differences on different operating systems and machines, and the spread of currentTime reporting is relatively substantial. Recent improvements in Chrome on Android are not reflected on the desktop version.

## 4 BROWSERS

We have performed tests on the most used browsers, and they have different properties. Note that these profiles change over time, and these measurements were performed in September 2015 (October for Chrome dev on Android).

**Chrome**

The Chrome browser does support variable playback rate on computers, and as such is able to provide a fairly good user experience.

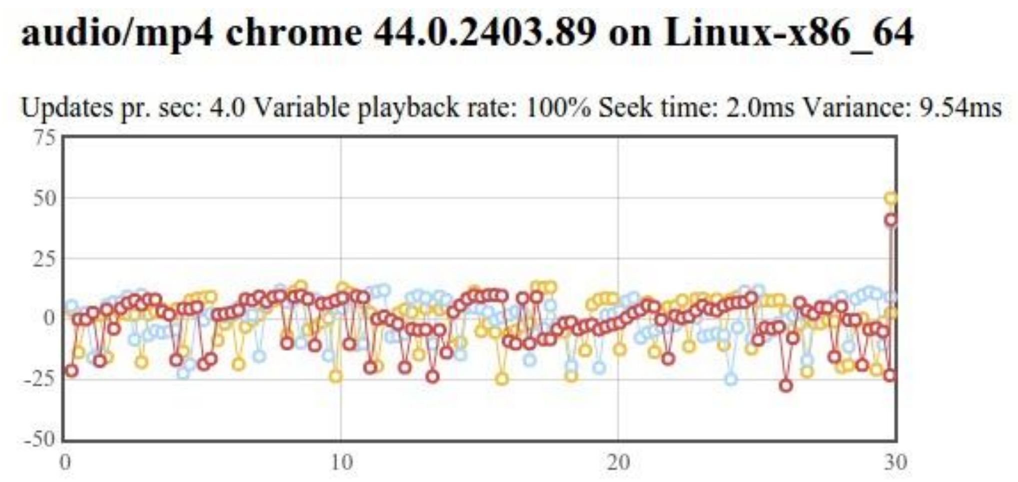
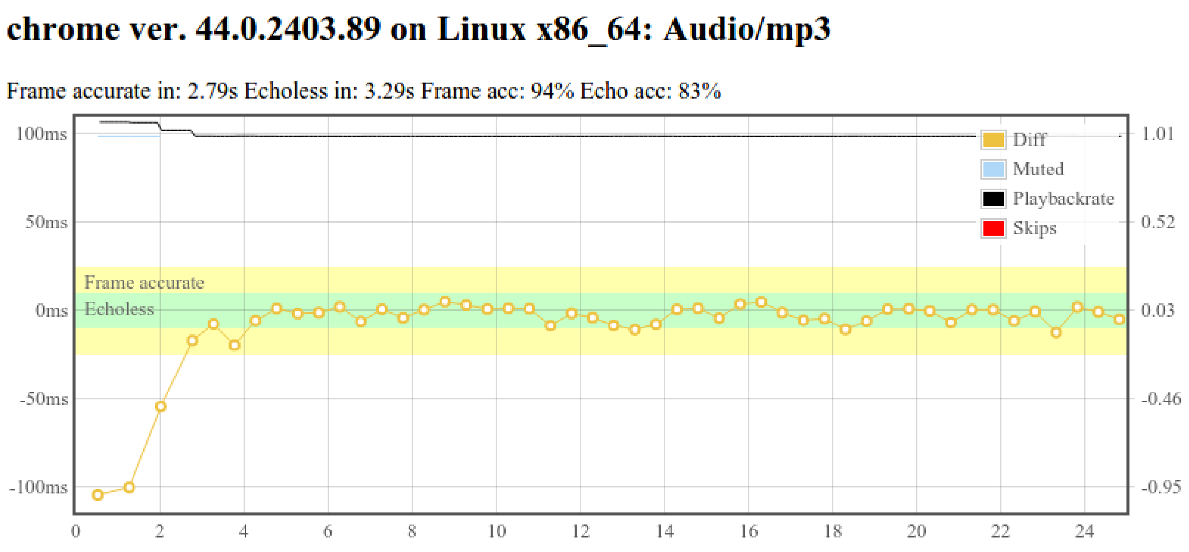


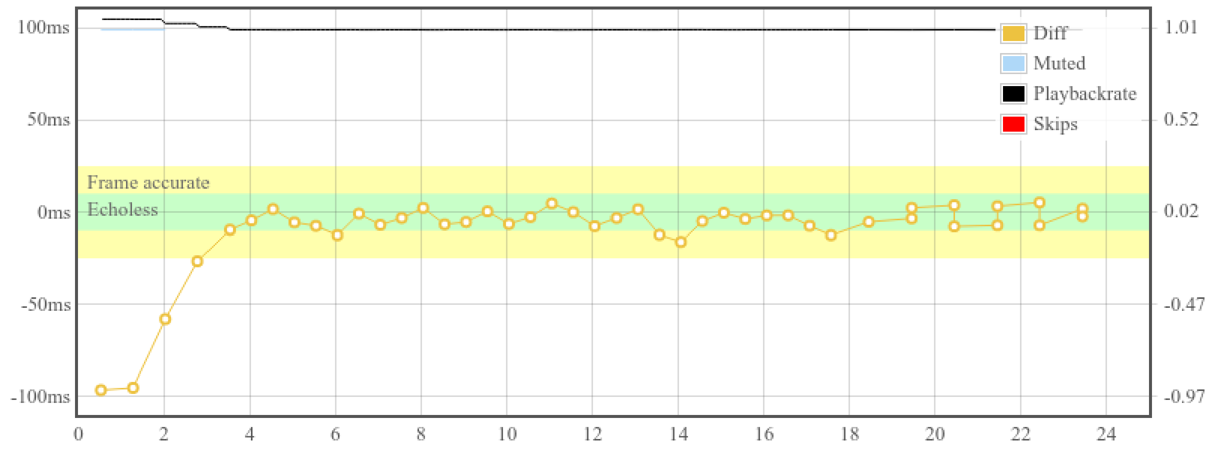
*Illustration 5: Chrome is generally within 25ms of the mean value*

While Chrome is somewhat rough reporting the currentTime, averaging a few measurements gives us a relatively accurate playback position. The need for averaging the currentTime property does slow down our adaptation, as we gather close to a second worth of samples to estimate the actual position.



## chrome ver. 44.0.2403.89 on Linux x86_64: Video/ mp4

Frame accurate in: 3.54s Echoless in: 3.54s Frame acc: 100% Echo acc: 100%



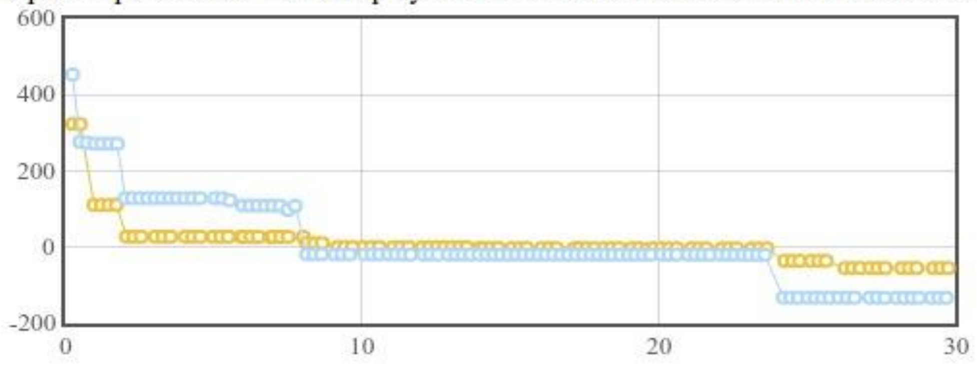
Chrome manages to synchronize both audio and video very nicely due to the smooth adjustment of playback rate. This adds very little stress to the media stack, and involves no new network connections or other disruptive actions. It manages to achieve lip sync in less than 3 seconds, is echoless just after that and mostly stays within +-10ms for the rest of the playback. The relatively uncertain reporting of positions means that playback rate and positions fluctuate somewhat, but this is still quite a good experience on all desktop operating systems.

### Chrome on Android

On Android, Chrome performs somewhat differently than on desktops. It largely reports very even values, but it tends to have drop-offs. Some of this could be to high player drift.

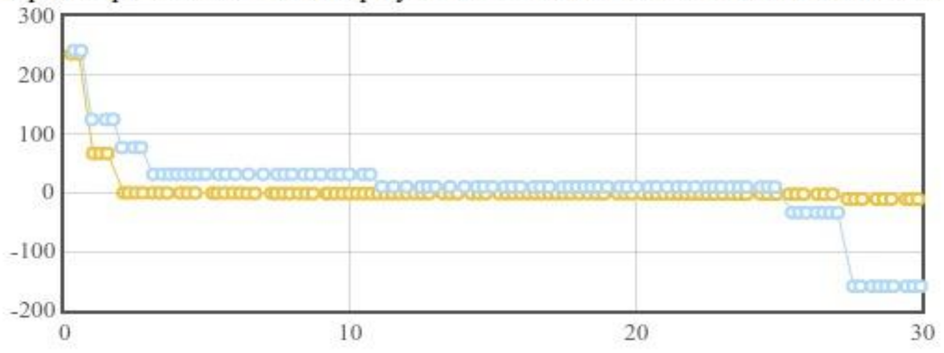
## video/webm chrome 45.0.2454.94 on Linux-armv7l

Updates pr. sec: 3.5 Variable playback rate: % Seek time: 262.0ms Variance: 97.54ms



## audio/mp4 chrome 45.0.2454.94 on Linux-armv7l

Updates pr. sec: 3.0 Variable playback rate: % Seek time: 138.0ms Variance: 59.92ms

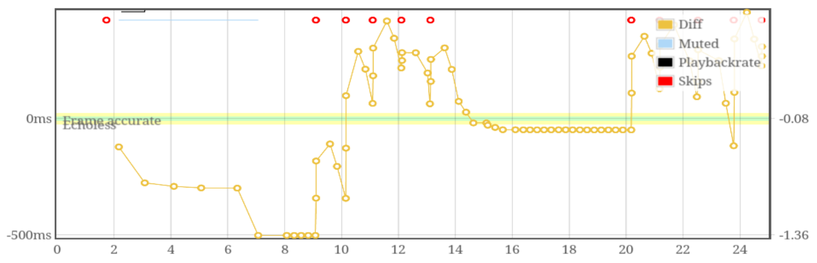


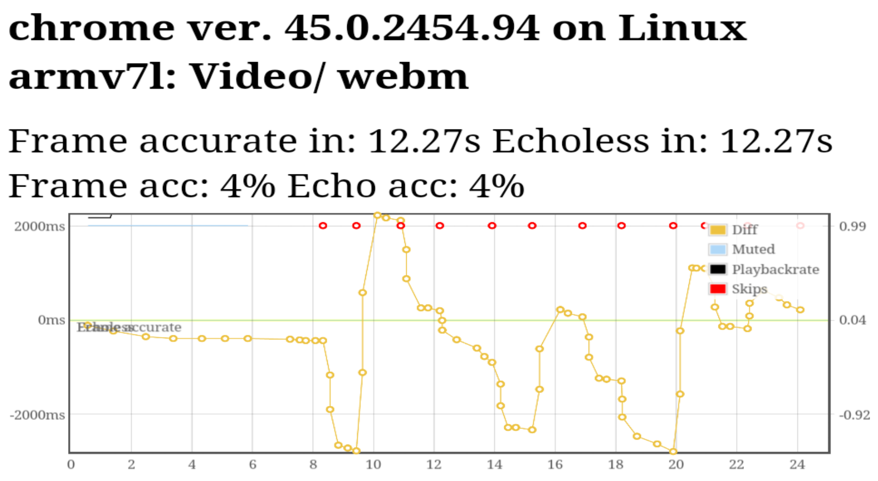
The experience provided by Chrome on Android is not particularly pleasant, although it has improved in the last few versions. The lack of variable playback rate makes seeking a necessity, and in relatively low power devices like smartphones, the seek accuracy is less than perfect.  Support for variable playback rate should amend this, and we expect that Chrome on Android will be a very good browser for multi device media playback when this functionality is added.

---

## chrome ver. 45.0.2454.94 on Linux armv7l: Audio/mp3

Frame accurate in: 14.64s Echoless in: s Frame acc: 5% Echo acc: %



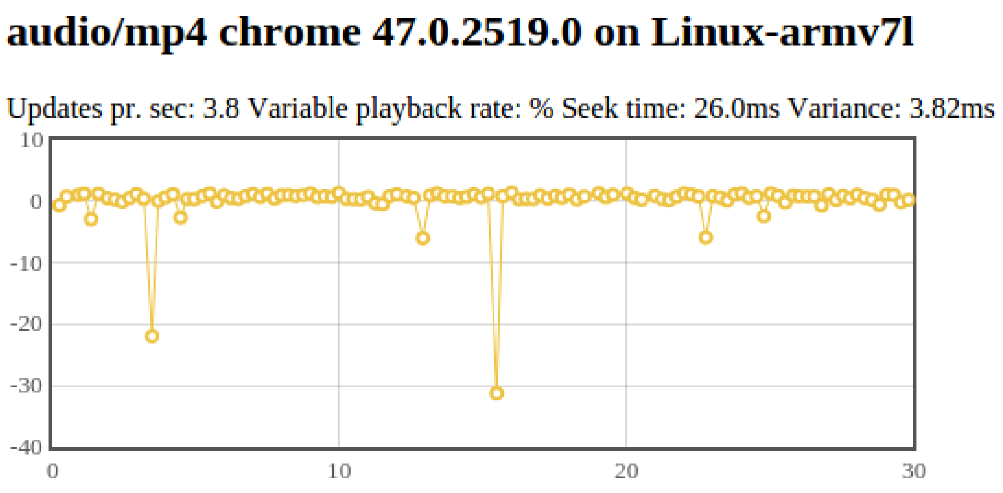

**chrome ver. 45.0.2454.94 on Linux armv7l: Video/ webm**

Frame accurate in: 12.27s Echoless in: 12.27s
Frame acc: 4% Echo acc: 4%

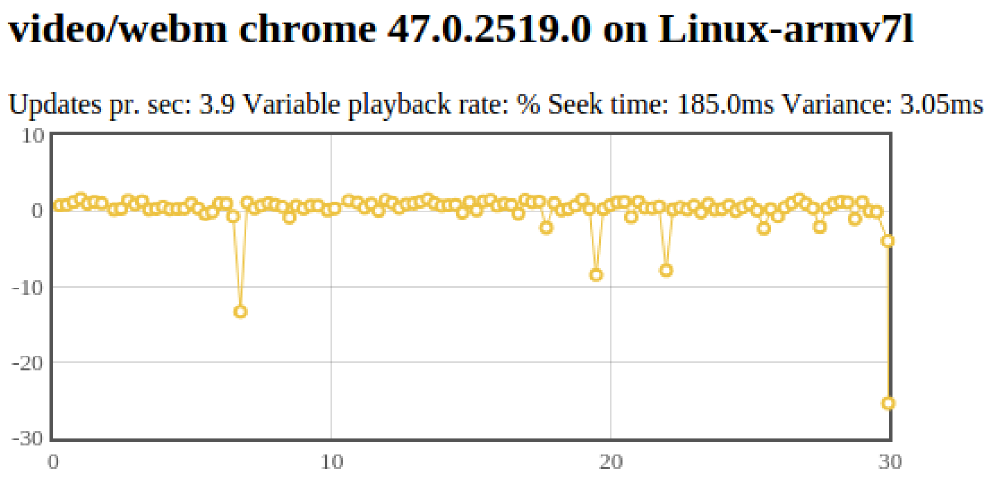
*Illustration 6: The red circles are seek events, and these substantially subtract from the user experience.*

### Chrome Dev on Android

We also ran the experiments on Chrome Dev for Android, as it’s rumored to have a variety of video related optimizations due to WebRTC efforts. The difference is quite noticeable:


**audio/mp4 chrome 47.0.2519.0 on Linux-armv7l**

Updates pr. sec: 3.8 Variable playback rate: % Seek time: 26.0ms Variance: 3.82ms


video/webm chrome 47.0.2519.0 on Linux-armv7l

Updates pr. sec: 3.9 Variable playback rate: % Seek time: 185.0ms Variance: 3.05ms

*Illustration 7: Both for audio and video, Chrome Dev on Android is actually more precise in it's currentTime reporting than the desktop version.*

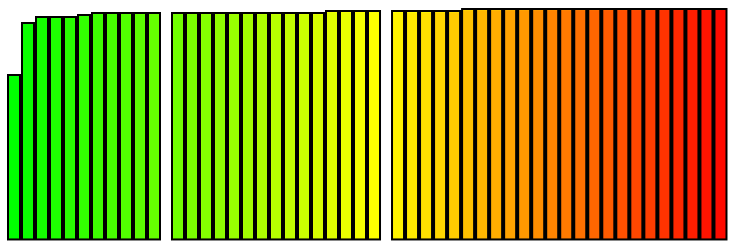


*Illustration 8: A very nice and consistent profile from Chrome 47 on Android*

When it comes to the user experience of Chrome 47, it's still lacking variable playback rate, although the seek performance has improved greatly. However, on an actual listening test, audio/mp4 is fine but audio/mp3 reports the wrong currentTime. The effect is that our sync library believes all is well (to within a few milliseconds of the ideal target), while the actual audio being played is incorrect. We did not measure how wrong, but we're guessing closer to seconds than milliseconds. Audio/ogg seems to report ok, but the seek performance is much worse than the other codecs, leading to a very unpleasant experience.

## chrome ver. 47.0.2526.6 on Linux armv7l: Audio/mp4

Frame accurate in: 12.09s Echoless in: 12.09s
Frame acc: 65% Echo acc: 65%

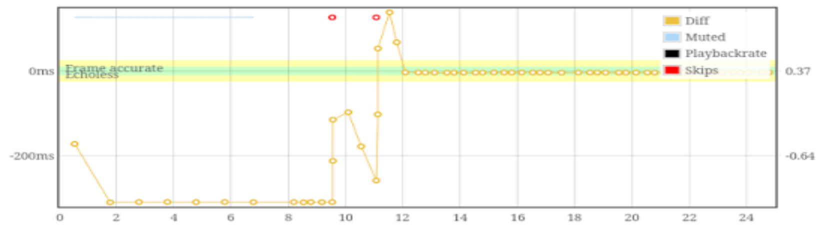


*Illustration 9: Much improved seek performance on Chrome 47*

## chrome ver. 47.0.2526.6 on Linux armv7l: Video/ webm

Frame accurate in: s Echoless in: s Frame acc:
% Echo acc: %

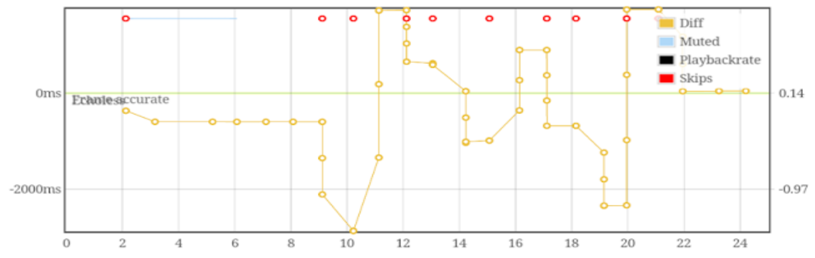
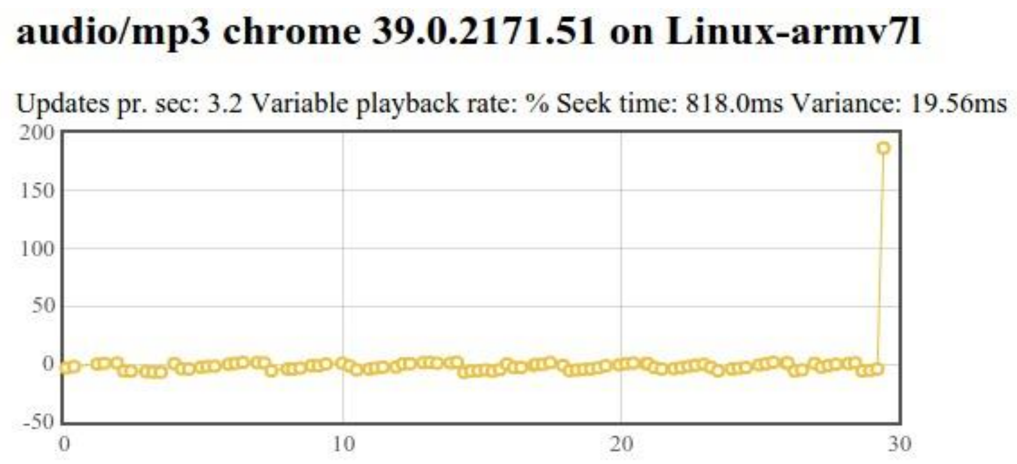


*Illustration 10: After seeks, the currentTime is now largely horizontal lines, reflecting the much improved seek performance. Lacking variable playbackRate still makes this a poor experience though.*

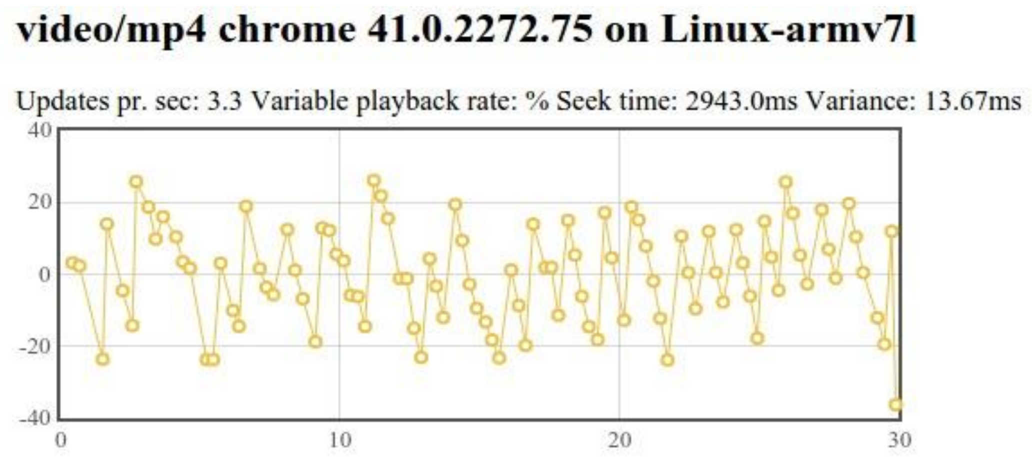
**Chrome on Chromecast**

We also performed some tests on the original Chromecast dongle. The reporting of currentTime is fairly accurate, fluctuating around a likely correct point similar to Chrome on desktops. It does however not support variable playback rate, and seeking is very slow - up to several seconds. This makes for a mediocre synchronized media playback experience at best. As the Chromecast seems quite consistent in its seek performance, we are however able to make it synchronize well, but it takes quite some time. With support for variable playback rate on Chromecast, it will likely perform very well.



**audio/mp3 chrome 39.0.2171.51 on Linux-armv7l**

Updates pr. sec: 3.2 Variable playback rate: % Seek time: 818.0ms Variance: 19.56ms



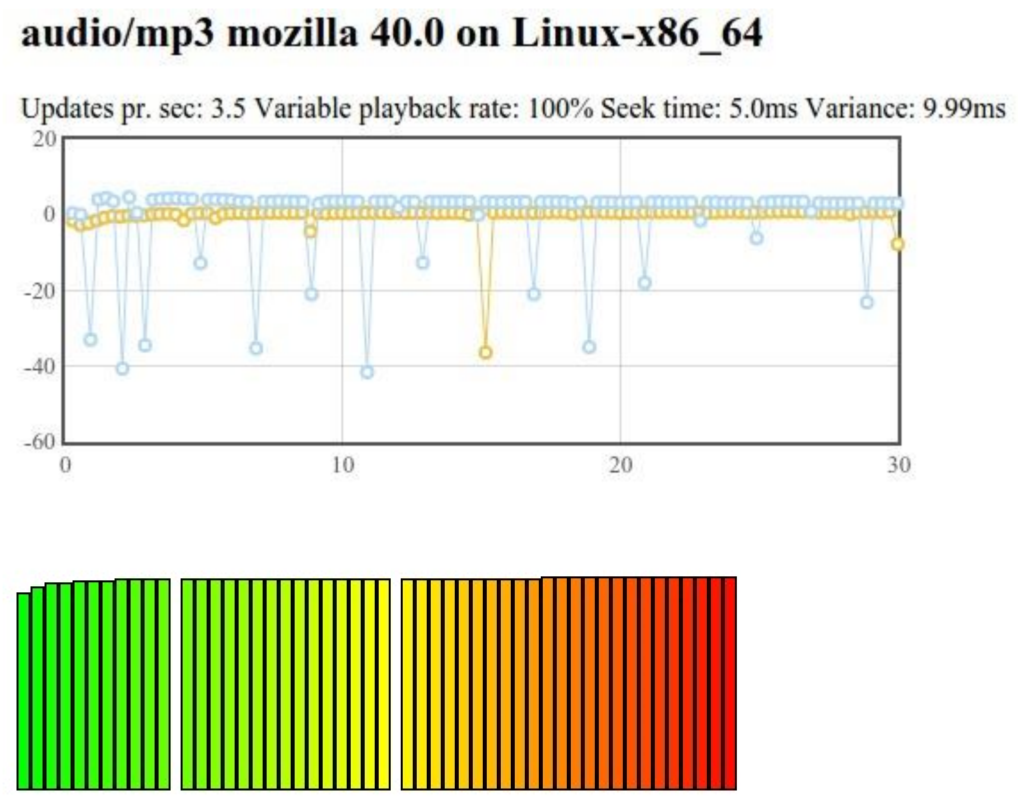
**video/mp4 chrome 41.0.2272.75 on Linux-armv7l**

Updates pr. sec: 3.3 Variable playback rate: % Seek time: 2943.0ms Variance: 13.67ms

*Illustration 11: Apart from a lone bad sample at the end, the Chromecast seems to report currentTime in a similar manner to Chrome on desktops. The resource constraints makes audio quite a bit better than video.*

**Firefox**

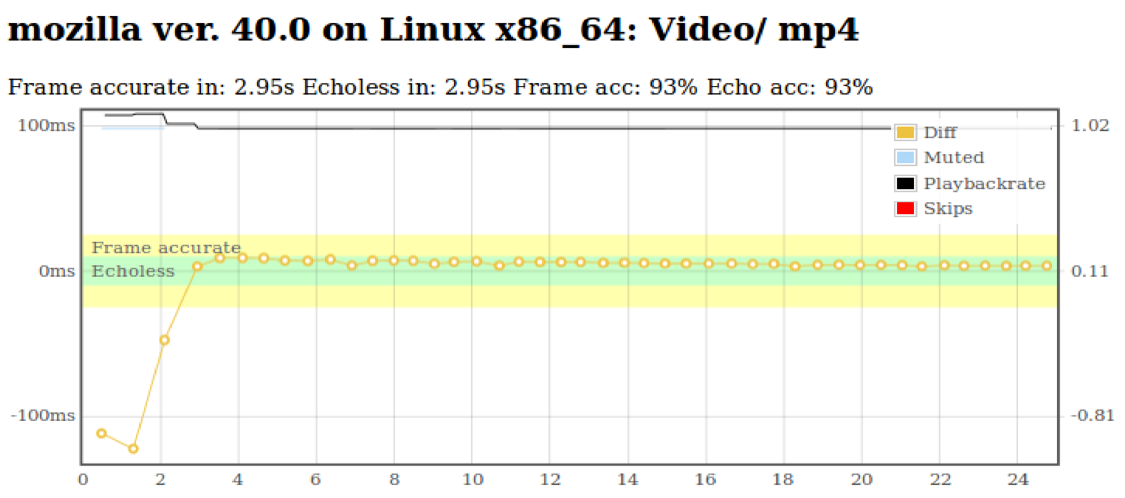
Firefox is available on most platforms, and performs well. It is very similar in performance on all desktop operating systems, with OSX being slightly less precise due to a similar effect to that of Chrome.



**audio/mp3 mozilla 40.0 on Linux-x86_64**

Updates pr. sec: 3.5 Variable playback rate: 100% Seek time: 5.0ms Variance: 9.99ms

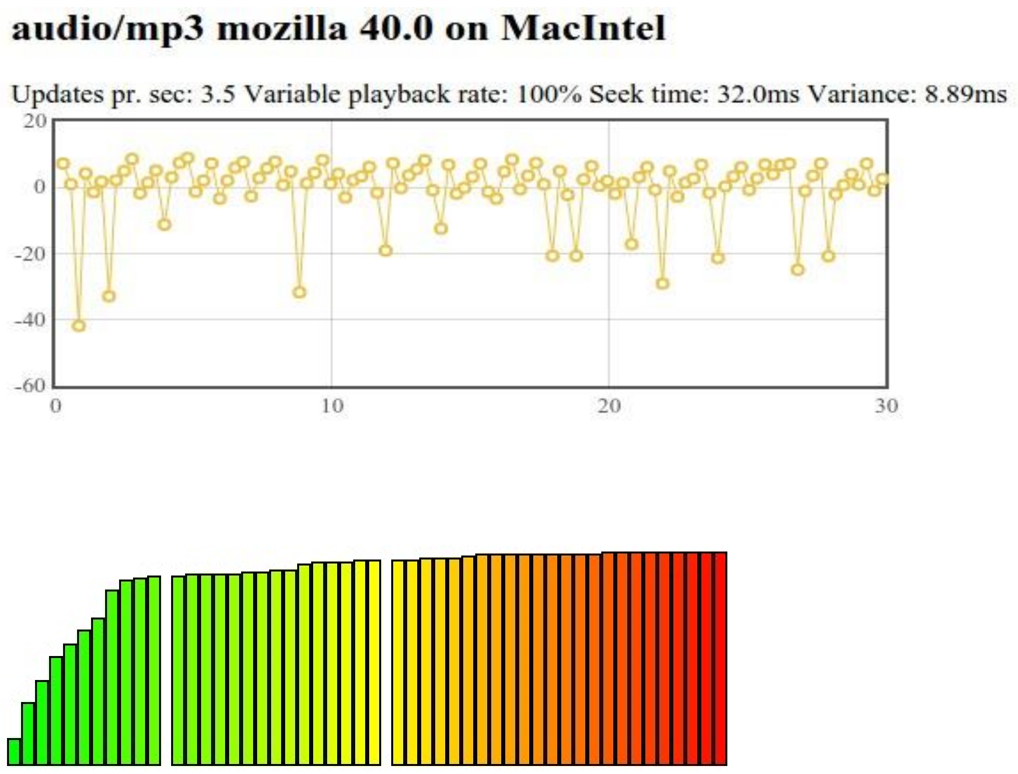
*Illustration 12: The profile of Firefox looks more off than it is for practical purposes. The big spikes (possibly due to single-process design?) are easily ignored.*



**mozilla ver. 40.0 on Linux x86_64: Video/ mp4**

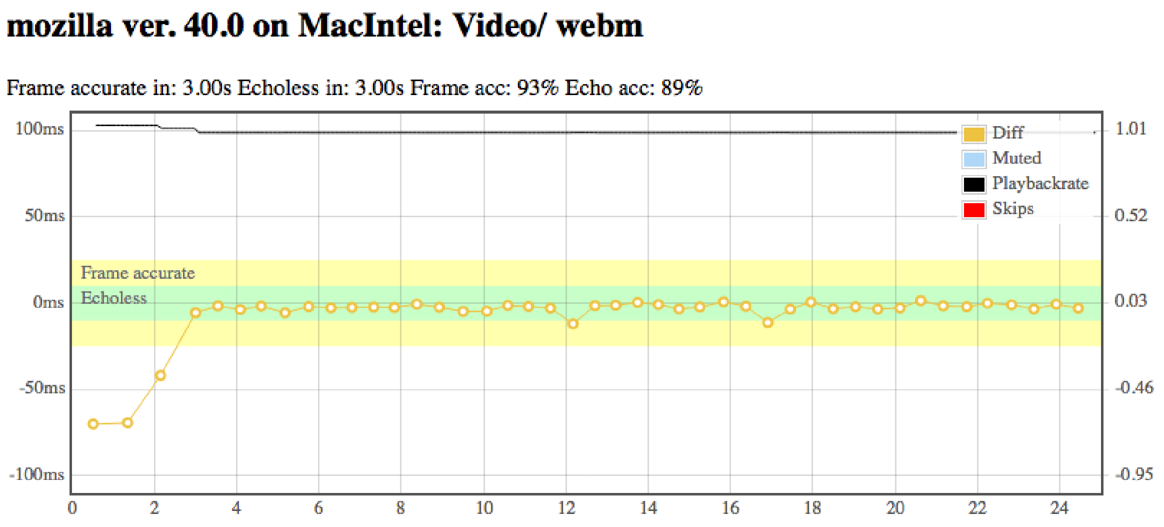
Frame accurate in: 2.95s Echoless in: 2.95s Frame acc: 93% Echo acc: 93%

*Illustration 13: Variable playback rate smoothly moving directly into echoless playback in less than 3 seconds*


**audio/mp3 mozilla 40.0 on MacIntel**

Updates pr. sec: 3.5 Variable playback rate: 100% Seek time: 32.0ms Variance: 8.89ms

*Illustration 14: On OSX, Firefox presents a much wider spread of measurements*


**mozilla ver. 40.0 on MacIntel: Video/ webm**

Frame accurate in: 3.00s Echoless in: 3.00s Frame acc: 93% Echo acc: 89%

*Illustration 15: Firefox is quite well behaved after smoothing the currentTime values*

**Firefox on Android**

**video/webm mozilla 41.0 on Linux-armv7l**

Updates pr. sec: 2.3 Variable playback rate: % Seek time: 2463.0ms Variance: 93.30ms

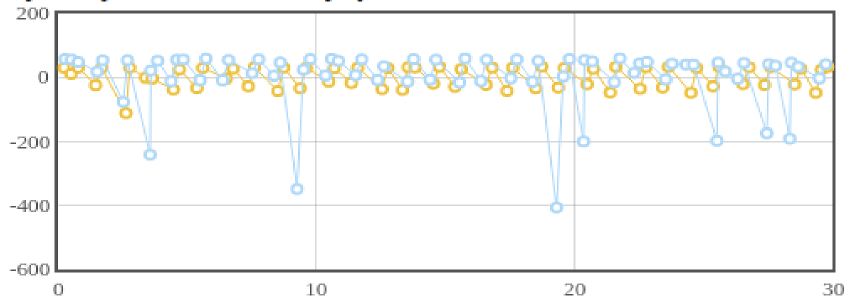
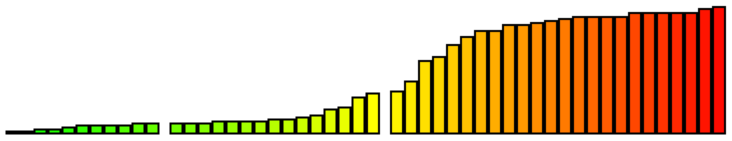
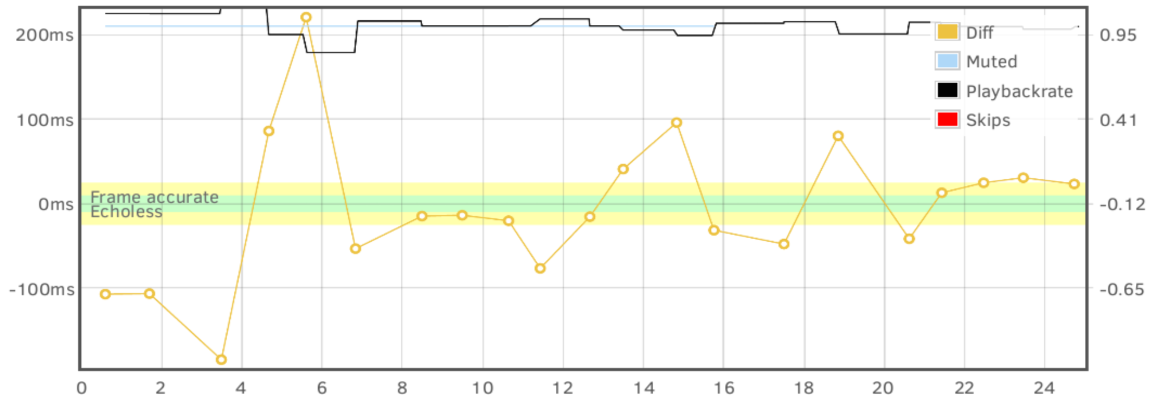


*Illustration 16: Firefox on Android does support variable playback rate*



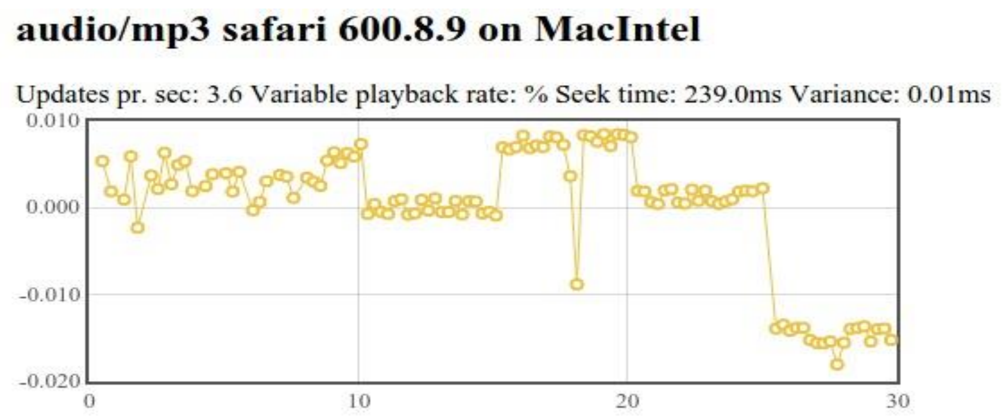
**mozilla ver. 41.0 on Linux armv7l: Audio/Vorbis**

Frame accurate in: 8.49s Echoless in: s Frame acc: 33% Echo acc: %



## Safari

Safari is very good at reporting it’s current playback position, but it lacks support for variable playback rate. Note that the Y axis spans only 0.03 milliseconds, so for all practical purposes, Safari describes a straight line.


audio/mp3 safari 600.8.9 on MacIntel

Updates pr. sec: 3.6 Variable playback rate: % Seek time: 239.0ms Variance: 0.01ms

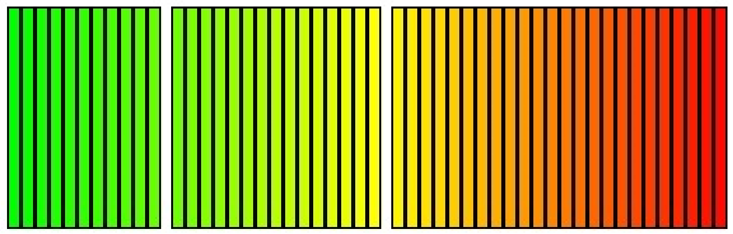
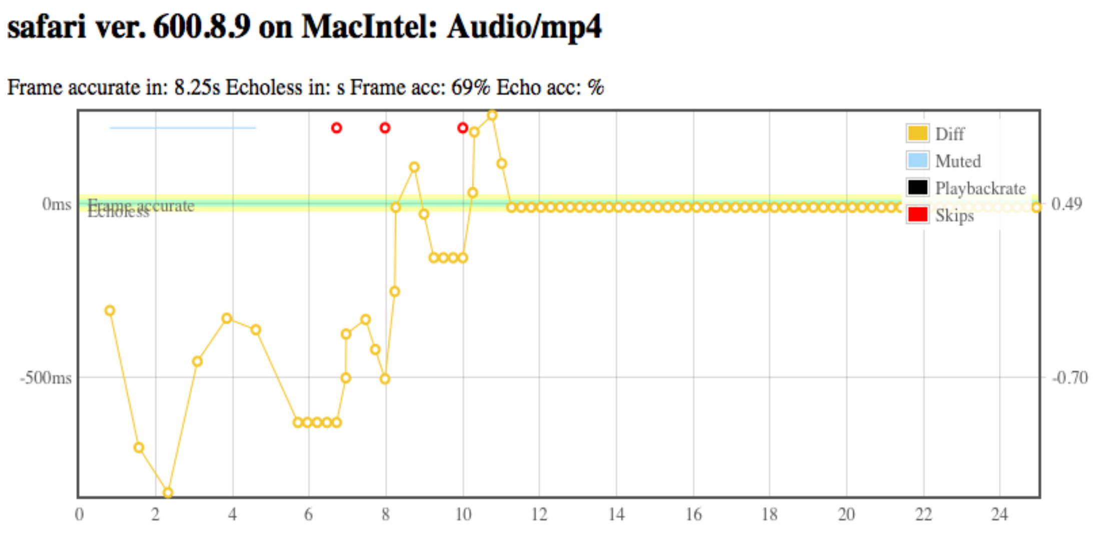


*Illustration 17: The profile of Safari is very flat indeed, with 100% of measurements within 0ms of the target.*

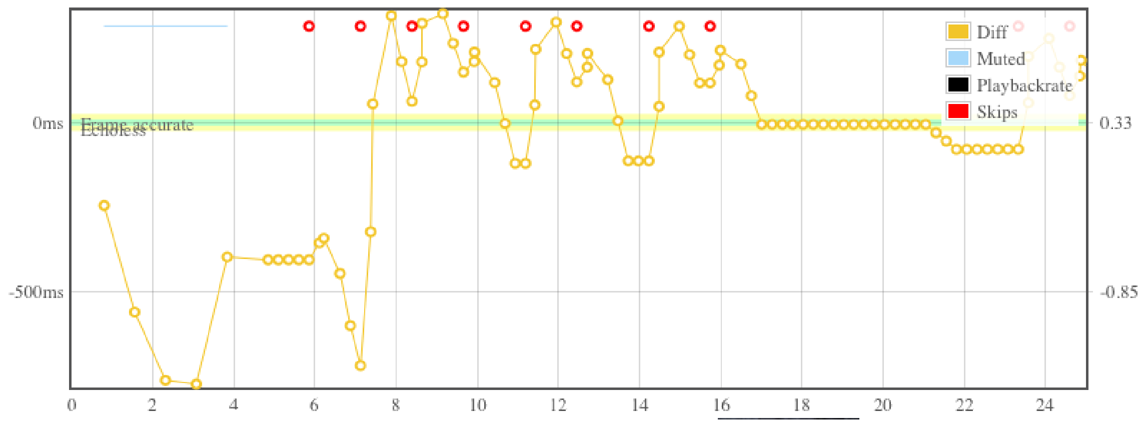

safari ver. 600.8.9 on MacIntel: Audio/mp4

Frame accurate in: 8.25s Echoless in: s Frame acc: 69% Echo acc: %

Due to the lack of variable playback rate, the MediaSync library needs to seek. The seeks are noted as red circles in the plot. The first seek is after a bit over 6 seconds, as the library spends some time trying to adjust the playbackRate property and realise that it can be updated but has no effect. The seek is analyzed to measure how long the audio element actually takes to perform the seek, the next seek then tries to compensate by jumping further ahead, making it possible to hit closer to the ideal target. As can be seen here, playback is within lip sync after about 11 seconds, although it never reaches echo less playback (seeking often leads to thrashing). The user experience is not particularly good, as seeking is very noticeable, and due to the long time frame accuracy is achieved.

**safari ver. 600.8.9 on MacIntel: Video/ mp4**

Frame accurate in: 10.68s Echoless in: 10.68s Frame acc: 23% Echo acc: 27%

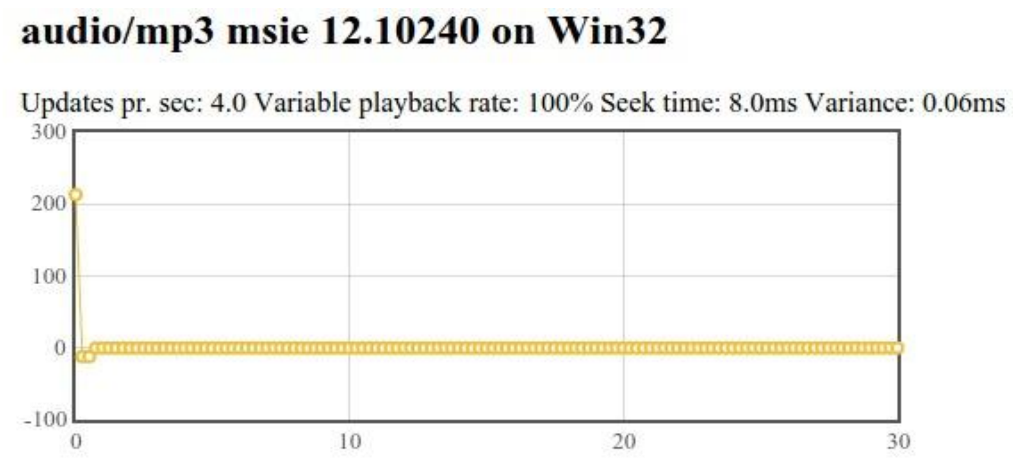


Video playback is even more of a challenge for the seek-based adjustments, as the increased data rates and processing power needed to decode is larger than for audio. Our test plays a 720p video. Here, the video element seeks 8 times, at which point it becomes very well synchronized, but for some reason, it looses sync after a few seconds and start seeking again. We see that Safari quite often will have such dips in the playback, triggering further seeks. Safari here needs about 17 seconds to get in sync, and failed to permanently synchronize the playback within 30 seconds.

We were unable to get Safari to run our tests on iOS at this time.

## Edge

Edge takes over for Internet Explorer. It shows a similar profile to IE11.



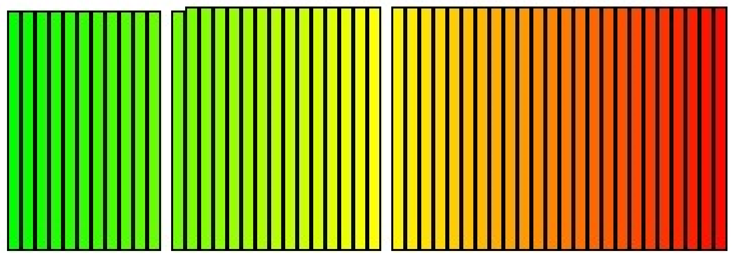
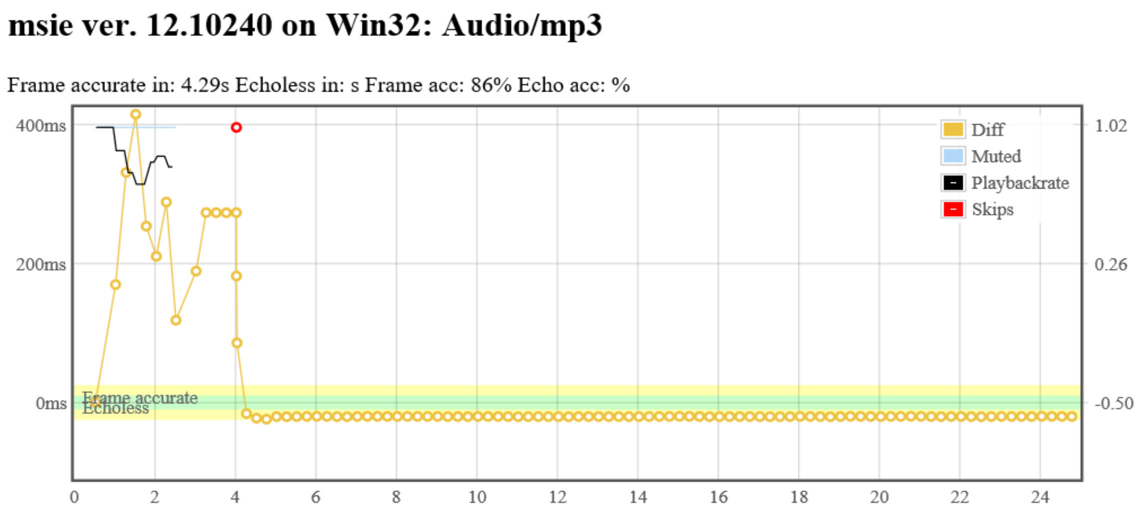


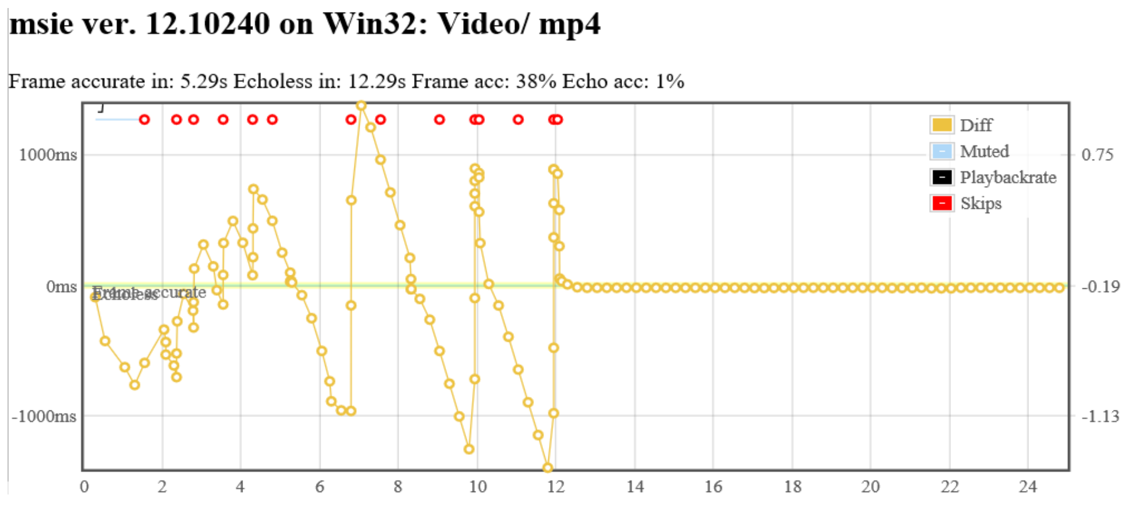
*Illustration 18: Edge does quite well, apart from a few bad measurements at the beginning of playback.*

With accurate reporting and variable playback rate support, Edge should be perfectly set to perform excellent. However, there seems to be an issue with the implementation where changing the playbackRate property also leads the browser to seek a non deterministic distance. This forces us to use seeking to correct the media elements.


msie ver. 12.10240 on Win32: Audio/mp3

Frame accurate in: 4.29s Echoless in: s Frame acc: 86% Echo acc: %

*Illustration 19: Edge responds to playbackRate updates with undetermined behaviour,but seeking is very accurate*


msie ver. 12.10240 on Win32: Video/ mp4

Frame accurate in: 5.29s Echoless in: 12.29s Frame acc: 38% Echo acc: 1%

*Illustration 20: Video does not provide as good a user experience on Edge,but fixing variable playback rate will amend this*

# 5 CONCLUSIONS

In order for multi device media playback to be timed (synchronized) precisely and reliably, we see a need for better controls of media elements. For quite a few browsers, only minor adjustments or bug fixes are necessary for a reasonably good user experience. It is clear that functioning variable playback rate is the single most important factor. Being able to adjust playback rate reduces stress on media elements and makes for a much better user experience. Seeking is very disruptive for end users.

We also see that a test suite for browsers should address timing issues, defining standardized tests and metrics. As multi device timed playback of media content gains adoption, it will be important that browsers are consistent to ensure that end users can use a variety of browsers and devices without audio/visual artifacts and synchronization issues.

In this report, we have not verified that the actual media output (audio, visual) is aligned correctly with the reported currentTime.  We have however observed that Chrome Dev (47) on Android appeared to be timed correctly, but was actually playing back the wrong content.  In other experiments we have performed, it does appear that browsers in general report their positions correctly relative to audio and video output.

Also note that the MediaSync library is a general purpose library, without optimizations for particular browsers.  It is quite possible to improve timed playback for particular browsers. However, as we have seen browsers change their behaviour between software updates, we opted for a general library that can provide fairly good results on most browsers.  With standardized functionality and testing, this can of course be vastly improved, or even better, solved internally in the browsers (media elements).

Our tests has also not measured the time between a play command is issued and when the playback actually starts.  As media elements are unaware of external timing, they will fail to compensate for time spent connecting, buffering and filling any decoding pipelines.  As such, all these media elements start playing in the wrong place, and need adjustments to compensate.  With timing aware media elements, they could compensate for the startup time internally while seeking, enabling a much nicer user experience.

## 6 REFERENCES

[1] Multi-device Timing CG: https://www.w3.org/community/webtiming/

[2] Timing Object: http://webtiming.github.io/timingobject/

[3] Timed playback mode:http://webtiming.github.io/timingobject/#media-elements-and-the-timing-object

[4] MediaSync library: http://webtiming.github.io/mediasync/

[5] Multi-device video playback: https://youtu.be/lfoUstnusIE