# Combinatorial Testing on Implementations of HTML5 Support

Xi Deng*‡, Tianyong Wu†‡, Jun Yan*†‡ and Jian Zhang†‡
* *Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences*
† *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences*
‡ *University of Chinese Academy of Sciences*
*dengxi15@otcaix.iscas.ac.cn, {wuty, yanjun, zj}@ios.ac.cn*

***Abstract*—The fifth version of HTML standard, which is widely accepted in the diverse landscape of browser vendors and their continuously upgrading releases, requires web browsers to support playback of multimedia natively, instead of by third-party plugins. Works on testing browsers' implementations of the HTML5 support, however, are not efficient enough till now. Regarding a browser's processing on HTML5 element tags of web pages and attributes of multimedia files, this paper treats the browser as a parameterized black-box and employs the combinatorial testing technique to design web pages to test its system behavior. Experiments are conducted on two sets of browsers. The first set includes nine popular ones in both desktop and mobile for discovering the distribution of multimedia related errors, and the second set contains five versions of the same browser for revealing the evolution of such errors. The experimental results indicate that the proposed approach is promising as it can reveal errors in browsers by various vendors and in various versions, and that the upgrades of the browser will not necessarily fix the existing bugs, and even introduce new ones, due to inefficient testing.**

## 1. Introduction

HTML5 designed by Web Hypertext Application Technology Working Group (WHATWG) [1] is the current version standard for HTML (HyperText Markup Language), a programming language used by most web pages and applications. Its richness in interactive capability and consideration for applying in low-powered devices make this standard widely-supported in modern web browsers. As an early draft of this specification is given out in 2011, versions of browsers such as Chrome, Firefox, Internet Explorer, Opera and Safari support HTML5 to a considerable degree.

The primary functionality requirement is to support interactive playback of multimedia natively in each browser, and that is what distinguishes HTML5 from its former versions [2], and also what people will rely on for years to come. In versions like HTML 4, the de facto standard way is to call the third-party Flash plugin or OS native media player. HTML5 alters this mode with new multimedia elements introduced in web pages for media objects to be directed to and for semantic meanings to be specified. Browsers supporting this new standard must enrich their capability in parsing process of HTML page and extend it with a multimedia framework that does decoding jobs.

The demand for a systematic testing for implementation in browsers is sharp, especially considering the diverse landscape of browser vendors and upgrading releases for mobile and desktop devices. Major known testing (regression testing) is performed by browsers' developers each time they are to have new small functional points effected or defects in previous releases fixed. It is not efficient enough because tests are limited to the functionalities of single points, while a little change would impact widely in a system. Complex problems that involve several aspects thus may not necessarily be triggered. Besides, W3C provides a test suite, which includes over 15,000 HTML5 test cases and continues to grow [3]. To have the whole test suite run is time-consuming, and employing it to discover problems about HTML5 multimedia support may be also not efficient enough, since it is designed for general purpose instead of dedicated focus.

We regard browsers' processing of media elements in HTML web page as a system that can be affected by various parameters, which are abstracted from the elements and properties of multimedia files. It is possible that errors lie in some specific parameter combinations, since the elements are with multiple attributes that can be assigned with various values further, and the loading and decoding of media files have a lot to do with the file formats and quality-related parameters of video and audio signal. In our work, we take the browsers as parameterized system and adopt combinatorial testing (CT) technique, which has been proven to be effective in triggering faults coming from combination effect of multiple parameter configurations [4].

Considering the process web browsers deal with media elements, our work focuses on HTML5 media elements for testing the parsing of elements in HTML page and the layout of visible contents, and on media files for further fetching and decoding and rendering. We extract four potential error-prone points from overall consideration of media elements and files, and construct a CT model for each of them. Then a test generation tool for CT is employed to produce compact test suites. We finally construct 162 web pages that contains elements and source files as assigned in test suites and

*Jun Yan is corresponding author.*

978-1-5090-6676-6/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSTW.2017.47

execute them on selected benchmarks (browsers) to detect bug distribution.

Benchmarks are selected for both horizontal and vertical analyses. For horizontal analysis, we choose nine commonly used, top ranking modern web browsers as our experiment objects. The same test web pages are selected to test the behaviors of different browsers against HTML5 multimedia feature. Results show that these browsers are not robust enough to support the combinations of configurable parameters from HTML elements and media files, and all of them have more or less defects on processing the generated HTML5 web pages. We also investigate the bugs inside historical versions of a same mobile browser as vertical analysis, and find that some bugs are associated with multiple versions and along with the upgrade of releases new bugs are introduced. Possible reason lies in the frequent upgrading and inefficient testing of mobile internet applications. These experiments indicate that our work is valuable in improving the quality of web browsers.

The rest of this paper is organized as follows: Section 2 presents the process of web browsers, notions in HTML5 media elements and media containers, as well as an introduction to combinatorial testing. Section 3 shows the details of our approach, and Section 4 is experimental setting and results. Some related work is given in Section 5 and conclusion and further research direction are in last part.

## 2. Background

In this part, we first explain how web browsers work to parse media elements and have source files playing correctly in the plain layout. Then we will have a close look at HTML5 media elements with attributes, and audio and video container structure that would influence browsers' processing. Concepts and methods of CT techniques are also introduced here.

### 2.1. Model of Web Browser

When facing a media element inside an HTML page, browsers will first parse this web page, treat this element as a single node named `HTMLMediaElement` in the DOM tree, collect enough layout information into another node in render tree for visible contents, and then leave these two nodes manipulated separately from other nodes in both trees. The process is shown in Figure 1.

For `HTMLMediaElement` node in DOM tree, a browser will parse the attributes of media element to schedule loading process. Then it creates a bridge mode for scheduling corresponding decoders in the multimedia framework. For the visible content in render tree, a function is called to calculate the intrinsic size information of this content. While the content is an image it is dealt within this render tree; or if it is a media content, it will be passed to the former bridge mode to paint this content out in browser windows.

The playback of media objects does not need to wait until all data of source file get downloaded. Instead, it is a dynamic process. For loaded data, browsers will render it

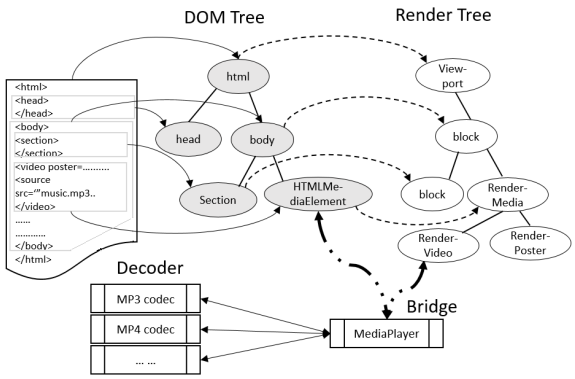



Figure 1. Parsing process of media element

out and keep some information like playback metadata and video data in memory cache for the later coming chunks of data. Thus, what matters in decoding process includes both the information reserved from previous chunks and the information provided in new chunks as well.

### 2.2. HTML5 Media Elements

A basic HTML web page is usually composed of a tree of multiple elements, but each element can be regarded as an individual component, once it has been parsed into the DOM node. For this reason, we can simply focus on the HTML5 media elements themselves to investigate their functionalities.

HTML elements are identified with start and end tag, and contents in between may include other elements, for instance the ⟨video⟩ and ⟨audio⟩ element can have source elements embedded. Each element can have HTML attributes specified, which define desired behaviors or indicate additional element properties. The semantics of elements and attributes and their values should conform to the HTML specification.

In HTML5 specification, media elements include ⟨video⟩ element and ⟨audio⟩ element, both of which can be used for audio and video files. They share attributes including

| | |
|---|---|
| `src` | URL |
| `crossorigin` | use-credentials/anonymous |

where the `src` and `crossorigin` are attributes' name and followed by their values. The `src` attribute of media elements gives the URL of the media resource. URL with different schemes may be dealt differently. In our work we choose four fetch schemes taken into account: HTTP protocols "http" and "https", file transfer protocol "ftp", and local file protocol "file". Besides absolute paths that can be with schemes, relative ones are also included. The `crossorigin` content attribute on media elements is a cross-origin resource sharing settings (CROS) attribute. It could be omitted by default, meaning this request does not apply CORS requests, or assigned with the two values for requests with or without user credentials.

The main difference between ⟨video⟩ and ⟨audio⟩ elements is that the former has a playback area for visual content like videos or captions, with extra following attributes

| | |
|---|---|
| `poster` | URL |
| `width` | Integer |
| `height` | Integer |

where the `poster` attribute gives the URL of an image file that the user agent can show while no video data is available, and the `width` and `height` are to specify dimensions of video playback area. Both poster and two dimension values are important for a ⟨video⟩ element to determine its layout.

An alternative way to specify media source file is to have ⟨source⟩ element embedded between the start and end tag of media element. This element contains attributes that relate with media playback as follows

| | |
|---|---|
| `src` | URL |
| `type` | element "/" subtype |

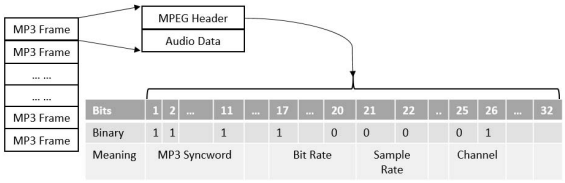
where the extra `type` attribute is to specify the MIME type of the resource file such that web browsers can decide whether the media resource could be handled. It is absent by default and must be a valid MIME type if being assigned. The MIME type begins with "audio" or "video" to specify the audio or video container format, and meanwhile indicating the audio coding format or audio and video coding format inside the container.

## 2.3. Audio Container Structure

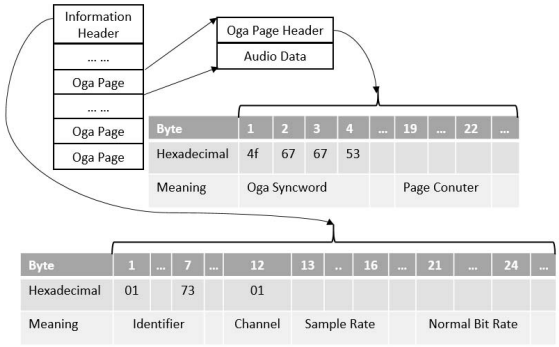
An audio container, or known as an audio file, contains audio bitstream for certain coding formats with some structural information and metadata that is the data about the bitstream. The coding format is the compression algorithm and decides how the digital audio is represented, while the container type describes how different chunks of data and metadata coexist in this computer file.

To obtain an audio bitstream involves converting from analog audio signal to digital data and compressing that digital data to be displayable in computer. The converting process is to sample the electrical voltage in audio signal at fixed time interval, and it is affected by two sampling parameters. The first is sample rate, which measures the number of samples per second, and the second parameter is channel number that depicts how many audio channels are applied to create a surround sound feeling. Sampled data will normally be compressed under certain coding formats like MP3, AAC or Vorbis and finally be turned into audio bitstreams that can be presented in computer with indicators like the bit rate, which is the number of bits that need to be processed per unit of time.

Structural information, like the syncword to identify the format of data and length information, and metadata, like the audio features, are provided along with audio streams inside audio containers, because the compressed data need to be framed for transferring through network. They are organized differently from container to container. In most containers, the above information is provided along with each frame as a frame header, and an example is MP3 container in Figure 2a; in other cases, it is provided as a separate file header, with limited format or size information in frame headers, and the example is Oga container in Figure 2b, where each Oga page is like a frame. For the former case, each chunk of data is well structured and may be easy to recover in case of dropped frames; while in the latter case, faults inside the information header are possible to impede the processing of the whole file.




(a) MP3 Container




(b) Oga Container

Figure 2. Audio container structures

## 2.4. Video Container Structure

Compared with the audio container, a video container needs to manipulate multi-tracks for both video and audio data. Thus it usually contains raw media stream with structure information and a stand-alone part to store all the file information.

Like audio data, video data has two parameters related to its quality: frame rate decides how many picture frames need to be sampled per second, and the parameter chroma sub-sampling rate decides the degree how resolutions of the chroma channels are cut down against that of luminosity channel.

To organize video tracks with audio tracks to be synchronized, a stand-alone file information part is applied in most video containers. The structure is demonstrated in Figure 3, where the file information part stores both video and audio track features, and video and audio frames interweave to form a complete video file. The advantage is

that it enhances the possibility for compatibility and relieves the data payload from a limited size. Frame headers of video may also help in synchronization, but itself may not necessarily contain enough information for the decoder to manipulate the chunk of data. As in Ogv and MP4 container, the video frame header provides encoder with only limited length information, while in WebM container, it has an extra uncompressed header for enough picture information of this frame.

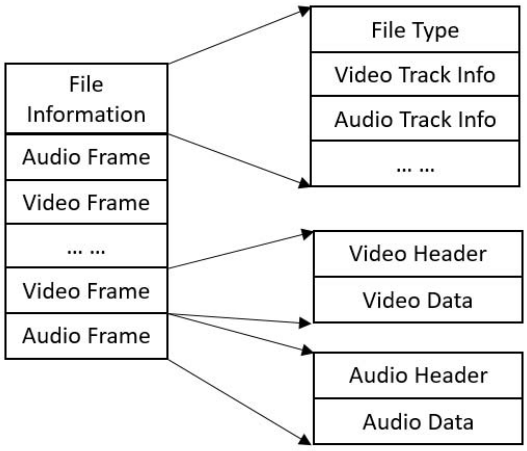



Figure 3. Video Container Structure

## 2.5. Combinatorial Testing Technique

From the above parts we regard browsers as a parameterized black-box. Combinatorial testing is an effective method applied in black-box testing, to generate test suite from models that depict the software under test (SUT).

The SUT has several input parameters which will influence the system behavior that we care about. Each parameter can be assigned a finite set of values. A test case is an assignment to all input parameters, assigning each parameter a valid value. Under some circumstances, parameters in CT model need to satisfy some certain constraints. If the constraints are not properly set, test suite yielded from this CT model will have some test cases be invalid and not executable.

Some research works show that many faults are caused by the interaction of a small number of parameters, typically no greater than 3 [4]. CT technique stands out because of its capability in generating a test suite that covers all value combinations of any $t$ parameters while keeping the test suit smaller enough than that generated via exhaustive testing technique. Here $t$ is called the covering strength.

# 3. Our Approach

In this section, we are to introduce our approach to test web browsers on their HTML5 multimedia support. An overview of our approach is shown in Figure 4, three parts consisted are: constructing the CT models, generating test suites and converting them to concrete test cases, and executing test cases on two sets of benchmarks to detect bugs.

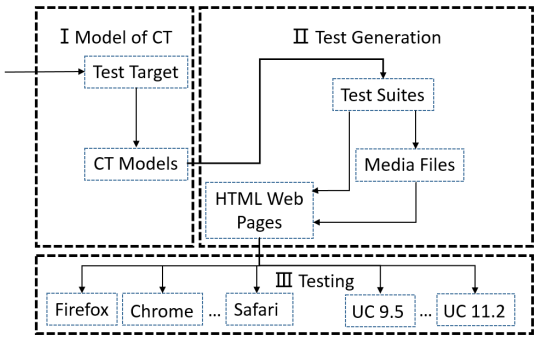



Figure 4. An overview of our approach

We first summarize four potential error-prone points based on learning of the media elements and media containers in Section 2. They are resource loading and visual contents' layout that mainly concerns about combinations of media elements and their attributes, and playback of audio and video containers that focus on their quality-related parameters and mismatches in headers. Then we build an input model for each point, with multiple parameters and parameter values. Those models can be used for generating test suites with combinatorial testing technique. Test suites are then converted to concrete test cases, each of which is a web page that contains media elements directing to media files with certain features. Then we execute those test cases on two sets of web browsers selected to perform horizontal and vertical analyses.

The next two subsections describe the CT models we build based on combinations of media elements with attributes and on media containers' parameters and header information.

## 3.1. Parsing of Media Element Components

As mentioned above, we concern about the browser's process of fetching media resources and obtaining right layout information, so we build two CT models for these concerns, respectively.

**3.1.1. Resource Loading.** In this part we aim to test whether the web browsers can correctly parse the HTML5 page to fetch the media resources according to various URL and type information, and pass them to proper multimedia decoder part and display.

It is possible that the resource type information, the element tag, the declared type in ⟨source⟩ element, if existing, may be different from each other, thus impede the resource fetch and decoder selection. Besides the CORS setting will affect the HTTP request mode thus influence the resource loading process.

We construct the CT model with six parameters. The media elements (EL) are necessarily needed, indicated in the first parameter. Parameter ES stands for situation where the resource URL is in `src` attribute inside the media element. Source element is optional, and if it exists the parameter SS stands for situation where the URL is inside this element, and the TY is the value of `type` attribute that is only present inside it. The CF parameter lists the container formats corresponding to the MIME types in parameter TY. Parameter CO indicates the setting of `crossorigin` attribute.

Note that when the `src` attribute in media element is specified, the source element would be omitted by the web browser. That is, the SS and TY parameters become invalid. To eliminate coverage hole, we add this additional constraint

$$ES \neq not\_set \rightarrow \big((SS = \#) \wedge (TY = \#)\big),$$

where the value "#" represents that the parameter does not take effect, and the value "not_set" means the `src` attribute in a media element is not specified..

Another constraint is for CROS setting that

$$\big((ES = relative) \vee (SS = relative)\big)$$

$$\rightarrow CO = default\_not\_set,$$

since the media object is obviously to be in the same domain with the web page when its `src` attribute is set to be a relative URL string.

The parameters and their values are listed as follows.

| | |
|---|---|
| EL : | {audio, video} |
| ES : | {http, https, ftp, file, relative, not_set} |
| SS : | {http, https, ftp, file, relative} |
| CO : | {anonymous, use-credentials, default_not_set} |
| TY : | {audio/mp3, audio/mp4, audio/ogg, video/mp4, video/webm, video/ogg} |
| CF : | {MP3, AAC, Oga, MP4, WebM, Ogv} |

**3.1.2. Visual Contents' Layout.** The second concern is to test the display of visual contents against various dimension settings. Visual contents are supplied through ⟨video⟩ element, and include video and poster images.

Typically, the size of playback area can be specified by developers through the `width` and `height` attributes, represented in parameter WR and HR in our model. Poster images and media objects also provide size information when the two attributes are not all set. It is possible that the size information of attributes, the poster images and the video objects may not coordinate with each other, thus depreciating the visual effects. So we consider the width and height of poster image respectively in PWR and PHR. Besides, we think the video and poster image formats may also have some influence. Parameter VCF is the container format, and parameter PF is that of poster image files.

The poster and the `width` and `height` attributes can be absent. If the poster is not present, the parameter PWR and PHR must be absent at the same time.

$$PF = None \rightarrow \big((FWR = None) \wedge (FHR = None)\big)$$

We construct the input model as follows, and take levels for these parameters to be ratios of pixel values to the intrinsic width or height of video object. Number 0.1 and 10 are to imply the degree of variation, and number 1 denotes the metric in poster image or element attributes are as the same as that in video objects.

| | |
|---|---|
| VCF : | {MP4, WebM, Ogv} |
| PF : | {PNG, JPEG, GIF, BMP, None} |
| WR, HR, PWR, PHR : | {0.1, 1, 10, None} |

## 3.2. Decoding and Rendering of Media Objects

Audio and video files can have different qualities according to the sample rate, bit rate and other features, of which various combinations may require the decoder to manipulate the file differently. It may not be taken into full consideration when implementing the browsers, thus making the browsers not able to handle certain combinations.

Since decoder commonly manipulates extra information, in the file header or stand-alone external file or headers in each frame, to help the decoding process, it is also possible that the decoder fails to play the files correctly when mismatch happens, which may due to improper transferring of media files from one location to another, or due to the generating process of media files with improper codec.

We extract two tricky points in processing of media files. The first one is on audio files to test native codec's ability of handling audio-related parameter combinations and mismatches that happen between raw audio data and the headers in audio containers, either the first frame header or file header. The second is for video ones, where mismatches may happen between raw video or audio data and headers.

**3.2.1. Playback of Audio Objects.** Parameters taken into consideration for audio files are the audio container formats (ACF), the sampling frequency (SR), and the number of channels (CN). And parameter FSR is the difference of the actual sample rate of audio data against what is indicated in the first frame header of MP3 and AAC container or in the file header of Oga container. Parameter FCN is that of channel numbers. We use "less" to represent a lower level in SR or CN, and "more"" to describe a higher level, we do not apply specific number values because in different headers the way they are indicated may not be the same. Constraints on the supported configurations for each container have been taken into consideration.

Bit rate is an indicator that is important for a codec to get duration or file size. According to bit-rates in each frame is the same or not, we set the BrMode to be "VBR" meaning each frame has a variable bit rate, and "CBR" meaning there is a constant one. Below is the parameters with their values.

ACF : {MP3, AAC, Oga}
SR : {8, 24, 48, 96}
FSR : {less, eq, more}
CN : {mono, stereo, 5.1}
FCN : {less, eq, more}
BrMode : {VBR, CBR}

**3.2.2. Playback of Video Objects.** For video containers, we are to explore not only combinations and mismatches happen between video data and the headers, but also care about mismatches for the audio track between audio data and file header rather than frame header.

The frame rate and chroma sub-sampling rate are depicted in parameter FR and CSS. Matching relations for actual metrics of raw video stream and meta information indicated by headers are in parameter SFR and SCSS. And parameter SSR and SCN depict the differences for audio chunks.

VCF : {MP4, WebM, Ogv}
FR : {1, 25, 50}
CSS : {4:4:4, 4:2:2, 4:2:0}
SSR, SCN, SFR, SCSS : {less, eq, more}

# 4. Experiment Results

From the above four CT models with strength set as two, we use our test generation tool Cascade [5] for test suites generation, and manually convert the test suites to a set of solid HTML web pages with corresponding media files indicated by media elements. Totally 162 test cases are generated. Two sets of benchmarks are selected for horizontal and vertical analyses, as listed in follows, and Table 1 shows the detailed benchmark information, where the abbreviation "Exp" is to specify experiments with the first or the second set of benchmarks.

(i) Horizontal analysis is on nine currently popular user agents according to StatCounter [6], including both desktop and mobile ones. Since Windows serves as a better graphic operating system, browsers update more frequent and have newer versions in Windows compared with Linux; that is why we do not include benchmarks on Linux OS.
(ii) Vertical analysis is on five historical versions ranging from version 9.5 to the most recent version 11.2 of UC mobile browser, which increases its market share gradually as release upgrading.

Then we have all concrete web pages open in the above two sets of benchmarks. For each test case we observe whether the media file can be loaded and correctly displayed in browser windows, and whether it is played without abnormal visual or auditory behaviors. From the results of the whole test suites on every benchmarks, we summarize some conditions of parameter combinations that may trigger faults and categorize them into several kinds, as demonstrated in the following parts. Subsection 4.1 is the test result of horizontal analysis, revealing problems lying in modern browsers from four error-prone aspects, and Subsection 4.2 is the result of vertical analysis which demonstrates problems that in upgrading versions.

Note that some browsers may not support certain media file formats or certain attributes, and they will ignore those unsupported features so that test cases relate with those unsupported features are not taken as failed ones if they do not get passed. For example the IE do not have support for other container types besides MP3 and MP4, thus test results involving other container types are all neglected in our analysis and summary.

TABLE 1. BENCHMARKS

| Exp. | Web Browser | Version | Platform | OS |
|---|---|---|---|---|
| i | Chrome | 52.0 | Desktop | Windows |
| | IE | 11.0 | Desktop | Windows |
| | Opera | 39.0 | Desktop | Windows |
| | Firefox | 48.0 | Desktop | Windows |
| | Edge | 20 | Desktop | Windows |
| | Safari | 9.1 | Desktop | Mac OS |
| | Safari | 10.0 | Mobile | IOS |
| | Chrome | 54.0 | Mobile | Android |
| | UC | 11.2 | Mobile | Android |
| ii | UC | 9.5 | Mobile | Android |
| | | 9.8 | | |
| | | 10.0 | | |
| | | 10.10 | | |
| | | 11.2 | | |

## 4.1. HTML5 Supporting in Modern Browsers

This part is the results for experiments on nine major browsers, concerning four potential error-prone points as introduced in the approach section. Abnormal visual appearances are shown in the Appendix section.

**4.1.1. Resource Loading.** From our experimental results, some browsers fail to load the media objects or display wrong durations of media files under certain test cases. Summarized conditions of parameter combinations that will hamper the loading process are shown in Table 2. Figure 1 and Figure 2 in the Appendix shows two kinds of abnormal behaviors.

The above conditions can be further categorized into three aspects as follows:

1) Failed fetching of data with CROS. For URL that starts with certain schemes, some web browsers fail to fetch the source media with the mode set to cross-origin state, as listed in the first three conditions in table
2) Lack of robustness for mismatched audio coding format. Certain browsers cant handle situations where the audio coding format (i.e. the compression algorithm of audio stream) in media resource mismatch with that declared in HTML5 attribute `type`, as listed in 4th and 5th conditions

TABLE 2. BUGS ON RESOURCE LOADING

| ID | Condition |
|---|---|
| 1 | URL[1] ∈{http, https}, CO ≠ default-not-set |
| 2 | URL = ftp, CO ≠ default-not-set |
| 3 | URL = file, CO ≠ default-not-set |
| 4 | ES = not-set,<br>CF ∈ {Oga, Ogv, WebM},<br>TY ∈ {audio/mp4,audio/mp3,video/mp4} |
| 5 | ES = not-set,<br>CF ∈ {AAC, MP3, MP4},<br>TY ∈ {audio/ogg,video/ogg,video/webm} |
| 6 | URL = ftp [2] |
| 7 | URL = ftp, CF ∈ {Oga, Ogv, MP4} [2] |
| 8 | URl ≠ http, CF = AAC |
| 9 | TY = video/ogg |

[1] URL is not a parameter defined in Section 3, here it stands for `src` defined either in media element or in source element. When given "URL = http" we mean "ES = http or ( ES = not_set and SS = http )".
[2] The problem happens occasionally.

3) Vulnerable loading process with FTP protocol. For URL under "ftp", media loading process is vulnerable in certain browsers. A possible reason is that this protocol demands high-quality network circumstance. The browser may fail to display the media file with correct time duration, especially for the media objects that are in a container format where each frame provides not enough information in its header.

We also count the occurrences of bugs causes by above conditions in top nine browsers in Table 3. In the second row each capital letter with a dot is the acronym of browser's name in Table 1. Here the sign "+" indicates the condition in this row will trigger faults inside the web browser in this column. The results show that defects exist widely in those modern browsers.

TABLE 3. OCCURRENCES OF BUGS ON RESOURCE LOADING

| ID | Desktop | | | | | | Mobile | | |
|---|---|---|---|---|---|---|---|---|---|
| | C. | IE. | O. | F. | E. | S. | S. | C. | UC |
| 1 | + | | + | + | + | | | + | |
| 2 | + | | + | + | | + | + | + | |
| 3 | + | | + | | | | | | |
| 4 | | + | + | | + | | | | |
| 5 | | + | + | | + | + | + | | |
| 6 | + | | + | | | + | + | + | |
| 7 | + | | | | | | | | |
| 8 | | | | + | | | | | |
| 9 | | | | | | | | | + |

**4.1.2. Visual Contents display.** We find that most browsers do not display visual contents as suggested in HTML5 specification, and other implementations that are not specified in the standard yield unpleasing visual effect under certain configurations. The list below demonstrates three kinds of defects.

1) Wrongly calculated size of playback area. HTML5 specification requires the intrinsic width of a ⟨video⟩ element's playback area to be that of the poster frame, if the poster is available and currently presented; otherwise, it is the that of the resource file [1]. The same goes for intrinsic height. The ratio of these two metrics plays the role when the `width` or `height` attribute is missing. What most web browsers do is that they neglect the role of the poster. When not both of the dimension attributes are specified, most web browsers will set the playback area according to the specified one, if it exists, and as well as the aspect ratio of the video resource, rather than first consider that of poster image.
2) Disproportionately resized images. Browsers should fit video contents and poster images at the largest possible size within the playback area, with their aspect ratio being preserved. But as for poster image with GIF format, IE browser alters its aspect ratio to fully fill it into the area, and UC browser resizes image content with all formats.
3) Bad visual experience with control bar. HTML5 leave control bar to be implemented freely within each browser, thus bringing some inconvenience. All browsers set the control bar at the end of the playback area, so when the aspect ratio of playback area is much less than that of video resource, the video will be shown pillar-boxed with the control bar to be far away from the video area. Another situation is that when the intrinsic width is much less than the default value, most web browsers tested here, except Firefox, will compress the width of the control bar unlimitedly, even to the degree where only part of the play button is left. Both situations bring inconvenience and bad visual experience. See Figure 3 in the Appendix.

**4.1.3. Playback of Audio Objects.** Table 4 shows the specific test result for combinations of audio attributes and various mismatches of these attributes between audio streams and headers. Note that a browser may treat media files with mismatched metadata as unsupported ones, for example the Firefox can not play Oga files with wrong sample rate information in header, and we do not regard it as a browser's defect. What we care about is the improper playback or wrong information displayed to users.

Results have shown two kind of mismatches may result in different impact on the audio file. The first kind is mismatch of sample rate, it may result in the wrong duration displayed, or even with sound being wrongly slowed down or speeded up (the first four kings of conditions). The second kind is mismatch of channel numbers, and it may cause the file to be played with distorted sound(the condition with id 5). Other problems are about certain bitrate mode, resulting in unplayable files or wrong durations, when satisfying the last three conditions.

TABLE 4. RESULTS ON PLAYBACK OF AUDIO OBJECTS

| ID | Condition | Result |
|---|---|---|
| 1 | ACF = MP3, FSR = more, BrMode = CBR | Audio is displayed with shorter duration in Opera, IE and Edge. |
| 2 | ACF = MP3, FSR = less, BrMode = CBR | Audio is displayed with longer duration in Opera, IE and Edge. |
| 3 | ACF = Oga, FSR = more | Sound is speeded up with shorter duration displayed in Opera, both Chrome, and UC. |
| 4 | ACF = Oga, FSR = less | Sound is slowed down with longer duration displayed in Opera, both Chrome, and UC. |
| 5 | ACF = Oga, CN $\neq$ stereo, FCN $\neq$ eq | Audio is played with distorted sound in Opera, both Chrome, and UC. |
| 6 | ACF = AAC, FSR $\neq$ eq, FCN $\neq$ eq | In Opera audio need to be clicked twice to play. |
| 7 | ACF = AAC, BrMode = VBR | Audio can not be played in Firefox, or may be displayed with wrong duration in Chrome, Opera and Edge. |
| 8 | ACF = MP3, BrMode = VBR | Audio may be displayed with wrong duration in Chrome in desktop. |
| 9 | ACF = AAC, SR = 96, BrMode = CBR | Audio can not be played in Edge and Opera. |

**4.1.4. Playback of Video Objects.** Table 5 shows the specific test result for combinations of video attributes, and mismatches of these attributes between video stream and metadata part (in MP4 and Ogv container) or frame header (in WebM container), and also mismatches of audio attributes between audio stream and metadata part. Same as in the former part, we care about browsers' abnormal playback under the media files with mismatches.

With regard to combinations and mismatches of attributes for video stream, there are two categories of bugs.

1) Inadequate support for high chroma sub-sampling rate files. Browsers' implementations have a common defect that they have not considered media files with higher quality well. If higher chroma sub-sampling rate mixed with a mismatched metadata part, browsers may have this web page crashed, as shown in the 4th condition.
2) Abnormal playback for certain frame rates and mismatches of video features. Defective behaviors include freezing (the 5th condition), macroblocking with blurred images (the 6th and 7th ones) as displayed in Figure 4 in the Appendix, and accelerating or decelerating of the playback of video files (the following two conditions).

Compared with mismatches happen in frame header in the previous part, we also find that for the same audio format, mismatches happen in stand-alone metadata part may have a greater impact. It may affect the playback of the whole files rather than just influence the duration information displayed.

TABLE 5. RESULTS ON PLAYBACK OF VIDEO OBJECTS

| ID | Condition | Result |
|---|---|---|
| 1 | VCF = MP4, CSS $\neq$ 4:2:0 | Firefox, Opera, IE and Edge play the audio sound with no video image, and Chrome in mobile and both Safari can not play. |
| 2 | VCF = Ogv, CSS = 4:2:2 | Chrome in mobile can not play this video. |
| 3 | VCF = WebM, CSS = 4:2:2 | Opera and both Chrome can not play this video. |
| 4 | VCF = WebM, CSS = 4:4:4, SCSS $\neq$ eq | Opera and Chrome in desktop have this web page crashed. |
| 5 | VCF = MP4, FR = 1 | UC plays this video with the image being freezed for the first several seconds. |
| 6 | VCF = Ogv, FR = 25, SCSS $\neq$ eq | UC in mobile plays this video with blurred image and macroblocking. |
| 7 | VCF = Ogv, FR = 50, SCSS $\neq$ eq | Opera and Chrome in desktop, and UC in mobile play it with blurred image and macroblocking. |
| 8 | VCF = Ogv, SFR $\neq$ less | Opera, Chrome and UC decelerate the playback of this video and display longer duration. |
| 9 | VCF = Ogv, SFR $\neq$ more | Opera, Chrome and UC accelerate the playback. |
| 10 | VCF = MP4, SSR $\neq$ eq | Opera plays the audio with badly stretched sound. |

## 4.2. HTML5 Supporting in Multiple Versions of Same Browser

This part displays and analyzes the testing result for five releases of UC browser from 9.5 to 11.2. Table 6 is the summarized testing results for both HTML5 elements and media files on this set of browsers. And Table 7 shows whether bugs summarized occur in each version. The earlier two versions have fewer "+" signs because they do not have support for AAC and Ogv files, and failed test cases that relate to those unaccomplished functionalities are not taken as bugs.

Summarized results show that although some bugs get gradually fixed along with release upgrading, other bugs may last in multiple versions. For example the bug with id 6 occurs in version 9.8 and is still not addressed in the most recent one. What's more, new bugs get introduced in newer releases. The example is the third condition which is when in ⟨audio⟩ element, and the `src` attribute is assigned with a URL that is under "ftp" protocol and directs to source file belongs to AAC or Ogv formats, UC browser will fail to play this media file. It first occurs in version 10.0, the release in which the range of supported formats extends

TABLE 6. SUMMARIZED RESULTS IN HISTORICAL VERSIONS OF UC

| ID | Condition | Result |
|---|---|---|
| 1 | EL = audio, CF $\in$ {MP4, Ogv, WebM} | Media objects can not be played. |
| 2 | DF = video/ogg | |
| 3 | EL = audio, URL = https, CF $\in$ {Ogv, AAC} | |
| 4 | ACF = MP3, FSR $\neq$ eq | Displayed duration is stretched or compressed. |
| 5 | ACF = AAC, FSR $\neq$ eq | |
| 6 | ACF = Oga, FSR $\neq$ eq | The sound is speeded up or slowed down. |
| 7 | ACF = Oga, CN $\neq$ 2, FCN $\neq$ eq | Audio is played with distorted sound. |
| 8 | VCF = MP4, CSS $\neq$ 4:2:0 | Video can not be played. |
| 9 | VCF = MP4, FR = 1 | Playback is blocked in the first several seconds. |
| 10 | VCF = Ogv, FR $\geq$ 25, SCSS $\neq$ eq | Video been played with blurred image and macroblocking. |
| 11 | VCF = Ogv, SFR $\neq$ eq | Playback is decelerated or accelerated. |

TABLE 7. OCCURRENCES OF BUGS IN HISTORICAL VERSIONS OF UC

| ID | 9.5 | 9.8 | 10.0 | 10.10 | 11.2 |
|---|---|---|---|---|---|
| 1 | + | | | | |
| 2 | | | | | + |
| 3 | | | + | + | |
| 4 | | + | | | |
| 5 | | | + | | |
| 6 | | + | + | + | + |
| 7 | | | + | + | + |
| 8 | + | + | | | |
| 9 | | | | + | + |
| 10 | | | + | + | + |
| 11 | | | + | + | + |

to have AAC and Ogv included. It may due to incomplete consideration about the new formats and insufficient testing before the new release is published to market.

## 5. Related Work

CT techniques have been studied for a long time and yield a lot of practical tools. CASA [7] is a meta-heuristic search solver for constrained covering array generation. ACTS [8] is a greedy solver which has been tested through a real world case study [9], and Cascade [5] is the greedy solver we applied in our work. In our previous work [10], we have explored the correctness of audio players on processing ID3v2 tags in MP3 files with this tool and revealed multiple defects inside the players.

As for HTML5 multimedia, several researchers focus on evaluating and enhancing their performance. Ilias et al. [11] analyzed video performance between Flash video and HTML5 video on three browsers. Meixner et al. did a series of works (e.g. [12]) on playback control, download and cache management of HTML5 videos.

Besides, another interest in HTML5 is the security field. Yoon et al. [13] investigated several kinds of attacks with HTML5 on the new functionalities which are provided by browsers as a replacement of plug-ins like Flash. Mao et al. [14] proposed a method with behavior state machines to detect script-injection attacks on mobile applications that are built by using HTML5.

## 6. Conclusion

Testing web browsers' implementations of HTML5 multimedia support is a valuable work as media is one of the major means of consuming web contents. We employ combinatorial technique that has not been applied to testing for browsers' functions on two sets of web browsers for revealing defects in current browsers and in browser's upgrading releases. With a test suite that can be run on different browsers to make sure they have consistent behaviors, our results demonstrate the extensively existed defects among current popular browsers, and show that occurrence of bugs is unpredictable and inevitable if with no systematic testing. The effectiveness of combinatorial testing on multimedia framework in browsers has been established, and we believe this method can be extended to test enforcement of other specifications that have similar functionality or structure.

## Acknowledgments

The authors would like to thank the anonymous reviewers and Zhiqiang Zhang for their helpful comments and suggestions. This work is supported in part by National Natural Science Foundation of China (Grant No. 91418206) and the National Key Basic Research (973) Program of China (Grant No. 2014CB340701).

## References

[1] WHATWG. (2016) HTML living standard. [Online]. Available: https://html.spec.whatwg.org/multipage/

[2] W3C. (2014) HTML5 differences from HTML4. [Online]. Available: https://www.w3.org/TR/html5-diff/

[3] ——. (2016) Test suites for web platform specificationsincluding WHATWG, W3C and others. [Online]. Available: https://github.com/w3c/web-platform-tests

[4] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, p. 11, 2011.

[5] Z. Zhang, J. Yan, Y. Zhao, and J. Zhang, "Generating combinatorial test suite using combinatorial optimization," *Journal of Systems and Software*, vol. 98, pp. 191–207, 2014.

[6] StatCounter. (2016) Top 9 desktop and mobile browsers from Oct 2015 to Oct 2016. [Online]. Available: http://gs.statcounter.com/#desktop+mobile-browser-ww-monthly-201510-201610

[7] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "Evaluating improvements to a meta-heuristic search for constrained interaction testing," *Empirical Software Engineering*, vol. 16, no. 1, pp. 61–102, 2011.

[8] L. Yu, Y. Lei, M. N. Borazjany, R. Kacker, and D. R. Kuhn, "An efficient algorithm for constraint handling in combinatorial test generation," in *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST*, 2013, pp. 242–251.

[9] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn, "Combinatorial testing of ACTS: A case study," in *Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST*, 2012, p. 30.

[10] Z. Zhang, X. Liu, and J. Zhang, "Combinatorial testing on ID3v2 tags of MP3 files," in *First IEEE International Workshop on Combinatorial Testing , IWCT*, 2012.

[11] I. S. H. C. Ilias, S. B. Munisamy, and N. A. A. Rahman, "A study of video performance analysis between Flash video and HTML5 video," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication,ICUIMC*, 2013.

[12] B. Meixner and C. Einsiedler, "Download and cache management for HTML5 hypervideo players," in *Proceedings of the 27th ACM Conference on Hypertext and Social Media*, ser. HT, 2016, pp. 125–136.

[13] S. Yoon, J. Jung, and H. Kim, "Attacks on web browsers with HTML5," in *10th International Conference for Internet Technology and Secured Transactions, ICITST*, 2015, pp. 193–197.

[14] J. Mao, R. Wang, Y. Chen, and Y. Jia, "Detecting injected behaviors in HTML5-based Android applications," *J. High Speed Networks*, 2016.

## Appendix

The Appendix lists the visible abnormal behaviors in web browsers when they are dealing with error-prone conditions that are listed in Section 4. We take Chrome as the example to demonstrate visible defects.

1) When Chrome can not fetch the source file or when it gets frames of media data but fails to decode them, it will display as Figure 1. Other browsers will give additional notes like "invalid source" in IE.

2) For URL under the "ftp" scheme and directing to media files that with insufficient information in frame headers, it is possible for Chrome to display files with wrong duration or aberrant control bar as follows.

3) Figure 3 demonstrates the situation when the aspect ratio of playback area is much less than that of video resource and the intrinsic width is much less than the default size.

4) Ogv files under certain frame rates with mismatched information about chroma sub-sampling rate will be played with macroblocking, as Figure 4 shows.


(a) Invalid file in audio element


(b) Invalid file in video element

Figure 1. Files can not be played


Figure 2. Files with wrong duration for FTP protocol in audio element

Figure 3. Pillar-boxed appearance with razed control bar

Figure 4. Video with macroblocking