October 17, 2016

# Automatic Detection of Video Content

Haskell Garon

Ertürk Diriksoy

# Automatic Detection of Video Content

Databases that allow third parties to upload content may rely on the third party to correctly indicate the media type for the content. For example, if a user uploads content to an advertising exchange, knowing whether the content is a video file or a display file can be useful for categorization purposes, for implementing a filtering option for a program or user searching the database, or for maintaining statistics or reports on the uploaded content, such as determining an average price paid for the content (such as a cost per thousand impressions ("CPM")). This can be used in enforcing pricing rules that are based on content type, and for reporting and revenue tracking purposes.

Users who upload content to a database can self-identify the media type of the file being uploaded – for example, there may be a field displayed by an uploading graphical user interface ("GUI") in which the uploader can input or select a media type, or can input or select a file type that is known to correspond to a particular media type. Users can also self-identify a media type of the file being uploaded without use of a GUI. For example, a user can so identify a media type of the file via an application program interface ("API") for uploading files. Additionally, some content uses a standardized format that indicates the media type of the content. For example, video ads can use standardized Video Ad Serving Template ("VAST") XML, which may provide identification of these ads as video ads.

However, sometimes users mislabel the uploaded content, or fail to specify a media type, such that no media type is associated with the uploaded file. It may also be the case that the uploaded data is corrupted, or that an upload GUI or API does not provide a means of identifying the media type of the uploaded file, and the media type of the file is unknown to the database. It may also be the case that uploaded content does not use a standardized format associated with a

media type. For example, some video ads use custom JavaScript to load video, rather than using the VAST format, making it more difficult to detect that these ads are video ads. In such a case, the content may be treated as having a default media type, such as being a display file (e.g. a display ad). A display file can be, for example, an image file or an HTML5 file. Furthermore, some uploaded ads are display ads that, when clicked, begin playing a video or fetch a video to be played. Such ads may be treated as static images by default. Without being able to automatically detect content media type, enforcing pricing rules, maintaining media-type-specific statistics, or employing media type filtering is difficult.

This document describes an assortment of techniques for detecting a media type of uploaded content (e.g. detecting non-VAST video ads uploaded to a database). These techniques can be implemented independently or can be implemented in any combination with each other. Generally speaking, three techniques are described herein:

1. Searching for a content-type tag or a file extension tag in an HTTP header of a get video snippet of code.

2. Parsing a DOM tree of a file.

3. Running a JavaScript trace to detect a Mobile Rich Media Ad Interface Definition ("MRAID") call to the method playVideo().

## 1. Searching for a content-type tag or a file extension tag in an HTTP header of a get video snippet of code

In one implementation, linktree detection is used to flag files as video content or as fetch-video content when the file includes code for fetching video content. This can be useful, for example, in flagging as video content display ads that, when clicked, fetch video ads. A library of file types associated with video content can be stored in a database. The library can associate,

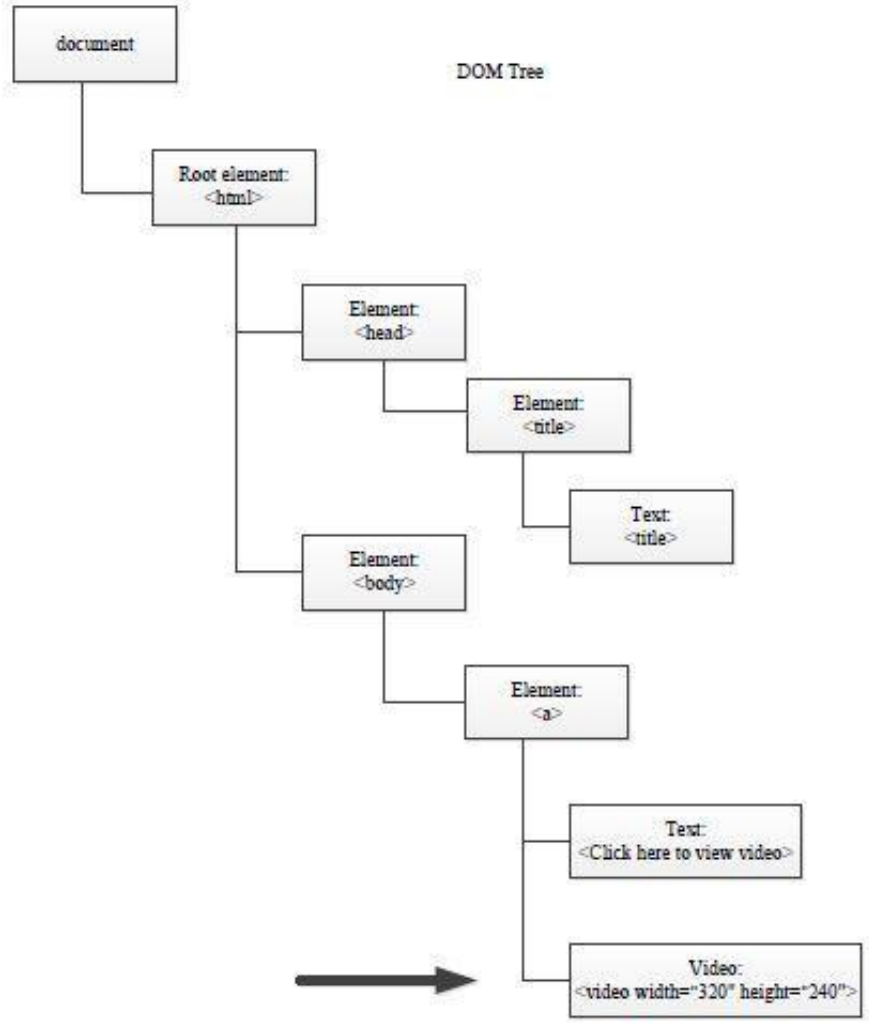
for example, MP4, FLV, WebM, MPEG, MOV, or any other file type extension with video content. Any such file type extension can be considered to be a video file type extension. In some implementations, the video library can include a large number of such file types extensions, such as 100 or more file type extensions. The library can be updated in any appropriate manner, such as via user input.

A server can implement a process that flags a first uploaded file as video content or as fetch-video content based on the first file including code for fetching a second file having a video file type extension. For example, a first file can be uploaded that includes code for fetching a second file, the code including an HTTP header indicating a file extension of the file to be fetched. The server can execute a process that receives all or part of the HTTP header as an input, and detects a file type included in a file type tag of the header, or a file extension tag. The server can attempt to match any detected file type in the HTTP header to a file type stored in the library. If the server finds such a match, the uploaded file can be flagged as video content, or as fetch-video content, via, for example, a metadata object associated with the uploaded file, or by storing the uploaded file in a video-content specific database. Additionally or alternatively, a similar technique can be implemented for a checking an HTTP header for other data, such as content type indicated in a content type tag. If a content type tag in the HTTP header indicates that the file to be fetched is a video file, the uploaded file can be flagged as a video file, or as a fetch-video file.

**2. Parsing a DOM tree of a file**

In one implementation, DOM analysis is used to detect that an uploaded file fetches video content, such as click-to-play video content. A DOM tree corresponding to the uploaded file can be parsed to find instances of an HTML video tag. Upon finding an HTML video tag, the

uploaded file can be flagged as video content, or as fetching video content. DOM analysis can be used to detect display content that does not automatically fetch video content, such as, for example, click-to-play videos with a “preload” attribute set to “none.” If such a video attribute is detected, the file can be flagged as video content, or as fetching video content.




### 3. Running a JavaScript trace to detect an MRAID call to the method playVideo()

In one implementation, a JavaScript trace is used to detect MRAID calls to the method playVideo(). As discussed above, some video ads use custom JavaScript to load videos. The

JavaScript may use MRAID calls, including the MRAID playVideo() call. A trace may allow for tracking a sequence of nested functions, and may be used to search for a nested call to the playVideo() method.

**Implementing Video Detection Techniques**

The above-described techniques can be used, for example, to detect that an ad contains video content or involves fetching video content. These techniques can be used in situations where a publisher who has opted out of video ads for an ad slot holds an auction for the ad slot. Video ads that include video content or that fetch video content can be filtered out of the auction by, for example, an ad exchange. These techniques can also be used for reporting purposes, in that revenue from detected ads that include video content or that fetch video content can be included in reports on video revenue.

Implementations of the techniques described herein are not limited to ad exchange uses. For example, these techniques can also be used by demand side platforms that wish to detect when users upload video ads, and use that detection to determine which publisher properties should be bid on. In other words, a demand side platform can use the techniques described herein for filtering and/or reporting purposes in a manner similar to any of the manners described above, prior to an ad being uploaded to a an ad exchange.

The above-described techniques can be used in any combination with each other. In some implementations, a technique that has the highest probability of success is first performed on an uploaded file, and only if that technique fails to detect video content is a second video detection technique performed. This can help to increase the probability of conserving computer resources by decreasing the likelihood of needing to perform multiple techniques. In other implementations, a technique that uses the lowest amount of computer resources, or a technique

that has a lowest average run time, may be performed first, and only if that technique fails to detect video content is a second video detection technique performed.

**Abstract**

This document describes an assortment of techniques for detecting a media type of uploaded content. Three techniques described above include searching for a content-type tag or a file extension tag in an HTTP header of a get video snippet of code, parsing a DOM tree of a file, and running a JavaScript trace to detect a Mobile Rich Media Ad Interface Definition ("MRAID") call to the method playVideo().