The main objective of this tool is to identify and extract the content of the ads (both text and image) present in a given list of webpages.

1) Tools Used

To extract the ads, following tools and algorithms were used-

- **Selenium Web driver** - It is a browser automation tool which automates web application for testing purposes and allows running automated scripts to perform various tasks. The Firefox web driver was used which automates the Firefox browser. Using the web driver, the web-page was loaded and was parsed to retrieve the advertisements present in that page. (https://code.google.com/p/selenium/wiki/FirefoxDriver)

- **Jsoup Parser**- In order to parse the HTML content fetched by the Selenium web driver, Jsoup parser was used. It is a java library which parses the HTML content and allows extracting and manipulating data using DOM, jquery-like methods. (http://jsoup.org/)

- **Aho-Corasick String Matching Algorithm**- It is used for comparing the URLs with a list of ad-keywords to check if a given link is an advertisement or not.

- Also the Easylist's list of advertisement filters which is also used by many Ad-block plus users was used to identify links which were ads from all the links. Two lists were used for the same.
    - General advertising keyword list- It contains a list of general advertising keywords. It was used to see if a given link is an ad or not.
    - Third-party network list- It contains a list of third-parties which are involved in delivering advertisements across the web. This list was used to check if the advertisement was from a third-party network or not.

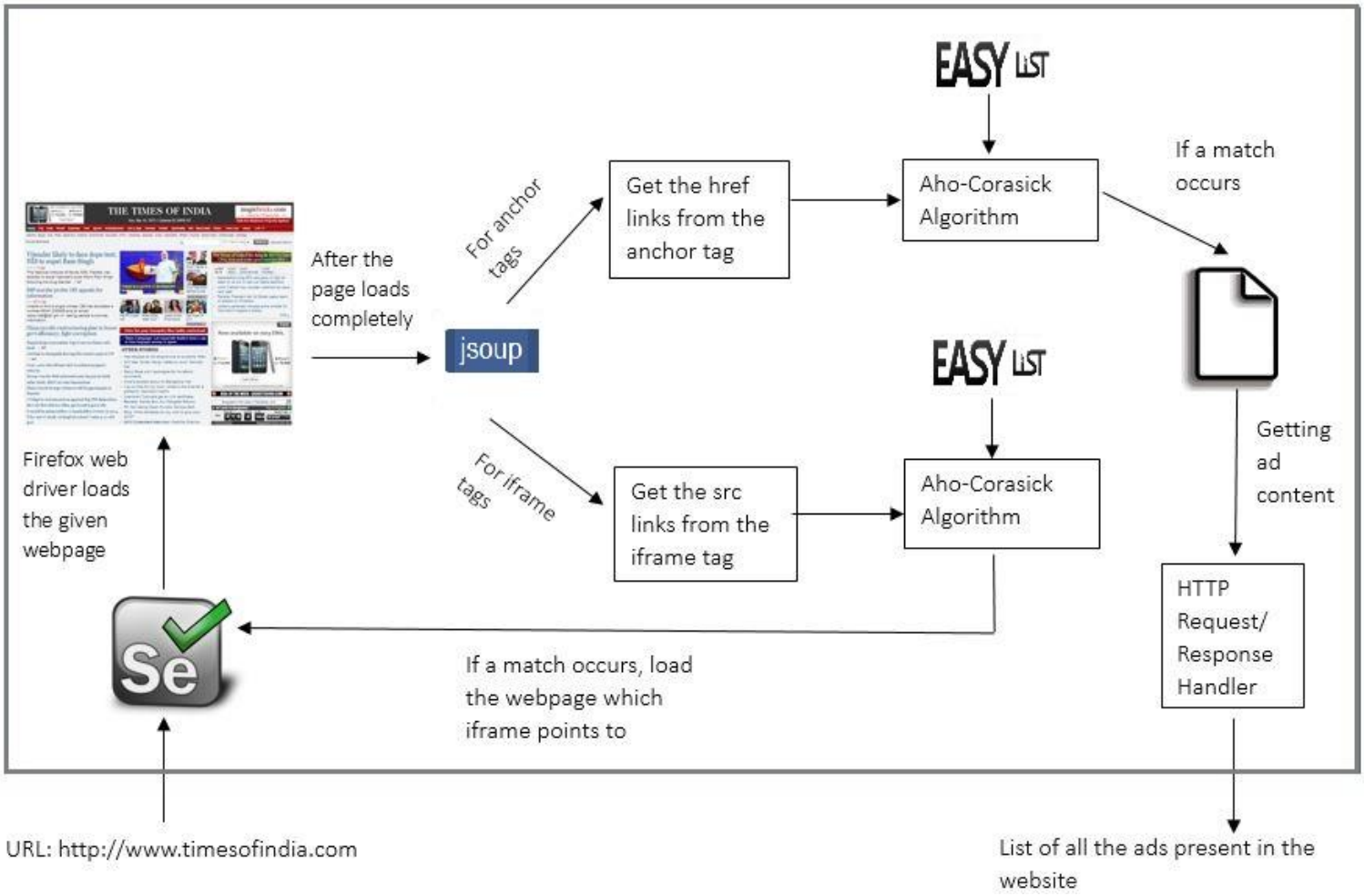
B. Extracting the advertisements

As there could be many links inside a HTML page, in order to identify the links which are ads, the URLs are compared with a list of ad-keywords to identify if a particular link is an ad or not. We used the list of filters from the Easylist subscription which is also used by most of the Adblock Plus users. In order to compare all the URL with the list of filter, Aho-Corasick string matching algorithm is used. These are the steps involved:-

1. Using Selenium's Firefox web driver, the web-page from which the ads are to be extracted is fetched. The Selenium web driver automates the Firefox browser i.e. creates an instance of the browser and waits for the page to load and allows JavaScript to execute if required.
2. Once the page loads completely, the HTML content of the page is fetched from the browser and is parsed using Jsoup to look for all the links present in the webpage.
3. For ads outside iframes, once the page completely loads, we parse the HTML content of the page using Jsoup to look for anchor tags. For each anchor tag, it compares the href link in that

anchor tag with the list of advertisement filters to see if that link is an ad or not. If it is an ad, it stores the link in a file.

4. For ads inside iframes, once the page loads including the contents of iframes, the web driver identifies all the iframes in the HTML page and compares the source links to the advertisement filter list using the Aho-Corasick string matching algorithm. If the string match occurs, the iframe content is parsed to look for anchor tags and then the links in these anchor tags are compared with a list of ad filters. If the string match occurs again, it is considered to be an ad and the link is stored in a file.
5. In order to get the content of these ads, we process HTTP get requests on the advertisement links collected and extract the Meta content, title of these landing web-pages from the HTTP response to identify the content of these advertisements.



3) Classes Used

Following are the main classes which performs the ad-extraction:-

Project- This is the main class responsible for fetching and filtering the URLs from which ads are to be extracted, creating a regex pattern for identifying advertisement links out of all the links, creating and executing threads, instantiating Selenium firefox drivers.

GetUrl- It contains a readhistory() function which reads a list of urls and filters them using url_filter.txt using pattern matching.

FDriver- This class is responsible for loading the webpage, and parsing the page to extract the content of the ads.

GetIMG- This class downloads and stores the images of the image ads.

Get_Landing_Page- This class downloads and stores the html content of the landing pages of advertisements.

4) Detailed Description

Class Project-

- This class takes as input a file-name containing a list of URLs from which ads are to be extracted. It calls the function readhistory() which filters the URLs which are either email sites, social networking sites etc and returns a filtered list of URLs. The file url_filter.txt is used which contains a list of patterns which is used to filter URLs which usually don't contain ads.
- Then regex patterns are created for identifying advertisement links out of all the links present in a webpage. We used two lists for the same- ad-keywords.txt contains general advertising filters while third_party_list.txt contains list of third-parties. Both of these lists have been created using EasyList which is used by most of the adblock plus users (www.easylist.adblockplus.org).
- We created 5 instances of Firefox drivers and 5 threads so that all these drivers can be executed in parallel.

Class FDriver-

- It loads the webpage and saves the HTML content of the loaded webpage in the folder HTML_Pages.
- It then parses the HTML content to look for anchor tags and iframes.
- For each anchor tag,
    - It fetches the href link and matches it with a list of filters to check if it is an advertisement link or not.
    - If it is an image ad, we download and save the image of the ad.
    - For text ads, we get the Display Title, content and URL.
    - We then add a row containing the content of the ad in an excel file using the put() method in the WriteExcel class.

- For each iframe tag,
    - We fetch the src and match it with the list of ad filters to see if the iframe contains an ad or not.
    - Switch the driver's control to this iframe so that it can access the contents of the iframe.
    - We get the content of the iframe and parse it to get all the anchor tags and repeat the same process as above for anchor tags.