Breaking-Bots: Evaluating Websites' Defenses against Web Crawlers

Akanksha Sharma
Nidhi Mundra
Virat Shejwalkar
akankshashar@umass.edu
nmundra@cs.umass.edu
vshejwalkar@cs.umass.edu

ABSTRACT

Global Internet traffic is growing at an estimated rate of 22% per year [5]. However, the Web traffic is dominated by numerous bots crawling the web with different intents, and a very high percentage of these bots have malicious intentions. Hence, websites employ various mechanisms to detect and block the bots. In this report we try to address - if and how websites detect web crawlers? We design experiments using Scrapy [7] based crawler and Selenium [6], a headless browser, based scraper. Selenium imitates real human behavior while visiting urls, hence it's response is assumed the best we can get. In different experiments, Scrapy based crawler has different behaviors such as aggressive, polite, concurrent, etc. We hypothesize that websites would respond differently for different crawler behaviors and detect/block them at different times during crawling. We collect HTTP response codes, error frequencies and messages, and tag frequencies of visited pages to analyze the responses of the websites. Based on the collected data, we categorize websites from highly defensive websites to poorly defensive websites.

KEYWORDS

Web Crawler, Selenium, Scrapy, k-means

1 INTRODUCTION

Web crawling technology is becoming extremely popular with our ever-increasing reliance on the Internet, especially on the search engines. Web crawlers (also known as spiders or bots) are essentially automated scripts trying to fetch data from the web in a systematic manner. To do this, web crawlers collect pages from the web and index them to support search engine queries. Many legit sites (such as Google and Bing) index downloaded pages to improve user experience by allowing users to find the pages faster.

When it comes to the World Wide Web there are both bad bots and good bots; there are crawlers with more sinister intentions. A recent report by Imperva Incapsula reported that 52% of the Web traffic is just bots, and 28.9% among them are the "bad" ones [4], such as impersonators, scrapers, spammers and hacker tools! Some of the effects of such crawlers are – overloading the servers, very high bandwidth consumption, and unauthorized access to content. Because of such intentional or inadvertent effects of crawler activities, it is important to have robust crawler detection techniques in websites. Currently, three main methods are used to detect the Web crawlers [8]:

- Log attributes detection method
- Method based on trap technology
- Method based on the web navigational patterns

In this regard, the aim of this project is to study websitesâĂŹ responses to different requests sent from our configurable web crawler. Using top 500 websites from Alexa.com [9], a paid list of most popular domains, we explored different crawler behaviors based on different User-Agent fields, different IP for different requests, time difference between two requests, number of concurrent requests, etc. We used Selenium - a headless browser to collect pages as a human would see; the aim is to juxtapose the responses that we got from our designed crawler. Based on the results from Scrapy crawler, we apply k-means clustering algorithm on features to categorize top websites into – very highly defensive, highly defensive, defensive, poorly defensive and very poorly defensive websites. Also, the project targets to use different proxies, with which crawler sends `GET` requests, to analyze their effect on the responses received from the websites.

2 BACKGROUND AND RELATED WORK

The following technologies and algorithms were used in the implementation:

Scrapy: Scrapy is a free and open source web crawling framework, written in Python, designed for web scraping and data extraction [7] which does all the heavy lifting that is needed to write a crawler.

Selenium: Selenium [6] is a web browser automation tool. Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that. Selenium Python bindings provide a simple API to write functional/acceptance tests using Selenium WebDriver. Through Selenium Python API you can access all functionalities of Selenium WebDriver in an intuitive way.

k-means Clustering: k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

2.1 Related Work

While analyzing web crawlers it is of prime importance to know the parameters that could potentially affect the way websites look and react to robots. Menshchikov et al. [1] conducted experiments

to design new tools and approaches to protect web resources from automatic information gathering. The experiments they conducted also used different behaviors of crawlers based on the time intervals between requests, HTTP packet structure, content type, errors in queries and crawler's behavioral parameters. They further validate the received data based on the age of the local copy of data and using the binary measure - relevance analysis as a quantification metric. Similar to our experiment, they experimented crawlers with various settings such as depth of crawling, the duration between requests, change of referrer, to mention a few. They did a very extensive research by using various time and structural based parameters and also implemented programs to study behavioral patterns of web crawlers for detection of malicious websites that resulted in automatic detection of crawlers on web resources and subsequently shutting their sources.

Another work by Bai et al. [2] made the use of statistical characteristics of network traffic in real high speed networks to identify differences between legitimate crawlers such as Google-bot and bogus web-crawlers. They used User-Agent field to separate responses for Google-bot and that of an illegitimate bot, and further categorized the responses based on IP addresses of Google crawlers. Using all the collected information they gathered general characteristics of the HTTP requests and based on their analysis they formed a DFA (Deterministic Finite Automaton) to detect HTTP messages which could differentiate between bogus and real crawlers with 95% accuracy. Diakaiakos et al. [3] propose a characterization study of search-engine crawlers based on their HTTP traffic. They used logs of academic servers located in three different countries to analyze crawler activities of five search engines. They identify effects of various behaviors of crawlers based on HTTP GET request frequency, differences in fetched HTML resources, distribution of visits to different parts of a web domain based on importance of the part and inter-arrival time of the crawler requests. Based on these effects, they propose a set of metrics to characterize any crawler behavior; the metrics are format preference, frequency of visit and resource coverage.

To contrast the configurations of our crawler and to make it work in two modes ie. human and bot-like, it was crucial to know the distinction that caused the difference. In [3], to study the behaviour of web crawlers (belonging to Google, Alta Vista, Inktomi, CiteSeer and FastSearch), an analysis was done by accessing the logs of five web servers hosted by academic and research sites in three different countries. By preprocessing and filtering these logs, using a log analyzer, all the HTTP interactions and traffic were observed for the request-reply codes exchanged between the crawlers and the servers. Another set of properties that they used was the timing of crawlers such as the arrival rate and the periodicity of crawler requests, etc. They contrasted the gathered information with the general HTTP trends to observe if the crawlers were being treated differently. Other than HTTP request, they even extended their research to other parameters, one of them being resource size distribution which dealt with differences in the size and type of resources downloaded by the five crawlers. Eventually, they came up with a set of metrics that dominated their research - namely, format reference, the frequency of visits of a crawler on a particular site, resources discovered by a crawler on a particular website. Similar to this, in our experimental setup, we have derived a set of configuration settings, that will distinguish between the two natures of the crawler (human and bot-like), based on concurrency of requests, User-Agent, referrer and other parameters.

All the previous works tend to perform analysis of different crawler behaviors in terms of websites' responses. We would like to extend this direction to perform classification of the websites based clustering of different HTTP responses returned from same websites to different behaviors of crawlers.

3 DATASET

Alexa Top Websites [9], a paid list of the most popular domains was chosen for this project. It is a web service that provides lists of web sites, ordered by Alexa Traffic Rank [10]. Using this web service, developers can page through lists of top sites and incorporate traffic data into their applications. All the domains in this dataset were ranked based on their respective popularity in terms of number of unique client IPs visiting each domain.

4 METHODOLOGY

The main goal of our project is to study websites' responses to different behaviors of web crawlers. Following crawler behaviors are considered while devising experiments:

- time difference in two requests
- different User-Agent fields
- concurrent requests on each domain
- polite crawling which obeys the `robots.txt` file of the domain
- crawling depth

We run a Python implemented crawler, using Scrapy framework, while configuring according to the mentioned behaviors and compare the results with the best possible responses we can get. In order to fetch the best responses, we crawl each website using Selenium – a headless browser.
We perform the same task using proxy of a censored country (China in our case) and study the differences in the responses. Based on the responses as features from different websites to different HTTP requests from crawlers, we identify possible similarities and differences in the mechanisms that the websites employ to detect and block web crawlers. On these features, we apply k-means clustering algorithm in order to group websites based on their similarities, and then classify them into very highly defensive, highly defensive, defensive, poorly defensive and very poorly defensive websites. To see the effect of crawler/s on a website, we measure different aspects of websites' response such as response time to legitimate requests when crawler/s is/are running. This allows us to propose an optimum crawler detection strategy for a website.

4.1 Scrapy Web Crawler

We use Scrapy to crawl all the domains in the dataset [9], and store the responses in sets of – URL, HTTP status, error types and messages (if any) in an output CSV file.

4.1.1 ***Design***. The crawler – `BreakingBotsSpider`, used in experiments has the following components:

(1) `name: Breaking-bot`. This is the default User-Agent.

(2) `start_urls`: The list of URLs from where the crawling of each domain should start. We fetch the complete list of top domains from the dataset [9], and add it in the list of start URLs.
(3) `custom_settings`: This is a class property which overrides the default settings of Scrapy Spider. For each experiment, we configure the settings to extract desirable results.
(4) `visited_urls`: This list stores the domains which have been visited, in order to avoid loops on revisiting the same link.
(5) `request_config`: This sets the configuration parameters including request method (GET/POST), headers, cookies, meta, encoding and don't filter.
(6) `start_requests`: This method overrides the default start_requests method of Scrapy, and starts request for each domain in start_url with specified request_config.
(7) `parse`: This method overrides the default parse method of Scrapy, and yields the response url and status for each request.
(8) `errback`: This method overrides the default errback method of Scrapy, and yields the response url, status, error type and message (if any) whenever Scrapy fails to scrape a particular URL.

The spider reads all the domains from the dataset [9], and starts request for all the each of them. It then parses the response of each successful crawl, and stores the errors for unsuccessful ones in an output CSV file.

4.1.2 ***Configuration.*** The Scrapy settings allows us to customize the behaviour of all Scrapy components, including the core, extensions, pipelines and spiders themselves. We configure following spider settings to bring the aforementioned variations in the crawler behaviour:

(1) `CONCURRENT_REQUESTS_PER_DOMAIN`: The maximum number of concurrent (i.e. simultaneous) requests that will be performed to any single domain.
(2) `DEPTH_LIMIT`: The maximum depth that will be allowed to crawl for any site.
(3) `USER_AGENT`: The default User-Agent to use when crawling.
(4) `AUTOTHROTTLE_START_DELAY`: The initial download delay.
(5) `ROBOTSTXT_OBEY`: If enabled, Scrapy will respect robots.txt policies.
(6) Parameters for choosing a different IP for each request: `RETRY_TIMES`: The maximum number of retries since proxies often fail `RetryMiddleware`: To enable retry of failed requests `RandomProxy`: To enable the use of random proxies. `HttpProxyMiddleware`: This sets the HTTP proxy to use for requests based on its meta value. `PROXY_MODE`: To set the proxy mode to refer a custom proxy or a list of proxies(use one proxy per request or same proxy for all the requests). `PROXY_MODE = 0` For each request a new proxy will be picked from the proxy list

4.2 Selenium Crawler

We use Selenium to visit top 100 domains from the dataset [9]. The purpose of using Selenium is to automate visiting pages and note the domains that are not accessible even by manually entering the urls in a browser. Furthermore, Selenium does not aggressively *crawl* web pages but takes time and loads them as a normal human would do. Hence, we assume that Selenium would seldom encounter bot detection techniques a website has employed. Consequently, we do not employ different behaviors as we have in BreakingBotSpider. We implemented Selenium crawler which can be configured for depth of crawling and websites to crawl.

4.3 Classification

In order to analyze the bot prevention strength and strategies of top websites, we categorize them into very highly defensive, highly defensive, defensive, poorly defensive and very poorly defensive websites based on the results of various experiments performed. For each experimental configuration, we request each website to collect the following features:

- Response status
- Response flags
- Response size
- Number of tags
- Error message
- Error type

Next, we enumerate error message and error type by one to one mapping of values, and clean the data by filling missing values, and removing some records.
These features and data records are then used to cluster top websites into 5 groups, using k-means clustering algorithm. We then analyze these cluster means and cluster data to label them into very highly defensive, highly defensive, defensive, poorly defensive and very poorly defensive ones.

4.4 Experiments

4.4.1 ***Scrapy Experiments.*** We performed various experiments in order to analyze the response of each website on bot-like and human-like crawler behaviors. Following configuration settings of Scrapy were analyzed separately to find the allowed/disallowed behaviors of each domain in the dataset [9]:

(1) **Default:** All the requests were made with a single IP, no depth limit, default concurrency of 10, and default User-Agent as "BreakingBots".
(2) **Low Concurrency High Delay:**
```
CONCURRENT_REQUESTS_PER_DOMAIN = 1
DOWNLOAD_DELAY = 1
RANDOMIZED_DOWNLOAD_DELAY = False
```
(3) **High Concurrency Low Delay:**
```
CONCURRENT_REQUESTS_PER_DOMAIN = 50
DOWNLOAD_DELAY = 0
RANDOMIZED_DOWNLOAD_DELAY = True
```
(4) **High depth limit:**
```
DEPTH_LIMIT = 20
```
(5) **Low depth limit:**
```
DEPTH_LIMIT = 1
```
(6) **Use standard User-Agent:**
```
USER_AGENT = Mozilla/5.0; (Macintosh; Intel Mac
```

Experimental Configuration	Expected Behavior
Default(C1)	Bot-like
Use of a censored proxy(C2)	Human-like
Multiple IPs(C3)	Bot-like
Use of non-sensitive Referrer(C4)	Bot-like
Use standard User-Agent (C5)	Human-like
High depth limit (C6)	Human-like
Low depth limit (C7)	Human-like
High Concurrency Low Delay (C8)	Variable
Low Concurrency High Delay (C9)	Human-like

Table 1: Expected behavior of crawlers

```
OS X 10_12_6) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/61.0.3163.100 Safari/537.36
```

(7) **Use of proxies using downloadermiddlewares for sending each request with a different IP:**
```
RETRY_TIMES = 10
DOWNLOADER_MIDDLEWARES:{
RetryMiddleware: 90,
RandomProxy: 100,
HttpProxyMiddleware: 110
}
PROXY_MODE = 0
```

(8) **Use of a censored proxy:**
```
EXPORT http_proxy=http://120.55.115.50:3128
```

(9) **Use of non-sensitive Referrer:**
```
REFERER = www.google.com
```

The Table 1 lists the expected behaviors of each experiment. Note that, while changing any of the parameters, remaining unchanged parameter values are the same as that in the default configuration.

4.4.2 ***Selenium Experiments.*** Using Selenium crawler we visit 100 top domains from the dataset [9] used, and note the websites that are not accessible by Selenium. We assume that these websites will not be accessible by BreakingBotSpider as well and omit them from analysis. Furthermore, we note the domains that BreakingBotSpider can crawl successfully and omit them from the list of domains to be crawled using Selenium. For this, we assume that if BreakingBotSpider can crawl a domain, Selenium should be able to crawl the domain as well. We have to prune the domain list for Selenium because of it's time consuming way of crawling through the domains.

5 RESULTS & DATA ANALYSIS

In this section we detail the results obtained and corresponding discussion. For each configuration of BreakingBotSpider crawler we collected – HTTP status codes, HTTP error messages (if any), frequency of HTTP error messages and number of tags in a visited page.

5.1 Analysis of frequency of status codes:

The frequency of different HTTP status codes for different configurations4.4 is as show in Table 2. It can be seen from the frequencies that the maximum change in responses of websites occurred when User-Agent and IP fields of the default configuration (C1) were changed. In case of IP, censored IP received very less number of 2xx codes while number of 4xx and 5xx were large in number. On the other hand, when User-Agent field was changed from "BreakingBotsSpider" to a known agent such as "Mozilla", number of 2xx increased and number of errors - 4xx and 5xx reduced. To better visualize the tabular results, bar chart is created as shown in Fig 1.

Configuration	HTTP Status Codes		
	2xx	4xx	5xx
C1	425	21	6
C2	161	13	47
C3	428	20	4
C4	427	20	6
C5	441	9	2
C6	413	21	4
C7	414	21	4
C8	413	21	4
C9	413	21	4

Table 2: Frequency of different status codes for different configurations of BreakingBotSpider

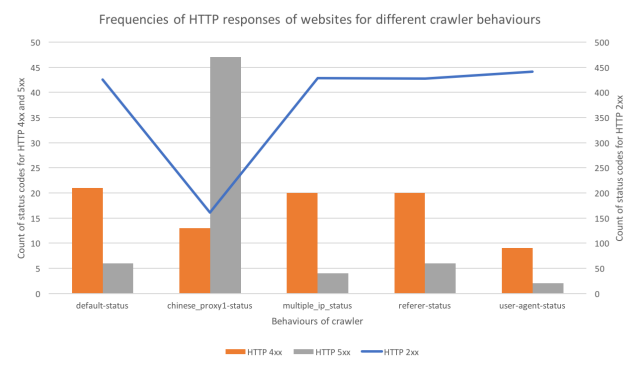


Figure 1: HTTP response frequencies for different crawler configurations; blue line is plotted on secondary y-axis on the right; bars are plotted on primary y-axis on the left

5.2 Analysis of defensive natures of websites

Based on the data collected from the different experiments based on different crawler configurations, we identified 48 features and using K-means algorithm clustered all the top 500 domain in 5 categories depending on their level of defense against web crawlers. The stats are shown in Fig 2.

5.3 Analysis of clustering classification of websites

From a total of 500 websites, we were able to perform clustering classification on 468 of them. Based on the corresponding results, we observed that 84% websites had high or moderate level of bot prevention, 12% of them had very high bot prevention, and only 4% of them had low or very low bot prevention. The websites with very high and high level of bot prevention mainly responded in form of

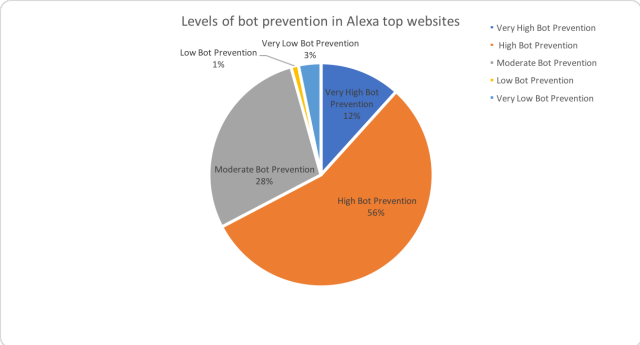


Figure 2: Number of websites in different categories; categories are made according to the level defensive nature of websites towards crawlers

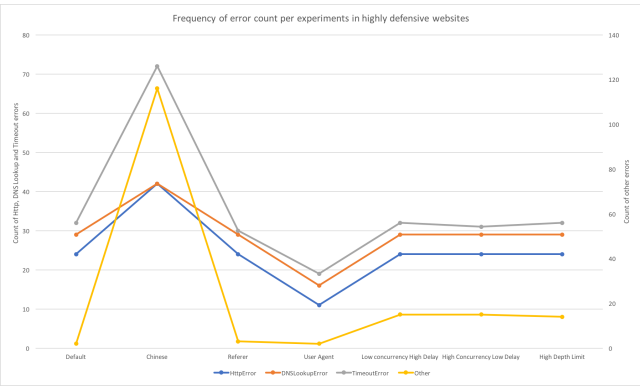


Figure 3: Frequencies of different error types for the highly defensive websites; yellow line is plotted on secondary y-axis on the right and rest are plotted on primary y-axis on the left

throwing timeout errors, DNS lookup errors and HTTP errors. 27% of the website responses were timeout errors, and HTTP and DNS lookup errors were received for nearly 17% websites each. Others responded with unidentifiable errors.

Furthermore, we also analyzed the frequencies and types of different errors that the highly defensive websites returned for different configurations. The Fig 3 shows the relevant results. It was observed that among the recognized errors, the frequency trends of different error types, viz. HTTPError, DNSLookupError and timeoutError, are almost similar except for the censored IP case where number of timeoutErrors are quite larger than the other two.

6 DISCUSSION

In this section, our conclusions are presented with the justification. **Web domains are highly sensitive to User-Agent fields.** With known User-Agent, number of HTTP errors reduced (4xx and 5xx) and 2xx increased. **Websites very highly sensitive to censored IPs.** This is evident from the huge dip in number of 2xx received when the domains are crawled with Chinese IP; most of the packets

Experimental Configurations	Expected Behavior	Actual Behavior
C1	Bot-like	Bot-like
C2	Human-like	Human-like
C3	Bot-like	Bot-like
C4	Bot-like	Bot-like
C5	Human-like	Human-like
C6	Human-like	Human-like
C7	Human-like	Human-like
C8	Variable	Human-like
C9	Human-like	Human-like

Table 3: Actual crawler behavior observed

sent from the crawler might be dropped at one of a censoring router in the path. Depth of crawling does not seem to have let alone effect on the responses received from the domains as with both very high depth limit and depth limit of just 1, the status codes that we received were not very different. This might be because, as far as crawler is using a valid User-Agent and following the *robot.txt* provided, the websites will not take any action against number of pages crawled in the domain. We also did not observe any change in the behavior (in terms of HTTP status codes) of the domains when crawler made 50 requests to a domain at once instead of just 1 with low delay in two subsequent requests. 50 might also be a small number in this case, but we did not try arbitrarily high concurrency to avoid overloading a domain. The Table **??** lists the actual behaviors of each experiment.

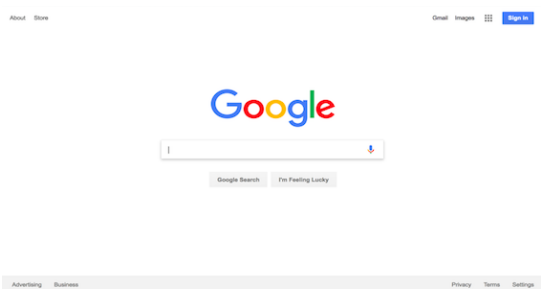
Based on these observations, we conclude that a websites are highly sensitive towards the fields that are in the headers of HTTP GET request of a crawler. This is quite intuitive as well because headers are the first impression of requester from the perspective of a domain.

From clustering classification, we observed that most of the websites had high or moderate level of bot prevention, and nearly 12% of the websites had very high bot prevention. Such high defense aligned with our expectations, as we were working with the top used and the most vulnerable websites around the globe, which makes them bound to integrate an impregnable bot prevention mechanism. The highly bot preventive websites mostly returned timeout errors followed by DNS lookup and HTTP errors. The maximum variance was observed when Chinese proxy was used and User-Agent was changed.

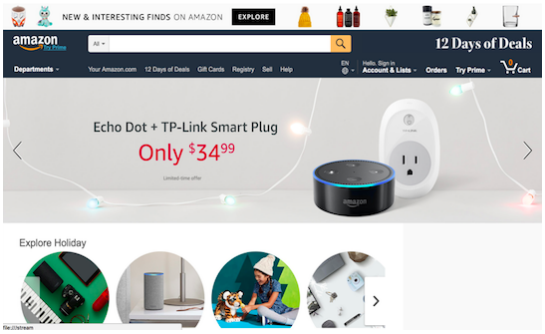
7 CONCLUSIONS & FUTURE WORK

We perform experiments to understand the behavior of web domains towards different types of web-crawlers. To this end, we develop Scrapy based web-crawler and Selenium based scraper with an aim to compare the responses obtained from the domains to different behaviors of the crawler. Behaviors include valid/invalid User-Agent, censored/non-censored IP, aggressive crawling and depth of crawling. We collect HTTP response codes, error codes and tag frequencies of all the pages fetched using crawler and analyze them. We found that the domains are most sensitive to header fields of the HTTP request of crawler such as User-Agent and type of IP;

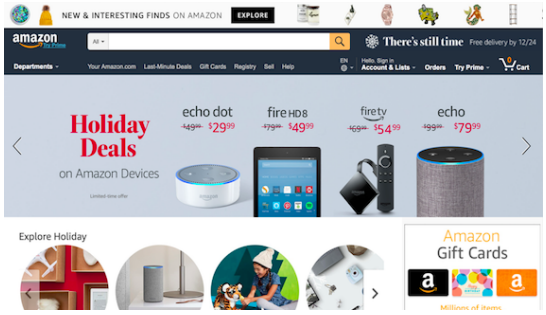
(a) Output HTML for google.com using Scrapy crawler



(b) Output HTML for google.com using Selenium crawler



(c) Output HTML for amazon.com using Scrapy crawler



(d) Output HTML for amazon.com using Selenium crawler

Figure 4: Figures on the left are fetched using crawler with User-Agent as "BreakingBots" and that on the right are from Selenium; it can be clearly seen that google.com is more sensitive than amazon.com for this particular configuration

surprisingly the depth of crawling or concurrency/delay of requests did not have substantial effect on the responses of the domains. Using K-means clustering we categorize the top-500 Alexa websites in 5 categories in increasing level of sensitivity towards crawlers. Points of future work are as follows:

- One of our aims was to experimentally compare the responses of websites to Scrapy and Selenium. Responses acquired from Selenium included dynamic Ajax calls, Javascript and also CSS merged into one single HTML. Whereas HTML responses received from the web crawler included only the raw HTML which made it difficult to compare them on a same basis. One of the ways to do this would be to use images of web pages obtained from both crawler and Selenium, and identify the differences to understand the detection mechanism used. Manually inspecting, it could be said that `google.com` is more sensitive that `amazon.com` because response of amazon to crawler and Selenium is very similar than that of google; this can be seen from the Fig 4.
- Sometimes, on detecting a web crawler, websites slow down their responses towards them. This is an intelligent way of dissipating resources on the crawler's side. Our experimental setup does not account for this scenario.

REFERENCES

[1] Alexander Menshchikov, Antonina Komarova, Yurij Gatchin, Anatoly Korobeynikov and Nina Tishukova, *A Study of Different Web-Crawler Behavior*, Proceedings of the 20th Conference on FRUST Association
[2] Quan Bai, Gang Xiong,Yong Zhao and Longtao He, *Analysis and Detection of Bogus Behavior in Web Crawler Measurement*, Procedia Computer Science 31 (2014) 1084-1091
[3] Marios D. Dikaiakosa, Athena Stassopouloub and Loizos Papageorgiou, *An investigation of web crawler behavior: characterization and metrics*, Computer Communications 28 (2005) 880-897
[4] Igal Zeifman. Bot traffic report 2016. https://www.incapsula.com/blog/bot-traffic-report-2016.html
[5] Jeff Hecht, The Bandwidth Bottleneck That Is Throttling the Internet, 2016, https://www.scientificamerican.com/article/the-bandwidth-bottleneck-that-is-throttling-the-internet/.
[6] Selenium - Web Browser Automation, http://www.seleniumhq.org
[7] Scrapy Official Documentation, https://doc.scrapy.org/en/latest/intro/overview.html.
[8] Dusan Stevanovic, Aijun An, and Natalija Vlajic. Feature evaluation for web crawler detection with data mining techniques. Journal of Expert Systems with Applications, 39:8707âĂŞ8717, 2012.
[9] Alexa.com, https://www.alexa.com, 09 11 2017.
[10] Alexa Top Sites, https://docs.aws.amazon.com/AlexaTopSites/latest/, 09 11 2017.