Websites vs. Web Crawler

CS653: Computer Networks Project

Divyesh Harit
University of Massachusetts Amherst
dharit@umass.edu

ABSTRACT

The behavior and resilience of websites against a web crawler with varying properties have been observed. The same has been done for a headless browser (Selenium headless Chrome) and the results have been articulated. Further headless browsing attempts under varying conditions have been attempted to further obtain new insights.

CCS CONCEPTS

• **Networks** → Network properties

KEYWORDS

Website, Web crawler, Headless browser

1 INTRODUCTION

"A Web crawler, sometimes called a spider, is an internet bot that systematically browses the World Wide Web, typically for the purpose of web indexing (*web spidering*). Web Search Engines and some other sites use Web crawling or spidering software to update their web Content indices of other sites' web content. [1]"

Like everything else, web crawlers can be used for both good and bad. A harmless crawler such as one that indexes the web, or [2], are all good. But they can be used for harmful purposes too, such as malware. Websites, especially the top ones, have to be resilient against such crawlers. They specify their preference in their "robots.txt" file. Here I test some of the top websites and see how they handle our crawler under various User-Agents. The "User-Agent" field is used by the websites to identify the method of access to the site. It would be interesting to see how the sites respond to various user-agents that we provide.

A headless browser provides a simulated browser-like environment without the GUI that can be run from the command line. Like web crawlers, they can be used for both innocuous operations as well as malicious ones. They work great for automated testing, but can also be used to perform DDoS (Denial of Service) attacks. So here, I also test the websites against a headless browser, such as Selenium headless Chrome. I also try to do it again under varying conditions, and observe the results.

2 MOTIVATION

The hope here is that at the end, when we have obtained the results after performing these sets of experiments, we would be in a position to answer some, or all, of the following questions:

1. Which approach performs better? Can we answer why this is the case? By better we mean that we're able to get through to (access) more websites.
2. Is there any trend of blocking or non-blocking of particular websites between the two approaches?
3. Which kind of errors are the most frequent? Why might that be?
4. Is there anything we can do to reduce these errors and get through to more sites?

3 RELATED WORK

There has been some research done when it comes to web crawlers. These include an Ontology-based web crawler [3], that provides a path for the crawler by semantic matching of already crawled web pages and ontologies. Using PostGreSQL, previous studies [4] have shown that the power of the crawlers can be increased. Spreading the operations of a crawler across multiple machines, as done in one of the works [5], has shown drastic increase in performance.

There's some lesser known works when it comes to Selenium. [6] listed out guidelines and tips on doing automated testing for web software. [7] performed actual web application testing experiments on Selenium under various modes. As demonstrated by [8], Selenium can be used for malicious purporses such as performing a brute force attack on a website.

4 EXPERIMENTS

I perform the following sequence of steps in my experiments:

4.1 OBTAINING TOP WEBSITES LIST

The first step is to actually know what websites we want to access. What better way than to target the top websites of the world? We simply need the names of these sites for this. One way of obtaining this to query AWS Alexa for the top websites. Alexa Top Sites [9] is a service owned by Amazon that maintains (and daily updates) list of hundreds of thousand top sites in the world, and returns the name of one site for $0.0025.

This requires creating a HmacSHA256 signature and sending a request with the signature, timestamp, user's unique access key ID and secret access key. Instead of trying to reinvent the wheel and

spend a lot of time on one of the more trivial aspects of the project, I decided to utilize AWS' openly available Java code instead of coding everything from scratch. I modified this code to get 500 sites, and store these sites names in a .csv file. I then load this .csv every time from my main Python 3.6 code. It looks something like:

	Site
2	google.com
3	youtube.com
4	facebook.com
5	baidu.com
6	wikipedia.org
7	yahoo.com
8	google.co.in
9	reddit.com
10	qq.com
11	taobao.com
12	amazon.com
13	tmall.com
14	twitter.com
15	vk.com
16	google.co.jp
17	live.com
18	instagram.com
19	sohu.com
20	jd.com

Figure 1: Example of the top_sites.csv that is loaded in every iteration of the code.

4.2 CREATING CRAWLER AND SENDING REQUESTS

This is one of the core steps of the project. I utilize a very clean Python library for this task: Requests. Requests allowed me to easily build HTTP headers, vary my user-agent field, and send requests. Here, while sending the requests, I set the user-agent as a custom user-agent that I create:

"DivBot / https://sites.google.com/view/web-crawler-cs653"

This is in line with web crawler best practices that I came across: Name of the crawler, followed by some information/link that describes the crawler in more detail. Here, the link I've provided here is the crawler web page that I created, part of which reads:

"If your website was just greeted by our crawler, don't worry! It's just a grad school project and we mean no harm. If you still wish to not be bothered by the crawler, please contact me at: my_email_address, and we will take your website off our list. The main goal of this project is to see how/if top websites block web crawlers. An additional goal is to compare these observations with that of a headless browser, such as Selenium. The project goals are described in more detail here: link_to_project_proposal."

Challenges faced in this step:

1. One initial mistake on my part was not setting a timeout when sending the requests. Thus, for some sites, simply sending the request kept waiting for an indefinite time, let alone getting a response. I then set a conservative timeout of 5 seconds, following which I move on to the next site.
2. Sometimes just sending a request would fail, and since max_retries is set to 0 by default by the underlying urllib3 that requests library uses, the program would crash. I then created a session for each request and set its max_retries to 5, and implemented error handling to deal with the error if it fails even after 5 retries.

4.3 USER-AGENT VARIATIONS

Since websites use the user-agent field to identify whether it's a browser, crawler, or something else that's trying to access it, it's crucial to experiment with this field to see if we get different, same, or interesting results. As stated initially, I used a custom user-agent for the first set of experiment.

1. An empty user-agent: ""
2. A browser-like user-agent (actually the exact same user-agent used by my browser, as noted from the inspect network tab of Chrome), to simulate a browser access: "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36"

4.4 RECORDING CRAWLER RESPONSES

If the site responds with a 200 HTTP code, we know that everything is A-Okay. If not, something went wrong. It might be a server-side error (500-level), a user-side error (400-level), or we weren't able to get through because of some other reason. Whatever the reason is, I record it, and increment a FAILED counter by 1. I then note down the various kinds of failures, and plot the results.

4.5 ACCESS WEBSITES USING SELENIUM HEADLESS BROWSER

Now that we're done with the crawler portion, we can focus on selenium. Here also, we're mostly concerned with the response code returned by the site. And unfortunately, selenium does not inherently provide a way to record the response of the site visited. To get around this limitation, we are forced to resort to using another library, selenium-requests [10]. This library extends the Selenium Webdriver for Firefox, Chrome and PhantomJS to provide the additional functionalities of requests. I realize that this may not the most accurate representation of Selenium, and that the results obtained here might not be completely true to Selenium, but in order to actually record the responses, this approach appears necessary.

I define the Chrome webdriver local path, and also define the Chrome path. For every request, I create a Chrome webdriver initialized with this webdriver path. To the Chrome webdriver, I add the following arguments:

1. "--headless", to enable headless browsing.
2. "--window-size", and set it equal to my machine's display resolution. This is to ensure Selenium operates

in the background and 500 windows don't open up while sending the requests and crash the machine.

4.6 RECORDING SELENIUM RESPONSES

The steps here are similar to the final steps of the crawler portion. If the site responds with a 200 HTTP code, we know that everything is A-Okay. If not, something went wrong. It might be a server-side error (500-level), a user-side error (400-level), or we weren't able to get through because of some other reason. Whatever the reason is, I record it, and increment a FAILED counter by 1. I then note down the various kinds of failures, and plot the results.

4.7 SELENIUM VARIATION EXPERIMENTS

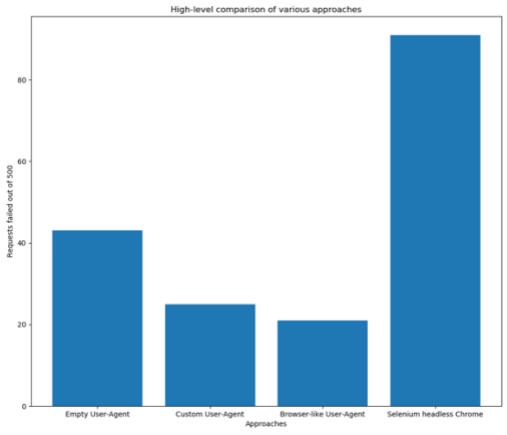
After receiving an unexpectedly high number of failures for Selenium (more about this in section 5), like the user-agent in crawler portion, I tried to experiment with selenium requests:

1. In one version, I let the program sleep for a few seconds between every request. This is because after looking deeper into some errors, I read that they can arise because of too many requests in a short amount of time and the server just blocks your access. So, sleeping between each request seemed like a good idea.
2. In another version, I tried to change the IP of the source of the requests. A lot of top 500 sites are Chinese and Russian websites, and quite a few of them failed to give a 200 response. So, I tried to send the requests using Chinese and Russian IPs. I used the following link to obtain IPs and ports:
   - ➔ Chinese IPs: http://gimmeproxy.com/api/getProxy?country=CN
   - ➔ Russian IPs: http://gimmeproxy.com/api/getProxy?country=RU

Results for both the approaches are provided in the next section.

5 RESULTS AND DISCUSSION

Here's an overview of the performance of the approaches:



The individual results are described in more detail below.

5.1 CUSTOM USER-AGENT CRAWLER

Our custom user-agent gives decent results. It fails for a total of 25 sites out of 500. The breakdown is provided below:

Max Retries Exceeded: ytimg.com, banvenez.com, wiley.com, livejournal.com, bp.blogspot.com, twimg.com, cloudfront.net, googleusercontent.com, uidai.gov.in, alicdn.com
Timeout: bestbuy.com, cnblogs.com
403: ebay.de, fiverr.com, hicpm5.com, application-77my.com, udemy.com, adexchangeperformance.com, buzzadnetwork.com, hitcpm.com, fwbntw.com, lie2anyone.com, media.tumblr.com
500: siteadvisor.com
503: Kissanime.ru, hdfcbank.com
The proportion of errors are summarized through this graph:

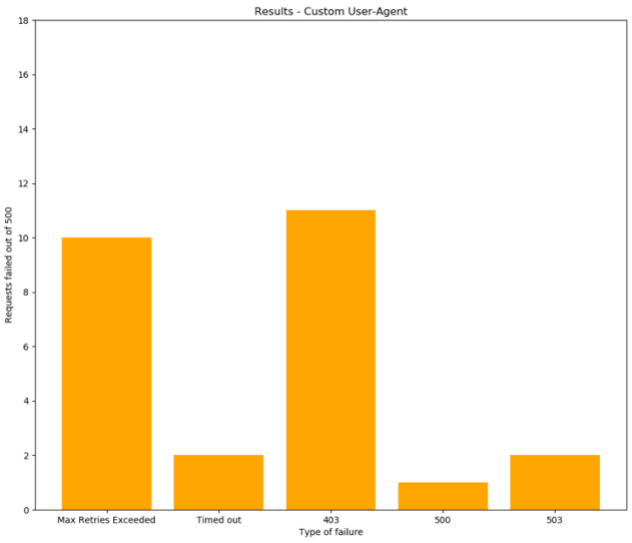


Figure 3: Custom user-agent results

As we will see, the performance here is better than that of an empty user agent, but not as good as a browser-like agent. Here we don't see that many types of errors; Max Retries Exceeded and 403 dominate.

5.2 EMPTY USER-AGENT CRAWLER

The empty user-agent gives poorest results out of the 3 crawler user-agent variations. It fails for a total of 43 sites out of 500. The breakdown is provided below:

Max Retries Exceeded: aliexpress.com, googleusercontent.com, cloudfront.net, us.blastingnews.com, twimg.com, bp.blogspot.com, alicdn.com, youdao.com, theverge.com, wiley.com, banvenez.com, ytimg.com, uidai.gov.in
Timeout: bestbuy.com, so.com
Connection aborted without response: shopify.com
400: soundcloud.com
401: Alibaba.com, pornhub.com

403: 4chan.org, adexchangeperformance.com, zipnoticias.com, rarbg.to, www.huanqiu.com, buzzadnetwork.com, hitcpm.com, fiverr.com, fwbntw.com, media.tumblr.com, lie2anyone.com, deviantart.com, onoticioso.com, hicpm5.com, udemy.com
404: independent.co.uk
408: application-77my.com
429: speedtest.net
463: roblox.com
503: kissanime.ru, hdfcbank.com, gyazo.com
520: Cambridge.org
The proportion of errors are summarized through this figure:

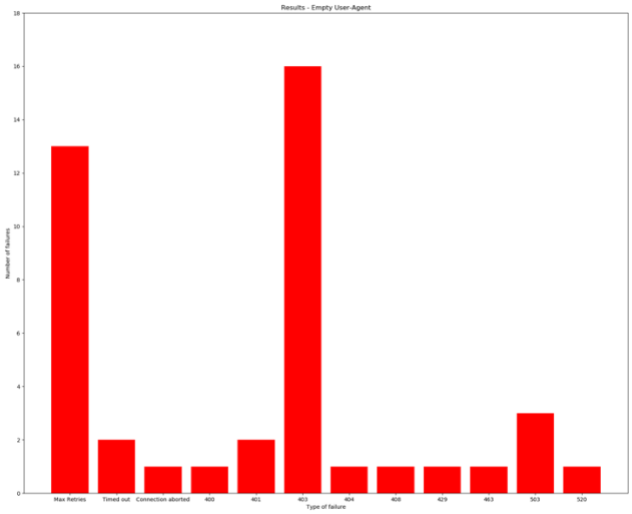


Figure 4: Empty user-agent results

Here we see a huge variety of errors! As stated before, performance is worse than the custom user-agent. Here also, 403 and Max Retries Exceed dominate. We see one weird 463 (an error I didn't even know existed!) for roblox.com. Here, we see the highest proportion of 403s out of the 3 crawler approaches.

5.3 BROWSER LIKE USER-AGENT CRAWLER

Our browser-like user-agent gives the best results out of the 3 crawler user-agent variations. It fails for a total of just 21 sites out of 500. The breakdown is provided below:

Max Retries Exceeded: googleusercontent.com, cloudfront.net, twimg.com, bp.blogspot.com, alicdn.com, wiley.com, banvenez.com, ytimg.com, uidai.gov.in
403: adexchangeperformance.com, buzzadnetwork.com, hitcpm.com, fwbntw.com, media.tumblr.com, lie2anyone.com, hicpm5.com
408: application-77my.com, hdfcbank.com
500: siteadvisor.com
503: kissanime.ru
The proportion of errors are summarized through Figure 5.

Here we see the best results out of the 3 crawler approaches. This good performance can be attributed to, well, acting *almost* like a browser when accessing! Obviously, it's not as good as an actual browser access, but it comes pretty close. It turns out to be the only crawler approach with more Max Retries Exceeded than 403s. This tells us that actual *errors* are less, but failed connection attempts are more in this case.

5.4 DEFAULT SELENIUM-REQUESTS

The results for selenium were a bit surprising. It "failed" for a total of 91 out of 500 sites. Most of them were Max Retries Exceeded, we even see 3 different kinds of this type of error! The breakdown is as follows:

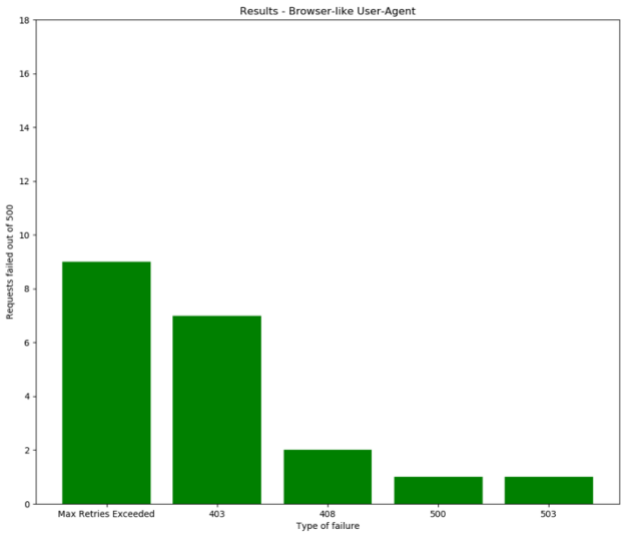


Figure 5: Browser-like user-agent results

Max retries - NewConnectionError: qq.com, alipay.com, hao123.com, microsoftonline.com, imdb.com, espn.com, googleusercontent.com, nicovideo.jp, tianya.cn, pixnet.net, amazonaws.com, youth.cn, xinhuanet.com, thestartmagazine.com, cloudfront.net, globo.com, force.com, twimg.com, china.com.cn, bg.blogspot.com, ikea.com, kakaku.com, huffingtonpost.com, caijing.com.cn, huanqiu.com, codeonclick.com, wordreference.com, ltn.com.tw, asos.com, kompas.com, livedoor.com, varzesh3.com, ign.com, hotstar.com, reverso.net, rednet.cn, samsung.com, haber7.com, ndtv.com, livedoor.jp, dmm.co.jp, go.com, repubblica.it, elmundo.es, wiley.com, espncricinfo.com, banvenezh.com, marca.com, ytimg.com, cambridge.org
Max retries - SSLError (CertificateError): soso.com, tribunnews.com, naver.com, onlinesbi.com, gmw.cn, dailymail.co.uk, china.com, people.com.cn, metropcs.mobi, steampowered.com, 39.net, breitbart.com, thesaurus.com, bilibili.com, setn.com, target.com, fromdoctopdf.com, maka.im, state.gov, ups.com, telegraph.co.uk, uidai.gov.in, naver.jp, seasonvar.ru, fedex.com, easypdfcombine.com, rediff.com, primosearch.com, lie2anyone.com, epochtimes.com
Max retries - SSLError (SSLError: Bad Handshake): hitcpm.com403

Timeout: rambler.ru
Connection Aborted: alicdn.com
403: adexchangeperformance.com, buzzadnetwork.com, udemy.com, fiverr.com, fwbntw.com, media.tumblr.com
404: csdn.net
408: application-77my.com
500: taboola.com
503: kissanime.ru, hdfcbank.com
521: ntd.tv

The proportion of errors are summarized through Figure 6:

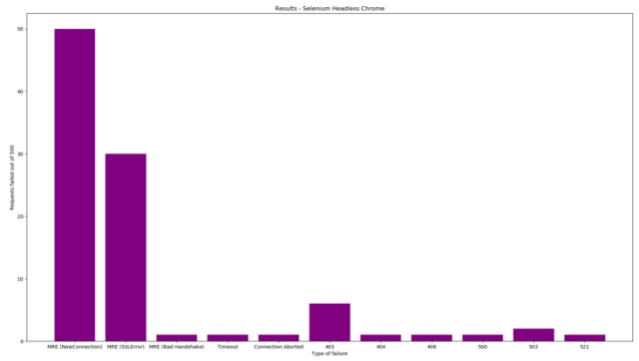


Figure 6: Selenium headless Chrome results

The results are very surprising. This approach "fails" for ~1/5 websites. It should be noted here that actual *errors* aren't that high, but failed connections due to Max Retires are; they make up for around 90% of all our errors. As noted earlier, we're limited in our implementation. These high number of max retries can be attributed to the method/library that we're using: selenium-requests. Some inherent property of this library is preventing successful connections to these sites. This holds true because I ran the experiment again with just Selenium webdriver, no selenium-requests. I use the "get_screenshot_as_file" method of the webdriver to store screenshots and manually checked the results for the sites that gave max retries for selenium-requests. They worked fine here! Like stated before in section 4.5, the results here may not be true representation of Selenium works, and it turns out to be the case for Max Retries Exceeded.

5.5 FAILED SELENIUM ATTEMPTS

Before going through the screenshots of actual Selenium, I looked more into Max Retries errors. A couple articles noted that it can be from the server side, when you try to send too many requests in a short duration and you just get blocked. To try and get around this, I let the program sleep for 3 seconds between each request. Unfortunately, there was absolutely no change in results: Max Retries were still prevalent.

Another thing I tried was trying to send using Chinese and Russian IPs, as a lot of top 500 sites were from these countries, and I wondered if the errors would mitigate by doing this. Unfortunately, after obtaining the IPs and ports from the method stated in section 4.7, and setting the proxy fields of the webdriver, no request was going through, the program just waited to send it. Maybe the changing of IP was incompatible with selenium-requests, as this approach works with Selenium on its own.

6 CONCLUSION

In this project, I've attempted to explore the behavior of websites, crawlers and Selenium. The project gave some interesting results, particularly the failures with selenium, but I attribute them to the nature of the alternative library used (selenium-requests) instead of Selenium itself. A total of 6 sites always give a 403, even from a browser, and they can be excluded from our results. In light of the results and this fact, I conclude that websites, at least the top ones, generally allow access to them, whether the user-agent is an empty one, our custom one with the link for more info, or a browser-like one. Maybe this is because we were simply innocuously trying to access their page, or maybe they also take some other factors into account other than just user-agent. An interesting extension of this project could be looking deeper into other fields that could be taken into account and varying them and comparing results with the current project.

A HEADINGS IN APPENDICES

Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction
A.2 Motivation
A.3 Related work
A.4 Experiments
A.2.1 Obtaining top websites list
A.2.2 Creating crawler and sending requests
A.2.3 User-Agent variations
A.2.4 Recording crawler responses
A.2.5 Access websites using Selenium headless browser
A.2.6 Recording Selenium responses
A.2.7 Selenium variation experiments
A.5 Results and Discussion
A.3.1 Custom User-Agent crawler
A.3.2 Empty User-Agent crawler
A.3.3 Browser-like User-Agent crawler
A.3.4 Custom User-Agent crawler
A.3.5 Default Selenium requests
A.3.6 Failed Selenium attempts
A.6 Conclusion
A.7 Acknowledgements
A.8 References

ACKNOWLEDGMENTS

This project would not have been possible without the guidance of Prof. Phillipa Gill – Thank you for the extremely valuable progress report feedback. A big thank you to Keen Sung, who suggested selenium-requests, without which it would have probably taken a lot of manual work to comprehend selenium results. And finally, thank you Internet, without which none of this would have been possible!

REFERENCES

[1] Web Crawler, Wikipedia: https://en.wikipedia.org/wiki/Web_crawler

[2] Project Arachnid, a crawler that crawls for digital fingerprints of child pornography images: https://www.cybertip.ca/app/en/projects-arachnid

[3] Ontology-based Web Crawler: http://research.ijcaonline.org/volume44/number18/pxc3878724.pdf

[4] YeSQL Web Crawler: https://arxiv.org/pdf/1212.5633.pdf

[5] Distributed Web Crawler: http://engineering.nyu.edu/~suel/papers/crawl.pdf

[6] Automated Software Web Testing with Selenium: https://www3.nd.edu/~veoc/resources/Papers/Selenium.pdf

[7] Research Study on Web Application Testing using Selenium Testing Framework: https://goo.gl/HXXwM9

[8] Selenium for Web App Pen-testing: https://www.frameloss.org/2011/07/23/selenium-for-web-app-pentesting/

[9] AWS Alexa Top Sites Service: https://aws.amazon.com/alexa-top-sites/

[10] Selenium-requests 1.3: https://pypi.python.org/pypi/selenium-requests/