Who is asking? Humans and machines experience a different scholarly web

Martin Klein
Los Alamos National Laboratory USA
mklein@lanl.gov
https: // orcid. org/ 0000-0003-0130-2097

Lyudmila Balakireva
Los Alamos National Laboratory USA
ludab@lanl.gov
https: // orcid. org/ 0000-0002-3919-3634

Harihar Shankar
Los Alamos National Laboratory USA
harihar@lanl.gov
https: // orcid. org/ 0000-0003-4949-0728

Libraries and archives are motivated to capture and archive scholarly resources on the web. However, the dynamic nature of the web in addition to frequent changes at the end of scholarly publishing platforms have crawling engineers continuously update their archiving framework. In this paper we report on our comparative study to investigate how scholarly publishers respond to common HTTP requests that resemble typical behavior of both machines such as web crawlers and humans. Our findings confirm that the scholarly web responds differently to machine behavior on the one hand and human behavior on the other. This work aims to inform crawling engineers and archivists tasked to capture the scholarly web of these differences and help guide them to use appropriate tools.

Scholarly Web, Web Crawling, Scholarly Publishing Designing and Delivering Sustainable Digital Preservation; The Cutting Edge: Technical Infrastructure and Implementation

I. Introduction

Web archiving has become an essential task for libraries worldwide. However, scoping this endeavor is a non-trivial issue. Fortunately, academic libraries may take guidance from a collection development policy that specifies, among other aspects, which part of the web to focus on for crawling and archiving. The collection development plans from Stanford University Libraries [1] and from Purdue University [2] are great examples that can help steer libraries' archiving efforts. National libraries, for example the National Library of Finland [3] or the Library and Archives Canada [4], on the other hand, often have a mandate to collect and archive (national) web resources. Ideally, those documents also narrow down the scope and provide direction as to which pockets of the web to focus resources on. For both types of libraries, the scholarly web typically is in scope of their archiving efforts. This can, for example, be because they are authored by representatives of the university/country or because it is understood that members of the respective communities will benefit from the long-term availability and accessibility of such resources. We refer to the scholarly web as a space where scholarly resources can be deposited (a pre-print server, for example) and where these resources have a URL and are accessible to a reader. For the remainder of this paper, we limit our definition of the scholarly web to the latter aspect, a part of the web from which scholarly resources can be consumed.

These individual web crawling and archiving efforts are organized and conducted by libraries themselves. They are more narrow in scope, smaller at scale, and run with fewer resources compared to, for example, large programs such as LOCKSS[1] or Portico[2], which are specialized in the preservation of journal publications. However, with the constantly changing nature of the web [5]–[7] and the realization that online scholarly resources are just as ephemeral as any other resource on the web [8],

[1] https://www.lockss.org/

[2] https://www.portico.org/

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

[g], libraries are facing the challenge to keep up with their crawling and archiving frameworks.

When it comes to identifying scholarly resources on the web, the Digital Object Identifier (DOI)[1] has become the de facto standard. In order to make a DOI actionable in a web browser, the recommended display is in the form of a HTTP DOI e.g, https://doi.org/10.1007/ 978-3-540-87599-4_38. When a user dereferences this HTTP DOI in a web browser, the server at doi.org (operated by the Corporation for National Research Initiatives (CNRI)[2]) responds with a redirect to the appropriate URL at the publisher. From there, the browser often follows further redirects to other URLs at the publisher and eventually to the location of the DOI-identified resource. The HTTP redirection is done automatically by the browser and the user often does not even notice it. In the above example the browser redirects to the article's Springer landing page hosted at https://link.springer.com/chapter/10.1007%2F978-3-540-87599-4_38. This scenario is very typical in a way that the DOI identifies an academic journal article and, unlike the HTTP DOI, the landing page itself is controlled by the journal's publisher.

Bringing both of these considerations together, we are motivated to investigate how scholarly publishers respond to common HTTP requests that resemble typical behavior of machines such as web crawlers. We therefore send such HTTP requests against thousands of DOIs, follow the HTTP redirects, and record data the publishing platforms respond with. To put responses to machine requests in context, we compare them to responses we received from requests that more closely resemble human browsing behavior.

In this paper we report on the results of this comparative study. Our findings provide insight into publishers' behavior on the web and inform crawling engineers and archivists motivated to capture the scholarly web to use appropriate tools for the task at hand. With the insight that popular web servers do not necessarily adhere to web standards or best practices [10], we have no reason to assume that scholarly publishers are any different. To the contrary, various reports document the sometimes complex relationship between publishers and web crawlers [11], [12]. We therefore believe our work is a worthwhile contribution to the crawling and web archiving as well as to the digital preservation community at large.

We aim to address the following research questions:

RQ1: Do scholarly publishers send the same response to different kinds of HTTP requests against the same DOI? If not, what are the noticeable differences?

RQ2: What characteristics does an HTTP request issued by a machine have to have in order to obtain the same result as a human?

RQ3: Does the DOI resolution follow the same paths for different HTTP requests?

II. Related work

A study of the support of various HTTP request methods by web servers serving popular web pages was conducted by Alam et al. [10]. The authors issue OPTIONS requests to web servers and analyze the "Allow" response header used by servers to indicate which HTTP methods are supported. The study finds that a large percentage of servers either erroneously report supported HTTP methods or do not report supported methods at all. While this study is related in concept, both its scope and methodology are significantly different from our here presented work. The focus of our work is on DOI redirects from the scholarly domain and not just web servers serving popular pages. Unlike Alam et al. we are actually sending a variety of HTTP requests against resources and analyze the responses where they only sent OPTIONS requests and analyzed responses for claims of supported requests.

DOIs are the de facto standard for identifying scholarly resources on the web and therefore a common starting point for crawlers of the scholarly web. We have shown previously that authors, when referencing a scholarly resource, use the URL of the landing page rather than the DOI of the resource [13]. These findings are relevant, for example, for web crawling engineers that need to avoid duplicate crawled resources.

[1] https://www.doi.org/

[2] https://www.cnri.reston.va.us/

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

Similarly, the motivation behind the recent study by Thompson and Jian [14] based on two Common Crawl samples of the web was to quantify the use of HTTP DOIs versus URLs of landing pages. They found more than 5 million actionable HTTP DOIs in the 2014 dataset and roughly 10% of them as their corresponding landing page URL in the 2017 dataset

Various efforts have proposed methods to make web servers that serve (scholarly) content more friendly to machines. There is consensus in the scholarly communication community that providing accurate and machinereadable metadata is a large step in this direction [15], [16]. Aligned with this trend, sitemap-based frameworks have recently been standardized to help machines synchronize metadata and content between scholarly platforms and repositories [17].

III. Experiment setup

A. Data Gathering

Obtaining a representative sample of the scholarly web is not a trivial endeavor. Aside from the concern that the sample should be large enough, it should also reflect the publishing industry landscape since, as for example outlined by Johnson et al. [18], the Science, Technology, and Medicine (STM) market is dominated by a few large publishers.

The Internet Archive (IA)[1] conducted a crawl of the scholarly domain in June of 2018 that lasted for a month and resulted in more than 93 million dereferenced DOIs. The IA crawler followed all redirects, starting from the HTTP DOI to the URL of the DOI-identified resource and recorded relevant data along the way. We refer to the result of dereferencing a DOI as a chain of redirects consisting of one or more links, each with their own URL.

We obtained a copy of the recorded WARC files ([1g]) from this crawl and extracted the entire redirect chain for all 93, 606, 736 DOIs. To confirm that this crawl captures a representative bit of the scholarly landscape, we were motivated to investigate the distribution of publishers in this dataset. We approached this by extracting the URLs of the final link in the redirect chains and examined their hosts. For example, dereferencing the HTTP DOI shown in Section I leads, after following a number of links in the redirect chain to the final URL of the resource at https://link.springer.com/article/10. 1007/s00799-007-0012-y. The host we extracted from this URL is springer.com.

Figure 1 shows the distribution of all hosts extractedfrom the IA crawl dataset. The x-axis lists all hosts and the y-axis (log scale) shows their corresponding frequency. We expected to see a pattern as displayed in Figure 1, given the market dominance of a few publishers and a long tail of small publishers with less representation in the overall landscape. Table 1 lists the top 10 hosts by frequency extracted from the dataset6. We can observe a good level of overlap between top publishers shown by Johnson et al. [18] (cf. Table 1, p. 41) and hosts shown in Table 1. These observations lead us to believe that we have a dataset that is representative of the broader scholarly publishing landscape.

In order to scale down the dataset to a manageable size, we randomly picked 100 DOIs from each of the top 100 hosts, resulting in a dataset of 10, 000 DOIs[2].

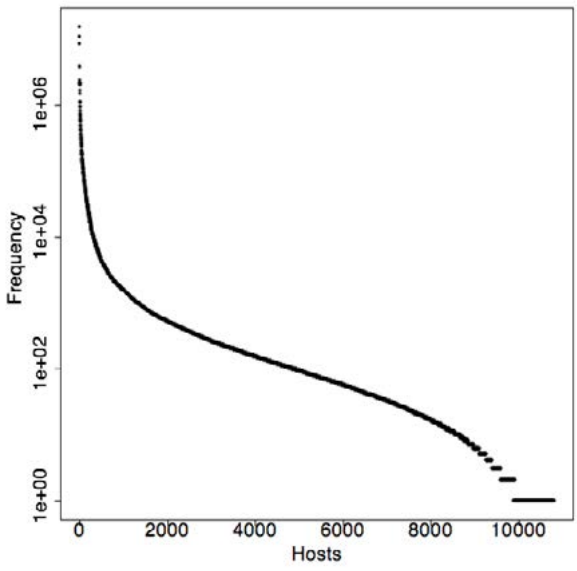

Figure 1: Dataset domain frequency

B. HTTP Requests

[1] https://archive.org//

[2] The dataset is available at: https://doi.org/10.6084/m9.figshare.7853462.v1

HTTP transactions entail request headers sent by the client (such as a web browser) and response headers sent by the web server and received by the requesting client. For a detailed description of defined request and response headers we refer to RFC 7231 [20]. RFC 7231 also specifies all HTTP request methods including the two very frequently used methods GET and HEAD. For detailed information about these methods we again refer to the RFC but note from its text that: "The GET method requests transfer of a current selected representation for the target resource." and "The HEAD method is identical to GET except that the server MUST NOT send a message body in the response…". With respect to the response headers, RFC 7231 states: "The server SHOULD send the same header fields in response to a HEAD request as it would have sent if the request had been a GET…".

Domain	Frequency
elsevier.com	15, 631, 553
springer.com	11, 011, 605
wiley.com	8, 533, 984
ieee.org	3, 941, 252
tandfonline.com	3, 780, 553
plos.org	2, 386, 247
oup.com	2, 199, 106
jst.go.jp	2, 162, 502
sagepub.com	2, 126, 063
jstor.org	2, 060, 760

Table 1: Top 10 domains of final URLs of dereferenced DOIs in our dataset

cURL[1] is a popular tool to send HTTP requests and receive HTTP responses via the command line. Listing 1 shows cURL sending an HTTP HEAD request against a HTTP DOI. The option -I causes the HEAD request method and the added L forces cURL to automatically follow all HTTP redirects. The Listing also shows the received response headers for both links in the redirect chain. The first link has the response code 302 (Found, see [20]) and the second link shows the 200 (OK) response code, which means this link represents the end of this redirect chain.

```
curl -IL http://doi.org/10.1016/j.wocn.2010.05.003

HTTP/1.1 302
Date: Sat, 16 Mar 2019 23:20:13 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 20
Location:
https://linkinghub.elsevier.com/retrieve/pii/S00...

HTTP/1.1 200
Date: Sat, 16 Mar 2019 23:20:14 GMT
Content-Type: text/html;charset=UTF-8
Content-Language: en-US
```

Listing 1: HTTP HEAD request against a DOI

```
curl -iL http://doi.org/10.1016/j.wocn.2010.05.003

HTTP/1.1 302
Date: Sat, 16 Mar 2019 23:20:13 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 20
Location:
https://linkinghub.elsevier.com/retrieve/pii/S00...

HTTP/1.1 200
Date: Sat, 16 Mar 2019 23:20:14 GMT
Content-Type: text/html;charset=UTF-8
Content-Language: en-US

<!DOCTYPE HTML PUBLIC ...>
<html>
<head>
....
</head>
<body>
....
</body>
</html>
```

Listing 2: HTTP GET request against a DOI

Listing 2 shows cURL sending an HTTP GET request against the same DOI and we can see the web server responding with the same response headers but now the last link in the redirect chain (response code 200) also includes the response body. Due to space constraints, we have removed most of the content and only show the basic HTML elements of the response body in Listing 2.

C. *Dereferencing DOIs*

For our experiment, we deployed four different methods to dereference each of the DOIs in our dataset. All four methods were run automatically by a machine since manually dereferencing 10, 000 DOIs and recording data for each link in the redirect chain is not feasible. However, since it is our intention to investigate how scholarly publishers respond to a variety of requests, we implemented two methods that resemble machines crawling the

[1] https://curl.haxx.se/

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

web and two that resemble humans browsing the web. Our methods are:

1. HEAD: Use cURL to send an HTTP HEAD request against the DOI. This lightweight method resembles machine behavior on the web as humans usually do not send HEAD requests.

2. GET: Use cURL to send an HTTP GET request against the DOI. This method also resembles machine behavior as these GET requests do not include typical parameters set by common web browsers.

3. GET+: Use cURL to send an HTTP GET request against the DOI along with the typical browser parameters:
   - user agent,
   - specified connection timeout,
   - specified maximum number of HTTP redirects,
   - cookies accepted and stored, and
   - tolerance of insecure connections.

   This method, while also based on cURL, resembles a human browsing the web with a common web browser due to the setting of these typical parameters.

4. Chrome: Use the Chrome web browser controlled by the Selenium WebDriver[1] to send an HTTP GET request against the DOI. This method is virtually the same as a human browsing the web with Chrome. This method is typically used for web functionality testing [21]–[23] and is therefore commonly considered a proper surrogate for humans browsing.

Each of our four methods automatically follows all HTTP redirects and records relevant data for each link in the redirect chain. The recorded data per link includes the URL, the HTTP response code, content length, content type, etag, last modified datetime, and a link counter to assess the total length of the redirect chain. Each redirect chain ends either successfully at the final location of the resource (indicated by HTTP code 200), at an error (indicated by HTTP response codes at the 400or 500-level), or when an exit condition of the corresponding method is triggered. Examples for an exit condition are a timeout (the response took too long) and the maximum number of redirects (links in the chain) has been reached. For our methods HEAD and GET these two values are the defaults of the utilized cURL version 7.53.1 (300 seconds and 20 redirects) and both values are specifically defined for our GET+ method as 30 seconds and 20 redirects. For our Chrome method we use the default settings of 300 seconds for the timeout and a maximum of 20 redirects. The GET+ and the Chrome methods further have the user agent:

"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36"

specified for all their requests. This user agent mirrors the settings of a desktop Chrome browser to further resemble a human browsing the web. These parameters are based on inspiration from conversations with representatives from the LOCKSS platform. They are therefore based on real-world use cases and hence not subject to an individual evaluation in this work.

It is worth mentioning that we ran our experiments on a machine operated by Amazon Web Services, which means we expect the machine to not have access to paywalled content identified by a DOI. This implies that, just like for the example shown in Section I, for the most part our redirect chains, if successful, ends at a publisher's landing page for the DOI-identified resource. We do not obtain the actual resource such as the PDF version of the paper, for example. The IA crawl, on the other hand, was conducted on IA machines that may have access to some paywalled resources.

IV. Experiment results

Our four methods dereferencing each of the 10, 000 DOIs results in 40, 000 redirect chains and recorded data along the way. For comparison we also include the data of redirect chains recorded by the IA during their crawl of the DOIs in our analysis. We therefore have a total of 50, 000 redirect chains to evaluate.

[1] https://docs.seleniumhq.org/projects/webdriver/

A. HTTP Response Codes Across Methods

Our first investigation was related to our RQ1 and the HTTP response code of the last link in all redirect chains. In an ideal world, all redirect chains would end with a link that indicates "success" and returns the HTTP response code 200, regardless of the request method used. However, from experience navigating the web and educated by previous related work [10], we anticipated to observe a variety of different responses, depending on our four methods.

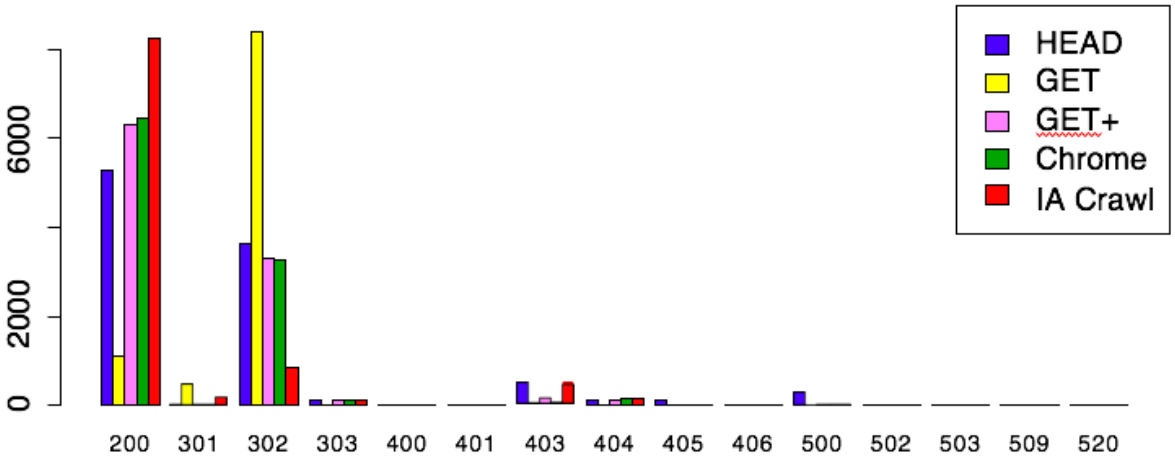

Figure 2: HTTP response codes of the last link in all redirect chains per request method

Figure 2 shows all observed response codes for last links and their frequencies distinguished by requesting method. Each set of five bars is assigned to one individual response code and each bar within a set represents a request method. Within a set, from left to right, the blue bar represents the HEAD method, the yellow bar GET, the pink bar GET+, the green bar Chrome, and the red bar the IA crawl.

We notice a spectrum of 15 different response codes from dereferencing our 10, 000 DOIs across five different methods. The distribution of our observed 50, 000 response codes is almost binary with 27, 418 being 200s and 19, 459 being 302s. Our two methods that resemble a human browser, GET+ (pink bars) and Chrome (green bars) requests result in more than 63% and 64% 200 response codes, respectively. These numbers are disappointing as we would expect more than two out of three HTTP DOIs to resolve to a successful state. The HEAD request method results in even fewer successful responses (53%). The IA crawl scores much better with 83% successful responses. We can only speculate as to the reasons why, especially since their crawls are done with the Heritrix web crawler[1] and this software is more closely aligned with our GET+ than our Chrome method, which returns the most successful results of any of our methods. It is possible though that the crawl parameters were more "forgiving" than ours, for example allowing for a longer timeout.

Our second observation from Figure 2 is that our GET request method results in a very poor success ratio (11%), rendering this method effectively useless for dereferencing DOIs. The majority of DOIs (84%) result in a 302 response code as indicated by the yellow bar in Figur 2. A redirect HTTP response code for the final link in a redirect chain intuitively does not make sense. However, after close inspection of the scenarios, we noticed that this response code is

[1] https://webarchive.jira.com/wiki/spaces/Heritrix/overview

indeed from the last link as the request most often times out. This means the web server simply takes too long to respond to such requests and our method cancels the request at some point. Since this GET method very closely resembles requests that would typically be made by machines, the suspicion arises that this web server behavior is designed to discourage crawling of scholarly publishers' resources. All other response codes do not play a significant role as they are returned in less than 5% of requests.

Figure 2 provides first strong indicators to answer RQ1: the scholarly web indeed responds differently to machines and humans.

B. HTTP Response Codes by DOI

Figure 3 offers a different perspective on the investigation into response codes of final links. The figure does not distinguish between individual response codes anymore but clusters them into four groups: 200-, 300-, 400-, and 500-level represented by the colors green, gray, red, and blue, respectively. Each horizontal line in Figure 3 represents one DOI in our dataset and each of them consists of five horizontal segments. Each segment represents one request method and its coloring indicates the corresponding response code. The image confirms that very few DOIs return with the same response code for all four of our methods. For example, only 880 DOIs return a 200 response code across all four request methods. If we take the IA crawl into consideration as well, the numbers drop even further, in our example to 777

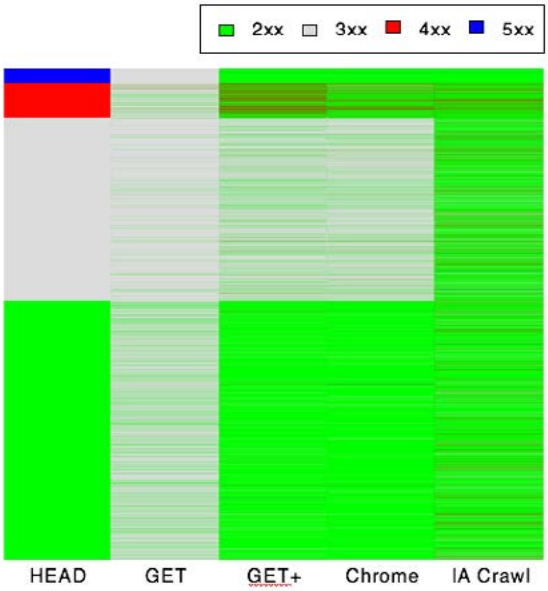

Figure 3: HTTP response codes of the last link in all redirect chains per request method by DOI

DOIs. It is interesting to note that, from visual inspection, the majority of 400 and 500-level responses for HEAD requests (690 and 286, respectively) indeed turn into 200 responses for GET+ and Chrome requests.

The impressions of Figure 3 provide further indicators that machine-based and human-based requests indeed result in different responses. They further hint at similarities between the responses for our GET+ and the Chrome method, which is relevant for crawl engineers and also part of answering our RQ2.

C. HEAD vs GET Requests

With our observation of the significant differences between our two machine-resembling request methods, we were motivated to investigate this matter further. In particular, we were curious to see how publishers respond to the lightweight HEAD requests compared to more complex GET requests. Figure 4 shows all DOIs that resulted in a 200 response code (indicated in green) for the HEAD method. The leftmost bar (HEAD requests) therefore is green in its entirety. The bar mirrors the 5, 275 DOIs (53% of the total) previously shown in Figure 2 (blue bar in the 200 category). The second, third, fourth, and fifth bar in Figure 4 represent the corresponding response codes of these DOIs for the respective request methods. We can observe that the vast majority of DOIs that result in a 200 for HEAD requests also result in a 200 for GET+ (93%), Chrome (96%), and the IA crawl (85%). This finding is not counterintuitive and it is encouraging in way that it would be a huge detriment to web crawling engineers if this picture was reversed, meaning we could not rely on response codes from HEAD requests being (mostly) the same for more complex GET requests. It is telling, however, that the simple GET request method does not echo the HEAD request but results in 83% 300level response codes instead.

The fraction of non-200 responses for the GET+, the Chrome, and the IA crawl are curious. As mentioned earlier, RFC 7231 states that web servers should respond with the same data for HEAD and GET requests but the shown differences indicate that the publishers' web servers do otherwise. The 5% of 400-level responses for the IA crawl (rightmost bar of Figure 4) might be explained by the different

time at which the crawl was conducted (June 2018) compared to our experiments (February/March 2019).

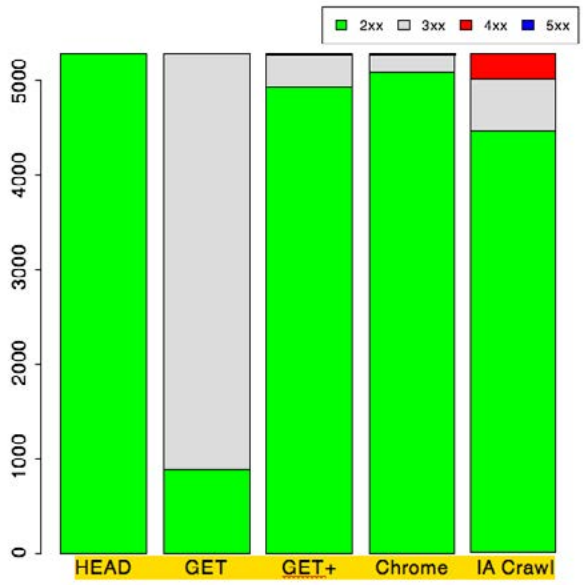

Figure 4: DOIs returning a 200 HTTP response code for HEAD requests and their corresponding response codes for other request methods

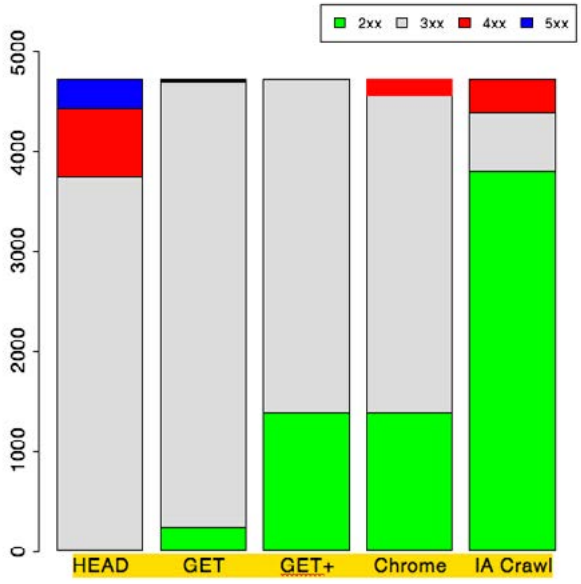

Figure 5: DOIs not returning a 200 HTTP response code for HEAD requests and their corresponding response codes for other request methods

Figure 5, in contrast, shows all DOIs (4, 725 in total) that resulted in a response code other than 200 for the HEAD request method. Consequently, the leftmost bar does not show any green portion at all. We find that 30% and 29% of these DOIs return a 200 code for the GET+ and Chrome method, respectively, and even 80% for the IA crawl. This picture does represent the above mentioned scenario where a developer can not trust the response to a HEAD request since publishers' web platforms seem to respond differently to other request methods.

Figures 4 and 5 clearly show different responses received when dereferencing the same resource with distinct request methods. They also confirm earlier findings related to answering RQ2 that the GET+ method seems to receive similar responses compared to the humanresembling Chrome method.

D. Redirects

Our next exploration was into the redirect chains and the number of links they consist of. The goal was to gain insights into whether the "path to the resource" as directed by the publisher is the same for machines if they even get that far and humans. As a first step we analyzed the total number of redirects for all chains per request method and show the results in Figure 6. We observe that the majority of chains for the HEAD, GET, GET+, and Chrome request methods are of length three or less. Given that the latter two methods result in more than 60% 200 response codes, this is relevant information for crawling engineers. The HEAD method has a noticeable representation with chains of length four (8%) and five (11%) where GET+ or Chrome methods rarely result in such long chains (around 3%). The GET method that mostly results in 300-level responses seems to fail quickly with more than 90% of chains being of length one or two. Note, however, that it may actually take a long time for a GET request to fail if it in fact waits for the timeout to expire. We can only speculate why the ratio of chains with length one is rather small for the IA crawl compared to our methods. Possible explanations are that the user agent used by the IA crawler makes a difference and that the partial access to paywalled content causes a different response and hence a different chain length. More analysis and further experiments run from different network environments are needed to more thoroughly assess this theory though. Figure 6

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

also shows 186 DOIs with a chain length of 21 links. 87 of them were returned from the HEAD request, two each from GET+ and Chrome, and 95 from the IA crawl. All of those DOIs are cases where the web server responds with one 302 code after another and virtually never stops. These scenarios are known as crawler traps and considered a serious detriment to crawler engineering as they can be diZcult to avoid. In our case, the maximum number of redirects was reached and hence the transaction was terminated by the client.

Figure 7 follows the same concept as Figure 6 but only shows the frequencies of chain lengths where the final link returned a 200 response. This data provides insight into how long (in terms of links, not seconds) it is worth waiting for the desired response and how many redirects to expect. We note that the majority of chains for the HEAD, GET+, and Chrome request methods are of length two, three, or four and, in addition, the HEAD method has a strong showing with chains of length four (8%) and five (10%). We also see a similar pattern with the IA crawl and a higher frequency of longer chains. It is interesting to note, however, that no chain in Figure 7 is recorded at length one. At the other end of the scale, there are indeed 15 chains of length 14 that all eventually result in a 200 response code for the HEAD request method.

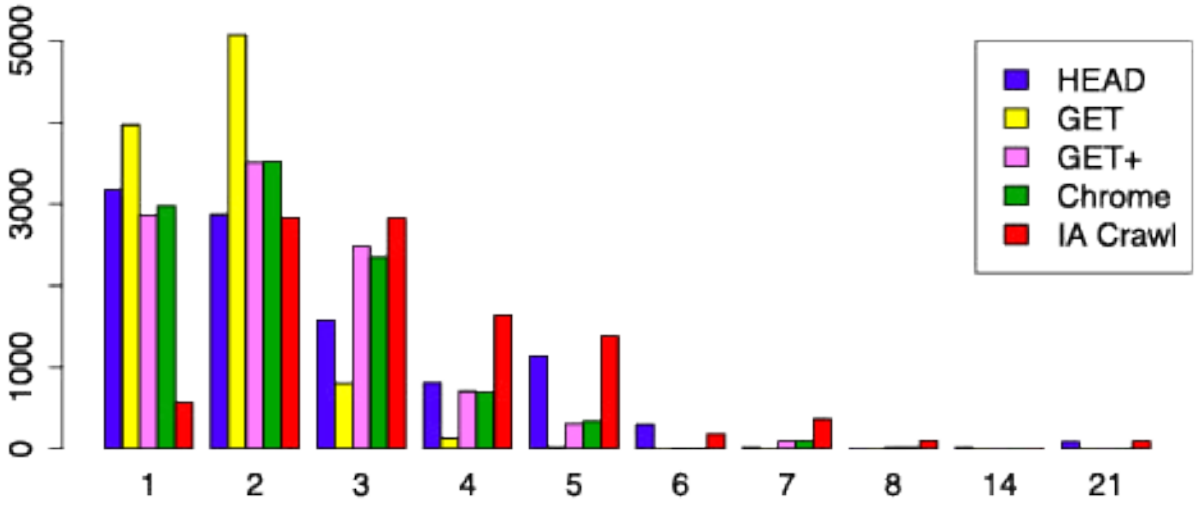

Figure 6: Frequency of number of redirects overall per request method

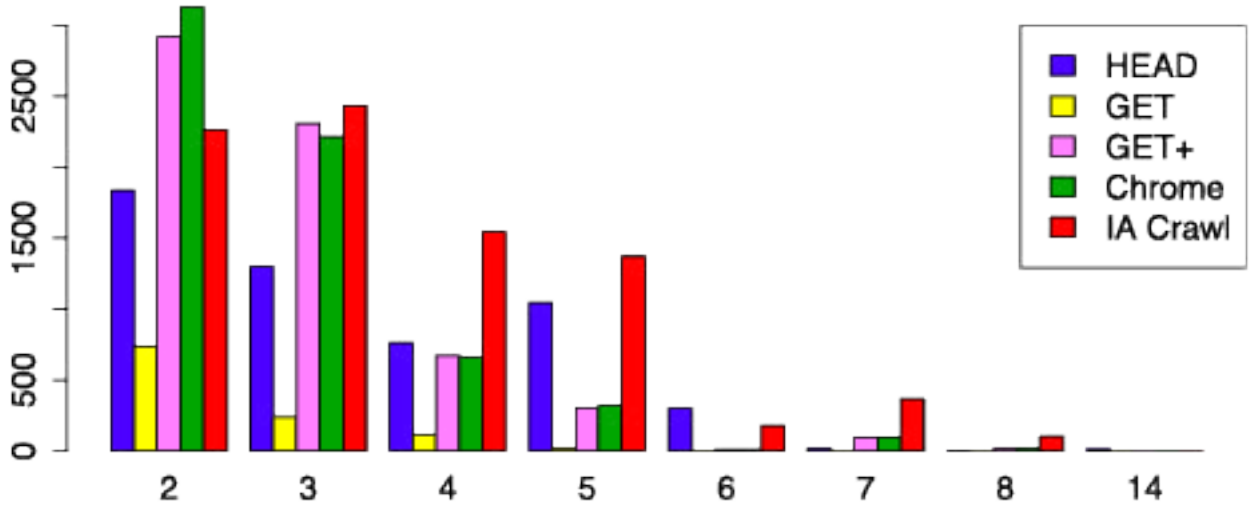

Figure 7: Frequency of number of redirects that lead to the final link with a 200 response code per request method

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

Figures 6 and 7 show that responses not only differ in terms of the provided response code but also in terms of length of the redirect chain, depending on the request method. This finding confidently answers our RQ3, DOI resolutions do not follow the same path for different HTTP requests, at least not as analyzed by path length.

V. Discussion and future work

This experimentation is in its early stages and we see potential for improvements and aspects for future work. As alluded to earlier, we ran our experiments outside the institutional network. We are very interested in seeing how our results compare to those obtained when running the experiment from within an organization that has access to paywalled content. We hypothesize that differences in responses can still be observed between machine and human requests. Further, a logical extension to our series of experiments is to utilize existing infrastructure, for example, the CrossRef APIs to reliably identify individual publishers and make better assessments of their specific behavior on the web. Our dataset consists of 10, 000 DOIs obtained from a crawl by the IA. Our tests have shown that open science platforms that issue DOIs such as Figshare or DataCite are underrepresented in our sample. We are interested in repeating our analyses for these publishers as well, hoping that they might be friendlier to machines. We have utilized the established understanding that a Chrome browser that is controlled by the Selenium WebDriver is indeed virtually the same as a human browsing the web. We are motivated, however, to provide empirical evidence that this is in fact true. We are planning to pursue several approaches such as comparing screenshots taken by a human and by the Chrome method and comparing textual representations (or DOM elements) of the final link in the redirect chain. Such an extension of the experiment may also call for the inclusion of other crawling frameworks, for example, systems based on headless browsers.

VI. Conclusions

In this paper we present preliminary results of a comparative study of responses by scholarly publishers to common HTTP requests that resemble both machine and human behavior on the web. We were motivated to find confirmation that there indeed are differences. The scholarly web, or at least part of it, seems to analyze characteristics of incoming HTTP requests such as the request method and the user agent and responds accordingly. For example, we see 84% of simple GET requests resulting in the 302 response code that is not helpful to crawling and archiving endeavors. 64% of requests by our most human-like request method result in desired 200 responses. These numbers are somewhat sobering we would expect a higher percentage of successful responses but they do serve developers in managing their expectations, depending on the tool and request method used. In addition, they help to address our question raised earlier: "Who is asking?" as it nowcan clearly be answered with: "It depends!".

Acknowledgments

We would like to thank Bryan Newbold and Jefferson Bailey at the Internet Archive for their contributions to this work. We are particularly thankful for the provided dataset and input about their crawling approach. We are also grateful to Nicholas Taylor from Stanford University Libraries for his input regarding approaches implemented by the LOCKSS framework. Lastly, we appreciate Herbert Van de Sompel's contributions to this work in its early conceptual stages.

References

[1] Stanford University Libraries, Collection development, http://library.stanford.edu/projects/webarchiving/collection development.

[2] Purdue University, Web Archive Collecting Policy, https://www.lib.purdue.edu/sites/default/files/spcol/purduearchiveswebarchiving-policy.pdf.

[3] National Library of Finland, Legal Deposit OZce, https://www. kansalliskirjasto.fi/en/legal-deposit-office.

[4] Library and Archives Canada, Legal Deposit, https://www.baclac.gc.ca/eng/services/legaldeposit/Pages/legal-deposit.aspx.

[5] J. Cho and H. Garcia-Molina, "The Evolution of the Web and Implications for an Incremental Crawler," in Proceedings of VLDB '00, 2000, pp. 200–20g.

[6] J. Cho and H. Garcia-Molina, "Estimating frequency of change," ACM Transactions on Internet Technology, vol. 3, pp. 256–2g0, 3 2003, ISSN: 1533-53gg.

[7] Z. Bar-Yossef, A. Z. Broder, R. Kumar, and A. Tomkins, "Sic Transit Gloria Telae: Towards an Understanding of the Web's Decay," in Proceedings of WWW '04, 2004, pp. 328–337.

[8] M. Klein, H. Van de Sompel, R. Sanderson, H. Shankar, L. Balakireva, K. Zhou, and R. Tobin, "Scholarly Context Not Found: One in Five Articles Suffers from Reference Rot," PLoS ONE, vol. g, no. 12, 2014. DOI: 10.1371/journal.pone.0115253.

[9] S. M. Jones, H. Van de Sompel, H. Shankar, M. Klein, R. Tobin, and C. Grover, "Scholarly Context Adrift: Three out of Four URI References Lead to Changed Content," PLoS ONE, vol. 11, no. 12, 2016. DOI: 10.1371/journal.pone.0167475.

[10] S. Alam, C. L. Cartledge, and M. L. Nelson, "Support for various HTTP methods on the web," CoRR, vol. abs/1405.2330, 2014. arXiv: 1405.2330. [Online]. Available: http://arxiv.org/abs/1405.2330.

[11] C. Hayes, Wiley using fake DOIs to trap web crawlers... and researchers, https://blogs.wayne. edu/scholar-scoop/2016/06/02/wileyusingfake dois to trap web crawlers and researchers/, 2016.

[12] L. A. Davidson and K. Douglas, "Digital object identifiers: Promise and problems for scholarly publishing," Journal of Electronic Publishing, vol. 4, no. 2, 1gg8.

[13] H. Van de Sompel, M. Klein, and S. M. Jones, "Persistent uris must be used to be persistent," in Proceedings of WWW '16, 2016, pp. 11g–120. DOI: 10.1145/ 2872518. 2889352. [Online]. Available: https://doiorg/10.1145/2872518.2889352.

[14] H. S. Thompson and J. Tong, "Can common crawl reliably track persistent identifier (PID) use over time?" CoRR, vol. abs/1802.01424, 2018. [Online]. Available: http://arxiv.org/abs/1802.01424.

[15] M. L. Nelson, J. A. Smith, and I. G. del Campo, "EZcient, automatic web resource harvesting," in Proceedings of the 8th Annual ACM International Workshop on Web Information and Data Management, ser. WIDM '06, 2006, pp. 43–50. DOI: 10.1145/1183550.1183560.

[16] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar, "Crawler-friendly web servers," SIGMETRICS Perform. Eval. Rev., vol. 28, no. 2, pp. g–14, 2000. DOI: 10.1145/362883.362894.

[17] M. Klein, H. Van de Sompel, and S. Warner, ResourceSync Framework Speciication (ANSI/NISO Z39.99-2017), http://www.openarchives.org/rs/1.1/resourcesync,2017.

[18] R. Johnson, A. Watkinson, and M. Mabe, The STM Report An overview of scientiic and scholarly publishing. International Association of Scientific, Technical and Medical Publishers, 2018. [Online]. Available: https://www.stm-assoc.org/2018_10_04_ STM_Report_2018.pdf.

[19] nternational Internet Preservation Consortium (IIPC), WARC Speciication, https://iipc.github. io/warc-specifications/.

[20] R. T. Fielding and J. Reschke, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, https://tools.ietf.org/html/rfc7231, 2014.

[21] A. Bruns, A. Kornstadt, and D. Wichmann, "Web application tests with selenium," IEEE Software, vol. 26, no. 5, pp. 88–g1, 200g, ISSN: 0740-745g. DOI: 10.1109/MS.2009.144.

[22] A. Holmes and M. Kellogg, "Automating functional tests using selenium," in AGILE 2006 (AGILE'06), 2006, 6 pp.–275. DOI: 10.1109/AGILE.2006.19.

[23] C. T. Brown, G. Gheorghiu, and J. Huggins, An introduction to testing web applications with twill and selenium. O'Reilly Media, Inc., 2007.