Sound Methodology for Downloading Webpages

Soumya Indela
University of Maryland
College Park, USA
sindela@umd.edu

Dave Levin
University of Maryland
College Park, USA
dml@cs.umd.edu

***Abstract*—Headlessly downloading webpages is a common and useful mechanism in many measurement projects. Such a basic task would seem to require little consideration. Indeed, most prior work of which we are aware chooses a relatively basic tool (like Selenium or Puppeteer) and assumes that downloading a page once yields all of its content—which may work well for static content, but not for dynamic webpages with third-party content. This paper empirically establishes sound methods for downloading webpages. We scan the Alexa top-10,000 most popular websites (and other, less popular sites) with different combinations of tools and reloading strategies. Surprisingly, we find that even sophisticated tools (like Crawlium and ZBrowse) do not get all resources or links alone, and that downloading a page even dozens of times can miss a significant portion of content. We investigate these differences and find that they are, surprisingly, not strictly due to ephemeral content like ads. We conclude with recommendations for how future measurement efforts should download webpages, and what they should report on.**

***Index Terms*—web crawling, Crawlium, ZBrowse**

I. Introduction

Today's web is highly dynamic, with extensive third-party inclusions [8, 2, 15] such as basic resources (e.g., `fonts.google.com`), user tracking (e.g., `scorecardresearch.com`), or advertising.

One can think of today's websites as complicated *inclusion graphs* [2], wherein the nodes are resources loaded from domain names, and there is a directed edge from resource r_1 to resource r_2 if r_1 caused r_2 to be loaded (e.g., if r_1 were a JavaScript file that generated a GET request for an advertisement r_2).[1] It is important to many research questions to be able to obtain as full a picture of a webpage's inclusion graph as possible. For instance, explorations into third-party inclusions [2, 8, 15], malware [9], and website performance [13] are all sensitive to how complete a view of the graph they can obtain at any point in time.

This paper asks a straightforward question: *What is the best way to download a webpage so as to obtain as much of the page's inclusion graph as possible at any one point in time?* Getting a full snapshot of a webpage's inclusion graph can be very difficult, owing to the fact that: (1) The domains, resources, and edges that make up the website's inclusion graph can vary dynamically, even from back-to-back refreshes. (2) Resources are loaded in myriad ways, such as through static inclusions in HTML, dynamic calls in JavaScript, WebSockets, and so on.

We look at this question from two key perspectives:

What effect does the *choice of tools* have on the obtained inclusion graph? Early efforts used basic tools such as `curl` or `wget`, but neither of these support executing JavaScript, which has been shown to be deployed in more than 93% of the most popular webpages [14]. More recent measurement efforts tend to use headless versions of more full-fledged browsers, and programmatic front-ends such as Puppeteer [16] or Selenium [17] to drive them. It would seem at first glance that, so long as the headers and the backing browser are the same, the inclusion graphs would be the same, too.

To evaluate this hypothesis, we compare two recent, sophisticated tools that arose from the research community: Crawlium [6] and ZBrowse [18]. We find that, surprisingly, both obtain parts of the resource graph that the other does not, even when providing the same headers and controlling for dynamic webpage content.

***How many times* should one refresh a page to obtain a more complete snapshot of a webpage's inclusion graph?** To answer this question, we repeatedly download websites and compare the nodes and edges in the inclusion graph with each refresh. We find that there is a surprisingly high variability in the dynamism of websites, with some (like `wikipedia.org`) being highly static over short periods of time, and others resulting in new content with virtually every refresh. The majority of websites are somewhere in the middle: that is, their inclusion graph can be captured in its entirety by reloading the page several times. We present and analyze an *adaptive* webpage loading strategy that fully obtains the inclusion graph without excessive reloads for any page.

We analyze both of these questions across the Alexa top-10,000 (and 10,000 sites selected randomly from the 10,001-1M Alexa-ranked sites), and using multiple `UserAgent` strings, to rule out potential differences between desktop and mobile version of websites. Collectively, our results lead us to a set of considerations that researchers should have when obtaining and analyzing inclusion graphs. To assist in these future efforts, we have made our code and data publicly available at `https://breakerspace.cs.umd.edu/web-topology`

[1] Arshad et al. [2] originally introduced these as inclusion *trees*, but we refer to them as inclusion *graphs* because we observe that nodes can commonly have more than one incoming edge.

978-3-903176-40-9 ©2021 IFIP

The rest of this paper is organized as follows. In Section II, we review background and related work. In Section III, we evaluate the effect that different tools can have by comparing the resource graphs obtained by two popular tools (Crawlium and ZBrowse), and find that neither of them gets the entire inclusion graph, but that they do complement one another. One tempting explanation for these results is that tools differ simply because websites are highly dynamic; in Section IV, we show this not to be the case, but rather some tools *consistently*. get parts of the inclusion graph that other tools do not. In Section V, we analyze how many times webpages should be loaded to obtain a full view of the inclusion graph, and propose an *adaptive* downloading strategy. Finally, in Section VI, we conclude with a set of recommendations.

II. Background and Related Work

Before webpages had dynamic content, it would suffice to use `curl` or `wget`, but neither of these include a JavaScript engine, and thus miss a large portion of the web's content. However, even as early as 2012, Nikiforakis et al. [14] showed that more than 93% of the most popular websites include JavaScript from external sources. Many research projects seek to measure as many third-party resource inclusions as possible [8, 10, 9, 3, 5, 4, 1, 11, 12], making more sophisticated, headless browsers a necessity.

To address this need, many researchers use Puppeteer [16], a Node.js library that gives programmatic control—and data collection—over a headless Chromium browser. For instance, with Puppeteer, one can perform various user actions (navigate to a page, scroll, follow links, etc.) and trigger on various browser events (when a page loads, when a request is or will be made, etc.). To collect information about the specific resources and links that comprise a webpage, one has to choose which events to trigger on.

In this paper, we consider two tools that arose from the research community, and that take complementary approaches:

- **ZBrowse** [18] uses Node.js's built-in `getResourceTree` method for obtaining the DOM tree after the webpage has been loaded. It also augments this tree by collecting data from two network event triggers: `requestWillBeSent` and `responseReceived`.
- **Crawlium** [6] triggers on the same network events as ZBrowse, plus others to capture data sent and received via web sockets, frame navigation, new execution contexts, the parsing of scripts, and when the console API is called. Rather than using Node.js's built-in resource tree construction, Crawlium builds its own tree from the collection of the various network events.

As their motivation for creating Crawlium, Arshad et al. [2] observe that merely obtaining the DOM tree (which can be obtained by most other tools) can miss out on critical resource inclusions, and thus incorporate the other triggers above. ZBrowse uses Node.js's DOM tree, but overcomes this limitation at least in part by augmenting the tree with inclusions learned from network events [18, 10]. Based on what data they collect, it would seem that Crawlium should capture a strict superset of the resources that ZBrowse does. Yet, surprisingly, our results will show this not to always be the case.

Throughout our study, we will demonstrate several measurement parameters that can have a significant effect on how much of the inclusion graph a tool is able to obtain. In particular, we will show that the `UserAgent` string, number of times the page is loaded, specific Node.js event handlers, and the method by which the inclusion graph is constructed can lead to significant differences. Surprisingly, many studies fail to specify precisely what these parameters are. The papers that introduce Crawlium [2] and ZBrowse [10] specify their tools' network events, but do not investigate multiple page loads.

Other studies have investigated the variation of page content from one page-load to another. Zeber et al. [19] and Englehardt and Narayanan [7]—as part of broader studies—both used OpenWPM, a Selenium-based web privacy measurement tool, to compare resources obtained between a pair of simultaneous or back-to-back crawls. Their results broadly agree, and indicate that the same third-party URLs are loaded 28% of the time and the typical overlap is about 90%. We study a slightly different question: how many page loads would we need in order to *exhaustively* obtain the inclusion graph? Our study also extends upon these prior efforts by comparing multiple tools and presenting an adaptive page-loading technique for obtaining more complete inclusion graphs.

III. What Effect Do Tools Have?

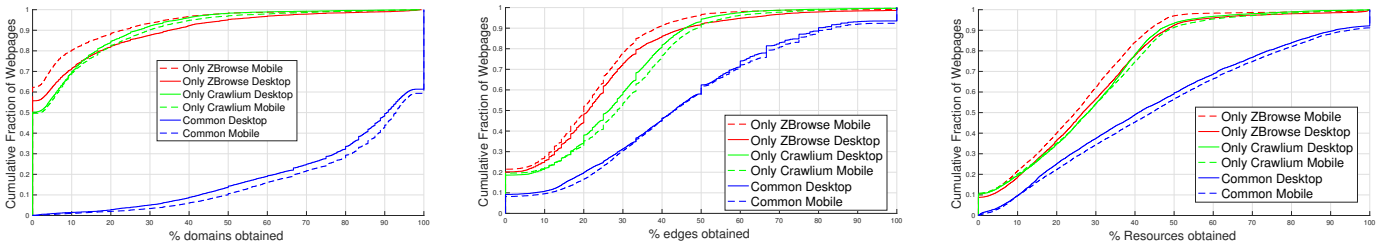
In this section, we ask: does choosing a different automation tool result in a different inclusion graph, even if all other fields (headers and `UserAgent` strings) are kept the same?

Because we are focused on obtaining the *inclusion graph*, this precludes tools that obtain web content but do not record the precise provenance necessary to construct the inclusion graph. For this reason, we do not explore OpenWPM [7], Puppeteer, or Selenium. Certainly, these could all be modified to obtain the inclusion graphs, and so doing would be an interesting (and useful) endeavor. The goal of our work is not to exhaustively compare all tools, but to demonstrate that even seemingly minor differences in two tools can result in significantly different results.

As described in Section II, we focus our study on two seemingly similar tools that natively report on the inclusion graph: Crawlium [2, 3, 5, 4] and ZBrowse [10, 8]. We chose these specific tools because they are, to our knowledge, the state-of-the-art of methods for collecting an inclusion graph, and we decided to consider inclusion graphs as they represent a superset of the information that one could extract from a webpage.

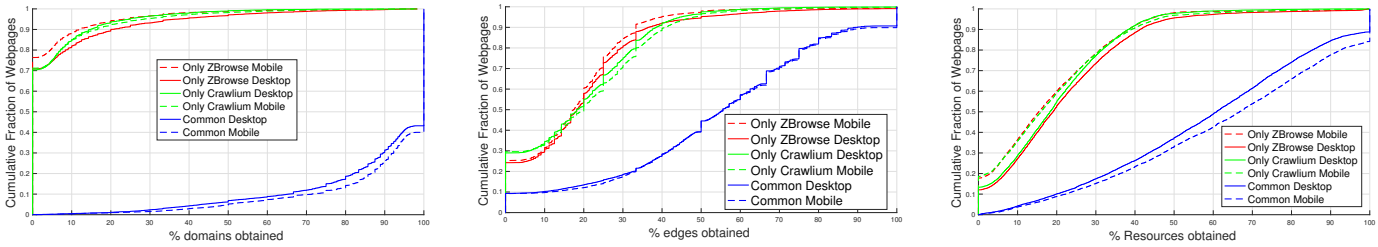
A. Methodology

To compare what the tools obtain, we use them to download each webpage in the Alexa top-10k most popular websites, as well as 10k less popular websites chosen uniformly at



(a) Domains. (b) Edges. (c) Resources.

Fig. 1: Comparison of the domains, edges, and resources obtained by Crawlium and ZBrowse when obtaining the **Alexa top-10k** sites. These plots compare when both *Desktop* and *Mobile* `UserAgent` strings are used.



(a) Domains. (b) Edges. (c) Resources.

Fig. 2: This is the same as Figure 1, but focused instead on less popular sites (a random selection of **10k sites from the Alexa top 10,001–1M** most popular websites). Less popular sites tend to have more in common between the two tools.

random from the websites with Alexa rank between 10,000 and 1M. We use the Alexa ranking instead of the Tranco list, because Tranco list contains the most popular URLs that are visited, be it by human action or not, while Alexa ranking list contains the most popular sites that users explicitly go to. For instance, `googletagmanager.com` is ranked 15 in the Tranco list while it is not ranked anywhere in Alexa top 10,000 websites. Alexa-ranked websites tend to include more third-party resources (including those that are Tranco-ranked), and thus we view Alexa as a sort of "worst case scenario" for crawling an inclusion graph.

These tools require protocols to be explicitly given (`https` versus `http`) and can fail on some pages without the `www` subdomain. To account for this, we try loading each page in the following order of prefixes, whichever succeeds first: `https://`, `https://www.`, `http://`, `http://www.`.

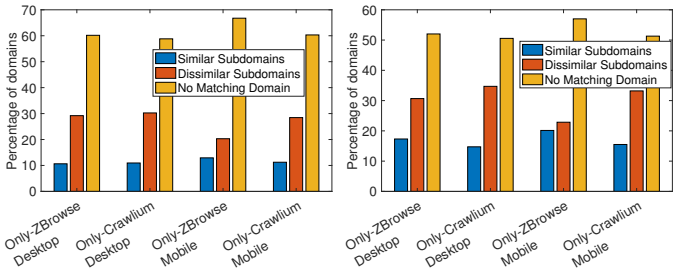
We will demonstrate in Section V that it is important to reload a webpage multiple times to ensure broad coverage of the page's inclusion graph. For the results in this section, we use the Adaptive strategy with $\delta = 3$: that is, for each individual page, and for each tool, we load the webpage repeatedly until there are three consecutive page loads that yield no new domains or edges, up to a maximum of 30 page loads. We disable caching between separate page loads, and we wait until the page has loaded completely (or timed out, at 2 minutes) before continuing.

To evaluate whether the tools' coverage differ for desktop versus mobile browsing, we run two separate trials with different `UserAgent` strings: one[2] (*Desktop*) that purports to be running Chrome in Mac OS X, and another[3] (*Mobile*) that claims to be Chrome on iOS. We performed several tests comparing desktop- and mobile-builds of the browsers, but observed no difference, so in our experiments we used the desktop-build but varied the `UserAgent` string.

After downloading each of the Alexa top-10k sites, we union together the resources (complete URL) obtained, the corresponding fully-qualified domains and the directed edges between them representing the redirections/loading to construct a complete inclusion graph for each individual site. Unfortunately, Crawlium and ZBrowse both sometimes fail to download webpages; to permit a direct comparison in this section, we only compare the 8,221 pages among the Alexa top-10k sites for which both tools were successful in downloading them. We observed two broad reasons for failure: exceeding our two-minute timeout (reducing this failure would have required drastically increasing our time to collect data), and ephemeral errors in reaching the websites (which we verified by manually visiting the websites at the time). We compare the data only for those websites which have successful downloads for both tools and both `UserAgent` strings. We accordingly downsampled from the less-popular webpages to 8,221, as well.

[2]`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36`

[3]`Mozilla/5.0 (iPhone; CPU iPhone OS 13_3 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/69.0.3497.105 Mobile/15E148 Safari/605.1`



(a) Alexa top-10k sites. (b) Less popular 10k Alexa sites.

Fig. 3: How the **domains** found by only one tool compare to the domains found by the other. The tools show no significant difference for less popular sites.

For each of the webpages we could successfully download, we obtained four inclusion graphs, accounting for the two tools and the two `UserAgent` strings. This gave us a total of 65,768 inclusion graphs ($4 \times 8{,}221 = 32{,}884$ for the most popular, and an equal number for the less popular sites). For a particular website, we alternated between Crawlium and ZBrowse and ran both the tools until no new data was obtained. We did this simultaneously for both the mobile and desktop `UserAgent` strings. Thus, the data for a single page load of any particular website for both devices is obtained with a difference of less than our timeout of two minutes (typically much less) to enable thorough comparison.

B. Results

We begin by comparing the percentage of data that is obtained only by Crawlium, only by ZBrowse, and common to both. Figure 1 presents the comparison between the (a) fully qualified domains, (b) inclusion graph edges, and (c) resources obtained by Crawlium and ZBrowse for Alexa top-10k sites. Figure 2 presents the equivalent data for the less-popular sites. We observe that the most and least popular sites show similar trends, but that in general there is more agreement between tools for less popular sites. For the remainder of this section, we focus on the top-10k most popular sites, but our overall observations hold independent of popularity.

We investigate the three data types in turn.

Differences in domains Figure 1(a) shows that, for the median webpage, Crawlium and ZBrowse agree on over 90% of the domains for both desktop and mobile. About 40% of webpages in the Alexa top-10k return the exact same domains to both tools (this corresponds to the jump in the common line at x=100%). However, there is still a significant fraction of webpages where the tools differ considerably. For 20% of webpages, they disagree by 38% of the domains for desktop and 33% for mobile.

When they disagree, Crawlium tends to obtain more domain names than ZBrowse, as expected, but surprisingly, ZBrowse is still able to obtain many domains that Crawlium does not. This is surprising because Crawlium subscribes to more network events than ZBrowse.

We initially hypothesized that the tools' differences may be superficial, and attributable to load balancing, such as one tool obtaining `gtms01.alicdn.com` and the other obtaining `gtms03.alicdn.com`. To better understand the nature of the differences between the two tools, Figure 3 shows *how different* the domains are. To ensure consistency in the differences, the data is compared not for a single page load but for multiple page loads (until three consecutive page loads yield no new data). Each group of numbers in this plot correspond to the percentage of domains that are unique to the specific tool (and `UserAgent`), broken down into three categories:

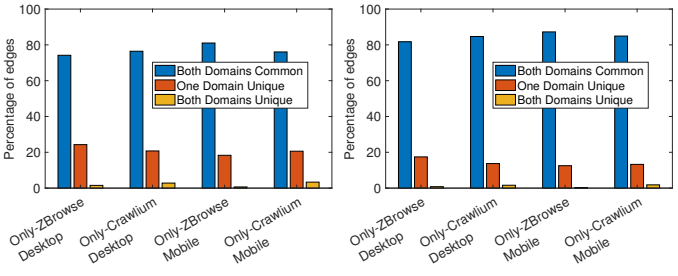
First are the domain names that, although unique, are very similar to a domain name that the other tool obtained. More precisely, the tool finds a domain d for which there is another domain d' found by the other tool that shares the same effective second-level domain (E2LD, e.g., "`example.com`" in "`www.example.com`") *and* the subdomain has a Levenshtein distance of at most 3 (not including the separating period). For instance, this category would include prefix differences, for example `http://www.maxtv.cn` obtained by one tool, while both obtain `https://maxtv.cn` The second category is ones where the E2LD matches but the subdomain differs by a Levenshtein distance of more than 3. Finally, the third category are domains found by the given tool for which there is no matching E2LD found by the other tool.

Figure 3 shows that only a small fraction (9–11%) of differences are attributable to very similar domains. Rather, the bulk of the differences are due to finding *completely different* domain names.

Differences in edges Crawlium and ZBrowse disagree significantly more on edges than they do on domains. As shown in Figure 1(b) , for the median website, the tools agree on only 43% of the edges (55% for less popular sites). Moreover, less than 8% of webpages yield the same exact edges in both tools. For 70% of webpages, ZBrowse regularly finds fewer unique edges in the inclusion graph than Crawlium, especially for mobile webpages. In the long tail, there are about 10% of pages for which ZBrowse finds considerably many more edges than Crawlium on the desktop. Much like with domain names, this indicates that, although Crawlium does generally outperform ZBrowse, there is no clear winner.

To better understand the nature of the differences, we investigated the two domains on either side of each unique edge. For an edge $d_1 \rightarrow d_2$ that is unique to one of the tools, there are three possibilities: either the other tool also had downloaded domains d_1 and d_2, or it had only one of the two, or it had neither.

Figure 4 shows the breakdown of the unique edges for both tools and `UserAgent` strings. Interestingly, more than 70% of the edges that are unique to the tools are due to the edges between domains both tools found in common. This is likely due to differences in how the two tools compute their inclusion graphs. Recall that ZBrowse uses Node.js's built-in method for obtaining the DOM tree, while Crawlium builds it from scratch



(a) Alexa top-10k sites. (b) Less popular 10k Alexa sites.

Fig. 4: When a tool obtains a unique **edge**, how often both tools observe the edge's domains. The tools show only slightly greater agreement for less popular domains.

by listening on all events. ZBrowse is less likely to discover edges with both unique domains when compared to Crawlium, owing in part to the fact that Crawlium obtains more domains that ZBrowse does not.

Once again, we conclude that Crawlium and ZBrowse offer complementary data: ZBrowse is more adept at identifying edges in the domains that the two tools share in common, but Crawlium obtains more domains, which allows it to find more edges, as well.

Differences in resources Finally, we turn to the differences in the resources the two tools obtain. Figure 1(c) shows that the two tools differ considerably in which resources they return, with the median *Desktop* webpage having 41% of the resources in common (45% for the median *Mobile* page, and 60% for less popular pages). Like with edges, slightly less than 10% of webpages had the same exact resources. This is to be expected based on the previous results; many websites load multiple resources from the same domains, so when domains differ, resources will, as well.

C. Recommendations

Based on the above results, we make the following recommendations:

Report what specific events are triggered. Tools and papers that use headless browser APIs like Puppeteer should clearly articulate which events they trigger on.

Compare against other tools. Future tools should compare directly to one another, and report on the differences in the domains, resources, and edges the tools are able to obtain.

To maximize coverage, use complementary tools concurrently. If the goal is to maximize coverage of a website—that is, to obtain as many domains, edges, or resources as possible—then one should consider using two complementary tools concurrently.

In the remainder of this paper, we use both Crawlium and ZBrowse concurrently.

IV. Is Disagreement Caused by Dynamism?

Section III showed that different tools can result in different inclusion graphs. One tempting explanation for this is that the differences arise from the fact that webpages today are highly dynamic, rather than something inherent to the tools themselves. In this section, we show this not to be the case.

Rather, we observe many instances in which one tool *consistently* obtains a given domain, edge, or resource that the other tool rarely or *never* gets. For example `https://www.nytimes.com` utilizes all 30 page loads to obtain data. Crawlium obtains `https://stats.g.doubleclick.net` in all 30 page loads, yet ZBrowse does not obtain it once. Similarly, ZBrowse obtains `https://pixel.adsafeprotected.com` in all 30 page loads and Crawlium obtains it in only 23.

To study this more broadly across all of the domains we measured, we compute, for every domain d loaded from every Alexa site a, how many times d appeared in each of the page loads of a. We compute this separately for both Crawlium and ZBrowse. To rule out any potential domain name differences caused by load balancing, we perform our analysis strictly over E2LDs: this should result in *greater* agreement between the two tools.

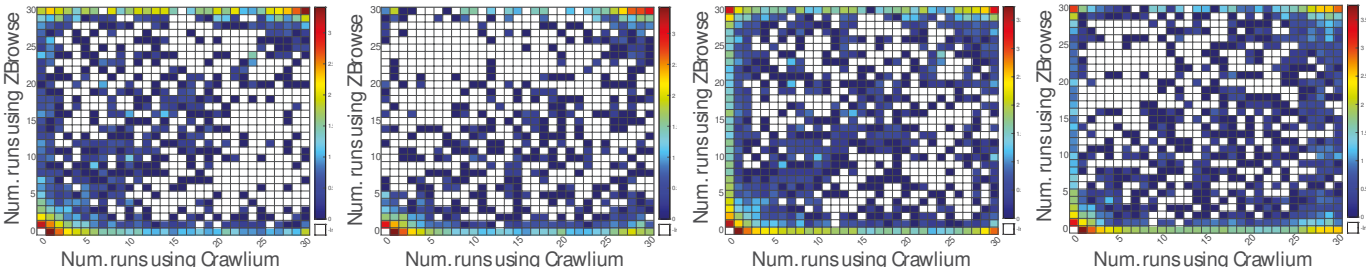
We plot these as heatmaps in Figures 5 and 6, where each value (x, y) is the number of (Alexa, domain) or (Alexa, edge) pairs that were obtained x times by Crawlium and y times by ZBrowse. In these plots, white cells correspond to zero, and all other cells are colored from blue (low) to red (high) on a logarithmic scale. Note that the cells at $(0, 0)$ are always white, because we only compare domains that were loaded at least once by at least one of the tools. If the two tools were in complete agreement, then all of the values would be along the $y = x$ diagonal.

Figure 5 shows the results for the webpages that required all 30 page loads in our adaptive strategy. We note that there are strong concentrations at the top-right corner (when both tools get almost all of the data all of the time) and the bottom left corner (when one tool got a data item once, and the other tool never did). These alone would correctly account for typical webpage dynamism. However, for domains, there are also strong concentrations along bottom row ($y = 0$; when Crawlium gets a domain that ZBrowse never does) and the top row ($y = 30$, when ZBrowse consistently gets a domain that Crawlium does not). Likewise, for edges, we also see a strong concentration along the left column ($x = 0$, when ZBrowse gets an edge that Crawlium never does).

Figure 6 shows the results for all of the webpages, regardless of how many times they needed to reload. Here, we see a stronger concentration along the diagonal $y = x$, but note that the diagonal is effectively the union of the webpages "top right corner" (i.e., when both tools get the data item in *all* of the refreshes).

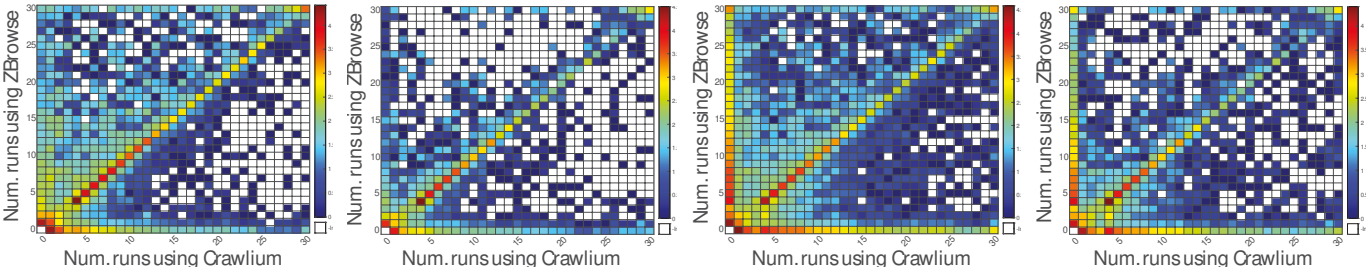
We provide several additional examples:

- `doubleclick.net` is loaded in all 30 page loads by 6 (and 4) Alexa websites using only Crawlium on Desktop (and Mobile) and on 6 other Alexa websites using only ZBrowse on both Mobile and Desktop.
- `googlesyndication.com` is obtained only by Crawlium on Mobile for 6 Alexa websites.



(a) Domains; popular. (b) Domains; less popular. (c) Edges; popular. (d) Edges; less popular.

Fig. 5: The number of runs in which different domains and edges are obtained by Crawlium and ZBrowse, limited to the domains that **required all 30 page loads**. (Desktop only shown; Mobile results are very similar.)



(a) Domains; popular. (b) Domains; less popular. (c) Edges; popular. (d) Edges; less popular.

Fig. 6: The number of runs in which different domains and edges are obtained by Crawlium and ZBrowse, covering all domains **regardless of the number of page loads**. (Desktop only shown; Mobile results are very similar.)

- `spotxchange.com` is loaded in a single page load by 187 (and 455) Alexa websites using Crawlium only and by 514 (and 281) other Alexa websites using only Zbrowse on Desktop (and Mobile).

These trends cannot be explained by mere randomness in webpage content, and instead demonstrate that there are systemic differences between the two tools we have studied.

V. How Many Refreshes?

In this section we ask: how often does a page need to be loaded in order to obtain all of its resources and links at a given point in time? Barring dynamic content such as advertising, it would seem as though the answer should be *once*: downloading a webpage a single time *ought* to obtain virtually all of the content. We show this not to be the case.

A. Methodology

For this part of our study, we download individual websites many more times than in other portions of our study, and as a result we focus here on a smaller set of domains: the Alexa top-1,000 most popular websites. As in Section III, we use the same order of `https`, `http`, and `www`, and we use both the Desktop and Mobile `UserAgent` strings. Also, we download using both Crawlium and ZBrowse, and union their nodes and edges into a single inclusion graph for each download.

We download each webpage 30 times, in back-to-back succession, restarting the tool and clearing the cache and cookies each time. We chose this number because we felt it was likely much higher than necessary (or at least higher than most researchers would be willing to invest), and we found larger values to have marginal returns for every website in our initial test set. The union of all 30 of these inclusion graphs provides our most complete view of the given webpage that we have.

		Domains					
		≥1%	≥50%	≥75%	≥90%	≥95%	≥99%
Webpages	≥1%	1	1	1	1	1	1
	≥50%	1	1	1	1	3	7
	≥75%	1	1	11	22	26	29
	≥90%	1	7	18	26	28	30
	≥95%	1	10	20	26	28	30
	≥99%	1	12	21	27	29	30

TABLE I: Number of page loads necessary to obtain *column%* of **domains** for *row%* of webpages from the Alexa top-1000, using a *Desktop* `UserAgent`.

Our primary analysis in this section consists of computing how many of the consecutive page loads were necessary in order to obtain a given percentage of the domains (fully qualified domain names), edges (directed edges between domains), and resources (complete URL) from the graph of all 30.

B. Results

Of the Alexa top-1,000 webpages we crawled, 982 have domains and resources for both Desktop and Mobile; out of which 942 have edges on Mobile, and 930 have edges on Desktop. We report on the 982 domains for which both tools responded.

		Resources					
		≥1%	≥50%	≥75%	≥90%	≥95%	≥99%
Webpages	≥1%	1	1	1	1	1	1
	≥50%	1	13	21	27	29	30
	≥75%	1	14	22	27	29	30
	≥90%	1	15	23	27	29	30
	≥95%	1	15	23	28	29	30
	≥99%	1	16	24	28	29	30

TABLE II: Number of page loads necessary to obtain *column%* of **resources** for *row%* of webpages from the Alexa top-1000, using a *Desktop* `UserAgent`.

Strategy	**Total # Loads**	**Avg. % Domains**	**Avg. % Edges**	**Avg. % Resources**
1 load (prior work)	982	76.4	73.0	24.7
Adaptive, $\delta = 1$	9781	95.2	93.4	51.5
Adaptive, $\delta = 3$	12,515	97.6	96.5	59.0
Adaptive, $\delta = 5$	14,357	98.1	97.2	63.8
Adaptive, $\delta = 10$	18,958	98.9	98.3	75.5
30 loads (all)	29,460	100	100	100

TABLE III: Requisite page loads and amount of content received for different download strategies, when run against the 982 of the Alexa top-1000 websites that responded with a *Desktop* `UserAgent`.

Table I shows the number of page loads necessary to obtain *column%* of the domain names from *row%* of the webpages in the Alexa top-1,000. For example, 22 page loads would yield at least 90% of the domains from at least 75% of the webpages. This table corresponds to using a Desktop `UserAgent`; we find very similar results when we use a Mobile one instead, in general requiring slightly fewer page loads.

The results for the number of page loads to obtain a given percentage of *edges* are very similar (we elide the table for space). As with domain names, we see that a surprisingly large number (26) is required for obtaining over 90% of the edges from over 90% of the webpages.

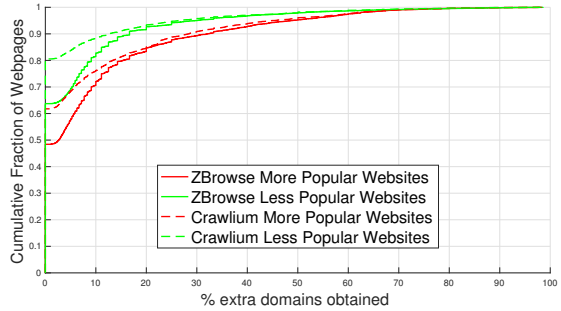
Finally, Table II shows how many page loads are necessary to obtain a given percentage of resources (for Desktop; similar numbers for Mobile). We find that webpages' resources are by far more dynamic than the corresponding fully-qualified domain names from which they are loaded and the edges that connect them.

C. Adaptive Reloading

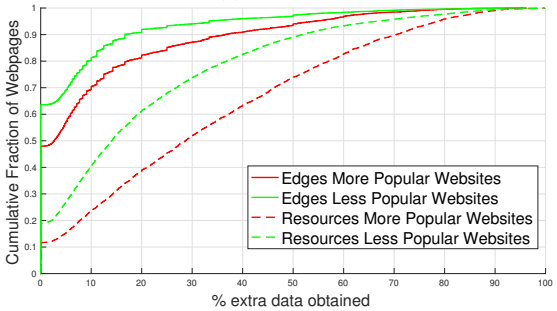
The above results indicate that the number of reloads necessary to obtain a large fraction of webpages' inclusion graphs can vary widely across webpages. Some require just a few reloads to get over 90% of the inclusion graph, while others require dozens of reloads. Picking a single number of reloads risks unnecessary overhead downloading webpages that do not need many, or risks under-sampling the webpages that do.

We next experiment with a simple adaptive reloading heuristic. The idea is to reload a webpage until there have been at least $\delta \geq 1$ consecutive reloads that yield no additional domains or edges beyond what has already been returned from previous loadings of the page. Note that this would require each webpage to be downloaded at least $\delta + 1$ times. For instance, with $\delta = 1$, websites like `wikipedia.org` that return the entire inclusion graph in a single page load would require two page loads.

Table III presents our results comparing a single page load (the standard in prior work) to this adaptive strategy with varying values of δ. This table presents the results for a `UserAgent` string purporting to be a Desktop client. In the case of a Mobile client, the percentages of domains, edges, and resources are nearly the same, but Mobile tends to require roughly 1000 fewer page loads for each of the Adaptive strategies.



(a) Domains.



(b) Edges and Resources.

Fig. 7: Percentage of domains obtained by Crawlium and ZBrowse, and percentage of edges and resources obtained by ZBrowse in subsequent page loads for the Alexa top-10k sites and 10k sites from among Alexa rank 10,001 to 1 million using *Desktop* `UserAgent` string.

These results show that the standard approach of performing a single page load fails to obtain on average 23.6% of the domain names and 27.0% of the edges in webpages' inclusion graphs. Our adaptive strategy is able to obtain significantly more content without having to exhaustively download the full 30 times. For instance, with 42.5% of the maximum number of page loads, the adaptive strategy with $\delta = 3$ misses only 2.4% of the domains and 3.5% of the edges, on average. There are diminishing returns for higher values of δ.

How much data comes after the first page load? Figure 7 shows the percent of new data (domains, edges and resources) obtained after the first page load using adaptive strategy with

$\delta = 3$ for Alexa top-10k sites and randomly selected 10k less popular sites on desktop (the plots for mobile are very similar). We observe that, compared to less popular websites, more popular websites obtain more new domains, edges and resources in subsequent page loads. Also, as shown in Figure 7(a) , ZBrowse obtains more new domains in subsequent page loads than Crawlium. Figure 7(b) shows the percentage of new edges and resources obtained using ZBrowse (the results were nearly identical for Crawlium).

How many page loads were needed in the rest of our study? Figure 8 shows the distribution of the number of page loads when using the adaptive strategy with $\delta = 3$ across websites we used in Sections III and IV. The majority of websites required four total downloads (meaning all of their content was obtainable in one); and there is a long tail. Only a small fraction require the maximum of 30 page loads—these are likely websites with so much dynamic content that they can never obtain a full snapshot of their content at any one point in time. We note also that Mobile versions of websites required slightly fewer page loads than their Desktop versions. Interestingly, less popular websites required fewer page loads across both Desktop and Mobile versions.

Collectively, these results shows that there is a wide variance in webpages' dynamism and complexity, and motivates moving away from one-size-fits-all approaches to measuring them.

Our adaptive strategy incurs significantly higher overhead compared to the standard single page load—even with $\delta = 3$, it requires nearly $12\times$ page loads on average. One possible area for improvement would be to adaptively load not until there are *no* more changes (as we have done), but rather until the changes are below some threshold fraction of all of the content received thus far.

D. Recommendations

Based on the above results, we make the following recommendations:

Load webpages more than once. Almost all webpages today have too much dynamism to get a significantly complete inclusion graph from a single page load.

Avoid one-size-fits-all solutions. There is simply too much variability across webpages in terms of the number of page loads required to obtain a given percentage of their resources. Prefer reload strategies tailored to specific webpages. Barring any prior knowledge of a webpage, one can apply adaptive techniques like the one we have presented.

VI. Conclusion

Downloading accurate and complete inclusion graphs of webpages is an important building block towards understanding myriad phenomena such as advertising [9], malicious inclusions [8, 2, 15], and performance optimizations [13]. Surprisingly, there is little consistency across various studies on how to crawl the web, and even sophisticated tools lack thorough comparisons to one another.

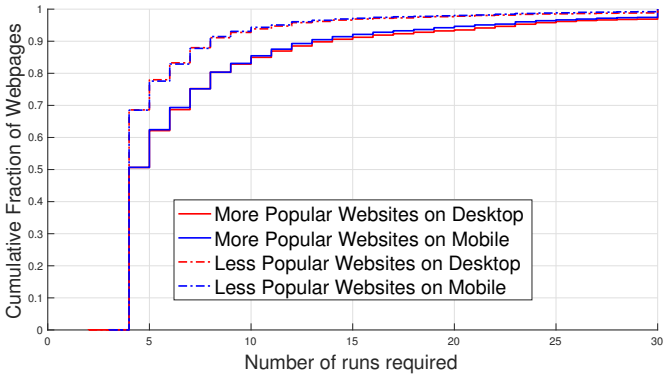


Fig. 8: Number of page loads for adaptive strategy ($\delta = 3$)

In this paper, we have sought to take the first step towards an empirical foundation for crawling webpages. To this end, we compared two state-of-the-art tools, Crawlium [6] and ZBrowse [18], to understand *which* tool to use and *how many* times to reload a page. We found the tools to be complementary, and recommend that both tools' techniques should be used to download webpages. We also found that downloading webpages a single time (as is often the case) misses more than 25% of the domain names from more than 25% of webpages. We recommend an adaptive strategy that trades off overhead (more page loads) for more coverage.

Our results demonstrate several features that have significant impact on the inclusion graph (whether the `UserAgent` string is that of a mobile or desktop device, how many times the page is loaded, which events the tools listen for, and so on). Our hope is that our results will lead to more papers exploring these various parameters and reporting on them so that others may evaluate and reproduce more accurately. To assist in such future efforts, we have made our code and data publicly available at

`https://breakerspace.cs.umd.edu/web-topology`

This work raises no ethical concerns.

Acknowledgments

We thank our shepherd, Poonam Yadav, and the anonymous reviewers for their helpful feedback. This work was supported in part by NSF grant CNS-1901325.

References

[1] Sajjad Arshad, Amin Kharraz, and William Robertson. Identifying extension-based ad injection via fine-grained web content provenance. 2016.

[2] Sajjad Arshad, Amin Kharraz, and William Robertson. Include Me Out: In-Browser Detection of Malicious Third-Party Content Inclusions. 2016.

[3] Muhammad Ahmad Bashir, Sajjad Arshad, Engin Kirda, William Robertson, and Christo Wilson. A Longitudinal Analysis of the ads.txt Standard. 2019.

[4] Muhammad Ahmad Bashir, Sajjad Arshad, William Robertson, and Christo Wilson. Tracing information flows between ad exchanges using retargeted ads. 2016.

[5] Muhammad Ahmad Bashir, Sajjad Arshad, and Christo Wilson. "Recommended For You" A First Look at Content Recommendation Networks. 2016.

[6] Crawlium (DeepCrawling): A Crawling Platform Based on Chrome (Chromium). https://github.com/sajjadium/Crawlium, 2020.

[7] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. 2020.

[8] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading. 2019.

[9] Umar Iqbal, Peter Snyder, Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. ADGRAPH: A Graph-Based Approach to Ad and Tracker Blocking. 2020.

[10] Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J Alex Halderman, and Michael Bailey. Security challenges in an increasingly tangled web. 2017.

[11] Dimitris Mitropoulos, Panos Louridas, Vitalis Salis, and Diomidis Spinellis. Time present and time past: analyzing the evolution of javascript code in the wild. In *IEEE/ACM International Conference on Mining Software Repositories (MSR)*, 2019.

[12] Winai Nadee and Korawit Prutsachainimmit. Towards data extraction of dynamic content from javascript web applications. In *International Conference on Information Networking (ICOIN)*, 2018.

[13] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. 2016.

[14] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. 2012.

[15] Karlis Podins and Arturs Lavrenovs. Security implications of using third-party resources in the world wide web. In *IEEE Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, 2018.

[16] Puppeteer: Headless Chrome Node.js API. https://github.com/puppeteer/puppeteer, 2020.

[17] Selenium. https://www.selenium.dev/.

[18] ZBrowse. https://github.com/zmap/zbrowse, 2017.

[19] David Zeber, Sarah Bird, Camila Oliveira, Walter Rudametkin, Ilana Segall, Fredrik Wollsén, and Martin Lopatka. The representativeness of automated web crawls as a surrogate for human browsing. In *Proceedings of The Web Conference 2020*, WWW '20, page 167178, New York, NY, USA, 2020. Association for Computing Machinery.