Reviving Dead Links on the Web with FABLE

Jingyuan Zhu
University of Michigan

Anish Nyayachavadi
University of Michigan

Jiangchen Zhu
Columbia University

Vaspol Ruamviboonsuk
Microsoft Inc.

Harsha V. Madhyastha
University of Southern California

ABSTRACT

The web is littered with millions of links which previously worked but no longer do. When users encounter any such broken link, they resort to looking up an archived copy of the linked page. But, for a sizeable fraction of these broken links, no archived copies exist. Even if a copy exists, it often poorly approximates the original page, e.g., any functionality on the page which requires the client browser to communicate with the page's backend servers will not work, and even the latest copy will be missing updates made to the page's content after that copy was captured.

To address this situation, we observe that broken links are often merely a result of website reorganizations; the linked page still exists on the same site, albeit at a different URL. Therefore, given a broken link, our system FABLE attempts to find the linked page's new URL by learning and exploiting the pattern in how the old URLs for other pages on the same site have transformed to their new URLs. We show that our approach is significantly more accurate and efficient than prior approaches which rely on stability in page content over time. FABLE increases the fraction of dead links for which the corresponding new URLs can be found by 50%, while reducing the median delay incurred in identifying the new URL for a broken link from over 40 seconds to less than 10 seconds.

CCS CONCEPTS

• **Information systems** → **World Wide Web**; **Digital libraries and archives**.

KEYWORDS

Web Page Rediscovery, Web Archives

ACM Reference Format:
Jingyuan Zhu, Anish Nyayachavadi, Jiangchen Zhu, Vaspol Ruamviboonsuk, and Harsha V. Madhyastha. 2023. Reviving Dead Links on the Web with FABLE. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3618257.3624832

1 INTRODUCTION

A crucial aspect of creating a web page is to include links which enable visitors to discover other related pages. However, much of the effort that page authors put into linking to appropriate pages on other sites goes to waste over time. When users visit a page created many years ago, they often find that some of the links on the page do not work [54, 61, 63]. For any particular link, the site hosting the linked page may no longer exist, that page may have been deleted, or the page's URL might have changed. This robs users of relevant context that the page's authors meant to provide, e.g., millions of external references included in Wikipedia articles no longer work [10, 37].

The oft-used solution to cope with this problem of 'link rot' is to rely on web archives such as the Internet Archive [13] or the Library of Congress [26]. A web archive crawls and saves snapshots of web pages so that users can later look up copies from the past of any page. For example, the Internet Archive augments any broken reference on Wikipedia with a link to an archived copy of the referenced URL [14]. Similarly, the Brave web browser [5] and the Cloudflare CDN [6] direct any user who encounters a broken link to an archived copy of the page at that URL.

Despite this widespread reliance on archived page copies to cope with broken URLs, it is important to recognize that a snapshot of a web page is often a poor substitute for the original page. Modern web pages typically include functionality that requires client browsers to interact with backend servers, e.g., to make a purchase or post a review on a product page; such functionality will not work on an archived snapshot of a page [36]. In addition, since page content is often updated over time, the last archived copy of a page may be stale, e.g., after a news article was last archived, some important corrections might have been posted [35]. Furthermore, since it is impractical to archive the entire web, even the Internet Archive – the largest web archive in operation today – has no copy for many pages [11, 28, 29, 57].

When users encounter a broken link for which no archived copies exist or they find that snapshots of the linked page offer poor fidelity, what recourse do they have? Thankfully, a broken link to a page does not always mean that the page no longer exists on the web. Rather, the link may be broken only because the page's URL has changed as a result of a reorganization of the site hosting it. Such a site would ideally redirect requests for any page's old URL to the corresponding new URL, but many reorganized sites fail to do so; see Table 1 for examples. In such cases, if a user visits the linked page using its new URL (which we refer to as an *alias* of the page's original URL), the user can access up-to-date page content and use all services on the page.

However, today, when a user encounters a broken link, the onus of searching the web and determining if the linked page still exists at an alternate URL falls on the user. To instead automate the process, we present FABLE (Finding Aliases for Broken Links Efficiently). To avail of FABLE's benefits, users can either install its browser add-on

IMC '23, October 24–26, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0382-9/23/10…$15.00
https://doi.org/10.1145/3618257.3624832

Problem with archived copy	Example broken URL with functional alias
Stale content	***Broken URL:*** http://www.hks.harvard.edu/centers/mrcbg/about/fellows/currentsrfellows Set of senior fellows listed in the last archived copy [4] is out-of-date compared to content at page's new URL ***New URL for the same page:*** https://www.hks.harvard.edu/centers/mrcbg/about/senior-fellows
Inaccessible service	***Broken URL:*** http://www.sup.org/book.cgi?id=21682 One can buy the book on the live page, but not on an archived copy [3] ***New URL for the same page:*** https://www.sup.org/books/title/?id=21682
Missing copy	***Broken URL:*** http://elections.nytimes.com/2010/house/new-york/03 There is no non-erroneous archived copy of the page's original URL, but the page is still on the web ***New URL for the same page:*** https://www.nytimes.com/elections/2010/house/new-york/3.html

Table 1: Examples of problems associated with using archived copies in cases where the page still exists at an alternate URL.

or any website's provider can augment broken external links that appear on the site's pages. In either case, when a user encounters a broken link while surfing the web, Fable's goal is to direct the user to its predicted new URL for the linked page. Fable's prediction can be occasionally incorrect. But, in the common case, Fable enables the user to not have to forego the information/functionality available on the page when no archived copy exists or if the user finds the copy to be insufficient. In enabling and evaluating this capability, we make three contributions.

First, we show how to accurately find the alias for most broken URLs that have one. We find that trying to find a live page whose content/title is similar to the last archived copy of a page's old URL does not work well; a page's content often significantly changes over time, and many pages on a site share similar titles. Instead, when a site is reorganized, we observe that the URLs of many pages change, and the new URLs are derived in a similar manner from their corresponding old URLs. Therefore, Fable unearths the aliases for a set of similar broken URLs (e.g., URLs in the same directory) by looking for a common URL transformation pattern from these URLs to the URLs of pages that currently exist on that site.

Second, in order to interactively point a user to the corresponding alias when they visit a broken URL, we judiciously split responsibilities between Fable's frontend and backend service. The backend analyzes broken URLs in batches to identify patterns evident in how the URLs on any site have been transformed when the site was reorganized. When a user subsequently visits a URL and encounters an error, Fable's frontend applies the relevant URL transformation pattern to locally predict the alias (if one exists). Fable's mapping of a URL to its alias typically completes during the time the user loads and inspects the latest archived copy for that URL.

Lastly, to the best of our knowledge, we present the first large-scale study of the prevalence of new URLs for broken links on the web. We have used Fable to identify aliases for 20K broken external links found on the pages of Wikipedia, Stack Overflow, and Medium. We estimate that the fraction of these linked pages for which Fable correctly found the new URL to be 1.5x that feasible with prior approaches that solely rely on similarity in page content.

Site	#Pages	#Unique links	#Broken links (%)
Wikipedia	40,000	1,024,435	297,594 (29.0%)
Medium	188,051	393,636	66,014 (16.8%)
Stack Overflow	265,027	161,454	30,920 (19.2%)

Table 2: Sizeable fraction of external links are broken: On three sites, number of pages crawled, number of unique external links seen, and the number and fraction of those links which are broken today.

2 MOTIVATION

We begin by quantifying the prevalence of broken links on the web. We also discuss the limitations of existing solutions for this problem.

2.1 Prevalence of broken URLs

We inspect the status of external links included on three sites: Wikipedia, Medium, and Stack Overflow. We select these as examples of sites whose pages 1) will be of value even several years after they are posted, 2) are not behind paywalls, and 3) often contain links to pages on other sites. On Wikipedia, we randomly sample the articles in which the Internet Archive has found at least one broken external link [10]. For Stack Overflow and Medium, we sample from all pages on these two sites using a data dump from Stack Exchange [23] and the sitemap for Medium [17].

For every link that we crawl, we declare the link as broken for one of three reasons: 1) we were unable to issue an HTTP request (e.g., DNS failure, TCP/TLS connection timeout), 2) we received an HTTP 404 error response, or 3) we deem the link to be a soft-404. For the last of these, we declare a URL u as broken if requests for it and for another randomly generated (and hence, invalid) URL u' redirect to the same target which is not the site's login page; we generate u' by replacing the suffix in u following the last occurrence of '/' with a random string of 25 characters [30].[1] While our detection of soft-404s misses some broken URLs (e.g., our method does not work when the URL to a site's homepage is broken, and it misses

[1] Additionally, 1) for URLs with a numeric string in the middle (e.g., article IDs in the URLs for news articles), we also consider u' where we replace this token since this string may dictate the server's response, and 2) if the response to u contains a canonical URL [9] within the HTML, we find that almost always indicates a non-erroneous response.

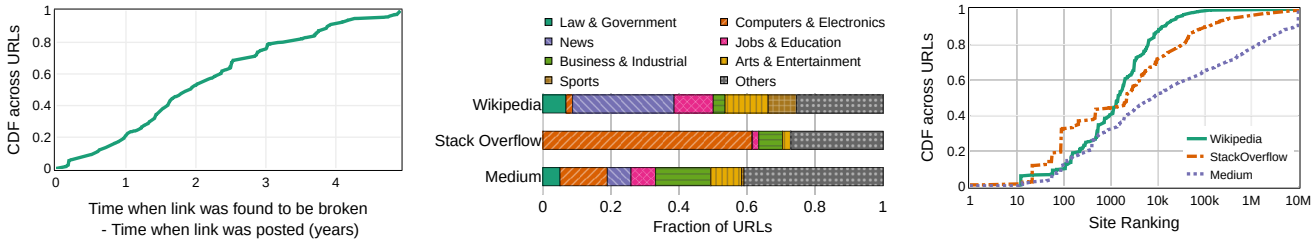


Figure 1: Links break a few years after they are posted, and broken links point to a wide variety of external sites: (a) Among the external links posted to Wikipedia since 2017, for 2000 randomly sampled links which are dead now, time gap between when link was created and when the InternetArchiveBot [14] first found the link to be broken. Distribution across broken URLs along two dimensions: (b) category and (c) Alexa ranking of the domains to which these URLs belong.

erroneous 200 status code responses [67]), we ensure that we do not classify a working URL as broken.

As seen in Table 2, 17–29% of external links are broken. Moreover, for every broken link in a Wikipedia article, we can identify the time gap between when the link was first added to the article and when it was marked as broken by the InternetArchiveBot [14]; we do so by using the article's edit history to inspect all versions of the article. We see that most broken links worked for only a few years after they were first created; Figure 1(a) shows that the median broken link became dysfunctional less than two years after it was posted.[2] These numbers do not bode well for the long-term sustainability of pointers to external information and services.

Though we focus on broken links found on the pages of three sites, the URLs corresponding to these links are from a wide range of domains, both with respect to site category and site popularity. We use the Public Suffix List [21] to map any URL's hostname to its domain, and query Klazify [16] and Alexa [8] to map any domain to a category and its ranking. Figures 1(a) and 1(b) show that there are some differences across the three sites we crawled, e.g., broken URLs found on Stack Overflow are predominantly from "Computers & Electronics" sites, whereas pages on Medium link to more broken URLs from lower-ranked domains. But, in all three cases, the broken external URLs point to pages from a wide variety of sites.

Note that, although our dataset only includes external links from user-generated content platforms (because, as mentioned above, these sites do not impose any limits on how many pages can be crawled), broken links are a pervasive issue, affecting a wide range of websites. For example, prior studies have observed that approximately 25% of the external links in articles published by The New York Times have become inaccessible [68], and at least 23% of the links included in the citations from research articles are invalid [52]. This rate of link rot aligns closely with our own findings across three different websites.

[2]Given the overheads involved in examining all versions of every article without exceeding Wikipedia's crawling limit, we restrict this analysis to a random sample of 2000 links from all the broken external links we found on Wikipedia. We find that the distribution in Figure 1(a) is similar for different randomly chosen subsets of 500 from these 2000 links.

2.2 Coping with broken URLs.

Reliance on web archives. The common recourse for dealing with broken links is to rely on archived copies of the linked pages, e.g., the Internet Archive patches broken links in Wikipedia articles by linking to archived snapshots of these links [27]. We observe that this method of coping with a broken link to a page suffices only if three criteria are met: 1) an archived copy of that page exists, 2) that copy reflects the page's content at the time the link was created, and 3) the intent in linking to that page was to point visitors to the content that existed on the page at that time. In practice, at least one of these properties is often untrue. Due to budgetary constraints, web archives do not have any copies for many URLs [28, 29] or attempt to archive a URL only once it no longer works [57]. On the other hand, many links on the web are either navigational (i.e., to point users to whatever content happens to be at the linked page when they follow the link) or transactional (i.e., help users discover some app-like functionality) [31]. For such links, users have to make do with potentially stale content in the archived copies or miss out on some of the functionality on the page.

To demonstrate the prevalence of these problems on the modern web, we study a random sample of 500 broken URLs from Table 2.

- First, 143 (28%) of these URLs have no archived copies on the Internet Archive's Wayback Machine.
- Second, of the remaining 357 URLs, 269 have multiple archived copies. For 29% of these 269 URLs, the core content of the page (i.e., parts of the page that remain after filtering out boilerplate [12, 50] such as sidebars, recommendations, etc.) differs significantly between the earliest and latest copy; based on prior work [59], we consider a TF-IDF similarity [62] of less than 0.8 as evidence that the page has been significantly modified. For these pages which are updated over time, even the last archived copy is likely to have stale content.
- Lastly, our manual analysis reveals that the fraction of pages which include functionality that requires client-server interactions (e.g., comment, purchase) increased from 29% before 2010 to 69% after 2015. This suggests that functionality that will not work on archived copies has become more prevalent over time.

Making URLs resilient. Users do not have to make do with archived page copies in cases where a link to a page is broken only

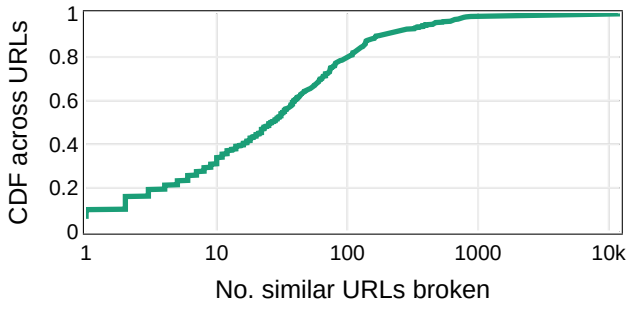


Figure 2: Many URLs on a site go dead together: For each of 500 broken URLs, number of similar URLs on the same site which also stopped functioning after the URL became dysfunctional. Note logscale on x-axis.

because the page's URL has changed; see Table 1 for examples. To preempt such link breakages, one could rely on content-based addressing, such as IPFS [15], wherein pages are retrieved by their content hashes instead of their locations. However, such an approach works well only for static blobs of content, and is ill-suited for navigational links to pages whose content is updated over time.

Alternately, to serve requests for any page irrespective of whether it is requested using its old or new URL, the provider of that page can include a page-specific unique ID in the page's URL [51]. But, website providers need to then make sure to preserve every page's ID in its URL even after reorganization. We find that sites which include page IDs in their URLs often violate this requirement. For example, the page that was previously at http://www.technologyreview.com/article/419483/measure-for-measure/ is now at https://www.technologyreview.com/2010/06/22/202620/measure-for-measure/; the page-specific ID included in the page's URL has changed from *419483* to *202620*.

Rediscovery of missing pages. To sidestep the challenges associated with making URLs on the web resilient to site reorganizations, prior work [41, 46] has studied how to discover the new URL (i.e., alias) corresponding to any broken URL. Mimicking what a human user is likely to do, they retrieve a copy of the page that was previously available at the URL which is now broken, extract a variety of features from this copy (such as page title, lexical signature [60], etc.), and then use these features to query web search engines. Existing solutions simply show the search results to the user, and they let the user identify which result, if any, is a link to the same page. One could instead automate the process of checking if a search result is a close match to the page's stored copy.

We observe that such an approach for discovering a page's new URL is prone to several fundamental limitations.

- *Poor accuracy:* First, the archived copy of the broken URL and the page that currently exists at any of the search results might share similar features only because they happen to be two similar pages. For example, the archived copy of http://marvel.com/comic_books/issue/22962/what_if_2008_1 [2] has the title ("What If? (2008) #1"). Though the live page at https://www.marvel.com/comics/issue/22964/what_if_2008_1 has the same title and similar content, it is a page for a different book.

1	URL	cbc.ca/news/story/2000/01/28/pankiw000128.html
	Title	Pankiw will not be silenced
	Alias	cbc.ca/news/canada/pankiw-will-not-be-silenced-1.249577
2	URL	cbc.ca/news/story/2000/07/12/mb_120700Potter.html
	Title	Potter book flies off shelves
	Alias	cbc.ca/news/canada/potter-book-flies-off-shelves-1.201722
3	URL	cbc.ca/news/story/2000/07/04/rancher000724.html
	Title	Rancher survives tornado
	Alias	cbc.ca/news/canada/rancher-survives-tornado-1.215189

Table 3: New URLs for the pages on a site are typically derived in the same manner from their old URLs: Example of a group of 3 broken URLs, their titles and corresponding aliases.

- *Poor coverage:* Attempting to find a page that is similar to a broken URL's last archived copy will fail to find any match when no copies exist for that URL – which is common, as we showed earlier in this section – or when the page has been significantly modified since it was last archived.[3] To improve coverage, one could reduce the degree of similarity with the archived copy needed for a search result to be considered a match. But, this will further degrade accuracy.
- *Poor efficiency:* Finally, the workflow outlined above requires performing multiple web search queries and crawling tens of web pages to find the alias for each broken link. Doing so offline for all broken links across the entire web is not practical and will make online low-latency discovery of aliases infeasible.

3 OVERVIEW OF FABLE

Guiding observations. We design Fable to help users rediscover the live page once the old links to it break because of a change in the page's URL. To address the aforementioned limitations of existing solutions for doing so, Fable's distinguishing characteristic is that it is not reliant on checking whether a page's content/title before its URL changed is similar to what is currently on the page. We observe that, when a URL ceases to work, it is usually the case that other similar URLs on that site (e.g., those in the same directory) also suffer a similar fate. Figure 2 demonstrates this property for 500 URLs chosen at random from those which have at least one successful and one erroneous archived copy on the Internet Archive. For the median URL u, we find that 26 other similar URLs (i.e., with the same prefix until the last '/' in u) were originally functional but stopped working after u ceased to work; they were successfully archived by the Internet Archive at least once but their archived copies after u stopped functioning are erroneous.

In addition, we posit that changes in page URLs are typically the result of programmatic reorganization of an entire site or subdomain, rather than manual movement of pages. Consequently, we observe that there is usually a pattern in how the old URLs for pages on a site map to their corresponding new URLs. Table 3 shows an example.

[3] Once a broken URL's alias is discovered, determining whether the page's content has sufficiently drifted that it no longer serves the purpose for which it had been linked to is a separate problem [44, 68] that is outside the scope of this work.

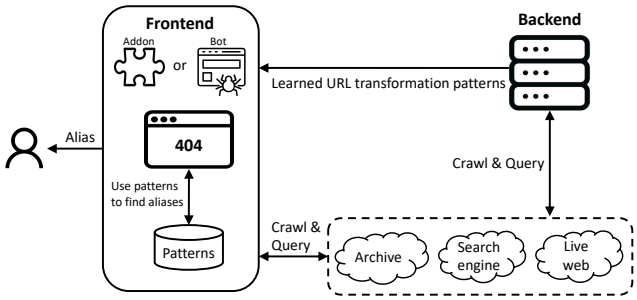


Figure 3: Overview of Fable's architecture.

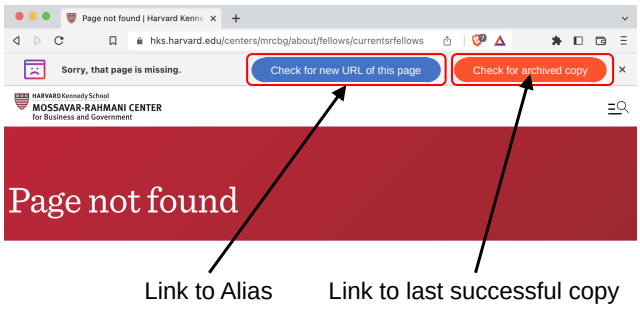


Figure 4: When a user who has installed Fable's browser add-on visits a link that returns an error, the add-on provides links to the latest archived copy (if one exists) and to the link's functional alias (if one is identified).

Workflow. To exploit the above observations, Fable combines a frontend – either a browser add-on or a bot that rewrites links on pages – and a backend service; see Figure 3.

When a user who uses Fable's browser add-on visits a broken URL, it gives the user the option of visiting either an archived copy of that URL or Fable's predicted alias for the URL; Figure 4 shows the user interface. The frontend predicts the alias based on URL transformation patterns that it has previously received from the backend. Alternately, website providers can use Fable's link rewriter in a manner similar to the InternetArchiveBot [14], which augments broken links on Wikipedia with links to archived page snapshots. Fable's bot can scan all pages on a site, identify the external links that do not work, use previously identified URL transformation patterns to try to identify an alias for each such link, and rewrite the link to add the alias if one is identified.

In either incarnation of Fable's frontend, we take care to ensure that the alias that it identifies for a broken link is presented as an alternative to, not a replacement for, the original link. Thus, when the identified alias turns out to be incorrect, the user still retains the option to visit the original link. Existing systems [19, 22] that augment web links with pointers to archived page copies similarly give users the option to visit either the live page or the archived copy. Moreover, for any broken URL, Fable restricts its attempt to find the alias to an alternate URL on the same site. Therefore, even if the URL transformation pattern shared by the backend is incorrect (either because it is buggy or compromised) or Fable's predicted alias turns out to be wrong, the user is trusting the same site provider when visiting the predicted alias as when the user chooses to follow the original link.

Both the add-on and the page rewriting bot rely on URL transformation patterns discovered by Fable's backend. For this, the backend first groups together all broken URLs that are in the same directory on a site. It then considers one group at a time and attempts to find aliases for all URLs that it is aware of in that group. In doing so, the backend learns how these URLs were transformed into their corresponding new URLs.

In our current implementation, the backend discovers broken URLs simply by crawling the public web. To better scale Fable and to help the backend discover more broken links on the web (e.g., links on pages that are not publicly accessible), an alternative design could have frontends share with the backend broken URLs that they are unable to map to aliases. However, this would have to be done with care, as the URLs shared by a frontend may potentially reveal sensitive information. Designing a privacy-preserving mechanism for this purpose is beyond the scope of our work.

Goals. In designing Fable, we are guided by three goals, in the following order of priority.

- **Accuracy.** Of the aliases found by Fable, we strive to maximize the fraction which are correct. If many of the identified aliases are suspect, users and website providers are less likely to use it.
- **Coverage.** We seek to find aliases for as many input URLs as feasible. Ideally, if the page that a URL was previously pointing to still exists, Fable should be able to find the page's new URL.
- **Efficiency.** We aim to minimize 1) the work (number of pages crawled, search queries issued, etc.) that Fable's backend must do to locate aliases for dead links, and 2) the delay between when a user encounters a broken link and when Fable is ready to direct the user to the alias.

4 DESIGN

We describe our design of the Fable backend and frontend by answering three questions.

- Given a set of similar broken URLs, how can the Fable backend find aliases for all of them, if the pages that those URLs previously pointed to still exist?
- In what form should the Fable backend represent the URL transformation patterns that it learns?
- How should a Fable frontend use these patterns to discover the alias for any dysfunctional URL?

Our reliance on the consistency in how old page URLs on a site map to the corresponding new URLs helps maximize Fable's coverage and accuracy. Whereas, we make Fable efficient by 1) reducing the need for the backend to query web search engines and crawl pages from the web, and 2) maximizing the frontend's ability to find aliases locally. Table 4 summarizes the key techniques.

4.1 Backend design

For all the broken URLs that the Fable backend collects – either by crawling the web (like in our current implementation) or via

Component	Goal	Technique	Section
Backend	Reduce volume of web search queries	Find aliases by identifying non-erroneous historical redirections	§4.1.1
	Minimize need to crawl pages from the live web Enable better accuracy than matching based on similarity in page content	Match broken URLs to search results based on URL transformation patterns	§4.1.2
Frontend	Minimize need to crawl or query any external service Find aliases for URLs with no archived copies	Infer aliases based on the transformation patterns learned by backend	§4.2.1
		Skip URLs for pages that are likely deleted	§4.2.2

Table 4: Summary of the techniques used in FABLE to maximize accuracy, coverage, and efficiency.

uploads from frontends – it attempts to discover the corresponding aliases using two methods. The primary goal here is to learn the patterns underlying how these URLs have been transformed so that, when a frontend encounters different but similar broken URLs, it can use these patterns to locally identify their aliases.

4.1.1 *Leverage historical redirections.*

Approach and challenge. The high-level observation that guides our first approach is that many URLs that no longer work today did previously redirect to the corresponding aliases. In other words, after reorganization, some sites initially redirect requests for any page's old URL to the new URL for that page. But, these sites subsequently lose the state necessary to perform such redirections. For example, the URL http://www.kde.org/announcements/announce-1.92.htm does not work today, but it previously redirected to its alias http://www.kde.org/announcements/announce-1.92.php [1].

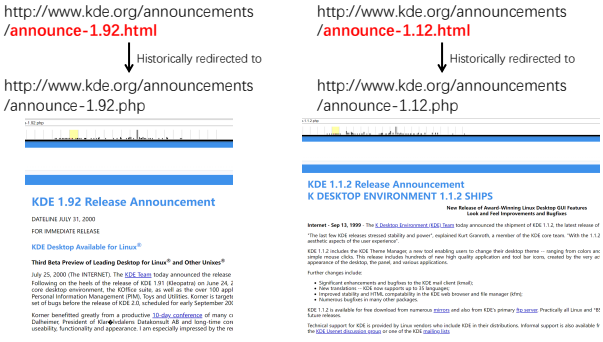
For any broken URL, we attempt to find such a historical redirection to the URL's alias by inspecting all archived copies for that URL on the Wayback Machine. The challenge here is that not all redirections are necessarily correct. Some redirections correspond to soft-404s where requests for a URL redirect to an unrelated URL.

Solution: Compare redirections for similar URLs. To help differentiate valid and invalid redirections, we leverage the fact that, for any URL that it has archived, Wayback Machine almost always also has archived copies for a number of other URLs in the same directory, i.e., have the same prefix until the last '/'. To account for dates and article IDs in URLs, we ignore any numbers at the end of each URL's prefix, e.g., the broken URLs in Table 3 are considered to be in the directory "cbc.ca/news/story/".

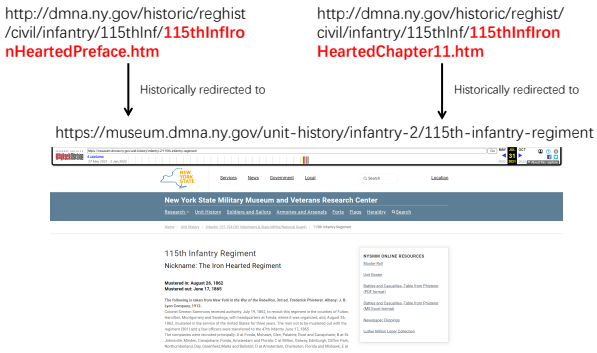
We adapt as follows our method from §2 for detecting soft-404s. Instead of comparing the response for a target URL with that site's responses for similar randomly generated URLs, we compare the historical redirection seen in the archived copy of a target URL with temporally nearby (within 90 days on either side) 3xx status code archived copies for up to 3 other URLs in the same directory. FABLE considers the historical redirection for a URL to be correctly pointing to the URL's alias only if the target of redirection is unique compared to the responses seen for other similar URLs. Figure 5 shows examples of correct and incorrect historical redirections identified using this method.

4.1.2 *Match URLs to search results.*

When the FABLE backend does not find any correct historical redirection from a broken URL to its alias, it relies on web search using the title/content from the last archived copy for that URL. However,



(a) Example good historical redirections



(b) Example erroneous historical redirections

Figure 5: Comparing historical redirections of similar URLs helps us identify which redirections are soft-404s.

FABLE does not compare the archived copy's content to any live page on the same site. Consequently, for most URLs, the backend does not crawl the pages at *any* of the search results in order to determine if one of them is a match.

Approach and challenges. After grouping broken URLs that are in the same directory, for each URL u_i in a group, the backend considers the URLs $\{u_{i1}, \ldots, u_{im}\}$ of the search results. It declares URL u_{ik} as the alias for u_i if the transformation from u_i to u_{ik} is such that other URLs in the group also have a search result with the same transformation.

To realize the above methodology, we need to characterize the transformation from every URL to each of its search results. Doing so is challenging for two reasons. First, not every portion of the new

URL	Alias candidate from search results	Pattern
U1: solomontimes.com/news.aspx?nwid=1121 *Title*: "No Need for Government Candidate: CEO Transparency Solomon Islands"	*C11*: solomontimes.com/letter/1121	solomontimes.com/UP/Pr
	C12: solomontimes.com/news/no-need-for-government-candidate-ceo-transparency-solomon-islands/1121	solomontimes.com/Pr/Pr/Pr
	C13: solomontimes.com/news/governments-prime-minister-candidate-pledges-reconciliation-as-priority/1112	solomontimes.com/Pr/UP/UP
U2: solomontimes.com/news.aspx?nwid=6540 *Title*: "High Court Rules against Lusibaea"	*C21*: solomontimes.com/news/high-court-rules-against-lusibaea/6540	solomontimes.com/Pr/Pr/Pr
	C22: solomontimes.com/news/high-court-to-review-lusibaea-case/5862	solomontimes.com/Pr/UP/UP
	C23: solomontimes.com/news/lusibaea-released-opposition-uproar/5814	solomontimes.com/Pr/UP/UP

Table 5: Step 1 of matching broken URLs in a directory to their aliases: Map the URL of every search result to a coarse-grained transformation pattern. Pr: *Predictable*, UP: *Unpredictable*

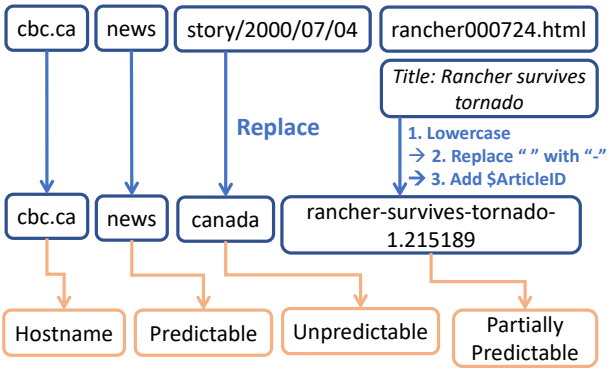


Figure 6: Contrast between precise and coarse-grained characterizations of URL transformation: How a human is likely to characterize the transformation from the third URL in Table 3 to its alias (blue arrows) versus the simplified pattern used by Fable (orange arrows).

Pattern	URL, Candidate
solomontimes.com/Pr/Pr/Pr	(*U1*, *C12*)
	(*U2*, *C21*)
solomontimes.com/Pr/UP/UP	(*U1*, *C13*)
	(*U2*, *C22*)
	(*U2*, *C23*)
solomontimes.com/UP/Pr	(*U1*, *C11*)

Table 6: Step 2 of matching broken URLs in a directory to their aliases: (Broken URL, alias candidate) pairs from Table 5 with the same coarse-grained pattern are clustered together. Candidates in the top cluster are chosen by the backend as the predicted aliases.

URL can be derived from the original URL and associated metadata (such as page title). For example, a new page ID may be introduced, as seen in Figure 6. Second, the set of transformation operations is unbounded, and the complexity of deriving the transformation between any two arbitrary URLs is exponential with respect to the number of operations in the transformation [43]. We find that even a well-optimized pattern inference engine such as Microsoft Excel's Flash Fill [25] takes over 5 seconds for the median (URL, alias candidate) pair. Therefore, deriving the transformation from every input URL to every search result for it will impose significant overhead.

Solution: Coarse-grained transformations suffice. We observe that it is unnecessary to precisely derive URL transformations because valid URLs within any site are sparsely distributed across the site's namespace. For example, given that the site cba.ca has a page whose URL ends with "rancher-survives-tornado", it is very unlikely for it to also have a page whose URL ends with "rancher_survives_tornado".

Hence, it suffices to characterize URL transformations in a coarse-grained manner as follows. When comparing two URLs – an input broken URL and the URL for an alias candidate (i.e., a search result) – we split up either URL into its components using '/' as the delimiter. We then tokenize the URL components and the page title in the input URL's last 200 status code archived copy using all non-alphanumeric characters as delimiters. We classify every component in the candidate alias URL as *Predictable*, *Unpredictable*, or *Partially predictable* depending on whether the component's tokens are a strict subset, have no overlap, or partially overlap with the set of tokens from the input URL and archived page title.[4] Figure 6 shows an example of how Fable's backend maps every alias candidate URL to a sequence of predictable, unpredictable, and partially predictable components.

Within each group of similar broken URLs, once the backend establishes the predictability for each URL-candidate pair (as shown in Table 5), it identifies which (if any) of the candidates is the alias for each URL in the group. To do so, as shown in Table 6, the backend clusters candidates which are mapped to the same pattern, i.e., the same sequence of predictability for the URL's components. It picks the cluster in which the pattern has the highest number of *Predictable* and *Partially predictable* URL components. If there is a tie, the backend picks the cluster which contains candidates for more broken URLs. In rare cases, the top ranked cluster could have multiple candidates for the same broken URL. Only in such cases does the backend need to crawl the live pages corresponding to the alias candidates and check if any of them is a match based on title/content.

Of course, not all broken URLs have aliases, and the alias for a URL may not appear in the search results for it. Therefore, if the top ranked cluster 1) has only one broken URL (i.e., there is no consistent transformation pattern observed across the URLs

[4]To classify a URL component as partially predictable, we also require at least half of its 2-grams (i.e., every pair of consecutive tokens) to overlap. This accounts for unrelated URLs which happen to have similar tokens, e.g., "chili_peppers_camron_top_the_chart" and "red-hot-chili-peppers-attack-the-chart-116269".

Old URL	New URL	Processed URL pattern
w3schools.com/html5/tag_i.asp	w3schools.com/tags/tag_i.asp	w3schools.com/tags/Pr
w3schools.com/html5/att_video_preload.asp	w3schools.com/tags/att_video_preload.asp	
w3schools.com/html5/html5_geolocation.asp	w3schools.com/html/html5_geolocation.asp	w3schools.com/html/Pr
w3schools.com/html5/html5_webstorage.asp	w3schools.com/html/html5_webstorage.asp	

Table 7: Example demonstrating challenges in using PBE. Old URLs within the same directory on a site get mapped to multiple new directories, and not all components in the new URLs are *"Predictable"*.

in a group), or 2) has no *Predictable* or *Partially predictable* URL components (i.e., the alias candidates have nothing in common with the original URLs), then we declare that no aliases were found for the input broken URLs.

4.2 Frontend design

The most straightforward way for the Fable frontend to leverage work done by the backend would be as follows. When, for example, a user visits a broken link, the frontend can lookup the latest archived copy of that link and issue search queries based on the title/content in that copy. The frontend can identify that search result as the alias which matches the coarse-grained pattern associated with the top cluster identified by the backend for this URL's directory. This procedure can be completed in less than 10 seconds for the median URL (§5.2). It also helps preserve user privacy, since any user's frontend installation does not need to share the URLs visited by the user with Fable's backend.

However, requiring the frontend to lookup an archived copy and issue a search query for every broken URL has several limitations. First, the frontend would be unable to find the alias for any URL with no archived copies. Second, when a user of Fable's browser add-on visits a broken URL, if an archived copy exists and the user chooses to inspect it first, the add-on can use this period to identify the URL's alias. But, the user might choose to visit the alias without inspecting the archived copy. Therefore, the frontend must aggressively minimize the delay that it incurs in identifying the alias for any given broken URL. Lastly, even for URLs of pages that no longer exist, the frontend will unnecessarily lookup archived copies and issue web searches in the attempt to find aliases for these URLs.

To deal with these shortcomings, in addition to the coarse-grained transformation pattern for every directory, the Fable backend shares two additional takeaways from its work in finding aliases.

4.2.1 *Transformation programs to infer aliases.*

Approach and challenges. First, given a broken URL, the frontend attempts to locally generate the alias without performing any web search. To enable this, whenever feasible, the backend uses the aliases that it discovers for the URLs in a directory to learn a precise URL transformation function for that directory.

We find programming by example (PBE) [39] well-suited for this purpose. The PBE approach entails providing a number of input-to-output examples, using which a program can be synthesized to transform input to output. Figure 7 shows an example of how a program generated from several (URL, alias) pairs can be used by the frontend to then locally predict the alias for any other broken

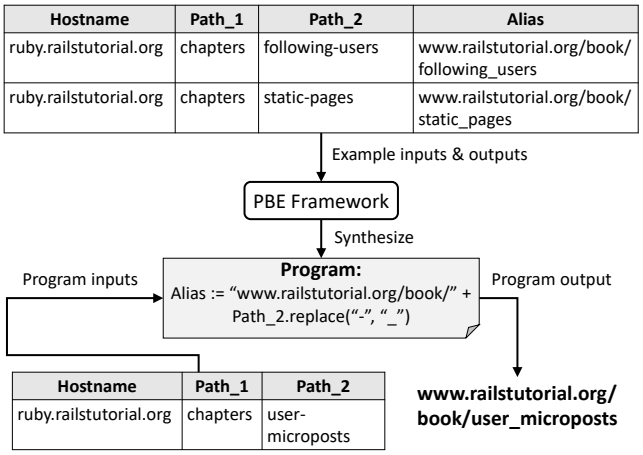


Figure 7: Programming by example in action. PBE framework takes in example inputs and outputs to learn a program, which can then be used to similar inputs into output.

URL in the same directory. We leverage Microsoft's PROSE PBE framework [20], which is the prediction engine underlying Excel's Flash Fill [25]. For every broken URL, apart from tokens in the URL, the program also takes as input auxiliary information such as page title and date of creation [18], which we extract from archived copies of that URL, when available.

The question then is: when and how to use PBE to enable the Fable frontend to locally generate the alias for any given broken URL? On the one hand, it is not always feasible to use this approach, e.g., in cases like in Figure 6 wherein a page's new URL includes a new page ID. On the other hand, attempting to learn a URL transformation program only when all components are deemed *"Predictable"* in the coarse-grained pattern for a directory would be too restrictive. For example, for the broken URLs in Table 7, it is intuitive to see the feasibility of learning the underlying transformation pattern, but some of the tokens in the new URLs (e.g., *"tags"*, *"html"*) are not derived from the original URLs. Moreover, PBE relies on learning a single program based on all inputs. Whereas, as seen in Table 7, URLs in the same directory can end up getting mapped to new URLs in different directories.

Solution: Group example inputs with similar aliases, and learn one program per group. We address all of the above-mentioned challenges as follows. First, we observe that learning a single program for all broken URLs in a directory suffices when all URL components are *"Predictable"* in the coarse-grained pattern

learned for this directory. When there are *"Partially predictable"* or *"Unpredictable"* components, there is ambiguity about whether multiple programs need to be inferred or no precise transformation function can be learned. To distinguish between these cases, second, we split up the broken URLs in a directory such that all aliases in a partition have the same prefix. If the URL components following the common prefix are not predictable, like for cbc.ca's URLs in Table 3, then we declare that no precise URL transformation program can be learned. If the components other than the prefix are predictable, like in Table 7, then we attempt to infer multiple programs for the same directory, one for each partition. Given a broken URL, the frontend applies every program learned for this URL's directory, and at most one of these inferred aliases will prove to be a functional URL.

4.2.2 Identifying deleted pages.

The second optimization in Fable's frontend is to skip doing work for URLs which are broken because the pages they previously led to no longer exist.

For this, we observe that, even when Fable's backend is unsuccessful in finding aliases for the first few URLs within a directory, this provides valuable information. Specifically, for the first four URLs in a directory, if Fable neither 1) finds any alias, nor 2) finds any candidate where the last URL component is either *Predictable* or *Partially predictable*, then we empirically observe that it finds no aliases for any other URLs in that same directory with high likelihood. The backend shares with every frontend all such directories in which it believes attempting to find an alias for the URLs in those directories is futile.

Thus, when a Fable frontend encounters a broken link on the web, it first checks if the URL matches a directory that likely corresponds to a set of deleted pages. If not, it applies any URL transformation programs applicable to this directory and checks to see if any of them yields a URL corresponding to a live page. If still unsuccessful, the frontend eventually falls back to issue a search query and checks if any result matches the coarse-grained pattern for aliases identified for this URL's directory by the backend.

5 EVALUATION

We evaluate Fable from three perspectives: 1) its ability to correctly discover the new URLs for broken links which do have functional aliases, 2) its efficiency in finding aliases, and 3) the utility of directing users to the aliases it finds instead of serving archived page copies. We compare Fable with two classes of prior solutions, described earlier in §2: 1) content hash-based addressing to make URLs resilient, and 2) reliance on similarity in page content/title to rediscover missing pages. We refer to our implementations of these two classes of solutions as ContentHash and SimilarCT.

The main takeaways from our evaluation are:

- When we compare Fable to prior solutions on a dataset of 20K broken external links sampled from pages on Wikipedia, Medium, and Stack Overflow, Fable finds correct aliases for 51% more broken links.
- Compared to SimilarCT, Fable's backend needs to crawl only $\frac{1}{20}^{th}$ as many pages from the live web and issue $\frac{2}{3}^{rd}$ as many search engine queries to find aliases for the same number of broken URLs.
- For the median broken URL, Fable's frontend completes its attempt to find the URL's alias in less than 10 seconds. This is more than 30 seconds lower than the latency with SimilarCT.
- Our analysis of a random sample of 100 broken URLs for which Fable finds aliases highlights the importance of not always relying on archived page copies: 9% have not been archived by Wayback Machine, 24% have stale content, and 70% include functionality that does not work on archived copies.

5.1 Coverage and Accuracy

We begin our evaluation by considering two questions: How many broken URLs is Fable able to find aliases for? What fraction of the aliases it finds are correct? Given the lack of ground truth data mapping old page URLs to the corresponding new URLs, prior work on rediscovering web pages [41, 46] only considered: given a cached copy of a live web page, do the search queries that they extract from that copy yield that page in the top few search results? To instead perform a more realistic evaluation, we first assemble a dataset of 1000 broken URLs where we know the correct alias (if one exists) for every URL. We then apply Fable's techniques on a larger dataset of 20K broken URLs where we lack ground truth.

5.1.1 Ground truth dataset.

Our ground truth dataset comprises two sets of 500 dysfunctional URLs each, which exploit site operators' intent evident in how those sites either previously responded or currently respond to requests for these URLs.

- We compile the first set (*Alias*) using a manually verified set of historical redirections, i.e., for every broken URL in this set, via manual examination of its archived copies on the Wayback Machine, we can confirm that the URL that it previously redirected to is indeed its alias.
- The second set (*NoAlias*) comprises URLs that previously pointed to pages which likely no longer exist now. We identify such URLs as ones that today elicit 410 status code responses, which indicate that *"access to the target resource is no longer available at the origin server and that this condition is likely to be permanent"* [7]. While we cannot confirm whether or not any particular 410 response is erroneous, whenever we do find an alias for any of these URLs, we manually check its correctness.

Since our knowledge of aliases here is based on redirections found on Wayback Machine, we withhold 3xx status code archived copies from Fable when running it to find aliases for broken URLs in this dataset. For the prior approaches that we compare against, we optimize their performance as follows.

- For ContentHash, we filter out every page's boilerplate content using Chrome's distiller [12] before computing hashes. This helps eliminate any changes caused by page templates, content recommendations, etc.
- For SimilarCT, as we mentioned previously, prior work leaves it to the user to decide if any of the search results for a URL is that URL's alias. To instead offer the same interface as Fable, we consider a search result to be a match if it is the only one whose title or content has a TF-IDF similarity [62] of 0.8 or more with the archived copy; prior work [59] recommends that a threshold

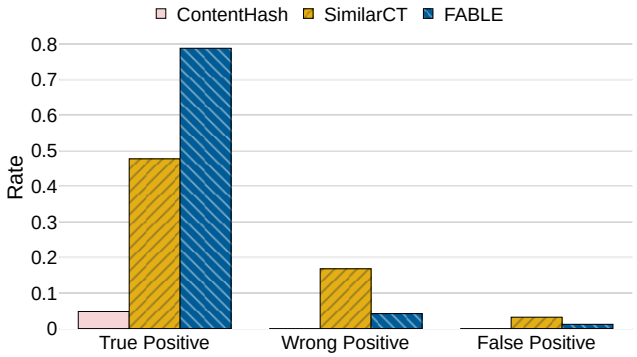


Figure 8: Compared to alternate approaches, Fable significantly improves coverage and accuracy. On a dataset of 500 URLs for which aliases are known and 500 URLs which likely have no alias, the fraction of true positive, wrong positive, and false positive matches found by Fable and existing approaches.

of 0.8 is appropriate to detect if a page has been only slightly modified.

Figure 8 shows the true-positive rate (fraction of *Alias* URLs which are matched to their known alias), wrong-positive rate (fraction of *Alias* URLs matched to an incorrect alias), and false-positive rate (fraction of *NoAlias* URLs for which a match is found).

True positives. First, Fable achieves a higher true-positive rate than prior approaches – close to 80% versus less than 50% – because of 1) its reliance on URL transformations, not content similarity, to match broken URLs to search results, and 2) its inference of aliases for URLs which have either never been archived or whose alias does not appear in the search results. For example, both existing approaches fail to match the broken URL https://www.udacity.com/courses/cs262 to its alias http://udacity.com/course/programming-languages--cs262; since the class has been updated over time, both the title and content of the page have significantly changed. In contrast, Fable is successful because it observes search results with the same URL transformation for other classes with similar URLs, e.g., https://www.udacity.com/course/2d-game-development-with-libgdx--ud405 is one of the alias candidates for http://udacity.com/courses/ud405. Moreover, Fable successfully discovers the vast majority of true positives found by SimilarCT; for only 4.4% of the 500 URLs in the *Alias* dataset, SimilarCT matches a broken URL to its correct alias without Fable being able to do the same.

Note that the true-positive rate that we observe for SimilarCT is significantly lower than that claimed by prior work [46]. We believe this is because prior work evaluated their techniques on functional URLs and "pretend" that they are broken. In contrast, our ground truth dataset consists of URLs that are truly broken. Since broken URLs are likely to have fewer recent archived copies than functional URLs, there are potentially many more changes between the last archived copy and the live page.

We find that the following factors prevent Fable from achieving a true-positive rate of 100% on this dataset.

- First, Fable fails to match 3% of the *Alias* URLs to their known alias due to the incompleteness of Google's and Bing's search indices; the aliases for these URLs do not appear in the search results even though the archived copy from which Fable formulates search queries closely matches the live page at the alias URL today, i.e., TF-IDF similarity between the latest archived copy and the live page is more than 0.8.
- For another 8% of the *Alias* URLs, neither does the alias appear in the results for Fable's search queries (because content on the live page has been updated significantly since when it was last archived) nor is Fable able to infer the alias (because it could not learn any precise URL transformation program from its attempts to find aliases for other URLs in the same directory).
- The remaining missed aliases can be attributed to Fable's procedure for matching every URL to one of the search results for it. Either Fable's criteria end up being too strict (for 6% of *Alias* URLs), or it picks incorrectly between multiple observed URL transformation patterns and matches the URL to an incorrect alias (for 4% of URLs).

Wrong and false positives. Second, SimilarCT results in more wrong-positives than Fable because the former can end up matching a broken URL to a different page that happens to have similar content, like in the example from marvel.com that we discussed in §2. Fable avoids doing so by examining other broken URLs in the same directory and their candidates, e.g., Fable correctly chooses https://www.marvel.com/comics/issue/22962/what_if_2008_1 as the alias. Although ContentHash has no wrong-positives or false-positives, its low true-positive rate means it has little practical value for making links to web pages resilient.

Finally, Fable's false-positive rate is quite low at ~1%. These false matches occur when, for a bunch of similar URLs, wrong candidates which share tokens with the corresponding broken URLs appear in the search results. For example, Fable incorrectly declares https://exclaim.ca/music/article/black_mountain-wilderness_heart (an article introducing a band's new album) as the alias for http://exclaim.ca/Contests/black_mountain_wilderness_heart/ (a music contest regarding the same band) because a similar broken URL, http://exclaim.ca/Contests/we_are_city-violent, has a search result with the same change: https://exclaim.ca/music/article/we_are_city-violent. Most of the false positives identified by Fable are also flagged by SimilarCT; only 0.4% of all *NoAlias* URLs are uniquely matched by Fable to an incorrect alias.

5.1.2 *Broken links from three sites.*

Next, we study Fable's coverage and accuracy on a larger dataset of broken URLs for which we lack ground truth information. We 1) consider all the broken external links we previously observed on Wikipedia, Medium, and Stack Overflow (Table 2), 2) group them into directories, 3) choose all URLs in directories with ≥ 5 URLs (as seen in Figure 2, for 80% of broken URLs, at least 4 other URLs in the same directory also stopped working when they became dysfunctional), and 4) pick all URLs that point to functional sites with English content. We focus on links to English sites because our current implementation can compute TF-IDF similarity only for English text. Though Fable seldom relies on comparing the title/content of pages, SimilarCT is crucially dependent on this

capability. Given an implementation for calculating the similarity between text in other languages, we can easily extend our implementations of Fable and SimilarCT to those languages.

From this dataset, we randomly sample 20,000 broken URLs, comprising 12,000, 4,200, and 3,800 broken external links from pages on Wikipedia, Medium, and Stack Overflow, respectively. Given the low coverage seen on the ground truth dataset with ContentHash, on this dataset, we focus on comparing Fable with SimilarCT.

Precision. The ideal way to address the lack of ground truth information in this dataset would be to contact the operators of the sites which host the broken URLs we consider. However, such an approach is impractical given that the URLs in our dataset are spread across a large number of sites and the operators of most sites are unlikely to respond to our queries.

Therefore, we instead evaluate the quality of aliases from users' perspective, i.e., do users consider the live page that currently exists at the Fable-identified alias for a broken link to serve the purpose that the page previously available at that link was meant to serve? The alias for a broken link is useful to users when either archived copies are not an adequate replacement for the live page, or no archived copy exists for that link. Since we lack an automated approach to identify the former at scale, we focus on the latter.

We asked members of the Wikipedia community to check a random sample of the aliases found by Fable for broken references on Wikipedia which have been marked as "permanently dead" [11]. For these links, not only is the linked page no longer accessible via the original link, but no archived copy exists for users to refer to. We randomly selected 103 aliases found by Fable for permanently dead links (selecting at most 2 per domain) and posted them on a public Wikipedia page [24]. To aid users in the task of verifying the aliases, our page also listed auxiliary information for every link such as the Wikipedia article in which the link appears and the anchor text of the link. We then solicited the Wikipedia community to check these aliases and share the observations they made in doing so.

Wikipedia users determined that 89 of the 103 aliases are correct, 6 are incorrect, and they were unsure about the remaining 8. An example URL for which they were uncertain about the correctness of the alias is http://igokisen.web.fc2.com/kl.html, which a Wikipedia user linked to in 2011 to point others to results from that year's edition of the Korean Baduk League. That URL no longer works today, and Fable identified http://igokisen.web.fc2.com/kr/kl.html as the alias. But, the page at the alias lists the results from the most recent edition of the league. Since there is no archived copy for the original link, it is hard to tell whether the user who originally created the reference did not link to an appropriate page, the content on the page has drifted, or the alias found by Fable is incorrect. Depending on whether we pessimistically consider all the ones marked unsure as incorrect or optimistically consider them as correct, Fable's accuracy is between 86% and 94%, i.e., 90% on average.

Coverage. Fable finds aliases for 4680 of the 20K URLs in our dataset. Table 8 shows that the fraction of broken links for which Fable discovers aliases varies from 18.8% on Medium to 26.2% on Stack Overflow. Our success in finding aliases for more of the URLs found on Stack Overflow stems from two factors. First, as seen earlier in Figure 1, the URLs found on Stack Overflow are biased toward higher ranked domains, most of which are categorized as 'Computers & Electronics'. We hypothesize that URLs on these domains are more likely to break due to site reorganizations, not page deletions. Second, our ability to find aliases for broken URLs found on Medium is hampered by the fact that a much lower fraction of these URLs have archived copies, in contrast to URLs found on Wikipedia and Stack Overflow.

Source	DNS+	404	Soft-404	**Total**
Wikipedia	1414	7458	3128	**12000**
Medium	737	2127	1336	**4200**
Stack Overflow	413	2270	1117	**3800**
% Alias found	15.8%	23.0%	27.9%	23.4%

Table 8: In our selected sample of broken external links on Wikipedia, Medium, and Stack Overflow, Fable's success rate in finding the alias as a function of how the URL is broken. DNS+ means failed DNS resolution or TCP/TLS connection setup, and Soft-404 includes URLs which redirect to an unrelated page.

	≤'10	'10–'15	'15–'21	**Total**
No. URLs	1846	5754	6414	**14014**
% Alias found	25.1%	31.5%	31.5%	30.6%

Table 9: Fable's success rate in finding the alias for a URL as a function of how long it has been since that URL stopped working. For URLs archived by Wayback Machine before they became dysfunctional, year of last archived copy with a 200 status code and fraction for which Fable found aliases.

Table 8 also shows that Fable's success rate is lower for URLs which fail to resolve as compared to URLs which result in erroneous redirections. Moreover, Fable's ability to find aliases is not limited to URLs that stopped working recently; Table 9 shows that when we compare URLs which were last successfully archived 7–12 years ago to those that became dysfunctional more recently, Fable's success rate is comparable.

SimilarCT finds aliases for 4286 URLs, close to the number of aliases found by Fable. However, what matters is the number of *correct* aliases found. Taking into account Fable's precision of 90% and its true-positive rate of 79% on the ground truth dataset, we can estimate the total number of broken URLs in this dataset that do have an alias. Using SimilarCT's metrics from the ground truth dataset, we can then estimate that the number of correct aliases discovered via this approach is only 2786. Thus, compared to SimilarCT, Fable correctly unearths aliases for 51% more broken URLs.

Table 10 shows the primary factors limiting Fable's ability to find aliases for more broken links. First, the dominant bottleneck for search-based alias discovery is that none of the search results are considered as a match. This might occur because either the page is absent from the search indices of both Google and Bing, Fable's criteria for considering a search result as a match is too strict, or the page simply no longer exists. We are unable to distinguish

Search	No. URLs
No valid archived copy	5629
No search results	1541
No matching search result	8195

Historical redirection	No. URLs
No 3xx archived copy	7890
Erroneous 3xx archived copy	7475

Inference	No. URLs
Not enough examples to infer	12650
Pattern not possible to learn	2790
No good alias inferred	15

Table 10: Breakdown of reasons for Fable's inability to find aliases using different methods.

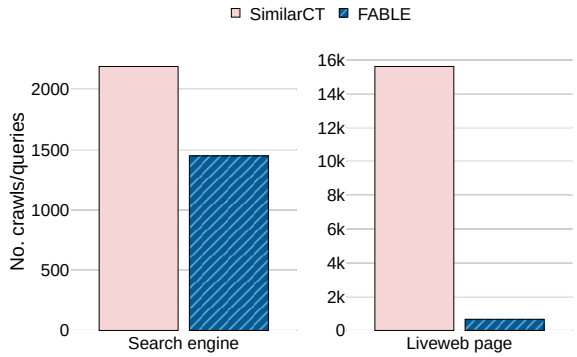


Figure 9: Efficiency of Fable backend. For 1000 broken URLs, in comparison to SimilarCT, Fable significantly reduces both the number of web search queries issued and pages crawled from the live web.

between these factors because we do not know which of the URLs in this dataset do have functional aliases. Second, the absence of 3xx status code archived copies and copies with erroneous redirections are both equally responsible for limiting the utility of historical redirections. Lastly, when Fable fails to infer the alias for an URL, this is either because it did not find any aliases for other similar URLs (suggesting that all of these pages no longer exist) or because the new URLs contain either partially predictable or unpredictable tokens (which prevents us from programmatically inferring them).

5.2 Efficiency

Next, we evaluate Fable's efficiency from two perspectives: the backend's throughput, and latency at the frontend. To ensure a fair comparison, we focus only on URLs for which SimilarCT has a chance of finding the alias. These are URLs for which 1) 200 status code archived copies exist, and 2) the alias identified by Fable appears in the search results. We then assume that SimilarCT matches each such broken URL to the same alias as the one found by Fable.

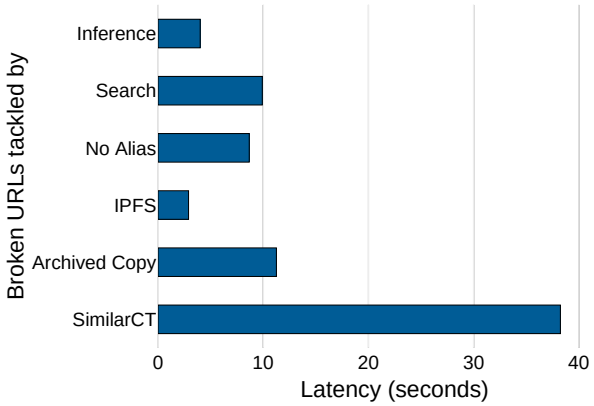


Figure 10: Median latency incurred at the Fable frontend as a function of whether or not it finds the alias and the method it uses when successful. We compare to the median latency for loading a page from either IPFS or the Wayback Machine, or for finding a broken link's alias using SimilarCT.

Backend throughput. As shown in Figure 9, compared to SimilarCT, Fable dramatically reduces the number of page crawls from the live web, by as much as 23x. Recall that Fable's backend needs to crawl the page at any of the search results only in the rare case when multiple URL transformation patterns appear credible. Fable also reduces the number of search engine queries by a third.

Frontend latency. Figure 10 shows that SimilarCT takes close to 40 seconds for the median URL. This is because of having to crawl pages linked from the search results one at a time until a match is potentially found. Since all the search results for a URL are from the same site, crawling pages at all of the top 10 results in parallel is not an option; doing so would typically exceed the rate at which a website permits its pages to be crawled.

In contrast, Fable's latency for the median URL is significantly lower, irrespective of whether it finds an alias for that URL or the method it uses to find the alias. When Fable finds an alias via inference, the median runtime is less than 5 seconds. Even when Fable fails to find any alias for a URL, it takes half as long as SimilarCT due to our policy of bailing on URLs in a directory once we determine that none of the URLs in that directory are likely to have aliases. In fact, Fable typically finds the alias with less delay than that involved in loading an archived copy from the Wayback Machine. As a result, in the common case, Fable's frontend would have identified a broken link's alias (if it exists) before the user finishes examining the archived copy of that link.

Systems that support content-based lookups offer lower latency than Fable, e.g., IPFS [66] reports median latencies of less than 3 seconds. But, the low coverage with this approach undermines its utility.

5.3 Utility of Aliases

In the last part of our evaluation, we demonstrate the utility of aliases discovered by Fable, as compared to relying on archived page copies.

		# URLs
Downsides for users	No archived copy	9
	Stale content	24
	Service not usable	70
Downsides for site providers	Loss of recommendations	60
	Loss of ad revenue	45
	Total	**93**

Table 11: Across 100 broken URLs for which Fable finds an alias, breakdown of reasons why reliance on web archives to serve requests for these URLs would be undesirable.

When a user visits a broken URL, if that page is still available at an alternative URL, directing the user to the new URL is beneficial in many ways as compared to serving an archived copy of the broken URL. To study how commonly these benefits apply, we pick 100 random URLs for which Fable found correct aliases. For each URL, we manually compare Wayback Machine's last successful archived copy for that URL to the page found today at the alias URL. Table 11 shows that, for almost all of these 100 broken URLs, reliance on archived copies would be suboptimal for users. Users will benefit from the aliases identified by Fable mainly in two ways.

- It is common for pages to offer functionality such as login, subscription, comments and feedback. For example, https://www.php.net/manual/en/function.setlocale.php, which is the alias of http://de3.php.net/manual/en/function.setlocale.php, enables users to leave notes. Such functionality cannot work on any archived copy.
- On many pages, since the main content (i.e., content left after ignoring boilerplate [50]) is updated over time, the last successful archived copy contains stale content. For example, the broken URL http://www.athleticsweekly.com/stats/records/world-records-and-best-performances-womens-indoor/ was pointing to a page that lists current world records in women's track and field. As time goes on, the records will change, as reflected at the alias https://athleticsweekly.com/stats/records/world-records-best-performances-womens-indoor/.

When a page still exists on the web, relying on an archived copy of the page is undesirable not only for users, but also for web providers. For example, when a link to a page is broken, the provider of that page might lose out on ad revenue and the ability to attract the user to their site by recommending other relevant pages. Table 11 shows that these downsides too commonly apply.

6 RELATED WORK

Detecting and studying broken URLs. Many tools and algorithms have been developed to detect web decay. Fable utilizes and improves upon Bar-Yossef et al.'s approach [30] to detect soft-404 pages. Parking sensors [67] identifies unused domains which only include automatically computed advertisements and links.

Prior work has also extensively studied the evolution of the web graph and web pages, both with respect to the persistence of links [30, 49, 52, 64] and stability of page content [32, 34, 49, 56]. Our observations are in line with these studies; we too find that a sizeable portion of URLs from the past are now invalid. These prior efforts have also observed that flux in page content is common, which is why it is often insufficient to serve archived copies in response to requests for broken URLs.

In contrast to all of this prior work, our contributions lie in systematically locating on the web the pages which were previously available at URLs which are now dysfunctional.

Rediscovering web pages. Many others have previously presented solutions to fix URLs that break when websites are reorganized [35, 45, 47, 48]. However, they largely rely on using a "lexical signature" extracted from a page's content as a robust hyperlink [59, 60], an approach which results in poor coverage, accuracy, and efficiency, as we have shown. Those who have observed, like us, the potential to leverage similarity across pages on a site in how their URLs are transformed when the site is reorganized [35] have not designed or evaluated a system which exploits this observation.

Duplicate page detection. Detection of a web page's duplicates has been widely researched [33, 42, 53, 65] in order to identify plagiarism and phishing, save storage by de-duplication, etc. Applying prior techniques from this line of work to find duplicates of archived copies is insufficient to find aliases for broken URLs because not all URLs are archived and archived copies can be stale.

Text-editing by example. Fable utilizes MS Excel's Flash Fill functionality [25], which is based on a string programming language [38]. Many others have used programming by example [40, 55, 58] to synthesize programs which automate text transformation given input-output example pairs. As far as we know, Fable is the first to utilize this functionality on URL transformations.

7 CONCLUSION

Given the prevalence of dynamic content and app-like functionality on the web, our work calls for a rethink of the common practice of coping with any dead link by using an archived copy. We demonstrated that a sizeable fraction of broken URLs on the web are a consequence of site reorganizations, and that it is indeed practical to automate the discovery of the new URLs for these pages. We believe the aliases found by Fable can help significantly improve the long-term availability of information and services on the web.

ACKNOWLEDGEMENTS

We thank Ayush Goel, the anonymous reviewers, and our shepherd, Zachary Bischof, for their valuable feedback. We also thank anonymous members of the Wikipedia community for their input on Fable. We are grateful to Aditya Chitta and Bryan Chehanske for their contributions in collecting data from Wikipedia. This work was supported in part by a grant from the Alfred P. Sloan Foundation and by credits for the Azure cloud platform awarded by Microsoft.

REFERENCES

[1] KDE 1.92 Release Announcement. https://web.archive.org/web/20060209082707/http://www.kde.org:80/announcements/announce-1.92.html.

[2] What If? (2008) #1 | Comic Books | Comics | Marvel.com. http://web.archive.org/web/20121017122005/http://marvel.com/comic_books/issue/22962/what_if_2008_1.

[3] After the Revolution: Youth, Democracy, and the Politics of Disappointment in Serbia - Jessica Greenberg. http://web.archive.org/web/20140701030455/http://sup.org/book.cgi?id=21682.

[4] Harvard Kennedy School - Mossavar-Rahmani Center for Business and Government :: About :: Fellows :: Senior Fellows: 2017-2018 (copy on July 12, 2017). https://web.archive.org/web/20170712144006/http://www.hks.harvard.edu/centers/mrcbg/about/fellows/currentsrfellows.

[5] Brave Browser and the Wayback Machine: Working together to help make the Web more useful and reliable. http://blog.archive.org/2020/02/25/brave-browser-and-the-wayback-machine-working-together-to-help-make-the-web-more-useful-and-reliable/.

[6] Cloudflare and the Wayback Machine, joining forces for a more reliable Web. https://blog.archive.org/2020/09/17/internet-archive-partners-with-cloudflare-to-help-make-the-web-more-useful-and-reliable/.

[7] 410 Gone - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/410.

[8] Alexa - Competitive Analysis, Marketing Mix, and Website Traffic. https://www.alexa.com/siteinfo.

[9] Canonical link element - Wikipedia. https://en.wikipedia.org/wiki/Canonical_link_element.

[10] Category:Articles with dead external links - Wikipedia. https://en.wikipedia.org/wiki/Category:Articles_with_dead_external_links.

[11] Category:Articles with permanently dead external links - Wikipedia. https://en.wikipedia.org/wiki/Category:Articles_with_permanently_dead_external_links.

[12] chromium/dom-distiller: Distills the DOM. https://github.com/chromium/dom-distiller.

[13] Internet Archive: Wayback Machine. https://archive.org/web/.

[14] InternetArchiveBot. https://meta.wikimedia.org/wiki/InternetArchiveBot.

[15] IPFS Powers the Distributed Web. https://ipfs.tech/.

[16] Klazify - Free Website Categorization & Logo API. Find company's category and logo from URL. https://www.klazify.com/.

[17] Medium Sitemap. https://medium.com/sitemap/sitemap.xml.

[18] Newspaper3k: Article scraping & curation — newspaper 0.0.2 documentation. https://newspaper.readthedocs.io/en/latest/.

[19] Perma.cc. https://perma.cc/.

[20] PROSE - Text Transformation - Microsoft Research. https://www.microsoft.com/en-us/research/project/prose-text-transformation/usage/.

[21] Public Suffix List. https://publicsuffix.org/.

[22] Robust Links - Make Your Link Robust. https://robustlinks.mementoweb.org/.

[23] Stack Exchange Data Dump : Stack Exchange, Inc. : Free Download, Borrow, and Streaming : Internet Archive. https://archive.org/details/stackexchange.

[24] User:FABLEBot/New URLs for permanently dead external links - Wikipedia. https://en.wikipedia.org/wiki/User:FABLEBot/New_URLs_for_permanently_dead_external_links.

[25] Using Flash Fill in Excel. https://support.microsoft.com/en-us/office/using-flash-fill-in-excel-3f9bcf1e-db93-4890-94a0-1578341f73f7.

[26] Web Archive, Available Online | Library of Congress. https://www.loc.gov/web-archives/.

[27] Wikipedia:Link rot - Wikipedia. https://en.wikipedia.org/wiki/Wikipedia:Link_rot#Internet_archives.

[28] Scott G Ainsworth, Ahmed Alsum, Hany SalahEldeen, Michele C Weigle, and Michael L Nelson. 2011. How much of the web is archived?. In *ACM/IEEE Joint Conference on Digital Libraries*.

[29] Ahmed AlSum, Michele C Weigle, Michael L Nelson, and Herbert Van de Sompel. 2014. Profiling web archive coverage for top-level domain and content language. *International Journal on Digital Libraries* (2014).

[30] Ziv Bar-Yossef, Andrei Z Broder, Ravi Kumar, and Andrew Tomkins. 2004. Sic transit gloria telae: Towards an understanding of the web's decay. In *WWW*.

[31] Andrei Broder. 2002. A taxonomy of web search. In *ACM SIGIR Forum*.

[32] Junghoo Cho and Hector Garcia-Molina. 1999. *The evolution of the web and implications for an incremental crawler.* Technical Report.

[33] Dennis Fetterly, Mark Manasse, and Marc Najork. 2003. On the evolution of clusters of near-duplicate web pages. In *IEEE/LEOS International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices*.

[34] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L Wiener. 2004. A large-scale study of the evolution of Web pages. *Software: Practice and Experience* 34, 2 (2004), 213–237.

[35] Ayush Goel, Jingyuan Zhu, and Harsha V. Madhyastha. 2022. Making Links on Your Web Pages Last Longer than You. In *HotNets*.

[36] Ayush Goel, Jingyuan Zhu, Ravi Netravali, and Harsha V. Madhyastha. 2022. Jawa: Web Archival in the Era of JavaScript. In *OSDI*.

[37] Turn all references blue. https://archive.org/details/mark-graham-presentation.

[38] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM SIGPLAN Notices* 46, 1 (2011), 317–330.

[39] Daniel Conrad Halbert. 1984. *Programming by example.* Ph. D. Dissertation. University of California, Berkeley.

[40] William R Harris and Sumit Gulwani. 2011. Spreadsheet table transformations from examples. *ACM SIGPLAN Notices* 46, 6 (2011), 317–328.

[41] Terry L Harrison and Michael L Nelson. 2006. Just-in-time recovery of missing web pages. In *ACM Conference on Hypertext and Hypermedia*.

[42] Monika Henzinger. 2006. Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *SIGIR*.

[43] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. 2017. Foofah: Transforming data by example. In *SIGMOD*.

[44] Shawn M Jones, Herbert Van de Sompel, Harihar Shankar, Martin Klein, Richard Tobin, and Claire Grover. 2016. Scholarly context adrift: Three out of four URI references lead to changed content. *PloS one* (2016).

[45] Martin Klein and Michael L Nelson. 2008. Revisiting lexical signatures to (re-)discover web pages. In *International Conference on Theory and Practice of Digital Libraries*. Springer, 371–382.

[46] Martin Klein and Michael L Nelson. 2010. Evaluating methods to rediscover missing web pages from the web infrastructure. In *ACM/IEEE Joint Conference on Digital Libraries*.

[47] Martin Klein, Jeffery Shipman, and Michael L Nelson. 2010. Is this a good title?. In *ACM Conference on Hypertext and Hypermedia*.

[48] Martin Klein, Jeb Ware, and Michael L Nelson. 2011. Rediscovering missing web pages using link neighborhood lexical signatures. In *ACM/IEEE Joint Conference on Digital Libraries*.

[49] Wallace Koehler. 2002. Web page change and persistence—A four-year longitudinal study. *Journal of the American society for information science and technology* 53, 2 (2002), 162–171.

[50] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. In *WSDM*.

[51] John Kunze and Richard Rodgers. 2008. The ARK identifier scheme. (2008).

[52] Steve Lawrence, Frans Coetzee, Eric Glover, Gary Flake, David Pennock, Bob Krovetz, Finn Nielsen, Andries Kruger, and Lee Giles. 2000. Persistence of information on the web: Analyzing citations contained in research articles. In *CIKM*.

[53] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *WWW*.

[54] John Markwell and David W Brooks. 2003. "Link rot" limits the usefulness of web-based educational materials in biochemistry and molecular biology. *Biochemistry and Molecular Biology Education* 31, 1 (2003), 69–72.

[55] Anders Miltner, Kathleen Fisher, Benjamin C Pierce, David Walker, and Steve Zdancewic. 2017. Synthesizing bijective lenses. In *POPL*.

[56] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. 2004. What's new on the Web? The evolution of the Web from a search engine perspective. In *WWW*.

[57] Anish Nyayachavadi, Jingyuan Zhu, and Harsha V Madhyastha. 2022. Characterizing "permanently dead" links on Wikipedia. In *IMC*.

[58] Peter-Michael Osera and Steve Zdancewic. 2015. Type-and-example-directed program synthesis. *ACM SIGPLAN Notices* 50, 6 (2015), 619–630.

[59] Seung-Taek Park, David M Pennock, C Lee Giles, and Robert Krovetz. 2004. Analysis of lexical signatures for improving information persistence on the World Wide Web. *ACM Transactions on Information Systems (TOIS)* 22, 4 (2004), 540–572.

[60] Thomas A Phelps and Robert Wilensky. 2000. *Robust hyperlinks cost just five words each.* University of California, Berkeley, Computer Science Division.

[61] Sarah Rhodes. 2010. Breaking down link rot: The Chesapeake project legal information archive's examination of URL stability. *Law Libr. J.* 102 (2010), 581.

[62] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[63] Carmine Sellitto. 2005. The impact of impermanent Web-located citations: A study of 123 scholarly conference publications. *Journal of the American Society for Information Science and Technology* 56, 7 (2005), 695–703.

[64] Diomidis Spinellis. 2003. The decay and failures of web references. *Commun. ACM* 46, 1 (2003), 71–77.

[65] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. 2008. SpotSigs: Robust and efficient near duplicate detection in large web collections. In *SIGIR*.

[66] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. 2022. Design and evaluation of IPFS: a storage layer for the decentralized web. In *SIGCOMM*.

[67] Thomas Vissers, Wouter Joosen, and Nick Nikiforakis. 2015. Parking sensors: Analyzing and detecting parked domains. In *NDSS*.

[68] Jonathan L Zittrain, John Bowers, and Clare Stanton. 2021. The Paper of Record Meets an Ephemeral Web: An Examination of Linkrot and Content Drift within The New York Times. *Available at SSRN 3833133* (2021).