Analysis of Website Consistency and Implications Regarding Domain Trustworthiness

---

A Senior Project in Computer Science for Seth Lifland
Yale University

Seth Lifland
April 28th, 2015
Faculty Advisor: Bryan Ford
Key Collaborator and Graduate Advisor: Daniel Jackowitz
Director of Undergraduate Studies in Computer Science: James Aspnes
Yale University

Abstract

We are all aware that some websites are more trustworthy than others, but website trustworthiness is a difficult concept to define. This paper examines website consistency as a potential metric for website trustworthiness. We define website consistency as the degree to which a website is formed from identical resources across multiple fetches of the website. There is more to website consistency than the pure visual appearance of a webpage; it is possible for a website to refer to the same resource with multiple URLs, thereby encoding extraneous identifying information in the URL itself. We define a perfectly consistent website as one such that every resource referred to by the top level URL is identical and maps to identical contents across all fetches of the site.

Our input data set is the Alexa Top 100 International sites list. We initially fetch each site several times using a scriptable browser and for each fetch, store a list of resources requested by the top-level site. A processing script then analyzes the results of all fetches to construct statistics about the consistency of each website. Additionally, in the case of slightly-varying resource URLs that map to the same contents the processing script attempts to reduce this set of "synonym" URLs to a single URL with only the parts that are common to all variations and determines whether fetching the "reduced" URL yields the same resource as any URL in the original "synonym set."

Our preliminary results demonstrate that website consistency varies widely across sites in our data set. Additionally, the success rate of reducing sets of synonym URLs to a single URL and acquiring a resource with the same contents varies widely. Future work is needed to adequately account for these variations, although we offer some conjectures as to their possible causes.

I. Motivation

Anyone who has used the Internet in the last 20 years is likely aware of the fact that some websites are more trustworthy than others. Examples of "untrustworthy" behavior range from tracking users through the use of cookies or fingerprinting to attempting to trick the visitor into installing malicious software locally on her machine. Trustworthiness in the case of websites is a difficult quality to define, and it is equally difficult to prove trustworthiness empirically. However, there are certain website behaviors that we can identify and compare across websites to make conjectures about which websites are more or less likely to be trustworthy.

This analysis serves as a useful first step towards measuring website trustworthiness by looking specifically at website consistency. We define website consistency as the degree to which a website is formed from the same parts across all fetches. Specifically, a top level domain generally requests a list of other resources each time it is fetched. For our input data set – the Alexa top 100 International sites list – we are interested in which and how many of these resources are consistent or somehow inconsistent across all fetches of a given website. Eventually, we hope to be able to use these data to form conclusions about the relative trustworthiness of various websites.

In addition to evaluating website trustworthiness, this analysis is also interested in evaluating methods of combatting untrustworthy behavior. One form of potentially untrustworthy behavior which the analysis is interested in combatting is user-tracking through information embedded into the URLs of resources themselves. For instance, a long query string in a URL might contain one element of the form "key=value" whose value has no effect on the resource retrieved, but which is being maintained only to identify the user. The analysis attempts to identify this behavior by looking for sets of "synonym" URLs in the lists of resources for all fetches of all sites. A group of URLs are considered synonyms if they vary slightly in structure but return resources whose contents hash to the same value. Based on the assumption that some of the information that varies between otherwise identical URLs which return identical resources could be used to track the user, this analysis attempts to strip out the unique parts of synonym URLs and determines whether the "reduced" URL is able to return the same resource as all of the synonym URLs. In cases where fetching the "reduced" URL successfully returns the same resource as any one of the original synonym URLs, we can feel secure in that by using the reduced URL, we are no longer cooperating with the website's attempts to track us as a user. Ultimately, we would ideally like to be able to take any URL that a website requests, identify elements that are likely extraneous and potentially used for tracking, and remove them before proceeding with the request.

II. Implementation

II.1 Organization

The end-to-end analysis system consists of several types of components. The detailed processing scripts are implemented in Python, the actual fetching of websites and writing of

results is done by a pair of JavaScript scripts using the SlimerJS scriptable browser, and the convenient processing of all the input sites and management of output files is handled by several bash shell scripts. An illustration of the system with all the source files named is provided below:

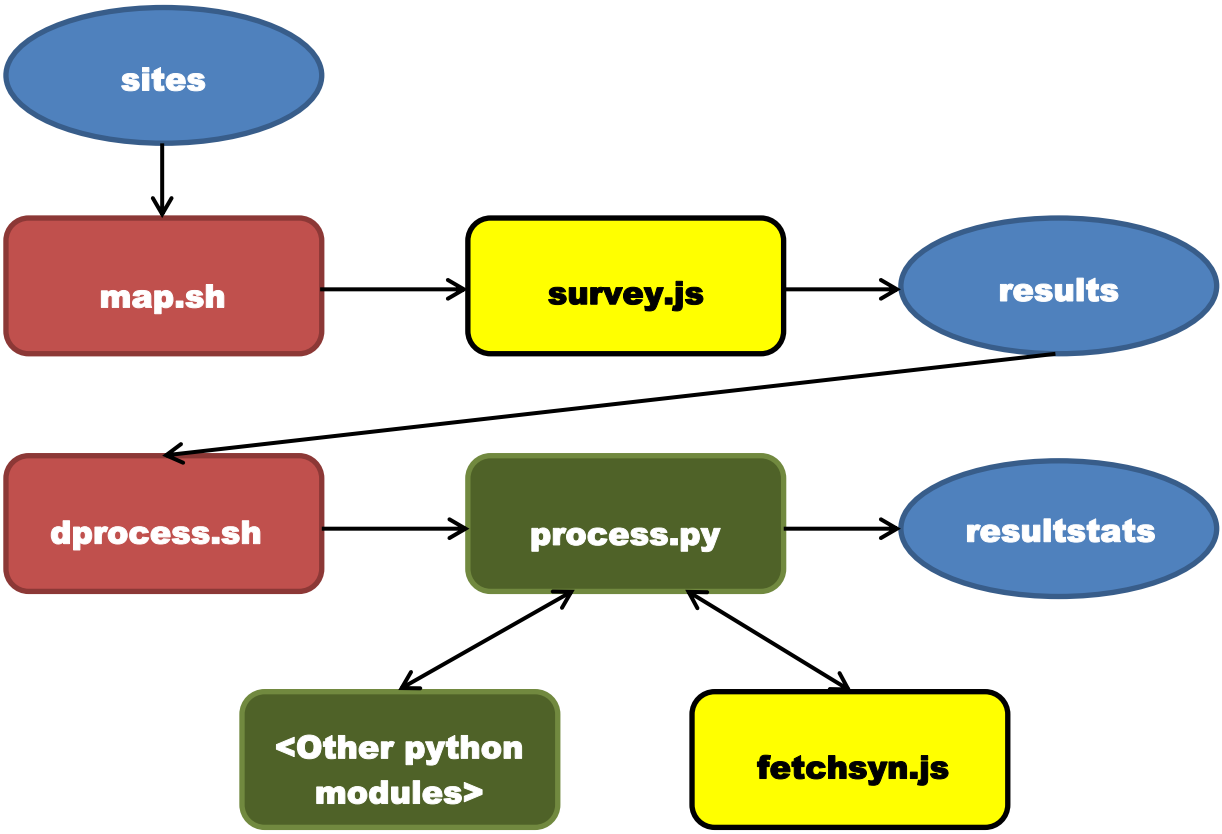


Figure 1: High-level organization of the website consistency analysis system

The following is a high-level description of each of these components.

- sites contains several possible lists of websites to evaluate, with just the top-level domain names printed one-per-line. For all of my experiments, I used the Alexa Top 100 International list.
- map.sh is responsible for taking every website in a given text file, feeding them to the survey.js script a given number of times, and storing the results in a strictly-organized directory that will be read later by dprocess.sh.
- survey.js uses SlimerJS to fetch a provided website and write the results to a provided file as a JSON object which can later be directly parsed by process.py.
- results contains the results of fetching all the sites a given number of times with map.sh. Within the results directory, map.sh creates a directory for domain name fetched, creates a numbered directory for each fetch, and writes the fetch result to a file called results.json.
- dprocess.sh (detailed-process) passes the results files for all fetches of all sites to process.py and ensures that the output is correctly stored in the resultstats directory.
- process.py is the core of the analysis program. It takes the list of results.json files corresponding to all fetches of a given site as input, computes various statistics on the resources present across all fetches, identifies “synonym URL sets,” computes and

attempts to fetch "reduced URLs" for each synonym set, and outputs results both on a per-site basis to a directory within resultstats and to a couple of aggregate results files, stored in resultstats/agg.

- The other python modules that process.py refers to provide functionality such as reducing synonym URL sets to reduced URLs, fetching reduced URLs, and sorting resource URLs into various potentially interesting data structures. Currently, the most important of these modules are urltable.py and synurl.py.
- fetchsyn.js is almost identical to survey.js in that it simply fetches a URL and writes the received resources to a location where they can be processed later. Fetchsyn, however, is invoked directly by process.py for fetching reduced URLs in order to determine whether they are valid substitutes for the synonym URLs from which they are derived.
- resultstats is the directory in which dprocess.sh stores the results of running process.py on all the top-level sites. There is a directory for each site inside resultstats containing a text file describing the detailed results of executing process.py on that site. Additionally, the results of fetching the reduced URLs calculated for a site are stored in the site's corresponding directory within resultstats so that they don't have to be refetched by default every time the processing script is run.[1] Resultstats also contains a directory "agg" (aggregate data) which contains text files and csv files that contain concise data from across processing all sites.

II.2 Processing Details

This section will describe in detail how process.py processes a single site. Process.py is run once per site and its input is the list of "results.json" files corresponding to each fetch of the site. These results files are directly interpreted as Python dictionaries. The analysis is most interested in the list of resources associated with each fetch, where a resource is represented by the URL of the resource received, the hash of its contents, and its size. It is on these resource lists that the vast majority of the processing acts. In order to facilitate efficient categorization of the resources, Process.py maintains several data structures as it processes the results.json file for each fetch of a site:

- a dictionary mapping a URL to the number of times it occurs across all fetches
- a dictionary mapping a URL to all possible hashes with which it is associated
- a dictionary mapping a hash to all possible URLs with which it is associated

Given these data structures, process can place each URL into one of the following categories:

- Consistent – same URL, same contents in all successful fetches
- Content-Inconsistent – same URL, contents not same in all fetches
- Synonym – different URL, same contents for all appearances
- Totally Inconsistent – URL and contents vary in all fetches
- Failed – URL was not fetched successfully

[1] It is possible to force dprocess.sh to refetch all reduced URLs for all sites by invoking it with the flag "-refetch"

The categorization algorithm is implemented inside Process.py in the function "categorize_ resources_by_fetch" and works like this:

```
For each resource list:
	For each resource in the resource list:
		If the size is 0:
			resource -> failed resources
		Else if resource hash is associated with multiple URLs:
			resource -> synonym resources
		Else if resource URL is associated with multiple hashes:
			resource -> Content-inconsistent resources
		Else:
			If resource fetched appears in resource lists for all successful
			trials:
				resource -> Consistent resources
			Else:
				resource -> Inconsistent resources
```

This algorithm is imperfect, but it maintains the invariant that no single resource URL is sorted into more than one category for a given fetch, although there can be some overlap between the categories. For instance, if there is some hash that multiple resource URLs return, these URLs could both appear consistently in the resource lists of all fetches, but they will still be identified as synonyms for the same hash, and therefore be categorized as such, despite also being consistent. One of the greatest difficulties of this analysis was conceiving of mutually exclusive categories for the different resource URLs that take into account their structure, the hashes of their contents, and their success status.

One improvement to the analysis that I have not had time to implement is to attempt to associate each URL within a resource list with the *single most similar* URL in every other fetch. This would avoid the above situation, because the two URLs from the same fetch which map to the same contents would be placed into separate sets with only identical URLs as seen across all fetches, and therefore would be categorized as consistent, whereas a "true" synonym URL would correspond to a different set where all the URLs from different fetches are slightly different, but whose contents hash to the same value.

Once the URLs are sorted into these sets, the only one which we process further is the synonym URL set. The additional processing involves taking each set of synonym URLs and attempting to reduce it into a single URL where all the unique parts have been stripped out. For instance, often a set of synonym URLs will each contain a long query string with elements of the form "key=value" in which all values are identical except for one or two. In this case, the output of the reduction algorithm would be a URL whose query string has simply omitted the "key=value" segments for which the values differed.

Recall that the analysis is interested in evaluating the effectiveness of stripping the unique parts from synonym URLs down and fetching the reduced versions as a potential method of inhibiting website tracking through extraneous URL parameters. Therefore at this point Process.py attempts to fetch these reduced URLs. However, it first attempts to fetch one of the original URLs from the "synonym set" as a "sanity check." We expect that fetching one of the original URLs with all the unique parameters left in should produce contents which hash to the

same value as the original hash it was associated with. If the sanity check succeeds, Process.py attempts to fetch the reduced URL. It then compares the original hash to the hashes of all the resources retrieved by fetching the reduced URL, and if it finds a match, the reduced URL is deemed a successful substitute for any synonym URL in the set.

Process.py handles fetching in the case of the sanity check and reduced URLs by initiating a subordinate process and passing the URL to fetchsyn.js which attempts to fetch the URL, and writes the results of the fetch to another .json file which the Process.py script can read from just as it did with the original results.json files. By default, when dprocess.sh is invoked, it will attempt to directly read the results of past reduced URL fetches from their expected locations in the file system because the fetch of all the sanity URLs and reduced URLs is extremely time-consuming and is not yet parallelized in any way.

There are four possible results when fetching a reduced URL:

- The fetch fails completely
- The sanity check fails, and therefore the reduced URL(s) associated with that synonym set is never tested
- The fetch succeeds, but no match for the original contents is found
- The fetch succeeds, and matching contents are retrieved

II.3 The Synonym URL Reduction Algorithm

The synonym URL reduction algorithm is probably the most complicated process in my code. At a high level, it involves splitting each URL in the synonym set into segments and constructing a reduced URL based on the "intersection" of the URLs. In my current algorithm, an "intersection" URL is calculated by incorporating each URL in the synonym set into the intersection URL incrementally. Each URL is first split into segments, where each segment is represented by a 3-tuple of the form (segment number, segment text, segment type) where possible values of segment type are scheme, network location, path, parameters, query, and fragment. These types are based on Python's built-in urlparse library, which takes a url specified as a string and splits it into a 6-tuple with these components. The following is an illustration of a generic URL with each segment type illustrated:

scheme	network loc.	path	params	query	fragment
https://	example.com	/a/b/foo_dir	;x=3;id=24	?bar=a&y=52	#cat_pictures

The algorithm further splits each URL component by the corresponding separator into a list of segments. For example, in the above URL, the "path" component would ultimately become the list [(3, "a", PATH_TY), (4, "b", PATH_TY), (5, "foo_dir", PATH_TY)]. Including the segment numbers in the tuples was more necessary for another data structure which used the split_url function defined in urltable.py because in that data structure the segments became dictionary keys corresponding to lists of possible segment text values for segments that had different possible values, but I needed a way to be able to reconstruct the original order of segments.

The input to the reduce_url function is a set of synonym URLs represented by these lists of tuples. The algorithm proceeds in the following way:

```
intersection_url <- synonym_urls[0]
for each url in synonym_urls[1:]:
**     check that url and intersection url are same length and similar in structure
       for each corresponding segment in both URLs
              if segment types and segment texts equal
                     continue
              else
                     intersection_url[seg][seg_text] <- “”
```

In this way, after processing all URLs, the intersection URL reflects only the segments that are identical to all URLs in the set. The new intersection URL is a list of segments where some have a text value equal to “”. The reduced URL is constructed with a standard reconstruct_url function in urltable.py that undoes the work of splitting the URL into segments, removes the empty segments, and assembles the result into a single reduced URL string.

The line marked with ** requires a bit of explanation. As it turns out, while synonym URLs in a set are often nearly identical, sometimes they can be different lengths or fundamentally differ in structure. Because my current algorithm depends on comparing synonym URLs that are of the same length and close to identical, I have to treat synonym URLs that don’t “fit the same mold” separately. In actuality, the algorithm I use is slightly more complicated; a synonym set is actually reduced to a list of synonym URLs. Therefore if a URL does not pass the check in the ** line, it becomes a template for an additional intersection URL. Whenever a new URL is processed, it is first checked against all existing templates, and if it cannot be incorporated into any of them, it becomes its own template. The results of the analysis also examine the ratio of final “reduced” URLs to original synonym URL sets. In practice, the ratio averages out to roughly 1.3:1. The similarity check itself is described in detail in the next section.

A different algorithm that I suspect might work better than this but have not had the chance to implement would involve looking segment-by-segment rather than URL by URL; that is, looking at the first segment of all URLs in the set at once and either picking the most popular segment if there’s a plurality or majority vote for one segment value, or simply omitting the segment or choosing at random if it differs across too many of the URLs in the set. However, this algorithm would have greater difficulty handling different structural templates within a synonym URL set. If two URLs in a synonym set adhere to different structural templates, their segments shouldn’t be directly compared for the same “segment slot” in a resulting URL. A similarity check would still be necessary to isolate the URLs into different sets based on structural templates, but after that the reduced URL could be constructed segment by segment. It is also possible that the order of certain segments can vary arbitrarily; for instance two “key=value” segments in a query string might be flip-flopped, and we don’t want this to affect the intersection URL. This segment-based URL set reduction could also allow us to be flexible with the order of the segments in certain parts of the URL, such as the query string, where the order is fairly arbitrary.

II.4 The Similarity Check

The similarity check is possibly the most flawed portion of the analysis as it stands. One invariant we would expect from a good similarity check is that given 3 URLs, we can do a similarity check on any pair, compute the intersection of the pair, and check the similarity of the intersection and the third URL, and get consistent results as if we had done the initial similarity checks and intersection on a different pair. Currently, I'm not sure that the combination of my similarity check and "intersection" method guarantees that. However, for the purpose of ensuring that only synonym URLs which are the same length and relatively similar in structure are intersected into the same reduced URL, my algorithm has proven sufficient.

The basic question the similarity check answers is: "Is the ratio of matching segments to total segments between two URLs greater than some threshold?" Currently, a simplifying assumption that I make is that different-length URLs cannot be similar. This is an unfortunate assumption, since it is often untrue in practice where one URL might simply have an extra query segment but be otherwise identical, for instance. I began trying to implement a length-independent, order-independent similarity checking algorithm within simurl.py, but ran out of time to get it working well.

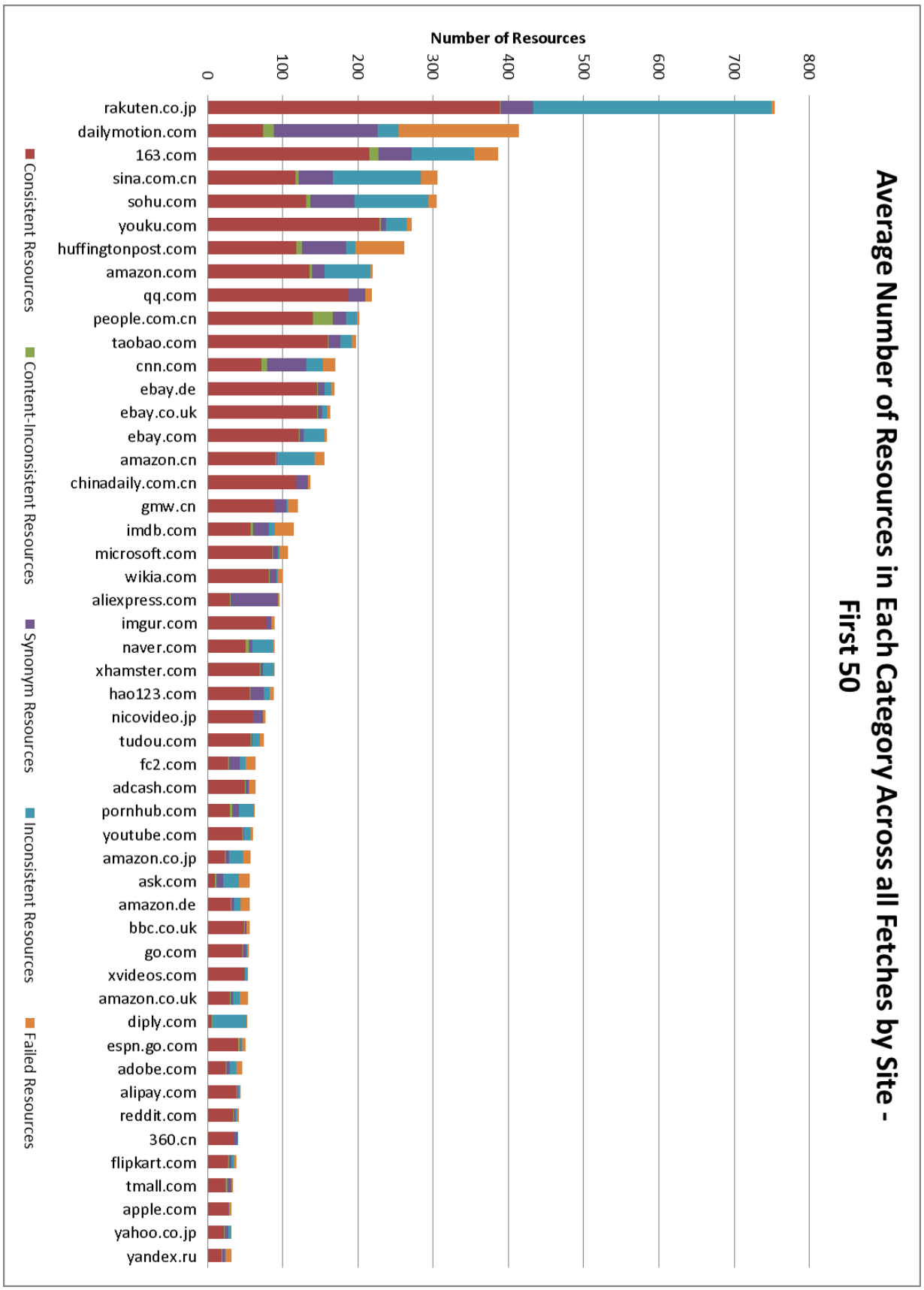
The current similarity check takes two URLs and a similarity threshold. It then computes a max score by counting the segments, each segment weighted by a scalar corresponding to the segment type.[2] Given the max score, it compares corresponding segments of the two URLs sequentially. If the segment text and type matches between the two, the similarity score is incremented by the weight corresponding to that segment type. After comparing all segments, if the ratio of the similarity score to the max score is above the given threshold, the check passes and the two URLs are considered similar. This algorithm has obvious drawbacks, including the fact that a single "extra" segment in the middle of one URL in two otherwise identical URLs will cause the similarity check to fail spectacularly. This is one of the reasons the similarity check automatically rules different-length URLs dissimilar. A segment-order-independent similarity check is needed to accommodate cases like that.

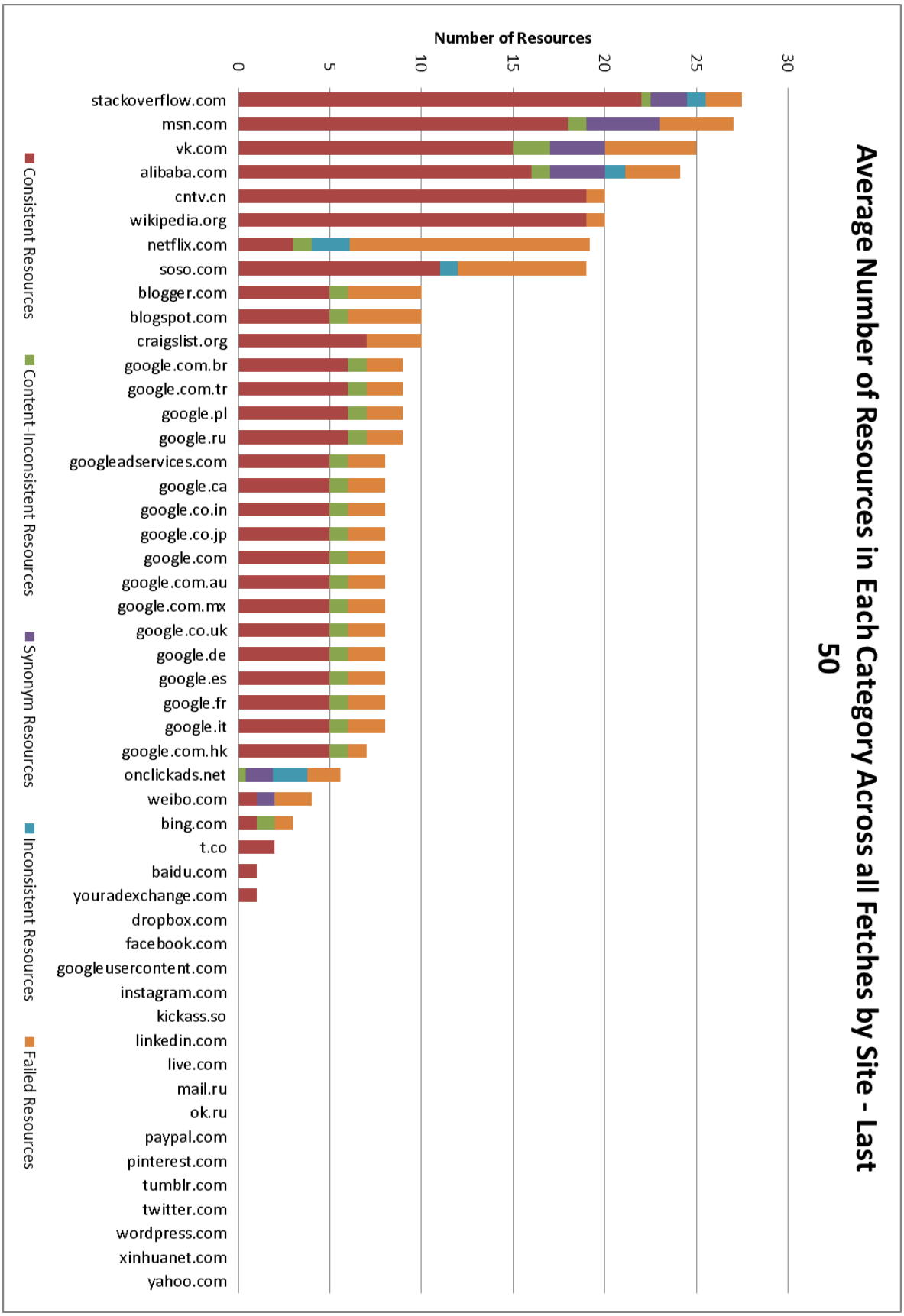
III. Results and Discussion

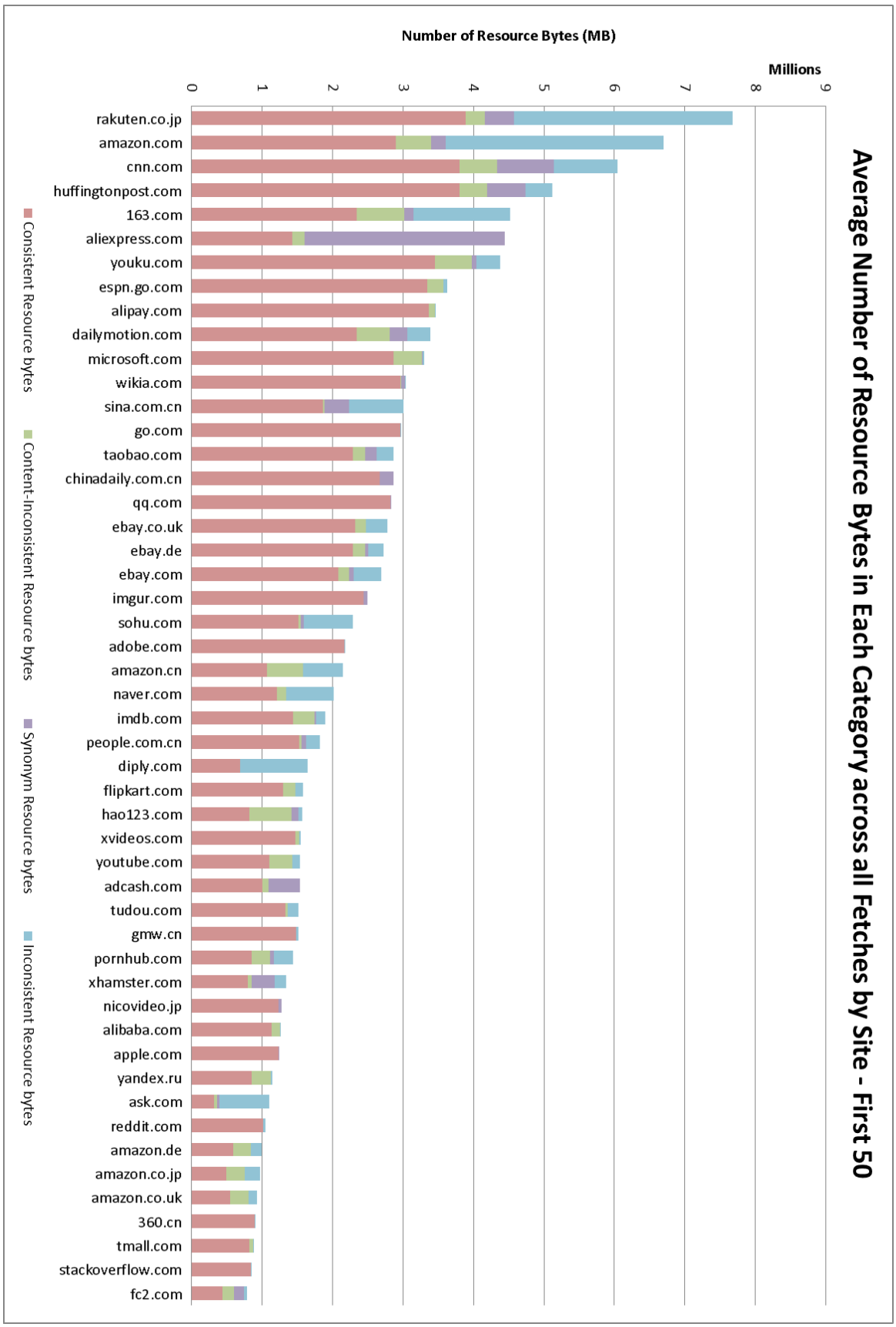
III.1 Website Resource Categorization Results

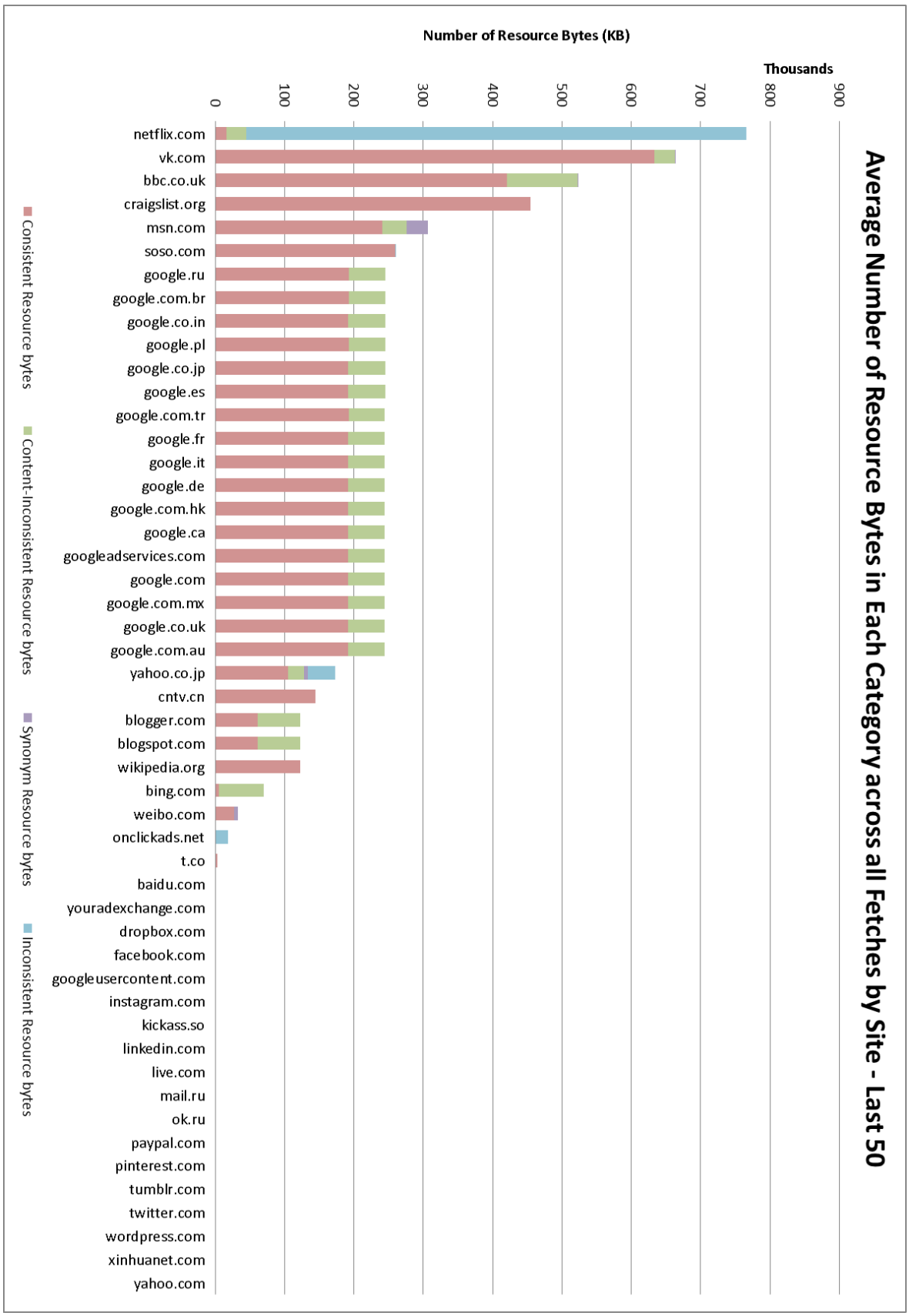
The following graphs present the initial results of the analysis on the Alexa Top 100 International sites list. The first set of graphs will show the results of categorizing the resources associated with each fetch for every site and then averaging the number of resources in each category across all fetches by site. The second set of graphs will show similar data, but in units of total resource *bytes* per category rather than simply number of resources. Because the data set is so large, I split these two graphs in half. The data are sorted from highest to lowest, so it is important to keep in mind that in the byte-weighted graphs, the y-axis scales differ by an order of magnitude. Also notice that the failed resources aren't represented in the byte-weighted graphs because my metric for classifying a resource as failed was that it had a size of 0.

[2] Currently, the network location is weighted twice as heavily as every other segment, under the assumption that network location matters more than potentially much longer strings of parameters or query segments


Average Number of Resources in Each Category Across all Fetches by Site - First 50
Number of Resources
0
100
200
300
400
500
600
700
800
rakuten.co.jp
dailymotion.com
163.com
sina.com.cn
sohu.com
youku.com
huffingtonpost.com
amazon.com
qq.com
people.com.cn
taobao.com
cnn.com
ebay.de
ebay.co.uk
ebay.com
amazon.cn
chinadaily.com.cn
gmw.cn
imdb.com
microsoft.com
wikia.com
aliexpress.com
imgur.com
naver.com
xhamster.com
hao123.com
nicovideo.jp
tudou.com
fc2.com
adcash.com
pornhub.com
youtube.com
amazon.co.jp
ask.com
amazon.de
bbc.co.uk
go.com
xvideos.com
amazon.co.uk
diply.com
espn.go.com
adobe.com
alipay.com
reddit.com
360.cn
flipkart.com
tmall.com
apple.com
yahoo.co.jp
yandex.ru
Consistent Resources
Content-Inconsistent Resources
Synonym Resources
Inconsistent Resources
Failed Resources


Average Number of Resources in Each Category Across all Fetches by Site - Last 50
Number of Resources
0
5
10
15
20
25
30
stackoverflow.com
msn.com
vk.com
alibaba.com
cntv.cn
wikipedia.org
netflix.com
soso.com
blogger.com
blogspot.com
craigslist.org
google.com.br
google.com.tr
google.pl
google.ru
googleadservices.com
google.ca
google.co.in
google.co.jp
google.com
google.com.au
google.com.mx
google.co.uk
google.de
google.es
google.fr
google.it
google.com.hk
onclickads.net
weibo.com
bing.com
t.co
baidu.com
youradexchange.com
dropbox.com
facebook.com
googleusercontent.com
instagram.com
kickass.so
linkedin.com
live.com
mail.ru
ok.ru
paypal.com
pinterest.com
tumblr.com
twitter.com
wordpress.com
xinhuanet.com
yahoo.com
Consistent Resources
Content-Inconsistent Resources
Synonym Resources
Inconsistent Resources
Failed Resources

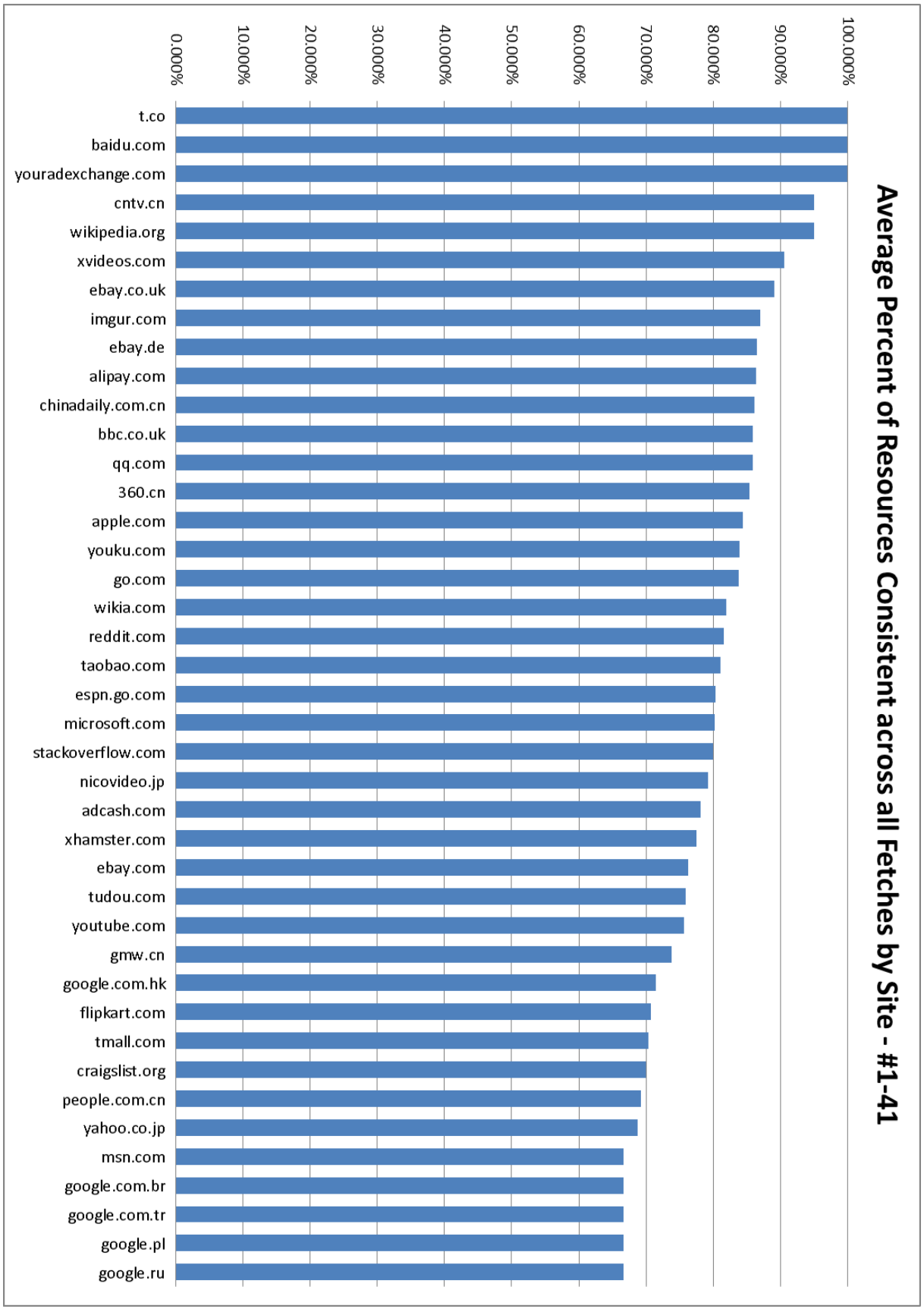

Average Number of Resource Bytes in Each Category across all Fetches by Site - First 50
Number of Resource Bytes (MB)
Millions
0
1
2
3
4
5
6
7
8
9
rakuten.co.jp
amazon.com
cnn.com
huffingtonpost.com
163.com
aliexpress.com
youku.com
espn.go.com
alipay.com
dailymotion.com
microsoft.com
wikia.com
sina.com.cn
go.com
taobao.com
chinadaily.com.cn
qq.com
ebay.co.uk
ebay.de
ebay.com
imgur.com
sohu.com
adobe.com
amazon.cn
naver.com
imdb.com
people.com.cn
diply.com
flipkart.com
hao123.com
xvideos.com
youtube.com
adcash.com
tudou.com
gmw.cn
pornhub.com
xhamster.com
nicovideo.jp
alibaba.com
apple.com
yandex.ru
ask.com
reddit.com
amazon.de
amazon.co.jp
amazon.co.uk
360.cn
tmall.com
stackoverflow.com
fc2.com
Consistent Resource bytes
Content-Inconsistent Resource bytes
Synonym Resource bytes
Inconsistent Resource bytes

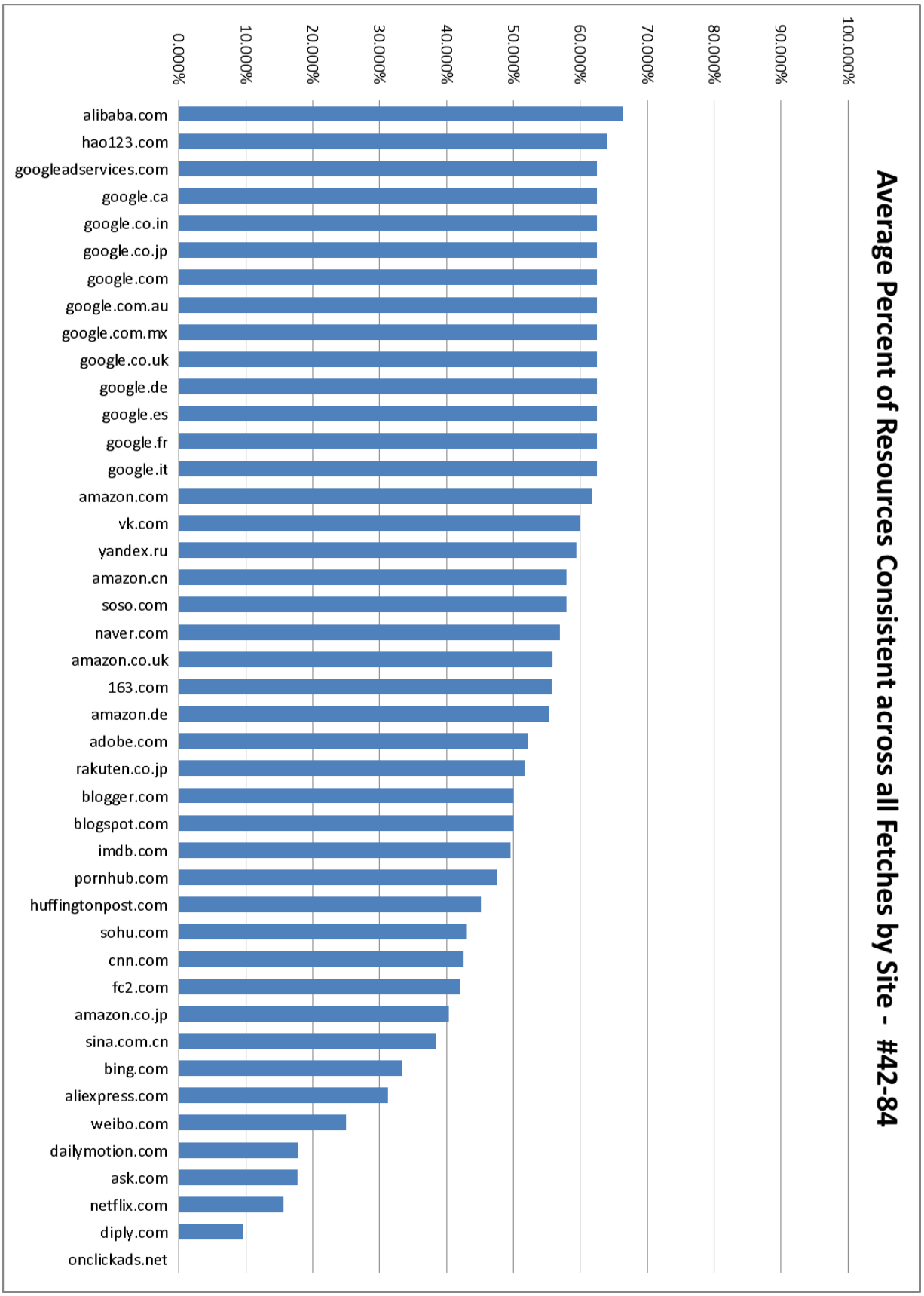

Average Number of Resource Bytes in Each Category across all Fetches by Site - Last 50
Number of Resource Bytes (KB)
Thousands
0
100
200
300
400
500
600
700
800
900
netflix.com
vk.com
bbc.co.uk
craigslist.org
msn.com
soso.com
google.ru
google.com.br
google.co.in
google.pl
google.co.jp
google.es
google.com.tr
google.fr
google.it
google.de
google.com.hk
google.ca
googleadservices.com
google.com
google.com.mx
google.co.uk
google.com.au
yahoo.co.jp
cntv.cn
blogger.com
blogspot.com
wikipedia.org
bing.com
weibo.com
onclickads.net
t.co
baidu.com
youradexchange.com
dropbox.com
facebook.com
googleusercontent.com
instagram.com
kickass.so
linkedin.com
live.com
mail.ru
ok.ru
paypal.com
pinterest.com
tumblr.com
twitter.com
wordpress.com
xinhuanet.com
yahoo.com
Consistent Resource bytes
Content-Inconsistent Resource bytes
Synonym Resource bytes
Inconsistent Resource bytes

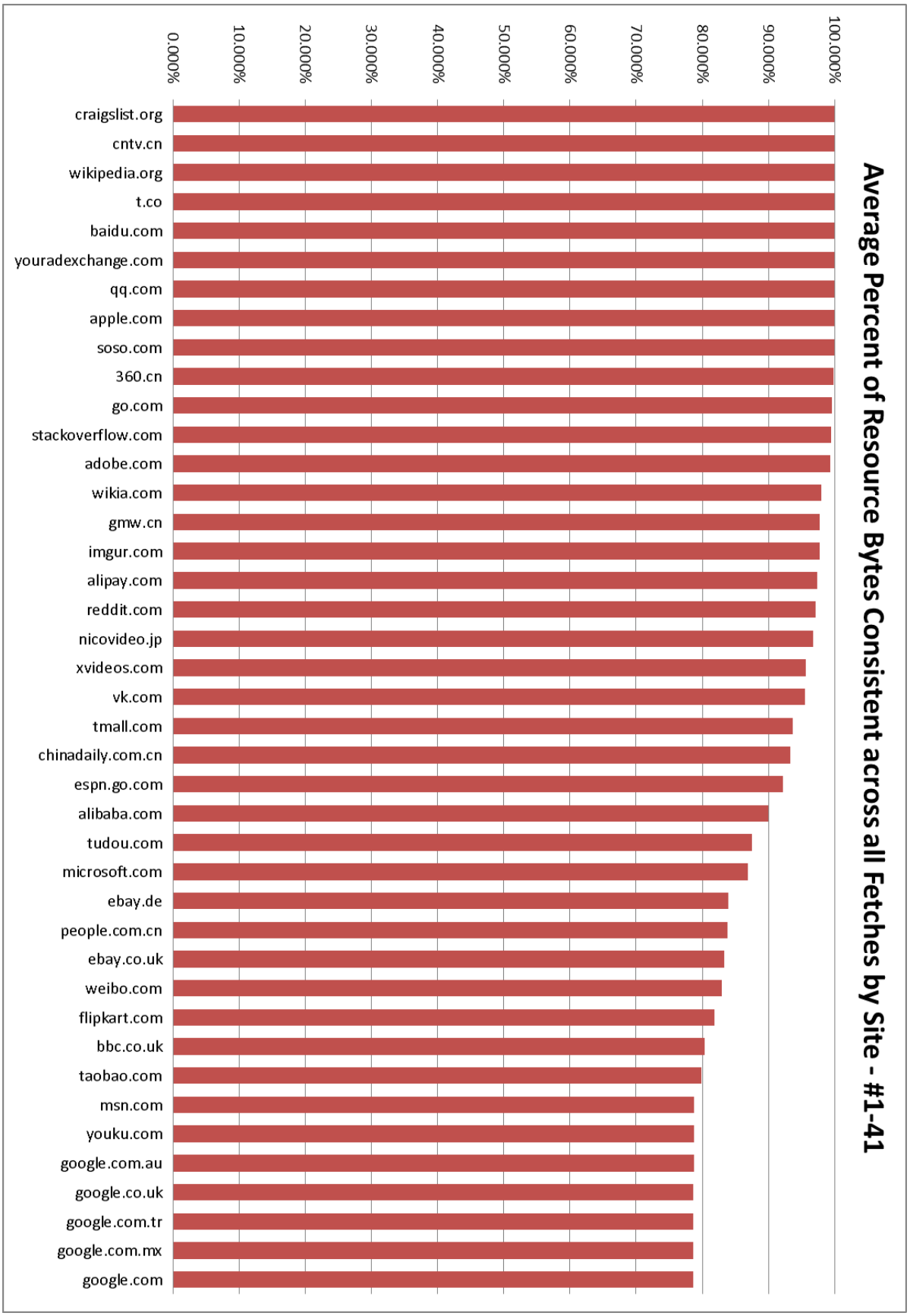
Even these initial data allow us to make some observations that might suggest something about the relative trustworthiness of different domains. However these data cannot stand on their own. One bit of additional analysis which could potentially bring some reason to these data would be to compare different domains within broad categories such as shopping, news, entertainment, social networking, games, adult, arts, business, reference, and so on. The Alexa website offers 17 categories, but it is likely that this kind of categorical organization would be more meaningful with a larger sample size of websites.

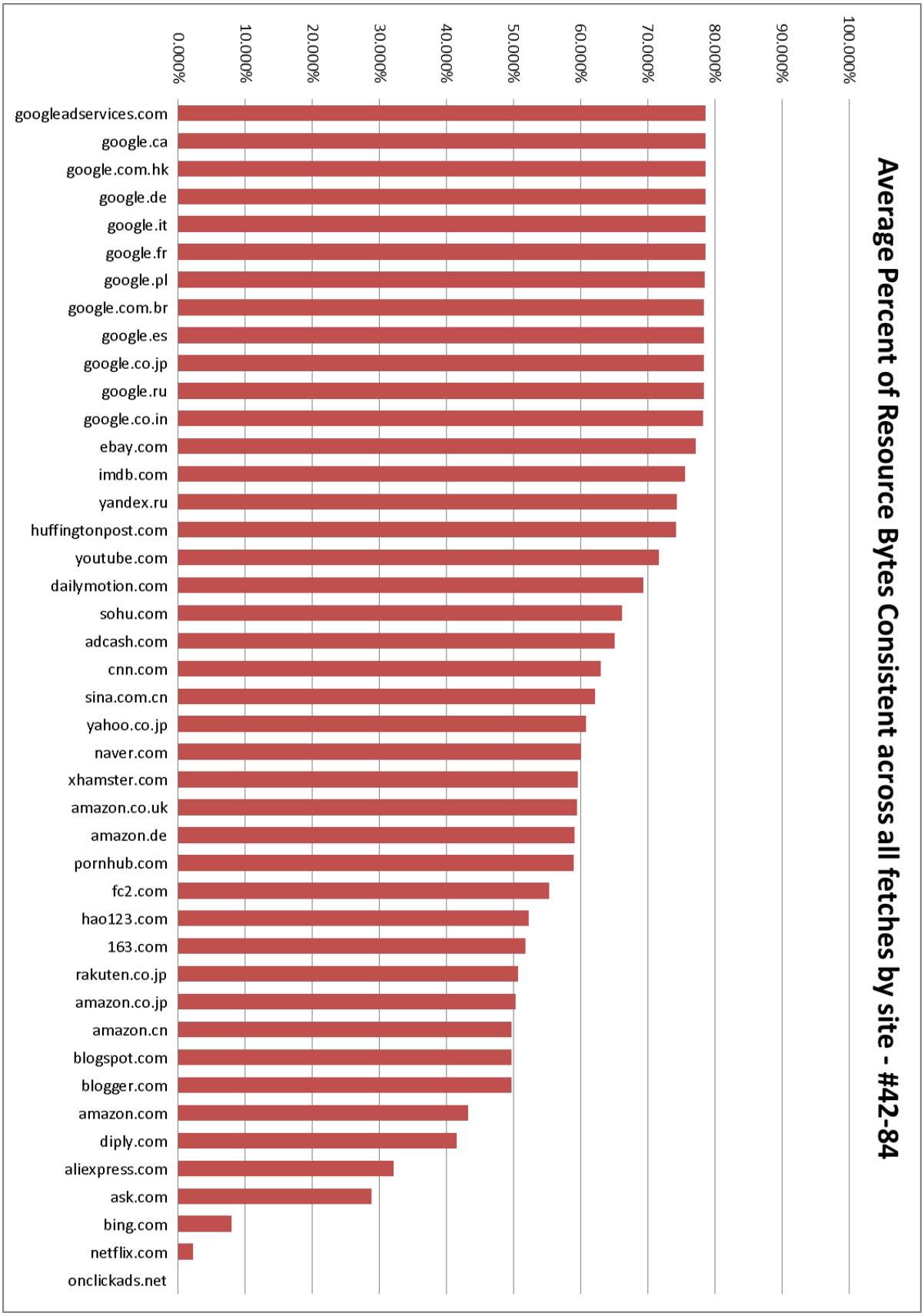
Notice at the low end of both graph pairs there are roughly 20 sites which report no resources fetched and no resource bytes. In the cases of these sites, the initial map.sh script and survey.js Slimer script were unable to successfully fetch the contents of the pages even once. Looking at these 20 sites, it appears that a large portion of them are social networking sites (Twitter, Tumblr, Instragram, Facebook, LinkedIn) or sites that otherwise require a log in (Yahoo, Paypal, Live). My hypothesis is that these sites may be especially conscious of meddling by bots, and therefore have protections in place to prevent utilities like SlimerJS from fetching and manipulating contents of pages within their domains. These sites which the analysis was unable to fetch are not included in the next set of graphs.

The following set of graphs present the average percent of consistent resources out of the total by site and then the average percent of consistent bytes by site.


Average Percent of Resources Consistent across all Fetches by Site - #1-41
0.000%
10.000%
20.000%
30.000%
40.000%
50.000%
60.000%
70.000%
80.000%
90.000%
100.000%
t.co
baidu.com
youradexchange.com
cntv.cn
wikipedia.org
xvideos.com
ebay.co.uk
imgur.com
ebay.de
alipay.com
chinadaily.com.cn
bbc.co.uk
qq.com
360.cn
apple.com
youku.com
go.com
wikia.com
reddit.com
taobao.com
espn.go.com
microsoft.com
stackoverflow.com
nicovideo.jp
adcash.com
xhamster.com
ebay.com
tudou.com
youtube.com
gmw.cn
google.com.hk
flipkart.com
tmall.com
craigslist.org
people.com.cn
yahoo.co.jp
msn.com
google.com.br
google.com.tr
google.pl
google.ru


Average Percent of Resources Consistent across all Fetches by Site - #42-84
0.000%
10.000%
20.000%
30.000%
40.000%
50.000%
60.000%
70.000%
80.000%
90.000%
100.000%
alibaba.com
hao123.com
googleadservices.com
google.ca
google.co.in
google.co.jp
google.com
google.com.au
google.com.mx
google.co.uk
google.de
google.es
google.fr
google.it
amazon.com
vk.com
yandex.ru
amazon.cn
soso.com
naver.com
amazon.co.uk
163.com
amazon.de
adobe.com
rakuten.co.jp
blogger.com
blogspot.com
imdb.com
pornhub.com
huffingtonpost.com
sohu.com
cnn.com
fc2.com
amazon.co.jp
sina.com.cn
bing.com
aliexpress.com
weibo.com
dailymotion.com
ask.com
netflix.com
diply.com
onclickads.net


Average Percent of Resource Bytes Consistent across all Fetches by Site - #1-41
0.000%
10.000%
20.000%
30.000%
40.000%
50.000%
60.000%
70.000%
80.000%
90.000%
100.000%
craigslist.org
cntv.cn
wikipedia.org
t.co
baidu.com
youradexchange.com
qq.com
apple.com
soso.com
360.cn
go.com
stackoverflow.com
adobe.com
wikia.com
gmw.cn
imgur.com
alipay.com
reddit.com
nicovideo.jp
xvideos.com
vk.com
tmall.com
chinadaily.com.cn
espn.go.com
alibaba.com
tudou.com
microsoft.com
ebay.de
people.com.cn
ebay.co.uk
weibo.com
flipkart.com
bbc.co.uk
taobao.com
msn.com
youku.com
google.com.au
google.co.uk
google.com.tr
google.com.mx
google.com


Average Percent of Resource Bytes Consistent across all fetches by site - #42-84
0.000%
10.000%
20.000%
30.000%
40.000%
50.000%
60.000%
70.000%
80.000%
90.000%
100.000%
googleadservices.com
google.ca
google.com.hk
google.de
google.it
google.fr
google.pl
google.com.br
google.es
google.co.jp
google.ru
google.co.in
ebay.com
imdb.com
yandex.ru
huffingtonpost.com
youtube.com
dailymotion.com
sohu.com
adcash.com
cnn.com
sina.com.cn
yahoo.co.jp
naver.com
xhamster.com
amazon.co.uk
amazon.de
pornhub.com
fc2.com
hao123.com
163.com
rakuten.co.jp
amazon.co.jp
amazon.cn
blogspot.com
blogger.com
amazon.com
diply.com
aliexpress.com
ask.com
bing.com
netflix.com
onclickads.net

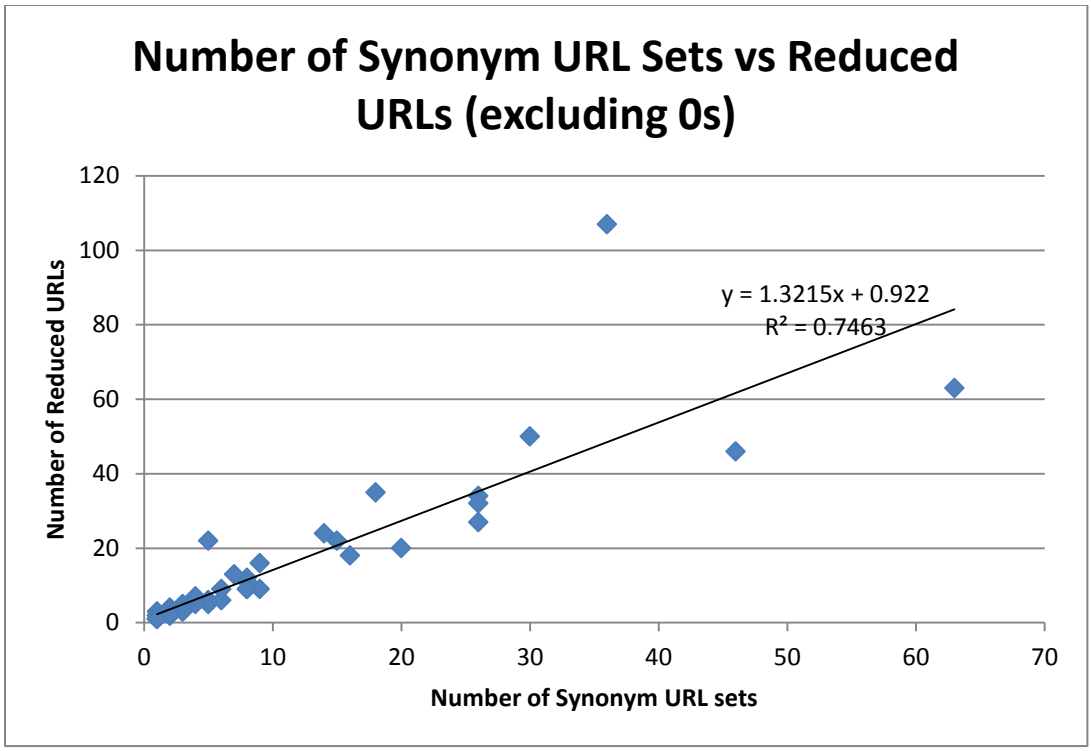
Unlike in the first sets of graphs, in these graphs I was able to keep the axes constant so that the graphs are less misleading when viewed together. I should point out that onclickads.net is not included in these data by accident; it was successfully fetched and it did in fact have no consistent resources across all fetches.

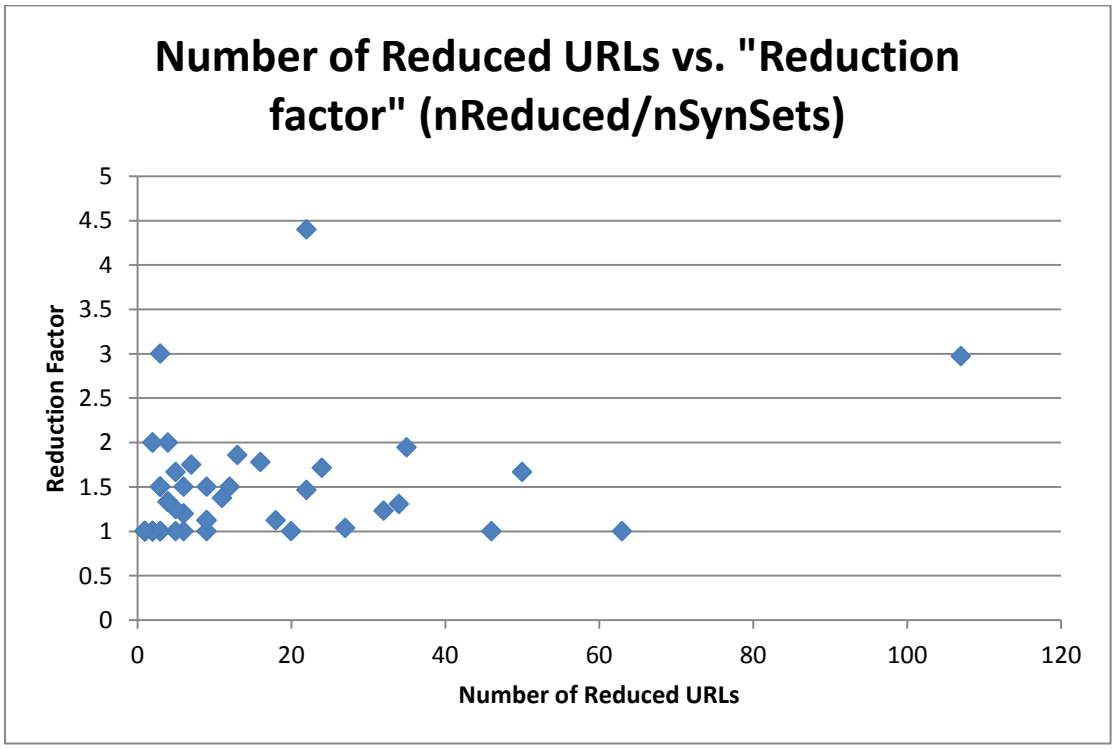
One thing that is immediately obvious is that there are more sites with a close to 100% resource consistency rate by byte than there are by number. Additionally, there are several more sites that have an 100% consistency rate by byte than there are which have an 100% consistency rate by number. This is because failed resources are counted in consistency by number of resource URLs but do not factor into number of resource bytes, because failed resources are stored as having a size of 0. It is also important to note that some of the domains at or near the top of the % consistency charts are very low in the raw number of resources charts. Being 100% consistent seems somewhat less meaningful in the case of sites like Baidu.com, whose total resource bytes came in at a puny 81 bytes (which is surprising, and could indicate that the fetch script did not perform as intended in this case).

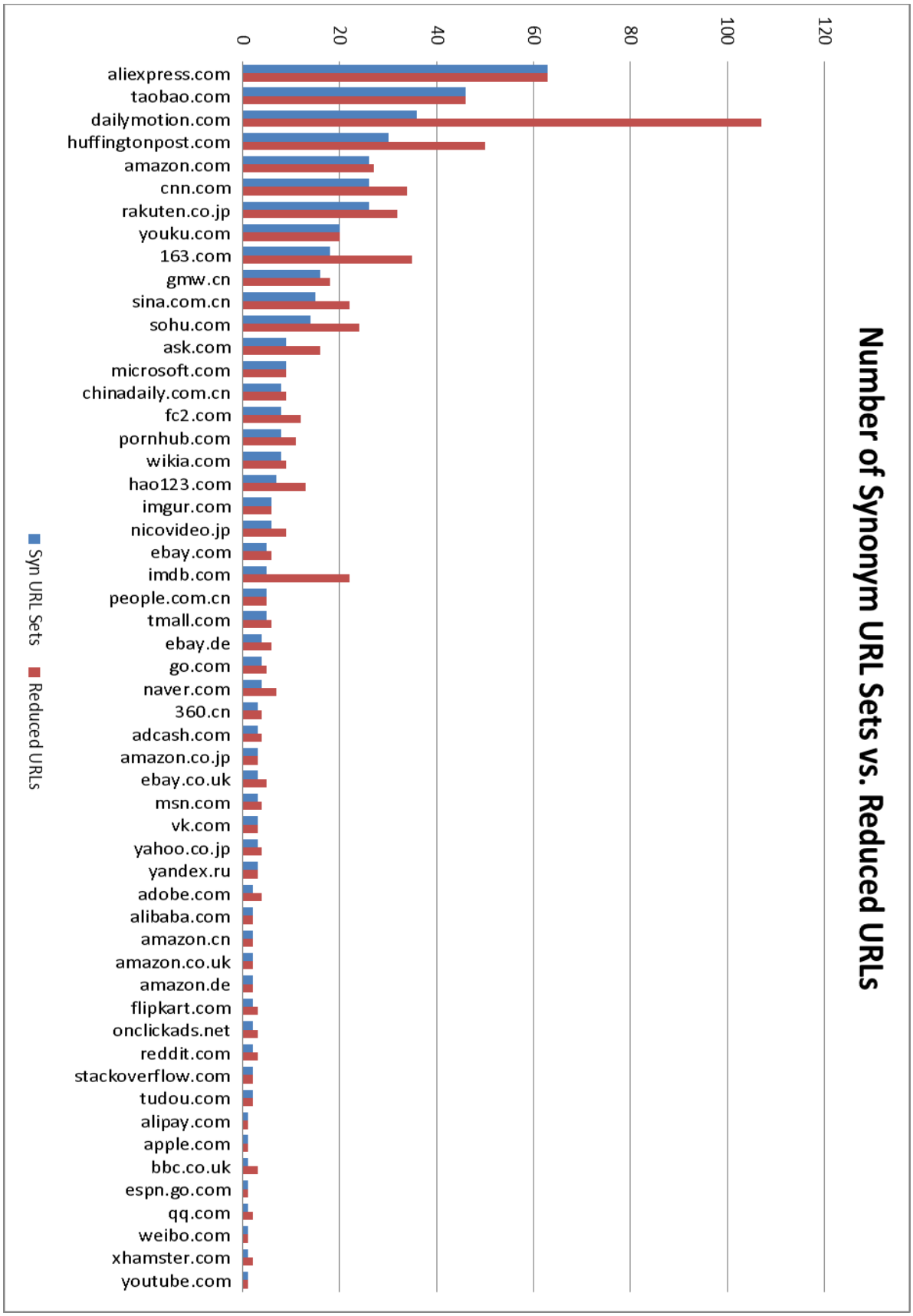
It is also interesting that there is a sharp cliff at the low end of both of these graph pairs, suggesting that something might be fundamentally different about how those sites on the low end manage resources. One final takeaway is that it is potentially reassuring that the percentage of consistent bytes tends to be higher than the percentage of consistent URLs. This suggests that a good portion of the content seen by viewers of these top 100 websites is fairly consistent across fetches. On the other hand, any inconsistency is cause for concern, and it does not necessarily require a lot of resource bytes for a website to engage in potentially untrustworthy behavior.

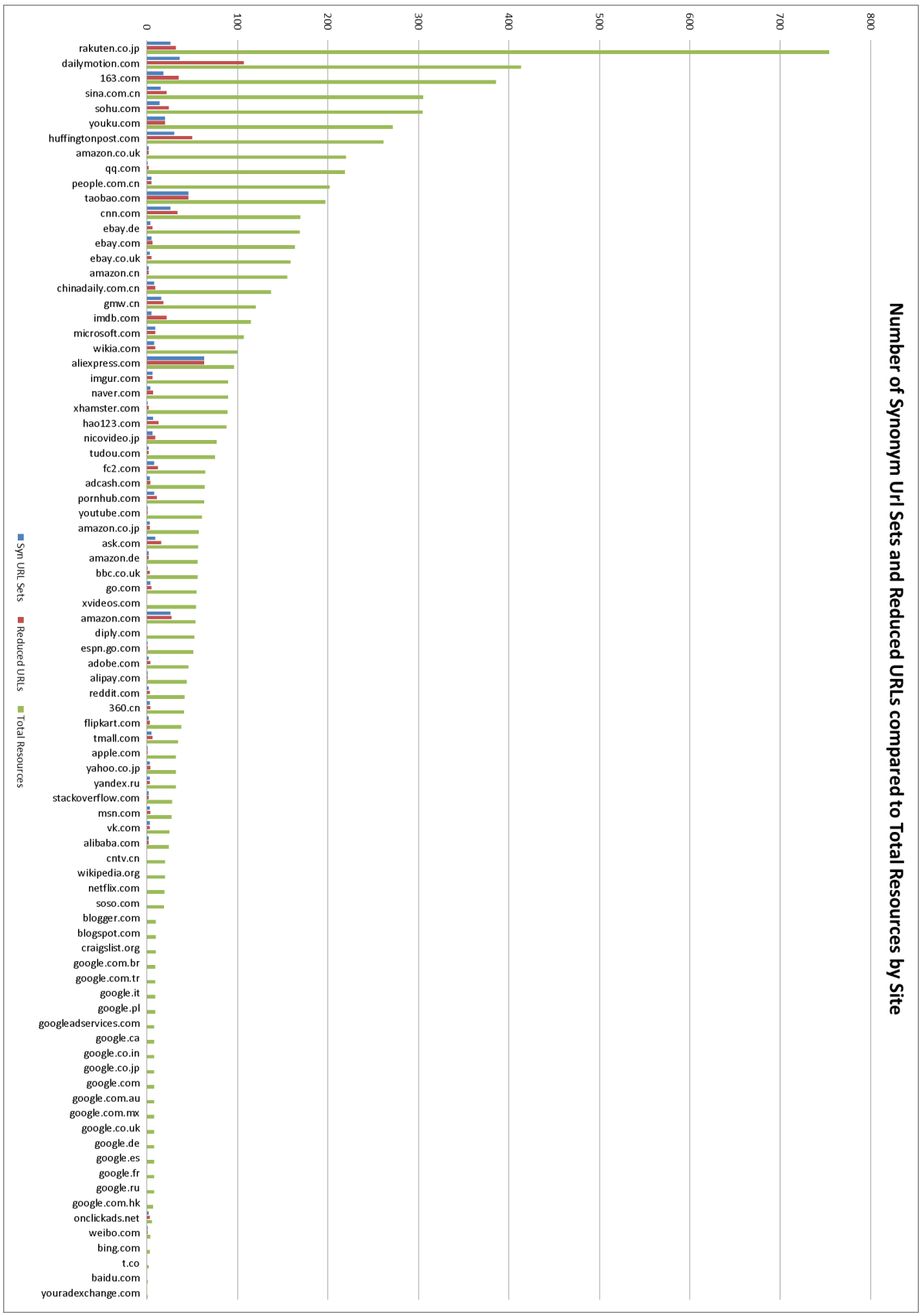
III.2 Synonym URL sets and Reduced URL Evaluation

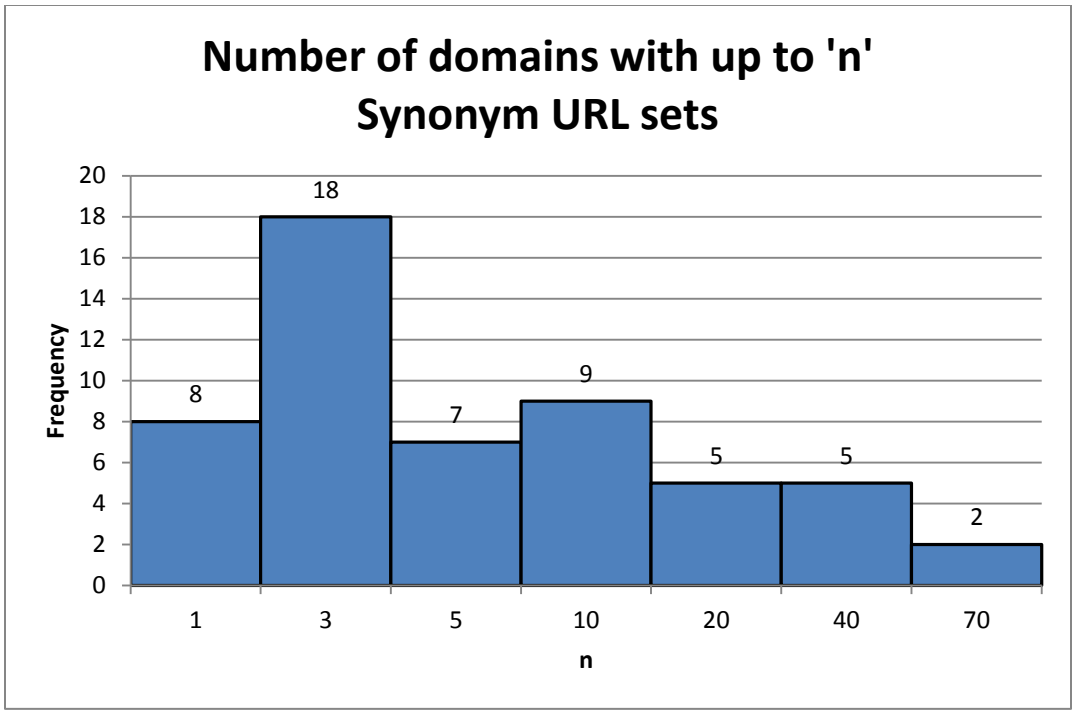
The next set of graphs looks at the number of synonym URL sets and reduced URLs for each site. In particular, these graphs give a sense of the general ratio of reduced URLs to synonym URL sets. In most cases, a synonym URL sets can be reduced to only one or two structural templates. These graphs are highly redundant but provide several ways to conceptualize the data.

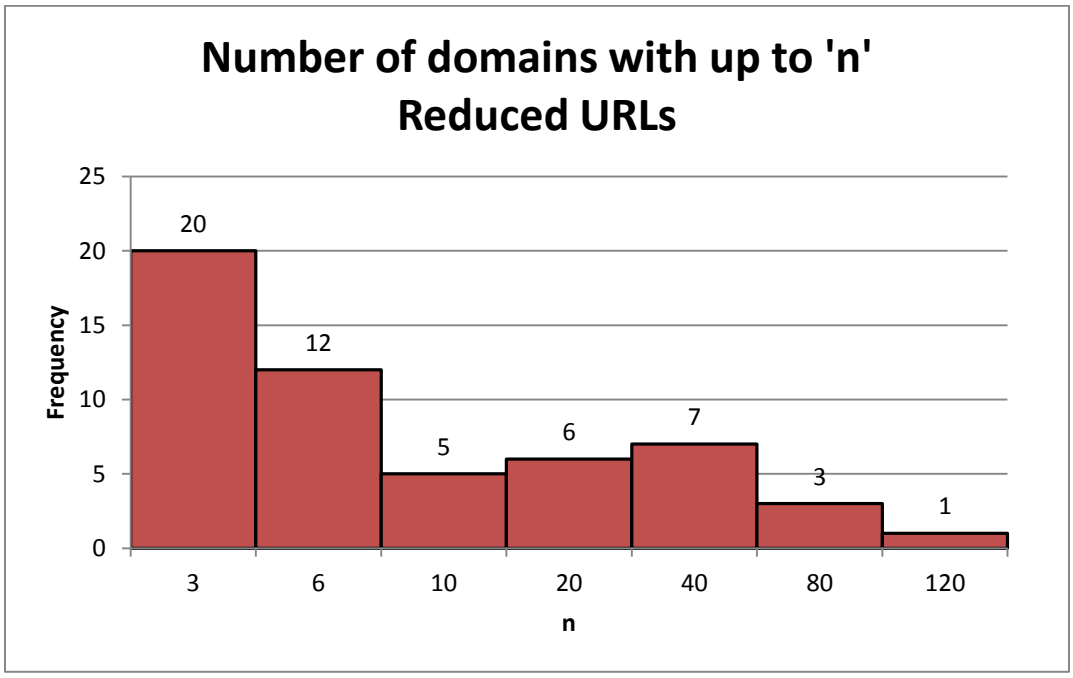

Number of Synonym URL Sets vs Reduced URLs (excluding 0s)
Number of Reduced URLs
Number of Synonym URL sets
y = 1.3215x + 0.922
R² = 0.7463
0
20
40
60
80
100
120
0
10
20
30
40
50
60
70


Number of Reduced URLs vs. "Reduction factor" (nReduced/nSynSets)
Reduction Factor
Number of Reduced URLs
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
20
40
60
80
100
120


Number of Synonym URL Sets vs. Reduced URLs
0
20
40
60
80
100
120
aliexpress.com
taobao.com
dailymotion.com
huffingtonpost.com
amazon.com
cnn.com
rakuten.co.jp
youku.com
163.com
gmw.cn
sina.com.cn
sohu.com
ask.com
microsoft.com
chinadaily.com.cn
fc2.com
pornhub.com
wikia.com
hao123.com
imgur.com
nicovideo.jp
ebay.com
imdb.com
people.com.cn
tmall.com
ebay.de
go.com
naver.com
360.cn
adcash.com
amazon.co.jp
ebay.co.uk
msn.com
vk.com
yahoo.co.jp
yandex.ru
adobe.com
alibaba.com
amazon.cn
amazon.co.uk
amazon.de
flipkart.com
onclickads.net
reddit.com
stackoverflow.com
tudou.com
alipay.com
apple.com
bbc.co.uk
espn.go.com
qq.com
weibo.com
xhamster.com
youtube.com
Syn URL Sets
Reduced URLs

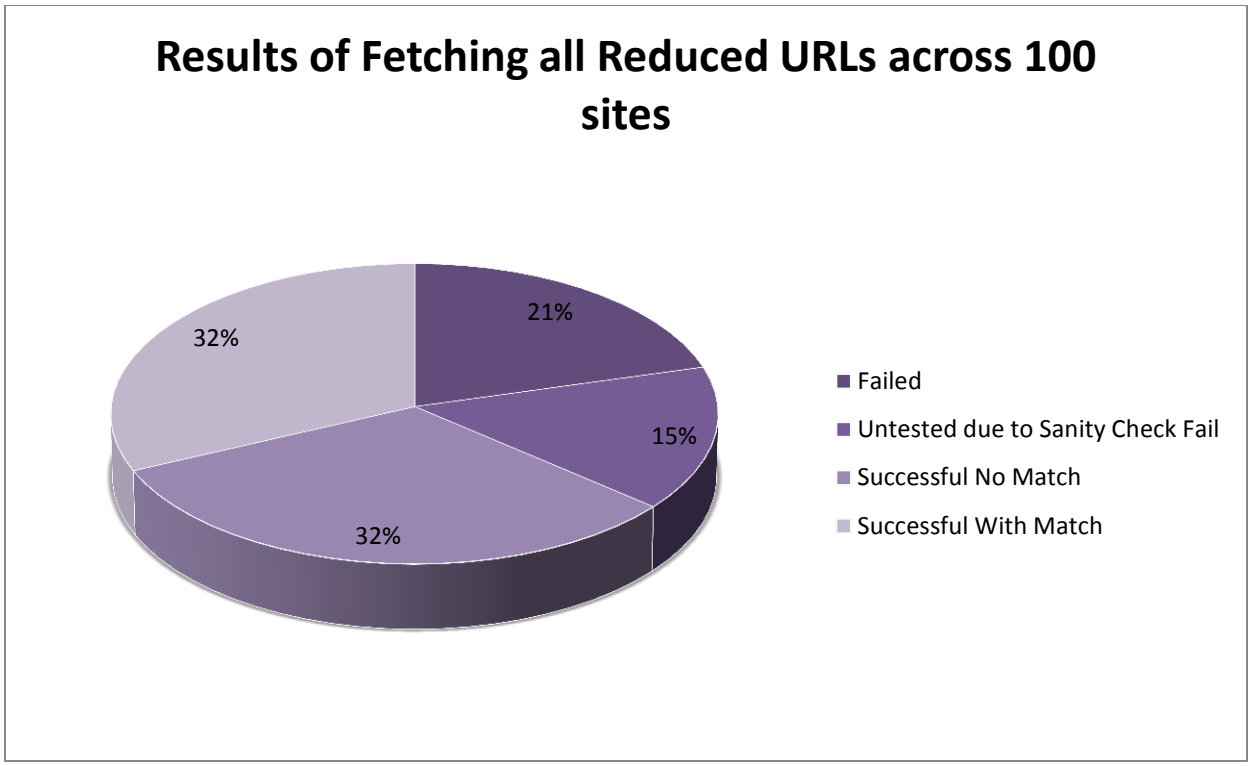

Number of Synonym Url Sets and Reduced URLs compared to Total Resources by Site
Syn URL Sets
Reduced URLs
Total Resources
0
100
200
300
400
500
600
700
800
rakuten.co.jp
dailymotion.com
163.com
sina.com.cn
sohu.com
youku.com
huffingtonpost.com
amazon.co.uk
qq.com
people.com.cn
taobao.com
cnn.com
ebay.de
ebay.com
ebay.co.uk
amazon.cn
chinadaily.com.cn
gmw.cn
imdb.com
microsoft.com
wikia.com
aliexpress.com
imgur.com
naver.com
xhamster.com
hao123.com
nicovideo.jp
tudou.com
fc2.com
adcash.com
pornhub.com
youtube.com
amazon.co.jp
ask.com
amazon.de
bbc.co.uk
go.com
xvideos.com
amazon.com
diply.com
espn.go.com
adobe.com
alipay.com
reddit.com
360.cn
flipkart.com
tmall.com
apple.com
yahoo.co.jp
yandex.ru
stackoverflow.com
msn.com
vk.com
alibaba.com
cntv.cn
wikipedia.org
netflix.com
soso.com
blogger.com
blogspot.com
craigslist.org
google.com.br
google.com.tr
google.it
google.pl
googleadservices.com
google.ca
google.co.in
google.co.jp
google.com
google.com.au
google.com.mx
google.co.uk
google.de
google.es
google.fr
google.ru
google.com.hk
onclickads.net
weibo.com
bing.com
t.co
baidu.com
youradexchange.com





Keep in mind that other than the graph showing number of synonym URLs compared to total resources, none of these graphs take into account sites for which no synonym URLs were identified. The key takeaway from these data is that the number of synonym URLs varies widely from site to site but usually remains small relative to the total number of resources for the site.

The next set of graphs shows the results of attempting to fetch the reduced URLs.

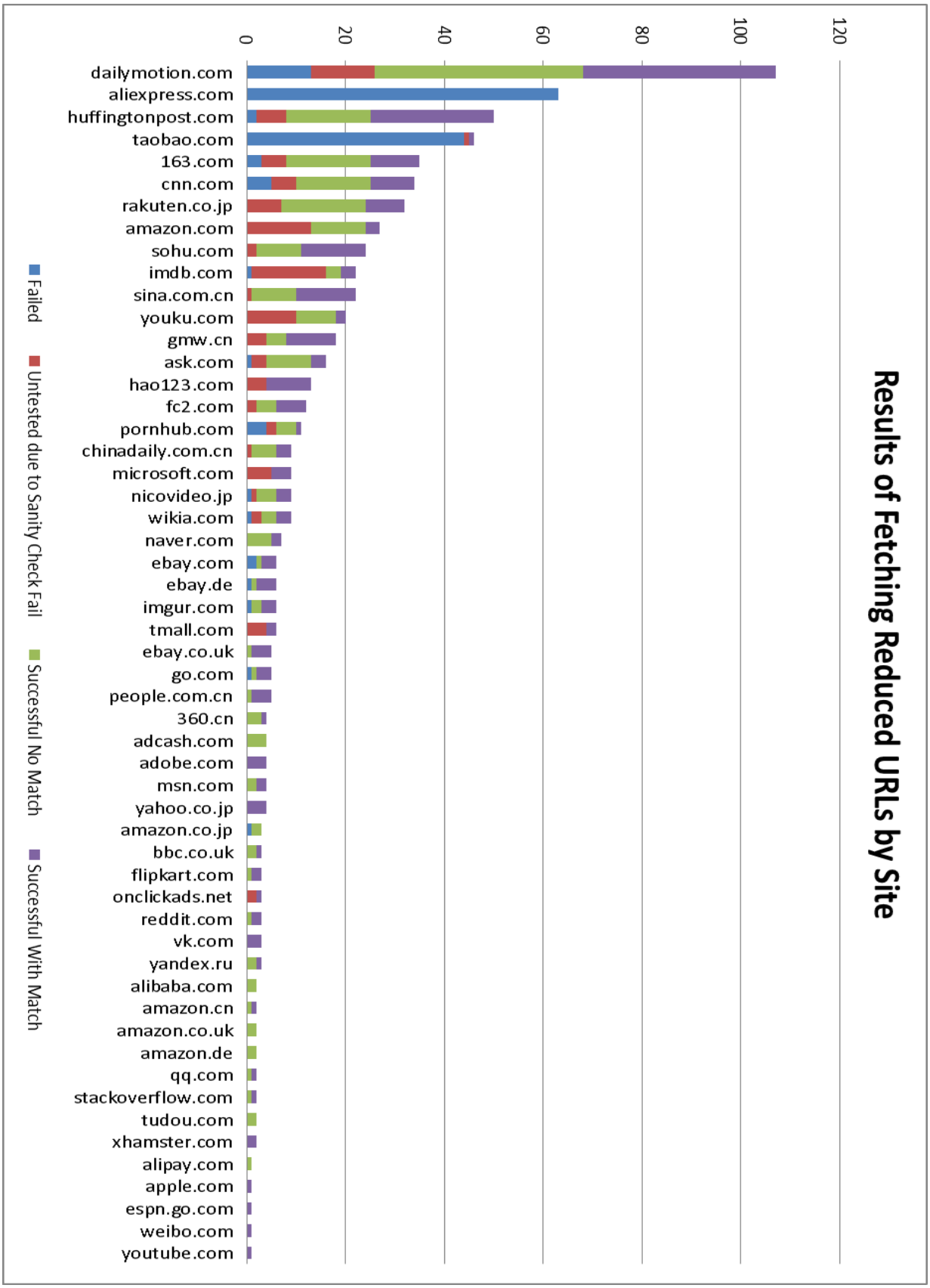


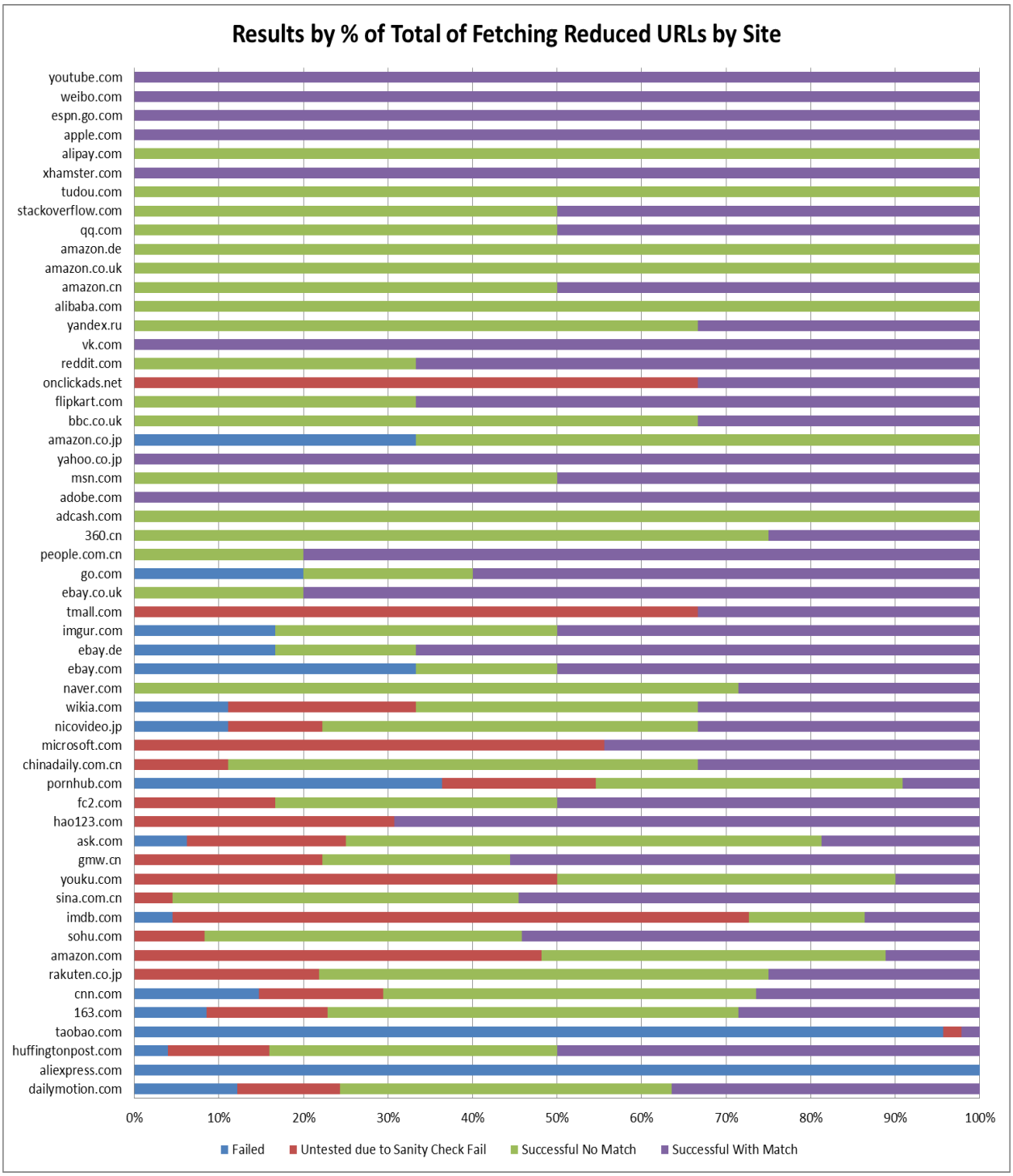
Overall Reduced URL Results	
Total	696
Failed	144
Untested due to Sanity Check Fail	108
Successful No Match	222
Successful With Match	222

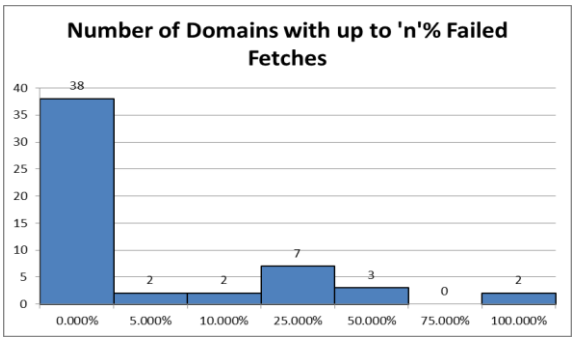
As a reminder, the "sanity check" refers to the fact that before fetching any reduced URL, the process script attempted to fetch one of the original synonym URLs to ensure that it could retrieve contents whose hash was equal to the hash initially associated with the synonym set. This was one of the most surprising results of my analysis to me, because I had assumed that the sanity check would never fail. It is an interesting question why the sanity check did fail. More research is needed to experimentally evaluate different possible answers, but there are a couple of conjectural answers. One possibility is that some of the resource URLs that wind up in synonym sets are generated with the intent that they can only be fetched once from the web server, so that when the script tries to re-fetch them they fail one way or another. Another possibility is that certain resource URLs somehow depend on the cookie state or other underlying browser state that is somehow set such that these URLs are valid when fetched after fetching the top-domain which requests them, but nonsensical when fetching them directly on their own.

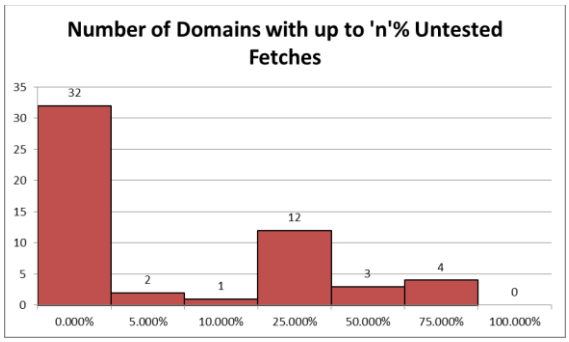
"Successful No Match" refers to reduced URLs which were successfully fetched, but which returned no resources whose contents hashed to the same value as the hash associated with the synonym set. "Successful With Match" refers to those reduced URLs which did successfully retrieve a resource with the same contents as the synonym URLs. "Failed" simply refers to

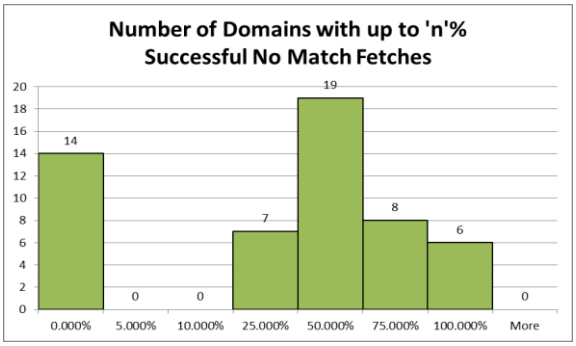
reduced URLs which were not successfully fetched. It is entirely a coincidence that the number of “Successful No Match” and “Successful With Match” URLs was the same. The next graph will demonstrate this, by breaking down the results of fetching reduced URLs by site. Note again that all 46 sites which had no synonym URLs are excluded. The set of graphs after that will show the same results broken down by percentage. As an advance warning, the percentage graph is hideous and I hesitate to even show it because I still consider it somewhat misleading.

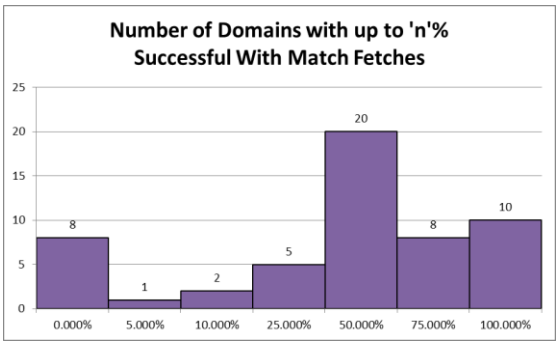

Results of Fetching Reduced URLs by Site
0
20
40
60
80
100
120
dailymotion.com
aliexpress.com
huffingtonpost.com
taobao.com
163.com
cnn.com
rakuten.co.jp
amazon.com
sohu.com
imdb.com
sina.com.cn
youku.com
gmw.cn
ask.com
hao123.com
fc2.com
pornhub.com
chinadaily.com.cn
microsoft.com
nicovideo.jp
wikia.com
naver.com
ebay.com
ebay.de
imgur.com
tmall.com
ebay.co.uk
go.com
people.com.cn
360.cn
adcash.com
adobe.com
msn.com
yahoo.co.jp
amazon.co.jp
bbc.co.uk
flipkart.com
onclickads.net
reddit.com
vk.com
yandex.ru
alibaba.com
amazon.cn
amazon.co.uk
amazon.de
qq.com
stackoverflow.com
tudou.com
xhamster.com
alipay.com
apple.com
espn.go.com
weibo.com
youtube.com
Failed
Untested due to Sanity Check Fail
Successful No Match
Successful With Match


Results by % of Total of Fetching Reduced URLs by Site
youtube.com
weibo.com
espn.go.com
apple.com
alipay.com
xhamster.com
tudou.com
stackoverflow.com
qq.com
amazon.de
amazon.co.uk
amazon.cn
alibaba.com
yandex.ru
vk.com
reddit.com
onclickads.net
flipkart.com
bbc.co.uk
amazon.co.jp
yahoo.co.jp
msn.com
adobe.com
adcash.com
360.cn
people.com.cn
go.com
ebay.co.uk
tmall.com
imgur.com
ebay.de
ebay.com
naver.com
wikia.com
nicovideo.jp
microsoft.com
chinadaily.com.cn
pornhub.com
fc2.com
hao123.com
ask.com
gmw.cn
youku.com
sina.com.cn
imdb.com
sohu.com
amazon.com
rakuten.co.jp
cnn.com
163.com
taobao.com
huffingtonpost.com
aliexpress.com
dailymotion.com
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Failed
Untested due to Sanity Check Fail
Successful No Match
Successful With Match









These data are difficult to understand and interpret, but the most obvious observation is that the current method of URL reduction is not hopeless; there are certainly sites for which a good number of the reduced URLs were successful and retrieved matching contents. This fact is apparent from looking at the purple segment of the bars in the chart showing the results of fetching all reduced URLs by site; in sites such as dailymotion.com, sohu.com, and Huffingtonpost.com a substantial number of the reduced URLs proved successful. It is equally clear that the current method of creating reduced URLs is not a be-all-end-all solution for eliminating unique parts of URLs which map to identical contents. More than 2/3 of the reduced URLs tested were inadequate replacements for the original synonym URLs for one reason or another.

These data demonstrate that this method of constructing reducing synonym URLs has some promise in combatting user-tracking through the use of extraneous information embedded within URLs, but more research is needed to discover and evaluate additional methods of synonym URL reduction.

IV. Conclusion and Future Work

The data gathered in this analysis represent a significant first step in evaluating website trustworthiness and possible methods to mitigate untrustworthy behavior. The resource categorization data discussed in the first half of section III offer meaningful insight into the consistency of some of the most popular websites worldwide on the Internet. However, more analysis is needed to derive meaning from the data. In the future, data from a wider array of input

sites must be gathered so that these consistency metrics can be compared across sites within the same broad category; nobody expects online shopping websites to be structured in the same way as news sites or social networks.

There is also more analysis that can be done evaluating the meaning of different classes of inconsistency; for example, consistent URLs which return varying contents, and resources whose URLs and contents are unique to a single fetch. Ideally, the resource categorization algorithm should construct "similarity sets" where each set contains no more than one URL per fetch, and every URL in the set is more similar to every other URL in the set than to any other URL from the same fetch. This would also allow the algorithm to definitively identify "similar" URLs, which vary slightly in both structure and contents, but may serve the same structural purpose in their respective fetches of the site. The code in urltable.py was an attempt to implement these similarity sets, but rather than limiting the sets to one URL per fetch any URL can be sorted into the set if it is similar to the contents of the set up to a certain threshold.

In addition, more work needs to be done to find ways to successfully fetch the contents of pages from bot-protected domains like Facebook and Twitter, which the existing SlimerJS script was unable to fetch successfully even once. It would be ideal to have a browser extension that runs in the background of volunteer users' computers and aggregates in real-time the same sort of result pages that the SlimerJS script attempts to generate by fetching sites. Such an extension would allow researchers to generate data for sites that are much harder to harvest data from remotely because of anti-bot or other protections. Obviously such an extension would raise its own plethora of privacy concerns, but initially at least, with a small and trustworthy user base in a research community it is a viable option for generating data for these protected websites.

It might also be interesting for iterations of the script to dive deeper than the top-level site name. For instance, fetching the top-level domain of search engines like Google.com is useful, but it does not represent the actual user experience of google, which is spent mainly navigating to and from search results pages, as well as image, regular, and map search. The above-mentioned hypothetical browser extension could also aggregate this sort of data as the user naturally navigates the web.

The data discussed in the second half of section III demonstrate that synonym URL reduction has clear potential as a strategy to circumvent user-tracking by information embedded within URLs, but in its current form the reduction is far from perfect. Significantly more of the reduced URLs fail in one way or another than return contents identical to those retrieved by the original URLs.

Some of the biggest weaknesses in the analysis code include the URL reduction algorithm and the similarity check algorithm that I described in section II. I described potential improvements in detail above, but in brief, the similarity check needs to be improved to allow URLs of different lengths, one of which may include "extra" internal segments, to be evaluated as similar. It would also be valuable to explore an alternative URL reduction algorithm that looks at all corresponding segments across a structurally similar set of URLs at once to choose a candidate for the corresponding "reduced" URL segment, possibly giving less attention to the

order of the segments within different URLs especially in cases like the query string of a URL where such ordering is entirely arbitrary.

An additional weakness that must be addressed is that currently the resource categorization algorithm makes the simplifying assumption for all sites that no resource URL can appear twice within a resource list for the same fetch. This is a harmful and often untrue assumption, and modifying the code to be able to get rid of that assumption could lead to significant changes in the categorization of some resource URLs.

While there is still much work to be done analyzing existing data and generating additional data for processing, and there still exist some significant shortcomings in algorithms within the processing code, the website analysis and processing system has proven a successful baseline for facilitating observations about website consistency, and the data it has generated prove that there is hope for URL reduction as a potential strategy for combatting user-tracking through extraneous URL components.

V. Acknowledgements

I would like to thank my advisor in the Yale CS department, Bryan Ford, for his patience, guidance, creative wisdom, and exceptional teaching, and my collaborator and “de facto” advisor, Daniel Jackowitz, for guiding me throughout the semester and helping me figure out what questions need to be asked. I would also like to thank my family and friends for their support, and everyone in the Yale Computer Science department who keeps the show running.