HTML Resource Package Experiments

Justin Lebar

June 7, 2010

1 Introduction

As a CS191W project, we implemented a basic version of HTML resource packages[1] in Firefox and wrote a formal specification for the feature.[2] Resource packages allow websites to aggregate together sets of resources into one zip or tar-gz file. Instead of issuing requests for each resource individually, browsers download a resource package with a single HTTP request.

In theory, reducing the number of HTTP requests necessary to load a page should improve its load speed, especially on high-latency networks, where the overhead of issuing a request is high. In this writeup, we document two experiments we conducted to judge the efficacy of resource packages in improving the speed of loading webpages. Our results are encouraging.

2 Experiments

2.1 Website survey

We first conducted an informal survey of websites to get an idea of how many resources webpages usually include and how large they usually are.

Table 1 includes the results of our survey. We should note that we ran our tests only once, so our results aren't particularly accurate. (We found that even the number of requests varied greatly between two consecutive page loads for some pages.) All tests were run with an empty browser cache.

Two patterns emerge from our data: First, the average resource on many pages is about 4KB and usually is smaller than 10KB (although there are clear exceptions to this). Second, even rich-content pages include wildly differing numbers of resources.

The main conclusion we draw is that despite the popularity of image spriting and js/css unification as techniques to decrease the number of resources on a page, many pages—even highly-optimized pages such as Facebook's homepage—include many relatively small resources. If our goal is to reduce the number of

[1] Introduced at `http://limi.net/articles/resource-packages/`

[2] `http://stanford.edu/~jlebar/moz/respkg`

	Size	Requests	KB/req	Eth (s)	3G (s)
nytimes.com	4.8 MB	170	28	9	27
cnn.com	1.3 MB	141	9.2	14	23
cakewrecks.com	1.1 MB	129	8.5	9	48
facebook.com/home.php	421 KB	100	4.2	6	13
bbc.co.uk	312 KB	78	4.0	15	39
amazon.com	424 KB	80	5.3	7	16
en.wikipedia.org	230 KB	54	4.2	3	10
Google image search	144 KB	25	5.8	1	4
youtube.com	235 KB	17	14	3	15

Table 1: Results from our survey of websites. The *eth* column lists seconds to `onload` dispatch over unmodified ethernet connection; *3G* column lists seconds to `onload` over a 700 KB/s, 440ms (roundtrip) connection.

HTTP requests necessary to load a page, there is clearly room for improvement even on well-optimized sites.

2.2 Resource package tests

We conducted experiments using our proof-of-concept implementation of resource packages in Firefox to determine whether resource packaging would yield faster page load times.

We constructed an artificial benchmark for this test so we could run it locally without experiencing the variability of real network speeds. The page consists of 126 resources, mostly thumbnail images, totaling 1.3MB. This makes it similar on average to cnn.com or cakewrecks.com in terms of total size and number of resources. Our results are summarized in Table 2.

Consistent with our expectations, resource packages never made a page slower, and the largest gains from using resource packages were on high-bandwidth links. This demonstrates that resource packaging is beneficial across a large range of connection types.

Perhaps the most intriguing data in our table are for the connections with 880ms of latency. In those cases, without resource packages, the page loaded at almost the same speed regardless of how much bandwidth was available. We suspect that this is an effect of TCP slow-start interacting with the serialization of HTTP requests: TCP slow-start increases the stream's window size as we send more packets. Without resource packages, each of our TCP connections wastes half an RTT for each HTTP request it issues (i.e. we wait for all the packets for our last request to arrive, and then we send our next HTTP GET request, which takes half an RTT to arrive at the server). In contrast, when we use a resource package, we fully utilize our TCP stream. At high latencies, forcing a flush of the TCP window on every HTTP request should add significant overhead. With resource packages, we see performance gains from adding bandwidth up

KB/s	RTT (ms)	No pkg (s)	Pkg (s)
175	55	8.6	8
175	110	9.6	8.4
175	220	11.4	9.0
175	440	15.0	11.4
175	880	24.5	15.5
350	55	4.8	4.3
350	110	5.8	4.8
350	220	7.7	7.1
350	440	12.4	7.9
350	880	23.6	12.2
700	55	3.5	2.4
700	110	3.9	2.7
700	220	6.4	3.5
700	440	12.0	7.5
700	880	23.4	9.2
1400	55	3.3	1.4
1400	110	3.2	1.5
1400	220	6.0	2.7
1400	440	11.9	5.7
1400	880	23.3	9.0

Table 2: Time to load a page with 126 resources totalling 1.3MB over various (simulated) network conditions with and without resource packages.

to 700 KB/s.

3 Conclusions

Overall, we saw significant gains from using resource packages. When given 1400 KB/s of bandwidth, our benchmark page loaded more than 2x faster with resource packages than without at all latencies. Our benchmark represents a best-case scenario in some ways, since we were able to encapsulate all of our page's resources into a single package. But even if only some of a page's resources could be packaged, we'd expect to see gains proportional to the number of packaged images.

Crucially, resource packages are easy to add to a webpage (just add one line of HTML in the page's head), and backwards compatible with older browsers, which will just ignore the packages and download the resources individually. Given the performance improvements we've seen with our proof-of-concept implementation, we hope that a full implementation will be well-received by browser vendors and web developers alike.