FlexiWeb: Network-Aware Compaction for Accelerating Mobile Web
What’s the impact of web latency?
Amazon

100ms delay leads to 1% decrease in sales.

Source: https://speakerdeck.com/deanohume/faster-mobile-websites
Walmart

100ms

Delay

1%

revenue

Source: https://speakerdeck.com/deanohume/faster-mobile-websites
400ms delay results in 9% increase in traffic.

Source: https://speakerdeck.com/deanohume/faster-mobile-websites
500ms
Delay

25%
searches

Source: https://speakerdeck.com/deanohume/faster-mobile-websites
What does it take to load a web page?

- Prompt for Unload
- DNS Resolution
- TCP Handshake and Connection
- Send Request and Wait for Response
- Request Sub Resources
- Execute Scripts and Apply CSS

Number of RTTs dominates the web latency

$\times 400$
Existing Approaches?
Follow the best practices

HTTP/2
SPDY

New optimizations every few weeks
Data Compression Proxies
(or Cloud Assisted Browsing)

- Data Compression: Off
- Data Compression: On
- Compression Proxy
- Content-aware Compression:
  eg. minify JS
- Transcoding Images:
  eg. PNG to WebP
- http website

Browser Options:
- Google
- SILK
- UC Browser
Data Compression Proxies (or Cloud Assisted Browsing)

Client’s network condition is ignored
Static transformation of content
Related Work

- “Towards a spdy’ier mobile web?”, ACM CoNEXT 2013
- “Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices”, NSDI 2015
Is Compression proxy always useful?
Measurement Setup

Clients: Multiple Android devices with Google Chrome browser

Network Conditions:

- 2 Cellular service providers: AT&T and T-Mobile

Choose 4 different locations (based on similar RTT and throughput values)

<table>
<thead>
<tr>
<th>Network Conditions</th>
<th>RTT (ms)</th>
<th>Throughput (Mbps)</th>
<th>Loss Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent</td>
<td>100</td>
<td>5</td>
<td>0.006</td>
</tr>
<tr>
<td>Good</td>
<td>200</td>
<td>2</td>
<td>0.006</td>
</tr>
<tr>
<td>Fair</td>
<td>400</td>
<td>1</td>
<td>0.04</td>
</tr>
<tr>
<td>Poor</td>
<td>600</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Measurement Setup

Schemes :

Direct : All the requests are directly sent to the web server

Compression Proxy (Google’s data compression proxy) : All the requests are sent to the compression proxy

Data Collection :

Each phone downloads the Alexa’s top 500 websites
All experiments were performed with cold browser cache
Page Load Time (PLT) is recorded for each page
Results are averaged over 10 trials
Data Compression Proxy Performance:

- Gain of 32% in bad network conditions
- 28% degradation in excellent network conditions
Why does performance degrade in excellent network conditions?
Measurement Results

Circuitous Routing

Paths were inflated by up to 8 hops, which increased the RTT by up to 80 ms

Processing Overhead

For example, the encoding and decoding of WebP is ~10X and ~1.4X slower than JPEG
Measurements: Controlled settings

Set up our own compression proxy on Amazon EC-2

Compression Proxy: Apache + mod_pagespeed (with recommended settings)

Replayed captured webpages to avoid change in content
Measurement Results

Object Size VS. Average Load Time

Good/Excellent conditions only fetch objects >30KB via proxy
Fair/Poor conditions fetch all objects via proxy
FlexiWeb : A framework that determines both *when* to use a compression proxy and *how* to use it, based on the client’s network conditions.
flexiweb: overview
splitting requests

Request Splitting Module

Uses measurement based mapping to dynamically send the request either directly to the web server or to the proxy.

<table>
<thead>
<tr>
<th>Network Condition</th>
<th>0-1 KB</th>
<th>1-3 KB</th>
<th>3-6 KB</th>
<th>6-10 KB</th>
<th>10-20 KB</th>
<th>20-40 KB</th>
<th>&gt; 40 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent</td>
<td>Direct</td>
<td>Direct</td>
<td>Direct</td>
<td>Direct</td>
<td>Direct</td>
<td>Proxy</td>
<td>Proxy</td>
</tr>
<tr>
<td>Good</td>
<td>Direct</td>
<td>Direct</td>
<td>Direct</td>
<td>Direct</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
</tr>
<tr>
<td>Fair</td>
<td>Direct</td>
<td>Direct</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
</tr>
<tr>
<td>Poor</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
<td>Proxy</td>
</tr>
</tbody>
</table>

Challenge: How to figure out the size of the object without downloading it?
predicting object size

Browser downloads the main html file *(index.html)* with object URLs

How to predict the size range of an object using only its URL?

- Issuing a HTTP Head request? Incurs an additional RTT

Multi-Class Classification Problem: Classifying a URL into one of the size ranges

We use *Random Forest Classifier* to classify the objects in different size ranges
predicting object size

Feature Extraction: We extract features from the object URL using “bag of words” technique.

For example: http://i.cdn.turner.com/cnn/.e/img/3.0/global/icons/gallery_icon2.png

Yields following bag of words: {i, cdn, turner, com, cnn, .e, img, 3.0, global, icons, gallery_icon2, png}

Occurrence of each word is used as a feature for training the classifier.

Client uses the classifier to predict the object size range.
Assessing Network Conditions

*Network measurement module* tracks RTT, loss rate, and TCP throughput to the client.

**Browser always downloads the main html file via the proxy**

Proxy calculates the median RTT and TCP throughput to the client using packets exchanged during the main html file download.

Proxy sends this “*Network Condition Report*” back to the client via object response headers.
Network aware compression

**Goal**: Adaptively transform a web page’s content to deliver the page within the user’s attention span (2 to 5 sec)

We only focus on adaptively transforming the images on the web page

Images make up around 60% percent of the bytes on a average web page

Text and JS are compressed similar to traditional proxy assisted browsers

Source: httparchive.org
Set of transformations:
- WebP with 85% quality,
- WebP with 65% quality,
- WebP with 45% quality,
- WebP with 25% quality,
- WebP with 5% quality

Each transformation has an associated cost and utility:
- **Cost**: Time required to download the transformed image from the proxy to client
- **Utility**: PSNR of the resulting transformation

**Challenge**: How to choose the right transformation based on the cost and utility?
Network aware compression

Let's assume that there are $N$ images in a web page and a total page load time budget of $B$ seconds.

**Goal**: Maximize the sum of utilities of all the selected versions of $N$ images

subject to

1. Exactly one version of each image can be selected
2. Total cost of all the selected versions can not exceed the total budget $B$

This problem can be mapped to the Online Multi-Choice Knapsack Problem (MCKP)
Implementation and Setup

**Client Side Implementation** : Using Google’s Chromium open source android browser

**Proxy Side Implementation** : Using `mod_pagespeed` and **apache** web server

**Network Conditions** :
- **Controlled Settings** : Dummynet to emulate the network conditions to reduce the variability
- **Cellular Networks** : **AT&T** and **T-Mobile**
Implementation and Setup

**WebPage Requested**: Alexa top 500 web sites visited by mobile users

**Metric**: Page Load Time

**Schemes**:

1. **Direct**: All the requests are directly sent to the web server
2. **Compression Proxy**: All the requests are sent to compression proxy (with WebP quality 75%)
3. **FlexiWeb**
Performance of FlexiWeb

FlexiWeb provides up to 38% gain in excellent network condition
8% gain in poor network conditions
Average precision >90% for objects of size < 1Kb & > 20Kb

We can predict other object sizes with at least 70% precision

\[
Precision(\%) = \left( \frac{TruePositive}{TruePositive + FalsePositive} \right) \times 100.
\]
Chrome for Android can now save more data by blocking website images
Conclusions

We show that today’s compression proxies can increase mobile web page load times

Content transformations should be network aware

Based on these observations we implemented FlexiWeb, a framework that support network aware proxy usage