Optimizing Web Content Download in Low-Performance Networks.

Ian Hunter
B.A.(Mod.) Computer Science

Final Year Project April 2014
Supervisor: Prof. Séamus Lawless

School of Computer Science and Statistics

O’Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

__________________________________   ________________________________
Name                                        Date
Acknowledgements

I would like to express my appreciation to Séamus Lawless for his support throughout the project, his valued guidance and trusting in this self-proposed project.

I would also like to thank Tom Mason from Trinity’s Internet Society for setting up a virtual machine for me to test in, and Ian O’Keeffe from the Knowledge and Data Engineering Group for setting up one for me on the college’s servers.

Stephen Rogers and Scott Cunningham should also be mentioned here for tolerating my never ending questioning, and for being efficient team members in the course’s group project that was run concurrently to this project, leaving me with more time to devote my efforts on this research effort.

I would also like to thank all participants in the user tests as without them I would have no user test results.

And last, but not least, I would like to thank my parents for housing me, feeding me and generally being great parents for my duration of college, without whom I wouldn’t be sitting down and writing this report.
Abstract

Slow internet and high data costs are large barriers to accessing the internet in certain areas of the world. Faster Internet speeds result in increased commercial sales, higher levels of productivity and consumer enjoyment.

This study is an investigation into potential alleviations of these costs by reducing data sizes, so that transfer time and data costs are reduced. Several experimental techniques are explored.

The investigation will involve creating a system to materialize these concepts and evaluate their effectiveness in terms of reduction and impact on users. This system will take the form of a two-tier proxy and perform reductions at multiple stages of the data transmission process.

This investigation will hopefully contribute to the current research that is being run by multiple large-scale Internet-based companies such as Google and Mozilla that focus on speeding up the web for the masses.
# Table of Contents

**Acknowledgements**

**Abstract**

**Table of Contents**

**Table of Figures**

**Chapter 1. Introduction**

1.1 - Research Motivation 1

1.2 - Research Objectives 3

1.3 - Technical Approach / Methodology 3

1.4 - Overview of Report 4

**Chapter 2. Background & Related Work**

2.1 - Reduction over HTTP 6

2.2 - HTML Reduction 7

2.3 - Image Reduction 8

2.4 - Emerging Techniques 10

2.5 - Related Work 10

**Chapter 3. Problem Overview**

3.1 - Techniques Explored 12

3.2 - Development Decisions 12

**Chapter 4. Design and Implementation**

4.1 - Requirements 14

4.2 - Issues affecting design 14

4.3 - Target User Model 14

4.4 - Data Storage 15

4.5 - Complete System Overview 15

4.5.0 - Message Path 16

4.5.1 - Lazy Image Loader 17

4.5.2 - Customization Extension 18

4.5.3 - Local Proxy 19

4.5.4 - Flask Web Server 19

4.5.5 - Remote Proxy 20

4.5.5.1 - Intelligent Content Removal 20

4.5.5.2 - Webpage Compression 21

4.5.5.3 - Raster to Vector Conversion 21

4.5.6 - Unknown Webserver 26

4.5.7 - Evaluation Software 26

**Chapter 5. Analysis & Evaluation**

5.1 - Statistical Tests 27

5.1.1 - Final Statistical Data Sets 28

5.1.2 - Forced GZip Compression 30

5.1.3 - Raster to Vector Conversion 31
## Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 1.1</td>
<td>Expected population growth for year 2050</td>
<td>1</td>
</tr>
<tr>
<td>Fig. 1.2</td>
<td>Mobile versus desktop usage</td>
<td>2</td>
</tr>
<tr>
<td>Fig. 1.3</td>
<td>Layperson view of the Internet</td>
<td>3</td>
</tr>
<tr>
<td>Fig. 1.4</td>
<td>Physical Setup</td>
<td>4</td>
</tr>
<tr>
<td>Fig. 2.1</td>
<td>Uptake of compression in the ‘Fortune 1000’</td>
<td>6</td>
</tr>
<tr>
<td>Fig. 2.2</td>
<td>Original &amp; artifacted Image</td>
<td>8</td>
</tr>
<tr>
<td>Fig. 2.3</td>
<td>Original and Lossy Compression</td>
<td>9</td>
</tr>
<tr>
<td>Fig. 2.4</td>
<td>Example of limiting colour range</td>
<td>9</td>
</tr>
<tr>
<td>Fig. 2.5</td>
<td>Enlarged raster image versus enlarged vector image</td>
<td>10</td>
</tr>
<tr>
<td>Fig. 2.6</td>
<td>Opera Turbo</td>
<td>11</td>
</tr>
<tr>
<td>Fig. 4.1</td>
<td>System Overview</td>
<td>16</td>
</tr>
<tr>
<td>Fig. 4.2</td>
<td>Effects of lazy loading images</td>
<td>18</td>
</tr>
<tr>
<td>Fig. 4.3</td>
<td>Chrome extension</td>
<td>19</td>
</tr>
<tr>
<td>Fig. 4.4</td>
<td>Example SQLite3 database entry</td>
<td>19</td>
</tr>
<tr>
<td>Fig. 4.5</td>
<td>JavaScript &amp; CSS styled comments</td>
<td>21</td>
</tr>
<tr>
<td>Fig. 4.6</td>
<td>HTML Styled comments</td>
<td>21</td>
</tr>
<tr>
<td>Fig. 4.7</td>
<td>Internet Explorer-specific HTML comments</td>
<td>21</td>
</tr>
<tr>
<td>Fig. 4.8</td>
<td>Comparison of PNG images to SVG images</td>
<td>22</td>
</tr>
<tr>
<td>Fig. 4.9</td>
<td>Sample original raster image</td>
<td>22</td>
</tr>
<tr>
<td>Fig. 4.10</td>
<td>Example feature space with clustering</td>
<td>23</td>
</tr>
<tr>
<td>Fig. 4.11</td>
<td>Previous sample image with k-means clustering applied</td>
<td>23</td>
</tr>
<tr>
<td>Fig. 4.12</td>
<td>Various binary images and their counterpart contours</td>
<td>24</td>
</tr>
<tr>
<td>Fig. 4.13</td>
<td>Resultant vector image</td>
<td>25</td>
</tr>
<tr>
<td>Fig. 4.14</td>
<td>Path co-ordinate reduction</td>
<td>25</td>
</tr>
</tbody>
</table>
Fig. 5.1 - Example dataset entry for a JSON transfer compressed with GZip

Fig. 5.2 - Content removal on a traditional desktop PC

Fig. 5.3 - Emulated content removal on a 320x480 mobile device

Fig. 5.4 - The effects of GZip compression

Fig. 5.5 - The effects of GZip compression on textual content

Fig. 5.6 - A comparison of sizes between raster images and vectors

Fig. 5.7 - Example requests saved in a single session with lazy loading enabled

Fig. 5.8 - Alternate view of saved requests

Fig. 5.9 - User feedback for forced GZip

Fig. 5.10 - User feedback for forced GZip and Image compression

Fig. 5.11 - User feedback for forced GZip, content removal and Image compression

Fig. 5.12 - User feedback for the lazy loader

Fig. 5.13 - User feedback for raster to vector conversion

Fig. 6.1 - Stepping effect of k-means on natural imagery

Fig. 6.2 - Feature loss caused by similar colours
Chapter 1.  Introduction

1.1 - Research Motivation

A lot of modern research goes into maximizing connection speeds in developed countries so that companies can offer their customers the highest bandwidth rates both in desktop and mobile environments. Emerging technologies such as 4G wireless networks or optical fibre connections are primarily targeted at richer countries who can afford to install these high cost technologies.

The developing world seems to be being left behind in these developments, but is starting to be considered more important as companies start to tailor devices towards these markets. The vast majority of the population of the world reside in non-developed countries, and it seems strange that commercial research has not made significant effort in this area. As birth rates decline in developed countries and continue to rise in underdeveloped countries, the significance of this market is still expected to grow largely [Fig 1.1]

There is definite agreement between involved parties that networking with the wider world has huge benefits to a developing community. As online connections can transcend barriers such as physical distance and social standing, this can lead to accelerated growth, ease of access to education, job creation, reduced migration pressure and increased productivity in many industry areas [2]. Indeed, the lack of a solid network infrastructure could further increase the gap between the developing and developed countries.

Areas such as Libya have abysmal connection speeds and high data tariffs - the average connection speed being under 256kb/s [3]. As there is only minor commercial interest in these countries, there is little competition and many areas are monopolised, reducing the motivation for further development [4].

As mobile phones are a cheaper and more accessible point of entry to the internet than larger desktop or
laptop PCs, there has consequently been a significant shift in usage to mobile phones worldwide, and noticeably more so in developing countries. Hence a lot of web traffic will be transmitted over GPRS and similar channels.

Fig. 1.2 - Mobile versus desktop Usage

Phone manufacturers have taken note of this and are starting to produce products with cheaper components but near same functionality as the full price phones. This is also accelerating the change to handheld devices.

For many developing countries setting up a powerful large-scale wireless internet system would simply be too expensive and many are reluctant to upgrade from legacy wired networks, which may have been a costly investment also. Thus, development of wireless networks in poorer countries has been slow, despite the large growth of mobile users.

As such, this research effort is motivated to investigate a possible alleviation of the access costs associated with internet access in poorer regions, both mobile and desktop.

This research effort has benefits in the western world also, for reducing costs or maximizing user throughput, for example when multiple users share a weak wireless connection at a crowded café or at a bus station. Amazon and similar e-commerce platforms place significant importance on making sure pages load quickly. Astonishingly, they claim a 1% revenue increase for every 100ms of improvement [5].
1.2 - Research Objectives

The primary objective of this research is to investigate the efficiency of various data reducing techniques and their consequential execution time in order to try and improve webpage loading times on congested/slow networks and thus increase the access to information through these networks. The different approaches explored in this report will be each analysed to investigate whether the amount of data removed is significant enough to warrant further research, whether execution time is detrimental to users’ experience and if a system containing these elements would lead to users being able to browse more efficiently in terms of time and data transfer fees.

In order to demonstrate the applicability and viability of the approaches identified, a prototype system will be designed and developed which implements various experimental techniques. This will be further detailed in Chapter 4.3. This system will be used to try and improve a user’s connection usage and to generate subsequent statistical data which will be used to evaluate the intermediary reduction server concept as one that could be used in the real world.

A secondary goal for this investigation is to release the aforementioned system as open source, so that others can adapt and build upon the source code as there are (as of time of writing) no similar systems that have been released - outside of proprietary companies - that developers can build upon.

1.3 - Technical Approach / Methodology

In order to download a webpage, a user must connect to a webserver located somewhere on the internet. A request is made to download the page and is sent to the local network’s router/modem. This request is forwarded to a users Internet Service Provider (ISP) which then uses a Domain Name Server (DNS) to locate the requested server and the request is passed to the closest machine to this server. This forwarding process continues getting increasingly closer until it - hopefully - finds the target server. It then returns a webpage via the reverse of the route it has just taken. To a normal user however a normal web connection consists of a client to server interaction (C -> S).

![Client <--------?--------> Server]

Fig. 1.3 - Layperson view of the internet

In the approach proposed by this research, in order to reduce the amount of data that is downloaded by the
client, an intermediary server will be responsible for intercepting the response and modifying it so that the amount of data is reduced. Once the data has been minimized as much as possible, the data is then sent to the client.

In some cases, the data will be modified to a form that the browser will not be able to read - In order to restore meaning to the content, a local program is needed to intercept requests and translate them. For example, an image could be compressed using a non-supported format and will need to be decompressed before being displayed.

This new setup is now of the form (C -> LocalProgram -> RemoteProgram -> S). The programs described will take the form of a network proxy. A Proxy is an intermediary between two computers, forwarding on information from one to the other. A common use for a proxy would be to act as a cache in order to relieve a server of load and many ISPs use one for exactly this purpose.

The user will have to run the given local proxy (L) on their machine and point their proxy settings in their browser towards it. This local proxy in turn points to our remote proxy (R), which is hosted on a known server. It is here that the connection finally makes its way out to the desired server to retrieve a webpage.  
(C -> L -> R -> S)

The route on the returning journey is simply the reverse. (S -> R -> L -> C)

```
Browser <-> LProxy <----------------> RProxy <------------------------> WebServer
```

![Fig. 1.4 - Physical setup](image)

Most of the implementation effort will be on the remote server. It is responsible for taking the webpage's content and reducing it down as much as possible. This will be achieved using various exploratory techniques such as optimizing web code, compression of various data formats, converting raster images to vector images, etc.

This minimized content is then sent to the local proxy. The local proxy is responsible for converting any data from a format that a browser would not be able to read into one that it can (e.g. current browsers do not support tar.bz2 compression).

### 1.4 - Overview of Report

The following chapter is a review of previously released research efforts in the area of webpage reduction both by way of an intermediary service and of optimizations that web developers currently implement. This section will also discuss the background of different browsers and relevant web standard developments.
and their subsequent uptake.

The ‘Problem Overview’ chapter is a discussion and justification of the selected techniques, datasets and development decisions that have been chosen for this research project.

The ‘Design and Implementation’ chapter provides a high level overview of the different technologies and researched methods used in the designed system. These will each be outlined in further detail with particular attention to their implementational and theoretical advantages and disadvantages. Gathering and analysis of statistical data will also be described in this section. Any issues in development will be mentioned here and analysed.

Next, The ‘Analysis and Evaluation’ chapter will present the resultant data and derived conclusions. Any deviating data points will be discussed and evaluated considering the design of the system. There will be a subsequent section that will discuss any conclusions that can be drawn from the generated datasets and its impact on our overall objective.

Finally, this report will summarise the findings of this research effort in a ‘Conclusion’ section. This will review both the software and theoretical aspects of the study and also suggest any subsequent research that could be investigated based upon our findings.

A Bibliography is included at the end of the report for further reading and a Glossary for quick recaps on terms used in this report.
Chapter 2.  Background & Related Work

2.1 - Reduction over HTTP

Webpages are transferred between machines via the HyperText Transfer Protocol (HTTP). These messages consist of a header containing meta information about the transfer and the body content containing the data being sent.

HTTP supports four different encoding of messages - GZip, Deflate, Compress and Identity [6]. GZip, Deflate & Compress are all compression algorithms, Deflate being arguably the fastest as it does not include metadata about the encoding [7]. The Identity encoding is simply used to state that the content has not been altered.

These encodings can allow webpage sizes to be reduced by a significant amount. Most modern browsers support the encodings, however the uptake on web servers has been less impressive as can be seen below in Fig 2.1, which graphs the websites of the “Fortune 1000” - America’s most profitable companies [8].

Because two thirds of websites do not have compression enabled, it is quite frequent that more data than is necessary is transmitted. Commercial studies have estimated that uncompressed content causes 99 human years to be wasted every day [9].

![Compression Uptake in 2010](image)

Fig 2.1 - Uptake of compression in the ‘Fortune 1000’
Mozilla’s Firefox now has support for LZMA (Lempel–Ziv–Markov chain algorithm) [10] encoded content and Google has been pushing for their own encodings such as SDCH (Shared Dictionary Compression over HTTP ) [11] to be included in the next version of HTTP. As HTTP 2.0 approaches [12], it is yet to be seen whether there will be many new ways to reduce content via this protocol and if it might be more accessible for server owners to implement.

2.2 - HTML Reduction

As of the time of writing, HTML 5.0 is the newest revision of the HyperText Markup Language (HTML) standard. With it came CSS 3.0 and the introduction of media queries [13]. Media queries allow web designers to style their webpages differently for different devices.

Conditional styles brought forth Responsive Web Design (RWD) in which the layout of a site adapts to suit the user’s screen and/or display type. All layout rules are sent to the client machine, which then apply the rules to suit. These rules mean that a lot of data could be sent which will not be displayed to the user at all. As more and more sites adopt HTML5, users will start to see more and more responsive websites.

HTML content is transferred ‘as-is’, which is to say that files appear on the client exactly as they appear on the local server. This includes comments, excessive whitespace and hidden items which will be unseen in regular browsing. Many HTML tags have ending tags which are not compulsory, but they are usually included anyway by developers as they have complex contextual rules.

Many tech companies that produced web applications do not want to share their code openly with the world, but yet they need to send client executable code such as JavaScript to the user. In order to try and dissuade users from modifying their code, companies employ the use of programs called obfuscators. These programs change the form of the given code into one that is hard to decipher, but still executes correctly.

Input Code:

```javascript
var a = 1 + 1;
console.log(a);
```

Output Code:

```javascript
eval(function(p,a,c,k,e,d){e=function(c){return c;if(!"`.replace(/\^/,.String)(while(c--)[d[c]=k[c]]||c=[function(e){return d[e]};e=function()\{return\\w+\};c=1;while(c--)(if(k[c]){p=p.replace(new RegExp("\"b\"-\"e\"+"\"b\".\"g\",k[c]))return p\)(\"2.3=1+1\",4,4,’console\log[a].split(\'\’),0,\})
```

Example generated by [JsObfuscate.com](http://JsObfuscate.com)

This can add many lines of code to a file and potentially also increase execution time, depending on the algorithms used. This particular example changes a simple logging of the number two, into several lines of incomprehensible JavaScript, employing a variety of obscure techniques that are hard to decrypt without large effort.

There are also some programs that try to reduce the size of web content by similar pattern matching methods.

Input Code:
function toggle(element)
    var attrName = "visibility:hidden";
    if (Dom.hasAttr(element, attrName)){
        Dom.Show(element, attrName);
    } else {
        Dom.Hide(element, attrName);
    }
}

Output Code:
    function toggle(b){var a="visibility:hidden";if(Dom.hasAttr(b,a)){Dom.Show(b,a)}else{Dom.Hide(b,a)};}

Example generated by Refresh-SF.com

The most common techniques in these minimizers are removal of whitespace characters and decreasing the sizes of variable names to be a single character or, if needed, two characters. These small changes can decrease the size of a large JavaScript library significantly. For example, the popular library “jQuery” has a minimized version which is a third of the size of the original (Version 2.0.2).

2.3 - Image Reduction

Images are becoming increasingly prominent on webpages now that Western networks are faster, and subsequently, higher resolution images are being used also. Without compression algorithms, images can be vastly larger than needed.

There are several methods used in the area of Computer Vision to reduce the size of images without causing a noticeable change to the human eye. One of these is to perform a lossy compression on the image.

Lossy compression takes advantage of the fact that the human eye cannot distinguish between minute differences in colours and estimates pixel values to become similar values to their neighbours, allowing for smaller compression ratios. In doing so, some information in the image is lost. With very aggressive lossy compression, ‘artifacts’ can appear - significant sections of an image estimated to be similar to their neighbours, so much so that it is noticeable in the form of square sections that can be seen by the naked eye. Fig 2.2 shows an example of aggressive compression and the subsequent artifacts that appear.

![Fig. 2.2 - Original & heavily artifacted image caused by lossy compression (81KB -> 1KB)](image)

Lossless compression is a technique that preserves the image exactly, but as a consequence has lower
reduction ratios. As it cannot optimize the data it is compressing, there is a higher limit of compression size. Fig. 2.3 shows a visually unchanged image that has had aggressive lossless compression applied. Note that this is a particularly hard image to losslessly compress as there are a lot of colours and variety in the image, images with less variety should experience larger compression ratios as lossless algorithms easily reduce sections that are coloured the same.

![Fig. 2.3 - Original and Lossless Compression (81.3KB -> 80.4KB)](image)

Restricting the amount of colours in the image is another technique used, which results in neighbouring areas of similar colour which can be compressed easily. However, on images with gradually changing colours, there is a stepping effect which can be seen in Fig. 2.4.

![Fig. 2.4 - Example of limiting colour range (4.45MB -> 635KB)](image)

Other techniques include cropping, ignoring every n-th pixel in the image or converting images to greyscale.

Different image formats support these techniques to varying degrees. JPEG images focus on lossy compression, PNGs on colour palettes, BMPs on lossless compression, etc.

There have been several alternative image formats proposed for the web - in HTML 5.0, Scalable Vector Graphics (SVGs) were introduced. Vector images differ from traditional ‘raster’ images in that they contain information about how the image was produced, rather than the colour information for each pixel. As the information details the shapes within the image, the image can be enlarged without any loss of data as can be seen in Fig. 2.5.
In many cases, vector images are also smaller than raster images and are preferable in browsers that support the <svg> HTML tag.

2.4 - Emerging Techniques

Google is one of the foremost innovators focussed on speeding up the web. One suggestion they proposed is an alternative transport protocol for HTTP called SPDY (pronounced Speedy), which is designed to minimize latency over a network by allowing concurrent requests for content - HTTP 1.1 only supports one connection at a time. Initial tests have shown that the target 50% load time speedup is achievable [14]. SPDY is proposed for inclusion in HTTP 2.0.

Another development by Google is the WebP image format, a lossless image format that incorporates several compression types, including LZ77, Predictive Encoding and Google’s own compression algorithm VP8. The images incorporate a “colour cache” which stores recently used colours, so they can be reused later with smaller reference values.

2.5 - Related Work

On the first of September 2009, Opera Software officially released version 10 of their browser featuring its new ‘Turbo’ function [15]. Turbo proposes to compress webpages to be up to 80% smaller than their original size. It accomplishes this by a combination of VP8 and their own compression algorithms [16]. However, the system is closed source and any development outside of Opera Software will have to be written and researched from scratch. This system does not - to the best of the author’s knowledge - modify the actual content of the webpages.
During the course of this research project, Google released a stable version of their own compression proxy [17], which had previously only been available to beta testing users. Google are more open about the techniques they have used, which involve transcoding images to their own WebP format, forcing minimum GZip compression and enabling SPDY if it is available [18]. Google has released optimization libraries that they based some of the system on, however, these libraries are mainly for use by server administrators to speed up individual sites rather than for client-focused compression [19].
Chapter 3. Problem Overview

3.1 - Techniques Explored

As Opera Turbo already existed prior to this project (see section 2.5), the author has scoped this research effort to several areas that have not publicly been investigated in a system of this sort, and some alternatives to existing implementations.

Firstly, this project explores removing surplus content from webpages. This involves removal of whitespace, comments and other inconsequential information that is transferred. Also investigated in this feature is the reduction of JavaScript code via compressors.

GZip, Compress and Deflate are used to compress webpages which are then decompressed by the browser before displaying. Having only three available compression methods, which date from HTTP 1.0, alternative compression methods are explored to investigate a potential successor.

As images take up a large chunk of a page’s data, these are an important target for reduction. If images are not on-screen, there should be no reason that they should be requested from the server. Therefore, the next feature we explore is lazy-loading of images, in which they will only load when the user will be looking at them.

Finally, conversion of raster images to vector images is explored to see whether generated vectorized versions of the images are suitable replacements for their raster counterparts. This will initially be analysed in terms of how well the conversion replicates the visual data, and secondly whether the conversion process reduces the size of the image and if it is worth the computational time.

3.2 - Development Decisions

The main environment used for development in this investigation is Google Chrome Ver.32 on Microsoft Windows 7, although the proxy should be useable with any Windows browser that supports changing of proxy settings. Specific secondary browsers supported are Microsoft Internet Explorer and Mozilla Firefox. A GNU/Linux port of the client proxy is possible with slight modification, although it is outside the scope of this project. The remote proxy is run on a server running Debian 6.0.

A desktop environment was chosen instead of a mobile environment for ease of development. Many techniques would have been much harder to implement working within the constraints of mobile application systems and added a significant amount of time pressure.

It was possible to somewhat emulate a mobile browser for the content removal feature by resizing the desktop browser to the same dimensions as the target mobile device. However, this is not a complete solution as there are several websites that display differently based on operating system and not just screensize.

To port the existing system to mobile would simply require a reimplementation of the local proxy, all transforms performed on the remote server would remain the same, and the customization chrome extension could be replicated by navigating to the url directly.
The lazy-loading section of this research is specific to Chrome browsers only. This is because it is outside the context of the remote proxy and requires scripts to be injected into the page to inspect the position of the image on the screen, which is not available on all browsers. It takes the form of a Chrome Extension (Version 2). This will be detailed further in section 4.2.
Chapter 4 .  Design and Implementation

4.1 - Requirements

The requirements for this research project are based on creating a proxy that reduces the download size of requested webpages and, secondly, forays into experimental techniques which have, as of time of writing and to the author’s knowledge, been unexplored. Specifically, the requirements are:

➢ Develop a proxy system to route traffic through, which allows modification of downloaded content.
➢ Implement unnecessary content removal on downloaded HTML and associated files.
➢ Enforce compression for all downloaded content transfers.
➢ Develop an extension for one selected browser, which only allows requests for images that are in the current view of the user.
➢ Design and implement a process to automate vectorizing raster images ‘on-the-fly’.
➢ Support major browsers, unless otherwise detailed.
➢ Allow users to selectively enable and disable components of the system for their specific needs

4.2 - Issues affecting design

As there are hundreds of browsers and Operating System combinations, it is unfeasible to attempt to guarantee support for every mixture. Hence, the browsers that the majority of users utilise will be checked for compatibility, but obscure browsers will not.

Another feasibility issue is creating a lazy-loading extension for multiple browsers. Google Chrome provides methods in the documentation for browser extensions that allow developers to modify requests and their associated headers. However, these methods are specific to Chrome and other browsers either do not support the interception of requests, or have completely different syntactical and semantic rules, meaning code is not as portable. As this feature is of an investigative nature, it is not necessarily a requirement for it to be cross-browser compatible.

4.3 - Target User Model

This system is aimed at users with poor or reduced network bandwidth, and thus, poor download speeds. As such, the time saved by using the system must be significantly greater than the time taken to generate the reduced content. However, it should also be noted that the system is designed to aid future development in this area. Therefore, emphasis is placed on the functional features of the system, rather than on aesthetics or ease of use. Subsequent researchers are also target users of this system and it is important that the system is documented informatively, that it is easily modified and that the system is thoroughly assessed.
4.4 - Data Storage

The remote proxy is the nexus of the system, and hence is responsible for much of the logging generation. Primarily, it stores the difference in sizes between received and sent content. This file will be used for analysis in the next chapter, Analysis and Evaluation. Each type of content has individual statistics, which will mean that process effects can be analysed for their suitability of application to that particular content.

4.5 - Complete System Overview

The following diagram represents the designed system as a whole and its various components. Each of the following sections will discuss specific parts of the above system focussing on their design, integration and functionality.
4.5.0 - Message Path

This section describes the path that messages take in a normal web transaction, based on Fig. 4.1 - Paths mentioned are shown in the diagram with a corresponding reference number along their route.

When a request is sent from the client for a webpage component such as the HTML, an image or similar,
the request is first sent through the local proxy [explained further in Section 4.5.3] (Path #1). The message then leaves the client’s PC and is sent directly to the remote proxy server (Path #2). The remote server then routes the message to the target webserver (Path #3). The overall request path is represented as green arrows in the given diagram.

Once a response is formulated by the target webserver, it sends it to the remote proxy (Path #4), as from its point of view, it was the one that originally requested the data. After receiving the response, the remote proxy server will perform some operations on the data and issue its own response to the client (Path #5). The client picks up the message in the local proxy first, which then forwards on the message to the browser (Path #6), where the requested data will be displayed. The response path can be followed by the representative red arrows in the diagram.

4.5.1 - Lazy Image Loader

The most logical way of reducing the amount of data that is downloaded is to request less data, so it does not have to be downloaded at all. This feature takes the form of a Chrome extension that runs a JavaScript script on every page in the background.

The Chrome extension library lets programs perform tasks when web elements are requested. The lazy image loading script interrupts any requests for images to be downloaded and checks if they would be visible to the users if they were loaded at this stage. Any requests for images that would not be visible are cancelled.

An additional script is used to manually request images after they have been cancelled. After a short sleep duration, unretrieved images have their url modified - adding a ‘?’ character and subsequently removing it - in order to trigger an automatic re-request of that image.

The extension has only been implemented on the Chrome browser as a proof of concept, but libraries such as GreaseMonkey or Scriptish that exist on several browsers, may be able to support lazy loading in the future through their recent addition of @run-at directives.

Performance information about this feature can be gathered from the console, represented in the overall system Fig.4.1 as ‘A’. Writing to client-side files with JavaScript is not allowed on almost every browser, because of security concerns.

Fig 4.2 shows an example of the lazy loader in action. The first image is how the site looks without this feature enabled. The red box represents what the viewer can see. Many images are loaded in that may not be looked at if the user does not scroll down.

The second image shows the same user positioning, but with the lazy loader enabled. Many image requests have been blocked. Note that the one remaining image is not in fact an image, but an embedded video.

Finally, the third image shows how images are loaded in once the user has scrolled down. Notice that if we were to leave the website at this stage, we would still save on four image requests.
4.5.2 - Customization Extension

To acquire information about the user’s browser a JavaScript function is run to find out several characteristic features of the viewing platform. This includes information like browser name and version, screen size and other meta information.

This script is run in a Chrome extension, which is also used to let the user provide a specific level of reduction that the proxy will implement. This webpage is hosted on our remote webserver using Flask [Explained further in 4.5.4]. The fact that this feature displays another is represented on the system overview by a “Displays” note beside the dashed path. As well as a reduction level, the extension also asks the user for an ID, which will uniquely identify the browser amongst all the other users of our proxy and allow the proxy to customize its settings for each connection.

Every submission will update the current settings for the user, so if a user switches to Firefox, resizes their browser and decides they would like the most reduction possible, the database entry [See 4.5.4] will change and all future queries will take this into account.
4.5.3 - Local Proxy

The local proxy is written in Node.js and is responsible for the transmission between the browser and the remote proxy server. The local proxy attaches another field to the header of every sent request. This field, named “fyp-id”, contains the id that uniquely identifies the user’s browser. This id is generated on launch of the proxy, and any subsequent launches will use this id unless the id file is deleted.

There was an attempt to incorporate multiple types of compression on the different data transmitted by the remote proxy to see which one performed the best. However, after receiving the compressed data, the local proxy was unable to decompress it because the content seemed to have been modified in the transmission process. Unfortunately, this issue was not able to be resolved during the course of the project.

4.5.4 - Flask Web Server

The Chrome extension for customization [see 4.5.2.1 above] is a container for a webpage. This webpage is hosted on our remote server via Flask - a lightweight Python web framework. On submission of the form, the values of the form elements and the results of the JavaScript script are sent to the backend, which enters these values in a SQLite3 relational database using the id value as its primary key.

<table>
<thead>
<tr>
<th>ID</th>
<th>Width</th>
<th>Height</th>
<th>Browser</th>
<th>Version</th>
<th>C_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>29223548039</td>
<td>1366</td>
<td>728</td>
<td>Chrome</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>1666061719</td>
<td>1219</td>
<td>704</td>
<td>Chrome</td>
<td>33</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig 4.4 - Example SQLite3 Database entries
4.5.5 - Remote Proxy

The modifications made by the remote proxy are controlled by an external script to the program MITMProxy. MITMProxy handles all transmission of data and gives intercepting access to the external script. External scripts can modify what the proxy does upon receiving a request, and before delivering a response. Performance information of each feature performed here can be gathered at this stage, as represented by “B” in the System Overview diagram.

When the remote proxy receives a request from the local proxy, it removes the additional header from the incoming request and attaches it to the MITMProxy ‘flow’ object, which persists between a request and it’s subsequent response. This is so that external webservers do not detect the additional header field and reject transmissions, but the remote proxy will have knowledge of the id when a corresponding response is received.

4.5.5.1 - Intelligent Content Removal

As explained in Section 3.2, one of the modifications the remote proxy will make is to remove content that do not have any effect on the user’s view of a webpage. For this, a combination of several regex patterns and string manipulation methods were used to strip out unnecessary content.

After querying our database for the details of our browser (with the primary key taken from our ‘flow’ object as seen above), the proxy strips out the media queries that are not relevant to our target browser. All queries are parsed and conditions are evaluated, considering that media queries can consist of multiple conditions also e.g. “@media screen and min-device-height”. If conditions are not met, the enclosed css rules are removed, otherwise they are left as they are.

We also remove browser specific css properties. For example, Mozilla browsers use -moz-<attribute> attributes and webkit-based browsers such as Chrome or Safari use -webkit-<attribute> attributes.

An attempt was made to integrate Yahoo’s YUIcompressor[31], but it failed to work on programs that produced warnings or made use of some particular external libraries. More often than not, YUIcompressor broke webpage interaction or severely hampered user experience. Instead of proceeding with this effort, YUIcompressor was excluded from the project.

Regex patterns are then applied to strip semantically meaningless whitespace from the webpage. At most one whitespace character will be shown onscreen, and if developers want to display any more, they will use HTML-escaped characters like &nbsp; (Non-Breaking SPace).

Various types of comments are also stripped from the webpage. There are some differences with HTML comments as they can be used to run browser-specific code such as Internet Explorer. The proxy will not strip the contents of an exclusive comment block if the database query matches up with an Internet Explorer browser and the version specified.
/*
This is a multi-line comment
The code below calculates the volume of a cone
*/
function(){
    console.log("Hello");
}
//Single line comment.
//This function calculates the area of a cone

Fig 4.5 - JavaScript & CSS styled comments

<!-- This is a multiline comment -->
Fig 4.6 - HTML styled comments

<!--[if IE 6]>This is a multiline conditional comment <![endif]-->
Fig 4.7 - Internet Explorer-Specific HTML comments

4.5.5.2 - Webpage Compression

As seen in Fig. 2.1, webpage compression is something that is still not fully adopted by most sites. The reverse proxy forces a minimum of GZip compression for every transmission possible. This is done by encrypting any messages that pass through the proxy with GZip compression and changing the Content-Encoding header information of that transmission to reflect the change.

Using the Python Image Library (PIL), images were also able to be compressed. As an additional test, the images were resized to half the size and enlarged again. This reduced the data by a large amount as the new pixels created when enlarged were similar to their neighbours and hence easily optimized by PIL’s compressor.

4.5.5.3 - Raster to Vector Conversion

One experimental technique that was tested was converting raster images into vector images as hand-coded vector images can be significantly smaller. as can be seen in the below image, Fig 4.8 .
Obviously, hand-coded SVGs are not feasible for a realtime system. A series of steps were used to perform the transformation as an automated process:

Firstly, a k-means-clustering operation[32] is performed on the image. K-means clustering is a technique to approximate the amount of colours a image has, while staying as close as possible to the overall colours of the image. All colours on an image are put in a ‘feature space’ where it is possible to group similar colours by proximity. The amount of clusters in the system’s implementation is a set value, but can be changed for more reduction or accuracy.
Fig. 4.10 - Example feature space with clustering

These clusters determine the new colour of the contained points. The colour that is chosen is based on a weighted average dependent on locality in feature space and quantity of colours in that space.

The next operation performed was a conversion to several binary images. By saving the different resultant colours of the K-means clustering transform, the program is able to use each colour as a different threshold for the binary image.

Using these thresholded binary images, we apply OpenCV's (a Computer Vision library) Contour function, which extracts points which contain the boundary of a shape[33].
These points are then converted into SVG paths using a custom function converting the outputted Numpy array (a data-type representing multi-dimensional arrays provided by the Numpy scientific python library). The constructed SVG paths take the form `<path d="M x1 y1 L x2 y2 z"/>` where ‘M’ indicates that the following coordinate is to be moved to (without drawing), ‘L’ indicates that the next point is to be drawn to from the current point, and ‘z’ representing the end of the drawing path.

The colour of the border and the colour inside of the generated path is set to the colour that was originally extracted from the k-means clustering that got the binary image associated with that path. The paths are put into the surrounding `<svg> </svg>` tags backwards, as if they were put in forwards, some sections would be covered up because of concavities in the binary images creating areas that lie inside others.
The newly generated image then replaces the old image and is sent to the client.

This feature is by far the most computationally expensive of the system. Because of this several optimizations were made in order to reduce the time for conversion. While twenty seconds does not sound like an overly long time for a conversion to run, in the context of the web it’s very slow as users expect images quickly. This computational time threatened to nullify the time saved by reducing the transfer speed.

The first optimization made was to reduce the k-means clustering iterations performed on the image. With more iterations the algorithm gains increased accuracy, but naturally more computing time. Reducing this to a single iteration reduced the operation time significantly.

Using the default Contour algorithm flags without further investigation was causing the resultant image to be vastly larger than the original. After much investigation into this issue without knowing the cause, it was finally tracked down and then changed to the new option. This simple flag change meant that instead of creating coordinates for every point on the calculated path, only the points that are at the end of a straight line are output [20].

Whilst previous optimizations made image conversion execute at a tolerable speed, if multiple images needed to be loaded, the process becomes quite slow and inconvenient. Therefore the larger images were ignored and only images below a certain threshold were converted.
4.5.6 - Unknown Webserver

The unknown web servers that we connect to are simply externally owned servers that host websites for clients to access through the Internet. The HTTP protocol that is used to transfer webpages from the web servers to the clients is outlined in Section 2.1.

Nearly all webpages consist of a combination of JavaScript, HTML and CSS. The backend logic of websites may be coded in many languages, ranging from Java to PHP.

4.5.7 - Evaluation Software

To provide informative evaluation of the techniques used, Google Sheets - a spreadsheet web application - was used. The generated dataset was imported into the app and spreadsheet formulae were applied to manipulate the data into the desired forms. Google Sheets also provide tools for visualizing data and these were used to produce the graphs in the following chapter.
Chapter 5. Analysis & Evaluation

5.1 - Statistical Tests

The most important figures to evaluate with this system are the resultant data sizes after each feature has run. Ideally, the processed data will have a reduced size, or at least, no increased size. Much of this data was gathered during user tests and casual personal use, generating a large data set.

5.1.1 - Final Statistical Data Sets

Two different datasets were generated by the system. Firstly, the remote proxy server generated logs of all traffic that passed through the server and their relevant transformations. These logs were written to a file and then imported into Google Sheets for analysis. The large amount of data logged meant that the chance of statistical errors was reduced.

<table>
<thead>
<tr>
<th>Type</th>
<th>Original Size</th>
<th>Modified Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>GZIP [application/json]</td>
<td>152367</td>
<td>32284</td>
</tr>
</tbody>
</table>

**Fig 5.1** - Example dataset entry for a JSON transfer compressed with GZIP

The Lazy Loader is run in-browser, and cannot write to the file system for built-in security purposes. Instead, the generated output is shown in the console trace and then inserted later into Google Sheets manually. This file writing restriction unfortunately meant that less entries were able to be recorded.

One statistic that might have been useful to gather information about would be the size of the images that were avoided. However, the system can not detect how big the images are unless it downloads the image itself, defeating the purpose of the feature. Another possibility would be to use the Content-Length Header tag, but this is not always included in transfers and may not be strictly correct.

A problem with generating output for this feature is that it is quite hard to detect when a user is finished with a website. The output in this system is produced when the system detects a website switch, but there were no facilities to detect when windows or tabs were closed, so some webpage statistics are never produced. Site switching does however, allow us to generate a satisfactory amount of data.
5.1.1 - Intelligent Content Removal

This feature was evaluated twice, once with normal desktop settings - Fig. 5.2, and once with emulated mobile conditions - Fig. 5.3.

For desktop browsing, there is not a significant decrease in size for HTML or CSS pages. This will majorly consist of whitespace and comment removal as media queries are, on many RWD sites, written "mobile first" [21], meaning that additional rules for display are added as screen sizes get larger.

![Content Removal](image)

**Fig 5.2** - Content removal on a traditional desktop PC [1911227 -> 1853583]

This design pattern proves to have some evidence behind it as can be seen in Fig. 5.3 when the feature was tested with an emulated typical mobile device size - iPhone 3, Samsung Galaxy, HTC Butterfly, etc.

When run on a smaller screen, the intelligent content removal feature can remove CSS rules that are specific to larger screens. As more companies increasingly adopt RWD practises, it is possible that the 13% removal observed in the above graph could be sizably increased in coming years.
Fig 5.3 - Emulated content removal on a 320x480 mobile device [11884123 -> 10336269]

Some websites will not benefit as much from this feature as a moderate amount of websites reduce their content at the server level, so whitespace and comment removal becomes redundant. This is mostly only seen in larger corporations, however.

One issue that occurred with this data was whilst browsing some of Trinity College Dublin’s websites, some sites contained a browser hack that only worked for Netscape 1.0 which was released in 1995. This hack involved inserting HTML comments into JavaScript so that netscape would execute the code, but not Internet Explorer. Several browsers have been released since then, which ignore any HTML style comments inside JavaScript. Unfortunately this system naively removes these comments and breaks JavaScript functionality, but this was the only site I encountered that had this problem, in all duration of the project.
5.1.2 - Forced GZip Compression

When implemented on all content, the resultant graph was quite disappointing.

Breaking down the figures by content type, it can be seen that the items with the lowest compression ratios were media files such as MP4 or Shockwave Flash. Looking at individual items, this was not always true and there were a few exceptions which benefitted from the compression. Stripping out media files and focusing on text-based data, the new graph can be seen below:

---

**Fig 5.4** - The effects of GZip compression [ 74196661 -> 71861470 ]

**Fig 5.5** - The effects of GZip compression on textual content [ 1041739 -> 338318 ]
5.1.3 - Raster to Vector Conversion

This feature was hard to evaluate. As can be seen in Fig. 5.6, it reduces data on some images, and not on others. This may be dependant on the complexity of the image. Without GZip compression, the images are slightly larger than their original raster counterparts, but as discovered in section 5.1.2, Gzip works remarkably well on textual content, leading to the significant drop in size we can see later on in the graph as indicated by the red lines.

![Example Vectorization Session](image.png)

Fig 5.6 - A comparison of sizes between raster images and vectors.

5.1.4 - Lazy Loading

The data for the lazy loading feature is quite erratic. This is because websites differ greatly in length and placing of content. Fig 5.6 shows the savings of one session, and it can be seen that some periods experience no improvement at all, whereas other periods save items regularly.
The following image (Fig. 5.8), shows these figures in a form that is easier to see the distribution of removal counts. The majority of sites are unaffected by this process, but there are still a considerable amount of requests removed, which is important as images take up a significant percentage of web traffic.

**Fig. 5.8 - Alternate view of saved requests**

This feature will have more impact on longer sites, and potentially more with smaller screens. This makes it an ideal concept for integration into a mobile browser.
5.2 - User Tests

As this research is targeted at improving data costs for the general public, it was decided that end-user evaluation should be considered. Aside from statistical merit, the system also has to be functional and minimally impactful on the user. User tests were chosen for this part of the evaluation instead of statistical criteria as they provide much more information about the system and are reflective of what the real world implications would be of this system. After gaining approval from the relevant university bodies, user trials were held to validate each technique.

Whilst a simulated congested network would have been ideal, there were complexities in setting this up. If implemented on the remote server, transfers were cancelled and sent with default transmission settings when a stall occurred. Javascript’s asynchronous nature did not allow for development on the local proxy. Instead users were given a detailed description of the context of the experiment and asked to integrate this into their marking of the experiment.

5.2.1 - Format of User Tests

Interested participants were given information sheets describing the project, its aims and the data protection policy of the experiment. The experiment and its context was also verbally relayed to the participants. If the participant decided to continue with the experiment they were given a consent form to sign confirming they had been informed of the research experiment and what it entailed.

The experiment took the form of ten two-minute long sessions. In these supervised sessions participants browsed the internet with differing sections of the system enabled. There were also some unmodified periods during the experiment in order to identify any intentional or unconscious bias.

The first session was an unmodified session, so that participants could get an idea for the base speed of the connection and use it as a comparison for the following sessions.

Participants were given three areas to evaluate the session by: Speed of browsing, functionality of webpages and their overall experience of the session. There was also an additional comment box for anything of note that happened during the experiment.

Each area was to be graded on a scale as follows: Unusable, Inconvenient, Noticeable, No Difference, Unsure. The grading scheme was based on a Likert Scale, without the positive points, as working on a very fast connection it was doubtful any technique would noticeably improve the connection.

5.2.2 - Results

From the results of the user tests, some features can definitely be confirmed to be non-impactful on a potential user. Fig 5.8 shows the feedback from users with GZip applied to all pages. Apart from a single user, no participants detected any impact from this feature.
After enabling image compression additionally to the system, several users found the blurred images frustrating when looking at images consisting of small components such as text. Again, no difference was found in the speed of the websites, and some participants wrote that they thought the internet might have sped up slightly. This is discussed further in Section 5.2.3.

After enabling content removal, no additional problems were found in functionality, but one user felt that the blurred images were impacting their experience more and marked both sections as Noticeable. We can verify from this that content-removal is a transparent transformation, like GZip.
Fig. 5.11 - User feedback for forced GZip, content-removal and image compression

The Lazy Loader seemed to be a positive feature according to feedback. Some users complained about blurry images that were still left in the cache of the browser, but even ignoring this most users felt that the system was no different to the original unmodified session.

Fig. 5.12 - User feedback for the Lazy Loader

The vectorization feature got mixed results. Participants browsing sites with many images felt that the load times were far too significant to be useful for quick browsing. Those with smaller or less images mostly noticed the changes, but felt that the changes did not affect them too much.
Fig. 5.13 - User feedback for raster to vector conversion

One problem that occurred during the user tests, was that the new vector images did not resize correctly when sizes were directly specified in the HTML. Instead they displayed as normal but with any pixels outside the specifications hidden. This is a problem with the browser’s separate interpretation of vector images to raster images, as it worked in the same way when using an original SVG image. This problem did not occur when sizes were imposed by the css or by other means.

Example: <img src="image.png" height="40" width="50" />

5.2.3 - Notes about the process

Most of the participants were technically minded and hence prone to browsing similar sites. Many participants were also known to the investigator prior to doing this experiment and could potentially hold some bias towards rating of the sessions.

Users had differing concepts of what constituted each of the rating levels, despite the investigator trying to give a solid context to the experiment. Some users were quite analytical and went about testing each of the features, whereas other users only evaluated after the session was complete.

Users appeared to have different browsing styles - some users erratically switched sites to find something to browse, whereas other users went directly to sites, browsed through them and then went onto another target site.

Some users browsed websites that were not affected by enabled sections of the system. For example, some users used sites like YouTube or Vimeo to watch videos, but this meant that the system was not being utilized as much as desired, so the investigator requested that participants avoid video streaming sites. Chrome extensions do not work on the default search results of Google by design, so users were also encouraged to visit websites rather than reading summaries of search results.
Chapter 6. Conclusion

6.1 - Software Feature Evaluation

The objective of this project was to investigate possible reductions of webpage download sizes in order to reduce access costs for accessing the Internet in low speed networks. The system that was produced consists of several techniques designed to meet this objective. These techniques resulted in differing levels of success.

6.1.1 - Vectorization

Whilst the concept behind vectorizing images seems logical, the automatization of this process seems to not be nearly as effective as hand-coded images such as seen in Fig. 4.8. It also seems to work best with simplistic images, whereas the internet contains many photos and natural imagery. Items such as the stepping effect in Fig 6.1 below, were known beforehand, but not expected to be as impactful as they were.

![Fig. 6.1 - Stepping effect of k-means on natural imagery](image)

After analysing the resultant images, data sizes are slightly larger (averaging about 125% of the original image), but once GZip compression is applied, these sizes decrease significantly (GZip works very well on textual input, see Fig 5.2). However, there are several cases where the processed image is larger than the original image. For proper inclusion in a system like this, it would be important that it is contextually, rather than consistently, applied to utilize the reductions to their full potential.

The algorithm for this feature could be built upon. Occasionally important features are removed because there is such emphasis on colour separation, see Fig. 6.2 where my chin and mouth are removed as it is very similar in colour to my chest. This is a particularly bad image for the system as the muddy colours are all quite similar causing a lot of stepping and unwanted effects.
There is the possibility of using alternative algorithms in the future to improve the transformation quality and data size, but with limited execution time, the possibilities for a perfect solution are small.

In the user trials, the people who responded best to the vectorization changes were browsing websites with smaller images, as they paid them less attention and only needed to glance at them to get their meaning.

6.1.2 - Forced GZip Compression
The results from the statistical analysis clearly show that applying text-based compression is a good idea. There are some file types that do not benefit from the process though, so perhaps further conditional compression could be implemented if it does reduce the content.

From feedback in the user tests, users did not seem to notice it being enabled, so conditional compression could be a viable approach for future investigation.

The statistical rates of reduction are quite high on textual content and it could be conceivable that this area is where Opera Turbo and the Google Chrome proxies retrieve their high data reduction ratios from.

6.1.3 - Intelligent Content Removal
The amount of data that is removed is not as significant as was hoped for, but as was mentioned in 5.1.1, responsive web design is a relatively new concept, first proposed in 2010 by Ethan Marcotte [22]. As this design pattern becomes standard, more companies will move their sites to this format, increasing the amount of data reduction we can make. If responsive web design remains the same as it is now, it is debatable whether it is worth including or not. However user tests have shown that the application of the transformation is transparent and hence there is no reason to not include it until proven otherwise.

6.1.4 - Lazy Loader
Users did not notice the lazy loading impacting on their browsing performance at all. It was hard to evaluate this feature, but clearly a substantial amount of requests are being denied to warrant inclusion in
the system. Long websites that take up several screen-lengths benefit the most from this feature as can be seen in Fig 5.7, where some sites avoiding loading up to 17 images. This feature could be expanded out to other media types or to iframes - which are a HTML document element which allows webpages to be embedded into a parent webpage.

6.2 - Overall Evaluation

Several data-reducing techniques were investigated during the course of this research as per the primary objective detailed in Section 1.2. These techniques were analysed for the size reductions they effected and the impact on users in terms of load times and changes to website functionality. Most of the experimental features were confirmed to be of benefit to congested or slow networks, in differing proportions.

This system has been released as open-source under an MIT License¹, so that future development and research can build upon this existing work without having to reimplement features from scratch.

The impact of this research gives a potential alleviation to services in these networks, meaning that access costs and cost-per-site values can be reduced. Hopefully this project can give some insight into alleviation possibilities for those considering implementing such a system at a larger scale.

6.3 - Ethical Concerns

An issue that should not go unmentioned is that potentially, a proxy can be dangerous to a user’s privacy in the wrong hands. As all content goes through the server, the host could potentially log all information about a client.

In this research, metadata is gathered for the purposes of analysis. Only the type of transmission and the size of transfers are logged. However, the content of webpages are visible to the program while being run. For example, this is how the intelligent removal feature in Section 4.5.5.1 operates. As this information is available to the program, it is possible to log and gather data about the user. This includes any personal or sensitive information the user may not wish to disclose to third parties, for example browsing medical condition symptoms.

It is also possible to force advertisements on users by injecting content into a webpage that passes through the system. These unsolicited advertisements could be tailored towards the user by the information that the proxy-owner has collected about them.

As MITMproxy - short for Man-In-The-Middle Proxy - can also operate on data transferred by HTTPS, seemingly secure connections may be intercepted and modified. This is perilous when dealing with an important and critical transaction, such as purchasing items from an online store.

The system also makes it possible to detect specific webpages and replace them with it's own content. This

¹ https://bitbucket.org/mrpineapple/webpage-size-reduction-proxy/src/
is touched on by the existing system. If a user has not registered themselves with the system, all requests are replaced with a webpage requesting they do so. A malicious version of this could be replying with replica websites which post to their own server instead of the desired one, and hence gather passwords and credit card details.

From the points highlighted above, it is imperative that users are knowledgeable about any proxy servers they connect to and are able to make a judgement to trust the service or not.

### 6.4 - Further Research

As the web is ever increasing in size, complexity and ubiquity, there is significant interest in research such as this. Investigation is not limited to academia either, several of the tech giants such as Google, Opera and Mozilla are all making large forays into speeding up the web, both for developed and developing countries.

The prototype system developed by this research could be expanded to include many new features and expand on existing ones.

As previously mentioned in section 6.1.2, the vector conversion algorithm could be optimized to address some of the issues encountered and to also speed up conversion time. One aspect of SVG paths that was not taken advantage of were “arc components” - curved sections that could potentially be used to encode complex shapes.

Opera Turbo & Google Chrome’s compression proxies are both integrated into the browser, toggleable at the touch of a button. If the local proxy script was run at launch for some open source browser, this would reduce the user effort in a similar way.

The success of the lazy loading feature warrants its expansion to other media types such as video, iframes and other content of definite size.

The browsing experience on a mobile device is quite different to that of an emulated desktop version - involving touch gestures and other individual characteristics unique to each device. Hence, a desktop emulated version of the mobile lazy loader would be of little academic benefit as replicating gesture interaction and other characteristics of mobile environments is not easily achieved. It also suffers from the implementation difficulties mentioned in Section 3.3. It would be useful to port this feature to a mobile browser as part of a further research effort and measure its performance against the existing desktop version.

Further investigation could go into different compression algorithms as were attempted in this project, but dropped as a feature because of technical issues. Conditional application of features could be explored also when found to apply well in certain situations, such as the application of GZip to textual content, as discovered in Section 5.1.2

One data reduction technique that the researcher came across during the project was the transformation of animated .gif files into HTML5 videos. This technique claims to reduce file sizes to 16 times smaller than
the original size [23]. Unfortunately, this was late into the project, but would be a very fitting component of the system if someone were to expand upon it.

It would be beneficial to do larger-scale user tests in order to make sure responses are accurate. It might make sense to include the excluded items of the Likert scale as some respondents felt the system was occasionally faster that the original session, and on reflection, as a perception experiment responses shouldn’t be assumed. It would be useful to have this larger survey happen on a low-latency connection.

This research focused on desktop development as it was more achievable in the given timeframe however, it would certainly be of value to deploy and evaluate the approach in a mobile context as different user trial and statistical information may be observed and potentially improve the results of this study.
Appendix

Amazon: currently the largest e-commerce retailer
Browser: A program to retrieve, view and navigate webpages
Client: The computer which accesses a webpage.
CSS: “Cascading Style Sheets” - Documents that instruct how a browser interprets the components of a webpage visually. Example: a red background, an image that is always in the top right hand corner regardless of scrolling position.
Chrome: Google’s Internet browser
Chrome Extension: plug-in files for the Chrome browser that allow a variety of scripts to be run. Also come in the form of webpage popups.
Flask: a web framework for Python that connects visual components to backend processes. Able to perform operations on data from the client’s end on the server. Example: storing form info in a database.
Firefox: Mozilla’s Internet browser
GreaseMonkey/Scriptish: Third party extensions that allow more functionality to browser extensions than originally provided.
GNU/Linux: a unix-like operating system
HTTP: “Hypertext Transfer Protocol” - the protocol in which HTML and other web content is sent over to deliver a webpage
HTTPS: “Hypertext Transfer Protocol over Secure Socket Layer” - A more secure version of HTTP involving encrypting a connection
HTML: “HyperText Markup Language” - Documents that describe the content of a webpage, including structural semantics, images and inclusion of scripts such as CSS and JavaScript
IFrame: A HTML element that allows embedding of one webpage into another.
Internet Explorer: Microsoft’s Internet browser
JavaScript: A high level programming language run in the browser that can interact with the client and alter the current webpage. It has various common libraries like AJAX and Jquery, that build upon it’s functionality.
JSON: “JavaScript Object Notation” - JavaScript objects described as key-value pairs textually.
K-Means Clustering: a technique to quantise the amount of colours a image has, while staying as close as possible to the overall colours of the image.
Lossy Compression: a type of data reduction that discards parts of the data in the process
Lossless Compression: a type of data reduction that preserves the integrity of the data
LZ77: Dictionary encoding compression algorithm used in PNG and the basis for many other lossless algorithms
Node.js: JavaScript that can be run outside the browser and is increasingly common to be used as a server backend.
Opera: Opera Software’s Internet browser. Has a smaller market share than the others mentioned at time of writing.
Python: A high-level programming language with inspirations from functional languages such as Haskell. Instead of traditional brackets to denote code sections, whitespace is used instead.
PC: Personal Computer
PNG: “Portable Network Graphics” - an image format that uses lossless compression
Raster Image: A type of image format that holds information about each pixel.
Regex: an abbreviation of “Regular Expressions”. Used to search through strings based on pattern
matching. A implementation of Regex parsing is built into the core libraries of Python 2.7

**Router/Modem:** A device that forwards network traffic to the next stage in the connection

**RWD:** “Responsive Web Design” - A design approach for the web which adapts to the user’s screen based on gathered information about screen sizes. Sometimes referred to as Adaptive Web Design, or AWD.

**Server:** The computer which provides webpages

**Stepping Effect:** When a gradual change in colour in an image is limited in colour, segmentation occurs creating edges, where previously there was none visible to the human eye.

**SDCH:** “Shared Dictionary Compression over HTTP” - an algorithm used to reduce data sizes before and after transmission

**Shockwave Flash:** A file format used to contain vector graphics, ActionScript and other multimedia

**SPDY:** A proposed networking protocol which tries to improve webpage loading times and webpage security

**SQLite3:** A popular database system which stores databases in single cross-platform files

**SVG:** “Scalable Vector Graphic” - an XML based image vector image. Displayable in HTML5

**Vector Image:** A type of image format that contains information about how an image should be drawn via coordinates and known shapes.

**VP8:** A video-compression algorithm created by Google to replace Shockwave Flash players.
Bibliography


Image Attributions

If not attributed, image is either my own property or a generated graph from Google Spreadsheets or a diagram from Draw.io.

Attached CD with System code

Alternatively, clone the repository:
https://bitbucket.org/mrpineapple/webpage-size-reduction-proxy

Installation instructions are included in the README.txt file or in the repository’s Wiki.