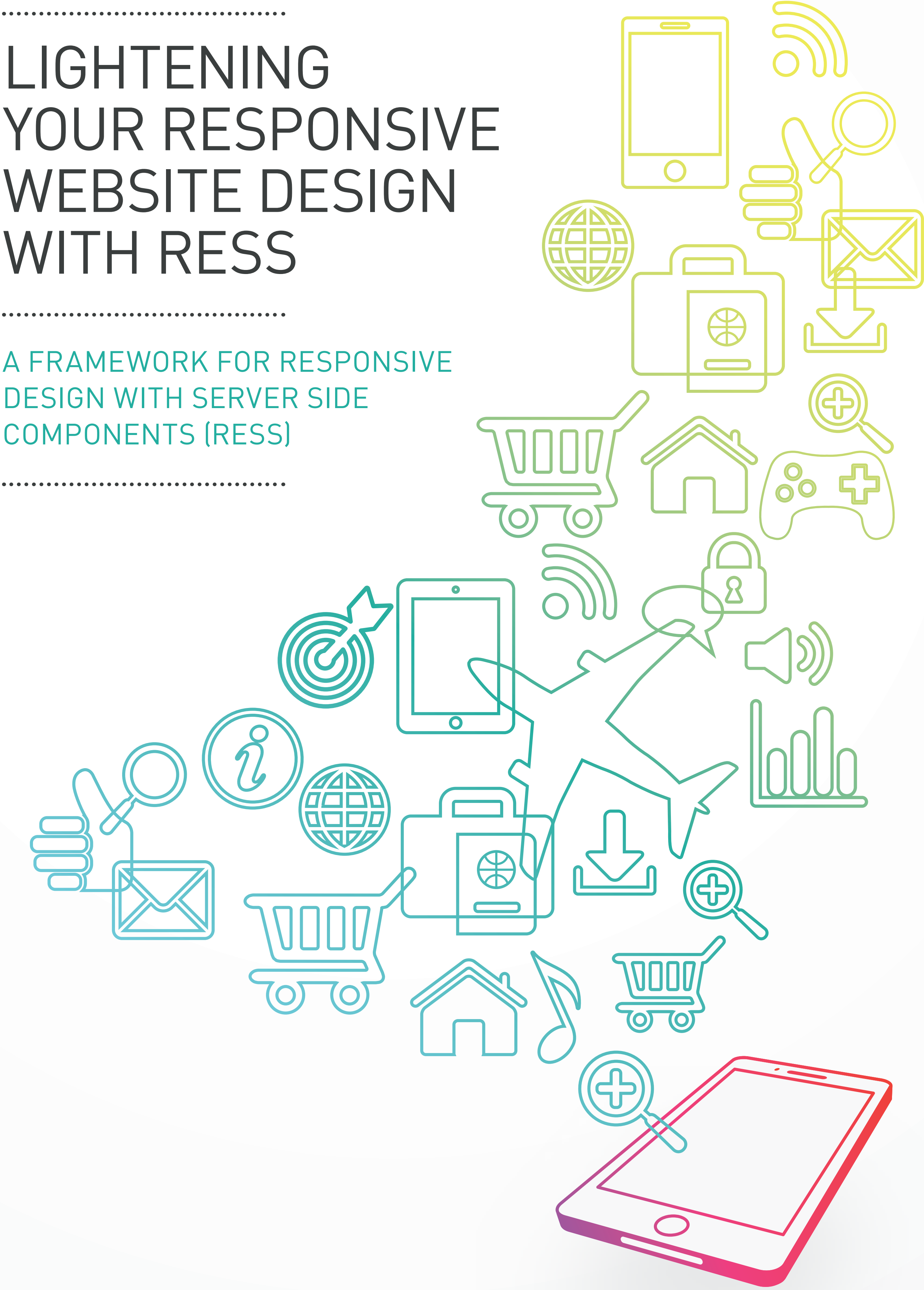
dotMobi™

LIGHTENING YOUR RESPONSIVE WEBSITE DESIGN WITH RESS

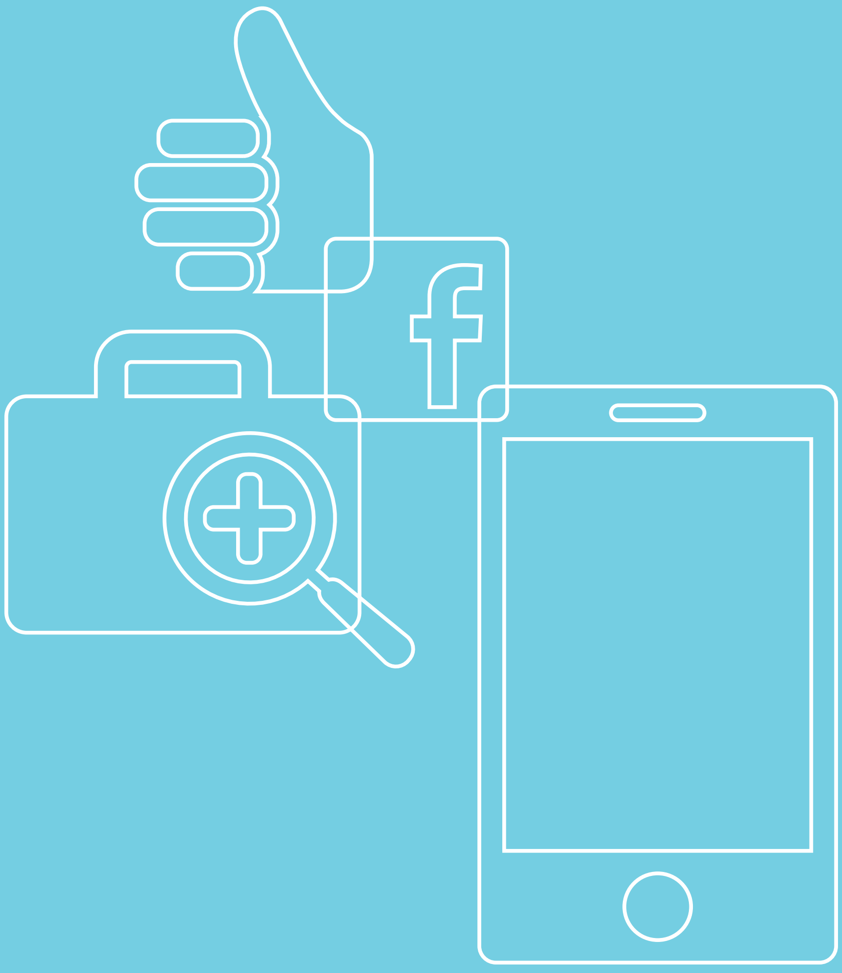
A FRAMEWORK FOR RESPONSIVE DESIGN WITH SERVER SIDE COMPONENTS (RESS)



DeviceAtlas™

CONTENTS

RESS



INTRODUCTION

Responsive Web Design (RWD) has made a huge impact as an approach to designing for mobile. It has also caused a lot of debate: on the plus side, it allows for one codebase to cater to devices with a range of different screen sizes. On the debit side, serving the same payload to all devices irrespective of device capabilities, screen size and resolution, and indeed connection speed, can effectively close off entire markets to companies who don't optimize.

It's hard to over-emphasize the importance of basic UX issues such as page load times but if you need a good case study **read what happened to YouTube** when they lightened their pages (summary: entire new territories opened up to them).

With billions of currently active mobile subscriptions in the world, access to the web via mobile devices continues to grow apace. Larger brands know that adapting their content to the increasingly varied devices and contexts that people use to get online has a positive effect on engagement and sales.

Customers have high expectations when it comes to user experience and much research has shown that **abandonment rates increase in direct proportion to load times**. Responsive has offered a way for companies to provide a unified website experience across different device types, but it often comes at the expense of heavy sites that aren't optimized well for mobile devices and compromised connectivity.

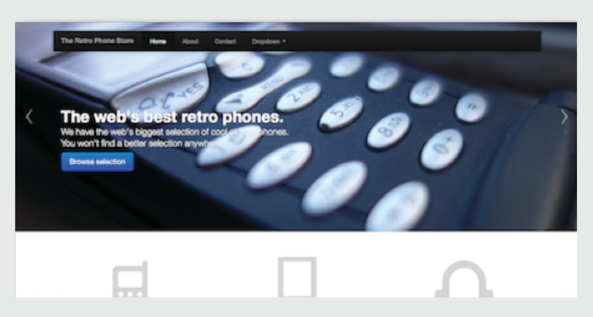
RESS (Responsive Web Design with Server Side components) offers a way to get the best of both worlds: a responsive website design that fully optimized for different device types. By using DeviceAtlas as the server side component, you can make significant performance improvements over 'classic' RWD and enjoy the concurrent uplift in reach, without the need for any ongoing maintenance. Your responsive site will work on almost any device, rather than the desktop, tablet and smartphone buckets of typical RWD implementations, and load faster in all cases. All it takes is a few lines of code, and some simple configuration.

WHY YOU SHOULD DO THIS

With classic RWD, all devices are sent the same images even if they can't show them at their native resolution. This is inefficient at best and market-limiting at worse. Heavier pages only work well on high speed connections, on high end devices with generous data plans.

This technique will significantly reduce image weight with 3 easy steps, 4 lines of code and 1 line of configuration. And once implemented, it requires no maintenance. Your website will have adaptive page weights as well as a responsive design.

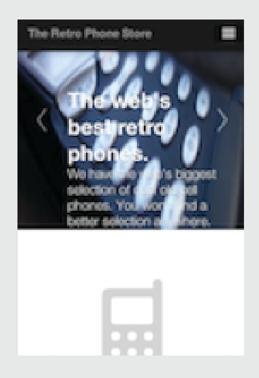
Our example site with fully responsive site design, with responsive page weights:



The web's best retro phones.

We have the web's biggest selection of cool old cell phones. You won't find a better selection anywhere.

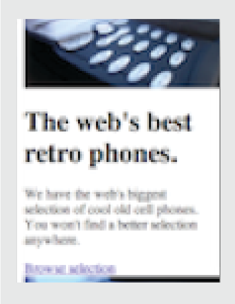
DESKTOP
1360 X 768 1,027 KB



The web's best retro phones.

We have the web's biggest selection of cool old cell phones. You won't find a better selection anywhere.

IPHONE
320 X 480 153 KB



The web's best retro phones.

We have the web's biggest selection of cool old cell phones. You won't find a better selection anywhere.

NOKIA 6300
240 X 320 25 KB

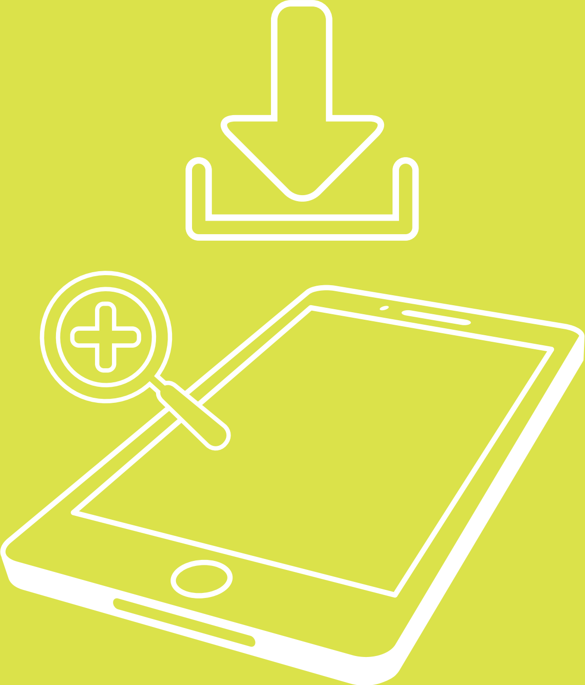
HOW TO IMPLEMENT IT

There are three levels of optimization demonstrated here, each of which builds on the previous level. They can be implemented together or separately.

1. Image payload reduction via server side detection
2. JavaScript & CSS payload reduction
3. Further optimizations based with bandwidth detection

This framework assumes you are using Apache web server but the technique described will work equally well with NGINX, or any programming language. We created a demo site with an **industry-average breakdown of HTML, images and JS** to illustrate the impact of this approach.

INSTRUCTIONS



STEP-BY-STEP INSTRUCTIONS

STEP 1

- Install **Google’s PageSpeed**, an open source web and assets optimization project from Google.The installation process usually activates the module for the default website but you might need to ensure that it works with your virtual hosts, if configured.

  You can read how to do this **here**. Basically you just have to add a line to each one, or get them all to inherit from the default configuration server-wide.

- Restart your web server.

STEP 2

- Get a **DeviceAtlas Cloud** license (a trial licence is available **here**. it’s free and will work fine as a proof of concept).
- Unpack the ZIP file.
- Enter your license key in the DeviceAtlasCloud/Client.php file. DeviceAtlas will be used to decide the optimal size target for resizing images.

STEP 3

- Copy the DeviceAtlas PHP file to a directory where it is executable by the web server. In this case, we’ve created a directory in the root of the site called DeviceAtlasCloud.
- Enter the following code at the top of your HTML file or site template to set up a couple of variables that we can use throughout the page.

```
<?php
   include ‘DeviceAtlasCloud/Client.php’;                                  // instantiate client
    $results = DeviceAtlasCloudClient::getDeviceData();                     // fetch properties for current device
    $props = $results[‘properties’];                                        // store in $props
    $width = (isset($props[‘displayWidth’])) ? $props[‘displayWidth’] : “”; // set $width to correct width or “” if
unknown
?>
```

STEP 4

- The final step is to make sure that all of your images that may need resizing have a width attribute set to use the `$width` variable, enter the following code

```
<img src=”img/slide-01.jpg” width=”<?php echo $width; ?>” alt=”image description” />
```

WHAT'S HAPPENING UNDER THE COVERS

- Images have their width attribute automatically set to the maximum display width for each device by the `$width` variable.
- PageSpeed notices the `width="…"` tag for each image and resizes it down if necessary, replacing the image source attribute with a reference to a resized version. There is no need to set the height attribute because PageSpeed will automatically keep the aspect ratio intact. Resized images are cached so there isn't really any significant impact on the server. Refer to the PageSpeed configuration notes below for more fine-grained control over this cache.

Notes: Only add this variable width tag to images that require resizing for each device – not to images that are already small enough like bullet icons and so forth.

Be aware that setting the width attribute for each image will need to coexist with any css that you define for image display. If you use a cms, you may have to use a different technique for this depending on what access the cms gives you to the underlying html.

Background images may require a different approach, depending on how they are utilized on the site but PageSpeed will read inline `style="…"` tags.

RESULTS

RESS RESULTS

To measure the impact we tested download speed of our demo site for different devices and network speeds, using the Charles Proxy and real devices on various bandwidths.

RESULT 1:
PAGE WEIGHTS SLASHED

First let's take a look at the efficiencies generated in actual payloads for different devices. We will look at what this means in terms of speed and increased user experience a little later.

Before employing RESS the overall over-the-network page size (with GZIP compression from Apache) was 1,027 KB, regardless of device.

Measuring for two mobile devices at each end of the high/low spectrum, an iPhone and a Nokia 6230 with RESS, the overall page size has dropped by **75% for the iPhone** and **84% for the Nokia 6230**. The reduction in image sizes accounting for the majority of this. Importantly, **there is no user-perceptible difference between the site before and after**: the image data that was removed could not easily have been seen by person holding the phone.

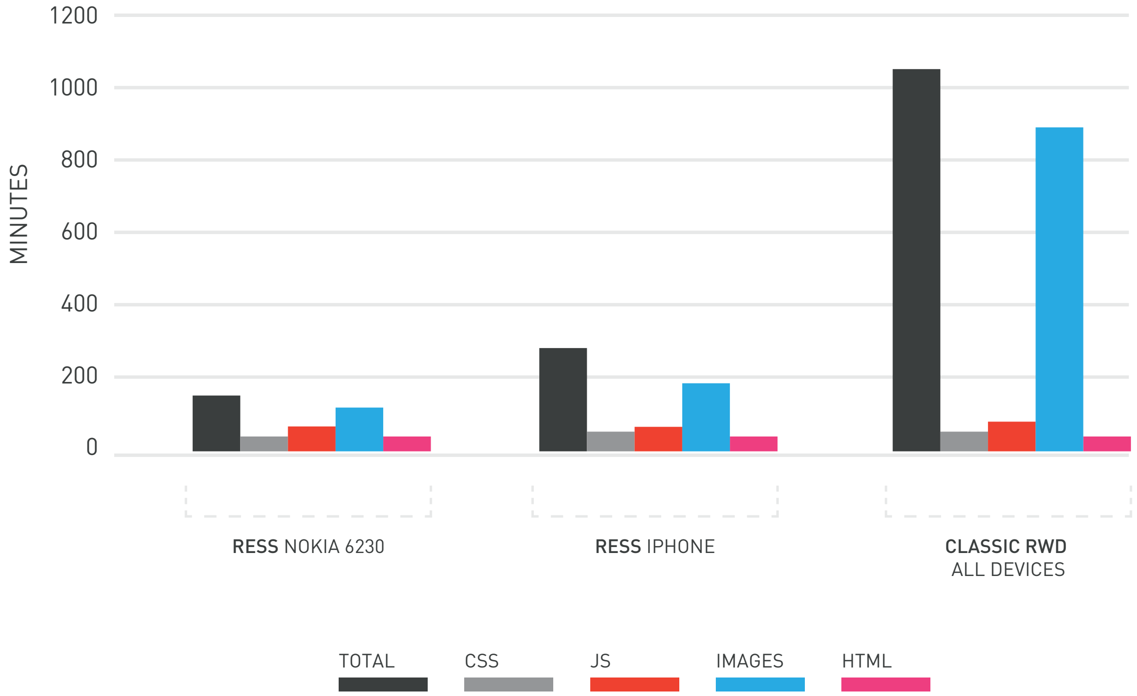
THE DATA

DEVICE	ALL DEVICES	IPHONE	NOKIA 6230
Network size per component	BEFORE	AFTER (75% lighter)	AFTER (84% lighter)
HTML	3 KB	3 KB	3 KB
Images	941 KB	177 KB	89 KB
JavaScript	55 KB	51 KB	51 KB
CSS	27 KB	21 KB	21 KB
Total	1,027 KB	253 KB	164 KB

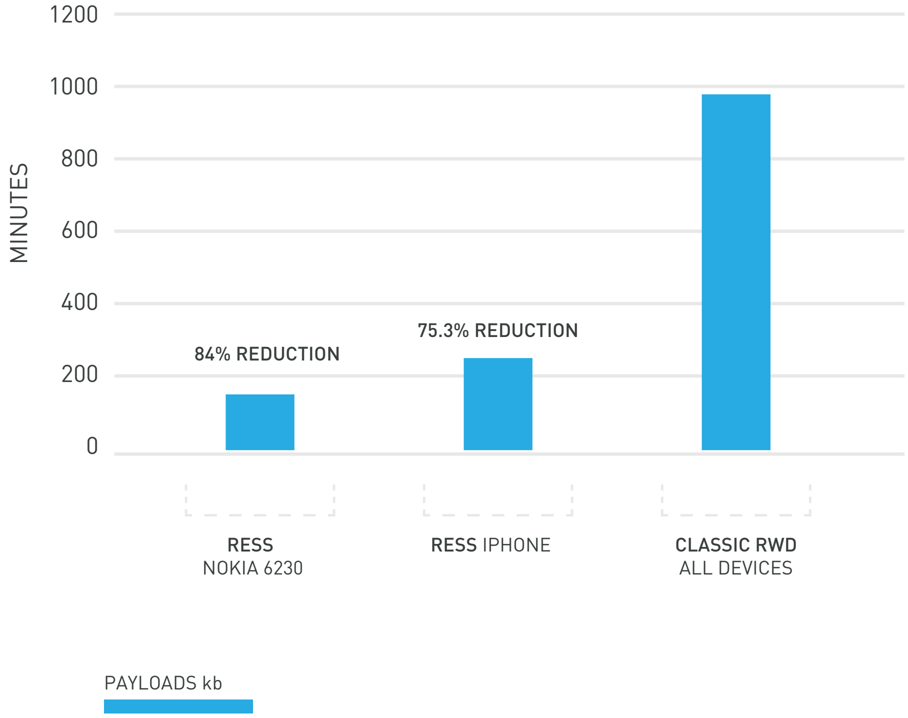
THIS HAS A HUGE IMPACT ON REAL WORLD CUSTOMERS:

- Much faster page load times > better engagement, conversions, lower abandonment rates, higher customer satisfaction
- Wider device and network compatibility > improved reach
- Lower data plan impact > more return visits

ASSET BREAKDOWNS



CODE PAYLOADS: WITH/WITHOUT RESS



SUMMARY

The original 'classic' RWD design may have looked alright on small screens, but the original images were about 5 times wider (in pixel terms) than the average phone display, so load times were never optimal. The sheer size of the page meant that it didn't even finish loading on some devices we tested.

RESULT 2: FASTER PAGE LOADS

The payload efficiencies outlined above lead naturally to increased user experience in the form of faster page loads.

Employing the RESS technique addresses this weakness of 'classic' RWD. By measuring the before and after page load speeds for a retina iPhone over a 3G versus a GPRS network, page load times are significantly improved:

Results on Android devices are similar. On lower-end devices more dramatic improvements are experienced because the image resizing gains are larger.

DEVICE	CONNECTIVITY	LOAD TIME BEFORE	LOAD TIME AFTER	SPEED IMPROVEMENT
iPhone	3G	14s	6s	2.3x Faster
iPhone	GPRS	2m 30s	35s	4.3x faster

FURTHER OPTIMIZATIONS

OPTIMIZATION 2: CSS & JAVASCRIPT

So far, we’ve looked only at the main source of bloat on pages: images. But screen size shouldn’t be the sole factor for optimizing – user contexts and constraints demand more of a multi-device publishing strategy. If you know that the requesting device doesn’t support JavaScript or rich CSS you can lighten further by excluding them.

STEP 1

- Add the following PHP code to the top of your HTML file:

```
$highEndDevice = (isset($properties[‘browserRenderingEngine’]) &&
in_array($properties[‘browserRenderingEngine’], array(‘Gecko’, ‘Trident’,
‘WebKit’, ‘Presto’)));
```

This code identifies a low-end device based on its rendering engine. If the device appears to be a low-end device we’ll jettison the CSS and JavaScript because low-end phones have issues with both the file size and the rendering of CSS, and usually won’t run the JavaScript.

STEP 2

- Add this piece of code in the `<head>` section of your webpage to remove `css` where necessary.

```
<?php if ($highEndDevice): ?>
    <link href=”css/bootstrap.css” rel=”stylesheet”>
    <link href=”css/bootstrap-responsive.css” rel=”stylesheet”>
    <link href=”css/additional.css” rel=”stylesheet”>
<?php endif; ?>
```

STEP 3

- Add the following code at the end of your html file to remove the JavaScript

```
<?php if ($highEndDevice): ?>
    <!-- Le javascript
    ================================================== -->
    <!-- Placed at the end of the document so the pages load faster -->
    <script src=”js/jquery.js”></script>
    .
    .
<?php endif; ?>
```

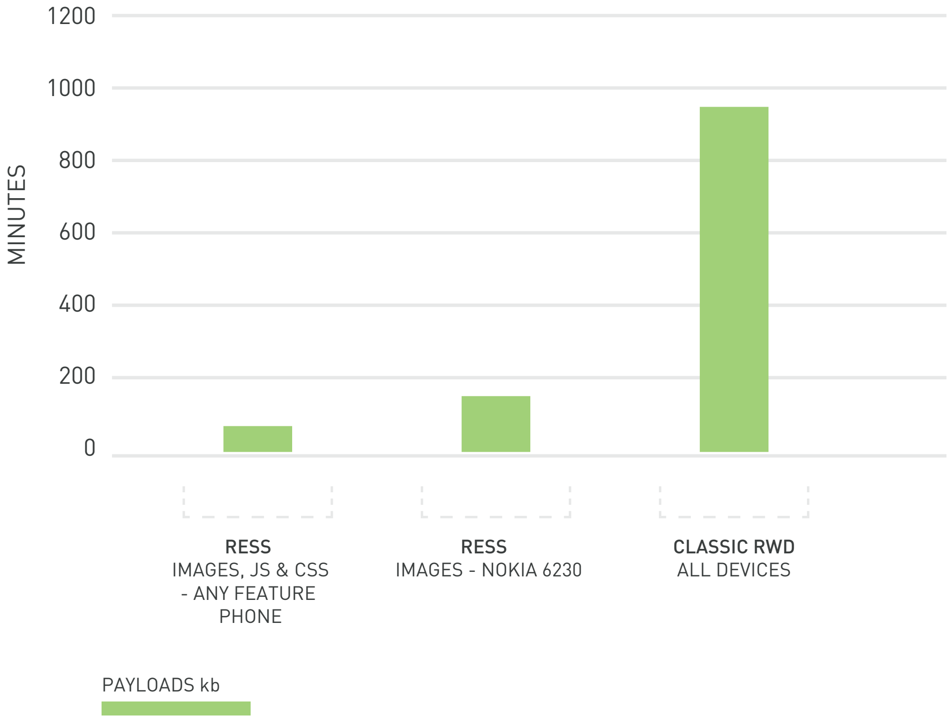
RESULT

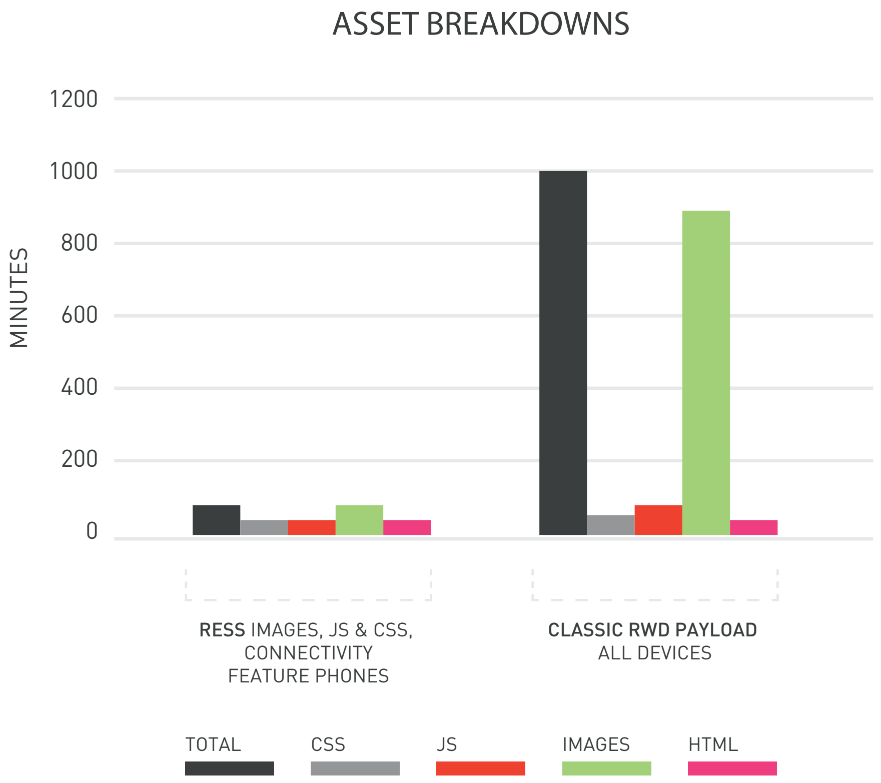
This removes a further 72KB from the payload and actually makes the page look better on low-end devices in addition to being quicker to render.

Our simple RWD page has now gone from a fixed size of about 1 MB to a highly varying one that goes as low as about 92KB, or 11x smaller than the initial page. Not a bad result for less than 10 lines of code. The net result is that our page is now viewable on almost anything, anywhere quickly, from television to feature phone. You may not be targeting TVs and feature phones, but now they will come to you.

Note: In testing this approach on real devices the HTML5 `doctype` tag did not cause problems – the page loaded on pretty much every device we tried.

CODE PAYLOADS:
CLASSIC RWD V RESS (IMAGES, JS, CSS)





OPTIMIZATION 3: CONNECTIVITY ANALYSIS

Anybody who has connected to the internet over airport Wi-Fi, conference Wi-Fi, or from a poorly connected location knows just how frustrating it is to use the web when pages are large, no matter what the device; laptop, tablet or phone.

By measuring the available connectivity, you can *dynamically* apply similar image compression techniques according to bandwidth available to the user. If the connection is poor, images can be aggressively compressed without reducing their pixel size. This can make a huge difference to the browsing experience. The resulting page loads much faster with only a slight impact on the experience—page layout and overall appearance are preserved. Depending on the compression levels chosen many people won't even notice.
DeviceAtlas has a very useful feature called **Connectivity Analysis** to do exactly this, allowing the developer to make some useful choices about what to send the client when bandwidth is limited. [1]

This example is quite simple, yet very effective. If the detected bandwidth available to the device falls below a certain threshold we will redirect the browser to a different virtual host. This virtual host is served by the same web server and serves the exact same page, but triggers a different set of options

[1] Making bandwidth information available to the browser is something that the W3C are working on but the Network Information API is still in draft status so it's not going to be widely deployed in the near future.

for PageSpeed. By changing the image compression level from its default to something much lower, say 20%, images are still very much recognizable, but many times smaller.

This is an example of what is possible rather than a definitive technique:

STEP 1

- Add a new vhost to your server config and configure PageSpeed to use different settings for this

STEP 2

- Restart your web server.

  Note: we are using `site.com` and `lo.site.com` as our vhosts. The `documentroot` is the same in each case – the same html is being served for both vhosts.

```
<VirtualHost *:80>
ServerAdmin webmaster@localhost
ServerName site.com
DocumentRoot /var/www/site.com
ModPagespeed on
</ VirtualHost>

<VirtualHost *:80>
 ServerAdmin webmaster@localhost
ServerName lo.site.com
DocumentRoot /var/www/site.com
ModPagespeed on
ModPagespeedImageRecompressionQuality 20
</ VirtualHost>
```

STEP 3

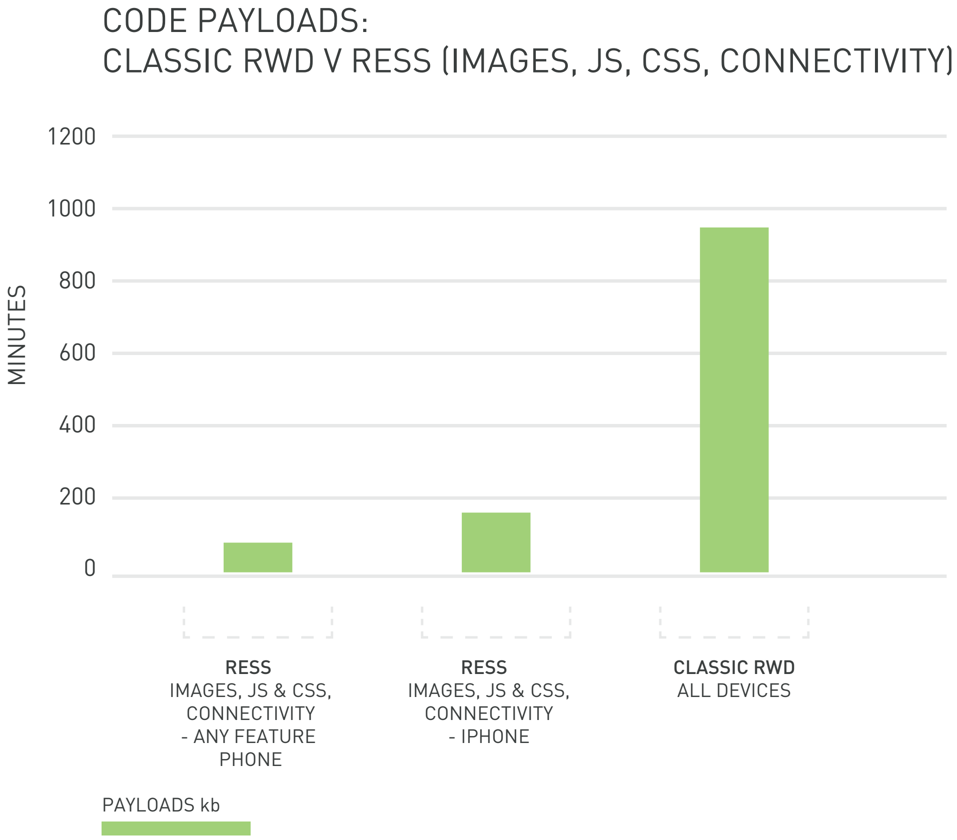
- Add the connectivity checking code to your site template. This switches virtual host if connectivity looks to be poor:

```
require_once:'DeviceAtlasNPC.php'; //check network performance
session_start();
$deviceAtlasNPC = new DeviceAtlasNPC(); // instantiate NPC
$quality = $deviceAtlasNPCK>getQuality(); // test network performance
$path = $_SERVER['SCRIPT_NAME'];
switch($quality) {
case:DeviceAtlasNPC::HIGH_QUALITY:
if:($_SERVER['HTTP_HOST']:==:'lo.site.com'):{
header("Location::http://site.com".$path:);
}
break;
case:DeviceAtlasNPC::MEDIUM_QUALITY:
if ($_SERVER['HTTP_HOST'] == 'lo.site.com') {
header("Location: http://site.com".$path );
}
break;
case:DeviceAtlasNPC::LOW_QUALITY:
if ($_SERVER['HTTP_HOST'] == 'site.com') {
header("Location::http://lo.site.com".$path );
}
break;
default:
}
```

The initial redirect to the lower bandwidth version of the site does have some impact on the load time, but this is far outweighed by the net savings in doing so, particularly for those with constrained bandwidth.

High bandwidth customers should be almost unaffected. The cost of this redirect step on a slow GPRS connection is approximately 1 second, but the resulting savings can add up to minutes.

RESULT 1:
FURTHER PAYLOAD SAVINGS



RESULT 2:
FURTHER PAYLOAD SAVINGS

DEVICE	ORIGINAL SITE		ADD IMAGE RESIZING		ADAPTIVE JS & CSS		ADAPT TO CONNECTIVITY	
	Page Size	Load Time	Page Size	Load Time	Page Size	Load Time	Page Size	Load Time
iPhone 3G	1027 KB	14s	253 KB	6s	253 KB	6s	153 KB	5s
iPhone GPRS	1027 KB	2m 30s	253 KB	40s	253 KB	40s	153 KB	25s
Feature Phone (2G)	1027 KB	∞	164 KB	35s	92 KB	25s	25 KB	12s

CONCLUSION

CONCLUSION

With all of these optimizations in place, we now have page weights that scale dynamically from about 1 MB all the way down to 25 KB. The page incorporates the best of RWD and server-side optimizations to yield a *dynamic range* (a ratio of the largest page weight to lowest page weight served) factor of over 40.

As a result, the "reach" of this page has been extended from desktop and smart devices in well-connected locations to almost anything, anywhere, regardless of connection type.

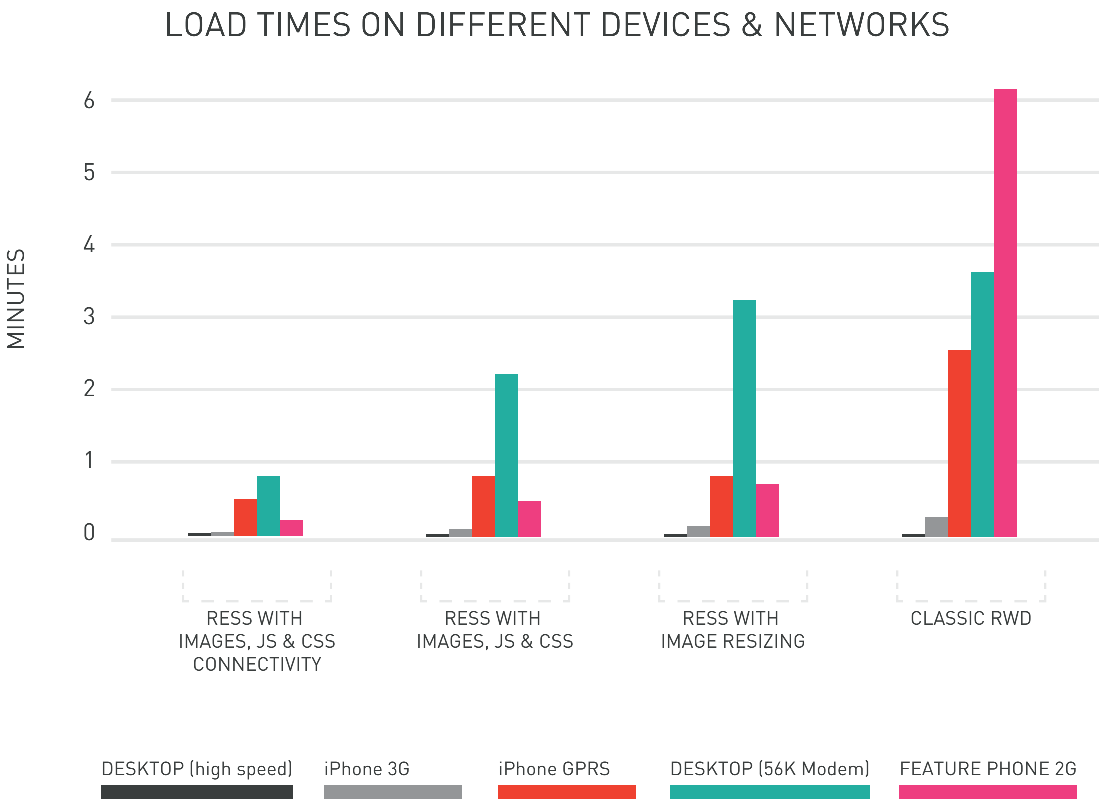
DEVICE	ORIGINAL SITE		ADD IMAGE RESIZING		ADAPTIVE JS & CSS		ADAPT TO CONNECTIVITY	
	Page Size	Load Time	Page Size	Load Time	Page Size	Load Time	Page Size	Load Time
iPhone 3G	1027 KB	14s	253 KB	6s	253 KB	6s	153 KB	25s
iPhone GPRS	1027 KB	2m 30s	253 KB	40s	253 KB	40s	153 KB	25s
Feature Phone (2G)	1027 KB	∞	164 KB	35s	92 KB	25s	25 KB	12s

These loading times are all worst-case scenarios tested with an empty cache. Subsequent page loads as you traverse the site will feel much faster.

Coming back to the page linked at the top of this paper, Chris Zacharias, speaking of his experience optimizing YouTube's page weight, **said**:

> **"[Previously] entire populations of people simply could not use YouTube because it took too long to see anything. By keeping your code small and lightweight, you can literally open your product up to new markets."**

By using some of the techniques outlined in this article you may be able to achieve similar results for your website.



ABOUT DOTMOBI

dotMobi focuses on giving content publishers the tools they need to ensure the Web will work on mobile phones and connected devices with speed, accuracy and relevant content. We are immersed in all things mobile web. Mobile is in our DNA.

A wholly owned subsidiary of Afilias, dotMobi was founded in 2005 by leading mobile operators, network device manufacturers, and Internet content providers, including Ericsson, Google, GSM Association, Hutchison 3, Microsoft, Nokia, Orascom Telecom, Samsung Electronics, Syniverse, T-Mobile, Telefónica Móviles, Telecom Italia Mobile (TIM), Visa and Vodafone

DeviceAtlas™	DeviceAtlas is one of the largest open repositories of mobile device profiles, based on W3C recommendations. It provides the supporting tools, techniques and assistance that you need to take that data and use it to rock your mobile users' world. www.deviceatlas.com
goMobi™	goMobi™ is the world's first content mobilization platform, a hybrid of a traditional content management system and a practically automatic mobile website builder. It's the smart, simple way for businesses and designers & developers to build a mobile Web presence. www.gomobi.info
mobiThinking™	mobiThinking™ is here to help you market your mobile site. It's packed with insight, analysis and opinions from the world's mobile marketing gurus. www.mobithinking.com
mobiForge™	mobiForge™ is the dotMobi developer forum -- a center for mobile Web developer tools, resources and support. More than 50,000 developers and designers meet here to compare notes, share tips, upload ideas and download expertise. www.mobiforge.com
mobiReady™	mobiReady™ evaluates mobile-readiness using industry best practices and standards. Test your mobile website and get a free report plus in-depth analysis to determine how well your site performs on a mobile device. ready.mobi

CONTACT INFORMATION

dotMobi
2 La Touche House
IFSC
Dublin 1
Ireland

Email: sales@deviceatlas.com
Phone: +353.1.854.1100
Fax: +353.1.791.8569