How Can Web Developers Optimise for Lower-Power Smartphones in the Developing World?

Edward George Prince

6151713

Computer Science

Department of Computing

29th April 2019

Abstract

This paper looks at methods for optimising modern web applications for users with low-powered smartphones. By analysing a testbed application, it is discovered that performance optimisations can be made without sacraficing user experience. This was found using the Google Development Tools to analyse an app built with React, Node.js, Socket.io and Express.js.

Contents

1 Introduction **4**
1.1 Terminology 4

2 Literature Review **5**
2.1 Introduction 5
2.2 CPU 5
2.3 Mobile Web Browser 6
2.4 Development Issues 6
2.5 Summary 7

3 Methodology **8**
3.1 Introduction 8
3.2 Testbed Application 8
3.3 Performance Profiling 8

4 Design and Implementation **9**
4.1 Case 1 - Bundling 9
4.2 Case 2 - Server Decision Tools 9
4.3 Summary 9

5 Evaluation and Results **10**

6 Discussion **11**

7 Project Management **12**
7.1 Timeline 12
7.2 Communication 12
7.3 Legal, Social and Ethical Implications 12

8 Conclusion **13**

9 Bibliography **14**

10 Appendices **15**
10.1 Unbundled Profiling 15
10.1.1 At No Throttle 15
10.1.2 At 4x Throttle 15
10.1.3 At 6x Throttle 16
10.2 Bundled Profiling 17
10.2.1 At No Throttle 17
10.2.2 At 4x Throttle 17
10.2.3 At 6x Throttle 18

1 Introduction

Chapter 4 explores a selection of the relevant existing literature surrounding and encapsulating the topic. Due to the subject being in its infancy in relation to most areas of Computer Science - there is a distinct lack of relevant literature. Network-based optimisations are a popular article subject - but as it bears little need in developed countries - less powerful CPU based optimisations are not a hot topic for academic research. Whilst the usefulness is definitely questionable in developing countries - many developing country populations still use older smartphones, and could benefit from more research in this area - ultimately providing better user experiences for global user bases. Additionally - if an application can reduce the required usage of the CPU - it can also reduce the CPU power consumption and required cooling - ultimately increasing battery life, a feature useful anywhere in the world. The methodology chapter presents the formation and execution of the experiments, and details on how results could be replicated. The experiments are split up into component sections, each identifying a different area for potential optimisation. Each section has a brief summary, a prediction of the expected results upon modification and re-profiling, and the actual results, how the compare, and why they might differ to the predictions. The chapter is concluded with a summary of all of the results, and how they relate to optimisations for low-powered smartphones.

1.1 Terminology

Devtools Google Chrome Development Tools
CDN Content Delivery Network

2 Literature Review

2.1 Introduction

As of 2018, there were 3.7 billion mobile devices connecting to the internet (Stevens, 2018) showing that a huge proportion of people use their mobile devices, and there's nothing to suggest that the growth in mobile usage will do anything other than increase.

Given that we know 90% of users stop using an app due to poor performance, and 86% will uninstall the app altogether (AppDynamics and University of London, 2014), there is the possibility for unoptimised web apps to lose huge numbers of potential users due to the app performance not meeting our standard usability criteria.

Nejati and Balasubramanian discuss the difficulty in isolating the effects of the mobile browser in determining the root of poor performance (Nejati, Balasubramanian, 2016). This project will differ from that research by making use of the Google Chrome Dev Tools mobile simulator - taking many of the factors out of consideration, allowing isolation on only changes to the CPU performance. They also identify the issue of mobile browsers being much slower than desktop, as agreed with in other research on mobile optimisations in the cloud era (Wang et al. 2013), a problem given the dramatic increase in mobile devices globally.

The project will also rely on prior research on optimisation of images, (Thiagarajan et al. 2012) - leaving the focus to be on the fundamental pillars of web development - HTML, CSS and JavaScript.

2.2 CPU

Why are Web Browsers Slow on Smartphones? It is suggested that network round-trip time is a major problem. Whilst this may have been true at the time, due to the increase in network capacity, and the trends shown so far, it could be assumed that time-based network problems become increasingly less relevant, with the hardware-based issues becoming more prominent. Particularly in developed countries, modern smartphones with the latest hardware are readily available, which is why this project focuses on mobile phones in usage in the developing world.

Over the last 20 years, Moore's Law has begun to plateau (Simonite, 2016), and hardware capability increases along with it.

Networks on the other hand are continuing to get faster and offer better coverage. Taking Kenya (a developing country) as an example, figure 2 shows that network speeds are increasing at a dramatic rate:

"Of all ITU [International Telecommunication Union] regions, the strongest growth was reported in Africa, where the percentage of people using the Internet increased from just 2.1% in 2005 to 24.4% in 2018" (ITWeb, 2018). Containing a large number of developing countries, by looking at Africa as a whole, and Kenya as a microcosm - we can see there is a likelihood of a fast network future - and with the plateau in Moore's Law, a likelihood of stagnation in CPU performance increases. A fast network future may lead to an emergence in prioritisation of CPU-based optimisation, over decreasing app sizes - which would have an increasingly negligible effect as network speeds improve. There has also been extensive research on optimization over networks already - reducing content, using CDN's, compressing content, utilising browser caching techniques (Iliev, 2014), and front end optimisations for modern devices. This research will avoid liminality by relying on the research already conducted on networks, and instead fill the gap left for optimising web applications to perform highly for low-end smartphones.

An In Depth Study of Browser Performance This article demonstrates research around locating mobile browser bottlenecks. It concludes that computational activities are the main bottlenecks, highlighting the need for research on optimising for lower-powered CPU devices, where the effects of computational bottlenecks will be be exceedingly more relevant.

It also states that this is not the case for desktops, where network activity is the dominant bottleneck - also emphasizing the importance of optimising CPU-intensive applications for mobile devices. Wang et al. suggest that the most costly area for TTI is resource loading - which, as the sources are credible, makes sense - however no speculation is offered about the state of the future of processors and networks, making the research relevant currently, but perhaps not in years to come.

2.3 Mobile Web Browser

Mobile Web Browser Optimisations in the Cloud Era There are also many comparisons made between the mobile browser and desktop browser, over the same network speed - finding that the mobile version can take up to nine times as long to load. As they are on the same network - we know that this is therefore down to the hardware and mobile browser - not server-to-client issues. Suggestions have been made that more effort should be put into making mobile browser optimisations - with which future developers could utilise to make web applications more globally usable.

2.4 Development Issues

Analyzing Mobile Browser Energy Consumption Shows that reducing JavaScript and utilising HTML functionality leads to a less power-hungry application.

JavaScript being the most power-hungry part of the traditional web application. It also shows the importance of optimising CSS rules, with a reduction to only necessary rules, along with the migration of multiple CSS files into one file for a given page seeing a drop of 5 Joules from the power supply.

Mobile Web Browser Optimisations in the Cloud Era It also looks at the benefits of web page prefetching by predicting which links the user will click, and prefetching/loading the relevant data to render the page should the link be clicked more quickly. Does low CPU capability make this a relevant point or not?

The other large aspect they discuss, is the usage of cloud-based architecture. This might increase network latency, and there are other potential security issues, but focusing purely on optimisation, this style of architecture allows processing to be partially taken away from client devices, reducing the workload of the client CPU's. This could be introduced into the chat web app by analysing the impact of moving some intensive functionality from the client to the server.

2.5 Summary

3 Methodology

3.1 Introduction

3.2 Testbed Application

3.3 Performance Profiling

4 Design and Implementation

4.1 Case 1 - Bundling

4.2 Case 2 - Server Decision Tools

4.3 Summary

5 Evaluation and Results

6 Discussion

7 Project Management

7.1 Timeline

7.2 Communication

7.3 Legal, Social and Ethical Implications

8 Conclusion

9 Bibliography

10 Appendices

10.1 Unbundled Profiling

10.1.1 At No Throttle

Table 1: Unbundled at No Throttle

Run (0x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	4.7	3111.8	43.4	0.6	60.9	453.4
2	4.1	3431.6	48.2	0.7	62.4	5100
3	4.8	3180.1	47.4	0.9	65.6	440.6
4	1	3197.6	50.7	0.4	36.1	156.2
5	4.7	3899.6	77.9	0.7	72.8	539.3
6	7	3834.2	54.9	1.3	88.1	477.8
7	4.9	3941.5	62.4	12.8	83.9	617.5
8	5.3	3694.9	53.2	1.4	94.1	515.8
9	5.2	4164.4	59.1	0.7	84.2	670.3
10	5.7	3434.5	59.3	3	157	804.3
11	4.6	3581.2	55.8	2.6	77.2	682.4
12	4.2	3206.8	45.5	0.6	71.9	524.1
13	5.8	4151.7	53.5	0.7	88.5	500.6
14	4.1	4656.3	59.2	0.8	78.9	555.5
15	12	4185.6	57.1	1.2	80.9	664.8
16	4.2	3698.1	51.7	1.1	84.2	492.3
17	4.3	3724.6	49.5	2.6	78.6	496.5
18	5.4	3618.6	53.3	0.9	83.2	496.6
19	5.4	4018.2	57.1	1	85.8	525
20	4.6	3551.9	50.9	0.7	82.7	490.1
Average	5.1	3714.16	54.505	1.735	80.85	760.155

10.1.2 At 4x Throttle

Table 2: Unbundled at 4x Throttle

Run (4x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	15.5	12827.6	164.2	2.1	186.8	645.7
2	6.2	10566.3	159.9	1.9	214.1	734.5
3	12.9	13499	155.4	2.1	205.1	671
4	19.8	12701.8	155.1	2.2	193.6	898

Run (4x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
5	10.7	9852.5	187.1	6.7	372	713.9
6	11	12833.3	178.4	7.9	261.2	947.5
7	7.2	11398	180.7	6.7	290.4	661.1
8	16.1	12382.4	165.8	2.4	192.2	799.7
9	12.1	12313.2	181.9	2.3	313.2	932.5
10	15.8	14077.1	180.6	2.1	274.5	750.2
11	18.3	7225.1	169.9	2.3	190.4	774.6
12	9.3	13893.1	187.2	2.4	121.5	594.6
13	13.2	13131.3	199.9	5.2	234.9	789.8
14	9.8	14363.4	167.2	1.7	199.9	967.2
15	19	11134.7	168.8	6.6	219	553
16	19.5	12194.9	177.9	3.7	235	627.1
17	11.8	12514.6	187	2.8	186.3	942.6
18	14.6	12724.4	160.5	3	282.4	797.2
19	26.1	15305.5	172.1	3.2	398	816.5
20	16.3	12208.5	176.5	6.3	242.7	584.7
Average	14.26	12357.335	173.805	3.68	240.66	760.07

10.1.3 At 6x Throttle

Table 3: Unbundled at 6x Throttle

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	8.5	19643.1	211.8	2.4	246.7	577.7
2	6.4	14932.7	191.6	2.6	644.9	631.9
3	23.7	19366.7	254.1	3.3	244.1	552.2
4	18.3	19754.3	184.8	4.3	351.3	538
5	11.4	19008.9	227.9	2.9	715.7	668.5
6	25.9	21328.3	108.5	14.9	1622.9	1117.2
7	9.6	14051.8	239.6	3.1	365	1273.9
8	9.2	19393.9	188.9	4.9	334.8	715.2
9	17.5	21682.2	271.2	4.1	529.6	980.9
10	21.3	22173.7	243.6	4	835.8	851.5
11	9.2	17532.4	256	4.4	320.8	921.1
12	8.2	19373.5	251.3	2.7	213.5	654.6
13	8	17301	224.3	1.8	387.6	1423.8
14	11.2	20298.6	118	2.9	512.2	1036.1
15	11.5	16325	232.9	10.1	220.3	797.8
16	9.8	18459.1	178.3	1.4	303.8	684.5
17	22.3	18405.5	380.6	3.9	249.3	852.9

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
18	28.6	19070.7	224.5	1.9	1249.2	562.1
19	15.2	13584.4	177.1	1.1	307.5	794.6
20	10.1	20168.3	260	3.1	319	662.9
Average	14.295	18592.705	221.25	3.99	498.7	814.87

10.2 Bundled Profiling

10.2.1 At No Throttle

Table 4: Bundled at No Throttle

Run (0x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	15.8	3283.4	50	0.7	53.6	149.3
2	16.4	3276.8	54	1.7	63.7	327.6
3	14.4	3327.7	82.7	1.1	82.8	290.8
4	13.7	3294.7	52.6	0.7	46.8	343.1
5	14.6	3656.5	65.8	1.1	88.6	391.3
6	14	3192.7	46.6	0.9	48.3	151.6
7	13.6	3272.8	52.5	0.7	46.9	122.1
8	15	3363.2	51.7	0.7	47.2	195.6
9	15.2	3704.2	75.4	1.1	48.5	222.8
10	14.4	3178	57.6	1.4	40.5	204.6
11	15.8	3632.6	59.2	2.4	61.9	139.3
12	13.6	3217.8	52.1	1	40.9	134.4
13	17	3649.2	49.6	0.8	55.7	266.3
14	15.8	3138.1	54.6	1.9	45.6	152.1
15	14.7	3534.7	57.3	1.6	69.2	300
16	10.7	2367.5	44.3	0.8	34.1	82.7
17	17.4	3422	56.3	1	47.5	383.2
18	15.7	3422.3	50.5	0.8	44.4	152.7
19	26	3279.9	50	2.1	52.8	400.9
20	14.7	3589.4	69	0.9	54.5	351.3
Average	15.425	3340.175	56.59	1.17	53.675	238.085

10.2.2 At 4x Throttle

Table 5: Bundled at 4x Throttle

Run (4x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	48.3	12215.3	162.9	0.8	94.3	111.3
2	55.7	13474.3	172.7	5.5	180.6	302.2
3	33.9	12209.6	167.8	5.9	211.7	260.7
4	32.9	11922.5	160.7	0.6	251.3	378.4
5	33	10846.5	172.4	0.5	243.4	434.5
6	31.5	11881.3	175	4.5	115.2	136.6
7	62.7	12602.4	174.8	0.9	132.6	144.9
8	64.3	12968.5	176.2	0.9	235	207.6
9	49	7734.3	168.8	26.3	253.6	183.1
10	23.6	11292.8	161	0.3	215.6	481.1
11	38.4	13622.1	171.4	1.3	161.5	367
12	16.8	11135.7	153.9	0.3	229.3	366.1
13	28.7	10936.2	161	0.7	292.6	306
14	32.8	10993.3	177	2	100.9	201.8
15	60.8	11098.9	176.4	2	205	194.9
16	26.9	13087.2	157.7	9.3	246.4	154.1
17	31	12473.6	162.4	1.9	179.5	829.6
18	26.6	12748	167.7	2.3	235.9	241.2
19	26.6	12748	167.7	2.3	235.9	241.2
20	34.3	13010.6	170.5	7.6	118.2	209.2
Average	37.89	11950.055	167.9	3.795	196.925	287.575

10.2.3 At 6x Throttle

Table 6: Bundled at 6x Throttle

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	40	17984.3	222.7	0.5	239.7	168.3
2	36.4	18407.9	380.5	17	191	396.8
3	74.2	19338.6	232	1.3	151.7	365.2
4	65.3	20081.5	157.3	1.9	378.3	591.8
5	30.6	121064.4	237.4	5.1	185.7	276.7
6	44.6	17446.2	215.9	0.9	165.2	210
7	31.8	17675.2	241.1	1.1	237.8	236.4
8	68.5	19803.6	289.5	22.9	234.3	266.5
9	34.1	13028.6	231.6	0.9	93.4	490.3
10	25.9	18598	204.4	4.9	153.2	186.2
11	37.3	17001.4	245.4	0.3	483.1	306

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
12	38.6	17856.6	219.2	0.8	255.2	120.8
13	22.4	17026.3	214	0.4	472.8	411.7
14	35.7	20385.4	213.8	26.7	341.1	264.9
15	45.9	17995.6	235.1	2.6	314.4	221
16	30.7	18087.9	266.2	1.4	182.5	288.3
17	93.3	19167.8	219.5	0.8	249.1	366
18	107.2	19614.1	273.9	7	748.3	189.3
19	104.1	19798	223.4	20.7	501	251.4
20	21.3	16025.1	222.7	1	345.4	519.9
Average	49.395	23319.325	237.28	5.91	296.16	306.375