Impact of Device Performance on Mobile Internet QoE

Mallesham Dasari, Santiago Vargas, Arani Bhattacharya, Aruna Balasubramanian
Samir R. Das, Michael Ferdman
Department of Computer Science, Stony Brook University

ABSTRACT

A large fraction of users in developing regions use relatively inexpensive, low-end smartphones. However, the impact of device capabilities on the performance of mobile Internet applications has not been explored. To bridge this gap, we study the QoE of three popular applications – Web browsing, video streaming, and video telephony – for different device parameters. Our results demonstrate that the performance of Web browsing is much more sensitive to low-end hardware than that of video applications, especially video streaming. This is because the video applications exploit specialized coprocessors/accelerators and thread-level parallelism on multi-core mobile devices. Even low-end devices are equipped with needed coprocessors and multiple cores. In contrast, Web browsing is largely influenced by clock frequency, but it uses no more than two cores. This makes the performance of Web browsing more vulnerable on low-end smartphones. Based on the lessons learned from studying video applications, we explore offloading Web computation to a coprocessor. Specifically, we explore the offloading of regular expression computation to a DSP coprocessor and show an improvement of 18% in page load time while saving energy by a factor of four.

CCS CONCEPTS

• **General and reference → Experimentation**; • **Human-centered computing → Ubiquitous and mobile computing**; **Ubiquitous and mobile devices**; *Mobile Applications*;

KEYWORDS

Mobile Applications, Quality of Experience, Hardware Accelerators.

ACM Reference Format:
Mallesham Dasari, Santiago Vargas, Arani Bhattacharya, Aruna Balasubramanian, Samir R. Das, Michael Ferdman. 2018. Impact of Device Performance on Mobile Internet QoE. In *2018 Internet Measurement Conference (IMC '18), October 31-November 2, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3278532.3278533

1 INTRODUCTION

Mobile smartphones have now penetrated a significant fraction of the world's population. They vary widely in terms of cost and performance. For example, the costs of smartphones currently on the market range between \$50 – \$1000 [1, 7]. The cost largely depends on the hardware specifications. A \$600 phone such as OnePlus5 has eight cores, running up to a 2.4 GHz clock frequency and 6 GB RAM, whereas a cheaper \$60 phone (e.g., Dell Venue Pro) has only two cores up to a 1 GHz clock frequency and 512 MB RAM.

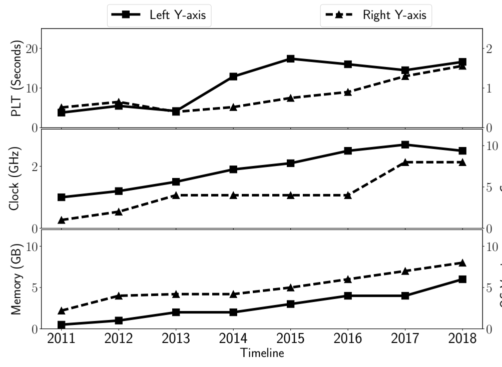
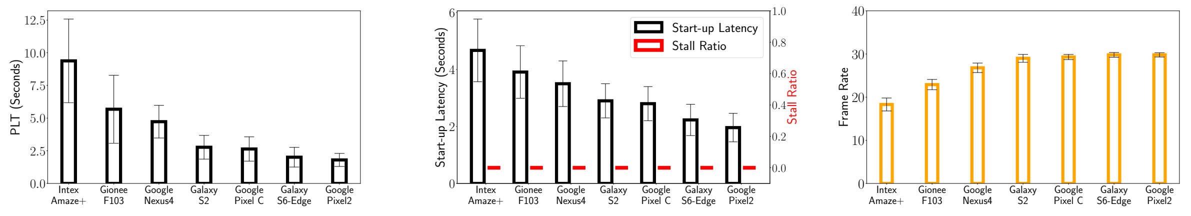


Figure 1: Evolution of Web page performance and device parameters over the last 8 years. The growth in device performance is not on par with the growth of application demands.

A key question arises: how much of an application's quality of experience (QoE) depends on the phone's hardware given that widely different phones with very different price points are available in the market? This question is specifically important because it is well known that *compute* is a key performance bottleneck for mobile applications such as browsing [23, 31]. However, it is not well understood which aspect of computation/hardware specifications is significant to performance. Knowing the hardware component that has the most impact on end-user performance is crucial to designing better phones under a budget.

The problem is more acute among low-end phones. As an example, our results show that mobile Web page loads on two popular phones in India, the Intex Amaze 4 (≈\$60) and Gionee (≈\$150), are **5**× to **3**× worse, respectively, than the Google Pixel2 (≈\$700) under the same network conditions (§2). The problem is not specific to low-end phones alone. Despite the improvements in hardware, the QoE of applications has not improved because of increased application complexity and a mismatch between QoE requirements and hardware enhancements. See Figure 1, which uses the page load history [4] and device data mined from over 480 Android smartphone specifications over the past 8 years. The figure shows that page load times (PLT) have *increased* by four × despite the improvements in devices, hardware, and networks. This difference also underscores the importance of mobile web experience on low-end hardware.

To address the question posed, we characterize the QoE of common mobile applications under four different hardware components: (1) clock frequency, (2) memory, (3) number of cores, and (4) Android governors. (The governors control the CPU frequency) Our goal is to understand how each of these device parameters affect



(a) Web Browsing (Google Chrome) **(b) Video Streaming (YouTube)** **(c) Video Telephony (Skype)**

Figure 2: Mobile application performance across diverse devices: (a) Web browsing, (b) Video Streaming, (c) Video Telephony. The horizontal axis shows the device type; their corresponding specifications are tabulated in Table 1.

Device Name	Application Processor	Number of Cores	OS Version	Clock Min-Max (MHz)	GPU Type	RAM Size (GB)	Release Date	Release Cost
Intex Amaze+	Spreadtrum SC9832A	4	6.0	300-1300	Mali-400	1	Jan, 2017	$60
Gionee F103	MediaTek MT6735	4	5.0	300-1300	Mali-T720	2	Oct, 2015	$150
Nexus4	Snapdragon S4 Pro	4	5.1.1	384-1512	Adreno 320	2	Nov, 2012	$200
SG S2-Tab	Exynos 5433	8	5.0.2	400-1300	Mali-T760	3	Sept, 2015	$450
Pixel C-Tab	Tegra X1	4	8.0.0	204-1912	Maxwell	3	Dec, 2015	$600
Pixel2	Snapdragon 835	8	8.0.0	300-2457	Adreno 540	4	Oct, 2017	$700
SG S6-edge	Exynos 7420	8	6.0.1	400-2100	Mali-T760	3	April, 2015	$880

Table 1: Mobile devices used in our experiments and their corresponding specifications.

the QoE of three of the most popular mobile applications: Web browsing, video streaming, and video telephony.

We find that Web and video applications have very different architectures. As a result, different hardware specifications affect the two classes of applications differently. For example, Web applications are significantly affected by clock speeds, but video applications are virtually unaffected. In contrast, changing the number of cores affects video applications but has no significant impact on Web applications. To dig deeper, we isolate the effect of the hardware parameter on the different aspects of the applications to shed light on not only *how* the hardware component affects the QoE but also *why*.

Our key finding is that Web performance is impacted by low-end phones. In particular, slow clock speeds affect Web browsing adversely. Web page loads slow down by 5× when clock frequency drops from 1512 MHz to 384 MHz. Interestingly, video applications are largely unaffected by this change even though video processing is a computationally intensive operation. This is because video decoding uses dedicated hardware decoders, available even on low-end phones. Also, video applications use parallel operations among multiple CPU cores for post-processing (such as muxing and demuxing of audio/video). In contrast, web applications do not use multiple cores effectively. Given that even low-end phones have multi-core processors, the performance of video applications is impacted little on low-end phones, but the performance of Web applications is severely affected.

Finally, similar to video applications, we experiment with offloading Web computation (hardware offloading) to an existing DSP coprocessor/hardware accelerator on the Nexus4 phone. We find that leveraging hardware offloading is a promising alternative to improving Web performance under a slow CPU clock. Our preliminary analysis with offloading only regular expression computations shows an improvement of 18% in page load time along with a 4× reduction in energy.

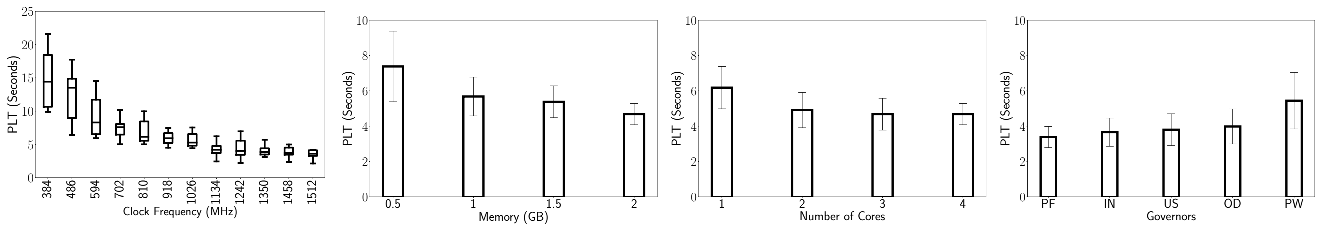
2 QOE ACROSS LOW & HIGH-END DEVICES

As a first step, we study the performance of the three Internet applications – Web browsing, Video streaming, and Video telephony – across seven different smartphones. The phones are chosen so that there is significant diversity in terms of hardware/OS and cost (Table 1). The cost ranges from $60 to $880, and the maximum CPU clock frequencies range from 1.3 GHz to 2.4 GHz. We first describe the default experimental setup before turning to the results.

2.1 Measurement Setup

Web Browsing: We measure browsing performance over **Chrome** 63.0.3239.111 in terms of page load time (PLT). PLT is the time elapsed between when the URL is sent to the server and when the DOMLoad event is fired [36]. We load the top 50 Web pages from Alexa [38], clear the cache (including DNS), and estimate the average PLT. We use the WProf tool [36] to analyze the critical path of the page load process and break down the critical path into computation and network activities. Compuation activities include HTML parsing, Javascript evaluation, and rendering. Network activities involve requesting and downloading the objects on the Web page (such as html, css, js, and image files). We automate the page loads for repeatability using the Chrome remote debugging protocol [34] over the Android Debug Bridge (ADB) [2].

Video Streaming: We use **YouTube** to measure video streaming performance using two QoE metrics: *start-up latency* (network-centric) and *stall ratio* (device-centric). Start-up latency is the time from when the request was issued to when the application starts displaying frames. The stall ratio is the amount of time the video stalls during the playback expressed as a fraction of playback time. Both of these metrics can be measured using YouTube player APIs [3]. The performance is measured over a 5 min FullHD (1080p) video clip. Note that the received video quality is the same in all the experiments because we have high network bandwidth. We use ADB [2] to programmatically request the video content for repeatability.



(a) Clock Frequency **(b) Memory** **(c) Number of Cores** **(d) Governors**

Figure 3: Impact of device parameters on Web browsing (Google Chrome)

Video Telephony: We use **Skype** to measure the performance of video telephony. We measure QoE in terms of *call setup delay* (network-centric) and *frame rate* (device-centric) metrics. The frame rate is measured as the number of frames shown per second and call setup delay is the time it takes for the client to get the response once the receiver answers the call. Because Skype does not provide APIs to extract such QoE information, we use an indirect approach. We configure Skype to display the frame rate and call setup delay on the screen during the video call. Similar to [9], we screen-record the Skype call using the AZ screen recorder [28] and extract the QoE information using an optical character recognition tool [33].

Because Skype is an interactive application, it requires an active participant on both ends. In our setup, when the Skype call is placed from the mobile device to a laptop, the laptop runs a virtual webcam [21] that plays a video file in Skype instead of the camera feed; at the mobile end, the video can be viewed during the Skype call. To automate (starting and ending) the Skype calls, we use the AndroidViewClient (AVC) library [5].

Network Setup: Because the focus of this work is to measure the impact of the device hardware, the experiments are setup to minimize the impact of the network and the Web/video servers. We host the video and pages on a desktop on our LAN created using an Aruba Access Point (AP) with a 72 Mbps link speed, 10 ms RTT, and 0% loss. The mobile device connects to the server over our LAN. For each workload, we repeat the experiment 20 times and present the average and standard deviation. For each experiment, we use default values for non-treatment variables.

2.2 QoE Across Devices

Fig. 2 shows the performance of the three applications across the devices. Based on the device model, a significant difference in performance exists even though all experiments are done in the same network conditions.

For Web page loads (Fig. 2a), there is a 7 sec difference in average PLT between the low-end Intex Amaze+ phone and the high-end Google Pixel2. The standard deviation in PLT is also higher (>3 seconds) in the Intex Amaze+ than in the Pixel2. This must stem from the device itself because the network conditions remain unchanged.

In the case of YouTube (Fig. 2b), a linear increase occurs in start-up latency from 2 to 5 seconds from the high-end to low-end devices. However, after the start-up latency, there is zero impact on the stall ratio. In effect, when the user waits for the video to start, there is practically no difference in QoE between the low-end and the high-end device. For Skype (Fig. 2c), the frame rate decreases from 30fps to 18fps between the high- and low-end devices.

For the most part, application QoE correlates with device cost. A cheaper device provides poorer performance. The only outlier is the Pixel2, which outperforms the SG S6-edge despite being less expensive. The underlying reason for this difference is how these two phones use big and little cores in the big.LITTLE architecture to trade between performance and power consumption.

Based on this study, our goal is to (i) understand why video applications are not affected by low-end phones and transfer the lessons learned to the Web, and (ii) study which hardware component has the greatest effect on performance both for Web and video applications to inform future hardware design.

3 IMPACT OF DEVICE PARAMETERS

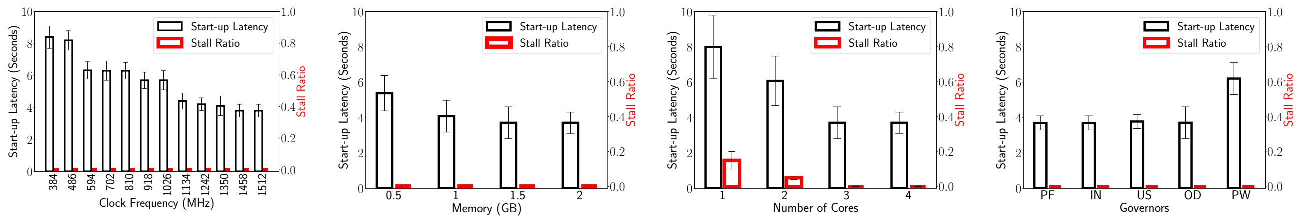
Four device parameters related to available resources (Table 1) can potentially impact application performance – CPU clock, memory capacity, number of cores, and Android governor. The first three parameters are self-explanatory. The Android governor is a set of scaling algorithms used by Android to change the clock frequency based on the CPU utilization and battery life. We observe four common frequency governors used by most Android phones: the Ondemand (OD), Powersave (PW), Interactive (IN), and Performance (PF) governors, each with a different trade-off between power and performance [12].

Experimental Setup: The effect of a given resource is isolated by changing its value while keeping the remaining setup constant. We change the clock, number of cores, and governors using ADB commands on a rooted phone. We change the memory capacity by creating RAM disks [19] from available memory and assigning these RAM disks to the application. The experiments are repeated over three phones—the Pixel2, Intex Amaze+, and Nexus4. These three phones were chosen to represent a high-end, low-end, and medium-end phone. We present the results from Nexus4 in detail and summarize the results from the other two for brevity. Similarly, for brevity, we present the PLT results only for the Chrome browser. We have experimented with two other browsers (Firefox and Opera Mini), which qualitatively have the same experience.

Figures 3-5 show the impact of these parameters – CPU clock, memory, number of cores, and governors on Google Chrome, YouTube, and Skype. The experimental setup is the same as that of §2.1.

3.1 QoE of Web Browsing

The PLT increases by 4× when the CPU clock frequency drops from 1512 MHz to 384 MHz (Fig. 3a). This trend is similar to that of Fig. 2a, where the page loads much slower on low-end devices than on higher-end devices. This performance degradation is due



(a) Clock Frequency (b) Memory (c) Number of Cores (d) Governors

Figure 4: Impact of device parameters on video streaming (YouTube)

to two reasons: a slower clock results in (1) slower page processing (e.g., parsing, scripting, rendering, and painting) and (2) slower packet processing (§4.2) that in turn slows down the downloading of objects.

We estimate the time on the critical path involving computation and network activities using the WProf tool [23, 36]. Network time on the critical path increases from an average of 2 seconds when the clock speed is 1512 MHz to 6 seconds when the clock speed is decreased to 384 MHz – a 66% increase. Compute time increases by 76% for the same CPU slowdown.

We find that compute time increases even more compared to the network time for more complex Web pages. We further dissect the computational activities to find the root cause of the application bottlenecks. We observe that scripting time increases the most as the CPU clock slows down; it accounts for 51% of the overall compute times at high CPU frequencies and 60% at slower CPU frequencies. The layout and painting times account for only 4% of the compute time on the critical path. To confirm the impact of slower Javascript execution, we experiment with different categories of Web pages (e.g., business, health, shopping, news, and sports) and find that news and sports Web pages are affected the most (about 6×) because they have more scripting than the other categories.

Apart from the clock frequency, the PLT is not affected by other parameters significantly. For example, the PLT increases by about 2× when memory is reduced from 2GB to 512MB. The PLT increases by roughly 50% when the Powersave governor is used relative to the others. This is because this governor prefers the slowest clock to trade off performance for power savings.

PLT changes only modestly when the number of cores is reduced from four to one. This is because the browser does not exploit the thread-level parallelism on multi-core mobile devices. We confirm this observation by measuring the CPU utilization across cores and find that, during Web page loads, only two of the cores are utilized irrespective of the number of cores available (see Fig. 3c).
`Takeaway1:` Web browsing underutilizes the multiple cores and suffers significantly with slower CPU clock. A key component of improving Web page loads, especially with a slow CPU clock, is to improve the efficiency of scripting.

3.2 QoE of Video Streaming

Fig. 4 shows the start-up latency (network-centric) and stall ratio (device-centric) metrics of YouTube for the four device parameters on Nexus 4. The start-up latency increases from 1.2 to 3.5 seconds as the clock speed decreases; however, there is no impact on the stall ratio. This trend is similar to the one observed in Fig. 2b across low-end and high-end phones.

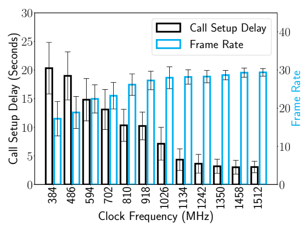
In practice, the stall ratio is a more important QoE metric because the start-up latency is only a one-time effect. The stall ratio is not affected by a slow CPU even though network throughput drops when the CPU is slow. The reasons for this are several video-specific optimizations: i) most smartphones (even low-end phones) support hardware-based video coding. The video coding is offloaded to dedicated hardware accelerators and are not bottlenecked by a slow CPU. Moreover, YouTube serves device-specific video content (e.g., it does not stream FullHD video on an Intex phone). ii) after video decoding, the post-processing tasks such as muxing and demuxing of audio and video indeed happen on the CPU, which could potentially be impacted by a slower clock. The Android multimedia framework is highly parallelized and exploits multiple cores, unlike Web browsing, and thus, the impact of the slower clock is not prominent. We confirm this observation by measuring the CPU utilization during the video experiments across cores. Figure 4c further shows that the performance of video applications degrades as the number of cores decreases. There is an increase of 4 seconds in start-up latency as well as a 15% increase in the stall ratio under a single core. iii) YouTube and other streaming services [10, 24] prefetch video content; YouTube prefetches 120 seconds' (called read-ahead time) worth of content. Therefore, even under slower clocks, the read-ahead time is reached within 40 seconds of the video start-up, resulting in zero stalls.

For memory and governors, YouTube has a similar trend in start-up latency as Web browsing does, with zero stalls.
`Takeaway2:` Specialized coprocessors reduce the role of the general-purpose CPU for video streaming. To the extent that the CPU is used, multiple cores can be exploited. Thus, the impact of low-end phones is largely masked for the QoE of video streaming because even low-end phones have at least two cores and specialized coprocessors.

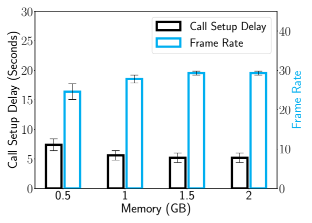
3.3 QoE of Video Telephony

The key difference between streaming and telephony is that telephony is interactive. This means that, unlike streaming, video frames cannot be prefetched by the application. We measure the QoE with the call setup delay (network-centric) and frame rate (device-centric) metrics as described in §2.1. Fig. 5 shows the effect of device parameters on QoE during the Skype video call.

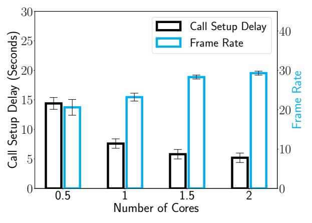
We observe an 18-second increase in call setup delay when the CPU clock drops from 1512 MHz to 384 MHz. This effect is due to the increase in network packet processing caused by slow CPU speeds because the external network condition remains the same. The frame rate drops to 17 frames per second (fps) at slow CPU speeds from 30 fps at high CPU speeds. The decreased frame rate occurs despite the fact that video coding is offloaded to hardware



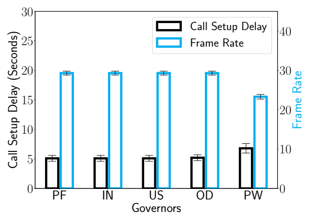
(a) Clock Frequency



(b) Memory



(c) Number of Cores



(d) Governors

Figure 5: Impact of device parameters on video telephony (Skype)

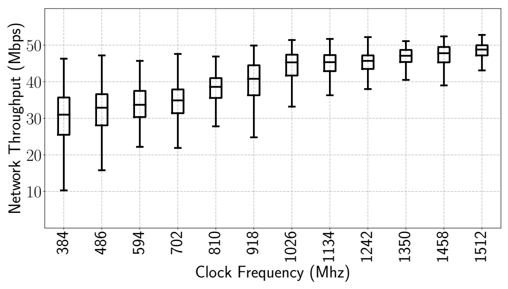


Figure 6: Impact of clock frequency on the network.

similar to video streaming. This discrepancy is due to two reasons: first, unlike video streaming, there is no prefetching. Therefore, packet processing in the kernel stack becomes a bottleneck. Second, video telephony is interactive in that it requires both sending and receiving live video. During this, it requires encoding, decoding, muxing, and demuxing of the audio and video (recall that video streaming has only decoding and demuxing). Although most of the coding is offloaded, the post-processing is limited by poor CPUs. Apart from the clock, Skype has similar trends as YouTube with other parameters — memory, number of cores, and governors.

Interestingly, the Skype adaptive bitrate (ABR) algorithm is more aggressive than YouTube. Skype's ABR algorithm [30] changes the call video quality for slow CPUs (as it does for poor network conditions) when the software perceives poor throughput. In effect, the client requests low-resolution videos under slower clock frequencies.

`Takeaway3:` The key takeaway is that video telephony is linearly affected by slower CPU speeds mainly because of the packet processing overhead. This is different from video streaming, where the effect of the network processing is masked by prefetching.

4 DISCUSSIONS

In this section, we discuss (a) the implications of clock frequency on network throughput and (b) a possible Web page optimization for low-end devices based on our results in §3.

4.1 Impact of Clock Frequency on Network

We find that clock frequency not only affects application processing *but also has a second-order effect on network throughput because of slow packet processing,* which in turn impacts application performance. While packet processing overheads in the transport layer are known to cause performance bottlenecks and have been investigated in the data center context, including the use of kernel bypass and specialized NIC-level processing (e.g., [22]), little attention has been paid to this aspect in the context of mobile applications.

To demonstrate the impact, we conduct a study using the IPerf tool [16] from a server that generates continuous traffic to the Nexus4 smartphone. We measure the average throughput over 5 minutes and repeat the experiment 20 times for 12 clock frequencies. Fig. 6 shows the effect of clock frequency on network throughput. When the clock frequency is reduced from 1512 MHz to 384 MHz, the average throughput drops from 48 Mbps to 32 Mbps. This decrease in throughput is entirely internal to the device. Recall (§2.1) that we host the content on our LAN. The reason for the decreased TCP throughput is that packet processing is computationally intensive, and a slow CPU increases the packet processing time.

This second-order effect has significant implications, especially for Web and Video telephony applications. As we discussed in §3, these applications perform poorly under slow CPU speeds partly because of the TCP processing delays.

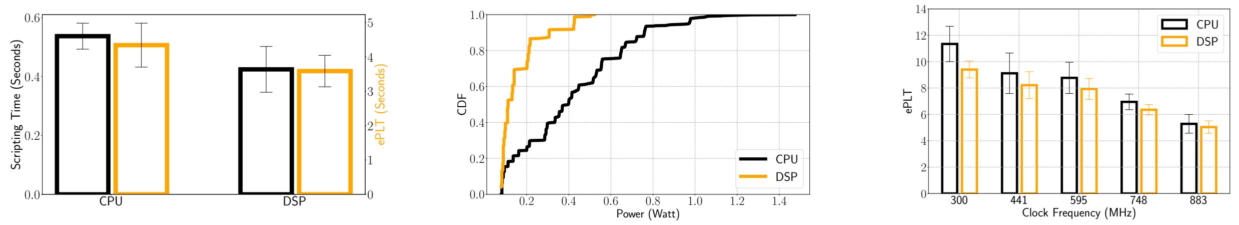
`Takeaway4:` A takeaway is that we require research on improving TCP processing, not only in the context of data centers but also in the context of mobile applications.

4.2 Accelerating Web Page Load

Based on the lessons learned from video applications, we explore how offloading computation to a coprocessor may improve the performance of Web page loads under slower clocks. Many modern mobile phones include GPUs, DSPs, and other specialized hardware accelerators. We study the effect of offloading Web computations to a DSP. To this end, we examine the computation performed on the CPU during Web page loads and identify that Javascript execution is a major time-consuming component. We then drill down into the execution of the script functions for the slowest set of Web pages in our study (news and sports pages) and find that a significant fraction (20% of scripting time) of the page load time is spent in regular expression evaluation (e.g., for URL matching and list operations). This makes a case for exploring the possibility of offloading regular expression evaluation to the DSP.

We conducted our analysis by offloading Javascript regular expression functions with the help of the Qualcomm Hexagon SDK [27]. We converted the regular expression functions from Javascript into direct C-language calls and ported the functions to the *aDSP* processor of the Google Pixel 2 phone (which has the Snapdragon 835 Application processor). The communication between the CPU and DSP was performed using FastRPC remote procedure calls.

We used `Node.js` to measure the runtime performance of the offloaded functions and analyzed their impact on Web page load times. To do this, we extracted the page dependency graphs with



(a) Javascript execution (left axis) and emulated page load times (right axis)

(b) CDF of power consumption during Javascript execution

(c) Emulated page load times with and without DSP offloading at low clock frequencies

Figure 7: Evaluations for DSP offloading of Javascript functions

WProf [36], which preserves the dependency and computation timing information of the entire Web page load process. We then derived the emulated page load time (ePLT) by re-evaluating the WProf dependency graphs after replacing the execution time of all functions that contain the offloaded regular expressions with their measured run times on the DSP.

Fig. 7 shows the impact of offloading regular expressions for the top 20 sports Web pages. Offloading just these functions to the DSP provides a noticeable improvement in the Web page load times when the mobile device is run with the default frequency governors, where the CPU frequency is set by the OS (Fig. 7a). Moreover, we observe an even greater improvement—an almost 4× reduction—in median power consumption (Fig. 7b). As expected, the page load time improvements caused by offloading are largest (up to 25%) when the Web page is loaded at slower CPU frequencies (Fig. 7c).

The potential improvement with this new approach depends on the code patterns that are suitable for DSPs. Note that DSPs can be inexpensive (≈\$10 [13]), so adding them does not significantly increase the cost of mobile devices. In fact, even current low-end phones have many domain-specific hardware accelerators. Our approach can be employed on either server side or client side in web browsers if the browsers allow low-level code execution. However, there is a trade-off. Allowing low-level code on the browser side introduces security vulnerabilities. Situating the DSP optimization on the server side may introduce larger binaries than the original Javascript code, which may adversely affect page load time.

`Takeaway5:` Our results suggest that offloading the computationally intensive parts of Web browsing to coprocessors has potential, especially for low-end phones, and should be further explored.

5 RELATED WORK

Web Performance: Extensive literature on characterizing and improving Web performance exists. WProf [36] and WProf-M [23] characterize the bottleneck of desktop and mobile browsing using page-load dependencies. The key observation in these works is that the network is the bottleneck in desktop browsing, whereas computation is the bottleneck in mobile browsing. Polaris [25] and Vroom [29] are designed to improve Web performance by prioritizing network object loads taking into account dependencies. Shandian [37] and Prophecy [26] use a Web proxy to improve page-load performance. Though these methods optimize network activities to improve page loads, recent works including Webcore [40] and GreenDroid [11] optimize the mobile hardware architecture to improve PLT and minimize energy consumption. A preliminary analysis of the device hardware on Web browsing is shown in [8][20].

Video Performance: Similar to Web browsing, considerable work has addressed improving video QoE focusing on network resource provisioning [14, 39]. Pytheas [18] and CS2P [32] propose data-driven approaches to study the impact of different parameters that impact QoE. They show that the QoE can be largely improved by adapting the bitrate using data-driven throughput prediction. Huang *et.al.* [15] consider client playback buffer occupancy rate adaptation, unlike network-only solutions [6, 17, 35].

Different from these works, our studies focus on understanding the impact of device parameters on Web and video applications.

6 CONCLUSIONS AND FUTURE WORK

In this work, we analyze the impact of device hardware on key mobile Internet applications – Web browsing (Google Chrome), video streaming (YouTube), and video telephony (Skype). Our study uses seven different smartphone devices with a range of capabilities and widely different costs – from \$60 to \$800. We observe that Web applications are adversely affected by low-end device hardware, but video applications, especially streaming, are only modestly affected by low-end hardware. This is largely because video applications offload video decoding to a hardware accelerator and do not rely on the CPU. The needed hardware accelerators are available even on low-end phones. Video applications also parallelize their tasks across multiple cores available in low-end phones, and they are not significantly affected under slow clock speeds. Based on the lessons learned from studying video QoE, we explore the usefulness of offloading Web browsing tasks to a coprocessor. Our preliminary analysis after offloading regular expression evaluations in Javascript to a low-power DSP shows an improvement of 18% in Web page load time along with a 4× reduction in energy consumption.

Our study highlights the impact of device-side performance on mobile applications. While we have studied only hardware parameters, a comprehensive future study should also include software parameters such as OS and browser versions and TCP and TLS overheads in the network stack. Also, studying the joint impact of network conditions and device-side parameters will be useful.

ACKNOWLEDGEMENTS

We thank our shepherd Narseo Vallina-Rodriguez and the reviewers for their feedback, which greatly improved the presentation of this paper. This work is partially supported by NSF grants CNS-1718014 and CNS-1566260, a grant from MSIT, Korea under the ICTCCP Program, and a Google Research Award.

REFERENCES

[1] Smartphone Stats 2017. [n. d.]. https://www.apple.com/iphone-xs/.

[2] ADB. [n. d.]. developer.android.com/tools/help/adb.html.

[3] YouTube Player API. [n. d.]. https://developers.google.com/youtube/android/.

[4] Http Archive. [n. d.]. https://httparchive.org/reports.

[5] Android View Cient (AVC). [n. d.]. github.com/dtmilano/AndroidViewClient.

[6] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a predictive model of quality of experience for internet video. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 339–350.

[7] Jorge L Contreras and Rohini Lakshané. 2017. Patents and Mobile Devices in India: An Empirical Survey. *Vand. J. Transnat'l L.* 50 (2017), 1.

[8] Mallesham Dasari, Conor Kelton, Javad Nejati, Aruna Balasubramanian, and Samir R Das. 2017. Demystifying Hardware Bottlenecks in Mobile Web Quality of Experience. In *Proceedings of the SIGCOMM Posters and Demos*. ACM, 43–45.

[9] Mallesham Dasari, Shruti Sanadya, Christina Vlachou, Kyu-Han Kim, and Samir R. Das. 2018. Scalable Ground-Truth Annotation for Video QoE Modeling in Enterprise WiFi. In *Quality of Service (IWQoS), 2018 IEEE/ACM 26th International Symposium on*. ACM/IEEE, 1–6.

[10] HBO Go. [n. d.]. https://play.hbogo.com/.

[11] Nathan Goulding-Hotta, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Po-Chao Huang, Manish Arora, Siddhartha Nath, Vikram Bhatt, Jonathan Babb, et al. 2011. The greendroid mobile application processor: An architecture for silicon's dark future. *IEEE Micro* 31, 2 (2011), 86–95.

[12] https://android.googlesource.com/kernel/common/+/android 4.4/Documentation/cpu-freq/governors.txt. [n. d.]. Android Governors.

[13] http://www.ti.com/processors/dsp/overview.html. [n. d.]. DSPs.

[14] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of IMC*. ACM, 225–238.

[15] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[16] IPerf. [n. d.]. https://iperf.fr/.

[17] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. 2016. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 286–299.

[18] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation.. In *NSDI*, Vol. 1. 3.

[19] RAM Disks. Linux. [n. d.]. https://kerneltalks.com/linux/how-to-create-ram-disk-in-linux/.

[20] Sumit Maheshwari, Dipankar Raychaudhuri, Ivan Seskar, and Francesco Bronzino. 2018. Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications. In *Proceedings of the Third ACM/IEEE Symposium on Edge Computing*. ACM/IEEE.

[21] Manycam. [n. d.]. https://manycam.com/.

[22] Akshay Narayan, Frank Cangialosi, Prateesh Goyal, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. 2017. The Case for Moving Congestion Control Out of the Datapath. In *Proceedings of Hotnets*. ACM, 101–107.

[23] Javad Nejati and Aruna Balasubramanian. 2016. An in-depth study of mobile browser performance. In *Proc. WWW 2016*. 1305–1315.

[24] Netflix. [n. d.]. https://www.netflix.com/.

[25] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking.. In *NSDI*. 123–136.

[26] Ravi Netravali and James Mickens. 2018. Prophecy: Accelerating Mobile Page Loads Using Final-state Write Logs. In *15th USENIX NSDI 18*. USENIX Association.

[27] Qualcomm Development Network. [n. d.]. https://developer.qualcomm.com/software/hexagon-dsp-sdk/tools.

[28] Az Screen Recorder. [n. d.]. https://az-screen-recorder.en.uptodown.com/android.

[29] Vaspol Ruamviboonsuk, Ravi Netravali, Muhammed Uluyol, and Harsha V Madhyastha. 2017. Vroom: Accelerating the Mobile Web with Server-Aided Dependency Resolution. In *Proceedings of SIGCOMM*. ACM, 390–403.

[30] Iraj Sodagar. 2011. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia* 18, 4 (2011), 62–67.

[31] Moritz Steiner and Ruomei Gao. 2016. What slows you down? Your network or your device? *arXiv preprint arXiv:1603.02293* (2016).

[32] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 272–285.

[33] Tesseract. [n. d.]. https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/.

[34] Chrome Developer Tools. [n. d.]. https://chromedevtools.github.io/devtools-protocol/.

[35] Naresh Vattikuti, Mallesham Dasari, Himanshu Sindhwal, and Bheemarjuna Reddy Tamma. 2015. Towards bandwidth efficient TDMA frame structure for voice traffic in MANETs. In *Electronics, Computing and Communication Technologies (CONECCT), 2015 IEEE International Conference on*. IEEE, 1–6.

[36] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf.. In *NSDI*. 473–485.

[37] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. 2016. Speeding up Web Page Loads with Shandian.. In *NSDI*. 109–122.

[38] Alexa Websites. [n. d.]. https://www.alexa.com/topsites.

[39] Fatima Zarinni, Ayon Chakraborty, Vyas Sekar, Samir R Das, and Phillipa Gill. 2014. A first look at performance in mobile virtual network operators. In *Proceedings of IMC*. ACM, 165–172.

[40] Yuhao Zhu and Vijay Janapa Reddi. 2017. Optimizing General-Purpose CPUs for Energy-Efficient Mobile Web Computing. *ACM Transactions on Computer Systems (TOCS)* 35, 1 (2017), 1.