Impact of Device Performance on Mobile Internet QoE

Mallesham Dasari, Santiago Vargas, Arani Bhattacharya, Aruna Balasubramanian
Samir R. Das, Mike 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. In order to bridge this gap we study the QoE of three popular applications: Web browsing, video streaming, and telephony, for different device parameters. Our results demonstrate that the Web applications are much more sensitive to low-end hardware compared to 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 coprocessors and multiple cores. Whereas, Web browsing is largely influenced by clock frequency and uses no more than 2 cores. This makes Web browsing more vulnerable to performance on low-end smartphones. Based on the lessons learnt from studying video applications, we explore offloading Web computation to a co-processor. Specifically, we explore offloading of regular expression computation to a DSP coprocessor and show an improvement of 18% in page load time while saving the energy by a factor of 4.

1. INTRODUCTION

Mobile smartphones have now penetrated a significant fraction of world 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 8 cores, running up to 2.4 GHz clock frequency and 6 GB RAM, while a cheaper $60 phone (e.g., Dell Venue Pro) only has 2 cores with up to 1 GHz clock frequency and 512 MB RAM.

A natural question arises: how much of an application's QoE depends on the phone's hardware specs given that widely different phones with very different price points are available in the market. This question is specifically important given that it is well known that *compute* is a key performance bottleneck for mobile applications such as browsing [19, 29]. However, it is not clear which aspect of compute/hardware specification is significant to performance. Knowing which hardware component has the most impact on end-user performance is crucial to designing better phones under a budget.

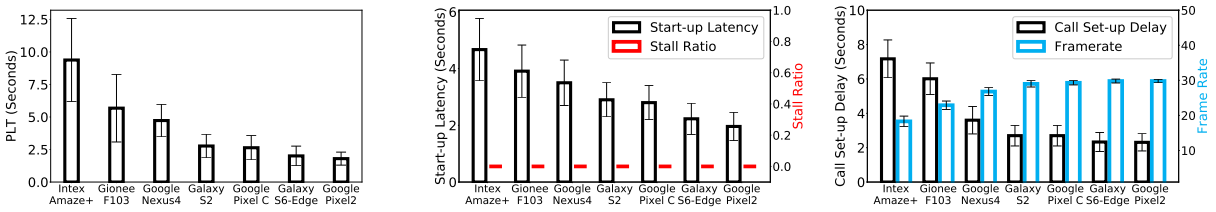
The problem is more acute among low-end phones. As an example, our results show that mobile Web page loads on two popular phones in India, Intex Amaze 4 (≈$60) and Gionee (≈$150) are **5**× to **3**× worse, respectively, compared to Web page loads on Google Pixel2 (≈$700) under the same network conditions (§2).

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 to achieve a good trade off between application performance and power consumption.) Our goal is to understand how each of these device parameters affect 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. On the other hand, changing the number of cores affect 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 only not only *how* the hardware component affects application QoE but also *why*.

Our key finding is the Web performance is affected by low-end phones. As a first step, Web browsing is significantly affected by slower clock speeds. Web page loads slow down by $5\times$ when clock frequency reduces from 1512 MHz to 384 MHz. Interestingly, video applications are largely unaffected by slowing the clock even though video processing is a compute-intensive operation. This is because video decoding uses dedicated hardware decoders, available even on low-end phones.

Further, video applications use parallel operations among multiple CPU cores for post-processing (such as muxing and demuxing of audio/video). Web applications do not use multiple cores effectively. The result is that the performance of video applications are less affected by low-end phones, but



(a) Web Browsing (b) Video Streaming (c) Video Telephony

Figure 1: 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 Cost
Intex Amaze+	Spreadtrum SC9832A	4	6.0	300-1300	Mali-400	1	$60
Gionee F103	MediaTek MT6735	4	5.0	300-1300	Mali-T720	2	$150
Nexus4	Snapdragon S4 Pro	4	5.1.1	384-1512	Adreno 320	2	$200
SG S2-Tab	Exynos 5433	8	5.0.2	400-1300	Mali-T760	3	$450
Pixel-Tab	Tegra X1	4	8.0.0	204-1912	Maxwell	3	$600
Pixel2	Snapdragon 835	8	8.0.0	300-2457	Adreno 540	4	$700
SG S6-edge	Exynos 7420	8	6.0.1	400-2100	Mali-T760	3	$880

Table 1: Mobile devices used in our experiments and their corresponding specifications including cost, CPU and memory capacity.

the performance of Web applications are severely affected.

Finally, we find that leveraging hardware offloading is a promising alternative to improving Web performance under slow CPU clock. Our preliminary analysis with offloading only regular expression evaluations shows an improvement of 18% in page load time along with a $4\times$ reduction in energy consumption.

2. QOE ACROSS LOW AND HIGH-END DEVICES

As a first step, we study the performance of the three Internet applications—Web browsing, Video streaming, and Video telephony across 7 different smartphones. The phones are chosen so that there is a 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 default experimental setup before going over 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 [33]. We load the top 50 Web pages from Alexa [35], clear the cache and estimate the average PLT. We use the WProf tool [33] to analyze the critical path of the page load process and break down the critical path into compute and network activities. Compute activities include HTML parsing, Javascript evaluation, and rendering. Network activities refer to loading the objects. We automate the page loads for repeatability using Chrome remote debugging protocol [32] over Android Debug Bridge (ADB) [2].

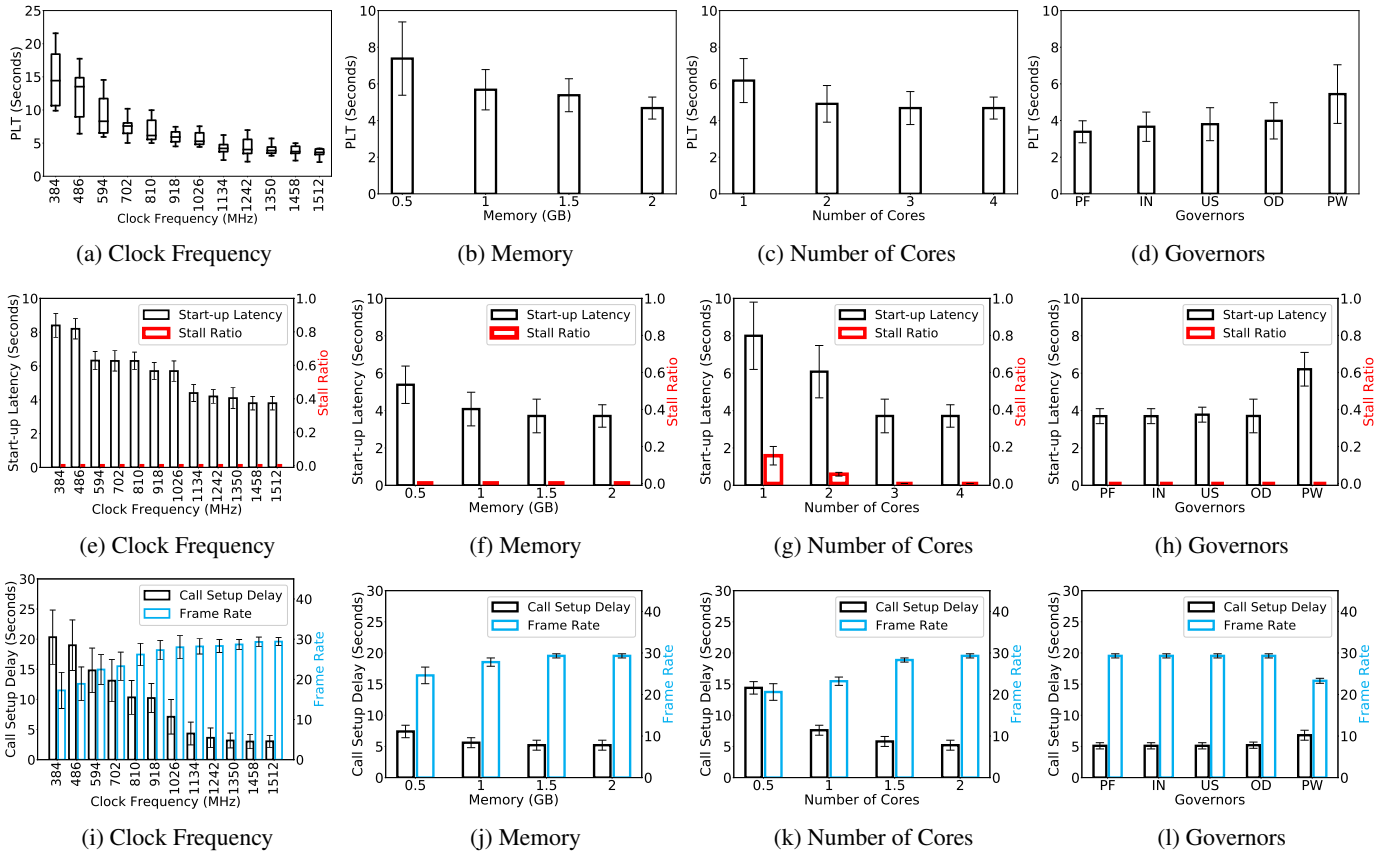
Video Streaming: We use **YouTube** to measure video streaming performance using two QoE metrics: *start-up latency* (network-centric) and *stall ratio* (device-centric). Start-up latency is the time from when the request was issued to when the the application starts displaying frames. Stall ratio is 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 minute FullHD (1080p) video clip. We use ADB [2] to programmatically request the video content for repeatability.

Video Telephony: We use **Skype** to measure performance of video telephony. We measure QoE in terms of *call setup delay* (network-centric) and *frame rate* (device-centric) metrics. Frame rate is measured as the number of frames shown per second and call setup delay is time it took for the client to get the response once the receiver answers the call. As Skype does not provide APIs to extract the QoE, we screen record [24] the Skype technical information and extract the frame rate using an OCR tool [31].

Since Skype is an interactive application, it requires an active participant on both ends. When the Skype call is placed from the mobile to a laptop, the laptop runs a virtual Webcam [17] that plays a video file in Skype instead of 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 [4].

Network Setup: As the focus of this work is to measure the impact of the device hardware, the experiments are set up 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 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.

2.2 QoE Across Devices



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

(e) Clock Frequency (f) Memory (g) Number of Cores (h) Governors

(i) Clock Frequency (j) Memory (k) Number of Cores (l) Governors

Figure 2: Impact of device parameters on Web browsing (a-d), video streaming (e-h) and telephony (i-l).

Fig. 1 shows the performance of the three applications across the devices. Based on the device model, there is a significant difference in performance even though all experiments are done in the same network conditions.

For Web page loads (Fig. 1a), there is a 7 seconds 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 Intex Amaze+ compared to Pixel 2. This must stem from the device itself, since the network condition remain unchanged.

In the case of YouTube (Fig. 1b), there is a linear increase 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, once 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. 1c), frame rate decreases from 30fps to 18fps between the high- and low-end devices.

For the most part, application QoE is correlated with the device cost. A cheaper device provides poorer performance. The only outlier is Pixel2 which outperforms SG S6-edge despite being less expensive. The underlying reason for this difference is how these two phones uses 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 learnt to the Web, and (ii) Study which hardware component has the most 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 observed four common frequency governors used by most of Android phones: Ondemand, Powersave, Interactive and Performance governors each with a different trade-off between power and performance. More details of Android governors can be found at [10].

Experiment set up: The effect of a given resource is isolated by changing its value while keeping the remaining set up constant. We change the clock, number of cores and governors using ADB commands on a rooted phone. We change memory capacity by creating RAM disks [16] from available memory and assigning these RAM disks to the application. The experiments are repeated over three phones—Nexus 4, Intex Amaze+, and Pixel 2. These three phones were chosen to represent a high-end, low-end, and medium-end phone. We present the results from Nexus 4 in detail, and summa-

rize the results from the other two for brevity.

Fig. 2 shows the impact of these parameters —CPU clock, memory, number of cores and governors on Google Chrome, YouTube and Skype. The measurement setup is identical to §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. 2a). The trend is similar to Fig. 1a where the page loads much slower on low-end devices compared to higher-end devices. This performance degradation is because of two reasons: slower clock results in (1) slower page processing activities (e.g., parsing, scripting, rendering and painting) and (2) slower packet processing (§4.2) in turn slowing down downloading of objects.

We estimate the time on the critical path involving compute and network activities using WProf tool [33, 19]. The 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 increased to 384 MHz – a 66% increase. The compute time increases by 76% for the same CPU slow down.

We find that the compute time increases even more compared to network time for more complex Webpages. We further dissect the compute activities to find the root cause of compute bottleneck at the application. We observe that scripting times increase 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 only account for 4% of the compute time on the critical path. To confirm the impact of slower Javascript execution, we experiment different categories of webpages (e.g., business, health, shopping, news, and sports) and find that news and sports webpages are affected the most (about 6×) which 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 roughly increases by 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 only changes modestly when the number of cores is reduced from 4 to 1. 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.

`Takeaway1:` Web browsing underutilizes the multi-cores and suffers significantly at slower CPU clock. A key component for improving Web page loads, especially at slow CPU clock, is to improve the efficiency of scripting.

3.2 QoE of Video Streaming

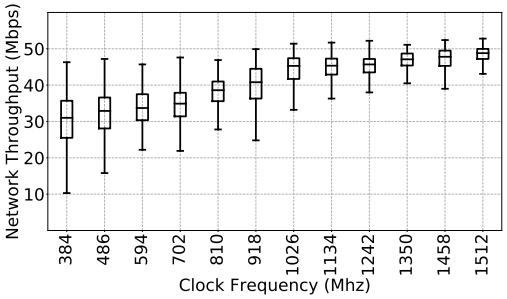


Figure 3: Impact of clock frequency on network.

Fig. 2(e-h) shows the startup latency (network-centric) and stall ratio (device-centric) metrics of YouTube for the four device parameters on Nexus 4. The startup 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. 1b across low-end and high-end phones. Experiments on Intex Amaze+ and Pixel 2 show quantitatively similar trends (not shown here).

In practice, the stall ratio is a more important QoE metric, since the startup latency is only a one-time effect. The stall ratio is not affected under slow CPU, even though network throughput drops when the CPU is slow. The reasons for this are several video-specific optimizations: i) Most of the 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 Intex phone). ii) After video decoding, the post processing such as muxing and demuxing of audio and video indeed happen on the CPU which could potentially be impacted under slower clocks. But, the Android multimedia framework is highly parallelized and exploits multi-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 2g further shows that performance of video applications degrade as the number of cores decreases. There is an increase of 4 seconds in start-up latency as well as 15% of stall ratio under single core. iii) YouTube and other streaming services [20, 8] 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 general purpose CPU for video streaming. To the extent that CPU is used, multiple cores can be exploited. Thus, the im-

pact of low-end phones is largely masked for the QoE of video streaming since even low-end phones have at least 2 cores and specialized co-processors.

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 call setup delay (network-centric) and frame rate (device-centric) metrics as described in §2.1. Fig. 2 (i-l) shows the effect of device parameters on QoE during the Skype video call.

We observe a 18 second increase in call start-up delay when the CPU clock reduces from 1512 MHz to 384 MHz. This effect is due to the increase in network packet processing caused by slow CPU speeds, since 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 is despite the fact that video coding is offloaded to hardware similar to video streaming. This descrepancy is due to two reasons: first, unlike video streaming there is no prefetching. Therefore, packet processing in the kernel stack is becoming bottleneck. Second, video telephony is interactive as in it requires both sending and receiving of the 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). Even though most of the coding is offloaded, the post processing is limited by poor CPUs. Apart from clock, Skype has similar trends as YouTube with other device parameters — memory, number of cores and governors.

Interestingly, the adaptive bitrate (ABR) algorithm used by Skype is more aggressive than YouTube. The ABR algorithm [28] used by Skype changes the video quality during Skype call for slow CPUs (as it does for poor network conditions) since the skype client perceives poor throughput. In effect, the client often requests low resolution videos under slower clock frequency.

`Takeaway3:` The key takeaway is that video telephony is linearly affected by slower CPU speeds mainly due to 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 on network throughput and energy, and (b) a possible Web page optimization for low-end devices based on our observations in §3.

4.1 Impact of Clock Frequency on Network

One of our findings is that the clock frequency not only affects application processing, *but has a second-order effect on network throughput because of slow packet processing.* This in turn impacts application performance as well. While packet processing overheads in the transport layer are known to cause performance bottlenecks and have been well-investigated in the data center context, including use of kernel bypass and specialized NIC-level processing (see, e.g., [18]), there has been little attention to this aspect in the context of mobile applications.

To demonstrate the impact, we do a study using the IPerf tool [13] from a server to the Nexus4 smartphone. IPerf generates continuous traffic, and we measure the average throughput over 5 minutes duration. We repeat the experiment 20 times for 12 clock frequencies. Fig. 3 shows the effect of clock 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 in our set up, we host the content in our LAN. The reason for the decreased TCP throughput is that packet processing is compute 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 and Figures 2(a) and (i), these applications perform poorly under slow CPU speeds partly because of the TCP processing delays.

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

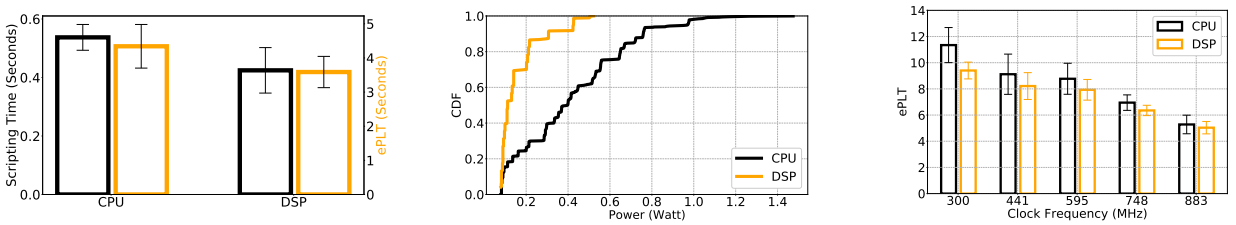
4.2 Accelerating Web Page Load

Based on the lessons learnt from video applications, we explore how offloading computation to a co-processor may improve 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 computation to a DSP.

To this end, we examined the computation performed on the CPU during Web page load and identify that Javascript execution is a major time consuming component. We then drilled down into the execution of the script functions for the slowest set of Web pages in our study (news and sports pages), and found 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 [23]. 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 `nodejs` to measure the runtime performance of



(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 4: Evaluations for DSP offloading of Javascript functions

the offloaded functions and analyzed their impact on the Web page load times. To do this, we extracted the page dependency graphs with WProf [33], 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. 4 shows the impact of offloading regular expression evaluation for the top 20 sports Web pages. We find that offloading just these functions to the DSP already 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. 4a). Moreover, we observe an even greater improvement—almost $4\times$ reduction—in median power consumption (Fig. 4b). As expected, the page load time improvements due to offload are largest (up to 25%) when the Web page is loaded at slower CPU frequencies (Fig. 4c).

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

5. RELATED WORK

Web performance: There is extensive literature on characterizing and improving Web performance. WProf [33] and WProf-M [19] characterize the bottleneck of desktop and mobile browsing respectively using page-load dependencies. The key observation in these works is that the network is the bottleneck in desktop browsing while compute is the bottleneck in mobile browsing. Klotski [6], Polaris [21], and Vroom [25] are all designed to improve Web performance by prioritizing network object loads taking into account dependencies. Shandian [34], Prophecy [22], Nutshell [26], and Parcel [27] use a Web proxy to improve page load performance. While these methods optimize network activities to improve page load, recent works including as Webcore [37] and GreenDroid [9] optimize the mobile hardware architecture to improve PLT and minimize energy consumption.

Video Performance: Similar to Web browsing, there has been considerable work in improving video QoE focussing on network resource provisioning [11, 36]. Pytheas [15] and CS2P [30] 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 bitrate using data-driven throughput prediction. Huang *et.al.* [12] consider client playback buffer occupancy rate adaptation unlike network-only solutions [5, 14].

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

6. CONCLUSIONS

In this work, we analyze the impact of the device hardware on key mobile Internet applications – Web browsing (Google Chrome), video streaming (YouTube), and video telephony (Skype). Our study is motivated by a survey of 7 diverse smartphone devices, ranging from $60 to $800. We find that Web applications are adversely affected by low-end device hardware but video applications, especially streaming, is 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 hardware accelerators are even available on low-end phone. Video applications also parallelize their task across multiple cores available in low-end phones, and therefore the application is not significantly affected under slow clock speeds. Based on the lessons learnt from studying video QoE, we explore the usefulness of offloading Web tasks to a co-processor. Our preliminary analysis with offloading regular expression evaluations in Javascript to a low-power DSP shows an improvement of 18% in page load time along with a $4\times$ reduction in energy consumption.

7. REFERENCES

[1] Smartphone Stats 2017. https://perma.cc/ya27-x5hf.
[2] ADB. developer.android.com/tools/help/adb.html.
[3] YouTube Player API. https://developers.google.com/youtube/android/player/.
[4] Android View Cient (AVC). https://github.com/dtmilano/androidviewclient.
[5] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 339–350. ACM, 2013.
[6] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V Madhyastha, and Vyas Sekar. Klotski:

Reprioritizing web content to improve user experience on mobile devices. In *NSDI*, pages 439–453, 2015.

[7] Jorge L Contreras and Rohini Lakshané. Patents and mobile devices in india: An empirical survey. *Vand. J. Transnat'l L.*, 50:1, 2017.

[8] HBO Go. https://play.hbogo.com/.

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

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

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

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

[13] IPerf. https://iperf.fr/.

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

[15] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *NSDI*, volume 1, page 3, 2017.

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

[17] Manycam. https://manycam.com/.

[18] Akshay Narayan, Frank Cangialosi, Prateesh Goyal, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. The case for moving congestion control out of the datapath. In *Proceedings of Hotnets*, pages 101–107. ACM, 2017.

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

[20] Netflix. https://www.netflix.com/.

[21] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster page loads using fine-grained dependency tracking. In *NSDI*, pages 123–136, 2016.

[22] Ravi Netravali and James Mickens. Prophecy: Accelerating mobile page loads using final-state write logs. In *15th USENIX NSDI 18*. USENIX Association, 2018.

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

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

[25] Vaspol Ruamviboonsuk, Ravi Netravali, Muhammed Uluyol, and Harsha V Madhyastha. Vroom: Accelerating the mobile web with server-aided dependency resolution. In *Proceedings of SIGCOMM*, pages 390–403. ACM, 2017.

[26] Ashiwan Sivakumar, Chuan Jiang, Yun Seong Nam, Shankaranarayanan Puzhavakath Narayanan, Vijay Gopalakrishnan, Sanjay G Rao, Subhabrata Sen, Mithuna Thottethodi, and TN Vijaykumar. Nutshell: Scalable whittled proxy execution for low-latency web over cellular networks. In *Proceedings of MOBICOM*, pages 448–461. ACM, 2017.

[27] Ashiwan Sivakumar, Shankaranarayanan Puzhavakath Narayanan, Vijay Gopalakrishnan, Seungjoon Lee, Sanjay Rao, and Subhabrata Sen. Parcel: Proxy assisted browsing in cellular networks for energy and latency reduction. In *Proceedings of CoNEXT*, pages 325–336. ACM, 2014.

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

[29] Moritz Steiner and Ruomei Gao. What slows you down? your network or your device? *arXiv preprint arXiv:1603.02293*, 2016.

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

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

[32] Chrome Developer Tools. https://chromedevtools.github.io/devtools-protocol/.

[33] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. Demystifying page load performance with wprof. In *NSDI*, pages 473–485, 2013.

[34] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. Speeding up web page loads with shandian. In *NSDI*, pages 109–122, 2016.

[35] Alexa Websites. https://www.alexa.com/topsites.

[36] Fatima Zarinni, Ayon Chakraborty, Vyas Sekar,

Samir R Das, and Phillipa Gill. A first look at performance in mobile virtual network operators. In *Proceedings of IMC*, pages 165–172. ACM, 2014.

[37] Yuhao Zhu and Vijay Janapa Reddi. Optimizing general-purpose cpus for energy-efficient mobile web computing. *ACM Transactions on Computer Systems (TOCS)*, 35(1):1, 2017.