Improving the QoE of Internet-based Mobile Applications

ABSTRACT

Applications for mobile Internet have proliferated with the increase in smartphone usage. Popular applications include Web browsing, video streaming and interactive telephony. A critical performance issue in these applications is quality of experience (QoE) of end-users. It is now widely understood that resource control mechanisms for such applications should optimize for QoE rather than more traditional quality of service (QoS) metrics that capture parameters such as latency or throughput. The reason for this is that these QoS metrics do not always correlate well with the application-specific user experience or may have complex dependencies. In our work, we study the QoE aspects of three popular mobile applications – YouTube for video streaming, Skype for telephony and Google Chrome for Web browsing. Our general goal is to aid resource control mechanisms to optimize for the QoE of applications.

In the first part of our work, we conduct a survey of existing work on QoE optimizations and highlight their limitations. We then propose a machine learning-based QoE modeling approach. The model learns the QoE of applications under diverse network conditions and helps the network administrator to efficiently provision resources to maximize the QoE. The proposed approach improves the model accuracy significantly relative to prior approaches.

In the second part of our work, we perform a detailed study on how underlying device hardware performance may impact the QoE of mobile applications when there are sufficient network resources available. For this study, we microbenchmark the applications with respect to device hardware parameters such as CPU clock speed, memory, GPU and number of CPU cores. We find that Web browsing is the most affected application when hardware resources are limited among the applications studied. The reason is that Web browsing is inefficient in exploiting any of the device accelerators or coprocessors. We show that offloading Web page load related computations opportunistically to the DSP coprocessor can lead to upto 18% improvement in page load times while saving the energy consumption by a factor of 4.

1. INTRODUCTION

The massive use of smartphone Internet applications makes their quality of experience (QoE) increasingly critical in our daily life. As of the year 2018, 51.12% of the global web traffic and 63% of the video traffic is mobile [77]. This is majorly due to the growing availability of diverse mobile Internet applications such as mobile browsing, video streaming, interactive telephony and many other social network applications.

However, with the advent of heterogeneous devices and networks, it has become difficult to uncover if a user is having QoE issues and find the root cuase of any issue. Apart from server-side issues and Internet delays, the users face several issues on client-side ecosystem – the client connected network, underlying device configuration such as OS and hardware, and the inherent application complexity.

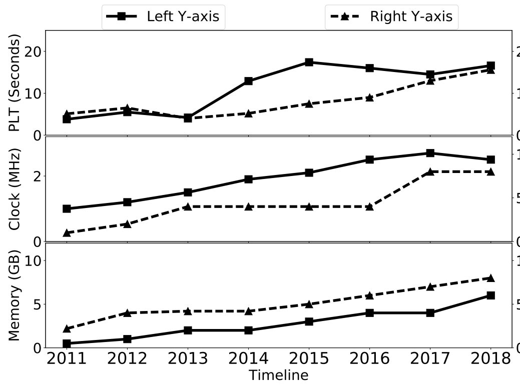


Figure 1: Evolution of webpage performance and device parameters over last 8 years. The growth in device performance is not on par with the growth of application demands.

Traditional network resource provisioning is based on QoS parameters (such as network throughput, latency, and loss) rather than QoE. Also, the smartphone manufacturerers does not consider the QoE of applications while enhancing the device parameters. Because of this, smartphones are not able to cope with the QoE requirements of applications (Fig. 1). Based on the pageload history [11] and device data mined from over 480 Android smartphone specifications over the past 8 years, we find that page load times are increased by 4× despite the improvement in device hardware (clock: 1Ghz to 2.7Ghz, memory: 512MB to 6GB, cores: 2 to 8) and network (3G to 4G and 802.11n to 802.11ac). Therefore, QoE based design trends of network resource provisioning and device hardware enhancement could potentially improve the experience of these applications.

In this work, we aim to improve the QoE of Internet-based mobile applications. To achieve this, we evaluate the QoE of three popular applications — Web browsing, Video Streaming, and Video Telephony. In particular, we study the network- and device-side bottlenecks and make an effort to improve the QoE of these applications.

In the first part of our work, we focus on QoE based network resource provisioning to address network-side bottlenecks. Recent success of using machine learning (ML) in networking enabled several network optimizations for QoE management. For example, network administrators map traditional QoS parameters to QoE using ML algorithms and control network resources according the end-user QoE. To achieve this, the prior work collects the QoE information from applications and corresponding network statistics to train the ML model. However, majority of the existing work rely on applications to get the ground-truth, use inappropriate QoE metrics, not scalable across diverse applications (§2.1). To solve this problem, we systematically study the limitations of prior work. We then propose a generic QoE prediction model by using scalable and content independent objective QoE metrics.

In the second part of our work, we study the impact of different device configurations to find the root cause of device-side bottlenecks. Given that widely different phones with very different price points are available in the market, a natural question arises: how much of an application's QoE depends on the phone's hardware specs. This question is specifically important given that it is well known that compute is a key performance bottleneck for mobile applications such as browsing [60]. 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.

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.

In summary, our contributions and findings are following:

- We identify the limitations of prior work on video QoE modeling. We propose scalable and content independent QoE metrics and study three popular video telephony applications – Skype, Hangouts, and FaceTime with diverse network conditions (§2.3).

- We propose a scalable and application independent video QoE prediction model and achieve a median 90% accuracy (§2.5).

- We microbenchmark the QoE of three popular mobile Internet applications— Web browsing, video streaming, and video telephony with different device parameters. We find that Web browsing is most sensitive to device hardware whereas video applications are exploiting specialized hardware accelerators (§3).

- Taking inspiration from video applications, we offload certain browsing computation onto DSP coprocessor opportunistically and achieve 18.5% improvement in speeding up page loads with 4× reduction in energy consumption (§3.5).

2. VIDEO QOE LABELING

Over the past decade, mobile video traffic has increased dramatically (from 50% in year 2011 to 60% in 2016 and is predicted to reach 78% by 2021) [40]. This is due to the proliferation of mobile video applications (such as Skype, FaceTime, Hangouts, YouTube, and Netflix etc). These applications can be categorized into video telephony (Skype, Hangouts, FaceTime), streaming (YouTube, Netflix), and upcoming virtual reality and augmented reality streaming (SPT [5], theBlu [6]). Users demand high Quality of Experience (QoE) while using these applications on wireless networks, such as WiFi and LTE. This poses a unique challenge for network administrators in enterprise environments, such as offices, university campuses and retail stores.

Guaranteeing best possible QoE is non-trivial because of several factors in video delivery path (such as network conditions at the client side and server side, client device, video compression standard, video application). While application content providers focus on improving the server-side network performance, video compression and application logic, enterprise network administrators seek to ensure that network resources are well provisioned and managed to provide good experience to multiple users of diverse applications. In this pursuit, network administrators can only rely on passive in-network measurements to *estimate* the exact user experience on the end-device. In this context, several prior works have developed a mapping of QoS to QoE for video applications [25, 23, 28], by using machine learning (ML) models and network features.

In order to train any QoS to QoE model, one needs accurate ground-truth annotation of the QoE. To obtain this, prior work has leveraged application specific APIs such as Skype technical info [39] or YouTube API [23]; call duration [17]; or instrumented client-side libraries for video delivery platform [25]. While these solutions perform well for specific applications and providers, network administrators have to *deal with a plethora of video applications* and they *cannot control the application logic* for most applications. Nor does every application expose QoE metrics through APIs. Thus, to develop QoS to QoE models in the wild, we need an application-independent approach to measure QoE. Different applications exhibit diverse artefacts when quality deteriorates. For instance, streaming quality is impacted largely by buffering and stall ratio, whereas telephony quality is impacted by bit rate, frames per second, blocking and blurring in the video.

In this work, we propose a generic video telephony QoE model which does not rely on application support. Additionally, it is scalable to diverse content, devices and categories of video application. We take a similar approach as Jana *et al* [30], where we record the screen on the mobile device and estimate video quality. Different from previous works, we exploit video compression methods (§ 2.2) and identify four new metrics: *perceptual bitrate (PBR)*, *freeze ratio*, *length* and *number of video freezes* to measure the video QoE. We further demonstrate that our metrics are insensitive to the content of the video call and they only capture the *quality* of the video call (§ 2.4). We make a rigorous analysis of our model performance by conducting a large scale user-study of 800 video clips across 20 videos and more than 200 users. We conduct an extensive analysis on different types of users' devices and OS (Android vs. iOS), video content and motion.

We micro-benchmark our metrics under different network conditions to show that the metrics are agnostic to motion and content of the video. Further, we show that these metrics capture spatial and temporal artefacts of video experience. We then validate our metrics by mapping them to actual users' experience. To this end, we obtain Mean Opinion Score (MOS) from our user-study and apply ML models to map our metrics with users' MOS. We use `Adaboosted` decision trees in predicting the MOS scores, as we describe in §2.4.

2.1 Motivation for Scalable QoE Annotation

In this section, we describe the need for new QoE metrics for enterprise networks and the limitations of existing work.
Lack of QoE information from applications: While several works have motivated and addressed the problem of network-based QoE estimation [23], little attention has been paid to the problem of collecting the QoE ground-truth. Most works have relied on application-specific information. This approach is effective for QoE optimizations by content

providers, as they only focus on a single application [25] or initial training of QoS to QoE model [23]. Nevertheless, administrators of access networks have to ensure good experience for a wide range of diverse applications being simultaneously used. Table 1 shows that not all popular video applications provide QoE information. Even for applications like Skype, availability of technical information depends on the version of the application and on the OS (e.g., no technical information for iOS).

To apply QoE estimation models in real networks, we need to remove the dependency of QoE metrics on application features, such as Skype technical information. One way to achieve this is to simply record the video as it plays on the mobile device and analyze this video for quality.

Lack of scalable and reliable QoE measures: Prior work on video quality evaluation leverages *subjective and/or objective* metrics. Subjective metrics are measured with MOS collected through user surveys. They capture absolute QoE but are tedious to conduct and to scale. QoE metrics need to scale to thousands of videos in order to train models that map QoS to QoE. Alternatively, objective metrics can be computed from the video. Objective metrics are further classified in two categories: reference and no-reference based. A reference-based metric uses both sent and received video, and compares the quality of sent vs. received frames. As it is challenging to retrieve and synchronize reference videos for telephony applications, no-reference based quality metrics are preferred. Jana *et al* [30] have proposed a no-reference metric for QoE estimation in Skype and Vtok. They record received videos for each of these mobile telephony applications and compute three no-reference metrics: *blocking, blurring* and *temporal variations*. Then, they combine these three metrics into one QoE metric by using MOS from subjective user study. Their study shows that blocking does not impact MOS of a video clip. While they show that their *blur* and *temporal variation* metrics correlate well with MOS, they do not evaluate these metrics over a wide range of clips.

We conduct similar experiments with Skype to evaluate blur metric of prior works. Our experimental setup is described in Section 2.3. To capture video blur, Jana *et al* [30] employ discrete cosine transform (DCT) coefficients [31] from the compressed data by computing the histogram of DCT coefficients thereby characterizing the image quality. The DCT coefficients are obtained in transform coding of video compression process, as described in Section 2.2. The assumption here is that blurred image has high-frequency coefficients close to zero. Hence, the method studies the distribution of zero coefficients instead of absolute pixel values. Although the method estimates out-of-focus blur accurately, it falls short in estimating realistic blur and sensitivity to noise. The authors also point out that the method is very sensitive to uniform background and images with high luminance components. In Fig. 2, we evaluate blur for 20 video sequences[1] using DCT metric. We have collected these videos in a representative manner to cover diverse content, different types of motion and we have downloaded them in Full HD resolution. The same 20 videos are converted to low quality by compressing and decreasing the resolution, to observe the difference of DCT metric between high and low quality videos.

[1]The videos are located at *https://gofile.io/?c=iyfQND*.

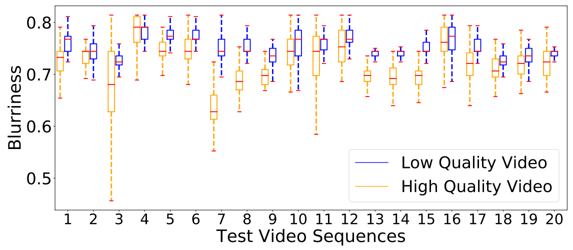


Figure 2: Blur detection using DCT coefficients [30, 31]. DCT coefficients fail to distinguish between high and low quality video due to content diversity.

We notice two aberrations of the DCT metric by Jana *et al* [30]: First, the metric is indeed content-specific i.e., although it produces high blur values for some low quality videos, it also shows high blur values for some high quality videos. For instance, videos 2 and 4 contain mostly over-illuminated and dark images and are equally tagged as blurred in both low and high quality scenarios. Second, DCT metric fails to detect accurate blur levels, even if the image is blurred heavily i.e., it shows low blur differences (<0.2 blurriness) between high and low quality videos even though we created extremely low-quality videos (240p resolution and high ($>200\%$) compression). Even for a single video, this DCT metric shows a lot of variation, such as for videos 3 and 11, raising concerns about its accuracy. We evaluate the accuracy of this blurriness metric in Section 2.5, showing a MOS error larger than 1.2. We also experiment with four other well-known blur metrics [29, 33, 36, 32], but none of these methods are consistent across diverse videos. This challenges the scalability and versatility of this blur metric in the wild.

The *temporal variation* metric proposed by Jana *et al* [30] aims to capture video stalls and considers the ratio of missed frames to total number of frames in a video, but the metric requires the number of frames sent over the network. We need the reference video to compute the total number of frames in the video, thus the metric is not entirely no-reference. In enterprise networks, a QoE metric needs to be applicable to diverse contents and to not rely on access to reference video. Further, there are many other previous works [87, 26, 37, 34] focused on measuring the temporal jerkiness. We find these methods either are parametrized or are sensitive to resolution and to frame rate of the video or are unable to scale across diverse video contents. The above limitations motivate us to propose new metrics that can accurately measure blurriness and freezes across diverse video content.

Need for a model per application category: When analyzing a recorded video for an application, the QoE metric has to be sensitive to the application category as different applications have to meet diverse performance guarantees. For instance, streaming quality is impacted largely by buffering and stall ratio, whereas telephony quality is impacted by bit rate, frames per second, blocking and blurring in the video. Table 2 lists examples of QoE metrics corresponding to different applications. Therefore, one needs to capture different artefacts when designing models for multiple application categories.

To address the aforementioned requirements, we seek to answer the question: *How to scalably label the quality of a video call, without any support from the application?*

Application	Ground-truth
Skype	✓*on some versions
Hangouts/Duo	×
FaceTime	×
YouTube/Netflix	✓

Table 1: Availability of QoE ground-truth

2.2 Background

Before designing our no-reference QoE metrics, we present a video coding primer and QoE artefacts in video telephony. In Section 2.3, we harness these coding properties to carefully craft accurate metrics that capture such artefacts.

2.2.1 *Video Coding Primer*

Video coding is performed in three critical steps:

- **Frame prediction:** The encoder takes images of video and divides each image into macroblocks (typically 16x16, 16x8, 8x16, 8x8, 4x4 pixel blocks). The macroblocks are predicted from previously encoded macroblocks, either from current image (called *intra-frame prediction*) or from previous frames and future frames (called *inter-frame prediction*). Depending on the intra- and inter-macroblocks, frames are classified as I, P and B frames. The I frame uses only intra-frame prediction i.e., it does not employ any previously coded frames as reference. Whereas P frames are predicted using previously coded frames and B frames are encoded from previous and future frames. The encoder typically combines both intra- and inter-prediction techniques to exploit spatial and temporal redundancy, respectively. Intra-frame prediction involves different prediction modes [68] while finding candidate macroblocks. Inter-frame prediction uses motion estimation by employing different block matching algorithms (such as Hexagon based or full exhaustive search) to identify the candidate macroblock across frames. Finally, a residual is calculated by taking the difference (measured typically using mean absolute difference (MAD) or mean squared error (MSE)) between the predicted and current macroblock. The prediction stage produces 4x4 to 16x16 blocks of absolute-pixel or residual values.

- **Transform coding and quantization:** These absolute values are then transformed and quantized for further compression. Typically, two transform coding techniques (block or wavelet based) are used to convert pixel values into transform coefficients. A subset of these transform coefficients are sufficient to construct actual pixel values, which means reduced data upon quantization. The most popular transform coding is DCT over 8x8 macroblocks.

- **Entropy coding:** Finally, coefficients are converted to binary data which is further compressed using entropy coding techniques, such as CABAC, CAVLC or Huffman. Richardson presents details of video compression [68].

We highlight that inter-frame prediction depends on motion content in the video. The higher the motion in the video is, the lower the compression rate is. Similarly, intra-frame compression is affected by frame contents such as color variation. For instance, the blurrier the video is, the higher the compression rate is. In the next section, we design QoE metrics exploiting these video coding principles.

Application Category	Suitable Metric
Adaptive Streaming	Startup delay, Buffering ratio, Stall ratio
Video Telephony	Bitrate, Fps, Blocking, Blurring
Progressive Downloads	PSNR, SSIM
VR/AR, Game Streaming	Latency, buffering ratio, Stall ratio

Table 2: Metrics based on application category

2.2.2 *QoE Artefacts in Video Telephony*

When a user is in a video telephony call, she can experience different aberrations in video quality, as follows.

Video freeze is a temporal disruption in a video. A user may experience freeze when the incoming video stalls abruptly and the same frame is displayed for a long duration. Additionally, freeze may appear as a fast play of video, where the decoder tries to recover from frame losses by playing contiguous frames in quick succession, creating a perception of `fast` movement. Both these temporal artefacts are grouped into freeze. This happens mainly due to network loss (where some frames or blocks lost) or delay (where the frames are dropped because the frames are decoded too late to display).

Blurriness appears when encoder uses high quantization parameters (QP) during transform coding. Typically, servers use adaptive encoding based on network conditions. In adaptive encoding, server attempts to adjust QP to minimize bitrate in poor network conditions, which degrades the quality of encoded frame. High QP reduces the magnitude of high frequency DCT coefficients almost down to zero, consequently losing information and making it impossible to extract original DCT coefficients at the time of de-quantization. Loss of high-frequency information results in blurriness.

Blocking: Most of the current coding standards (such as H.26x and VPx) are block based, where each image is divided into blocks (from 4x4 to 32x32 and recent 64x64 block in H.265). The blocking metric captures the loss of these blocks due to high network-packet loss. The decoder can introduce visual impairments at the block boundaries or place random blocks in place of original blocks, if the presentation timestamp elapses, which creates bad experience.

Call startup delay is the duration of call setup from the time the caller initiates the call until the callee receives it. We observe an 11 s worst case and 7 s median delay when there is 20% network packet loss, whereas we obtain a median 3 s delay in best network conditions. Although call startup delay does measure user experience, it can only provide information at the beginning of the call, and not during the call.

In this work, we focus on freeze and blurriness artefacts. In our extensive experiments over Skype, Hangouts, FaceTime, we do not observe any blocking artefacts, hence we omit this metric for video telephony. We notice video freezes (temporal artefact) and blurring (spatial artefact). For video freezes, we explore multiple metrics that capture freeze ratio for whole clip, number of freezes per clip and duration of freezes. We do not consider startup delay, as it is an one-time metric.

2.3 Design for Scalable QoE Annotation

In this section, we first describe our framework and measurement methodology. We then explain our QoE metrics and evaluate them across different applications and devices. We validate that our metrics capture video artefacts caused by diverse network conditions.

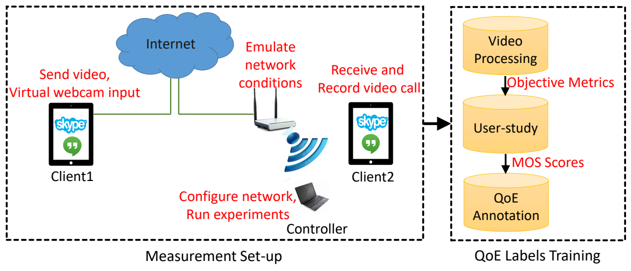


Figure 3: Measurement set-up and system architecture. *Left:* Video telephony measurements and recording video calls. *Right:* Our framework processing recorded videos offline and extracting objective metrics to predict QoE labels.

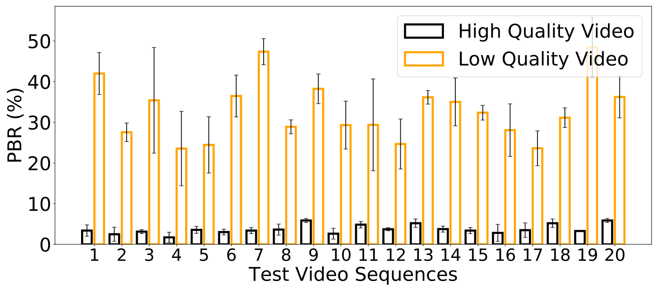


Figure 4: Blur detection using PBR. PBR is larger than 20% for low quality videos, whereas PBR is smaller than 5% for high quality videos. Therefore, the PBR metric is scalable across diverse video contents.

2.3.1 *Measurement Methodology*

Set-up: In Figure 3, we show our measurement set-up and QoE labeling framework. Since video telephony is an interactive application, it requires at least two participating network end-points (clients). In our set-up, we use a mobile device on our local enterprise WiFi network (Client 1) and an Amazon-provided instance as the second end-point (Client 2). Both clients can run Skype or Hangouts[2]. When the video call is placed from Client 1 to Client 2, Client 2 runs a virtual web-cam that inputs a video file in telephony application instead of camera feed; at Client 1, the video is received during the call. At the Amazon Client 2, we use *manycam* [56], a virtual web-cam tool. This tool can be used with multiple video applications in parallel for automation. In our LAN, a controller sits to emulate diverse network conditions and to run experiments on the connected mobile device through Android debug bridge (ADB) interface (via USB or WiFi connection). To automate the video call process, we use AndroidViewClient (AVC) library [12]. Once the received call is accepted, we start the screen recording of the video call session. The videos are recorded using Az screen recorder [3]. The recorded videos are sent to video processing module to calculate the objective QoE metrics. We use Ffmpeg [2] tool to extract our QoE metrics. Ffmpeg is a video framework with large collection of coding libraries. To validate our metrics and translate them into user experience, the videos are then shown to users to rate their experience. Finally, we retrieve MOS from all users and map our QoE metrics to MOS. We delve into the details of our user study and modeling in Section 2.4.

Network conditions: As we experiment real-time with interactive applications (Skype, Hangouts, FaceTime), we need to emulate real user experience under any condition and to obtain samples with MOS values of all levels (1–5). Hence, we conduct tests with ideal network conditions and with throttled network. To create average and bad network conditions, we use Linux utility `tc` and we introduce hundreds of ms of delay, packet loss (10–20% ensuring that the call is setup), or low bandwidth (2–10 Mbps).

Test video sequences: We use the same 20 test video sequences as in Section 2.1. We collect these videos from Xiph media [57] and YouTube. We select 20 representative videos that contain different types of motion and content. All videos are downloaded in Full HD (1920x1280) resolution.

[2]For FaceTime, we use 2 local clients (iPhone/iPad and MacBook Air) on different sub-networks, as creating virtual iOS and web-cam is challenging.

2.3.2 *Our QoE Metrics*

PBR: Video bitrate has been considered a standard metric for perceptual quality of video [39, 16]. We compute the bitrate of recorded videos as a QoE metric. Typically, the bitrate is lower for low resolution video and higher for high resolution video and depends on the level of video compression. Therefore, we capture video blur with `encoded bitrate` by compressing the recorded video. We employ the observation that the blurrier the video is, the higher compression efficiency is. However, we find that the bitrate metric is sensitive to motion in the video, because the block movement for high motion videos is very high and it is low for low motion videos. This results in different encoding bitrates. As described in Section 2.2, low motion videos take advantage of inter-frame prediction, which makes the encoded bitrate motion-sensitive. Therefore, we use intra-coded bitrate by *disabling the inter frame prediction while compressing video.* We experiment with different encoding parameters like quantization parameter (QP) and de-blocking filter techniques and we choose high QP value (30) while coding, to get large bitrate difference when encoding high- and low-quality videos. To achieve robustness to video content, we compute the *relative change between the recorded bitrate and the intra-coded bitrate of the compressed video.* We define this change as the perceptual bitrate (PBR), as it only captures quality of the image. Since we calculate PBR on the recorded video, we do not need a reference video, hence PBR is a no-reference metric.

We validate the PBR metric with respect to the same 20 videos used to compute blur using DCT coefficients in Section 2.1. We create two sets of videos with high and low quality and record them during a Skype call under best network conditions. This experiment aims to validate our blur capturing metric, hence we avoid video freezes by sending low quality videos over best network conditions. Fig. 4 shows PBR for both low and high quality videos. We observe a PBR larger than 20% for low quality videos, whereas PBR is smaller than 5% for high quality videos. Unlike blur detection using previous methods, we see a clean separation

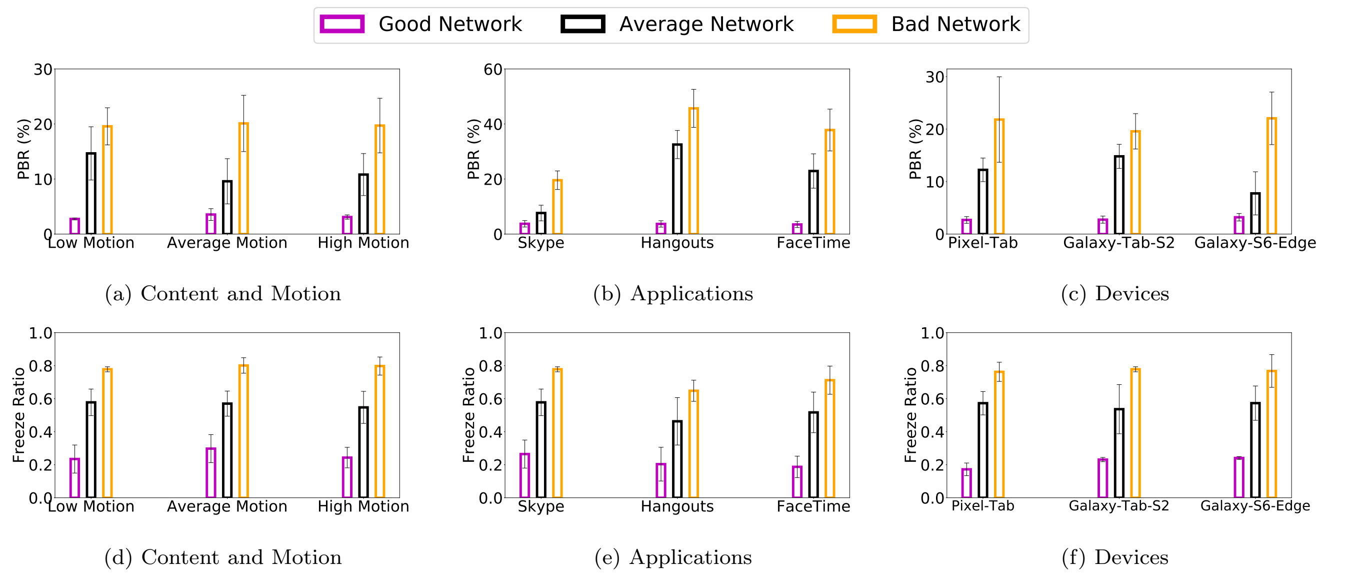


Figure 5: PBR and Freeze Ratio for different videos with respect to content, application and device settings. Results for diverse network conditions are shown (from bad, average, good).

of average PBR between low and high quality videos. Therefore, PBR can be used to detect blurriness in videos without any ambiguity.

Freeze Ratio: As described in Section 2.2, freeze ratio is another important QoE artefact. Freeze ratio is the number of repeated frames over total frames in a given window, i.e., it denotes the amount of time the video is paused because of network disturbance. We use Ffmpeg's *mpdecimate* [2] filter in calculating freeze ratio. The *mpdecimate* algorithm works as follows: The filter divides the current and previous frame into 8x8 pixels blocks and computes sum of absolute differences (SAD) for each block. A set of thresholds (*hi, lo and frac*) are used to determine if the frames are duplicate. The thresholds *hi* and *lo* represent number of 8x8 pixel differences, so a threshold of 64 means 1 unit of difference for every pixel. A frame is considered to be duplicate frame if none of the 8x8 blocks yields SAD greater than a threshold of *hi*, and if no more than *frac* blocks have changed by more than *lo* threshold value. We experiment with several other error methods such as MSE and MAD, but we notice similar results among all three, thus we choose SAD for minimal computation overhead. We use threshold values of 64*12 for *hi*, 64*5 for *lo* and 0.1 for *frac* for all our experiments. The metric does not work if the entire video is having a still image.However, we think that it is a reasonable assumption that most of video telephony applications do not generate such video content.

In addition to freeze ratio, we also compute length and number freezes in video. We define a freeze if the video is stalled for more than one second. We observe that users do not perceive the stall when the length of freeze is very short or if there are very few such short freezes in the video. Therefore, we employ `freeze ratio, length and number of freezes` metrics to capture the temporal artefacts of QoE.

2.3.3 *Micro-Benchmarking of Video Artefacts*

In this section, we show the scalability of our metrics through measurements across different applications and devices. As described above, we use a total of 4 metrics: PBR, freeze ratio, length and number of freezes in the video. For brevity, we show measurements for only PBR and freeze ratio. We find similar trends with other metrics as well. We measure these metrics with 6 different videos from the 20 videos described in Section 2.1, that cover high video-motion and content diversity. These experiments are run on Skype and Samsung Galaxy Tab S2 unless otherwise specified. All videos are recorded under Full HD (1920x1280) resolution with 60 Fps. Each experiment consists of 20 minutes video call with a total of 18 hours of video recordings. The videos are recorded under different network conditions to obtain different video quality levels. We use the following network conditions: good case (0% loss, 0 latency and 100 Mbps bandwidth), average case (5% loss, 100 ms and 1 Mbps bandwidth) and bad case (20% loss, 200 ms, and 512 Kbps bandwidth). Fig. 5 shows PBR and freeze ratio across different applications and devices. Our observations are the following.

Video Motion and Content Diversity: The average PBR for best network conditions is always smaller than 5%. PBR is larger than 20% under bad settings (see In Fig. 5). The average freeze ratio for best network conditions is smaller than 0.3 and for bad conditions, it is larger than 0.8 (Fig. 5). We observe that both PBR and freeze ratio are highly impacted by network conditions and follow same trend with network quality irrespective of content and motion in the video. Therefore, the videos can be labeled accurately because of clear separation between network dynamics.

Application Diversity: In Fig. 5, we present PBR and freeze ratio for three applications: Skype, FaceTime and Hangouts. For Skype, we observe an average PBR of less than 5% with good network and 20% with bad network. Whereas, both FaceTime and Hangouts reach more than 40% PBR under bad network. This discrepancy is due to application logic: Skype does not compromise quality of video, hence low PBR. But, this causes Skype video calls to be stalled more frequently compared to other applications un-

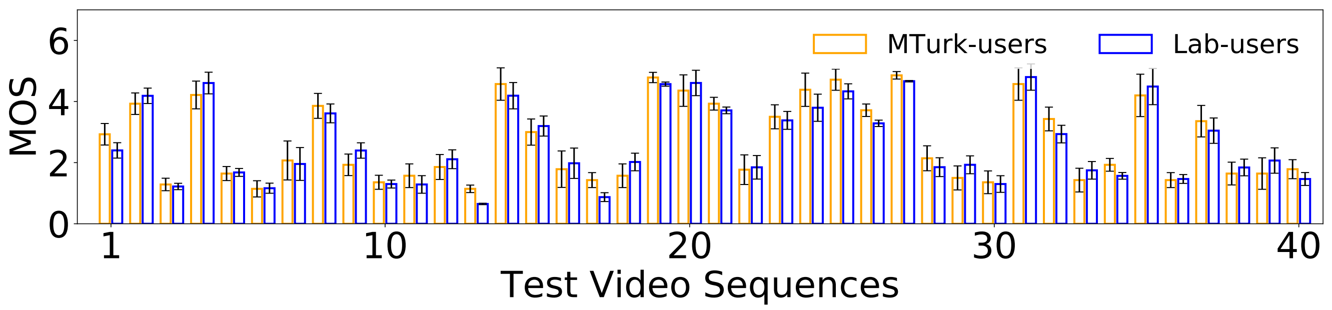


Figure 6: MOS and standard deviation of QoE scores from 30 lab and 30 online users. Although the scores vary across the users both in lab and online, the distribution of MOS is similar in two cases.

der bad network scenarios. Whereas, bitrate adaptation of FaceTime and Hangouts sacrifices quality to provide temporally smooth video. Moreover, quality and bitrate are highly impacted by the underlying video codec the application uses. In our experiments, Skype uses H.264 whereas Hangouts uses VP8. We observe that using different video codecs and adaptive bitrate (ABR) algorithms impacts video quality during call.

The freeze ratio for Skype is similar to FaceTime and Hangouts. However, as network conditions worsen, Skype yields more freezes compared to other applications. Despite the additional freezes, the average freeze-ratio for Skype is only 0.1 larger than other applications. This is due to the longer freezes of FaceTime and Hangouts than of Skype.

Device Diversity: We aim to devise a model that is independent of the device that runs video telephony. Our metrics should capture similar video QoE on different devices given that screen recorder and network conditions are the same. To this end, we evaluate our metrics on three devices: Samsung Galaxy Tab S2, Samsung Galaxy Edge S6 and Google Pixel Tab. In Fig. 5, we observe that both PBR and freeze ratio show same distribution under different network settings on all devices. Hence, our metrics are device independent.

2.4 From Video Artefacts to QoE

We validate our QoE metrics presented in Section 2.3 with subjective measurements. We seek to ensure that our metrics accurately capture video telephony QoE artefacts. Video artefacts vary across applications and network conditions. Additionally, they can appear together (e.g., when we have both blurriness and stutter) or independently. Intermingled video artefacts result in diverse QoE levels (from bad to excellent), standardized as MOS by ITU-T [72][3]. We carry out a user study and obtain a MOS for each video.

2.4.1 Data Preparation

As described in Section 2.3, the dataset is prepared from recorded video calls. We use 20 videos described in Section 2.1, for video calls on Skype, FaceTime and Hangouts. For each video and application, we repeat experiments on multiple devices. The video calls are recorded under different network conditions to produce good, average and bad quality videos. We record video calls for Skype, Hangouts on Samsung Galaxy (SG) S6 edge, Google Pixel Tab, SG S2 Tab and for FaceTime on iPad Pro (model A1674) and iPhone 8 Plus. We post-process these videos into 30 seconds video clips and evaluate QoE for each clip. We host a web server with these videos and ask users to rate the experience.

2.4.2 User-Study Setup

We conduct the user-study in two phases: 1) in two labs, 2) online using a crowd sourced platform. The lab user study is conducted at a university campus and an enterprise lab. The online user study is conducted on Amazon Mechanical Turk platform [1]. Online users are from the United States and are in age range 20–40. We collect results from 60 lab users and more than 150 online users[4]. We faced several challenges while deploying a user-study in the wild, such as ensuring that 1) users watch the whole video clip without fast forwarding and then rate their experience, 2) users can rate the video if and only if they watch the video, 3) users watch on screens of similar size to avoid huge QoE variance i.e., small screen users do not perceive poor quality compared to large screen users. We address these challenges by carefully designing the user-study website and avoiding noisy samples. First, we disable the video controls in the web page so that users cannot advance the video. Second, we introduce random attention pop-up buttons while watching the video and ask users to click on the attention button to confirm that they are watching the video. A user cannot move to next video until the current video is rated. Finally, we restrict users from taking the study on mobile devices, to avoid variance due to small screens. The user-study web page can be run on Chrome, Firefox and Safari. Once the user completes rating all the videos, results are pushed to our server. Then, users are granted a server-generated code for processing payments in MTurk. Amazon MTurk restricts users from taking again the same study, thus our users are unique.

2.4.3 Lab vs. MTurk Users

We compare the MOS across lab and online users for the same set of 40 video clips and we find similar distributions across 80 clips. Fig. 6 shows the MOS bar plots of user ratings. The lower the rating is, the lower the QoE is. For each video sequence, we plot the average MOS across users with error bars for standard deviation. We observe that lab and online users exhibit similar distributions of MOS. The standard deviation of MOS between lab and online users is smaller than 1 MOS for 96% of videos. We find that 4% of

[3]MOS takes values from 1 to 5, with 5 being excellent.

[4]Our conducted user-studies are approved by the Institutional Review Board (IRB) of our institution.

these video clips have more than 1 MOS standard deviation and we remove them from our analysis as outliers. Such videos have a large portion of uniform background, hence their content challenges classification between average and good MOS.

2.5 Modeling Our Metrics to MOS

We now present our MOS prediction model that corroborates that our metrics accurately capture QoE artefacts and users' MOS. We build a model to map our objective metrics to subjective evaluations (MOS from user-study). Typically network administrators need to estimate a subjective evaluation such as MOS. However, QoE assignment in 5 classes can be cumbersome and may give little information on network quality. Hence, we map MOS scores to 3 classes (i.e., bad, average, good) that can be used by LTE or WiFi deployments to enhance resource allocation. We first explain our modeling methodology from freeze and blur metrics to MOS, and then describe how to translate MOS into 3-class QoE.

Typically, objective QoE metrics are mapped to MOS scores using non-linear regression [21]. Our regression model employs the average MOS from 15 users as ground-truth per video clip and it is based on ensemble methods [58]. Taking the ensemble of models, we combine predictions of base estimators to improve generalizability and robustness over a single model. Moreover, a single model is always vulnerable to over-fitting and is complicated, which can be avoided in the form of ensemble of weak models. The ensemble is usually produced by two methods: *averaging* or *boosting*. We employ boosting method, in particular `AdaBoost`[41], where the base estimators are built sequentially and one tries to reduce the bias of the combined estimator, thereby combining several weaker models and producing an accurate model.

`AdaBoost` algorithm fits a sequence of weak models (for example, small decision trees in our framework nearly equivalent to random guessing) on different versions of data. The predictions from each weak model are combined to produce a strong prediction with a weighted sum scheme. At each iteration, called boosting iteration, a set of weights ($w_1, w_2, , w_N$, where N is number of samples) are applied to training data. The weights are initialized with $w_i = 1/N$, in order to train a weak model on the original data. In the following iterations, the weights of training samples are individually modified and model is applied again on the re-weighted data. At any given boosting stage, the weights are changed depending on prediction at the previous step. The weights are proportionately increased for incorrectly predicted training samples, while the weights are decreased for those samples that were correctly predicted. The samples which are difficult to predict become more important as the number of iterations increases. The next-iteration weak model then focuses on the incorrectly predicted samples in the previous step. Details of the Adaboost algorithm are given in [41][22].

Moving from 5 MOS scores to 3 classes, one has to estimate two thresholds m_1, m_2 in MOS (bad label: MOS $< m_1$, average: $m_1 \leq$ MOS $< m_2$, good: MOS $\geq m_2$). To this end, we first train the regression model and then search the MOS scores space with a sliding window of 0.05 for the two thresholds. We iterate over all such possible thresholds to maximize the accuracy of the trained model. We compute the true labels and prediction labels from the test MOS scores and predicted scores respectively using the corresponding thresholds in each iteration. We then select the thresholds which give highest accuracy from prediction labels. Therefore, our framework is divided into two phases: predicting MOS scores and labeling the scores with optimal thresholds.

Table 3: Models performance on Skype data

Model	**Precision (%)**	**Accuracy (%)**	**Recall (%)**	**MSE**
SVR	89.33	89.33	89.33	0.36
MLP	90.77	90.77	89.62	0.36
KNN	82.91	81.67	81.67	0.63
RF	89.72	89.34	89.34	0.41
ADT	92.21	92.00	92.00	0.29

Table 4: Models performance on FaceTime data

Model	**Precision (%)**	**Accuracy (%)**	**Recall (%)**	**MSE**
SVR	88.13	86.00	86.00	0.32
MLP	90.00	89.77	89.30	0.28
KNN	74.98	72.00	72.30	0.48
RF	90.37	90.00	90.00	0.28
ADT	90.28	90.00	90.00	0.26

We first evaluate our metrics by fitting five most common regressors: Support Vector Regressor (SVR), Random Forests (RF), Multi-Layer Perceptron (MLP), K-Nearest Neighbor (KNN) and Adaboosted Decision Tree Regressors (ADT). Each model is evaluated under 10 fold cross-validation. We perform a fine grid search to tune the hyper-parameters for all the models. We select the best parameters from the grid search and use the best estimator for the rest of the evaluations. This is repeated for Skype, FaceTime and Hangouts applications separately. The performance of each model is presented in terms of precision, accuracy and recall for three applications after labeling stage. We present micro-average metrics of accuracy, precision and recall, as the macro-average does not yield class importance [4]. Micro-average aggregates the contributions from all the classes to compute the average metric. We also report the mean squared error (MSE) for all models. We observe at least 88% accuracy for all the models in all applications, except KNN regressor. This discrepancy is due to the weakness of KNN with multiple features. We observe that KNN does not generalize our dataset well and predicts most of the samples incorrectly. The mis-prediction is due the fact that each sample in higher dimension is an outlier as the distance metric (euclidean distance in our model) becomes weak with more features [65], unless the data is well separated in all dimensions. Among all models, ADT has consistent and better accuracy in all applications with a maximum accuracy of 92% in Skype, 90% in FaceTime and 93.33% in Hangouts. We also use boosting with other models, but boosting did not improve the accuracy. Therefore, as ADT performs better than other models, we use this model for all the other evaluations unless otherwise specified. The best hyper-parameters for ADT from grid search are: `n_estimators = 10, learning_rate = 0.1` and `linear` loss [58].

Fig. 7 shows a scatter plot of user-study MOS vs. predicted MOS for the three applications. The MSE in MOS for the three applications is smaller than 0.4 with ADT for all the applications. We also observe three clear clusters approximately divided by the MOS scores $m_1 = 2$ and $m_2 = 4$, that coincide with our thresholds for labeling the scores into 3 classes. This also justifies our design choice to finally em-

Table 5: Models performance on Hangouts data

Model	**Precision (%)**	**Accuracy (%)**	**Recall (%)**	**MSE**
SVR	91.36	90.67	90.67	0.44
MLP	89.34	88.48	88.48	0.45
KNN	87.02	86.67	86.67	0.49
RF	91.22	90.67	90.67	0.43
ADT	93.75	93.33	93.33	0.38

Table 6: Model performance across devices for Skype

Devices		Model Performance		
Training	Testing	Precision (%)	Accuracy (%)	Recall (%)
SG-S6 Phone	SG-S6 Phone	89.90	88.57	88.57
	SG-S2 Tab	88.41	88.00	88.00
	Pixel Tab	88.52	87.62	87.62
SG-S2 Tab	SG-S6 Phone	83.73	84.43	83.00
	SG-S2 Tab	93.04	91.43	91.43
	Pixel Tab	82.69	82.86	82.86
Pixel Tab	SG-S6 Phone	84.43	84.00	84.00
	SG-S2 Tab	84.40	86.49	86.49
	Pixel Tab	88.20	94.40	94.00

Table 7: Model performance across devices for FaceTime

Devices		Model Performance		
Training	Testing	Precision (%)	Accuracy (%)	Recall (%)
iPad Pro	iPad Pro	93.60	92.00	92.00
	iPhone 8 Plus	90.18	89.93	89.93
iPhone 8 Plus	iPad Pro	88.34	88.34	88.34
	iPhone 8 Plus	92.50	92.00	92.00

Table 8: Model performance across devices for Hangouts

Devices		Model Performance		
Training	Testing	Precision (%)	Accuracy (%)	Recall (%)
SG-S6 Phone	SG-S6 Phone	90.03	90.00	90.00
	SG-S2 Tab	84.26	84.77	84.77
	Pixel Tab	82.86	82.00	82.00
SG-S2 Tab	SG-S6 Phone	89.61	89.21	89.21
	SG-S2 Tab	91.76	90.00	90.00
	Pixel Tab	85.41	84.77	84.77
Pixel Tab	SG-S6 Phone	86.79	86.00	86.00
	SG-S2 Tab	85.05	85.00	85.00
	Pixel Tab	86.17	86.67	86.67

ploy three labels (bad, average, good) out of 5 MOS scores.
Model performance for Skype: Table 3 shows performance of Skype model across different regressors. The MSE in MOS is 0.29 in case of ADT and it is 0.63 with KNN regressor. Similarly, precision, accuracy and recall for ADT is the highest, while KNN being lowest. ADT model gives a best threshold of $m_1 = 2$ and $m_2 = 3.8$ in separating the MOS scores into labels. While all the other models produce a threshold of $m_1 = 2$ and $m_2 = 3.4$. Here, the best performance in ADT results from (*i*) its low MSE and (*ii*) the wide range for average class separation, i.e., it labels all the samples from MOS 2 to 3.8 as average class, while other models yield average class from MOS 2 to 3.4. The performance gap is due to the distribution of our average class samples spread over wider range of bad or good labels. Using $m_1 = 2$ and $m_2 = 3.8$ thresholds, we get 30%, 40% and 30% of our 300 samples in bad, average and good labels.

To show that our Skype model is device independent, we further evaluate our model across three devices: SG-S6 phone, SG-S2 Tab and Pixel Tab. Table 6 shows precision, accuracy and recall for all three devices. We measure performance by training on one device, and testing on other devices. We observe that the performance is always better when trained and tested on the same device compared to training on one device and testing on other device. However, we find a difference of less than 7% in accuracy when trained and tested across different devices, with an accuracy of at least 83%. This corroborates that our model is robust across devices and it can be trained and tested without device constraints i.e., our metrics can collected on a certain device and can be applied to any other devices for Skype.
Model performance for FaceTime: Table 4 shows performance of FaceTime model across different regressors. Similar to Skype, we observe similar performance ($>$ 89% accuracy) for all models except the KNN regressor. Here, although the RF regressor is performing better (90.37% precision) than ADT, the MSE in MOS is larger than ADT. Interestingly, all models produce same thresholds of $m_1 = 2$ and $m_2 = 3.4$ in labeling the scores. Here, the samples are distributed uniformly across three classes unlike Skype, hence all regressors are performing almost equally. However, KNN regressor still suffers in FaceTime model due to weakness with many features as explained above. Using these thresholds, we get 30%, 36% and 34% of the 200 samples in bad, average and good labels.

To validate that our FaceTime model is device independent, we train and test across iPad and iPhone devices. Table 7 shows that when training and testing on same device, we observe 92% accuracy. Whereas, training and testing across devices yields at least 88% accuracy. Hence, we observe a difference of 4% accuracy across device training and testing. Our FaceTime model is also device-independent. Note that, we are not comparing the performance of our model training on Android devices and testing on iOS devices and vice-versa, because the recording set-up is different these environments.
Model performance for Hangouts: Table 5 shows performance of Hangouts model across different regressors. Similar to other applications, ADT outperforms other models with 93.33% accuracy with an average MSE of 0.38. Similar to FaceTime, we observe that all models produce same thresholds of $m_1 = 2$ and $m_2 = 3.5$ in labeling the scores. Using the above thresholds, we get 32%, 34% and 34% of the 300 samples in bad, average and good labels respectively.

We further evaluate the Hangouts model across three devices: SG-S6 phone, SG-S2 Tab and Pixel Tab. Table 8 shows that an accuracy of at least 86.67% when training and testing on same device, while inter-device train and test gives an accuracy of at least 82%. Overall, we observe less than 8% accuracy difference when trained and tested across different devices. This shows that the model is independent of where training and tested are conducted.

2.6 Comparison with Baseline and Feature Importance

We compare our model prediction error with previous work for Skype application. As we need no-reference metrics for the baseline comparison, we use the DCT blur metric used by Jana *et al* as spatial metric and frame-drop metric in [37] as temporal metric. We fit the ADT model with these two metrics as well as with our metrics using the MOS scores from our user-study. Fig. 8 shows the performance of Skype application across the 20 videos described in Section 2.1. Clearly, for a single video, both baseline and our model yield less than 0.2 MSE in MOS whereas as the number of videos increases, the MSE grows larger than 1.3 MOS for

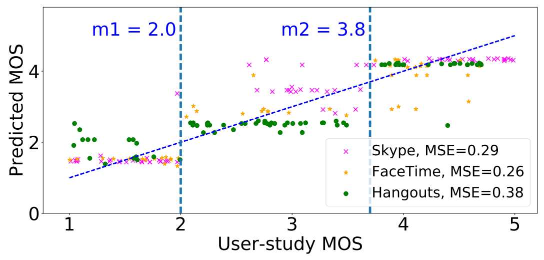


Figure 7: User-study vs. predicted MOS for the three applications. We also show 3 clear QoE clusters divided by our thresholds $m_1 = 2$ and $m_2 \approx 3.8$.

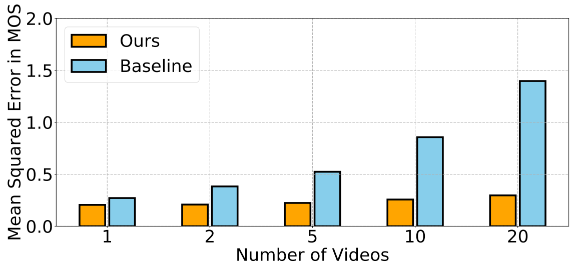


Figure 8: Comparison of our metrics vs. baseline. Baseline has an up to 1.4 MOS estimation error across 20 video contents, hence failing to distinguish good/bad from average QoE.

baseline metrics. Whereas, our model has a maximum of 0.4 MSE. The baseline metric's high MOS error with large number of videos is due to DCT metric's inability to scale across diverse video content. In fact, in our experiments we measure feature importance, which is defined as the amount each feature improves performance weighted by the number of samples this feature is responsible for. We observe that feature importance for DCT is as low as 0.1, and it is 0.9 for frame-drop metric. Therefore, previous metrics fail to model video telephony QoE across diverse videos. We observe similar results for FaceTime and Hangouts.

We also evaluate the importance of each proposed metric. Fig. 9 shows the importance of features for ADT over all three applications. Out of all features, freeze ratio is dominating with at least 0.5 feature importance on all applications. For Skype, we observe highest ($>$ 0.7) importance to freeze ratio. The rest of the metrics are almost equally important, and removal of any of these features causes an up to 0.2 accuracy degradation. Whereas, for FaceTime and Hangouts, we notice high importance for PBR (0.28 for FaceTime and 0.41 for Hangouts) due to their compromise in quality over freezes.

2.7 Summary

Enterprise network administrators need QoE information to model QoS to QoE relationship and to efficiently provision their resources. Currently, applications do not provide QoE ground-truth. In this work, we address this problem for video telephony by introducing four scalable, no-reference QoE metrics that capture spatial and temporal artefacts of video quality. We investigate the performance of our metrics over three popular applications – Skype, FaceTime and Hangouts – across diverse video content and five mobile devices. Finally, we map our metrics with a large-scale MOS user-study and show a median accuracy of 90% in annotating the QoE labels according to MOS. Our metrics outperform state-of-the-art work while capturing exact user rating. We plan to extend this study to video streaming, such as YouTube and Netflix, and VR/AR applications.

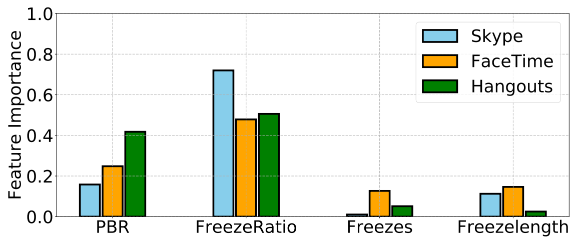


Figure 9: Feature Importance for the three applications.

In conclusion, our contributions are the following:

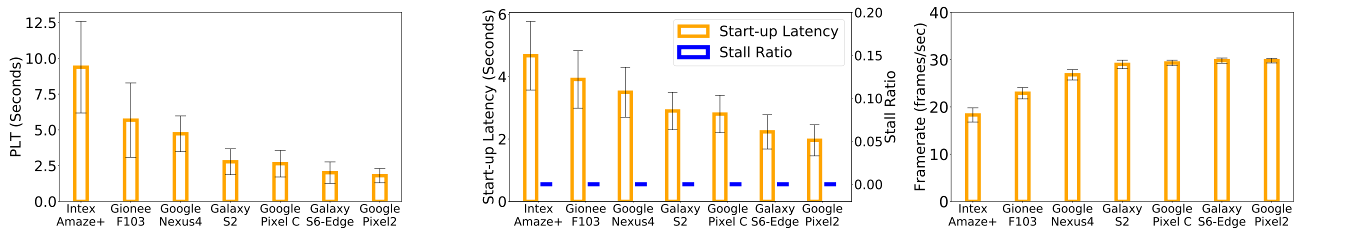
- We uncover limitations of existing work for QoE annotation of video telephony in enterprise networks (Section 2.1).
- We introduce new QoE metrics for video telephony that are content, application and device independent (Section 2.3).
- We micro-benchmark our metrics with the three most popular applications, i.e., Skype, FaceTime and Hangouts, and five different mobile devices (Section 2.3).
- We develop a model to map our QoE metrics to MOS and demonstrate a median 90% accuracy across applications and devices (Section 2.4).

3. IMPACT OF DEVICE PERFORMANCE

While mobile smartphones have now penetrated a significant fraction of world population, the devices vary widely in terms of cost and performance. For example, the smartphones in the market range from ≈\$50 to ≈\$1000 [7, 19], and the cost largely depends on the hardware specifications. For example, a \$600 phone such as OnePlus5 has 8 cores, running upto 2.4 Ghz clock and 6 GB RAM, while a cheaper \$60 phone (e.g., Dell Venue Pro) only has 2 cores with up to 1 GHz clock and 512 MB RAM.

However, the impact of the hardware specs on the performance of mobile Internet applications is not well-studied. A large fraction of past research on mobile Internet performance has focused on the effect of the *network* on the applications [15, 53, 74, 69, 73, 88, 18] but not the device. Some studies do address the compute performance of mobile Internet applications [89, 52, 55], but the underlying technologies developed are more suited for higher end devices.

Given that a large fraction of mobile users in developing regions (62%-68% by some account [81]) uses smartphones with relatively less expensive lower end hardware, it is important to understand the impact of the device hardware on the performance of the mobile apps. In fact, with the increased 4G penetration, network speed is no longer the bottleneck in many developing countries [7, 19, 47, 48]. Our measurements 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 (§3.1). This underscores the importance of



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

Figure 10: 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 9.

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
SG S6-edge	Exynos 7420	8	6.0.1	400-2100	Mali-T760	3	$880
Pixel2	Snapdragon 835	8	8.0.0	300-2457	Adreno 540	4	$700

Table 9: The table shows the set of diverse mobile devices used in our motivation experiment and their corresponding specifications including cost, CPU capabilities, and memory capacity.

studying the influence of the device hardware on the performance and explore ways to improve the mobile web experience on lower end hardware.

In this paper, we focus on three types of mobile Internet applications: *(i)* Web browsing, *(ii)* video streaming, and *(iii)* video telephony. We study performance in terms of Quality of Experience (QoE) and energy. We especially focus on how the *CPU clock frequency* impacts performance given that the clock frequency has the most impact on the performance among other aspects of the specification (§3.1.3).

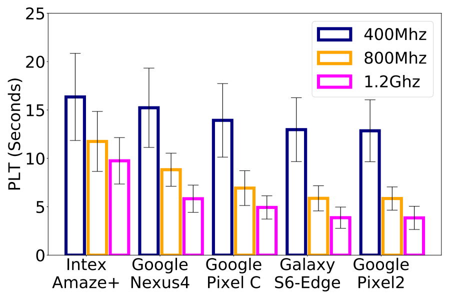
One of our key findings is that a slow CPU speeds not only affects compute performance, but has a second-order effect on the network performance. In our measurements, for example, the TCP performance drops more than 20 Mbps, when the CPU clock of the phone drops from 1512 Mhz to 384 Mhz. This is owing to the fact that a slower clock increases TCP processing delays by as much as ≈12 ms. The implication is that the application is doubly impacted by the slower clock that slows down both the network latency/throughput as well as the compute performance inside the application.

Our goal now is in *isolating* the effect of CPU clock on the network and the compute activities of the application. This isolation is critical to shed light on how the application performance can be improved for low-end devices: should the optimization focus on improving the TCP packet processing or the application compute? While straightforward for video applications, for Web browsing, this isolation is challenging. Loading of Web objects, a network process, and processing of these objects, a compute process, are intertwined during the page load process and cannot be cleanly separated. To this end, we leverage the WProf tool [83, 60] that breaks down the Web page load process into network and compute processes on the critical path. The key findings of our study are:

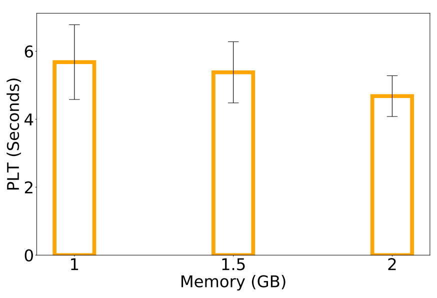
- **Mobile Web performance is significantly affected by slow clocks:** Web page loads slow down by a factor of 5 on average when CPU clock speed reduces from 1512 Mhz to 384 Mhz. This slow down is both because of compute processing delays and the second-order effect in network processing. Our isolation experiments show that compute processing is more of a bottleneck under slow CPU speeds, accounting for over 60% of the page load time. Within the compute activities, Javascript evaluation is affected most by the CPU slow down, and rendering and painting are affected the least.

- **A slow clock does not significantly affect video streaming because of offloading and pre-fetching:** The QoE of video streaming is largely unaffected by slow CPU speeds, despite video streaming being a compute-intensive application. By isolating the impact of compute and network, we find that this paradox is because of two reasons. From the compute standpoint, streaming applications use special hardware decoders, even available on low-end phones, to process video data rather than decode on the CPU. From the networking standpoint, streaming applications pre-fetch enough data and are unaffected by the second-order network slow-downs caused by slow CPU.

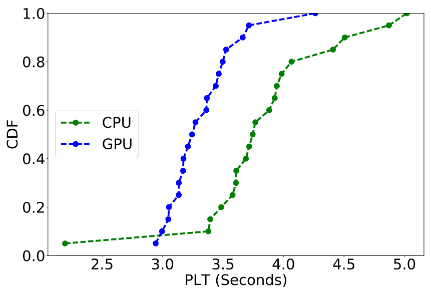
- **Leveraging hardware offloading is a promising approach to improve Web performance under slow clock:** Similar to the hardware offloading techniques used in video applications, offloading Javascript to a low-power DSP can potentially improve Web performance. Our preliminary analysis with offloading only regular expression evaluations shows that the improvement can be as much as 18% in page load time along with a 4× improvement in power consumption.



(a) Clock



(b) Memory



(c) GPU

Figure 11: Impact of CPU, memory, and GPU resources on Page Load Time (PLT). Varying the CPU speeds has the most impact on PLT compared to the impact of varying memory availability or leveraging the GPU for rendering.

3.1 Measurement Methodology

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 9 shows the different devices. The cost ranges from about \$60 to \$880, and the maximum CPU clock frequencies range from 1300 MHz to 2457 MHz. The first two phones in the table, Intex Amaze+ and Gionee F103 are popular phones in India.

We first describe default experimental setup before going over to the results.

3.1.1 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 [83]. We load the top 50 Web pages from the Alexa [86] and estimate the average PLT. We automate the page loads for repeatability using Chrome remote debugging protocol [46] over Android Debug Bridge (ADB) [8].

Video Streaming: We use **YouTube** to measure video streaming performance using two QoE metrics: *start-up latency* and *stall ratio*. 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 [10]. The performance is measured over a 5 minute FullHD format (1080p) video clip. We use ADB [8] 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 *frame rate*, a commonly used metric [38]. Frame rate is measured as the number of frames shown per second and is available from the Skype technical information displayed during the call. Unfortunately, Skype does not provide APIs to extract the frame rate information from this technical information display. Instead, we screen record [3] the Skype technical information and extract the frame rate using an optical character recognition tool [80].

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 [56] that plays a video file in Skype instead of camera feed; at the mobile end, the video can be seen during the Skype call. We use the same 5 minute video used in YouTube experiments. To automate the Skype call such as starting and ending calls, we use AndroidViewClient (AVC) library [12].

Network Set Up: 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. For video streaming and video telephony, we host the videos on a windows laptop on our LAN. In case of Web, we host the pages on a desktop on our LAN. The LAN is 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. We verified that the network performance of the LAN is consistent.

For each workload, we repeat the experiment 20 times and present the average and standard deviation. During the experiments, the average CPU load is less than 15%.

3.1.2 Application QoE Across Low- and High-end Devices

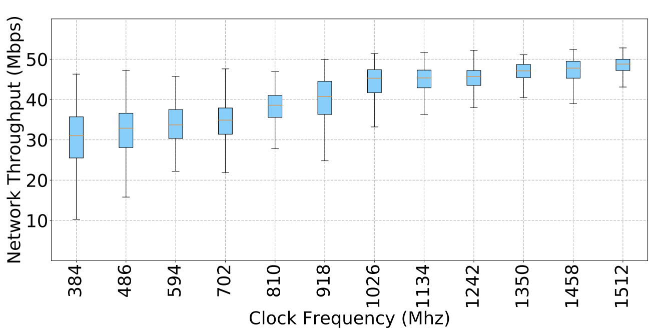
Figure 10 shows the performance of the three applications across the devices. Based on the device model, there is significant different in performance even though all experiments are done with the same network conditions.

For Web loads (Figure 10a), there is a 7 seconds difference in average PLT between the low-end Intex Amaze+ phone and the high-end Google Pixel2. The variance in PLT is also higher (>3 seconds) in the Intex Amaze+ compared to Google Pixel 2. This variance must stem from the device itself, since the network conditions are the same.

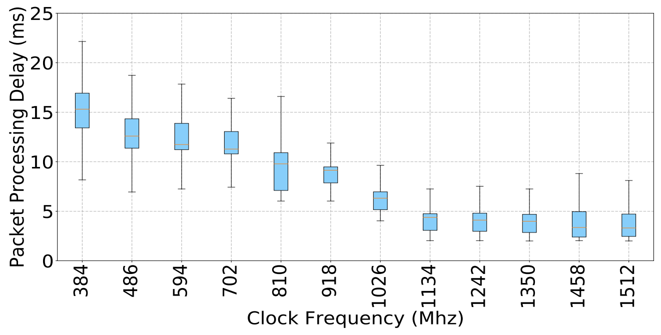
In the case of YouTube (Figure 10b), there is a linear increase in start-up latency from 2 to 5 seconds from the high-end to low-end devices. However, after the start-up latency, there is zero impact on the stall ratio. In effect, once the user waits for the video to start, there is practically no difference in QoE between the low-end and the high-end device. For Skype (Figure 10c), 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. It, however, has a better specification.

The experiments were conducted under the default *fre-*



(a) Throughput vs. Clock



(b) Packet processing delay vs. Clock

Figure 12: Second-order effect of CPU speed on network. (a) TCP throughput reduces from an average of 48 Mbps to 32 Mbps under slow clock under the same network conditions, b) The cause for this reduction in throughput is internal to the device; the average TCP packet processing delay increases by 12 ms at slow CPU speeds. Experiments conducted on Nexus 4.

quency governor setting. Frequency governors [44] on Android scales CPU speeds according to power and performance considerations. In the default case, Governors set the best CPU speeds according to the load on the CPU and available power. The experiments were done with the phones connected to external power so that power does not impact performance.

3.1.3 *Individual Impacts of CPU, Memory and GPU*

In this section, we drill down on the device specification that impacts the performance most. Five device specifications in Table 9 can potentially impact application performance—CPU speeds, GPU configuration, memory capacity, OS, and number of cores. Prior work [42, 14] has shown that the Internet applications do not typically use more than two cores and thus the performance of applications does not change with increase in the number of cores. Also, the OS does not significantly impact the application performance [20].

Figure 11 shows the impact of the remaining three resources—CPU, memory, and GPU—on Web loads. The experiments are the same as described in §3.1.1. The effect of a given resource is isolated by changing its value while keeping the remaining set up constant.

The CPU clock frequency is changed using governors [44] mentioned before. Figure 11a shows the effect of CPU clock on 5 different devices.[5] The CPU clock has significant impact on application performance, improving PLT by an average of 43% and 71% when going from a slow CPU (400 MHz) to fast CPU (1.2 GHz) on the Intex Amaze+ phone (a low-end phone) and the Google Pixel 2 (a high-end phone), respectively. In terms of time this is about 6 seconds and 9 seconds respectively.

Figures 11b and 11c show that the effect of memory and GPU is much less drastic compared to CPU. These experiments were conducted on the Nexus 4 phone. We change memory capacity by creating RAM disks [54] (in steps of 512MB) from available memory and assigning these RAM disks to the application. We use Chrome browser settings to enable or disable GPU accelerated rendering of a web page. Improving memory availability from 1 GB to 2 GB improves PLT modestly by 17.6% or about 1 second. The PLT with GPU accelerated page rendering has just 0.5 seconds less than CPU accelerated rendering.

The key takeaway is that CPU speed has the most impact on Web QoE. We verified this trend in Video streaming and telephony as well; we omit the graphs for brevity. Accordingly, in this paper, we focus on the effect of CPU clock on application performance.

3.2 Second-Order Effect of CPU Clock on the Network

One of our findings when studying the impact of CPU speed on application performance is that the CPU clock not only affects application processing, *but has a second-order effect on network latency/throughput because of the packet processing delays.* This in turn impacts application performance as well. In our experiment, we generate traffic using the IPerf tool [50] from a server to the Nexus 4 smartphone. IPerf generates continuous traffic, and we measure the average throughput over 5 minutes duration. We repeat the experiment 20 times for 12 different CPU clock frequencies.

Fig. 12a 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 (§3.1.1) that in our set up, the network condition is held near-constant since we use an overprovisioned LAN and the video and Web content are served from a server located in the LAN.

The reason for the decreased TCP throughput is that packet processing is compute intensive, and a slow CPU increases the packet processing delays. Figure 12b shows the average delay to process a packet under different clock frequencies for the same IPerf experiment. Android timestamps packets when it reaches the kernel which can be obtained using the IOCTL system call. We measure the packet processing delay as the time between when the packet reaches the application and when it is received at the kernel.

Figure 12b shows that the packet processing delay when the CPU frequency is 384 MHz is 12 ms more compared to when the CPU frequency is 1512 MHz, per packet. This increase in packet processing significantly affects TCP throughput, even when the network conditions remain the same. We repeated these experiments on the higher end Google Pixel 2 phone and found similar trends; even though the packet

[5]The two remaining devices were not tested because changing the CPU clocks required rooting the phone.

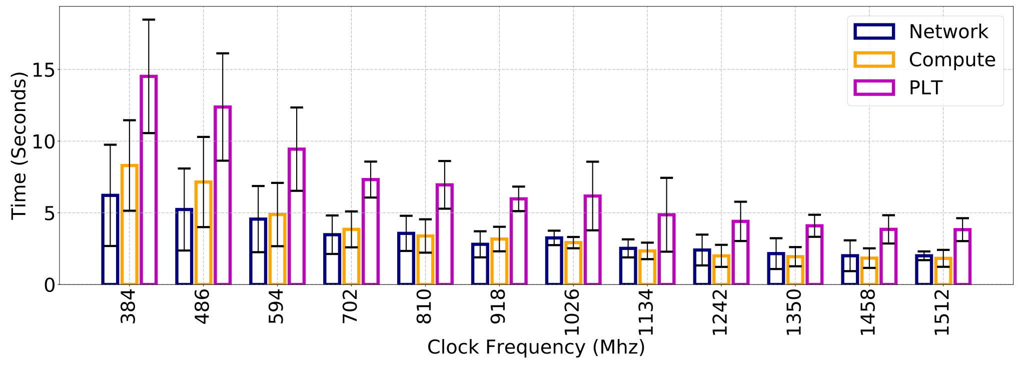


Figure 13: Isolating the effect of network activities and compute activities as the CPU speed changes. The PLT of the Web page is the sum of the time taken for compute activities and the time taken for the network activities on the critical path [83, 60]. The PLT is the same as that shown in Figure 13. As the CPU slows down, the compute activities are affected more than the network activities.

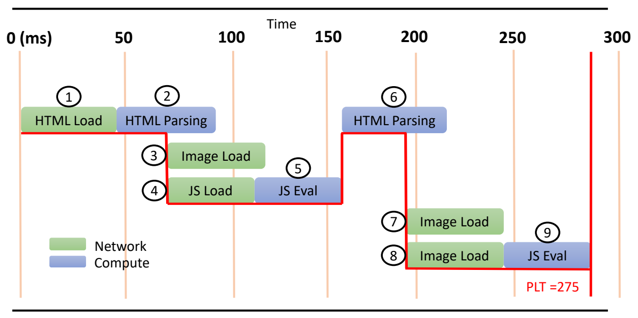


Figure 14: An example page load process divided into its network and compute activities, obtained from the WProf tool [83, 60]. The red line shows the critical path during the page load process, and represents the page load time.

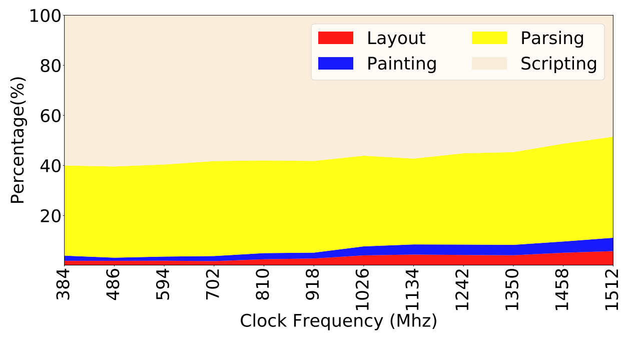


Figure 15: The time spent on compute activities during page load is further divided into individual activities—Scripting (or Javascript evaluation), HTML Parsing, Layout, and Painting. The scripting time increases significantly when the CPU slows down.

processing was considerably faster ($<$ 2ms under high clock) compared to Nexus 4, the difference in processing between the slow and the fast CPU frequency settings was 10ms.

This second-order effect has significant implications. The CPU capacity can affect application QoE both by slowing down the compute and the network components of the application. Isolating these effects is critical for designing new optimizations.

3.3 Web Browsing

Different from our earlier experiments (§3.1) which were conducted across devices, in this section we study Web QoE across 12 different CPU speeds on the same device. The experiments are repeated over three smartphones—Nexus 4, Intex Amaze+, and Google Pixel 2 (see Table 9 for specs). The experiments involve loading the top 50 Websites from the Alexa suite [86]. We use the same experimental set up as described in §3.1.1. The CPU speeds are changed using the governor [44]. We present the results from Nexus 4 in detail and summarize the results from the other two phones for brevity. Recall that the Web pages are hosted in a local server and accessed via a well-provisioned LAN so that the performance is not affected by the external network conditions.

Figure 13 shows the average PLT across different CPU clock frequencies. The PLT increases 77% when the CPU clock frequency drops from 1512 MHz to 384 MHz. The trends are similar to Figure 10a where the page loads much slower on low-end devices compared to higher-end devices. In other words, the performance of Web page loads on low-end devices is similar to that of a slower clock on a high-end device.

Impact of Network Vs. Compute: Our goal now is to determine if the poor Web QoE is because the network throughput drops under slow CPU speeds or because of slower processing. Recall that the slow CPU speeds not only affects compute but also the network throughput due to the packet processing delays (§3.2). The problem is that during Web page loads, network and compute activities are interspersed, making it hard to cleanly separate the effect of network and compute.

Instead, we leverage the WProf tool that extracts the timings of the network and compute activities of the page load process. Figure 15 shows the output provided by the WProf tool [83, 60] when loading an example page. In this example, the network components are shown in *green* and the compute components are shown in *blue*. WProf [83] (and the version for mobile browsers, WProf-M [60]) first identifies the timing of each activity. WProf then extracts the dependencies

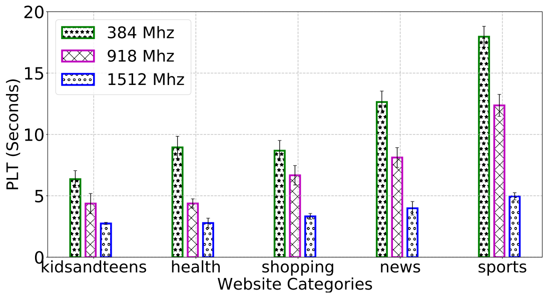


Figure 16: PLT for three different CPU frequencies for different Webpage categories. The sports Webpages are Javascript-heavy and are affected the most by the CPU slow down.

between the activities and draws the dependency graph. For example, in Figure 15, HTML parsing (6) cannot start until the Javascript (JS) is evaluated (5) because of dependencies between parsing and Javascript evaluation. The critical path is the bottleneck path in the dependency graph (shown using the red line); the length of the critical path provides the PLT.

Using WProf, we estimate the time on the critical path involving compute activities (HTML parsing, scripting, rendering, and CSS evaluation) and network activities (object loading) as shown in Figure 14. The sum of the network and compute activities gives the PLT (the PLT numbers are from Figure 13). 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 (not shown here).

Figure 16 further dissects the compute activities into HTML parsing, JS scripting, CSS evaluation, layout and painting. 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 low CPU frequencies. The layout and painting only account for 4% of the compute time on the critical path.

The takeaway from Figure 16 is that, a key component for improving Web page loads, especially at slow CPU speeds, is to improve the efficiency of scripting. Figure 17 demonstrates this in another way – Websites with a large number of Javascript are affected the most when the CPU slows down. In this experiment, we choose the top 50 pages from the Alexa suite [86] under different categories – kids and teen, health, shopping, news, and sports. The sports Webpages spend the most time on scripting and clearly suffers the maximum slow down (up to 13 sec) when CPU frequency drops from the fastest to the slowest. On the other hand, kids and teen Webpages increases by < 3.5 seconds. We leverage this observation to improve Web page load performance (§3.5.2).

3.4 Video

We experiment with YouTube and Skype applications to study the performance of video streaming and telephony, respectively. The set up and the QoE metrics used are described in §3.1.1. Similar to the Web experiments, we analyze application performance across 12 different CPU frequencies in one device and repeat the experiments over three smartphones. As before, we present the results from Nexus 4 and summarize the results from the other two phones.

3.4.1 Video Streaming: YouTube

Figure 18a shows the startup latency and stall ratio of YouTube for 12 different CPU clock frequencies 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. Even though the bitrate adaptation algorithm is turned on during our experiments, we do not see any change to the video resolution during the experiment.

This trend is similar to the one observed in Figure 10b across low-end and high-end phones. Experiments on Intex Amaze+ and Google Pixel 2 show quantitively similar trends (not shown here).

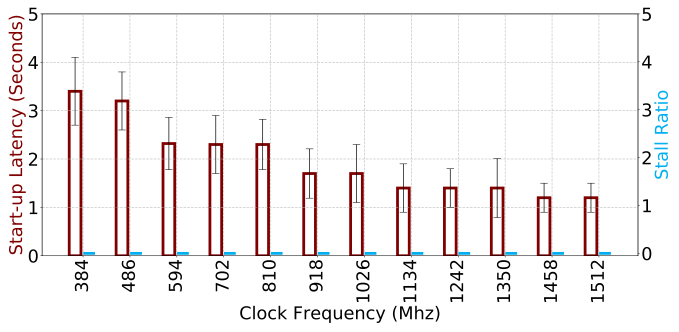
Network: During video streaming, video content is downloaded from the network and then processed by the video decoder. Unlike the Web, the network and compute operations are not inter-leaved, making it easier to separate the two. To isolate the effect of network on video performance, we measure application QoE before video decoding. We first estimate the *time-to-first-frame*, as the time it takes to download video content before the first frame is displayed. Our experiments show that the difference in time-to-first-frame between slow and fast CPU can explain the startup latency performance in Figure 18a. The time-to-first-frame increases when the CPU speed is low because it takes longer to process the TCP packets, affecting the startup latency (results omitted for brevity). In fact, the start up latency is not affected by the compute operations at all.

In practice, the stall ratio is a more important QoE metric, since the startup latency is only a one-time effect. We first look at why the stall ratio is not affected under slow CPU conditions, even though network throughput drops when the CPU is slow. YouTube and other video streaming services [61, 43] prefetch video content; YouTube prefetches 120 seconds worth of content. The Read Ahead Convergence (RAC) time is the time required for YouTube to fetch 120 seconds worth of content.

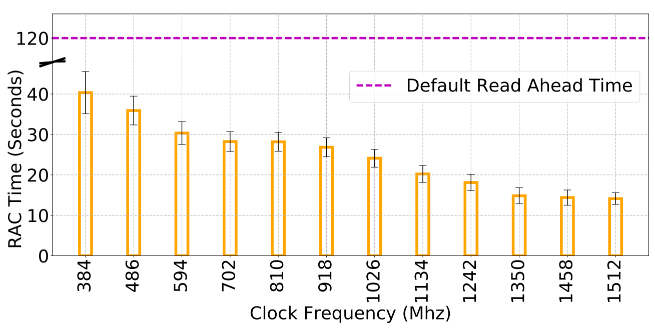
Figure 18b shows the RAC time across different CPU speeds. There is a difference of 25 seconds in RAC time between the slow and fast CPU clock. However, even under slow clock, the RAC time to fill the buffer is well under 120 seconds required to play the video. Therefore, the increase in RAC time does not introduce any stalls, resulting in no effect on the stall ratio.

Compute: Isolating the effect of compute for YouTube is harder because YouTube is a closed application, making it hard to modify the application. Instead, we conduct a set of experiments on VLC player [82], an open-source streaming application, and change the VLC player to use either software or hardware rendering/decoding. Our goal is to understand why streaming applications do not take a hit under slow CPU, even though video streaming is a compute-intensive application.

Hardware rendering involves offloading rendering and video decoding to hardware, which may either be the GPU or an in-built hardware accelerator. Software rendering refers to when the rendering and decoding is done in software. We cross-compile the Android VLC open source [82] using Android SDK tool chain [71].



(a)



(b)

Figure 17: (a) Effect of CPU speeds on YouTube: The start-up latency is increased by 50% but the stall ratio is not affected by the slow clock across the 12 CPU speeds. (b) Isolating the effect of network on stall ratio: The read-ahead convergence time (RAC) is the time required for YouTube to prefetch a read-ahead buffer worth of data (typically 120 seconds of video content). Even under slow clock, the time it takes to prefetch content is well below 120 seconds, which is the time it takes to play the content. As a result, read-ahead buffer is always full and the user does not experience stalls.

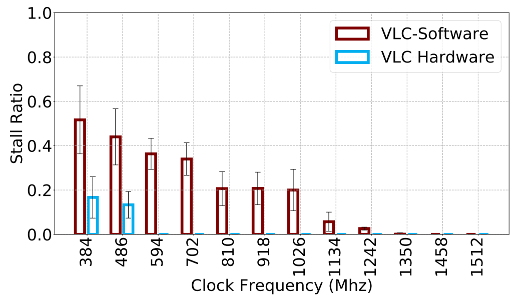


Figure 18: Isolating compute performance for video streaming by rendering videos locally via VLC player. When VLC is set to render in software, performance *is* affected by slow CPU. However, when rendering is offloaded to hardware, CPU clock does not affect video rendering performance. All devices in our experiment (Table 9) offload rendering to a hardware decoder.

Figure 19 shows that when decoding and rendering is performed in software, the stall ratio increases to 0.5 under slow CPU. In contrast, under fast CPU, the video experiences nearly no stalls. However, when decoding and rendering is offloaded to hardware, CPU speed only has a small effect on stall ratio when using the VLC player. Our earlier experiment shows that over YouTube, the effect of CPU speeds on stall ratio is in fact zero.

We verify (using Android logs) that YouTube indeed offloads video decoding to an in-built hardware, different from the GPU, and does not perform rendering in software. In fact, the hardware video decoder is available across all devices we tested, including the low-end Intex Amaze+ device. As a result, the stall ratio is not affected by CPU speeds.

3.4.2 *Video Telephony: Skype*

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. The performance of telephony is measured in terms of frame rate and the measurement set-up is described in §3.1.1.

Figure 20 shows the effect of clock frequency on frame rate during the Skype video call. The frame rate drops to 17 frames-per-second (fps) at slow CPU speeds from 30 fps at high CPU speeds.

Network: To see the isolated effect of clock on network, we measure two network-centric metrics: call set-up delay and bitrate during the Skype call. The call set up delay is the time between when the mobile client initiated the call and when the server accepted the call. We measure the call set up delay using the our screen recordings, as described in §3.1. The bit rate is the number bits received per second. The bit rate is different from the frame rate since it does not include rendering. Both bit rate and call set-up delay only depend on the network performance.

Figure 21 shows the impact of CPU speeds on the call set-up delay. 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. Figure 22 shows the impact of clock on bitrate. Again, we see that the bitrate decreases from 700 to 400 Kbps at slow CPU clock speeds. The key takeaway is that video telephony is significantly affected by slower CPU speeds because of the network processing delays. This is different from video streaming, where the effect of the network processing delays is masked by prefetching.

Compute: In terms of compute, there is little difference between streaming and telephony; both involve processing and rendering video frames on the screen. Similar to YouTube, Skype offloads rendering to the hardware decoder. As a result, changing the CPU clock frequency has little effect on telephony from the compute standpoint.

3.5 Discussions

In this section we discuss (a) the energy implications of changing the clock speed, and (2) a possible Web page optimization for low-end devices based on our observations in §3.3.

3.5.1 *Energy Consumption*

CPU clock frequency has an impact on the device's energy consumption and here low end devices may win. To evaluate energy consumption we use the same set up as the one described in §3.1.1. The experiments are conducted on the Nexus 4 phone and we use the Snapdragon Profiler [63] to log the energy consumption. The profiler samples energy

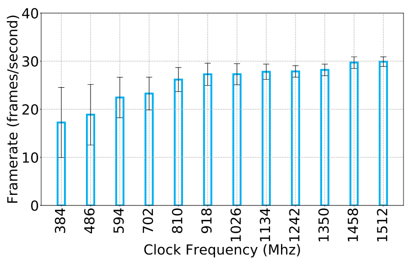


Figure 19: Skype QoE: The frame rate decreases by 33% when the CPU clock speed decreases.

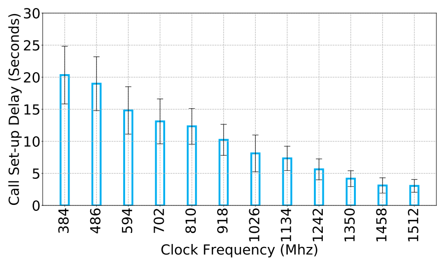


Figure 20: Effect of CPU speeds on the network-centric metric, call set-up delay.

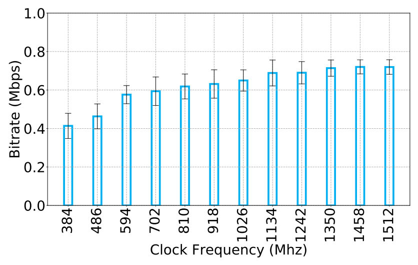
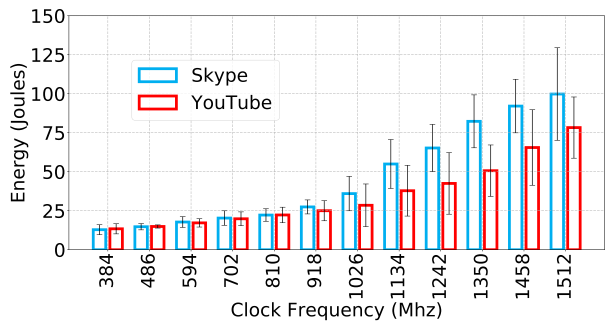
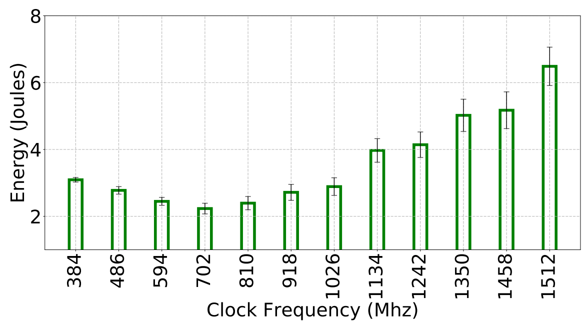


Figure 21: Effect of CPU speeds on the network-centric metric, bitrate.



(a) *Youtube and Skype*



(b) *Web page load*

Figure 22: Energy consumption vs. clock rate for (a) video streaming and telephony and (b) Web page load

at a high sampling rate of 20 times a second.

Figure 23 shows the energy consumption for all three applications considered in this study with increasing clock frequency. Figure 23(a) shows the energy consumption while running YouTube and Skype at different frequencies. With increase in CPU clock frequency energy consumption increases steadily - somewhat faster for telephony (Skype) than streaming (YouTube). One reason for this is that YouTube prefetches video content. This allows the network to be inactive after the required content is prefetched. But Skype is interactive requiring that the network remain active throughout the session.

Overall, the energy consumptions are more drastic than the QoE improvements in these two applications with faster clocks. For example, from the slowest end to fastest end of clock frequency 1) Skype consumes 7× more energy while providing about 2× better frame rate; 2) YouTube provides a modest improvement of startup latency, ≈1 sec vs. ≈3 sec for about a 5 min video, but consumes 5× more energy.

For Web page load (Figure 23(b)), the observation is different. As CPU frequency increases, the energy consumption first decreases and then increases. This is because initially the faster page load compensates more than the increased power draw due to faster clock. Recall that energy consumption = avg. power draw × time. Similar to the video example, energy consumption is more drastic than QoE improvement (Figure 13).

Thus, an optimal frequency exists if one is interested in optimizing the energy consumption for applications. Currently, when the frequency governor [44] is set to powersaving, the application is set to use the lowest CPU frequency. Instead, one can design a governor that takes into account the marginal performance improvement and the resulting energy consumption.

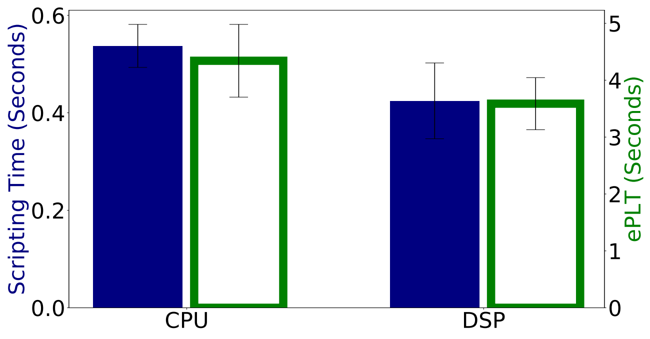
3.5.2 *Speeding Up Web Page Load*

We learnt from our Web experiments (§3.3) that CPU frequency impacts web browsing performance more than the other applications considered. This is a concern for low-end phones as browsing performance disproportionately suffers on these platforms. Video applications, on the other hand, are not severely impacted by CPU speeds because they offload compute to hardware.

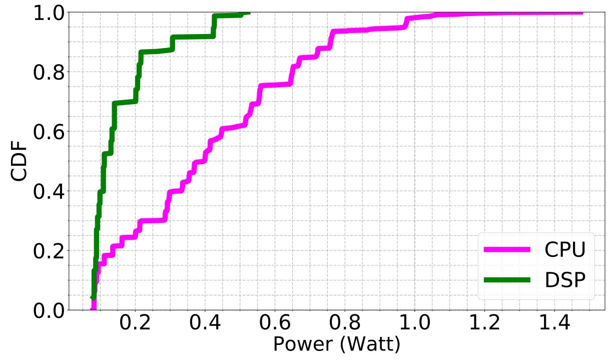
In this section, we conduct a *what-if* analysis to explore if offloading compute to a co-processor can also improve performance of Web page loads under low clock frequencies. Many mobile applications exploit co-processors like GPU and DSP and other specialized hardware accelerators. These co-processors provide faster processing and lower energy consumption if used effectively for appropriate tasks (e.g., data parallel computing for GPUs or floating point operations for DSPs.)

To this end, we systematically look at computations during Web page load. Scripting/Javascript execution is the most time-consuming computation (Figure 16). By manually evaluating scripting functions for the slowest set of Web page loads in our experience, i.e., the sports pages (Figure 17), we find that a significant time is spent in regular expression (regex) evaluations (e.g., for URL matching, list operations). Our goal is to explore offloading the regular expression evaluation to the DSP. GPU is not appropriate as these computations are not data parallel.

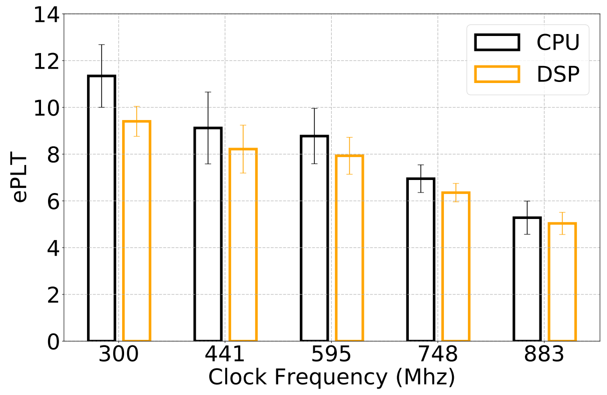
The following paragraphs describe the experiments and the evaluations leading to plots in Figure 24. In summary, offloading the regex part of Javascript execution to the DSP provides a modest improvement in estimated PLT when the mobile device is run with default frequency governors, where the CPU frequency is set by the OS (Figure 24a). A significant improvement is, however, noticed in the power con-



(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 CPU frequencies

Figure 23: Evaluations for DSP offloading of Javascript functions

sumption – almost 4× reduction in median power consumption (Figure 24b). The PLT improvements are larger (up to 25%) when the Web page is loaded under slower CPU speeds (Figure 24c). While not completely conclusive, these evaluations show that offloading of the compute intensive part of Web browsing to co-processors does have potential and should be explored.

Details of Evaluation: We use Qualcomm Hexagon SDK [64] to offload Javascript functions to the DSP. As the SDK is limited to low-level languages such as C/C++, it is not straight-forward to port Javascript code to the DSP. Instead, we extract regex functions from the Javascript and convert them to C-language libraries.

We then port the converted C code onto the *aDSP* processor on a Google Pixel 2 phone (which has the Snapdragon 835 Application processor). The Inter Process Communication (IPC) between CPU and DSP takes place through the FastRPC remote procedure call.

Since the offloading cannot be directly performed on the browser, we use nodejs to run these functions to evaluate their runtime performance. In effect, we record the page load process and extract the dependency graph with WProf [83]. WProf preserves the dependency and computation timing information during the page load. For each Javascript evaluation that contains regex, we estimate the time taken if the regex evaluation were offloaded. Based on this new timing, we re-evaluate the dependency graph and compute the new page load time that we call emulated PLT or ePLT. We experiment on the top 20 sports Web pages.

3.6 Summary

In this work, we analyze the impact of the device hardware on key mobile Internet applications – Web browsing, video streaming (YouTube), and video telephony (Skype). Our study is motivated by a survey of 7 diverse smartphone devices, ranging from $60 to $800 in cost, where show that device hardware does have an impact on the application's QoE. Our study then specifically focuses on the impact of CPU speeds on QoE. Our key takeaway is that Web browsing is significantly impacted by slow CPU speeds. At slow CPU frequencies, compute time increases but network processing time also increases. The latter is because of a second-order effect we observe, where slow CPU significantly increase the latency of TCP packet processing. Video applications, especially streaming, is less affected by slow CPU speeds. This is largely because (a) streaming applications offload decoding to special hardware that is available in all phones including low-end phones, and (b) streaming applications prefetch video data, and this prefetching masks the network processing latency. We supplement the performance study by exploring possible offloading mechanisms for Web page loads that have potential for improving the Web experience for low-end phones while being energy efficient at the same time.

4. RELATED WORKS

4.1 Network-specific Optimizations

There is an extensive prior work on video QoE modeling. The vast majority of works [23, 25, 28] introduce machine learning methods to map QoS to QoE, and assume availability of the QoE ground-truth. Features for QoE estimation can be collected from various vantage points, such as on the application server [25], on the end-device (application-client

statistics or packet capturing) [39, 38, 35, 23, 24, 27], or the access network (e.g., WiFi AP or LTE base station) [16, 30, 17]. Compared to the above works, we focus on annotating QoE ground-truth, hence easing the extension of QoS to QoE mappings to all video-telephony applications. Our work significantly advances state-of-the-art QoE labeling at the training phase of QoS to QoE, by introducing accurate metrics that capture spatial and temporal video quality.
Spatial quality assessment: Prior works have proposed multiple metrics to capture video blurriness (spatial artefacts). Jana et al. [30] introduce metrics for freezes, blocking and blur over Skype and Vtok. However, as we show in Section 2.1, this blur metric is highly content dependent. We have found similar results for major prior works [29, 33, 36, 32] on no-reference blur detection. In Section V, we present more than 1 MOS lower estimation error for our metrics compared to prior work over 20 diverse videos. We have not observed blocking artefacts in any of Skype, FaceTime or Hangouts applications.
Temporal quality assessment: Several works [87, 26, 37, 34] have investigated the effect of video freezes on user QoE. Wolf and Pinson [87] propose to use the motion energy temporal history of videos, along with framerate of video. This requires additional information from original video hence, making it a reduced reference metric. Similarly, the temporal metric of Jana et al. [30] is a reduced-reference metric. The temporal metric by Pastrana-Vidal and Gicquel [34] is sensitive to resolution and content of the video. Different from all of these works, we present three content-independent, no-reference temporal metrics to assess temporal video artefacts.

4.2 Device-specific Optimizations

There is an extensive literature on studying and improving Web and video performance for mobile devices. We discuss these in next.
Web performance: There has been considerable work in improving the Web page load time by optimizing the network activities of the page load process. Works including Klotski [15], Polaris [62], and Vroom [70] prioritize network object loading by taking into account the dependencies across network activities. Shandian [84], Nutshell [75], and Parcel [76] use a Web proxy to perform the network activities, and only send the resulting Document Object Model (DOM) to the mobile device. Zhen *et al.* [85] uses a client-only approach to predict the sub-resource of webpage from a given URL and then to speculatively load these predicted sub-resources.

Google's compression proxy [9] compresses web content to significantly reduce the use of expensive cellular data. Flexiweb [74] implements a framework based on Flywheel which uses object size and the network condition information to ensure that the middlebox does not affect the page load time. Balachandran *et al.* [13] models mobile user browsing experience (QoE) in a cellular network. They use mobile traffic data of HTTP session records and radio level information(including handover, throughput, power level) to predict web QoE metrics. Qian *et al.* [66] provide detailed measurement study on mobile browsers focusing on cellular data and energy usage.

Leo *et al.* [59] develop a parallel algorithm that shows the advantage of using multicore architectures for mobile browsers. The Webcore work [89] optimizes the mobile hardware architecture to improve PLT and minimize energy consumption.
Video Performance: Rajaraman *et al.* [67] dissect the energy consumption of live video streaming from smartphones. Hoque *et al.* [45] explores ways of optimizing energy consumption by looking at different characteristics of the device such as network used and user habits. In [78], the authors study the energy consumption of streaming applications on mobile GPU. These works minimize energy consumption in mobile devices by customizing the applications but not study the corresponding the QoE.

Balachandran *et al.* [24] propose new QoE metrics for Internet video. Jian *et al.* [51, 79] propose data-driven approaches to cover all the parameters that impact QoE, in the video delivery path. They show that the QoE can be largely improved by adapting bitrate by data-driven throughput prediction. Huang *et al.* [49] study the effect of network on quality of video streaming applications in mobile devices.

Different from these works, our studies focusses on studying the impact of device hardware on Web and video applications.

5. REFERENCES

[1] Amazon Mechanical Turk. https://www.mturk.com/.

[2] FFmpeg. ffmpeg.org/.

[3] http://az-screen-recorder.en.uptodown.com/android.

[4] Micro- vs. Macro-Average. https://www.clips.uantwerpen.be/ vincent/pdf/microaverage.pdf.

[5] Space Pirate Trainer. www.spacepiratetrainer.com/.

[6] theBlu: Encounter. wevr.com/project/theblu-encounter.

[7] Smartphone Stats 2017. https://perma.cc/ya27-x5hf.

[8] ADB. developer.android.com/tools/help/adb.html.

[9] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. Flywheel: Google's data compression proxy for the mobile web. In *NSDI*, volume 15, pages 367–380, 2015.

[10] YouTube Player API. https://developers.google.com/youtube/android/player/.

[11] Http Archive. https://httparchive.org/reports.

[12] Android View Cient (AVC). https://github.com/dtmilano/androidviewclient.

[13] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, and He Yan. Modeling web quality-of-experience on cellular networks. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 213–224. ACM, 2014.

[14] Geoffrey Blake, Ronald G Dreslinski, Trevor Mudge, and Krisztián Flautner. Evolution of thread-level parallelism in desktop applications. *ACM SIGARCH Computer Architecture News*, 38(3):302–313, 2010.

[15] 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.

[16] Ayon Chakraborty, Shruti Sanadhya, Samir R Das, Dongho Kim, and Kyu-Han Kim. Exbox: Experience

management middlebox for wireless networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 145–159. ACM, 2016.

[17] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. Quantifying skype user satisfaction. In *ACM SIGCOMM*, 2006.

[18] Ling-Jyh Chen and Ting-Kai Huang. Effective file transfer in mobile opportunistic networks. *Mobile Opportunistic Networks: Architectures, Protocols and Applications*, page 205, 2016.

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

[20] Luis Corral, Ilenia Fronza, Nabil El Ioini, Andrea Janes, and Peter Plant. Preserving energy resources using an android kernel extension: a case study. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, pages 23–24. ACM, 2016.

[21] Li Cui and Alastair R Allen. An image quality metric based on corner, edge and symmetry maps. In *BMVC*, pages 1–10, 2008.

[22] Harris Drucker. Improving regressors using boosting techniques. In *ICML*, 1997.

[23] Aggarwal et.al. Prometheus: toward quality-of-experience estimation for mobile apps from passive network measurements. In *ACM HotMobile*, 2014.

[24] Balachandran et.al. A quest for an internet video quality-of-experience metric. In *ACM Hotnets*, 2012.

[25] Balachandran et.al. Developing a predictive model of quality of experience for internet video. In *ACM SIGCOMM*, 2013.

[26] Borer et.al. A model of jerkiness for temporal impairments in video transmission. In *IEEE QoMEX*, 2010.

[27] Chen et.al. Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis. In *ACM IMC*, 2014.

[28] Fiedler et.al. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24(2), 2010.

[29] Golestaneh et.al. No-reference quality assessment of jpeg images via a quality relevance map. *IEEE signal processing letters*, 21(2):155–158, 2014.

[30] Jana et.al. Qoe prediction model for mobile video telephony. *Multimedia Tools and Applications*, 75(13):7957–7980, 2016.

[31] Marichal et.al. Blur determination in the compressed domain using dct information. In *IEEE ICIP*, 1999.

[32] Marziliano et.al. A no-reference perceptual blur metric. In *IEEE ICIP*, 2002.

[33] Mittal et.al. No-reference image quality assessment in the spatial domain. *IEEE Transactions on Image Processing*, pages 4695–4708, 2012.

[34] Pastrana-Vidal et.al. Automatic quality assessment of video fluidity impairments using a no-reference metric. In *VPQM*, 2006.

[35] Seufert et.al. A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys & Tutorials*, pages 469–492, 2015.

[36] Tong et.al. Blur detection for digital images using wavelet transform. In *IEEE ICME*, 2004.

[37] Usman et.al. A no reference video quality metric based on jerkiness estimation focusing on multiple frame freezing in video streaming. *IETE Technical Review*, 34(3):309–320, 2017.

[38] Yu et.al. Can you see me now? a measurement study of mobile video calls. In *IEEE INFOCOM*, 2014.

[39] Zhang et.al. Profiling skype video calls: Rate control and video quality. In *IEEE INFOCOM*, 2012.

[40] Cisco VNI Forecast. Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper. 2017.

[41] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[42] Cao Gao, Anthony Gutierrez, Madhav Rajan, Ronald G Dreslinski, Trevor Mudge, and Carole-Jean Wu. A study of mobile device utilization. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, pages 225–234. IEEE, 2015.

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

[44] Android Governors. https://androidmodguide.blogspot.com/p/blog-page.html.

[45] Mohammad Ashraful Hoque, Matti Siekkinen, Jukka K Nurminen, and Mika Aalto. Dissecting mobile video services: An energy consumption perspective. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pages 1–11. IEEE, 2013.

[46] https://developer.chrome.com/devtools. Chrome developer tools.

[47] https://www.jio.com/en-in/4g plans. 4g data plans, jio.

[48] https://www.ndtv.com/business/jios-sub-rs-200-prepaid-recharge-plans-with-more-data-than-before-budget 2018-1808464. 4g data plans, jio.

[49] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 165–178. ACM, 2010.

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

[51] 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.

[52] Christopher Grant Jones, Rose Liu, Leo Meyerovich, Krste Asanovic, and Rastislav Bodik. Parallelizing the web browser. In *Proceedings of the First USENIX Workshop on Hot Topics in Parallelism*, 2009.

[53] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. Improving user perceived page load

times using gaze. In *NSDI*, pages 545–559, 2017.

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

[55] Haohui Mai, Shuo Tang, Samuel T King, Calin Cascaval, and Pablo Montesinos. A case for parallelizing web pages. In *HotPar*, 2012.

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

[57] media.xiph.org/video/derf/. Xiph.org Video Test Media.

[58] Ensemble Methods. http://scikit-learn.org/stable/modules/ensemble.html.

[59] Leo A Meyerovich and Rastislav Bodik. Fast and parallel webpage layout. In *Proceedings of the 19th international conference on World wide web*, pages 711–720. ACM, 2010.

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

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

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

[63] Qualcomm Development Network. developer.qualcomm.com/software/snapdragon-profiler.

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

[65] Vladimir Pestov. Is the k-nn classifier in high dimensions affected by the curse of dimensionality? *Computers & Mathematics with Applications*, 65(10):1427–1437, 2013.

[66] Feng Qian, Subhabrata Sen, and Oliver Spatscheck. Characterizing resource usage for mobile web browsing. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 218–231. ACM, 2014.

[67] Swaminathan Vasanth Rajaraman, Matti Siekkinen, and Mohammad A Hoque. Energy consumption anatomy of live video streaming from a smartphone. In *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*, pages 2013–2017. IEEE, 2014.

[68] Iain E Richardson. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004.

[69] Vaspol Ruamviboonsuk, Ravi Netravali, Muhammed Uluyol, and Harsha V. Madhyastha. Vroom: Accelerating the mobile web with server-aided dependency resolution. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 390–403, New York, NY, USA, 2017. ACM.

[70] Vaspol Ruamviboonsuk, Ravi Netravali, Muhammed Uluyol, and Harsha V Madhyastha. Vroom: Accelerating the mobile web with server-aided dependency resolution. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 390–403. ACM, 2017.

[71] Android SDK. https://developer.android.com/sdk/download.html.

[72] BT Series. Methodology for the subjective assessment of the quality of television pictures. *Recommendation ITU-R BT*, pages 500–13, 2012.

[73] Muhammad Zubair Shafiq, Jeffrey Erman, Lusheng Ji, Alex X. Liu, Jeffrey Pang, and Jia Wang. Understanding the impact of network dynamics on mobile video user engagement. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, pages 367–379, New York, NY, USA, 2014. ACM.

[74] Shailendra Singh, Harsha V Madhyastha, Srikanth V Krishnamurthy, and Ramesh Govindan. Flexiweb: Network-aware compaction for accelerating mobile web transfers. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 604–616. ACM, 2015.

[75] 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 the 23rd Annual International Conference on Mobile Computing and Networking*, pages 448–461. ACM, 2017.

[76] 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 the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 325–336. ACM, 2014.

[77] Smartphone Web Stats. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/.

[78] Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. Energy efficient continuous multimedia processing using the tegra k1 mobile soc. In *Proceedings of the 7th ACM International Workshop on Mobile Video*, pages 15–16. ACM, 2015.

[79] 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.

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

[81] Unity. http://hwstats.unity3d.com/mobile/.

[82] Android VLC. https://code.videolan.org/videolan/vlc-android.

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

[84] Xiao Sophia Wang, Arvind Krishnamurthy, and David

Wetherall. Speeding up web page loads with shandian. In *NSDI*, pages 109–122, 2016.

[85] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. How far can client-only solutions go for mobile browser speed? In *Proceedings of the 21st international conference on World Wide Web*, pages 31–40. ACM, 2012.

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

[87] Stephen Wolf and M Pinson. A no reference (nr) and reduced reference (rr) metric for detecting dropped video frames. In *VPQM*, 2009.

[88] Mengbai Xiao, Viswanathan Swaminathan, Sheng Wei, and Songqing Chen. Dash2m: Exploring http/2 for internet streaming to mobile devices. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 22–31, New York, NY, USA, 2016. ACM.

[89] 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.