Improving Mobile Internet Quality of Experience

Mallesham Dasari
Computer Science Department, Stony Brook University, Stony Brook, NY

ABSTRACT

Mobile Internet applications have become predominant with the prolific increase in smartphone usage. Typically, these applications are Web browsing and video applications such as streaming and interactive telephony. The problem here is that the applications are influenced by a number of factors such as the network parameters, hardware performance and the inherent application complexity. This makes it hard to pinpoint and diagnose the problem, bringing new challenges in providing a good quality of experience (QoE) for these applications. To this end, we first conduct a systematic survey of existing literature that makes effort to improve the mobile Internet QoE. We then show the evidence for limitations of prior work in QoE optimizations in terms of network and device performance. Our contributions of this work are twofold. First, we propose a machine learning based QoE model that is hooked on top of WiFi access point or a cellular base station. The model learns the QoE of applications under diverse network conditions and help the network administrator in network resource provisioning according to the end-user QoE rather than traditional quality of serive (QoS) parameters. Second, we dig deep into the effect of underlying device performance to understand device side bottlenecks while providing unlimited network resources. This investigation brings a variety of strategies to content providers in placing the optimizations at mobile device or the network side which is especially important for the people of developing regions. Our experimental results shows that the behavior of these applications varies significantly from one another with respect to both network and hardware resources. We show the performance of these applications by selecting most popular applications such as YouTube for streaming, Skype for telephony and Google chrome for Web browsing.

1. INTRODUCTION

2. APPLICATION INDEPENDENT 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) [37]. 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 between network Quality-of-Service (QoS) and end-user QoE for video applications [22, 20, 25], by using machine learning (ML) models and network features from PHY to TCP/UDP layers. ML models for QoE can be deployed at a number of vantage points in the network, such as access points, network controllers, cellular packet core.

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 information [36] or YouTube API [20]; call duration [14]; or instrumented client-side libraries for video delivery platform [22]. 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, network administrators need an application-independent approach to measure QoE. Note that application independence does not mean that modeling is application-agnostic, as a single QoE model cannot apply to all applications. 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* [27], where we record the screen on the mobile device and estimate video quality. Different from previous works, we exploit video compression methods (Section 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 (Section 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 Section 2.4.

In summary, 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).

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 [20], 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 [22] or initial training of QoS to QoE model [20]. 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).

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

Table 1: Availability of QoE ground-truth

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* [27] 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. For example, one can wonder if *blur* is sensitive to the video content.

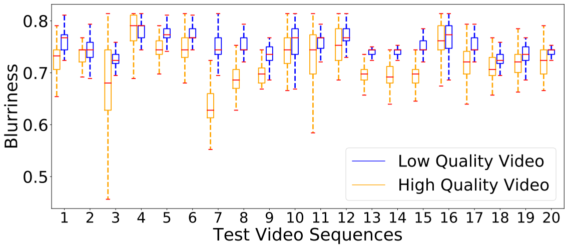


Figure 1: Blur detection using DCT coefficients [27, 28]. DCT coefficients fail to distinguish between high and low quality video due to content diversity.

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* [27] employ discrete cosine transform (DCT) coefficients [28] 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. 1, 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.

We notice two aberrations of the DCT metric by Jana *et al* [27]: 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

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

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 [26, 30, 33, 29], 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* [27] 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 [63, 23, 34, 31] 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 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 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 [54] 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 [54].

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.

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

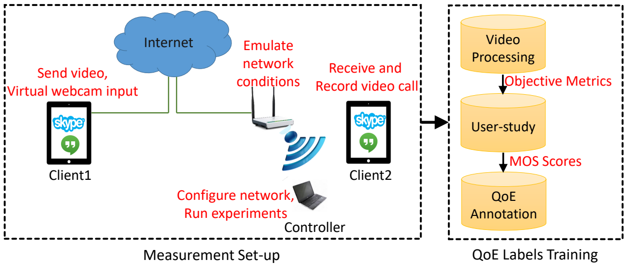


Figure 2: Measurement set-up and system architecture. *Left:* Video telephony measurements and recording video calls. *Right:* Our framework processing recorded videos offline and extracting objective metrics to predict QoE labels.

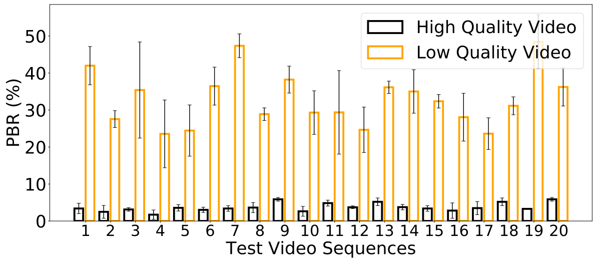


Figure 3: Blur detection using PBR.

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.

2.3.1 *Measurement Methodology*

Set-up: In Figure 2, 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* [49], 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 [10]. 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.

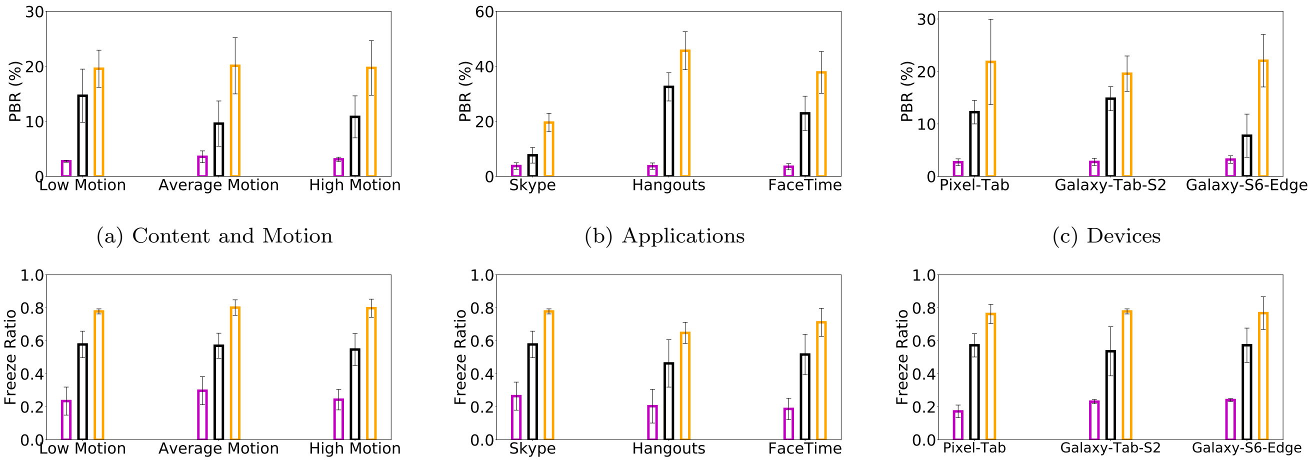
Network conditions: As we experiment real-time with interactive applications (Skype, Hangouts, FaceTime), we need to emulate real user experience under any condition and to obtain samples with MOS values of all levels (1–5). Hence, we conduct tests with ideal network conditions and with throttled network. 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 [50] and YouTube. We select 20 representative videos that contain different types of motion and content. All videos are downloaded in Full HD (1920x1280) resolution.

2.3.2 *Our QoE Metrics*

PBR: Video bitrate has been considered a standard metric for perceptual quality of video [36, 13]. 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 advan-

[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.



(a) Content and Motion

(b) Applications

(c) Devices

(d) Content and Motion

(e) Applications

(f) Devices

Figure 4: 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).

tage 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. 3 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 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 (for instance a black screen throughout the video). 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. 4 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. **??**). The average freeze ratio for best network conditions is smaller than 0.3 and for bad conditions, it is larger than 0.8 (Fig. **??**). 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. **??**, **??**, 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 under 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. **??**, **??**, 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 [56][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 video clips and ask the users to rate their experience after watching each sample.

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. 5 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 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.

[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. Evidence to be provided upon paper acceptance.

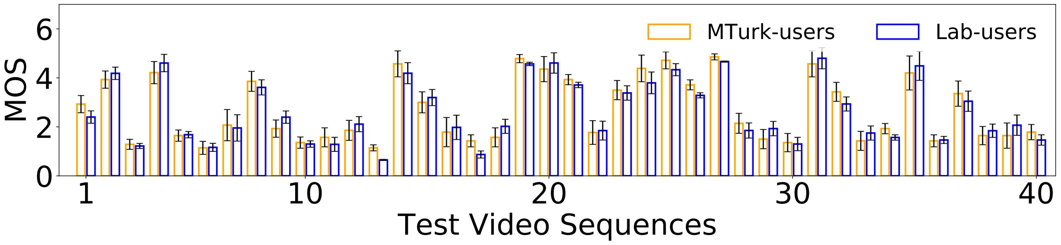


Figure 5: MOS and standard deviation of QoE scores from 30 lab and 30 online users. Although the scores vary across the users both in lab and online, the distribution of MOS is similar in two cases.

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 [18]. Our regression model employs the average MOS from 15 users as ground-truth per video clip and it is based on ensemble methods [51]. 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`[38], 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 [38][19].

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

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

features [53], 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 [51].

Fig. 6 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 employ 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

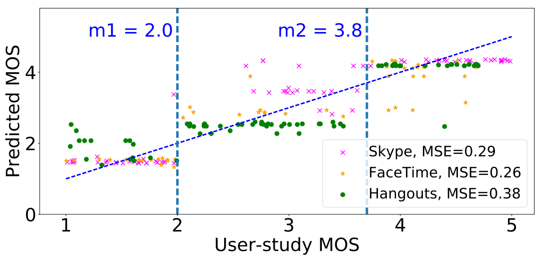


Figure 6: User-study vs. predicted MOS for the three applications. We also show 3 clear QoE clusters divided by our thresholds $m_1 = 2$ and $m_2 \approx 3.8$.

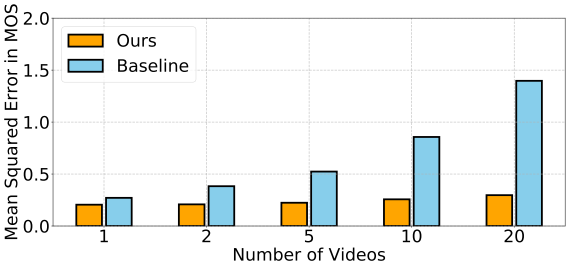


Figure 7: Comparison of our metrics vs. baseline. Baseline has an up to 1.4 MOS estimation error across 20 video contents, hence failing to distinguish good/bad from average QoE.

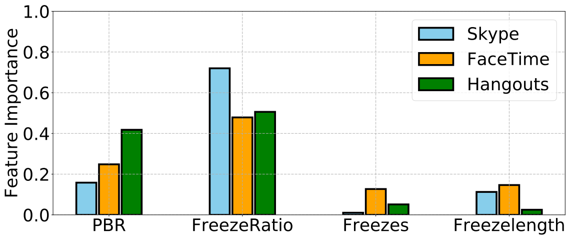


Figure 8: Feature Importance for the three applications.

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 [34] 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. 7 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 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. 8 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 Related Work

There is an extensive prior work on video QoE modeling. The vast majority of works [20, 22, 25] 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 [22], on the end-device (application-client statistics or packet capturing) [36, 35, 32, 20, 21, 24], or the access network (e.g., WiFi AP or LTE base station) [13, 27, 14]. 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. [27] 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 [26, 30, 33, 29] 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 [63, 23, 34, 31] have investigated the effect of video freezes on user QoE. Wolf and Pinson [63] 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. [27] is a reduced-reference metric. The temporal metric by Pastrana-Vidal and Gicquel [31] 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.

2.8 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.

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, 16], 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 [12, 46, 58, 55, 57, 64, 15] but not the device. Some studies do address the compute performance of mobile Internet applications [65, 45, 48], 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 [60]) 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, 16, 42, 43]. 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 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 [61, 52] 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.

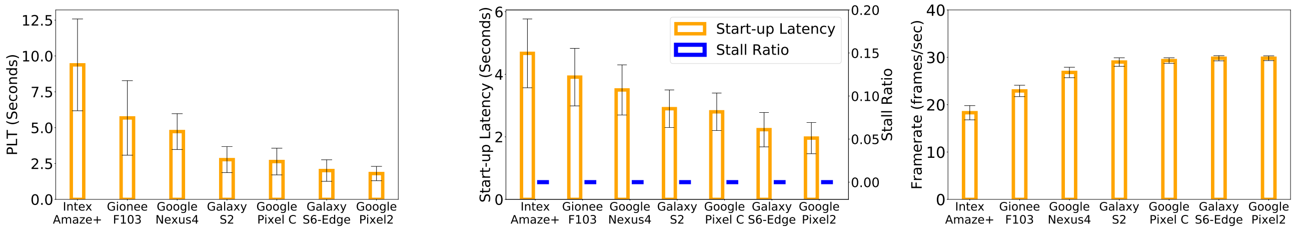
- **A slow clock does not significantly affect video streaming because of offloading and pre-fetching:** The QoE of video streaming is largely unaffected by slow CPU speeds, despite video streaming being a compute-intensive application. By isolating the impact of compute and network, we find that this 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.

- **Increasing CPU speeds beyond a inflection point has poor performance/power trade-off:** For all three applications, increasing CPU speeds significantly increases energy consumption. However, the corresponding improvement in performance beyond a point is not commensurate with the energy cost.

- **Leveraging hardware offloading is a promising approach to improving Web performance under slow CPU clock:** Similar to the hardware offloading techniques used in video applications, offloading Javascript evaluation 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.

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.



(a) Web Browsing

(b) Video Streaming

(c) Video Telephony

Figure 9: 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.

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 [61]. We load the top 50 Web pages from the Alexa [62] and estimate the average PLT. We automate the page loads for repeatability using Chrome remote debugging protocol [41] 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 [9]. 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 [35]. 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 [59].

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 [49] 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 [10].

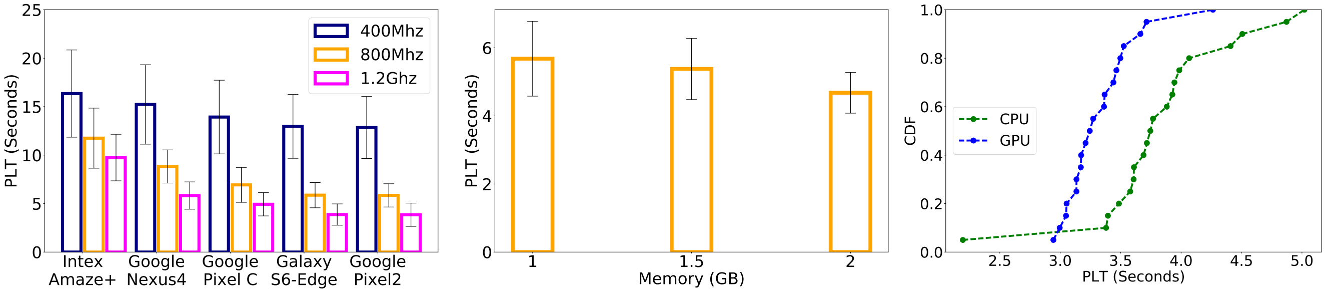
Network Set Up: As the focus of this work is to measure the impact of the device hardware, the experiments are set up to minimize the impact of the network and the Web/Video servers. For video streaming and video telephony, we host the videos on a windows laptop on our LAN. In case of Web, we host the 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 9 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 9a), 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



(a) Clock

(b) Memory

(c) GPU

Figure 10: 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.

Google Pixel 2. This variance must stem from the device itself, since the network conditions are the same.

In the case of YouTube (Figure 9b), 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 9c), 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 *frequency governor* setting. Frequency governors [40] 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 [39, 11] 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 [17].

Figure 10 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 [40] mentioned before. Figure 10a 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 10b and 10c 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 [47] (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.

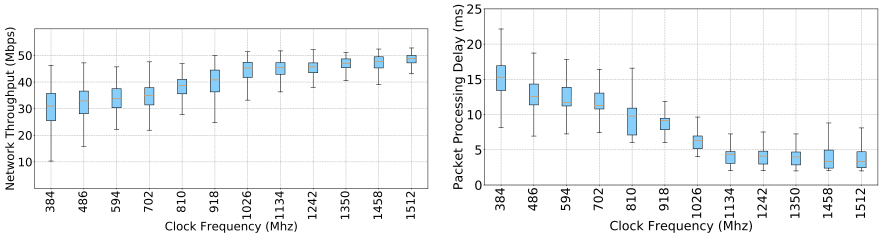
4. SECOND-ORDER EFFECT OF CPU CLOCK ON THE NETWORK

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 [44] 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. 11a 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 over-provisioned 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 11b shows the average delay to process a packet under different clock frequencies for the same IPerf experiment. Android times-

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



(a) Throughput vs. Clock

(b) Packet processing delay vs. Clock

Figure 11: 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.

tamps 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 11b 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 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.

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

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

[11] 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.

[12] 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.

[13] 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.

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

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

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

[17] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[34] 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.

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

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

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

[38] 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.

[39] 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.

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

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

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

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

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

[45] 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.

[46] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. Improving user perceived page load times using gaze. In *NSDI*, pages 545–559, 2017.

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

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

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

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

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

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

[53] 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.

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

[55] 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.

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

[57] 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.

[58] 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.

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

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

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

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

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

[64] 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.

[65] 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.