Coal Not Diamonds: How Memory Pressure Falters Mobile Video QoE

Talha Waheed[†]
LUMS

Zahaib Akhtar[‡]
Amazon Prime Video

Ihsan Ayyub Qazi
LUMS

Zafar Ayyub Qazi
LUMS

ABSTRACT

The popularity of video streaming on smartphones has led to rising demands for high-quality mobile video streaming. Consequently, we are observing growing support for higher resolution videos (e.g., HD, FHD, QHD) and higher video frame rates (e.g., 48 FPS, 60 FPS). However, supporting high-quality video streaming on smartphones introduces new challenges—besides the available network capacity, the smartphone itself can become a bottleneck due to resource constraints, such as low available memory. In this paper, we conduct an in-depth investigation of memory usage on smartphones and its impacts on mobile video streaming. Our investigation – driven by a combination of a user study, user survey, and experiments on real smartphones – reveals that (i) most smartphones observe memory pressure (i.e., low available memory scenarios), (ii) memory pressure can have a significant impact on mobile video QoE when streaming high-quality videos, e.g., resulting in the mean frame drop rate of 9-100% across smartphones and significantly lower user ratings, and (iii) the drop in mobile video QoE happens primarily due to the way in which video processes interact with kernel-level memory management mechanism, with opportunities for improving mobile video QoE through better adaptation by video clients.

CCS CONCEPTS

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

KEYWORDS

Mobile Video Streaming, Memory Pressure, Smartphones

ACM Reference Format:
Talha Waheed[†], Zahaib Akhtar[‡], Ihsan Ayyub Qazi, and Zafar Ayyub Qazi. 2022. Coal Not Diamonds: How Memory Pressure Falters Mobile Video QoE.

[†] Talha Waheed is currently at UIUC. The work was completed while he was at LUMS.
[‡] Work done for this paper is not related to Amazon employment.

CoNEXT '22, December 6–9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9508-3/22/12…$15.00
https://doi.org/10.1145/3555050.3569120

In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3555050.3569120

1 INTRODUCTION

Smartphones have become the primary devices for watching videos online [6]. In 2021, smartphones accounted for 63% of YouTube watch time worldwide [7]. Consequently, we are witnessing rising demands for high quality videos resulting in growing support for higher resolution videos (e.g., HD, FHD, QHD) and higher video frame rates (e.g., 48 FPS, 60 FPS). However, supporting such video types on smartphones introduces new challenges—besides the available network capacity, the smartphone itself can become a bottleneck due to resource constraints, such as low available memory.

In this paper, we conduct an in-depth investigation of memory usage on smartphones and its impact on mobile video streaming. Specifically, our study sheds light on the following questions: (i) how *frequently* do smartphones—based on users' natural usage patterns—experience memory pressure (low memory situations)? (ii) what is the *impact* of memory pressure on mobile video streaming performance? and (iii) how do *system-level mechanisms* for managing memory pressure in smartphones interact with video clients? We focus this study on Android devices as Android is the most popular mobile operating system today with more than 70% market share worldwide [14].

To shed light on the first question, we conduct an IRB approved user study involving 80 consenting participants and monitor the memory state of their smartphones for a period ranging from 1 to 18 days. Our study includes smartphones from 12 different manufacturers and a total of nearly 9950 hours of logged memory data. Our study shows that devices often operate under high memory utilization and most devices observe high memory pressure signals. In particular, 80% of the devices in our study had a median memory utilization of at least 60%, 63% experienced some form of memory pressure, and 19% of the devices received more than 10 memory pressure signals every hour on average, notifying critically low memory from the operating system (see Table 1). These insights, coupled with the popularity of video streaming on smartphones, makes it important to analyze the impact of memory pressure on video streaming performance.

To assess the impact of memory pressure on video performance, we perform controlled video experiments on multiple real smartphones, whose specifications cover a wide range of smartphones. To quantify video performance, we measure client-level metrics (e.g., frame drop rate and client crash rate) as well as report user ratings

derived from a survey. Our study shows that smartphones can experience significant frame drops under high memory pressure; with mean frame drop rates ranging from 9%-100% across smartphones, when streaming a 60 FPS video at 1080p resolution. In particular, an entry-level smartphone (1 GB RAM) experienced a frame drop rate of more than 75% on average for high resolution videos (720p, 1080p) and frequent crashes whereas a medium-end device (2 GB RAM) and a higher end (3 GB RAM) device experienced frame drop rates of up to 25% and 9% for 1080p videos, respectively. Our user survey involving 99 participants—who were shown videos under normal and moderate memory pressure states—shows that users assigned a significantly lower mean opinion score (MOS) to videos streamed under memory pressure.

To understand why video performance degrades under memory pressure, we perform fine-grained system-level analysis, using Google's system profiling tool, Perfetto [19]. Our investigation reveals how frame drops occur primarily due to the way video processes interact with kernel memory management daemons (e.g., *lmkd*, *kswapd*) responsible for finding free memory and managing disk I/O (e.g., *mmcqd*). We find the waiting time of video processes increases significantly under memory pressure, e.g., the time spent by video processes in preempted state increased by 97.8% under moderate memory pressure.

The findings of our study have implications for different stakeholders in the mobile video streaming ecosystem. First, our work shows that incorporating memory pressure as a feature in the design of future video adaptation algorithms can lead to improved performance.[1] Second, we find that memory pressure signals generated by the operating system to notify applications can be effectively used by video adaptation algorithms for *reacting* to memory pressure. Third, we identify two video features, namely video bitrate and frame rate, whose adaptation under memory pressure can lead to significant improvement in video QoE.

For Internet video platforms, our work has two important implications. First, providers should collect memory pressure signals as it has a role to play in determining client-side QoE. This additional visibility can help better disambiguate the complexities associated with troubleshooting client performance issues in the wild [26]. Second, platforms should consider offering a wider range of video encodings (e.g., bitrates and frame rates) to improve video QoE especially for low-end and medium-end smartphones.

For mobile operating systems, our study suggests that there is scope for reducing the interference caused by memory management daemons through improved scheduling. For example, we observe that *kswapd* frequently switches cores, however, if the allocation of cores is coordinated between daemons and video processes, reduced context switching overhead can potentially lead to improved performance. Finally, our work shows that video QoE can be improved under memory pressure if devices were equipped with more CPU resources (e.g., greater number of cores or cores with higher frequency). However, doing so can increase the manufacturing cost of smartphones. Table 1 provides a summary of our key insights.

[1]Video adaptation algorithm, such as adaptive bitrate algorithms, have traditionally focused on network bottlenecks (e.g., [24, 32, 35]), which are coupled with coarse-grained device-level measures (e.g., capping the maximum video resolution based on device screen resolution [21]).

To the best of our knowledge, this is the first study that analyzes the impact of memory pressure on mobile video performance. Altogether, we make the following key contributions.

- **Understanding real-world memory usage on smartphones:** We conduct a user study involving 80 users to understand how frequently mobile devices experience memory pressure, based on their natural usage patterns. As part of this study, we developed an Android application that can capture memory pressure signals from the kernel and record memory statistics. Based on the collected data, we share insights about the memory consumption of devices, their memory pressure state, and the time devices spent in high memory pressure states (§3).
- **Impact of memory pressure on video QoE:** To quantify the impact of memory pressure on mobile video QoE, we conduct controlled video streaming experiments under memory pressure—across different resolutions, frame rates, and video genres on three different smartphones with different memory/CPU configurations. We measure video client's rendering performance as well as conduct a user survey with 99 participants to capture actual user-perceived video QoE (§4).
- **Interaction effects of OS mechanisms on video QoE:** We quantify the interference caused by kernel-level memory management and disk I/O processes, such as *kswapd*, *lmkd*, and *mmcqd*, and study their interaction effects with video streaming under high memory pressure (§5).
- **Discussion on how video QoE can be improved under memory pressure:** We demonstrate potential opportunities for improving mobile QoE under memory pressure that involve adapting the video frame rate and/or the video bit rate. We also discuss other future directions, including, OS-level memory management and scheduling schemes (§7).

To make our work easy to reproduce, we have made our survey data, mobile app, experimental logs, and evaluation scripts available online in a public repository[2] (see Appendix A).

2 ANDROID MEMORY MANAGEMENT

Physical memory. The memory of a device is divided into fixed size *pages* that can be used by different processes. Typically, a page is 4 KB of memory. There are two types of pages: (1) *used pages*, which are being actively used by processes, and (2) *free pages*, which are pages that are not being used for anything. Used pages are further divided into *cached pages* (i.e., pages that are backed by a file in storage)[3] and (ii) *anonymous pages*, which are not backed by any file in storage. Note that system memory is used by both the CPU and the device GPU (if any) [17].

Memory management. Memory pressure refers to a state in which the device is short on memory thereby requiring the operating system to free memory by throttling or killing less important processes. Memory pressure increases when the memory usage of an application grows or a user ends up opening too many applications in the system. To avoid application crashes as well as poor user-perceived performance, Android makes use of two system daemons to deal with low memory situations: (i) the kernel swap daemon (*kswapd*)

[2]https://github.com/nsgLUMS/mobileVideo

[3]These could either be pages used by user-level apps—whose size is denoted by *cached PSS*—or the kernel, whose size is denoted by *cached kernel* in Android.

Key Insights	Description
(1) Frequency of memory pressure signals	63% of the smartphones in our user study experienced some form of memory pressure. We also observe that 19% of the devices received more than 10 memory pressure signals every hour on average, notifying critically low memory.
(2) Time spent in low memory states	10% of the smartphones in our study spent > 50% of the time in high memory pressure states and 35% spent ≥ 2% of the time in such states.
(3) Impact of memory pressure on frame drops	Entry-level smartphone (1 GB RAM) experienced more than 75% average frame drops for high resolution videos (720p, 1080p) and frequent crashes. Nexus 5 (2 GB RAM) experienced average frame drops up to 25%.
(4) Impact of memory pressure on subjective quality of experience (QoE)	Our user survey with 99 participants, which we conducted for capturing user-perceived video QoE related to frame drops, showed that users' experience degraded significantly under high memory pressure.
(5) Why video QoE degrades?	We find that the waiting time of video processes increased significantly under memory pressure due to interference from system daemons responsible for finding free memory and managing disk I/O (e.g., time spent in *Runnable (Preempted)* state increased by 97.8% in moderate memory pressure compared to the normal state).
(6) Opportunities for improvement	We show that mobile video QoE can be improved by adapting frame rate and/or bit rate. We also show that memory pressure signals that are generated by the OS can be effectively used for reacting to memory pressure.

Table 1: Key insights from our user study, user survey, and the experimental evaluation.

and (ii) low-memory killer daemon (*lmkd*) [13]. When the amount of free memory falls below a certain threshold, *kswapd* starts to reclaim used memory in the background.[4] If *kswapd* cannot free enough memory, the system sends memory pressure signals (e.g., using `OnTrimMemory()` in Android) to applications to ask them to reduce their memory allocations. If this is not sufficient and the amount of free memory falls even further, *lmkd* becomes active and starts killing processes in order of their priorities to make more free memory available [13, 31]. This means that under high memory pressure regimes, a video client application may have to operate alongside the background activity of *kswapd*, and will be susceptible to getting killed by *lmkd*.

Memory pressure signals for applications. Android generates different memory pressure signals to which applications can listen and respond by reducing their memory usage. These signals are generated when *kswapd* is not able to find enough free memory and are represented by multiple levels, which indicate the *degree* of memory pressure on a device. For foreground/running apps, these levels include: Moderate, Low, and Critical [5].[5] The Moderate signal is generated when the device is running moderately low on memory and *kswapd* has already started reclaiming cached memory but the application is not yet killable. The Low signal is generated when the lack of available memory will directly impact the foreground app's performance. Finally, the Critical signal is generated when the system is unable to keep almost any background process running, which can further degrade performance and eventually lead to the killing of the foreground app itself.[6]

Direct reclaim and thrashing. If despite the mechanisms used by memory management daemons, there is insufficient free memory available to allocate to a process, *direct reclaim* occurs [2]. The kernel responds by blocking the allocation until it can free up the memory requested to be allocated. This often requires disk I/O to flush out a modified storage-backed page or wait for *lmkd* to kill a process. This can cause an extra I/O wait in any thread, including the foreground application's main UI thread. Besides direct reclaim, another risk associated with low memory is *thrashing* [30]. This is

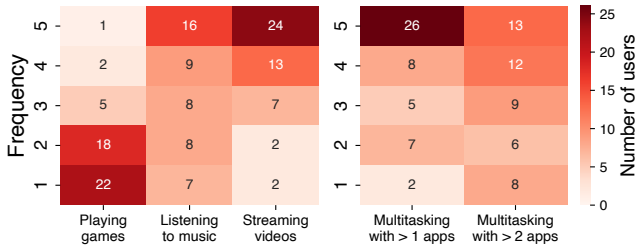

Figure 1: The heatmaps show, on a scale of 1-5, how frequently users engage in certain activities on their device.

a state of very frequent page faults, i.e., the kernel has to frequently pay disk I/O to bring recently deleted/flushed out storage-backed pages back into memory. Disk I/O involves kernel daemons like *mmcqd*, which manages queued I/O operations on storage, to kick in. Increased running of *mmcqd* can be critical as it is strictly prioritized over foreground processes and therefore can steal CPU time from them. Such I/O waits or stolen CPU times can be crucial for video client applications as video frames have to be rendered in real-time and a delay can cause frames to be dropped.

Killing of processes. Android classifies processes into different priority groups based on their expected responsiveness [22], each of which is assigned a score called `oom_adj` score based on the process priority, size of occupied memory and other factors. Low priority processes are assigned higher scores. When choosing to kill a process, *lmkd* picks the process with the highest `oom_adj` score. Internally, *lmkd* relies on memory pressure signals from the kernel to decide which process groups (i.e., processes with certain `oom_adj` scores) become eligible to be killed. The memory pressure, which is an estimate of memory utilization, is calculated as: $P = (1 - R/S) * 100$, where R and S are the number of reclaimed cached pages and the number of scanned memory pages, respectively. Thus, if most pages can be reclaimed, P would be low. However, if the number of cached pages decreases, this will lead to high memory pressure regimes signifying that the system is running under low memory. When $60 < P < 95$, processes with high `oom_adj` become eligible to be killed whereas when $P \geq 95$, foreground apps become eligible to be killed.

3 USER STUDY OF MOBILE DEVICES

In order to quantify how frequently mobile devices experience memory pressure based on users' natural usage patterns, we conducted a user study involving several Android devices. We recruited 80 users

[4] *kswapd* reclaims unmodified cached pages memory pages by simply deleting them from memory as they are storage-backed. It reclaims modified cached pages, and anonymous pages by compressing them to `zRAM`, which serves as an in-memory swap space in Android.

[5] Under normal conditions, the memory pressure state referred to as 'Normal'.

[6] More specifically, Android generates these memory pressure events by tracking the number of cached/background processes in the least-recently used (LRU) list. Because Android tries to aggressively cache processes at all times, a decreasing number of cached processes indicate increasing memory pressure. On a 1 GB RAM Nokia 1 device using Android 10 (Go edition), the threshold for Moderate, Low, and Critical events are set to 6, 5, and 3 cached/empty processes respectively.

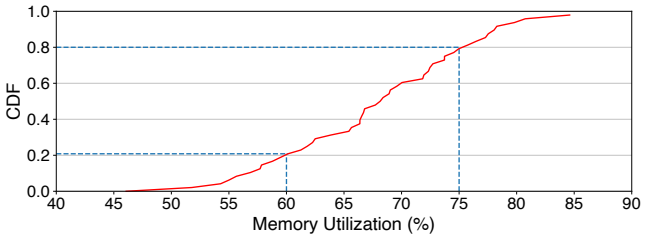

Figure 2: The CDF of the median RAM utilization across mobile devices in our study.

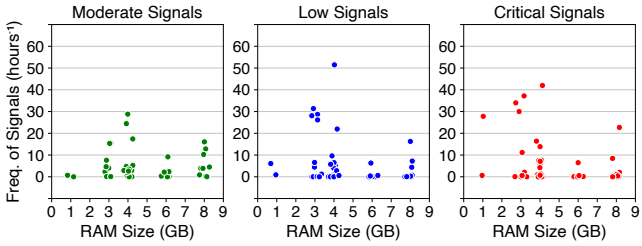

Figure 3: Frequency of memory pressure events across devices.

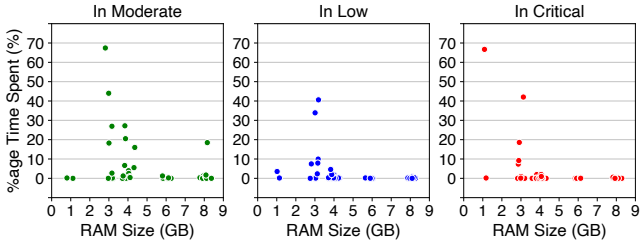

Figure 4: Percentage of time spent by mobile devices in our dataset in different memory pressure states.

and tracked the memory state of their smartphones over time. 81.3% of our users were under 25 years of age, 13.3% between 25-50 years of age and 5.4% were above 51 years in age. Most of the participants in our study were either university students or staff members. Our study included smartphones from 12 different manufacturers and a total of about 9950 hours of logged memory data, which equals to an average observation period of 124 hours per smartphone. The total device memory for the surveyed devices ranged from 1 GB to 8 GB. Before conducting this study, we obtained user consent for installing the application and informed the users about the exact information that will be collected from their devices. In addition, we also conducted a short survey from the study participants to inquire about their device usage patterns. The study was approved by the the Institutional Review Board (IRB) of the host institution.

User study methodology. We developed an Android application, `SignalCapturer`, that records the memory usage patterns of Android devices. The application was made available on Google Play Store from where the users installed the application. We have made the source code of the application public (see Appendix A). The information our application collects periodically (every second) includes the amount of available memory, current memory pressure state of the device (e.g., Moderate, Critical), whether the device is in interactive state, and the number of running services. We also collected information about the device's total memory, manufacturer, Android version, and the number of CPU cores. However, our application could not collect information about when a video was being streamed on the device. This is because, to maintain user privacy, Android does not allow an application to know which other applications are running on the device [18]. Therefore our findings in this user study hold for the general device workload, and are not specific to video streaming. However, we find through a survey of the users who installed the application that videos were very frequently streamed on the devices (see the paragraph on *Application Usage Patterns* below). `SignalCapturer` has a negligible CPU and memory footprint; on an entry-level smartphone with 1 GB RAM (Nokia 1), it has a mean memory footprint of 17 MB and a mean CPU usage of 0.3%.

Data cleaning and reporting. When reporting results from our study, we only include user devices for which we have more than 10 hours of memory data in interactive state (i.e, when the device screen is on). We do this to ensure that we have a reasonable sample of each user's memory data and that our analysis is not affected by log entries when the device is in idle state. This resulted in 48 users, each with more than 10 hours of memory data in interactive state.

Application Usage Patterns. We conducted a short survey from users who installed our mobile application to inquire about their device usage patterns. Users were asked to rate their frequency of usage, on a scale from 1 to 5, of three activities: *playing games*, *listening to music*, and *streaming videos*. We found that streaming videos was the most frequent activity followed by listening to music as shown in Figure 1. In addition, we asked how often they multitask, on a scale from 1 to 5, with more than one and two applications in the background. We find that multitasking is common with a large fraction of users reporting that they multitask with more than two applications running in the background.

Distribution of memory utilization. Figure 2 shows the CDF of the median RAM utilization across all the 48 devices. Observe that 80% of the devices had a median memory utilization of at least 60% whereas 20% of the devices experienced a median utilization of more than 75%. In the regime, when the memory utilization is 60% or more, the system marks certain process categories as kill-able based on `oom_adj` scores and *kswapd* can become active if free memory falls below a certain threshold.

Frequency of memory pressure signals. Next, we consider the frequency of different memory pressure signals generated by the devices. Figure 3 shows scatter plots of the frequency of receiving memory pressure signals on the y-axis (in terms of the number of signals per hour) as a function of RAM size of devices in our dataset across different memory pressure states. We find that 63% of the devices in our survey received at least one Moderate, Low, or Critical memory signal per hour. We also observe that 19% of the devices receive more than 10 Critical memory signals per hour and 17% of the devices receive more than 10 Moderate signals per hour. A small number of devices (6.3%) received more than 70 (Moderate, Low, and/or Critical) memory signals per hour.

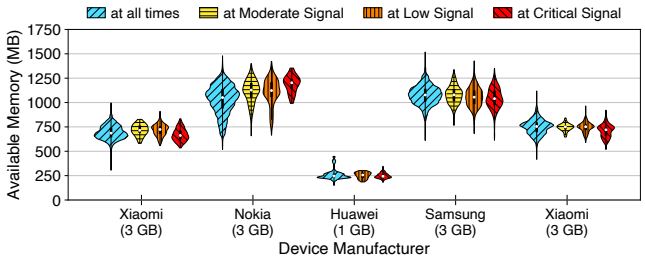

Figure 5: A violin plot showing the distribution of available memory across memory states for five devices that spent the most time in high memory pressure states.

Time spent in different memory pressure states. Figure 4 shows the fraction of time spent in high memory pressure states as a function of the RAM size of the devices. The figure shows that 27% of the devices spent ≥ 2% of the time in Moderate memory pressure state and 10% of the devices spent more than 4% time in Critical memory pressure state. There were two devices that spent more than 40% of the time in Critical memory.

Distribution of available memory. Finally, we show the distribution of available memory, which is the sum of free and cached bytes, across different memory states, for five devices that spent the largest fraction of time out of Normal memory state (see Figure 5). This is important because low memory thresholds that trigger *kswapd* and *lmkd*, and lead to generation of different memory pressure signals can differ across devices based on device memory as well as due to vendor-specific customizations. We make three observations from our data: (i) the available memory corresponding to each memory pressure signal has a significant *spread* indicating large variations in memory allocations by applications, (ii) across different memory pressure events, the mean available memory is generally *lowest* for Critical, followed by Low and then Moderate states[7], and (iii) the available memory at which different memory events get generated *differs* across devices, reflecting both vendor choices as well as device memory (e.g., we observed that devices from the same vendors had higher available memory thresholds for generating memory pressure events if they had larger RAM sizes).

Transitions between different memory pressure states. Figure 6 shows the trend of transitions from one memory pressure state to the next.[8] The upper row of bar plots show that when devices move into a particular memory pressure state, what is the next memory pressure state they move into. Specifically, the figure shows the percentage of times the devices move to a given memory pressure state. The bottom row of boxplots shows the durations of times devices spend in a particular memory pressure state before moving to some other memory pressure state next. The figure shows that high memory pressure regimes persist for a long time. For example, observe that after the Critical memory pressure

[7]One exception is the 3 GB Nokia device in our dataset for which the mean available memory is reported to be larger at Critical memory pressure. We conjecture that by the time our mobile app records the memory metrics, the operating system may have already reclaimed a large amount of memory, thus resulting in overestimation of the available memory.

[8]The data shown in Figure 6 is from the nine devices in our study that remained in non-normal memory pressure states for more than 30% of the time.

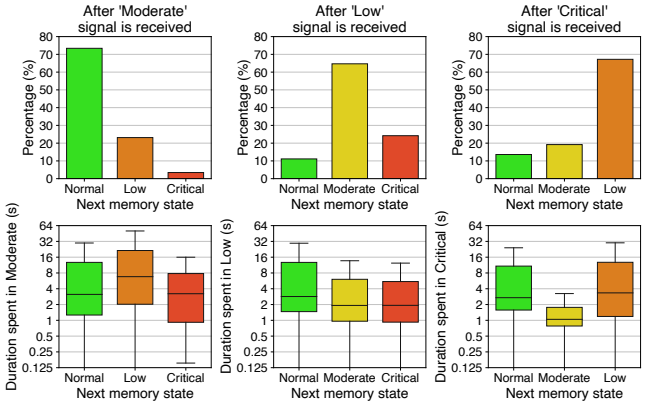

Figure 6: Statistics regarding transitions into different memory pressure states after a particular memory pressure state signal is received.

state signal is received, the devices move to another high memory pressure state, Low, 67.2% of the times. Moreover, before moving to Low, they remain in the Critical state for a significant duration of 12.8 s (at the 75% percentile). Only a small fraction, 13.6% of times, do the devices move to the Normal memory pressure state after the Critical state. This too, however, come after the devices have spent a large duration of time, 10.8 s at the 75% percentile, in the Critical memory pressure state. These results imply that existing kernel mechanisms are not sufficient to alleviate memory pressure. This motivates the need for applications to themselves act to ameliorate their performance under memory pressure, and to improve kernel memory management mechanisms.

Summary. Overall, our study of real smartphone users shows that devices often operate under high memory utilization regimes, frequently observe memory pressure signals, and the kernel is unable to quickly alleviate memory pressure. These insights coupled with the popularity of video streaming on mobile devices—as also evidenced from our user survey—makes it important to quantify and analyze the impact of memory pressure on video streaming performance.

4 IMPACT OF MEMORY PRESSURE

In this section, we evaluate the impact of memory pressure on mobile video streaming QoE using controlled experiments. To quantify video performance, we measure (video) client-level metrics (e.g., frame drop rate and client crash rate) as well as report ratings derived from the user survey. To obtain user ratings—which capture the actual user experience—we conducted an IRB approved study of 99 participants, who rated the quality of their experience of watching videos under different memory pressure states.

4.1 Experimental Methodology

Experimental Setup. We use the dash.js framework [1] to conduct video streaming experiments similar to several earlier works (e.g., [23, 24, 32]). We use five different videos in our experimental evaluation, each of which represent different genres. The videos were encoded by the H.264/MPEG-4 codec at different video resolutions (240p to 1440p), frame rates (30 and 60 frames per second), and bit rates (as recommended by YouTube [20]). The videos are divided

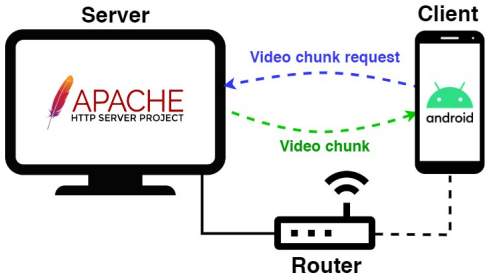

Figure 7: The experimental setup consists of a mobile client which connects to the video server over a WiFi LAN to stream a DASH video.

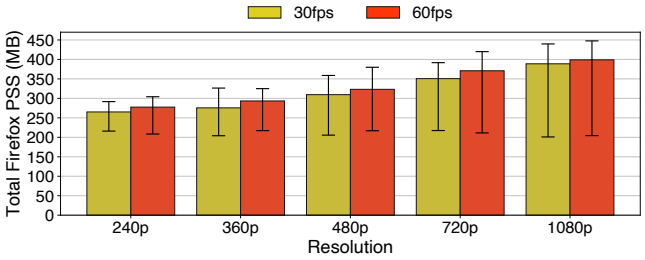

Figure 8: The total PSS of the video client process on Nexus 5 across different resolutions and two encoded frame rates of 30 and 60 FPS with under no memory pressure.

into approximately 4 second chunks similar to earlier works [24, 32]. In our setup, as shown in Figure 7, the client video player was a Firefox browser (running on a mobile device). Apart from Firefox, we also conducted evaluation over the Chrome mobile browser, and ExoPlayer—a popular media player used by Android applications [10], and reported the results in Appendix B. The video server (Apache version 2.4.7) ran on a different machine and communicated with the video player over WiFi on a dedicated LAN. The video player was configured to have a playback buffer capacity of 60 seconds. We ensured that the network never became a bottleneck in the video streaming, *i.e.*, for all video qualities, the playback buffer filled up quickly and then remained at maximum capacity throughout the playback duration. This allowed us to study the impact of memory pressure on the video performance—without the network bandwidth becoming a confounding variable. We repeated each experiment five times and report the mean results with 95% confidence intervals, unless stated otherwise. For single video experiments, we report results for the video, '*Dubai Flow Motion in 4K – A Rob Whitworth Film*" [8] from YouTube. At the time of writing, the video had received over 4.4 million views on YouTube. We show video streaming performance results for all videos in Figure 12.

Mobile devices. We conducted our evaluation on three smartphones with different RAM sizes: (1) Nokia 1, a popular entry-level phone with 1 GB RAM (Quad-core 1.1 GHz, 4.5in screen), (2) Nexus 5, which has 2 GB RAM (Quad-core 2.33 GHz, 4.95in screen), and (3) Nexus 6P with 3 GB RAM (Octa-core 4×1.55 GHz and 4×2.0 GHz, 5.7in screen). The devices run the vendor's Android OS distribution. The specifications of these devices represent a wide range of smartphones and are categorized as *low-end* and *middle-end* smartphones based on their RAM sizes [33]. According to a study of one of the largest online social networks, in 2019 low-end (0.5-1 GB RAM) and middle-end (1.5-3 GB RAM) smartphones made up 50% of devices in developed regions (North America, Western Europe, Australia, Japan, and New Zealand), 75% in transitioning regions (Eastern and Central Europe, and some Asian countries), and over 75% in developing regions (primarily African, Latin American, and Asian countries) [33].

Metrics and user survey. To measure the memory used by the video client, we record the Proportional Set Size (PSS) of the application through *dumpsys* [9]. PSS is composed of the private memory of that process plus the proportion of shared memory with other processes and is used by Android to measure an application's memory footprint [22]. To assess the rendering quality at the video client, we measure the rendered Frames Per Second (FPS) and the corresponding frame drops. If the video client suffers from slow rendering, it is forced to skip frames to maintain 1× rate (displaying 1 second of video per second), resulting in frame drops. When this happens, the user will perceive stuttering in the video. To measure the impact of memory pressure and corresponding frame drops on overall user-perceived QoE, we conduct a user survey with 99 participants and report differential mean opinion scores (details in §4.3). Finally, because video clients can crash under memory pressure, we also report video client crash rates.

Emulating different memory pressure regimes. We introduce memory pressure following the same methodology as in [22, 34], i.e., through a custom (native) Android application. For this purpose we use a publicly available application developed in a recent work [34]. This application allocates memory until a target memory pressure regime is achieved. For example, to induce critical memory pressure, it continues to allocate memory until it starts receiving Critical memory pressure signals from the kernel. To conduct controlled video streaming experiments at different memory pressure states, we start the video streaming session after the targeted memory pressure signal is received and do not keep any background apps. In addition to these controlled experiments, we also conduct a subset of experiments by exerting memory pressure organically by opening multiple background applications and verifying our findings from the controlled settings.

4.2 Memory footprint

We first analyze the memory footprint of the video client and evaluate how it changes when the video is encoded at different resolutions and frame rates on the 2 GB RAM device (Nexus 5). Figure 8 shows the PSS of a video at different resolutions and two encoded frame rates with no applied memory pressure or opened background applications.[9] Observe that the PSS increases significantly with both video resolutions as well as the encoded frame rate. For example, the mean PSS increases by nearly 125 MB when moving from 240p to 1080p, which corresponds to a average increase of ~31.3 MB when moving to the next higher resolution. Similarly, increasing the frame rate from 30 FPS to 60 FPS resulted in a mean increase in PSS of 20 MB across video resolutions from 240p to 1080p.

[9]The error bars in Figure 8 represent minimum and maximum PSS values.

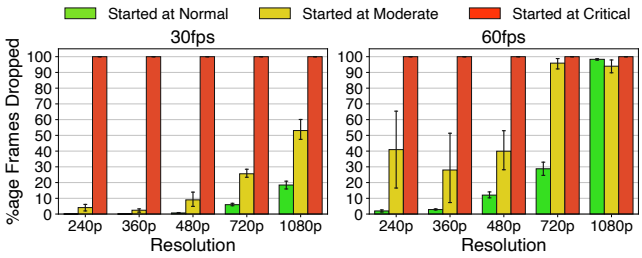

Figure 9: The average frame drops for different video qualities and two frame rates on a Nokia 1 smartphone.

Crash rate	30FPS, 480p	30FPS, 720p	60FPS, 480p	60FPS, 720p
Normal (%)	0	0	0	0
Moderate (%)	40	100	40	100
Critical (%)	100	100	100	100

Table 2: Video client crash rate across different memory pressure states on a Nokia 1 device.

4.3 Video streaming under memory pressure

Next, we quantify the impact of memory pressure on video performance across smartphones with different RAM sizes. To do so, we start video streaming experiments in three different memory pressure states: (1) *Normal* state, when no memory pressure signals are received from the kernel, (2) *Moderate* state, when the moderate memory pressure signal is received from the kernel, and (3) *Critical* state, when the critical memory pressure signal is received from the kernel.

Video performance over an entry-level smartphone. Figure 9 shows the percentage of frames dropped for video runs on the 1 GB RAM device (Nokia 1). According to a recent study [33], smartphones with 1 GB or less RAM had a market share of nearly 30% in developing regions in 2019 [33]. We note the following key trends:

- *Frame drop rate increases with memory pressure.* There is generally a significant increase in the percentage of frames dropped with increasing memory pressure. For example, when streaming a 1080p video at 30 FPS, we observed a drop rate of 19% at Normal, 53% at Moderate, and nearly 100% at the Critical memory pressure state. Under Critical memory pressure, in most cases the video was either unplayable or the video client crashed.
- *Clients experience significant crashes at high memory pressures states.* Table 2 shows the percentage of runs that experienced crashes. We observe that at Moderate and Critical states, the video client experienced frequent crashes. We also tested the video performance on another identical Nokia 1 phone with the same specs and observed the same performance, which suggests that the smartphone was not malfunctioning.
- *Frame drop rate generally increases at higher resolutions and higher frame rates.* Under memory pressure, the video client experiences more frame drops at higher video resolutions. For example, compared to 720p, streaming a 1080p video at 30 FPS increases the drop rate by nearly 100% at Moderate memory pressure. Similarly, the video client experiences more frame drops at 60 FPS. For instance, as compared to 720p video at 30 FPS, a 60 FPS video experiences 70% more frame drops at Moderate memory pressure.

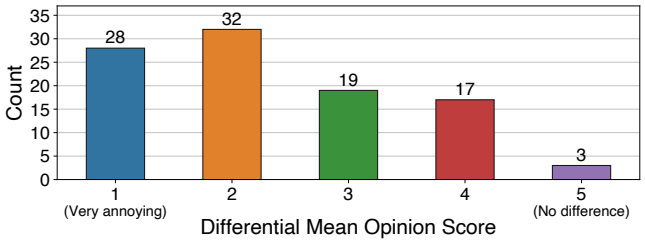

Figure 10: The differential mean opinion scores from our user survey with 99 participants. The users were asked to rate the relative experience of watching two videos, one under Normal state and the other under Moderate memory pressure.

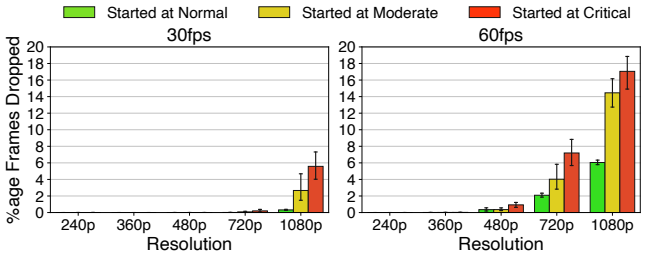

Figure 11: The average frame drops for different video qualities and two frame rates on a Nexus 5 smartphone.

Crash rate	30FPS, 720p	30FPS, 1080p	60FPS, 480p	60FPS, 720p
Normal (%)	0	0	0	0
Moderate (%)	10	100	0	100
Critical (%)	100	100	70	100

Table 3: Video client crash rate across different memory pressure states on the Nexus 5 (2 GB RAM) device.

Impact on overall user experience. To quantify the impact of frame drops on actual user experience, we conducted a user study involving 99 participants. Users were recruited from a university campus via email and included students and staff. Users were shown two samples of our video, each encoded at 60 FPS and 240p, but experiencing different memory pressure states—*Normal* and *Moderate*—resulting in a frame drop rate of 3% and 35%, respectively. Users were asked to rate their relative experiences of watching the two videos on a scale of 1-5 (with 5 denoting no noticeable difference between the two videos and 1 denoting that the second video was very annoying with respect to the first). Figure 10 shows the frequency distribution of differential mean opinion scores (DMOS), which indicates that the vast majority of users found a *noticeable* difference between the videos, with 60 users giving a 1 or 2 rating.

Impact on mobile devices with larger RAM. Figure 11 shows the frame drop rate on the 2 GB RAM Nexus 5 smartphone. Overall, we observe lower yet significant frame drops relative to the 1 GB RAM smartphone. In particular, with a 30 FPS video and low video resolutions (240p to 480p), we do not observe any frame drops. However, at 60 FPS with high resolutions, the phone experiences significant drops, e.g., 17% drops on average with the 1080p video under critical memory pressure. Nexus 5 also experiences crashes at high memory pressure as shown in Table 3. We observe similar

Process State	Normal (s)	Moderate (s)	Increase (%)
Running	69.0	63.2	-8.5
Runnable	58.2	72.4	24.2
Runnable (Preempted)	13.3	26.4	97.8

Table 4: The mean time spent by video client process threads in different process states at Normal and Moderate memory pressure states.

	Normal	Moderate	Increase
Mean number of preemptions	378.3	10457.3	26.6 x
Mean time *mmcqd* runs after preemption	0.1 s	1.3 s	16.8 x
Mean time video client waits to get CPU back	0.2 s	5.4 s	27.5 x

Table 5: Mean statistics for when the kernel preempts the CPU from some video client process thread to schedule *mmcqd* under Normal and Moderate memory pressure states.

trend of frame drops on Nexus 6P (a 3 GB RAM device); (i) frame drops happen at only 720p or higher resolutions, (ii) highest frame drops (∼9% on average) occur at 1080p resolution with a frame rate of 60 FPS.

Performance over different videos. We also evaluated the performance under memory pressure over different types of videos. In particular, we considered five different videos from separate genres; travel [8], sports [16], gaming [15], news [4], and nature [3]. Figure 12 shows the percentage of frames dropped when viewing these five videos under different memory pressure states. The figure shows the results from a 2 GB Nexus device, at both 30 FPS and 60 FPS, with different video resolutions. We observe a similar trend for all videos; (i) frame drops at 30 FPS are either low or negligible, and (ii) frames drops are significant at 60 FPS and increase with memory pressure and at higher video resolutions.

Performance under organic memory pressure. As explained in §4.1, we emulate different memory pressure regimes in our experiments through a custom Android application that allocates memory until a target memory pressure regime is reached. To evaluate whether video performs similarly under memory pressure present in the wild, we evaluated video performance under organically generated memory pressure through background applications. We conducted our experiments on the Nokia 1 device. For Normal memory pressure state, we did not open any background application whereas for inducing Moderate memory pressure state we opened 8 background applications before opening the browser to run the video. These background applications were selected from the top free applications available on Google Play Store and did not include any game. Similar to the synthetic memory pressure experiments, we observe higher frame drops at higher memory pressure states. For example, with 480p 60 FPS video, we observed that 11.7% frames dropped at Normal memory pressure, while 30.6% frames dropped at Moderate memory pressure. These frame drops are similar to what we had observed under memory pressure applied through the custom application.

5 WHY VIDEO QOE DEGRADES?

In this section, we analyze why video performance degrades under memory pressure. To do so, we conduct a system-level trace analysis through Google's system profiling tool, Perfetto [19]. We report profiling results on a 1 GB RAM device (Nokia 1), whose Video QoE is most impacted by memory pressure.[10]

Impact on video client processes. Table 4 shows the total time spent by video client threads[11] in different states under Moderate pressure relative to the Normal pressure state. These states include: (1) *Running*, when a thread is running on one of the CPU cores, (2) *Runnable*, when a thread is waiting to be scheduled but the CPU is unavailable, (3) *Runnable (Preempted)*, when a video thread is waiting for the CPU *after* having been preempted by the kernel to schedule a higher priority process. We find that under memory pressure, video threads spent 8.5% less time running and spend significantly more time waiting for the CPU (e.g., 24.2% more in the *Runnable* state, and 97.8% more time in the *Runnable (Preempted)* state) as shown in Table 4. Next, we examine which processes are responsible for increase in the waiting times of video threads, and how much, if at all, are these delays contributed by memory management daemons.

Top running threads. To identify processes that contribute to the waiting time experienced by the video client threads, we examine the top running threads, in terms of the total duration a thread ran during playback, under Normal and Moderate memory pressure. We find there are two threads whose running time increases significantly under Moderate memory pressure—the daemon that manages queued I/O operations on the storage, *mmcqd*, and the kernel swap daemon, *kswapd* (both introduced in §2). For *mmcqd*, we observe that its running time increases from 0.4 s to 4.6 s and it becomes the 6th most running thread on the mobile device (from being the 50th most running thread under Normal). For *kswapd* we observe that it runs for 2.3 s in Normal memory pressure and is the 14th most running thread on the device. In Moderate memory pressure however, it runs for 22 s becoming the *most* run thread on the CPU. For comparison, the second-most run thread (a Firefox thread) runs for only 7.9 s. This substantive increase is also reflected in Figure 13 which shows the percentage of time spent by *kswapd* in different process states under Normal and Moderate memory pressure. Observe that it spends considerably less time sleeping (75% to 31%), and substantially more time running (6% to 56%) under Moderate memory pressure.

Preemption events. Next, we analyze the events where the kernel preempts a video client thread in favor of some other higher priority thread. Such events are important as they can lead to frame drops and degrade user experience. For each such higher priority thread, we measure: (1) the total number of preemptions, (2) the total time it runs continuously on the CPU after the preemption, and (3) the total time spent by client video threads waiting to use the CPU again. We find that compared to Normal, under Moderate pressure *mmcqd* shows a large increase in each of these three statistics. As shown in Table 5, we observe that under Moderate memory pressure, the number of preemption events involving *mmcqd* increase by a factor of 26.6×, and the total time video threads have to wait to get CPU back increases by a factor of 27.5×. With these substantial increases, *mmcqd* becomes the thread with the highest values for each of these three statistics on the device. This is because, as discussed in §2, disk I/O increases under memory pressure, resulting in *mmcqd*

[10]The subsequent results in this section represent video runs conducted at Normal and Moderate memory pressure for a 480p video at 60 FPS. We conducted three runs for each memory state and report the mean values.

[11]We consider three key video client threads: *SurfaceFlinger*, *MediaCodec*, and *Firefox*.

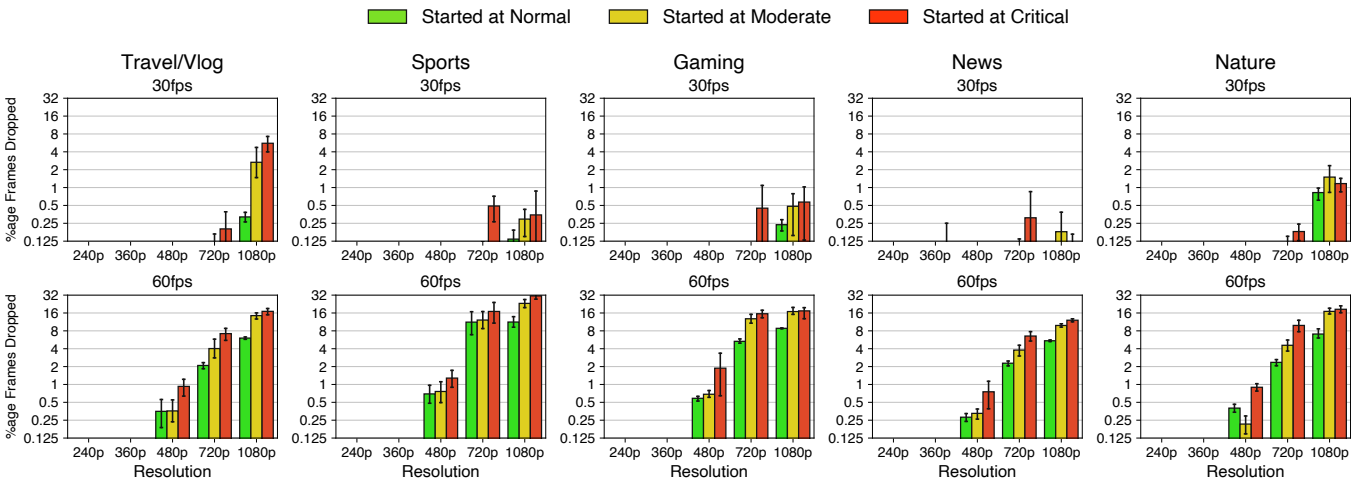

Figure 12: Rendering performance across videos with different genres across different memory pressure states for both 30 FPS and 60 FPS videos.

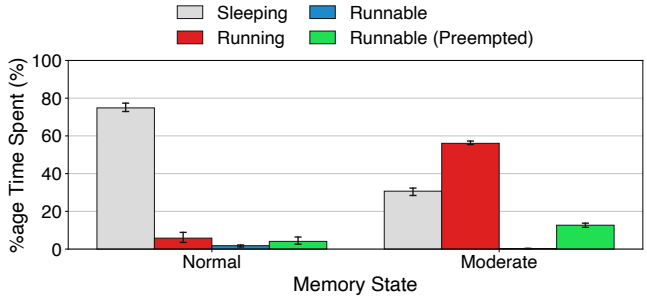

Figure 13: Percentage time spent by the kernel daemon, *kswapd*, in different process states under Normal and Moderate memory pressure states.

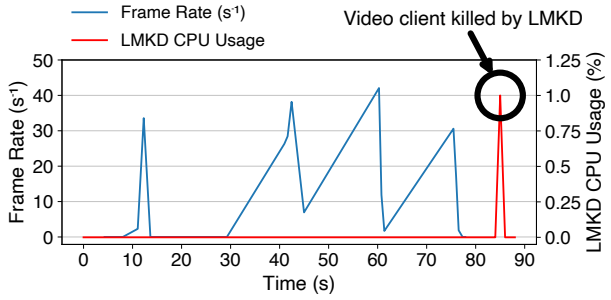

Figure 14: Frame rate and *lmkd* CPU utilization during a video session which crashed due to high memory pressure.

stealing CPU time from foreground processes as it has a higher CPU scheduling priority.

On the other hand, in both Normal and Moderate memory states, the CPU is almost never preempted for *kswapd*. This is because while *mmcqd* has a strictly higher scheduling priority than foreground processes, *kswapd* has the same priority as foreground processes. For instance, we find that 77.9% of Firefox threads have the same priority as *kswapd*. Such threads account for 76.7% of all Firefox threads' running time. This means that for most of the time, Firefox threads will have to fairly share the CPU with the CPU-hungry thread—*kswapd*.

Video client crashes. Another key memory management daemon (described in §2) is *lmkd*, which is responsible for killing processes to free memory. To identify when it becomes active, we tracked its CPU utilization during video playback under Normal and Moderate memory pressure.[12] We find that it almost never becomes active under Normal memory pressure, but frequently becomes active under Moderate pressure often resulting in the browser, or the browser tab process, that is playing the video, to get killed. As an example, see Figure 14. It shows the instantaneous frame rate and the CPU utilization over one such video session under Moderate memory pressure. Notice that when we observe the video crashing at 85 s, there is a spike in *lmkd*'s CPU utilization indicating that it became active to kill the video client browser.

Performance degradation with background applications. Figure 15 shows the instantaneous rendered FPS and the number of processes killed during a video session on Nokia 1 under organic Normal and Moderate memory pressure (using the same setup as in §4.3).[13] The x-axis shows the time elapsed since the enter key was pressed on the mobile device to load the webpage that contained the video player. Observe that there were many more processes that had to be killed during Moderate memory pressure than during Normal memory pressure.

Summary. We find that the video processes drop frames primarily due to two reasons: (i) interference by *mmcqd*, the kernel thread that manages queued I/O operations on storage, which steals CPU

[12]We tracked the CPU utilization through the Linux `top` command. Video and device configuration remained the same as in our profiling results.

[13]We tracked the number of killed processes through Android's command-line tool `logcat` [12].

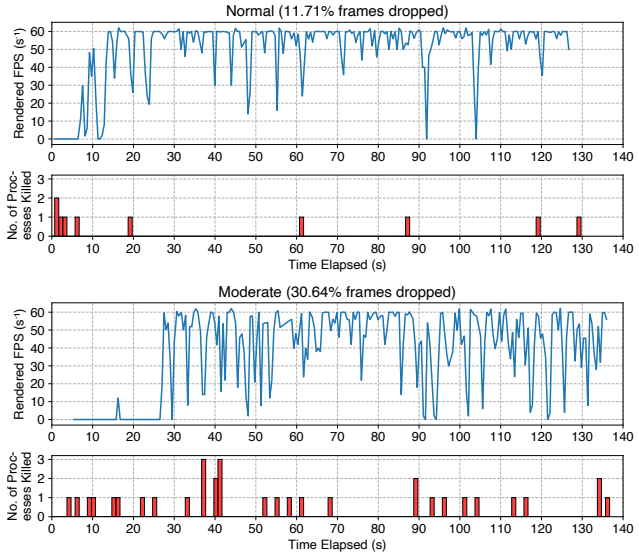

Figure 15: Rendered FPS and the number of processes killed over a video run under Normal (top two plots) and Moderate memory pressure states (bottom two plots). These memory pressure states are generated organically by opening background applications.

cycles from video threads as it has a higher scheduling priority than foreground processes, and (ii) high CPU utilization of the *kswapd* under high memory pressure, which in turn decreases available CPU cycles for video threads. Additionally, we observe that video client processes crash when *lmkd* kills them to free memory.

6 OPPORTUNITIES FOR IMPROVEMENT

In this section, we highlight potential opportunities for improving video performance by using application level mechanisms and memory pressure signals.

Application-level mechanisms to reduce frame-losses: There are significant opportunities for reducing memory pressure and improving video performance by employing application-level mechanisms (e.g., adapting the video resolution or the encoded frame rate in response to memory pressure signals). Figure 16 shows the impact of varying the encoded frame rate during a video session at three different resolutions. We observe that a video can continue to be rendered at high resolution by decreasing the encoded frame rate. For example, at 1080p resolution (top sub-graph), the rendered FPS is zero when the frames are encoded at 60 FPS, however, the frame losses reduce to about zero when we switch to 24 FPS. However, across resolutions, there are different encoded frame rates that may be best suited for a specific video under a given device memory state. Deciding on which combination of resolution and encoded frame rate will yield the best performance may also depend on the specific type of video and needs more comprehensive investigation. We leave this for future work.

Using memory pressure signals to dynamically adapt video: We show that memory pressure signals received by applications can be used as a potential signal for adapting the video (e.g., switching to different encoded frame rate or resolution). Figure 17 shows the

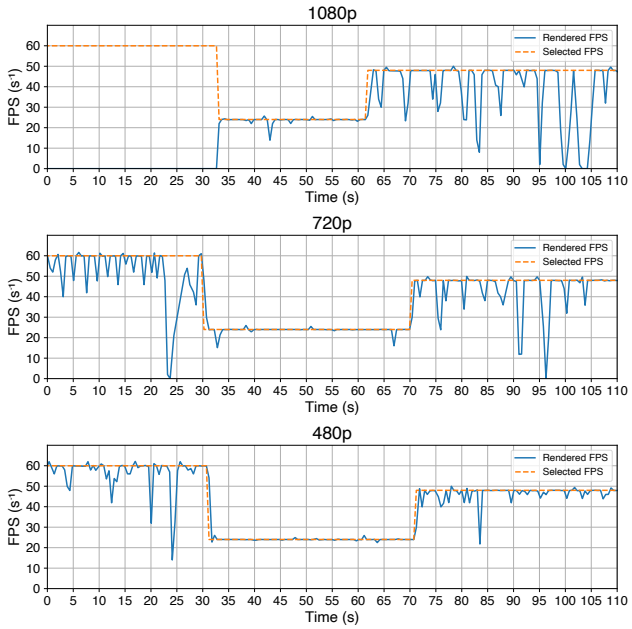

Figure 16: The impact of varying frame rate across different video resolutions on a Nokia 1 device. For each video resolution, the encoded frame rate was varied between 24, 48, and 60 FPS.

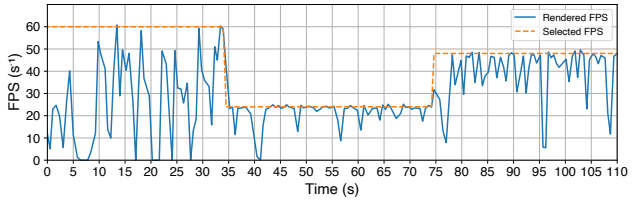

Figure 17: The impact of varying the video frame rate under Moderate memory pressure on a Nokia 1 device. The video is encoded at 480p and the encoded frame rate varies between 60, 24, and 48 FPS.

rendered FPS in a video run under Moderate memory pressure if we vary the frame-rate between 60, 24 and 48. The memory pressure was introduced organically through background applications as done before in §4.3. Observe that there are significant drops in FPS when 60 FPS is selected. However, we can mitigate these losses if decrease the frame rate to 24 FPS.

7 DISCUSSION

The results and insights have important implications for different stakeholders in the Internet video streaming ecosystem. We discuss below the broader implications.

Implications for Internet Video Providers. Keeping in view the potential impact of device-level bottlenecks, our work has two important implications for Internet Video providers. First, providers should measure device memory conditions as it has a role to play in determining client-side QoE. This additional visibility into device memory pressure can help better disambiguate the complexities associated with troubleshooting client performance issues in the wild [26]. To achieve this, providers can augment their routinely

collected client side telemetry (e.g., average network throughput) by capturing memory pressure signals as well. Second, Internet Video providers should consider offering a larger range of video encodings to adapt not only video resolutions but also the framerate. Most providers today construct bitrate ladders to enable smooth playback over variable network conditions and do not consider device limitations [28, 29]. As evidenced by our work, this can lead to sub-optimal performance especially for low-end and medium-end smartphones. As such, video providers should consider offering resolutions that vary over a range of frame rates. Low-end devices can then select lower frame rate streams thereby reducing the memory and computational footprint to achieve improved playback. For example, we have observed that modulating the video *frame rate* based on memory pressure signals can improve video performance.

Video Adaptation Algorithms. Given that device resources, such as memory, can become potential bottlenecks for video streaming performance, bitrate adaptation algorithms—which traditionally have focused on network bottlenecks [24, 32, 35] and coarse-grained device-level measures (e.g., capping the maximum video resolution based on device screen resolution [21])—should explicitly account for multiple bottleneck resources (e.g., memory, CPU, and network) in their design. These enhanced algorithms can then be rolled out with relative ease through *application-level mechanisms* such as regular over-the-air application updates to achieve performance improvements for memory constrained devices.

Video client implementations. Video players can run atop browsers or as custom mobile applications. Consequently, the memory pressure induced by a video client is bundled with the memory overhead of the underlying implementation platform (e.g., browser or a mobile app). While our experimental evaluation was carried out over the mobile Firefox browser, we find that Chrome and a video mobile app (implemented over ExoPlayer, which is an application-level media player for Android [10]) induces a lower memory overhead. A lower memory overhead does help in improving performance but is insufficient to prevent frame drops and application crashes; see Appendix B for evaluation results for a video mobile app and Chrome. Moreover, video client-level mechanisms that we explored in this work are complementary to optimizations in the browser and mobile app implementations.

OS developers. Mobile operating systems, such as Android, employ system daemons (e.g., *lmkd*, *kswapd*, and *mmcqd*) to make available more free memory. However, we show that they can frequently *interfere* with video processes especially under high memory pressure. Moreover, our study suggests that there is scope for reducing this interference with improved scheduling of system daemons and video processes. For example, we observe that *kswapd* frequently switches cores, however, if the allocation of cores is coordinated between daemons and video processes, reduced context switching overhead can potentially lead to improved performance.

Original Equipment Manufacturers. Our work highlights the importance of having sufficient CPU resources under high memory pressure regimes. For entry-level phones, this indicates that allocating more CPU resources even with a small RAM can improve video performance under memory pressure. This insight is complementary to an earlier study focusing on Web browsing under memory pressure [34].

8 RELATED WORK

Measurement studies on low-end mobile devices. Recent studies [25, 33, 34] have investigated the impact of low-end devices on mobile QoE.

Dasari et al. [25] study the impact of device hardware capabilities on the performance of web browsing, video streaming, and video telephony. They find that web browsing is more sensitive to low-end hardware than video streaming. This is because unlike web browsing, video applications can exploit specialized co-processors/accelerators and thread-level parallelism on the multiple cores present on low-end devices. This analysis, however, is for performance under normal memory conditions. Our study on the other hand investigates video streaming performance under memory pressure and finds that memory pressure can significantly affect video streaming in low-end devices due to interference with kernel-level daemons.

Nasser et al. [33] present a large-scale longitudinal study of Android device RAM resources and show that, contrary to intuition, the memory gap between devices in developed and developing regions is not diminishing but widening. Consequently, they try to optimize web browsing performance on low-end devices through a system that trims page content in a way that balances page functionality against memory usage. Note that, similar to Dasari et al. [25], they do not consider memory pressure. Our study uses the longitudinal study for building motivation, but instead of web browsing, focuses on video streaming performance under memory pressure.

Qazi et al. [34] perform a study similar to ours, but in the context of Web browsing. They analyze the memory usage of mobile web browsers and find that the browser memory footprint can be orders of magnitude larger than the corresponding Web page sizes. This can lead to slow browsing performance and/or Web page crashes when a device is running low on memory. They propose debloating JS and using memory-efficient image formats to reduce the memory footprint of Web pages and improve Web performance. Our work builds on this study and we use the application developed by it to apply memory pressure on mobile devices. However, our work differs from it in three different ways. First, our study is the first to our knowledge that shows how frequently mobile devices, based on users' natural usage pattern, experience memory pressure. Secondly, instead of Web browsing performance, we quantify the impact of memory pressure on video streaming performance. Third, we provide an in-depth system analysis of how kernel-level mechanisms for managing memory pressure interact with video streaming process to degrade performance.

Video performance and bottlenecks. There is a large body of work, such as [24, 27, 32, 35–37], on adapting video quality given network bottlenecks. Our work is complementary to these studies as it highlights the device memory as a potential bottleneck for video streaming. Client-side video bottlenecks have been investigated to some extent by prior measurement studies albeit without consideration for memory pressure [25, 26]. Ghasemi et al. [26] investigated dropped frames and download stack latency in the case of laptop and desktop machines. Dasari et al. [25] analyze the impact of device parameters on different Internet applications but do not consider high memory pressure regimes. Our work focuses

on memory pressure on mobile devices and provides insights on its impact on video performance.

9 CONCLUSION

Given the growing demands for high quality videos for mobile video streaming, this work sheds light on the memory consumption patterns of mobile devices, quantifies the impact of memory pressure on video performance, and identifies root causes of performance degradation. This work also highlights opportunities for improving video performance through better adaption by the video clients and better OS level memory management and scheduling schemes. This work has implications for different stakeholders in the smartphone ecosystem, including video streaming platforms, OS developers, and Original Equipment Manufacturers.

ACKNOWLEDGEMENTS

We thank Abdul Manan for his help in developing experimental setups used in the paper, and Muhammad Shahpar Nafees Khan for help with performing early stage experiments. We are also grateful to our shepherd, Aruna Balasubramanian, and the anonymous reviewers for their comments and feedback.

REFERENCES

[1] Akamai. 2016. dash.js. https://github.com/Dash-Industry-Forum/dash.js/.
[2] Android: Low RAM Configuration. https://source.android.com/devices/tech/perf/low-ram.
[3] Bali in 8k ULTRA HD HDR - Paradise of Asia (60 FPS). https://youtu.be/fajeL728XG8.
[4] Clarissa Ward presses Taliban fighter on treatment of women. https://youtu.be/RIw7smlkIaU.
[5] ComponentCallbacks2,. https://tinyurl.com/97kc9pu5.
[6] Devices used to watch online video worldwide as of August 2019. https://www.statista.com/statistics/784351/online-video-devices/.
[7] Distribution of worldwide YouTube viewing time as of 2nd quarter 2021, by device. https://www.statista.com/statistics/1173543/youtube-viewing-time-share-device/.
[8] Dubai Flow Motion in 4K – A Rob Whitworth Film. https://youtu.be/BLL-kW_TpT4.
[9] Dumpsys. https://developer.android.com/studio/command-line/dumpsys.
[10] ExoPlayer. https://developer.android.com/guide/topics/media/exoplayer.
[11] ExoPlayer: Flexible media playback for Android (Google I/O '17). https://youtu.be/jAZn-J1I8Eg.
[12] Logcat command-line tool. https://developer.android.com/studio/command-line/logcat.
[13] Memory allocation among processes. https://developer.android.com/topic/performance/memory-management.
[14] Mobile Operating System Market Share Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide.
[15] NIGMA vs OG - TI CHAMPIONS GAME - DPC EU DREAMLEAGUE S14 DOTA 2. https://youtu.be/Ek-gfQo6ryE.
[16] Novak Djokovic vs Denis Shapovalov (4K 60FPS) MATCH HIGHLIGHTS Court Level View 2021 ATP CUP. https://youtu.be/lnoba3DZQZw.
[17] Overview of memory management. https://developer.android.com/topic/performance/memory-overview.
[18] Package visibility filtering on Android. https://developer.android.com/training/package-visibility.
[19] Perfetto. https://perfetto.dev/.
[20] Recommended YouTube Upload Encode Settings. https://support.google.com/youtube/answer/1722171?hl=en#zippy=%2Cbitrate.
[21] Supported media formats. https://developer.android.com/guide/topics/media/media-formats.
[22] Understanding Android Memory Usage (Google 1/O'18). https://tinyurl.com/33yk98s7.
[23] Z. Akhtar, Y. Li, R. Govindan, E. Halepovic, S. Hao, Y. Liu, and S. Sen. Avic: A cache for adaptive bitrate video. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, page 305–317, New York, NY, USA, 2019. Association for Computing Machinery.
[24] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang. Oboe: Auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 44–58, New York, NY, USA, 2018. Association for Computing Machinery.
[25] M. Dasari, S. Vargas, A. Bhattacharya, A. Balasubramanian, S. R. Das, and M. Ferdman. Impact of device performance on mobile internet qoe. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, pages 1–7, New York, NY, USA, 2018. ACM.
[26] M. Ghasemi, P. Kanuparthy, A. Mansy, T. Benson, and J. Rexford. Performance characterization of a commercial video streaming service. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, 2016.
[27] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, 2014.
[28] A. V. Katsenou, J. Sole, and D. R. Bull. Content-gnostic bitrate ladder prediction for adaptive video streaming. In *2019 Picture Coding Symposium (PCS)*, 2019.
[29] A. V. Katsenou, J. Sole, and D. R. Bull. Efficient bitrate ladder construction for content-optimized adaptive video streaming. 2021.
[30] Y. Liang, J. Li, R. Ausavarungnirun, R. Pan, L. Shi, T.-W. Kuo, and C. J. Xue. Acclaim: Adaptive memory reclaim to improve user experience in android systems. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 897–910. USENIX Association, July 2020.
[31] Y. Liang, Q. Li, and C. J. Xue. Mismatched memory management of android smartphones. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, Renton, WA, July 2019. USENIX Association.
[32] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, 2017.
[33] U. Naseer, T. A. Benson, and R. Netravali. Webmedic: Disentangling the memory-functionality tension for the next billion mobile web users. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, HotMobile '21, page 71–77, New York, NY, USA, 2021. Association for Computing Machinery.
[34] I. A. Qazi, Z. A. Qazi, T. A. Benson, G. Murtaza, E. Latif, A. Manan, and A. Tariq. Mobile web browsing under memory pressure. *SIGCOMM Comput. Commun. Rev.*, 50(4):35–48, Oct. 2020.
[35] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016.
[36] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 272–285, New York, NY, USA, 2016. ACM.
[37] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, 2015.

A APPENDIX: CODE AND DATA

We have created a public repository[14] to reproduce experimental results reported in this work and share the data we collected through our user studies. To replicate all experimental results, we provide our testbed setups, and explain how to use them. In addition, for each experiment we also provide collected logs/traces and scripts used to parse, analyse, and plot graphs presented in this work. For the user study discussed in §3, we open-source the Android application used to collect the results, share the data collected, and provide code used to parse, analyze and plot graphs. For the user study discussed in §4.3, we share the videos shown to users, the exact questions they were asked, their responses, and the code used to parse and plot the responses.

The repository comprises of seven folders. `setup_video` contains the setup used for streaming video on mobile device browsers while `setup_mem-pressure` contains the setup used to apply memory pressure to mobile devices. These setups are used for experiments across multiple sections of the paper. The other five folders correspond to the experiments/user studies discussed in §3, §4.2,

[14] Available at https://github.com/nsgLUMS/mobileVideo and https://doi.org/10.6084/m9.figshare.21340758

§4.3, §5 and §6 respectively. We provide a description for each folder's content below. If additional help is required, please reach out to any of the authors.

A.1 Video Setup

The folder `setup_video` contains the testbed setup for streaming DASH video on a mobile phone browser. We use a modified version of Pensieve's video setup [32] which enables us to log frame drop statistics. We provide an overview of the setup and share instructions for installation and use. We also provide scripts we used to convert videos into DASH-compatible format along with their usage instructions.

A.2 Memory Pressure

The `setup_mem-pressure` folder contains two applications used to apply/control memory pressure on mobile devices. The first is a modified version of `MP Simulator`—an Android app developed in a recent work [34] that synthetically applies memory pressure on an Android device. [15] The second is a Node.js web application that can be used to remotely interact with `MP Simulator` using both an interactive Web UI and REST API calls. Usage instructions are provided for both applications in their respective sub-directories.

A.3 Experiments and User Studies

Survey of memory usage in real-world devices (§3). The folder `sec3*` contains the source code of `SignalCapturer`, the application used to survey memory consumption patterns in Android devices. It also contains a link to download all logs collected by `SignalCapturer`, an IPython notebook to parse, clean and analyze them, and plot the figures present in the section.

Memory footprint (§4.2). The folder `sec4.2*` contains an automation script that uses the video and memory pressure setups to run video on Nexus 5 under multiple memory pressure states and periodically logs browser processes' memory footprint. The logs are collected using the `dumpsys meminfo` [9] command. Instructions to use the script are also provided. The folder also contains the logs for our experiments and an IPython notebook to parse and plot them.

Video streaming under memory pressure (§4.3). The folder `sec4.3*` contains code and data pertaining to (i) the video streaming experiments under memory pressure, and (ii) the user study to quantify the impact of frame drops on user's QoE. To reproduce the video streaming experiments, we provide code and data similar to how we provided it for `sec4.2*`. For the user study, we share the videos shown to the users, explain how these videos were produced, share users' responses on the questions asked, and provide the code used to plot the results.

Analysis of why performance degrades under memory pressure (§5). The folder `sec5*` contains three sub-folders. The `initial_assessment` folder contains instructions, scripts, and results for reproducing an initial assessment we conducted through the Linux `top` command (this was used to produce Figure 14). The `trace_analysis` folder contains code and data pertaining to the system-level trace analysis conducted through Perfetto [19]. The folder includes instructions on how to record traces on Perfetto using the same configurations we used. It further includes the traces we recorded, scripts to parse information from them, and IPython notebooks to analyze them and plot graphs. The `bg_apps` folder includes code, data, and instructions for reproducing Figure 15.

Oppurtunities for improvement (§6). The folder `sec6*` contains code and data pertaining to experimental results discussed in §6, and instructions on how to reproduce them.

[15] This app only works on rooted Android devices.

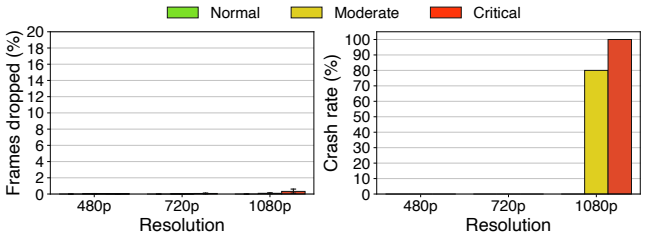

Figure 18: Video drops and crash rate with ExoPlayer on a 2 GB Nexus 5 device.

B PERFORMANCE ACROSS DIFFERENT VIDEO PLAYERS

Our video experiments are mainly on one client video player—Firefox. Here we share results from repeating the experiments on a native Android application video player called *ExoPlayer* [10] and on another mobile browser, Chrome.

B.1 Performance on a native application

To understand how video streaming fares under memory pressure on native Android applications, e.g. YouTube, we conduct our experiments on ExoPlayer [10].[16] Figure 18 shows frame drops and crash rates for ExoPlayer. We observe that under memory pressure, contrary to Firefox, ExoPlayer drops a significantly smaller number of video frames. However, like Firefox, ExoPlayer also suffers from significant crashes under high memory pressure. We conjecture that the decrease in frame drops can be partly attributed to the lower memory footprint of ExoPlayer, which has a smaller footprint then Firefox.

B.2 Performance on Google Chrome

To measure the performance of video streaming under memory pressure on a different browser, we selected Chrome. Figure 19 shows frame drops and crash rates on Chrome. We observe that under memory pressure, contrary to Firefox, Chrome drops a smaller number of video frames. However, like Firefox, Chrome also suffers from significant crashes under high memory pressure. We conjecture that the decrease in frame drops can be partly attributed to the lower memory footprint of Chrome, which has a smaller footprint than Firefox. We conjecture that the reason Chrome performs better than Firefox is that it is more memory efficient (as shown by Qazi et al. [34]).

[16] ExoPlayer is an open-source media player developed by Google that can be used in a native Android application to play audio/video both locally and over the Internet. ExoPlayer is widely adopted in the industry: as of 2017, over 140,000 Android apps used ExoPlayer including YouTube, Netflix, Facebook, Spotify, etc. [11].

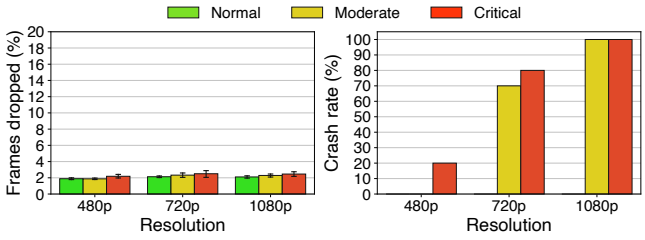

Figure 19: Video drops and crash rate with Google Chrome on a 2 GB Nexus 5 device.