Understanding Mobile Video Streaming Performance(Youtube) in Low-End Devices In a Reasonably Large Measurement Study

Ahmad Faraz Khan
LUMS

Haider Ali
LUMS

ABSTRACT

The main aim of the study is to better understand the mobile video streaming performance of one of the most used video streaming services like youtube in low end devices, with that this study also analyzes how the performance varies across multiple factors that can affect the Quality of Experience (QoE) for users. We analyze the QoE stats such as re-buffering time, startup latency and dropped frames etc. in various controlled experiments. The factors that we use in our experimentation include memory, CPU cores and clock frequency etc. to measure the changes in performance. Our results demonstrate, for most parameters if we deteriorate these device level parameters the QoE falls at a high rate. Related research provides analysis on specific factors and very few studies involve low-end devices let alone a large scale study, this study will add a lot of ground to the understanding of a popular streaming service(youtube) and will also promote further research on low-end devices with the hope that after understanding the contributing factors of performance work can be done to mitigate the effect of those factors to provide better quality of service to low-end device users.

KEYWORDS

bitrate, dropped frames, startup latency, rebuffering

ACM Reference Format:
Ahmad Faraz Khan and Haider Ali. 2019. Understanding Mobile Video Streaming Performance(Youtube) in Low-End Devices In a Reasonably Large Measurement Study. In , .

1 INTRODUCTION

Mobile devices are becoming increasingly popular in developing countries. These mobile devices also include a large portion of smartphone devices. A pilot study conducted by [6] in a developing region illustrated that in some villages near Maradi (Niger) how animations have the potential to be educational and entertainment for people with limited access to information and knowledge which in turn could create new entrepreneurial opportunities for farmers. Youtube in this regard can do much more especially for promoting free E-education in these developing countries.

There exists a plethora of low-end devices especially in developing countries. Low CPU, memory and other factors in low-end devices cause the quality of experience for a user of this device to deteriorate and can become an annoyance. Due to this very reason billion dollar companies are investing resources to bring applications such as Google's youtube Go and Android Go OS to these developing countries for low-end devices. These devices most of the time do not provide the same quality of experience to users as a high end device would for the same service. To fill this gap we need to understand in detail the parameters that contribute to this prevailing gap. For building this understanding a reasonably large scale study is required. The most rapidly growing internet traffic is video streaming traffic. Content providers, ISPs and advertising agencies all want to provide the best video streaming service to their users to attract as much traffic as possible. The global video streaming market size is anticipated to reach 124.57 Billion dollars by 2025, according to a new report by Grand View Research, Inc. [4] This study can inform all the stakeholders (e.g., content providers and Internet service providers) to improve the user experience of mobile video streaming. Due to the popularity of youtube (second most visited website worldwide [3]) it is used all across the world and even in developing countries such as Pakistan, Bangladesh and India. By understanding the factors that contribute in the QoE for users with low-end devices youtube can target these markets and attract more users to its pool.

Video streaming service is a topic of interest for many researchers [2, 10, 14]. However, unlike prior works done on video streaming on mobile devices [12] very few works have considered research on low-end devices and even then the devices considered were not very low-end with four or more cores and one or more than one GB RAM[6]. Our focus is mainly on a device that can be considered low-end from all parameters.

Our main findings were:

- We observed that rebuffering in single core was a lot more than on dual-core in chrome browser due to the application stopping when single core could not handle the video streaming application. The same trend followed in youtube app as well but not as extreme as in the case of chrome. In app we saw an increase in 5X rebuffering rate and also an increase in median startup latencies by almost 6 seconds. The dropped frames also increased by almost 50%
- On different bandwidths in wifi 500MB and 100MB upper limits we saw a median increase of 1.3 and 28 seconds in startup latencies and rebuffering rate respectively. In cellular networks at different bandwidths(Ufone 3G 111 KBps and 44 KBps) median total dropped frames was 23X higher, rebuffering rate was 3X higher and startup latency was 3 seconds more in the lower bandwidth connection. While comparing 3G with wifi at equal bandwidth we found 9X higher median total dropped frames in wifi, however, the rebuffering in wifi was 2X lower compared to 3G.
- While applying a memory pressure of 150 MB on the device the rebuffering increased more than 4X, startup latency increased by 3X and total dropped frames increased by 2X.

, ,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

- Rebuffering We also saw rebuffering time for chrome is 0.1 seconds and for application it is 85 seconds which is very much large.
- During the comparison of youtube go with youtube app we found that median memory used by youtube Go is 58 MB and youtube application is 77 MB, median total dropped frames for youtube Go is 653 and for youtube app, it is 897. While median CPU utilization for youtube Go is 13.3% and for youtube application, it is 32.4%.
- Almost 34% increase was observed in the case when the resolution was raised from 144p to 480p.
- We also performed experiments in dual core mode by decreasing the clock frequency from 1.3 GHz to 598 MHz of both cores in userspace mode, but we found that none of our QoE factors or device parameters were affected much by this change.

2 PREVIOUS DATA COLLECTION TECHNIQUES

Previously work has been done on collection of youtube metadata as well as its analysis. Some approaches that have been previously used include developing tools from scratch, [14] have developed a tool called TubeKit that collects 17 attributes using PHP and MagpieRSS etc. for crawling, however their tool collected attributes that were not very much related to the video QoE or the ABR algorithm as their attributes included Title, description, Text comments etc. A similar approach of crawling was used by [2] for extracting statistics such as ID, video length, average rating etc. [12] have used a tool called YoMoApp that collects passive QoE related data such as initial delay and stalling etc. from youtube videos, one of the drawbacks of this tool is that it does not show QoE statistics such as dropped frames and it does not work on low end devices which we are experimenting on. [9] have developed a Wrapper App that captures the youtube stats for nerds stats with the aid of and bridge(ADB).[13] looks at the possibility of using MITM(Man in the Middle) based approaches to collect QoE data, but it suggests this technique from the ISPs perspective which is not possible as it causes privacy issues but can be used at the client side when there are no privacy issues. [11] uses the techniques of asking users about the quality of experience that they experience, this technique however, is not suitable for controlled experiments and the results may vary from user to user. [10] has used mitmproxy to capture traffic that the client sends to the youtube stats server, they use this technique to measure the stalls between events. [15] collected data with the help of the content distributor(Yahoo).

3 MEASUREMENT SETUP

Youtube has a feature called stats for nerds that reports various QoE statistics such as the video format, bandwidth and dropped frames etc. Since our experimentation involved low end phones we wanted to use an approach that would cause the least amount of load on the device, for this we used Mitmproxy that was set up on a Linux OS operating on a laptop machine. The laptop acted as a proxy and all the traffic from the mobile device passed through the mitmproxy server that was set up on the laptop machine. With Mitmproxy we were able to capture the requests sent by the client mobile device to the youtube server, these requests also included qoe stats reported by the device which were being used by the ABR algorithm at the youtube server for serving the appropriate quality of the video. We used a filter to separate these QoE requests and store them in files from which we later extracted information through self-made tools. During experimentation the mobile device and the laptop were connected tothe same network. We collected traces on 3G networks from Ufone (A Pakistan based cellular network provider) and also on a router the bandwidth of which was tune-able for performing controlled experiments. For collecting traces on the cellular network we used a phone with 3G Ufone sim inserted and we connected it to the cellular network, after which we created a hotspot on that device and connected both our laptop and our low-end device to that hotspot, then we set our laptop's IP as the proxy address on our low-end device. The average network speed from the router was 4.2 Mbps but could be tuned by setting an upper limit on the maximum bandwidth. We also collected device level parameters synchronized with the network traces while the video was playing. For the device level parameters we took advantage of the Android Debug Bridge(ADB) from which we extracted the CPU usage and memory(PSS) of the application on which the video was streaming. We also collected our data on both the android browser and the youtube application. Each section will further describe the method in detail that we used for each experiment. In total we collected traces of of over 18 hours of youtube videos from which after cleaning our data from the anomalies we got approximately 16 hours of data traces. The device we used was a Huawei Y511 with a 512 MB RAM, Android Jellybean OS, 1.3GHz dual-core processor. We were able to root the device to control the number of cores, memory and clock frequency of each core to perform controlled experiments. There were only 4 types of resolutions provided by youtube to our device that included 144p, 240p, 360p and 480p corresponding to the bitrate formats given by youtube in our statistics of 160, 133, 134 and 135 respectively.

4 CHALLENGES

One challenge we faced was the syncing of CPU and memory extraction with the network statistic extraction. If during extraction of data the CPU and memory extraction is not synchronized which happened whenever we tried to calculate the CPU and memory through ADB at a very fine granularity of say 1 second it would slow down the device and ADB would consume a lot of CPU because of this we either got readings after more than one second or we would get more readings of memory than CPU which indicated that they are not being extracted at the same granularity even after setting the ADB flag of duration (-d) to 1. To overcome this we extracted CPU and memory at 2 or more than 2 seconds of granularity. Each data dumped by mitmproxy had a different structure for different browser types as well as the application. For this we had to make a generic tool that could extract data from any type of mitmdump files that had the QoE stats. Another challenge that we faced was that due to the device being low end most of the time the client device stopped sending the stats requests, for that we generated an event after every 30 second which was the granularity of the stat requests originally. After every event generation the client device transmitted the stat for that time and we were able to capture

it. We also tried to collect data through selenium but it slowed down our low-end device to an extent that the stats of a particular instant reached us after a very long time and we could not be sure which time instance do these readings represent. For this we chose Mitmproxy which does not pose a lot of load on the low-end device. The data we were collecting from the browser only reported buffer health, current video time, timestamp(starts when the video is played) and the bitrate format for this we could only infer statistics such as stall ratio, startup-latency and buffer health, we used the app data for analysis of other QoE stats which provided almost all the required statistics. We have assumed that the FPS remains constant as it is not provided by youtube QoE requests. However, from youtube-dl we did find that 135 and 160 bitrate format corresponds to 25fps.

5 OBSERVATIONS

These were some of the things we observed on the collected data, in the following sub-sections we will discuss the measurement setup in detail for each experiment and the results obtained with possible reasoning.

5.1 Comparing single core performance with dual-core

5.1.1 ***Rebuffering in single core and dual-core on chrome browser***. Comparing single core performance with dual-core It has been found that the video applications exploit specialized co-processors/accelerators and thread-level parallelism on multi-core mobile devices giving a better performance on multicore devices compared to single core devices as illustrated by Ahmad et al [13]. For this experiment we collected traces by turning off and on a core of our dual core devices and collected traces of the same video in both cases. We also kept the internet speed to a 100 KBps for both and used the same device. In between we also cleared the cache of the application so that each time the video is downloaded again. We performed this experiment on the chrome while the android governor was on userspace mode. In first case both our cores were working at 1.3 GHz in userspace mode, when we turned off a core only one was working at 1.3 GHz of frequency. The CDF in Fig1 shows our results. The median re-buffering in seconds for dual core came out to be close to 0.18 seconds whereas for a single core it came out to be 124.5 seconds. This would happen as the video would often get stuck and the process would stop responding in single core mode.

5.1.2 ***Rebuffering and starup latency on youtube app in single core and dual-core***. We also collected the same data over youtube app by keeping the video and other parameters constant. The network conditions were also kept same for both single and dual core, however, in this we did not limit the bandwidth to 100 KBps. In that too we observed an increase in rebuffering rate in single core compared to dual-core setup. The median rebuffeing in seconds for dual-core lies at 5 seconds, whereas in single core it is about 28 seconds as shown by the CDF in Fig2. This is a 5X increase in rebuffering after turning off a core. On the same data we also calculated the startup latencies on the same data and found that for dual core the median latencies came out to be 4 seconds compared to 10 seconds in single cores as shown in Fig3.

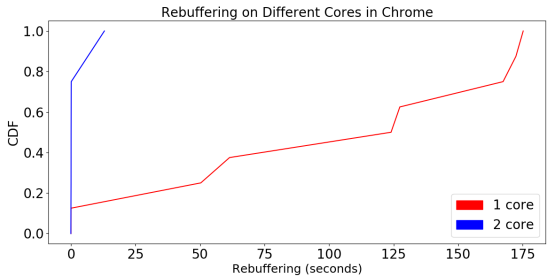

Figure 1: Difference in re-buffering on chrome browser in seconds on single and dual core

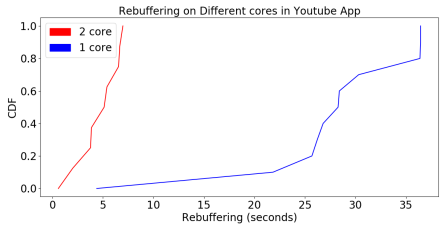

Figure 2: shows rebuffering in youtube app with single and dual core setups

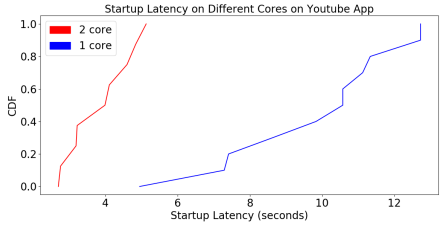

Figure 3: shows startup latency in youtube app with single and dual core setups

5.1.3 ***Dropped frame comparison on youtube app in single core and dual-core***. With increase in rebuffering and startup latency we will also expect an increase in the dropped frames in a single core. Frames are dropped due to rendering path problems [15] which in our case are caused by low-end device parameters such as a single core. To observe this phenomenon we also compared the dropped frames. For this we kept the network conditions, android governor(userspace), video bitrate format(135) and memory conditions same, we also cleared the cache after each trace was collected. We found the median total dropped frames in single core to be 1461 compared to 942 in dual-core setup as shown in Fig 4 almost a 64% increase in median dropped frames. The CPU usage of the application also increases in single core because in dual core parallelism can be utilized, however, in single core the burden of work falls on a single processor stressing it out as can be observed in the CDF in Fig 5 which shows the CPU usage by the process in a single core and dual core setup.

This goes to show that low-end single core devices that cannot utilize parallelism for video streaming suffer with a lot of rebuffering rate and increased startup latencies with more dropped frames reducing the quality of experience for the user.

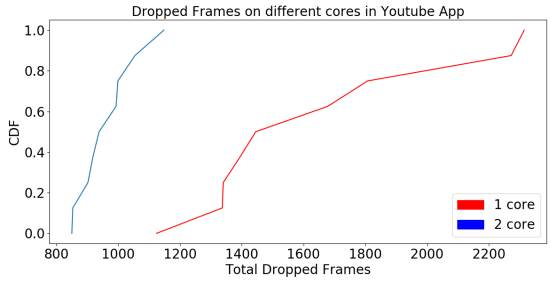

Figure 4: depicts the total dropped frames in youtube app for each core number

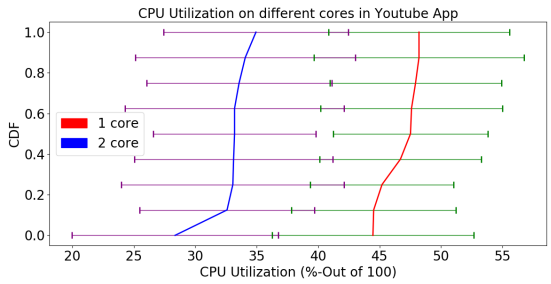

Figure 5: illustrates the difference between the CPU utilization of youtube app in different number of cores

5.2 Analysis on different Networks

We have done the following network comparisons with logical reasoning on youtube application.

5.2.1 ***Different bandwidths comparison on WiFi:*** To just test the general trend that Does changing the bandwidth on WiFi affect startup latency and rebuffering time? We have collected the two traces by constraining the bandwidth through router. The Figure 6 and 7 answer the above mentioned question. The expected result is that high bandwidth network should perform well which is true through our results. This analysis is mentioned just because it is a large scale study. The median startup latency for high bandwidth is 5 seconds and for low bandwidth it is 6.3 seconds as shown in Figure 7. The median rebuffering time for high bandwidth is 85 seconds and for low bandwidth, it is 113 seconds.

5.2.2 ***Different bandwidth comparison on 3G:.*** We collected two types of traces on 3G network, in one case we kept the average bandwidth of our network close to 111 KBps and in the other one we kept the average bandwidth close to 44KBps. We then compared the three QoE metrics that are startup latency, rebuffering and dropped frames between both these traces as illustrated in Figure 9, Figure 8 and Figure 10 respectively. In Figure9, the median startup latency for high bandwidth is 5 seconds and for low bandwidth, it is 3 seconds

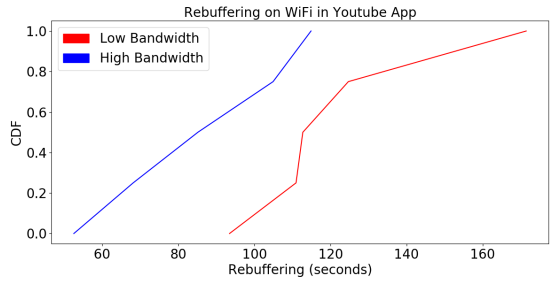

Figure 6: shows the difference in rebuffering in seconds for WiFi at different bandwidths

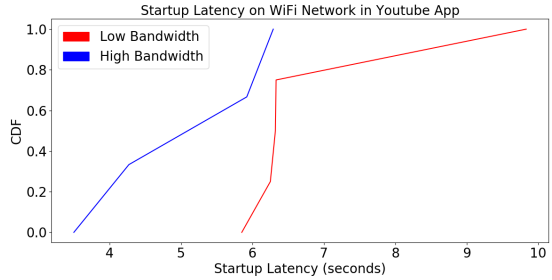

Figure 7: shows the difference in startup latency in seconds for WiFi at different bandwidths

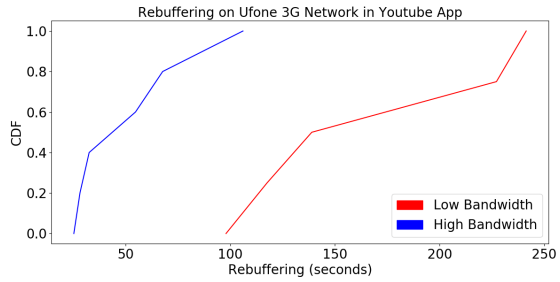

Figure 8: shows the difference in rebuffering in seconds for cellular networks(Ufone 3G) at different bandwidths

higher that is 8 seconds. In Figure 8, the median rebuffering time for high bandwidth is 43.5 seconds and for low bandwidth, it is 3X higher that is 139 seconds.The results for high startup latencies and rebuffering time on low bandwidth were expected but the result we got for dropped frames was not expected. Ideally, dropped frames should not depend on bandwidth but Figure 4 shows that for high bandwidth, median dropped frames are 94 and for low bandwidth, median dropped frames are 2205 which is 23X higher. [8] states that when there is low bandwidth, there will be less time for decoding and rendering the frame which will lead to high dropped frames. The same is the case in our analysis. [8] found a threshold download rate above which there will be very low dropped frames. Finding threshold is left for future work in our case.

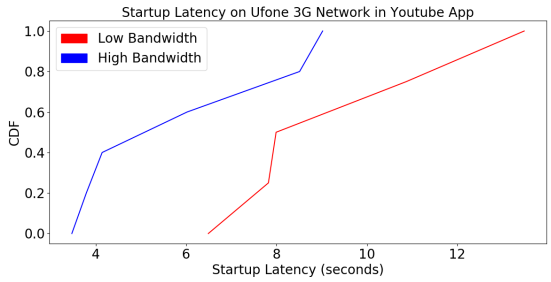

Figure 9: shows the difference in startup latency in seconds for cellular networks(Ufone 3G) at different bandwidths

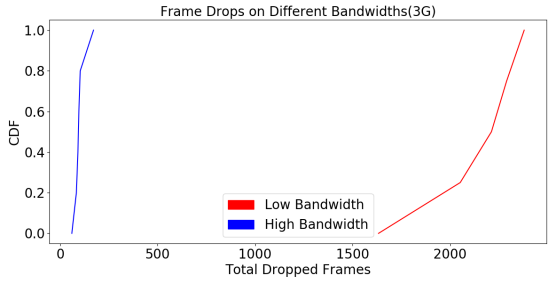

Figure 10: shows the difference in Dropped Frames for cellular networks(Ufone 3G) at different bandwidths

5.2.3 ***Cellular Network (3G) and WiFi comparison:*** We collected the data on two networks of 3G and WiFi having comparable bandwidths. The average bandwidth of 3G was around 111 KBps and average bandwidth for WiFi was around 113 KBps with relatively similar variability. But there are huge differences in their protocols which was the major reason that we got different results. One reason of difference is also that we collected 3G data under mobility. The CDFs of startup latency, rebuffering time and dropped frames are given in Figure 12, Figure 11 and Figure 13 respectively.

The median startup latency of WiFi is 4 seconds and for 3G, it is 5.1 seconds. This difference is low but 80% of startup latencies in Figure 12 show a marked difference between the two. The median rebuffering time for WiFi is 21 seconds and for 3G, it is 44 seconds which is 2X higher. The reason that startup latencies and rebuffering times for 3G are higher than WiFi is that there is frequent packet loss in 3G under mobility and there are more handovers [5]. More handovers mean more connection loss therefore there will be more rebuffering due to handovers as compared to packet loss in our case. We compared 3G data under mobility with WiFi because there is a marked difference between the two categories and it has the potential to give different and interesting results. While maintaining the similar bandwidths in the two networks, we have been able to see potential differences between the two. The comparison between 3G data under no mobility and WiFi is left for future work.

As shown in Figure 13, the median total dropped frames for 3G is 94 and for WiFi, it is 896 which is 9X higher. It has been observed that variability in WiFi in our data is more than 3G which could be

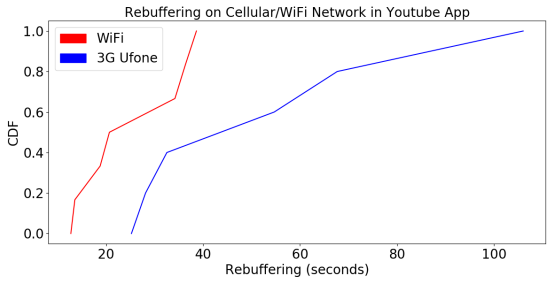

Figure 11: shows the difference in rebuffering in seconds for cellular networks(Ufone 3G) and WiFi

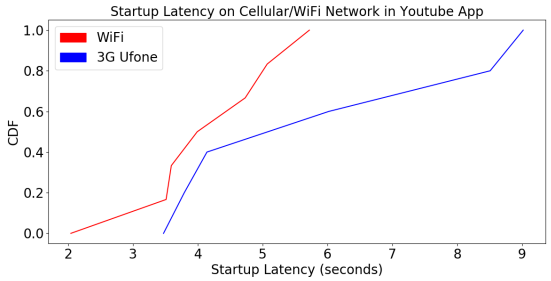

Figure 12: shows the difference in startup latencies in seconds for cellular networks(Ufone 3G) and WiFi

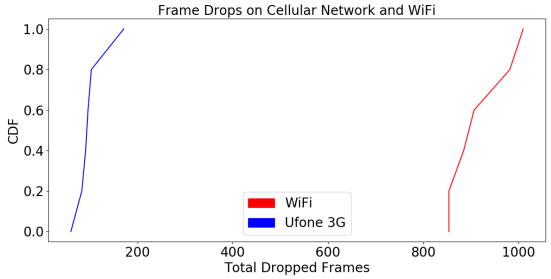

Figure 13: shows the difference in dropped frames for cellular networks(Ufone 3G) and WiFi

the possible reason for frame drops as there can be more packet loss in WiFi due to variability. [15] also states that a common response of a video decoder to dropped IP packets is to momentarily freeze the video by repeating the last good video frame. In our case, there can also be the case that dropped packet caused frame repetition and can be considered as a dropped frame. Other possible reason could be the one mentioned above in [8] that low download rate cause frames to drop. It is because relatively higher variability in WiFi network can decrease download rate as compared to 3G.

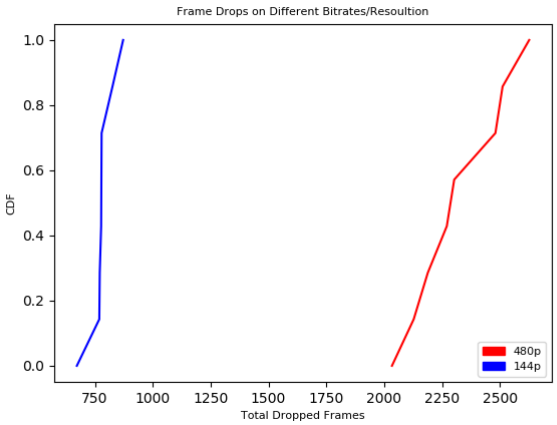

Figure 14: shows the difference in total frame drops between different resolutions 480p and 144p (135 and 160 bitrate formats respectively)

5.3 Affect on the QoE upon changing video resolution in correspondence with device factors

Popular video streaming services like youtube provide videos at different bitrates as the ABR algorithm adjusts to provide better QoE. The bitrate changes at chunk level granularity. In our case we observe only the number of dropped frames for the same video at different bitrates. Some other QoE indexes such as smoothness can also be observed for better understanding of QoE at different bitrate formats, however, it this metric is difficult to calculate in low-end devices with the aid of mitmproxy as we are getting traces at a granularity of 30 seconds, so we leave this for future work in this field. The measurement setup for this included fixing the number of cores at upper limit of the RAM at 512 MB and then collecting the data over the same network. The only thing we varied was the bitrate format at 135 and 160 respectively. Here the bitrate format 135 corresponds to 480p resolution and the 160 bitrate format corresponds to 144p resolution.

The 135 bitrate format means more number of bits being rendered by the application and if the application is unable to render these high number of bits it may choose to drop frames to maintain the framerate which we assume remains constant. Our data shows that the median number of total dropped frames for 135 bitrate fromat are 2289 whereas for 160 we get a median number of 777 total dropped frames as shown in Fig14. Almost 34% increase can be observed in the case of 135 bitrate format. Our results contradict those of [8] because we performed a controlled experiment while keeping the bitrate constant instead of [8] which experimented on Yahoo in auto mode.

5.4 Constraining memory and observing its affects on video streaming performance

For this we kept the bitrate format constant at 135 and other things like android governor (userspace), number of cores, video and clock

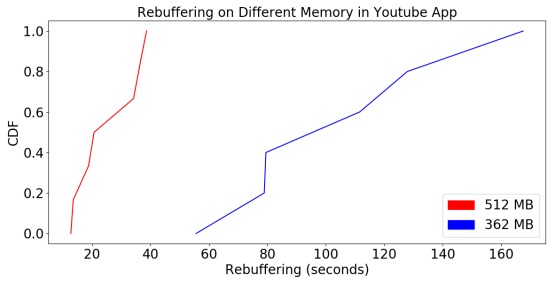

Figure 15: shows the CDF of rebuffering in seconds at no memory pressure and after applying 150MB of memory pressure

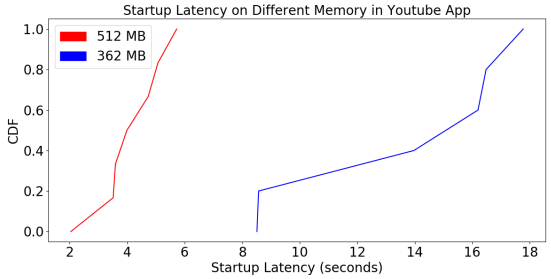

Figure 16: shows the CDF of startup latency in seconds at no memory pressure and after applying 150MB of memory pressure

frequency was also kept constant. We only variated the memory by creating the memory pressure. We collected two types of traces, one in which no memory pressure was created and upto 512 MB of memory was available to the process while in the other case we created a memory pressure of 150 MB.

5.4.1 ***Variations in rebuffering after applying memory pressure.*** The device we have used(Huawei Y511) has a memory of 512MB as advertised by the company. This is a memory of a low end phone but for simulating the memory conditions of even lower end phones we created a memory pressure of 150 MB. For this we had rooted the phone and we used a memory pressure simulator application called MP simulator that was publicly available (citation). Our focus was on checking the variations in QoE stats(rebuffering, startup latency, dropped frames). Fig15 shows a CDF that illustrates the rebuffering changes after and before applying memory pressure, when no memory pressure is applied the median rebuffering is at 21 seconds while after applying pressure it becomes 96 seconds on median. That is an approximate increase of 4X. This goes to show the impact memory has on the QoE on low-end devices.

5.4.2 ***Variations in startup latency after applying memory pressure.*** We also observed changes in startup latency in seconds for both the cases described above as shown in Fig16. The median for the case in which no memory pressure was applied came out

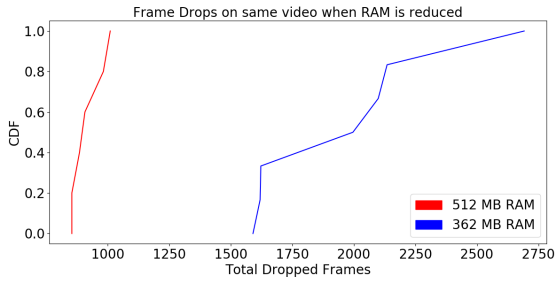

Figure 17: shows the total dropped frames after and before applying memory pressure of 150MB on the same video

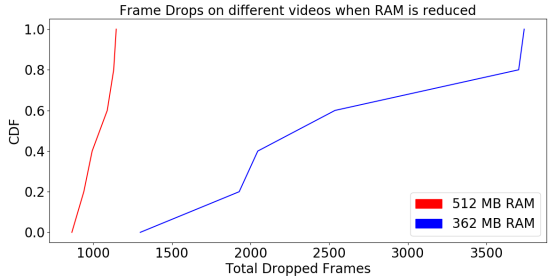

Figure 18: shows the total dropped frames after and before applying memory pressure of 150MB on different videos

to be 4 seconds which was more than 3X less than the 15 seconds observed after applying a pressure of 150MB.

5.4.3 ***Variations in total dropped frames after applying memory pressure.*** From the above calculated data we also inferred the dropped frames in each case, the median dropped frames for no pressure were 900 as shown by the CDF in Fig17 whereas it was 2005 for a memory pressure of 150 MB. The probable reason for this behavior is that due to restricted memory for the application, it is unable to maintain a buffer and for maintaining a constant bitrate and FPS the application is forced to drop frames.

We also calculated another type of data in which we changed our video each time to observe that whether this trend follows all type of videos. Even for different videos we observed the same trend and almost an equal ratio between both cases of memory pressure and no pressure, our median 2296 and 1050 respectively. 2X increase in dropped frames. The results are also visible in the CDF shown in Fig18.

5.5 Comparison between Youtube Application and Chrome

We have collected separate traces with bandwidth of same network and observed a marked difference in rebuffering time between youtube application and Chrome browser. The possible reason could be that youtube application is designed for high end phones and therefore there is more rebuffering in youtube application. The CDF is shown in Figure 19. The median rebuffering time for chrome is 0.1 seconds and for application, it is 85 seconds which is

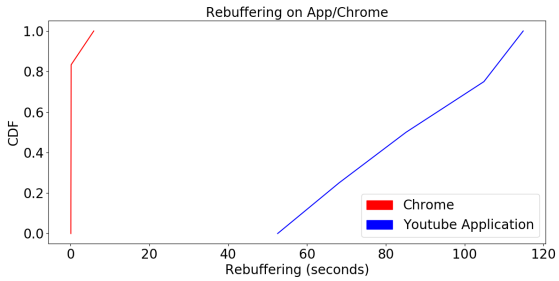

Figure 19: shows the difference in rebuffering time between youtube application and chrome

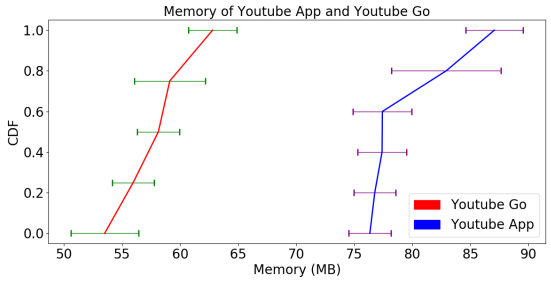

Figure 20: shows the difference in memory between youtube application and youtube Go

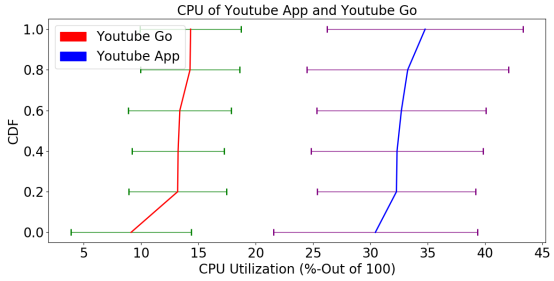

Figure 21: shows the difference in cpu between youtube application and youtube Go

very much large as compared to no rebuffering in chrome.Youtube Application should be optimized to work in low end mobile phones. For this purpose, we have tested another product of Youtube that is specially designed for low end phones which is Youtube Go.

5.6 Comparison between Youtube Application and Youtube Go

We have collected two separate traces for youtube application and youtube Go in a controlled environment. We tried to maintain a controlled environment as we fixed resolution of youtube application to 480p. But youtube Go does not allow to control resolution much. To cater this challenge, we used good bandwidth network

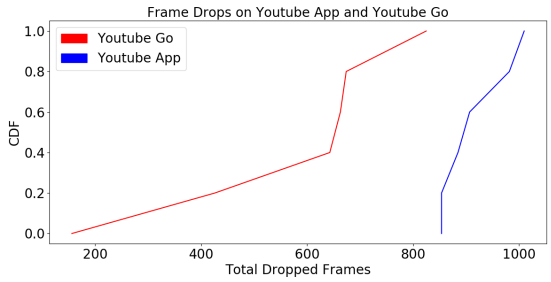

Figure 22: shows the difference in dropped frames between youtube application and youtube Go

so that resolution of youtube Go could be maintained at 480p resolution. After collecting data from youtube Go, we discarded low resolution traces and used 360p and 480p traces for comparison against youtube application. This setup is much controlled and can be used for comparison.

We used three device level metrics for comparison: CPU utilization(%), PSS Memory(MB) and Dropped Frames.

Figure 20,21 and 22 shows the memory, cpu and dropped frames CDFs on youtube application and youtube Go. Median memory used by youtube Go is 58 MB and youtube application is 77 MB. Median dropped frames for youtube Go is 653 and for youtube app, it is 897. Median CPU utilization for youtube Go is 13.3% and for youtube application, it is 32.4%. The possible reason that youtube Go is working well for low end phones is that it changes resolution more aggressively with proper device level optimizations. Youtube Go uses smaller size files as shown by CDF of memory too.

6 DISCUSSION

So far we have only considered a comparison of CPU, memory and dropped frames in the youtube go app with the youtube app. However, there are various parameters that can be considered for further study on youtube go application such as considering other QoE factors like rebuffering, startup latency, smoothness etc. This comparison study will help understand the improvements that have been made and the areas where improvements can still take place. With that we can also compare the youtube go's performance to other video streaming services such as Netflix. This will also help us understand the differences in ABR algorithms of both the services and further analysis on how these ABR algorithms adapt in network and device conditions can be done. More research is required to find how good ABR algorithms in adjusting to device parameters like CPU cores and memory, especially to cater to low-end devices.

We also tried a few measurements on Netflix using selenium to gather data by crawling the html after stats were displayed on the screen using the Netflix provided shortcuts, however, this technique could not be used in phones as Netflix does not work on phone browsers and can only work through app. On laptop we did observe some trends that showed an aggressive nature of the Netflix ABR algorithm.

Our approach of using Mitmproxy has a few drawbacks. We could not get several QoE parameters that are very cruicial for analysis such as frame rate that we had to assume will remain constant for certain bitrate format using the information from youtube-dl. One other drawback was that youtube reports its stats to the server after every 30 seconds which makes it hard to measure QoE stats such as smoothness and bitrate changes. Other techniques such as using Selenium were also tried but they slowed down our low-end device to an extent that our readings got delayed and could not be used for analysis.

So far we have only collected data on one of the local cellular network providers, however, there are many providers present in the market and are widely used, an interesting analysis can be performed how changing the cellular network affects a low-end device for a complete large scale study, with this we can also cater for the number of handovers, mobility and latency that might vary for each cellular network.

7 RELATED WORK

To summarize the related work, a lot of work has been done on video streaming applications considering the importance of it as most of the traffic now is video streaming traffic on the internet and is increasing every day. Some prior studies have focused on developing tools for collection of data from the video streaming services for better analysis, these include tools such as TubeKit developed by [14] that collects 17 attributes using PHP and MagpieRSS etc. that included Title, description, Text comments etc. Other works like [12] have concentrated on developing tools for calculating the QoE stat related traces such as bitrate, inintial delay and stalls etc. they have named this tool YoMoApp.Others have performed analysis on the QoE of users with respect to parameters such as distance from Cloud [11]. Video streaming research is also found on mobile devices, one study done on yahoo's video streaming service [8] considers the affect on QoE at CDN, network and client level. Very few works have considered research on low-end devices and even then the devices considered were not very low-end with four or more cores and one or more than one GB RAM like [7] and they focused only on stall ratio and startup latency. Our focus is mainly on a device that can be considered low-end from all parameters.

8 CONCLUSION

In this paper we presented a performance study of youtube while looking at QoE stats such as startup latency, rebuffering time, dropped frames etc. We used control experiments to perform measurement studies. We observed the trends in performance of youtube video streaming service across multiple platforms such as Chrome browser, android browser, youtube app and youtube go. We also observed the affect memory, CPU cores and other device level parameters that influence the low-end devices the most had on youtube streaming service. We also varied the bitrates and other QoE factors to study how it changes other parameters like dropped frames, CPU and memory. For more info, see our github repository at [1].

REFERENCES

[1] [n.d.]. https://github.com/AFKD98/Understanding-Mobile-Video-Streaming-Performance-Youtube-in-Low-End-Devices-In-a-Reasonably-Large-M
[2] Abdolreza Abhari and Mojgan Soraya. 2010. Workload generation for YouTube. *Multimedia Tools and Applications* 46, 1 (2010), 91.
[3] Anon [n.d.]. The top 500 sites on the web The sites in the top sites lists are ordered by their 1 month Alexa traffic rank.The 1 month rank is calculated using

a combination of average daily visitors and pageviews over the past month. The site with the highest combination of visitors and pageviews is ranked 1. Retrieved May 17, 2019 from https://www.alexa.com/topsites

[4] Anon. [n.d.]. Video Streaming Market Worth $124.57 Billion By 2025 | CAGR: 19.6%. Retrieved May 17, 2019 from https://www.grandviewresearch.com/press-release/global-video-streaming-market

[5] Dziugas Baltrunas, Ahmed Elmokashfi, Amund Kvalbein, and Özgü Alay. 2016. Investigating packet loss in mobile broadband networks under mobility. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 225–233.

[6] Julia Bello-Bravo and Ibrahim Baoua. 2012. Animated Videos as a Learning Tool in Developing Nations: A Pilot Study of Three Animations in Maradi and Surrounding Areas in Niger.. In *The Electronic Journal of Information Systems in Developing Countries 55, 1 (2012),*. 1–12. https://doi.org/10.1002/j.1681-4835.2012.tb00394.x

[7] Mallesham Dasari, Santiago Vargas, Arani Bhattacharya, Aruna Balasubramanian, Samir R Das, and Michael Ferdman. [n.d.]. Impact of Device Performance on Mobile Internet QoE. Retrieved May 17, 2019 from https://dl.acm.org/citation.cfm?id=3278533

[8] Mojgan Ghasemi, Partha Kanuparthy, Ahmed Mansy, Theophilus Benson, and Jennifer Rexford. 2016. Performance characterization of a commercial video streaming service. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 499–511.

[9] Theodoros Karagkioules, Dimitrios Tsilimantos, Stefan Valentin, Florian Wamser, Bernd Zeidler, Michael Seufert, Frank Loh, and Phuoc Tran-Gia. 2018. A Public Dataset for YouTube's Mobile Streaming Client. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 1–6.

[10] Vengatanathan Krishnamoorthi, Niklas Carlsson, Emir Halepovic, and Eric Petajan. 2017. BUFFEST: Predicting buffer conditions and real-time requirements of HTTP (S) adaptive streaming clients. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 76–87.

[11] Asif Ali Laghari, Hui He, Muhammad Shafiq, and Asiya Khan. 2016. Assessing effect of Cloud distance on end user's Quality of Experience (QoE). In *2016 2nd IEEE international conference on computer and communications (ICCC)*. IEEE, 500–505.

[12] Florian Wamser Pedro Casas Alessandro Dalconzo Michael Seufert, Nikolas Wehner and Phuoc Tran-Gia. 2017. Unsupervised QoE field study for mobile YouTube video streaming with YoMoApp. 2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)(2017). https://doi.org/10.1109/qomex.2017.7965688

[13] Werner Robitza, Arslan Ahmad, Peter A Kara, Luigi Atzori, Maria G Martini, Alexander Raake, and Lingfen Sun. 2017. Challenges of future multimedia QoE monitoring for internet service providers. *Multimedia Tools and Applications* 76, 21 (2017), 22243–22266.

[14] Chirag Shah. 2010. Supporting research data collection from YouTube with TubeKit. *Journal of Information Technology & Politics* 7, 2-3 (2010), 226–240.

[15] Muhammad Arslan Usman, Muhammad Rehan Usman, and Soo Young Shin. 2016. A no reference method for detection of dropped video frames in live video streaming. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 839–844.