A Mobile Video Traffic Analysis: Badly Designed Video Clients Can Waste Network Bandwidth

Hyunwoo Nam[*], Bong Ho Kim[†], Doru Calin[†] and Henning Schulzrinne[*]
[*]Department of Electrical Engineering, Columbia University, New York, NY
[†]Bell Laboratories, Alcatel-Lucent, Murray Hill, NJ

***Abstract*—Video streaming on mobile devices is on the rise. According to recent reports, mobile video streaming traffic accounted for 52.8% of total mobile data traffic in 2011, and it is forecast to reach 66.4% in 2015. We analyzed the network traffic behaviors of the two most popular HTTP-based video streaming services: YouTube and Netflix. Our research indicates that the network traffic behavior depends on factors such as the type of device, multimedia applications in use and network conditions. Furthermore, we found that a large part of the downloaded video content can be unaccepted by a video player even though it is successfully delivered to a client. This unwanted behavior often occurs when the video player changes the resolution in a fluctuating network condition and the playout buffer is full while downloading a video. Some of the measurements show that the discarded data may exceed 35% of the total video content.**

***Keywords*—Video Streaming, Mobile Wireless, HTTP Progressive Video, Over The Top Applications**

I. Introduction

Today's popular video streaming services such as YouTube and Netflix use HTTP adaptive bit-rate streaming. A video server contains several video files that encode the same video content at multiple bit-rates. The available bandwidth in the network and CPU capacity of the client's device are considered while the video player adjusts the quality of the video stream. The video files are chopped into small segments, and then streamed to the client over HTTP in order. This rate-adaption mechanism leads to byte waste. For example, when the video player changes the resolution while downloading a video, it needs to re-download the entire size of the affected segment.

Regardless of the file format and size of the video, the video server pushes the requested video content to the client as network conditions permit. The video content is buffered locally on the device and played back. Hence, if the network bandwidth available to the client is smaller than the encoded data rate of the video, the client has to wait until there is sufficient space in the buffer. Regardless of whether the client pauses or not while playing a video, some video content providers such as YouTube and Netflix continue to push the requested video content to the client. A part of the downloaded video content can be discarded without being watched if the client chooses to quit the video before it ends.

This paper focuses on HTTP adaptive bit-rate streaming; we analyze YouTube and Netflix video streaming while watching the videos on mobile devices (iOS and Android) over wireless networks (Wi-Fi, 3G and LTE) under varying network conditions. As shown in Figure 1, we have designed the Video Streaming Packet Collector (VSPC) to capture and analyze

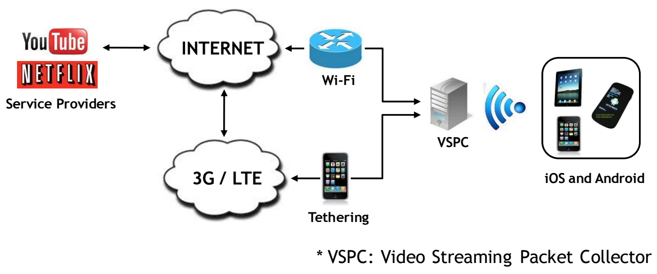


Figure 1: Mobile application traffic measurement testbed

TABLE I: iOS and Android mobile devices

Devices	OS versions	Resolutions	Memory
iPad 3	iOS 6.1.2	1920 x 1080	1024 MB
iPhone 4S	iOS 6.1.2	640 x 960	512 MB
iPhone 3G	iOS 4.1.2	320 x 480	128 MB
Nexus 7	Android 4.2.1	1280 x 800	1 GB
Nexus S 4G	Android 4.1.1	480 x 800	512 MB

TCP/IP and HTTP packets on video streaming between a client and a video server. Table I shows the hardware specifications of the selected iOS and Android mobile devices that we used in our experiments.

After analyzing HTTP-based video streaming, we found that a video player establishes a sequence of TCP connections by sending HTTP GET messages while playing a video. The behavior of downloading video contents varies depending on the operating system (OS), the hardware performance of the client device and the network condition. Compared to Android, for example, YouTube video player for iOS sends more HTTP GET messages via new TCP connections to download the duplicate video content for the possible re-play activities by the client. We also found that Netflix video player for iOS periodically requests a small chunk of the video (every 10 sec in our measurements) while the video player for Android aggressively downloads the entire video at one go.

Our analysis indicates that a significant amount of video content can be discarded by the video player, even without being stored in the video playout buffer while the video is being played. One of the measurements shows that over 35% of the total video content can be lost by the video player. We found that the undesirable behavior often occurred when the video player established new TCP connections in the middle of downloading the video that was requested previously. Once

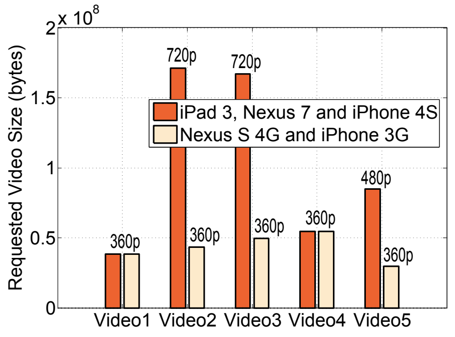


Figure 2: Video resolution with mobile devices over Wi-Fi networks

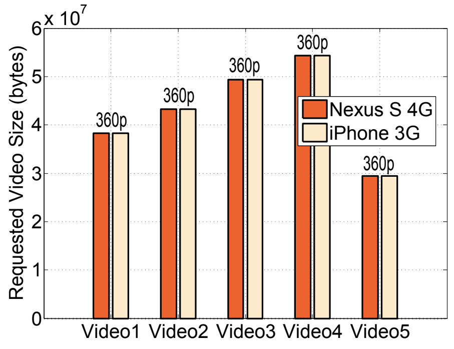


Figure 3: Video resolution with iOS and Android over Wi-Fi networks

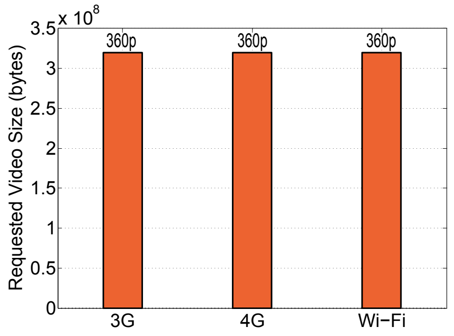


Figure 4: Video resolution with an iPhone 3G over Wi-Fi, 3G and LTE networks

it opened a new TCP connection, it rejected all the incoming video packets via the previous connection. In this paper, we focus on finding the underlying causes. Based on the measurements, we found out that the client received the unnecessary video data when the video player changed resolutions and the video playout buffer was full while downloading the video.

The remainder of the paper is organized as follows. The second section of the paper looks at the related work. In Section 3, we analyze YouTube and Netflix video streaming. In Section 4, we focus on finding problems that cause the wasted video data. We measure the amount of the discarded video traffic in Section 5 and we address the future work in Section 6. Finally, we summarize our conclusions in Section 7.

II. Related work

Much work has been done to characterize HTTP-based video streaming. Gill et al. [1] examined usage patterns, file properties, popularity, referencing characteristics and transfer behaviors of YouTube. Zink et al. [2] collected and analyzed TCP/IP and HTTP headers of the packets between YouTube servers and clients. The early works focused more on studying the characteristics of video such as the distribution of video durations and file sizes and how clients access the video contents provided by YouTube.

Huang et al. [3] analyzed popular HTTP-based video streaming services (Hulu, Netflix and Vudu), but this work focused more on the bandwidth estimation conducted by the client, using fixed devices (PCs and Play-station 3) in a wired network. Finamore et al. [4] focused on analyzing the differences between the network traffic patterns when accessed from PCs over wired networks and from mobile devices over Wi-Fi networks. They showed that the video delivery mechanism of YouTube is more efficient for PCs than for mobile devices due to the limited capabilities of the mobile devices. Hoque et al. [5] identified five different video streaming techniques and analyzed the energy efficiency of mobile video streaming services. Liu et al. [6] compared the performance of YouTube video streaming between Android and iOS mobile devices. They showed that the YouTube video player for iOS downloads more duplicate video data than the video player for Android while playing a video. Rao et al. [7] analyzed the traffic pattern of YouTube and Netflix on both PCs and mobile devices, but did not consider the video packet loss caused by the video delivery mechanisms.

Our analysis is conducted while playing YouTube and Netflix videos on iOS and Android mobile devices over Wi-Fi, 3G and LTE networks. Different from the prior works, in addition to studying the characteristics of HTTP-based video streaming, we emphasize on finding the video packet loss in HTTP-based video streaming. Noticeably, in some cases, a significant amount of video content may be discarded by a video player after transferring data over the limited air-interface, resulting in undesirable waste of resources.

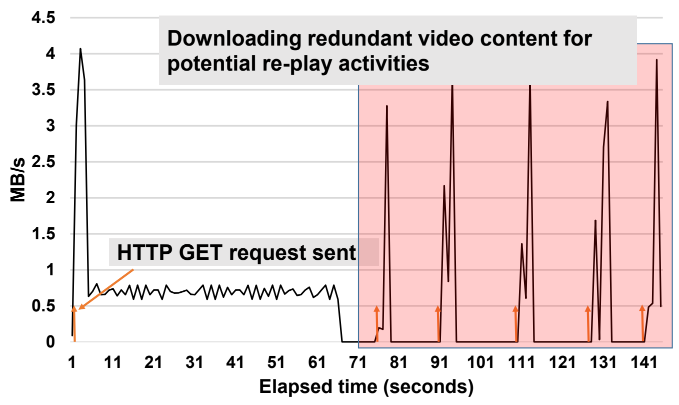
III. Video streaming analysis

Using our testbed (Figure 1), we analyzed the network traffic behaviors of YouTube and Netflix while playing several videos on mobile devices (iOS and Android) over wireless networks (Wi-Fi, 3G and LTE). The videos were played using the video players provided by the content providers, not using a Web browser installed in the user device.

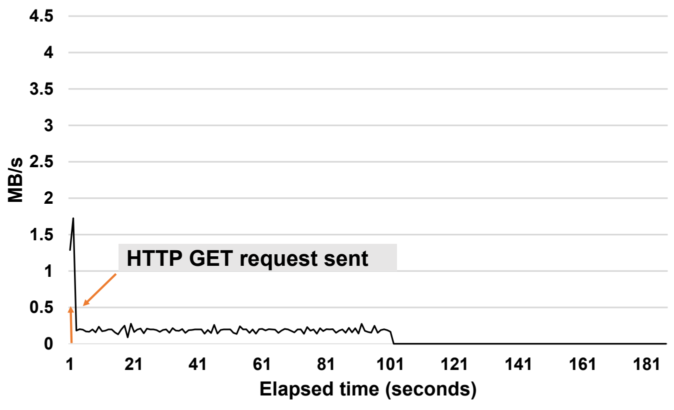
As a baseline analysis, we performed the following three experiments.

- Analyze and contrast the resulting resolution of the same video contents delivered to several mobile devices over Wi-Fi (Figure 2): These experiments indicate that a device capable of higher performance receives a video with higher resolution.
- Analyze and contrast the resulting resolution of the same video contents delivered through iOS and Android over Wi-Fi (Figure 3): These experiments indicate that the video content size remains the same regardless of the operating system.
- Analyze and contrast the resulting resolution of the same video contents delivered to an iPhone 3G via different access networks, namely Wi-Fi, 3G and LTE (Figure 4): These experiments indicate that the video content size remains the same regardless of the network type.

These baseline experiments indicate that a video quality is highly dependent on the hardware specification of a client's device when a client requests a video. The device specification can be obtained from user-agent information in the HTTP GET



(a) iOS (iPad 3)



(b) Android (Nexus S 4G)

Figure 5: TCP throughput while playing a YouTube video on an iPad 3 and a Nexus S 4G over Wi-Fi networks

messages. Hence, such device considerations directly impact Over The Top (OTT) resource consumption, as well as the Quality of Experience (QoE) for the end users. This poses more significant challenges on the wireless capacity planning, which traditionally has been ignoring the capacity consumption per device type, and as it relates to an enforced QoE in a wireless network.

We found that video players send multiple HTTP GET messages to the video content servers while playing YouTube and Netflix videos. For Netflix video streaming, the video player for iOS generates periodic HTTP GET messages while maintaining a single TCP connection. Each spike in Figure 6 corresponds to the video packet transmission from Netflix after the periodic HTTP GET messages (10 sec on average) from the client's device. Unlike iOS, the Netflix video player running on Android devices requests the whole video at one go. Each time it establishes a TCP connection, it uses a different TCP port number from the previous connection.

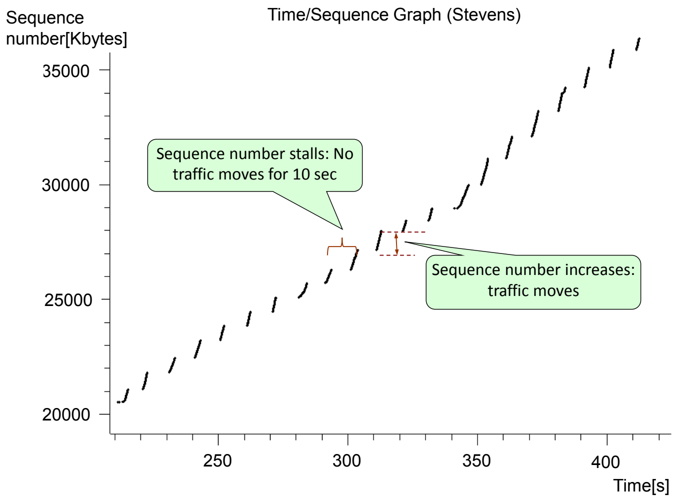


Figure 6: TCP sequence number with Netflix video trace over LTE

Unlike Netflix, the traffic behavior is quite dynamic with YouTube video servers. The dependency of HTTP GET messages varies depending on the operating systems (OSs). For example, our experimental results show that the YouTube video player for iOS typically sends more HTTP GET messages via new TCP connections than the video player for Android while playing a video. As investigated by Yao Liu et al. in the paper [6], one of the reasons is that the YouTube video player for iOS sends additional HTTP GET messages to download duplicate video data for the possible re-play activities after completely downloading the video content. We see the additional traffic on iOS devices (Figure 5a), but do not find it on Android devices (Figure 5b).

Using PCs, we established multiple video sessions via the same wireless connection to load the network. Conditions beyond the access point are unknown, since the testbed is established over the public Internet. Under loaded network conditions, the clients for YouTube and Netflix often experienced buffer underflow (downloading rate < video encoded rate) and the display froze from time to time. Throughout the measurements, we found that the YouTube video player for iOS sent more HTTP GET messages than the video player for Android. Our measurements indicate that the video player for iOS established multiple TCP connections in parallel to download small chunks of the video content, while the video player for Android maintained a single TCP connection under the congested network conditions.

IV. Problem Finding

Based on extensive measurements, we found that the video players for YouTube and Netflix frequently terminated the TCP connection while playing a video. They established another TCP connection, followed by another HTTP GET message to continue receiving the video content. For example, Figure 7 is a simplified video traffic flow diagram between a user device and a YouTube video content server. When a subscriber connects to a video streaming server, the client sends an HTTP GET message (packet 1). The GET request message includes the unique id of the requested video and the user-agent information such as the OS and the video player running on the device. Then the server responds to the client with addresses of a video

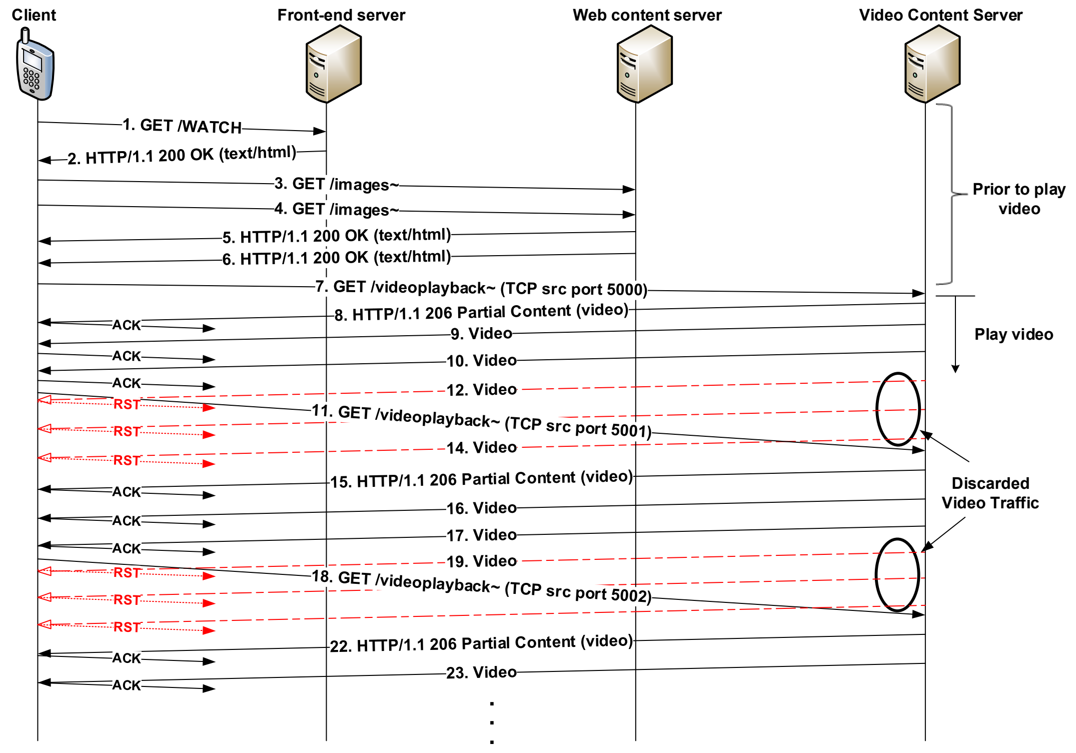

Client
Front-end server
Web content server
Video Content Server
1. GET /WATCH
2. HTTP/1.1 200 OK (text/html)
3. GET /images~
4. GET /images~
5. HTTP/1.1 200 OK (text/html)
6. HTTP/1.1 200 OK (text/html)
7. GET /videoplayback~ (TCP src port 5000)
8. HTTP/1.1 206 Partial Content (video)
9. Video
10. Video
12. Video
11. GET /videoplayback~ (TCP src port 5001)
14. Video
15. HTTP/1.1 206 Partial Content (video)
16. Video
17. Video
19. Video
18. GET /videoplayback~ (TCP src port 5002)
22. HTTP/1.1 206 Partial Content (video)
23. Video
ACK
RST
Prior to play video
Play video
Discarded Video Traffic

Figure 7: YouTube video traffic flows

content server which contains the video file and a Web content server, where the client will download background images from (packet 2). The client transmits a set of HTTP GET messages (packets 3 and 4) in parallel to download background images (packets 5 and 6). The images mostly consist of Web-page images and snap shots of other videos related to the requested video. When the client clicks the play button on the application, it triggers downloading the video content from the video content server by sending an HTTP GET message (packet 7, TCP source port: 5000).

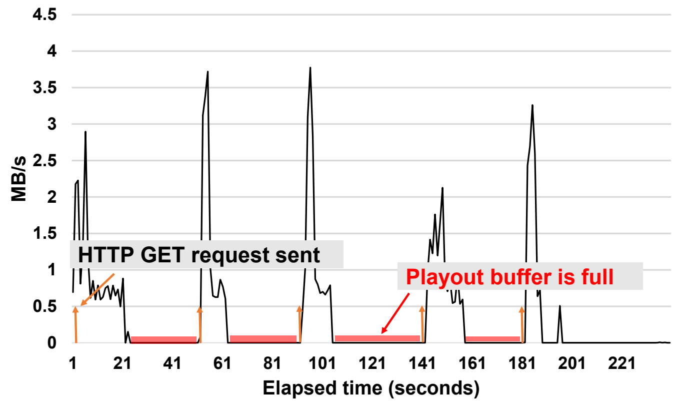
While downloading the video content, the YouTube video player terminates the TCP connection by sending a TCP FIN segment, and establishes another TCP connection (packet 11, TCP source port: 5001). The frequency of this behavior varies with different OSs and different network conditions. As we addressed in the previous section, the video player for iOS sends more HTTP GET messages via new TCP connections when the network is busy. Our research findings point out that a significant amount of video content that had been delivered via the terminated TCP ports was discarded by the video player. This behavior is clearly reflected in Figure 7 between the packets 7 and 15. Packets 8, 9 and 10 are accepted by the client. However, before receiving packet 12, the client transmitted another HTTP GET message via a new TCP connection with port number 5001 (packet 11). The client sends a TCP RST packet to the server each time it receives a video packet via the previous TCP connection. The TCP RST is used to stop the server from sending more video content through the closed port and prevent the server from being left in a state awaiting further transmissions. The client may still receive some video packets through the terminated TCP connection if the server already sent them to the client before noticing that the connection was terminated by the client. These on-fly packets are not accepted by the client since the TCP connection had already been terminated. In other words, the affected video content is simply dropped without being stored in the video playout buffer. Packets 12, 13, 14, 19, 20 and 21 are discarded by the video player in this example.

A. *Dependency of the GET messages on the network conditions*

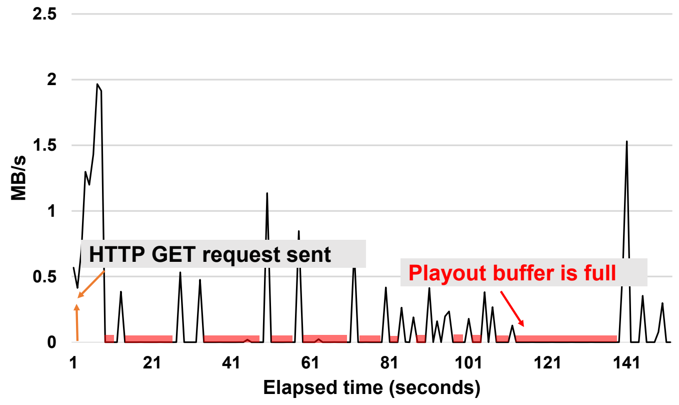
Our measurements show that the video players sent more HTTP GET messages via new TCP connections under fluctuating network conditions. One possible explanation for this behavior, as Finamore et al. also described in the paper [4], is the following: for HTTP adaptive bit-rate streaming, a video content server contains several video files that encode a single video content at multiple bit-rates. Each encoded video file is segmented, and the segment size is typically between two and ten seconds [8], [9]. When a video player changes the quality in the middle of downloading a segment, it needs to re-download the whole size of the segment encoded at the newly requested bit-rate.

B. *Dependency of the GET messages on the device types*

As did Liu et al. [6], we conjecture that the number of HTTP GET messages sent via new TCP connections varies depending on the playback buffer management policies of the video players running on different OSs. When the playout buffer is full, the video player has to stop downloading until there is sufficient space in the buffer. During our experiments, the YouTube video player for Android established a new TCP



(a) Android (Nexus 7) - YouTube



(b) Android (Nexus S 4G) - Netflix

Figure 8: TCP throughput while playing YouTube and Netflix videos on a Nexus 7 and a Nexus S 4G over Wi-Fi networks

connection to resume downloading the video after consuming a certain amount of video content stored in the buffer (Figure 8a). The Netflix video player for Android halted but kept the current connection alive when the buffer was full until it could process the video content (Figure 8b). The Netflix video player for iOS downloaded a small amount of video content every 10 sec, which was of benefit to avoiding the full buffer cases. However, the YouTube video player for iOS also established new TCP connections when the buffer was full.

V. Analysis of Video Packet Loss

We calculated the amount of discarded video traffic using Equation 1 while playing YouTube and Netflix videos on the mobile devices over Wi-Fi, 3G and LTE networks. The hundreds of randomly selected videos include various genres (e.g., action movies, sports, live concerts and animations), popularity, length (from five minutes to a half hour for YouTube and from 45 minutes to an hour for Netflix) and video quality (high quality - HQ and high definition - HD).

During the experiments, we created fluctuating network conditions by using RF devices that cause interference at 2.4 GHz, such as a baby monitor and a cordless telephone. We also intentionally throttled the network bandwidth by using an Iperf tool.

$$Discard\ ratio = 1 - \frac{Goodput}{Total\ throughput} \qquad (1)$$

Table II shows the discard ratio on average. For YouTube, Android devices show much less discard ratio compared to iOS devices. That is because the YouTube video player for iOS sends more HTTP GET messages via new TCP connections than the video player for Android does in order to download the duplicate video content for the potential re-play activities. Due to the small size of memory, an iPhone 3G is likely to establish more new TCP connections than an iPad 3 and an iPhone 4S do while downloading a video. Hence, it typically shows higher discard ratio especially when it downloads a video at a high speed.

TABLE II: Discard ratio (%) on average while playing YouTube and Netflix videos on mobile devices over Wi-Fi, 3G and LTE networks under varying network conditions

Devices	YouTube	Netflix
iPad 3	11.7	0.1
iPhone 4S	11.2	0.1
iPhone 3G	20.7	Not Avail.
Nexus 7	1.6	11.1
Nexus S 4G	9.2	1.4

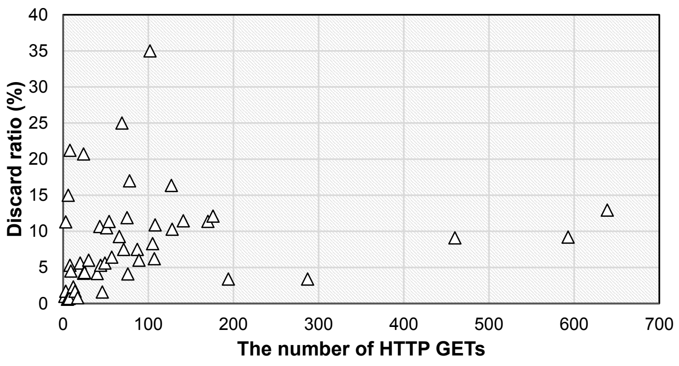


Figure 9: Total number of GETs vs. discard ratio (%) while playing sample YouTube and Netflix videos on mobile devices over Wi-Fi, 3G and LTE networks under varying network conditions

For Netflix, the Netflix video player for iOS shows less discard ratio than the video player for Android. As we stated before, the video player for iOS periodically requests small chunks of video content while maintaining a single TCP connection. Therefore, it has an advantage of avoiding the cases where the buffer is full while downloading a video. The Netflix video player for Android aggressively downloads a video as the YouTube video player for iOS does. In the experiments, most of the discarding video content occurred when the video playout buffer was full. Compared to a Nexus 7, a Nexus S 4G shows less discard ratio. That is because only the lowest video resolution is viewed on the device without switching between video resolutions due to the low hardware performance. We did not experiment with an iPhone 3G because the current video application for Netflix only supports iOS 5 or later.

As shown in Figure 9, the discard ratio does not proportionally increase with the number of GET messages. Instead, it is more related to other conditions such as the performance of a client's device, distance between the client and the server and network conditions while downloading videos. For example, if the round trip time (RTT) between a client and a server is long and the receiver TCP window size is large, the new HTTP GET message may arrive at the server with some significant delay. During our measurements, especially under fluctuating network conditions, we found out that the RTT between the client and the content server was much longer than the one when the network condition was good. During that delay, it is possible for the server to send multiple video packets, which will be discarded by the video client. Also, if the network experiences congestion, the HTTP GET messages may be lost and re-transmitted so that the discarded ratio may be further increased. Our analysis shows that the total video content discarded at the user device, after being delivered over the wireless network, may exceed 35%. These resources should be used for valuable purposes if appropriate actions are taken to prevent delivering video content that gets wasted. In addition to the wasted bandwidth on the down-link stream, which refers to the down-link network resources, there is unnecessary up-link traffic from the client (hosted by the mobile device) and control traffic accompanying the up-link traffic, which would have not occurred if there was no wasteful down-link traffic.

VI. Future Work

Towards the goal of improving the state of the art, we plan to conduct more detailed measurements and analysis of the behaviors of existing video players. We will identify and categorize the shortcomings in using the plain HTTP protocol that the video players are trying to overcome. We aim at proposing a comprehensive solution for mobile video streaming over wireless networks. We envision that our work will combine the best practices of the existing video players, possibly augmented by novel approaches.

VII. Conclusion

This paper explored and analyzed the two most popular HTTP-based video streaming services (YouTube and Netflix) in terms of video traffic behavior in the network, while playing the videos on mobile devices (iOS and Android) over wireless networks (Wi-Fi, 3G and LTE). In the experiments, we point out that the network traffic behavior of downloading videos on-line depend on hardware performance, software running on clients' devices and network conditions between clients and video content servers. Our measurements show that when a client requests a video, a video resolution is selected based on the device types regardless of OSs on clients' devices or network interfaces that clients access.

While delivering a video to a client over HTTP, we also observed that a noticeable amount of video content is being discarded without being stored in the video playout buffer, after the successful delivery to the client. The content discarding occurs when a TCP connection is repeatedly terminated and established. In such cases, the video packets arrived at the client through the terminated TCP connection are discarded. Our experimental results indicate that the average of video packet loss is 10.1%, and it may exceed 35% of its complete content. It causes additional mobile data usage paid by consumers and misuse of the limited network resources. Considering the increasing tendency of watching videos by the mobile users and the scarcity of the network bandwidth, understanding the application traffic behavior is very important in order to develop an effective video delivery mechanism.

References

[1] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube Traffic Characterization: A View From the Edge," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. IMC, San Diego, California, USA, Oct. 2007.

[2] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch Global, Cache Local: YouTube Network Traffic at a Campus Network - Measurements and Implications," *Computer Science Department Faculty Publication Series.Paper 177*, 2008.

[3] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard," in *Proceedings of the 2012 ACM conference on Internet measurement conference*, ser. IMC, Boston, Massachusetts, USA, Nov. 2012.

[4] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, ser. IMC, Berlin, Germany, Nov. 2011.

[5] M. A. Hoque, M. Siekkinen, J. K. Nurminen, and M. Aalto, "Investigating Streaming Techniques and Energy Efficiency of Mobile Video Services," *Computing Research Repository - arXiv:1209.2855*, 2012.

[6] Y. Liu, F. Li, L. Guo, B. Shen, and S. Chen, "A Comparative Study of Android and iOS for Accessing Internet Streaming Services," in *Proceedings of the 14th international conference on Passive and Active Measurement*, ser. PAM, Hong Kong, China, Mar. 2013.

[7] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network Characteristics of Video Streaming Traffic," in *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies*, ser. CoNEXT, Tokyo, Japan, Dec. 2011.

[8] C. Müller, S. Lederer, and C. Timmerer, "An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments," in *Proceedings of the 4th Workshop on Mobile Video*, ser. MoVid, Chapel Hill, North Carolina, USA, Feb. 2012.

[9] S. Lederer, C. Müller, and C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys, Chapel Hill, North Carolina, USA, Feb. 2012.