# Towards Dynamic QoS-aware Over-The-Top Video Streaming

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

***Abstract*—We present a study of traffic behavior of two popular over-the-top (OTT) video streaming services (YouTube and Netflix). Our analysis is conducted on different mobile devices (iOS and Android) over various wireless networks (Wi-Fi, 3G and LTE) under dynamic network conditions. Our measurements show that the video players frequently discard a large amount of video content although it is successfully delivered to a client.**

**We first investigate the root cause of this unwanted behavior. Then, we propose a Quality-of-Service (QoS)-aware video streaming architecture in Long Term Evolution (LTE) networks to reduce the waste of network resource and improve user experience. The architecture includes a selective packet discarding mechanism, which can be placed in packet data network gateways (P-GW). In addition, our QoS-aware rules assist video players in selecting an appropriate resolution under a fluctuating channel condition. We monitor network condition and configure QoS parameters to control availability of the maximum bandwidth in real time. In our experimental setup, the proposed algorithm (and platform) achieves nearly 21% improvement in downlink bandwidth saving, creating a better user experience under challenging radio conditions.**

***Index Terms*—Video Streaming, Mobile Wireless, HTTP Progressive Video, HTTP Adaptive Bit-rate Streaming, Over The Top Applications, Video QoE**

## I. Introduction

Today's popular OTT video content providers such as YouTube and Netflix stream their videos on demand (VOD) without any technical support from network operators. The popularity of OTT video content is growing steadily. According to Cisco [1], video streaming traffic accounts for 52.8% of total mobile data traffic in 2011, and it is expected to reach 66.4% in 2015. The majority of mobile traffic on video streaming is associated with YouTube and Netflix, which generates 88% of total mobile video streaming traffic [2].

Most OTT video content providers use HTTP adaptive bit-rate (ABR) streaming such as Apple HTTP Live Streaming (HLS), Microsoft IIS Smooth Streaming, Adobe HTTP Dynamic Streaming and Dynamic Adaptive Streaming over HTTP (DASH). A video server contains several video files encoded at multiple bit-rates. The video files are chopped into small segments and then they are streamed over HTTP to the client on demand. The video player adjusts the quality of video stream based on the available bandwidth in the network measured at the application layer and the CPU capacity of the client's device.

The self-adjusting mechanism implemented in a video player is designed to provide clients with the highest Quality-of-Experience (QoE) for given network conditions. However, due to the lack of direct knowledge of access networks, frequent user mobility and rapidly changing channel conditions, the video player is difficult to effectively trace the network conditions. This may cause the video player to select an inappropriate resolution while playing a video. A video player may periodically send feedback to a video content server to accurately estimate network capacity, but bandwidth constraints often limit the frequency of end-to-end feedback [3]. Alternatively, Software-Defined Networking (SDN) may enable OTT video content providers to partially provision network resource in collaboration with network operators. However, SDN infrastructure in Wide Area Network (WAN) has not been practically standardized yet [4].

Providing a seamless video viewing experience is significantly important for network operators to increase the number of subscribers in their networks. The better the experience, the longer and more subscribers will consume video contents in their networks. To achieve this, we propose to build a dynamic QoS-aware video streaming platform in an LTE network. As a perspective of a network operator, our proposed platform does not require any technical support from OTT video content providers. In this paper, we first attempt to understand video traffic behavior of current OTT video services, and improve OTT video content delivery in an LTE system.

**Analysis of OTT video content delivery and its impact on mobile networks -** We first examine video traffic of the two popular OTT video services (YouTube and Netflix) while playing videos on mobile devices (iOS and Android) over wireless networks (Wi-Fi, 3G and LTE) under varying network conditions. During our experiments, we observed that the video players frequently discard a large amount of successfully downloaded video packets. This unwanted behavior occurs when the video player often changes resolution in unstable network conditions and the video playout buffer is almost full during a download. The video player also discards some video packets when a client moves a playback slide bar before completely downloading a requested video previously. Our analysis shows that the average video packet loss is 10.1%, and that the loss may exceed 35% of its complete content. Needless to say, this behavior consumes network resources and causes additional mobile data usage on clients.

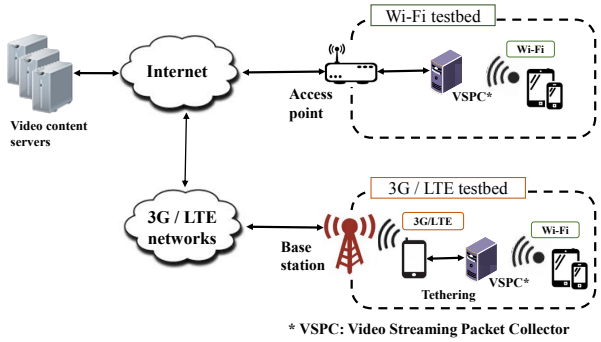



**Figure 1:** A testbed for a video traffic measurement using VSPC

**Improving network efficiency and video QoE in LTE -** P-GW (also known as PDN Gateway) provides connectivity from an LTE user to external packet data networks. It is responsible for performing policy enforcement, user IP-address allocation, packet filtering and charging. In the proposed platform, our selective packet discarding mechanism implemented in P-GW drops the potentially wasted video content in advance before it is delivered to a client. It prevents unnecessary mobile data usage paid by users and misuse of the limited network resource over the air interface. In order to improve video QoE for end-users, our proposed QoS rules in P-GW are designed to dynamically manipulate LTE QoS parameters such as A-AMBR and MBR that indicate the maximum possible data rates, based on network conditions between a client and a base station (eNodeB). By throttling TCP throughput, our QoS rules assist a video player to choose a proper resolution under fluctuating network conditions.

Our contributions can be summarized as follows:

1) We discover the underlying causes of the video packet loss on HTTP-based mobile video streaming (Section II and III); and
2) From the network operator's point of view, we strengthen an existing OTT video delivery system. Our evaluation shows up to 20.58% improvement in saving the downlink bandwidth on the air interface, and the proposed mechanism enhances the video watching experience by reducing buffer starvation (Section IV and V).

The remainder of the paper is organized as follows. In the second section, we elaborate on the analysis of YouTube and Netflix video streaming. In Section III, we focus on finding problems that cause the waste of video content, and our proposed solutions are described in Section IV. We evaluate our proposal in Section V and look at the related work in Section VI. Finally, we summarize our conclusions in Section VII.

## II. HTTP-based Video Streaming Analysis

In this section, we present a traffic characterization study of two popular OTT video services: YouTube and Netflix.

**Testbed setups -** As shown in Figure 1, we have designed the Video Streaming Packet Collector (VSPC) on a Linux machine to capture and analyze TCP/IP and HTTP packets on video streaming between a client and a video server. Using our testbed, we played hundreds of YouTube and Netflix videos on different mobile devices (iOS and Android) via different wireless networks (Wi-Fi, 3G and LTE) under varying network conditions. Using `netem`, a networking emulation tool [5], we intentionally shaped TCP throughput and added latency between the VSPC and the clients. Table I shows the hardware specifications of the selected iOS and Android mobile devices used in our experiments. We played videos using standalone video applications provided by the video content providers, not the Web browsers installed in the mobile devices.

**TABLE I:** iOS and Android mobile devices

| Devices | OS versions | Screen 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 | 1024 MB |
| Nexus S 4G | Android 4.1.1 | 480 x 800 | 512 MB |

Based on these measurements, we point out that mobile devices show distinct approaches of downloading videos, depending on the operating system, the hardware specification of the client's device and the network condition.

### A. An Analysis of YouTube Video Streaming

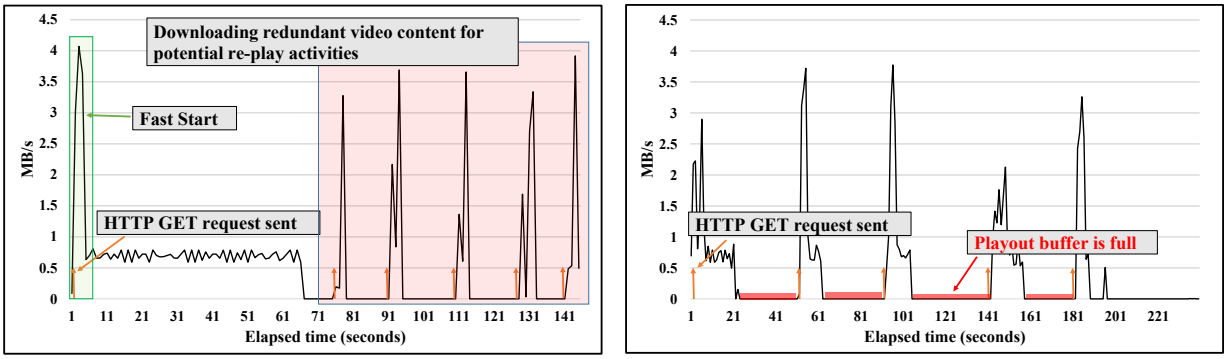
We analyzed how YouTube streams videos to mobile devices over wireless networks. In our experiments, we played 450 videos on the mobile devices under varying network conditions. The videos were randomly selected in terms of the diversity in genre (animation, action movie, music video, live concert and sports), length (from five minutes to two hours) and video quality (high quality - HQ 360p and high definition - HD 720p).

Our analysis on YouTube video streaming can be summarized as follows:

**Selecting resolution based on hardware specification -** YouTube video player selects resolution regardless of the operating system and the radio interface, but it is affected by hardware performance. The video players on iPad 3, Nexus 7 and iPhone 4S chose HD (720p) resolution when they requested the videos. The video players on iPhone 3G and Nexus S 4G selected HQ (360p) resolution due to the small size of the display screen.

**Sending plain HTTP GET messages to request a video -** YouTube video player for iOS uses a plain HTTP GET request including a header field that specifies the byte range of the video file (e.g., `Range: bytes=100~1000`). The video content server then responds with an `HTTP 206 Partial Content` message (status code: 206) and sends the requested range of the video. Unlike iOS, the video player for Android only defines the starting point in the HTTP header (e.g., `Range: bytes=100~`). Then, the video content server pushes the video from the requested starting point to the end of the video file.

**Performing a fast start downloading -** As shown in Figure 2a, a YouTube video content server sends its content with a high speed for the first few seconds. This mechanism is also known as *Fast Start* [6] that provides a way for a video player




**(a)** iOS (iPad 3) **(b)** Android (Nexus 7)

**Figure 2:** TCP throughput at iOS and Android devices while downloading a YouTube video over Wi-Fi networks

to fill the initial buffer at speeds higher than the bit-rate of the requested video content.

**Sending a sequence of HTTP GET messages while downloading a video -** YouTube video player repeats sending an HTTP GET message and receiving a partial video content while playing a single video. The video player establishes a new TCP connection every time it sends a new HTTP GET message. The number of HTTP GET messages varies depending on the operating system, the hardware performance of the client's device and the network condition.

*1) Dependency of operating systems:* Our experimental results show that YouTube video player for iOS typically sends more HTTP GET messages than the one for Android during a download. As investigated by Yao Liu et al. [7], one of the reasons is that YouTube video player for iOS sends additional HTTP GET messages to request duplicate video contents after the video is completely downloaded. The redundant video content is stored in the buffer for possible re-play activities by the clients. We see this additional traffic on iOS devices (Figure 2a) but do not observe it on Android devices.

*2) Dependency of the size of video playout buffer:* The number of HTTP GET messages is also related to the size of video playout buffer, which generally varies depending on hardware specifications. Through the experiments, we found out that the video player often closed an active TCP connection after a certain amount of video contents is downloaded. This behavior occurred more frequently on iPhone 3G and Nexus S 4G, which have a smaller memory size compared to iPad 3, Nexus 7 and iPhone 4S. After consuming a certain amount of video content stored in the buffer, it resumes downloading the video by sending an additional HTTP GET message via a new TCP connection (Figure 2b).

*3) Dependency of network condition:* Using `netem`, we intentionally shaped the bandwidth in the network and added latency on video streaming while playing the videos. Based on these measurements, we found out that when the network is congested, the video player for iOS sends a sequence of HTTP GET messages. Each HTTP GET message requests a small part of the video. It repeats sending the message until the video is completely downloaded. However, under stable network conditions it sends only one HTTP GET message that requests the entire video file at once. We do not observe this behavior on Android devices.

**ABR streaming not available on Android -** We found out that YouTube video player for Android does not support HTTP adaptive bit-rate streaming. According to the study of HLS by Andrew [8], YouTube supports HLS for iOS devices. Android, however, lacks the support of native HLS. During the experiments, the video player for iPad 3 kept switching resolutions between HQ 360p and HD 720p under fluctuating network conditions. On the contrary, the video player on Nexus 7 kept the high resolution (HD 720p) until the end, which caused severe buffer underflows while playing the video.

### *B. An Analysis of Netflix Video Streaming*

When a video player requests a Netflix video, it receives a manifest file containing the information of video qualities over an SSL connection. Therefore the information cannot be retrieved via packet capturing. According to Netflix [9], iOS and Android mobile devices basically support video streaming in 480p, and the HD (720p and 1080p) videos can be viewed on devices that are capable of higher performance such as Sony PlayStation 3 and Apple TV. This indicates that the video quality of Netflix is also selected based on the hardware specification of the client's device.

Our experimental results can be described as follows:

**Two separate TCP connections -** Unlike YouTube, Netflix video player simultaneously establishes two TCP connections (video and audio) with a video content server to stream a video.

**Periodic HTTP GET messages from iOS -** We found out that the Netflix video player for iOS generates periodic HTTP GET messages to download a video. The HTTP header in each HTTP GET message specifies the short range of the video or audio file to be downloaded. The video and audio files are requested at a different pace. During the experiments, the video player requested the video file every 10.4 seconds and the audio file every 10.1 seconds on average. Unlike iOS, the traffic behavior on Android is quite straightforward. The Netflix video player running on Android devices requests the whole video and audio files at once.

**TABLE II:** The analysis of YouTube and Netflix video streaming

| | Operating systems | Fast start | ABR | Num. of concurrent TCP connections while playing a video | Requested size of video per a TCP connection |
|---|---|---|---|---|---|
| YouTube | iOS | ✔ | ✔ | 1 | Varies depending on network conditions [1] |
| | Android | ✔ | ✘ | 1 | Entire video file requested at once |
| Netflix | iOS | ✔ | ✔ | 2 (video and audio separated) | A small chunk of video requested in periodic messages |
| | Android | ✔ | ✔ | 2 (video and audio separated) | Entire video file requested at once |

[1] A small chunk of a video file repeatedly requested until it downloads a whole video file under unstable network conditions while a whole video file requested at once over a single TCP connection under stable network condition

**ABR streaming available on both iOS and Android:** We found out that Netflix video player provides HTTP adaptive bit-rate streaming for both iOS and Android. The low resolution was first played at the beginning and subsequently shifted to higher quality depending on the network conditions.

Our analysis of YouTube and Netflix video streaming can be briefly summarized in Table II. We conducted the same experiments via different wireless access networks, namely 3G and LTE. For example, we played the same YouTube and Netflix videos on the same mobile devices via 3G and LTE networks. We compared the HTTP GET messages in both cases and did not find any differences caused by the wireless interfaces. The same quality of video was played on the mobile device, and the video player downloaded the content in the same way regardless of the wireless interfaces.

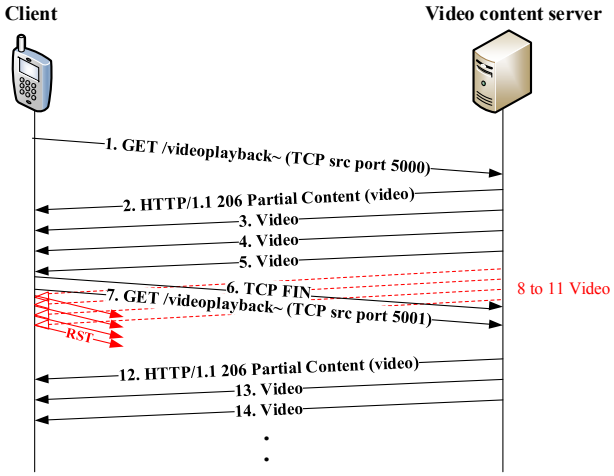



**Figure 3:** Video traffic flow diagram between a client and a YouTube video content server

## III. Badly Designed Video Players Waste Network Bandwidth

Our analysis on HTTP-based video streaming indicates that a video player sends a sequence of HTTP GET messages to download a video and a large amount of video content may get discarded, even without being stored in a playout buffer during a download. This unwanted behavior occurs when a video player terminates an open TCP connection before completely downloading the requested video content.

As an example, Figure 3 shows a simplified video traffic flow diagram between a client and a YouTube video content server. When a client plays a video, the video player sends an HTTP GET message (packet 1, TCP source port 5000) to download the video file. The video packets 3, 4 and 5 are successfully delivered to the video player. However, before receiving the next video packets, the video player closes the port number 5000 (packet 6) and transmits another HTTP GET message via a new TCP connection with the source port number 5001 (packet 7). In the meantime, those video packets that were sent by the video content server prior to noticing the termination, continue to arrive at the TCP port 5000. In addition, a TCP RST packet is sent to the video content server each time the video player receives a video packet via the terminated TCP port. Consequently, the TCP layer does not deliver those packets (packets 8 to 11) to the video player.

While experimenting, we found out that there are mainly three cases that cause this problem.

1) When a video player changes resolution, it closes an open flow and establishes a new TCP connection. For adaptive bit-rate streaming, a video content server contains several video files encoded at multiple bit-rates. Each encoded video file is segmented and the segment length is typically between two and ten seconds [10], [11]. When a video player changes the quality, it terminates the open connection and sends a new HTTP GET message. The video player may receive the segments encoded at the previously requested bit-rate, before the newly requested GET message arrives at the video content server. When such events occur, the video packets through the terminated TCP port will be discarded.
2) When the video playout buffer is almost full, a video player closes an open TCP connection. After carefully analyzing the TCP/IP and HTTP packets, we found out that this often occurs when the playout buffer size is too small to download a video at a high speed (e.g., playing videos on iPhone 3G). When a video player closes the open TCP connection, however, the TCP layer may keep downloading the video content as much as the remained RWND size. Consequently, the video packets received via the terminated TCP connection will be discarded.
3) When a client moves a playback slide bar while playing a video, a video player establishes a new TCP connection. Every time the slide bar is moved, a video player immediately terminates an open TCP connection and sends a new HTTP GET message that contains a new requested range of bytes of the video. The video packets that continue to arrive at the closed TCP connection will no longer be accepted by the video player.

**TABLE III:** Average and standard deviation of discard ratio (%) while playing YouTube and Netflix videos on mobile devices over Wi-Fi, 3G and LTE networks under fluctuating network conditions

| Devices | YouTube Avg. (Stdev.) | Netflix Avg. (Stdev.) |
|---|---|---|
| iPad 3 | 11.77 % (1.23) | 0.13 % (0.09) |
| iPhone 4S | 11.25 % (1.22) | 0.5 % (0.48) |
| iPhone 3G | 13.01 % (9.02) | Not Avail. |
| Nexus 7 | 1.79 % (0.45) | 11.11 % (9.1) |
| Nexus S 4G | 9.23 % (1.81) | 1.413 % (0.68) |

### A. Calculating Discard Ratio

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

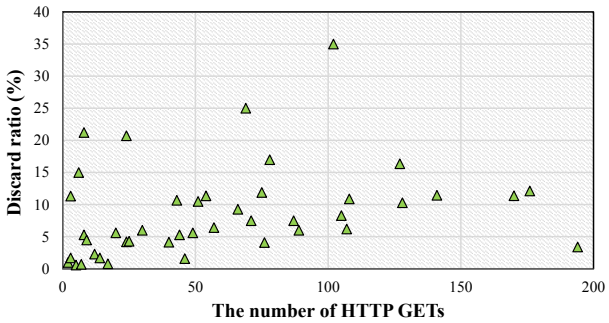
**Experimental setups -** Using Equation 1, we calculated the amount of discarded video traffic ratio. The specific experimental setups are:

- One hundred of YouTube and Netflix videos were played on mobile devices via Wi-Fi, 3G and LTE networks in our testbed (Figure 1). During the experiments, the video players selected one resolution among HQ (360p) and HD (720p) based on their own self-adjusting mechanism;
- Using `netem` [5], we artificially manipulated packet delay (avg. 50 ms $\pm$ 10 ms random variation), packet loss rate (avg. 5%), packet duplication rate (avg. 3%), packet corruption rate (avg. 3%) and packet re-ordering rate (avg. 5%) between mobile devices and our VSPC tool in Figure 1.
- Using `iperf` [12], we also generated heavy TCP traffic to the same network to overload the network, and manually moved the slide bars on the video players while playing the videos[1].

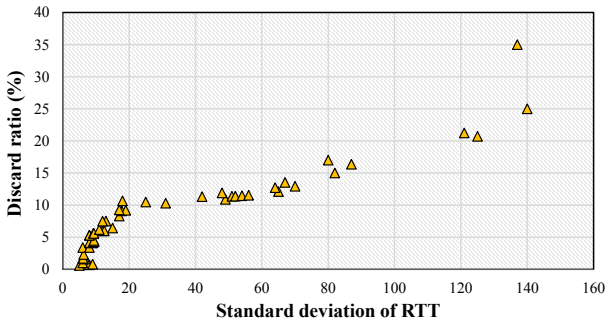
Table III shows the experimental results. We did not experiment with iPhone 3G and Netflix because the current video application for Netflix only supports iOS 5 or later. Our analysis is briefly summarized below.

- **For YouTube, Android shows much less discard ratio compared to iOS.** That is mainly because YouTube video player for iOS sends more HTTP GET messages with new TCP connections than the video player for Android, in order to download the redundant video content for potential re-play activities. YouTube video player on Android devices does not support ABR streaming, thus it does not switch resolutions. This resulted in the lower discard ratio compared to the case of iOS.
- **For Netflix, iOS shows much less discard ratio compared to Android.** As we stated before, the video player for iOS periodically requests a small chunk of video, which provides a means of avoiding buffer overflow cases while downloading a video.
- **Discard ratio is affected by hardware performance of a client's device.** For Netflix, Nexus S 4G shows less discard ratio compared to Nexus 7. That is because only the low video resolution is selected on the device due to the small size of the display screen. For YouTube, iPhone 3G and Nexus S 4G typically sent more HTTP GET messages due to the small size of the playout buffer, which caused higher discard ratio in comparison to other devices with larger size of the playout buffer such as iPad 3 and Nexus 7.

[1]For each video, we moved the slide bar ten times every 30 seconds from currently playing time to the unbuffered point.




**(a)** Total number of GETs vs. discard ratio (%)




**(b)** RTT vs. discard ratio (%)

**Figure 4:** Discard ratio (%) for the YouTube and Netflix video samples

### B. Key Observations

Through Section II and III, our key findings can be summarized as follows:

1) The number of GET messages sent by a video player while downloading a video is not associated to the network interface, namely Wi-Fi, 3G and LTE;
2) HTTP-based OTT video players may discard a large amount of video packets although they are successfully delivered over the air interface. The discard ratio does not proportionally increase with the number of GET messages (Figure 4a); and
3) Instead, discard ratio is more related to other conditions such as the performance of clients' devices and the network conditions while playing videos (Table III and Figure 4b). For instance, the large amount of video content which is sent by the video content server before receiving a TCP RST packet will be discarded when the round trip time (RTT) between the server and the client is long and the receiver TCP window size is large.

As we described before, the more HTTP GET messages via new TCP connections a video player sends, the more delivered

content is likely to be discarded. Also, if the network is congested, the HTTP GET messages may get lost or retransmitted, which will further increase the discard ratio.

Consequently, these experiments indicate that the network traffic behaviors of YouTube and Netflix video streaming depend on operating systems, hardware specifications of clients' devices and network conditions. Hence, such device considerations are directly related to wireless network resource consumption as well as video QoE for end-users.

## IV. Improving OTT Video Content Delivery in LTE

Wireless network resources such as radio spectrum and backhaul transport between the base station and the core network are limited and expensive. As described in Section III, current OTT video players may waste a large amount of network resources and cause additional mobile data usage on clients. In order to improve OTT video content delivery, we propose a dynamic QoS-aware video streaming algorithm in LTE.

### A. QoS in LTE

We first describe a brief background of QoS in LTE. In an LTE system, QoS is implemented on a set of bearers between a client and P-GW. QoS determines how an IP packet flow is handled at eNodeB when it experiences congestion in terms of scheduling policy, queue management and rate shaping. There are two types of bearers: dedicated bearer and default bearer. A default bearer is established when a client is connected to an LTE network, and several dedicated bearers can be added when it needs QoS-enabled services such as VoIP and video streaming.

There exist two types of dedicated bearers: GBR type and Non-GBR type. In a GBR mode, it provides minimum and maximum guaranteed bit-rate per an Evolved Packet System (EPS) bearer using GBR and MBR parameters. In a non-GBR mode, on the other hand, it provides best-effort packet delivery services. Although non-GBR bearers do not provide guaranteed bit-rates, it still enables managing QoS using A-AMBR and UE-AMBR parameters.

- A-AMBR: This indicates the maximum possible bit-rate for all of best effort services on a specific access point name (APN).
- UE-AMBR: This represents the maximum possible bit-rate for all of best effort services on a particular client. It prevents a client from taking all the available bandwidth from the other LTE clients over the same air interface.

The 3GPP standards have defined nine QoS class of identifiers (QCIs) in total which are characterized by priority, packet delay budget and packet error loss rate. According to the standardized QCIs in LTE [13], TCP-based progressive video streaming is assigned to QCI 8 and 9, which indicate non-GBR type, 300 ms packet delay tolerance and $10^{-6}$ acceptable packet error loss rate.

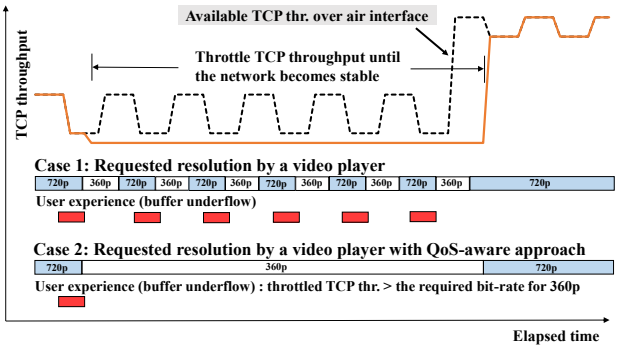



**Figure 5:** The impact of controlling TCP throughput on video QoE

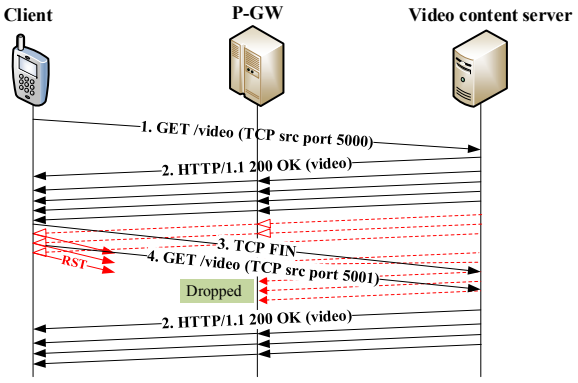



**Figure 6:** P-GW drops the potential wasted video packets in advance before delivering them to a client

### B. Dynamic QoS-aware Video Content Delivery in LTE

When network conditions are fluctuating, today's OTT video players may repeat switching resolution while playing a video. This may result in frequent buffer underflows and a large amount of packet loss, as shown in Figure 5 (Case 1). To solve this problem, we propose differentiated QoS solutions that dynamically change QoS parameters based on network conditions between clients and eNodeBs. The proposed architecture requires no additional implementation at server and client sides. It is designed to conduct the following two objectives: **Swiftly downgrading QoS parameters based on network capacity over the air interface -** OTT video players such as YouTube specify the requested video resolution in the HTTP GET message. P-GW can inspect the URL and obtain the requested bit-rate information selected by the video player. For example, it typically requires at least 0.8 Mb/s for HQ (360p) videos and 5.4Mb/s for HD (720p) videos. With our proposed architecture, if the requested video quality is not suitable under the network capacity (e.g., available bandwidth and buffer status) reported by eNodeBs, our algorithm implemented in P-GW decides to degrade QoS parameters such as A-AMBR and MBR that indicate the maximum possible data rates, in order to throttle TCP throughput of the video streaming flow until the network becomes stable.

This brings the same effect of sending a notification to the video player advising that it is better to select the low resolution over the network. As a result, a video player prevents the frequent changes of video resolution that may

cause byte waste under unstable network conditions, and thus improves video QoE (Case 2 in Figure 5). We describe the experimental results in Section V.

**Discarding potential wasted video content in advance before delivering it to a client -** In the previous section, we addressed that a video player sends a TCP RST segment each time it receives an unexpected video packet via the terminated TCP port. We intend to drop unnecessary video packets in advance before delivering them to the client over the air interface. In our proposed architecture, P-GW acts as a firewall that performs TCP header inspection and discards the unwanted traffic. When it captures the TCP FIN or RST segments sent from the video player, it starts discarding incoming video packets of which destination is the closed TCP port (Figure 6). This has the advantage of saving downstream bandwidth from P-GW to the client.

## V. Performance Evaluation of The Dynamic QoS-aware Video Streaming Architecture

In this section, we perform evaluation of our dynamic QoS-aware video streaming platform. We conduct video QoE measurements for end-users and compare the performance in terms of discard ratio while playing YouTube and Netflix videos on mobile devices over Wi-Fi networks.

**Building a testbed in Wi-Fi -** Instead of using LTE simulators such as MATLAB [14] and OPNET [15], we have designed a testbed over Wi-Fi interface to take realistic OTT video streaming traffic into account. As shown in Figure 7, in our prototype, a Wi-Fi access point and a proxy server, respectively, act as an eNodeB and P-GW. All the video packets between a client and a video content server pass through the proxy server.

We designed a set of QoS rules (Algorithm 1) to control the video streaming flows. These rules are designed to implement our proposed QoS-aware video streaming platform in LTE. Let $BR_{req.}$ be the requested bit-rate selected by the video player. Let $BW_{avail.}$ and $SNR_{avg.}$ respectively denote the available bandwidth to the video stream and the signal-to-noise ratio (SNR) over the air interface. During the experiments, we calculated the average of SNR for every five seconds, and compared the value with the predefined SNR noise threshold ($N_{thr.}$) to decide if the network was fluctuating or not. We note that the proxy server acting as P-GW only throttles TCP throughput when the video player requests the inappropriate resolution under varying network conditions. It will decrease the maximum allowable TCP throughput on the video streaming flow, which leads the video player to switch to lower resolutions quickly.

The specific testbed setups are:

- A hundred of YouTube and Netflix videos were randomly selected. During the experiments, the average playing time of each video was about 10 minutes;
- Two clients (Client A and B) on iPads simultaneously requested the same video in the same network. We only applied our QoS algorithm to Client B and compared the performance against the baseline measured through Client A. The video players dynamically selected one resolution among 360p, 480p and 720p based on the network conditions;
- We installed Linksys WRT54GL (802.11b/g - 54 Mbps) Wi-Fi access point, and installed an open-source firmware, DD-WRT, on it. We wrote a script on the access point to periodically (every five seconds) report feedback on the network conditions to the proxy server. The feedback contains the transferred RX / TX bytes (used to calculate the available bandwidth) and signal / noise level (dBm) on the air interface;
- To make the network fluctuate, we turned on and off RF devices that cause Wi-Fi interference at 2.4 GHz, such as baby monitors and cordless telephones. To load the network, we also intentionally added the TCP traffic using a network testing tool, `iperf`;
- During the experiments, the average SNR was 47 dBm in the clean environment, but it went down to 18 dBm with interference. Based on the measurements, the SNR noise threshold ($N_{thr.}$) was set to 25 dBm;
- ***In Step 1***, using `netem`, we set the maximum available TCP throughput 0.2 MB/s, 0.4 MB/s and 0.7 MB/s for 360p, 480p and 720p resolution, respectively; and
- ***In Step 2***, using `netfilter` and `iptables` [16], we discarded the unwanted traffic at the proxy server.

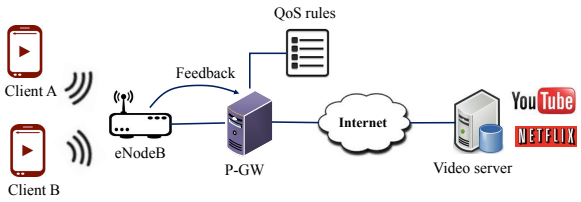


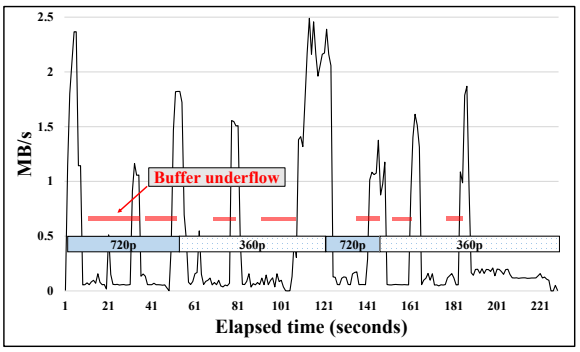

**Figure 7:** Testbed setups for evaluation

**Algorithm 1**

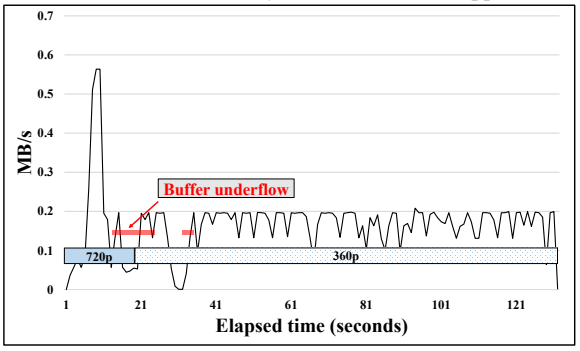
**if** an HTTP GET msg received from a video player **then**
  **if** $BR_{req.} \geq BW_{avail.}$ *or* $SNR_{avg.} \leq N_{thr.}$ **then**
    (***Step 1***) Throttle TCP throughput of the video streaming flow until the network conditions become stable
  **end if**
**end if**

**if** TCP RST or FIN received from a video player **then**
  $SET_{closed} \leftarrow TCP_{srcport}$
**end if**

**if** a video pkt received from a video server **then**
  **if** $TCP_{dstport}$ *in* $SET_{closed}$ **then**
    (***Step 2***) Discard the video packet
  **else**
    Pass the video packet to the video player
  **end if**
**end if**




**(a)** Client A without a dynamic QoS-aware approach




**(b)** Client B with a dynamic QoS-aware approach

**Figure 8:** TCP throughput while playing a YouTube video on iPad 3

**Improving video QoE for end-users -** To evaluate the video QoE, we measured how long the client experienced buffer underflows while watching a video. As a result, Client B with our QoS-aware approach experienced the average of 32 seconds less buffer underflows compared to Client A.

For example, Figure 8 shows the TCP throughput while playing the same YouTube video on iPad 3 devices in our testbed. We measured the TCP throughput until the video players on Client A and B fully downloaded the video files under the fluctuating network conditions. As depicted in Figure 8a, Client A experienced many buffer underflows while playing the video (72 seconds out of 225 seconds). During the experiment, it sent 100 HTTP GET messages in total and the discard ratio was 11.6%.

On the other hand, while employing our QoS-aware algorithm (Figure 8b), the TCP throughput of the video streaming on Client B was adjusted to better cope with the measured fluctuations in channel quality. After the TCP throughput got strangled at time 9, it took only 10 seconds for the video player to change the quality[2]. As the low resolution was selected, it played the video with much fewer buffer underflows. As a result, Client B experienced buffer freezing only 15 seconds out of 132 seconds until it fully downloaded the entire video file. Unlike Client A, it only sent 12 HTTP GET messages, and showed a discard ratio of 0.41%.

**Saving bandwidth over the air interface -** We stored a TCP dump file for each experiment, and calculated the discard ratio to compare the performance. As shown in Table IV, employing our dynamic QoS-aware algorithm yields lower discard ratio. For instance, our proposed solution reduced the discard ratio up to 20.58% (case of iPhone 3G and YouTube), compared to the baseline.

[2]During our experiments, we found that Netflix and YouTube video players switched to lower resolution in 11.8 seconds on average after the TCP throughput was throttled.

**TABLE IV:** Discard ratio (%) on average while playing YouTube and Netflix videos on mobile devices over Wi-Fi networks under fluctuating network conditions

| Devices | YouTube | | Netflix | |
|---|---|---|---|---|
| | Client A | Client B | Client A | Client B |
| iPad 3 | 13.54 % | 0.87 % | 0.16 % | 0.01 % |
| iPhone 4S | 12.72 % | 0.03 % | 0.5 % | 0.38 % |
| iPhone 3G | 20.72 % | 0.14 % | Not Avail. | Not Avail. |
| Nexus 7 | 2.16 % | 0.49 % | 14.86 % | 0.13 % |
| Nexus S 4G | 8.54 % | 0.01 % | 11.25 % | 0.15 % |

## VI. Related Work

Several researchers have studied on characterizing HTTP-based video streaming.

Zink et al. [17] analyzed YouTube traffic in a university campus network. By analyzing TCP/IP and HTTP packets, they characterized the duration and popularity of the YouTube videos, and access patterns for YouTube video streaming. Based on their measurements, they proposed proxy-caches for video streaming to save the network traffic and enhance the user experience.

Rao et al. [18] identified the streaming strategies used by YouTube and Netflix via Wi-Fi interface. They showed that the streaming strategies vary depending on the video players and the types of container used for delivering video content to a client.

Hoque et al. [19] conducted a measurement study of three popular video streaming services (YouTube, Dailymotion and Vimeo) on mobile devices over Wi-Fi and 3G networks. They analyzed the energy efficiency of the five different video streaming techniques used by the mobile video streaming services.

Liu et al. [7] analyzed and compared the performance of YouTube video streaming between Android and iOS mobile devices. They showed that Android and iOS use different approaches to download a video. After analyzing the traffic patterns of YouTube and different buffer management methods, they found out that iOS devices receive more duplicate YouTube video content than Android devices do.

Balachandran et al. [3] introduced a proxy-based architecture for streaming media services over wireless networks. They have designed a proxy server that enables to report feedback on network conditions to a video server on behalf of a client.

Alcock et al. [20] investigated the streaming flow control technique conducted by YouTube. They traced video packets of YouTube traffic over residential DSL and academic networks. They found out that YouTube sends large bursts of video contents, which may lead to frequent packet loss and significantly reduce throughput while delivering videos to clients.

Huang et al. [21], [22] have shown that many factors such as the size of a video chunk, dynamic TCP congestion control algorithm and competing flows in the same network make it hard to pick a proper video streaming rate on clients. To resolve the issue, they have introduced playout buffer-based rate adaptation for HTTP-based video streaming.

**Differences from the prior work:** In addition to studying the characteristics of HTTP-based video streaming, we focus on finding the root cause of video packet loss on mobile devices over Wi-Fi, 3G and LTE networks. Noticeably, in some cases, a significant amount of video content may be discarded by a video player after transferring content over the limited air interface, resulting in undesirable waste of resources. To improve OTT video content delivery, we strengthen the 4G architecture evolved with our selective packet discarding mechanism to mitigate the misuse of network resources. We also designed a dynamic QoS algorithm to improve video QoE.

## VII. Conclusions

This paper explored and analyzed the two most popular HTTP-based video streaming services (YouTube and Netflix) on mobile devices (iOS and Android) over three wireless networks (Wi-Fi, 3G and LTE). After performing many experiments, we point out that the network traffic behavior of playing videos on mobile devices depend on hardware performance, video players running on the devices, and network conditions. While delivering a video to a client over HTTP, we also observed that a noticeable amount of video content gets discarded without being stored in the video playout buffer, after the successful delivery to the client device. The discarded video content occurs when a TCP connection is repeatedly terminated and established. In such cases, the video packets that arrived at the client through the terminated TCP connection get discarded.

To reduce the waste of network traffic and enhance video QoE for end-users, we propose a dynamic QoS-aware video streaming architecture in LTE. Based on feedback on network conditions over the air interface, P-GW is designed to assist a video player in selecting a proper resolution under fluctuating network conditions, by dynamically throttling the maximum allowable TCP throughput on the video streaming flow. By monitoring TCP/IP and HTTP packets in real time, it also enables to discard the unnecessary video packets in advance before being delivered to the client. Our experimental results show that the proposed solution can save significant downlink bandwidth (up to 20.58% improvement) over the air interface, and provide a better viewing experience on mobile devices. We believe that our proposed platform can support network operators to resolve inefficiency of today's OTT video streaming with minimum investment.

## References


[1] Cisco, "Visual Networking Index: Forecast and Methodology, 2012-2017," Cisco, Tech. Rep., May 2013.

[2] Citrix, "Bytemobile Mobile Analytics Report," Citrix Systems, Tech. Rep., Jun. 2013.

[3] K. Balachandran, D. Calin, E. Kim, and K. Rege, "Clearmedia: A Proxy-based Architecture for Streaming Media Services over Wireless Networks," in *Personal, Indoor and Mobile Radio Communications, PIMRC. IEEE 18th International Symposium on*, Athens, Greece, Sep. 2007.

[4] L. Li, Z. Mao, and J. Rexford, "Toward Software-Defined Cellular Networks," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, Darmstadt, Germany, Oct. 2012.

[5] A. Jurgelionis, J. Laulajainen, M. Hirvonen, and A. Wang, "An Empirical Study of Netem Network Emulation Functionalities," in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, Maui, Hawaii, Jul. 2011.

[6] Y. Pourmohammadi-Fallah, S. Zahir, and H. Alnuweiri, "A Fast-Start Rate Control Mechanism for Video Streaming Applications," in *Digest of Technical Papers. International Conference Consumer Electronics, ICCE*, Las Vegas, NV, USA, Jan. 2005.

[7] 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.

[8] F.-L. Andrew, "A Review of HTTP Live Streaming," Jan. 2010, unpublished.

[9] Netflix Tchnical Center. [Online]. Available: https://contactus.netflix.com/Help/

[10] 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.

[11] 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.

[12] Iperf. [Online]. Available: http://iperf.sourceforge.net/

[13] 3GPP, "3GPP TS 23.203 v8.3.1, Technical Specication, Policy and charging control architecture (Release 8)," 3GPP, Tech. Rep., Sep. 2008.

[14] MathWork - LTE System Toolbox. [Online]. Available: http://www.mathworks.com/products/lte-system/

[15] OPNET - application and network performance. [Online]. Available: http://www.opnet.com/

[16] The netfilter.org project. [Online]. Available: http://www.netfilter.org/

[17] 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.

[18] 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.

[19] 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.

[20] S. Alcock and R. Nelson, "Application Flow Control in YouTube Video Streams," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, Apr. 2011.

[21] 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*, Boston, Massachusetts, USA, Nov. 2012.

[22] T.-Y. Huang, R. Johari, and N. McKeown, "Downton Abbey Without the Hiccups: Buffer-based Rate Adaptation for HTTP Video Streaming," in *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, Hong Kong, China, Aug. 2013.