YoMo: A YouTube Application Comfort Monitoring Tool

Barbara Staehle, Matthias Hirth, Rastin Pries, Florian Wamser, Dirk Staehle
University of Würzburg, Institute of Computer Science, Würzburg, Germany
{bstaehle,hirth,pries,wamser,dstaehle}@informatik.uni-wuerzburg.de

ABSTRACT

Out of the large number of multimedia content sharing platforms YouTube is the most popular one. This is reflected by the large number of studies which focus on analyzing YouTube characteristics. Techniques for quantifying the instantaneous YouTube QoE and predicting an imminent QoE degradation have in contrast never been proposed. The latter task is even more important if network management actions shall be carried out to avoid a YouTube QoE degradation. In this work we describe YoMo, a tool which constantly monitors the YouTube application comfort. This measure quantifies the application operation condition and allows a QoE prediction. Experiments show that YoMo is able to exactly anticipate an upcoming YouTube QoE degradation.

1. INTRODUCTION

According to the 2009 Cisco Visual Networking Index [3], 30% of all customer Internet traffic consists of downloading or streaming videos. According to Cisco Systems, this share will increase to over 60% by the year 2013. For the case of the U.S., roughly half of this Internet video traffic is due to user-generated content whereof again roughly the half is due to YouTube. Consequently, many authors try to find reasons for the success of YouTube. The most prominent example of this category of studies is the work of Cha et al. [1] who analyze the distribution and evolution of the YouTube video popularity and user behavior. Cheng et al. [2] compare the characteristics of YouTube videos and the structure of the underlying social network. Gill et al. [7] examine YouTube usage patterns, file properties, and transfer behaviors.

In contrast, the QoE of YouTube has to the best of our knowledge not yet been in the focus of research studies. The large number of QoE models for video streaming [6] are not applicable, as they assume UDP as transport layer protocol. In this case, delay, jitter, and packet loss are directly related to the video QoE as they cause artifacts or missing frames. YouTube videos, in contrast, are transported via HTTP over TCP. No packets are lost or delivered out of order, and the only quality degradation which may be caused by the transmission, is a stalling of the video. For this case, the approach of Gustafsson et al. [8] would be applicable which allows to derive the QoE from video parameters, the packet loss rate, and the number of buffering events after the video has been terminated.

The YouTube player has no feedback loop which adapts the video playback to the network conditions. Our goal is therefore to use the YouTube QoE as an input for a network management tool which allows to maintain the user QoE. Therefore, we need to know the user satisfaction in real time. Moreover, a management tool needs a prediction of the QoE, i.e. it needs to be notified if the YouTube player is about to stall in order to avoid this. Dalal et al. [4] already proposed a QoE prediction mechanism UDP video streaming. It however uses lost and retransmitted packets only, a method which does not work in case of TCP streaming.

In this work we introduce the YouTube monitoring tool YoMo. It constantly monitors the YouTube application comfort (AC) which is simply the amount of playtime buffered by the YouTube player. In general, AC characterizes how well the application is doing. This allows YoMo to derive a simple binary YouTube QoE in real time which is either "good" if the video plays and "bad" if the video stalls. Additionally, YoMo is able to predict a QoE degradation, namely the stalling of the video. In an earlier study [5] on QoE-based radio resource management, we showed that a cooperation of YoMo and a bandwidth shaping tool allows a continuous YouTube playback in a congested wireless mesh network.

The lack of literature on YouTube QoE is the reason why we think that YoMo is interesting for more areas than network monitoring. Therefore, we discuss the idea of using the AC to predict the QoE and its implementation by YoMo in the following. Section 2 gives an overview on the core idea and implementation details. YoMo's functionality is evaluated in Section 3. Section 4 summarizes the contributions of this paper and gives an outlook to future work.

2. MONITORING THE YOUTUBE AC

To be able to monitor the YouTube AC, YoMo has to fulfill several tasks: Firstly, it has to detect that a YouTube flow exists which has to be monitored. Secondly, it has to collect as much information as possible about the YouTube

This work was funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Förderkennzeichen 01 BK 0800, GLab). The authors alone are responsible for the content of the paper.

flow and thirdly, it has to monitor the YouTube AC. To make our approach more easy to understand, we first of all analyze the technology behind YouTube in Section 2.1, before we introduce the main ideas of YoMo and their implementation in Section 2.2. How the the amount of buffered playtime is estimated is described in detail in Section 2.3.

2.1 The Technology Behind YouTube

The YouTube player is a proprietary Flash application which concurrently plays a Flash video (FLV) file and downloads it via HTTP. At the beginning of this so-called pseudo streaming, the client fills an internal buffer and starts the video playback as soon as a minimum buffer level, γ, is reached. During the time of simultaneous playback and downloading, the buffer grows as long as the download bandwidth is larger than the video rate and shrinks otherwise. If the buffer runs empty, the video stalls and the YouTube player state changes from "playing" to "buffering". This state is hidden to the normal user, but can be retrieved from the YouTube API by JavaScript or ActionScript.

Each YouTube video is encoded as an FLV file which is a container format for media files developed by Adobe Systems. An FLV file encapsulates synchronized audio and video streams. The header starts with an FLV signature and contains information about the tags in the body of the file. The tags encapsulate the data from the streams and contain information on their payload. This information includes the payload type, the length of the payload, and the time to which the tag payload applies. FLV files may also contain metadata encapsulated in a tag with a script data payload. The available properties depend on the software used for the FLV encoding and may include the duration of the video, the audio and video rate, and the file size.

2.2 The Main YoMo Functionality

The YouTube player opens a new TCP connection each time it downloads a new FLV file or if the user jumps to another time in the video. Each FLV file has a header with the FLV signature, the beginning of a new YouTube video flow is hence marked by this signature. YoMo runs at the client and parses all incoming TCP flows in order to detect this signature. Once a flow containing FLV data is recognized, the data is continuously parsed in order to retrieve the available meta information from the FLV file. Detecting the YouTube flow is thus easily done. The AC monitoring task is more complex and will be explained in the following.

The YouTube AC is defined as the buffer status of the YouTube player. This is simply the time, β, the player can continue playing if the connection to the server is interrupted. Fig. 1 shows β as the difference between the currently available playtime T and the current time of the video t. YoMo constantly computes and visualizes β in a GUI and checks whether β falls below an In such a situation, like the one depicted in Fig. 1, the QoE is still good, as the video is playing, but the AC is bad, as $\beta < \beta_a$ and the video is about to stall soon. Hence, YoMo predicts an upcoming stalling and has to notifiy a network management tool or decrease the video bandwidth in order to avoid this.

2.3 Estimating the Buffered Playtime

YoMo computes the buffered playtime as $\beta = T - t$. It decodes the FLV tags in real time, and hence exactly knows the currently available playtime T which is the time stamp

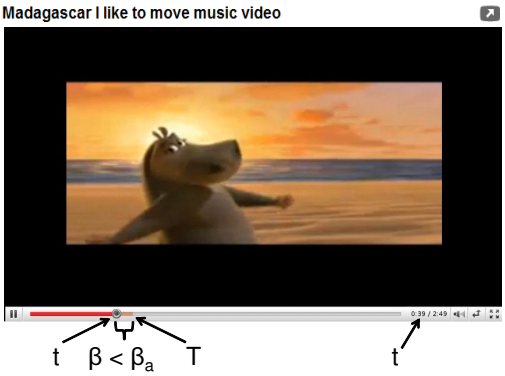

Figure 1: The YoMo Parameters

of the last completely downloaded tag. Intuitively, t could easily be calculated as the time difference between the actual time and the time when the player starts to play the video. During our measurements we found that this is not as easy as assumed. The reason for this is that the playback of a YouTube video does not start immediately after the player has loaded, but only after an amount γ of bytes has been downloaded. In [5], we show results from experiments with different videos and different connection speeds. The experiments reveal that γ is varying between 50 and 300 kB and is independent of the connection speed but is different for each of the 10 considered videos. We analyzed the coefficient of correlation between γ and different video characteristics including information about the frame types of the original H264 file embedded in the FLV tags, but were not able to find a clear correlation which allows to derive γ from the properties of the displayed video.

It is hence not possible to calculate the amount of time which lies between the time when the user issues the request for the video and the time when the video actually starts to play. We therefore implemented two different methods for calculating t which we discuss in the following. *Method 1* uses the assumption that the video starts to play as soon as the first FLV tag is completely downloaded. Clearly, this introduces a small error in the calculation of β which decreases however with an increasing connection speed. *Method 2* stands for the way of obtaining t from the YouTube player API which can be accessed by scripting languages only. In order to make YoMo applicable for the use in productive environments, it has to work with the original YouTube web page which can not be modified. It is also unrealistic to redirect all YouTube traffic to a dedicated web page where scripts for YoMo are running. Hence, YoMo uses a Firefox plugin which runs a JavaScript that retrieves t from the YouTube player. The plugin additionally sends the actual value of t to YoMo.

3. THE YOMO ACCURACY

YoMo and the Firefox plugin may be downloaded from the G-Lab website[1]. In the remainder of this section, we investigate how exactly YoMo can predict a QoE degradation. For this purpose, a client is connected to the Internet via a proxy which is able to modify the connection speed. The proxy may also interrupt the connection and thereby cause a video to stall. The client does not access the original

[1]http://www.german-lab.de/go/yomo

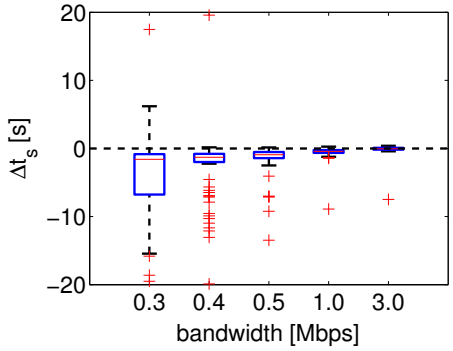

Figure 2: Stall Time Estimation Error, Method 1

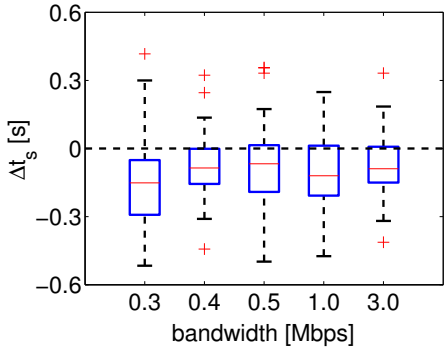

Figure 3: Stall Time Estimation Error, Method 2

YouTube side, but runs a measurement web page which embeds the YouTube player. This allows to dump the YouTube player state and thereby to get the exact stalling time. The client additionally runs YoMo which logs the estimated stall time which we consider to be the time when $\beta \leq 0.5$ sec. This is due to an experiment with 100 randomly chosen videos where observed that $\beta = 0$ sec is a sufficient but not a necessary condition for a stalling video as many videos already stall if $\beta \approx 0.5$ sec.

In Fig. 2 and Fig. 3, we depict the estimation error Δt_s between the time when YoMo considers the video to stall and the video actually stalls for Method 1 and 2 respectively. For each considered bandwidth, a box depicts the inter quartile range of the estimation errors and whiskers which are 1.5 times longer than the interquartile range. Values beyond this range are shown by red crosses. Let's discuss Fig. 2 which represents the estimation accuracy of Method 1, first. It shows that the error decreases with an increasing bandwidth. This is just a logical consequence of neglecting the time required for downloading γ, which gets smaller if the Internet connection is fast. While this method is thus sufficiently accurate for a broadband Internet access, it results in YoMo estimating the video to stall up to 20 seconds earlier as it actually did in the case of a slower connection.

The results for the experiment with estimation Method 2, shown in Fig. 3, in contrast visualize an bandwidth-independent error. Moreover does YoMo estimate the video on average to stall only roughly 0.1 sec earlier than it actually did, which is a significant improvement over Method 1. In most cases, YoMo underestimates the remaining play time, i.e. predicts the time of stalling earlier than it actually happened. The maximal estimation error in this direction is 0.5 sec. In some cases, YoMo overestimated the remain play time with a maximal error below 0.5 sec. Taking the inherent error of our assumption that a video stalls if $\beta < 0.5$ into account, these results demonstrate that YoMo, with Method 2 for the buffer estimation, is working as intended. In [5] we are moreover able to show that this accuracy is suitable for a QoE guaranteeing radio resource management.

4. CONCLUSION AND OUTLOOK

Application comfort monitoring presents an approach to monitor the usage of applications, their quality requirements and the experienced application comfort at the client. AC monitoring allows a QoE prediction which is very valuable input for network management tools. In the scope of YouTube, or more generally, Flash video streaming over TCP, the AC is measured as the buffered playtime. YoMo consists of a Java application and a Firefox plug-in. This plug-in monitors the state, in particular the current playtime, of the Flash player. A packet sniffer detects new Flash video transfers, extracts the videos metadata, and monitors the available playtime. Both components together allow to determine exactly the buffered playtime.

We were able to demonstrate that YoMo is able to accurately estimate the time when the YouTube player is stalling. YoMo is lightweight and easy to install while it provides valuable information to an ISP. If users run YoMo, both parties may greatly benefit as the provider gets information for free which it can use for improving the user QoE. Our future work will therefore be dedicated to examining the suitability of YoMo for QoE-based network management in various scenarios more closely.

5. REFERENCES

[1] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *SIGCOMM'07*, San Diego, CA, USA, October 2007.

[2] X. Cheng, C. Dale, and J. Liu. Statistics and Social Network of YouTube Videos. In *IWQoS'08*, Enschede, The Netherlands, June 2008.

[3] Cisco Systems Inc. Cisco Visual Networking Index - Forecast and Methodology, 2008-2013. White Paper, June 2009.

[4] A. C. Dalal, D. Musicant, J. Olson, B. McMenamy, S. Benzaid, B. Kazez, and E. Bolan. Predicting User-Perceived Quality Ratings from Streaming Media Data. In *ICC'07*, Glasgow, Scotland, UK, June 2007.

[5] B. Staehle, M. Hirth, F. Wamser, R. Pries, and D. Staehle, "YoMo: A YouTube Application Comfort Monitoring Tool," University of Würzburg, Tech. Rep. 467, March 2010.

[6] U. Engelke and H. J. Zepernick. Perceptual-based Quality Metrics for Image and Video Services: A Survey. In *NGI'07*, Trondheim, Norway, May 2007.

[7] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From the Edge. In *SIGCOMM'07*, San Diego, CA, USA, October 2007.

[8] J. Gustafsson, G. Heikkila, and M. Pettersson. Measuring Multimedia Quality in Mobile Networks with an Objective Parametric Model. In *ICIP'08*, San Diego, CA, USA, December 2008.