In-Network Support For Over-The-Top Video Quality of Experience

Francesco Lucrezia, Guido Marchetto and Fulvio Risso

Department of Control and Computer Engineering
Politecnio di Torino, Italy
Email: name.surname@polito.it

***Abstract*—This paper discusses the effects of the network congestion on the goodness of a streaming video session and proposes a solution that the network itself can adopt to recover from the possible Quality of Experience degradation. We consider a Broadband Access Network scenario, where congestion is most likely to occur in current communication networks. In particular, we concentrate on the Asymmetric digital subscriber line (ADSL) technology, which is predominant in the last-mile link toward the user machine. The proposed solution is based on an online heuristic evaluation of the mismatch between the bandwidth requirements of the video and the throughput actually offered in the bottleneck link. When a possible Quality of Experience degradation is observed, our tool reacts by limiting the concurrent traffic. The YouTube application is used as a reference throughout the entire paper due to its wide popularity. Moreover, High Definition videos are mainly considered as they are the most bandwidth demanding and hence the most sensitive to network impairments.**

Keywords–HTTP streaming; Quality of Experience; YouTube; TCP splitting.

I. INTRODUCTION

Since web-content providers started to make available a very huge amount of data, the Internet traffic is inevitably growing, especially the one related to multimedia applications. In particular, HTTP-based streaming traffic is growing steadily due to the increasing popularity of content providers such as Netflix, Hulu, and YouTube. Over-The-Top (OTT) videos embedded in the Internet applications can be watched by each individual equipped with an Internet connection and this also makes customers more and more demanding. For example, the spread of High Definition (HD) video retrieving is continuously increasing. For these reasons, the delivery of OTT content is one of the current challenges that network operators have to deal with.

One of the most important key points for the support of this kind of service is the provision of an adequate Quality of Experience (QoE) to the users. Namely, users have to be satisfied when watching the video, otherwise they are induced to switch to other providers for future content retrieves or, even worse, to suddenly leave before the end of the video. In both cases, this may cause a swift decrease of the provider revenues, e.g., due to advertisement fee losses. User satisfaction can be measured in several ways [1], ranging from the quality of the offered content to the experience offered by the deployed user interface. The former is clearly the most difficult to control as it actually depends on the network condition, which is variable in time and, in general, cannot be deterministically predicted.

Some solutions have been proposed [2][3], which target the mapping between the source of impairments and the final user QoE. However, they are closely related to the specific application under study. Specific solutions are then necessary for the HTTP-streaming case, which is the focus of our work. For such kind of application, the two main QoE metrics are the starting playback delay and the stalling events [4]. Hence, the main source of impairment to consider is the possible congestion of bottleneck links between the server and the clients, especially for the bandwidth demanding HD videos. In the current Internet, this problem might basically reside in the Broadband Access Networks providing the connectivity to final users, where the ADSL is predominant as last-mile technology. In these links, congestion might be due to the connection sharing among different users (e.g., tens in a small office sharing a single ADLS connection, as well as hundreds connected to different DSLAMs, or even both) or also when a single customer produces a lot of additional TCP traffic due to Internet applications that run automatically when host devices are connected to the network (e.g., BitTorrent file transfers).

Some solutions also exist for QoE provisioning in the OTT video streaming scenario, e.g., [5][6]. These all are effective, but require software modifications in the client machine, which is often unfeasible. In this paper we focus on an in-network solution, i.e., a countermeasure that the network itself can adopt to avoid OTT video QoE degradation. In other words, our solution is transparent to the video application, thus avoiding user intervention or client modifications. A trivial approach would clearly be the prioritization of video streaming traffic. However, this would be a static Quality of Service (QoS) solution — rather than one addressing users' QoE — and furthermore would suffer from the well-known starvation problem. At the same time, reservation-based techniques (e.g., [7][8][9] might be helpful in solving the problem, but these are not currently deployed in the Internet. Instead, the idea is to make use of the TCP splitting technique [10] in order to dynamically measure the throughput of the video session and possibly react to avoid stalls when it is lower than expected, thus really addressing a QoE problem. We consider the YouTube application as a reference in the rest of the paper as it is probably the most popular video content provider worldwide. However, it is worth noticing how the solutions described here might be easily extended to other HTTP-streaming services or, even more in general, to other TCP-based streaming applications (e.g., remote visualization tools [11]) as the main operating principles are similar.

The paper is organized as follows. Section II presents the YouTube service and a preliminary characterization work we performed on this application to specifically identifies its

 ISBN: 978-1-61208-377-3

internals, since they are continuously evolving. Section III discusses the proposed solution, while Section IV describes our testbed and reports on some experimental results. Finally, Section V concludes the paper.

II. YouTube

YouTube is a website created in 2005 and that rapidly became one of the most popular video-sharing application. It uses HTTP streaming for video content delivery (in particular, it recently adopted the Dynamic Adaptive Streaming over HTTP (MPEG-DASH) protocol), while it supports both Adobe Flash Player and HTML5 for video visualization at the client side. The transfer of the video content occurs by means of HTTP requests sent by the client for each chunk in which the entire video is divided according to the short-duration media segments (also called fragments) of the MPEG4 video format specifications. Concerning the service architecture, YouTube is based on a front-end Web server providing the home page of the website, while it relies on external resources for content delivery. In particular, when clicking on a link to watch a video, the web server redirect the first HTTP request to a specific video server selected according to the client position, RTT timing and other performance factors [12].

The main factors that characterize the transfer mode from a network perspective are the size of the data chunks, the frequency of the HTTP GET messages and the number of TCP connections used to transfer the chunks during the session. An extensive YouTube traffic characterization is available in literature (e.g., [13], [14]), which can provide an overview of such mechanisms. For example, it is well known that, like in other OTT services, video transfers start with a buffering phase during which a large amount of byte is downloaded by means of short-spaced HTTP GET messages and then proceed with a more smoothed download phase (i.e., the steady state) [15]. However, since YouTube internals are continuously evolving, we performed a characterization work to extract up-to-date information concerning the specific parameters that are of interest in our context. The Google Chrome Desktop browser is considered for this characterization work and in the rest of the paper.

The analysis of some capture files created by means of the tcpdump network analyzer led as to conclude that as soon as a request for a video is sent, the client opens a variable number from two to four different TCP connections with the server and the same applies for the entire duration of the video. This behavior is part of the logic that the application uses to react to a change in the network conditions or to a specific user action (e.g., when he pauses the video). In particular, the number of parallel TCP connections opened by the client plays a fundamental role in the observed performance. Indeed, it is the mean YouTube uses to counter the side effect of the ACK-driven congestion-control of the TCP protocol that causes the throughput lowering of a single TCP connection when dealing with traffic flows experiencing large RTT, packet losses, or congestions. In this way, the video session also becomes more robust and more aggressive with respect to the other competing applications that use a single TCP connection. It is worth noticing that the usage of parallel TCP connections is currently adopted also by several other OTT video applications.

Our analysis also pointed out that the number of chunks transferred by each connection, the size of the chunks, the amount of data downloaded during the buffering phase, as well as the precise pattern following by the variable bitrate in the steady state, strongly depend on the specific video considered and on its resolution. However, all videos analyzed had in common the fact that chunks are of two types: smaller chunks of dimension less than 500 KB and bigger chunks, larger than 1 MB. The same applies for HD videos with the only difference that the larger chunks can be bigger than 7MB for 1080p resolution and bigger than 4MB for 720p. A very common dimension (for any resolution) of the smaller chunks is 479232 bytes; this is an indication of the fact that chunks of smaller dimension actually belong to the audio stream and are transferred by one of the parallel connections opened by the system. Figure 1 details the specific behavior of two videos we used during our analysis.

III. In-network QoE support

Despite the abovementioned robustness of YouTube due to the utilization of parallel TCP connections, the presence of additional TCP traffic that contends for network resources might be a source of impairment. This might cause congestion over bottleneck links, with possible throughput decrease and consequent QoE degradation for video streaming users, especially when HD videos are considered. For this reason, our target is to guarantee an adequate bitrate to YouTube flows when they compete with other applications.

The specific problem statement is the following.

Problem statement. Given the bottleneck link capacity, the number of YouTube TCP flows and the number of competing TCP flows:

- Find the minimum bitrate that must be guaranteed for the streaming session in order to avoid stalling events (or a switching to a lower video resolution);
- Detect the instant in which congestion treats the video session;
- Act accordingly.

As stated in the previous sections, the bottleneck links in the current Internet infrastructure are in the access portion of the network. In particular, in this paper we consider a Broadband Access Network scenario, and the bottleneck links are then represented by last-mile links based on the ADSL technology.

We also point out that in the rest of this study we consider to know in advance which connections carry video streaming traffic and hence must be protected. The identification of these sessions is in fact a completely orthogonal problem and is outside the scope of the paper. Some possible solutions might rely on well-known traffic classification techniques (e.g., [16]) or further extensions of them.

A. Splitting the TCP connection

A TCP splitter [10] is a process able to intercept TCP connections and put itself in the middle of a communication. This is done by redirecting incoming SYN packets to a specific port where the splitter is listening (used to handle the connection with the sender) and then opening another TCP connection with the original recipient of the SYN packet. The splitter is responsible to pass data from one TCP connection to the other and vice versa, transparently to the end users. This

 ISBN: 978-1-61208-377-3

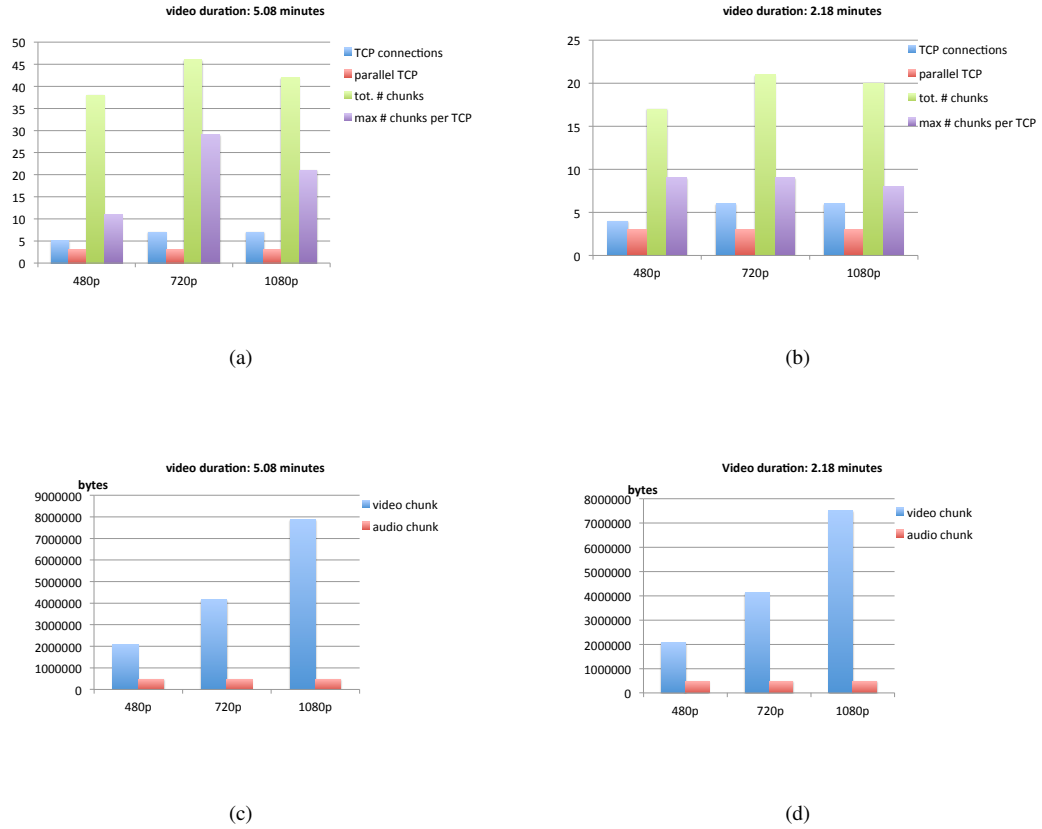

Figure 1. YouTube video characterization in terms of number of chunks, number of TCP connections, chunks per TCP connection and parallel TCP connections.

is usually adopted to increase TCP performance in the case of long end-to-end delays, which might reduce TCP overall throughput. The basic idea is that with two TCP connections instead of one, the weaker link has less influence on the whole path and the end-to-end segment established by the connection is broken into two segments shortening the overall RTT.

In our case, instead, the TCP splitter can be used for a totally different purpose. The idea is based on the fact that if the TCP splitter is located between the server and the bottleneck link — specifically, on the Broadband Remote Access Server (BRAS) of the ADSL system, the data received by the splitter from the server cannot be delivered toward the client at the same rate. In fact, since the receiver socket of the splitter continuously acknowledges packets coming from the server through a high-speed backbone network, the speed of data transfer remains higher than the speed in the bottleneck link. In this way, the splitter becomes the manager of a ”virtual” flow control driven by the bottleneck link and by the related speed with which the TCP socket delivers packets into this link. In a normal file transfer, the splitter receiver is faster than the splitter transmitter for the entire connection duration, because the file is by nature transferred at the maximum speed offered by the network. In a streaming video, instead, after the initial buffering phase (similar to a file transfer from this point of view), the download is throttled by a more sophisticated application logic, mentioned in the previous section. In particular, the server transmits only necessary data, i.e., the video chunks at the defined video bitrate, or a bit faster. Hence, in a given time unit, the amount of data received from the server should be the same of that sent to the client through the bottleneck link, even if at different speed. This is key to assure that the video is actually delivered with the expected bitrate to the client. If this condition is not satisfied, it means that the throughput offered by the bottleneck link is not sufficient to support that video bitrate.

This is the key point of our solution: if a process running on the BRAS is able to compare the throughput of the video session in the server-splitter link with that obtained in the splitter-client link, this process can infer when the video session is suffering and hence react to protect it, thus potentially avoiding stalls (or decreases in encoding quality in the case of DASH streaming). To better clarify these concepts, we report on some results we obtained in our emulated ADSL scenario, depicted in Figure 7 and better described in Section IV-A. Figure 2 and Figure 3 compare the video delivery pattern at an emulated BRAS with and without a TCP splitter running on it, when there is no congestion in the bottleneck link. In particular, Figure 2 shows the progress of received and transmitted bytes of a YouTube HD video at 1080p resolution into the emulated BRAS, without the splitter, while Figure 3

 ISBN: 978-1-61208-377-3

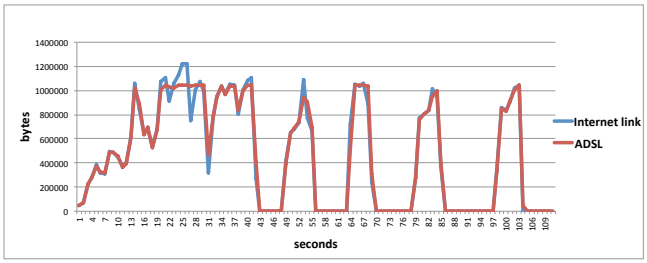

Figure 2. Video delivery pattern at the BRAS without TCP splitting

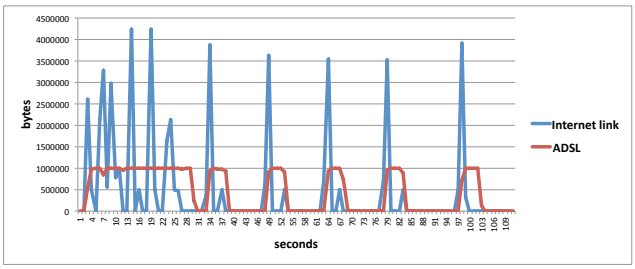

Figure 3. Video delivery pattern at the BRAS with TCP splitting

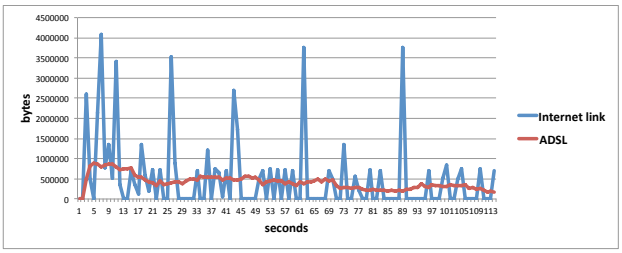

Figure 4. Video delivery pattern at the BRAS with TCP splitting and congested bottleneck link

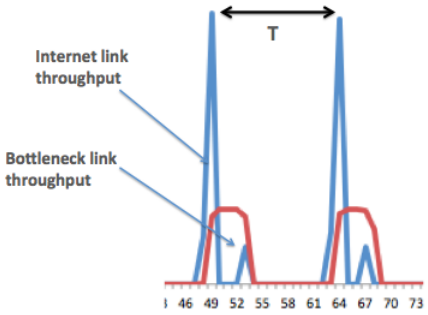

Figure 5. Heuristic

shows the progress of the same HD video with the action of the splitter. The emulated ADSL capacity is set to 7 Mbps in downstream and 400 Kbps in upstream. As said, the network is not congested.

The red line is the quantity of bytes injected at every second into the bottleneck link, while the blue line is the amount of byte received from the YouTube server.

The particular shape of the second pattern stems from the two main factors mentioned above: the peaks represent the download of each chunk characterizing the HTTP streaming, while the trend of the two series is due to the TCP splitting functioning, which permits to exploit the capacity of the fast Internet link by breaking the path of the TCP self-clocking mechanism within the network node. Notice the different maximum amplitude of the peaks between Figure 2 and Figure 3: without the splitter, the maximum amount of received bytes clearly depends on the bottleneck link capacity. In both cases, it is evident how, given a proper time unit (9 seconds in this specific case), the amount of bytes received from the server is equal to the one sent to the client, i.e., the application is not suffering.

Figure 4, instead, shows the video delivery pattern when the bottleneck link is congested. The video was the same of the previous graphs at the same resolution of 1080p. Notice how the traffic pattern significantly changes in the bottleneck link and how the amount of bytes forwarded to the client no longer follows in any way the pattern received from the YouTube server. In fact, we experienced two stalls during this experiment.

B. Video suffering detection

In order to detect when a video session is suffering and needs protection, it is necessary to keep track of the periodic throughput at the splitter. In particular, it is necessary to evaluate the difference between the number of received bytes from the server and the number of transmitted bytes to the client, in the same time interval. Since both the minimum required data rate and the correct time interval to consider cannot be known in advance, we propose a simple heuristic to determine such condition: the area described by the amount of received and transmitted bytes should be the same in the time interval defined by the spaced chunk download in the steady state phase of the video session (Figure 5). The idea is that if the area described by the received bytes is greater than the one described by the transmitted ones, it is likely that the bottleneck link is congested and cannot support the video bitrate of the session. When video suffering is detected, the system reacts by launching a QoS script (described in the next section) that protects the video session. The throughput evaluation has to start after that the buffering phase is finished. In our heuristic, this is detected by observing when the server stops transmitting for some seconds.

The heuristic relies on two vectors `rx[n]` and `tx[n]` where `n` is the discrete time in seconds. Figure 6 details this procedure. The variable `max_attempts` is introduced to be more conservative and avoiding to react in case of temporary congestion.

C. Corrective actions

After having realized that the video is suffering from a situation of congestion, the system should be able to react with some actions. At this point, it is possible to map our QoE problem to a simpler QoS one. In fact, we can control the bandwidth consumption on the link connecting the BRAS to the clients by means of proper scheduling and shaping algorithms applied at the network interface on that link. In essence, we can specify the bitrate to assign to a given streaming flow (which is dynamic and given by the `rx_thrgh` value described in the previous subsection) and

 ISBN: 978-1-61208-377-3

```
1: congestion = 0
2: Each second n:
3: if rx[n] >0 and rx[n − 1] == 0 then
4:     T ← n - n_0
5:     rx_thrgh = ( Σ_{i=n_0}^{n} rx[i])/T
6:     tx_thrgh = ( Σ_{i=n_0}^{n} tx[i])/T
7:
8:     if rx_thrgh >tx_thrgh then
9:         congetion++
10:
11:        if congestion >= max_attempts then
12:            launch QoS script
13:        end if
14:    else
15:        congestion = 0
16:    end if
17:    n_0 ← n
18: end if
```

Figure 6. Heuristic to detect video suffering

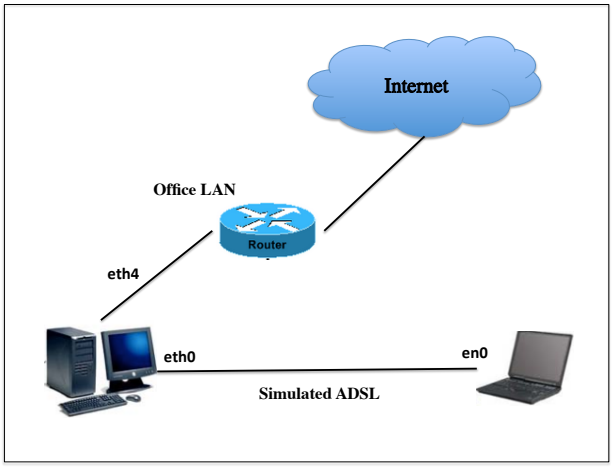

Figure 7. Laboratory environment

guarantee it through shaping and scheduling of the traffic. The algorithms for such a task are various and well-known. All have in common the definition of classes and a proper packet marking scheme, so that incoming packets can be put in the correct queue before being scheduled for transmission.

Although different solutions are possible, we adopted the Hierachical-Token-Bucket algorithm, a classful shaper and scheduler available in the Linux kernel that can efficiently provide bandwidth sharing. The required parameters are the IP addresses of the flows to control and the related rates to be assigned.

IV. Experiments

A. Test environment

In order to evaluate our solution, we realized a test environment in our lab, emulating an ADSL scenario. Figure 7 depicts this testbed. The set of elements involved are the following:

- A laptop acting as client PC.
- A desktop PC running the Linux operating system, which emulates a BRAS node.
- Proper tools to emulate the last-mile ADSL link characteristics: *ipfw* and *dummynet* in the client-side machine and the Linux module *NetEm* in the network node.
- A tcp-splitter written in C language, *pepsal* [17], installed in the emulated BRAS.
- The Linux *Traffic Control (tc)* tool for QoS actions

A client browser running on the laptop automatically generates both YouTube traffic and competing file transfer flows while a *dummynet* script is used to limit the upstream bandwidth on the emulated last-mile link. The desktop PC emulating the BRAS is a GNU/Linux machine with two Ethernet cards, one connected to the client PC and the other to the Internet through the high-speed access connection available in our University. At this node, *dummynet* is used for limiting the downstream bandwidth on the emulated last-mile link,

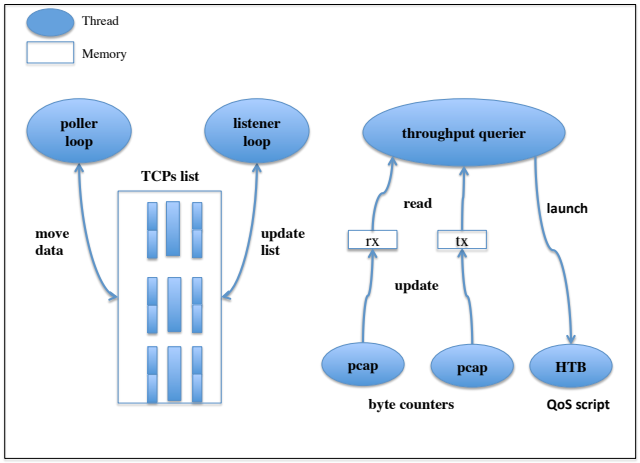

Figure 8. Main components of the tool

while a *tc* script is used for QoS actions.

We linked the TCP splitter, the heuristic video suffering detection, and the *tc* QoS script control in a unique, multithread program depicted in Figure 8. In essence, an ad-hoc thread is used for reading packets belonging to the streaming flows coming from the Internet toward the external interface of the splitter and outgoing to the client toward the internal interface. The cumulative number of bytes read from these two sniffers are saved in two global variable, `rx_bytes` and `tx_bytes`, and used by another thread that wakes up every second and stores in its memory space the instantaneous data rates. After reading the cumulative bytes from the interfaces, this thread resets the global variables to zero so that they become ready for the next sample. The sampling interval of 1 sec is selected as compliant with the YouTube dynamics, in particular with the idle time between chunks download in the steady state and the download time of each chunk, which involve some seconds.

 ISBN: 978-1-61208-377-3

TABLE I. EXPERIMENTAL RESULTS

	With tool	Without tool
No stalls	92%	0
Stalls	8%	100%

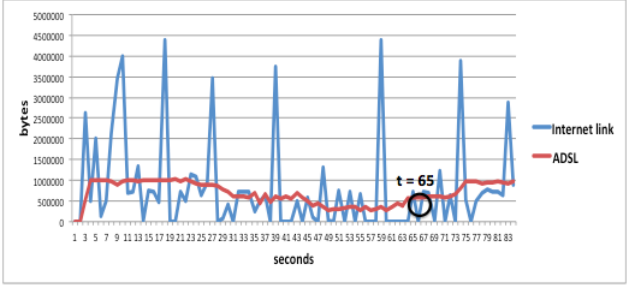

Figure 9. At istant t = 65 the process runs the QoS script

B. Results

In order to evaluate the effectiveness of our solution, we run some experiments considering several HD videos, which are the most bandwidth demanding and hence the most sensitive to the network congestion. In particular, we consider 50 different HD YouTube video sessions, established one at a time in the testbed. In each test, the selected HD video flow competes for the emulated ADSL link with additional file transfers. We analyzed the occurrence of video playback stalls with and without our tool running on the emulated BRAS. Table I reports on the results. First of all, we can see how congestion can significantly affect the perceived QoE of HD videos as at least one stall is observed in all the experiments when our tool is not active on the BRAS. Moreover, we can observe how our heuristic actually reacts to congestion situations in 92% of cases, avoiding stalls and thus increasing perceived QoE. For the sake of completeness, Figure 9 shows the video delivery pattern in a case where the video is suffering from congestion caused by two additional TCP connections; at instant $t = 65$ the reaction mechanism is triggered by our heuristic and it can be noted how the streaming transmission rate increases consequently, thus avoiding stalls.

For what concern the overhead caused by our tool, we have to consider that it is a splitter process working in user-space implementing the computation of the areas described by the download pattern over the time. The evaluation of the area is a simple sum of bytes counted every second. The major constraint is on the number of flows being able to analyze. Since the splitter works in user-space as a single process, the number of open sockets may be constrained by the machine operating system. Moreover, the main memory could be over-loaded before the splitter reaches the maximum number of opened connections. The limitation on the maximum number of opened sockets can be overcome by implementing the tool in the kernel-space. This might be considered for a possible commercial solution. The amount of resources involved depends on the number of flows to be tracked. In our tests we ran one session at a time and the resources consumed was negligible.

V. Conclusion and Future Work

The paper investigates a possible solution for protecting the overall video quality of an HTTP streaming service under the scenario of congestion caused by competing and concurrent TCP flows. The developed tool exploits the particular traffic pattern of HTTP-based videos, which is maintained at the entrance of the bottleneck link if a TCP splitter is used. By comparing incoming and outgoing patterns, the tool is able to detect when the video is suffering and reacts by protecting video downloads, thus limiting stalling events at the client. Our experimental results showed the effectiveness of the proposed approach, which was able to avoid video stalls in 92% of the considered cases. Possible future work regards the improvement of our heuristic method to detect video suffering, for example by also considering the pattern of HTTP GET messages flowing in the opposite direction, and in particular the time spacing among them during the steady state in both congested and uncongested scenarios.

References

[1] K. ur Rehman Laghari and K. Connelly, “Toward total quality of experience: A qoe model in a communication ecosystem.” IEEE Communications Magazine, vol. 50, no. 4, 2012, pp. 58–65.

[2] M. Fiedler, T. Hossfeld, and P. Tran-Gia, “A generic quantitative relationship between quality of experience and quality of service,” Network, IEEE, vol. 24, no. 2, March 2010, pp. 36–41.

[3] S. Jelassi, G. Rubino, H. Melvin, H. Youssef, and G. Pujolle, “Quality of experience of voip service: A survey of assessment approaches and open issues,” Communications Surveys Tutorials, IEEE, vol. 14, no. 2, Second 2012, pp. 491–513.

[4] R. Schatz, T. Hossfeld, and P. Casas, “Passive youtube qoe monitoring for isps,” in Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, July 2012, pp. 358–364.

[5] H. Hu, X. Zhu, Y. Wang, R. Pan, J. Zhu, and F. Bonomi, “Qoe-based multi-stream scalable video adaptation over wireless networks with proxy,” in Communications (ICC), 2012 IEEE International Conference on, June 2012, pp. 7088–7092.

[6] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, “Sdn-based application-aware networking on the example of youtube video streaming,” in Software Defined Networks (EWSDN), 2013 Second European Workshop on, Oct 2013, pp. 87–92.

[7] M. Baldi and G. Marchetto, “Pipeline forwarding of packets based on a low-accuracy network-distributed common time reference,” Networking, IEEE/ACM Transactions on, vol. 17, no. 6, Dec. 2009, pp. 1936 –1949.

[8] M. Baldi, M. Corra, G. Fontana, G. Marchetto, Y. Ofek, D. Severina, and O. Zadedyurina, “Scalable fractional lambda switching: A testbed,” Optical Communications and Networking, IEEE/OSA Journal of, vol. 3, no. 5, 2011, pp. 447–457.

[9] M. Baldi and G. Marchetto, “Time-driven priority router implementation: Analysis and experiments,” Computers, IEEE Transactions on, vol. 62, no. 5, 2013, pp. 1017–1030.

[10] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, RFC 3135: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations, Internet Engineering Task Force, Jun. 2001.

[11] G. Paravati, V. Gatteschi, and G. Carlevaris, “Improving bandwidth and time consumption in remote visualization scenarios through approximated diff-map calculation,” Computing and Visualization in Science, vol. 15, no. 3, 2013, pp. 135–146.

[12] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. Munafo, and S. Rao, “Dissecting video server selection strategies in the youtube cdn,” in Distributed Computing Systems (ICDCS), 2011 31st International Conference on, June 2011, pp. 248–257.

[13] 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 ’07, 2007, pp. 15–28.

 ISBN: 978-1-61208-377-3

[14] 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 '11. New York, NY, USA: ACM, 2011, pp. 345–360. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068849

[15] 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 '11, 2011, pp. 25:1–25:12.

[16] K. Takeshita, T. Kurosawa, M. Tsujino, M. Iwashita, M. Ichino, and N. Komatsu, "Evaluation of http video classification method using flow group information," in Telecommunications Network Strategy and Planning Symposium (NETWORKS), 2010 14th International, Sept 2010, pp. 1–6.

[17] C. Caini, R. Firrincieli, and D. Lacamera, "Pepsal: a performance enhancing proxy designed for tcp satellite connections," in Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd, vol. 6, May 2006, pp. 2607–2611.

 ISBN: 978-1-61208-377-3