Understanding the Intermittent Traffic Pattern of HTTP Video Streaming over Wireless Networks

(Invited Paper)

Ibrahim Ben Mustafa, Mostafa Uddin, Tamer Nadeem
Old Dominion University
Email: {iben, muddin, nadeem}@cs.odu.edu

***Abstract*—We are experiencing huge growth of video streaming traffic, which is creating big challenges for video providers in guaranteeing a satisfactory level of viewing experience to end users. Furthermore, the increase in video streaming demands on mobile devices over dynamic wireless links is creating another obstacle toward providing a high quality video service. In order to overcome most of these challenges, HTTPs adaptive video streaming technology was introduced, along with other great features for streaming videos. However, we found that HTTPs adaptive protocol can still suffer under certain situations and conditions. Mostly, these issues are likely experienced when multiple concurrent players compete over the same bottleneck. Several studies have proposed a network side solution at the home gateway or at the cloud aiming to assist the video players to maximize the viewing experience to all users sharing the same bottleneck. Although these proposed systems could provide some enhancements to the video streaming, they are unable to provide fine-grained monitoring and understanding of the video traffic to apply desire level of dynamic resource management. Considering the above issues, in this paper we conduct extensive analysis of the video traffic of YouTube; the most popular HTTPs adaptive video player. In our study, we argue that through a deep understanding and careful analysis of the HTTPs video traffic, valuable information about the competing streams can be obtained and could be utilized in developing a network based solution that can significantly improve the video QoE and assist the video players to perform much better.**

I. INTRODUCTION

With the exponential proliferation of smartdevices (i.e., smartphone, tablet, smartTV etc.), and the growth of entertainment and multimedia application, we are experiencing prominent increase in videos data traffic over wireless (i.e., WiFi and cellular) networks. As indicated by some studies, the amount of video traffic of YouTube and Netflix alone comprise 50% of the total traffic in the peak hours [1]. Such increase of video traffic is creating big challenges for video providers in guaranteeing a satisfactory level of viewing experience to the end users. In addition to the huge increase in the resources required to serve more streaming demands, the unstable nature of wireless links and the frequent changes in network loads create another obstacle toward providing a high quality video service.

In order overcome most of the above challenges, HTTP adaptive video streaming technology was introduced, along with other great features for streaming videos. When streaming videos over HTTP adaptive protocol, the video playing rate can be dynamically and seamlessly adjusted according to the change in the network condition. Moreover, to prevent a possible waste in the device and network resources in case that the user does not watch the entire video, the video with this protocol is periodically delivered in parts as the user continues watching. Moreover, the video now can be cached and delivered from any conventional HTTP web server and can easily traverse the NAT. As a result of these features, most of video providers nowadays such as YouTube, Netflix, and Vimeo have already adapted this technology in streaming videos.

However, despite the aforementioned features, practical implementation, represented in today's commercial players, reveals that viewing quality can still suffer with this protocol under certain situations and conditions. For instance, our measurements show that YouTube app can suffer from several serious issues that would directly impact the QoE of the end user including instability in the perceived quality, a long starting-up time, bandwidth underutilization, and unfairness between the users. Mostly these issues are likely experienced when multiple concurrent players compete over the same bottleneck due to the intermittent traffic generated by streaming protocol as indicated by several studies [2], [3], [4].

Having multiple users concurrently streaming videos through the same WiFi access point (at shopping center, coffee shop,..etc), or the same base station of cellular network is a very common scenario. Therefore, looking beyond an individual case, and make the optimization global (over multiple concurrent streams) is essential for an efficient streaming solution. To this end, several studies have proposed a network side solution at the home gateway or at the cloud aiming to assist the video players to maximize the viewing experience to all users sharing the same bottleneck [5]. Although these proposed systems could provide some enhancements to the video streaming, they are unable to provide fine-grained, scalable and dynamic resource management policies. More specifically, they lack monitoring and understanding the video traffic to apply a desired level of resource management. In addition, many of these proposed systems work by intercepting the HTTP requests and responses to collect information about the the streamed videos. Unfortunately, such monitoring is no longer valid as most of video providers have been switching to HTTPS with the goal of preserving their users' security and privacy. For instance, YouTube and Vimeo have already encrypted their traffic, while Netflix has announced to switch to HTTPs by the end of 2016 [6].

Considering the above issues, in this paper we conduct extensive analysis of the video traffic from one of the most popular HTTP adaptive video player to answer the following two questions: is there still a way to assist and optimize the performance of the commercial video players even with an encrypted traffic? And, how this can we effectively assist the video players? In our studies, we argue that through a deep understanding and careful analysis of the HTTPs video traffic, valuable information about the competing streams can be obtained and utilized in developing a network based solution that can significantly improve the video QoE and assist the video players to perform much better.

II. Overview of HTTP adaptive Video Streaming

There are several HTTP streaming technologies that are being widely deployed by the industry including Apple HTTP live streaming (HLS), Microsoft smooth streaming, Adobe HTTP Dynamic Streaming, and Dynamic Adaptive Streaming over HTTP (DASH). All these technologies follow nearly the same principle where the original content is processed into multiple bitrate profiles or versions, and then these versions are splitting into small segments, where each segment represents a short duration of playback time. These segments are made available for downloading on a conventional HTTP web server along with a manifest file, which describes the available bitrate profiles, the segments URLs, .etc, through sending a HTTP get request. Prior to the streaming session, the manifest file is typically provided to the client, and according to the client device capabilities and current network condition, the client requests the best segment version that fits its situation. Therefore, all the logic and decision is made by the video player at the client side leaving the server to only responding to the client requests. In this way, the client can dynamically adapt to the change in the network condition by adjusting the video bitrate to the end-to-end available bandwidth, which can have a significant enhancement on the performance . For instance, the video player can request a lower profile when it encounters a major drop in the available bandwidth to avoid possible stalls in the playback if it sticks with the same quality profile. One of the major challenges in designing the adaptive algorithm of the video players is the decision of requesting the segment version that maximizes the viewing experience. In fact, most of the work that have been done in the literature tried to enhance the performance of video players through designing a better adaptive algorithm.

Typically, HTTP video players have two main states: *buffering state*, in which the players tries to fill its buffer with video frames at the beginning of the streaming session, and *steady state*, in which the player starts periodically requesting video chunks after the buffer is filled up. The adaptive player typically maintains two thresholds, an upper and lower thresholds. The player pauses downloading video chunks as soon as the buffer filled and reached to the upper threshold, and it resumes downloading once the buffer drops to the lower threshold. Figure 1 illustrates these two states in which the player starts buffering video chunks at the beginning of the streaming session, and when the buffer reaches its max threshold at time 70, the player goes off and enters the steady state. At time 78, the player goes on again and send a chunk request to the server when the buffer goes below the minimum threshold. This behavior recur repeatedly until the video chunks are fully downloaded. These intermittent traffic pattern (ON/OFF periods) introduces a major

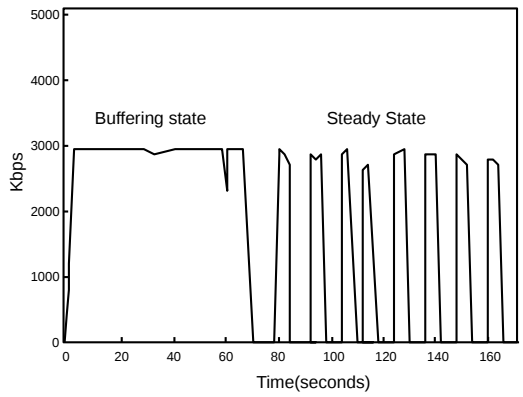

Fig. 1: Player state: buffering state and steady state.

challenge for competing video players to accurately estimate the available bandwidth. This pattern is the root cause of the degradation in the video quality for most commercial video players in the market today, thus it is very important to consider this traffic pattern in any future solution.

III. Experiment Setup

In the experiment, we use a laptop as Wi-Fi Access Point (Wi-Fi AP) which is running Ubuntu OS. This Wi-Fi AP is also connected to the internet using ethernet interface. In the Wi-Fi AP, we installed Open vSwitch (OVS) [7] and added the wireless interface (`wlan0`) of AP as a port with the OVS bridge. Consequently, all the traffic coming or going to any of the connected smartphones should pass through this OVS. In addition, we use Linux Traffic Control (`tc`) [8] of the Wi-Fi AP to control or limit the bandwidth of the video traffic. In the experiment setup, we have used three Android smartphones (two Samsung S5 and one Nexus 5) which are all connected with the Wi-Fi AP. In the experiment, we use `iperf` [9] to generate UDP traffic as a background traffic. In our smartphones, we have installed and used the latest version of YouTube app. This YouTube app comes with a *"stats for nerds* option that enable us to observe the quality requested by each player in addition to the buffer status and the estimated throughput value.

IV. Traffic and Performance Analysis

In this section, we start analyzing real video traffic streamed over HTTPs protocol for different network conditions and scenarios. We divide our analysis into two main scenarios: *non-competing* scenarios, where only one video player is streaming, and *competing* scenarios where two or more video players are competing for the bandwidth over the same bottleneck.

A. Non-Competing Scenario

In this scenario we capture and study the traffic patterns generated by only one video player without any competition from other players in the wireless network. We use different videos encoded with different bit-rates and streamed from a regular http server. For the first experiment, we stream a video with 480p quality resolution which encoded at 650kbps (selected manually among different available resolutions) with different bandwidth capacity to see its impact on the video traffic patterns. Note that, we use the `tc` tool in the Wi-Fi AP to control the available bandwidth of the video player running in smartphone. At the beginning we allow the video player to stream at 3200kbps, and then we reduce the bandwidth after 27 seconds to 1200kbps for 30 seconds before reducing it again to 650kbps. As we observe from figure 2, the duration of OFF period at the steady state shrinks as the throughput rate drops and getting closer to the video encoding rates till the OFF periods totally vanish at the last 30 seconds. Therefore, there is a strong correlation between the length of the OFF period and the video streaming and encoding rates. The existence of the OFF periods clearly indicates that the playback of the current quality profile is utterly stable and the likelihood of experiencing degradation in the viewing quality (e.g, switching to lower quality or stalling in the playback) is basically not possible as long as there is no drop in the throughput. On the other hand, the disappearance of the idle period of video stream as a result of a drop in the throughput may indicate either the throughput is equal to (or lightly above the encoding rate), or below the encoding rate. In the former case, the video playback operation will not be affected, while in the latter case, the video playback may or may not be affected depending on some factors such as the drop level, buffer condition, and length of the video. Therefore, it is extremely important for the performance to distinguish between these two cases and determine whether the drop in the bandwidth would affect the playback rate.

One technique for distinguishing between the two cases mentioned above is to determine the video average bitrate. Since the video traffic is encrypted, knowing the exact value of the video bitrate is not possible. However, the estimated value would be obtained if we can calculate the average chunk size and divide

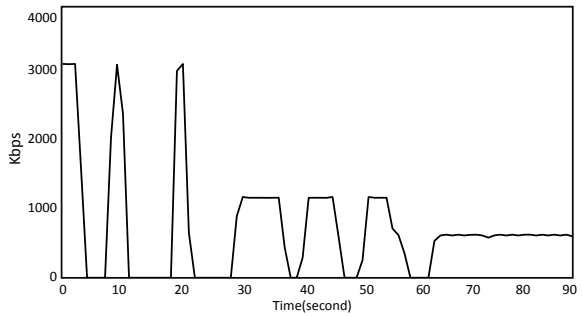

Fig. 2: ON/OFF periods correlation with streaming and encoding rates.

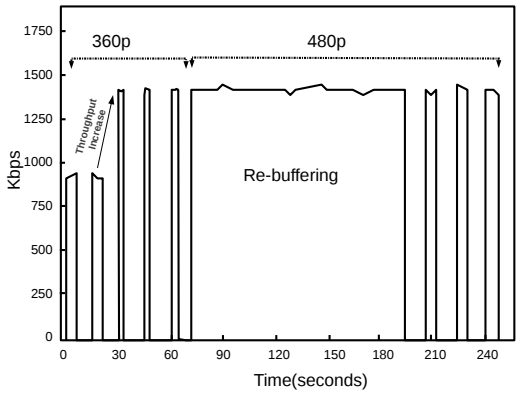

Fig. 3: Increase in the bandwidth causes a change in the traffic pattern.

it by the average duration of ON and OFF periods. In fact, this becomes quite possible with some new tools and technologies such as OpenvSwitch that allow for collecting statistical information about the traffic flows in real time. For example, we see in figure 2 that At second 57 the throughput drops from 1200kbps to about 650kbps which completely removes the OFF periods from the traffic patterns. Now by dividing the average chunk size (800KB) by the average length of both ON and OFF periods (10 seconds), the estimated bitrate will be 650kbps which is slightly above the actual value(600kbps). Therefore, the drop in this example can definitely affect the video traffic in long play, and a well-designed assistant system should react to prevent such possible degradation in the video view quality through some techniques discussed in section V.

Having this situation, it will be more efficient to estimate the number of seconds in the buffer at the time of the drop. This is very important for the performance of the overall streaming system. For instance, if we know an estimated value of the video encoding rate and know that the buffer has about x seconds of video frames at the time of the drop, then the system can infer when the buffer turns empty and start harming the viewing quality. In case the degradation can take long time, we can safely defer its intervention for quite long time waiting for the condition to be inherently improved (e.g, wait for one stream to finish downloading a video). As a matter of fact, each streaming application has its own setting for the buffer. For instance, our measurement reveals that YouTube app on all smartphones uses a 20MB buffer. This means that at the time of the drop, the player can continue playing the current quality for about 20MB divide by the estimated video bitrate. Thus when the available bandwidth drops below the encoding rate, we can have an estimated knowledge about when the player might switch the quality or be subjected to playback stalls.

We also identify another scenario that causes the idle periods to vanish. Figure 3 shows the intermittent traffic pattern of a video player streaming video of 400kbps bitrate (360p resolution)disappears for considerable period of time when the available bandwidth increased from 900kbps to 1400kbps. As can be observed from the figure, the increase in the throughput happen at second 30, and starting from second 68 the OFF periods are completely disappeared for nearly 140 seconds (persistent pattern) before showing up again at time 207. This could be interpreted as the player increases the quality and re-enters buffering state in which some of the low quality packets in the buffer are replaced by high quality packets. This conclusion can be confirmed by looking at the chunk size before and after the buffering state. We can clearly see from figure 3 that the video chunks streamed after the persistent period is much larger in size than the chunks streamed before. This change in the average chunk size is a clear sign of an increase in video quality as the chunks of a high quality profile typically have higher bitrate and thus size than the lower quality. Note that the OFF periods can vanish with no changes in the throughput, which indicates the client most likely jumps to time offset in the video playback.

While switching to the higher quality video, the player flushes all the buffered video packets of lower quality before starting to re-buffer higher quality video. Thus frequent switch of video quality causes a significant waste of resources in mobile devices as

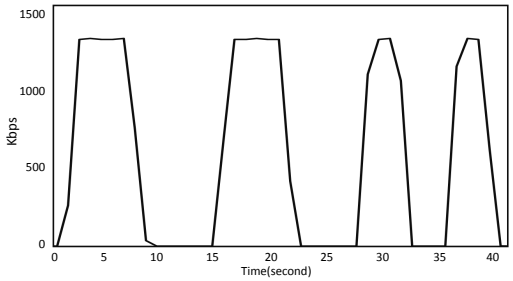

Fig. 4: The traffic pattern of high motion video.

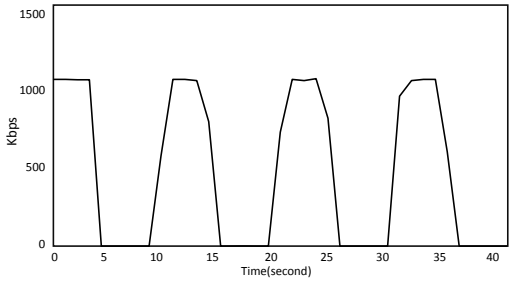

Fig. 5: The traffic pattern of low motion video.

well as increases the overall load on the server. In addition, such losses of packets over cellular network can become expensive to the user.

In videos, we see mixture of slow and high motion clips such as sport games or action movie, where in other videos, we only see slow motion clips such as news. Note that, the type of a motion clip in a video has direct impact on the video encoding rate. Similarly, the regularity of the ON/OFF period also depends on the variability of the video encoding rate. For example, in Figure 4, the first two chunks of the video is for high motion scenes, and have higher encoding rates and data size compare to the following two chunks with slower motion scenes. Thus the change between the high and slow motion clips changes the ON/OFF periods to have different length as in Figure 4. On the other hand, the video chunks represented in 5 have slow motion scenes with almost the same encoding rates and date size. Thus, in this case, we observe no changes in the length of ON/OFF periods.

B. Competing Scenario

In this section, we start analyzing the video traffic in more realistic scenarios, where multiple devices runs video streams with other concurrent flows in the wireless network.Figure 6 shows the flows pattern of two devices playing video streams over the same

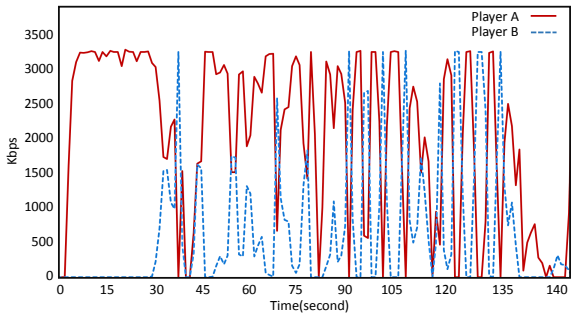

Fig. 6: Two video Players compete for bandwidth over the same bottleneck.

wireless AP. In order to ensure that both devices have exactly the same flow characteristics, we place both devices at the same distance from the AP and make the players requesting the same video. In addition, we generate background traffic in the wireless network using Iperf running on a third device to mimic real-life scenario and make the flows compete.

We start one player (player A), and after 30 seconds we start the other player (player B), so at the beginning the player A achieves high throughput, around 3200kbps, which enables high quality streaming profile. However, as soon as the competing flow of player B shows up, the throughput temporally drops to 1700kbps. Figure 6 shows an aggressive competition between the two flows which results in an extreme fluctuation in the throughput, as we can see, the player A is clearly getting much higher throughput in average than the player B, which is experiencing a very low throughput. This unfairness in sharing the bandwidth continues for about two minutes before eventually and slowly converse to a fair state as a result of using TCP as the transport protocol. This slow increase in the throughput of player B has a terrible impact on the QoE of the client. First, it forces player B to request low quality for a considerable amount of time. Second, the slow increase in throughput makes player B passes through all the quality levels before reaching to the final quality that fits with the fare portion of the bandwidth. This multiple switches cause player B to flush out most the packets from the buffer to replace them with the higher quality with every increase in quality. This would cause re-requesting a huge portion of the video content which can be a very expensive under certain scenarios.

We also examine the effect of competition between three players. We start running two players and after

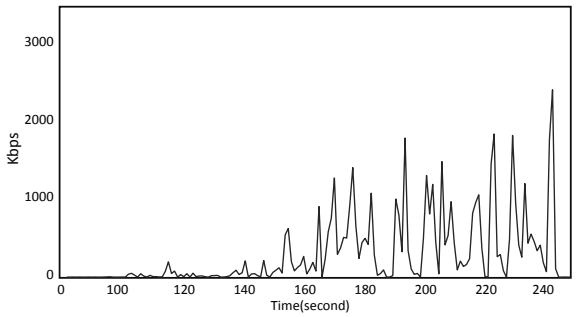

Fig. 7: The flow of one player competing with two other players in the wireless network.

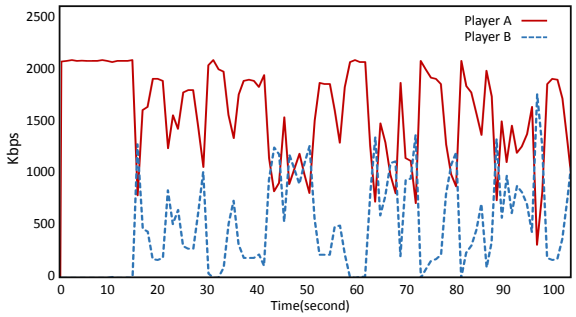

Fig. 8: The traffic pattern of two players streaming different video bitrates while competing over the same bottleneck.

80 seconds we start the third player. For clarity, figure 7 shows only the flow pattern of the third player. Similar to previous experiment, we see a slow increase in the throughput, and the third player also suffers from obtaining enough bandwidth to buffer its packets and starts the playback for its first 40 seconds. This in fact create quite a start-up delay before starting the playback. Under this circumstance, the user could get frustrated, and decide not to stream video over the network. Note that having a short start-up delay time is one of the most important metrics for the QoE. Therefore, it is very necessary to have a mechanism that insures a good performance for all players.

Figure 8 shows the impact of different video bitrates on the flow competition between two players, player A and player B. We disable the Youtube auto quality selection on both players, and manually set the quality levels at 720p (130kbps) for player A, and 360p (55kbps)for player B. We start both players at the same time under the same conditions (i.e., same video and same distance from Wi-Fi AP). Figure 8 shows that player A with higher bitrate stream wins the competition and dominates the bandwidth. This

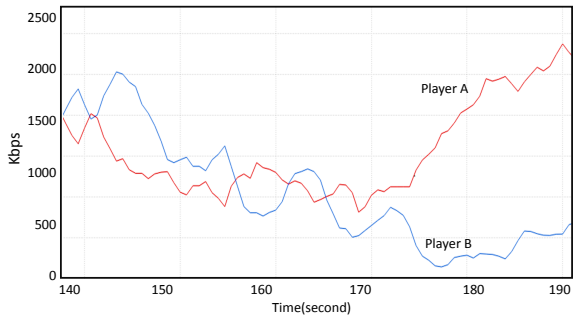

Fig. 9: The impact of the wireless link condition on throughput competition

explains, why player A in the first experiment has higher throughput than player B which starts later. The reason is that when player A starts, it gets enough throughput to request high quality video, while player B does not find much available bandwidth, thus end up requesting low quality video.

Typically, the wireless link conditions of different devices in the same network vary according to different conditions such as their distances from the AP. Figure 9 shows the result of an experiment in which we use two devices with different link condition, at the beginning, we start both players while placing both devices close to the AP, and then we slowly start moving one device away from the AP. As a result, the player of the moved-away device (player B) starts to observe throughput reduction around time 170s, and player B also looses the competition against player A which starts to experiencing higher throughput. Note that, in this case, we have two causes that are contributing to the suffering of the video streaming flow: the link condition and the competition among the video streaming flows. Although, we have no control on the former cause, but with smarter network management we can address the second cause to have acceptable video streaming experience for all players.

To understand the impact of the intermittent traffic pattern of HTTPs adaptive streaming protocol on the QoE, we study and analyze the traffic pattern of two players at the steady state. Figure 10 shows the smoothed throughput of these two players. Before second 350, both players were stable and achieving good throughputs, but after 350 second, we can observe a drop in their throughput for about 100 seconds followed by a dramatic increase in the throughput of player A and a major drop in throughput for player B.

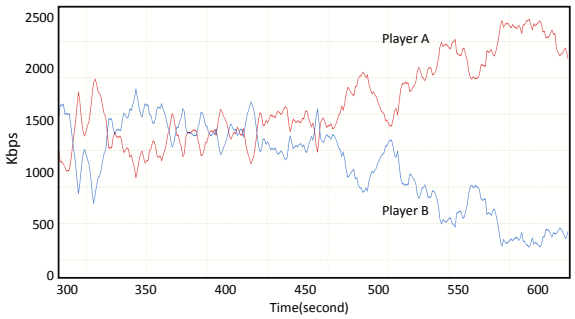

Fig. 10: The smoothed throughput averages of two video streams at steady state.

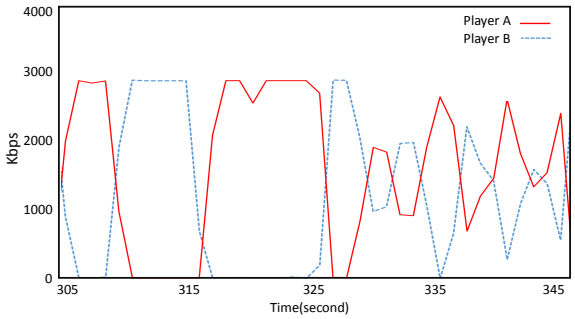

Fig. 11: The throughput competition of two flows at steady state.

To understand why player B looses the competition, Figure 11 zooms into the first 40 seconds of the previous figure. This figure shows how both player were utilizing the off period of each other for the first 25 seconds, and because of a long OFF period of player B (between second 315 and 325), player A experiences a huge increase in the bandwidth during that period. This increase in the throughput causes player A to switch to a higher quality profile and reentering buffering state, while player B looses its idle periods. Therefore, we can infer that player B was relaying on the player A's released bandwidth (at OFF periods) in maintaining the current quality level. As a result, both players start fighting again for bandwidth causing their throughputs to go below their current video bitrates. Consequently, their buffers start quickly draining, and because player A can slightly gain more bandwidth than player B in the competition, the buffer of player B get drained before player A. This causes player B to switch down three quality level at one time (from 720p resolution, encoded at 1200kbps, to 240p encoded at 400kbps as confirmed by examining the change in the player's playback resolution) in order to prevent stalls in the playback. This experiment highlights one main problem raised from the intermittent traffic pattern of HTTP adaptive players and confirms the need of a mechanism to enhance the performance.

V. Discussion

In previous sections, we show how to analyze and obtain some useful information about the HTTP video flows. In addition, we highlighted some performance issues with commercial players regarding concurrent video stream over the shared wireless network. As confirmed by our experiments, most of these issues are mainly caused by the aggressive competition between the players for the available network resources. Therefore, any future effort that aims to enhance the performance of video players and the viewing quality for all clients, should concentrate on mitigating this competition in such away that ensures the fairness among the player and, maximizes the utilization of the available bandwidth.

One possible technique, for instance, is to monitor and control the bandwidth utilized by each stream at the network edge. Each time a new stream joins the network, the system should instantly react and reallocate a fair portion of the bandwidth to the new stream. This technique might require reallocating the bandwidth, and distributing it again among all players. However, to avoid major degradation in the quality of the existing (old) players, the system should be aware of the conditions and status of all players and the characteristics of their flows before performing bandwidth reallocation. As we discussed in section IV A, the condition of the player can be estimated from the characteristic of its flow, thus we can use these in taking the bandwidth allocation decision.

Consider the scenario in figure 12, where two video streams are in the steady state. Assume a new video stream joins in with the previous two video streams. In the Figure 12, it is clear from the length of the OFF period and the size of video chunks that the *player B* is streaming at a much higher rate than its video average bitrate. When, cutting as much as half of the allocated bandwidth from the *player B* will not harm the QoE of its user, while cutting the same portion from the *player A* is likely to cause major degradation in its quality. Therefore, a well-designed system should reallocate the network resource based on the current condition of all video streaming flow in the network in order to maximize the overall QoE of all users.

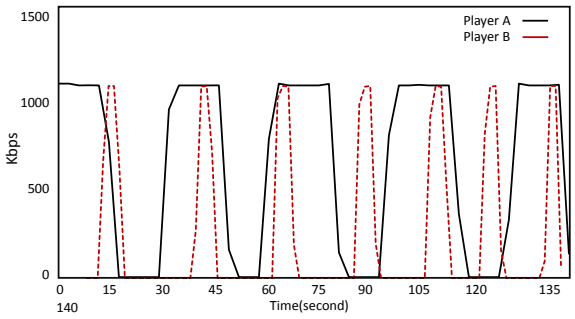

Fig. 12: Video stream flows of two video Players at steady state.

Generally speaking, different factors such as startup delay time, playback stalls, quality of the image, .etc, have different impact on the QoE. For instance, stalls in the playback has far more negative impacts on the QoE than reducing the quality one level. Therefore, it is important to consider the impact degree of these factors on the QoE for different clients when reallocating the bandwidth. For example, considering a scenario where we have two video streams; one is in steady state, and another is in buffering state. Assume at this moment, a new video stream joins the other two video streams. This would require reducing the available bandwidth for both existing streams to allocate some for the new comer. In this circumstance, however, it is better to reallocate some bandwidth from the steady-state video stream to the new video stream and keep the bandwidth at the same level for the buffering-state stream. Because, according to our analysis from previous sections, it is more likely that the steady-state video stream has more buffered video chunks to playback compared to the buffered-state video stream. Therefore, reducing the bandwidth for the steady-state stream may result, in the worst case, in reducing the quality level. On the other hand, reducing the bandwidth for the buffering-state video stream can lead to non-acceptable stalls in the playback. Our experiments shows that the main cause of such stalls with the commercial players is the use of weighted average in estimating the throughput instead of using the instantaneous value. One reason of using this technique is to minimize the impact of outliers resulting from the temporary drop or raise in the bandwidth.

VI. Conclusion

In this paper, we have conducted extensive analysis of the video traffic from popular HTTPs adaptive video player (YouTube). Initially, we have analyzed the video streaming traffic of non-competing scenarios, where we wanted to understand how the HTTPs adaptive video player behaves under different network conditions. Furthermore, we have analyzed multiple concurrent video streaming traffic for competing scenarios, where two or more video players are competing for the shared bandwidth. This study enables us to understand the unfairness in sharing the bandwidth among the players, and the its impact on the QoE. Finally, based on the analysis of video streaming flows, we have discussed how to enhance the video QoE and assist the video players to perform much better for all clients.

References

[1] 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*. ACM, 2012, pp. 225–238.

[2] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 97–108.

[3] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen, "Server-based traffic shaping for stabilizing oscillating adaptive streaming players," in *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2013, pp. 19–24.

[4] M. A. Hoque, M. Siekkinen, J. K. Nurminen, M. Aalto, and S. Tarkoma, "Mobile multimedia streaming techniques: Qoe and energy saving perspective," *Pervasive and Mobile Computing*, vol. 16, pp. 96–114, 2015.

[5] R. Houdaille and S. Gouache, "Shaping http adaptive streams for a better user experience," in *Proceedings of the 3rd Multimedia Systems Conference*. ACM, 2012, pp. 1–9.

[6] "Most internet traffic will be encrypted by year end. here's why."

[7] A. Unknown, "Open vswitch, an open virtual switch," *Date Unknown but prior to Dec*, vol. 30, p. 2, 2010.

[8] I. Delchev, "Linux traffic control," in *Networks and Distributed Systems Seminar, International University Bremen, Spring*, 2006.

[9] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," *htt p://dast. nlanr. net/Projects*, 2005.