What happens when adaptive video streaming players compete with Long-Lived TCP flows?

Koffka Khan
Department of Computing and Information Technology
The University of the West Indies, Trinidad and Tobago, W.I
Email: koffka.khan@gmail.com

Wayne Goodridge
Department of Computing and Information Technology
The University of the West Indies, Trinidad and Tobago, W.I
Email: wayne.goodridge@sta.uwi.edu.com

ABSTRACT

Competition among adaptive video streaming players severely diminishes user-QoE. When players compete at a bottleneck link many do not obtain adequate resources. This imbalance eventually causes ill effects such as screen flickering and video stalling. This scenario worsens when Long-lived TCP flows compete with the video flows. It is a known fact that adaptive video players perform poorly in the presence of Long-lived TCP flows. This work evaluates current heuristic adaptive video players at a bottleneck link in the presence of Long-lived TCP flows. Experimental setup includes the TAPAS player and emulated network conditions. The results show ELASTIC outperforms PANDA, FESTIVE and the Conventional players.

Keywords — adaptive video streaming, bottleneck, flickering, stalling, TAPAS, ELASTIC, PANDA, FESTIVE

I. INTRODUCTION

Adaptive video players are not able to get a fair share when coexisting with a TCP greedy flow [1]. TCP long-lived flows completely shut off TCP short-lived flows [21], [19], [29], and [34]. This causes performance problems for TCP short-lived flows, which generally carry interactive or delay sensitive data, such as video data/flows. TCP short-lived flows are becoming increasingly dominant in Internet traffic. This, together with competition from TCP long-lived flows, makes it an important area for adaptive video players. The outcome is the "downward spiral" effect. ON-OFF traffic patterns are the main contributor. Remedies include increasing segment size and filtering of bandwidth estimates. Examples of TCP long-lived connections are Internet chat and messaging (MSN, Skype). Device-to-device communication utilizes frequent "keepalive" messages. Devices transmit these messages periodically. Issues, such as, over-consumed network resources result. In addition, other issues occur during a TCP long-lived connection. These include TCP congestion and TCP connection recovery. Traffic features of TCP long-lived flows are specific to applications usage and their resulting characteristics.

TCP short-lived flows spend most of their time in the slow start phase. In this phase, the congestion window increases at a seemingly exponential rate. In contrast, TCP long-lived flows spend most of their time in the congestion avoidance phase. This phase utilizes the Additive Increase Multiplicative Decrease (AIMD) congestion control strategy (see Figure 1). According to [10] bandwidth sharing: TCP long-lived flows are shown to hurt TCP short-lived flows in terms of end-to-end delay and consequently throughput.

Figure 1: Congestion control and congestion avoidance. [source: https://hpbn.co/building-blocks-of-tcp/]

TCP long-lived flows occupy most of the buffer space from the sender’s end point. It creates huge queuing delays for TCP short-lived flows. Consequently, TCP short-lived flows only send few packets. TCP short-lived flows share many features with TCP long-lived flows, such as self-clocking, backing off and going to time out. However, TCP short-lived flows must try to utilize as much bandwidth as possible, when coexisting with TCP long-lived flows.

Figure 2: Multiple versus persistent connection.
We now consider an initial scenario setting, where three TCP long-lived flows pass through a bottleneck link. TCP long-lived flows are persistent connections (see Figure 2). The TCP long-lived flows are in the slow start phase, but quickly switches to the congestion avoidance phase and performs AIMD. The maximum congestion window is limited by the bottleneck link capacity. The sender now transmits multiple TCP short-lived flows. Congestion affects TCP short-lived flows since their window evolution is subject to TCP slow start rules. Thus, the TCP short-lived flows enter their slow start phases. Their congestion windows grow exponentially. However, before congestion is met, devices time out or terminate their flows. However, for TCP long-lived flows to time out, the overall throughput of all flows (TCP short-lived and long-lived) must exceed the bottleneck link capacity. Thus, the timing out of TCP-long lived flows occur when many packets are lost from the corresponding window of data [10].

In some cases, TCP is not able to fully utilize the transport or network layer resources. This happens because applications do not produce data fast enough. They produce small amounts of data at a relatively constant rate. This results in small bursts of packets. In extreme cases, applications produce single packets less than the maximum segment size of the connection. A typical example is the Skype [9] live streaming applications. Skype transfers data over TCP at a constant rate of 32 Kbit/s. Also falling in this category are applications utilizing permanent TCP connections and sending keep-alive packets (see Figure 3) during inactive periods. An example is Bit Torrent [3], [35], [14], which exhibits this behavior during choke periods.

In some cases, applications produce bursts of data, which become separate from each other by idle periods. Web browsing with persistent HTTP connections exhibits such behavioral characteristics. The user clicks on a link to load a web page. This causes a transfer period. He/she then, reads the page. This causes another idle period. He/she then, clicks on another link. This causes, yet another transfer period etc... These intermittent data traffic competes with video flows for bandwidth. Finally, in comparison to co-existing TCP flows, UDP flows utilize more than their fair share of the bandwidth [10] (see Figure 4).
sync biases in network state sampling, (2) Stateful bitrate selection: to compensate between biased bitrate and estimated bandwidth interaction, (3) Delayed update: to account for stability and efficiency tradeoff, and (4) Bandwidth estimator: to increase robustness to outliers.

Figure 5: Overview of the FESTIVE adaptive video player. [16]

The authors in [20], who proposed the PANDA algorithm, noted that since TCP throughput observed by a client would indicate the available network bandwidth, it could be used as a reliable reference for video bitrate selection. However, this is no longer true when HTTP Adaptive Streaming (HAS) [4] becomes a substantial fraction of the total network traffic or when multiple HAS clients compete at a network bottleneck. It was observed that the discrete nature of the video bitrates results in difficulty for a client to correctly perceive its fair-share bandwidth. Hence, this fundamental limitation would lead to video bitrate oscillation and other undesirable behaviors that negatively impact the video viewing experience. They offered a design at the application layer using a “probe and adapt” principle for video bitrate adaptation (where “probe” refers to trial increment of the data rate, instead of sending auxiliary piggybacking traffic), which is akin, but also orthogonal to the transport-layer TCP congestion control.

The authors illustrate a four-step model (see Figure 6) for an HAS rate adaptation algorithm: (1) Estimating: the

<table>
<thead>
<tr>
<th>The FESTIVE Player</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose randomized target buffer size</td>
</tr>
<tr>
<td>Randomized Scheduler</td>
</tr>
<tr>
<td>1. Compute reference rate</td>
</tr>
<tr>
<td>Increase/decrease rate as a function of bitrate</td>
</tr>
<tr>
<td>2. Cost-effective strategy to current reference rate</td>
</tr>
<tr>
<td>Harmonic B/W Estimation</td>
</tr>
<tr>
<td>Throughput</td>
</tr>
<tr>
<td>Chunks</td>
</tr>
<tr>
<td>Requests</td>
</tr>
<tr>
<td>time</td>
</tr>
<tr>
<td>Stateless/Delayed Bitrate Update</td>
</tr>
<tr>
<td>Stateful/Delayed Bitrate Update</td>
</tr>
<tr>
<td>3. Use harmonic mean over last 20 chunks</td>
</tr>
<tr>
<td>4. Harmonic B/W Estimation</td>
</tr>
</tbody>
</table>

At the beginning of each downloading step n:
1) Estimate the bandwidth share \( \hat{x}[n] \) by
\[
\hat{x}[n] - \hat{x}[n-1] = n(w - \max(0, \hat{x}[n-1] - \hat{x}[n-1] + w))
\]
2) Smooth out \( \hat{x}[n] \) to produce filtered version \( \hat{y}[n] \) by
\[
\hat{y}[n] = S(\{\hat{x}[m] : m \leq n\})
\]
3) Quantize \( \hat{y}[n] \) to the discrete video bitrate \( r[n] \in R \) by
\[
r[n] = Q(\hat{y}[n]; \ldots)
\]
4) Schedule the next download request via
\[
\hat{T}[n] = \frac{r[n]}{\hat{y}[n]} + \beta \cdot (B[n-1] - B_{\text{min}})
\]

Figure 6: PANDA four-step model. [20]

ELASTIC [7] proposes an approach (cf. Figure 8) that designs one controller that throttles the video level \( t \) to drive the playout buffer length \( t \) to a set-point \( q_t \). This eliminates the ON-OFF traffic pattern. The player is always in ON phase unless \( t \) is the highest level and \( q > Q_{\text{max}} \). The basic concept is based on the playout buffer model, design a feedback control system that computes \( l(t) \) to steer \( q(t) \) to a threshold \( q_t \). The received rate \( r(t) \), is considered as a (measurable) disturbance since it cannot be manipulated. ELASTIC provide a received video rate that oscillates around the fair share, with an increased number of video level switches. However, the main result involved long-lived TCP flows [37], where experimental evaluation showed that ELASTIC is able to get the fair share when competing with TCP long-lived flows.

Figure 7: The request-response timing between client and server in the Buffering and Steady states. [1]

1: On segment download:
2: \( \Delta T \leftarrow \text{getDownloadTime}() \)
3: \( S \leftarrow \text{getSegmentSize}() \)
4: \( d \leftarrow \text{isPlaying}() \)
5: \( q \leftarrow \text{getQueueLength}() \)
6: \( r \leftarrow l(S/\Delta T) \)
7: \( q_t \leftarrow q_t + \Delta T \cdot (q - q_t) \)
8: \( \text{return Quantize}(r/(d - k_pq - k_sq_t)) \)

Figure 8: ELASTIC Controller Pseudo-code. [7]
III. EXPERIMENTAL SETUP

The Controller code was written in python. TAPAS [8] an open-source Tool for rApid Prototyping of Adaptive Streaming control algorithms. TAPAS is a flexible and extensible video streaming client written in python that allows researchers to easily design and carry out experimental performance evaluations of adaptive streaming controllers without needing to write the code to download video segments, parse manifest files, and decode the video. TAPAS have been designed to minimize the CPU and memory footprint so that experiments involving a large number of concurrent video flows can be carried out. The player logs experimental data results. The TAPAS player communicates with the video server in the form of a GET request.

A virtual network is setup on the same host machine creating a custom emulation framework (see Figure 9). Our setup consists of client players, video servers, and a bottleneck link. The server resides on a Windows 10 machine. All experiments are performed on a Windows 10 client with an Intel(R) Core(TM)i7-5500U CPU 2.40GHz processor, 16.00 GB physical memory, and an Intel(R) HD Graphics processor. It serves video data to the client(s) who are on a Ubuntu operating system hosted on VMware. The virtual machine is allocated 12GB of physical memory. TAPAS is installed on Ubuntu 15.04 Linux. The TAPAS Adaptive Video Controller client makes different video segment bitrate level requests to the Apache server.

TAPAS allow multiple instances of the player to be created enabling multi-client scenarios. This work involves the interaction between adaptive streaming algorithm at the controller and TAPAS players. All traffic between clients and servers go through the bottleneck, which uses VMware settings which allow bandwidth limits to be set during the experiment. TAPAS support both the HTTP Live Streaming (HLS) and Dynamic Adaptive Streaming over HTTP (DASH) format.

The ten-minute-long MPEG-DASH video sequence “Elephant’s Dream” is encoded at twenty different bitrates, between 46 Kbps to 4200Kbps and five different resolutions, between 320x240 to 1920x1080, is used to run the experiments (cf. Table II). The video is encoded at 24 frames per second (fps) using the AVC1 codec. Fragment duration of 2s is used and is recorded in the mpd playlist accordingly. All the DASH files (.m4s fragments and .mpd playlists) are placed on the Apache server. We implemented three client-side algorithms in the TAPAS controller. The conventional approach is present by default and is used as a baseline in which to compare against other algorithms. TAPAS is lightweight in built, thus allowing the same receiving host to run a large number of separate video player instances at the same time at different command line interfaces. Thus, it allows the multi-client scenarios which are essential to the work in this paper.

The experiment considers a bottleneck link with two total video connections. The available bandwidth is set to b = 10Mbps for the two player experiments. QoE metrics are described as follows:

![Figure 9: Network testbed setup.](image)

i. The unfairness metric (for two players) is the average of the absolute bitrate difference between the corresponding chunks requested by each player (cf. Equation below, where p1 and p2 are player 1 and player 2, respectfully). The bitrate is the number of bits required to encode one second of playback.

\[
Unfairness = \frac{Average(\sum_{i=0}^{n-1}|r_{i,p1} - r_{i,p2}|)}{tu}
\]  

ii. The utilization metric is defined as the aggregate throughput during an experiment divided by the available bandwidth in that experiment (cf. Equation below, where \(tu\) is the throughput at time \(t\) and \(bw\) is the experimental available bandwidth).

\[
Utilization = \frac{\sum_{i=0}^{n-1}tp_i}{bw}
\]  

In the experiment (E2) the instability, inefficiency, and unfairness (different formulae used for the multi-player scenario) metrics, and re-buffering ratios is used to compare the performances of the considered algorithms.

i. Instability: The instability for player \(i\) at time \(t\) is given in Equation below, where \(w(d) = k - d\) is a weight function that puts more weight on more recent samples. \(k\) is selected as 20 seconds.

\[
Instability = \frac{\sum_{d=0}^{d=k-1}|r_{i,t-d} - r_{i,t-d-1}| * w(d)}{\sum_{d=0}^{d=k-1}r_{i,t-d} * w(d)}
\]  

ii. Inefficiency: The inefficiency at time \(t\) is given in Equation below. Consider \(N\) players sharing a bottleneck link with bandwidth, \(w\), with each player \(x\), playing a bit rate, \(b_{x,t}\), at time \(t\). A value
close to zero implies that the players in aggregate are using as high an average bitrate as possible to improve user experience.

\[
\text{Inefficiency} = \frac{\sum_{x \in \mathbb{R}_+} b_{x,t} - W_x}{w}
\] (8)

iii. Unfairness: Let \( J_{\text{Fair}}_t \) be the Jain fairness index (cf. Equation below) calculated on the average received rates, \( r_t \), (cf. Equation below) at time \( t \) over all players. The unfairness at time \( t \) is defined as \( \sqrt{1 - J_{\text{Fair}}_t} \). A lower value implies a fairer allocation.

\[
r_t = \frac{\text{downloaded bytes}}{\text{time interval}}
\] (9)

\[
JFI = \frac{\left(\sum_{i=1}^{n} r_t^2\right)^2}{n \sum_{i=1}^{n} r_t^2}
\] (10)

iv. Re-buffering ratio: is the ratio of the time spent in re-buffering and the total playtime of the stream Equation below.

\[
\text{Re-buffering ratio} = \frac{\text{total re-buffering time}}{\text{experiment duration}}
\] (11)

IV. RESULTS

We first present the level curves which represent the incoming bitrates of players, see Tables 1, 2 and 3. We observe ELASTIC outperforms the other players in all three experiments. This is because ELASTIC has only ON periods (no OFF periods are present) which enables it to aggressively compete for bandwidth against Long-lived TCP flows.

Table 1: Long-lived TCP flows occupying 1/3 of the bottleneck bandwidth capacity.

<table>
<thead>
<tr>
<th></th>
<th>FESTIVE</th>
<th>ELASTIC</th>
<th>PANDA</th>
<th>Conventional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>0.58</td>
<td>0.66</td>
<td>0.63</td>
<td>0.69</td>
</tr>
<tr>
<td>Unfairness</td>
<td>0.008</td>
<td>0.074</td>
<td>0.089</td>
<td>0.098</td>
</tr>
<tr>
<td>Re-buffering ratio</td>
<td>0.097</td>
<td>0.070</td>
<td>0.082</td>
<td>0.114</td>
</tr>
<tr>
<td>Instability</td>
<td>0.113</td>
<td>0.140</td>
<td>0.097</td>
<td>0.194</td>
</tr>
<tr>
<td>Average Quality</td>
<td>2.04</td>
<td>2.37</td>
<td>2.18</td>
<td>1.94</td>
</tr>
</tbody>
</table>

Table 2: Long-lived TCP flows occupying 1/2 of the bottleneck bandwidth capacity.

<table>
<thead>
<tr>
<th></th>
<th>FESTIVE</th>
<th>ELASTIC</th>
<th>PANDA</th>
<th>Conventional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>0.76</td>
<td>0.81</td>
<td>0.77</td>
<td>0.78</td>
</tr>
<tr>
<td>Unfairness</td>
<td>0.060</td>
<td>0.065</td>
<td>0.053</td>
<td>0.080</td>
</tr>
<tr>
<td>Re-buffering ratio</td>
<td>0.082</td>
<td>0.037</td>
<td>0.054</td>
<td>0.083</td>
</tr>
<tr>
<td>Instability</td>
<td>0.090</td>
<td>0.056</td>
<td>0.056</td>
<td>0.083</td>
</tr>
<tr>
<td>Average Quality</td>
<td>3.00</td>
<td>3.85</td>
<td>3.61</td>
<td>2.58</td>
</tr>
</tbody>
</table>

Table 3: Long-lived TCP flows occupying 2/3 of the bottleneck bandwidth capacity.

V. CONCLUSION

Competition among adaptive video streaming players severely diminishes user-QoE. When players compete at a bottleneck link many do not obtain adequate resources. This imbalance eventually causes ill effects such as screen flickering and video stalling. This scenario worsens when Long-lived TCP flows compete with the video flows. It is a known fact that adaptive video players perform poorly in the presence of Long-lived TCP flows. This work evaluates current heuristic adaptive video players at a bottleneck link in the presence of Long-lived TCP flows. Experimental setup includes the TAPAS player and emulated network conditions. The results show ELASTIC outperforms PANDA, FESTIVE and the Conventional players.

REFERENCES


AUTHOR DETAILS

Koffka Khan received the M.Sc., and M.Phil. degrees from the University of the West Indies. He is currently a PhD student and has up-to-date, published numerous papers in journals & proceedings of international repute. His research areas are computational intelligence, routing protocols, wireless communications, information security and adaptive streaming controllers.

Wayne Goodridge is a Lecturer in the Department of Computing and Information Technology, The University of the West Indies, St. Augustine. He did is PhD at Dalhousie University and his research interest includes computer communications and security.