Server-Based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players *

Saamer Akhshabi,
Lakshmi Anantakrishnan,
Constantine Dovrolis,
College of Computing
Georgia Institute of Technology
s.akhshabi, lakshmi3, constantine@gatech.edu

Ali C. Begen
Video and Content Platforms Research and Advanced Development
Cisco Systems
abegen@cisco.com

ABSTRACT

Prior work has shown that two or more adaptive streaming players can be unstable when they compete for bandwidth. The root cause of the instability problem is that, in Steady-State, a player goes through an ON-OFF activity pattern in which it overestimates the available bandwidth. We propose a server-based traffic shaping method that can significantly reduce such oscillations at the expense of a small loss in bandwidth utilization. The shaper is only activated when oscillations are detected, and it dynamically adjusts the shaping rate so that the player should ideally receive the highest available video profile while being stable. We evaluate the proposed method experimentally in terms of instability and utilization comparing with the unshaped case, under several scenarios.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Performance, Measurement, Algorithms

Keywords

Adaptive streaming, video streaming over HTTP, player competition, player oscillation, video shaping, stabilization

1. INTRODUCTION

The recent takeover of adaptive streaming over HTTP as the dominant technology for video delivery has not been without drawbacks. Recent studies have found some important performance issues with this technology [5, 7, 8, 9, 10, 11, 13]. The most alarming problem, in our view, is that *instability* can result when multiple players share the same network bottleneck, causing players to oscillate between different video profiles [5]. This instability can also cause bandwidth underutilization and unfairness between players. In this paper, we focus on the instability problem and propose a traffic shaping method to mitigate it.

Next we summarize the root-cause of the problem (see [5] for a longer discussion). An adaptive streaming player typically operates in one of two states: the Buffering-State, during which the player requests a new video chunk when the previous chunk has been downloaded, and the Steady-State, during which the player requests a new video chunk periodically, once per chunk duration (T seconds). During Steady-State, the player attempts to maintain a constant playback buffer size. Such periodic requests lead to an ON-OFF activity pattern in which the player is either ON, downloading a chunk, or OFF, staying idle. In parallel, the player estimates the network available bandwidth based on the per-chunk TCP download throughput, and it relies on that estimate to select the bitrate (video profile) for subsequent chunks [7]. If two or more video players share the same bottleneck link, and depending on the temporal overlap of their ON-OFF periods, they may both overestimate the available bandwidth. Consider, for instance, the case that the ON period of player X overlaps with the OFF periods of other players; player X will then overestimate the available bandwidth because its downloads do not share the bottleneck with other players. Player X would then decide to switch to a higher video profile, even though that bitrate may not be sustainable, resulting in congestion. In that case player X would switch back to a lower profile, causing oscillations. Fundamentally, the problem is that player X cannot correctly observe the available bandwidth when it is in the OFF state.

We propose a server-based traffic shaping method that aims to eliminate the root cause of the instability problem, i.e., to avoid the OFF periods during Steady-State. Specifically, when the server detects that a player oscillates between different profiles, it activates a *shaping module* (or "shaper") for that player. The shaper limits the throughput for each chunk to the encoding rate of that chunk; so, as long as the available bandwidth is higher than the shaping rate,

*This is an extended version of a paper with the same title that appeared at ACM NOSSDAV 2013 [6].

the download duration will be roughly equal to the chunk duration T. Consequently, the player will remain ON even when it operates in Steady-State.

Houdaille and Gouache [9] have also proposed a traffic shaping method, deployed at broadband links, to improve the stability, fairness, and convergence delay of adaptive streaming players. That work assumes that the bottleneck is always the broadband downstream link and so the shaping rate is determined based on that link's capacity and the number of active players. We do not make any assumptions about the location and capacity of the shared bottleneck or about the number of competing players. On the other hand, we assume that a player receives most successive video chunks from the same video server (so that the shaping module can maintain the required state for each player); this may not be true in the presence of caches or when the player requests different chunks from different CDNs/servers. Finally, our traffic shaping method is server-based, not requiring any cooperation from the client, and so it is entirely different than client-based adaptation methods such as AdapTech [7] or FESTIVE [11]. A comparison between the two approaches in terms of stability, efficiency and fairness is left for future work.

2. STABILIZATION METHOD

In this section, we present the proposed traffic shaping method. The shaper is not player-specific; it should work with any HTTP adaptive video streaming player. In the following, we first outline the basic idea behind the proposed shaping method and then describe in more detail each module of the algorithm. A flow chart of the algorithm is also shown in figure 1.

2.1 Basic Idea

The server's first task is to detect if a player is oscillating between different video profiles (See Section 2.3). In that case, the server activates the shaper for that player. The proposed scheme is *reactive* in nature: it reacts to oscillations instead of trying to prevent them. The reason is that shaping may have an execution overhead for the server, and so it should be activated only when necessary.

The shaper has two objectives: (i) to stabilize the player by eliminating the root cause of the instability problem, i.e., to avoid the OFF periods during Steady-State, and (ii) to allow the player to request the highest available profile that will not lead to oscillations. Without the second objective, a simple stabilization method would be to force the player to request the lowest available profile. That would of course be detrimental in terms of quality of experience (QoE).

The shaping rate is adjusted for each chunk (See Section 2.2). To eliminate OFF periods (or, in practice, to reduce them as much as possible) the shaping rate is equal to the chunk's encoding rate. For instance, if the chunk duration is $T = 2$ s, and the chunk's size is 2 MB, its average encoding rate is 8 Mbps. If the shaping rate is also set to 8 Mbps, and the network available bandwidth is more than that, the chunk will be delivered to the receiver at the shaping rate, and the download duration will be (approximately) T. In other words, *there will be no OFF period after the end of the download*. Without shaping though, TCP would attempt to transfer the chunk as fast as possible, potentially in much less than T seconds, creating an OFF period after the end of the download. As previously discussed, such OFF periods can cause bandwidth overestimation and oscillations, when the player is sharing the bottleneck with other adaptive streaming players.

After the shaper has stabilized a player, the available bandwidth may increase or decrease. New streams may start, existing streams may end, and other sources of traffic may appear or disappear. The server needs to allow for such variations and adjust the shaping rate accordingly. It turns out that a persistent available bandwidth decrease will cause congestion, and so it will trigger the player to switch to (and stay at) a lower profile than the shaping rate (See Section 2.5).

The opposite case is more challenging because a shaped player cannot detect that the available bandwidth has increased (its throughput is limited by the shaping rate). To deal with that possibility, the shaper is occasionally de-activated, and the server estimates the available bandwidth based on the TCP throughput of those unshaped chunks. If the available bandwidth has increased, the shaping rate is also increased accordingly (See Section 2.6).

The flow chart of the proposed stabilization method is shown in figure 1. The algorithm is executed upon receiving each chunk request and before sending the corresponding chunk.

2.2 Relation between Shaping Rate and Chunk Encoding Rate

Recall that the server aims to eliminate OFF periods by shaping each chunk so that its download takes T seconds. This can be accomplished, in the absence of network congestion, if the shaping rate is equal to the average encoding rate for that chunk. The encoding rate for a chunk can be determined at the server based on the size and duration of the chunk.

In practice, the shaping rate is set to a slightly higher value than the encoding rate. The reason is that adaptive streaming players are conservative, and they request a video profile of rate r only if the estimated available bandwidth is more than r/c, where c is slightly less than 1 (e.g., 0.9). Consequently, if the server shapes chunks exactly at their encoding rates, the player would keep downshifting over time. We refer to c as the *shaping slack parameter*; our earlier work suggests that commercial players often use a value of c around 0.8 [7]. Also considering TCP's inefficiency and other sources of noise, we set $c = 0.7$. Because of c, the download duration of shaped chunks is typically less than T, and thus there is a short OFF period after each download.

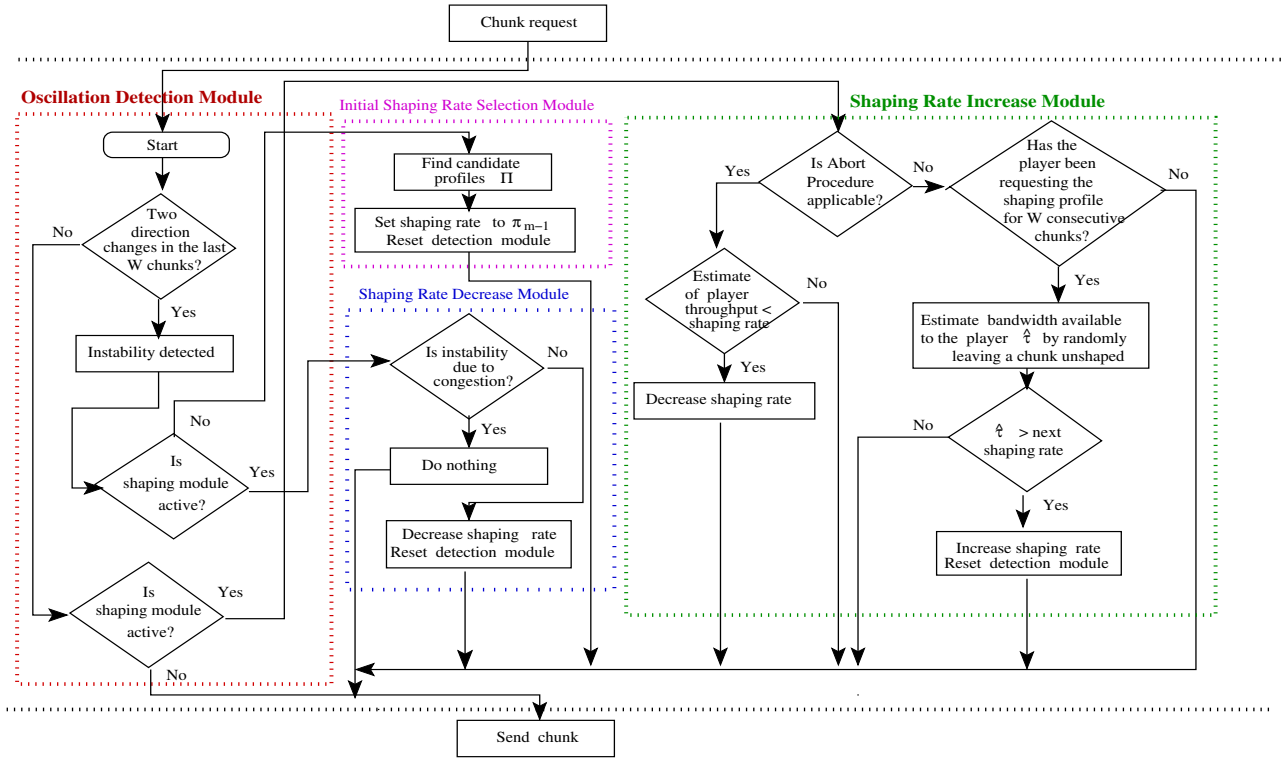

Figure 1: The flow chart of the stabilization algorithm which is executed upon receiving each request and before the corresponding chunk is sent.

2.3 Oscillation Detection

In the context of adaptive streaming, an oscillation occurs when the requested video profiles alternate between periods of upshifts and downshifts in relatively short time scales [5]. Figure 2 demonstrates this pattern for a typical player. The oscillation detection module aims to identify such oscillatory patterns.

The detector checks for changes in the direction of the player's requested profiles. A direction change is defined as a change in the requested profile (upshift or downshift) that is different than the last such change. Figure 2 shows two direction changes: from chunks $5 \rightarrow 6$, and from $9 \rightarrow 10$. When two or more direction changes occur within W successive chunks, an oscillation is detected and the player is flagged as unstable. These two parameters control the sensitivity of the oscillation detection module: increasing the number of required direction changes and/or reducing W, reduces the sensitivity of the detector. The parameter W is referred to as the *detection window size* and its default value is 30 chunks. The values of these two parameters were chosen such that we can detect almost all observed oscillations soon after their starting point.

2.4 Initial Shaping Rate Selection

To find the *highest sustainable profile*, the server considers a set of *candidate profiles*. This set includes

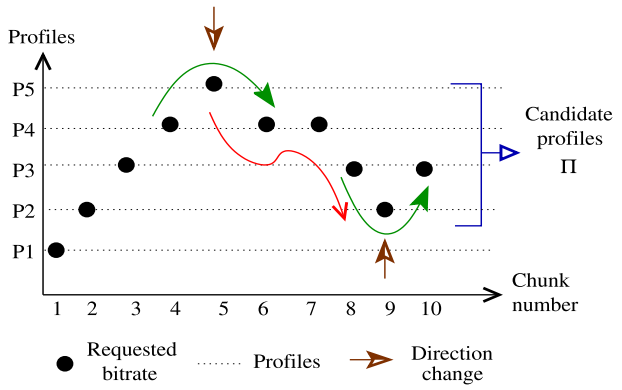

Figure 2: An oscillation in the time series of requested profiles by a player.

all profiles that have been requested between the last two direction changes, triggering the oscillation detection module to flag the player as unstable. Let us denote this set by Π; suppose that the set includes m profiles, with the i-th profile denoted as π_i. In the example of Figure 2, the instability is detected at chunk #10 and $\Pi = \{P_2, P_3, P_4, P_5\}$. The highest profile π_m is obviously not sustainable; otherwise the player would not switch to a lower profile. The shaper starts with a rather aggressive choice, setting the initial shaping rate to the next highest profile, π_{m-1}.

2.5 Shaping Rate Decrease

While shaping a particular player, the server keeps checking whether the player is still oscillating. If that is the case, the server needs to distinguish between oscillations due to OFF periods (they may still occur because of the shaping slack parameter) and oscillations due to short-term available bandwidth reductions (causing a drop in the requested profile followed by a gradual increase). In the former, the shaping rate should be decreased, while in the latter there is no reason to modify the shaping rate. To distinguish between the two cases, we rely on the following heuristic: The server identifies the most commonly requested profile P_q in the last W requested chunks before the last two direction changes. If those direction changes involve profiles that are less than P_q, we infer that the oscillation was due to a short-term drop in the available bandwidth, and we do not modify the shaping rate. Otherwise, the shaping rate is decreased to the one lower profile.

2.6 Shaping Rate Increase

When the available bandwidth increases, a shaped player would not be able to detect the increase because its throughput is limited by the shaper. To avoid this important problem, the server needs to occasionally estimate the network available bandwidth. To do so, after the player has been stabilized, the server de-activates the shaper for randomly chosen chunks (with 20% probability), and it measures the download throughput for those chunks.

These throughput measurements can be easily accomplished in an instrumented server. In our testbed, the server is running on a Linux stack (version 2.6.32-33) that allows applications to read the congestion window (*cwnd*) and smoothed round-trip time (*srtt*) TCP variables. So, we can estimate the connection's throughput as the ratio $\tau = \frac{cwnd}{RTT}$. In practice, we collect multiple samples of this ratio during the transmission of an unshaped chunk, and the throughput is estimated as the median of these ratios, denoted by $\hat{\tau}$. We have confirmed experimentally that this estimate closely tracks the throughput of the corresponding TCP connection.

If the estimated available bandwidth is higher than the shaping rate, the shaping rate is increased to the next higher profile (subject to the slack parameter c), as long as that rate is less than the available bandwidth. Note that an increase of the shaping rate does not mean that the player will automatically start requesting that next higher profile. That transition may take place several chunks later, depending on the player's adaptation algorithm.

Until the player makes the previous upwards transition, the server keeps measuring the per-chunk throughput. If the player does *not* switch to the higher profile, while the throughput of a chunk is less than a fraction ψ of the shaping rate, the server infers that the player cannot switch to the profile that corresponds to the current shaping rate. In that case the server returns to the previous, lower shaping rate. We refer to this corner-case of the algorithm as the *abort-rate-increase procedure.* In order to allow for some slack, we set $\psi = 0.875$.

3. TESTBED AND METRICS

The experimental testbed and setup is similar to that described in [7] and [5]. The competing players run on the same host. That host also runs a packet sniffer (Wireshark [14]) and a network emulator (DummyNet [17]). Wireshark allows us to capture and analyze the traffic from/to the HTTP server offline. DummyNet allows us to control the *downstream available bandwidth* (also referred to as *avail-bw* below). The client-host is connected to the Gigabit Georgia Tech campus network through a Fast Ethernet interface.

We use two different players in our experiments. The first is the *Simpler Player* introduced in [5], which is based on the open-source Adobe OSMF player [15]. We have instrumented that player to log internal variables such as playback buffer size, requested bitrate, chunk download time, and chunk throughput over time. The second player is the commercial Smooth Streaming player [16]. We infer the same variables for that player using packet captures, as described in [7].

The testbed includes a machine hosting the server. This host runs an Apache2 server running on a Ubuntu 10.04 LTS system, and is also connected to the Georgia Tech campus network through a Fast Ethernet interface. The traffic shaper is installed as an Apache2 *output filter module*. We modified the open-source `mod-bw` module to implement our shaping method. We use two different *content generator modules* in the Apache2 server: one for handling Adobe Dynamic Streaming requests (`mod-f4fhttp` [2]) and another for Microsoft Smooth Streaming requests (`mod-smoothstreaming` [4]).

For Adobe OSMF, the video content is obtained from a DASH dataset published by Lederer et al. [12] and packaged using Adobe's File Packager [1]. For Smooth Streaming, we use the "Big Buck Bunny" 720p content provided by Microsoft [3]. In the following, P_r represents a profile of r Mbps available for a given video stream at the server (e.g., $P_{2.75}$).

We define the following two performance metrics: (i) the *instability* metric is defined as the fraction of successive chunk requests by a player in which the requested bitrate does not remain constant, (ii) the *utilization* metric is defined as the aggregate throughput during an experiment (measured using Wireshark), divided by the avail-bw in that experiment.

4. DEMONSTRATION

In this section, we illustrate the function of every component of the proposed method with two simple experiments. We have performed many such experiments with the *Simpler Player* introduced in [5], and to a smaller degree, with the Smooth Streaming player. Due to space

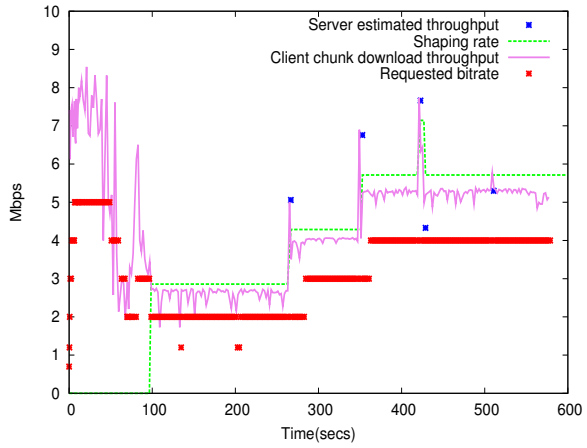

Figure 3: Requested bitrate, chunk throughput, and shaping rate for Player-1.

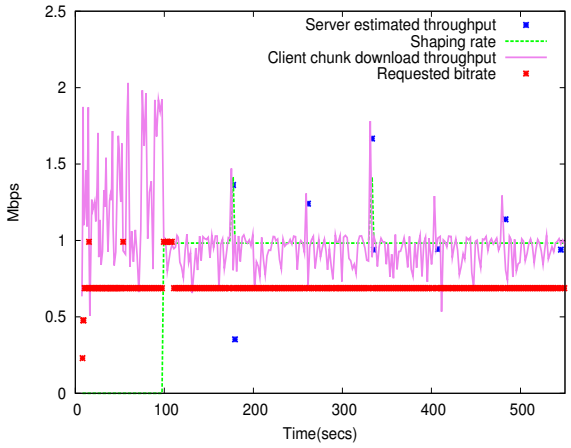

Figure 4: Requested bitrate, chunk throughput, and shaping rate for a Smooth Streaming player in an experiment with two competing Smooth Streaming players.

constraints, we only show here two of these experiments that highlight the most important aspects of the proposed shaping method: one with the Simpler Player and another using Smooth Streaming.

In the first experiment, the server provides six video profiles between 0.7 Mbps and 5 Mbps. The bottleneck capacity is set to 10 Mbps. Player-1 starts streaming first. After about 50 seconds, three more players start at about the same time. Two of these players terminate after 220 seconds. The remaining two players continue until they complete their streaming session six minutes later. We focus on Player-1.

Figure 3 shows time series for the requested bitrate, chunk throughput, and shaping rate for Player-1. During the first 50 seconds the player does not face any competition; it quickly switches to the highest available profile ($P_{5.0}$) and remains stable. After the three additional players join, the available bandwidth for Player-1 drops. This forces the player to decrease its requested bitrate. Even worse, the player becomes unstable because of the available bandwidth overestimation associated with OFF periods [5]. The instability results in two direction changes occurring at $t = 82$ s and $t = 99$ s, which fall within the detection window size W of 30 chunks. This is detected by the server, which activates the shaper. The set of candidate profiles is determined as $\Pi = \{P_{2.0}, P_{3.0}\}$. The server starts shaping the stream to the lower of these two profiles, $P_{2.0}$.

When two players terminate at $t = 279$ s, the available bandwidth for Player-1 suddenly increases from 2.5 Mbps to 5 Mbps. The opportunistic shaping rate increase mechanism becomes active at around the same time. The shaper leaves a chunk unshaped, estimates the new available bandwidth, and increases the shaping rate by one profile. The player reacts by increasing its requested profile to $P_{3.0}$ a few chunks later. A similar event takes place at $t = 349$ s when Player-1 switches to $P_{4.0}$.

There are two more interesting points in this experiment. First, at $t = 135$ s and $t = 205$ s there are two direction changes that are ignored by the shaper because they are recognized as short-term available bandwidth drops. Second, at $t = 420$ s, the *abort-rate-increase procedure* becomes activated after the shaper attempts to increase the shaping rate from $P_{4.0}$ to $P_{5.0}$ because the player is not able to receive the throughput that corresponds to the new higher shaping rate.

Note that Player-1 is in fact quite stable during this experiment and it does not show the frequent oscillations observed in earlier work with unshaped players [5]. Overall, this experiment's instability metric is 4.9%.

Next, we present an experiment with two competing Smooth Streaming players. Figure 4 shows similar time series for the requested bitrate, chunk throughput, and shaping rate for one player. The two players share a 2 Mbps bottleneck link. There are eight video profiles between 0.23 Mbps and 2.96 Mbps. After the streaming session starts, each player quickly switches to the highest sustainable profile $P_{0.68}$, given its fair share of the available bandwidth. However, due to OFF periods and the associated overestimation, the players occasionally switch to the next profile ($P_{0.99}$), which is not sustainable (considering the HTTP/TCP/IP headers overhead). The server detects this instability at around $t = 99$ s and shapes both players to $P_{0.68}$. The two players remain stable afterwards. We have repeated the same experiment without the stabilization module; in that case the players keep oscillating between the previous two profiles throughout the experiment.

5. RESULTS

In this section, we compare the performance of adaptive video players in terms of instability and utilization, between two cases: without traffic shaping and with the proposed traffic shaping method. We refer to the former as the *unshaped case* and the the latter as the *shaped case.*

The comparisons are done under three scenarios. First, we examine how the number of competing players affects stability and utilization, under a fixed bottleneck capacity. Second, we examine how a persistent TCP transfer affects the oscillations of multiple adaptive streaming players, and how effective the traffic shaper is in that case. Third, we examine how a mix of shaped and unshaped players compete for the same bottleneck. In all three scenarios, the experiments are performed using the Simpler OSMF-based player.

5.1 Number of Competing Players

Assuming that the capacity of the bottleneck link is fixed, how do the instability and utilization change as we vary the number of competing players?

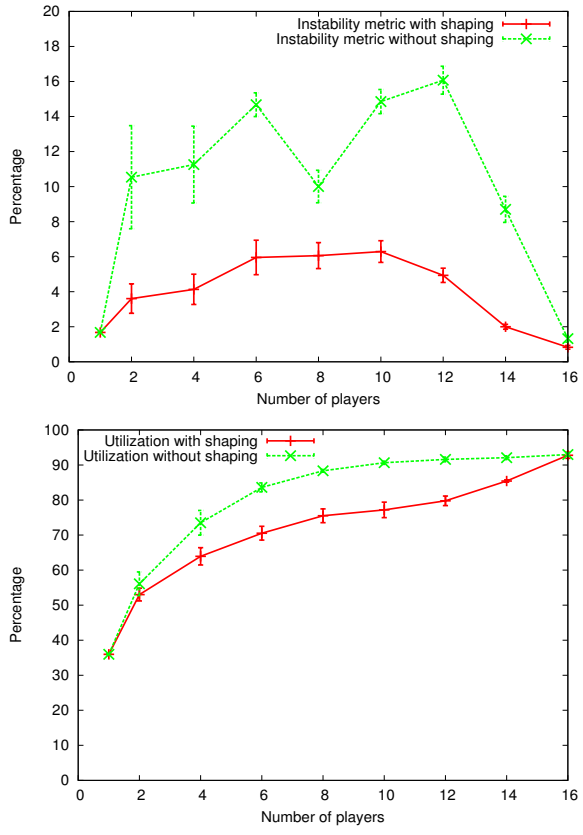

Figure 5: The instability and utilization metrics as a function of the number of competing players N for shaped and unshaped players.

Figure 5 shows the instability and utilization metrics as a function of the number of competing players, denoted by N, with shaped and unshaped players. The link capacity is 10 Mbps. Each point is obtained by repeating the same experiment between two to eight times. We show 95% confidence intervals.

In the case of unshaped players, we observe a similar instability pattern with the earlier work [5]. The instability metric peaks at some mid-range N value; the reason for this pattern has been discussed in detail in [5]. In the case of shaped players, the instability metric is significantly lower (See Table 1 for p-values using the t-test). Specifically, the instability metric drops to around 5%. It is hard to reduce this metric even more because the shaper is reactive: it is activated only after some oscillation is detected.

The results for the shaped case do not show the same trend as in the case of unshaped players. In particular, we cannot reject the hypothesis that the mean instability is constant as N is increased. This demonstrates that the effective operation of the proposed stabilization mechanism does not have a strong dependence on the number of competing players.

For most values of N, the utilization metric shows a statistically significant difference between shaped and unshaped players (See table 1 for p-values). However the difference in the actual utilization is not large. This implies that stabilization does *not* come at a big cost of utilization. We certainly do not need to stabilize the players by forcing them to use an overly low profile.

Metric	N=2	N=4	N=6	N=8
Instability	$< 10^{-4}$	$< 10^{-6}$	$< 10^{-16}$	$< 10^{-9}$
Utilization	0.09	$< 10^{-3}$	$< 10^{-8}$	$< 10^{-7}$
Metric	N=10	N=12	N=14	N=16
Instability	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-4}$
Utilization	$< 10^{-6}$	$< 10^{-9}$	$< 10^{-12}$	0.92

Table 1: P-values for the hypothesis tests ("equality-of-means" between shaped and unshaped players, using the t-test) of Figure 5.

5.2 In the Presence of a TCP Bulk Transfer

What happens when adaptive streaming players share the same bottleneck with a persistent TCP transfer? Does the performance of the adaptive streaming players improve when the stabilization mechanism is deployed at the server? We perform two sets of experiments, one with three shaped and another with three unshaped players. In both cases, they compete with a persistent TCP transfer, and the bottleneck link capacity is 10 Mbps. The TCP connection starts three seconds after the players start streaming. Each experiment is repeated four times.

Table 2 shows 95% confidence intervals for the instability and utilization metrics. The instability metric is much less in the shaped case (3.4% versus 10.7%). The aggregate

Metric	Shaped	Unshaped	p-value
Instability	3.4 ± 0.6	10.7 ± 1.3	$< 10^{-14}$
Utilization	84.9 ± 1	89 ± 1.4	$< 10^{-4}$
Players' bw share	36.9 ± 1	43.2 ± 1.1	$< 10^{-9}$

Table 2: The instability and utilization metrics, as well as the players' bandwidth share, in the presence of a bulk TCP transfer. P-values for an equality-of-means test are shown in the rightmost column.

utilization is quite high in both cases because the TCP flow tends to fill up the bottleneck. The players' bandwidth share (the aggregate bandwidth consumed by the three players) is slightly less in the case of shaped players, but the magnitude of the difference is small. The TCP connection takes up the largest share of the bottleneck's capacity. DOUBLE CHECK: As a result, the average throughput of the TCP connection is 4.7 ± 0.3 Mbps and 4.4 ± 0.1 Mbps in the shaped and unshaped cases, respectively.

5.3 Mix of Players

What happens when a mix of shaped and unshaped players compete at the same bottleneck? How does the stability and throughput of these two sets of players compare? Does the presence of shaped players help to reduce the instability even of the unshaped players? Do unshaped players get higher throughput?

We examine these questions by performing a set of mixed-player experiments where two shaped and two unshaped players compete at a 12 Mbps bottleneck link. We also perform a set of base experiments where all players are unshaped. Each experiment is repeated four times.

Table 3 shows the instability and utilization metrics (95% confidence intervals) for three types of players: shaped players in mixed-player experiments, unshaped players in mixed-player experiments, and unshaped players in base experiments. P-values for an equality-of-means test between the mixed-player and base experiments are also shown.

Shaped players have the lowest instability. However in the set of mixed-player experiments, their presence does not seem to help stabilize the competing unshaped players (These results are in contrast with the ones reported in [6]. In the future, we plan to repeat these experiments under different scenarios and understand the conditions under which the existence of shaped players can mitigate the instability in unshaped ones.)

The bandwidth share of the shaped players is less than the bandwidth share of the unshaped players. However, the utilization is not significantly less in the case of the shaped players and the corresponding p-value is 0.21.

6. CONCLUSIONS

We proposed a shaping mechanism that can be used to mitigate the instability problem of adaptive streaming players when they share a network bottleneck and compete for bandwidth. The stabilization mechanism aims to eliminate the root cause of the problem, i.e., the ON-OFF download pattern of adaptive streaming players in Steady-State. The proposed mechanism does not require the players' cooperation and it is entirely implemented at the server. We conducted experiments with real adaptive streaming players and showed that the proposed mechanism works as expected in practice. We have also compared the performance (in terms of instability, utilization) of adaptive streaming with and without shaping. The proposed shaping mechanism results in a major reduction in the instability metric, practically stabilizing the competing players, at the expense of a small loss in utilization.

Metric	Shaped	Unshaped	All-Unshaped	p-value
Instability	3.7 ± 0.7	12.7 ± 1.3	12.7 ± 1.3	$< 10^{-5}$
Utilization	67.5 ± 1.9		69 ± 2.9	0.21
BW share	27.7 ± 0.6	39.3 ± 1.7	-	$< 10^{-10}$

Table 3: The instability and utilization metrics for a mix of shaped and unshaped players.

7. REFERENCES

[1] Adobe HTTP Dynamic Streaming File Packager. http://help.adobe.com/en_US/HTTPStreaming/1.0/Using/WS9463dbe8dbe45c4c-c126f3b1260533756d-7ffc.html.

[2] Adobe HTTP Origin Module. http://help.adobe.com/en_US/HTTPStreaming/1.0/Using/WS7b362c044b7dd076-735e76121260080a90e-8000.html.

[3] IIS Smooth Streaming HD Sample Content. http://www.microsoft.com/en-us/download/details.aspx?id=18199.

[4] Smooth Streaming Module for Apache. http://h264.code-shop.com/trac/wiki/Mod-Smooth-Streaming-Apache-Version1.

[5] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A.C. Begen. What happens when http adaptive streaming players compete for bandwidth? In *ACM NOSSDAV*, 2012.

[6] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A.C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players? In *ACM NOSSDAV*, 2013.

[7] S. Akhshabi, S. Narayanaswamy, A.C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptive video players over http. *Signal Processing: Image Communication*, 27:271–287, 2012.

[8] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A quest for an internet video quality-of-experience metric. In *ACM HotNets Workshop*, 2012.

[9] R. Houdaille and S. Gouache. Shaping http adaptive streams for a better user experience. *ACM MMSys*, 2012.

[10] T.Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *ACM IMC*, 2012.

[11] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *ACM CoNEXT*, 2012.

[12] S. Lederer, M. Christopher, and T. Christian. Dynamic adaptive streaming over http dataset. *ACM MMSys*, 2012.

[13] A. Mansy, B. Ver Steeg, and M. Ammar. Sabre: A client based technique for mitigating the buffer bloat effect of adaptive video flows. Technical report, Georgia Tech, 2012.

[14] A. Orebaugh, G. Ramirez, J. Burke, and J. Beale. *Wireshark and Ethereal network protocol analyzer toolkit*. Syngress Media Inc, 2007.

[15] OSMF Player. `http://www.osmf.org`.

[16] Smooth Streaming Player. `http://www.iis.net/download/SmoothClient`.

[17] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM CCR*, 27(1):31–41, 1997.