TcpHas: TCP for HTTP Adaptive Streaming

Chiheb Ben Ameur, Emmanuel Mory, Bernard Cousin, Eugen Dedu

TcpHas: TCP for HTTP Adaptive Streaming

Chiheb Ben Ameur
IRISA, University of Rennes 1
Rennes, France
chiheb.benameur@yahoo.fr

Emmanuel Mory
Orange Labs
Cesson Sévigné, France
emmanuel.mory@orange.com

Bernard Cousin
IRISA, University of Rennes 1
Rennes, France
bernard.cousin@irisa.fr

Eugen Dedu
Univ. Bourgogne Franche-Comte
Montbéliard, France
eugen.dedu@univ-fcomte.fr

***Abstract*—HTTP Adaptive Streaming (HAS) is a widely used video streaming technology that suffers from a degradation of user's Quality of Experience (QoE) and network's Quality of Service (QoS) when many HAS players are sharing the same bottleneck link and competing for bandwidth. The two major factors of this degradation are: the large OFF period of HAS, which causes false bandwidth estimations, and the TCP congestion control, which is not suitable for HAS given that it does not consider the different video encoding bitrates of HAS.**

This paper proposes a HAS-based TCP congestion control, TcpHas, that minimizes the impact of the two aforementioned issues. It does this by using traffic shaping on the server. Simulations indicate that TcpHas improves both QoE, mainly by reducing instability, and QoS, mainly by reducing queuing delay and packet drop rate.

I. Introduction

Video streaming is a widely used service. According to 2016 Sandvine report [1], in North America, video and audio streaming in fixed access networks accounts for over 70% of the downstream bandwidth in evening hours. Given this high usage, it is of extreme importance to optimize its use. This is usually done by adapting the video to the available bandwidth. Numerous adaptation methods have been proposed in the literature and by major companies, and their differences mainly rely on the entity that does the adaptation (client or server), the parameter used for adaptation (the network or sender or client buffers), and the protocols used; the major companies having finally opted for HTTP [2].

HTTP Adaptive Streaming (HAS) is a streaming technology where video contents are encoded and stored at different qualities at the server and where players (clients) can choose periodically the quality according to the available resources. Popular implementations are Microsoft Smooth Streaming, Apple HTTP Live Streaming, and MPEG DASH. Still, *this technology is not optimal for video streaming*, mainly because its HTTP data is transported using the TCP protocol. Indeed, video data is encoded at distinct bitrates, and TCP does not increase the throughput sufficiently quickly when the bitrate changes. TCP variants (such as Cubic, Illinois, and Westwood+) specific to high bandwidth-delay product networks achieve high bandwidth more quickly and seem to give better performance for HAS service than classical TCP variants such as NewReno and Vegas [3], but the improvement is limited.

Another reason is the highly periodic ON–OFF activity pattern specific to HAS. Server-based shaping methods at application layer have been proposed to reduce its occurrence. They are cross-layer because they interact with the TCP layer and its parameters such as the congestion window, *cwnd*, and the round-trip time, *RTT*. Hence, implementing HAS traffic shaping at the TCP level is naturally more practical and easier to manage; in addition, this should offer better bandwidth share among HAS streams, reduce congestion events and improve the Quality of Experience (QoE) of HAS users.

Despite the advantages of using the transport layer for HAS, and in contrast with other types of streaming, where methods at the transport layer have already been proposed (RTP and TFRC [4]), to the best of our knowledge, *there is no proposition at the transport level specifically designed for HAS*. For commercial video providers YouTube, Dailymotion, Vimeo and Netflix, according to [5], "The quality switching algorithms are implemented in the client players. A player estimates the bandwidth continuously and transitions to a lower or to a higher quality stream if the bandwidth permits." The streaming depends on many parameters, such as player, video quality, device and video service provider etc., and uses various techniques such as several TCP connections, variable chunk sizes, different processing for audio and video flows, different throttling factors etc. To conclude, all these providers use numerous techniques, all of them based on client.

Therefore, in this paper, we propose a HAS-based TCP congestion control, TcpHas, that aims to minimize the aforementioned issues and to unify all these techniques. It uses four submodules: bandwidth estimator, optimal quality level estimator, *ssthresh* updating, and *cwnd* updating to the shaping rate. Simulation results show that TcpHas considerably improves both QoS (queuing delay, packet drop rate) and QoE (stability) and performs well even with several concurrent clients.

The remainder of this paper is organized as follows: Section II presents server-based shaping methods and describes possible optimizations at TCP level. Then, Section III describes TcpHas congestion control and Section IV evaluates it. Section V concludes the article.

II. Background and related works

Many server-based shaping methods have been proposed in the literature to improve QoE and QoS of HAS. Their functioning is usually separated into two modules:

1) Estimation of the optimal quality level, based on network conditions, such as bandwidth, delay, and/or history of selected quality levels, and available encoding bitrates of the video.

2) Shaping function of the sending rate, which should be suitable to the encoding bitrate of the estimated optimal quality level.

The next two subsections describe constraints and proposed solutions for each module. The last subsection presents some possible ways of optimization, which provides the basis for the TcpHas design.

A. Optimal Quality Level Estimation

A major constraint of optimal quality level estimation is that the server has no visibility on the flows that share the home network.

Ramadan et al. [6] propose an algorithm to reduce the oscillations of quality during video adaptation. During streaming, it marks each quality as unsuccessful or successful, depending on whether it has led to lost packets or not. We note that, to discover the available bandwidth, this method increases throughput and pushes to packet drop, which is different from our proposed method, where the available bandwidth is computed using an algorithm.

Akhshabi et al. [7] propose a server-based shaping method that aims to stabilize the quality level sent by the server by detecting oscillation events. The shaping function is activated only when oscillations are detected. The optimal quality level is based on the history of quality level oscillations. Then, the server shapes its sending rate based on the encoding bitrate of the estimated optimal quality level. However, when the end-to-end available bandwidth increases, the HAS player cannot increase its quality level when the shaper is activated. Also, to verify whether the optimal quality level has been increased or not, the server is obliged to deactivate the shaper to let the TCP congestion control algorithm occupy the remaining capacity available for the HAS stream. What is missing is a good estimation of the available bandwidth for the HAS flow.

B. Traffic Shaping Methods

Ghobadi et al. propose a shaping method on the server side called *Trickle* [8]. This method was proposed for YouTube in 2011, when it adopted progressive download technology. The key idea of Trickle is to place a dynamic upper bound on the congestion window (*cwnd*) such that TCP itself limits the overall data rate. The server application periodically computes the *cwnd* bound from the product between the round-trip time (*RTT*) and the target streaming bitrate.

The selection of the shaping rate by a server-based shaping method does not mean that the player will automatically start requesting that next higher quality level [7]. Furthermore, it was reported [3] that *ssthresh* has a predominant effect on accelerating the selection of the desired optimal quality level by the HAS client. Indeed, when *ssthresh* is set higher than the product of shaping rate and round-trip time, the server becomes aggressive and causes congestions and a reduction of quality level selection on the player side. In contrast, when *ssthresh* is set lower than this product, the congestion window *cwnd* takes several round-trip times to reach the value of this product. Hence, to improve the performance of the shaper, *ssthresh* needs to be modified too.

C. Optimization of Current Solutions

What can be noted from the different proposed methods for estimating the optimal quality level is that an efficient end-to-end estimator of available bandwidth can improve their performance, as shown in Subsection II-A. In addition, the only parameter from the application layer needed for shaping the HAS traffic is the encoding bitrate of each available quality level of the corresponding HAS stream. As explained in Subsection II-B, the remaining parameters are found in the TCP layer: the congestion window *cwnd*, the slow-start threshold *ssthresh*, and the round-trip time *RTT*. We are particularly interested in adjusting *ssthresh* to accelerate the reactivity of the HAS player. Naturally, what is missing here is an efficient TCP-based method for end-to-end bandwidth estimation. We also need a mechanism that adjusts *ssthresh* based on the output of the bandwidth estimator scheme.

Both this mechanism and estimation schemes used by various TCP variants are introduced in the following.

1) Adaptive Decrease Mechanism: In the literature, we found a specific category of TCP variants that set *ssthresh* using bandwidth estimation. Even if the estimation is updated over time, TCP uses it only when a congestion event is detected. The usefulness of this mechanism, known as *adaptive decrease mechanism*, is described in [9] as follows: "*it provides a congestion window that is decreased more in the presence of heavy congestion and less in the presence of light congestion or losses that are not due to congestion*". The adaptive decrease mechanism is described in Algorithm 1 when detecting a congestion, i.e., when receiving three duplicated ACKs or when the retransmission timeout expires. The algorithm uses the estimated bandwidth, $\widehat{Bwe}$, multiplied by RTT_{min} to update the *ssthresh* value.

Algorithm 1 TCP adaptive decrease mechanism.

if $3\,duplicate\,ACKs\,are\,received$ **then**
  $ssthresh = \widehat{Bwe} \times RTT_{min}$
  if $cwnd > ssthresh$ **then**
    $cwnd = ssthresh$
  end if
end if
if $retransmission\,timeout\,expires$ **then**
  $ssthresh = \widehat{Bwe} \times RTT_{min}$
  $cwnd = initial\_cwnd$ ▷ i.e. $2 \times MSS$
end if

▷ where $\widehat{Bwe}$ is the estimated bandwidth and RTT_{min} is the lowest RTT measurement

2) Bandwidth Estimation Schemes: The most common TCP variant that uses bandwidth estimation to set *ssthresh* is Westwood. Other newer variants have been proposed, such as Westwood+ and TIBET. The only difference between them is the bandwidth estimation scheme used. Because of space limitation, we describe only TIBET in the following.

The basic idea of TIBET [10] is to perform a run-time sender-side estimate of the average packet length and the

average inter-arrival separately. The bandwidth estimation scheme is applied to the stream of the received ACKs and is described in Algorithm 2 [11], where $acked$ is the number of segments acknowledged by the last ACK, $packet\_size$ is the average segment size in bytes, now is the current time and $last\_ack\_time$ is the time of the previous ACK reception. $Average\_packet\_length$ and $Average\_interval$ are the low-pass filtered measures of the packet length and the interval between sending times.

Algorithm 2 Bandwidth estimation scheme.

1: **if** $ACK\ is\ received$ **then**
2: $sample\_length = acked \times packet\_size \times 8$
3: $sample\_interval = now - last\_ack\_time$
4: $Average\_packet\_length = alpha \times Average\_packet\_length + (1 - alpha) \times sample\_length$
5: $Average\_interval = alpha \times Average\_interval + (1 - alpha) \times sample\_interval$
6: $Bwe = Average\_packet\_length / Average\_interval$
7: **end if**

$Alpha$ ($0 \leq alpha \leq 1$) is the pole of the two low-pass filters. The value of $alpha$ is critical to TIBET performance: If $alpha$ is set to a low value, TIBET is highly responsive to changes in the available bandwidth, but the oscillations of Bwe are quite large. In contrast, if $alpha$ approaches 1, TIBET produces more stable estimates, but is less responsive to network changes. If $alpha$ is set to zero Algorithm 2 will be equivalent to Westwood, where the sample Bwe varies between 0 and the bottleneck bandwidth.

TIBET uses a second low-pass filtering, with parameter γ, on the estimated available bandwidth Bwe to give a better smoothed estimation $\widehat{Bwe}$. γ is a variable parameter, equal to e^{-T_k}, where $T_k = t_k - t_{k-1}$ is the time interval between the two last received ACKs. This means that bandwidth estimation samples Bwe with high T_k values are given more importance than those with low T_k values.

Simulations [11] indicate that TIBET gives bandwidth estimations very close to the correct values, even in the presence of other UDP or TCP flows. It is also robust against packet clustering and ACK compression in contrast to Westwood. Moreover, TIBET is less dependent to *cwnd* than Westwood+. Indeed, Westwood+ estimates bandwidth only once per RTT, hence its estimation is always bounded by $\frac{\min(cwnd,rwnd)}{RTT}$.

III. TcpHas Description

As shown in the previous section, a protocol specific to HAS needs to modify several TCP parameters and consists of several algorithms. Our HAS-based TCP congestion control, TcpHas, is based on the two modules of server-based shaping solution: optimal quality level estimation and sending traffic shaping itself, both with two submodules. The first module uses a bandwidth estimator submodule inspired by the TIBET scheme and adapted to HAS context, and an optimal quality level estimator submodule to define the quality level, $\widehat{QLevel}$, based on the estimated bandwidth. The second module uses $\widehat{QLevel}$ in two submodules that update the values of *ssthresh* and *cwnd* over time. This section progressively presents TcpHas by describing these four submodules.

A. Bandwidth Estimator of TcpHas

As described in Section II, TIBET performs better than other proposed schemes. It reduces the effect of ACK compression and packet clustering and is less dependent on the congestion window, *cwnd*, than Westwood+.

As explained in Section II-C2, the parameter γ used by TIBET to smooth Bwe estimations is variable ($\gamma = e^{-T_k}$). However, this variability is not suited to HAS. Indeed, when the HAS stream has a large OFF period, the HTTP Get request packet sent from client to server to ask for a new chunk is considered by the server as a new ACK. As a consequence, the new bandwidth estimation sample, Bwe, will have an underestimated value and γ will be wrongly reduced. We therefore make parameter γ constant.

B. Optimal Quality Level Estimator of TcpHas

TcpHas' optimal quality level estimator is based on the estimated bandwidth, $\widehat{Bwe}$, as described in Subsection III-A. This estimator is a function that adapts HAS features to TCP congestion control and replaces $\widehat{Bwe}$ value by the encoding bitrate of the estimated optimal quality level $\widehat{QLevel}$. One piece of information from the application layer is needed: the video encoding bitrates of the HAS stream that are available in the server. These encoding bitrates are given in the index file of the HAS stream. The *EncodingRate* vector of TcpHas contains the available encoding bitrates in ascending order. Our estimator is defined by the function *QLevelEstimator*, described in Algorithm 3, which selects the highest quality level whose encoding bitrate is equal to or lower than the estimated bandwidth, $\widehat{Bwe}$.

Algorithm 3 QLevelEstimator function.

1: **for** $i = length(EncodingRate) - 1$ **downto** 0 **do**
2: **if** $EncodingRate[i] \leq \widehat{Bwe}$ **then**
3: break
4: **end if**
5: **end for**
6: $\widehat{QLevel} = i$

$\widehat{QLevel}$ parameter is updated only by this function. However, the time and frequency of its updating is a delicate issue:

- We need to use the adaptive decrease mechanism because when a congestion occurs, $\widehat{QLevel}$ needs to be updated to the new network conditions. Hence, this function is called after each congestion detection.
- Given that TcpHas performs a shaping rate that reduces $\widehat{OFF}$ occupancy, when TcpHas detects an $\widehat{OFF}$ period, it may mean that some network conditions have changed (e.g. an incorrect increase of the shaping rate). Accordingly, to better estimate the optimal quality level, this function is called after each $\widehat{OFF}$ period.

The *EncodingRate* vector is also used by TcpHas during application initialization to differentiate between a HAS application and a normal one: when the application returns an

empty vector, it is a normal application, and TcpHas just makes this application be processed by classical TCP, without being involved at all.

C. Ssthresh Modification of TcpHas

The TCP variants that use the TCP decrease mechanism use RTT_{min} multiplied by the estimated bandwidth, $\widehat{Bwe}$, to update *ssthresh*. However, given that the value of *ssthresh* affects the reactivity of the player, it should correspond to the desired shaping rate instead of $\widehat{Bwe}$. Also, the shaping rate is defined in Trickle [8] to be 20% higher than the encoding bitrate, which allows the server to deal better with transient network congestion.

Hence, for TcpHas we decided to replace $\widehat{Bwe}$ by $EncodingRate[\widehat{QLevel}] \times 1.2$, which represents its shaping rate:

$$ssthresh = EncodingRate[\widehat{QLevel}] \times RTT_{min} \times 1.2 \quad (1)$$

The timing of *ssthresh* updating is the same as that of $\widehat{QLevel}$: when detecting a congestion event and just after an idle $\widehat{OFF}$ period. Moreover, the initial value of *ssthresh* should be modified to correspond to the context of HAS. These three points are presented in the following.

1) Congestion Events: Inspired by Algorithm 1, the TcpHas algorithm when detecting a congestion event is described in Algorithm 4. It includes the two cases of congestion events: three duplicated ACKs, and retransmission timeout. In both cases, $\widehat{Qlevel}$ is updated from $\widehat{Bwe}$ using the *QLevelEstimator* function. Then, *ssthresh* is updated according to (1). The update of *cwnd* is as in Algorithm 1.

Algorithm 4 TcpHas algorithm when congestion occurs.

```
if 3 duplicate ACKs are received then
    QLevel = QLevelEstimator(Bwe)
    ssthresh = EncodingRate[QLevel] × RTT_min × 1.2
    if cwnd > ssthresh then
        cwnd = ssthresh
    end if
end if
if retransmission timeout expires then
    QLevel = QLevelEstimator(Bwe)
    ssthresh = EncodingRate[QLevel] × RTT_min × 1.2
    cwnd = initial_cwnd                          ▷ i.e. 2 × MSS
end if
```

2) Idle Periods: As explained in [12], [13], [3], the congestion window, *cwnd*, is reduced when the idle period exceeds the retransmission timeout ***RTO***, and *ssthresh* is updated to $\max(ssthresh, 3/4 \times cwnd)$. In HAS context, the idle period coincides with the OFF period. In addition, we denote by $\widehat{OFF}$ the OFF period whose duration exceeds ***RTO***. Accordingly, reducing *cwnd* after an $\widehat{OFF}$ period will force *cwnd* to switch to slow-start phase although the server is asked to deliver the video content with the optimal shaping rate.

To avoid this, we propose to remove the *cwnd* reduction after the $\widehat{OFF}$ period. Instead, as presented in Algorithm 5, TcpHas updates $\widehat{Qlevel}$ and *ssthresh*, then sets *cwnd* to *ssthresh*. This modification is very useful in the context of HAS. On the one hand, it eliminates the sending rate reduction after each $\widehat{OFF}$ period, which adds additional delay to deliver the next chunk and may cause a reduction of quality level selection on the player side. On the other hand, the update of *ssthresh* each $\widehat{OFF}$ period allows the server to adjust its sending rate more correctly, especially when the client generates a high $\widehat{OFF}$ period between two consecutive chunks.

Algorithm 5 TcpHas algorithm after an $\widehat{OFF}$ period.

```
if idle > RTO then                          ▷ OFF period detected
    QLevel = QLevelEstimator(Bwe)
    ssthresh = EncodingRate[QLevel] × RTT_min × 1.2
    cwnd = ssthresh
end if
```

3) Initialization: By default, TCP congestion control uses an initial value of *ssthresh*, $initial\_ssthresh$, of 65535 bytes. However, in HAS context, $initial\_ssthresh$ is better to match an encoding bitrate. We decided to set it to the highest quality level at the beginning of streaming for two reasons: 1) give a similar initial aggressiveness as classical TCP and 2) avoid to be too higher than highest encoding bitrate to maintain the HAS traffic shaping concept.

This initialization should be done in conformity with (1), hence the computation of ***RTT*** is needed. Consequently, TcpHas just updates the *ssthresh* when the first ***RTT*** is computed. In this case, our updated *ssthresh* serves the same purpose as $initial\_ssthresh$. TcpHas initialization is presented in Algorithm 6.

Algorithm 6 TcpHas initialization.

```
QLevel = length(EncodingRate) − 1 ▷ highest quality level
cwnd = initial_cwnd                          ▷ i.e. 2 × MSS
ssthresh = initial_ssthresh                  ▷ i.e. 65535 bytes
RTT = 0
if new ACK is received then
    if RTT ≠ 0 then      ▷ i.e. when the first RTT is computed
        ssthresh = EncodingRate[QLevel] × RTT × 1.2
    end if
end if
```

D. Cwnd Modification of TcpHas for Traffic Shaping

As shown in Section II-B, Trickle does traffic shaping on the server-side by setting a maximum threshold for *cwnd*, equal to the shaping rate multiplied by the current ***RTT***. However, during congestion avoidance phase (i.e., when $cwnd > ssthresh$), *cwnd* is increased very slowly by one MSS each ***RTT***. Consequently, when *cwnd* is lower than this threshold, it takes several ***RTTs*** to reach it, i.e. a slow reactivity.

To increase its reactivity, we modify TCP congestion avoidance algorithm by directly tuning *cwnd* to match the shaping rate. For this, we use smoothed ***RTT*** computations through a low-pass filter. The smoothed ***RTT*** that is updated at each ACK reception is:

$$\widehat{RTT}_k = \psi \times \widehat{RTT}_{k-1} + (1 - \psi) \times RTT_k \quad (2)$$

where $0 \leq \psi \leq 1$. TcpHas algorithm during the congestion avoidance phase is described in Algorithm 7, where $EncodingRate[\widehat{QLevel}] \times 1.2$ is the shaping rate.

Algorithm 7 TcpHas algorithm in congestion avoidance.

1: **if** $new\ ACK\ is\ received$ **and** $cwnd \geq ssthresh$ **then**
2: $\quad cwnd = EncodingRate[\widehat{QLevel}] \times \widehat{RTT} \times 1.2$
3: **end if**

Finally, TcpHas coexists gracefully with TCP on server, the transport layer checking whether the *EncodingRate* vector returned by application is empty or not, as explained before.

IV. TcpHas Evaluation

The final goal of our work is to implement our idea in real software. However, at this stage of the work, we preferred instead to use a simulated player because of the classical advantages of simulation over experimentation, such as reproducibility of results, and measurement of individual parameters for better tuning of the parameters of our idea.

In this section, we evaluate TcpHas using ns-3 simulator, version 3.17. In our scenario, HAS players share the same bottleneck link and compete for bandwidth in a home network. We first describe the network setup used in the simulations. Then, we describe the parameter settings used during the simulations for both evaluation and configuration of TcpHas. Then, we show the behavior of TcpHas. Finally, we analyse results for 1 to 9 competing HAS flows in the home network.

A. Network Architecture in ns-3

Fig. 1 presents the architecture we used, which is compliant with the fixed broadband access network architecture used by Cisco to present its products [14]. The HAS clients are located inside the home network, a local network with 100 Mbps bandwidth. The Home Network (HG) is connected to the DSLAM. The bottleneck link is located between HG and DSLAM and has 8 Mbps. The queue of the DSLAM uses Drop Tail discipline with a length that corresponds to the bandwidth-delay product. Nodes BNG (Broadband Network Gateway) and IR (Internet Router), and links AggLine (that simulates the aggregate line), ISPLink (that simulates the Internet Service Provider core network) and NetLink (that simulates the route between the IR and the HAS server) are configured so that their queues are large enough (1000 packets) to support a large bandwidth of 100 Mbps and high delay of 100 ms without causing significant packet losses.

We generate Internet traffic that crosses ISPLink and AggLine, because the two simulated links are supposed to support a heavy traffic from ISP networks. For Internet traffic, we use the *Poisson Pareto Burst Process* (PPBP) model [15], considered as a simple and accurate traffic model that matches statistical properties of real-life IP networks (such as their bursty behavior). PPBP is a process based on the overlapping of multiple bursts with heavy-tailed distributed lengths. Events in this process represent points of time at which one of an infinite population of users begins or stops transmitting a

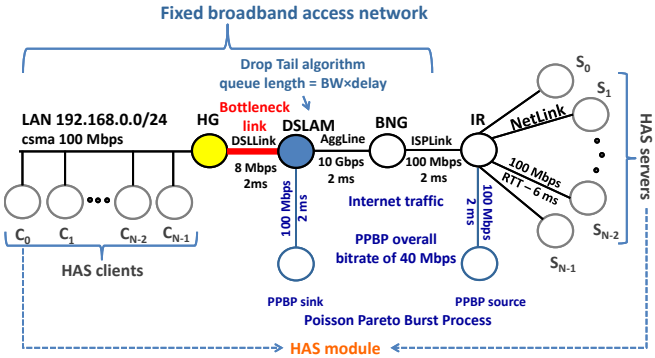


Fig. 1: Network architecture used in ns-3.

Video quality level L_{C-S}	0	1	2	3	4
Encoding bitrate (kbps)	248	456	928	1632	4256

TABLE I: Video encoding bitrates.

traffic burst. PPBP is closely related to the $M/G/\infty$ queue model [15]. We use the PPBP implementation in ns-3 [16], [17]. In our configuration, the overall rate of PPBP traffic is 40 Mbps, which corresponds to 40% of ISPLink capacity.

We use an ns-3 simulated HAS player similar to the emulated player described in [3], with a chunk duration of 2 seconds and a playback buffer of 30 seconds (maximum video size the buffer can hold). Note that [3] compares four TCP variants and two router-based traffic shaping methods, whereas the current article proposes a new congestion control executed on server. The player is classified as *Rate and Buffer based* (RBB) player, following classification proposed in [18], [19]. Using buffer occupancy information is increasingly proposed and used due to its advantages for reducing stalling events. In addition, the bandwidth estimator we use consists in dividing the size of received chunk by its download duration. The buffer occupancy information is used only to define an aggressiveness level of the player, which allows the player to ask a quality level higher than the estimated bandwidth.

All tests use five video quality levels with constant encoding rates presented in Table I, and correspond to the quality levels usually used by many video service providers.

We also use the HTTP traffic generator module given in [20], [21]. This module allows communication between two nodes using HTTP protocol, and includes all features that generate and control HTTP Get Request and HTTP response messages. We wrote additional code into this HTTP module by integrating the simulated HAS player. We call this implementation the HAS module, as presented in Fig. 1. Streaming is done from S_f to C_f, where $0 \leq f < N$. The round-trip propagation delay between S_f and C_f is 100 ms.

For all evaluations, we use competing players that are playing simultaneously during K seconds. We set K = 180 seconds, which allows the HAS players to reach stationary behavior when they are competing for bandwidth [22].

B. TcpHas Parameter Settings

We set $\gamma = 0.99$ (see Subsection III-A) to reduce the oscillation of bandwidth estimations, $\widehat{Bwe}$, over time. We set the initial bandwidth estimation value of $\widehat{Bwe}$ to the highest encoding bitrate, as explained in Subsection III-C.

The parameter $alpha$ of the TIBET estimation scheme (see Algorithm 2) is set to 0.8. A higher value produces more stable estimations but is less responsive to network changes, and a lower value makes TcpHas too aggressive.

The parameter ψ used for low-pass filtering the *RTT* measurements in Subsection III-D is set to 0.99, which leads to better performance in our tests, especially when network conditions degrade.

C. TcpHas Behavior

We present results for a scenario with 8 competing clients. The optimal quality level that should be selected by the competing players is n° 2.

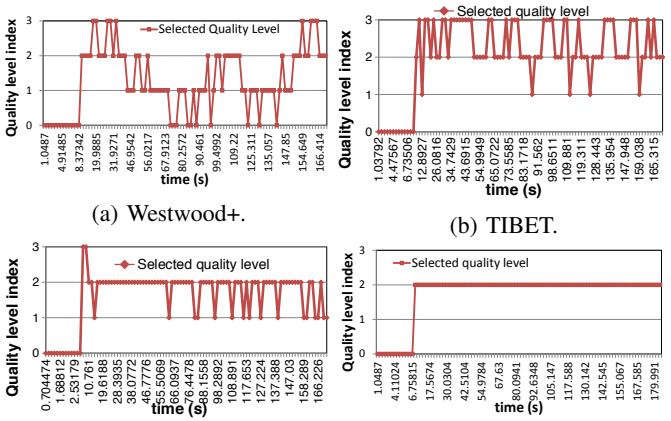
We also show the behavior of Westwood+, TIBET and Trickle for the same scenario. We do not show the behavior of Westwood because Westwood+ is supposed to replace Westwood since it performs better in case of ACK compression and clustering. Concerning Trickle, it is a traffic shaping method that was proposed in the context of progressive download, as described in II-B. In order to adapt it to HAS, we added to it the estimator of optimal quality level of TcpHas, the adaptive decrease mechanism of Westwood+ (the same as TIBET), and applied the Trickle traffic shaping based on the estimated optimal quality level. This HAS adaptation of Trickle is simply denoted by "Trickle" in the reminder of this paper.

Fig. 2 shows the quality level selection over time of one of the competing players for the above methods. During the buffering phase, all players select the lowest quality level, as allowed by slow start phase. However, during the steady-phase the results diverge: Westwood+ player frequently changes the quality level between n° 0 and n° 3, which means that the player yields an unstable HAS stream and risks to generate stalling events. TIBET and Trickle players have an improved performance but still with frequent quality changes. In contrast, TcpHas player is stable at the optimal quality level, hence it performs the best among all the methods.

D. Performance Evaluation

Here, we vary the number of competing players from 1 to 9. We select a maximum of 9 competing HAS clients because in practice the number of users inside a home network does not exceed 9. We use one objective QoE metric: the *instability of video quality level* as defined in [23], [13], [3] (0% means the same quality, 100% means the quality changes *each* second). We also use two QoS metrics: the *average queuing delay* and the *average packet drop rate at the bottleneck*. This packet drop rate gives an idea of the congestion severity of the bottleneck link and is defined during a *K*-second test duration as follows:

$$DropPkt(K) = \frac{number\ of\ dropped\ packets\ during\ K\ seconds}{K} \times 100 \quad (3)$$



(a) Westwood+. (b) TIBET.

(c) Trickle. (d) TcpHas.

Fig. 2: Quality level selection over time.

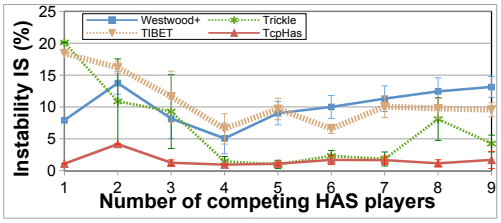
The mean performance measurements of the QoE and QoS metrics among competing players and for 16 runs are shown in Fig. 3. We chose 16 runs because the relative difference between mean measurements of instability of 16 and 64 runs is less than 4%. We present the performance unfairness measurements among HAS clients with vertical error bars.

Fig. 3a shows that TcpHas yields the lowest instability rate (less than 4%), with a negligible instability unfairness between players. Trickle shows similar instability for 4 and 7 players, but for the other cases it has a high instability rate, the reason being that Trickle still initializes *cwnd* after each $\widehat{OFF}$ period which causes a low sending rate. In contrast, the instability of Westwood+ and TIBET is much greater and increases with the number of competing players.

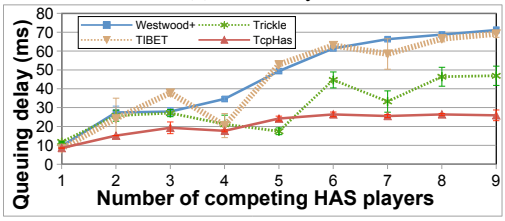
As presented in Fig. 3b, although the queuing delay of the four methods increases with the number of competing clients, TcpHas and Trickle show a lower queuing delay than Westwood+ and TIBET. The reason is that both TcpHas and Trickle shape the HAS flows by reducing the sending rate of the server which reduces queue overflow in the bottleneck. Additionally, we notice that TcpHas reduces more the queuing delay than Trickle; TcpHas has roughly half of the queuing delay of Westwood+ and TIBET. Besides, TcpHas does not increase its queuing delay more than 25 ms even for 9 competing players, while Trickle increases it to about 50 ms. This result is mainly due to the high stability of the HAS quality level generated by TcpHas which offers better fluidity of HAS flows inside the bottleneck. The same reason applies for the very low packet drop rate at the bottleneck of TcpHas, given in Fig. 3c. In addition, due to its corresponding traffic shaping method that reduces the sending rate of HAS server, the packet drop rate of Trickle is quite similar to that of TcpHas, as shown in Fig. 3c.

V. Conclusion

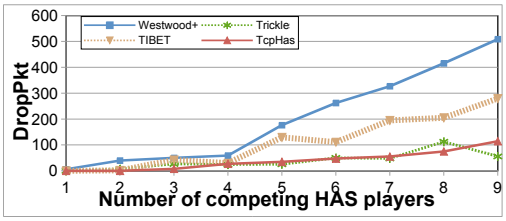
This paper first presents and analyses server-based shaping methods that aim to stabilize the video quality level and improve the QoE of HAS users.



(a) Instability.



(b) Queuing delay.



(c) Packet drop rate.

Fig. 3: QoE and QoS measurements when increasing the number of HAS competing clients.

Based on this analysis, we propose and describe TcpHas, a HAS-based TCP congestion control that acts like a server-based HAS traffic shaping method. It is inspired by the TCP adaptive decrease mechanism and uses the end-to-end bandwidth estimation of TIBET to estimate the optimal quality level. Then, it shapes the sending rate to match the encoding bitrate of the estimated optimal quality level. The traffic shaping process is based on updating *ssthresh* when detecting a congestion event or after an idle period, and on modifying *cwnd* during congestion avoidance phase.

We evaluate TcpHas when several HAS clients share the bottleneck link and compete for the same home network. Simulation results indicate that TcpHas considerably improves both HAS QoE and network QoS. Concerning QoE, it offers a high stability of quality level and has tendency to select the optimal quality level. Concerning QoS, it reduces queuing delay, and reduces considerably the packet drop rate in the shared bottleneck queue. Moreover, TcpHas performs well when increasing the number of competing HAS clients.

As future work, we intend to optimize TcpHas so that it adapt its performance to Live streaming service and unstable network conditions, and also to implement and evaluate TcpHas in real software and networks.

REFERENCES

[1] Sandvine, "Global internet phenomena report," https://www.sandvine.com, Jun. 2016.

[2] E. Dedu, W. Ramadan, and J. Bourgeois, "A taxonomy of the parameters used by decision methods for adaptive video transmission," *Multimedia Tools and Applications*, vol. 74, no. 9, pp. 2963–2989, May 2015.

[3] C. Ben Ameur, E. Mory, and B. Cousin, "Combining traffic shaping methods with congestion control variants for HTTP adaptive streaming," *Multimedia Systems*, pp. 1–18, 2016, available online.

[4] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol specification," Sep. 2008, RFC 5348.

[5] M. A. Hoquea, M. Siekkinena, J. K. Nurminena, M. Aaltob, and S. Tarkoma, "Mobile multimedia streaming techniques: QoE and energy saving perspective," *Pervasive and Mobile Computing*, vol. 16, Part A, pp. 96–114, Jan. 2015, netflix, youtube, vimeo... streaming techniques.

[6] W. Ramadan, E. Dedu, and J. Bourgeois, "Avoiding quality oscillations during adaptive streaming of video," *International Journal of Digital Information and Wireless Communications (IJDIWC)*, vol. 1, no. 1, pp. 126–145, Nov. 2011.

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

[8] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis, "Trickle: Rate limiting youtube video streaming." in *Usenix Annual Technical Conference*, 2012, pp. 191–196.

[9] S. Mascolo and G. Racanelli, "Testing TCP Westwood+ over transatlantic links at 10 gigabit/second rate," in *Protocols for Fast Long-distance Networks (PFLDnet) Workshop*, 2005.

[10] A. Capone, F. Martignon, and S. Palazzo, "Bandwidth estimates in the TCP congestion control scheme," in *IWDC*. Springer, 2001, pp. 614–626.

[11] A. Capone, L. Fratta, and F. Martignon, "Bandwidth estimation schemes for TCP over wireless networks," *IEEE Transactions on Mobile Computing*, vol. 3, no. 2, pp. 129–143, 2004.

[12] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," 2009, rFC 5681.

[13] C. Ben Ameur, E. Mory, and B. Cousin, "Evaluation of gateway-based shaping methods for HTTP adaptive streaming," in *Quality of Experience-based Management for Future Internet Applications and Services (QoE-FI) Workshop, IEEE International Conference on Communications (ICC)*, 2015, pp. 1–6.

[14] Cisco, "Broadband network gateway overview," 2013, http://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k_r4-3/bng/configuration/guide/b_bng_cg43xasr9k/b_bng_cg43asr9k_chapter_01.html.

[15] M. Zukerman, T. D. Neame, and R. G. Addie, "Internet traffic modeling and future technology implications," in *INFOCOM, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1. IEEE, 2003, pp. 587–596.

[16] D. Ammar, "PPBP in ns-3," 2016, https://codereview.appspot.com/4997043.

[17] D. Ammar, T. Begin, and I. Guerin-Lassous, "A new tool for generating realistic internet traffic in ns-3," in *4th International ICST Conference on Simulation Tools and Techniques*, 2011, pp. 81–83.

[18] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 325–338, August 2015.

[19] X. Yin, V. Sekar, and B. Sinopoli, "Toward a principled framework to design dynamic adaptive streaming algorithms over http," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM, Oct. 2014, p. 9.

[20] Y. Cheng, "HTTP traffic generator," 2016, https://codereview.appspot.com/4940041.

[21] Y. Cheng, E. K. Çetinkaya, and J. P. Sterbenz, "Transactional traffic generator implementation in ns-3," in *6th International ICST Conference on Simulation Tools and Techniques*, 2013, pp. 182–189.

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

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