Combining Traffic Shaping Methods with Congestion Control Variants for HTTP Adaptive Streaming

Chiheb Ben Ameur, Emmanuel Mory, Bernard Cousin

HAL Id: hal-01249646
https://hal.science/hal-01249646

Submitted on 2 Jan 2016

Combining Traffic Shaping Methods with Congestion Control Variants for HTTP Adaptive Streaming

Chiheb Ben Ameur
Orange Labs
Rennes, France
Chiheb.benameur@orange.com

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

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

ABSTRACT

HTTP adaptive streaming (HAS) is a streaming video technique widely used over the Internet. However, it has many drawbacks that degrade its user quality of experience (QoE). Our investigation involves several HAS clients competing for bandwidth inside the same home network. Studies have shown that managing the bandwidth between HAS clients using traffic shaping methods improves the QoE. Additionally, the TCP congestion control algorithm in the HAS server may also impact the QoE because every congestion control variant has its own method to control the congestion window size. Based on previous work, we describe two traffic shaping methods, the Hierarchical Token Bucket shaping Method (HTBM) and the Receive Window Tuning Method (RWTM), as well as four popular congestion control variants: NewReno, Vegas, Illinois, and Cubic. In this paper, our objective is to provide a detailed comparative evaluation of combining these four congestion control variants with these two shaping methods. The main result indicates that Illinois with RWTM offers the best QoE without causing congestion. Results were validated through experimentation and objective QoE analytical criteria.

Keywords

Traffic shaping; Congestion control; Quality of Experience; HTTP Adaptive Streaming; Bandwidth management.

1. INTRODUCTION

HTTP adaptive streaming (HAS) is a streaming video technique based on downloading video segments of short duration. These segments are called chunks, and they are streamed from a HAS server to a HAS client through the network. Each chunk is encoded at multiple quality levels. After requesting a chunk by an HTTP GET request message, when the chunk is received, the player on the client side stores it in a playback buffer. The HAS player operates in one of two states: a buffering state and a steady state. During the first state, the player requests a new chunk as soon as a previous chunk has been downloaded, until the playback buffer is filled. However, during the steady state, the player requests chunks periodically in order to maintain a constant playback buffer size. The steady state includes periods of activity (ON periods) followed by periods of inactivity (OFF periods) [9], [11].

The Quality of Experience (QoE) of an HTTP adaptive stream depends primarily on three criteria:

1. Video quality level stability [1], [11]: A frequent change of video quality level bothers the user. Therefore, quality level fluctuation should be avoided to improve the QoE.
2. Fidelity to optimal quality level selection: The user prefers to watch the best video quality level, when possible. Therefore, the HAS player should select the optimal quality level, which is the highest feasible quality level allowed by the available bandwidth.
3. Convergence speed: The user prefers to view the optimal quality level as soon as possible. Accordingly, the HAS player should rapidly select the optimal quality level. The delay that the player requires before the optimal quality level has been attained is called the convergence speed [1].

We studied a general use case in which several HAS clients are located in the same home network. In this use case, QoE degradations can be grouped into two main causes:

▪ Congestion events:

Video packets sent from the server to the client pass through many network devices. Each device has one or many queues that use a queuing discipline to schedule network packets. The implemented algorithm decides whether to route or drop incoming packets in order to avoid network congestion. The main bottleneck occurs near the home gateway, and more precisely, in the link between DSLAM and home gateway [12]. In fact, the DSLAM may considerably reduce the bandwidth offered to the home network, and it is more likely to drop packets than any other network device. To minimize network effects on the delivery, the TCP protocol implements a "congestion control algorithm" on the sender side, which reduces the bitrate of packets sent to the receiver when a packet is lost. However, this bitrate reduction may degrade QoE. In addition, there are many congestion control variants with different methods of managing the congestion window size, *cwnd*, and detecting congestion events. These differences may change the QoE between variants.

▪ Concurrence with other streams - OFF periods issue:

The HAS player estimates the available bandwidth by computing the download bitrate for each chunk when it has

finished downloading; this is done by dividing the chunk size by its download duration. As a consequence, the player cannot estimate the available bandwidth during OFF periods, because no data are being received. When a HAS stream concurs with other streams in the same home network, accurate bandwidth estimation becomes more difficult. For example, when two HAS streams are competing for bandwidth and the ON period of the first player coincides with the OFF period of the second player, the first player will overestimate its available bandwidth. This overestimation may lead the player to select a higher quality level for the next chunk. This selection may lead to a congestion event and a resulting fluctuation of quality levels between the two players. Research has demonstrated that traffic shaping can considerably limit this problem [1, 2, 11, 21, 25]. Traffic shaping consists of selecting a target bitrate for each HAS session in the home network based on bitrates of the available quality levels and the available bandwidth. It then shapes the outgoing traffic to each HAS client based on the selected target bitrate.

The objective of our study is to combine two solution categories, TCP congestion control variants to reduce the negative effects of congestion events, and traffic shaping methods, to restrict the drawbacks of the concurrence between HAS streams in the home gateway. The optimal combination will have the highest grade of QoE, i.e. the best possible video quality level stability, best fidelity to optimal quality level selection, and best convergence speed.

We note that there are many implementations of HAS that are currently deployed, such as Dynamic Adaptive Streaming over HTTP (MPEG-DASH), Microsoft Smooth Streaming (MSS), Apple HTTP Live Streaming (HLS), and Adobe HTTP Dynamic Streaming (HDS). For this reason, we wish to emphasize that in this paper we only choose HAS traffic shaping methods that do not change the HAS implementation either in the player or in the server. This choice provides adaptability to any HAS client or server implementations and thus offers a larger scope of application of our presented work.

The remainder of this paper is organized as follows. In Section 2, we describe the background and related works. In Section 3, we detail our methodology and experimental implementation. Section 4 presents the results and discussion. In Section 5, we conclude the paper and suggest future directions for our work.

2. BACKGROUND AND RELATED WORK

In this section, we describe the TCP congestion control variants and the HAS traffic shaping methods used in this work, and explain the distinctions between them.

2.1 TCP congestion control variants

All TCP congestion control variants have two common phases: a slow start phase and a congestion avoidance phase. The slow start phase consists of increasing *cwnd* rapidly by one maximum segment size (MSS) for each received acknowledgment (ACK), i.e. the *cwnd* value is doubled for each round trip time (*RTT*). This rapidity has an objective of reaching a high bitrate within a short duration. When the *cwnd* size exceeds a threshold called *ssthresh*, the TCP congestion control algorithm switches to the second phase: the congestion avoidance phase. This phase slowly increases the *cwnd* until a congestion event is detected.

TCP congestion control variants are classified according to two main criteria [13]:

1- The first criterion is the increase of *cwnd* during the congestion avoidance phase and the decrease of *cwnd* immediately following congestion detection. Generally, the increase is additive, and the *cwnd* size increases by one MSS for each RTT. For decreasing *cwnd*, the standard variants employ multiplicative decreasing, i.e. the *cwnd* size is weighted by a multiplicative decrease factor $(1-\beta)$, where $0 < \beta < 1$. This category is called the Additive Increase Multiplicative Decrease (AIMD) approach. Other variants using different techniques are classified as non-AIMD approaches.

2- The second criterion is the method by which the algorithm detects congestion. We distinguish three modes: loss-based, delay-based, and loss-delay-based modes. The loss-based mode considers any detection of packet loss as a congestion event. A majority of TCP congestion control variants that use the loss-based mode consider receiving three duplicated ACKs from the receiver as an indication of a packet loss and, as a consequence, as an indication of a congestion event. However, the delay-based mode considers a significant increase in the *RTT* value as the only indication of a congestion event. The third mode, the hybrid mode, combines the delay-based and loss-based modes to improve congestion detection.

In order to facilitate our study, we chose four well-known congestion control variants, and we classify them according to the two criteria cited above:

 - NewReno [3]: This variant is designed as the standard TCP congestion control approach. It uses the AIMD approach with the loss-based mode. Two mechanisms are employed immediately following congestion detection: fast retransmit and fast recovery [14]. Fast retransmit consists of performing a retransmission of what appears to be the missing packet (i.e. when receiving 3 duplicate ACKs), without waiting for the retransmission timer to expire. After the fast retransmit algorithm sends this packet, the fast recovery algorithm governs the transmission of new data until a non-duplicate ACK arrives. The reason for using fast recovery is to allow the continual sending of packets when the receiver is still receiving packets, even if some packets are lost.

- Vegas [4]: This non-AIMD variant is an Additive Increase Additive Decrease (AIAD) variant. It is a delay-based variant that accurately estimates *RTT* for every sent packet and adjusts *cwnd* size based on actual throughput and expected throughput. If *RTT* increases, *cwnd* decreases by one MSS, and vice versa. Vegas is the smoothest TCP congestion control variant [15]; it is able to allocate a fair share of bandwidth with minimal packet loss events.

- Illinois [5]: This is a TCP loss-delay-based congestion variant that employs a particular classification of the AIMD approach, C-AIMD, which involves a concave window size curve. Packet loss is used for primary congestion inference to determine the direction (increase or decrease) of *cwnd*, with a delay for secondary congestion inference to adjust the value of the window size change. More precisely, when the average queueing delay is small (small increase of *RTT*), the sender supposes that the congestion is not imminent and specifies a large additive increase α and small multiplicative decrease β. In the opposite case, when the average queuing delay is large (large increase of *RTT*), the sender supposes that the congestion is imminent and selects a small α and large β. Illinois measures *RTT* for each received ACK to update α and β. Moreover, it retains the same fast recovery and fast retransmit phases as NewReno. Illinois was designed for high-speed and high-latency networks, where the bandwidth-delay product is relatively high. Consequently, it enables higher throughput than NewReno.

- Cubic [6]: This variant is loss-based, but it uses a non-AIMD approach. A cubic function is used to increase the *cwnd* in the congestion avoidance phase immediately after the fast recovery phase, and a multiplicative decrease approach is used to update the *cwnd* after congestion event detection. The cubic function has a concave region followed by a convex region. The plateau between the two regions, or the inflexion point (denoted by W_{max}), corresponds to the window size just before the last congestion event. The cubic function enables a slow growth around W_{max} to enhance the stability of the bandwidth, and enables a fast growth away from W_{max} to improve scalability of the protocol. Upon receiving an ACK during the congestion avoidance phase at time t, Cubic computes the new value of *cwnd* corresponding to the cubic function at time t. As a consequence, Cubic uses the time instead of the *RTT* to increase the *cwnd*. Cubic employs a new slow-start algorithm called HyStart [8] (hybrid slow start), which finds a safe exit point to the slow start, the *ssthresh* value, at which the slow start can finish and safely move to congestion avoidance before *cwnd* overshoot occurs. HyStart employs the *RTT* delay increase and the inter-arrival time between consecutive ACKs to identify the safe exit point, and to modify the *ssthresh* value [8]. This variant does not make any change to the fast recovery and fast retransmit of standard NewReno. Cubic is the smoothest

loss-based congestion control variant [15]: it is characterized by a congestion window that falls less abruptly and that remains constant over a wide range of elapsed time. It is also designed for high-speed and high-latency networks.

For precise analysis, based on the descriptions of congestion control algorithm variants and their source code, we describe below the update of the congestion window size, *cwnd*, and the slow start threshold value, *ssthresh*, for different events:

- Congestion events: there are two cases
  o When a congestion event is detected, the Fast Recovery / Fast Retransmit (FR/FR) phase reduces the *ssthresh* value and sets the *cwnd* value to *ssthresh+3*, for the purpose of remaining in the congestion avoidance phase. The *ssthresh* value after a congestion event is updated as follows:

Algorithm 1 *ssthresh* update after a congestion event
1: **NewReno**: *ssthresh* = max(*cwnd/2*, *2 MSS*)
2: **Vegas:**   *ssthresh* = min(*ssthresh*, *cwnd - 1*)
3: **Illinois:**   *ssthresh* = max(*cwnd.(1-β)*, *2 MSS*)
4: **Cubic:**   *ssthresh* = max(*cwnd.(1-β)*, *2 MSS*)

where MSS is the maximum segment size, and β is the multiplicative decrease factor.

  o When the retransmission timeout expires before receiving any ACK of the retransmitted packet, *ssthresh* is reduced as indicated in Algorithm 1 , and *cwnd* is set to a small value and restarts from the slow start phase.

- Idle period: When the server sends a packet after an idle period that exceeds the retransmission timeout (RTO), *cwnd* and *ssthresh* are computed for the four congestion control variants as in Algorithm 2:

Algorithm 2 *cwnd* and *ssthresh* updates after idle period
1: *ssthresh* = max(*ssthresh*, *¾ cwnd*)
2: **for** i=1 to int(*idle/RTO*) **do**
3:   *cwnd* = max (min (*cwnd* , *rwnd*)/2, 1 *MSS*)
4: **end for**

In the HAS context, an idle period coincides with an *OFF* period between two consecutive chunks. An *OFF* period whose duration exceeds *RTO* is denoted by *OFF**.

Furthermore, we additionally want to emphasize that the rate of a TCP connection can be approximated, if we assume that transients due to slow start and fast recovery can be neglected, by $\min(cwnd, rwnd)/RTT$ [26], where *rwnd* is the TCP receive window indicated by the receiver and *RTT* is the round trip time between the sender and the receiver. Obviously, this approximation is valid for the four TCP congestion control variants described above. As we have shown, these variants employ different algorithms to modify the congestion window, *cwnd*, during the congestion avoidance phase and after congestion detection.

Accordingly, the generated rate as well as its variation over time are different from one variant to another.

2.2 Traffic shaping methods

Many studies have been conducted to improve HAS performance for cases in which several HAS clients are located in the same home network. The ON-OFF periods characterizing the HAS player in its steady state involve three substantial problems when HAS players are competing: player instability, unfairness between players, and bandwidth underutilization [9]. The cause of these problems is the inability to estimate the available bandwidth during the OFF period, because no data are being received.

Three types of solutions are proposed to improve HAS user experience: client-based, server-based, and gateway-based solutions. They differ with respect to the device in which the shaping solution is implemented. Below, we cite relevant methods for each type of solution:

- The client-based solution involves only the HAS client in order to reduce its OFF period durations. One of the recent client-based methods is proposed in the FESTIVE method [7]. It randomizes the events of chunk requests inside the player in order to reduce the periodicity of ON periods. Consequently, most of the incorrect estimations of bandwidth could be avoided when several HAS clients compete for bandwidth. However, this method is not efficient enough to prevent all incorrect estimations. In addition, it modifies the HAS player implementation, which is contradictory to our specifications described in the Introduction. Moreover, the client-based solution does not provide the coordination between HAS clients that is required to further improve bandwidth estimations and QoE.

- The server-based solution involves only the HAS server. It proceeds according to two steps: First, finding the optimal quality level for each provided HAS flow, and second, shaping the sending rate of the HAS server according to the encoding rate of this level. In [25], the authors propose a server-based method: it consists in detecting the oscillations between quality levels on the server side and deciding which optimal quality level must be selected. Although this method improves the QoE, it cannot conveniently respond to the typical use cases of several concurrent HAS clients that do not share the same HAS server: the shared link is on the HAS client side. Moreover, this server-based solution requires an additional processing task, which becomes burdensome and costly when many HAS clients are demanding video contents from the same HAS server. In addition, the server-based solution is unable to acquire information about the other competing flows with their corresponding HAS clients. Hence, the selection of the optimal quality level at the server is a vague estimation. This estimation is less accurate than a quality level selection based on a sufficient knowledge about the access network of the corresponding HAS client(s).

- The gateway-based solution that consists of applying the HAS traffic shaping in the gateway is more convenient than client-based and server-based solutions; in fact, the gateway can acquire information about the HAS traffic of all clients of the same home network, which is not possible either at the server or at the client. In addition, the gateway-based solution is able to perform traffic shaping without inducing any modification of HAS implementation code either in the server or the client. Hence, in this paper, our evaluations only consider the gateway-based shaping solution. For the gateway-based solution, the authors assumed that the home gateway can intercept the manifest file during the HAS session initialization and can obtain the characteristics of the available video quality levels of every session. This solution introduces a bandwidth manager in the gateway that defines a shaping rate for each connected active HAS client in the home network. The bandwidth manager should be able to update the number of active connected HAS clients in the home gateway by sniffing the SYN and FIN flags in TCP packets. Therefore, the difference between the gateway-based methods is the manner in which they shape the bandwidth for each HAS session. The two main gateway-based methods found in the literature and used in our comparative study are HTBM [1] and RWTM [2]. They are briefly described in the following:

2.2.1 HTBM

HTBM uses the Hierarchical Token Bucket (HTB) queuing discipline to shape the HTTP adaptive streams. HTB is integrated in Linux with the *traffic controller* tool of the *iproutes2* utility package. It uses one link, designated as the parent class, to emulate several slower links, designated as the children classes. Different types of traffic may be served by the emulated links. HTB is exploited by the bandwidth manager of HTBM in order to define a child class for each HAS session. HTB also employs the *tokens and buckets* concept, combined with the class-based system for better control over traffic and for shaping in particular [16]. A fundamental component of the HTB queuing discipline is the *borrowing mechanism*: children classes borrow tokens from their parent once they have exceeded the shaping rate. A child class will continue to try to borrow until it reaches a defined threshold of shaping, at which point it will begin to queue packets that will be transmitted when more tokens become available.

Accordingly, using HTBM enables the shaping of the HTTP adaptive streams for each HAS session in the gateway, as indicated by the bandwidth manager, by merely

4

delaying packets that are received from the HAS server. The authors of [1] indicate that HTBM improves the user's QoE; it improves the stability of video quality level, the fidelity to optimal quality level, and the convergence speed.

2.2.2 RWTM

The second shaping method, Receive Window Tuning Method (RWTM), is a gateway-based shaping method that was proposed in [2]. It is implemented in the TCP layer and is based on TCP flow control at the receiver side. Indeed, during a TCP session, each receiver specifies the maximum number of bytes that it is able to buffer. This value is called the receiver's advertised window, denoted by *rwnd*, and its size is specified in the *rwnd* field in the header of each TCP packet sent from the receiver to the sender. The sender receives the *rwnd* size from the receiver and limits its sending window, *W,* so that the number of packets sent in each *RTT* does not exceed *rwnd; W=min(rwnd, cwnd)*.

The RWTM method consists of modifying, in the gateway, the *rwnd* field of each TCP ACK packet sent from a HAS client *C* to its HAS server *S* in order to limit the sending rate of the HAS server to *rwnd/RTT_{C-S}*. RWTM uses the defined shaping rate of the bandwidth manager for each connected active HAS client, and estimates the RTT_{C-S} value to compute the next value of *rwnd*. We note that the estimation of RTT_{C-S} is accomplished using only TCP ACK packets sent from HAS clients to the HAS server by using passive estimation. The *rwnd* is computed once for each ON period. RWTM was tested in [2] and [21], and results indicated that RWTM enhances the user's QoE: it improves the stability, the fidelity, and the convergence speed.

In [21], we showed that RWTM outperforms HTBM when using the Cubic variant as congestion control on the server side. However, due to the dissimilarity of TCP variants, an extended evaluation using other variants and additional scenarios will give us a better understanding of the interaction between shaping methods and TCP variants.

3. METHODOLOGY AND EXPERIMENTAL IMPLEMENTATION

In this section, we provide a description of the metrics used to measure performance, the scenarios that cover many operating conditions, and the framework that has been developed to emulate our use case.

3.1 Performance metrics

We define three metrics in this section that we use to evaluate the QoE and to understand how each combination behaves. To do so, we present in Table 1 the main parameters that are used to define the metrics.

Table 1. Description of parameters

Parameter	Description
i	Discrete time index
$L_C(i)$	Video quality level index of client *C* at time *i*.
$L_{C,opt}(i)$	Theoretical optimal value of $L_C(i)$
$Q_C(i)$	Video encoding bitrate of client *C* at time *i*

We note that the optimal quality level value, $L_{C,opt}(i)$, corresponds to the quality level that the client C should select at time *i* under the shaping rate defined by the bandwidth manager. This shaping rate is chosen in a manner that ensures the fairest share of the available home bandwidth between clients with prioritization to achieve the maximum use of the available home bandwidth. This entails that some clients could have a higher quality level than others when their fair share of available home bandwidth is not sufficient to maximize the use of the available home bandwidth. Below, we define analytically three performance metrics that describe the three criteria of QoE mentioned in the Introduction:

3.1.1 Video quality level stability

Many research studies indicate that HAS users are likely to be sensitive to frequent and significant quality level switches [22, 23]. We use the instability metric, $IS_C(K)$, which measures the instability for client *C* for a *K*-second test duration in conformity with its description in [7] as the following equation:

$$IS_C(K) = \frac{\sum_{i=0}^{K-1} |Q_C(K-i) - Q_C(K-i-1)| \times w(i)}{\sum_{i=1}^{K} Q_C(K-i) \times w(i)} \quad (1)$$

$IS_C(K)$ is the weighted sum of all encoding bitrate switching steps observed within the last *K* seconds divided by the weighted sum of the encoding bitrates selected in the last *K* seconds. The lower the $IS_C(K)$ value is, the higher the stability of the video quality level is.

More precisely, this formula uses the encoding bitrates of the selected quality levels over time, $Q_C(i)$, instead of the quality level index over time, $L_C(i)$. In fact, the absolute difference between two encoding bitrates that are displayed on the client side during two successive seconds, $K - i - 1$ and $K - i$, and denoted by $|Q_C(K-i) - Q_C(K-i-1)|$, gives more significant indication of the observed video quality change than when using the absolute difference between the quality level indexes. Hence, we can offer an adequate representation of the user expectation.

Moreover, in this formula, the authors of [7] use the weight function $w(i) = K - i$ in order to add a linear penalty to more recent quality level switches. In fact, their justification is that the switching of quality level is becoming more disturbing for users' experience when the video playback position is far from the beginning of the video stream.

3.1.2 Fidelity to optimal quality level

In [7], the authors define two additional goals to achieve within our use case: 1) fairness between players: players should be able to converge to an equitable allocation of network resources; 2) efficiency among players: players should choose the highest feasible quality levels to maximize the user's experience. Furthermore, in [9], the authors address the bandwidth underutilization issue that may prevent the possible improvement of QoE. So,

maximizing the use of bandwidth can be considered as a QoE criterion. Accordingly, in order to provide one formula that satisfies these three criteria, we define our metric called *infidelity to optimal quality level*.

The infidelity metric, $IF_C(K)$, of client C for a K-second test duration, measures the duration of time over which the HAS client C requests optimal quality:

$$IF_C(K) = \left\{ \frac{\sum_{i=1}^{K} \theta_C(i)}{K} \middle| \theta_C(i) = \begin{cases} 1, if\ L_C(i) \neq L_{C,opt}(i) \\ 0, else \end{cases} \right\} \quad (2)$$

The lower the $IF_C(K)$ value is, the higher the fidelity to optimal quality is.

Here, we note that the theoretical optimal quality level $L_{C,opt}(i)$ aims to resolve the dilemma between the two criteria of maximum use and fair share of bandwidth between HAS players. In fact, considering that only the fair share of bandwidth may cause bandwidth underutilization, in some cases it may leave some residual bandwidth allocated to nobody. Hence, based on the optimal quality level, the value of the infidelity metric is representative of user expectation.

3.1.3 Convergence speed
The convergence speed metric was previously defined in [1]. We provide an analytical definition as follows:

$$V_{C,T}(K) = \left\{ \min_{d \in \{1,\dots K\}}(d) \middle| L_C(i) = L_{C,opt}(i) \ \forall\ i \in [d, d+T] \right\} \quad (3)$$

This metric is the time that the player of HAS client C takes to reach and remain at the optimal quality level for at least T seconds during a K-second test duration. The reason of selecting this criterion for evaluating the QoE in our use case is observations made in [1], [2], [9], and [21]: they show that when HAS players compete for bandwidth, the convergence to optimal quality level may take several seconds or may be very difficult to be achieved. Accordingly, the speed of this convergence is a valuable QoE criterion for our evaluations. The lower the $V_{C,T}(K)$ value is, the faster the convergence to the optimal quality level is.

Additionally, we define two other metrics (*CNG* and fr_{OFF*}, described below) that enable us to measure the reaction of home gateway and HAS players.

3.1.4 Congestion rate
The congestion detection events influence to an extreme degree both the QoS and QoE of HAS because the server decreases its sending rate after each congestion detection. Hence, by analyzing the code description of the four TCP congestion control algorithms (NewReno, Vegas, Illinois, and Cubic), we found that the congestion event appears when the value of parameter *slow start threshold* (*ssthresh*) decreases (see Algorithm 1). Hence, we define a metric called *congestion rate*, denoted by $CNG_{C-S}(K)$, that

computes the rate of congestion events that are detected on the server side, corresponding to the HAS flow between client C and server S during a K-second test duration as shown in equation (4):

$$CNG_{C-S}(K) = \frac{D_{C-S}^{ssthresh}(K)}{K} \times 100 \quad (4)$$

where $D_{C-S}^{ssthresh}(K)$ is the number of times the *ssthresh* has been decreased for the C-S HAS session during the K-second test duration.

3.1.5 Frequency of OFF* periods per chunk
This metric is important to measure the frequency of *OFF** periods. An *OFF* period whose duration exceeds *RTO* is denoted by *OFF** (as indicated in Subsect. 2.1). This frequency is equal to the total number of *OFF** periods divided by the total number of downloaded chunks. This metric is denoted by fr_{OFF*}.

For result analysis, we use the QoE metrics to quantitatively discuss the user's experience, and use *CNG* and fr_{OFF*} metrics to explain the performance of each combination of traffic method and congestion variant.

3.2 Scenarios
We define five scenarios that are typical of concurrence between HAS clients in a same home network (scenarios 1, 2, and 3), and how the HAS client reacts when some changes occur (scenarios 4 and 5):

1. Both clients start to play simultaneously and continue for 3 minutes. This scenario illustrates how clients compete.
2. Client 1 starts to play, the second client starts after 30 seconds, and both continue together for 150 seconds. This scenario shows how the transition from one client to two clients occurs.
3. Both clients start to play simultaneously, client 2 stops after 30 seconds, and client 1 continues alone for 150 seconds. This scenario shows how a transition from two clients to one takes place.
4. Only one client starts to play and continues for 3 minutes. At 30 seconds, we simulate a heavy congestion event with a provoked packet loss of 50% of the received packets at the server over a 1-second period. This scenario shows the robustness of each combination against the congestions that are induced by external factors, such as by other concurrent flows in the home network.
5. Only one client is playing alone for 3 minutes. We vary the standard deviation value of RTT_{C-S} (round trip time between the client and the server) for each set of tests. This scenario investigates the robustness against RTT_{C-S} instability.

The test duration was selected to be 3 minutes to offer sufficient delay for players to stabilize.

6

3.3 General framework

We propose a testbed architecture presented in Figure 1 that emulates our use case described in the Introduction. The choice of only two clients is sufficient to show the behavior of concurrence between many HAS flows in the same home network.
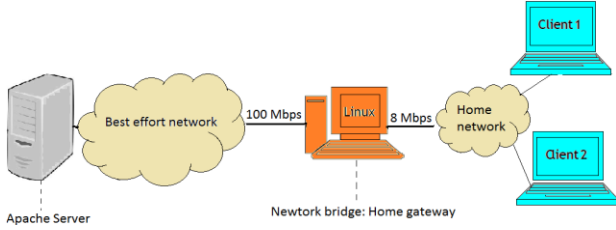

Figure 1. Architecture of the testbed

In this section, we describe the configurations of each component presented in Figure 1:

- HAS clients

We used two Linux machines as HAS clients. We developed an emulated player in each client that reproduces the behavior of the HAS player without decoding and displaying a video stream. The playback buffer size is specified to be 15 chunks, and the chunk duration is 2 seconds. In [18], the authors indicate that the bitrate adaptation algorithm depends on bandwidth estimation and playback buffer occupancy. Furthermore, players also define an aggressiveness level, as described in a previous study [19]. For example, the Netflix player is more aggressive than the Smooth Streaming player [19]. An aggressive player enables the user to ask for a video quality level that is slightly higher than the estimated available bandwidth. Moreover, aggressiveness is important for minimizing the "downward spiral effect" phenomenon [20]. This phenomenon consists of underestimating the available bandwidth, which leads to a lower video quality level selection. Accordingly, taking into consideration [18], [19], and [20], we used a bitrate adaptation algorithm based on bandwidth estimation in which we define an aggressiveness $\rho_C(t)$ at time t that depends on playback buffer occupancy as follows:

$$\rho_C(t) = \sigma.R_C(t)/B_C \qquad (5)$$

where $R_C(t)$ is the filling level of the playback buffer at time t, B_C is the size of the playback buffer of client C, and σ is the aggressiveness constant. The fuller the playback buffer is, the closer to σ the aggressiveness is.

All tests use a HAS player with an aggressiveness constant of $\sigma=0.2$. This enables the HAS player to add a maximum of 20% to its available BW estimation.

- Home network

In the modeled home network, the clients are connected directly to the gateway. The total download bitrate, or home available bandwidth, is limited to 8 Mbps. We choose this value because it is lower than twice the video encoding bitrate of the highest quality level. Accordingly, two clients

in the home network cannot select the highest quality level at the same time. In this case, one client should select quality level n° 4 and the other should select the quality level n° 3 as optimal qualities. We do not test a use case in which two clients have the same optimal quality level, because this is a very specific case, and dissimilarity between optimal quality levels is more general.

- Home gateway

The emulated home gateway consists of a Linux machine configured as a network bridge to forward packets between the home network and the best effort network.

We emulate the queuing discipline of the home gateway by using the Stochastic Fairness Queueing discipline (SFQ) [24]. SFQ is a classless queuing discipline that we configured using the Traffic Controller emulation tool (*tc*). SFQ schedules packets based on flow identification (the source and destination IP addresses and the source port) and injects them into hash buckets during the enqueuing process. Each bucket represents a unique flow. Additionally, SFQ employs Round Robin fashion for dequeuing packets by taking into consideration the bucket classification. The goal of using buckets for enqueuing and Round Robin for dequeuing is to ensure fairness between flows so that the queue is able to forward data in turn and prevents any single flow from drowning out the remaining flows. We also configured SFQ in order to support the Drop Tail queue management algorithm when the queue becomes full. Hence, this configuration of the queuing discipline is classified as a Drop Tail class. The queue length of SFQ, which is indicated by parameter *limit* within the *tc* tool, is set to the bandwidth-delay product.

In the gateway, we implemented a bandwidth manager that selects a shaping rate for each connected active HAS client in a manner such that each client should attain its optimal quality level described in Subsect. 3.1. The shaping rate for each client was chosen as indicated in [1] and [2]; it is 10% higher than the encoding bitrate of the optimal quality level for each client. The two shaping methods HTBM and RWTM are implemented in the gateway, and they shape bandwidth in accordance with the decisions of the bandwidth manager.

- Best effort network

The best effort network is characterized by the presence of network devices to route packets. The round trip time $RTT_{C-S}(t)$ in a best effort network is modeled as follows [10]:

$$RTT_{C-S}(t) = a_{C-S} + q(t)/\varsigma \qquad (6)$$

where a_{C-S} is a fixed propagation delay between client C and server S, $q(t)$ is the queue length of a single congested router (the home gateway in our use case), and ς is the transmission capacity of the router. $q(t)/\varsigma$ models the queuing processing delay. To comply with equation (6), we used the normal distribution with a mean value a_{C-S} and a standard deviation equal to $0.07.a_{C-S}$. The standard

deviation emulates the queuing processing delay $q(t)/\varsigma$. This emulation is accomplished by using the "*netem delay*" parameter of the *traffic controller* tool in the gateway machine interface.

- HAS server

The HAS server is modeled by an HTTP Apache Server installed on a Linux machine operating on Debian version 3.2. We can change the congestion control variant of the server by varying the parameter *net.ipv4.tcp_congestion_control.* All tests use five video quality levels denoted by 0, 1, 2, 3, and 4. Their encoding bitrates are constant and equal to 248 kbps, 456 kbps, 928 kbps, 1,632 kbps, and 4,256 kbps, respectively. HTTP version 1.1 is used to enable a persistent connection.

4. RESULTS

In this section, we compare the different combinations of TCP congestion control variants in the server and shaping methods in the gateway in the five scenarios. Altogether, we evaluate eight combinations: four TCP congestion control variants combined with two shaping methods. We evaluate QoE by discussing the QoE metrics *IS, IF,* and *V*. We also use the *CNG* and *fr_{OFF*}* metrics to observe how each combination reacts. For each scenario, we repeated each test 60 times and we computed an average value of each metric. The number of 60 runs is justified by the fact that the difference of the average results obtained after 40 runs and 60 runs are lower than 6%. This observation was verified for all scenarios. Accordingly, 60 runs are sufficient to achieve statistically significant results.

This section is organized as follows. First, we begin by evaluating performance in scenario 1, and we analyze the variation of *cwnd* for each combination. Second, we evaluate the performance of scenarios 2 and 3 to study the effect of transition from one to two clients (and vice versa) on the performance of each studied combination. Third, we present the performance of scenario 4 to measure the robustness of the combinations against induced congestions. Fourth, we study scenario 5 to measure the robustness against the instability of RTT_{C-S} for each combination. Finally, we discuss all results by presenting a summary of observations and defining the combination that is suitable for each particular case.

4.1 Scenario 1

In this scenario, two clients are competing for BW and are playing simultaneously. The available home bandwidth permits only one client to have the highest quality level, n° 4. We make the assumption that the client who gets the highest quality level n° 4 is identified as client 1. Optimally, the first player in our use case should obtain quality level n° 4 with an encoding bitrate of 4,256 kbps, and the second player should have quality level n°3 with an encoding bitrate of 1,632 kbps.

In this section, we present our evaluation results and discuss them. Then, we analyze the *cwnd* variation for each combination in order to understand the reason for the observed results.

4.1.1 Measurements of performance metrics

The average values of QoE metric measurements for client 1 and client 2 are listed in Tables 2 and 3, respectively.

Table 2. QoE for client 1 in scenario 1

Performance metric	Shaping method	TCP congestion control variant			
		NewReno	Vegas	Illinois	Cubic
Instability (%) $IS_1(180)$	W/o*	4.95	2.15	8.35	7.47
	HTBM	1.89	1.08	1.56	1.86
	RWTM	1.69	4.10	1.88	1.63
Infidelity (%) $IF_1(180)$	W/o	41.33	52.31	74.14	50.46
	HTBM	49.57	47.81	7.75	20.45
	RWTM	45.87	32.24	6.17	5.02
Convergence speed (s) $V_{1,60}(180)$	W/o	100.93	102.11	174.13	145.03
	HTBM	101.83	87.11	21.10	52.06
	RWTM	94.51	104.00	24.22	19.55

Table 3. QoE for client 2 in scenario 1

Performance metrics	Shaping methods	TCP congestion control variants			
		NewReno	Vegas	Illinois	Cubic
Instability (%) $IS_2(180)$	W/o	5.82	3.06	7.85	5.82
	HTBM	1.17	0.95	1.05	1.15
	RWTM	1.09	0.95	1.03	1.13
Infidelity (%) $IF_2(180)$	W/o	26.64	70.77	39.27	36.33
	HTBM	4.72	3.62	4.21	4.47
	RWTM	2.49	2.30	2.47	2.61
Convergence speed (s) $V_{2,60}(180)$	W/o	96.25	137.01	126.33	92.81
	HTBM	12.41	6.95	9.73	13.26
	RWTM	6.73	5.03	6.54	8.95

Our first overall observation is the large dissimilarity between QoE measurements of the different combinations. This observation is a valuable result that confirms that each combination induces a change of HAS player behavior. Consequently, using HAS traffic shaping without taking into consideration the TCP congestion control employed in the HAS server cannot guarantee a good user experience; hence, the prominence of our proposed work.

The results show that traffic shaping considerably improves the QoE metric measurements for a majority of cases, especially for instability, which is largely reduced (e.g. a reduction of instability rate by a factor of 2.6 from 4.95% to 1.89% when employing HTBM with NewReno, and a reduction by a factor of 4.5 from 7.47% to 1.63% when employing RWTM with Cubic, as shown in Table 2). Furthermore, RWTM shows better performance than HTBM in the majority of cases. Moreover, client 2 always has better performance than client 1 with both shaping methods: the reason is that the optimal quality level of client 2 (i.e. quality level n° 3) is lower than that of client 1 (i.e. quality level n° 4): obviously, the quality level n° 3 is easier to achieve. In addition, the gap between the QoE metric measurements of the two shaping methods is higher for client 1 than client 2: For example, when considering the Cubic variant, the gap of infidelity rate of client 1 between RWTM and HTBM is 15.43% (5.02% vs.

20.45%); this is higher than that of client 2, which is equal to 1.86% (2.61% vs. 4.47%). Consequently, the dissimilarity of performance between different combinations is more visible for client 1. For this reason, we limit our observation to client 1 in the remaining text of this subsection.

Concerning the QoE measurements, based on Table 2, we present the most important observations related to client 1:

- Combining NewReno or Vegas variants with HTBM or RWTM does not improve the QoE. Additionally, these four combinations have high infidelity value (near 50%) and very high congestion speed value (around 90 ~100 ms), but a low value of instability. These values indicate that the player was stable at a low quality level during the first half of the test duration and has difficulties converging to its optimal quality level.

- HTBM has better QoE with Illinois than with Cubic: it is slightly more stable, 16% more faithful to optimal quality, and converges 2.4 times faster.

- RWTM has better QoE with Cubic than with Illinois: it is slightly more stable, slightly more faithful to optimal quality level, and converges 1.24 times faster.

In order to be more accurate in our analysis, we use the two defined metrics: the frequency of *OFF** periods per chunk, fr_{OFF*}, and the congestion rate, *CNG*. In Table 4, we present the average value over 60 runs for each metric and for each combination, related to client 1 and scenario 1.

Table 4. fr_{OFF*} and *CNG* for client 1 in scenario 1

Metric	Shaping method	TCP congestion control variant			
		NewReno	Vegas	Illinois	Cubic
CNG	W/o	46.13	43.00	66.11	85.65
	HTBM	44.06	40.50	58.68	191.72
	RWTM	0.10	8.26	0.76	1.11
fr_{OFF*}	W/o	0.42	0.35	0.27	0.40
	HTBM	0.31	0.32	0.06	0.16
	RWTM	0.32	0.41	0.24	0.24

RWTM presents a negligible congestion rate, while HTBM has a very high rate of congestion, especially when the Cubic variant is used. Moreover, HTBM reduces the frequency fr_{OFF*} better than RWTM, mainly with Illinois and Cubic. These results have a direct relationship to the shaping methods described in Subsect. 2.2:

- HTBM was designed to delay incoming packets, which causes an additional queuing delay. In all of the tests, we verified that HTBM induces a queueing delay of around 100 ms in scenario 1 for client 1. On one hand, this delay causes an increase of congestion rate because it increases the risks of queue overflow in the gateway, even when the QoE is good, such as with Cubic or Illinois variants. The dissimilarity of congestion rate between congestion controls variants is

investigated in the next Subsect. 4.1.2. On the other hand, the RTT_{C-S} value also jumps from 100 ms to 200 ms, which increases the retransmission timeout value, *RTO*, to approximately 400 ms, hence reducing *OFF** periods. The fr_{OFF*} of HTBM is noticeably lower than RWTM and the case without shaping (W/o). In addition, the assertion "the higher the QoE metric measurement, the lower the fr_{OFF*} value" seems to be valid; for example, HTBM presents better QoE with Illinois than with Cubic, and fr_{OFF*} is lower with Illinois than with Cubic.

- Nevertheless, RWTM was designed to limit the value of the receiver's advertised window, *rwnd*, of each client. Therefore, no additional queuing delay is induced by RWTM. Hence, the congestion rate is very low. Additionally, the RTT_{C-S} estimation is performed only once per chunk. So, the *cwnd* value is constant during the ON period, even if RTT_{C-S} varies. In our configuration, the standard deviation of RTT_{C-S} is equal to 7 ms, i.e. $0.07.a_{C-S}$, as described in Subsect. 3.3. Consequently, eliminating *OFF** periods will not be possible. Instead, the fr_{OFF*} value will be bounded to a minimum value that characterizes RWTM when the QoE measurements are the most favorable. When testing with the four congestion control variants, this fr_{OFF*} value is equal to 0.24 for the selected standard deviation. This means that RWTM can guarantee, in the best case, one *OFF** period every 4.17 chunks. This frequency is useful, and will be discussed in the next subsection and in further detail in scenario 5.

4.1.2 Analysis of cwnd variation

To explain the results of scenario 1, we used the *tcp_probe* module in the HAS server. This module shows the evolution of the congestion window, *cwnd*, and the slow start threshold, *ssthresh*, during each run. For each combination, we selected a run the performance values of which are the nearest to its average values of Tables 2 and 4, i.e. instability *IS*, infidelity *IF*, convergence speed V, frequency of *OFF** periods per chunk fr_{OFF*}, and congestion rate *CNG*. Then, we present their *cwnd* and *ssthresh* evolution in Figures 2 through 9. We also indicate the moment of convergence by a vertical bold dotted line. We observed that this moment corresponds to the second from which the TCP congestion control is often processing under the congestion avoidance phase; i.e. when *cwnd > ssthresh*. In addition, from the moment of convergence, we observe that *ssthresh* becomes more stable and is practically close to a constant value.

Figure 2 shows that the combination NewReno with HTBM cannot guarantee convergence to the optimal quality level. The congestion rate is not very high compared with other

TCP congestion variants. After 50 seconds, *cwnd* was able to reach the congestion avoidance phase for short durations, but the continuous increase of *cwnd* with the additive increase approach caused the detection of congestion. Moreover, the multiplicative decrease approach after congestions employed by NewReno was very aggressive; in effect, as described in Subsect. 2.2, the new *cwnd* value will be reduced by half (more precisely, to *cwnd/2 + 3 MSS* following the FR/FR phase) and *ssthresh* will also be reduced to *cwnd/2*. This aggressive decrease prevents the server from rapidly reaching a desirable *cwnd* value and, as a consequence, prevents the player from correctly estimating the available bandwidth and causes a lower quality level selection. Furthermore, the fr_{OFF*} value was relatively high (around 0.3 *OFF** period per chunk), which is more than twice that of the Illinois and Cubic variants. This value is also caused by the multiplicative decrease approach that generates a lower quality level selection. Due to the shaping rate that adapts the download bitrate of the client to its optimal quality level, the chunk with a lower quality level will be downloaded more rapidly, which results in causing more frequent *OFF** periods. For this reason, the player was not able to stabilize on the optimal quality level, resulting in a poor QoE.
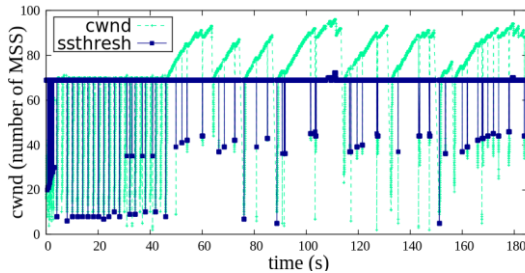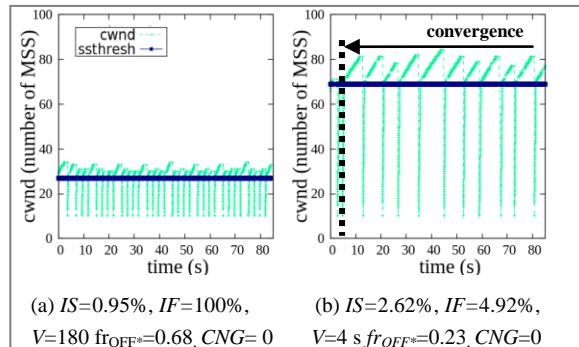

Figure 2. *Cwnd* **variation of {NewReno HTBM}**
IS=5.48%, *IF*=35.68%, *V*=180s, fr_{OFF*}=0.2, *CNG*=43.33

When combining NewReno with RWTM, we observed that test results diverged and could be classified into two categories: those with an infidelity value of 100% and that do not converge (Figure 3(a)), and those with a low value of infidelity and that converge rapidly (Figure 3(b)). In both figures, *ssthresh* is always invariable. Both figures have no congestion events, which is due to the use of RWTM.

(a) *IS*=0.95%, *IF*=100%, *V*=180 fr_{OFF*}=0.68, *CNG*= 0

(b) *IS*=2.62%, *IF*=4.92%, *V*=4 s fr_{OFF*}=0.23, *CNG*=0

Figure 3. Cwnd variation of {NewReno RWTM}

The *OFF** periods are more frequent in Figure 3(a) (fr_{OFF*} = 0.68) than in Figure 3(b) (fr_{OFF*} = 0.23). Although both figures present a constant value of *ssthresh,* we observe that the only difference between them is the initial value of *ssthresh*. Figure 3(a) has a lower value of *ssthresh* than Figure 3(b): 27 *MSS* vs. 69 *MSS.* The additive increase approach of NewReno during the congestion avoidance phase prevents the server from rapidly increasing the *cwnd* value during ON periods. Therefore, the player was not able to reach the optimal quality level n° 4 at any time. The cause of the dissimilarity between the initial values of *ssthresh* in the two figures is explained in [17]. Some implementations of NewReno use the size of the receiver's advertised window, *rwnd*, to define the initial value of *ssthresh,* but in fact, this value may be arbitrarily chosen. Accordingly, the combination of NewReno with RWTM could have high QoE if the initial value of *ssthresh* is well-chosen.

When combining Vegas with HTBM, we obtain a *cwnd* variation, as shown in Figure 4. The convergence moment (at 87 s in Figure 4) occurs when *cwnd* becomes often set higher than *ssthresh* (i.e. TCP congestion control is often processing under the congestion avoidance phase) and *ssthresh* is often set at the same value. We can observe the additive increase and additive decrease aspect of *cwnd* in the congestion avoidance phase after convergence. The additive decrease of *cwnd* involved in Vegas is caused by the queuing delay increases resulting from HTBM. This additive decrease has the advantage of maintaining a high throughput and reducing the dropping of packets in the gateway. Therefore, the congestion rate, *CNG,* is relatively low because it is reduced in Figure 4 from around 75 congestion events per 100 seconds to only 15. The additive decrease also has the advantage of promoting convergence to the optimal quality level, unlike multiplicative decrease. As a result, the delay-based aspect with the additive decrease approach improves the stability of the HAS player after convergence. In contrast, Vegas uses a slightly low value of *ssthresh* (60 *MSS*) and employs the additive increase approach for *cwnd* updates during the congestion avoidance phase. As a consequence, the server cannot rapidly increase the *cwnd* value during the ON period, which results in slow convergence. Therefore, the player
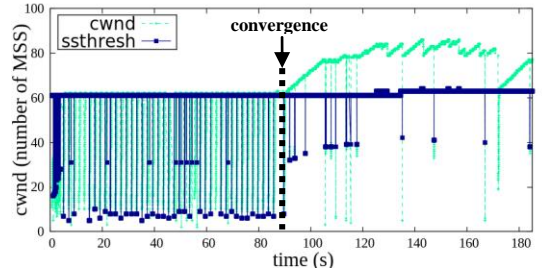

Figure 4. *Cwnd* **variation of {Vegas HTBM}**
IS=1.31%, *IF*=46.74%, *V*=87 s, fr_{OFF*}=0.4, *CNG*=46.11

was not able to reach the optimal quality level n° 4 at any time before the moment of convergence. Consequently, the frequency of the *OFF** period increases before the convergence moment; hence, the high value of fr_{OFF*}.

The performance worsens when Vegas is combined with RWTM. As presented in Figure 5, the player was not able to converge. Instead, we observed many timeout retransmissions characterized by *ssthresh* reduction and *cwnd* restarting from *slow start*. The timeout retransmissions are generated by Vegas when only a duplicate ACK is received and the timeout period of the oldest unacknowledged packet has expired [4]. Because of that, Vegas generates more timeout retransmissions than NewReno. Hence, the *CNG* value is worse than in the other combinations of RWTM. Moreover, *OFF** periods are frequent during the first 45 seconds, because the player requests quality level n° 3. Subsequently, *OFF** periods become less frequent (they occur only at 79, 125, 138, 150, 165, and 175 s) because the player was able to switch to an optimal quality level (n° 4). Hence fr_{OFF*} related to the whole test duration is equal to an acceptable value (0.29 *OFF** period per chunk). The player becomes able to request the optimal quality level n° 4 predominantly in the second period (after 45 seconds), but it is incapable of being stable for more than 60 seconds because of the retransmission timeout events.
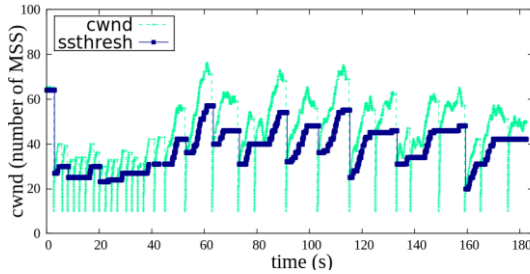

Figure 5. *Cwnd* **variation of {Vegas, RWTM}**
*IS***=5.32%,** *IF***=31.15%,** *V***= 180s,** fr_{OFF*}**=0.29**, *CNG***=6.11**

When we use the loss-delay-based variant Illinois, significant improvement of performance is observed with the two shaping methods:

In Figure 6, despite the rapid convergence, a high rate of congestions (that reduces the *ssthresh* and *cwnd* values but maintains the *cwnd* higher than *ssthresh*, as described in Algorithm 1) and timeout retransmissions (that reduces *ssthresh*, drops *cwnd*, and begins from the slow start phase) was recorded. Consequently, the frequent reduction of *ssthresh* was the cause of the high rate of *CNG*: in this example, *CNG* is equal to 51.11. *CNG* is higher than that recorded for NewReno. The cause is the high value of *ssthresh* of approximately 115 *MSS*. The variable *ssthresh* was able to rapidly return to a fixed value after retransmissions, due to the update of α and β using accurate RTT_{C-S} estimation (see Subsect. 2.1). As a consequence, *cwnd* restarts from the slow start phase after timeout

detection and rapidly reaches the high value of *ssthresh*. Hence, the HAS player converges despite high congestion. In addition, *OFF** periods were negligible, with only two periods after congestion. This is why fr_{OFF*} was very low (0.03). In the congestion avoidance phase, *cwnd* was able to increase and reach high values, even during short timeslots. This was due to the concave curve of *cwnd* generated by Illinois, which is more aggressive than NewReno. As a consequence, the player could be stabilized with optimal quality level n° 4.
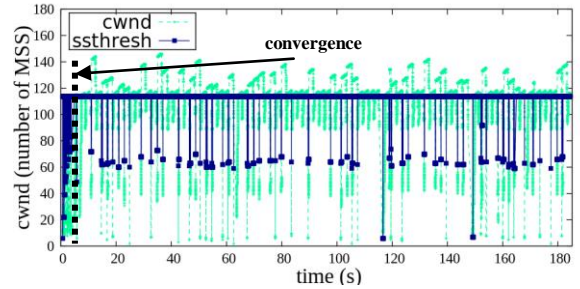

Figure 6. *Cwnd* **variation of {Illinois, HTBM}**
*IS***=2.00%,** *IF***=7.66%,** *V***=5s,** fr_{OFF*}**=0.03**, *CNG***=51.11**

When using RWTM with Illinois, the player converges, as presented in Figure 7. The congestion rate is very low (*CNG=0.55*), but congestions are caused by the aggressiveness of Illinois (the concave curve of *cwnd* in the C-AIMD approach) and its high *ssthresh* value (120 *MSS*). Congestions slow down the convergence speed and slightly reduce the QoE due to the multiplicative decrease approach of Illinois. As shown in Figure 7, one congestion event delayed the convergence time to 27 seconds. In addition, Illinois has the ability to select the suitable *ssthresh* value (110 *MSS* in Figure 7) that minimizes congestion events in the future, in spite of the sensitivity of RWTM to congestions. *OFF** periods still exist, but with low frequency (fr_{OFF*} = 0.22).
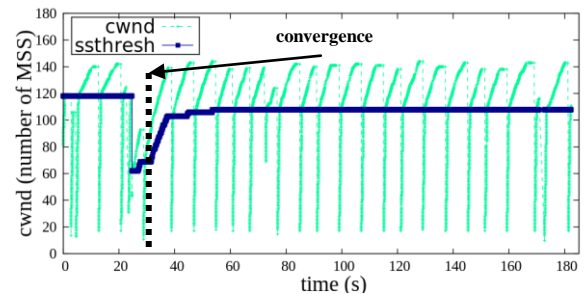

Figure 7. *Cwnd* **variation of {Illinois RWTM},**
*IS***=2.40%,** *IF***=5.47%,** *V***=27s,** fr_{OFF*}**=0.22**, *CNG***=0.55**

The Cubic variant yielded good performances with both shaping methods. The variations of *cwnd* when Cubic is combined with HTBM and RWTM are presented in Figures 8 and 9, respectively.

In Figure 8, the player converges tardily after a delay of 33 seconds. The cause is mainly the low value of *ssthresh* that is selected by the Cubic algorithm. As explained in Subsect. 2.1, the HyStart algorithm, implemented in Cubic, defines

this *ssthresh* in order to have a less aggressive increase of *cwnd*. The *ssthresh* becomes lower when the RTT_{C-S} increases. Knowing that HTBM increases RTT_{C-S} by introducing an additional queuing delay, HyStart decreases *ssthresh* to be approximately 57 *MSS*. This is why the player cannot upgrade to its optimal quality level n° 4 before convergence. The second cause is the multiplicative decrease approach of Cubic and the high rate of congestions caused by HTBM. This second cause makes the convergence to optimal quality level more difficult because the server is not able to increase its reduced congestion window *cwnd* during the ON period, as it should be increased.

After convergence, many congestions were recorded, and *OFF** periods were negligible. The *ssthresh* becomes more stable around 75 *MSS*: this is well-set by the HyStart algorithm. This enhances stability in the congestion avoidance phase with a more uniform increase of *cwnd*, as shown between 60 and 80 seconds in Figure 8. Furthermore, there is a set of large cubic curves with inflection points close to the *ssthresh* value. The variable *cwnd* is more present in the convex region, which is more aggressive when moving away from the inflection point.
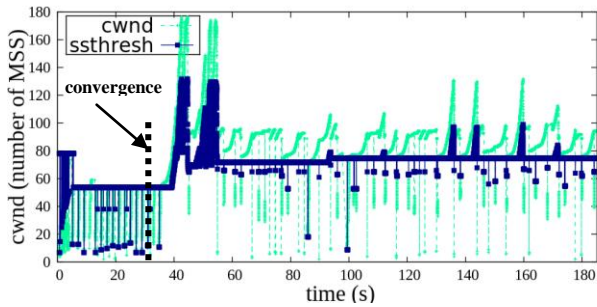

Figure 8. *Cwnd* **variation of {Cubic HTBM},**
***IS*=1.98%, *IF*=19.03%, *V*=33s, fr_{OFF}*=0.16, *CNG*=186.11**

In Figure 9, the player converges rapidly in only 8 seconds. The *ssthresh* begins with a low value (60 *MSS*) for a few seconds during the buffering state, and then the HyStart algorithm implemented in Cubic rapidly adjusts the *ssthresh* value and enables the server to be more aggressive. Comparing with Figure 7, selecting a lower initial value of *ssthresh* is better for accelerating convergence, because otherwise there are more risks of congestion that slow down the convergence speed.

Congestions are infrequent: only two congestions are visible in Figure 9 at seconds 70 and 130, and they are resolved by fast retransmission in accordance with Algorithm 1 and by using Hystart. As a consequence, separated congestion events do not dramatically affect the performance, as when Illinois is used with RWTM (Figure 9). The Cubic algorithm chooses the inflection point to be around 140 *MSS,* which is much higher than the *ssthresh* value, so that the concave region becomes more aggressive

than the convex region. The *OFF** periods persist, even with Cubic, but with a low frequency: $fr_{OFF*}= 0.22$.
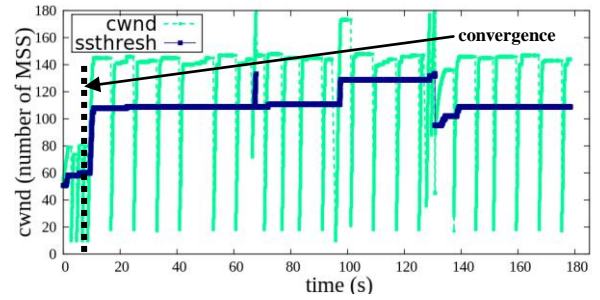

Figure 9. *Cwnd* **variation of {Cubic RWTM},** *IS*=1.78%,
IF=5.5%, *V*=8s, fr_{OFF*}=0.22, *CNG*=1.66

Accordingly, the Cubic variant is able to adjust its congestion window curve in different situations. When many congestions occur, the cubic curve becomes rather convex to carefully increase *cwnd*. When many *OFF** periods occur, the cubic curve becomes rather concave, and is thus more aggressive than the concave curve of Illinois in order to rapidly achieve the desired send bitrate and compensate for the reduction of the *cwnd* value. However, Cubic begins by estimating a low value of *ssthresh* that is adjusted over time by the HyStart algorithm, which is beneficial only when using RWTM as a shaping method. Using HTBM slows down convergence considerably and affects the infidelity metric.

4.2 Scenarios 2 and 3

In this section, we present the five performance measurements of client 1 for the first three scenarios described in Subsect. 3.2. We make the assumption that the optimal quality level of client 1 is n° 4. We do not present NewReno and Vegas variants because they demonstrated low performance. The average values of QoE metrics for client 1 in the first three scenarios are listed in Table 5, and the average values of *CNG* and fr_{OFF*} in the first three scenarios are listed in Table 6. Both tables show the total mean values (denoted by MV) over the three scenarios. MVs are the global performance values proposed for consideration to compare between different combinations.

Table 5. QoE for client 1 in scenarios 1, 2, and 3

TCP variant	Scenario	Performance metric					
		Instability (%)		Infidelity (%)		Convergence speed (seconds)	
		HTBM	RWTM	HTBM	RWTM	HTBM	RWTM
Cubic	1	1.86	1.63	20.45	5.02	52.06	19.55
	2	3.44	1.43	32.90	3.42	64.13	10.98
	3	2.19	1.63	18.49	4.81	34.65	14.34
	MV*	**2.49**	**1.56**	**23.95**	**4.42**	**50.28**	**14.96**
Illinois	1	1.56	1.88	7.75	6.17	21.10	24.22
	2	3.20	1.56	29.75	4.42	59.58	13.28
	3	1.85	1.76	7.92	5.66	21.03	18.80
	MV	**2.20**	**1.73**	**15.14**	**5.42**	**33.90**	**18.56**

Table 6. fr_{OFF*} and for client 1 in scenario 1, 2 and 3

Metric	Scenario	Cubic		Illinois	
		HTBM	RWTM	HTBM	RWTM
CNG	1	191.72	1.11	58.68	0.76
	2	375.62	0.82	33.11	0.68
	3	173.48	0.66	56.27	0.76
	MV	**246.92**	**0.86**	**49.35**	**0.73**
fr_{OFF*}	1	0.16	0.24	0.06	0.24
	2	0.23	0.24	0.21	0.24
	3	0.13	0.26	0.05	0.26
	MV	**0.17**	**0.25**	**0.10**	**0.25**

Both tables indicate two valuable points:

On one hand, RWTM has better QoE metric measurements than HTBM with both Cubic and Illinois variants. Moreover, RWTM not only has a lower congestion rate, *CNG*, than HTBM, but it also has a negligible *CNG* with the two TCP variants for all three scenarios. RWTM also preserves a constant value of fr_{OFF*}. Consequently, even the transition from one to two clients and vice versa (i.e., scenarios 2 and 3, respectively) does not disturb RWTM, which preserves its inherit characteristics of negligible congestion rate and its fr_{OFF*} rate around 0.25. This preservation has positive consequences for the user's QoE. Although the gap between the QoE metrics measurements of {Cubic RWTM} and {Illinois RWTM} is not very significant, {Cubic RWTM} yields better values. Accordingly, we can say that the use of Cubic or even Illinois is beneficial for improving the user's experience, with a slight preference for Illinois.

On the other hand, HTBM presents better QoE with Illinois than with the Cubic variant. In conjunction, it has a fivefold lower congestion rate (49.35 vs. 246.92) and lower *OFF** period frequency fr_{OFF*}. This observation is valid not only for total mean values, MV, but also with every scenario (1, 2, and 3). Therefore, Illinois is distinctly better than Cubic for the HTBM shaping method, even when the number of active HAS clients in the home gateway changes between one and two clients. Accordingly, the loss-delay-based variant with the C-AIMD approach used by Illinois has more favorable impacts on QoE, *CNG*, and fr_{OFF*} than the loss-based variant with the AIAD approach using the HyStart algorithm employed by Cubic.

4.3 Scenario 4

The objective of this section is to evaluate the robustness of each combination against the congestions that are induced by other flows. Therefore, we employed scenario 4, as described in Subsect. 3.2, in which a heavy congestion is induced. To be able to compare performances correctly, a reference scenario, denoted by WL, consisting of a HAS client working alone in the home network, is used. No loss is observed in the reference scenario. We do not present the NewReno and Vegas variants because they showed poor performance. Altogether, we have four combinations to evaluate: Cubic and Illinois combined with two shaping methods, HTBM and RWTM. The average values of the QoE metrics of the client in scenario 4 are provided in Table 7, and the average values of *CNG* and fr_{OFF*} are listed in Table 8.

Table 7. QoE for client 1 in scenario 4

TCP variant	Scenario	Performance metric					
		Instability (%)		Infidelity (%)		Convergence speed (s)	
		HTBM	RWTM	HTBM	RWTM	HTBM	RWTM
Cubic	WL[1]	1.08	1.07	3.71	1.79	7.61	4.10
	4	4.86	6.40	48.2	46.14	120.3	129.3
Illinois	WL	1.08	1.07	2.23	1.66	5.37	4.01
	4	2.7	2.92	15.6	17.81	35.48	42.75

[1]Without loss [2]Degradation percentage

Table 8. fr_{OFF*} and *CNG* for client 1 in scenarios 1, 2, and 3

Metric	Scenario	Cubic		Illinois	
		HTBM	RWTM	HTBM	RWTM
CNG	WL	34.2	0.98	38.51	0.79
	4	216.19	120.36	146.68	143.54
fr_{OFF*}	WL	0.03	0.36	0.03	0.43
	4	0.40	0.41	0.09	0.31

The measurements in the two tables indicate three major observations:

- The lowest QoE metric measurements are recorded for the Cubic variant for both shaping methods: their instability is around 5~6%, their infidelity is near 50%, and their convergence speed is approximately 125 ms. We also notice that the congestion rate, *CNG*, is very high, between 120 and 220, and the frequency fr_{OFF*} is important around 0.4 (i.e., one *OFF** period occurs for every 2.5 chunks, on average). We observe not only lower measurements, but also a higher degradation rate in performance: the gap of QoE metric measurements, *CNG* and fr_{OFF*}, is clearly large between scenario WL and scenario 4. Accordingly, Cubic is not suitable as a TCP congestion control variant of the HAS server for both shaping methods when heavy congestion occurs. This result can be verified by examining Figures 8 and 9 in Subsect. 4.1.2, where Cubic has difficulties with rapidly defining the suitable *ssthresh* value before convergence and after congestion, respectively.

- The RWTM shaping method presents higher degradation in QoE metric measurements than HTBM when we compare scenario WL to scenario 4 for both TCP congestion variants, Cubic and Illinois. The cause is mainly related to the fact that HTBM is used to generate congestion events and maintain high QoE under normal circumstances, which is not the case with RWTM. Accordingly, we can say that RWTM is more sensitive to induced congestions than HTBM. This result can be verified when examining Figures 7 and 9 in Subsect. 4.1.2, in which a single congestion event instantaneously degrades performance.

From the first and the second observations, we can deduce that the best combination that maintains optimal QoE metric measurements with low degradation and has the lowest frequency of *OFF** periods, fr_{OFF*}, is ensured by the combination {Illinois HTBM}.

- The second-best combination is {Illinois RWTM}. Here, the QoE metric measurements are acceptable, but the degradation rate is higher than the best combination {Illinois HTBM}. This degradation indicates that {Illinois RWTM} cannot adequately resist against induced congestions, especially when we have a highly congested link between client and server. However, this combination could successfully be used with a link under less frequent congestions.

4.4 Scenario 5

In this section, we present the performance metric measurements when the standard variation of RTT_{C-S} varies. The behavior of the resulting curves will indicate how performance degrades when the standard deviation of RTT_{C-S} increases. The variation of QoE (instability, infidelity, and convergence speed) is presented in Figure 10, and the variation of *CNG* and fr_{OFF*} is listed in Figure 11. We have two major observations:
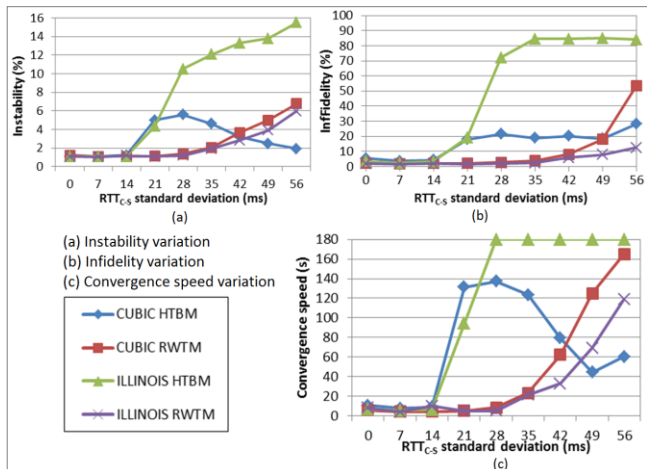

(a) Instability variation
(b) Infidelity variation
(c) Convergence speed variation

- CUBIC HTBM
- CUBIC RWTM
- ILLINOIS HTBM
- ILLINOIS RWTM

Figure 10. QoE variation

(a) *CNG* variation
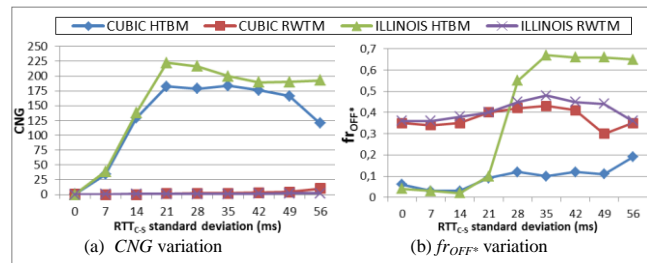(b) fr_{OFF*} variation

Figure 11. Variation of *CNG* and fr_{OFF*}

On the one hand, the {Illinois RWTM} (purple cross) and the {Cubic RWTM} (red square) curves are convex and close to each other and are predominantly below the other curves. The three QoE metric values are good until an RTT_{C-S} standard deviation of around 40 ms, where

performance degradation begins to be visible. Moreover, the combination {Illinois RWTM} preserves its performance better and has a less aggressive degradation rate for higher RTT_{C-S} standard deviation, especially from 35 ms. Accordingly, we can say that using RWTM with Illinois is safer when RTT_{C-S} is very unstable. Otherwise, {RWTM Cubic} can also be used, most usefully when the standard deviation of RTT_{C-S} is lower than 35 ms. We also observe in Figure 11 that both combinations have a similarly low congestion rate, *CNG,* and similar frequency of *OFF** periods, fr_{OFF*}. Based on this observation, we can deduce that RWTM preserves its inherent characteristics with Cubic and Illinois variants, and that the degradation of QoE metric measurements for highly unstable RTT_{C-S} is mainly caused by the congestion control algorithms used by Cubic and Illinois variants. Since RWTM seems to be more adequate with Illinois, we can say that the loss-delay-based and C-AIMD approach of Illinois helps more than the loss-based and cubic RTT-independent approach of Cubic to preserve good performance for highly unstable RTT_{C-S} values.

On the other hand, HTBM is less robust against RTT_{C-S} instability. The green and the blue curves that present the combination of HTBM with Illinois and Cubic, respectively, show a significant degradation of QoE metric measurements when the standard deviation of RTT_{C-S} is above 14 ms. However, {HTBM Illinois} is more sensitive to RTT_{C-S} instability than {HTBM Cubic}. This means that combining the loss-delay-based congestion control variant Illinois with the HTBM shaping method that increases the queuing delay, entails harmful drawbacks for QoE when the RTT_{C-S} is unstable. We can also validate this observation in Figure 11: the congestion rate *CNG* of {HTBM Illinois} and {HTBM Cubic} are predominantly close to each other, but the frequency of *OFF** periods explodes with {HTBM Illinois} for RTT_{C-S} standard deviation higher than 20 ms. This implies that the additional delay caused by HTBM is practically the same for both congestion control variants Cubic and Illinois, but the effects on fr_{OFF*} and QoE are quite different and involve more drawbacks for the Illinois variant. In contrast, HTBM with Cubic has fewer drawbacks and presents QoE metrics measurements that are relatively constant for instability and infidelity from RTT_{C-S} standard deviations around 20 ms. This results can be explained by the fact that the Cubic variant does not use RTT_{C-S} to compute its congestion window *cwnd* during the congestion avoidance phase, as explained in Subsect. 2.1.

4.5 Discussion

After comparing the results of five scenarios, we have made numerous observations, but in this subsection, we want to summarize the most important observations. First, NewReno and Vegas variants do not provide good performance in the HAS context, excepting the combination {NewReno RWTM} that could perform well if the initial

value of *ssthresh* is well-chosen. Second, we summarize the observations of the five scenarios for the four combinations in Table 9. Thus, we assign a score for each combination that ranges between "--" and "++": -- (bad), - (insufficient), +/- (average), + (good), and ++ (excellent). This score is based on the analysis of results for each scenario.

Table 9. The final score for each combination

Scenario	Combination			
	RWTM		HTBM	
	Cubic	Illinois	Cubic	Illinois
{1,2, 3}	++	++	+/-	+
4	--	+	--	++
5	+	++	+/-	--

The best combination is {Illinois RWTM}: it yields good performance when two clients compete for bandwidth and is robust against high RTT_{C-S} variation, but it is somewhat vulnerable to heavy congestions that could be caused by external factors. In the second position, we have two combinations:

- {Cubic RWTM}: Unfortunately, it is very vulnerable to congestions and slightly sensitive to high RTT_{C-S} variation.

- {Illinois HTBM}: It has the advantage of being robust against heavy congestions. However, it is very sensitive to RTT_{C-S} variation. Furthermore, it causes a high rate of congestion in the gateway that could disturb other sessions in concurrence with HAS sessions.

5. CONCLUSION AND FUTURE WORK

A comparative evaluation has been developed in order to study the effect of combining two well-known traffic shaping methods (HTBM and RWTM) in the gateway with four very common TCP congestion algorithms (NewReno, Vegas, Illinois, and Cubic) in the server in the context of HTTP adaptive streaming technique. We examined the user's QoE by applying objective metrics. Furthermore, we observed the evolution of the congestion window on the server side in order to explain the behavior of each combination and its relationship with QoE metrics. We also used the congestion rate and the frequency of *OFF* periods that exceeds retransmission timeout as indicators. We have addressed many scenarios: two HAS clients competing for the home bandwidth simultaneously, adding or removing a HAS client, inducing a heavy congestion in the gateway, and increasing the instability of the round trip time, *RTT*, between the HAS server and the HAS client. The results show that there is a significant discordance in performance between combinations. The best combination that improves the QoE, reduces the congestion rate, and reduces the OFF periods in the majority of scenarios is when combining the loss-delay-based congestion control variant, Illinois, which uses the C-AIMD approach, with the TCP flow control-based method, RWTM. The characteristics of Illinois and RWTM seem to be similarly robust against high *RTT* instability. This combination does not disturb other real-time streams in the home network because it does not induce additional queueing delay and it considerably reduces the congestion rate. However, this combination is slightly vulnerable to heavy congestions that could be caused by external factors such as other concurrent streams.

Having extended our knowledge about the combination of TCP congestion control variants with shaping methods in this work, we intend as future work to design a new TCP congestion control variant that is compatible with all specifications of HAS and shaping methods.

6. REFERENCES

[1] Houdaille Rémi, and Stéphane Gouache. "Shaping HTTP adaptive streams for a better user experience." In Proceedings of the 3rd Multimedia Systems Conference. p. 1-9. ACM, 2012.

[2] Ben Ameur Chiheb, Emmanuel Mory, and Bernard Cousin. "Shaping HTTP adaptive streams using receive window tuning method in home gateway." In Proceedings of the 33rd Performance Computing and Communications Conference. p. 1-2. IEEE, 2014.

[3] Floyd Sally, Thomas Henderson, et al. "The NewReno modification to TCP\ x27s fast recovery algorithm." RFC 3782, 2012.

[4] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. "TCP Vegas: New techniques for congestion detection and avoidance", Vol. 24, no. 4. ACM, 1994.

[5] Liu Shao, Tamer Başar, and Ravi Srikant. "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks", Performance Evaluation 65, no. 6 (2008): 417-440.

[6] Ha Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." ACM SIGOPS Operating Systems Review 42.5 (2008): 64-74.

[7] Jiang Junchen, Vyas Sekar, and Hui Zhang. "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive." In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies. p. 97-108. ACM, 2012.

[8] Ha Sangtae and Injong Rhee. "Taming the elephants: New TCP slow start." Computer Networks 55.9 (2011): 2092-2110.

[9] Akhshabi Saamer, Narayanaswamy Sethumadhavan, et al. "What happens when HTTP adaptive streaming players compete for bandwidth?." In Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video. p. 9-14. ACM, 2012.

[10] Misra Vishal, Wei-Bo Gong, and Don Towsley. "Fluid-based analysis of a network of AQM routers

supporting TCP flows with an application to RED." In Proceedings of ACM SIGCOMM Computer Communication Review. Vol. 30. No. 4. ACM, 2000.

[11] Villa Bjørn J. and Poul E. Heegaard. "Group based traffic shaping for adaptive HTTP video streaming by segment duration control." In Proceedings of the 27th International Conference on Advanced Information Networking and Applications. p. 830-837. IEEE, 2013.

[12] Stewart Lawrence, Hayes David, et al. "Multimedia-unfriendly TCP congestion control and home gateway queue management." In Proceedings of the Second Annual ACM Conference on Multimedia Systems. p. 35-44. ACM, 2011.

[13] Hayes David and Grenville Armitage. "Improved coexistence and loss tolerance for delay based TCP congestion control." In Proceedings of the 35th Conference on Local Computer Networks. p. 24–31. IEEE, 2010.

[14] Ho Cheng-Yuan, et al. "Fast retransmit and fast recovery schemes of transport protocols: A survey and taxonomy." Computer Networks 52.6 (2008): 1308-1327.

[15] Jamal Habibullah, and Kiran Sultan. "Performance analysis of TCP congestion control algorithms." International Journal of Computers and Communications 2.1 (2008): 18-24.

[16] Brown Martin A. "Traffic Control HOWTO." Guide to IP Layer Network (2006).

[17] Allam Mark,Vern Paxson, and William Stevens. "RFC 2581: TCP congestion control." (1999).

[18] Yin Xiaoqi, Vyas Sekar, and Bruno 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, p. 1-9. ACM, 2014.

[19] Akhshabi Saamer, Narayanaswamy Sethumadhavan, et al. "An experimental evaluation of rate-adaptive video players over HTTP." Signal Processing: Image Communication 27.4 (2012): 271-287.

[20] Huang Te-Yuan, Handigol Nikhil, et al. "Confused, timid, and unstable: picking a video streaming rate is hard." In Proceedings of the Internet Measurement Conference. p. 225-238. ACM, 2012.

[21] Chiheb Ben Ameur, Emmanuel Mory and Bernard Cousin, "Evaluation of gateway-based shaping methods for HTTP adaptive streaming", In Proceedings of Quality of Experience-based Management for Future Internet Applications and Services Workshop. p. 1777-1782. IEEE International Conference on Communications, 2015.

[22] Cranley Nicola, Philip Perry, and Liam Murphy. "User perception of adapting video quality". International Journal of Human-Computer Studies, p.637-647, 2006.

[23] Mok Ricky KP, Chan Edmond, et al. "Inferring the QoE of HTTP video streaming from user-viewing activities." In Proceedings of the First ACM SIGCOMM Workshop on Measurements Up the Stack. p. 31-36. ACM, 2011.

[24] Eric Dumazet "Stochastic fairness queuing discipline", http://manpages.ubuntu.com/manpages/trusty/man8/ tc-sfq.8.html. Ubuntu Manuals, 2010.

[25] Akhshabi Saamer, Narayanaswamy Sethumadhavan, et al. "Server-based traffic shaping for stabilizing oscillating adaptive streaming players." In Proceedings of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video. p. 19-24. ACM, 2013.

[26] Jean Yves Le Boudec, "Rate Adaptation, Congestion Control and Fairness: A Tutorial." p. 1-44. Ecole Polytechnique Federale de Lausanne (EPFL), 2014.