Avoiding zigzag quality switching in real content adaptive video streaming

Wassim Ramadan, Eugen Dedu, and Julien Bourgeois

University of Franche-Comté,
Laboratoire d'Informatique de l'Université de Franche-Comté
Montbéliard, France

Abstract. A high number of videos, encoded in several bitrates, are nowadays available on Internet. A high bitrate needs a high and stable bandwidth, so a lower bitrate encoding is usually chosen and transferred, which leads to lower quality too. A solution is to adapt dynamically the current bitrate so that it always matches the network bandwidth, like in a classical congestion control context. When the bitrate is at the upper limit of the bandwidth, the adaptation switches constantly between a lower and a higher bitrate, causing an unpleasant zigzag in quality on the user machine. This paper presents a solution to avoid the zigzag. It uses an EWMA (Exponential Weighted Moving Average) value for each bitrate, which reflects its history. The evaluation of the algorithm shows that loss rate is much smaller, bitrate is more stable, and so received video quality is better.

Key words: Real time content, Video streaming, Rate control, Congestion control

1 Introduction

Nowadays, the number of videos encoded in several bitrates and accessible for everyone increases significantly day by day. Their contents are generally delivered to final user using streaming services over Internet. These services as well as the demand for high video quality (e.g. HD and 3D videos) are in constant progression. They require more and more bandwidth, hence available bandwidth variation must be taken into account to shorten buffering time at the receiver.

Currently, one video bitrate is chosen at the beginning of a video streaming; the transmission is controlled at the network layer (TCP or UDP) and application is not involved at all. Hence, two choices exist when playing streamed video content. The first is to choose a low video bitrate, and the video is played directly, without interruption. The second is to choose a bitrate higher than average bandwidth, buffer multimedia data at the user and play the video when the buffer has sufficient data; because the buffer empties, the user will be confronted to many play/pause during the streaming. Both cases are unpleasant to the user eye: the first has the advantage to watch a fluid video but with low quality, while the latter allows to have a good quality but with either short or long waiting time and with frequent interruptions. Users wishing to have an instant access without delay to multimedia content and with the best possible quality can use a new kind of multimedia streaming services, *multimedia content adaptation*.

For video streaming in a network with highly variable bandwidth, a *content adaptation* is a way to adapt the video bitrate to the network characteristics, hence to improve the video quality perceptible by the final user. A cooperative approach between application layer and network layer can be used. Transport protocol handles the congestion control on the network side, while the application handles the video bitrate control on the server side. Bitrate control can be done by changing the quantisation parameters, or by changing the FPS (frames per second) etc.

An undesirable effect which appears when multiple video qualities are available and when bandwidth changes, is that the bitrate control leads to constant switching between two qualities (bitrates); one is smaller than the bandwidth and the second is greater. For example, when a user is connected to Internet through a 2.5Mb/s link and the video is available in 2 and 3Mb/s qualities, then an adaptive streaming algorithm will constantly switch between these two qualities; obviously, the best solution would be to stay with 2Mb/s much more time before retrying 3Mb/s quality. We call this problem *zigzag quality switching*.

Few papers treat this issue, and they do not solve it completely. This paper presents a solution, called ZAAL (**Z**igzag **A**voidance **Al**gorithm), to this problem. It uses a successfulness value of each bitrate, and its average is constantly updated. This average is used to take the decision whether a next higher quality can be chosen or not, thus preventing the zigzag.

This paper is organized as follows. Section 2 formulates the problem which we try to solve, and section 3 compares it to similar existing problems. Section 4 presents our ZAAL algorithm as a solution to the problem, and its performance is evaluated through real experiments in section 5. Finally, section 6 concludes this article and presents some perspectives.

2 Problem formulation

The zigzag quality switching is the fact that an adaptive video streaming sender application keeps switching the video quality between two values, especially when one value is greater and the other is smaller than the bandwidth capacity.

In a network with highly variable bandwidth, video adaptation is very important. It aims to adapt the video bitrate to the network characteristics and thus to improve the video quality perceptible by the final user. The adaptation can be done at several OSI layers, but we take the application layer to exemplify. To adapt video to network conditions, the sender application controls some video parameters, such as the bitrate, FPS (Frames per second) or image size; for example, if more bandwidth is available, application increases the video bitrate, and if less bandwidth is available, application decreases it. To reach this goal, application should constantly retrieve the bandwidth (each 100ms, each 2 sec. or each I image for MPEG video for example) and adapt its bitrate accordingly. To minimise the quality fluctuation, it is not preferable, in our opinion, to change quality each tiny period (at each sent packet for example). Sometimes the application has the choice among multiple available video qualities (e.g. many videos on Internet are encoded with several qualities). This video adaptation is known in the literature as "rate adaptive video control".

Fig. 1 allows to better understand how video adaptation works. The transport protocol has a congestion control which gives the rate at which the packets leave the socket buffer (and afterwards the machine, and enter the network). When current bandwidth is smaller than video bitrate, the socket buffer fills with packets. When the buffer becomes full, exceeded packets generated by application will fail to be written. We call these packets "failed packets". The goal of an adaptive video content application is to readjust video bitrate in this case by choosing a lower video quality. When new bitrate does not cause failed packets, application will retry a higher quality to match again the available bandwidth and to enhance the video quality.

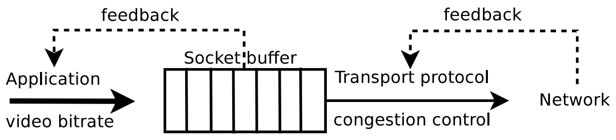


Fig. 1. Video data flow on the sender side.

Our previous paper VAAL (Video Adaptation at Application Layer) [12] is an example of video adaptation method. It uses transport protocol buffer overflow as a solution to find out the available bandwidth and to adapt video content bitrate to the discovered bandwidth. Each 2 seconds, the server application computes the number of failed packets. This number is used to control the video bitrate afterwards. A high number means smaller bandwidth and smaller bitrate. Zero error indicates either a stable or more bandwidth, so bitrate of sent video can be increased. The experiments presented in this paper use VAAL.

An adaptive application, such as the one presented above, keeps switching between two qualities: one with bitrate higher than bandwidth and with failed packets, and another one with bitrate smaller than bandwidth and with no error. We noticed this phenomenon in our experiments (see Fig. 4(a) for a clear example, where during the first minute the video bitrate is continuously toggling between 0.5 and 1 Mb/s). This is what we called "zigzag quality switching" problem; it has two undesirable effects:

– Numerous bitrate changes, leading to unstable video quality. It is preferable to keep the video quality stable as much as possible while maximising it.
– Many lost packets (when bitrate is higher than bandwidth), leading to lower video quality and wasted ressources (CPU and network) for these lost packets. Indeed, videos are generally more affected by packet losses than lower bitrate.

Note that *the problem to be solved is not to reduce the number of quality changes*, for example when such changes use much resources (CPU, disk etc.) *It is neither to use the highest available quality at a certain moment* (which can potentially lead to zigzag, for example when right afterwards the bandwidth decreases). The problem to be solved is the *zigzag*, i.e. avoiding to reuse a quality which has recently been used (one or more times) and has proved unsuccessful. As such, a solution which does not take into account the recent history somehow cannot solve this problem.

3 Positioning comparing to related work

We found that our problem has similarities with other problems, and with two techniques already found in the literature.

3.1 Similar methods

One method which can be used to solve zigzag quality switching is presented in [10]. It is proposed for multicast streaming video but can be adapted to unicast transmissions too. The sender sends several layers (base and enhancement), and the receiver subscribes dynamically to one or several of them. The receiver uses a timer for each level of subscription (layer). At the beginning the timer is short. When the timer expires and no loss was experienced, the receiver subscribes to the next new level and the timer for that level is started. If on the contrary the level led to lost packets, the receiver goes back to previous level and the timer for the level with losses is multiplicatively increased.

In this method, there is no superior limit for the timer, which means that the quality increasing is forbidden for a very long time, even if bandwidth becomes greater in the meantime. Also, each time, only the timer of the current bitrate is updated, which means that good conditions for higher levels do not ameliorate the timer of lower levels. Both these characteristics are unrealistic. On the contrary, our EWMA-based algorithm does not have these drawbacks.

A metric to evaluate the jerkiness of a video, given by the number of quality changes, is given in [4]. It uses a formula to calculate the Effective Frame Rate (EFR) of a video:

$$EFR = \sum_{i=1}^{N-1} \left(\frac{fps_i - P * qualitychange(max(i-W,0),i)}{N} \right) \tag{1}$$

where fps_i is the number of frames delivered in the ith second of the video, W is the window size, P is a weighting factor for variation in frame rate, and $qualitychange(max(i-W,0),i)$ is the number of quality changes in the range $i-W$ to i.

This formula is used to limit the number of quality changes during each window (W). As such, it counts the number of quality changes, no matter *where* they appear inside the window, which is however an important parameter of the jerkiness. For example, 10 quality changes in the first 2 seconds of a video of 1 minute are worse visually than 10 quality changes dispersed through the whole video. The goal of EWMA, used in our solution, is exactly to take into account the time of the change too. Moreover, in [4] the window is static (coarse-grained moving) and fixed (same size), while in our case we use a sliding and dynamic window.

Finally, our goal is not to limit the number of quality changes, but to avoid quality increases which lead to quality decreases right afterwards. For all these remarks, the formula in this paper is not interesting in avoiding the zigzag problem we want to tackle.

The adaptation video method described in [3] does not cope directly with zigzag problem but presents a way for smoothing sent video bitrate and reducing the frequency of video quality changes. The main idea is that the application does not switch to a higher video quality (or video layer in a hierarchical video encoding) until the sender is certain that the video will continue playing even after a reduction of the congestion window.

This paper presents results for AIMD and SQRT congestion controls for video streaming. To guarantee the continuous video playing, the sent video quality should be the highest quality which verify: $C < R - \beta R^l$, where C is the bitrate at which the video is encoded, R is the transmission rate and $\beta = 1/2$ and $l = 1$ for AIMD based congestion control and $l = 1/2$ for SQRT based one. This formula implies that the highest video quality does not exceed half of the transmission rate for AIMD, and it is $\beta\sqrt{R}$ smaller than the bandwidth for SQRT.

Results given in [3] show that this formula works well for AIMD based congestion control streaming application but the video bitrate is most of the time very low compared to available bandwidth. On the other hand, it has a little impact on SQRT based algorithm, i.e. the quality changes are not significantly reduced. Finally and most importantly, this formula does not really solve the zigzag quality switching, but merely reduces the bitrate artificially.

3.2 Comparison with bandwidth estimation techniques

These techniques aim to estimate the available bandwidth at the time of the measurement.

A well-known technique is packet pair [7]. The basic idea is that the sender sends a pair of packets of the same size to a destination (it could be a specialised server or simply an aware receiver). The destination host then responds by sending an echo for each received packet. By measuring the changes in the time spacing between the two packets, the sender can estimate the available bandwidth of the network path as following:

$$bw = \frac{s}{t_2 - t_1} \tag{2}$$

where:

- bw is the bandwidth of the bottleneck link
- s is the packet size
- t_1 and t_2 are the arrival time of the first and second packet respectively.

The accuracy of this technique depends on several assumptions. The most important one is that the two packets should be enqueued *one after the other* at the bottleneck link, which is not guaranteed when a router has a non FIFO queue or when another packet is inserted between the two packets. Other variants of packet pair technique are developed [1, 9, 2] to mitigate this effect.

Another known technique is packet train [8]. A burst of packets is sent between source and destination. Only packets in the same train are used for measurement. A train is formed by packets for which the spacing between two consecutive received packets does not exceed some inter-packet (inter-train) gap. Like in packet pair, inter-arrival times between packets in the same train are used for estimating the available bandwidth.

All these methods have several drawbacks:

1. Sending/receiving additional data packets is needed to estimate the available bandwidth. If video data packets themselves are used for this, receiver must distinguish them from other data, for example by adding a new field in the packet header.
2. Probing packets should be specifically dispersed, i.e. they need a specific timing when they are sent. This makes the use of the video data packets themselves difficult for probing purposes.
3. If a new connection is used for probing packets, sender/receiver should send/listen to a different socket (port).
4. Not only the sender application but also the receiver application must be modified to be able to respond to those probing packets. Changing both endpoints is known to be very difficult to deploy in reality.

Finally, and the most important, these techniques have a different goal than ours. As they do not take into account the bitrates used in the past, they cannot avoid the zigzag.

The previous cited reasons show that current bandwidth estimation techniques are not appropriate to solve the zigzag quality switching problem.

3.3 Comparison with network congestion control

Our problem is similar to classical network congestion control problem, but not identical. We consider both window-based (written as TCP [11] in the following) and equation-based (written as TFRC [5] in the following) congestion controls.

In fact, both TCP/TFRC and ZAAL try to solve a multi-criterion optimisation problem, but the criteria involved to achieve this goal are not really identical.

Zigzag: ZAAL aims to avoid the zigzag between two consecutive qualities, which occurs when (1) the inferior quality is lower than bandwidth, hence no loss, hence the quality is increased, *and* (2) the superior quality is higher than bandwidth, hence a few losses, hence the quality is decreased to the inferior quality again. TCP/TFRC have fine-grained sending rate; TCP, being a *data* transport protocols, does not pay attention to such zigzag, while TFRC smooths the sending rate, without avoiding the zigzag.

Losses: TCP/TFRC aim to send at the maximum rate, even if this yields a few losses. On the contrary, ZAAL aims to avoid losses as much as it can. In reality, losses cannot be completely avoided unless a very low bitrate is used. So for ZAAL it is better to maintain a quality without or with few losses, instead of the next higher quality with high loss rate. In fact, ITU.T G.1070 [6] recommends that the end-to-end IP packet loss rate in video streaming should be less than 10%. As an example, during a video transmission ZAAL prefers a 2Mb/s sending rate without losses to a 3Mb/s sending rate (50% more packets) with 20% lost packets.

Sending rate: TCP/TFRC aim to send at the highest possible rate. This is why they *constantly increase* sending rate (using various laws, which differ from one TCP variant to another). VAAL aims that too, but does not increase quality if ZAAL considers, for zigzag/loss avoidance as shown above, that the next higher quality should not be tried at this time. Thus VAAL can be seen as quality increasing/decreasing proposition and ZAAL as quality increasing-only blocker.

This comparison shows that classical congestion control algorithms have different goals, hence they are not an appropriate to solve the zigzag problem.

4 Zigzag-avoiding algorithm overview

To solve the zigzag quality switching generated by a video adaptation method, an additional specific algorithm can be used, such as ZAAL. ZAAL algorithm works by avoiding constantly using bitrates higher than the available bandwidth. For that, it maintains an average value for each bitrate, called *successfulness* in the following. When the adaptive algorithm considers to increase bitrate (and only in this case), ZAAL checks if the successfulness of the higher bitrate is lower than a threshold, called β; if this is the case, a higher bitrate *cannot* be chosen. Otherwise said, application uses a higher bitrate i only if its successfulness $S_i > \beta$. After this process, the average successfulness is updated.

Note that ZAAL is not an adaptation method. It is used only to prevent an adaptation method from frequently switching the video quality. The only information ZAAL needs comes from the adaptation method, whether some bitrate causes lost packets or not.

4.1 Algorithm

ZAAL algorithm uses the *successfulness* value each time an adaptation period ends, which depends on the adaptation algorithm (e.g. 2 sec. in case of VAAL). Average successfulness value is calculated separately for each bitrate (with different weights), denoted by S_i, which indicates if bitrate of index i can be used for the next period or not. As such, this average value expresses the application last attempts to use the corresponding bitrate. In brief, when a bitrate generates failed packets to the transport protocol buffer, corresponding successfulness value is greatly reduced; when a bitrate was successful, the successfulness value is greatly increased; finally, when a bitrate is used for a long time, the successfulness values corresponding to higher bitrates are slowly increased. As a general rule, the smaller the successfulness average value, the more the corresponding bitrate causes failed packets and application must avoid using it.

The average successfulness S_i (where i is bitrate index) of each bitrate, which changes according to the history of that bitrate, is calculated using an EWMA algorithm (Exponential Weighted Moving Average). Using EWMA allows to give greater weight to recent history compared to older history, since obviously current bandwidth is better expressed by recent bitrate usage than by older ones. Additionally, different weights are used, based on the bitrate involved.

At the beginning of a video transmission, all S values are set to 1. Then they are calculated each time the application wants to adapt the video bitrate to the available bandwidth, using the following general formula:

$$S_i = (1 - \alpha/d)S_i + s(\alpha/d) \tag{3}$$

where:

- s is the successfulness at the time of measurement (the current "observation") and can be either 0 or 1. 0 value is used when the bitrate did not give good results (hence its successfulness average value will decrease). 1 value is used when the bitrate gave good results, either because the bitrate did not cause failed written packets, or because the bitrate was not used recently (hence its successfulness average value will increase).
- α is the degree of weighting increase/decrease, a constant smoothing factor between 0 and 1. A higher α discounts older successfulness values faster.
- d is a division factor allowing to speed up or slow down the average value increasing depending on the bitrate involved. In our algorithm, d has three values: 1, 2 and 4. For $s = 1$, the greater the d, the slower the increasing of the average S_i value. They will be explained more later.

S values are arithmetic reals between 0 and 1. They can change in three cases:

1. First, when the application increases the video bitrate, i.e. the new bitrate k is higher than the actual one j. This appears when the application does not sense lost packets for a while with actual video quality. In this case S should increase ($s = 1$) for all bitrates i lower than or equal to the current bitrate. Increasing speed must be high ($d = 1$). So:

$$S_{i|i \leq j} = (1 - \alpha)S_i + \alpha \tag{4}$$

2. Second, when the application maintains the bitrate. This happens either when some packet losses occur but their rate is acceptable to maintain the

current quality j, or successfulness of the higher bitrate k (S_k) is low and application avoids using it. S value increases for all qualities but with different division factor values (different speeds). So:
   - $d = 1$ and $s = 1$ (big increase) for all bitrates i lower than the current one j:
$$S_{i|i<j} = (1-\alpha)S_i + \alpha \tag{5}$$
   - $d = 2$ and $s = 1$ (small increase) for current bitrate j:
$$S_j = (1-\alpha/2)S_j + \alpha/2 \tag{6}$$
   - $d = 4$ and $s = 1$ (very small increase) for the bitrate $k = j+1$ higher than the current one j:
$$S_{k(k=j+1)} = (1-\alpha/4)S_k + \alpha/4 \tag{7}$$
3. Third, application decreases the video bitrate. This appears when the available bandwidth is not enough anymore for the actual bitrate (i.e. bandwidth decreases during the current period), or when the application tries a bitrate higher than the bandwidth. In this case, average value must be reduced for the current bitrate j, other bitrates average values do not change. Hence, $s = 0$ and $d = 1$ (big decrease). So:
$$S_j = (1-\alpha)S_j \tag{8}$$

In our experiments we intuitively set $\alpha = 0.3$ and $\beta = 1 - \alpha = 0.7$. Other values were analysed but either they lead to high number of adaptation iterations for the algorithm to converge, or they minimize its effects. Determining the best values of α and β and how they affect the stability and the adaptivity of ZAAL is still future work.

4.2 Discussion

We present in this section some characteristics of zigzag-avoiding algorithm. To simplify results, we use $\alpha = 0.3$ and $\beta = 0.7$. We can notice the following:

1. At the beginning, average values are set to 1. They are greater than $\beta = 0.7$, so there is no restriction using any bitrate, i.e. application is authorised to use all bitrates.
2. Studying the decreasing phase of the algorithm we can notice:
   - As mentioned previously, when a bitrate is greater than the available bandwidth, application avoids to use it for a while by decreasing its successfulness average value. Based on equation 8, the first time this occurs, $S_i = 1 - 0.3 * 1 = 0.7$. Hence, a bitrate which causes failed writing packets cannot be used two consecutive times.
   - The smallest S value for a bitrate i can appear when it is used with $S_i = 0.7 + \epsilon$ and it leads to failed packets. So, its new S value is equal to $S_i = 0.7 * (0.7 + \epsilon) = 0.49 + \epsilon$.

   - A higher bitrate does not necessarily mean a smaller S value. When reducing an S value for a bitrate i, higher bitrates do not change.
3. Studying the increasing phase of the algorithm we can notice:
   - The maximum number of times ZAAL prevents an application to increase bitrate occurs when the higher bitrate k has the minimal S value of $S_k = 0.49 + \epsilon$ (see previous point) and ends when S_k becomes > 0.7. Using equation 7, this corresponds to 7 unsuccessful attempts by the application for the higher bitrate k, followed by an 8th successful attempt. Fig. 2 shows the evolution of this value: there are 7 unsuccessful attempts between $0.49 + \epsilon$ and a value greater than 0.7.
   - S values between 0.7 and 1 are only possible when the bitrate does not cause failed packets in two cases: it increases slowly beyond 0.7 when the quality is stable (see equation 6), or it increases quickly beyond 0.7 when higher qualities are possible (see equation 5).
4. Finally, using an exponential average rather than a linear average allows to remember past values for a longer time when increasing average values S, and to give more weight to recent history than to old history.

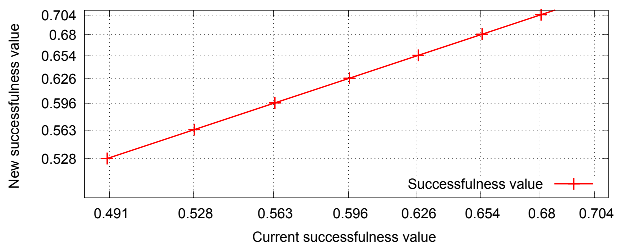


Fig. 2. Successfulness average increasing during the biggest period of time.

4.3 ZAAL complexity

- ZAAL is a simple and easy to implement algorithm. Indeed, ZAAL consists of 3 `if` clauses with a loop over all bitrates inside each clause.
- ZAAL is scalable, since it has a linear complexity in number of bitrates (a loop on all bitrates), and also in number of clients (ZAAL is executed independently for each of them). Moreover, it has a negligible execution time. Indeed, it needs to calculate just one value per available bitrate during each adaptation period (e.g. each 2 sec.), using a simple formula, as shown above. Hence, using ZAAL does not really affect the server processing capacity when the number of clients increases. The small additional impact of ZAAL can be easily compensated by its other positive side (i.e. when using ZAAL, the server avoids processing packets which would otherwise be lost on the network).

5 Experimental results

Fig. 3 shows the real network used to realise experiments. A video streaming connection is made between a sender and a receiver with wired interface for both. An intermediate machine (called shaping machine in the following) is added between the sender and the receiver. This shaping machine has two interface cards: a wired interface connected to the sender and a wireless one (54Mb/s) connected to an access point (AP). The receiver is connected to the same AP via a wired interface. The video streaming uses a real video, available in four qualities: 3Mb/s, 2Mb/s, 1Mb/s and 512kb/s. The video has 180s.

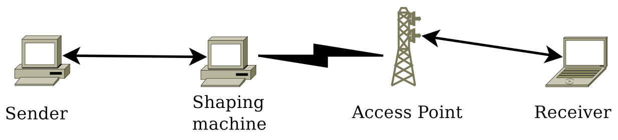


Fig. 3. Network topology used for experiments.

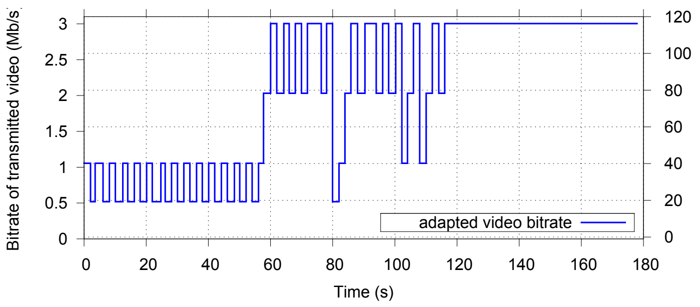
Two series of tests are done. For the first series (traffic shaping series), just one flow is present at any moment during the video transmission, which can thus use all the available bandwidth. On the other hand, the output bandwidth of the shaping machine is changed three times to simulate a variable bandwidth: 600kb/s for the first minute, 2300kb/s for the second minute, and traffic shaping was stopped for the last minute (hence the output bandwidth is the original wireless bandwidth). This series allows to see the effect of ZAAL in a network where bandwidth changes over time. For the second series, ten flows are present during the whole transmission, which clearly exceed the available bandwidth when using the highest bitrate. This allows to see what happens when multiples flows sense the available bandwidth, especially to check if this leads to a wide oscillation in performance (i.e. if some flows monopolise the entire bandwidth). All the flows use VAAL [12] as adaptation algorithm. Additionally, either all the flows use ZAAL, or none of them.

This section presents two results: ZAAL avoids zigzag quality switching, and using ZAAL leads to similar quantitative results.

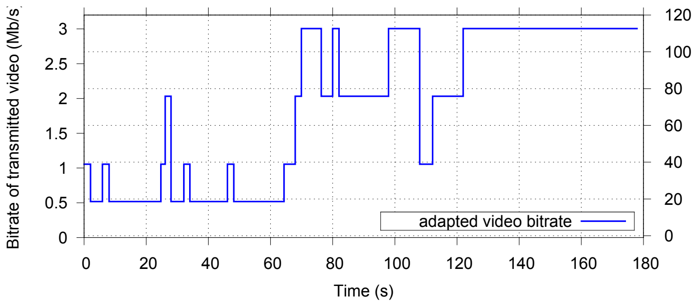
5.1 Zigzag avoidance

In this section we present the quality variation with and without ZAAL. In the following figures, the x-axis represents the time from 0 to 180s, the duration of a video transmission, and the y-axis shows the video bitrate (with or without ZAAL).

One flow in case of traffic shaping As expected, ZAAL minimises the zigzag effect: during the first minute in Fig. 4(a) (where ZAAL is not used), the video bitrate is continuously toggling between 0.5 and 1Mb/s, while when ZAAL is used, in Fig. 4(b), application uses 0.5Mb/s bitrate most of the time. The same conclusions can be drawn from the second minute. During the last minute, bandwidth is wide enough to support 3Mb/s, and ZAAL finally allows this bitrate.



(a) without ZAAL: many zigzags occur



(b) with ZAAL: few zigzag occur

Fig. 4. Quality adaptation for one flow in case of traffic shaping, under the *same* network conditions.

Further analysis of Fig. 4(b) (with ZAAL) confirms the useful properties of ZAAL algorithm. First, application waits for at least 2 periods of time before trying a bitrate which has recently caused losses (e.g. at the beginning, 1Mb/s did not work between 0s and 2s, hence it was retried not at 4s, but at 6s). Second, when a bitrate causes losses for many consecutive times, application waits more and more time to retry it (e.g. 1Mb/s at 24s, then at 32s, afterwards at 46s, and finally at 64s). Third, the maximum period during which the video quality was prevented to increase is 14s, as shown in section 4.2, point 3 (e.g. the first unsuccessful attempt at 10s followed by the next successful attempt at 24s, the same for 50s and 64s); during that time there were no losses (bandwidth much higher than bitrate), however ZAAL correctly prevented the bitrate increasing.

More specifically, Tab. 1 presents the number of zigzags during the first and the second minute (there is no difference during the third minute). It is clear that ZAAL leads to much fewer zigzags (4 against 13 in first minute, 1 against

10 in second minute, so about 80% less in total). Naturally, this has a very big impact on the video quality perceived.

Method	*First minute*	*Second minute*	*Total*
Without ZAAL	13	10	24
With ZAAL	4	1	5

Table 1. Number of zigzags with and without ZAAL.

Ten flows In this experiment, we noticed that all flows have the same tendency when choosing video bitrates. Fig. 5 shows the tendency of one flow out of the ten concurrent flows. We can see that the bitrate changes often between 1Mb/s and 2Mb/s, which indicates that one flow does not stay all the time at a high bitrate (e.g. 3Mb/s); if this happened, it would reduce the available bandwidth for other flows. Also, as in the previous test, application adapts often video quality while avoiding bitrates causing lost packets (e.g. in Fig. 5, 3Mb/s bitrate causes lost packets at 26s, then it is not used until 68s). At the same time, it avoids frequent changes in video bitrate during transmission while improving the use of the bandwidth.

This test checks also the fairness when using ZAAL for ten concurrent flows. Fig. 6 shows the percentage of sent packets by each flow at application layer when using ZAAL. It shows that ZAAL maintains the fairness among concurrent flows on server (i.e. all flows have nearly equal percentage of sent packets). On the other hand, the fairness on the network is guaranteed by using a TCP-friendly congestion control.

5.2 Performance comparison

Even if reducing packet loss rate is not the main goal of ZAAL, we investigate how ZAAL affects it. We consider the number of received packets and the number of lost packets. Video quality over network transmission is more affected by packet losses rather than video bitrate value. Table 2 presents numerical results of the same experiments. For one flow in case of traffic shaping, it is clear that the number of received packets is lower when ZAAL is used (e.g. only 40263 received packets for ZAAL compared to 42043 without ZAAL), because of ZAAL preventing high bitrate for some period. On the other hand, when ZAAL is used the flow has 30% fewer lost packets (under the same network conditions). The average of the ten concurrent flows gives yet better results for ZAAL, i.e. number of received packets are about 5% more in case of ZAAL, and the loss rate is about 50% smaller too.

To resume:

– in the first experiment, it cannot be easily decided which is better in terms of number of sent and received packets, but using ZAAL is more useful because it avoids the zigzag and leads to a more stable video quality;

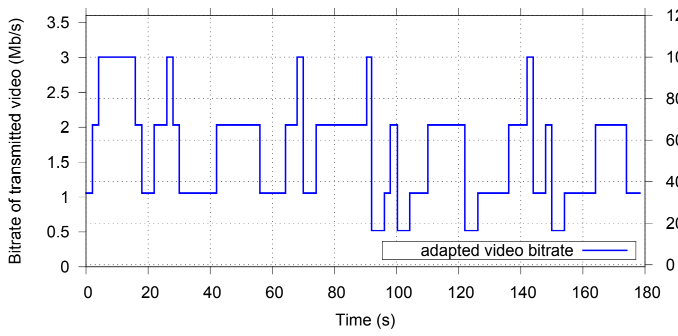


Fig. 5. Quality adaptation with ZAAL for flow number 1 in ten concurrent flows test: few zigzags occur.

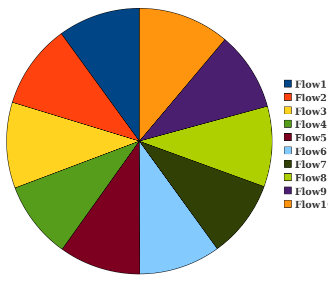


Fig. 6. Percentage of sent packets by each flow at the application layer using ZAAL: the percentages are nearly equal.

– in the second experiment, ZAAL is better in terms of sent and received packets, avoiding the zigzag in the same time.

We can conclude that ZAAL is better and even if sometimes its number of received packets is lower, it reduces the rate of lost packets while maximising the use of the bandwidth.

	Traffic shaping			*10 concurrent flows*		
Method	*Sent pkts*	*Received pkts*	*Lost pkts*	*Sent pkts*	*Received pkts*	*Lost pkts*
Without ZAAL	47795	42043	5752 (12%)	41191	32307	8884 (21.6%)
With ZAAL	43913	40263	3650 (8.4%)	38105	33889	4216 (11%)

Table 2. Number of sent and received packets (average of all flows) with and without ZAAL.

6 Conclusions and perspectives

This article has presented a simple brand new method to avoid the undesirable constant switching in quality occurring during a video streaming using content adaptation. It is a general solution, since it can be integrated to any adaptation method, and no matter if the video is encoded in multiple qualities or in multi-layers. The proposed solution uses a history of quality successfulness to infer if a quality should be chosen at a given time. It is a non intrusive method, i.e. it does not change the video transmission and does not need feedbacks from the network. Experiments confirm that with our solution the problem of zigzag quality switching appears very rarely, without much influence on the video throughput.

A perspective of our work is to consider a hybrid solution, which uses a bandwidth estimation method (to decide whether to increase or not the quality) when the maximum period of non-increasing with our method is reached. However, a very interesting future work is to analyse a more general algorithm with VAAL/ZAAL constraints, but applicable to the whole class of network congestion control methods.

References

1. Robert L. Carter and Mark E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, 27-28:297–318, October 1996.
2. Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Trans. Netw.*, 12:963–977, December 2004.
3. Nick Feamster, Deepak Bansal, and Hari Balakrishnan. On the interactions between layered quality adaptation and congestion control for streaming video. In *11th International Packet Video Workshop*, April 2001.
4. Wu-chi Feng. On the efficacy of quality, frame rate, and buffer management for video streaming across best-effort networks. *Journal of High Speed Networks*, 11:199–214, 2002.
5. Sally Floyd, Mark Handley, Jitendra Padhye, and Joerg Widmer. TCP Friendly Rate Control (TFRC): Protocol specification, September 2008. RFC 5348.
6. ITU-T. Opinion model for video-telephony applications, April 2007.
7. V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18:314–329, August 1988.
8. Raj Jain and Shawn A. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4:986–995, 1986.
9. Srinivasan Keshav. Packet-pair flow control. *IEEE/ACM Transactions on Networking*, February 1995.
10. Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. *SIGCOMM Comput. Commun. Rev.*, 26:117–130, August 1996.
11. Jon Postel. Transmission control protocol, September 1991. RFC 793.
12. Wassim Ramadan, Eugen Dedu, and Julien Bourgeois. VAAL, video adaptation at application layer and experiments using DCCP. In *WPMC, 13-th Int. Symposium on Wireless Personal Multimedia Communications*, pages 1–5, Recife, Brazil, October 2010.