Copyright ⓒ2016 KSII

Server Side Solutions For Web-Based Video

Arkadiusz Biernacki
Institute of Computer Science
Silesian University of Technology
Akademicka 16, 44-100 Gliwice, Poland
E-mail: arkadiusz.biernacki@polsl.pl

Received May 25, 2015; revised September 1, 2015; revised September 30, 2015; accepted January 10, 2016; published April 30, 2016.

Abstract

In contemporary video streaming systems based on HTTP protocol, video players at the client side are responsible for adjusting video quality to network conditions and user expectations. However, when multiple video clips are streamed simultaneously, an intricate application logic implemented in the video players overlays the TCP mechanism which is responsible for a balanced access to a shared network link. As a result, some video players may not obtain a fair share of network throughput and may be vulnerable to an unstable video bit-rate.

Therefore, we propose to simplify the algorithms implemented in the video players, which are responsible for the adjustment of video quality and constrain their functionality only to sending feedback to a server about a state of the player buffer. The main logic of the system is shifted to the server, which is now responsible for bit-rate selection and prioritisation of the video streams transmitted to multiple clients.

To verify our proposition, we performed several experiments in a laboratory environment which show that when the server cooperates with the clients, the video players experience fewer quality switches and the system achieves better fairness when allocating network throughput among the video players. However, this comes at the cost of worse utilisation of network bandwidth.

Keywords: Video streaming, Adaptive Video, HTTP Video, Video scheduling, Multimedia QoS, QoE

http://dx.doi.org/10.3837/tiis.2016.04.017 **ISSN : 1976-7277**

1. Introduction

Throughout the past years, web-based video sharing services such as YouTube, Hulu or Dailymotion became very popular. YouTube users alone request millions of videos every day. Consequently, the popularity of this kind of services results in a drastic shift in Internet traffic statistics, which reports the increase of traffic for web-based video sharing services [1].

Video streaming in the above-mentioned services is either web-based or HTTP-based; therefore, being transported using TCP. HTTP and TCP are general purpose protocols and were not primarily designed for streaming of multimedia. Thus, attempts are being made to adapt the delivery of multimedia content to the Internet environment. One of such attempts tries to introduce an additional layer of application control to transmitted video traffic. An application may limit the rate at which data is passed to a network stack for transmission. The TCP control mechanisms still apply, although the effect of an application control flow may be reduced if the connection is already limited by a receiver or a congestion window [2].

In the case of multimedia transmission, many HTTP servers limit the data rate by dividing a file into chunks and writing them into a network socket at fixed time intervals [2][3]. As a result, TCP is used to transport this traffic, but its transmission rate is controlled by an application layer. If a video bit-rate is lower than end-to-end available bandwidth, the traffic characteristics will not resemble a typical TCP flow. Instead, ON-OFF cycles are produced, where during the ON time, a block of data is transferred at the end-to-end available bandwidth that can be used by TCP, and during the OFF period, the TCP connection remains idle.

An HTTP server usually transmits multiple streams simultaneously. Such a situation takes place when, for example, there are several concurrent sessions initiated by clients located within an Internet service provider (ISP) network as shown in **Fig. 1**. The clients' players compete for bandwidth over the shared access link, which may become a bottleneck if the bandwidth demand exceeds the link capacity.

On the one hand, on a network level, the capacity assignment is handled arbitrarily by the control mechanism of TCP. Sometimes, it is possible that streams with similar needs get strongly different shares of the available capacity. It may take place, for example when there are differences in the RTT (round-trip time) between a server and the connected clients, what is a quite common situation. As it was showed in [4], competing streams with higher RTT times receive poorer bandwidth compared to those with lower RTT. Consequently, video players located in wireless and mobile networks, which have usually higher RTT, may struggle with low buffer levels and, in order to prevent play-out interruptions, will be forced to reduce the quality of video. Moreover, the overlapped ON-OFF cycles produced by the video players make things worse, for they increase traffic burstiness, which will lead to performance degradation of the transmission [5].

On the other hand, the application level, the transmission is controlled by video players which may implement the different logic of pulling video chunks from a server. A video stored within the chunks have different quality; thus, the client software decides which chunks should be downloaded, taking into account if network conditions allow to play-out the video at the requested quality, e.g. by measuring network throughput or by observing a level of the player buffer. It is said that the video player adjust or adapts video quality to the current network environment. As it has been observed, multiple clients connected to a single server, left without control, started competing among each other, what led to different performance issues, e.g. video streams directed to the players having poorer network conditions were displaced by those which were directed to the ones located in more favourable network environments [6][7].

Furthermore, the strategies implemented in clients' software responsible for the adjustment of video quality, i.e. pulling respected chunks from a web server, are beyond any external control. Therefore, it is quite probable that if an algorithm implemented in a video player aims at maximisation of video quality, its greedy behaviour will impact other players which employ more fair strategies. As a result, some users can get a far better quality, compared to the others, depending solely on a video player they are using.

In order to prevent the above undesirable scenarios, we propose to shift the adaptation logic from a client to the side of an HTTP server. In our solution, the server has control over an assignment of video chunks to the client. Thus, instead of pulling the video chunks from the server by the client, the server pushes the chunks to multiple clients. Similarly to the pull-based approach, the decision about the allocation of the chunks may be based on the current network conditions. To simplify the implementation and to keep the number of parameters at a minimum, in our work the decision is based on the state of the player buffer, which to a certain extent reflects these conditions. Contrary to the pull-based technique, in which players keep information about their buffer state only to themselves, in our push-based approach, the server has global information about all ongoing streaming sessions and buffers state of their respective receivers. The clients no longer decide about quality and timing of downloaded video chunks but merely report their buffer state to the server. Thus, we are able to avoid the scenario where clients implement competing play-out algorithms which try to outsmart one another. Furthermore, as our experiments will show, the push-based technique increases fairness among video players and, to a certain extent, stabilises quality of video transmitted to users.

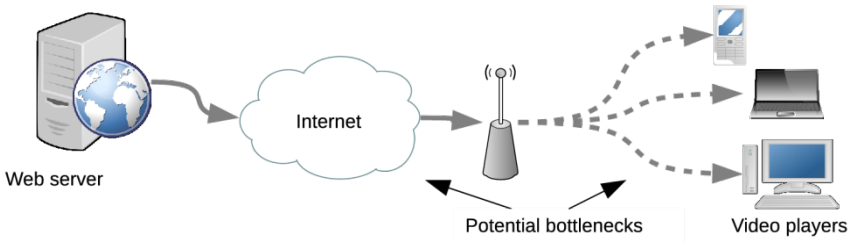


Fig. 1. Video streaming scenario employing HTTP

2. Traffic control at an application level

2.1 Client-side approach

As it was mentioned, the modern video players implement the technique of video bit-rate switching. From the technical point of view, the content, which is stored at a web server, is encoded at different bit-rate levels. Next, an adaptation algorithm selects a video level, which is to be streamed, based on a state of a network environment or on a state of the video player. In the former case, the video player estimates how fast the server can deliver video (i.e. the available capacity), for example by measuring an arrival rate of video data. Then, the player chooses the video bit-rate which corresponds to the estimated network throughput. In the latter case, instead of estimating network capacity, the algorithm directly observes and controls the playback buffer believing that the buffer occupancy reflects the end-to-end system capacity, including current load conditions of the network [8]. Thus, when the buffer reaches a certain size for a given quality level, the system is allowed to increase the video bit-rate. Similarly, the selected video quality is reduced if the buffer shrinks below the threshold. Hence, the quality

no longer depends directly on bandwidth availability, and during network outages, buffer under-runs and play-out interruptions may be avoided.

We translated the above idea into pseudo-code presented in **Algorithm 1**, which will be exploited for our further analysis and comparisons. The algorithm conserves current video bit rate *q* as long as the buffer occupancy *b* remains within the range denoted by B_{max} , line 1 and B_{min} , line 6. This buffer range plays a role of a cushion which absorbs rate oscillations. If the buffer level reaches either high or low limit, the adaptation algorithm switches the bit-rate up or down respectively. With the increasing quality, the video bit-rate rises non-linearly, therefore, it is not practical for the buffer threshold to depend directly on the amount of data accumulated in the buffer measured in bytes, but rather on the amount of data measured in video frames, which may be translated into a number of seconds of video preloaded in the player buffer.

Input: q – video bit-rate [kbit/s]
b – buffer occupancy [s]
$B_{\min}$, $B_{\max}$ – buffer thresholds [s]
$Q_{\min}$, $Q_{\max}$ – the lowest and the highest video bit-rate [kbit/s]

```
1:  if B_max < b then
2:    if q < Q_max then
3:      q ← q + 1
4:    end if
5:  end if
6:  if B_min > b then
7:    if q > Q_min then
8:      q ← q − 1
9:    end if
10: end if
```

Algorithm 1. Adaptive algorithm for HTTP video based on buffer observation implemented at a client side

2.2 Server-side approach

The pull-based approach has some drawbacks, mainly concerning fairness and stability of video quality resulting from competition among video players. In order to overcome these disadvantages, we propose to shift the play-out logic presented in **Algorithm 1** to the server side, as shown in **Fig. 2**. This alteration allows us to deal with the problems of unfair sharing and oscillations of video-bit rates by implementing the Weighted Round Robin (WRR) scheduler at the server, which will be responsible for management of video chunks. The players, instead of using network QoS parameters for planning and scheduling the download of video chunks, merely report to the server a level of their buffers. On the base of this feedback information, the WRR scheduler decides which clients should be treated with a priority at a given time and which clients can wait for their service. It is assumed that the clients send information about their buffers repeatedly with time interval Δt. The clients do not implement any complex logic except measuring their buffer level; therefore, the implementation of the client part is trivial.

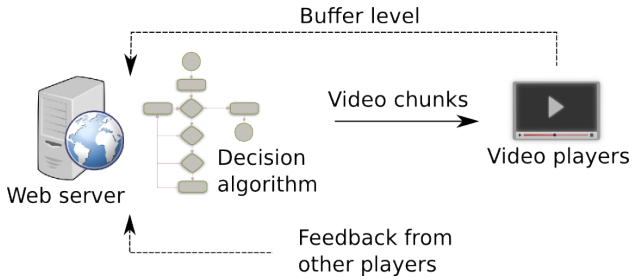


Fig. 2. Based on the client feedback, the server decides on the scheduling of video chunks

The server side logic is based on modified **Algorithm 1**, which now includes elements of the WRR scheduler responsible for handling of stream priorities. The logic is split into the separate algorithms handling two primary conditions: the client reports that its buffer is either below the specified threshold B_{min} or above the specified threshold B_{max} .

We start with the short analysis of the **Algorithm 2** which tests the former condition in line 1. Assuming that the player is at risk of a buffer under-run, we check in line 2 if it possible to increase its service priority. In such a situation, the server sends more frequently video chunks to this particular client than to the others. However, if the priority has been already increased and apparently it did not help, as the algorithm returned again to line 5, the server tries to switch the video-bit rate to a lower level. The server repeats these steps as long as the level of the player buffer is higher than the threshold B_{min} or the client receives the lowest possible quality of video, simultaneously having privileged status denoted by a higher priority than the others.

Input: q_i – video quality played by the i-th video player [s]
b_i – reported buffer level by the i-th video player [s]
p_i – priority assigned to the i-th video player [s]
q_i – quality level received by the i-th video player [s]
$B_{\min}$, $Q_{\min}$ – described in algorithm 1

```
1:  if b_i < B_min then
2:      if p_i ≤ 0 then
3:          p_i ← p_i + 1
4:      else
5:          if q_i > Q_min then
6:              q_i ← q_i − 1
7:              p_i ← 0
8:          end if
9:      end if
10: end if
```

Algorithm 2. Decrease of a video bit-rate by the server.

The latter condition is handled by the **Algorithm 3**. After discovering in line 1 that the level of the client buffer exceeds the threshold, the server checks the priority of the client in line 2. If the priority is high, the server adjusts it to a neutral value, thereby depriving the client of any special treatment. If the client still reports that its buffer level is above the threshold , the algorithm in its next call will try to increase the video-bit rate sent to the client, see line 6. The algorithm will repeat the above steps as long as either the level of the client buffer does

not exceed the specified threshold or the client achieves the highest possible video-bit rate or the server has not enough resources to further increase the video-bit rate.

The proposed scheduling mechanism may be implemented at an HTTP server side or in a residential gateway as a network throughput manager, which can allocate the throughput for individual sessions employing a shaping mechanism based on the WRR algorithm. We assume that the gateway sees all traffic going through, is potentially capable of identifying the streams belonging to different streaming sessions and takes appropriate actions on them.

Input: q_i, b_i, p_i – described in algorithm 2
$B_{\max}$, $Q_{\max}$ – described in algorithm 1
Q – maximal network throughput

```
1:  if b_i > B_max then
2:    if p_i > 0 then
3:      p_i ← p_i − 1
4:    else
5:      if q_i < Q_max and ∑ q_i < Q then
6:        q_i ← q_i + 1
7:      else
8:        if p_i ≥ 0 then
9:          p_i ← p_i − 1
10:       end if
11:     end if
12:   end if
13: end if
```

Algorithm 3. Increase of a video bit-rate by the server.

3. Previous works

Due to the popularity of web-based video systems, play-out algorithms have became an active area of research in the last few years. There are many proposition not only of new algorithms, but also new architectures for whole streaming systems which should improve the performance of video.

As it has been demonstrated, the classical pull-based approach has drawbacks that occur particularly when multiple video players compete among themselves for available throughput needed to transfer video chunks of variable size. The authors of [9] show that inaccurate measurement of received data results in instabilities of a video play-out and a degradation of video quality. They attribute the responsibility for this issue to the ON-OFF pattern of traffic produced by the application layer: even if one video player obtained its fair share sufficiently during the ON period, it could fail to correctly estimate available bandwidth due to the overlapped ON periods with the other player. As a consequence, the former player would switch to a lower rate than the network conditions allow, and network throughput remained underutilised. The authors of [10] examined a video player competing against an another video player, and against a long-lived TCP flow. Interestingly, they demonstrated that inaccurate throughput estimation occurs even when the competing flow does not exhibit the ON-OFF behaviour. Therefore, some research works, e.g. [11], try to improve the algorithm by predicting the future bandwidth, while the others, e.g. [8][12][13], propose an algorithm based on measurement of buffer occupancy. Although, the performance of the later approach has not been examined to the same extend like in the case of the bandwidth estimation algorithms, the buffer reactive algorithms perform fine in many cases, but sometimes they have tendency to

too frequent oscillation between video bit rates [8]. Thus, there are concepts of more sophisticated buffer management, e.g. in [14] the authors present an algorithm which eliminates ON-OFF periods from the traffic through two buffer management states. The authors claim that the proposed scheme improves fairness in the system by 45% compared to the conventional adaptive player. Some of the authors propose more elaborate and complex pull-based strategies, e.g. FESTIVE [15] or PANDA [16] play-out strategies, which take not only network bandwidth or a player buffer into account, but also consider stability, efficiency and fairness. In [17], so called network control plane, designed to take into account scalability and adaptiveness issues, is placed on top of the controlled network. The plane cooperates with distributed buffer-based adaptation techniques implemented at the clients but does not interact with a video server.

Some researches see a chance of overcoming the problems with the bit-rate instabilities and an effective usage of throughput by an engagement of server side mechanisms. For example, in [7] the authors suggested a traffic shaping method, implemented at home gateways, to reduce an extent of observed instability and unfairness in competing video players. Another proposition is a server-based method of traffic shaping that can reduce oscillations of a video bit-rate received by a player [18]. Liu et al. in [19] follow a similar approach where the rate is shaped according to the maximisation of a QoE metrics. The authors of [20] presented a method of video pacing that reduces unnecessary traffic and simultaneously conserves earlier video quality. In [21], the authors suggested to dynamically adjusts a segment size of TCP and a number of video streams in order to optimize throughput of a connection. Another proposal is to locate a traffic shaper between a video server and users' players in order to adaptively transform a video bit-rate to current network conditions; however, this proposition requires a feedback from the players [22]. Finally, in [23] the authors employ a complex strategy in which the use in-network quality optimization agents monitoring the available throughput using sampling-based measurement techniques and optimize QoE of each client. This in-network optimization is achieved by applying centralized as well as distributed algorithms what requires coordination between a server and clients.

As mentioned, our proposition belongs to the category of push-based systems. The main contribution of our work is a unique play-out strategy which employs scheduling and prioritisation mechanisms at a web server what allows for a centralised and active management of video streams. Furthermore, another advantage over the other mentioned approaches is that the clients do not have to implement any elaborate play-out algorithms, for they only report the state of their buffer to a server. The proposed approach improves received video quality compared to traditional pull-based technique: the video streams experience fewer quality switches and the whole system achieves better fairness when allocating network throughput among the video players.

4. Traffic segmentation and scheduling

4.1 Paced streaming

As mentioned earlier, the flow control implemented at the application layer limits the transmission rate of the video stream by a periodical transfer of video blocks separated by idle periods. One of the methods used for transferring data blocks is the so-called "Zippy pacing", which delays a delivery of a new chunk after the video server finishes sending the former chunk [24]. Technically, the video content is divided into N chunks, $\{c_1, \ldots, c_N\}$, where the duration of chunk c_i is d_i seconds ([s]). At first, the server sends the chunks with no delay

until the play-out buffer reaches a sufficient amount of video data. After the initial burst delivery, "Zippy pacing" calculates the pacing delay δ_i [s] assuming that it takes τ_i [s] for the video server to finish sending the chunk c_i ,

$$\delta_i = \max\{d_i - \tau_i, \delta_{\min}\} \tag{1}$$

where δ_{min} indicates the minimum delay time and should ensure a session fairness. Its value may be configured, e.g. by a server administrator.

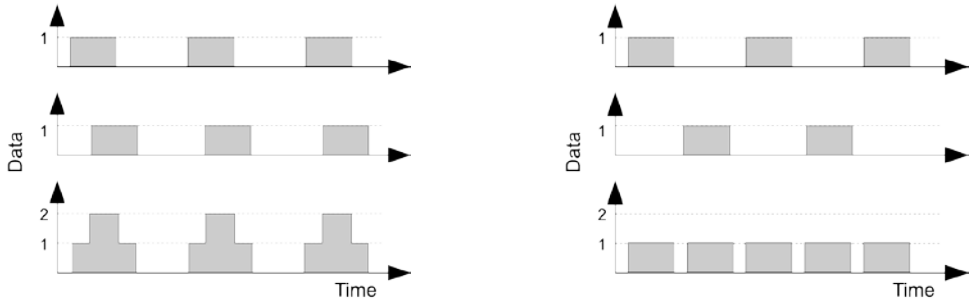
The video server tries to send the next chunk c_{i+1} , if available network throughput allows, at the time δ_i after the video server completes sending the chunk c_i . This transmission strategy does not require feedback on the client's part, as the server algorithm alone determines the parameters of the traffic pattern. Depending on the system, a different length of the chunks has been observed, e.g. the dominant block size for Flash videos is 64 kB, and 256 kB for HTML5 on the Internet Explorer [3].

4.2 Multiplexed segments

Theoretically, the variable bit rate video additionally streamed with the "Zippy pacing" algorithm can improve network utilization by exploiting the potential of statistical multiplexing gains. However, this potential can be diminished when the transmitted video chunks are not dispersed but overlap in a time domain.

In [2], it was suggested that the ON-OFF pattern of traffic induced by the application logic is responsible for the packet loss, especially given that no loss was observed during the initial buffering phase. The application pushes the video block onto the TCP stack, while the congestion and receive windows are empty by the time the block has been written. As the idle time prior to writing the block is not long enough for a reduction of the congestion window, the entire block is transmitted immediately as a burst of packets, and, as a result creating congestion [25]. Moreover, other competing streams may increase their congestion window in the interim, further contributing to an overload of the network link. Such short-term congestions lead to a much higher probability of packet loss and force TCP to reduce the congestion window what reduces throughput for the video stream.

There is also the problem of a temporal overlap of ON-OFF periods among competing players. In **Fig. 3a**, we may see the situation where the ON-OFF periods of the two players are adjusted randomly and partially overlap. Statistically, without any special scheduling mechanisms, this is the most common scenario and the players may not estimate in such a scenario their fair share correctly [9]. Thus, the more desired situation is when the ON periods do not overlap but are spatially separated, as shown in **Fig. 3b**. For this purpose, we use the WRR algorithm which assigns a time slice for every video block in a circular order, furthermore setting priority to some of the players when needed. The details of this mechanism are presented in the next section.



(a) Random ON-OFF pattern (b) Round Robin schedule for the ON periods

Fig. 3. Scenarios for the multiplexed ON-OFF periods of two video competing video players

4.3 Scheduling of video segments

Following the notation introduced in **Eq. (1)**, the server now streams data to *M* players and every stream is divided into *N* chunks. We assume that every video chunk c_{ij} has length d_{ij} where $i \in \{1..M\}$ and $j \in \{1..N\}$. Basically, the chunks in the video stream are treated as virtual queues, what illustrates **Fig. 4.** In every turn, the WRR scheduler picks and transfers a single chunk from the *i*-th queue. The selected chunk is followed by $c_{i+1,j}$ until *i*=*M*, which denotes that all the virtual queues completed their turn. Then the transmission starts again from the first queue and the variables i and j receive new values: $i \leftarrow 1$ and $j \leftarrow j+1$. The algorithm continues as long as there are chunks to be send.

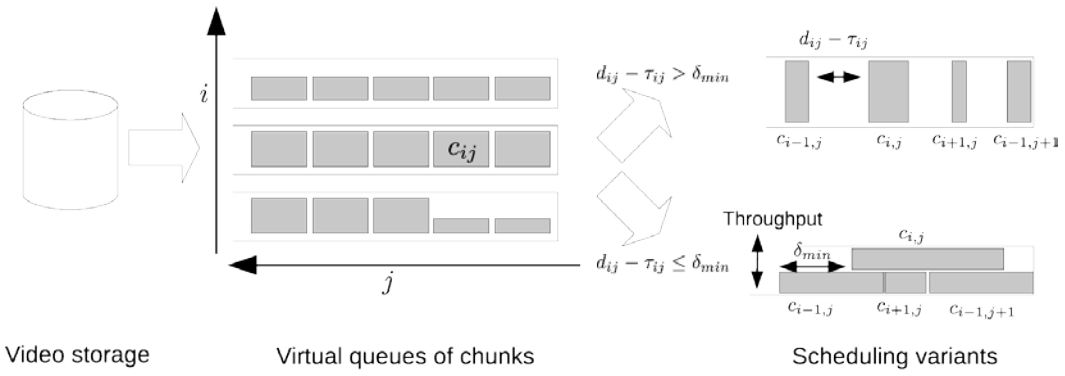


Fig. 4. "Zippy pacing" transmission of multiple video streams employing the WRR scheduling for video chunks. Depending on the available network throughput, two variants of the transmission are possible.

The pacing delay δ_{ij} of a chunk is calculated as

$$\delta_{ij} = \max\{d_{ij} - \tau_{ij}, \delta_{\min}\}, \tag{2}$$

where, similarly to **Eq. (1)**, τ_{ij} is time needed by the server to finish sending the chunk c_{ij}. The video server sends the next chunk $c_{i+1,j}$ at the time δ_{ij}, after it completes the transfer of the chunk c_{ij}. The value of δ_{min} is set in our experiments arbitrarily to 0.1 s.

From **Eq. (2)** we can deduce that two temporal scheduling patterns take place. In the first case, when the available network bandwidth exceeds the aggregate video bit rate of all the concurrently transmitted video streams, there are OFF gaps between the transmitted chunks, see again **Fig. 4**. In the second case, the available network throughput is lower than the aggregated video bit rate, and as a result, the second argument in **Eq. (2)** is taken into account, i.e. $d_{ij} - \tau_{ij} \leq \delta_{min}$. In such circumstances, the transmission of each new chunk will start every δ_{min} seconds. The transmitted video chunks will overlap with each other and they will compete for the limited network throughput, leaving no OFF gaps in the multiplexed data transmission. However, the chunks are still scheduled deterministically; thus, compared to the randomly adjusted chunks from **Fig. 3**, the risk of overlapped chunks which are perfectly aligned due to the control mechanism of a server application, is minimised.

4.4 Prioritisation of video streams

As we previously described, **Algorithms 2** and **3** may change the priority of the scheduled video streams. Technically, when **Algorithm 3** sets the priority of the *i*-th stream to „high", the scheduling mechanism sends an additional chunk of data to the respective player, as presented in line 4 of **Algorithm 4**. If the priority of the stream is neutral, the scheduler sends only one chunk of data, line 7. In the case of low priority, no chunk should be sent and the service of the player should be skipped, line 10. As a result, after introducing the priorities, in every turn, up to two chunks can be pulled from a virtual queue independently from the other queues. Therefore, the chunk index now is no longer a global variable but is dependent on the queue index, i.e. we replace c_{ij} with $c_{i,j[i]}$.

```
Require: M – number of players
         c_{i,j} – the j-th chunk transmitted to i-th player
 1: while ∃ not empty stream do
 2:   for i = 1 to M do
 3:     if priority i-th stream is high then
 4:       send(c_{i,j[i]} and c_{i,j[i]+1})
 5:     end if
 6:     if priority i-th stream is neutral then
 7:       send(c_{i,j[i]})
 8:     end if
 9:     if priority i-th stream is low or i-th stream is empty then
10:       continue
11:     end if
12:   end for
13: end while
```

Algorithm 4. Prioritisation of video streams

In our work, for simplicity, we assumed that the transmission of the streams starts simultaneously at the same time increased by additional time resulting from chunk scheduling algorithm. Nonetheless, it is only matter of technical details to allow the described technique to handle a churn of the streams.

5. Experiments

5.1 Laboratory set-up

In order to capture performance of the examined algorithms, we prepared a test environment which was able to emulate operating parameters encountered in wire, wi-fi and mobile networks. The environment consisted of: a web server, video players, a network emulator and a measurement module implemented in the video player, as shown in **Fig. 5**.

We implemented the web server in Python using, among others, *http.server* module. We extended its functionality by adding a transmission control at the application layer presented in **Algorithms 2**, **3** and **4**.

As a video player, we chose VLC media player with a web-streaming plug-in [26]. Both the player and the plug-in have an open-source code, thus, we were able to manipulate and completely change the adaptation logic without affecting the other components. As a consequence, the plug-in allowed us to implement and to integrate **Algorithm 1** with the player.

The Linux kernel module *netem* [27] emulated the network environment. The module is capable of altering network QoS parameters such as throughput or delay of a network link; thus, it allowed us to test data transmission in different network environments.

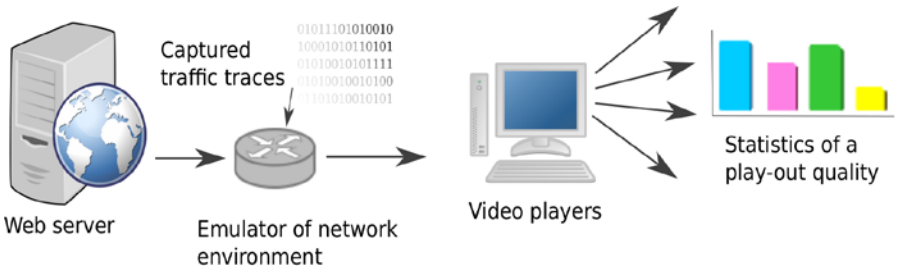


Fig. 5. Laboratory environment employed for the experiments

We transmitted several video files, acquired from [28] and presented in **Table** 1, through the laboratory environment with variable network conditions. The bandwidth traces were obtained from measurements conducted in wire, wi-fi and HSPA networks. For this purpose, we implemented a custom made analysis tool that transferred data in a fixed, configurable rate using UDP packets. Every transmitted packet had its sequence number, enabling the receiver to precisely detect possible packet loss. The packets were sent at a fixed rate, and the packet reception rate was logged. Because download performance is much more important than upload performance for a one-way video streaming scenario, all tests measured the download performance.

Table 1. Video clips used for the experiments

Name	Genre	Bitrate levels
Big Buck Bunny Elephants Dream Red Bull Playstreets The Swiss Account Valkaama Of Forest and Men	animation animation sport sport movie movie	150 kbit/s – 320x240, 300 kbit/s – 480x360, 600 kbit/s – 854x480, 1.2 Mbit/s – 1280x720, 2.5 Mbit/s – 1920x1080

The captured log was used as a template for the bandwidth shaper implemented in the mentioned earlier *netem* module. In addition to the bandwidth throttling, the *netem* module also adds a delay of 5 ms to wire, 20 ms to wi-fi and 100 ms to HSPA connections to emulate the average latency which was measured during the gathering of the bandwidth traces. Furthermore, we used also the bandwidth traces as a source for reproducing packet losses. Thus, the statistics of the losses and their distribution should mirror the losses in a real network. In order to obtain desirable, average network throughputs, which were used in the experiments, the traces were rescaled. Thus, having identical bandwidth traces, we were able to perform a quite fair and realistic comparison of the pull- and push-based approaches.

We believe that the above-described methodology provides an attractive middle ground between simulation and real network experiments. To a large degree, the emulator should be able to maintain the repeatability, reconfigurability, isolation from production networks, and manageability of simulation while preserving the support for real video adaptive applications.

Using our laboratory environment, we compared the pull- and push-based approaches. Additionally, we added to the comparison two server-side algorithms presented in [19] and [18] and mentioned in Section 3, further referred as Liu-Men and Akhshabi respectively. The parameters of the algorithms are specified in **Table 2**. In all approaches, we played first 600 s of every video clips presented in **Table 1**. Every experiment was repeated five times and its results are presented as box plots which include information not only about average results of the experiments but also about their variability.

Table 2. Algorithms and their parameters used in the experiments

Algorithm	Parameters
Client side (**Algorithm 1**)	$B_{min} = 3s$, $B_{max} = 7s$, $Q_{min} = 150kbit/s$, $Q_{max} = 2.5Mbit/s$
Server side (**Alg. 2** & **3**)	B_{min} , B_{max} , Q_{min} , Q_{max} – as above, $\Delta T = 5s$
Liu-Men [19]	the scenario *clients with no grade*
Akhshabi et al. [18]	the scenario *shaped*

5.2 Performance measures

The quality of experience (QoE) is defined as the overall acceptability of an application or service quality perceived by the end-user. The QoE, based on popular methods reflecting

human perception, is a subjective assessment of multimedia quality. A user is usually not interested in performance measures such as packet loss probability or received throughput, but mainly in the current quality of the received content. However, the quality assessment is time-consuming and cannot be done in real time; therefore, we concentrate on these parameters which we believe impact the QoE at most. In this respect, we assess the performance of the play-out algorithms using the following measures: efficiency, stability, fairness and bandwidth utilisation.

An adaptive algorithm based on the buffer assessment by its nature will avoid breaks in video play-out occurring because of an empty buffer. The algorithm will rather decrease the play-out quality, thus minimising the probability of the buffer under-run and simultaneously keeping the stalling time at a possible minimum what will lead to relatively low video quality and poor utilisation of available network bandwidth. Hence, in our work we assess how effectively the algorithm utilises available network resources by computing a value of the following formula

$$\text{efficiency} = (\sum_{i}^{M} \frac{\sum_{j}^{N} (q_{ij} / \tilde{q_{ij}})}{N}) / M. \tag{3}$$

Eq. (3) computes the relation between the quality level q_{ij} of the chunk c_{ij} (as introduced in section 4.3) to the theoretical quality level $\tilde{q}_{ij}$ which is possible to achieve for the chunk in given network conditions. The formula is computed and averaged for every player M connected to the server. The minimum value of the formula is Q_{min} / Q_{max} if the play-out quality of every chunk c_{ij} is Q_{min}, although the network conditions allow for the Q_{max} quality. The value of the formula can reach one if, and only if, for every chunk $q_{ij} = \tilde{q}_{ij}$.

The play-out algorithm may try to maximise the value of **Eq. (3)** by adjusting the play-out quality to given network conditions as frequently as it is possible. Such behaviour will result in rapid oscillations of video quality, what will be negatively perceived by users [29][30]. For this reason, we introduce the second measurement, which sums the quality switches of every player *i* and then counts their average number for *M* players connected to the server. We compute the formula as follows

$$\text{switches} = \frac{\sum_{i}^{M} \sum_{j}^{N-1} |q_{ij+1} - q_{ij}|}{M}. \tag{4}$$

We can imagine a scenario in which one of the players is selfish and demands from the server, with above average frequency, video chunks of the highest quality. If the player has a connection of high throughput, the traffic generated by this player will probably dominate the shared link with other players, pushing aside their traffic. The selfish player may achieve quite a good efficiency, however at the expense of the other players, especially situated in wireless environments with less stable link performance. Therefore, we employ Jain's fairness index [31]

$$\text{fairness} = (\sum_{i}^{M} \overline{q}_i)^2 / (M \sum_{i}^{M} \overline{q}_i{}^2), \tag{5}$$

where $\overline{q}_{ij}$ is an average quality of video played by the *i*-th player. The result ranges from the worst case

equal to 1/*M*, when a single player dominates the others, to the best case equal to one, when all users receive the same quality.

Finally, we investigate how the proposed solution influences network recourses. For this purpose, we compute utilisation of network bandwidth as

$$\text{utilisation} = \text{traffic} / (\text{bandwidth} \cdot \text{interval}), \tag{6}$$

where *traffic* is an number of bytes sent through the network in a given time interval *interval*, which is set to 1 s.

6. Results

We start our experiment with a transient analysis of a scenario with M=24 players divided equally into three groups. The players, which belong to these three groups, are located in wire, wi-fi or wireless mobile networks respectively. Every player has a link with throughput up to about 6000 kbit/s. In the transient analysis, we compare two groups of players where the first group employs the pull-based approach and the second a group uses the push-based approach. Both groups reside in a mobile network. The obtained traces, showing buffer occupancy and video bit-rate, computed as averages from the results registered separately for every player, are presented for each group.

As presented in **Fig. 6**, the registered bandwidth for the mobile network is quite variable with several peaks reaching about 6000 kbps and several valleys during which network collapsed and transferred no data. Buffer utilisation for the pull-based strategy shows a correlation with the registered bandwidth for a mobile player: as denoted in the **Fig. 6A**, during the periods where the throughput is relatively high, the buffer of the player stores about 9 s of the video clip, what was marked in the figure at about 20 s and 340 s of the video play. Similarly, when the throughput drops significantly, the buffer become empty as marked at about 550 s. The high volatility and ragged shape of the bandwidth trace translate into the high variability of the video bit-rate received by the player, as depicted in **Fig. 6B**. Consequently, abrupt changes between the lowest and the highest bit-rates of video are fairly common.

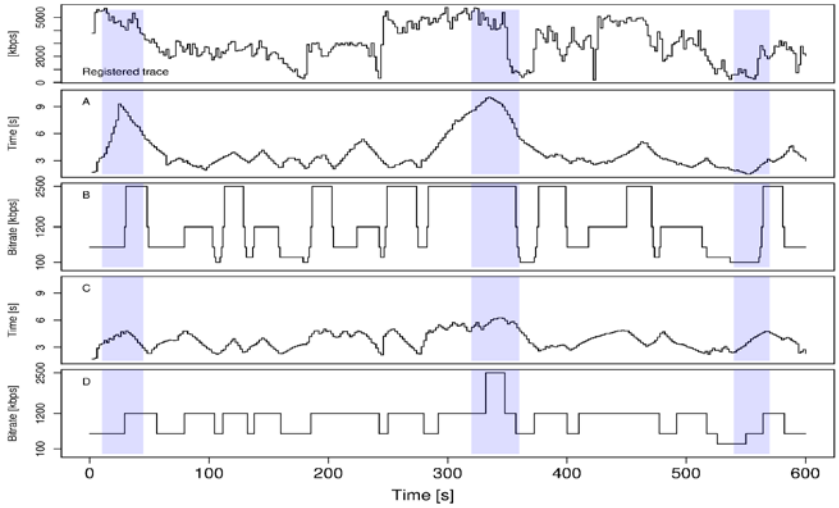


Fig. 6. Comparison of performance between streaming controlled by a client and a server: Buffer utilisation (A) and streamed bit-rate (B) for the client controlled streaming; buffer utilisation (C) and streamed bit-rate for the server controlled streaming (D).

When the streaming of video is controlled by a server, the buffer occupancy flattens, especially within marked intervals, what was presented in **Fig. 6C**. The buffer stores up to 6 s of the video, what is about 3 s less compared to the pull-based streaming. As a result, the video bit-rate is smoother, and it hardly goes beyond the range of 250 kbit/s and 700 kbit/s, as depicted in **Fig. 6D**. The push-based solution avoids buffer under-runs and rapid switches between the lowest and highest video qualities; however, it comes at the expense of streaming the video at a lower bit-rate compared to the pull-based scenario. Taking into account **Eq. (4)**, the number of bit-rate switches for the push-based approach is also lower compared to the pull-based one.

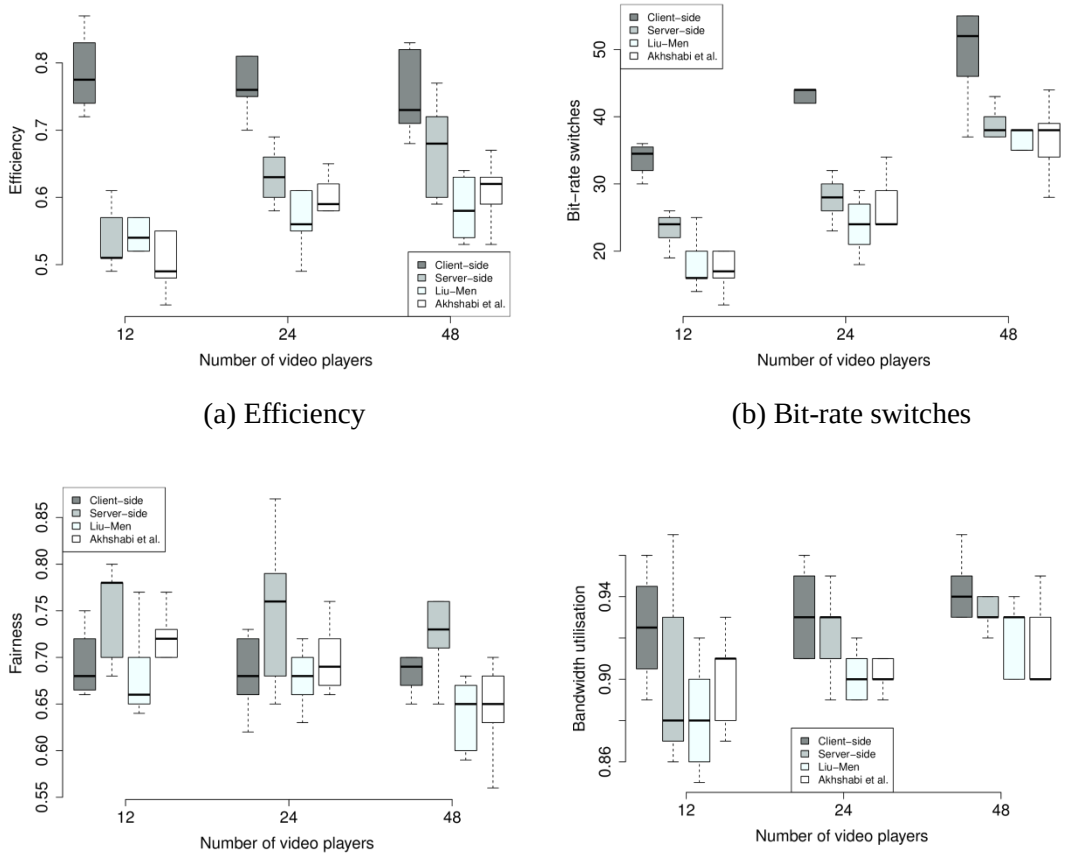
As the performance of the push-based approach seems to be quite promising, we extend our analysis and compute averaged performance measures: efficiency, number of switches, fairness and bandwidth utilisation, defined in **Eqs. (3)-(6)** respectively, for all streams handled by our experimental system. We consider the scenarios where the system transmits simultaneously 12, 24 and 48 streams. Similarly to the first experiment, every player has at its disposal a link with bandwidth up to about 6000 kbps; however, the variability of the throughput, link delay and probability of packets loss are different and depend on the network environment to which the link belongs.

As it could be expected from the transient analysis, the pull-based strategy in terms of efficiency, defined in **Eq. (3)**, dominates over the push-based one, what is presented in **Fig. 7a**. However, the results reveal also certain trend: with more streams handled by the system, the difference between the strategies decreases. And so, for 12 players the difference in the efficiency is about 50% in favour of pull-based technique, however, it shrinks to about 30% for 24 players and diminishes to less than 10 % for 48 sources. For 12 players involved, the Liu-Man and Akhshabi strategies have similar performance to the proposed push-based strategy. Nonetheless, with the increase of the players number, the efficiency of these strategies slightly fall behind the push-based approach.

The number of bit-rate switches for the server-side strategy is significantly lower compared to the client-side strategy, especially for the systems with a lower number of clients, as it was presented in **Fig. 7b**. The pacing technique probably has better performance when the available network bandwidth exceeds the aggregate video bit rate of all concurrently transmitted video streams, i.e. $d_{ij} - \tau_{ij} > \delta_{min}$ in **Eq. (2)**. When this condition is met, video chunks have a chance of being transmitted separately and isolated by periods of no data transmission. Therefore, when the chunks are not overlaid (compare with **Fig. 4**), the prioritisation mechanism proposed in **Algorithm 3** may better handle scenarios with fewer multiplexed streams. When the number of stream rises, the WRR scheduler, described in section 4.3, is less effective as the service time between the virtual queues increases and may be too long to maintain a bit-rate of the video streamed to the player connected to these queues. For 12 video players, the Liu-Man and Akhshabi approaches are more stable compared to the push-based approach. However, with an increasing number of the players, the difference between the mentioned three approaches diminishes.

The push-based technique achieves also better fairness compared to the pull-based one as demonstrated in **Fig. 7c**. For 12 players the difference is about 20%, then it shrinks to about 15% for 24 players, and about 10% for 48 players. For a dozen of players, the fairness of the Akhshabi algorithm is better than the fairness of the Liu-Man strategy, however, the push-based strategy clearly dominates both of them. For the higher number of players, the Liu-Man and Akhshabi approaches have the similar score, which is nevertheless lower

compared to the push-based approach.



(a) Efficiency (b) Bit-rate switches

(c) Fairness (d) Bandwidth utilisation

Fig. 7. Performance comparison for the client and server side strategies for a different number of video players located in a mixed network environment

The utilization of network bandwidth, defined in **Eq. (6)**, is quite similar for all the examined approaches, as depicted in Fig. 7d. It oscillates between about 0.86 to 0.96 for 12 players, and between about 0.89 to 0.96 for 24 and 48 players. Although, the pull-based approach achieves the highest score in all the scenarios, the difference among all the approaches does not exceed several percent.

The preceding analysis showed differences between the client- and server-based approaches; however, as mentioned, the players were located in a mixed network environment. Therefore, the question arises: to what extent the environment influences the evaluated algorithms? In the following experiments, we try to answer this question.

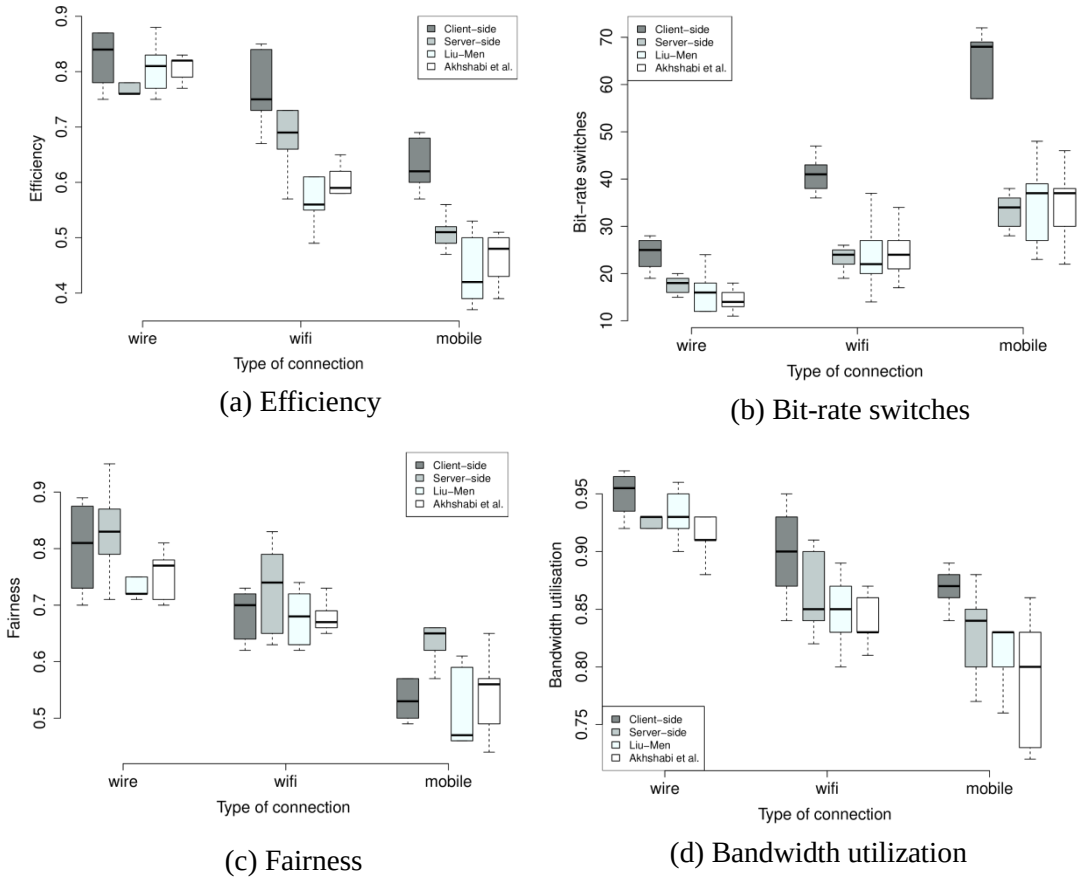


Fig. 8. Performance comparison for the client and server side strategies for 24 video players located in different network environments

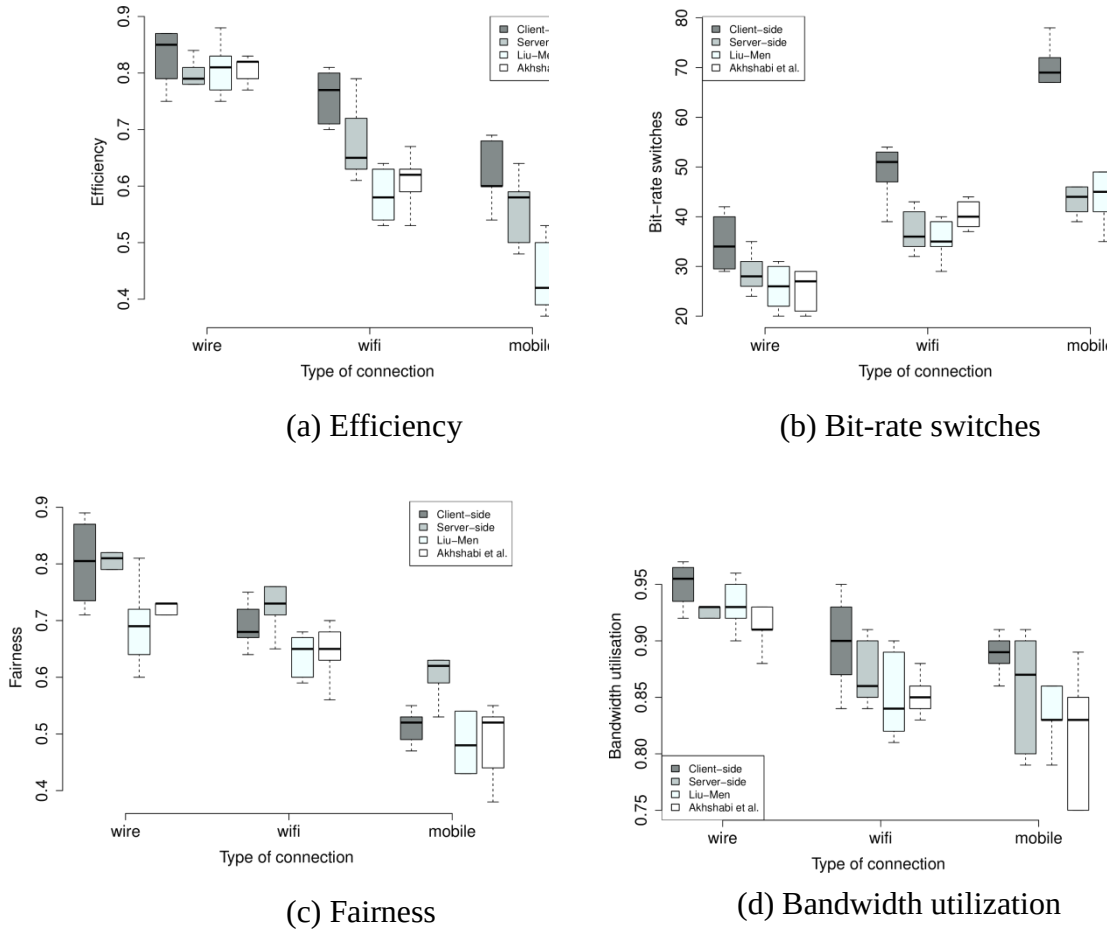
Compared to the push-based strategy, in the case of 24 video players, the pull-based one achieves better efficiency in all types of network environment, what is shown in **Fig. 8a**. Generally, the less stable network condition, the difference between the strategies rises in favour of the client-side approach. Simultaneously, with the decrease of network stability, the efficiency of both strategies drops from about 0.8 in average for a wired to about 0.5 to 0.6 in average for a mobile network. The Liu-Man and Akhshabi strategies have similar scores in a wired network, what places them between the scores of the client-side and the proposed server-side strategies. In the wireless and mobile environments, the Liu-Man and Akhshabi propositions achieve lower results compared to the push- and pull-based approaches.

For the players operating in a wired environment, there are minor differences between the client- and server-side approaches in the number of bit-rate switches, as presented in **Fig. 8b**. However, when we move to less stable network conditions, the gap between the approaches widens. For the wi-fi environment, the pull-based strategy has over a dozen more switches compared to the push-based one, while, in the mobile network, the difference increases to a several dozen. Generally, the number of switches for the two other strategies presented in the literature is similar to the push-based system; nevertheless, the strategies have slightly better scores for wired networks, and a bit worse score for the wireless and mobile cases.

For 24 players placed in a wired environment, the fairness for the pull- and push-based

algorithms is quite similar and achieves about 0.8, what is depicted in **Fig. 8c**. After leaving the stable environment, the fairness for both solution decreases. Nonetheless, the decrease is more significant for the client-side approach, what is noticeable especially for the mobile connections, as the server-side strategy achieves here about 15% better fairness. The score for the Liu-Man and Akhshabi systems in all the cases is lower than the score of push-based system.

The wired environment has the highest bandwidth utilisation, which is similar for all the examined systems, see **Fig. 8d**. In the wireless and mobile networks, the highest result belongs to the pull-based system. The score for push-based system is only little higher than the score for the strategies proposed in literature.



(a) Efficiency (b) Bit-rate switches

(c) Fairness (d) Bandwidth utilization

Fig. 9. Performance comparison for the client and server side strategies for 48 video players located in different network environments

In our last step, we try to assess, what will be performance of the proposed approach when we engage 48 video players. Similarly to the experiment with 24 players, difference among efficiency for 48 players for the wired network is statistically insignificant, see **Fig. 9a**. When we move to the wireless and mobile environments, the pull-based approach still achieves the best score, followed by the push-based system and the propositions acquired from literature.

The number of switches, **Fig. 9b**, is highest for the pull-based system. The other examined systems achieve quite similar scores. Also similarly to the results presented for 24 players in **Fig. 8c**, the push-based approach achieves the best fairness, see **Fig. 9c**.
In the case of 48 video players, the pull- and push-based approaches have in nearly all the cases the highest bandwidth utilization, as presented in **Fig. 9d**. When we take into account the mobile system, the Liu-Man and Akhshabi strategies have lower values.

Summarizing our experiments, the proposed push-based based strategy shows certain advantages over the pull-based one. As the experiments show, the server-side approach results in a more regular and predictable level of buffer occupancy of a video player, what translates into a smaller number of video bit-rate switches and smoother play-out quality. It is especially visible when the number of competing streams is about a dozen. When the number of multiplexed stream increases to a few dozens, the benefits of server-side strategy are less noticeable. When we compare the performance of client- and server-side approaches in wire, wi-fi and mobile networks separately, the results for the wired network are fairly similar. Nonetheless, the server-side strategy clearly shows its superiority in a less stable, wireless network environment. For two and four dozens of players simultaneously streaming data from a server, the server-side technique achieves a fairer distribution of video quality between the players and this quality is more stable. Nevertheless, this comes at the price of lower efficiency and lower utilisation of network bandwidth by the whole system.
The other two reference strategies acquired from the literature, Liu-Man [19] and Akhshabi [18], in the most cases obtain lower efficiency and fairness compared to the push-based algorithm. When taking into account stability of the examined propositions, the reference strategies and our proposition achieve a fairly similar score, what is probably results from the fact that a primary goal of these strategies is to avoid oscillations of video quality. The utilisation of network bandwidth for the presented in the literature strategies is little lower compared to the pull- and push-based propositions what may be a result of intensive traffic shaping employed by the Liu-Man and Akhshabi implementations.

7. Conclusions

Usually, in the systems transmitting adaptive web-based video, a video player is responsible for adjusting video quality to network conditions and user expectations. When single link transfers multiple streams, they start competing for the same access capacity. The control mechanism of TCP, responsible for fair bandwidth allocation among competing streams, is overridden by an intricate application logic of the video players which try to outsmart one another.

We showed that shifting the logic responsible for adjusting video quality from clients to a server may solve some problems encountered in client driven systems: an improper estimation of network throughput by video players, which consequently leads to selecting to high or too low video quality; unstable video bit-rate leading to frequent switches of video quality and unfair allocation of bandwidth among heterogeneous clients. In our work, video streams controlled by the server-side approach experience more stable quality and better fairness when allocating network throughput between video players. However, better stability comes at the cost of worse utilisation of network bandwidth.

When we take two other approaches described in the literature which also rely on the server-side support, our approach has similar efficiency and stability, but has better fairness.

This may be a consequence that the proposed in the literature approaches focus rather on stability of video quality and do not implement any global coordination mechanism among video players.

Furthermore, simplification of client algorithms could prevent a race between developers that are trying to implement new complex solutions which finally will lead only to outwitting the other players when network resources are constrained. As our research shows that the server side solutions has a plenty room for an improvement: one can experiment with parameters of the proposed algorithm or replace them with a quite different type of an algorithm which will judge network condition, e.g. by monitoring network throughput.

References

[1] Cisco, "Cisco visual networking index: forecast and methodology, 2014-2019," *Technical report*, 2015. Article (CrossRef Link).

[2] S. Alcock and R. Nelson, "Application flow control in YouTube video streams," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 24-30, 2011. Article (CrossRef Link).

[3] A. Rao, Y.S. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous, "Network Characteristics of Video Streaming Traffic," *CoNEXT*, Tokyo, article no. 25, Japan, 2011. Article (CrossRef Link).

[4] Lili Qiu, Yin Zhang, and Srinivasan Keshav, "Understanding the performance of many TCP flows," *Computer Networks*, vol. 37, no. 3, pp. 277-306, 2001. Article (CrossRef Link).

[5] Hans-Peter Schwefel and Lester Lipsky, "Impact of aggregated, self-similar ON/OFF traffic on delay in stationary queueing models (extended version)," *Performance Evaluation*, vol. 43, no. 4, pp. 203-221, 2001. Article (CrossRef Link).

[6] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," *ACM MMSys*, vol. 11 pp. 157-168, 2011. Article (CrossRef Link).

[7] R. Houdaille and S. Gouache, "Shaping http adaptive streams for a better user experience," in *Proc. of the 3rd Multimedia Systems Conference*, pp. 1-9, 2012. Article (CrossRef Link).

[8] Te-Yuan Huang, Ramesh Johari, and Nick McKeown, "Downton abbey without the hiccups: Buffer-based rate adaptation for http video streaming," in *Proc. of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, pp. 9-14. ACM, 2013. Article (CrossRef Link).

[9] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen, "What Happens When HTTP Adaptive Streaming Players Compete for Bandwidth?," in *Proc. of NOSSDAV*, pp. 9-14, 2012. Article (CrossRef Link).

[10] T. Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *Proc. of the 2012 ACM conference on Internet measurement conference*, pp. 225-238, 2012. Article (CrossRef Link).

[11] Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K. Sinha, "Can Accurate Predictions Improve Video Streaming in Cellular Networks?" *HotMobile*, pp. 57-62, 2015. Article (CrossRef Link).

[12] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson, "Using the Buffer to Avoid Rebuffers: Evidence from a Large Video Streaming Service," *arXiv preprint arXiv:1401.2209*, 2014. Article (CrossRef Link).

[13] Florian Wamser, David Hock, Michael Seufert, Barbara Staehle, Rastin Pries, and Phuoc Tran-Gia, "Using buffered playtime for QoE-oriented resource management of YouTube video streaming," *Transactions on Emerging Telecommunications Technologies*, vol. 24, no. 3, pp. 288-302, 2013. Article (CrossRef Link).

[14] Jiwoo Park and Kwangsue Chung, "Rate adaptation scheme for HTTP-based streaming to achieve fairness with competing TCP traffic," in *Proc. of Information Networking (ICOIN), 2015 International Conference on*, pp. 222-226. IEEE, 2015. Article (CrossRef Link).

[15] Junchen Jiang, Vyas Sekar, and Hui Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proc. of the 8th international conference on Emerging networking experiments and technologies*, pp. 97-108. ACM, 2012. Article (CrossRef Link).
[16] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C. Begen, and David Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *Selected Areas in Communications, IEEE Journal on*, vol. 32, no. 4, pp. 719-733, 2014. Article (CrossRef Link).
[17] Giuseppe Cofano, Luca De Cicco, and Saverio Mascolo, "A control architecture for massive adaptive video streaming delivery," in *Proc. of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, pp. 7-12. ACM, 2014. Article (CrossRef Link).
[18] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C. Begen, "Server-based traffic shaping for stabilizing oscillating adaptive streaming players," in *Proc. of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 19-24, 2013. Article (CrossRef Link).
[19] Xinying Liu and Aidong Men, "QoE-aware Traffic Shaping for HTTP Adaptive Streaming," *International Journal of Multimedia & Ubiquitous Engineering*, vol. 9, no. 2, pp. 96-115 2014. Article (CrossRef Link).
[20] Kozo Satoda, Hiroshi Yoshida, Hironori Ito, and Kazunori Ozawa, "Adaptive video pacing method based on the prediction of stochastic TCP throughput," in *Proc. of Global Communications Conference (GLOBECOM), 2012 IEEE*, pp. 1944-1950. IEEE, 2012. Article (CrossRef Link).
[21] Yuan-Tse Yu and Sheau-Ru Tong, "Adaptive Transmission Control Protocol-trunking flow control mechanism for supporting proxy-assisted video on demand system," *International Journal of Communication Systems*, vol. 25, no. 10, pp. 1363-1380, 2012. Article (CrossRef Link).
[22] Jenq-Shiou Leu and Sheng-Fu Chen, "TRASS: A transmission rate-adapted streaming server in a wireless environment," *International Journal of Communication Systems*, vol. 24, no. 7, pp. 852-871, 2011. Article (CrossRef Link).
[23] Niels Bouten, Ricardo de O. Schmidt, Jeroen Famaey, Steven Latre, Aiko Pras, and Filip De Turck, "QoE-Driven In-Network Optimization for Adaptive Video Streaming Based on Packet Sampling Measurements," *Computer Networks*, vol. 81, no. C, pp. 96-115, 2015. Article (CrossRef Link).
[24] K. J. Ma, R. Bartos, and S. Bhatia, "Scalability of HTTP pacing with intelligent bursting," in *Proc. of Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, pp. 798-801, 2009. Article (CrossRef Link).
[25] M. Handley, J. Padhye, and S. Floyd, "RFC 2861. TCP congestion window validation," 2000. Article (CrossRef Link).
[26] Christopher Mueller and Christian Timmerer, "A VLC media player plugin enabling dynamic adaptive streaming over HTTP," in *Proc. of the 19th ACM international conference on Multimedia*, pp. 723-726, 2011. Article (CrossRef Link).
[27] S. Hemminger, "Network emulation with NetEm," in *Linux Conf Au*, pp. 18-23, 2005. Article (CrossRef Link).
[28] Stefan Lederer, Christopher Mueller, and Christian Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *Proc. of the 3rd Multimedia Systems Conference*, pp. 89-94, 2012. Article (CrossRef Link).
[29] Michael Zink, Oliver Kuenzel, Jens Schmitt, and Ralf Steinmetz, "Subjective impression of variations in layer encoded videos," in *Quality of Service IWQoS 2003*, pp. 137-154. Springer, 2003. Article (CrossRef Link).
[30] P. Ni, A. Eichhorn, C. Griwodz, and P. Halvorsen, "Fine-grained scalable streaming from coarse-grained videos," in *Proc. of the 18th international workshop on Network and operating systems support for digital audio and video*, pp.103-108. ACM, 2009. Article (CrossRef Link).
[31] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," *arXiv preprint cs/9809099*, 1998. Article (CrossRef Link).

Arkadiusz Biernacki received his B.Eng. degree in Mathematics and the M.Sc. and Ph.D degrees in Computer Science from the Silesian University of Technology, Poland, in 2000, 2002 and 2007 respectively. From 2007 he is an Assistant Professor at the Silesian University of Technology in Poland. His research interests focus on network traffic modelling, computer system simulations and multimedia performance.