DOI: 10.15439/2015F91
ACSIS, Vol. 6

Analysis of video delay in Internet TV service over adaptive HTTP streaming

Marek Dąbrowski, Robert Kołodyński, Wojciech Zieliński
Orange Polska, Centrum Badawczo-Rozwojowe,
ul. Obrzeżna 7, 02-691 Warszawa
Email: marek.dabrowski@orange.com,
Email: robert.kolodynski@orange.com,
Email: wojciech.zielinski@orange.com

Abstract—**The paper deals with OTT TV service based on adaptive HTTP streaming to deliver video content to various devices over unmanaged, best-effort IP network. One of major drawbacks of adaptive streaming technology is a significant video latency comparing to traditional TV broadcast. In this paper, causes of video latency in Internet TV architecture are identified and quantified by theoretical analysis of protocol behaviour and by testbed measurements.**

I. Introduction

VIDEO entertainment over the Internet has become a very popular service all over the world. Customers benefit from services offered by multitude of providers, which include pure OTT (Over The Top) players: local providers or worldwide giants like Netflix or Amazon, customer equipment manufacturers (Apple), content providers and TV stations (HBO, BBC), as well as legacy network and cable operators (Orange, Telefonica, Comcast). A VOD (Video on Demand) service is mainly offered by OTT service providers, but live TV is also gaining popularity, especially for watching live sports in the case of globally popular events. Television audience during events like 2014 Brazil Football World Cup is reaching its peaks all over the world and important part of this audience is watching on their PCs, smartphones and tablets [2].

Most commercially offered TV services over the Internet use so-called adaptive HTTP streaming technology [4]. It assumes that player is able to adapt to temporary network conditions by choosing among several profiles (versions of a stream encoded with certain bitrate), available on the server. The continuous stream is divided into fragments of certain size ("chunks") and delivered to clients using standard HTTP protocol. The format of delivered video fragments and manifest file (an index which allows clients to reach specific stream version) is governed by a streaming protocol, among which the most popular ones are: Microsoft SmoothStreaming, Apple HTTP Live Streaming, MPEG-DASH [4].

A. Video latency in OTT TV

Live OTT TV service may suffer from significant video latency. A continuous video stream is divided into chunks (files), which suggests that certain amount of buffering must be applied in the streaming server. In addition, buffering is required in the end-device to circumvent network jitter and server overloads. As a consequence, end-to-end delay experienced by user is much larger than in the case of traditional broadcast, DTT, cable or IPTV service. Normally, a constant and stable delay is not a problem for viewers of movies or other non-live programs. However, the problem intensifies when someone is watching for example a football match, and may surprisingly hear his neighbors cheering over a scored goal, which he will only see on his screen in next minute, due to the delay introduced by OTT streaming. This problem may become more and more important with the advent of Social TV phenomenon. You may not really hear your neighbor shouting over the goal, but you will immediately see the comments posted by other viewers on Twitter, or you will see the news notification on your mobile phone, before you actually see the goal scored on the screen of your tablet. The issue of end-to-end (e2e) delay is thus becoming an important factor for overall QoE (Quality of Experience) of OTT services [3].

The discussed effect of e2e delay in OTT live TV is illustrated in Fig. 1, which depicts a photo taken while watching live transmission of a football match. The photo shows two screens at the same time: big TV screen connected to cable TV (DVB-C), and laptop screen, displaying an OTT TV service. One can see that match time shown on the laptop (OTT TV) is about 1 minute behind what is presented on the cable TV. Viewers of OTT TV are clearly experiencing a disadvantageous situation, not being able to follow the match truly live.

Work carried out within EUREKA CELTIC project NOTTS (Next generation Over-The-Top multimedia Services) [1]

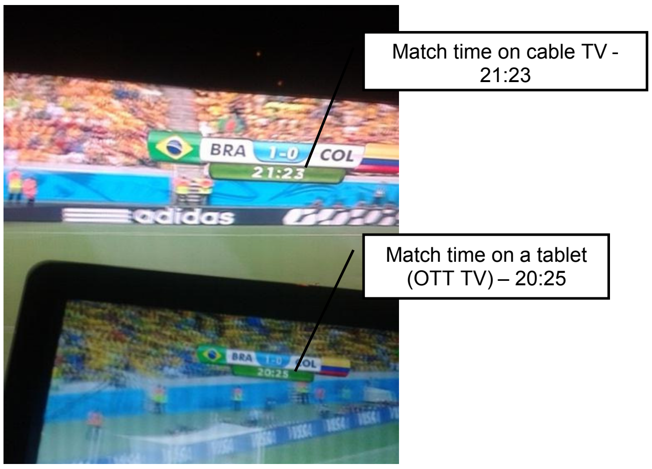
Match time on cable TV - 21:23
Match time on a tablet (OTT TV) – 20:25

Fig. 1. Illustration of video latency problem in OTT Live TV

This study aims to identify, quantify and explain the causes of delay experienced by end user of OTT TV. The analysis will be supported by measurements in a testbed reproducing operational service architecture of OTT TV and video service offered by Orange Polska.

B. *Delay budget in end to end delivery chain*

Fig. 2 depicts typical architecture of OTT content delivery system and identifies major components which may contribute to e2e delay experienced by user.

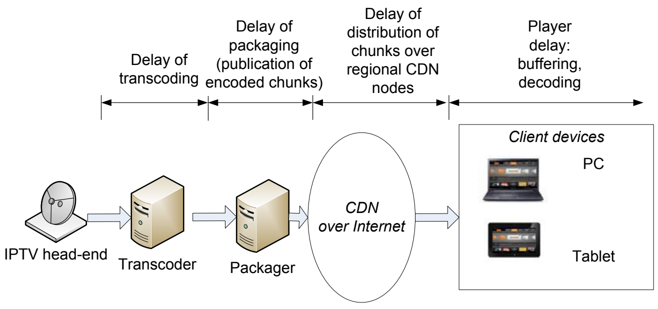
Delay of transcoding
Delay of packaging (publication of encoded chunks)
Delay of distribution of chunks over regional CDN nodes
Player delay: buffering, decoding
Client devices
PC
Tablet
IPTV head-end
Transcoder
Packager
CDN over Internet

Fig. 2. Delay components in live OTT video

- **IPTV head-end**. Input content for OTT delivery chain is obtained from IPTV or satellite TV headend.
- **Transcoder** applies video compression, using several profiles appropriate for transmission over the Internet. H.264 is currently most popular compression standard, with HEVC (H.265) considered as future candidate.
- **Packager** applies streaming format (MS Smoothstreaming, MPEG-DASH, HLS,…). It divides continues stream to chunks of fixed size, prepares the manifest and publishes files on HTTP server.
- **CDN** (Content Delivery Network) is used for stream delivery to regional nodes in a wide area network. Since HTTP standard is used for message delivery, a typical Internet CDN is capable for supporting video streaming [6].
- **Video player** on the client device performs buffering, decoding and video playout. Length of receiving buffer, which is a major source of e2e delay, is a result of compromise between short e2e latency (small buffer), or better resilience against packet-level jitter and losses that may occur in the transport network (long buffer).

II. Adaptive streaming characteristics

In this section, essential characteristics of adaptive streaming technology will be analyzed from the point of view of impact on e2e video delay.

A. *Transcoder behavior*

The transcoder takes as input a continuous video stream, decodes it and encodes again, producing video fragments suitable for further processing by the packager. The encoding standard used in tested scenarios is H.264, the same as the input stream. The format of output file is fmp4, containing the amount of video equal to the packager's chunk duration. Remark that the encoder and packager use the same configuration of chunk duration, and are thus not totally independent in their operation.

Illustrative explanation of encoder impact on video delay is presented in Fig. 3. After the end of time period corresponding to chunk duration, the input video frames are stored in encoder's buffer. Next, they are processed by the encoder and the encoded chunk is saved on the encoder's storage disk as fmp4 file. The time of processing a video chunk by the encoder is non-negligible, and thus the video chunk that is saved on the disk at the output is delayed comparing to the input stream by the value of D_{enc}.

Remark that configuration of encoding profile may impact on the value of this delay, as better quality profiles surely require more processing at the encoder.

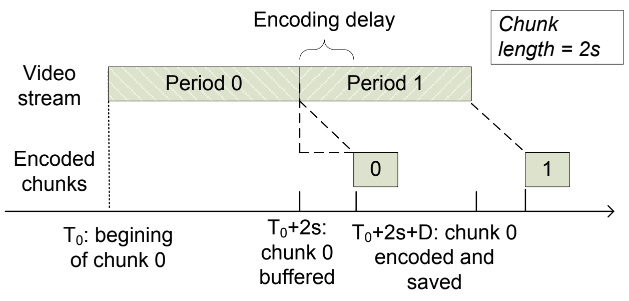

Fig. 3. Illustration of encoder behavior

B. *Packager behavior*

The following two parameters are crucial for operation of packager (see Fig. 4):

- **Chunk (fragment) length:** amount of video (expressed in time units) that is encoded and packaged in a single HTTP message transmitted over the network. The default value in Microsoft SmoothStreaming is 2s.
- Number of **lookahead** fragments: succeeding fragments that have to be collected by the packager before releasing a given chunk. The default value is 2.

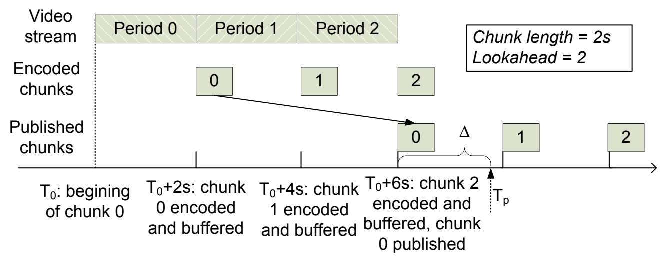
Video stream
Period 0
Period 1
Period 2
Chunk length = 2s
Lookahead = 2
Encoded chunks
Published chunks
Δ
Tp
T0: begining of chunk 0
T0+2s: chunk 0 encoded and buffered
T0+4s: chunk 1 encoded and buffered
T0+6s: chunk 2 encoded and buffered, chunk 0 published

Fig. 4. Illustration of packager behavior

For the purpose of example, let us assume that the *chunk length* is 2s. The upper timeline in Fig. 4 shows a continuous video stream that is being served to the encoder. At the end of each period of 2s, the encoder produces a chunk (packet with encoded portion of the video). Thus, the chunk numbered 0, containing video period starting at *T0* and lasting 2s, is produced at time *T0+2s* and at the same moment it is stored by the packager in its internal buffer for further processing. However, since the *lookahead* parameter is set to 2, the packager will wait for next 2 consecutive chunks, because some information about these chunks must be built-in the header of chunk 0. Since chunk number 2 is available at time *T0+6s*, only then the chunk number 0 may be published and made available for clients.

Remark that player may request live stream at an arbitrary moment T_p (see Fig. 4). The first (newest) chunk available at this random moment T_p, is the chunk number 0, which is already aged *3*chunk length*, plus the duration of Δ, which is random. We may suppose that Δ is uniformly distributed between 0 and *chunk length*, with average value *chunk length/2*.

Thus, on average, the packager introduces delay equal to (l is the *lookahead*, and t_f is the *chunk length*):

$$D_{pack} = (l+1)\times t_f + \frac{t_f}{2} \tag{1}$$

C. Player behavior

Video player on the end-device is a major delay contributor in e2e delay budget. Microsoft SmoothStreaming introduces the following three parameters which have significant impact on behavior of the player when it starts receiving a live video stream:

- **Buffer**: size of receiver buffer (number of seconds of stored video). Default value is 5s.
- **Backoff**: when the player requests a live stream, it actually does not reach for the recent (current) video chunk, but rather for content that is delayed by a sum of backoff and offset parameters. Default value is 6s.
- **Offset**: together with backoff time, the value of this parameter determines playback delay in relation to actual "live" position. Default value is 7s.

When player requests to receive a live video stream, it downloads first a manifest file, which describes technical parameters necessary for the player to decode the stream and advertises the chunks that are available for download on the server. The timestamp of the latest (newest) chunk available within the manifest window is t_0.

However, the player does not normally reach for the chunk t_0. First, it goes back in time by the value of *backoff* plus *offset*. The sum of *backoff* and *offset* determines the timestamp of a chunk, from which the player starts downloading video fragments to fill its buffer (t_{start}). Now, the player immediately requests for next chunks, until it fills its buffer or reaches the limit determined by the offset value (player may not reach for chunks newer than "t_0 – *backoff*". We should now distinguish two situations: buffer ≤ offset, buffer > offset.

Player behavior when buffer is smaller or equal to offset

The player immediately requests for sufficient number of chunks to fill entire buffer. It gets them as fast as network bandwidth can support. Now it is ready to start video playback, beginning with the oldest chunk stored in the buffer. The timestamp of first chunk that will be displayed by player is:

$$t_{start} = t_0 - backoff - offset \tag{2}$$

The video delay as seen by the user will thus be t_0-t_{start}, that is:

$$D_{play1} = backoff + offset \tag{3}$$

Remark that chunk t_0 does not really contain "live" position of video stream, due to delay introduced previously by operation of encoder and packager.

Described behavior of the player is illustrated below in Fig. 6, which depicts chunks that are advertised when the player joins a live stream. The advertised window length is equal to 60s, which corresponds to 29 chunks of length 2s. The player parameters assumed for the purpose of example are: *buffer*=6s, *backoff*=6s, *offset*=20s.

The first (newest) chunk reached by the player has timestamp equal to t_0 – *26s*. Since the buffer size is smaller than the offset, all the buffer may be filled immediately by retrieving 3 chunks (6s) without waiting for any new chunks to be produced by the server. After retrieving enough chunks to fill the buffer to required length, the player starts playback, starting with the chunk t_{start}.

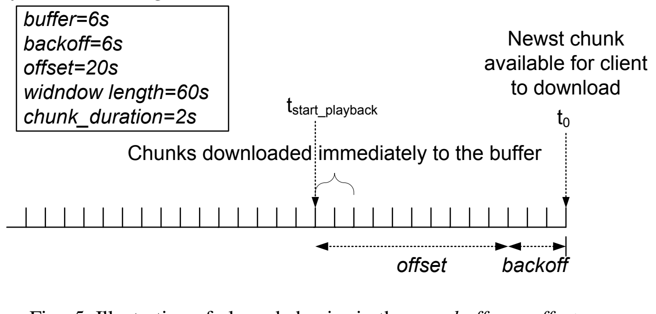

Fig. 5. Illustration of player behavior in the case *buffer* < *offset*

Player behavior when buffer is greater than offset

In the case when buffer > offset, the buffer cannot be immediately filled because the player is not allowed to fetch chunks that are newer than t_0 - *backoff*. So, it immediately (as fast as the network bandwidth can support) fetches the amount of video chunks corresponding to the duration of an offset, and then waits for new chunks to arrive, to fill the remaining part of the buffer. After the time *(buffer – offset)* the buffer is filled and the player may start video playback, beginning from the chunk with timestamp t_{start}. But t_{start} is now additionally delayed from t_0 by *(buffer – offset)* because player had to wait that time to fill the buffer. So:

$$\begin{aligned} t_{start} &= t_0 - backoff - offset - (buffer - offset) \\ &= t_0 - backoff - buffer \end{aligned} \tag{4}$$

The video delay is equal to $t_0 - t_{start}$, that is:

$$D_{play2} = backoff + buffer \tag{5}$$

Described behavior of the player is illustrated below in Fig. 7. The advertised window length is equal to 60s, which corresponds to 29 chunks of length 2s. The example player parameters are the following: *buffer*=20s, *backoff*=6s, *offset*=7s.

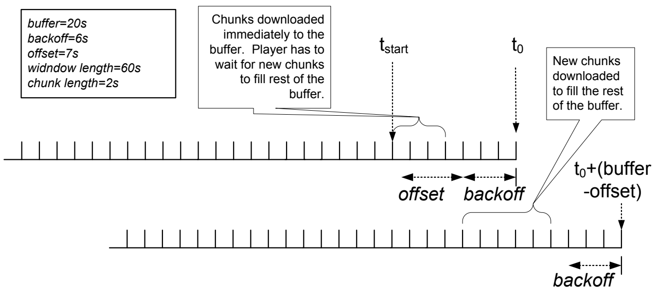
buffer=20s
backoff=6s
offset=7s
widndow length=60s
chunk length=2s
Chunks downloaded immediately to the buffer. Player has to wait for new chunks to fill rest of the buffer.
tstart
t0
New chunks downloaded to fill the rest of the buffer.
offset
backoff
t0+(buffer -offset)
backoff

Fig. 6. Illustration of player behavior in the case *buffer* > *offset*

The first (newest) chunk reached by the player it the one with timestamp $t_{start} = t_0 - backoff - offset$. Since the buffer length is greater than the offset, all the buffer may not be filled immediately. The player thus retrieves rapidly (as fast as bandwidth may support) only "offset" portion of video chunks, and waits (*buffer-offset*) to gather enough newly arrived chunks to fill the rest of the buffer. Then, the player starts playback, starting with the chunk t_{start}.

Summarizing and merging equations (3) and (5) corresponding to different cases of player parameter settings, the formula for player delay can be written as:

$$D_{play} = backoff + \max(buffer, offset) \tag{6}$$

Remark that since packets sent over the network may be delayed or lost, causing a retransmission, the delay calculations should be treated as being "at least" values and the actual delay experienced may be greater than that.

Playback startup delay

We may expect that playback startup delay (between moment when user clicks on "play" button and moment when content actually starts playing) should grow with the size of the player buffer length. This is quite understandable because while joining the live stream the player must wait until the buffer is sufficiently filled, according to its configured value. More precisely, if the player buffer size is configured smaller that the value of offset, the player immediately asks for video chunks to fill the buffer completely. The chunks are thus downloaded almost instantly and the time player waits for filling the buffer is practically not observable. On the other hand, if the buffer size configured in the player is greater that the offset, the player cannot retrieve immediately the number of chunks required to fill the buffer. Thus it has to wait until sufficient number of new chunks appear on the origin server. The time it has to wait is equal to buffer minus offset (amount of video time that is missing in the current window stored on the origin server):

$$D_{play\_start} = \max(0, buffer - offset) \tag{7}$$

III. Experimental setup

A. Testbed architecture

A series of experiments have been performed for confirmation of protocol analysis from section II. Measurements have been carried out in a testbed, which reflects architecture of commercial OTT TV service of Orange Polska. Remark that names of equipment elements are only provided for information of the reader. It is not a goal of experiments described in this paper to evaluate particular vendor solutions. The characteristics that have been studied and measured are intrinsically related with generic technology (HTTP adaptive streaming) and only to a lesser extent depend on particular vendor implementation.

- Descrambler: Cisco DCM. It outputs a single decrypted TV channel in the form of IP multicast Transport Stream (TS), for further preparation of adaptive streaming content.
- Encoder: Ffmpeg v2.2 transcodes the content into H.264 stream packaged in fMP4 (fragmented MP4) format.
- Origin Server: Unified Streaming Platform (USP) v1.5.7 packages fMP4 content into the SmoothStreaming file format and produces manifest file. The content and manifest is served to clients by Apache HTTP server.
- CDN: Akamai Verivue.
- PC player: a web-based player developed in MS Silverlight.
- Mobile player: a reference application provided by the vendor of streaming player software.

B. End-to-end delay measurement

The instrumentation used for measurements of delay in OTT testbed is presented in Fig.8. The configuration of encoder machine allows us for adding current timestamp as an overlay, visually "burned" in video picture. This entry-point timestamp (measurement point A) can be visually compared with the current time on the user device (measurement point B), after passing entire delivery and decoding process. Both clocks (in measurement point A and B) are synchronized with central clock by NTP protocol.

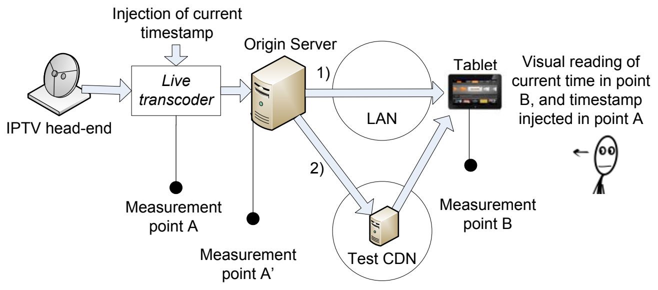

Fig. 7. End-to-end measurement testbed setup

The tesbed allows us for performing measurements including, or not, the impact of CDN. In the first case (path 1 on Fig.8) the end device retrieves content directly from the Origin Server, through a LAN laboratory network. The impact of network latency can thus be considered as negligible and CDN is totally eliminated. In the second case (path 2) the player reaches content through test CDN, consisting of a single cache node.

The test executor launches video player on a tablet connected to the test network and manually (visually) reads the measurement results.

Current timestamp in point A (T_A) is embedded in each video frame. Simultaneous readout of this timestamp and current timestamp (absolute time) in measurement point B (T_B) lets us estimate total time of processing given video frame in entire content delivery chain. The e2e delay can be calculated in any given moment as:

$$D_{e2e} = T_B - T_A \tag{8}$$

Accuracy of assumed measurement method is limited due to visual readout of timestamps. Normally, human tester may read the timestamp from computer and tablet screen with granularity of around 1 second. More fine-grained measurement of time would require some automation of the method and more precise instrumentation. For limiting the impact of human error, each measurement has been repeated several times. Taking into account that typically e2e delay in OTT delivery chain may be in the order of 20 sec – 1min, the granularity of assumed method seems to be sufficient.

Remark that presented method actually measures e2e delay, which is a sum of several delay components:

$$D_{e2e} = D_{enc} + D_{pack} + D_{CDN} + D_{play} \tag{9}$$

Additional actions must be taken to split it into particular components, as explained below.

C. Encoder delay measurement

Factors which impact on delay introduced by encoding process include: encoder implementation efficiency, performance of hardware on which it is being run, whether the encoder itself is software or hardware based, numerous parameters that can be set on the encoder and may alter its performance.

The transcoder installed in the testbed and used in the scope of this study is a software-based solution ffmpeg 2.2, running on Centos 6.5 64 bit system, installed as virtual machine (Oracle VM VirtualBox, 1GB RAM, 2 CPU). The virtual machine was running on Windows Server 2008 R2 Standard 64-bit (HP ML150: 2xIntel Xeon CPUE5504 2GHz, 4GB RAM).

Fig.9 gives more details into the configuration of transcoder, presenting video processing steps and the detailed points where timestamp was embedded into the video for purpose of measuring delays.

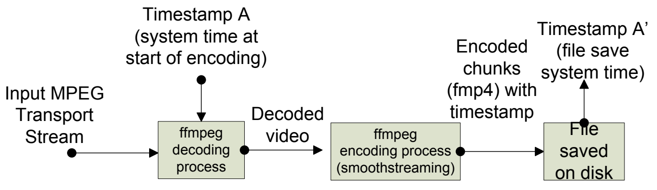
Timestamp A (system time at start of encoding)
Input MPEG Transport Stream
ffmpeg decoding process
Decoded video
ffmpeg encoding process (smoothstreaming)
Encoded chunks (fmp4) with timestamp
Timestamp A' (file save system time)
File saved on disk

Fig. 8 Encoder configuration (with timestamp embedding) for measurements of encoding delay

As the first step within the transcoder module, FFmpeg decodes the input stream to produce raw video, which is then encoded to smoothstreaming compatible format by the encoder. However, prior to encoding, ffmpeg process “burns” a timestamp in each produced video frame. The timestamp value in point A (T_A) corresponds to current system time when given frame has entered the transcoding process.

The output of the encoder is an fmp4 file, containing amount of video corresponding to a duration of a chunk. Remark that although the encoding and packaging processes are logically separated, the encoder is not totally independent of the packager as it prepares an encoded portion of the video which suits the packager’s chunk size.

The encoded chunks (in fmp4 format) are then saved to the storage of transcoder machine. The time of file modification is recorded in the file system as a normal operation of the computer’s operating system and it is considered as a timestamp in point A’ ($T_{A'}$). By comparing timestamp A’ of a chunk, with timestamp A of the last video frame of each chunk, we can estimate the delay introduced by the whole transcoding process.

$$D_{enc} = T_{A'} - T_A \tag{10}$$

D. Packager delay measurement

In order to evaluate impact of packager in the e2e delay budget, we have performed a set of measurements using the same methodology as described for the e2e delay, but with a specific setting of player parameters. By setting *buffer=1*, *offset=0* and *backoff=0* we reduce the impact of player practically to zero. Without backoff and offset the player reaches for the newest available chunk while joining the live stream (see Fig.10). Since this single chunk is sufficient to fill the buffer, the player may start playback immediately after receiving it. In a real network situation such configuration is not recommended since it is very susceptible to network impairments. However in the "idealized" testbed environment we were able to properly play a live stream with such non-realistic parameter setting.

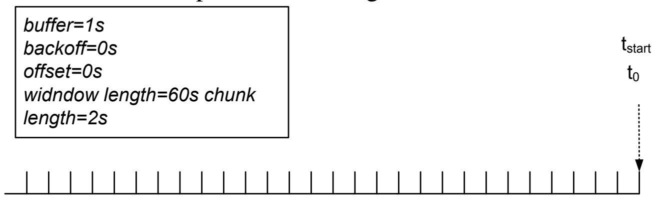

Fig. 9. Illustration of player behavior in the case buffer=1, offset=0, backoff=0

Since player starts playback immediately after receiving the newest chunk from the origin server, we may expect that observed delay in measurement point B is only related with the packager and encoding delay ($D_{play}=0$). So, the assumed procedure was to measure delay in e2e relation (without CDN: $D_{CDN}=0$) and substract from it known value of encoding delay, obtained by previous measurement of D_{enc} in the same setup.

$$D_{pack} = D_{e2e} - D_{enc} \tag{11}$$

E. CDN delay measurement

As depicted in Fig.8, testbed configuration allows for performing measurements with, or without CDN in the delivery chain. The assumed indirect methodology for evaluating impact of CDN itself assumes comparing the end-to-end delay results measured "with" and "without" CDN.

$$D_{CDN} = D_{e2ewithCDN} - D_{e2ewithoutCDN} \tag{12}$$

F. Player delay measurement

The methodology of evaluating impact of the player alone assumes performing measurements in end-to-end mode (see section III.B), without CDN ($D_{CDN}=0$) and substracting from result the values of delay of encoder and packager, known from prior measurements in the same setup. Thus,

$$D_{play} = D_{e2e} - D_{enc} - D_{pack} \tag{13}$$

IV. MEASUREMENT RESULTS

A. Encoder delay impact

The encoder delay has been measured according to the methodology described in section III.C, with chunk length changed from 1s to 10s (remark that although chunk length is a parameter of packager, the encoder must be configured accordingly in order to produce encoded video fragments that are suitable for the packager).

In addition, several encoding profiles have been tested (baseline, main), with several settings of ffmepg tool encoding parameters (medium, fast, ultrafast). The results of experiments are presented in Table I. Reported measured delay is an average calculated over five repetitions of each experiment.

TABLE I.
MEASURED ENCODER DELAY

Encoder profile	**Chunk size**	**Avg measured delay [s]**
Baseline, fast	1	1.49
	2	1.74
	5	1.78
	7	1.76
	10	1.79
Baseline, medium	1	1.54
	10	2.11
Main, fast	1	1.73
	10	1.94
Main, ultrafast	1	1.36
	10	1.19

The encoder delay in testbed environment is roughly between 1.5 and 2 seconds. We recognize that obtained results could differ for another encoder type, running in different environment. Therefore, we stress that the results are relevant for particular hardware/software configuration of our testbed and cannot be generalized in straight forward way to other types of encoders available on the market.

B. Packaging delay impact

The packager delay has been measured as described in section III.D, with various chunk length set on the packager (1 to 10s), and with different values of lookahead parameter (1, 2, 4, 6 fragments).

The results are presented in Table II and Fig.11 (subset of results with *lookahead*=2). The measured delay is compared with theoretical value D_{pack} from equation (1).

TABLE II.
MEASURED PACKAGER DELAY

Chunk length [s]	**Lookahead**	**Measured packager delay [s]**	**Theoretical value of D_{pack} (eq.1) [s]**
1	2	5.51	3.5
2	2	7.86	7
5	2	18.22	17.5
7	2	25.44	24.5
10	2	34.61	35
2	1	6.46	5
5	1	13.02	12.5
7	1	19.64	17.5
10	1	24.61	25
2	4	12.66	11
10	4	57.41	55
5	4	28.82	27.5
2	6	16.26	15

One can observe that measured packager delay can be quite well approximated by formula (1).

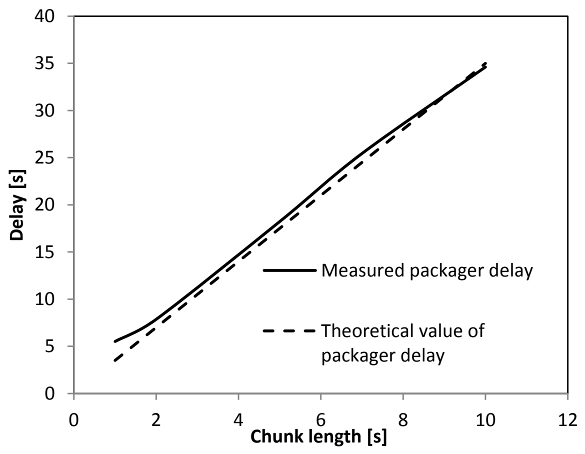

Fig. 10 Measured packager delay, with lookahead=2

C. CDN delay impact

The delay impact of CDN has been assessed using methodology described in section III.E. Measurements were done with different value of chunk size on packager (1, 2, 5, 7, 10s) and with fixed value of *lookahead* = 2. The results of measurements, averaged over 5 repetitions of each experiment, are presented in Table III. The average delay impact of CDN is indirectly estimated as difference of delay measured with, and without CDN in the testbed.

TABLE III.
MEASURED CDN DELAY

Chunk length [s]	**Average delay with CDN [s]**	**Average delay w/out CDN [s]**	**Average delay of CDN [s]**
1	6	6.6	**0.6**
2	8.8	9	**0.2**
5	14.4	15.8	**1.4**
7	18.6	22	**3.4**
10	26.2	30.8	**4.6**

We can observe that CDN does not introduce significant delay in the end-to-end chain. The observed delay is between 0.2 and 4.6s, depending on the chunk length.

D. Video player delay impact

Delay has been measured in end-to-end mode, without a CDN (see section III.F). The delay of encoder and packager has been eliminated by subtracting the results of previous measurements from section IV.A and IV.B, performed with the same parameter setting.

In first series of experiments, the delay was measured with different values of *buffer* length in video player. The values of two other player parameters were fixed to *backoff*=6s, *offset*=7s. Note that the values *buffer*=5s, *backoff*=6s and *offset*=7s are considered as default in the Microsoft Smoothstreaming protocol. Two values are reported as result of experiments (see Table IV and Fig.12):

- *"Start delay"* corresponds to stream startup time. It was measured with a stop watch, as time between clicking "play" on a player, and actual start of video playout. Reported value is an average over 5 repetitions of each experiment.
- *"Player delay"* corresponds to the observed difference between watched video and actual "live" position, measured as described in section III.F. The reported values are an average and minimum over 5 repetitions of each experiment.

TABLE IV.
MEASURED PLAYER DELAY AS FUNCTION OF BUFFER LENGTH

Player parameters			**Startup delay [s]**		**Player delay [s]**		**Theoretical D_{play} (eq.6) [s]**
buffer [s]	**back off [s]**	**off set [s]**	**Avg**	**D_{play_start} (eq.7) [s]**	**Average**	**Min**	
3	6	7	1.26	0	14.06	13.26	13
5	6	7	1.75	0	14.26	13.26	13
7	6	7	2.56	0	15.46	15.26	13
10	6	7	3.67	3	18.26	15.26	16
13	6	7	4.10	6	17.46	16.26	19
15	6	7	6.68	8	19.26	18.26	21
17	6	7	7.73	10	20.26	19.26	23
18	6	7	10.81	11	23.46	23.26	24
20	6	7	11.73	13	25.66	24.26	26
25	6	7	15.59	18	28.46	27.26	31
30	6	7	23.00	23	36.46	36.26	36

One may observe that the player delay can be quite well predicted using formula (6).

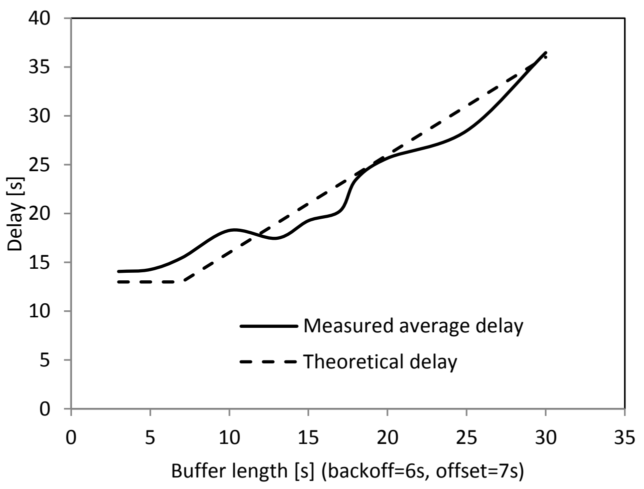

Fig. 11. Measured player delay as function of player buffer length

Fig. 13 shows the measured playback startup delay. We may observe that it is well approximated by formula (7).

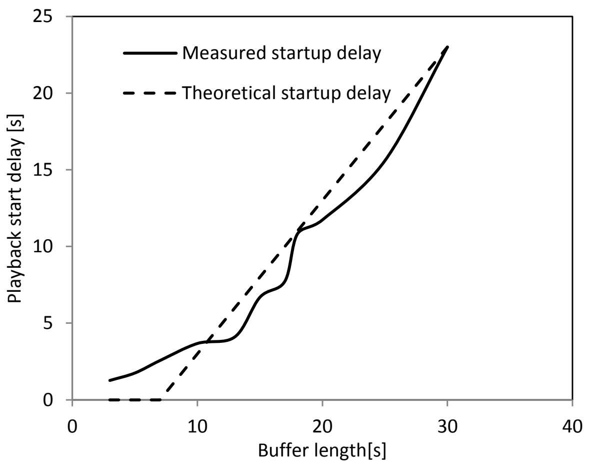

Fig. 12. Measured playback start delay as function of player buffer length

In the second set of experiments, the player *offset* parameter was varied, with fixed *buffer* length equal to 5s, and fixed *backoff* equal to 6s. The results are presented in Fig.14.

Note that the playback startup delay does not significantly depend on value of *offset* parameter, because in this particular case the buffer is usually smaller than the offset (except from the first two measurements).

Once again the results confirm validity of formula (6) for predicting latency of the player. Above the value of *offset*=60s chunks that should be retrieved are out of the range of advertised window, which means that player wants to download chunks that are too old and do not exists anymore on the server. Thus, formula (6) does not apply.

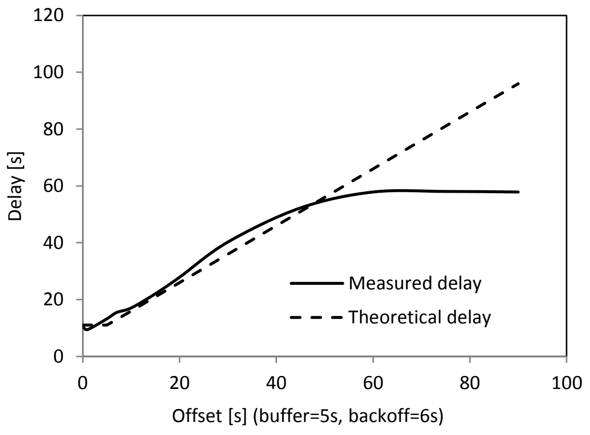

Fig. 13. Measured player delay as function of player buffer offset

In the last set of experiments, the player *backoff* time has been varied in the range from 0 to 90s, with constant *buffer*=5s and *offset*=7s. The results are presented in Fig.15.

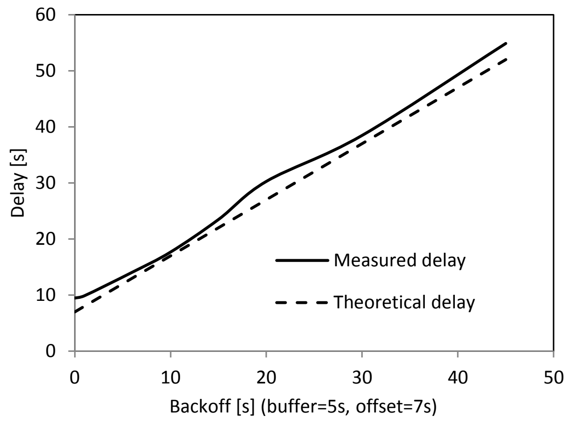

Fig. 14. Measured player delay as function of player buffer backoff

Once again, the results confirm that the delay introduced by the player can be estimated by equation (6).

V. CONCLUSIONS

The paper has presented results of analysis and measurements of user-perceived delay in Live TV service delivered over the Internet using adaptive HTTP streaming technique. The following elements of content delivery architecture have been identified as major contributors to overall latency: source stream transcoding, packaging (applying adaptive streaming format), delivery over CDN, and buffering on end-device.

Presented results are of analytical as well of experimental type and may have practical importance for video service providers as hints for setting key system parameters, taking into account both technical constraints and user Quality of Experience.

REFERENCES

[1] EUREKA / CELTIC NOTTS: http://projects.celticplus.eu/notts/
[2] Broadband TV news: http://www.broadbandtvnews.com/2014/06/12/world-cup-matches-to-set-new-streaming-records/, last accessed on 23.04.2015
[3] R.Merkuria, P.Cesar, D. Bulterman, “Digital TV: The Effect of Delay when Watching Football”, EuroITV’12, 10th European Conference on Interactive TV and Video, Berlin, July 2012, http://dx.doi.org/10.1145/2325616.2325632
[4] T.Stockhammer, “Dynamic adaptive streaming over HTTP: standards and design principles, ACM MMSys '11, dx.doi.org/10.1145/1943552.1943572
[5] Microsoft SmoothStreaming: http://msdn.microsoft.com/en-us/library/microsoft.web.media.smoothstreaming.smoothstreamingmediaelement.liveplaybackoffset%28v=vs.95%29.aspx , last accessed on 26.06.2015
[6] K.Kaczmarski, M.Pilarski, “Content Delivery Network Monitoring”, Federated Conference on Computer Science and Information Systems (FedCSIS), 2012, pages 633 – 639, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6354443&isnumber=6354297