Inferring Live Streaming Delays in the Wild

Megumi Ninomiya
IIJ Research Lab.

Kenjiro Cho
IIJ Research Lab.

***Abstract*—In live video steaming over HTTP, a client requests a series of video segments created on the fly, using the standard HTTP pull model. Fetching and buffering video segments are under clients' control, which incurs extra playback delay between a real-world live event and its appearance in video playback for the client. Previous studies, on the client side, show that playback delay is in the order of tens of seconds, but they are limited to small-scale measurement.**

In this paper, we present a model of playback delays in live streaming over HTTP by analyzing actual server logs, and propose a practical method to infer playback delays for each client only from Web server logs, in order to investigate playback delays for the entire users. By applying the proposed method to the server logs of a large-scale live streaming event, we have captured the distribution of playback delays which are normally distributed and found that the initial playback delay is the dominant factor in playback delays. From the results, we have identified two major factors of playback delays: segment length and startup buffering on the client side. We found typical buffer sizes used by different client types, and that the vast majority of playback delays falls into the range: mean plus-or-minus two segment-lengths.

I. Introduction

The protocols used for video streaming are shifting from video-specific protocols to generic HTTP-based protocols, because the HTTP-based protocols work over most firewalls and are supported in popular Web browsers. Video streaming service providers can use standard servers and load-balancers designed for general Web services. Thus, video streaming over HTTP is quickly becoming the mainstream for live streaming.

However, the playback delay in live streaming over HTTP is much larger and more variable than dedicated streaming protocols such as RTP over UDP [1]. In live streaming over HTTP, clients request segment files which are created on the fly. A segment file can be created only after the end of its period, and a client could arrive just before a new segment file becomes available. Thus, segmentation, by definition, introduces at least one, but up to two segment long delay. Furthermore, an additional delay is incurred by the network condition as well as client implementations such as segment fetching and buffering strategies. In practice, a client needs to buffer multiple segments before starting playback to absorb variability. As a result, users experience considerably different delays even when watching the same live video.

The presence of playback delay is well known to content providers and streaming service providers. Users are also aware of the time lag from the corresponding TV broadcasting, and sometimes experience news on SNS coming tens of seconds earlier than the corresponding scene on the live streaming.

Moreover, popular live streaming services are often combined with other live feeds such as a live score feed for a sporting event and a live feed from SNS on the event. For such services, the provider needs to insert a certain delay to the feeds, in order to avoid the news on the feeds appearing earlier than the live streaming. The length of the inserted delay is often determined only empirically. However, we could use a statistical method if the distribution of playback delay were available.

Traditional methods for measuring playback delay require installing a measurement tool on the client side. As such, it is difficult to scale up for a large number of clients.

In this paper, we present a model of playback delay in live streaming over HTTP, and propose a method to infer playback delays for each client only from Web server logs.

We have applied this method to real-world Web server access logs collected from the "Summer Koshien" high school baseball tournament, one of Japan's most popular sporting events. We first validate the proposed method by comparing the inferred playback delays against the measurements at sampled clients. Then, we infer the playback delays for the entire clients in the server logs. We found that the initial playback delay is the dominant factor in playback delays, and the segment length and startup buffering on the client side are two major contributing factors. We also found typical buffer sizes used by different client types. From the inferred playback distribution, we show that the vast majority of playback delays falls into the range: mean plus-or-minus 2 segment lengths.

The contributions of the paper are (1) the proposed model of playback delay for live streaming, (2) the proposed technique to infer playback delays only from general server logs, (3) identifying major delay factors contributing to our understanding of live streaming behaviors, and (4) the Koshien results showing practical delay bounds among users.

II. Background

In this section, we provide the basics of content delivery and playback methods in streaming over HTTP, and describe the differences of live streaming and on-demand streaming.

A. HTTP Streaming

In streaming over HTTP, video data is divided into a series of segment files by fixed time length, and placed on Web servers for delivery. When starting video playback, a client first requests a manifest file from the Web server using the HTTP GET method. A manifest file contains a list of URLs for available segment files, and the client requests segment files

in sequence referring to the list. A client initially buffers a few segments, and then, starts playing the video. During playback, a client repeatedly requests subsequent segment files trying to keep the data volume in the buffer. There are different HTTP streaming technologies such as HTTP Dynamic Streaming (HDS) by Adobe, HTTP Live Streaming (HLS) by Apple, Smooth Streaming (MSS) by Microsoft, and MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) standardized by ISO. They are similar in the concept, and our model applies to all of these.

B. Differences between On-Demand and Live Streaming

There are two types of streaming over HTTP: one is on-demand streaming and the other is live streaming. Both types share the basic mechanisms using segmentation and HTTP, but, in live streaming, segment files are created on the fly and the manifest file is updated every time a new segment file is created. Therefore, a client needs to repeatedly request the manifest file and segment files one after another.

Furthermore, the buffering size at a client is different. For on-demand streaming, since all the segment files exist beforehand, clients can have sufficient buffering at the starting time to avoid buffer underflow. On the other hand, in live streaming, the amount of buffering is a trade-off between liveness and the risk of buffer shortage. Larger buffering at the starting time requires starting from an older segment, and thereby impairs the liveness. Hence, clients usually use a smaller buffer for live streaming than for on-demand streaming.

III. Model of Playback Delay

In this section, we propose a model for playback delay in live streaming over HTTP. There are multiple factors which contribute to playback delays. They are roughly divided into two groups: the initial delay at the video starting time and the pause delays caused by buffer underflows during the playback.

It is important to note that when no buffer underflow occurs during the playback, the initial delay is kept for the whole playback period. Thus, the initial delay is the major delay factor when the playback is smooth. On the other hand, if a buffer underflow causes a short pause for the playback, the playback is delayed for this period of time. The pause delays accumulate every time a pause occurs as long as video segments are played in order without skipping any segment. In this model, we consider only consecutive sequences and, when a segment skipping occurs, we count it as another sequence. Thus, it is enough to capture the initial delay and the pause delays.

A. Initial Playback Delay

The initial playback delay is illustrated in Fig. 1. The camera at the top captures video and the video is split into scenes of fixed length l; $Tr(i)$ is the start time of scene i. Then, scene i is encoded into segment file i and uploaded to the cache server along with the updated manifest file at $Tu(i)$. Typically, multiple cache servers are used for load balancing, and we omit other intermediaries such as an encoding server and a content source server in this model. At some point, client k arrives and first downloads the latest manifest file at $Tm(i)$. The latest segment is i in the manifest, but for buffering (2 segments in the example), the client starts downloading from segment $(i-2)$ at $Ts(i-2)$. The client feeds the download segments into the decoder, and finally the scene $(i-2)$ appears on the display at $Tp(i-2)$.

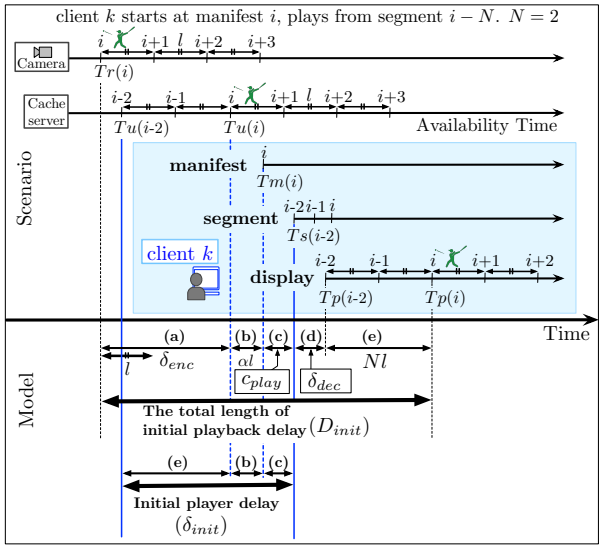

Fig. 1. Scenario and model of initial playback delay

We classify the initial playback delay into 5 factors:

(a) Encoder Delay is the total delay on the encoding side, from the camera input to the upload time of the corresponding segment file on the cache server. The encoder delay includes the segmentation delay of l seconds, inherent to the video segmentation. A segment file can be created only after the corresponding period so that the delay is always larger than l. The other factors include encoding time and file uploading time; video can be encoded into multiple bit-rates for adaptive bit-rate and there could be multiple data transfers between the camera and the cache server. However, the encoder delay does not change much for different segments and is considered a constant.

(b) Arrival Timing Delay is the delay caused by the arrival timing within the segment interval $[0, l)$, which is also inherent to video segmentation. When a client requests the manifest file for the first time, this timing falls within the segment interval $[0, l)$ so that the latest manifest file is up to l seconds old from its updated time on the cache server.

(c) Segment Request Delay is the time period between requesting a manifest file and requesting the corresponding segment. The segment request delay includes the manifest processing time and the segment transfer time.

(d) Decoder Delay is the total delay on the decoder, after the player's placing the video data into the decoder input buffer until the video appears on the display. The decoder delay also does not change much for different segments so it is considered a constant in our model.

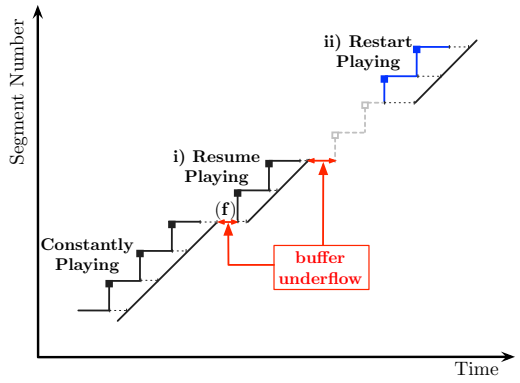

Fig. 2. Two types of pause recovery from buffer underflows

(e) Backtracking Delay is the delay caused by a player to play back from older segments. For initial buffering, a player needs to start from an older segment. For example, if the initial buffering is 2 segments and the latest segment is i, a player starts playing from $(i-2)$, adding $2l$ seconds to the delay.

B. Pause Delays by Buffer Underflows

When a client fails to download next segment in time, the decoder buffer underflows, resulting in a pause in video playback. There are two types of pause recovery, with or without segment skipping, as illustrated in Fig. 2. If the client is able to download the next segment only slightly behind time, the client continues to play it in order without skipping any segment. In this case, the pause incurs extra delay for playback, and such delays are cumulative. On the other hand, if it is far behind, the client gives up continuous play, and starts over by downloading the latest manifest file. Some segments are skipped as a result of the restart. In terms of playback delay, this case is identical to a new arrival so that a restart can be treated as a separate sequence. Thus, for investigating playback delay, we consider only consecutive sequences and extract pause delays in a sequence.

C. Model of Playback Delay

Based on the classified factors, we define a simple model for the playback delay for client k. The delay for playing segment i consists of the encoder delay (a), δ_{enc}, the player delay (b)(c)(e) and pause delays by buffer underflows, $D_{play}(i)$, and the decoder delay (d), δ_{dec}.

$$D(i) \quad = \quad \delta_{enc} + D_{play}(i) + \delta_{dec} \tag{1}$$

The player delay $D_{play}(i)$ can be inferred from server logs, and consists of the initial player delay for the initial segment and the accumulated pause delays:

$$D_{play}(i) = \begin{cases} \delta_{init} & \text{if first segment} \\ \delta_{init} + \sum_i P(i) & \text{otherwise} \end{cases} \tag{2}$$

Here, $P(i)$ is the pause time for segment i and 0 when no pause occurs. We will show how to infer $P(i)$ in the next section.

In the initial player delay, the arrival timing delay (b) is a fraction of l and expressed as αl where α is $[0, 1)$ and $E[\alpha] = 0.5$ for random arrival. The backtracking delay (e) is Nl, where N is the number of backtracked segments for the initial buffering and constant for a player implementation. The segment request delay (c) can be approximated as a constant c_{play}. Then, δ_{init} becomes:

$$\delta_{init} = (\alpha + N)l + c_{play} \tag{3}$$

Note that δ_{init} is equivalent to the time from the upload time to the download time of the first segment $(i-2)$, as shown by the bottom arrow in Fig. 1.

δ_{enc} and δ_{dec} is independent of segments, and approximated as constants. Further, δ_{enc} includes the segmentation delay l and the rest of the encoding delay. Then, δ_{enc} and δ_{dec} become $\delta_{enc} = l + c_{enc}$ and $\delta_{dec} = c_{dec}$. Thus, the initial delay for client k is:

$$\begin{aligned} D_{init} \quad &= \quad \delta_{enc} + \delta_{init} + \delta_{dec} \\ &= \quad (1 + \alpha + N)l + c_{play} + c_{enc} + c_{dec} \end{aligned} \tag{4}$$

From the server logs, we cannot observe c_{enc} and c_{dec}, but it is possible to obtain these values separately, for example, by performing a simple lab test. The rest of the values can be inferred from server logs.

Finally, the playback delay for client k is:

$$D(i) = \begin{cases} D_{init} & \text{if first segment} \\ D_{init} + \sum_i P(i) & \text{otherwise} \end{cases} \tag{5}$$

IV. Inference Methodology

There are limited information in server logs. We can extract only request sequences of segment files for each client. Our inference method is to retrofit segment request sequences into the model presented in the previous section.

A typical server access log entry contains client IP address, timestamp (the time the server finishes processing the request in second), requested file name, status code, and User-Agent. For manifest files and segment files, we can obtain only their downloading times. We can distinguish different segment files by file name, but not manifest files as its file name is always the same. For distinguishing clients, we use the pair of client IP address and User-Agent, and extract only request sequences of consecutive segments.

A. Preprocessing

In the preprocessing, we extract the availability time of each segment file, and consecutive request sequences for each client.

Availability Time: The availability time of a segment file is an approximation of the segment upload time on the server. Because the upload time of a segment file is not available in logs, we use the time for the first request of a given segment file as the availability time, an approximated time of the upload. We will validate this approximation in Sec VI.

Consecutive Request Sequences: To extract consecutive request sequences for each client, we first classify requests by unique pairs of IP address and User-Agent. Then, we extract request sequences of consecutive segments for each client. When there is a hole in a sequence, they are considered as separate sequences. We exclude sequences which are too short for video viewing analysis, empirically using 5 consecutive segments as the minimum view length.

B. Inference Method

For each request sequence, we infer the initial player delay δ_{init} in Equation 3 and the pause delay of each segment $P(i)$ in the request sequence.

Once we have the availability time for each segment, δ_{init} for each request sequence is approximated as the difference between the availability time for the first segment in the sequence and the download time of this segment.

For the accumulated pause delays, we infer the finishing time of playing each segment file, taking buffering into consideration. If the download of a segment file is earlier than the finishing time of the previous segment, the segment is buffered without buffer underflow or pause. On the other hand, if the download time is later than the finishing time of the previous segment, buffer underflow occurs resulting in a pause.

Let $T_d(i)$ be the download time, and $T_f(i)$ be the finishing time for segment i. Assuming some delay c_{buf} between the segment download time and its play's starting time, we can compute the finishing time as follows:

$$T_f(i) = \begin{cases} T_d(i) + l + c_{buf} & \text{if first segment or} \\ & T_d(i) \geq T_f(i-1) \\ T_f(i-1) + l & \text{otherwise} \end{cases} \quad (6)$$

Then, the pause time for segment i is:

$$P(i) = \begin{cases} 0 & \text{if first segment or} \\ & T_d(i) < T_f(i-1) + c_{buf} \\ T_d(i) - T_f(i-1) + c_{buf} & \text{otherwise} \end{cases} \quad (7)$$

In our analysis in Sec VII, we conservatively set $c_{buf} = 0$.

Using these methods, we can infer the playback delay for each client only from Web server logs.

V. Datasets

In this section, we present an overview of the live streaming event and the data collection.

A. Overview of Summer Koshien Live Streaming

The National High School Baseball Championship at Koshien Stadium, commonly known as "Summer Koshien", is the largest amateur sporting event in Japan [2].

The games have been live-streamed over HTTP since 2012 [3], and the number of accesses in 2015 was more than doubled from the previous year [4]. The live streaming was provided by www.asahi.com, which is a major website of Japanese news company in 2015. Table I presents an overview of the live streaming of Summer Koshien in 2015. Over the two weeks of the tournament, there were approximately 4.6 billion total requests and 2.1 million unique IP addresses. During the final game, the peak traffic of 238Gbps was recorded.

TABLE I
Overview of the live streaming of Summer Koshien 2015

period of time	# total requests (billions)	Sent data (TB)	# total clients (millions)	# Unique IPs (millions)
14days (122hours)	4.6	1,456	6.6	2.1

TABLE II
Access ratio by client type

Client Type	PC Browsers	Mobile Apps	Mobile Browsers
# requests	80%	4%	16%
# clients	49%	9%	42%

For content distribution, up to 28 Web servers (nginx) were used for load balancing at IIJ [5], an ISP in Japan. The original content was encoded by Asahi Broadcasting [6] and uploaded to the ingest server at IIJ using Real Time Messaging Protocol (RTMP). Two types of content were generated at the ingest server: HTTP Dynamic Streaming (HDS) for personal computers, and HTTP Live Streaming (HLS) for mobile devices. The segment length is fixed to 8 seconds for all contents, which is conservatively selected for stability rather than for liveness. For both device types, the video was encoded into two bitrates, 450kbps and 750kbps, for adaptive bit-rate streaming.

We classify clients by User-Agent into 3 groups: PC Browsers, Mobile Apps, and Mobile Browsers. The PC Browsers includes IE, Chrome, Safari and Firefox. The Mobile Apps are dedicated mobile applications for the event that were available for iOS and Android. The Mobile Browsers are general browsers on iOS and Android.

Table II compares client types by the number of requests and clients. Clients are identified by client IP address and User-Agent. The PC Browser is 80% in the number of requests, but only 49% in the number of clients, because the average viewing time of PC users is much longer than that of mobile users. Within mobile users, the majority used general browsers rather than the apps.

B. Server Logs

We collected all the access logs for the games from the Web servers used for the content distribution. Table III lists the logged items for each request; the log format is similar to the Apache Combined Log Format. The timestamp is the completion time of the request processing, represented in seconds. Having only one second resolution in the timestamp is a major limitation in our analysis, but it is a common practise for server logs and still provides meaningful results for the segment length of 8 seconds in our dataset.

C. Client Side Data

In order to evaluate the proposed inference method, we need ground truth data of playback delays. To this end, we placed 3 types of players (Mac, iOS and Android) and TV side by side at home, and recorded their displays by a video-camera during

TABLE III
SERVER LOG FORMAT

1	client IP address
2	server IP address
3	timestamp
4	HTTP request
5	response status code
6	content body size
7	referrer
8	User-Agent

TABLE IV
CLIENT SIDE DATA OVERVIEW

OS	Player	Network	Content	# samples
OS X 10.9.5	Firefox 40.0	CATV	HDS	985
iOS 8.1.2	App. (AppleCoreMedia)	LTE	HLS	108
Android 4.4.2	App. (VisualOn_OSMP+)			103

TABLE V
CONSECUTIVE REQUEST SEQUENCES

Client Type	PC Browsers	Mobile Apps	Mobile Browsers
# consecutive request sequences(millions)	18.2	1.1	3.7
# segment file requests(billions)	1.63	0.08	0.28
extracted segments / all segments	91.7%	89.9%	84.8%
mean sequence size in segments	89.6	77.0	77.8
median sequence size in segments	57	31	25

the games. To observe variability in the initial playback delays, we repeated measurement cycles that launches the player, runs it for 2 minutes, terminates it, and waits for one minute. To launch and terminate the player, we used the cron utility on OS X over 12 sampled days, but did it manually on the mobile devices over one sampled day each.

Later, we manually measured the delays in the streaming playback from the recorded video, by identifying specific scenes and then matching the same scenes appearing on the TV and the streaming playback. The obtained delay samples are 985 for Mac, 108 for iOS, and 103 for Android as shown in Table IV. We used a video editing application to find the same scenes manually. However, since we collected easy-to-find scenes, we believe that the granularity of this method is under one second.

Although this dataset is only from a single vantage point and only for 3 clients, we can validate our proposed method using this ground truth dataset as shown in the next section.

VI. MODEL VALIDATION

In this section, we validate our proposed model. First, we validate the preprocessing methods, by examining the availability time and consecutive request sequences extracted from the server logs. Subsequently, we validate the proposed model by comparing inferred playback delays from the server logs against the measured playback delays.

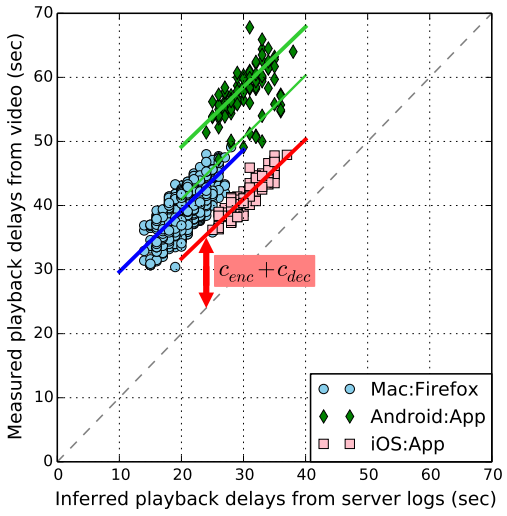

Fig. 3. Validation of playback delay inference

A. Validation of Preprocessed Data

Availability Time: we use the availability time of a segment file as an approximation of the upload time of the segment file. The assumption is that, if there are enough number of requests for each segment file, the earliest request is close to the upload time. The question here is how many requests per segment file is required to use the availability time as approximation.

We validate the availability time by observing the time intervals for consecutive segment files. The upload times of consecutive segment files should be apart by the segment time length l. Accordingly, the intervals of the availability time of consecutive segment files should be l seconds.

We compute mean and standard deviation for intervals of the availability time in consecutive segment files against the number of requests per segment file. We confirmed that both mean and standard deviation converge as the number of requests per segment file increases; the point of convergence is 200 requests per segment file in our dataset. Thus, for delay inference, we used the segment files having more than 200 requests, and consecutive sequences of such segments longer than 450 segments (one hour in time length).

Consecutive Request Sequences: Table V summarizes the extracted consecutive sequences. We have a sufficient number of sequences that cover most of the segments in the logs. The median sequence size is 57 segments (456 seconds) for PC Browsers, 31 segments (248 seconds) for Mobile Apps, and 25 segments (200 seconds) for Mobile Browsers.

B. Validation of Playback Delay Inference

For evaluating the proposed method, we compare the measured delays with the inferred delays derived from the corresponding log entries. In this dataset, we observed only a small number of pause delays so that the inferred delays are mostly about the initial playback delays.

TABLE VI
COMPARISON OF MEASURED VALUE AND INFERRED VALUE

client	slope of regression line	correlation coefficient	difference between measured value and inferred value	
			mean	std.
Mac:Firefox	0.9519	0.97	20.1	2.4
iOS:App	0.9314	0.91	10.9	1.1
Android:App	0.9359	0.83	28.6	1.8

TABLE VII
CALCULATED INITIAL PLAYBACK DELAY

Client Type	PC Browsers	Mobile Apps & Browsers
mean of the initial player delay (sec)	12	21
backtracking delay (sec)	8	17
number of backtracked segment (N)	1	2
calculated initial playback delay from the proposed model (sec)	20	28
inferred initial playback delay from server logs (sec)	20.7	29.4

Fig. 3 compares the inferred playback delay from the server logs on the X-axis with the measured delay from the captured video on the Y-axis. Each client constitutes a cluster or two that are parallel to the diagonal line, meaning that the differences are roughly constant for each cluster. The difference is expected as we cannot observe the delays on the encoder and the decoder that are c_{enc} and c_{dec} in Equation 5. Table VI shows the details of the differences.

The results confirm that the proposed method successfully infers the playback delay except constant delay components on the encoder and the decoder sides. The actual delay can be inferred by adding these missing constants that can be measured separately, which we leave for future work.

As a side note, Android:App has two clusters but on the same sampled day. We suspect that it is caused by the client implementation, probably on the selection of manifest files. As for the differences of client types, Android:App has much longer playback delays than iOS:App, even though the applications are similar in design and they use the same HLS format. The difference is probably due to differences in design and implementations of their system components. It also suggests that different implementations have a significant impact on the total playback delay.

VII. INFERRING STREAMING DELAY

In this section, we first estimate the number of backtracked segments N for the initial buffering for different client types. Next, we look at the distribution of pause delay. Finally, we apply our method to infer playback delays for the entire server logs, and look into the distribution of playback delays.

A. Calculating Initial Player Delay from the Model

In our inference method, the initial player delay δ_{init} is derived simply from the availability time and the download time of the first segment in a sequence. Thus, we do not need

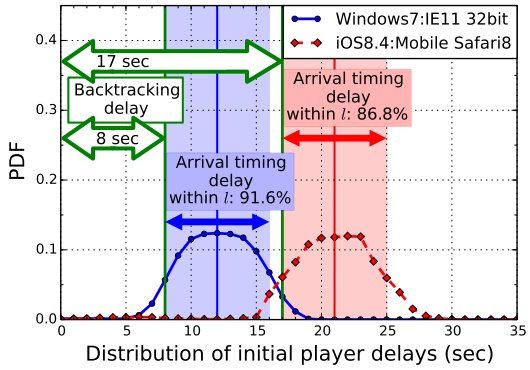

Fig. 4. Distributions of the initial player delays for two most popular User-Agent types

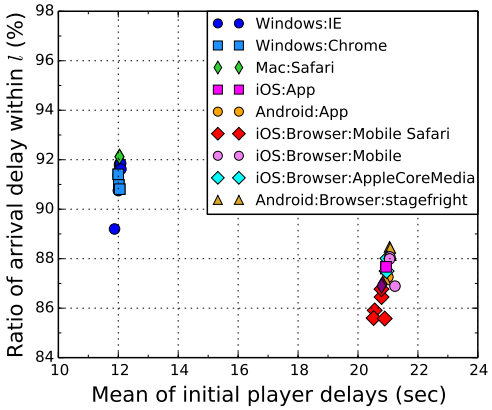

Fig. 5. Mean initial player delays of different User-Agent types

the number of backtracked segments N for initial buffering that is different for each client types.

However, we can derive N for different client types, by applying inferred results to the model. Then, it becomes possible to calculate the initial player delay δ_{init} for a client type from Equation 3, without using any dataset.

In a simple case, a client, on arrival, first downloads the manifest file with the latest segment i, and then, downloads the first segment $(i - N)$. Because we can infer the latest segment in the manifest file using the availability time, we can infer N.

In reality, however, it is not that simple because many clients downloads the manifest file multiple times before downloading the first segment, and it often takes several seconds. As a result, it is difficult to identify the corresponding manifest download for the first segment file.

Thus, we extract sequences in which the first manifest file and the first segment file are downloaded back to back, within one second, the timestamp resolution of the logs. This corresponds to $(c_{play} = 0)$ in our model.

We used the extracted sequences and inferred the initial player delay δ_{init} from these sequences. Fig. 4 depicts the distribution of δ_{init} for the two most popular User-Agent types: Windows7:IE11 32bit and iOS8.4:Mobile Safari8. The distribution is supposed to be a mixture distribution of a

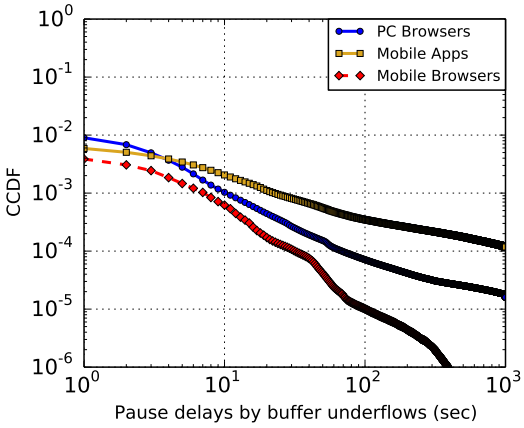

Fig. 6. Distributions of pause delays

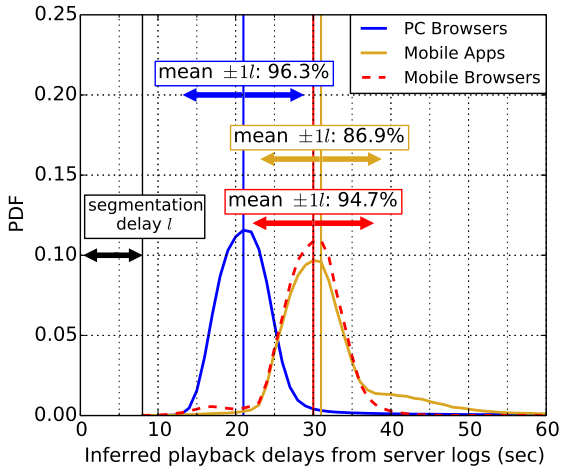

Fig. 7. Distributions of inferred playback delays for each client type

rectagular distribution for the arrival timing delay $[0, l)$ and a normal distribution for measurement variability. In fact, each distribution in the figure looks like such a distribution, and the vast majority of the values fall into the range of l seconds caused by the arrival delay. Thus, the backtracking delay corresponds to $(mean - l/2)$, the left edge of the grayed area in the figure.

We applied this method to other User-Agent types, and the results are plotted in Fig. 5. In the figure, the mean value of δ_{init} is plotted on the X-axis and the population within the range of l is plotted on the Y-axis.

The User-Agent types are clearly divided into two groups: one for PC and the other for mobile. Each group includes multiple User-Agent types with different versions. The backtracking delay is 8 seconds for PC Browsers, and 17 seconds for Mobile Apps and Mobile Browsers. Therefore, we estimate the number of backtracked segments N to be 1 for PC Browsers and 2 for Mobile Apps and Mobile Browsers.

Applying these numbers for N and using $(\alpha = 0.5)$ and $(c_{play} = 0)$ in Equation 3, we obtain the initial player delay δ_{init} to be 20 seconds for PC Browsers, and 28 seconds for Mobile Apps and Mobile Browser as shown in Table VII.

These numbers are derived only from the extracted sequences with $(c_{play} = 0)$ but also match the original sequences. The inferred initial player delay from the original sequences is 20.7 for PC Browsers and 29.4 seconds for Mobile Apps and Mobile Browsers.

B. Impact of Pause Delay

The distribution of pause delay $P(i)$ is shown in Fig. 6. Less than 1% of the segments experienced pause delays. Large pause delays seem to be caused by bugs in implementations where a player does not restart after a long pause (likely triggered by a user action) and continues fetching the stale next segment. The longest pause delays are beyond 10 hours, and they are certainly not what users intend to watch.

C. Inferring Playback Delay

Finally, we apply the proposed method to the entire server logs of Summer Koshien 2015 in order to observe the distribution of playback delays for the entire users. We first removed sequences considered to be non-live, using 60 seconds for the threshold of the inferred playback delay. These values are clear outliers of the distribution of proper live viewing in Fig. 7. The outliers seem to be caused by bugs; for example, some players restart from the beginning in the manifest file after the live streaming ended.

TABLE VIII
SUMMARY OF THE INFERRED PLAYBACK DELAYS

Client Type	PC Browsers	Mobile Apps	Mobile Browsers
mean playback delay (sec)	21.8	32.1	29.6
mean initial playback delay (sec)	20.7	30.8	28.8
mean pause delay (sec)	1.8	1.6	1.0
proportions of mean ± 1 l	96.3%	86.9%	94.7%
proportions of mean ± 2 l	98.2%	96.5%	99.5%

Fig. 7 shows the distribution of inferred playback delays for each client type. Each distribution looks like a normal distribution. From Table VIII, more than 96.5% of delays are in the range of $\pm 2l$ from mean of the distribution. The tail of the distribution is bounded, if we exclude clear outliers.

Because the encoder and decoder delays, c_{enc} and c_{dec}, are not included in the plot, the actual delay distribution would be shifted to the right by $c_{enc} + c_{dec}$. However, the basic shape of the distribution would not change much, although the value for c_{dec} would be different for different implementations and would contribute to higher variability.

Also, we can observe that the major delay factor is the initial playback delay. The mean pause delay is less than 2 seconds and its contribution to the playback delay is small. Hence, the major playback delay factors are segment length l and the backtracked segments N in the initial playback delay.

VIII. Related Work

There are many viewers for large-scale live events such as the Olympics, World Cup or Super Bowl [7], [8], and live streaming of User Generated Content (UGC), like eSports

provided by twitch [9] and YouTube Live [10], have formed a certain market in recent years [11], [12].

It is well known that the playback delay in live streaming over HTTP is much larger and more variable than that in dedicated streaming protocols [1]. There exist studies that investigated into delays in live streaming over HTTP, but they require measurement on the client side. Swaminathan *et al.* introduced a delay model for HTTP-based live streaming, and measured the delay by embedding time information in video frames [13]. Li *et al.* looked into commercial Internet TV for mobile devices, and analyzed delays combining server logs and analytic modules on clients [14]. Kupka *et al.* analyzed football live streaming, and showed delay from the download time of a segment to the completion time of the playback [15].

In contrast, we propose a method to infer delays only from server logs. Our proposed method enables to infer delays from commonly used Web server logs, and allows providers to estimate playback delays without implementing tools on the client side.

Another contribution of our study is to have identified startup buffering to be the major factor in playback delay, which was not analyzed much in the previous studies.

In order to reduce playback delays, it helps using smaller segment lengths [13], [16]. However, it also increases the number of requests and server loads. Our proposed model contributes to our understanding of this trade-off for tuning server configurations.

We analyzed server logs from Summer Koshien in 2014, and introduced an idea for inferring delays [17]. In 2014, however, we did not have User-Agent recorded in the logs, or client-side data for evaluating the results. In the present work, we were able to analyze client behaviors classified by User-Agent, and evaluate the results by the ground truth dataset measured on the client side.

IX. Conclusion

We have proposed a simple model of playback delays in live streaming over HTTP, and proposed a practical method to infer playback delays for each client only from general Web server logs. We have applied this method to server logs from a large-scale live streaming event.

By comparing with the measured playback delays on the client side, we have confirmed that the inferred playback delays from server logs match the measured playback delays. From the results, we have shown the distribution of playback delay for the entire users, and found that more than 96.5% of playback delays fall into the range: mean plus-or-minus 2 segment-lengths. Moreover, we have estimated the number of backtracked segments for initial buffering for different client types, which allows to calculate the initial player delay without using further data.

Our proposed model and inference method are general enough for applying to other live streaming events. Although our results are specific to the analyzed event, we believe that our findings are also general. Having a higher resolution of timestamp in server logs would allow more accurate and detailed analysis.

There are trade-offs to select an appropriate segment length and the size of the startup buffering. Although the playback delay is just one metric for live steaming services, it is an important factor for certain types of live video streaming. One of our contributions is to have identified the impact of startup buffering on playback delay, which would be useful input for designing live-streaming systems.

One limitation of our work is due to the lack of encoder and decoder delays. As part of our future work, we plan to improve the accuracy of our method by direct measurements of these delays.

Acknowledgment

We would like to thank Dr. Tomohisa Akafuji of Asahi Broadcasting Corporation, and the Asahi Shimbun Company for providing a valuable dataset for this study. We also would like to express our appreciation to Bunji Yamamoto, Daisuke Okaniwa and Hiroshi Abe of IIJ for collecting the dataset and providing the data analysis infrastructure.

References

[1] V. Swaminathan, "Are we in the middle of a video streaming revolution?" *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1s, pp. 40:1–40:6, Oct. 2013.

[2] Wikipedia, "Japanese high school baseball championship," http://en.wikipedia.org/wiki/Japanese_High_School_Baseball_Championship.

[3] B. Yamamoto, "The latest streaming technology," *Internet Infrastructure Review*, vol. 25, November 2014.

[4] M. Ninomiya, "Report on Access Log Analysis Results for Streaming Delivery of the 2014 Summer Koshien," *Internet Infrastructure Review*, vol. 27, May 2015.

[5] IIJ, "Internet initiative japan inc." http://www.iij.ad.jp.

[6] ABC, "Asahi Broadcasting Corporation," http://www.asahi.co.jp.

[7] C. O'Riordan, "The story of the digital olympics: streams, browsers, most watched, four screens," BBC Internet Blog, Tech. Rep., 2012.

[8] J. Erman and K. Ramakrishnan, "Understanding the super-sized traffic of the Super Bowl," in *ACM IMC'13*, Barcelona, Spain, 2013, pp. 353–360.

[9] twitch, http://www.twitch.tv.

[10] "Now you can live stream on youtube," http://youtubecreator.blogspot.jp/2013/12/now-you-can-live-stream-on-youtube.html.

[11] M. Kaytoue, A. Silva, L. Cerf, W. Meira, Jr., and C. Raïssi, "Watch Me Playing, I Am a Professional: A First Study on Video Game Live Streaming," in *ACM WWW '12*, Lyon, France, 2012, pp. 1181–1188.

[12] K. Pires and G. Simon, "YouTube Live and Twitch: A Tour of User-generated Live Streaming Systems," in *ACM MMSys '15*, Portland, Oregon, 2015, pp. 225–230.

[13] V. Swaminathan and S. Wei, "Low latency live video streaming using HTTP chunked encoding," in *IEEE MMSP 2011*, Oct 2011, pp. 1–6.

[14] Y. Li, Y. Zhang, and R. Yuan, "Measurement and Analysis of a Large Scale Commercial Mobile Internet TV System," in *ACM IMC '11*, Berlin, Germany, 2011, pp. 209–224.

[15] T. Kupka, C. Griwodz, P. Halvorsen, D. Johansen, and T. Hovden, "Analysis of a Real-world HTTP Segment Streaming Case," in *EuroITV '13*, Como, Italy, 2013, pp. 75–84.

[16] N. Bouzakaria, C. Concolato, and J. Le Feuvre, "Overhead and performance of low latency live streaming using MPEG-DASH," in *IISA 2014*, July 2014, pp. 92–97.

[17] M. Ninomiya and K. Cho, "How Live is Live Streaming over HTTP? Inferring Playback Delay from Server Logs (in Japanese)," in *Internet Conference*, 2015, pp. 43–50.