Low Latency Live Video Streaming Using HTTP Chunked Encoding

Viswanathan Swaminathan [#1], Sheng Wei [#2]

[#] *Advanced Technology Labs, Adobe Systems Incorporated*
345 Park Ave, San Jose, CA 95110, USA
[1] vishy@adobe.com [2] swei@adobe.com

***Abstract*—Hypertext transfer protocol (HTTP) based streaming solutions for live video and video on demand (VOD) applications have become available recently. However, the existing HTTP streaming solutions cannot provide a low latency experience due to the fact that inherently in all of them, latency is tied to the duration of the media fragments that are individually requested and obtained over HTTP. We propose a low latency HTTP streaming approach using HTTP chunked encoding, which enables the server to transmit partial fragments before the entire video fragment is published. We develop an analytical model to quantify and compare the live latencies in three HTTP streaming approaches. Then, we present the details of our experimental setup and implementation. Both the analysis and experimental results show that the chunked encoding approach is capable of reducing the live latency to one to two chunk durations and that the resulting live latency is independent of the fragment duration.**

I. Introduction

Progressive download over HTTP has been widely used for video on demand (VOD) applications, where end users download media files from the web server progressively and watch them online. Recently, video streaming solutions using HTTP for both live video and VOD have become available[1][2]. HTTP video streaming leverages the widespread deployment of the web infrastructure. More importantly, HTTP video streaming can be scaled to a large number of users, leveraging the HTTP caches in content delivery networks (CDNs).

However, most of the current HTTP live video streaming solutions[1] are based on HTTP requests/responses at the level of a video fragment, which is a small segment of well formed media in time. As shown in Fig. 1, the client sends an HTTP request for a specific video fragment and receives the fragment via an HTTP response from the server. It is important to note that the server can send out a response only when an entire fragment has been published. Therefore, the HTTP scheme causes the latency of the live video (i.e., time difference between the time when the live event happens and when it is played back at the client) to be dependent on the fragment duration. This introduces a latency of at least one fragment duration to the total live latency in addition to the network delay. In the worst case, as in the example shown in Fig. 1, if the request for fragment 1 arrives at the server right before fragment 2 is published, the resultant live latency is two fragment durations. Since latency affects live video experiences, it is important to reduce the live latency in the HTTP streaming solutions.

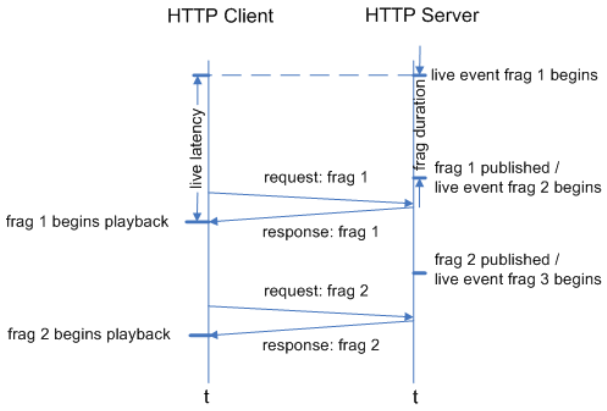


Fig. 1. Example of a *fragment*-based HTTP live video streaming. The live latency is at least 1 to 2 fragment durations.

A simple and direct solution to address the latency issues is to reduce the duration of the fragments. However, since the fragment duration is inversely proportional to the number of pairs of requests (and responses) for a certain length of video content, the reduced fragment duration would cause an explosion in the number of HTTP requests and responses, which greatly impacts the performance of HTTP servers and caches.

We propose an HTTP chunked encoding based approach to break the correlation between live latency and fragment duration. HTTP chunked encoding is a data transfer mechanism in HTTP 1.1[3] that enables the HTTP server to send out partial responses (called chunks) before the entire requested content is generated. We use chunked encoding to further divide a video fragment into multiple chunks and regard a video chunk as the basic unit for the responses. We show that the chunked encoding approach can reduce the live latency to between 1 and 2 chunk durations while keeping the total number of requests and responses unaltered. To the best of our knowledge, this is the first use of HTTP chunked encoding in live video streaming. The contributions of this paper include: (1) the first use of HTTP chunked encoding to break the correlation between live latency and fragment duration and reducing the live latency to between 1 and 2 chunk durations, and (2) an analytical model to quantify and compare the latencies in three live streaming approaches, namely *fragment*-based, *server-wait*, and *chunked encoding* approaches.

978-1-4577-1434-4/11/$26.00 ©2011 IEEE

II. Related Work

Video streaming has become a popular application that generates more than half of the Internet traffic[4]. Begen et al.[1] provide a comprehensive summary of video streaming over the Internet. There are two main categories of video streaming: push-based techniques and pull-based techniques. In the push-based scheme, the server pushes the video content to the client using a persistent or multicast session, e.g., real-time streaming protocol (RTSP) in conjunction with real time protocol (RTP)[5] and Adobe's real time messaging protocol (RTMP)[6]. In the pull-based approach, the client actively requests video from the server (e.g., via HTTP) and plays it back. Progressive download widely used in online video applications[7] is a good example of pull-based approach. Recently, several HTTP streaming frameworks and systems have become available, e.g., Adobe Flash Media Server[8] and OSMF Player[9], Microsoft Smooth Streaming[10], and Apple HTTP Live Streaming[11]. In October 2010, MPEG's Dynamic Adaptive Streaming over HTTP (DASH) standard has reached the committee draft state[12]. HTTP-based approaches leverage the simplicity and widespread deployment of HTTP servers and caches that are cost effective compared to the push-based approaches. However, all the existing HTTP streaming solutions are based on 2∼4 or even 10 second video fragments and do not provide a low latency live video experience.

HTTP chunked transfer encoding supported by HTTP 1.1[3] enables a web server to transmit partial responses in chunks before the complete response is ready. It has been widely used in web applications with dynamically generated content, where the length of the response is unknown until the full content is generated. To the best of our knowledge, HTTP chunked encoding has not been used in live video streaming for user experience optimization.

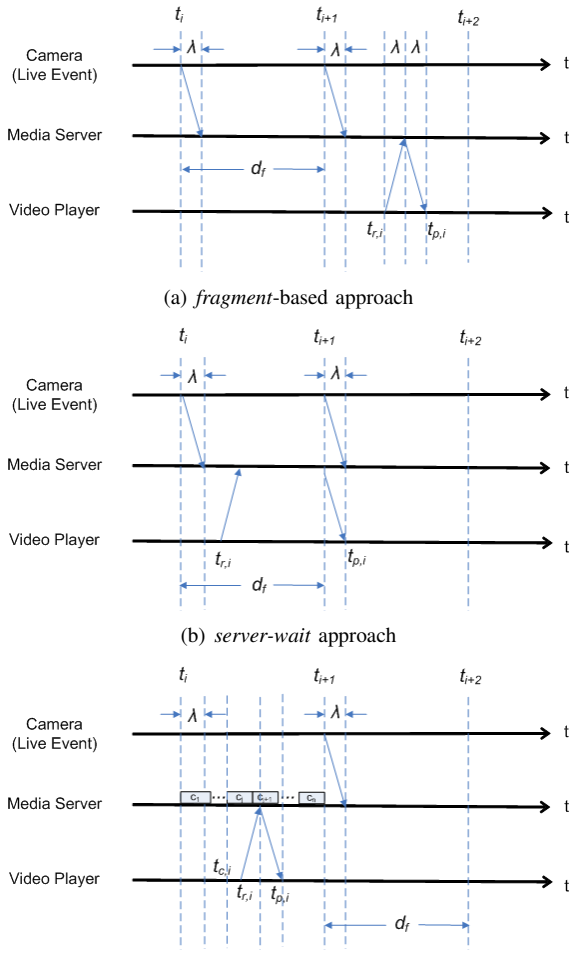
III. Description and Analysis of HTTP Live Streaming Approaches

In this section, we develop an analytical model to quantify the live latency in HTTP live video streaming. We first introduce a model to analyze the live latency in the existing *fragment*-based approach, in which we confirm that the live latency is approximately 1 to 2 fragment durations. Then, we propose two new approaches for low latency in HTTP live streaming, namely *server-wait* and *chunked encoding* approaches, and analyze the live latency in each of them using the same analytical model. Besides live latency, we also analyze another metric, namely starting delay, which is defined as the delay between the time when the client sends out the first fragment request and when it begins playing the first fragment. The starting delay determines how long the end user has to wait before the video begins, which is another important factor in the live video experience. Table I summarizes the symbols for the parameters and metrics that we use in the analytical model.

[1]This includes the encoding and decoding latencies. For simplicity, we aggregate these two different latencies as they are infinitesimally small compared to the fragment duration.

TABLE I
Definition of used parameters in the analytical model.

Parameters	Definition
t_i	Time when the live event of fragment i begins.
λ	Network delay.[1]
$t_{r,i}$	Time when client requests fragment i.
$t_{p,i}$	Time when client begins playing back fragment i.
$t_{c,i}$	Time when the most recent chunk in fragment i begins.
d_f	Fragment duration.
d_c	Chunk duration.
L_i	Live latency of fragment i.
D_i	Starting delay of fragment i, i.e., $t_{p,i} - t_{r,i}$.



(a) *fragment*-based approach

(b) *server-wait* approach

(c) *chunked encoding* approach

Fig. 2. Analytical model for live latency and starting delay.

A. Fragment-Based Approach

The *fragment*-based approach is used in the existing HTTP live streaming solutions[9][10][11]. The basic unit of HTTP requests and responses is a video fragment, which is defined in the magnitude of seconds (e.g., 4 seconds in a typical live streaming use case). Pseudocode 1 describes the detailed procedure of the *fragment*-based scheme. The client first requests the metadata to bootstrap the video session (henceforth referred as bootstrap) from the server, which indicates the most recent fragment that has been published. Then, the client

requests the fragment indicated by the bootstrap, and the server responds with the entire requested fragment.

Pseudocode 1 *fragment*-based HTTP live streaming

```
1: Client requests bootstrap information;
2: Server responds with bootstrap indicating fragment i has been published;
3: repeat
4:    Client requests fragment i;
5:    Server responds with fragment i;
6:    Client plays fragment i;
7:    i ← i + 1;
8: until Client stops playing;
```

We analyze the live latency and starting delay of the *fragment*-based scheme using the three time lines (camera, server, and client) shown in Fig. 2(a). The live latency of video fragment i can be expressed as:

$$\begin{aligned} L_i &= t_{p,i} - t_i \\ &= t_{r,i} - t_i + 2\lambda \\ &= d_f + 2\lambda + t_{r,i} - t_{i+1} \end{aligned} \tag{1}$$

In the *fragment*-based approach, the client would request fragment i after fragment i has been published to the media server and before fragment $i+1$ is read.[2] Therefore, $t_{r,i}$ is within the following range:

$$t_{i+1} + \lambda \leq t_{r,i} \leq t_{i+2} + \lambda \tag{2}$$

Combining Equations (1) and (2) and considering $d_f = t_{i+2} - t_{i+1}$, we obtain:

$$d_f + 3\lambda \leq L_i \leq 2d_f + 3\lambda \tag{3}$$

The starting delay of the *fragment*-based approach can be calculated as:

$$D_i = t_{p,i} - t_{r,i} = 2\lambda \tag{4}$$

B. Server-Wait Approach

The *server-wait* approach is a new streaming scheme with a small modification to the *fragment*-based approach. The fundamental change is that the client requests the current video fragment that is being published and not the one that has already been completely ready and published. In order to accomplish this, the client requests the next fragment after the one indicated by the bootstrap information. Since the fragment has not been completely ready at the time when the client requests it, the server would wait until the completion of the fragment before it sends out the response. We describe the detailed procedure of the *server-wait* scheme in Pseudocode 2. The modifications compared to the *fragment*-based approach lie in lines 4∼6, where the client requests the next fragment, and the server waits for the completion of the fragment.

[2]In actual scenarios, a client buffer may be used and the request may be delayed. However, this analysis focuses on minimum live latency and ignores the client buffer.

Pseudocode 2 *server-wait* HTTP live streaming

```
1: Client requests bootstrap information;
2: Server responds with bootstrap indicating fragment i has been published;
3: repeat
4:    Client requests fragment i + 1;
5:    Server waits until fragment i + 1 is published;
6:    Server responds with fragment i + 1;
7:    Client plays fragment i + 1;
8:    i ← i + 1;
9: until Client stops playing;
```

Fig. 2(b) shows the timelines and major events in the *server-wait* scheme. Similar to Equation (1), we calculate the live latency as shown in Equation (5) to be approximately one fragment duration.

$$L_i = t_{p,i} - t_i = d_f + \lambda \tag{5}$$

Similar to Equation (4), the starting delay can be expressed as:

$$D_i = t_{p,i} - t_{r,i} = t_{i+1} + \lambda - t_{r,i} \tag{6}$$

Since the client requests the current fragment that is being published, we have the following range for $t_{r,i}$:

$$t_i + \lambda \leq t_{r,i} \leq t_{i+1} + \lambda \tag{7}$$

Combining Equations (6) and (7) and considering $d_f = t_{i+1} - t_i$, we obtain the range for D_i:

$$0 \leq D_i \leq d_f \tag{8}$$

C. Chunked Encoding Approach

The *server-wait* approach reduces the live latency to approximately one fragment duration. However, it still does not break the correlation between live latency and fragment duration. Also, the increased starting delay (up to one fragment duration) raises another concern to the user experience. In order to break the correlation and further reduce the live latency, we propose an HTTP chunked encoding based approach. Our intuition is that the only way to break the fragment dependence is to introduce a finer granularity than that of a video fragment for HTTP responses. On the other hand, in order not to cause the number of requests/responses to explode, the basic unit for HTTP request (i.e., fragments) should not become smaller. These two goals seem contradictory to each other. However, we show that both of them can be satisfied by leveraging HTTP chunked encoding as defined in the HTTP 1.1 specification[3].

HTTP chunked encoding enables the HTTP server to send out partial responses to a request for dynamically generated content. The partial responses can be assembled by the client to form up an entire response. More importantly, the client is able to consume any received partial response before the reception of all the parts. Note that chunked encoding breaks up the basic unit of responses while maintaining that of requests, which satisfies both goals we discussed for live

latency reduction. Pseudocode 3 shows the detailed procedure of the *chunked encoding* approach. We evenly divide a video fragment into n smaller chunks for chunked encoding. Similar to the *server-wait* approach, the client requests the fragment that is being published (fragment $i+1$ in line 4). On the server side, although fragment $i+1$ is not complete, there are j chunks that are already published. In this case, the server would send out chunk j immediately after it receives the client request and keep sending the remaining chunks of fragment $i+1$ once they are ready. The client can play chunk j as soon as it is delivered.

Pseudocode 3 *chunked encoding*-based HTTP live streaming

```
1:  Client requests bootstrap information;
2:  Server responds with bootstrap indicating fragment i has
    been published;
3:  repeat
4:     Client requests fragment i + 1;
5:     Server checks that chunk j of fragment i + 1 has been
       published;
6:     while j ≤ n do
7:        // n is the number of chunks in a fragment
8:        Server sends out chunk j;
9:        Client plays chunk j
10:       j ← j + 1;
11:    end while
12:    i ← i + 1;
13: until Client stops playing;
```

The analysis of timelines for the *chunked encoding* approach is shown in Fig. 2(c). Since the client can start playing chunk j as soon as it is delivered as a partial response, the live latency becomes:

$$L_i = t_{p,i} - t_{c,i} = t_{r,i} + 2\lambda - t_{c,i} \tag{9}$$

According to Fig. 2(c) and Pseudocode 3, the client request for fragment $i+1$ arrives in between the completion times of chunk j and chunk $j+1$. Therefore,

$$t_{c,i} + d_c \leq t_{r,i} + \lambda \leq t_{c,i} + 2d_c \tag{10}$$

Combining (9) and (10), we have the following range for live latency:

$$d_c + \lambda \leq L_i \leq 2d_c + \lambda \tag{11}$$

From Fig 2(c), the starting delay stays the same as the *fragment*-based approach:

$$D_i = t_{p,i} - t_{r,i} = 2\lambda \tag{12}$$

D. Summary and Discussion

We summarize the analysis of live latency and starting delay for the three schemes in Table II. We assume the typical case that the network and encoding/decoding latency is in the order of hundreds of milliseconds, and the fragment/chunk duration is in the order of seconds. Therefore, $\lambda << d_c$ and $\lambda << d_f$. Based on this assumption, we can conclude that the *server-wait* approach reduces the live latency from 1∼2 fragment durations to 1 fragment duration, but it increases the starting delay to up to 1 fragment duration. The *chunked encoding* approach further reduces the live latency to 1∼2 chunk durations while still maintaining the original low starting latency. In a typical use case when we set d_f=4 sec and d_c=1 sec, we can obtain 4x reduction in the live latency while maintaining the same starting delay (2λ, where $\lambda << d_c$). More importantly, in the *chunked encoding* scheme, the live latency is independent of fragment duration, which enables the possibility of increasing the fragment duration and thus reducing the number of requests and responses in HTTP live video streaming. This results in reducing the load on HTTP servers/caches and improving their performance considerably. This is another huge benefit we obtain from the *chunked encoding* scheme.

TABLE II
SUMMARY OF LIVE LATENCY AND STARTING DELAY ANALYSIS

Approach	Live Latency	Starting Delay
fragment-based	$(d_f + 3\lambda) \sim (2d_f + 3\lambda)$	2λ
server-wait	d_f	$0 \sim d_f$
chunked encoding	$(d_c + \lambda) \sim (2d_c + \lambda)$	2λ

IV. Design and Implementation of HTTP Chunked Encoding Based Live Video Streaming

We designed and implemented the *chunked encoding* scheme into a live video streaming prototype. The overall architecture of the prototype is shown in Fig. 3. The live video encoder encodes the live video captured by a camera into a common video format (e.g., mp4 and f4v). Then, the encoded video is published to the media server via a low latency push-based media streaming protocol, such as RTP or RTMP.[3] The live video packager at the media server packages the video stream into video chunks and stores them into a media file. The media file is deployed on an HTTP server that provides live video streaming to the video player via HTTP requests and responses. In this section, we discuss the design and implementation issues in the following components: live video packager, media file, HTTP server, and video player.

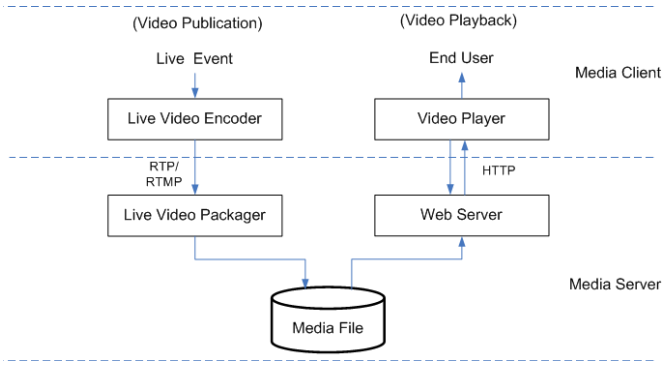


Fig. 3. Architecture and data flow of the live video streaming system.

[3]We use RTMP in the implementation of our prototype.

A. Live Video Packager

The live video packager receives the live video stream from the video encoder in a frame-by-frame manner. It outputs the fragmented video stream to a media file. Each fragment is further divided into smaller chunks with a constant chunk duration, d_c. We use constant chunk duration to ensure that the times needed for publishing and playing a video chunk are approximately equal, which ensures that the server is able to send out the next chunk before the depletion of the current chunk from the client buffer. We use a time-based approach to package the live video stream into chunks. The live video packager keeps a buffer to hold the incoming video frames. When the time interval between the oldest and newest frames in the packaged content exceeds d_c, the live video packager flushes the content of the buffer as a new chunk to the media file.

B. Media File

The media file stores the video content that has been published and serves as the source for live video streaming to the end user. In the existing HTTP live video streaming scheme without chunked encoding, the format of the media file is based on fragments, as shown in Fig. 4(a). With chunked encoding, as shown in Fig. 4(b), video chunks become the basic unit in the media format. However, the new format still maintains the fragment information (e.g., metadata (post) indicates the end of a fragment), because the HTTP request from client is still on a fragment basis.

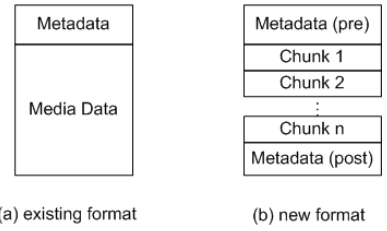


Fig. 4. Format of media files (one fragment).

C. HTTP Server

The HTTP server sends out video fragments on a chunk-by-chunk basis upon a fragment request from the client. We use a pull-based mechanism to check periodically whether a chunk is published and ready to be sent. We show the processing flow of the HTTP server in Fig. 5. The server waits for a pre-configured time and checks the media file for new chunks. If there is any new chunk available, it forms up a response with the chunk and sends it to the client. If the current chunk is the last one of the entire fragment, which is indicated by the metadata (post) in Fig. 4(b), the server sends out a zero chunk which ends the response to the current fragment request. Note that a push mechanism is arguably a superior approach that would give better server performance. Implementing a push based approach and its performance impact is currently left as future work.

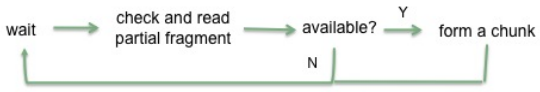


Fig. 5. Flow of the HTTP server.

D. Video Player

The video player has two tasks in the *chunked encoding* approach. First, it requests the current fragment that is being published to the media server. Second, it plays back the video as soon as receiving each chunk from the HTTP server.

One possible downside of using a low latency system (irrespective of whether it is traditional real time or HTTP streaming) is that the video buffer at the player may not maintain a large buffer of video content, because any content is played back as soon as it is received. If there is a temporary slowdown or packet loss in the network, it may cause the video buffer to be depleted before the next chunk arrives. In this case, the video playback may freeze and the live latency would increase by at least the freezing time. By the time when the network recovers, the video playback can still continue as the accumulated chunks would still be received via the reliable HTTP connection. However, the live latency cannot catch up to compensate for the freezing time unless the player decides to skip ahead.

We address this issue by enabling a fast forwarding mechanism in the video player.[4] The fast forwarding mechanism ensures that the client plays back the video content faster than the wall clock time, in the case where the length of the video buffer is beyond a threshold value. In our implementation of the prototype, we set the threshold value for fast forwarding as the chunk duration d_c.

V. Experiment Results

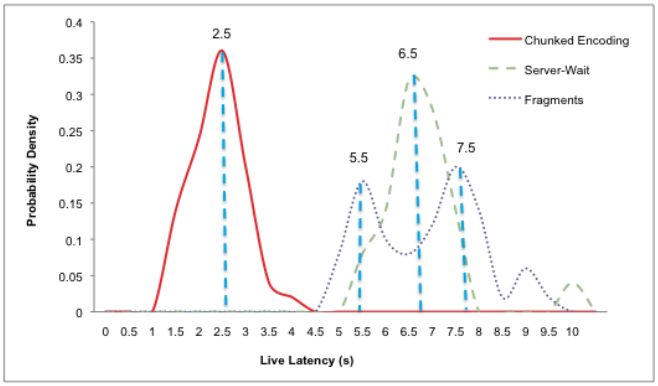
A. Live Latency Measurements

We embed timecode in the video while encoding to measure the live latency. In particular, the encoder embeds a timecode (system time) to a video stream every 15 frames. When the client plays a video frame that has a timecode, it fetches the system time and subtracts the timecode from it to obtain the live latency for that frame. By collecting the calculated live latency every 15 frames at the client, we can obtain the profile of live latency over time. Furthermore, we either ensure that the system time is synchronized between the encoder and the player or we run the two on the same machine when collecting experimental results.

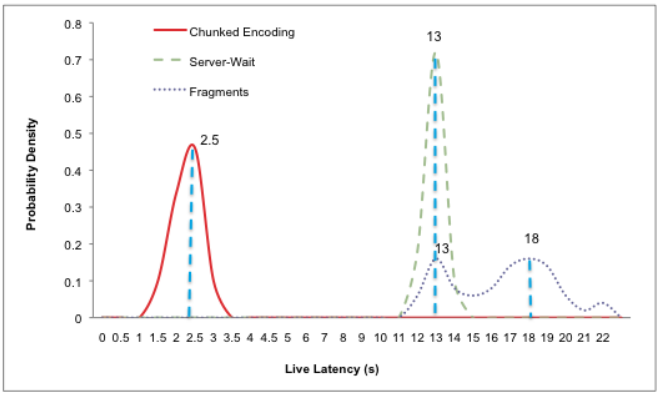
B. Live Latency

We measure the live latency of the *fragment*-based, *server-wait*, and *chunked encoding* approaches in order to verify our analytical results in Section III. Since the client may begin the live video streaming at any random time point, we repeat the experiment using each approach 50 times to obtain a distribution of possible live latency values. Fig. 6

[4]We use the fast forwarding mechanism in the OSMF player[9].



(a) d_f=4 sec



(b) d_f=10 sec

Fig. 6. Probability density of live latency using *fragment*-based, *server-wait*, and *chunked encoding* schemes: (a) when the fragment duration is 4 seconds, and (b) when the fragment duration is 10 seconds.

shows our experimental results of the frequency (probability density) at which each latency value occurs. In Fig. 6(a), we set fragment duration d_f=4 sec and chunk duration d_c=1 sec. We observe that the most likely latency value in the *fragment*-based, *server-wait*, and *chunked encoding* approaches is 5.5∼7.5 sec, 6.5 sec, and 2.5 sec, respectively. This matches with our analytical results in Section III (d_f∼2d_f, d_f, and d_c∼2d_c, respectively). In Fig. 6(b), we set d_f=10 sec and d_c=1 sec. The latency values in the three approaches follow the same pattern that matches our analytical results. Note that in Fig. 6(b) the most likely live latency in the *chunked encoding* approach is the same as that in Fig. 6(a) (2.5 sec). This validates that the live latency in our *chunked encoding* approach is independent of fragment duration.

C. Reliability

We evaluate the reliability of the *chunked encoding* approach by executing the implemented prototype on a lossy wireless network. Fig. 7 shows our experimental results for a 800-second video playback, where d_f=4 sec and d_c=1 sec. The live latency stays constantly around 2.5 sec in more than 90% of the time. During the playback, we observe around 10 times that the video buffer is depleted that causes a jitter in the live latency. However, the latency can recover very soon (within seconds) because of the use of the fast forwarding scheme. Also, the jitters can be avoided by increasing the client buffer, which is a common approach in any HTTP or non-HTTP live streaming approaches for improving user experiences.

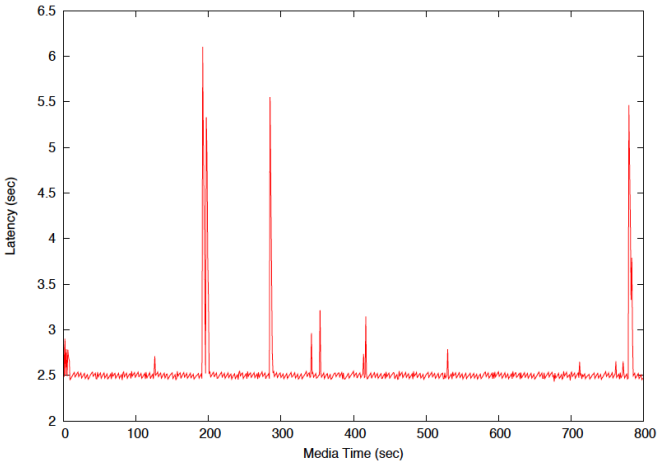


Fig. 7. Reliability test using the *chunked encoding* approach. The video streaming is on a lossy wireless network and lasts for 800 seconds.

VI. Conclusion

We have introduced a new HTTP chunked encoding approach to reduce the latency in live video streaming. We compared the *chunked encoding* approach with the *fragment*-based and *server-wait* schemes in the proposed analytical model and real experiments. Both the analysis and experimental results showed that the chunked encoding approach reduces the latency from 1∼2 fragment durations to 1∼2 chunk durations while still keeping a low starting delay. We showed that the low latency HTTP live video streaming approach enables the possibility of using large fragments to improve HTTP server and cache performances.

References

[1] A. Begen, T. Akgul, M. Baugher, Watching Video over the Web. IEEE Internet Computing, Vol. 15, No. 2, 2011, pp. 54-63.
[2] S. Akhshabi, A. Begen, C. Dovrolis, An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP, MMSys 2011, pp. 157-168.
[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt
[4] Cisco Visual Network Index (VNI), 2009, http://www.cisco.com
[5] H. Schulzrinne, A. Rao, R. Lanphier, Real Time Streaming Protocol (RTSP), RFC 2326, April 1998. http:/www.rfc-editor.org/rfc/rfc2326.txt
[6] Real-Time Messaging Protocol (RTMP) specification 1.0, Adobe Systems Incorporated, http://www.adobe.com/devnet/rtmp.html
[7] YouTube, http://www. youtube.com
[8] Adobe Flash Media Server, http://www.adobe.com/products/flashmediaserver/
[9] Open Source Media Framework, http://www.opensourcemediaframework.com/
[10] Microsoft Smooth Streaming, http://www.iis.net/download/SmoothStreaming
[11] Apple HTTP Live Streaming, http://developer.apple.com/resources/http-streaming/
[12] MPEG advances Dynamic Adaptive Streaming over HTTP (DASH) toward completion, http://mpeg.chiariglione.org