Signal Processing: *Image Communication* 26 (2011) 378–389

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Client intelligence for adaptive streaming solutions

Dmitri Jarnikov [a,b,*], Tanır Özçelebi [b]

[a] *Irdeto, the Netherlands*
[b] *Eindhoven University of Technology, the Netherlands*

ARTICLE INFO

Available online 21 March 2011

Keywords:
IPTV
Adaptive
Streaming
Client intelligence
Modeling decision-making
Internet TV

ABSTRACT

In state-of-the-art adaptive streaming solutions, to cope with varying network conditions, the client side can switch between several video copies encoded at different bitrates during streaming. Each video copy is divided into chunks of equal duration. To achieve continuous video playback, each chunk needs to arrive at the client before its playback deadline. The perceptual quality of a chunk increases with the chunk size in bits, whereas bigger chunks require more transmission time and, as a result, have a higher risk of missing transmission deadline. Therefore, there is a trade-off between the overall video quality and continuous playback, which can be optimized by proper selection of the next chunk from the encoded versions. This paper proposes a method to compute a set of optimal client strategies for this purpose.

1. Introduction

Nowadays many people explore the possibilities for consuming multimedia over IP networks using Internet TV, IPTV or Mobile TV solutions. The majority of solutions on the market use real-time streaming. The advantages of real-time streaming – real-time content delivery (live events viewed as they happen) and the ability to play content as soon as transmission is started – barely balance the burden on content providers to provide streaming servers and develop sophisticated techniques for overcoming users' bandwidth constraints and for traversing firewalls [1]. Also, traditional streaming lacks scalability, since, in practice, it is based on unicast (RTP/UDP [2] or proprietary protocols).

Recently, new technologies that employ progressive download have emerged with the aim of overcoming weaknesses of streaming while providing a similar quality of experience (QoE) to end-users. To access content via progressive download, a user-device makes an HTTP/TCP [3] request for (typically consecutive) parts of the content from the server. Given sufficient pre-roll delay, this approach enables solutions that require only a standard web server, guarantee reliable high-quality delivery without data loss and easily cross most firewalls. Moreover, the use of HTTP allows these solutions to benefit from proxy caching, which reduces transmission latency and decreases server/network loads.

The concept of progressive download for delivering multimedia content has spawned protocols that allow transmitting content as a set of time-bounded segments called *chunks*. Chunks can be separated physically (a separate file for each chunk of content) or logically (all content chunks stored in a single file). For protocols that are based on physically separated chunks, progressive download starts with the client downloading a description file that contains an ordered list of uniform resource identifiers (URI) referring to media files (each containing a single chunk of content) that the client may consume. Protocols that are based on logically separated chunks rely on a file format that allows addressing parts of the content at a number of predefined access points (time offsets). For such protocols, at the start of the progressive download session, the client downloads

* Corresponding author at: Irdeto, the Netherlands and Eindhoven University of Technology, the Netherlands.
E-mail addresses: d.s.jarnikov@tue.nl, djarnikov@irdeto.com (D. Jarnikov), t.ozcelebi@tue.nl (T. Özçelebi).

0923-5965/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2011.03.003

a description file that contains the name of the media file and the rules to create a URI that points to a chunk with a given time offset. Physical separation can be used with HTTP Live Streaming [4] and 3GPP adaptive HTTP Streaming [5], whereas Microsoft SmoothStreaming [6] is a solution example that uses logical separation of chunks.

Since each chunk has a unique address in progressive download, web caching can be utilized, which makes progressive download more scalable than streaming.

The progressive download approach poses a big disadvantage, i.e. increased delay when bandwidth is insufficient for timely transmission of content. This is addressed by [4–6], where multiple copies of the same content with different bit-rates, spatial/temporal resolutions and/or other encoding characteristics are provided. Each copy, also called a *quality level*, consists of a sequence of time-aligned chunks. A client can switch between quality levels at run-time on a chunk-by-chunk basis to react to varying network conditions. Due to its adaptive nature, this new class of protocols is called 'adaptive streaming protocols'.

Although adaptive streaming protocols have been designed for Internet TV, they can also be used in the IPTV environment. Formally, the main difference between the two is that Internet TV delivery is done over an unmanaged network whereas IPTV delivers content over a managed network. Internet TV content, in theory, is delivered without resource reservation and quality-of-service (QoS) provisions. In reality, the vast majority of the currently deployed systems are based on content delivery networks (CDN), peer-to-peer (P2P) networks or a hybrid of the two [7]. In these technologies, connections between the edge-servers and the end-users have no QoS provisions ('last mile' problem), thus making a way to the use of adaptive streaming protocols to adapt to changing network conditions. In contrast to Internet TV, IPTV has a controlled delivery environment, because it is managed by the service provider. Nevertheless, recent developments of IPTV solutions that extend to Mobile TV and home-network, facing a 'last mile' problem that is similar to the Internet TV case. For this reason, adaptive streaming protocols have been also considered for IPTV [8].

The decision-making process on the client device is the common denominator for existing adaptive streaming protocols and it is a determining factor for the overall QoE. With adaptive streaming, it is the responsibility of the client to observe changing network conditions, predict the transmission time[1] for the next chunk from different quality levels and choose the quality level that minimizes the risk of late chunk delivery while maximizing the quality. The main challenge in using adaptive streaming protocols is making the most appropriate quality level choice in real-time under given network conditions.

To address this issue, we propose an algorithm that performs bandwidth measurements and switches between quality levels trying to maximize the user-perceived quality in the long run. The proposed algorithm may be applied to solutions that use different delivery methods—IPTV solutions where available bandwidth hardly changes and only as a result of an additional stream being delivered to the same household, Internet TV solutions where end-to-end QoS cannot be guaranteed, Mobile TV solutions where the bandwidth available to the user changes based on the cell load and radio-signal interference creates short-term bandwidth fluctuations, and IPTV/Internet TV solutions where the client is connected to the access network via a wireless network that is generally susceptible to bandwidth fluctuations. Depending on the deployment scenario, the parameters of the solution must be tuned.

The proposed algorithm can be deployed on PCs and consumer electronics (CE) devices. The latter is particularly important in the view of the growing demand for mobile video TV and services [9] along with the increasing number of Internet-connected TVs and set-top boxes.

The paper is organized as follows. Section 2 describes the challenges in choosing the right quality level in adaptive streaming. Section 3 describes the proposed strategy for switching between quality levels. Section 4 outlines the experimental setup and presents the evaluation results. Finally, Section 5 presents conclusions.

2. Adaptive streaming challenges

In classical video streaming, at the server side, video encoding is typically done without any consideration of changing network conditions. The rate control algorithms of the state-of-the-art reference encoders typically maintain a mathematical model of the decoder input buffer and try to encode video at a constant bit-rate (CBR). Given a fixed channel throughput and a decoder buffer size, this enables encoders to guarantee continuous playback of video at the receiving side. For example, the hypothetical reference decoder (HRD) [10] and the video buffering verifier (VBV) [11] models are used by the AVC/H.264 and MPEG reference encoders, respectively. HRD and VBV are based on the *leaky bucket* model, and they assume that video to be streamed is drained by a CBR channel with a rate equal to the video encoding bit-rate. The goal is to avoid buffer violations (underflows and overflows) caused by implicit bit-rate variations in the encoded bit-stream. At the encoder, the leaky bucket model simulates that the bits arrive at the input buffer of the decoder continuously and at a constant rate, complying with the CBR channel assumption. On the other hand, bits are removed from the decoder input buffer at a variable rate, depending on the size of the current frame to be decoded.

In a real streaming scenario on a best-effort network, this approach is not very accurate. Firstly, the CBR channel assumption of HRD and VBV models holds only if the network infrastructure provides guaranteed QoS, which is not the case for best-effort networks. In practice, the channel throughput varies over time due to competing traffic, changing to a different routing path, or, even, switching between different content sources [4]. This may cause increased data delays or data loss, resulting in decoder buffer underflow or overflow and leading to unwanted pauses and QoE loss. Secondly, the buffer size

[1] Transmission time is the time from the moment the client dispatches an HTTP request for the chunk until the moment the last bit of the chunk is received by the client.