Early Termination and Repositioning the Video

HTTP streaming systems often make use of the **HTTP byte-range header** in the HTTP GET request message, which specifies the specific range of bytes the client currently wants to retrieve from the desired video. This is particularly useful when the user wants to reposition (that is, jump) to a future point in time in the video. When the user repositions to a new position, the client sends a new HTTP request, indicating with the byte-range header from which byte in the file should the server send data. When the server receives the new HTTP request, it can forget about any earlier request and instead send bytes beginning with the byte indicated in the byte-range request.

While we are on the subject of repositioning, we briefly mention that when a user repositions to a future point in the video or terminates the video early, some prefetched-but-not-yet-viewed data transmitted by the server will go unwatched—a waste of network bandwidth and server resources. For example, suppose that the client buffer is full with B bits at some time t_0 into the video, and at this time the user repositions to some instant $t > t_0 + B/r$ into the video, and then watches the video to completion from that point on. In this case, all B bits in the buffer will be unwatched and the bandwidth and server resources that were used to transmit those B bits have been completely wasted. There is significant wasted bandwidth in the Internet due to early termination, which can be quite costly, particularly for wireless links [Ihm 2011]. For this reason, many streaming systems use only a moderate-size client application buffer, or will limit the amount of prefetched video using the byte-range header in HTTP requests [Rao 2011].

Repositioning and early termination are analogous to cooking a large meal, eating only a portion of it, and throwing the rest away, thereby wasting food. So the next time your parents criticize you for wasting food by not eating all your dinner, you can quickly retort by saying they are wasting bandwidth and server resources when they reposition while watching movies over the Internet! But, of course, two wrongs do not make a right—both food and bandwidth are not to be wasted!

7.2.3 Adaptive Streaming and DASH

Although HTTP streaming, as described in the previous subsection, has been extensively deployed in practice (for example, by YouTube since its inception), it has a major shortcoming: All clients receive the same encoding of the video, despite the large variations in the amount of bandwidth available to a client, both across different clients and also over time for the same client. This has led to the development of a new type of HTTP-based streaming, often referred to as **Dynamic Adaptive Streaming over HTTP (DASH)**. In DASH, the video is encoded into several different versions, with each version having a different bit rate and, correspondingly, a different quality level. The client dynamically requests chunks of video segments of a few seconds in length from the different versions. When the amount of available

bandwidth is high, the client naturally selects chunks from a high-rate version; and when the available bandwidth is low, it naturally selects from a low-rate version. The client selects different chunks one at a time with HTTP GET request messages [Akhshabi 2011].

On one hand, DASH allows clients with different Internet access rates to stream in video at different encoding rates. Clients with low-speed 3G connections can receive a low bit-rate (and low-quality) version, and clients with fiber connections can receive a high-quality version. On the other hand, DASH allows a client to adapt to the available bandwidth if the end-to-end bandwidth changes during the session. This feature is particularly important for mobile users, who typically see their bandwidth availability fluctuate as they move with respect to the base stations. Comcast, for example, has deployed an adaptive streaming system in which each video source file is encoded into 8 to 10 different MPEG-4 formats, allowing the highest quality video format to be streamed to the client, with adaptation being performed in response to changing network and device conditions.

With DASH, each video version is stored in the HTTP server, each with a different URL. The HTTP server also has a **manifest file**, which provides a URL for each version along with its bit rate. The client first requests the manifest file and learns about the various versions. The client then selects one chunk at a time by specifying a URL and a byte range in an HTTP GET request message for each chunk. While downloading chunks, the client also measures the received bandwidth and runs a *rate determination algorithm* to select the chunk to request next. Naturally, if the client has a lot of video buffered and if the measured receive bandwidth is high, it will choose a chunk from a high-rate version. And naturally if the client has little video buffered and the measured received bandwidth is low, it will choose a chunk from a low-rate version. DASH therefore allows the client to freely switch among different quality levels. Since a sudden drop in bit rate by changing versions may result in noticeable visual quality degradation, the bit-rate reduction may be achieved using multiple intermediate versions to smoothly transition to a rate where the client's consumption rate drops below its available receive bandwidth. When the network conditions improve, the client can then later choose chunks from higher bit-rate versions.

By dynamically monitoring the available bandwidth and client buffer level, and adjusting the transmission rate with version switching, DASH can often achieve continuous playout at the best possible quality level without frame freezing or skipping. Furthermore, since the client (rather than the server) maintains the intelligence to determine which chunk to send next, the scheme also improves server-side scalability. Another benefit of this approach is that the client can use the HTTP byte-range request to precisely control the amount of prefetched video that it buffers locally.

We conclude our brief discussion of DASH by mentioning that for many implementations, the server not only stores many versions of the video but also separately stores many versions of the audio. Each audio version has its own quality level and bit rate and has its own URL. In these implementations, the client dynamically selects both video and audio chunks, and locally synchronizes audio and video playout.