Lecture Handout: Video-on-Demand Streaming

Overview

In this lecture, we look at how video-on-demand (VoD) platforms like YouTube and Netflix stream stored videos over the Internet. The key ideas we discussed are content distribution networks (CDNs), splitting video into chunks, compressing chunks at multiple qualities, and client-based buffering and bandwidth adaptation to provide smooth playback under varying network conditions.

From User -> CDN -> Video playback

When a user clicks on a video on a browser or taps “Play” in the YouTube (YT) app, the request is usually routed not to a central “origin” server, but to a nearby Content Delivery Network (CDN) cache that stores popular videos. This indirection reduces latency and offloads traffic from the origin by serving content from locations close to the user in network terms. It also helps in load balancing and handling traffic spikes.

CDNs hold copies of the same video at multiple sites; the Domain Name System (DNS) server and application logic choose a “good” CDN node based on factors like geography, load, and peering relationships. Once the CDN node is selected, the browser or YT app uses HTTP-based (GET requests) delivery of video chunks, so it runs over UDP (best effort for video) or sometimes TCP, and fits naturally into the existing web infrastructure.

Chunks, qualities, and stored video

A stored VoD file is not streamed as one long byte stream; instead, it is preprocessed into a sequence of short segments (chunks), each typically on the order of 1-2 seconds of playback. This allows continuous streaming and playback. Every chunk is encoded at multiple quality levels (for example, 240p, 480p, 720p, 1080p, up to 4K), so the same time interval of the video exists in several bitrate “representations”.

The platform then distributes these multi-encoded chunks to CDN servers so that for each time index (i), the CDN can serve chunk (i) at any of the available resolutions depending on what the client later requests. Higher resolutions (e.g., 1080p and 4K) require substantially more bandwidth than lower resolutions (e.g., 240p or 480p). For example, check the table in the accompanying slides.

Client buffering and adaptive streaming

On the client side, the player does not simply request “the whole video”; it issues HTTP GETs for individual chunks, fills a playback buffer ahead of the current playhead, and tries to keep that buffer from running dry. This reduces pauses and interruptions during playback. A deeper buffer gives more protection against transient throughput dips but also increases startup delay and makes user actions like seeking feel less instantaneous.

But the network is dynamic, particularly with bandwidth changes that often occur. For example, a user is connected to a 5G network but is in a car from Los Angeles to San Francisco. This means YT app needs to

continuously stream video without pausing. Adaptive streaming protocols like DASH (Dynamic Adaptive HTTP Streaming) try to solve this issue. The client periodically estimates available downlink bandwidth and chooses, for the next few chunks to buffer, the highest-quality representation that seems safe given the bandwidth estimate, i.e., size of next chunk < available bandwidth * chunk length (+ tolerance). When available bandwidth reduces, the client temporarily switches to lower-resolution chunks to avoid pause; when conditions improve, it moves back up to higher-quality chunks.

Food for thought

- Buffering ahead of the user's playback time point helps in smooth playback even when the network delay or bandwidth varies. How do you think YT handles whenthe user jumps in time (seeks) to a different part of the video? **Hint:** YT shows a curve when you hover over or drag the timer (progress bar) to show which parts of the video has attracted more views. Also, YT allows content creators to insert “chapters” in the video. How might these features help in seeking?
- In the lecture, I mentioned buffering for traditional videos, which are mostly long format. Today's video platforms also have short-form videos (e.g., YouTube Shorts, TikTok), which are typically less than a minute long. Do you think buffering and adaptive bitrate logic are still relevant for such short videos? **Hint:** Buffering across the swiping direction of the user?