Adaptive Bitrate Streaming used for videos

This method involves encoding the content at multiple bitrates and resolutions, resulting in several rendition versions of the same video. During playback, players enhance the user experience by selecting the best possible quality and dynamically adjusting it based on network conditions.

At Pinterest, we utilize HLS and DASH for video streaming on iOS and Android platforms, respectively. HLS is supported through iOS's AVPlayer, while DASH is supported by Android's ExoPlayer. Currently, HLS accounts for approximately 70% of video playback sessions on iOS apps, and DASH accounts for around 55% of video sessions on Android.

clients must obtain both the manifest files and partial media files from the CDN through network requests. Manifest files are typically small-size files but provide essential metadata about video streams, including their resolutions, bitrates, etc. In contrast, the actual video and audio content is encoded in the media files. Downloading resources in both file types takes time and is the primary contributor to users' perceived latency.

As illustrated in Figure 1, after receiving video URLs from the API endpoint, players begin the streaming process by downloading the main manifest playlist from the CDN. The content within the main manifest provides URLs for the rendition-level manifest playlists, prompting players to send subsequent requests to download them. Finally, based on network conditions, players download the appropriate media files to start playback. Acquiring manifests is a

prerequisite for downloading media bytes, and reducing this latency leads to faster loading times and a better viewing experience. In the next section, we will share our solution to this problem.

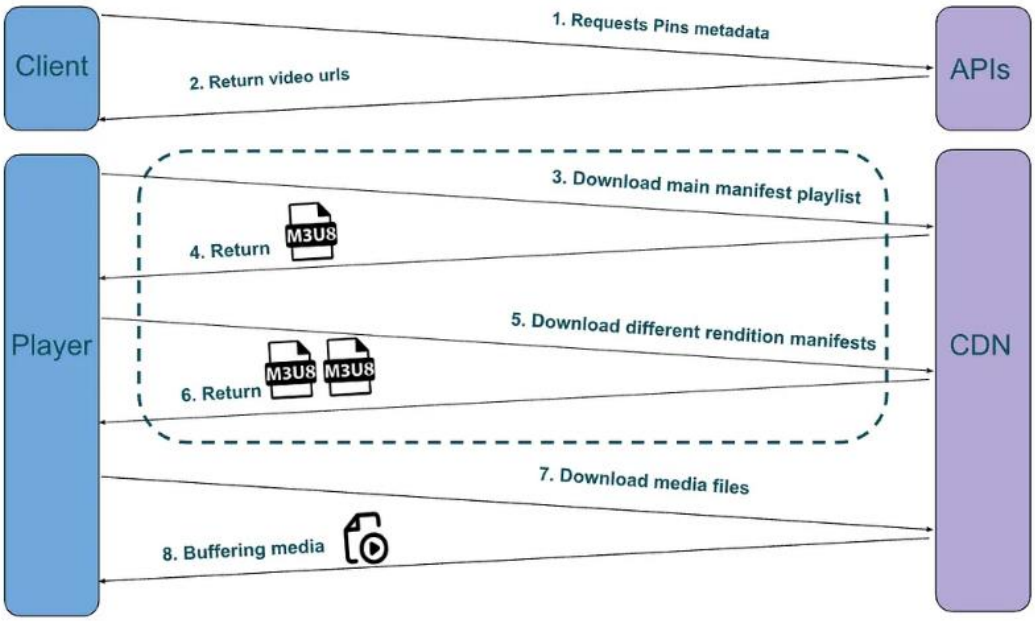

Figure 1. Standard HLS Video Startup Flow (asset source1, source2)

- **Download different bitrate manifests:** After downloading the main manifest, the Player would then download specific manifest files for the chosen video stream (based on network conditions and player capabilities). These segment manifests list the URLs of the individual video and audio segments.

By embedding the manifest file bytes directly into the API response (Step 2), the Player receives all the necessary information about the video structure and available segments upfront. This eliminates the need for the Player to make separate requests to the CDN to download the main manifest and then the segment manifests.

To process API requests, the backend is required to retrieve video manifest files, causing a rise in the overall latency of API responses. This issue is particularly prominent for HLS, where numerous manifest playlists are needed for each video, resulting in a significant increase in latency due to multiple network calls. Although an initial attempt to parallelize network calls provided some relief, the latency regression persisted.

We successfully tackled this issue by incorporating a MemCache layer into the manifest serving process. Memcache provides significantly lower latency than network calls when the cache is hit and is effective for platforms like Pinterest, where popular content is consistently served to various clients, resulting in a high cache hit rate. Following the implementation of Memcache, API overhead was effectively managed.

Why does HLS require way more responses

- For an HLS stream with multiple quality levels (resolutions and bitrates), the player initially downloads a **master playlist** (also sometimes referred to as a top-level playlist or an index playlist).
- This master playlist doesn't contain the URLs of the actual video segments. Instead, it lists the available **variant streams**. Each variant stream represents the same content encoded at a different bitrate and resolution.
- Within the master playlist, each variant stream is associated with a URL pointing to its own **media playlist**.
- Additionally, the master playlist can contain references to other media playlists for alternative audio tracks, subtitles, and closed captions using the `EXT-X-MEDIA` tag.

2. Media Playlists (for each variant stream):

- For each quality level listed in the master playlist, there is a separate **media playlist**.
- These media playlists contain the actual URLs of the individual video and audio segments (typically `.ts` files) for that specific bitrate.
- The media playlist also includes metadata about the segments, such as their duration, encryption information (if any), and whether the playlist is live or static.

3. Adaptive Bitrate Switching:

- The player constantly monitors the user's network conditions (bandwidth, latency).
- Based on these conditions, the player dynamically switches between the different variant streams listed in the master playlist by fetching and using the corresponding media playlist.
- This allows for seamless playback even when network conditions fluctuate. If the bandwidth decreases, the player can switch to a lower bitrate stream to avoid buffering. If the bandwidth improves, it can switch to a higher bitrate stream for better quality.