Spotify Streaming, Packets, What Moves on the Wire

By Michael Allen Mendy.

Abstract

Spotify delivers on-demand audio over HTTPS/TCP using HTTP **range requests** against CDN-hosted track files. Clients fetch **fixed-size byte ranges** (≈512 kB) and manage their own buffering, prefetch, and caching; they **don't** use the segment/manifest model of MPEG-DASH or HLS. This paper sketches Spotify's control/data plane, codecs, transport choices, caching/evolution (e.g., the P2P era), and contrasts that with DASH/HLS. We also outline what packet-level features are observable to a passive network analyst, without describing any DRM bypass techniques, and point to representative research on traffic analysis.

Data plane: what actually moves on the wire

- **Objects & addressing.** Each track is stored as an encoded file on Spotify CDNs. During playback the client issues **HTTP GET with Range: headers** to fetch chunks from a nearby edge. Successful responses are **HTTP 206 Partial Content**. Typical chunk size is **~512 kB**.

  Transport. Plain **TCP over TLS (HTTPS)**. Spotify has publicly documented experiments with Google's **BBR** congestion control to reduce stalls (no QUIC requirement).

CDN. Spotify standardized on **Fastly** for many edge workloads; range-request handling at the CDN/access-point boundary is operationally important (e.g., ensuring Accept-Ranges so 206, not 200, is returned).

Chunking vs codec frames. While codecs produce frames on the order of tens of milliseconds, Spotify **aggregates many frames per HTTP range fetch** (hundreds of kilobytes). The app's buffer and prefetch policy (e.g., when to fetch the next 512 kB) drives smoothness and seeks latency.

Caching implications. Because tracks are stored as large static files, they cache very effectively at the CDN layer. Popular songs see very high cache-hit ratios, which helps Spotify scale to hundreds of millions of active users with predictable CDN economics.

Control plane: auth, prefetch, cache

Prefetching. Spotify's commercial hardware SDK docs include explicit **prefetch** states/errors, reflecting a control layer that schedules next-chunk fetches and offline downloads.

Local cache. Desktop/mobile clients maintain a sizable on-disk cache to reduce startup latency and network usage; users can clear it in-app. (Offline media remains encrypted at rest.)

Then & now: P2P retired. Early desktop clients also sourced audio via **P2P**. Spotify discontinued P2P in **2014** as CDNs scaled.

Formats, bitrates, and encryption

Codecs/bitrates. Spotify documents quality tiers roughly **24, 96, 160, 320 kb/s** on native apps; the **web player** uses **AAC 128/256 kb/s**. (Exact codec can vary by platform.)
Mastering/Ingest. For creators/labels, Spotify prefers **FLAC** ingest and notes ongoing **lossless** testing for eligible listeners (distribution format is independent of ingest).

DRM at a glance. Tracks retrieved from the CDN are **encrypted**; clients obtain decryption keys via authenticated control flows. We avoid details, see open-source clients for high-level architecture only.

How this differs from DASH/HLS

DASH/HLS deliver media as **time-based segments** (often 2–6 s) with a **manifest** (MPD for DASH; M3U8 for HLS) and perform **per-segment ABR** (adaptive bitrate) across multiple representations. **Spotify** instead treats each track as a **single file** and does **byte-range fetching**—no public MPD/M3U8, no time-indexed segment cadence. Bitrate selection is coarse (quality levels) rather than continuous ABR across a ladder during one song.

Packet-level observables (without breaking encryption)

Even with TLS, a passive observer can often see:

Flow structure: TLS handshake to a CDN host (e.g., *.scdn.co/*.akamaized.net depending on region), followed by a bursty pattern of **~512 kB** downloads (range fetches).

Timing & sizes: Range-request cadence, stall/retry behavior, and seek-driven discontinuities (more, smaller ranges around seeks). These are the kinds of side channels used in **traffic-analysis** research on the Spotify web player (classification from packet sizes/timings—not payload).

HTTP semantics: Presence of **206 Partial Content** for ranges vs. **200** if the CDN mis-configures Accept-Ranges.

Practical Implications for Engineers

Latency & resiliency. The size of HTTP range requests and the logic of Spotify's prefetch policy directly shape playback performance. Larger ranges reduce protocol overhead but can increase recovery time when a packet is lost or retransmission occurs. Conversely, smaller ranges yield faster recovery at the expense of more frequent requests and higher control-plane load. Spotify has reported measurable improvements in stall reduction and playback smoothness by adopting Google's **BBR congestion control**, which optimizes congestion windows to keep buffers filled without creating excessive queuing delays.

Edge correctness. Correct CDN behavior is fundamental to Spotify's design. Clients depend on **206 Partial Content** responses for both seeking and efficient buffering. If a CDN node fails to honor Accept-Ranges and instead serves a **200 OK** (full file).

Research & security notes (non-circumvention)

Traffic analysis. Academic/teaching work shows that **song identity** can sometimes be inferred from encrypted traffic **timing/size** patterns on the **web player**. Mitigations include padding, jitter, and cache behavior tuning.

DRM. While open-source clients describe high-level flows, bypassing encryption or redistributing decrypted audio violates terms of service and, in many jurisdictions, the law. This paper deliberately avoids any circumvention details.

Flowchart Comparison

Below is a conceptual flowchart contrasting Spotify's range-based model with DASH/HLS's manifest + segment approach. This highlights the **simplicity of Spotify's pipeline** versus the **manifest-driven ABR logic** in DASH/HLS.

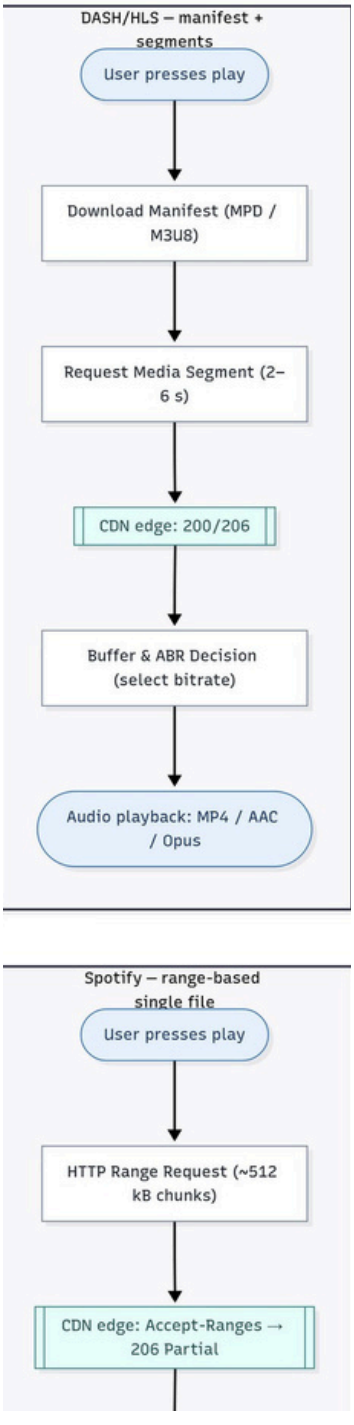

*Figure **1**. Spotify uses fixed-size HTTP **range requests** (~512 kB) to fetch parts of a single file, while DASH/HLS relies on **manifest-driven segments** (2–6 s each) with ABR logic.*

Traffic Analysis of Spotify Range Requests

This is a packet-level view of Spotify traffic reveals a distinctive burst pattern consistent with its range-request architecture. Instead of a continuous stream, the client issues HTTP GET requests with Range headers that fetch ~512 kB of audio data at a time. Each response arrives as a 206 Partial Content, resulting in a sawtooth-shaped profile where large bursts of traffic are followed by idle gaps as the client buffer fills. In a tool like Wireshark, this manifests as periodic spikes in throughput roughly every 2–3 seconds, aligned with buffer management decisions. While the actual payload remains encrypted under TLS, these timing and size characteristics form a recognizable fingerprint of Spotify playback and have been leveraged in research on encrypted traffic classification.

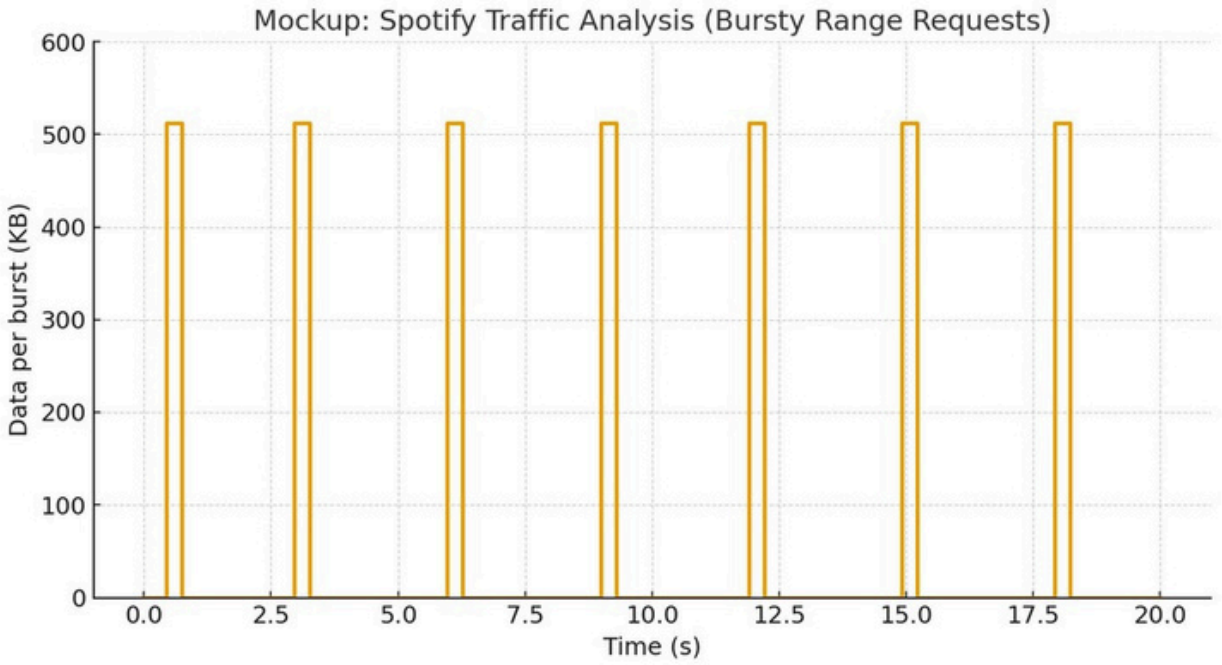

Figure 2. *Mockup of Spotify packet traffic showing bursty download behavior. Each spike corresponds to an HTTP range request retrieving ~512 kB of audio data from the CDN, followed by an idle period as the client buffer plays out. This burst–silence pattern differentiates Spotify's range-based streaming model from the continuous segment flow typical of DASH or HLS.*

Conclusion

Spotify's design emphasizes simplicity, caching efficiency, and fast seeks, trading away fine-grained adaptive bitrate switching in favor of coarse quality tiers. DASH/HLS, in contrast, provide greater ABR flexibility and are optimized for heterogeneous networks, long-form video, and live content.

For engineers, the comparison is instructive: the right protocol depends on the media type, user expectations, and operational scale. Spotify's model works for music because tracks are short, popular, and cache-friendly; DASH/HLS thrive where network adaptation and segment-level control matter most.