Network Optimization

---

1. The Critical Rendering Path (CRP) & The 14KB Rule

The **Critical Rendering Path** is the sequence of steps the browser takes to convert HTML, CSS, and JS into pixels on the screen. To optimize network performance, you must understand how the browser prioritizes these assets.

The 14KB First Packet (TCP Slow Start)

Web servers don't send the entire HTML file at once. Due to the **TCP Slow Start** algorithm, the initial "congestion window" (CWND) is typically **14KB**.

- **The Goal:** Fit your "Above-the-Fold" (ATF) content—the HTML and critical CSS required to render the initial view—into that first 14KB.
- **Impact:** If the critical CSS is inside that first packet, the browser can begin rendering without waiting for a second round trip.

Inlining vs. External Resources

- **Inlining (Critical CSS):** Putting CSS directly in <style> tags in the <head>.
  - *Pro:* Eliminates the network request for CSS; improves First Contentful Paint (FCP).
  - *Con:* It can't be cached independently and increases the size of every HTML document.
- **External:** Standard <link> tags.
  - *Pro:* Better caching and separation of concerns.
  - *Con:* Blocks rendering until the CSS is fetched and parsed.

---

2. Minimizing & Consolidating Network Requests

Every request carries overhead: DNS lookup, TCP handshake, and TLS negotiation.

Modern Trade-offs (HTTP/1.1 vs. HTTP/2+)

- **HTTP/1.1 Thinking:** We used to combine everything (CSS/JS bundling, Image Sprites) because browsers could only handle ~6 concurrent connections per domain.
- **HTTP/2+ Thinking:** With **Multiplexing**, having 50 small files is no longer a performance killer. However, bundling still helps with **compression efficiency** (Gzip/Brotli work better on larger files).
- **Base64 Assets:** Encoding small icons into CSS/HTML.
  - *Decision:* Use only for tiny icons (<1KB). Base64 increases file size by ~33% and

prevents the browser from downloading the image in parallel with CSS parsing.

---

3. JavaScript Loading Strategies

JavaScript is "parser-blocking" by default. If the browser sees a <script> tag, it stops building the DOM until the script is fetched and executed.

Strategy	Behavior	Best Use Case
Normal	Block parsing → Fetch → Execute → Resume parsing	Tiny, essential inline scripts.
Async	Fetch in parallel → Execute immediately (blocks parsing)	Independent scripts (Ads, Analytics).
Defer	Fetch in parallel → Execute after DOM is fully parsed	**Standard for modern apps.** Maintains order.

---

4. Avoiding Redirects & HSTS

A redirect (e.g., http://example.com → https://example.com) is a performance "silent killer," especially on mobile 4G/5G networks where latency is high.

- **HSTS (HTTP Strict Transport Security):** A header that tells the browser: "For the next year, only talk to me via HTTPS." The browser will then perform the redirect **internally** before hitting the wire, saving a full round trip.

---

5. Resource Hinting: Helping the Browser Peer Ahead

Resource hints are "clues" we give the browser to prioritize certain connections or files.

- **dns-prefetch:** Resolves a domain name before a request is made. (Low cost, good for 3rd party domains).
- **preconnect:** DNS + TCP + TLS handshake. Use for critical 3rd party origins (e.g., Google Fonts, CDN).
- **preload:** Tells the browser to download a high-priority resource *now* because it's needed for the current page (e.g., a late-discovered LCP image or a font).
- **prefetch:** Tells the browser to download a resource in the background because it might

be needed for the **next** page navigation.
- **prerender:** (Use with caution) Downloads the next page and renders it in a hidden tab.

---

6. Fetch Priority (fetchpriority)

Standard hints only go so far. fetchpriority is an attribute you can add to tags (<img>, <script>, <link>) to signal relative importance.

- **Use Case:** Setting fetchpriority="high" on your **Largest Contentful Paint (LCP)** image ensures the browser prioritizes it over other images that might be discovered earlier in the HTML.

---

7. Early Hints (HTTP 103)

Early Hints fill the "Server Think Time" gap.

1. Browser requests HTML.
2. Server is busy querying the DB (takes 200ms).
3. **Normally:** Browser sits idle.
4. **With Early Hints:** Server sends a 103 status code with Link: <...>; rel=preload headers *immediately*. The browser starts downloading CSS/JS while the server is still generating the HTML.

---

8. HTTP Protocol Upgrades (H1 vs H2 vs H3)

- **HTTP/2:** Introduced **Multiplexing** (sending many files over one connection) and **Header Compression**. It solved the "6-connection limit" but still suffers from **TCP Head-of-Line (HOL) Blocking**—if one packet is lost, the whole stream waits.
- **HTTP/3 (QUIC):** Moves from TCP to **UDP**. It solves HOL blocking at the transport layer. If one packet is lost, only that specific stream is affected, not the whole connection. This is a game-changer for unstable mobile networks.

---

9. Compression: Gzip vs. Brotli

- **Brotli:** Generally provides **15–20% better compression** than Gzip for text assets (HTML, CSS, JS).
- **Strategy:** Use Brotli for static assets pre-compressed at build time. Use Gzip for dynamic content if the server CPU overhead for Brotli is too high.

10. HTTP Caching Strategies

In a system design interview, demonstrate you know the difference between **validation** and **expiration**.

- **Cache-Control: immutable:** Used for versioned/hashed assets (e.g., app.af32c.js). Tells the browser "this file will never change; don't even check with the server until the expiry date."
- **ETag / Last-Modified:** "Weak" caching. The browser asks the server, "Is this still valid?" If yes, the server sends a **304 Not Modified**, saving the body payload but still costing a round trip.

---

11. Service Worker Caching

Service workers act as a proxy between the browser and the network.

- **Stale-While-Revalidate:** Serve the cached version immediately (fast!) but fetch the update in the background for the next time. Best for UI shells.
- **Cache-First:** Best for static assets (images, fonts).
- **Network-First:** Best for data that must be fresh (account balances), falling back to cache only if offline.

---

12. Additional System Design Techniques

- **CDN (Content Delivery Network):** Moves assets to "Edge" nodes physically closer to the user, reducing the physical distance data must travel.
- **Image Optimization:** Use the <picture> tag for **Responsive Images** (serving smaller sizes to mobile) and modern formats like **WebP** or **AVIF**.
- **Code Splitting:** Using import() to break the JS bundle into smaller chunks so users only download the code needed for the current route.
- **Connection Reuse:** Ensure Keep-Alive is enabled so connections stay open for multiple requests.

---

13. Interview Perspective: How to Pitch This

When asked to "Optimize a slow web app" in an interview, follow this mental flow:

1. **Measurement:** Mention **Core Web Vitals**. "I'd look at LCP to see if it's a network/image issue, and FCP for CRP issues."
2. **The "Big Rocks":** Start with **HTTP/2+**, **Compression**, and a **CDN**. These are

high-impact, low-effort.
3. **The Critical Path:** Discuss inlining critical CSS and using defer for scripts to get pixels on the screen faster.
4. **Intelligent Loading:** Mention preload for the LCP image and prefetch for the next logical page the user might visit.
5. **Offline/PWA:** Mention Service Workers for instant subsequent loads.

Trade-off Example: > "While inlining CSS improves FCP by saving a request, I'd only inline the 'Above-the-Fold' CSS because inlining everything bloats the HTML and prevents us from leveraging the browser's cache for that CSS on other pages."