**What is Latency and How to Reduce it?**

Latency is the time between a user taking an action—like clicking a button or loading a webpage—and receiving a response from the system.

In simple terms, latency is the time delay between:

* When a user makes a request
* When they receive the response

[[A diagram of a number of meters

AI-generated content may be incorrect.](https://substackcdn.com/image/fetch/$s_!Mwnv!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F23aeaa06-f9d2-4986-8062-82ee094db6f1_591x237.png)](https://substackcdn.com/image/fetch/$s_!Mwnv!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F23aeaa06-f9d2-4986-8062-82ee094db6f1_591x237.png" \t "_blank)

Lower latency means faster responses and a better user experience.

In this article, we’ll explore the causes of high latency and how to reduce latency at different layers of your system.

**What Causes High Latency?**

* **Geographical Distance:** The farther a user is from your server, the longer data takes to travel. Even though data moves close to the speed of light, crossing thousands of miles takes more time than traveling a few hundred.
* **Overloaded Servers:** When a server receives more requests than it can handle, it slows down. This overload can happen due to sudden traffic spikes, inefficient resource usage, or inadequate server capacity. As servers struggle to keep up, each request takes longer to process, and latency spikes.
* **Slow Database:** If your database queries take too long—due to large tables, missing indexes, or poorly written queries—responses take longer.
* **Inefficient Code Paths:** Sometimes latency hides in the application’s code. Overly complex code, unnecessary calculations, and complicated logic can introduce small delays that add up.
* **Network Congestion:** Heavy network traffic, limited bandwidth, and busy intermediaries between the user and your server can slow requests. Employing techniques like load balancing across different network paths, using faster protocols (like HTTP/2 or HTTP/3), and minimizing payload sizes can help reduce the impact of congestion.

**Types of Latency (from AlgoMaster’s “Top 15 Strategies”)**

1. **Network Latency**  
   Delay caused by data transmission across the network—from your device to the server and back. Causes include physical distance, DNS resolution time, TCP/TLS handshakes, inefficient routing, and security inspections via proxies or firewalls.
2. **Application Latency**  
   The time your backend application takes to handle a request. This includes receiving the request, processing logic, making calls to other services or databases, and generating a response.
3. **Database Latency**  
   The round-trip delay of querying the database, having it execute the query, and returning the result. Slow database performance can bottleneck the entire system.
4. **Client-side Latency**  
   Even after the response reaches your device, there’s still time needed to render and display content (e.g. browser rendering, UI updates) before the user sees anything useful.

**Why Does Latency Matter?**

Higher latency leads to sluggish, clunky user experiences. Even tiny delays can hurt engagement:

* **Amazon** estimates that every 1-second increase in latency could cost them **$1.6 billion** in annual sales.

Lower latency = faster, smoother interactions = happier users = better business outcomes.

**15 Top Strategies to Reduce Latency (from AlgoMaster)**

Here are practical ways to tackle latency across your system:

**1–4: For Network Latency**

* **Reduce physical distance** between client and server (e.g., through geographic server distribution or edge servers).
* **Improve DNS resolution** by choosing fast, reliable DNS providers or adding caching.
* **Optimize connection setup**: minimize TCP handshakes and TLS negotiation overhead.
* **Avoid inefficient routing** and reduce overhead from proxies/firewalls.

**5–8: For Application Latency**

* **Simplify business logic** and eliminate unnecessary computation.
* **Use asynchronous/concurrent patterns** instead of blocking operations.
* **Limit chained service calls**, especially in microservice architectures.
* **Handle errors and retries smartly** to avoid long timeouts and delays.

**9–11: For Database Latency**

* Optimize queries and indexes for faster execution.
* Implement **caching layers** (e.g., using Redis) to reduce database hits.
* Use efficient caching strategies—like cache-aside, write-through, write-around, or write-back—depending on your consistency and performance needs.

**12–15: For Client-side Latency**

* **Minimize rendering time** (e.g., optimize front-end code and assets).
* **Bundle and lazy-load resources** to reduce initial payloads.
* **Leverage browser caching** wisely.
* **Use persistent connections** (e.g., in Redis) to avoid handshake overhead and reduce latency on repeated requests.

**Summary Table**

| **Latency Type** | **What It Is** | **Key Strategy to Reduce It** |
| --- | --- | --- |
| **Network** | Delay from sending request to server and back | Reduce hops, optimize DNS, handshake mitigation, server proximity |
| **Application** | Time spent computing logic, calling services | Refactor logic, async calls, reduce dependencies |
| **Database** | Time taken for query execution and result delivery | Use caching, optimize queries, choose effective strategies |
| **Client-side** | Time to render and show response on user device | Optimize rendering, caching, persistent connections |

**15 interview question from low latency**

**1. What is latency, and how does it differ from throughput?**

* **Latency**: The time delay between sending a request and receiving a response.
* **Throughput**: The total amount of data processed in a given time period.
* **Analogy**: Latency is like delivery time for a single package; throughput is how many packages are delivered per hour.

**2. Explain the different types of latency in a distributed system.**

* **Network latency**: Time taken for data to travel over the network.
* **Application latency**: Time the server spends processing the request.
* **Database latency**: Time taken to retrieve or update data.
* **Client-side latency**: Time to render and display data in the UI.

**3. Why is low latency important in HFT, gaming, or real-time analytics?**

* In **HFT**, milliseconds can mean millions in gains/losses.
* In **gaming**, low latency ensures smooth gameplay and responsiveness.
* In **real-time analytics**, timely data processing is critical for decision-making (e.g., fraud detection).

**4. How does the TCP three-way handshake contribute to latency?**

* TCP handshake requires **three messages** (SYN → SYN-ACK → ACK) before data transfer starts.
* Each round trip adds latency, especially over long distances.
* Solutions: **TCP Fast Open**, persistent connections, or switching to UDP when reliability isn’t required.

**5. What’s the difference between latency and jitter, and why does jitter matter?**

* **Latency**: Delay in a single transmission.
* **Jitter**: Variation in latency between multiple transmissions.
* **Why it matters**: Inconsistent timing (jitter) causes problems in real-time audio/video, making playback choppy.

**6. What strategies can you use to reduce network latency over long distances?**

* Use **CDNs** or edge computing to serve content closer to users.
* Optimize routing (BGP tuning, peering agreements).
* Reduce data size (compression, minification).
* Switch to **HTTP/3** (QUIC) for faster handshake and multiplexing.

**7. How do DNS lookups impact latency, and how can you optimize them?**

* Each DNS lookup adds ~20–120 ms.
* Optimizations:
  + Use **fast DNS resolvers** (Cloudflare 1.1.1.1, Google 8.8.8.8).
  + Enable DNS caching (client and server).
  + Reduce external domains your app depends on.

**8. Explain how HTTP/2 and HTTP/3 can reduce latency compared to HTTP/1.1.**

* **HTTP/2**: Multiplexing, header compression, server push — reduces blocking.
* **HTTP/3**: Uses QUIC over UDP — faster handshake, built-in encryption, better performance in lossy networks.

**9. How would you profile and identify latency bottlenecks in a backend application?**

* Use **profiling tools** (e.g., Perf, dotTrace, YourKit).
* Measure latency at:
  + Network entry points (API Gateway logs).
  + Service logic execution times.
  + Database queries (slow query logs).
* Add distributed tracing (e.g., OpenTelemetry, Jaeger).

**10. What are some ways to make I/O operations non-blocking to reduce latency?**

* Use **asynchronous programming** (async/await in .NET, Java NIO).
* Use **message queues** for decoupling (RabbitMQ, Kafka).
* Batch I/O requests instead of sending them one by one.

**11. How can you minimize latency in a microservices architecture?**

* Reduce **service-to-service calls** (combine logic where needed).
* Use **parallel calls** instead of sequential.
* Add **caching** for frequent requests.
* Optimize **serialization/deserialization** formats (e.g., Protobuf vs JSON).

**12. How does database indexing reduce latency, and what trade-offs does it introduce?**

* **Indexes** speed up lookups by avoiding full table scans.
* Trade-offs:
  + More disk space usage.
  + Slower writes/updates (index needs updating).
  + Requires careful choice of indexed columns.

**13. What is the cache-aside pattern, and how does it help reduce latency?**

* **Cache-aside**:
  1. App checks cache first.
  2. If miss → fetch from DB → store in cache.
  3. Reduces latency by serving data from faster in-memory store (Redis, Memcached).

**14. How would you handle cache misses in a low-latency environment without significantly impacting performance?**

* Use **stale-while-revalidate**: Serve old data while fetching new in the background.
* Pre-warm cache with frequent queries.
* Use **tiered caching** (local + distributed).

**15. Imagine your trading app starts seeing a 5–10 ms increase in order execution latency. How would you debug and resolve it?**

* **Step 1**: Check if issue is network or application-related (latency tracing).
* **Step 2**: Inspect recent code/config changes.
* **Step 3**: Profile execution path for bottlenecks (logging, tracing).
* **Step 4**: Test against historical latency baselines.
* **Step 5**: Optimize identified bottlenecks (DB, network routes, CPU usage).