Engineer

Latency

the wait between asking and the first of the answer

What it is

Latency is how long a system takes to start responding: the time from a request leaving the client to the first byte of the reply coming back. Engineers also call it delay, lag or, for a network path measured there and back, round-trip time (RTT); in web performance the same idea goes by time to first byte (TTFB). Response time is its larger sibling, running to the last byte of the answer.

It is easy to confuse with throughput and bandwidth, and the three answer different questions. Bandwidth is how many bits a link can carry per second. Throughput is how much work a system completes per second. Latency is how long one piece of work waits. A nightly batch job can have excellent throughput and terrible latency, and a chat server the reverse.

Every millisecond of it comes from one of a few places:

- **propagation**, the signal crossing the distance: light in optical fibre covers roughly 200 km per millisecond, so a 1,000 km path costs at least 10 ms there and back before any software runs;
- **transmission**, putting the bytes on the wire, which matters for large payloads on slow links;
- **queueing**, waiting behind other work in a router buffer, a thread pool, a connection pool or a lock, the one component that grows with load;
- **processing**, the actual work: the query, the template, the encryption;
- **protocol round trips**, the conversation before the first useful byte. A new HTTPS connection over TCP needs a TCP handshake and a TLS handshake first, and TLS 1.3 cut its full handshake to a single round trip.

The IETF's IP Performance Metrics group pinned the network version down in RFC 2681: round-trip delay runs from the first bit of a packet sent to the last bit of the reply received. Two of its choices carry over to any service. A reply that never arrives is counted as undefined, informally infinite, rather than quietly left out of the sample. And the statistics it describes are percentiles, the median and the minimum. It also observes that the minimum reflects propagation and transmission alone, while everything above the minimum indicates congestion, a split that holds for an API as well as for a path between routers.

For a single web request, curl will break the wait into its stages:

```
curl -s -o /dev/null "$URL" -w '
dns        %{time_namelookup}
connect    %{time_connect}
tls        %{time_appconnect}
first byte %{time_starttransfer}
total      %{time_total}
'
```

Each figure is seconds since the start, so the gaps between them are the costs: looking up the name, opening the connection, the TLS handshake, and then the server's own thinking time up to the first byte.

Use the concept in a decision

Put "What is latency?" to work in a real decision. We can help relate the concept to the system you are building or buying. Monitoring and alerts that speak up while a fault is still small, and the figures that show how often it stays that way.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Built the company's first observability dashboard — the company's first view of its own systems

Built the company's first observability dashboard Get in touch →

Why it matters

Latency is what a user feels; throughput is what the server bill reflects. When a page is slow, the decision is where to spend. The five components above need different fixes; four of them show why buying hardware is not a general remedy. Distance is fixed by moving closer: a cache, a CDN, a region near the users. Round trips are fixed by having fewer of them: reusing connections, batching queries, fetching in parallel. Processing is fixed by doing less work: an index, a better

query plan. Adding capacity can reduce queueing; faster processors, more memory or better I/O may also reduce processing time when those resources are the bottleneck. A larger server is not automatically a faster one: measure where the time goes before choosing hardware.

Averages make the problem look smaller than it is, because users meet the tail. A page that calls ten back-end services in parallel waits for the slowest. If each call independently has a 1% chance of being slow, then $1 - 0.99^{10}$, about 9.6% of page loads, hit at least one slow call. A rare delay in the parts becomes a common one in the whole.

That is why serious targets are written in percentiles, for example 99% of requests answered within 300 ms over 30 days, and why the 99th percentile is usually the number worth watching. The median says how the service feels on a good day; the high percentiles say how it feels to the customers who are about to leave.

What it looks like in practice

On a geographical map application, latency in rendering map layers and querying location data was hurting both users and the reliability of the backend. The work focused on spatial query processing in a backend that used a legacy Microsoft SQL Server database. Moving to PostgreSQL with PostGIS cut average response time from 2.5 seconds to under 250 milliseconds, a 10x improvement in spatial query performance. The work behind it was GiST and SP-GiST indexes on the spatial columns, more than 50 million records migrated, and configuration tuned for concurrent access; backend CPU load dropped by 65%.

Latency also caps throughput where one row is contended. A note on selling the last ticket exactly once puts the ceiling for a single item at roughly the sales one lock carries divided by how long the lock is held. Anything slow inside the lock lowers that ceiling: a payment call made while holding it brings it down to about two sales a second. Even a safety measure has a price here. A synchronous standby in PostgreSQL adds latency to every commit, and that lengthens exactly the lock hold time the ceiling divides by.

And to find where the time goes, the first-party monitoring on one platform records API latency in its own database and traces requests over OpenTelemetry, so a slow request can be followed across the frontend, the API and the database rather than guessed at.

Where it goes wrong

The average hides the tail. A mean of 100 ms is compatible with one request in fifty taking three seconds. Record latency as a histogram and read its percentiles. Then mind how they combine: the 99th percentiles of five servers cannot be averaged into the fleet's 99th percentile. Prometheus makes the point in its own documentation, where quantiles computed by a summary cannot be aggregated, while histogram buckets can be summed first and the quantile taken afterwards.

The timer starts in the wrong place. A server that measures from the moment its handler begins misses the time a request spent in the load balancer, the accept queue or the wait for a pooled connection. Load generators that wait for each answer before sending the next request send fewer requests exactly when the system stalls, so the stall barely shows in the results. Measure from the client as well, and let the server report its share in a standard header:

```
Server-Timing: db;dur=53, app;dur=47.2
```

The conversation is too chatty. A page that runs one query per row of a list pays one round trip per row. With the database in another region at 30 ms a round trip, a hundred rows cost three seconds before any query does real work. One query with a join, or one batched call, pays the distance once.

Retries multiply the load. A client that gives up after one second and retries three times turns one slow server into four times the traffic, which makes it slower still. Give each request an overall deadline, back off exponentially with random jitter between attempts, and cap the number of retries.

Further reading

- RFC 2681: A Round-trip Delay Metric for IPPM — round-trip delay defined to the bit, with lost replies and percentiles handled explicitly.
- RFC 7679: A One-Way Delay Metric for IP Performance Metrics — the one-way counterpart, published as an Internet Standard.
- Server Timing — the W3C header that lets a server tell the browser where its time went.
- Histograms and summaries — why quantiles from a summary cannot be combined and histogram buckets can.

- RFC 8446: TLS 1.3 — the handshake design, including the round trip it removes.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is latency?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/latency/