

Q1 Why Web Pages Feel Slow Slides 24–25

Names: _____

A browser loads a page made of a **base HTML file plus 6 embedded objects** referenced within its code (images, scripts, stylesheets). The round-trip time between browser and server is **RTT = 0.1 s**. Every object here is very small, so **ignore transmission time entirely** — the only cost that matters is round trips.

Under **non-persistent HTTP 1.0**, fetching one object costs **two** round trips: one to open the TCP connection, one to send the request and receive the first bytes back. In **persistent HTTP 1.1**, a single connection can be reused to send multiple requests and responses, with only one RTT for the initial connection setup.

1. A **sequential** browser fetches all objects one at a time, opening and closing a connection for each. How many round trips is that, and how long does the page take to fully load?

Round trips: _____ Total time: _____ s

2. Generalize your count. Fetched this same way, how long would a page with a base file and **30** embedded objects take?

Round trips: _____ Total time: _____ s

3. Now the browser opens up to **4 parallel connections** at a time, but still uses non-persistent HTTP. How long will it take to load the full page and 6 embedded objects? *Hint: the answer is not 0.4 s — why not?*

Round trips: _____ Total time: _____ s

4. Finally, the browser opens **one persistent connection** and pipelines: after receiving the HTML it sends all followup requests back-to-back without waiting for each response. How long will it take to fully load the page in the best case?

Round trips: _____ Total time: _____ s

5. Nothing physical changed across the three scenarios—same link, same distance, same server—yet we got different performance for each. Parts 1–3 ran the **same protocol** and differed only in how the client chose to use it, while part 4 needed a new capability the protocol itself had to provide. System design always has trade-offs; what is the cost to make part 4 faster and who pays it?

Discuss with your partner

Opening 4 connections instead of 1 made *your* page faster. What does it do to the server, and to everyone else sharing the link? If it is that easy a win, why not open 50? If using pipelining, what happens if the first embedded object is a large video file?

We have not covered this protocol, but everything you need is below. The transcript shows a complete session with a mail server. C: = client → server message, S: = server → client message. Try to interpret the exchange below.

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

1. The server answers 250 three times, each time followed by a different sentence. Which part of a reply is a program meant to act on, and which part is there for a human debugging? What could go wrong in a client that decided what to do by matching the words?
2. HTTP was described as a **stateless** protocol, because subsequent requests (even when pipelining) do not depend on prior ones. This protocol is **stateful**: what information must the server remember as the session goes on, and why does it need it? Does the order of the client's messages matter — what should happen if RCPT TO arrives before any MAIL FROM?
3. Alice writes a long email in which **one of the lines** consists of exactly one period, with nothing else on that line. Trace what a server receives and what happens to her message. If you email your partner a line that is just a "." — will it work? What might be happening differently?
4. Propose a protocol change that avoids the problem of Alice's message being mistaken for a control signal. *Hint: look at how HTTP produces a response to a client.* How does this change impact the client?

Discuss with your partner

Nothing in this session was verified for security. The client announced HELO crepes.fr and MAIL FROM: <alice@crepes.fr>, and the server simply believed both. What stops you from connecting to this server and sending mail as anyone you like? What would the protocol need in order to stop you — and why do you think it was not built that way?