Computer Network I

Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

(riccardo.caccavale@unina.it)

Application Layer
Web and HTTP

- Until the early 1990s the Internet was used **primarily by researchers**, **academics**, and university **students** to log in to remote hosts, to transfer files from local hosts to remote hosts and vice versa, to receive and send news, and to receive and send electronic mail.

- Although these applications were (and continue to be) extremely useful, the Internet was essentially unknown outside of the academic and research communities.

- In the early 1990s the **World Wide Web** [Berners-Lee 1994] was the **first Internet application** that caught the general public.

Application Layer
Web and HTTP

- The **World Wide Web** (WWW or simply Web) is a **collection of information** such as documents, images, video, audio, etc. that can be accessed over the Internet according to a specific protocol called **HyperText Transfer Protocol** (HTTP).

- HTTP-based communication is typically client-server:
  - A **client program** that translates users' requests into HTTP messages. This is implemented into the **web browsers** (Chrome, Firefox, Edge, Safari, etc.).
  - A **server program** that executes the **HTTP request** and returns a **HTTP response**.

- HTTP mainly defines the **structure of messages** and how the client and server should exchange such messages.

- The Web and its protocols **serve as a platform** for web-based applications such as YouTube, Web-based e-mail (e.g., Gmail), and most mobile Internet applications, including Social Networks.

Application Layer
URL

- The information on the web are called **resources** (or objects) and are identified by a **Uniform Resource Locator** (URL) which is a string composed as follows:

  [protocol]://[usrinfo@][host][:port][/path][?query][#fragment]

- Where:
  - [protocol] is the **protocol** to use while accessing the resource (HTTP, HTTPS, FTP, etc.).
  - [usrinfo] (optional) are **user's information** such as username and password followed by a @ (e.g., username:password@). This is **mostly deprecated** due to security issues.
  - [host] is the **name or the IP** address of the server.
  - [port] (optional) is the **port** to use (often inferred from the protocol).
  - [path] is the **path of the resource** within the server (e.g., /path/to/resource).
  - [query] is preceded by the ? and specifies **possible requests**.
  - [fragment] preceded by the # **identifies an element** in the resource (e.g., a form).

Application Layer
URL

- Most Web pages consist of a **base HTML** (HyperText Markup Language) file and several references to additional **objects** (images, videos, Java applets, etc.).
  - Example: a Web page containing a HTML text and five JPEG images **has six objects**: the base HTML file plus the five images.

- An example of URL is:

  http://www.someSchool.edu/someDepartment/picture1.gif

- Where:
  - *http* is the **protocol**.
  - *www.someSchool.edu* is the **hostname**.
  - */someDepartment/picture1.gif* is the **path** to the object.

Application Layer

URL

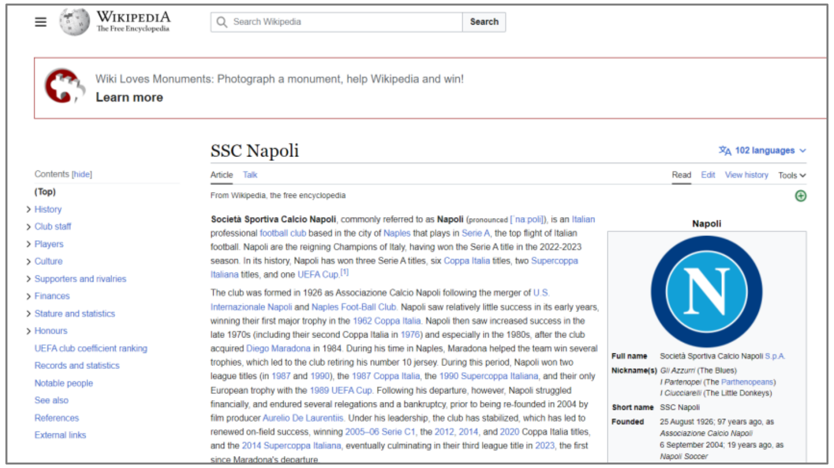
- Real example of URL query:

  https://en.wikipedia.org/w/index.php?title=SSC_Napoli

- Where:
  - *https* is the **protocol**.
  - *en.wikipedia.org* is the **hostname**.
  - */w/index.php* is the **path** to the page.
  - *?title=SSC_Napoli* is the **query**.

- In this case, it is the same as asking for the page:

  https://en.wikipedia.org/wiki/SSC_Napoli



Wiki Loves Monuments: Photograph a monument, help Wikipedia and win!
Learn more

SSC Napoli

102 languages

Article Talk

Read Edit View history Tools

Contents [hide]

(Top)
- History
- Club staff
- Players
- Culture
- Supporters and rivalries
- Finances
- Stature and statistics
- Honours
- UEFA club coefficient ranking
- Records and statistics
- Notable people
- See also
- References
- External links

From Wikipedia, the free encyclopedia

Società Sportiva Calcio Napoli, commonly referred to as **Napoli** (pronounced [ˈnaːpoli]), is an Italian professional football club based in the city of Naples that plays in Serie A, the top flight of Italian football. Napoli are the reigning Champions of Italy, having won the Serie A title in the 2022-2023 season. In its history, Napoli has won three Serie A titles, six Coppa Italia titles, two Supercoppa Italiana titles, and one UEFA Cup.[1]

The club was formed in 1926 as Associazione Calcio Napoli following the merger of U.S. Internazionale Napoli and Naples Foot-Ball Club. Napoli saw relatively little success in its early years, winning their first major trophy in the 1962 Coppa Italia. Napoli then saw increased success in the late 1970s (including their second Coppa Italia in 1976) and especially in the 1980s, after the club acquired Diego Maradona in 1984. During his time in Naples, Maradona helped the team win several trophies, which led to the club retiring his number 10 jersey. During this period, Napoli won two league titles (in 1987 and 1990), the 1987 Coppa Italia, the 1990 Supercoppa Italiana, and their only European trophy with the 1989 UEFA Cup. Following his departure, however, Napoli struggled financially, and endured several relegations and a bankruptcy, prior to being re-founded in 2004 by film producer Aurelio De Laurentiis. Under his leadership, the club has stabilized, which has led to renewed on-field success, winning 2005–06 Serie C1, the 2012, 2014, and 2020 Coppa Italia titles, and the 2014 Supercoppa Italiana, eventually culminating in their third league title in 2023, the first since Maradona's departure.

Napoli

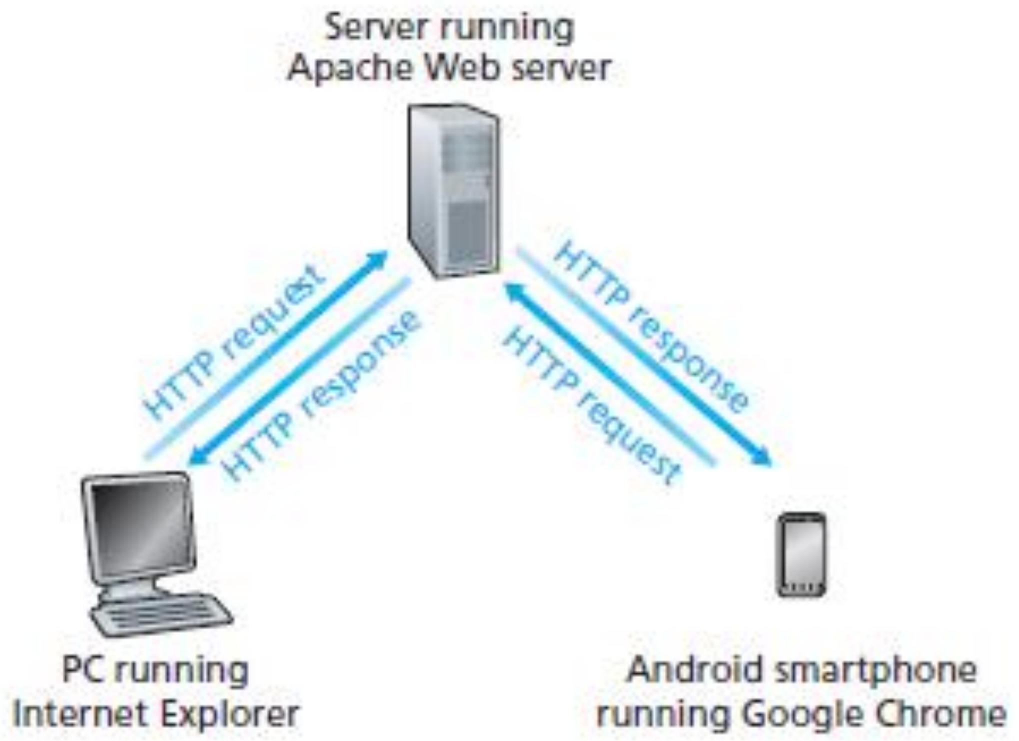
Full name	Società Sportiva Calcio Napoli S.p.A.
Nickname(s)	*Gli Azzurri* (The Blues) *I Partenopei* (The Parthenopeans) *I Ciucciarelli* (The Little Donkeys)
Short name	SSC Napoli
Founded	25 August 1926; 97 years ago, as *Associazione Calcio Napoli* 6 September 2004; 19 years ago, as *Napoli Soccer*

en.wikipedia.org/wiki/SSC_Napoli

Application Layer

HTTP

- HTTP defines how Web clients request objects (e.g., Web pages) from Web servers and how servers transfer them to clients.
- When a user requests an object (for example, a Web page by clicking on a hyperlink), the **browser** sends HTTP request messages for the **objects in the page** to the server.
- The server receives the requests and responds with HTTP response messages that contain the **objects**.


Server running Apache Web server
HTTP request
HTTP response
HTTP response
HTTP request
PC running Internet Explorer
Android smartphone running Google Chrome

Application Layer
HTTP

- HTTP uses **TCP as its underlying transport protocol** (rather than UDP).

- The HTTP client **first initiates a TCP connection** with the server. Once the connection is established, the browser and the server processes access TCP through their **socket interfaces**.
  - On the client side the socket interface is the door between the client process and the TCP connection.
  - On the server side it is the door between the server process and the TCP connection.

- The client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface. Similarly, the HTTP server receives request messages from its socket interface and sends response messages into its socket interface.

Application Layer
HTTP

- One of the reasons why HTTP uses TCP as a transport protocol is that **several useful services are provided by TCP**, most of all **reliable data transfer**.

- Recall reliability: TCP guarantees HTTP request/response messages to eventually arrive intact at the destination (if possible).

- Here we see one of the great advantages of a layered architecture: HTTP-based applications need not worry about reachability, data loss, etc.

- HTTP applications can be **simpler** (both logically and computationally), by **delegating** most of the work to the lower-level stack.

Application Layer

HTTP: Statelessness

- Simplicity is very important for HTTP-based servers dealing with **large number of requests per second** (servers run *around 1000 HTTP requests per second*, while whole google search infrastructure receives *around 100000 queries per second*).

- HTTP applications are typically **stateless**: the server does not maintain any information about the interaction (sequence of requests/responses) with a specific client.
  - Example: if a client asks for the **same object twice** in a row, the server does not consider this request redundant, but **it just resends the object**, as it has completely forgotten the past.

- Not all applications are stateless, several applications are instead **stateful**.

Application Layer

HTTP: Persistent and Non-persistent Connections

- In many applications (such as HTTP-based), the client and server may communicate for an **extended period of time**, by sending **multiple pairs of request-response**.
  - This flow of messages may be made back-to-back, periodically (at regular intervals), or intermittently.

- When relying on a TCP connection, we can have:
  - **Persistent connections**: all requests and their corresponding responses are sent over the same TCP connection.
  - **Non-persistent connections**: for each request-response pair a new TCP connection is established.

- By default, todays HTTP applications use **persistent connections**, while HTTP clients and servers can be configured to use non-persistent connections if needed. Early versions of HTTP (HTTP 1.0) used non-persistent connections by default.

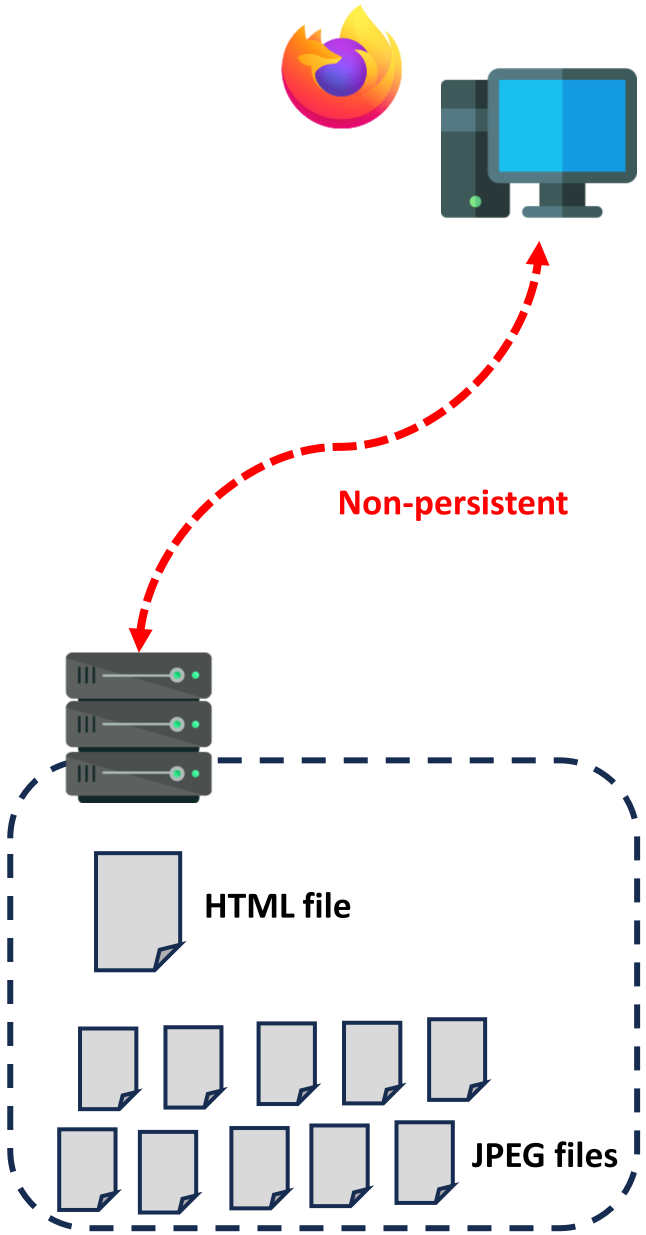
Application Layer

HTTP: Non-persistent Connection (Example)

- In a non-persistent connection TCP connection is closed every time **a request is served** (e.g., an object is sent).

- Example: assume to ask for a Web page consisting of a base HTML file and 10 JPEG images (all 11 of these objects reside on the same server).

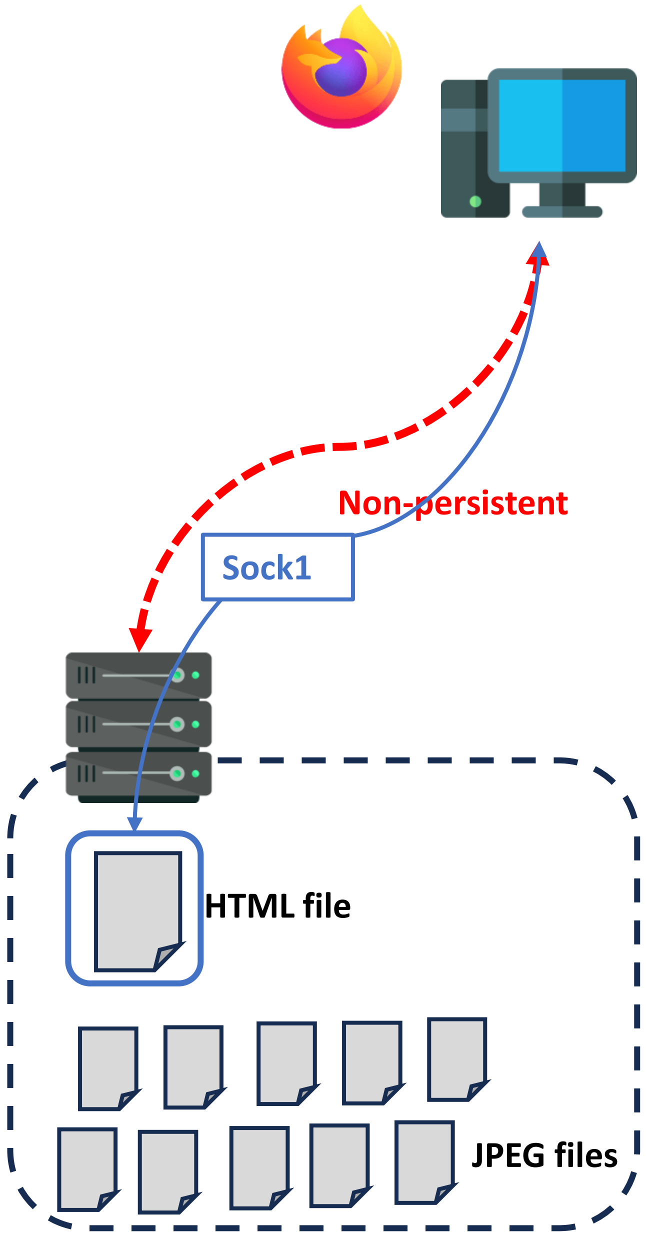
- The URL for the base HTML file is:
  - http://www.someSchool.edu/someDepartment/home.index



Application Layer

HTTP: Non-persistent Connection (Example)

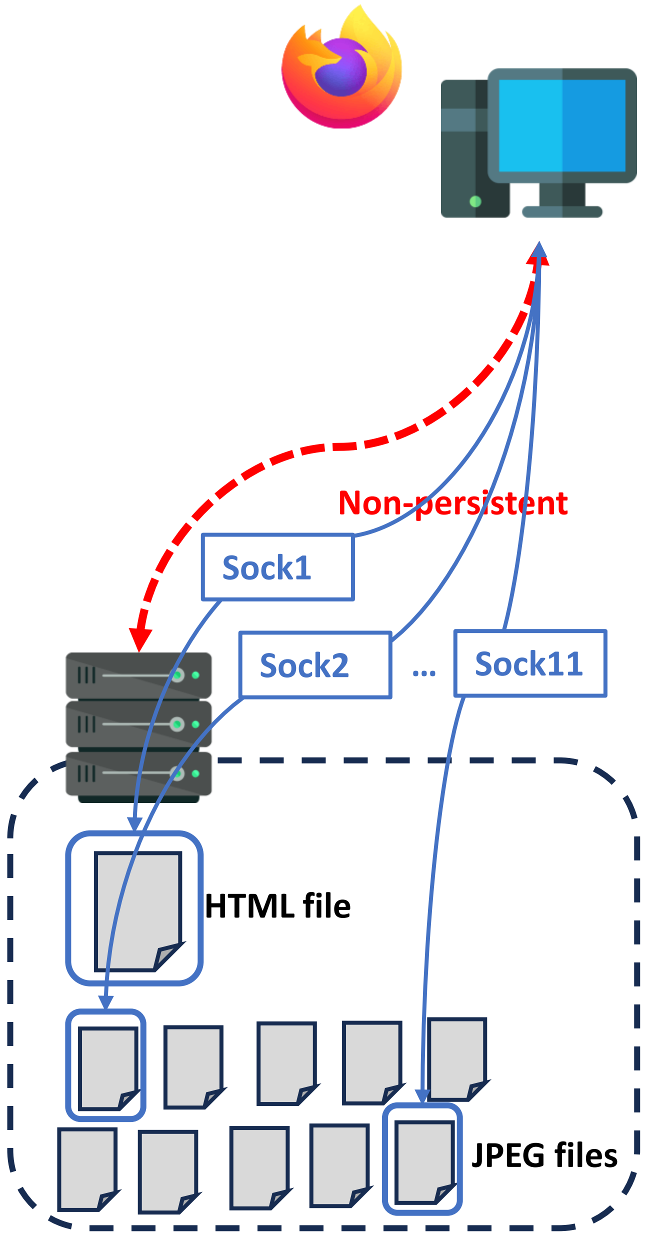
- The connection takes place as follows:
  1. The HTTP **client process initiates a TCP connection** to the server www.someSchool.edu on port number 80 (default HTTP port), we will have two sockets: one at the client and one at the server.
  2. The HTTP **client sends an HTTP request message** to the server via its socket including the path name /someDepartment/home.index.
  3. The HTTP **server process receives the request message** via its socket, **retrieves** the object /someDepartment/home.index (from RAM or disk), **encapsulates** the object in an HTTP response message, and **sends back** the response message to the client via its socket.
  4. The HTTP **server process tells TCP to close** the TCP connection, the TCP will (later) terminate the connection once it is sure that the client has received the response message intact (reliability).
  5. The HTTP **client receives the response message**. The TCP **connection terminates**. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, examines the HTML file, and **finds references to the 10 JPEG objects**.
  6. The first four **steps are then repeated** for each of the referenced JPEG objects.



Application Layer

HTTP: Non-persistent Connection (Example)

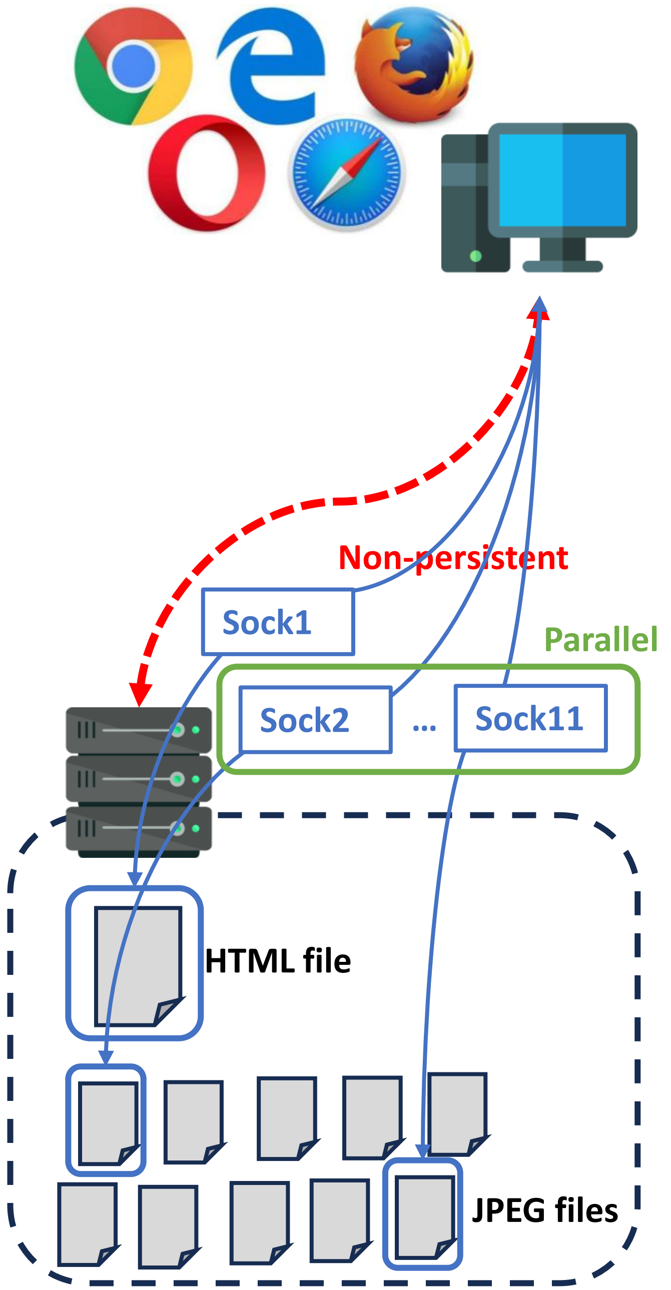
- Since the connection does not persist over different objects, we need **11 TCP connection** (and 11 pairs of sockets) to transfer the whole page, hence the elements of a page may arrive in **different moments**.
- The way web pages are displayed **depends on the browser**: two different browsers may interpret (hence, display to the user) a Web page in somewhat different ways.
- HTTP **defines only the communication protocol** between the client HTTP program and the server HTTP program, not how contents are displayed.



Application Layer

HTTP: Non-persistent Connection (Example)

- The different **connections** may be established in **sequence** or in **parallel**.
  - In this example, once we know that 10 JPEG images are needed (i.e., the main page is received), we may download all of them in parallel.
- In modern browsers users may also **configure** the degree of parallelism involved:
  - Most browsers open 5 to 10 parallel TCP connections, and each of these connections handles one request-response transaction.
  - Clearly, the use of parallel connections **shortens the response time but increases the computational cost**.
- Despite parallelism, establishing multiple (TCP) connections may still induce a significant **time overhead.**



Application Layer

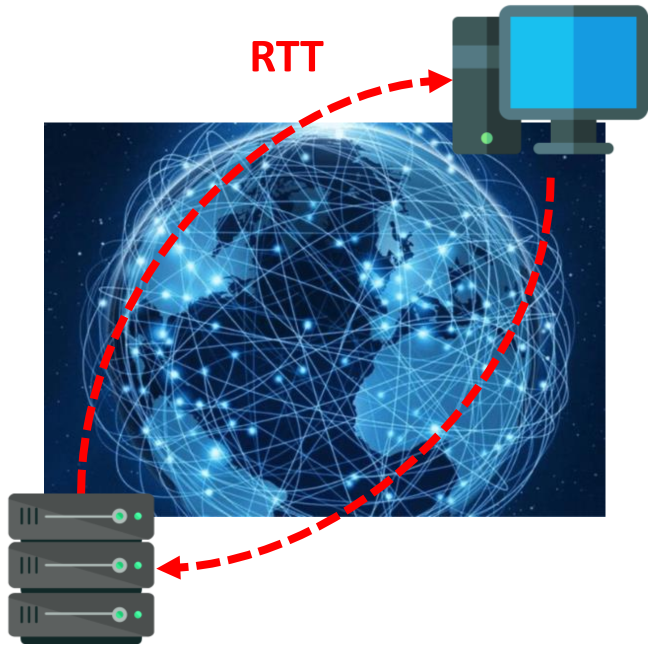
HTTP: Non-persistent Connections (RTTs)

- It is **difficult to precisely estimate times** on Internet. We can estimate the overhead it takes to finalize a HTML request in terms of **round-trip times** (RTTs).

- The **RTT** is the time it takes for a small packet to travel from client to server and back to the client.

- When a user clicks on a hyperlink the browser initiates a TCP connection between with the Web server.

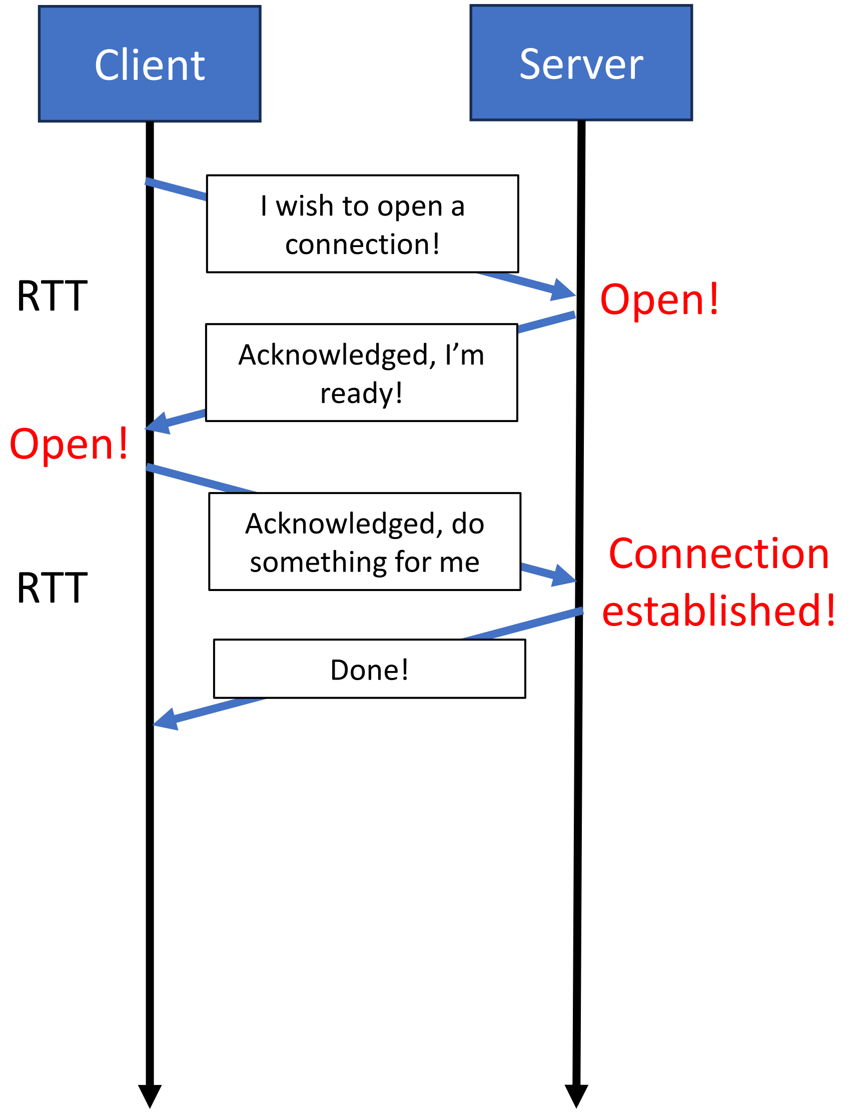
- TCP connections are guaranteed (connection-oriented) to do that, a **three-way handshake** is performed…



Application Layer

HTTP: Non-persistent Connections (RTTs)

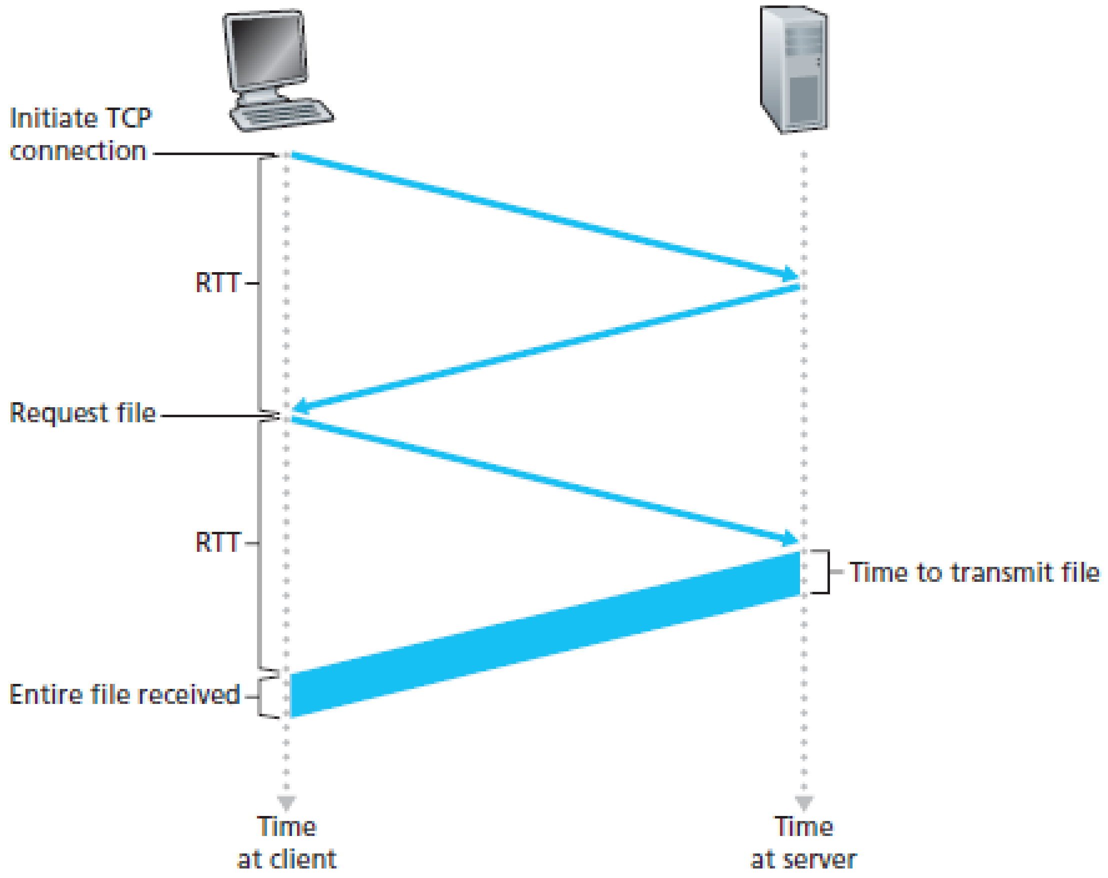
- TCP **three-way handshake** procedure is performed through the following 3 steps:
  - The client sends a small TCP segment to the server, to signal that a connection is requested.
  - The server acknowledges and responds with a small TCP segment.
  - The client acknowledges back to the server.
- From RTT's perspective, it takes 2 RTTs to establish a TCP connection (assuming no packet-loss).
- Notice that a similar handshake also takes place when **connection is closed**.



Application Layer

HTTP: Non-persistent Connections (RTTs)

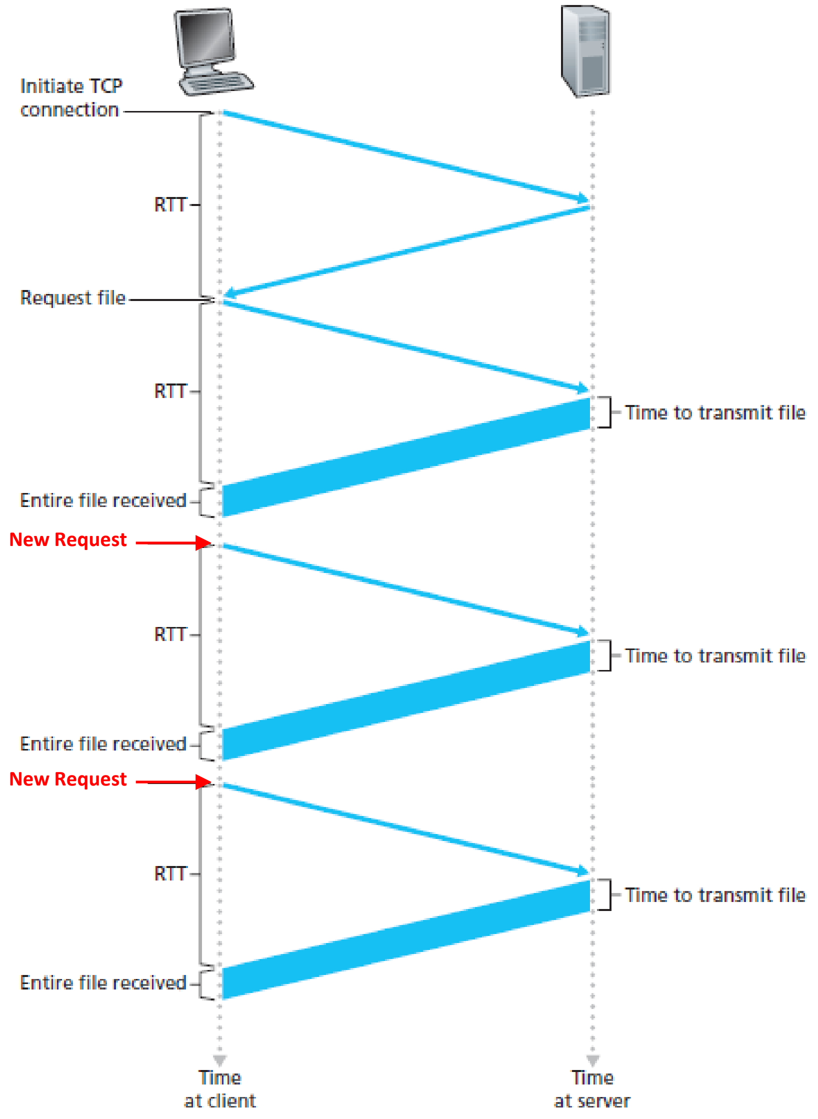
- The first two parts of the three-way handshake take **one RTT**.
- In HTTP, the client sends the HTTP **request message** combined with the third part of the three-way handshake (the acknowledgment).
- Once the request message arrives, the server sends the HTML file. This HTTP request/response eats up **another RTT**.
- Roughly, the total response time **is two RTTs plus the transmission time** at the server of the HTML file.



Application Layer

HTTP: Persistent Connections

- In persistent connections, the server **leaves open** the TCP connection after sending a response.
- Requests and responses between the same client and server can be sent over the same connection: **an entire Web page** (the HTML file and the 10 images from the previous example) can be sent over **a single persistent TCP connection**.
- **Multiple Web pages** residing on the same server can also be sent from the server to the same client over a single persistent TCP connection.
- Following this approach, we can save roughly **2-4 RRTs per object**.



Application Layer

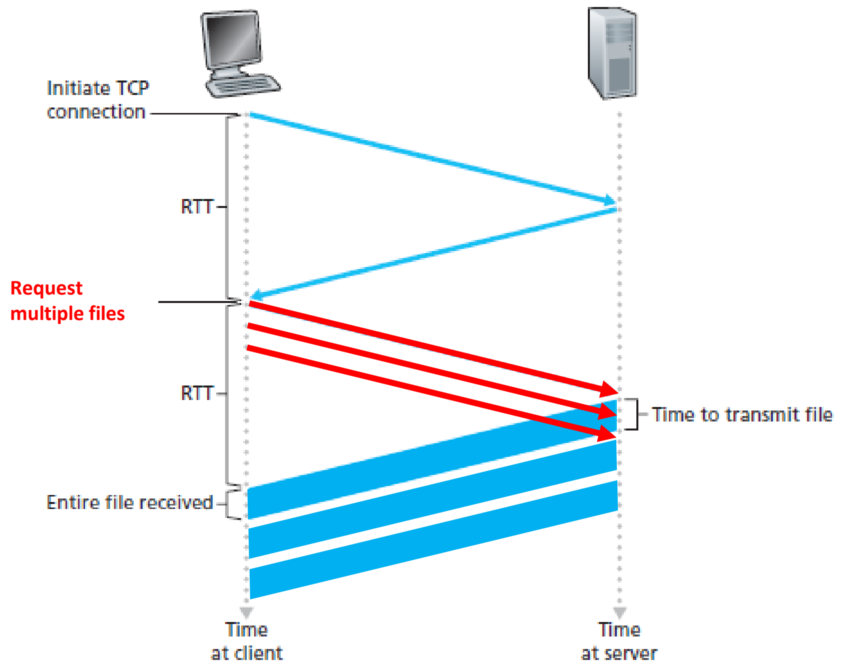
HTTP: Persistent Connections with Pipelining

- **Pipelining**: requests for objects can be made back-to-back, without waiting for replies to pending requests.

- In the last years (from HTTP/2) multiple requests and replies can be interleaved in the same connection.

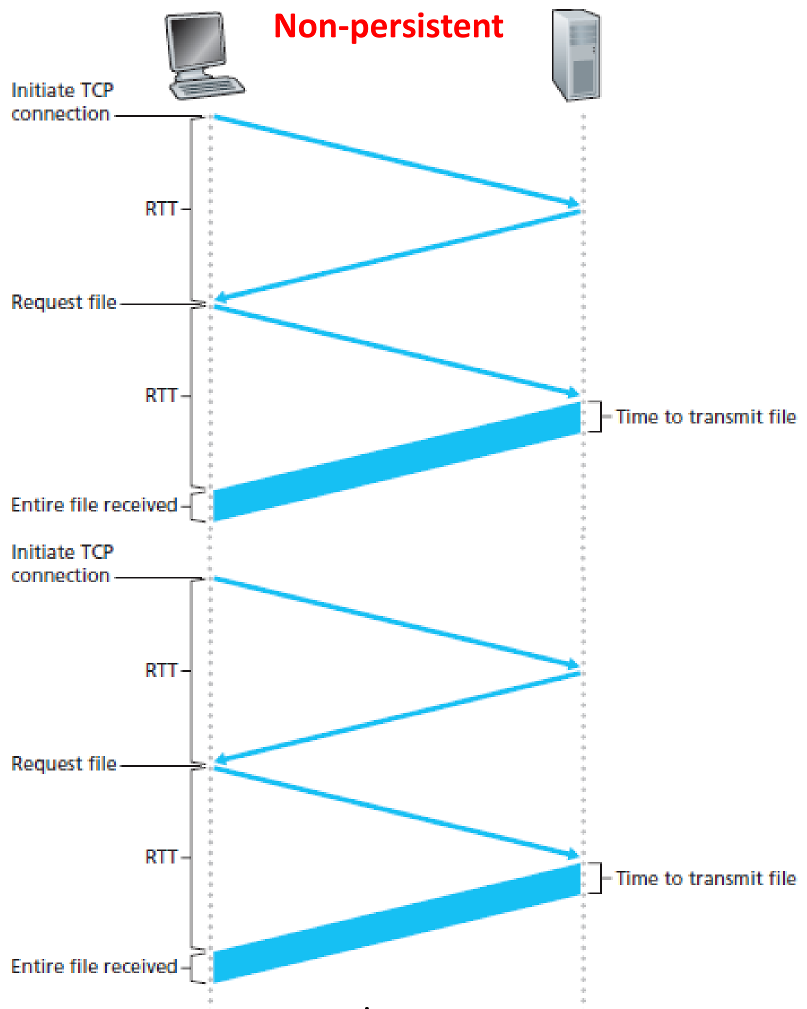
- There is also a mechanism to prioritize HTTP requests and replies within this connection.

- Persistent connections with **pipelining is currently the default** mode in HTTP.

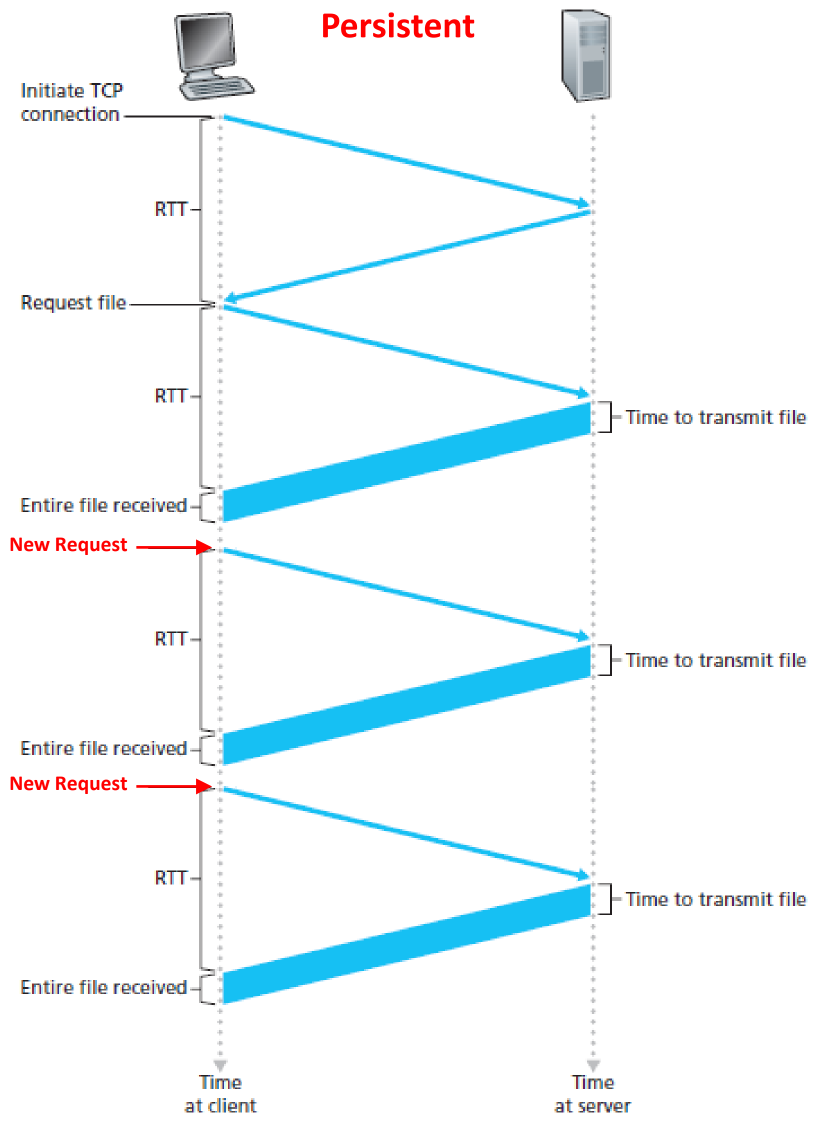


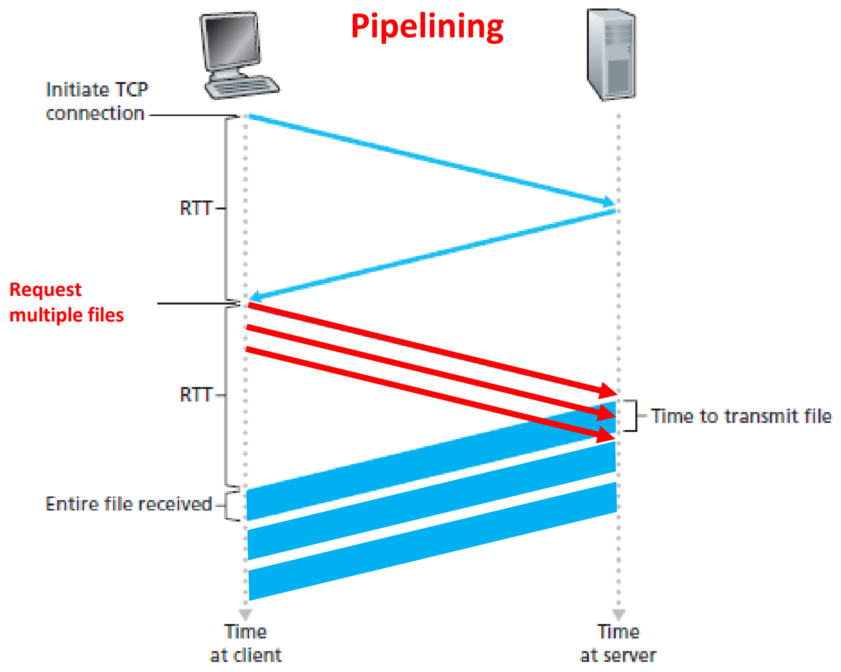
Application Layer

HTTP: Persistent vs. Non-persistent Connections



continues…





Rough comparison of RRTs needed to request 3 files considering the 3 modalities

Application Layer

HTTP: Persistent vs. Non-persistent Connections

- Persistent:
  - Faster, especially using pipelining.
  - Less resources needed (CPU and memory).
  - More complex to implement, especially using pipelining.
  - Connection may be left open even if unused. Typically, the HTTP server closes a connection when it isn’t used for a certain time (timeout).
- Non-persistent:
  - Easy to implement and nicely fitting the statelessness of HTTP.
  - Needs more resources, for each of these connections, TCP buffers must be allocated and TCP variables must be kept in both the client and server.
  - Slower, 1-2 additional RRTs needed per request.
  - Connections cannot be left open.