Computer Networks COL 334/672

Congestion Control

Tarun Mangla

Slides adapted from KR

Sem 1, 2024-25

Recap: TCP

- Connection establishment
- Reliability
- Flow control
- Congestion control
  - Loss-based
  - Delay-based
  - Network-assisted

Evolving transport-layer functionality

- TCP, UDP: principal transport protocols for 40 years
- different "flavors" of TCP developed, for specific scenarios:

>4 papers out of 62 papers on TCP in Sigcomm24

Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets "in flight"; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, "background" TCP flows

Sharing the network

Session Chair: Prateesh Goyal (Microsoft Research)

Keeping an Eye on Congestion Control in the Wild with Nebby Research Track

Ayush Mishra (National University of Singapore); Lakshay Rastogi (Indian Institute of Technology, Kanpur); Raj Joshi, Ben Leong (National University of Singapore)

SUSS: Improving TCP Performance by Speeding Up Slow-Start Research Track

Mahdi Arghavani, Haibo Zhang, David Eyers (School of Computing, University of Otago, New Zealand); Abbas Arghavani (School of Innovation, Design and Engineering, Mälardalen University, Sweden)

Principles for Internet Congestion Management Research Track

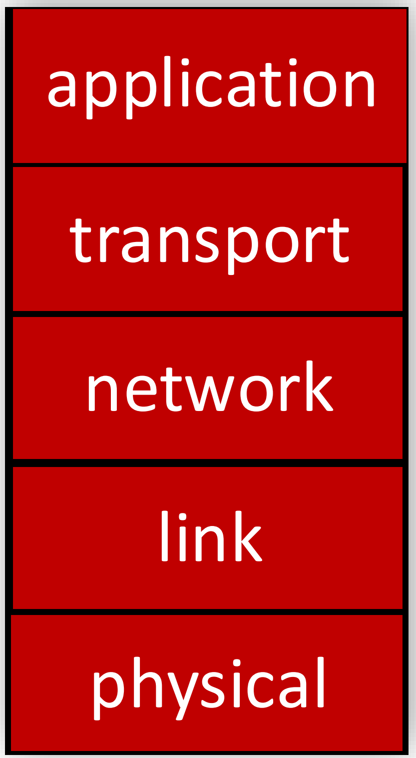
Lloyd Brown (UC Berkeley); Albert Gran Alcoz (ETH Zürich); Frank Cangialosi (BreezeML); Akshay Narayan (Brown University); Mohammad Alizadeh, Hari Balakrishnan (MIT); Eric Friedman (ICSI and UC Berkeley); Ethan Katz-Bassett (Columbia University); Arvind Krishnamurthy (University of Washington); Michael Schapira (Hebrew University of Jerusalem); Scott Shenker (ICSI AND UC Berkeley)

CCAnalyzer: An Efficient and Nearly-Passive Congestion Control Classifier Research Track

Ranysha Ware, Adithya Abraham Philip (Carnegie Mellon University); Nicholas Hungria (Carnegie Mellon Unversity); Yash Kothari, Justine Sherry, Srinivasan Seshan (Carnegie Mellon University)

Internet Layered Architecture

- *application:* supporting network applications
  - HTTP, IMAP, SMTP, DNS
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits “on the wire”



Networked Applications

- How to write a networked application?

  Using socket programming

- Multiple networked applications

  Can you give examples?

N/w Apps

1. File server
2. Email
3. Video streaming
4. Web browsing
5. Torrent
6. ssh
7. whatsapp / chatting apps
8. Video conferencing

Popular Applications

- HTTP → Web browsing
- Email
- DNS → Infrastructure apps
- P2P
- Video Streaming

HTTP

- Overview of HTTP
- Request/response message format
- State management
- Caching
- Request pipelining

Web and HTTP

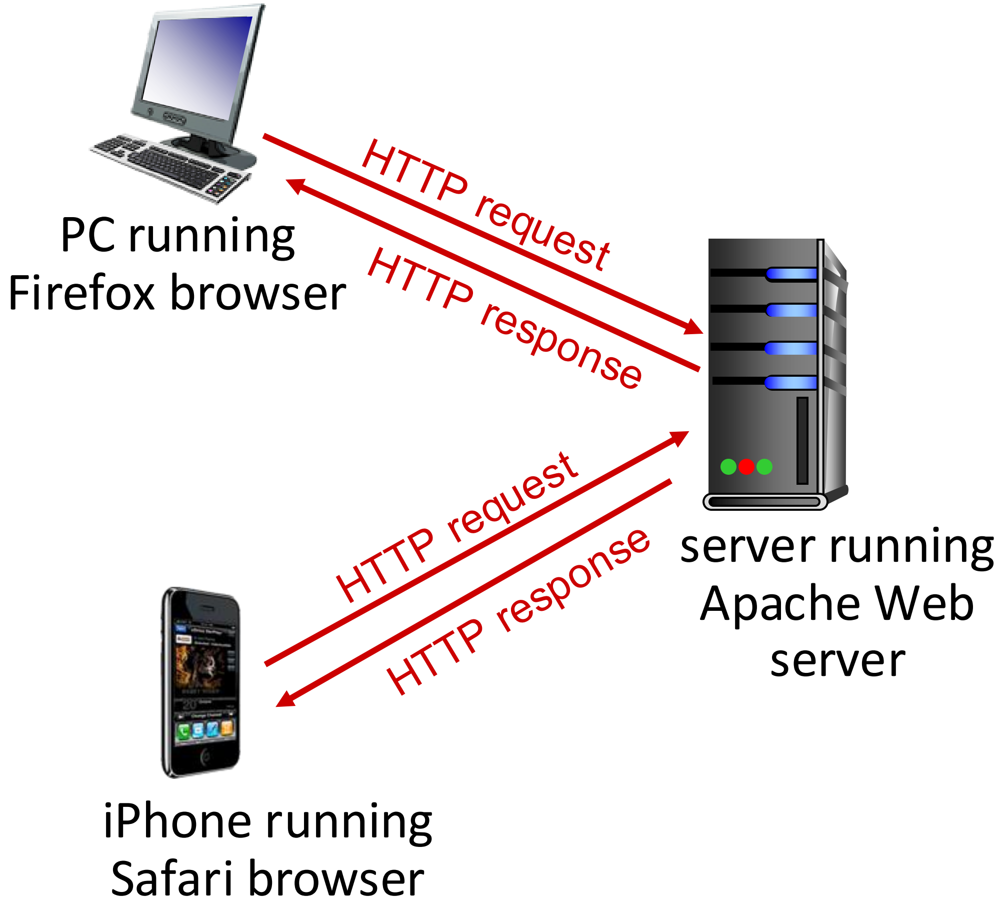
- World Wide Web or Web was the 2nd killer app over the Internet
- Goal of Web: organize and retrieve information over Internet
- Web is based on two sister protocols: HTML and HTTP
- HTML: HyperText Markup Language
  - language to create and structure web page
  - web page consists of *objects,* each of which can be stored on different Web servers
  - base HTML-file which includes several referenced objects, each addressable by a URL

$$\underbrace{\texttt{www.wikipedia.org}}_{\text{host name}}\underbrace{\texttt{/wiki/http}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

- Web's application-layer protocol
- Request/response protocol or client/server protocol
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

Which transport protocol? ***TCP***

Downloading a webpage

How to download multiple web objects?

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

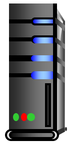
downloading multiple objects required multiple connections

Non-persistent HTTP: example

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**

(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80



1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

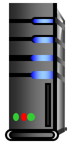
3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**

(containing text, references to 10 jpeg images)



4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

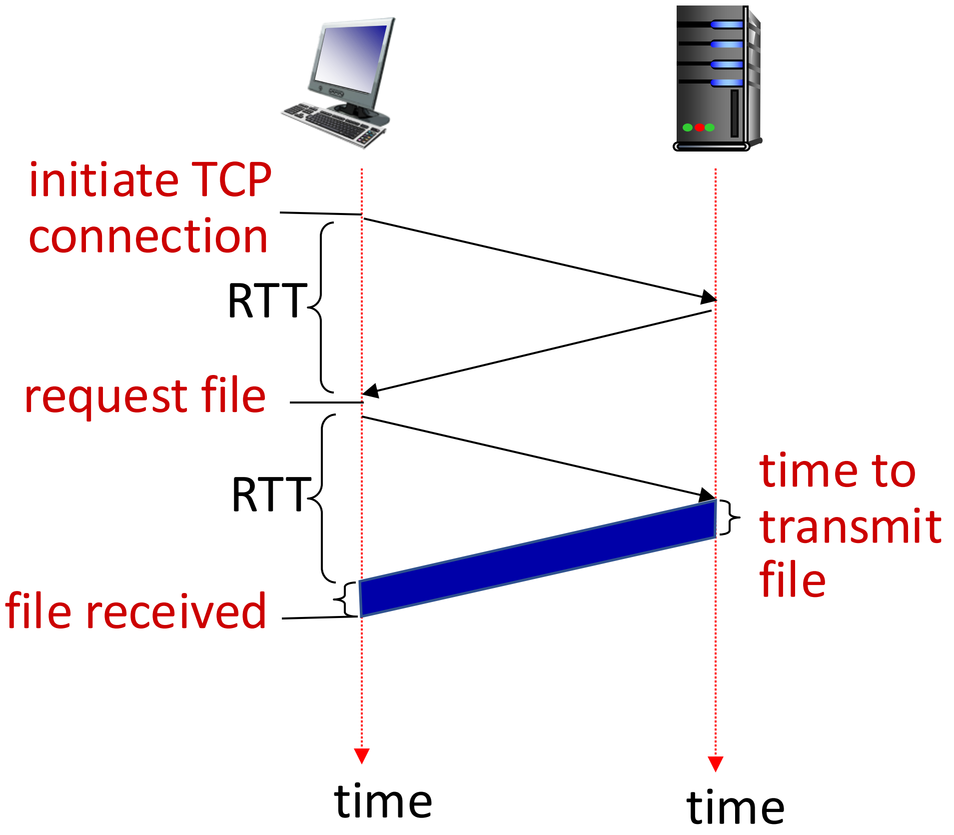
6. Steps 1-5 repeated for each of 10 jpeg objects

time

Non-persistent HTTP: response time

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

Persistent HTTP (HTTP1.1):

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects (cutting response time in half)

HTTP

- Overview of HTTP
- **Request/response message format**
- State management
- Caching
- Request pipelining

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

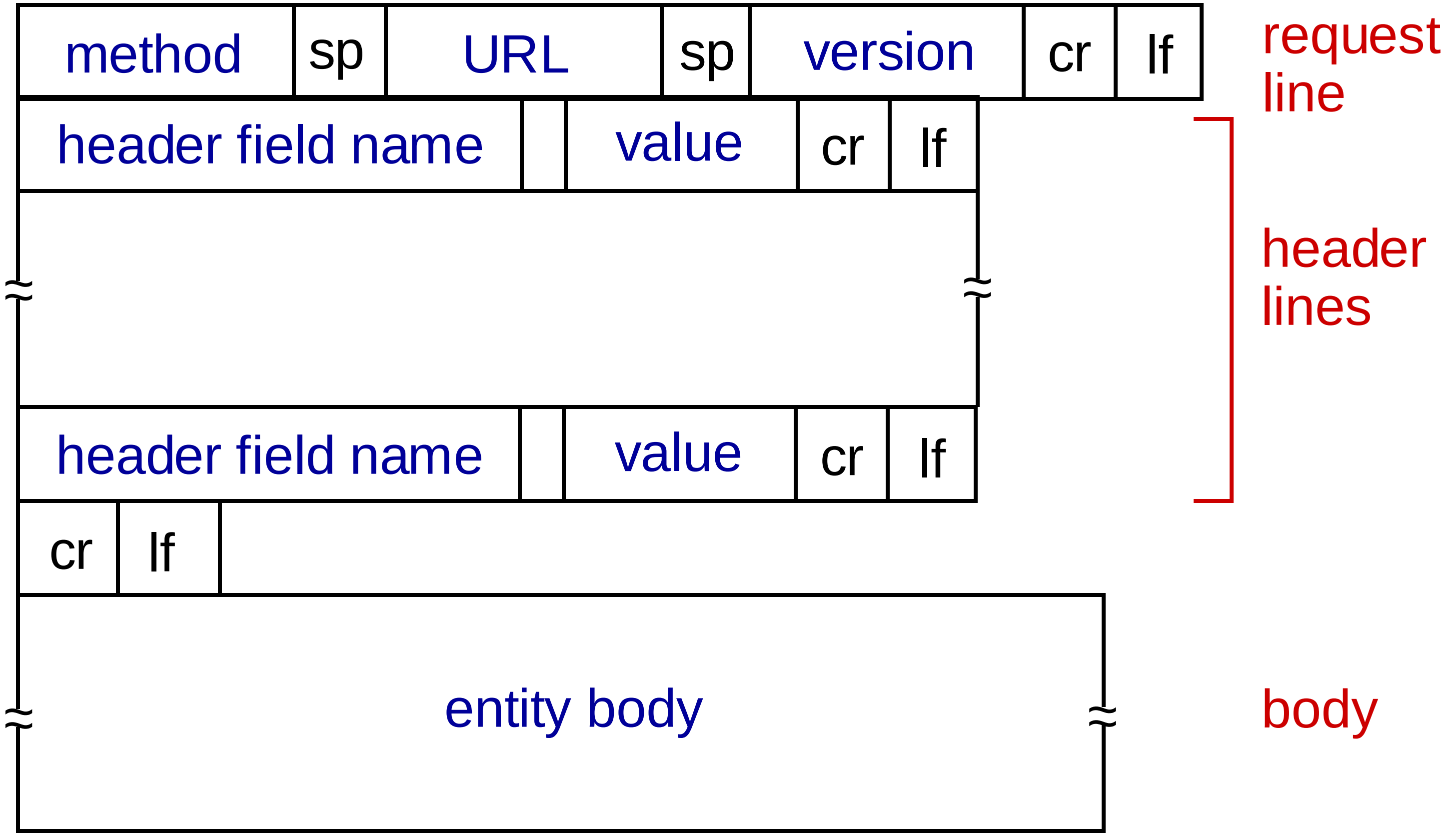
carriage return character

line-feed character

request line (GET, POST, HEAD commands)

carriage return, line feed at start of line indicates end of header lines

HTTP request message: general format



Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

```
www.somesite.com/animalsearch?monkeys&banana
```

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

HTTP response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
```

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

  200 OK
  - request succeeded, requested object later in this message

  301 Moved Permanently
  - requested object moved, new location specified later in this message (in Location: field)

  400 Bad Request
  - request msg not understood by server

  404 Not Found
  - requested document not found on this server

  505 HTTP Version Not Supported

Attendance