College of Information Studies

University of Maryland Hornbake Library Building College Park, MD 20742-4345

Service Model

Session 3

INST 346

Technologies, Infrastructure and Architecture

Goals for Today

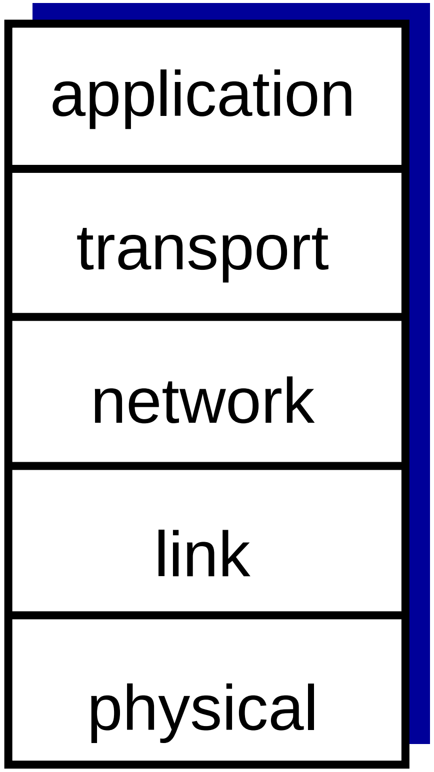
- Postgame the homework
- Application-layer Internet API
- Getahead: Hypertext Transfer Protocol
- What to expect on Thursday’s quiz

Muddiest Points

- Queueing delay formulas (La/R, …)
- Circuit switching
  - Frequency Division Multiplexing
- Transmission delay vs. propagation delay
- ISP vs. IXP vs. Content Provider Network

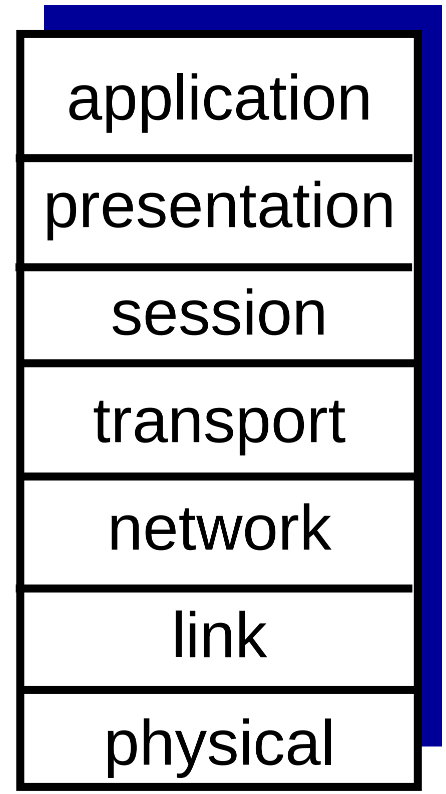
Internet protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *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, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



ISO/OSI reference model

- *presentation:* allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack "missing" these layers!
  - these services, *if needed*, must be implemented in application



Why layering?

dealing with complex systems:

- explicit structure allows identification of, and describing relationship between complex system's pieces
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
- some efficiency penalty
  - Worth it for general applications
  - Not practical in some specialized cases (e.g., planetary missions)

Chapter 2: application layer

our goals:

- conceptual, aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - (peer-to-peer paradigm)
- learn to create network applications
  - socket API
- learn about **protocols** by examining popular application-level protocols
  - Web: HTTP
  - Email: SMTP / POP3 / IMAP
  - Domain Name Service

Some network apps

- email
- Web
- streaming video (YouTube, Hulu, Netflix, …)
- remote login
- FTP
- P2P file sharing
- voice over IP (e.g., Skype)
- multi-user network games
- real-time conferencing
- social networking
- search
- …

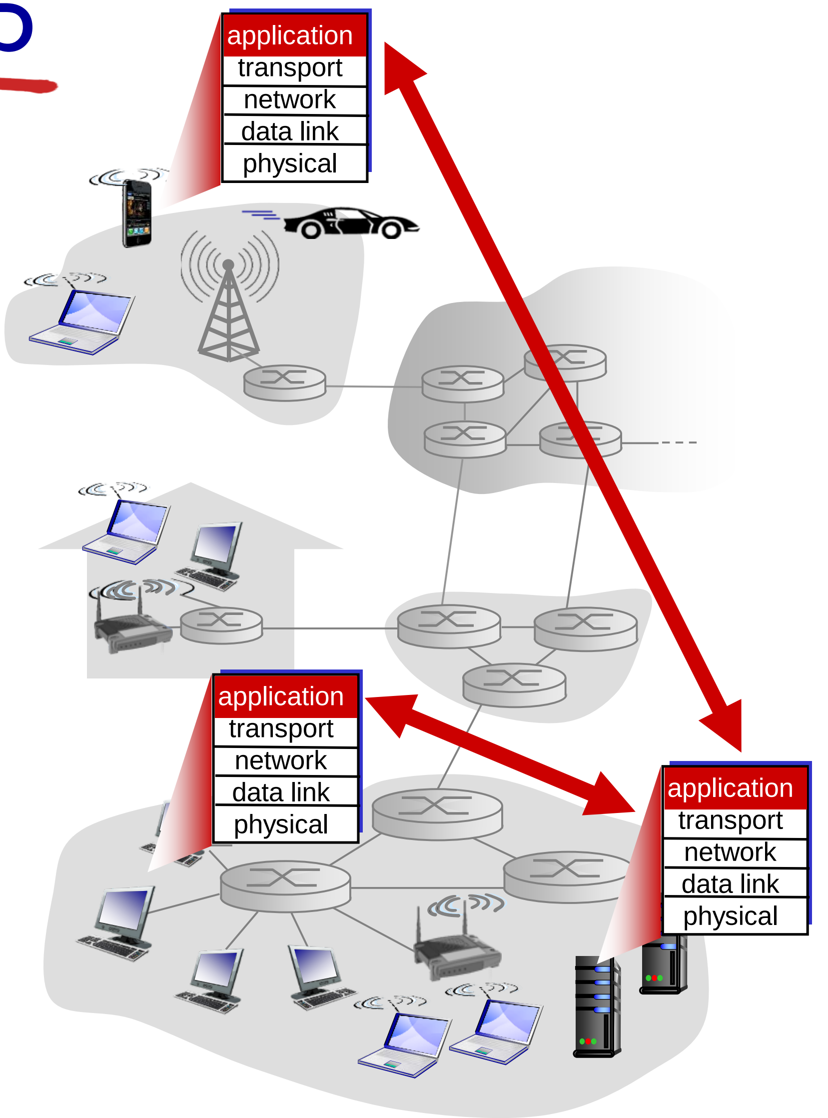
Creating a network app

write programs that:

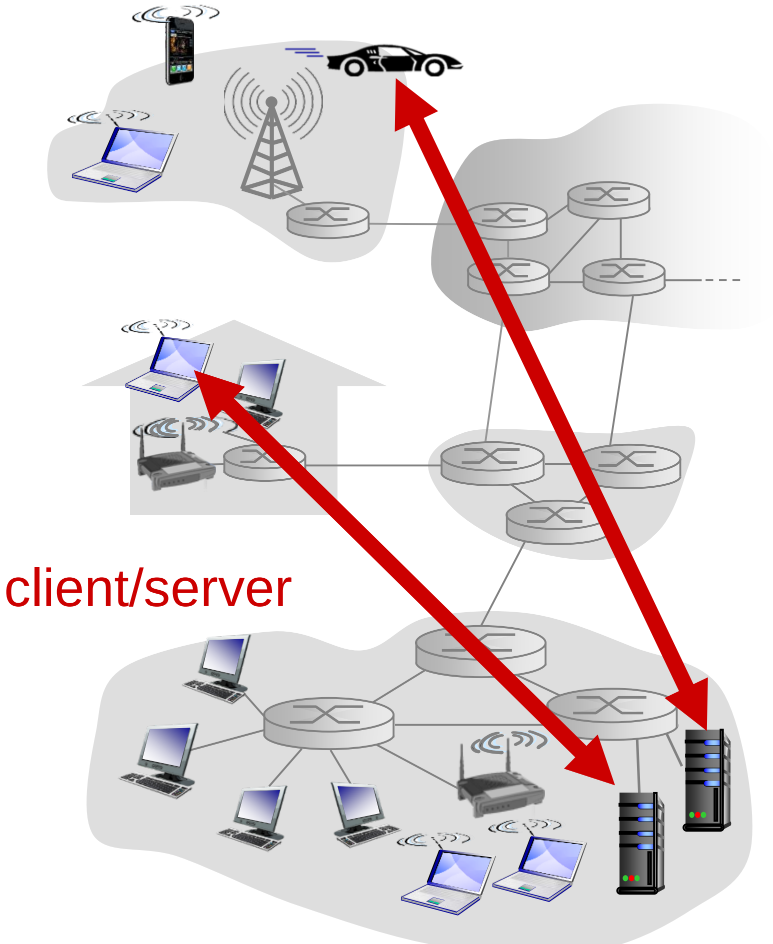
- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



Client-server architecture (e.g., Web)

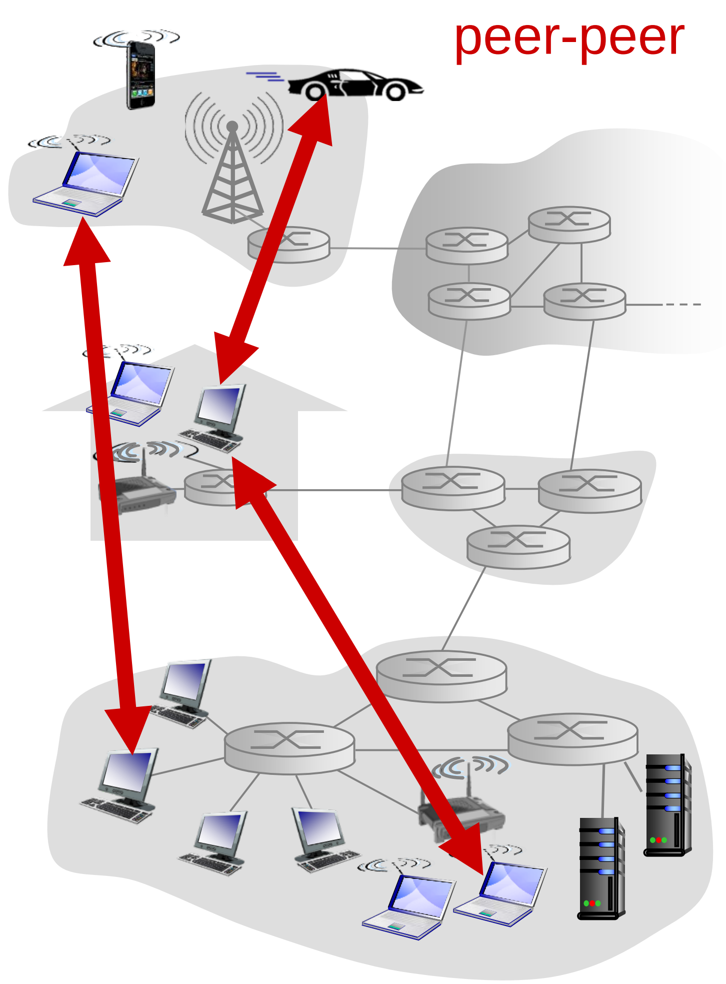


server:
- always-on host
- permanent IP address
- data centers for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P architecture (e.g., Skype)

- *no* central server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Processes communicating

process: program running within a host

- within a host, processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

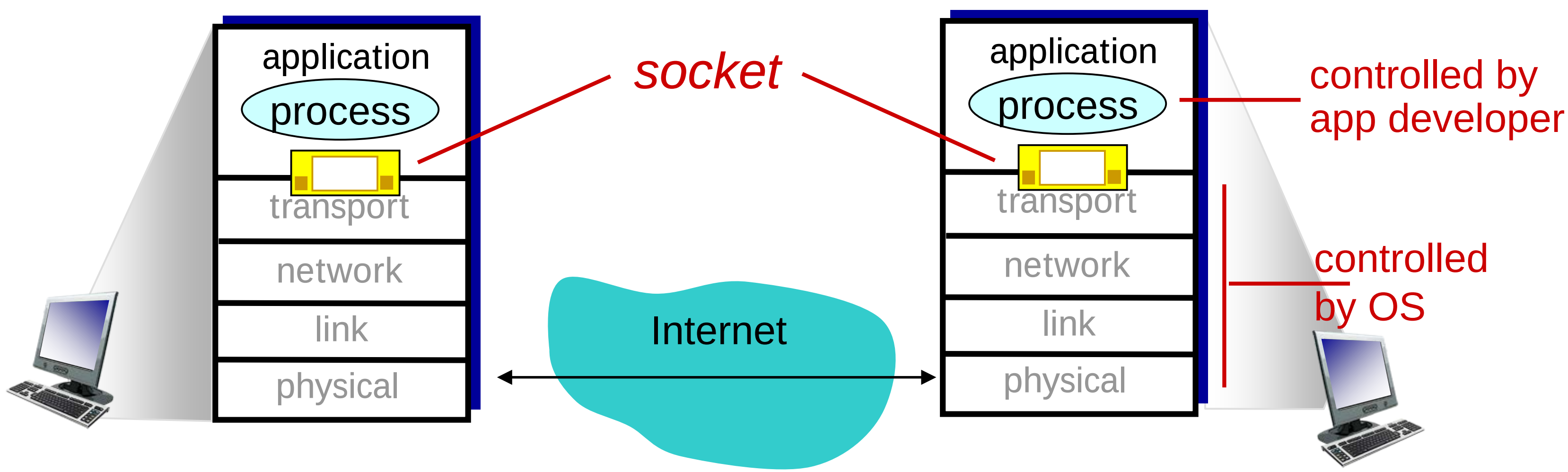
client process: process that initiates communication

server process: process that waits to be contacted

- note: P2P applications have both client & server processes

Sockets

- process sends/receives messages to/from its socket
- socket is analogous to an outbox or inbox
  - sending process places a message in an "outbox" (socket)
  - sending process relies on transport infrastructure to deliver message to "inbox" (socket) at receiving process
- sockets are identified by numbers
  - some sockets are defined by convention (e.g., 80=Web server)



The address of a socket

- to send or receive messages, a process must have a *socket*
- to identify that socket, the socket must have a unique *identifier*
- Each host has a unique 32-bit IP address
  - but many processes can be running on same host
- the unique *identifier* for a socket includes both the *IP address* of the host and the *port number(s)* associated with that process
- examples of "well known" port numbers:
  - Web server: 80
  - mail server: 25
- to identify the gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- processes can spawn new processes (and thus new port numbers)
  - So (if desired) each process can communicate with just one other process at a time

App-layer protocol must define:

- types of messages
  - e.g., request, response
- message syntax
  - what fields in messages
  - how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

"open" protocols:

- e.g., HTTP, SMTP
- defined in "Requests for Comment" (RFC's)
- designed for interoperability

proprietary protocols:

- e.g., Skype

What transport service does an app need?

data integrity
- some apps require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing
- some apps (e.g., Internet telephony, interactive games) require low delay

throughput
- some apps (e.g., video) need some minimum throughput
- other apps ("elastic apps") make use of whatever throughput they get

security
- encryption, data integrity, …

Transport service requirements: common apps

application	data loss	throughput	timing
file transfer	no loss	elastic	no limits
e-mail	no loss	elastic	no limits
Web documents	no loss	elastic	no limits
audio/video	loss-tolerant	5 kbps-1 Mbps	100's msec
video	loss-tolerant	10 kbps-5 Mbps	few secs
interactive games	loss-tolerant	few kbps up	100's msec
text messaging	no loss	elastic	yes and no

Internet transport protocols

UDP service (raw Internet):

- *packet delivery service:* no connection setup effort
- ***unreliable*** *data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security

TCP service (pseudo-circuit):

- *connection-oriented:* simulates a circuit between client and server processes (takes time to set up)
- ***reliable*** *transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming media	HTTP, RTP [RFC 1889]	UDP or TCP
Internet telephony	SIP, RTP, proprietary	UDP (TCP fallback)

Securing TCP (Preview of Session 22)

TCP (and UDP)

- no encryption
- passwds sent into socket traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- apps use SSL libraries, that "talk" to TCP

Getahead

The (World-Wide) Web

- a web page consists of *base HTML-file* which includes *several referenced objects*
- an object can be HTML file, JPEG image, Java applet, audio file,…
- each object is addressable by a *URL,* e.g.,

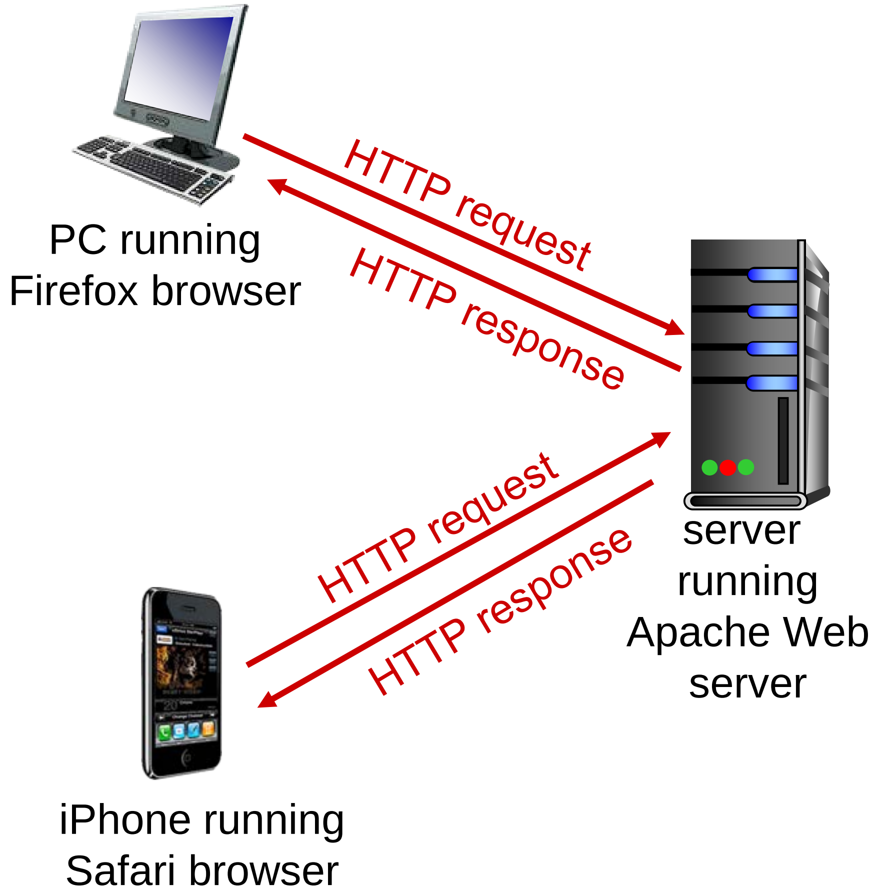
```
http://www.someschool.edu/someDept/pic.gif
```

protocol: `http://`
host name: `www.someschool.edu`
path name: `/someDept/pic.gif`

HTTP overview

HTTP: hypertext transfer protocol

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



HTTP uses TCP

- client creates a socket and initiates a TCP connection to port 80 on server
- server creates a new socket for this connection, forwards the TCP to that socket, and accepts the connection there.
- HTTP messages (application-layer protocol messages) are exchanged between browser (HTTP client) and Web server (HTTP server)
- Eventually, the TCP connection is closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP (1.0)

suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

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

2. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client of new port number

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

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

5. HTTP client receives response message containing html file, displays html.

6. HTTP server closes TCP connection.

7. Parsing html file, the browser finds 10 referenced jpeg objects and starts again at 1. (10 times!)

time

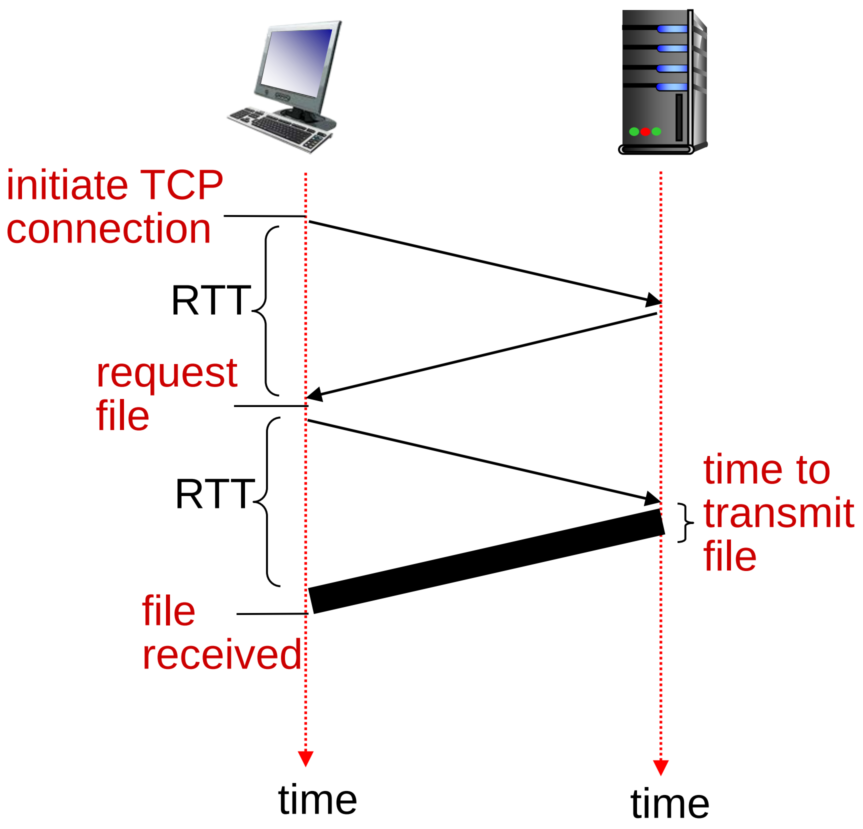
HTTP (1.0) response time

Define Round Trip Time (RTT) as time for a small packet to travel from client to server and back

HTTP response time:

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

  2*RTT+ file transmission time



HTTP evolution

original HTTP (1.0)

- Set up TCP connection
- Send one object
- Close TCP connection
- Can require many TCP connection setups for one Web page (slow!)

persistent HTTP (1.1 & later)

- Set up the TCP connection once
- Send all the objects on one Web page
  - Don't close the connection after each request
  - Avoids repeated connection setups
- Timeout: Close the TCP connection after some period with no activity

Better network utilization

original HTTP (1.0):

- requires 2 RTTs per object
- plus Operating System (OS) overhead for *each* TCP connection
- workaround: browsers can open parallel TCP connections to fetch referenced objects

persistent HTTP:

- server leaves connection open after sending response
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

Quizzes

- Goals
  - Begin to prep for Exam 1
  - Incentive to keep up with the readings

- Format
  - Every Thursday (except exam weeks); see schedule
  - 5 minutes at start of class, sharply timed
  - On paper (write directly on the quiz)
  - Open book, open notes, open Web
  - No communication with anyone until quiz ends!

Before You Go

On a sheet of paper, answer the following (ungraded) question (no names, please):

What was the muddiest point in today's class?