

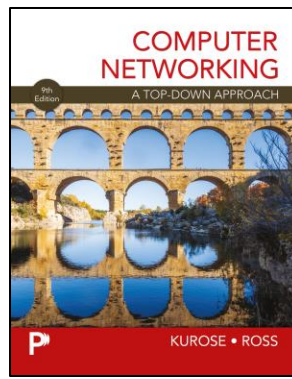
CSC 361

Computer Networks

Application Layer

(Chapter 2)

Wenjun Yang
Summer 2026



Application Layer: Overview (1 of 7)

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- video streaming and content distribution networks
- socket programming with UDP and TCP



Application Layer: Overview (2 of 7)

Our goals:

- conceptual and implementation aspects of application-layer protocols
 - transport-layer service models
 - client-server paradigm
 - peer-to-peer paradigm
- learn about protocols by examining popular application-layer protocols and infrastructure
 - HTTP
 - SMTP, IMAP
 - DNS
 - video streaming systems, CDNs
- programming network applications
 - socket API

Some Network Apps

- social media
 - Web
 - text messaging
 - e-mail
 - multi-user network games
 - streaming stored video (YouTube, Hulu, Netflix)
 - P2P file sharing
 - voice over IP
 - real-time video conferencing (e.g., Zoom)
 - Internet search
 - remote login
 - ...
- Q:** your favorites?

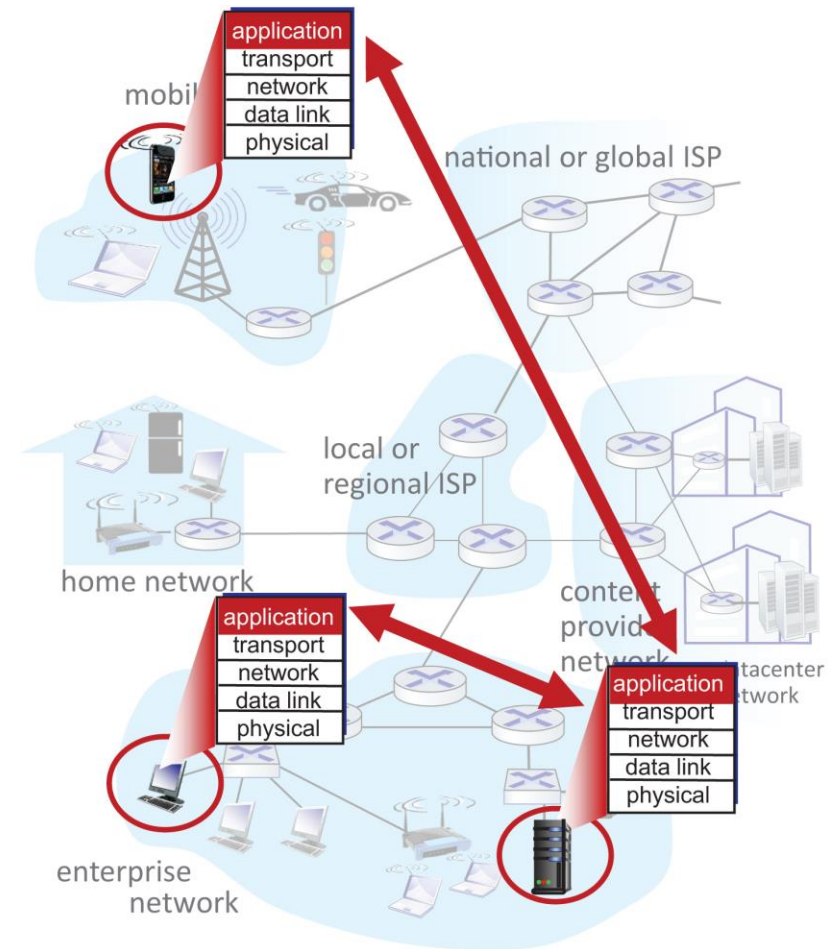
Creating a Network App

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



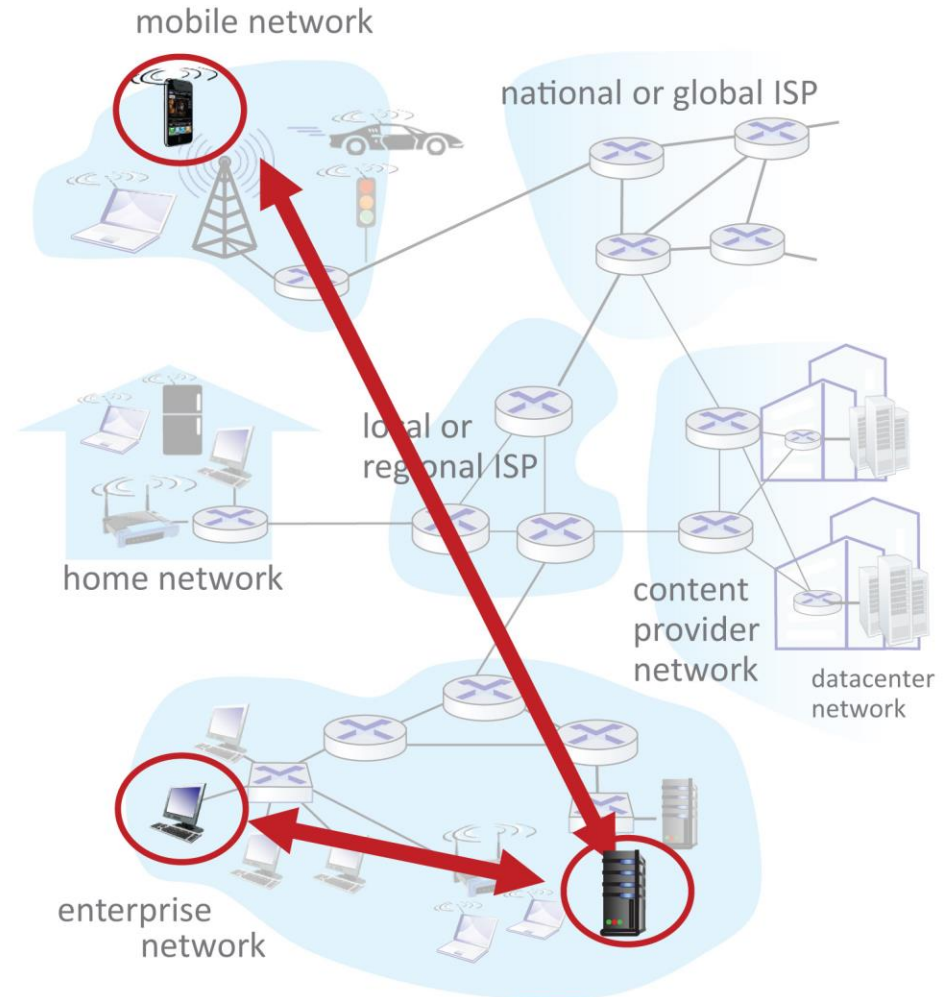
Client-Server Paradigm

server:

- always-on host
- permanent IP address
- often in data centers, for scaling

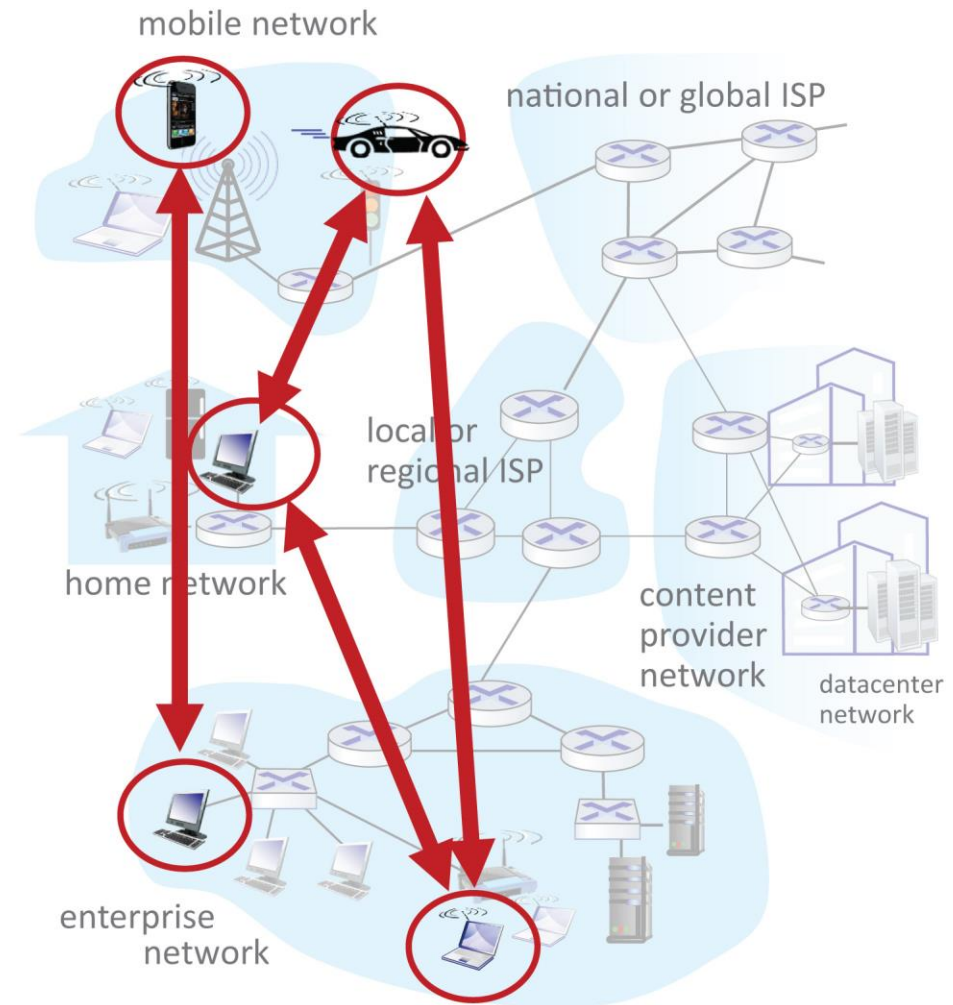
clients:

- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do **not** communicate directly with each other
- examples: HTTP, IMAP, FTP



Peer-Peer Architecture

- no always-on 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
- example: P2P file sharing [BitTorrent]



Processes Communicating

process: program running within a host

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

clients, servers

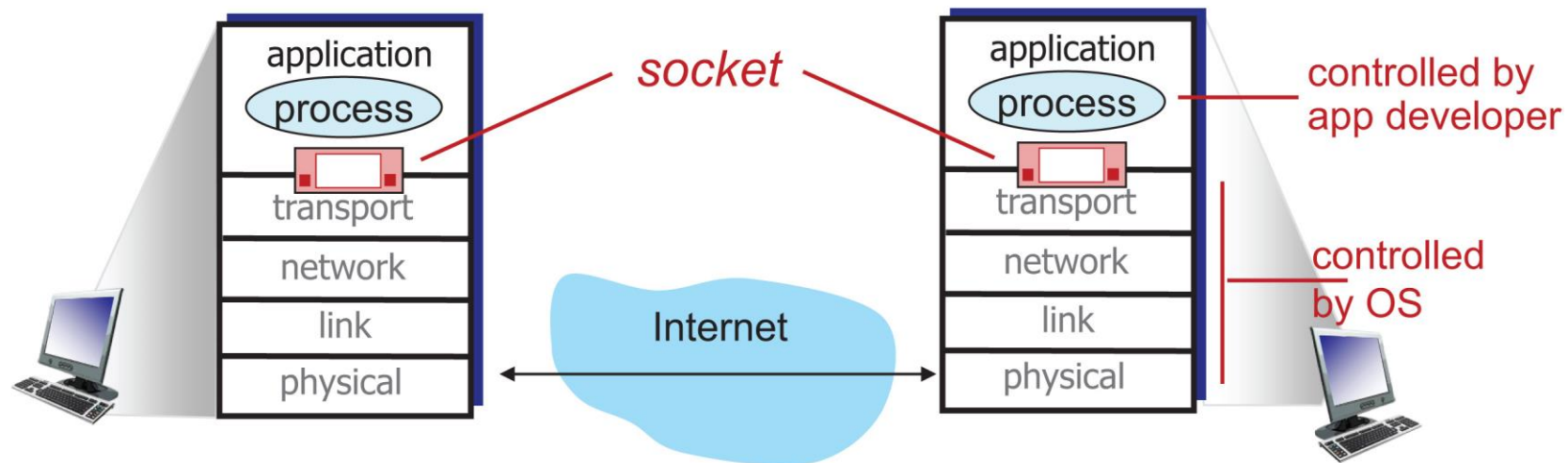
client process: process that initiates communication

server process: process that waits to be contacted

- note: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
 - two sockets involved: one on each side



Addressing Processes

- to receive messages, process must have **identifier**
- host device has unique 32-bit IP address
- **Q:** does IP address of host on which process runs suffice for identifying the process?
 - **A:** no, many processes can be running on same host
- **identifier** includes both **IP address** and **port numbers** associated with process on host.
- example port numbers:
 - HTTP server: 80
 - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
 - **IP address:** 128.119.245.12
 - **port number:** 80

An Application-Layer Protocol Defines:

- **types of messages exchanged**,
 - 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:

- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Zoom

What Transport Service Does an App Need?

data integrity

- some apps (e.g., file transfer, web transactions) 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 to be “effective”

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

security

- encryption, data integrity, ...

Transport Service Requirements: Common Apps

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

Internet Transport Protocols Services

TCP service:

- **reliable transport** between sending and receiving process
- **flow control:** sender won't overwhelm receiver
- **congestion control:** throttle sender when network overloaded
- **connection-oriented:** setup required between client and server processes
- **does not provide:** timing, minimum throughput guarantee, security

UDP service:

- **unreliable data transfer** between sending and receiving process
- **does not provide:** reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: why bother? **Why** is there a UDP?

Internet Applications, and Transport Protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP [RFC 7230, 9110]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Securing TCP

Vanilla TCP & UDP sockets:

- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (TLS)

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

TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn
- cleartext sent into “socket” traverse Internet **encrypted**
- more: Chapter 8

Application Layer: Overview (3 of 7)

- Principles of network applications
- **Web and HTTP**
- E-mail, SMTP, IMAP
- The Domain Name System
DNS
- video streaming and content distribution networks
- socket programming with UDP and TCP



Web and HTTP

First, a quick review...

- web page consists of **objects**, each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file,...
- web page consists of **base HTML-file** which includes **several referenced objects**, **each** addressable by a **URL**, e.g.,

www.someschool.edu/someDept/pic.gif

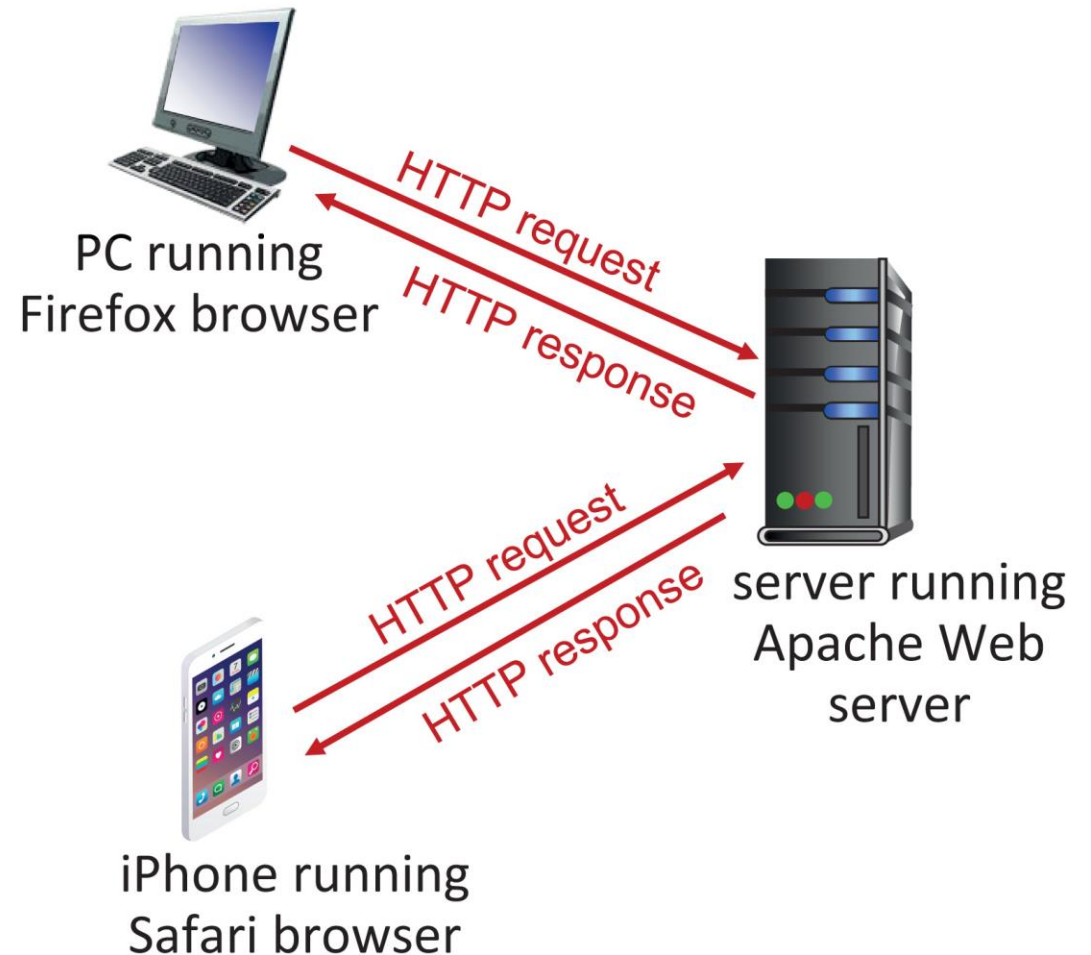
host name

path name

HTTP Overview (1 of 2)

HTTP: hypertext transfer protocol

- Web's application-layer protocol
- client/server model:
 - **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 (2 of 2)

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection 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 Connections: Two Types

Non-persistent HTTP

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

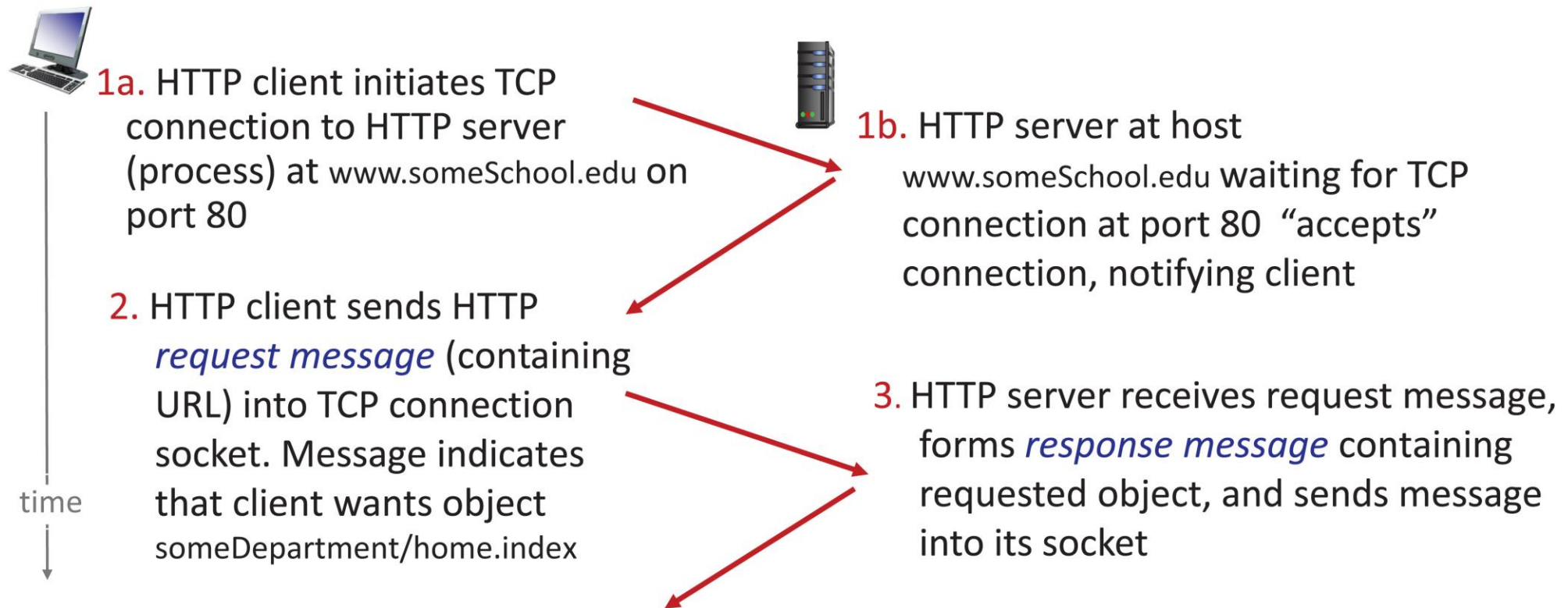
downloading multiple objects
required multiple connections

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over **single** TCP connection between client, and that server
- TCP connection closed

Non-Persistent HTTP: Example (1 of 2)

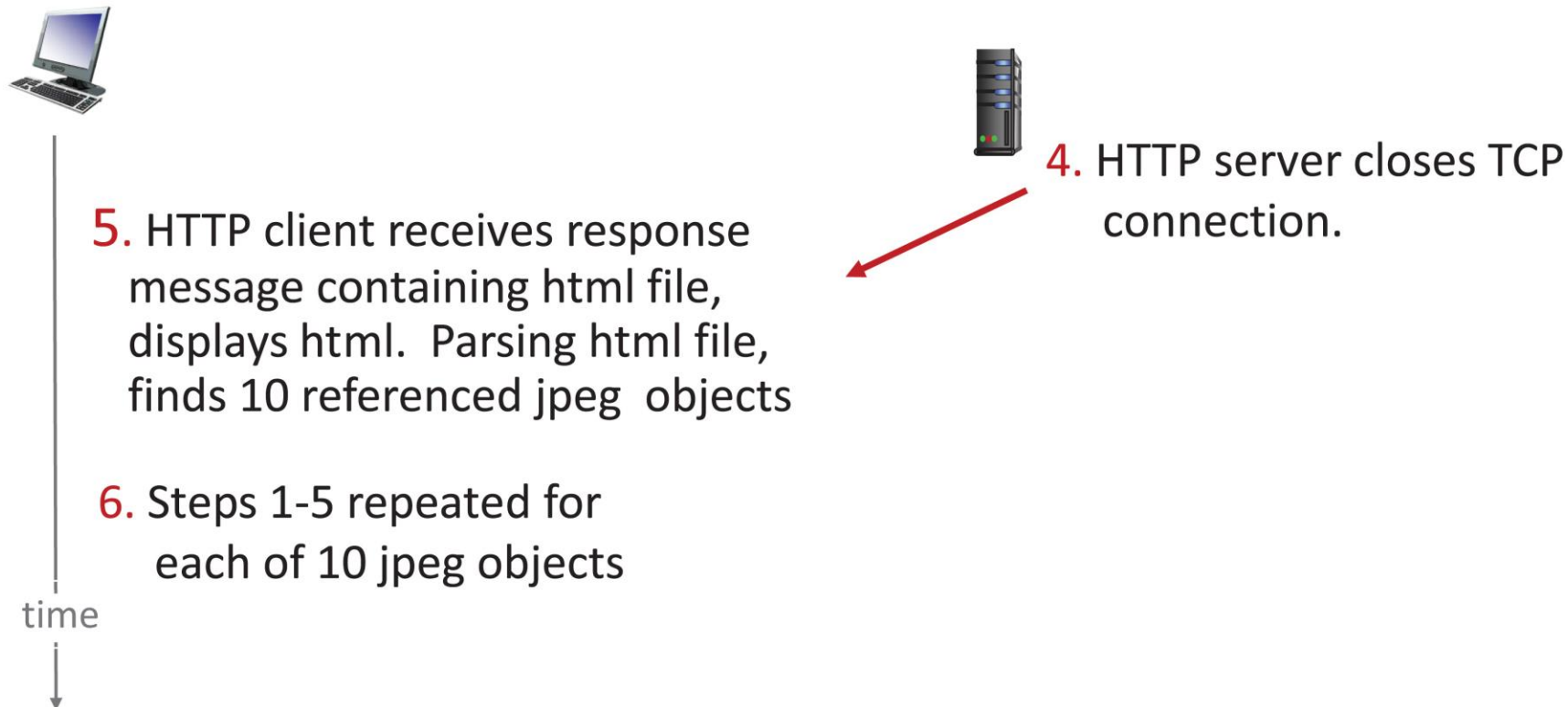
User enters URL: www.someSchool.edu/someDepartment/home.index
(containing text, references to 10 jpeg images)



Non-Persistent HTTP: Example (2 of 2)

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

(containing text, references to 10 jpeg images)

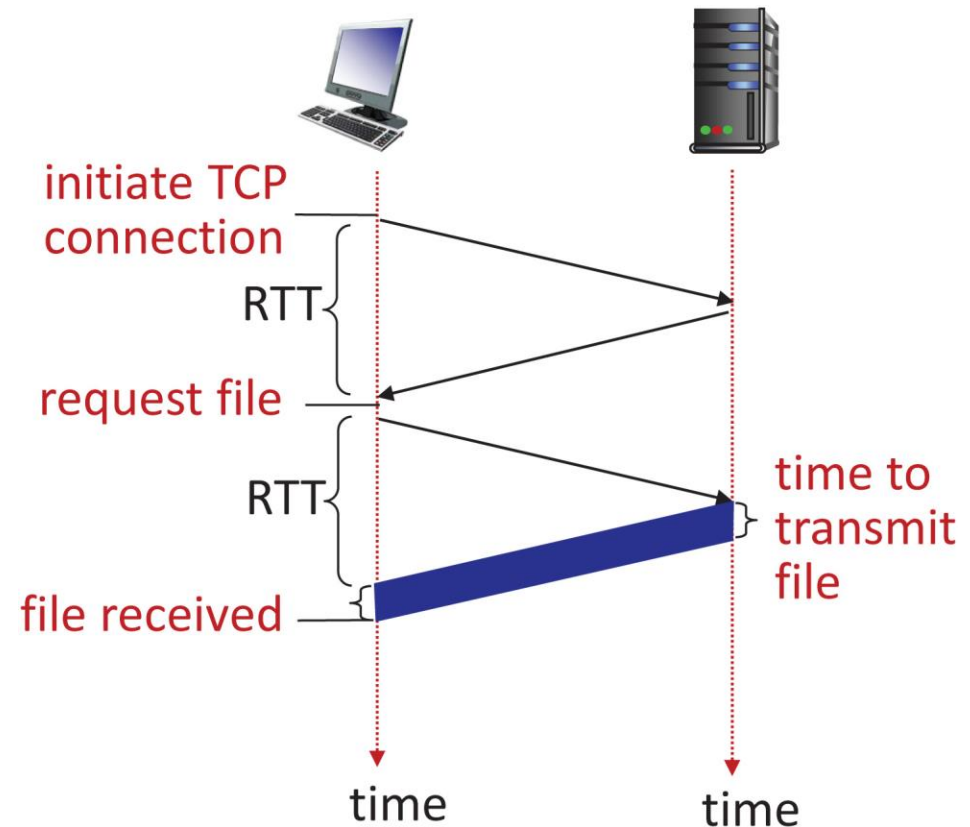


Non-Persistent HTTP: Response Time

RTT (definition): time for a small packet to travel from client to server and back

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 = $2RTT + \text{file transmission time}$

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)