# Lecture 3

# The Application Layer

## Part 1



SDU

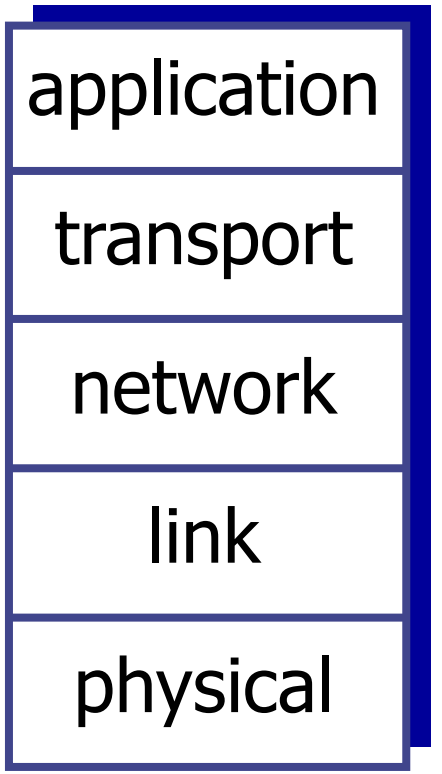
# Subjects of today:

- The Role of the Application Layer
- Apps and protocols
- Architectures
- Interfacing the Application Layer
- Web and HTTP

# 3.1 The Role of the Application Layer

# The Internet Protocol Stack (again…)

- *application:* **supporting network applications**
- *transport:* process-process data transfer
- *network:* routing of datagrams from source to destination
- *link:* data transfer between neighboring network elements
- *physical:* bits "on the wire"

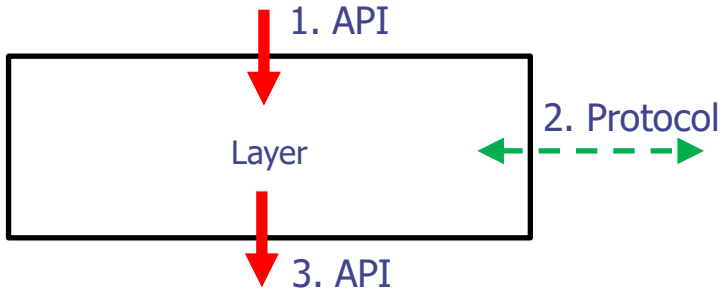





SDU

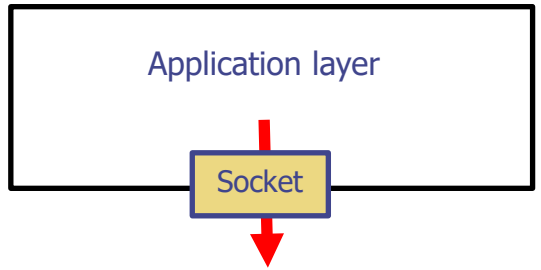
# Every layer must...

1. offer services to an upper layer.
2. comply with agreed protocols.
3. utilize services from the underlying layer.



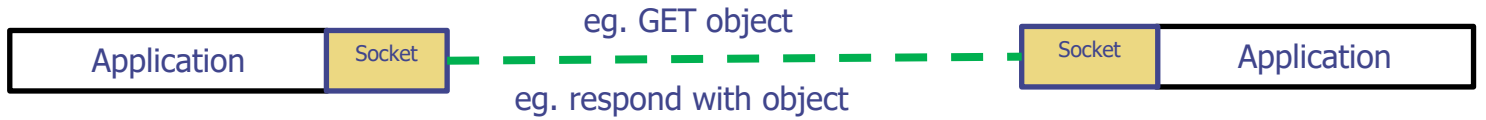
# The application layer…

- has no upper layer.
- hosts the applications.
- utilizes services from the transport layer through sockets.

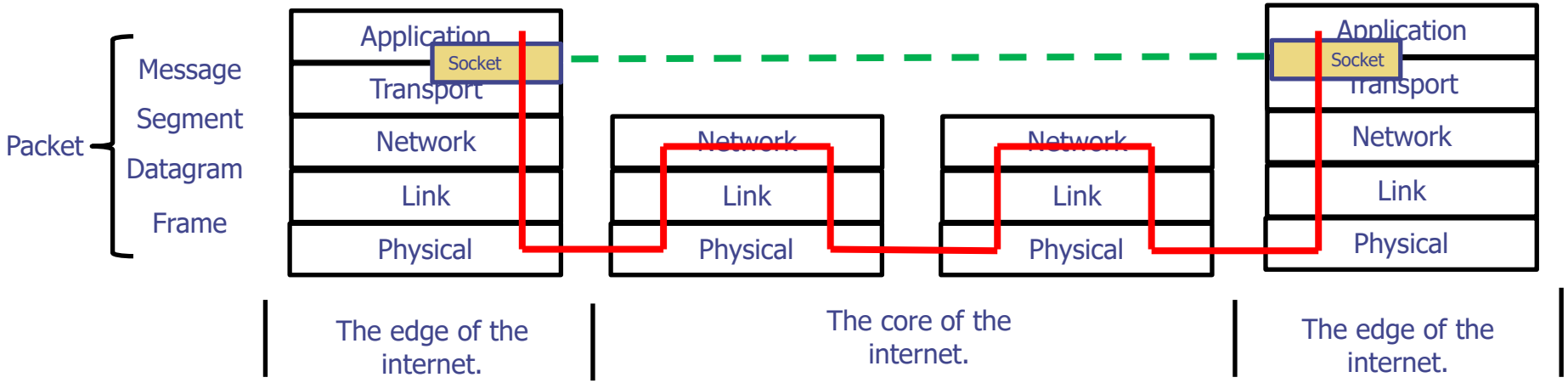


# Scoping

## Application view




## Real view



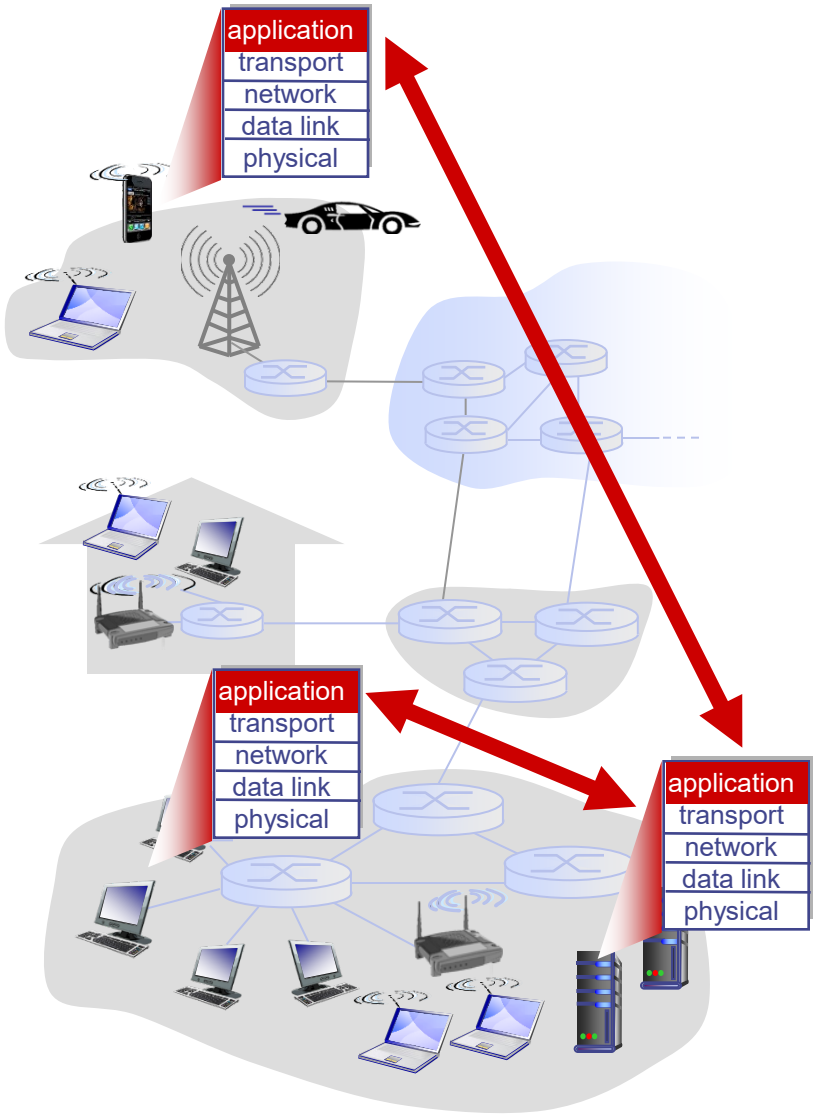
# Creating a network app

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 as:

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



# 3.2 Apps and Protocols

# Example network apps

- Web browsing
- E-mail
- remote login
- P2P file sharing
- Text messaging
- Voice over IP (VoIP)
- Real-time video conferencing
- Streaming stored video
- Multi-user network games
- Social networking
- Search
- IoT Device Communication
- Etc.



SDU

# Some application layer protocols

| Application | Protocol examples |
|---|---|
| Mail | SMTP (sending)<br>POP3 (receiving)<br>IMAP (Syncing) |
| Web | HTTP |
| Login | TELNET<br>SSH |
| File sharing | FTP<br>SFTP |
| | |
| Proprietary application | Standard protocols |
| Proprietary application | Proprietary protocols |



SDU

# 3.3 Architectures



SDU

# Application architectures

**Possible structure of applications:**

- client-server
- peer-to-peer (P2P)
- Publisher-subscriber (especially at IOT)
- The cloud

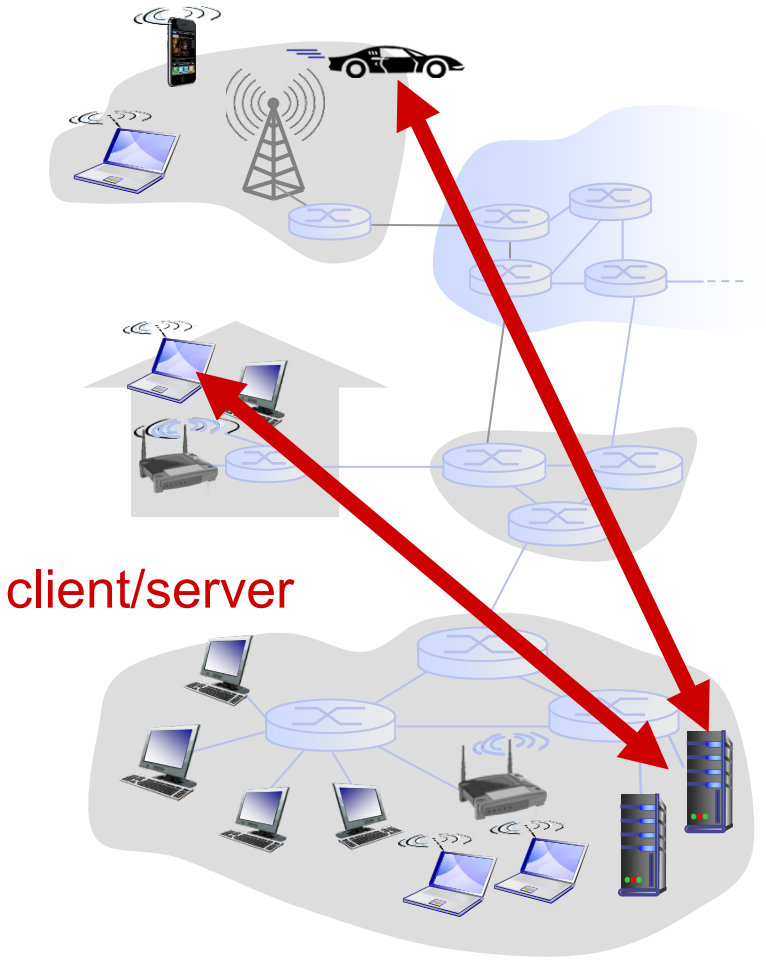
# Client-server architecture

Server:

- Always-on host
- Permanent IP address
- Data centers for scaling

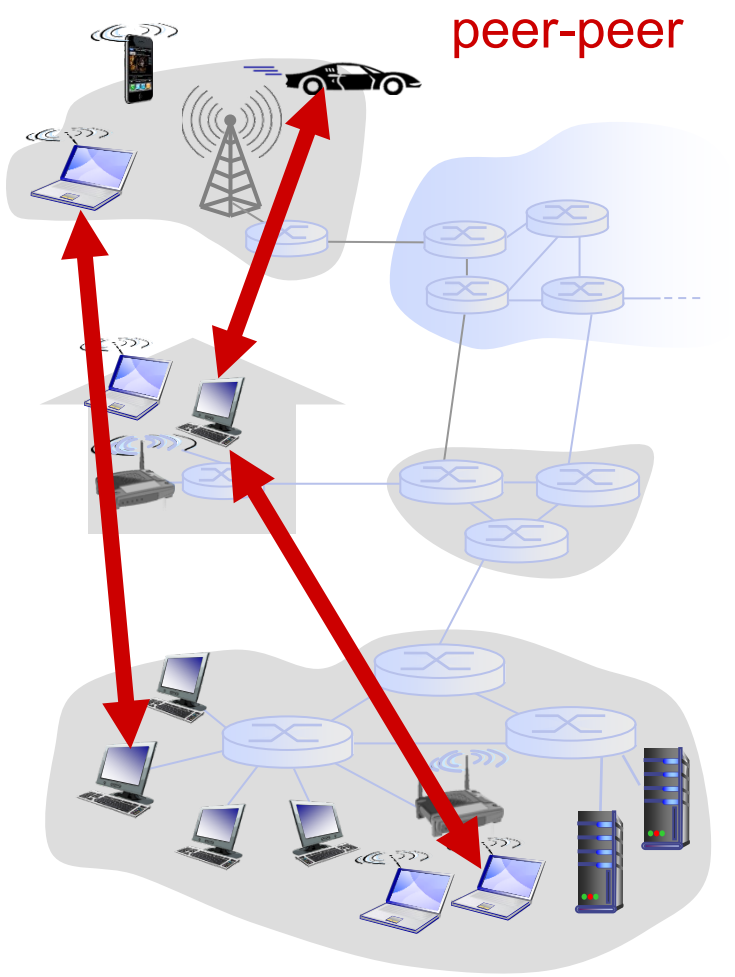
Clients:

- Communicate with server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other



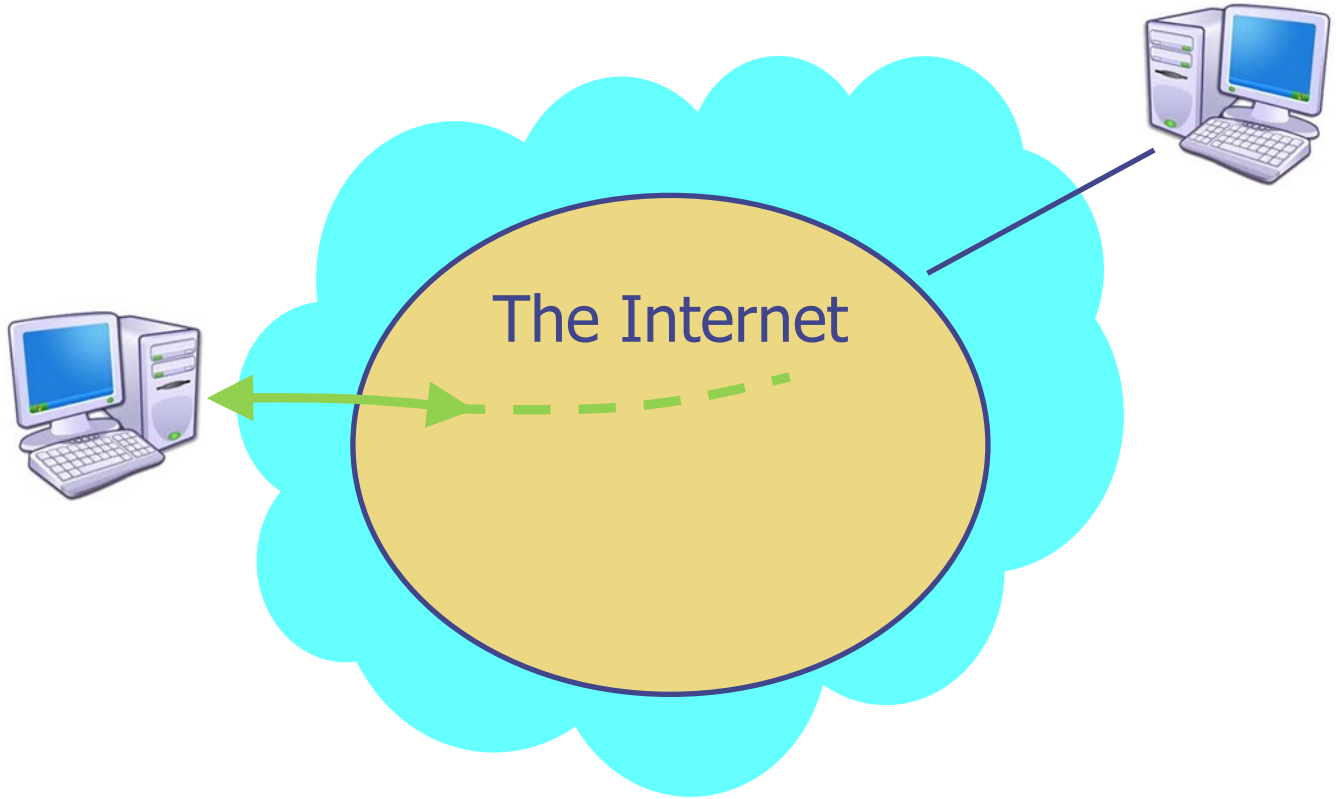
# P2P architecture

- *Not* 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



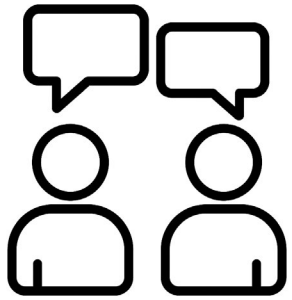
# Cloud architecture

- Decentralized ”virtual” server
  - Dynamic resource management
  - Scalable
  - Complex

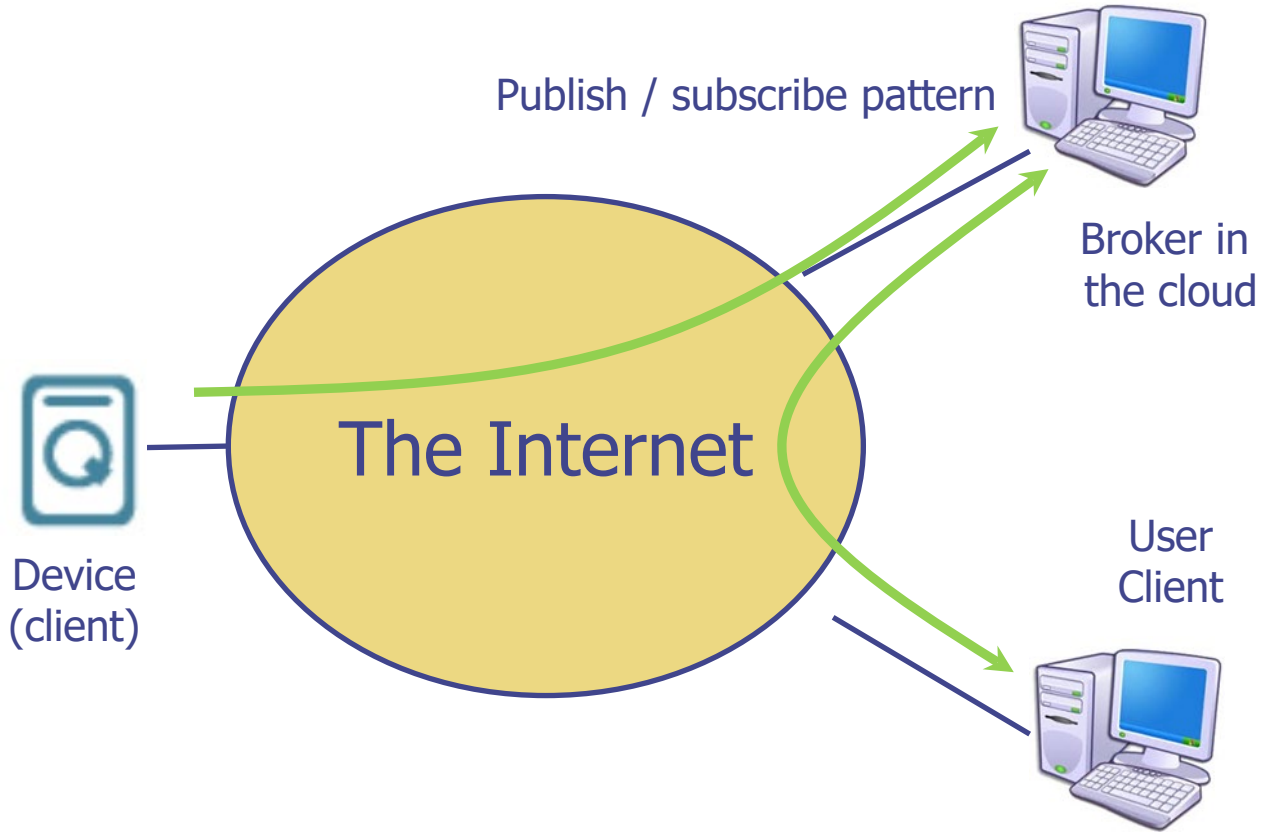


# Publisher-Subscriber architecture

- Register *Topic* at *Message Broker*
- Example: MQTT (see https://mqtt.org/)
  - Lightweight and Efficient
  - Scalable
  - Bi-directional Communications



Which applications could be relevant for this?



# 3.4 Interfacing the Application Layer

# 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

**Definition**

*client process:* process that initiates communication

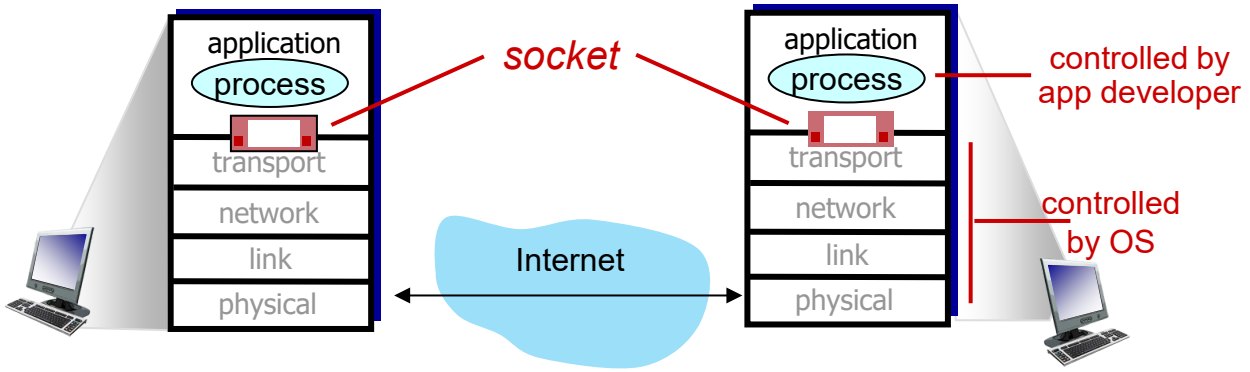
*server process:* process that waits to be contacted



SDU

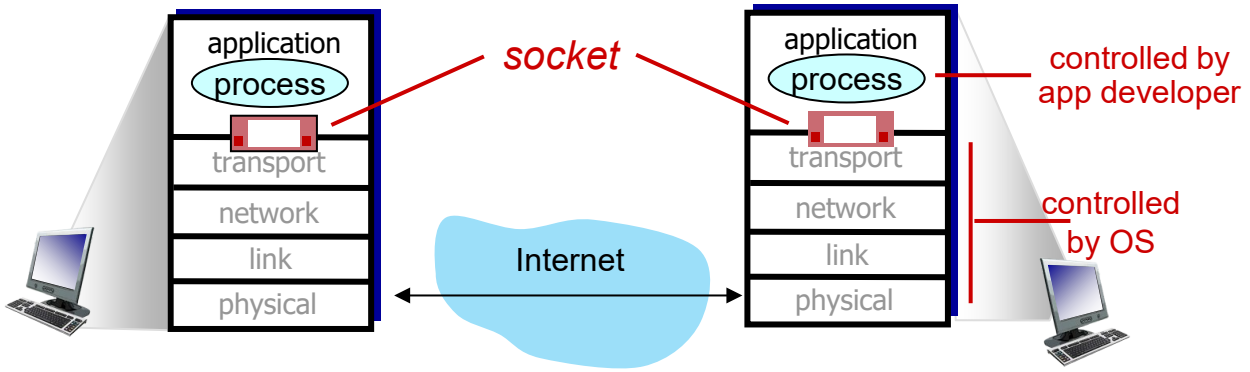
# Sockets

- Process sends/receives messages to/from its socket
- Socket analogous to door
- Two sockets involved: one on each side



# Sockets

- Sending process shoves message out of the door
- Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process




- Receiving process gets message from the door
- Receiving process relies on transport infrastructure on other side of door to deliver message from socket at sending process

# Addressing Processes

- To receive messages, a process must have *identifier*
- Host device has unique 32-bit IP address
- *Q:* Does the IP address of a 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



SDU

# App-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 and allows for interoperability

# What transport service does an app need?

## Reliable data transfer

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

## 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

## Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

## Security

- encryption, data integrity, …

# Transport service requirements: common apps

| application | data loss | throughput | time sensitive |
|---|---|---|---|
| file transfer | 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<br>video:10kbps-5Mbps | yes, 100's ms |
| stored audio/video | loss-tolerant | same as above | yes, few secs |
| interactive games | loss-tolerant | few kbps up | yes, 100' s ms |
| text messaging | no loss | elastic | Depends |

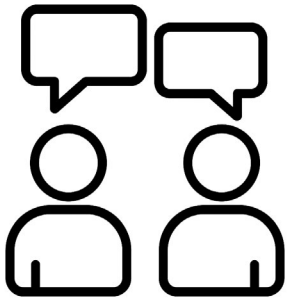
# Internet transport protocols services

**TCP service:**

- Reliable transport between sending and receiving process
- Flow control: sender won't overwhelm receiver
- Congestion control: throttle sender when network overloaded
- Does not provide: timing, minimum throughput guarantee, security
- Connection-oriented: setup required between client and server processes

**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,



Why bother? Why is there a UDP?



SDU

# Applications and their 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 multimedia | HTTP, RTP [RFC 1889] | TCP or UDP |
| Internet telephony | SIP, RTP, proprietary | TCP or UDP |

# Securing TCP

**TCP & UDP**

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

**Transport Layer Security (TLS)**

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

**TLS implemented at App. layer**

- Apps use TLS libraries, that use TCP in turn

**TLS socket API**

- Cleartext passwords sent into socket traverse Internet encrypted
- *See Chapter 8*



SDU

# 3.5 Web an HTTP

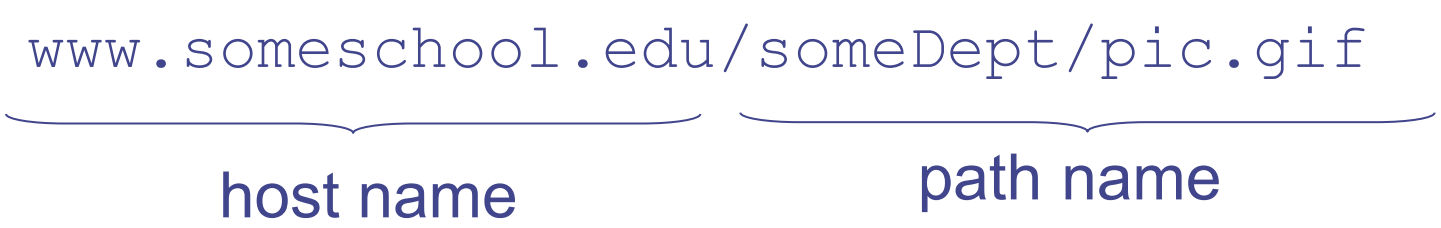


SDU

# Web and HTTP

- *A web page* consists of *objects*
- The object can be HTML file, JPEG image, Java applet, audio file,…
- A web page consists of *base HTML-file* which includes *several referenced objects*

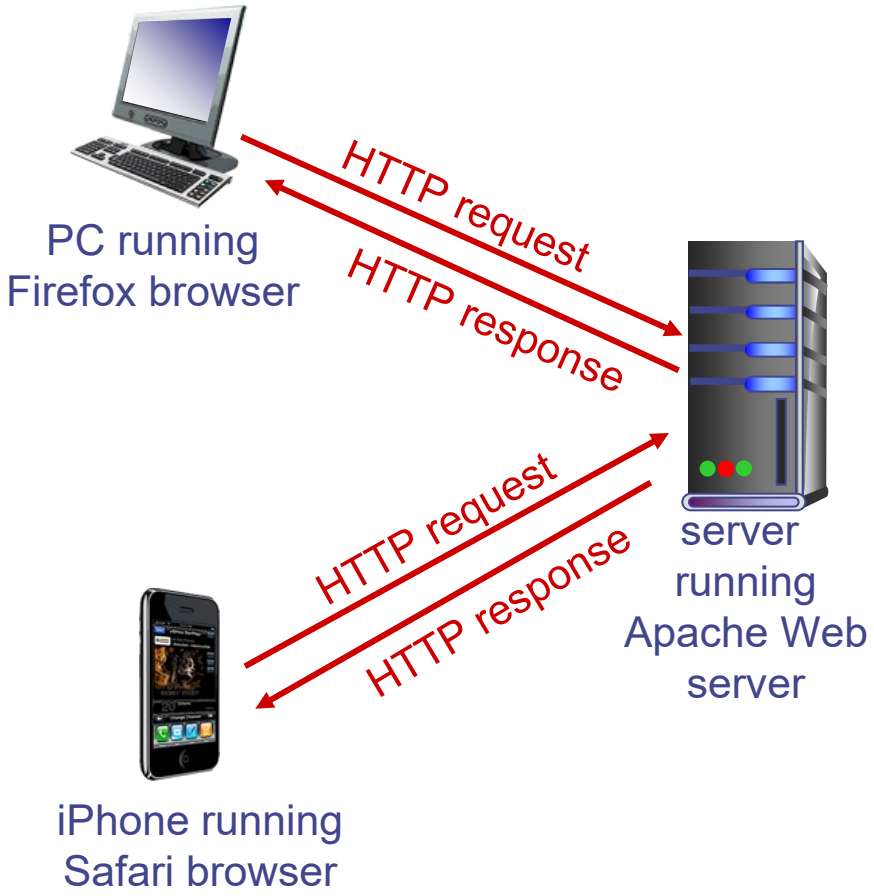
- Each object is addressable by a *URL,* e.g.:



# HTTP Overview

- **H**yper**T**ext **T**ransfer **P**rotocol
  - Text with hyperlinks
  - Web's application layer protocol

- client/server model
  - *client:* browser that requests, receives and displays Web objects
  - *server:* Web server sends objects in response to requests
  - Both use the HTTP protocol



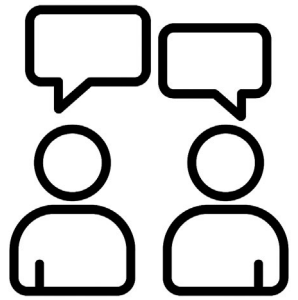
# HTTP overview (continued)

- 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
  - A state for http can be achieved through *cookies*

Maintain a state is complex!

- Past history (state) must be maintained
- If server/client crashes, their views of “state” may be inconsistent, must be reconciled



What could a state be for http?

# HTTP connections

## Non-persistent HTTP

- Max one object sent over the TCP connection
  - connection then closed

## Persistent HTTP

- Multiple objects can be sent over a single TCP connection between client and server

# Non-persistent HTTP

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



SDU

# Non-persistent HTTP (cont.)

4. HTTP server "initiates" closing of the 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

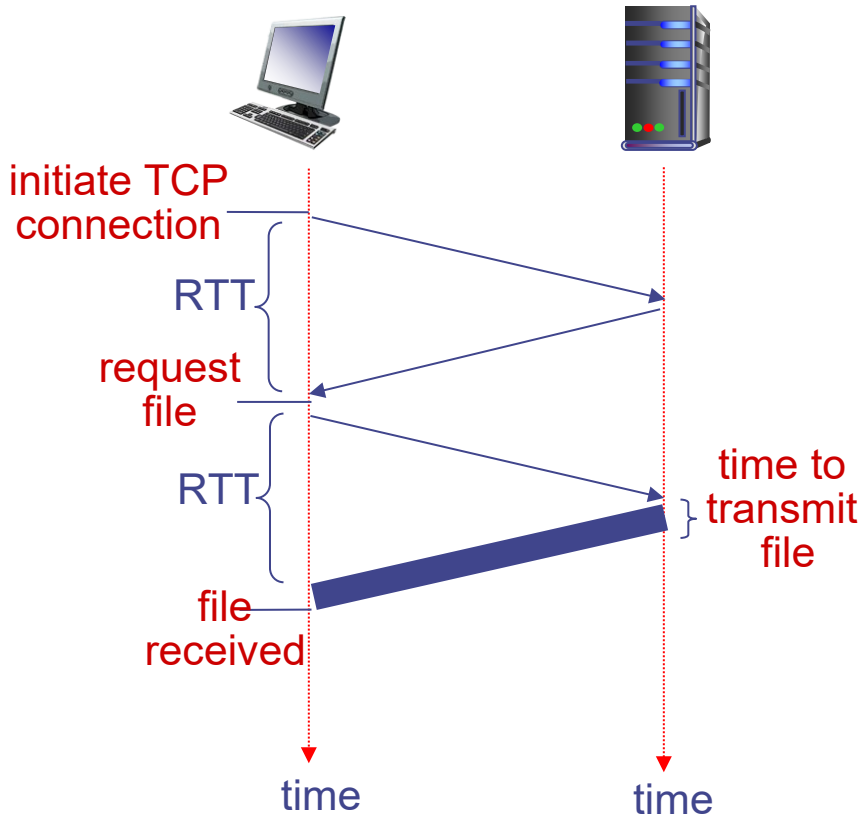
# Non-persistent HTTP: response time

RTT (definition): 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

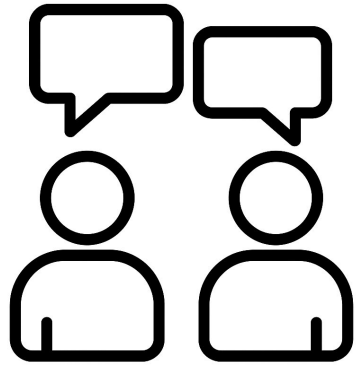
- Response time = 2RTT+ file transmission time



# Persistent HTTP

*Non-persistent HTTP issues:*

- Requires 2 RTTs per object
- OS overhead for *each* TCP connection
- Browsers often open parallel TCP connections to fetch referenced objects



Why can multiple parallel connections be problematic?

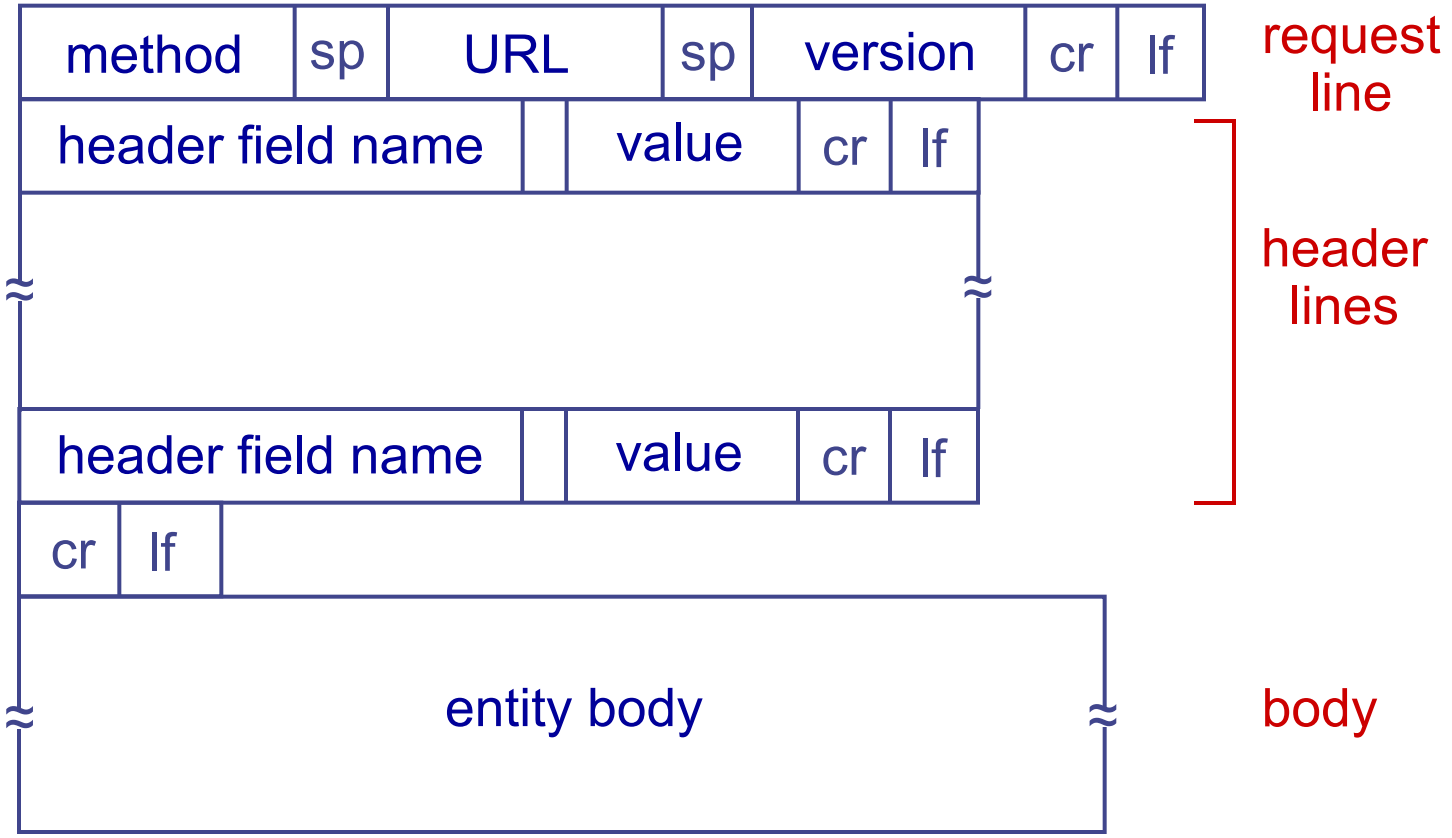
*Persistent HTTP:*

- 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
- Just one RTT for all the referenced objects



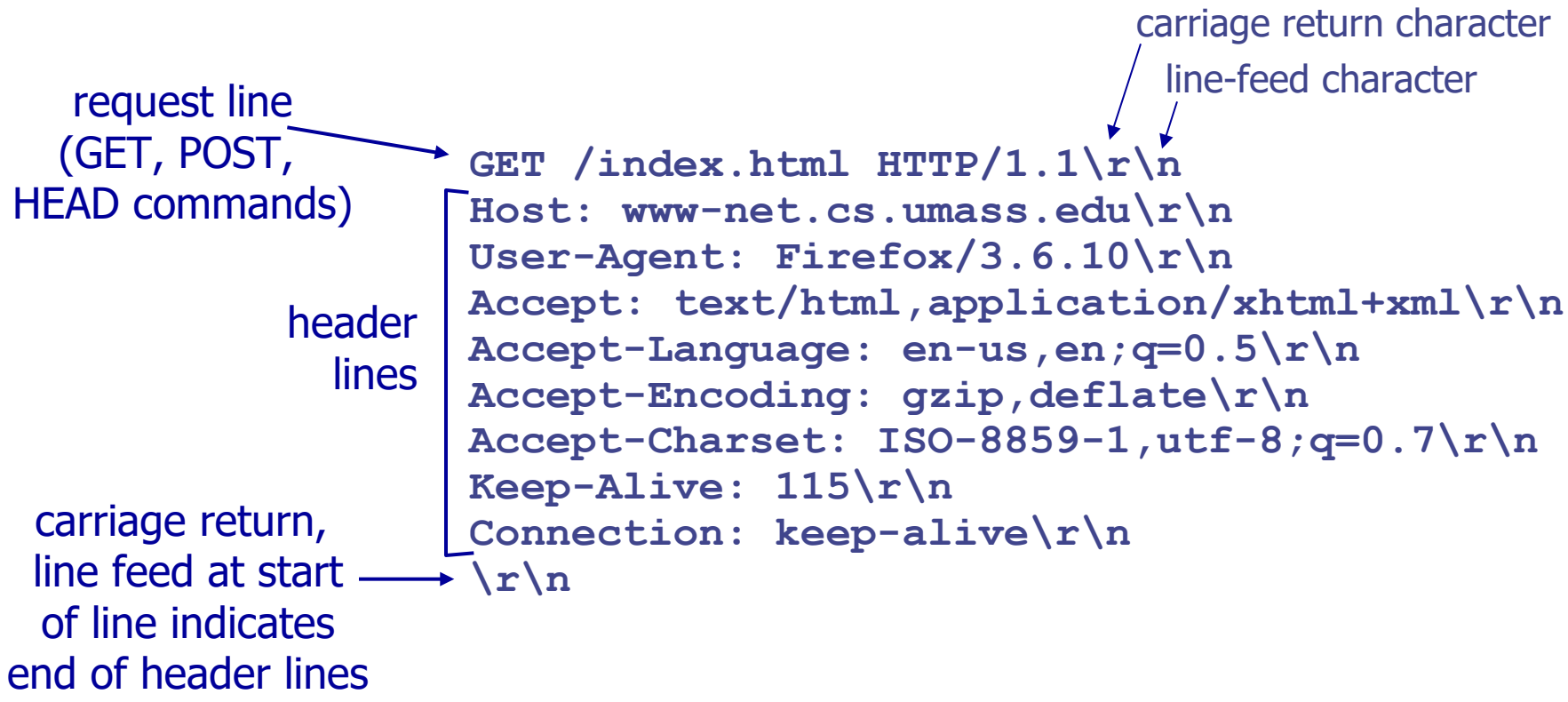
SDU

# HTTP request message: general format

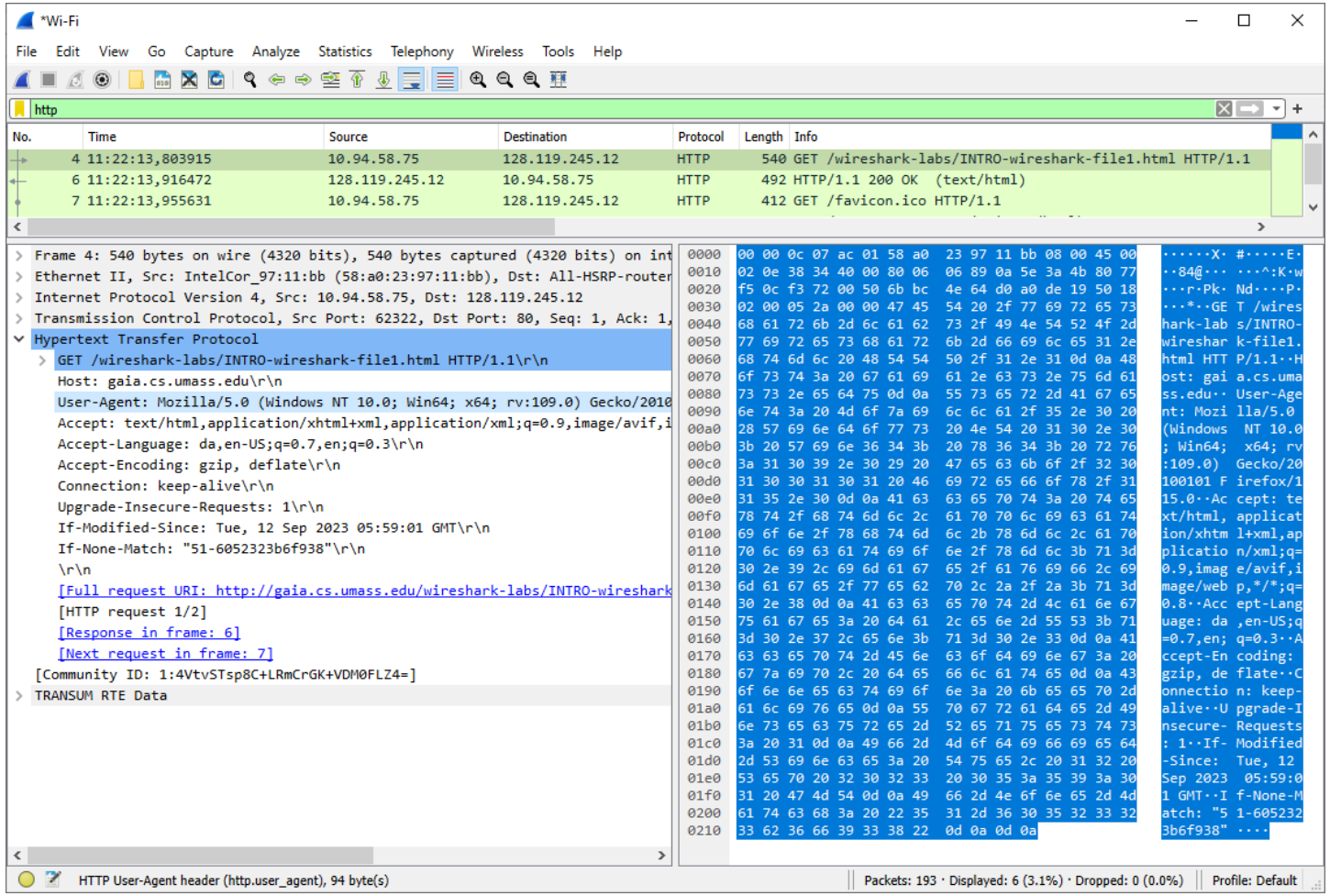


# HTTP request message

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



# HTTP GET seen in wireshark.



# HTTP response status message

Protocol status line:
- status code
- status phrase

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

# 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 msg

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

**400 Bad Request**
  - request msg not understood by server

**404 Not Found**
  - requested document not found on this server

**505 HTTP Version Not Supported**



SDU

# Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - Asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

**FYI: Since June 22 the standard is HTTP3!**



SDU

# Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line: `www.somesite.com/search?room&U181`



SDU

# Conditional GET

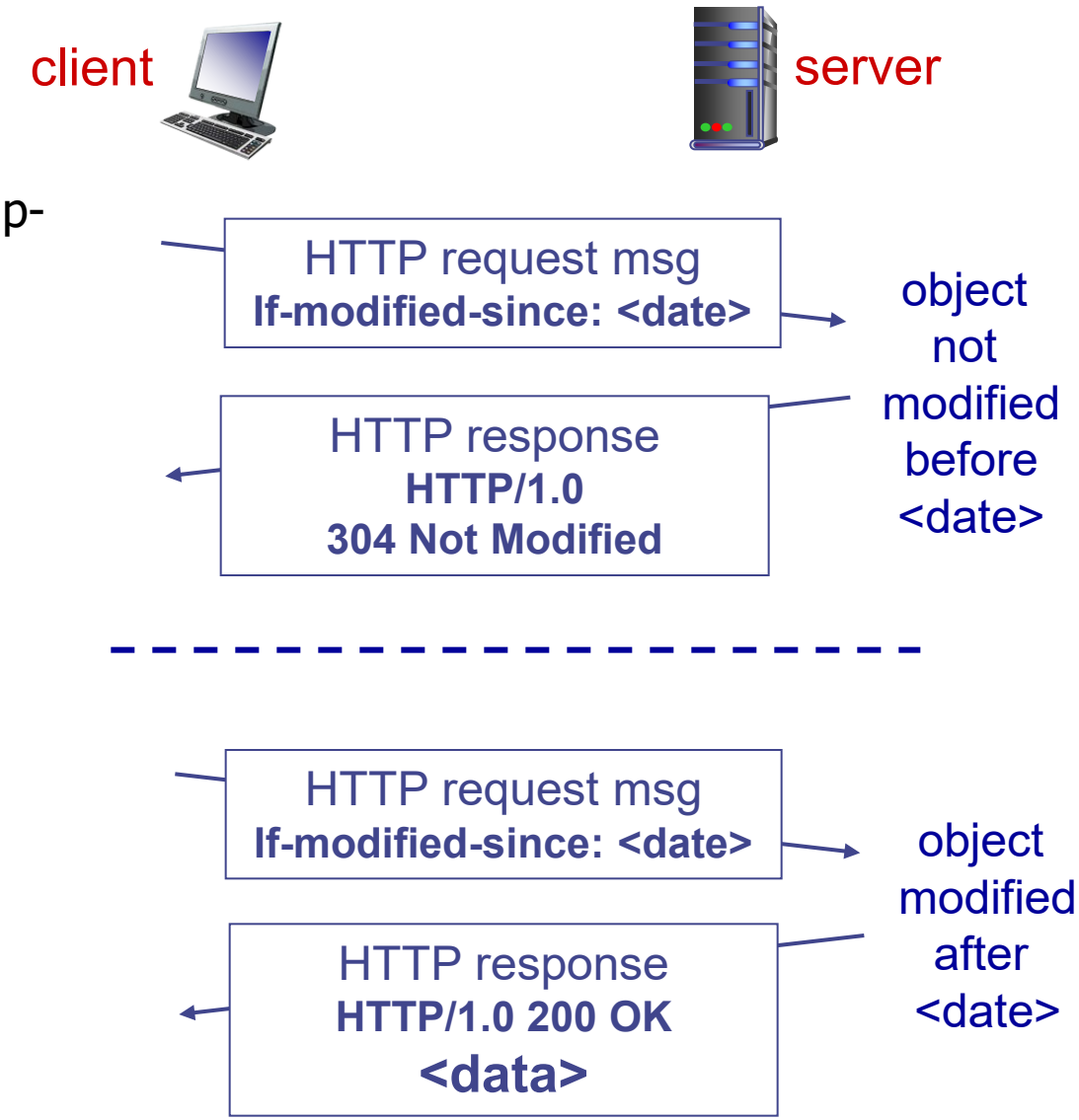
- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization

- *Cache:* specify date of cached copy in HTTP request

```
If-modified-since: <date>
```

- *Server:* response contains no object if cached copy is up-to-date:

```
HTTP/1.0 304 Not Modified
```



# Telnet as Web Client

1. Telnet to your favorite Web server:

```
telnet gaia.cs.umass.edu 80
```

- opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu.
- anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

- by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

# A Note on E-mail

- We will not be covering it in class!
- However:
  - SMTP (Simple Mail Transfer Protocol)
    Purpose: Sending emails from a client to a server or between servers.
  - POP3 (Post Office Protocol v3)
    Purpose: Downloading emails from the server to a local device.
  - IMAP (Internet Message Access Protocol)
    Purpose: Reading and managing emails stored on a remote server

- Telnet could be used to send e-mails, but is mostly closed due to security reasons.

# Wireshark Labs

- Lab 2: HTTP
- There are no demands for hand-ins of the Labs, but do the journals described in the labs anyway, for discussions at the class and with your classmates.
- Tip: Be aware of the note about the IF-MODIFIED-SINCE issue with some browsers!



SDU

For next time:

- Read Kurose & Ross. The rest of chapter 2, until page 198.
- Finish the Wireshark Lab 1 and 2.