



CAN201 – Lecture 2

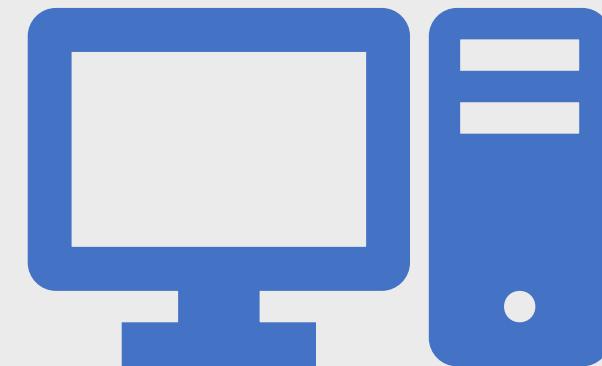
Module Leader: Dr. Fei Cheng Dr. Gordon Boateng

Application layer: overview

- Principles of network applications
- Web and HTTP/1/2/3
- The Domain Name System DNS
- Socket programming with UDP and TCP

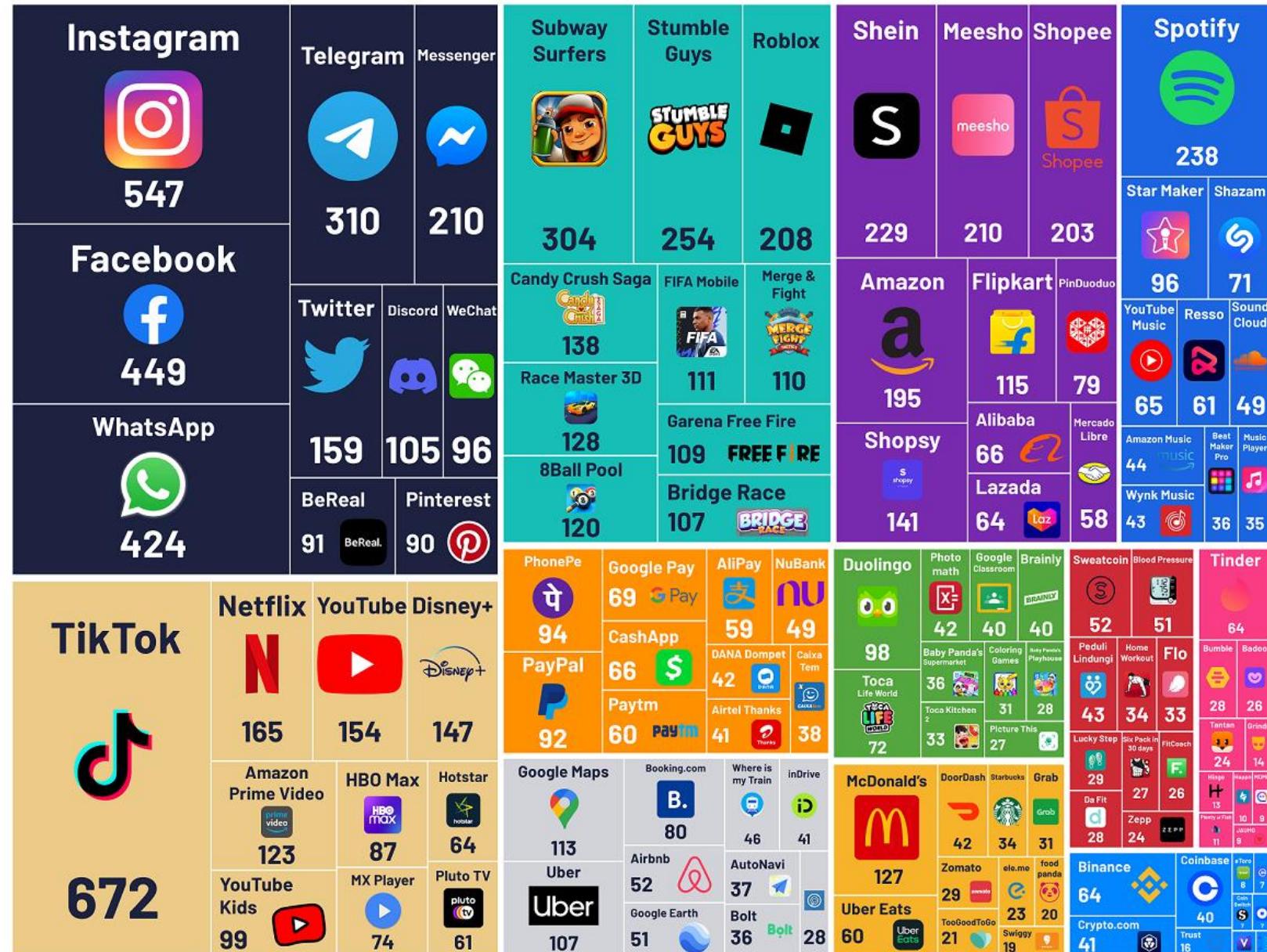
Lecture 2 – Application Layer (1)

1. Introduction to Application Layer
2. Principles of Network Applications
3. Web Application - HTTP

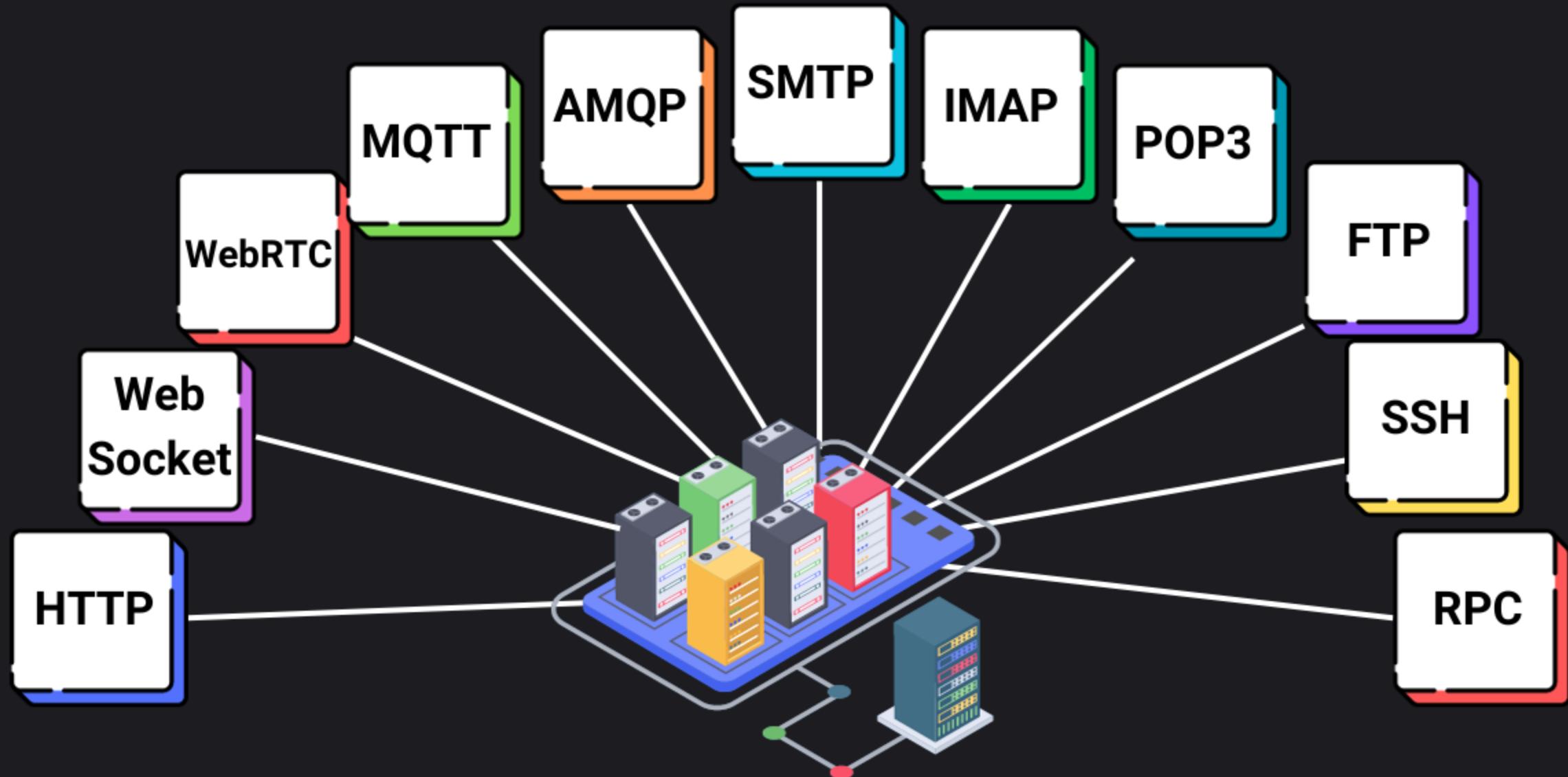


Some network apps

- Social media: fb / rednote / X
- 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., Tencent Meeting)
- ChatGPT
- remote login: ssh
- ...



APPLICATION PROTOCOLS



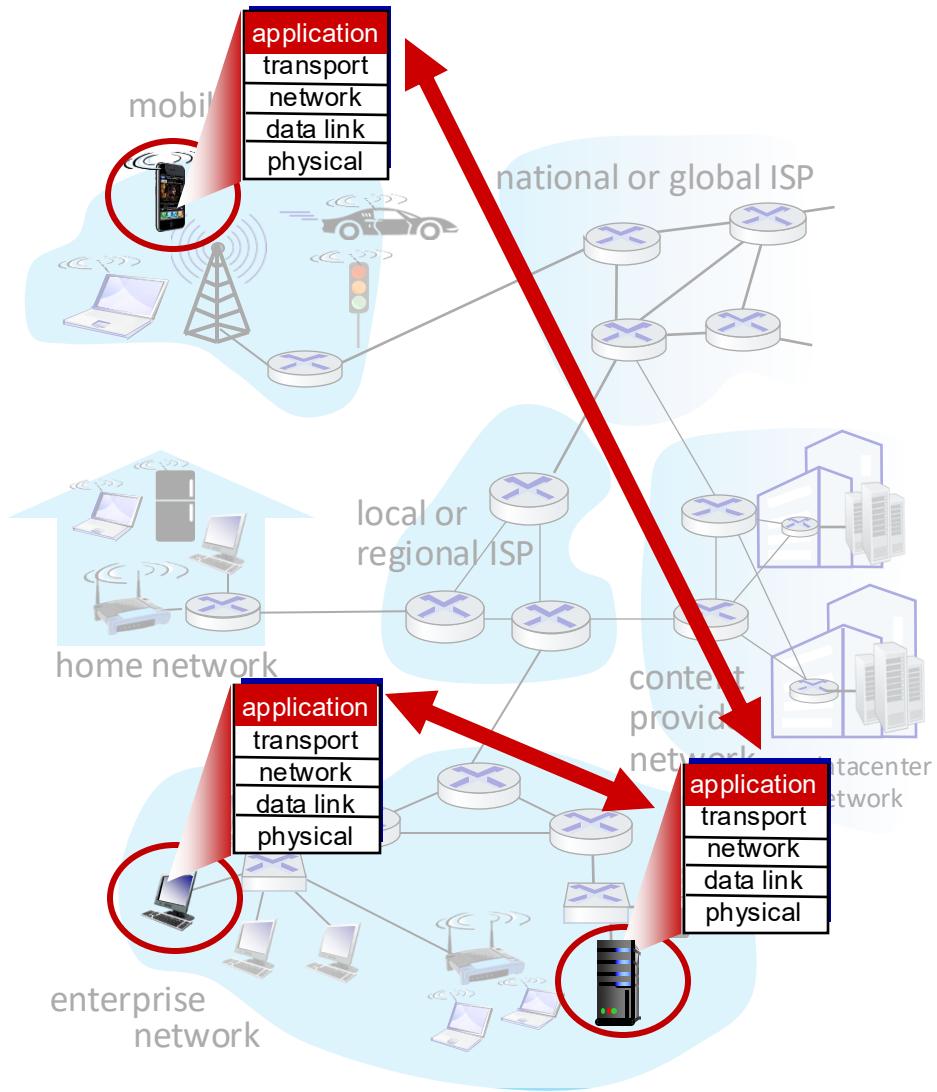
Creating a network app

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



Yes



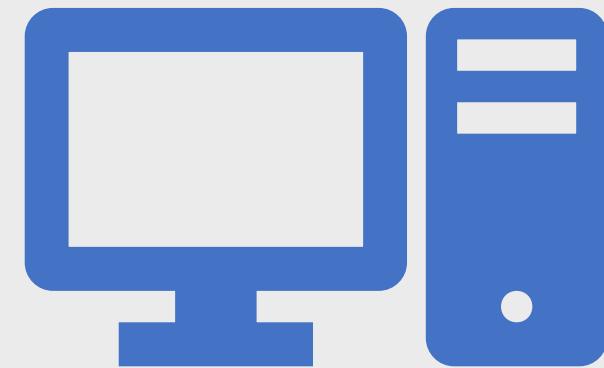
No



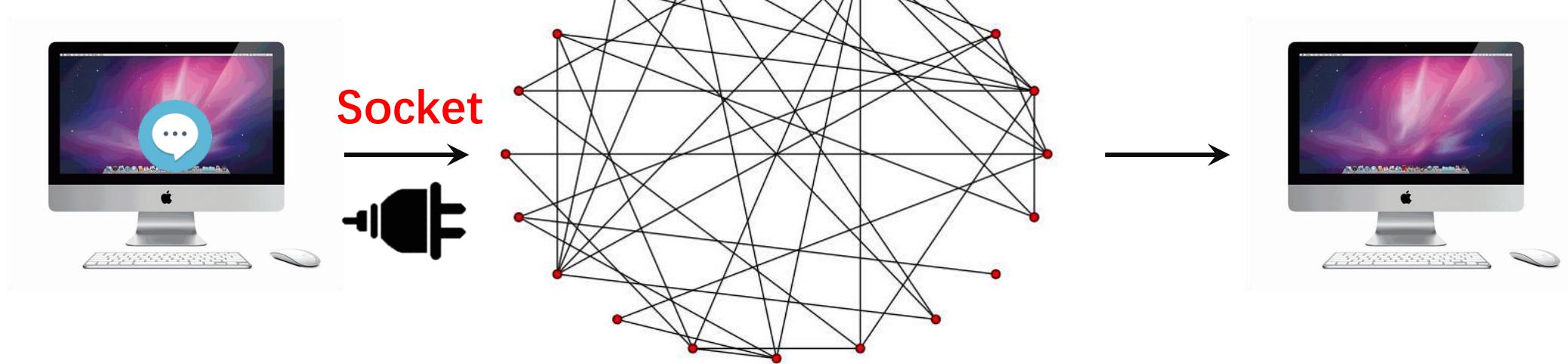
openWRT

Lecture 2 – Application Layer (1)

- 1. Introduction to Application Layer**
- 2. Principles of Network Applications**
- 3. Web Application - HTTP**

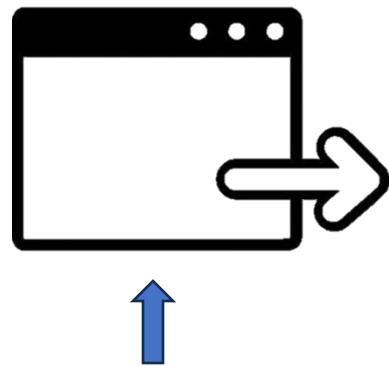


What do we use to deliver messages?



What do we use to deliver messages?

Host

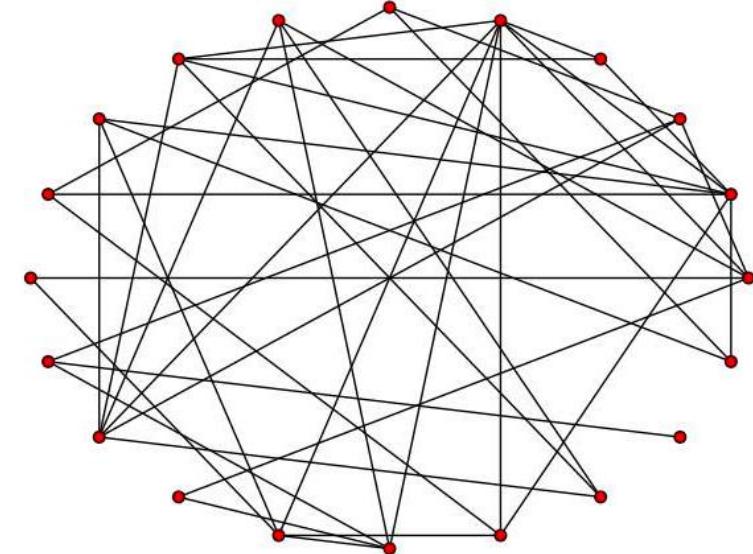


A program(process)

Sockets



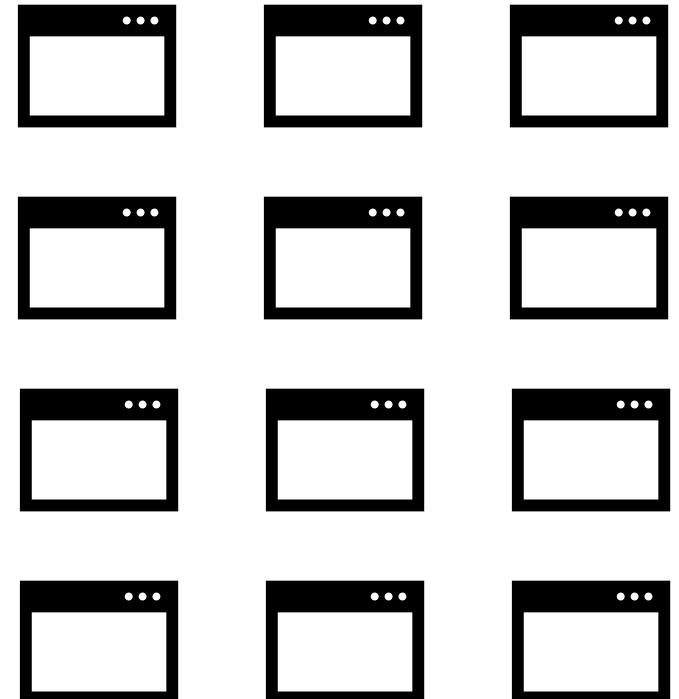
Network



Processes communicating

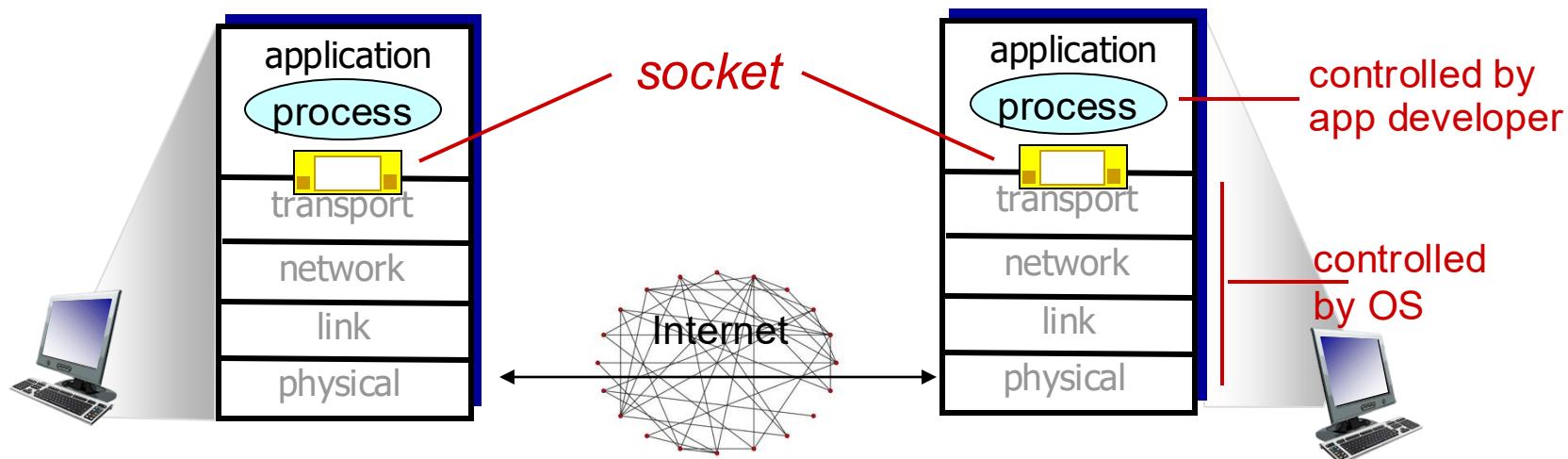
Process: program running within a host

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



Sockets

- Process sends/receives messages to/from its **socket**
- Socket analogous to door
 - Sending process "push" message out of the door
 - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
 - 2 sockets involved: one on each side



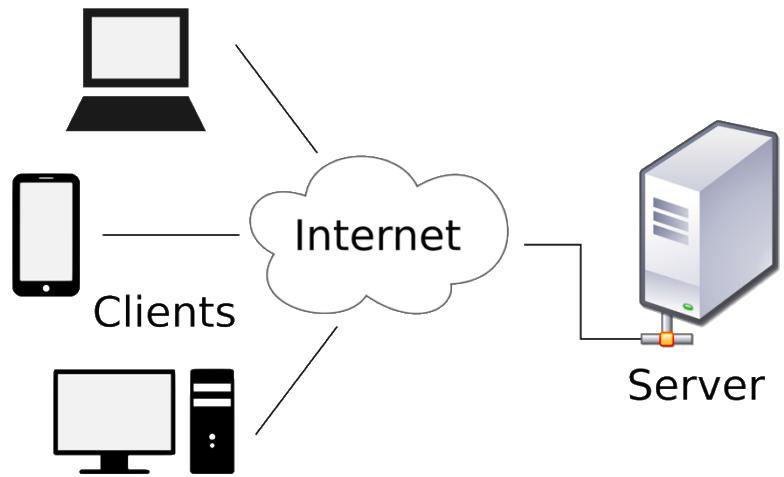
What do we need next?

Addressing processes

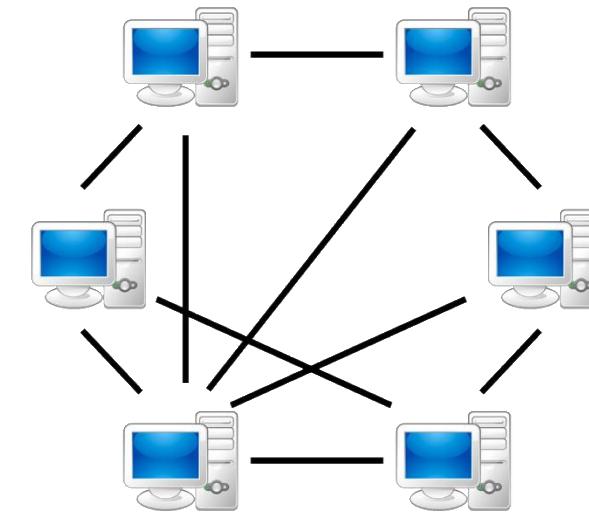
- To receive messages, process must have *identifier*
- Host device has unique 32-bit IPv4 and/or 128-bit IPv6
- Process network identifier:
 - IPv4:port 192.168.1.100:80
 - [IPv6]:port [240e:3a1:4cb1:69d0:f40c:4269:74a2:7ea3]:80



Paradigms for network Apps



Client-server

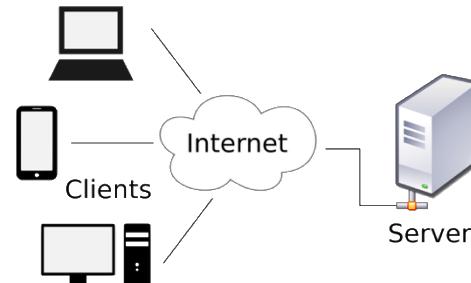


Peer-to-peer

Client-server Paradigm

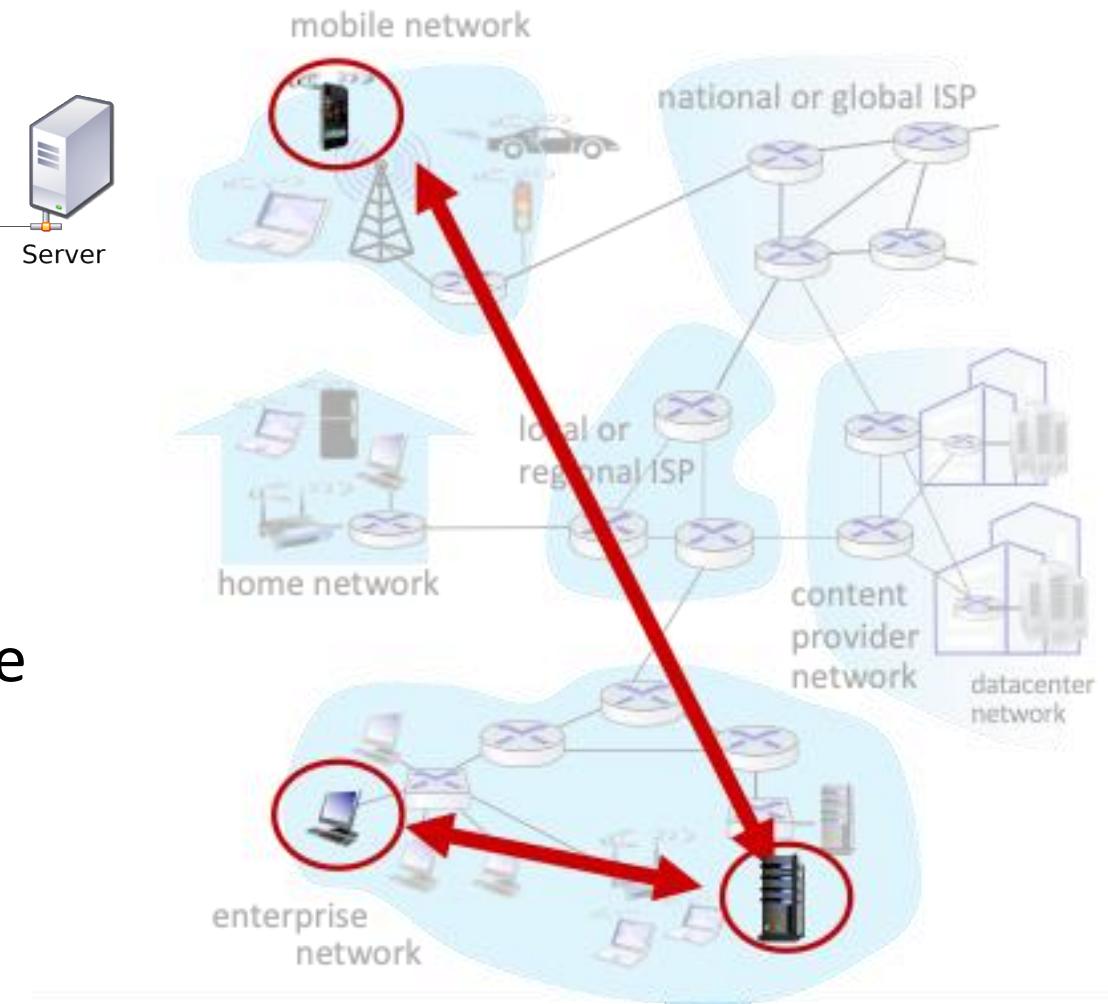
- **Server**

- Always-on host
- Permanent IP address
- Often in data centers, for scaling

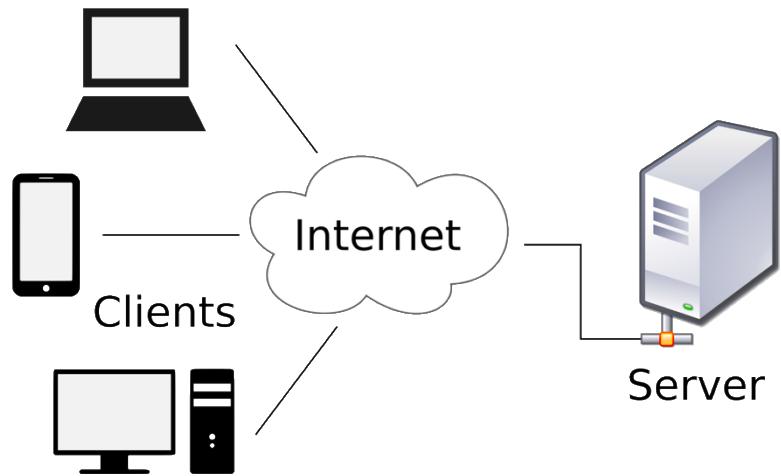


- **Clients**

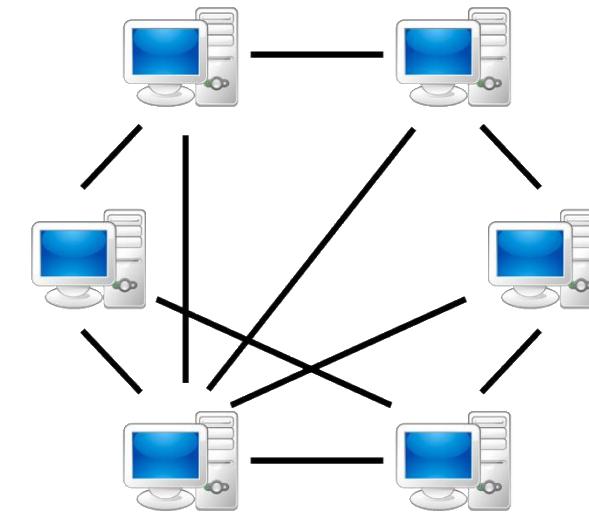
- Contact, communicate with server
- May be intermittently connect to the internet
- May have dynamic IP address
- Do not communicate directly with each other



Paradigms for network Apps



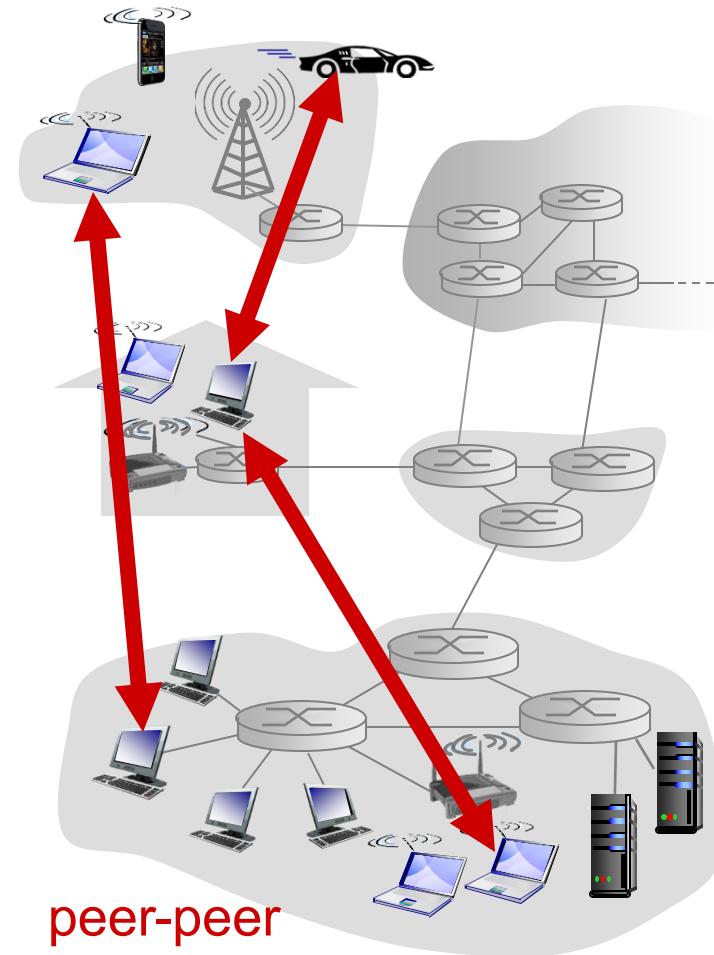
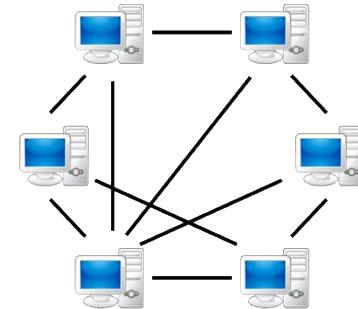
Client-server



Peer-to-peer

P2P Paradigm

- No always-on server is needed
- End systems directly exchange data
- Client process / server process on the same host
- 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
- Maybe dynamic IP addresses
-> complex management
- Q: Did you use Thunder Downloader(迅雷)?
Why does it download so fast?



Try to have your cloud server (optional)

- Most cloud providers provide student discount.
- Ali Cloud, Tencent Cloud...

App-layer protocol defines

- **Paradigm :** CS or/and P2P
- **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
- **Message timing:** when and how

Open protocols:

- Defined in RFCs
- e.g., HTTP, SMTP, FTP

Private protocols:

- e.g., Skype, Games, **you own protocols...**

What kinds of transport service do we need?

Transport service requirements

Data Integrity

- 100% reliable data transfer or
- Tolerate some loss

Throughput

- Some apps require minimum amount of throughput to be “effective”
- Others do not require

- Some apps require low delay
- while others not require

Timing

- Some apps require encryption
- Data integrity check

Security

Transport service requirements: common apps

Applications	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 video/audio	Loss-tolerant	Based on quality*	Yes 100 ms
Streaming video/audio	Loss-tolerant	Based on quality*	Yes few second
Interactive games	Loss-tolerant	Few kpbs	Yes 100 ms
Text messaging	No loss	Elastic	Yes or No

* Audio: 5k to 1Mbps, Video: 10kpbs - 10 Mbps or more

Internet transport protocols services

TCP ->



<- UDP

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 offer: timing, minimum throughput guarantee, security**
- **Connection-oriented: setup required between client and server processes**

Internet transport protocols services

UDP service:

- **Unreliable data transfer between sending and receiving process**
- **Does not offer: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,**
- Looks so bad... But...

Why do we still need UDP?

Simple and fast!

Application vs transport protocols

Application	App layer protocol	Underlying transport Protocol
E-mail	SMTP [RFC 2821], POP3, IMAP	TCP
Remote terminal Access	Telnet [RFC 854], SSH	TCP
Web	HTTP [RFC 2616] HTTPS	TCP
File Transfer	FTP [RFC 959], SFTP	TCP
Multimedia	HTTP / RTP [RFC 1889]	TCP or UDP
VoIP	SIP, RTP or proprietary	TCP or UDP

Securing TCP - Secure Sockets Layer - SSL

TCP & UDP

- No encryption
- Cleartext psws -> Internet

SSL

- Provides encrypted TCP connection
- Data integrity
- End-point authentication

SSL is at app layer

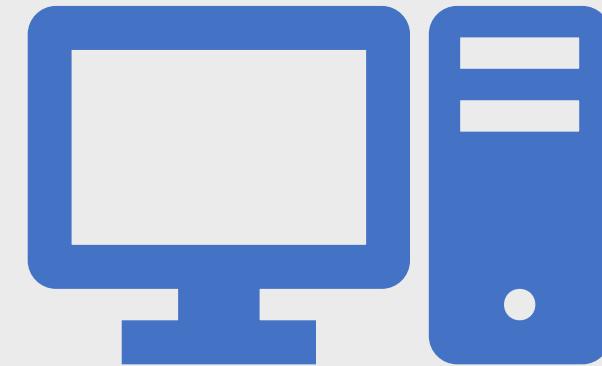
- Apps use SSL libraries, that “talk” to TCP

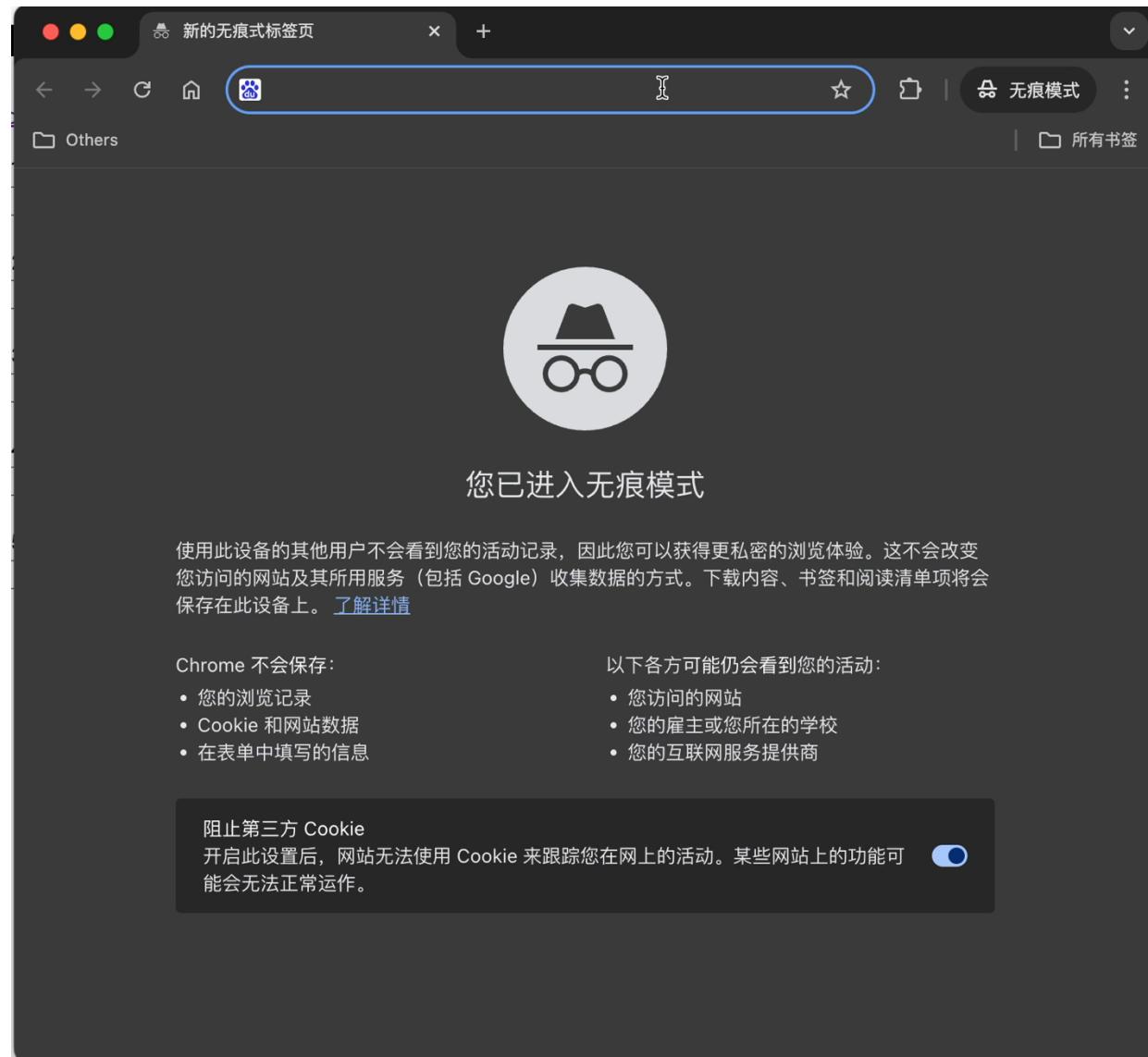
SSL socket API

- Cleartext psw -> encrypted psw -> Internet
- Lecture 11/12 will talk more

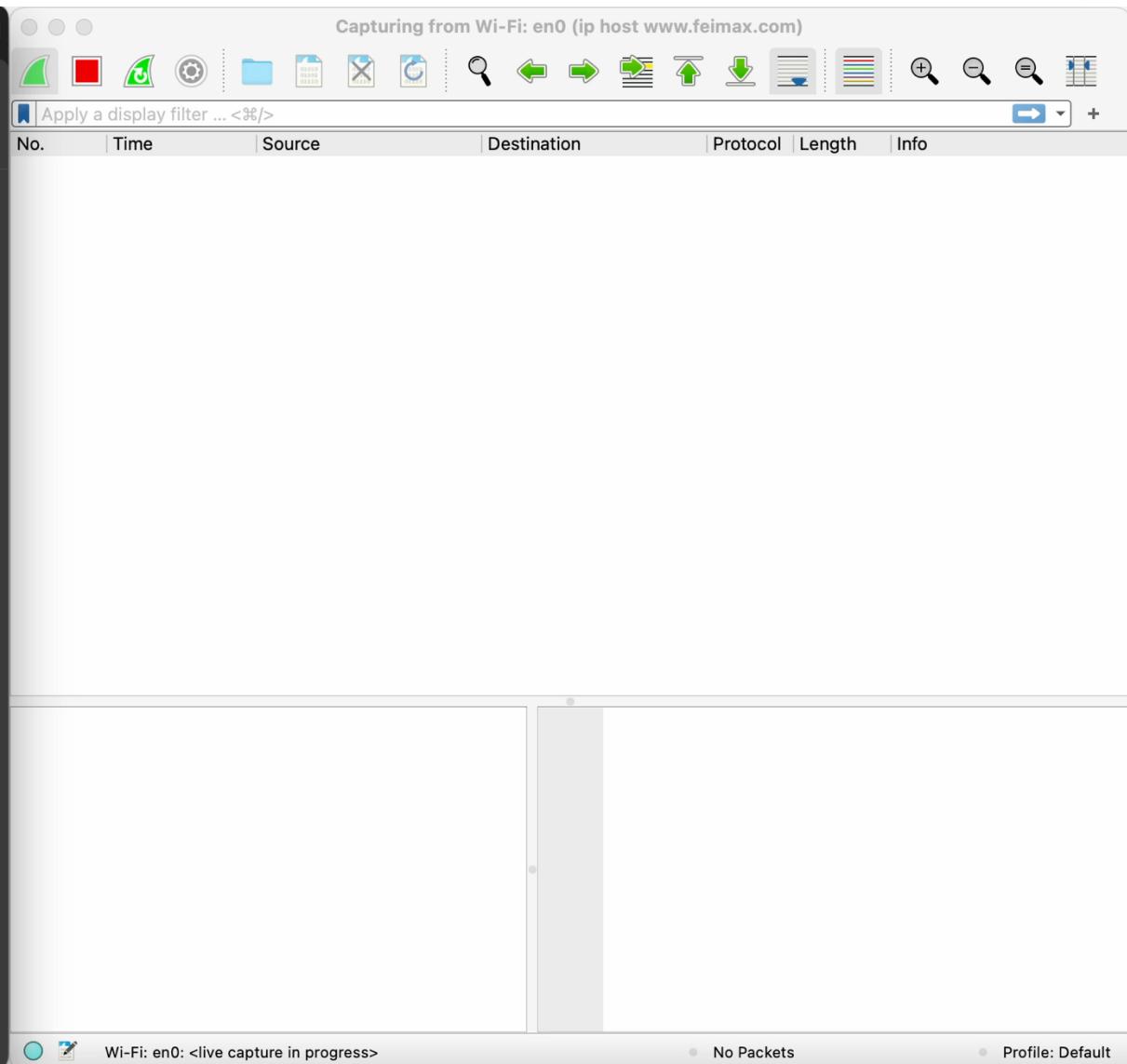
Lecture 2 – Application Layer (1)

1. Introduction to Application Layer
2. Principles of Network Applications
3. Web Application - HTTP





Chrome



Wireshark

Welcome to Feimax's page—a simple and pure HTML site.

Welcome to my little corner of the world! Here, I share snippets of my life and heartfelt reflections.

This is a photo I took yesterday (2024-09-17) of a very big moon:



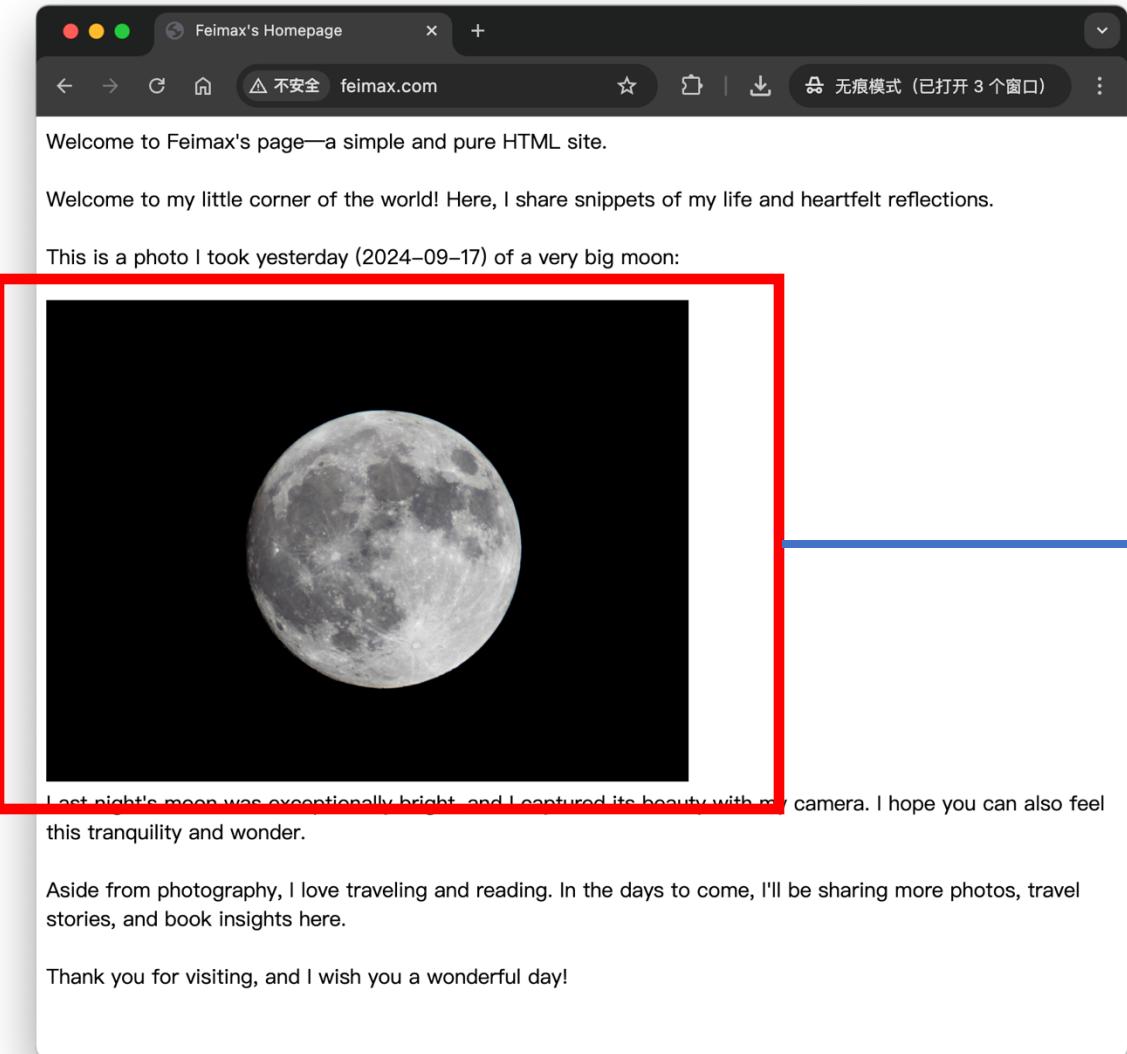
Last night's moon was exceptionally bright, and I captured its beauty with my camera. I hope you can also feel this tranquility and wonder.

Aside from photography, I love traveling and reading. In the days to come, I'll be sharing more photos, travel stories, and book insights here.

Thank you for visiting, and I wish you a wonderful day!

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Feimax's Homepage</title>
</head>
<body>
    Welcome to Feimax's page—a simple and pure HTML site.<br><br>
    Welcome to my little corner of the world! Here, I share snippets of my life and hear...
    This is a photo I took yesterday (2024-09-17) of a very big moon:<br><br>
    <img src='20240917.jpg' style="width: 500px;" />
</br>
    Last night's moon was exceptionally bright, and I captured its beauty with my camera...
    Aside from photography, I love traveling and reading. In the days to come, I'll be s...
    Thank you for visiting, and I wish you a wonderful day!
</body>
```

HTML code (response)



HTML code (response)

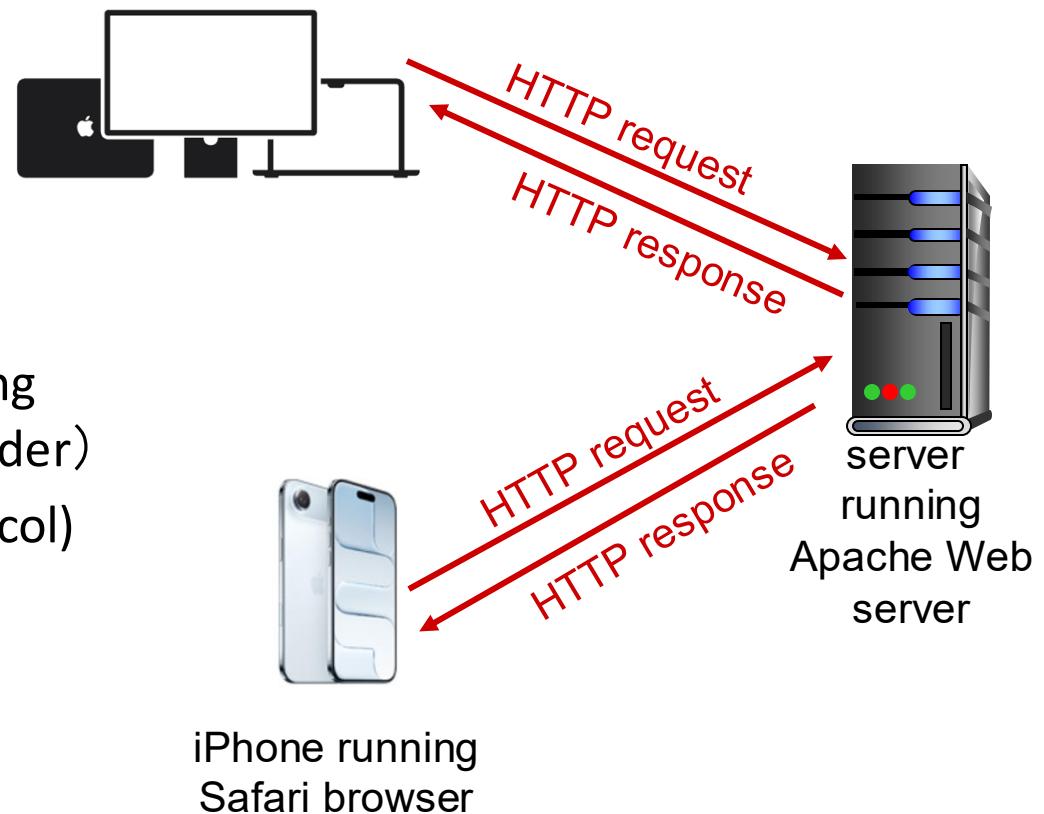
Page

Features of HTTP (1.1)

HTTP: hypertext transfer protocol

(超文本传输协议)

- Application layer protocol
- Client/server model
 - *Client*: browser that requests, receives, (using HTTP protocol) and show Web objects (Render)
 - *Server*: Web server sends (using HTTP protocol) objects in response to requests



Features of HTTP (1.1)

Uses TCP:

1. Client initiates TCP connection (creates socket) to server, port 80(443 for https)
2. Server accepts TCP connection from client
3. HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
4. TCP connection closed

HTTP is “stateless”

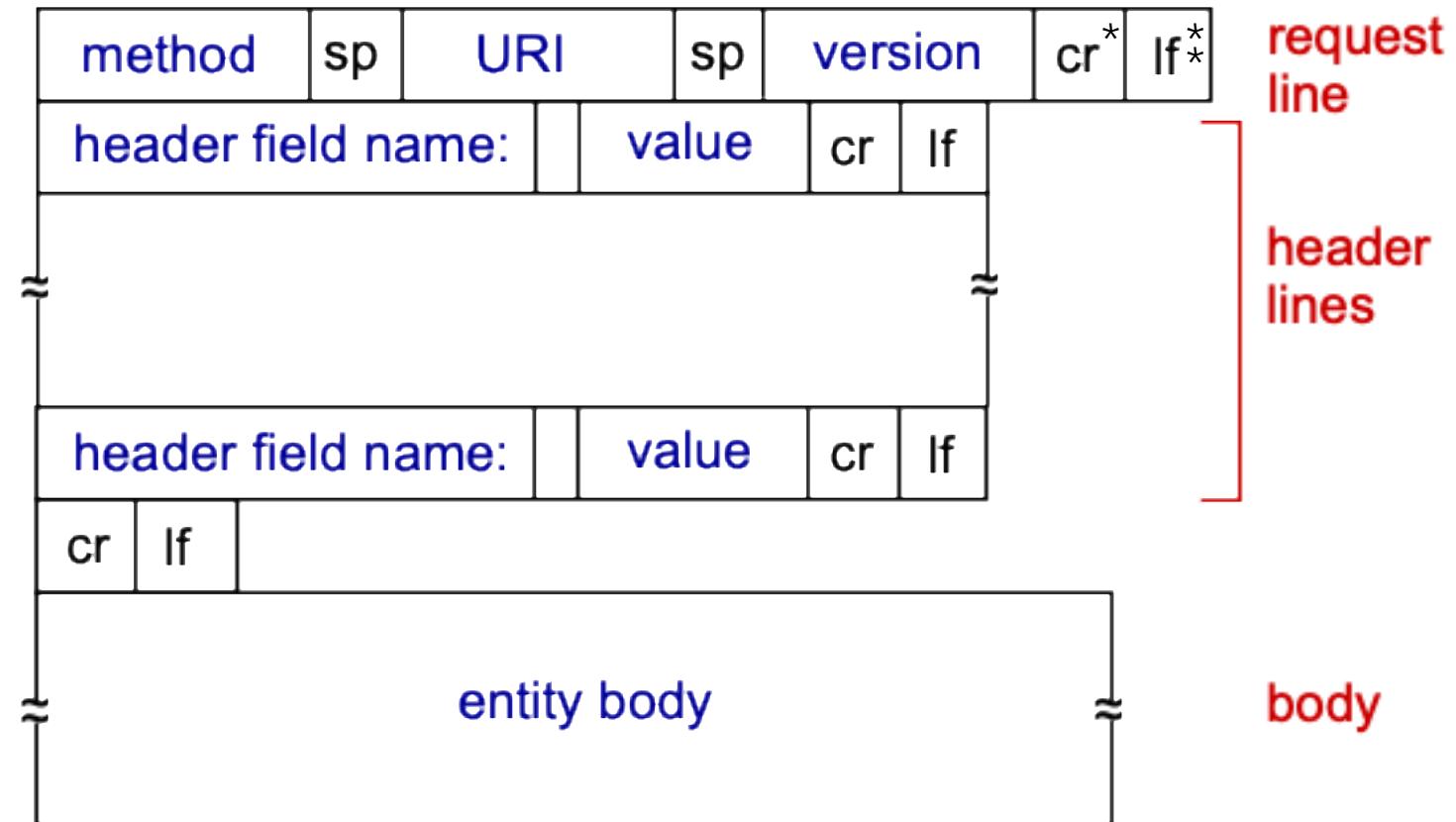
- server maintains no information about past client requests

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 request message: general format

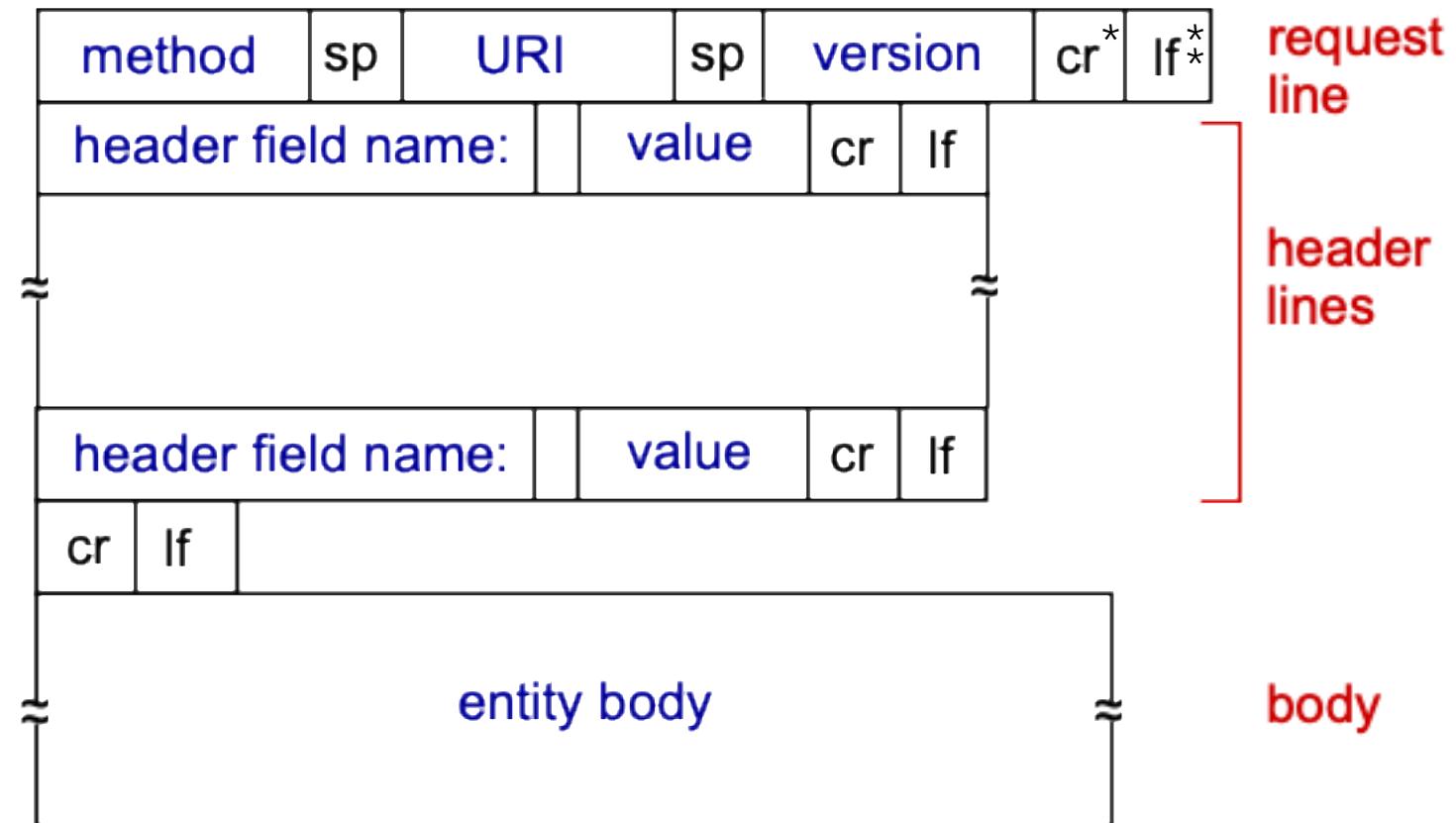


*carriage return character 回车

** line-feed character 换行

Note: No IP address is included in the HTTP request. Why?

HTTP request message: general format

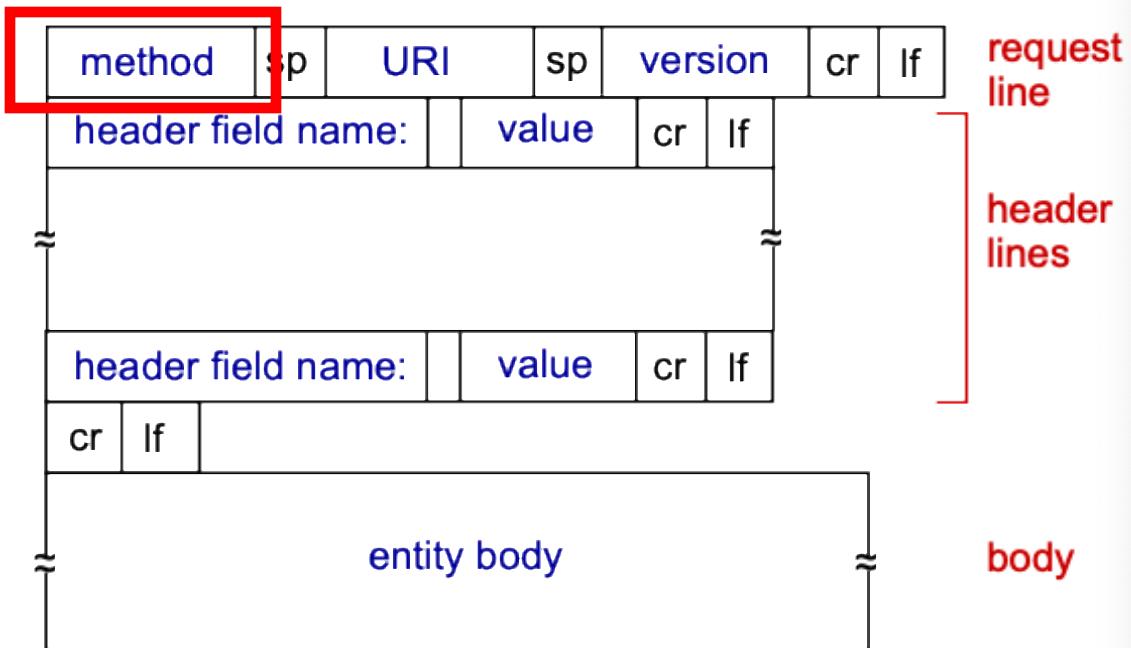


*carriage return character 回车

** line-feed character 换行

Note: No IP address is included in the HTTP request. Why?

HTTP request message: general format



Wireshark · Packet 20 · Wi-Fi: en0 (ip host www.feimax.com)

Frame 20: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits) on interface en0 at 10:00:00:00:00:00 [eth0] at 10:00:00:00:00:00 [eth0]

Ethernet II, Src: Apple_74:26:5d (f4:d4:88:74:26:5d), Dst: Tp-LinkT_0b:d8:5c (00:0c:29:0b:d8:5c)

Internet Protocol Version 4, Src: 192.168.0.24, Dst: 106.14.20.246

Transmission Control Protocol, Src Port: 59244, Dst Port: 80, Seq: 1, Ack: 1, Len: 489

HyperText Transfer Protocol

GET / HTTP/1.1\r\nHost: www.feimax.com\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\nAccept-Encoding: gzip, deflate\r\nAccept-Language: zh\r\n\r\n

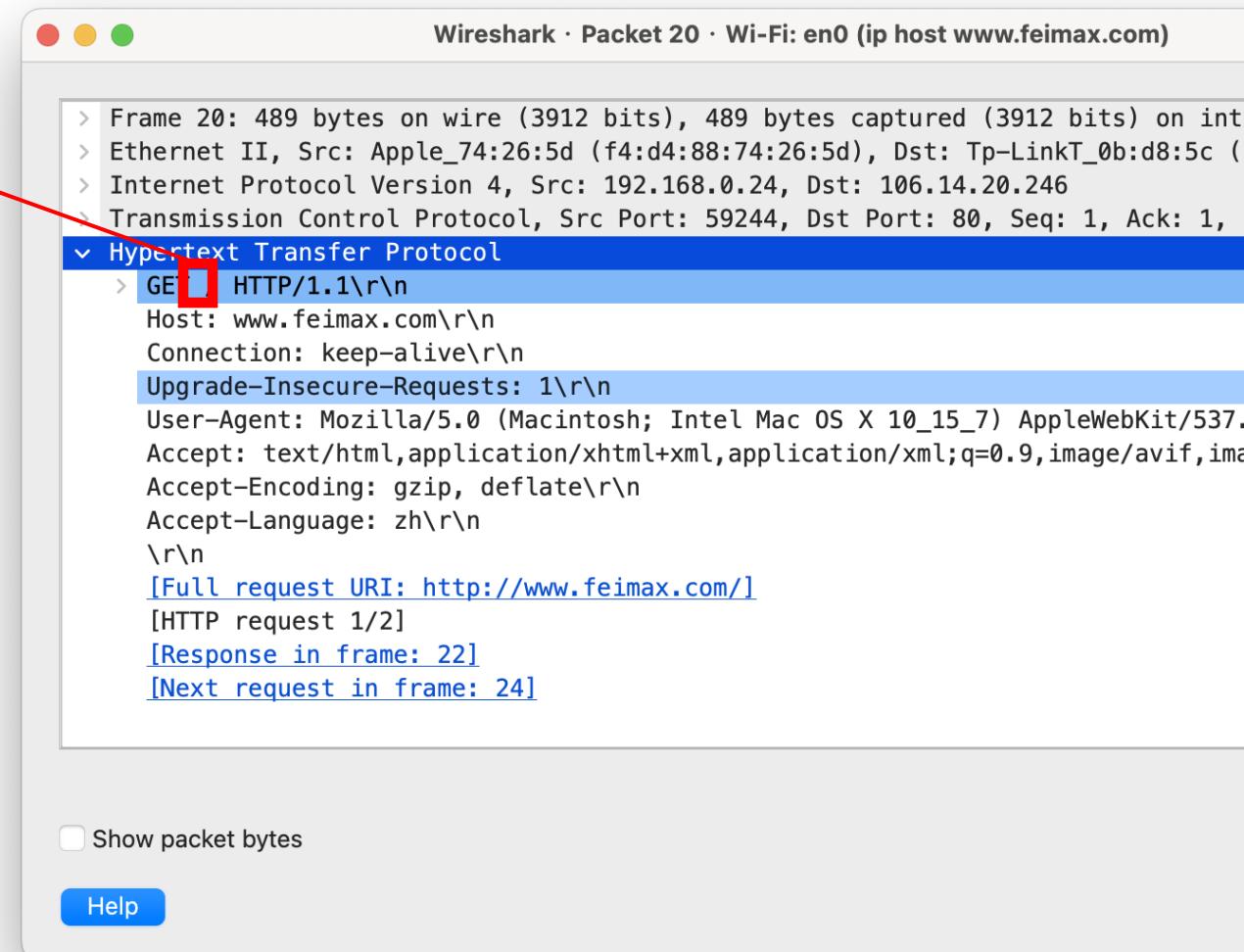
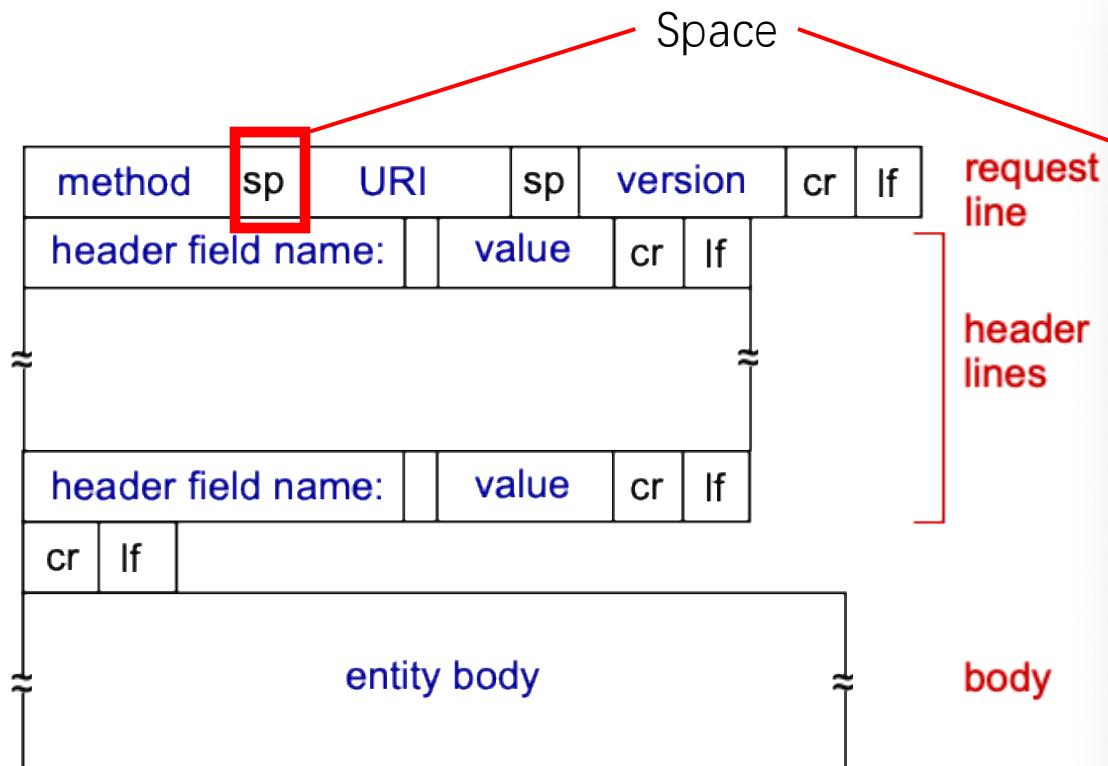
[Full request URI: http://www.feimax.com/]
[HTTP request 1/2]
[Response in frame: 22]
[Next request in frame: 24]

Show packet bytes

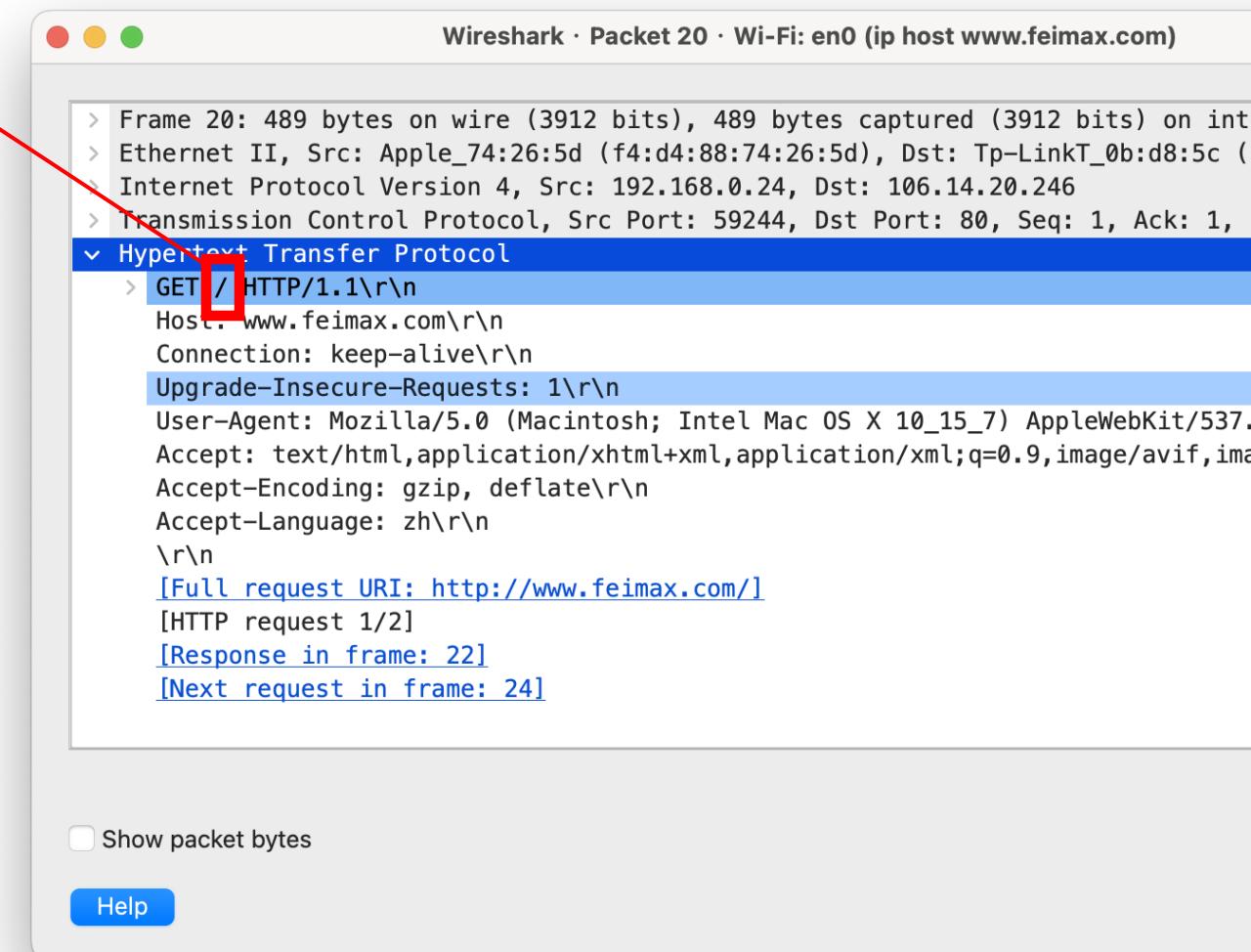
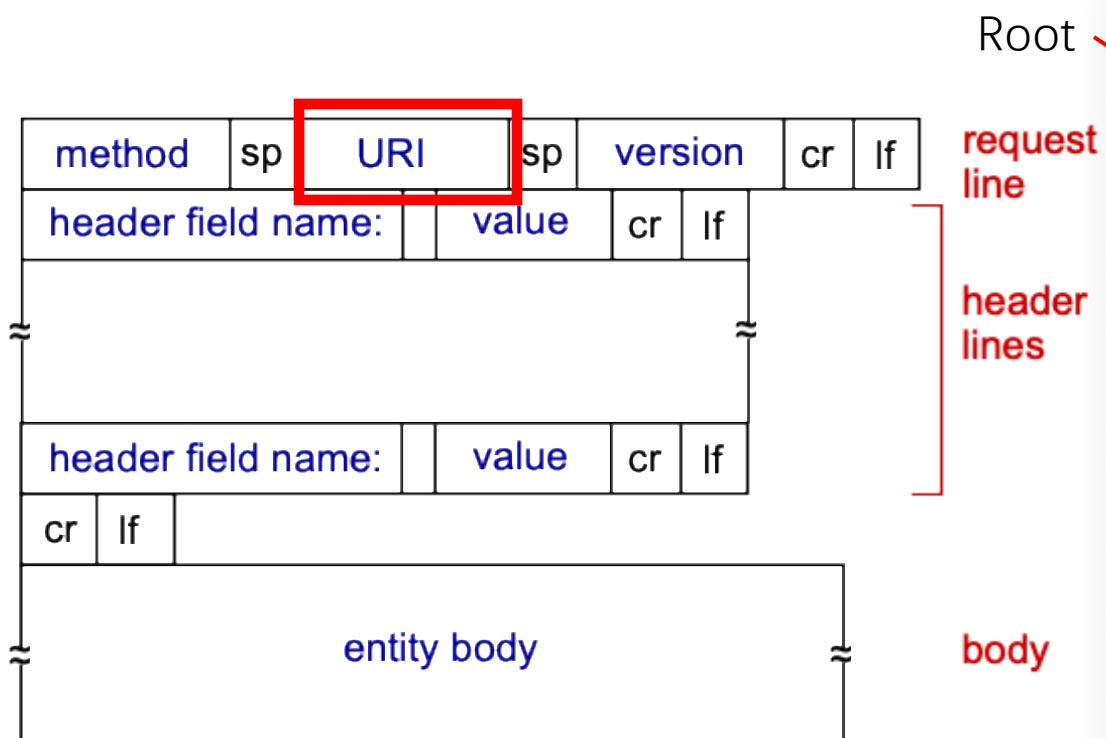
Help

This screenshot shows the Wireshark interface displaying an HTTP request. The request line is highlighted with a red box. The method is GET, the URI is /, and the version is HTTP/1.1. The host is www.feimax.com. The request includes standard headers such as Connection, Upgrade-Insecure-Requests, User-Agent, Accept, Accept-Encoding, and Accept-Language. The body of the request is empty.

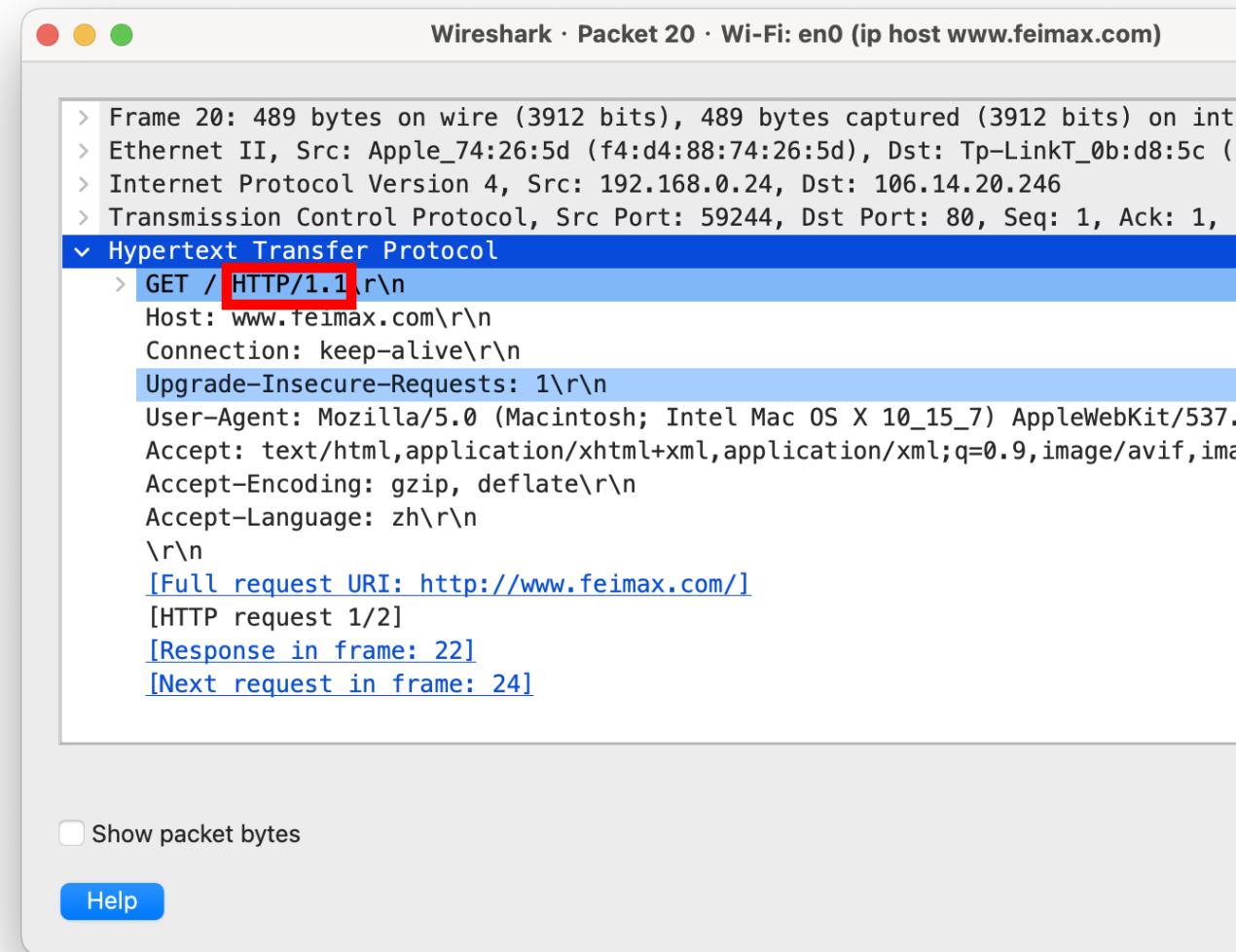
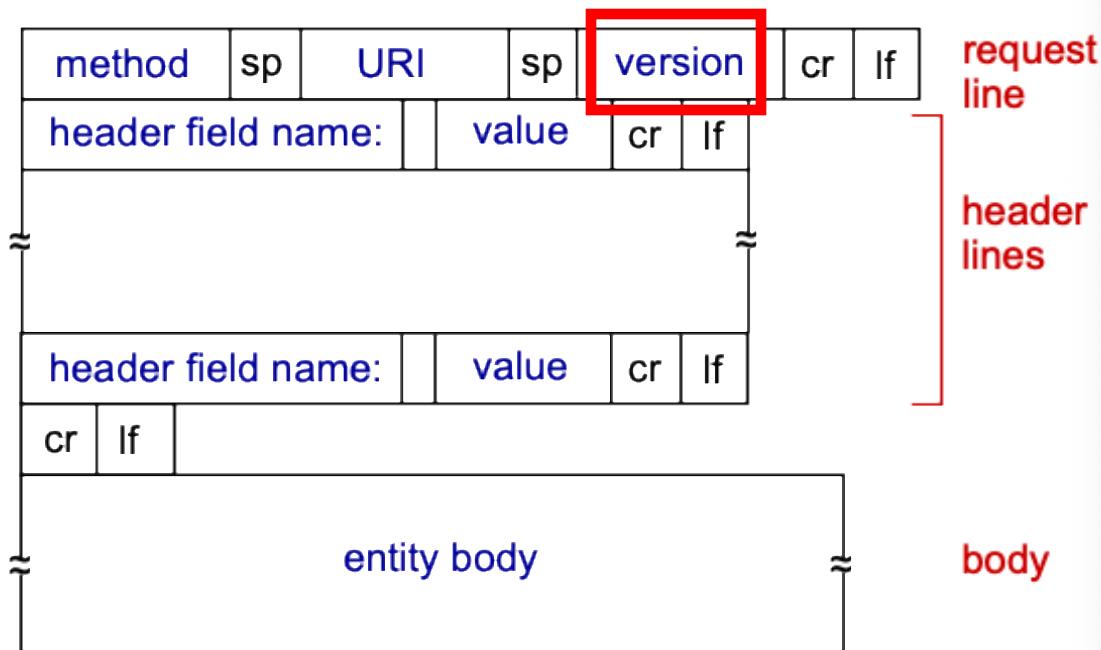
HTTP request message: general format



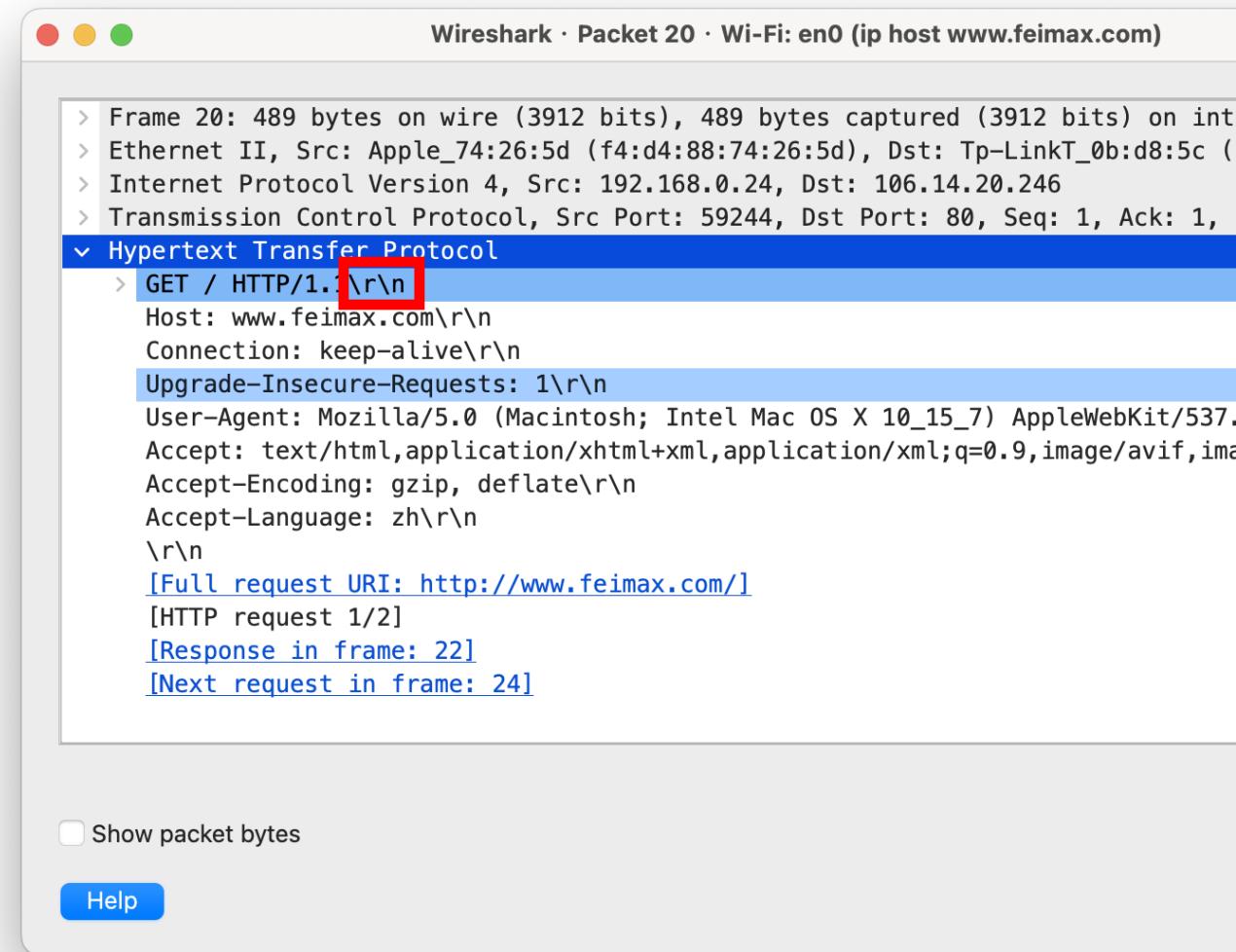
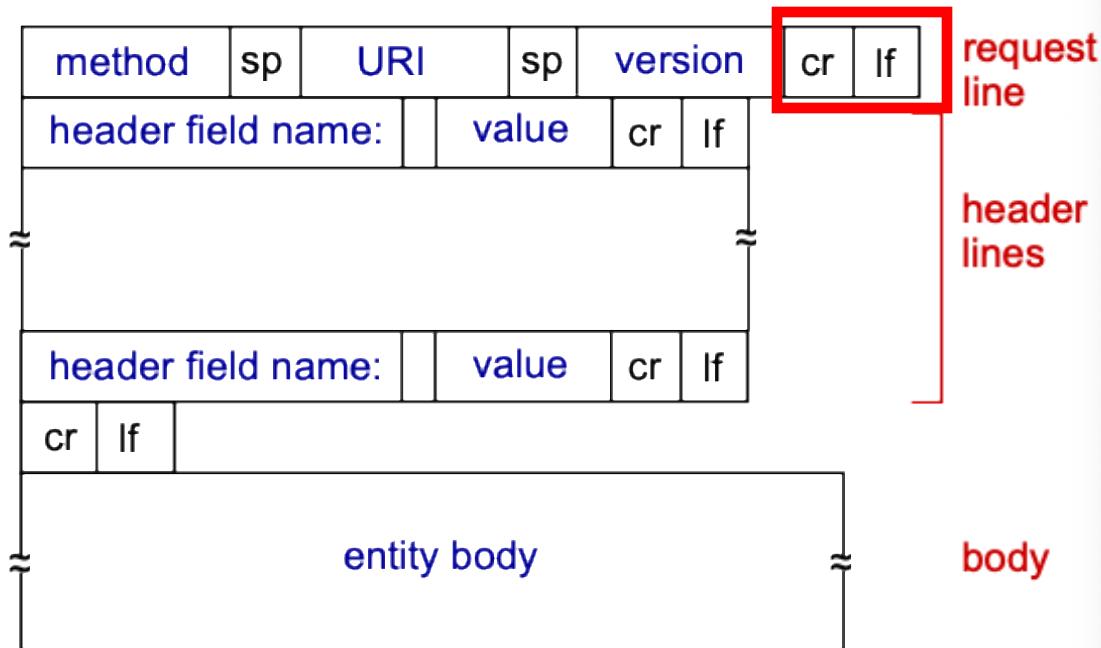
HTTP request message: general format



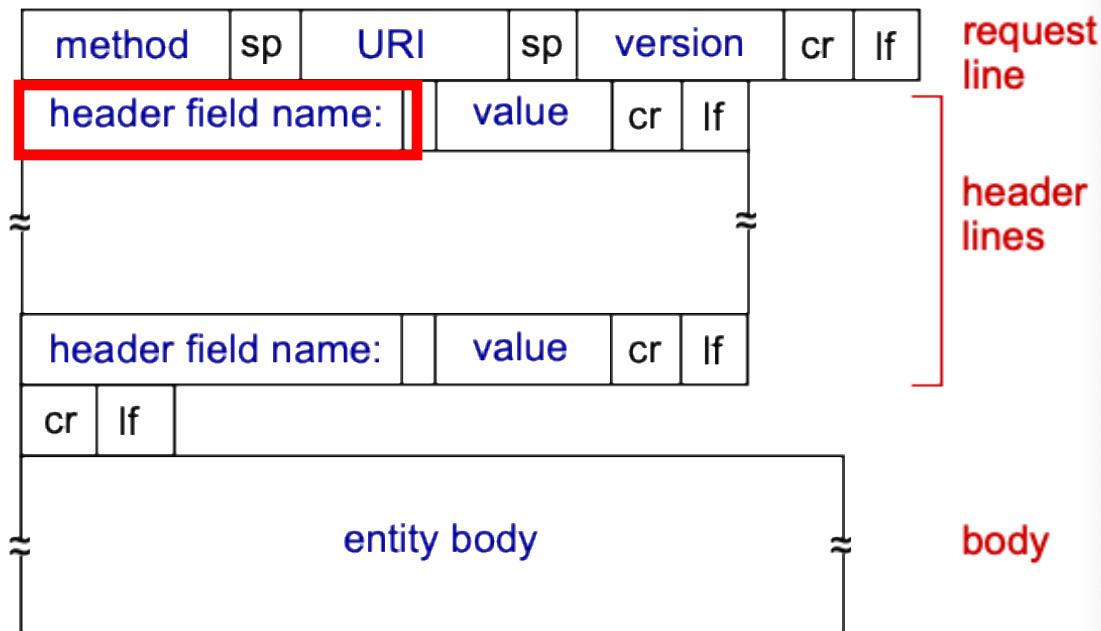
HTTP request message: general format



HTTP request message: general format



HTTP request message: general format



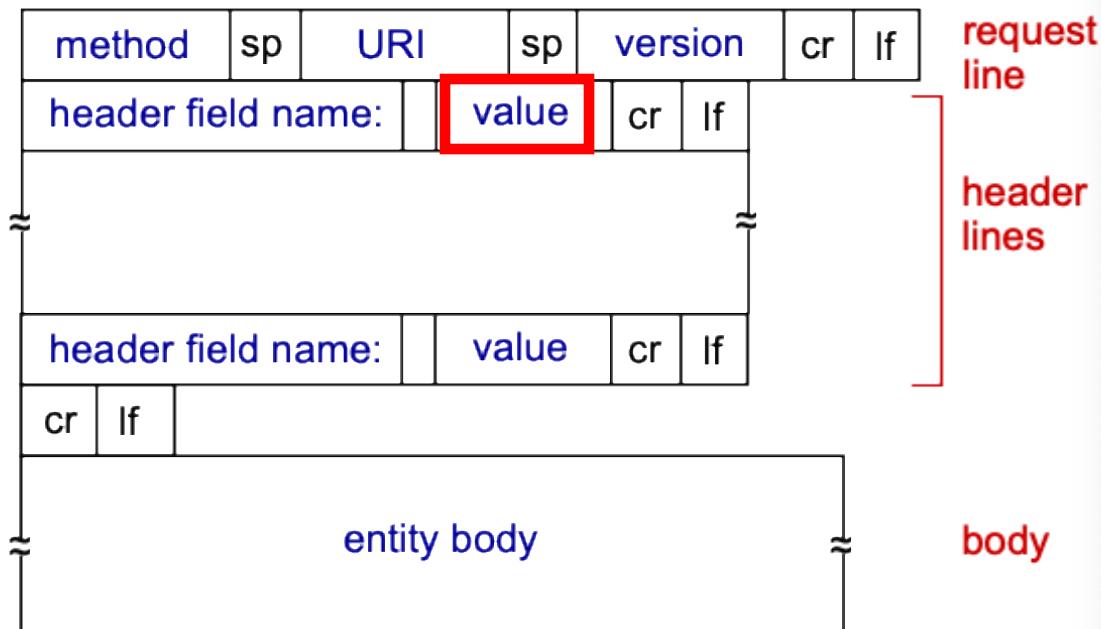
Wireshark · Packet 20 · Wi-Fi: en0 (ip host www.feimax.com)

Frame 20: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits) on interface en0 at 10:15:27.000000000 UTC
Ethernet II, Src: Apple_74:26:5d (f4:d4:88:74:26:5d), Dst: Tp-LinkT_0b:d8:5c (00:0c:29:0b:d8:5c)
Internet Protocol Version 4, Src: 192.168.0.24, Dst: 106.14.20.246
Transmission Control Protocol, Src Port: 59244, Dst Port: 80, Seq: 1, Ack: 1, Len: 489
Hypertext Transfer Protocol
GET / HTTP/1.1\r\nHost: www.feimax.com\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\nAccept-Encoding: gzip, deflate\r\nAccept-Language: zh\r\n\r\n[Full request URI: http://www.feimax.com/] [HTTP request 1/2] [Response in frame: 22] [Next request in frame: 24]

Show packet bytes

Help

HTTP request message: general format



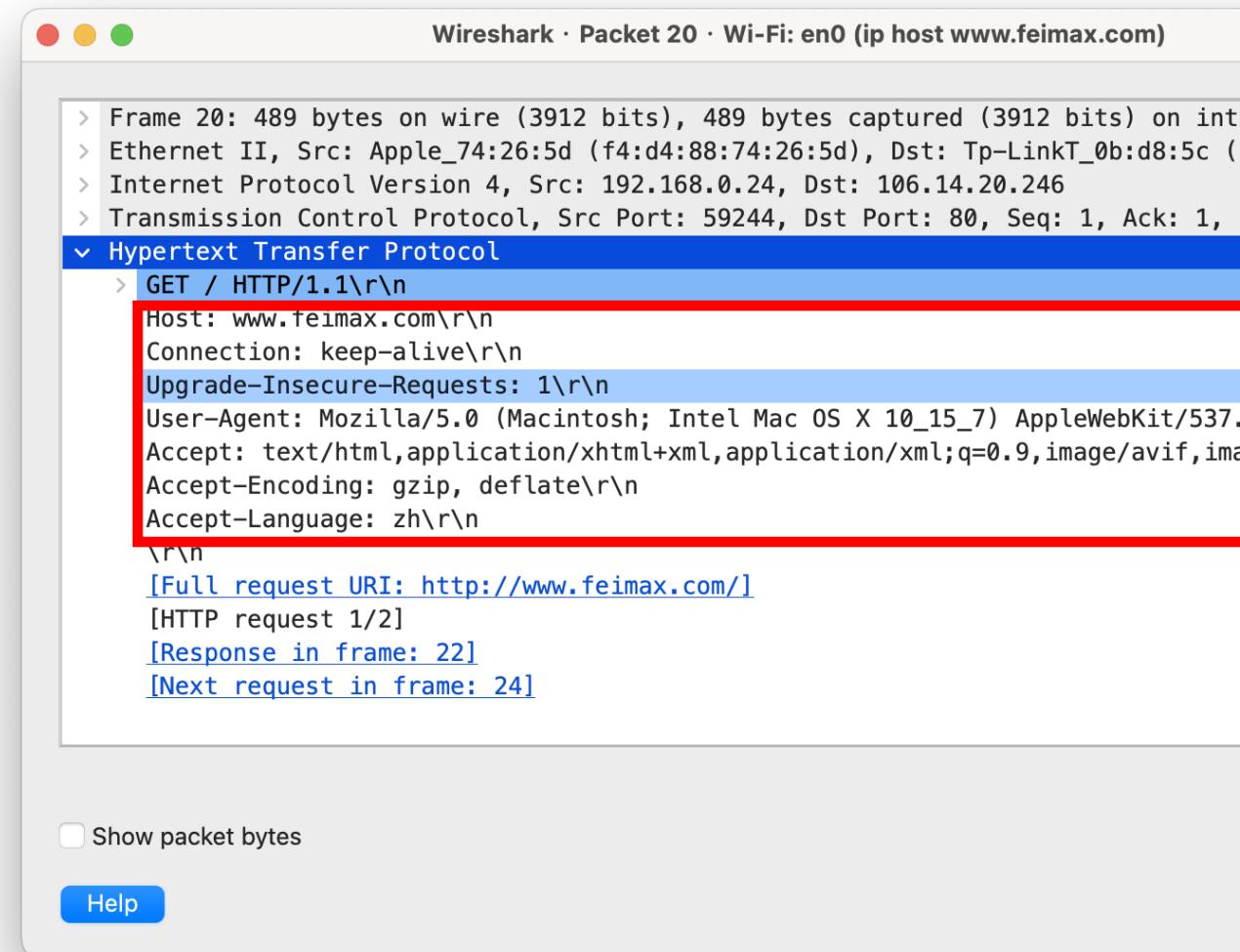
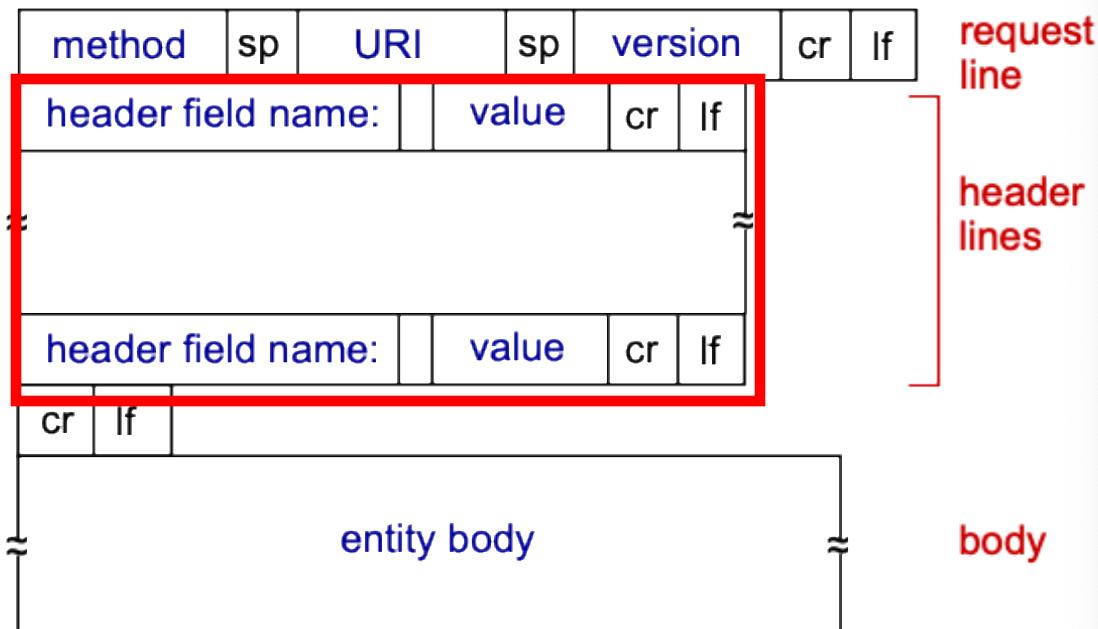
Wireshark · Packet 20 · Wi-Fi: en0 (ip host www.feimax.com)

```
> Frame 20: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits) on interface en0 at 10:00:00:00:00:00 [eth0]
> Ethernet II, Src: Apple_74:26:5d (f4:d4:88:74:26:5d), Dst: Tp-LinkT_0b:d8:5c (00:0c:29:0b:d8:5c)
> Internet Protocol Version 4, Src: 192.168.0.24, Dst: 106.14.20.246
> Transmission Control Protocol, Src Port: 59244, Dst Port: 80, Seq: 1, Ack: 1, Len: 489
└ Hypertext Transfer Protocol
  > GET / HTTP/1.1\r\n
    Host: www.feimax.com\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4285.147 Safari/537.36
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: zh\r\n
\r\n
[Full request URI: http://www.feimax.com/]
[HTTP request 1/2]
[Response in frame: 22]
[Next request in frame: 24]
```

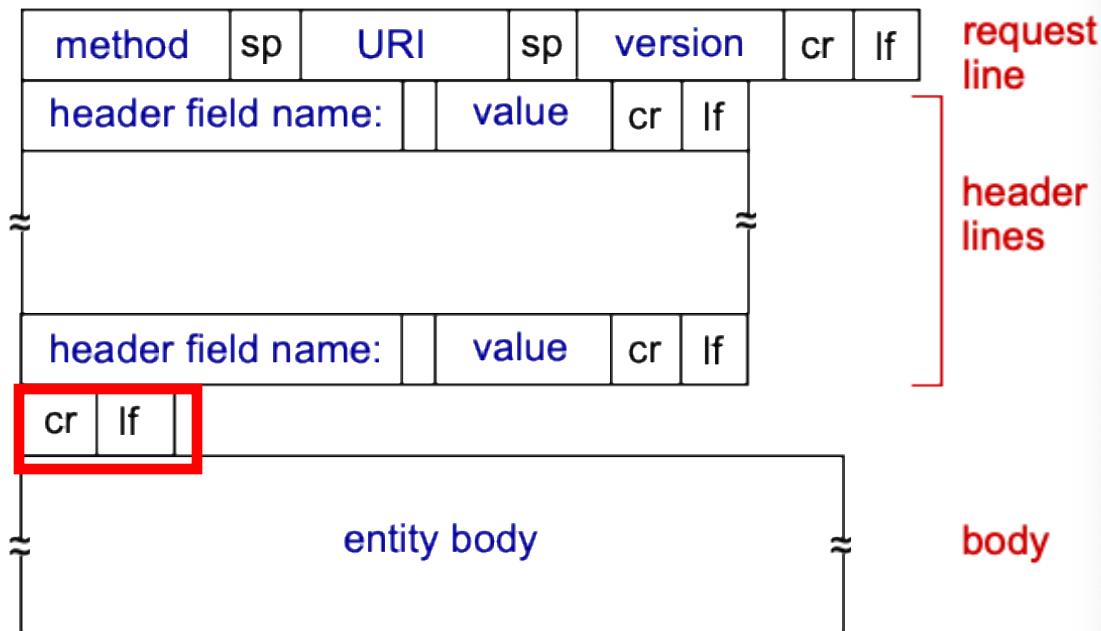
Show packet bytes

Help

HTTP request message: general format



HTTP request message: general format



Wireshark screenshot showing an captured HTTP request message:

- Frame 20: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits) on interface Wi-Fi: en0 (ip host www.feimax.com)
- Ethernet II, Src: Apple_74:26:5d (f4:d4:88:74:26:5d), Dst: Tp-LinkT_0b:d8:5c (00:0c:29:0b:d8:5c)
- Internet Protocol Version 4, Src: 192.168.0.24, Dst: 106.14.20.246
- Transmission Control Protocol, Src Port: 59244, Dst Port: 80, Seq: 1, Ack: 1, Hypertext Transfer Protocol
 - GET / HTTP/1.1\r\n
 - Host: www.feimax.com\r\n
 - Connection: keep-alive\r\n
 - Upgrade-Insecure-Requests: 1\r\n
 - User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36
 - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
 - Accept-Encoding: gzip, deflate\r\n
 - Accept-Language: zh\r\n

A red box highlights the \r\n sequence at the end of the header block. Below the packet list, there are links to the full request URI, response information, and the next request in the frame.

Wireshark · Packet 20 · Wi-Fi: en0 (ip host www.feimax.com)

> Frame 20: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits) on interface en0, id 0

> Ethernet II, Src: Apple_74:26:5d (f4:d4:88:74:26:5d), Dst: Tp-LinkT_0b:d8:5c (f4:2a:7d:0b:d8:5c)

> Internet Protocol Version 4, Src: 192.168.0.24, Dst: 106.14.20.246

> Transmission Control Protocol, Src Port: 59244, Dst Port: 80, Seq: 1, Ack: 1, Len: 423

✓ Hypertext Transfer Protocol

 GET / HTTP/1.1\r\n

 [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]

 Request Method: GET

 Request URI: /

 Request Version: HTTP/1.1

 Host: www.feimax.com\r\n

 Connection: keep-alive\r\n

 Upgrade-Insecure-Requests: 1\r\n

0040 db c2 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31	..GET / HTTP/1.1
0050 0d 0a 48 6f 73 74 3a 20 77 77 77 2e 66 65 69 6d	..Host: www.feim
0060 61 78 2e 63 6f 6d 0d 0a 43 6f 6e 6e 65 63 74 69	ax.com.. Connecti
0070 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a	on: keep -alive..
0080 55 70 67 72 61 64 65 2d 49 6e 73 65 63 75 72 65	Upgrade- Insecure
0090 2d 52 65 71 75 65 73 74 73 3a 20 31 0d 0a 55 73	-Request s: 1..Us
00a0 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c	er-Agent : Mozill
00b0 61 2f 35 2e 30 20 28 4d 61 63 69 6e 74 6f 73 68	a/5.0 (M acintosh
00c0 3b 20 49 6e 74 65 6c 20 4d 61 63 20 4f 53 20 58	; Intel Mac OS X
00d0 20 31 30 5f 31 35 5f 37 29 20 41 70 70 6c 65 57	10_15_7) AppleW
00e0 65 62 4b 69 74 2f 35 33 37 2e 33 36 20 28 4b 48	ebKit/53 7.36 (KH
00f0 54 4d 4c 2c 20 6c 69 6b 65 20 47 65 63 6b 6f 29	TML, lik e Gecko)
0100 20 43 68 72 6f 6d 65 2f 31 32 38 2e 30 2e 30 2e	Chrome/ 128.0.0.
0110 30 20 53 61 66 61 72 69 2f 35 33 37 2e 33 36 0d	Safari /537.36.

Bytes 66-68: Request Method (http.request.method)

Show packet bytes

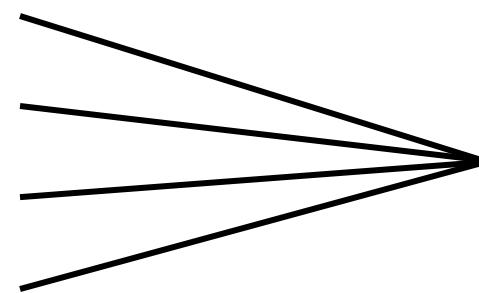
Help Close

Human Readable!

HTTP 1.1 request method

- What do we need from a web protocol?

- Get
- Create
- Modify
- Delete



resource / data / information

CRUD

HTTP 1.0 1.1 Additional methods

- GET
- POST
- PUT
- DELETE
- PATCH
- HEAD
- TRACE
- OPTIONS
- CONNECT

HTTP 1.1 request headers

- Key-value pairs, metadata about the request and the client:
 - Host name
 - Authentication
 - Content types
 - User-agent information
 - Caching / Cookies
 - Types of connections
 - ...

Types of connections

Non-persistent HTTP

- At most one object sent over TCP connection
 - connection then closed
- Downloading multiple objects required multiple connections

Persistent HTTP

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

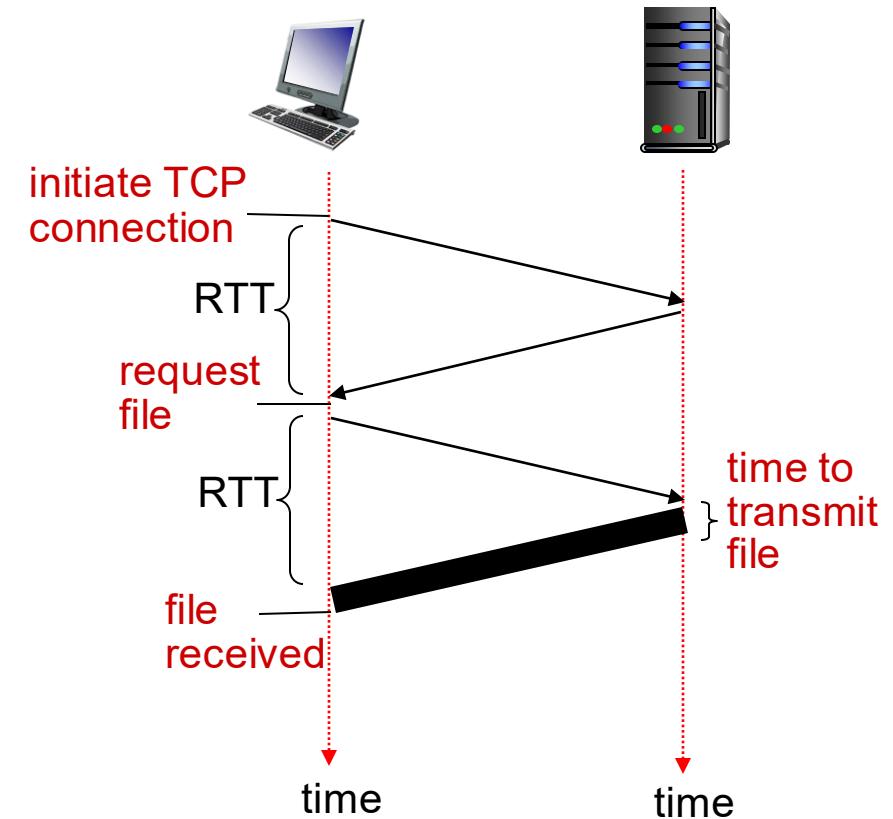
Non-persistent HTTP: response time

RTT (Round Trip Time): time for a small packet to travel from client to server and back round trip time

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 =

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



Persistent HTTP

Non-persistent HTTP issues:

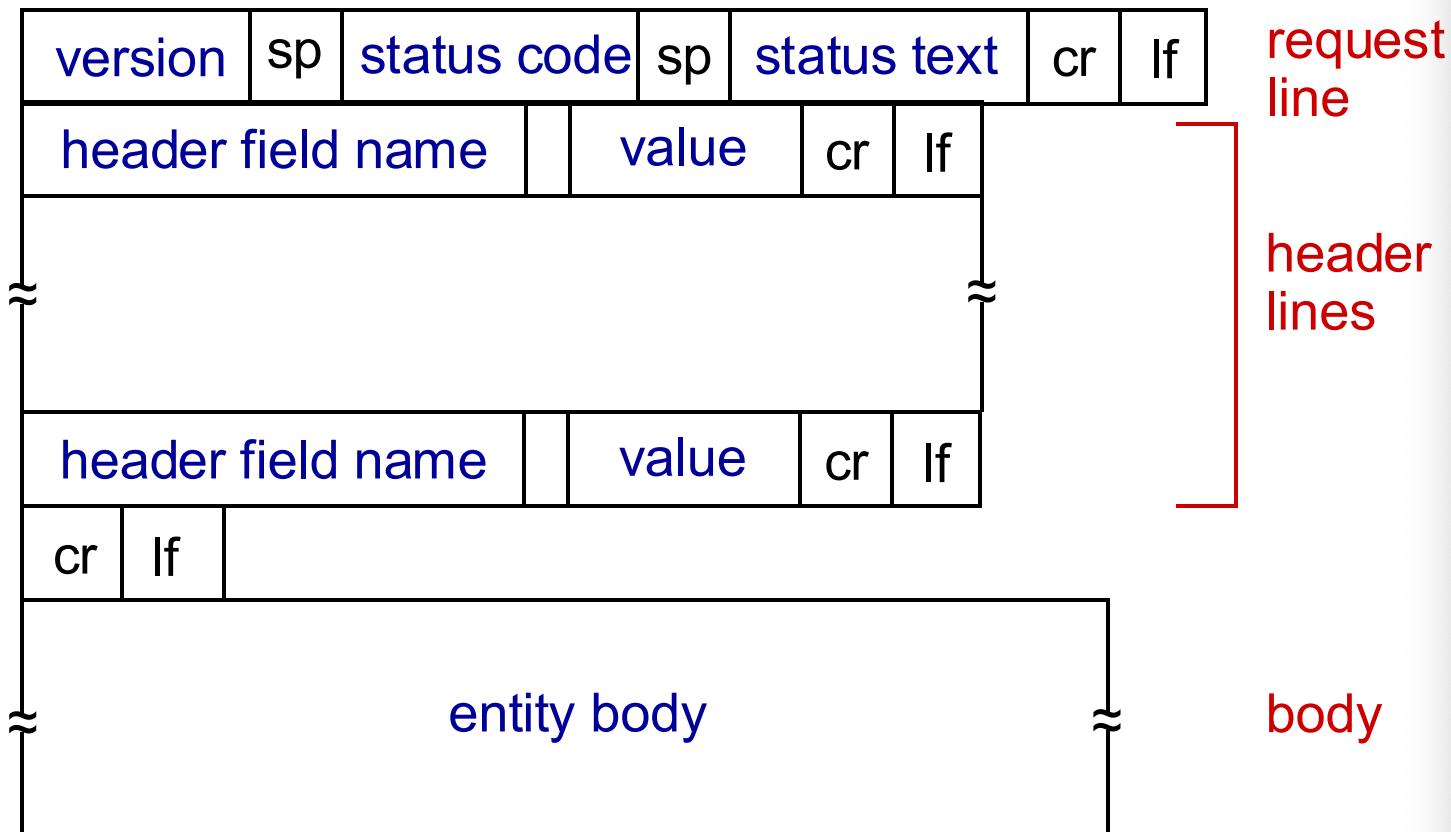
- Requires 2 RTTs per object
- OS overhead for *each* TCP connection
- Browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP issues:

- 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

Q: Is persistent HTTP perfect? <=

HTTP response message: format



Wireshark · Packet 22 · Wi-Fi: en0 (ip host www.feimax.com)

```
> Frame 22: 784 bytes on wire (6272 bits), 784 bytes captured (6272 bits)
> Ethernet II, Src: TP-LinkT_0b:d8:5c (f4:2a:7d:0b:d8:5c), Dst: Apple_
> Internet Protocol Version 4, Src: 106.14.20.246, Dst: 192.168.0.24
> Transmission Control Protocol, Src Port: 80, Dst Port: 59244, Seq: 1, Ack: 1, Len: 784
< HTTP/1.1 200 OK\r\n
< Server: nginx\r\n
< Date: Tue, 17 Sep 2024 12:51:15 GMT\r\n
< Content-Type: text/html\r\n
< Transfer-Encoding: chunked\r\n
< Connection: keep-alive\r\n
< Vary: Accept-Encoding\r\n
< Content-Encoding: gzip\r\n
< \r\n
[HTTP response 1/2]
[Time since request: 0.022098000 seconds]
[Request in frame: 20]
[Next request in frame: 24]
[Next response in frame: 2888]
[Request URI: http://www.feimax.com/]
> HTTP chunked response
Content-encoded entity body (gzip): 511 bytes -> 825 bytes
File Data: 825 bytes
< Line-based text data: text/html (22 lines)
< !DOCTYPE html>\r\n
```

Show packet bytes

Help

HTTP response status codes

- Status code appears in 1st line in server-to-client response message.
- Some sample codes:
 - 200 OK
 - 301 Move Permanently
 - 400 Bad Request
 - 404 Not Found
 - 505 HTTP Version Not Supported

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

Status-Code =
"100" ; Section [10.1.1](#): Continue
"101" ; Section [10.1.2](#): Switching Protocols
"200" ; Section [10.2.1](#): OK
"201" ; Section [10.2.2](#): Created
"202" ; Section [10.2.3](#): Accepted
"203" ; Section [10.2.4](#): Non-Authoritative Information
"204" ; Section [10.2.5](#): No Content
"205" ; Section [10.2.6](#): Reset Content
"206" ; Section [10.2.7](#): Partial Content
"300" ; Section [10.3.1](#): Multiple Choices
"301" ; Section [10.3.2](#): Moved Permanently
"302" ; Section [10.3.3](#): Found
"303" ; Section [10.3.4](#): See Other
"304" ; Section [10.3.5](#): Not Modified
"305" ; Section [10.3.6](#): Use Proxy
"307" ; Section [10.3.8](#): Temporary Redirect
"400" ; Section [10.4.1](#): Bad Request
"401" ; Section [10.4.2](#): Unauthorized
"402" ; Section [10.4.3](#): Payment Required
"403" ; Section [10.4.4](#): Forbidden
"404" ; Section [10.4.5](#): Not Found
"405" ; Section [10.4.6](#): Method Not Allowed
"406" ; Section [10.4.7](#): Not Acceptable
"407" ; Section 10.4.8: Proxy Authentication Required
"408" ; Section [10.4.9](#): Request Time-out
"409" ; Section [10.4.10](#): Conflict
"410" ; Section [10.4.11](#): Gone
"411" ; Section [10.4.12](#): Length Required
"412" ; Section [10.4.13](#): Precondition Failed
"413" ; Section [10.4.14](#): Request Entity Too Large
"414" ; Section [10.4.15](#): Request-URI Too Large
"415" ; Section [10.4.16](#): Unsupported Media Type
"416" ; Section [10.4.17](#): Requested range not satisfiable
"417" ; Section [10.4.18](#): Expectation Failed
"500" ; Section [10.5.1](#): Internal Server Error
"501" ; Section [10.5.2](#): Not Implemented
"502" ; Section [10.5.3](#): Bad Gateway
"503" ; Section [10.5.4](#): Service Unavailable
"504" ; Section [10.5.5](#): Gateway Time-out
"505" ; Section [10.5.6](#): HTTP Version not supported
extension-code

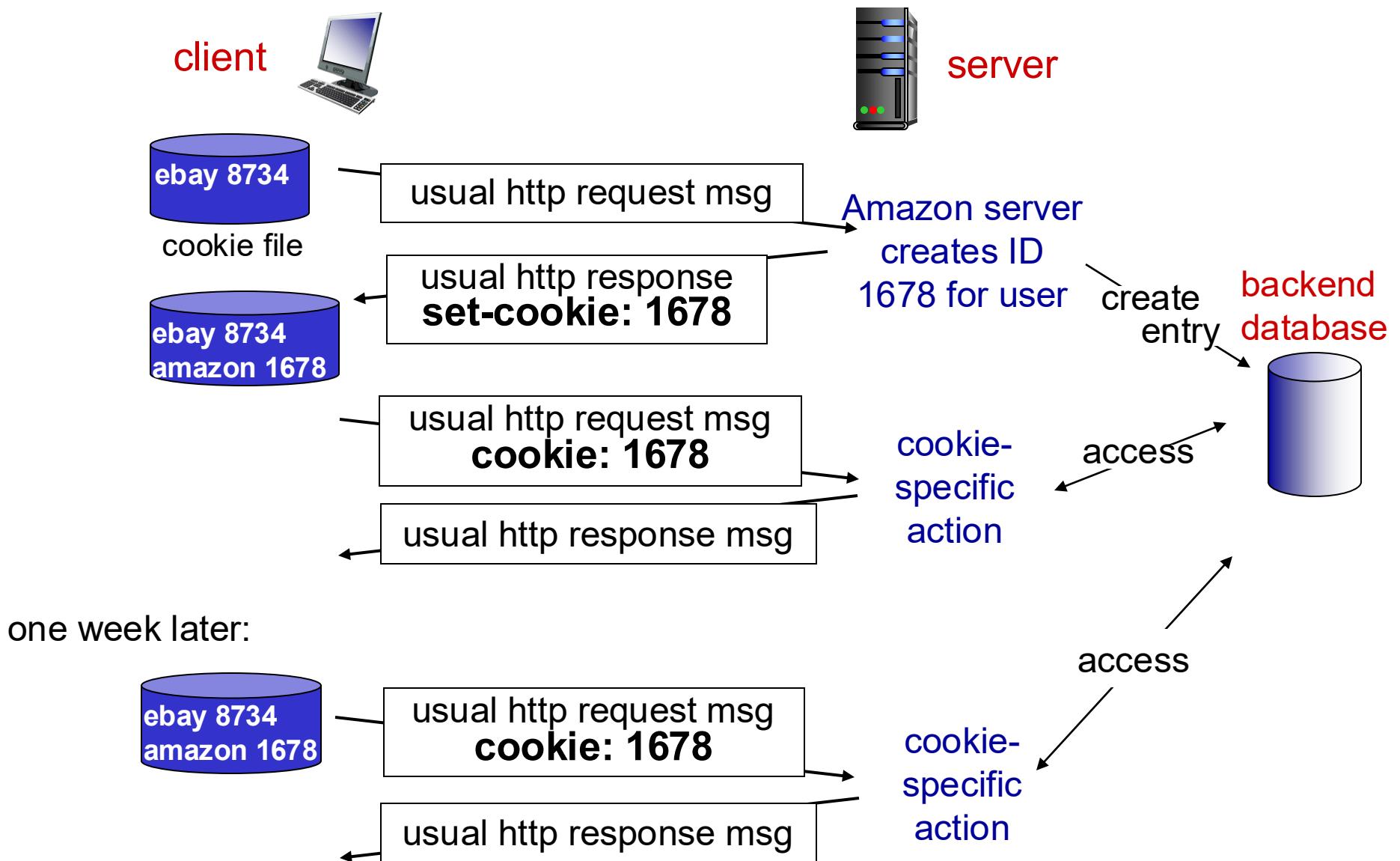
HTTP is “stateless” !
So...

User-server state: cookies

- Many Web sites use cookies
- Four components:
 1. cookie header line of HTTP **response** message
 2. cookie header line in next HTTP **request** message
 3. cookie file kept on user's host, managed by user's browser
 4. back-end database at Web site

Example:

- Bob always access Internet from PC
- Visits Taobao site for first time
- When initial HTTP requests arrives at site, site creates:
 - unique ID -> cookie
 - entry in backend database for ID



Cookies

What cookies can be used for:

- Authorization
- Recommendations
- User session state
- ...

aside

Cookies and privacy:

- Cookies permit sites to learn a lot about you
- Remember to clean your cookies

Wireshark Test

▼ Hypertext Transfer Protocol

▶ GET / HTTP/1.1\r\n

Host: www.feimax.com\r\n

Connection: keep-alive\r\n

Upgrade-Insecure-Requests: 1\r\n

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8\r\n

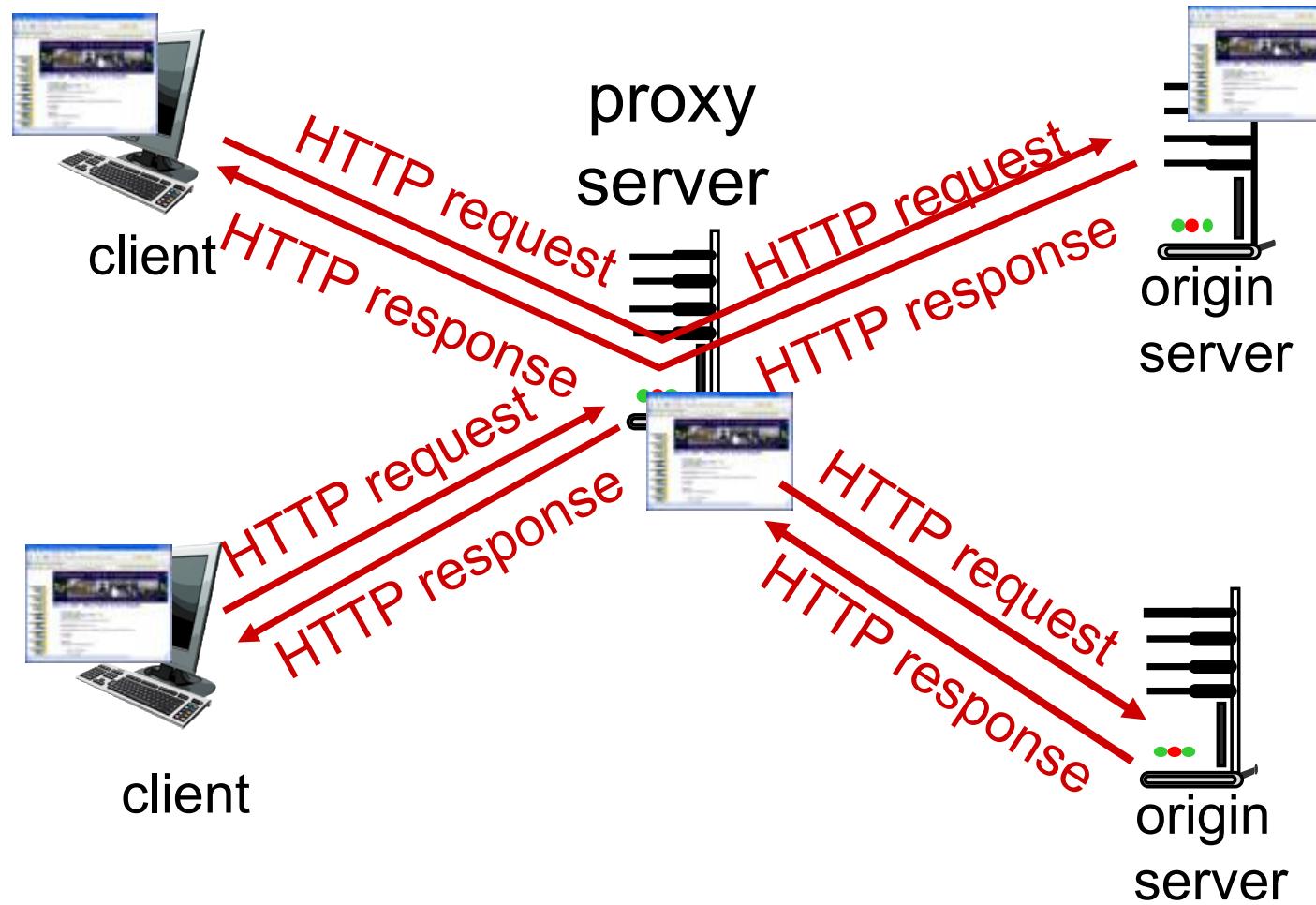
Purpose: prefetch\r\n

Accept-Encoding: gzip, deflate\r\n

Accept-Language: zh,en;q=0.9,zh-TW;q=0.8\r\n

▶ Cookie: wp-settings-time-1=1567750800; wp-settings-1=libraryContent%3Dbrowse%26in\r\n\r\n

Web caches



More about Web caching

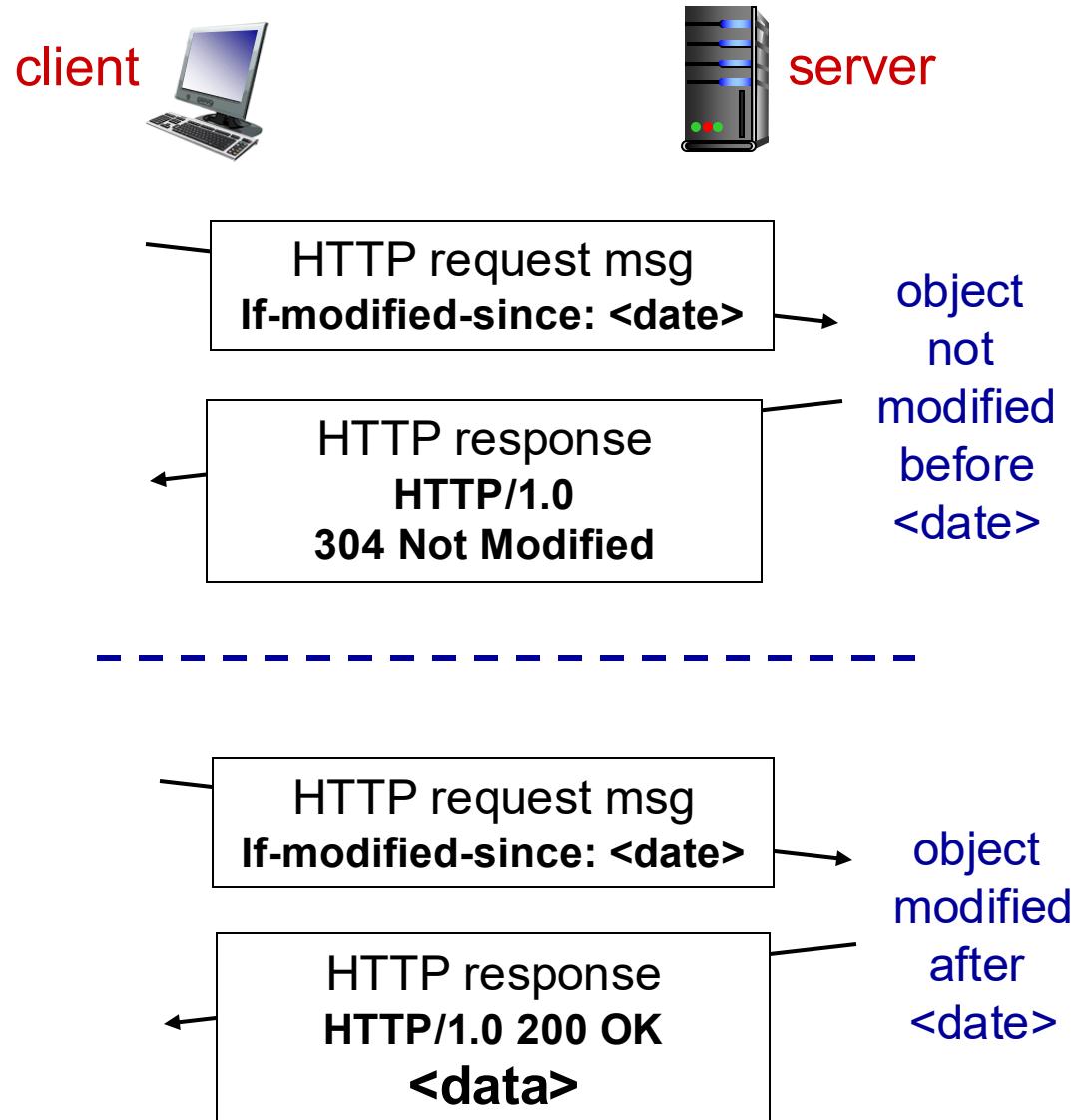
- Cache acts as both client and server
 - server for original requesting client
 - client to origin server
- Typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

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



HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (**head-of-line (HOL) blocking**) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

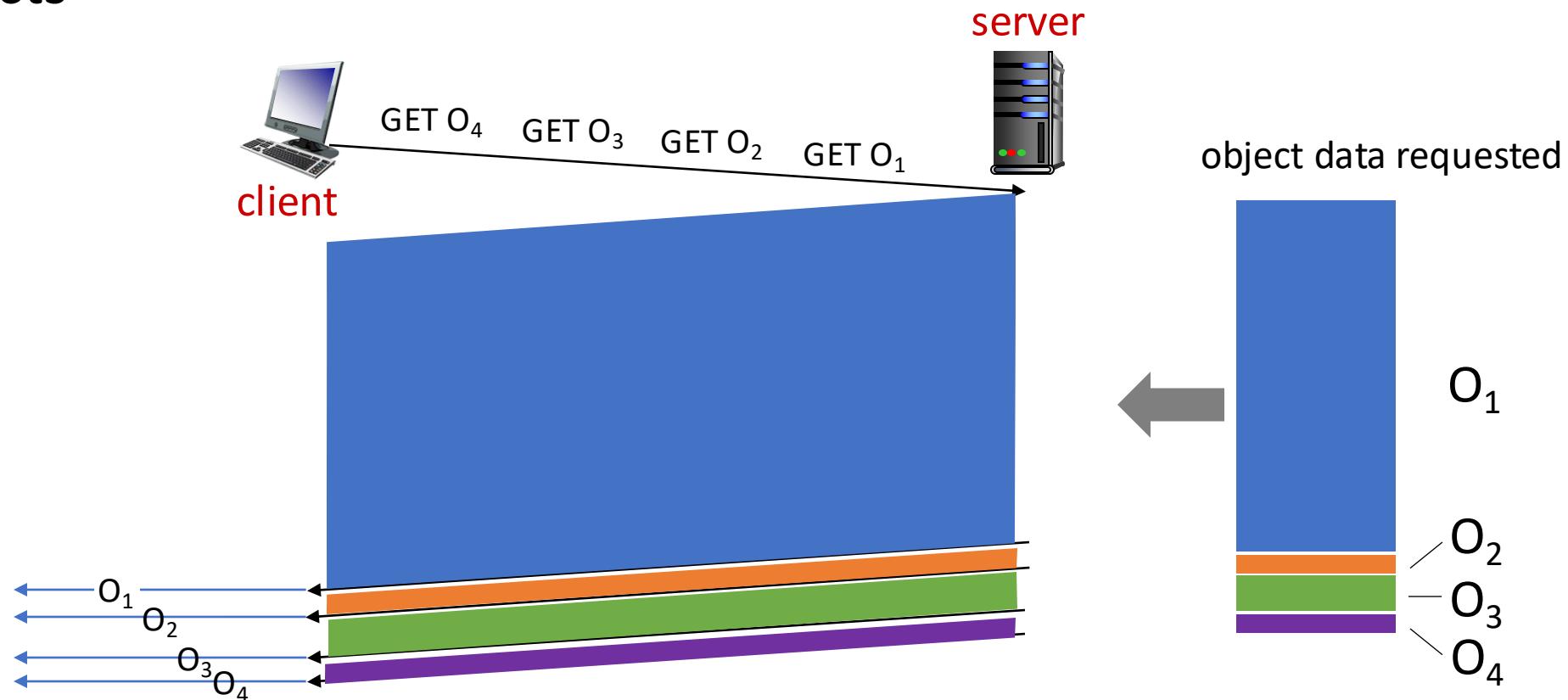
Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

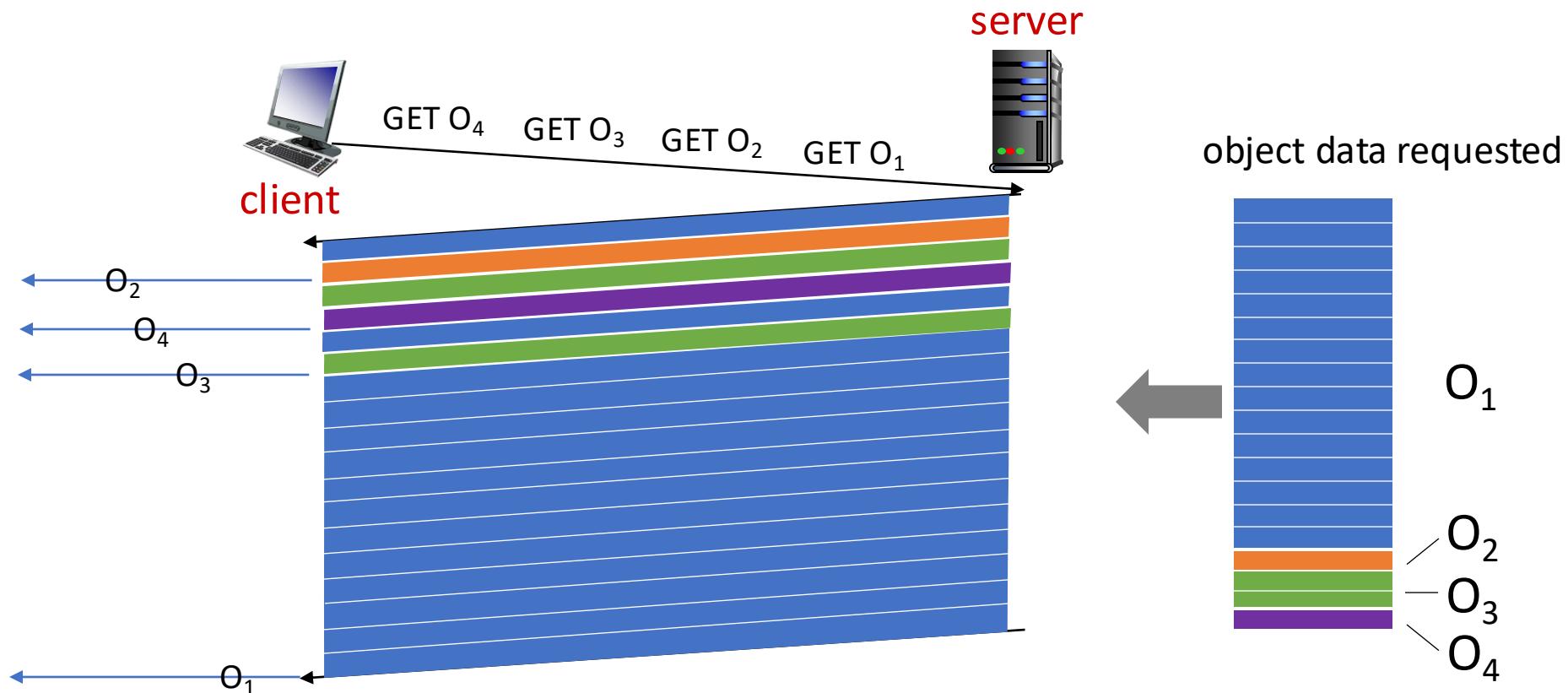
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2 , O_3 , O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



O₂, O₃, O₄ delivered quickly, O₁ slightly delayed

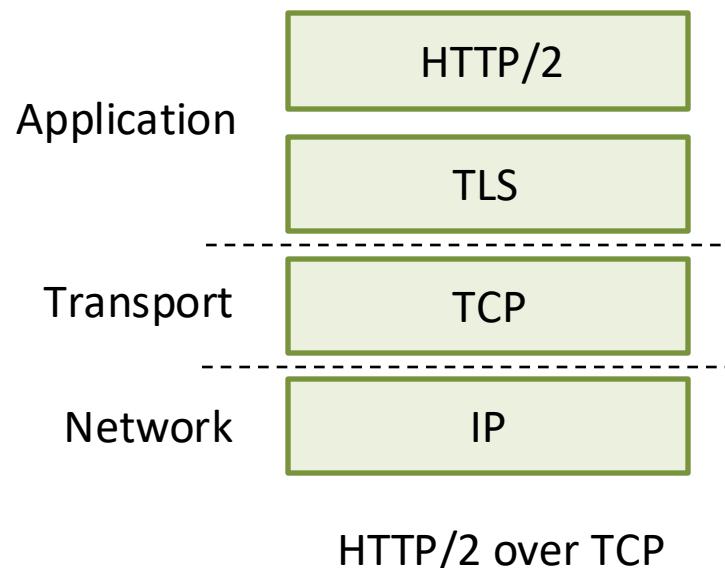
HTTP/2 to HTTP/3

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmissions
 - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- no security over raw TCP connection
- **HTTP/3:** adds security, per object error- and congestion-control (more pipelining) over UDP

QUIC: Quick UDP Internet Connections

- application-layer protocol, on top of UDP
 - increase performance of HTTP
 - deployed on many Google servers, apps (Chrome, mobile YouTube app)

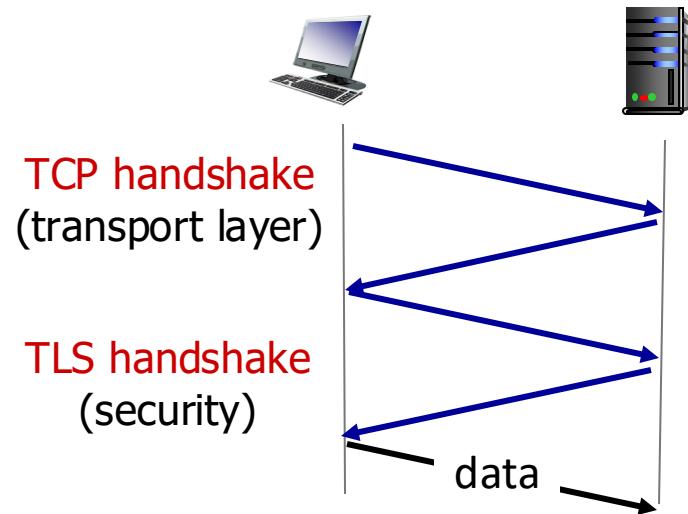


QUIC: Quick UDP Internet Connections

adopts approaches we'll study in chapter for connection establishment, error control, congestion control

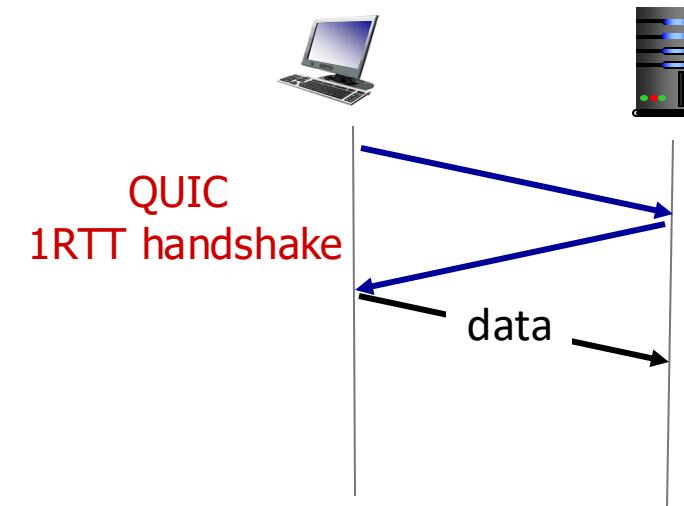
- **error and congestion control:** “Readers familiar with TCP’s loss detection and congestion control will find algorithms here that parallel well-known TCP ones.” [from QUIC specification]
- **connection establishment:** reliability, congestion control, authentication, encryption, state established in one RTT

QUIC: Connection establishment



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

- 2 serial handshakes



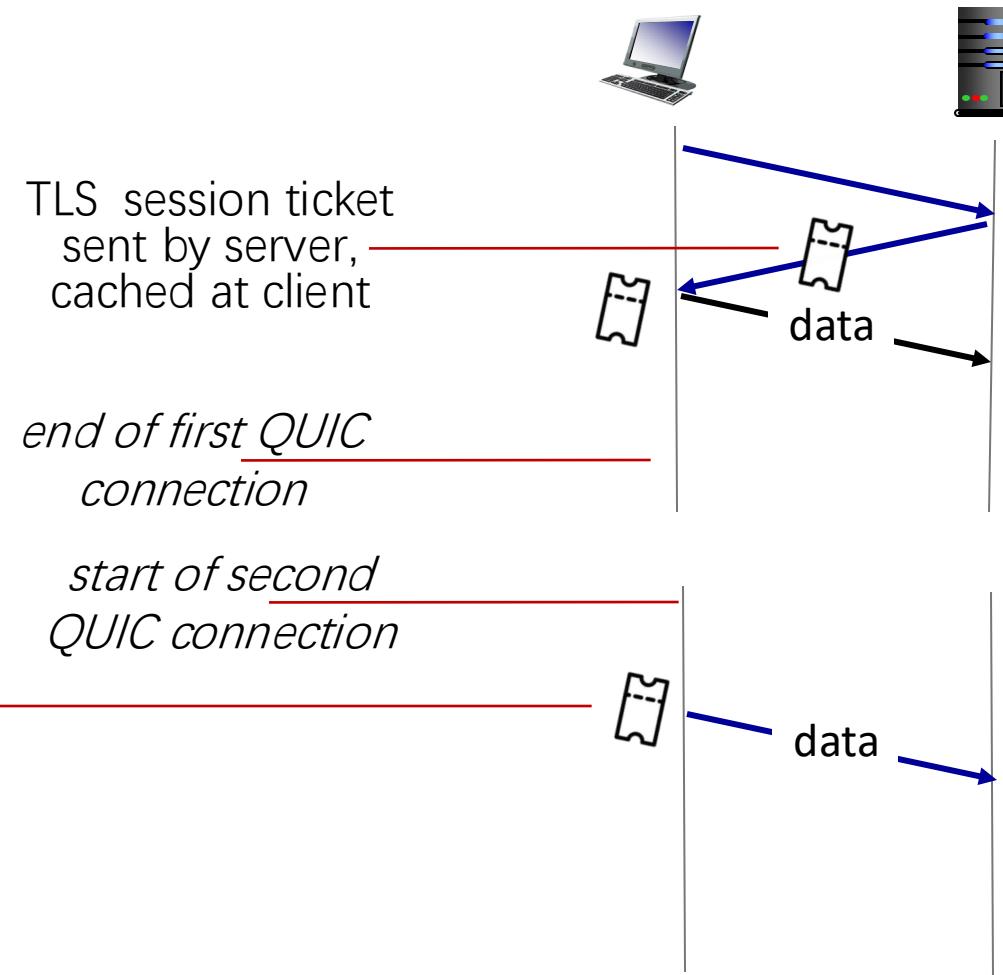
QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

QUIC: 0-RTT Connection establishment

Client uses last session ticket for encryption, authentication for new connection

- 0 RTT handshake delay



Feature	HTTP/1.1	HTTP/2	HTTP/3
Transport layer	TCP	TCP	QUIC (UDP-based)
Multiplexing	✗ (head-of-line blocking)	✓ (but blocked by TCP loss)	✓ (independent streams)
Header compression	✗	✓ (HPACK)	✓ (QPACK)
Server push	✗	✓	✓ (but declining usage)
Encryption	Optional (TLS)	Practically mandatory (TLS)	Mandatory (TLS 1.3)
Handshake latency	High (TCP + TLS)	Lower	Lowest (0-RTT)
Connection migration	✗	✗	✓ (connection ID)

Thanks.