Chapter 2: Application Layer

Part 1: Application Layer-HTTP

Instructor: HOU, Fen

2025

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network applications
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - SMTP / POP3 / IMAP
  - DNS

Some Network Applications

- E-mail
- Remote login
- File transfer

- Web
- Instant messaging
- P2P file sharing
- Internet telephone
- Real-time video conference

- Multi-user network games
- Streaming stored video clips
- Massive parallel computing

Creating a Network Application

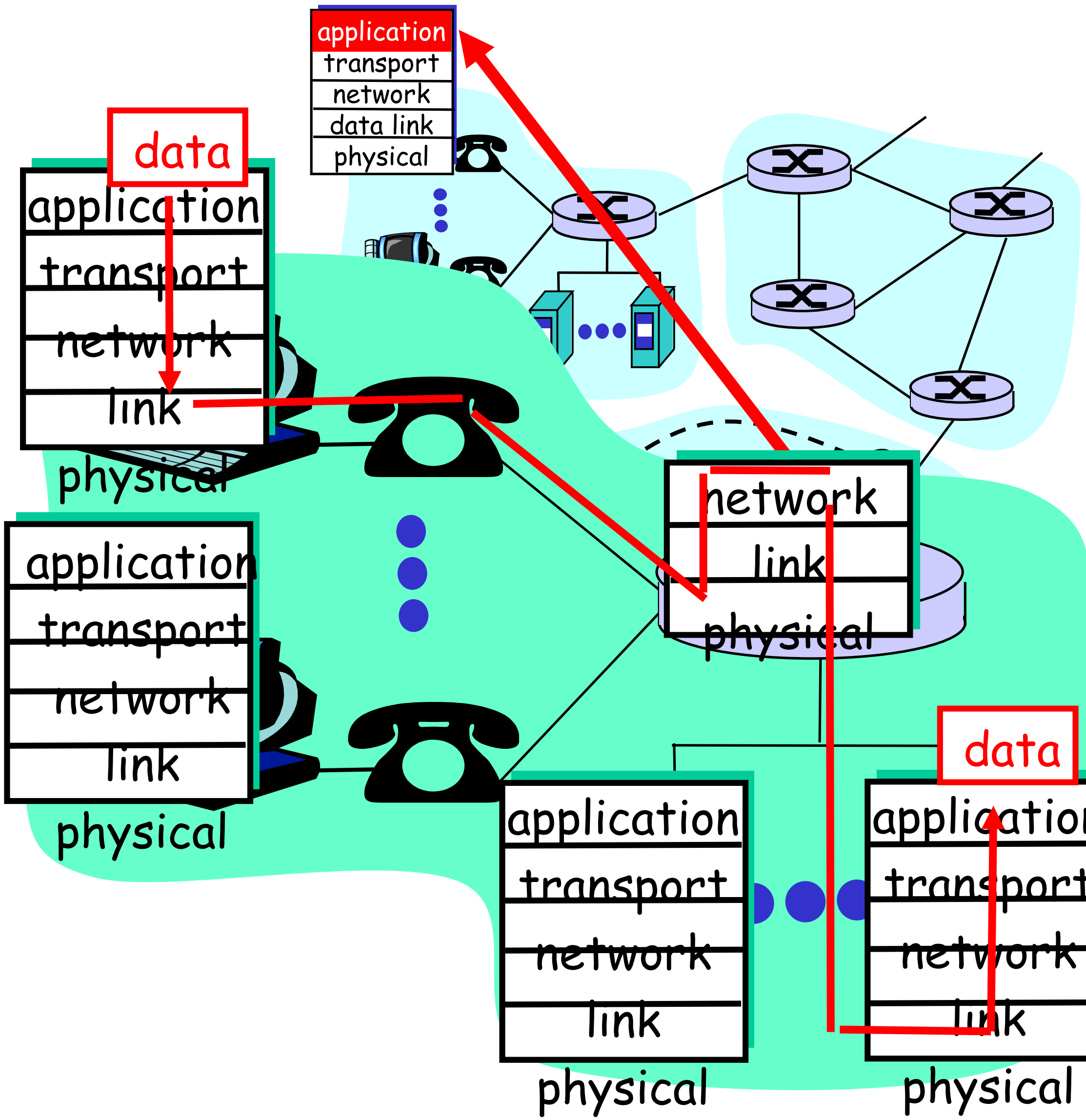
Write programs that

- run on different **end systems** and
- communicate over a network.
- e.g., Web: Web server software communicates with browser software

No application software written for devices in network core

- Network core devices do not function at app layer. They function at lower layers (network layer and below)

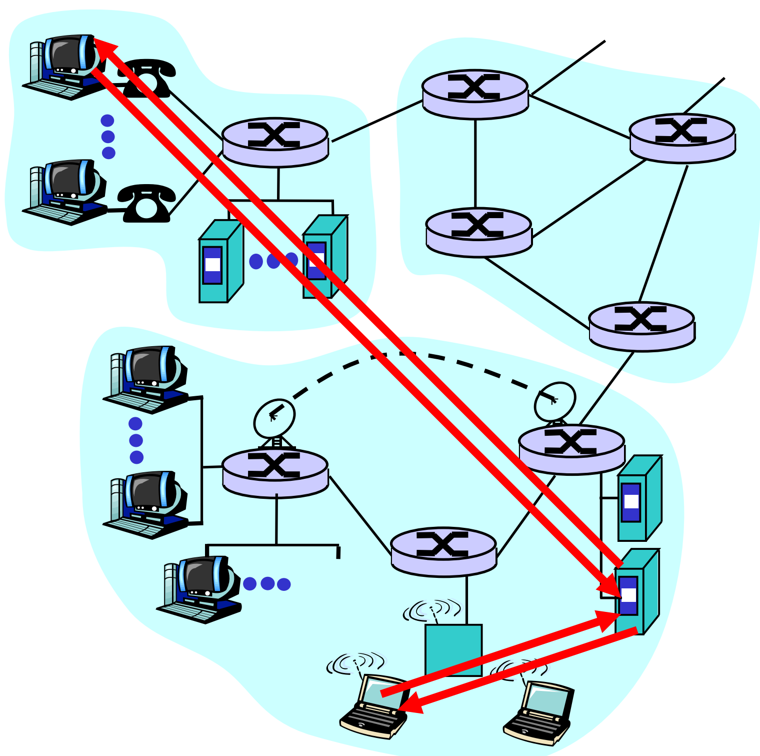
This design allows for rapid app development



Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server Model/Architecture



Server:

- always-on host
- Fixed, well-known and permanent IP address
- server farm for scaling

Clients:

- Either sometimes-on or always-on. That is, it may be intermittently connected
- do not communicate directly with each other
- communicate with server
- may have dynamic IP addresses

Pros and cons:

- Infrastructure intensive
- Easy to manage and secure
- Costly to provide
- Single point of failure problem

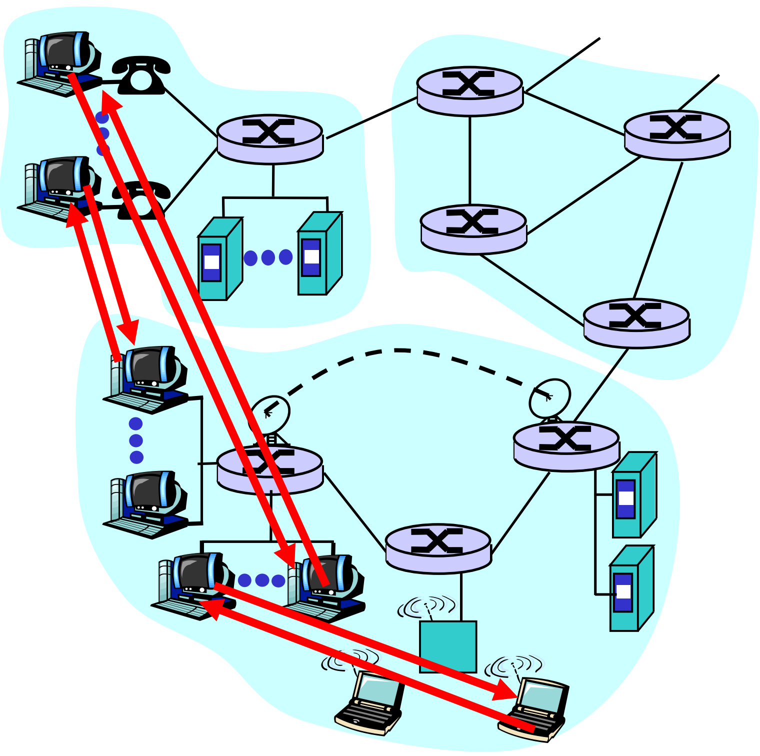
Pure P2P Model/Architecture

Features:

- Each host run programs that performs both client and server function
- No fixed clients or servers
- peers are intermittently connected. No always on server.
- arbitrary end systems directly communicate

Pros and cons:

- Highly scalable
- Difficult to manage
- Challenge to secure such as privacy risk, online attacks, etc.



Hybrid of Client-server and P2P

Features

- Use central servers to do registration
- Use P2P for the data transfers

Instant messaging

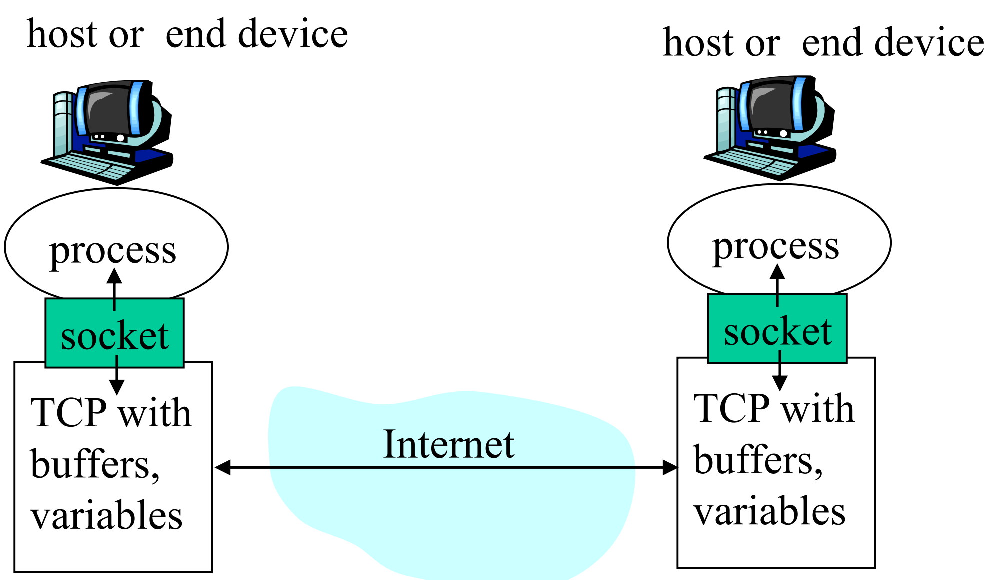
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies
- Chatting between two users is P2P

How are the Messages/data transmitted between end systems? ----Processes communicating

Host : an end system/device/computer running application programs.

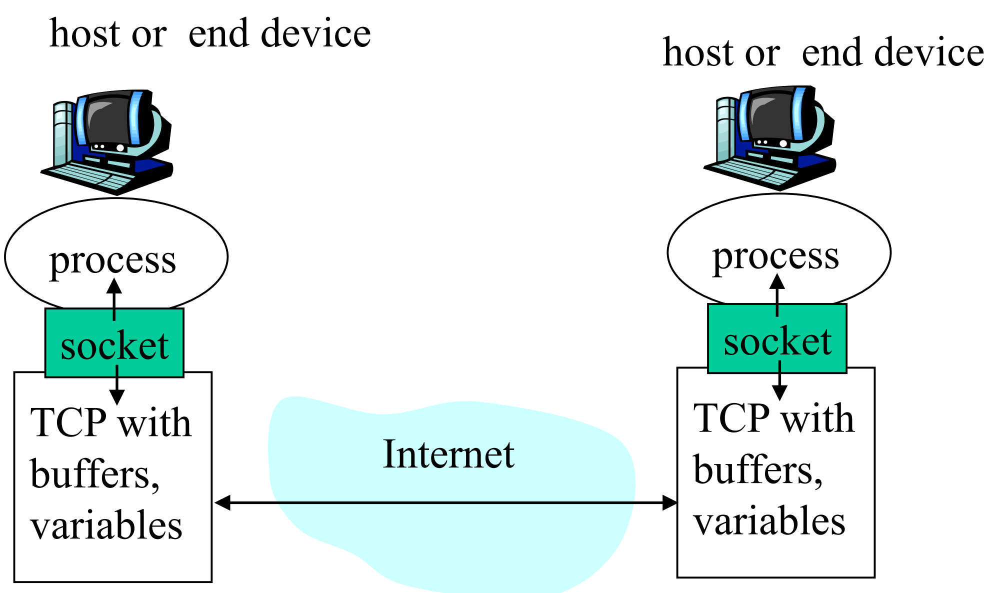
Process : program running within a host.

- processes in different hosts communicate by exchanging messages
- Client process: process that initiates communication
- Server process: process that waits to be contacted



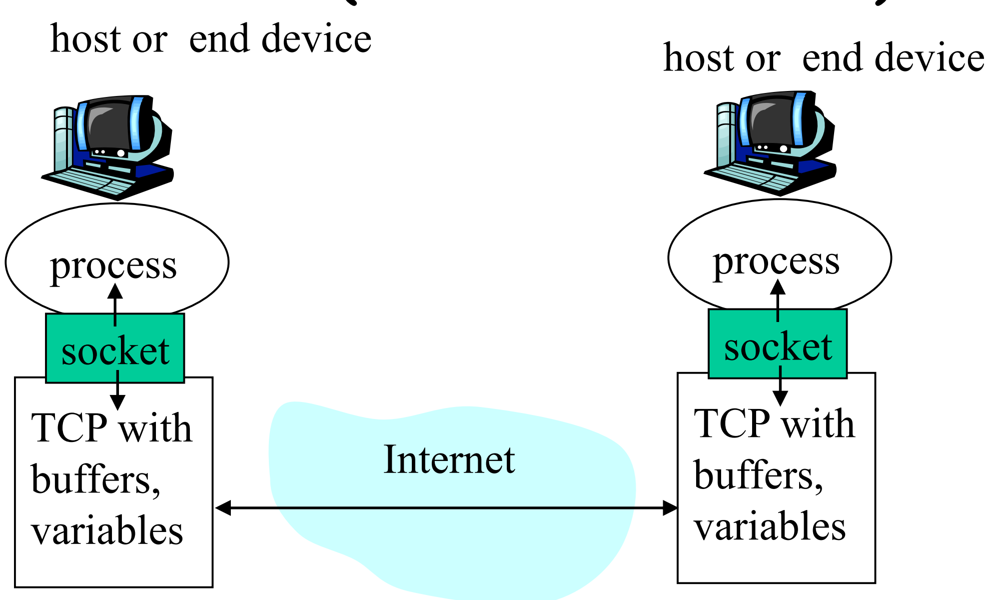
Process Identifier

- For a process to receive messages, it must have an identifier.
- Every host has a unique 32-bit IP address (For IPV4)
  - Does the IP address of a host is sufficient for identifying a process?
  - No, many processes can be running on the same host.



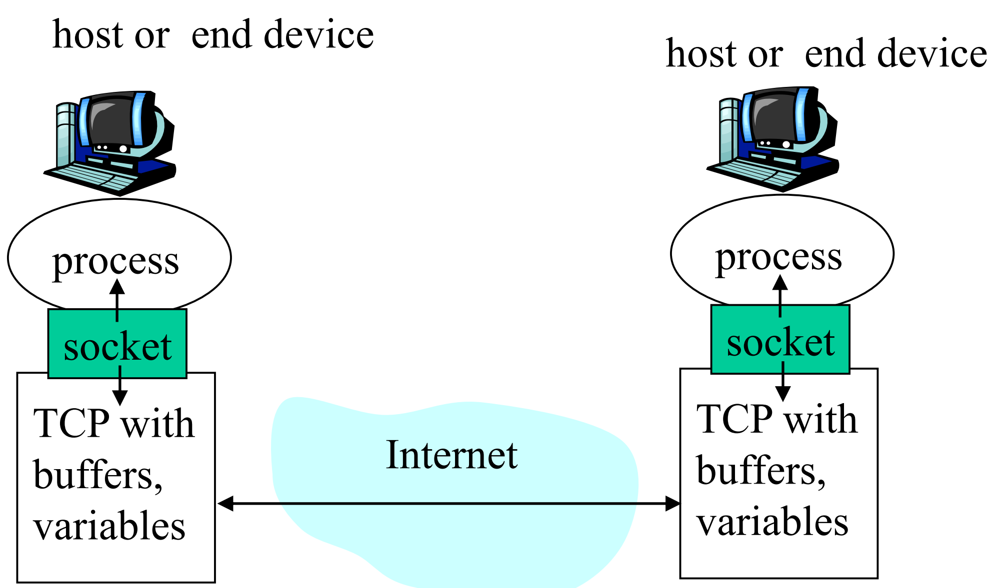
Process Identifier

- Identifier includes both the IP address and port number associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
  - DNS server: 53
- Example Identifier: (80: 113.45.12.201)



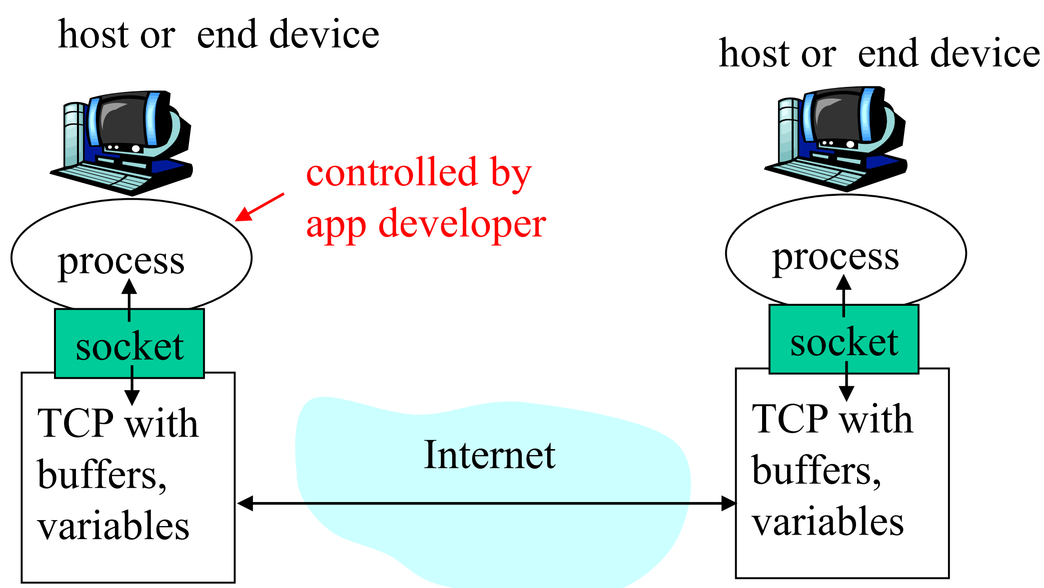
Socket

- With the process identifier, server process can be identified and connected by the client.
- How can a server (i.e., Web server) identify the requests from different clients and connect the communications with them?
- Socket is to achieve this function.



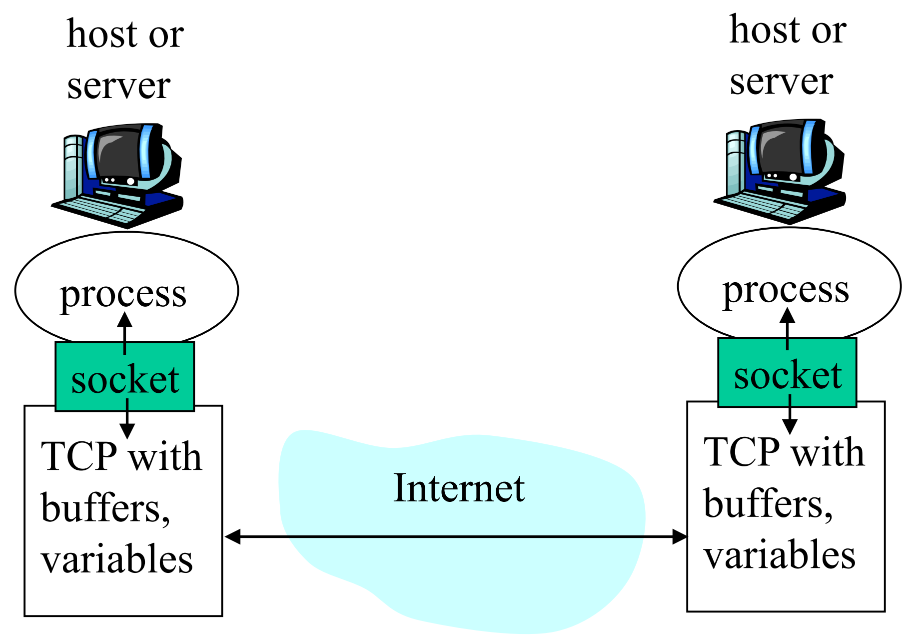
Socket: it is a software interface through which a process sends data into, and receives data from, the network.

- Socket is a programming interface between the application layer and the transport layer. Therefore, socket is called an Application Programming Interface (API).
- It has two sides: on the application-layer side of the socket, the application developer has control of everything; on the transport-layer side, the application developer can have control on (1) the choice of transport protocol and (2)fix a few transport-layer parameters such as maximum buffer size and maximum segment sizes.



A Analogous Example

- Host analogous to house
  - IP address versus mail address
- process analogous room
- Socket analogous to door
  - Process sends/receives messages to/from its socket
  - sending process pushes message out of door
  - sending process relies on transport layer protocol to transmit data to the receiving host and go to the receiving process through its socket.
  - Socket/API can control (1) choice of transport protocol; (2) ability to set a few parameters



What transport service does an app need?

Data loss (reliable data transfer)

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

Timing

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

Security

- some apps (e.g., e-commerce) need high secure service.

Throughput

- some apps ("bandwidth-sensitive apps", e.g., multimedia) require minimum throughput (i.e., guaranteed throughput service).
- other apps ("elastic apps", e.g., e-mail, web transfer) make use of whatever bandwidth they get (i.e., best-effort service)

Transport service requirements of common apps

Application	Data loss	Bandwidth	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 video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP (Transmission Control Protocol) service:

- *connection-oriented:* setup required between client and server processes
- *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 bandwidth guarantees

UDP (User Datagram Protocol) service:

- Connectionless: don't need to set up connection before communication.
- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

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 multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	Proprietary	typically UDP

Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS

Web and HTTP (HyperText Transfer Protocol超文本传输协议)

- At the early state, file transfer, remote access, e-mail for researchers, academics and students.
- In the early 1990s, World Wide Web
  - Web operates on demand: users receive what they want when they want it.
  - Easy implementation and low cost: users can easily post the information on the web.

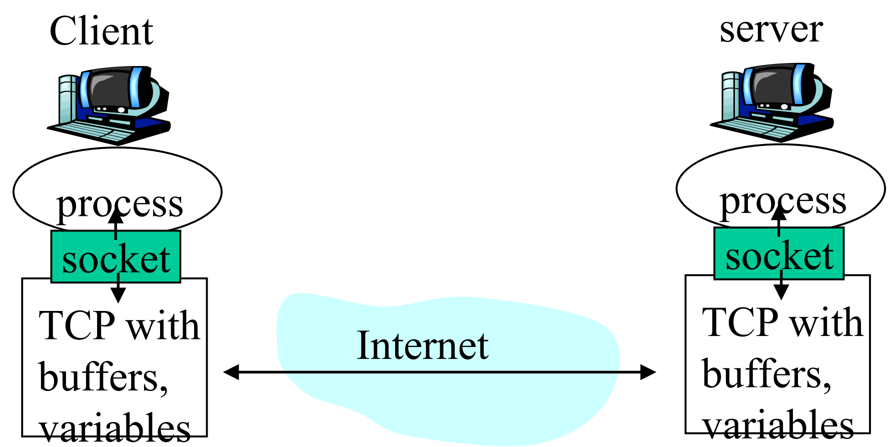
Web and HTTP

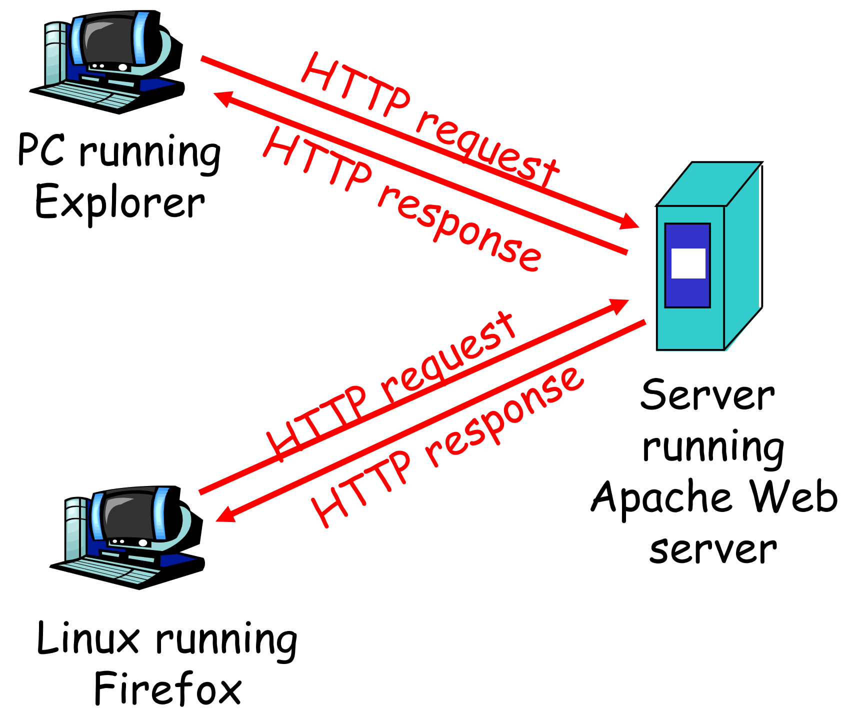
- A Web page consists of objects
- An object is simply a file such as HTML file, JPEG image, video clip, audio file,…
- Usually, a Web page consists of a base HTML file and several referenced objects
- Each object is addressable by a URL (Uniform Resource Locator).
- Example URL:

$$\underbrace{\texttt{www.um.edu.mo}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP Overview

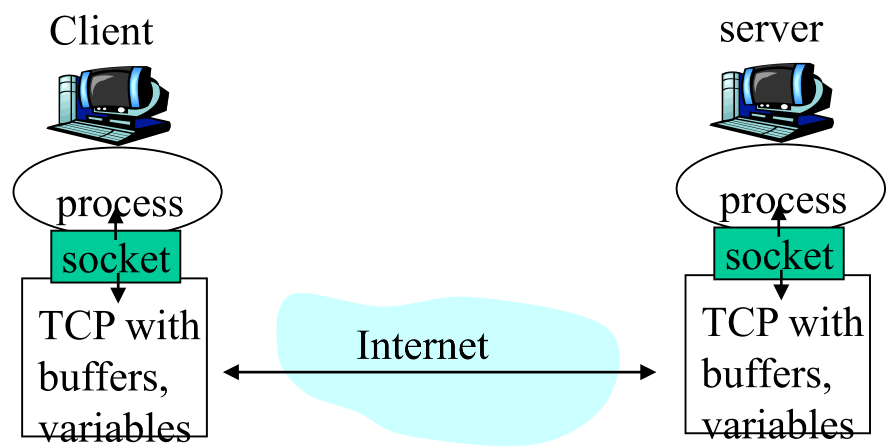
- It is the Web's application layer protocol
- It works at client/server mode
  - Client and server processes run on different end systems.
  - Communicate with each other by exchanging HTTP messages
  - Web browsers (client) implement the client side of HTTP
  - Web servers (server) implement the server side of HTTP, and each object is addressed by a URL.

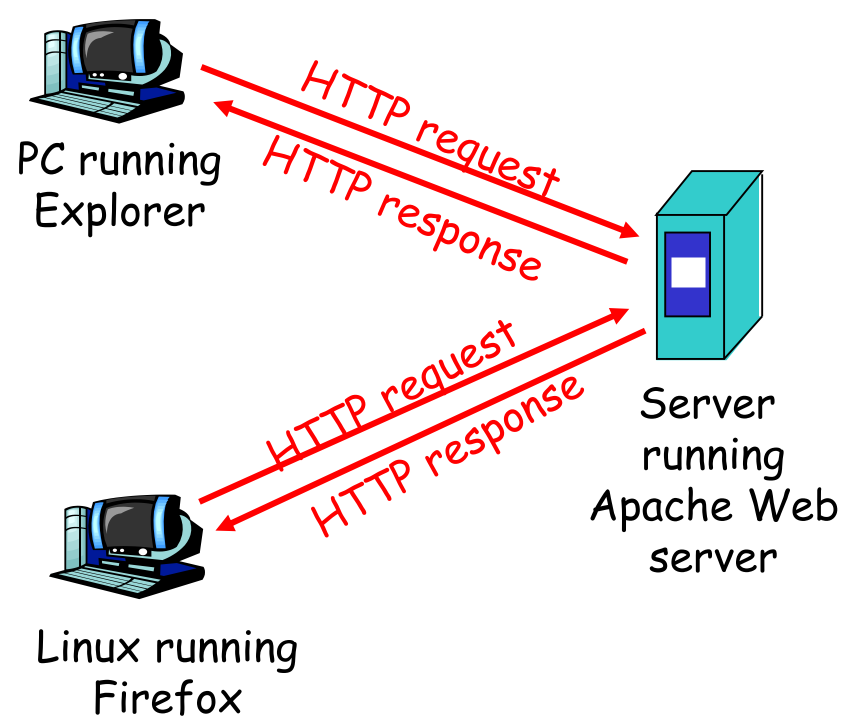




HTTP Overview

- It uses TCP as its transport layer protocol
  - client browser 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 connections

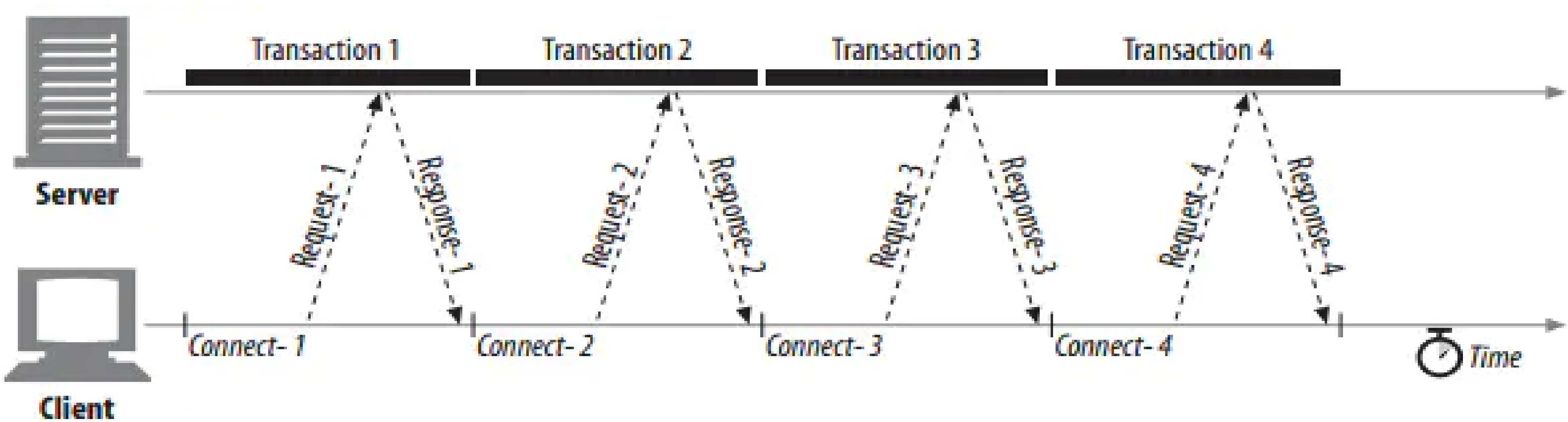
Non-persistent HTTP

- At most one object is sent over single TCP connection.
- HTTP/1.0 uses non-persistent HTTP

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

HTTP Connections



Non-persistent HTTP

Non-persistent HTTP

Suppose user enters URL

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

(contains a base HTML 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 (cont.)

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

- A new connection must be established for each requested object
- OS must work and allocate host resources for each TCP connection

Persistent HTTP

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between the same client/server are sent over this connection

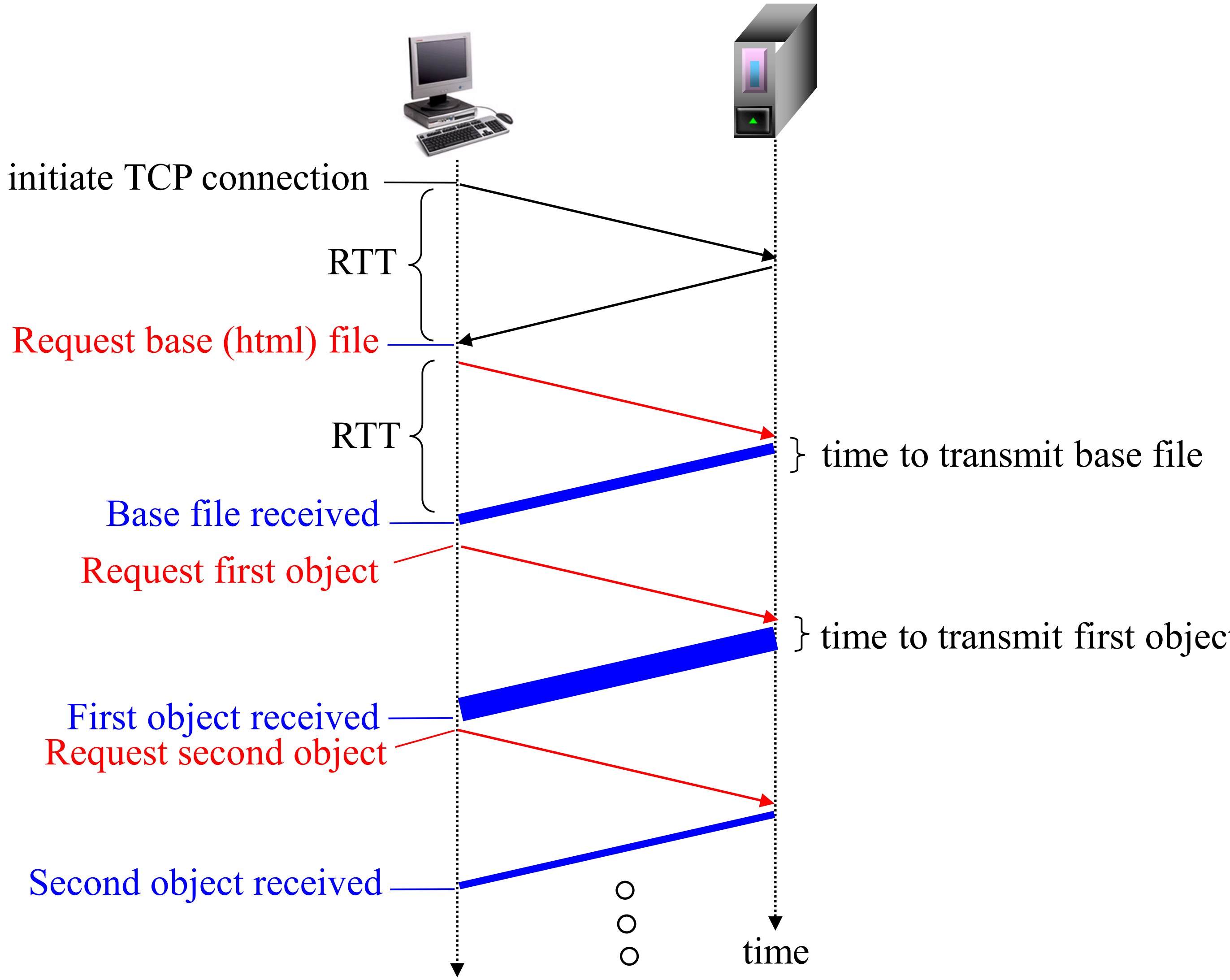
Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:

- client sends requests as soon as it encounters a referenced object
- Reduce the web response time
- default in HTTP/1.1

Persistent HTTP Without Pipeline



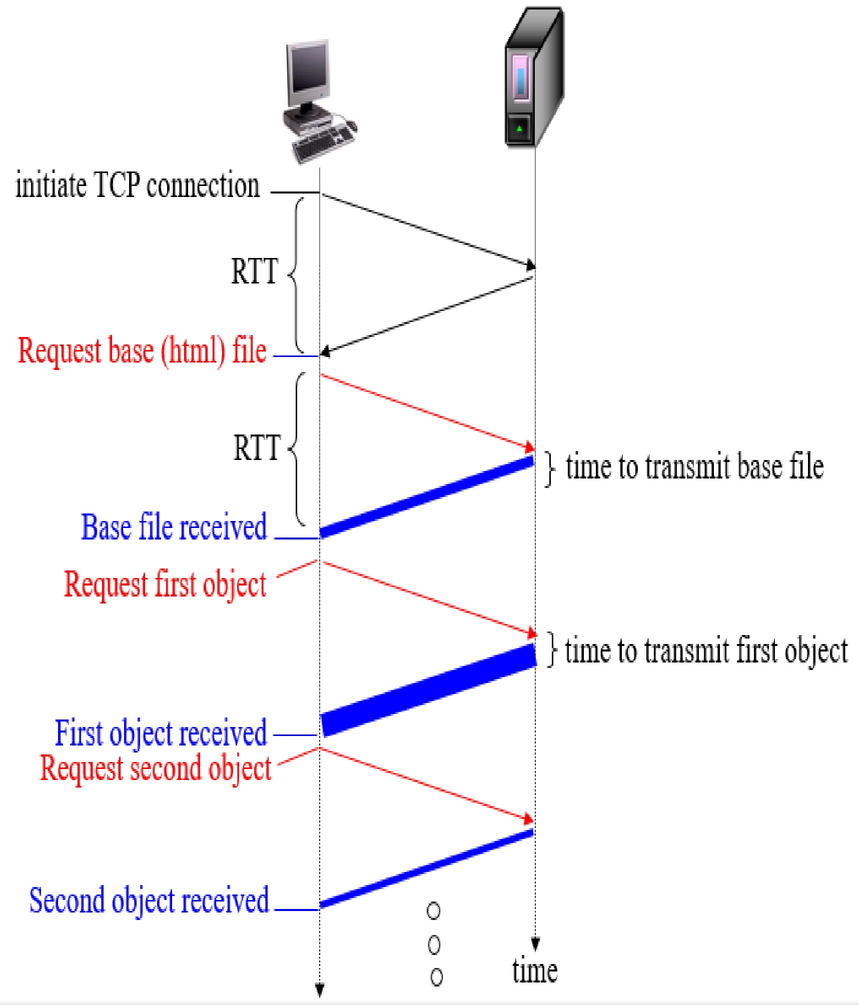
Definition of RTT (Round-trip Time)往返时延: time to send a small packet to travel from client to server and back.

Response Time Model: Persistent HTTP Without Pipeline

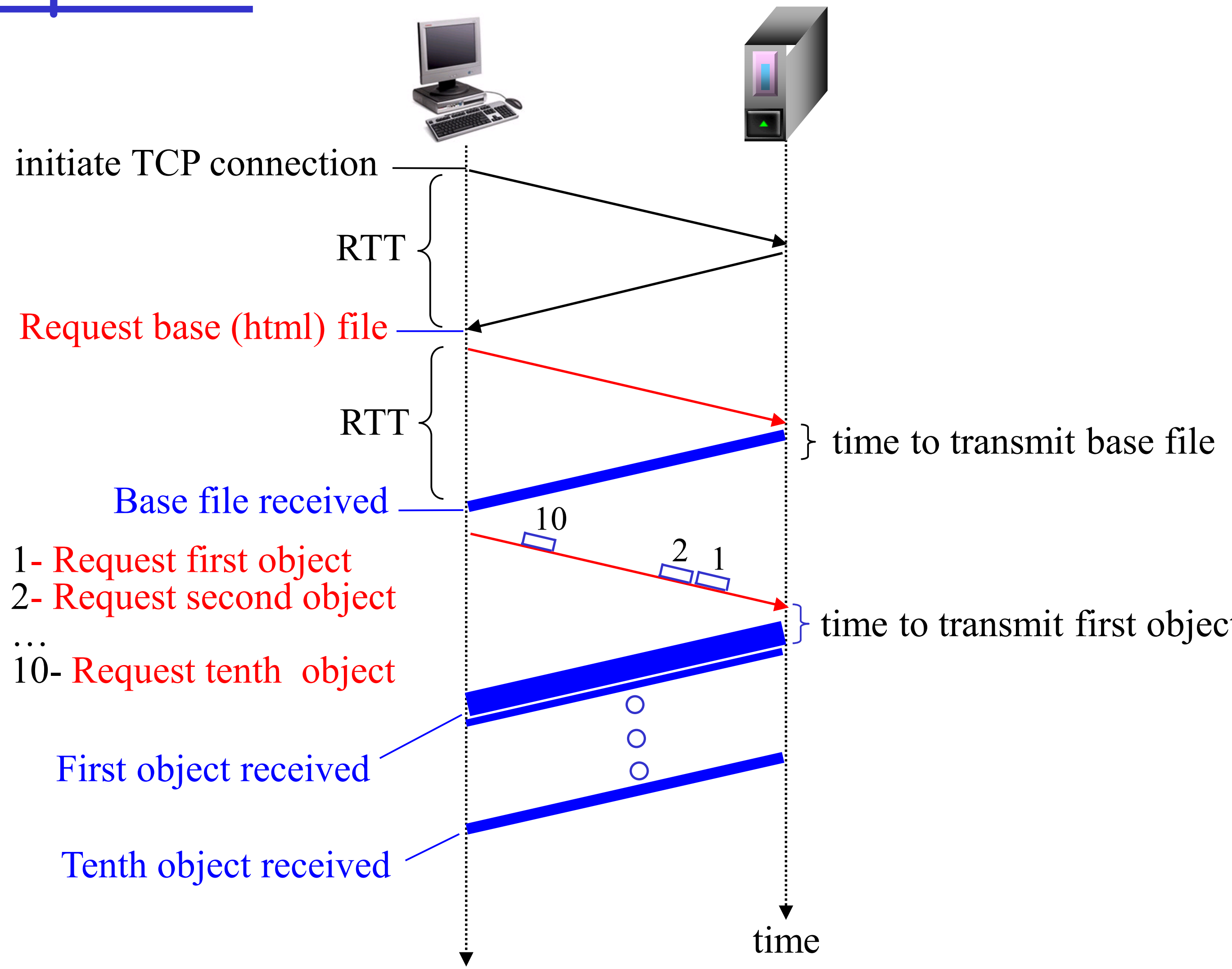
Response Time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- Time ≈ (N+2)RTT + (N+1) file/object transmit time

N = number of objects in the web page.



Response Time Model: Persistent HTTP With Pipeline



Response Time ≈ 3RTT+ (N+1) file transmit time

HTTP Overview

HTTP is “stateless” (无状态) protocol

- server maintains no information about past client requests

Advantages

- Simplify the server design
  - No need to allocate storage to maintain past history.
  - No need to synchronize the state with clients if they crash
- Server can handle huge number of service requests from different browsers

Disadvantage

- Cannot keep track of users’ history and provide more intelligent service.
- May be necessary to resend the repeated information.

User-server state: cookies

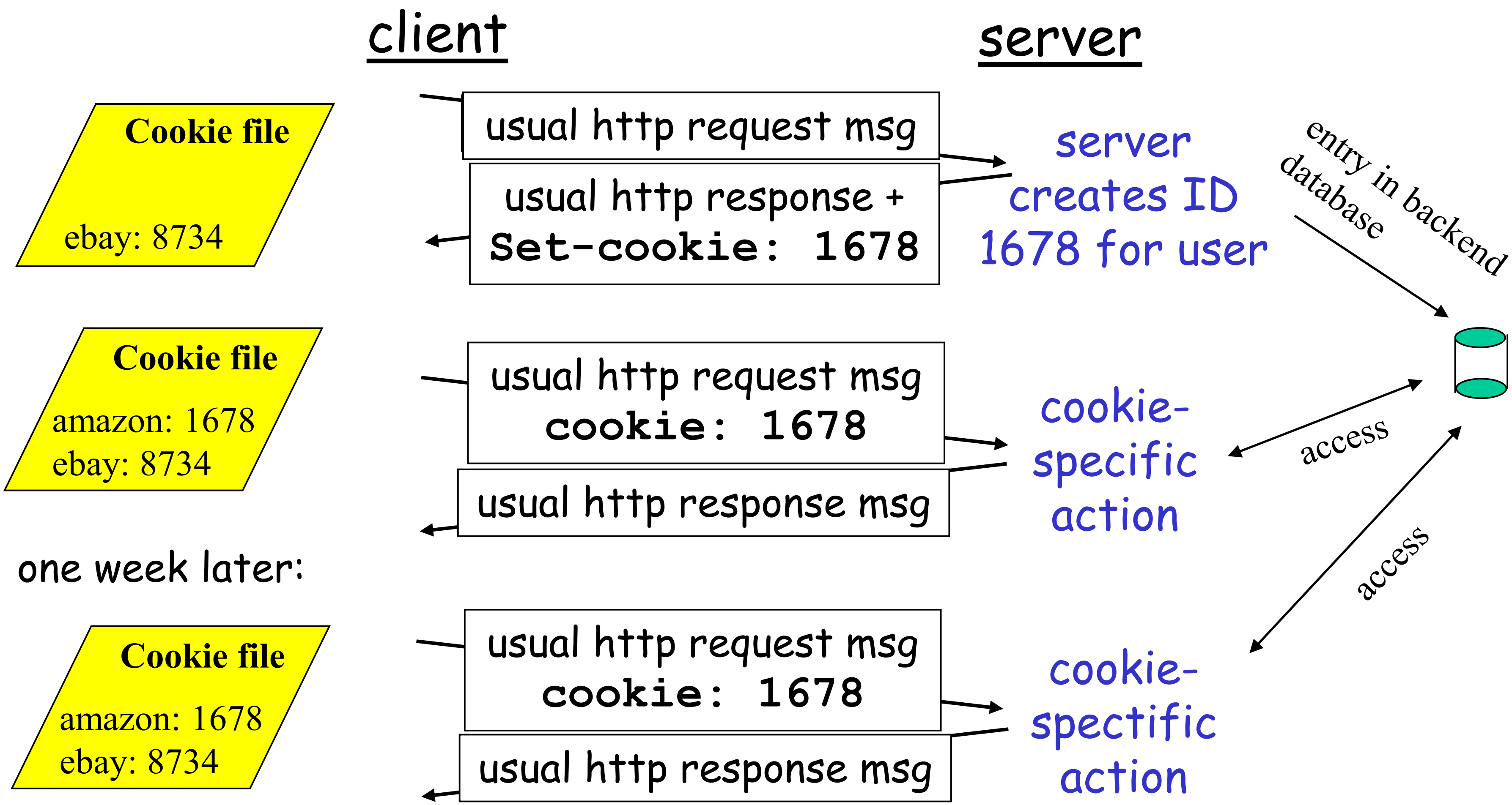
Some Web sites store information in a small text file on your computer. This file is called a cookie.

Many major Web sites use cookies

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site at the first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for her ID

Cookies: keeping "state" (cont.)



User-server state: cookies

Four components about Cookie:

1) cookie header line in the HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user's host and managed by user's browser
4) back-end database at Web site

Web caching 缓存(proxy server代理服务器)

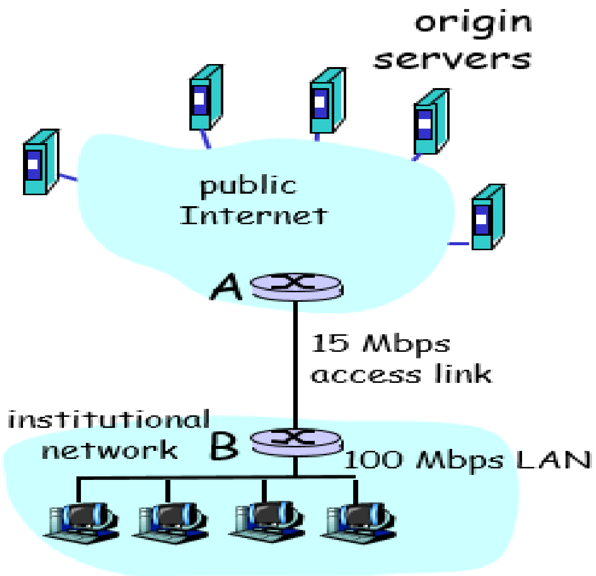
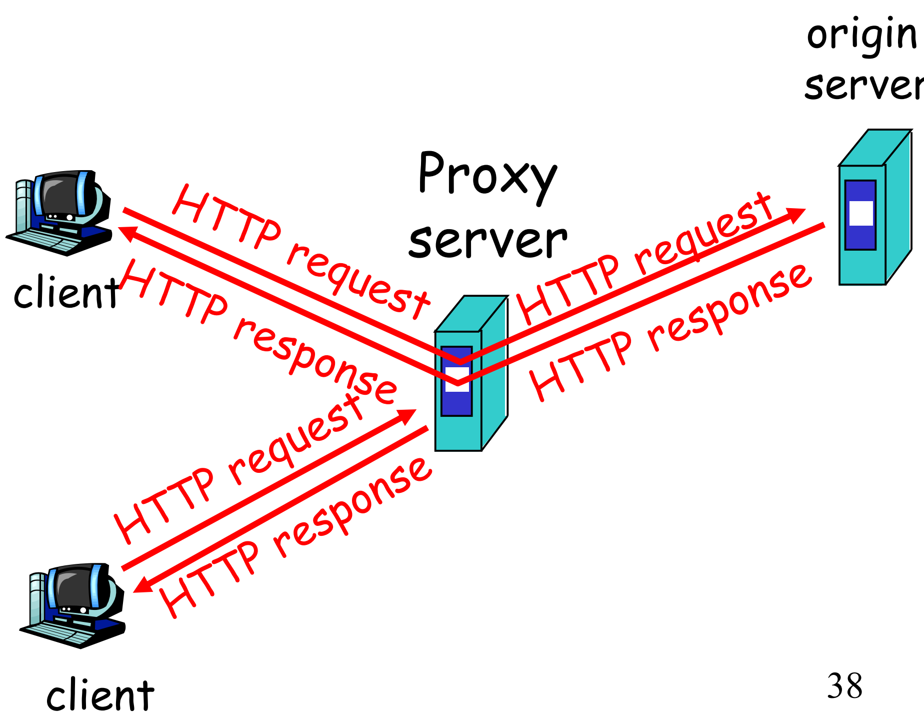


Fig. Bottleneck between an institutional network and the Internet

Web cache: is also called as a proxy server, it is a network entity that answers HTTP requests on behalf of an origin Web server.

Goal: answer client request without involving origin server

- Browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - Otherwise, cache requests object from origin server, then returns object to client



Why Do We Need Web Caching?

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

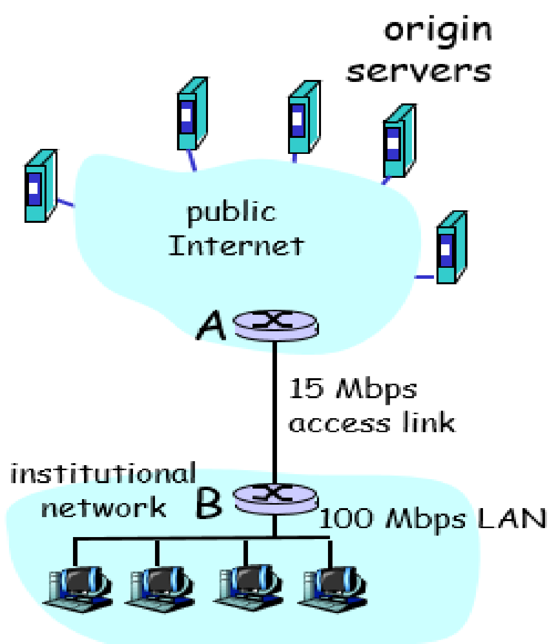
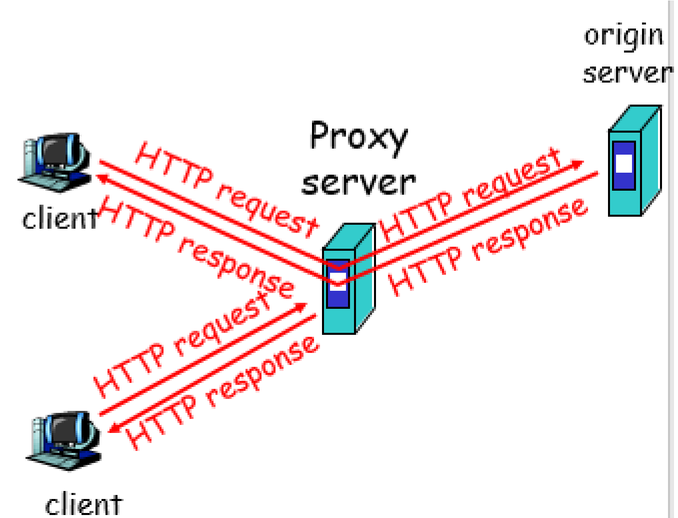


Fig. Bottleneck between an institutional network and the Internet

Why Web caching?

- Reduce traffic on an institution's access link to the Internet.
- Reduce traffic in the Internet as a whole.
- Reduce response time for client requests.



Caching example

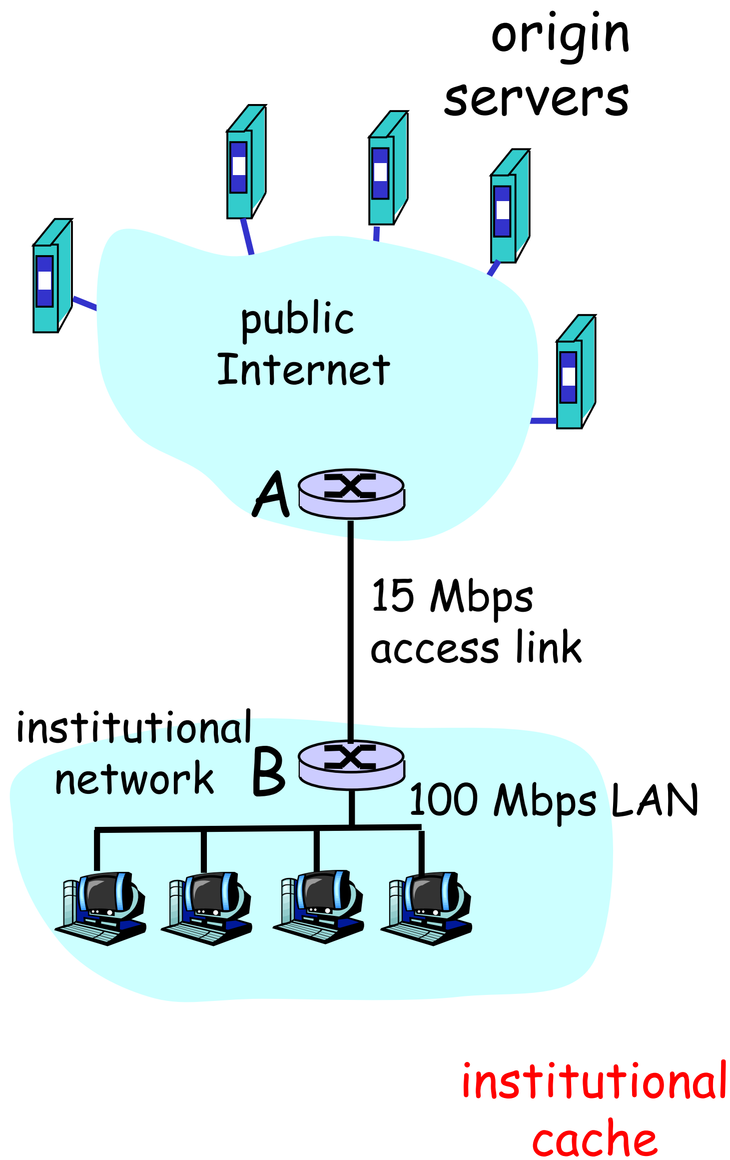
Assumptions

- average object size = 1Mb=1000,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from router A to any origin server and back to the router A = 2 sec
- Assume that the traffic intensity is 15%, 60%,and 100%, the delay is 1ms, 10ms, and 1 minute, respectively.

Delay

- Traffic intensity on the LAN (i.e., utilization on LAN) = 15%
- Traffic intensity on the access link (i.e., utilization on access link) = 100%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + 1 minute + 1 milliseconds



Caching example (cont)

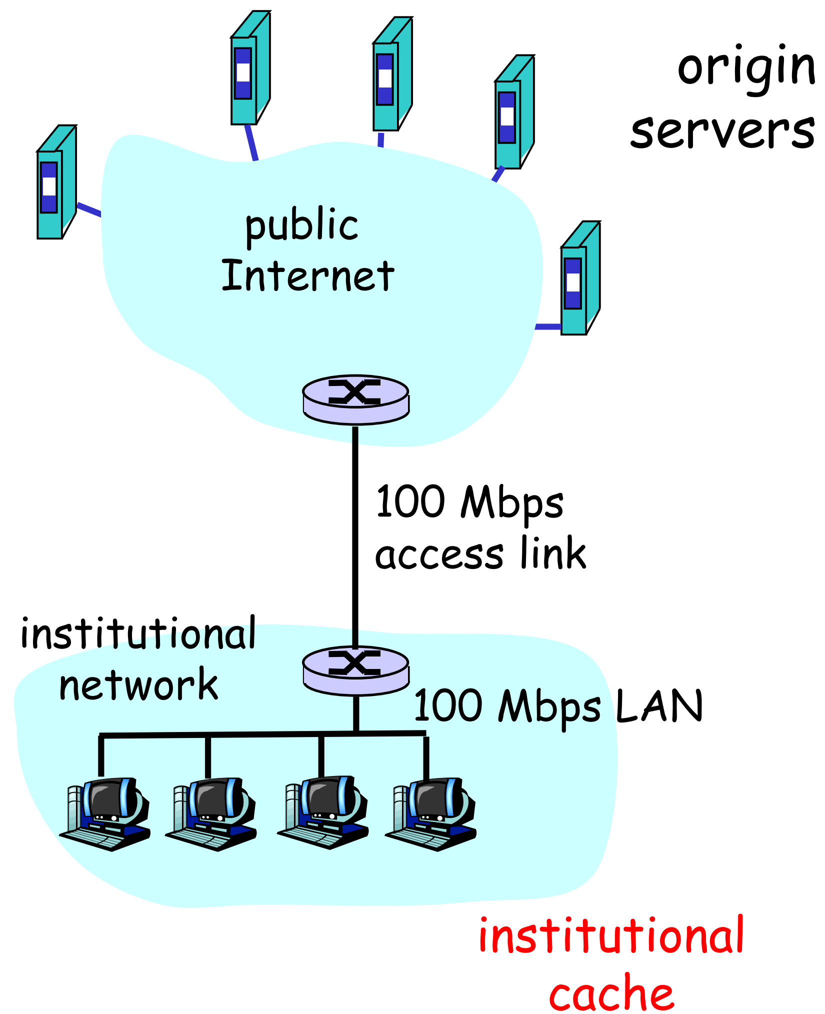
Possible solution

- increase bandwidth of access link to 100 Mbps

Consequences

- Traffic intensity on LAN = 15%
- Traffic intensity on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + 1ms + 1ms
- often a costly upgrade



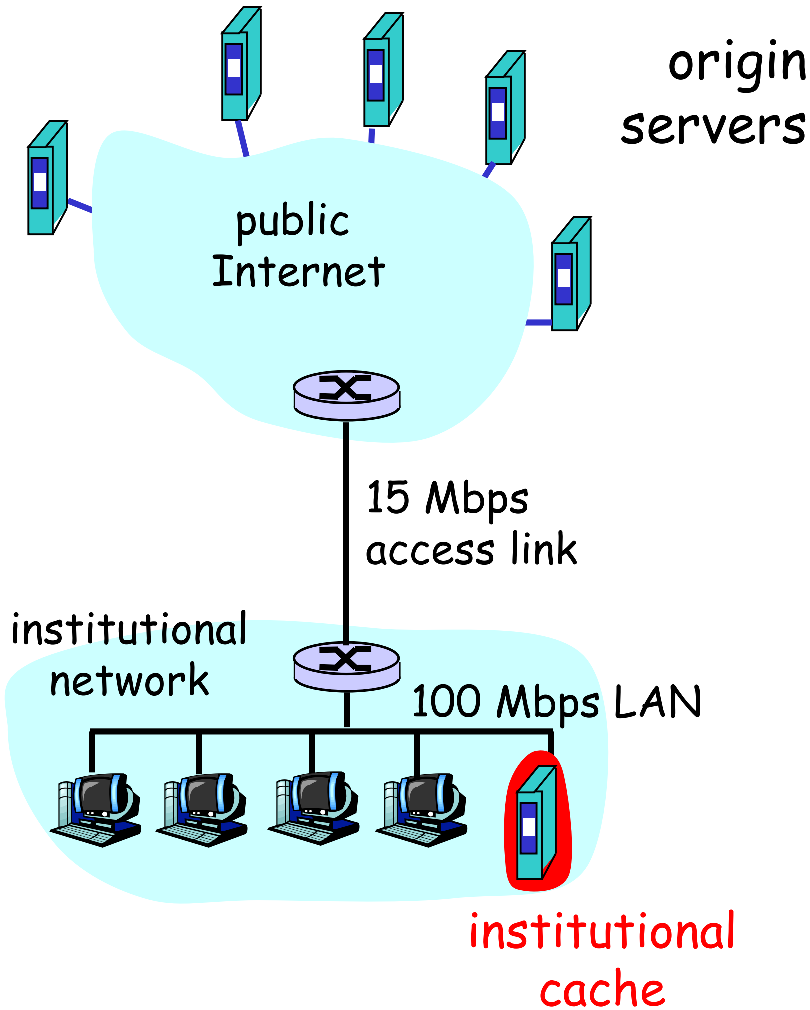
Caching example (cont)

Install cache

- Assume that the hit rate is 40%

Consequence

- 40% requests will be responded almost immediately
- 60% requests responded by origin server
- utilization of access link reduced to 60%, highly reducing the delays (say 10 ms)
- The average delay = Internet delay + access delay + LAN delay = 0.6*(2 + 0.01 + 0.001) secs + 0.4 * (0.001) secs < 2secs



Caching Challenges

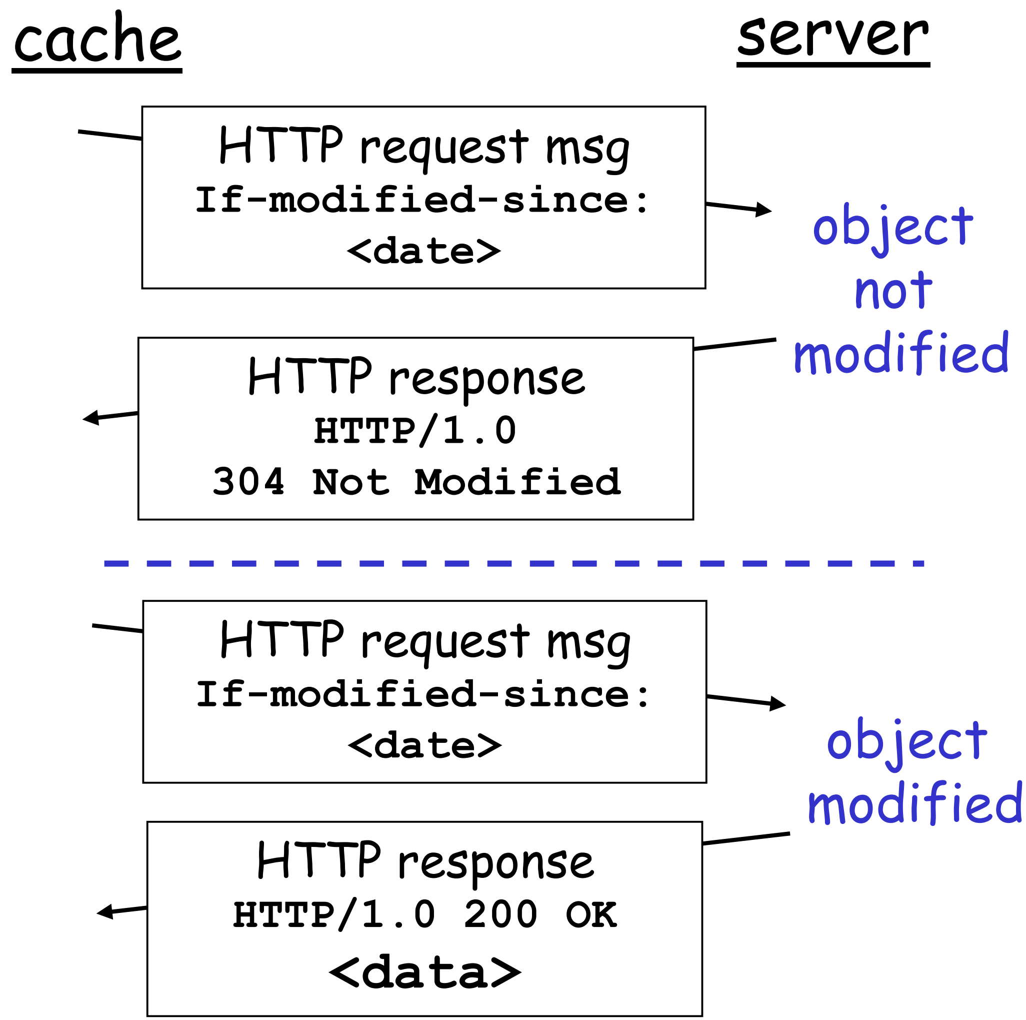
- Cache Consistency
  - Whether a stored objected is stale or fresh?
- Cache storage management

Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: do an up-to-data check by sending a conditional GET with specifying 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`



Caching Challenges

- Cache Consistency
  - Whether a stored objected is stale or fresh?

- Cache storage management
  - Caches must achieve high hit probability.
  - Once the caches is full, objects must be removed to make room to cache new responses.
  - Different replacement rules
    - Cost of fetching an object
    - Cost of storing an object
    - The number of accesses to the objects in the past
    - The probability of the object to be accessed in the near future
    - Expiration time
    - ….

Summary

- Principle of network application
- Application architectures
  - client-server
  - P2P
  - hybrid
- application service requirements:
  - reliability, bandwidth, delay, security
- Internet transport service model
  - connection-oriented, reliable: TCP
  - Connectionless, unreliable: UDP
- Specific application and protocol
  - Web and HTTP
  - Cookie, Web cache.