

Unit 2

▼ Networking Applications

- Text messaging
- e-mail
- multi-user network games
- video streaming platforms
- P2P file sharing
- Voice over ip (skype)
- Real-time video conferencing
- Web browsers
- remote login/access

▼ Creating a network app

- run on different end systems
- communicate over network
- ex. Web server software communicates with web browser software
- No need to write software for network core devices all the manipulation and support is done on application.

▼ Client - Server Paradigm

- Server:
 - always-on host
 - permanent IP address
 - often in data centers
- Client:
 - contact / communicate with server

- maybe intermittently connected
- may have dynamic IP address
- do not directly communicate with each other
- ex. HTTP, IMAP, FTP

▼ Peer-Peer architecture

- No Always-on servers
- end systems can directly communicate with each other
- 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
- ex. P2P file sharing

▼ Process Communication

- Process : Program running within a host
- within same host, two process communicate using Inter-process communication (Defined by os)
- Process in different hosts communicate by exchanging messages.

clients, servers

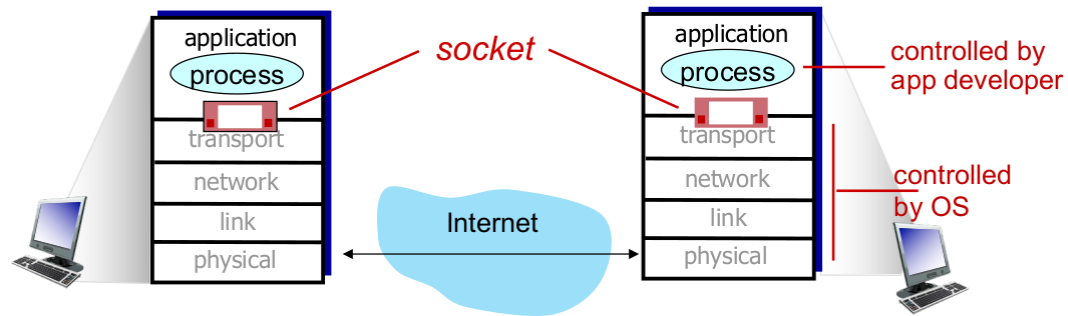
client process: process that initiates communication

server process: process that waits to be contacted

▼ Socket

- Process send / receive messages to / from it's socket
- Socket act as a door for end system towards network

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



▼ Addressing processes

- to receive msg, process must have identifier
- host devices has unique 32-bit IP address
- identifier includes both IP address and port number associated with process on host
- ex.
 - HTTP port: 80
 - FTP port : 21
 - HTTPS port : 443

▼ Application layer protocols

- File transfer protocol
 - Port numbers : 21 (for control connection), 20(for data connection)
- HTTP
 - Port number : 80
- SMTP
 - Port number : 25
- POP3

- Port number : 110

▼ 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
- 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 confidentiality.

▼ Internet Transport Layer services

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


▼ Security TCP connections

- Vanilla TCP & UDP Sockets:
 - No encryption
 - clear text passwords sent into socket traverse Internet in clear text (!)
- 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
 - clear text sent into "socket" traverse Internet encrypted

▼ Web & HTTP

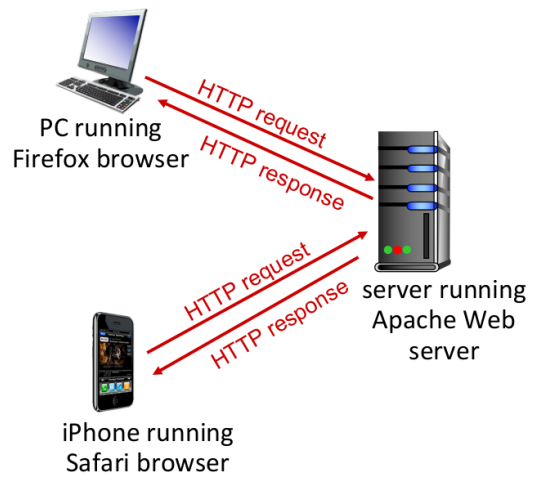
- web page consists of objects, each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file, etc
- web page consists of base HTML-file which includes several referenced objects, each addressable by a URL, e.g.,

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



▼ HTTP

- 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 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

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

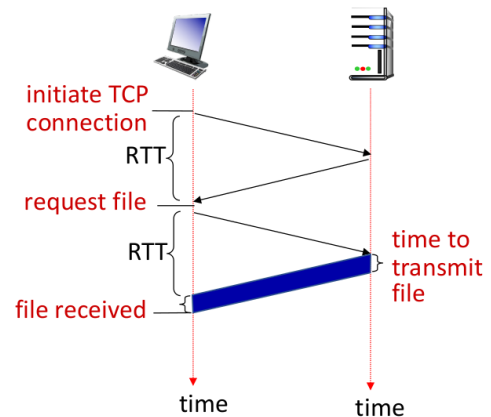
- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-persistent HTTP: 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 + file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

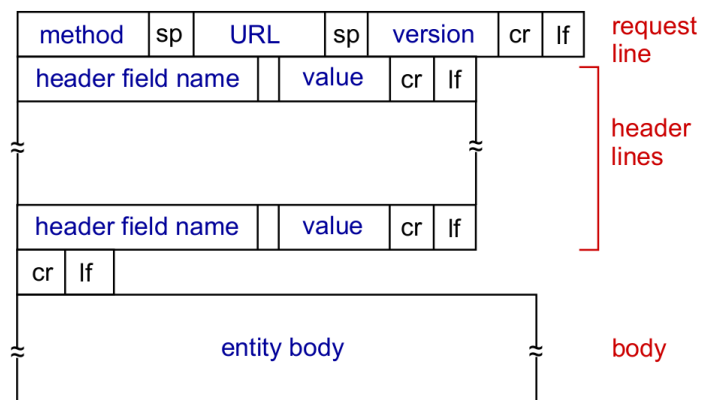
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

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)

- 2 types of HTTP Messages:
 - Request

HTTP request message: general format



Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

`www.somesite.com/animalsearch?monkeys&banana`

HEAD method:


- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

- Response

HTTP response message

status line (protocol  status code status phrase) **HTTP/1.1 200 OK**

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 message

301 Moved Permanently

- requested object moved, new location specified later in this message (in Location: field)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

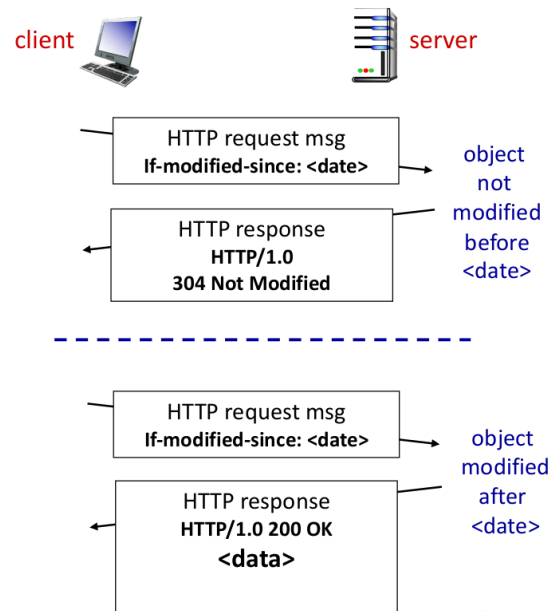
505 HTTP Version Not Supported

▼ Conditional GET

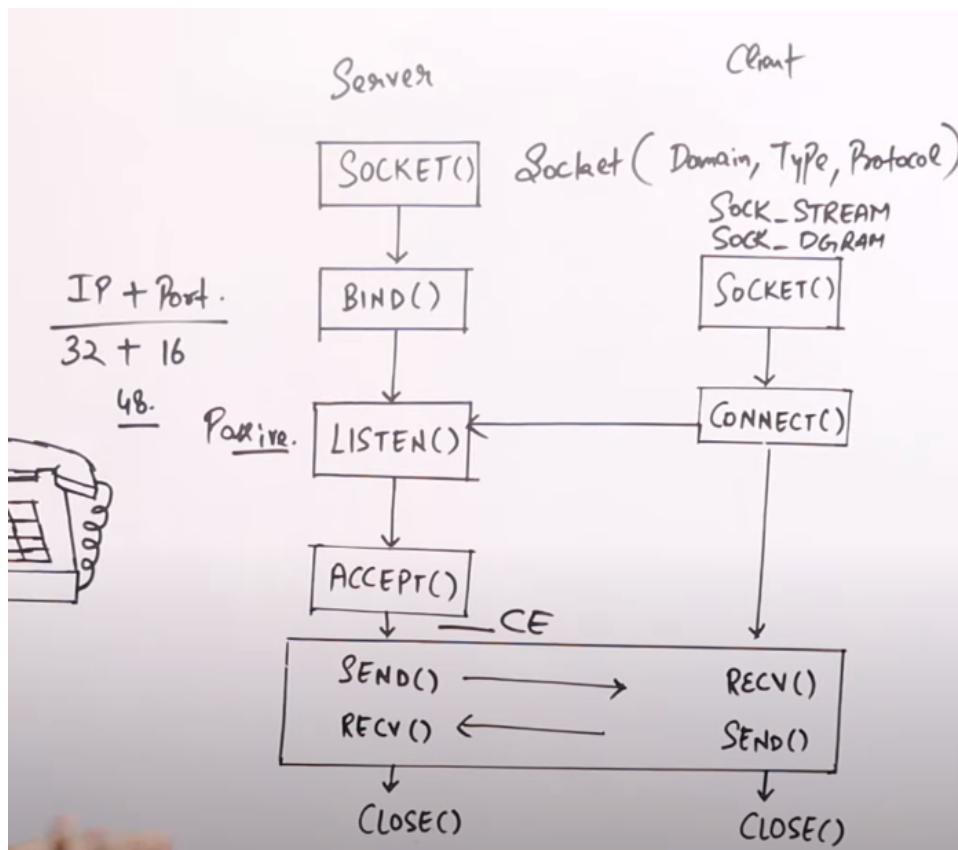
Conditional GET

Goal: don't send object if cache has up-to-date cached version

- no object transmission delay (or use of network resources)
- **client:** 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



▼ Socket programming



- Refer to video for more:

- <https://www.youtube.com/watch?v=XTVTIEhGS6w>

Not mentioned Topics

Cookies

Web caches

HTTP2 / HTTP3 (not necessary)

DNS ⇒ find it on youtube no documentation needed