ECCS-3631
Networks and Data Communications

Module 6: Application Layer and Socket Programming
Module 6-1: Client Server, P2P, SSL, TLS, HTTP

Dr. Ajmal Khan

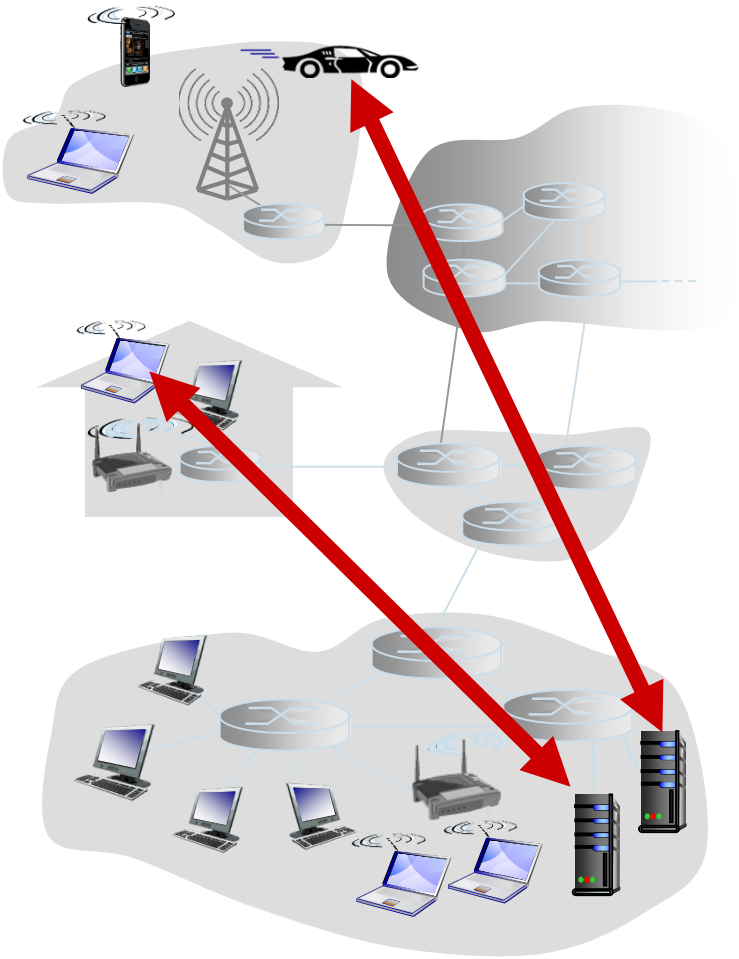
What is Network Applications?

- Network Applications are network software applications that utilize the Internet or other network hardware infrastructure to perform useful functions, for example file transfer within a network, sending email, Facebook, Twitter, etc.
- These applications have been the driving force behind the Internet's success, motivating people in homes, schools, governments, and businesses to make the Internet an integral part of their daily activities.
- At the core of network application development is writing programs that run on different end systems and communicate with each other over the network. For example, in the Web application there are two distinct programs that communicate with each other: the browser program running in the user's host (desktop, laptop, tablet, smartphone, and so on); and the Web server program running in the Web server host.
- When developing your new application, you need to write software that will run on multiple end systems. This software could be written, for example, in C, Java, or Python. Importantly, you do not need to write software that runs on network core devices, such as routers or link-layer switches.

Network Applications Architectures

- The application architecture is designed by the application developer and dictates how the application is structured over the various end systems.
- An application developer will likely choose on one of the two predominant architectural paradigms used in modern network applications: the client-server architecture or the peer-to-peer (P2P) architecture.

What is Client-Server Architecture?



Server:

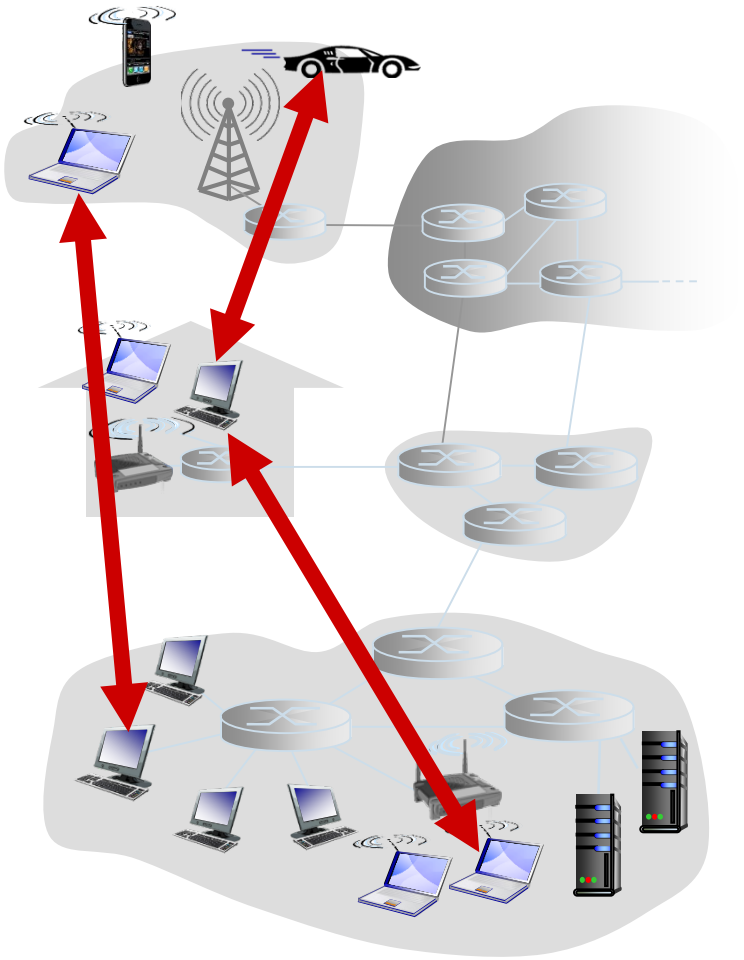
- always-on host
- permanent IP address
- data centers for scaling

Clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

What is P2P 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
- Examples: BitTorrent, IP Telephony,

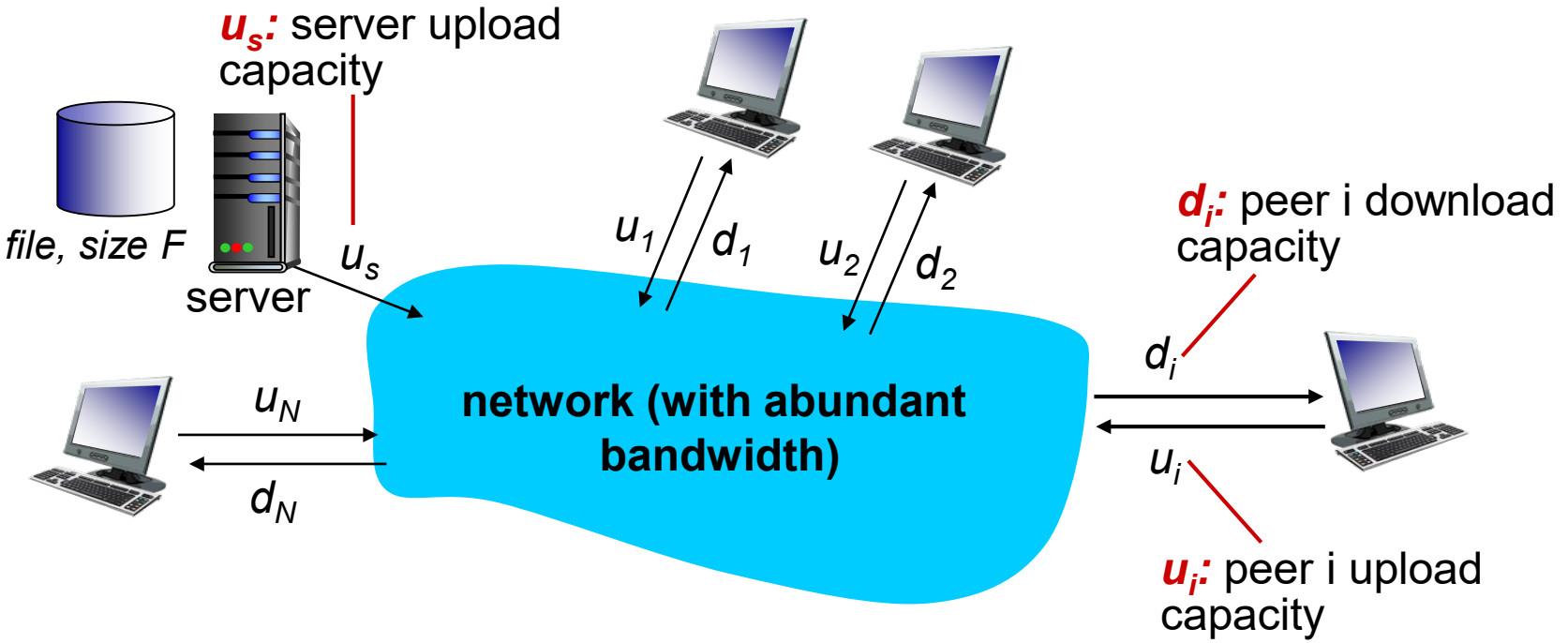


File Distribution: Client-Server vs P2P

Distribution Time: It is the time to get a copy of the file to all N peers.

Question: how much time to distribute file (size *F*) from one server to *N peers*?

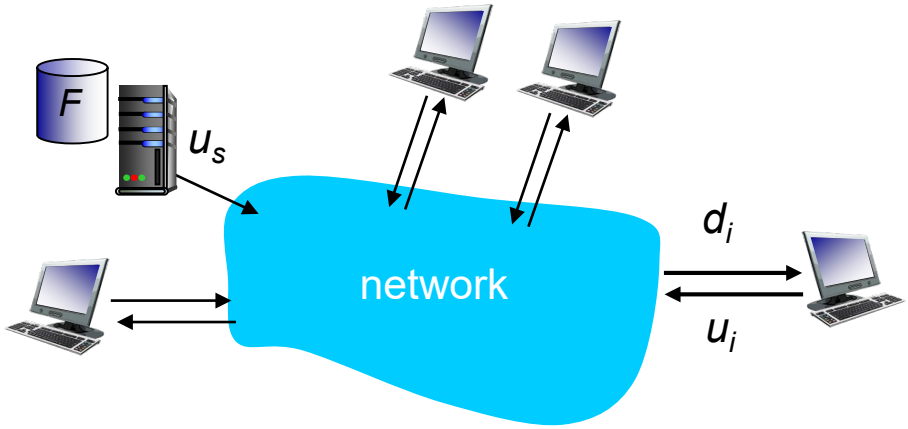
- peer upload/download capacity is limited resource



File Distribution: Client-Server

server transmission: must sequentially send (upload) N file copies

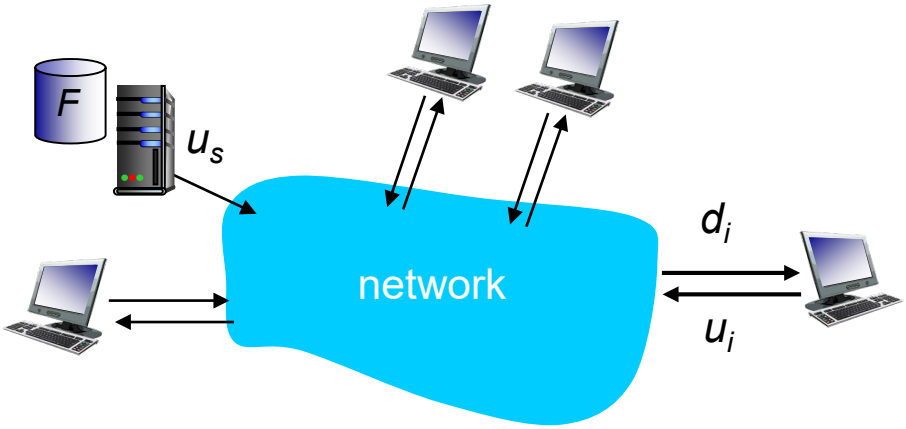
N is the number of clients



- *client:* each client must download file copy

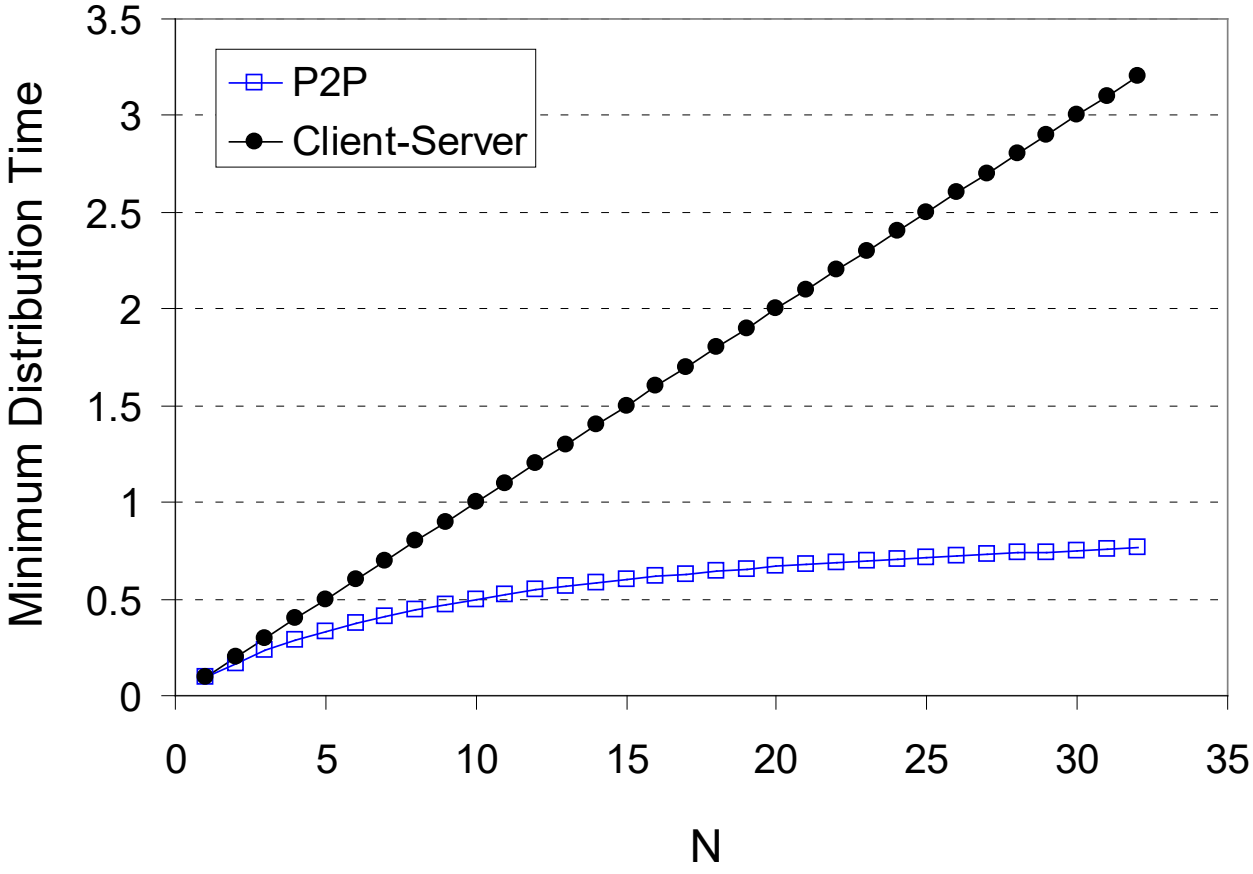
File Distribution: P2P

server transmission: must upload at least one copy



- *client:* each client must download file copy

Client-server vs. P2P: example



Processes Communicating

- *Process:* A process can be thought of as a program that is running within an end system (host).
- Within same host, two processes communicate using inter-process communication (defined by operating system), not the objective of this course
- processes in different hosts communicate by exchanging messages
- A sending process creates and sends messages into the network; a receiving process receives these messages and possibly responds by sending messages back.

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

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

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?

- *Ans:* 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 www.onu.edu web server:
  - IP address: 140.228.10.180
  - port number: 80

Application Layer Protocol Defines

types of messages exchanged,

- e.g., request, response

message syntax:

- what fields in messages & how fields are defined

message semantics

- meaning of information in fields

rules for when and how processes send & respond to messages

open protocols:

defined in RFCs

allows for interoperability

e.g., HTTP, SMTP

proprietary protocols:

e.g., Skype

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	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	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec 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

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,

Securing TCP

TCP & UDP

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

SSL

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

- When an application uses SSL, the sending process passes cleartext data to the SSL socket; SSL in the sending host then encrypts the data and passes the encrypted data to the TCP socket.
- The encrypted data travels over the Internet to the TCP socket in the receiving process.
- The receiving socket passes the encrypted data to SSL, which decrypts the data.
- Finally, SSL passes the cleartext data through its SSL socket to the receiving process.

SSL and TLS

- Functions of SSL: Encryption, Authentication, and Data Integrity
- SSL means Secure Sockets Layer. TLS means Transport Layer Security.
- Every SSL version is now deprecated. TLS versions 1.2 and 1.3 are actively used.
- SSL supports older algorithms with known security vulnerabilities. TLS uses advanced encryption algorithms.
- The SSL protocol uses the MD5 algorithm—which is now outdated—for Message Authentication Code (MAC) generation. TLS uses Hash-Based Message Authentication Code (HMAC) for more complex cryptography and security.
- MD5 hashes are 128 bits in length.
- Forward secrecy (FS), also known as perfect forward secrecy (PFS), is a cryptographic security feature that protects sensitive data by frequently changing the keys used to encrypt and decrypt information

Web and HTTP

What is web page?

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

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

host name | path name

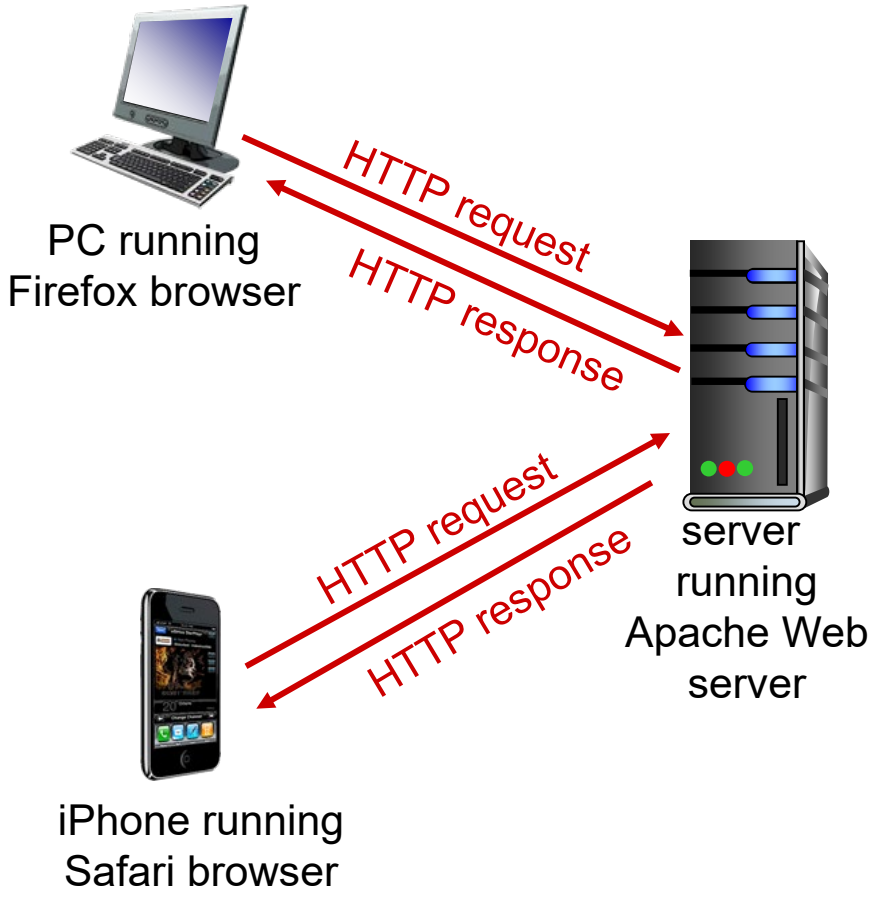
HTTP overview

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 (continued)

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

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

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

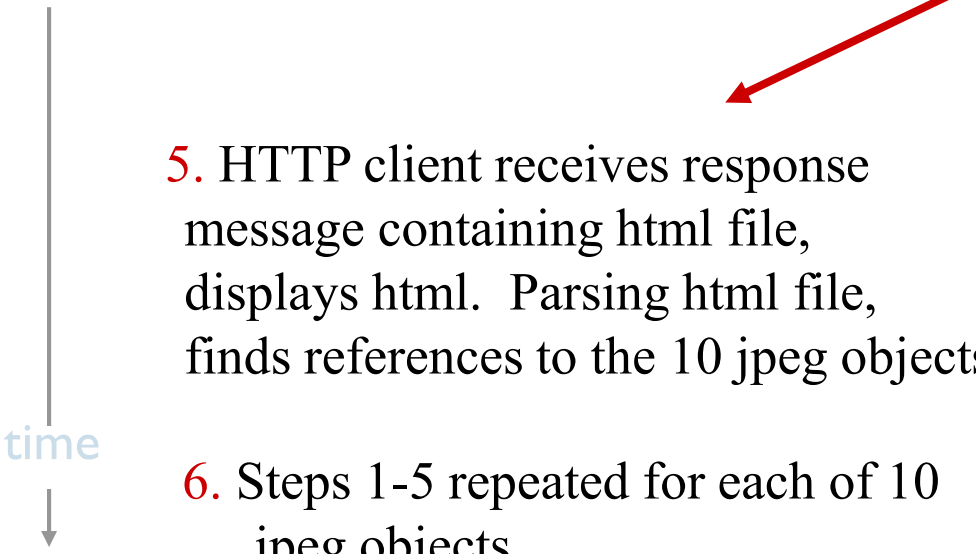
1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

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



4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds references to the 10 jpeg objects

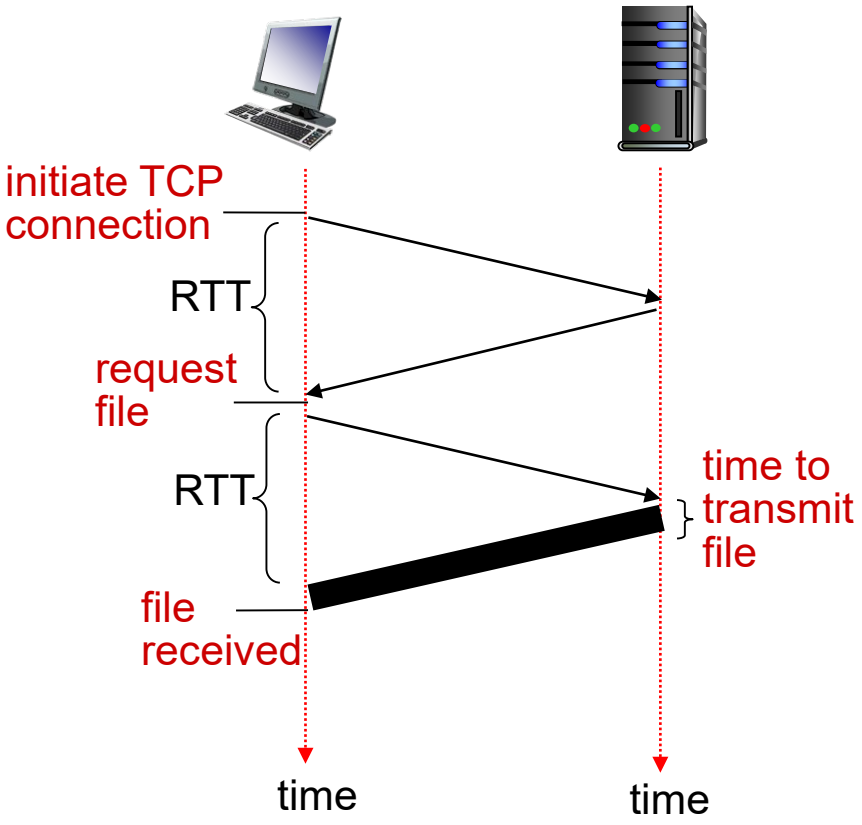
6. Steps 1-5 repeated for each of 10 jpeg objects

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
- non-persistent HTTP 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

persistent HTTP:

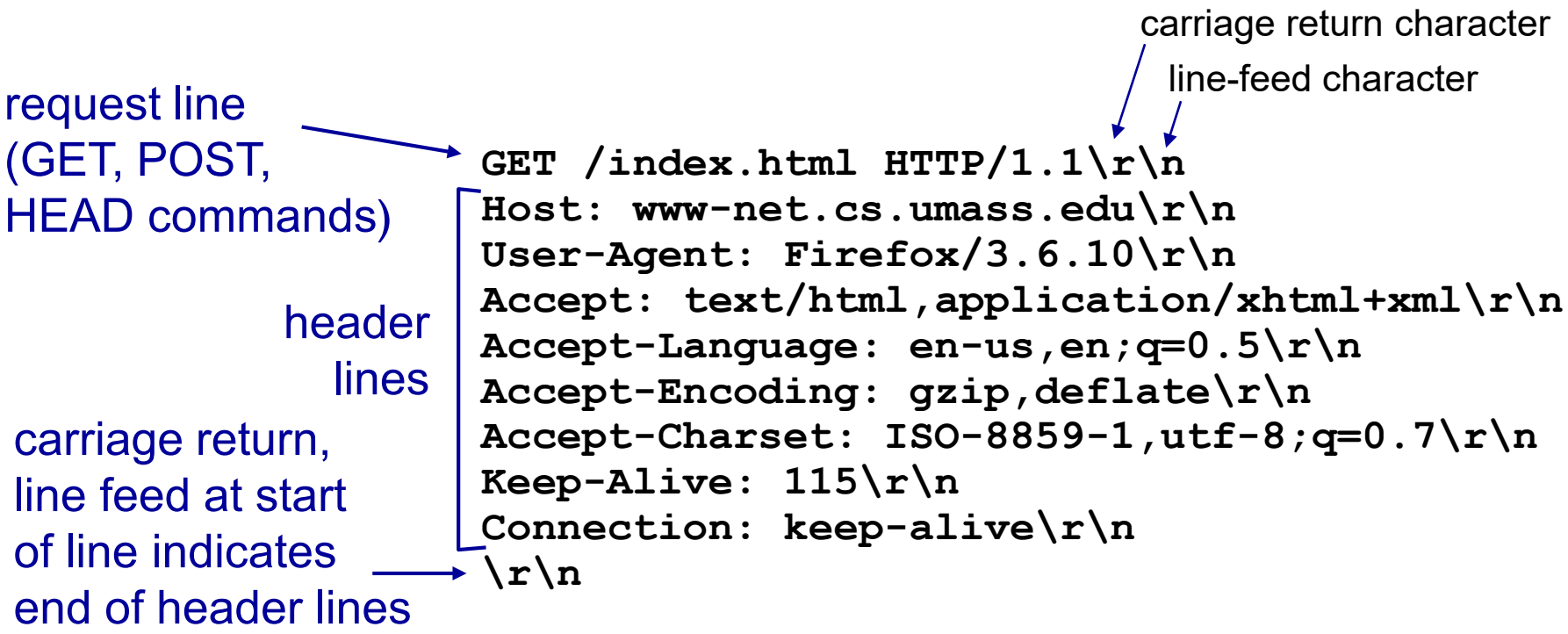
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP Request Message

two types of HTTP messages: *request*, *response*

HTTP request message:

- ASCII (human-readable format)



HTTP Response Message

status line (protocol 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