National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 05
Chapter 2

4th February, 2025

Nauman Moazzam Hayat
nauman.moazzam@lhr.nu.edu.pk

Office Hours: 11:30 am till 01:00 pm (Every Tuesday & Thursday)

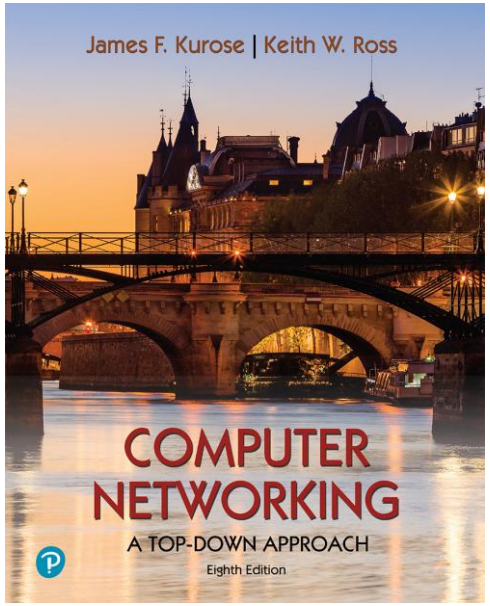
Chapter 2
Application Layer

A note on the use of these PowerPoint slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach
8th edition n
Jim Kurose, Keith Ross
Pearson, 2020

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS

- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

Application layer: overview

Our goals:

- conceptual *and* implementation aspects of application-layer protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-layer protocols and infrastructure
  - HTTP
  - SMTP, IMAP
  - DNS
  - video streaming systems, CDNs
- programming network applications
  - socket API

Some network apps

- social networking
- Web
- text messaging
- e-mail
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- P2P file sharing
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Internet search
- remote login
- …

Q: your favorites?

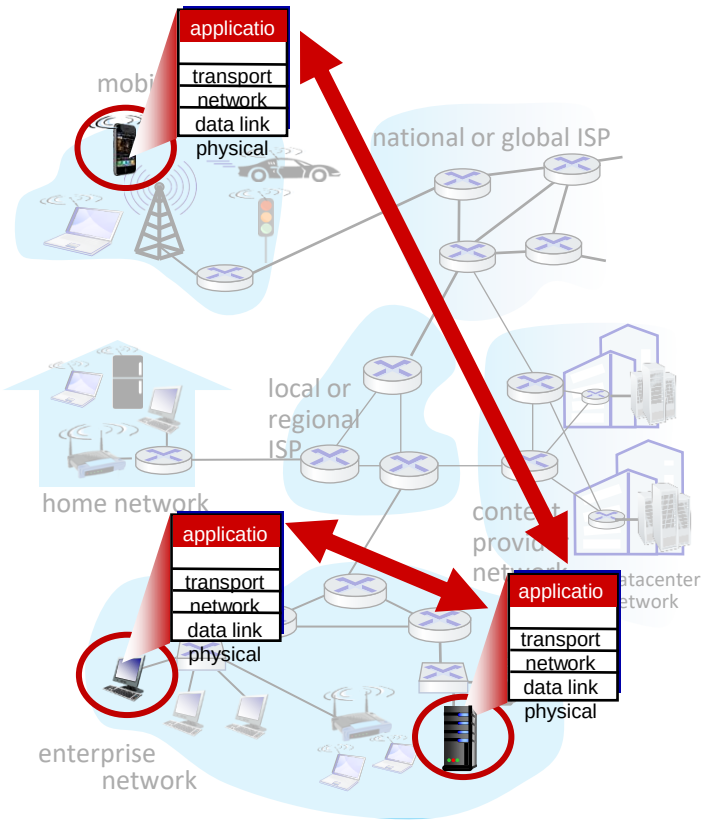
Creating a network app

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



Client-server paradigm

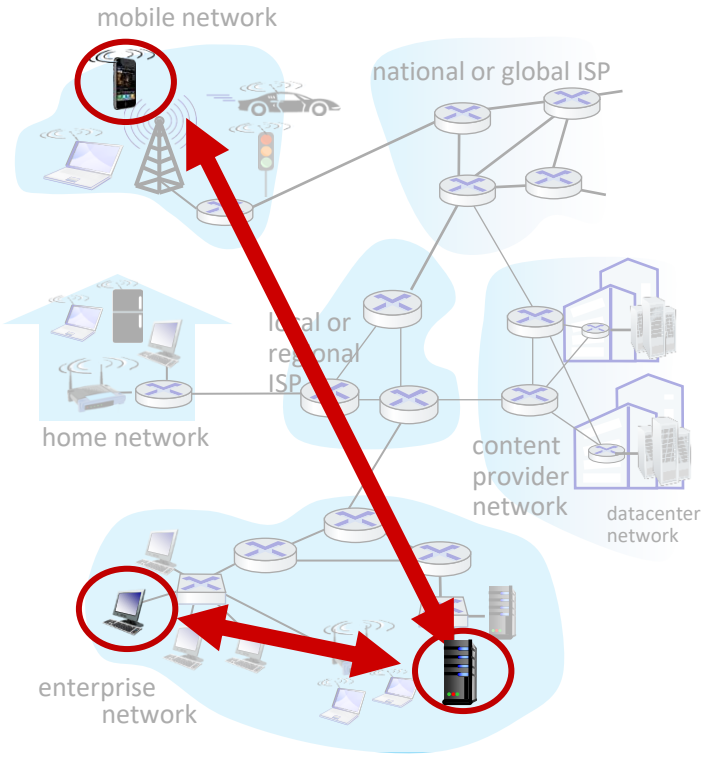
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

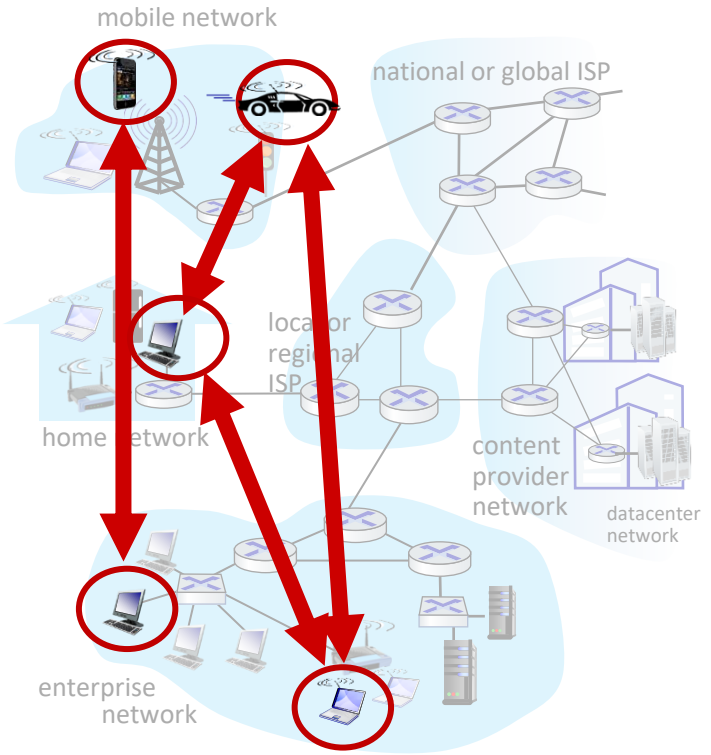
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



Peer-peer 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
- example: P2P file sharing [BitTorrent]



Network Edge (Client, Server, Peer)

Network edge comprises of the millions and billions of end systems / hosts and applications which reside in them

An end system (or host) can either request service (***client***) or provide service (***server***) or act as both interchangeably (***peer***).

Server

- A server is a service provider providing access to network resources:
  - A server can have multiple roles (e.g web servers, mail servers, print servers, Remote Access Servers (RAS), Directory Servers (DNS) etc)
  - Always on host
  - Permanent IP address
  - Most servers reside in large data centres

Client

- A client is a requestor of these services
  - May be intermittently on
  - may have dynamic IP address
  - do not communicate directly with each other

Peer

- A Peer-to-Peer network doesn't have dedicated servers. All hosts are equal and they both provide and request service i.e. they have both client & server functionalities.
  - Not always on server
  - arbitrary end systems directly communicate
  - peers are intermittently connected and change IP addresses
  - complex management
  - Examples are Skype, BitTorrent, Napster

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS

- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

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

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

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

How do we distinguish between two or more processes running on the same host?

Port Numbers

Then what is the difference between Process ID (PID) & Port Numbers?

A process ID is a unique identifier assigned to a running process in an operating system. It is used to track and manage individual processes. On the other hand, a port number is a communication endpoint in a network, used to identify a specific process or service running on a computer. Port numbers are essential for enabling communication between different processes or services over a network. While a process ID is specific to a single operating system instance, a port number is used for network communication between multiple systems.

Let's consider a scenario where you have a web server running on your computer. The web server is a process that listens for incoming requests on a specific port number, typically port 80 for HTTP or port 443 for HTTPS.

In this case, the process ID (PID) is a unique identifier assigned to the running instance of the web server process. The operating system assigns this PID to keep track of the process, manage its resources, and allow interaction with it.

The port number, on the other hand, is used to identify the specific service or process within the network. When the web server process starts, it binds itself to a specific port number (e.g., port 80). This means that the web server process will listen for incoming requests on that port number.

When a client (such as a web browser) wants to access a webpage hosted by your web server, it initiates a connection to your computer's IP address on port 80. The operating system receives this incoming network request on port 80 and forwards it to the web server process associated with that port.

The web server process, identified by its PID, receives the incoming request, processes it, and sends back the requested webpage to the client over the same connection.

So, in summary, the PID identifies the running instance of the web server process on your computer, while the port number (in this case, port 80) allows incoming network requests to be directed to the correct process for handling and responding to those requests.

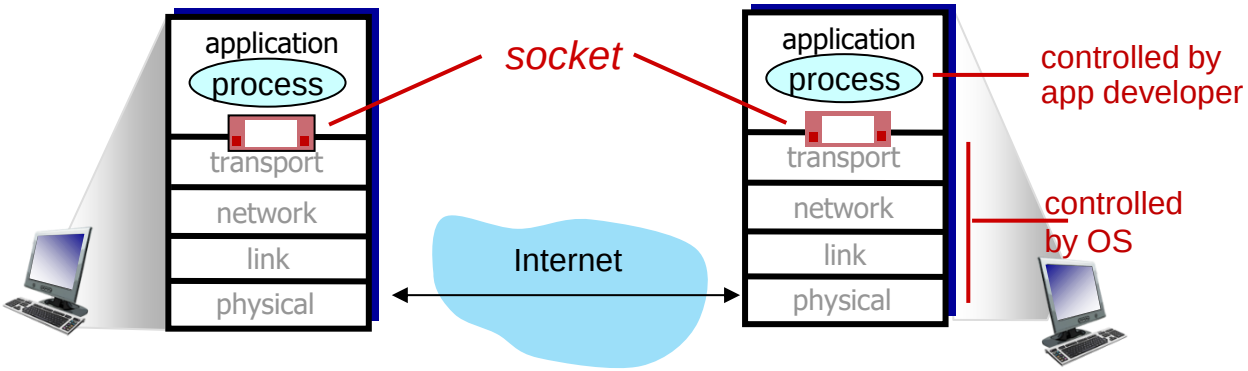
Addressing processes

- 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?
  - *A:* no, *many* processes can be running on same host

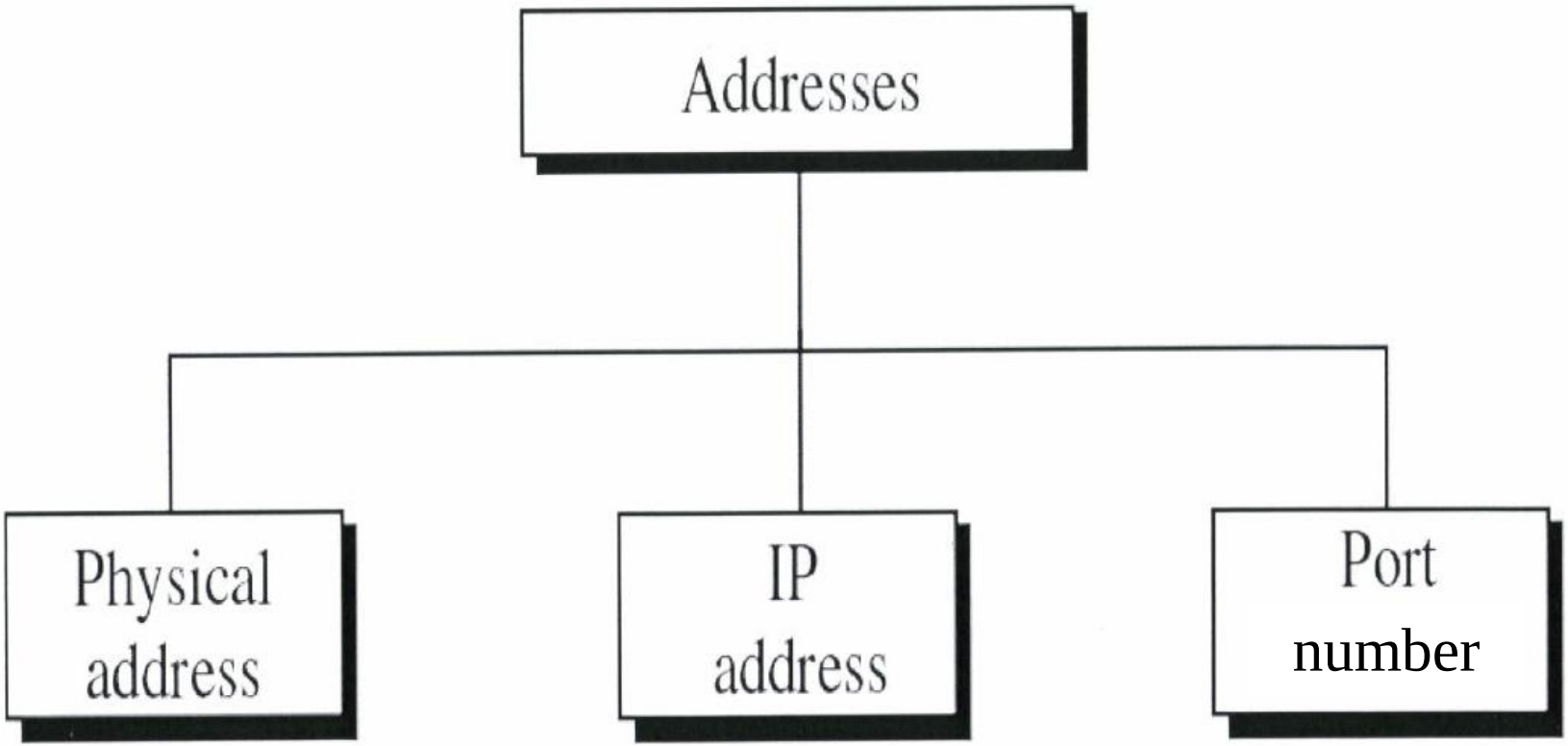
- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

Sockets (A combination of port number and IP address)

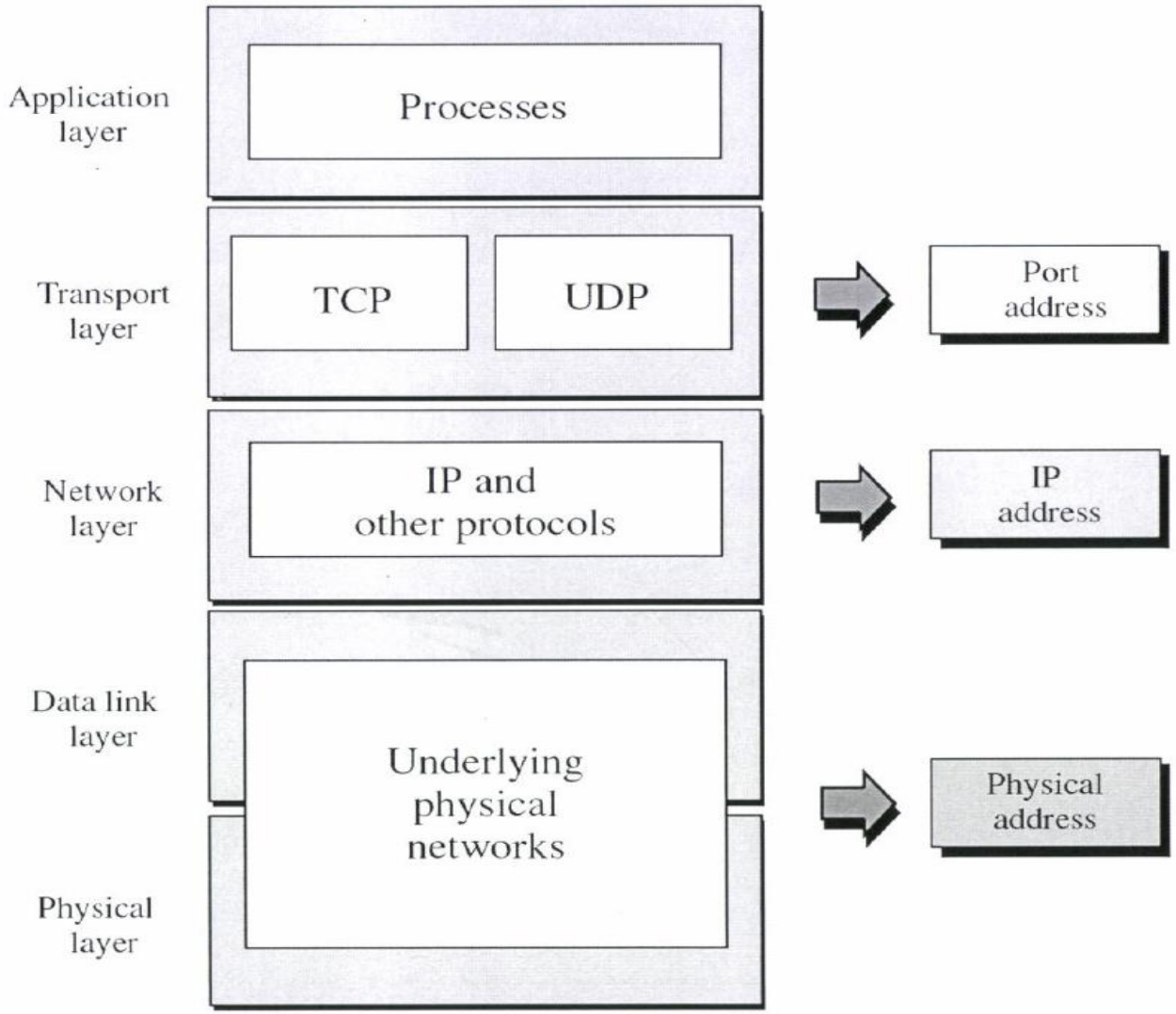
- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - two sockets involved: one on each side



Addressing in TCP/IP



TCP/IP Layers and Addresses



An application-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, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype, Zoom

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/download	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, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	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
- *connection-oriented:* setup required between client and server processes
- *does not provide:* timing, minimum throughput guarantee, security

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.

Q: why bother? *Why* is there a UDP?

Internet applications, and transport protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP [RFC 7230, 9110]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Securing TCP

Vanilla TCP & UDP sockets:

- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext (!)

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
- cleartext sent into "socket" traverse Internet *encrypted*
- more: Chapter 8

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

Web and HTTP

First, a quick review…

- 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,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

Uniform Resource Locator (URL)

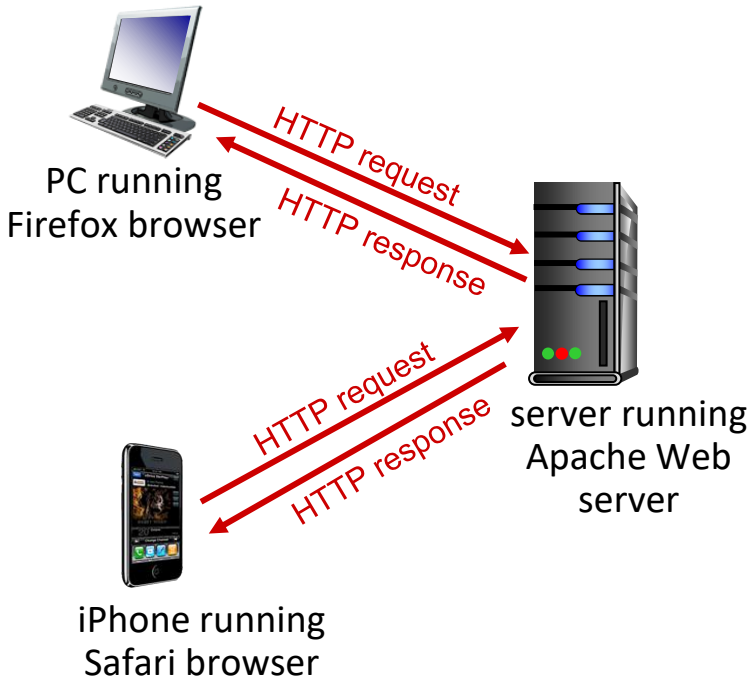
```
protocol://host-name[:port]/directory-
  path/resource
```

- *protocol*: http, ftp, https, smtp, rtsp, *etc*.
- *hostname*: DNS name (or domain name), IP address
- *port:* defaults to protocol's standard port; *e.g.* http: 80 https: 443
- *directory path*: hierarchical, reflecting file system (on the server side)
- *resource*: Identifies the desired resource

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)

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

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: two types

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: example

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

Non-persistent HTTP: example (cont.)

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**
(containing text, references to 10 jpeg images)

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

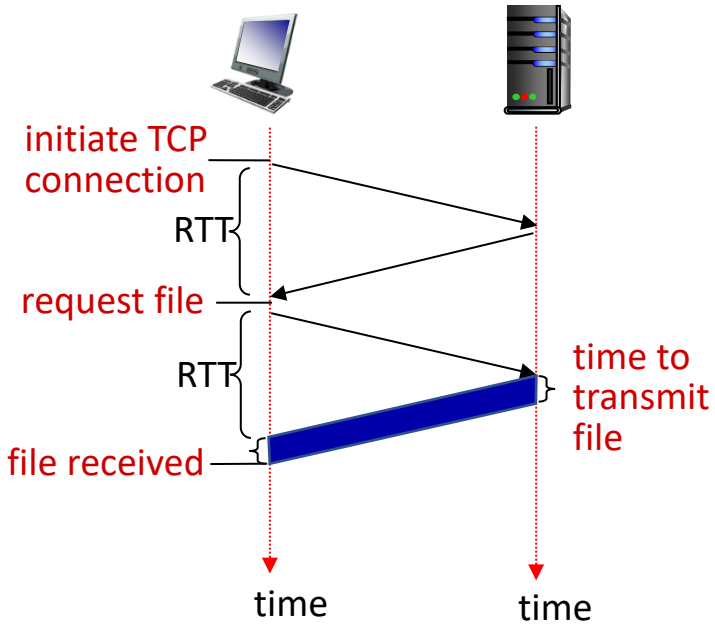
time

Non-persistent HTTP: 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

Non Persistent HTTP Shortcomings

- Most Web pages have multiple objects
  - *e.g.*, HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
  - *One item at a time*
- Brand New TCP connection per requested object! (even for small object), thus significant TCP resources need to be allocated at both server & client side (TCP buffers)
- Burden on Web Servers which are servicing multiple simultaneous clients
- Also each object suffers a delivery delay of 2 RTTs (one to establish the TCP connection and one to request & receive the object)

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)

Assignement # 1 (Chapter – 1) (Already Announced)

- *1st Assignment will be uploaded on Google Classroom after the lecture in the Stream Section, on Thursday 30th January, 2025*

- *Due Date: Thursday, 6th February, 2025 (During the lecture)*

- *Hard copy of the handwritten assignment to be submitted directly to the Instructor during the lecture.*

- *Please read all the instructions carefully in the uploaded Assignment document, follow & submit accordingly*

Quiz # 1 (Chapter – 1) (Already Announced)

- *Quiz # 1 for Chapter 1 to be taken in the class on Thursday, 6th February, 2025 during the lecture time (or in the next class in case of a public holiday.)*

No Retake

Be on time