



NETVÆRKS- OG KOMMUNIKATIONSSIKKERHED

TCP/IP

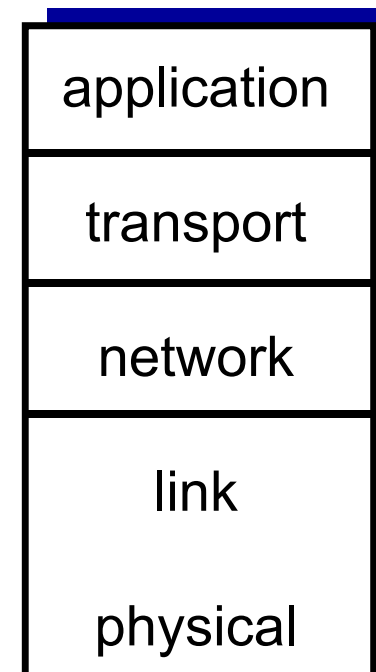


Agenda

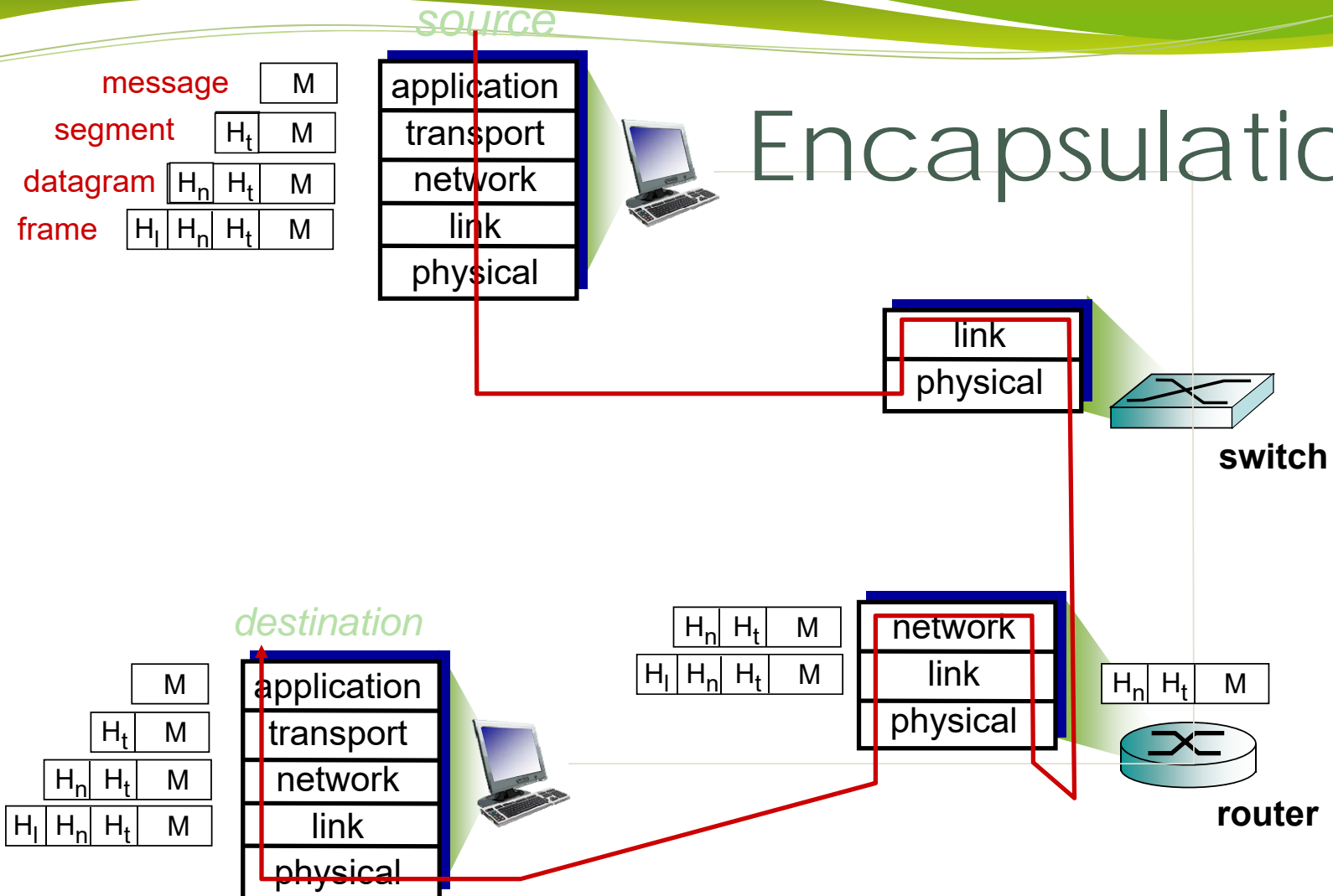
- Wireshark
- HTTP
- DNS
- SMTP
- Challenge
- TCP/UDP
- IP
- ARP

Internet protocol stack

- *application*: supporting network applications
 - FTP, SMTP, HTTP
- *transport*: process-process data transfer
 - TCP, UDP
- *network*: routing of datagrams from source to destination
 - IP, routing protocols
- *link*: data transfer between neighboring network elements
 - Ethernet, 802.111 (WiFi), PPP
- *physical*: bits “on the wire”



Encapsulation



App-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
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype



HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
 - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

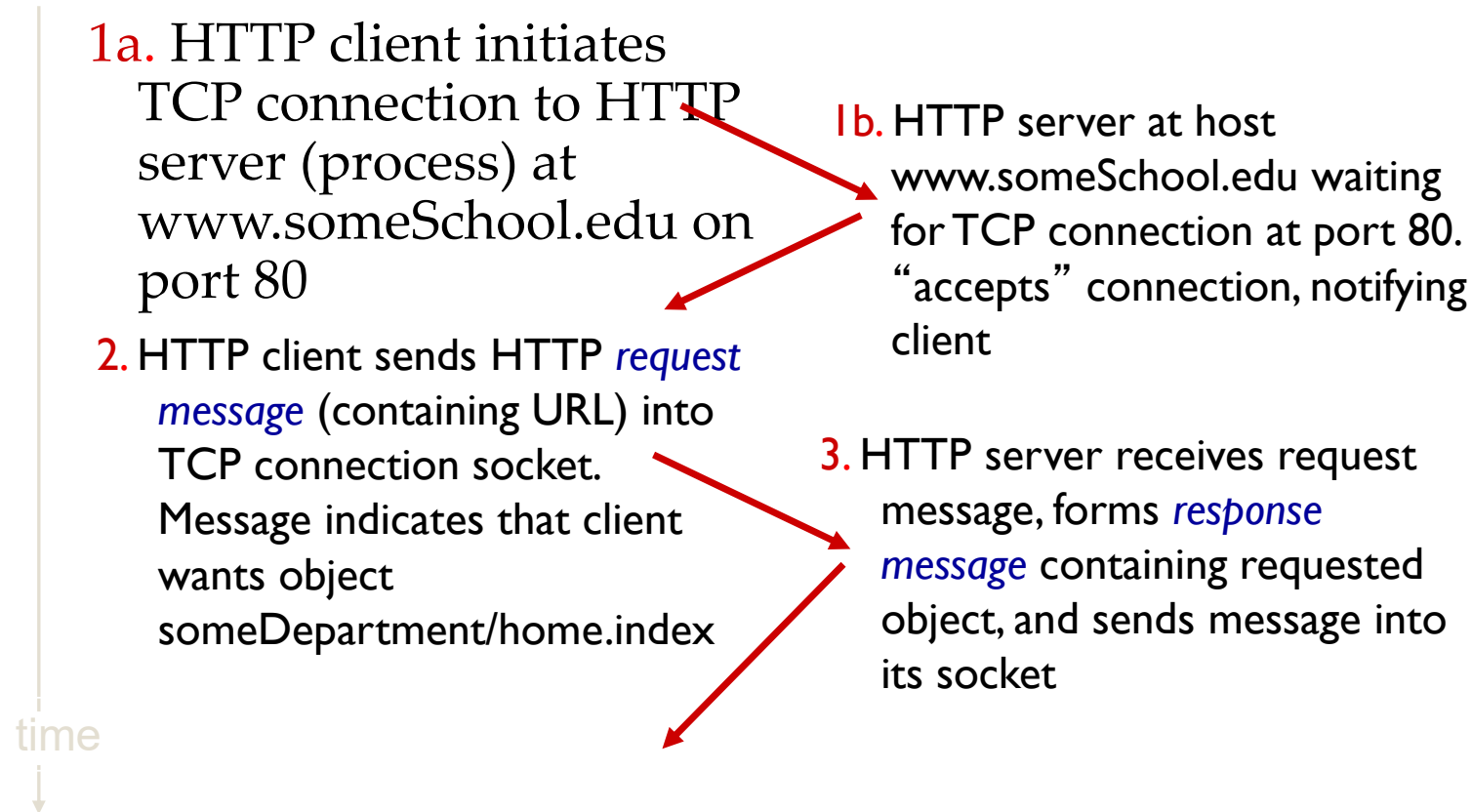
- multiple objects can be sent over single TCP connection between client, server

Non-persistent HTTP

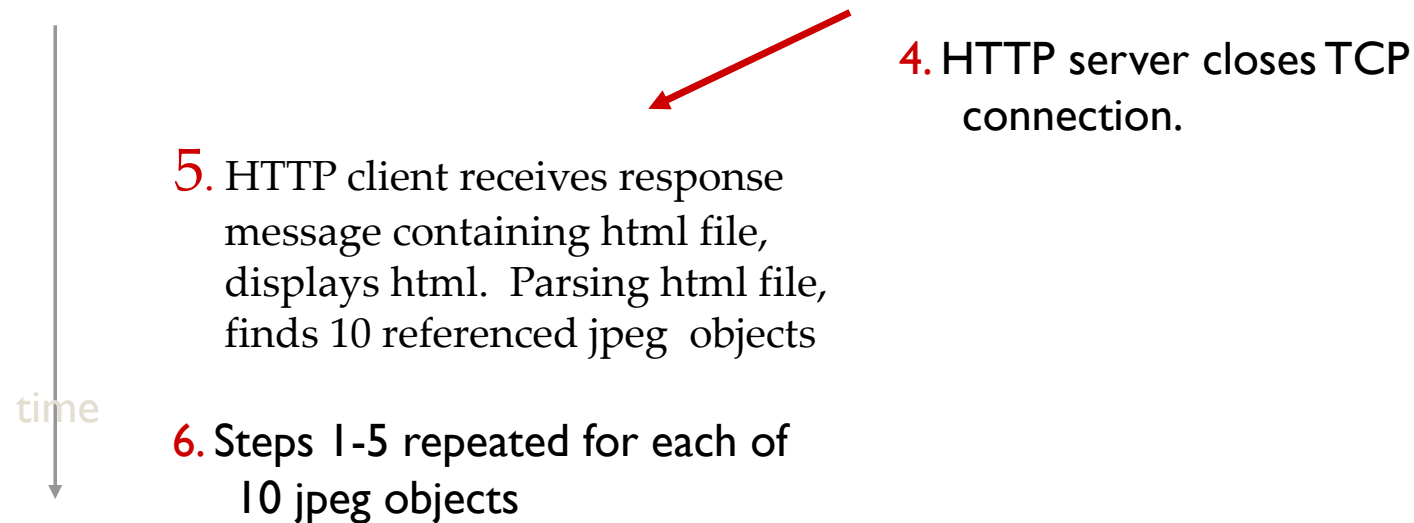
suppose user enters URL:

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

(contains text,
references to 10
jpeg images)



Non-persistent HTTP (cont.)

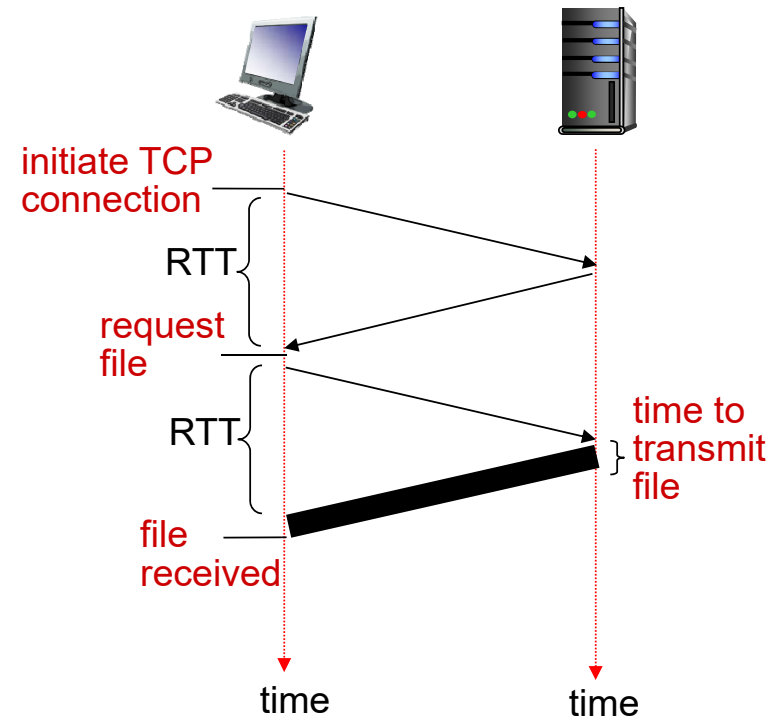


Non-persistent HTTP: response time

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



Persistent HTTP

non-persistent HTTP

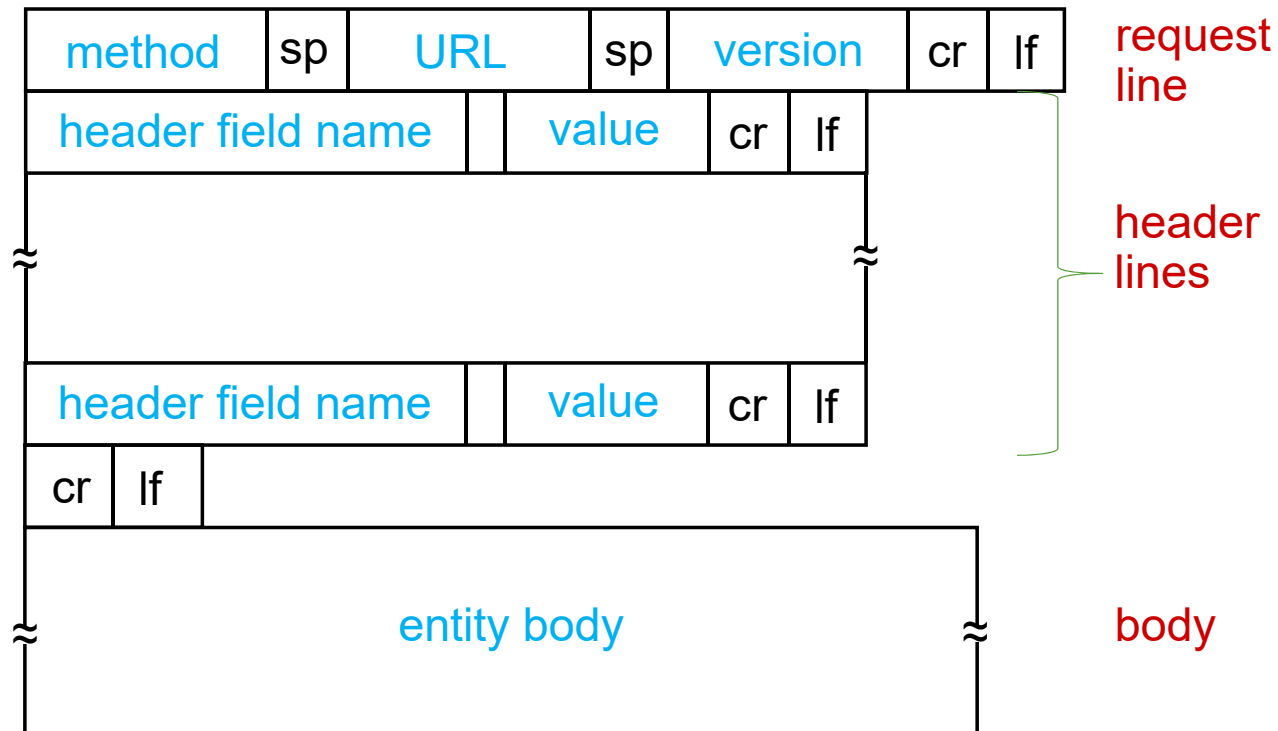
issues:

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

persistent HTTP:

- 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: general format



Trying out HTTP (client side) yourself

1. Telnet to your favorite Web server:

```
telnet www.kallasoft.dk 80
```

opens TCP connection to port 80
(default HTTP server port) at www.kallasoft.dk
anything typed in sent
to port 80 at www.kallasoft.dk

2. type in a GET HTTP request:

```
GET /index.php HTTP/1.1  
Host: www.kallasoft.dk
```

by typing this in (hit carriage
return twice), you send
this minimal (but complete)
GET request to HTTP server

3. look at response message sent by HTTP server!



Trying out HTTP (client side) yourself

4. Make sure that you capture the messages and look at them in wireshark

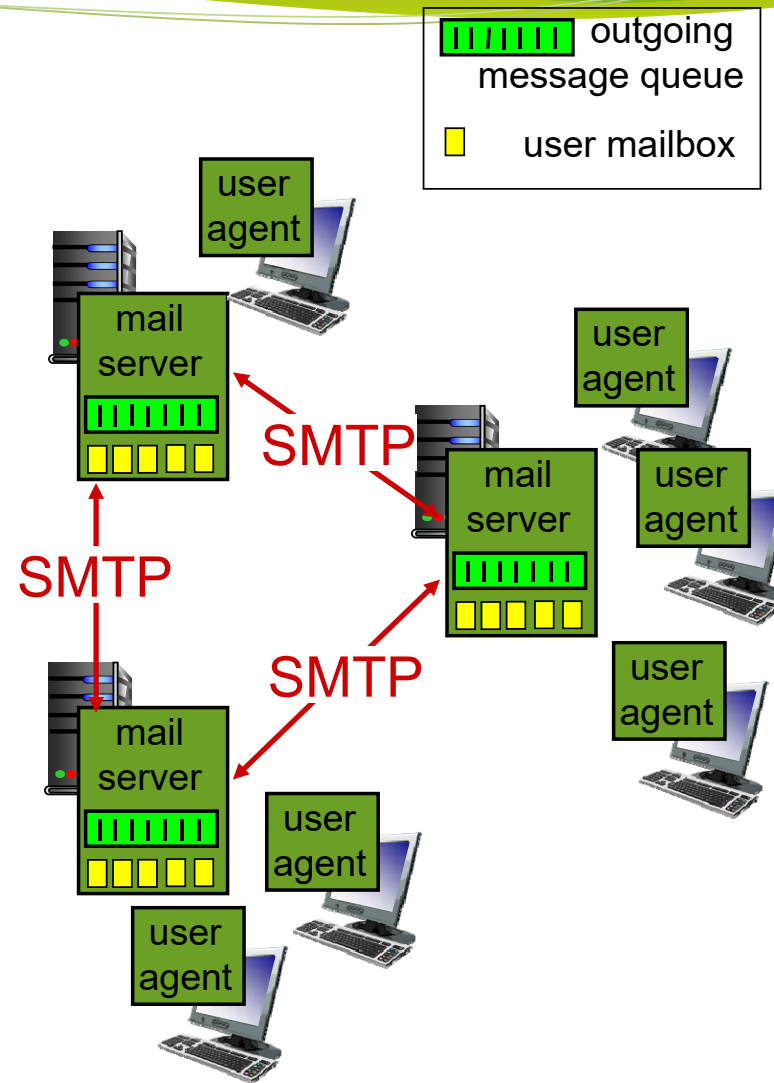
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

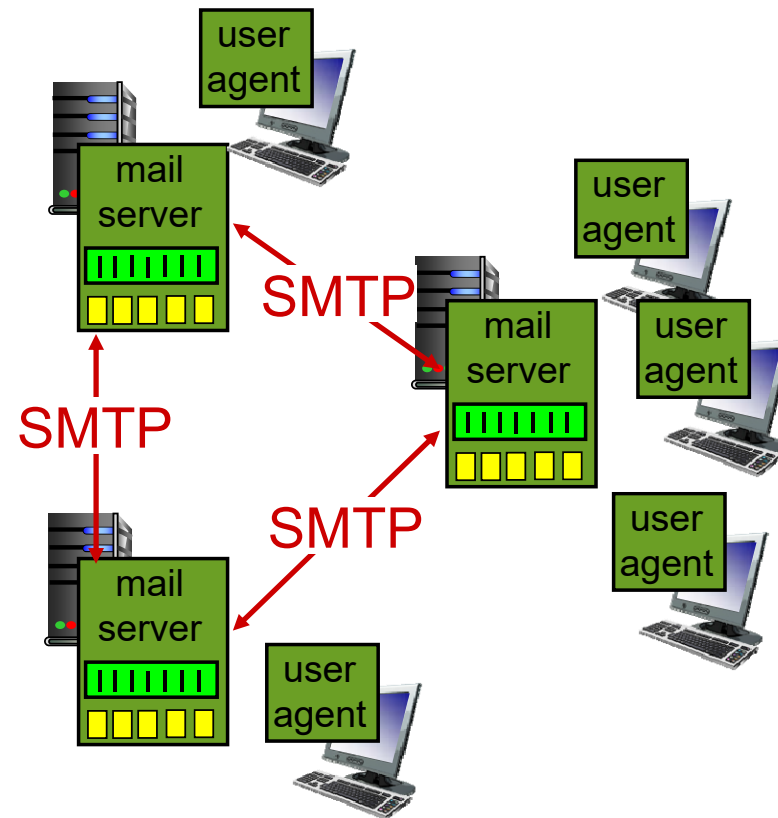
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
 - client: sending mail server
 - “server” : receiving mail server

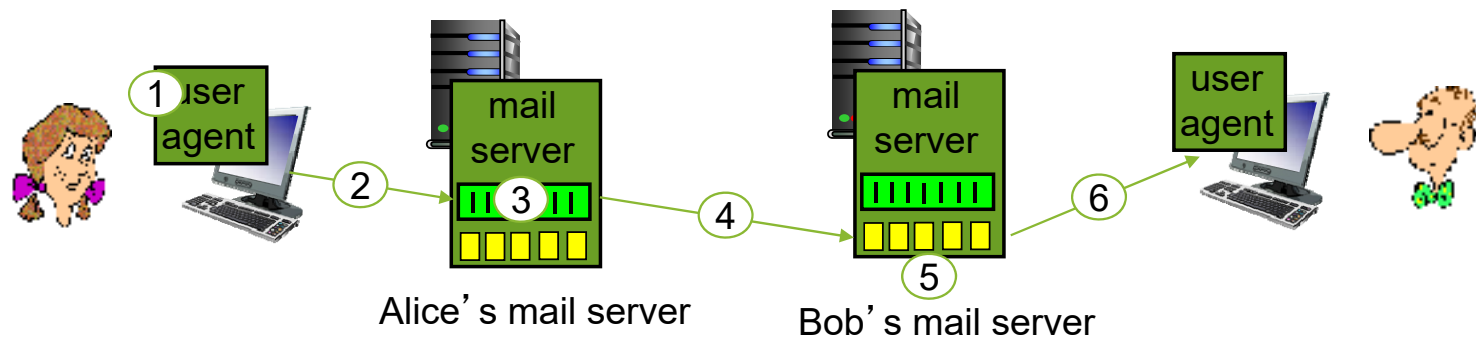


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
 - handshaking (greeting)
 - transfer of messages
 - closure
- command/response interaction (like HTTP, FTP)
 - **commands**: ASCII text
 - **response**: status code and phrase
- messages must be in 7-bit ASCII

Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message “to” bob@someschool.edu
- 2) Alice’s UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob’s mail server
- 4) SMTP client sends Alice’s message over the TCP connection
- 5) Bob’s mail server places the message in Bob’s mailbox
- 6) Bob invokes his user agent to read message



Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Try SMTP interaction for yourself:

- Go to `https://www.guerrillamail.com/`
- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client
(reader)

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF . CRLF to determine end of message

comparison with HTTP:

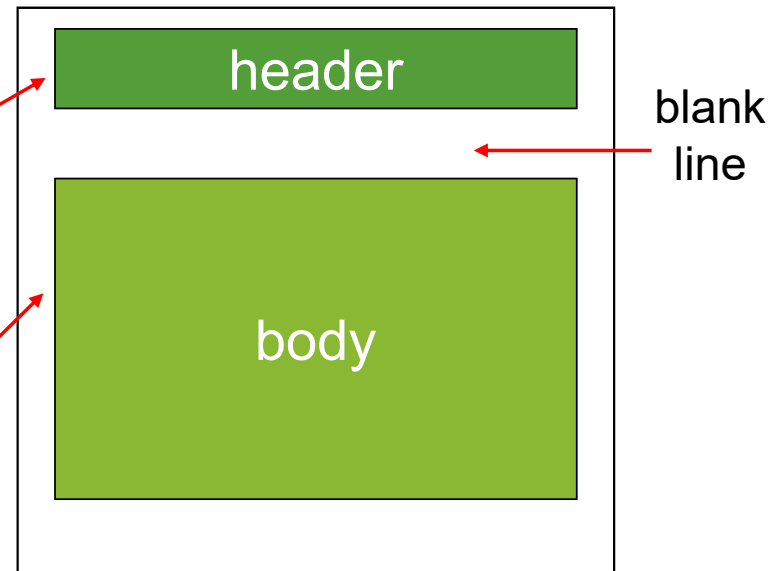
- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

Mail message format

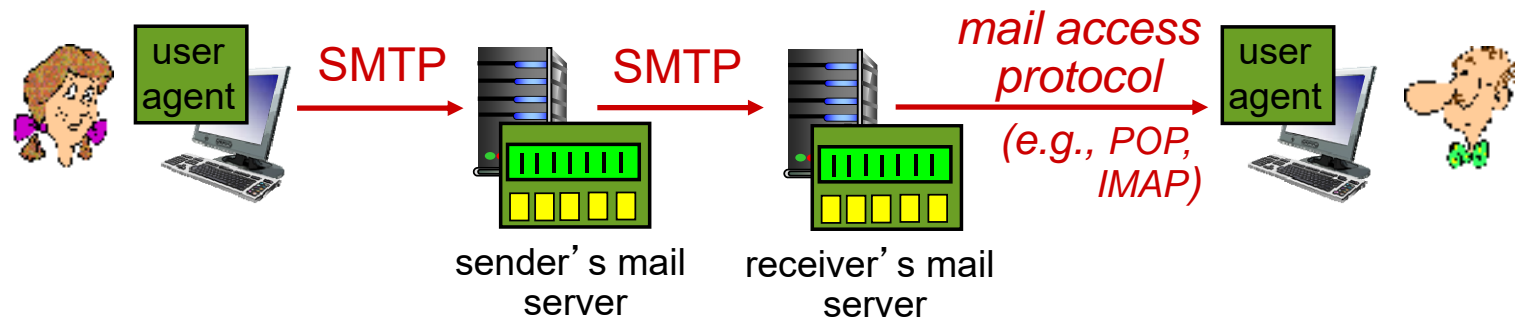
SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

- header lines, e.g.,
 - To:
 - From:
 - Subject:*different from* SMTP MAIL FROM, RCPT TO: commands!
- Body: the “message”
 - ASCII characters only



Mail access protocols



- **SMTP**: delivery/storage to receiver's server
- mail access protocol: retrieval from server
 - **POP**: Post Office Protocol [RFC 1939]: authorization, download
 - **IMAP**: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
 - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name” , e.g.,
www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database*
implemented in hierarchy of many *name servers*
- *application-layer protocol*:
hosts, name servers communicate to *resolve* names (address/name translation)
 - note: core Internet function, implemented as application-layer protocol
 - complexity at network’ s “edge”

DNS: services, structure

DNS services

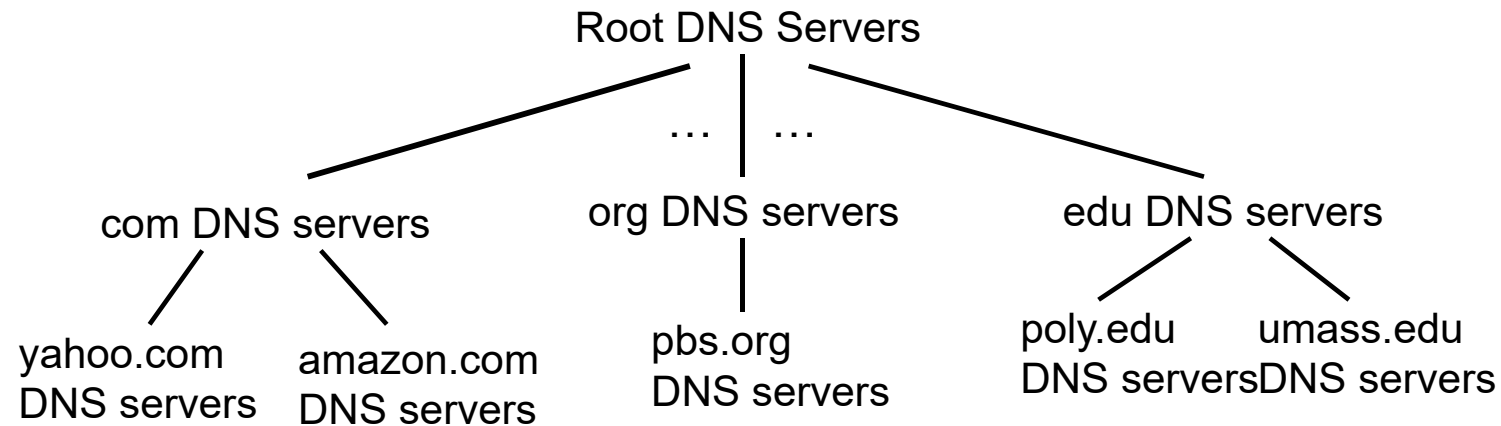
- hostname to IP address translation
- host aliasing
 - canonical, alias names
- mail server aliasing
- load distribution
 - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

DNS: a distributed, hierarchical database



client wants IP for www.amazon.com; 1st approx:

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
 - contacts authoritative name server if name mapping not known
 - gets mapping
 - returns mapping to local name server



*13 root name
“servers”
worldwide*

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization' s own DNS server(s), providing authoritative hostname to IP mappings for organization' s named hosts
- can be maintained by organization or service provider

Local DNS name server

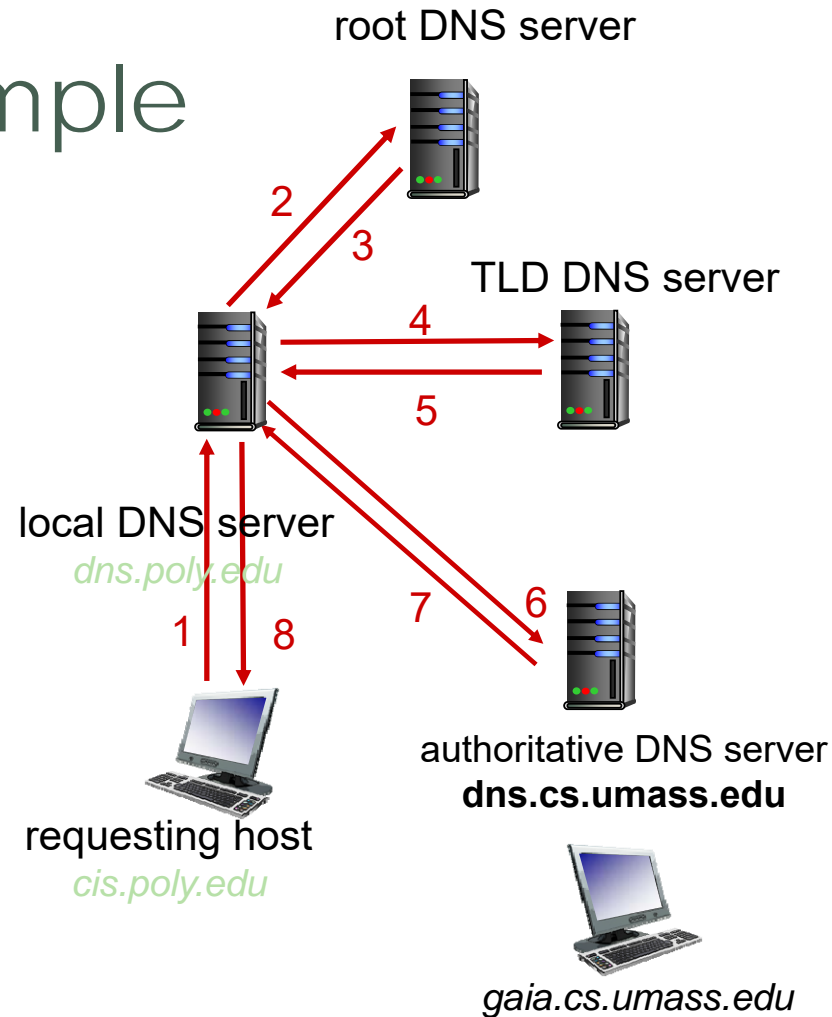
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
 - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
 - has local cache of recent name-to-address translation pairs (but may be out of date!)
 - acts as proxy, forwards query into hierarchy

DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

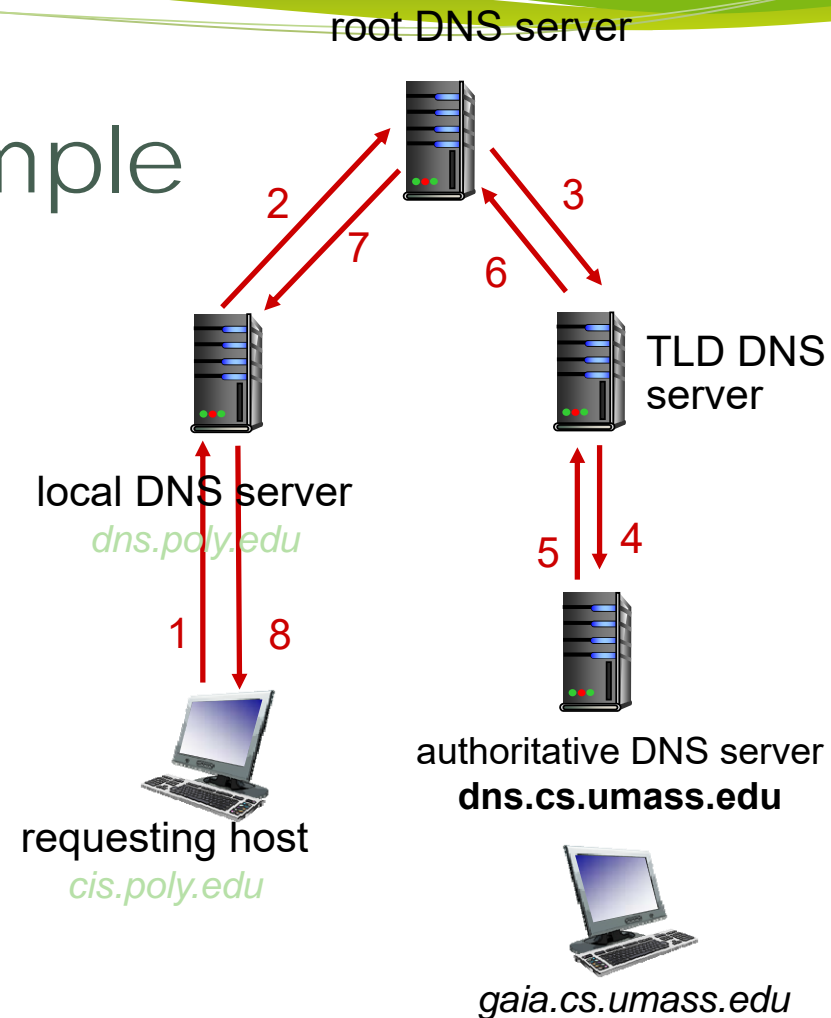
- ❖ contacted server replies with name of server to contact
- ❖ “I don’t know this name, but ask this server”



DNS name resolution example

recursive query:

- ❖ puts burden of name resolution on contacted name server
- ❖ heavy load at upper levels of hierarchy?



DNS records

DNS: distributed db storing resource records (RR)

RR format: (name, value, type, ttl)

type=A

- **name** is hostname
- **value** is IP address

type=NS

- **name** is domain (e.g., foo.com)
- **value** is hostname of authoritative name server for this domain

type=CNAME

- **name** is alias name for some “canonical” (the real) name
- **www.ibm.com** is really **servereast.backup2.ibm.com**
- **value** is canonical name

type=MX

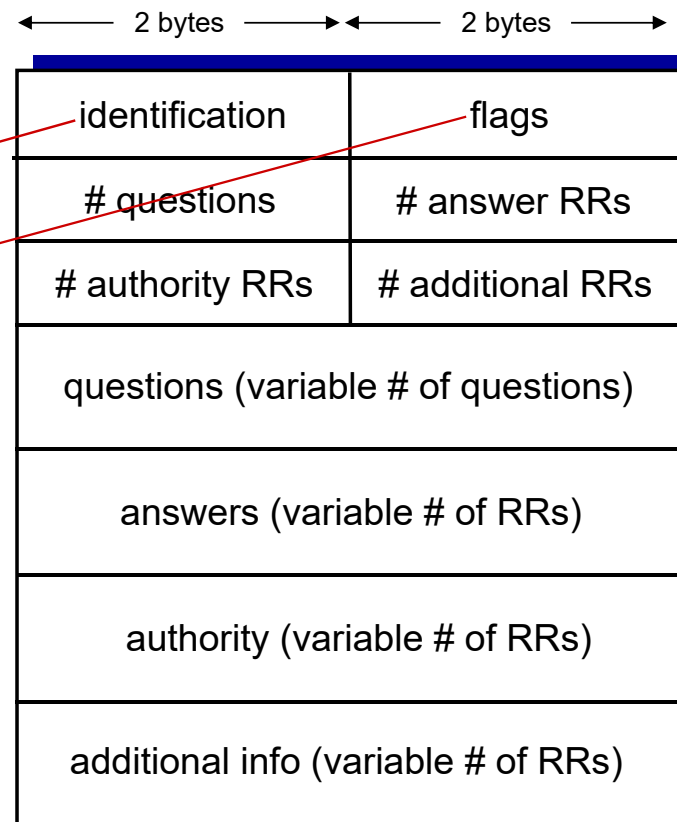
- **value** is name of mailserver associated with **name**

DNS protocol, messages

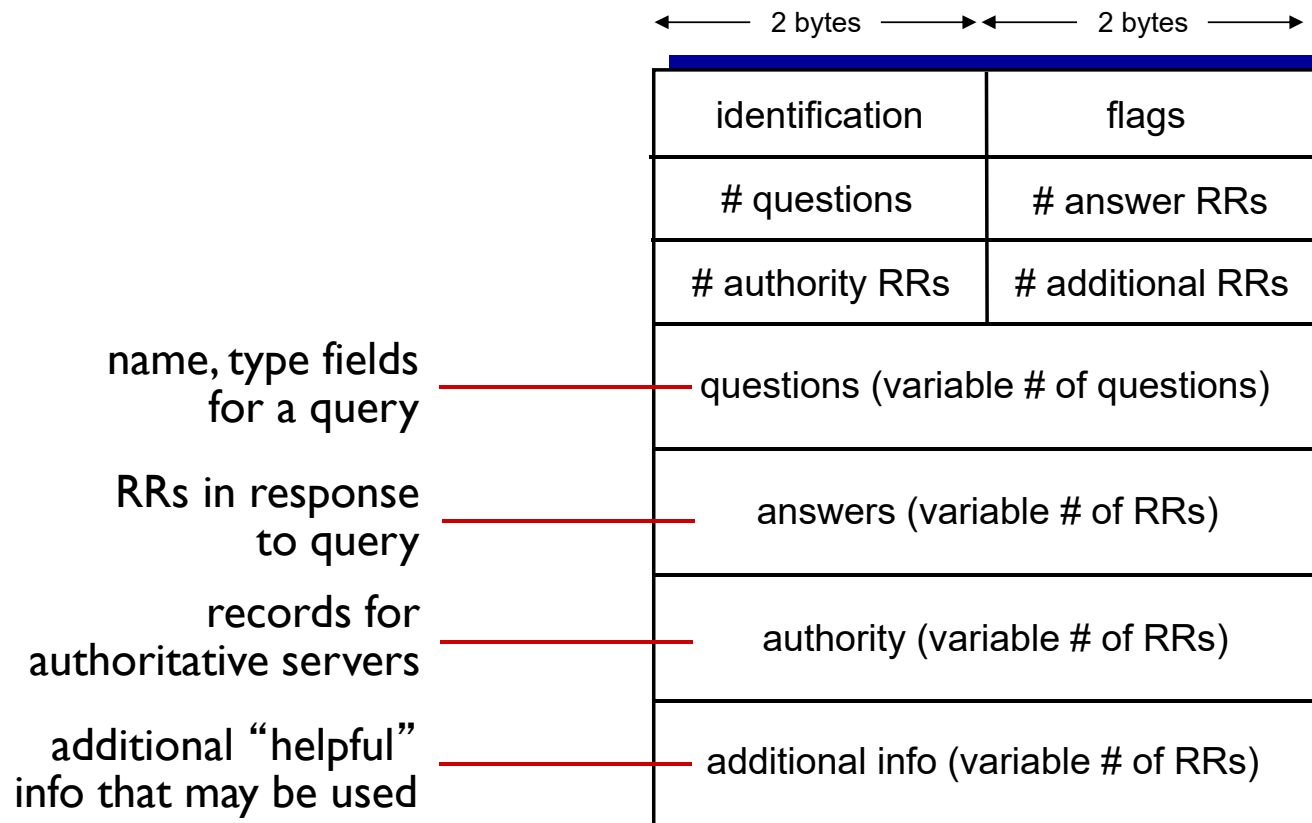
- *query* and *reply* messages, both with same *message format*

msg header

- ❖ **identification**: 16 bit # for query, reply to query uses same #
- ❖ **flags**:
 - query or reply
 - recursion desired
 - recursion available
 - reply is authoritative



DNS protocol, messages

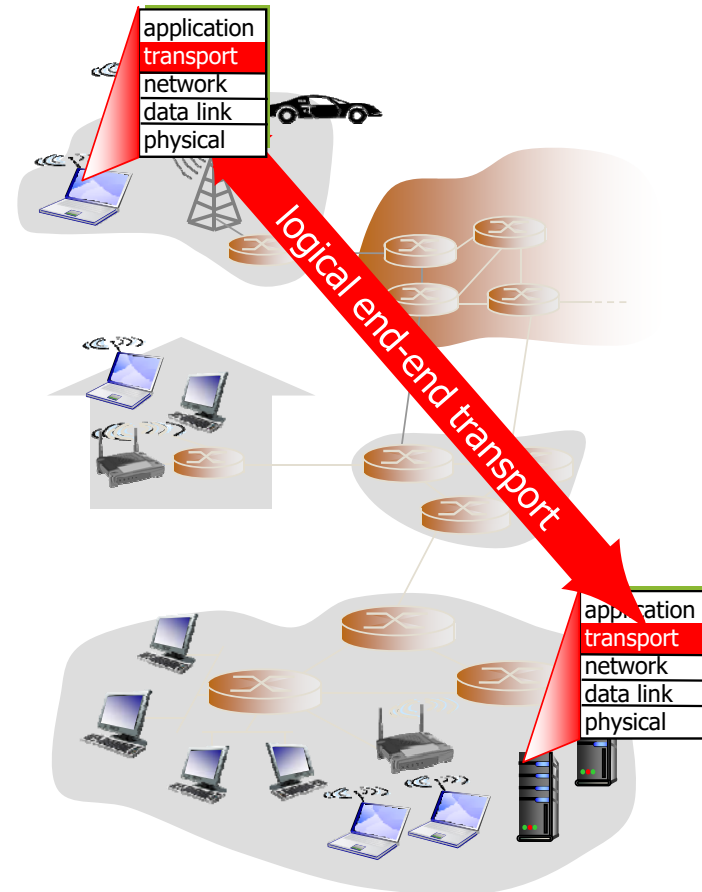


Inserting records into DNS

- example: new startup “Network Utopia”
- register name networkutopia.com at *DNS registrar* (e.g., Network Solutions)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts two RRs into .com TLD server:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server type A record for www.networkutopia.com; type MX record for networkutopia.com

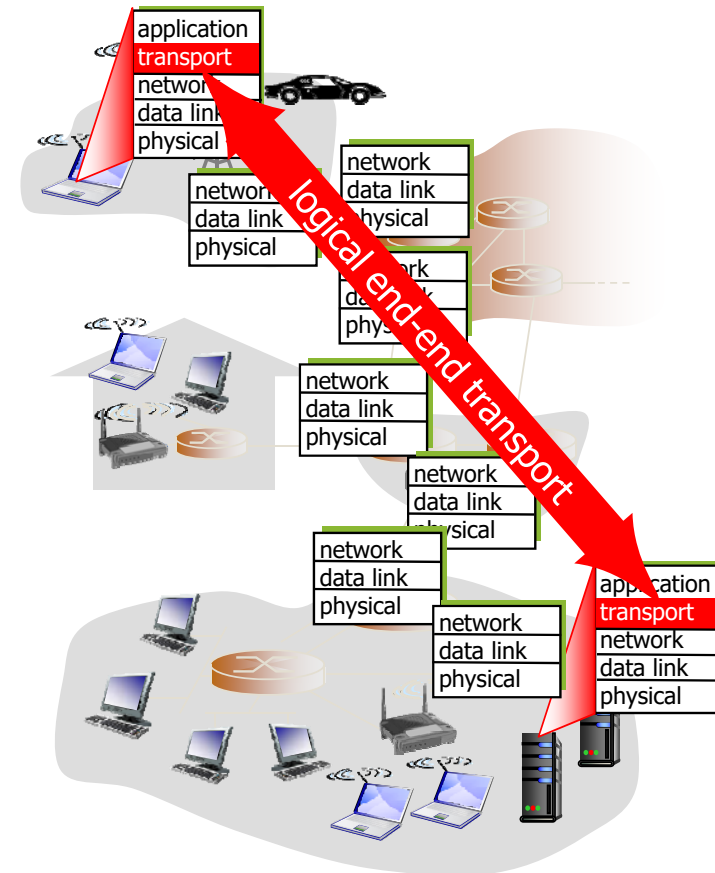
Transport services and protocols

- ❖ provide *logical communication* between app processes running on different hosts
- ❖ transport protocols run in end systems
 - send side: breaks app messages into *segments*, passes to network layer
 - rcv side: reassembles segments into messages, passes to app layer
- ❖ more than one transport protocol available to apps
 - Internet: TCP and UDP



Internet transport-layer protocols

- reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup
- unreliable, unordered delivery: UDP
 - no-frills extension of “best-effort” IP
- services not available:
 - delay guarantees
 - bandwidth guarantees



UDP: User Datagram Protocol [RFC 768]

- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
 - lost
 - delivered out-of-order to app
- *connectionless*:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

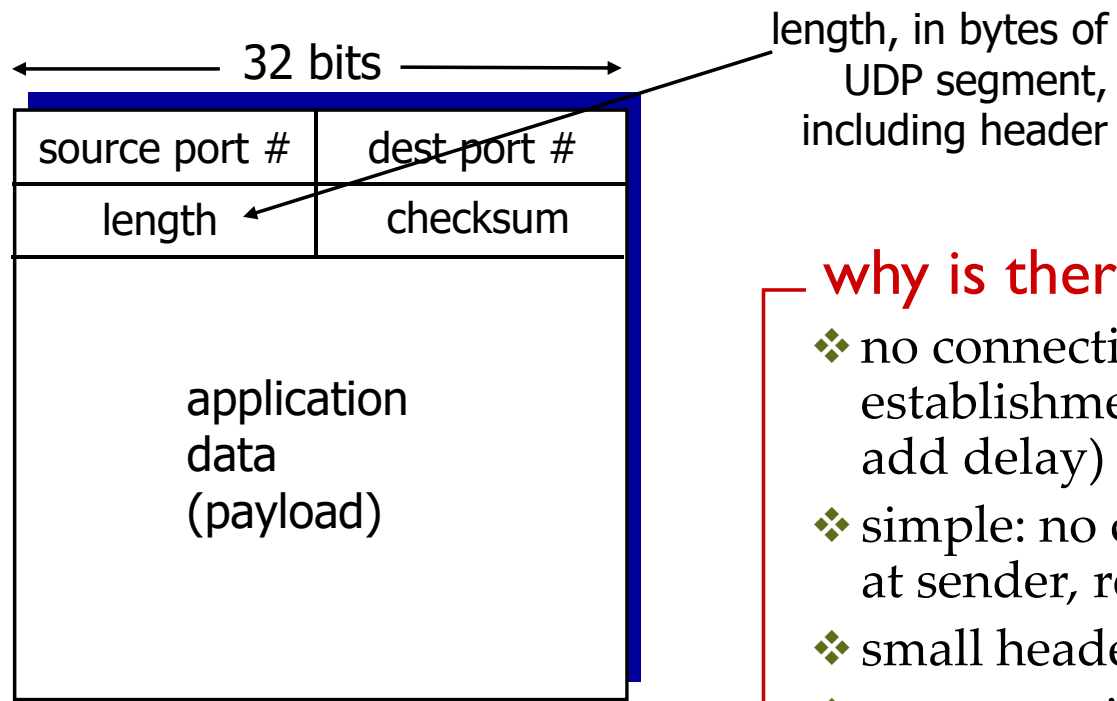
❖ UDP use:

- streaming multimedia apps (loss tolerant, rate sensitive)
- DNS
- SNMP

❖ reliable transfer over UDP:

- add reliability at application layer
- application-specific error recovery!

UDP: segment header



UDP segment format

why is there a UDP?

- ❖ no connection establishment (which can add delay)
- ❖ simple: no connection state at sender, receiver
- ❖ small header size
- ❖ no congestion control: UDP can blast away as fast as desired

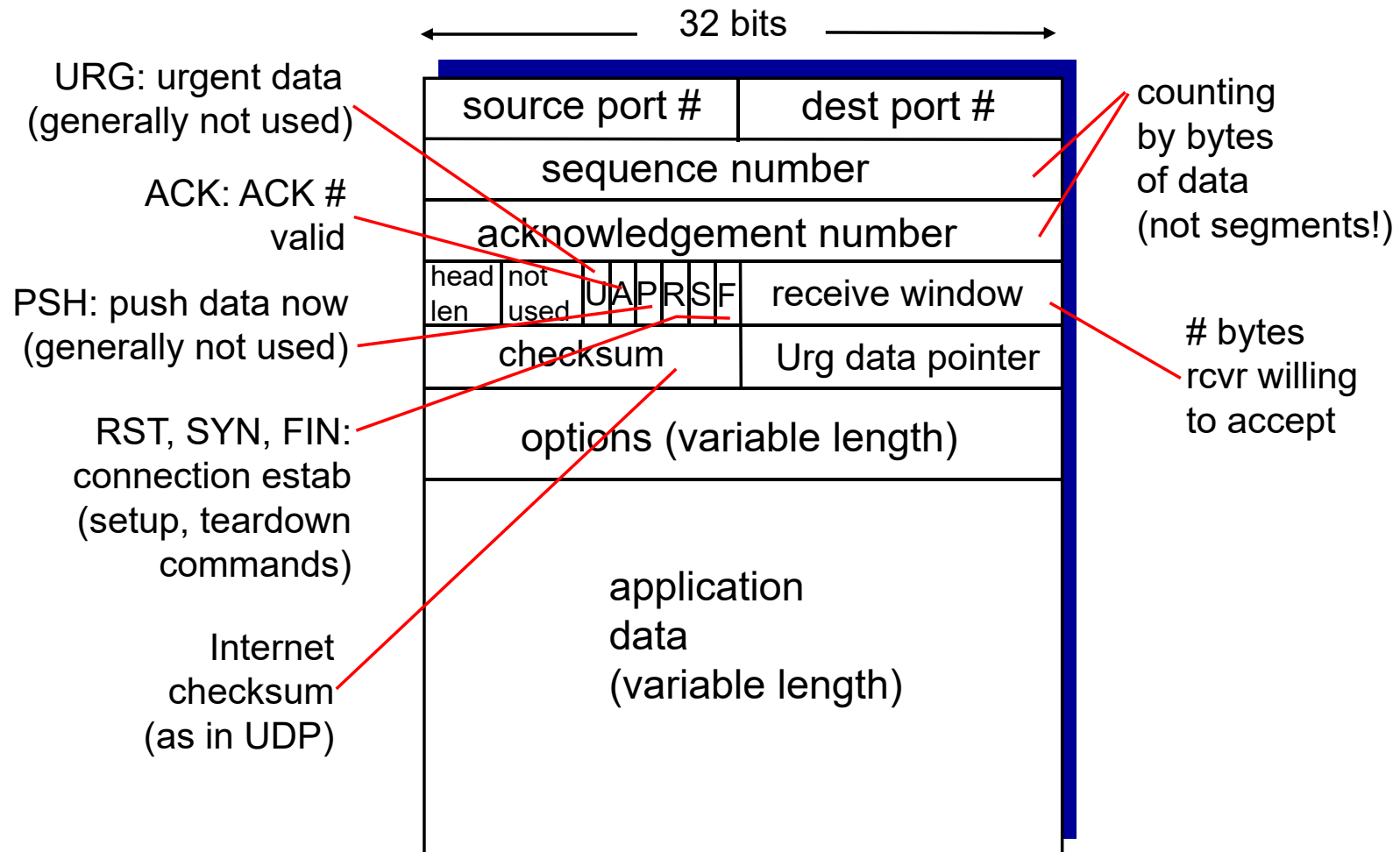
TCP: Overview

RFCs: 793, 1122, 1323, 2018,

2581

- **point-to-point:**
 - one sender, one receiver
- **reliable, in-order *byte stream*:**
 - no “message boundaries”
- **pipelined:**
 - TCP congestion and flow control set window size
- ❖ **full duplex data:**
 - bi-directional data flow in same connection
 - MSS: maximum segment size
- ❖ **connection-oriented:**
 - handshaking (exchange of control msgs) initializes sender, receiver state before data exchange
- ❖ **flow controlled:**
 - sender will not overwhelm receiver

TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

- byte stream “number” of first byte in segment’s data

acknowledgements:

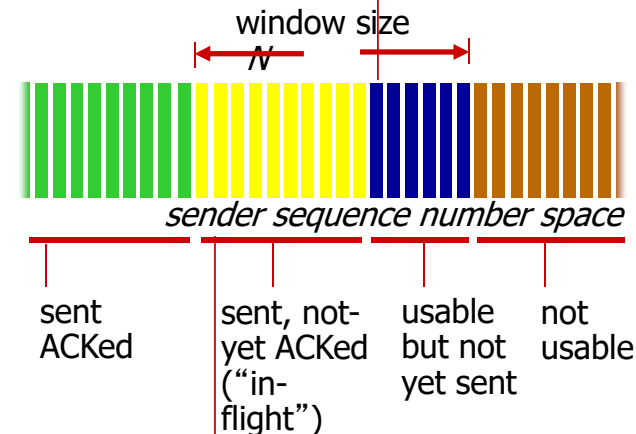
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- **A:** TCP spec doesn’t say, - up to implementor

outgoing segment from sender

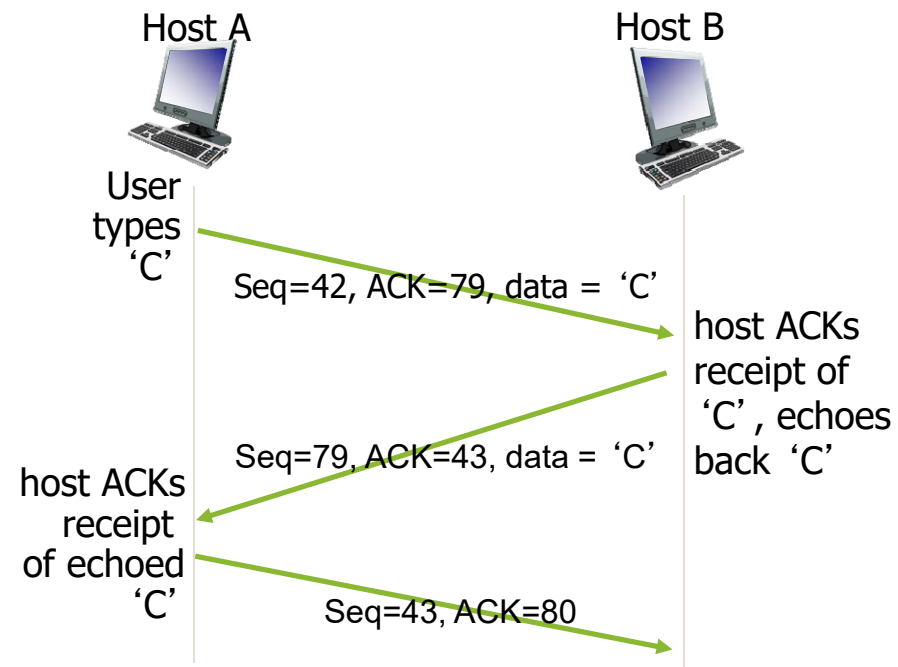
source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer



incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer

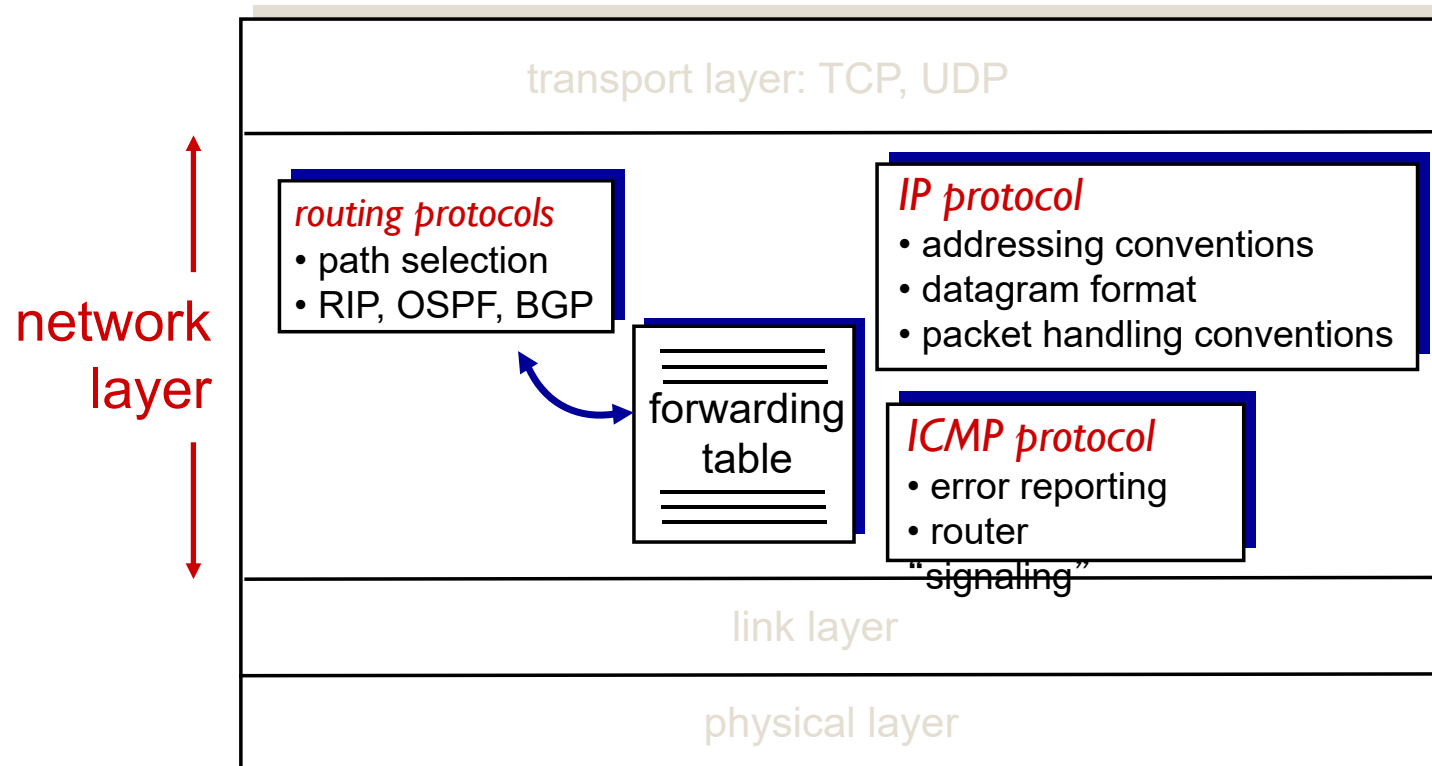
TCP seq. numbers, ACKs



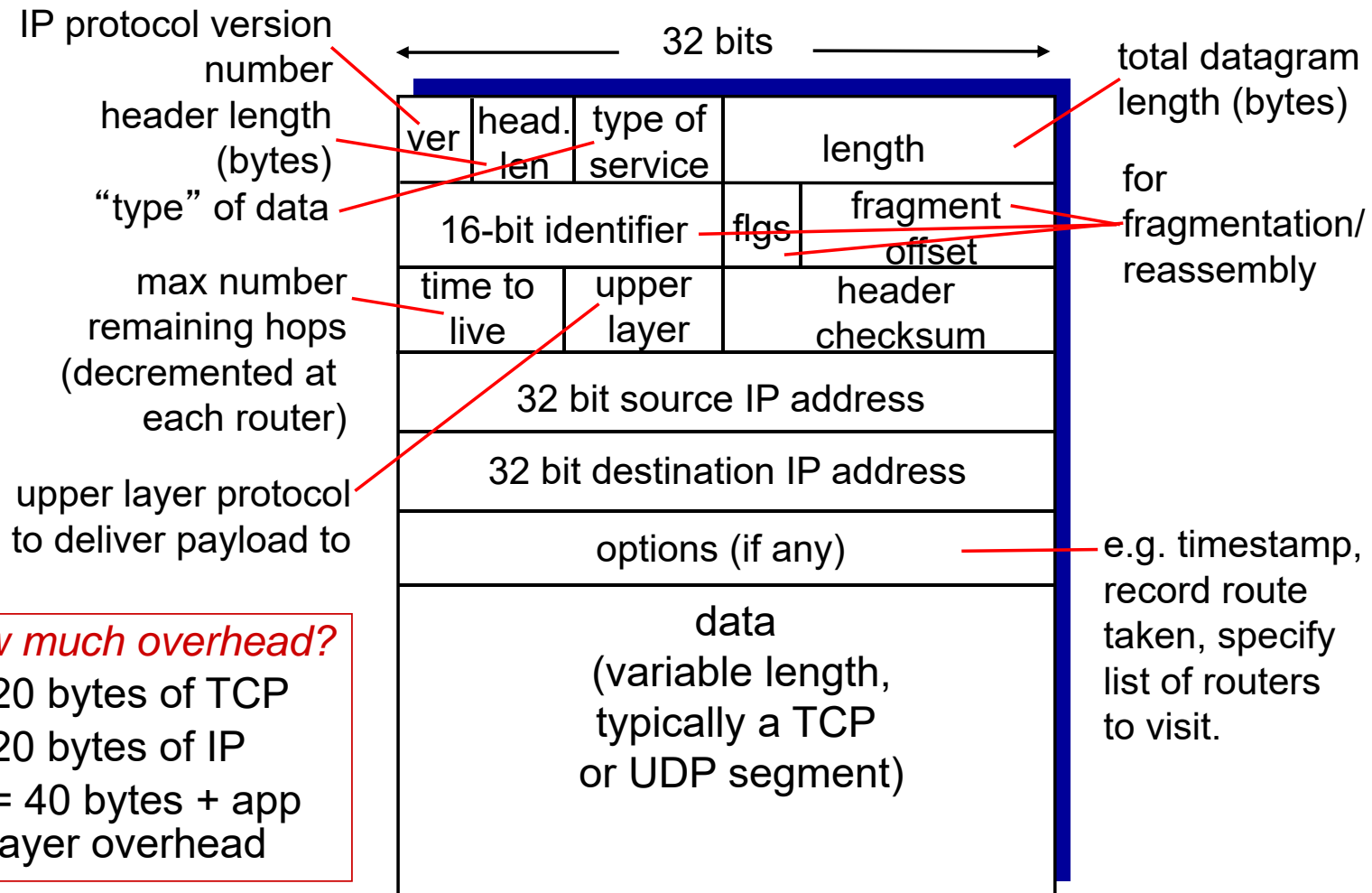
simple telnet scenario

The Internet network layer

host, router network layer functions:

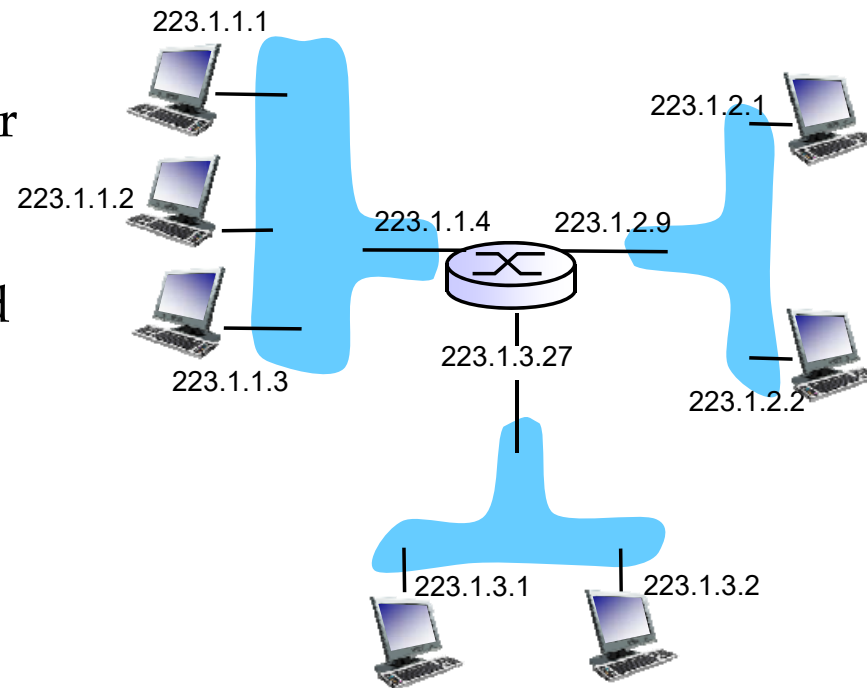


IP datagram format



IP addressing: introduction

- *IP address*: 32-bit identifier for host, router *interface*
- *interface*: connection between host/router and physical link
 - routers typically have multiple interfaces
 - host typically has one active interface (e.g., wired Ethernet, wireless 802.11)
- *one IP address associated with each interface*



223.1.1.1 = $\underbrace{11011111}_{223} \underbrace{00000001}_1 \underbrace{00000001}_1 \underbrace{00000001}_1$

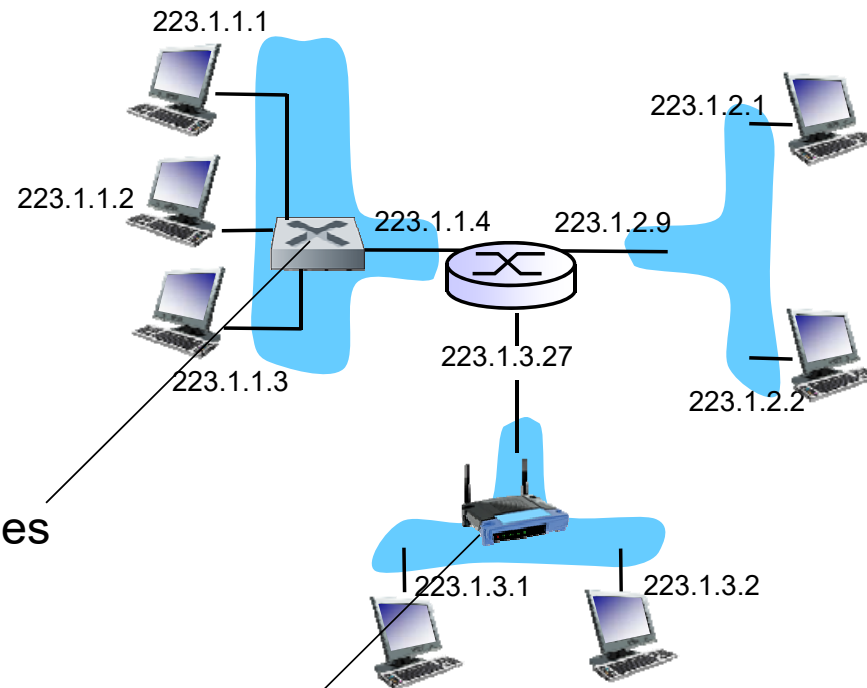
IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapter 5, 6.

A: wired Ethernet interfaces connected by Ethernet switches

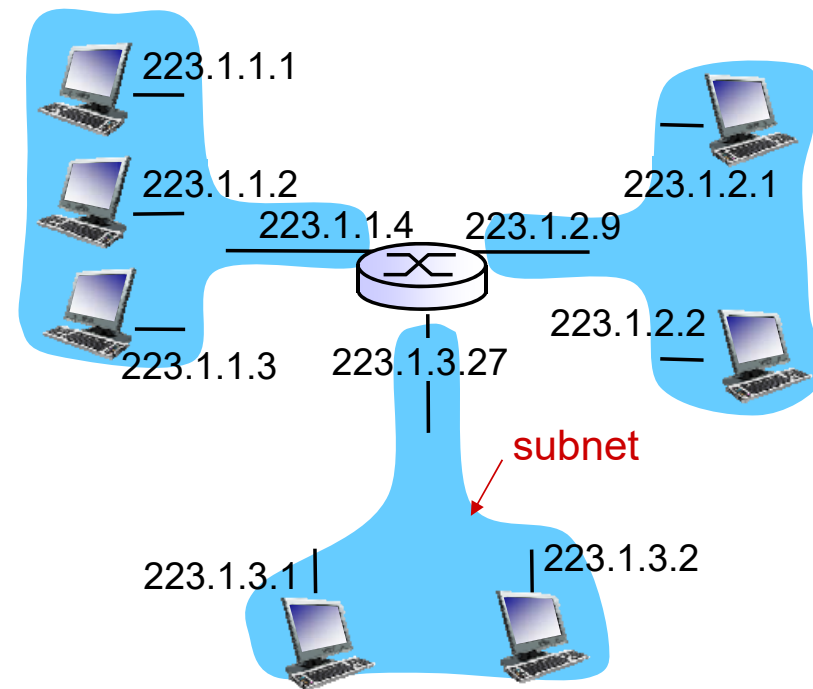
For now: don't need to worry about how one interface is connected to another (with no intervening router)



A: wireless WiFi interfaces connected by WiFi base station

Subnets

- IP address:
 - subnet part - high order bits
 - host part - low order bits
- *what ' s a subnet ?*
 - device interfaces with same subnet part of IP address
 - can physically reach each other *without intervening router*



network consisting of 3 subnets

ICMP: internet control message protocol

- used by hosts & routers to communicate network-level information
 - error reporting: unreachable host, network, port, protocol
 - echo request/reply (used by ping)
- network-layer “above” IP:
 - ICMP msgs carried in IP datagrams
- **ICMP message**: type, code plus first 8 bytes of IP datagram causing error

<u>Type</u>	<u>Code</u>	<u>description</u>
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

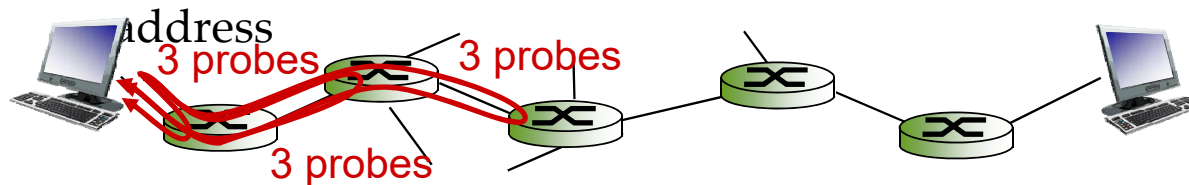
Traceroute and ICMP

- ❖ source sends series of UDP segments to dest
 - first set has TTL =1
 - second set has TTL=2, etc.
 - unlikely port number
- ❖ when n th set of datagrams arrives to n th router:
 - router discards datagrams
 - and sends source ICMP messages (type 11, code 0)
 - ICMP messages includes name of router & IP

- ❖ when ICMP messages arrives, source records RTTs

stopping criteria:

- ❖ UDP segment eventually arrives at destination host
- ❖ destination returns ICMP “port unreachable” message (type 3, code 3)
- ❖ source stops



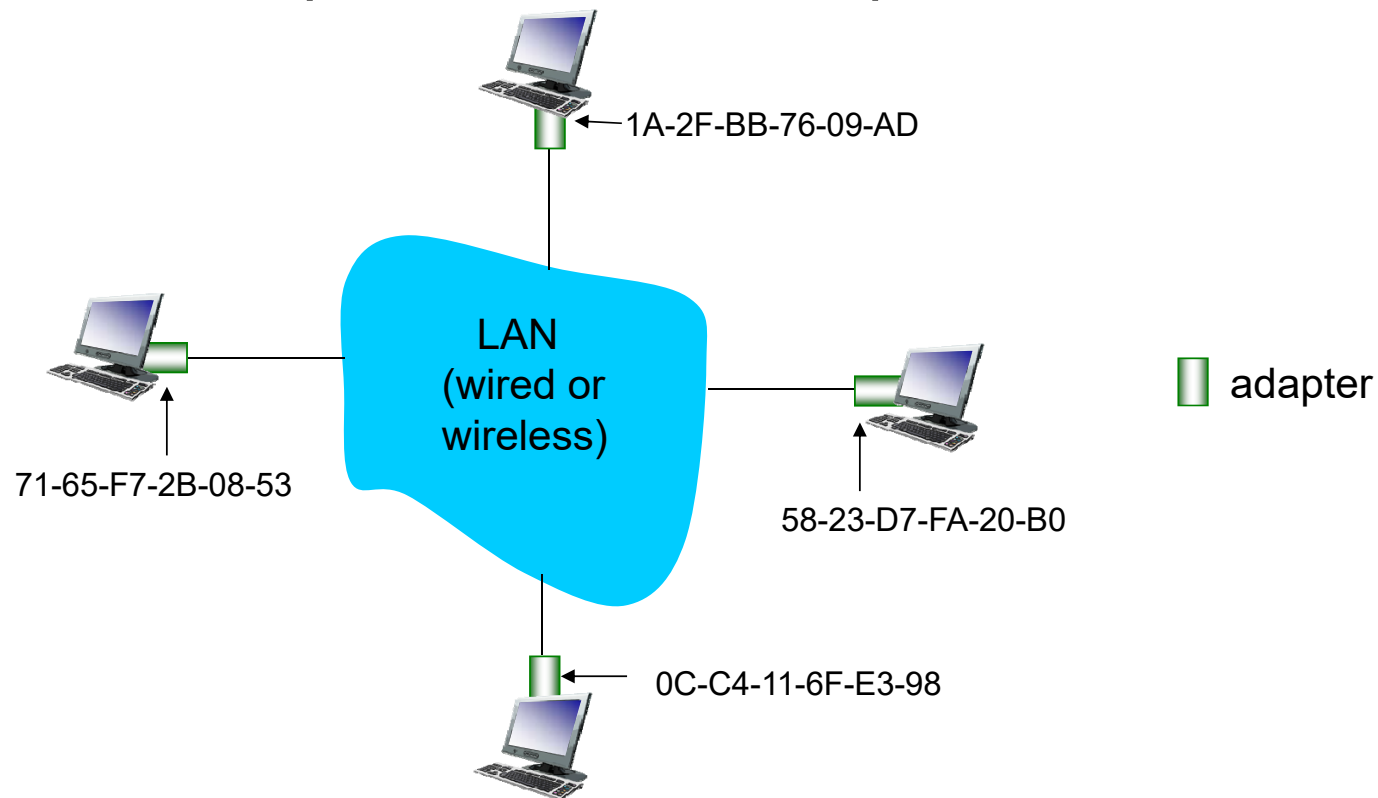
MAC addresses and ARP

- 32-bit IP address:
 - *network-layer* address for interface
 - used for layer 3 (network layer) forwarding
- MAC (or LAN or physical or Ethernet) address:
 - function: *used 'locally' to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
 - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
 - e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation
(each “number” represents 4 bits)

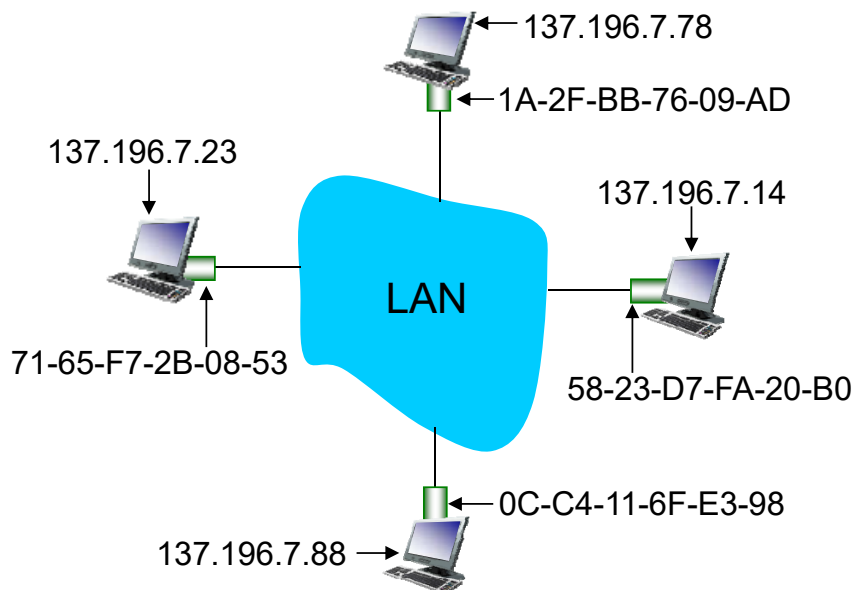
LAN addresses and ARP

each adapter on LAN has unique **LAN** address



ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?



ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:

< IP address; MAC address; TTL >

- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP protocol: same LAN

- A wants to send datagram to B
 - B' s MAC address not in A' s ARP table.
- A **broadcasts** ARP query packet, containing B's IP address
 - dest MAC address = FF-FF-FF-FF-FF-FF
 - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
 - frame sent to A' s MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
 - soft state: information that times out (goes away) unless refreshed
- ARP is “plug-and-play” :
 - nodes create their ARP tables *without intervention from net administrator*