

# Networking 101

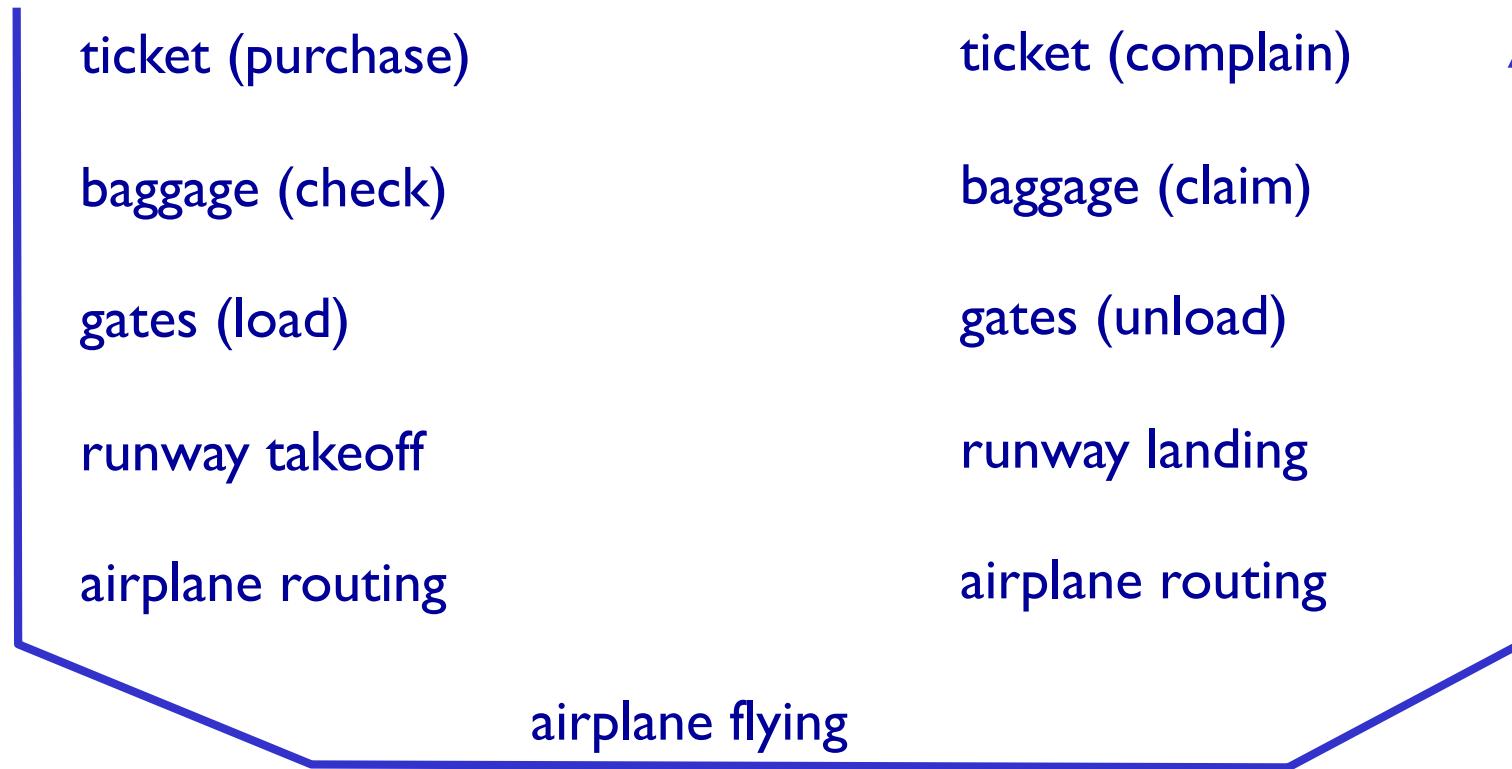
# Today: EECS 489 in one day

- Today: A whirlwind tour through how networking and the Internet work
- Thursday: we'll start learning how to attack it!
- If you want to *really* learn this, take EECS 489
  - Today's lecture just skims the surface

# Internet

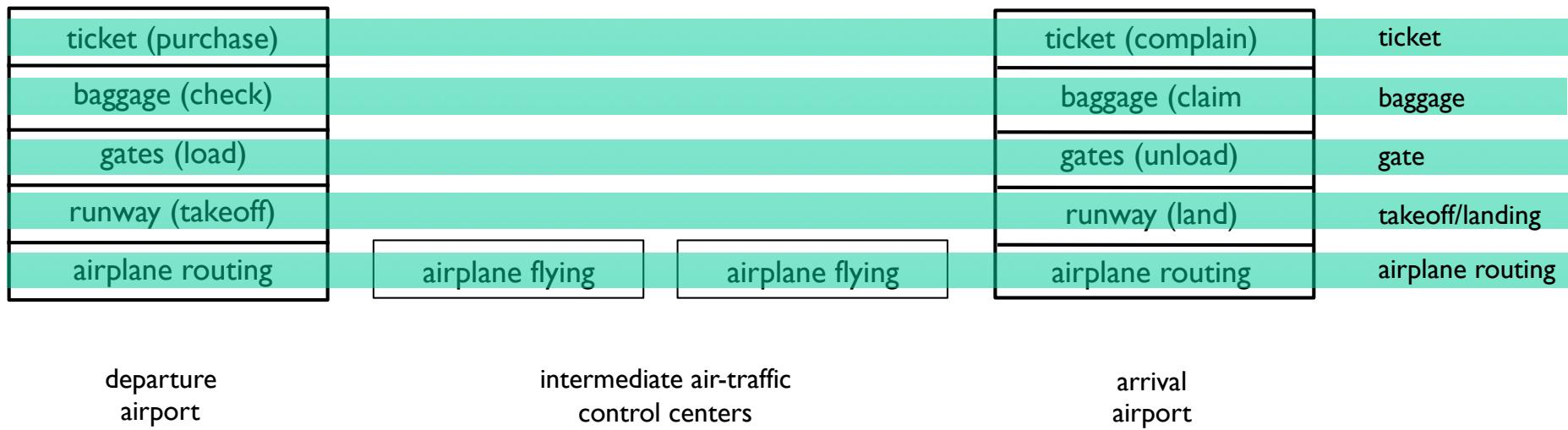
- Mostly about naming, addressing, and routing

# Organization of air travel



- ❖ a series of steps

# Layering of airline functionality



*layers:* each layer implements a service

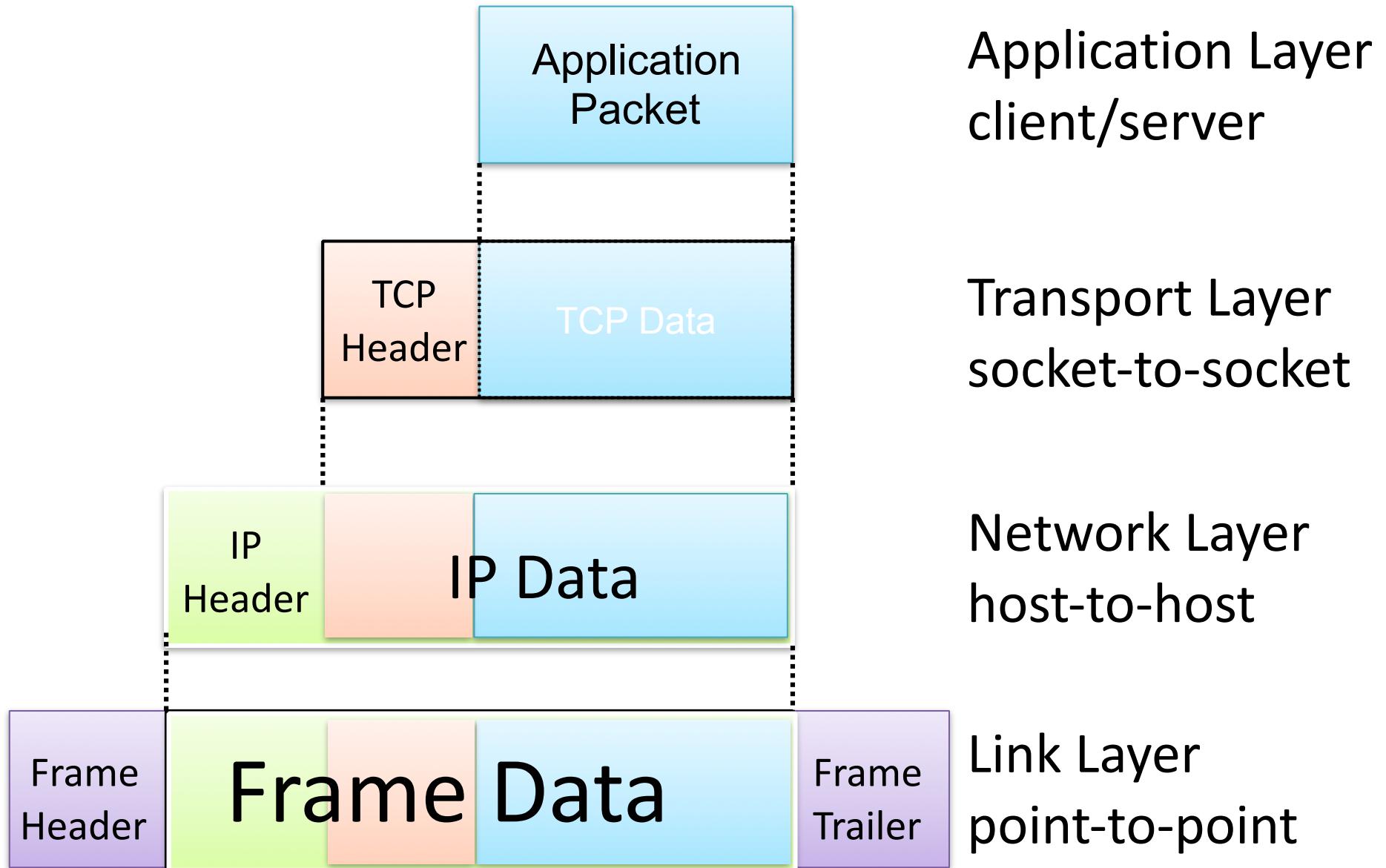
- via its own internal-layer actions
- relying on services provided by layer below

# Why layering?

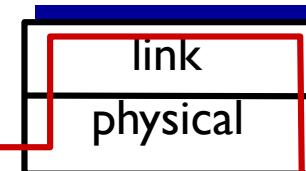
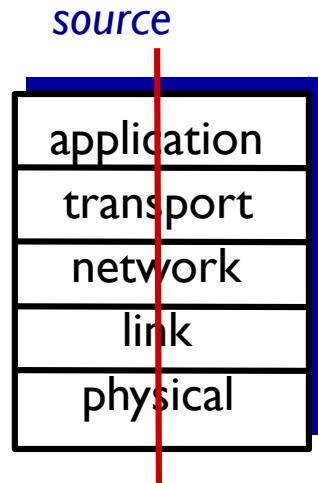
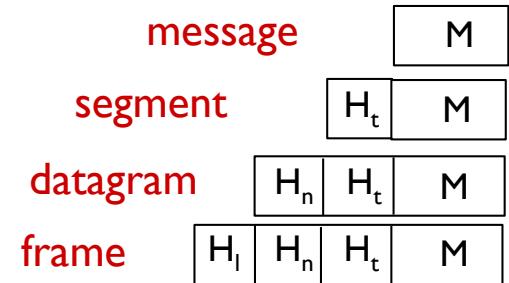
dealing with complex systems:

- ❖ explicit structure allows identification, relationship of complex system's pieces
  - layered *reference model* for discussion
- ❖ modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in gate procedure doesn't affect rest of system
- ❖ layering considered harmful?

# Internet Packet Encapsulation

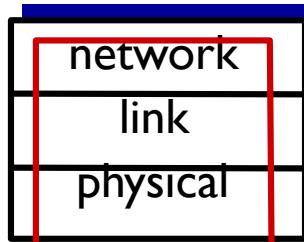
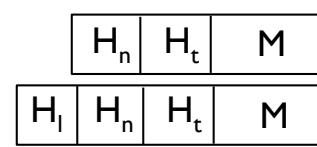
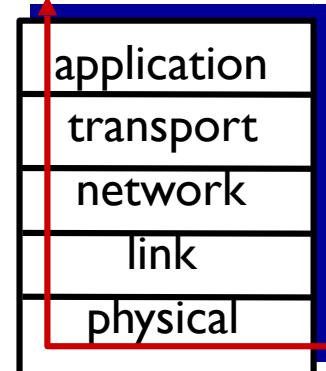


# Encapsulation



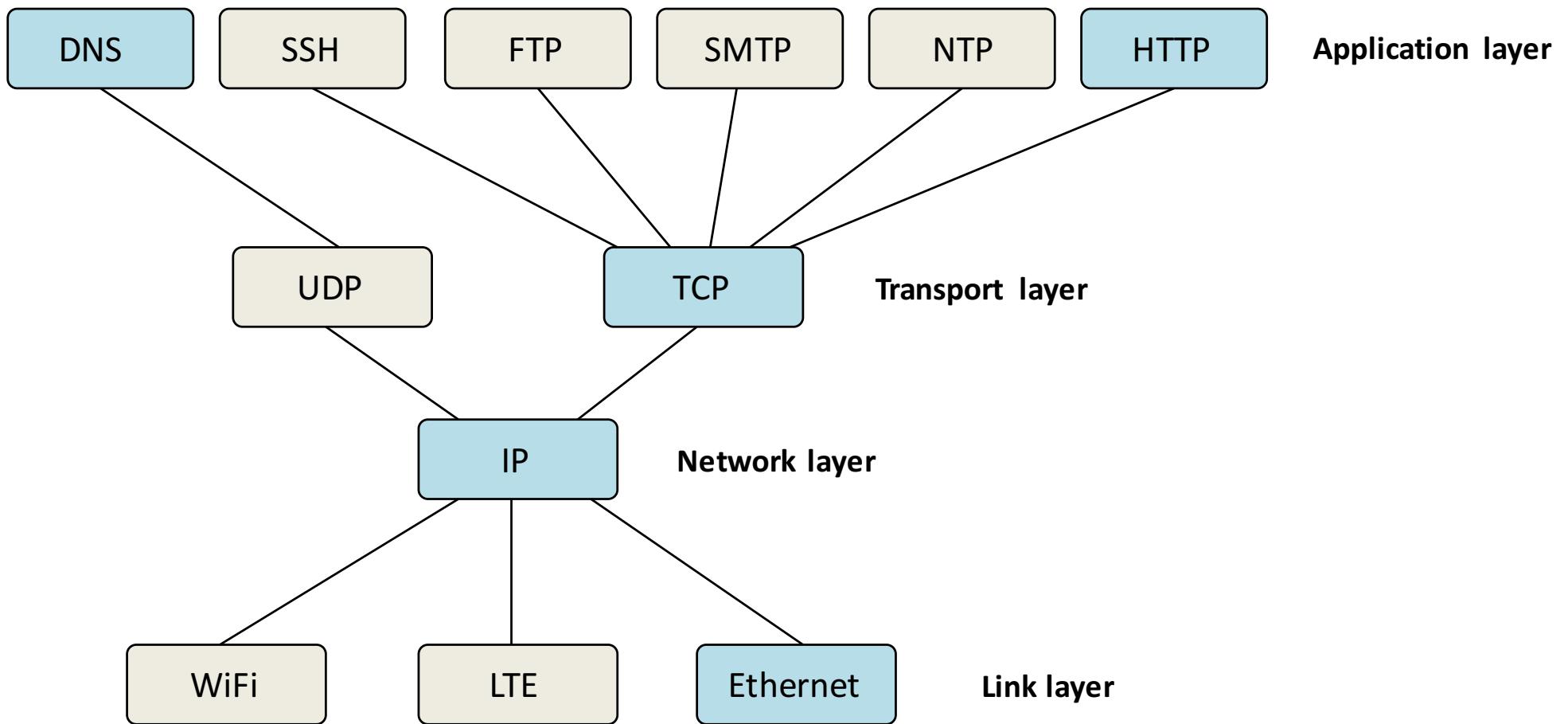
switch

destination



router

# Layering of Protocols



# What a 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

**Types of Protocols**

**open protocols:**

defined in Internet RFCs  
allow for interoperability  
e.g., HTTP, SMTP

**proprietary protocols:**

e.g., Skype

# HTTP request message

- ❖ two types of HTTP messages: *request, response*
- ❖ **HTTP request message:**
  - ASCII (human-readable format)

request line  
(GET, POST,  
HEAD commands)

header  
lines

carriage return,  
line feed at start  
of line indicates  
end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return character  
line-feed character

# HTTP response message

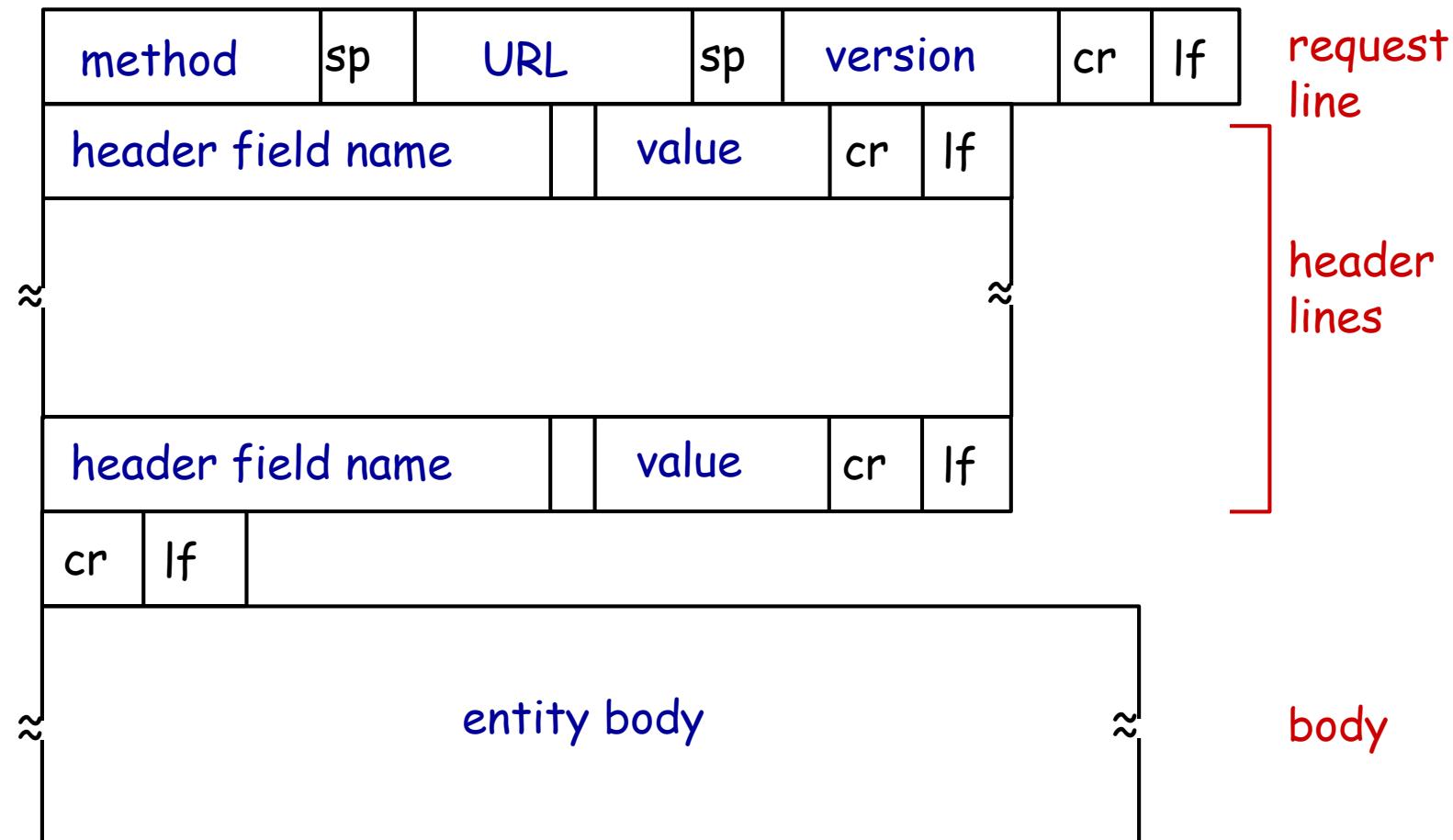
status line  
(protocol  
status code  
status phrase)

header  
lines

data, e.g.,  
requested  
HTML file

```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS) \r\nLast-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-1\r\n\r\ndata data data data data ...
```

# HTTP request message: general format



# 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**

# DNS: domain name system

*people:* many identifiers:

- SSN, name, passport #

*Internet hosts, routers:*

- IP address (numeric) - used for addressing datagrams
- “name” (symbolic), e.g., [www.yahoo.com](http://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 and 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

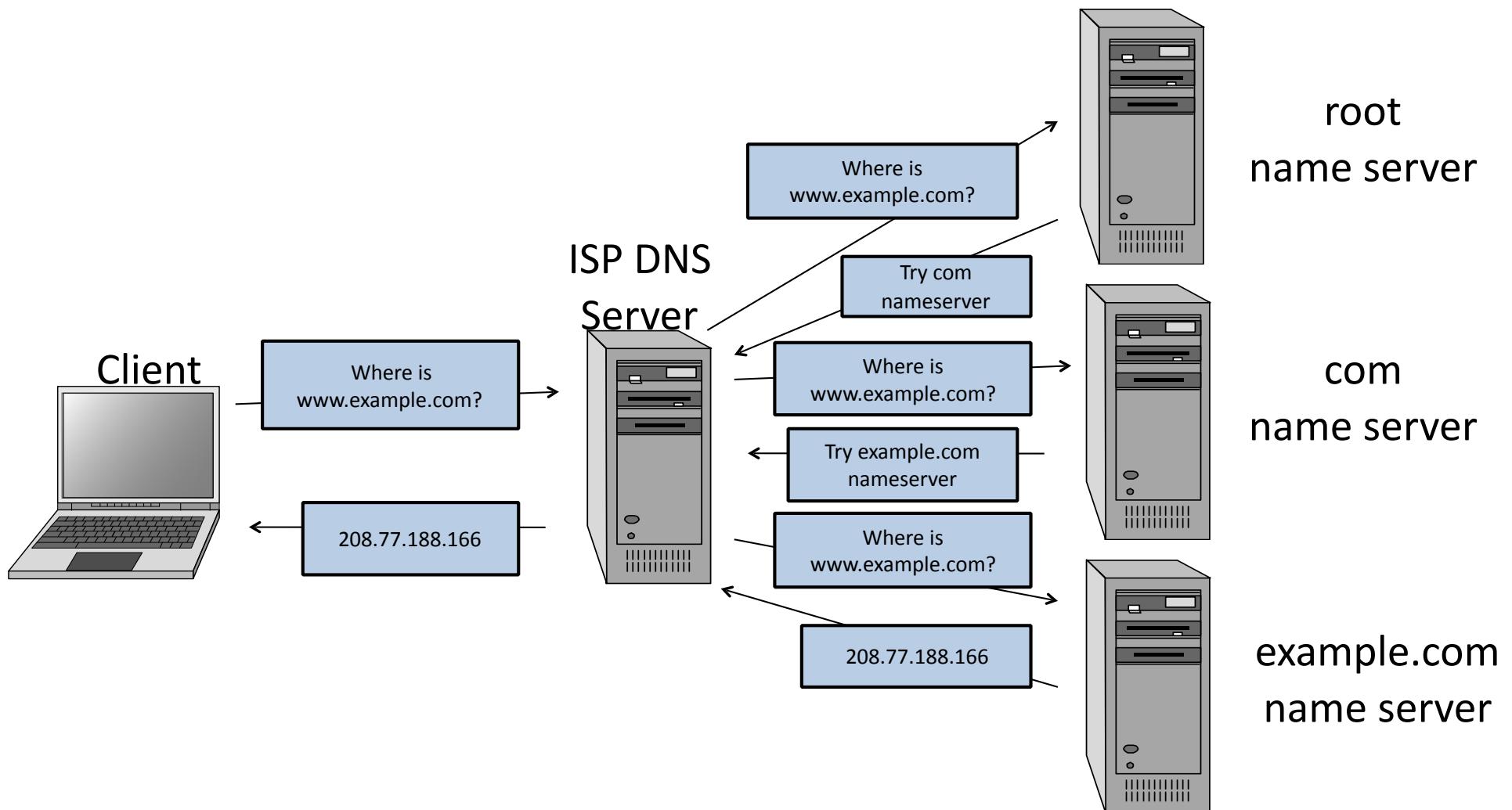
*A: doesn't scale!*

*Cannot update which machine joins/leaves the Internet.*

# Name Resolution

- Zone: collection of connected nodes with the same authoritative DNS server

Resolution method when answer not in cache:



# 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 integrity, ...

# Internet transport protocols services

---

## TCP service:

- ❖ **reliable transport** between sending and receiving process
- ❖ **flow control**: sender won't overwhelm receiver *buffer space*
- ❖ **congestion control**: throttle sender when network overloaded
- ❖ **does not provide**: timing, minimum throughput *layer* guarantee, security
- ❖ **connection-oriented**: handshake required between client and server processes

*initialer*  $\geq$  *receiver*.

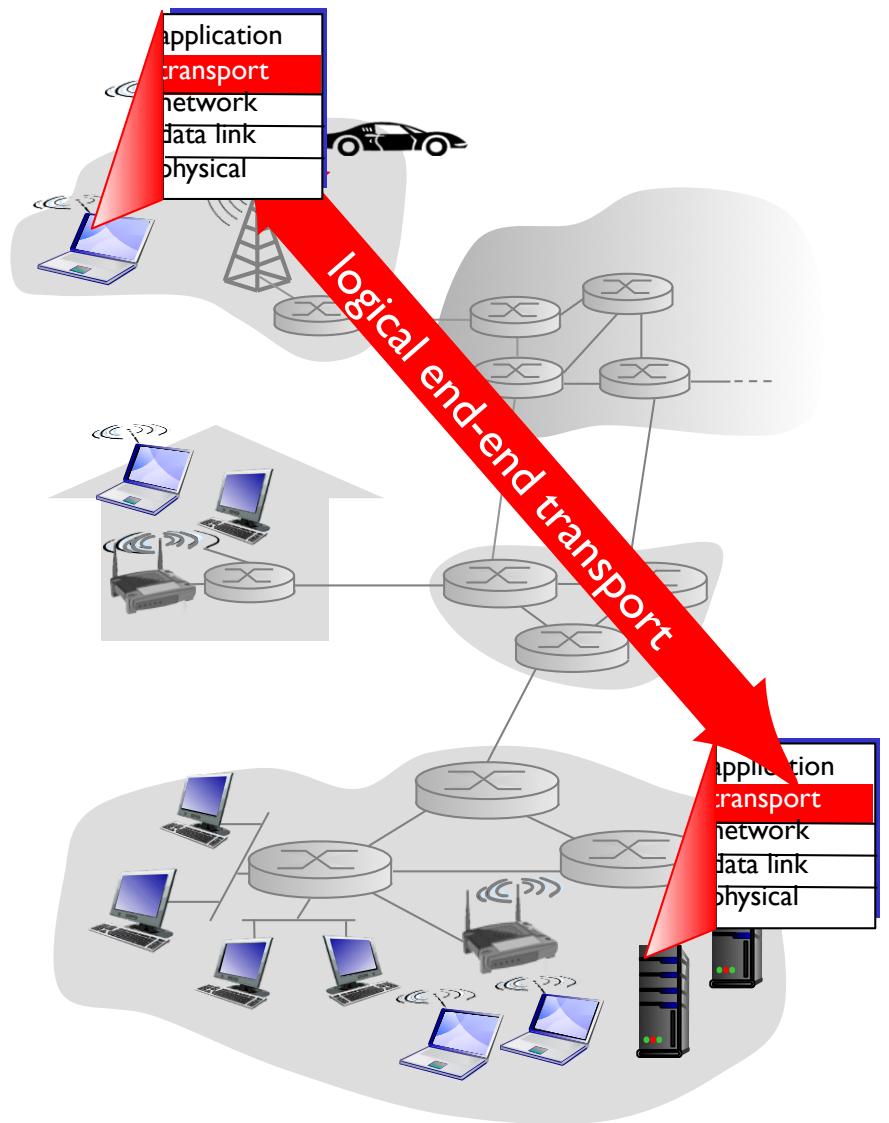
## UDP service:

- ❖ **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?

# 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



Transport Layer

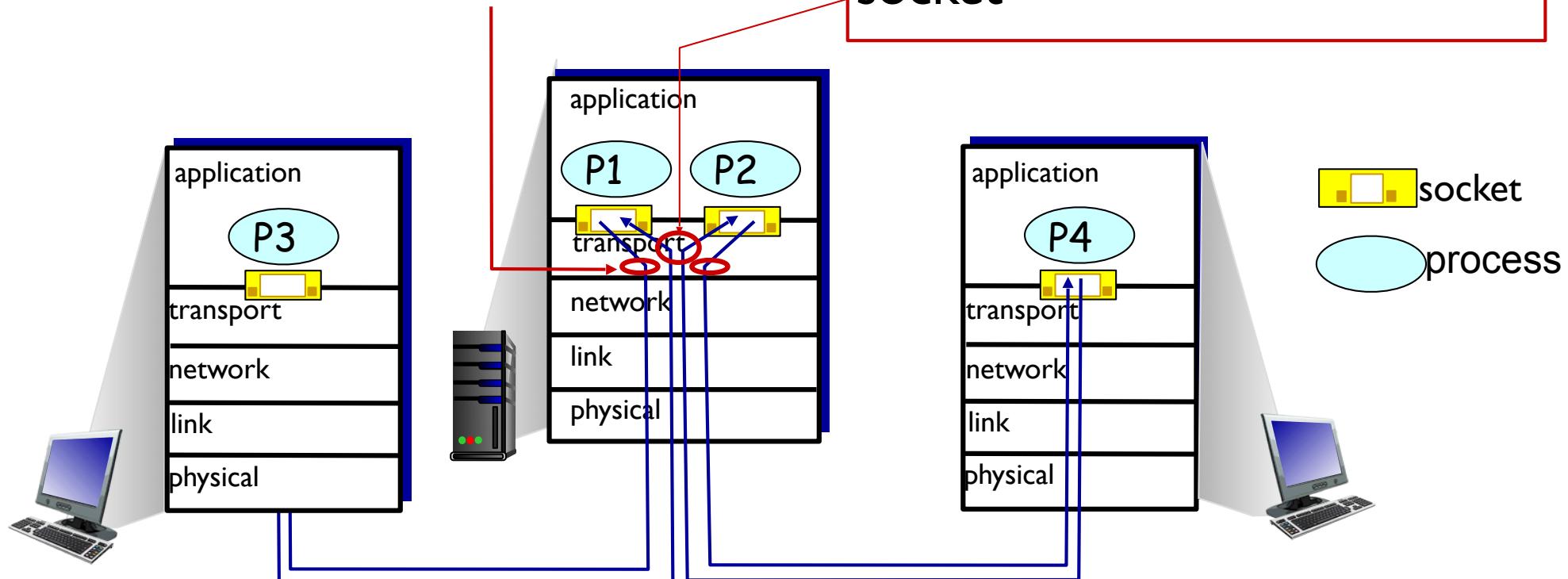
# Multiplexing/demultiplexing

*multiplexing at sender:*

handle data from multiple sockets, add transport header (later used for demultiplexing)

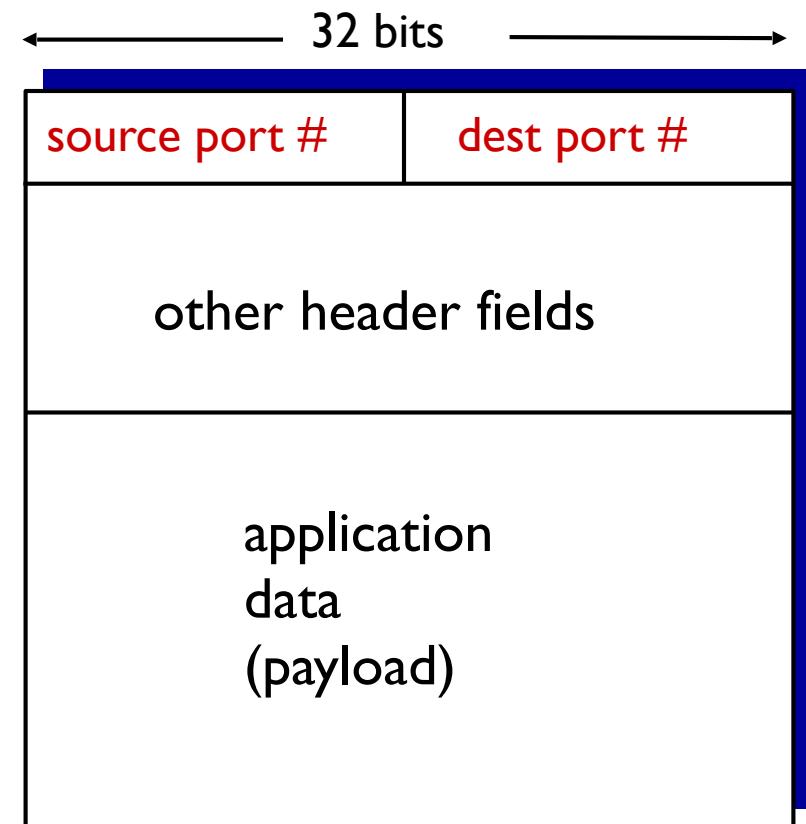
*demultiplexing at receiver:*

use header info to deliver received segments to correct socket



# How demultiplexing works

- ❖ host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- ❖ host uses *IP addresses & port numbers* to direct segment to appropriate socket

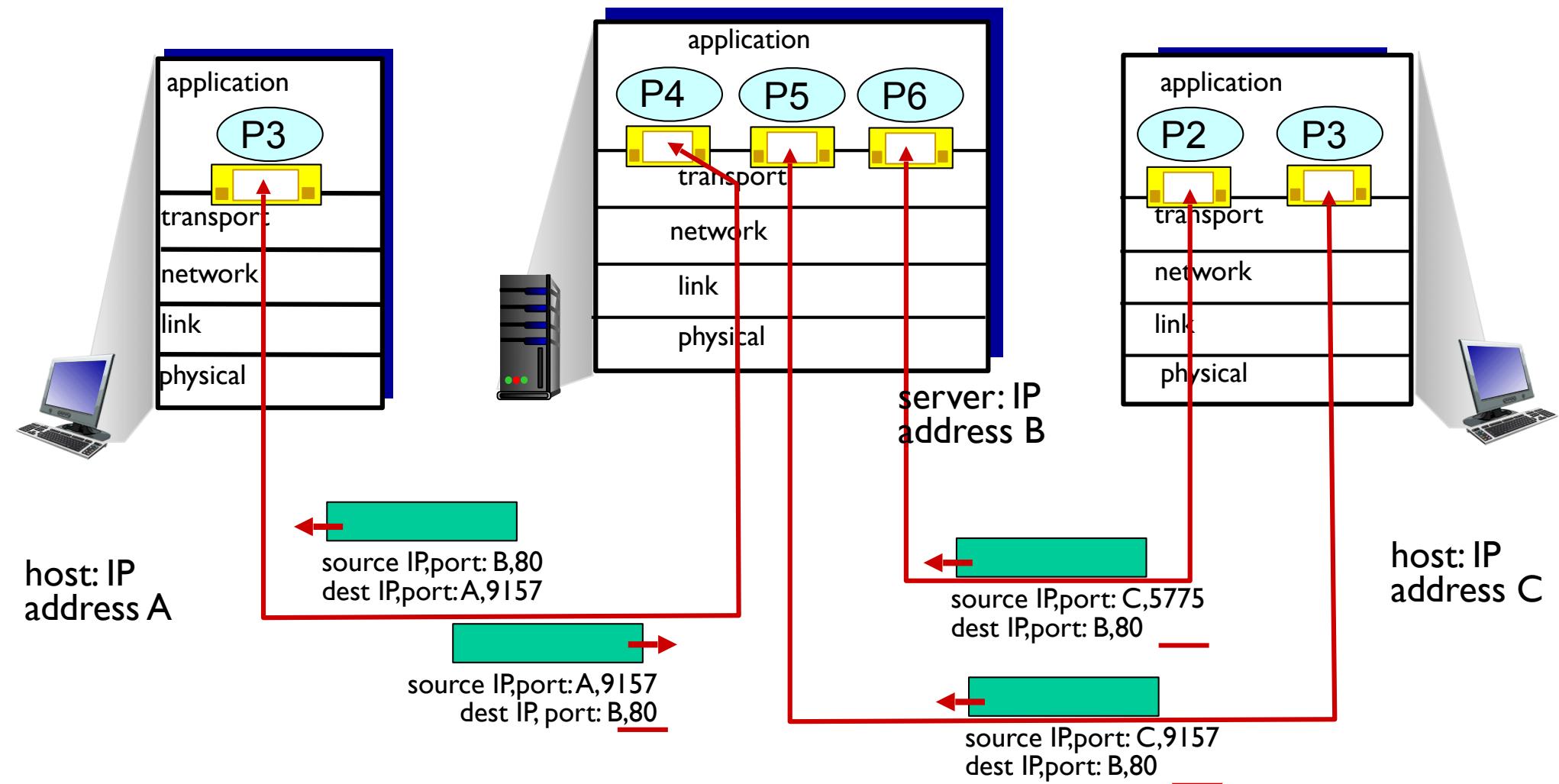


TCP/UDP segment format

# Connection-oriented demux

- ❖ TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- ❖ demux: receiver uses all four values to direct segment to appropriate socket
- ❖ server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- ❖ web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

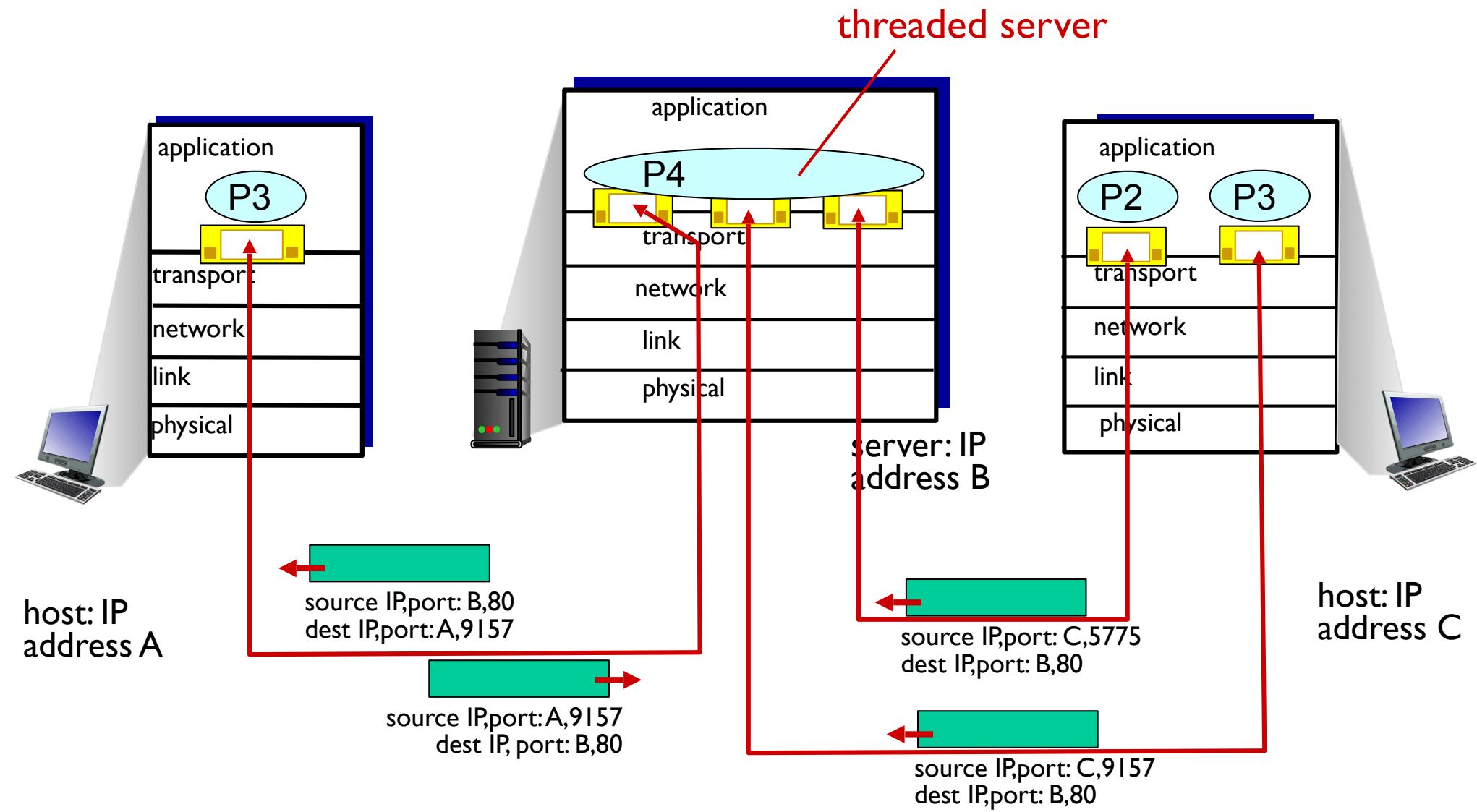
# Connection-oriented demux: example



three segments, all destined to IP address: B,  
dest port: 80 are demultiplexed to *different* sockets

Transport Layer

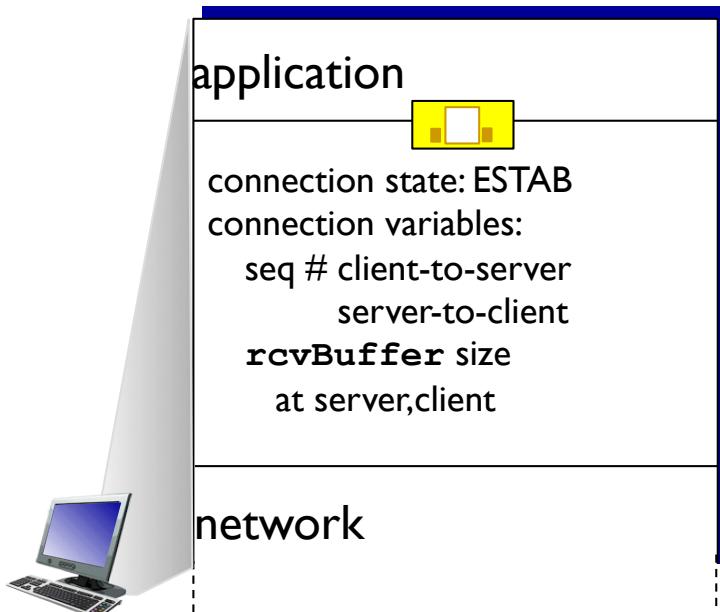
# Connection-oriented demux: example



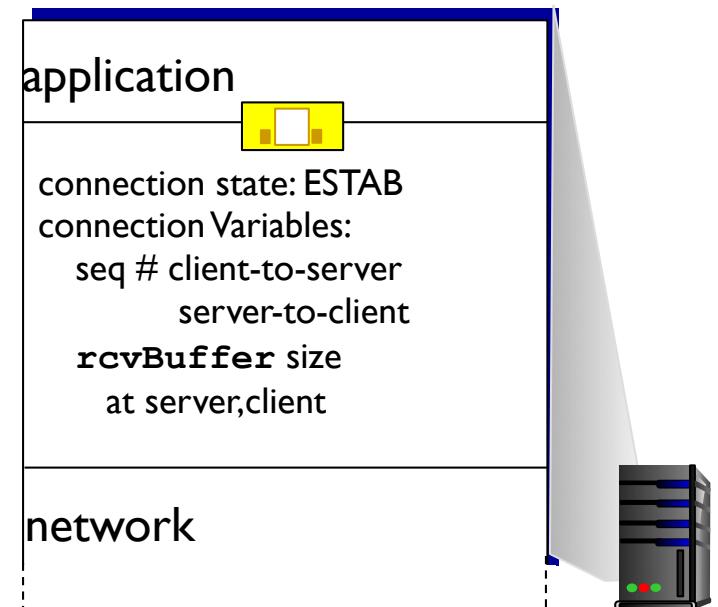
# Connection Management

before exchanging data, sender/receiver “handshake”:

- ❖ agree to establish connection (each knowing the other willing to establish connection)
- ❖ agree on connection parameters



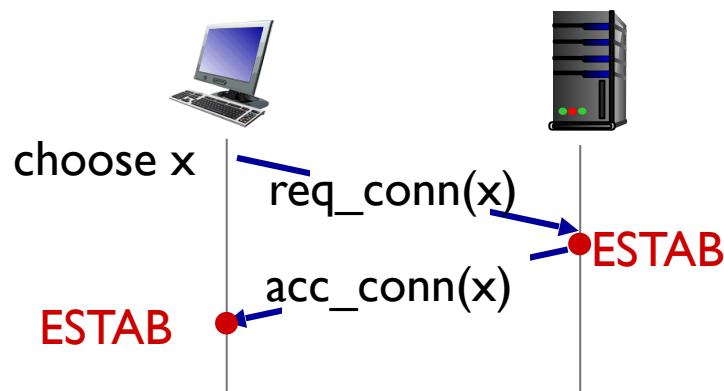
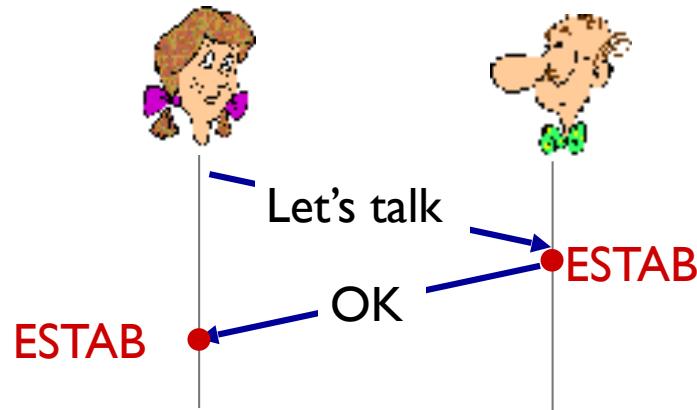
```
Socket clientSocket =  
    newSocket("hostname", "port  
number");
```



```
Socket connectionSocket =  
    welcomeSocket.accept();
```

# Agreeing to establish a connection

2-way handshake:

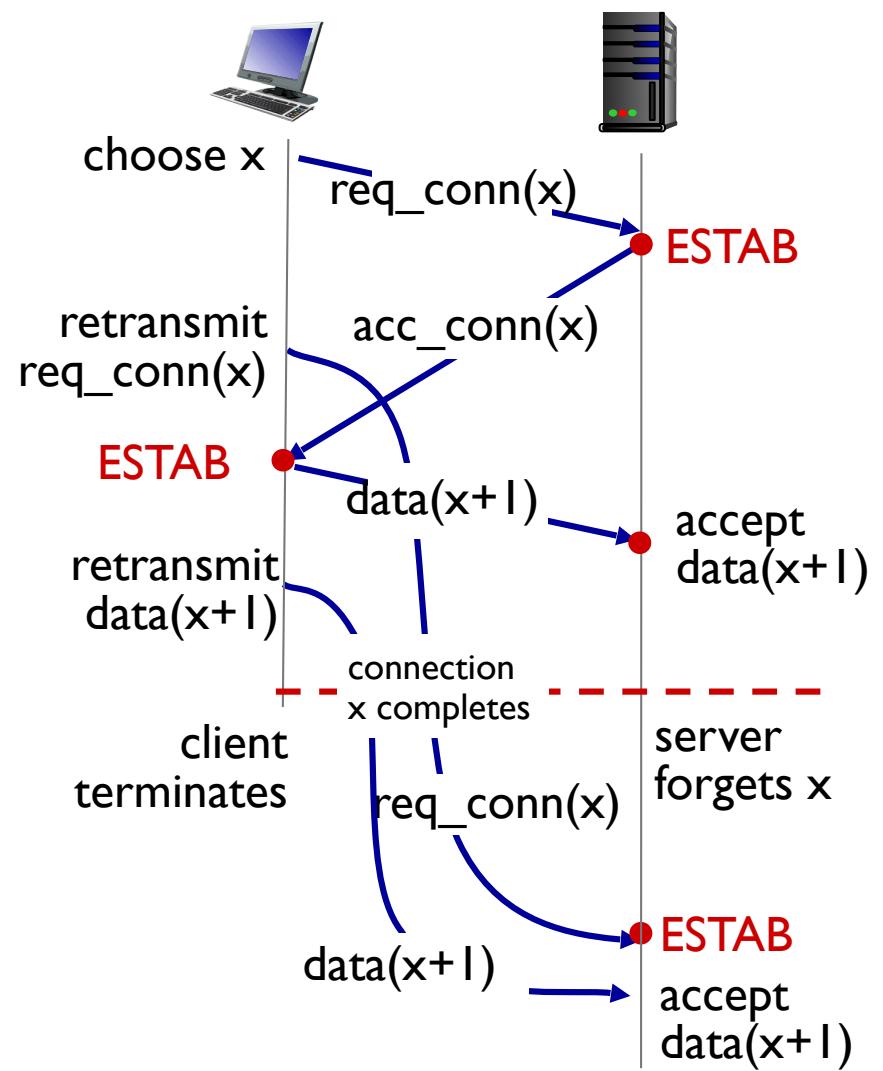
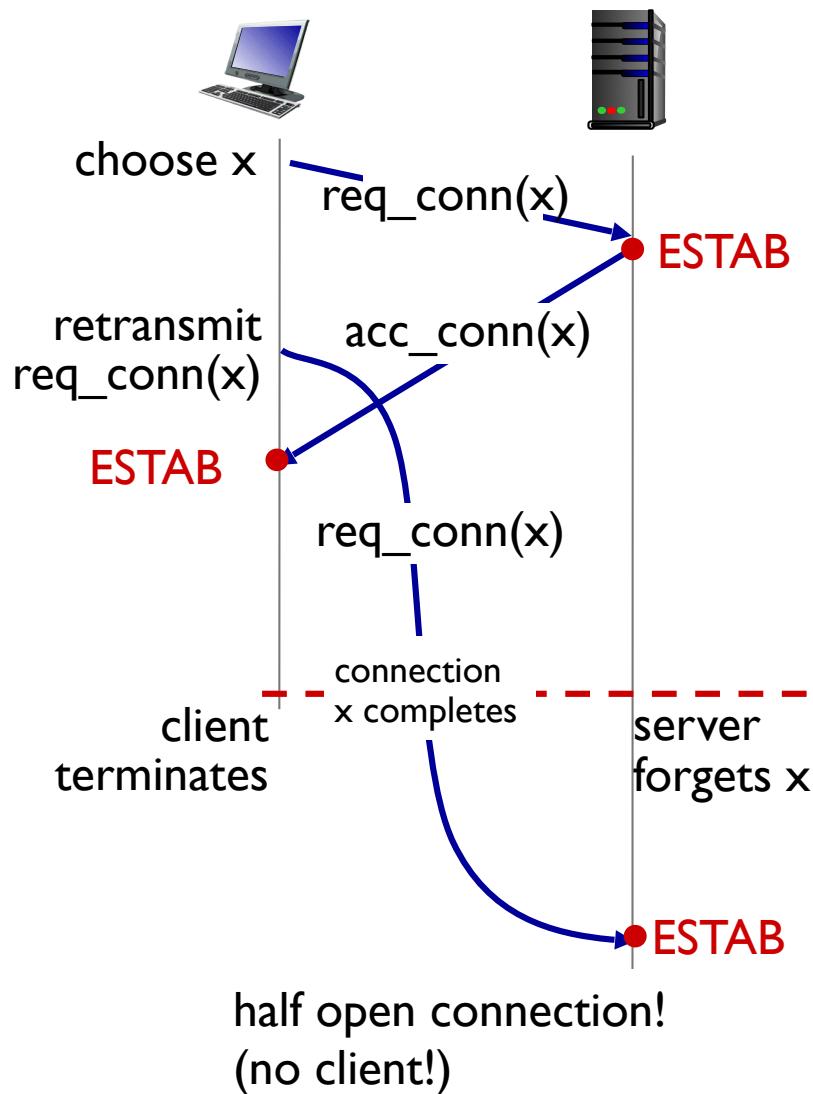


**Q:** will 2-way handshake always work in network?

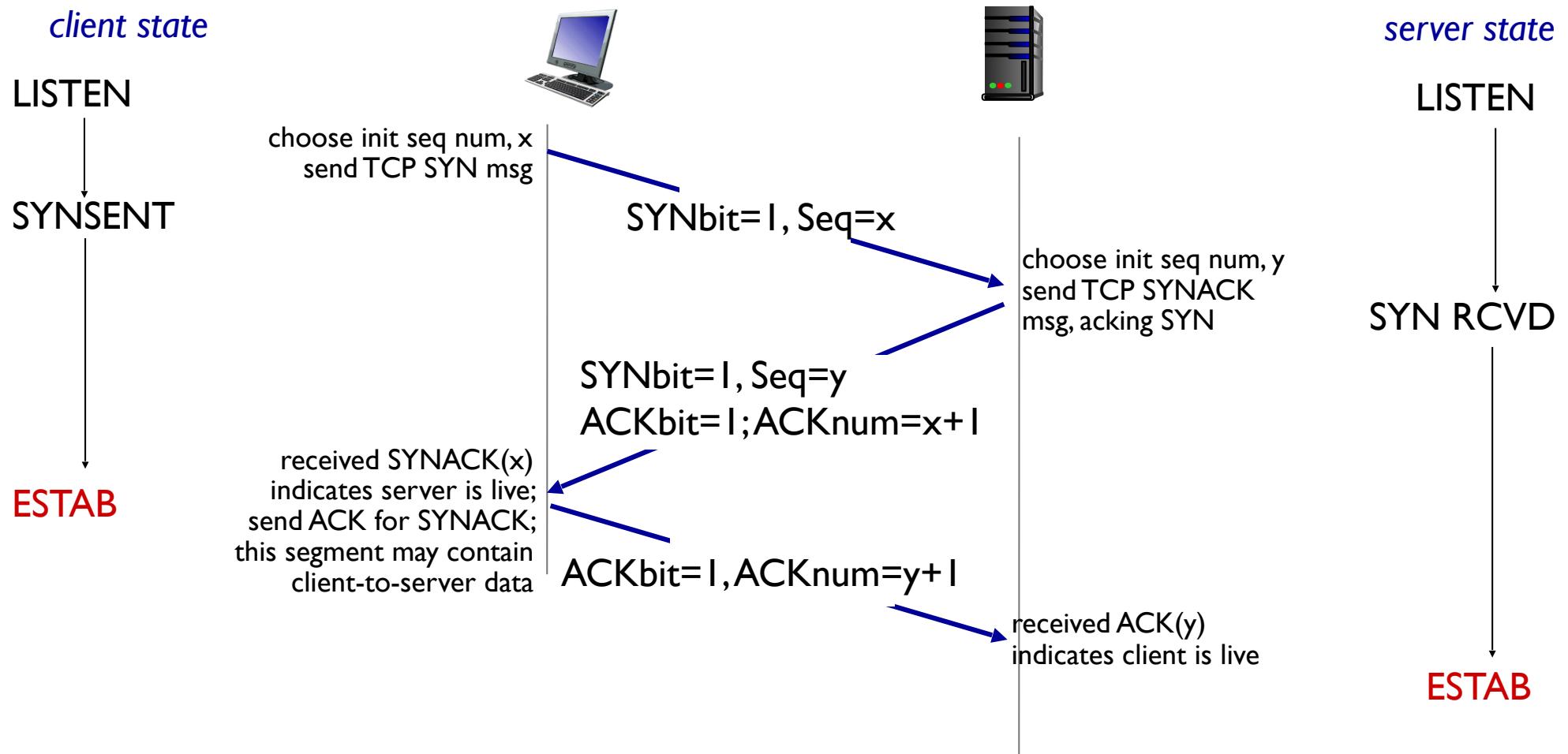
- ❖ variable delays
- ❖ retransmitted messages (e.g. `req_conn(x)`) due to message loss
- ❖ message reordering
- ❖ can't "see" other side

# Agreeing to establish a connection

2-way handshake failure scenarios:



# TCP 3-way handshake



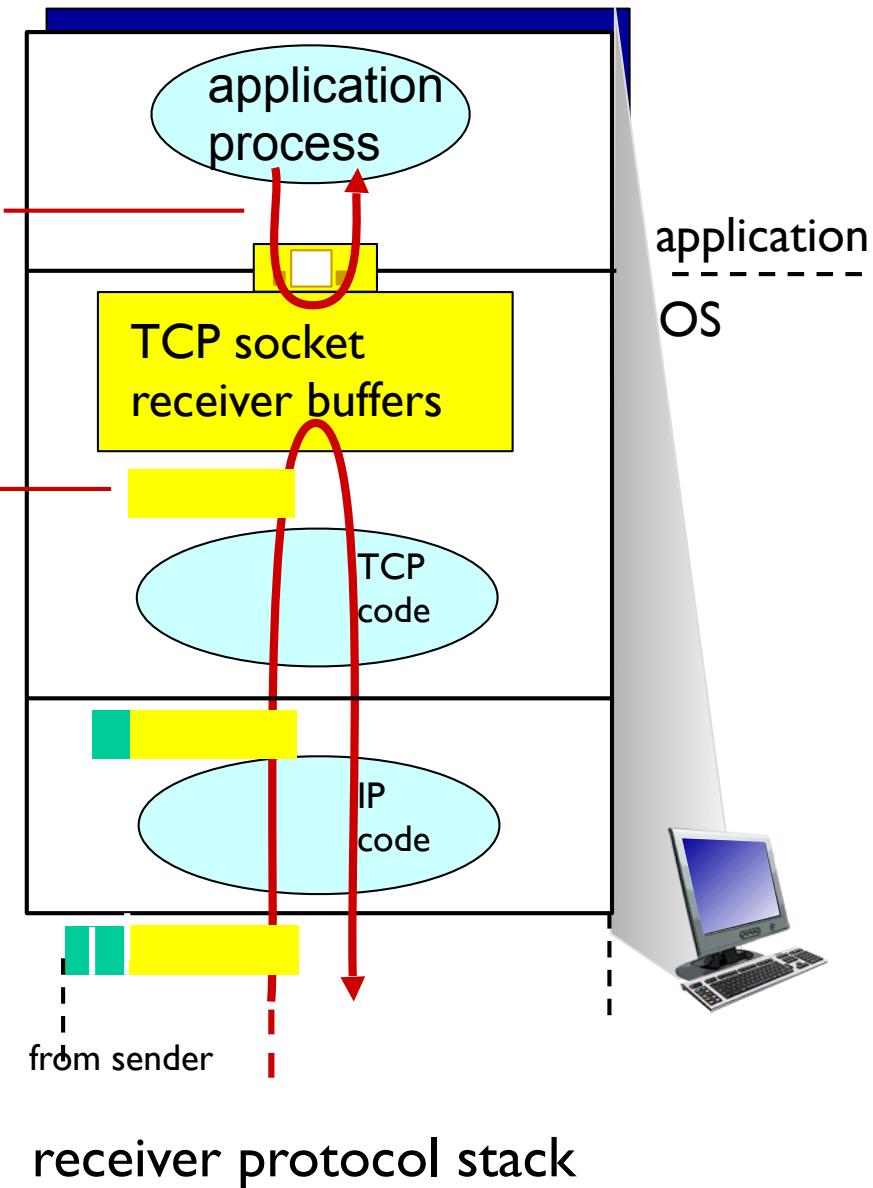
# TCP flow control

application may  
remove data from  
TCP socket buffers ....

... slower than TCP  
receiver is delivering  
(sender is sending)

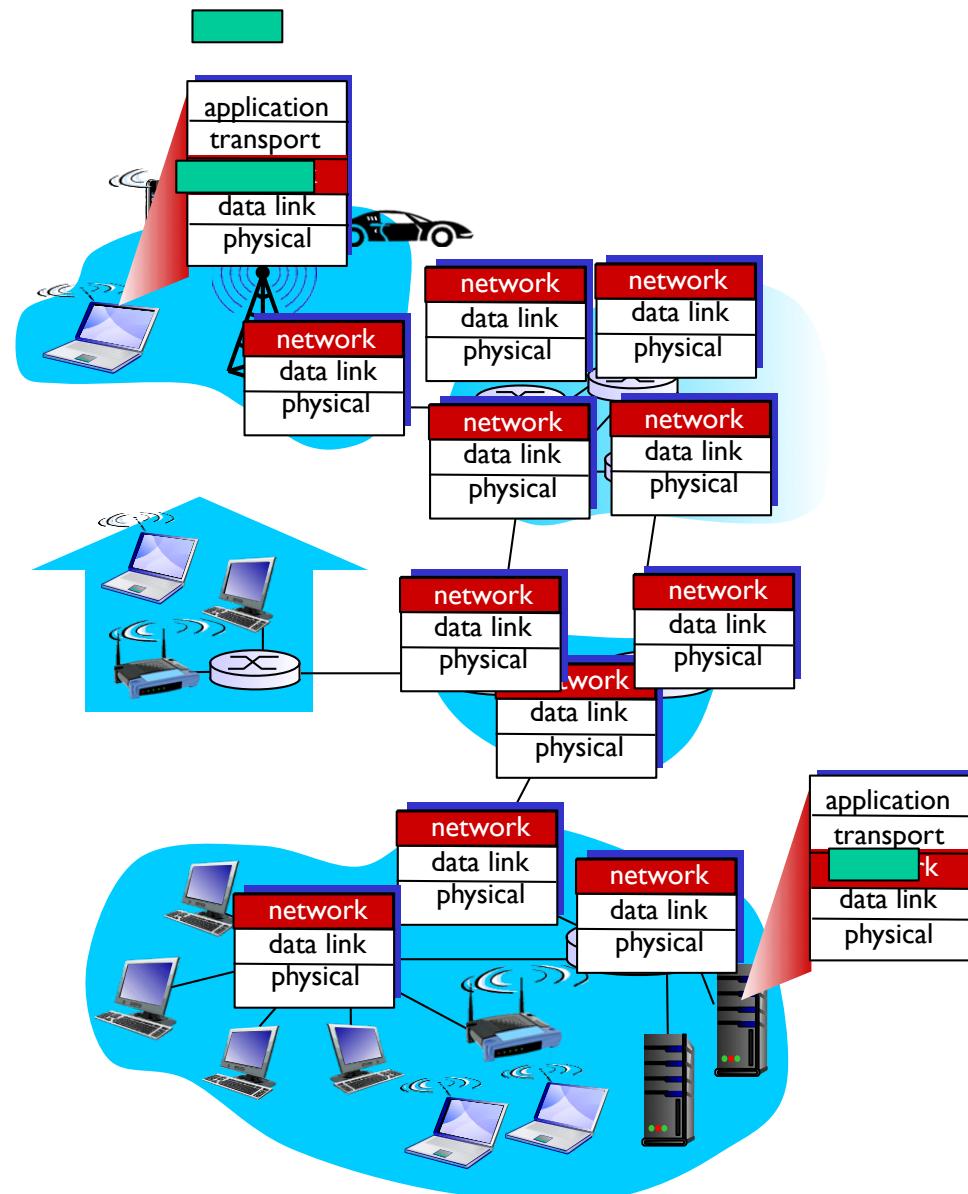
## **flow control**

receiver controls sender, so  
sender won't overflow receiver's  
buffer by transmitting too much,  
too fast



# Network layer

- ❖ transport segment from sending to receiving host
- ❖ on sending side encapsulates segments into datagrams
- ❖ on receiving side, delivers segments to transport layer
- ❖ network layer protocols in **every** host, router
- ❖ router examines header fields in all IP datagrams passing through it



# Internet Protocol

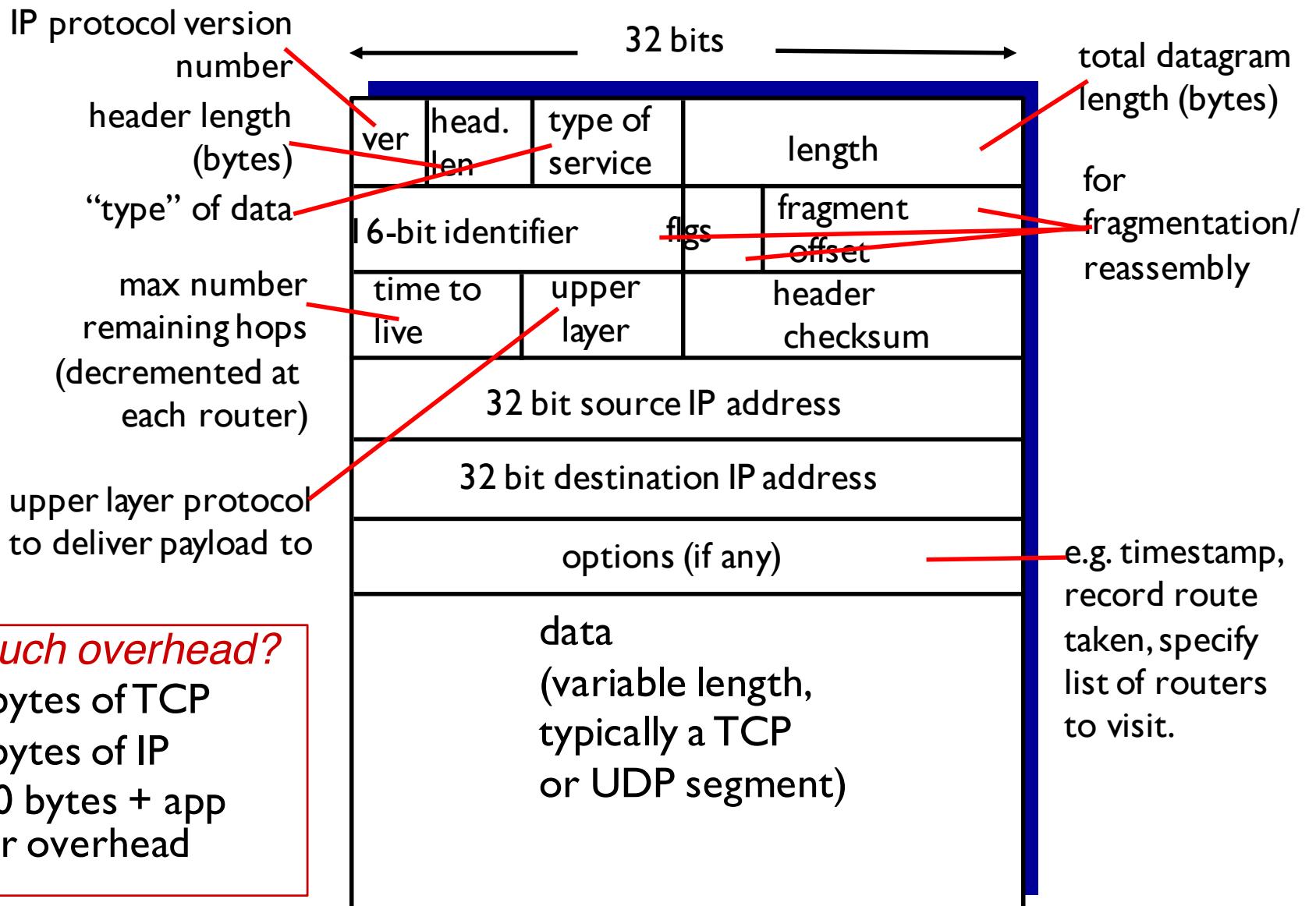
- Connectionless
  - Each packet is transported independently from other packets
- Unreliable
  - Delivery on a best effort basis
  - No acknowledgments
  - Packets may be lost, reordered, corrupted, or duplicated
- IP packets
  - Encapsulate TCP and UDP packets
  - Encapsulated into link-layer frames

Data link frame

IP packet

TCP or UDP packet

# IP Packet Format



# IP Addressing

- IP address used to route datagrams through network.
- IPv4 32 bit address, IPv6 128 bit address.
- Divided into two parts: network and host.
- Network part: Used to route packets. (Street name)
- Host part: Used to identify an individual host. (House number)
- Usually represented in dotted decimal notation:  
141.212.112.111
- Each number represents 8 bits: 0–255.

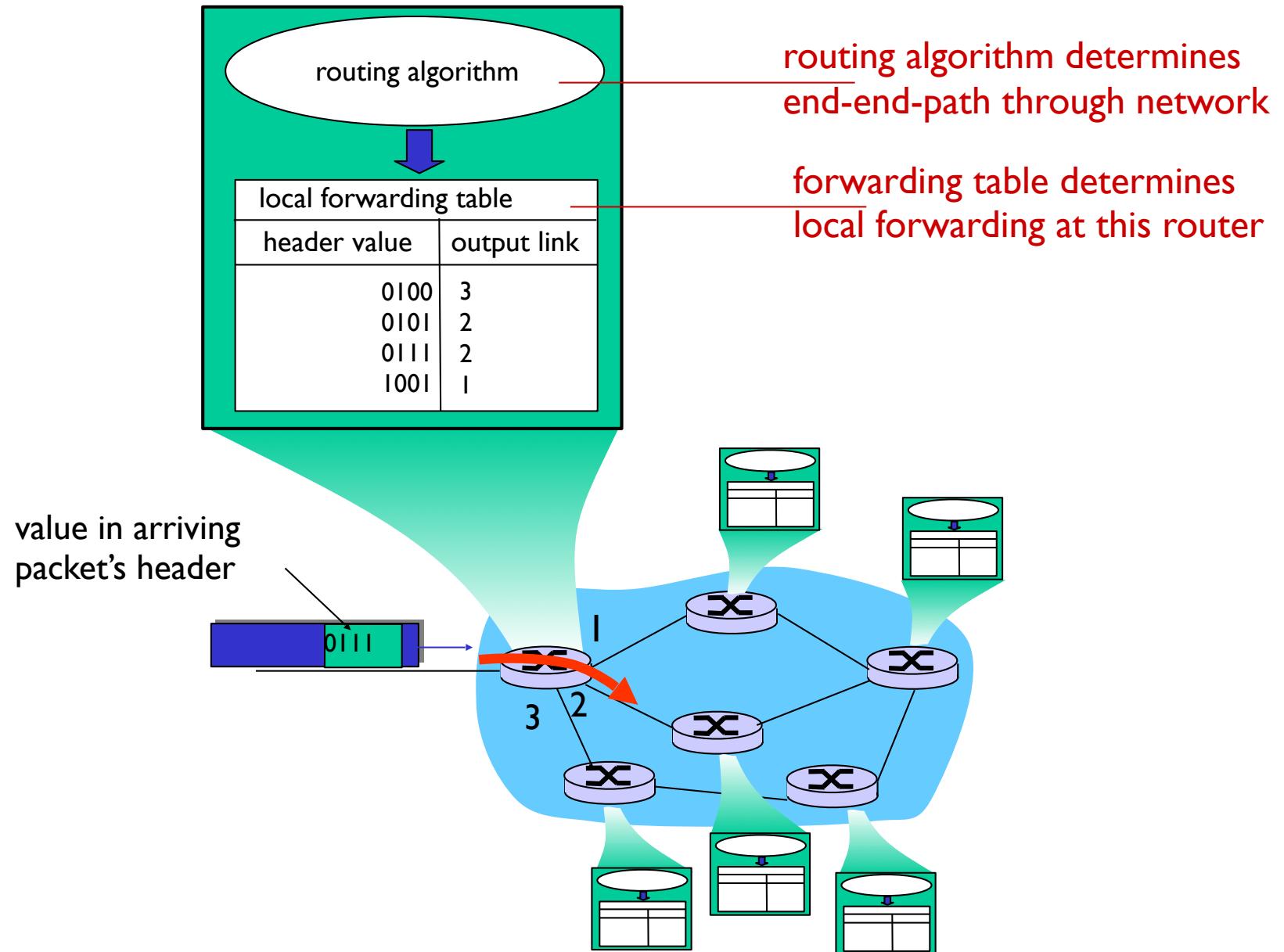
# Two key network-layer functions

- ❖ *forwarding*: move packets from router's input to appropriate router output
- ❖ *routing*: determine route taken by packets from source to dest.
  - *routing algorithms*

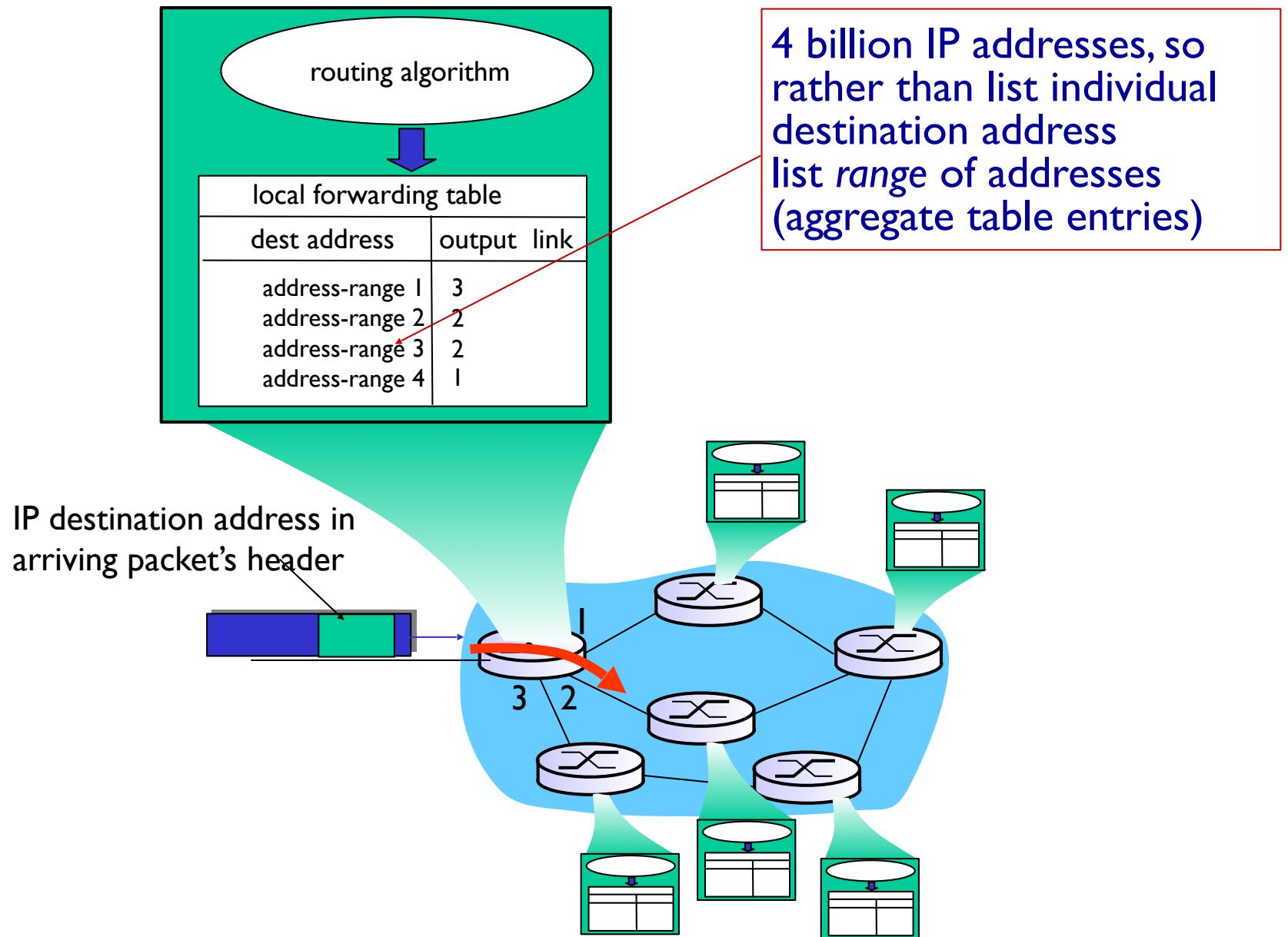
*analogy:*

- ❖ *routing*: process of planning trip from source to dest
- ❖ *forwarding*: process of getting through single intersection

# Interplay between routing and forwarding



# Datagram forwarding table



# Datagram forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

# Longest prefix matching

*longest prefix matching*

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** **** *****	0
11001000 00010111 00011000 **** *****	1
11001000 00010111 00011*** **** *****	2
otherwise	3

examples:

DA: 11001000 00010111 00010110 1010001

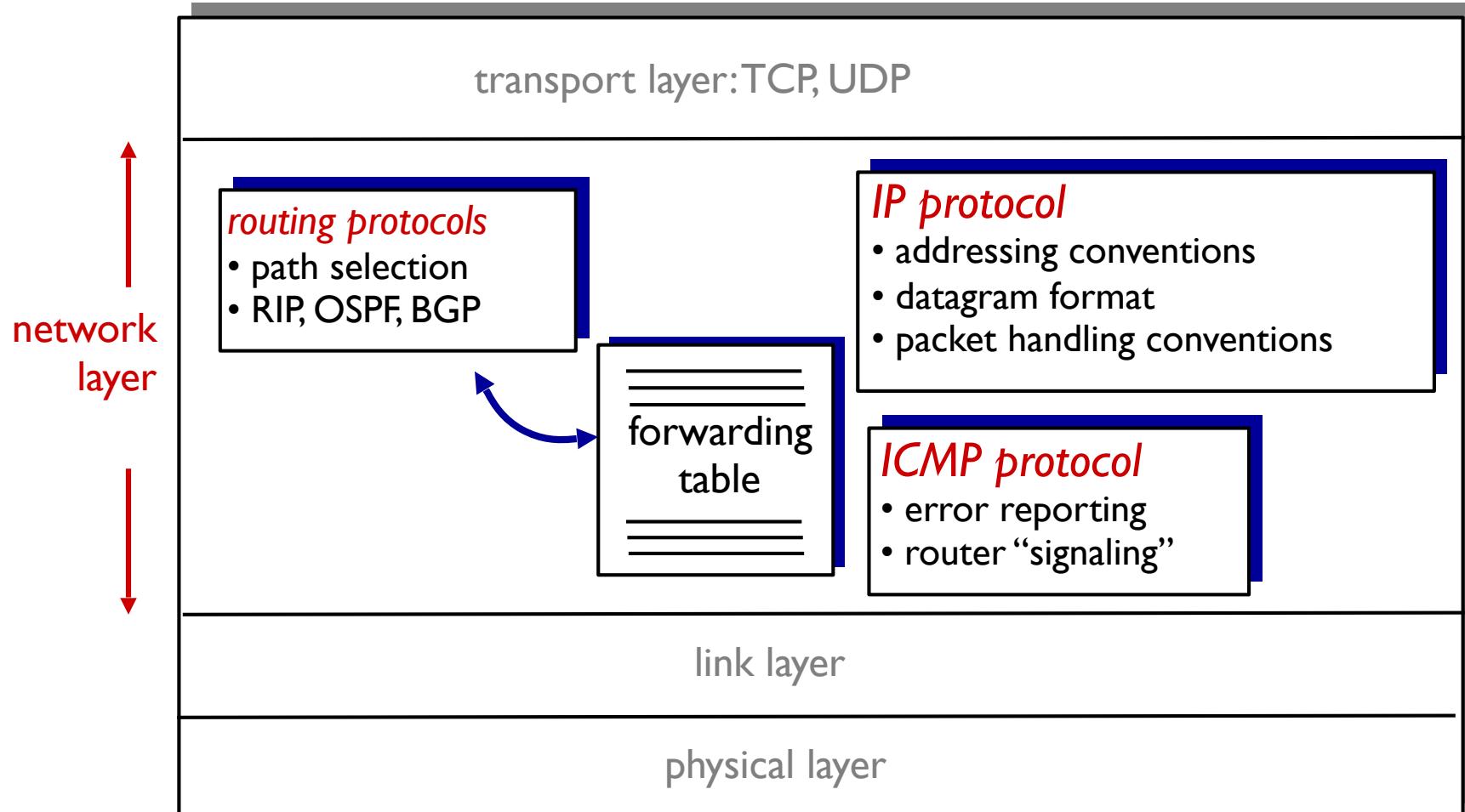
which interface?

DA: 11001000 00010111 00011000 10101010

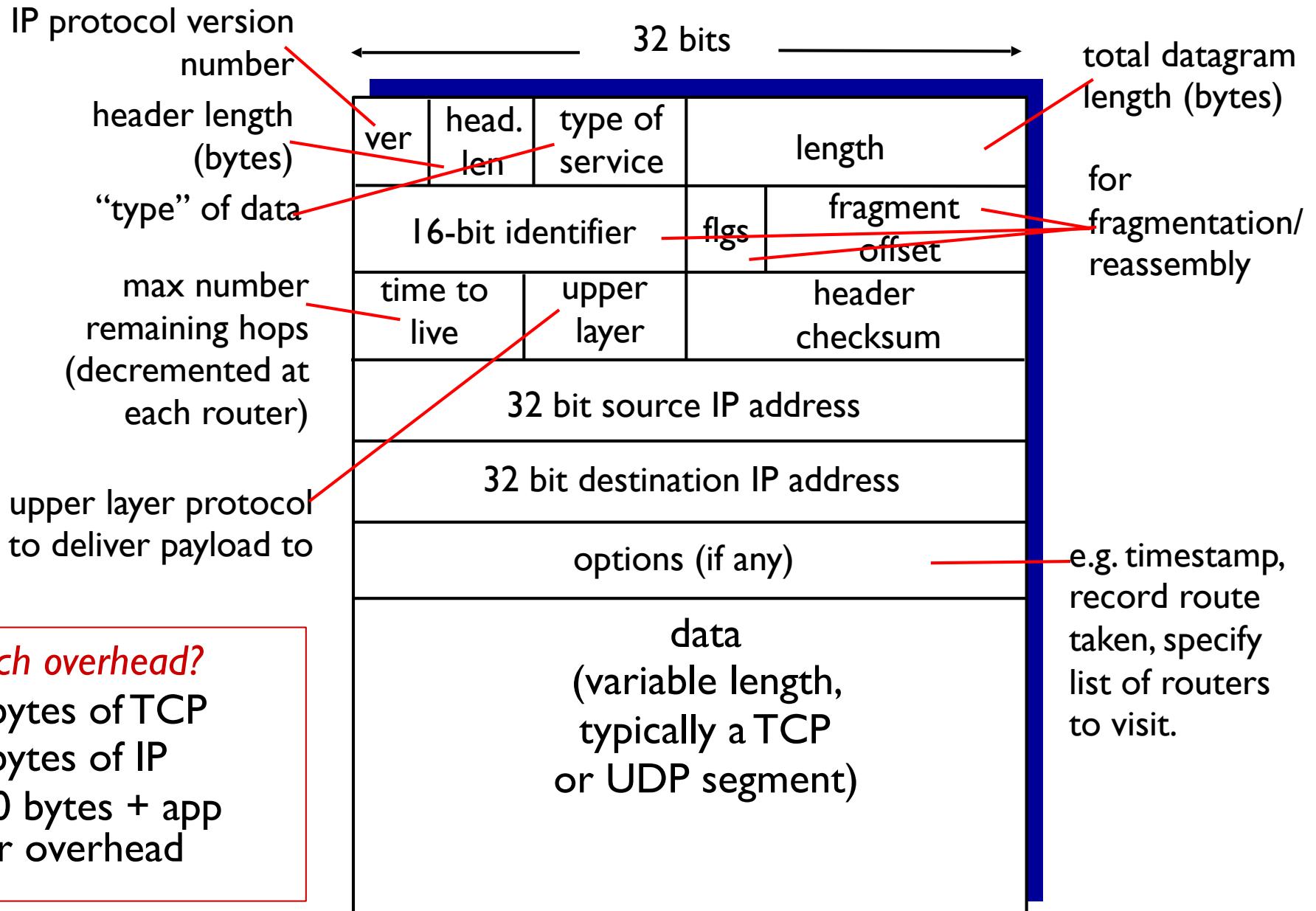
which interface?

# The Internet network layer

host, router network layer functions:



# IP datagram format

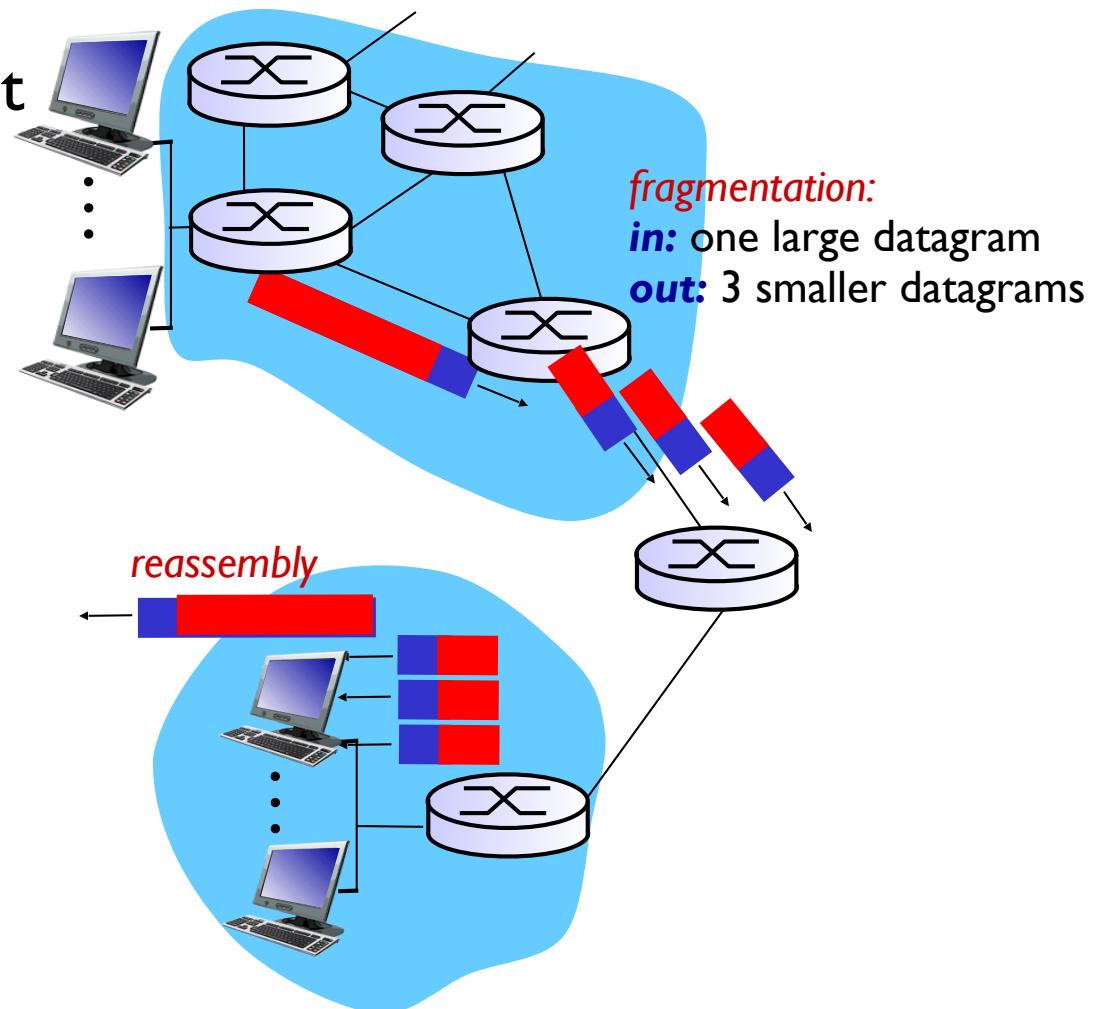


## how much overhead?

- ❖ 20 bytes of TCP
- ❖ 20 bytes of IP
- ❖ = 40 bytes + app layer overhead

# IP fragmentation, reassembly

- ❖ network links have MTU (max.transfer size) - largest possible link-level frame
  - different link types, different MTUs
- ❖ large IP datagram divided (“fragmented”) within net
  - one datagram becomes several datagrams
  - “reassembled” only at final destination
  - IP header bits used to identify, order related fragments



# IP fragmentation, reassembly

**example:**

- ❖ 4000 byte datagram
- ❖ MTU = 1500 bytes

	length =4000	ID =x	fragflag =0	offset =0	
--	-----------------	----------	----------------	--------------	--

*one large datagram becomes several smaller datagrams*

1480 bytes in data field

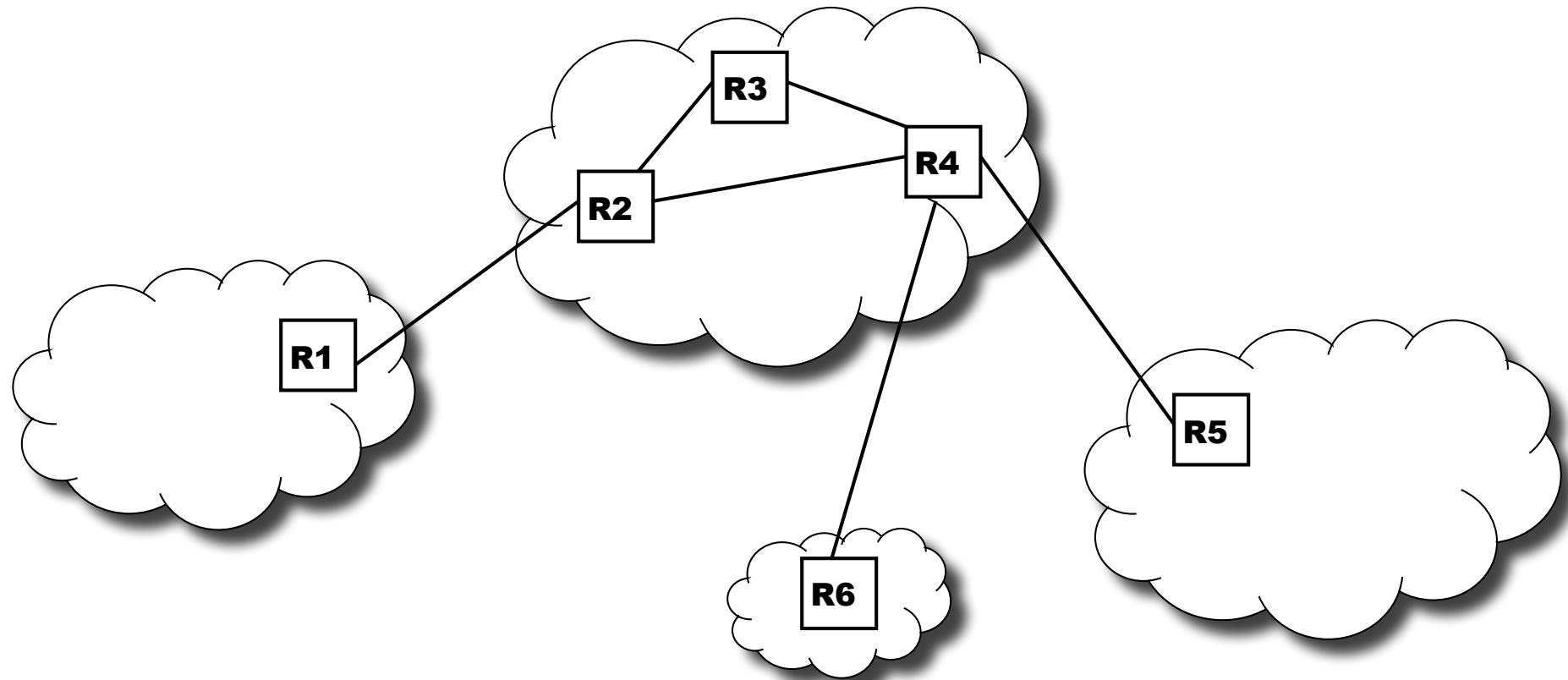
offset =  
 $1480/8$

	length =1500	ID =x	fragflag =1	offset =0	
--	-----------------	----------	----------------	--------------	--

	length =1500	ID =x	fragflag =1	offset =185	
--	-----------------	----------	----------------	----------------	--

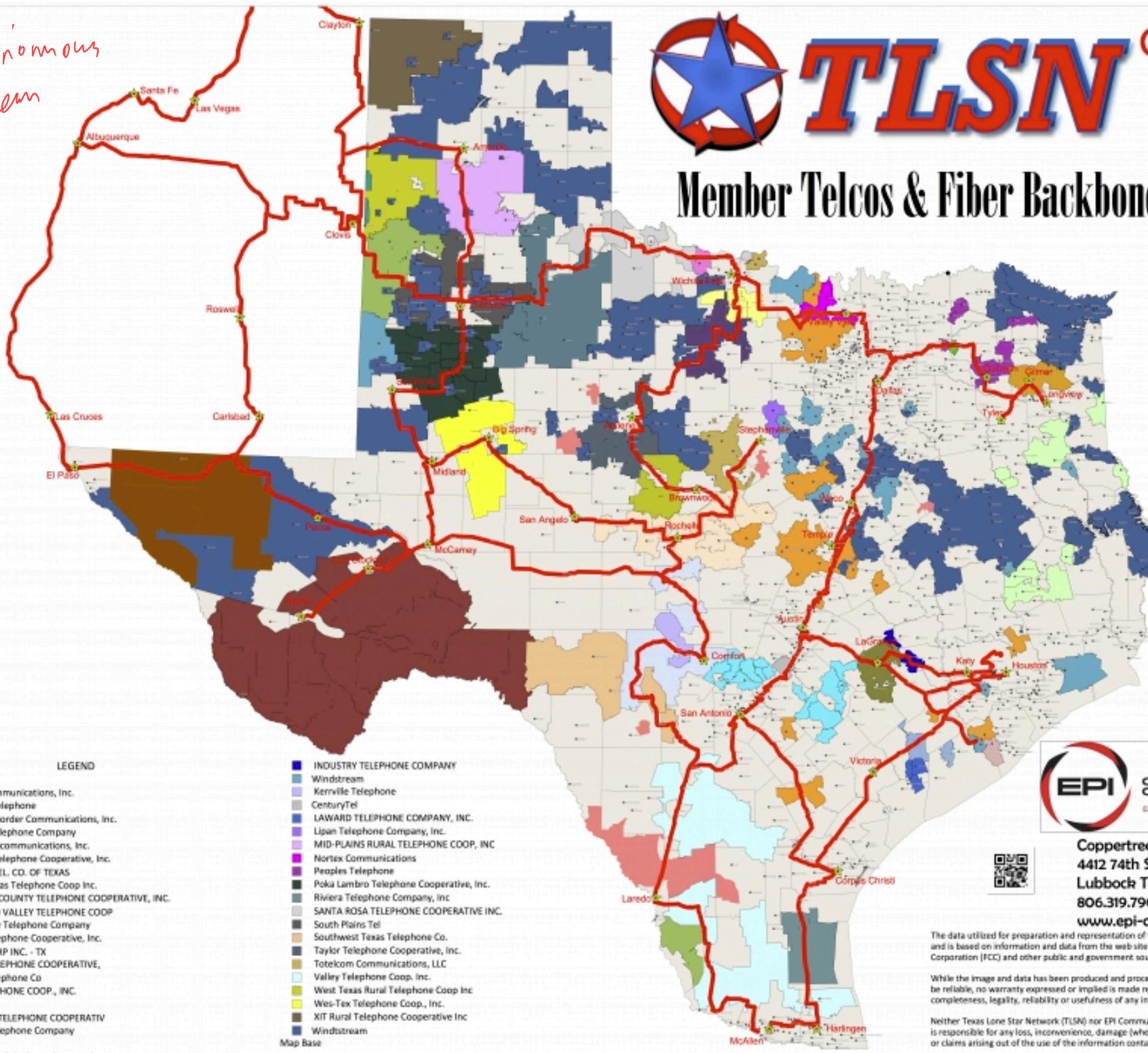
	length =1040	ID =x	fragflag =0	offset =370	
--	-----------------	----------	----------------	----------------	--

# Autonomous Systems (AS)



- The Internet is a collection of autonomous systems.
- AS: A set of routers and networks under the same administrative control.  
*BGP*      *OSPF*
- Inter-domain vs. intra-domain routing.

RTS  
Ranomous  
System



Member Telcos & Fiber Backbone



Coppertree Business Park  
4412 74th Street, Suite C-100  
Lubbock Texas 79424  
806.319.7909  
[www.epi-cc.com](http://www.epi-cc.com)



The data utilized for preparation and representation of this image is in public domain and is based on information and data from the web site of the Federal Communications Corporation (FCC) and other public and government sources.

While the image and data has been produced and processed from sources believed to be reliable, no warranty expressed or implied is made regarding accuracy, adequacy, completeness, legality, reliability or usefulness of any information.

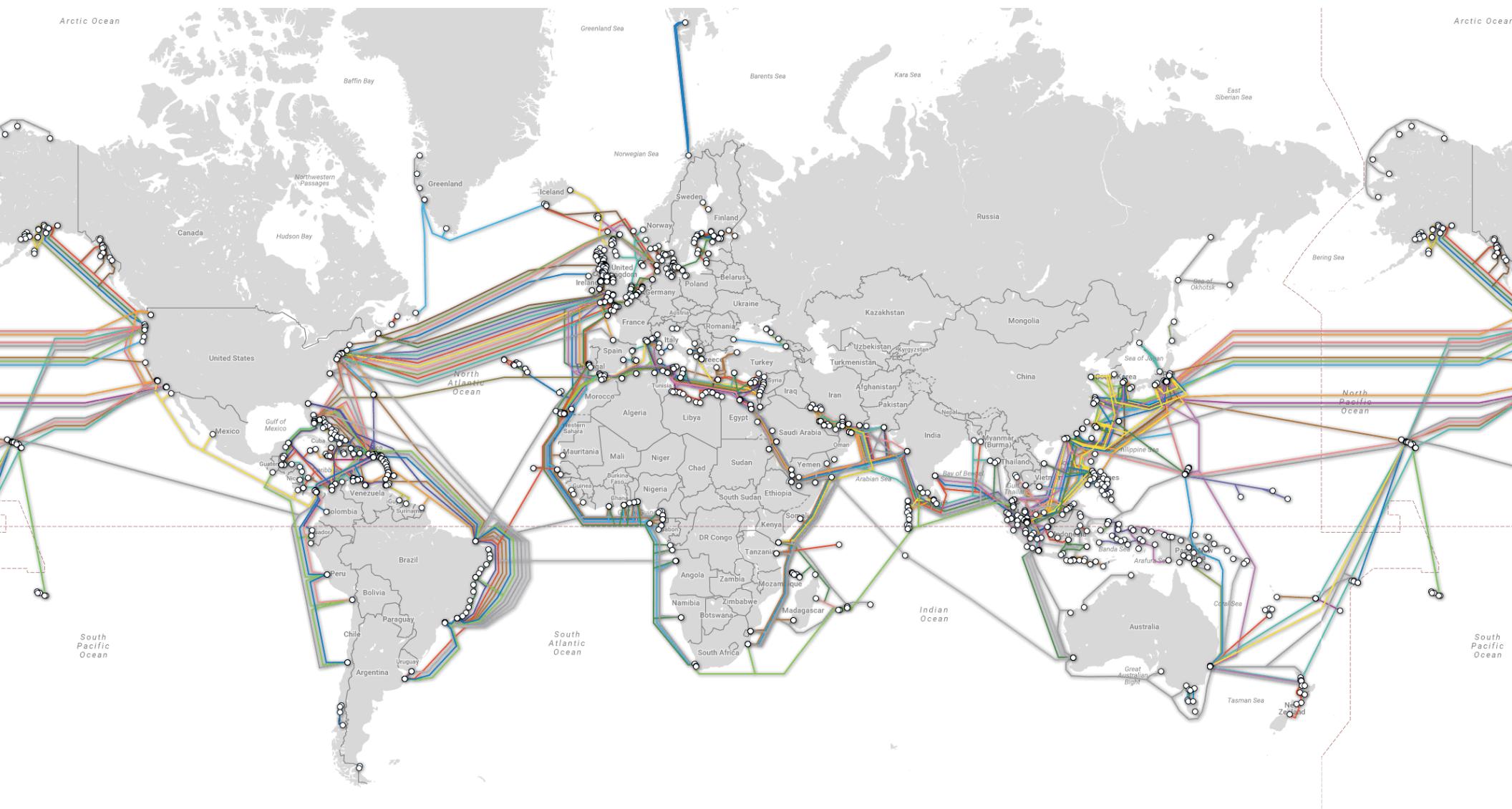
Neither Texas Lone Star Network (TLSN) nor EPI Communications Consultants, LLC (EPI) is responsible for any loss, inconvenience, damage (whether special or consequential) or claims arising out of the use of the information contained on this site.

TLSN: Does not have direct fiber connectivity to all Telco Members Fiber.

# Level3 U.S. fiber (2014)



# Global marine fiber (2017)



# Hierarchical routing

our routing study thus far - idealization

- ❖ all routers identical
- ❖ network “flat”

... *not true in practice*

**scale:** with 600 million destinations:

- ❖ can't store all dest's in routing tables!
- ❖ routing table exchange would swamp links!

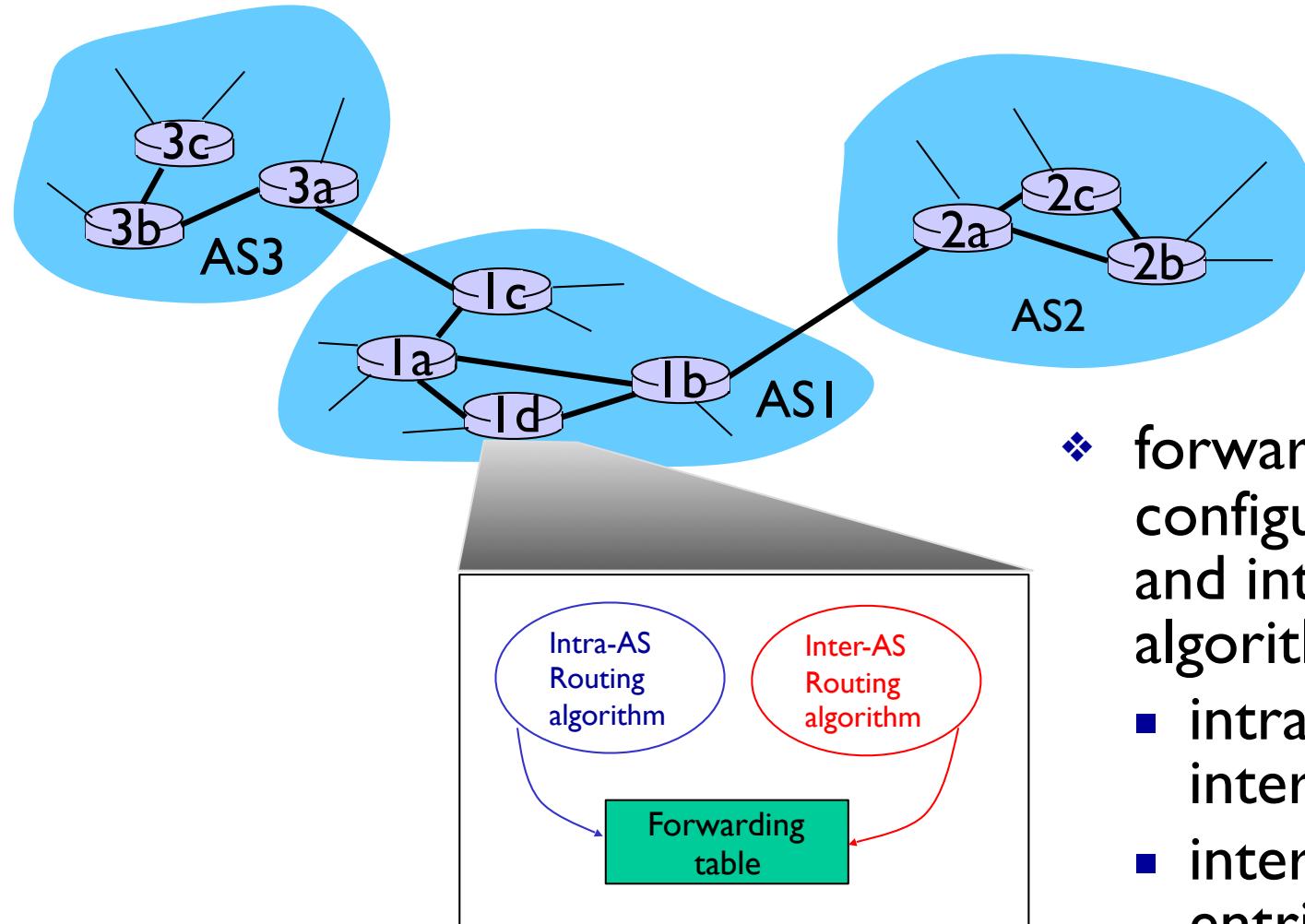
***administrative autonomy***

- ❖ internet = network of networks
- ❖ each network admin may want to control routing in its own network

# Hierarchical routing

- ❖ aggregate routers into regions, “autonomous systems” (AS)
  - ❖ routers in same AS run same routing protocol
    - “intra-AS” routing protocol
    - routers in different AS can run different intra-AS routing protocol
- gateway router:*
- ❖ at “edge” of its own AS
  - ❖ has link to router in another AS

# Interconnected ASes



- ❖ **forwarding table**  
configured by both intra- and inter-AS routing algorithm
  - intra-AS sets entries for internal dests
  - inter-AS & intra-AS sets entries for external dests

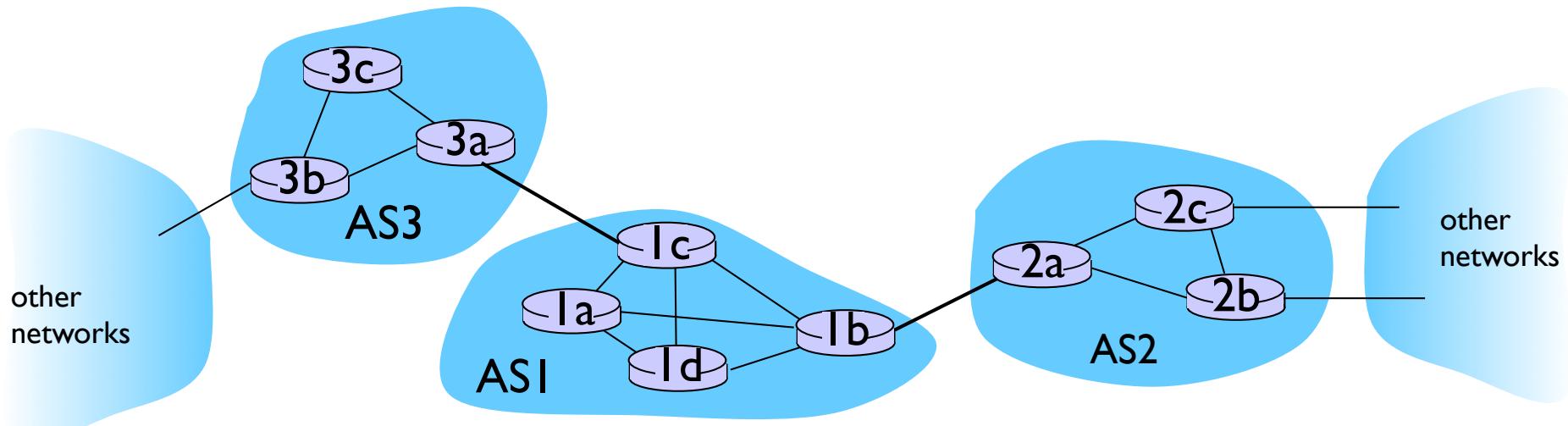
# Inter-AS tasks

- ❖ suppose router in ASI receives datagram destined outside of ASI:
  - router should forward packet to gateway router, but which one?

*AS1 must:*

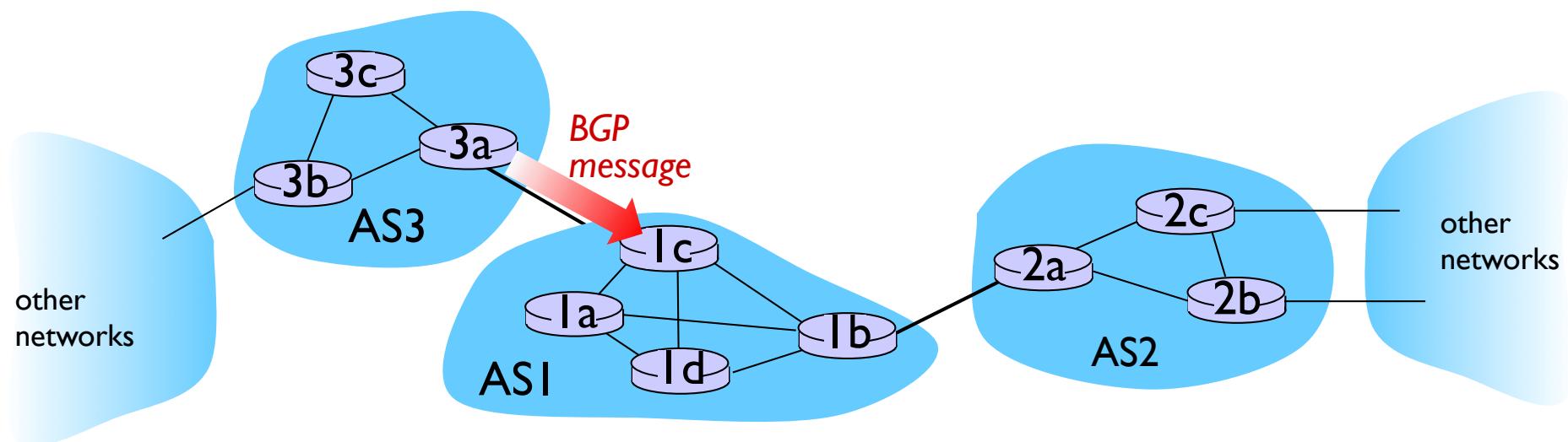
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in ASI

*job of inter-AS routing!*



# BGP basics

- ❖ **BGP session:** two BGP routers (“peers”) exchange BGP messages:
    - advertising *paths* to different destination network prefixes (“path vector” protocol)
    - exchanged over semi-permanent TCP connections
  - ❖ when AS3 advertises a prefix to AS1:
    - AS3 *promises* it will forward datagrams towards that prefix
    - AS3 can aggregate prefixes in its advertisement

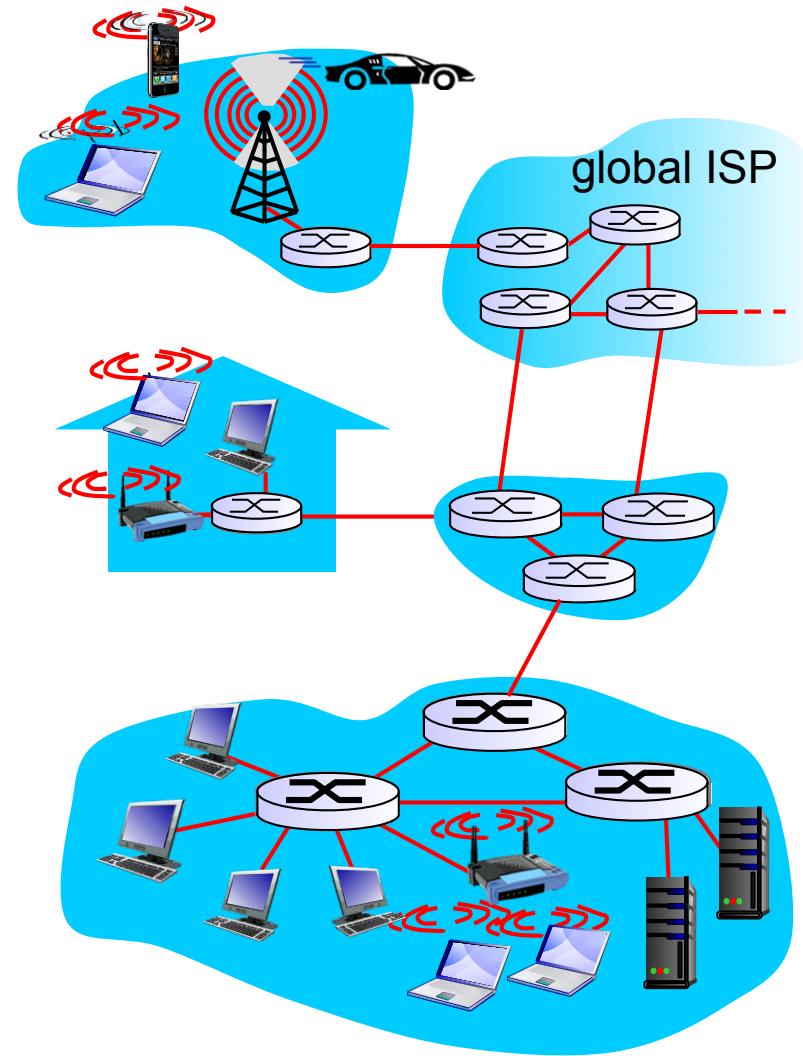


# Link layer: introduction

## terminology:

- ❖ hosts and routers: **nodes**
- ❖ communication channels that connect adjacent nodes along communication path: **links**
  - wired links
  - wireless links
  - LANs
- ❖ layer-2 packet: **frame**, encapsulates datagram

**data-link layer** has responsibility of transferring datagram from one node to **physically adjacent** node over a link



# Link layer: context

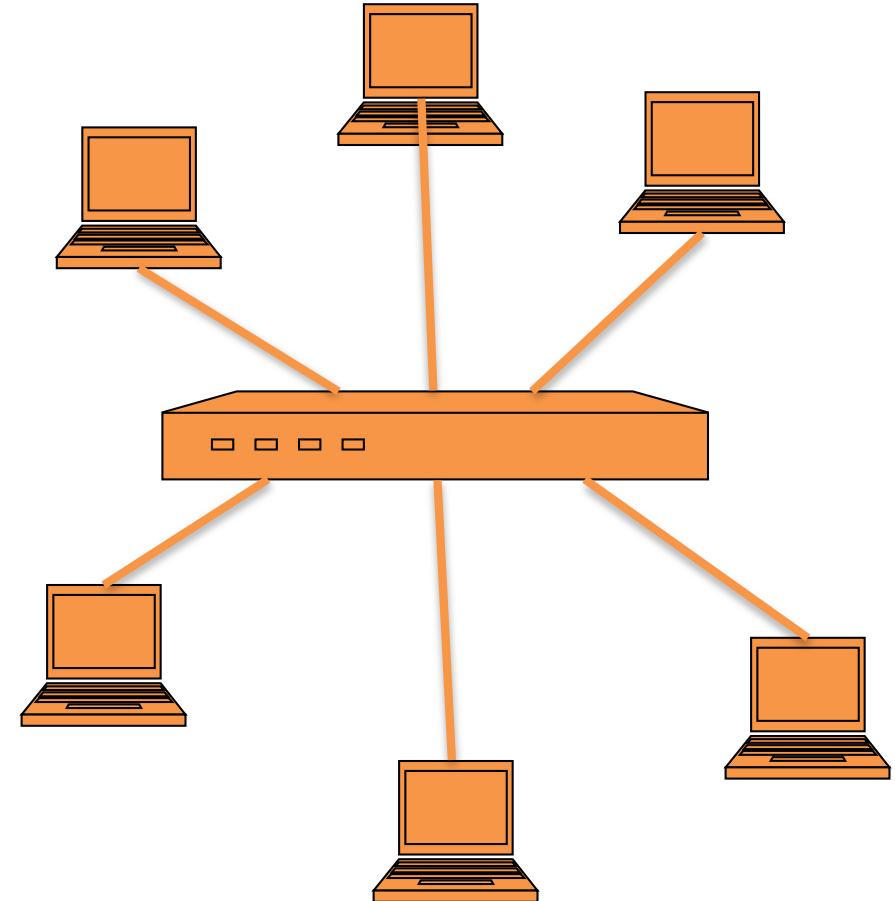
- ❖ unit of data transferred by different link protocols over different links:
    - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
  - ❖ each link protocol provides different services
    - e.g., may or may not provide rdt over link
- transportation analogy:*
- ❖ trip from Princeton to Lausanne
    - limo: Princeton to JFK
    - plane: JFK to Geneva
    - train: Geneva to Lausanne
  - ❖ tourist = **datagram**
  - ❖ transport segment = **communication link**
  - ❖ transportation mode = **link layer protocol**
  - ❖ travel agent = **routing algorithm**

# MAC Addresses

- Most network interfaces come with a predefined MAC address
- A MAC address is a 48-bit number usually represented in hex
  - E.g., 00-1A-92-D4-BF-86
- The first three octets of any MAC address are IEEE-assigned Organizationally Unique Identifiers
  - E.g., Cisco 00-1A-A1, D-Link 00-1B-11, ASUSTek 00-1A-92
- The next three can be assigned by organizations as they please, with uniqueness being the only constraint
- Organizations can utilize MAC addresses to identify computers on their network
- MAC address can be reconfigured by network interface driver software

# Switch

- A **switch** is a common network device
  - Operates at the link layer
  - Has multiple ports, each connected to a computer
- Operation of a switch
  - Learn the MAC address of each computer connected to it
  - Forward frames only to the destination computer



# Link layer services

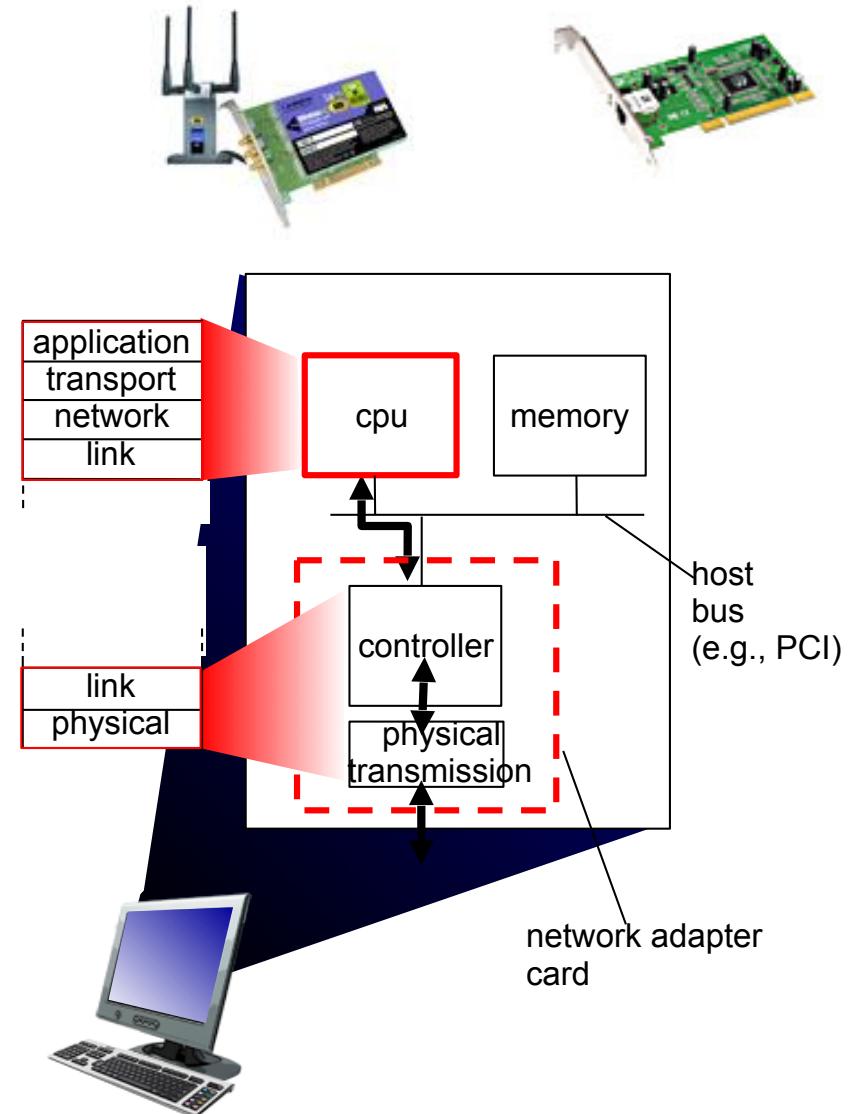
- ❖ *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - “MAC” addresses used in frame headers to identify source, dest
    - different from IP address!
- ❖ *reliable delivery between adjacent nodes*
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - Q: why both link-level and end-end reliability?

# Link layer services (more)

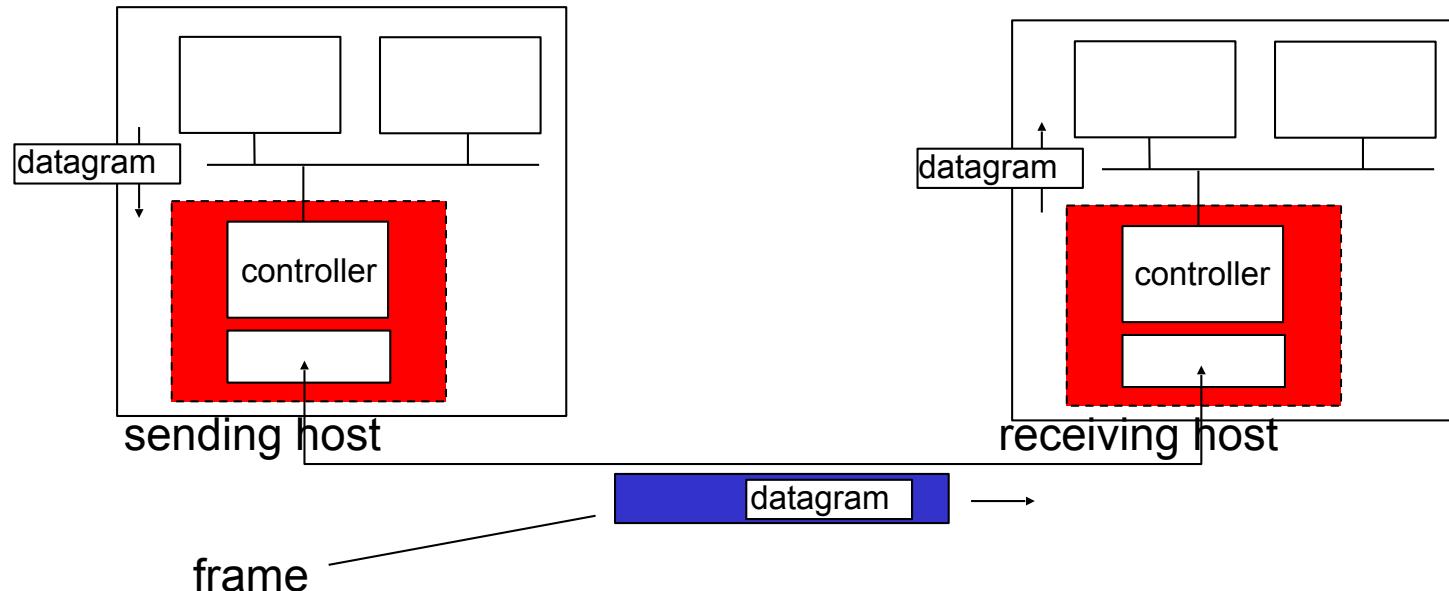
- ❖ **flow control:**
  - pacing between adjacent sending and receiving nodes
- ❖ **error detection:**
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- ❖ **error correction:**
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- ❖ **half-duplex and full-duplex**
  - with half duplex, nodes at both ends of link can transmit, but not at same time

# Where is the link layer implemented?

- ❖ in each and every host
- ❖ link layer implemented in “adaptor” (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- ❖ attaches into host’s system buses
- ❖ combination of hardware, software, firmware



# Adaptors communicating



- ❖ **sending side:**
  - encapsulates datagram in frame
  - adds error checking bits, reliability, flow control, etc.
- ❖ **receiving side**
  - looks for errors, reliability, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

# Multiple access links, protocols

two types of “links”:

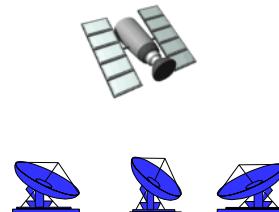
- ❖ **point-to-point**
  - PPP for ISP access
  - point-to-point link between Ethernet switch, host
- ❖ ***broadcast (shared wire or medium)***
  - old-fashioned Ethernet
  - upstream HFC
  - 802.11 wireless LAN



shared wire (e.g.,  
cabled Ethernet)



shared RF  
(e.g., 802.11 WiFi)



shared RF  
(satellite)



humans at a  
cocktail party  
(shared air, acoustical)

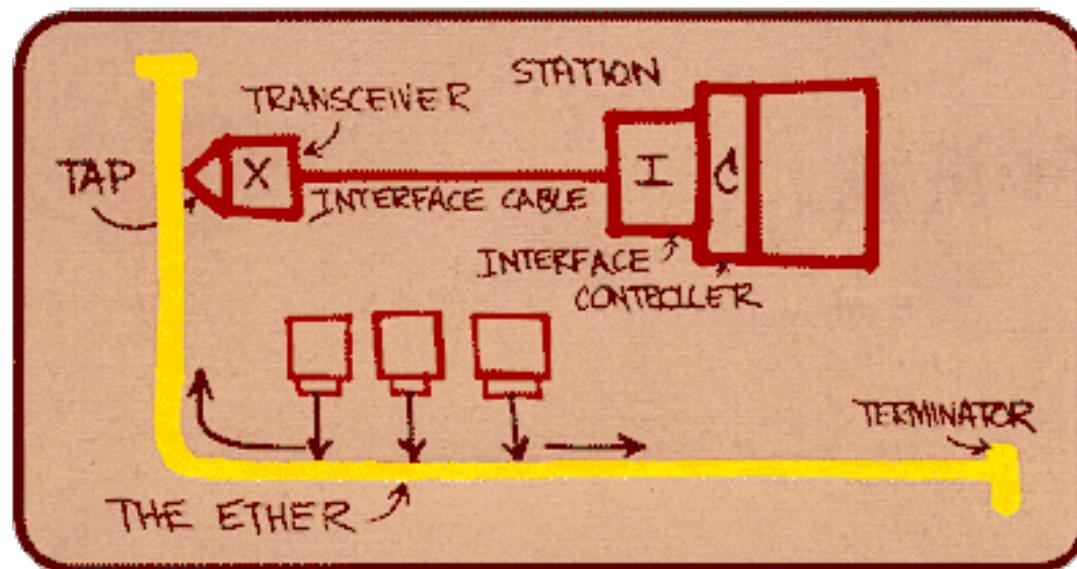
# ARP protocol: same LAN

- ❖ A wants to send IP 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*

# Ethernet

“dominant” wired LAN technology:

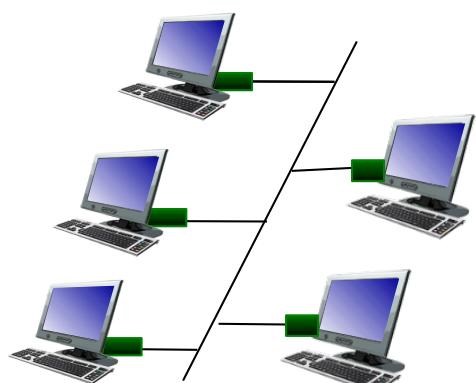
- ❖ cheap (~\$20 for NIC)
- ❖ first widely used LAN technology
- ❖ simpler, cheaper than token LANs and ATM
- ❖ kept up with speed race: 10 Mbps – 10 Gbps



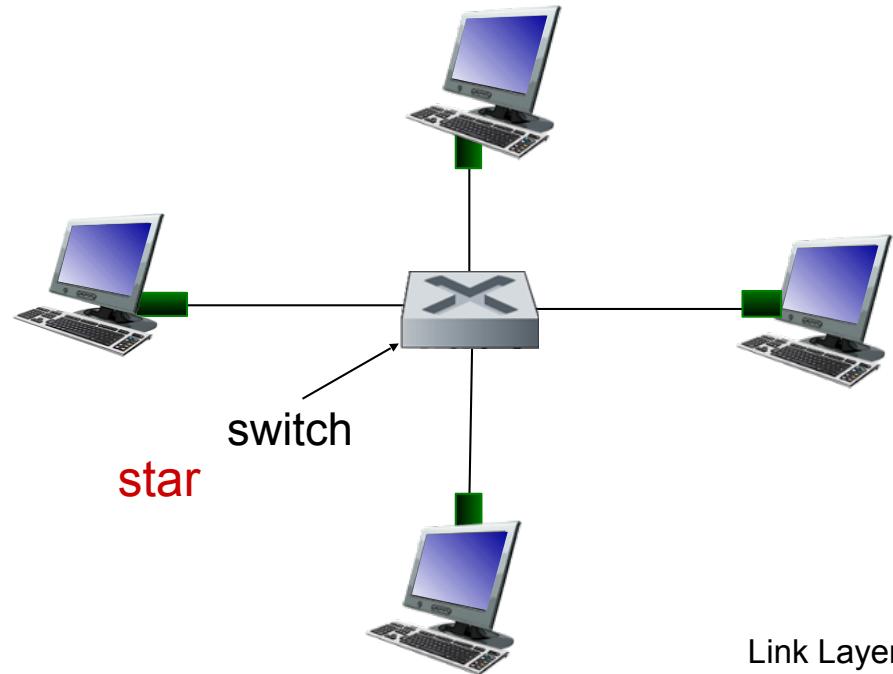
Metcalfe's Ethernet sketch

# Ethernet: physical topology

- ❖ **bus:** popular through mid 90s
  - all nodes in same collision domain (packets can collide with each other)
- ❖ **star:** prevails today
  - active **switch** in center
  - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)



bus: coaxial cable

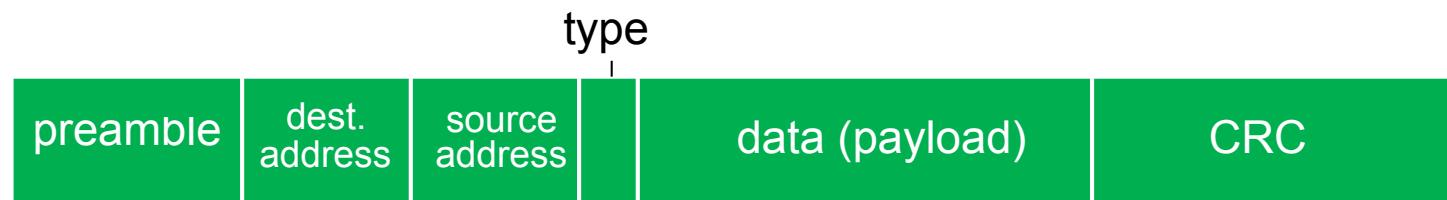


Link Layer

# Ethernet frame structure

---

sending adapter encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**



## *preamble:*

- ❖ 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- ❖ used to synchronize receiver, sender clock rates

# Ethernet frame structure (more)

- ❖ **addresses:** 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- ❖ **type:** indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- ❖ **CRC:** cyclic redundancy check at receiver
  - error detected: frame is dropped



# Ethernet: unreliable, connectionless

- ❖ ***connectionless***: no handshaking between sending and receiving NICs
- ❖ ***unreliable***: receiving NIC doesn't send ACKs or NAKs to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer reliability functionality (e.g., TCP), otherwise dropped data lost
- ❖ Ethernet's MAC protocol: unslotted ***CSMA/CD with binary backoff***