

# Redes de Computadores

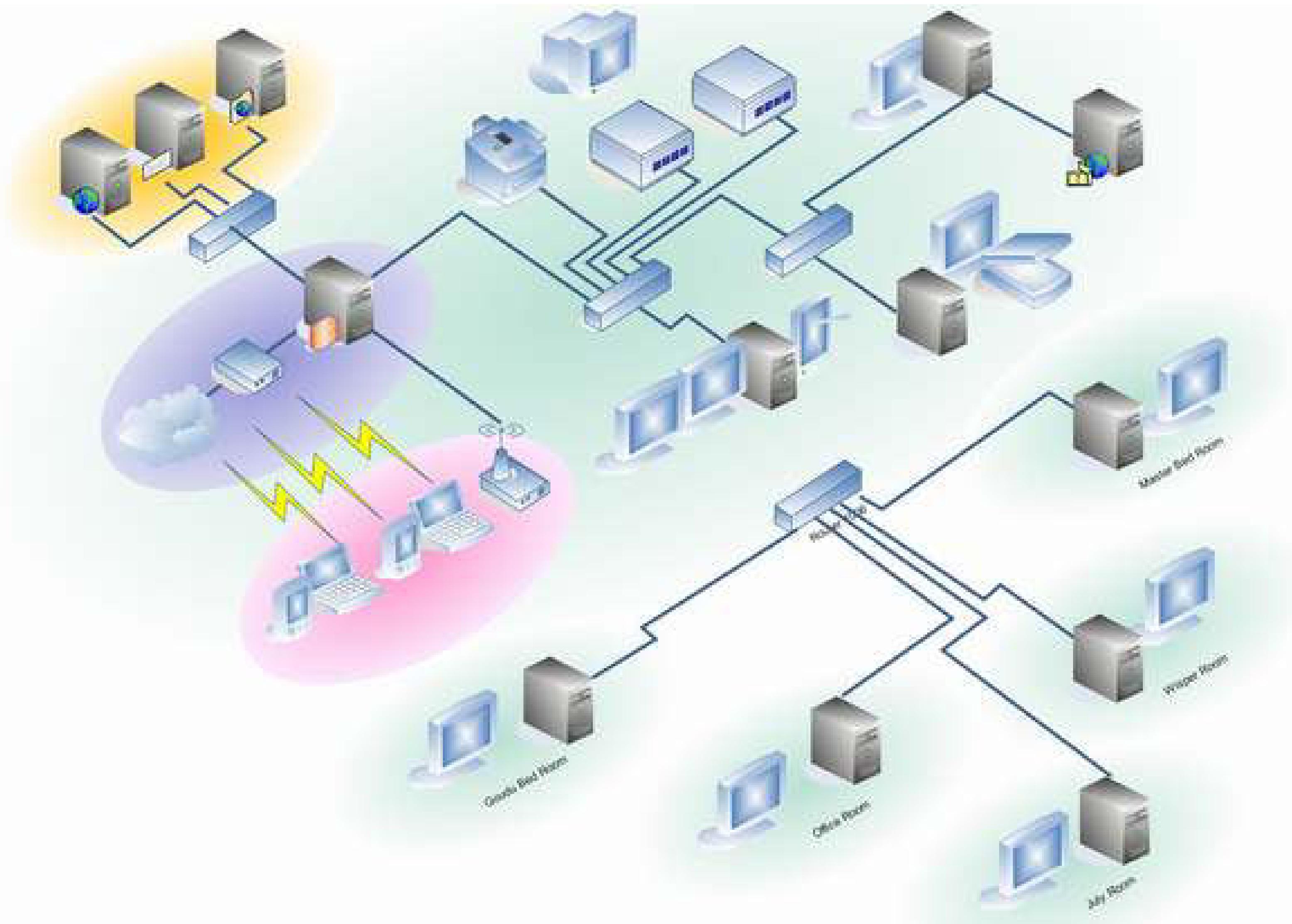
## LEIC-A

*Presentation*

**Prof. Paulo Lobato Correia**  
*IST, DEEC – Área Científica de Telecomunicações*

# Objectives

- Presentation of “Redes de Computadores”
  - Context
  - Brief communications history
  - Computer networks: an Internet-based approach
- Plan the semester:
  - Classes
  - Problems
  - Labs
  - Exams
  - Bibliography



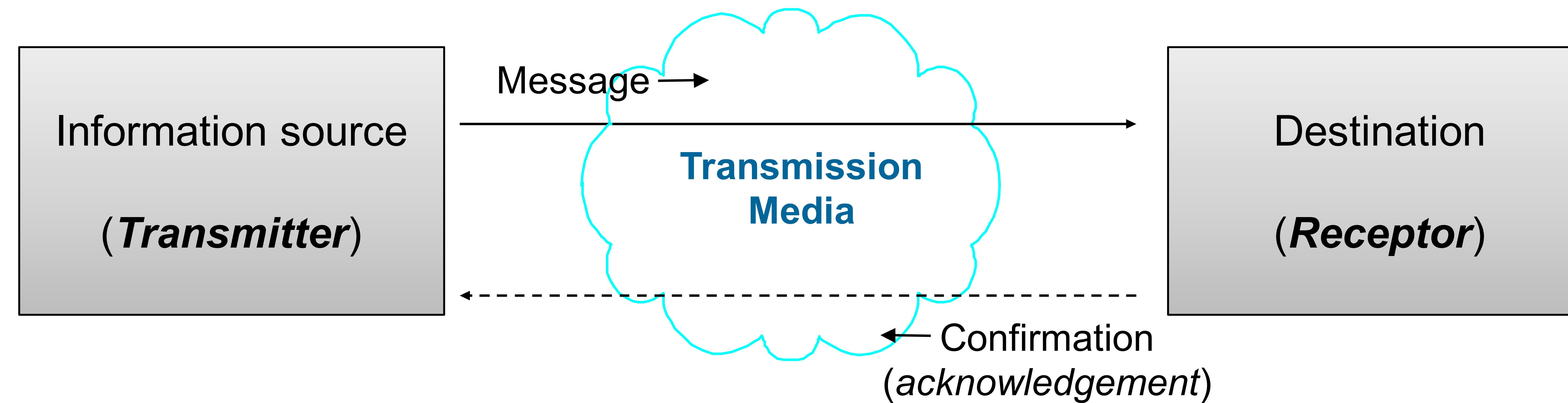
## Telecommunications Applications

- Personal communications (telephone, mobile, zoom, ...);
- Broadcast (radio, TV);
- *World Wide Web*;
- Social networks;
- Electronic mail;
- File transfer;
- ATM (*multibanco*);
- Electronic commerce;
- Remote work;
- Games;
- Measurement, monitoring and remote control (industrial processes, security, intelligent home);
- ...



## Basic Communications System Model

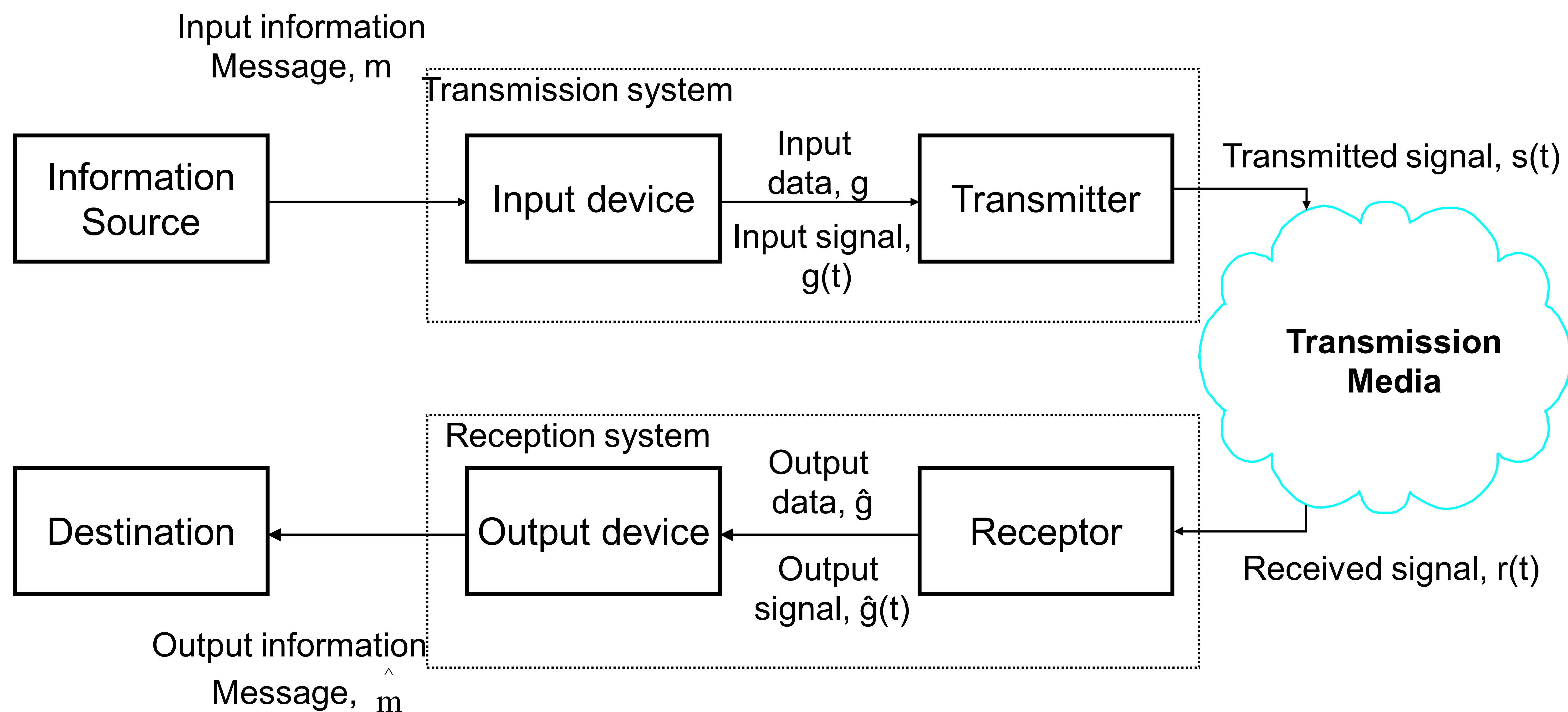
Computer networks are telecommunication networks interconnecting devices that transmit information among them.



Let's make it work. Imagine you want to establish a link between Monsanto and Cristo Rei:

- What type of **signal to transmit** ?
- How to **represent a message** using those signals?
- What **syntax and semantics** for computing device messages?

## *Communications System Model*



# Telecommunications: some dates

## Up to the XIX Century

Smoke signs; drums, etc...

1837 – **Telegraph**;

1844 – **Morse code** (binary communication);

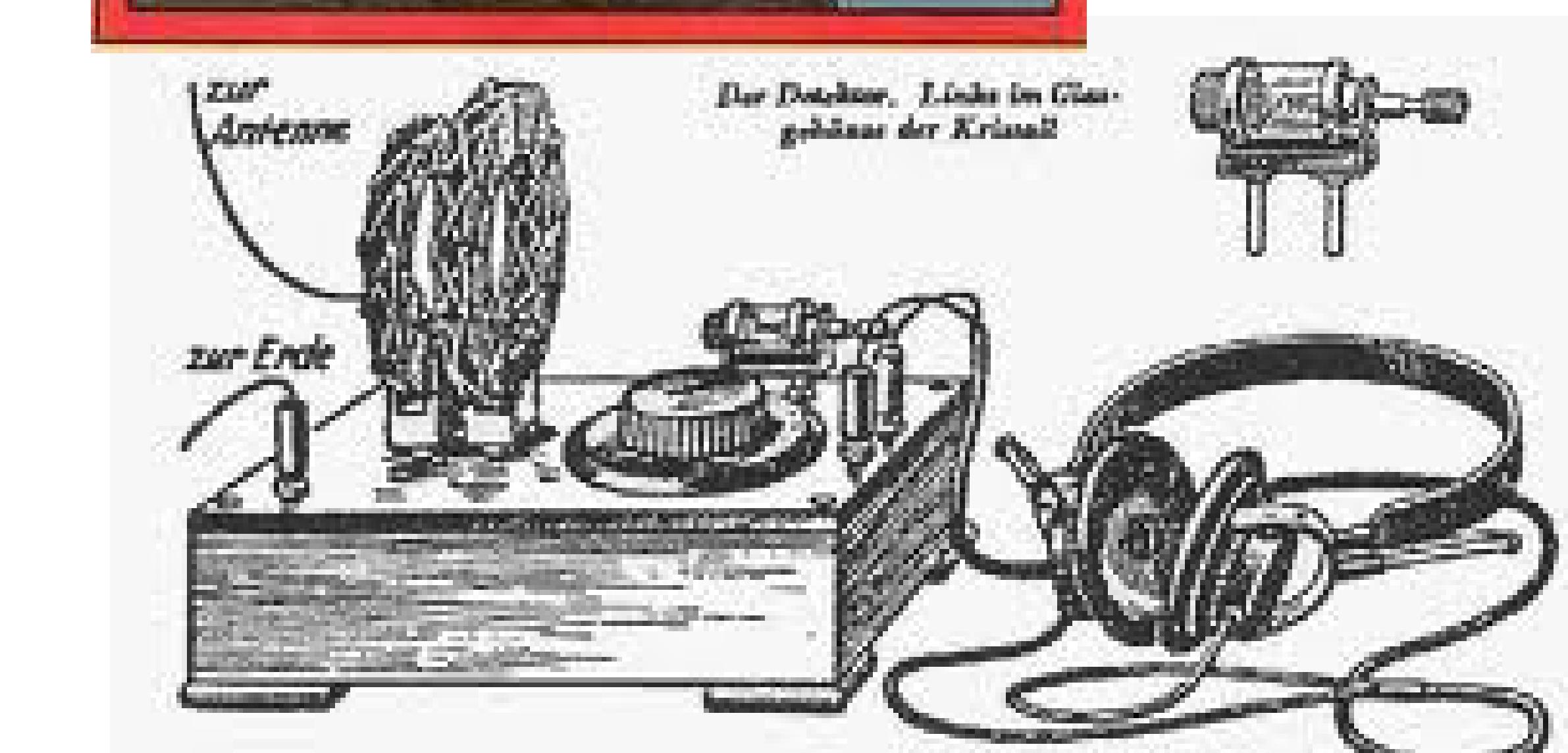
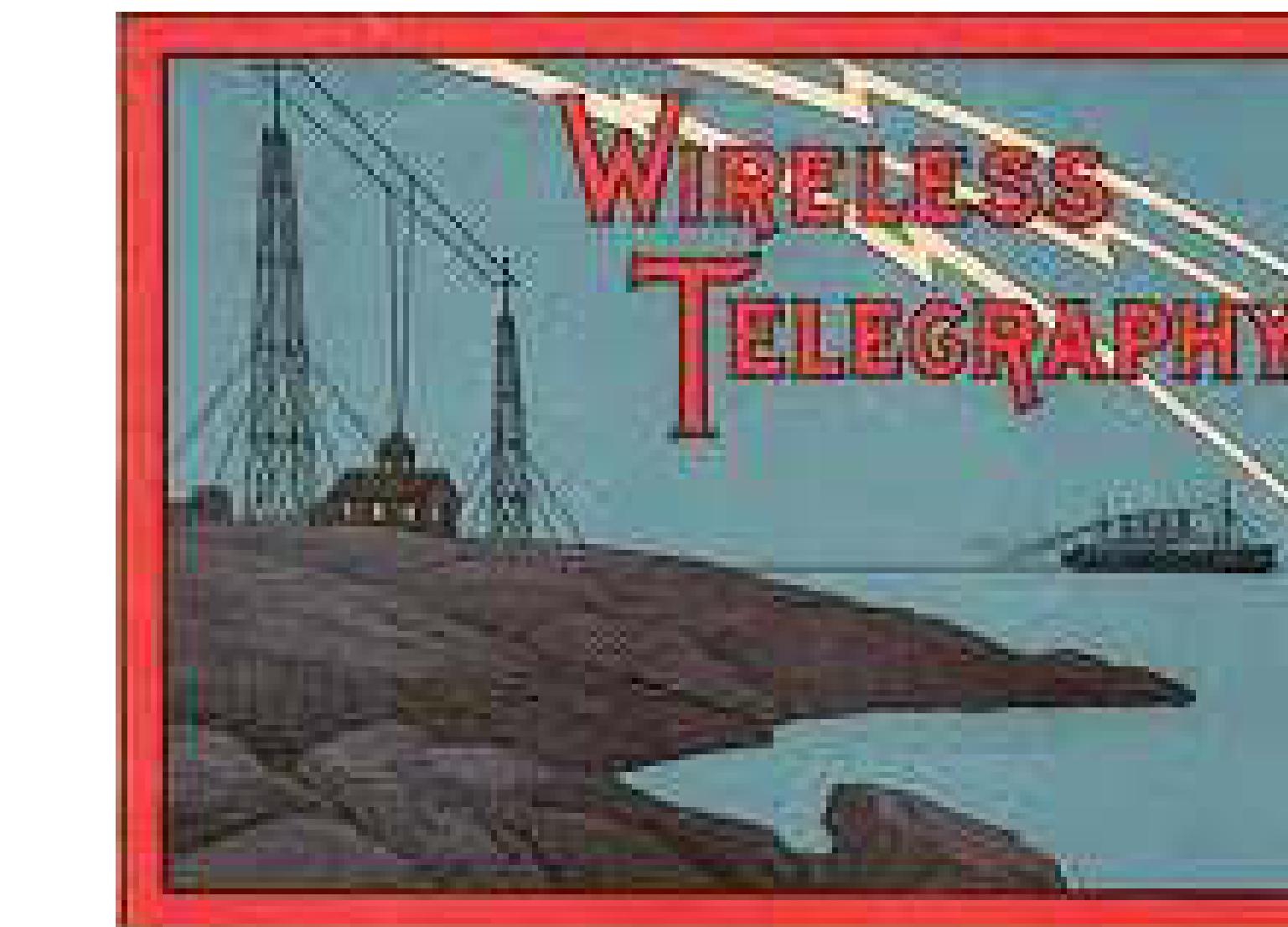
1866 – 1<sup>st</sup> transatlantic submarine cable;

1875 – 1<sup>st</sup> Lisboa-Brasil submarine cable;

1876 – **Telephone** (Bell);

1891 – Automatic switching (Strowger);

1894 – **Wireless telegraphy** (Marconi);



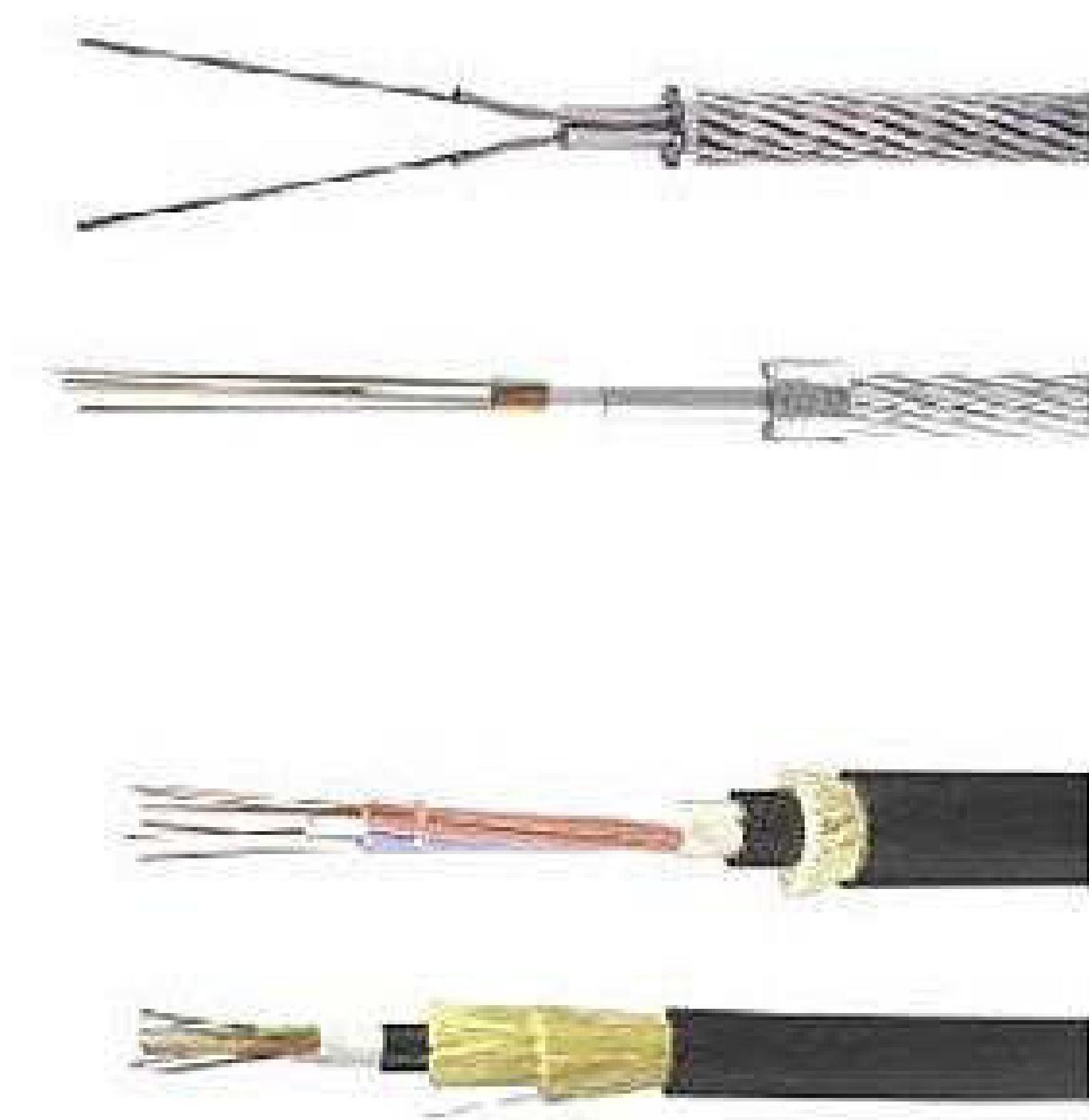
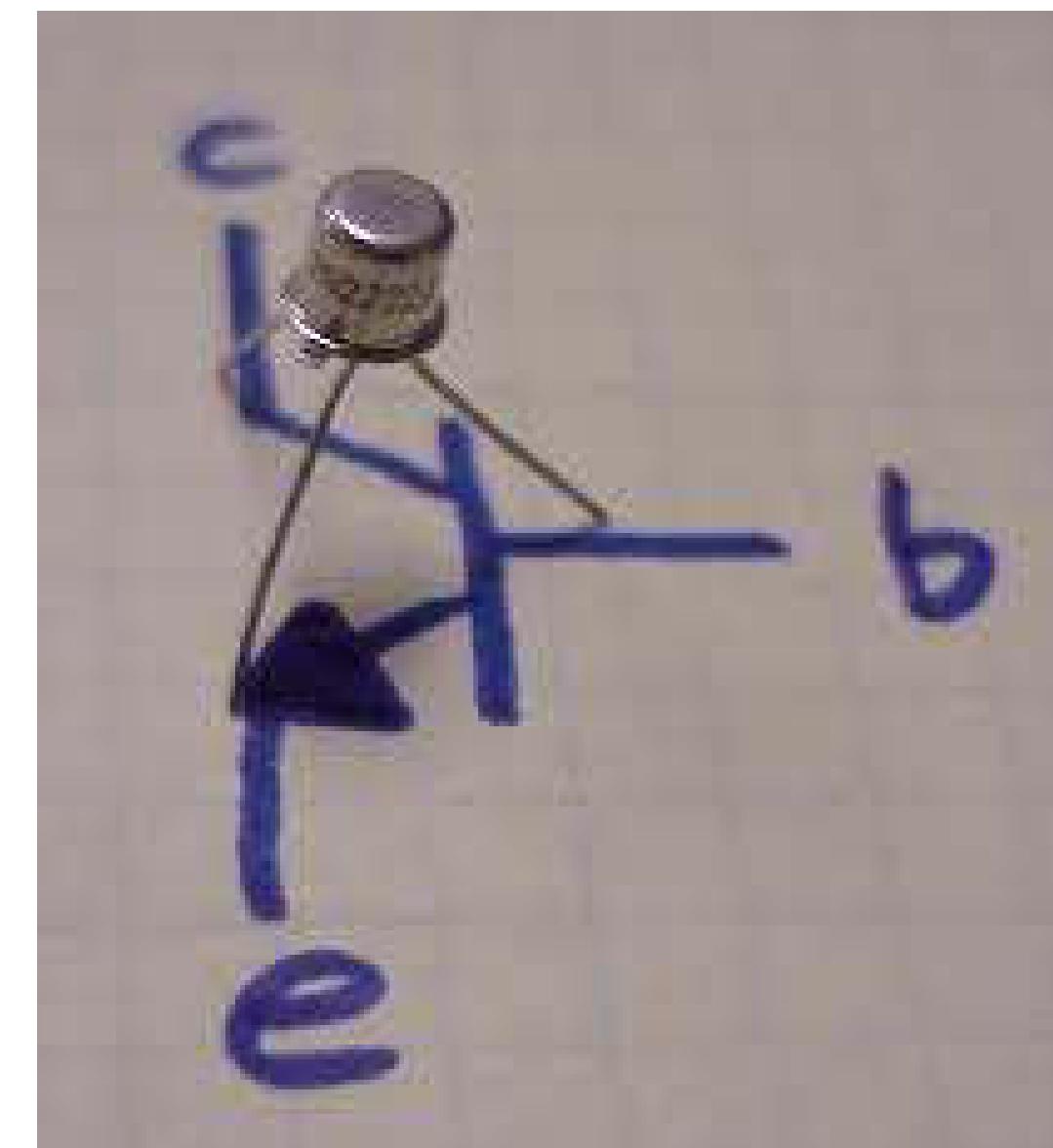
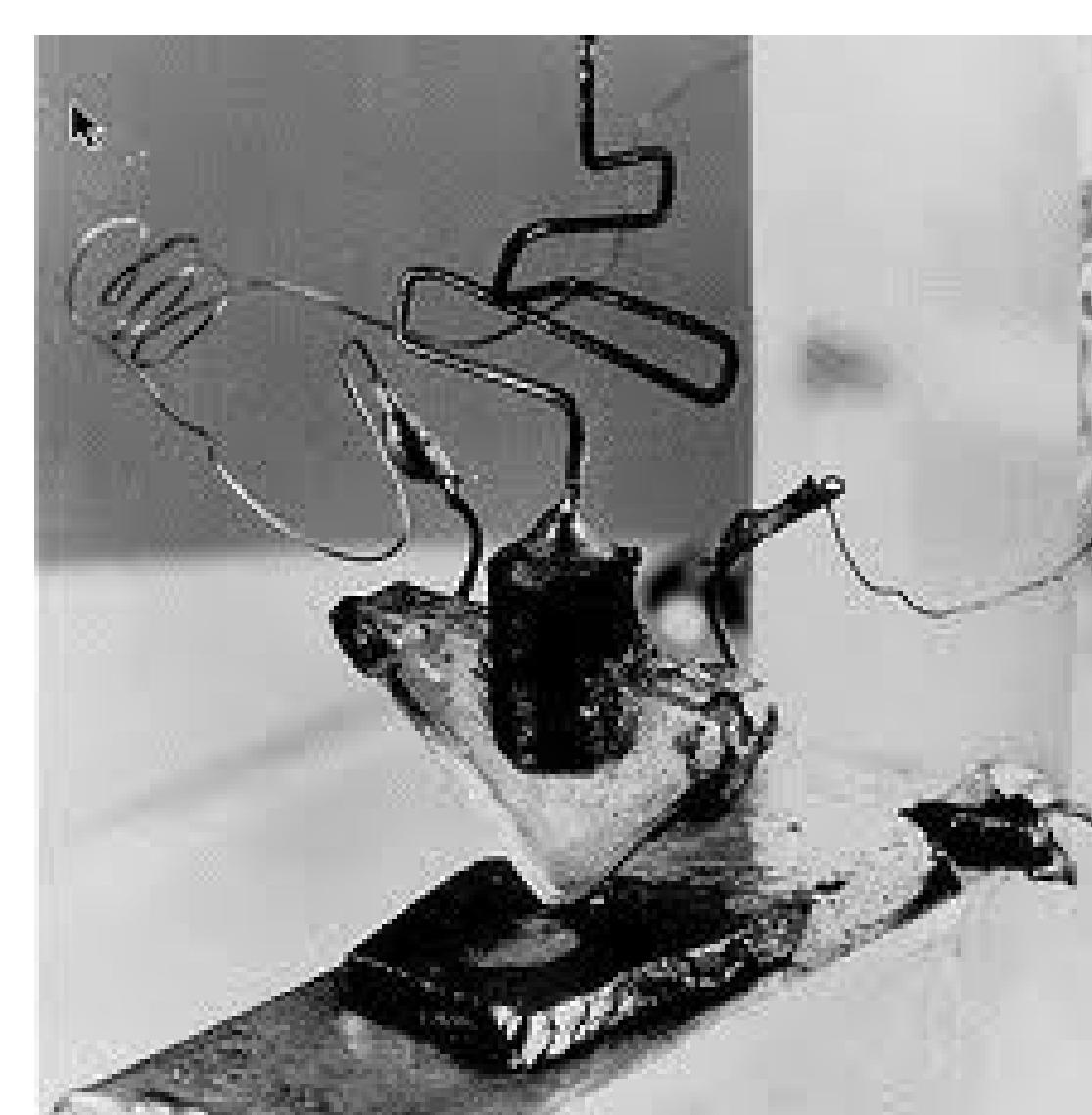
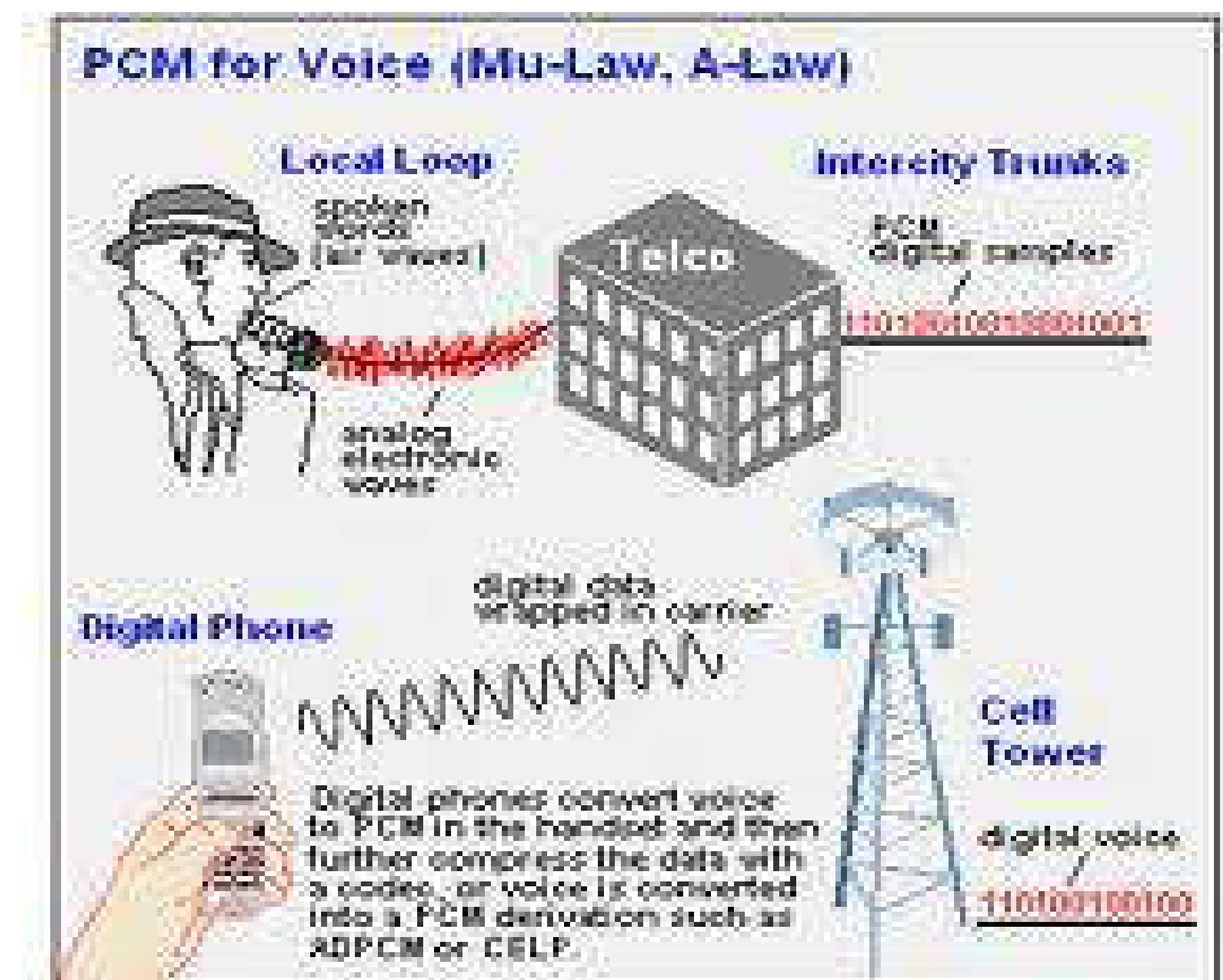
Sugestão: <https://www.ulisboa.pt/patrimonio/museu-faraday>

# Telecommunications: some dates

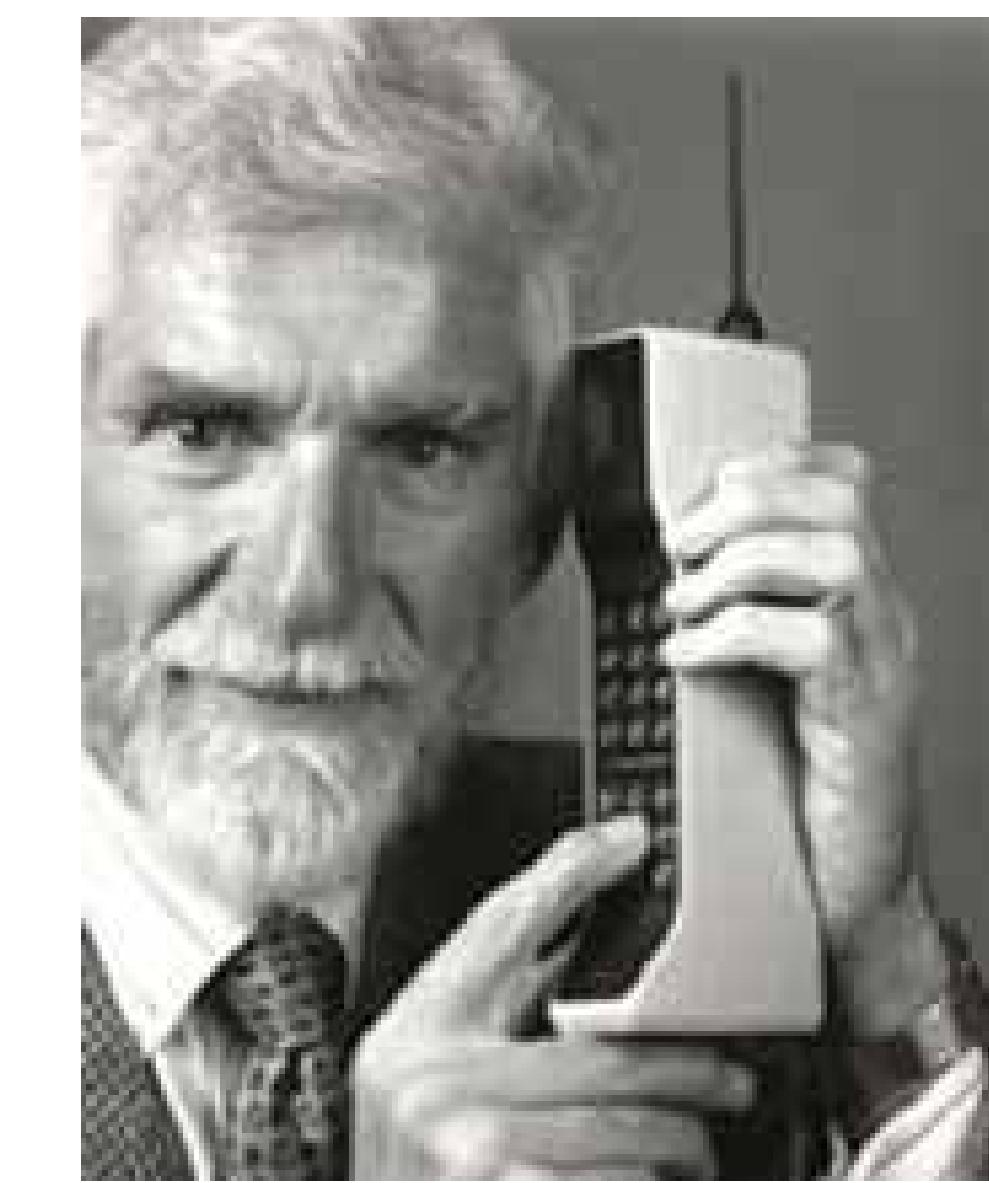
From Computer Desktop Encyclopedia  
© 2005 The Computer Language Co., Inc.

## XX Century: ±1<sup>st</sup> half

- 1928 – Sampling theory (Nyquist);
- 1936 – **PCM** (Reeves) – **digital** transmission;
- 1947 – **Transistor**;
- 1956 – 1<sup>st</sup> telephonic submarine cable (73 repeaters, 35 circuits);
- 1962 – 1<sup>st</sup> telecommunications **satellite** (12 circuits);
- 1966 – Proposal to use **optical fibre**;



# Telecommunications: some dates



## XX Century:

1967 – 1<sup>st</sup> **packet switching** network project (ARPAnet);

1973 – **Ethernet** (Metcalfe);

1977 – ISO starts developing **OSI model**;

1978 – 1<sup>st</sup> **analogue mobile cellular radio** system;

1981 – **TCP/IP**;

1982 – **Electronic mail**;

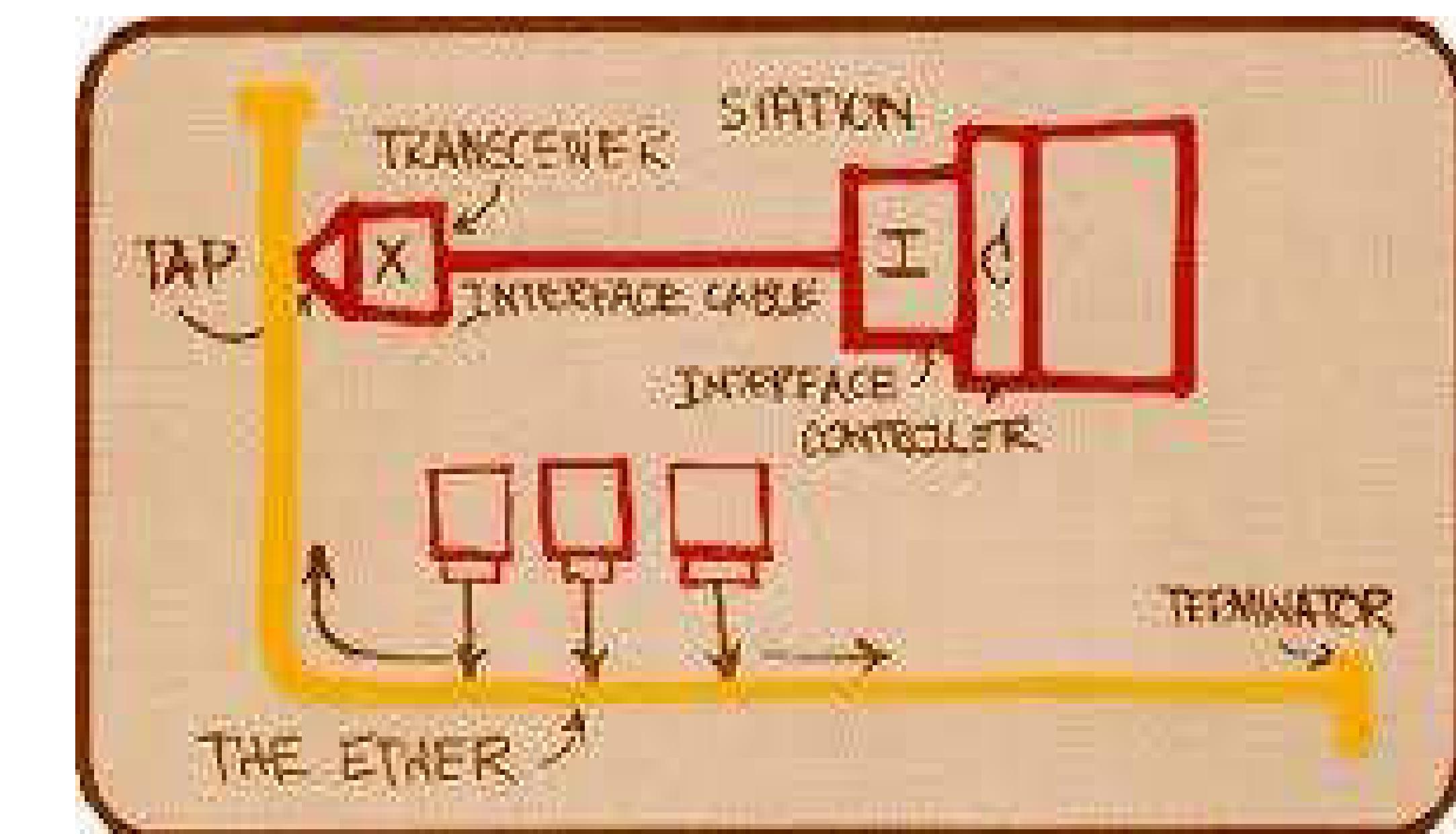
1984 – **IST**: VAX-11/780 (RAM: 4MB, disco: 1GB) with 80 terminals.

1990's – **World Wide Web**;

1991 – **GSM**;

1997 – Intelsat VIII Satellite (22 500 circuits);

1999 – Optical submarine cable TAT14/15 (40 Gbit/s,  $\sim 10^6$  circ.);



# Telecommunications

## XXI Century:

- Internet is becoming ubiquitous;

WORLD INTERNET USAGE AND POPULATION STATISTICS 2023 Year Estimates							
	World Regions	Population ( 2022 Est.)	Population % of World	Internet Users 31 Dec 2021	Penetration Rate (% Pop.)	Growth 2000-2023	Internet World %
North America	<a href="#">Africa</a>	1,394,588,547	17.6 %	601,940,784	43.2 %	13,233 %	11.2 %
Europe	<a href="#">Asia</a>	4,352,169,960	54.9 %	2,916,890,209	67.0 %	2,452 %	54.2 %
Latin America / Caribbean	<a href="#">Europe</a>	837,472,045	10.6 %	747,214,734	89.2 %	611 %	13.9 %
Middle East	<a href="#">Latin America / Carib.</a>	664,099,841	8.4 %	534,526,057	80.5 %	2,858 %	9.9 %
Oceania	<a href="#">North America</a>	372,555,585	4.7 %	347,916,694	93.4 %	222 %	6.5 %
World, Avg.	<a href="#">Middle East</a>	268,302,801	3.4 %	206,760,743	77.1 %	6,194 %	3.8 %
Asia	<a href="#">Oceania / Australia</a>	43,602,955	0.5 %	30,549,185	70.1 %	301 %	0.6 %
Africa	<a href="#">WORLD TOTAL</a>	7,932,791,734	100.0 %	5,385,798,406	67.9 %	1,392 %	100.0 %
NOTES: (1) Internet Usage and World Population Statistics estimates are for June 30, 2022. (2) CLICK on each world region name for detailed regional usage information. (3) Demographic (Population) numbers are based on data from the <a href="#">United Nations Population Division</a> . (4) Internet usage information comes from data published by <a href="#">Nielsen Online</a> , by the <a href="#">International Telecommunications Union</a> , by <a href="#">GfK</a> , by local ICT Regulators and other reliable sources. (5) For definitions, navigation help and disclaimers, please refer to the <a href="#">Website Surfing Guide</a> . (6) The information from this website may be cited, giving the due credit to <a href="#">www.internetworldstats.com</a> . Copyright © 2022, Miniwatts Marketing Group. All rights reserved worldwide.							

Source: Internet Penetration Rate and 5,385,798,4 Copyright © 2022

## Early Computers



1984 – IST: VAX-11/780 (RAM: 4MB, disco: 1GB)  
with 80 terminals DEC VT 220

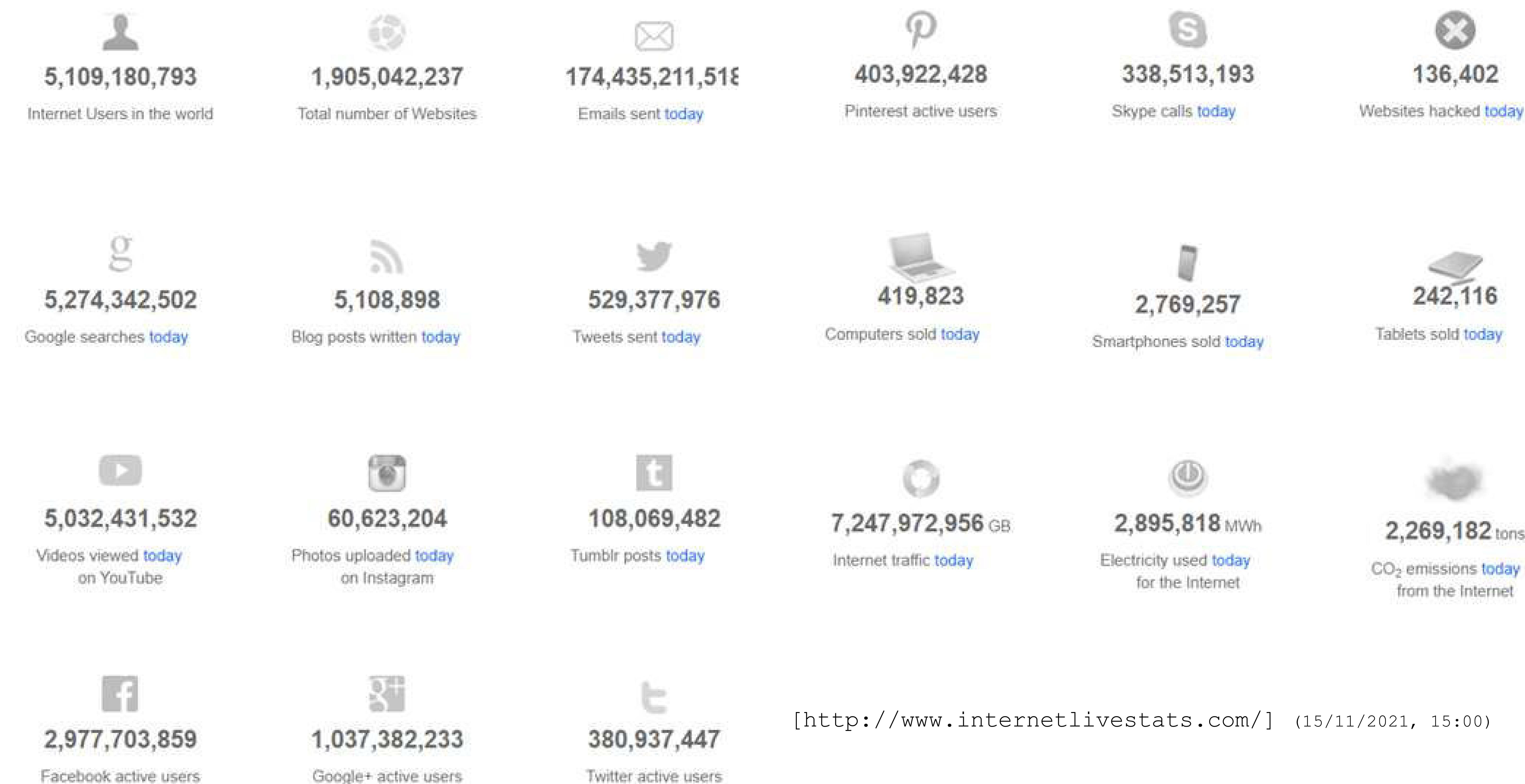
## The Internet



- The Internet is a *computer network* that interconnects millions of computing devices throughout the world.
- The Internet is an *infrastructure that provides services* to applications.
  - The applications are said to be **distributed applications**.

## The Internet

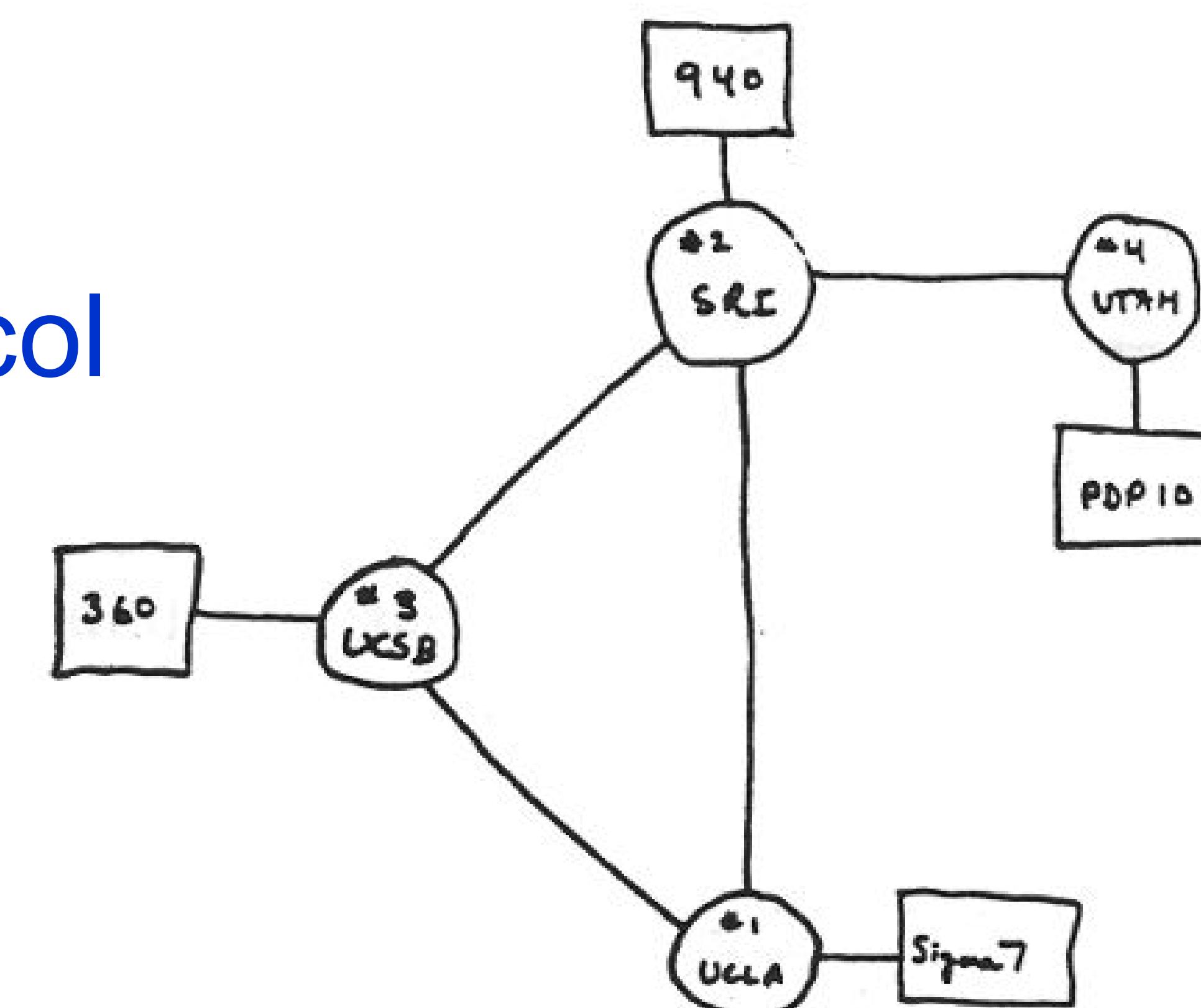
- The Internet is a *computer network* that interconnects millions of computing devices throughout the world.



## Internet History

### 1961-1972: Early packet-switching principles

- 1961: Kleinrock - queueing theory shows effectiveness of **packet-switching**
- 1964: Baran - packet-switching in military nets
- 1967: ARPAnet conceived by Advanced Research Projects Agency
- 1969: first ARPAnet node operational
- 1972:
  - ARPAnet public demonstration
  - NCP (Network Control Protocol) first host-host protocol
  - first e-mail program
  - ARPAnet has 15 nodes



THE ARPA NETWORK

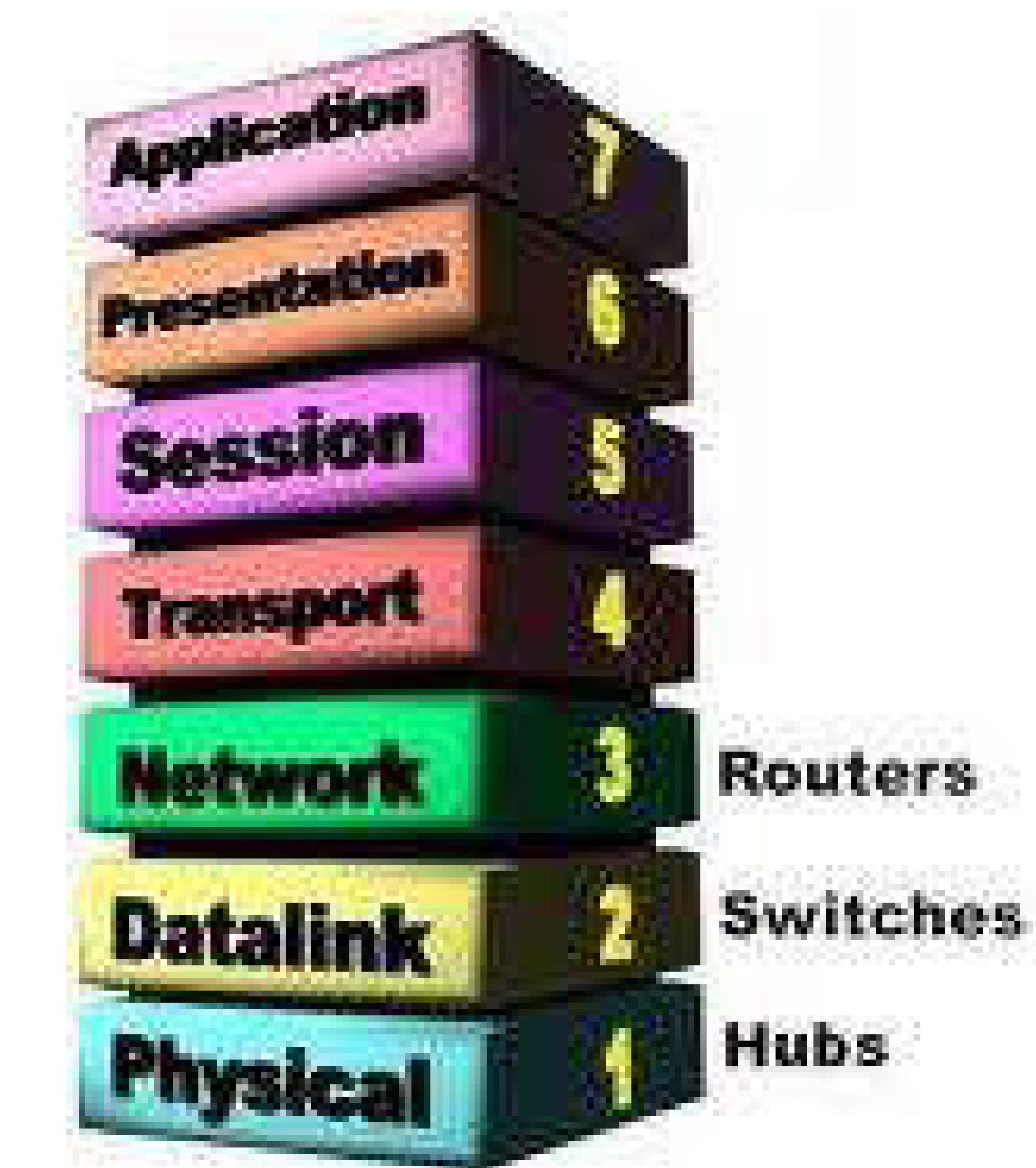
# Internet History

## 1972-1980: Internetworking, new and proprietary nets

- 1970: ALOHAnet satellite network in Hawaii
- 1974: Cerf and Kahn - **architecture for interconnecting networks**
- 1976: Ethernet at Xerox PARC
- late70's: **proprietary architectures**: DECnet, SNA, XNA
- late 70's: switching fixed length packets (ATM precursor)
- 1977: ISO starts developing the **OSI model**
- 1979: ARPAnet reaches 200 nodes

Cerf and Kahn's internetworking principles define today's Internet architecture :

- best-effort service model
- stateless routing
- decentralized control
- minimalism, autonomy - no internal changes required to interconnect networks



## Internet History

**1980-1990: new protocols, a proliferation of networks**

- 1982: **SMTP** e-mail protocol defined
- 1983: deployment of **TCP/IP**
- 1983: **DNS** defined for name-to-IP-address translation
- 1985: **FTP** protocol defined
- 1988: TCP congestion control



## Internet History

### 1990, 2000's: commercialization, the Web, new apps

- Early 1990s: **Web**
  - hypertext [Bush 1945, Nelson 1960's]
  - **HTML, HTTP**: Berners-Lee (1989-)
  - 1994: Mosaic, later Netscape
  - late 1990's: commercialization of the Web



### Late 1990's – 2000's:

- more apps: instant messaging, P2P file sharing
- **network security** to forefront
- est. 50 million host, 100 million+ users
- backbone links running at Gbps

## Internet History

*21<sup>st</sup> Century: more new applications, Internet is “everywhere”*

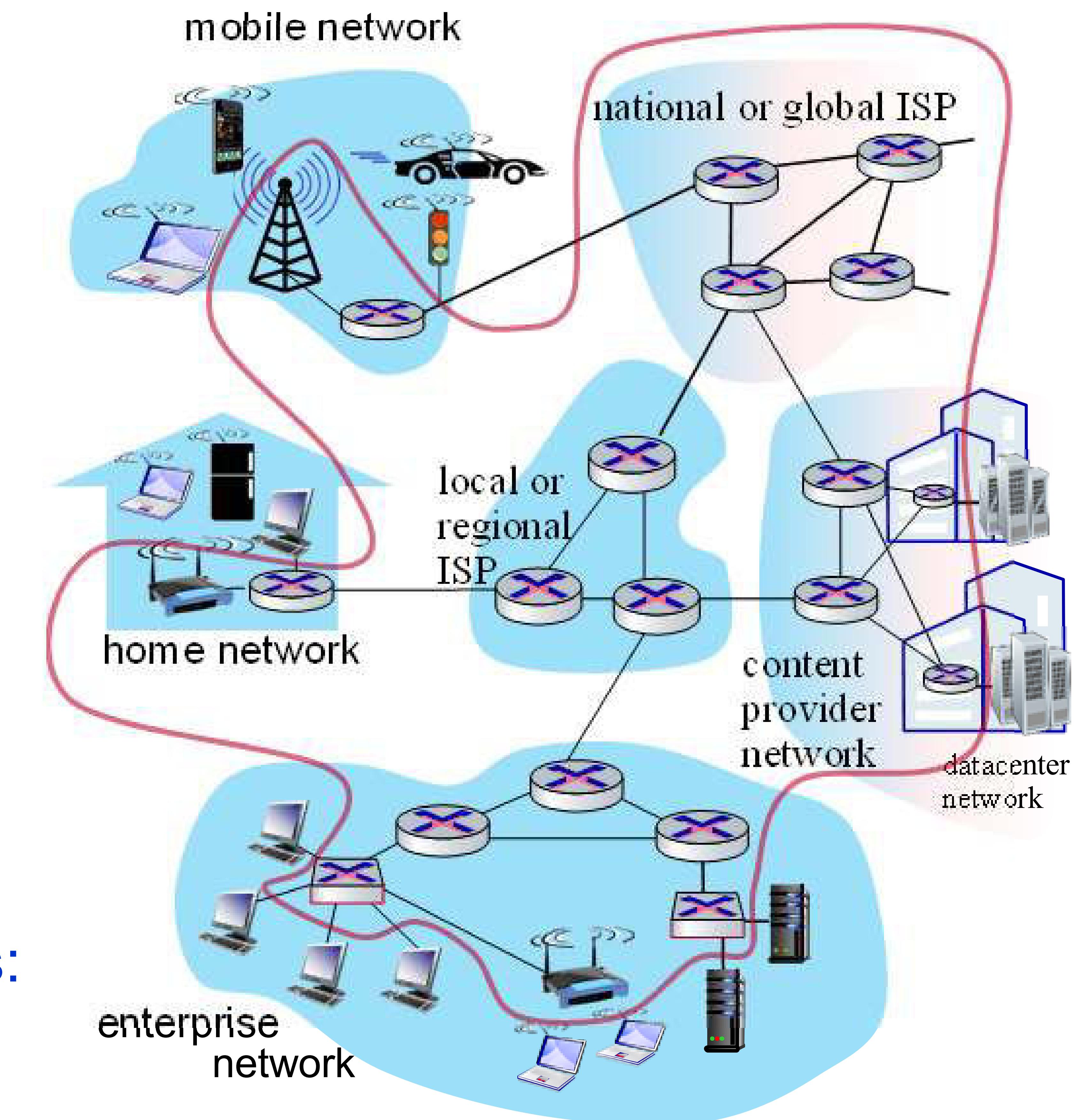
- **Internet everywhere**
  - rise of smartphones, wearables
- **Broadband access**
- High-speed **wireless** access: 4G, 5G, WiFi
- **Social networks:**
  - Facebook: ~ 2.5 billion users
- **Service providers (Google, FB, Microsoft) create their own networks**
  - bypass commercial Internet to connect “close” to end user, providing “instantaneous” access to search, video content, ...
- **Services in the “cloud”**  
(e.g., Google colab, Amazon Web Services, Microsoft Azure, ...)



# What is the Internet?

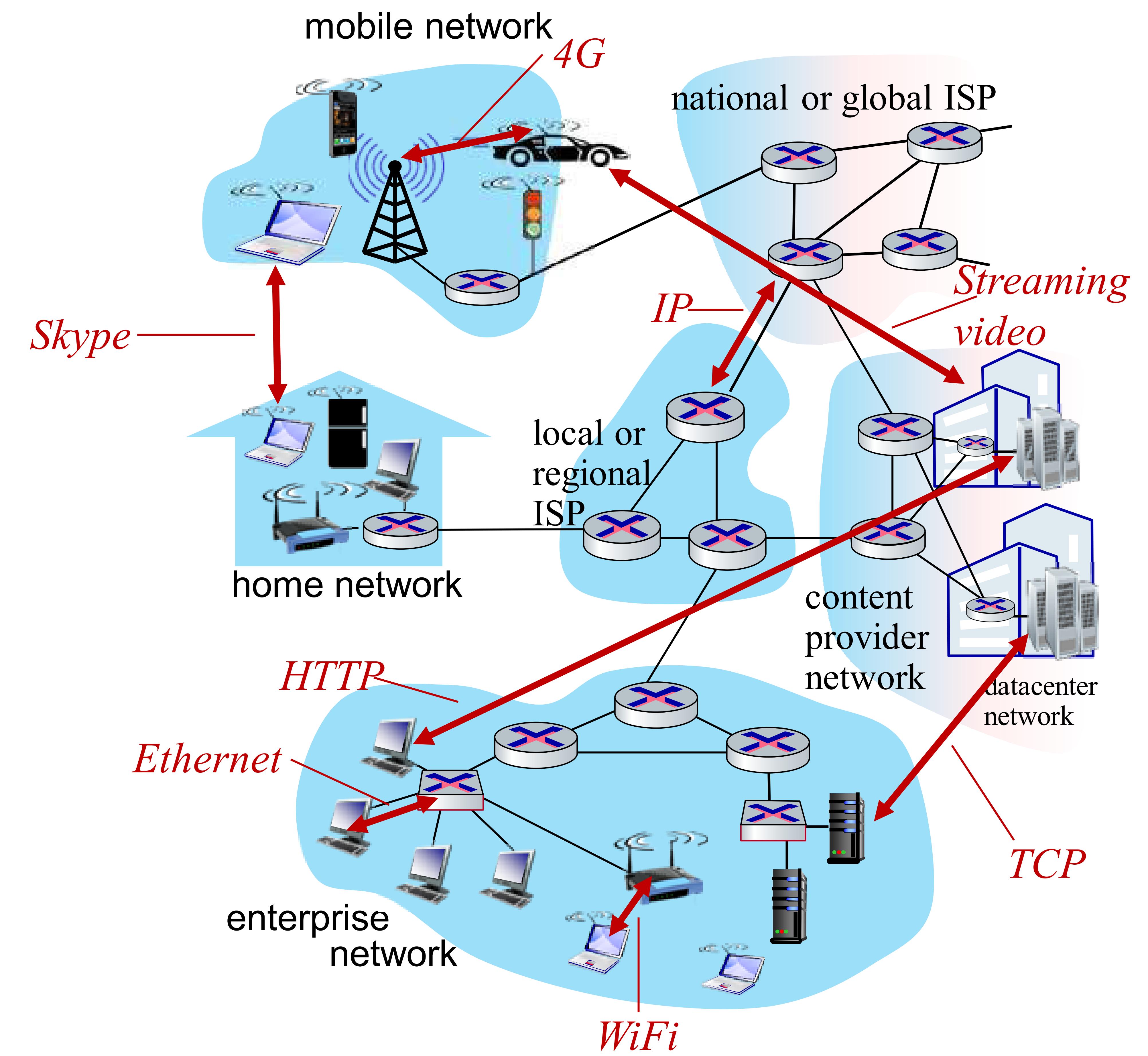
## Components

- Millions of connected devices:  
**hosts** = end systems
  - running *network apps* at *Internet's edge*
- **Router and switches**  
forward packets (chunks of data)
- **Communication links**
  - fiber, copper, radio, satellite
  - transmission rate = *bandwidth*
- **Networks**  
collection of devices, routers, links:  
managed by an organization



## *The Internet Needs some Rules...*

- **Internet:** “network of networks”
  - loosely hierarchical
  - Interconnected ISPs
- **Protocols** control sending and receiving of messages
  - e.g., TCP, IP, HTTP, Ethernet
- **Internet standards**
  - RFC: Request for Comments
  - IETF: Internet Engineering Task Force

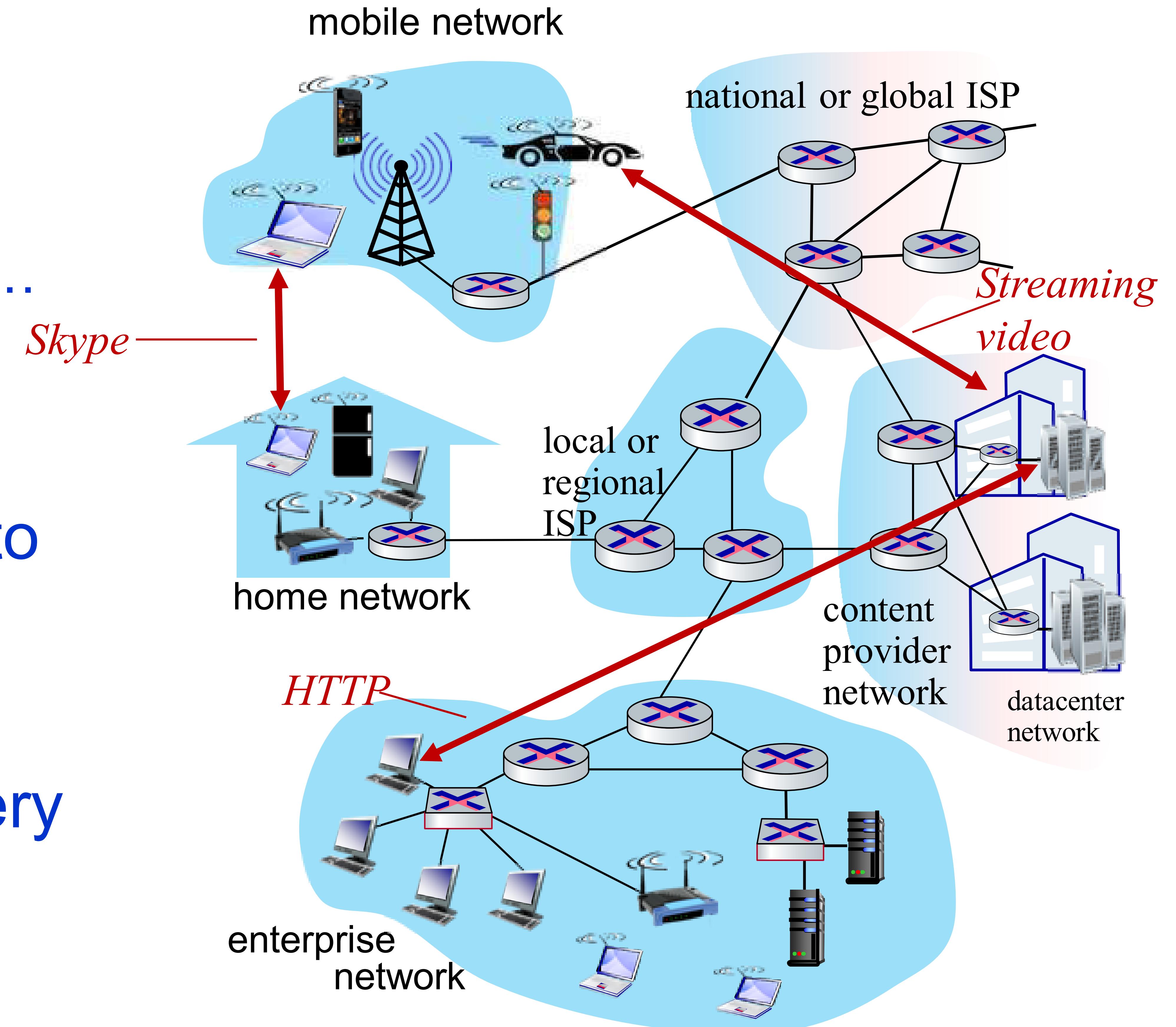


# The Internet: a Service View

- Communication *infrastructure* enables distributed applications:
  - Web, video streaming, multimedia, VoIP, email, social media, games, e-commerce, ...
- Communication *services* provided to applications can be:
  - **Reliable** data delivery from source to destination, or
  - “Best effort” (**unreliable**) data delivery

**TCP**  
7.139.18.103:7229  
 7.129.8.103:7229  
 29.8.103:7229  
 29.18.103:7229

**UDP**



# Redes de Computadores

## Objectives



### Main objectives:

- To be able to critically analyze the options available when designing computer networks, their architectures and protocols.
- Study the main Internet protocols.
- Learn network application programming using the sockets interface.

# Redes de Computadores

## Program

Chapter	Lectures
Presentation	0.5
1 – Introduction	1.5
2 – Application layer	2
3 – Transport layer	3
4 – Network layer	3
5 – Data link layer	3

# Redes de Computadores

## Classes Schedule

### Tóricas:

- RCT01 (Seg. 08:00 - 10:00 – PA1; Qui. 08:00 - 10:00 – VA3)
- RCT02 (Seg. 10:00 - 12:00 – PA1; Qui. 10:00 - 12:00 – VA3)

### Laboratório (LT4, LT5 – Torre Norte, piso 4):

- RCL03 ( Seg. 10:00 - 11:30 - LT5 ; Sex. 10:00 - 11:30 - LT5 )
- RCL04 ( Seg. 11:30 - 13:00 - LT5 ; Sex. 12:00 - 13:30 - LT5 )
- RCL05 ( Seg. 13:00 - 14:30 - LT4 ; Sex. 08:30 - 10:00 - LT4 )
- RCL06 ( Sex. 08:30 - 10:00 - LT5 ; Seg. 13:00 - 14:30 - LT5 )
- RCL07 ( Qui. 14:00 - 15:30 - LT5 ; Seg. 14:30 - 16:00 - LT5 )
- RCL08 ( Ter. 08:30 - 10:00 - LT5 ; Qui. 08:30 - 10:00 - LT5 )
- RCL09 ( Ter. 10:00 - 11:30 - LT5 ; Sex. 13:30 - 15:00 - LT5 )
- RCL10 ( Ter. 11:30 - 13:00 - LT5 ; Qui. 10:00 - 11:30 - LT5 )
- RCL11 ( Ter. 12:30 - 14:00 - LT4 ; Qui. 14:30 - 16:00 - LT4 )
- RCL12 ( Qui. 12:30 - 14:00 - LT5 ; Ter. 14:00 - 15:30 - LT5 )
- RCL13 ( Ter. 14:00 - 15:30 - LT4 ; Qui. 13:00 - 14:30 - LT4 )

<https://fenix.tecnico.ulisboa.pt/disciplinas/RC/2023-2024/1-semestre/turnos>

# Redes de Computadores

## Evaluation

### Evaluation:

- Exam – 45%
  - 16 January 2024 (8:00)
  - 29 January 2024 (15:30)
- Lab (groups of 2 students) – 45%
  - Internet socket programming *in C or C++ (due 15<sup>th</sup> December)* – 25%
    - “RC Auction”
  - Wireshark (1) + Mikrotik (3) – 20%
- Weekly quizzes (Fénix) – 10%
  - Based on the lectures of the week
- Final grade > 18/20 → oral exam.



# Redes de Computadores

## Contacts

Prof. Paulo Lobato Correia, [plc@lx.it.pt](mailto:plc@lx.it.pt)



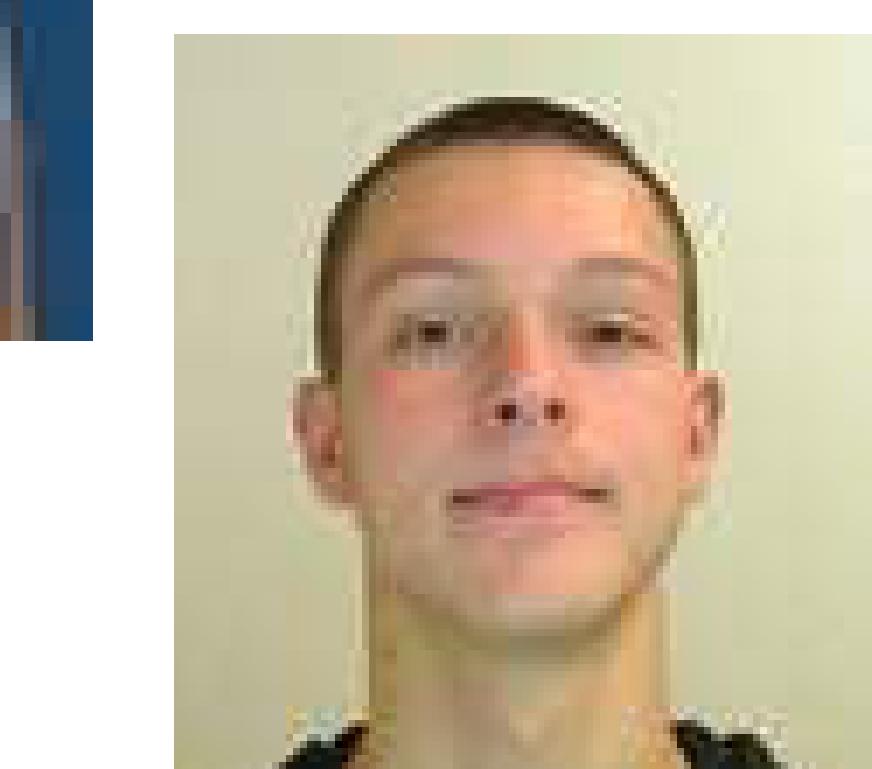
Prof. Francisco Sena da Silva, [sena@lx.it.pt](mailto:sena@lx.it.pt)



Prof. Ayman Radwan,



Inês Pissarra, [ines.pissarra@tecnico.ulisboa.pt](mailto:ines.pissarra@tecnico.ulisboa.pt)



Guilherme Gonçalves, [guilherme.silva.goncalves@tecnico.ulisboa.pt](mailto:guilherme.silva.goncalves@tecnico.ulisboa.pt)

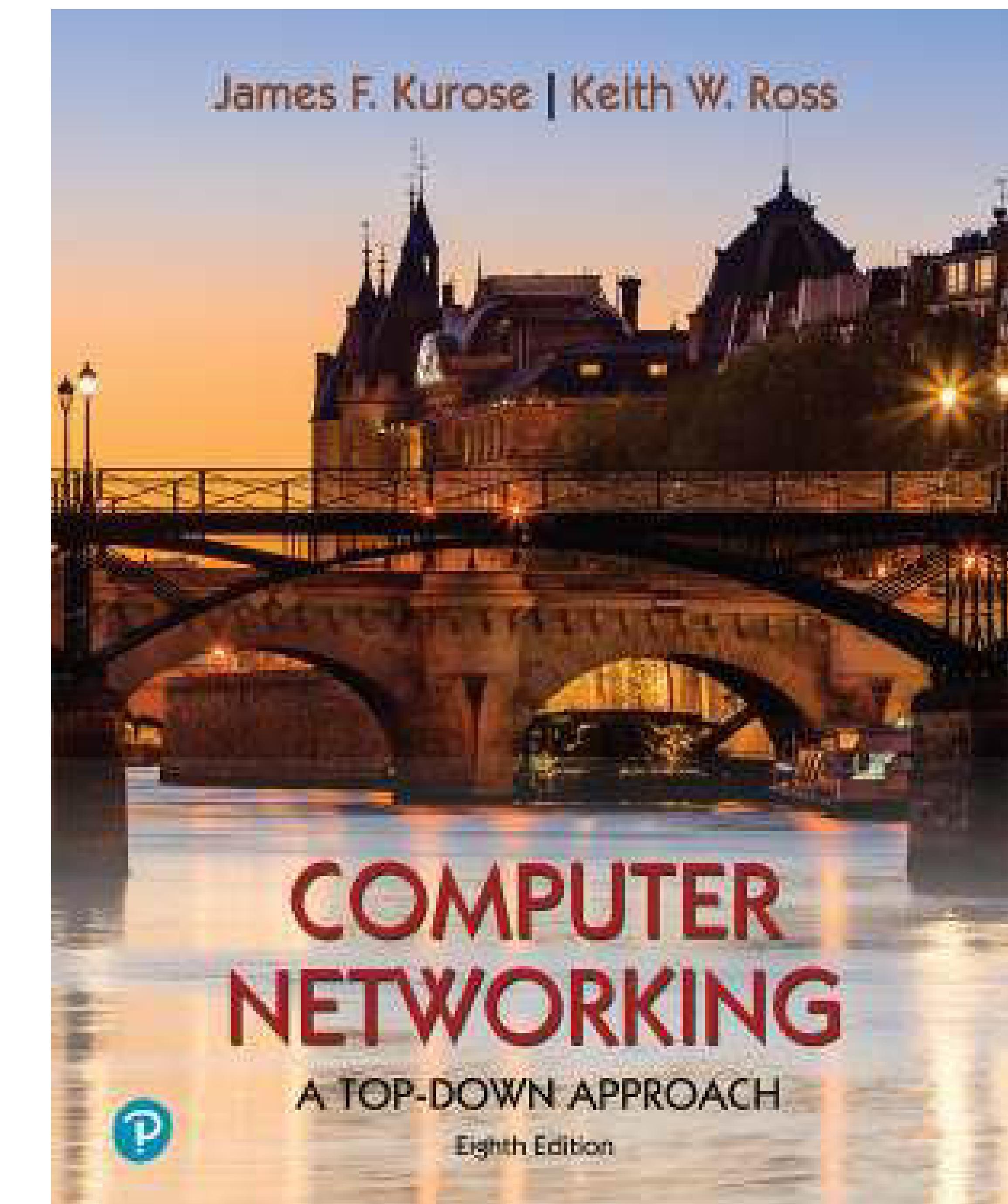
Internet:

<https://fenix.tecnico.ulisboa.pt/disciplinas/RC/2023-2024/1-semestre>

# Redes de Computadores

## Bibliography

- **Computer Networking, a Top-Down Approach**  
**James Kurose, Keith Ross, 2020, 8<sup>th</sup> Edition, Pearson**
- W. Richard Stevens, “Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1)”, 2nd edition, Prentice-Hall PTR , 1998
- Transparências das aulas
- Colecção de problemas
- Enunciados de testes e exames de anos anteriores



# Redes de Computadores

## Bibliography

- Larry Peterson, Bruce Davie, “Computer Networks: A Systems Approach”, 3rd edition, Morgan Kaufmann , 2003
- Michael J. Donahoo, Kenneth L. Calvert, “TCP/IP Sockets in C: Practical Guide for Programmers”, Morgan Kaufmann, 2000
- W. Richard Stevens, “TCP/IP Illustrated, Volume 1: The Protocols”, 1994, Addison-Wesley

# *Unidades*

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
$10^{-3}$	0.001	milli	$10^3$	1,000	kilo
$10^{-6}$	0.000001	micro	$10^6$	1,000,000	Mega
$10^{-9}$	0.000000001	nano	$10^9$	1,000,000,000	Giga
$10^{-12}$	0.00000000001	pico	$10^{12}$	1,000,000,000,000	Tera
$10^{-15}$	0.0000000000001	femto	$10^{15}$	1,000,000,000,000,000	Peta
$10^{-18}$	0.000000000000001	atto	$10^{18}$	1,000,000,000,000,000,000	Exa
$10^{-21}$	0.00000000000000001	zepto	$10^{21}$	1,000,000,000,000,000,000,000	Zetta
$10^{-24}$	0.000000000000000000000001	yocto	$10^{24}$	1,000,000,000,000,000,000,000,000	Yotta

# Novos prefixos binários (IEC 80000-13:2008):

- um kilobyte (1 kB) são 1000 bytes; um kibibyte (1 KiB) são 1024 bytes ( $2^{10}$  bytes).  
Também: mebi (Mi;  $2^{20}$ ), gibi (Gi;  $2^{30}$ ), tebi (Ti;  $2^{40}$ ), pebi (Pi;  $2^{50}$ ), zebi (Zi;  $2^{70}$ ) and yobi (Yi;  $2^{80}$ ).

# Redes de Computadores

## LEIC-A

### ***1 – Introduction***

**Prof. Paulo Lobato Correia**  
*IST, DEEC – Área Científica de Telecomunicações*

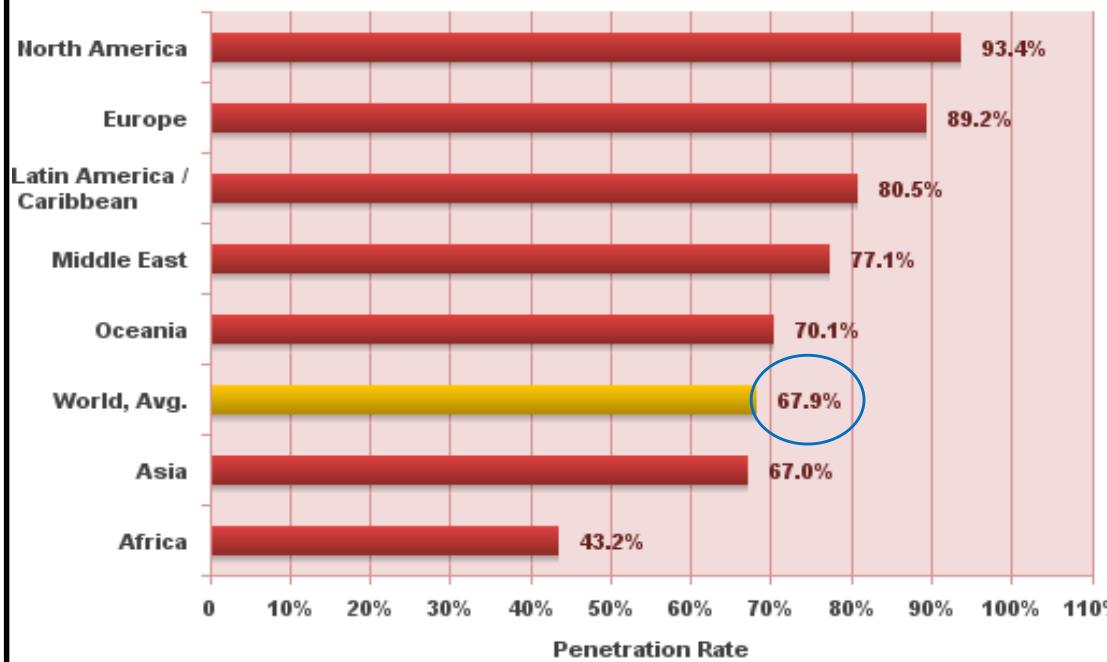
# *Objectives*

- Terminology
- What is a protocol?
- Network edge (hosts, access net, physical media)
- Network core (packet/circuit switching, Internet structure)
- Performance metrics: loss, delay, throughput
- Protocol layers, service models

# The Internet

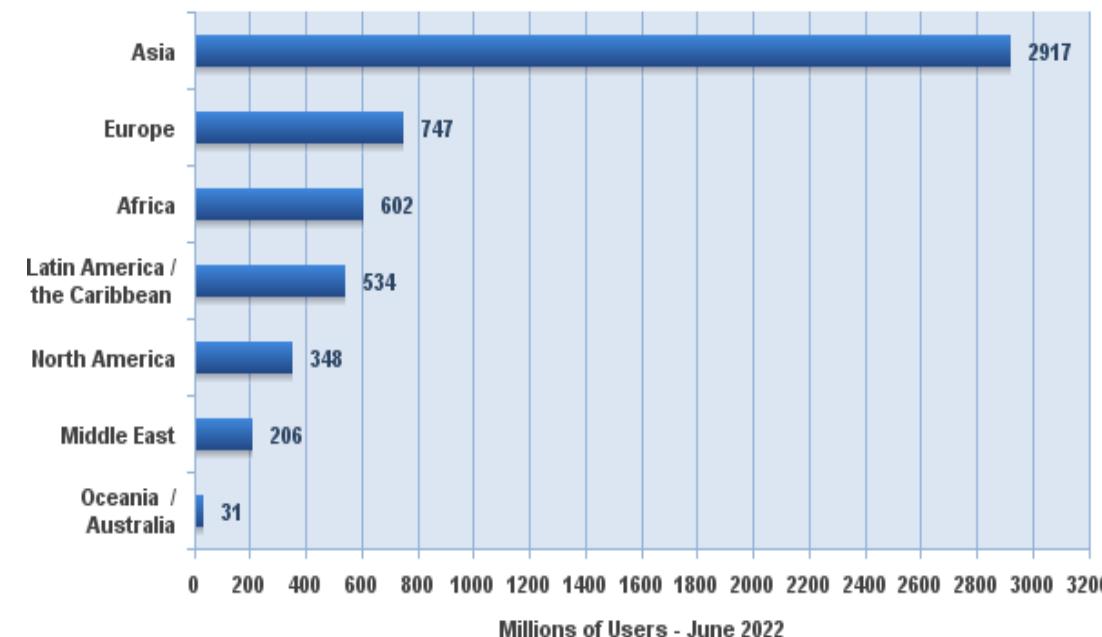
- The Internet is a *computer network* that interconnects millions of computing devices throughout the world.

**Internet World Penetration Rates  
by Geographic Regions - 2022**



Source: Internet World Stats - [www.internetworldstats.com/stats.htm](http://www.internetworldstats.com/stats.htm)  
 Penetration Rates are based on a world population of 7,932,791,734  
 and 5,385,798,406 estimated Internet users in June 30, 2022.  
 Copyright © 2022, Miniwatts Marketing Group

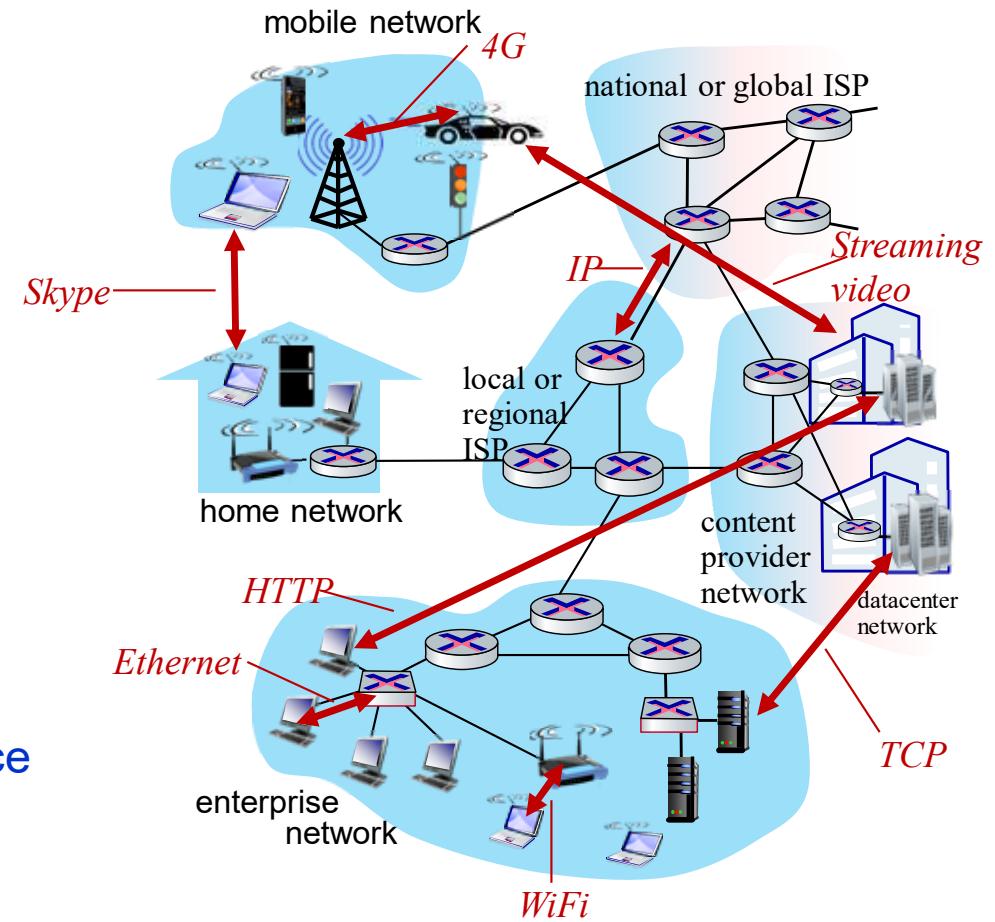
**Internet Users in the World  
by Geographic Regions - 2022**



Source: Internet World Stats - [www.internetworldstats.com/stats.htm](http://www.internetworldstats.com/stats.htm)  
 Basis: 5,385,798,406 Internet users estimated in June 30, 2022  
 Copyright © 2022, Miniwatts Marketing Group

## *The Internet Needs some Rules...*

- *Internet*: “network of networks”
  - loosely hierarchical
  - Interconnected ISPs
- *Protocols* control sending and receiving of messages
  - e.g., TCP, IP, HTTP, Ethernet
- *Internet standards*
  - RFC: Request for Comments
  - IETF: Internet Engineering Task Force



# What is a Protocol?

A protocol defines the **format** and the **order** of messages sent and received among network entities, as well as the **actions** taken on message transmission/receipt.

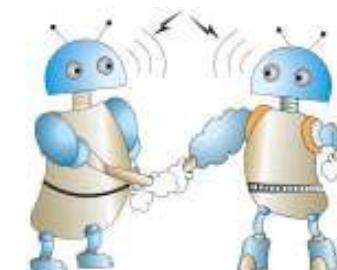
## Protocols followed by humans:

- “What’s the time?”
- “I have a question”
- Introductions
  - Specific messages are sent
  - Specific actions are taken when messages are received, or other events occur



## Network protocols:

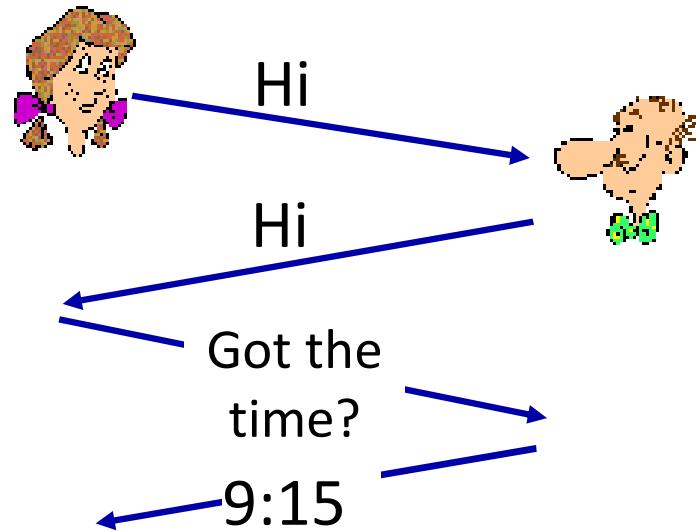
- Machines rather than humans
- Internet communication is governed by protocols



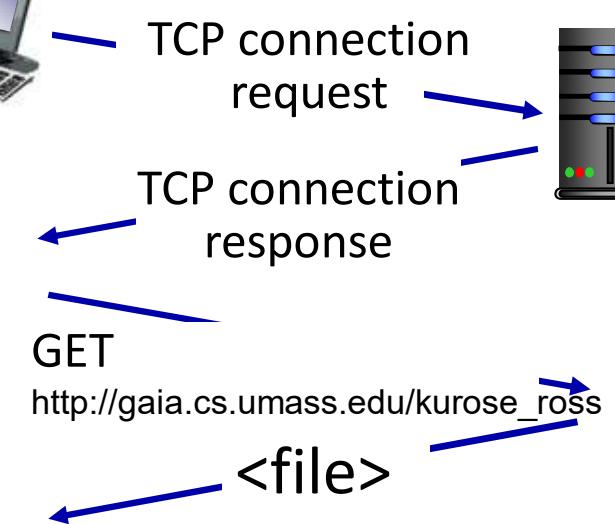
# What is a Protocol?



Human protocol



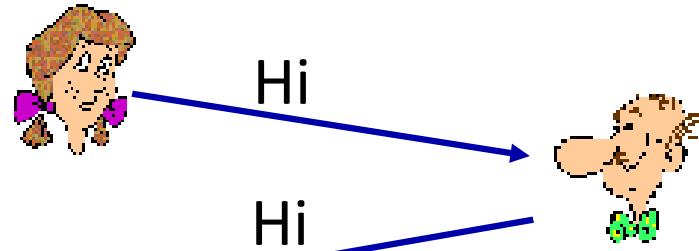
Computer network protocol



# What is a Protocol?



Human protocol



Hi

Hi

Got the  
time?

9:15

time

RC Auction

*login + list*

*login*

LIN 101101 abcdefgh\n

RLI REG 10 8\n

*list*

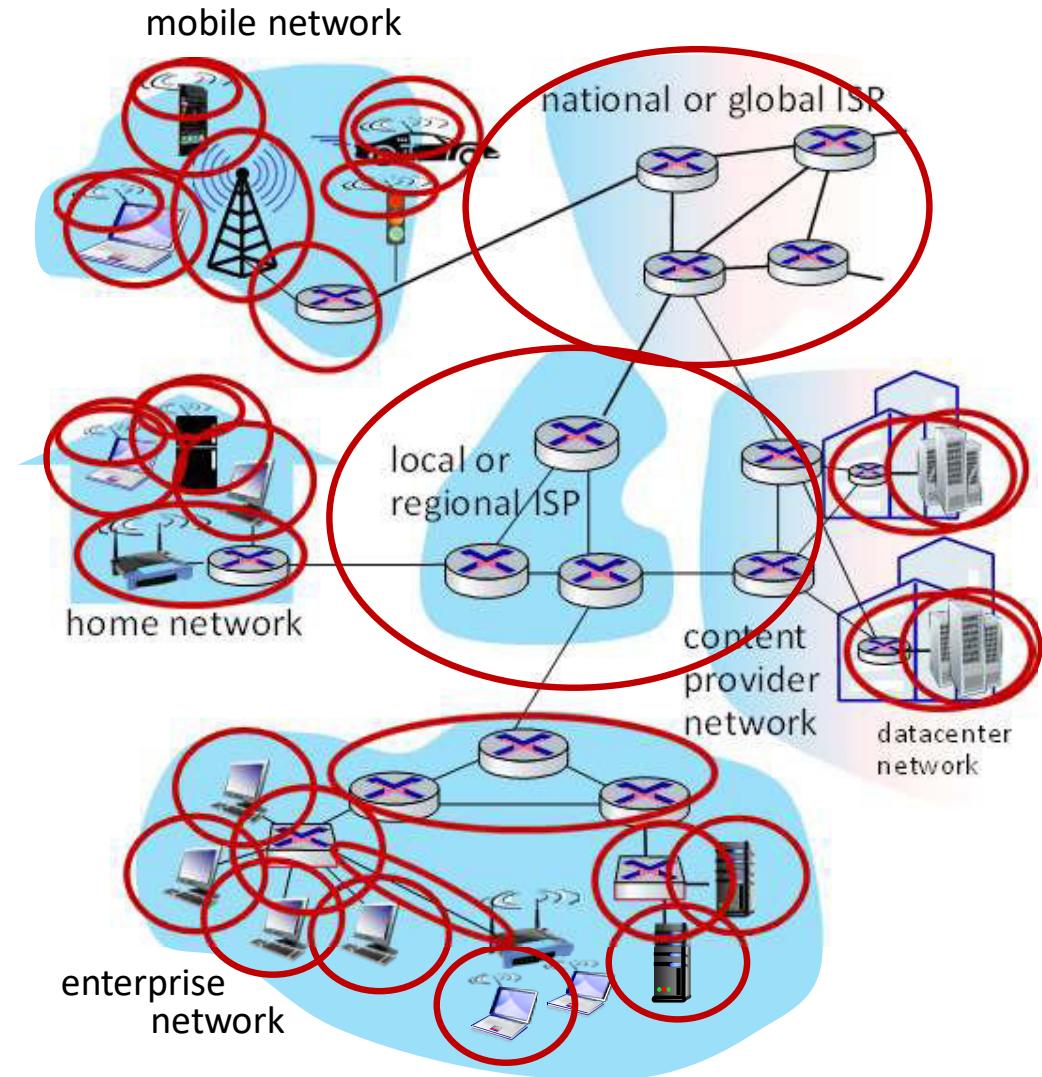
LST\n

RLS OK 078 1 123 1\n



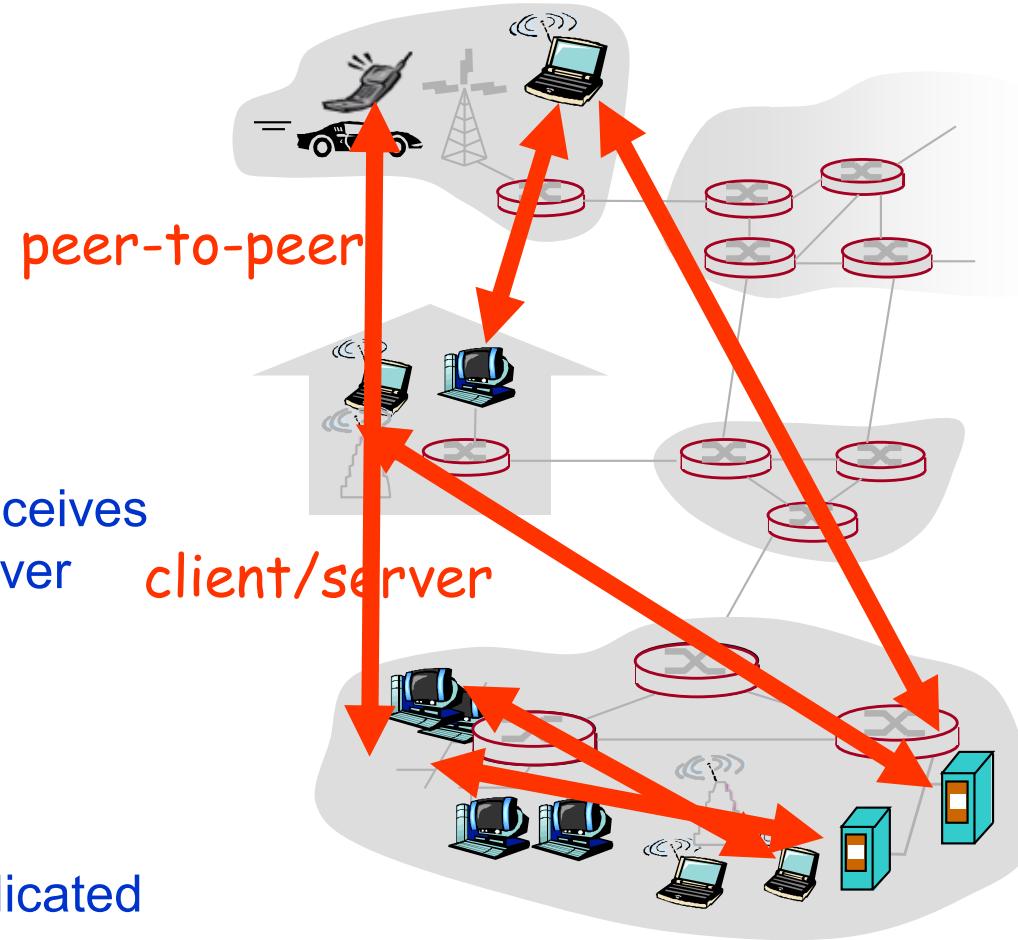
# Network Structure

- Network edge:  
applications and hosts
- Access networks, physical media: wired, wireless communication links
- Network core:
  - interconnected routers
  - network of networks



# The Network Edge

- End systems (hosts):
  - Run application programs, e.g. Web, e-mail, at the “edge of network”
- Client/server model:
  - Client host requests and receives service from always-on server (e.g. Web browser/server; e-mail client/server)
- Peer-to-peer model:
  - Minimal (or no) use of dedicated servers, e.g. Skype, BitTorrent



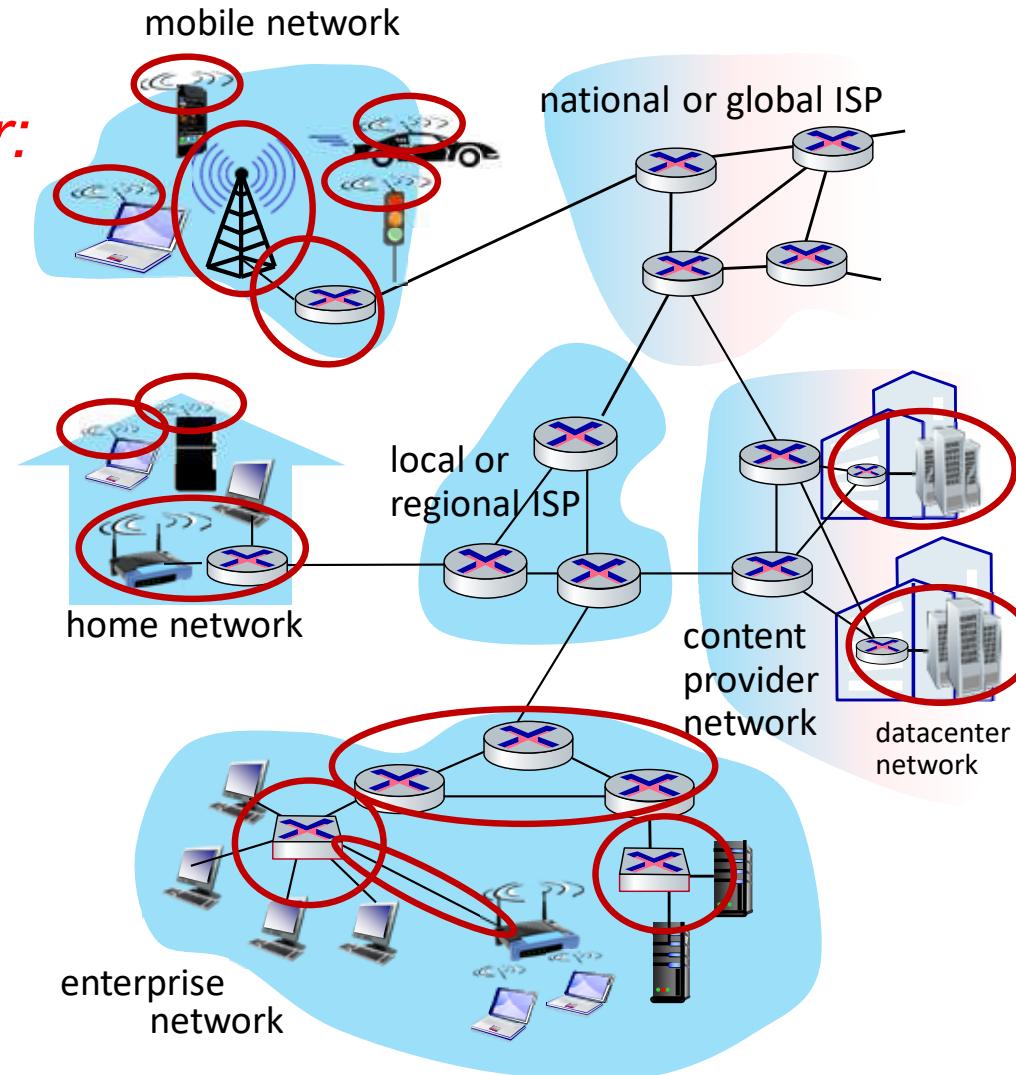
# Access Networks

*Connect end systems to edge router:*

- **Residential** access networks
- **Institutional** access networks (school, company)
- **Mobile** access networks (WiFi, 4G/5G)

*Access network differences:*

- Bitrate (bit/s)
- Shared or dedicated
- Wired or wireless
- ...



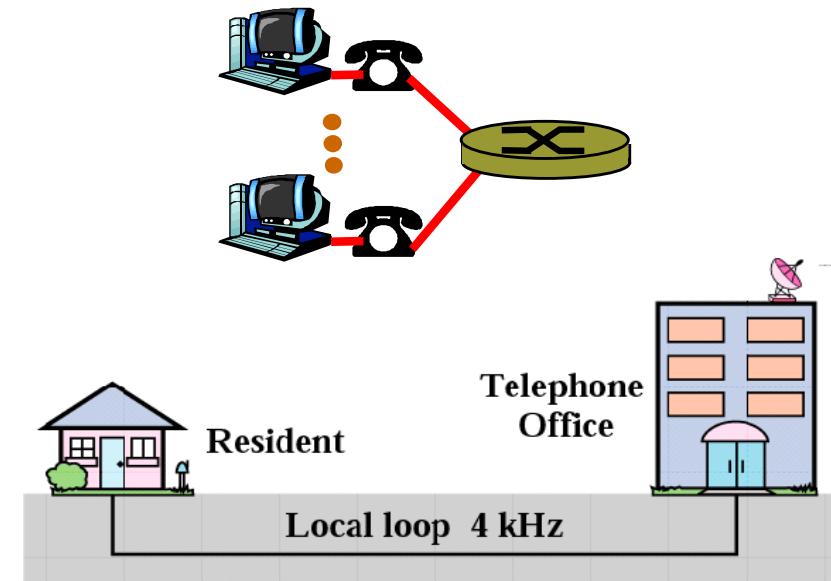
# Residential Access: *Point to Point*

## □ Dialup via modem

- Up to 56 kbit/s direct access to router (often less)
- Can't surf and phone at same time
- Not "always on"

## □ DSL: digital subscriber line

- Deployment: telephone company (typically)
- > 1 Mbit/s upstream (typically)
- > 12 Mbit/s downstream (typically)

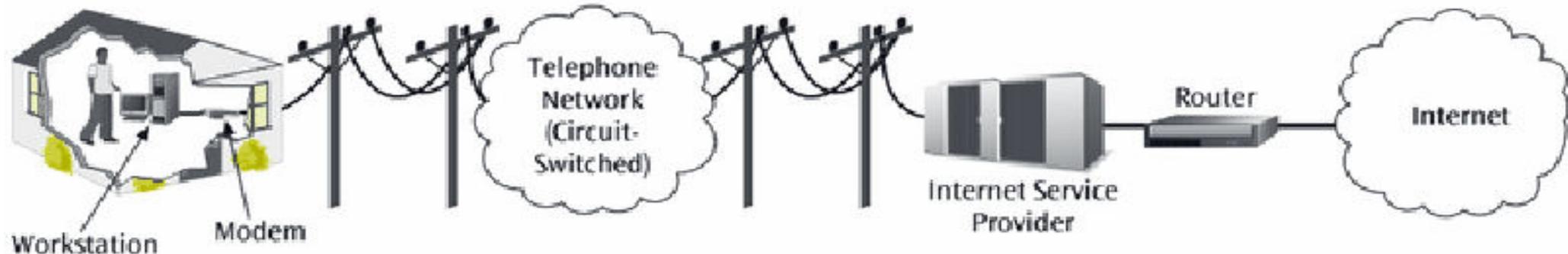
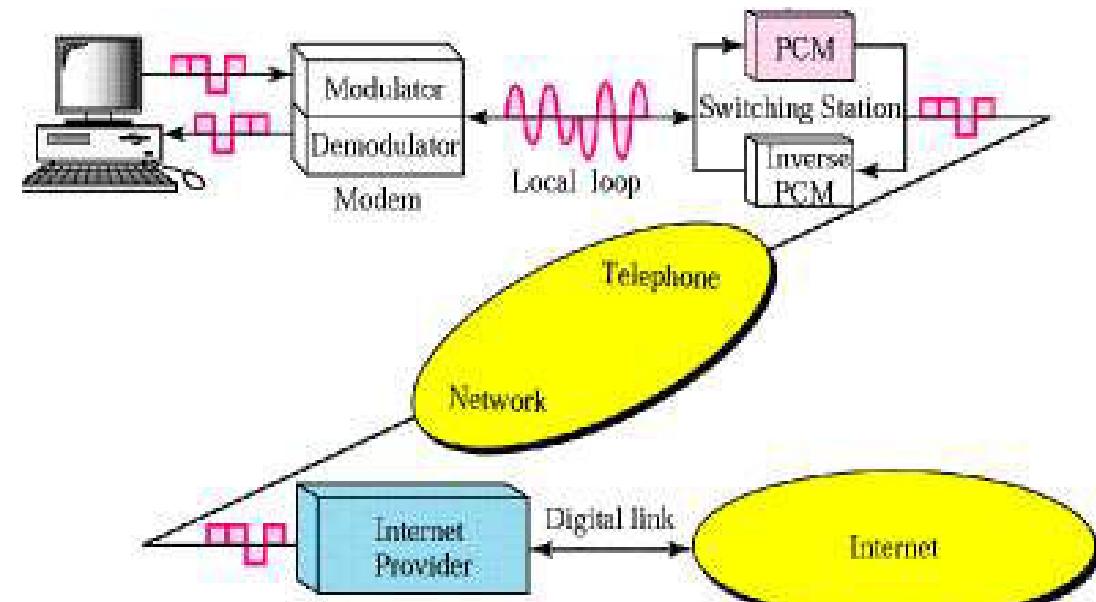


Both solutions use dedicated physical line to the switching central.

# Residential Access: Modem

Dial-up via modem:

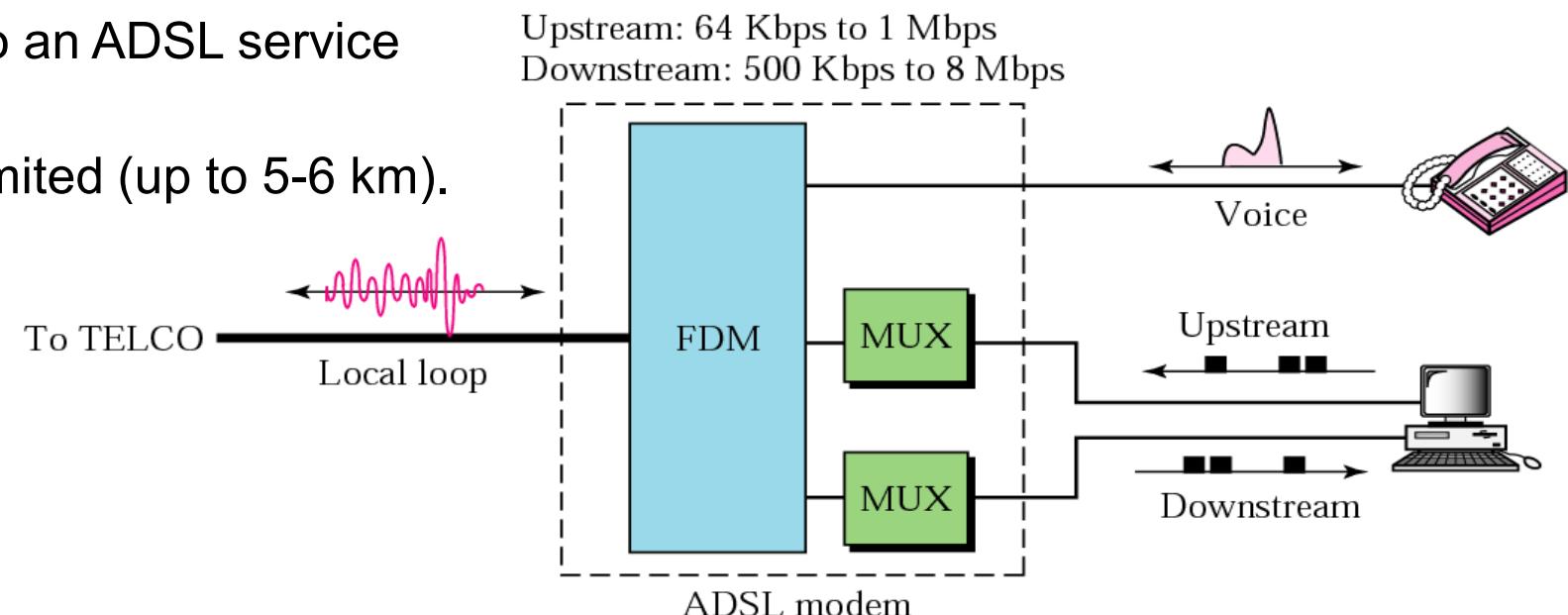
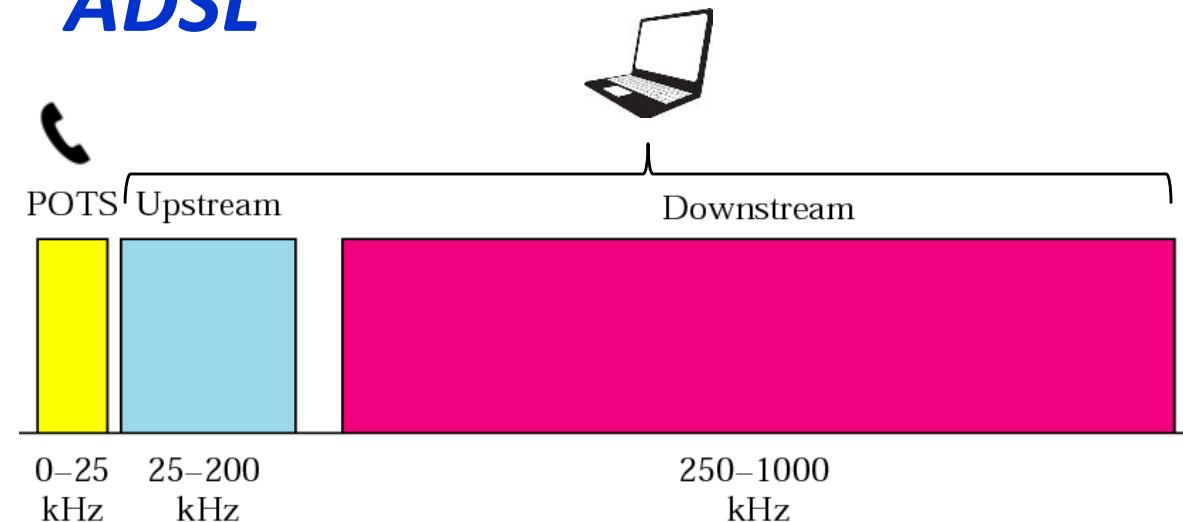
- Uses analog telephone network;
- Speeds up to 56 kbit/s;
- Connection to an Internet Service Provider (ISP).



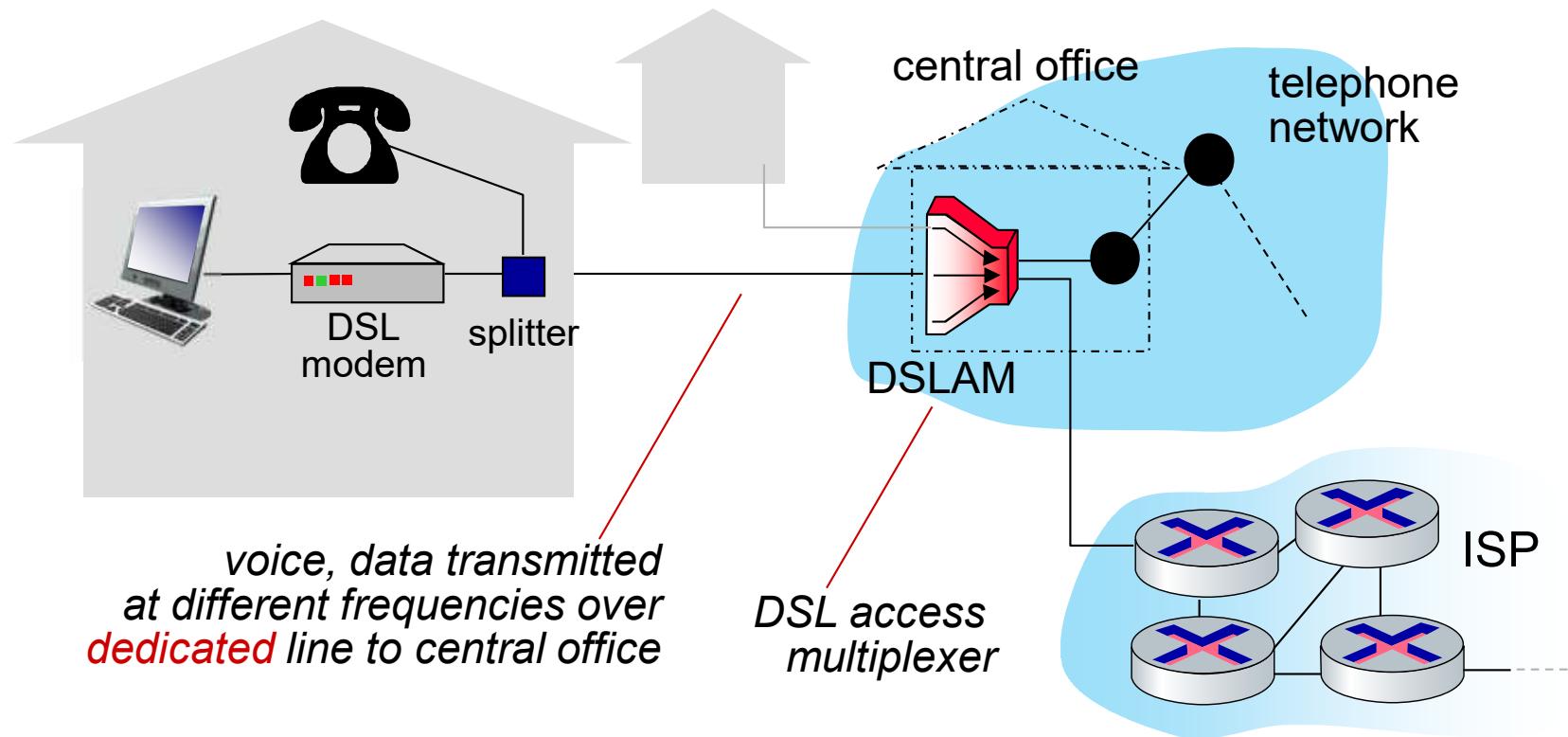
## Residential Access: ADSL

ADSL access:

- Uses the analog telephone infrastructure;
- Asymmetric connection:  
~1 Mbit/s (upstream),  
~12 Mbit/s (downstream);
- Connection to an ADSL service provider;
- Link length limited (up to 5-6 km).

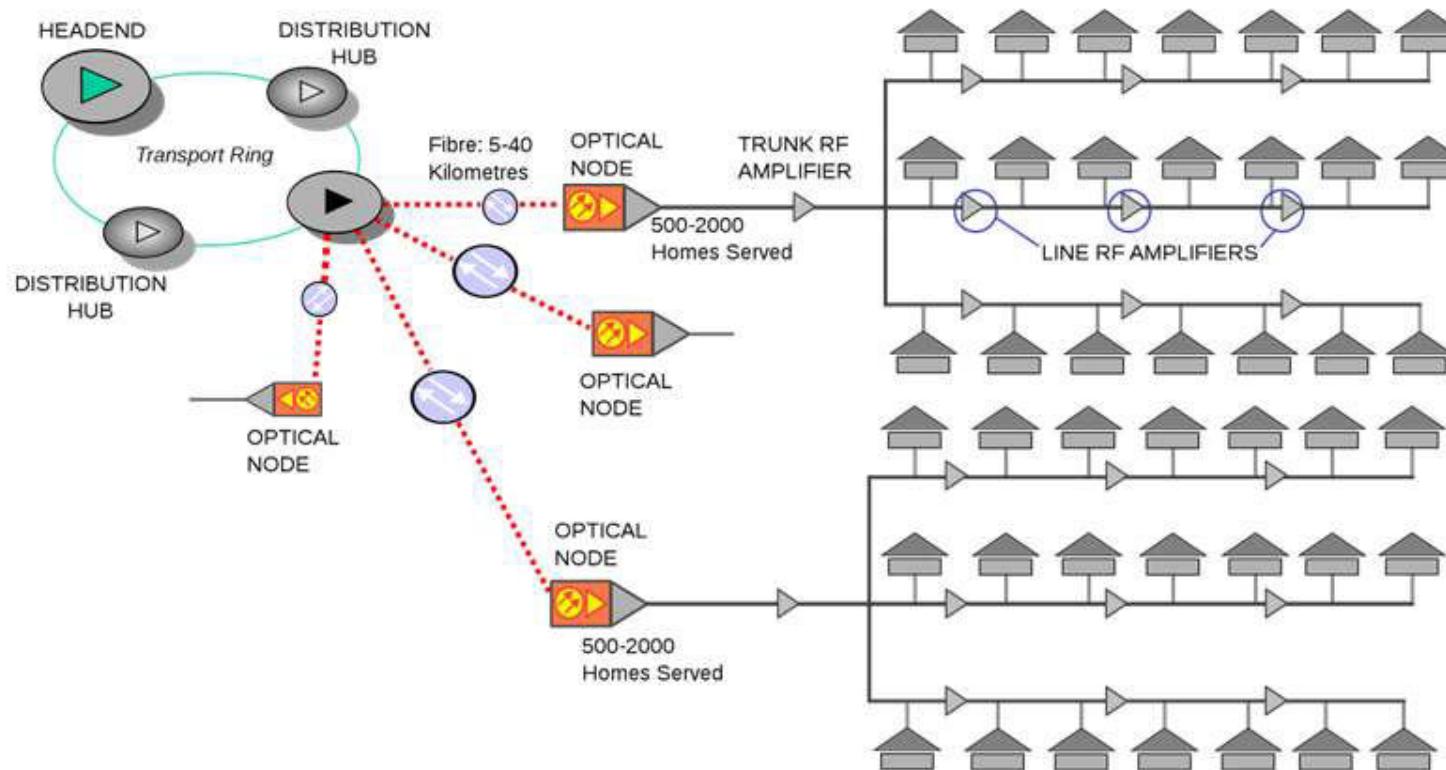


# Residential Access: ADSL



# Residential Access: Cable Modems

- HFC: hybrid fiber coaxial
- Network of cable and fiber attaches homes to ISP router
  - homes share access to router



# *Residential Access: Cable Modems*

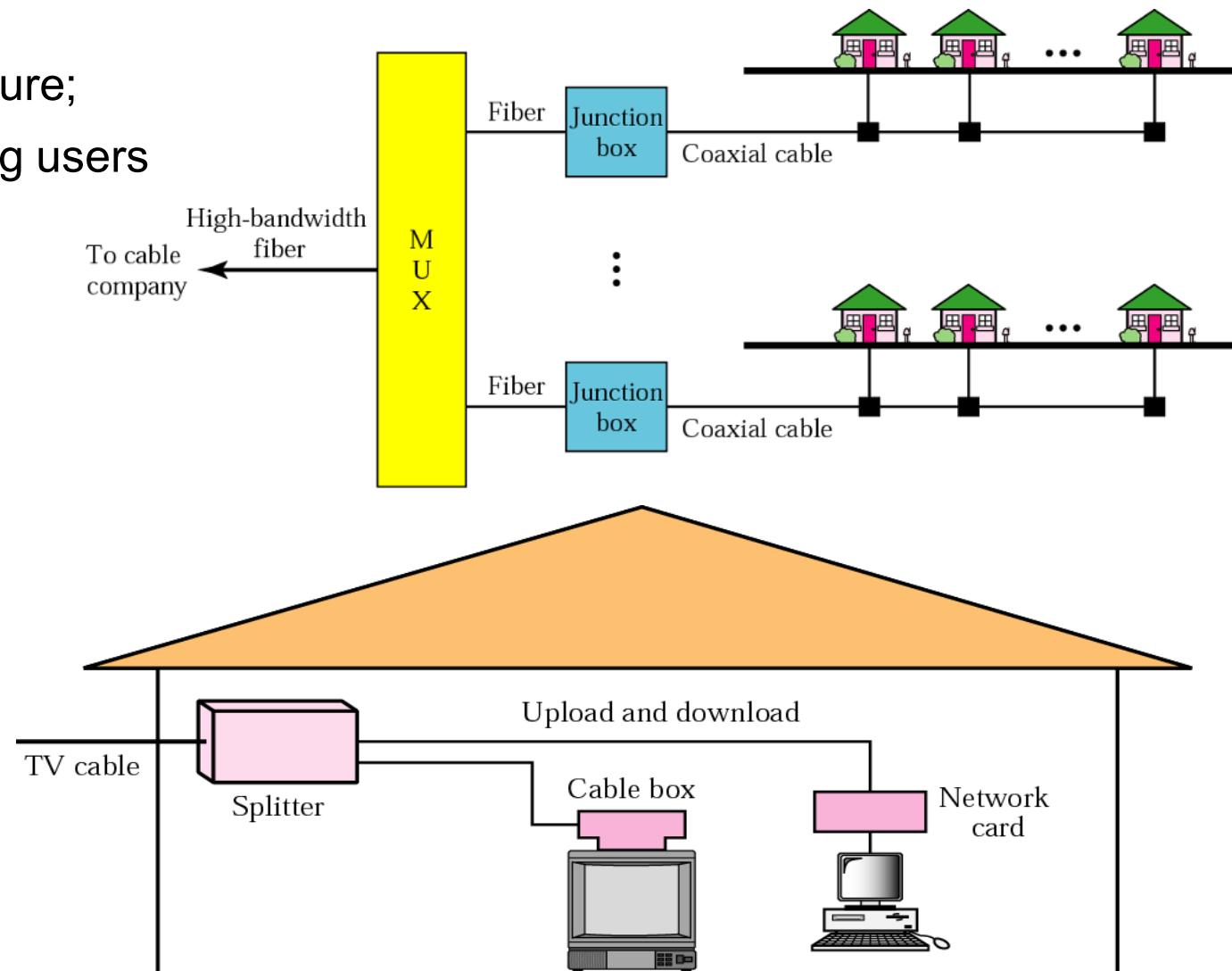
- HFC: hybrid fiber coaxial
- Internet access:
  - Data Over Cable Service Interface Specification (DOCSIS)

DOCSIS version <sup>[13]</sup>	Production date	Maximum downstream capacity	Maximum upstream capacity	Features
1.0	1997	40 Mbit/s	10 Mbit/s	Initial release
1.1	2001			Added VOIP capabilities and QoS mechanisms
2.0	2002		30 Mbit/s	Enhanced upstream data rates
3.0	2006	1 Gbit/s	200 Mbit/s	Significantly increased downstream/upstream data rates, introduced support for IPv6, introduced channel bonding
3.1	2013	10 Gbit/s	1–2 Gbit/s	Significantly increased downstream/upstream data rates, restructured channel specifications
4.0	2017		6 Gbit/s	Significantly increased upstream rates from DOCSIS 3.1

# Residential Access: Cable Modems

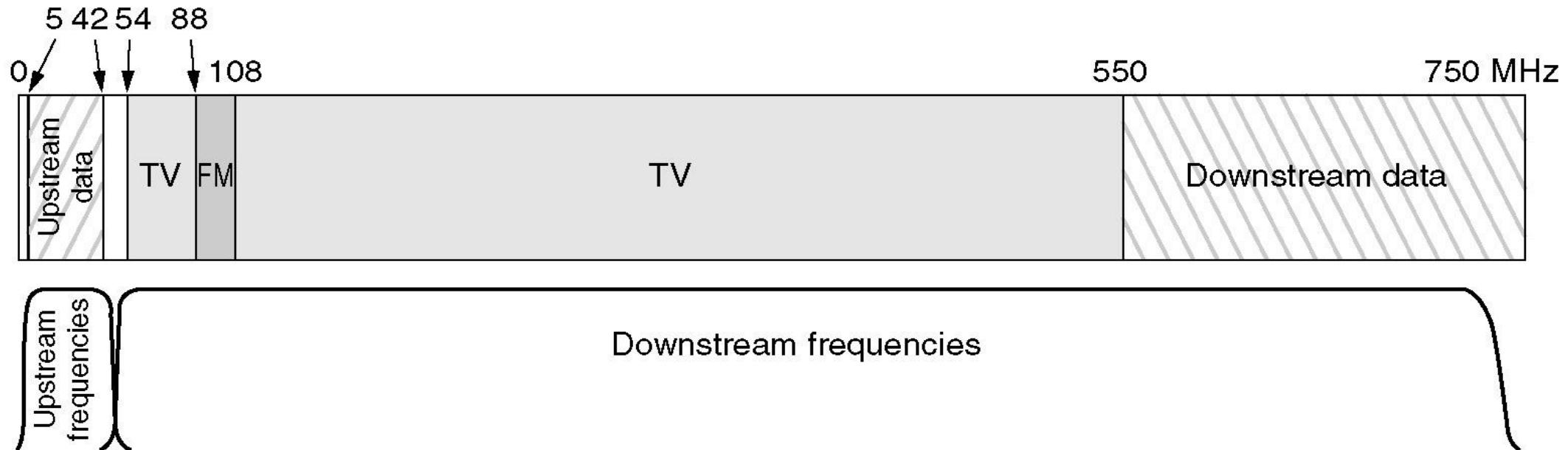
Cable modem access:

- Uses cable TV infrastructure;
- Connection shared among users in the same cable section;
- Asymmetric connection.



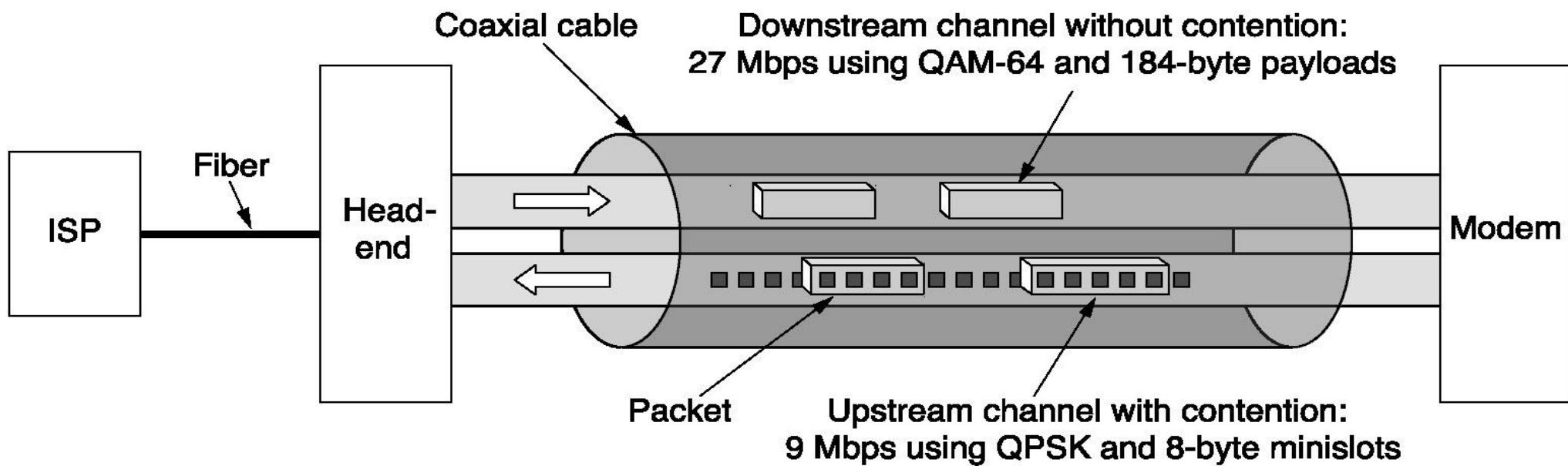
# Residential Access: *Cable Modems*

Example of available spectrum usage in a cable TV system:



# Residential Access: Cable Modems

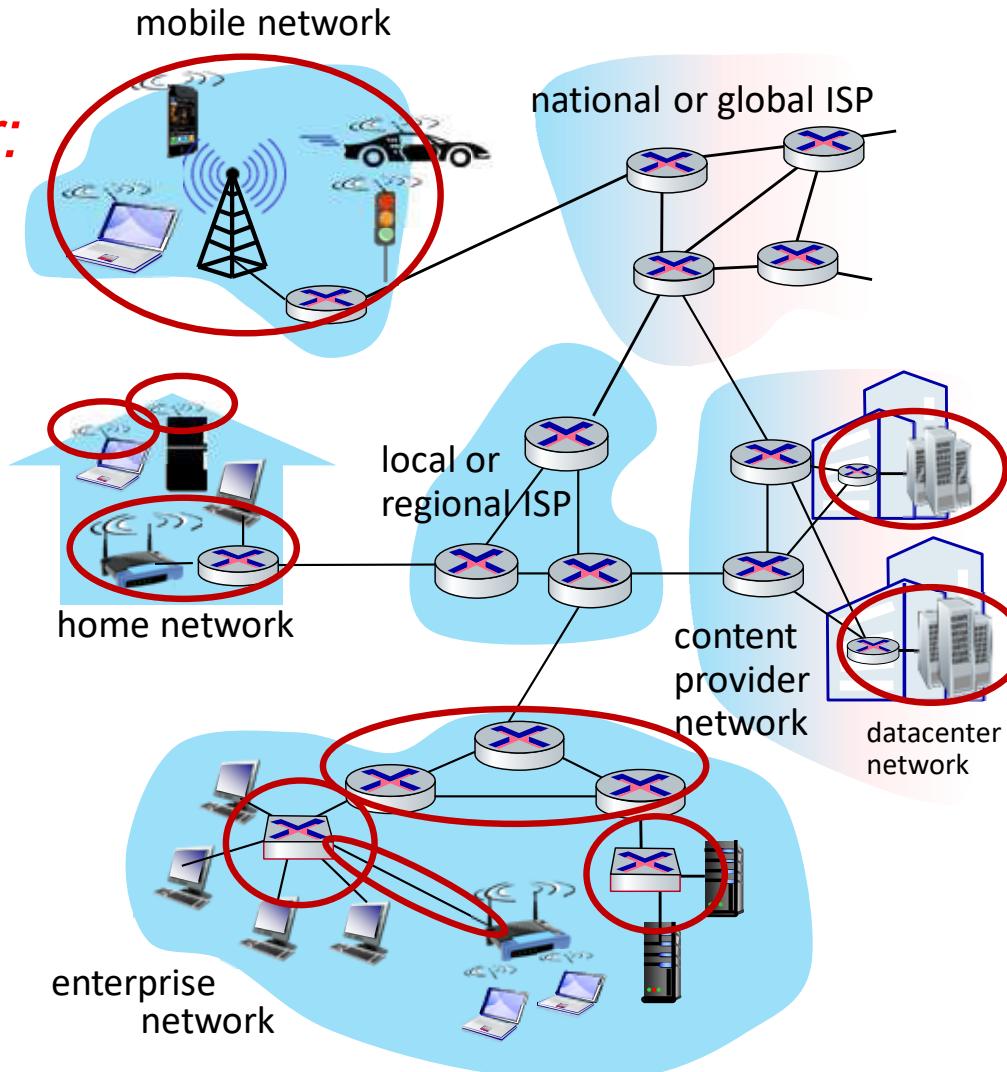
Example of upstream and downstream data channel implementation:



# Access Networks

*Connect end systems to edge router:*

- **Residential** access networks
- **Institutional** access networks (school, company)
- **Mobile** access networks (WiFi, 4G/5G)

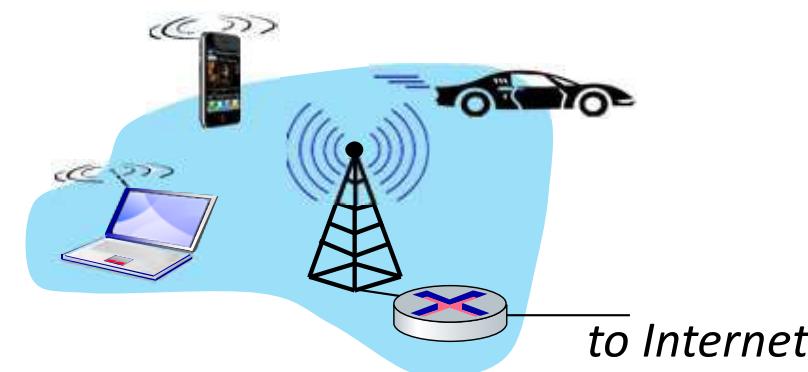
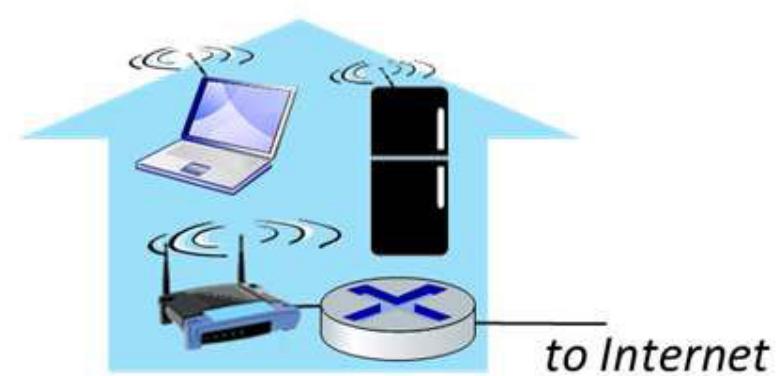


*Access network differences:*

- Bitrate (bit/s)
- Shared or dedicated
- Wired or wireless
- ...

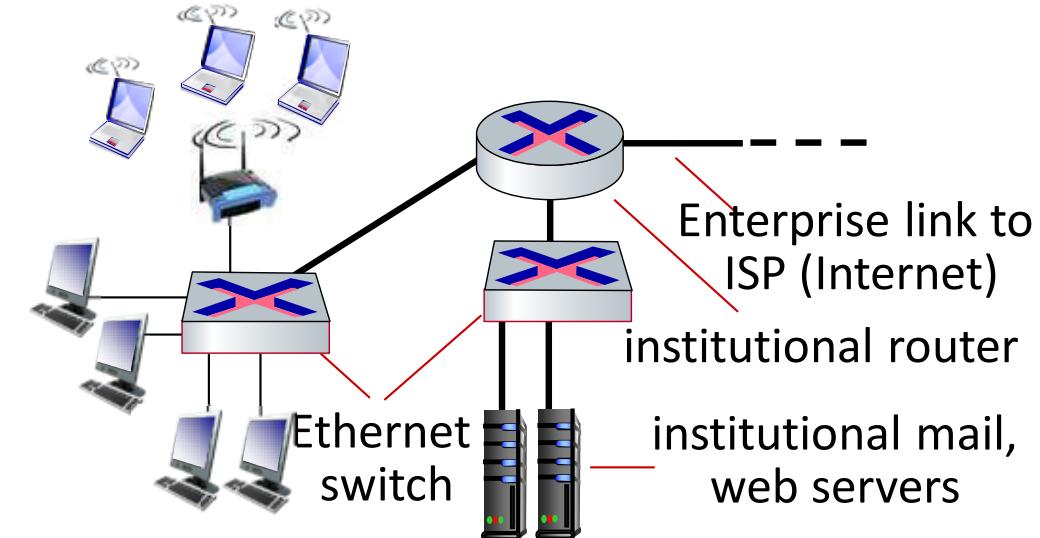
# Wireless Access Networks

- Shared wireless access network connects end system to router
  - Via a base station: “access point” (AP);
- Wireless LANs:
  - 802.11b/g/n/ac/... (**WiFi**):  
11, 54-600 Mbps, ...;
- Wider-area wireless access:
  - Provided by telecom operator (10's km);
  - ~1Mbps over cellular mobile (UMTS);
  - Always evolving: 4G (>10s Mbps),  
5G (60 Mbps–1 Gbps)...



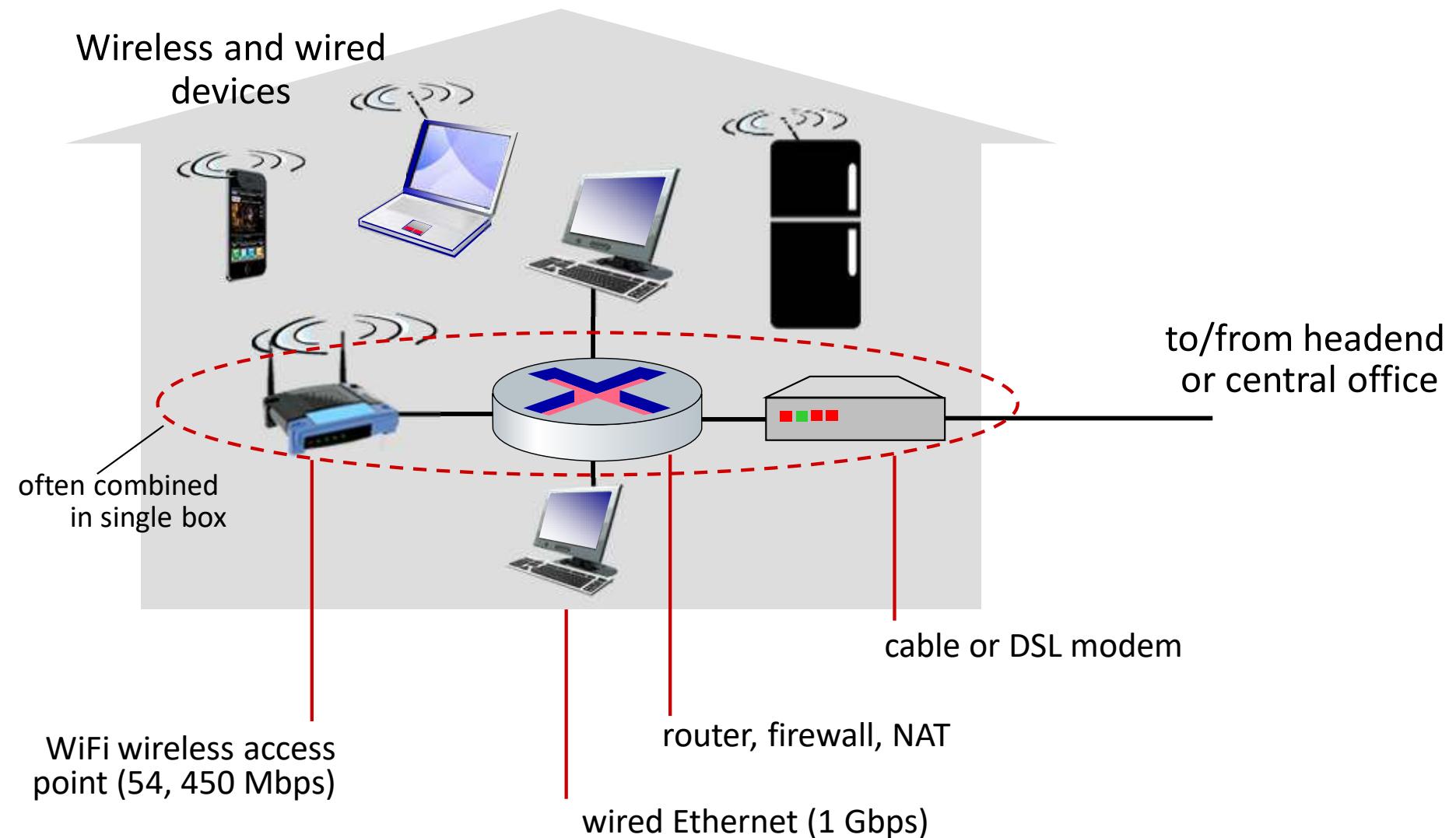
# Business Access: *Local Area Networks (LAN)*

- Companies, universities, etc.
  - Mix of wired, wireless link technologies, connecting a mix of switches and routers
- Typically use local area networks (LANs) for connection of end systems to edge router;
  - **Ethernet:**
    - 10 Mbs, 100Mbps, 1Gbps, 10Gbps, ...
  - **WiFi:**
    - Wireless access point: 11 Mbs, 54 Mbps, 450 Mbps, ...



LANs, Wireless LANs: chapters 6 and 7 (8<sup>th</sup> edition of the book).

# Home Networks

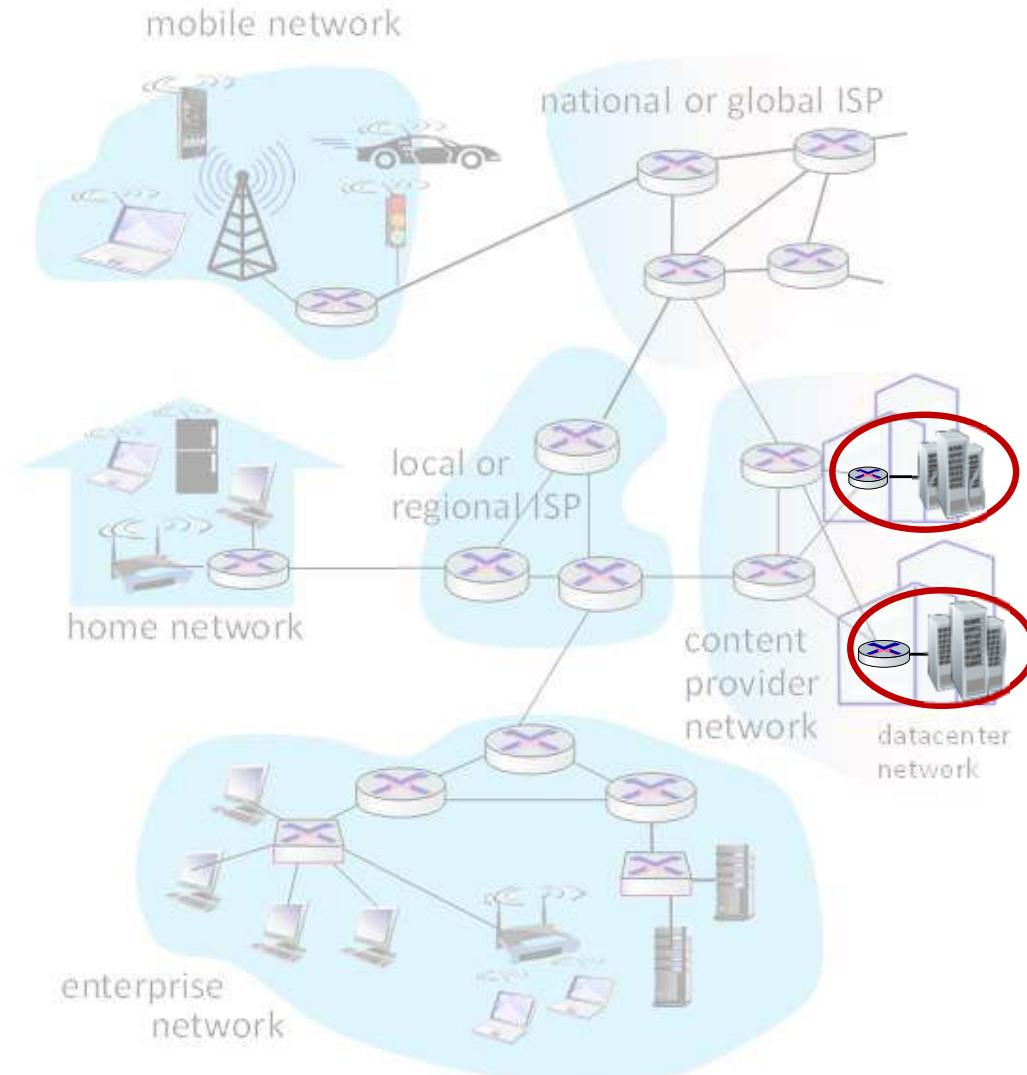


# Data Centre Networks

- High-bandwidth links (10s to 100s Gbps) connect hundreds to thousands of servers together, and to Internet



Courtesy: Massachusetts Green High Performance Computing Center ([mghpcc.org](http://mghpcc.org))



# *Physical Media*

- Data (packets composed of bits) propagates between transmitter and receiver pairs;
- Physical link: what lies between a transmitter and a receiver.

Two types of physical media:

- Guided/Wired:
  - Signals propagate in solid media: copper, fiber, coaxial cable;
- Unguided/Wireless:
  - Signals propagate freely, e.g., radio.

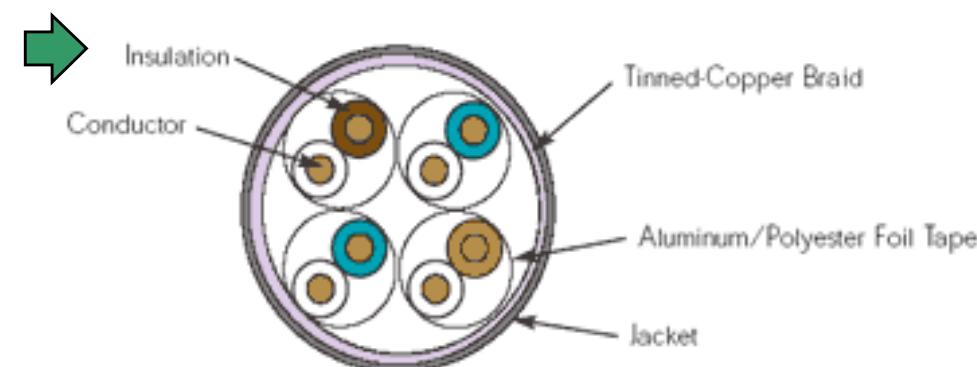
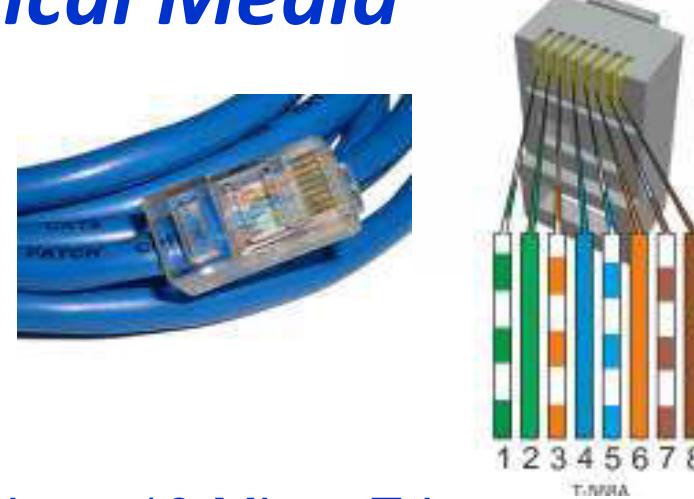
# Guided Physical Media

## Twisted Pair (TP)

- Two insulated copper wires:
  - Category 3: traditional phone wires, 10 Mbps Ethernet
  - Category 5: 100Mbps Ethernet, 100 m
  - Category 6: up to 1 Gbps
  - Category 7: up to 10 Gbps

...

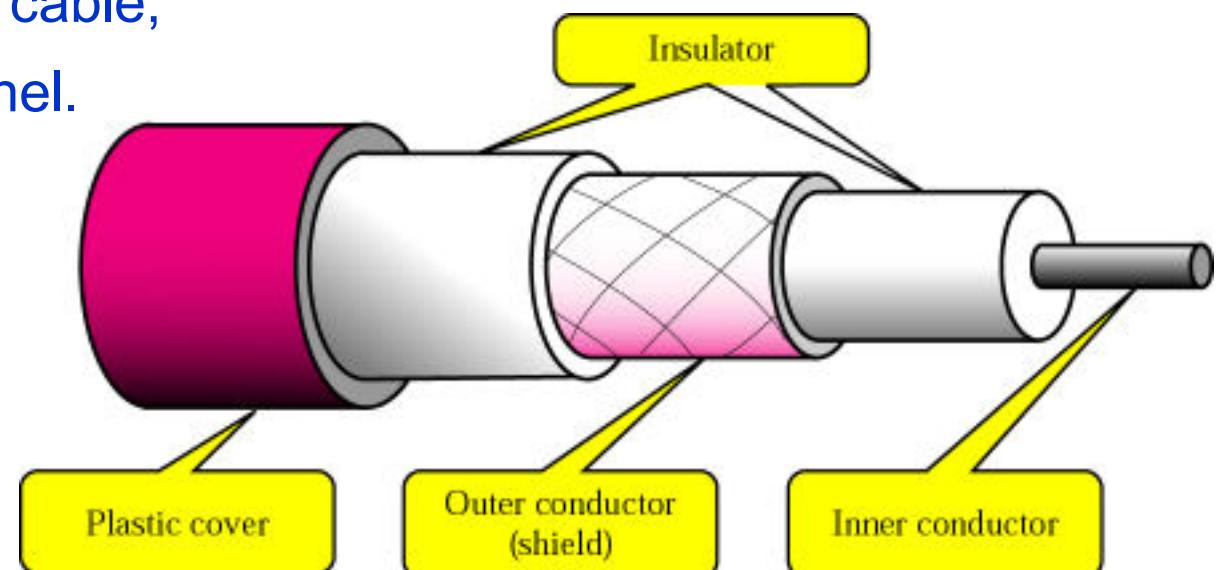
100 Gbps (15m)



# Guided Physical Media

## Coaxial cable:

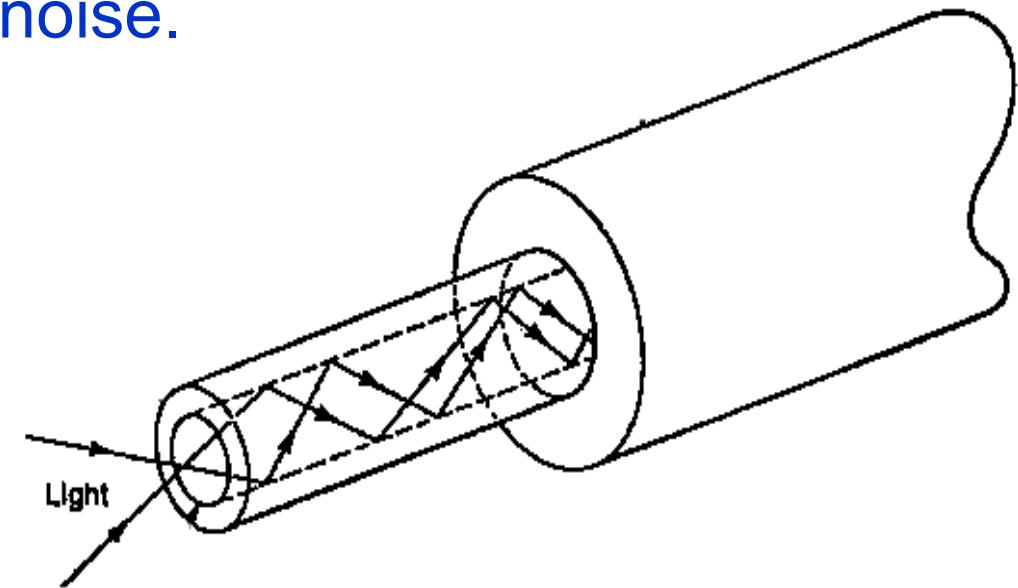
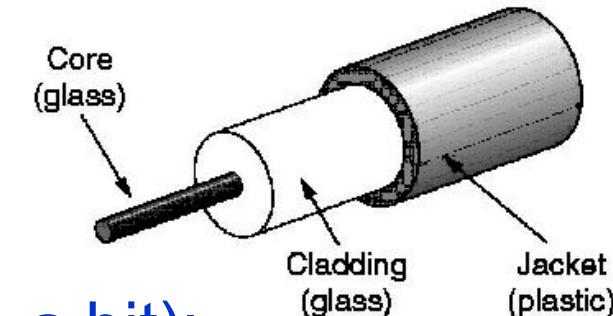
- Two concentric copper conductors;
- Bidirectional;
- Broadband:
  - Multiple channels on cable;
  - 100's Mbps per channel.



# Guided Physical Media

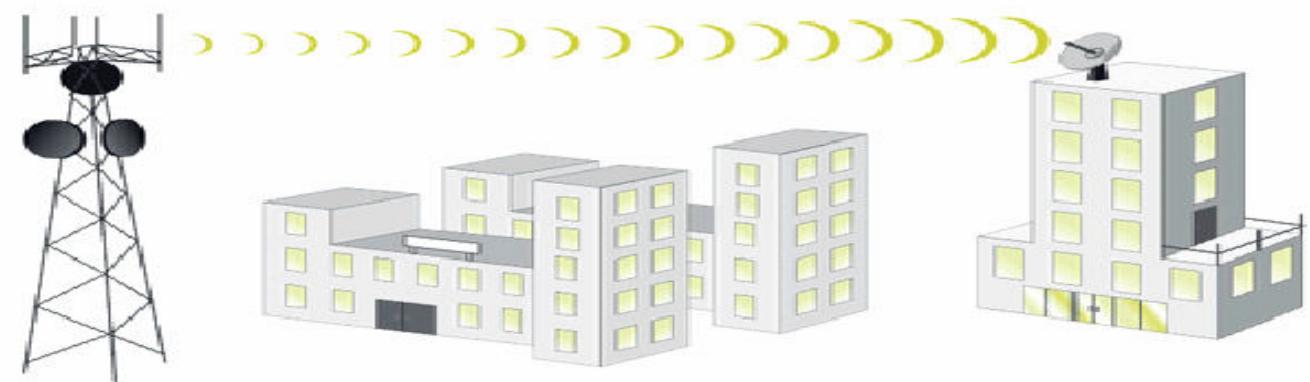
## Fiber optic cable:

- Glass fiber carrying light pulses (each pulse a bit);
- High-speed point-to-point transmission (10's-100's Gbps);
- Low error rate: repeaters spaced far apart;
- Immune to electromagnetic noise.



# *Unguided/Wireless Physical Media*

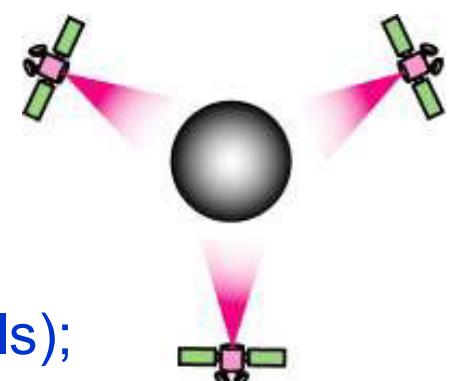
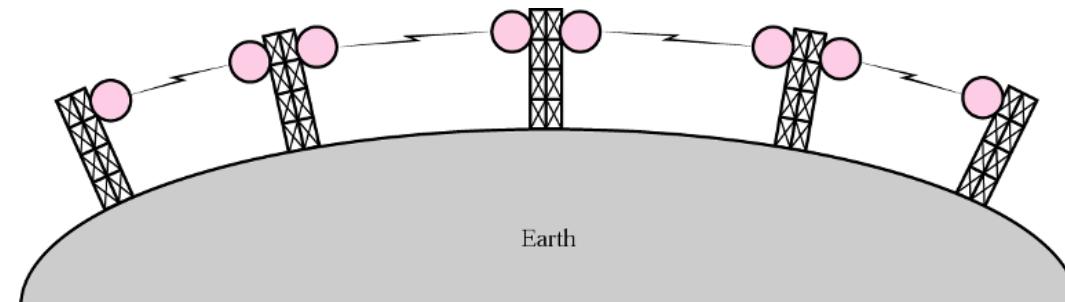
- No physical “wire”;
- Signal transmitted between sending and receiving antennas;
- Bidirectional;
- Propagation environment effects:
  - Reflection;
  - Obstruction by objects;
  - Interference.



# *Unguided Physical Media*

## Radio link types:

- Terrestrial microwave
  - e.g. up to 45 Mbps channels;
- Wireless LAN (WiFi)
  - 10-100's Mbps; 10's of meters;
- Wide-area (e.g., cellular)
  - 3G: ~ 1 Mbps; 4G: ~10 Mbps; 5G;
    - over ~10 km
- Satellite
  - kbps to 45 Mbps channel (or multiple smaller channels);
  - Geosynchronous (GEO) versus low altitude (LEO);
  - ~270 ms end-end delay (GEO).



# Unguided Physical Media

## Terrestrial microwave

- Distance to the horizon:

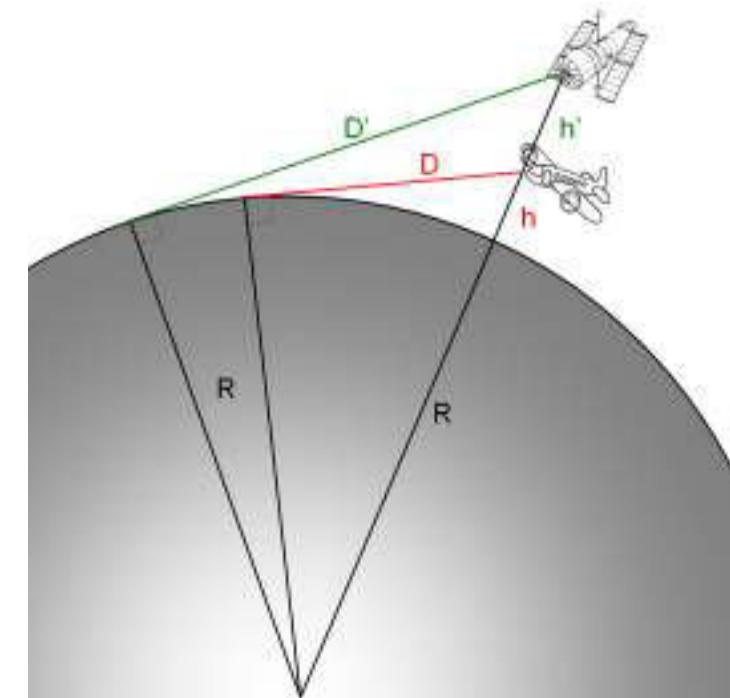
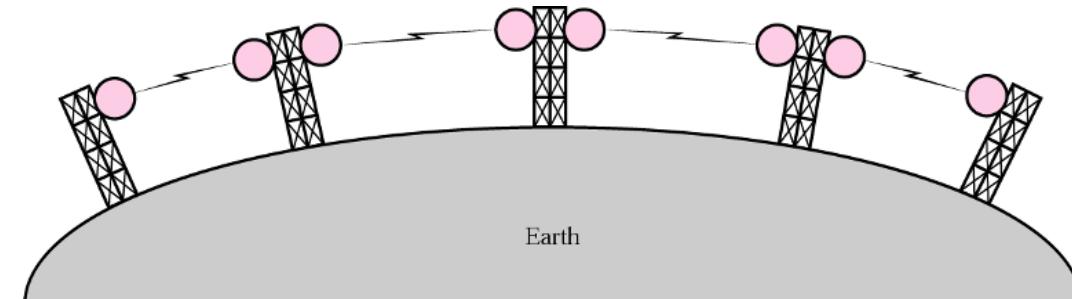
$$D^2 + R^2 = (R + h)^2$$

$$D = \sqrt{2hR + h^2}$$

$$R = 6371 \text{ km} = 6371000 \text{ m}$$

$$\text{with } h=1,8 \text{ m} \rightarrow D = 4,8 \text{ km}$$

$$\text{with } h=40 \text{ m} \rightarrow D = 22,6 \text{ km}$$

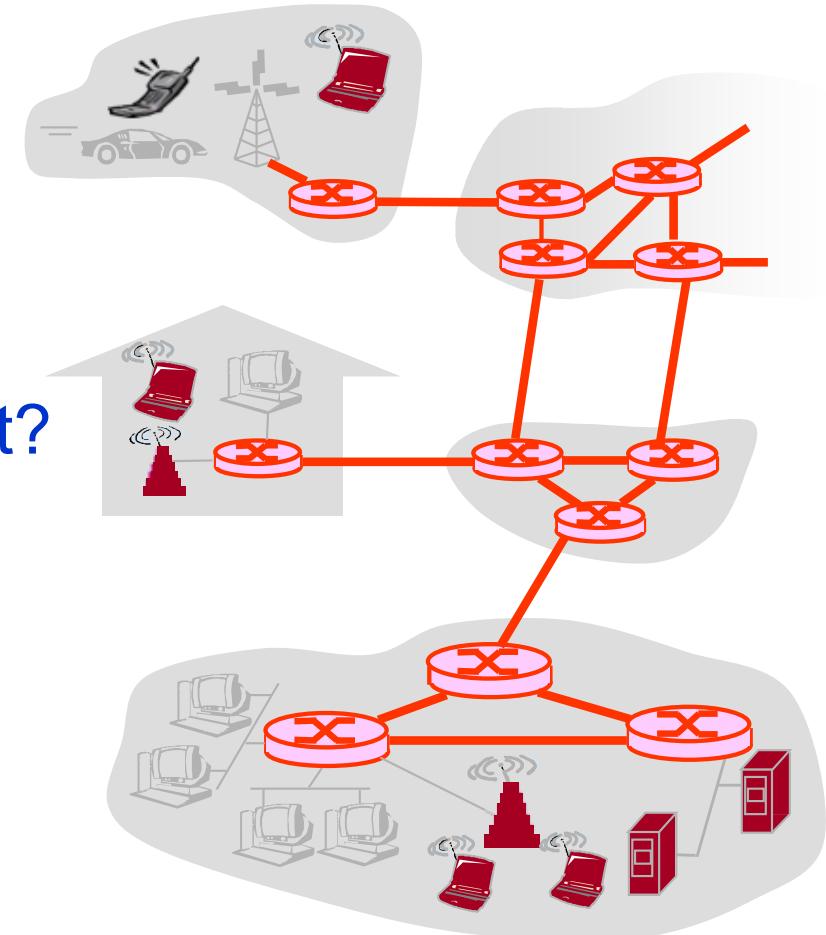


# *Objectives*

- Terminology
- What is a protocol?
- Network edge (hosts, access net, physical media)
- **Network core**
  - **Circuit switching, Packet switching, Internet structure**
- Performance metrics: loss, delay, throughput
- Protocol layers, service models

# The Network Core

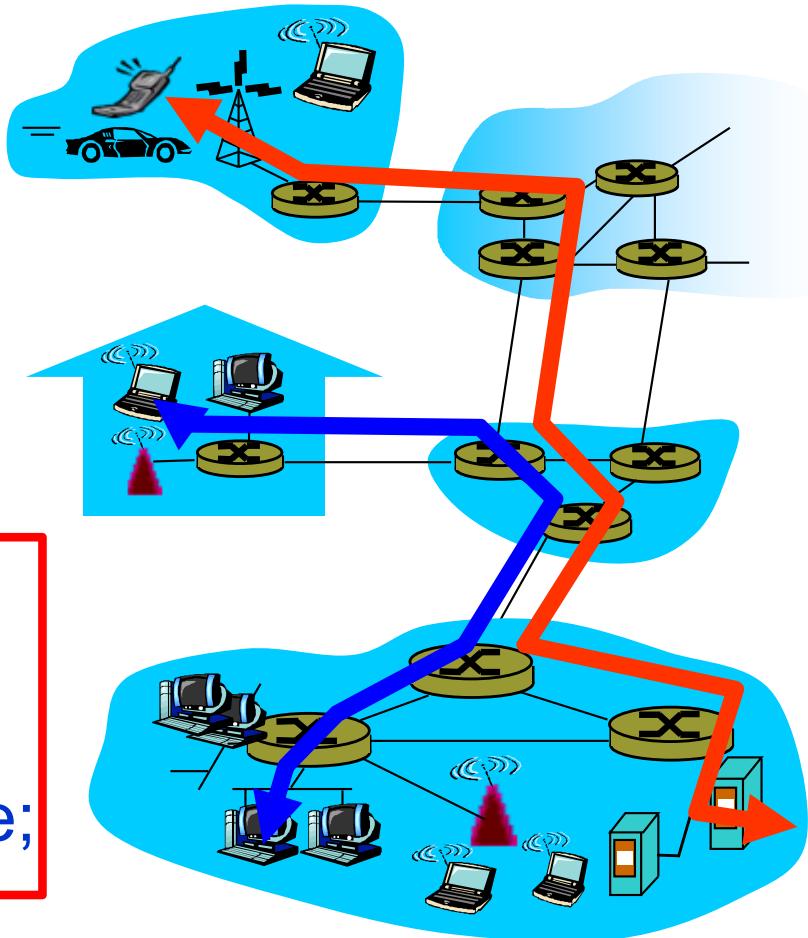
- Mesh of interconnected routers.
- *The fundamental question:*  
How is data transferred through the net?
  - Circuit switching:  
Dedicated circuit per call;  
e.g.: telephone network.
  - Packet-switching:  
Data sent through net in discrete  
“chunks”.



# Circuit Switching

End-end resources reserved for “call”:

- Link bandwidth;
- Switch capacity;
- Call setup required;
- Dedicated resources: **no sharing**;
- Circuit-like (guaranteed) performance;



Not used for computer networks!

# Circuit Switching

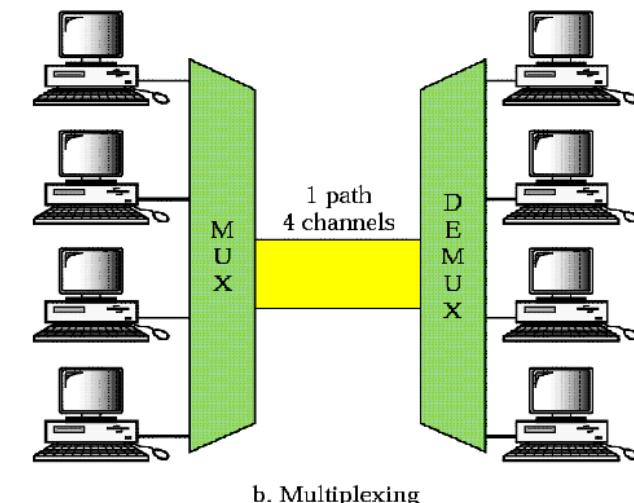
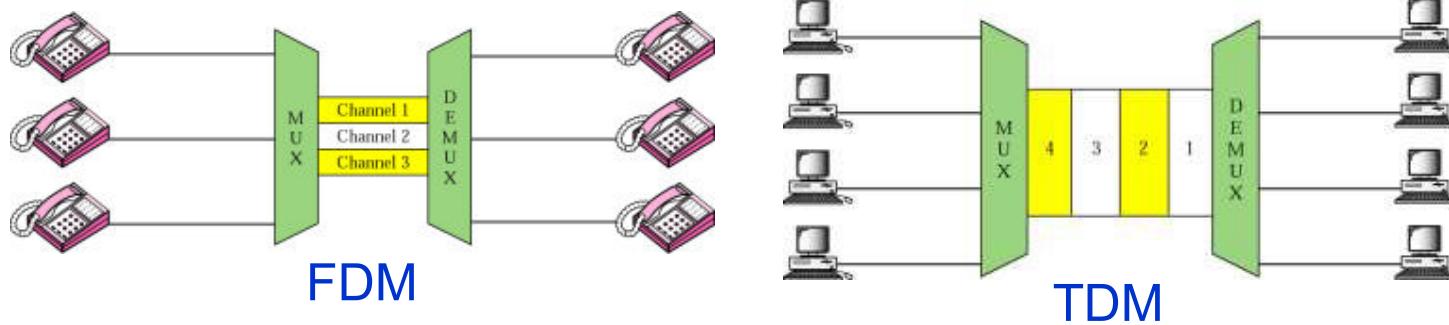


Network resources (e.g., bandwidth) divided into “pieces”:

- Pieces allocated to calls;
- Resource piece *idle* if not used by owning call (no sharing).

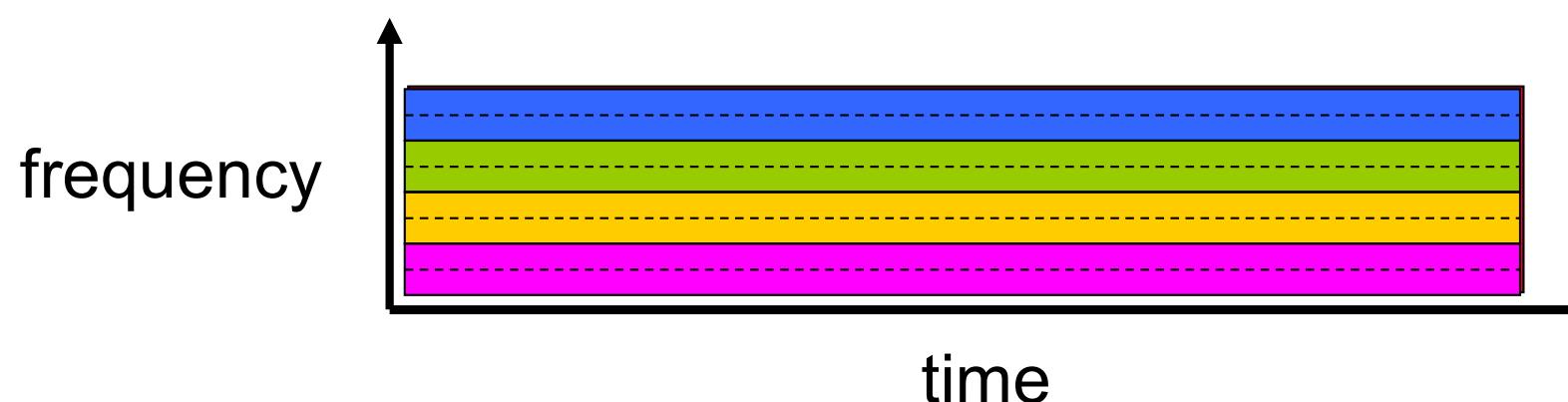
Dividing link bandwidth into “pieces” (**multiplexing**), e.g.:

- Frequency division (FDM);
- Time division (TDM).

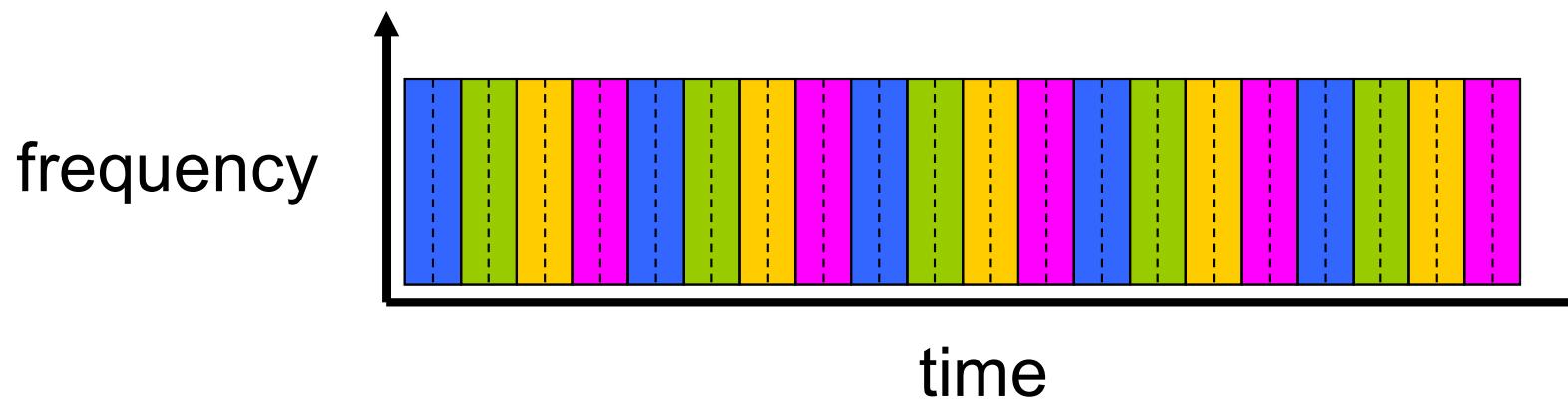


# *Circuit Switching: FDM and TDM*

FDM



TDM



Example:

4 users



## *Circuit Switching: Numerical Example*

- How long does it take to send a file of 640,000 bits from host A to host B over a circuit-switched network?
  - The bit rate of available links is 2.048 Mbps;
  - Each link is shared using TDM, with 32 slots/line;
  - It takes 500 msec to establish end-to-end circuit.

Let's work it out!

# Packet Switching

Each end-end data stream is divided into **packets**:

- Packets from different users **share** network resources;
- Each packet uses **full link bandwidth**;
- Resources **used as needed**

Bandwidth division into “pieces”  
Dedicated allocation  
Resource reservation



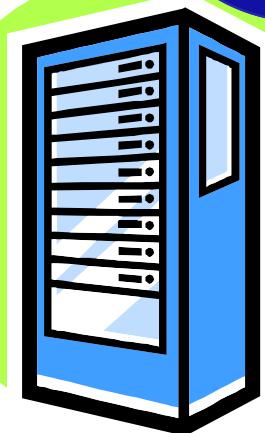
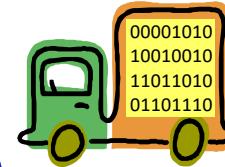
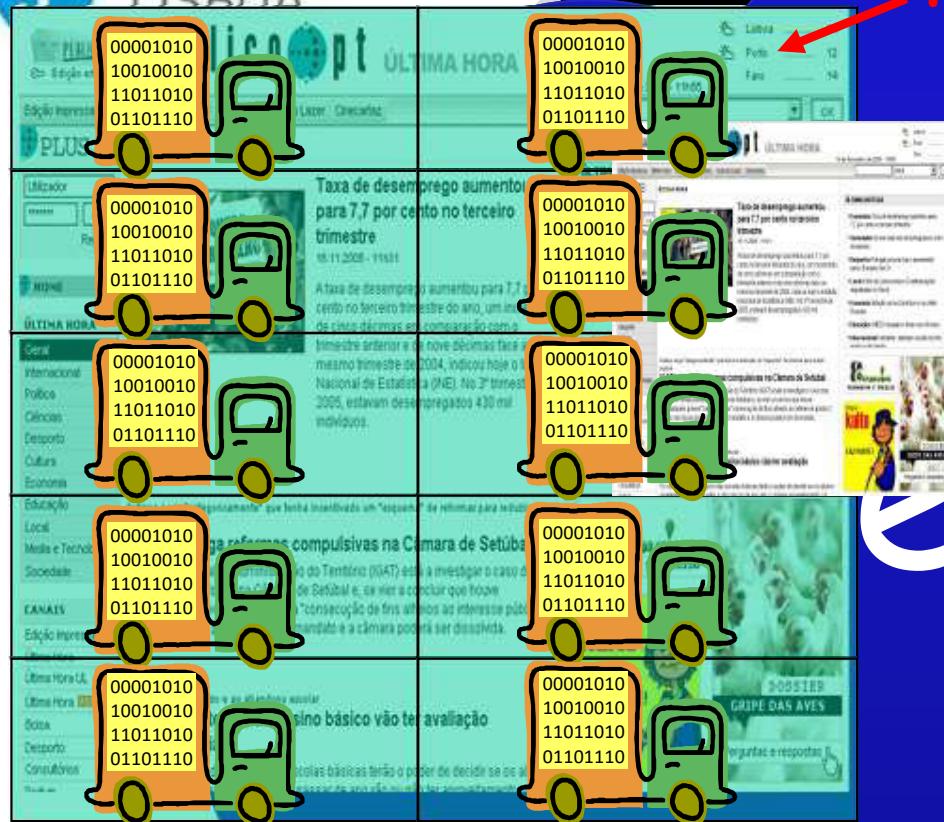
Resource contention:

- Aggregate demand can exceed resources available;
- **Congestion**: packets queue, waiting for link (eventual loss);
- **Store and forward**: packets move one hop at a time:
  - Node receives complete packet before forwarding.



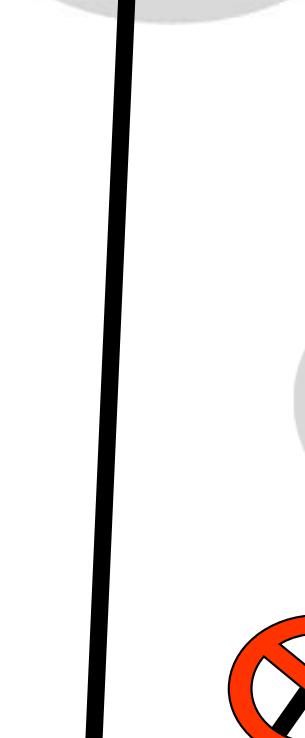
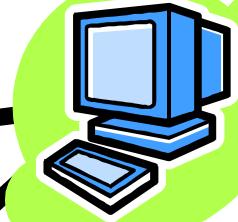
# "Surfing the net"

Packet

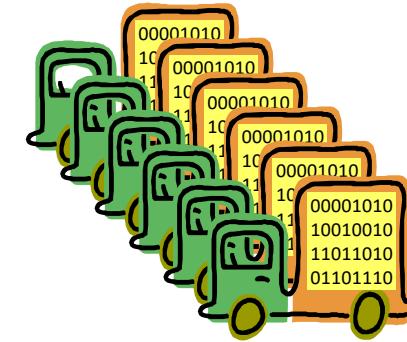




link



router



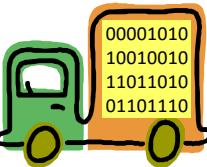
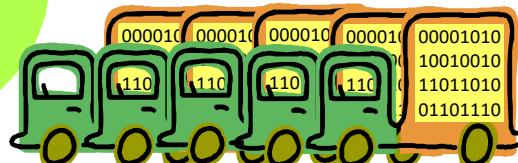
*Routing packets*



link



router

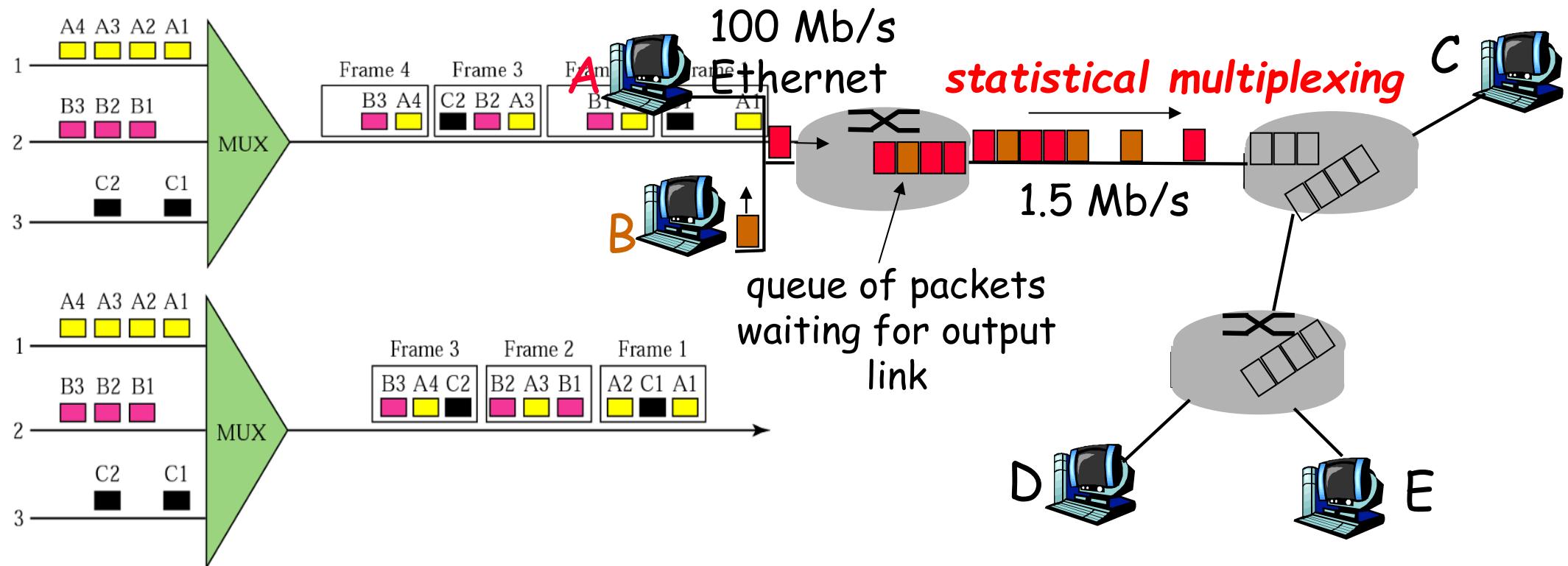


## Routing packets

# Packet Switching: Statistical Multiplexing

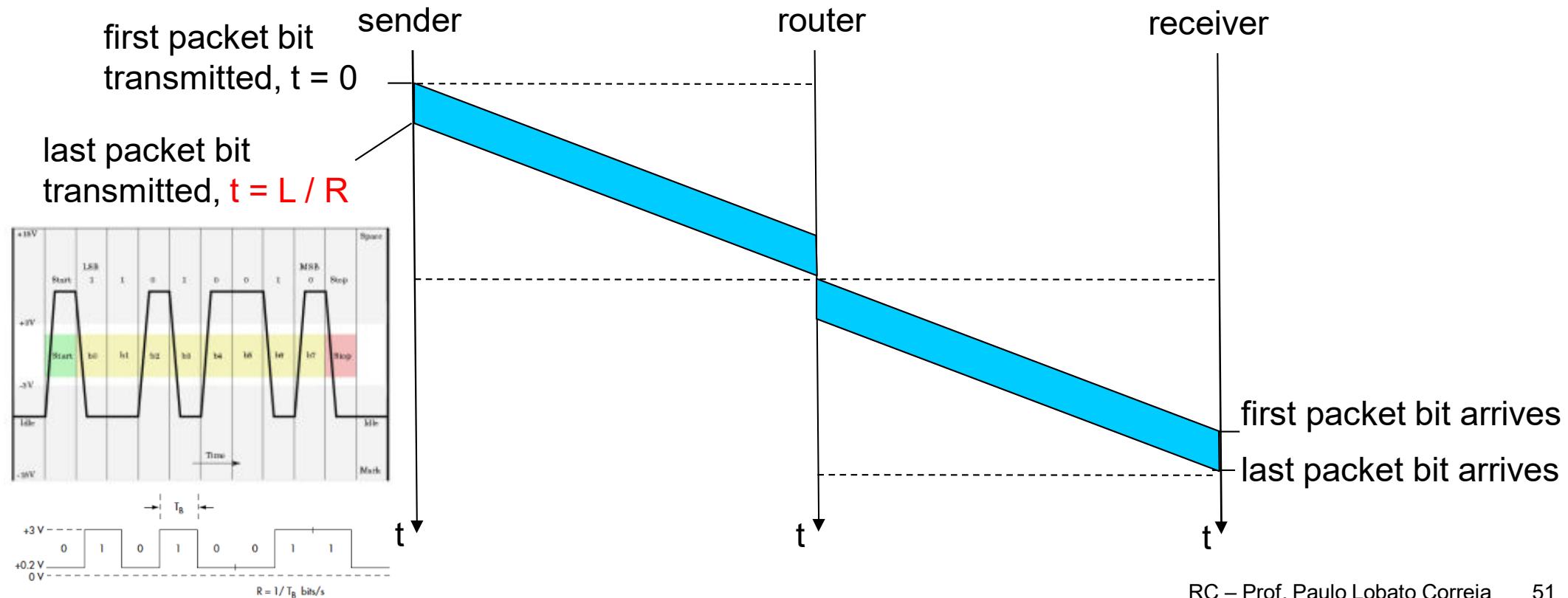
TDM: each host always gets the same slot in a revolving TDM frame.

Packet switching: sequence of packets has no fixed pattern;  
bandwidth is shared on demand → ***statistical multiplexing***.

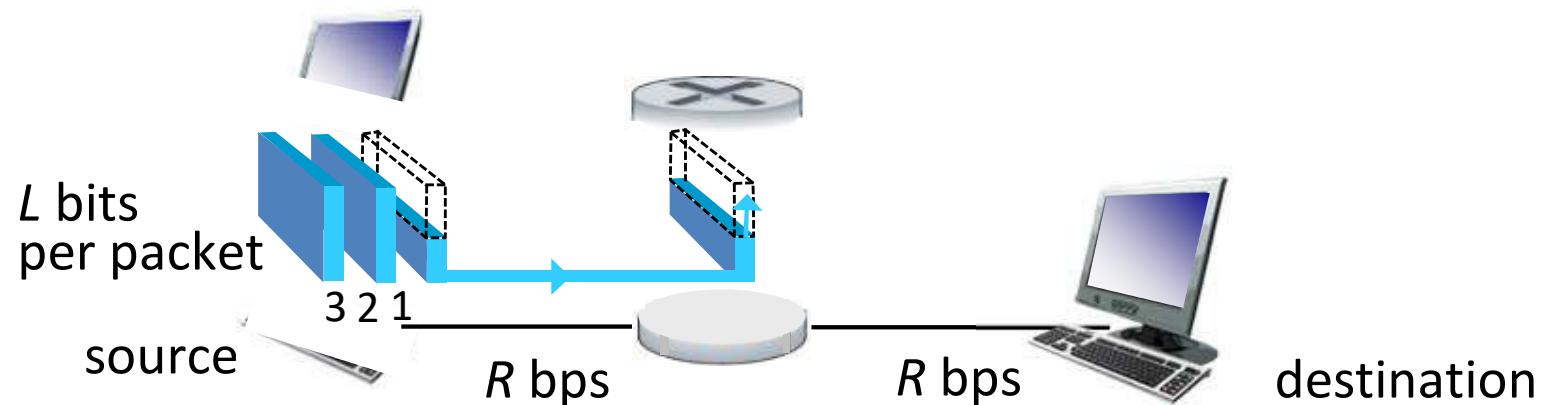


# Packet Switching: Store-and-Forward

- It takes  $L/R$  seconds to transmit a packet of  $L$  bits on link at  $R$  bps;
- **Store and forward:**  
entire packet must arrive at router before transmission on next link;



# Packet Switching: Store-and-Forward



## Packet transmission delay:

- takes  $L/R$  seconds to transmit (push out)  $L$ -bit packet into link at  $R$  bit/s

## Store and forward:

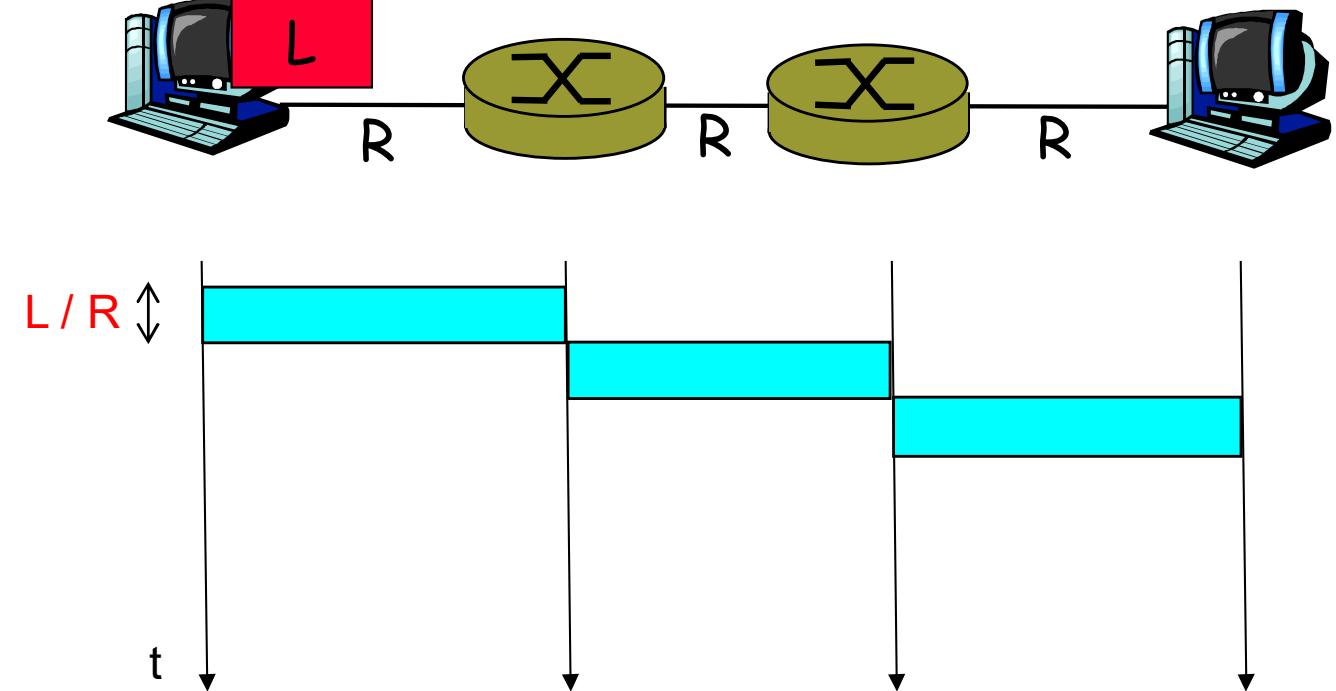
- entire packet must arrive at router before it can be transmitted on next link

# Packet Switching: Store-and-Forward

Problem 1

## Example:

- $L = 7.5 \text{ Mbits}$
- $R = 1.5 \text{ Mbps}$



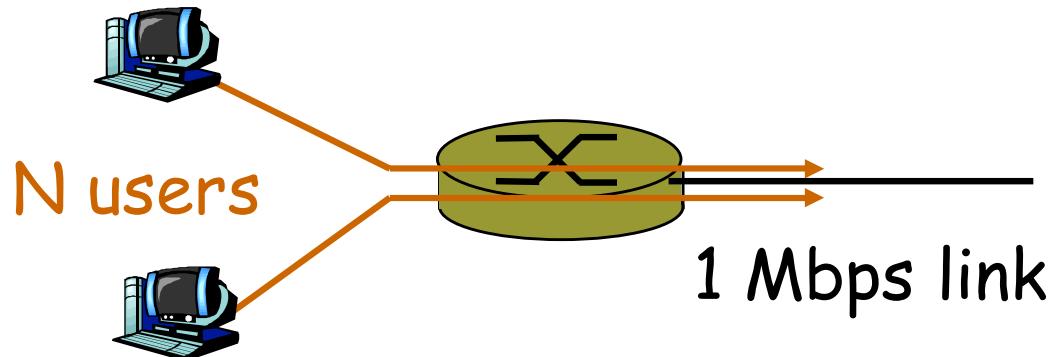
$$\text{delay} = 3L/R \text{ (assuming zero propagation delay)}.$$

$$\text{transmission delay} = 15 \text{ sec}$$

# Packet Switching versus Circuit Switching

*Packet switching allows more users to use network!*

- 1 Mb/s link
- Each user:
  - 100 kbit/s when “active”
  - Active 10% of time
- *Circuit-switching:*
  - 10 users
- *Packet switching:*
  - with 35 users, probability > 10 active at same time is less than .0004 !



# *Packet Switching versus Circuit Switching*

- Packet switching is great for bursty data:
  - Resource sharing;
  - Simpler, no call setup;
- With excessive congestion:
  - Packet delay and loss;
  - Protocols needed for reliable data transfer, congestion control;



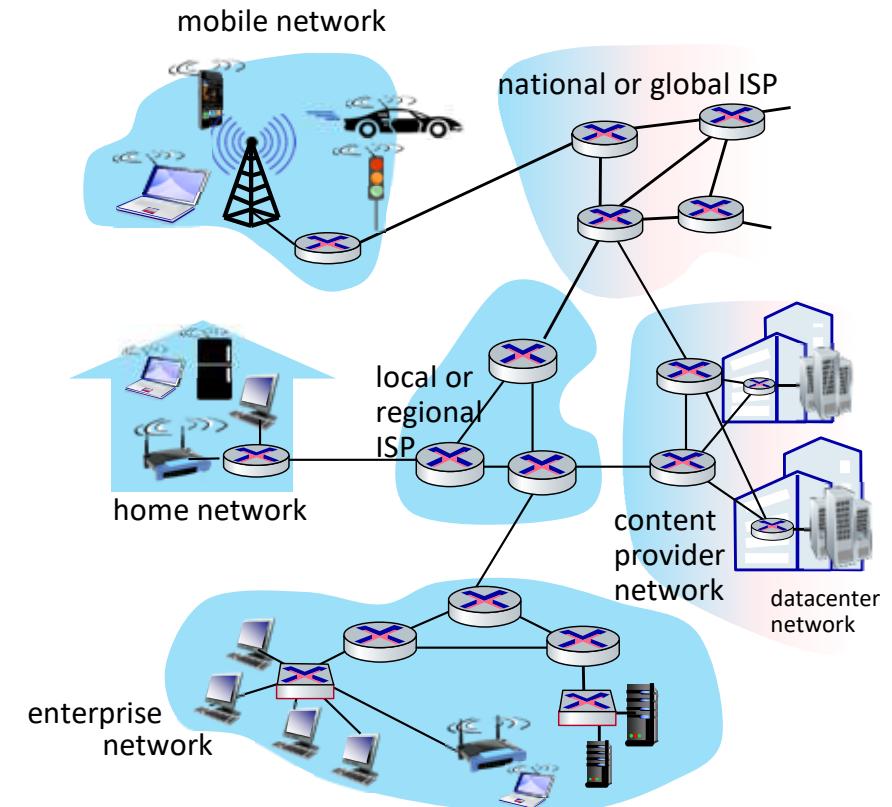
Q: How to provide circuit-like behavior?

- Bandwidth guarantees needed for audio/video applications!
- Still an unsolved problem (chapter 7)...

Q: human analogies of reserved resources (circuit switching)  
versus on-demand allocation (packet-switching)?

# *Internet structure: a “network of networks”*

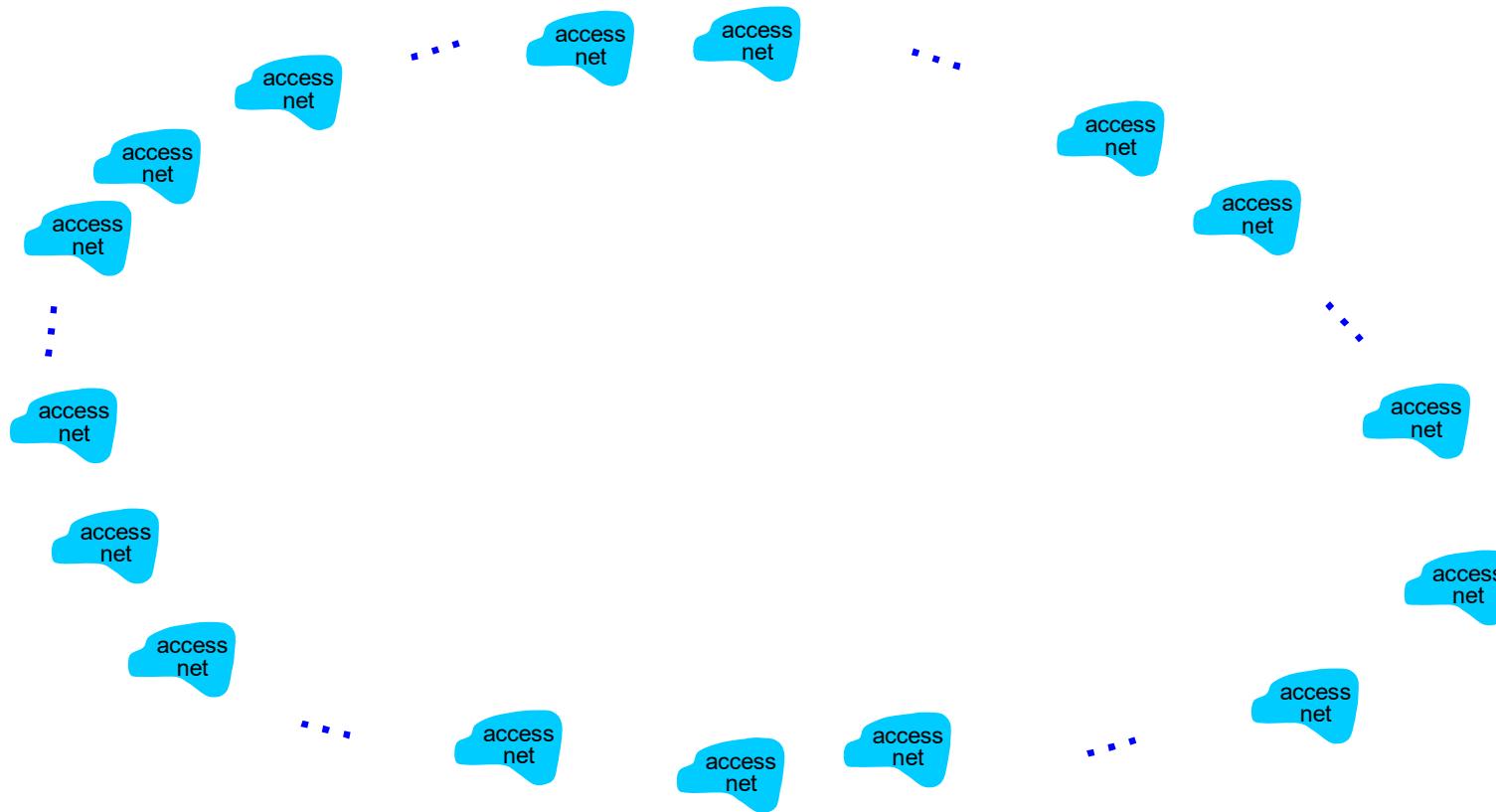
- hosts connect to Internet via **access** Internet Service Providers (ISPs)
- access ISPs in turn must be interconnected
  - so that *any two hosts (anywhere!)* can send packets to each other
- resulting network of networks is very complex
  - evolution driven by **economics, national policies**



*Let's take a stepwise approach to describe current Internet structure*

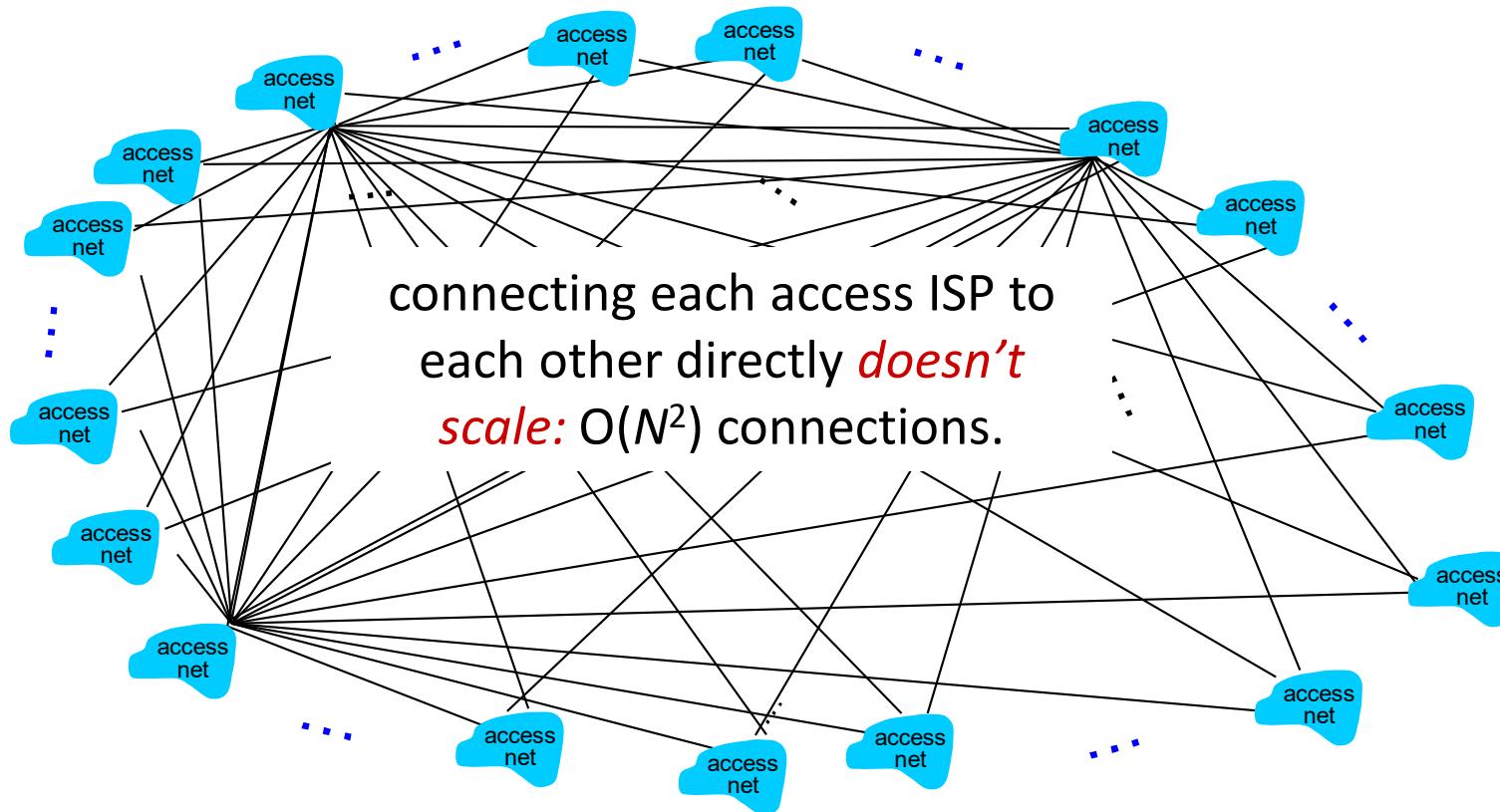
# Internet structure: a “network of networks”

**Question:** given millions of access ISPs, how to connect them together?



# Internet structure: a “network of networks”

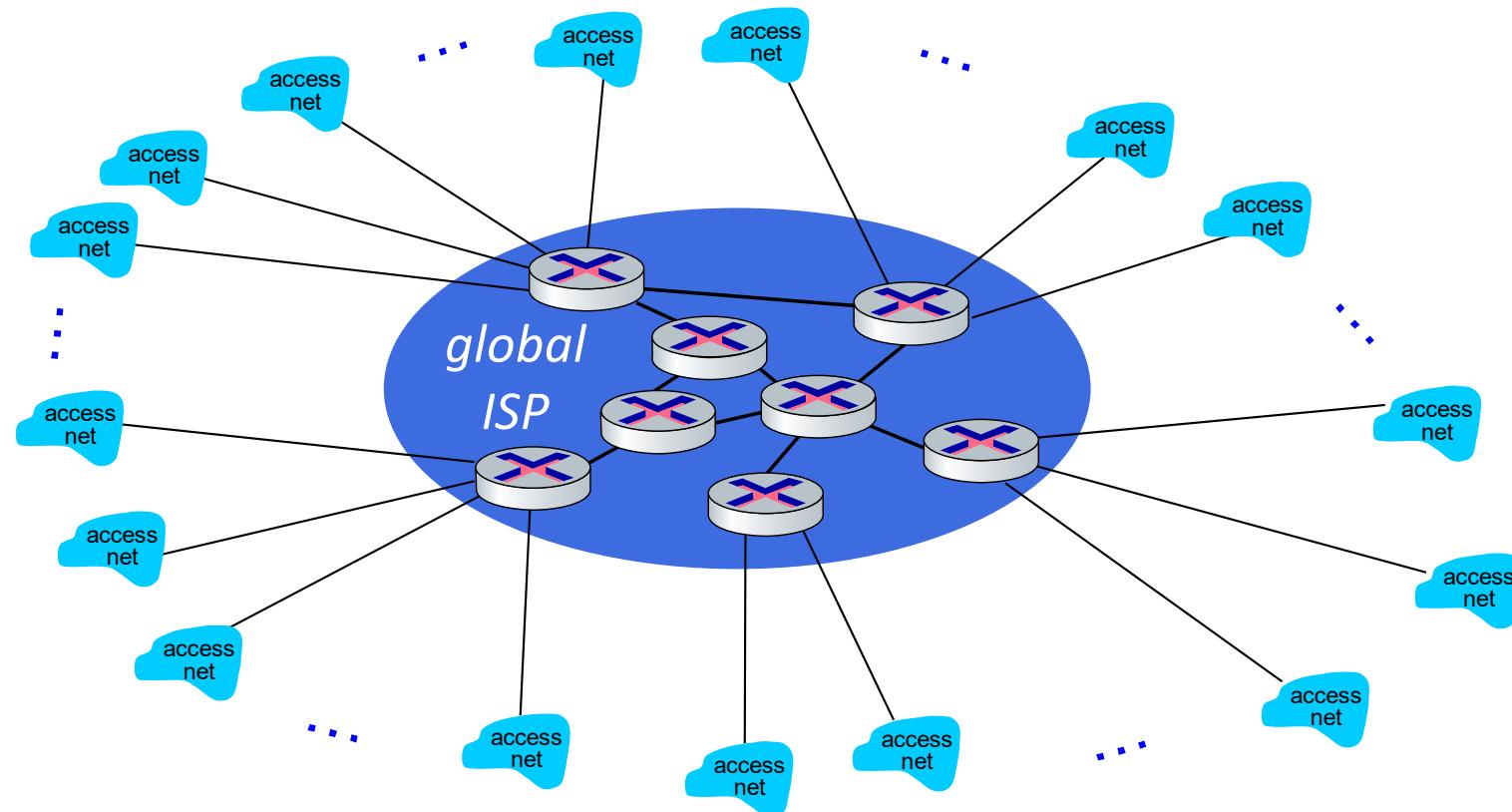
*Question:* given *millions* of access ISPs, how to connect them together?



# Internet structure: a “network of networks”

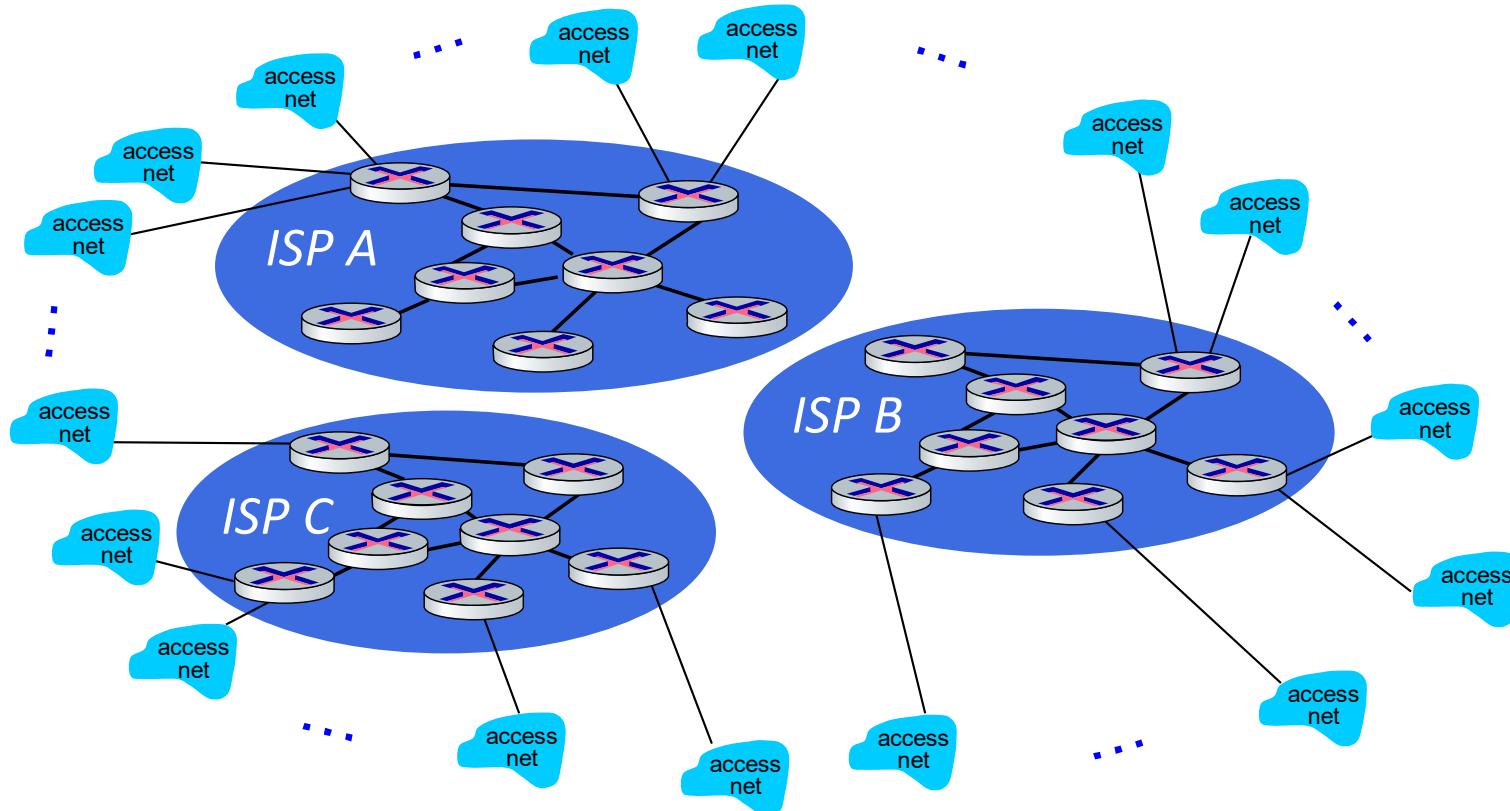
*Option:* connect each access ISP to one global transit ISP?

*Customer and provider* ISPs have economic agreement.



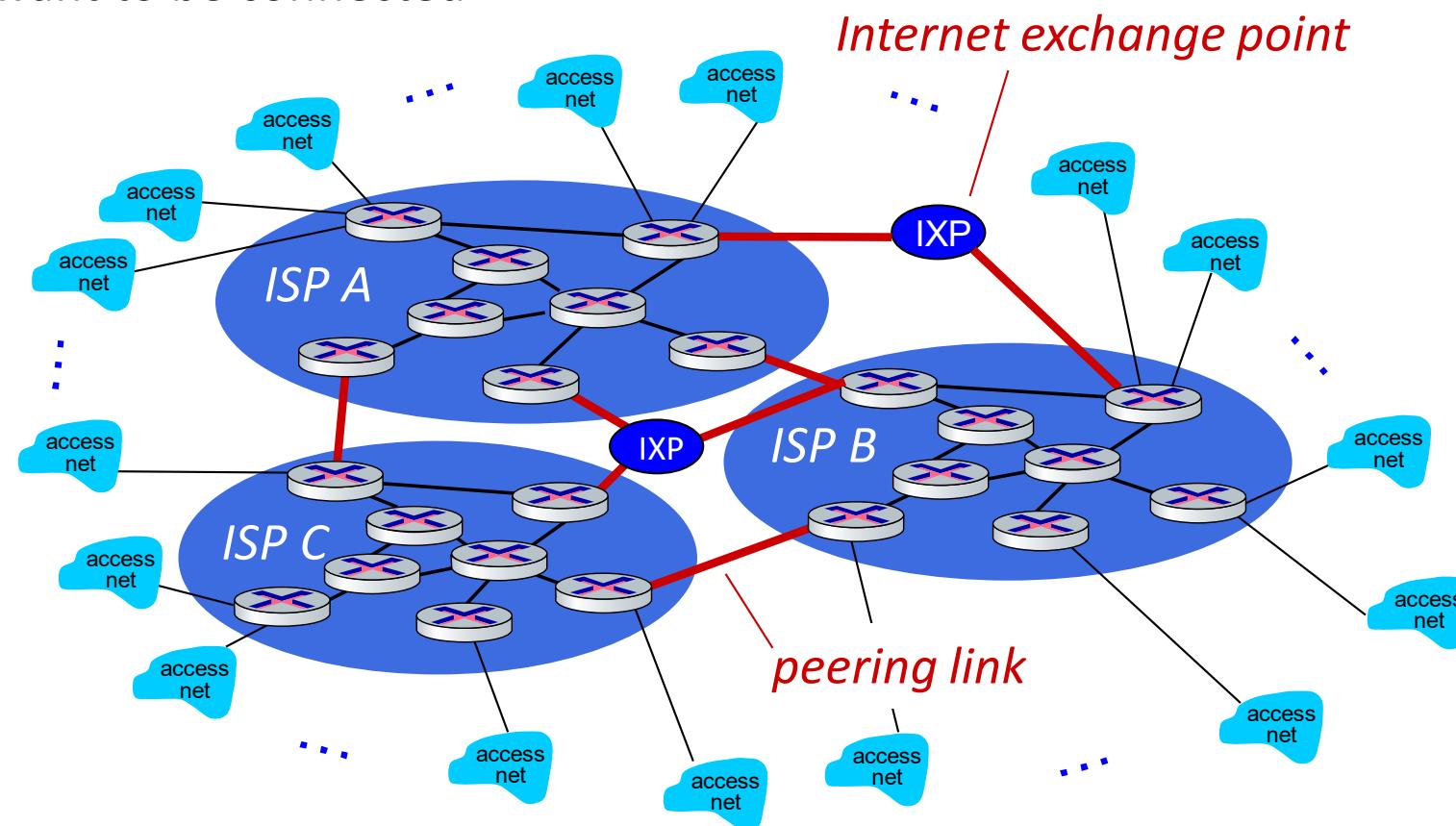
# Internet structure: a “network of networks”

But if one global ISP is viable business, there will be competitors ....



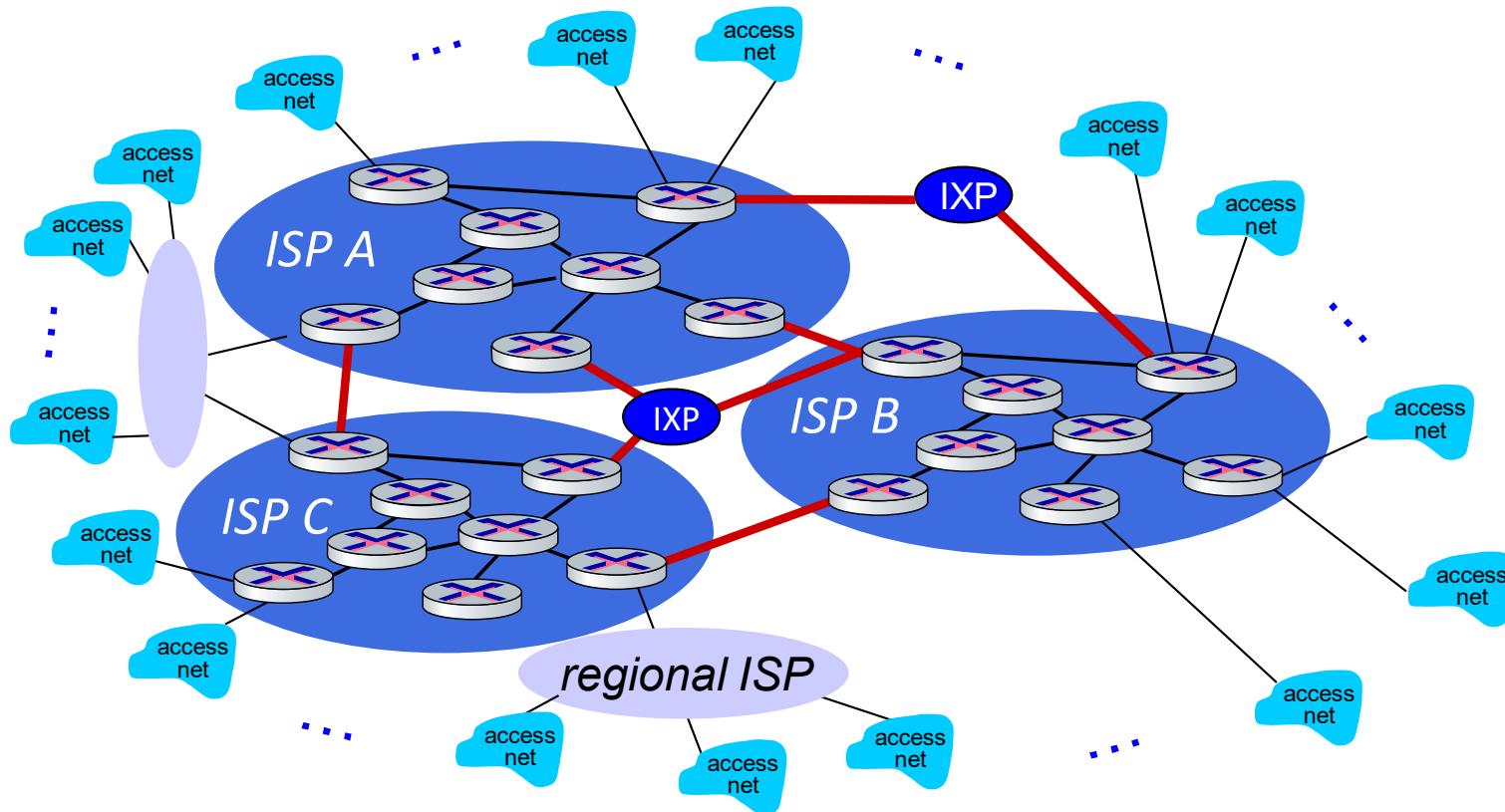
# Internet structure: a “network of networks”

But if one global ISP is viable business, there will be competitors .... who will want to be connected



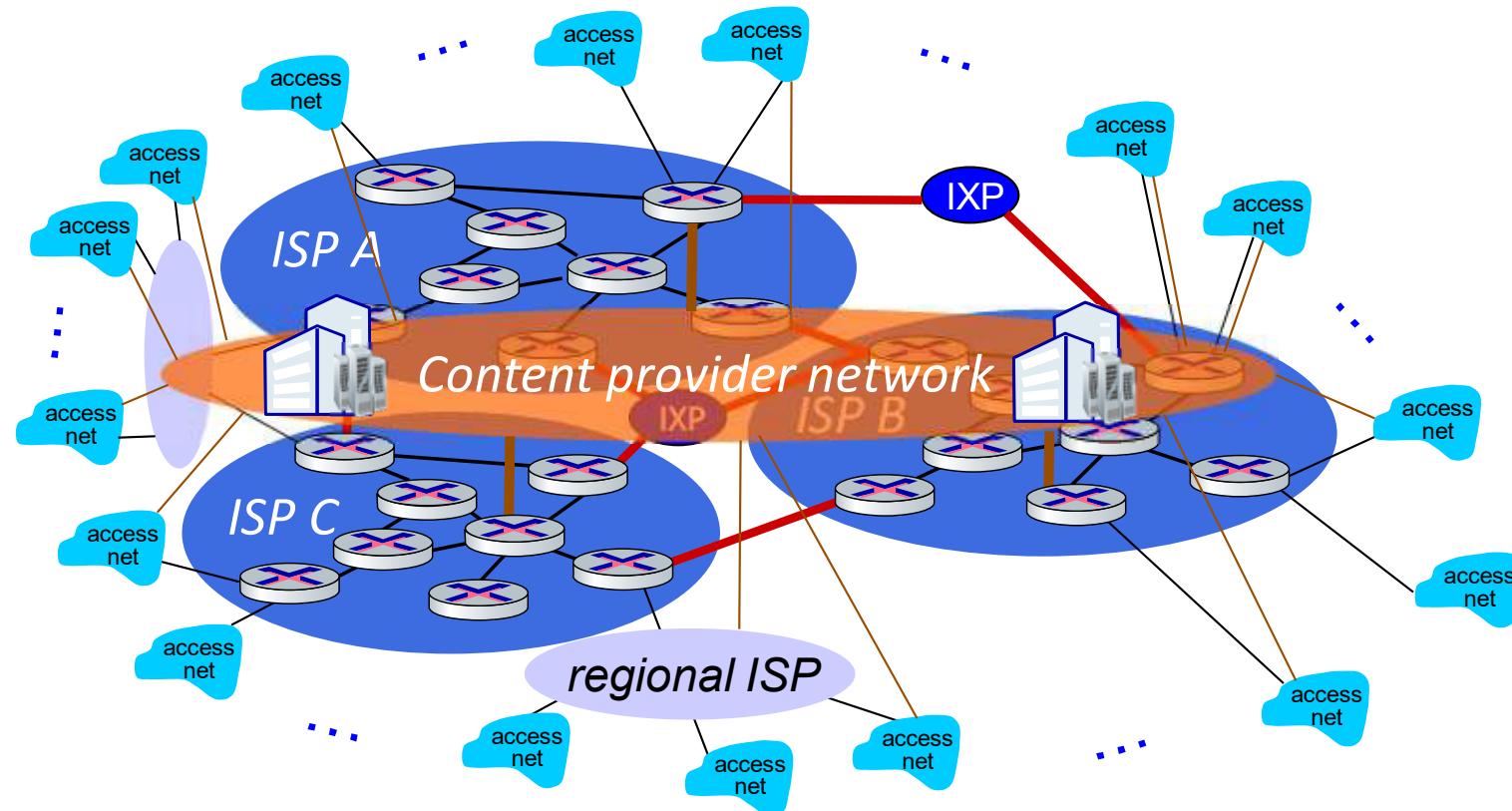
# Internet structure: a “network of networks”

... and regional networks may arise to connect access nets to ISPs

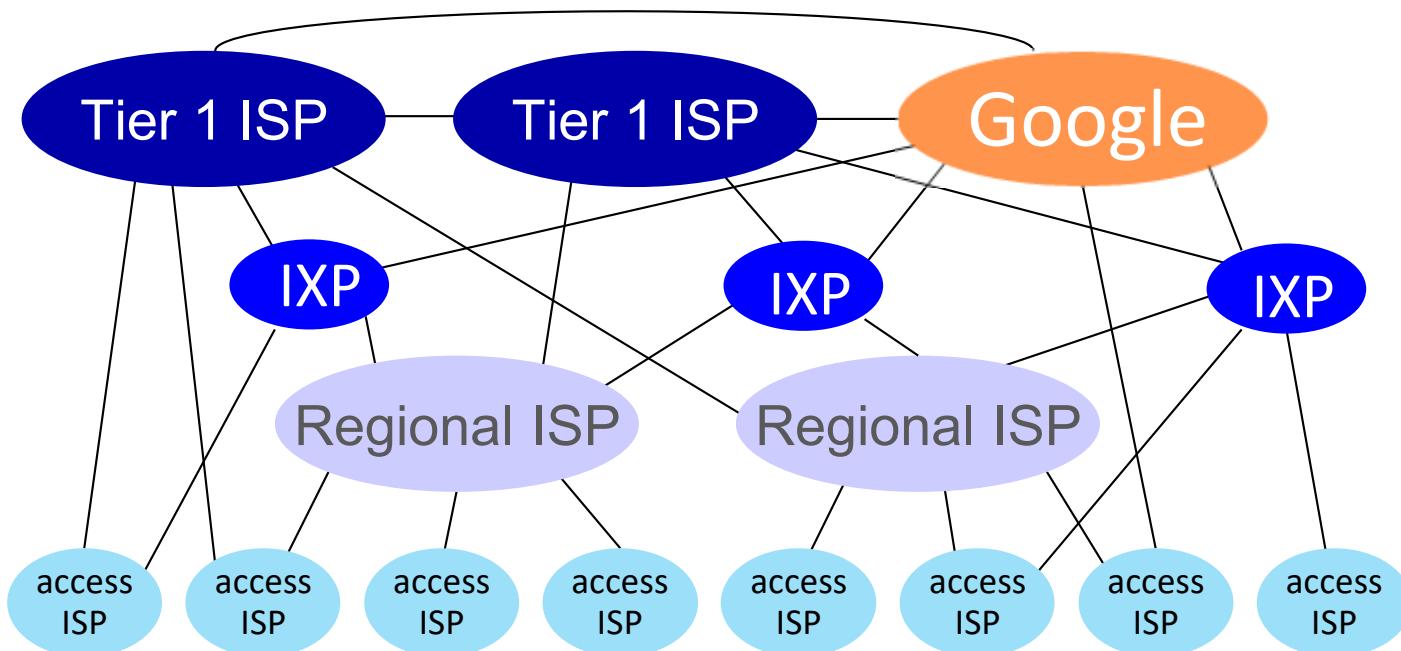


# Internet structure: a “network of networks”

... and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users



# Internet structure: a “network of networks”



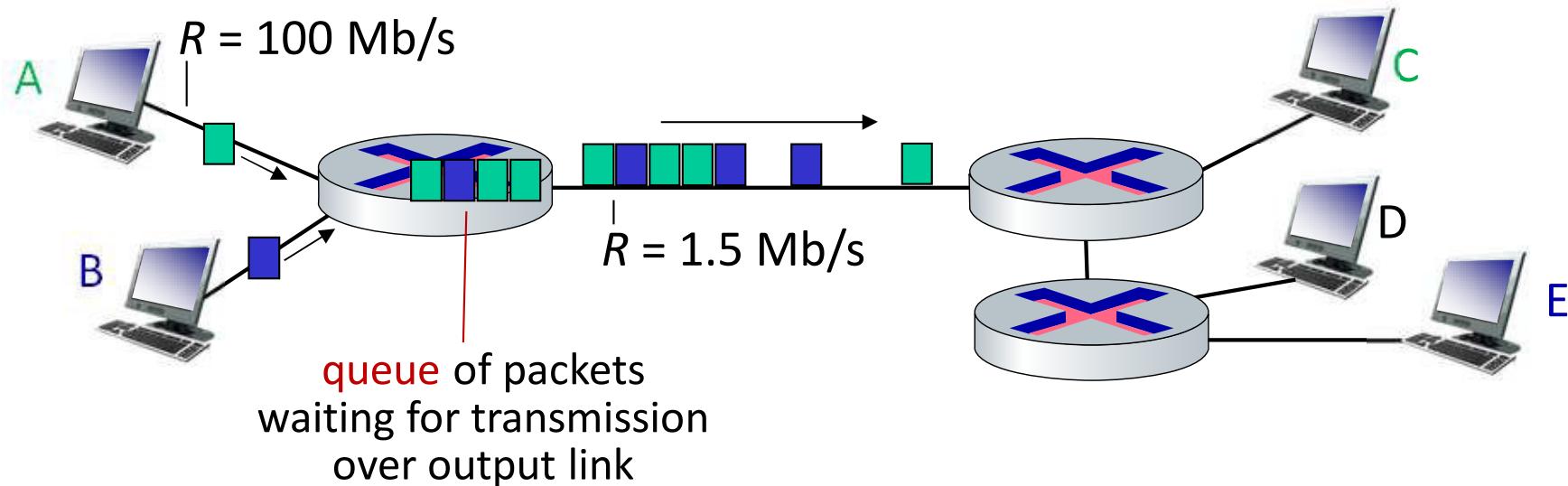
At “center”: small # of well-connected large networks

- **“tier-1” commercial ISPs** (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
- **content provider networks** (e.g., Google, Facebook): private network that connects its data centers to Internet, often bypassing tier-1, regional ISPs

# Objectives

- Terminology
- What is a protocol?
- Network edge (hosts, access net, physical media)
- Network core (circuit/packet switching, Internet structure)
- **Performance metrics:**
  - Loss
  - Delay
  - Throughput
- Protocol layers, service models

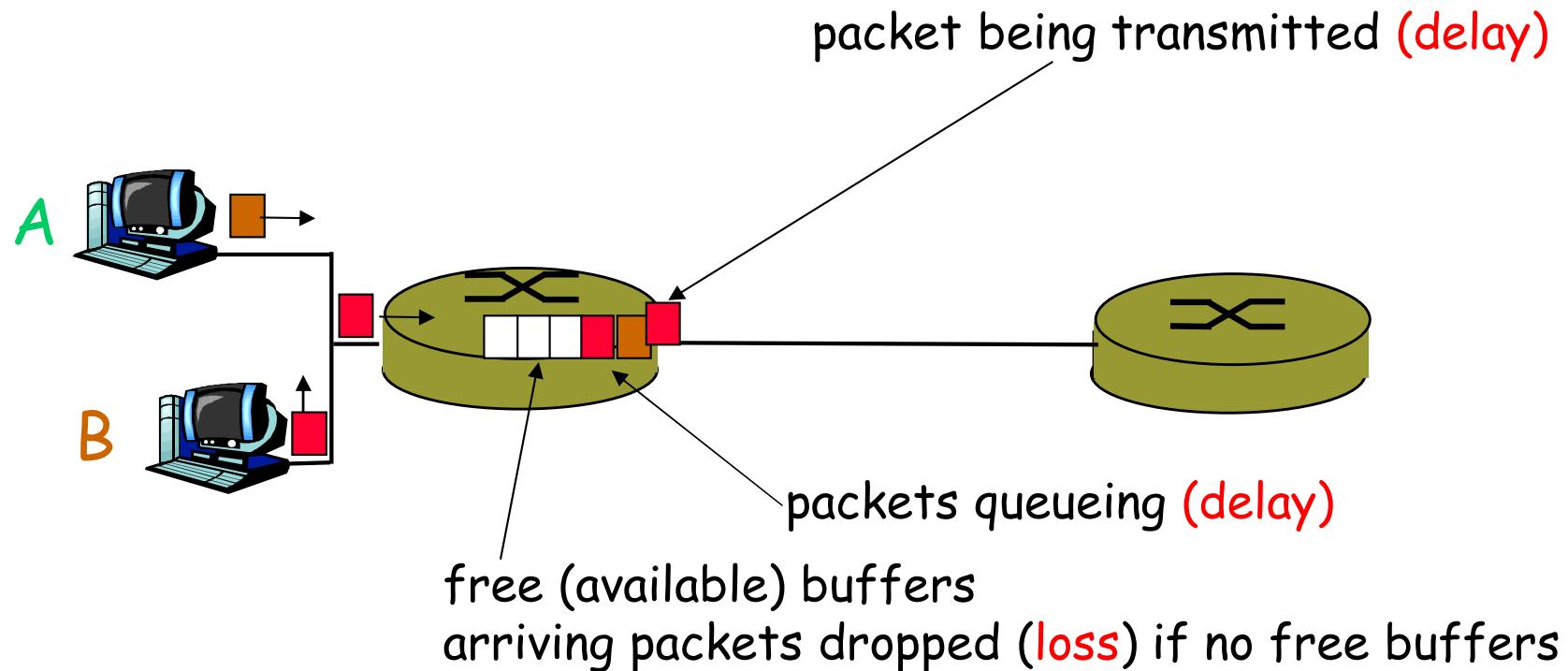
# Packet Switching: Delay and Loss



**Queueing** occurs when work arrives faster than it can be serviced



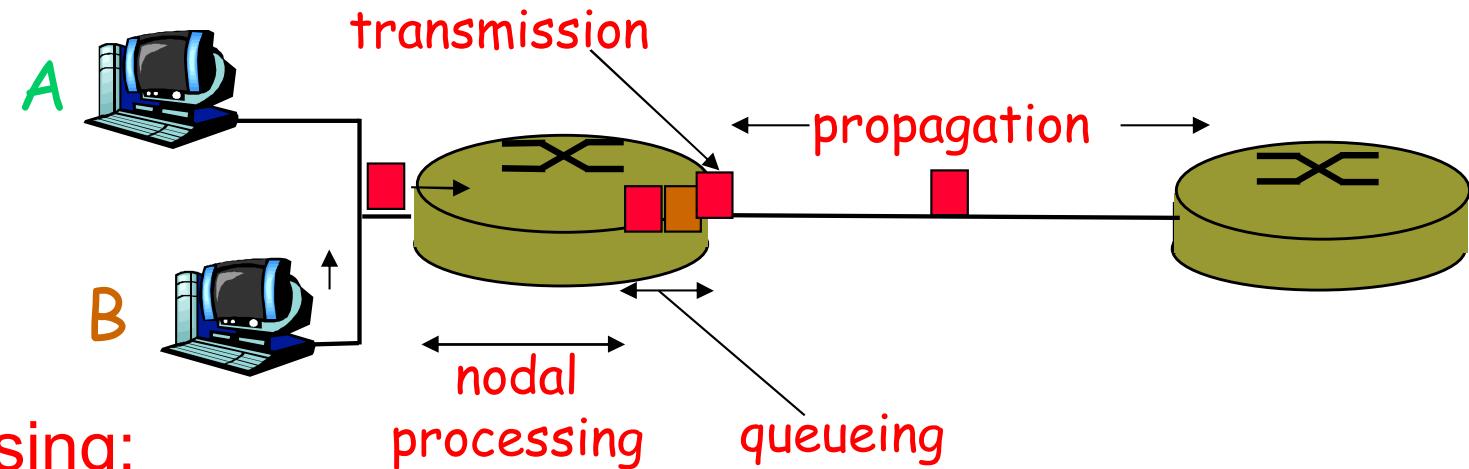
# Packet Switching: Delay and Loss



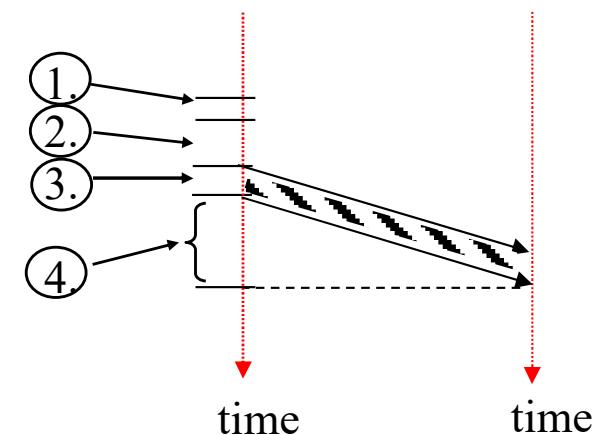
Arrival rate > transmission rate:

- packets will queue, waiting to be transmitted on output link
- packets can be dropped (**lost**) if memory (buffer) in router fills up

# Delay in Packet Switched Networks



- 1. Nodal processing:
  - Check bit errors;
  - Determine output link;
  - ...
- 2. Queueing
  - Time waiting for output link for transmission;
  - Depends on congestion level of router.



# Delay in Packet Switched Networks

## 3. Transmission delay:

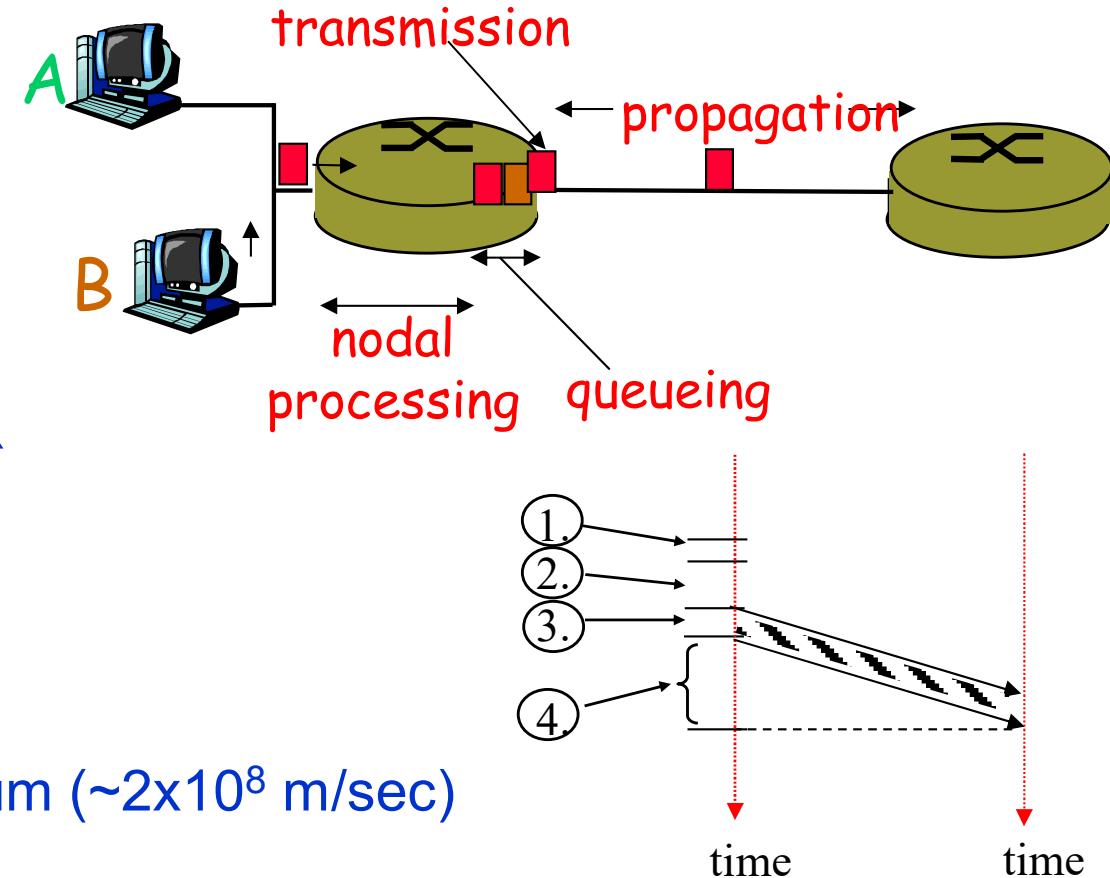
- $R$  = link bandwidth (bps);
- $L$  = packet length (bits).

Time to send bits into link =  $L/R$

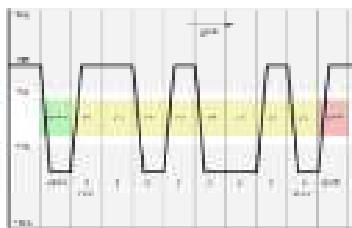
## 4. Propagation delay:

- $d$  = length of physical link
- $s$  = propagation speed in medium ( $\sim 2 \times 10^8$  m/sec)

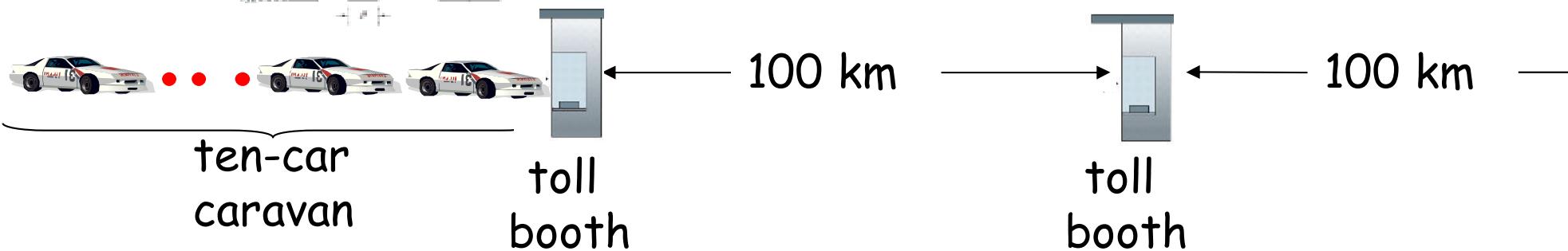
Propagation delay =  $d/s$



Note:  $s$  and  $R$  are very different quantities!

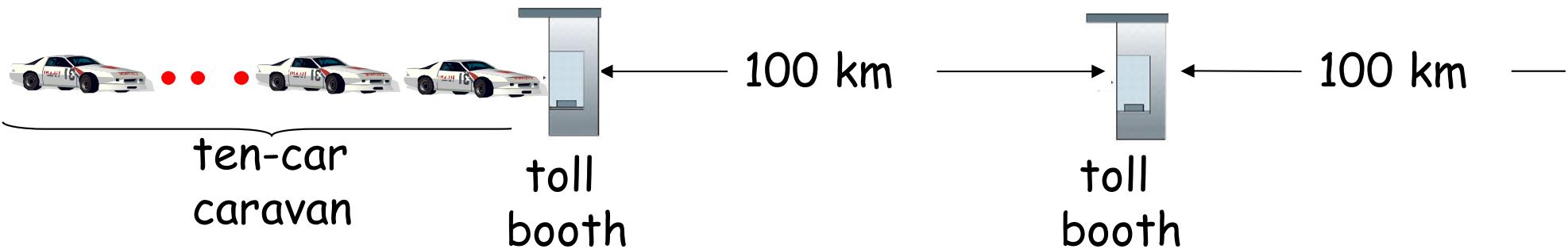


## *Caravan Analogy*



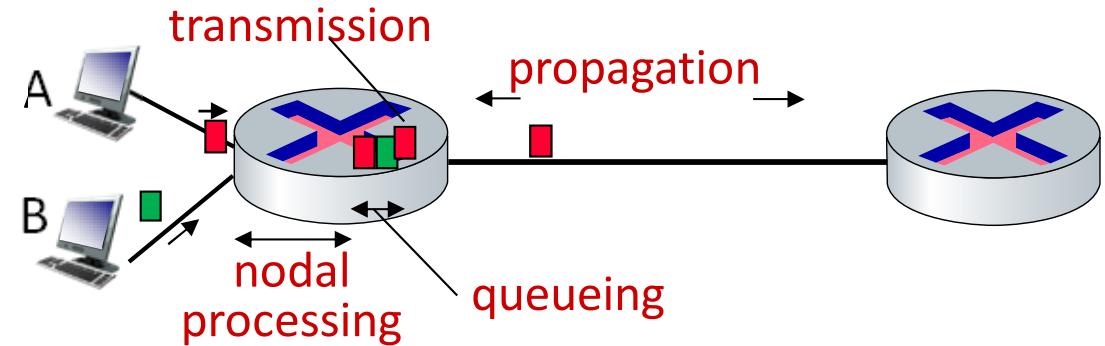
- car ~ bit;
  - caravan ~ packet  
(always travel together);
  - toll booth takes 12 sec to service one car (transmission time);
  - cars “propagate” at 100 km/h;
  - Q: How long until caravan is lined up before 2<sup>nd</sup> toll booth?
    - Time to “push” entire caravan through toll booth onto highway =  $12 \times 10 = 120$  sec  
(transmission time: L/R)
    - Time for last car to propagate (d/s) from 1<sup>st</sup> to 2<sup>nd</sup> toll both:  
 $100\text{km}/(100\text{km/h}) = 1\text{ h}$
    - A: 62 minutes

## Caravan Analogy



- Cars now “propagate” at 1000 km/h;
- Toll booth now takes 1 min / car;
- Q: Will cars arrive to 2<sup>nd</sup> booth before all cars serviced at 1<sup>st</sup> booth?
  - Yes! After 7 min, 1<sup>st</sup> car at 2<sup>nd</sup> booth and 3 cars still at 1<sup>st</sup> booth.
  - 1<sup>st</sup> bit of packet can arrive at 2<sup>nd</sup> router before packet is fully transmitted at 1<sup>st</sup> router!
- See applet at AWL Web site:  
[https://media.pearsoncmg.com/aw/ecs\\_kurose\\_comnetwork\\_7/cw/content/interactiveanimations/transmission-vs-propogation-delay/transmission-propagation-delay-ch1/index.html](https://media.pearsoncmg.com/aw/ecs_kurose_comnetwork_7/cw/content/interactiveanimations/transmission-vs-propogation-delay/transmission-propagation-delay-ch1/index.html)

# Nodal Delay



$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

$d_{\text{proc}}$  - processing delay

- typically < microsecs

$d_{\text{queue}}$  - queuing delay

- depends on congestion

$d_{\text{trans}}$  - transmission delay

- $d_{\text{trans}} = L/R$ , significant for low-speed links

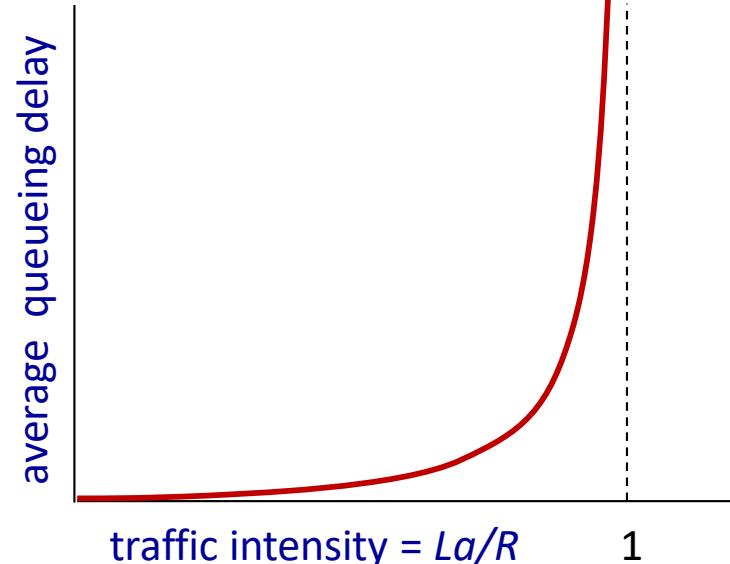
$d_{\text{prop}}$  - propagation delay

- $d_{\text{prop}} = d/s$ , a few microsecs to hundreds of msecs

$d_{\text{trans}}$  and  $d_{\text{prop}}$   
are very different

# Queueing Delay

- $R$  = link bandwidth (bps)
- $L$  = packet length (bits)
- $a$  = average packet arrival rate



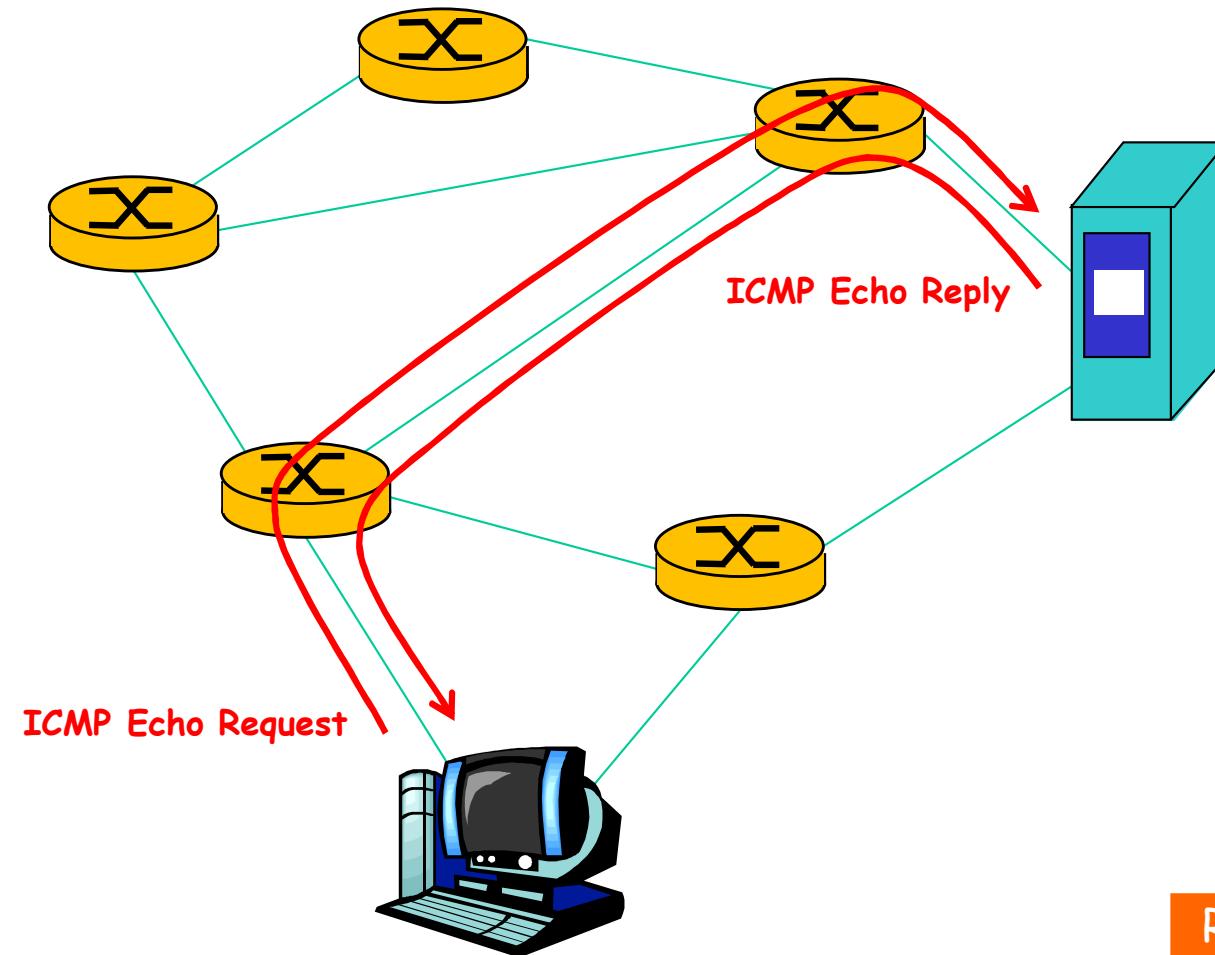
**Traffic intensity =  $L.a/R$**

- $L.a/R \sim 0$ : average queueing delay small
- $L.a/R \rightarrow 1$ : delays become (very) big
- $L.a/R > 1$ : more “work” arriving than can be serviced - average delay infinite!



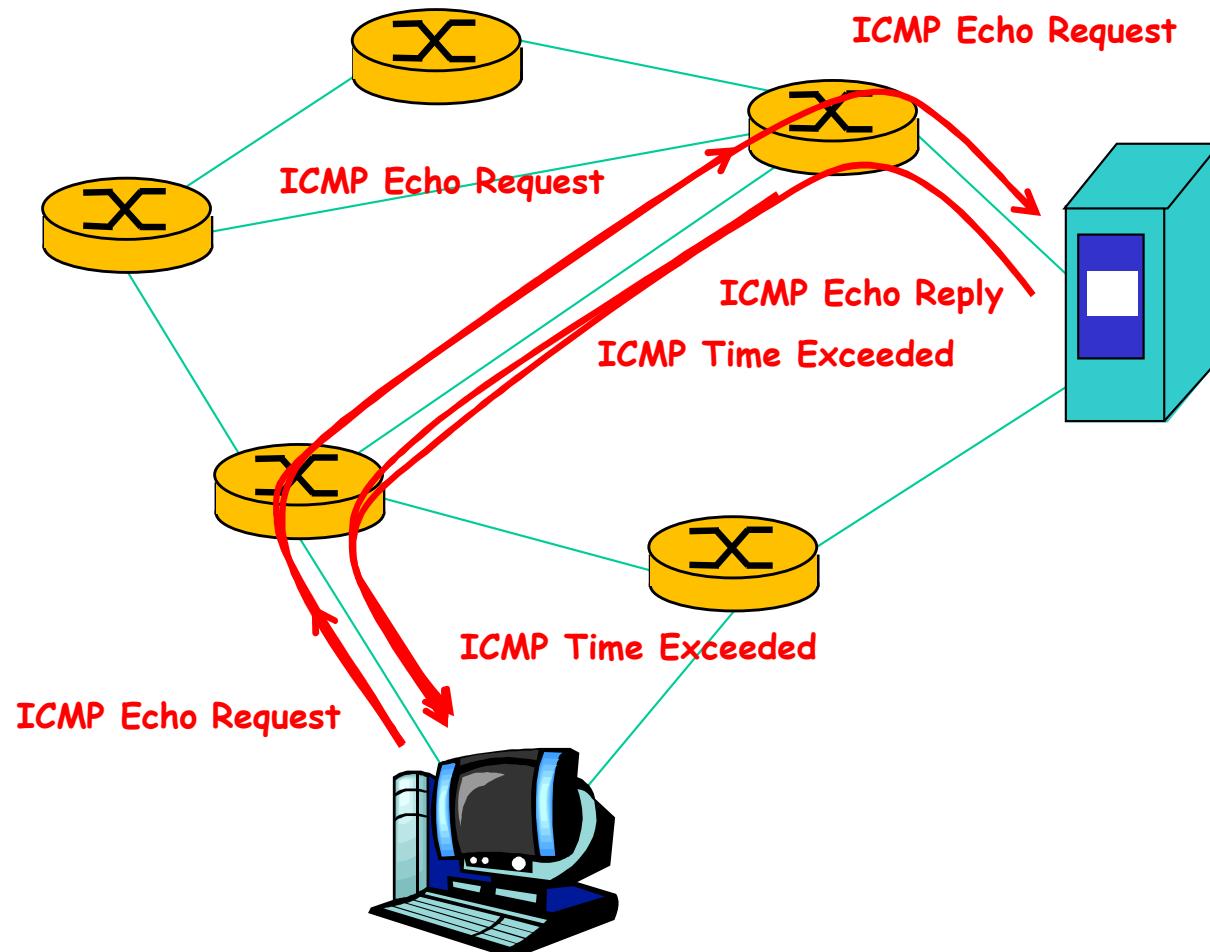
$La/R \rightarrow 1$

# Testing Internet Destinations: the *ping* Command



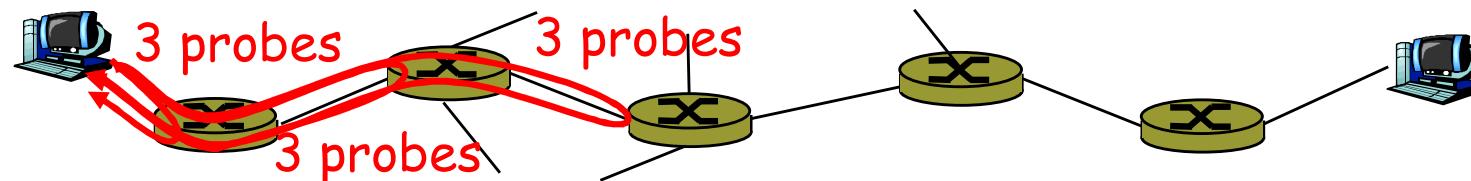
Run [ping](#)

# Discovering Internet Routes: traceroute



# “Real” Internet Delays and Routes

- What do “real” Internet delay & loss look like?
  - **Traceroute program:** (windows: *tracert*; linux: *traceroute*)
    - Provides delay measurement from source to router along end-end Internet path towards destination.
- For all  $i$ :
- Sends three packets that will reach router  $i$  on path towards destination;
  - Router  $i$  will return packets to sender;
  - Sender times interval between transmission and reply.



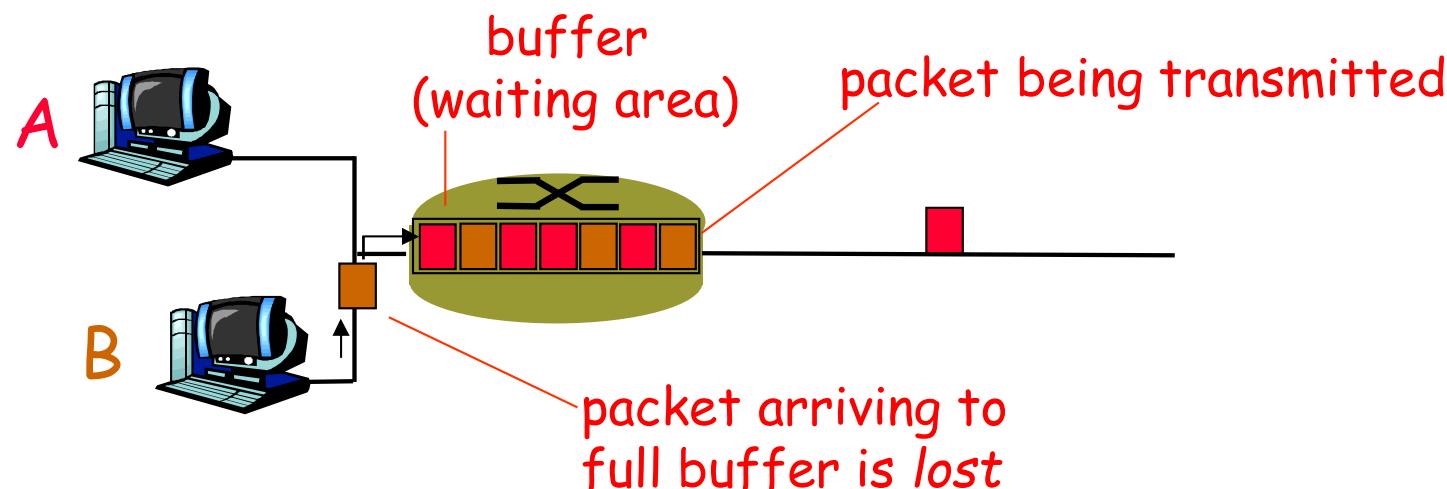
# *“Real” Internet Delays and Routes*

traceroute: gaia.cs.umass.edu to www.eurecom.fr

Three delay measurements from gaia.cs.umass.edu to cs-gw.cs.umass.edu						
1	cs-gw (128.119.240.254)	1 ms	1 ms	2 ms		
2	border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145)	1 ms	1 ms	2 ms		
3	cht-vbns.gw.umass.edu (128.119.3.130)	6 ms	5 ms	5 ms		
4	jn1-at1-0-0-19.wor.vbns.net (204.147.132.129)	16 ms	11 ms	13 ms		
5	jn1-so7-0-0-0.wae.vbns.net (204.147.136.136)	21 ms	18 ms	18 ms		
6	abilene-vbns.abilene.ucaid.edu (198.32.11.9)	22 ms	18 ms	22 ms		
7	nycm-wash.abilene.ucaid.edu (198.32.8.46)	22 ms	22 ms	22 ms		
8	62.40.103.253 (62.40.103.253)	104 ms	109 ms	106 ms		
9	de2-1.de1.de.geant.net (62.40.96.129)	109 ms	102 ms	104 ms		
10	de.fr1.fr.geant.net (62.40.96.50)	113 ms	121 ms	114 ms		
11	renater-gw.fr1.fr.geant.net (62.40.103.54)	112 ms	114 ms	112 ms		
12	nio-n2.cssi.renater.fr (193.51.206.13)	111 ms	114 ms	116 ms		
13	nice.cssi.renater.fr (195.220.98.102)	123 ms	125 ms	124 ms		
14	r3t2-nice.cssi.renater.fr (195.220.98.110)	126 ms	126 ms	124 ms		
15	eurecom-valbonne.r3t2.ft.net (193.48.50.54)	135 ms	128 ms	133 ms		
16	194.214.211.25 (194.214.211.25)	126 ms	128 ms	126 ms		
17	***					
18	***				*	means no response (probe lost, router not replying)
19	fantasia.eurecom.fr (193.55.113.142)	132 ms	128 ms	136 ms		

# Packet Loss

- Queue (buffer) preceding link has finite capacity;
- Packet arriving to full queue is dropped (lost);
- Lost packet may be retransmitted:
  - By previous node, by source end system, or not at all.

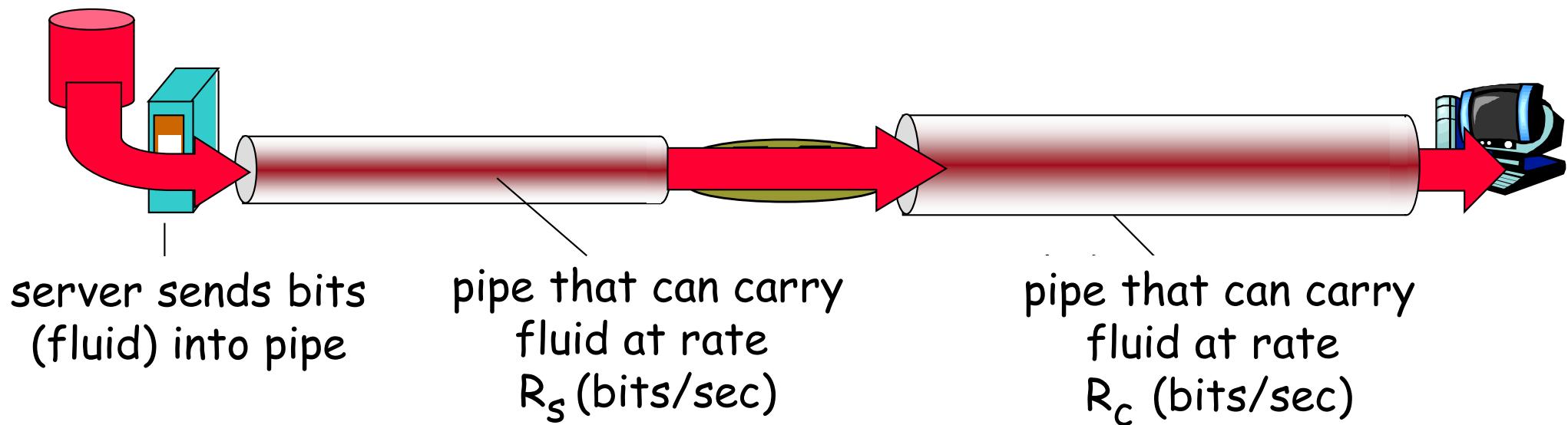


See applet at AWL Web site:

[https://media.pearsoncmg.com/aw/ecs\\_kurose\\_compnetwork\\_7/cw/content/interactiveanimations/queuing-loss-applet/index.html](https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/queuing-loss-applet/index.html)

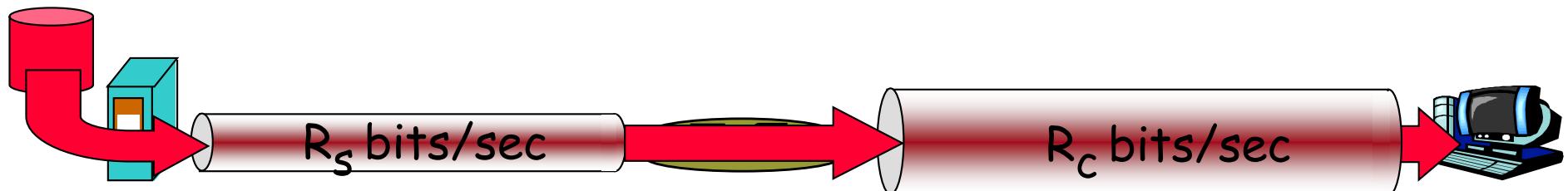
# Throughput

- *Throughput*: rate (bits/time unit) at which bits are transferred between sender and receiver:
  - *Instantaneous*: rate at a given instant;
  - *Average*: rate over a longer period of time.

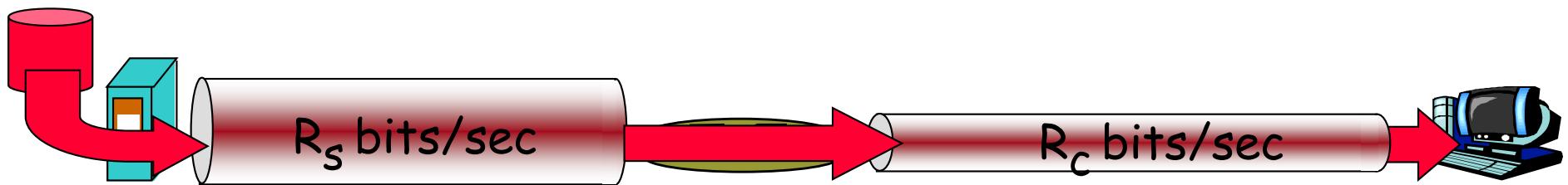


# Throughput

$R_s < R_c$  What is average end-end throughput?



$R_s > R_c$  What is average end-end throughput?

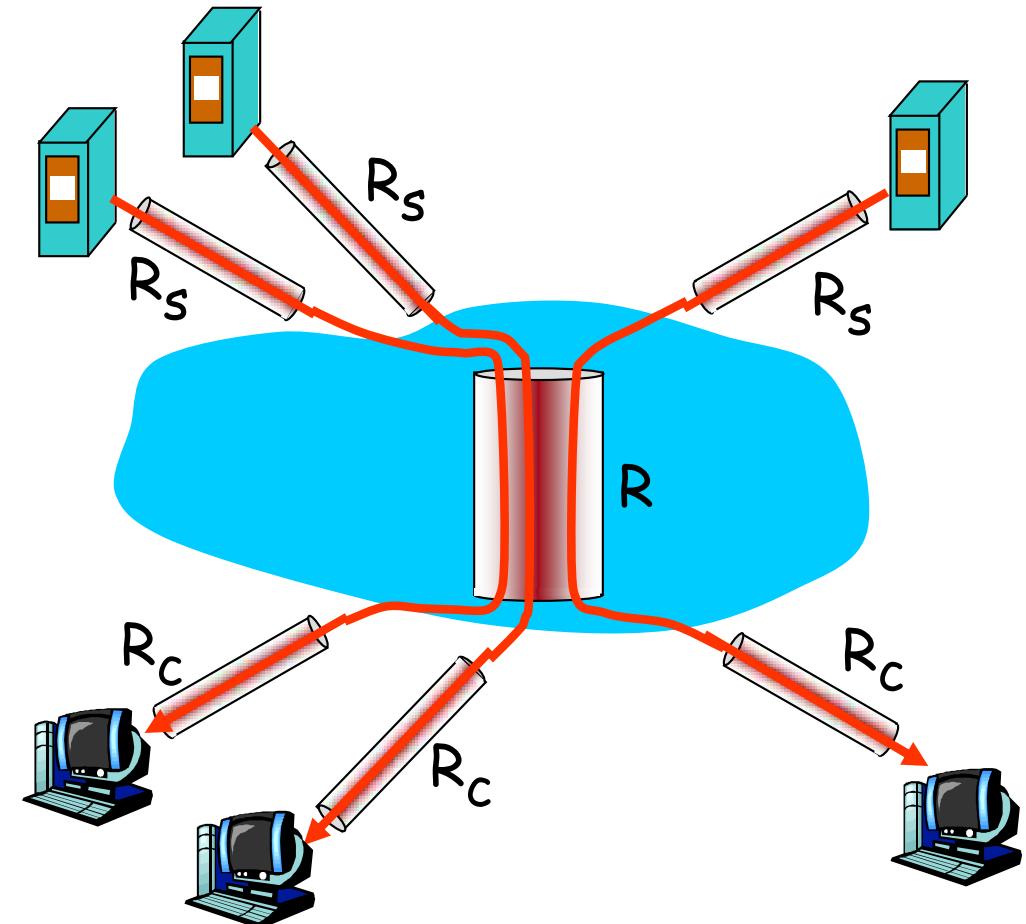


Bottleneck link:

the link on end-end path that constrains end-end throughput.

## Throughput: Internet Scenario

- Per-connection end-end throughput:  
 $\min(R_c, R_s, R/10)$
- In practice:  
 $R_c$  or  $R_s$  is often bottleneck



10 connections (fairly) share backbone bottleneck link  $R$  bits/sec

# *Objectives*

- Terminology
- What is a protocol?
- Network edge (hosts, access net, physical media)
- Network core (circuit/packet switching, Internet structure)
- Performance metrics (Loss, Delay, Throughput)
- **Protocol layers and Service models**

# Protocol “Layers”

Networks are complex!

- Many “pieces”:
  - hosts
  - routers
  - links of various media
  - applications
  - protocols
  - hardware, software

Question:

Is there any hope of *organizing* network structure?

# Network Architecture: *Needed Functionality*

## Functionality required:

- Mechanical specification of plugs, modulation type, ...
- Segmentation, reconstruction and delimitation of packets;
- Multiplexing/demultiplexing;
- Error and flow control;
- Routing;
- Congestion control;
- Data presentation formatting;
- Authentication;
- ...

## Modular approach:

- Easier to design and to understand;
- Flexibility, possibility of module interface standardization.

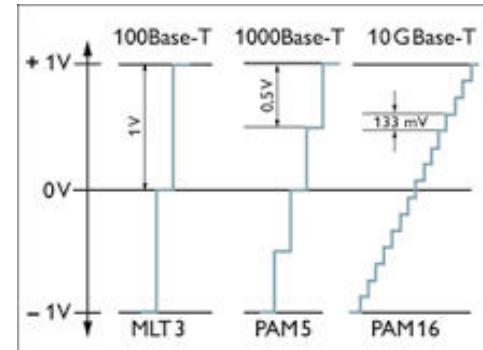


Figure 7. Voltage coding for Ethernet: Lower signal clearances for Gigabit Ethernet with PAM (Pulse Amplitude Modulation) increase the risk of disturbances when compared to Fast Ethernet with MLT (Multi Link Trunk).



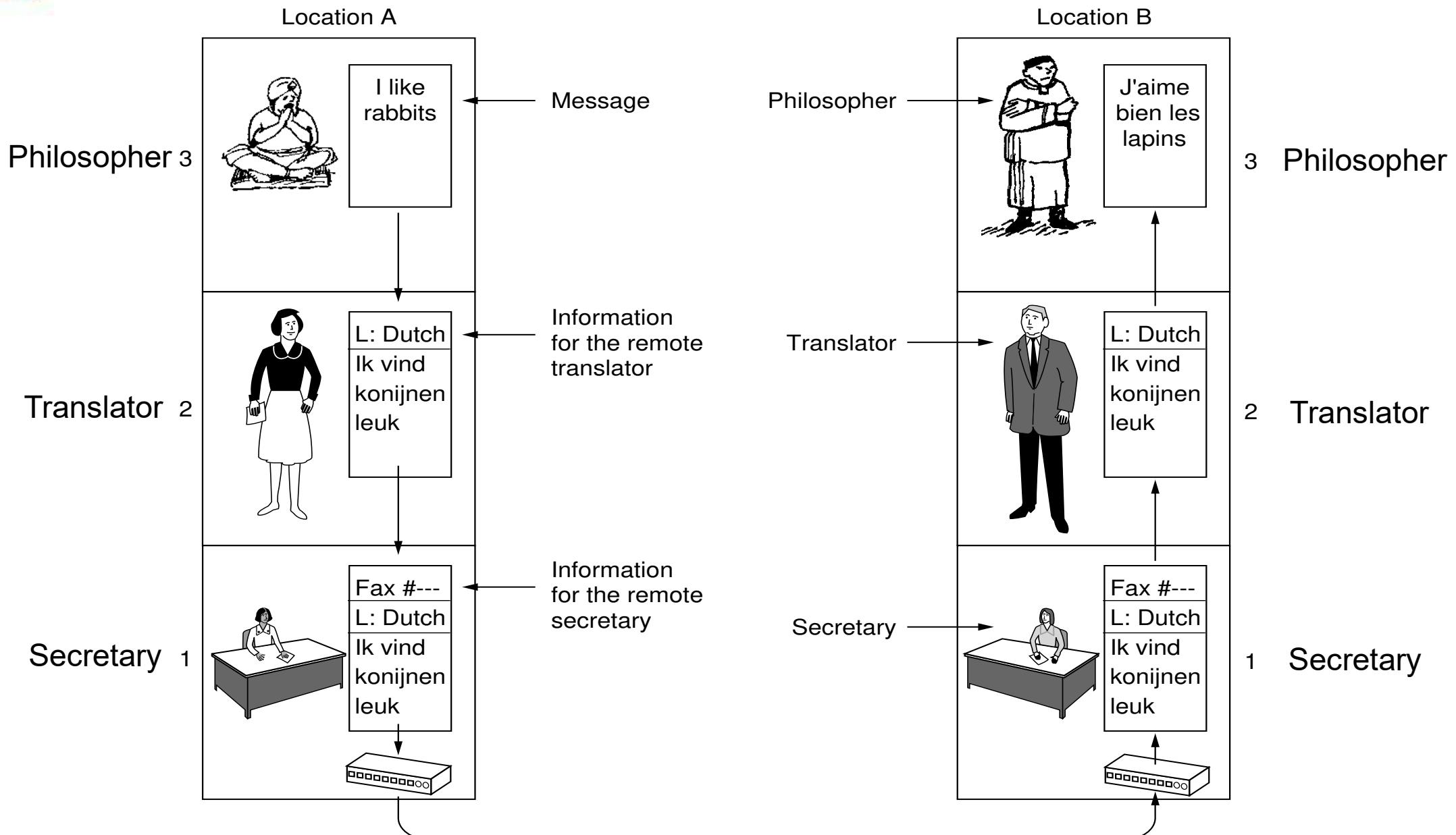
# Why Layering?

Dealing with complex systems:

- Explicit structure allows identification and perceiving the 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 access network doesn't affect rest of system.
- Layering opposition:
  - Duplication of functions;
  - Violations of layer separation principle.



# Layered Communication System



# *Organization of Air Travel*



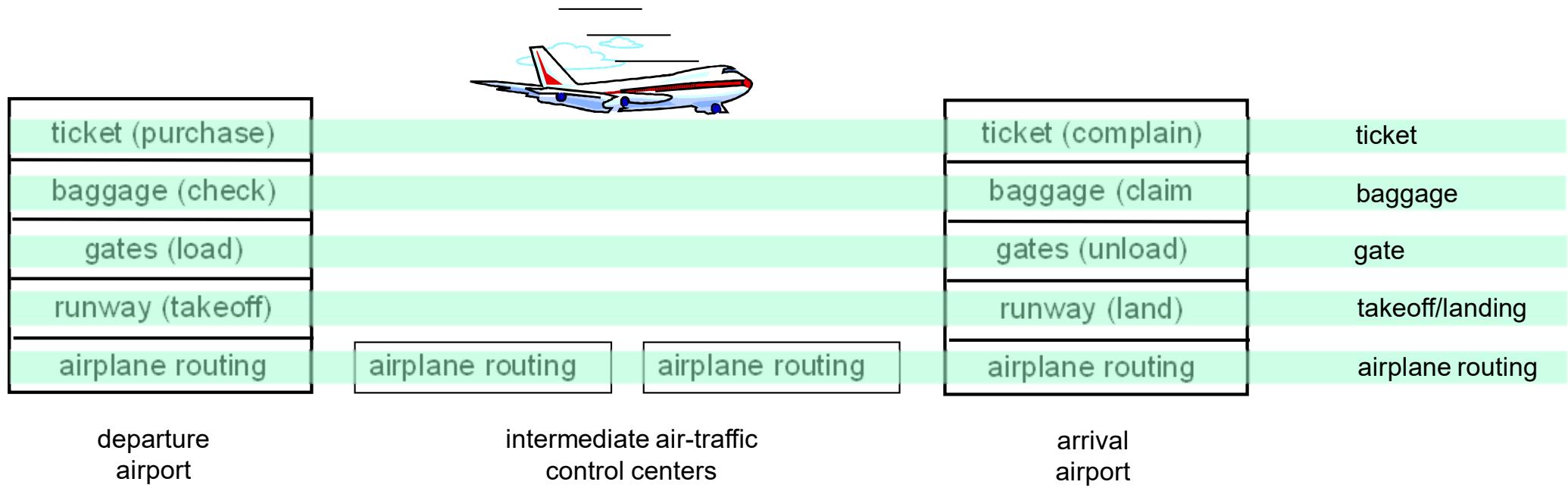
ticket (purchase)  
baggage (check)  
gates (load)  
runway takeoff  
airplane routing

ticket (complain)  
baggage (claim)  
gates (unload)  
runway landing  
airplane routing

airplane routing

- A series of steps

# *Layering of Airline Functionality*



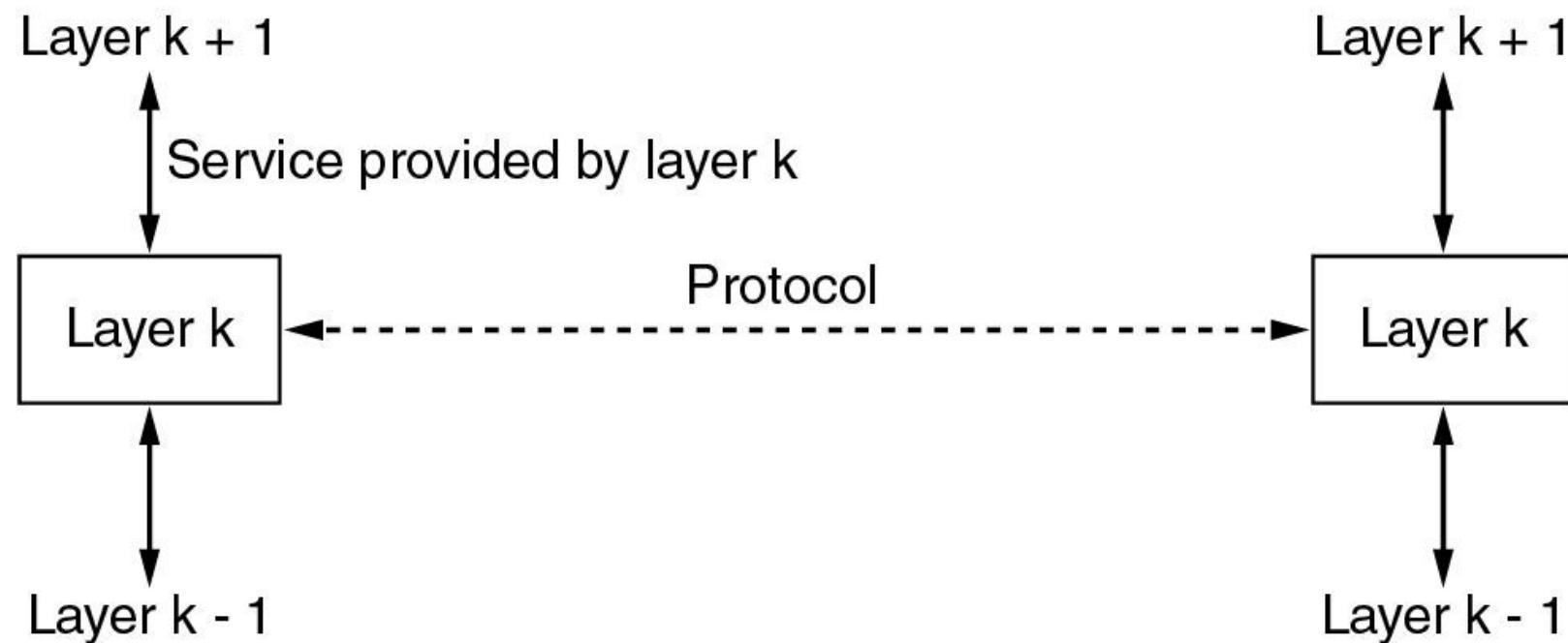
**Layers – each layer implements a service:**

- Via its own **internal-layer actions**;
- Relying on **services provided by layer below**.

# Network Architecture

**Protocol:** set of rules to be followed between two peer entities, to ensure a successful data exchange;

**Service:** each layer offers a service to the layer above and uses the services made available by the layer below.



# *Network Architecture: Protocols*

**Peer entities** of the same layer execute a distributed algorithm;

**Protocols** define the communication rules between peer entities:

- Format of the exchanged messages;
- Sequence to follow when sending and receiving messages;
- Actions to take when a message is sent or received;

Messages used by layer  $n$  protocol:  **$n$ -PDU (Protocol Data Unit)**:

- Header;
- Payload;
- Trailer.

# *Network Architecture: Service Interface*

A **service interface** specifies which services are offered by layer  $n$  to layer  $n+1$ ;

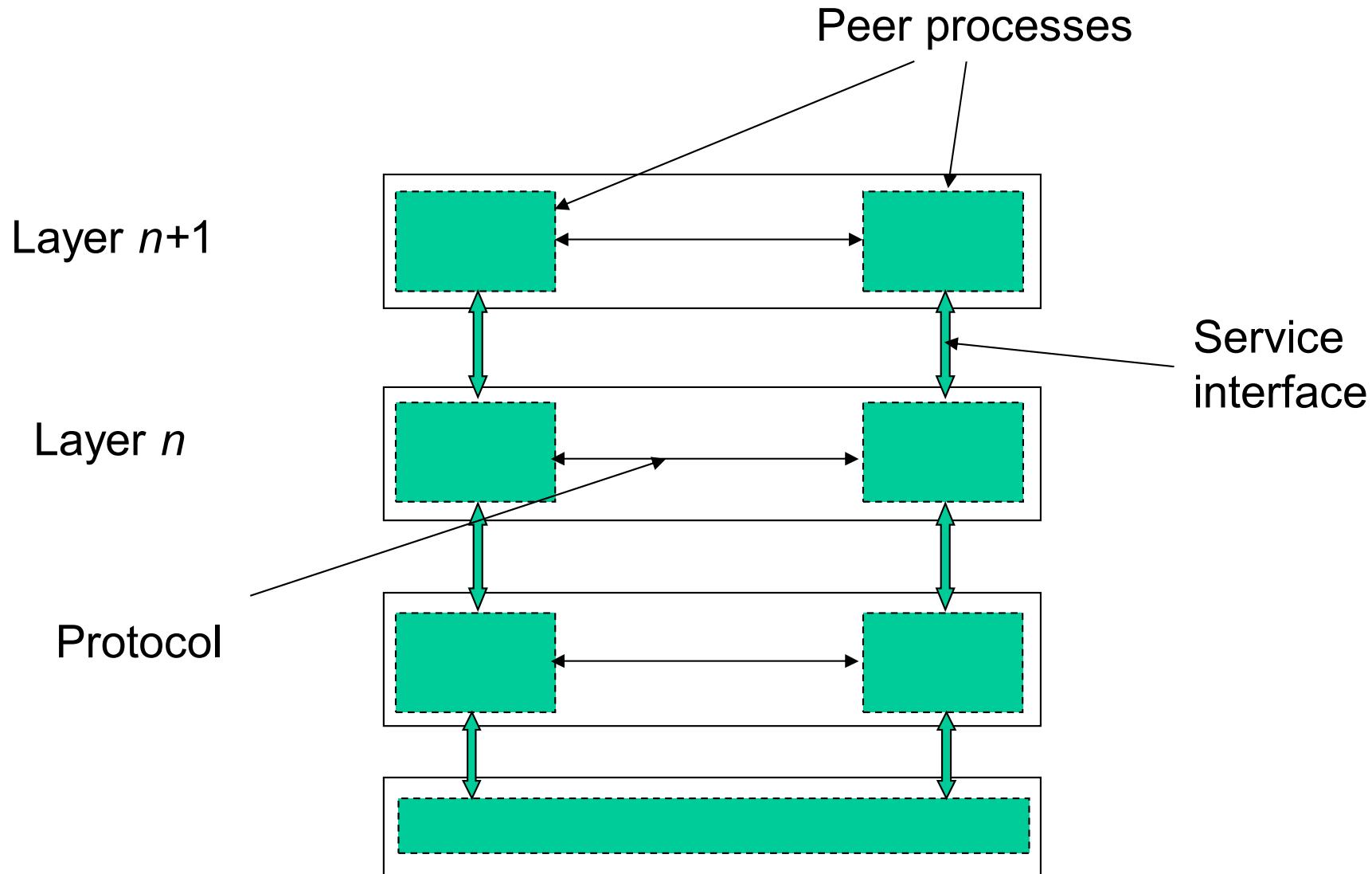
Connection-oriented service:

- Session establishment;
- Message exchange;
- Session termination;

Connectionless service:

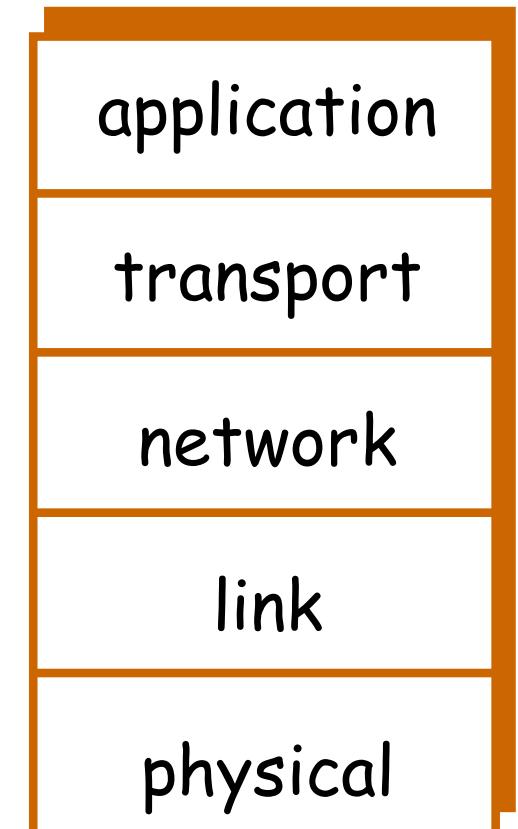
- No session establishment or termination.

# *Network Architecture: Layered Architecture*

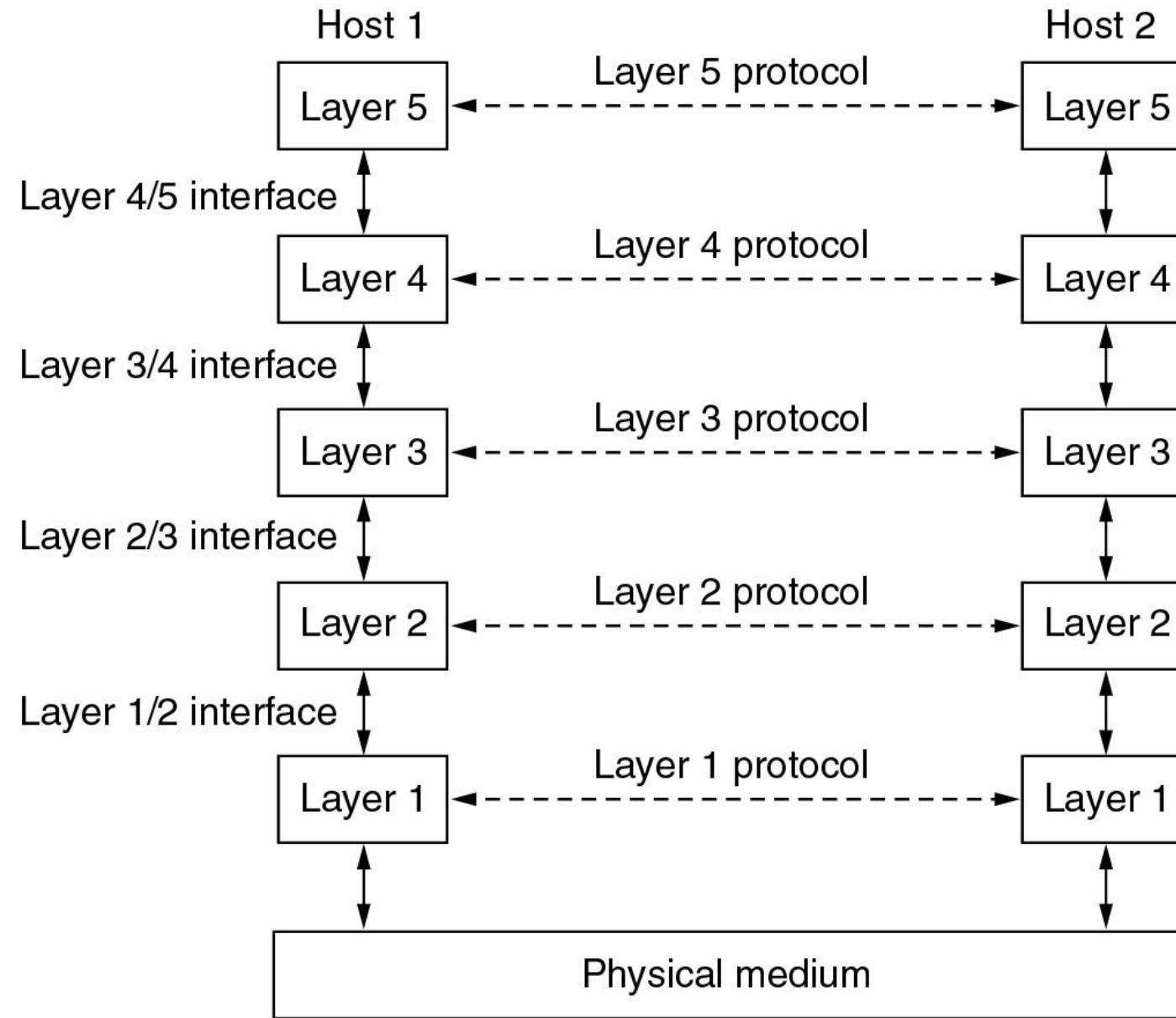


# *Internet Protocol Stack*

- **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
  - PPP, Ethernet
- **Physical:** bits “on the wire”
  - RS-232c, V.92

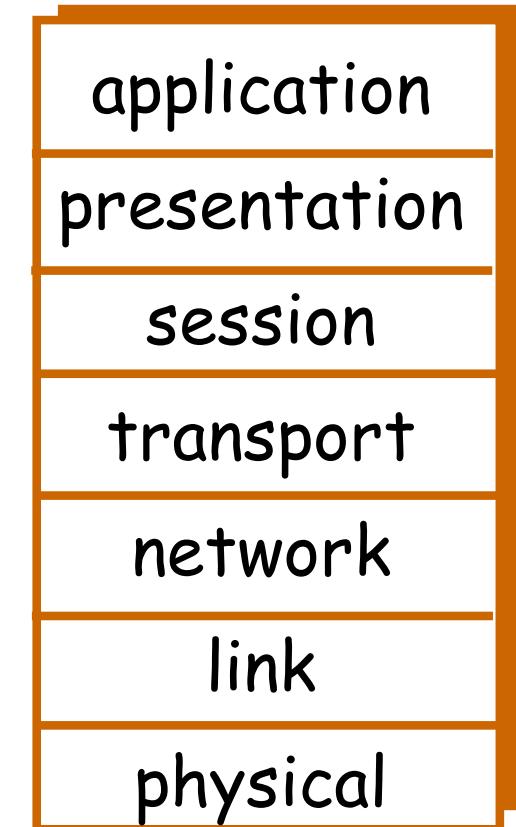


# Network Architecture

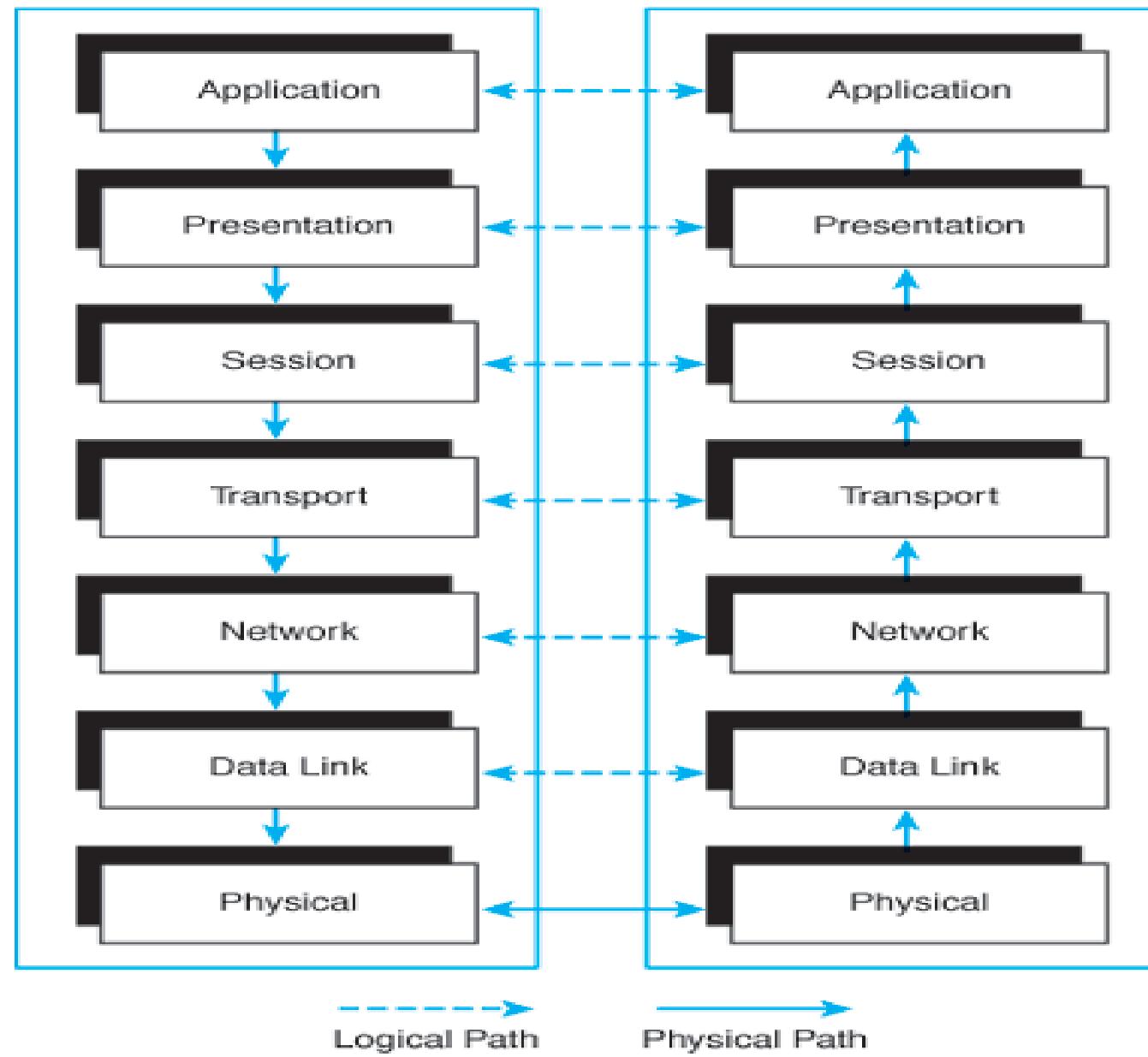


# ISO/OSI Reference Model

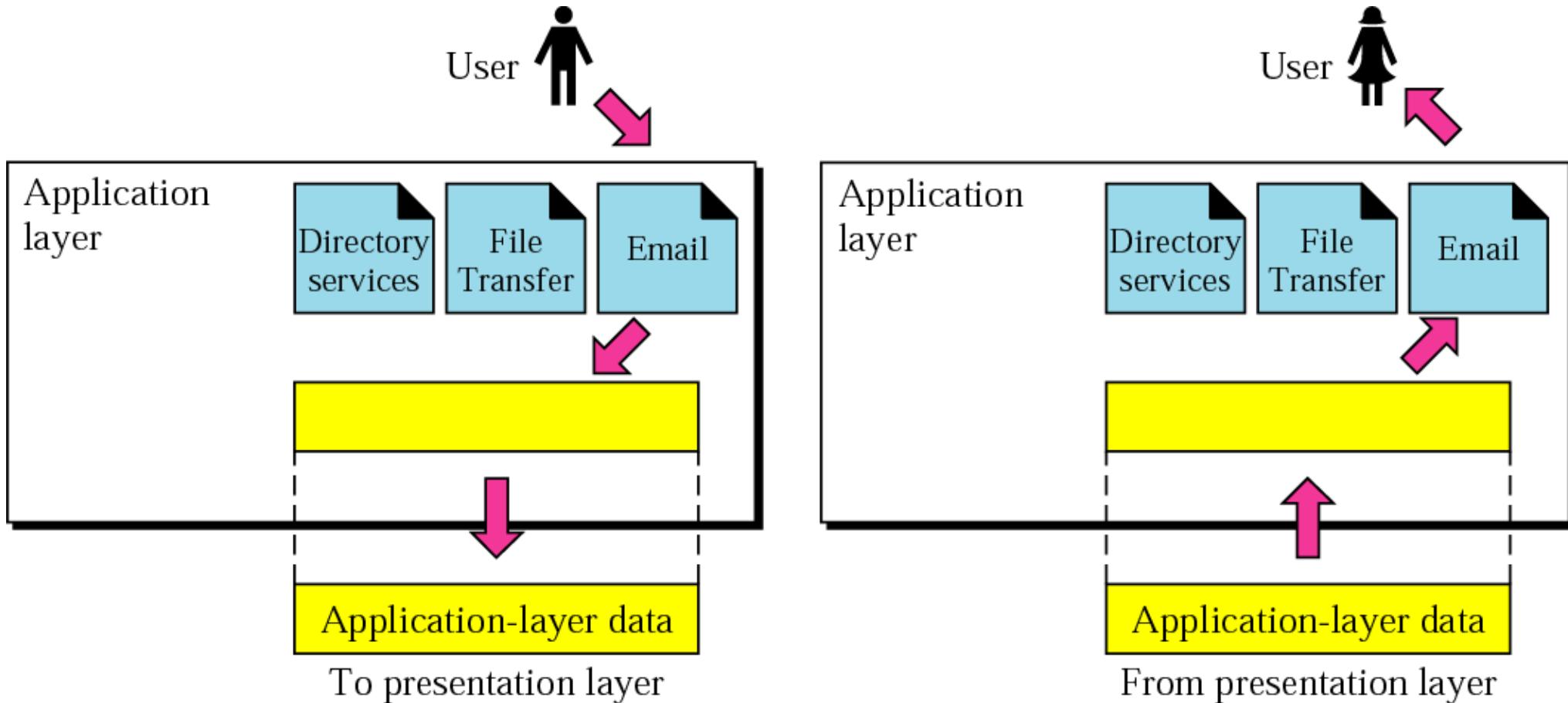
- **Presentation:** allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions;
- **Session:** synchronization, checkpoints, recovery of data exchange;
- Internet stack “missing” these layers!
  - These services, *if needed*, must be implemented in application;



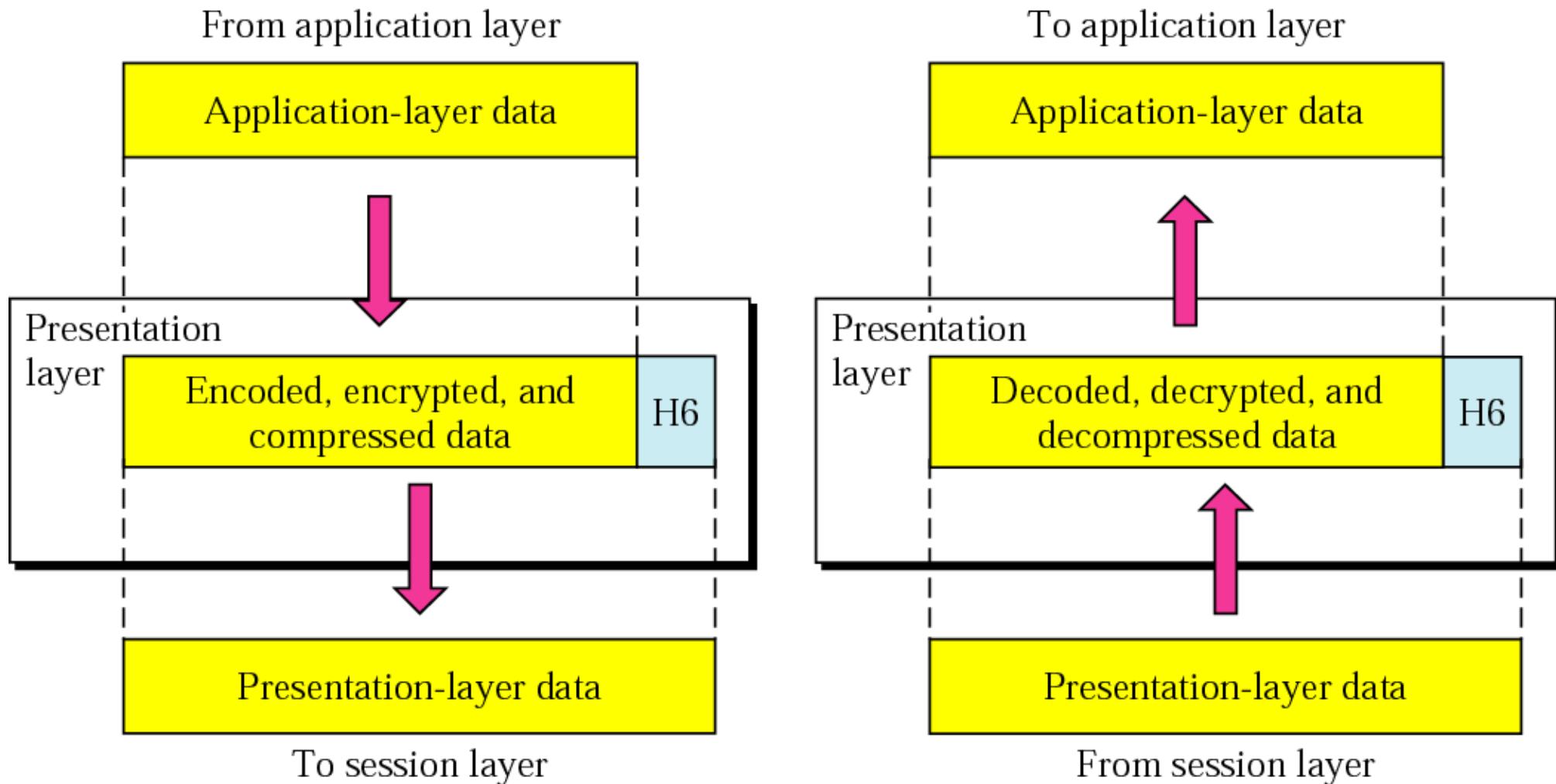
# ISO/OSI Reference Model



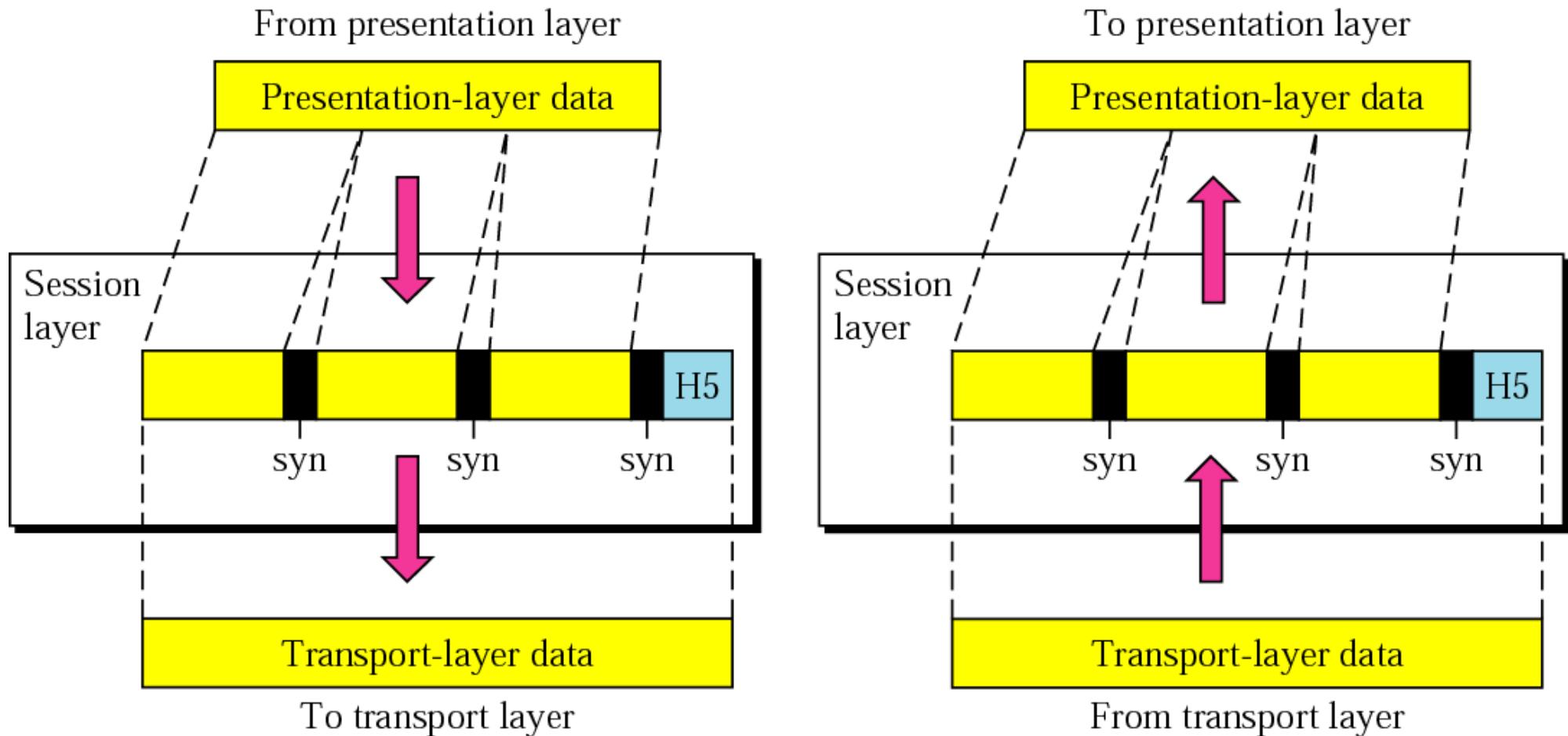
# OSI Model: Application Layer



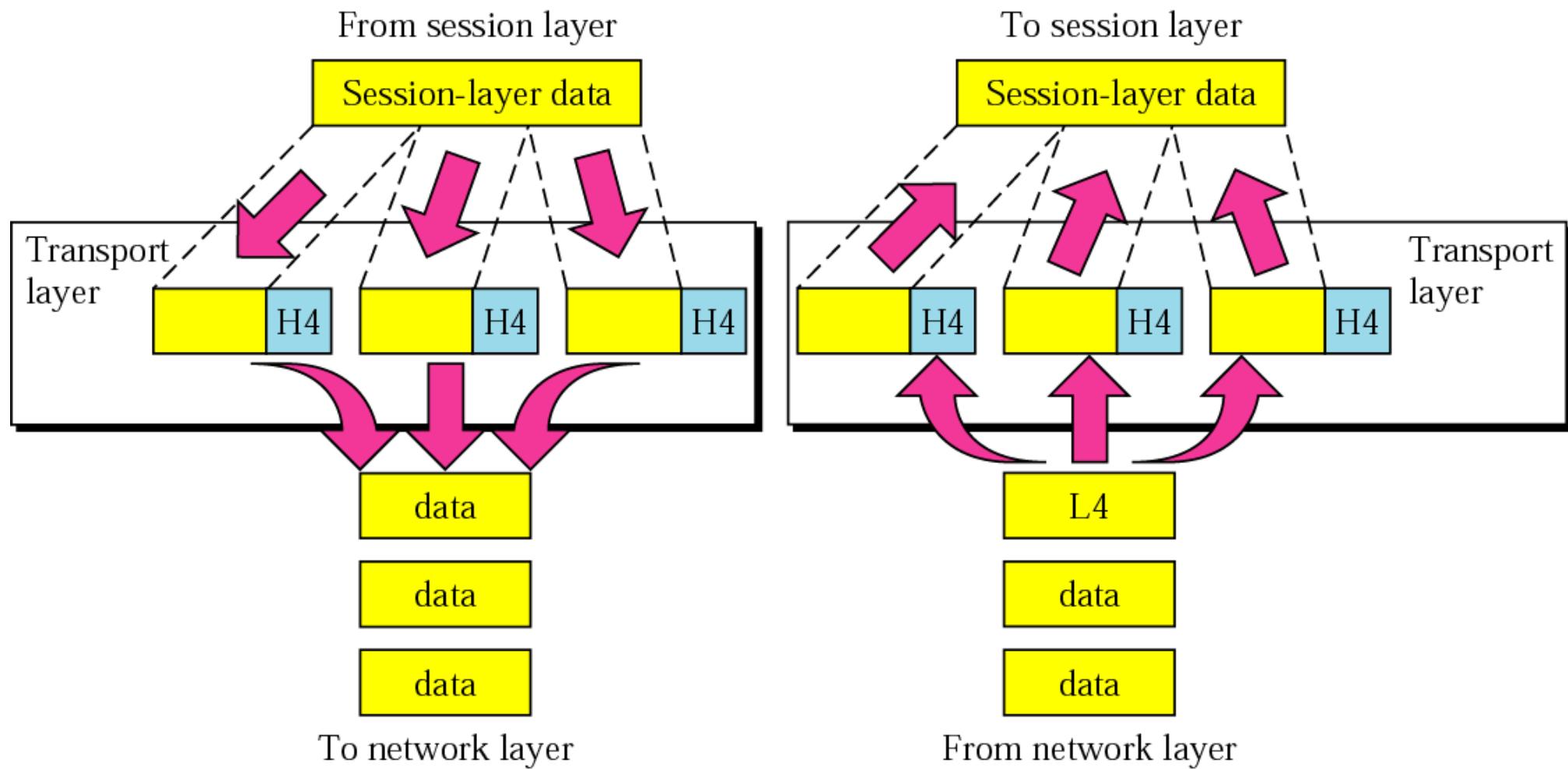
# OSI Model: Presentation Layer



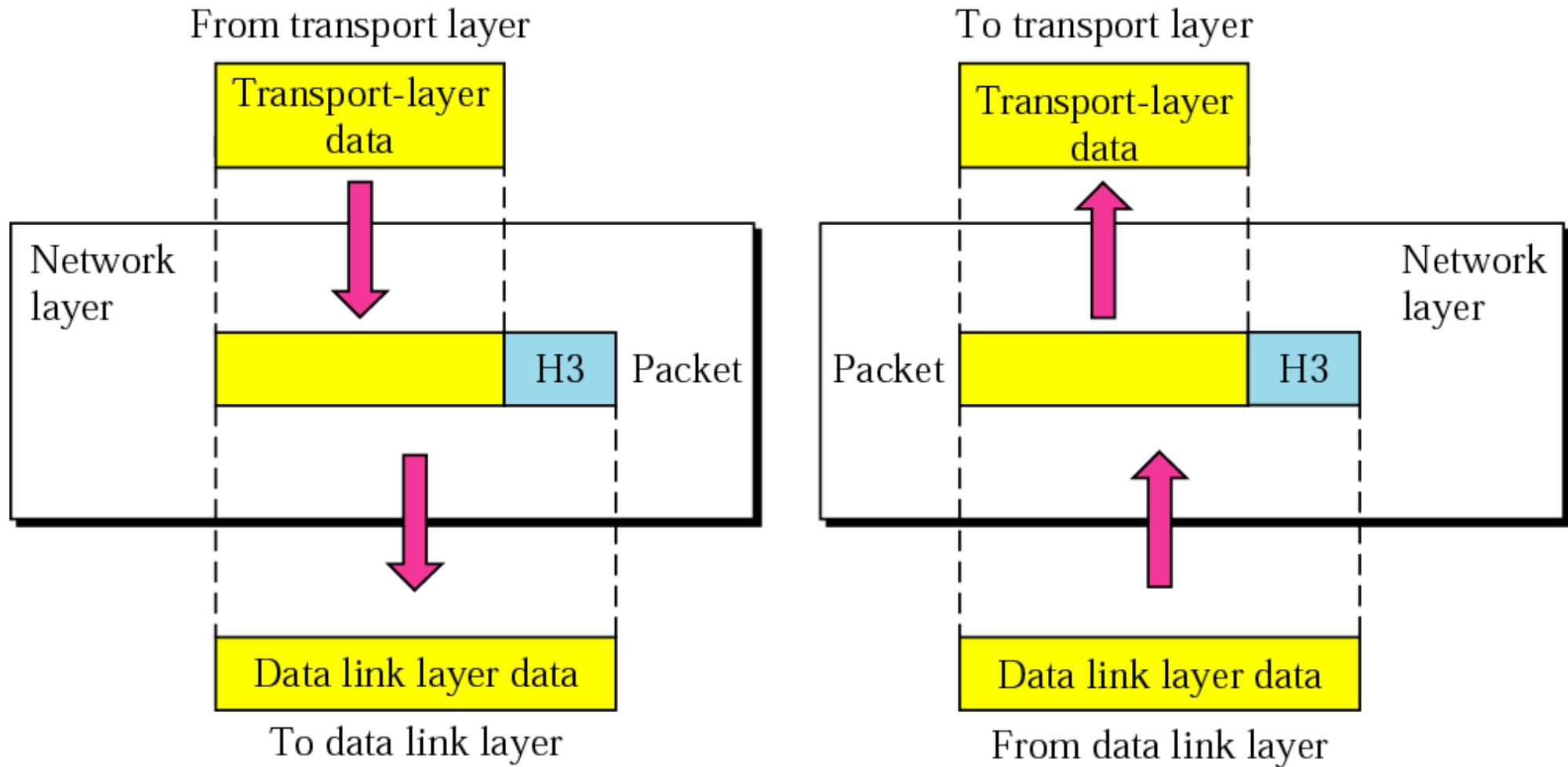
# OSI Model: Session Layer



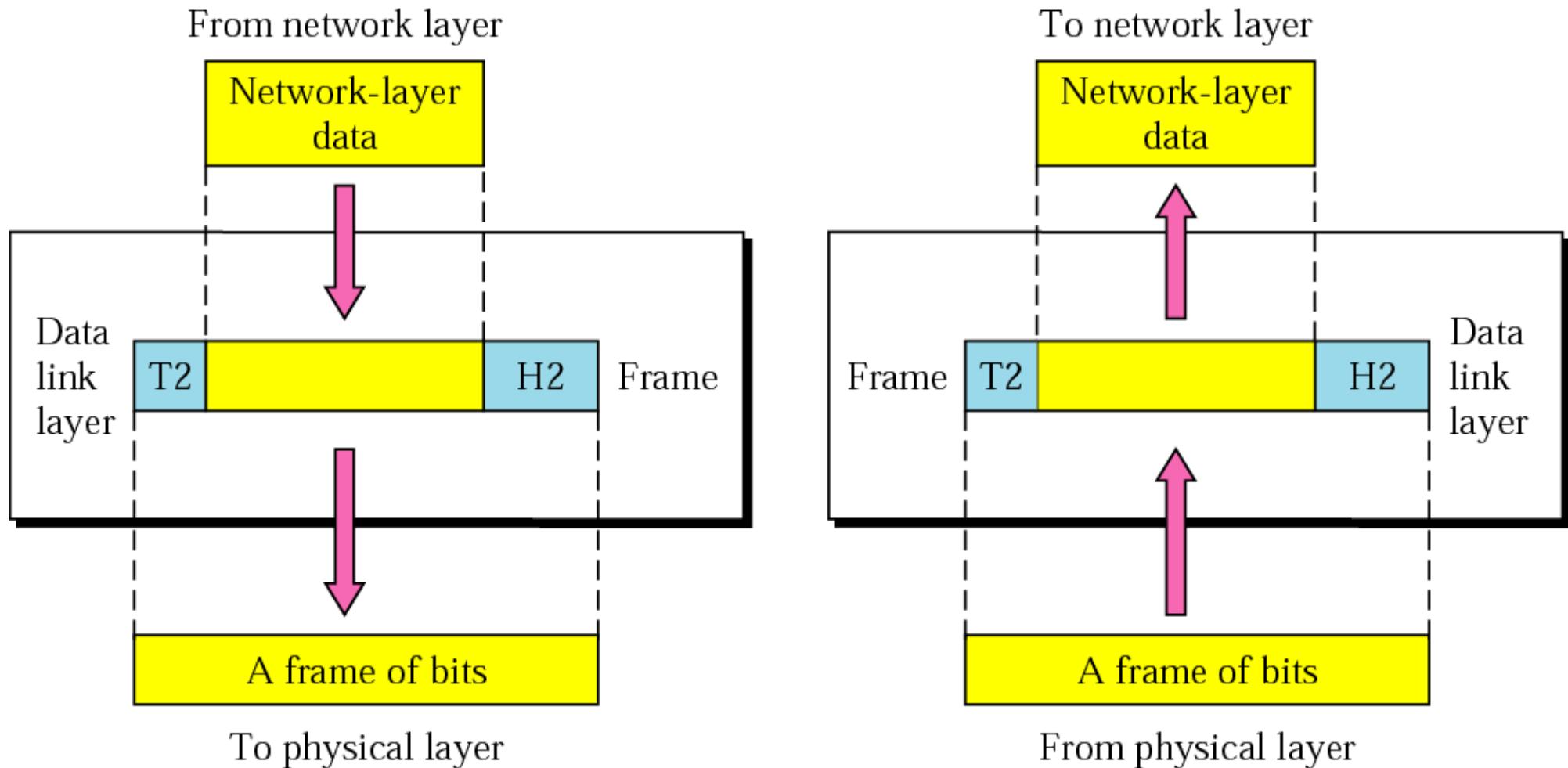
# OSI Model: Transport Layer



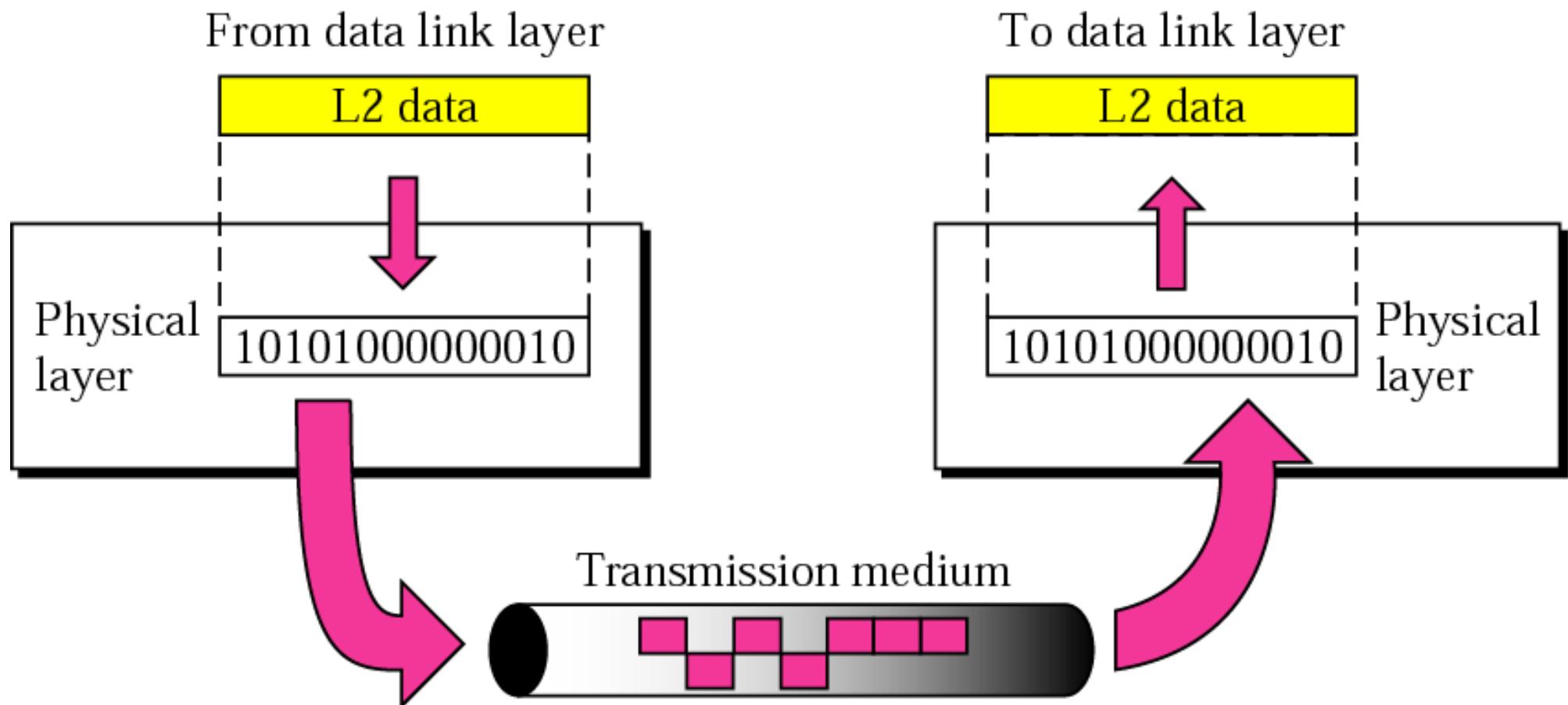
# OSI Model: Network Layer



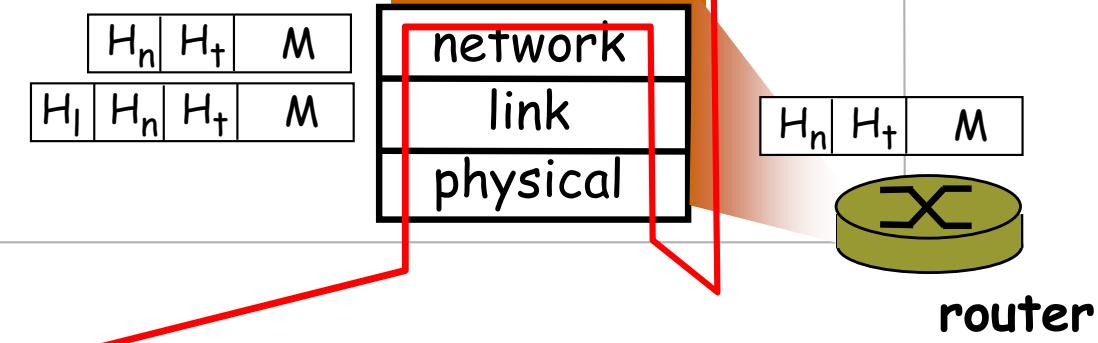
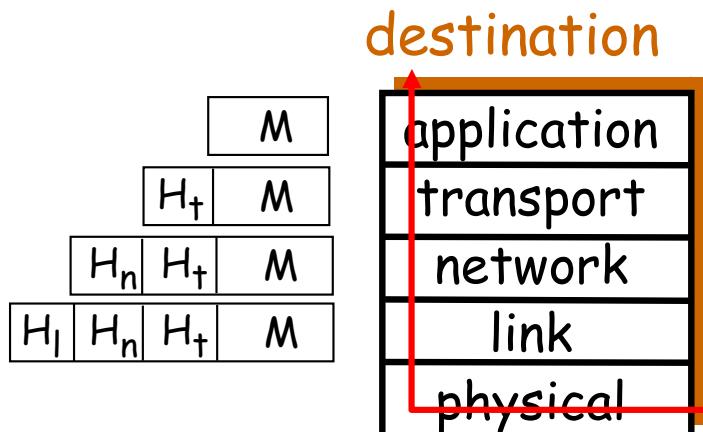
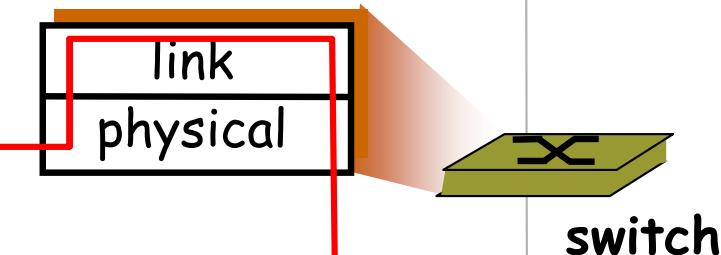
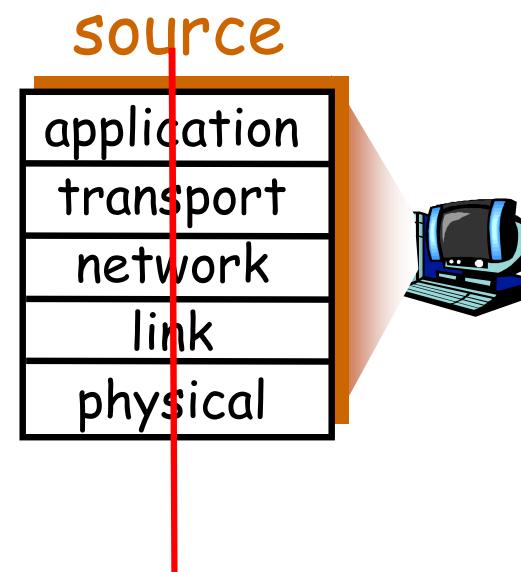
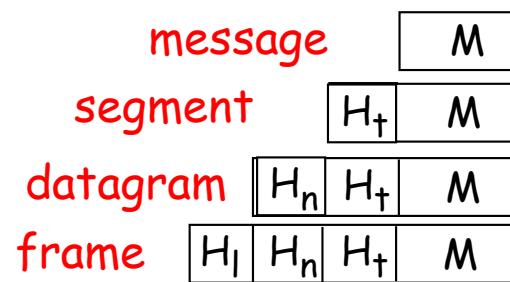
# OSI Model: Data Link Layer



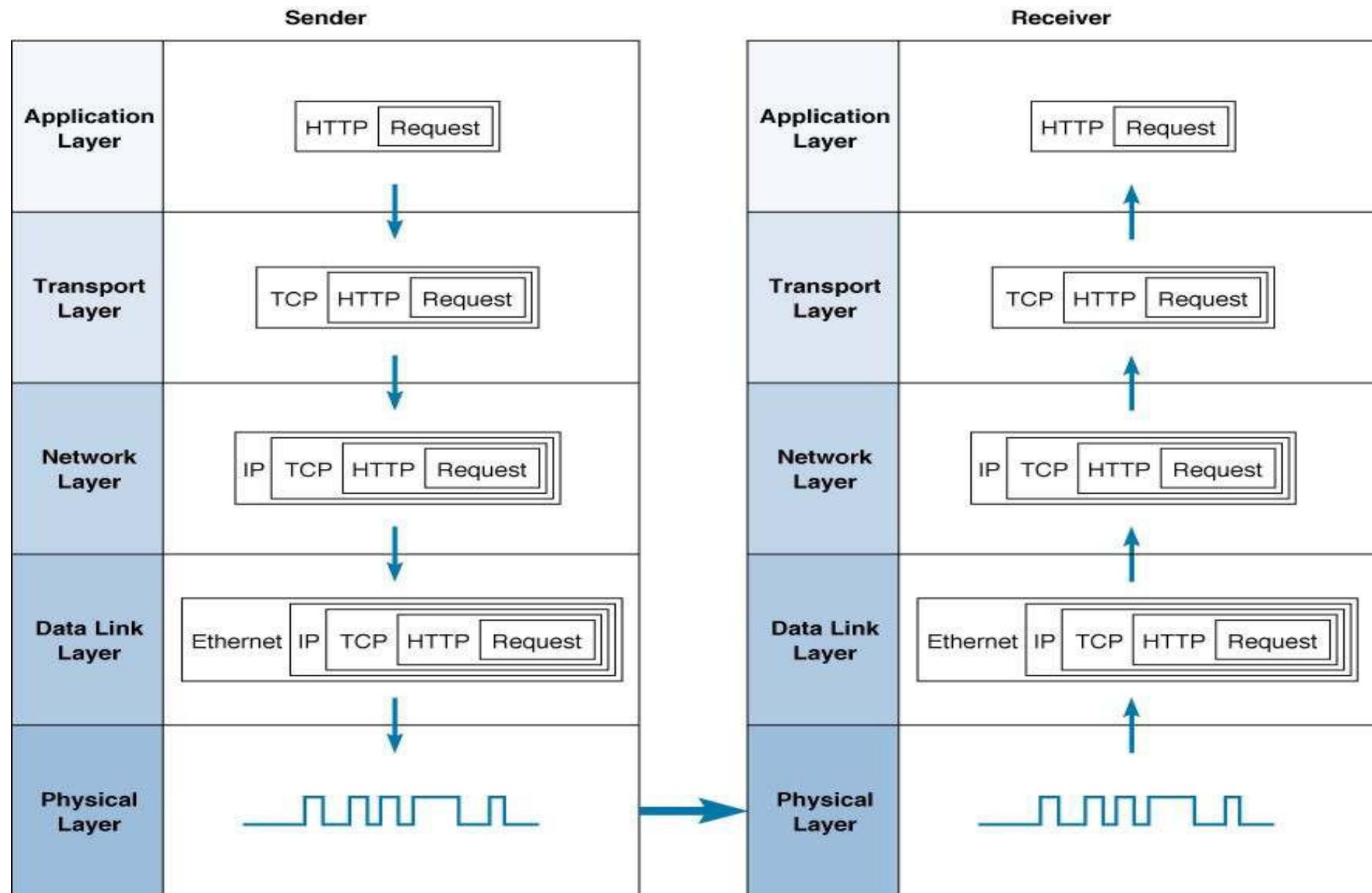
## OSI Model: Physical Layer



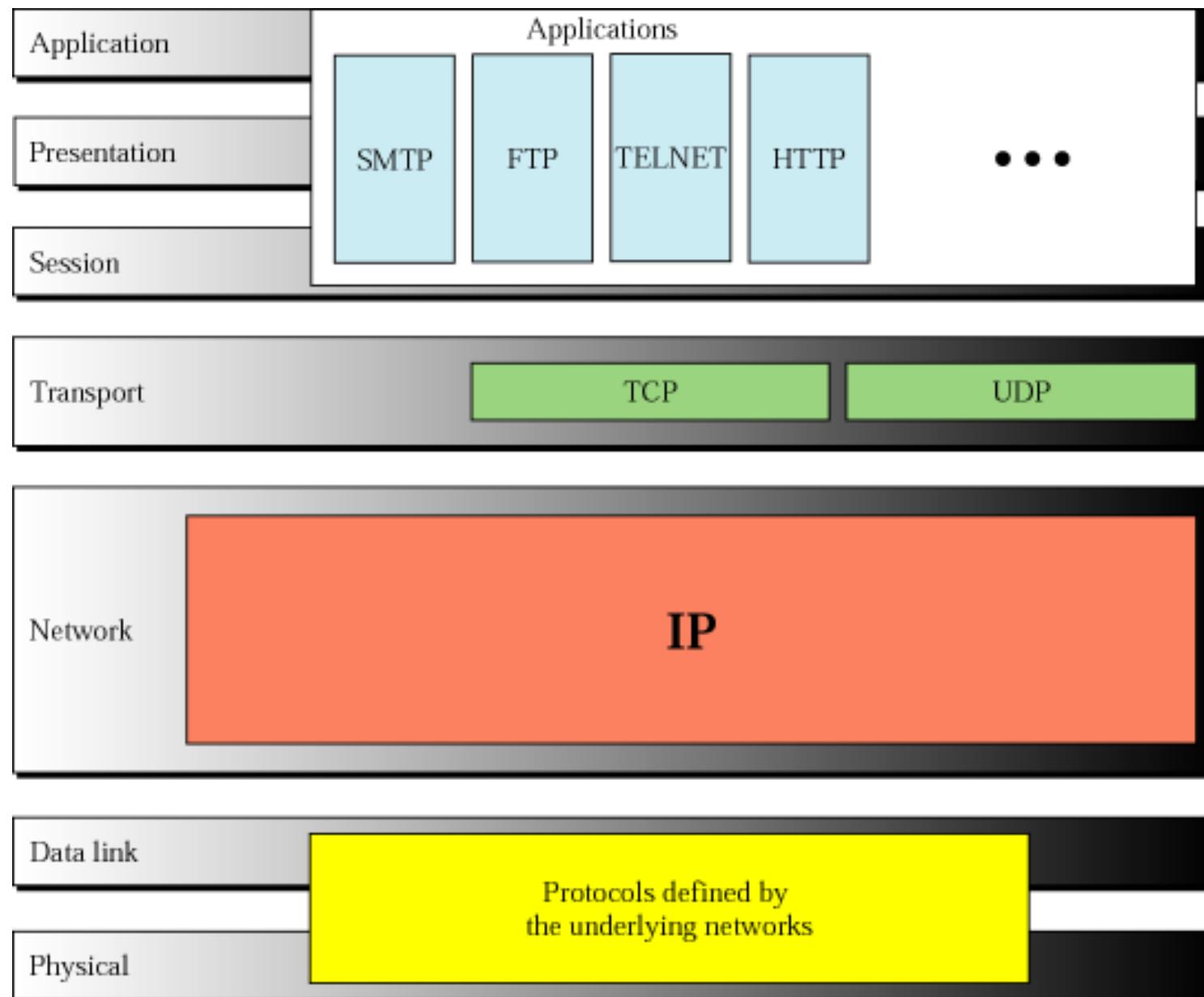
# Encapsulation: TCP/IP



# Encapsulation Example: HTTP



# TCP/IP vs. OSI



## *Introduction: Summary*

Covered a lot of material!

- Internet overview;
- What's a protocol?
- Network edge, core, access network:
  - Packet-switching versus circuit-switching;
  - Internet structure;
- Performance: loss, delay, throughput;
- Layering, service models.

You now have:

- Context, overview, “feel” of networking;
- More detail *to follow!*



# Redes de Computadores

LEIC-Alameda

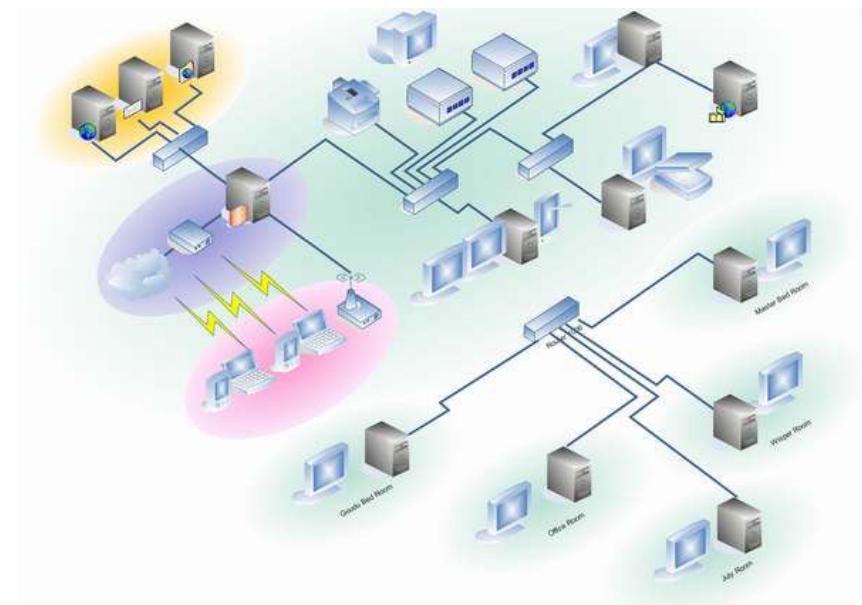
***2 – Application Layer***

Prof. Paulo Lobato Correia

*IST, DEEC – Área Científica de Telecomunicações*

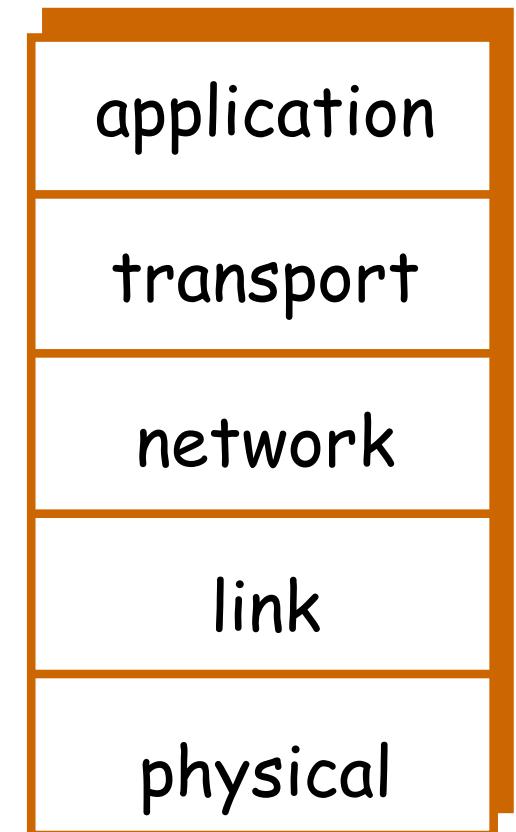
# Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- Electronic Mail
- DNS
- P2P Applications



# *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
  - PPP, Ethernet
- **Physical:** bits “on the wire”
  - RS-232c, V.92



# Applications are End-to-End

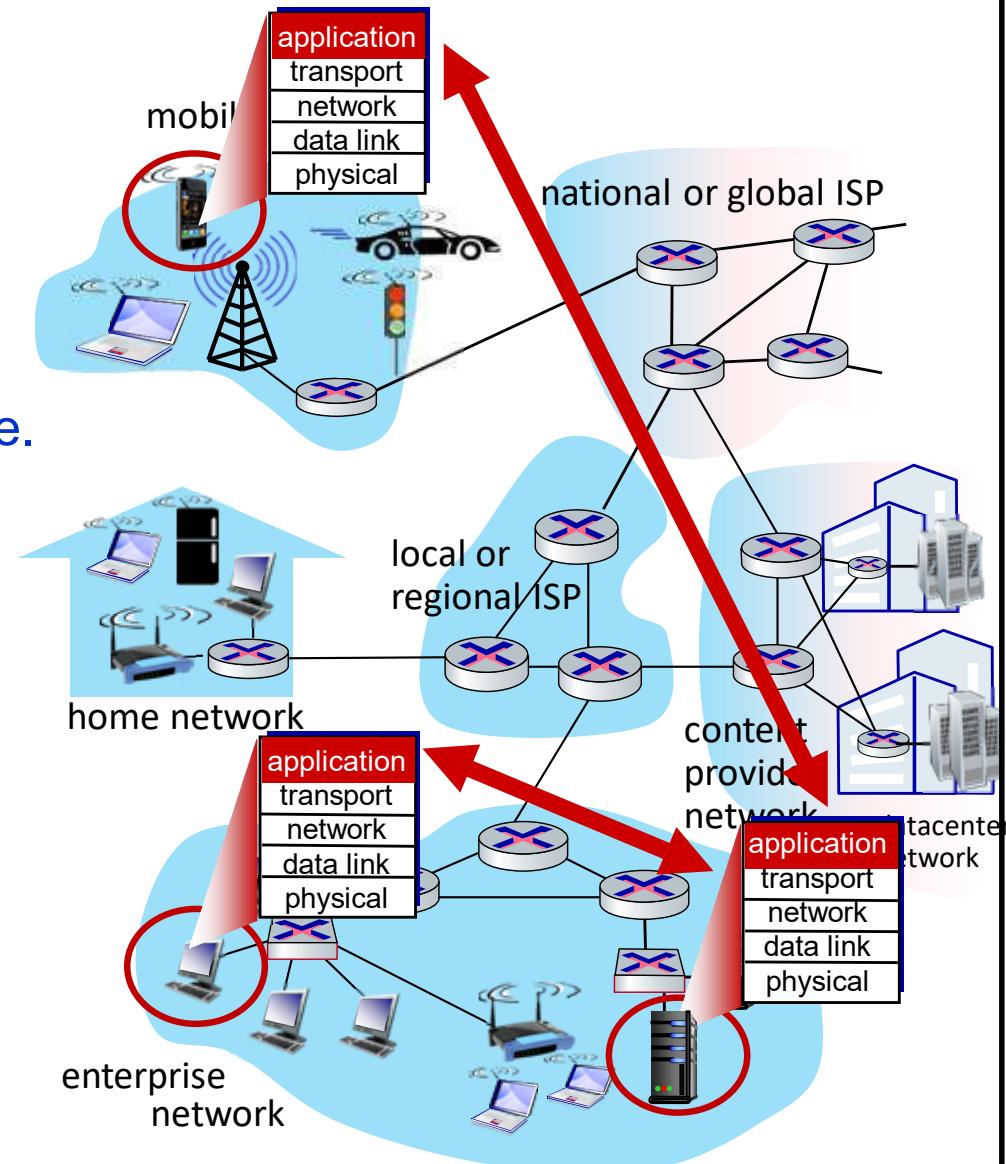
## Networks applications:

- Run on (different) *end systems*;
- Communicate over network;

Example: web browser software  
communicates with web server software.

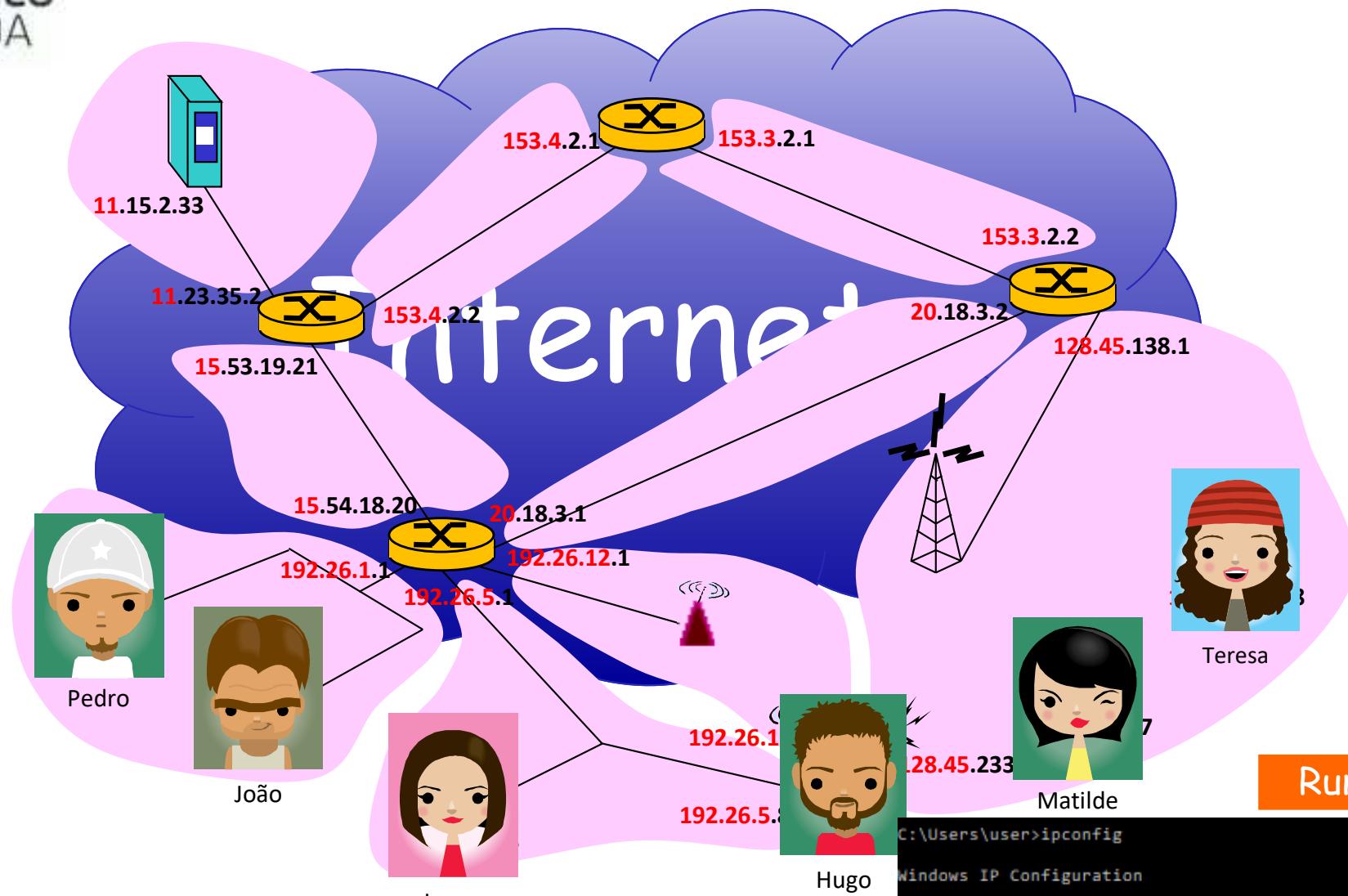
## No need to write software for network-core devices !

- Network-core devices do not run user applications;
- Applications on end systems allow for rapid application development and dissemination.





# Addressing



IPv4 address

193	.	32	.	216	.	9
11000001	00100000	11011000	00001001			

$2^{32} = 4.294.967.296$  addresses!

```
C:\Users\user>ipconfig
Windows IP Configuration

Ethernet adapter Ethernet:

  Connection-specific DNS Suffix  . :
  Link-local IPv6 Address . . . . . : fe80::2c03:d829:6ee2:23f2%8
  IPv4 Address . . . . . : 193.136.222.29
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 193.136.222.254
```

# Addressing Processes

To receive messages, a process must have an *identifier*.

- Host has a unique 32-bit *IPv4* and/or 128-bit *IPv6* address;

Q: Is the host IP address enough to identify the application process?

- No, *many* processes can be running on same host.

- *Identifier* includes both **IP address** and **port numbers** associated with process on host;



- Example of port numbers:

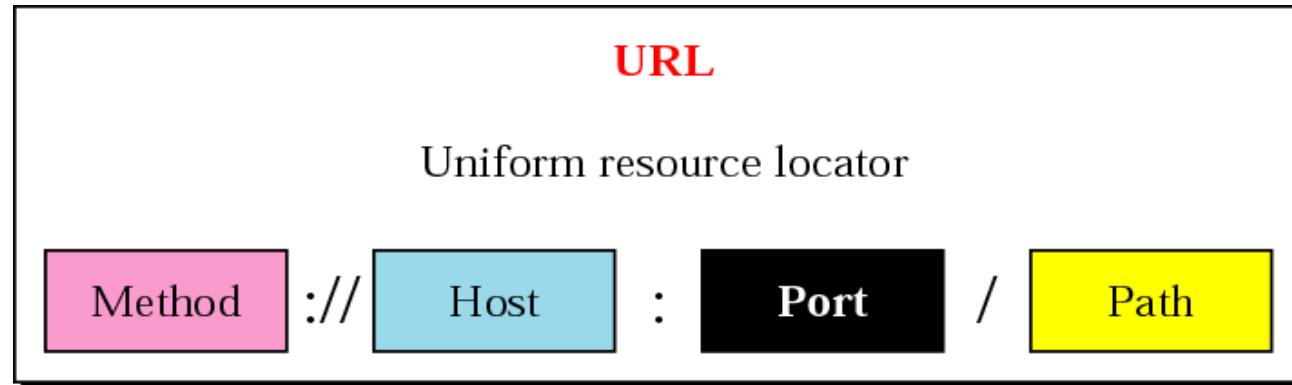
- HTTP server: **80**;
  - Mail server: **25**;



- To send HTTP message to **www.tecnico.ulisboa.pt** web server:

- **IP address:** 193.136.128.169
  - **Port number:** 80

# URL: Universal Resource Locator



Methods:

Name	Used for	Example
http	Hypertext (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
news	Newsgroup	news:comp.os.minix
news	News article	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Sending e-mail	mailto:JohnUser@acm.org
telnet	Remote login	telnet://www.w3.org:80

# *Application Layer Protocol*

## Application layer **protocol defines:**

- Types of messages exchanged
  - e.g., request, response;
- Message **syntax**
  - What fields exist in messages and how they are delimited;
- Message **semantics**
  - Meaning of information in fields;
- **Rules** for when and how application processes send and respond to messages

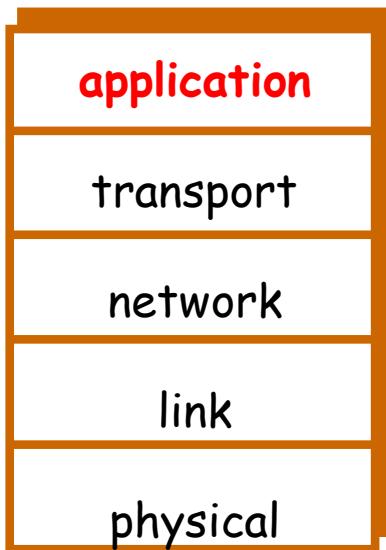
## Public-domain protocols:

- Defined in RFCs, which allows for **interoperability**;
- e.g., HTTP, SMTP

## Proprietary protocols:

- e.g., Skype

# *Application Layer* *uses Transport Layer Services*



What is required from the transport layer:

## Data integrity

- Some apps (e.g. audio) can tolerate some loss;
- Other apps (e.g. file transfer, telnet) require 100% reliable data transfer.

## Throughput

- Some apps (e.g. multimedia) require a minimum amount of throughput to be “effective”;
- Other apps (“elastic apps”) make use of whatever throughput they get.

## Security

- Encryption, data integrity, ...

## Timing

- Some apps (e.g. VoIP, interactive games) require low delay to be “effective”.

# *Transport Service Requirements*

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

# Transport Layer Services

## TCP service:

- *Connection-oriented*: setup required;
- *Reliable transport* between sending and receiving process;
- *Flow control*: sender won't overwhelm receiver;
- *Congestion control*: control transmission speed when network overloaded;
- **Does not provide**: timing, minimum throughput guarantees, security.

## UDP service:

- *Unreliable data transfer* between sending and receiving processes;
- **Does not provide**: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security.

Q: Why bother? Why is there a UDP?

# *Application 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 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary HTTP	TCP or UDP
streaming audio/video	[RFC 7320], 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)

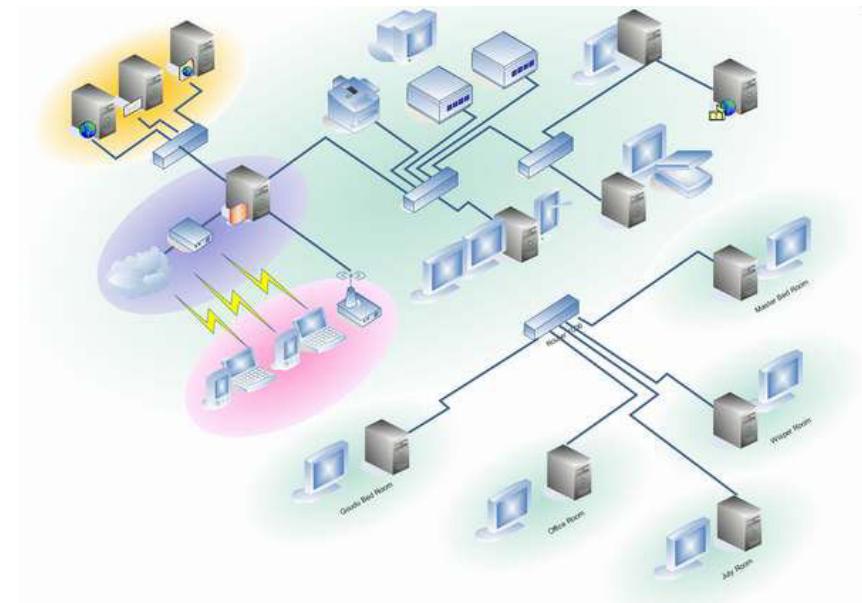
- provides encrypted TCP connections
- data integrity
- end-point authentication

### TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn
- cleartext sent into “socket” traverse Internet *encrypted*
- more: Chapter 8

# Objectives

- Principles of Network Applications
- **Socket Programming with TCP and UDP**
- Web and HTTP
- FTP
- Electronic Mail
- DNS
- P2P Applications



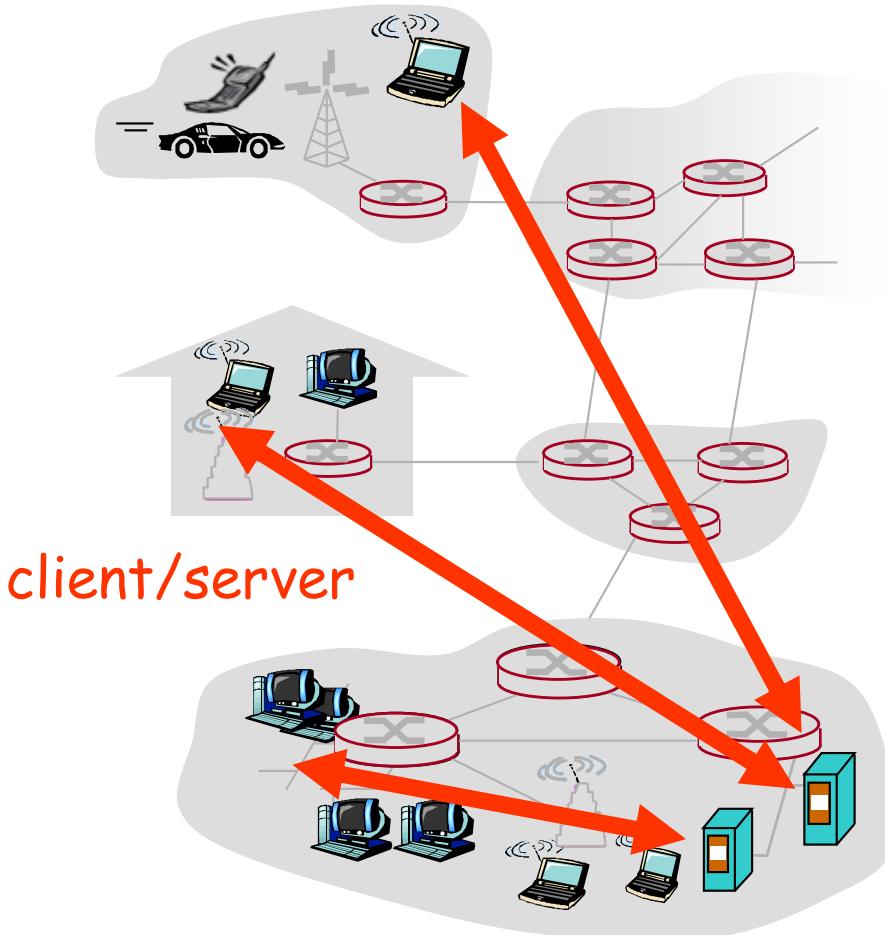
# Client-Server Architecture

## Server:

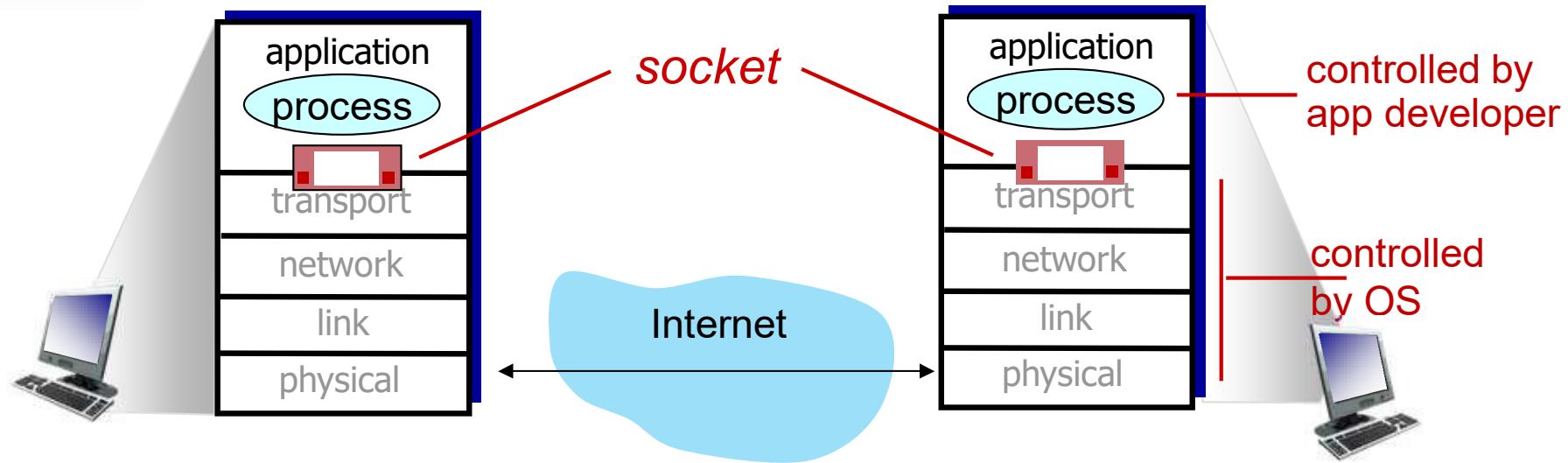
- Always-on host;

## Clients:

- Initiate communication with server, specifying server's **IP address** and **port number**;
- May be intermittently connected;
- Do not communicate directly with each other.



# Sockets API



- Process sends/receives messages to/from its **socket**
- Socket analogy to a door:
  - Sending process sends message out of the door;
  - Sending process **relies on transport infrastructure** on other side of door which brings message to socket at receiving process;

## Sockets API:

- (1) Choice of transport protocol;
- (2) Ability to set a few parameters.

# Socket Programming using TCP

Client must contact server:

- Server process must first be running;
- Server must have created socket (door) to welcome client contacts.

Client contacts server by:

- Creating client-local TCP socket;
- Specifying IP address, port number of server process;
- When **client creates socket**:
  - Client TCP establishes connection to server TCP.
- When contacted by client, **server TCP creates new socket for communication between server and client**:
  - Allows server to talk with multiple clients;
  - Source port numbers are used to distinguish clients.

**TCP provides reliable, in-order transfer of bytes ("pipe") between client and server**

# Socket Programming with UDP

UDP – no “connection” between client and server:

- No handshaking;
- Sender explicitly includes IP address and port of destination to each packet;
- Server must extract IP address and port of client from the received packet.

UDP – transmitted data may be received out of order, or lost!

*UDP provides unreliable transfer  
of groups of bytes (“datagrams”)  
between client and server*

# Socket Programming: TCP vs UDP

## TCP:

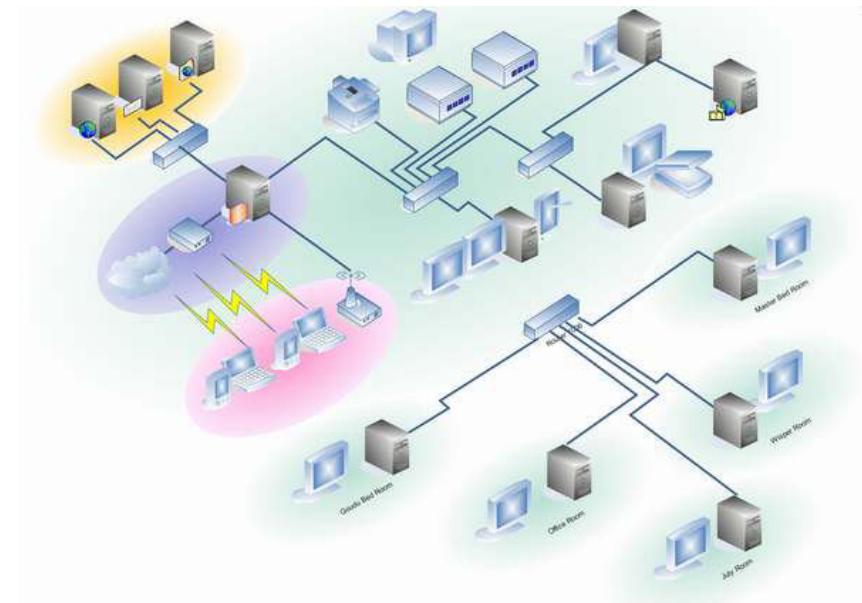
- `read()` and `write()`;
- Byte stream (and no byte is lost);
- Bytes read with `read()` may correspond to several `write()`;
- Bytes written with `write()` may need to be read with several `read()`;

## UDP:

- `sendto()` and `recvfrom()`;
- Preserves boundary between messages;
- Each message read with `recvfrom()` corresponds to a single `sendto()`;
- A message may be lost.

# Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- **Web and HTTP**
- FTP
- Electronic Mail
- DNS
- P2P Applications



# WWW: World Wide Web



- The *World Wide Web (WWW)*:
  - WEB pages and other resources accessible through the Internet.
- The most popular Internet service; client-server architecture.
- Some dates:
  - 1989 – The **concept** appeared in CERN (*Centre Européan pour Recherche Nucleaire*), when **Tim Berners-Lee** concluded that he could not create a research database using a single computer.  
The solution was to have the information spread over a number of computers, but interconnected using **hypertext** and using **URLs** (*Universal Resource Locators*);
  - 1993 – First graphical browser: **Mosaic**;
  - 1994 – **Netscape**;
  - ...



# WWW: World Wide Web (HTML + HTTP)

## □ *HyperText Markup Language (HTML)*

Language used to create (simple) WEB pages.

Other options:

□ Dynamic HTML – allows mouse-over techniques, layers, ...

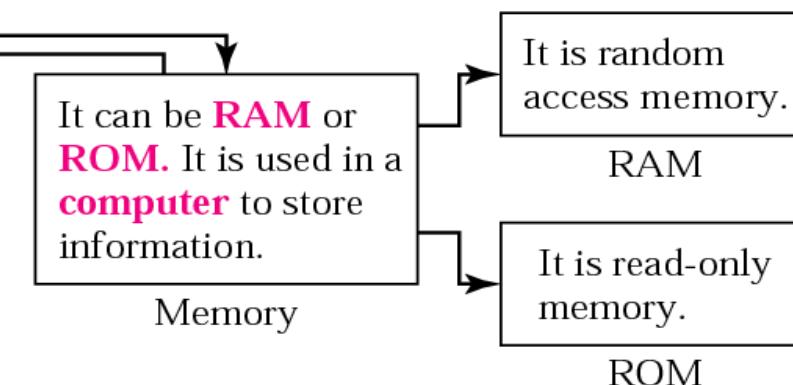
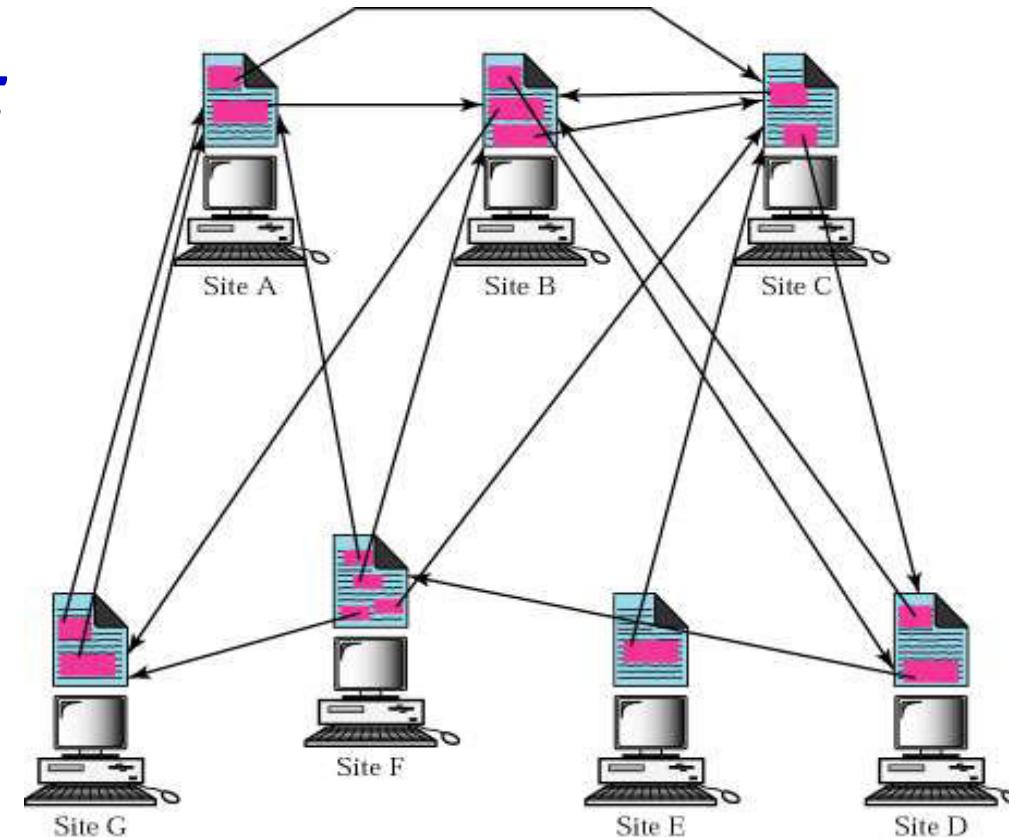
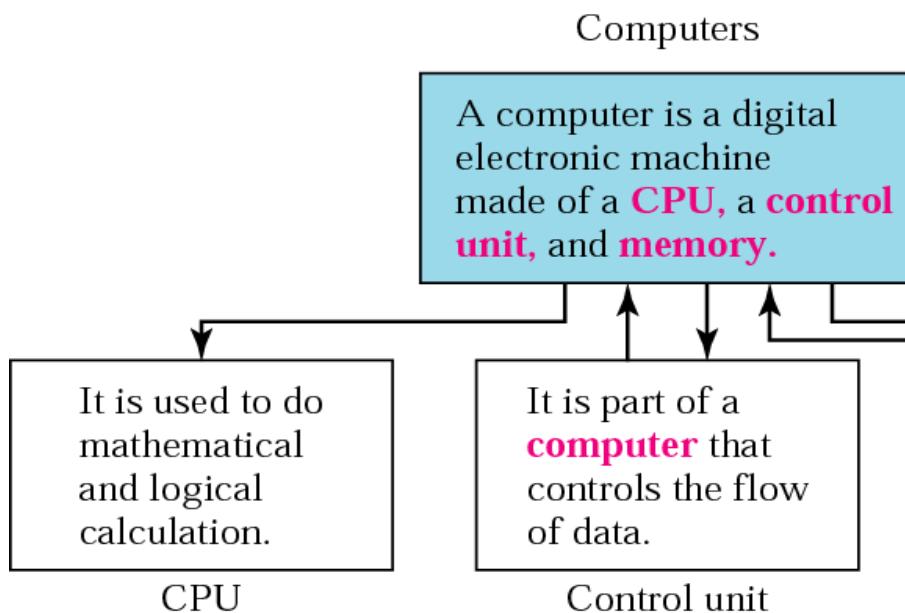
□ **XML (Extensible Markup Language)** – describes how to create a document, including its definition and contents; The syntax is similar to HTML, but allows defining new tags with their own properties.

## □ *HyperText Transport Protocol (HTTP)*

Protocol used to transfer WEB pages.



# HTML: Hypertext





# HTML



## Some HTML tags:

Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h n> ... </h n>	Delimits a level <i>n</i> heading
<b> ... </b>	Set ... in boldface
<i> ... </i>	Set ... in italics
<center> ... </center>	Center ... on the page horizontally
<ul> ... </ul>	Brackets an unordered (bulleted) list
<ol> ... </ol>	Brackets a numbered list
<li>	Starts a list item (there is no </li>)
 	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a Horizontal rule
	Displays an image here
<a href="..."> ... </a>	Defines a hyperlink

## Example:

```
<html>  
  
<head>  
<title>RC</title>  
</head>  
  
<body>  
<H2> RC </H2>  
  
<P> Informações: </P>  
  
...  
  
</body>  
</html>
```

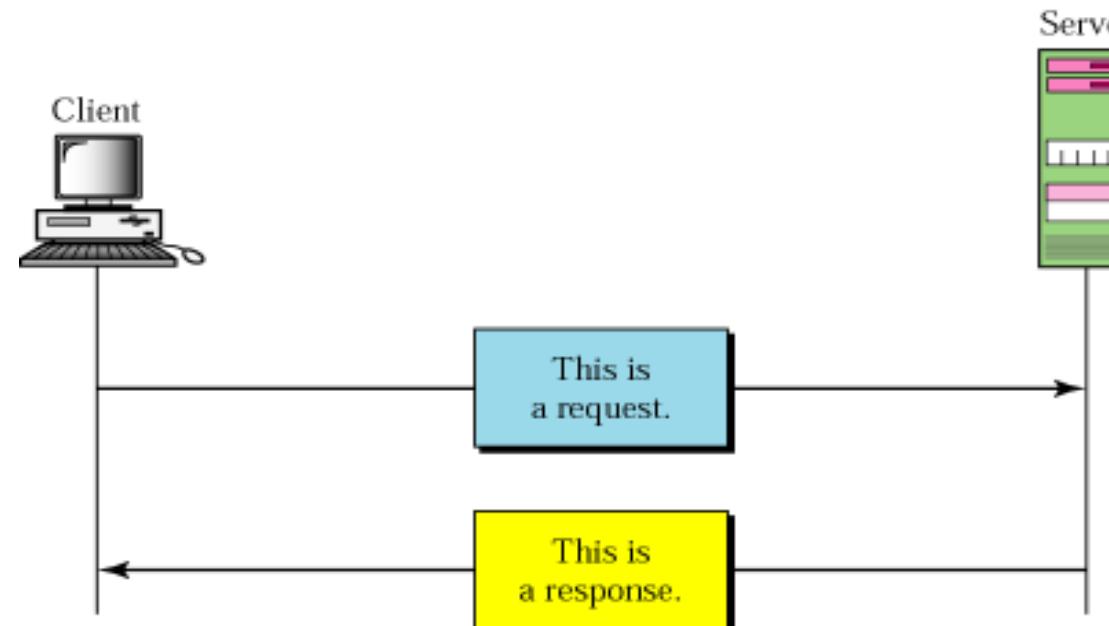
## The WWW Protocol: HTTP

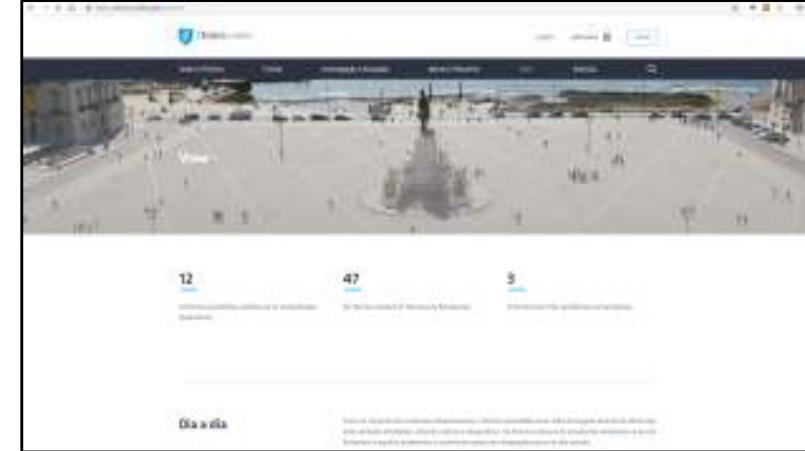


**HTTP** is the application layer protocol used in the WWW.

It uses **TCP** at the transport layer (default: port 80).

- Client: *browser* makes requests, receives and presents replies;
- Server: answers requests;  
**stateless** - keeps no information about previous requests;





## Some definitions:

- Web page consists of objects:

- HTML file;
  - JPEG images;
  - Audio files, ...

(objects can be stored on different web servers)

- The base object (HTML file) may reference other objects;
- Each object is addressable by a URL (Uniform Resource Locator)

Example: `http://tecnico.ulisboa.pt/pt/viver/`

host name

path name

# HTTP Overview

HTTP: hypertext transfer protocol:

- Web's application layer protocol;
- Client/Server model:
  - **Client:** browser that requests, receives and “displays” Web objects;
  - **Server:** Web server sends objects in response to requests.



# HTTP Overview

## HTTP is “stateless”:

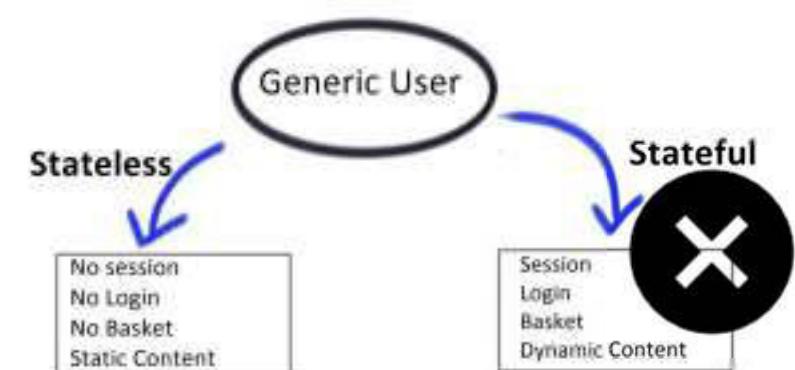
- Server maintains no information about past client requests.

## Uses TCP:

- Client initiates TCP connection (*creates socket*) to server, on 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.

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

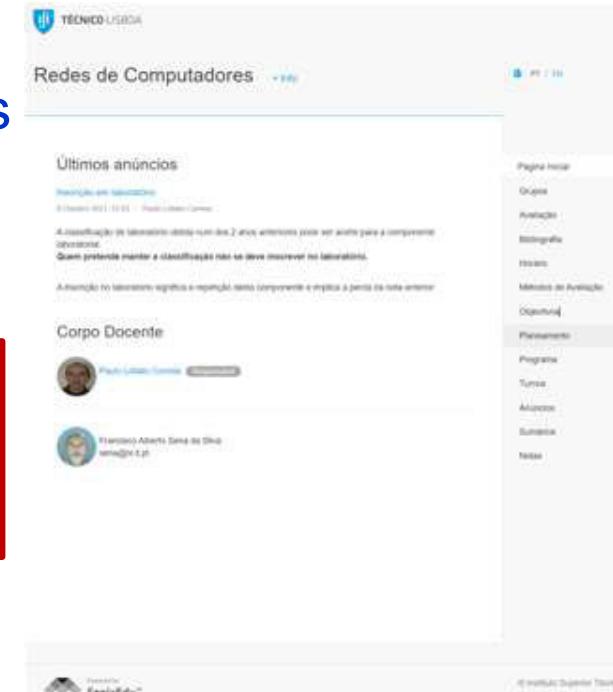
- At most one object is sent over a TCP connection;
- Browsers can open parallel connections (typically 5-10);
- HTTP/1.0.

## Persistent HTTP

- Multiple objects can be sent over a single TCP connection between client and server;
- When using *pipelining* a browser can send requests as soon as it identifies them;
- HTTP/1.1.

An **HTTP object** is identified by a URL, examples:  
- the HTML base webpage (e.g.: index.html)  
- each referenced image, sound, ...

```
<div class="row" style="padding-top:50px; padding-bottom:25px;">
  <div class="col-sm-9">
    <div>
      <a href="http://tecnico.ulisboa.pt" target="_blank"></a>
```



# Non-persistent HTTP: Response time

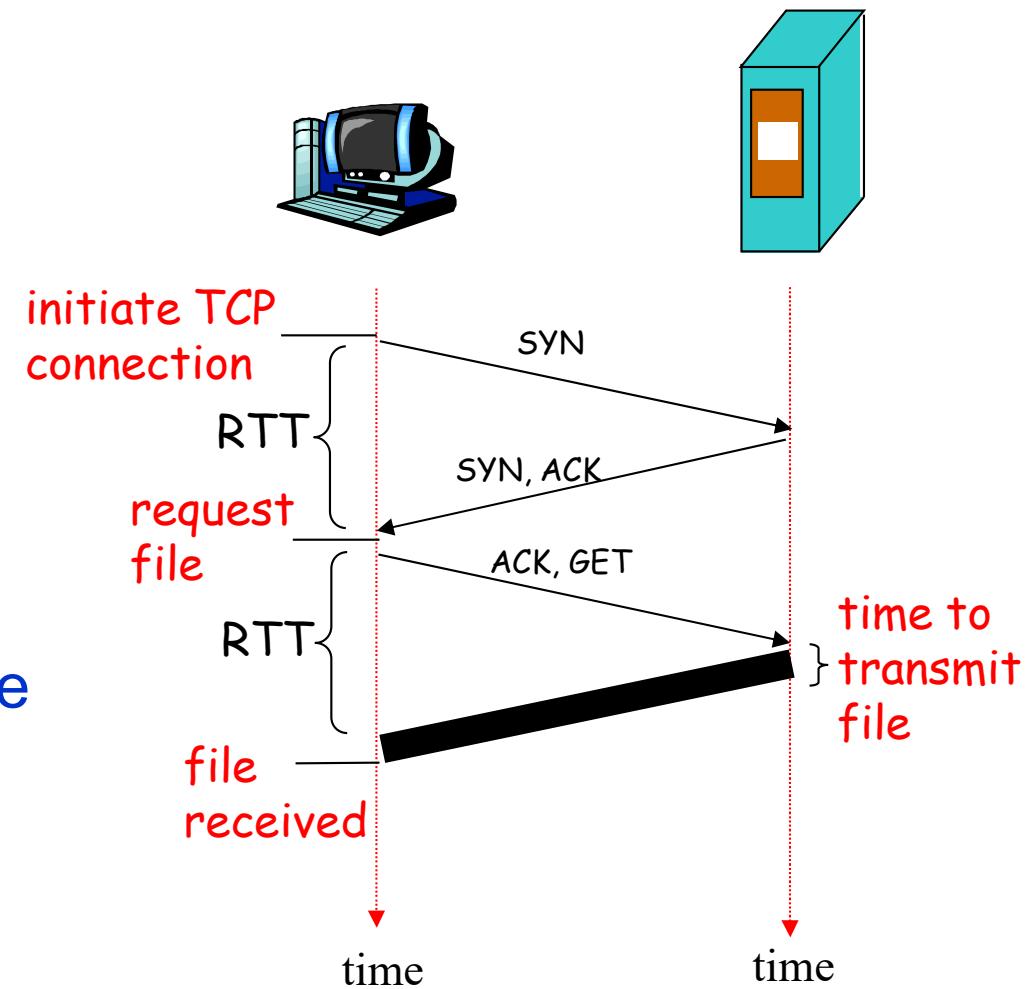
**RTT (round trip time):**

time for a small packet to travel from client to server and back.

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.

Total = 2.RTT+transmit time





## Non-persistent HTTP

# *HTTP Connections*

HTML + 1 image

## Persistent HTTP

## Non-persistent HTTP issues:

- Requires **2 RTTs per object**;
- OS overhead for each TCP connection (e.g., allocate buffers and variables in client and server);
- 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 are sent over the open connection (requiring **1 RTT per each object after the first**);
- With pipelining: 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: **(1) request, (2) response**

**(1) HTTP request message:**

- ASCII (human-readable format)

request line  
(GET, POST,  
HEAD commands)

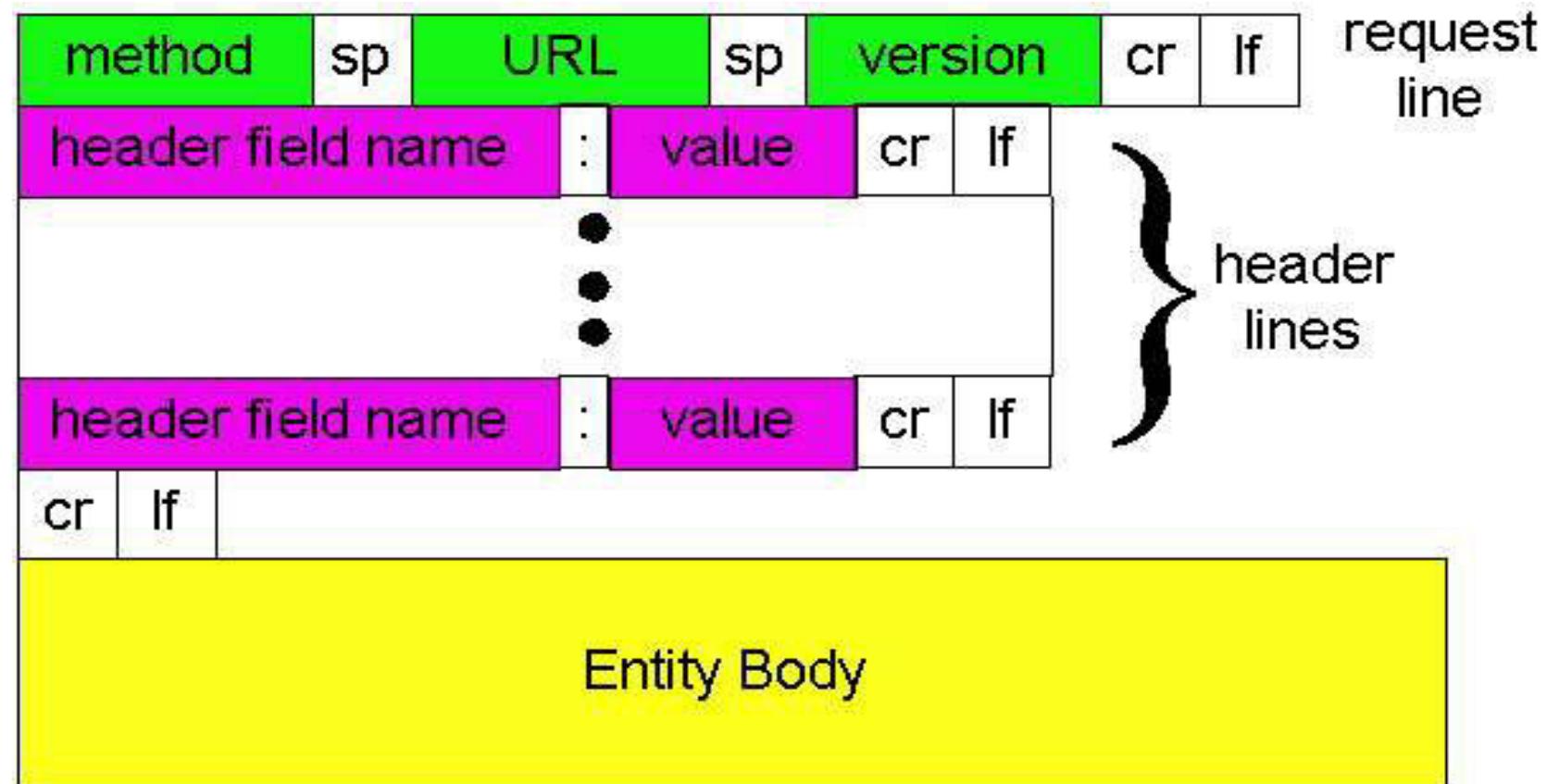
header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(message body)

Check out the online interactive exercises for  
examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# HTTP Request Message



# HTTP: Method Types

## HTTP/1.0 (RFC 1945)

- GET
- POST
- HEAD
  - Asks server to leave requested object out of response.

## HTTP/1.1 (RFC 2616)

- GET, POST, HEAD
- PUT
  - Uploads file in entity body to path specified in URL field;
- DELETE
  - Deletes file specified in the URL field.

# HTTP Response Message

status line  
(protocol  
status code  
status phrase)

header lines

data, e.g.,  
requested  
HTML file

HTTP/1.1 200 OK  
Connection close  
Date: Thu, 08 Aug 2019 12:00:15 GMT  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Mon, 24 Jun 2019 ....  
Content-Length: 6821  
Content-Type: text/html

data data data data data ...

# HTTP Response: Status Line Codes

The HTTP response is in the first line of server → client response message.  
A few examples of status line codes:

## 200 OK

- Request succeeded, requested object later in this message;

## 301 Moved Permanently

- Requested object moved, new location specified later in this message (Location:);

## 400 Bad Request

- Request message not understood by server;

## 404 Not Found

- Requested document not found on this server;

## 505 HTTP Version Not Supported



# Trying out HTTP (client side)

1. Netcat to your favorite Web server:

```
nc -v gaia.cs.umass.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at gaia.cs.umass.edu

2. Type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1  
Host: gaia.cs.umass.edu
```

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

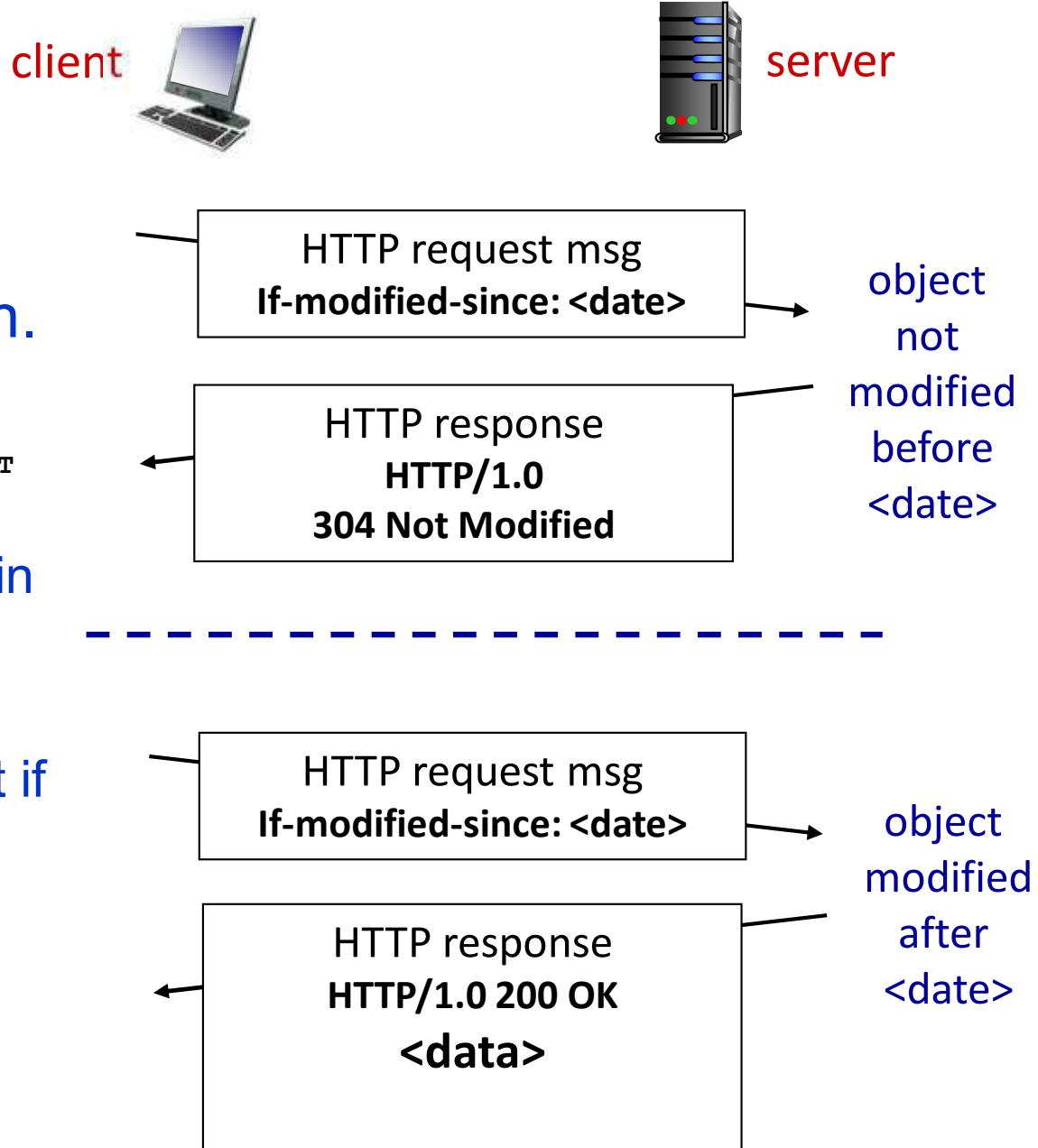
3. Look at response message sent by HTTP server!  
(or use **Wireshark** to look at captured HTTP request/response)

# HTTP: Browser Cache + Conditional GET

**Goal:** don't send object if cache already has up-to-date version.

```
HTTP/1.1 200 OK
Date: Mon, 08 Oct 2018 12:00:15 GMT
Last-Modified: Thu, 04 Oct 2018 12:45:26 GMT
...
...
```

- Client: specify date of cached copy in HTTP request header:  
**If-modified-since: <date>**
- Server: response contains no object if cached copy is up-to-date:  
**HTTP/1.0 304 Not Modified**



## *HTTP: Cookies* *Maintaining State*

Many major websites use cookies to maintain some state between transactions.

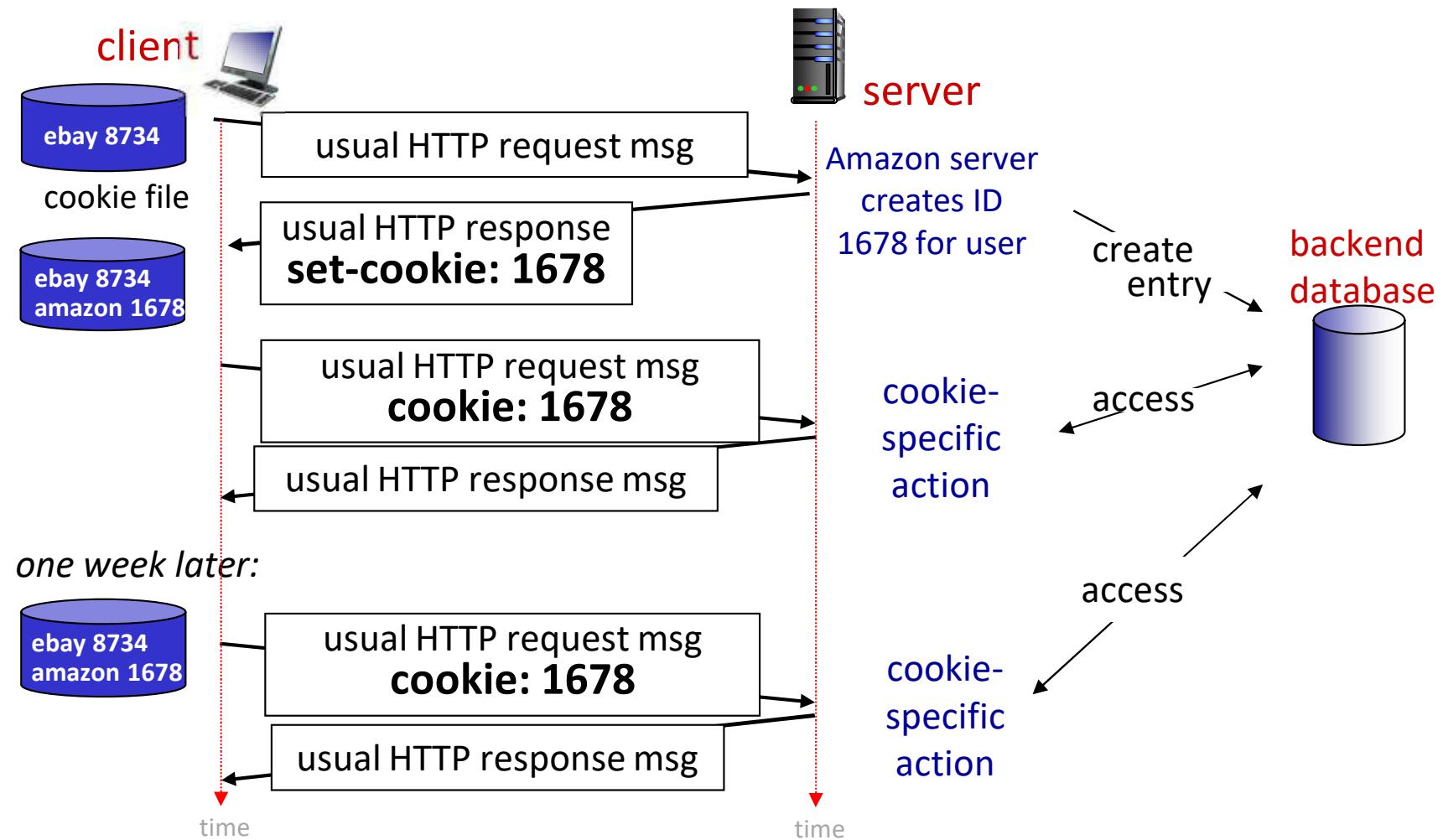
### Four components:

- 1) Cookie header line of HTTP *request* message;
- 2) Cookie header line in HTTP *response* message;
- 3) Cookie file kept on user's host, managed by user's browser;
- 4) Back-end database at Web server.

### Example:

- Susan always accesses Internet from her PC;
- Visits specific e-commerce site for first time;
- When initial HTTP requests arrives at site, site creates:
  - Unique ID;
  - Entry in backend database for ID.
- Subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

# Maintaining user/server state: cookies



# HTTP Cookies: Comments

## What cookies can bring:

- Authorization;
- Shopping carts;
- Recommendations;
- User session state (Web e-mail).

## How to keep “state”:

- Protocol endpoints:
  - **maintain state at sender/receiver** over multiple transactions;
- **Cookies:** HTTP messages carry state.

aside

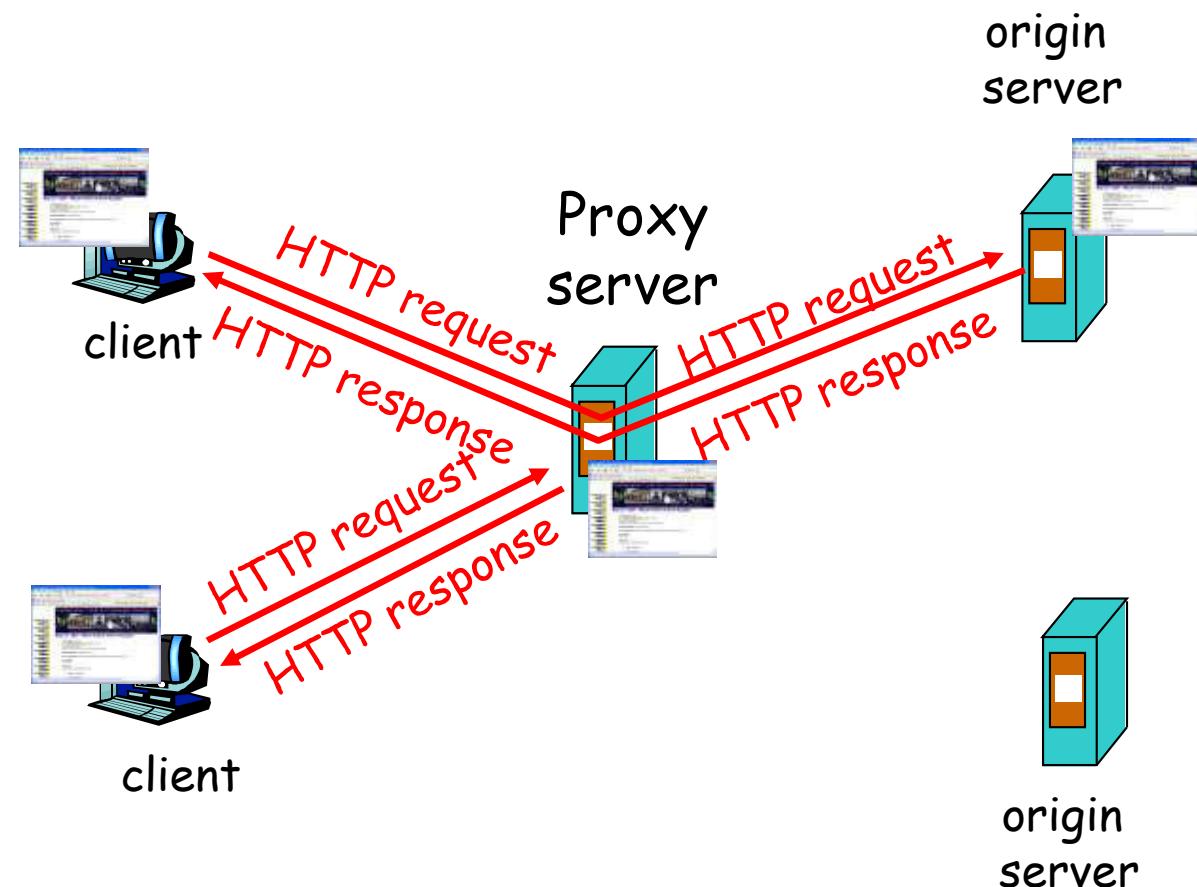
### Cookies and privacy:

- Cookies permit sites to learn a lot about you;
- You may supply name and e-mail to sites.

# Web Caching: Proxy Server

Goal: satisfy client request without involving the origin server.

- User sets browser to access the Web via a cache;
- Browser sends all HTTP requests to the cache:
  - If object is in cache: cache returns object;
  - Otherwise, cache requests object from origin server, then returns object to client.

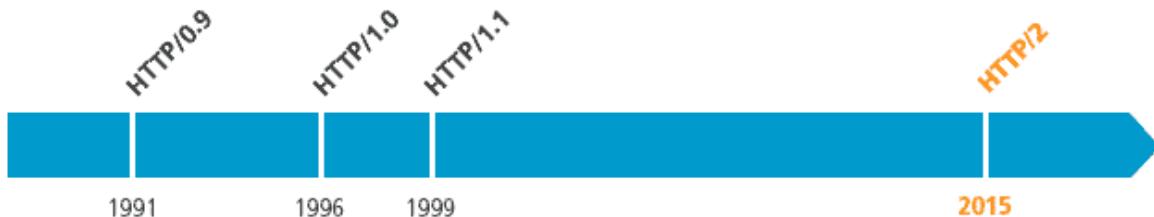


# *Web Caching*

- Cache acts as both client and server;
- Typically cache is installed by ISP (university, company, residential ISP).

## Why Web caching?

- Reduce **response time** for client request;
- Reduce **traffic** on an institution's access link;
- Internet dense with caches: enables “poor” content providers to effectively deliver content.



HTTP 1.0 – RFC 1945, May 1996, T. Berners-Lee, et. al

HTTP 1.1 – RFC 2068, Jan 1997  
RFC 2616, Jun 1999

...

RFC 7230, Jun 2014

HTTP/2 – RFC 7540, May 2015

"This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP version 2 (HTTP/2). HTTP/2 enables a **more efficient use of network resources** and a reduced perception of latency by introducing **header field compression** and **allowing multiple concurrent exchanges on the same connection**. It also introduces unsolicited push of representations from servers to clients."

This specification is an alternative to, but **does not obsolete, the HTTP/1.1 message syntax**. HTTP's existing semantics remain unchanged."

"... it allows interleaving of request and response messages on the same connection and uses an efficient coding for HTTP header fields. It also allows **prioritization** of requests, letting more important requests complete more quickly, further improving performance."

"... **fewer TCP connections can be used.** ..."

"HTTP/2 also enables more efficient processing of messages through use of **binary message framing**."

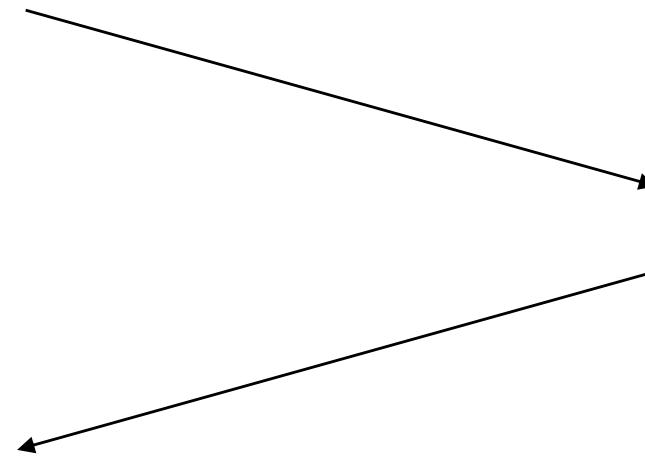
## HTTP/2

- **reduce latency** by enabling full request and response multiplexing;
  - **minimize protocol overhead** via efficient compression of HTTP header fields;
  - add support for **request prioritization**;
  - add **server push**.
- 
- **does not modify semantics of HTTP.** All the core concepts, such as HTTP methods, status codes, URIs, and header fields, remain in place.
- 
- **modifies how the data is formatted (framed) and transported** between client and server.
    - introduces a new binary framing layer that is not backward compatible with previous HTTP/1.x

# HTTP/2

## HTTP/2 Request

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```



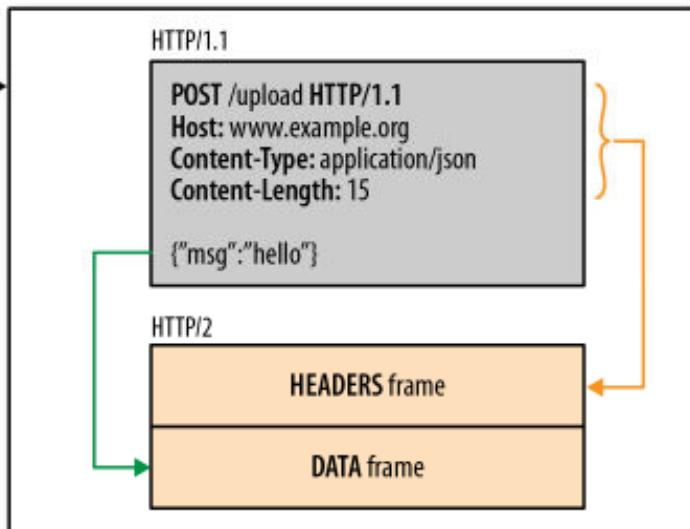
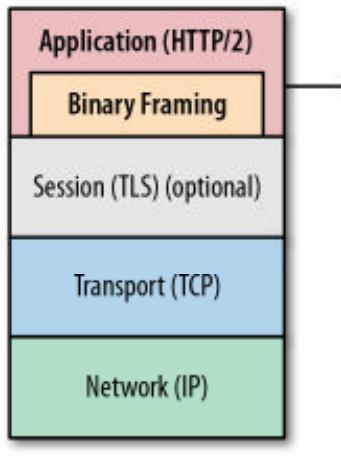
## HTTP 1.1 Response

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
...
```

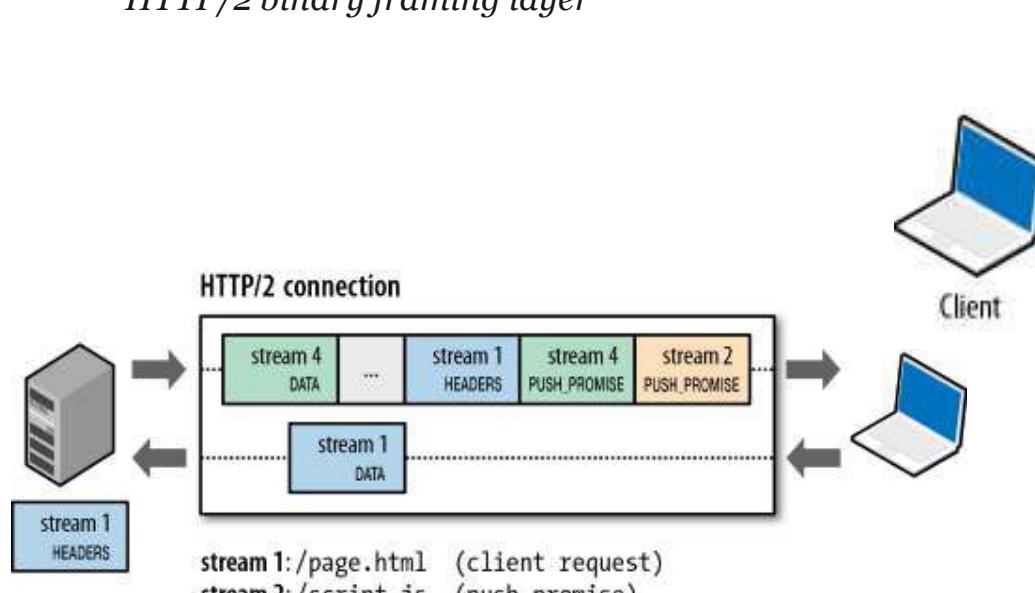
## HTTP/2 Response

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c
...
```

# HTTP/2



HTTP/2 binary framing layer



Server initiates new streams (promises) for push resources

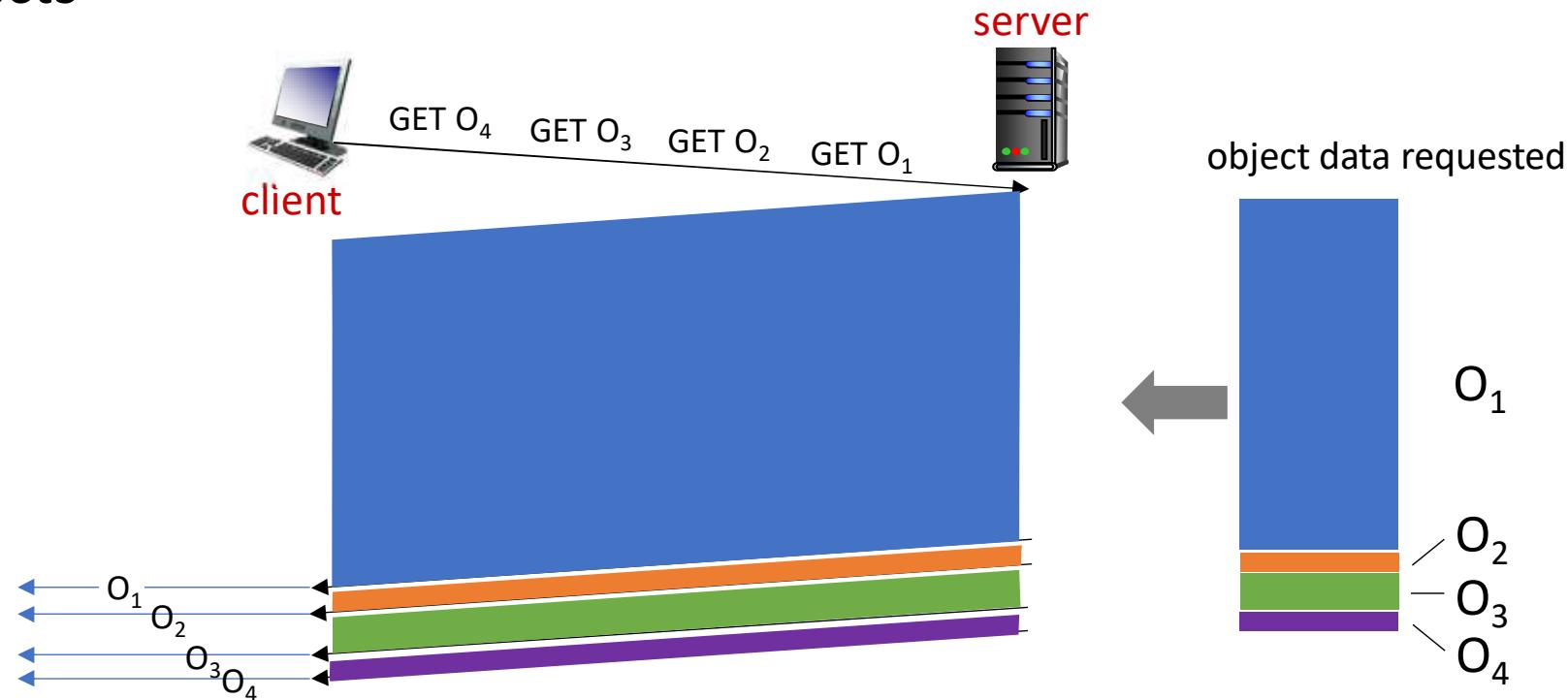
HTTP/2 connection

HTTP/2 request and response multiplexing within a shared connection

**[High Performance Browser Networking:** What every web developer should know about networking and web performance, Ilya Grigorik, O'Reilly Media, Sep 2013]

# HTTP/2: mitigating HOL blocking

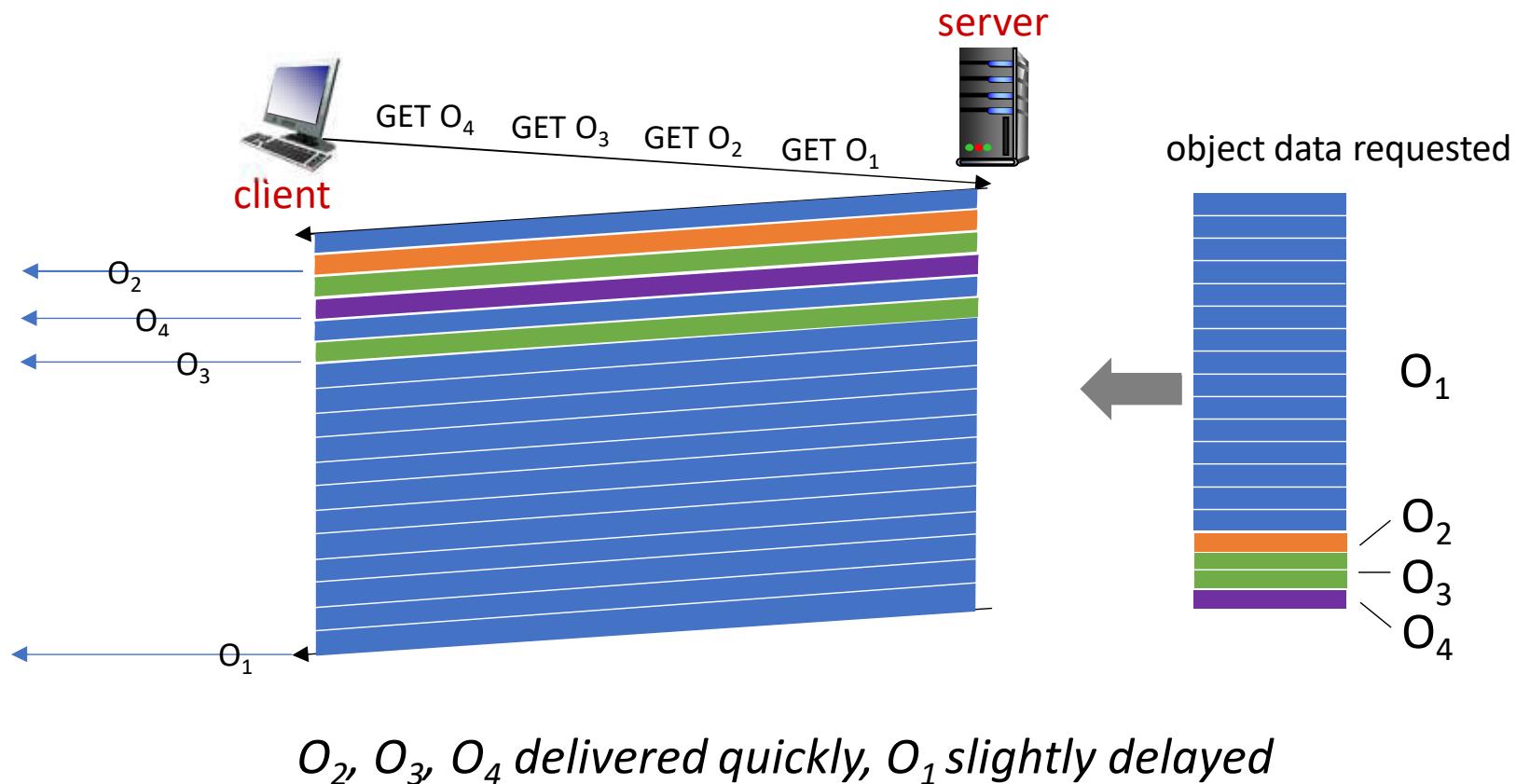
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



*objects delivered in order requested:  $O_2$ ,  $O_3$ ,  $O_4$  wait behind  $O_1$*

# HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



# HTTP/3 (2019)

HTTP/3

HTTP/2 introduced binary framing and multiplexing to **improve latency without modifying the transport layer**. However, a *lost or reordered packet causes all active transactions to experience a stall* regardless of whether that transaction was impacted by the lost packet.

HTTP/3 adds **security, per object error- and congestion-control** (more pipelining) **over UDP**.

Evolution of HTTP over Google's QUIC protocol - *UDP-based, stream-multiplexing, encrypted transport protocol*.

HTTP/3 **will not work on top of TCP**. HTTP/3 will speed the initial connection time taking advantage of *SSL session reuse*, which reduces overhead when multiple substreams are sent over a single connection. Adoption is expected to be gradual, as HTTP/2.

# HTTP/3

HTTP/3 is designed for QUIC, which is a transport protocol that handles streams by itself.

HTTP/2 is designed for TCP, and therefore handles streams in the HTTP layer.

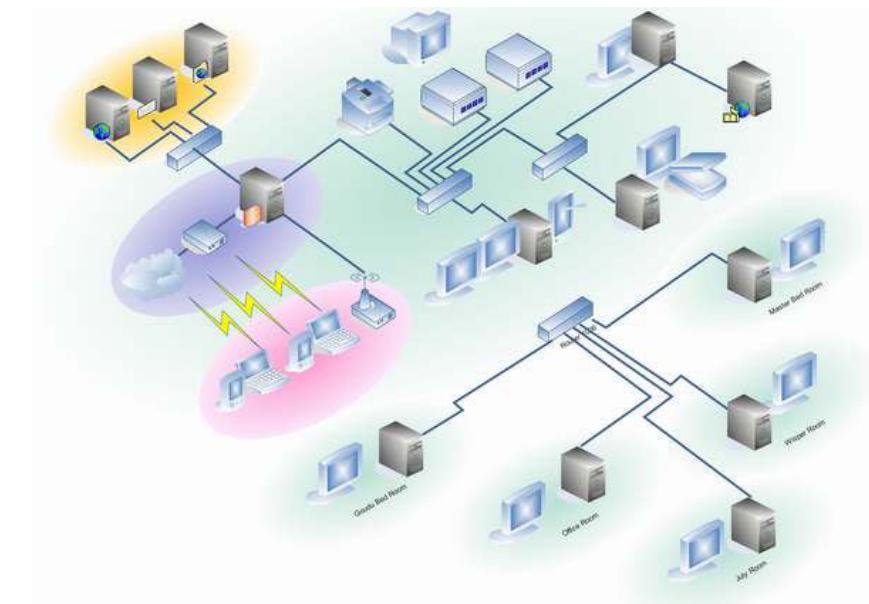
- **HTTP/3 has much faster handshakes** thanks to QUIC vs TCP + TLS.
- **HTTP/3 does not exist in an insecure or unencrypted version.**  
HTTP/2 can be implemented and used without HTTPS - even if this is rare on the Internet.

Browser support for HTTP/3

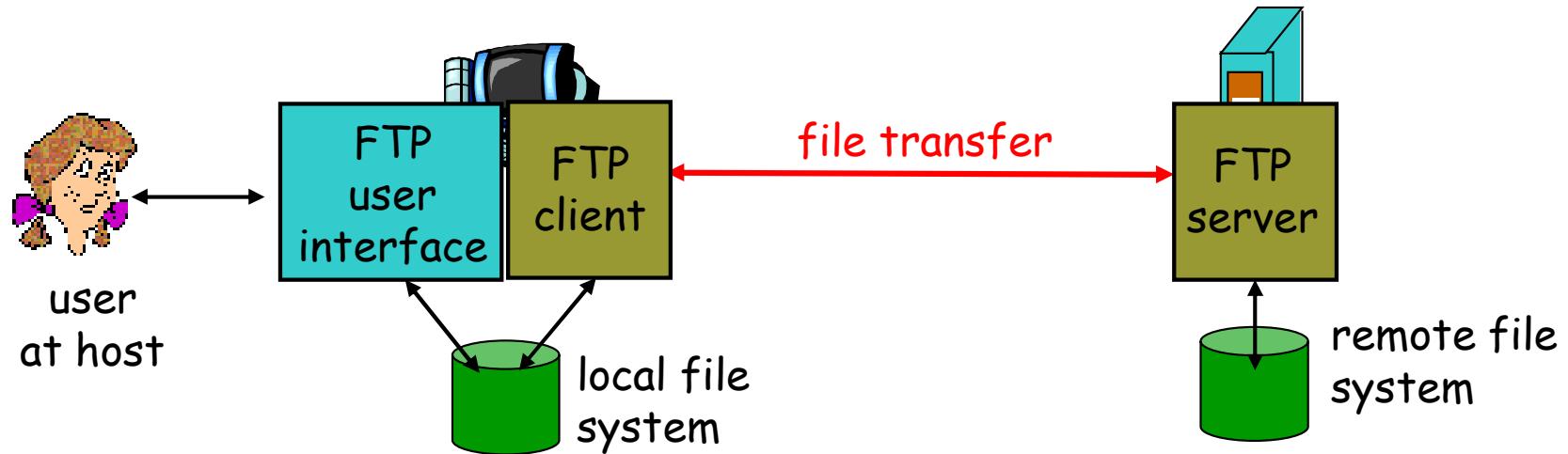
Browser	Version implemented (disabled by default)	Version shipped (enabled by default)	Comment	
Chrome	Stable build (79)	December 2019	87 <sup>[6]</sup>	April 2020 <sup>[25]</sup>
Edge	Stable build (79)	December 2019	87	April 2020
Firefox	Stable build (72.0.1)	January 2020	88 <sup>[9]</sup>	April 2021 <sup>[26]</sup>
Safari	Safari Technology Preview 104	April 2020	–	–

# Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- **FTP**
- Electronic Mail
- DNS
- P2P Applications

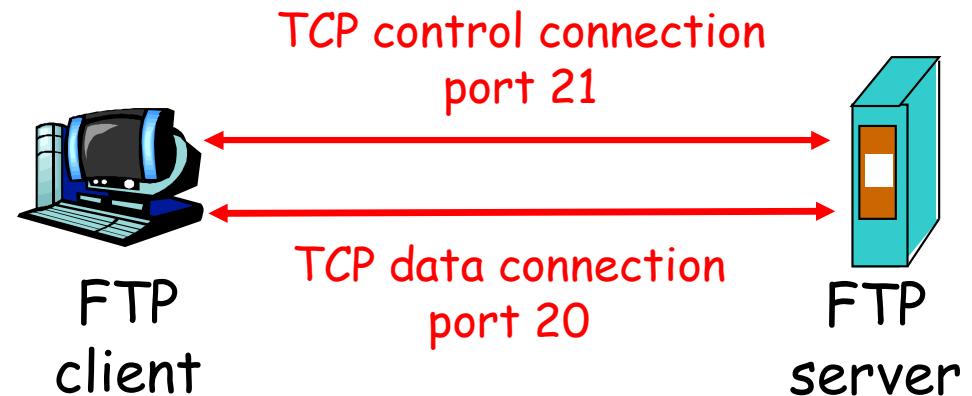


# FTP: File Transfer Protocol



- Transfer file to/from remote host.
- Client/Server model:
  - *Client*: side that initiates transfer (either to/from remote);
  - *Server*: remote host.
- FTP: RFC 959.
- FTP server: TCP port 21.

# FTP: Control and Data Connections



- FTP client contacts FTP server at port 21
  - TCP is transport protocol;
- Client authorized over control connection;
- Client browses remote directory by sending commands over control connection;
- When server receives **file transfer** command (e.g., *put*, *get*), server opens **2<sup>nd</sup> TCP connection (per file) to client (port 20)**;
- After transferring one file, server closes data connection;
- Server opens another TCP data connection to transfer another file;
- Control connection: “out of band”;
- **FTP server maintains state**: current directory, earlier authentication.

# FTP: Some User Commands

<b>ascii</b>	<i>set the mode of file transfer to ASCII (default - transmits seven bits per character)</i>
<b>binary</b>	<i>set transfer mode to binary (transmits eight bits per byte - used to transmit files other than ASCII)</i>
<b>bye</b>	<i>to exit the FTP environment (same as <b>quit</b>)</i>
<b>cd</b>	<i>to change directory on the remote machine</i>
<b>close</b>	<i>to terminate a connection with another computer</i>
<b>delete</b>	<i>to delete a file in the current remote directory (same as <b>rm</b> in UNIX)</i>
<b>get</b>	<i>to copy one file from the remote machine to the local machine</i>
<b>help</b>	<i>to request a list of all available FTP commands</i>
<b>lcd</b>	<i>to change directory on your local machine (same as UNIX <b>cd</b>)</i>
<b>ls</b>	<i>to list the names of the files in the current remote directory</i>
<b>mkdir</b>	<i>to make a new directory within the current remote directory</i>
<b>mget</b>	<i>to copy multiple files from the remote machine to the local machine</i>
<b>mput</b>	<i>to copy multiple files from the local machine to the remote machine</i>
<b>open</b>	<i>to open a connection with another computer</i>
<b>put</b>	<i>to copy one file from the local machine to the remote machine</i>
<b>pwd</b>	<i>to find out the pathname of the current directory on the remote machine</i>
<b>quit</b>	<i>to exit the FTP environment (same as <b>bye</b>)</i>
<b>rmdir</b>	<i>to remove (delete) a directory in the current remote directory</i>

# FTP Commands and Responses

## Some FTP commands:

Sent as ASCII text over control channel:

- **USER *username***
- **PASS *password***
- **LIST** - return list of files in current directory
- **RETR *filename*** - retrieves (gets) file
- **STOR *filename*** - stores (puts) file onto remote host.

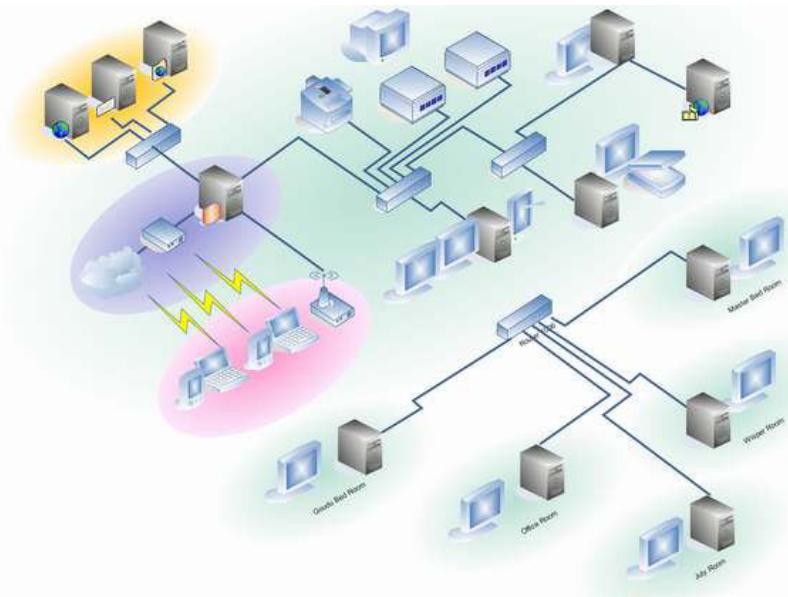
## Some return codes:

Status code and phrase (as in HTTP):

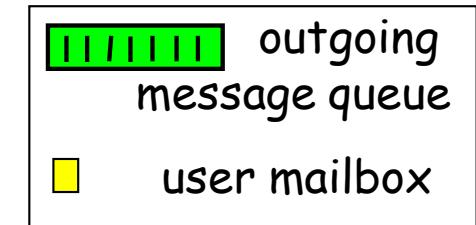
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

# Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- **Electronic Mail**
- DNS
- P2P Applications



# Electronic Mail

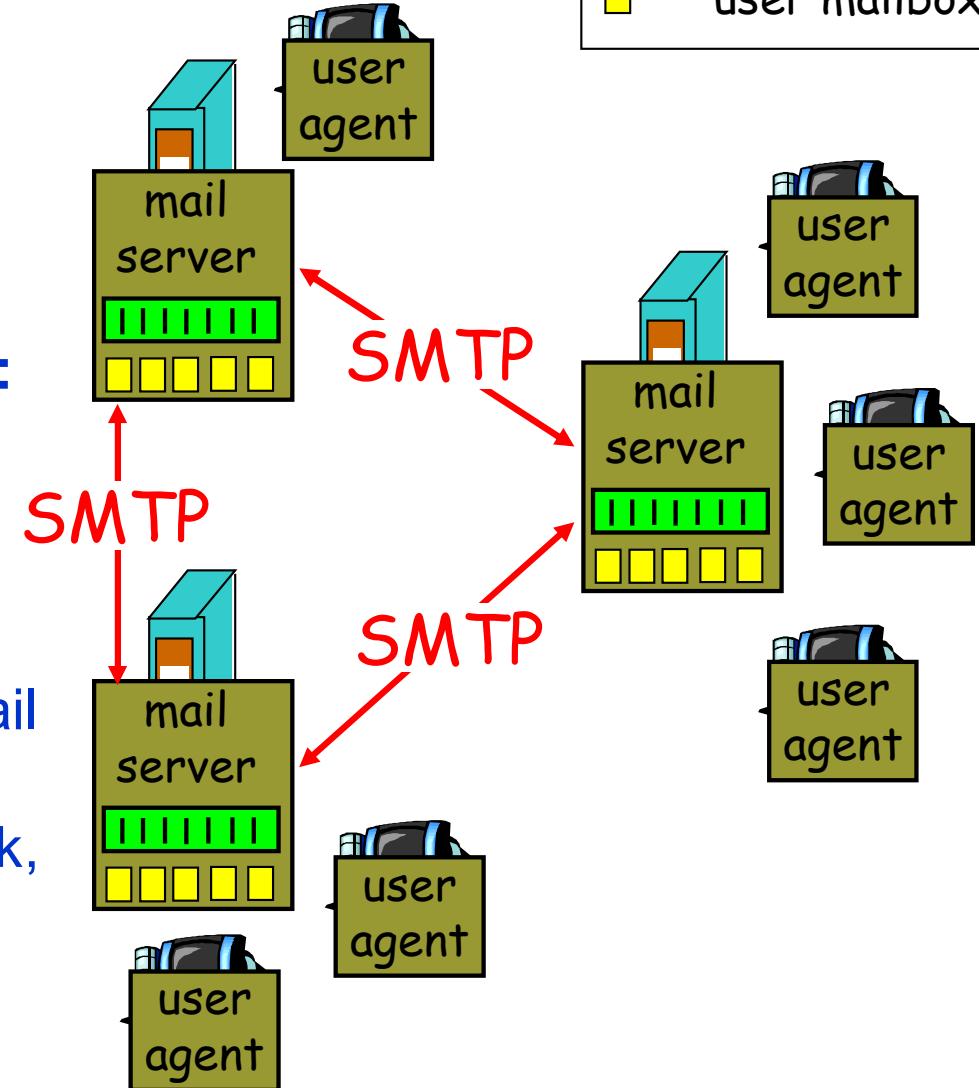


## Three major components:

- User agents
- Mail servers
- Simple mail transfer protocol:  
SMTP

### User Agent

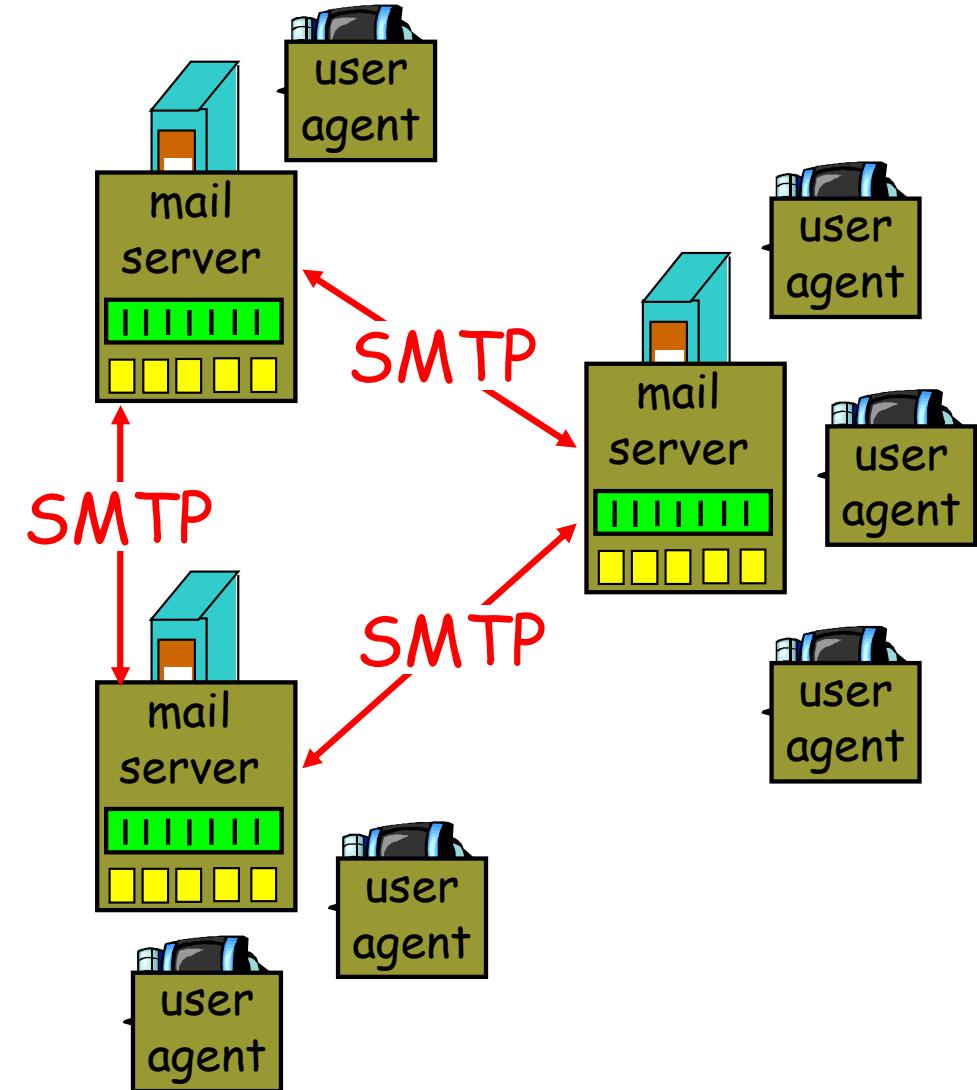
- The “mail reader”
- Composing, editing, reading mail messages
- Examples: Thunderbird, Outlook, eM, mailbird, Hiri, Spike, ...



# *Electronic Mail: Mail Servers*

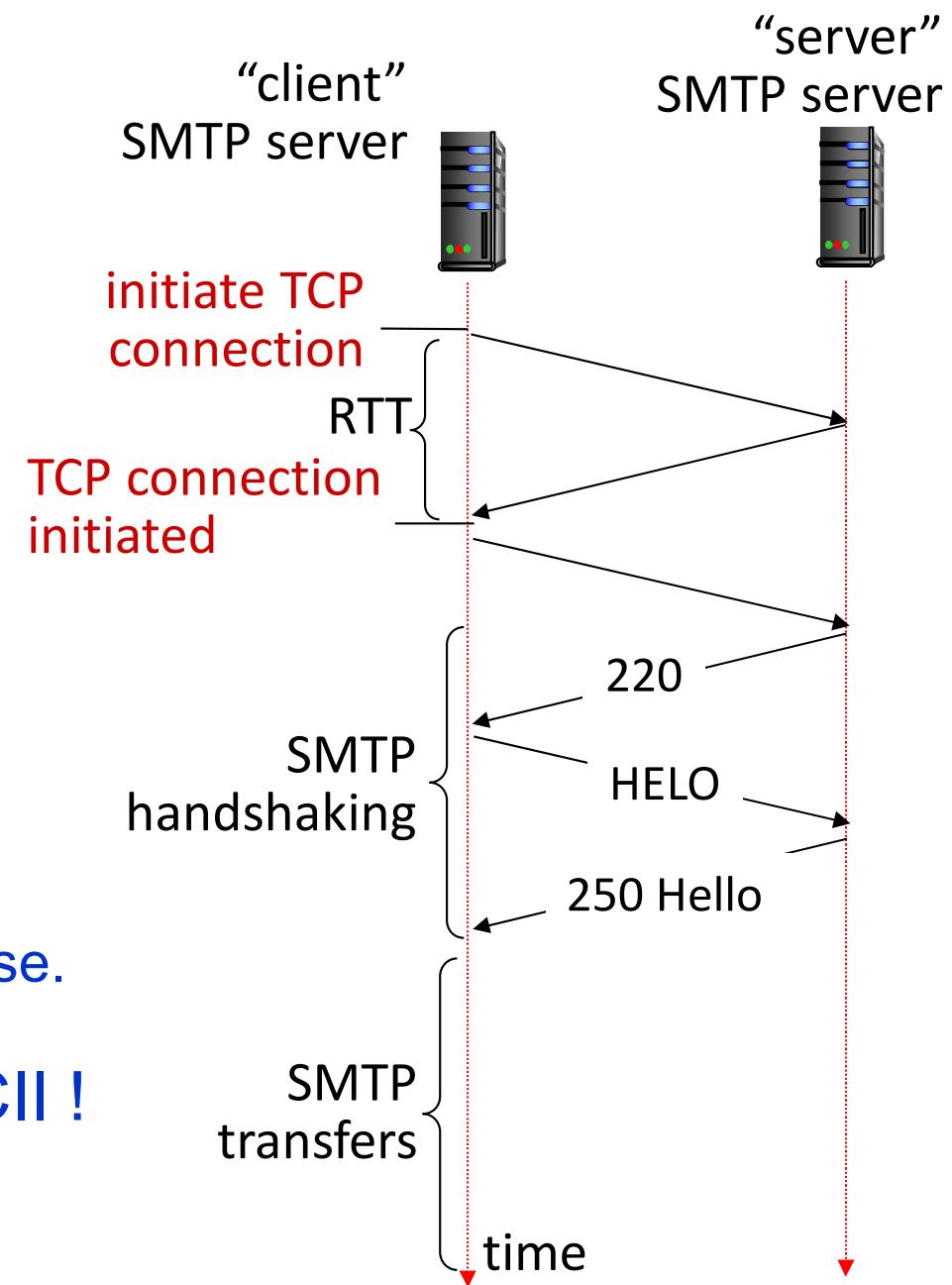
# Mail Servers

- ❑ 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.



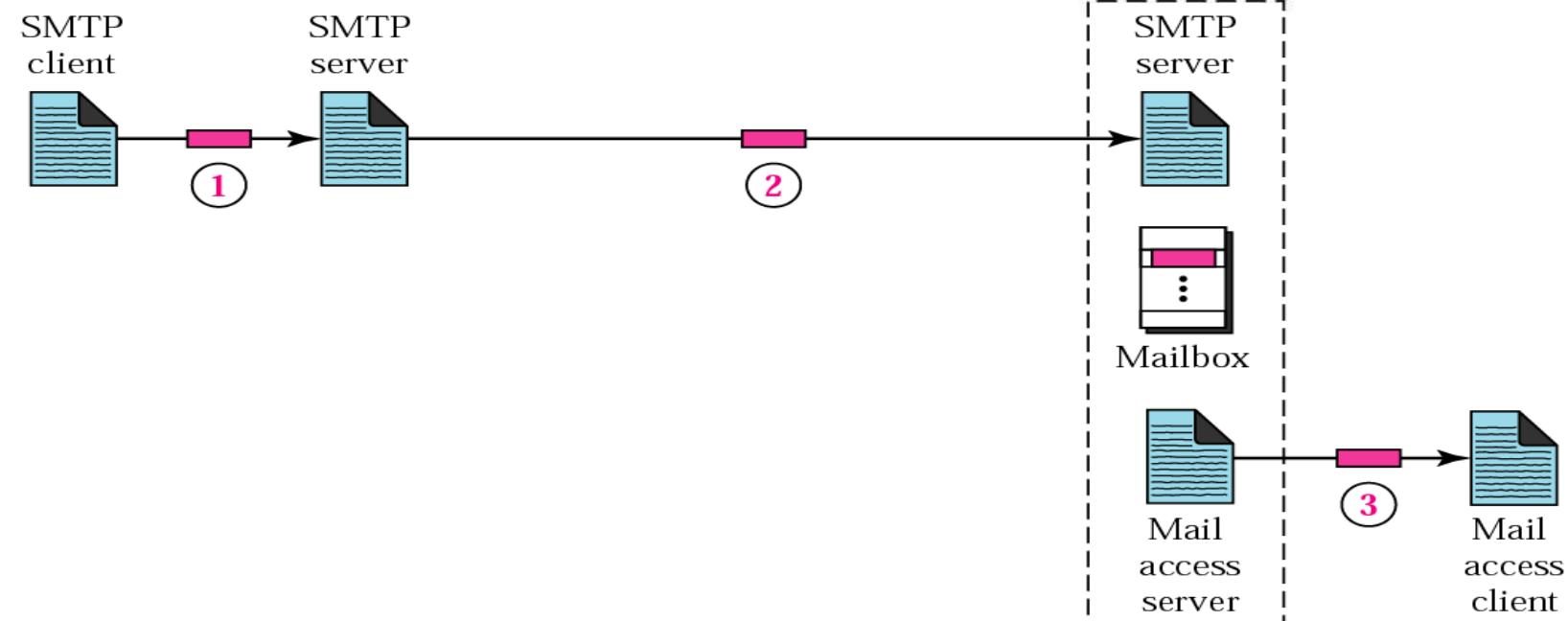
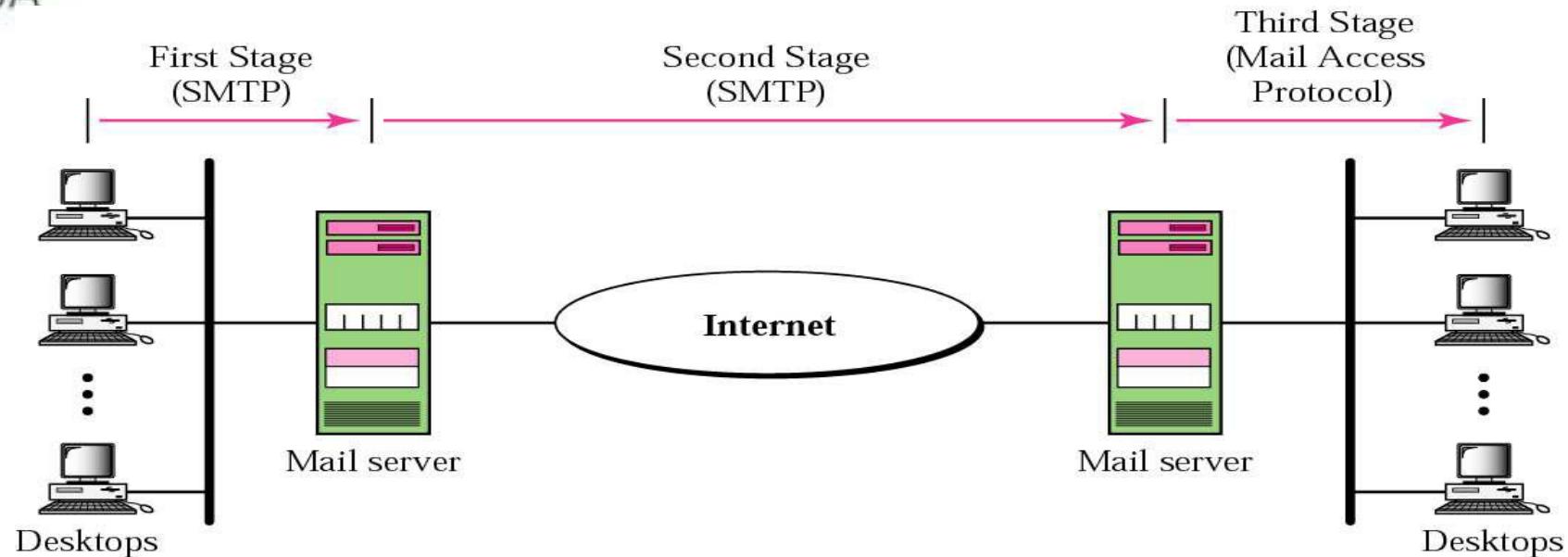
# Electronic Mail: SMTP Protocol [RFC 5321]

- Uses **TCP** to reliably transfer email message from client to server;
- Well-known **port: 25**;
- Three phases of transfer:
  - Handshaking (“greeting”);
  - Transfer of messages;
  - Closure.
- Command/Response interaction:
  - Commands: ASCII text;
  - Responses: status code and phrase.
- Messages must be in 7-bit ASCII !





# E-mail: Sending Message



## *SMTP: Example*

- nc servername 25
- See 220 reply from server;
- Enter commands:
  - HELO or EHLO
  - MAIL FROM
  - RCPT TO
  - DATA
  - QUIT

to send e-mail without using an e-mail client (reader).

## SMTP: Example

**S:** 220 lx.it.pt  
**C:** HELO meu.computador.pt  
**S:** 250 Hello meu.computador.pt, pleased to meet you  
**C:** MAIL FROM: <aluno@meu.computador.pt>  
**S:** 250 aluno@meu.computador.pt... Sender ok  
**C:** RCPT TO: <plc@lx.it.pt>  
**S:** 250 plc@lx.it.pt ... Recipient ok  
**C:** DATA  
**S:** 354 Enter mail, end with "." on a line by itself  
**C:** Teste do SMTP  
**C:** .  
**S:** 250 Message accepted for delivery  
**C:** QUIT  
**S:** 221 lx.it.pt closing connection

Exemplo: Telnet servidor.ist.utl.pt 25

# Mail Message Format

SMTP: protocol for **exchanging email messages** (RFC 5321, 7504).

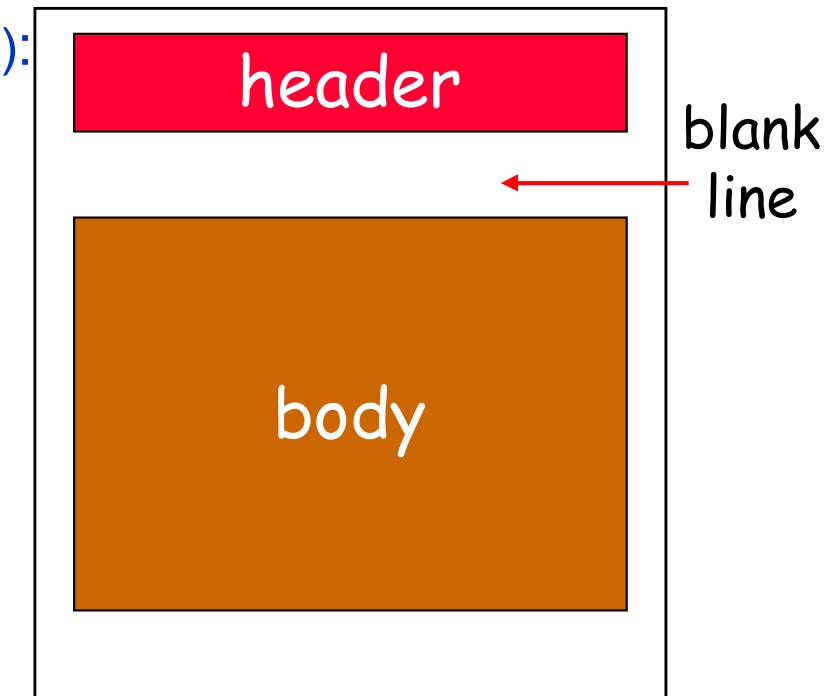
**SMTP**

Standard for **text message format** (RFC 5322):

- Header lines, e.g.:
  - From:
  - To:
  - Date:
  - Subject:
  - Received: (added by receiving SMTP server)

**Different from SMTP commands !**

- Body:
  - The “message”, ASCII characters only.



# E-mail: Formato da Mensagem



Sophia Fegan  
Com-Net  
Cupertino, CA 95014  
Sept. 16, 2002

Subject: Network

Dear Mrs. Fegan:  
We want to inform you that  
our network is working prop-  
erly after the last repair.

Yours truly,  
Behrouz Forouzan

Letter

Mail From: forouzan@deanza.edu  
RCPT To: fegan@comnet.com

---

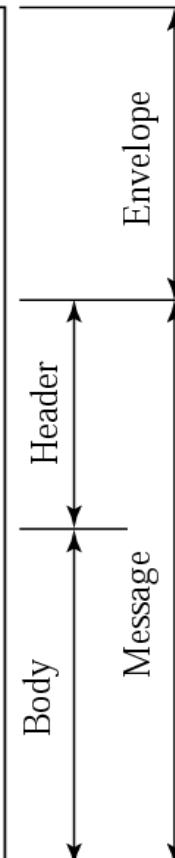
From: Behrouz Forouzan  
To: Sophia Fegan  
Date: 9/16/02  
Subject: Network

---

Dear Mrs. Fegan:  
We want to inform you that  
our network is working prop-  
erly after the last repair.

Yours truly,  
Behrouz Forouzan

E-mail

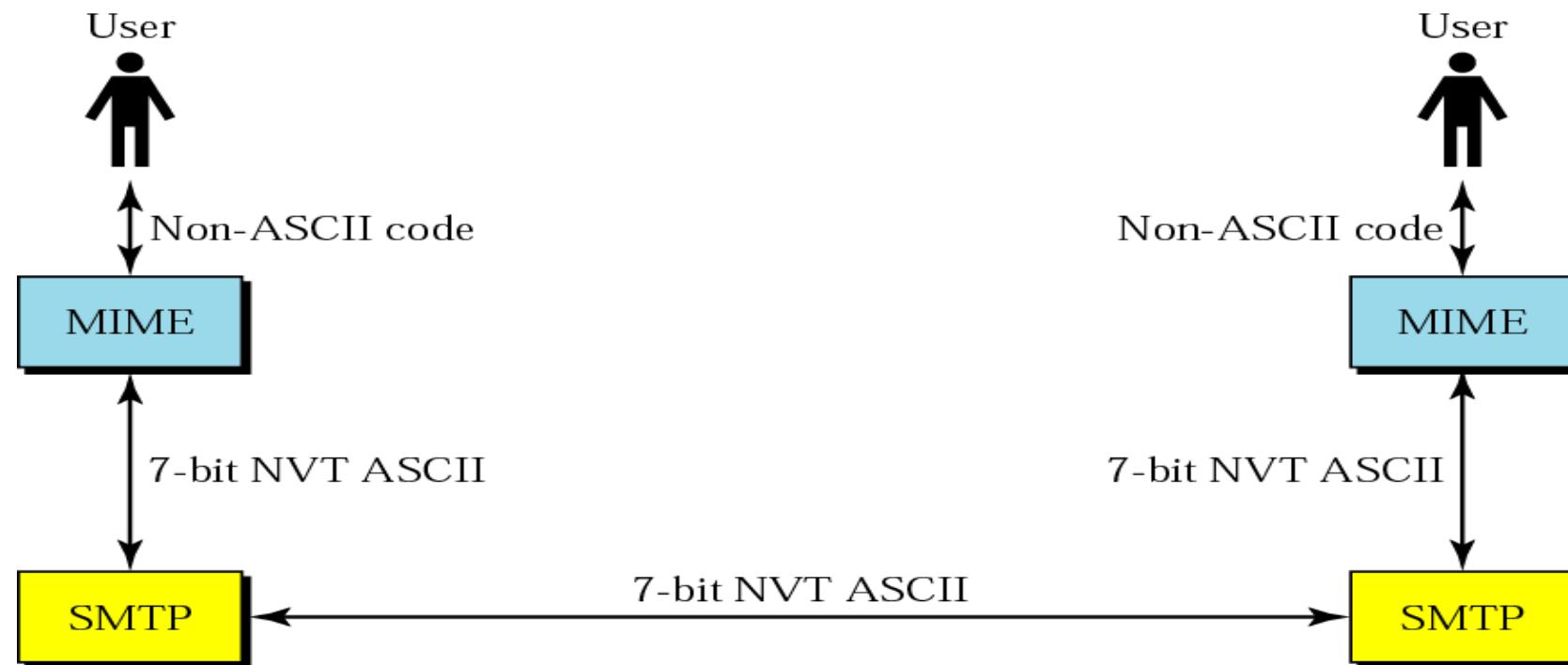


**SMTP**



# Message Format: Multimedia Extensions

- MIME: multimedia mail extension, RFC 2045, 2056.



# ***Message Format: Multimedia Extensions***

- Additional lines in message header declare MIME content type.

## RFC 822 headers added by MIME:

<b>Header</b>	<b>Meaning</b>
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

MIME version  
method used to encode data  
multimedia data  
type, subtype,  
parameter declaration  
encoded data

Subject: Picture of yummy crepe.  
MIME-Version: 1.0  
Content-Transfer-Encoding: base64  
Content-Type: image/jpeg  
base64 encoded data .....  
.....  
.....base64 encoded data

# *Message Format: Multimedia Extensions*

MIME types and subtypes defined in RFC 2045 :

Type	Subtype	Description
Text	Plain	Unformatted text
	Enriched	Text including simple formatting commands
Image	Gif	Still picture in GIF format
	Jpeg	Still picture in JPEG format
Audio	Basic	Audible sound
Video	Mpeg	Movie in MPEG format
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in PostScript
Message	Rfc822	A MIME RFC 822 message
	Partial	Message has been split for transmission
	External-body	Message itself must be fetched over the net
Multipart	Mixed	Independent parts in the specified order
	Alternative	Same message in different formats
	Parallel	Parts must be viewed simultaneously
	Digest	Each part is a complete RFC 822 message

# Message Format: Multimedia Extensions

From: elinor@abcd.com  
To: carolyn@xyz.com  
MIME-Version: 1.0  
Message-Id: <0704760941.AA00747@abcd.com>  
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm  
Subject: Earth orbits sun integral number of times

**MIME multipart message** containing formatting and audio.

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm  
Content-Type: text/enriched

Happy birthday to you  
Happy birthday to you  
Happy birthday dear <b>Carolyn</b>  
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm  
Content-Type: message/external-body;  
access-type="anon-ftp";  
site="bicycle.abcd.com";  
directory="pub";  
name="birthday.snd"

content-type: audio/basic  
content-transfer-encoding: base64  
--qwertyuiopasdfghjklzxcvbnm--

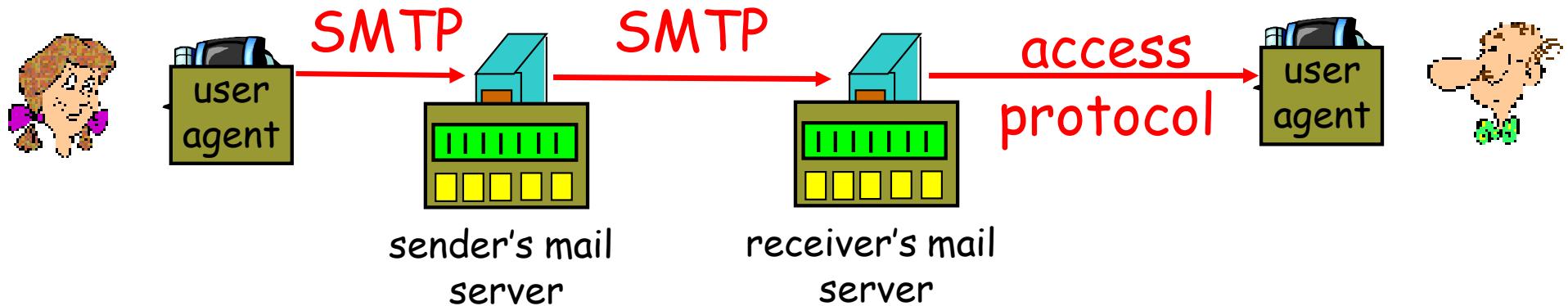
# *SMTP: Simple Mail Transfer Protocol*

- SMTP uses persistent connections;
- SMTP requires message (header & body) to be in 7-bit ASCII.

## Comparison with HTTP:

- HTTP: pull application;
- SMTP: push application;
- Both use ASCII command/response interaction, status codes;
- HTTP: objects are encapsulated in response messages;
- SMTP: multiple objects are sent in multipart messages.

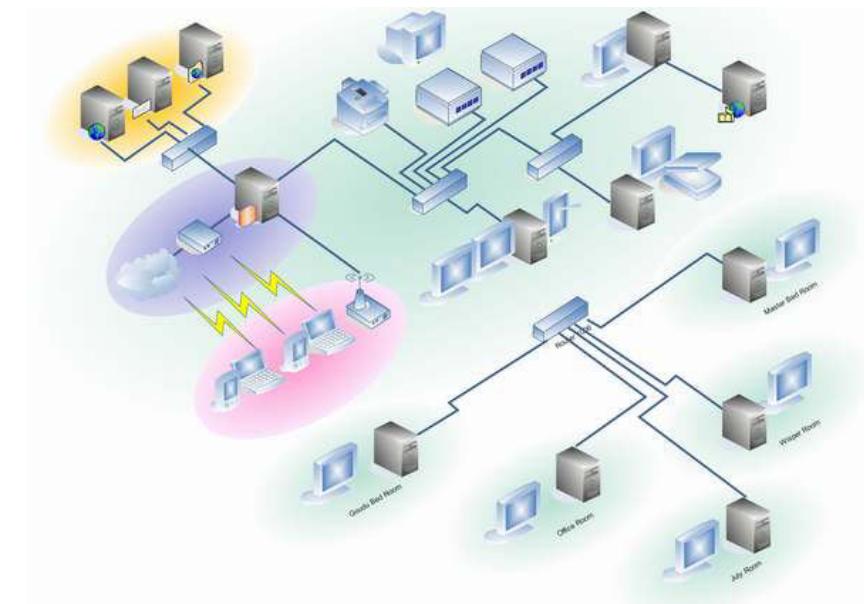
# Mail Access Protocols



- SMTP: delivery/storage to receiver's server;
- Mail access protocols are used to retrieve e-mail from server:
  - **POP**: Post Office Protocol [RFC 1939]
    - Authorization (agent <-->server) and download;
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]
    - More features (more complex);
    - Manipulation of stored messages on server;
  - **HTTP**: Gmail, Hotmail, Yahoo! Mail, etc.

# Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- Electronic Mail
- **DNS**
- P2P Applications



# DNS: Domain Name System

People use many identifiers:

- name, #IST, #cartão cidadão, #passport, ...

Internet hosts, routers:

- **IP address** (32 bit) - used for addressing datagrams; e.g., 193.136.128.1;
- **Hostname**, e.g., www.yahoo.com - used by humans;

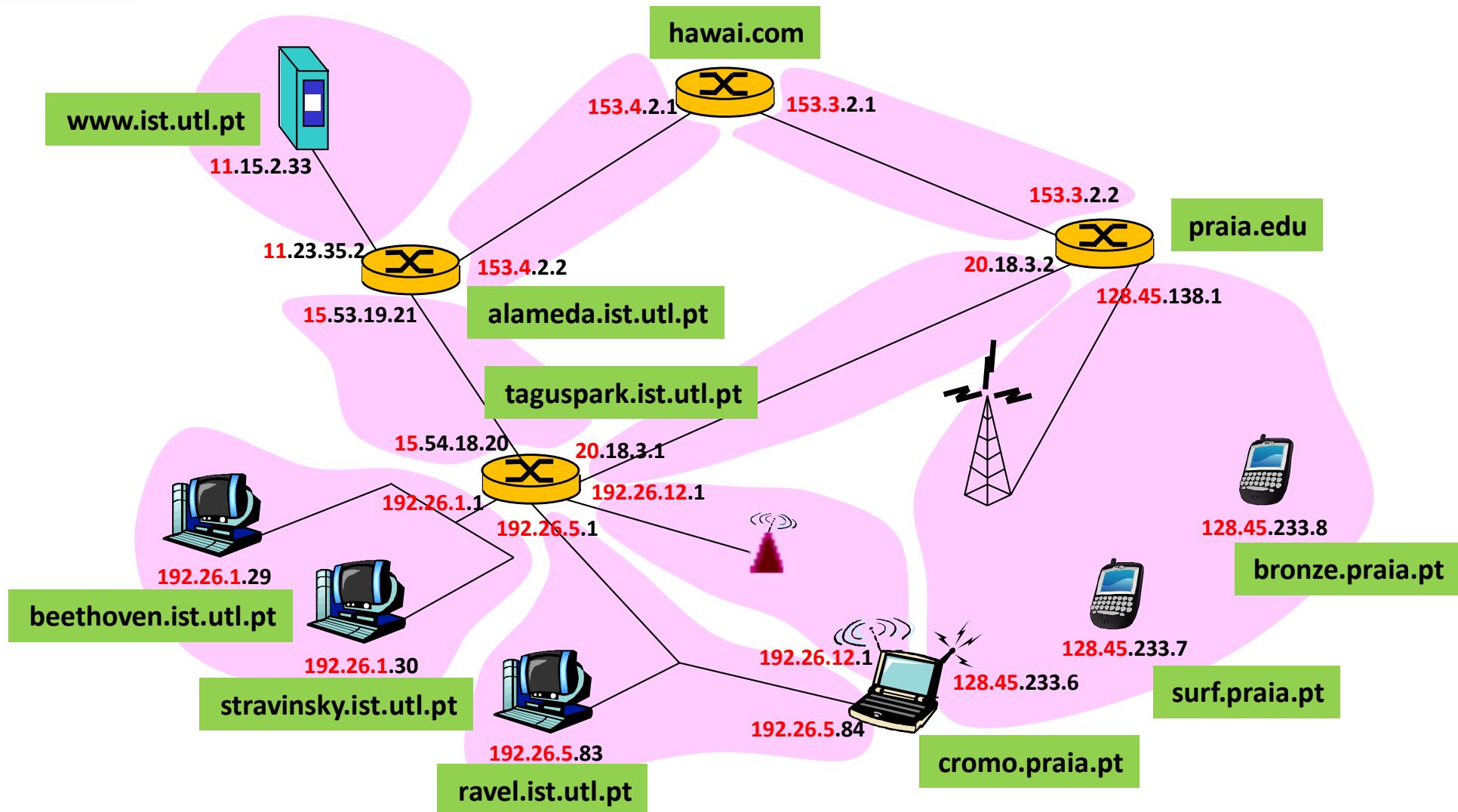
Q: How to map between IP addresses and hostnames?

```
struct addrinfo hints, *res;
n = getaddrinfo("tejo.tecnico.ulisboa.pt", PORT, &hints, &res);

struct addrinfo{                                     // (item in a linked list)
    ...
    socklen_t          ai_addrlen;      // address length (bytes)
    struct sockaddr *ai_addr;        // socket address
    ...
}
struct sockaddr_in {
    sa_family_t      sin_family;     // address family: AF_INET
    u_int16_t         sin_port;       // port (16 bits)
    struct in_addr sin_addr;        // internet address
}
struct in_addr{
    uint32_t          s_addr;         // IPv4 (32 bits)
};
```



# Internet Names: DNS



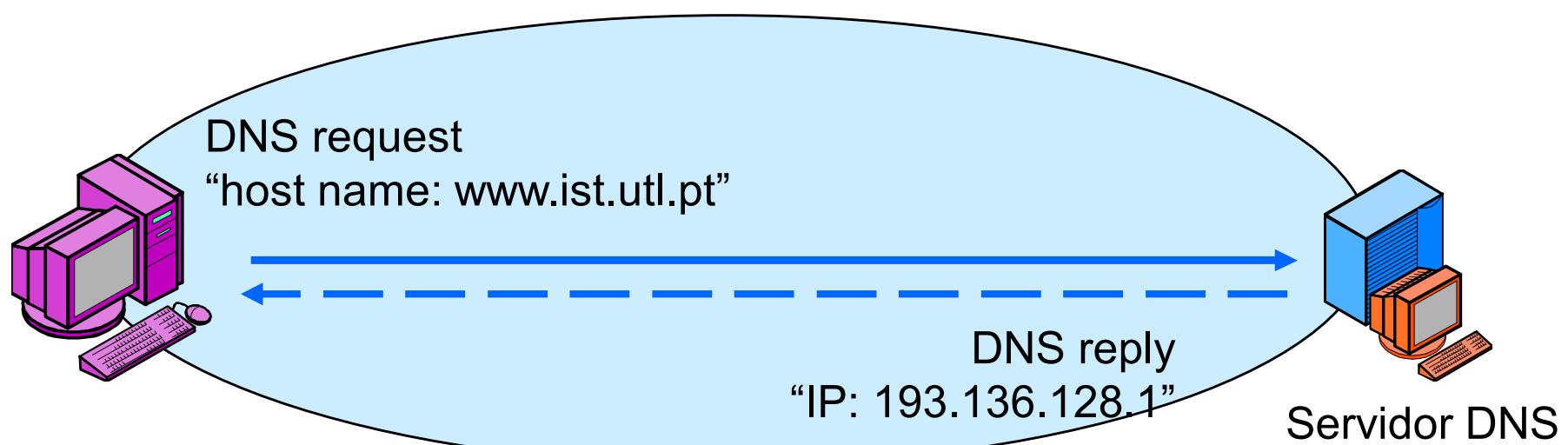
Run [nslookup](#)

# DNS: Domain Name System

How to map between IP addresses and hostnames?

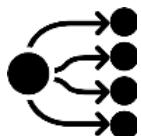
## Domain Name System (DNS):

- *Distributed database:* implemented in hierarchy of *name servers*;
- *Application-layer protocol:*  
hosts, routers, name servers communicate to *resolve* names
  - (*Name → Address*) The translation uses the services of **UDP (port 53)**.



## DNS services

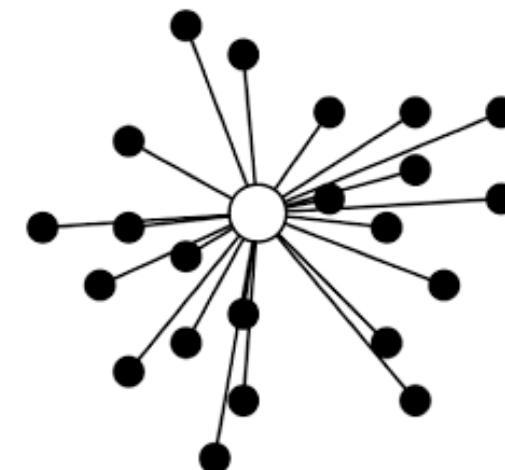
- Hostname to IP address translation;
- Host aliasing:
  - Canonical hostname and alias names;
- Load distribution:
  - Replicated Web servers: set of IP addresses for one canonical name.



## Why not centralize DNS?

- Single point of failure;
- Traffic volume;
- Distant centralized database;
- Maintenance issues.

*Doesn't scale!*



# Thinking about the DNS

Humongous distributed database:

- ~ billion records, each simple

Handles many *trillions* of queries/day:

- *many* more reads than writes
- *performance matters: almost every Internet transaction interacts with DNS - msecs count!*

Organizationally, physically decentralized:

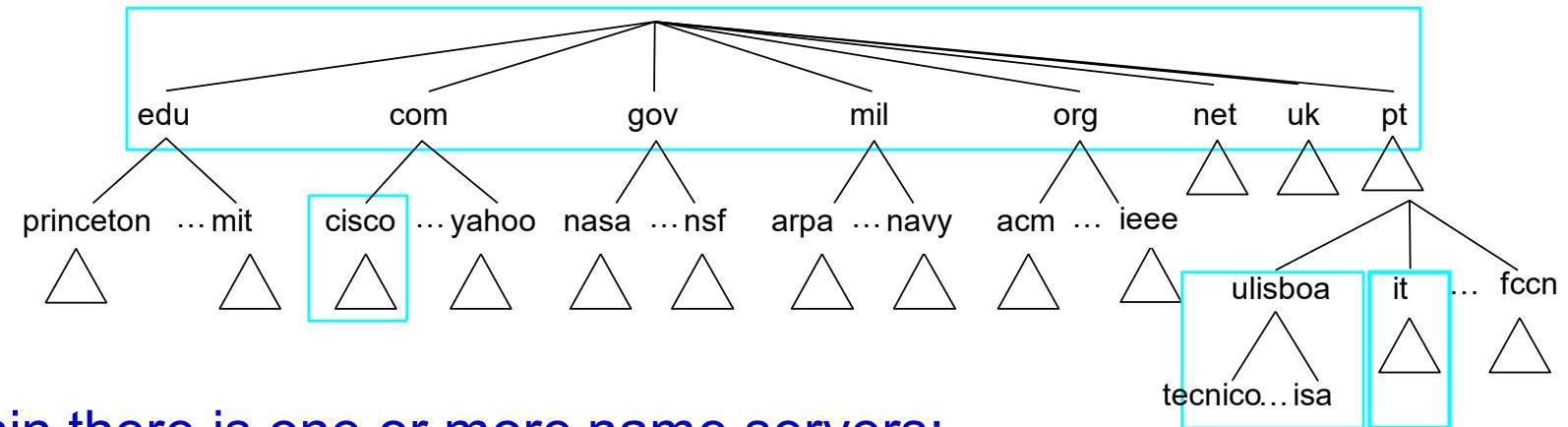
- millions of different organizations responsible for their records

“Bulletproof”: reliability, security

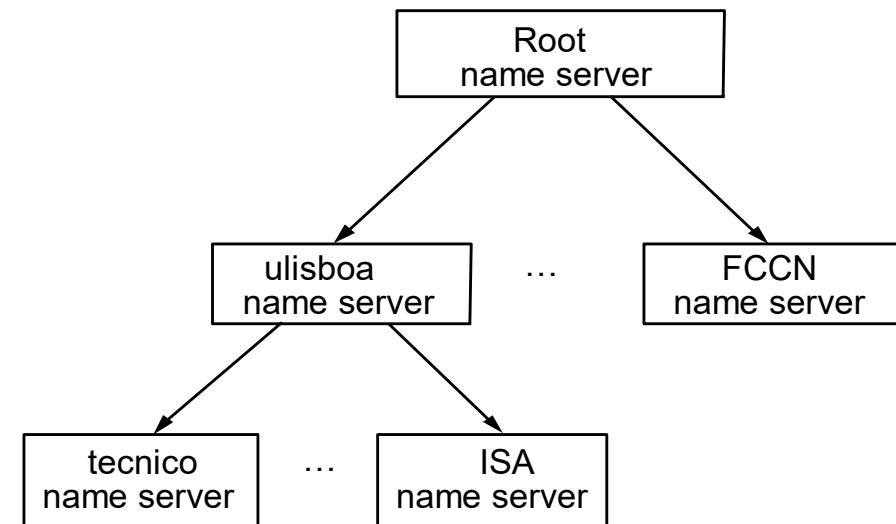


# DNS: Distributed, Hierarchical Database

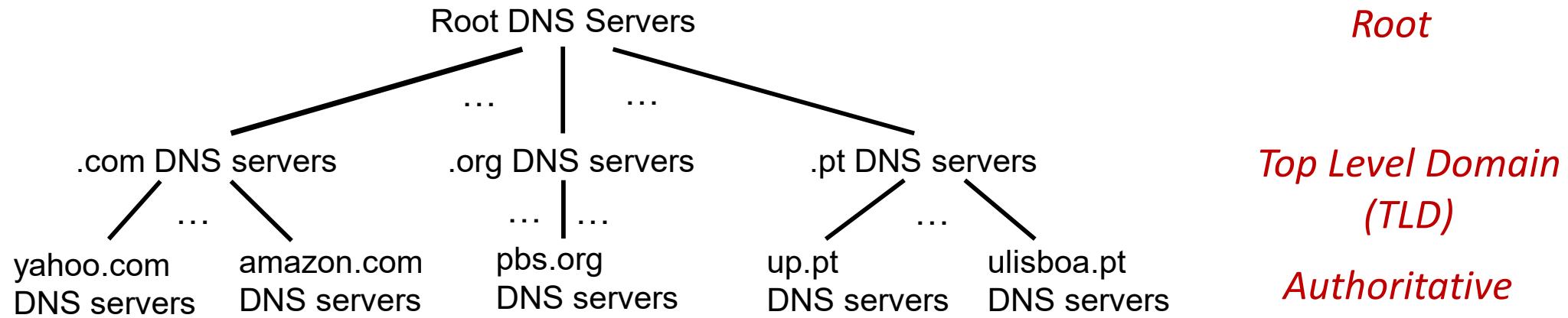
The naming hierarchy is divided into domains:



In each domain there is one or more name servers:

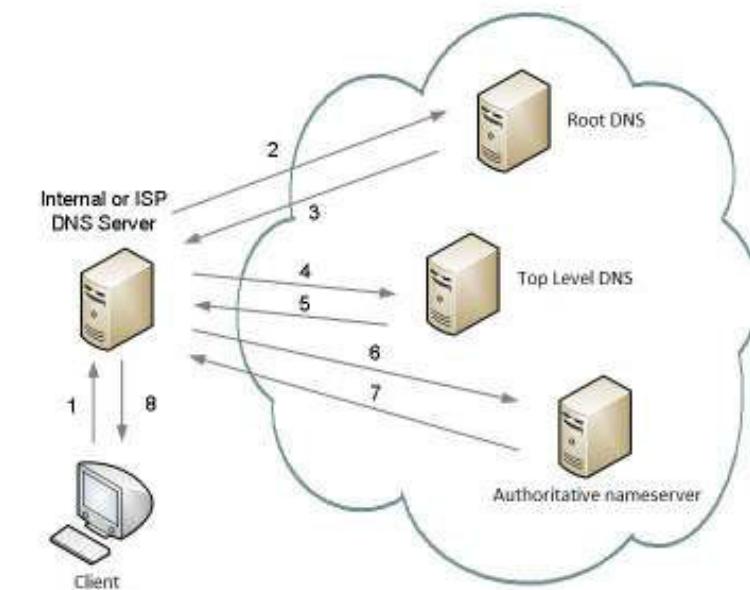


# DNS: a distributed, hierarchical database



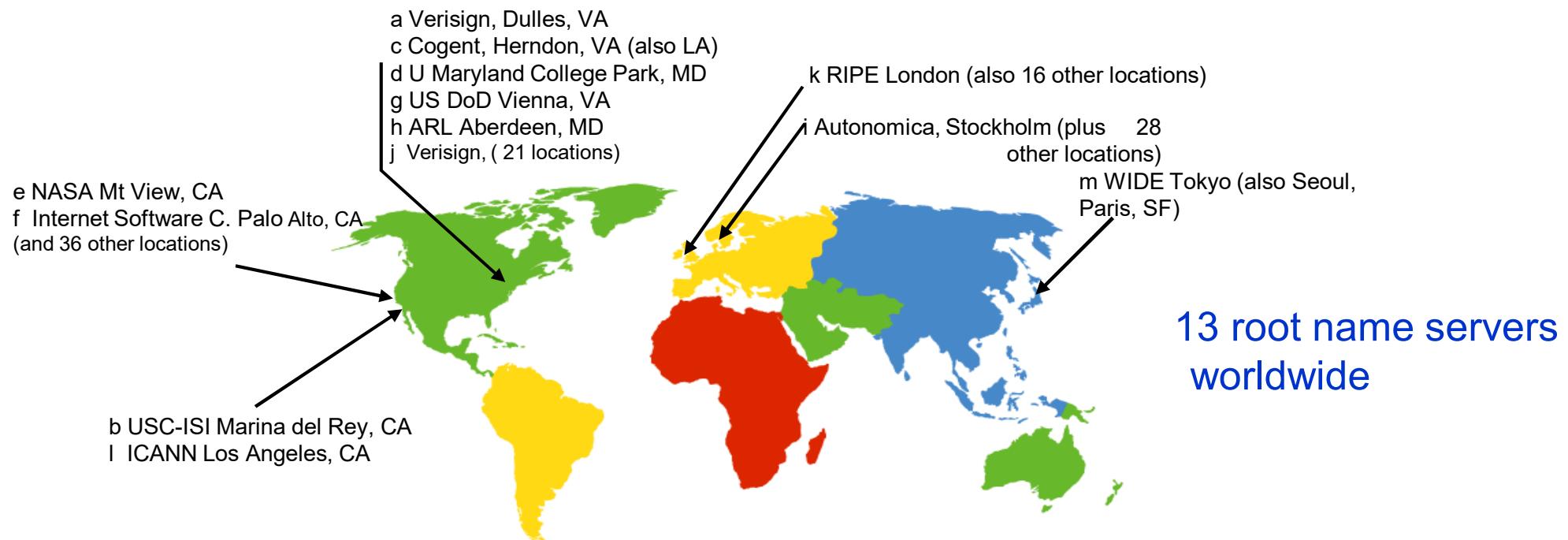
Client wants IP address for [www.amazon.com](http://www.amazon.com):

1. client queries **root DNS server** to find .com DNS server
2. client queries **.com TLD DNS server** to get amazon.com authoritative DNS server
3. client queries amazon.com **authoritative 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.



# Root Servers

The authoritative name servers that serve the DNS root zone, commonly known as the "root servers", are a network of hundreds of servers in many countries around the world. They are configured in the DNS root zone as 13 named authorities, as follows.

## List of Root Servers

Hostname	IP Addresses	Manager
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	192.228.79.201, 2001:500:84::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4	US Department of Defence (NIC)
h.root-servers.net	128.63.2.53, 2001:500:1::803f:235	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:3::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project



# DNS: Root Name Servers



# DNS: Root Name Servers

Porto, Portugal	
Operator	ICANN
IPv4	199.7.83.42
IPv6	2001:500:9f:42
ASN	20144



Lisbon, PT	
Operator	Internet Systems Consortium, Inc.
IPv4	192.5.5.241
IPv6	2001:500:2f:f
ASN	3557

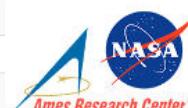


**D Root**

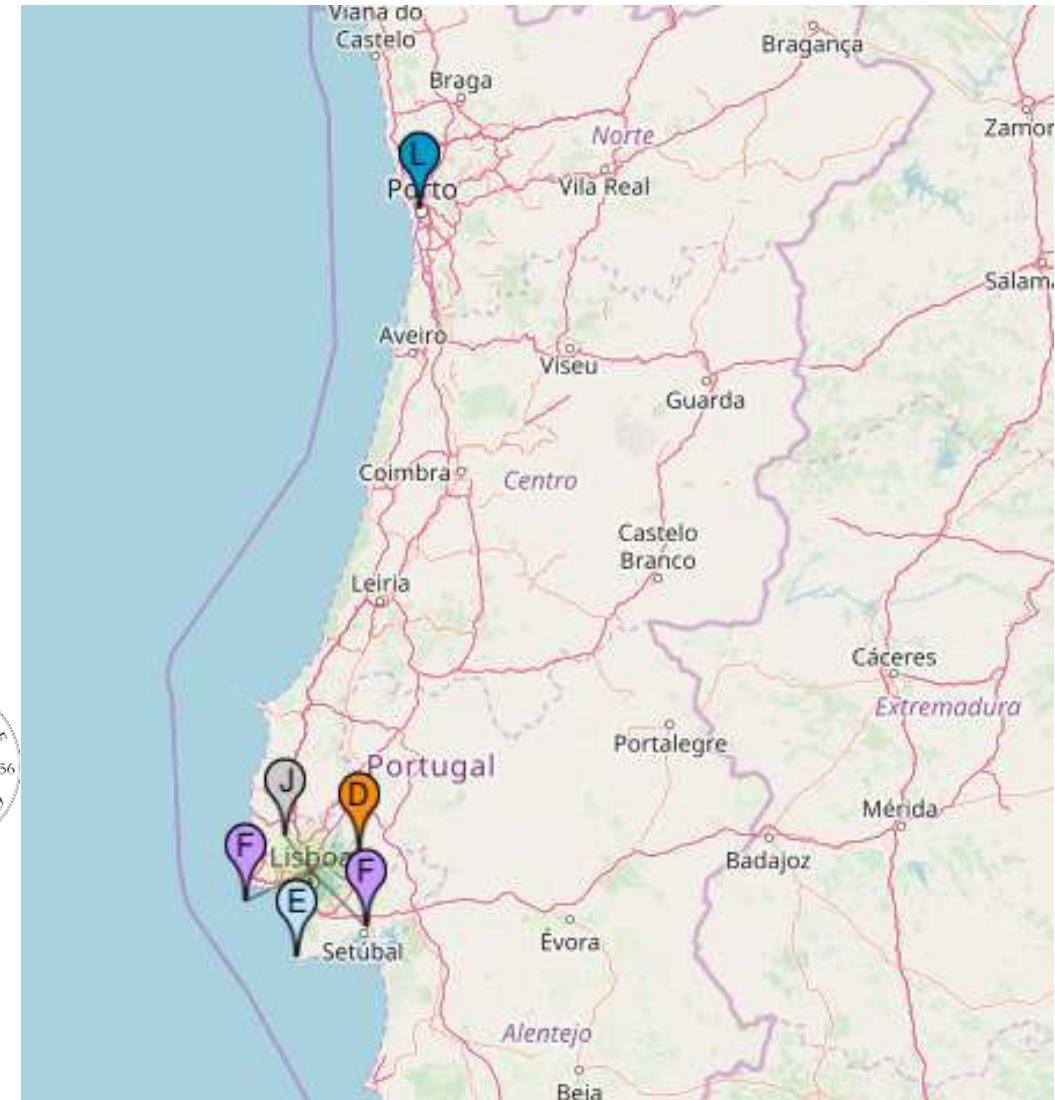
Lisbon, PT	
Operator	Verisign, Inc.
IPv4	192.58.128.30
IPv6	2001:503:c27::2:30
ASN	26415



Lisbon, PT	
Operator	NASA Ames Research Center
IPv4	192.203.230.10
IPv6	2001:500:a8::e
ASN	21556



**E Root**



[<http://www.root-servers.org/>]

# DNS: TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - Responsible for *com*, *org*, *net*, *edu*, *etc*, and all top-level country domains *pt*, *uk*, *fr*, *ca*, *jp*;
  - Network Solutions maintains servers for *com* TLD;
  - Educause for *edu* TLD.
- Authoritative DNS servers:
  - Organization's DNS servers;
  - Provide authoritative hostname to IP mappings for organization's servers (e.g., Web, mail);
  - Can be maintained by the organization or a service provider.

# DNS: Local Name Server



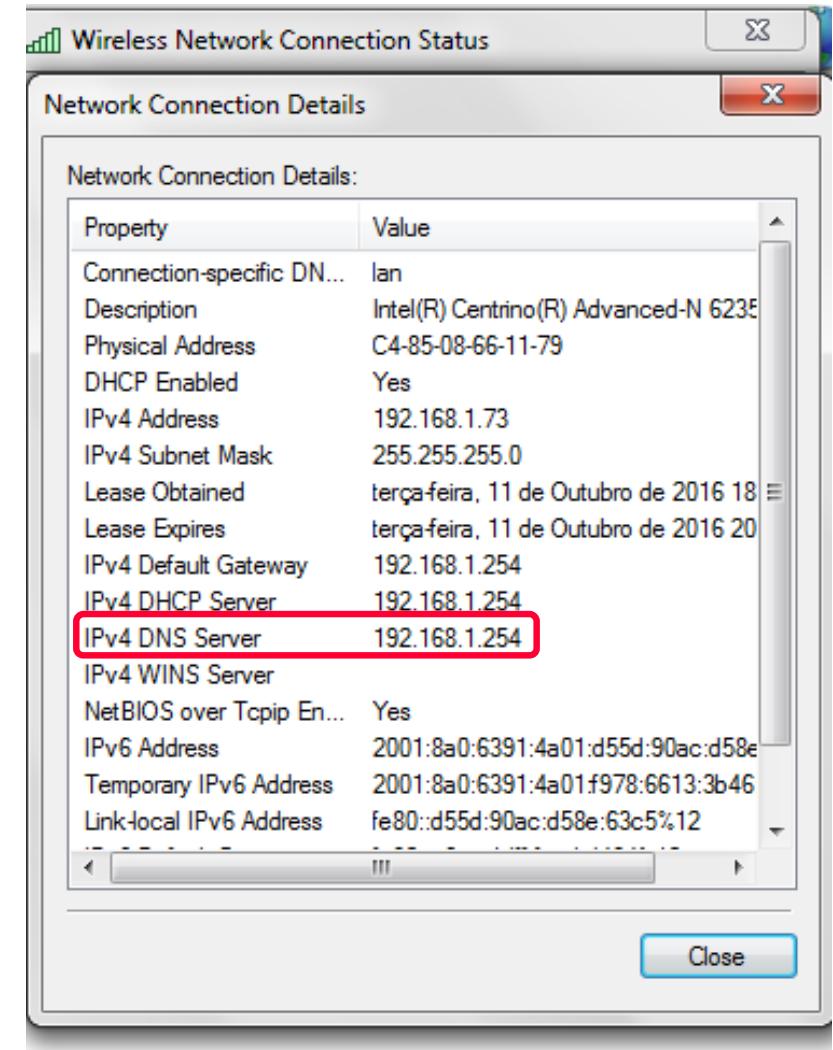
- Does not strictly belong to hierarchy;
- Each ISP (residential ISP, company, university) has one:
  - Also called “**default name server**”;
- When host makes DNS query, query is sent to its local DNS server:
  - Acts as proxy;
  - Forwards query into hierarchy.

IPv4 DNS servers:

212.113.177.241 (Unencrypted)

62.169.70.160 (Unencrypted)

192.168.68.1 (Unencrypted)

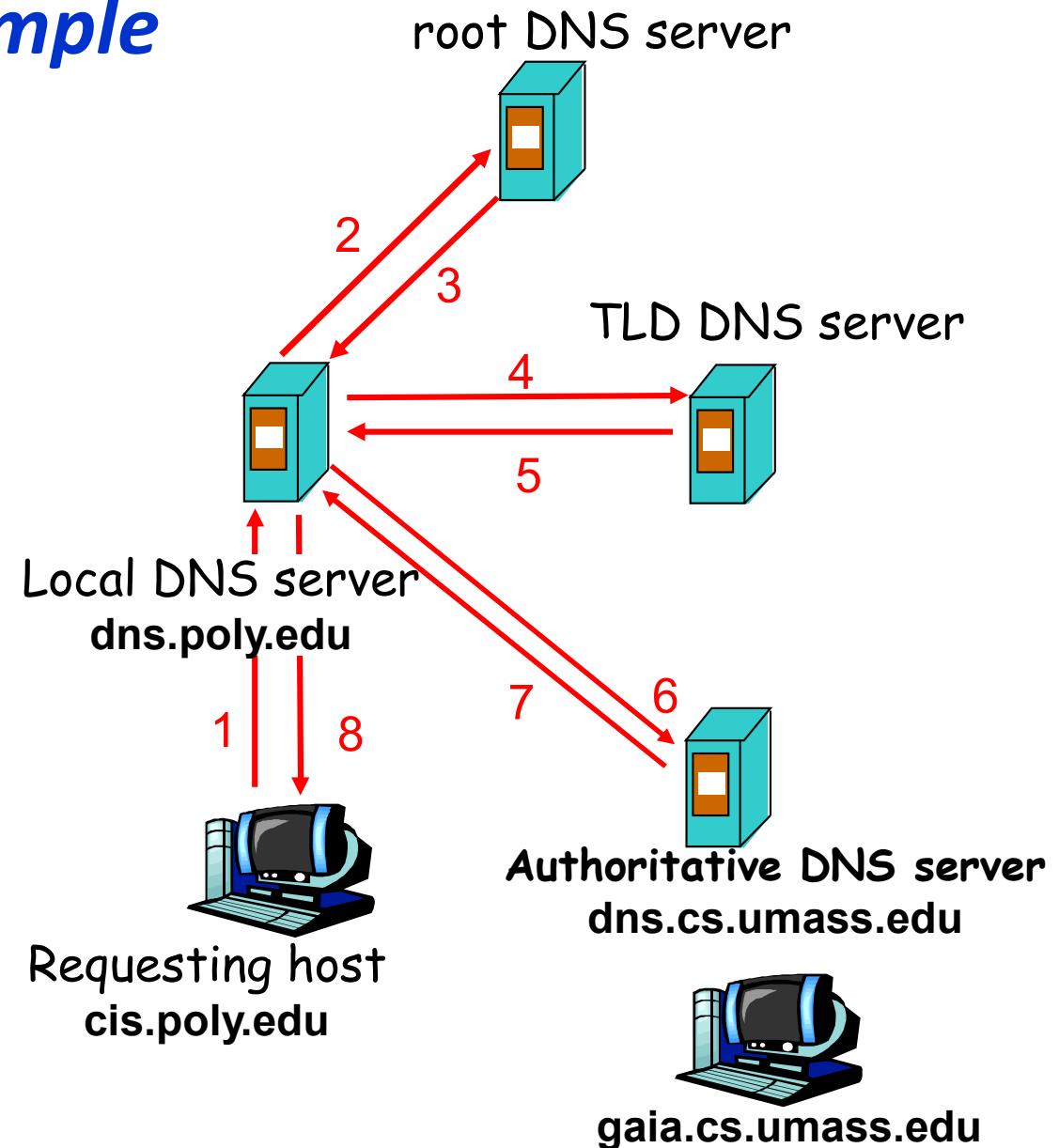


# DNS: Example

- 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”



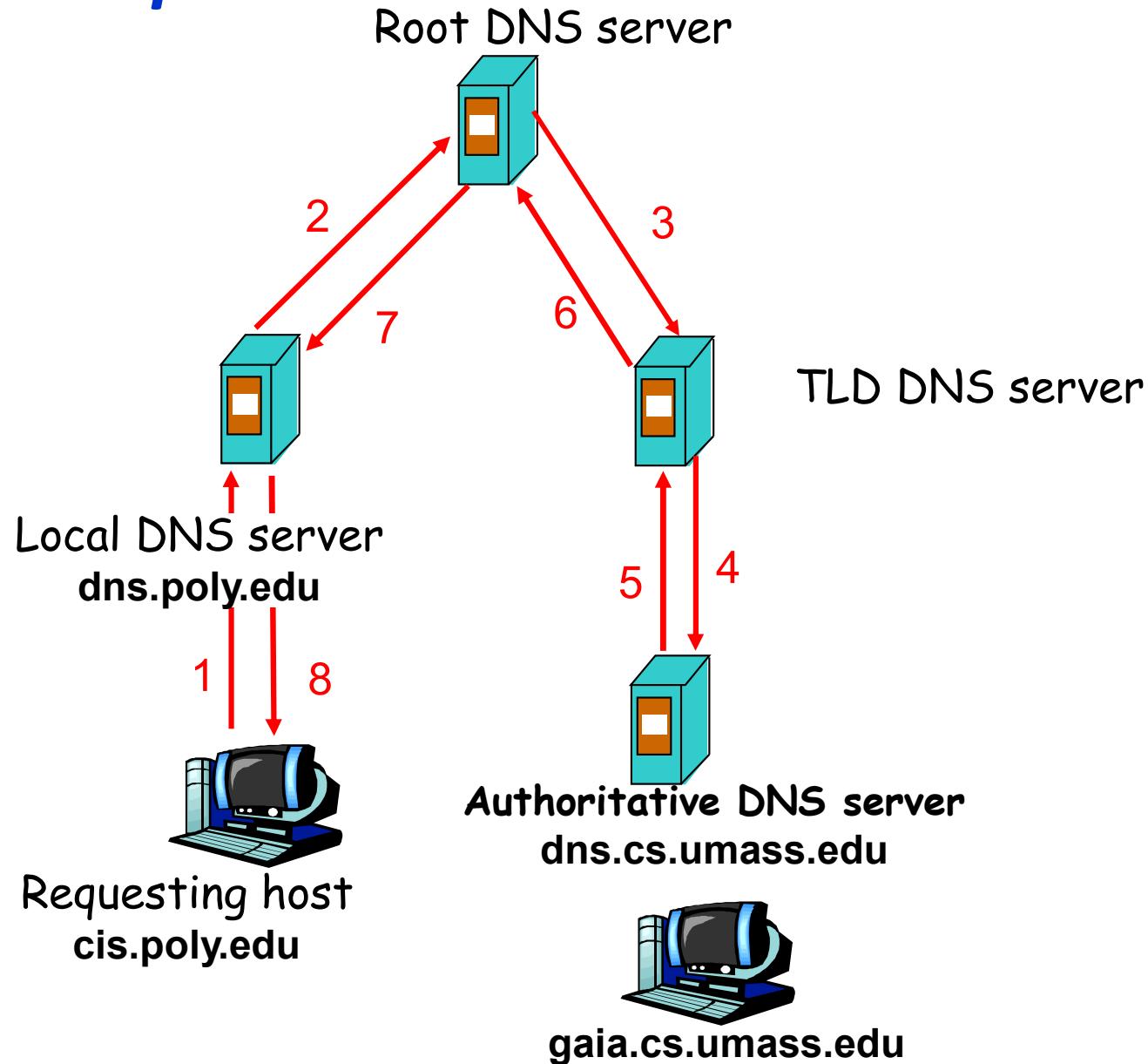
Test applet at:

[https://media.pearsoncmg.com/aw/ecs\\_kurose\\_compnetwork\\_7/cw/content/interactiveanimations/recursive-iterative-queries-in-dns/index.html](https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/recursive-iterative-queries-in-dns/index.html)

# DNS: Example

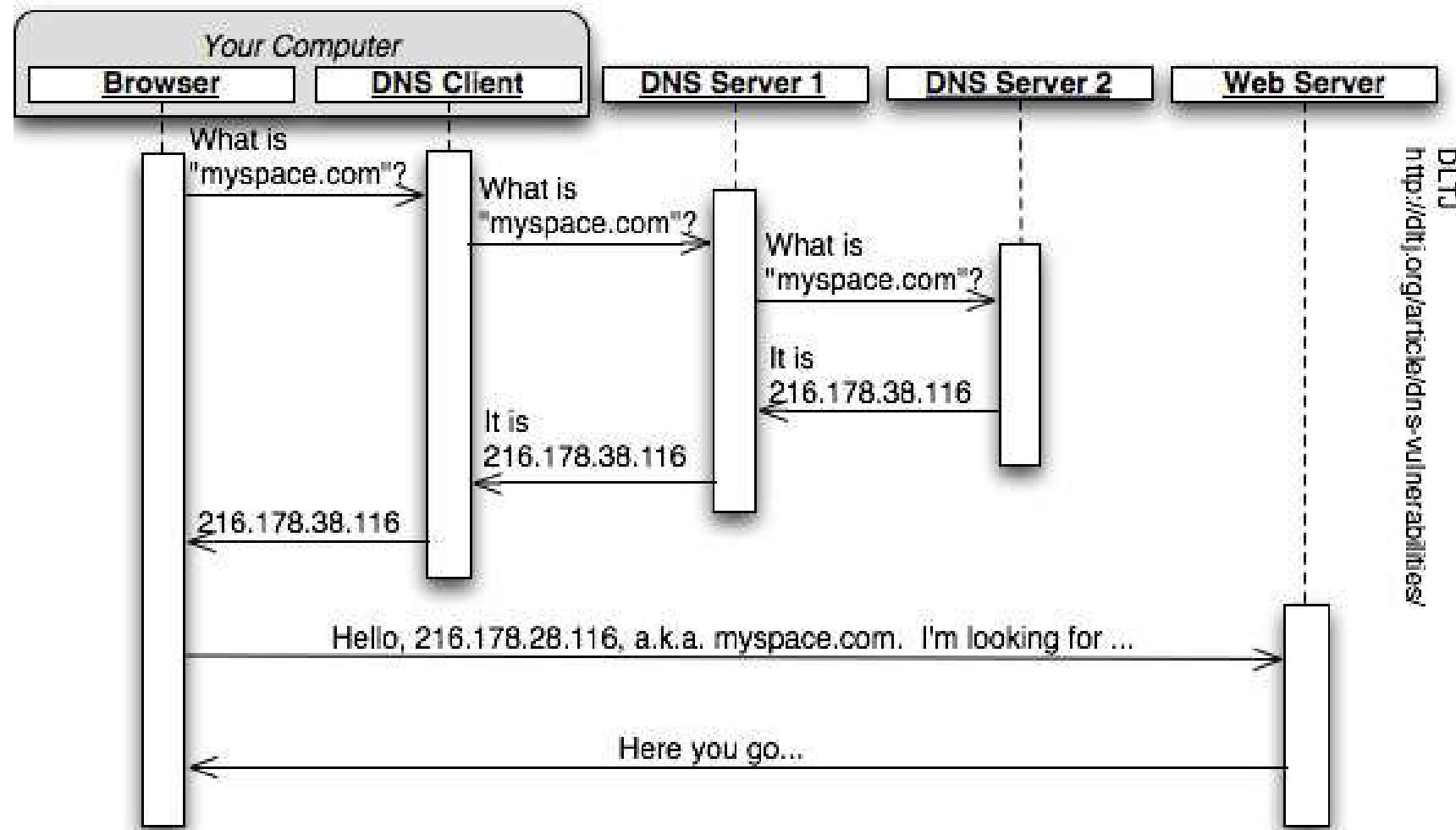
## Recursive query:

- Puts burden of name resolution on contacted name server
- Heavy load?

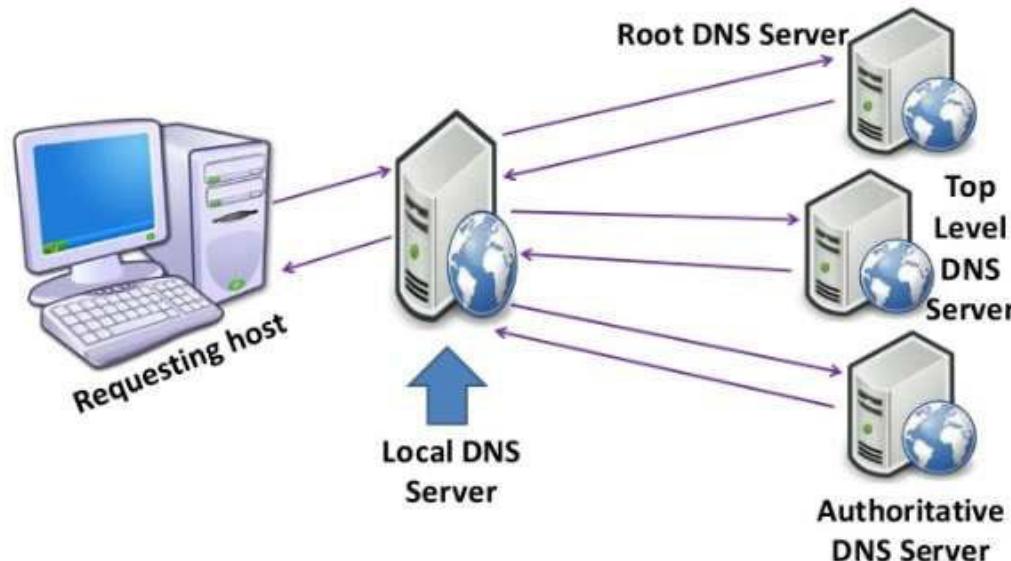


# DNS: Example

TPC: Prob. 4



# DNS Caching



- Once (any) name server learns a mapping, it *caches* that mapping:
  - Cache entries timeout (disappear) after some time (typically 2 days);
  - TLD servers are typically cached in local name servers:
    - Thus, *root name servers are not so often visited.*

# 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 the canonical name
- Type=MX
  - value is name of a mailserver associated with name

## *nslookup Test*

```
>nslookup www.google.pt
```

```
Server: 193.136.128.1
```

```
Address: 193.136.128.1#53
```

```
Non-authoritative answer:
```

```
Name: www.google.pt
```

```
Address: 173.194.45.31
```

```
Name: www.google.pt
```

```
Address: 173.194.45.23
```

```
Name: www.google.pt
```

```
Address: 173.194.45.24
```

# Using "dig" – DNS Lookup Utility

>dig www.google.pt

; ; ANSWER SECTION:

www.google.pt.	269019	IN	<b>CNAME</b>	www.google.com.
www.google.com.	603749	IN	<b>CNAME</b>	www.l.google.com.
www.l.google.com.	3	IN	<b>A</b>	74.125.39.103
www.l.google.com.	3	IN	<b>A</b>	74.125.39.104
www.l.google.com.	3	IN	<b>A</b>	74.125.39.147
www.l.google.com.	3	IN	<b>A</b>	74.125.39.99

; ; AUTHORITY SECTION:

l.google.com.	86391	IN	<b>NS</b>	f.l.google.com.
l.google.com.	86391	IN	<b>NS</b>	g.l.google.com.
l.google.com.	86391	IN	<b>NS</b>	a.l.google.com.
l.google.com.	86391	IN	<b>NS</b>	b.l.google.com.
l.google.com.	86391	IN	<b>NS</b>	c.l.google.com.
l.google.com.	86391	IN	<b>NS</b>	d.l.google.com.
l.google.com.	86391	IN	<b>NS</b>	e.l.google.com.

; ; ADDITIONAL SECTION:

a.l.google.com.	171858	IN	<b>A</b>	209.85.139.9
b.l.google.com.	12148	IN	<b>A</b>	74.125.45.9
c.l.google.com.	9179	IN	<b>A</b>	64.233.161.9
d.l.google.com.	17235	IN	<b>A</b>	74.125.77.9
e.l.google.com.	80225	IN	<b>A</b>	209.85.137.9
f.l.google.com.	18235	IN	<b>A</b>	72.14.235.9
g.l.google.com.	86333	IN	<b>A</b>	74.125.95.9

# Using “dig” – DNS Lookup Utility

```
>dig www.tecnico.ulisboa.pt
```

; ; QUESTION SECTION:

```
;www.tecnico.ulisboa.pt. IN A
```

; ; ANSWER SECTION:

```
www.tecnico.ulisboa.pt. 3600 IN A 193.136.128.169
```

; ; AUTHORITY SECTION:

```
tecnico.ulisboa.pt. 3600 IN NS a.ul.pt.
```

```
tecnico.ulisboa.pt. 3600 IN NS ns1.tecnico.ulisboa.pt.
```

```
tecnico.ulisboa.pt. 3600 IN NS ns2.tecnico.ulisboa.pt.
```

; ; ADDITIONAL SECTION:

```
a.ul.pt. 115 IN A 194.117.0.150
```

```
ns1.tecnico.ulisboa.pt. 3600 IN A 193.136.128.1
```

```
ns2.tecnico.ulisboa.pt. 3600 IN A 193.136.128.2
```

```
a.ul.pt. 115 IN AAAA 2001:690:21c0:a::150
```

```
ns1.tecnico.ulisboa.pt. 3600 IN AAAA 2001:690:2100:1::53:1
```

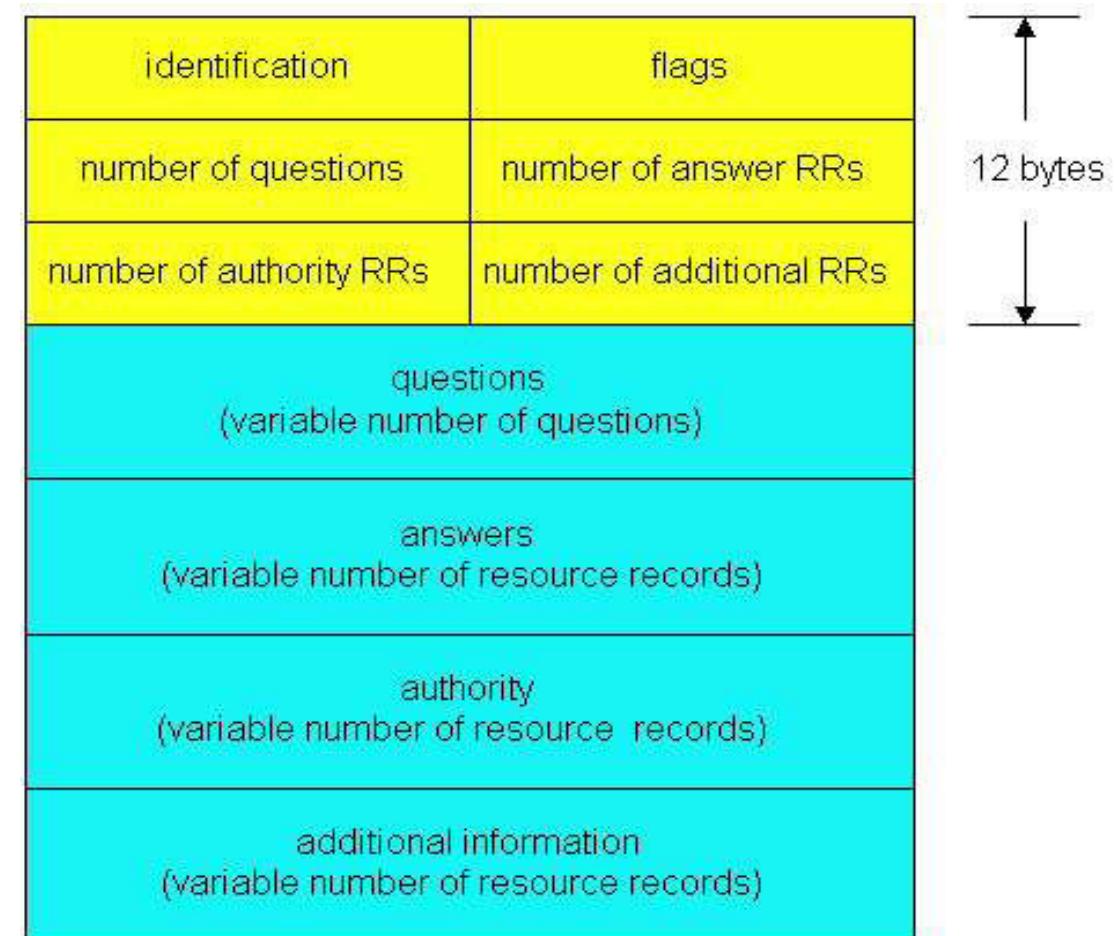
```
ns2.tecnico.ulisboa.pt. 3600 IN AAAA 2001:690:2100:1::2
```

# DNS Protocol, Messages

DNS protocol: *query* and *reply* messages, both with same message format

Message header:

- **Identification**: 16 bit number identifying the query; the reply to this query uses the same number;
- **Flags**:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



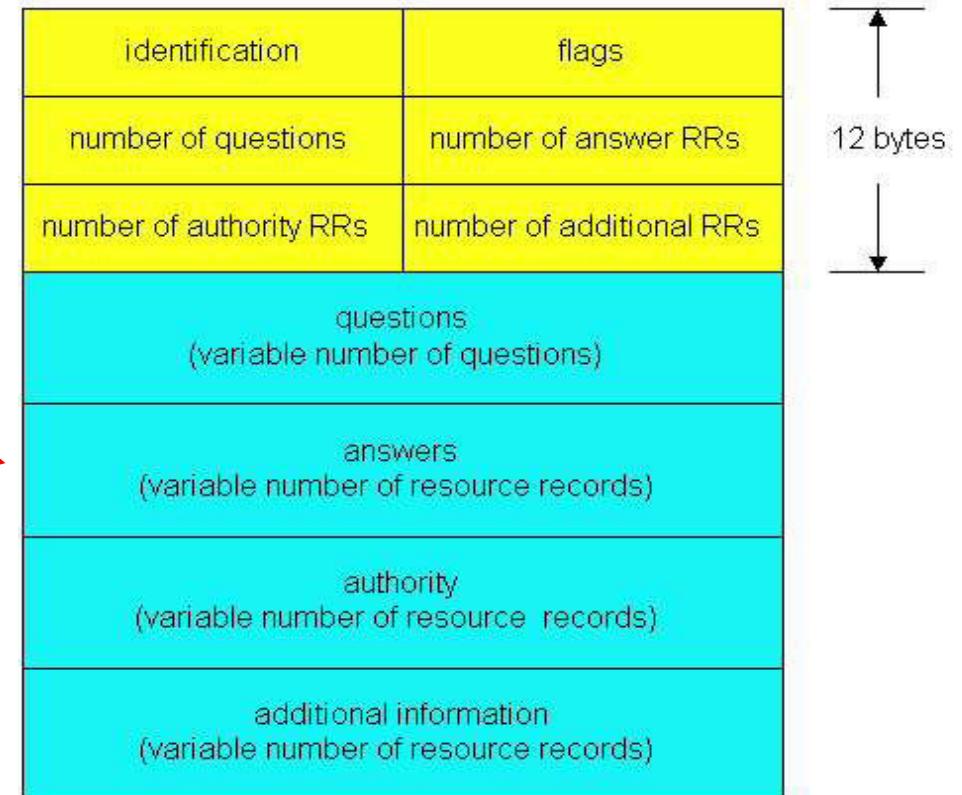
# DNS Protocol, Messages

Name and type fields  
for a query

RRs in response  
to query

records for  
authoritative servers

additional “helpful”  
info that may be used



# DNS Protocol

 dns.cap - Wireshark

9 92.189905	192.168.170.8	192.168.170.20	DNS	Standard query A www.netbsd.org
10 92.238816	192.168.170.20	192.168.170.8	DNS	Standard query response A 204.152.190.12

[-] Domain Name System (query)

[Response In: 10]

Transaction ID: 0x75c0

[+] Flags: 0x0100 (standard query)

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

[-] Queries

[+] www.netbsd.org: type A, class IN

Domain Name System (response)

[Request In: 9]

[Time: 0.048911000 seconds]

Transaction ID: 0x75c0

[+] Flags: 0x8180 (Standard query response, No error)

Questions: 1

Answer RRs: 1

Authority RRs: 0

Additional RRs: 0

[-] Queries

[+] www.netbsd.org: type A, class IN

[-] Answers

[+] www.netbsd.org: type A, class IN, addr 204.152.190.12

Name: www.netbsd.org

Type: A (Host address)

Class: IN (0x0001)

Time to live: 22 hours, 49 minutes, 19 seconds

Data length: 4

Addr: 204.152.190.12

# Inserting Records into DNS

- Example: new startup “Network Utopia”
- Register name networkuptopia.com at *DNS registrar* :
  - Provide names and IP addresses of authoritative name servers (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.networkuptopia.com;  
Type MX record for networkutopia.com

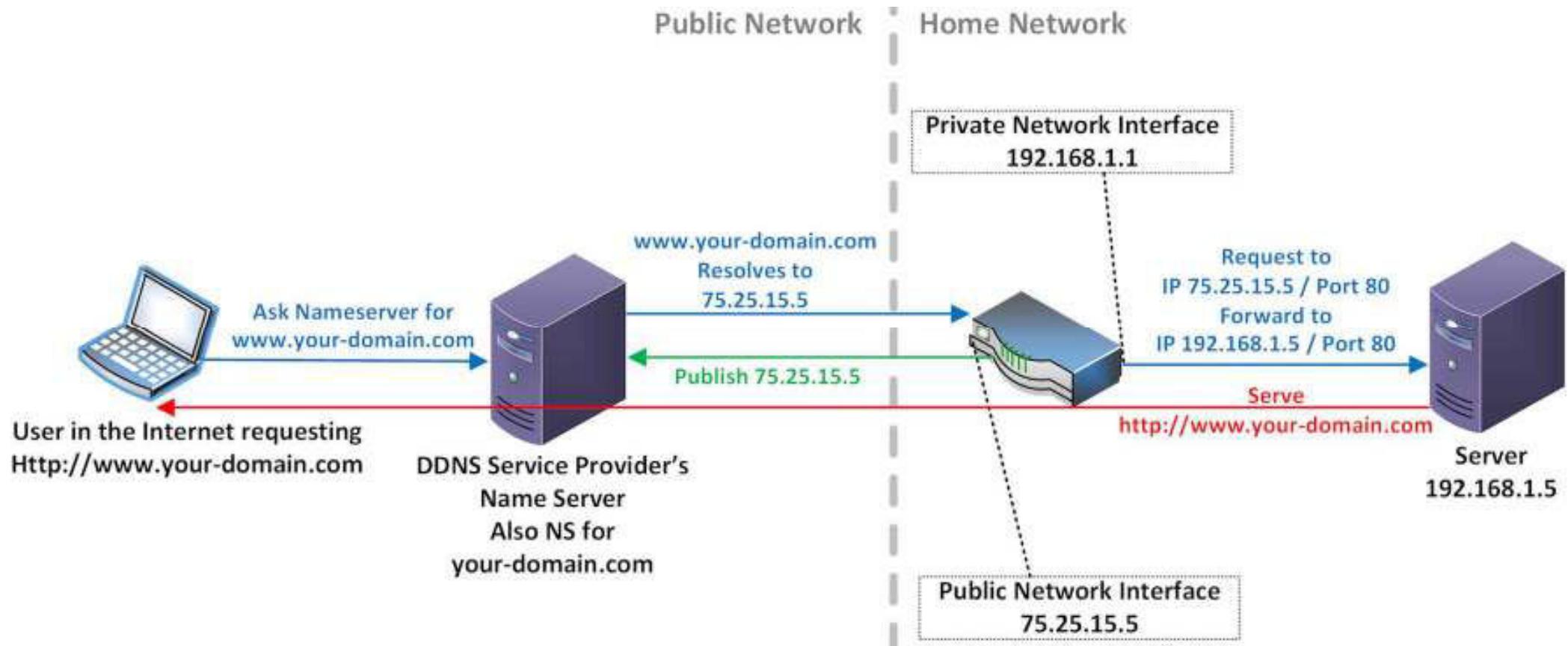
## Dynamic DNS

- DDNS allows to update the IP address of a domain in real time (instead of in a few days);
- Domain name can be assigned to a computer with a varying IP address;

Dynamic DNS is a method that allows you to **notify a Domain Name Server (DNS) to change your active DNS configuration on a device** such as a router or computer of its configured hostname and address. It is most useful when your computer or network obtains a new IP address lease and you would like to dynamically associate a hostname with that address, without having to manually enter the change every time. Since there are situations where an IP address can change, it helps to have a way of ***automatically updating hostnames that point to the new address every time.***

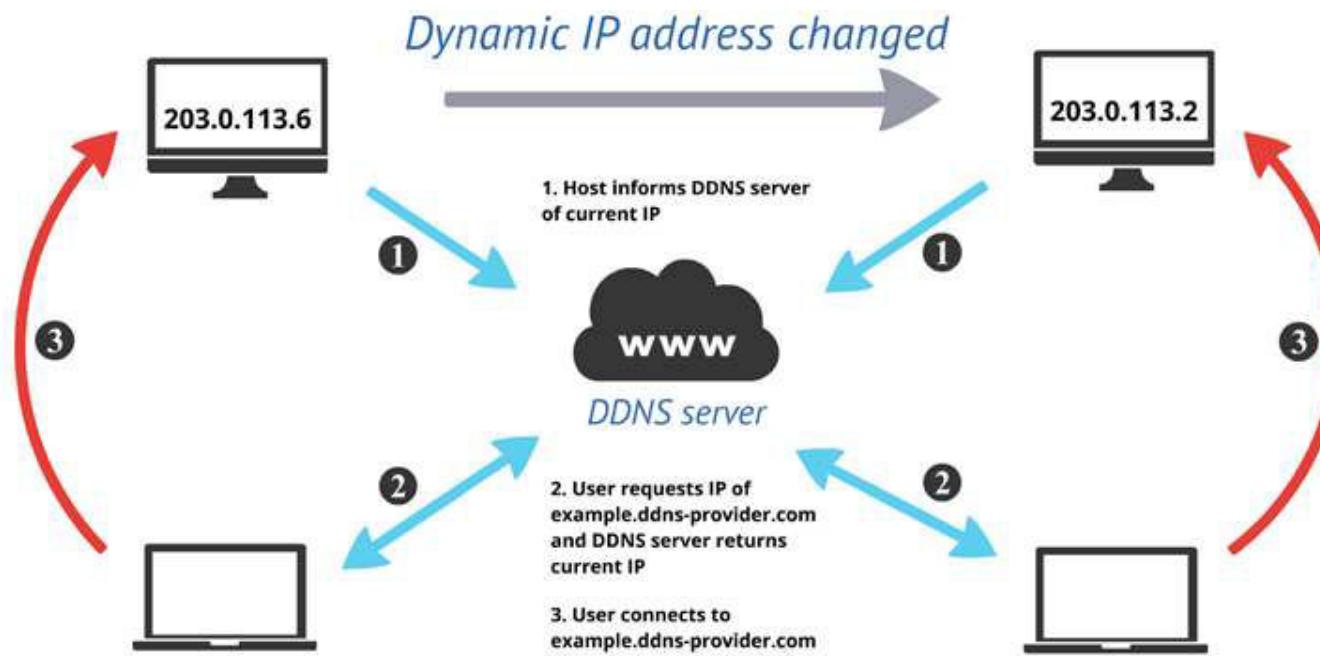
# Dynamic DNS

- DDNS allows to update the IP address of a domain in real time (instead of in a few days);
- Domain name can be assigned to a computer with a varying IP address;



# Dynamic DNS

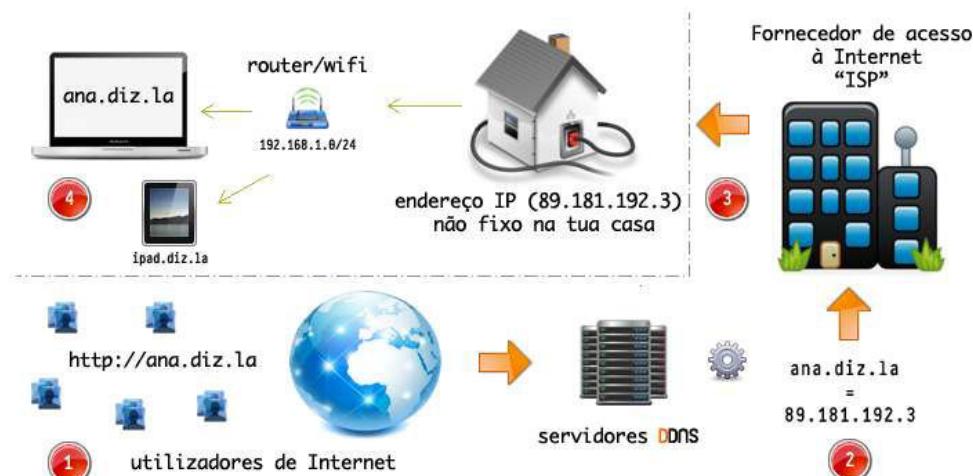
- Other sites on the Internet can establish connections to a machine, without needing to track the IP address themselves;
- It makes servers with dynamic IP addresses accessible on the Internet;
- Allows a domain name to point to a PC whose IP address changes;
- Allows to run servers at home – Internet, Email, RC project, ...



# DDNS: How to Use

Dynamic NS services can be found in many places online, for varying costs, but they all work more or less the same way:

- Sign up for a DDNS account and pick a hostname (check: [dnslookup.me/dynamic-dns/](http://dnslookup.me/dynamic-dns/));
  - Signing up will get you a simple hostname, and by configuring your router, your ISP-assigned and ever-changing IP will be updated automatically.
- Enter DDNS registration information in the router or use a DDNS client software, and setup the router and Web service to use the DDNS configuration
  - Enable DDNS and specify the hostname you created;



# DNS security

## DDoS attacks

(DNS Denial of Service)

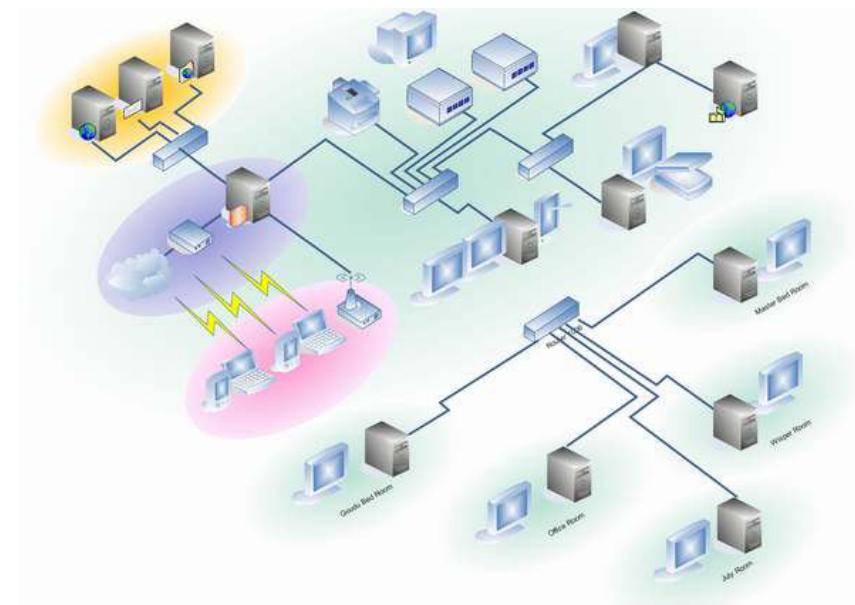
- Bombard root servers with traffic
  - not successful to date
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - potentially more dangerous

## Spoofing attacks

- Intercept DNS queries, returning bogus replies
  - DNS cache poisoning
  - RFC 4033: DNSSEC authentication services

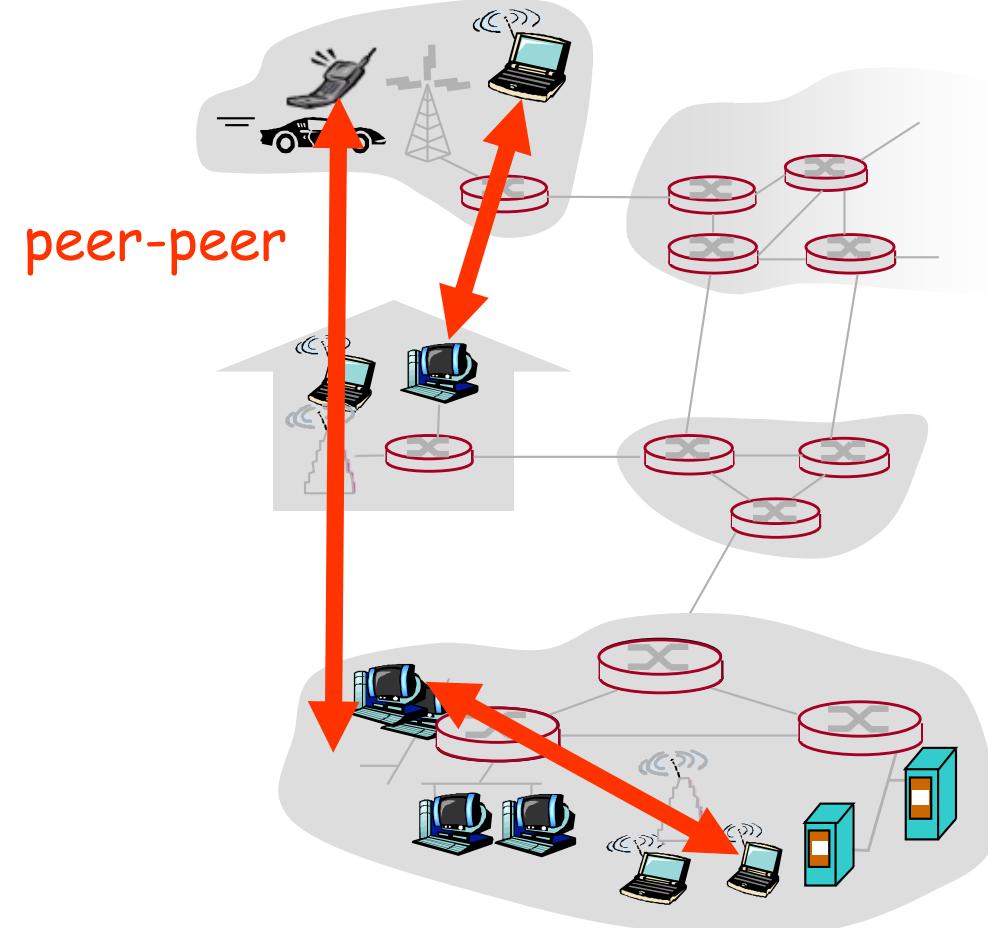
# Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- **P2P Applications**



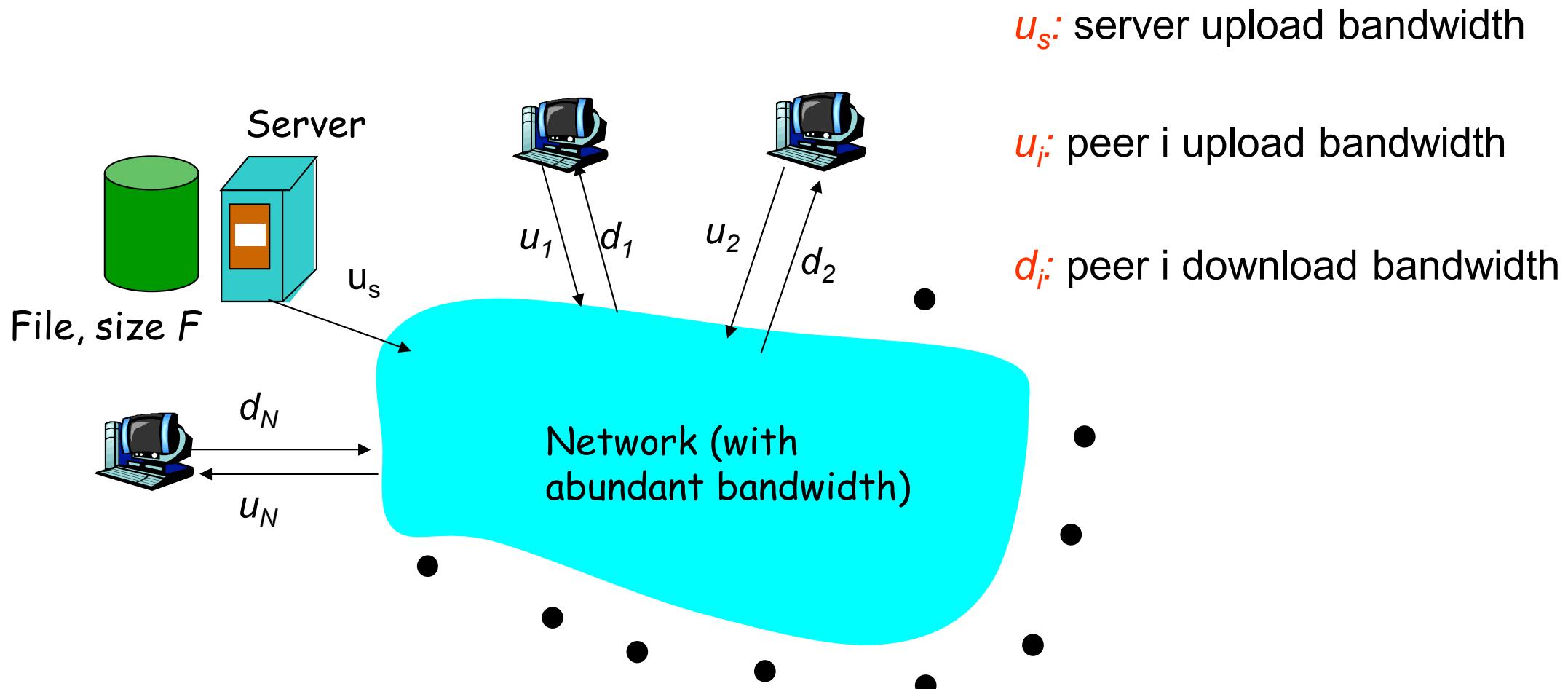
# Pure P2P Architecture

- There is *not* an always-on server;
- Arbitrary end systems communicate directly;
- Peers are intermittently connected and can change IP addresses.
  
- Three examples:
  - File distribution;
  - Searching for information;
  - Skype.



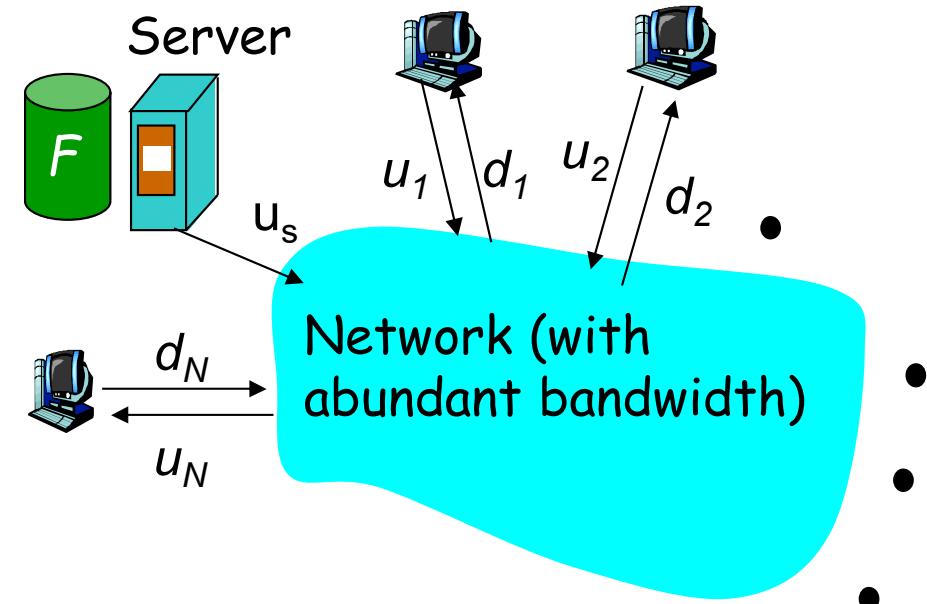
# File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to  $N$  peers?



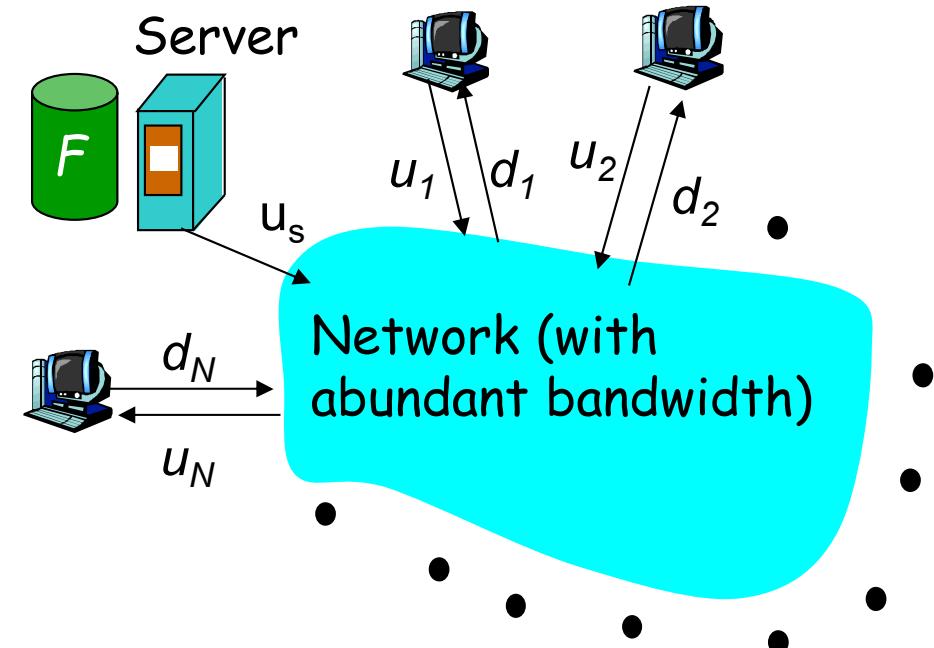
# File Distribution Time: Server-Client

Client-Server



## File Distribution Time: Server-Client

- Server sends  $N$  copies:
  - $NF/u_s$  time
- Client  $i$  takes  $F/d_i$  time to download;



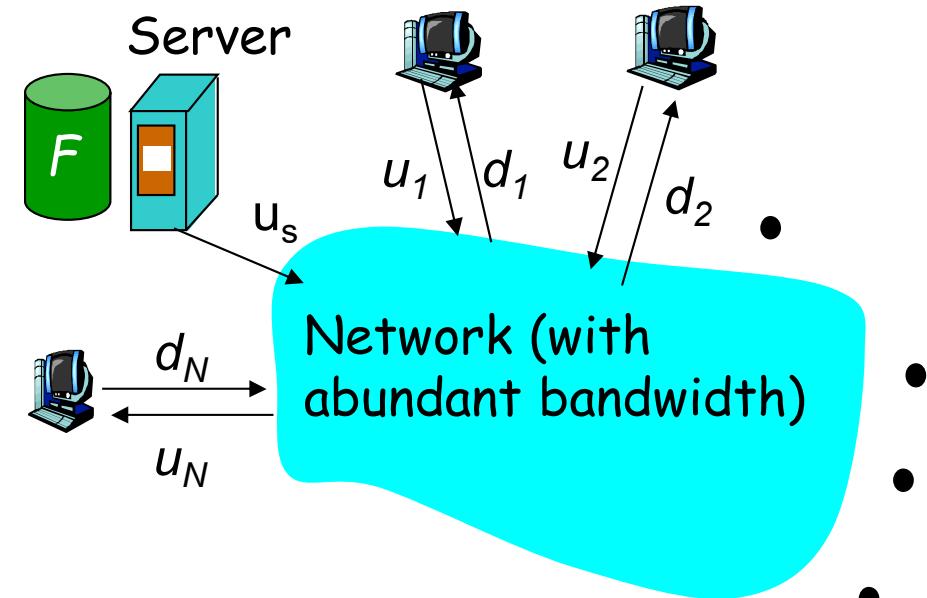
Time to distribute  $F$   
 to  $N$  clients using  
 client/server approach

$t_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in  $N$   
 (for large  $N$ )

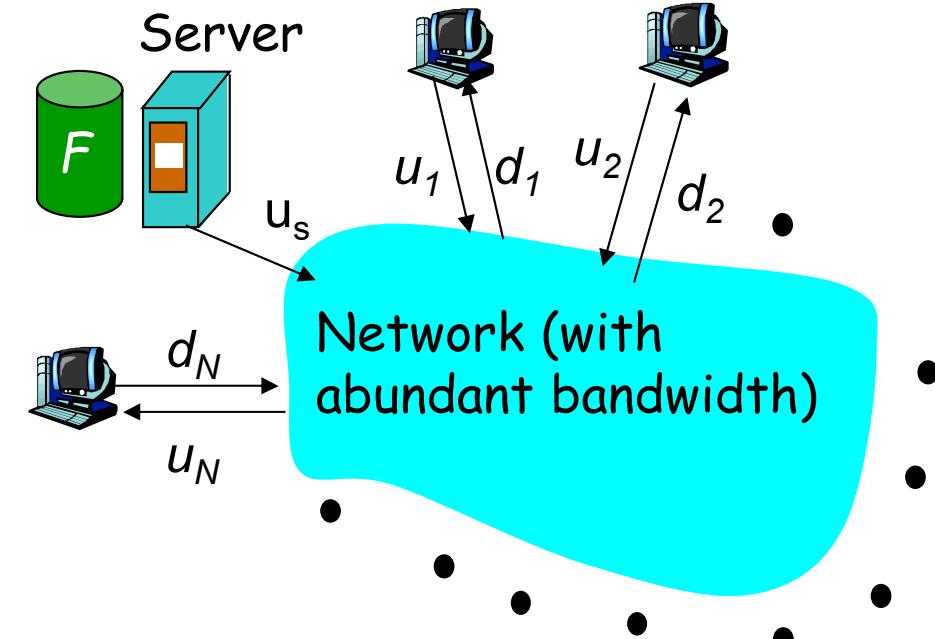
Peer to peer

## File Distribution Time: P2P



## File Distribution Time: P2P

- Server must send one copy:
  - $F/u_s$  time
- Client  $i$  takes  $F/d_i$  time to download;
- $NF$  bits must be downloaded (aggregate);
- Fastest possible upload rate:  $u_s + \sum u_i$

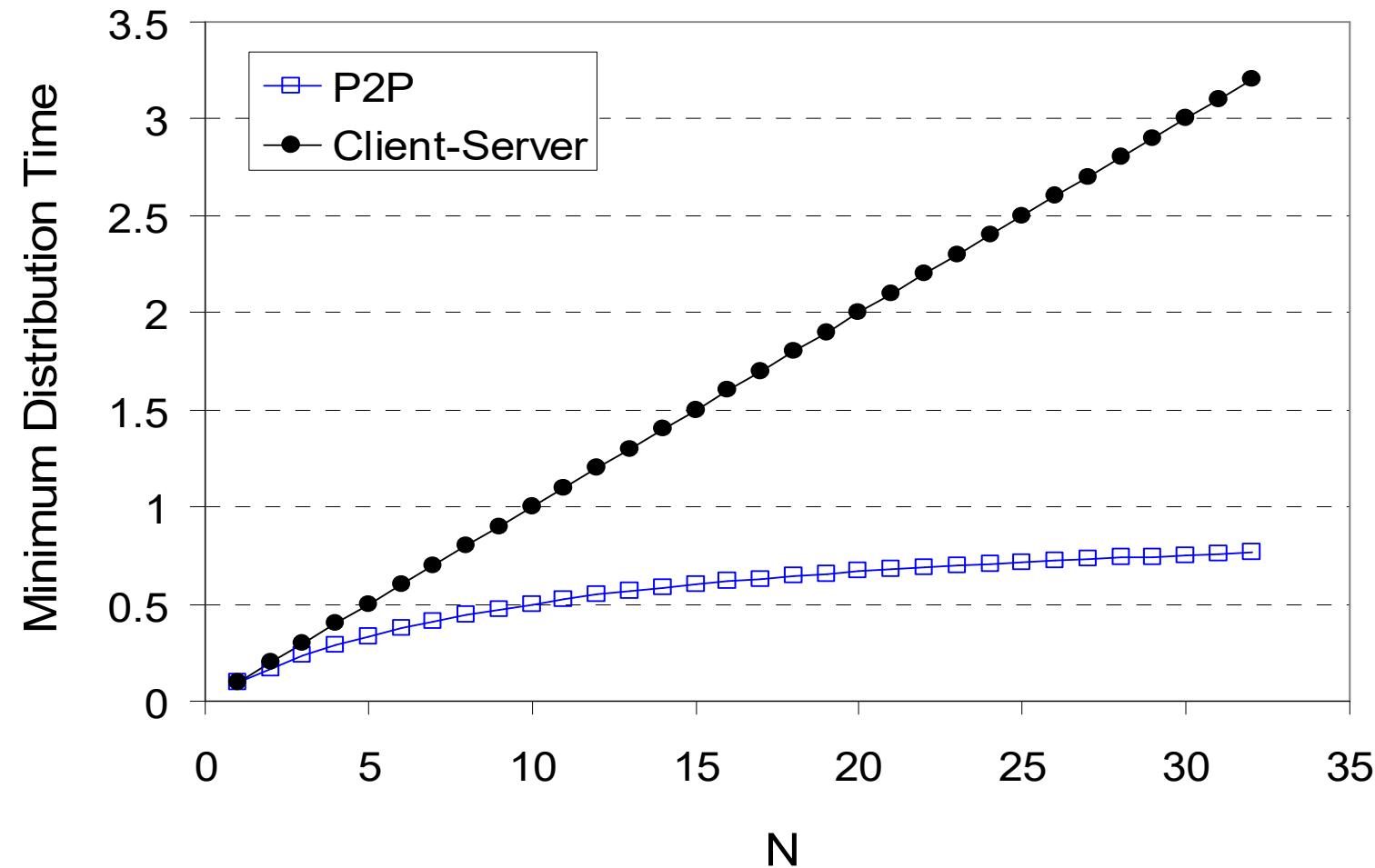


$$t_{P2P} = \max \{ F/u_s, F/\min(d_i), NF/(u_s + \sum u_i) \}$$

# Server-Client vs. P2P: Example

TPC: Prob. 5

Client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$ ,  $d_{\min} \geq u_s$



# *Summary*

- Application architectures:
  - Client-server;
  - P2P;
  - Hybrid.
- Application service requirements:
  - Reliability, bandwidth, delay.
- Internet transport service model
  - Connection-oriented, reliable: TCP;
  - Unreliable, datagrams: UDP.
- Specific protocols:
  - HTTP, FTP, SMTP, DNS;
  - P2P.

# Summary

## Learned about *protocols*

- Typical request/reply message exchange:
  - **Client** requests info or service;
  - **Server** responds with data, status code.
- Message formats:
  - **Headers**: fields giving information about data;
  - **Data**: info being communicated.

## *Important issues:*

- Control vs. data messages: in-band, out-of-band;
- Centralized vs. decentralized;
- Stateless vs. stateful;
- Reliable vs. unreliable message transfer.



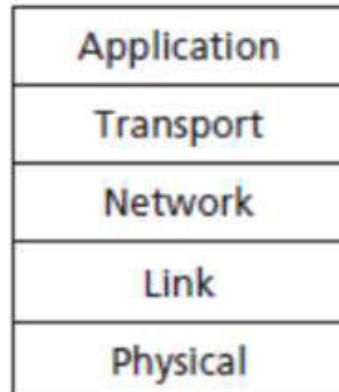
# Redes de Computadores

## LEIC-A

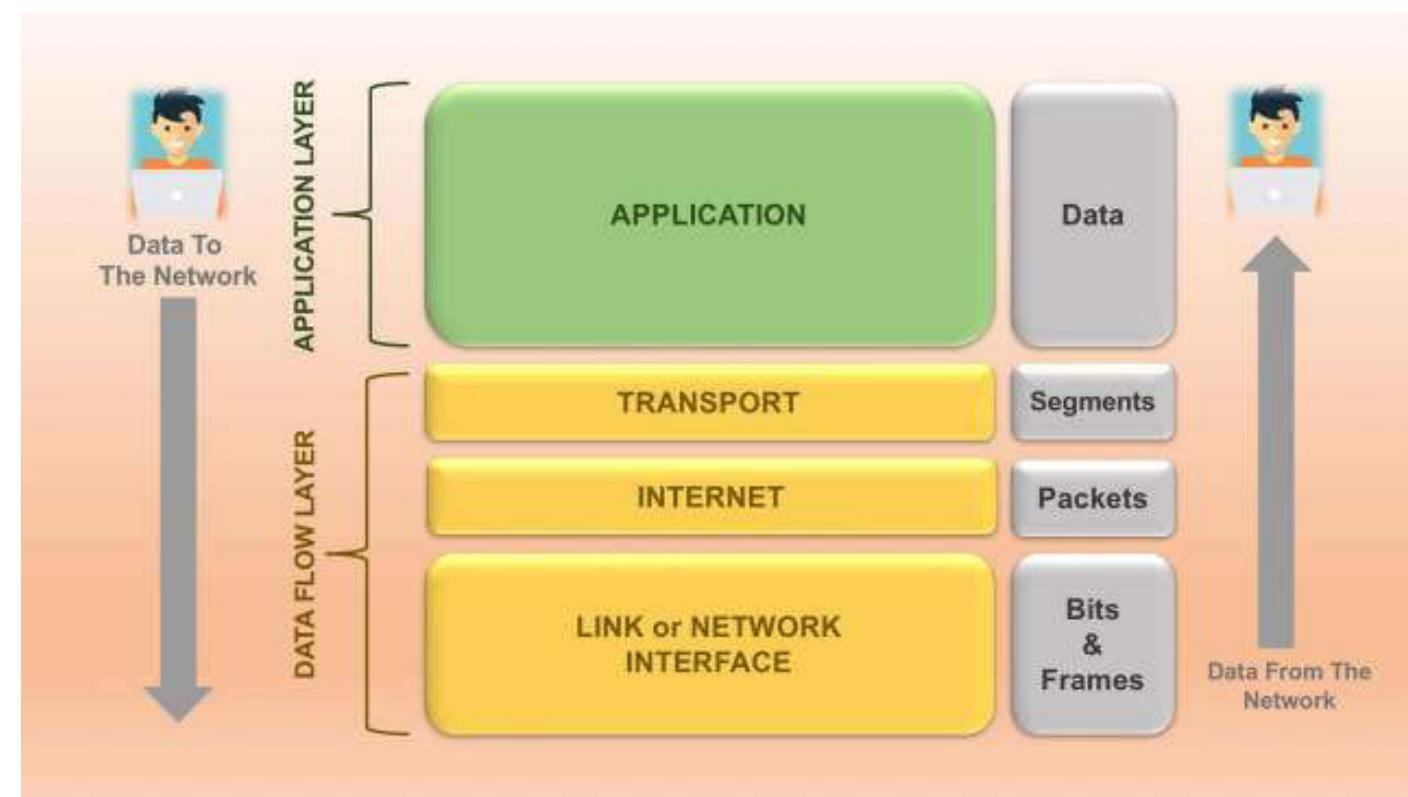
### 3 – *Transport Layer* *(part 1)*

Prof. Paulo Lobato Correia  
*IST, DEEC – Área Científica de Telecomunicações*

# Modelo TCP/IP



Five-layer  
Internet  
protocol stack

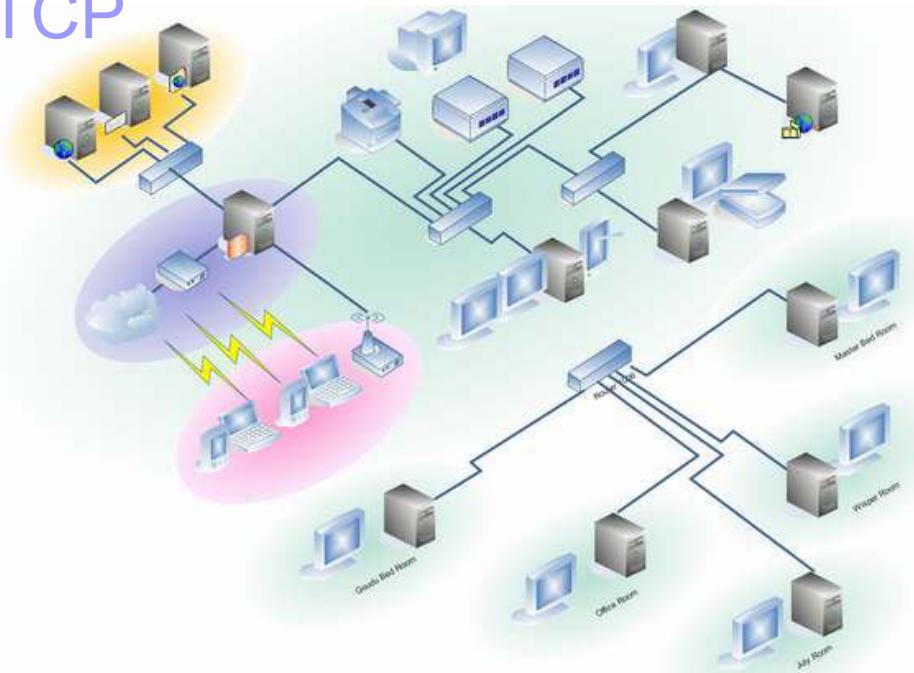


# Objectives

- Understand the principles behind transport layer services:
  - Multiplexing/demultiplexing;
  - Reliable data transfer;
  - Flow control;
  - Congestion control.
- Transport layer protocols in the Internet:
  - UDP: connectionless transport;
  - TCP: connection-oriented transport;
  - TCP congestion control.

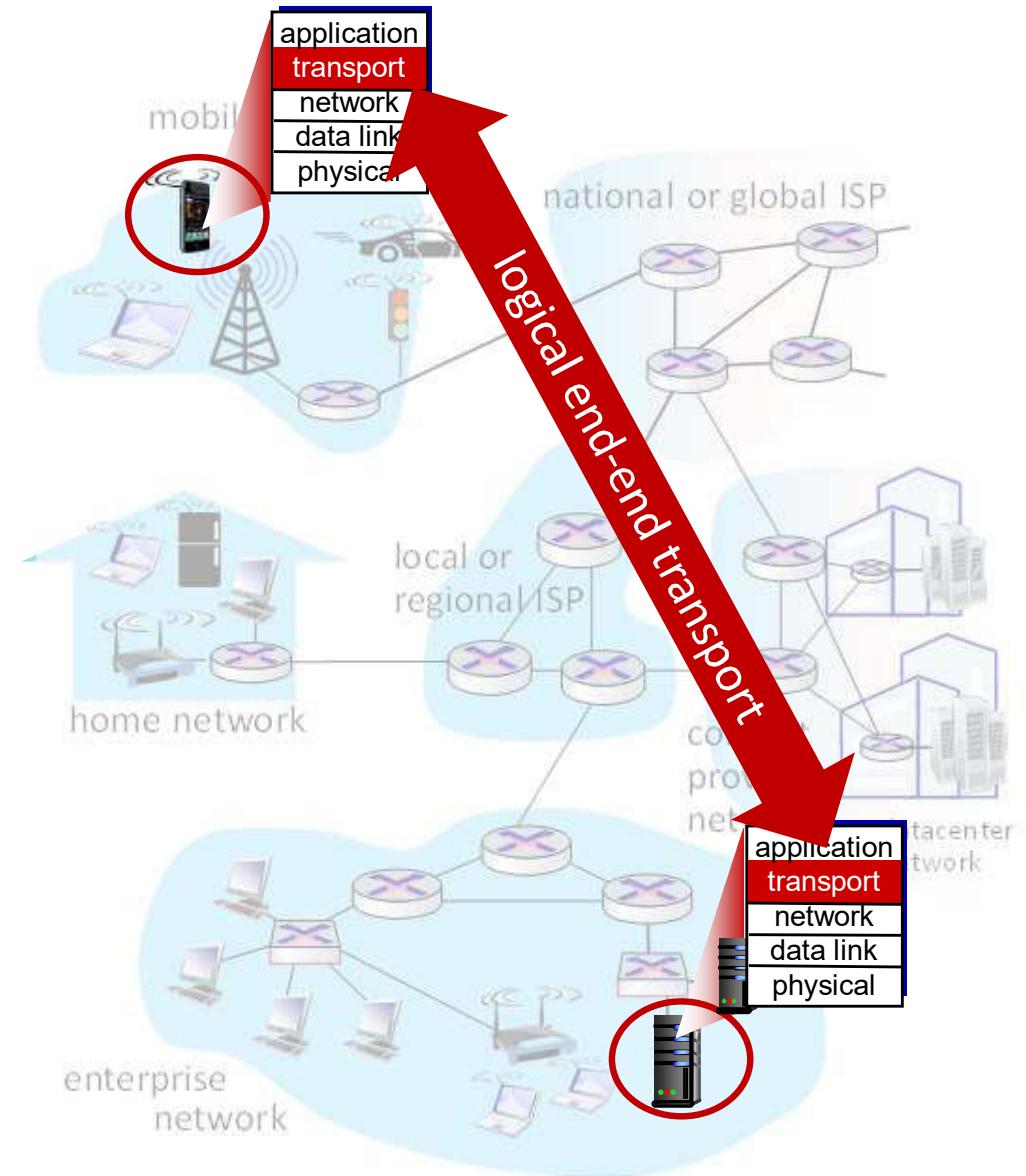
# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
  - Connection management;
- Principles of congestion control
- TCP congestion control



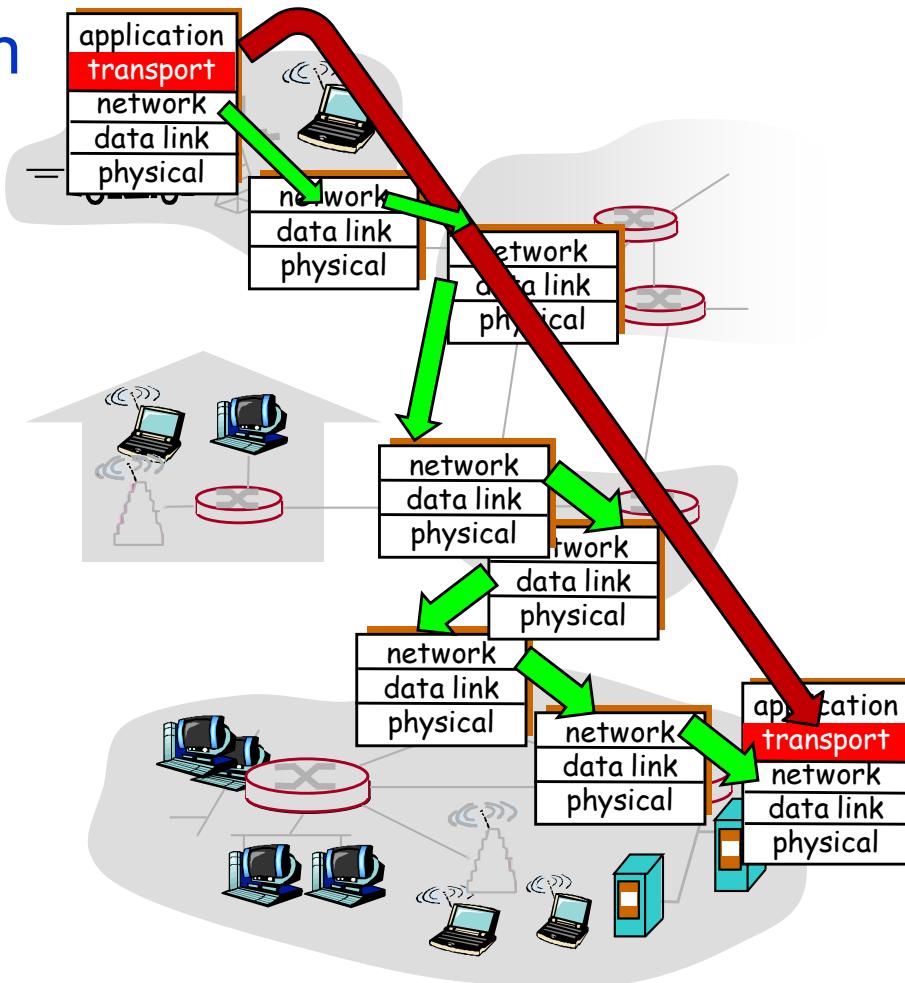
# Transport Services and Protocols

- Provide *logical communication* between application processes running on different hosts;
- Transport protocols run in **end systems**:
  - Sender side: **breaks application messages into segments**, passes them to the network layer;
  - Receiver side: **reassembles segments into messages**, passes them to the application layer;
- Two transport protocols available to Internet applications:
  - TCP and UDP.



# Transport vs. Network Layer

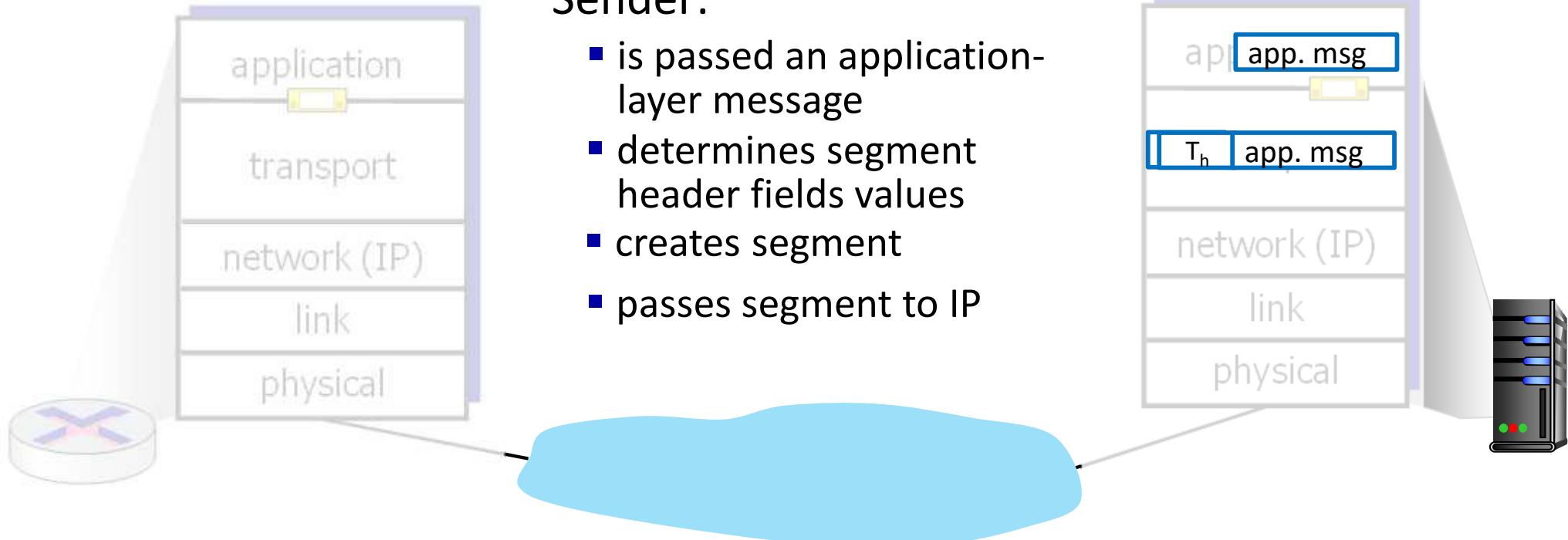
- **Transport layer:** logical communication between processes:
  - Relies on, and enhances, network layer services.
- **Network layer:** logical communication between hosts;



# Transport Layer Actions

Sender:

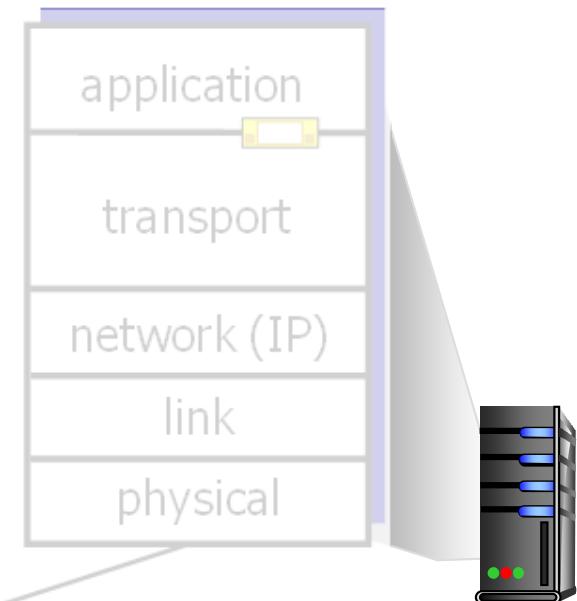
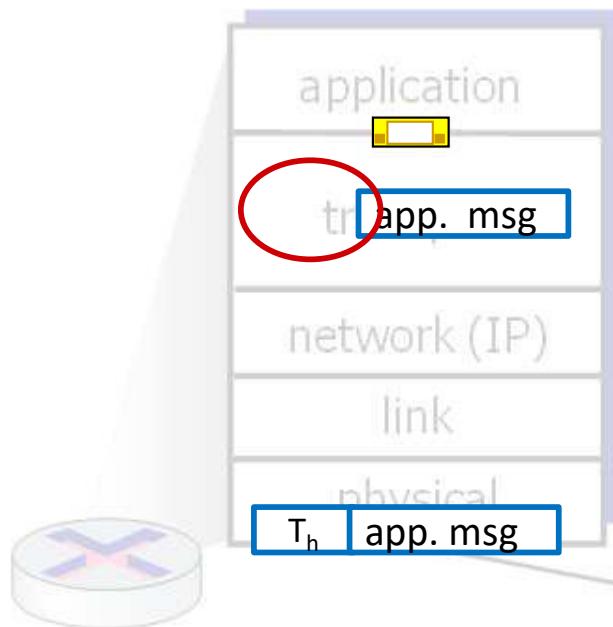
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP



# Transport Layer Actions

## Receiver:

- receives segment from IP
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket



# *Internet Transport Layer Protocols*

- Reliable, in-order delivery (TCP):

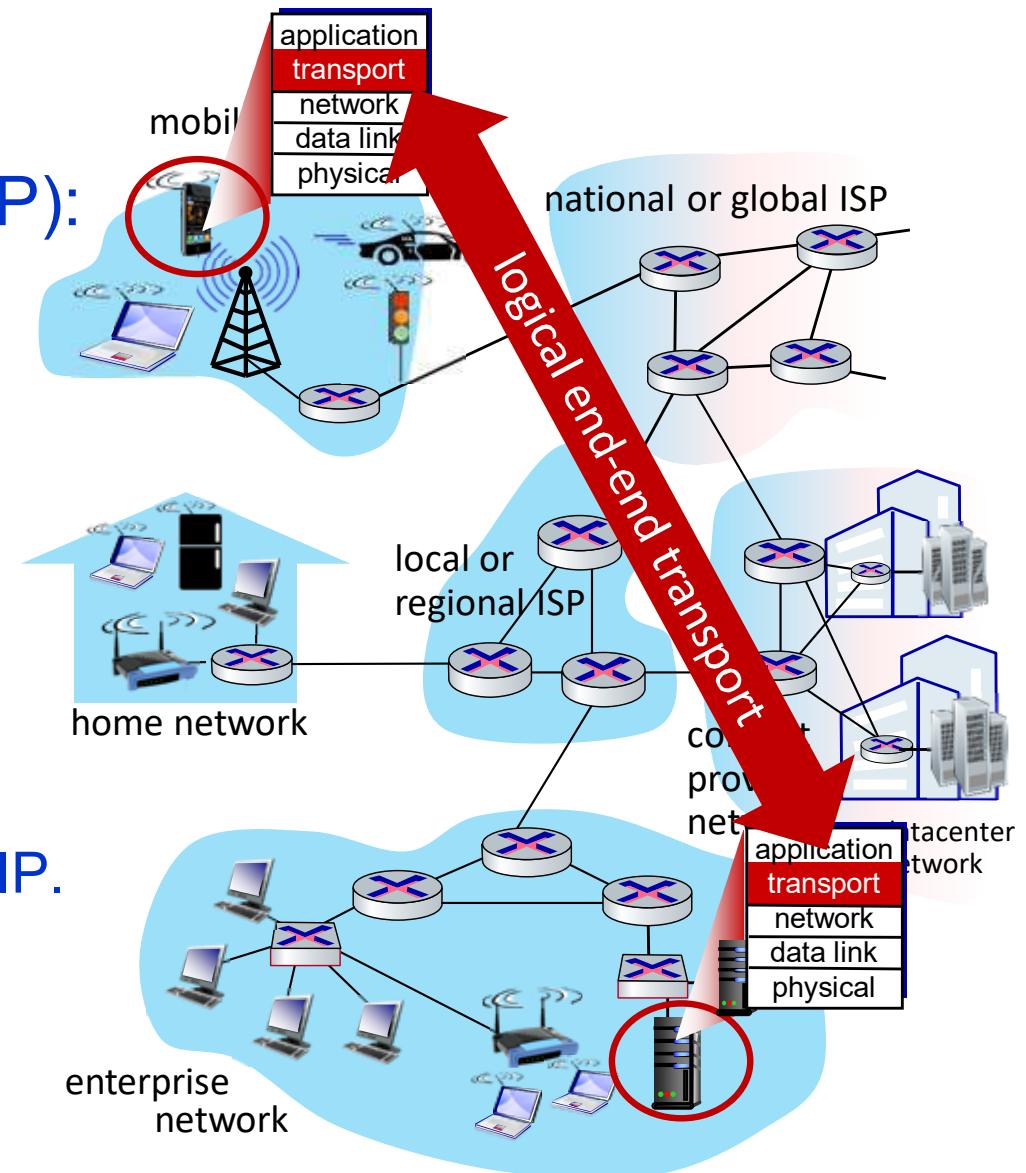
- Congestion control;
- Flow control;
- Connection setup.

- Unreliable, unordered delivery (UDP):

- Simple extension of “best-effort” IP.

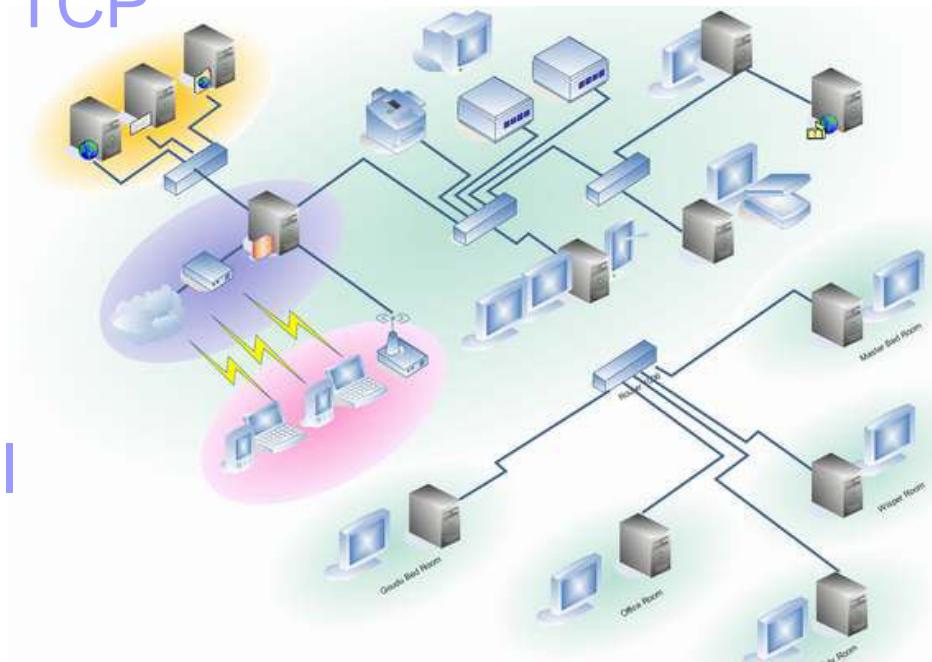
- Services not available:

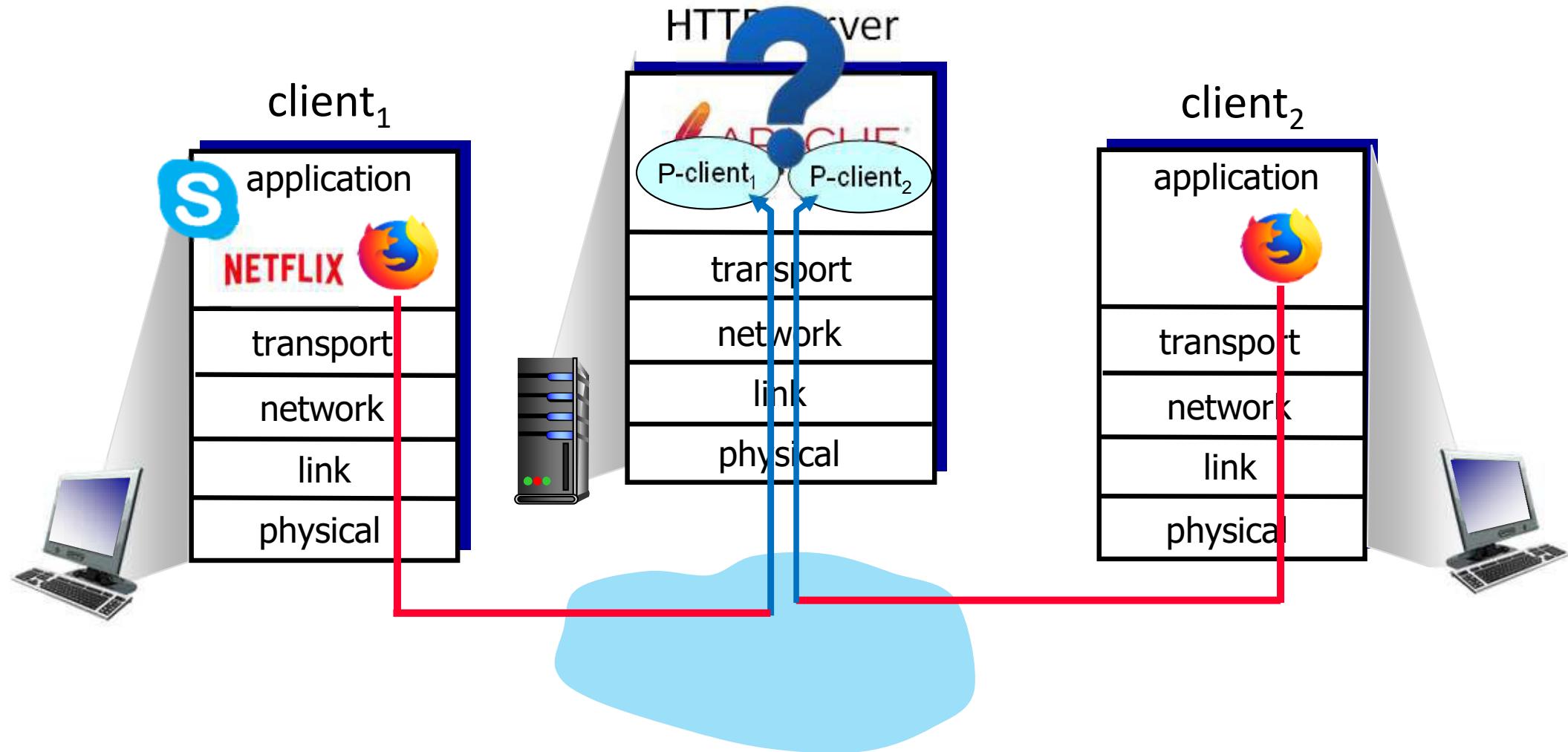
- Delay guarantees;
- Bandwidth guarantees.



# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
  - Connection management;
- Principles of congestion control
- TCP congestion control





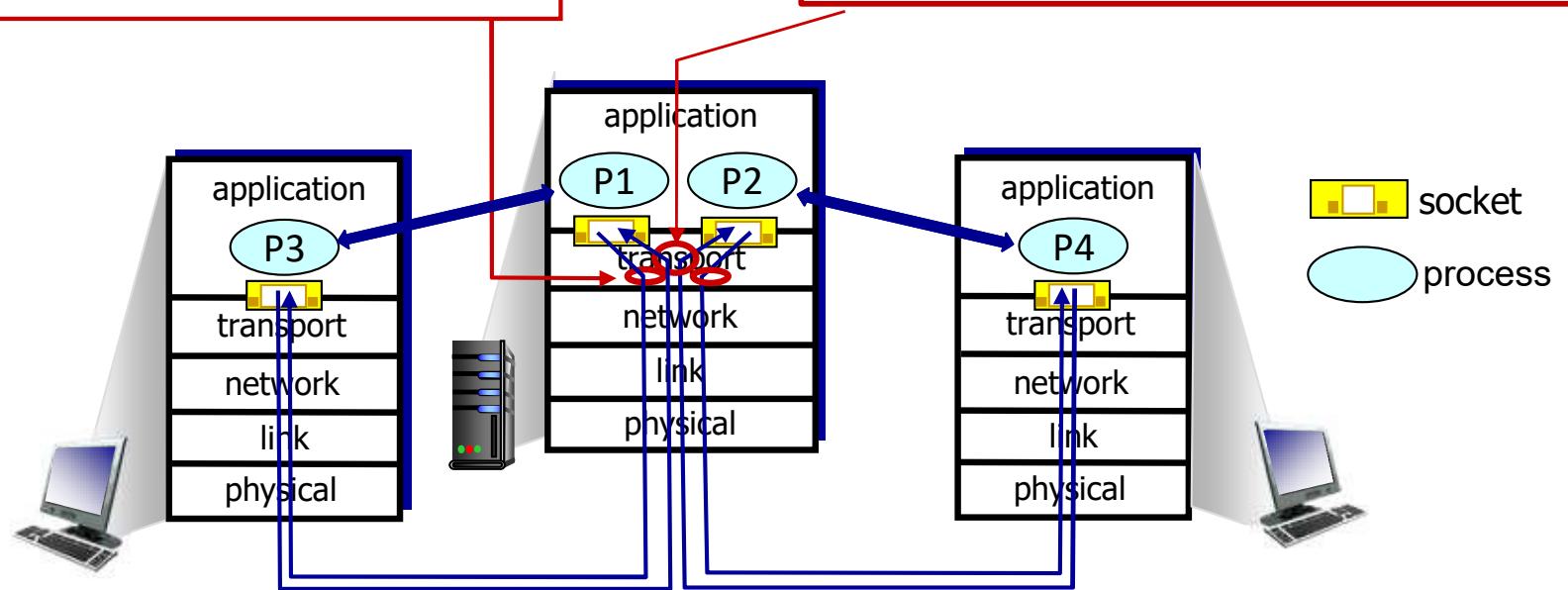
# Multiplexing / Demultiplexing

*multiplexing at sender:*

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

*demultiplexing at receiver:*

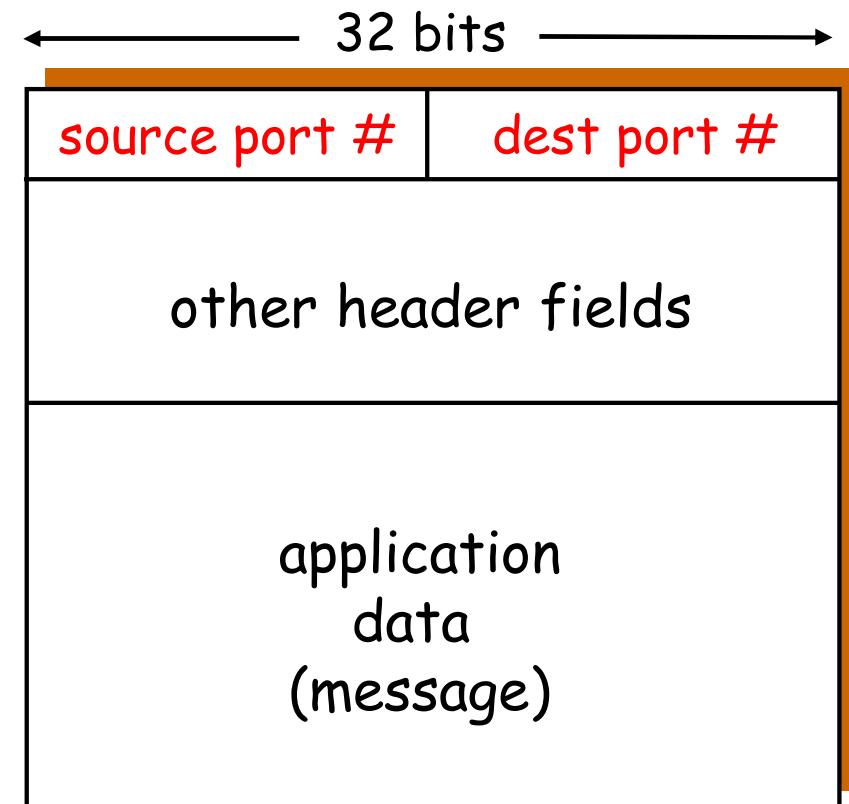
use header info to deliver received segments to correct socket



# Multiplexing / Demultiplexing

- Host receives IP datagrams:
  - Each datagram has source and destination IP addresses;
  - Each datagram carries 1 transport layer segment;
  - Each segment has **source and destination port numbers**.
- Host uses **IP address** and **port number** to direct segment to the appropriate socket.
- Well-known ports (0-1023):
  - HTTP: 80 (TCP)
  - DNS: 53 (UDP)

(<http://www.iana.org/assignments/port-numbers>)



TCP/UDP segment format

# UDP: Connectionless Multiplexing

- Create socket with port number:

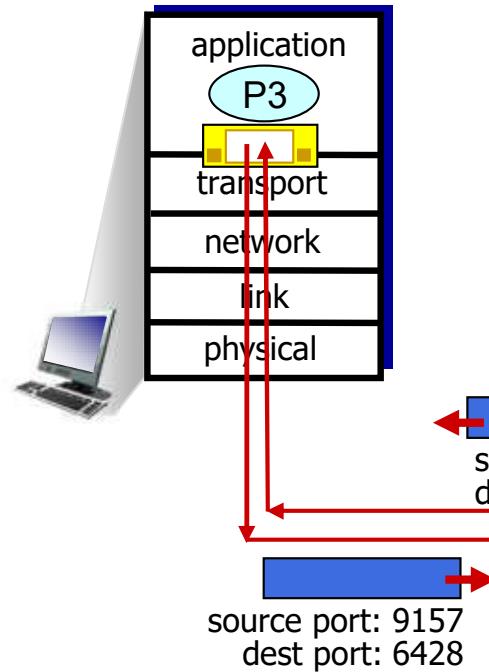
```
struct addrinfo hints,*res;  
  
hints.ai_family = AF_INET;          // IPv4  
hints.ai_socktype = SOCK_DGRAM;    // UDP socket  
hints.ai_flags = AI_PASSIVE|AI_NUMERICSERV;  
  
getaddrinfo(NULL, PORT, &hints, &res);  
  
MySocket = socket(res->ai_family, res->ai_socktype, res->ai_protocol);  
bind(fd, res->ai_addr, res->ai_addrlen);
```

- UDP sockets are identified by a 2-tuple:  
**(dest IP address, dest port number)**

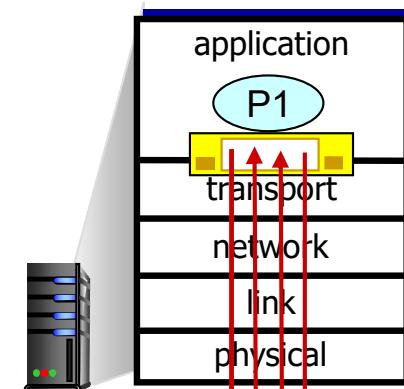
- When host receives UDP segment:
  - Check destination port number and direct UDP segment to corresponding socket.
- IP datagrams with different source IP addresses and/or different source port numbers *can be directed to the same socket.*

# Connectionless Demultiplexing: Example

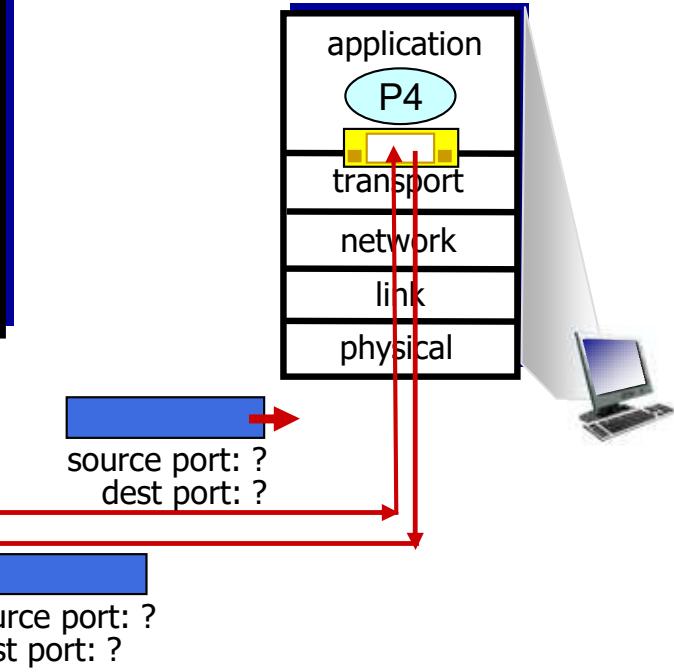
```
DatagramSocket mySocket2  
= new DatagramSocket  
(9157);
```



```
DatagramSocket  
serverSocket = new  
DatagramSocket  
(6428);
```



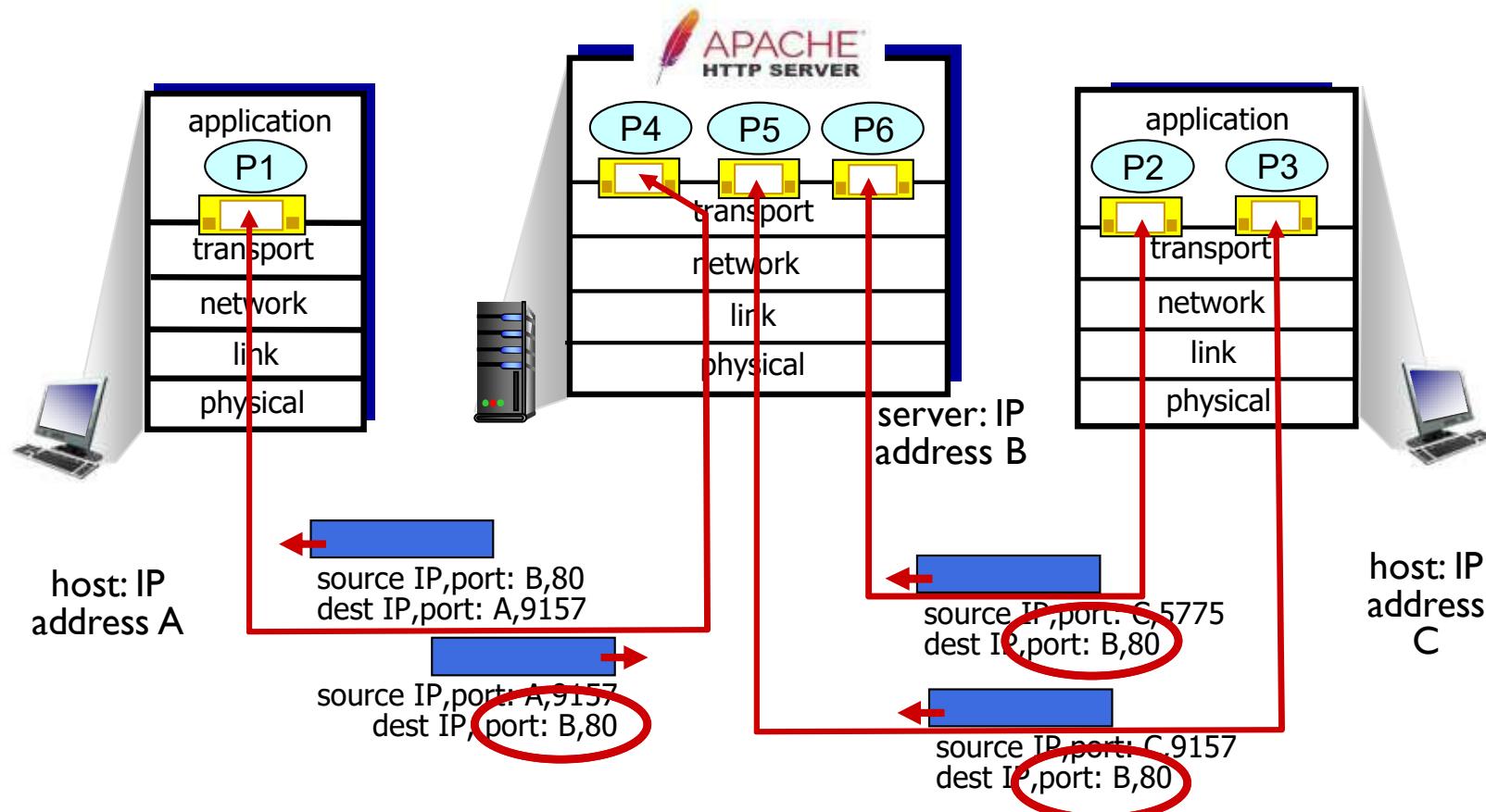
```
DatagramSocket mySocket1 =  
new DatagramSocket (5775);
```



# TCP: Connection-oriented Demultiplexing

- **TCP socket identified by 4-tuple:**
  - Source IP address;
  - Source port number;
  - Destination IP address;
  - Destination port number.
- Receiving host uses all four values to direct a segment to the appropriate socket;
- Server host may support many TCP sockets simultaneously:
  - Each socket identified by its own 4-tuple;
- **Web servers have different sockets for each connecting client:**
  - Non-persistent HTTP will have a different socket for each request.

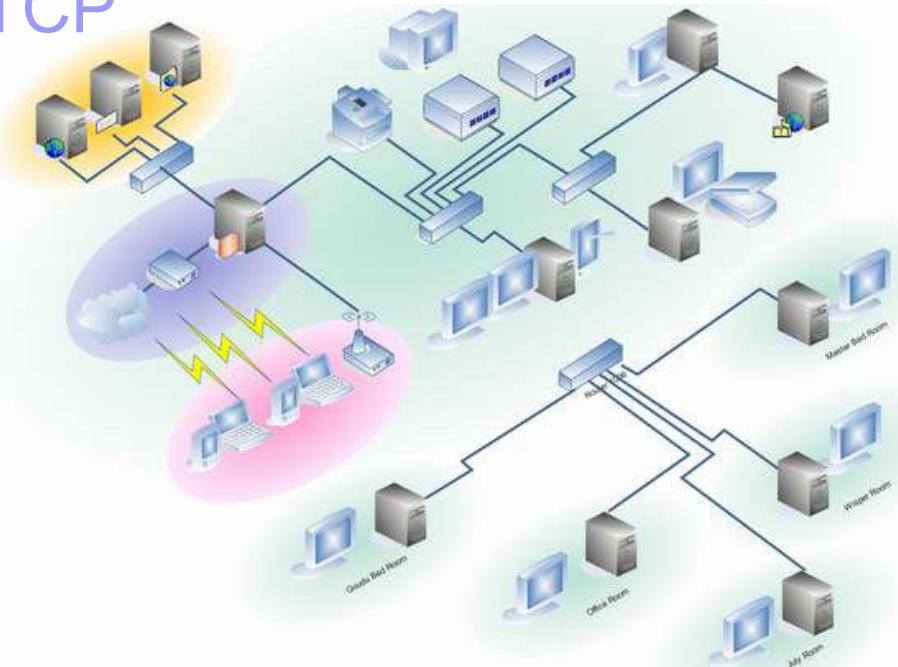
# Connection-oriented Demultiplexing: Example



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

# *Outline*

- Transport-layer services;
  - Multiplexing and demultiplexing;
  - Connectionless transport: UDP;
  - Principles of reliable data transfer;
  - Connection-oriented transport: TCP
    - Segment structure;
    - Reliable data transfer;
    - Flow control;
    - Connection management;
  - Principles of congestion control
  - TCP congestion control



# UDP: User Datagram Protocol [RFC 768]

- Simple, “bare bones”, Internet transport protocol;
- “**Best effort**” service – UDP segments may be:
  - Lost;
  - Delivered out of order to application.
- *Connectionless*:
  - No handshaking between UDP sender and receiver;
  - Each UDP segment is handled independently of others.

## Why is there a UDP?

- No connection establishment (which can add delay);
- Simple: no connection state at sender or receiver;
- Small segment header;
- No congestion control: UDP can “blast away” as fast as desired.

# UDP: User Datagram Protocol

- Often used for streaming multimedia applications:

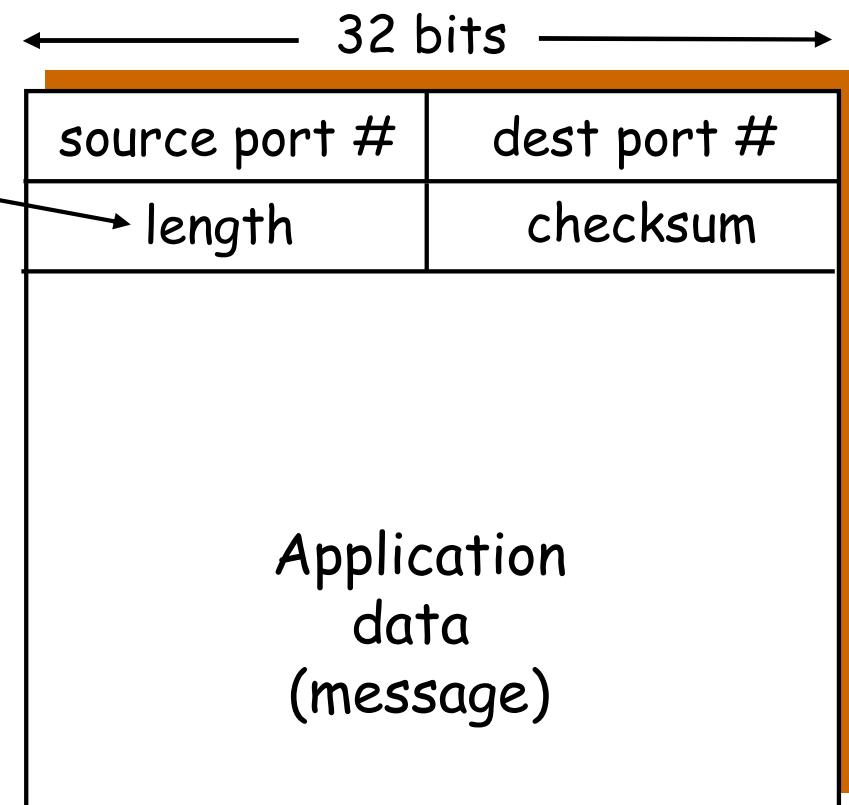
- Loss tolerant;
- Rate sensitive.

- Other UDP uses
  - DNS;
  - SNMP;
  - HTTP/3.

To have reliable transfer over UDP:

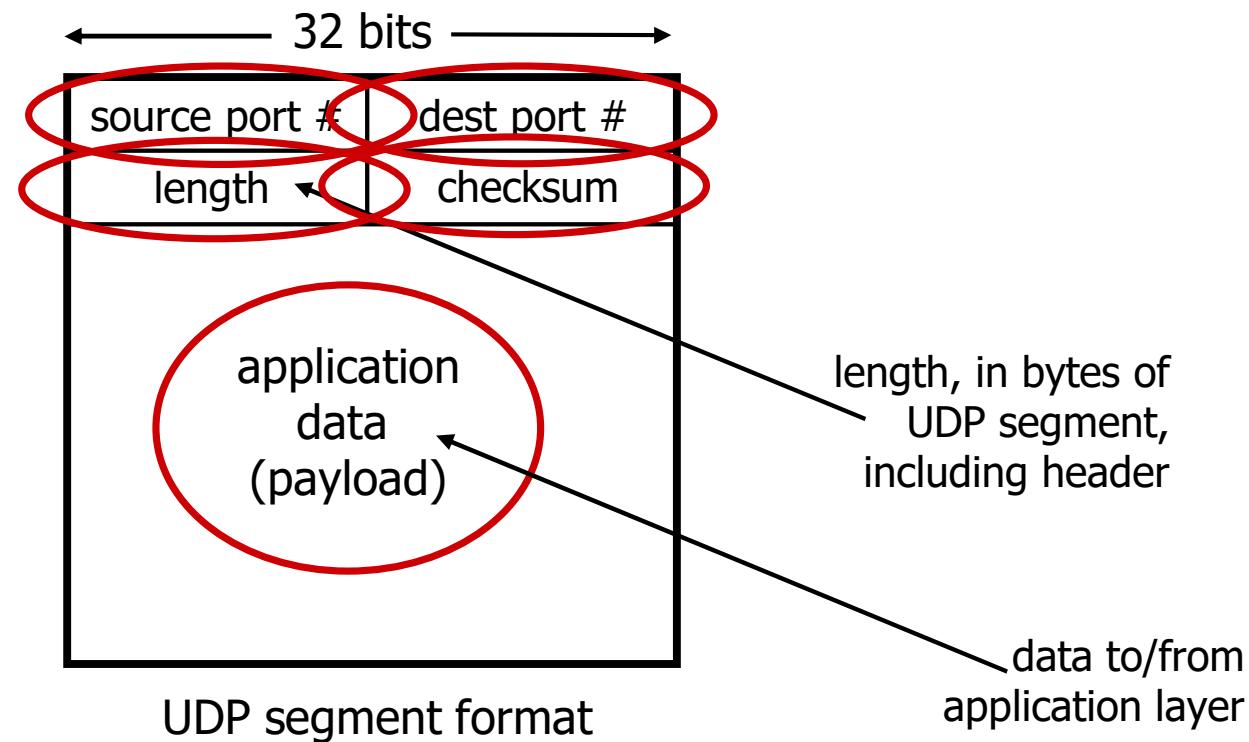
- Need to add reliability at application layer;
- Application-specific error recovery.
- Congestion control at application layer.

Length, in Bytes,  
of UDP segment,  
including header



UDP segment format

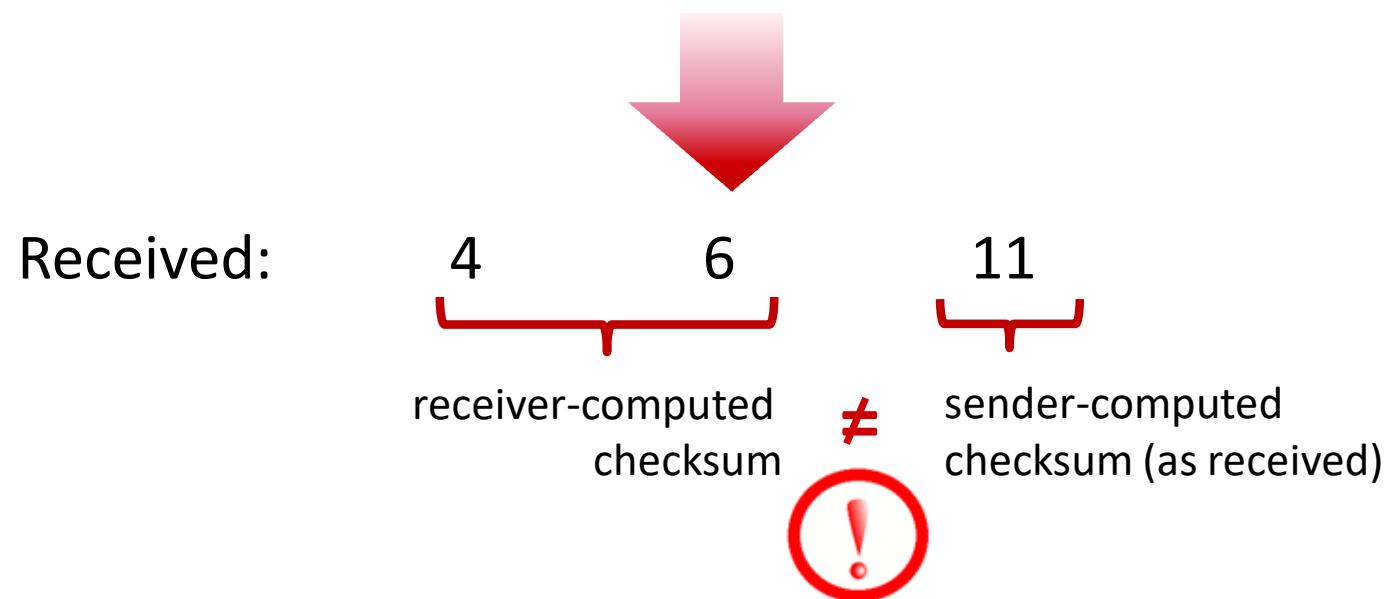
# UDP Segment Header



# UDP Checksum

**Goal:** detect errors (i.e., flipped bits) in transmitted segment

	1 <sup>st</sup> number	2 <sup>nd</sup> number	sum
Transmitted:	5	6	11



# UDP Checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment.

## Sender:

- Treat segment contents as sequence of 16-bit integers;
- Checksum: addition (1's complement sum) of segment contents;
- Sender puts checksum value into the **UDP checksum field**.

## Receiver:

- Compute checksum of received segment;
- Check if computed checksum equals checksum field value:
  - NO - error detected;
  - YES - no error detected.

*But there may be errors nonetheless?*

## UDP Checksum Example

- Note:
  - When adding numbers, a carryout from the most significant bit needs to be added to the result !
- Example: add two 16-bit integers:

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
	<hr/>															
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum (one's complement)	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

# Internet Checksum: Weak Protection!

example: add two 16-bit integers

	1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0	0 1
	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	1 0
	<hr/>	
wraparound	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1	
		
sum	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0	
checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1	

Even though numbers have changed (bit flips), **no** change in checksum!

## UDP Checksum

UDP checksum is computed over a pseudo header including:

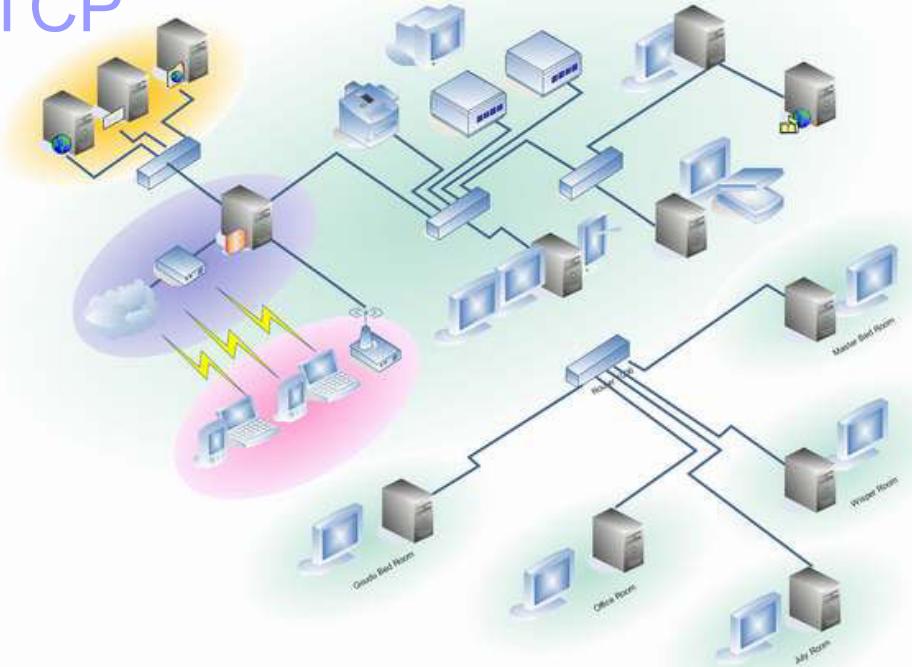
- the entire payload;
- the other fields in the header;
- some fields from the IP header.

IPv4 Pseudo Header Format

Offsets	Octet	0								1								2								3											
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
0	0	Source IPv4 Address																																			
4	32	Destination IPv4 Address																																			
8	64	Zeroes				Protocol												UDP Length																			
12	96	Source Port																Destination Port																			
16	128	Length																Checksum																			
20	160+	Data																																			

# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
  - Connection management;
- Principles of congestion control
- TCP congestion control



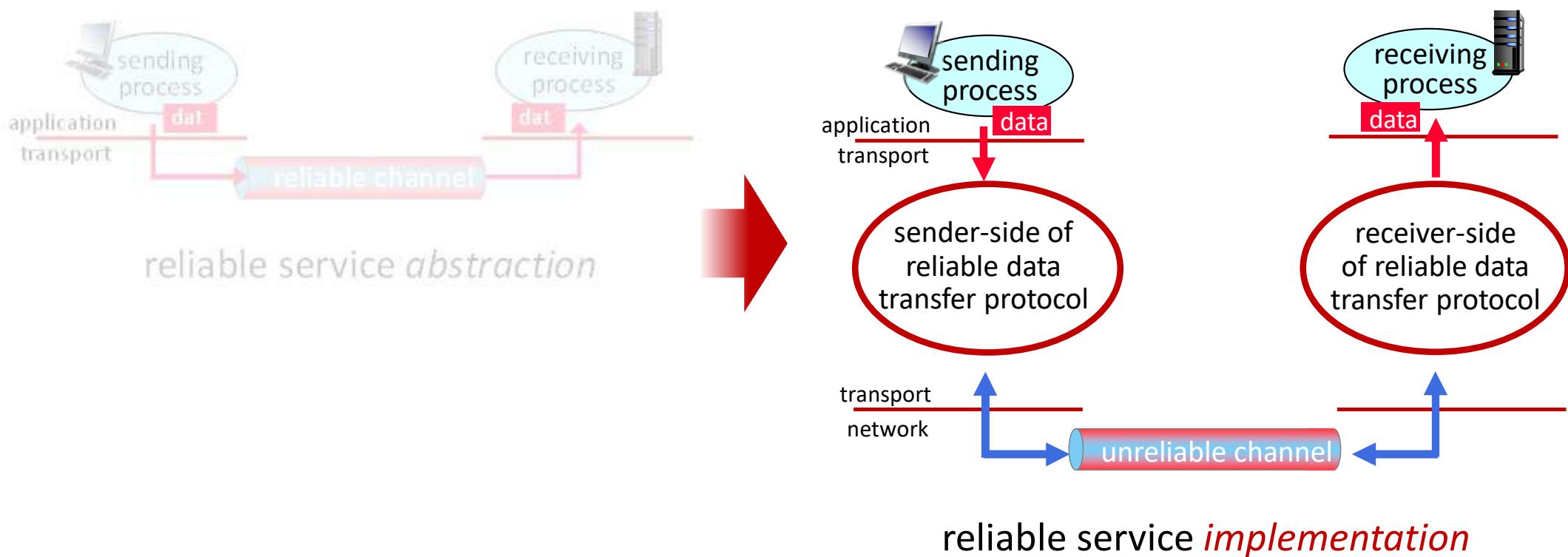
# Principles of Reliable Data Transfer



reliable service *abstraction*

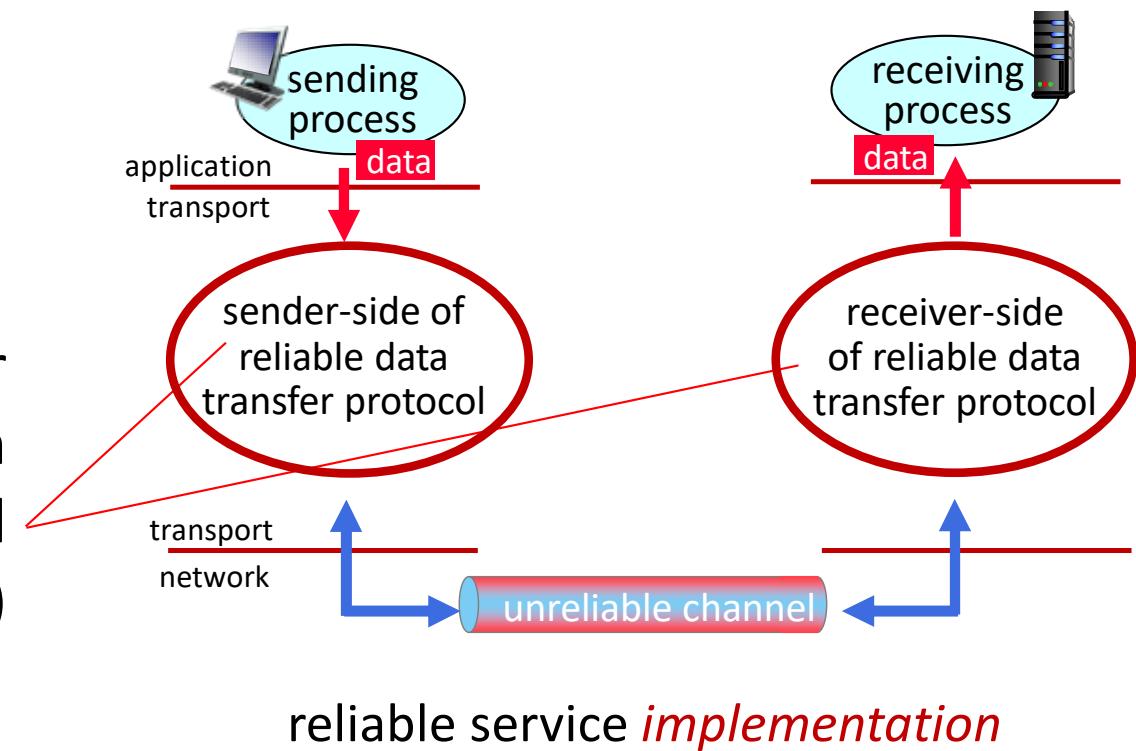
- Reliable transfer of information is important for many applications;
- Among the top-10 list of important networking topics!

# Principles of Reliable Data Transfer



# Principles of Reliable Data Transfer

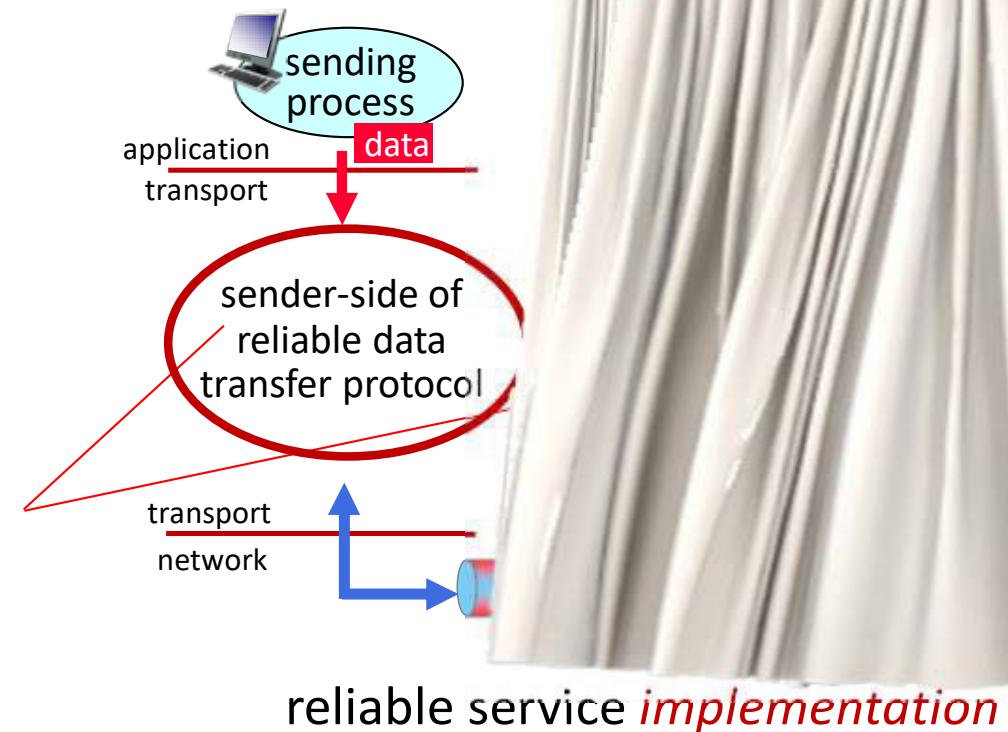
Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)



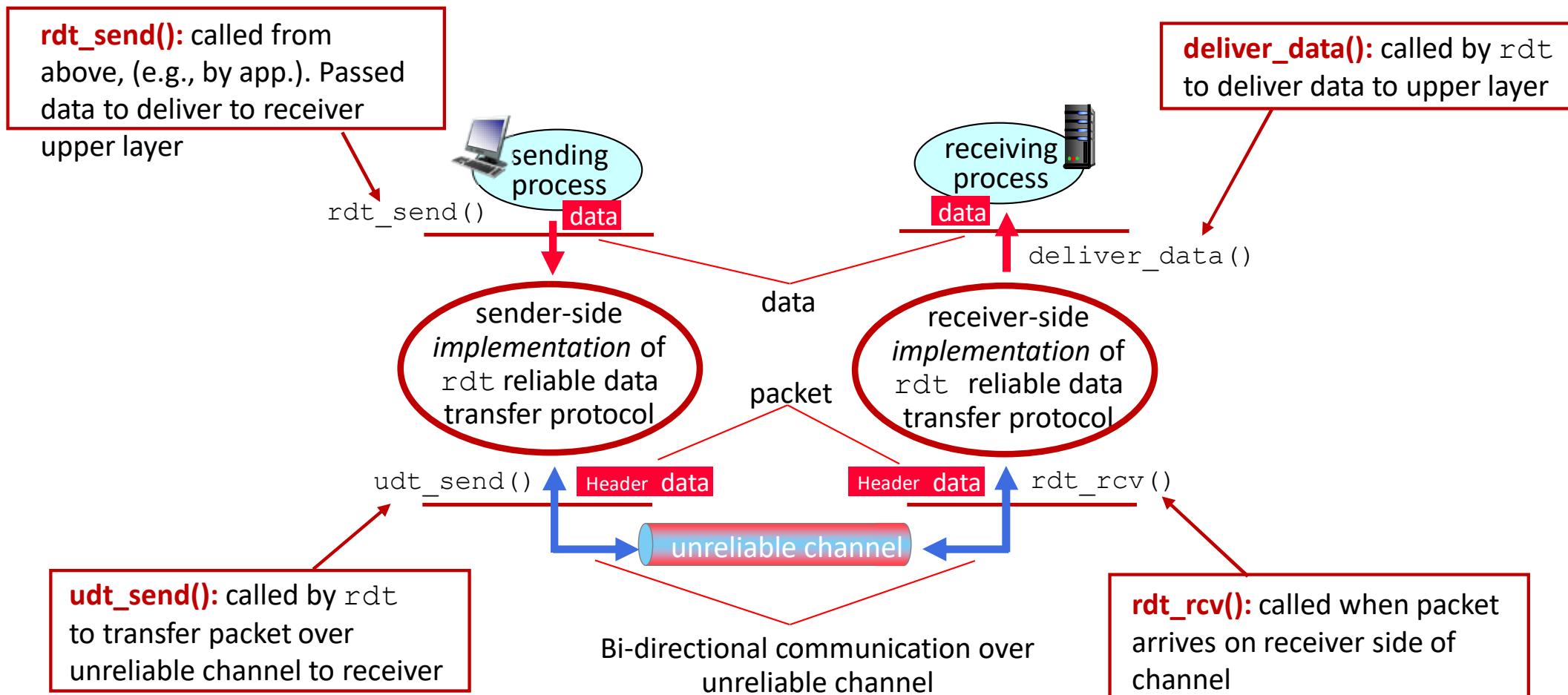
# Principles of Reliable Data Transfer

Sender, receiver do *not* know the “state” of each other, e.g., was a message received?

- unless communicated via a message



# Reliable Data Transfer Protocol (rdt): Interfaces

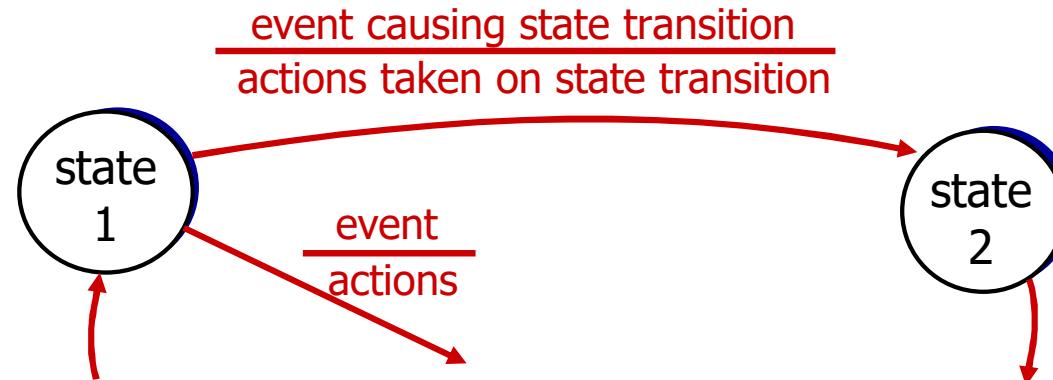


# Reliable Data Transfer: Getting Started

We will:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow in both directions!
- use finite state machines (FSM) to specify sender, receiver

**state:** when in this “state”  
next state uniquely  
determined by next  
event



# *rdt1.0:*

## *Reliable Transfer over a Reliable Channel*

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- *separate* FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel



# *rdt2.0: Channel with Bit Errors*

- underlying channel may flip bits in packet
  - checksum (e.g., Internet checksum) to detect bit errors
- *the question:* how to recover from errors?

*How do humans recover from “errors” during conversation?*

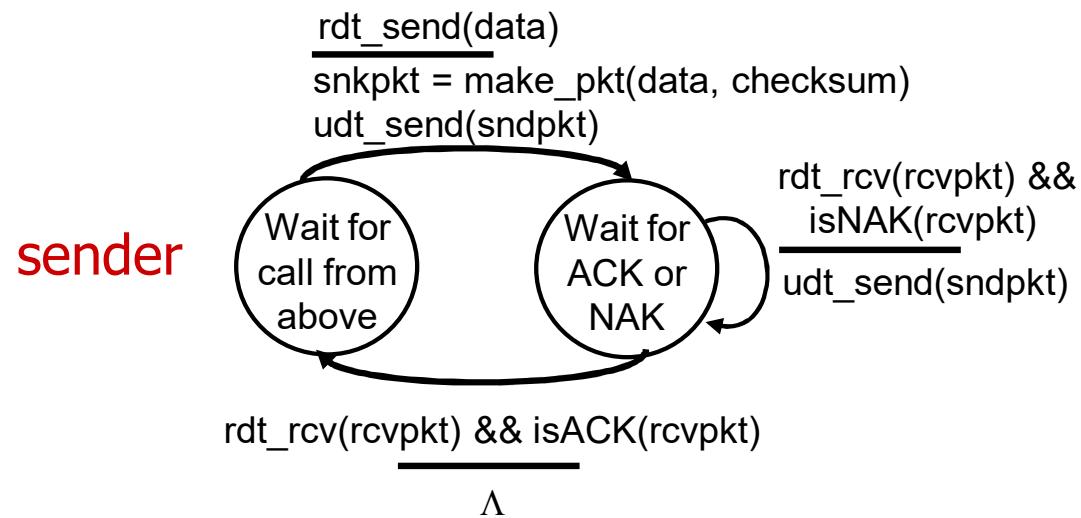
# rdt2.0: Channel with Bit Errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the question:* how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender *retransmits* pkt on receipt of NAK

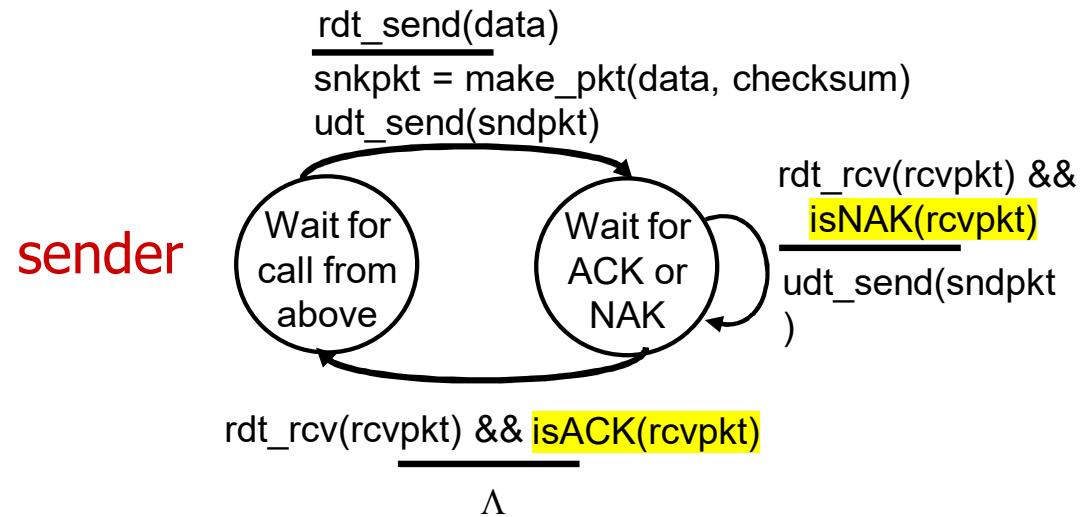
stop and wait

sender sends one packet, then waits for receiver response

# *rdt2.0: FSM Specification*



# *rdt2.0: FSM Specification*

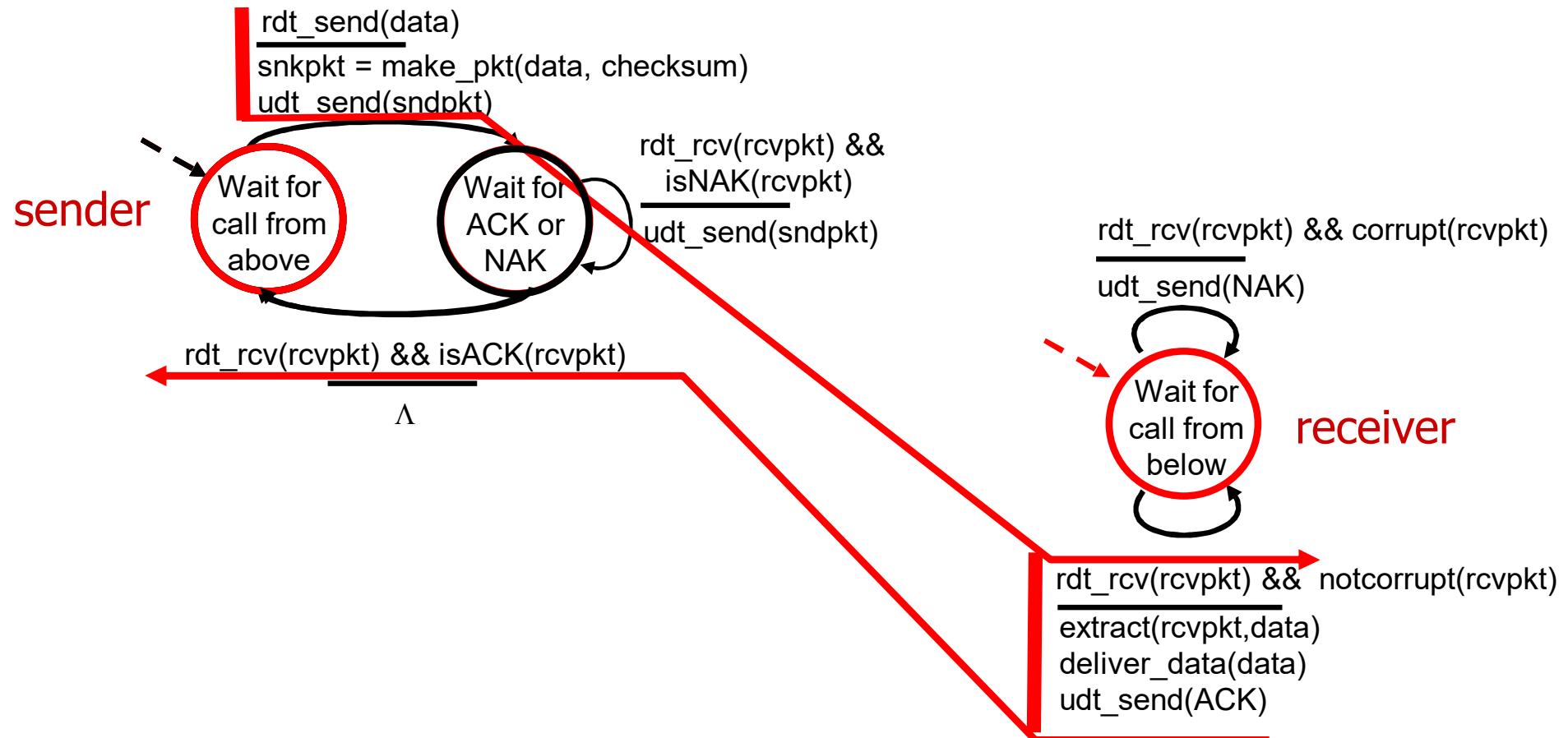


**Note:** “state” of receiver (did the receiver get my message correctly?) isn’t known to sender unless somehow communicated from receiver to sender

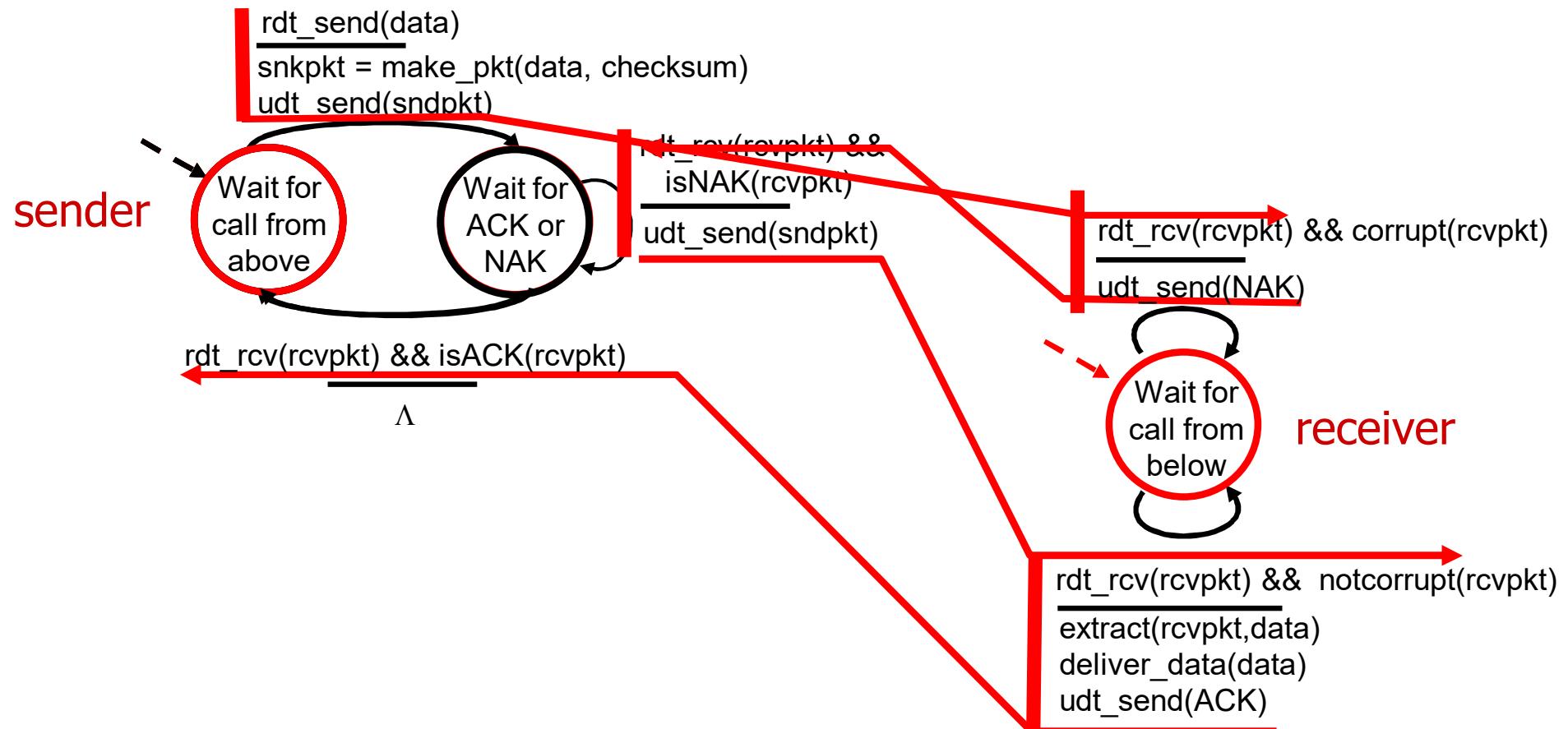
- that’s why we need a protocol!



# *rdt2.0: Operation with No Errors*



# *rdt2.0: Corrupted Packet Scenario*



# *rdt2.0 Has a Fatal Flaw!*

what happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

handling duplicates:

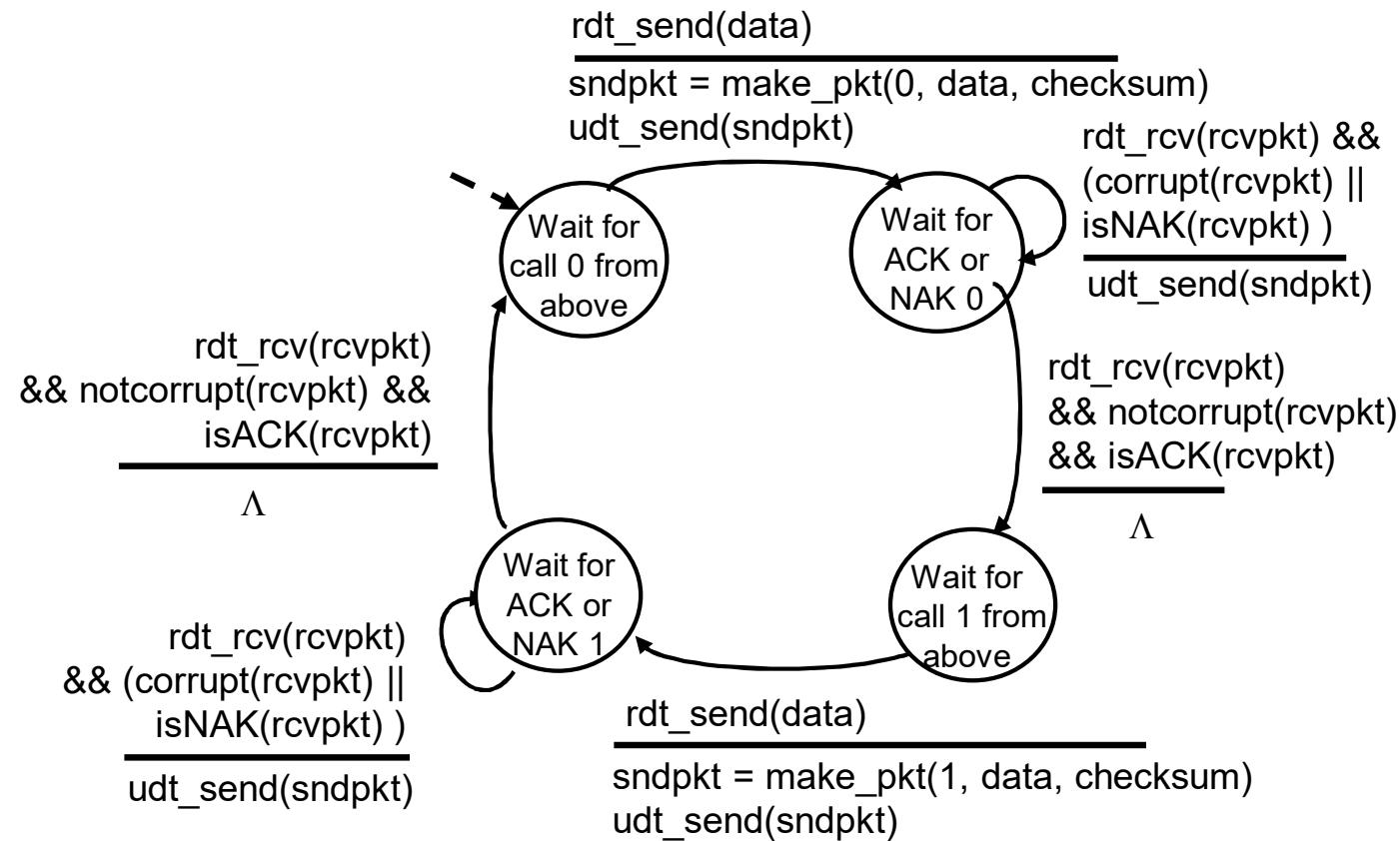
- sender retransmits current pkt if ACK/NAK corrupted
- sender adds ***sequence number*** to each pkt
- receiver discards duplicate pkt

stop and wait

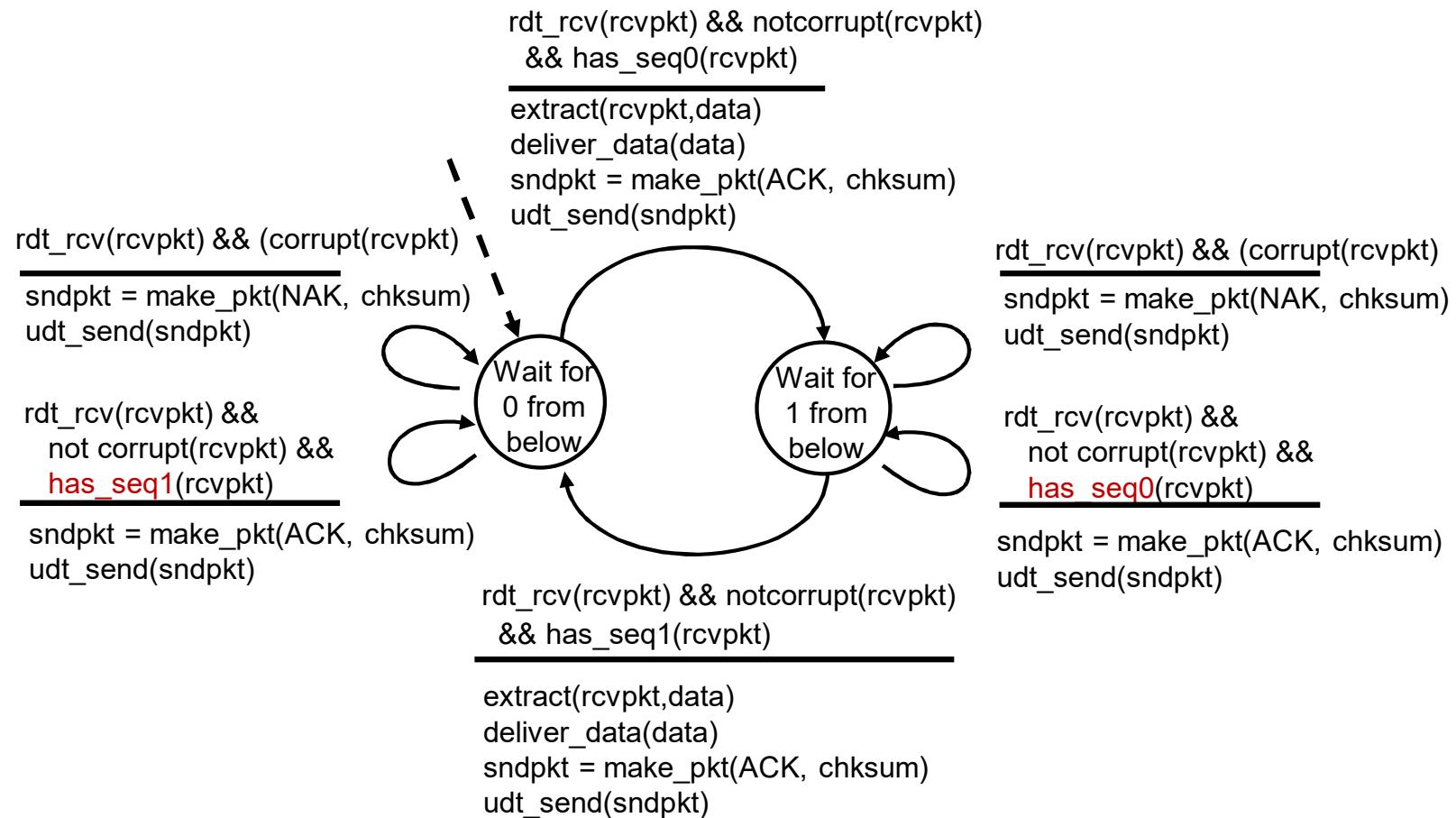
sender sends one packet, then waits for receiver response

# *rdt2.1:*

## *Sender, Handling Garbled ACK/NAKs*



# rdt2.1: *Receiver, Handling Garbled ACK/NAKs*



## rdt2.1: Discussion

### Sender:

- Sequence number (seq #) added to packet;
- **Two seq #'s (0 and 1) are sufficient. Why?**
- Must check if received ACK/NAK is corrupted;
- Twice as many states:
  - State must “remember” whether “current” packet has 0 or 1 seq #.

### Receiver:

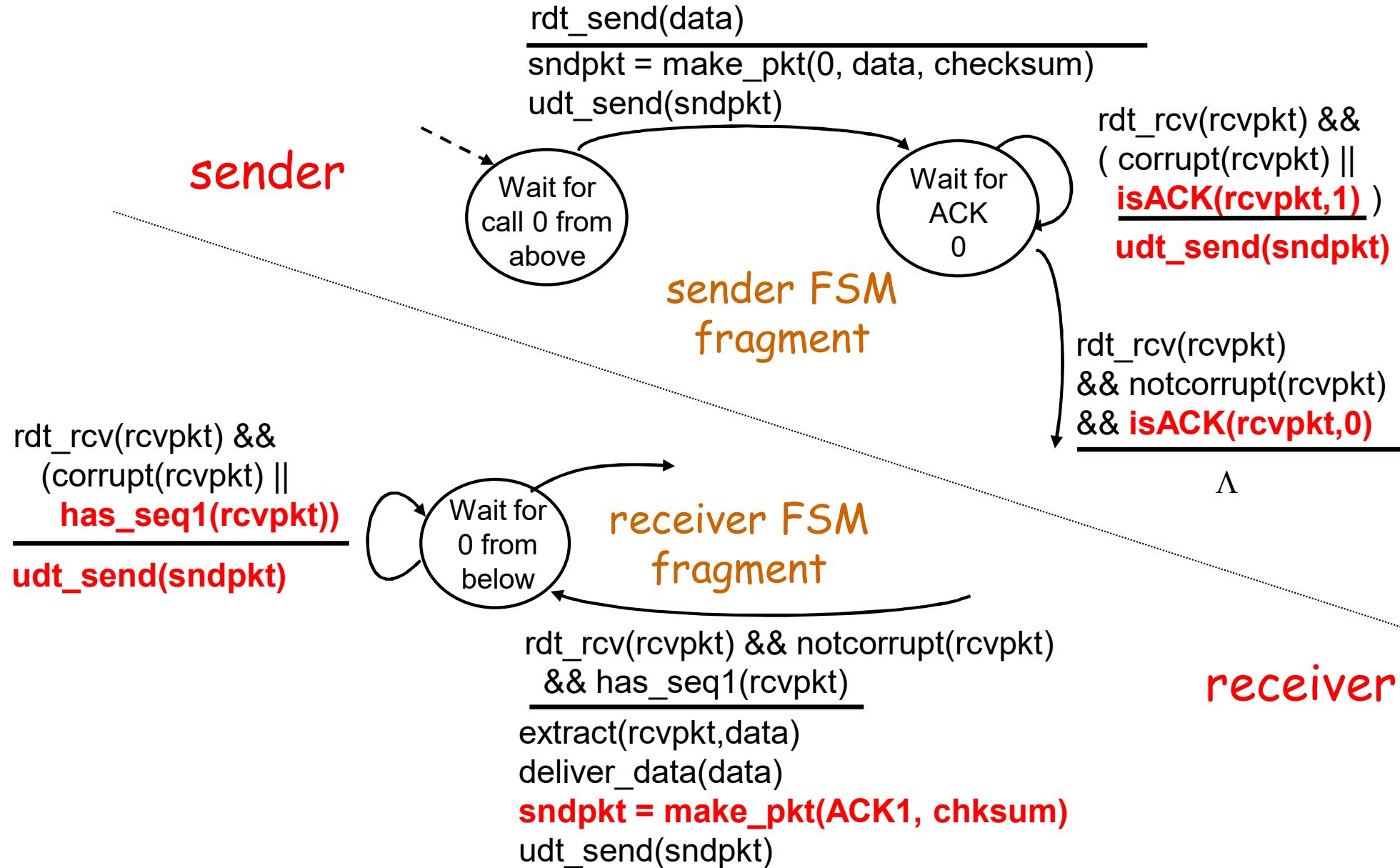
- Must check if received packet is duplicate:
  - State indicates whether 0 or 1 is expected packet seq #;

Note: receiver can *not* know if its last ACK/NAK was received OK at the sender.

## *rdt2.2: a NAK-Free Protocol*

- Same functionality as rdt2.1, but **using ACKs only**;
- **Instead of NAK, receiver sends ACK for last packet received OK:**
  - Receiver must *explicitly* include seq # of packet being ACKed.
- Duplicate ACK at sender results in same action as a NAK would have:  
*retransmit the current packet.*

## *rdt2.2: Sender, Receiver (Fragments)*



## *rdt3.0: Channels with Errors and Loss*

### New assumption:

The underlying **channel can also lose packets** (data or ACKs):

- Checksum, seq #, ACKs, retransmissions - help but are **not enough!**

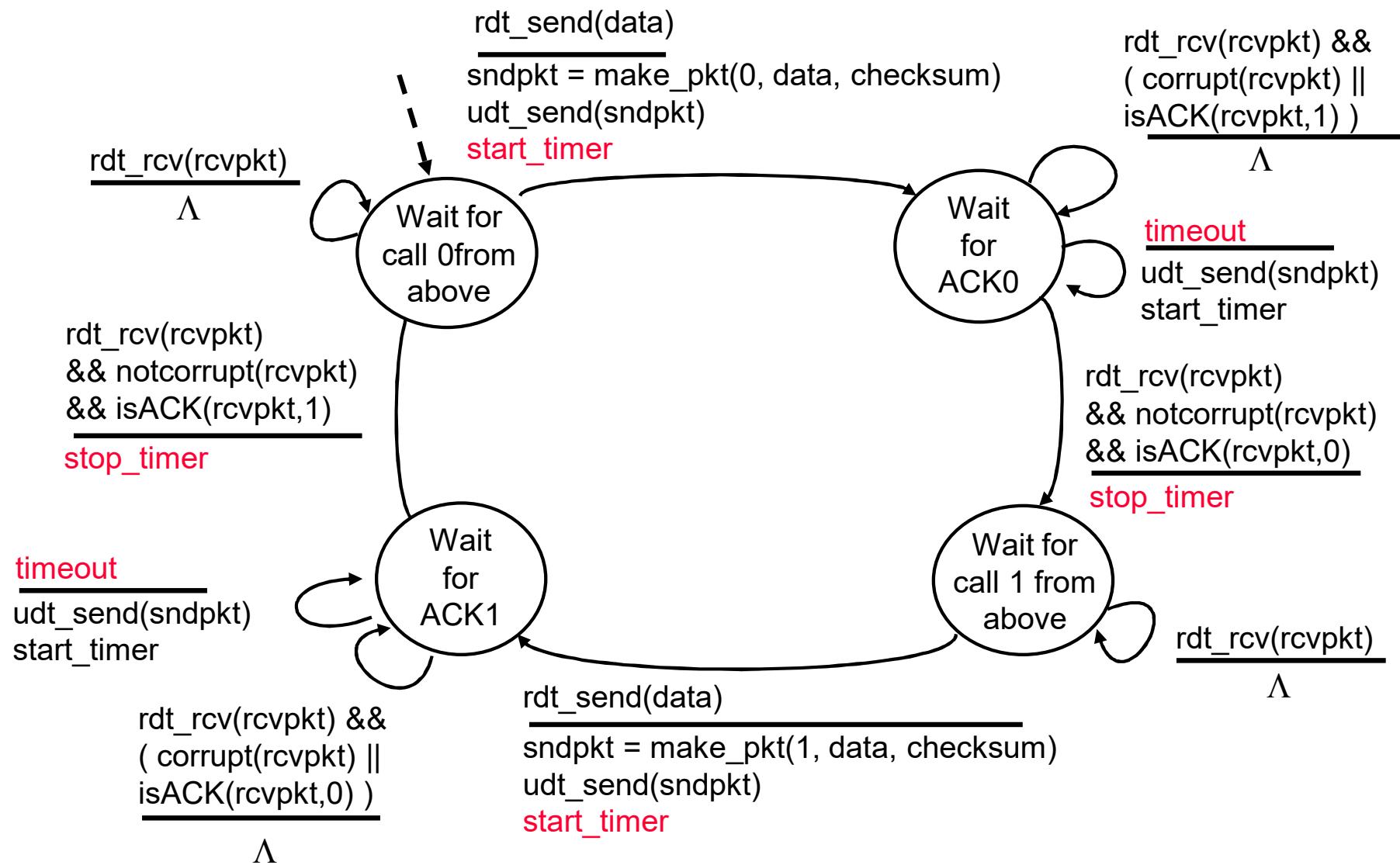
### Approach:

Sender waits “reasonable” amount of time for ACK.

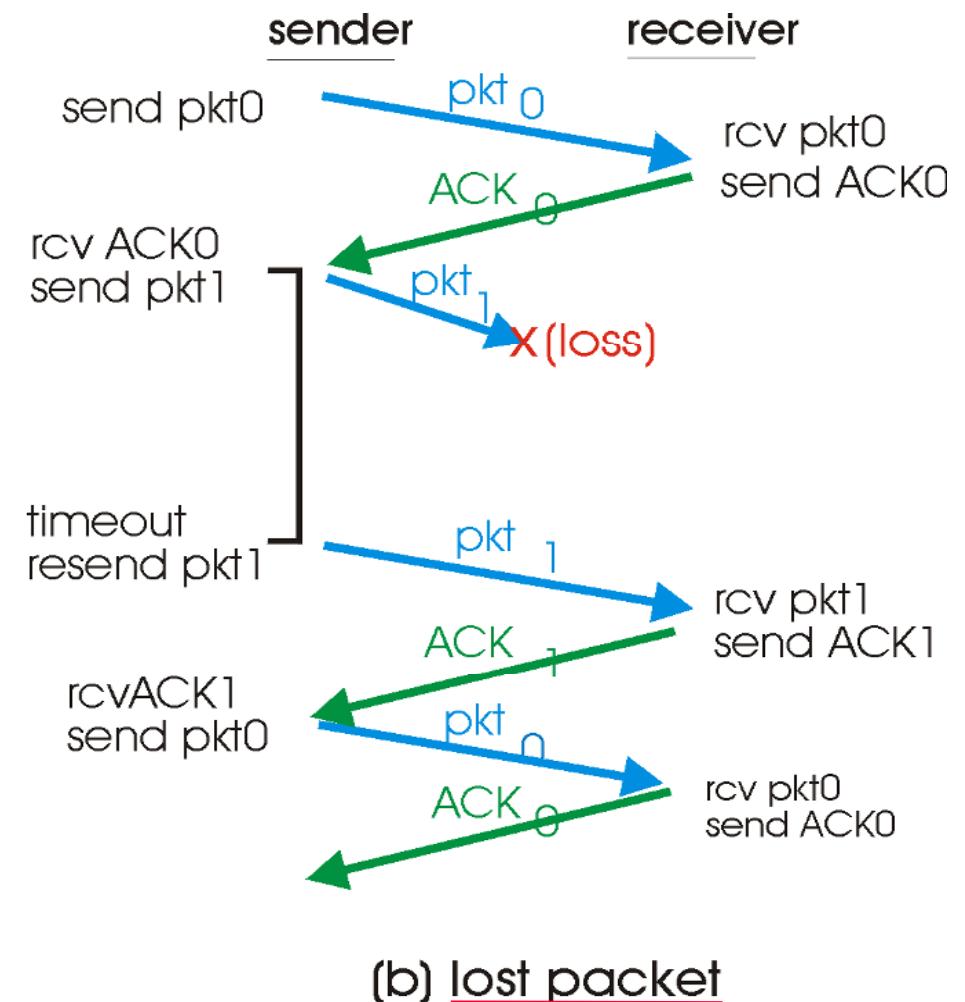
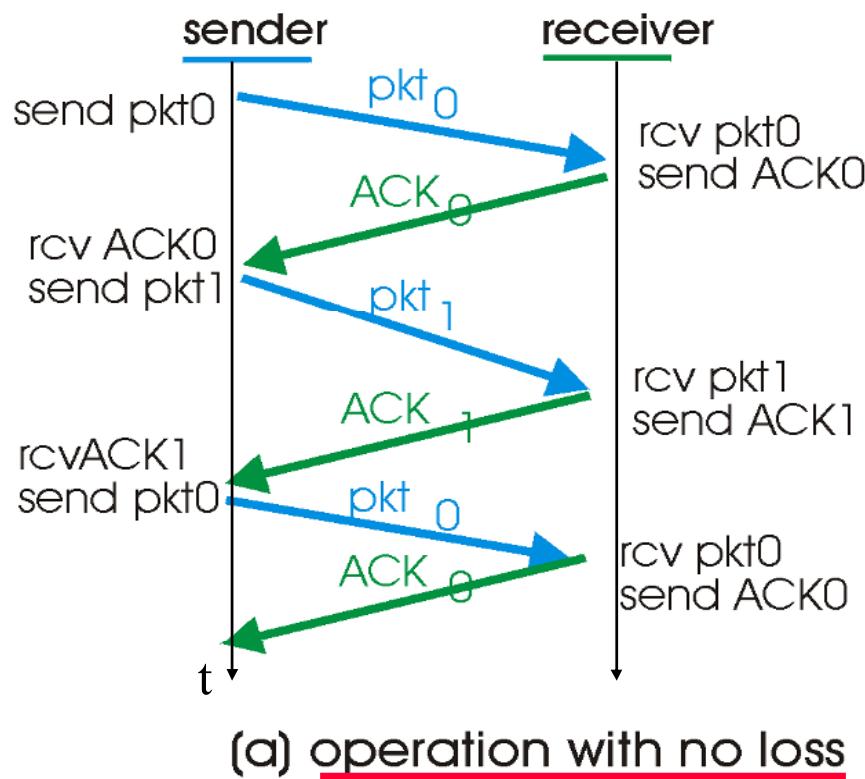
- Retransmits if no ACK received in this time;
- If packet (or ACK) was just delayed (not lost):
  - Retransmission will be duplicate, but use of seq. #'s already handles this;
  - Receiver must specify seq # of packet being ACKed.
- Requires **countdown timer**.



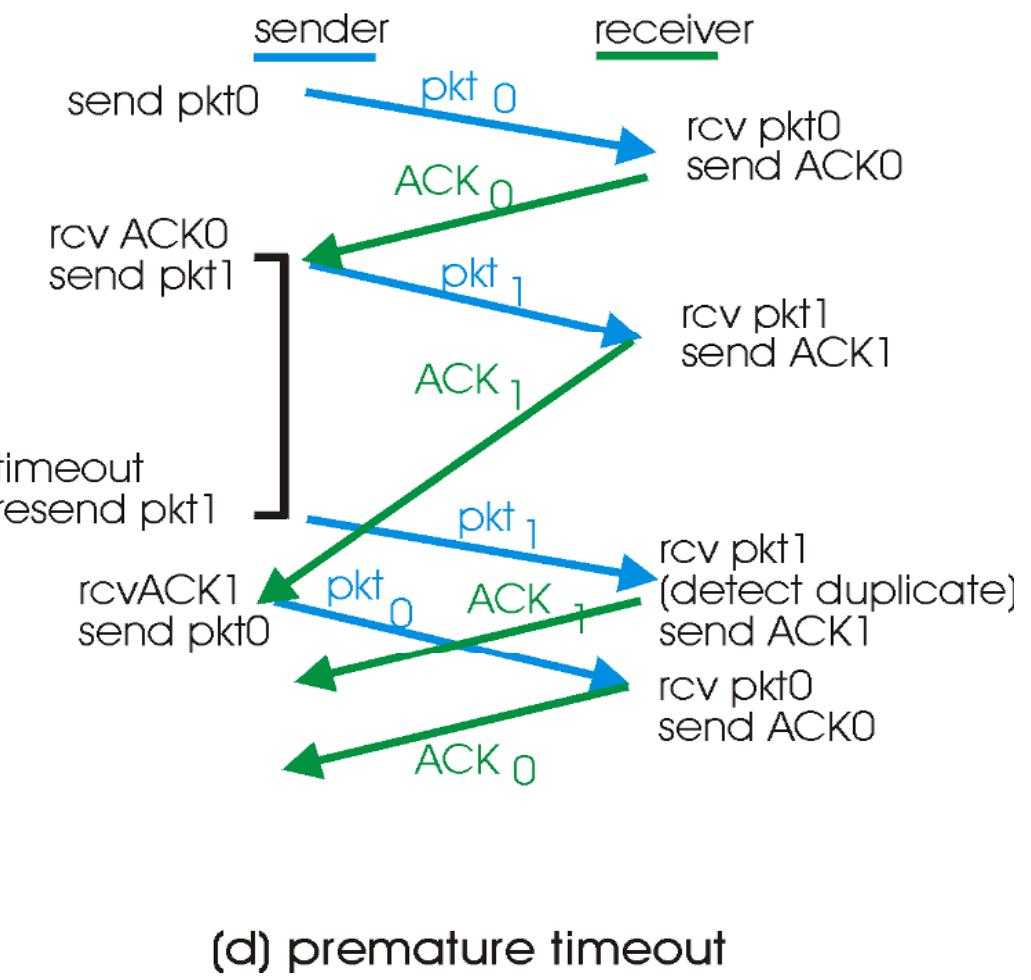
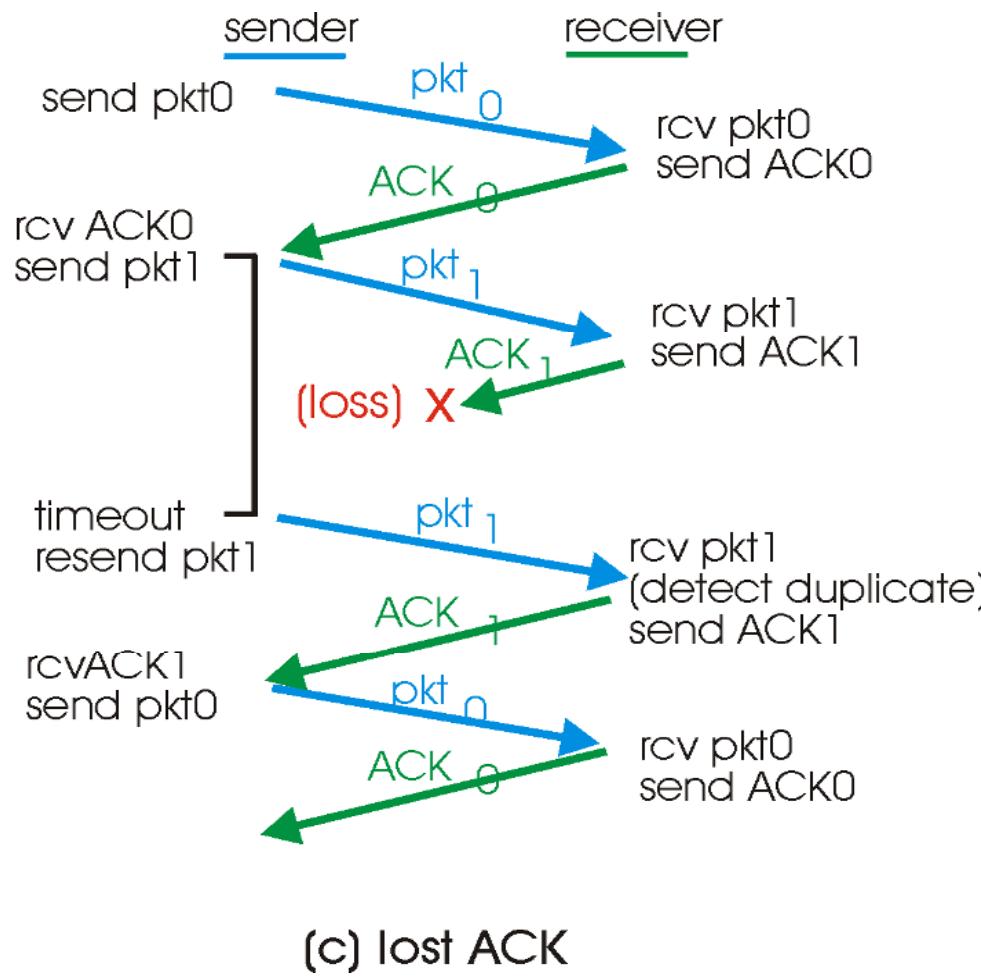
# rdt3.0 Sender



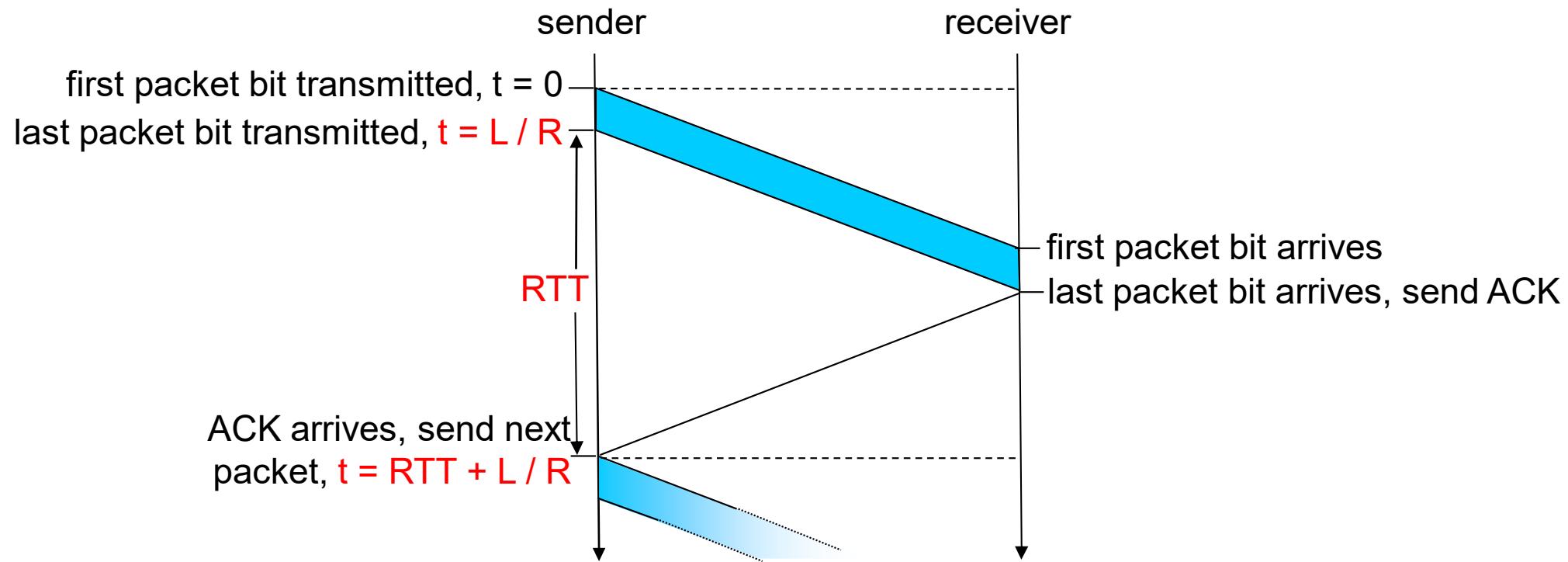
## rdt3.0 in Action – Stop and Wait



# *rdt3.0 in Action – Stop and Wait*



## *rdt3.0: Stop-and-Wait Operation*



$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

# *Performance of rdt3.0*

## *Stop and Wait*

- ❑ rdt3.0 works, but performance is quite poor...
- ❑ Ex: 1 Gbit/s link, 15 ms prop. delay, 8000 bit packet:

$$t_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bit/s}} = 8 \mu\text{s}$$

- ❑  $U_{\text{sender}}$ : utilization – fraction of time that sender is busy sending:

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

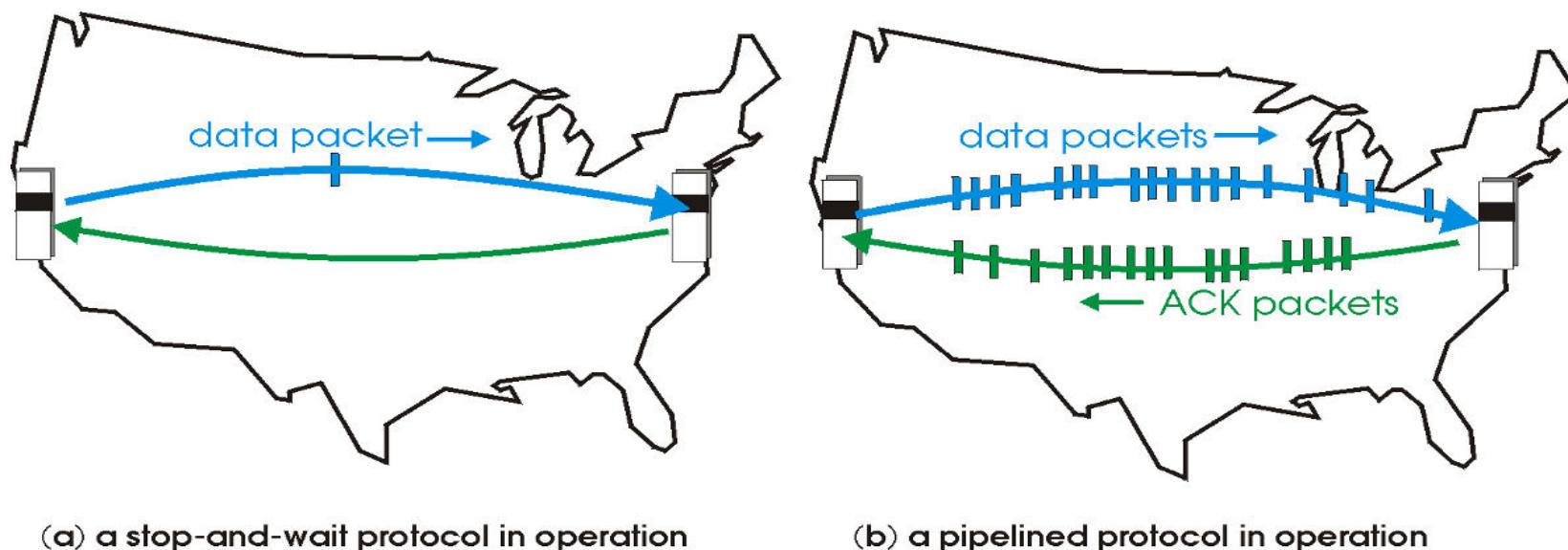
- ❑ 1 kB packet every 30 ms -> 33kB/s (267 kbit/s) throughput over 1 Gbit/s link;
- ❑ The protocol limits usage of physical resources!

# Sliding Window Protocols (or Pipelined Protocols)

## Sliding Window / Pipelining

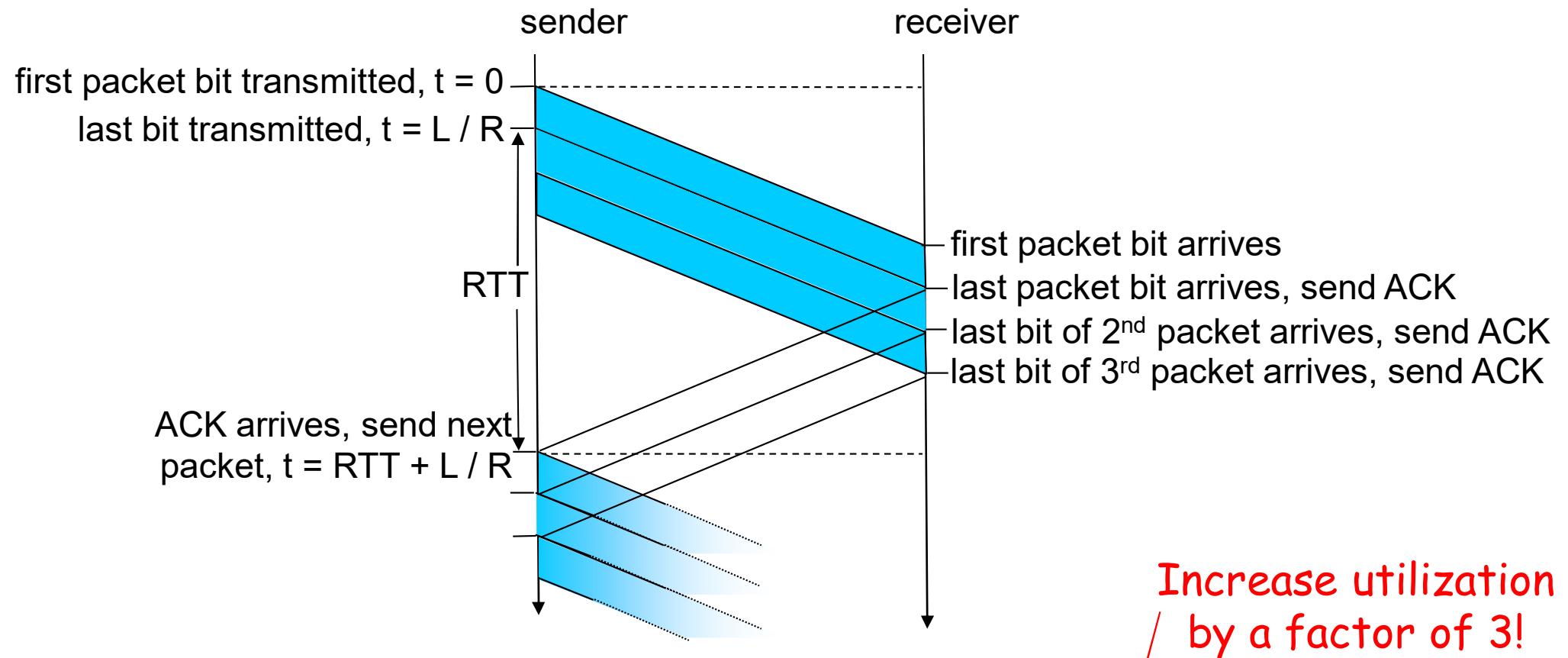
Sender allows multiple “in-flight”, yet-to-be-acknowledged, packets:

- Range of sequence numbers must be increased;
- Buffering at sender and/or receiver needed.



- Two generic forms of sliding window protocols: **Go-Back-N, Selective Repeat.**

# *Sliding Window Protocols: Increased Utilization*



$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

# *Sliding Window Protocols*

## **Go-back-N**

- Sender can have up to N unacked packets in pipeline;
- Receiver sends cumulative ACKs:
  - Doesn't ACK packet if there is a gap.
- Sender has timer for oldest unacked packet:
  - If timer expires, retransmit all unacked packets.

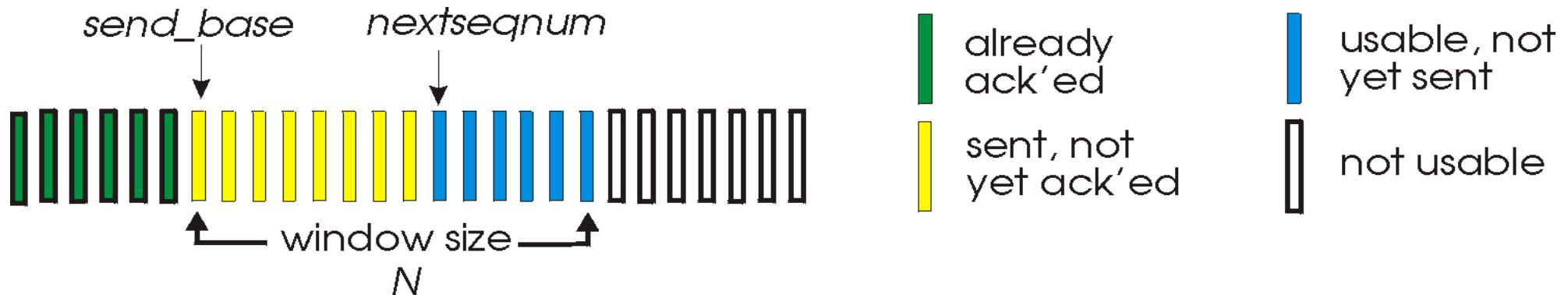
## **Selective Repeat**

- Sender can have up to N unacked packets in pipeline;
- Receiver ACKs individual packets;
- Sender maintains timer for each unacked packet:
  - When timer expires, retransmit only the unack packet.

# Go-Back-N

## Sender:

- k-bit sequence number ( $0 - 2^{k-1}$ ) in packet header;
- “Window” of up to N, consecutive unacked packets allowed:



- ACK( $n$ ): ACKs all packets up to, including seq. #  $n$  – “**cumulative ACK**”:
  - May receive duplicate ACKs.
- Timer for each in-flight packet;
- $\text{timeout}(n)$ : retransmit packet  $n$  and all higher seq. # packets in window.

Test applet at:

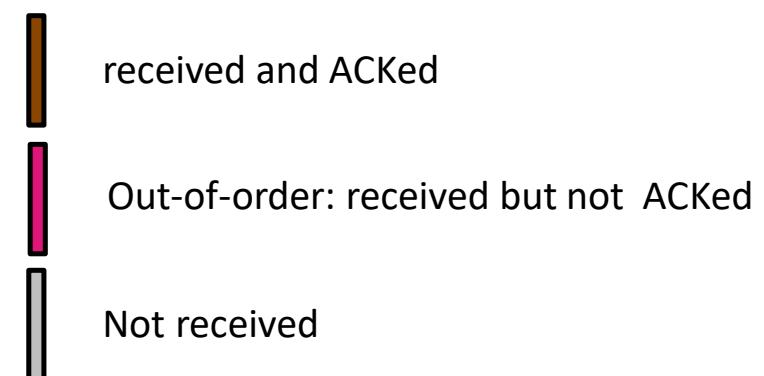
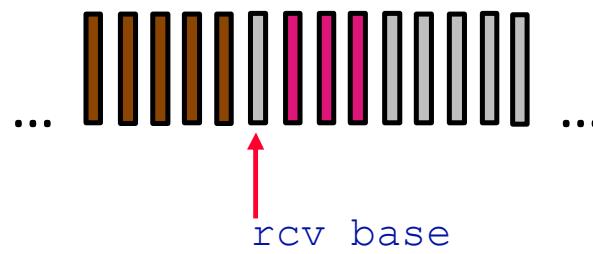
[https://media.pearsoncmg.com/aw/ecs\\_kurose\\_comnetwork\\_7/cw/content/interactiveanimations/go-back-n-protocol/index.html](https://media.pearsoncmg.com/aw/ecs_kurose_comnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html)

# Go-Back-N: Receiver

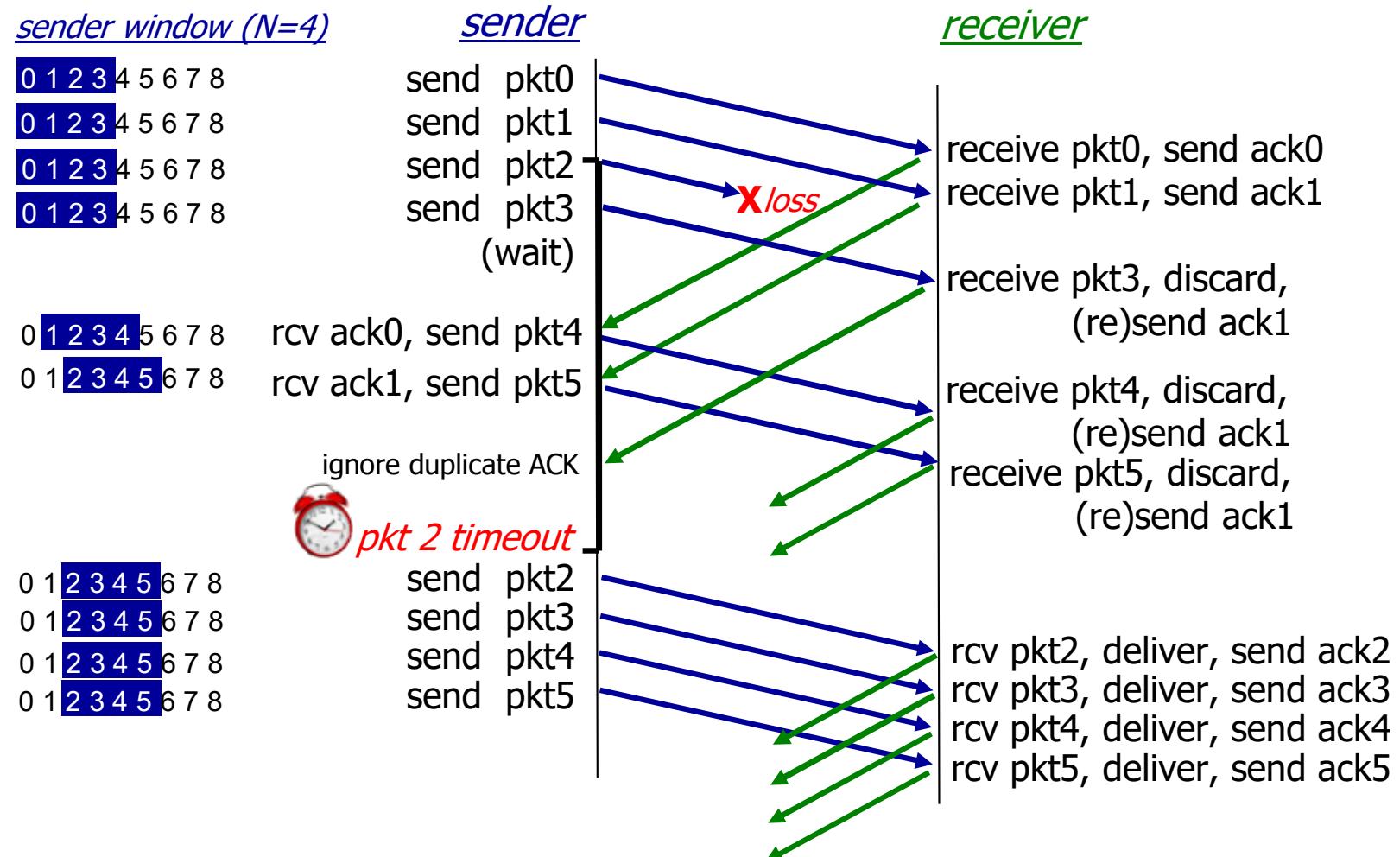
Always send ACK for correctly received packet with highest *in-order* seq #:

- May generate duplicate ACKs;
- Need only remember `rcv_base`.
- Out-of-order packet:
  - Discard (don't buffer) -> **no receiver buffering!** (implementation decision)
  - Re-ack packet with highest in-order seq #.

Receiver view of sequence number space:



# Go-Back-N in Action



# Go-Back-N

## Example:

- Sender window size  $N_w$ :

$$N_w = 6$$

- Assume channel keeps the order of sent frames;

- Frames are numbered modulo N:

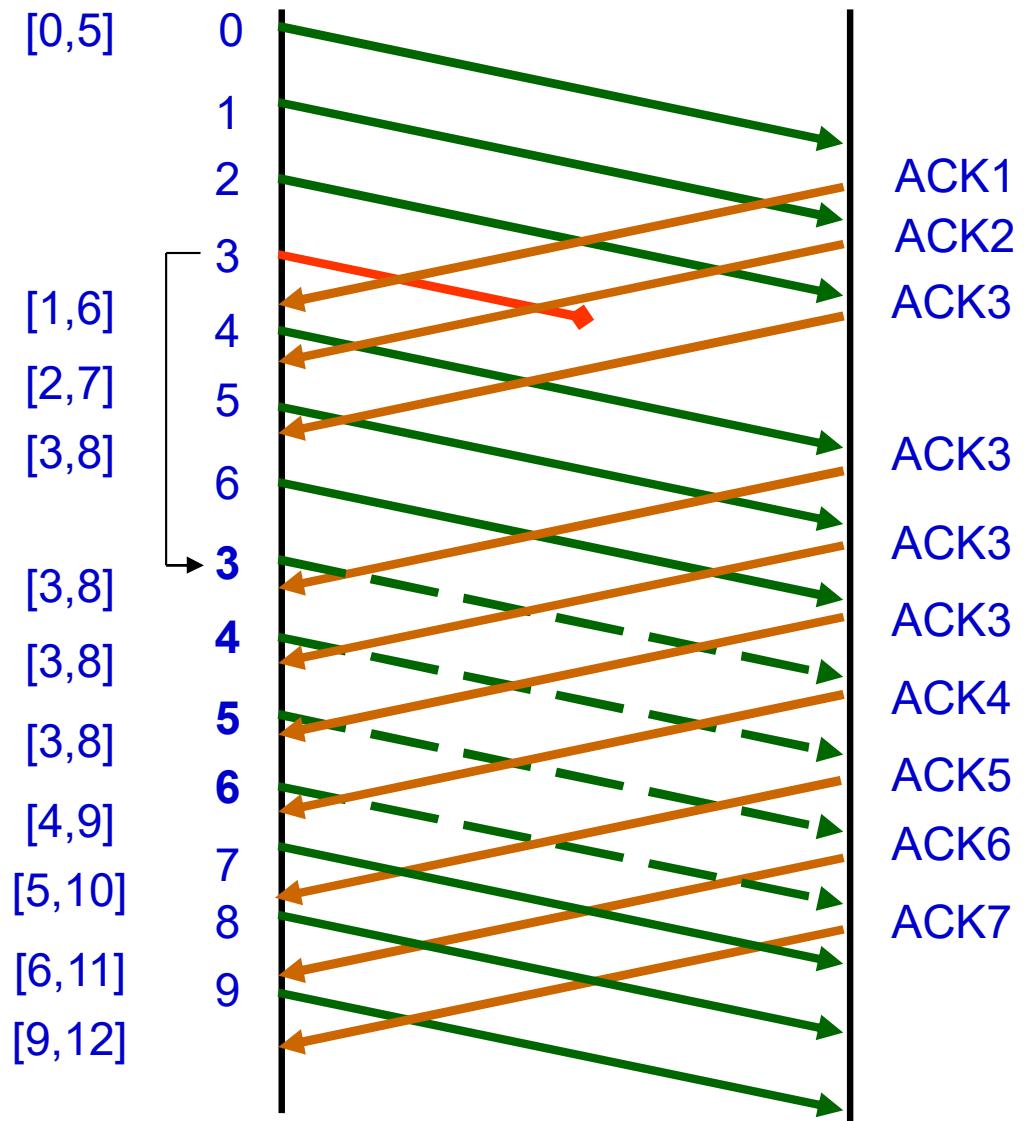
$$N = N_w + 1$$

- Flow control can be done by controlling the cadence of ACKs;

- Max sender window size:

$$N_w \leq N - 1, \text{ with } N = 2^k$$

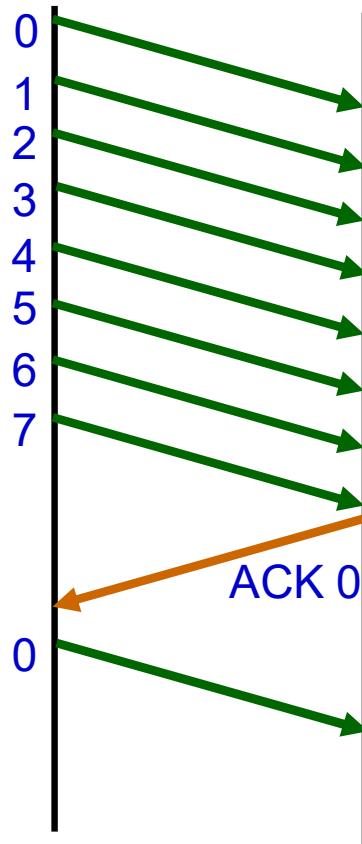
$k$  – number of bits used for frame numbering



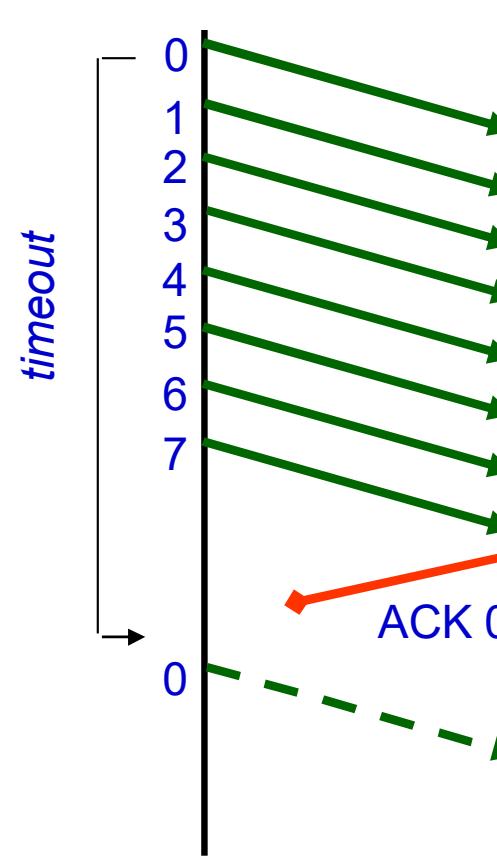
# Go-Back-N: Max Sender Window Size

Max sender window size:  $N_w \leq N - 1$ , with  $N = 2^k$

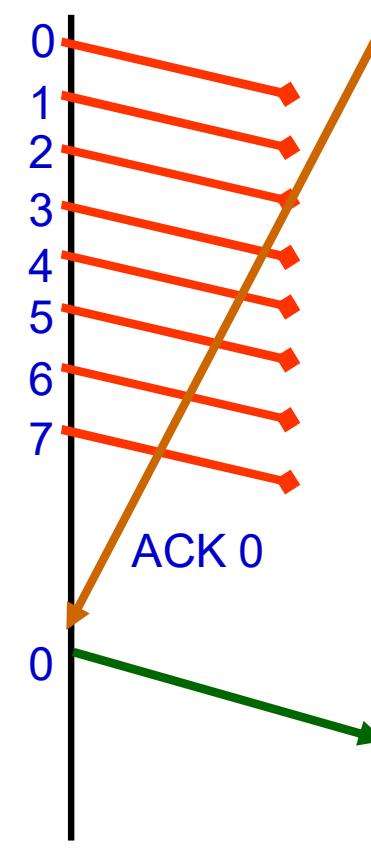
Example:  $N=8$  identifiers available ( $k = 3$ ), sender window size:  $N_w = 8$



Normal operation (no errors)



8 frames duplicated

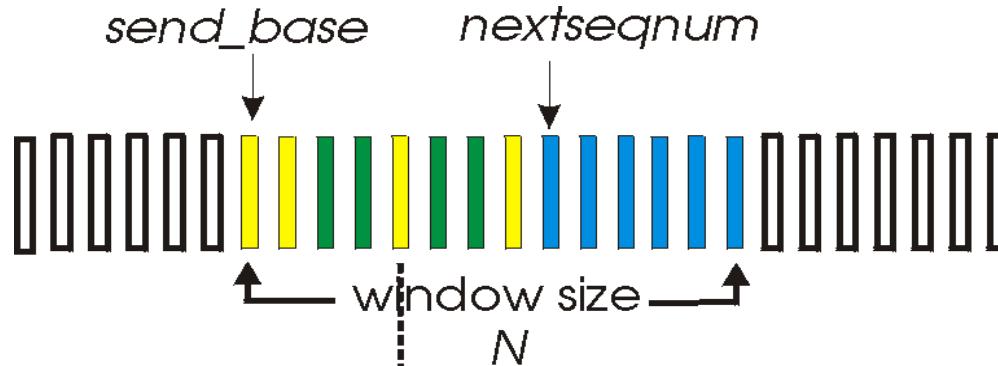


8 frames lost

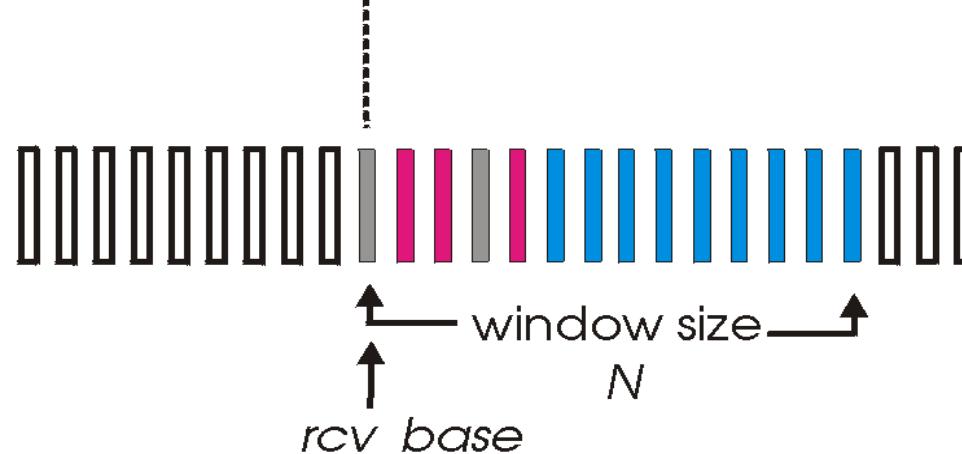
## Selective Repeat

- Receiver *individually* acknowledges all correctly received packets:
  - Buffer packets, as needed, for eventual in-order delivery to upper layer;
- Sender only resends packets for which ACK was not received:
  - Sender timer for each unACKed packet;
- Sender window:
  - N consecutive seq #'s;
  - Again limits seq #'s of sent, unACKed packets.

# Selective Repeat: Sender, Receiver Windows



(a) sender view of sequence numbers



Test applet at:

[https://media.pearsoncmg.com/aw/ecs\\_kurose\\_compnetwork\\_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html](https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html)

# Selective Repeat: Sender and Receiver

## Sender

data from above:

- if next available seq # in window, send packet

**timeout( $n$ ):**

- resend packet  $n$ , restart timer

**ACK( $n$ )** in [sendbase,sendbase+N]:

- mark packet  $n$  as received
- if  $n$  smallest unACKed packet, advance window base to next unACKed seq #

## Receiver

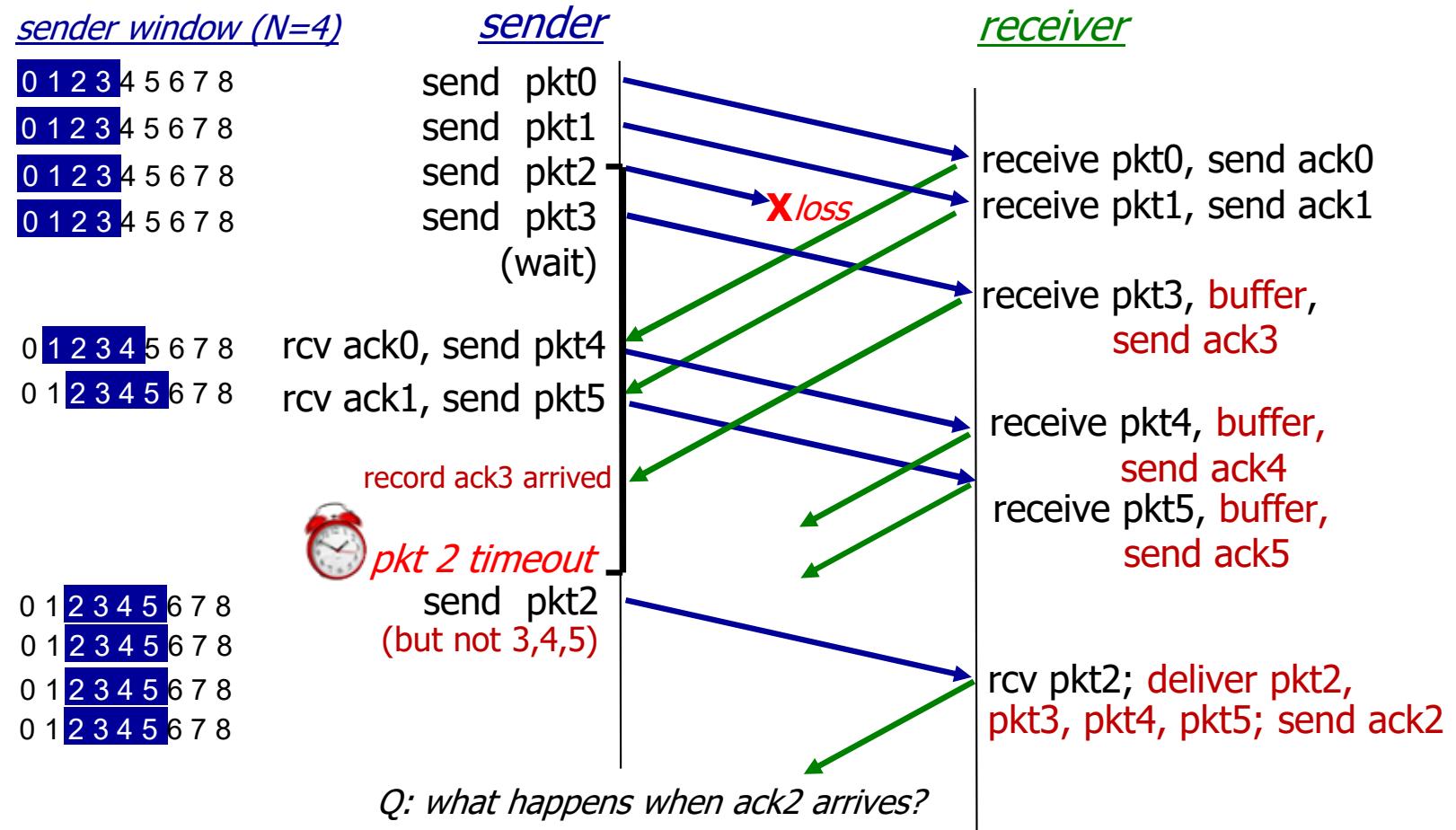
packet  $n$  in [rcvbase, rcvbase+N-1]

- send ACK( $n$ )
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

packet  $n$  in [rcvbase-N,rcvbase-1]

- ACK( $n$ )
- otherwise:
  - ignore

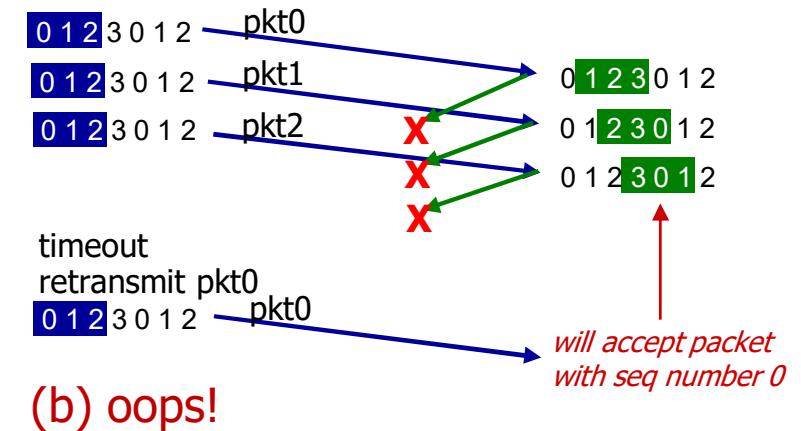
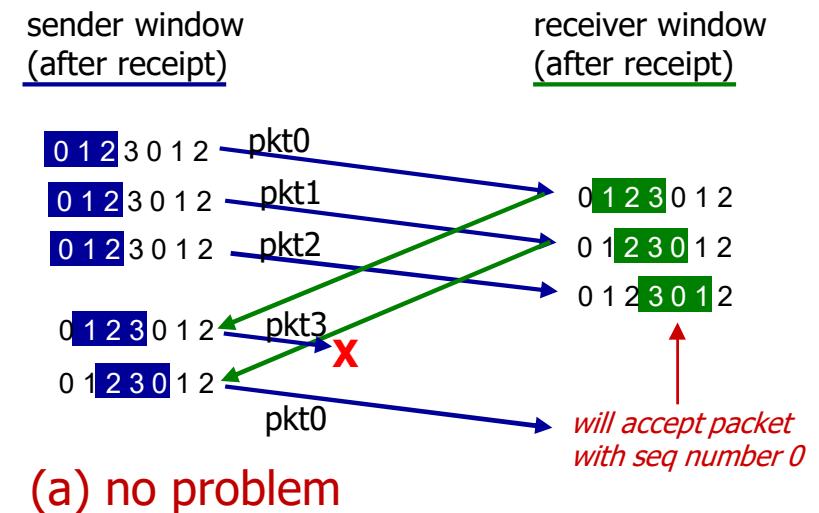
# *Selective Repeat in action*



# Selective repeat: a dilemma!

example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3

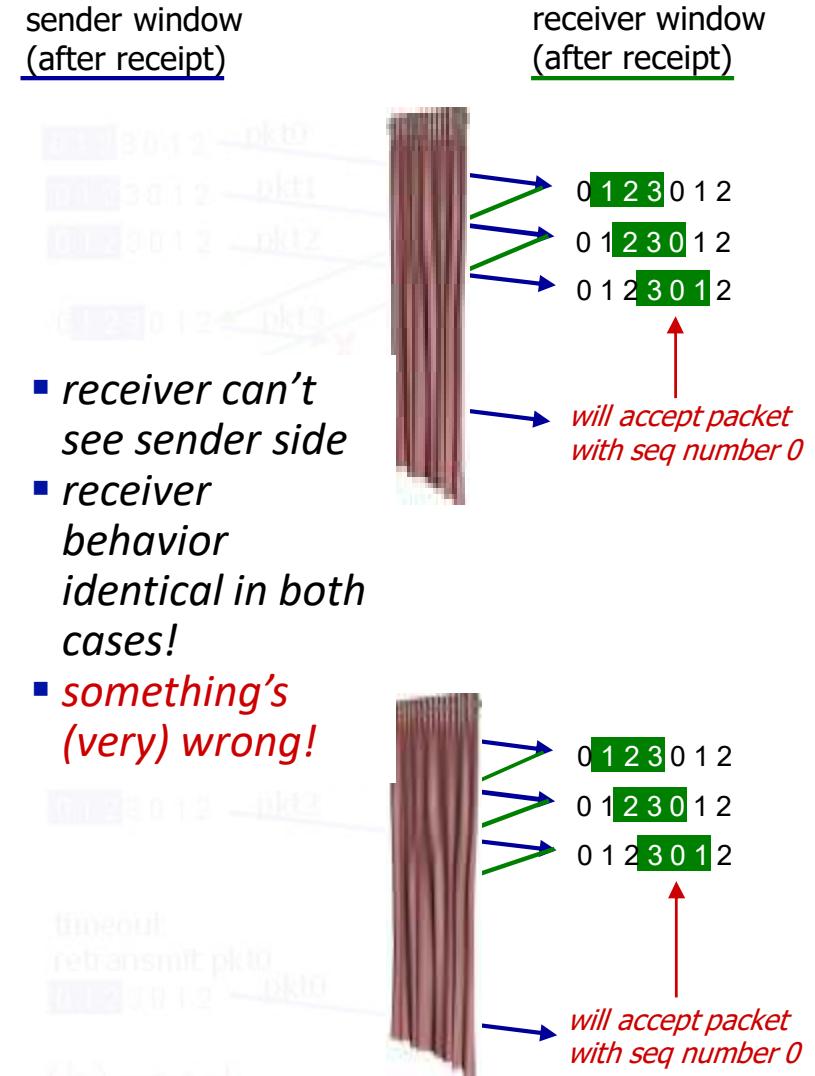


# Selective Repeat: a Dilemma!

Example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3

**Q:** What relationship is needed between sequence # size and window size to avoid problem in scenario (b)?

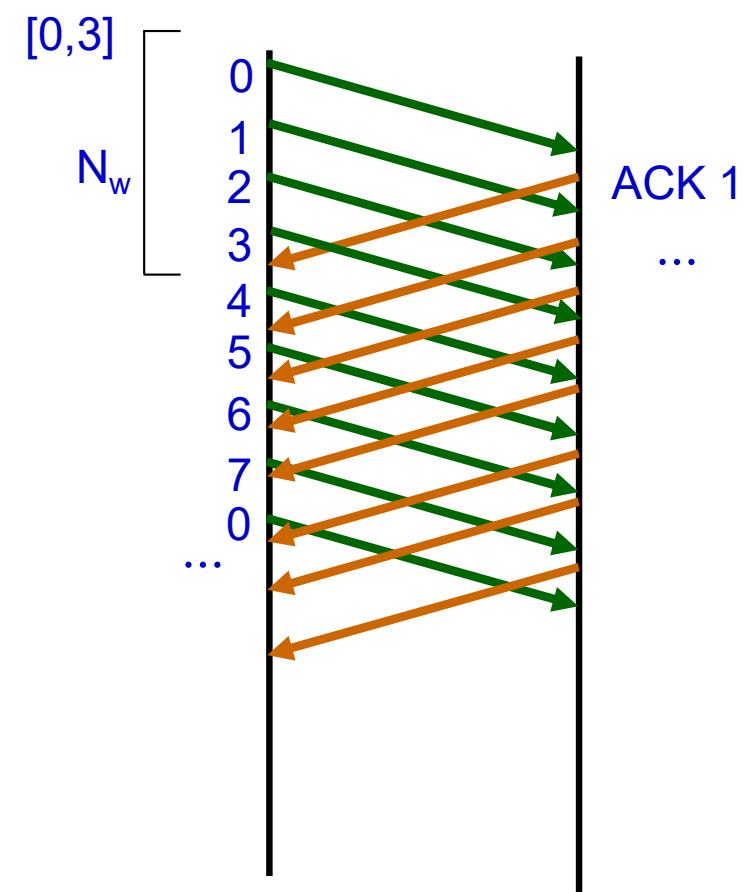


# Sliding Window: Continuous Transmission

## Example: Selective Repeat

- Numbering modulo  $N = 8$   
(Identifiers: 0, 1, ... 7)
- **Max window size:  $N_w = N/2 = 4$**
- Max usage (**efficiency**) is:  
(assuming all bits in the frames are useful bits)  $U = 1$

In this case, it is never needed to stop waiting for an ACK (sender window is never fully used).



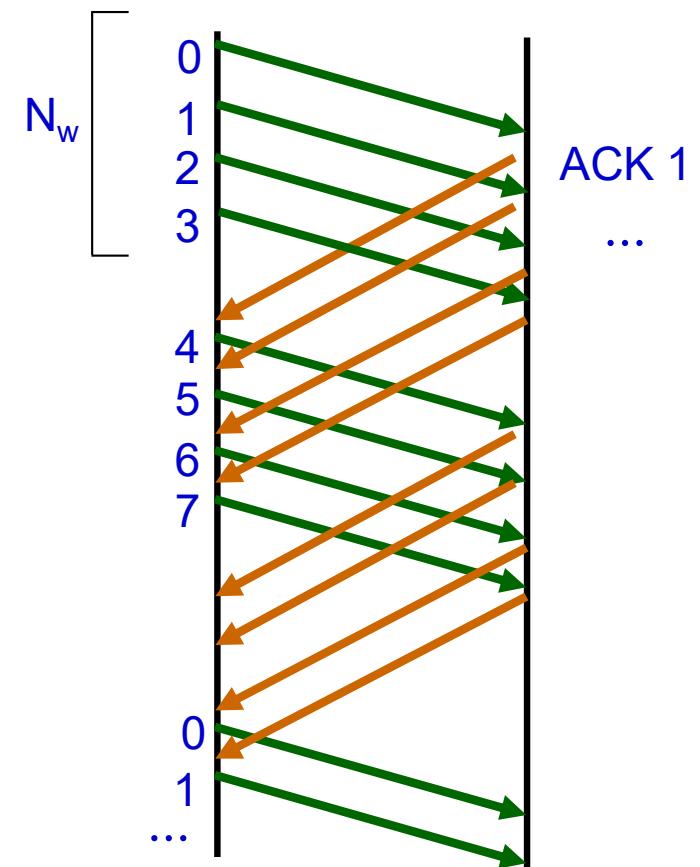
# *Sliding Window: Discontinuous Transmission*

## Example: **Selective Repeat**

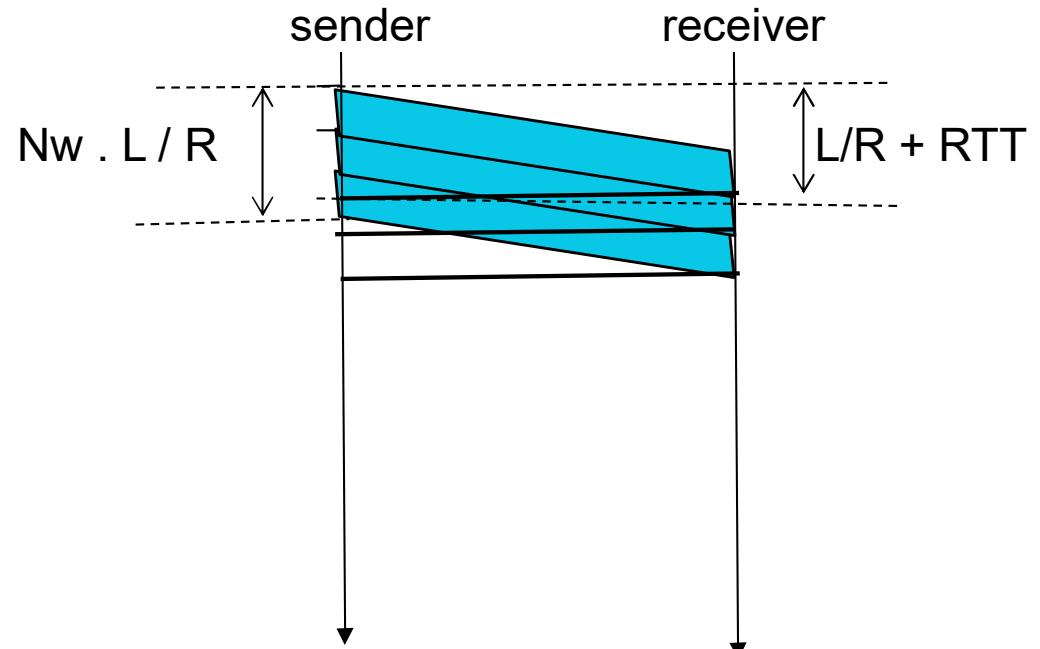
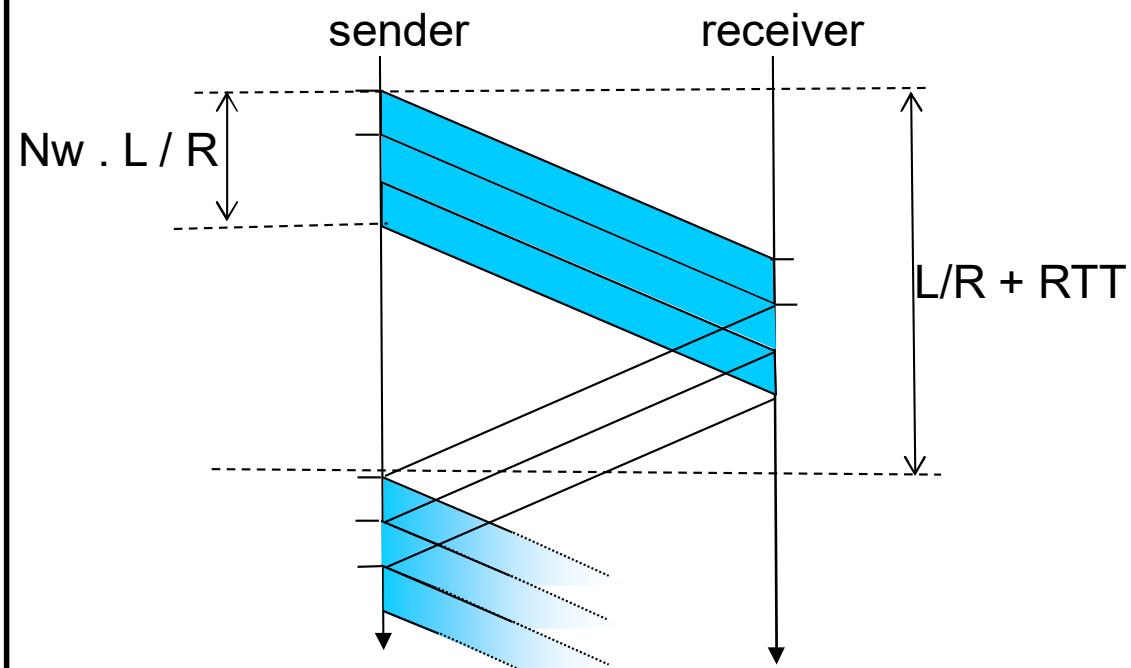
- Numbering modulo  $N = 8$   
(Identifiers: 0, 1, ... 7)
- Max window size:  $N_w = N/2 = 4$
- Assuming the processing and ACK transmission times can be ignored, the usage (**efficiency**) is:

$$U = \frac{N_w \cdot t_{frame}}{2 \cdot t_{prop} + t_{frame}} = \frac{N_w}{2 \cdot a + 1}$$

(assuming all bits in the frames are useful bits)



# Sliding Window Protocols: Utilization



If  $(L/R + RTT) > Nw \cdot L/R$

$$U = \frac{Nw \cdot L/R}{RTT + L / R}$$

else:

$$U = 1$$

# Sliding Window: Efficiency

TPC: Prob. 6 e 7

Continuous transmission:  $U = 1$

Discontinuous transmission:  $U = N_w / (2.a+1)$

**Piggybacking** – improves line usage in bidirectional connections:

- If a station has data and ACKs to send, the ACKs are temporarily delayed and included in the data frames;
- The ACK only uses a few bits in the information frame header, while a separate ACK frame needs the full header and FCS;
- Possible disadvantage: sender may *timeout* if the wait for the *piggybacking* ACK is too long.

## *Flow Control*

ARQ error control techniques also allow to perform flow control, by controlling the cadence of ACK transmission.



## *Chapter 3 (part 1): Summary*

- Principles behind transport layer services:
  - Multiplexing and demultiplexing;
  - Reliable data transfer;
  - Flow control;

### Next:

- Leaving the network “edge” (application, transport layers);
- Into the network “core”.

# Redes de Computadores

## LEIC-A, MEIC-A

### 4 – *Network Layer*

Prof. Paulo Lobato Correia  
*IST, DEEC – Área Científica de Telecomunicações*

# Network Layer

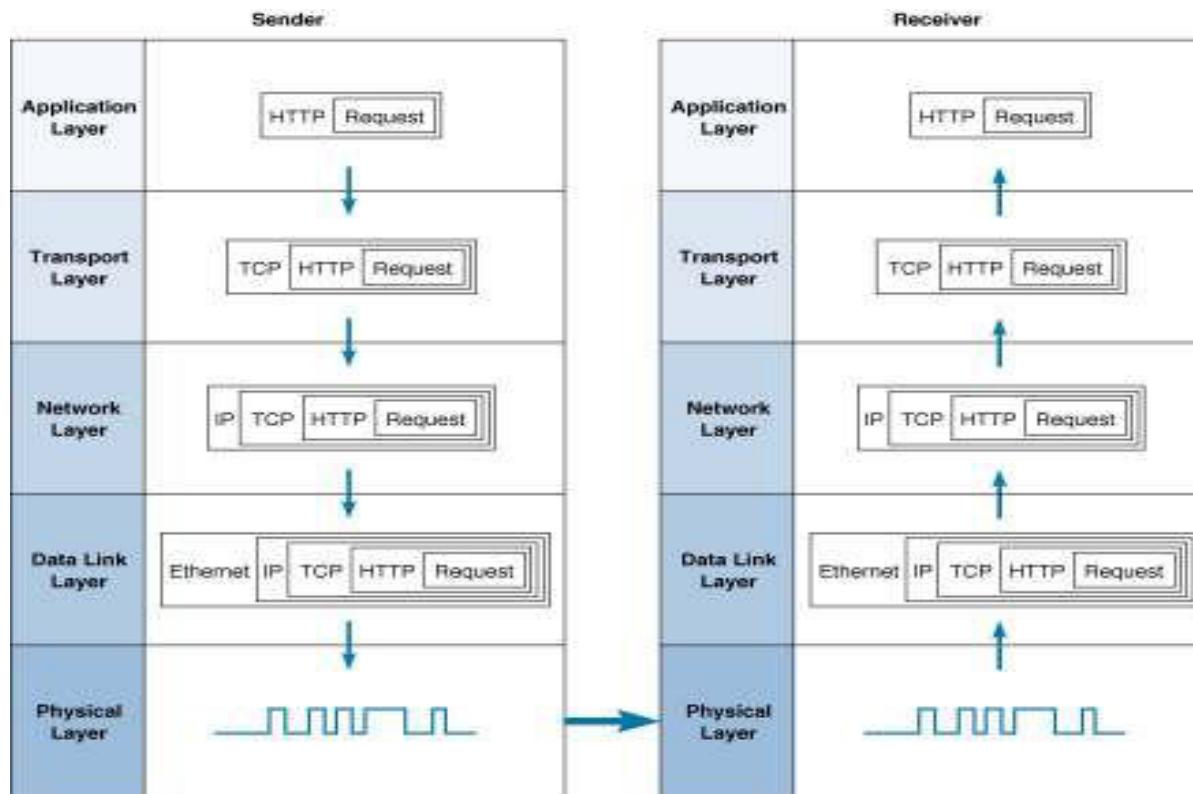
Application  
layer

GET /index.html HTTP/1.1  
Host: www.tejo.tecnico.ulisboa.pt

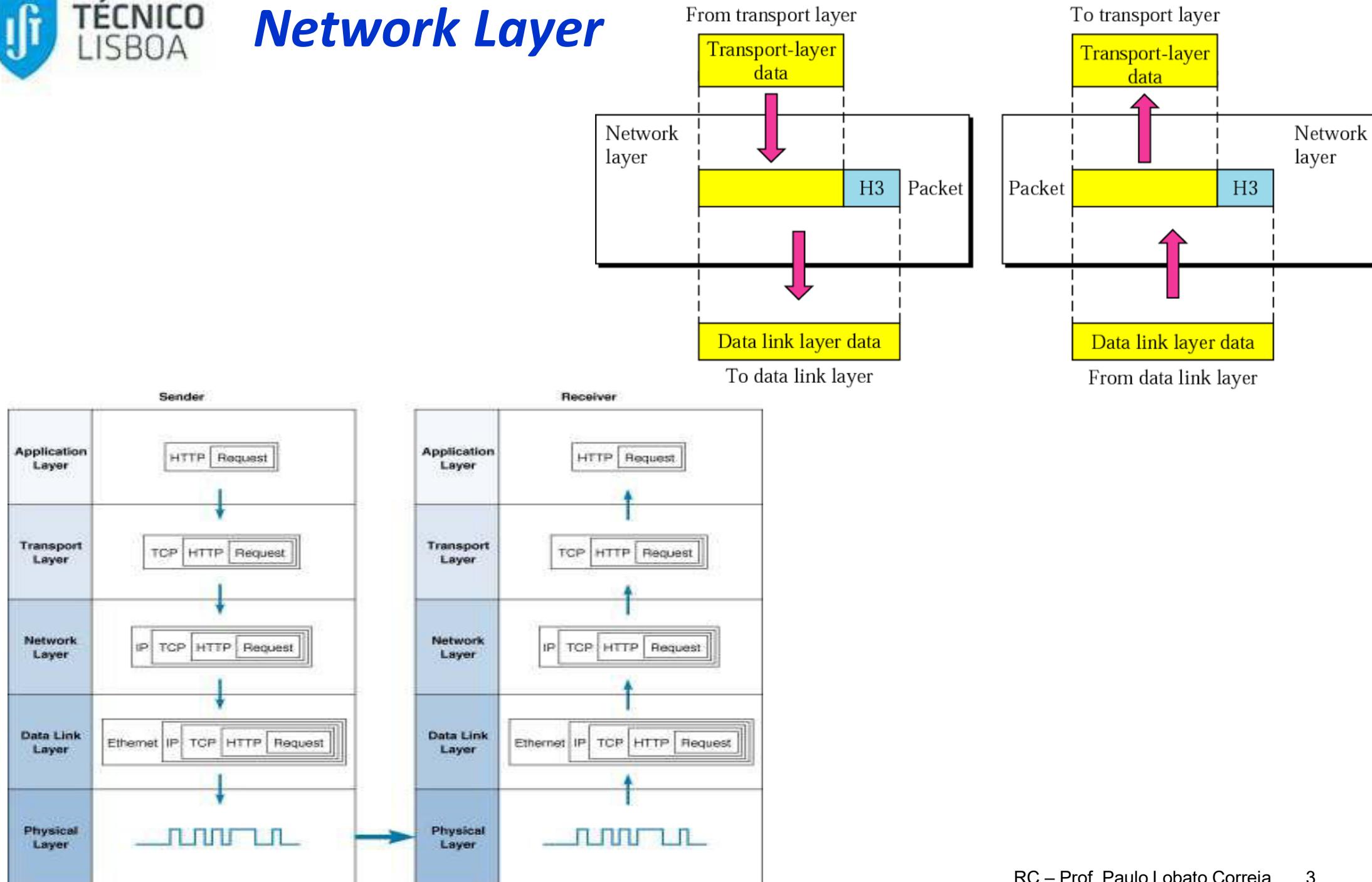
Transport  
layer

TCP Header | GET /index.html H  
TCP Header | TTP/1.1  
Host: www

TCP Header | .tejo.tecnico.ulisboa.pt

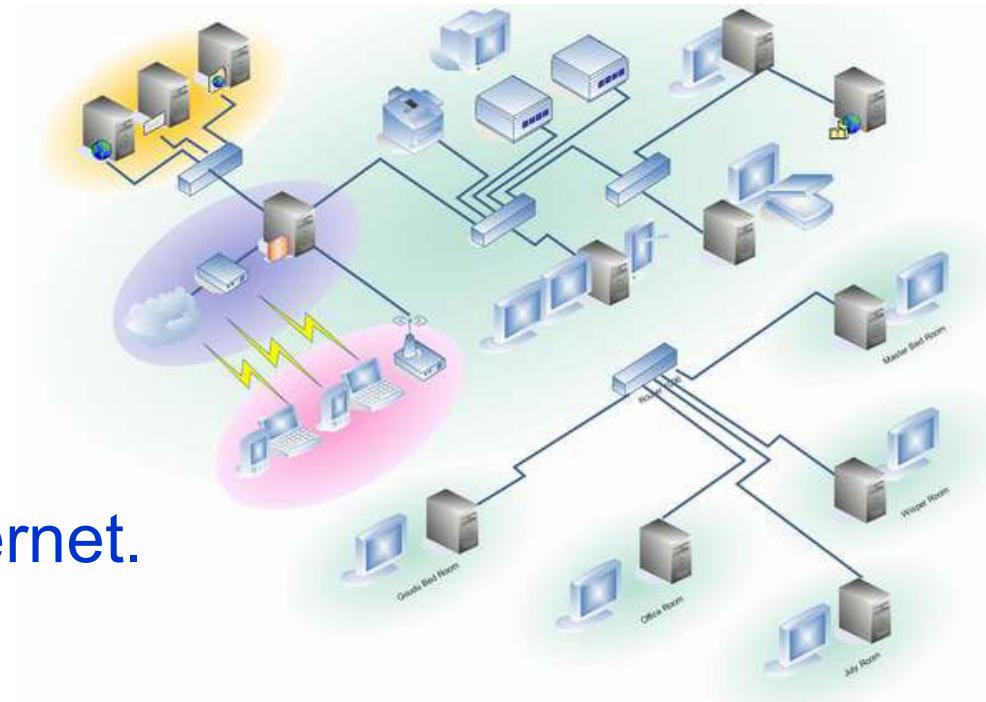


# Network Layer



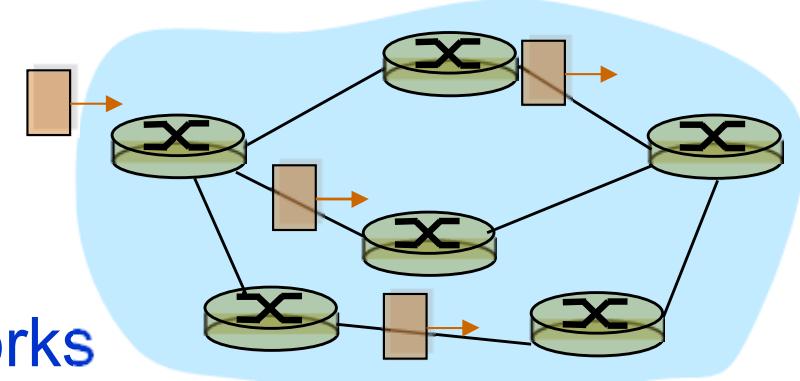
# Objectives

- Understand principles behind network layer services:
  - Network layer service models;
  - Forwarding versus routing;
  - Routing (path selection);
  - Dealing with scale.
- The network layer of the Internet.



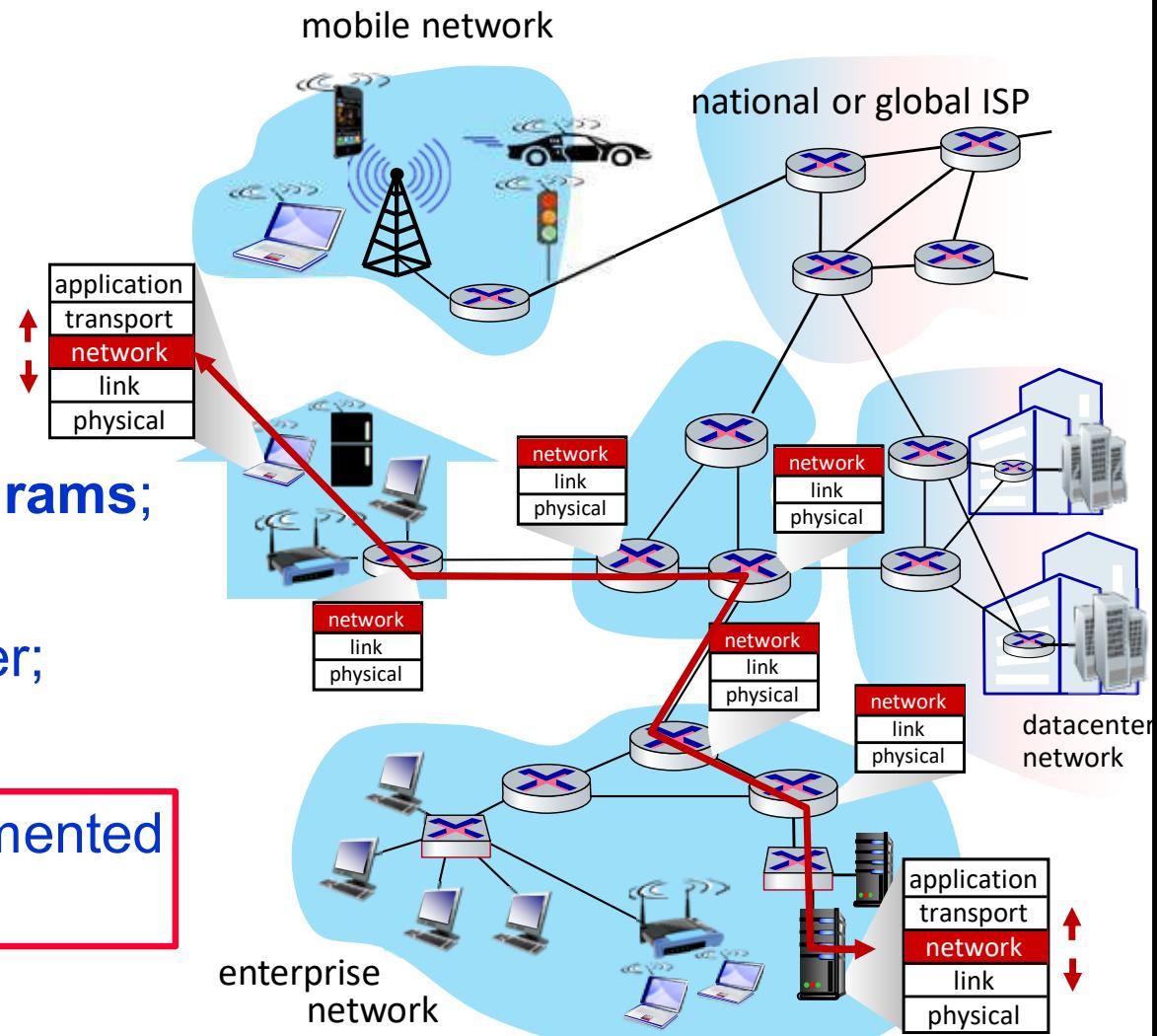
# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# Network Layer

- Delivers segments from sending to receiving host;
- Sending side: encapsulates segments into **datagrams**;
- Receiving side: delivers segments to transport layer;
- Network layer protocols are implemented in every host and router;
- Routers examine header fields in all IP datagrams passing by.



# Network Layer: Key Functions

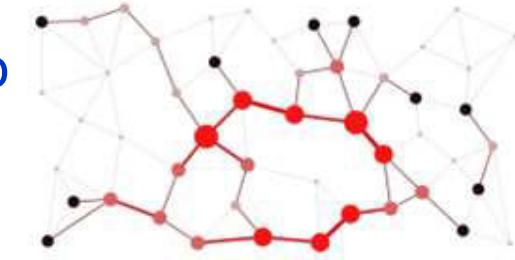
- *Routing:*

Determine route taken by packets from source to destination:

- *Routing algorithms.*

- *Forwarding:*

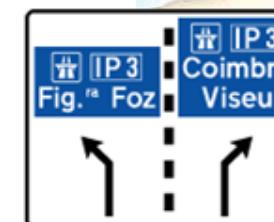
Move packets from router's input to appropriate router output.



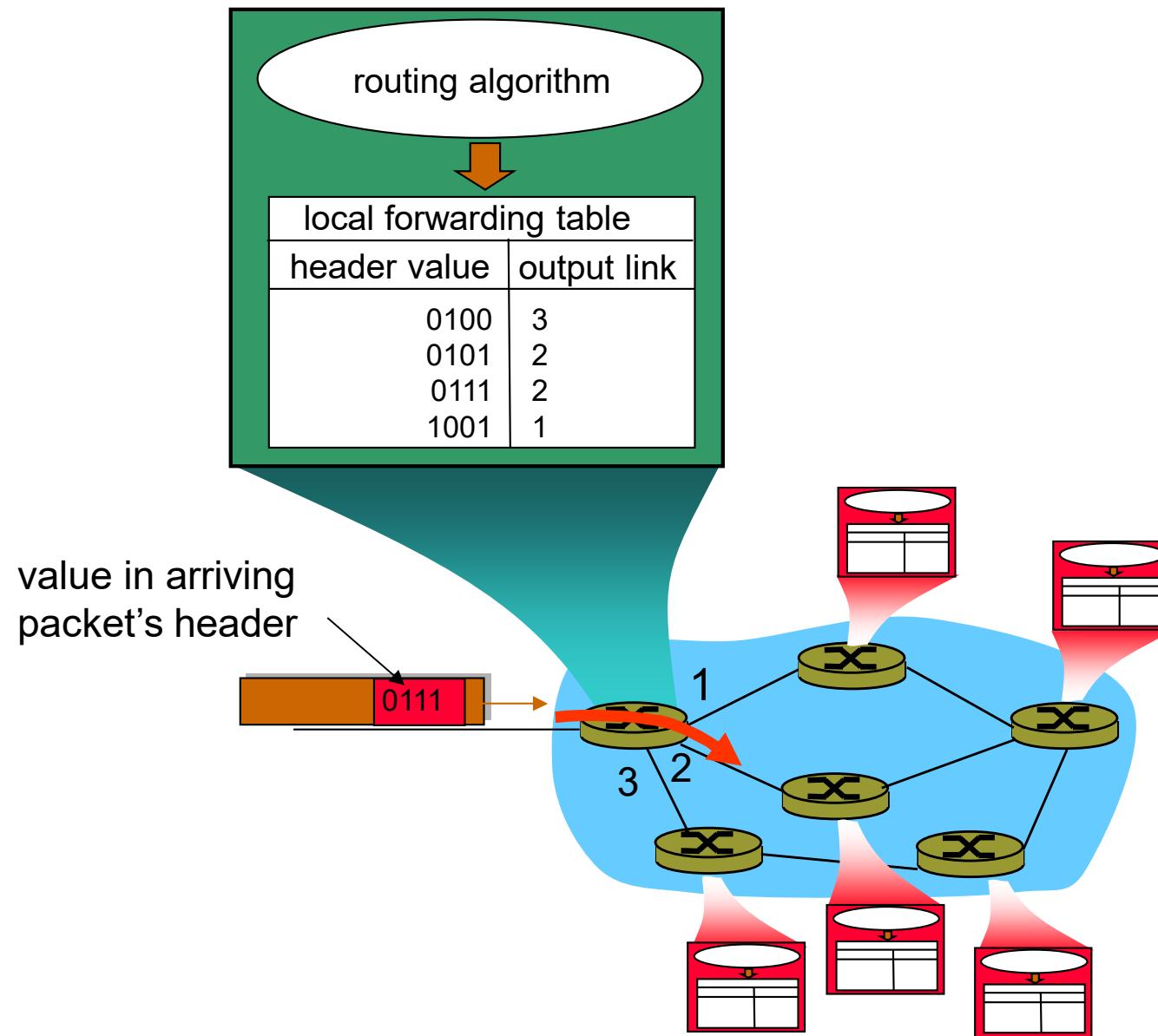
## Analogy:

- *Routing:* process of planning trip from source to destination;

- *Forwarding:* process of getting through a single interchange.



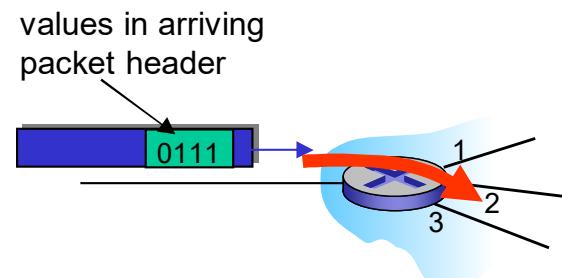
# Interplay between Routing and Forwarding



# Network Layer: *Data Plane + Control Plane*

## Data plane:

- **Local**, per-router function
- **Forwarding** implementation



## Control plane

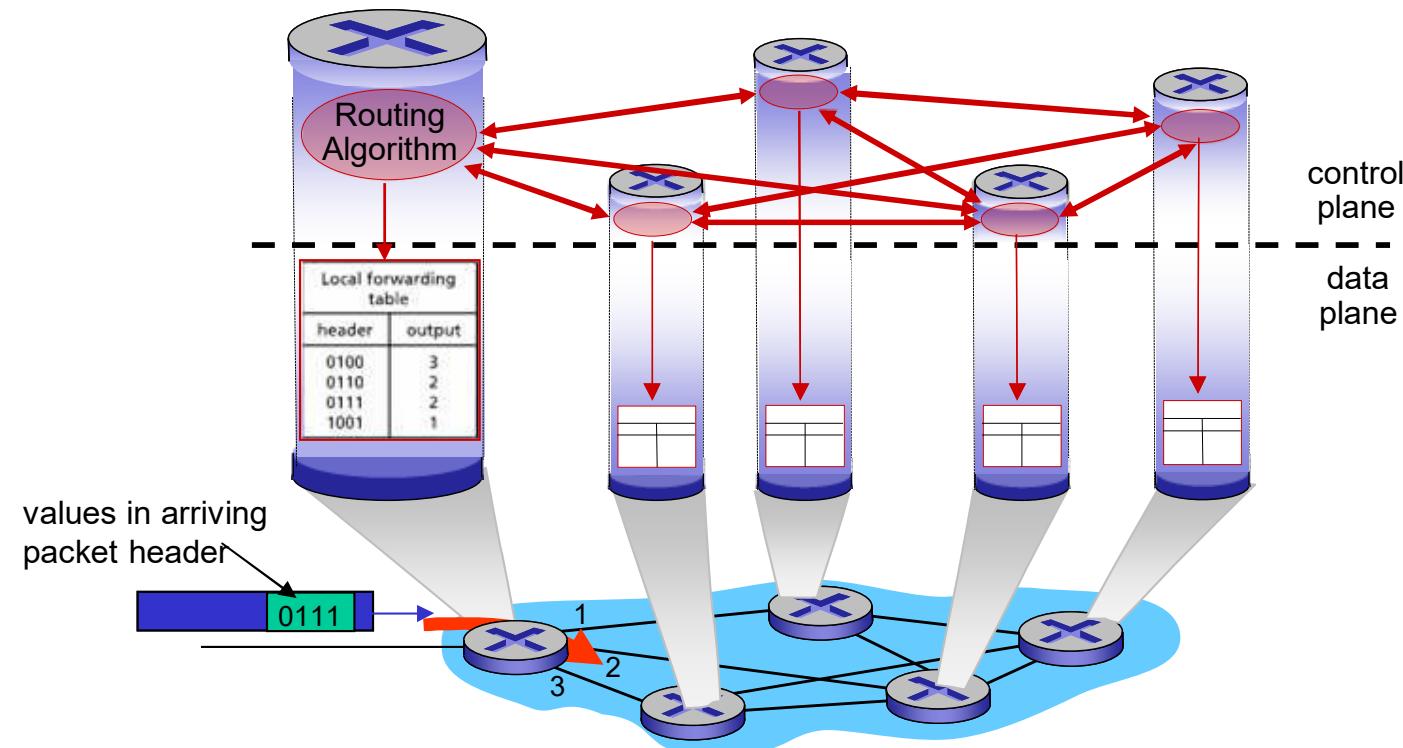
- **Network-wide logic**
- **Routing** implementation

- Two control-plane approaches:

- **Traditional routing algorithms:** implemented in routers
- **Software-defined networking (SDN):** implemented in (remote) servers

# Traditional Routing Algorithms (per-router Control Plane)

Individual routing algorithm components *in each and every router* interact in the control plane

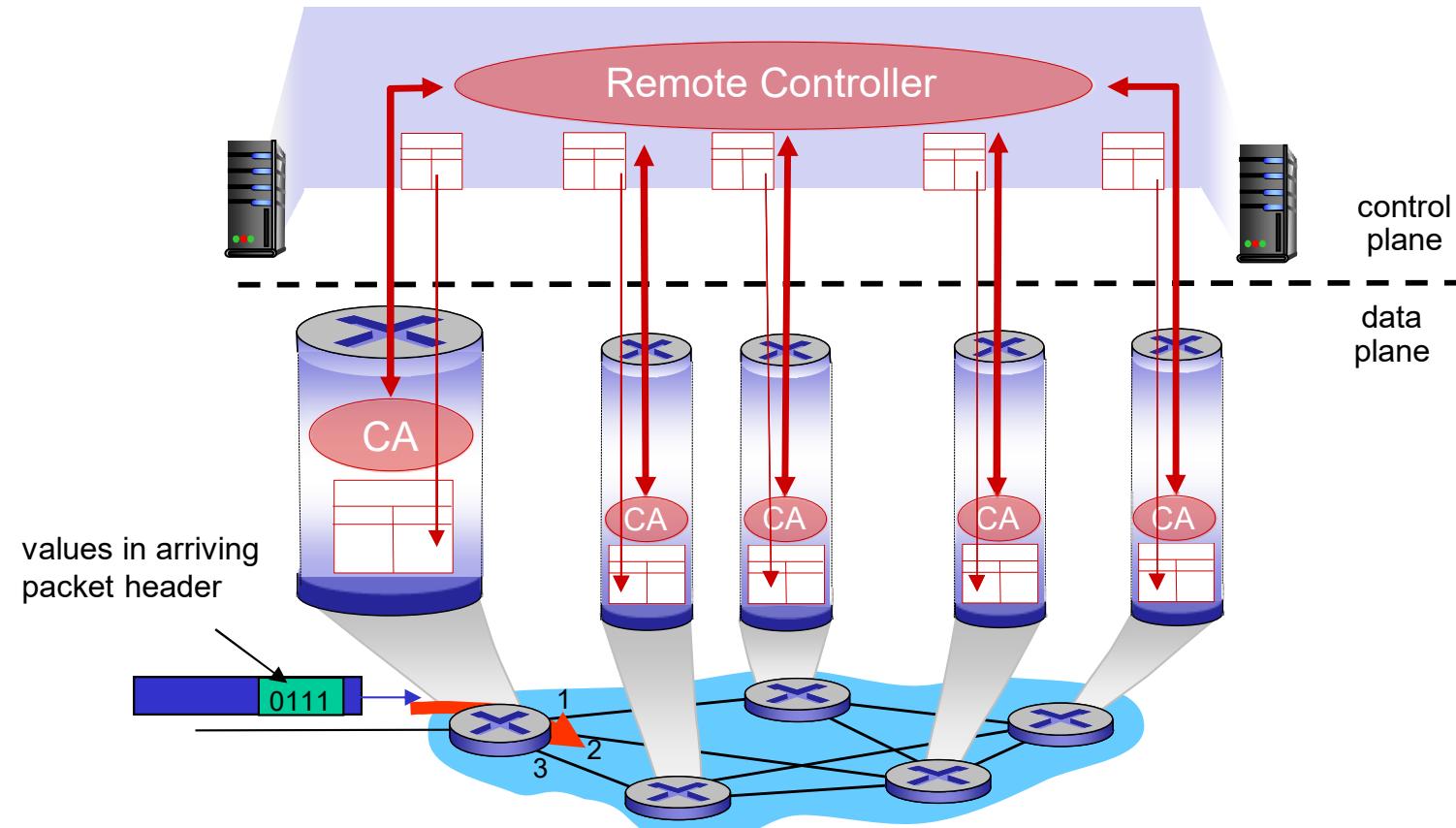


Traditional router includes control and data planes.

# Software-Defined Networking (SDN)

## Control Plane

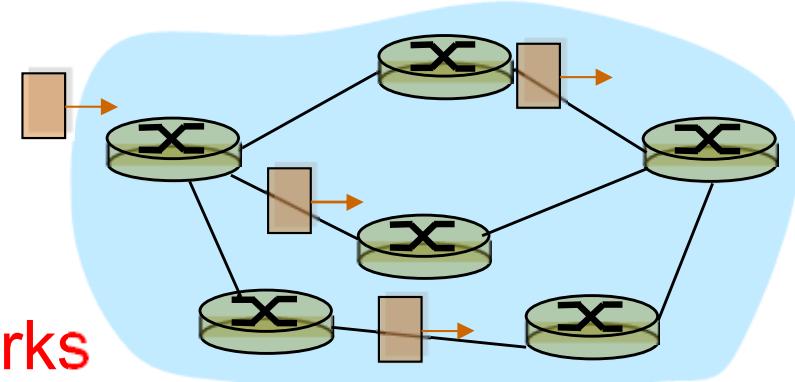
Remote controller computes and installs forwarding tables in routers



SDN (software defined network) router separates control plane (remote) from data plane (local).

# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# *Network Layer*

## *Connection and Connectionless Services*

- **Datagram** network provides network-layer *connectionless* service (e.g., Internet);
- **Virtual Circuit** (VC) network provides network-layer *connection oriented* service (e.g., X.25);
- Analogous to the transport-layer services, but:
  - Service: host-to-host (not end-to-end);
  - **No choice**: network provides **one or the other**;
  - Implementation: in network core.

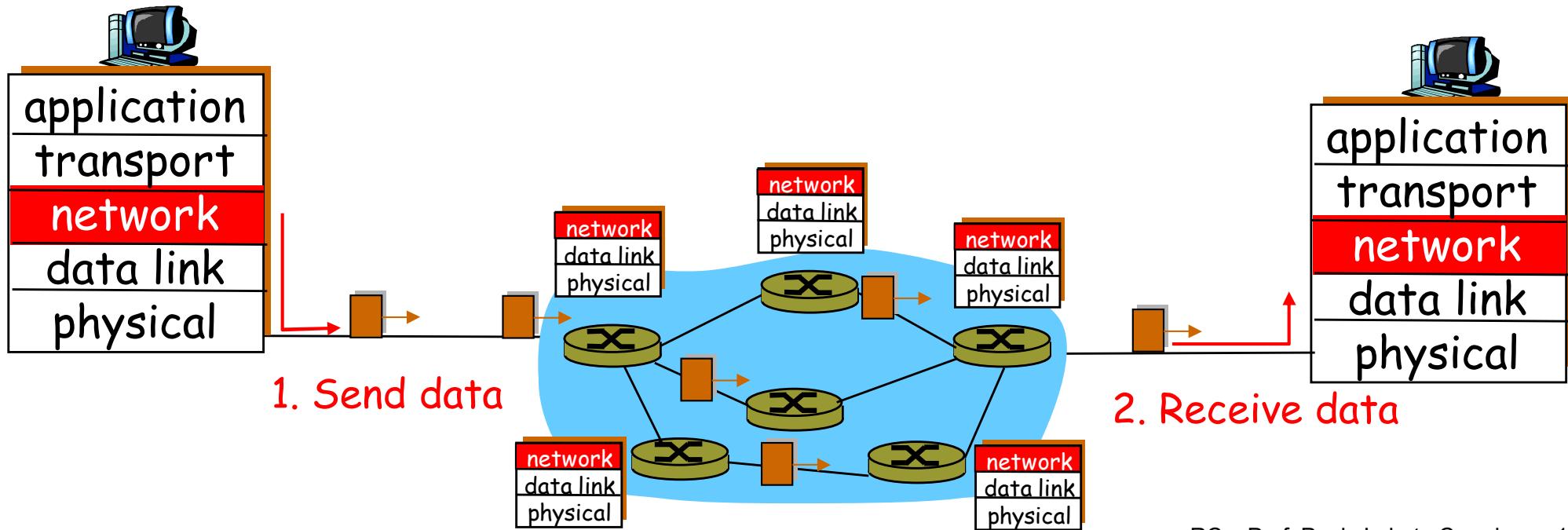
## Virtual Circuits

“Source-to-destination path behaves much like a telephone circuit”

- Performance-wise;
  - Network actions along source-to-destination path.
- 
- Call setup for each call *before* data can flow;
  - Each packet carries a VC identifier  
(and not the destination host address);
  - **Every router** on source-destination path **maintains “state”** for each passing connection;
  - Link and router resources (bandwidth, buffers) may be *allocated* to VC  
**(dedicated resources** = predictable service).

# Datagram Networks

- No call setup at network layer;
- Routers: don't keep state information about end-to-end connections:
  - **No network-level concept of “connection”;**
- Packets forwarded using destination host address:
  - **Packets between same source-destination pair may take different paths.**



# Datagram or VC Network: Why?

## Internet (Datagram):

- Data exchange among computers: “elastic” service, no strict timing reqs;
- “Smart” end systems (computers) can adapt, perform error recovery, ...
- Simple inside network, complexity at the “edge”;
- Many different link types: difficult to offer uniform service.

## ATM (Virtual Circuit):

- Evolved from telephony, which required strict timing, posed reliability requirements and needed a guaranteed service;
- Uses “dumb” end systems: the telephones;
- Complexity is inside the network.

# Network Layer Service Models

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)

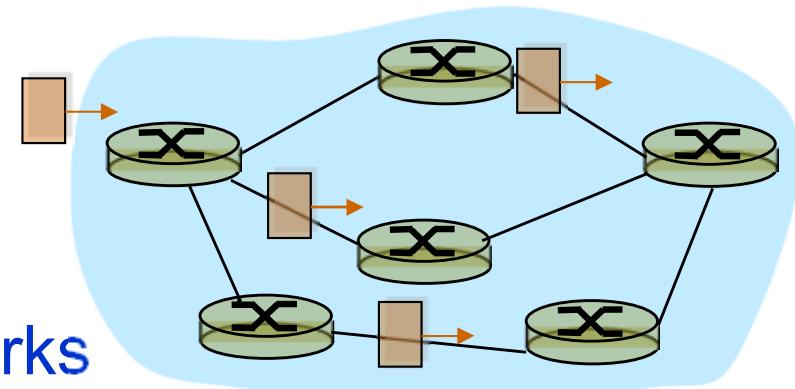
Internet “best effort” service model

**No** guarantees on:

- i. successful datagram delivery to destination
- ii. timing or order of delivery
- iii. bandwidth available to end-end flow

# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# IP Addressing



At the network layer each station must be uniquely identified to allow global communication among any pair of stations connected to the Internet.

**IP addresses should be unique and universal.**

**IP v4** address (RFC 760):

- Composed by 4 bytes (**32 bits**);
- It is usual to represent IP addresses using decimal notation, to ease human reading (e.g.: 193.136.128.1);
- Stations and routers manipulate binary addresses.

**1001000100111... (32 bits)**  
**(Used by hosts and routers)**

**193.136.128.1**  
**(For human reading)**

# IPv4 Addressing Space



The **addressing space** is the total number of available addresses.

N bit addresses provide  $2^N$  values;

With 32 bit addresses, the Internet IPv4 addressing space contains  $2^{32} = 4\,294\,967\,296$  addresses.

Without other restrictions more than 4000 million devices could be connected to the Internet using IPv4.

Decimal numbering (base 10):       $2022_{(10)}$

$$2022 = 2 \times 1000 + 0 \times 100 + 2 \times 10 + 2 \times 1$$

$$10^3=1000, 10^2=100, 10^1=10, 10^0=1$$

Binary numbering (base 2):       $1011\,0110_{(2)}$      $\rightarrow 182_{(10)}$

$$182 = 1 \times 128 + 0 \times 64 + 1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$$

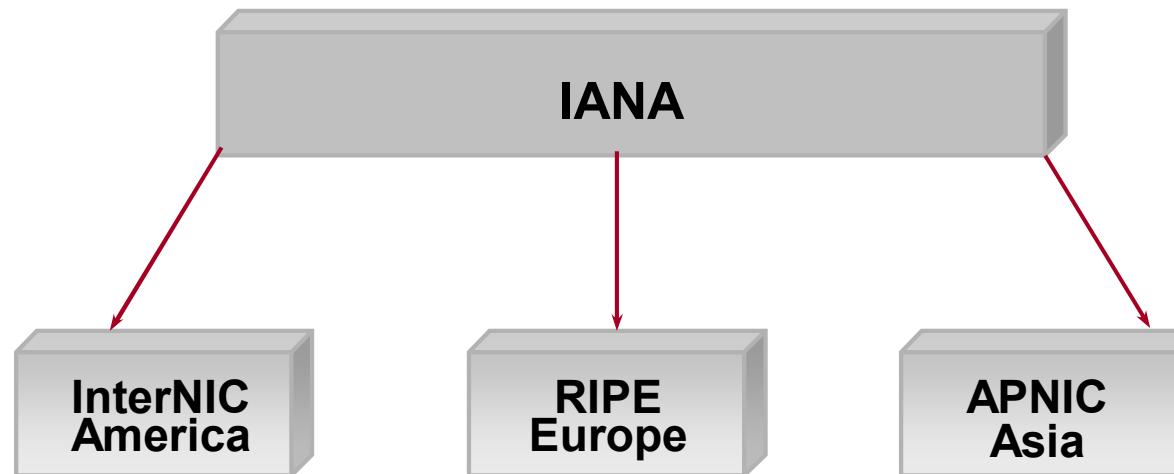
$$2^7=128, 2^6=64, 2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1$$

# IP Addressing

**Q:** How does an ISP get a block of addresses?

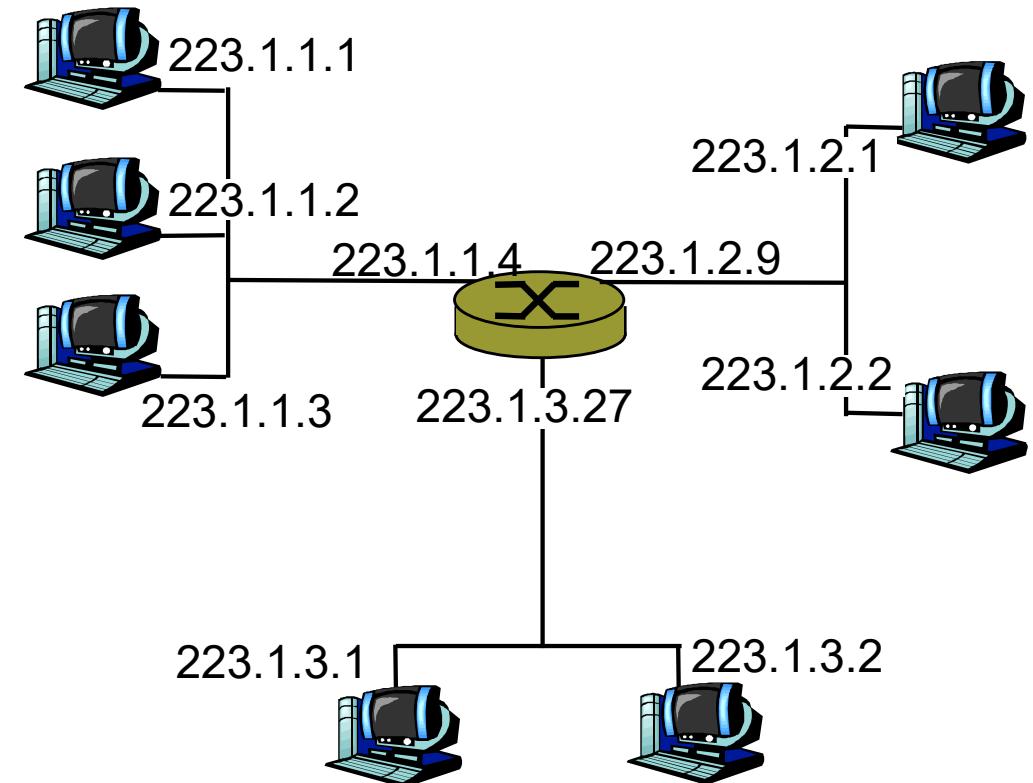
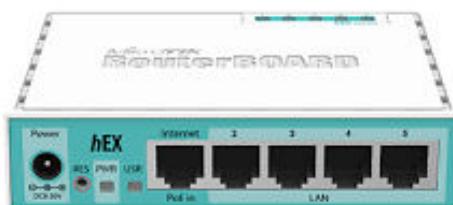
**A:** ICANN: Internet Corporation for Assigned Names and Numbers

- Allocates addresses through IANA: Internet Assigned Numbers Authority;
- Manages DNS;
- Assigns domain names, resolves disputes.



# IP Addressing

- IP address: 32-bit identifier for host and router *interfaces*.
- *Interface*: connection between host/router and a physical link:
  - Router's typically have multiple interfaces;
  - Host typically has one interface;
  - **IP addresses are associated with each interface.**



$223.1.1.1 = \underline{11011111} \underline{00000001} \underline{00000001} \underline{00000001}$

223      1      1      1

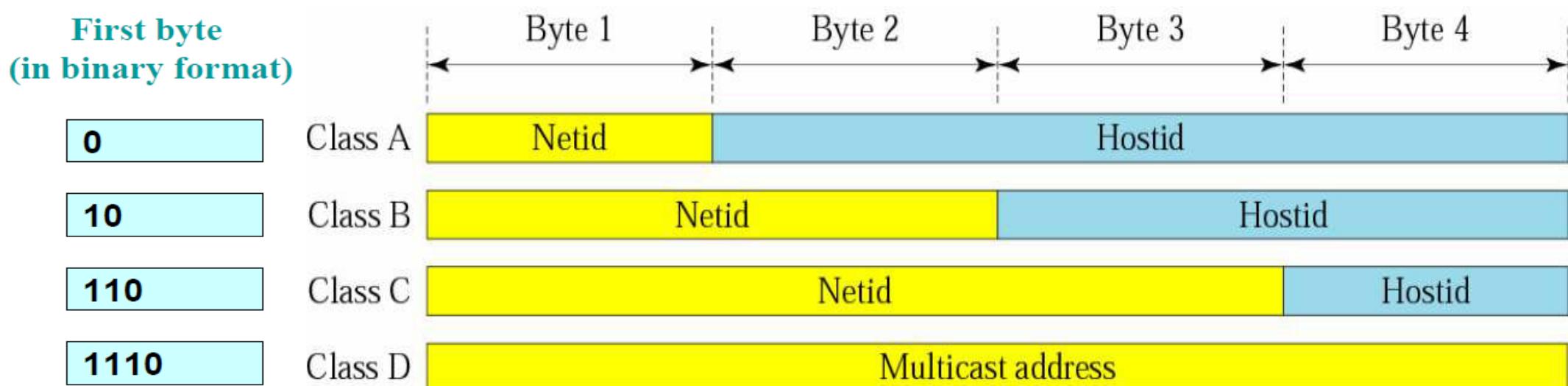
# IP Addressing

IP addresses have **two components**:

- Network identification (**NetID**);
- Station's interface identification (**HostID**);

Stations with the same network component communicate directly;

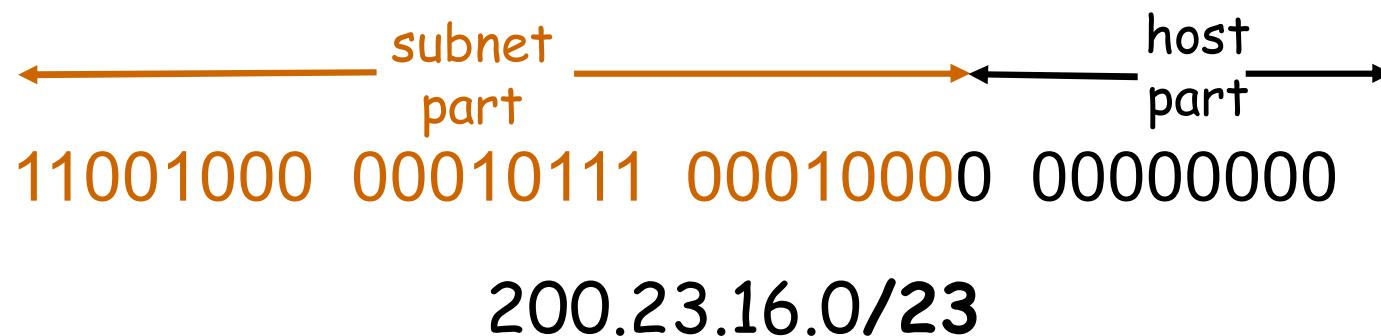
Stations with different network components communicate through routers;



# IP Addressing: CIDR

## CIDR: Classless InterDomain Routing

- Subnet portion of address of arbitrary length;
- Address format:
  - $a.b.c.d/x$ ,  
where  $x$  is the number of bits in the subnet portion of the address.

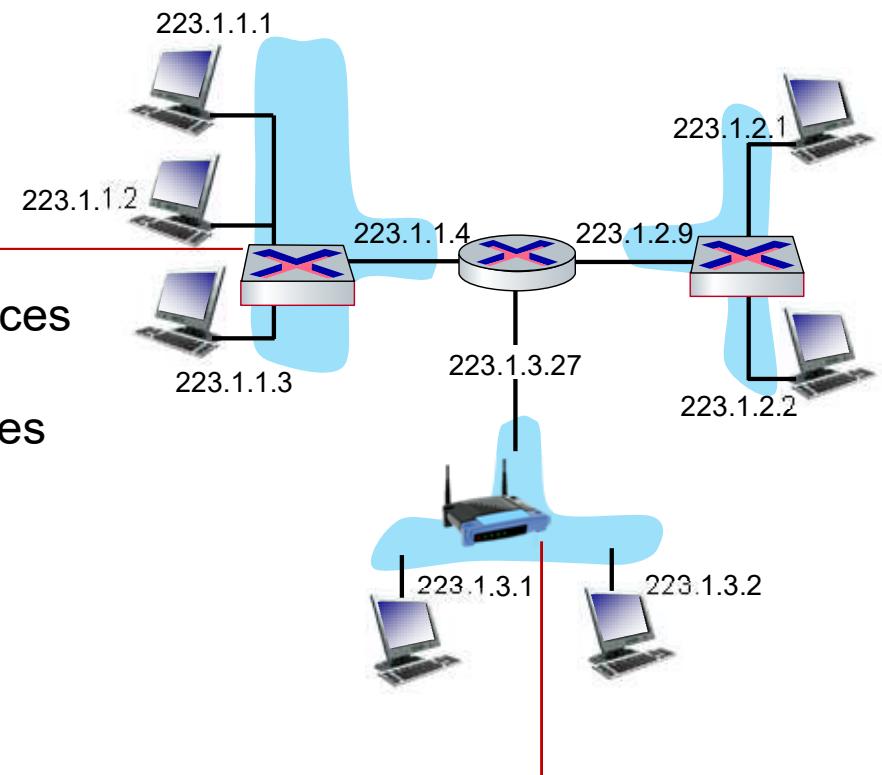


# Subnets

**Q:** How are interfaces actually connected?

**A:** We'll learn about that in chapters 6, 7

**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

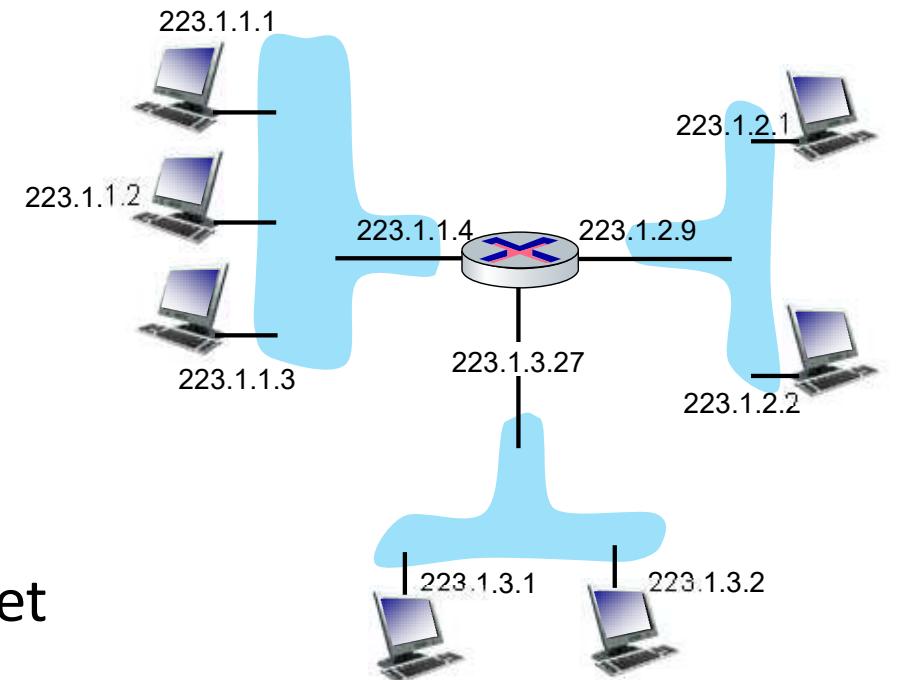
# Subnets

## ■ What's a subnet ?

- device interfaces that can physically reach each other **without passing through an intervening router**

## ■ IP addresses have structure:

- **subnet part:** devices in same subnet have common high order bits
- **host part:** remaining low order bits

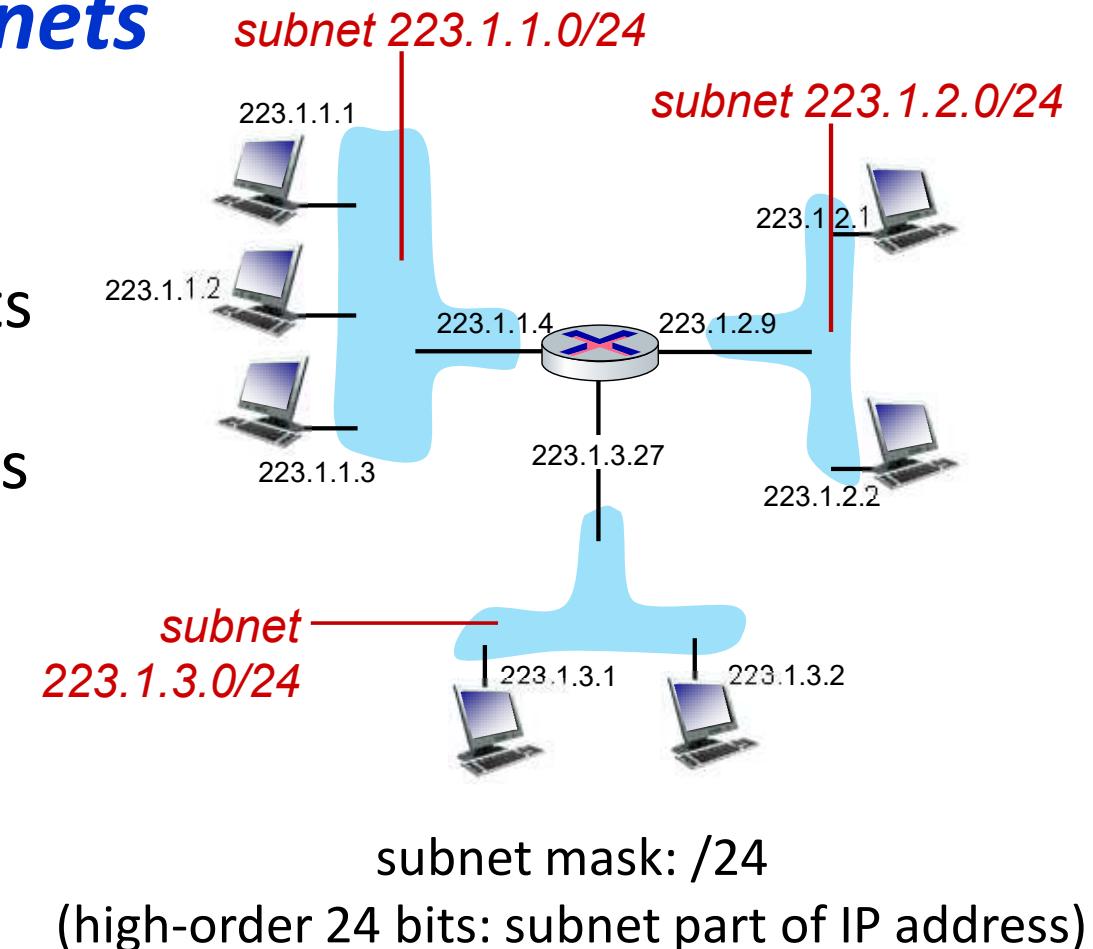


network consisting of 3 subnets

## Subnets

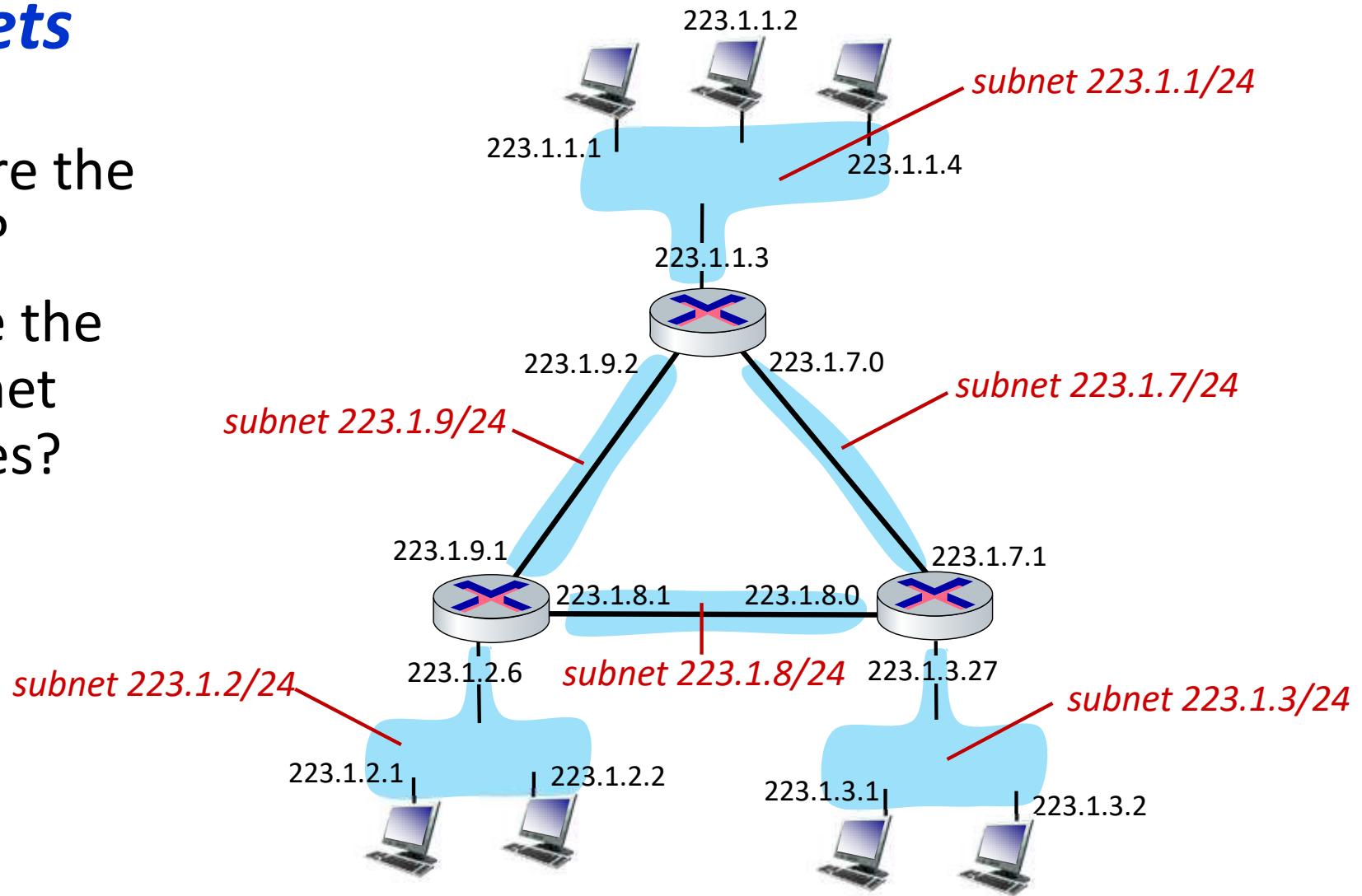
*Recipe for defining subnets:*

- detach each interface from its host or router, creating “islands” of isolated networks
- each isolated network is called a *subnet*



## Subnets

- where are the subnets?
- what are the /24 subnet addresses?



## IP v4 Addresses

Addresses beginning with 127 are special:

- They are reserved to reference the station itself;
- 127.0.0.1 – **localhost**

Addresses with all “station” bits set to 0 (zero) :

- Represent the **network address**;
- Example: the address 193.136.128.41 belongs to the class C network 193.136.128.**0/24**

Addresses with all “station” bits set to 1 (one) :

- Represent a **broadcast address**;
- Example: 193.136.128.**255**

## IP Addressing: Subnet Part

Q: How does the *network* get the subnet part of the IP address?

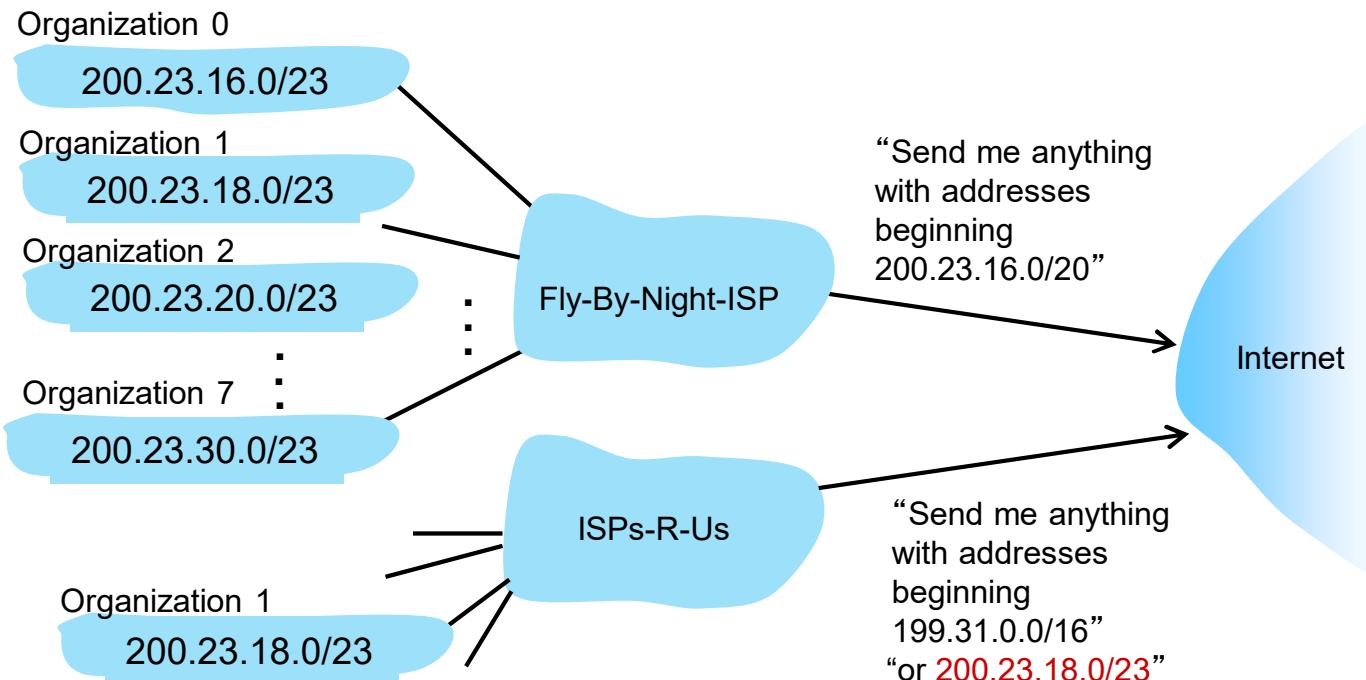
A: It gets allocated a portion of its provider ISP's address space.

ISP's block	<u>11001000 00010111 00010000 00000000</u>	200.23.16.0/20
Organization 0	<u>11001000 00010111 00010000 00000000</u>	200.23.16.0/23
Organization 1	<u>11001000 00010111 00010010 00000000</u>	200.23.18.0/23
Organization 2	<u>11001000 00010111 00010100 00000000</u>	200.23.20.0/23
...	.....	....
Organization 7	<u>11001000 00010111 00011110 00000000</u>	200.23.30.0/23

# Hierarchical Addressing: *Route Aggregation*

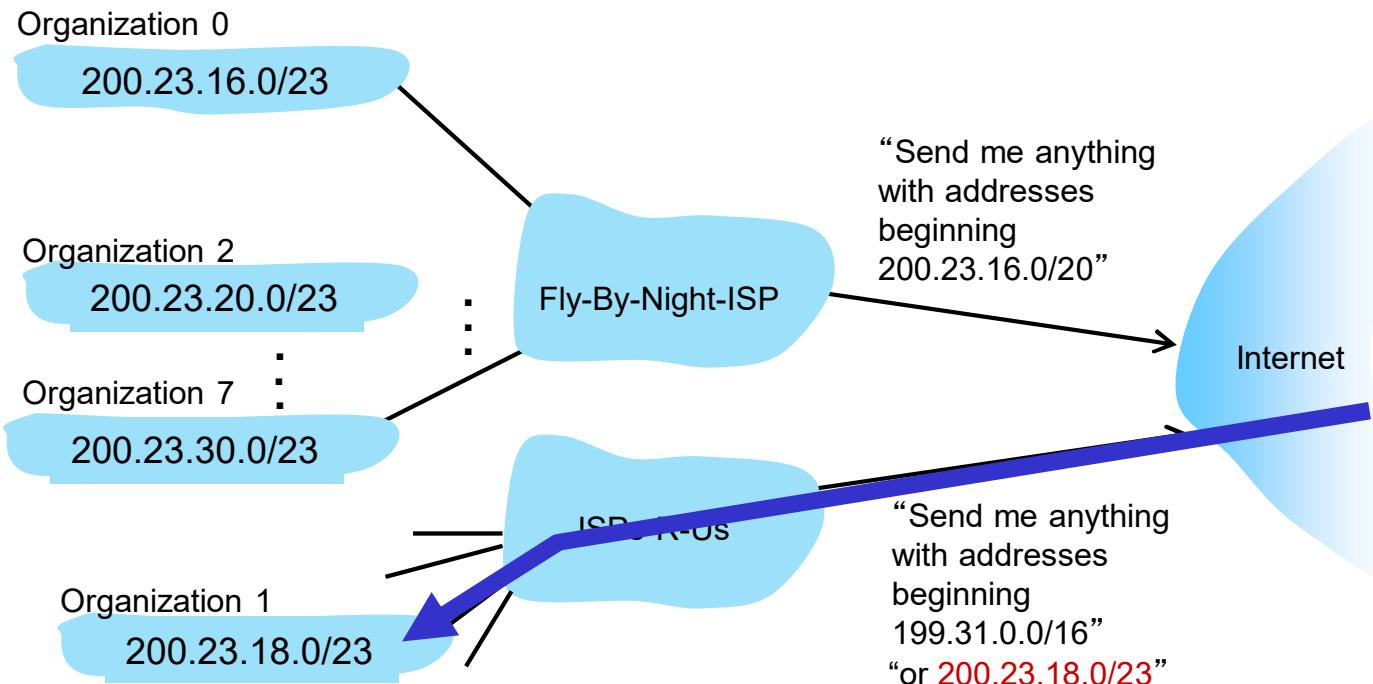
Hierarchical addressing allows efficient advertisement of routing information

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



# Hierarchical Addressing: Route Aggregation

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



# Datagrams – Forwarding Table

## Longest Prefix Matching

<u>Prefix Match</u>	<u>Link Interface</u>
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
otherwise	3

### Examples

Dest.Address: 11001000 00010111 00010110 10100001      Which interface?

Dest.Address: 11001000 00010111 00011000 10101010      Which interface?

# Addressing



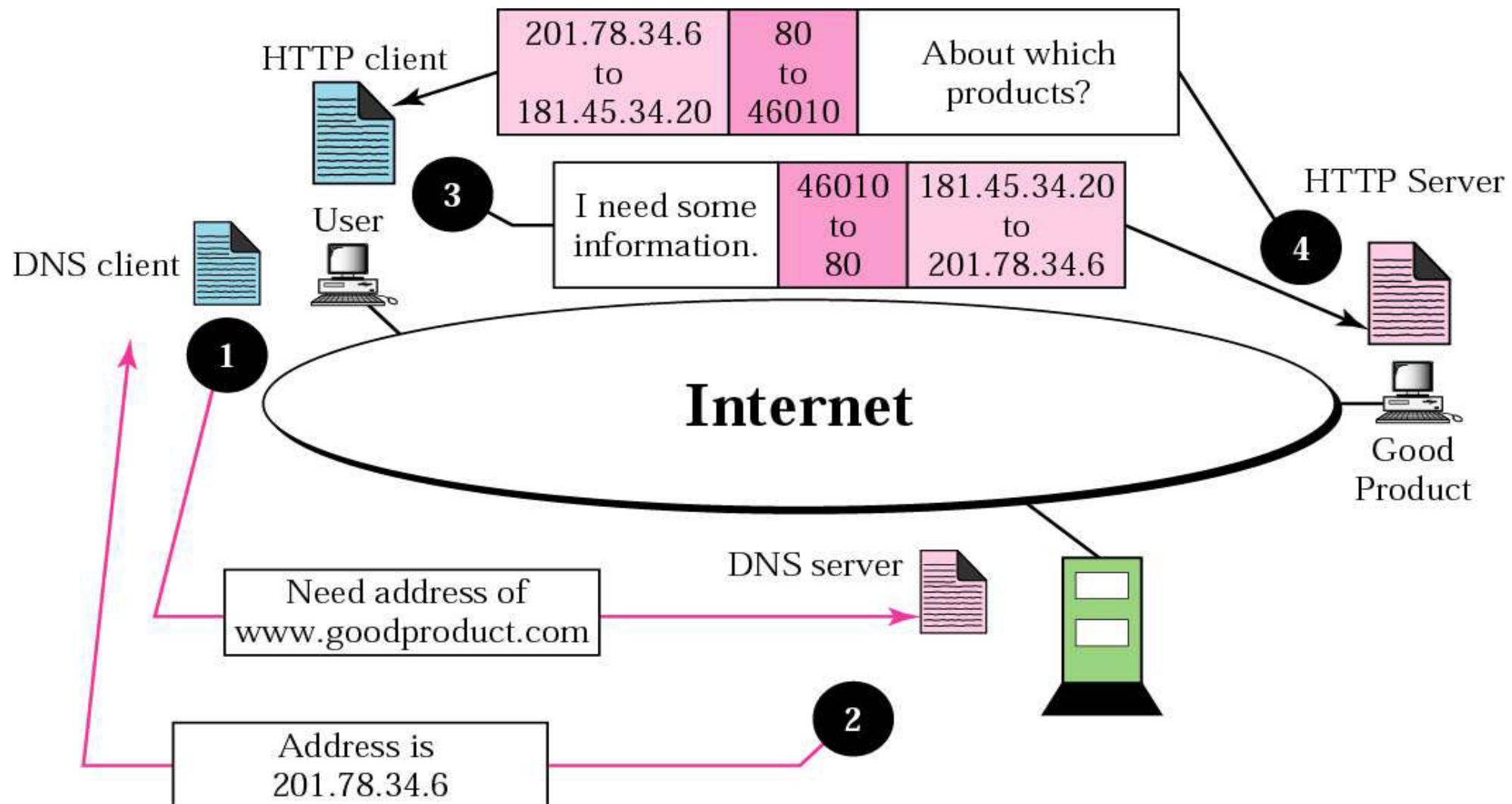
So far we have seen the usage of **3 types of addresses**:

- **Application** layer address (e.g.: www.tecnico.ulisboa.pt – web server);
- **Transport** layer address (e.g.: ports 52132 and 80 – web client and server ports, respectively);
- IP address at the **network** layer address (e.g.: 193.136.222.20 and 193.136.128.1 – origin and destination, respectively);

The user only knows the first type of address (application server), but for packets to be transmitted the source and destination ports and IP addresses need to be known.

- Destination port is specific of the application in usage (e.g.: 80 for HTTP);
- Origin port number is temporarily assigned by the station, which also knows its IP address;
- Destination IP address is obtained from the application layer address using the DNS (*Domain Name System*).

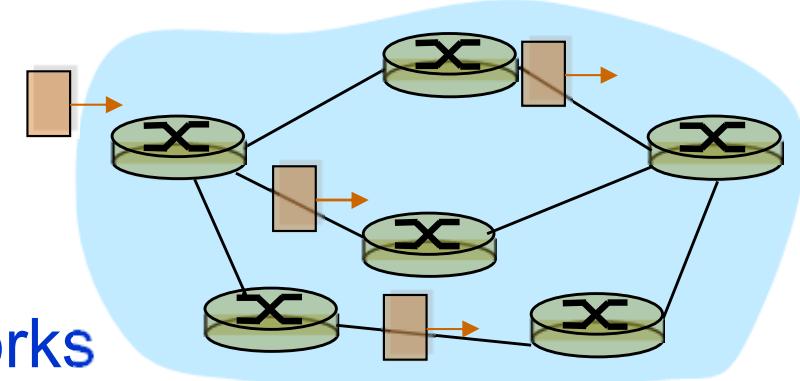
# Addressing: Example





# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and **forwarding tables**
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# *Forwarding Tables*

Row	Network/ Subnet	Mask (/Prefix)	Metric (Cost)	Interface	Next Router
1	128.171.0.0	255.255.0.0 (/16)	47	2	G
2	172.30.33.0	255.255.255.0 (/24)	0	1	Local
3	192.168.6.0	255.255.255.0 (/24)	12	2	G

Usually, forwarding tables do not contain one entry for each host: their size would be huge.

It is enough to include the destination network address.

That network is then responsible to deliver the message to the destination host (e.g., by diffusion).

# Forwarding Tables: Masks

## 1. Masking

Information	1	1	0	0
Mask	1	0	1	0
Result	1	0	0	0

## 2. Usual Values

Binary	Decimal
00000000	0
11111111	255

## 3. Example 1

IP Address	172.	30.	22.	7
Mask	255.	0.	0.	0
Result	172.	0.	0.	0

## 4. Example 2

IP Address	172.	30.	22.	7
Mask	255.	255.	0.	0
Result	172.	30.	0.	0

To check to which network an address belongs, a binary mask is applied (logical “**AND**”), to remove the host component of the IP address.

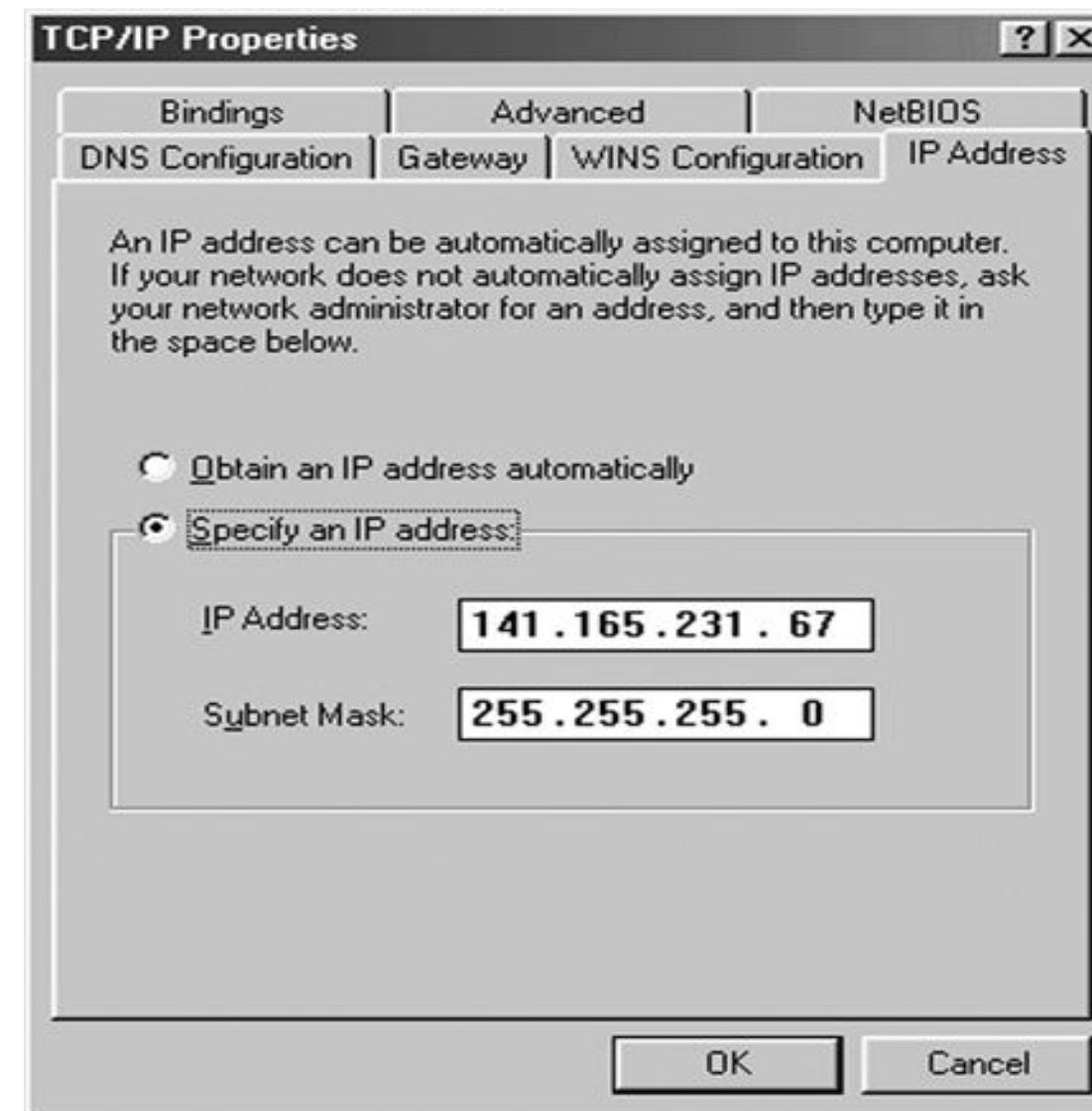
## Forwarding Tables: Masks

Example of applying a binary mask to the class C IP address: 234.136.25.50 to identify the network and host components.

IP Address	
11101010.10001000.00011001.00110010	
Subnet Mask	/24
11111111.11111111.11111111.00000000	
Network	Host
11101010.10001000.00011001	00110010

# Forwarding Tables

IP address configuration in the *Windows* environment.



# Forwarding

Example: Destination IP address = **172.30.33.6**

Row	Network/ Subnet	Mask (/Prefix)*	Metric (Cost)	Interface	Next- Hop Router
1	128.171.0.0	255.255.0.0 (/16)	47	2	G
2	172.30.33.0	255.255.255.0 (/24)	0	1	Local
3	192.168.6.0	255.255.255.0 (/24)	12	2	G

Router tests **1<sup>st</sup> row** of the forwarding table:

IP address = 172.30.33.6

Mask = 255.255.0.0

Result = 172.30.0.0

This result is different from the Network/Subnet value: 128.171.0.0



# *Forwarding*

Example: Destination IP address = **172.30.33.6**

Row	Network/ Subnet	Mask (/Prefix)*	Metric (Cost)	Interface	Next- Hop Router
1	128.171.0.0	255.255.0.0 (/16)	47	2	G
2	172.30.33.0	255.255.255.0 (/24)	0	1	Local
3	192.168.6.0	255.255.255.0 (/24)	12	2	G

Router tests **2<sup>nd</sup> row** of the forwarding table:

IP address = 172.30.33.6

Mask = 255.255.255.0

Result = 172.30.33.0

This result matches the Network/Subnet field value: 172.30.33.0



# *Forwarding*

**TPC: Prob. 8**

Row	Network/ Subnet	Mask (/Prefix)*	Metric (Cost)	Interface	Next- Hop Router
15	0.0.0.0	0.0.0.0 (/0)	5	3	H

If the mask takes the value 0.0.0.0 there is always a (length 0) matching – the result is always 0.0.0.0:

- This allows to define the ***default routing***, assumed when no other line(s) provide a match.



# Forwarding

For each packet:

- First, for each routing table row, apply the mask and look for matching:
  - Analyse the packet's destination IP address;
  - Apply the mask of that routing table's row;
  - Compare masking result with the value of the *Network/Subnet* field of that row;
  - If there is a positive match:
    - Add this row to the list of candidates to route this packet;
    - Else, ignore this row.



# Forwarding



- Second, search for the best (longest) matching:
  - If there is only one match, that is the best one;
  - If there is only one longest matching, that is the best one;
  - If there are several matchings with the longest length, select the row with the lowest cost metric:
    - It can be the lowest value (e.g.: cost);
    - It can be the largest value (e.g.: throughput);
- Third, forward the packet to a network interface:
  - Send the packet to the network interface indicated in the selected row;
  - In that network or subnet, send the packet to:
    - The *next-hop-router*, or:
    - The destination station, if the *next-hop router* field contains the value “local”.

# Forwarding

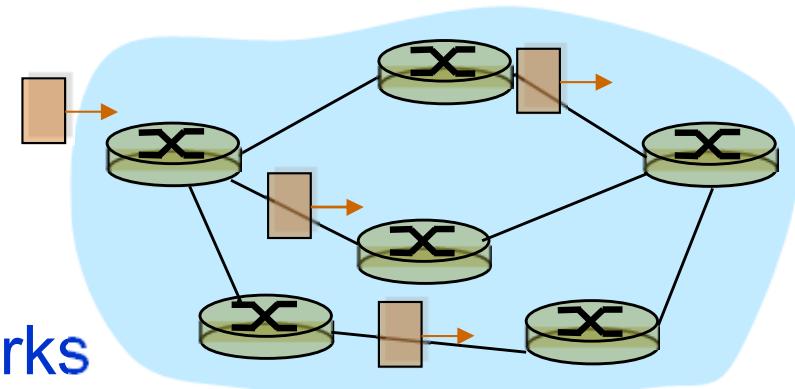


## Summary:

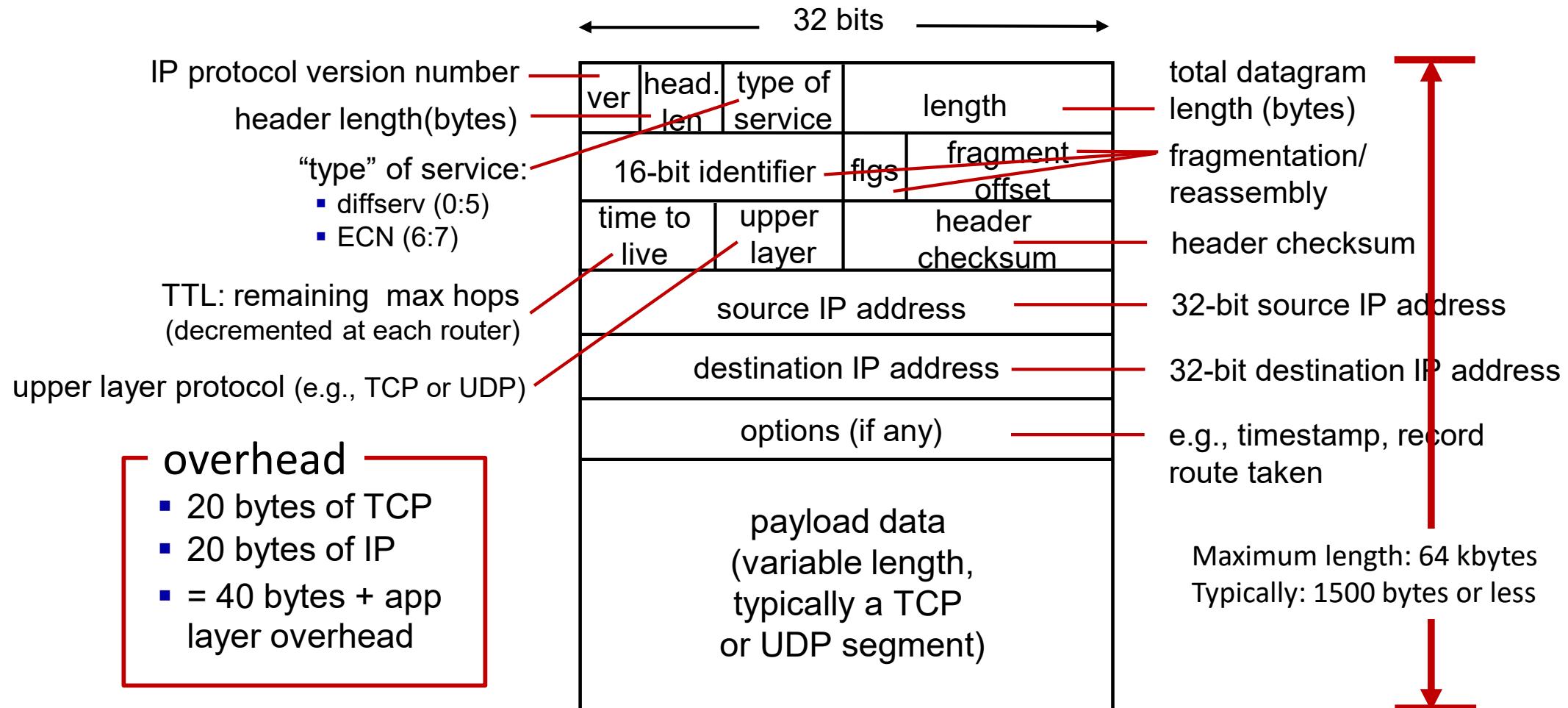
- A forwarding decision requires that each routing table row is tested, for each packet, to choose the best path;
  - Lengthy operation;
- Each packet is separately processed;
  - Router must have high processing power;
- With alternative routes, there may be several alternatives to forward/route a packet;
  - Choice will depend on the metric values in each row.

# Outline

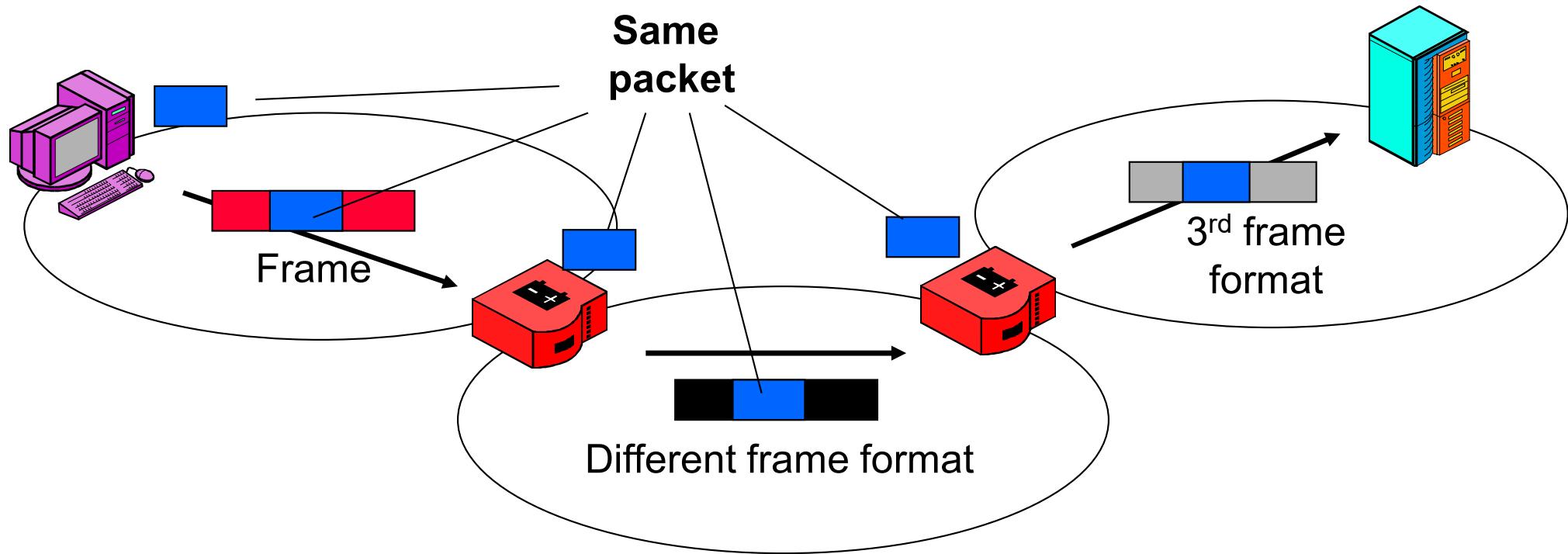
- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



## IP Datagram format



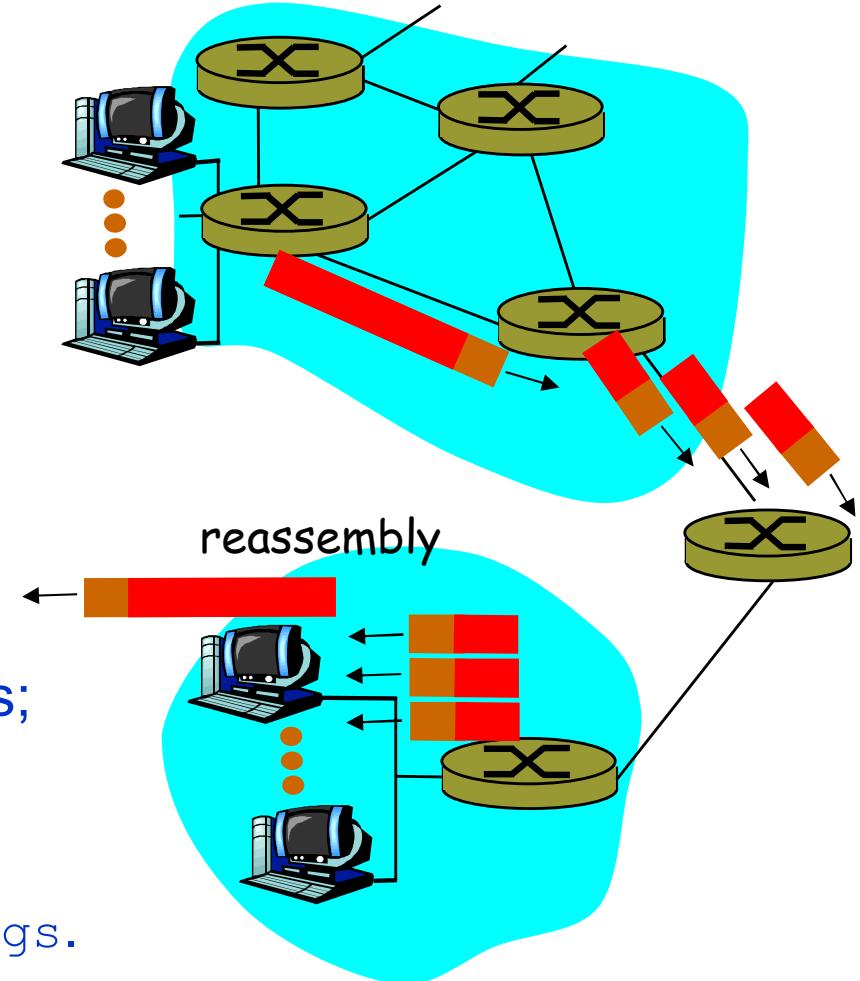
# IP Fragmentation and Reassembly: Why?



A router removes the packet from the origin network and retransmits it in the following network of the selected path (**store-and-forward**), using in each different network the appropriate frame format.

# IP Fragmentation and Reassembly

- Maximum Transmission Unit (MTU) is the largest possible data link frame:
  - Different link types → Different MTU values.
- Large IP datagrams are “fragmented” within the net:
  - One datagram becomes several datagrams;
  - “Reassembled” only at final destination;
  - IP header bits used to identify and reorder fragments, using fragment offset + flags.



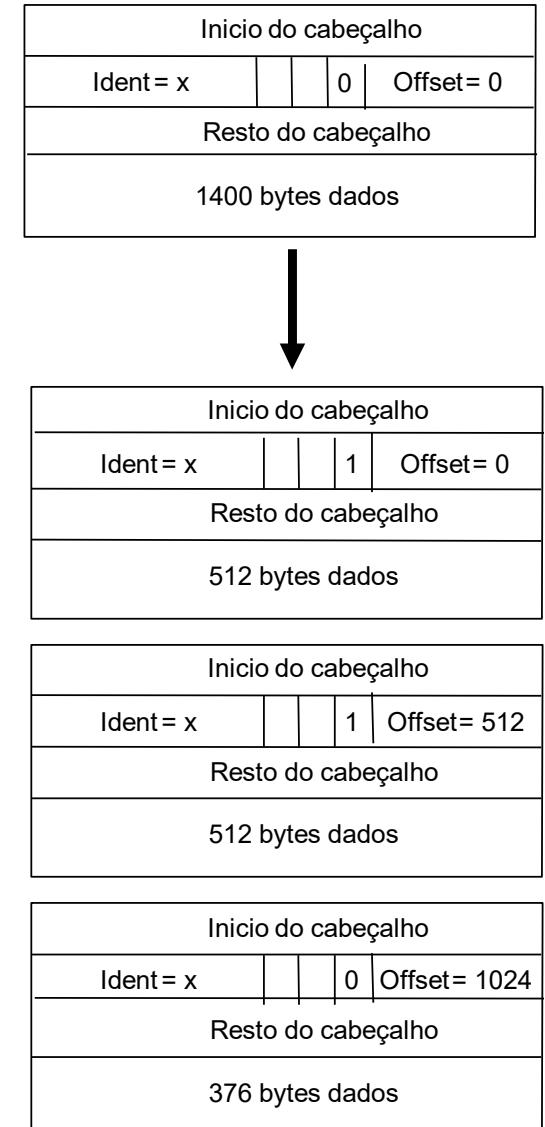
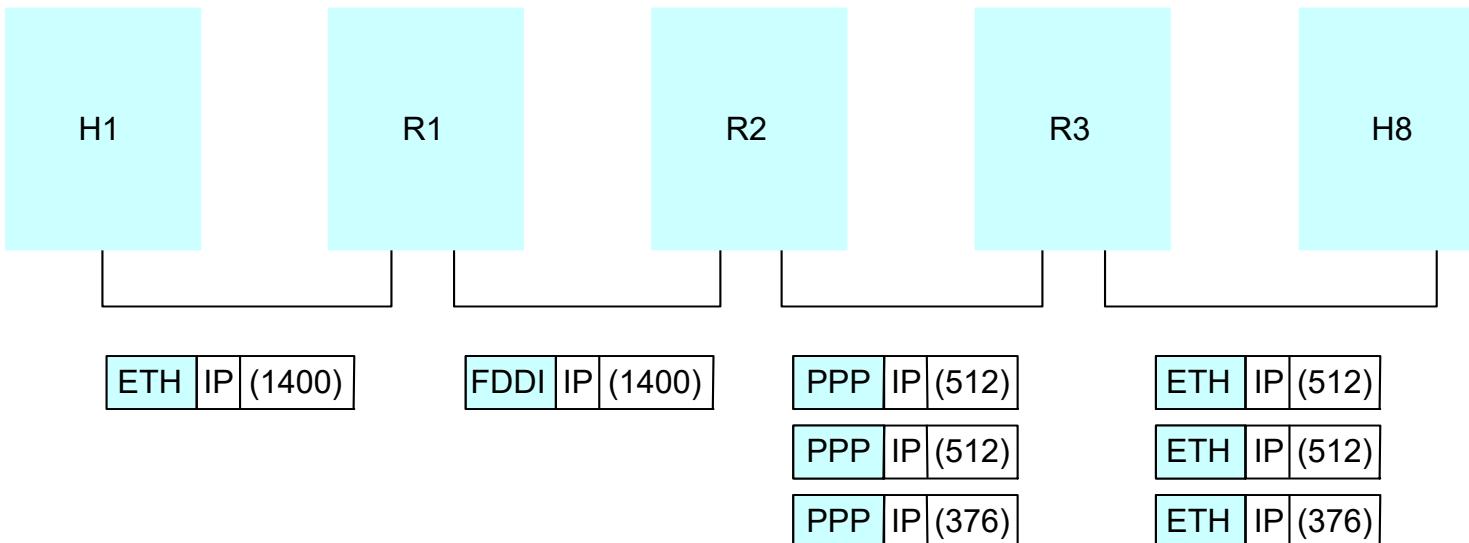
**Fragmentation:**  
 in: one large datagram  
 out: 3 smaller datagrams

# IP Fragmentation and Reassembly

Each network has its MTU value;

Strategy:

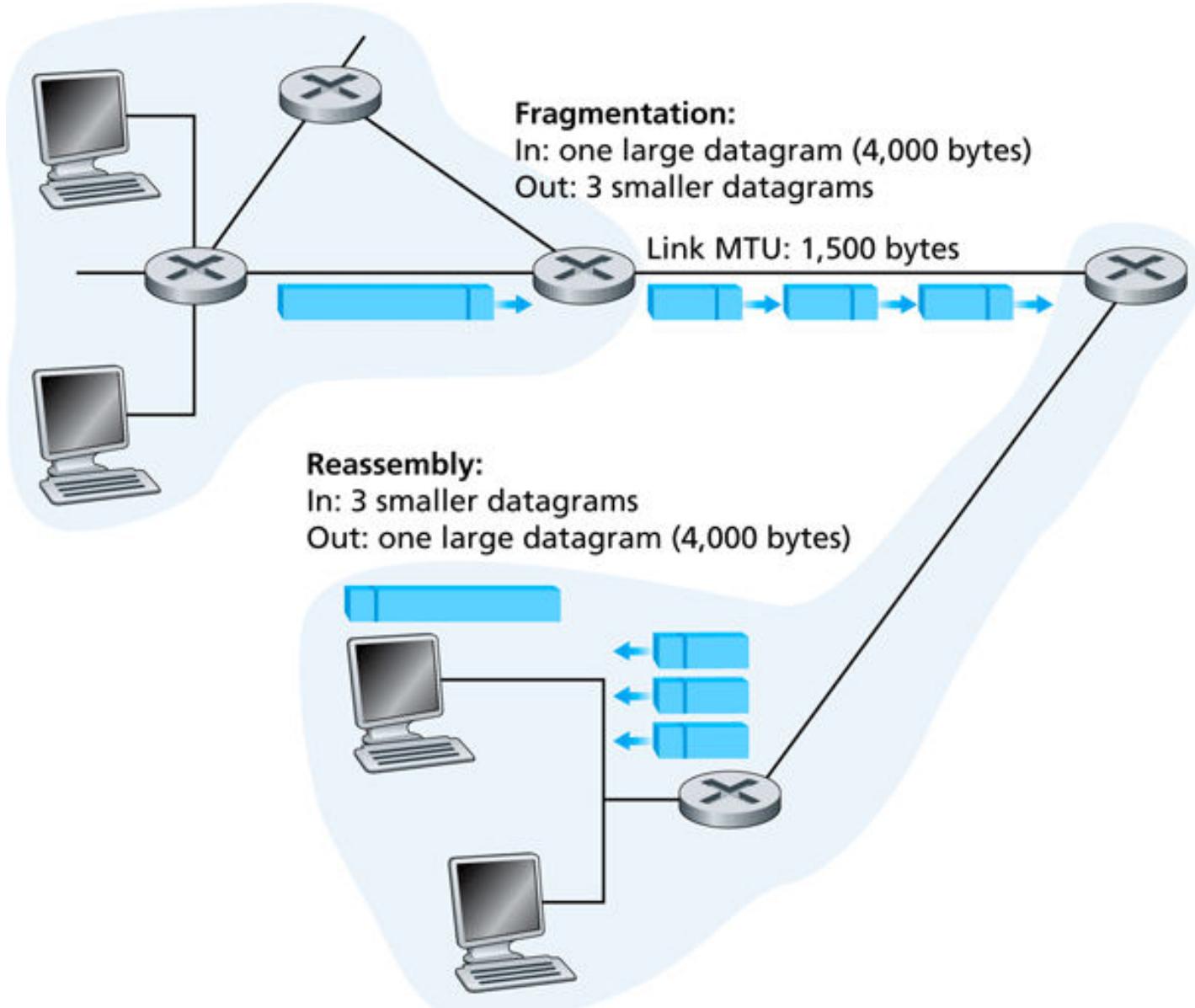
- Fragment when needed;
- Refragmentation is possible;
- Each fragment composes a datagram (same ID);
- Reconstruction is only done at the destination;
- **Offset** indicates the number of the previous fragments' byte, in multiples of 8 bytes.



# IP Fragmentation and Reassembly

## Example:

- 4000 byte datagram;
- 20 byte IP header;
- MTU = 1500 bytes.



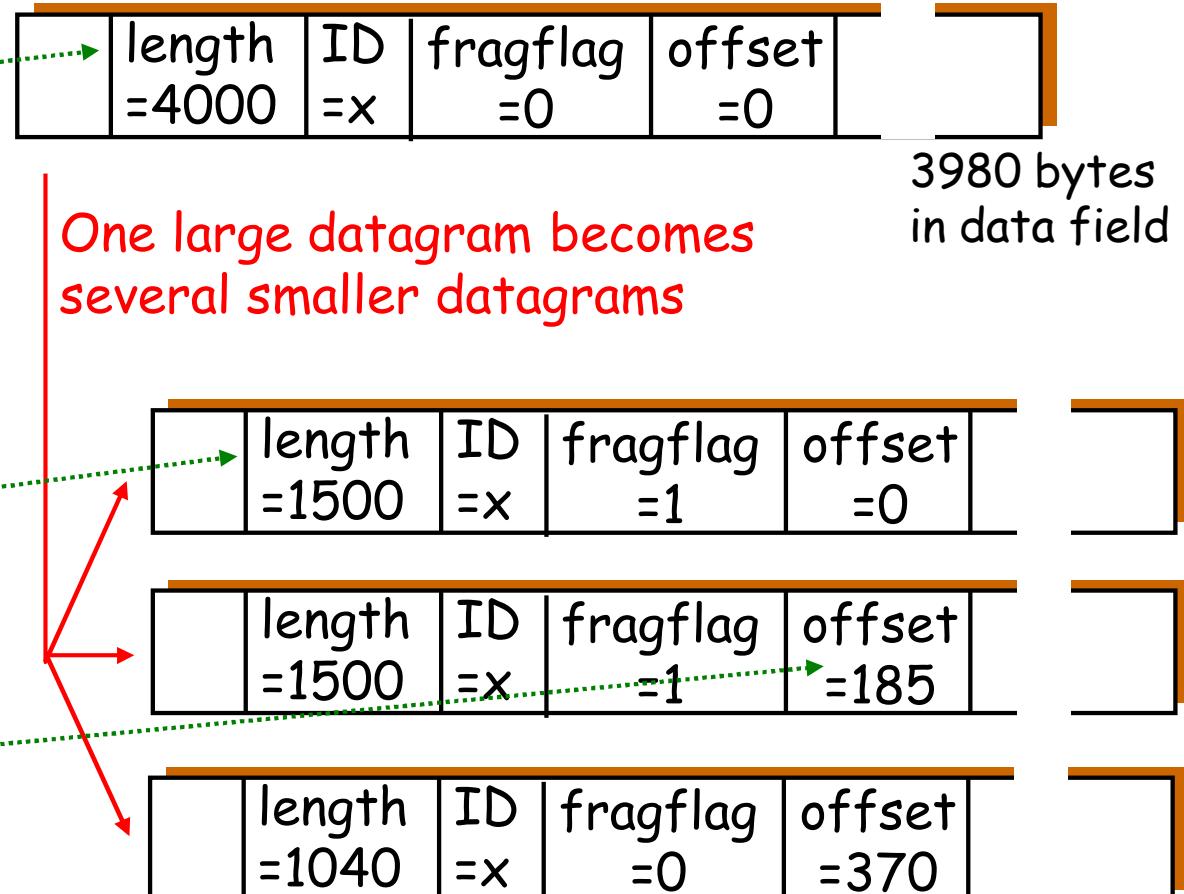
# IP Fragmentation and Reassembly

## Example:

- 4000 byte datagram (total length):
  - 20 byte IP header
  - 3980 data bytes
- MTU = 1500 bytes

total length = MTU = 1500 bytes  
 IP header (20) + data field (1480)

$$\text{offset} = \\ = 1480/8 = 185$$



What if offset is not integer?

e.g.: MTU = 1320 → offset = 1300/8 = 162.5

how to represent in binary (13 bits)?

# *IP Fragmentation and Reassembly*

**TPC: Prob. 9**

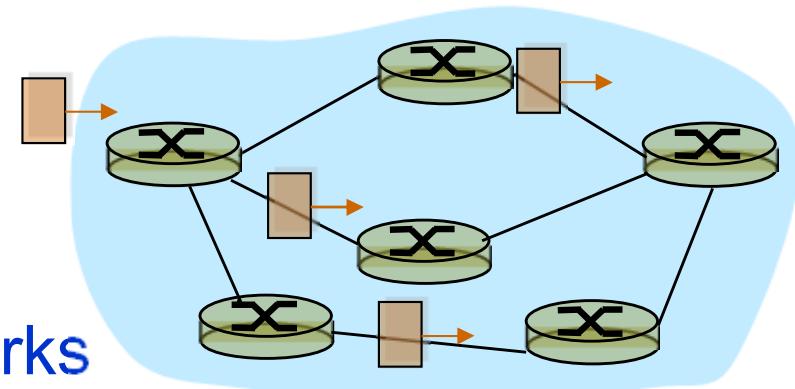
Example:

- 4000 byte datagram;
- 20 byte IP header;
- MTU = 1500 bytes.

Fragment	Bytes	ID	Offset	Flag
1st fragment	1,480 bytes in the data field of the IP datagram	identification = 777	offset = 0 (meaning the data should be inserted beginning at byte 0)	flag = 1 (meaning there is more)
2nd fragment	1,480 bytes of data	identification = 777	offset = 185 (meaning the data should be inserted beginning at byte 1,480. Note that $185 \cdot 8 = 1,480$ )	flag = 1 (meaning there is more)
3rd fragment	1,020 bytes $(= 3,980 - 1,480 - 1,480)$ of data	identification = 777	offset = 370 (meaning the data should be inserted beginning at byte 2,960. Note that $370 \cdot 8 = 2,960$ )	flag = 0 (meaning this is the last fragment)

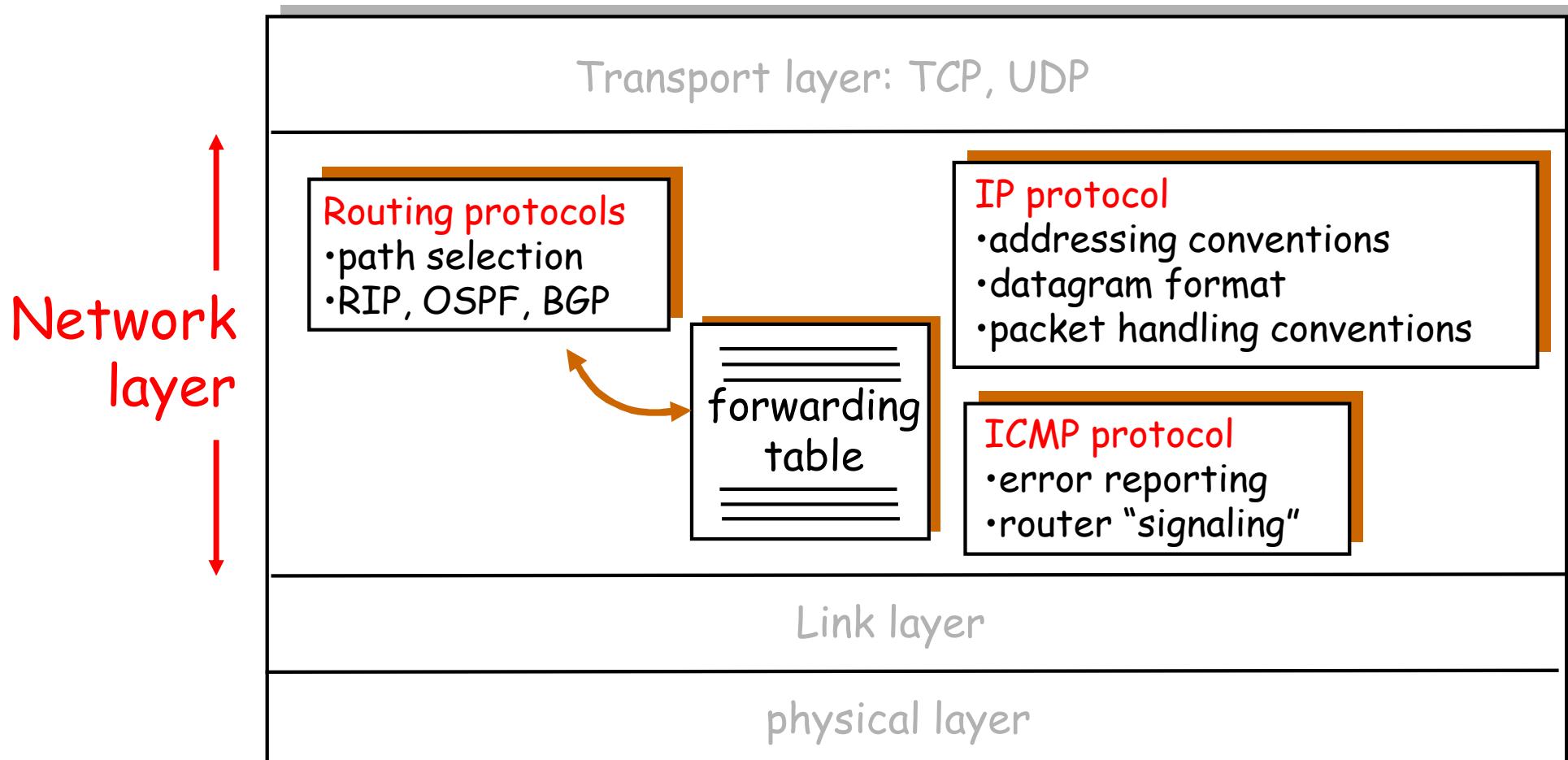
# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- **ICMP, DHCP**
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing

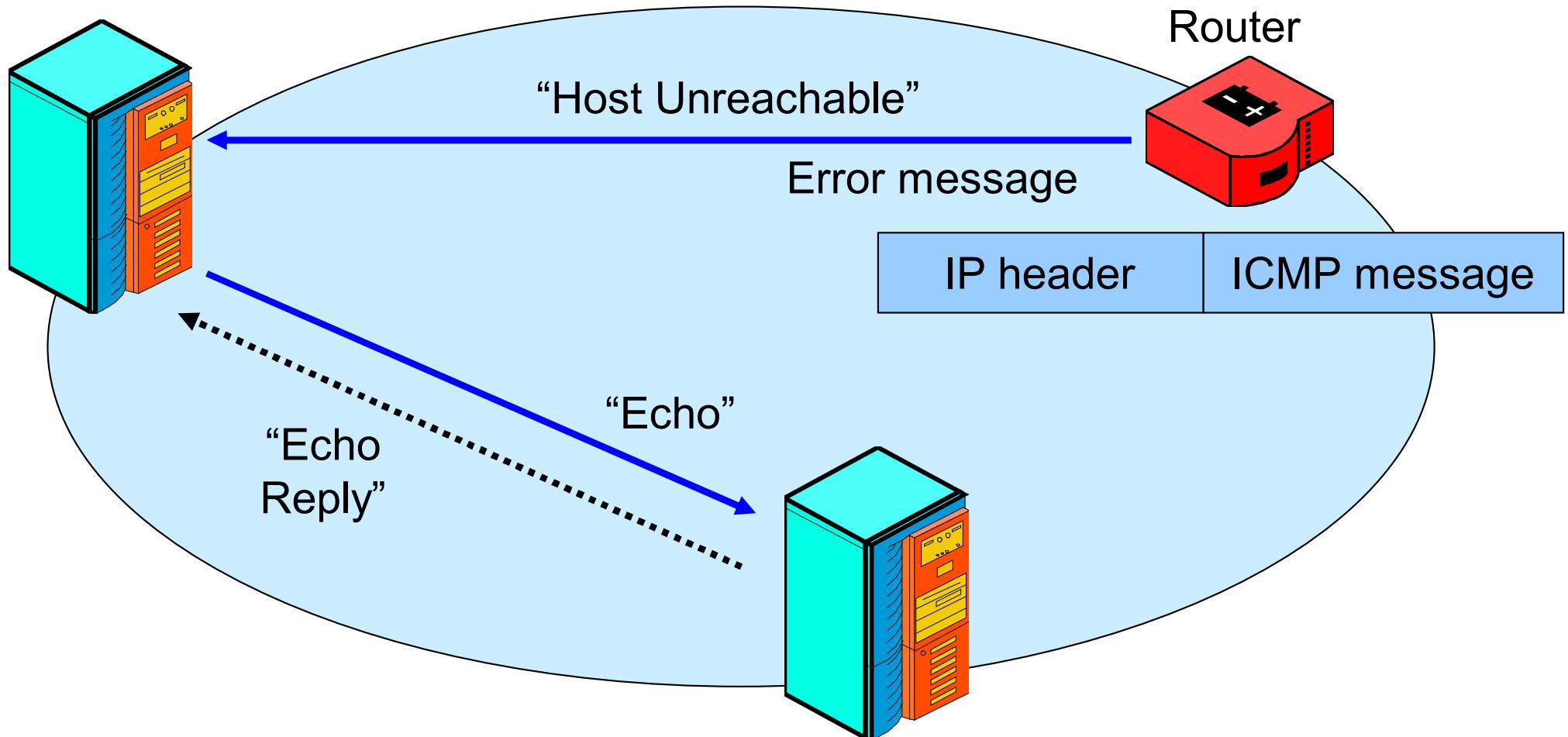


# The Internet Network Layer

Host, router network layer functions:



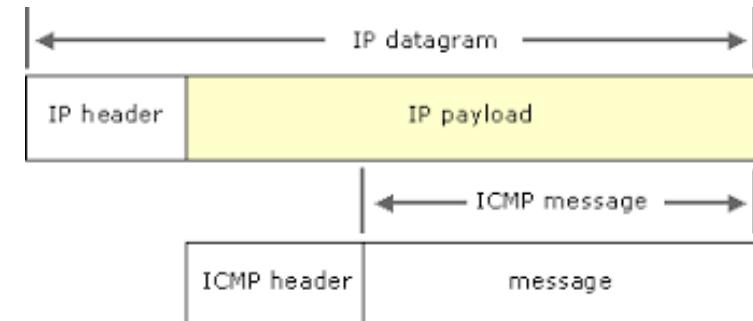
# *Internet Control Message Protocol (ICMP)*



# *Internet Control Message Protocol (ICMP)*

## ICMP (RFC 792):

- Used by hosts and routers to communicate network-level information:
  - Error reporting: unreachable host, network, port, protocol;
  - Echo request/reply (used by ping).
- Network-layer “above” IP:
  - ICMP messages carried in IP datagrams;
- ICMP message:
  - Type and code;
  - First 8 bytes of IP datagram causing error.



# *Internet Control Message Protocol (ICMP)*

Type	Code	description
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

# Internet Control Message Protocol (ICMP)

Type	Code	Checksum	
Unused			
IP Header + 64 bits of original datagram			

(a) Destination Unreachable; Time Exceeded; Source Quench

Type	Code	Checksum	
Identifier			
Sequence Number			

(e) Timestamp

Type	Code	Checksum	
Pointer			
Unused			

(b) Parameter Problem

Type	Code	Checksum	
Identifier			
Sequence Number			
Originate Timestamp			
Receive Timestamp			
Transmit Timestamp			

(f) Timestamp Reply

Type	Code	Checksum	
Gateway Internet Address			
IP Header + 64 bits of original datagram			

(c) Redirect

Type	Code	Checksum	
Identifier			
Sequence Number			

(g) Address Mask Request

Type	Code	Checksum	
Identifier			
Sequence Number			

(d) Echo, Echo Reply

Type	Code	Checksum	
Identifier			
Sequence Number			

(h) Address Mask Reply

# *Traceroute and ICMP*

- Source sends series of UDP segments to destination:
  - First has TTL =1;
  - Second has TTL=2, etc.
  - Unlikely port number.
- When  $n^{\text{th}}$  datagram arrives to  $n^{\text{th}}$  router (TTL):
  - Router discards datagram;
  - Router sends to source an ICMP message (type 11, code 0);
  - Message includes name of router and IP address;
- When ICMP message arrives, source calculates RTT;
- Traceroute does this 3 times.

## Stopping criterion:

- UDP segment eventually arrives at destination host;
- Destination returns ICMP “host unreachable” packet (type 3, code 3);
- When source gets this ICMP, stops.

# Traceroute

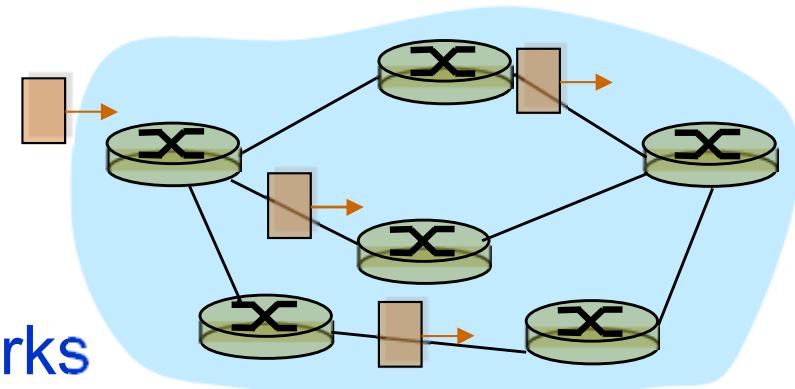
> traceroute www.google.com

```
traceroute to www.google.com (74.125.39.103), 30 hops max, 40 byte packets
```

```
1  gt-ci-tn.ist.utl.pt (193.136.138.254)  1.797 ms  2.207 ms  2.441 ms
2  gatekeeper1.ci.ist.utl.pt (192.168.253.3)  0.260 ms  0.277 ms  0.217 ms
3  brites2.utl.pt (193.136.134.11)  0.360 ms  0.445 ms  0.348 ms
4  Router3.GE.Lisboa.fccn.pt (193.136.1.89)  0.554 ms  0.564 ms  0.528 ms
5  ROUTER4.10GE.Lisboa.fccn.pt (193.137.0.20)  0.614 ms  0.557 ms  0.640 ms
6  fccn.rt1.mad.es.geant2.net (62.40.124.97)  38.229 ms  38.005 ms  37.992 ms
7  so-7-2-0.rt1.gen.ch.geant2.net (62.40.112.25)  33.593 ms  33.689 ms  33.744 ms
8  so-3-3-0.rt1.fra.de.geant2.net (62.40.112.70)  41.916 ms  41.926 ms  42.056 ms
9  TenGigabitEthernet7-3.ar1.FRA4.gblx.net (207.138.144.45)  42.218 ms  42.236 ms
   42.395 ms
10 74.125.50.189 (74.125.50.189)  42.117 ms  42.125 ms  42.223 ms
11 209.85.255.170 (209.85.255.170)  42.302 ms  42.299 ms  42.277 ms
12 209.85.254.116 (209.85.254.116)  42.729 ms  42.748 ms  42.762 ms
13 209.85.249.162 (209.85.249.162)  42.820 ms  209.85.249.166 (209.85.249.166)
   42.980 ms  42.981 ms
14 fx-in-f103.google.com (74.125.39.103)  42.842 ms  43.061 ms  43.040 ms
```

# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, **DHCP**
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# IP Addresses: How to Get One?

Q: How does a *host* get an IP address?

- Hard-coded by system administrator in a file:
  - Windows: control-panel->network->configuration->TCP/IP->properties;
  - UNIX: /etc/rc.config.
- **DHCP: Dynamic Host Configuration Protocol** (RFC 2131, updated by: 3396, 4361, 5494, 6842):
  - Dynamically get an address from server;
  - “Plug-and-play”.



# DHCP: *Dynamic Host Configuration Protocol*

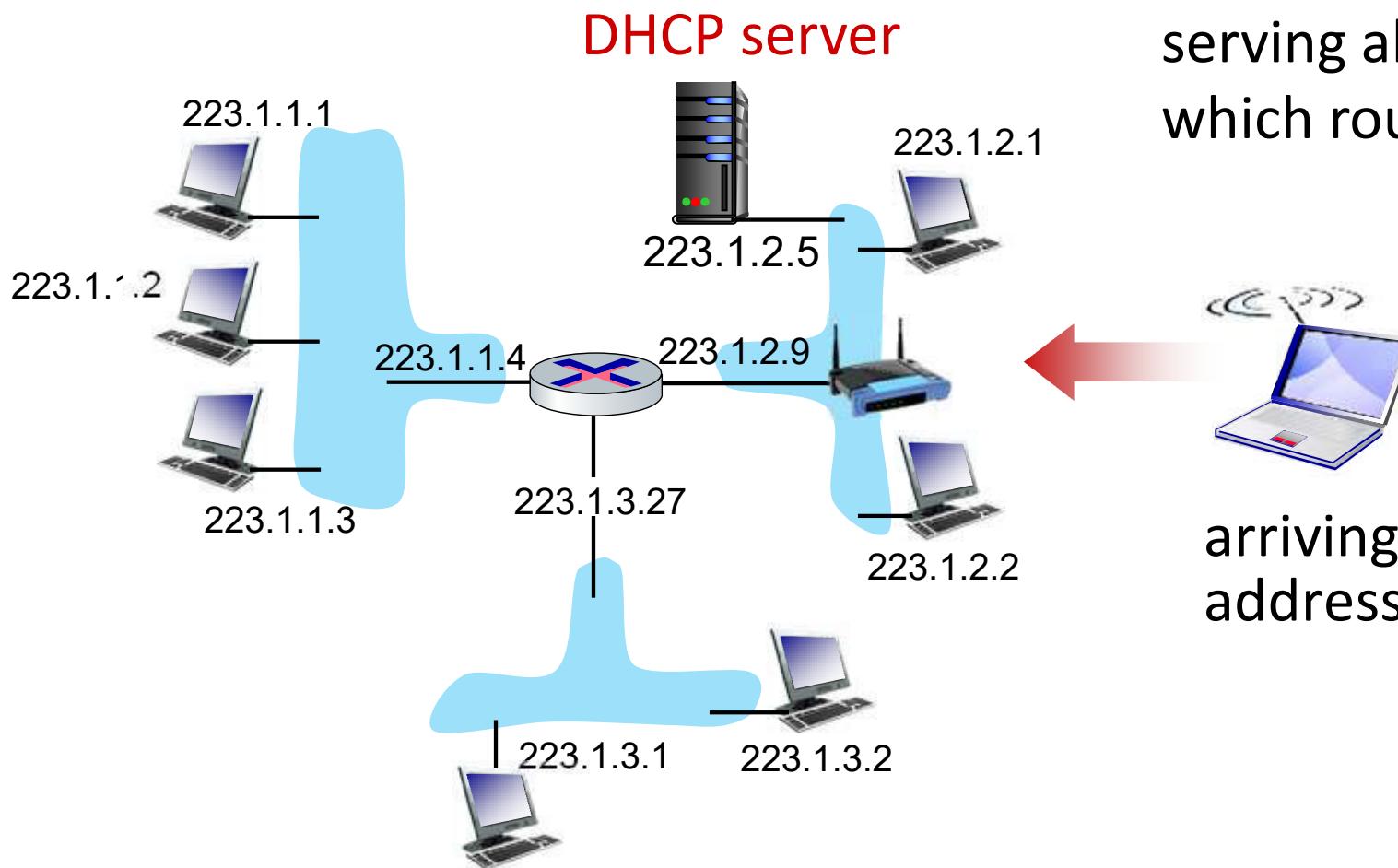
Goal: allow host to *dynamically* obtain its IP address from a network server when it joins the network:

- Allows reuse of addresses (only hold address while connected and “on”);
- Can renew its lease on address in use;
- Support for mobile users who want to join network.

DHCP overview:

- Host broadcasts “DHCP discover” message;
- DHCP server responds with “DHCP offer” message;
- Host requests IP address: “DHCP request” message;
- DHCP server sends address: “DHCP ACK” message.

# DHCP Client-Server Scenario

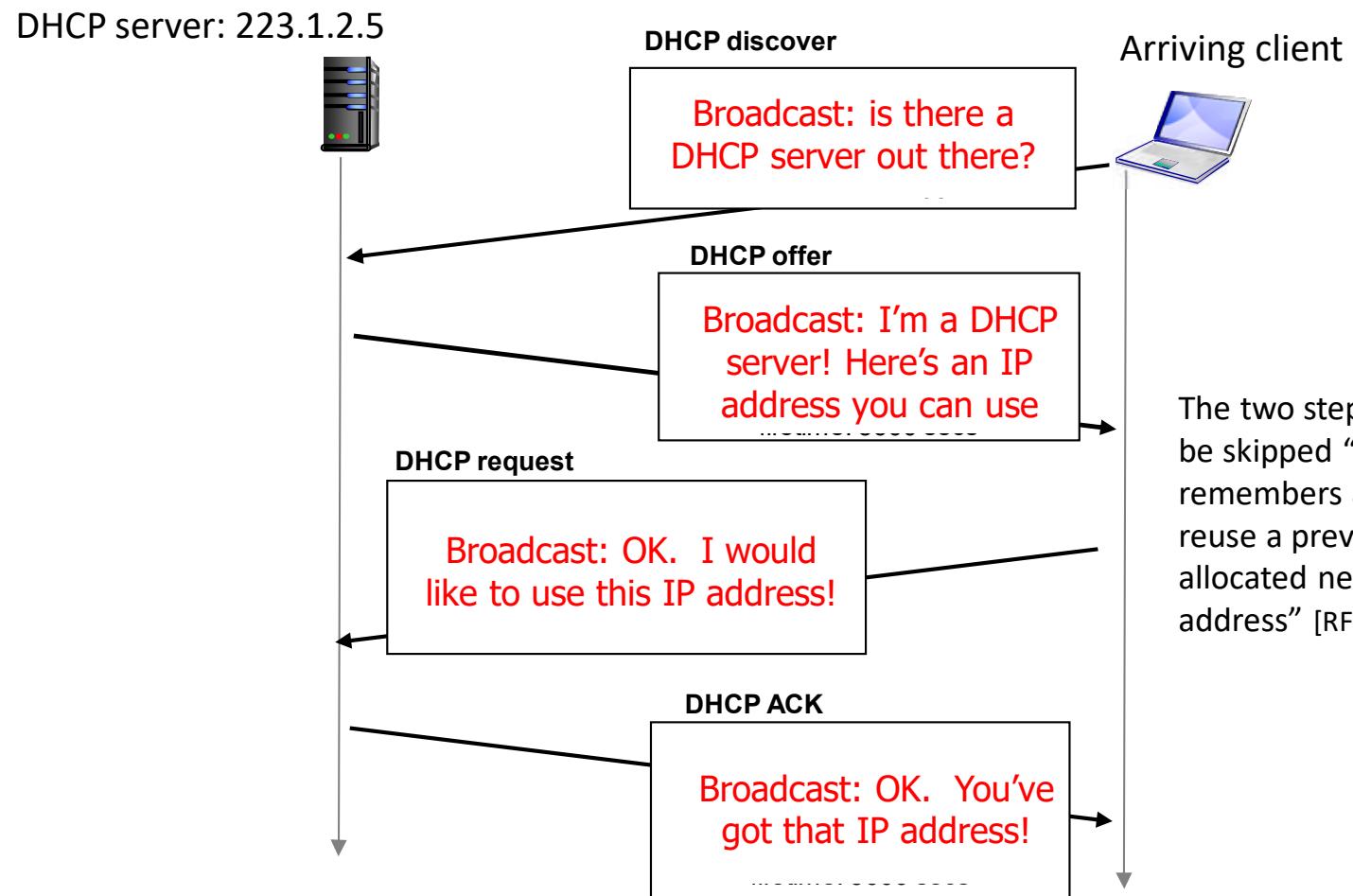


Typically, DHCP server will be co-located in router, serving all subnets to which router is attached



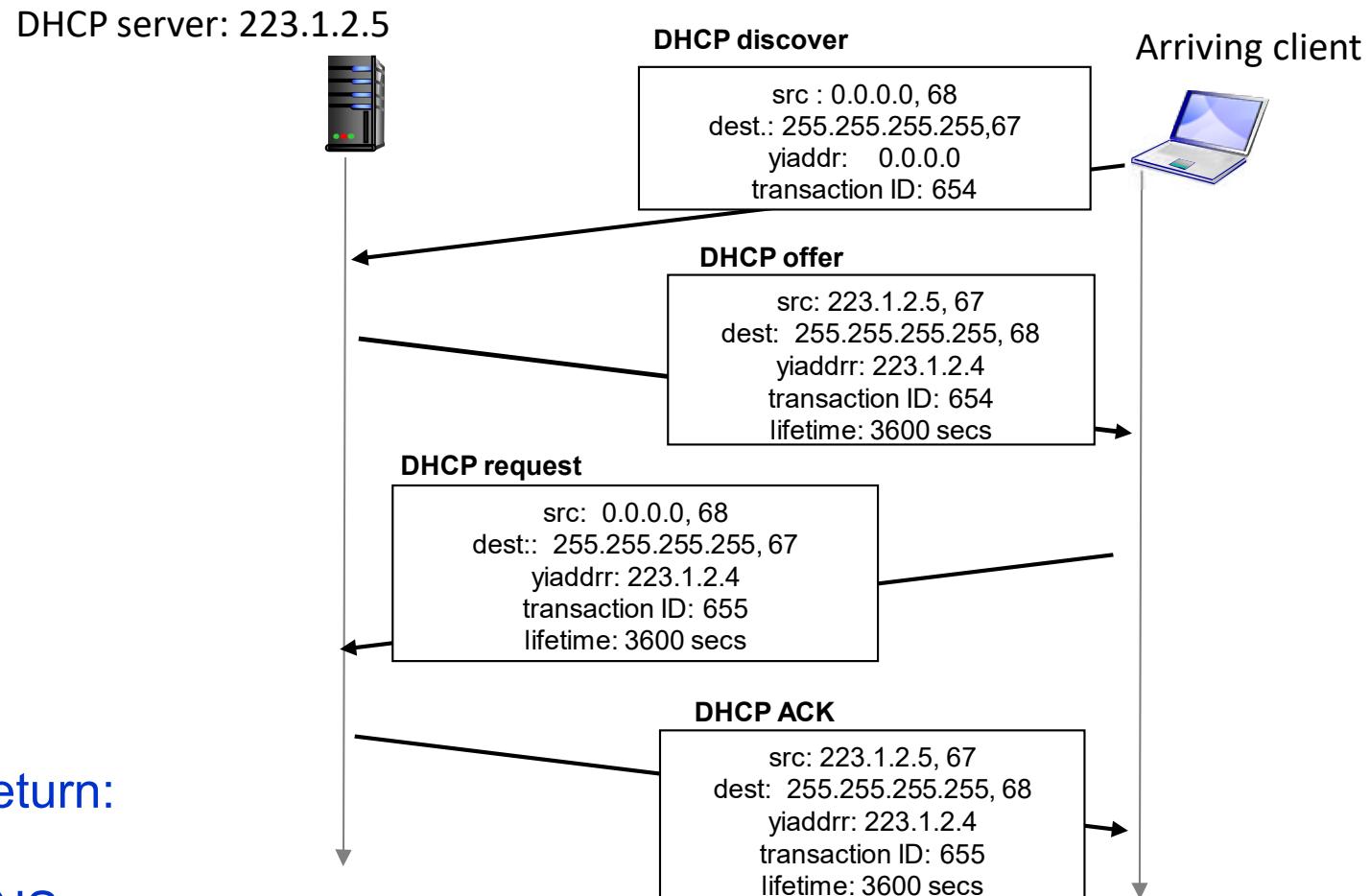
arriving DHCP client needs address in this network

# DHCP Client-Server Scenario



The two steps above can be skipped “if a client remembers and wishes to reuse a previously allocated network address” [RFC 2131]

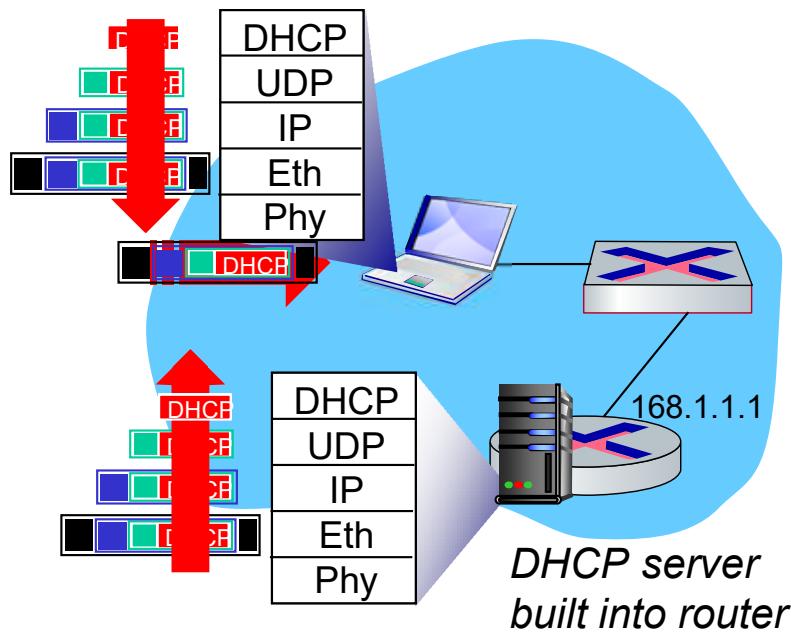
# DHCP Client-Server Scenario



DHCP can also return:

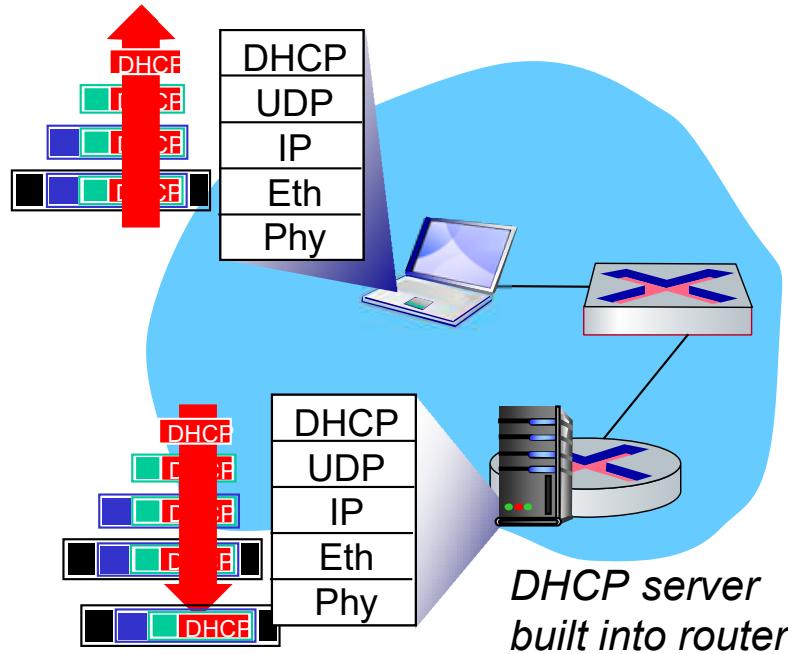
- network mask
- IP address of DNS sever
- address of first-hop router

## DHCP: example



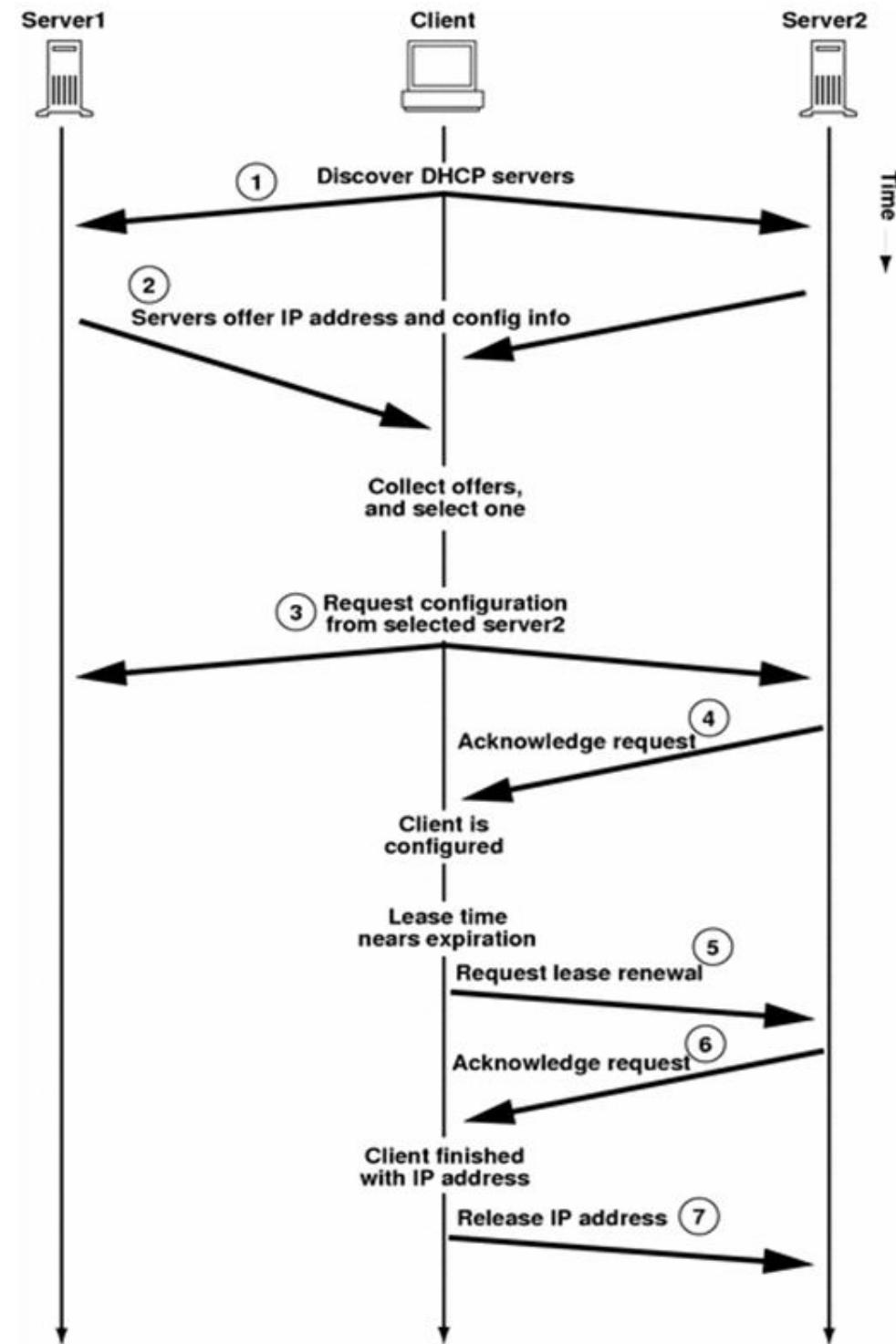
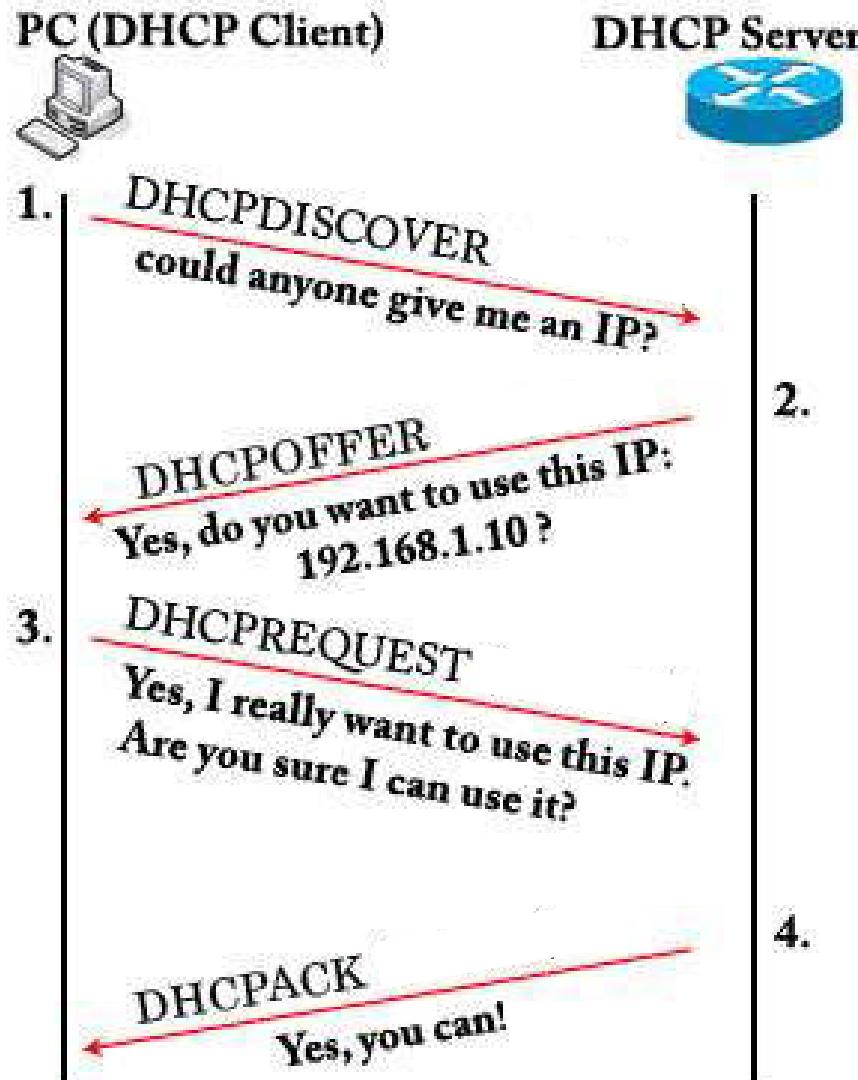
- Connecting laptop will use DHCP to get IP address (+ address of first-hop router, + address of DNS server).
- DHCP REQUEST message encapsulated in UDP segment, encapsulated in IP packet, encapsulated in Ethernet frame
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server

## DHCP: example



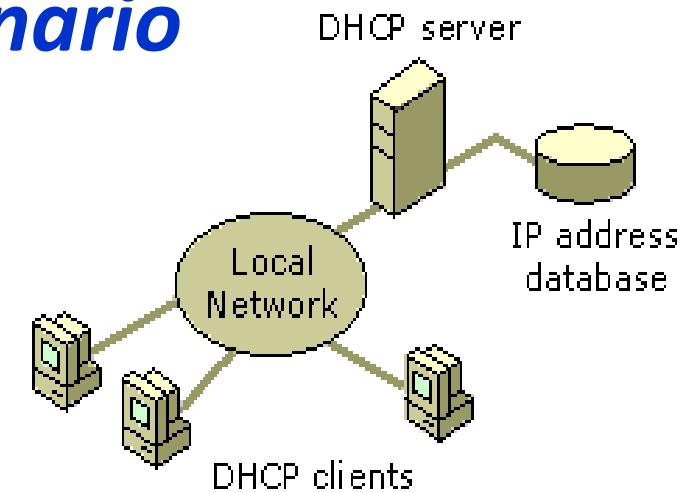
- DCP server formulates DHCP ACK containing client's IP address,  
+ IP address of first-hop router,
- + name & IP address of DNS server

# DHCP

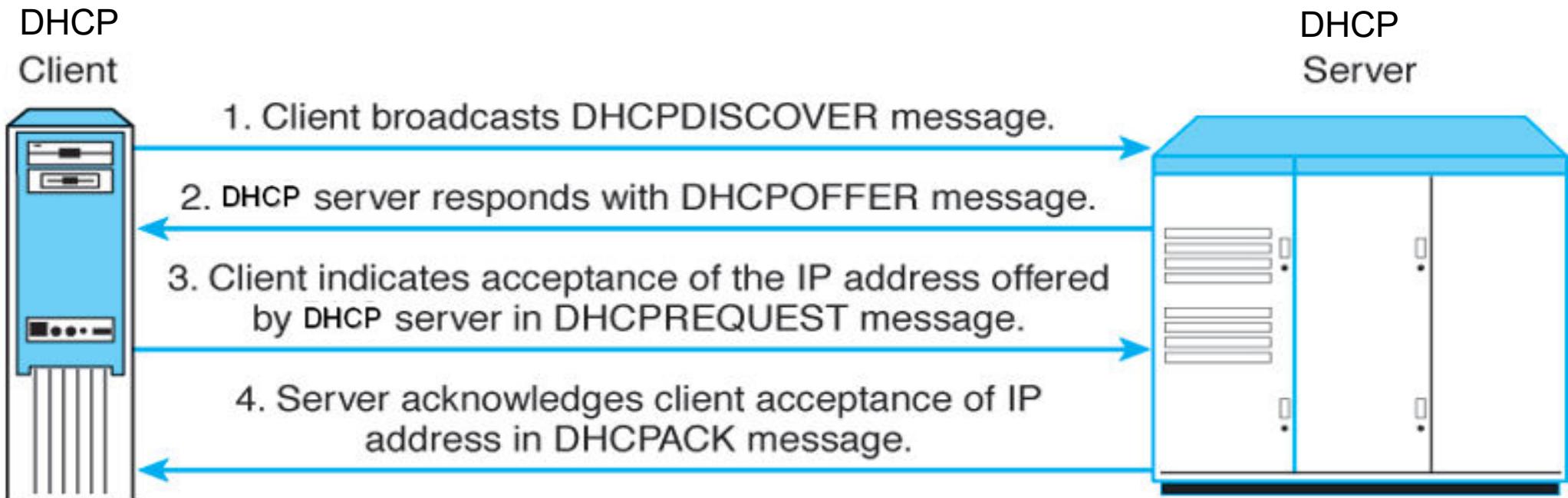


# DHCP Client-Server Scenario

- DHCP uses UDP.
- Client packets: source port 68, destination port 67;
- Server packets: source port 67, destination port 68.



DHCPv6 uses UDP port number 546 for clients and port number 547 for servers.



# DHCP

The screenshot shows a Wireshark capture window titled '(Untitled) - Wireshark'. The filter bar at the top contains 'bootp'. The main pane displays a list of network frames, and the details pane below shows the structure of a DHCP Request message.

No.	Time	Source	Destination	Protocol	Info
4	5.121896	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x9bebb465
6	5.188646	192.168.2.1	255.255.255.255	DHCP	DHCP NAK - Transaction ID 0x9bebb465
15	6.259175	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x3b5b6961
16	6.325627	192.168.2.1	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0x3b5b6961
17	6.328399	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x3b5b6961
18	6.395857	192.168.2.1	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x3b5b6961

Details pane content:

```
Client hardware address padding: 00000000000000000000000000000000
Server host name not given
Boot file name not given
Magic cookie: (OK)
+ Option: (t=53,l=1) DHCP Message Type = DHCP Request
+ Option: (t=61,l=7) Client identifier
+ Option: (t=50,l=4) Requested IP Address = 192.168.1.111
+ Option: (t=12,l=7) Host Name = "Paul-PC"
+ Option: (t=81,l=10) Client Fully Qualified Domain Name
+ Option: (t=60,l=8) Vendor class identifier = "MSFT 5.0"
+ Option: (t=55,l=12) Parameter Request List
End option
```

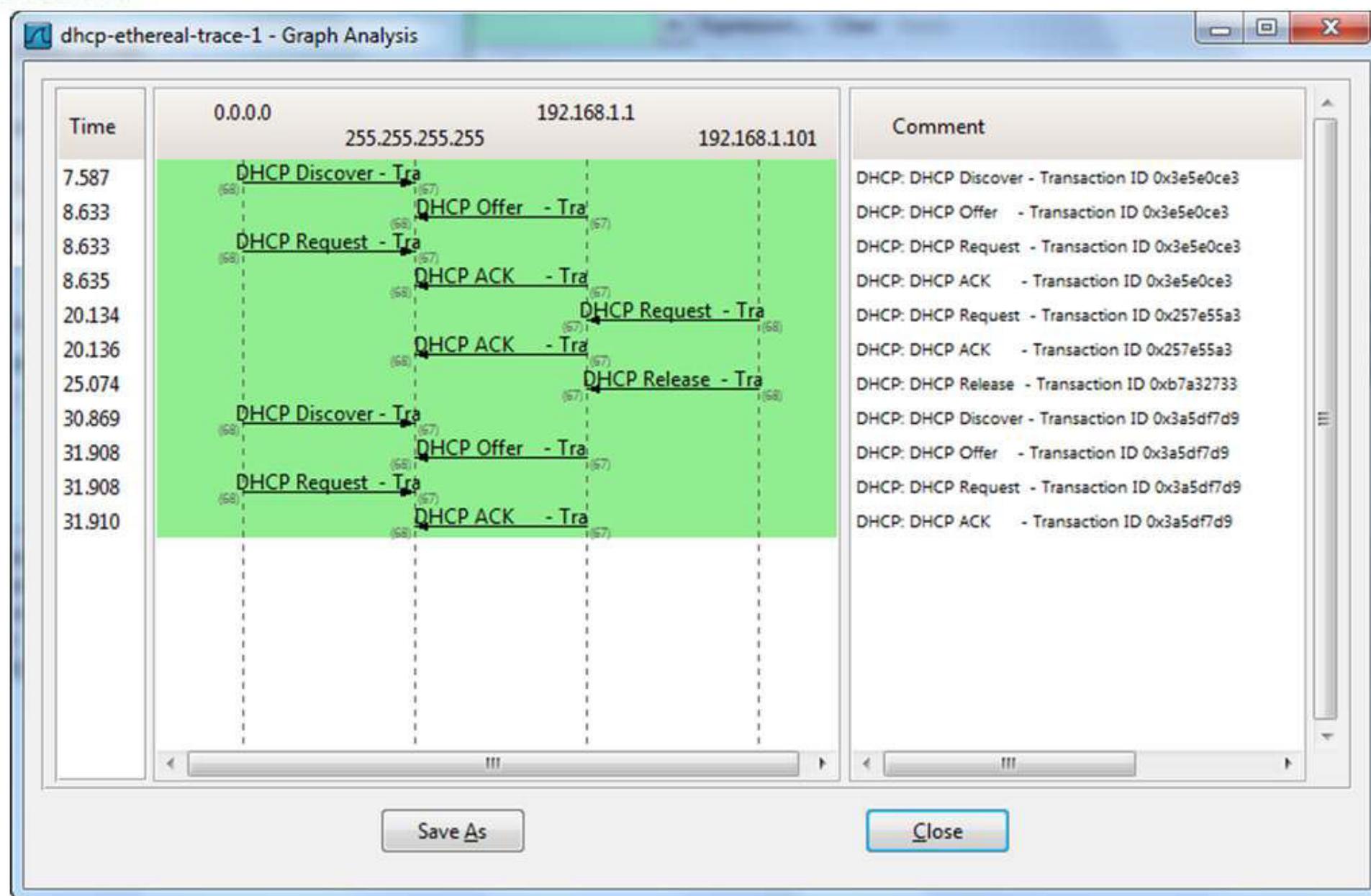
Hex and ASCII panes at the bottom show the raw bytes of the selected frame.

## Address Leasing

- Address leasing:
  - *T<sub>1</sub> Time (50% of Lease Time)* – time after which the terminal should **try to renew** the address leasing;
  - *T<sub>2</sub> Time (85% of Lease Time)* – time after which the terminal should **try to renew** the address leasing **again**, if the first trial has failed;
  - *Lease Time* – time after which the terminal should **stop using** the leased address, if not renewed in the meantime.

It is common for the server to set the lease time to several hours or days [Droms 2002].

# DHCP

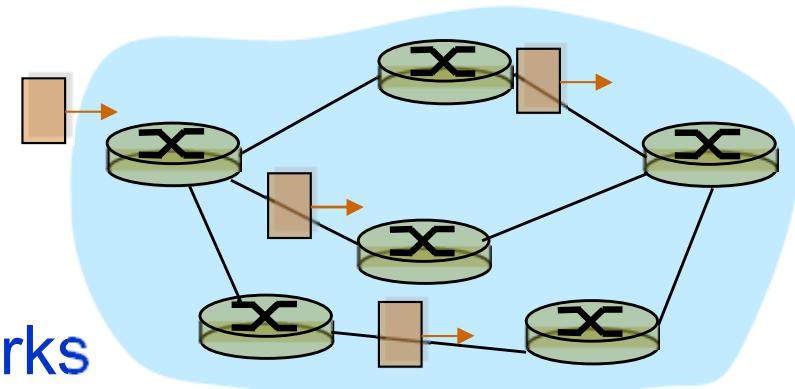


## *Other DHCP messages*

- *DHCP Decline:*
  - The client rejects the offer he was made and restarts the process;
- *DHCP Nack:*
  - The server informs it cannot satisfy the request done with the *DHCP Request* message;
- *DHCP Release:*
  - The client informs that he wants to terminate the address lease;
- *DHCP Inform:*
  - The client asks for just a few parameters (in this case, the client already has an IP address, but he asks, for instance, the address of a DNS server).

# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



## Private IP Addresses

A number of address blocks are assigned for private use.  
They are not recognized globally.

The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets (RFC 1918):

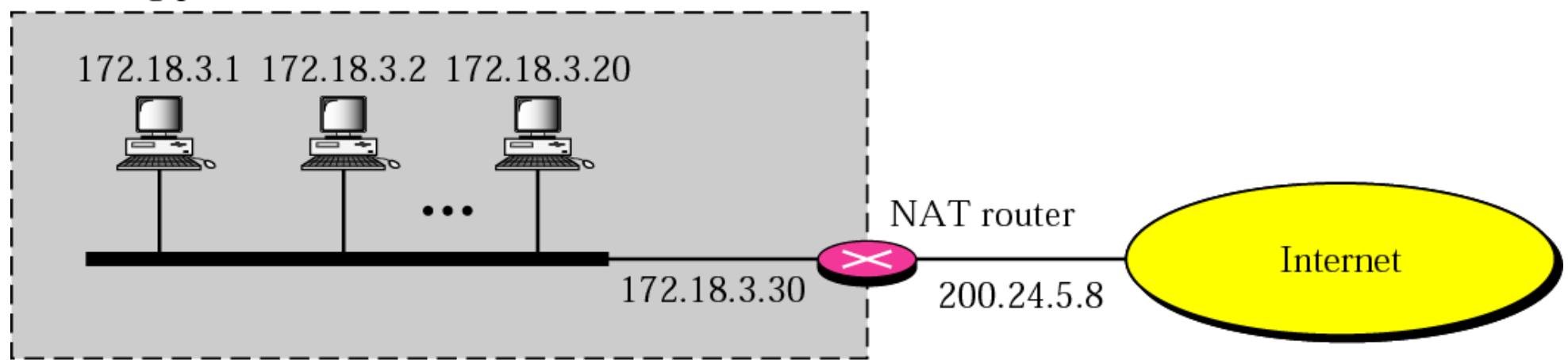
10.0.0.0	-	10.255.255.255	(10.0.0.0/8 prefix)
172.16.0.0	-	172.31.255.255	(172.16.0.0/12 prefix)
192.168.0.0	-	192.168.255.255	(192.168.0.0/16 prefix)

# Network Address Translation (NAT)

## Network Address Translation (NAT):

- Allows a (local) network to use a set of private addresses, for communication inside the network;
- Communication to outside the private network uses a set of (at least one) global IP address;
- All inside addresses are hidden from the outside – increased security.

Site using private addresses

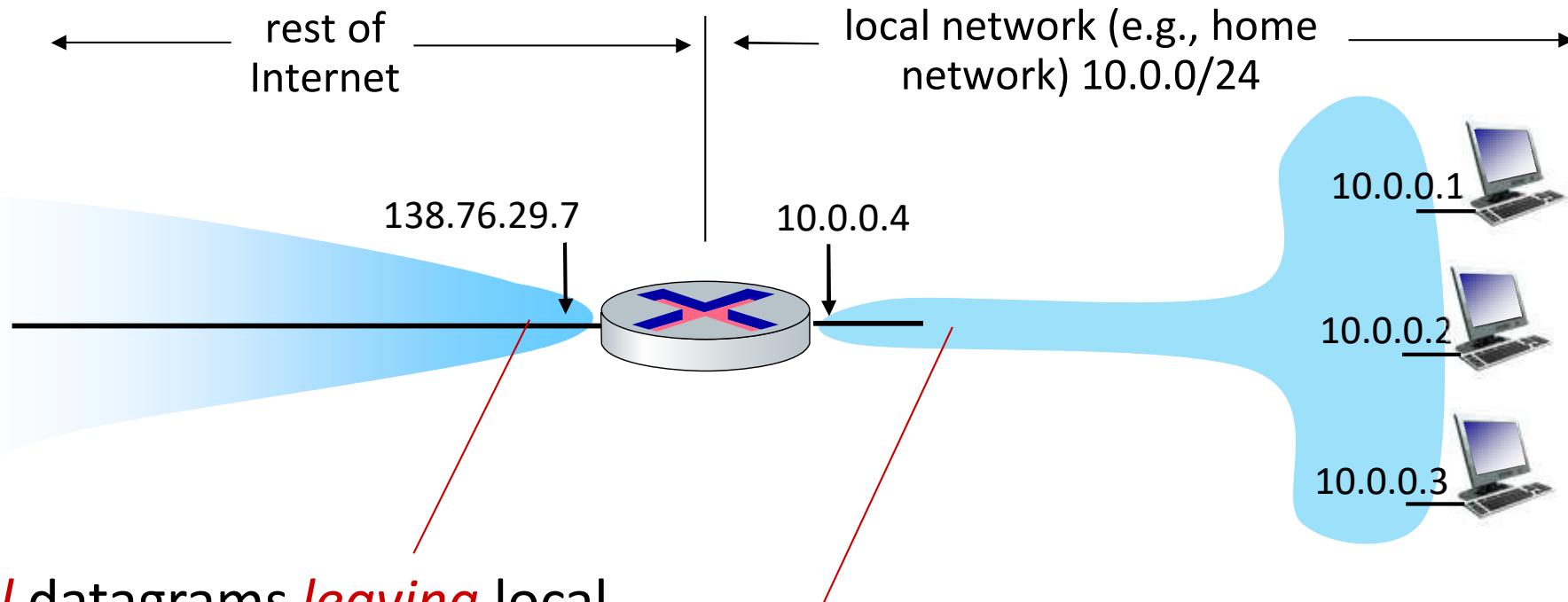


# Network Address Translation (NAT)

- Inside the network a set of *unregistered, or private, IP addresses* can be used (10.0.0.0 – 10.255.255.255; 172.16.0.0 – 172.31.255.255; 192.168.0.0 – 192.168.255.255).
- When a private network user sends a packet, the NAT replaces the internal sender IP address by the network external IP address. The correspondence is memorized.
- When a reply is received, the NAT restores the internal address after checking the memory.
- The NAT may use a fixed mapping table to support the reception of packets sent from outside the private network.
- If the internal address is not in memory, the packet is discarded.
- Can change ISP without changing addresses of devices in local network.

# Network Address Translation (NAT)

**NAT:** all devices in local network share just **one** IPv4 address as far as outside world is concerned



*all* datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different* source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

# Network Address Translation (NAT)

Implementation – NAT router must:

- *Outgoing datagrams:*

*Replace (source IP address, port #) of every outgoing datagram with (NAT IP address, new port #);*

*Remote host responds using (NAT IP address, new port #) as destination address.*

- *Remember (in NAT translation table):*

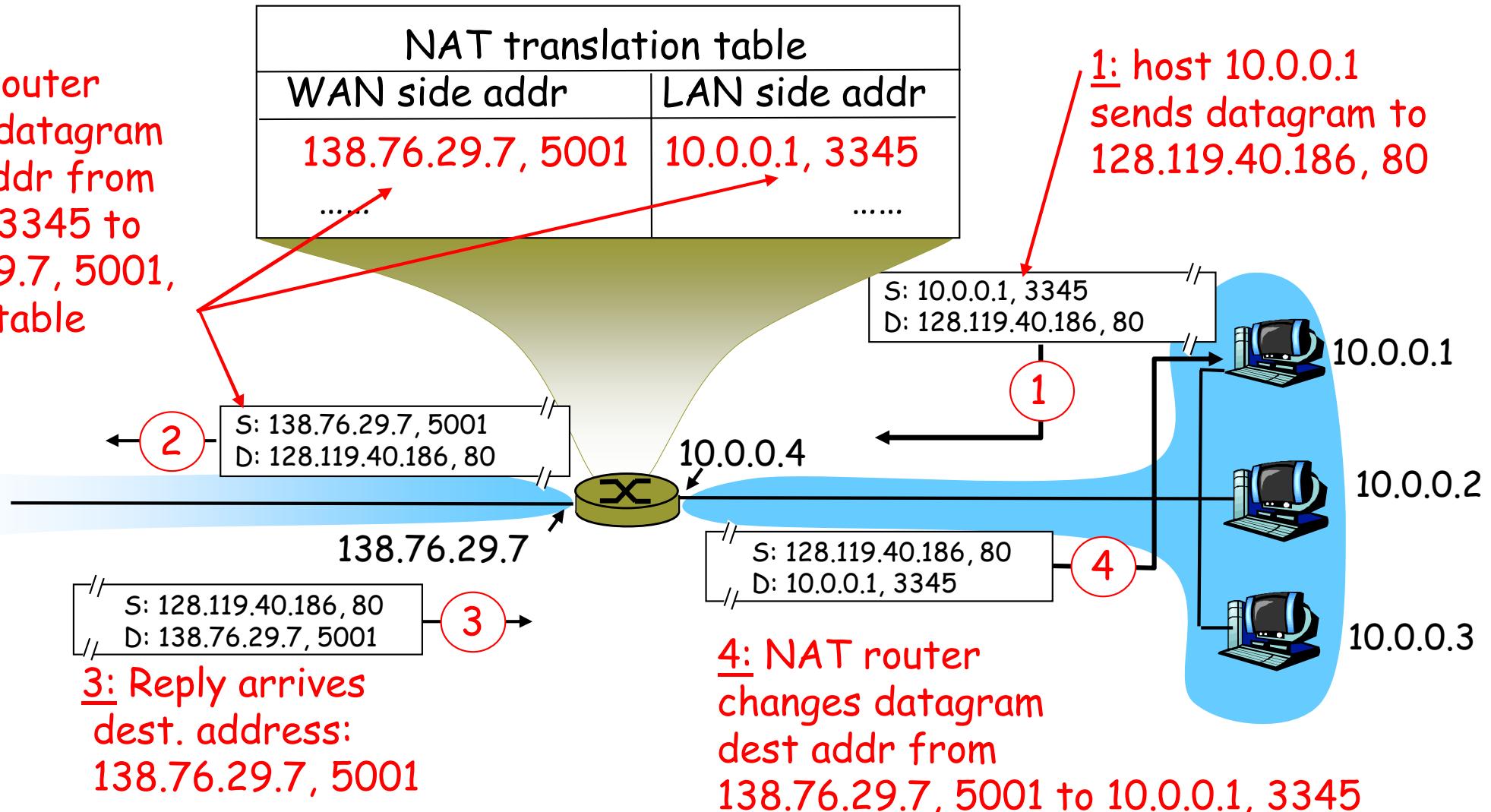
*Every (source IP address, port #) to (NAT IP address, new port #) translation pair.*

- *Incoming datagrams:*

*Replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in the NAT table.*

# Network Address Translation (NAT)

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

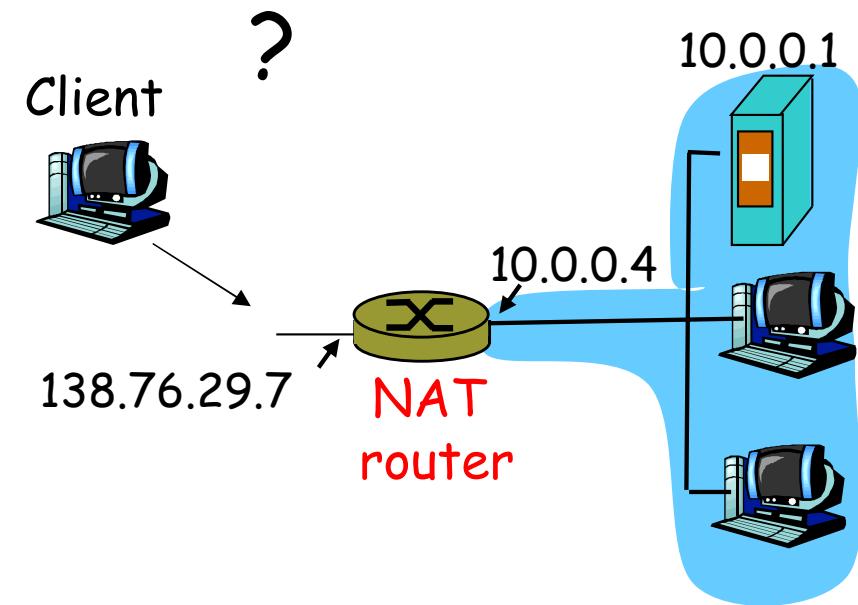


# ***Network Address Translation (NAT)***

- 16-bit port-number field:
  - > 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - Routers should only process up to layer 3;
  - Violates end-to-end argument:
    - NAT possibility must be taken into account by application designers, e.g., P2P applications;
    - Address shortage should instead be solved by IPv6...
- NAT is here to stay:
  - Extensively used in home and institutional nets, 4G/5G cellular nets

# NAT Traversal Problem

- Client wants to connect to server with address 10.0.0.1:
  - Server address 10.0.0.1 is local to LAN (client can't use it as destination address);
  - Only one externally visible NATted address: 138.76.29.7.
- Solution 1: statically configure NAT to forward incoming connection requests at given port to server:
  - E.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000.

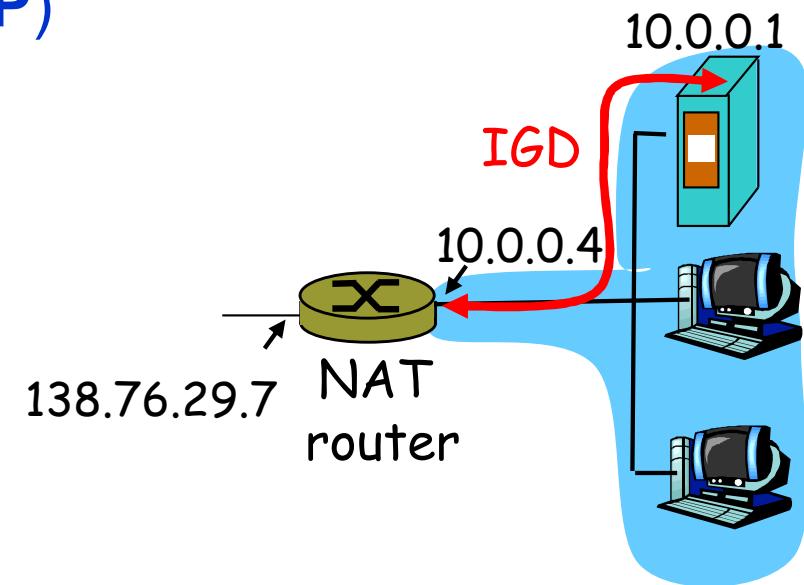


# NAT Traversal Problem

- Solution 2: Universal Plug and Play (UPnP)  
Internet Gateway Device (IGD) Protocol.

Allows NATted host to:

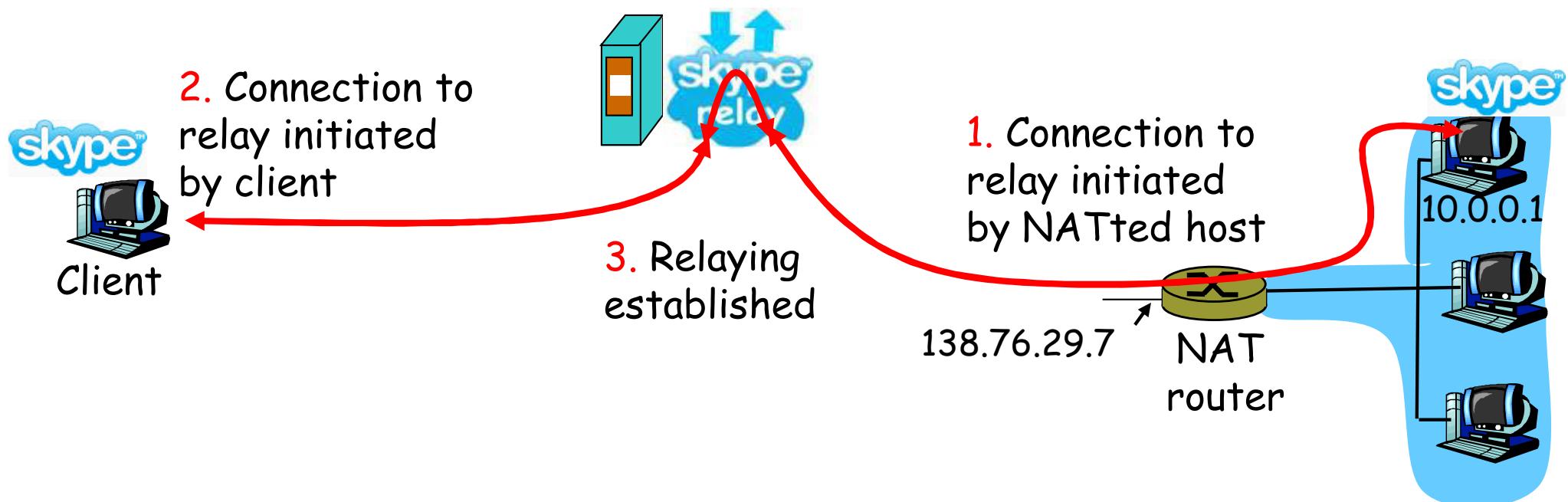
- ❖ Learn public IP address (138.76.29.7);
- ❖ Add/Remove port mappings  
(with lease times);



It automates static NAT port map configuration.

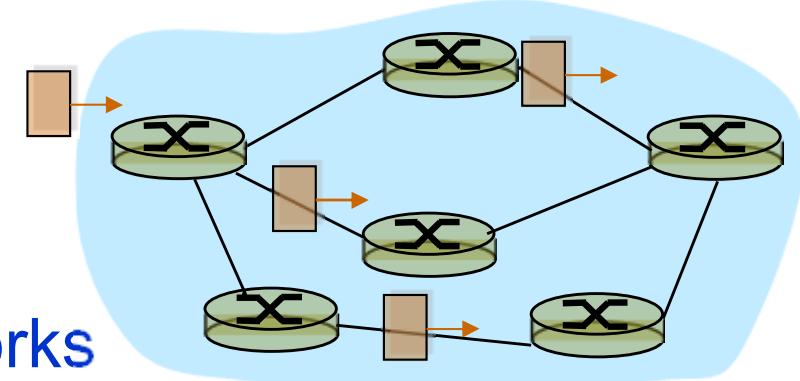
# NAT Traversal Problem

- Solution 3 – relaying (used in Skype):
  - NATed client establishes connection to relay;
  - External client connects to relay;
  - Relay bridges packets between two connections.



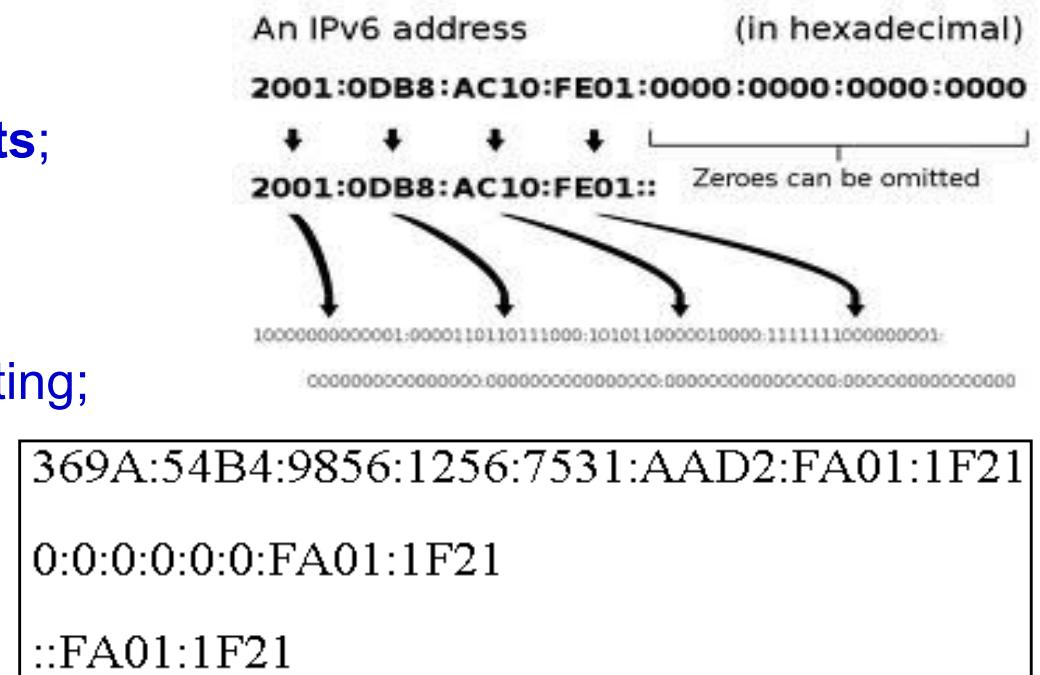
# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# IP v6 Addresses

- IP v4 addresses (32 bits) shortage:
  - Network and station components “waste” addresses;
  - Stations “use” addresses even when not connect to the Internet;
  - Internet keeps growing.
- IP v6:
  - New IP generation;
  - Addresses represented using **128 bits**;
  - Auto-configuration of addresses;
  - Header: simpler and more efficient;
  - Allow fast packet forwarding and routing;
  - Quality of service (QoS);
  - Authentication and encryption.



# IPv6 Unicast Addresses

There are four types of unicast addresses:

- **Global unicast** – conventional, publicly routable address (like IPv4 public addresses);
- **Link-local** – similar to private IPv4 addresses, to be used inside a single network segment.
- **Unique local** – also for private addressing, with the addition of being unique – joining two subnets will not cause address collisions.
- **Special** – loopback addresses; IPv4-address mapped spaces; 6-to-4 addresses (for crossing from an IPv4 network to an IPv6 network).

# *Types of Addresses and Assigned Prefixes*

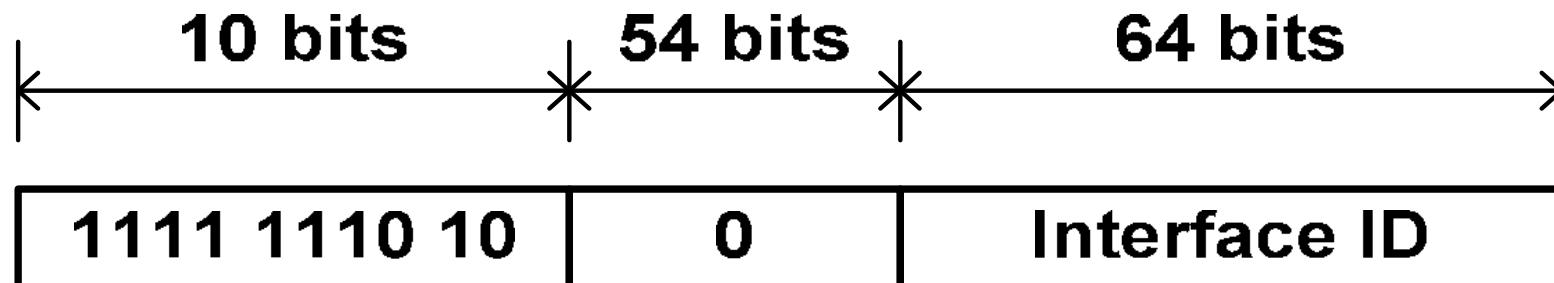
Allocation	Prefix binary	Prefix hex
Unassigned	0000 0000	::/8
Reserved	0000 001	0200/7
Loopback address		::1/128
Global unicast	001	2000::/3
Link-local unicast	1111 1110 10	FE80::/10
Reserved (formerly site-local unicast)	1111 1110 10	FEC0::/10
Local IPv6 address	1111 110	FC00::/7
Private administration	1111 1101	FD00::/8
Multicast	1111 1111	FF00::/8

Local IPv6 addresses = IPv4 private addresses;

- Not routable in the global IPv6 Internet.

## *Link-Local Unicast*

- For use on a single link
- Not routable
- Allow address autoconfiguration



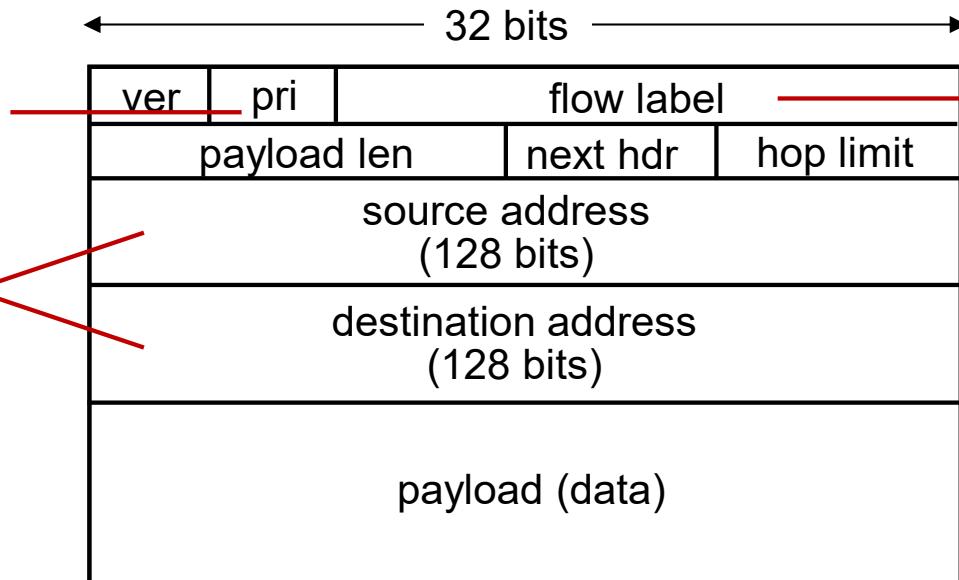
# *Address Configuration*

- Manual configuration
- Using DHCPv6
- Auto-configuration using the interface ID (last 64 bits of IPv6 unicast address)
  - Can be based on MAC address
  - Can be random number!
  - Network prefix always given by router

# IPv6 datagram format

**priority:** identify priority among datagrams in flow

**128-bit**  
IPv6 addresses



**flow label:** identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

## *Other Changes from IPv4*

- *Checksum:*
  - Removed to reduce processing time at each hop.
- *Options:*
  - Allowed, but outside of header;
  - Indicated by “Next Header” field.
- *ICMPv6* (new version of ICMP):
  - Additional message types, e.g. “*Packet too Big*”;
  - Multicast group management functions.

## *Other Changes from IPv4*

- *Fragmentation:*

IPv6 routers do not fragment, but drop the packets that are larger than the MTU (*min MTU = 1280 bytes*).

**IPv6 hosts are required to determine the optimal Path MTU before sending packets.**

To send a packet larger than the path MTU:

- ***sending node splits the packet into fragments;***
- *Fragment extension header* carries the information necessary to reassemble the original (i.e., unfragmented) packet (including *offset* value in bytes and the *more fragments* flag).



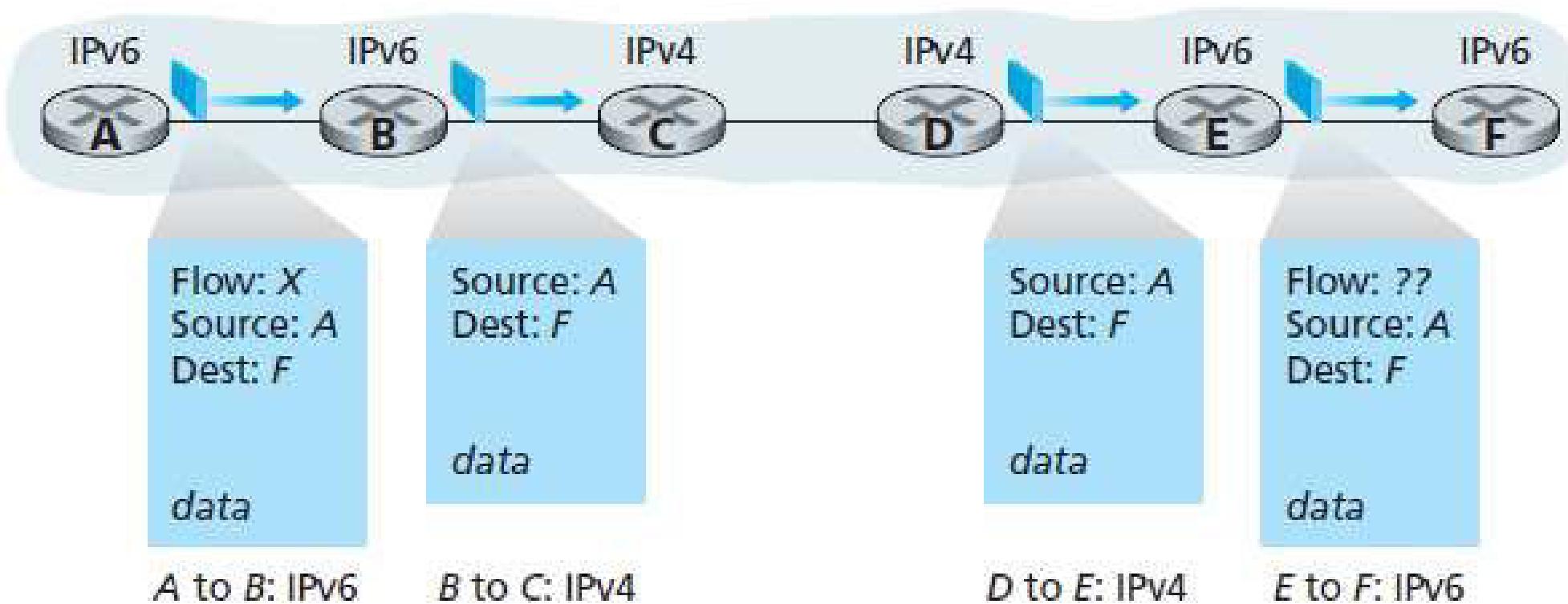
# Transition From IPv4 To IPv6



- Not all routers can be upgraded simultaneously:
  - No “flag days”;
  - How will the network operate with mixed IPv4 and IPv6 routers?
- *Double stack*: Some routers can translate between IPv4 and IPv6 headers.
- *Tunneling*: IPv6 carried as payload in IPv4 datagram among IPv4 routers.

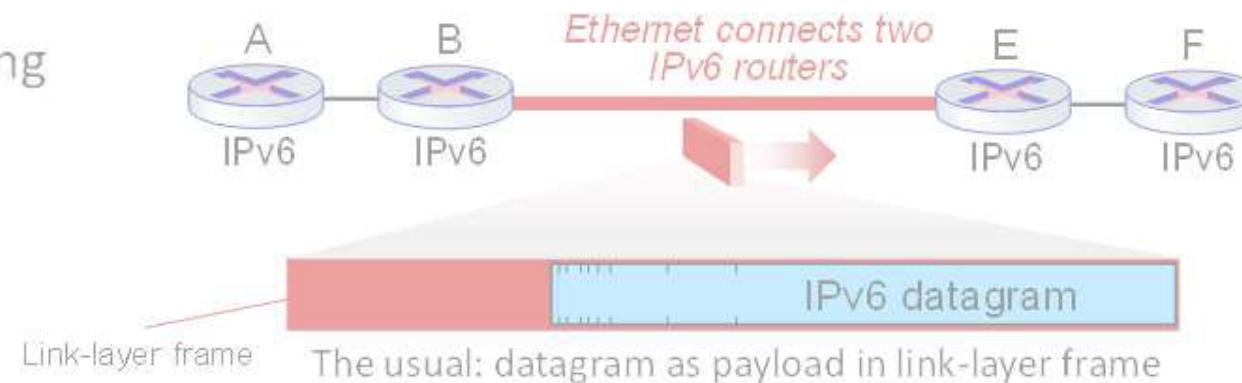
# Double Stack

**Double Stack:** Some information present in the original IPv6 Header is lost when translating to an IPv4 datagram!

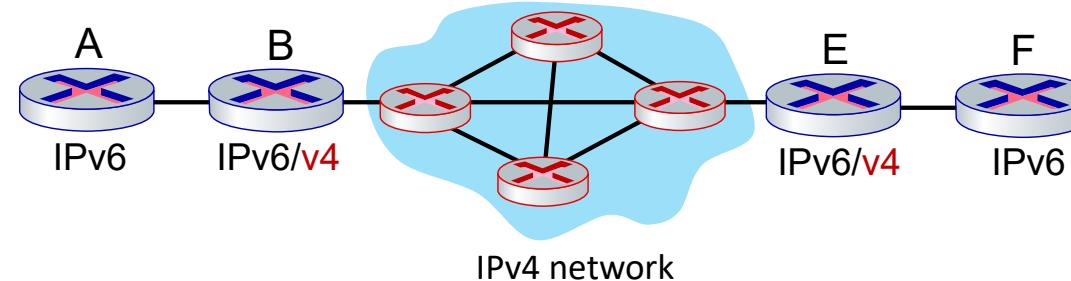


# Tunneling and Encapsulation

Ethernet connecting  
two IPv6 routers:

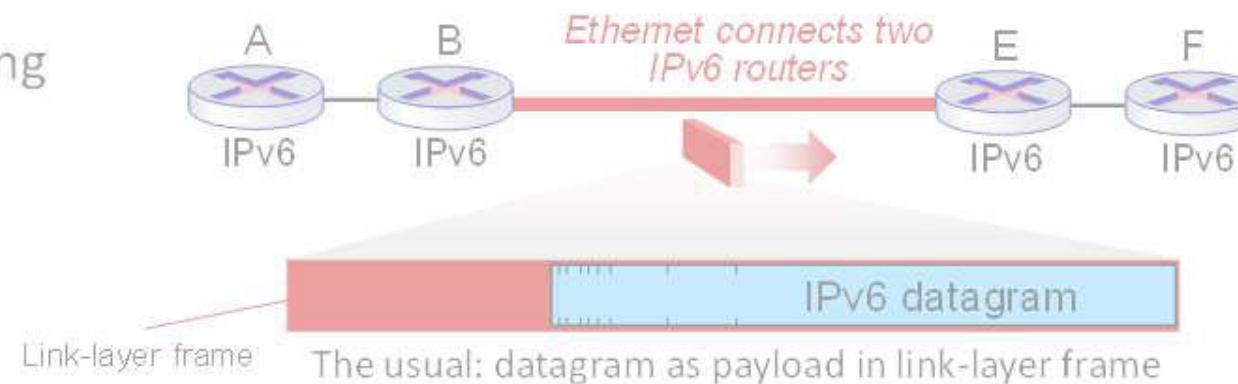


IPv4 network  
connecting two  
IPv6 routers

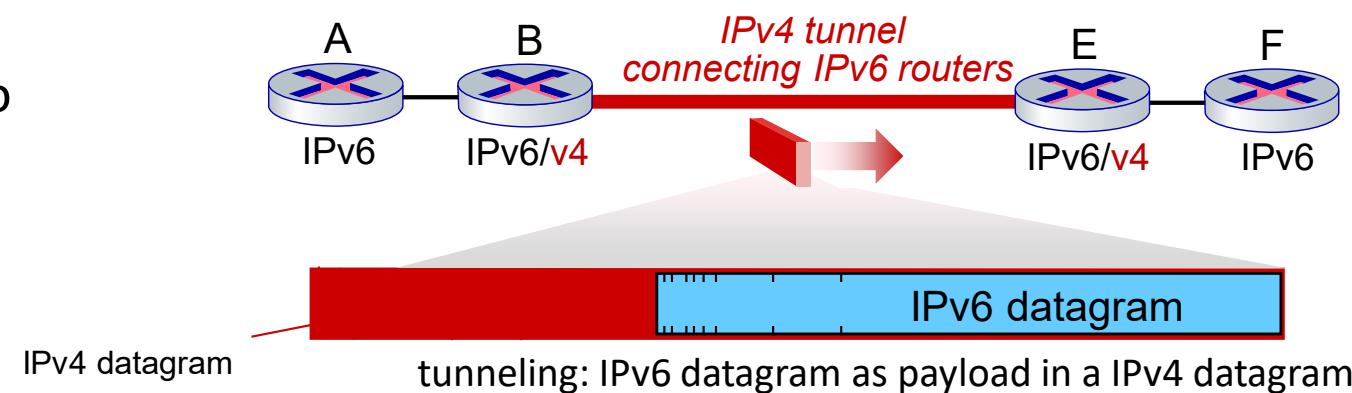


# Tunneling and Encapsulation

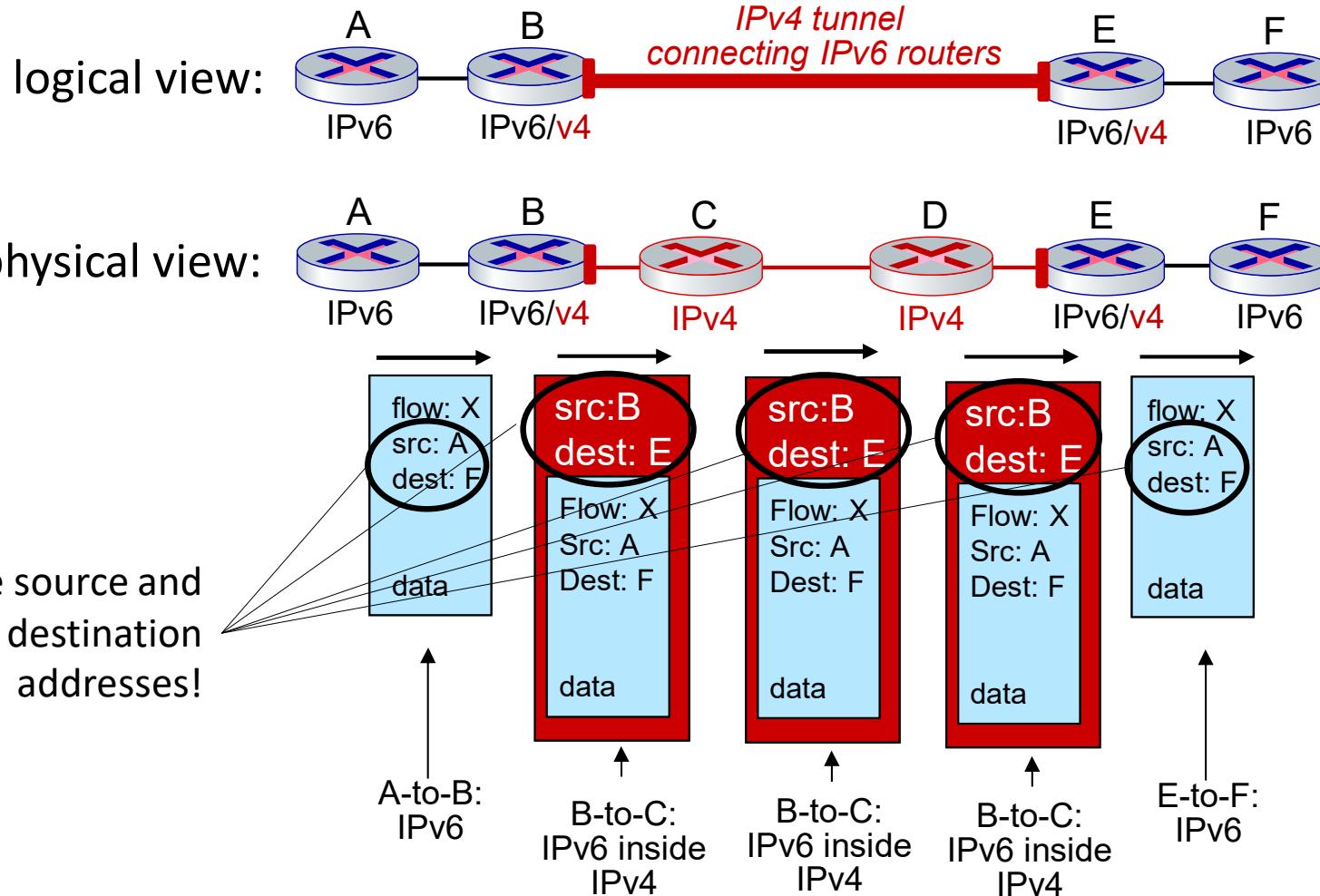
Ethernet connecting  
two IPv6 routers:



IPv4 tunnel  
connecting two  
IPv6 routers



# Tunneling

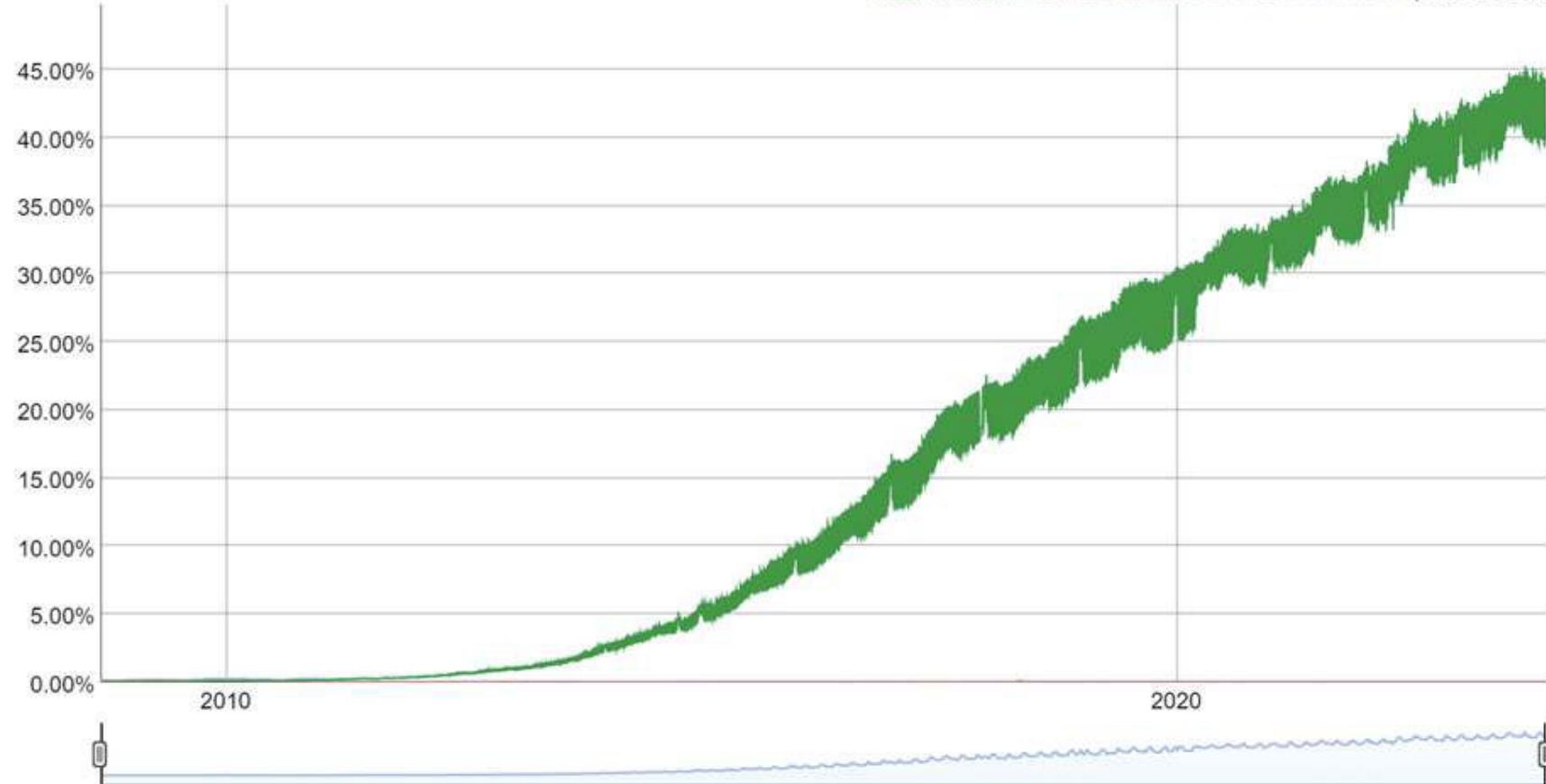


# IPv6 Adoption

## IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.

Native: 41.24% 6to4/Teredo: 0.00% Total IPv6: 41.24% | Nov 22, 2023

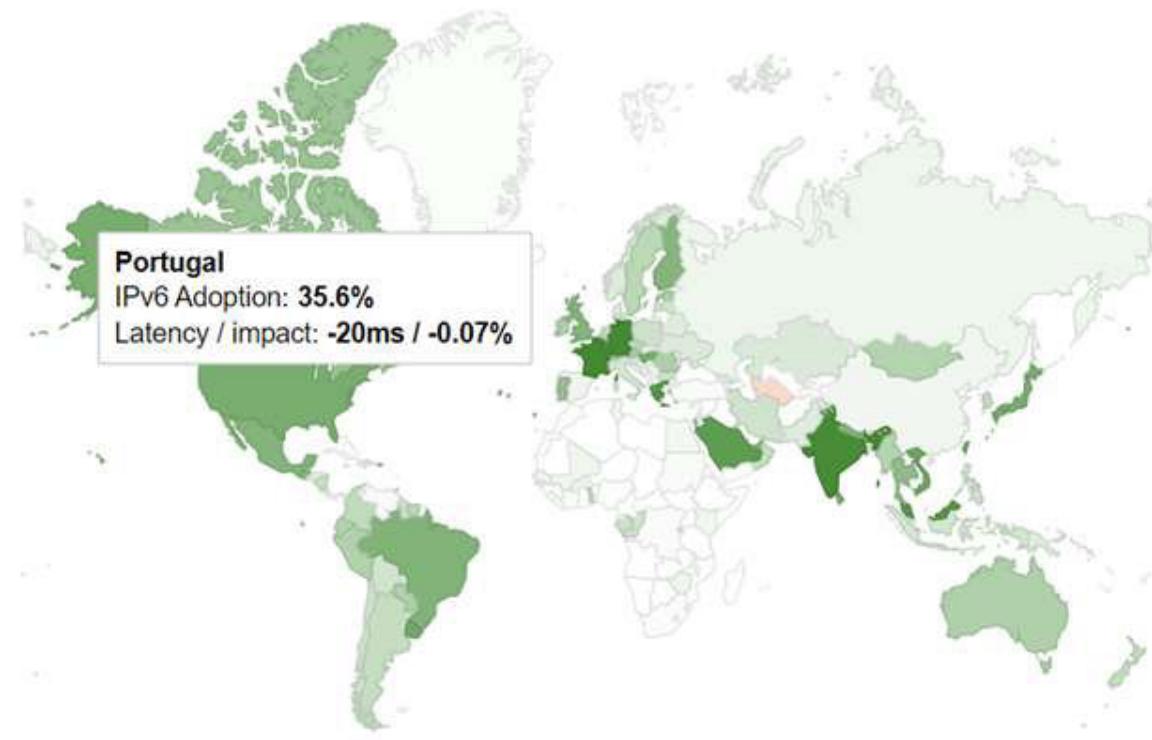


Google: ~ 40% of clients access services via IPv6

<https://www.google.com/intl/en/ipv6/statistics.html>

# IPv6 Adoption

Per-Country IPv6 adoption



[World](#) | [Africa](#) | [Asia](#) | [Europe](#) | [Oceania](#) | [North America](#) | [Central America](#) | [Caribbean](#) | [South America](#)

The chart above shows the availability of IPv6 connectivity around the world.

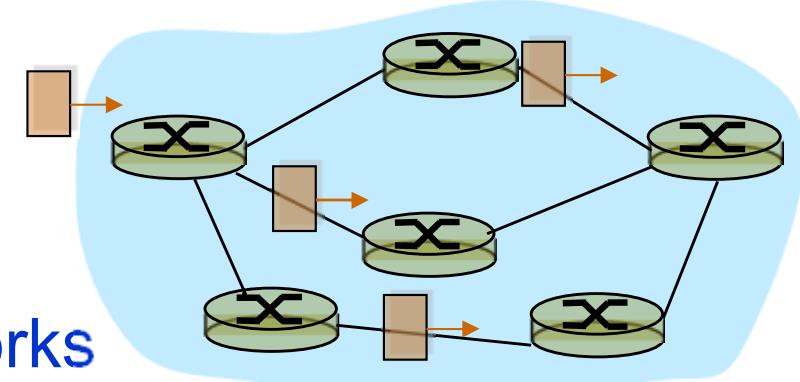
- Regions where IPv6 is more widely deployed (the darker the green, the greater the deployment) and users experience infrequent issues connecting to IPv6-enabled websites.

## Google: ~ 40% of clients access services via IPv6

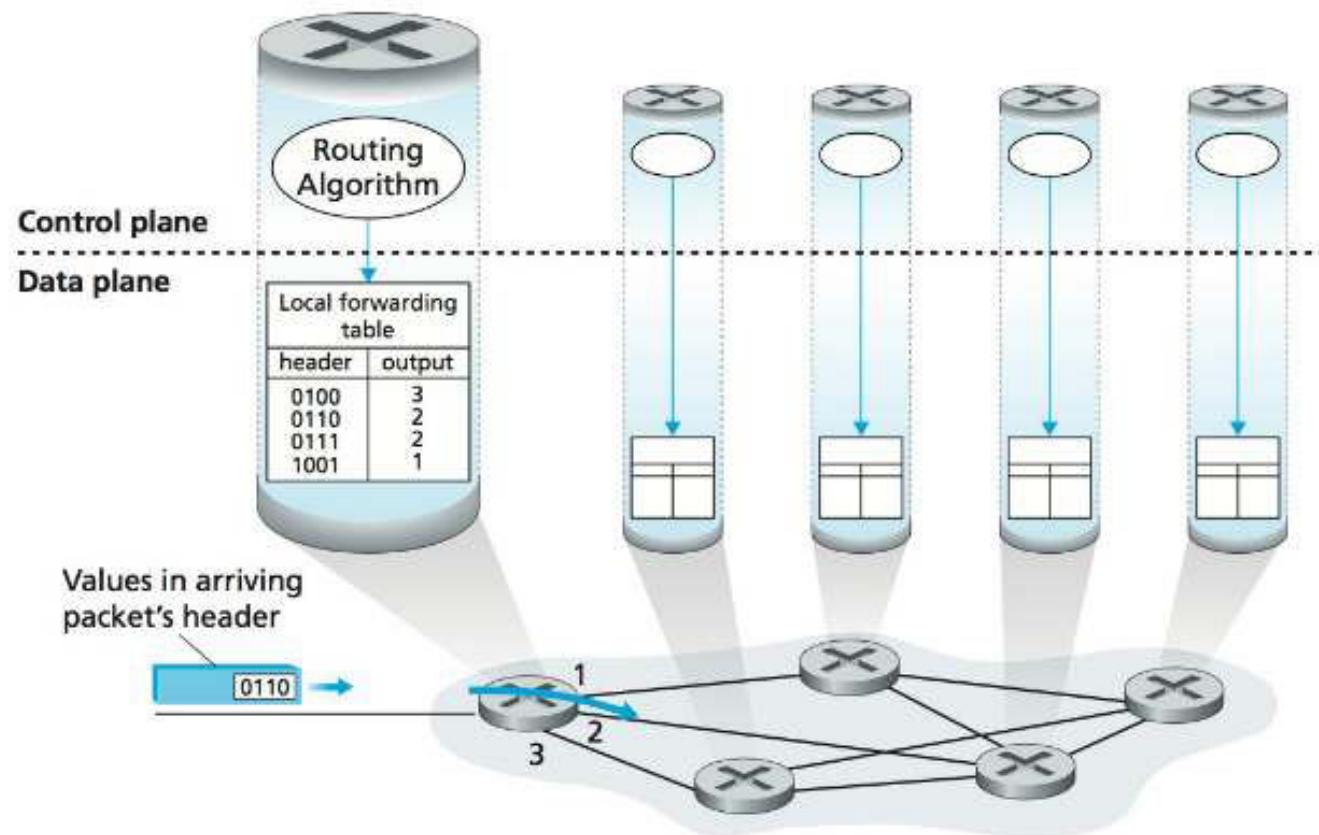
<https://www.google.com/intl/en/ipv6/statistics.html>

# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



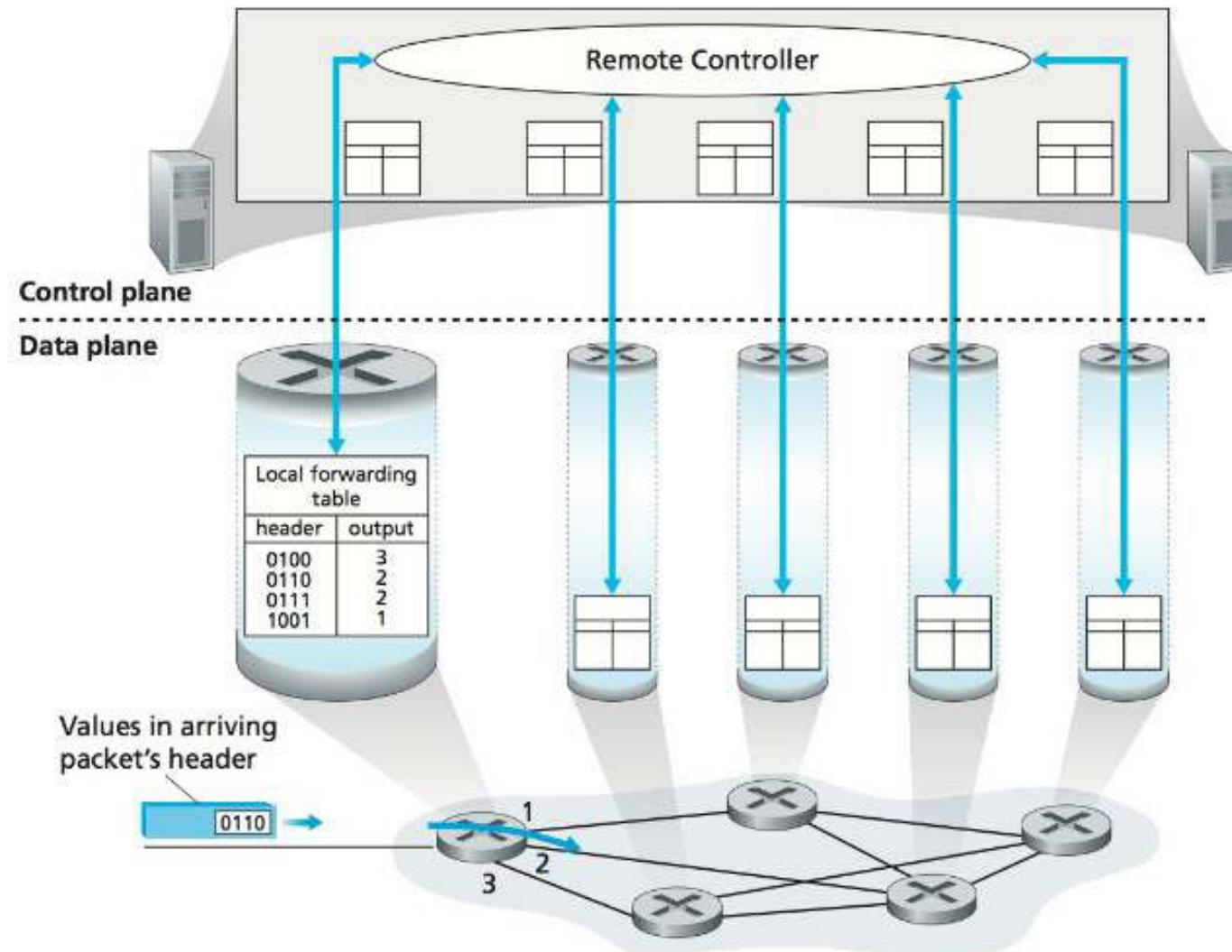
# The Internet Network Layer



**Figure 4.2** • Routing algorithms determine values in forward tables

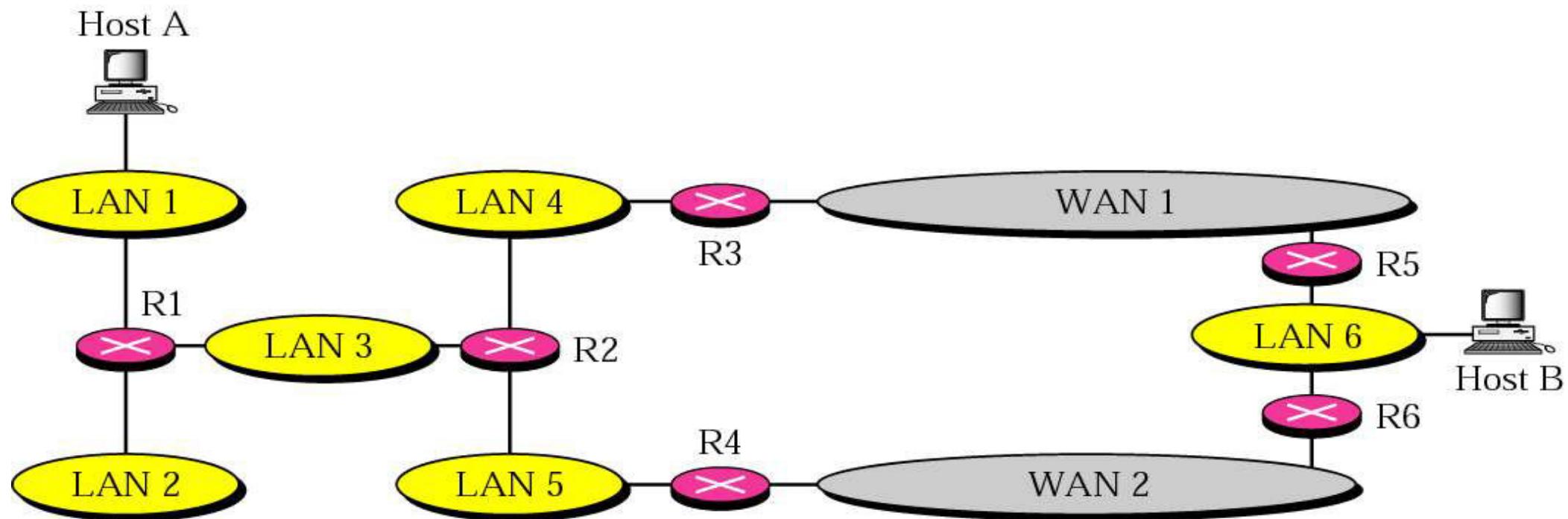
Traditional router includes control and data planes.

# The Internet Network Layer



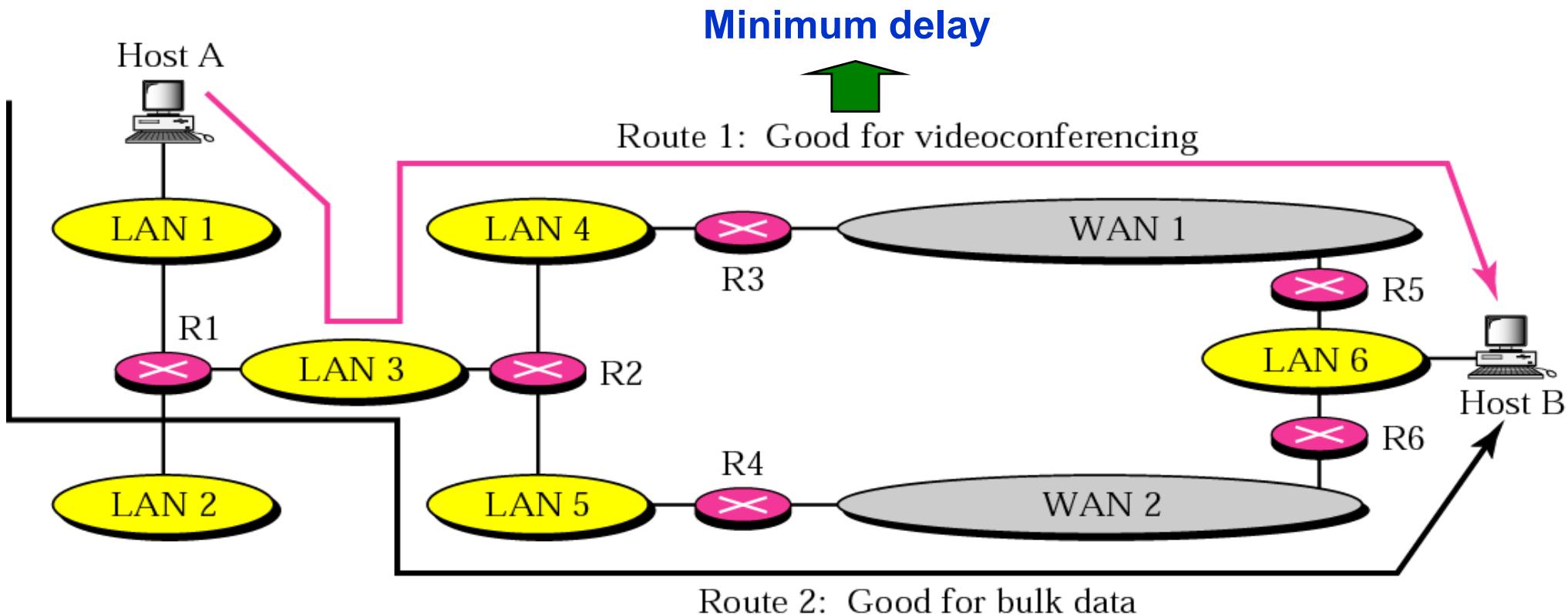
SDN (software defined network) router separates control plane (remote) from data plane (local).

# Routing



Which is the best path from A to B?

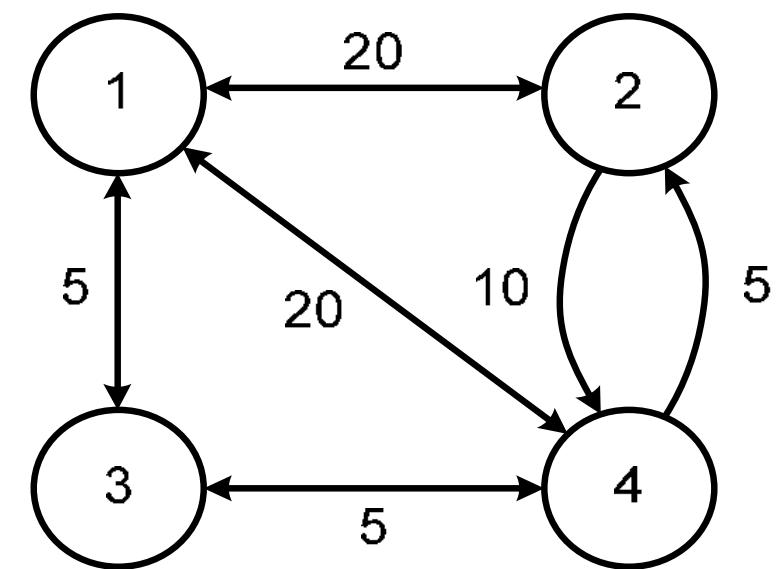
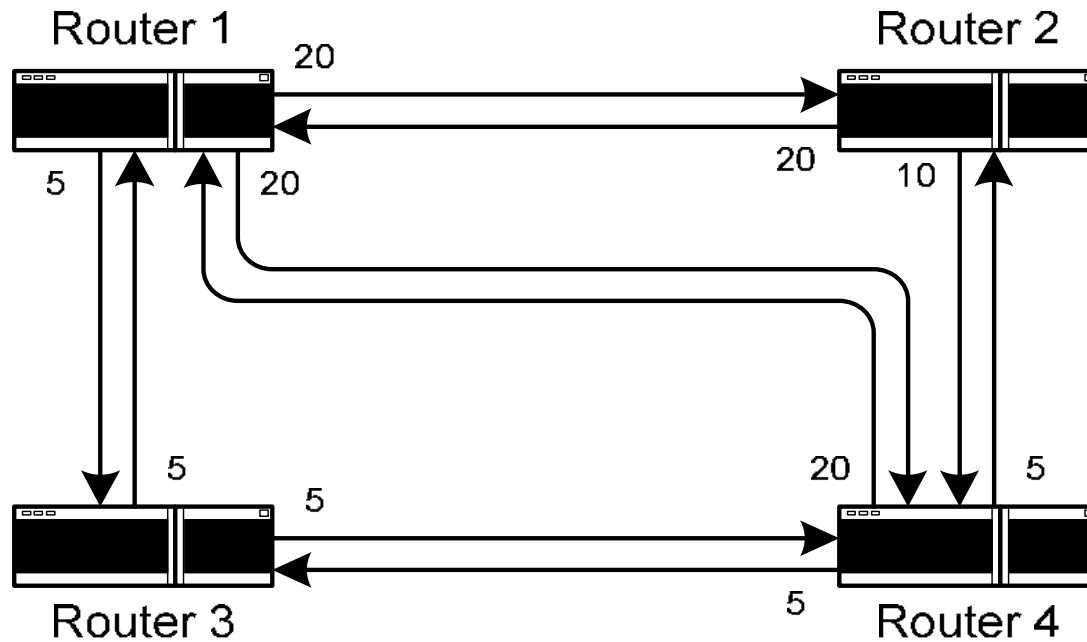
# Routing



Best path depends on the type of service.

For instance, minimal delay possible vs. reliability.

# Representation of Networks Through Graphs

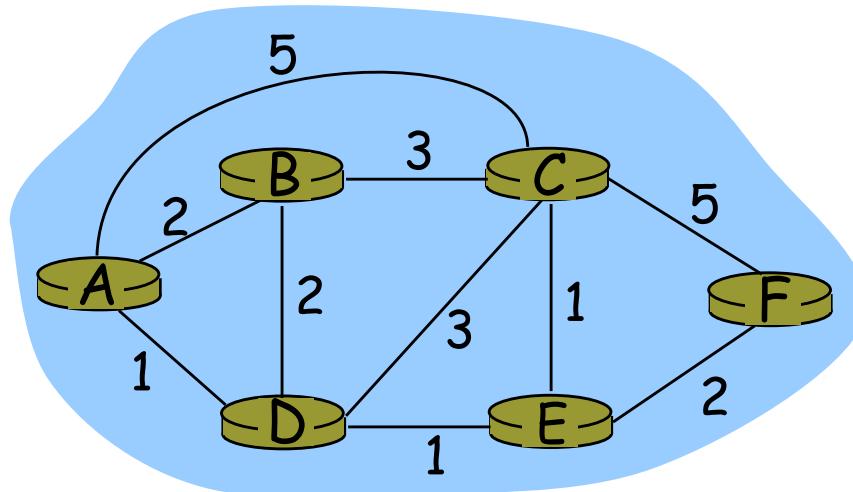


$$G = (N, L)$$

$$N = \{1, 2, 3, 4\}$$

$$L = \{(1,2), (2,1), (1,3), (3,1), (1,4), (4,1), (2,4), (4,2), (3,4), (4,3)\}$$

## Graph Abstraction: Costs



- $c(x, x') = \text{cost of link } (x, x')$ 
  - e.g.,  $c(C, F) = 5$
- Cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path  $(x_1, x_2, x_3, \dots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

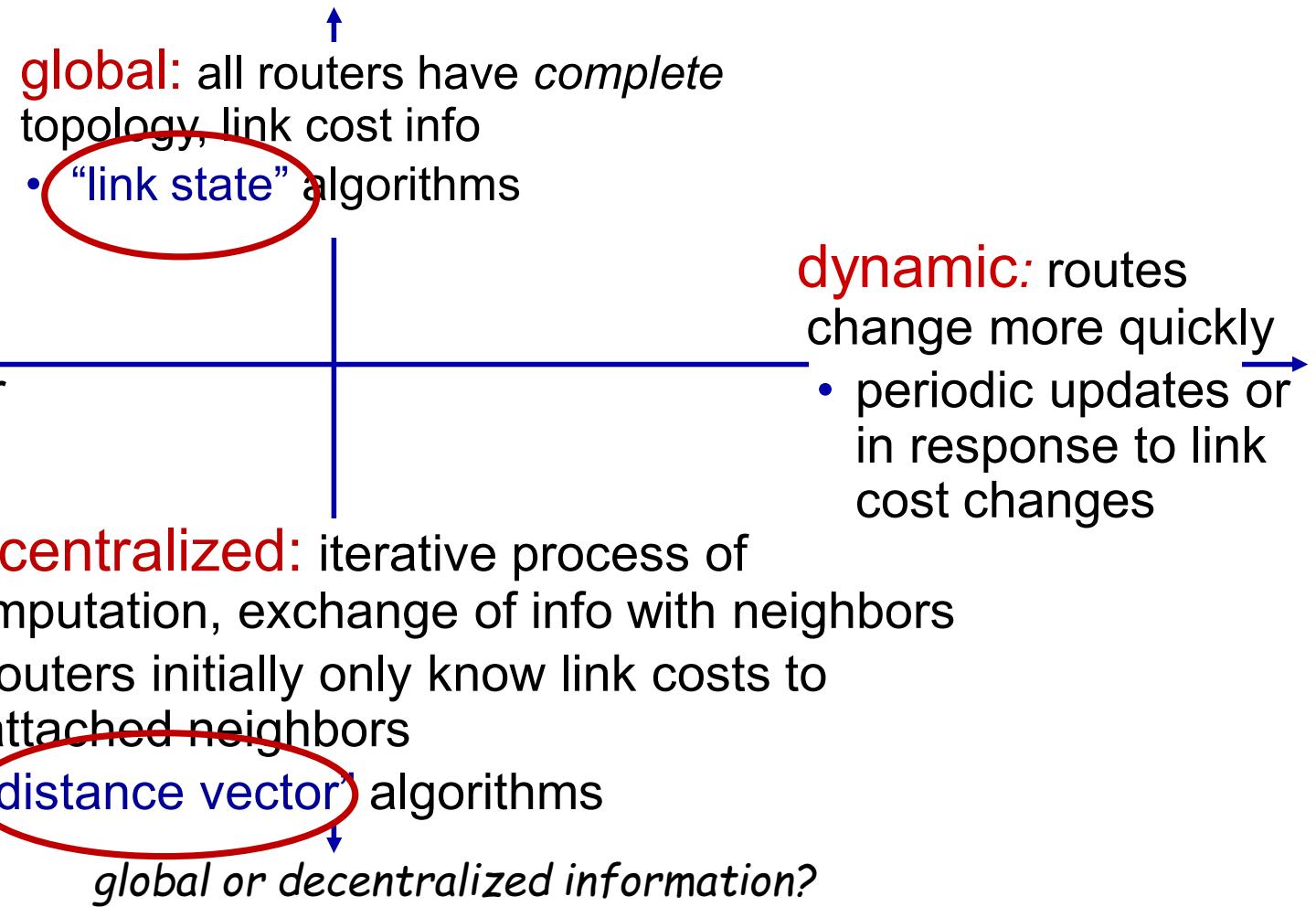
**Question: What's the least-cost path between A and F ?**

**Routing algorithm:** algorithm that finds least-cost path.

# Routing Algorithm Classification

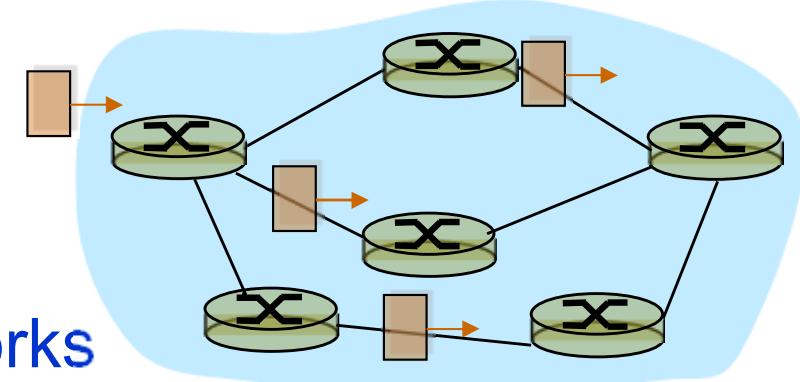
How fast do routes change?

**static:** routes change slowly over time



# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# A Link-State Routing Algorithm

## Dijkstra's algorithm

- Net topology and link costs known to all nodes:
  - Accomplished via “link state packet (LSP) broadcast”.
  - All nodes have the same information.
- Computes least cost paths from one node (“source”) to all other nodes:
  - Obtains **forwarding table** for that node.
- Iterative: after k iterations, know least cost path to k destinations.

## Notation:

**c(x,y)**: Link cost from node x to y;  $= \infty$  if not direct neighbors.

**D(v)**: Current value of cost of path from source to destination V.

**p(v)**: Predecessor node along path from source to V.

**N'**: Set of nodes whose least cost path definitively known.

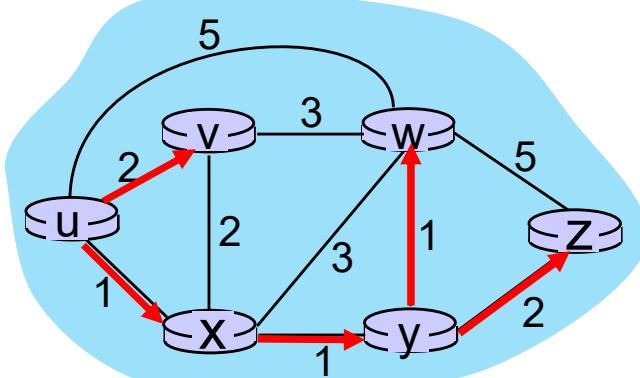
# Dijkstra's Algorithm

```
1 Initialization (computing costs for node A):  
2    $N' = \{A\}$   
3   for all nodes n  
4     if n adjacent to A  
5       then  $D(n) = c(A,n)$   
6     else  $D(n) = \infty$   
7  
8 Loop  
9   find m not in  $N'$  such that  $D(m)$  is a minimum  
10  add m to  $N'$   
11  update  $D(n)$  for all n adjacent to m and not in  $N'$  :  
12     $D(n) = \min( D(n), D(m) + c(m,n) )$   
13  /* new cost to n is either old cost to n or known  
14    shortest path cost to m plus cost from m to n */  
15 until all nodes in  $N'$ 
```

# Dijkstra's Algorithm: an example

Step	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	$\infty$	$\infty$
1	u, x	2, u	4, x	2, x	$\infty$	$\infty$
2	u, x, y	2, u	3, y	4, y	4, y	4, y
3	u, x, y, v		3, y			4, y
4	u, x, y, v, w					
5	u, x, y, v, w, z					

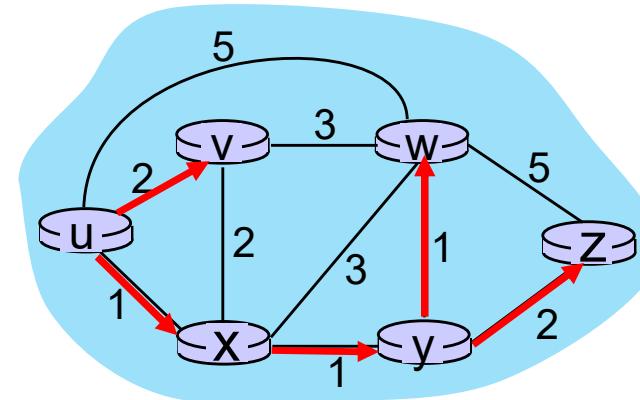
Initialization (step 0): For all  $a$ : if  $a$  adjacent to  $u$  then  $D(a) = c_{u,a}$



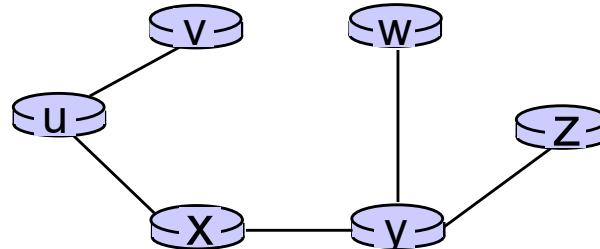
find  $a$  not in  $N'$  such that  $D(a)$  is a minimum  
 add  $a$  to  $N'$   
 update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$ :  

$$D(b) = \min ( D(b), D(a) + c_{a,b} )$$

# Dijkstra's Algorithm: an example



Resulting least-cost-path tree from u: Resulting forwarding table in u:



destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

route from u to v directly  
route from u to all other destinations via x

# Dijkstra's Algorithm: Discussion

## Algorithm complexity: $n$ nodes

- each of  $n$  iteration: need to check all nodes,  $w$ , not in  $N$
- $n(n+1)/2$  comparisons:  $O(n^2)$  complexity
- more efficient implementations possible:  $O(n \log n)$

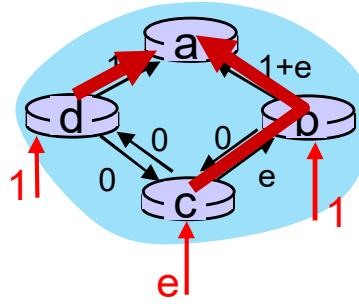
## Message complexity:

- each router must *broadcast* its link state information to other  $n$  routers
- efficient (and interesting!) broadcast algorithms:  $O(n)$  link crossings to disseminate a broadcast message from one source
- each router's message crosses  $O(n)$  links: overall message complexity:  $O(n^2)$

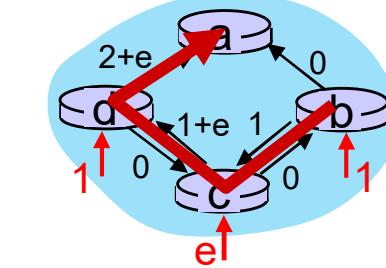
# Dijkstra's Algorithm: Oscillations Possible

TPC: Prob. 10

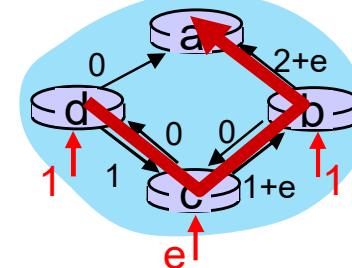
- When link costs depend on traffic volume, **route oscillations** possible
- Sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1,  $e$  ( $<1$ ), 1
  - link costs are directional, and volume-dependent



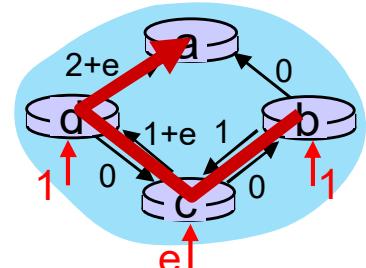
initially



given these costs,  
find new routing....  
resulting in new costs



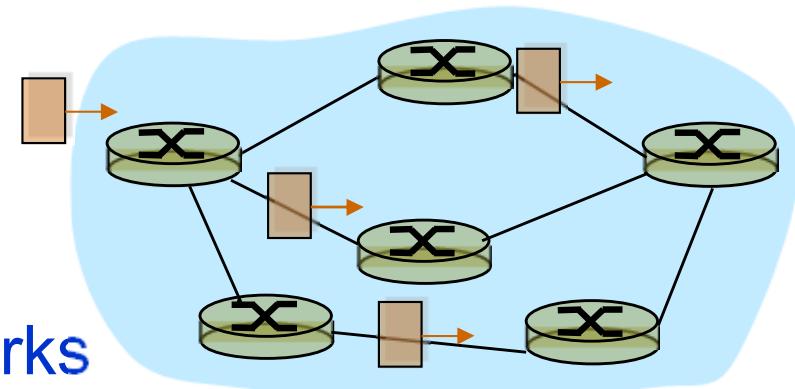
given these costs,  
find new routing....  
resulting in new costs



given these costs,  
find new routing....  
resulting in new costs

# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- What's inside a router
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# Distance Vector Algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

**Bellman-Ford equation**

Let  $D_x(y)$ : cost of least-cost path from  $x$  to  $y$ .

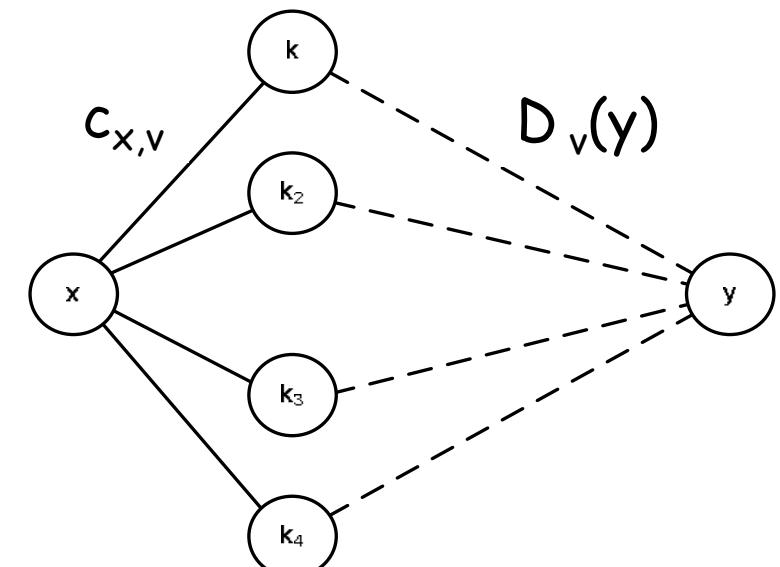
Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

*min* taken over all  
neighbors  $v$  of  $x$

$v$ 's estimated  
least-cost-path  
cost to  $y$

direct cost of link from  $x$  to  $v$



# Distance Vector Algorithm

## Basic idea:

- From time-to-time, each node sends its own distance vector (DV) estimate to its neighbors.
- Asynchronous distribution of distance vector estimates.
- When a node  $x$  receives new DV estimate from a neighbor, it updates its own DV using the Bellman-Ford equation:

$$D_x(y) \leftarrow \min_k \{c(x,k) + D_k(y)\} \quad \text{for each node } y \in N$$

- Under natural conditions, with minor changes, the estimate  $D_x(y)$  converges to the actual least costs:  $d_x(y)$ .

# Distance Vector Algorithm

## Iterative, asynchronous

Local iterations are caused by:

- Local link cost change;
- DV update message from neighbor.

## Distributed

- Each node notifies neighbors *only* when its DV changes:
  - Neighbors then notify their neighbors, if necessary.

Each node:

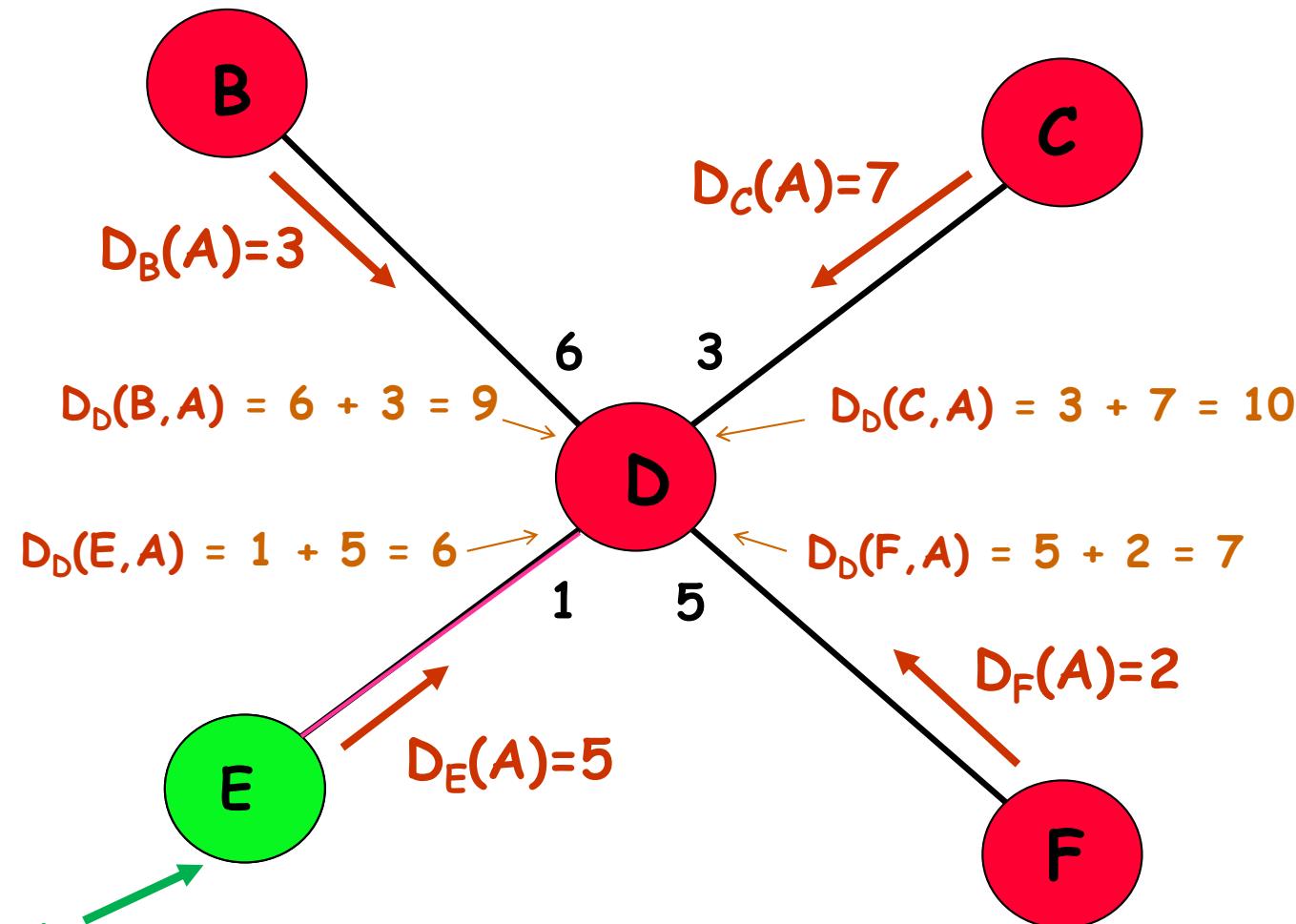
*wait* for (change in local link cost or message from neighbor);

*recompute* estimates;

If DV to any destination has changed, *notify* neighbors.

## From centralized to distributed routing

Best path from D to A?

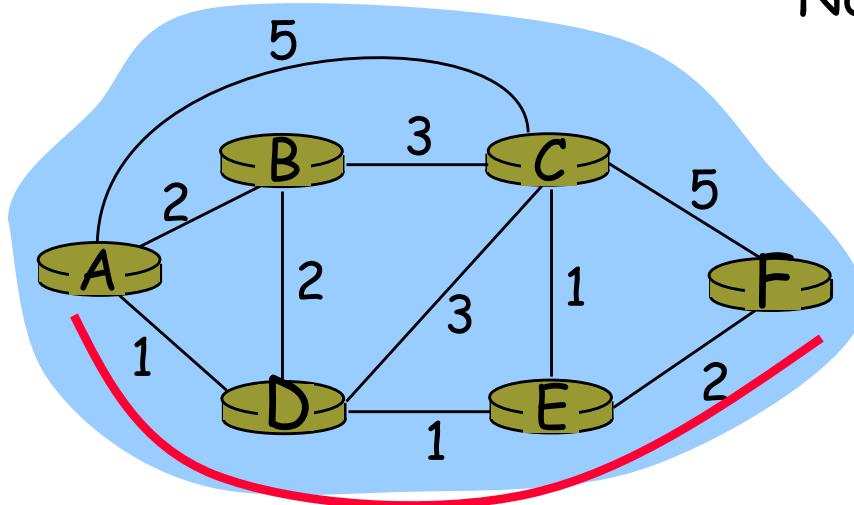


Selected next hop

$$D_D(A) = 6$$

$D_D(k, A) = c(D, k) + Dk(A)$  - distance estimate from D to A, via neighbor k

## Bellman-Ford Example



Node A -  $D_A(F)$ :

Clearly,  $D_B(F) = 5$ ,  $D_D(F) = 3$ ,  $D_C(F) = 3$

Bellman-Ford equation says:

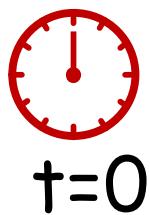
$$\begin{aligned}
 D_A(F) &= \min \{ c(A,B) + D_B(F), \\
 &\quad c(A,D) + D_D(F), \\
 &\quad c(A,C) + D_C(F) \} \\
 &= \min \{ 2 + 5, \\
 &\quad 1 + 3, \\
 &\quad 5 + 3 \} = 4
 \end{aligned}$$

Node that achieves minimum is next hop in shortest path → forwarding table

Node A:

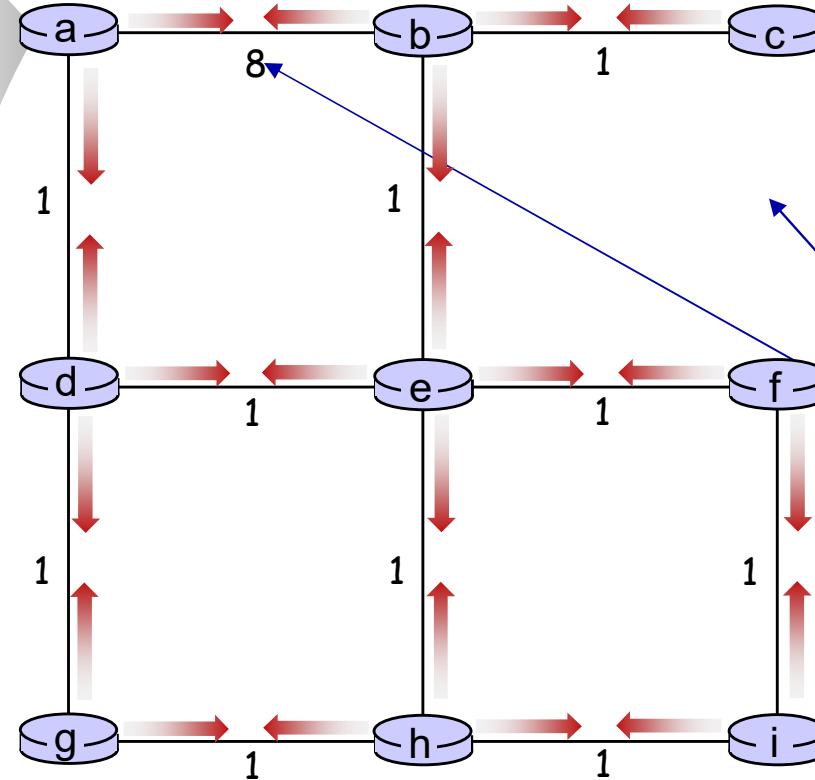
Destination	Next router
F	D

# Distance vector: example



- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

DV in
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



- A few asymmetries:
- missing link
  - larger cost

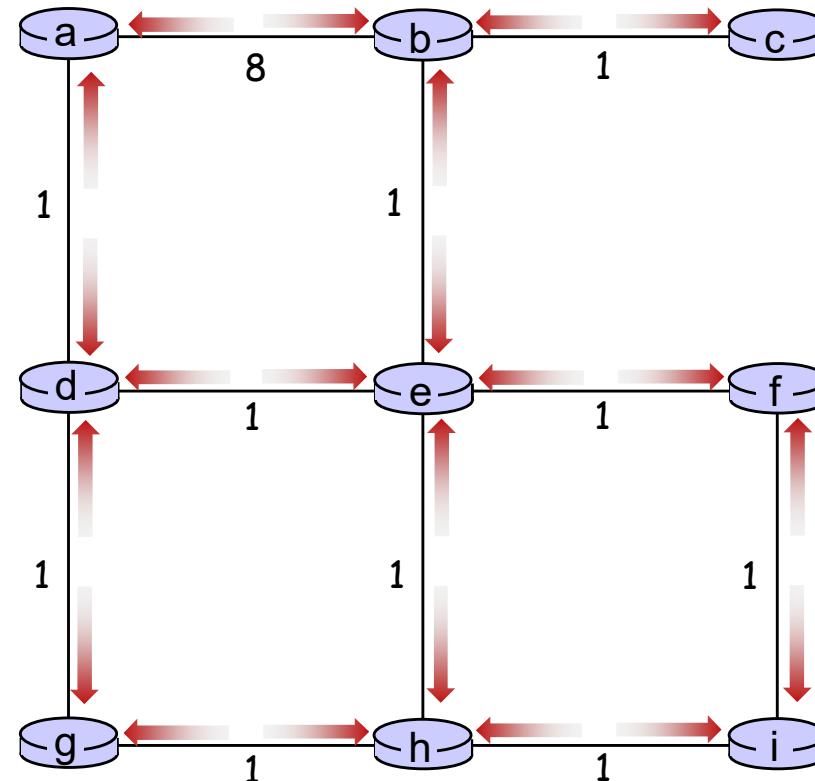
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



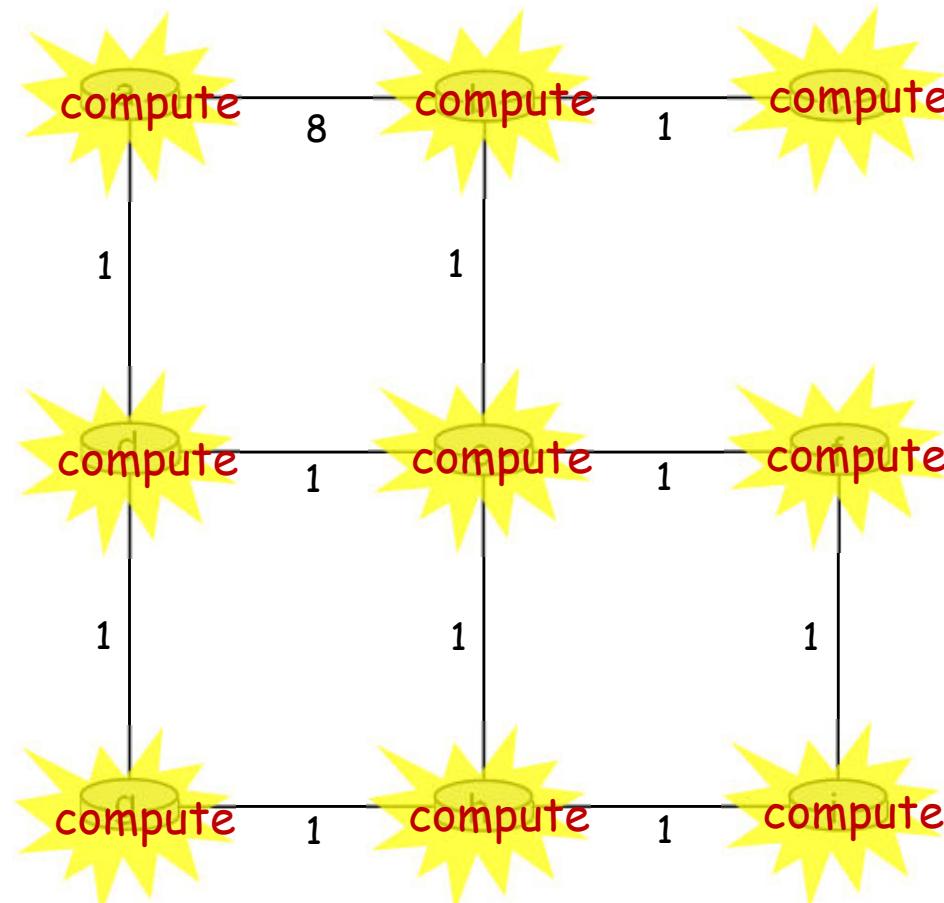
## Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



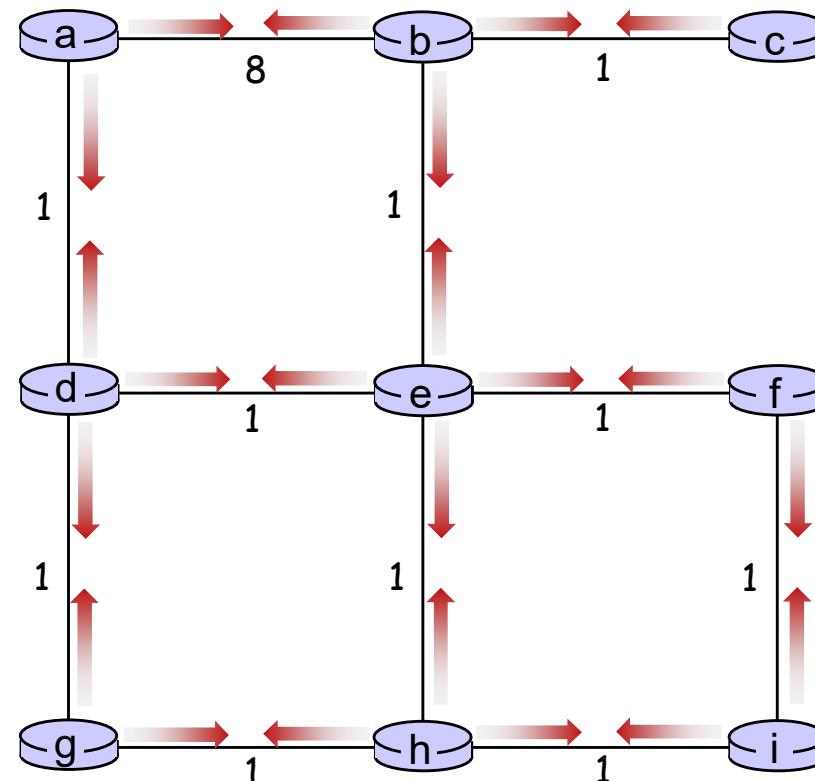
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



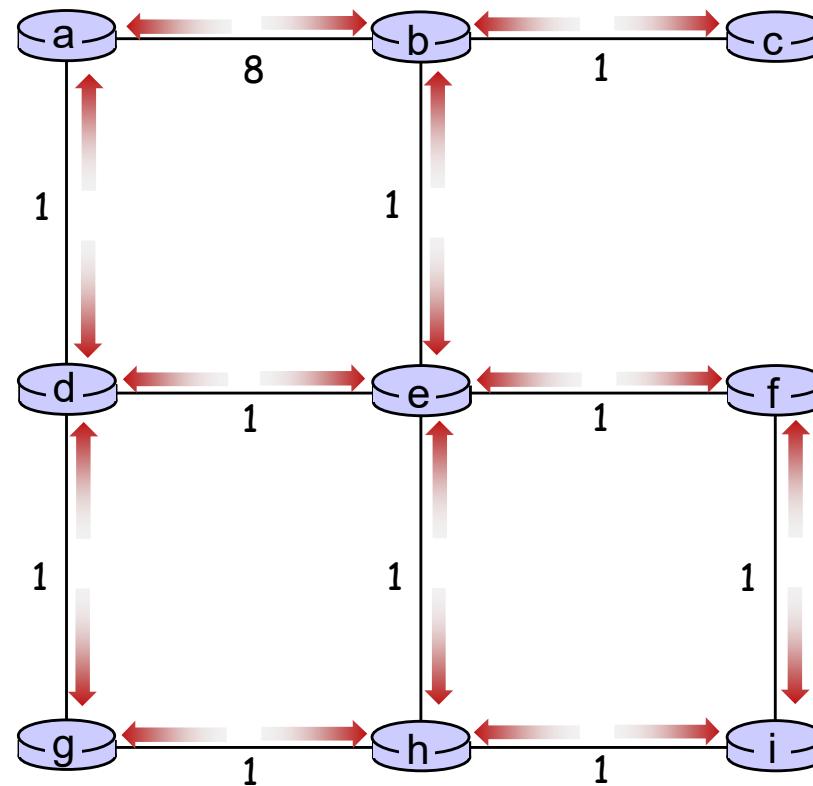
# *Distance vector example: iteration*



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



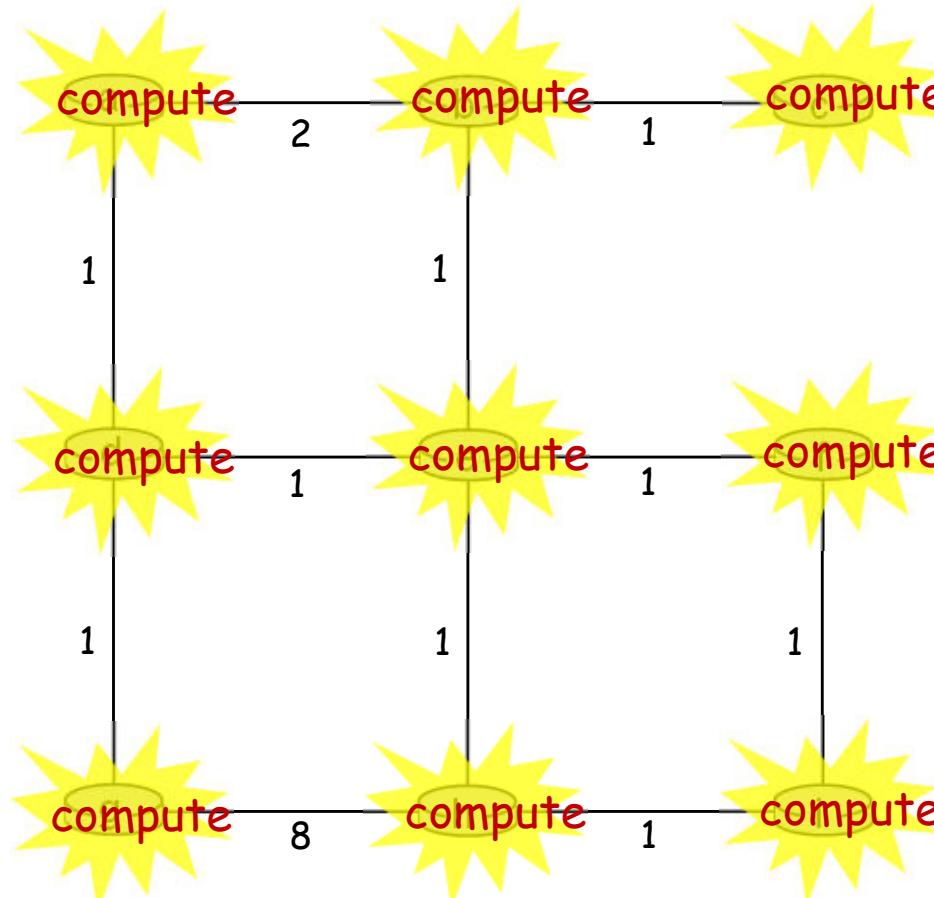
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



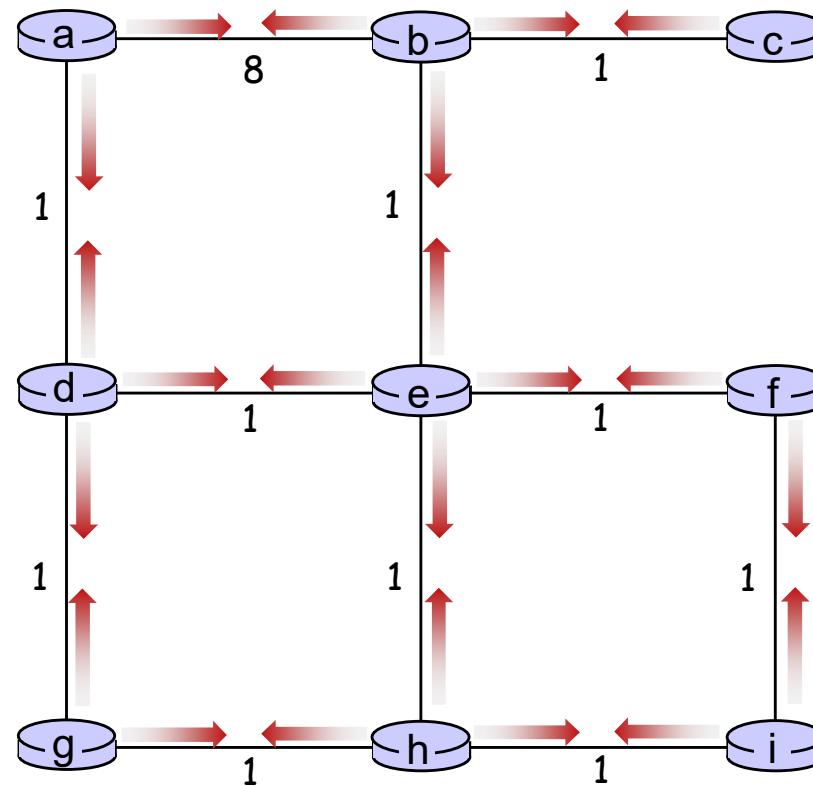
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



## *Distance vector example: iteration*

.... and so on

Let's next take a look at the iterative computations at nodes

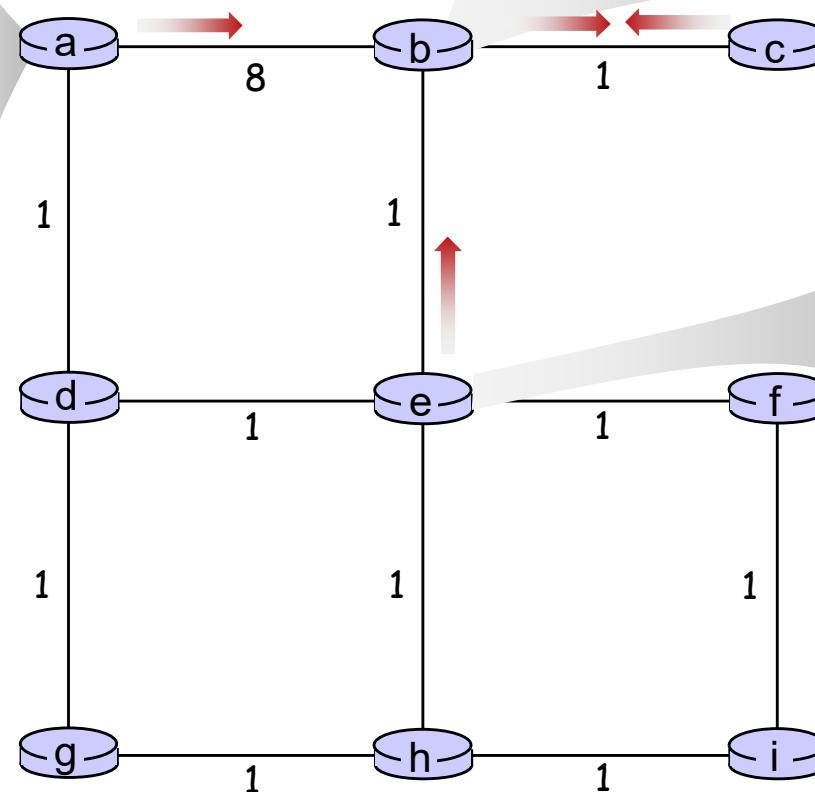


$t=1$

- c receives DVs from b

## Distance vector example: computation

DV in a:	
$D_a(a) = 0$	
$D_a(b) = 8$	
$D_a(c) = \infty$	
$D_a(d) = 1$	
$D_a(e) = \infty$	
$D_a(f) = \infty$	
$D_a(g) = \infty$	
$D_a(h) = \infty$	
$D_a(i) = \infty$	



DV in b:	
$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:	
$D_c(a) = \infty$	
$D_c(b) = 1$	
$D_c(c) = 0$	
$D_c(d) = \infty$	
$D_c(e) = \infty$	
$D_c(f) = \infty$	
$D_c(g) = \infty$	
$D_c(h) = \infty$	
$D_c(i) = \infty$	

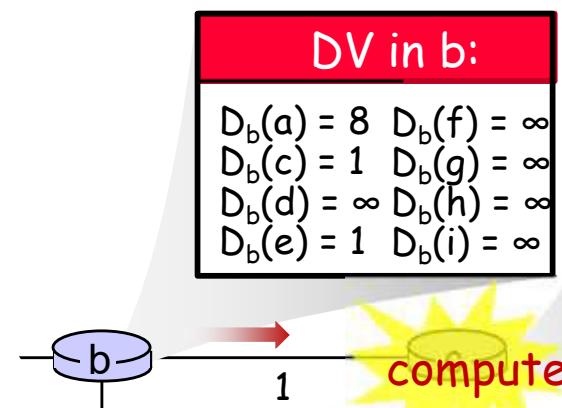
DV in e:	
$D_e(a) = \infty$	
$D_e(b) = 1$	
$D_e(c) = \infty$	
$D_e(d) = 1$	
$D_e(e) = 0$	
$D_e(f) = 1$	
$D_e(g) = \infty$	
$D_e(h) = 1$	
$D_e(i) = \infty$	



$t=1$

- c receives DVs from b  
computes:

$$\begin{aligned}
 D_c(a) &= \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9 \\
 D_c(b) &= \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1 \\
 D_c(d) &= \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty \\
 D_c(e) &= \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2 \\
 D_c(f) &= \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty \\
 D_c(g) &= \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty \\
 D_c(h) &= \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty \\
 D_c(i) &= \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty
 \end{aligned}$$



DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

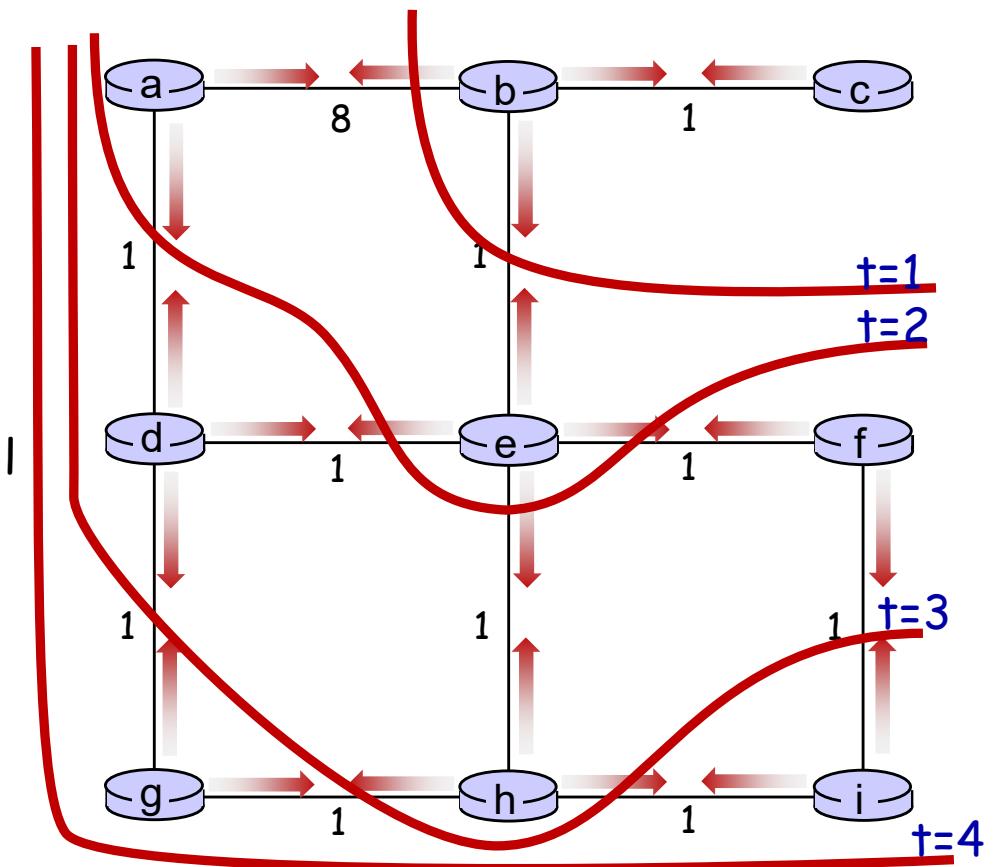
DV in c:
$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = 2$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

\* Check out the online interactive exercises for more examples:  
[http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

- ⌚ t=0 c's state at t=0 is at c only
- ⌚ t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
- ⌚ t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
- ⌚ t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at b, a, e and now at c, f, h as well
- ⌚ t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at b, a, e, c, f, h and now at g, i as well

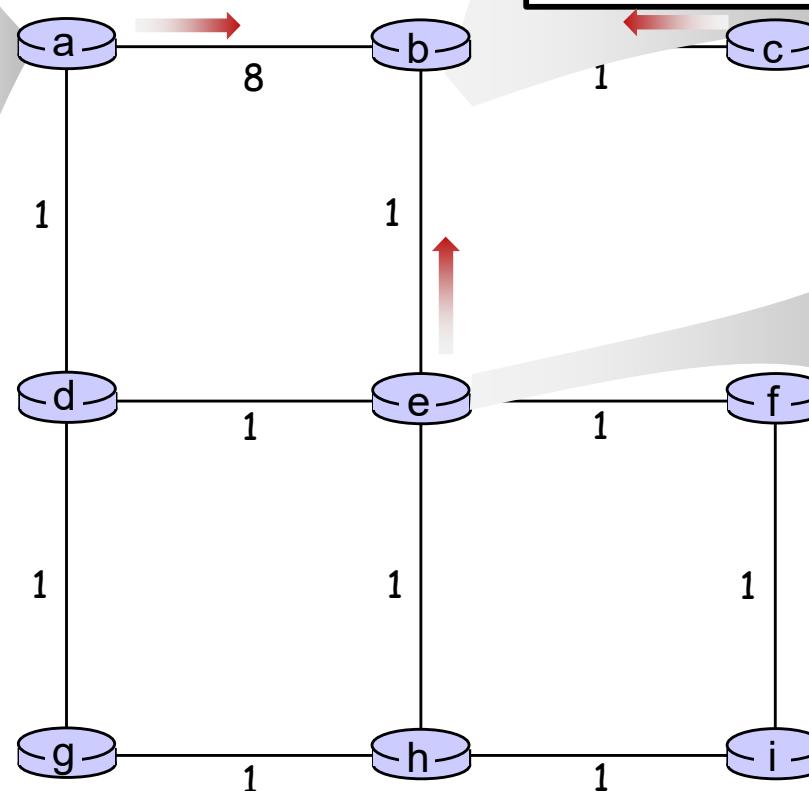




t=1

- b receives DVs from a, c, e

DV in
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



DV in b:
$D_b(a) = 8$
$D_b(c) = 1$
$D_b(d) = \infty$
$D_b(e) = 1$
$D_b(f) = \infty$
$D_b(g) = \infty$
$D_b(h) = \infty$
$D_b(i) = \infty$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

# Distance vector example: computation

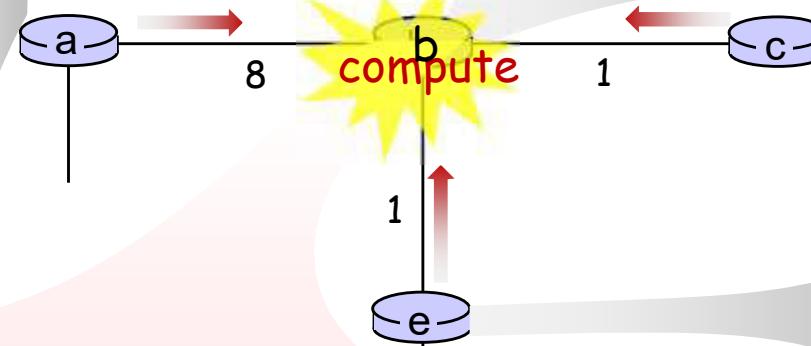


$t=1$

- b receives DVs from a, c, e, computes:

DV in a:

$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



DV in b:

$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

DV in b:

$D_b(a) = 8$	$D_b(f) = 2$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = 2$	$D_b(h) = 2$
$D_b(e) = 1$	$D_b(i) = \infty$

$$\begin{aligned}
 D_b(a) &= \min\{c_{b,a} + D_a(a), c_{b,c} + D_c(a), c_{b,e} + D_e(a)\} = \min\{8, \infty, \infty\} = 8 \\
 D_b(c) &= \min\{c_{b,a} + D_a(c), c_{b,c} + D_c(c), c_{b,e} + D_e(c)\} = \min\{\infty, 1, \infty\} = 1 \\
 D_b(d) &= \min\{c_{b,a} + D_a(d), c_{b,c} + D_c(d), c_{b,e} + D_e(d)\} = \min\{9, 2, \infty\} = 2 \\
 D_b(e) &= \min\{c_{b,a} + D_a(e), c_{b,c} + D_c(e), c_{b,e} + D_e(e)\} = \min\{\infty, \infty, 1\} = 1 \\
 D_b(f) &= \min\{c_{b,a} + D_a(f), c_{b,c} + D_c(f), c_{b,e} + D_e(f)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(g) &= \min\{c_{b,a} + D_a(g), c_{b,c} + D_c(g), c_{b,e} + D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty \\
 D_b(h) &= \min\{c_{b,a} + D_a(h), c_{b,c} + D_c(h), c_{b,e} + D_e(h)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(i) &= \min\{c_{b,a} + D_a(i), c_{b,c} + D_c(i), c_{b,e} + D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty
 \end{aligned}$$

# Distance vector example: computation



$t=1$

- e receives DVs from b, d, f, h

DV in d:

$D_c(a) = 1$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = 0$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in h:

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = 0$
$D_c(i) = 1$

DV in

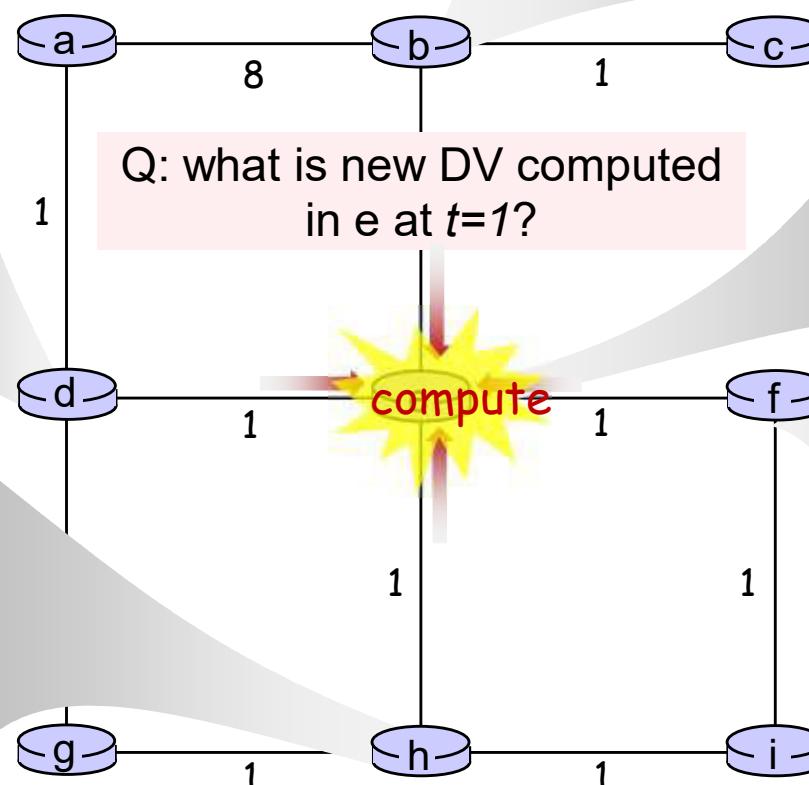
$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in e:

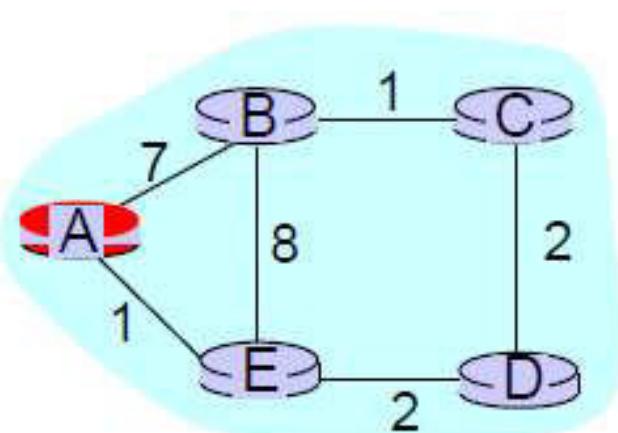
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

DV in f:

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = 0$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = 1$



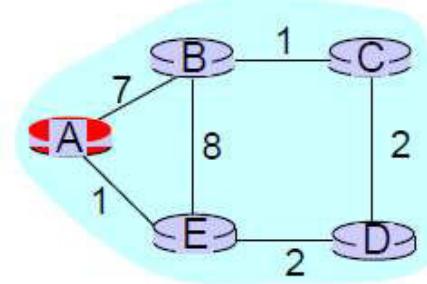
## Distance Table: Example



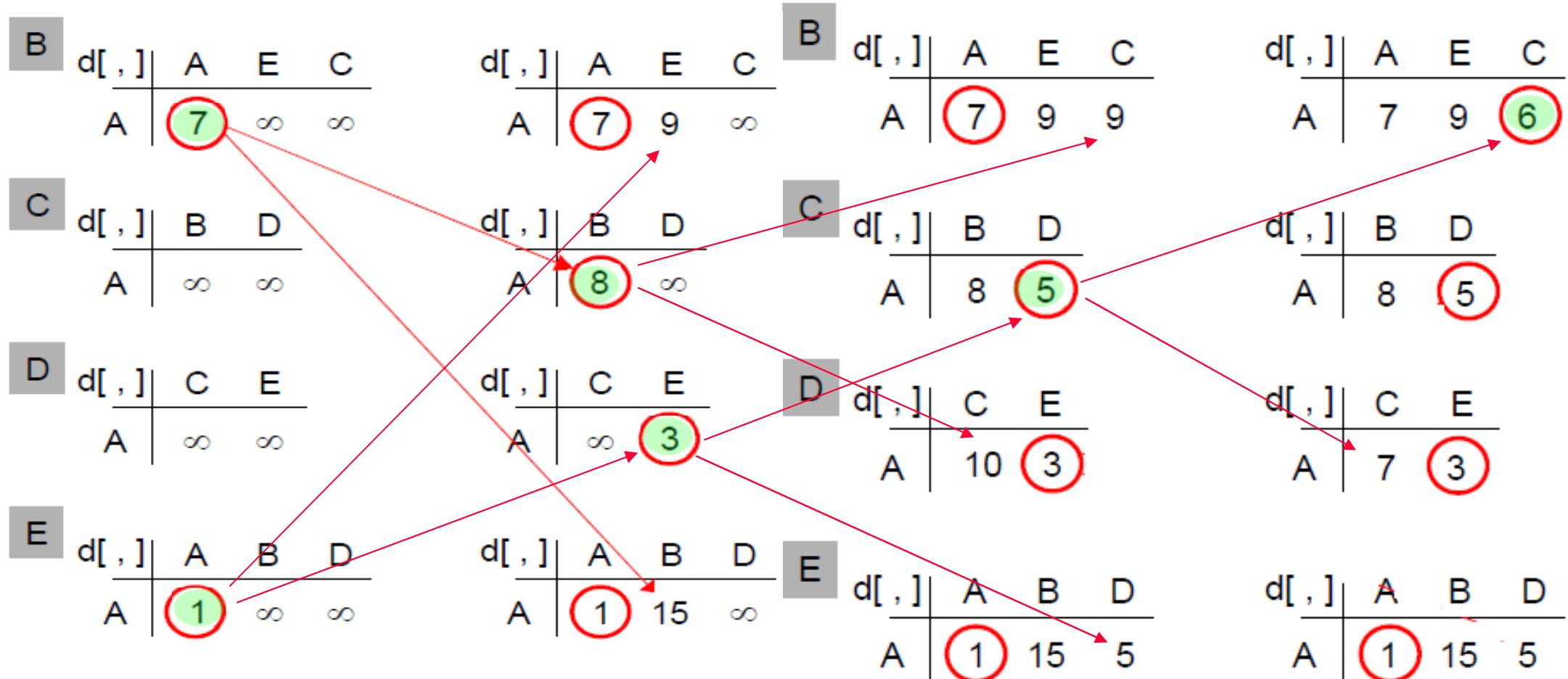
*Distances to Node A*

		neighbor						
		A	E	C	A	E	C	
		d[ , ]						
B	d[ , ]	A	E	C	A	E	C	
		A	7	$\infty$	$\infty$	7	9	$\infty$
C	d[ , ]	B	D		A	B	D	
		A	$\infty$	$\infty$	A	8	$\infty$	
D	d[ , ]	C	E		A	C	E	
		A	$\infty$	$\infty$	A	$\infty$	3	
E	d[ , ]	A	B	D	A	B	D	
		A	1	$\infty$	A	15	$\infty$	

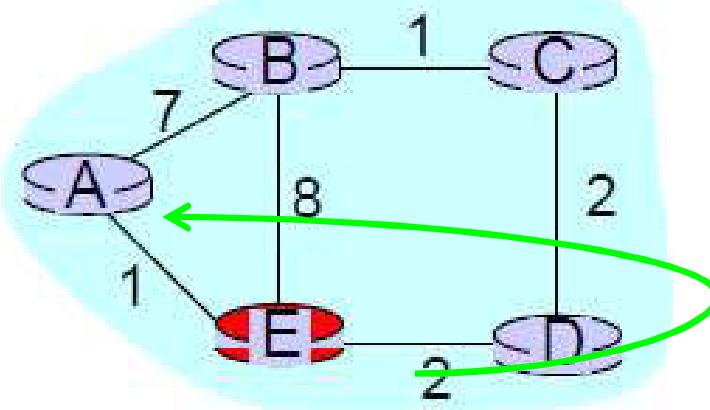
# Distance Table: Example



*Distances to Node A*



## Distance Table: Example



$$dtab_E[D,C] = c[E,D] + 2 = 4$$

$$dtab_E[D,A] = c[E,D] + 3 = 5$$

$$dtab_E[B,A] = c[E,B] + 6 = 14$$

Distance Table for Node E

		neighbor		
		A	B	D
destination	A	1	14	5 Loop
	B	7	8	5
C	6	9	4	
D	4	11	2	

$dtab_E[k,Y]$  - distance from E to Y, going through k

# Distance Vector: Link Cost Changes

## Link cost changes:

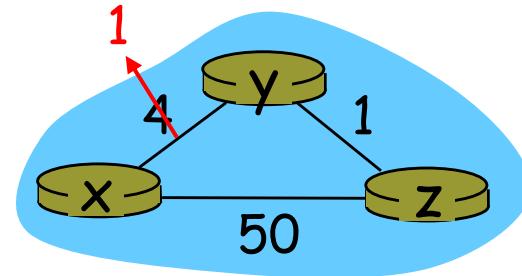
- Node detects local link cost change;
- Updates routing info, recalculates distance vector;
- If DV changes, notify neighbors.

"good  
news  
travel  
fast"

At time  $t_0$ , **y** detects the link-cost change, updates its DV, and informs its neighbors.

At time  $t_1$ , **z** receives the update from **y** and updates its table. It computes a new least cost to **x** and sends its neighbors its DV.

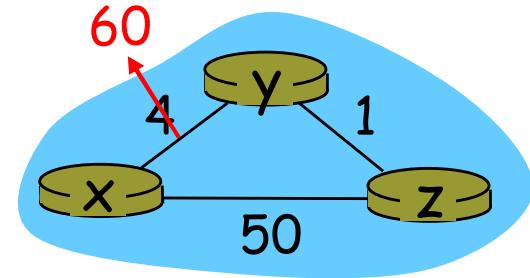
At time  $t_2$ , **y** receives **z**'s update and updates its distance table. **y**'s least costs do not change and hence **y** does not send any message to **z**.



# *Distance Vector: Link Cost Changes*

## Link cost changes:

- Good news travel fast.
- Bad news travel slow – “**count to infinity**” problem!
- 44 iterations before algorithm stabilizes...



## Solution → Poisoned reverse:

- If Z routes through Y to get to X :
  - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z).
- Will this completely solve count to infinity problem?

$y \rightarrow x$	$z \rightarrow x$
4(x)	5(y)
6(z)	5(y)
6(z)	7(y)
8(z)	7(y)
8(z)	9(y)
...	...

# *Comparison of LS and DV Algorithms*

TPC: Prob. 11

## Message complexity:

LS: With  $n$  nodes,  $E$  links,  $O(nE)$  messages sent.

DV: Exchange between neighbors only – Convergence time varies.

## Speed of Convergence:

LS: algorithm requires  $O(nE)$  messages.

- May have oscillations.

DV: Convergence time varies:

- May temporarily have routing loops;
- Count-to-infinity problem.

## Robustness: what happens if router malfunctions?

LS:

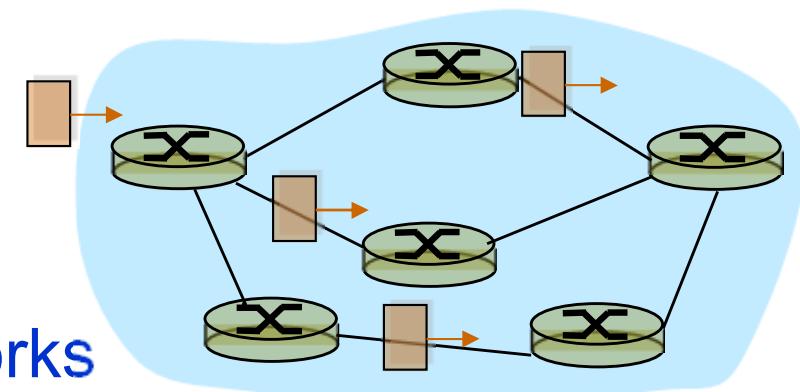
- Node can advertise incorrect *link* cost.
- Each node computes only its *own* table.

DV:

- DV node can advertise incorrect *path* cost.
- Each node's table used by others – errors propagate through network.

# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



## Hierarchical Routing

The routing study so far was based on an idealization:

- All routers assumed identical;
- “Flat” network;  
... *not* true in practice

**Scale – with >18 000 million destinations:**

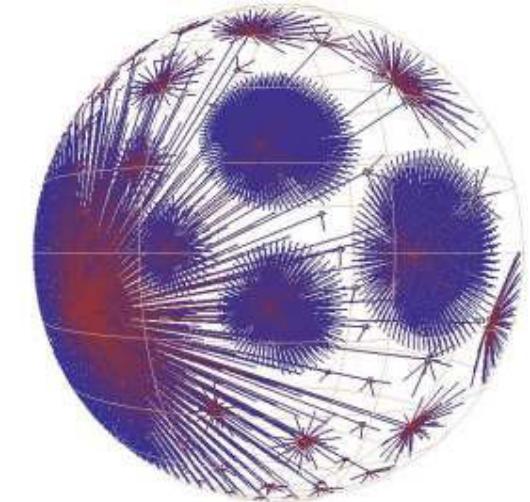
[<http://blogs.cisco.com/news/cisco-connections-counter>]

- **Can't store all destinations in routing tables!**
- **Routing table exchange would swamp links!**

**Solution: Administrative autonomy**

- *Internet = network of networks;*
- **Each network administration may want to control routing in its own network.**

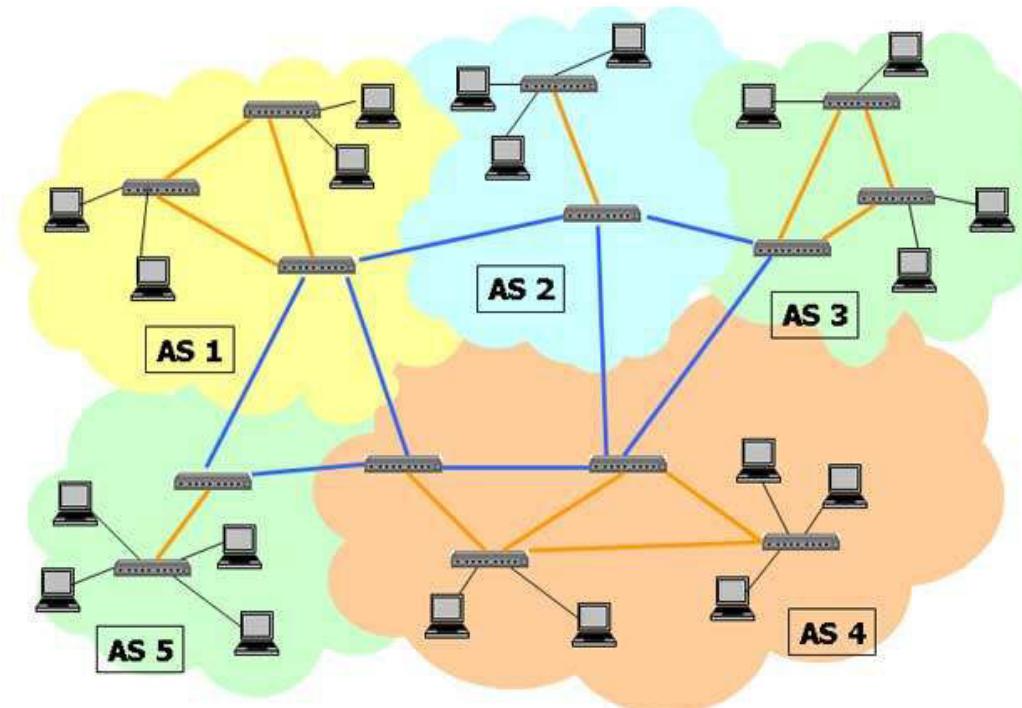
# Hierarchical Routing

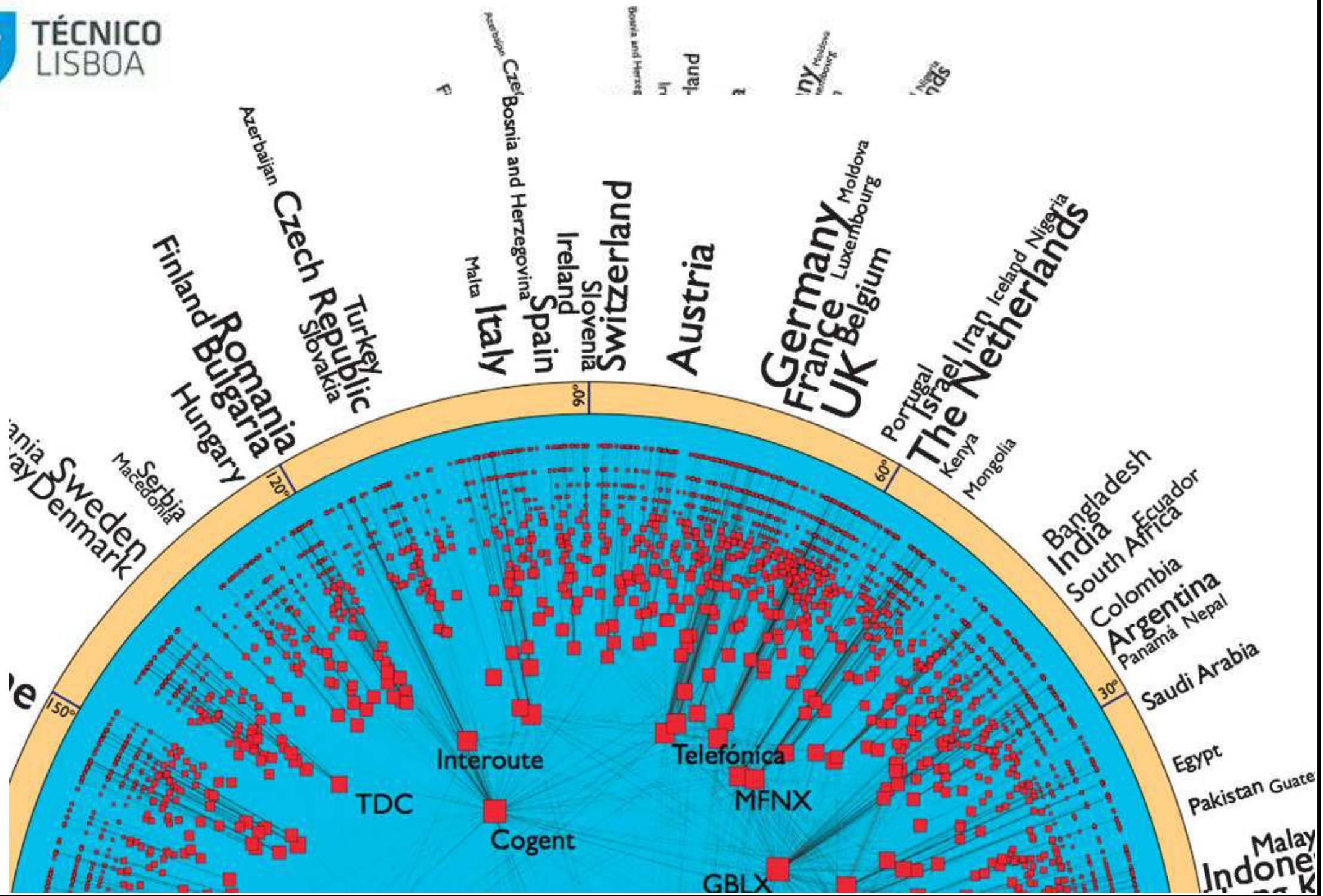


- Aggregate routers into regions:  
**“Autonomous systems” (AS)**
- Routers in same AS run the same routing protocol:
  - **“Intra-AS” routing** protocol;
  - Routers in different AS can run different intra-AS routing protocols.

## Gateway router:

- Direct links to routers in other ASs.



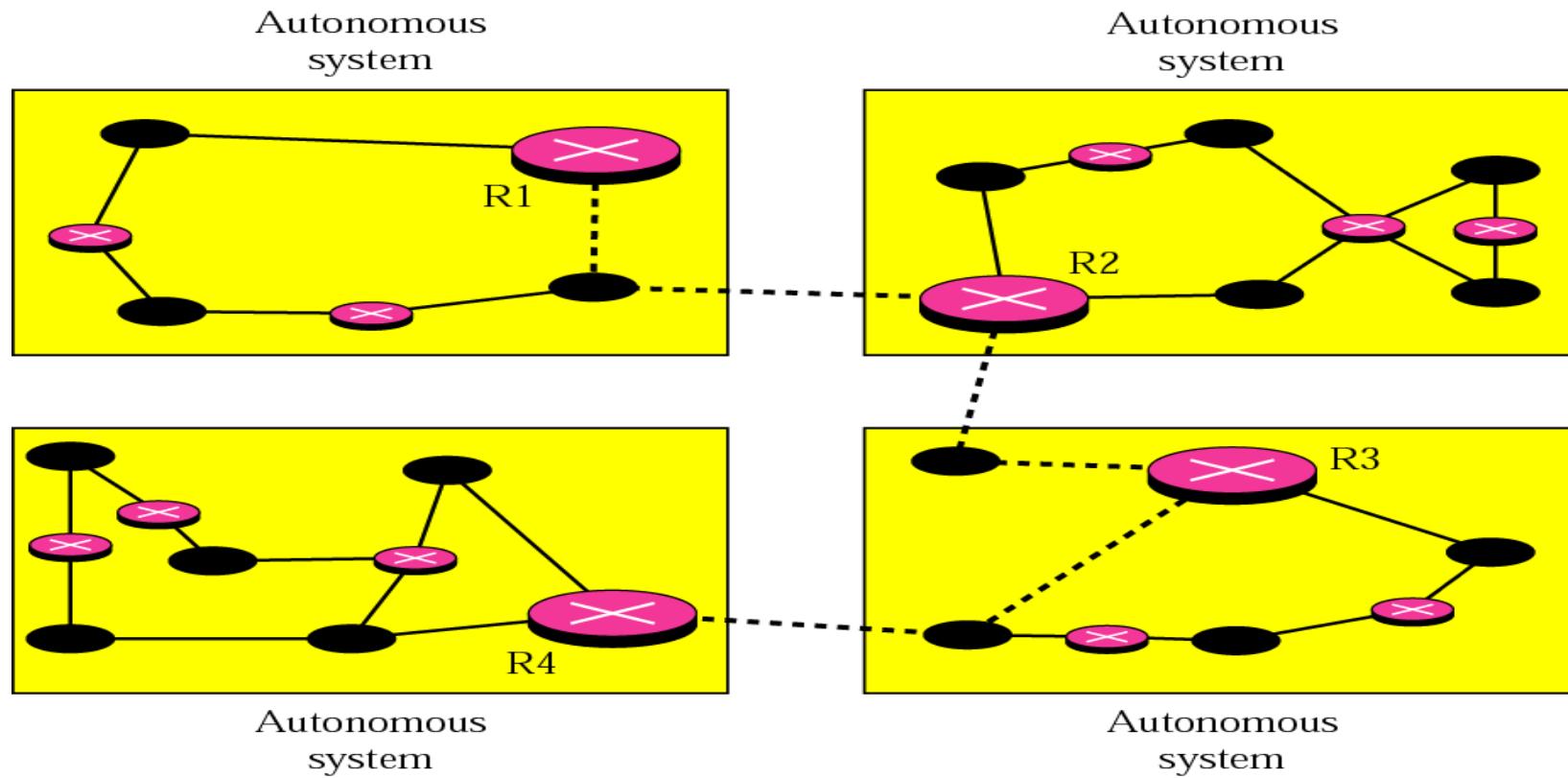


# Hierarchical Routing

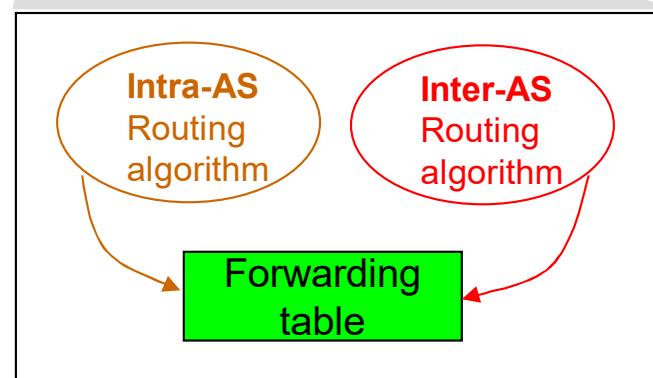
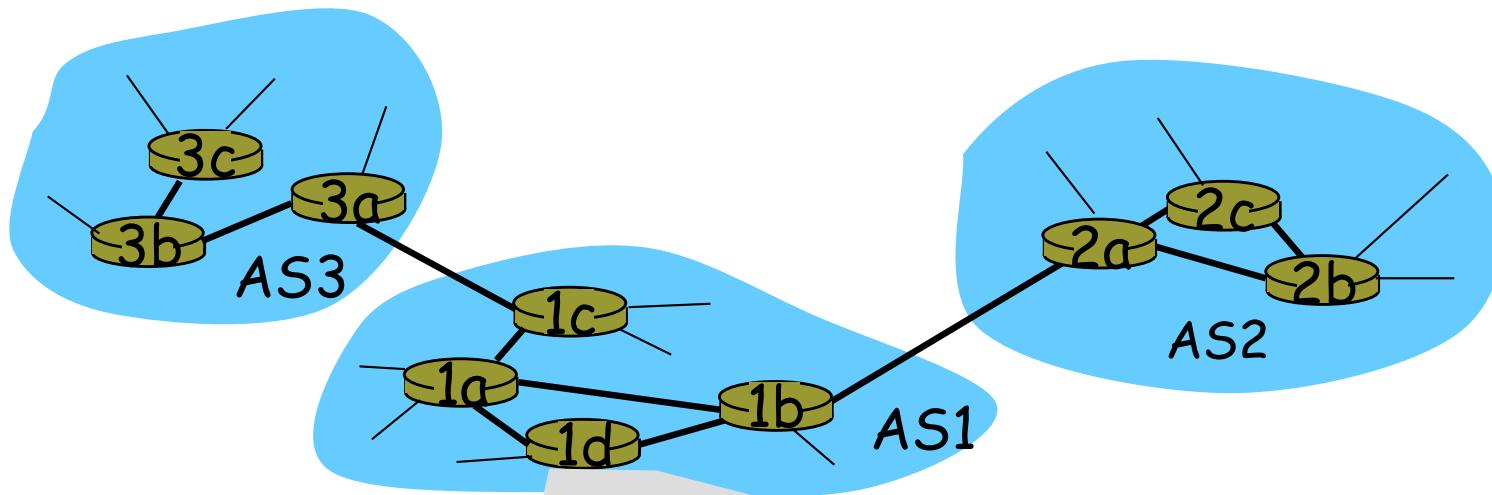
Hierarchical routing:

- Delivery is first done to the **AS**;
- Then, to the **destination network**;
- Finally packets are delivered to the **destination host** (*using data link layer*).

Size of routing tables can be considerably reduced!



# Interconnected ASes



- Forwarding table configured by both intra- and inter-AS routing algorithms:
  - **Intra-AS** sets entries for internal destinations.
  - **Inter-AS & intra-AS** set entries for external destinations.

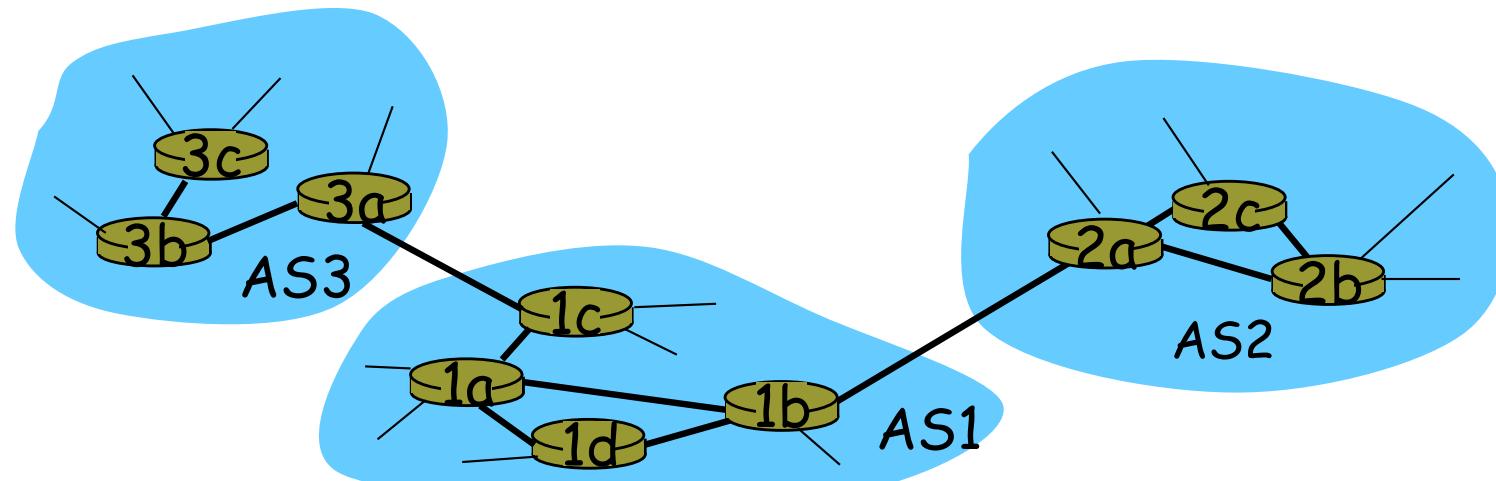
# Inter-AS Tasks

- AS1 router receives datagram with destination outside of AS1:
  - **Forward packet to which gateway router?**

AS1 must:

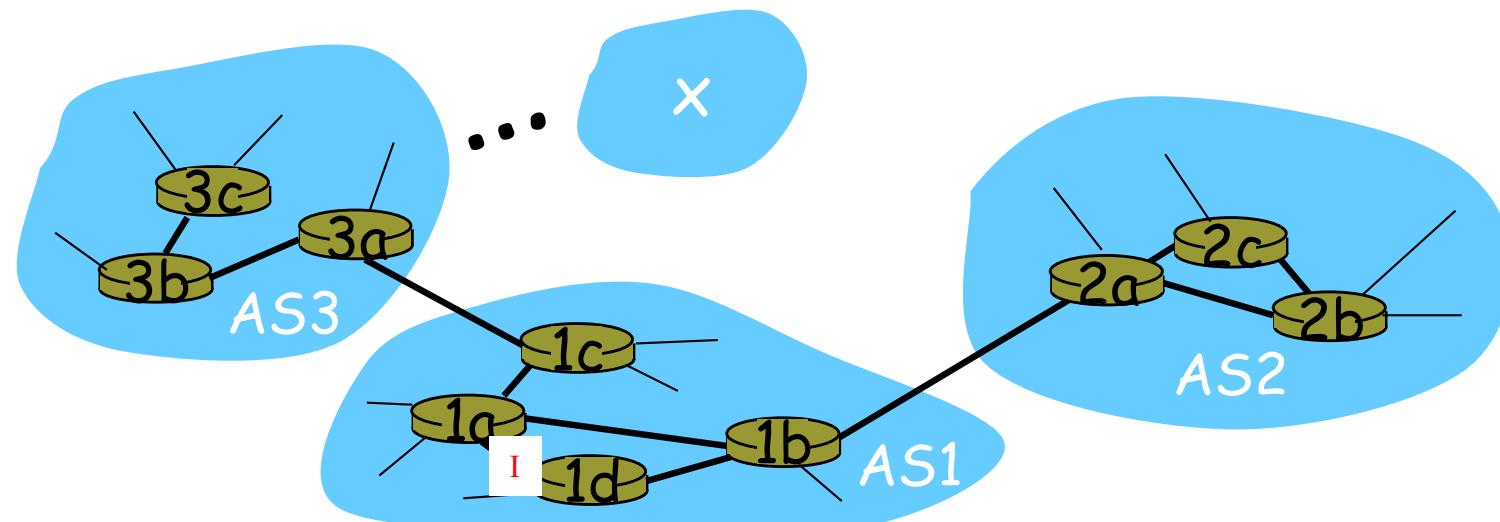
1. Learn which destinations are reachable through AS2 and which through AS3;
2. **Propagate this reachability information to all AS1 routers.**

Job of **inter-AS routing!**



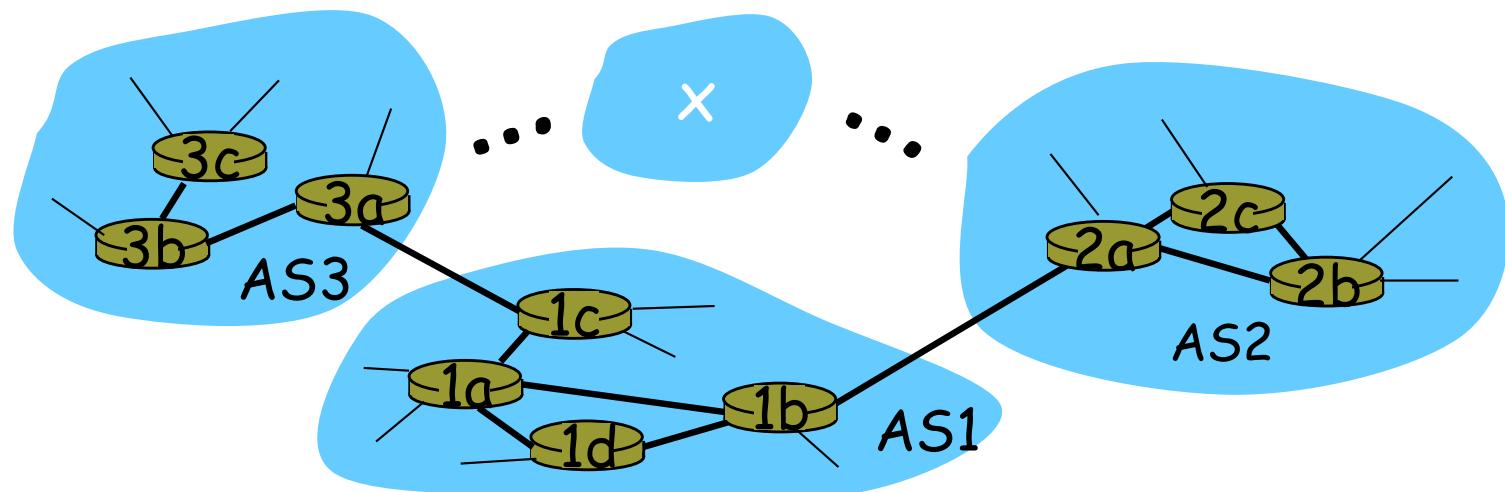
# Setting Forwarding Table in Router 1d

- AS1 learns (via inter-AS protocol) that **subnet  $x$  is reachable via AS3 (gateway 1c)** but not via AS2.
- **Inter-AS protocol propagates reachability information to all internal routers.**
- Router 1d uses intra-AS routing information to find the least cost path to 1c, via its interface  $I$  ;
  - Router 1d installs in its forwarding table the entry:  $(x, I)$ .



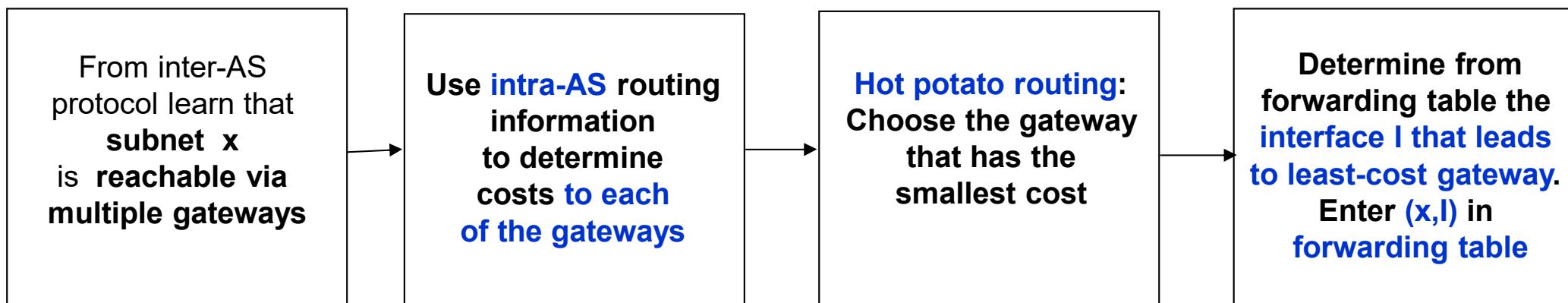
# *Choosing among Multiple ASes*

- If AS1 learns (inter-AS protocol) that subnet **x** is reachable from AS3 *and* from AS2.
- To configure forwarding table, router 1d must determine towards which gateway it should forward packets for destination **x**.



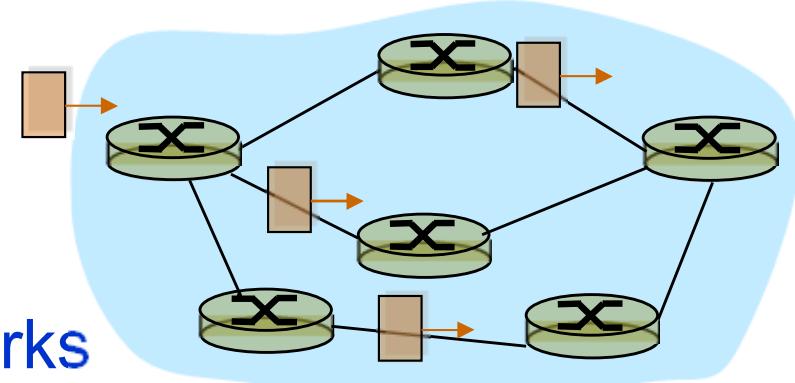
# Choosing among Multiple ASes

- If AS1 learns (inter-AS protocol) that subnet **x** is reachable from AS3 *and* from AS2.
- To configure forwarding table, router 1d must determine towards which gateway it should forward packets for destination **x**.
- **Hot potato routing:** send packet towards closest (lowest cost) of two gateway routers.



# Outline

- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing

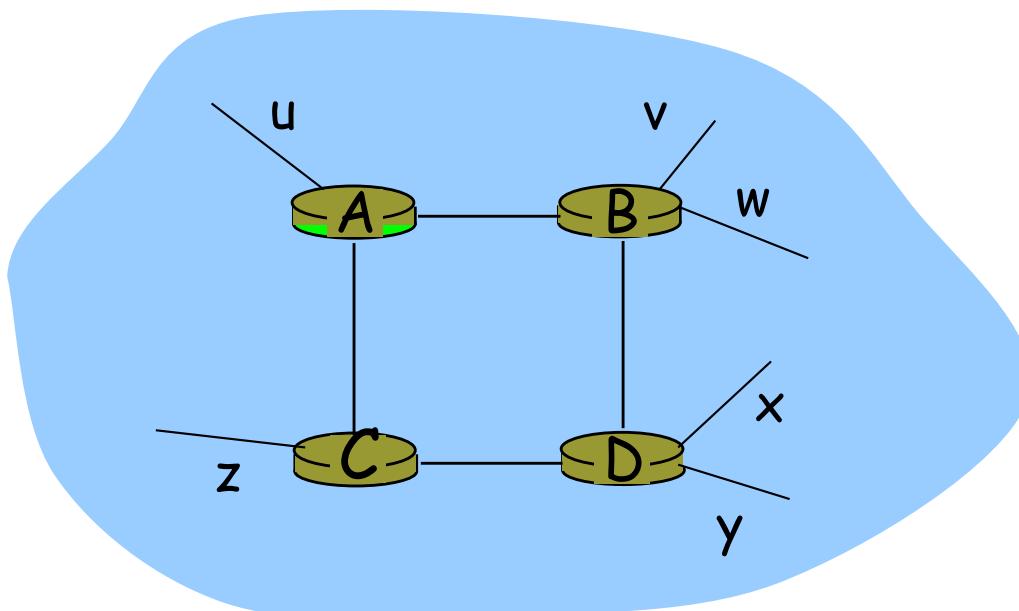


## *Intra-AS Routing*

- Also known as **Interior Gateway Protocols (IGP)**.
- Most common Intra-AS routing protocols:
  - **RIP**: Routing Information Protocol;
  - **OSPF**: Open Shortest Path First;
  - **IGRP**: Interior Gateway Routing Protocol  
(Cisco proprietary).

# RIP (Routing Information Protocol)

- Uses a **distance vector algorithm**;
- Distance metric: number of hops  
**(max distance = 15 hops, 16=∞);**



From router A to subnets:

destination	hops
u	1
v	2
w	2
x	3
y	3
z	2

## *RIP Advertisements*

- **Distance vectors:**  
exchanged among neighbors (**every 30 sec**,  
via ‘Response Message’, also called **advertisement**);
  
- **Each advertisement:**  
list of up to 25 destination subnets within the AS.

## RIP: Link Failure and Recovery

If no advertisement heard after 180 sec → neighbor/link declared dead:

- Routes via neighbor invalidated;
- New advertisements sent to remaining neighbors;
- Neighbors in turn send out new advertisements (if tables changed);
- Link failure info quickly propagates to entire net;

**Poisoned reverse** used to prevent *ping-pong* loops.

In RIP: infinite distance = 16 hops.

# *OSPF (Open Shortest Path First)*

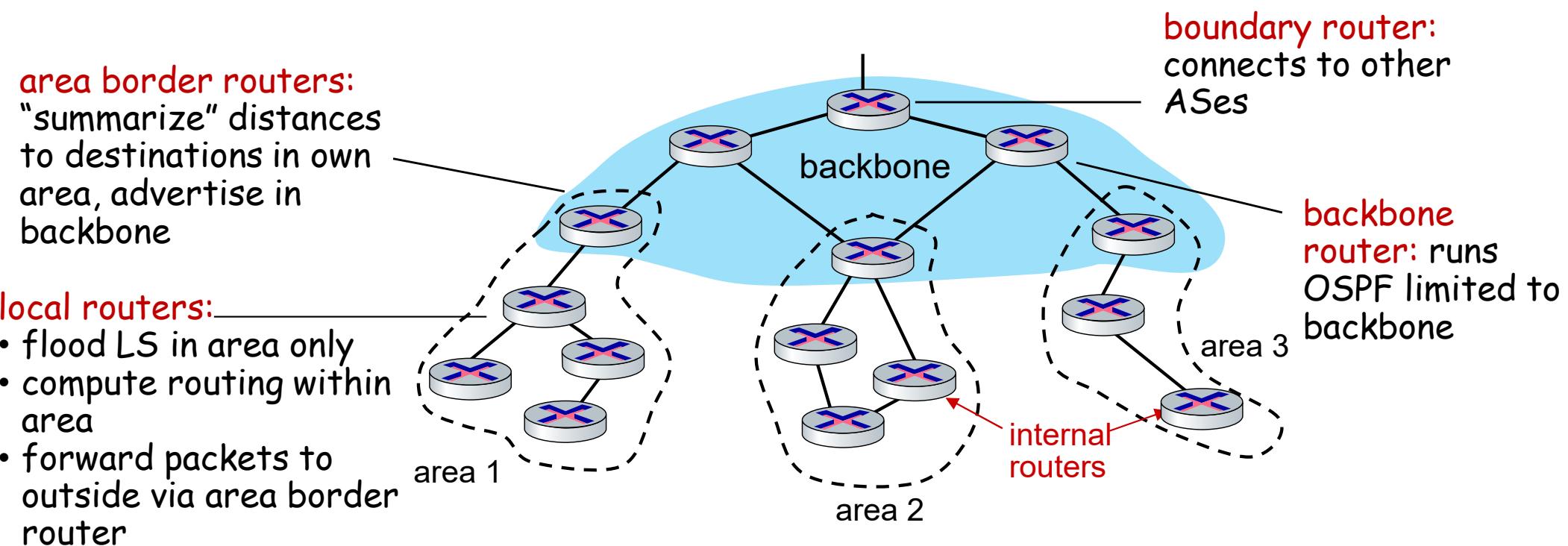
- “Open”: publicly available;
- Uses **Link State** algorithm:
  - Link state packet (LSP) dissemination;
  - Topology map at each node;
  - Route computation using Dijkstra’s algorithm.
- OSPF advertisement carries one entry per neighbor router.
- Advertisements disseminated to **entire AS** (via flooding):
  - Carried in **OSPF messages directly over IP** (rather than TCP or UDP). Process executing OSPF: **gated**.

## *OSPF “Advanced” Features (not available in RIP)*

- Security: all OSPF messages authenticated (to prevent malicious intrusion);
- Multiple same-cost paths allowed (only one path in RIP);
- For each link, multiple cost metrics for different TOS (e.g., satellite link cost set “low” for best effort; “high” for real time);
- Integrated uni- and multicast support:
  - Multicast OSPF (MOSPF) uses same topology database as OSPF;
- Hierarchical OSPF in large domains.

# Hierarchical OSPF

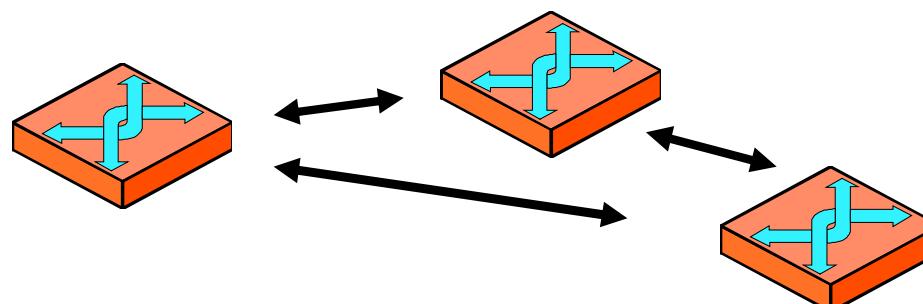
- Two-level hierarchy: local area, backbone.
  - link-state advertisements flooded only in area, or backbone
  - each node has detailed area topology; only knows direction to reach other destinations



# RIP vs OSPF

The choice is done by the AS manager;

- RIP is appropriate for small networks:
  - Easy to implement;
  - 15 hops is not a problem;
  - Table diffusion (*even to hosts, interrupting them*) is not a big problem;
  - Distance vector.
- OSPF is scalable:
  - Works with networks of any dimension;
  - Link-state;
  - Management complexity is compensated by the better efficiency in larger networks.



## *Internet Inter-AS Routing: BGP*

- BGP (Border Gateway Protocol):  
*the de facto inter-domain routing protocol*
  - “glue that holds the Internet together”
- Allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet:  
*“I am here, here is who I can reach, and how”*
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from **neighboring ASes**
  - iBGP: propagate reachability information to all **AS-internal routers**.
  - determine “good” routes to other networks based on reachability information and *policy*

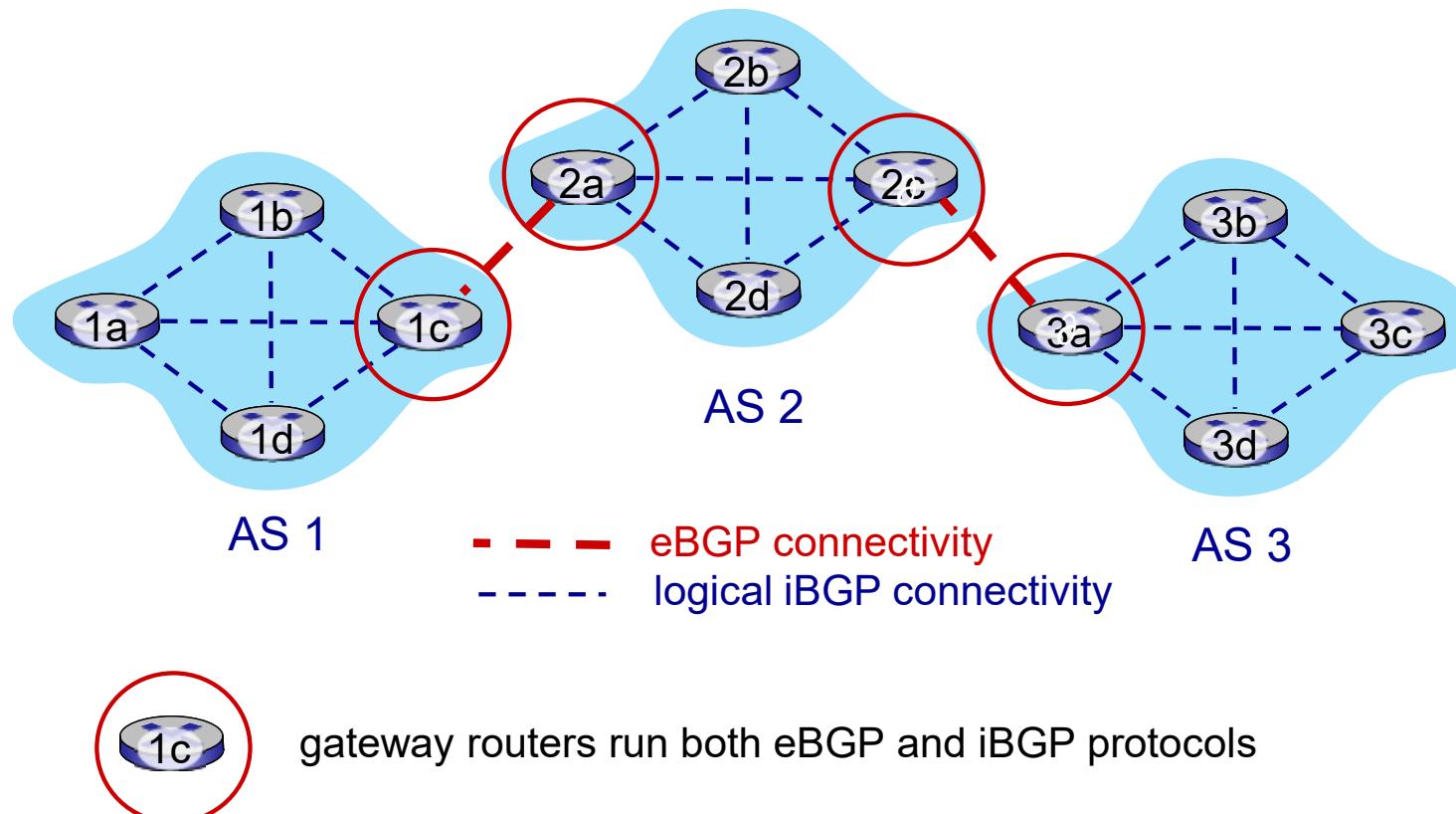
# *Path Attributes and BGP Routes*

- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of ASes through which prefix advertisement has passed
    - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- Policy-based routing:
  - gateway receiving route advertisement uses ***import policy to accept/decline path*** (e.g., never route through AS Y).
  - AS policy also determines whether to ***advertise*** path to other other neighboring ASes

## BGP Messages

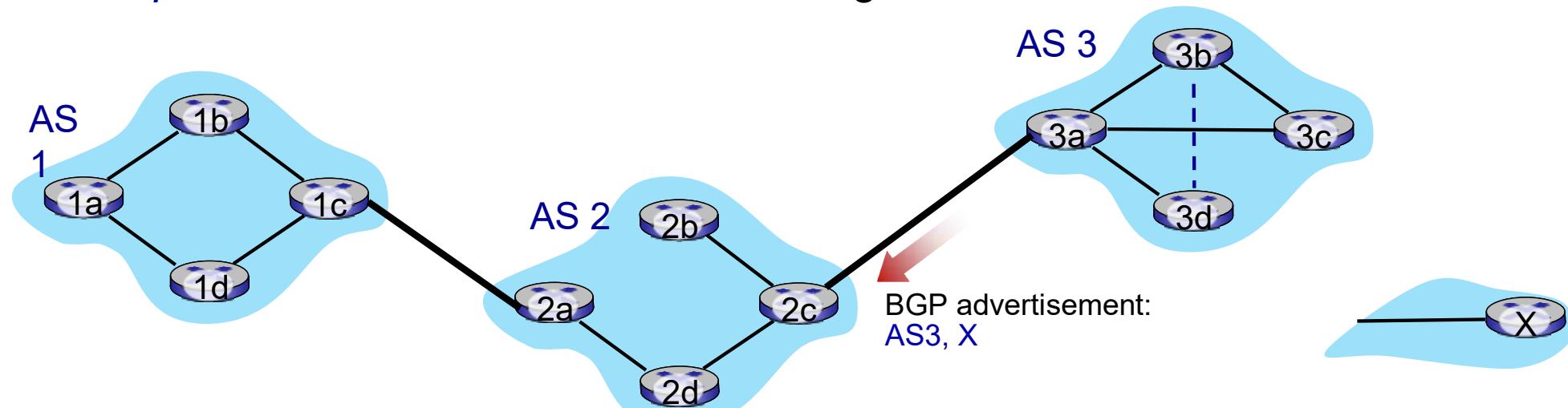
- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - **UPDATE**: advertises new path (or withdraws old)
  - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - **NOTIFICATION**: reports errors in previous msg; also used to close connection

## *eBGP, iBGP Connections*

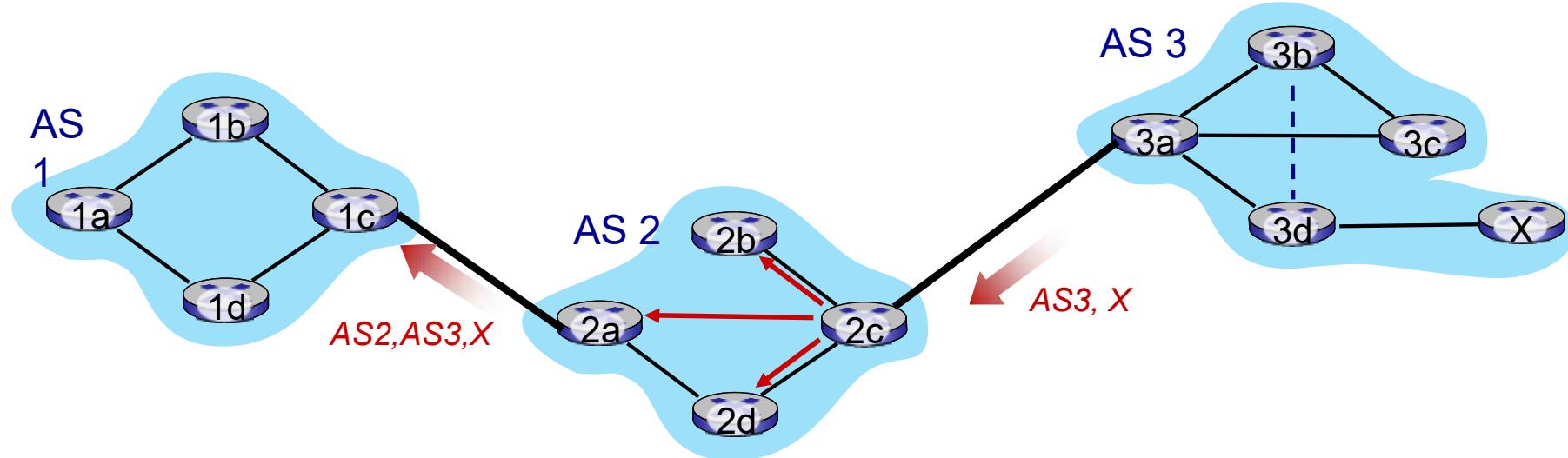


## BGP Basics

- **BGP session:** two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)
- when AS3 gateway 3a advertises path AS3,X to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X

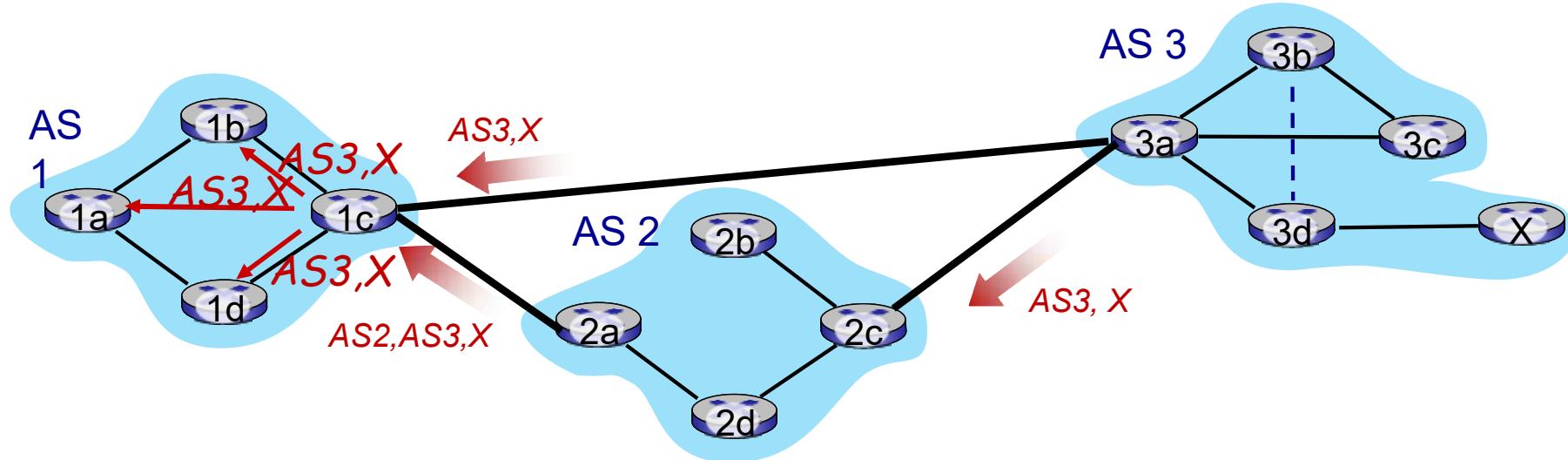


# BGP Path Advertisement



- AS2 router 2c receives path advertisement **AS3,X** (via eBGP) from AS3 router 3a
- based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- based on AS2 policy, AS2 router 2a advertises (via eBGP) path **AS2, AS3, X** to AS1 router 1c

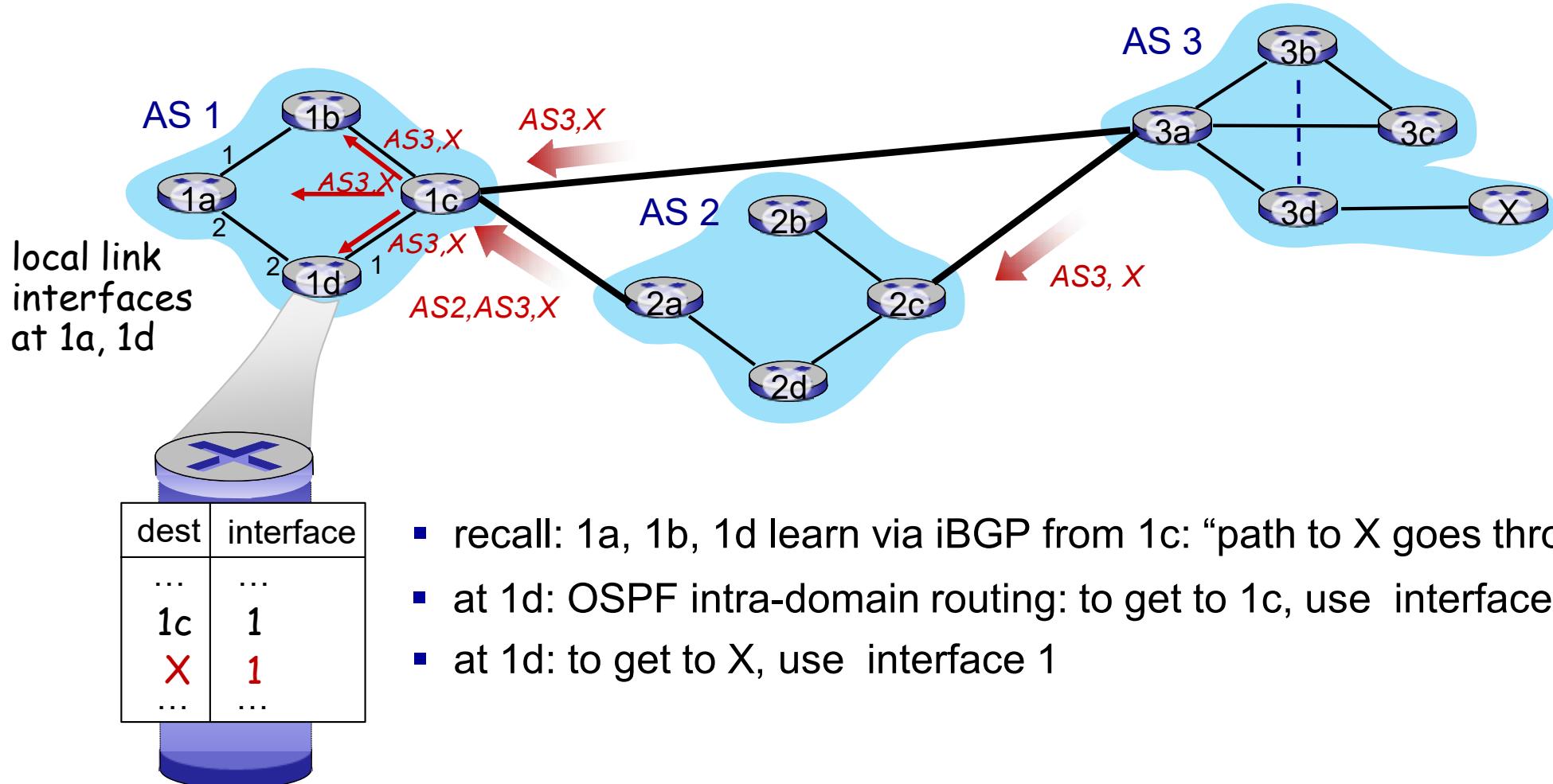
## BGP Path Advertisement (more)



Gateway router may learn about **multiple** paths to destination:

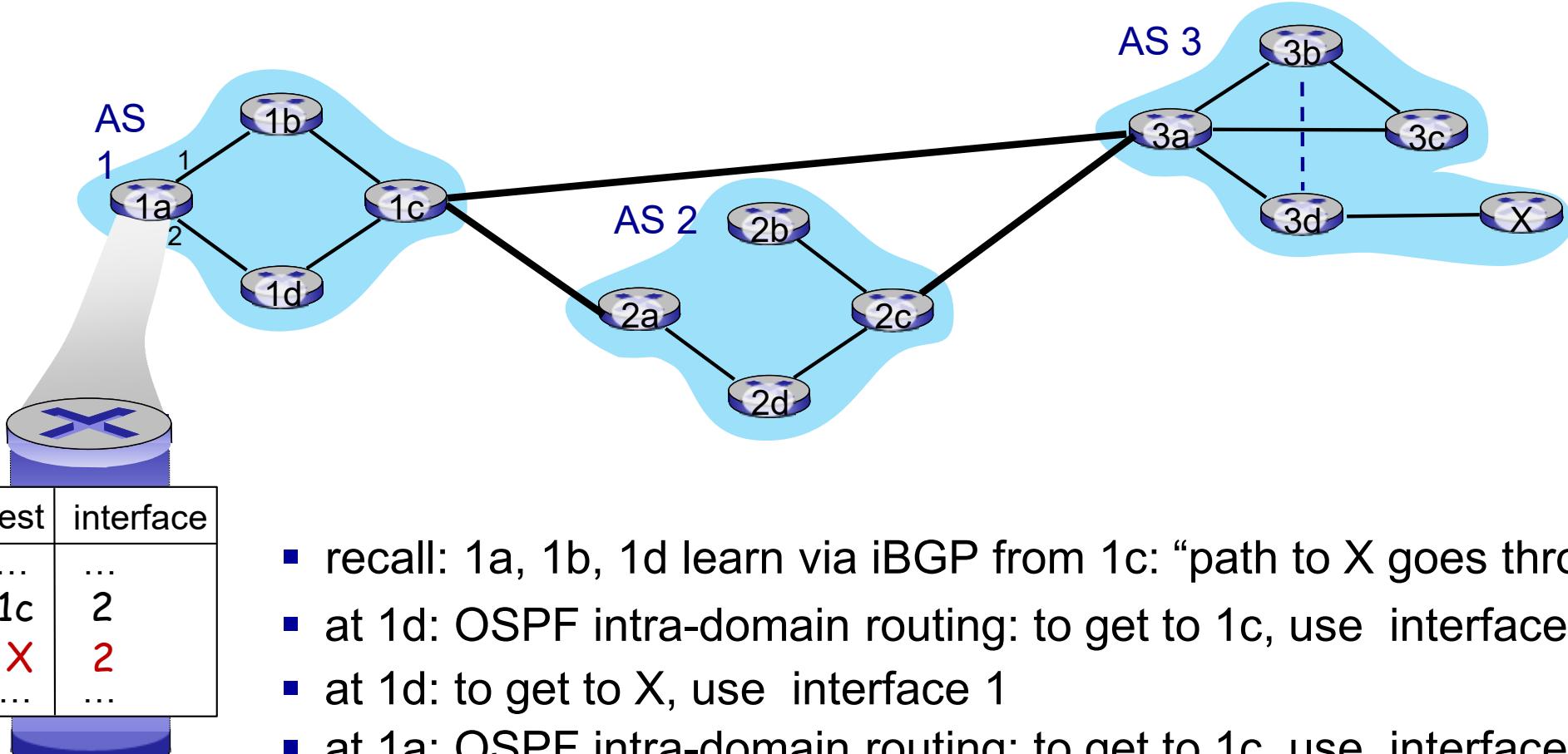
- AS1 gateway router 1c learns path **AS2,AS3,X** from 2a
- AS1 gateway router 1c learns path **AS3,X** from 3a
- based on *policy*, AS1 gateway router 1c chooses path **AS3,X** and advertises path within AS1 via iBGP

# BGP Path Advertisement



- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1

# BGP Path Advertisement



- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1
- at 1a: OSPF intra-domain routing: to get to 1c, use interface 2
- at 1a: to get to X, use interface 2

## *Why Different Intra-, Inter-AS Routing ?*

### Policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its network
- intra-AS: single admin, so policy less of an issue

### Scale:

- hierarchical routing saves table size, reduced update traffic

### Performance:

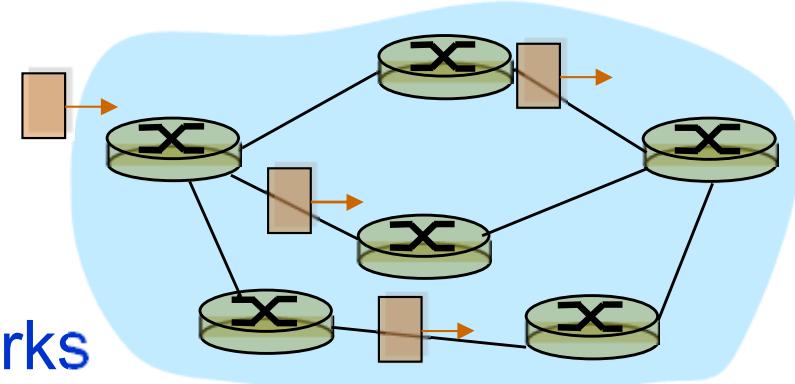
- intra-AS: can focus on performance
- inter-AS: policy dominates over performance

## *BGP Route Selection*

- Router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

# Outline

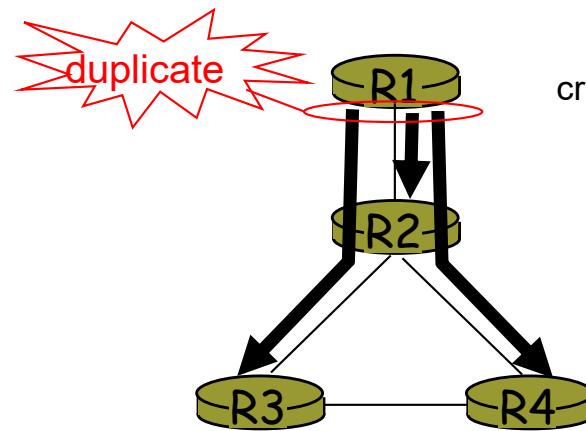
- Introduction
- Virtual circuit and datagram networks
- IPv4 addressing and forwarding tables
- Internet Protocol (IP) - Datagram format, Fragmentation
- ICMP, DHCP
- NAT, IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Hierarchical routing, RIP, OSPF, BGP
- Broadcast and multicast routing



# Broadcast Routing

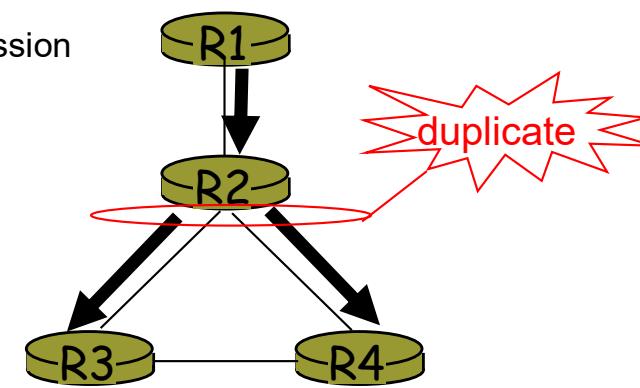
- Deliver packets from source to all other nodes.

Source duplication is inefficient:



source  
duplication

duplicate  
creation/transmission



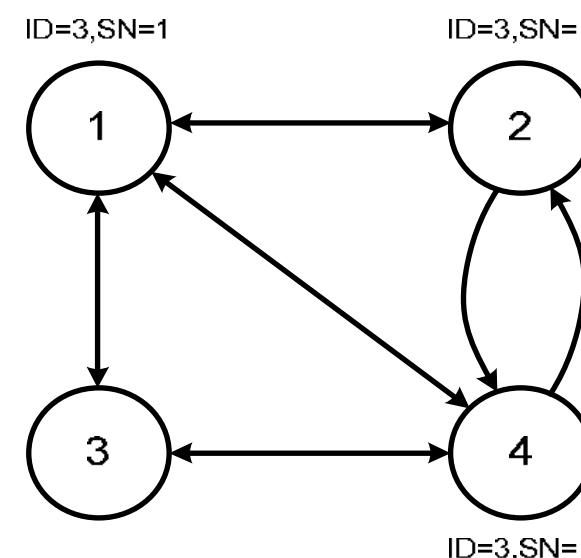
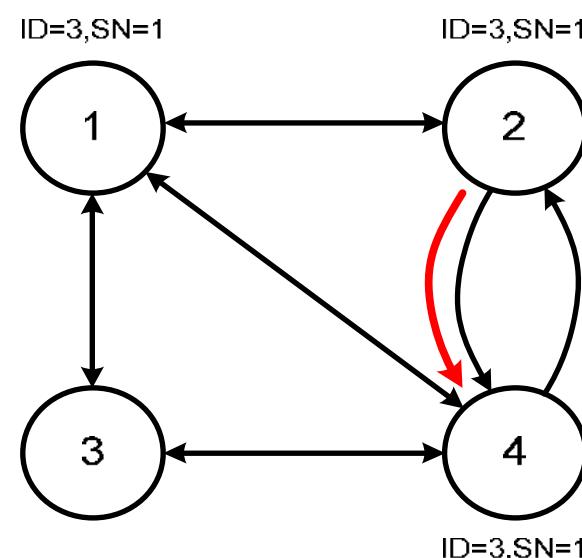
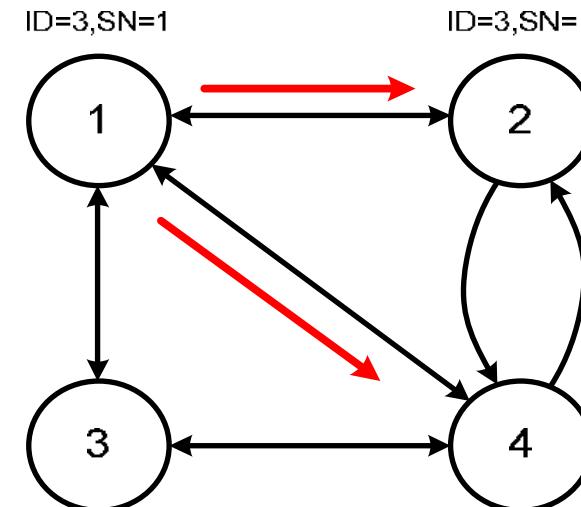
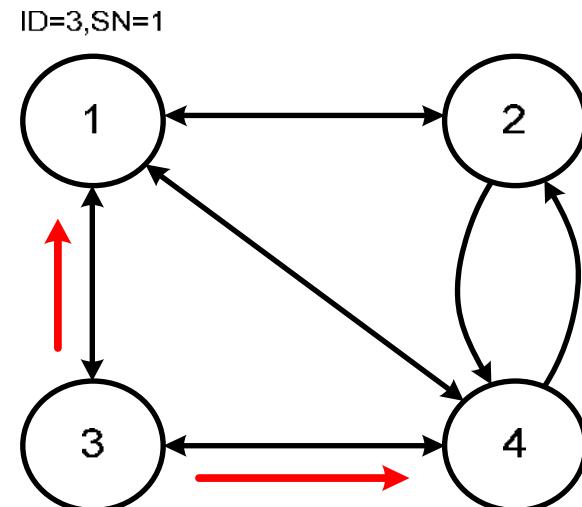
in-network  
duplication

Source duplication:  
How does source determine recipient addresses?

## *In-Network Duplication*

- Flooding - when node receives broadcast packet, sends copy to all neighbors:
  - Problems: cycles & broadcast storm.
- Controlled flooding - node only broadcasts packet if it hasn't broadcast the same packet before:
  - Node keeps track of **packet IDs** already broadcasted;
  - Or, **reverse path forwarding** (RPF): only forward packet if it arrived on the shortest path between node and source.
- Spanning tree:
  - No redundant packets are received by any node.

# Broadcast Routing



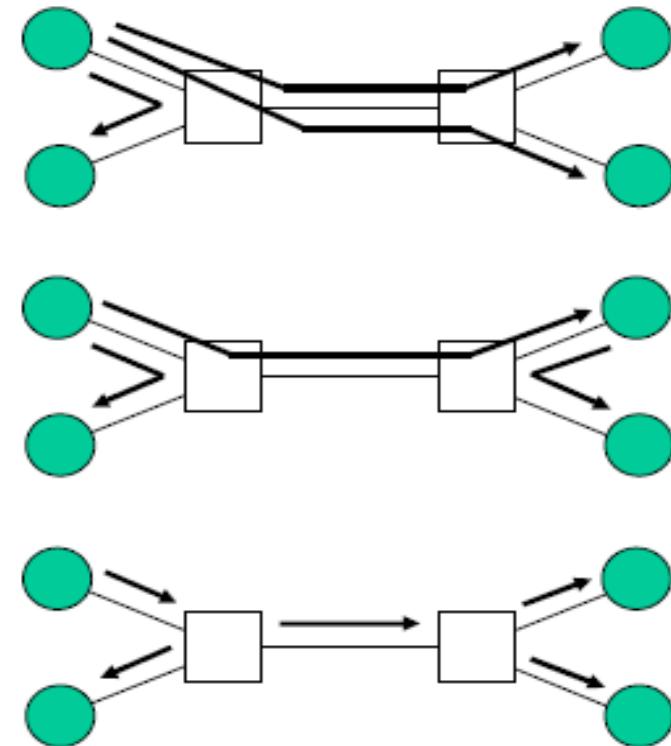
controlled flooding, broadcast of first message.

# Multicast

TPC: Prob. 12

When a source application generates a message, a copy of that message should be delivered to each one of the destinations belonging to the *multicast* group.

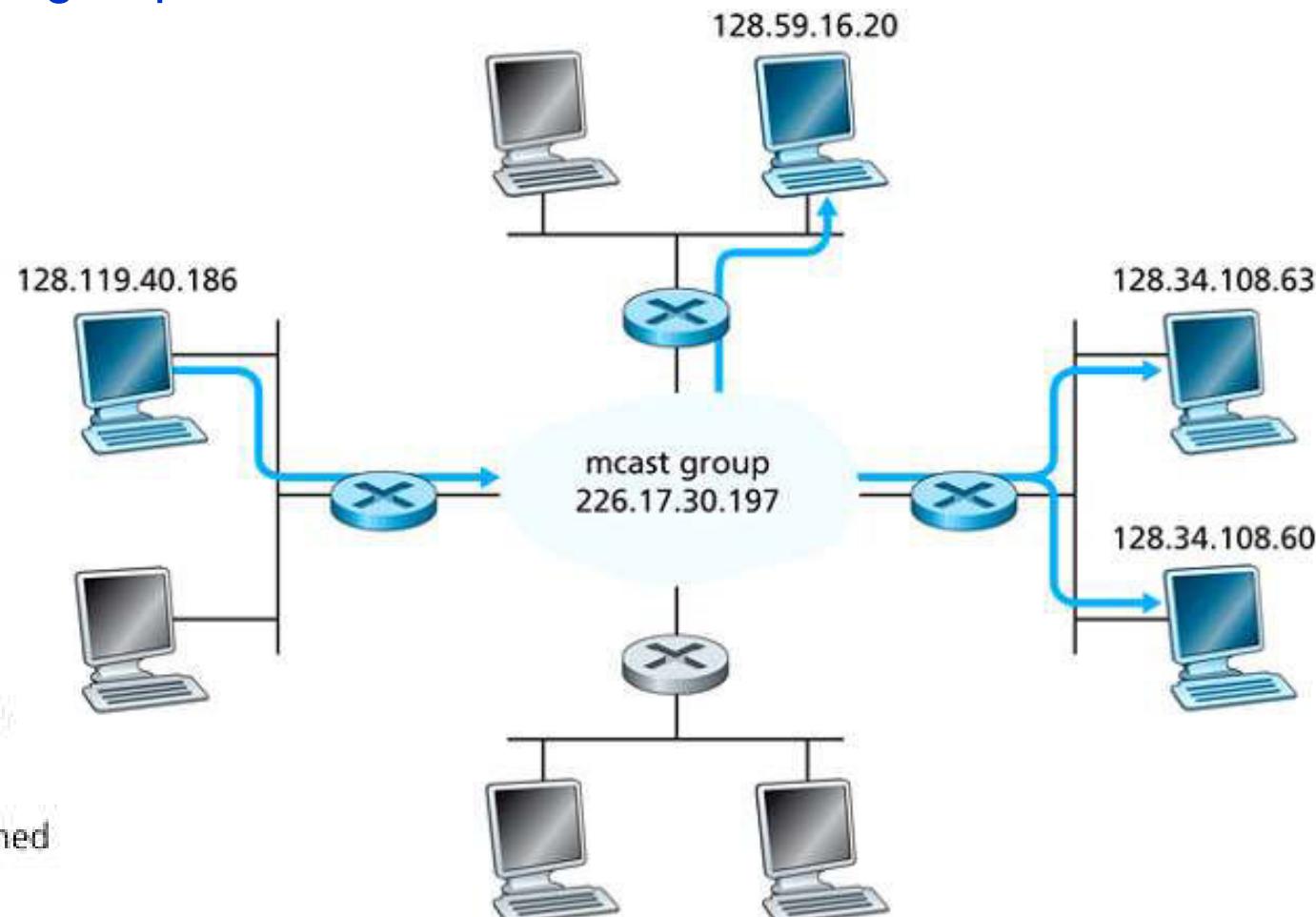
- *Multicast emulation:*
  - Source establishes *unicast* sessions with each one of the destinations.
- *Multicast in the application layer:*
  - Hosts (end stations) build a *multicast* logical tree on top of the network.
- *Multicast in the network layer:*
  - Routers build *multicast* tree.



# Multicast

## Multicast group:

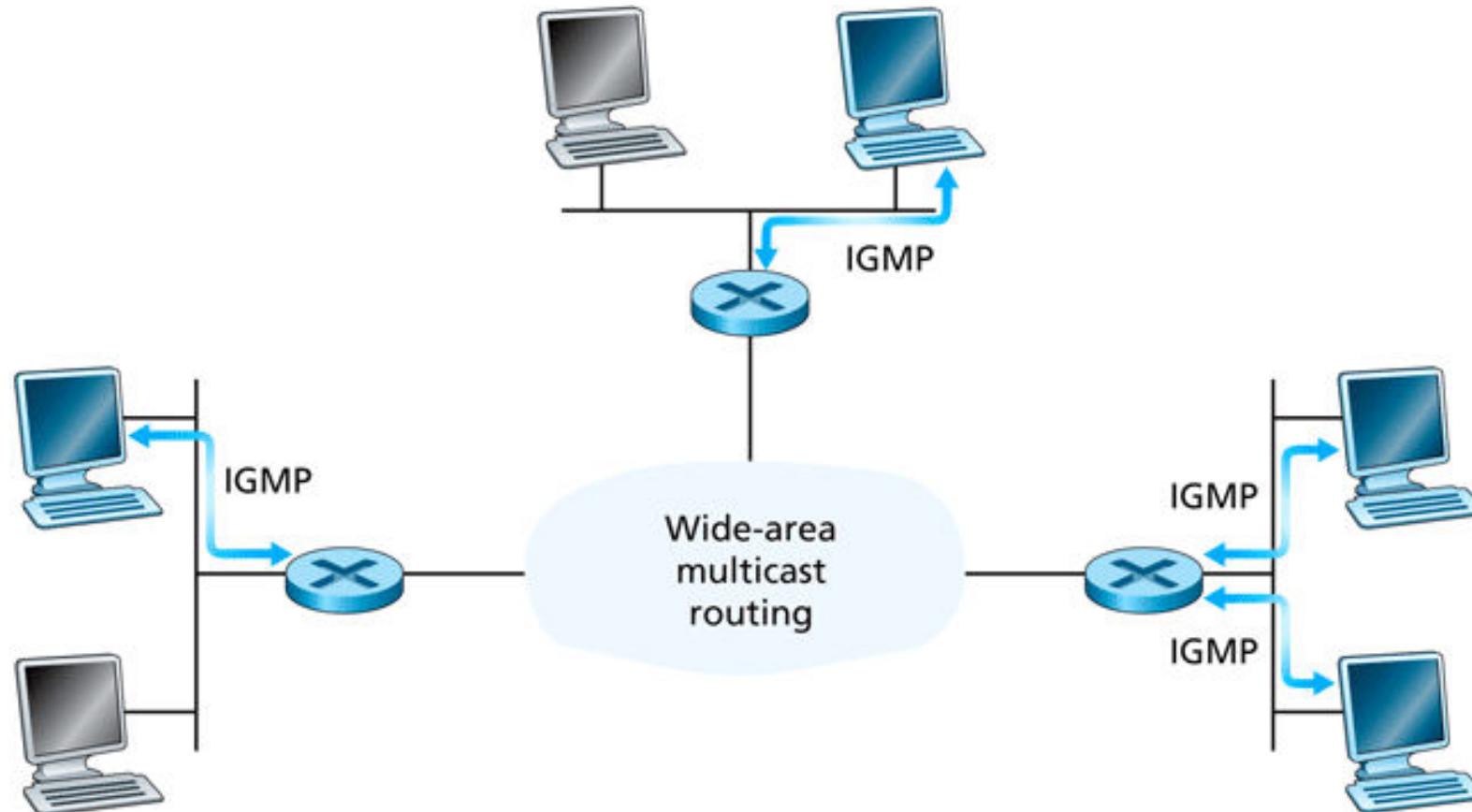
- A datagram addressed to the group is delivered to all members of the *multicast* group.



# Multicast

The two components of network-layer multicast in the Internet:

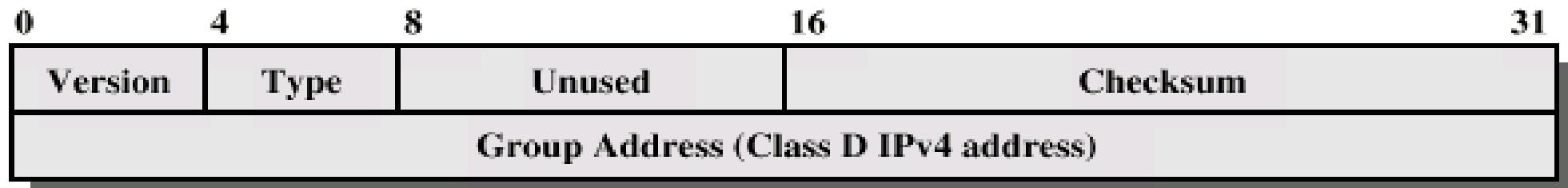
- **Multicast Group Membership Discovery** protocol
  - Internet Group Management Protocol (**IGMP**) - IPv4;
  - Multicast Listener Discovery (**MLD**) - IPv6;
- *Multicast routing protocols.*



# Internet Group Management Protocol (IGMP)

All destinations in a *multicast* group share the same IP address (class D).

- *Internet Group Management Protocol* (RFC 1112):
  - Operates between one host and the router to which it is directly connected;
  - Router wants to know, for each interface, which *multicast* groups have members connected to that interface;
  - Router invites hosts to indicate to which *multicast* groups they want to belong.



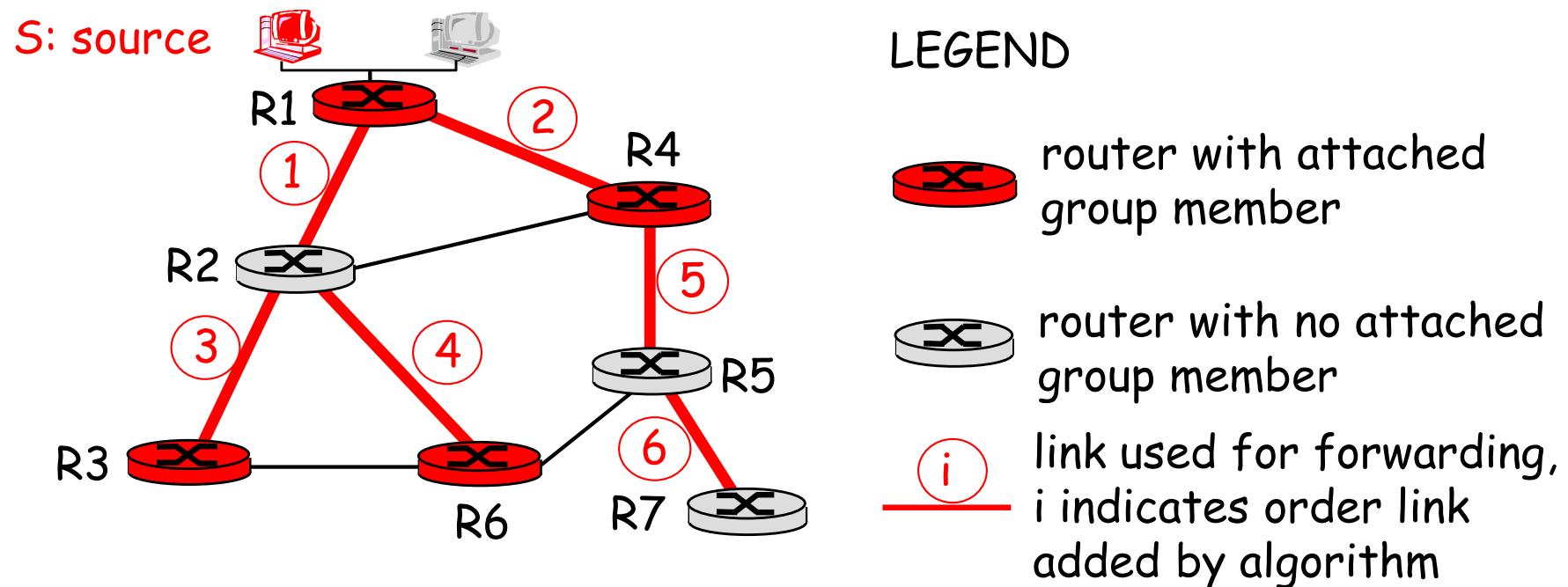
# *Approaches for Building Mcast Trees*

## Approaches:

- **Source-based tree** - one tree per source:
  - Shortest path trees (e.g., using Dijkstra);
  - Reverse path forwarding.
- **Group-shared tree** - group uses one tree:
  - Minimal spanning (Steiner) tree;
  - Center-based trees.

# Shortest Path Tree

- Mcast forwarding tree - tree of shortest path routes from source to all receivers:
  - Dijkstra's algorithm.



# Reverse Path Forwarding

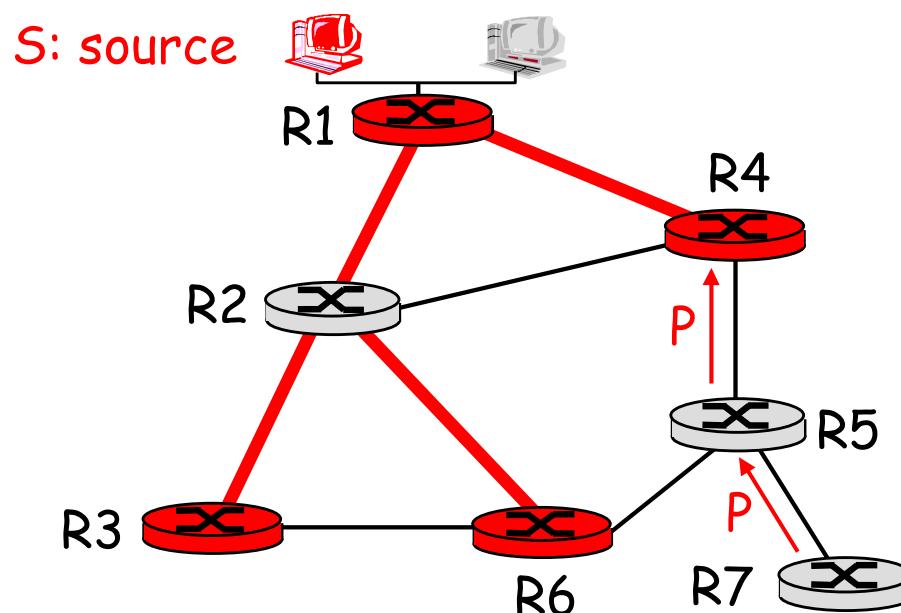
TPC: Prob. 13

- Rely on router's knowledge of unicast shortest path from it to sender;
- Each router has simple forwarding behavior:

***if*** (Mcast datagram received on incoming link on the shortest path back to center):  
***then***  
    flood datagram onto all outgoing links;  
***else***  
    ignore datagram.

# Reverse Path Forwarding: Pruning

- Forwarding tree contains subtrees with no Mcast group members:
  - No need to forward datagrams down those subtrees;
  - “Prune” messages sent upstream by router with no downstream group members.



## LEGEND

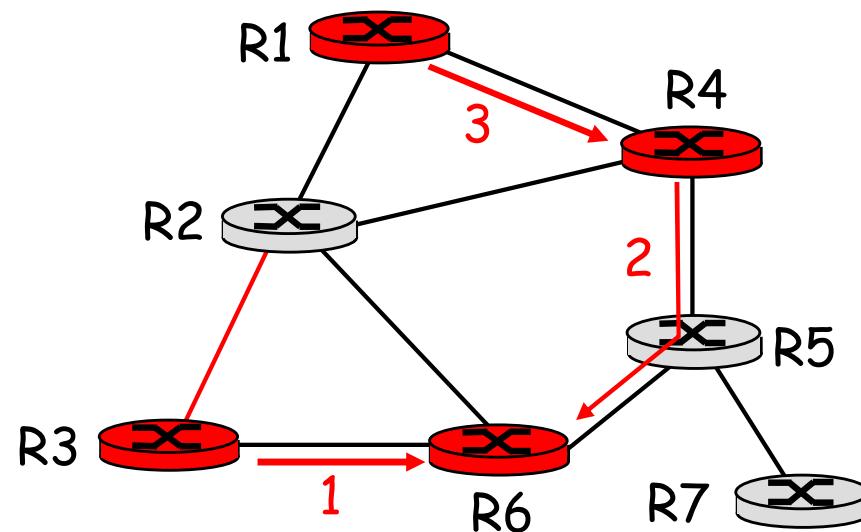
- router with attached group member
- router with no attached group member
- P → prune message
- links with multicast forwarding

## Center-based Trees

- Single delivery tree shared by all.
- One router identified as “*center*” of tree.
- To join:
  - Edge router sends unicast *join-msg* addressed to center router;
  - *Join-msg* “processed” by intermediate routers and forwarded towards center;
  - *Join-msg* either hits existing tree branch for this center, or arrives at center;
  - Path taken by *join-msg* becomes new branch of tree for this router.

## Center-based Trees: an Example

- Suppose R6 chosen as center:



### LEGEND

- router with attached group member
- router with no attached group member
- path order in which join messages generated

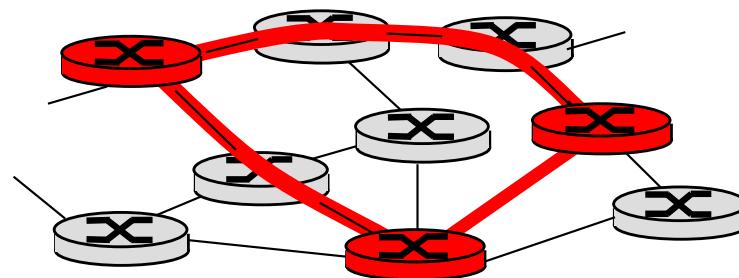
# Internet Multicasting Routing: **DVMRP**

- DVMRP: distance vector multicast routing protocol (RFC1075).
- *Flood and prune:*  
Reverse path forwarding, source-based tree:
  - RPF tree based on DVMRP's own routing tables constructed by communicating DVMRP routers;
  - No assumptions about underlying unicast;
- Initial datagram to Mcast group is flooded everywhere via RPF;
- Routers not wanting group: send upstream prune messages.

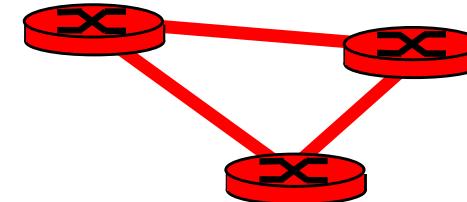
- Soft state: DVMRP router periodically (1 min.) “forgets” that some branches were pruned:
  - Mcast data flows down unpruned branch, again;
  - Downstream router: re-prune or else continue to receive data.
- Routers can quickly regraft to tree:
  - Following IGMP join at leaf.
- Odds and ends:
  - Commonly implemented in commercial routers;
  - Mbone (“multicast backbone”) routing done using DVMRP.

# Tunneling

**Q:** How to connect “islands” of multicast routers in a “sea” of unicast routers?



physical topology



logical topology

- Mcast datagram encapsulated inside “normal” (non-multicast-addressed) datagram;
- Normal IP datagram sent through “tunnel” via regular IP unicast to receiving Mcast router;
- Receiving Mcast router unencapsulates to get Mcast datagram.

# *Protocol Independent Multicast (PIM)*

- Not dependent on any specific underlying unicast routing algorithm (works with all);
- Two different multicast distribution scenarios:

## Dense:

- Group members densely packed, in “close” proximity.
- Bandwidth more plentiful.

## Sparse:

- Number of networks with group members is small compared to the number of interconnected networks;
- Group members are “widely dispersed”;
- Bandwidth not plentiful.

# PIM: *Sparse-Dense Dichotomy*

## Dense:

- Group membership by routers assumed until routers explicitly prune;
- *Data-driven* construction of Mcast tree (e.g., RPF);
- Bandwidth and non-group-router processing *prodigal*.

## Sparse:

- No membership until routers explicitly join;
- *Receiver- driven* construction of Mcast tree (e.g., center-based);
- Bandwidth and non-group-router processing *conservative*.

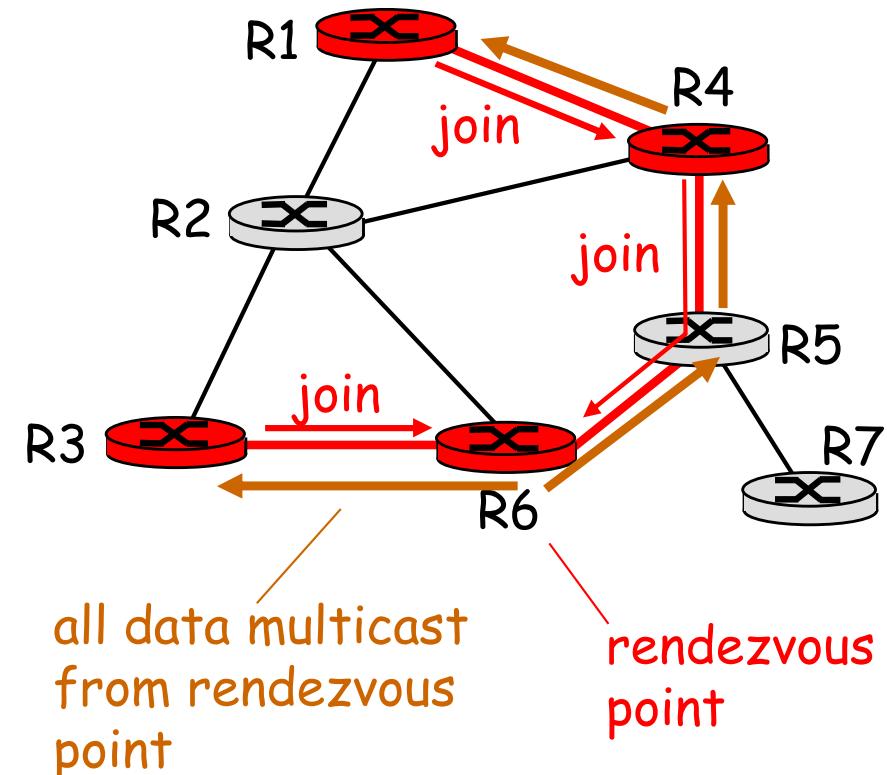
## PIM- Dense Mode

Flood-and-prune RPF, similar to DVMRP but:

- Underlying unicast protocol provides RPF info for incoming datagram;
- Less complicated (more efficient) downstream flood than DVMRP reduces reliance on underlying routing algorithm;
- Has protocol mechanism for router to detect it is a leaf-node router.

# PIM - Sparse Mode

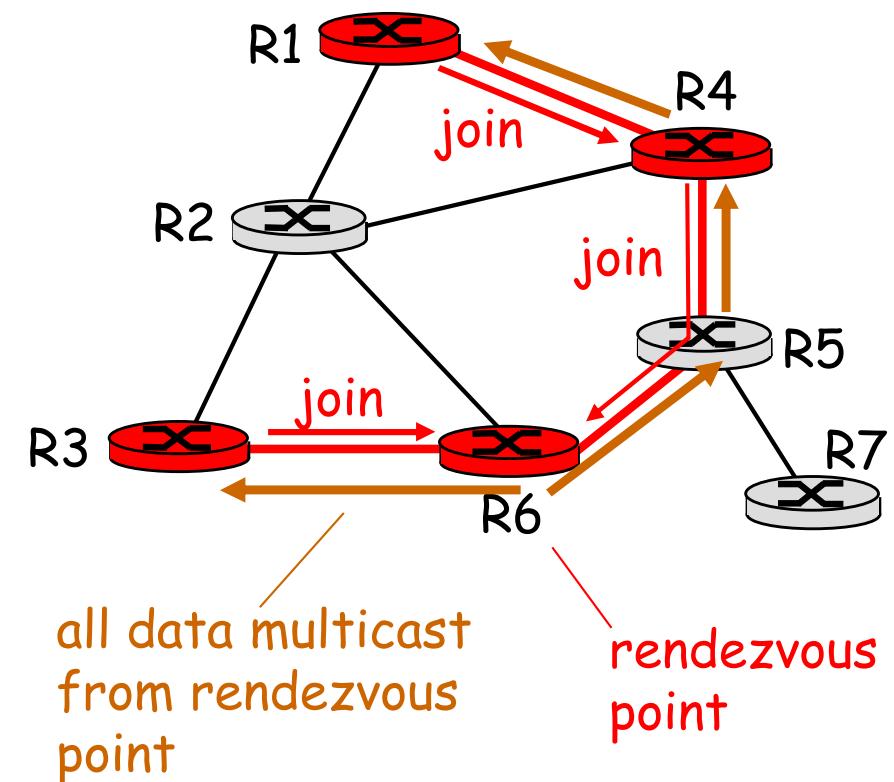
- Center-based approach;
- Router sends *join* message to rendezvous point (RP):
  - Intermediate routers update state and forward *join*;
- After joining via RP, router can switch to source-specific tree:
  - Increased performance: less concentration, shorter paths.



# PIM - Sparse Mode

## Sender(s):

- Unicast data to RP, which distributes down RP-rooted tree;
- RP can extend Mcast tree upstream to source;
- RP can send *stop* message if no attached receivers:
  - “No one is listening!”





# Redes de Computadores

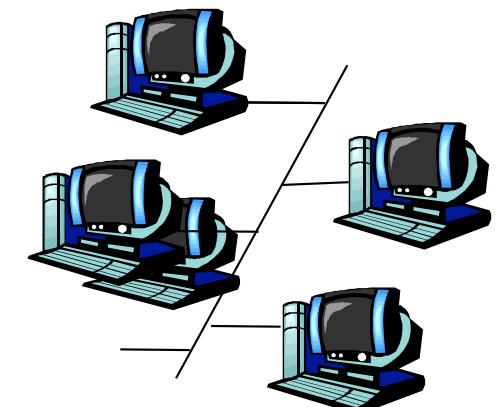
## LEIC-A

### *5 – Data Link Layer*

Prof. Paulo Lobato Correia  
*IST, DEEC – Área Científica de Telecomunicações*

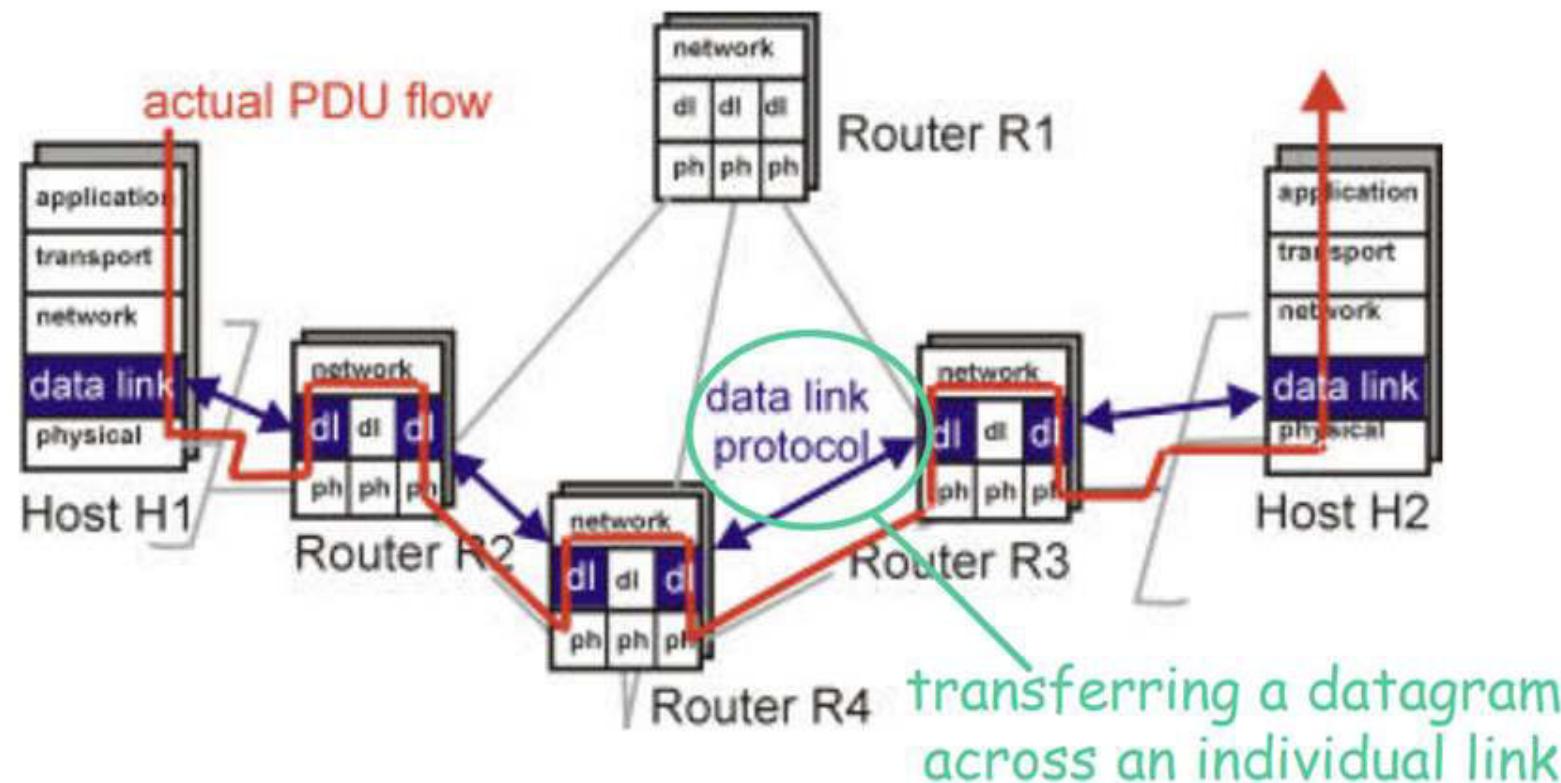
# *Outline*

- Introduction and services
- Link-layer addressing
- Error detection and correction
- Multiple access protocols
- Ethernet
- Link-layer switches
- IEEE 802.11 Wireless LANs
- Framing



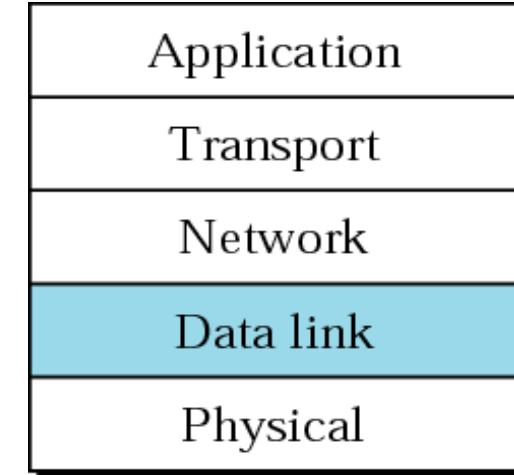
# Data Link Layer: Context

- Datagram transferred using **different link protocols** on **different links**:
  - e.g., Ethernet on first link, optical fiber intermediate link, WiFi on last link;
- Each link protocol provides different services:
  - e.g., may or may not provide reliable data transfer over link.





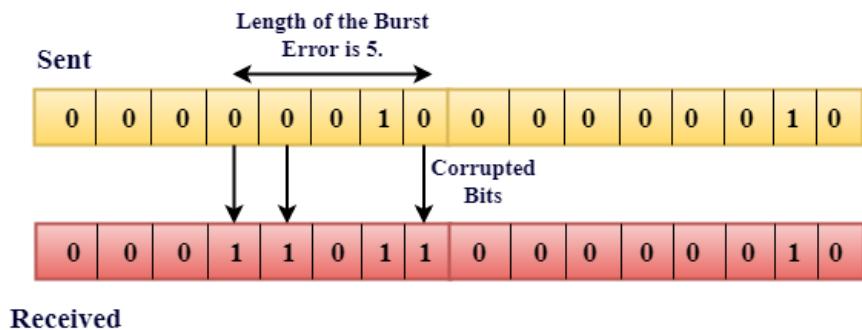
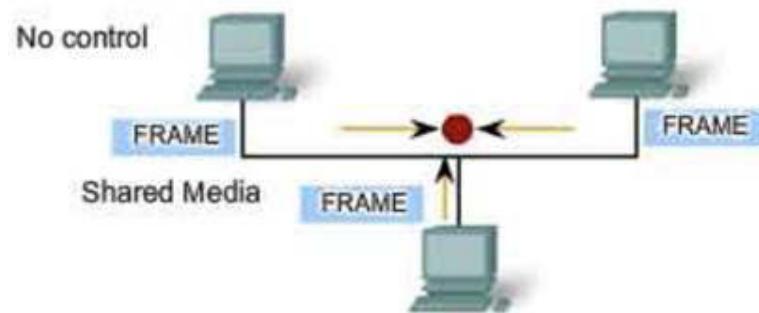
# Data Link Layer



The link layer deals with data transfer between devices belonging to the **same subnet**.

Data link layer tasks:

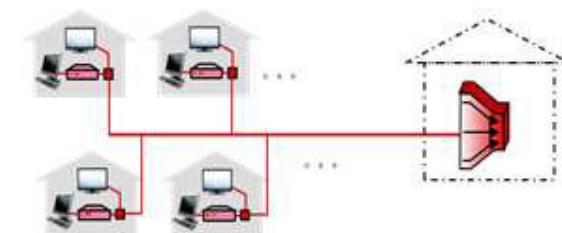
- Framing, Link Access / Media Access Control
- Reliable Delivery, Error Detection and Correction



# Data Link Layer Services

## □ *Framing, Link access :*

- Encapsulate datagram into frame, adding header + trailer;
- **Medium access control (MAC) protocol** - to access shared medium;
- “MAC” addresses used in frame headers to identify source, destination:
  - Different from IP address!



## □ *Reliable delivery between adjacent nodes:*

- Similar solutions to those adopted in transport layer;
- Seldom used on low bit-error links (fiber, some twisted pair);
- Wireless links: high error rates.

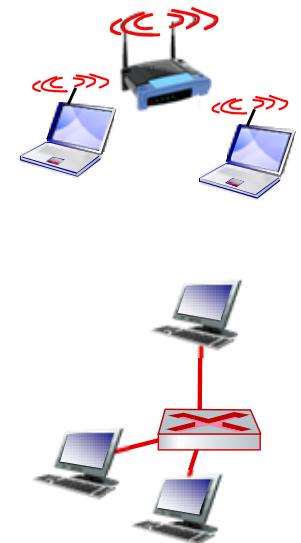
Q: Why both link-level and end-end reliability?



# Data Link Layer Services

## □ *Error detection:*

- Errors caused by signal attenuation and noise;
- Receiver may detect presence of errors:
  - Signals sender for *retransmission* or *drops frame*.



## □ *Error correction:*

- Receiver may identify *and correct* bit error(s) without resorting to retransmission.

## □ *Flow control:*

- To adjust pace between adjacent sending and receiving nodes;

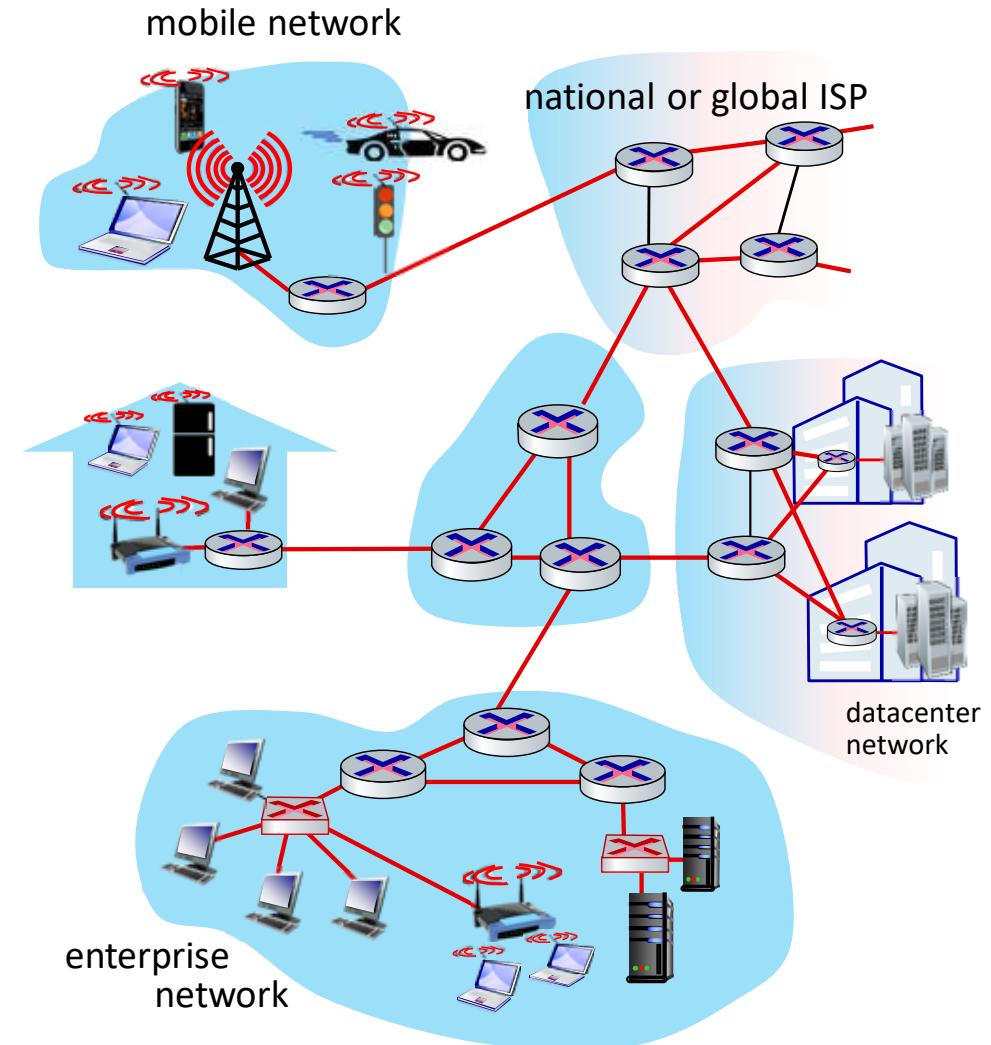
## □ *Half-duplex and full-duplex links:*

- With half duplex, nodes at both ends of link can transmit, but not at same time.

# Data Link Layer: Introduction

## Some terminology:

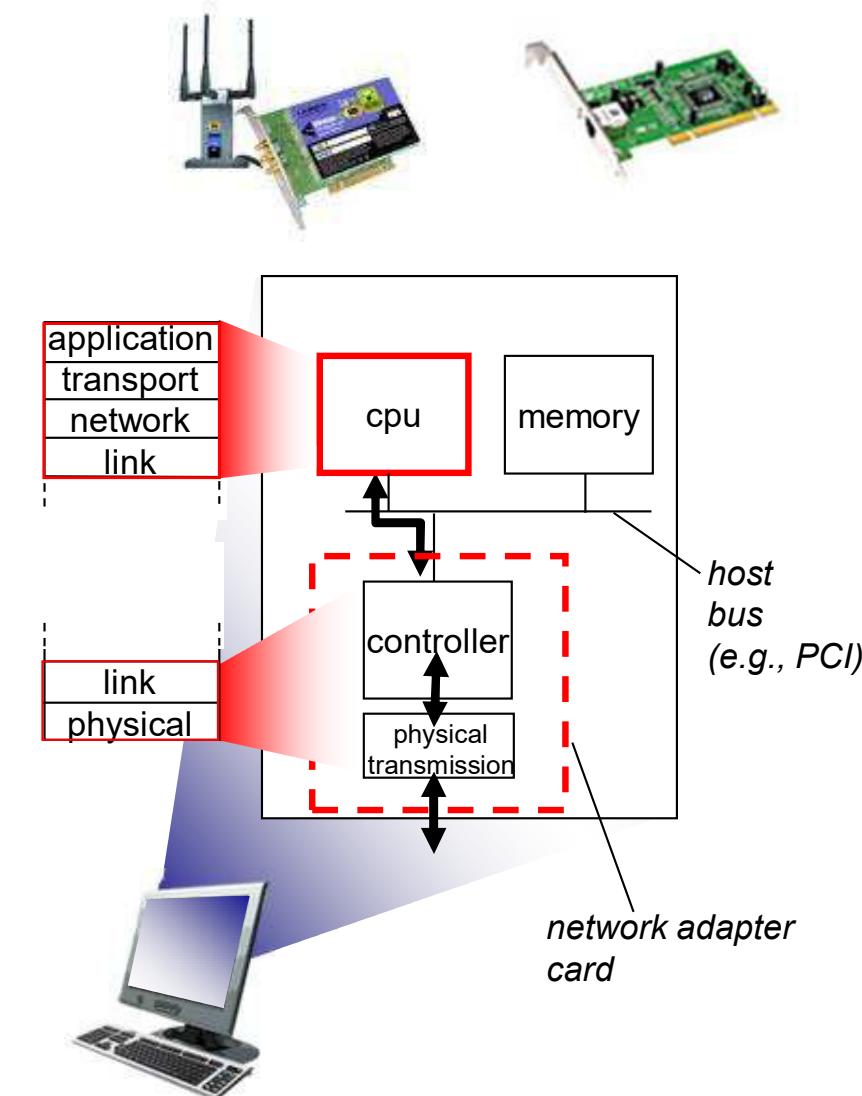
- Hosts and routers are **nodes**;
- Communication channels that connect adjacent nodes along communication path are **links**:
  - Wired links;
  - Wireless links;
  - LANs.
- Layer-2 packet is a **frame**.  
It encapsulates a datagram.



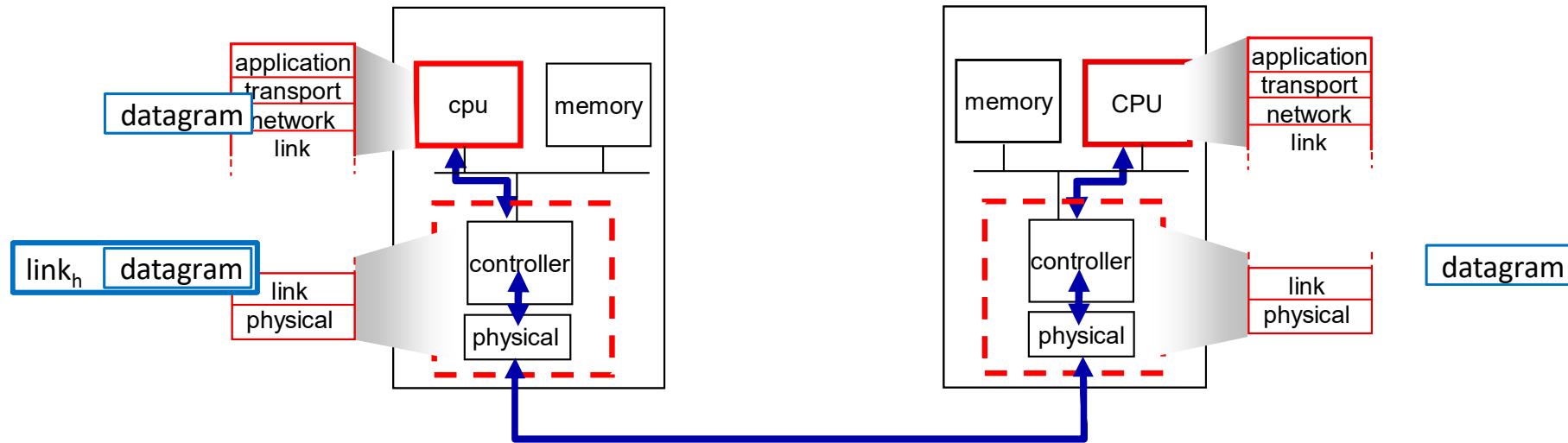
*Data link layer has the responsibility of **transferring datagrams** from one node to the physically adjacent node over a link*

# Where is the Link Layer Implemented?

- In each and every host;
- Link layer implemented in the “adaptor” (aka *network interface card* - NIC):
  - Ethernet card, PCMCI card, 802.11 card;
  - Implements link, physical layer.
- Attaches into host’s system buses;
- Combination of hardware, software, firmware.



# Interfaces Communicating



Sending side:

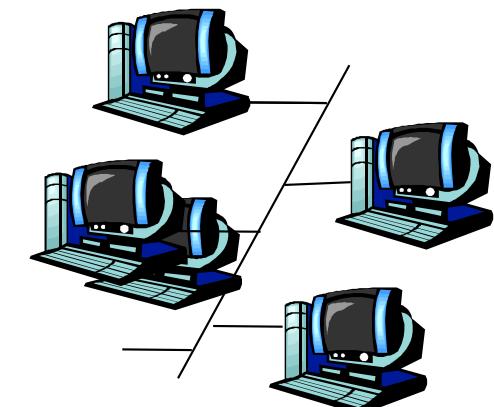
- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

Receiving side:

- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

# *Outline*

- Introduction and services
- Link-layer Addressing
- Error detection and correction
- Multiple access protocols
- Ethernet
- IEEE 802.11 Wireless LANs
- Link-layer switches
- Framing

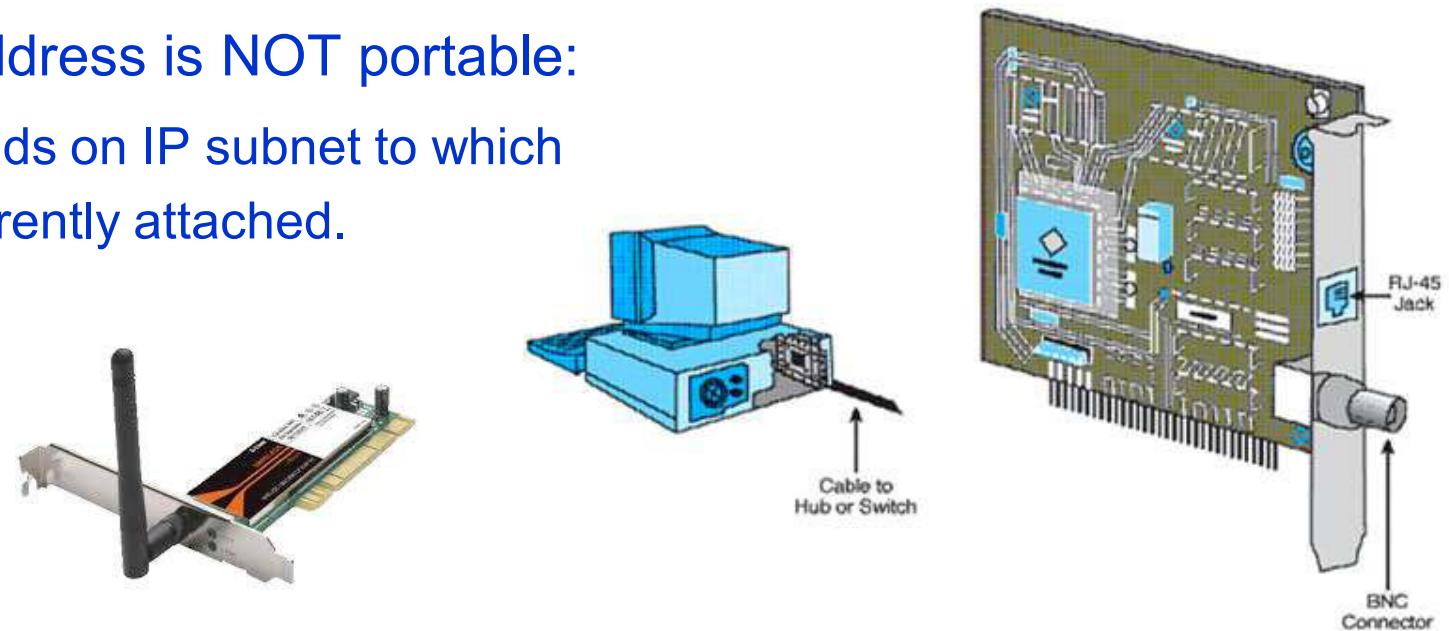


# Link-Layer Addressing

- Network Layer: IP address
  - Used to get datagram to the destination IP subnet;
  - 32-bit IPv4 address;
  - **Hierarchical**;
  - Not portable.
- Data Link Layer: MAC (or LAN or **physical**) address
  - Function: *get frame from one interface to another physically-connected interface (within the same network)*;
  - 48 bit MAC address (for most LANs):
    - Burned in NIC ROM; Sometimes adjustable by software;
  - **Portable**;

# MAC Addresses

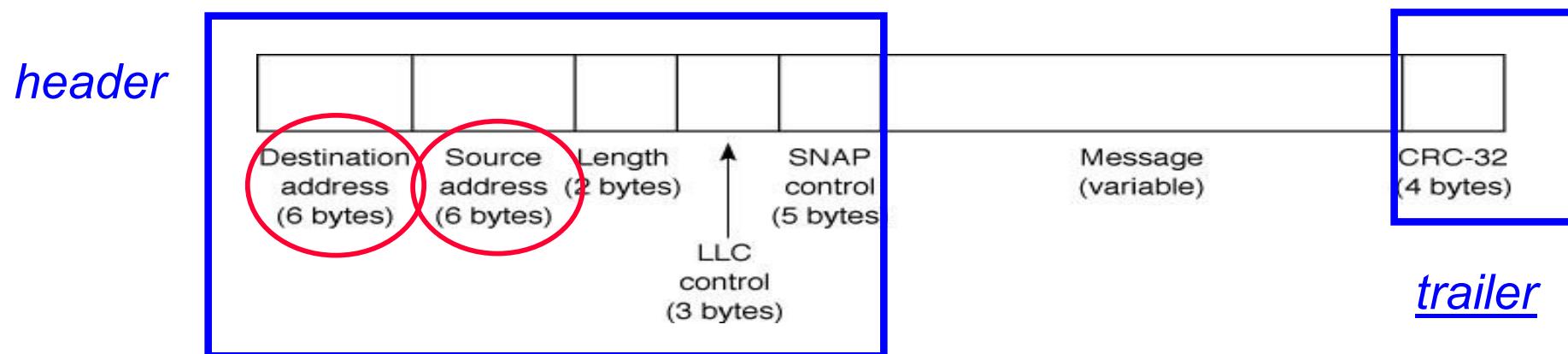
- Analogy:
  - (a) MAC address is like the “Cartão de Cidadão” Number;
  - (b) IP address is like the *postal address*;
- MAC flat address → portability:
  - Can move network interface card from one LAN to another;
- IP hierarchical address is NOT portable:
  - Address depends on IP subnet to which the node is currently attached.



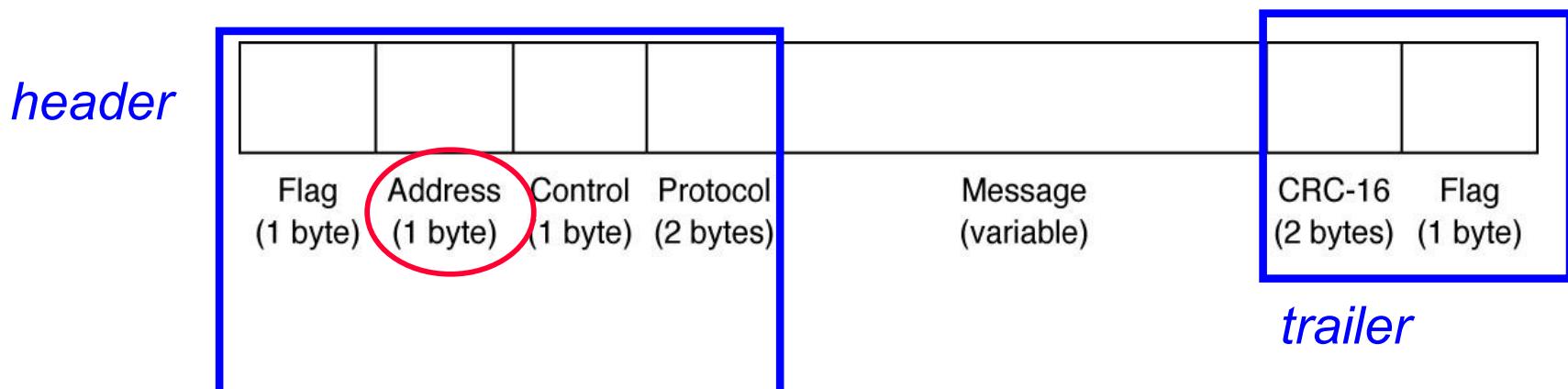
# Link-Layer Addressing

Examples:

- A Ethernet LAN uses 48 bit MAC addresses:

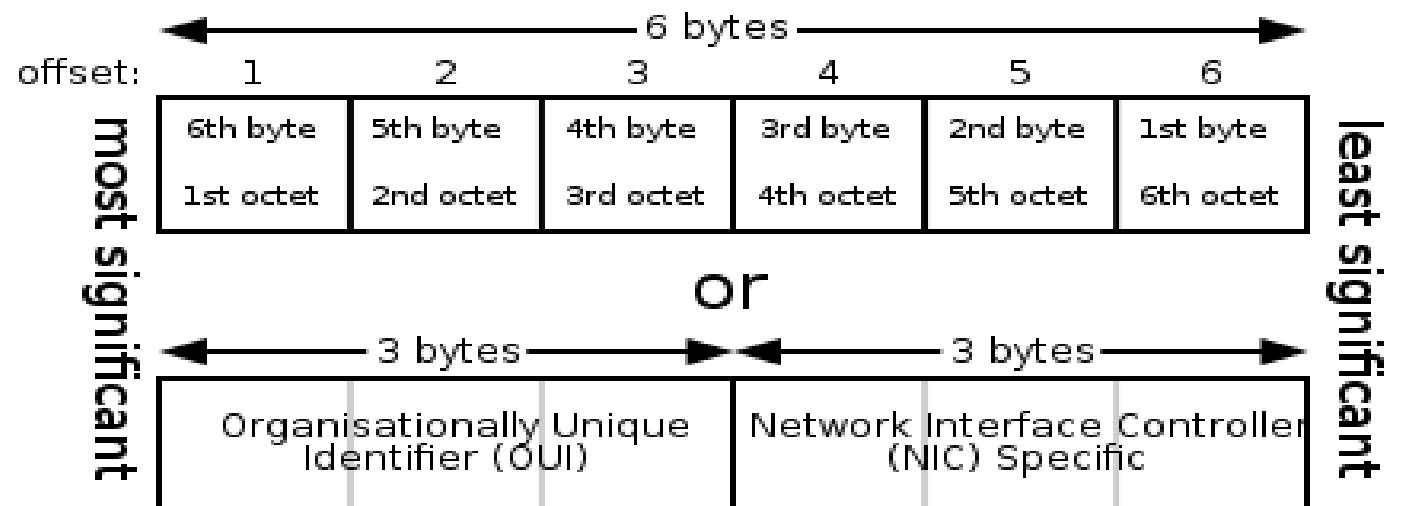


- A PPP connection (typical of dial-up accesses) uses 8 bit addresses:



## **MAC Addresses**

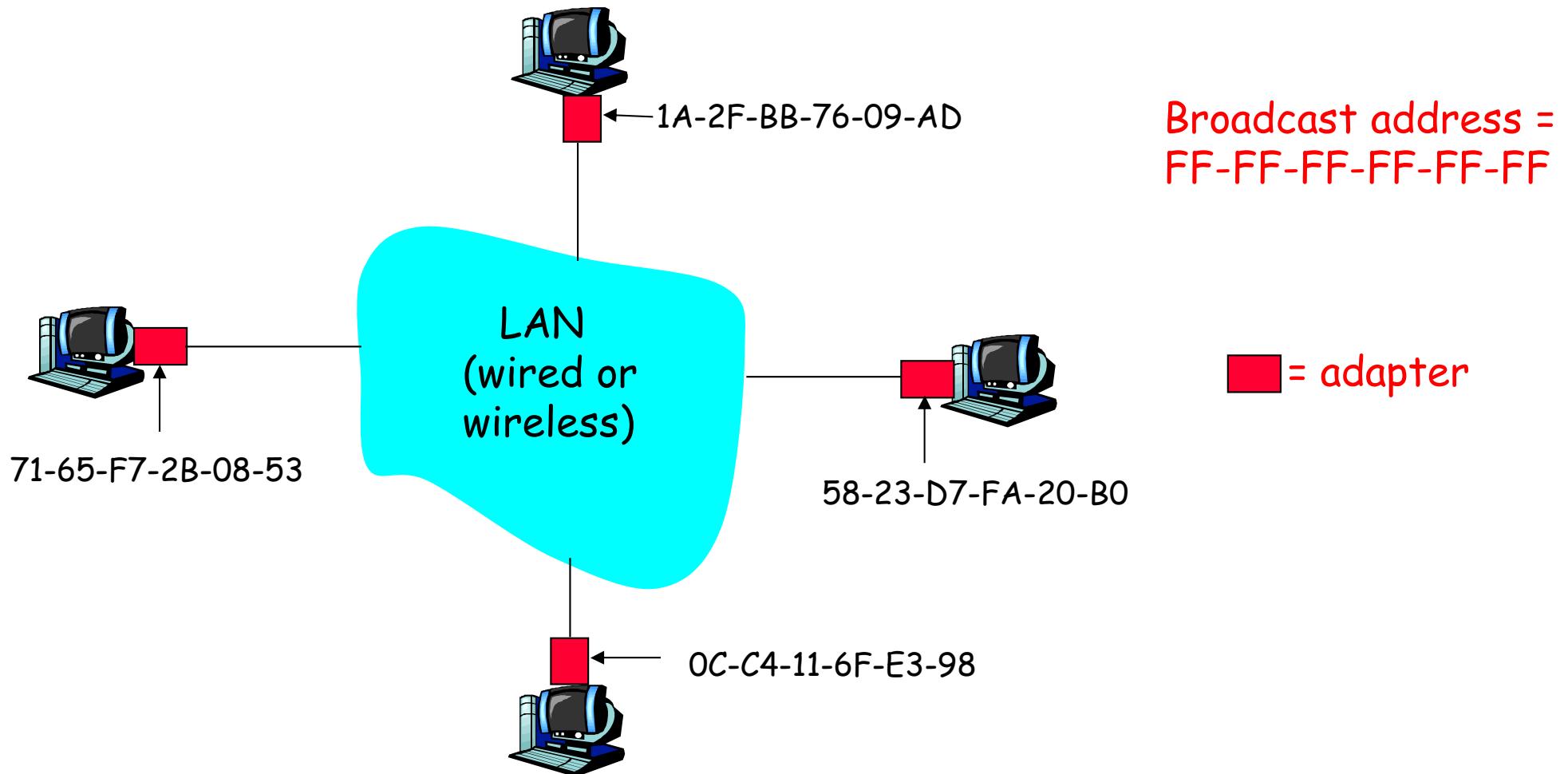
- MAC address allocation is administered by IEEE Registration Authority;
  - A manufacturer buys portion of MAC address space (to assure uniqueness);
  - Number of available addresses:  $2^{48} = 281\ 474\ 976\ 710\ 656$
  - MAC Address (Ethernet) – 48 bits: **MM-MM-MM-SS-SS-SS**



1A-2F-BB-76-09-AD

# MAC Addresses

Each adapter on a LAN has a unique MAC address



# Addressing

```
> ipconfig /all
```

```
C:\>ipconfig /all

Windows IP Configuration

  Host Name . . . . . : magalhaes
  Primary Dns Suffix . . . . . :
  Node Type . . . . . : Unknown
  IP Routing Enabled. . . . . : No
  WINS Proxy Enabled. . . . . : No

Ethernet adapter Wireless Network Connection:

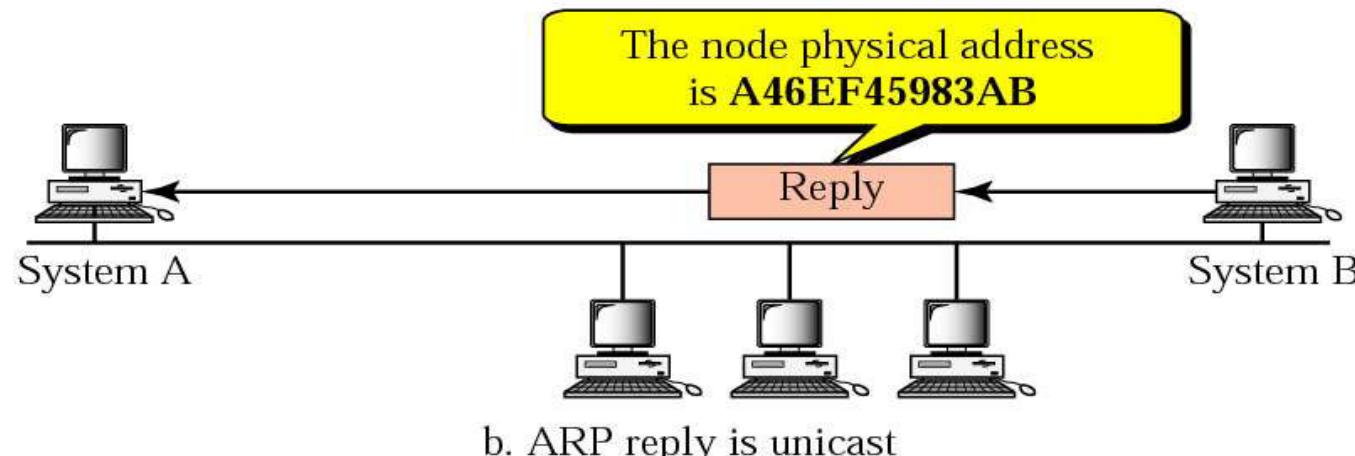
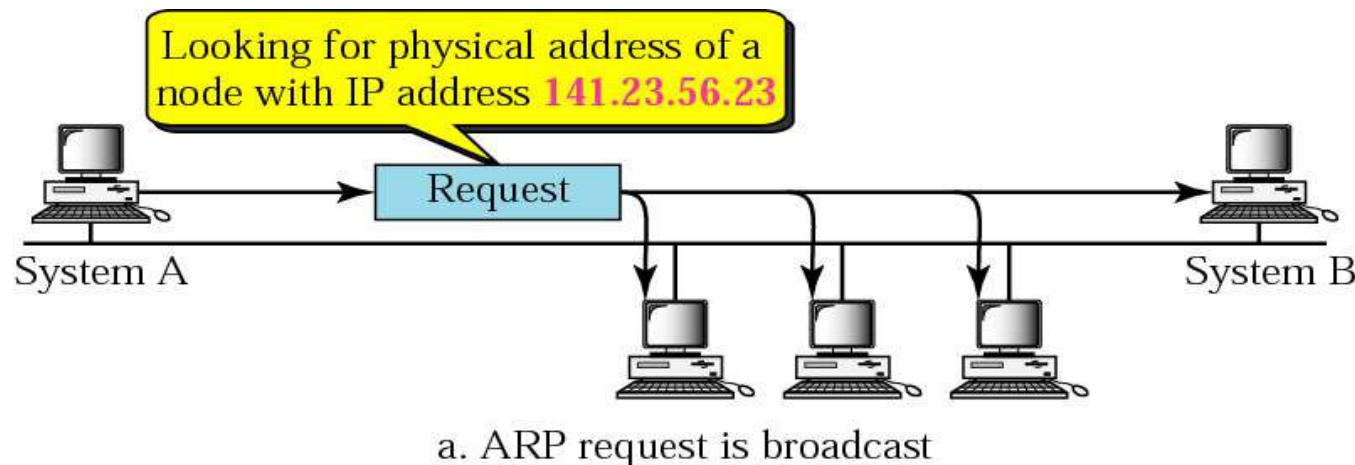
  Media State . . . . . : Media disconnected
  Description . . . . . : Intel(R) PRO/Wireless LAN 2100 3B Mini PCI Adapter
  Physical Address . . . . . : 00-04-23-5F-CA-08

Ethernet adapter Local Area Connection:

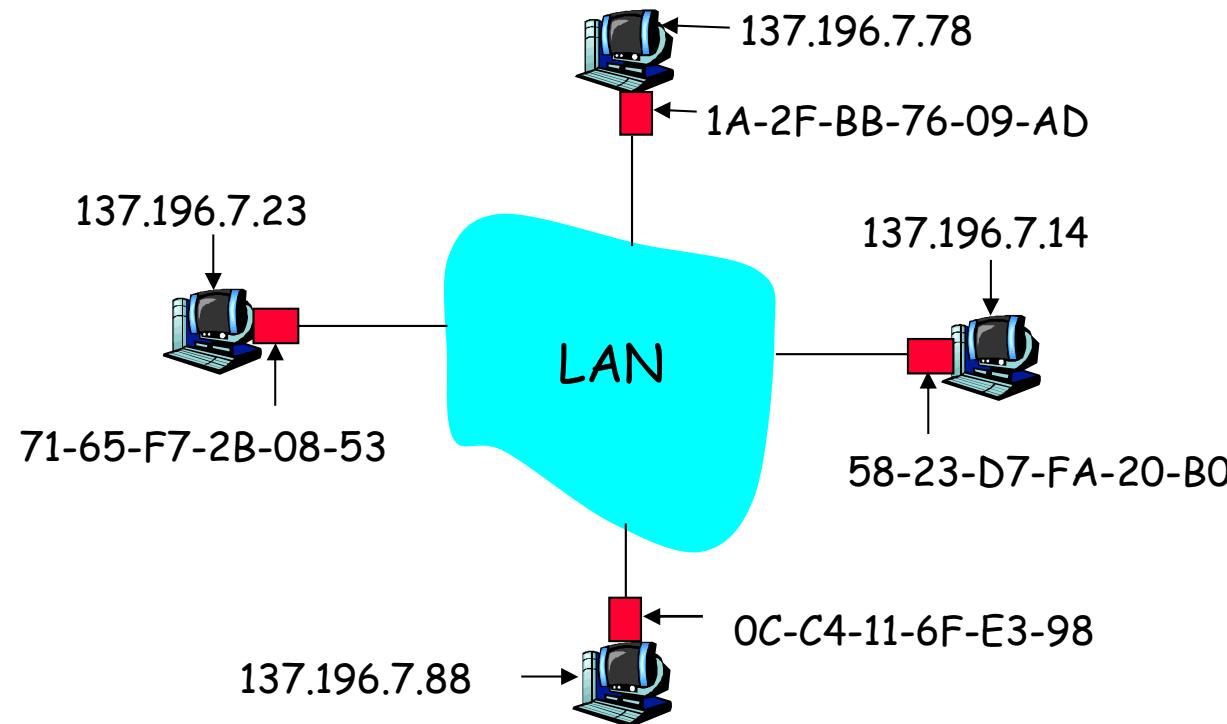
  Media State . . . . . : Media disconnected
  Description . . . . . : Realtek RTL8139/810x Family Fast Ethernet NIC
  Physical Address . . . . . : 00-0C-6E-76-8B-D0
```

# Address Resolution Protocol (ARP)

To deliver frames the physical (MAC) address of the host needs to be known.  
**ARP** allows to get the MAC address from the IP address.



# Address Resolution Protocol (ARP)



- Each IP node (host, router) on a LAN keeps an **ARP table**;
- ARP 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

(used in the same LAN)

Example:

- A wants to send datagram to B, and 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 machines on LAN receive the ARP query.
- B receives ARP packet and **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.*

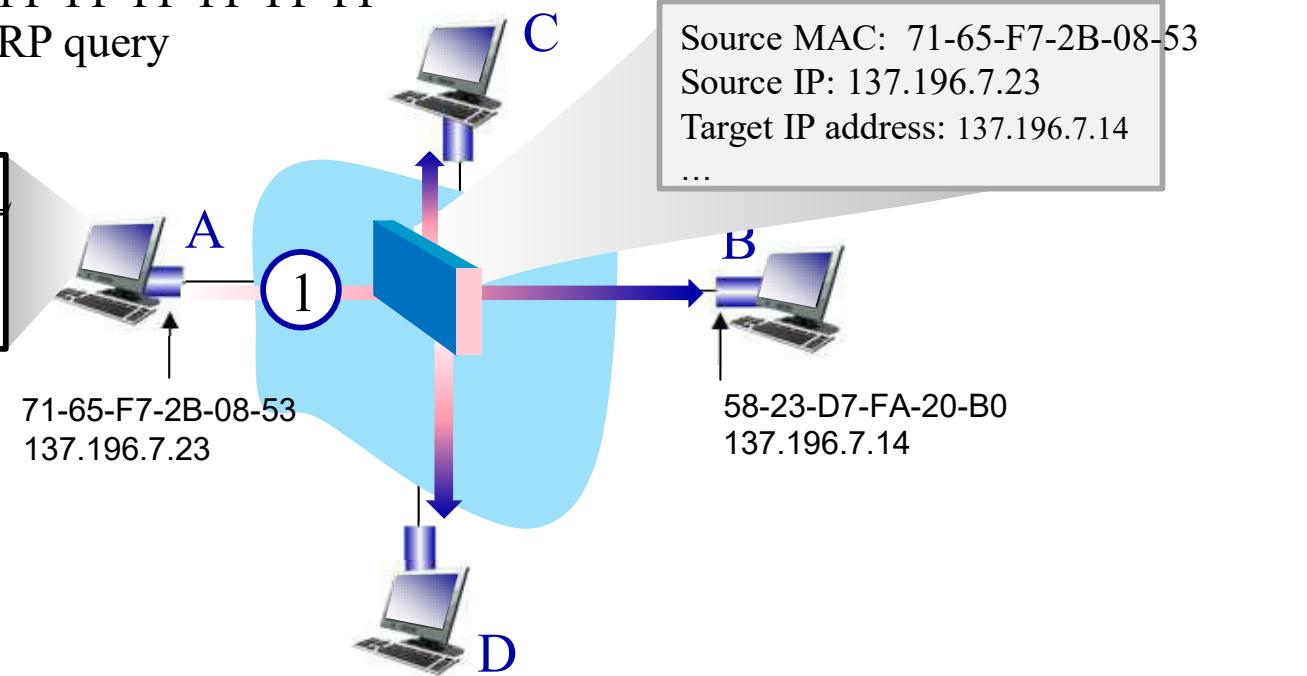
# ARP protocol in action

example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

- ① A broadcasts ARP query, containing B's IP addr
- destination MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query

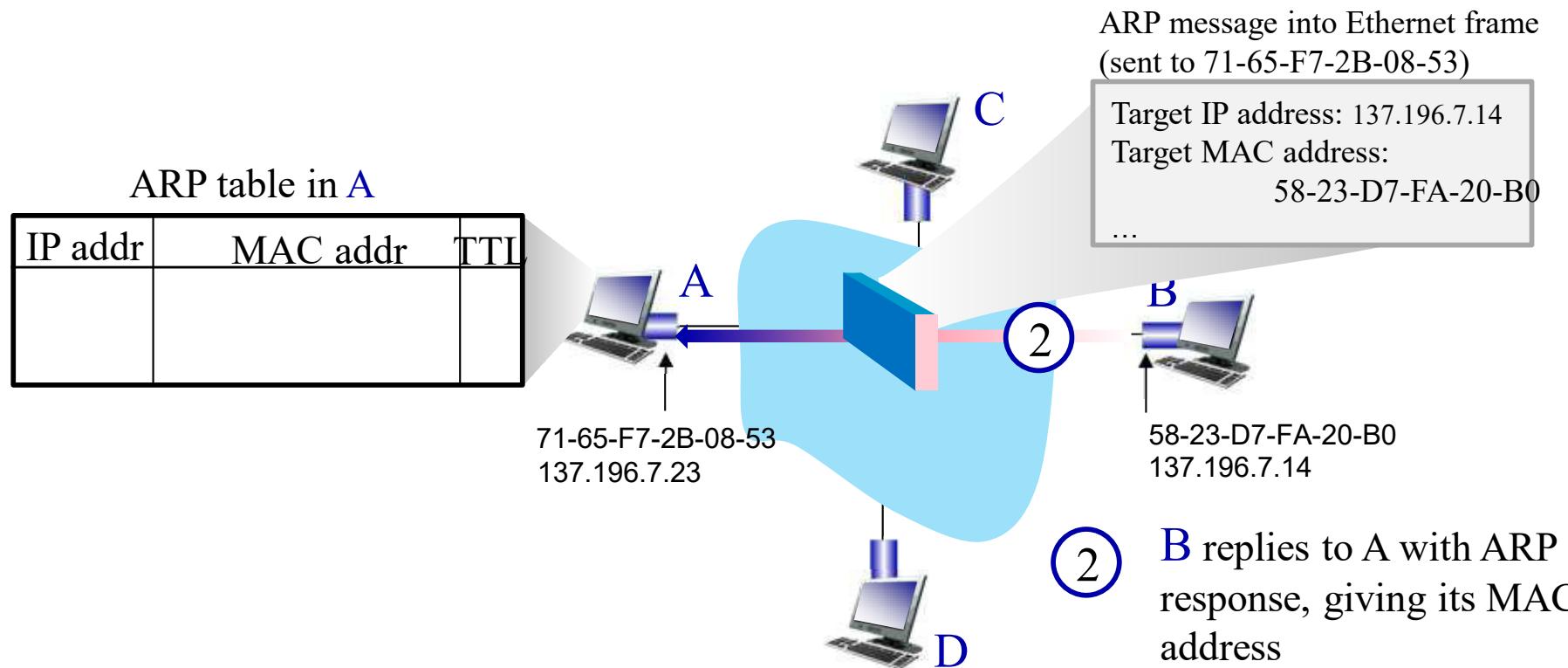
IP addr	MAC addr	TTI



# ARP protocol in action

example: A wants to send datagram to B

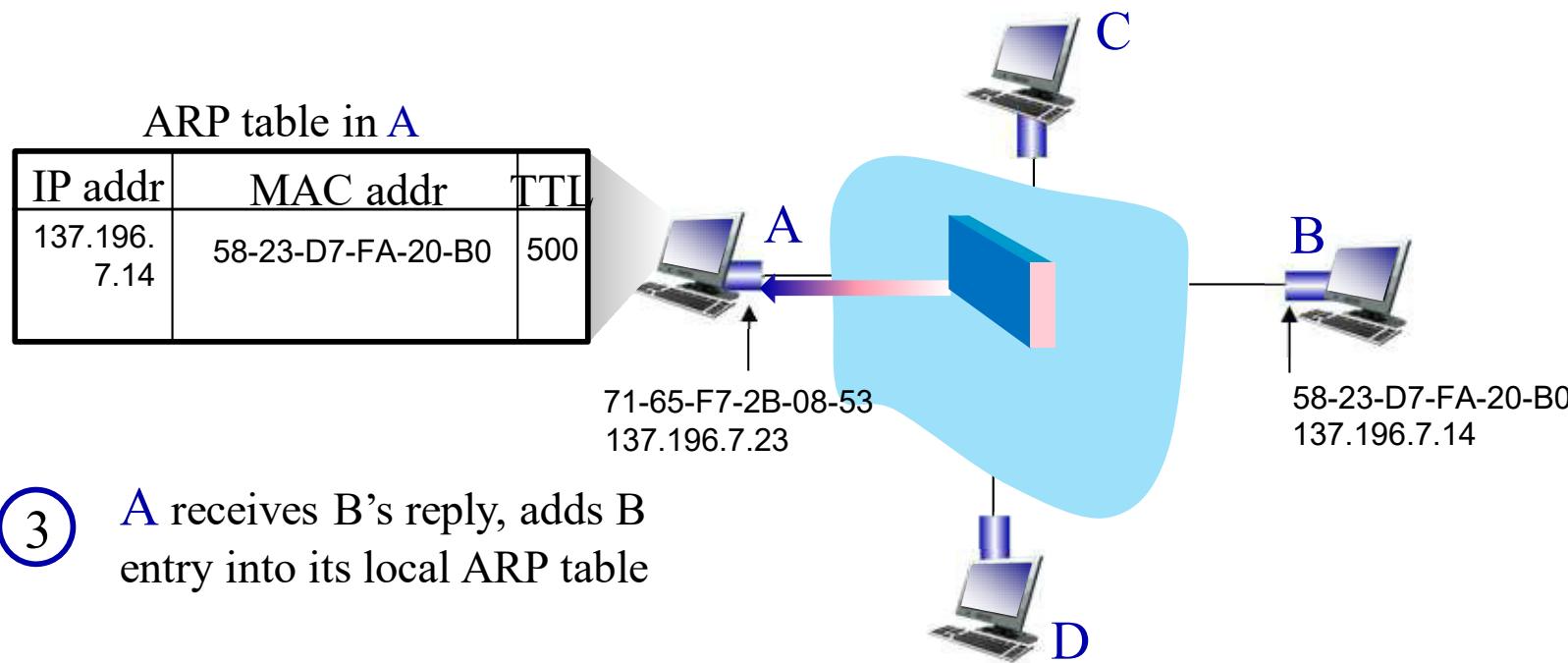
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# *ARP protocol in action*

example: A wants to send datagram to B

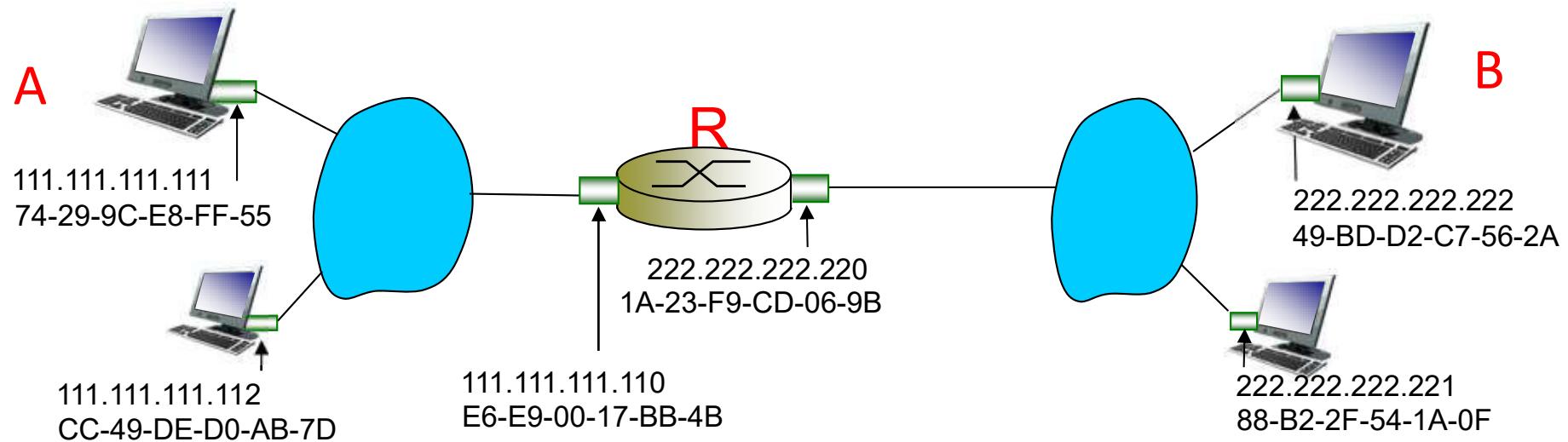
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# *Addressing: Routing to Another LAN*

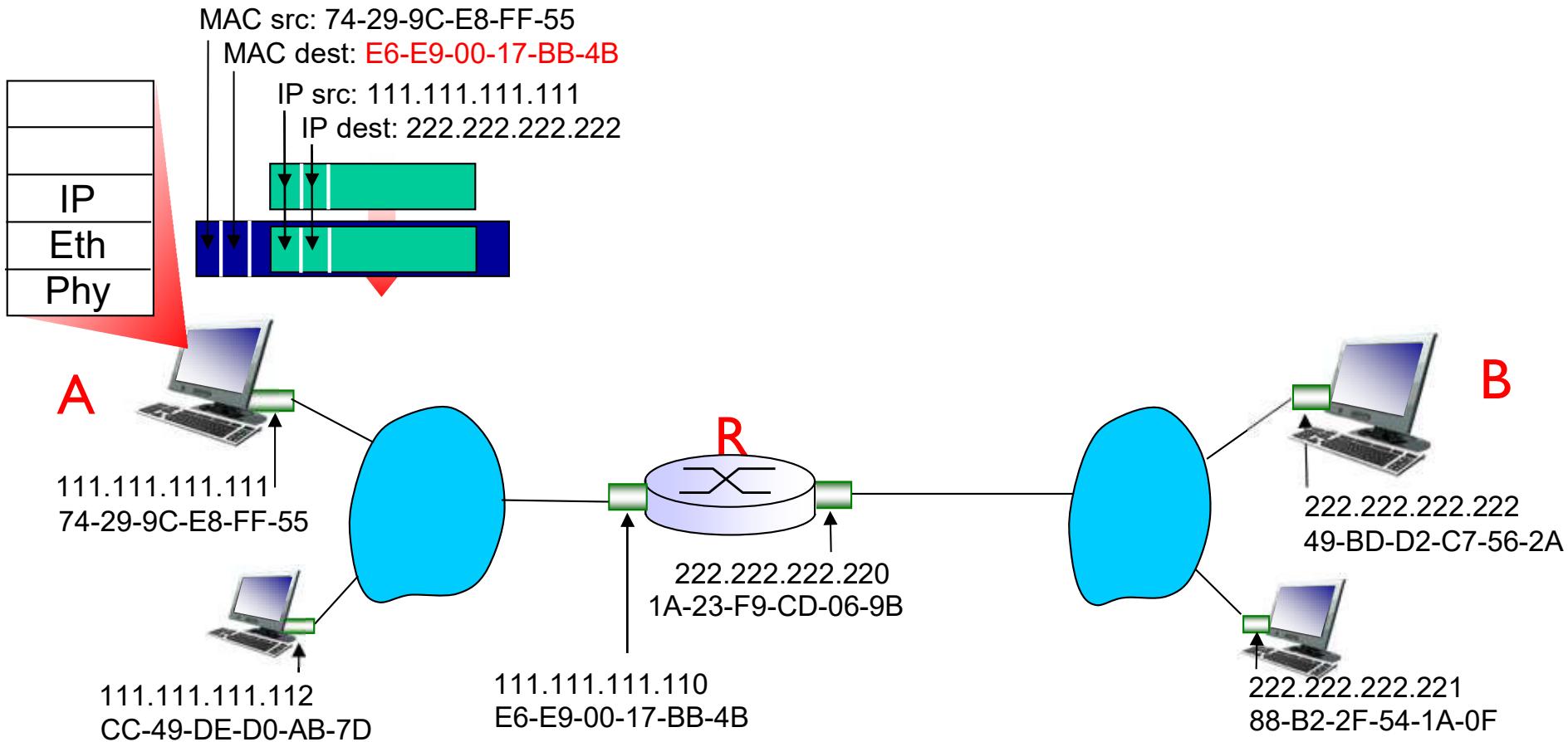
Send datagram from **A** to **B** via **R**:

- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)



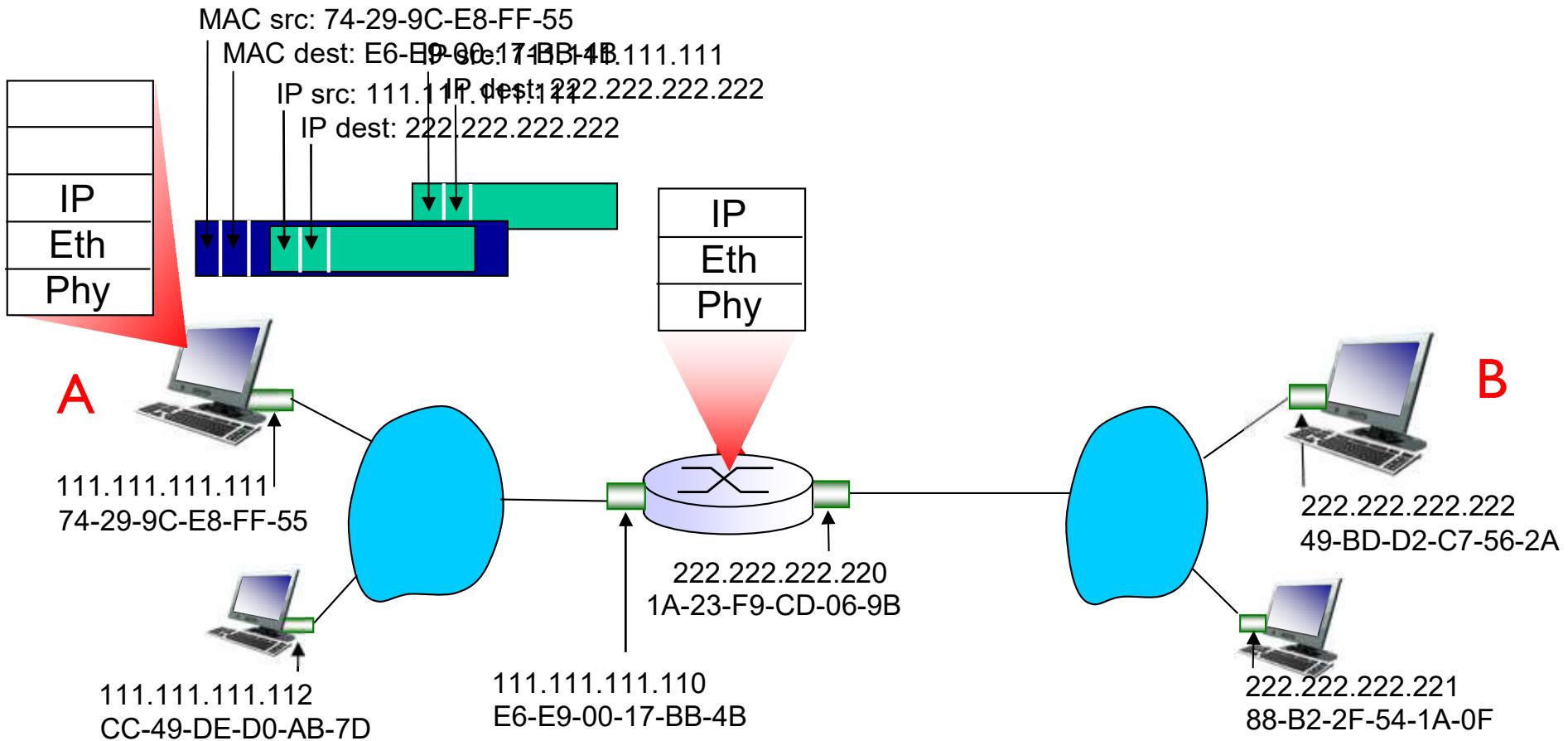
# *Addressing: Routing to Another LAN*

- A – creates IP datagram with: source *IP A*, destination *IP B*
- A – creates **link-layer frame** with: destination *R*'s MAC address, data field: *A-to-B IP datagram*



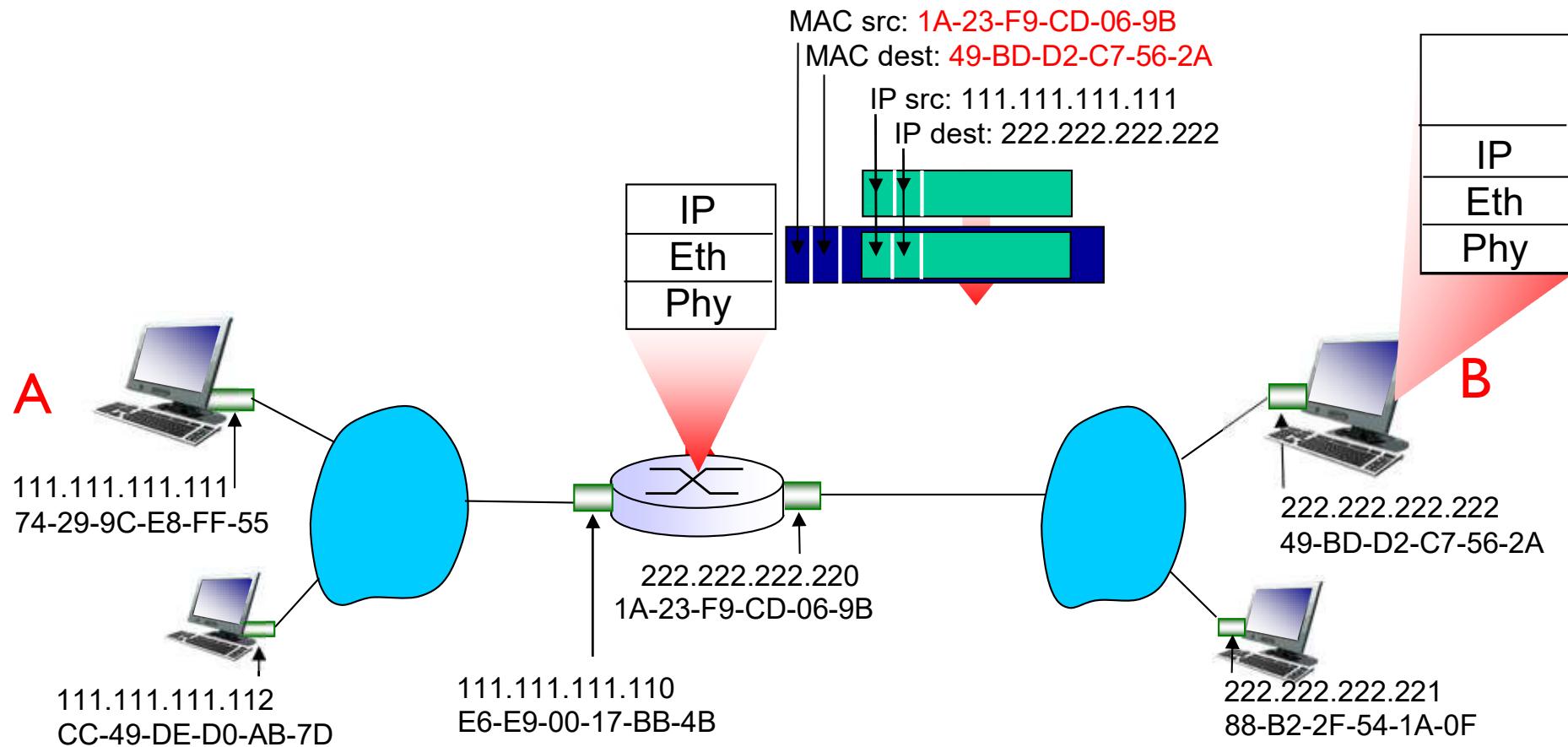
# Addressing: Routing to Another LAN

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



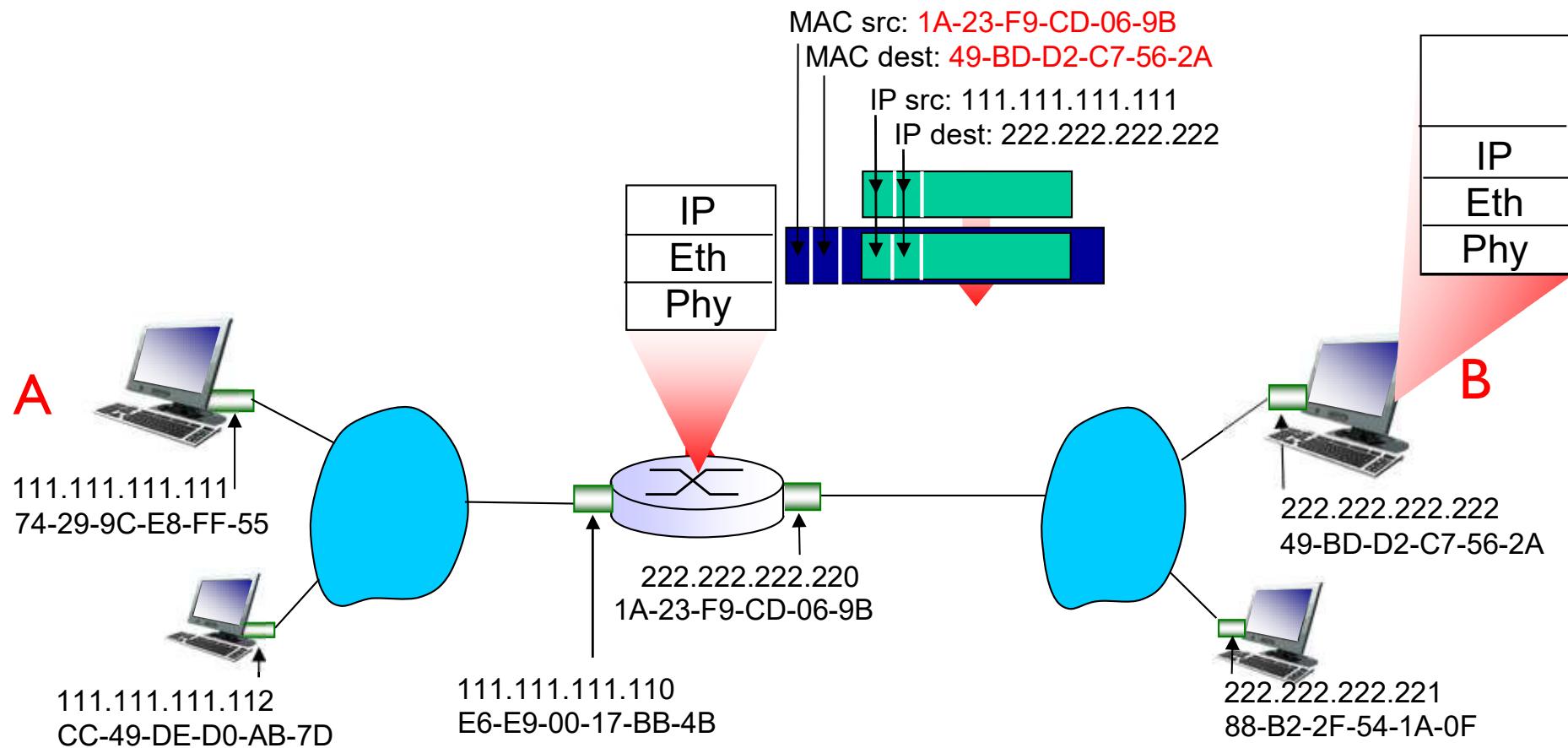
# *Addressing: Routing to Another LAN*

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram

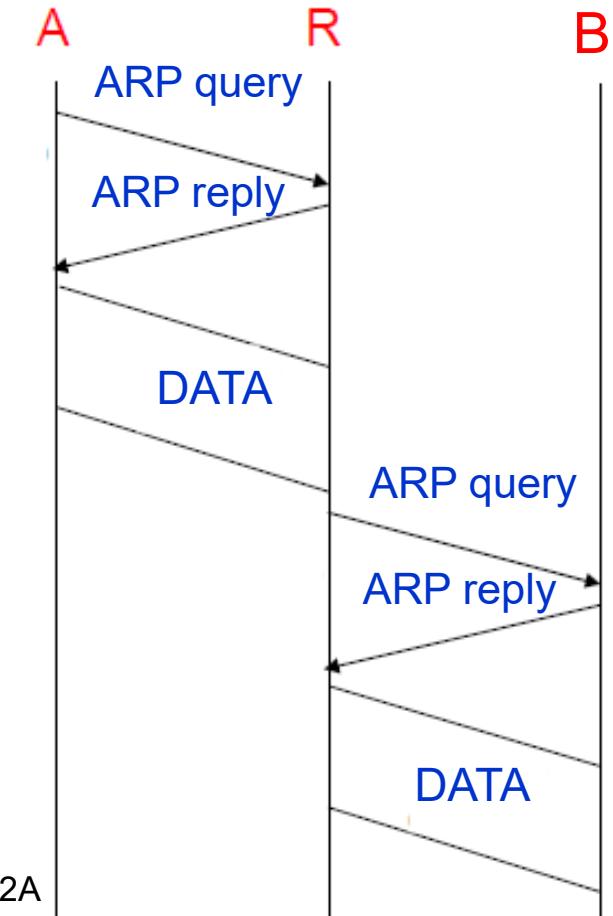
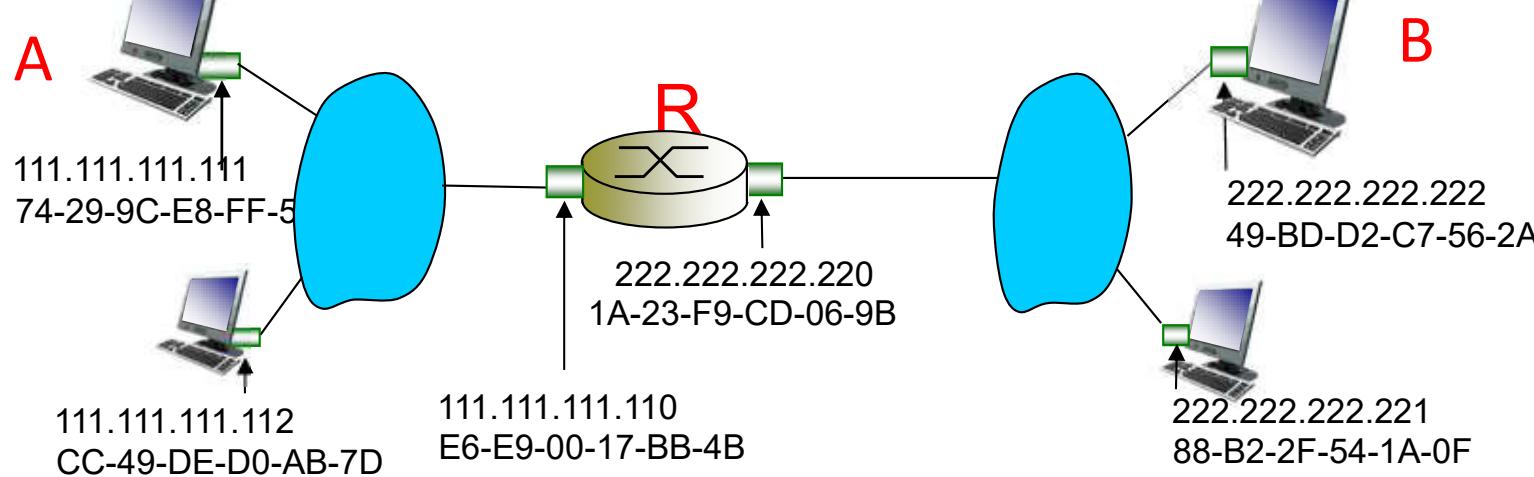


# *Addressing: Routing to Another LAN*

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram

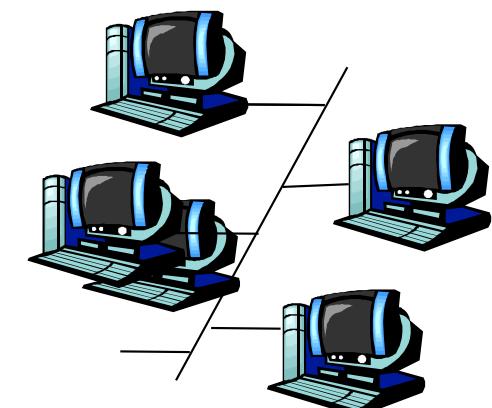


# *Addressing: Routing to Another LAN*



# Outline

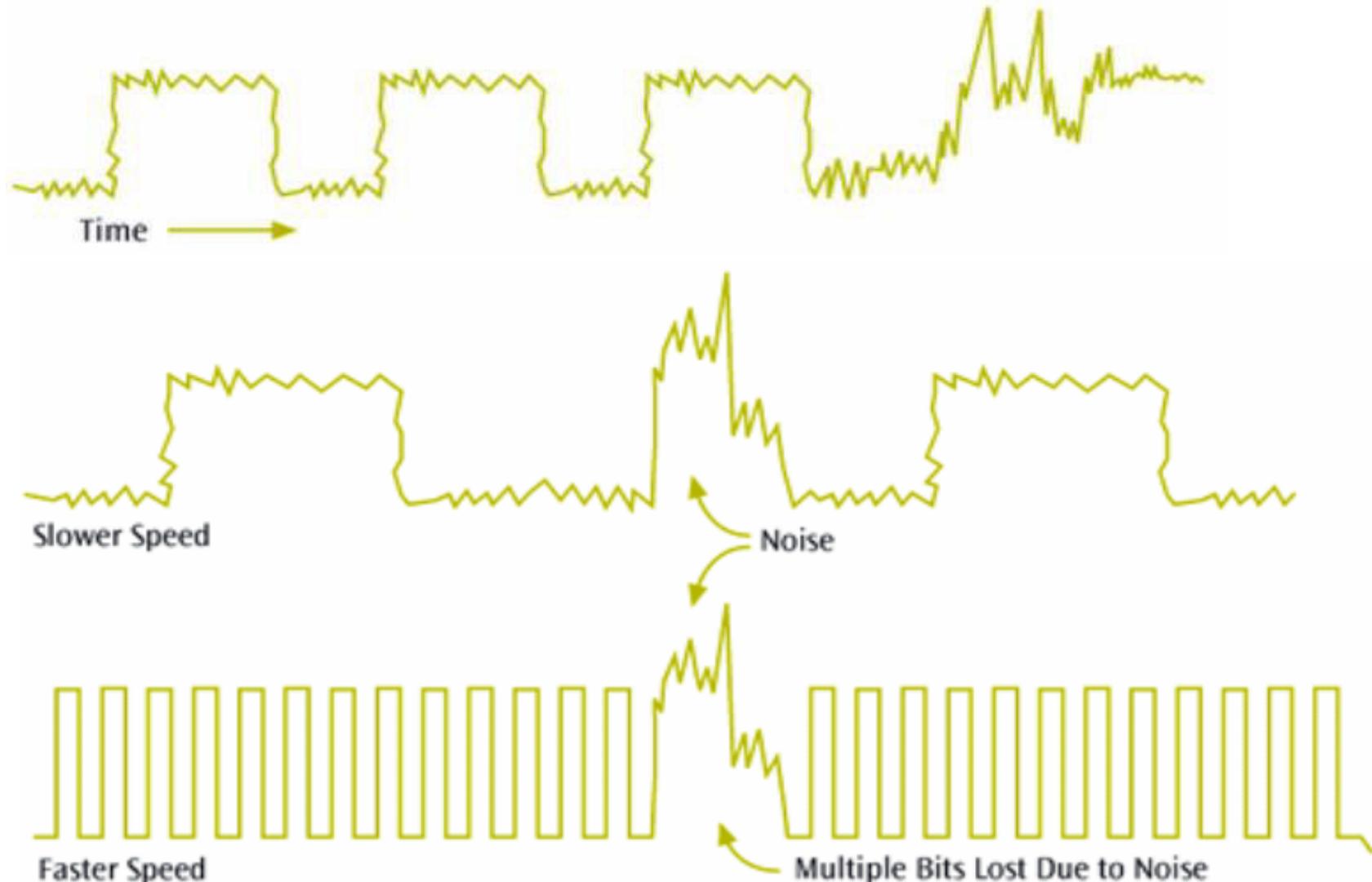
- Introduction and services
- Link-layer Addressing
- Error detection and correction
- Multiple access protocols
- Ethernet
- Link-layer switches
- IEEE 802.11 Wireless LANs
- Framing



# *Why are there Transmission Errors?*

Origin	Cause	Prevention
<b>Line break</b>	Storms, accidents	
<b>White noise</b>	Electron motion	Raise signal level
<b>Impulsive noise</b>	Lightening, voltage changes, car ignition, ...	Isolate or move wires
<b>Cross-talk</b>	Guard bands too small, Wires too close	Increase guard band or isolate wires
<b>Eco</b>	Bad quality connections	Fix or adjust equipment
<b>Loss/attenuation</b>	Signal intensity decreases with distance	Use repeaters or regenerators
<b>Intermodulation noise</b>	Combination of signals with different origins	Isolate or move wires
<b>Jitter</b>	Phase changes in the signals	Adjust the equipments
<b>Harmonic distortion</b>	Non-linear amplification in frequency	Adjust the equipments

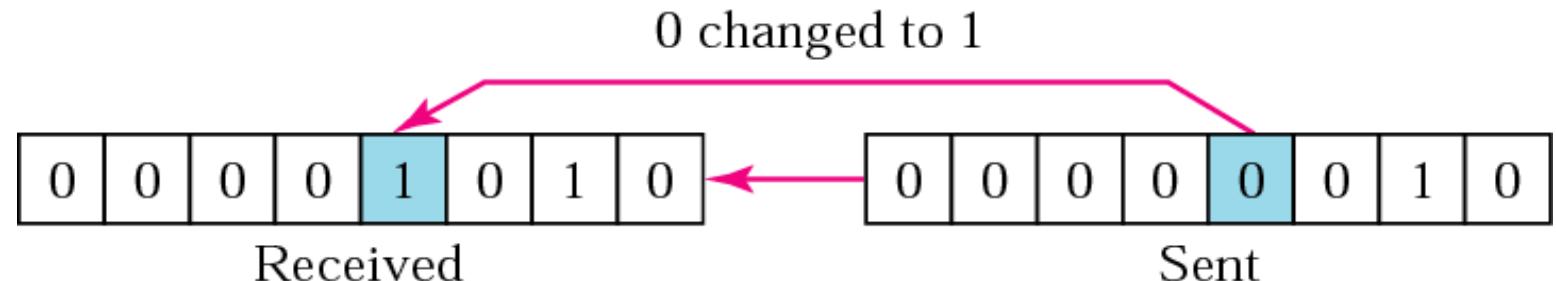
## Error Examples



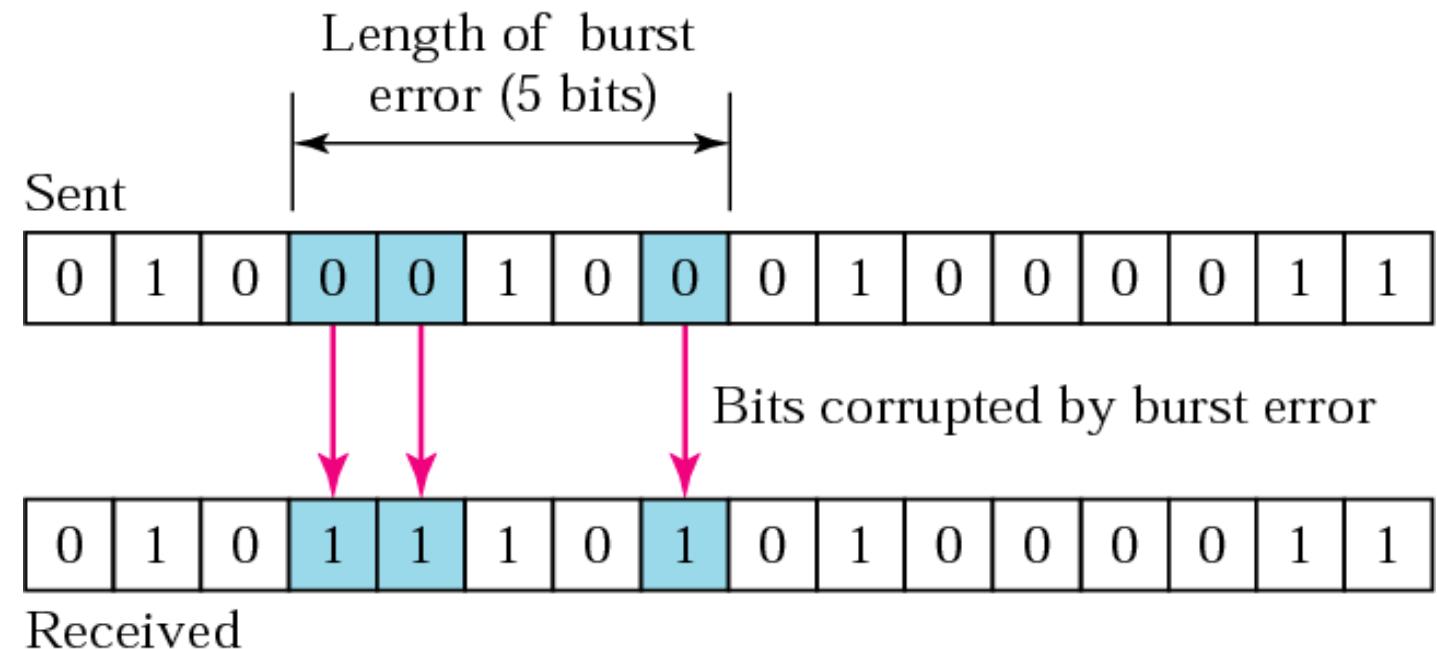
# Error Types

## Error types:

- Isolated:



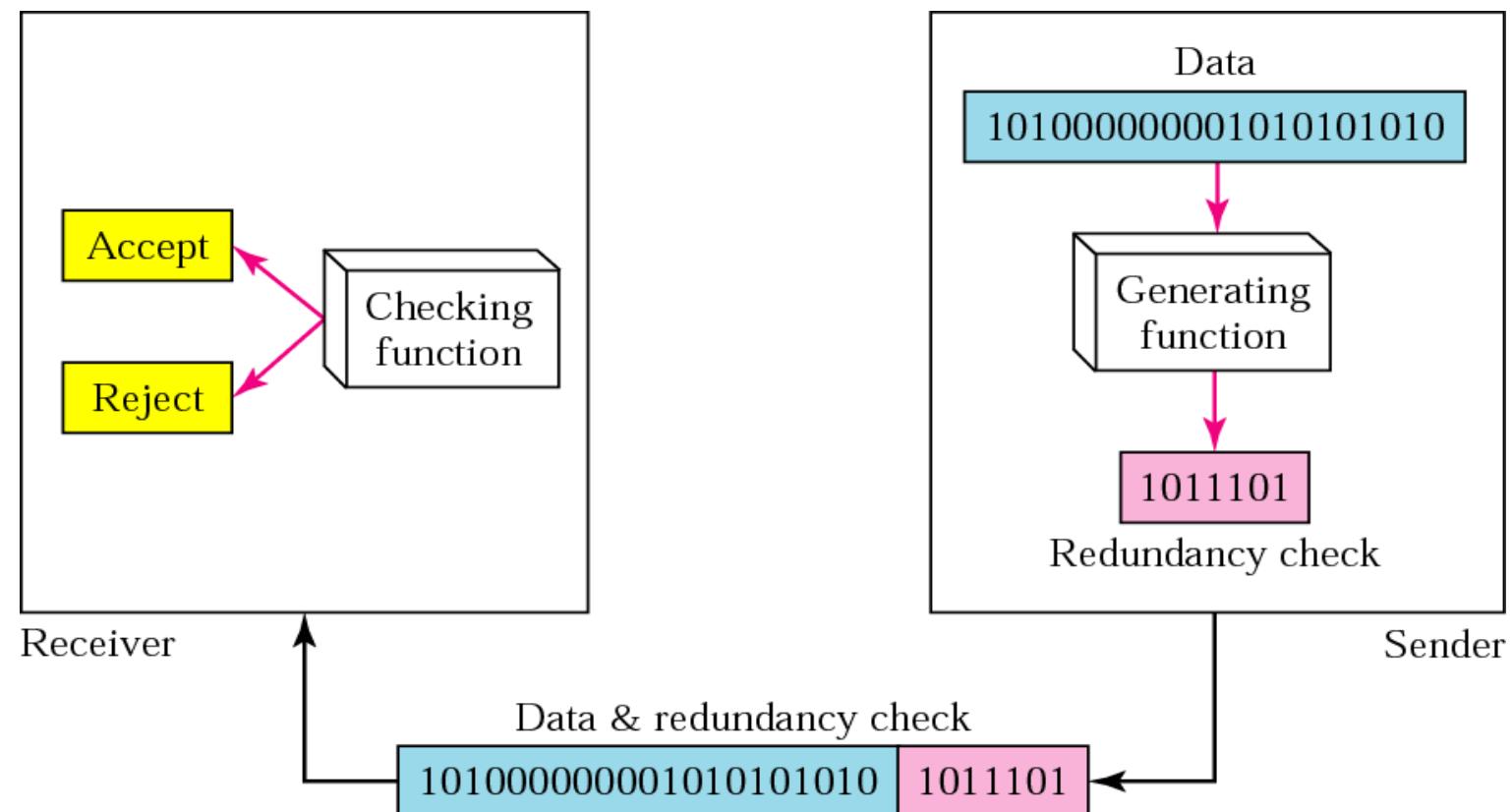
- Burst:



# Error Detection

## Error detection:

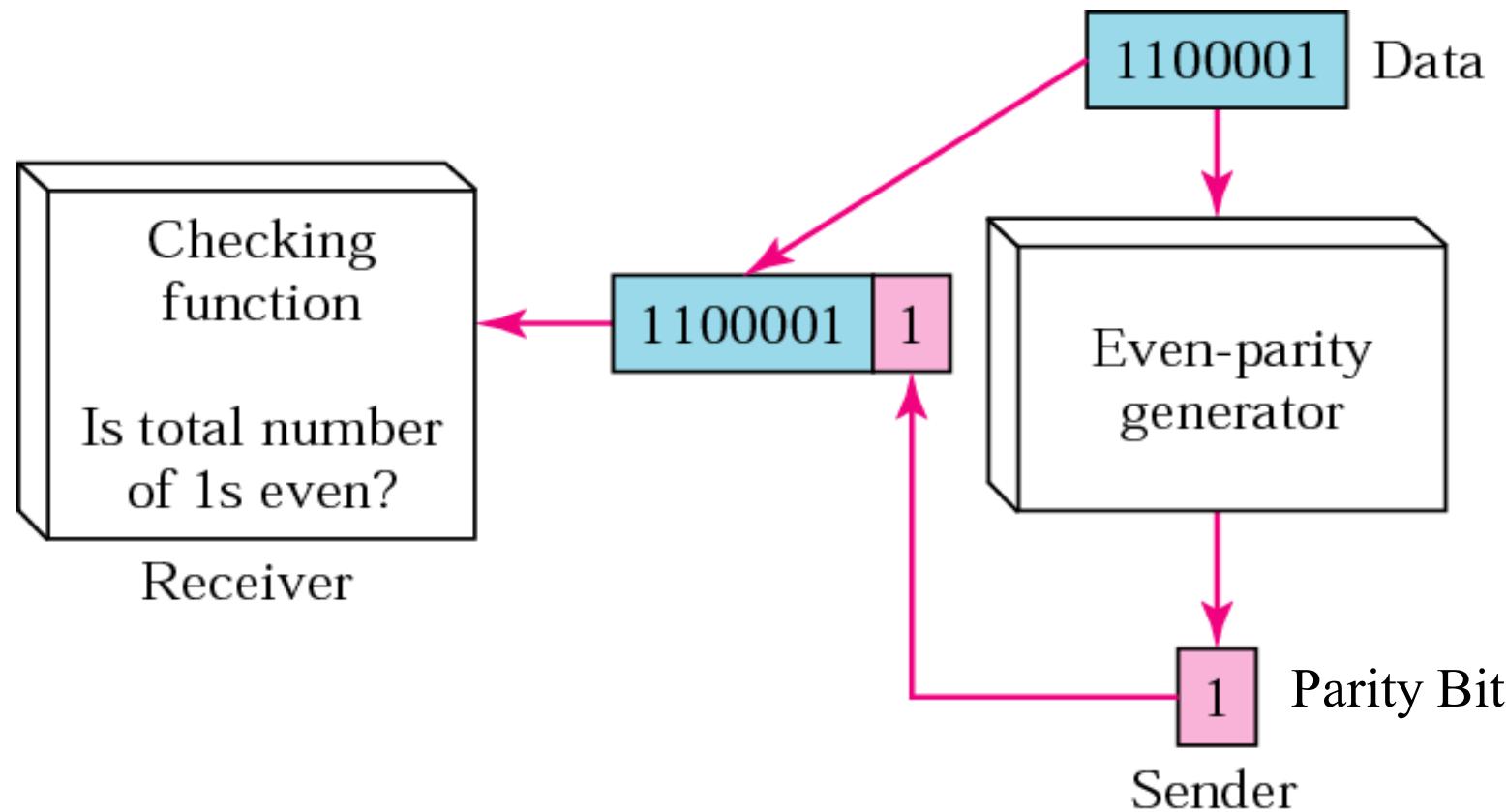
- ❑ Uses redundancy – bits are added to allow error detection in the receptor;



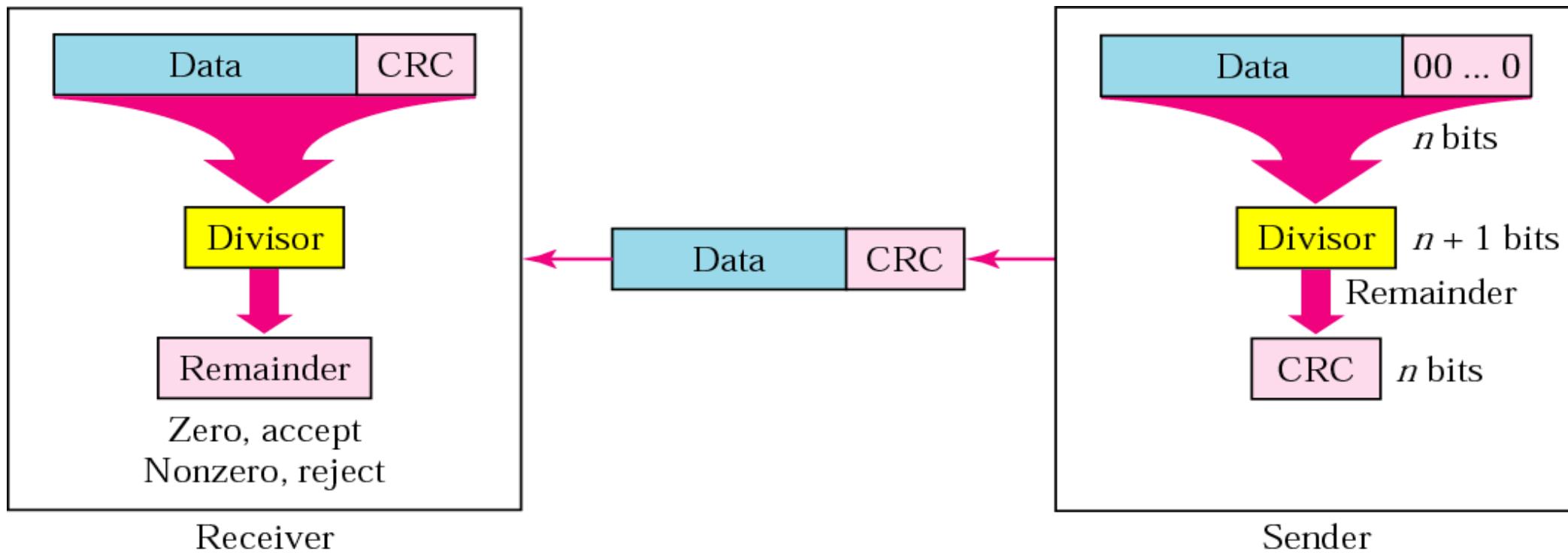
# Error Detection: Parity Bit

Does not detect an even number of bits in error;

A substantial overhead is introduced; example:  $1/8 = 12.5\%$ .



# Cyclic Redundancy Check (CRC)



Widely used in practice (e.g., Ethernet, 802.11 WiFi).

## Error Detection: CRC

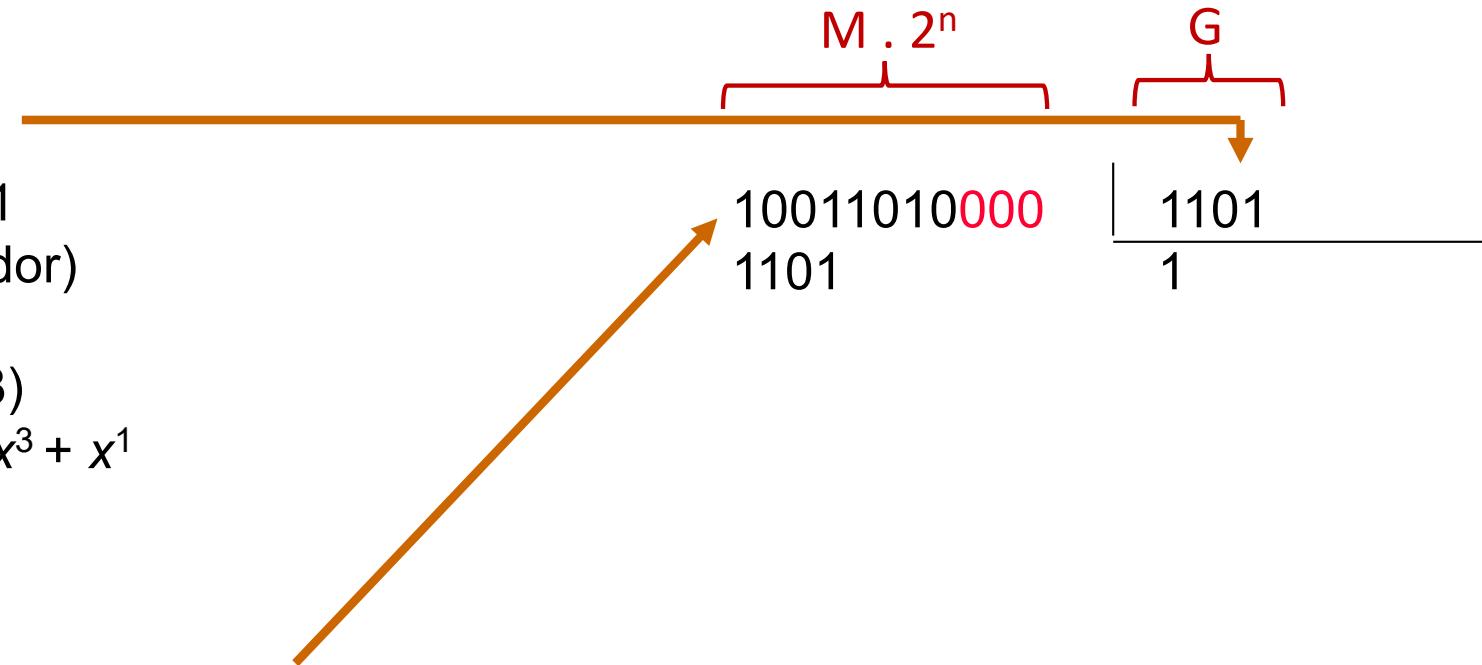
- Uses a generator polynomial  $G(x)$ , of degree  $n$ ;
- A message with  $m$  bits is viewed as a polynomial  $M(x)$  of degree lower than  $m$ ;
- CRC computation:
  - Dividend is  $x^n \cdot M(x)$ ;
  - Divisor is  $G(x)$ ;
  - Modulo 2 division (*exclusive or*) produces a remainder  $R(x)$ , of degree lower than  $n$ ;
  - The **transmitted message** is  $T(x) = x^n \cdot M(x) + R(x)$ ;
  - Note that  $T(x)$  is divisible by  $G(x)$ ;
- Error detection:
  - If the received message is divisible by  $G(x)$  then no error are detected.

## CRC: Example

1101 ( $n = 3$ )  
 $G(x) = x^3 + x^2 + 1$   
(polinómio gerador)

1001 1010 ( $m=8$ )  
 $M(x) = x^7 + x^4 + x^3 + x^1$   
(mensagem)

100 1101 0000  
 $x^3 \cdot M(x) = x^{10} + x^7 + x^6 + x^4$   
(dividendo)



## CRC: Example

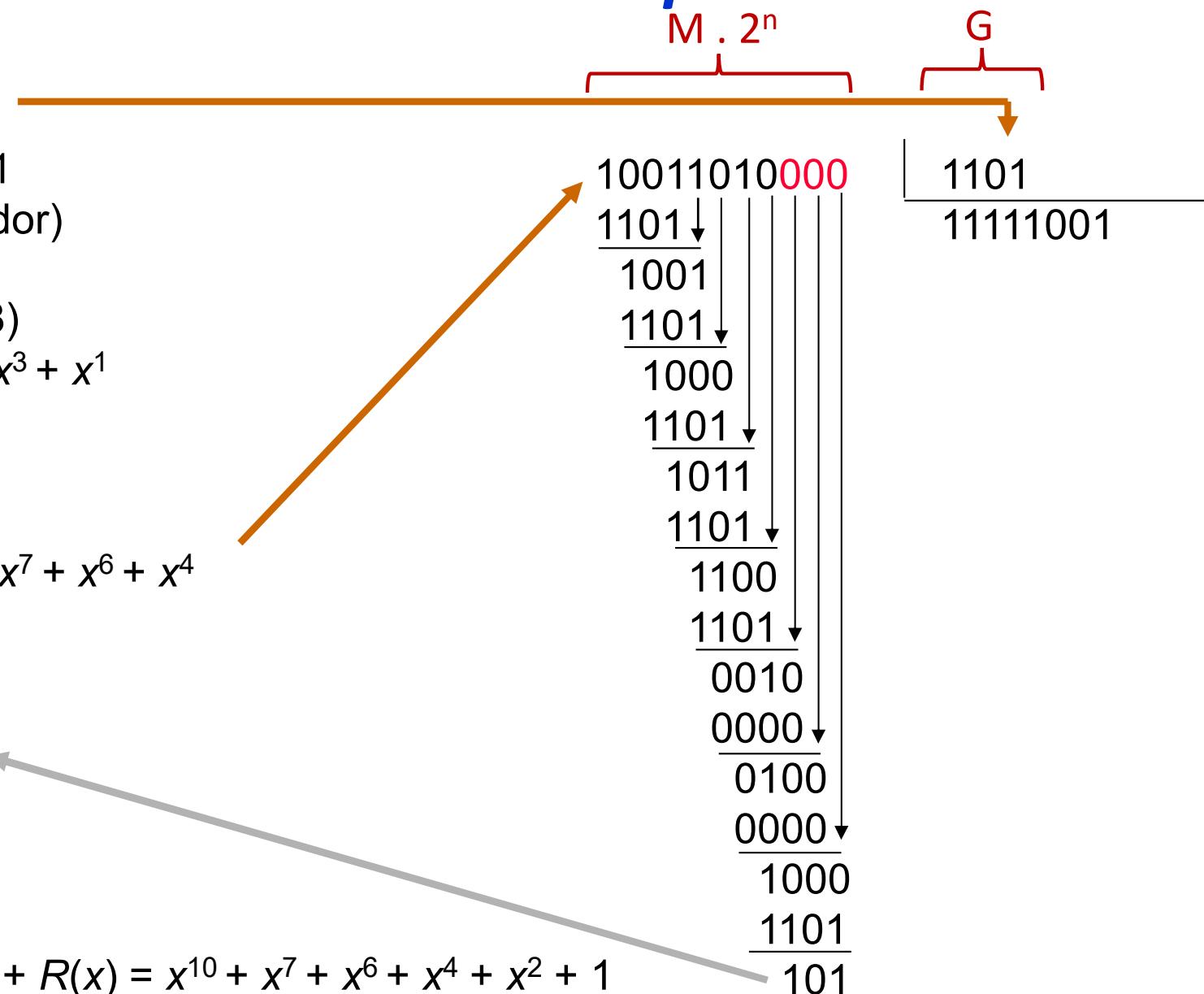
1101 ( $n = 3$ )  
 $G(x) = x^3 + x^2 + 1$   
 (polinómio gerador)

1001 1010 ( $m=8$ )  
 $M(x) = x^7 + x^4 + x^3 + x^1$   
 (mensagem)

100 1101 0000  
 $x^3 \cdot M(x) = x^{10} + x^7 + x^6 + x^4$   
 (dividendo)

101  
 $R(x) = x^2 + 1$   
 (resto)

10011010101  
 $T(x) = x^3 \cdot M(x) + R(x) = x^{10} + x^7 + x^6 + x^4 + x^2 + 1$   
 (mensagem transmitida – incluindo CRC)

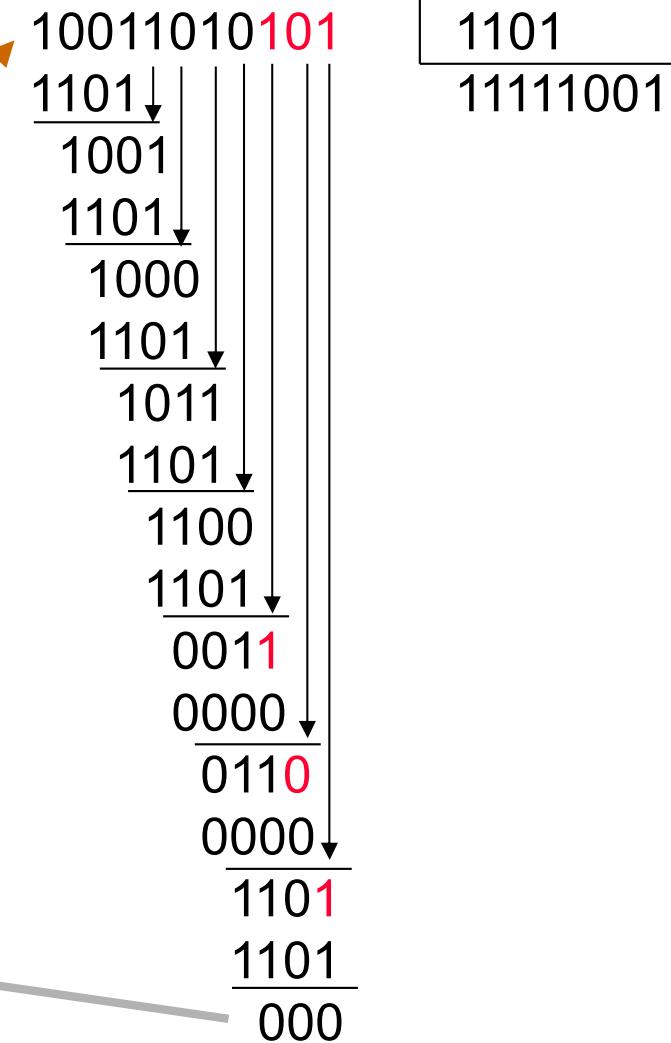


## CRC: Example (2)

1101 ( $n = 3$ )  
 $G(x) = x^3 + x^2 + 1$   
 (polinómio gerador)

1001 1010 101  
 $x^3 \cdot M(x) + R(x) = x^{10} + x^7 + x^6 + x^4 + x^2 + 1$   
 (mensagem recebida – incluindo CRC)

000  
 $R'(x) = 0$   
 (não há erros ou os erros não são detectados)



## CRC: Properties

Let  $T(x) + E(x)$ , be the received message, with  $E(x)$  being the error pattern:

- The error pattern is detected if and only if  $E(x)$  is not divisible by  $G(x)$ ;
- Single errors are detected if  $G(x)$  has more than one term;
- Odd number of errors are detected if  $G(x)$  has  $x + 1$  as a factor;
- If  $G(x)$  has degree  $n$  and includes the term  $1$  (i.e.,  $x^0$ ):
  - Burst errors of length up to  $n$  are detected;
  - Burst errors of length  $n + 1$  are detected with probability  $1 / 2^{n-1}$ .

Ethernet, WiFi:

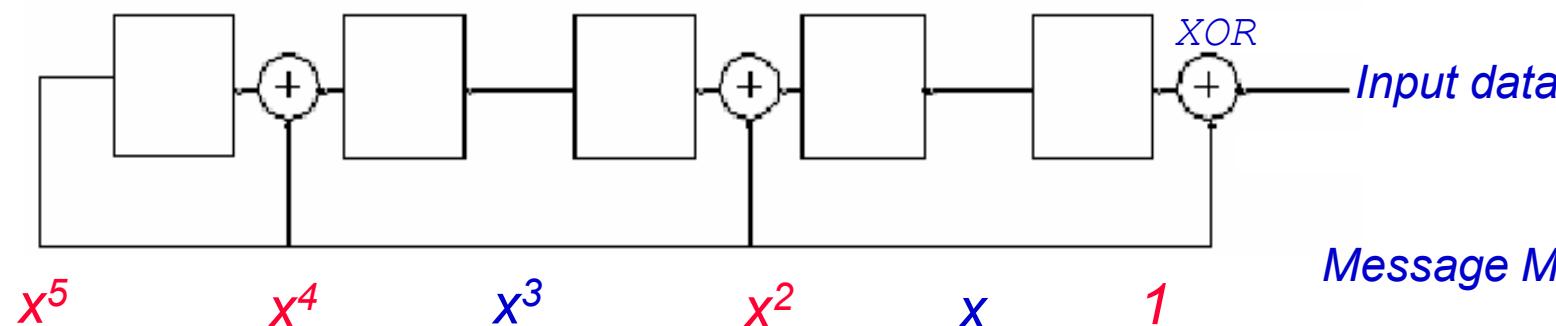
$$G_{\text{CRC-32}} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$$

$$G_{\text{CRC-32}} = 10000\ 0100\ 1100\ 0001\ 0001\ 1101\ 1011\ 0111 = 0x04C11DB7$$

# CRC: Hardware Implementation

CRC is easily implemented in hardware, by using shift registers:

$$G(x) = x^5 + x^4 + x^2 + 1$$



Message M=1010001101

*Input when calculating CRC:  
1010001101 00000*

*CRC =01110 (to be calculated)*

*(Input when calculating CRC:  
1010001101 01110)*

# Forward Error Correction

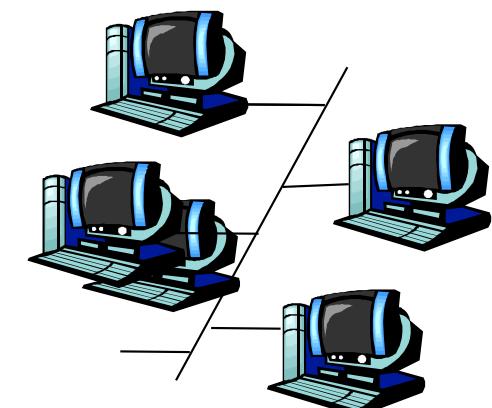
**Forward error correction** (FEC) uses codes with enough redundancy to allow the detection and correction of errors on the receptor, without requiring the message retransmission.

## Examples:

- Hamming code: detects and corrects isolated bit errors.
- There are more sophisticated techniques often used, such as Reed-Solomon codes.
- FEC is often used in environments with long propagation delays (example: satellite transmissions).
- There are hardware implementations of FEC techniques (example: V.34 modem).

# Outline

- Introduction and services
- Link-layer Addressing
- Error detection and correction
- Multiple access protocols
- Ethernet
- Link-layer switches
- IEEE 802.11 Wireless LANs
- Framing



# Multiple Access Links and Protocols

Two types of “links”:

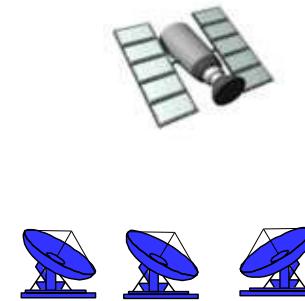
- Point-to-point:
  - PPP for dial-up access;
  - Point-to-point link between Ethernet switch and host.
- Broadcast (shared wire or medium):
  - Old-fashioned Ethernet;
  - Upstream HFC in cable networks;
  - 802.11 wireless LAN, 4G, 5G, satellite.



Shared wire (e.g.,  
cabled Ethernet)



Shared RF  
(e.g., 802.11 WiFi)



Shared RF  
(satellite)



Humans at a  
cocktail party  
(shared air, acoustical)

# Multiple Access Control (MAC) Protocols

- Single shared broadcast channel;
- Two or more simultaneous transmissions by nodes – interference:
  - Collision if node receives two or more signals at the same time.

## Multiple access control protocol:

- Distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit;
- Communication about channel sharing must use the channel itself!
  - No out-of-band channel for coordination.

# *Ideal Multiple Access Protocol*

## Broadcast channel of rate R bps

1. When one node wants to transmit, it can send at rate R.
2. When M nodes want to transmit, each can send at average rate  $R/M$ .
3. Fully decentralized:
  - No special node to coordinate transmissions;
  - No synchronization of clocks, slots.
4. Simple.

# MAC Protocols: a Taxonomy

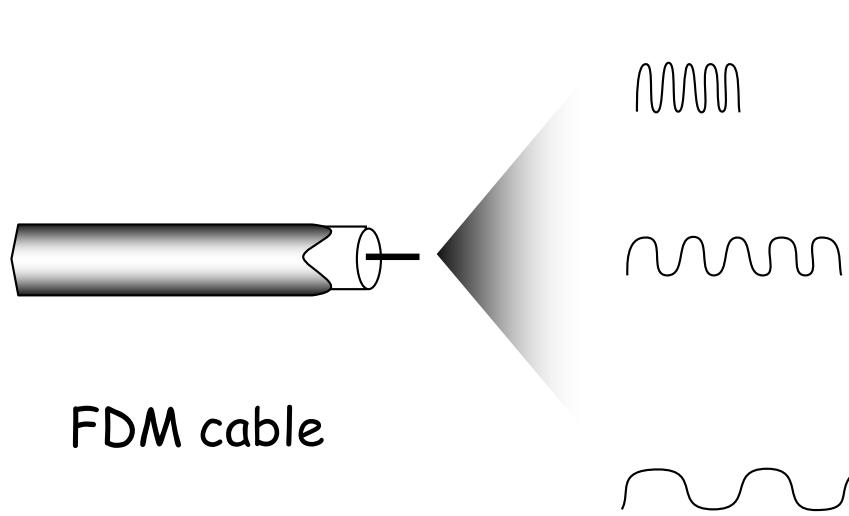
Three broad classes:

- Fixed Channel Partitioning
  - Divide channel into smaller “pieces” (time slots, frequencies, codes);
  - Allocate piece to node for exclusive use.
- Dynamic Allocation (“Taking turns”)
  - Nodes take turns  
(but, nodes with more to send can take longer turns);
  - Poll/Select; Token passing.
- Random Access
  - Channel not divided, allow collisions;
  - “Recover” from collisions.

# Fixed Channel Partitioning: FDMA

## FDMA – Frequency Division Multiple Access:

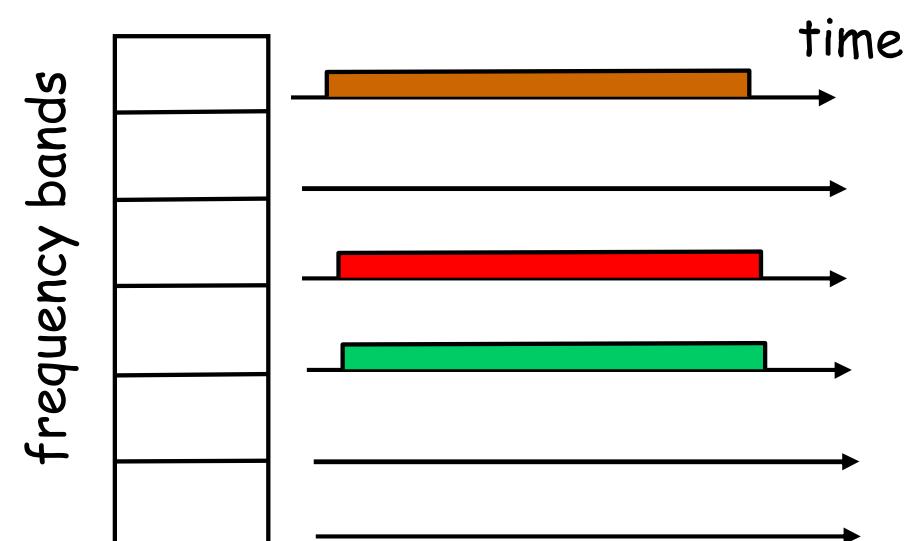
- Channel spectrum divided into frequency bands;
- Each station assigned a fixed frequency band;
- Unused transmission time in frequency bands go idle;



Example of 6-station LAN:

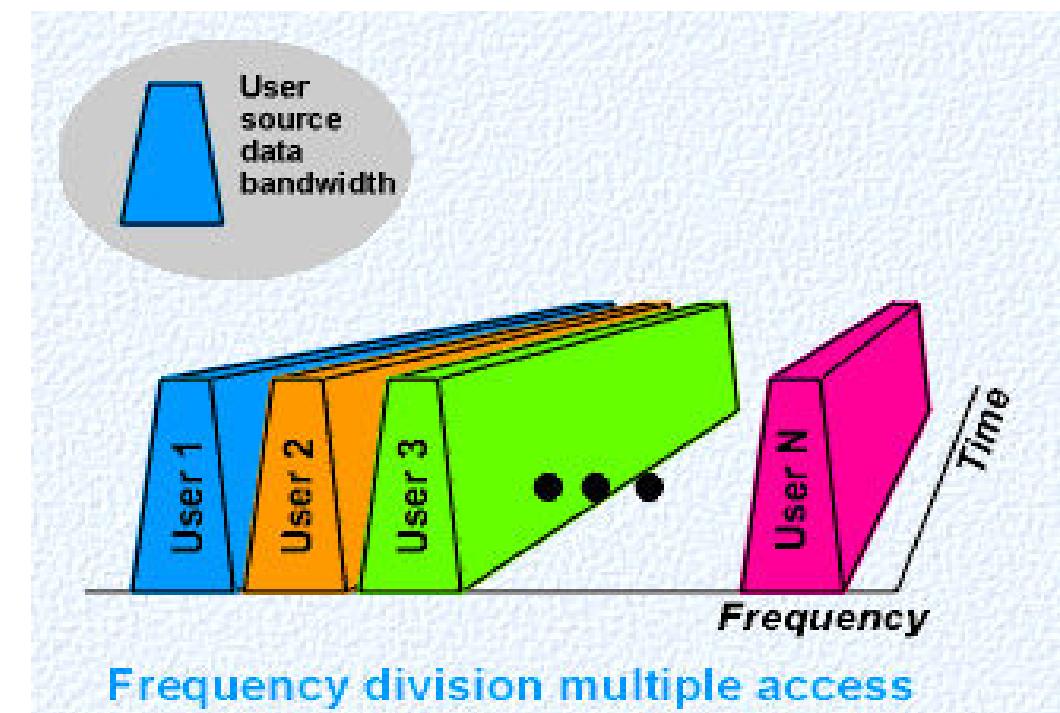
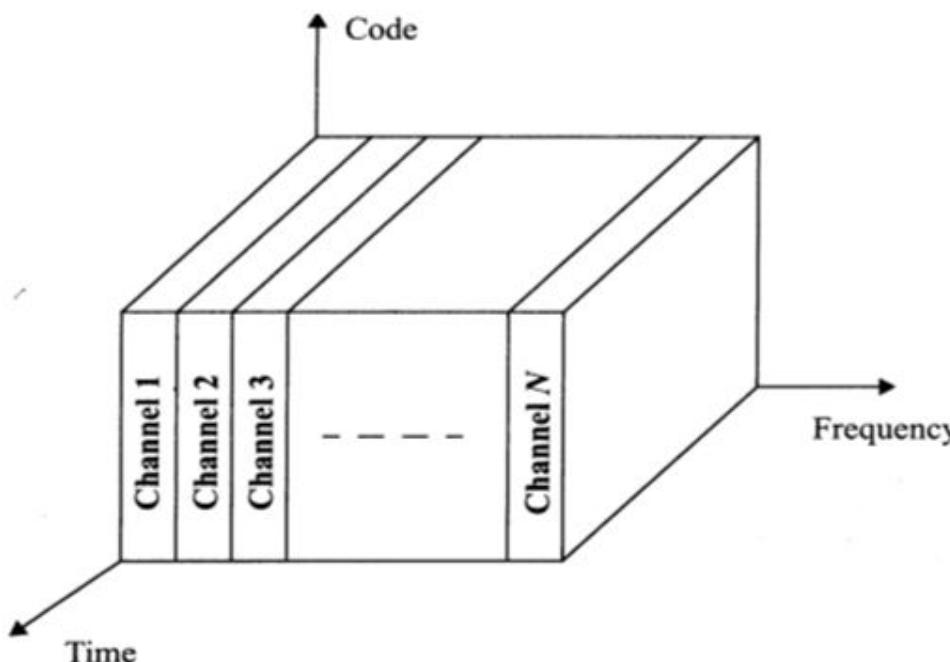
1,3,4 have packets;

Frequency bands 2,5,6 are idle.



# FDMA

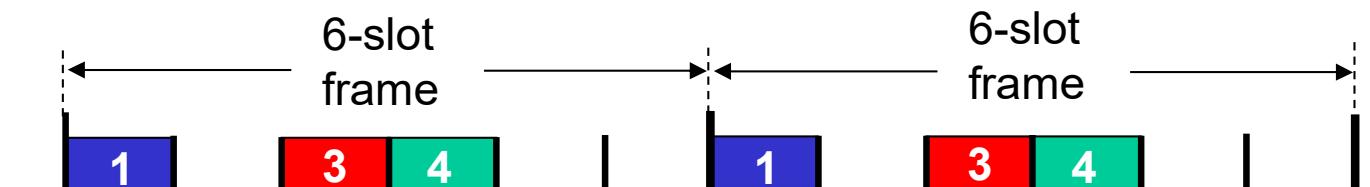
## FDMA – Frequency Division Multiple Access



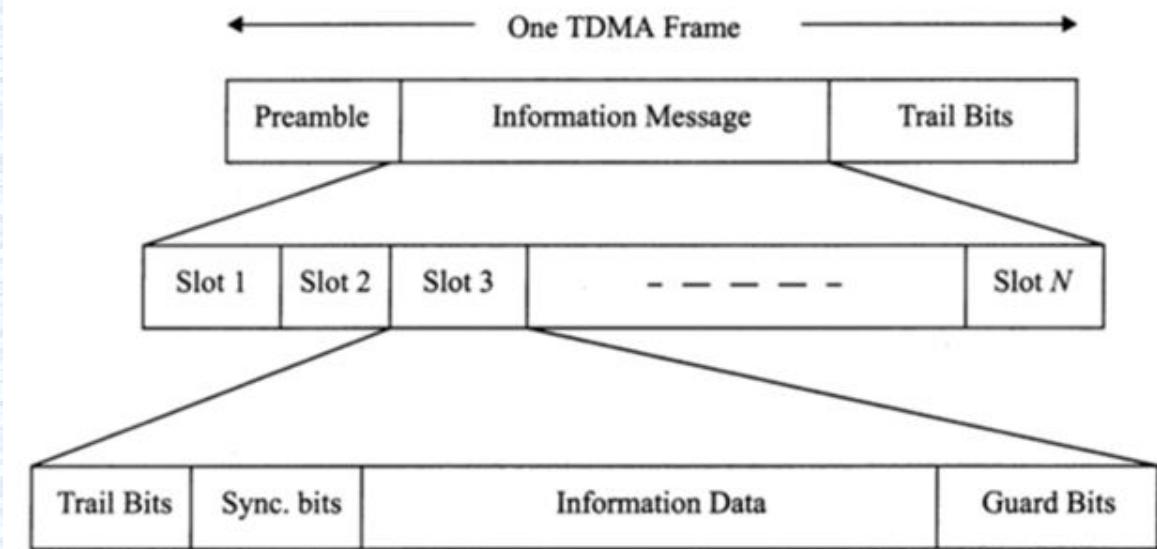
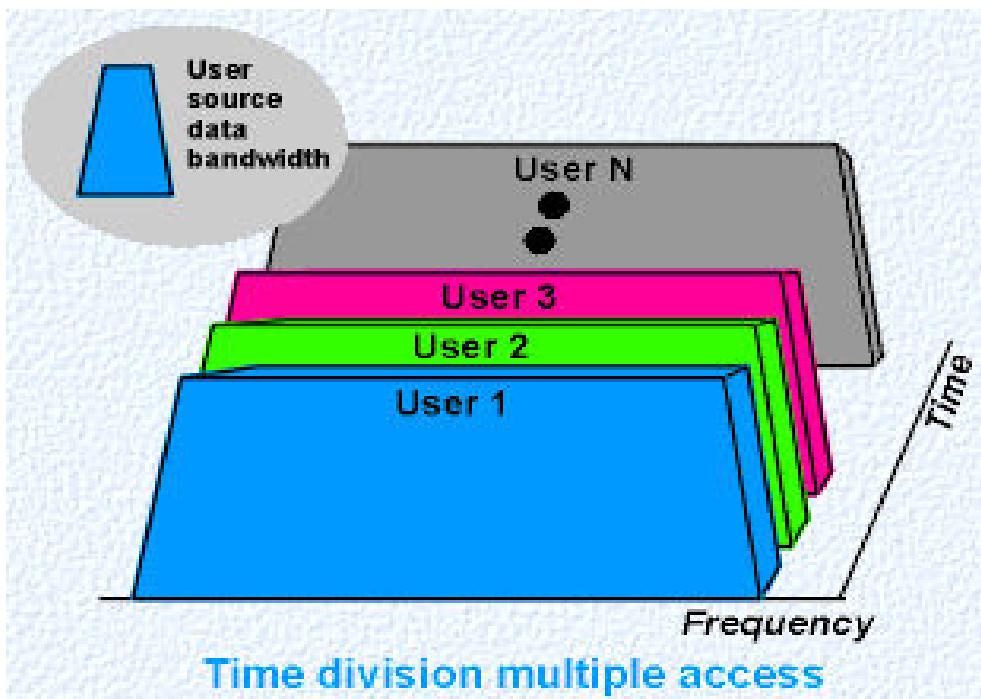
# Fixed Channel Partitioning: TDMA

## TDMA – Time Division Multiple Access:

- Access to channel in "rounds";
- Each station gets fixed length slot (length = packet transmission time) in each round;
- Unused slots go idle.



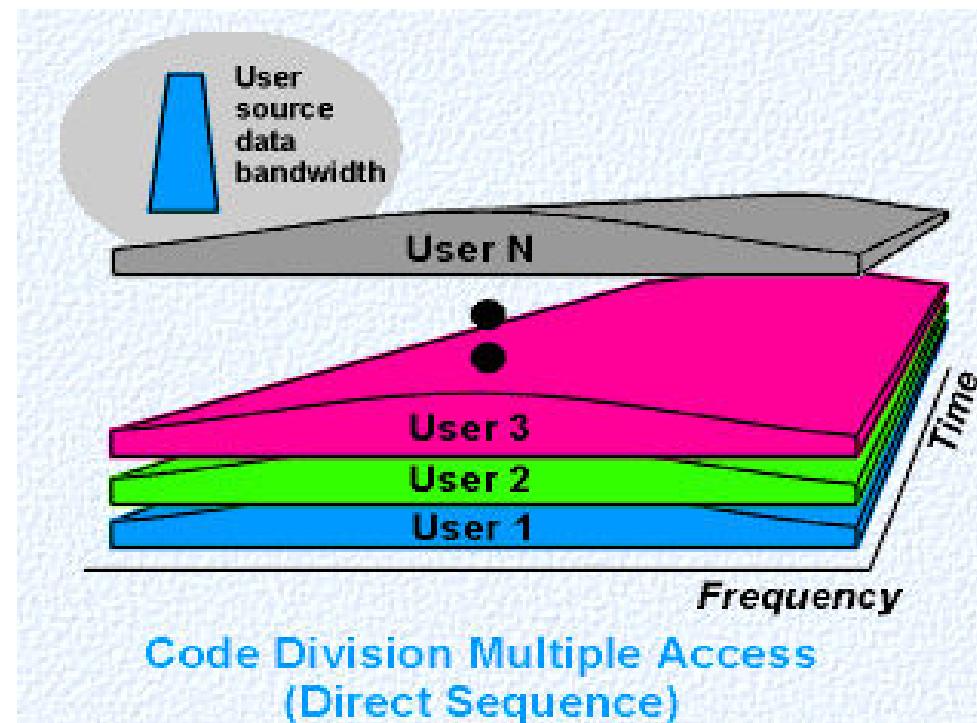
## TDMA

TDMA – *Time Division Multiple Access*

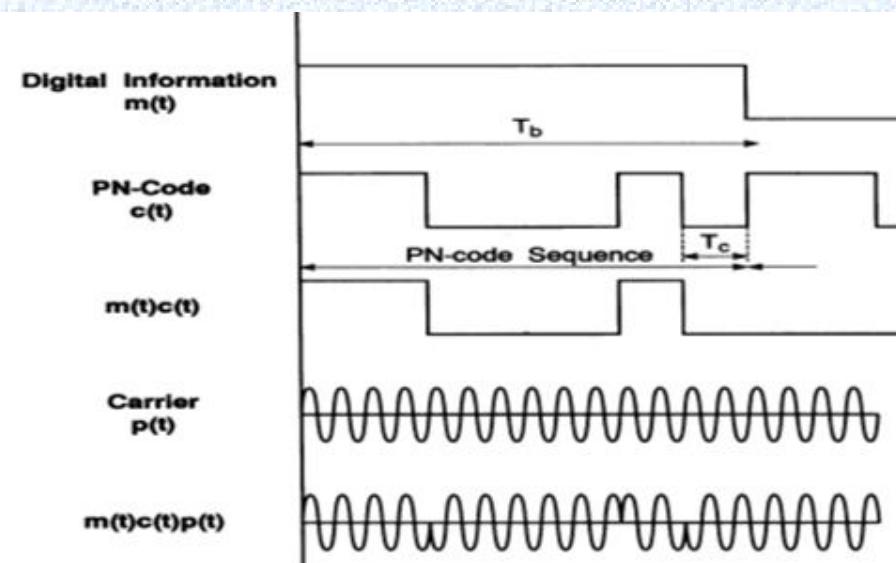
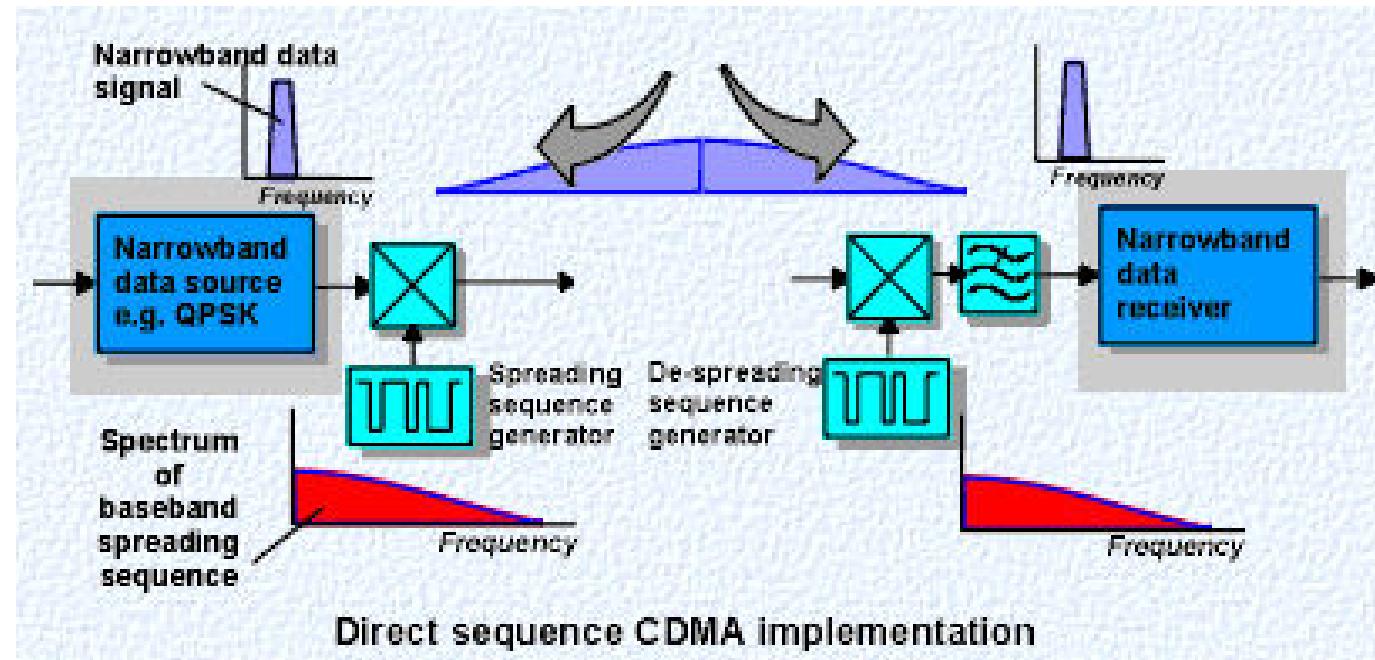
# Fixed Channel Partitioning: CDMA

## CDMA – *Code Division Multiple Access*

- Each user has access to the complete frequency band, during all the time.
- Users are distinguished by using different codes.



# CDMA



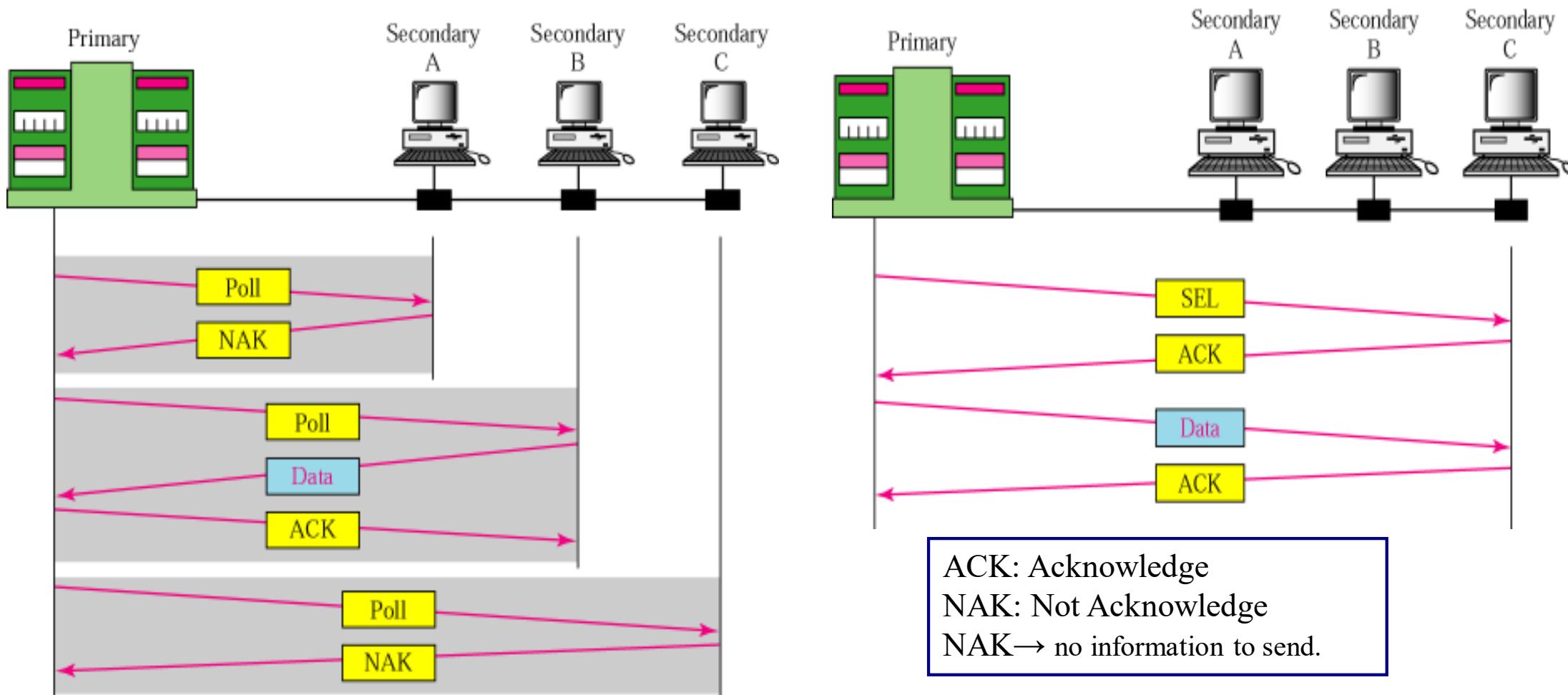
# Dynamic Allocation Protocols

Three broad classes:

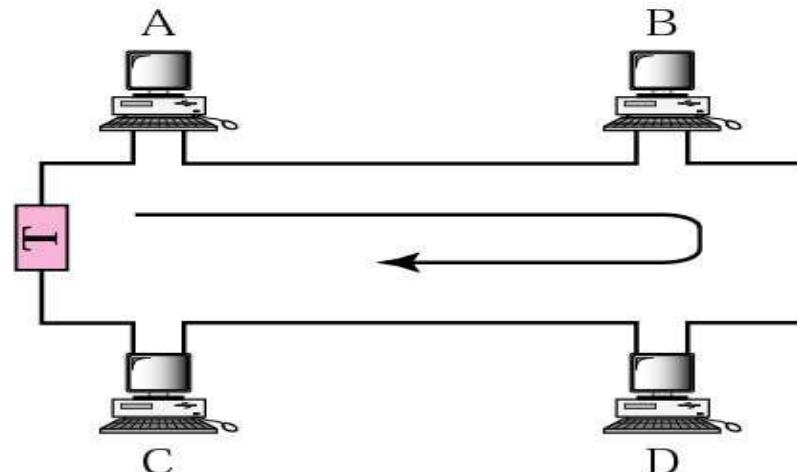
- Fixed Channel Partitioning
  - Divide channel into smaller “pieces” (time slots, frequencies, codes);
  - Allocate piece to node for exclusive use.
- Dynamic Allocation (“Taking turns”)
  - Nodes take turns  
(but, nodes with more data to send can take transmit more often);
  - Poll/Select; Token passing.
- Random Access
  - Channel not divided; collisions may occur;
  - “Need to recover” from collisions.

# Dynamic Allocation: Poll / Select

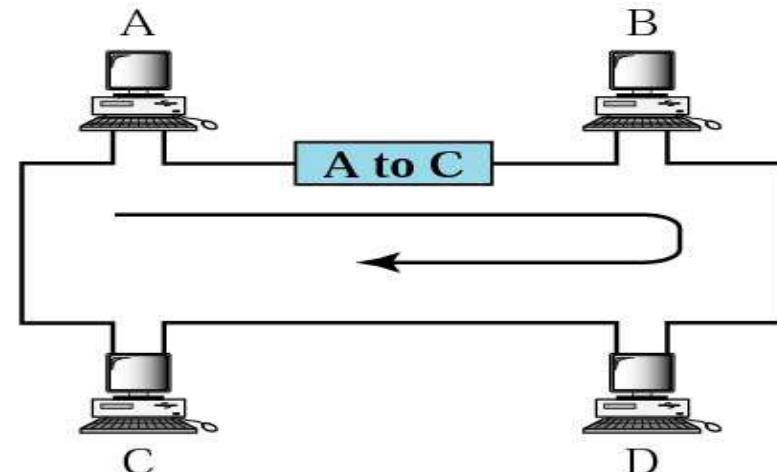
A central (*primary*) computer controls the activity of the others.



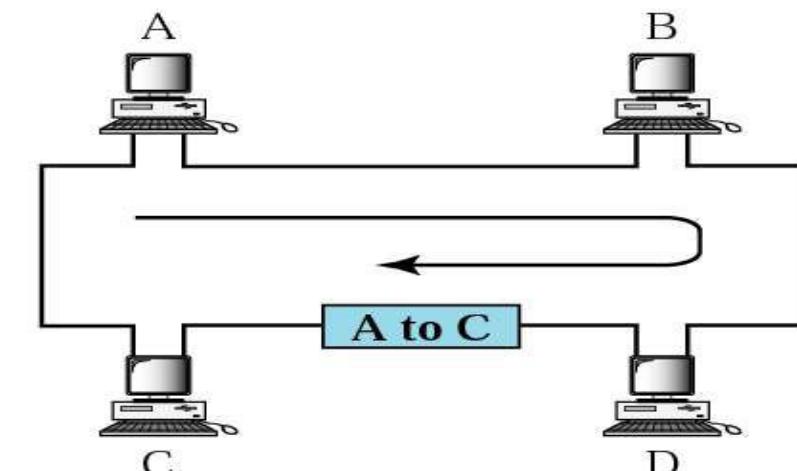
# Dynamic Allocation: Token Passing



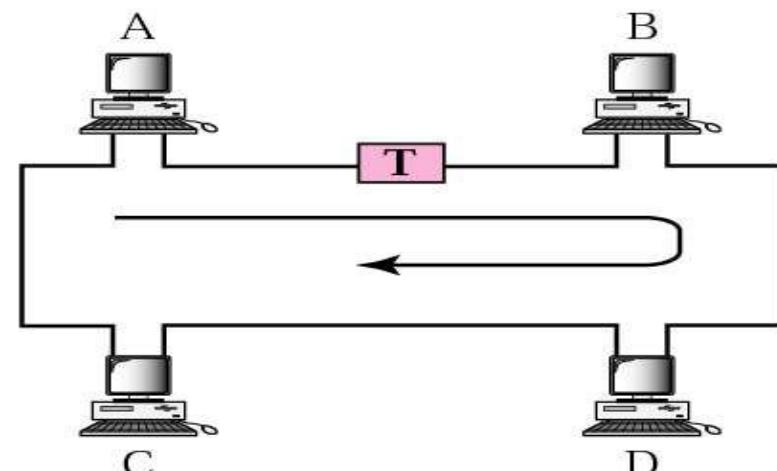
a. Station A captures the token



b. Station A sends data to station C



c. Station C copies data and sends frame back to A



d. Station A releases the token

# *Random Access Protocols*

Three broad classes:

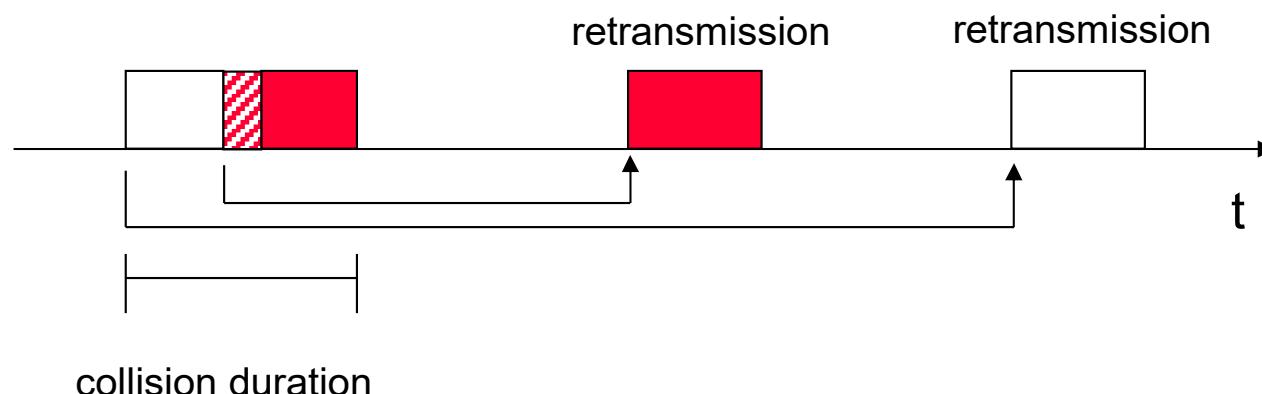
- **Fixed Channel Partitioning**
  - Divide channel into smaller “pieces” (time slots, frequencies, codes);
  - Allocate piece to node for exclusive use.
- **Dynamic Allocation (“Taking turns”)**
  - Nodes take turns  
(but, nodes with more to send can take longer turns);
  - Poll/Select; Token passing.
- **Random Access**
  - Channel not divided, allow collisions;
  - “Recover” from collisions.

# Random Access Protocols

- When node has packet to send:
  - Transmit at full channel data rate R;
  - No *a priori* coordination among nodes.
- Two or more transmitting nodes → “collision”.
- Random access MAC protocol specifies:
  - How to detect collisions;
  - How to recover from collisions (e.g., via delayed retransmissions).
- Examples of random access MAC protocols:
  - ALOHA;
  - Slotted ALOHA;
  - CSMA, CSMA/CD, CSMA/CA.

# ALOHA

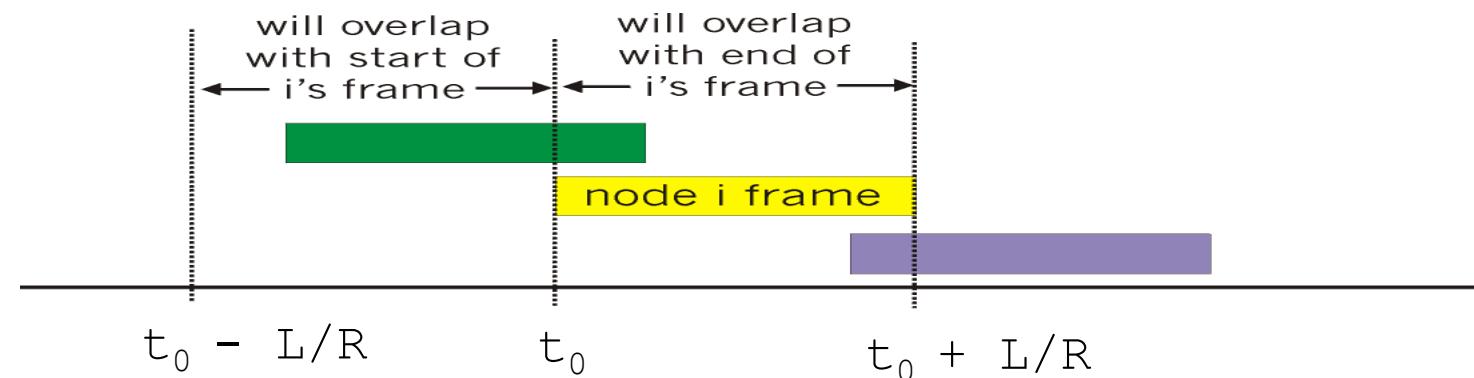
- Created at the University of Hawaii in 1970;
- Aloha → simple, **no synchronization**;
- **Hosts transmit at channel rate**;
- When frame is ready: transmit immediately;
- **Collision** if two or more frames overlap (frames are lost);
- If frame is lost, schedule **retransmission** for a future instant (randomly chosen).



# ALOHA Efficiency

## Collisions:

- Frame sent at  $t_0$  collides with any other frames sent in  $[t_0-L/R, t_0+L/R]$ .



$$P(\text{success by given node}) = P(\text{node transmits}) \cdot P(\text{no other transmission in } [t_0-L/R, t_0+L/R])$$

*Vulnerability period*

$$P\{\text{success}\} = P(k=0) = e^{-2G}$$

$$G = g \cdot L/R$$

Poisson process:  $P\{k \text{ arrivals}\} = \frac{(g \cdot t)^k}{k!} \cdot e^{-g \cdot t}$

# Slotted ALOHA

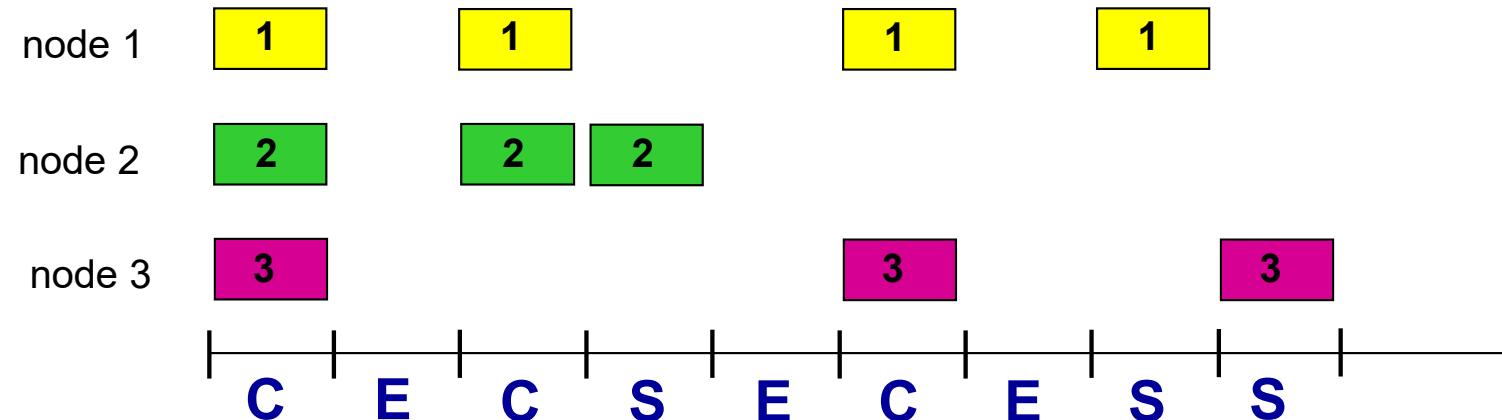
## Assumptions:

- All frames of same size  $L$ ;
- Time is divided into equal size slots (time to transmit 1 frame –  $L/R$ );
- Nodes start to transmit only at slot beginning;
- Nodes are synchronized;
- If 2 or more nodes transmit in a slot, all nodes detect the collision.

## Operation:

- When node obtains fresh frame, transmits in next slot:
  - *If no collision:* node can send new frame in next slot;
  - *If collision:* node retransmits frame in each subsequent slot with probability  $p$  until success.

## Slotted ALOHA



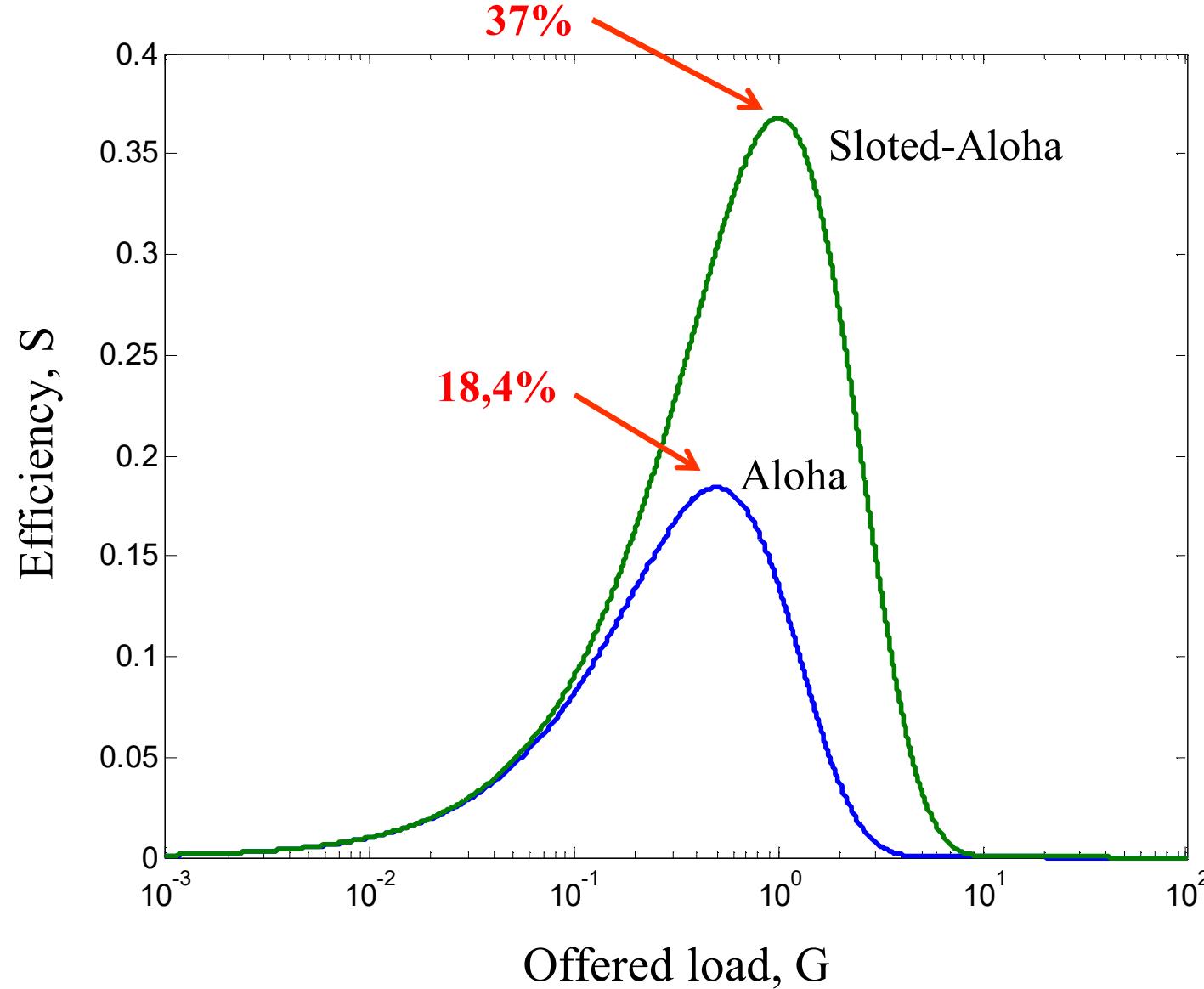
### Pros:

- Single active node can continuously transmit at full channel rate;
- Highly decentralized: only slot durations need to be in sync in every node;
- Simple.

### Cons:

- Collisions, wasting slots;
- Idle slots;
- Nodes may be able to detect collision in less than time needed to transmit packet;
- Clock synchronization.

## Aloha Efficiency



# *Carrier Sense Multiple Access (CSMA)*

## **CSMA:**

- Listen before transmit:
  - If channel sensed idle → transmit entire frame;
  - If channel sensed busy → defer transmission.
- Human analogy: don't interrupt others!

# CSMA Collisions

spatial layout of nodes



## Collisions can still occur:

- Propagation delay means two nodes may not hear each other's transmission;

### Collision:

- Entire packet transmission time wasted;

### Note:

- Role of distance & propagation delay in determining collision probability.

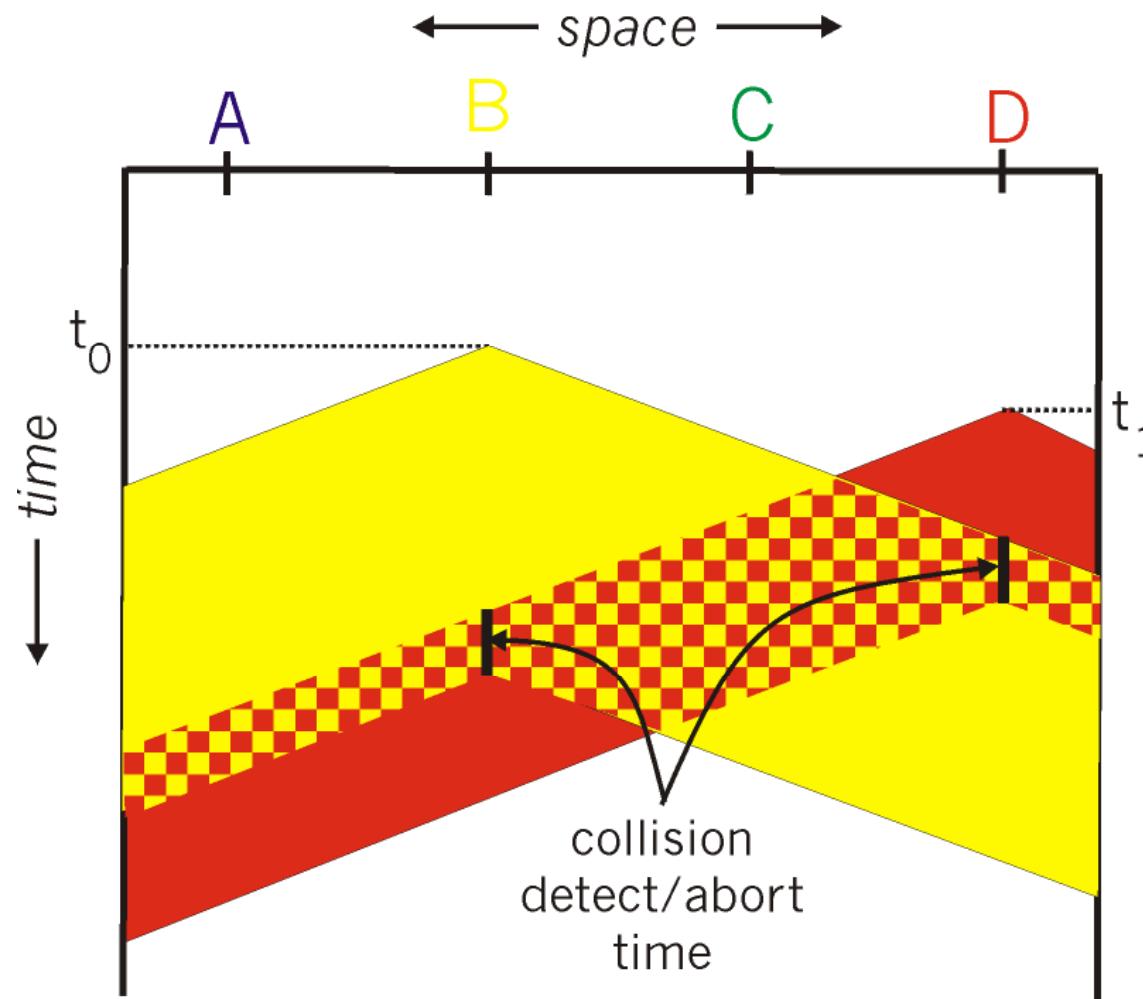


# CSMA/CD (*Collision Detection*)

## CSMA/CD:

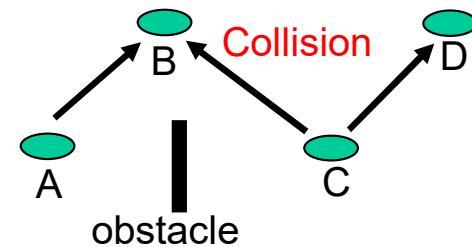
- Carrier sensing, deferral as in CSMA:
  - **Collisions detected** within short time;
  - Colliding transmissions aborted, reducing channel useless occupation;
- Collision detection:
  - Easy in wired LANs: measure signal strengths, compare transmitted, received signals;
  - Difficult in wireless LANs: received signal strength overwhelmed by local transmission strength ;
- Human analogy: the polite conversationalist ☺

## CSMA/CD Collision Detection



# Carrier Sense Multiple Access (CSMA)

- Listen to the channel before transmission:
  - If channel is busy, postpone frame transmission;
  - If channel is free, immediately start frame transmission;
- Persistent CSMA:
  - When channel is busy, the station transmits as soon as it becomes idle;
- Non-persistent CSMA:
  - When channel is busy, the station schedules the frame transmission for a future moment, randomly chosen;
- CSMA/CD:
  - Stations involved in a collision stop their transmission as soon as the collision is detected;
- CSMA/CA:
  - There may be “hidden” stations...



# *Summary of MAC Protocols*

*Fixed channel partitioning*, by time, frequency or code:

- Frequency Division, Time Division, Code Division;

*Dynamic allocation (“taking turns”):*

- Polling from central site, token passing;
- Bluetooth, FDDI, IBM Token Ring.

*Random access (dynamic):*

- ALOHA, S-ALOHA, CSMA, CSMA/CD;
- Carrier sensing: easy in some technologies (wire), hard in others (wireless);
- CSMA/CD used in Ethernet (IEEE 802.3);
- CSMA/CA used in Wi-Fi (IEEE 802.11).

## *Summary of MAC Protocols*

Fixed channel partitioning MAC protocols:

- Efficient with steady high load from all nodes;
- Inefficient at low or unbalanced loads:
  - $1/N$  bandwidth allocated even if only 1 active node!

Dynamic allocation (“taking turns”) protocols:

- Share channel *efficiently* and *fairly* at high loads;
- Low load: inefficient as active nodes have to wait for idle nodes to “pass their turn”;

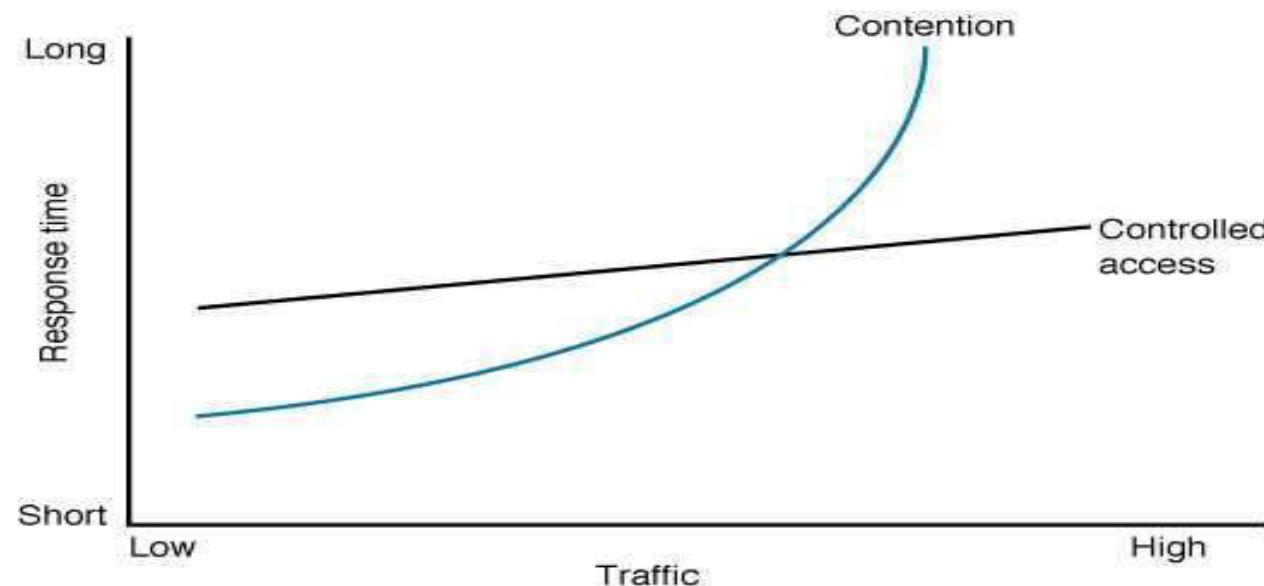
Random access MAC protocols:

- Efficient at low load: single node can fully utilize channel;
- High load: collision overhead.

# MAC Protocols

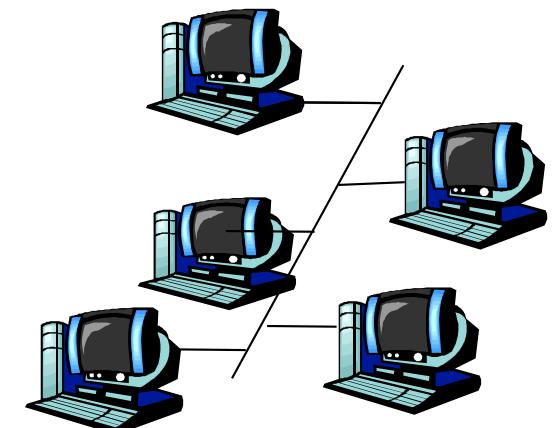
Comparing dynamic allocation with random access protocols:

- Dynamic allocation protocols allow a better channel usage at high loads;
- Random access protocols impose lower delays at low loads;
- Dynamic allocation protocols require central management or token management;
- Random access protocols need to be controlled against unstable behavior.



# *Outline*

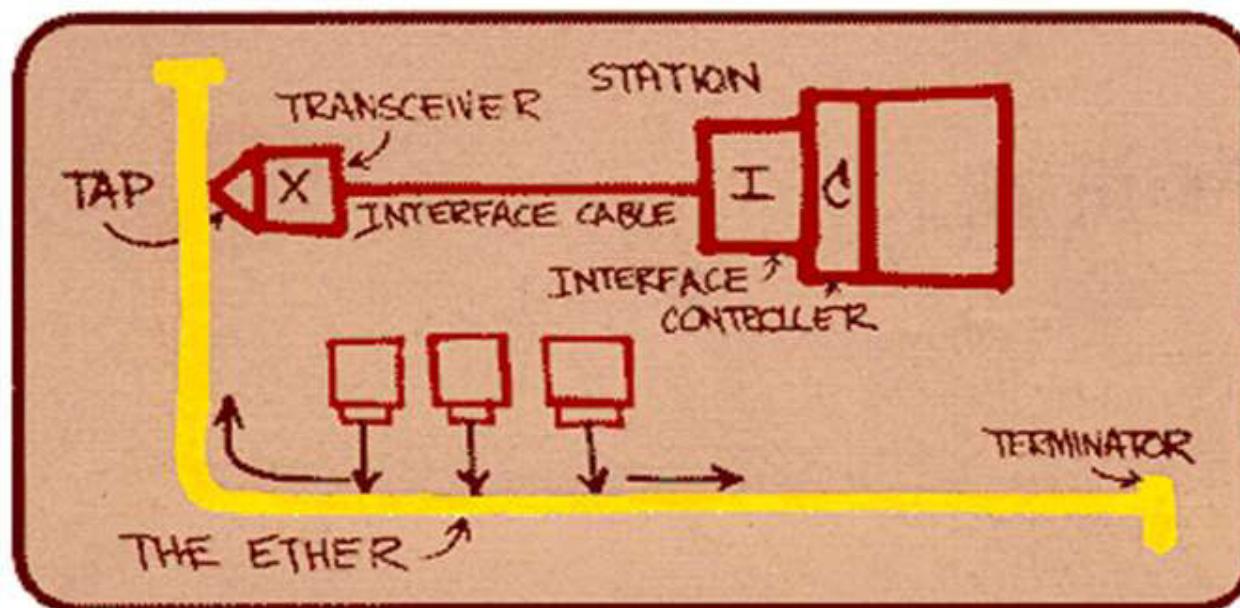
- Introduction and services
- Link-layer Addressing
- Error detection and correction
- Multiple access protocols
- Ethernet
- Link-layer switches
- IEEE 802.11 Wireless LANs
- Framing



# IEEE 802.3 – Ethernet

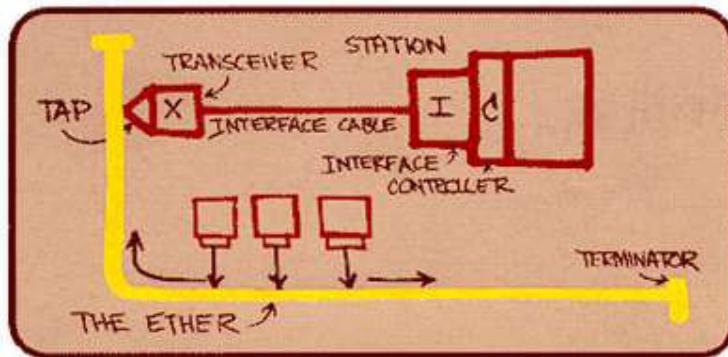
“Dominant” wired LAN technology:

- Cheap: < \$10 for network interface card (NIC);
- First widely used LAN technology;
- Simpler, cheaper than token LANs and ATM;
- Kept up with speed race: 10 Mbps ... 100 Gbps, 400Gbps, ...



Metcalfe's Ethernet sketch

# IEEE 802.3 – Ethernet



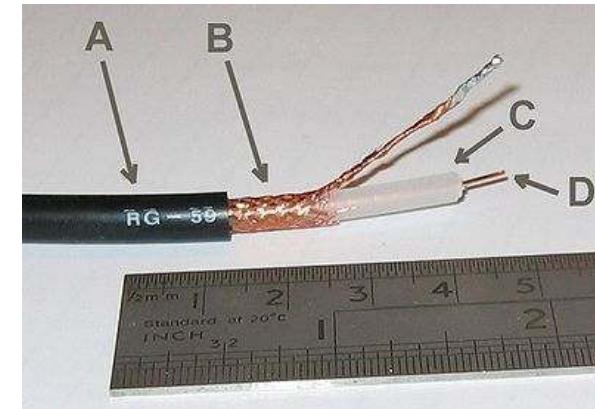
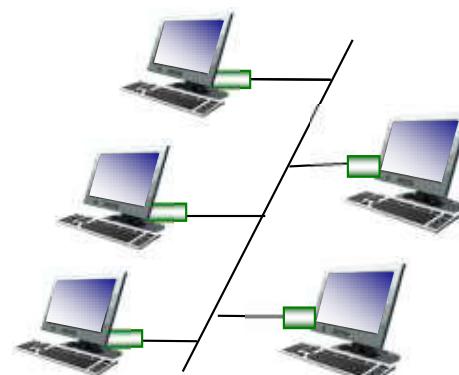
Dismantled vampire tap (10BASE5):

- Central metal-tipped insulated spike contacts cable core;
- smaller spikes contact cable shield.

Note black mark on cable sheath indicating suitable location for transceiver.

# Bus Topology

- Bus topology popular through mid 1990s:
  - All nodes in same collision domain (can collide with each other);

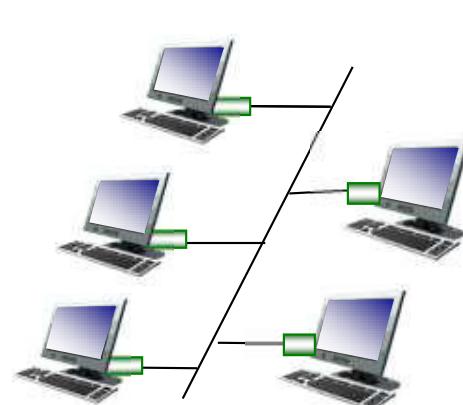


**bus:** coaxial cable

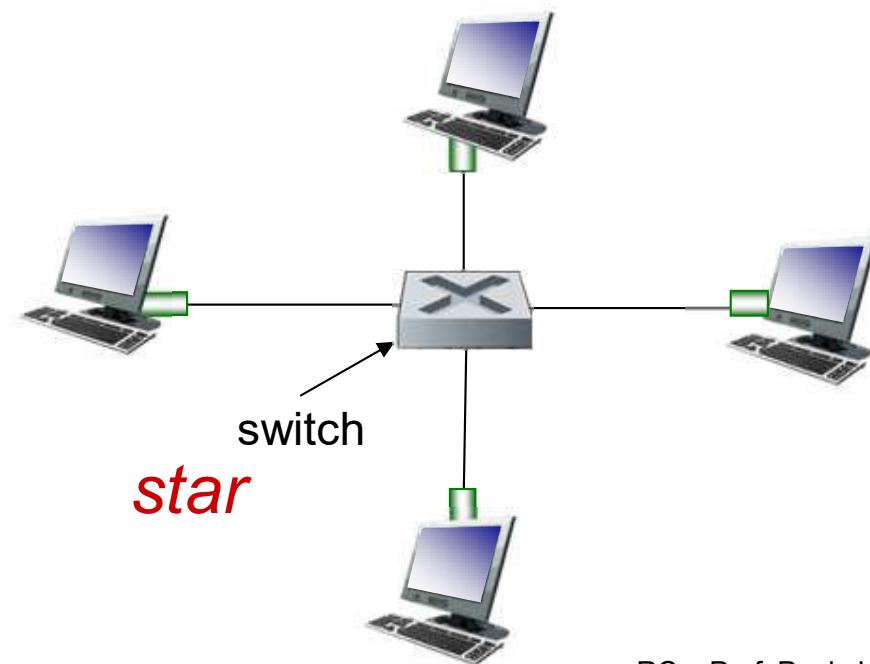
# Star Topology



- Bus topology popular through mid 90s:
  - All nodes in same collision domain (can collide with each other);
- Today: star topology prevails:
  - Active *switch* in center;
  - Each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other).

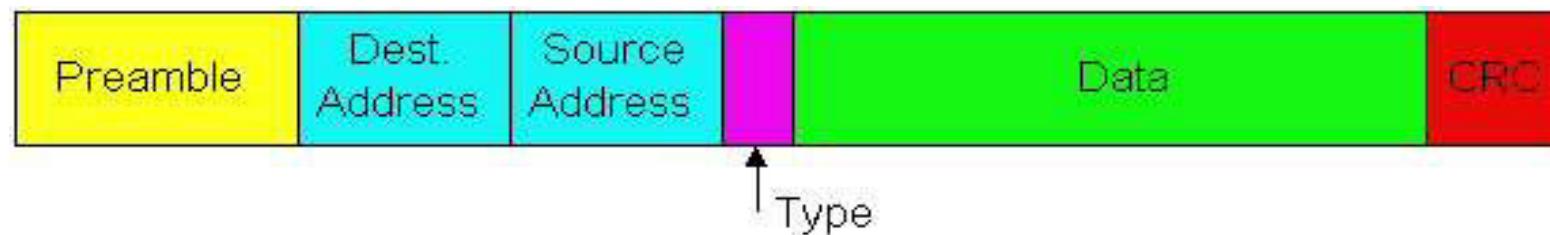


*bus*: coaxial cable



# Ethernet Frame Structure

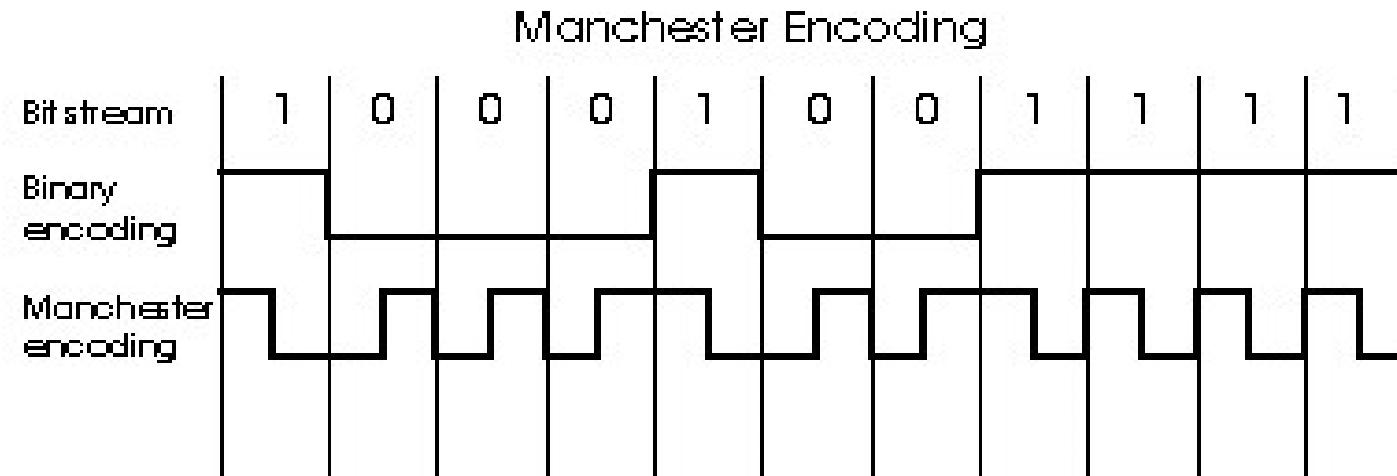
Sending adapter encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**:



## Preamble:

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011;
- Used to synchronize receiver and sender clock rates.

# Manchester Encoding

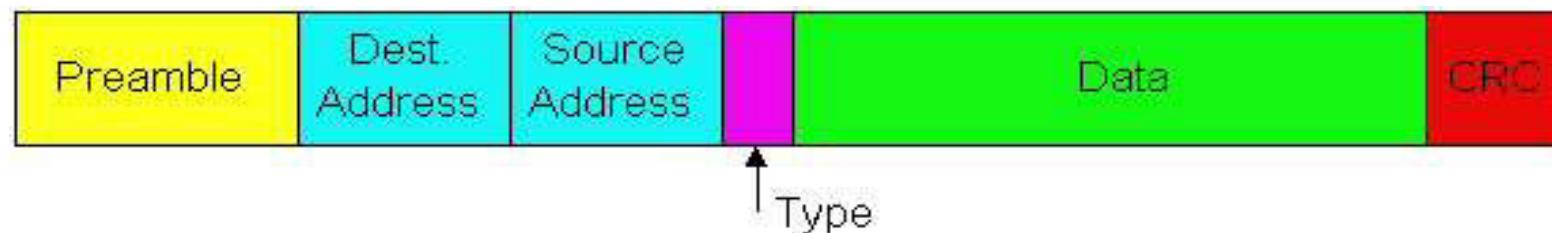


- Manchester encoding is used in 10BaseT;
- Each bit has a transition;
- Allows clocks in sending and receiving nodes to synchronize to each other:
  - No need for a centralized, global clock among nodes!

(This is physical-layer material)

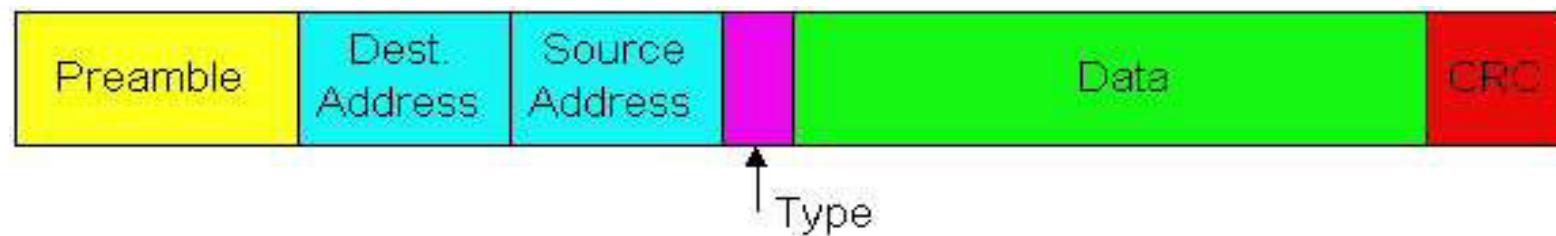
# Ethernet Frame Structure

- Addresses: 6 bytes
  - If adapter receives frame with matching destination MAC address (or broadcast address – e.g. ARP packet), it passes data in frame to the network layer;
  - Otherwise, adapter discards frame.
- Type: indicates higher layer protocol:
  - Mostly IP but others possible, e.g., Novell IPX, AppleTalk;



# Ethernet Frame Structure

- Data:
  - MTU: 46 to 1500 bytes.
- CRC:
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
  - Checked at receiver – if error is detected the frame is dropped.



# *Ethernet: Unreliable, Connectionless*

- **Connectionless:**  
No handshaking between sending and receiving NICs.
- **Unreliable:**  
Receiving NIC doesn't send ACKs or NACKs to sending NIC:
  - Stream of datagrams passed to network layer can have gaps (missing datagrams);
  - Gaps will be filled in if application is using TCP;
  - Otherwise, application will see the gaps.
- **Ethernet's MAC protocol:**  
Unslotted **CSMA/CD** with exponential binary backoff.

# Ethernet CSMA/CD Algorithm

1. NIC receives datagram from network layer → creates frame;
2. If **channel idle** (96 bit times), starts frame transmission;  
If **channel busy**, waits until channel idle, then transmits
3. If NIC transmits entire frame without detecting another transmission:  
→ success;
4. If NIC detects another transmission while transmitting - **collision**:  
→ aborts and sends jam signal (reinforce collision);
5. After aborting, NIC enters **exponential backoff**:  
→ After  $m^{\text{th}}$  collision, NIC chooses  $K$  at random from  $\{0,1,2,\dots,2^m-1\}$ ;  
→ NIC waits  $K \times 512$  bit times; then returns to Step 2.

## Jam Signal:

Make sure all other transmitters are aware of collision → 48 bits;

## Bit time:

1 ns for 1 Gbps Ethernet;

## Exponential Backoff:

*Goal:* adapt retransmission attempts to estimated load:

- Heavy load: random wait will be longer.

1<sup>st</sup> collision: choose K from {0,1}; delay is K x 512 bit transmission times;

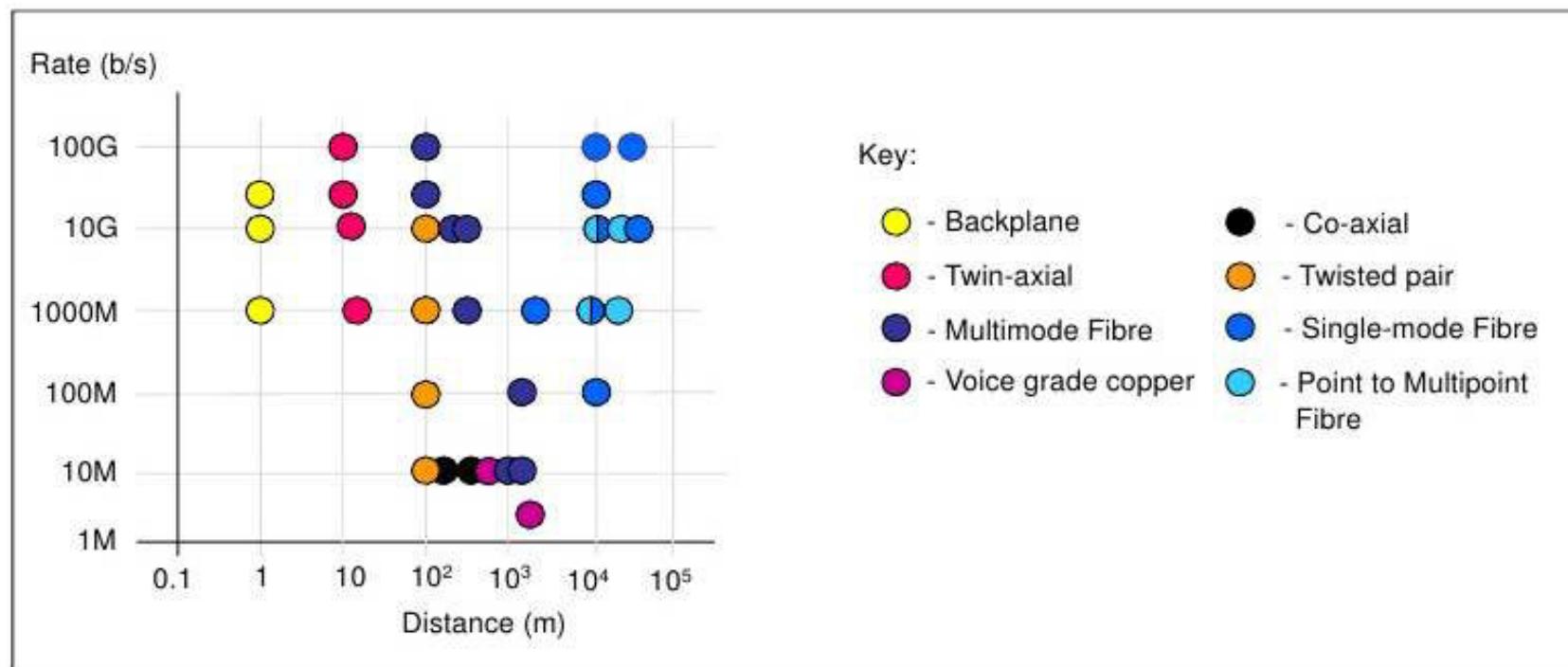
After 2<sup>nd</sup> collision: choose K from {0,1,2,3}...

After 10 collisions: choose K from {0,1,2,3,4,...,1023}

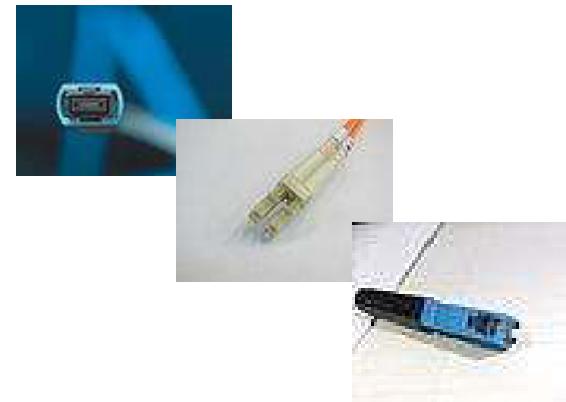
(K=1023 → wait time is about 0.5 msec, with 1 Gbps Ethernet)

## 802.3 Ethernet Standards: Link & Physical Layers

- *Many* different Ethernet standards:
  - Common MAC protocol and frame format;
  - Different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 100 Gbps, 400 Gbps;
  - Different physical layer media: fiber, cable;



# 802.3 Ethernet Standards: Link & Physical Layers



## 400 Gbit/s [ edit ]

Main article: [Terabit Ethernet](#)

The Institute of Electrical and Electronics Engineers (IEEE) has defined a new Ethernet standard capable of 200 and 400 Gbit/s in IEEE 802.3bs-2017.<sup>[25]</sup> 1 Tbit/s may be a further goal.<sup>[26]</sup>

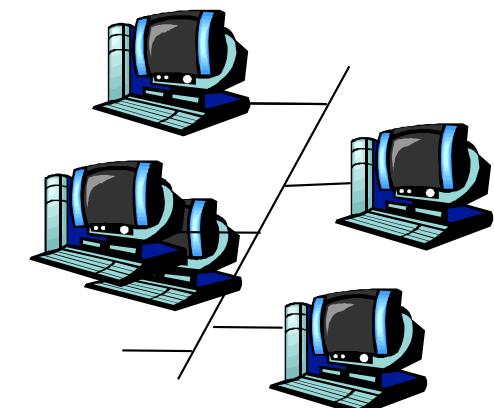
In May 2018, IEEE 802.3 started the 802.3ck Task Force to develop standards for 100, 200, and 400 Gbit/s PHYs and attachment unit interfaces (AUI) using 100 Gbit/s lanes.<sup>[24]</sup>

In 2008, Robert Metcalfe, one of the co-inventors of Ethernet, said he believed commercial applications using Terabit Ethernet may occur by 2015, though it might require new Ethernet standards.<sup>[27]</sup> It was predicted this would be followed rapidly by a scaling to 100 Terabit, possibly as early as 2020. It is worth noting that these were theoretical predictions of technological ability, rather than estimates of when such speeds would actually become available at a practical price point.<sup>[28]</sup>

Name	Standard (Clause)	Common connectors	Description
<b>Fiber-optical cable</b>			
400GBASE-SR16	802.3bs-2017 (123)	MPO	sixteen lanes (26.5625 Gbit/s) using individual strands of OM4/OM5 multi-mode fiber with 100 m reach or 70 m over OM3
400GBASE-DR4	802.3bs-2017 (124)	MPO	four PAM-4 lanes (53.125 GBd) using individual strands of single-mode fiber with 500 m reach (1310 nm)
400GBASE-FR8	802.3bs-2017 (122)	SC, LC	eight PAM-4 lanes (26.5625 GBd) using eight wavelengths (CWDM) over single-mode fiber with 2 km reach
400GBASE-LR8	802.3bs-2017 (122)	SC, LC	eight PAM-4 lanes (26.5625 GBd) using eight wavelengths (DWDM) over single-mode fiber with 10 km reach
400GBASE-FR4	802.3cu	SC, LC	four lanes/wavelengths (CWDM, 1271/1291/1311/1331 nm) over single-mode fiber with 2 km reach
400GBASE-LR4			four lanes over single-mode fiber with 10 km reach
400GBASE-SR8	802.3cm	SC, LC	eight-lane using individual strands of multi-mode fiber with 100 m reach
400GBASE-SR4.2			four-lane using individual strands of multi-mode fiber with 100 m reach
400GBASE-ER8	802.3cn	SC, LC	eight-lane using eight wavelengths over single-mode fiber with 40 km reach
<b>Other</b>			
400GBASE-KR4	802.3ck (tbd)		four-lane over electrical backplanes supporting an insertion loss of up to 28 dB at 26.56 GBd
400GBASE-CR4			four-lane over twin-axial copper with at least 2 m reach

# *Outline*

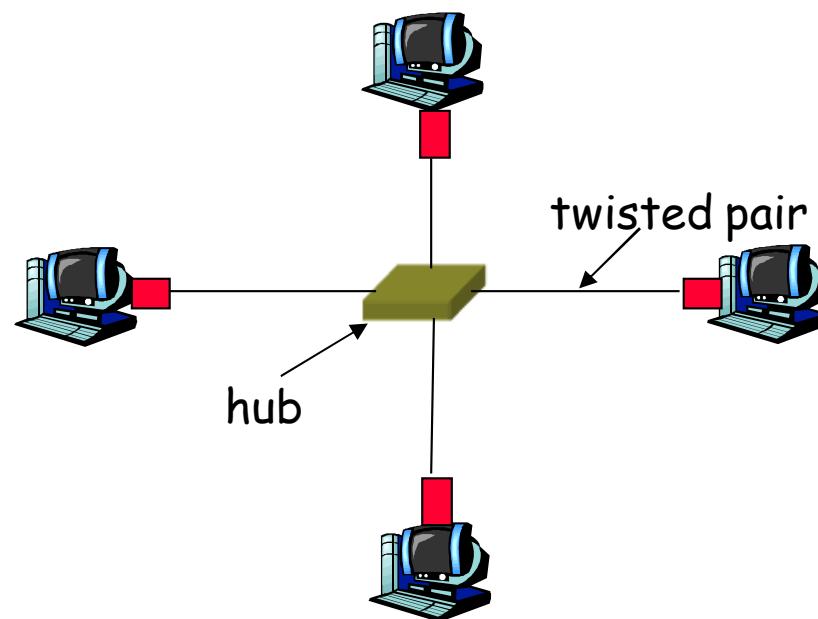
- Link-layer Addressing
- Introduction and services
- Error detection and correction
- Multiple access protocols
- Ethernet
- Link-layer switches
- IEEE 802.11 Wireless LANs
- Framing



# Hubs

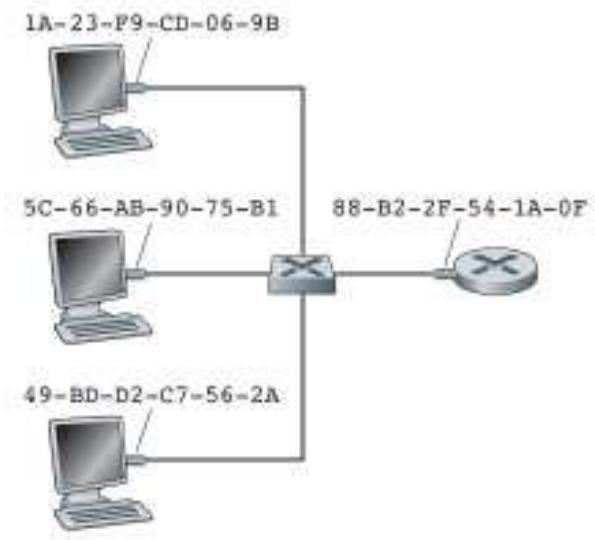
... physical-layer (“dumb”) repeaters:

- Bits coming in one link go out in *all* other links at same rate;
- All nodes connected to hub can collide with one another;
- No frame buffering;
- No CSMA/CD at hub: hosts detect collisions.



# Switch

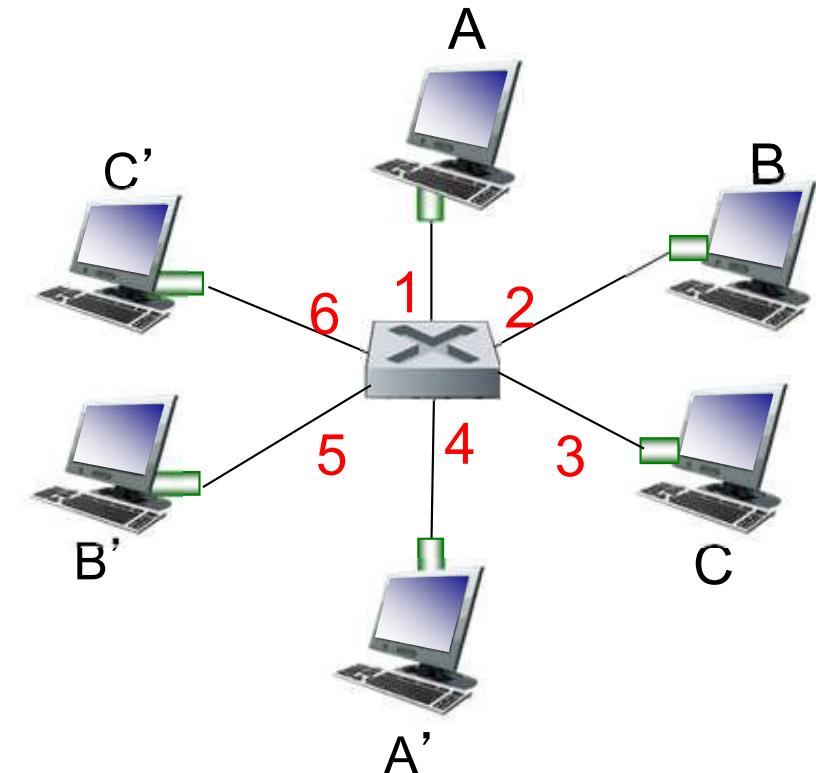
- Link-layer device;  
**Smarter than hubs, takes active role:**
  - Stores, forwards Ethernet frames;
  - Examines incoming frame's MAC address:
    - **Selectively forwards** frame to one or more outgoing links when frame is to be forwarded on segment;
    - Uses CSMA/CD to access segment:
- **Transparent**
  - Hosts are unaware of presence of switches.
- **Plug-and-play, self-learning**
  - Switches do not need to be configured.



# Switch

**Allows Multiple Simultaneous Transmissions**

- Hosts have dedicated, direct connection to switch.
- Switches buffer packets.
- Ethernet protocol used on each incoming link, but no collisions; full duplex:
  - Each link is its own collision domain.
- **Switching:** allows simultaneous communications A-to-A' and B-to-B' without collisions:
  - Not possible with dumb **hub**.



*switch with six interfaces  
(1,2,3,4,5,6)*

# Switch Table

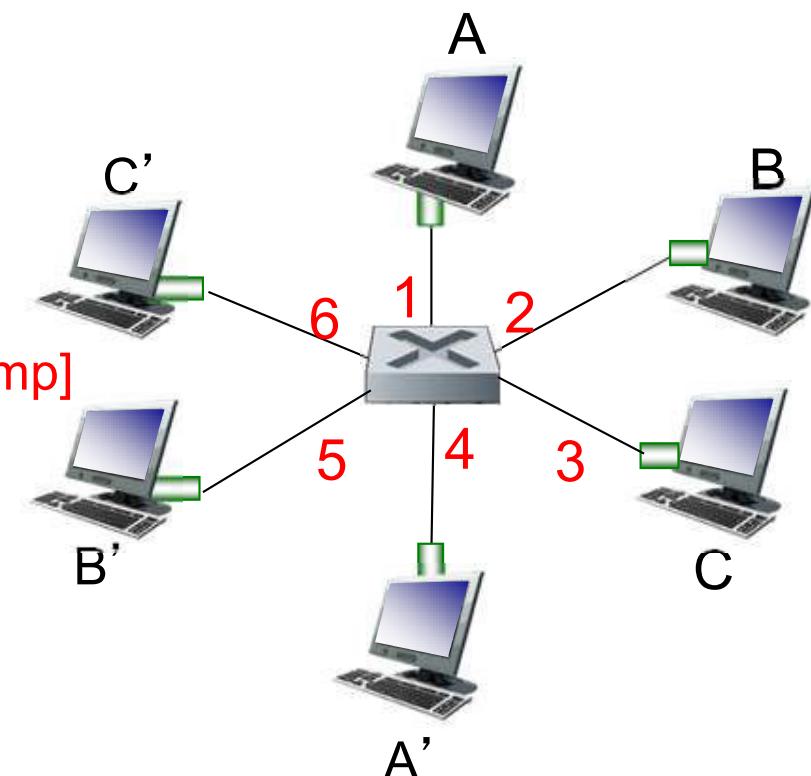
- Q: How does switch know that A' is reachable via interface 4, and B' is reachable via interface 5?

A: Each switch has a **switch table**; Entries contain:  
 [MAC address of host; interface to reach host; time stamp]

Looks like a routing table!

Q: How are entries created and maintained in switch table?

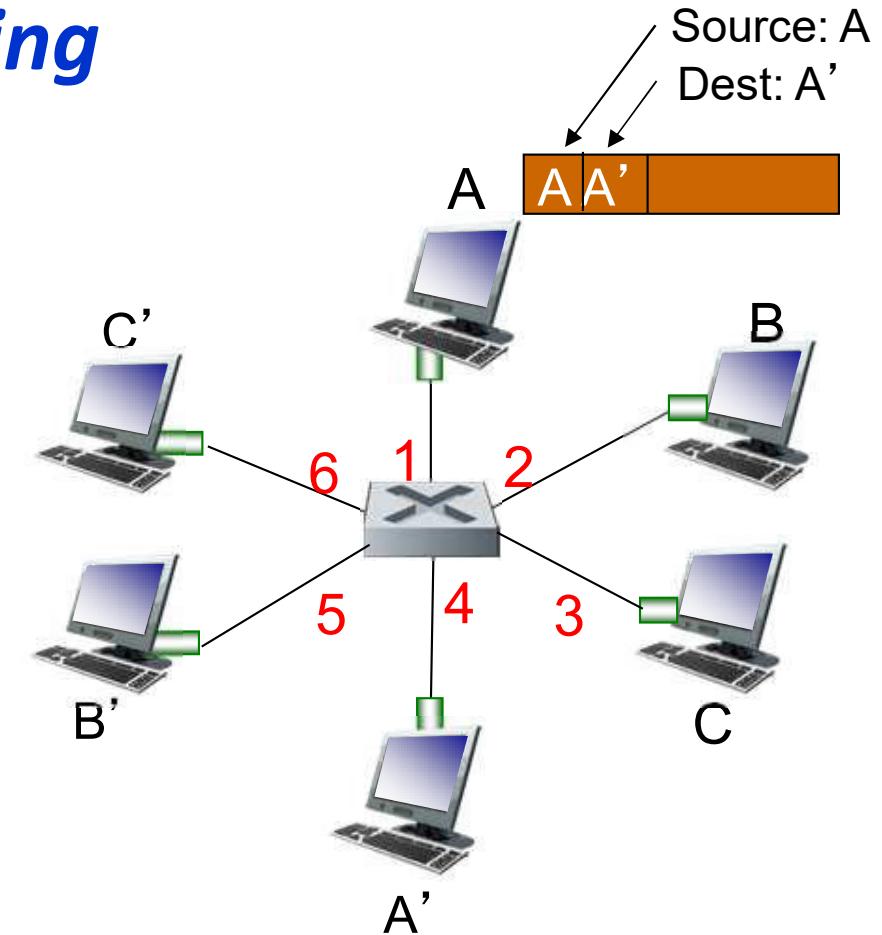
Something like a routing protocol?



*switch with six interfaces  
 (1,2,3,4,5,6)*

# Switch: Self-Learning

- Switch *learns* which hosts can be reached through which interfaces:
  - When frame is received, switch “learns” location of sender → it is the incoming LAN segment
  - Records sender/location pair in the switch table.



MAC addr	interface	TTL
A	1	60

*Switch table  
(initially empty)*

# Switch: Frame Filtering/Forwarding

When frame received:

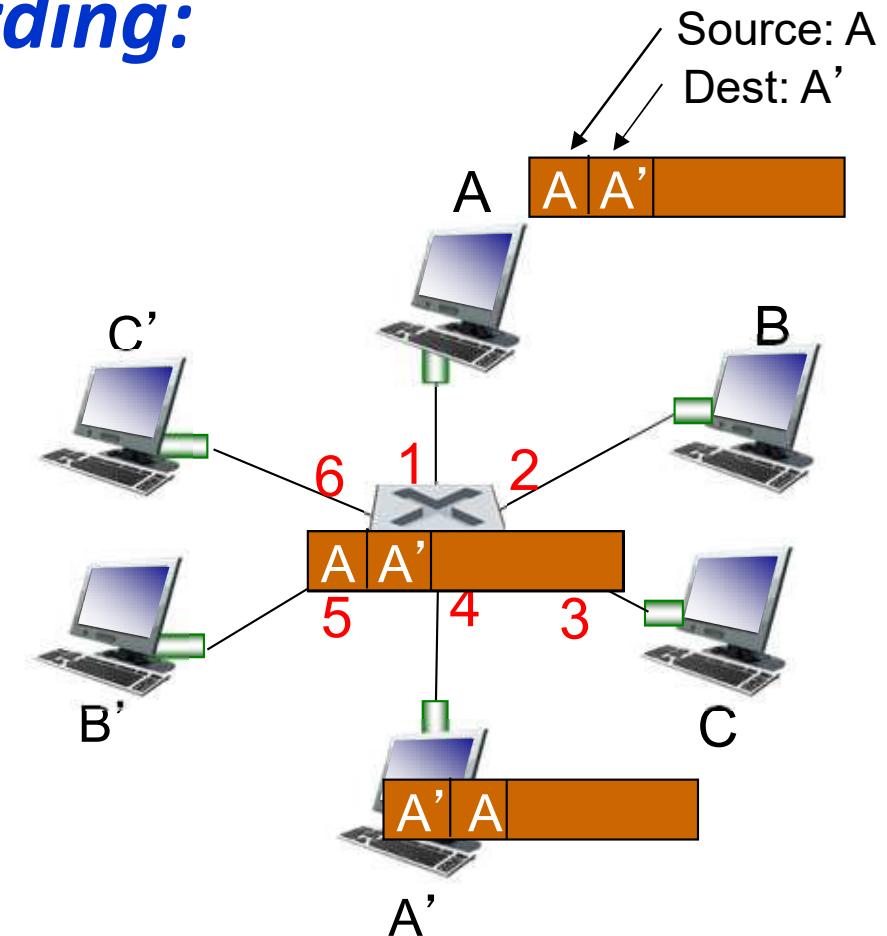
1. Record link associated with sending host;
2. Index switch table using MAC destination address;
3. **if** entry **found** for destination  
**then** {  
    **if** destination on segment from which frame arrived  
        **then** drop the frame;  
        **else** forward the frame on interface indicated  
    }  
**else** flood.

*forward on all but the interface  
on which the frame arrived*



# Self-learning, Forwarding: Example

- Frame destination unknown:  
*flood*
- Destination is a known location:  
*selective send*



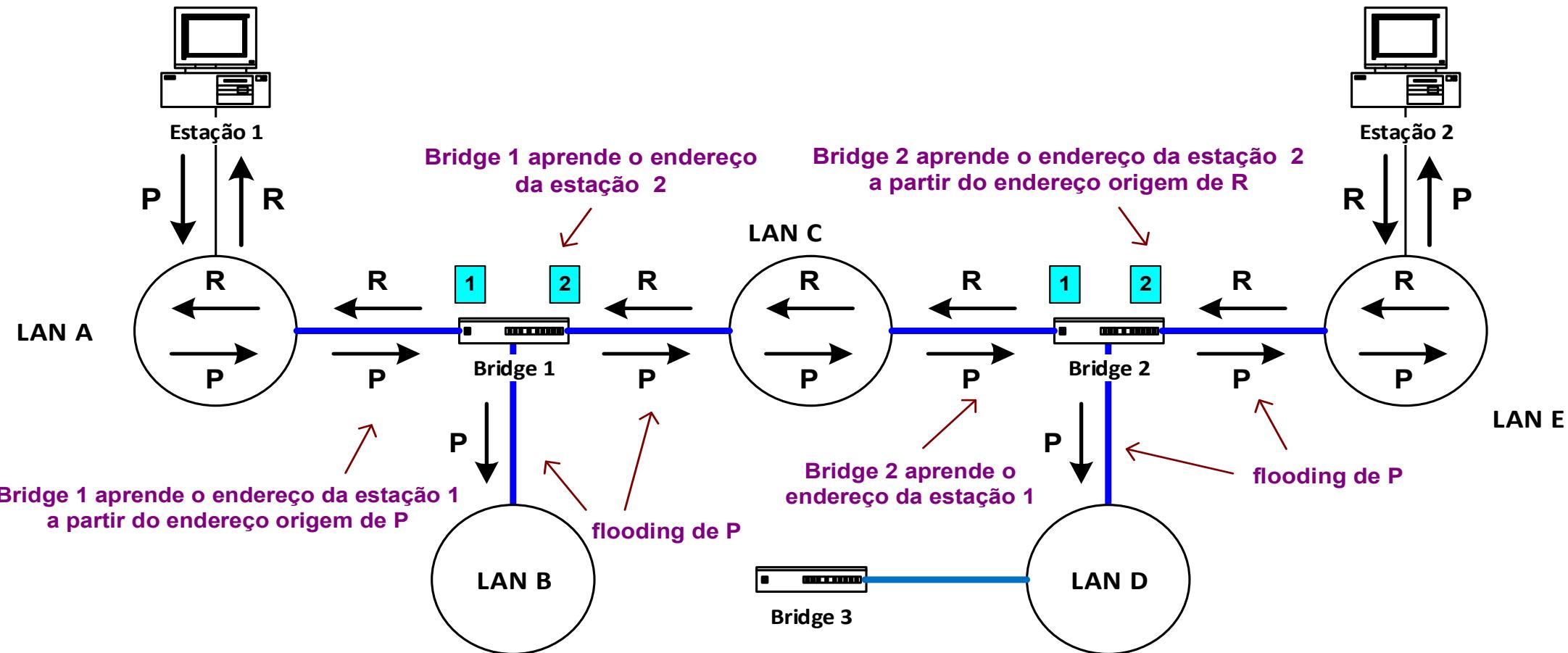
MAC addr	interface	TTL
A	1	60
A'	4	60

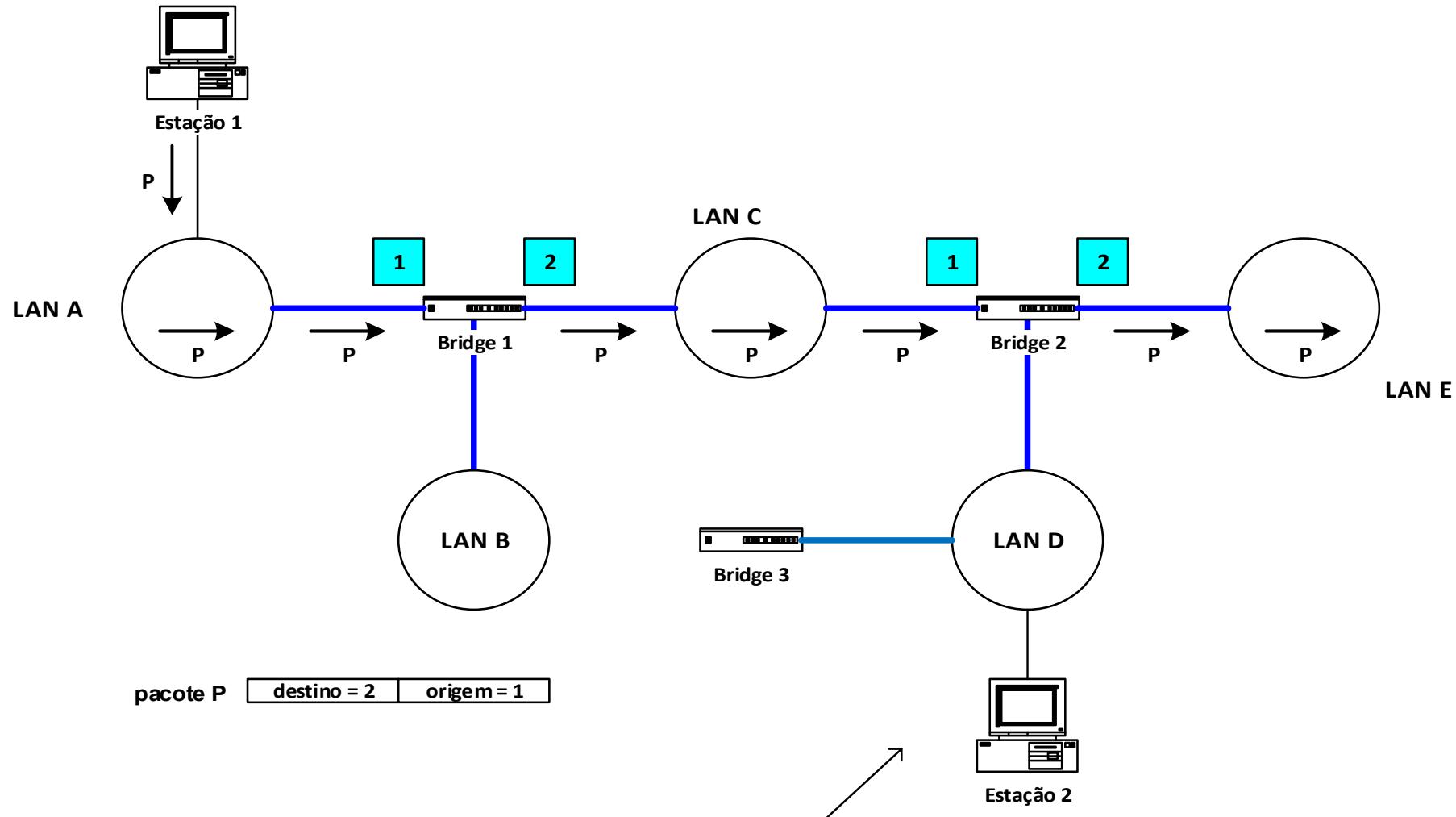
*switch table  
(initially empty)*

# Self-learning: Example

pacote P destino = 2 origem = 1

pacote R destino = 1 origem = 2



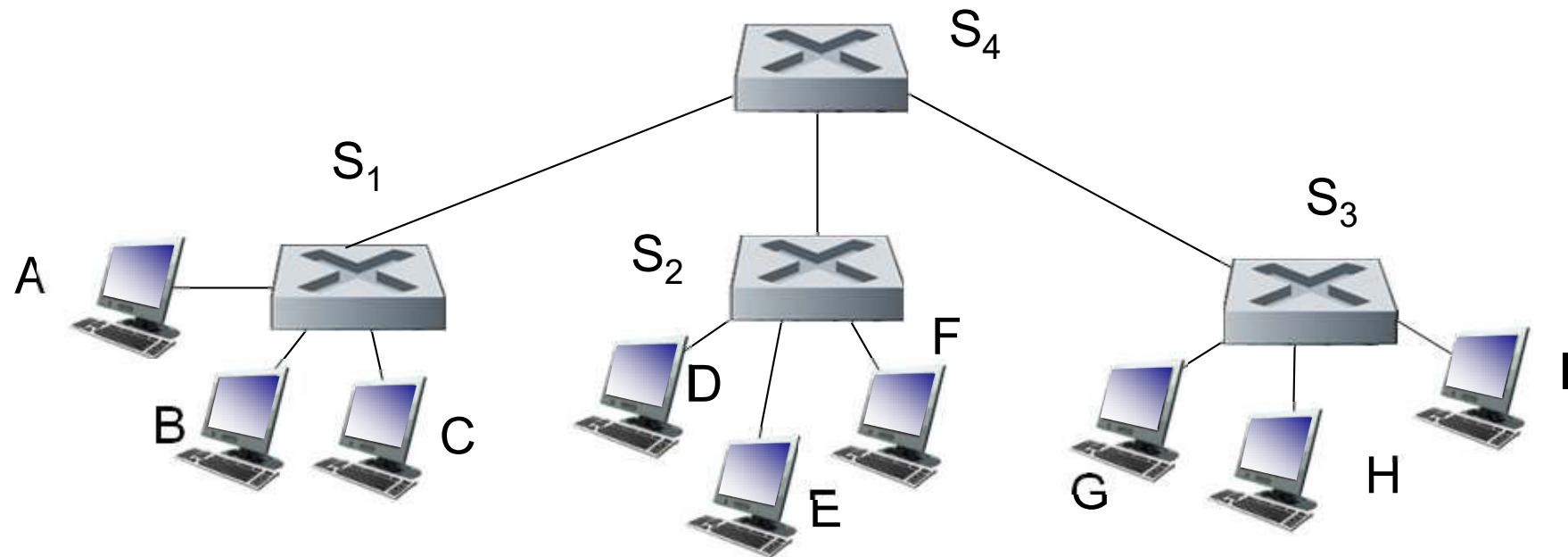


**Estação 2 muda de LAN: não recebe pacotes**

- até transmitir ou até
- expirar tempo de vida da sua entrada na tabela de encaminhamento

# Interconnecting Switches

- Switches can be interconnected:



Q: Sending from A to G:

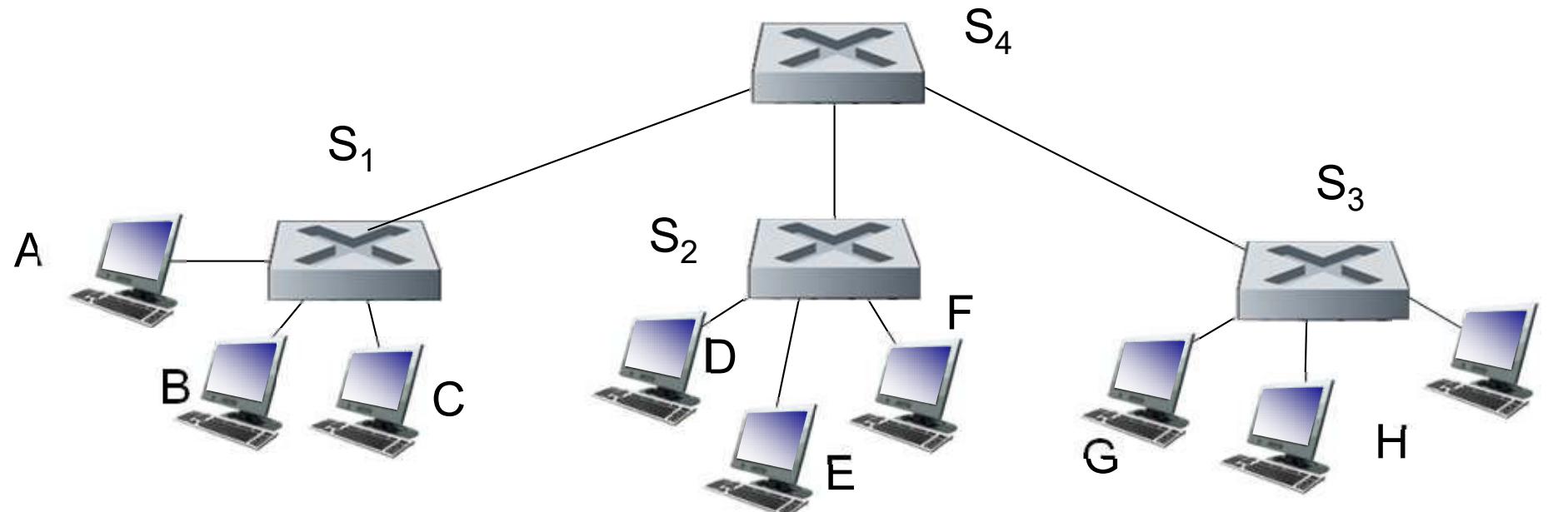
How does  $S_1$  know to forward frame destined to G via  $S_4$  and  $S_3$ ?

A: Self learning!

(Works exactly as in single-switch case!)

# Self-learning Multi-switch Example

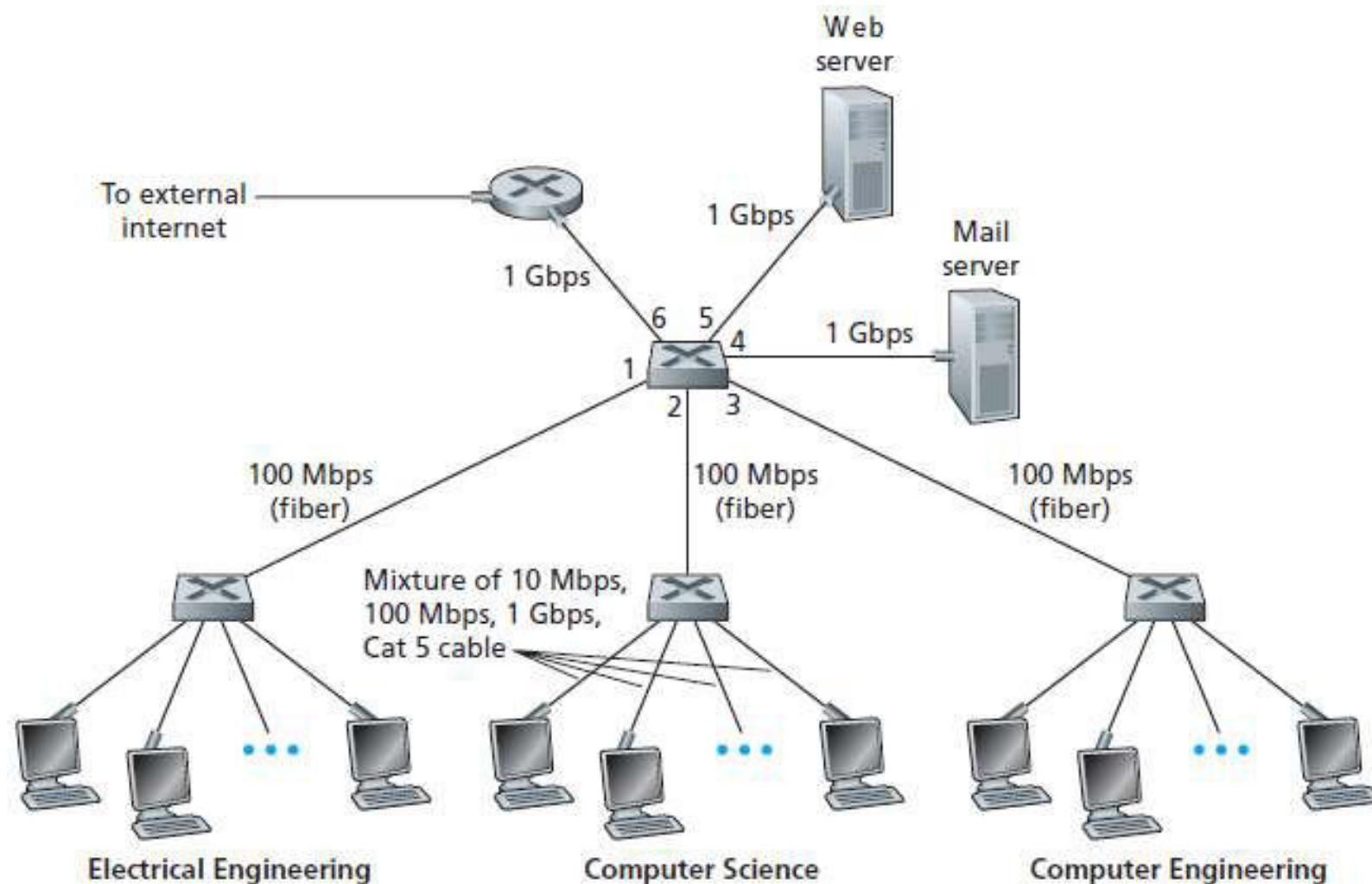
Suppose C sends frame to I; then I responds to C:



Q: Show switch tables and packet forwarding in  $S_1$ ,  $S_2$ ,  $S_3$ ,  $S_4$

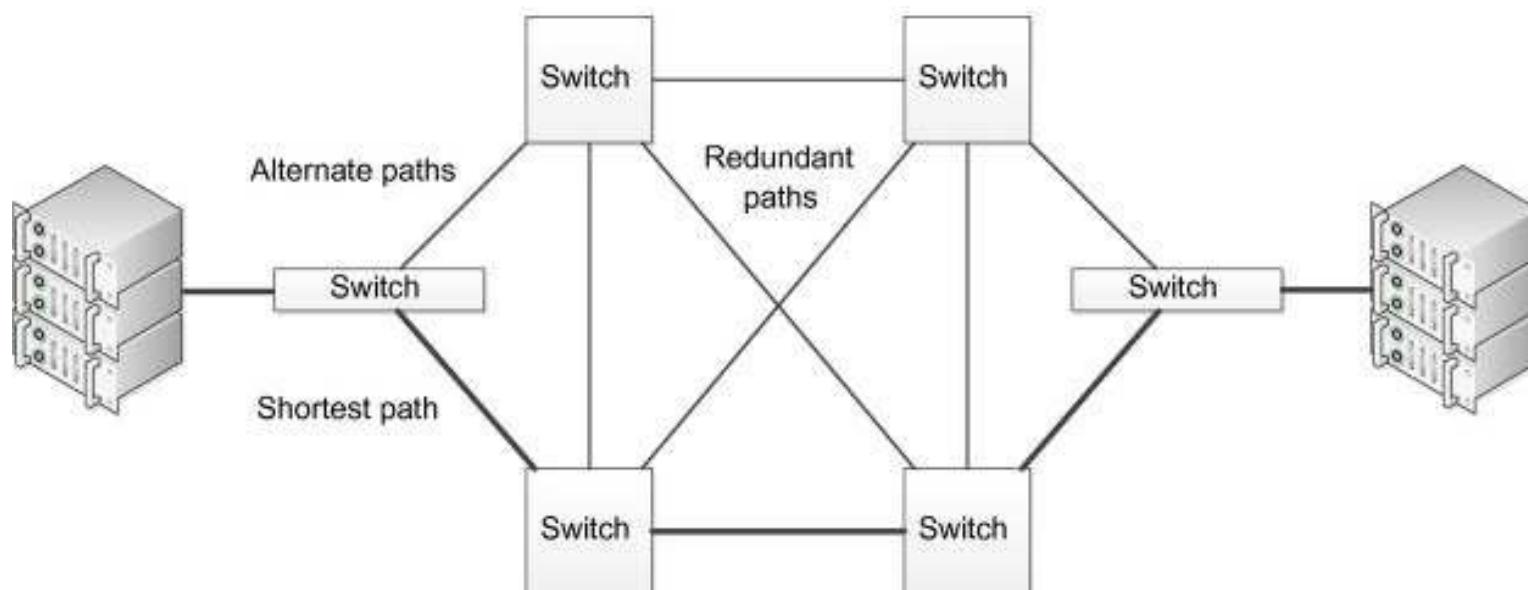
# Example

An institutional network connected (one single subnet) using four switches:



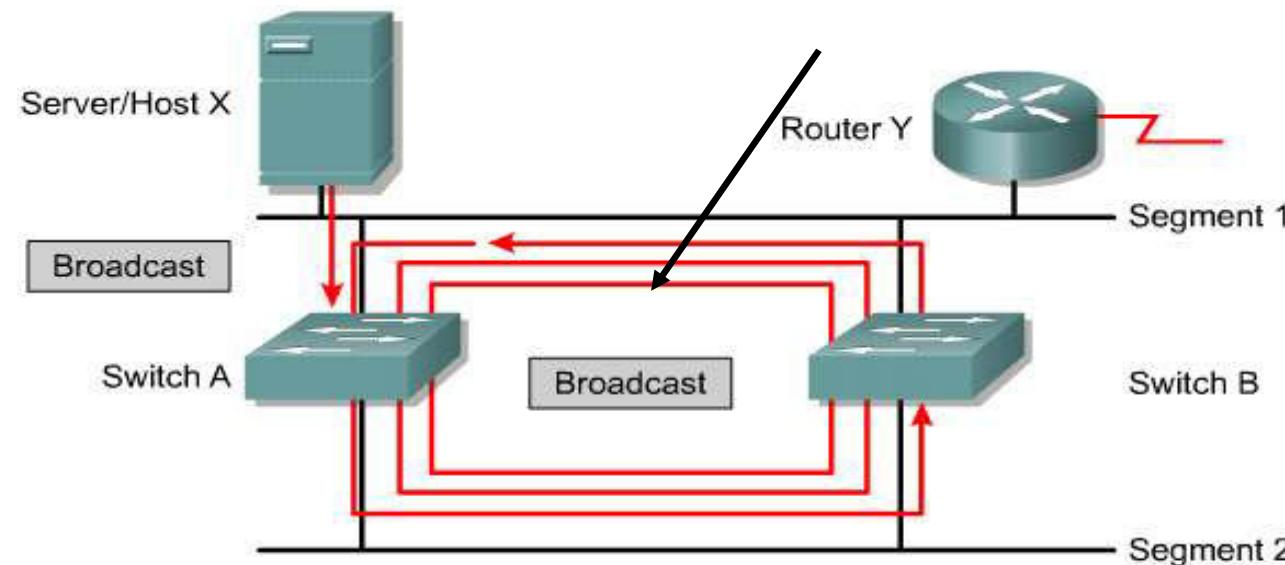
## *Redundant Topology*

- Networks introduce **redundant links** between switches or bridges to overcome the failure of links.
- These connections introduce **physical loops** into the network!
- Bridging loops are created so **if one link fails another can take over** the function of traffic forwarding!



## Redundant Topology

- Switches flood traffic out all ports when the traffic is sent to a destination not yet known.
- Broadcast and multicast traffic is forwarded out on every port (except the port on which the traffic arrived).
- This traffic can be caught in a loop.



- Host X sends a broadcast.
- Switches continue to propagate broadcast traffic over and over.

## *Redundant Topology*

In Layer 2 header there is no Time To Live (TTL):

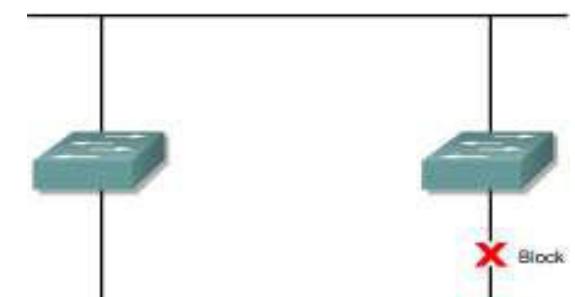
- A frame sent into a Layer 2 redundant topology of switches, **can loop forever...**

(In Layer 3 the TTL is decremented and the packet is discarded when TTL reaches 0).



This creates a dilemma:

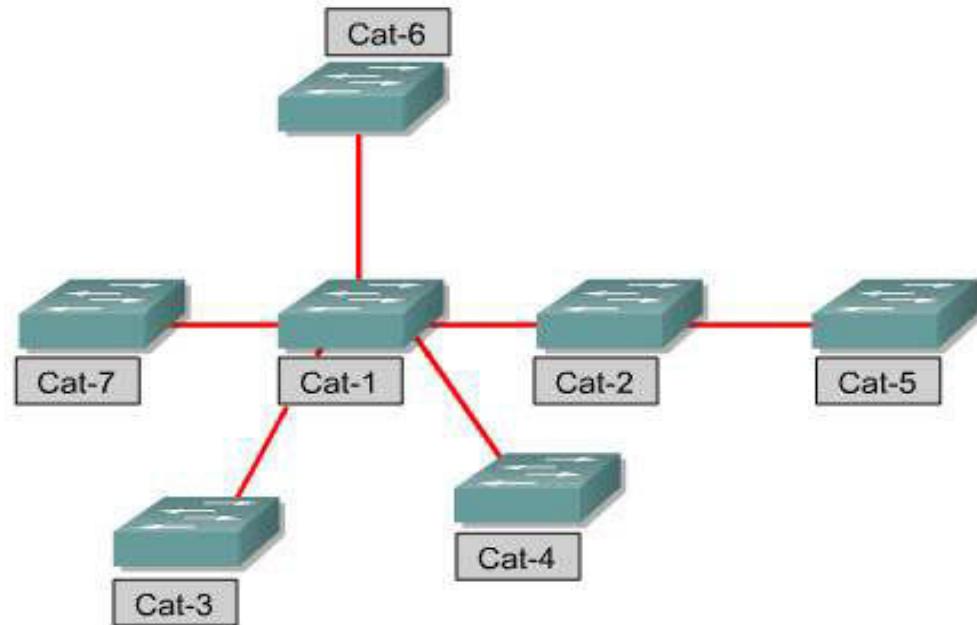
- **Loops are needed in the physical topology for reliability**
- **But, a switched network cannot have loops.**



# *Redundant Topology and Spanning Tree*

Solution:

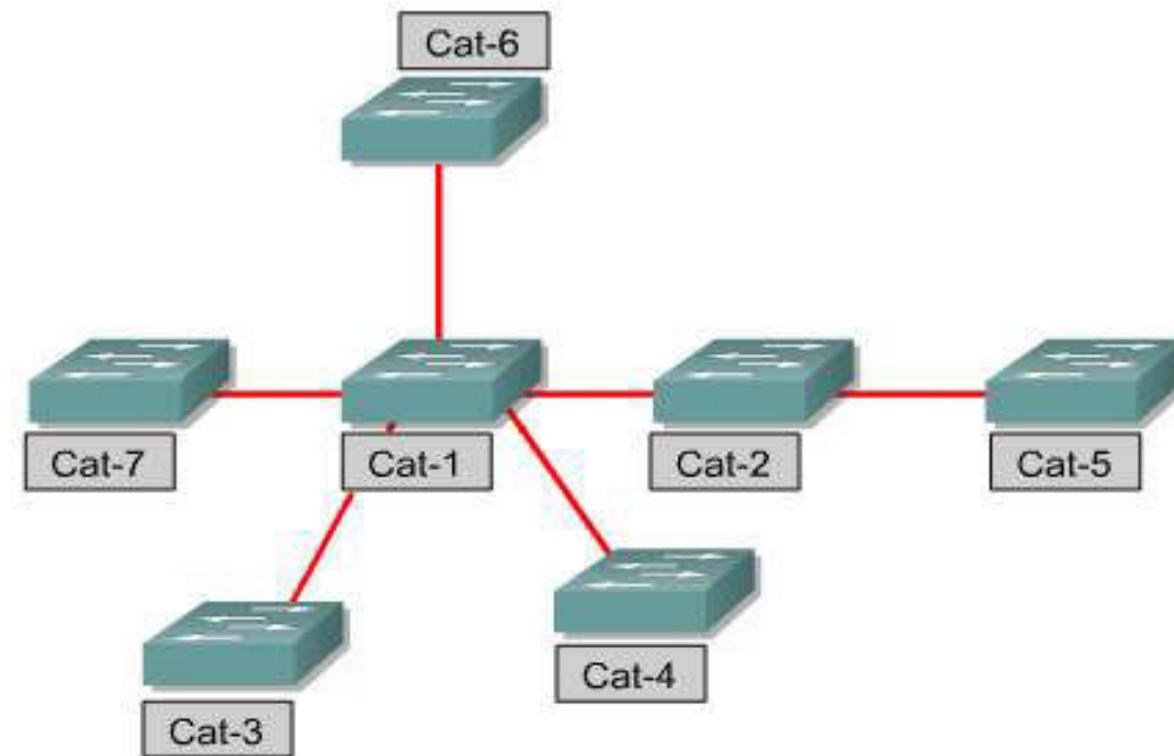
- Allow physical loops, but **create a loop free logical topology**.
- Traffic destined to Cat-5 from any host attached to Cat-4 will travel through Cat-1 and Cat-2.
- This happens even if there is a direct physical connection between Cat-5 and Cat-4.



The loop free logical topology created is a **tree** and has a star or extended star logical topology: the **spanning tree** of the network.

## Spanning-Tree Protocol

- Ethernet bridges and switches can implement the **IEEE 802.1D Spanning-Tree Protocol** and use the spanning-tree algorithm to construct a loop free shortest path network.



# *Spanning-Tree Protocol*

- The Spanning-Tree Protocol establishes a root node, called the **root bridge**.
- The Spanning-Tree Protocol constructs a **topology** that has **one path for reaching every network node**.
- The **resulting tree originates from the root bridge**.
- **Redundant links** that are not part of the shortest path tree are **blocked**.

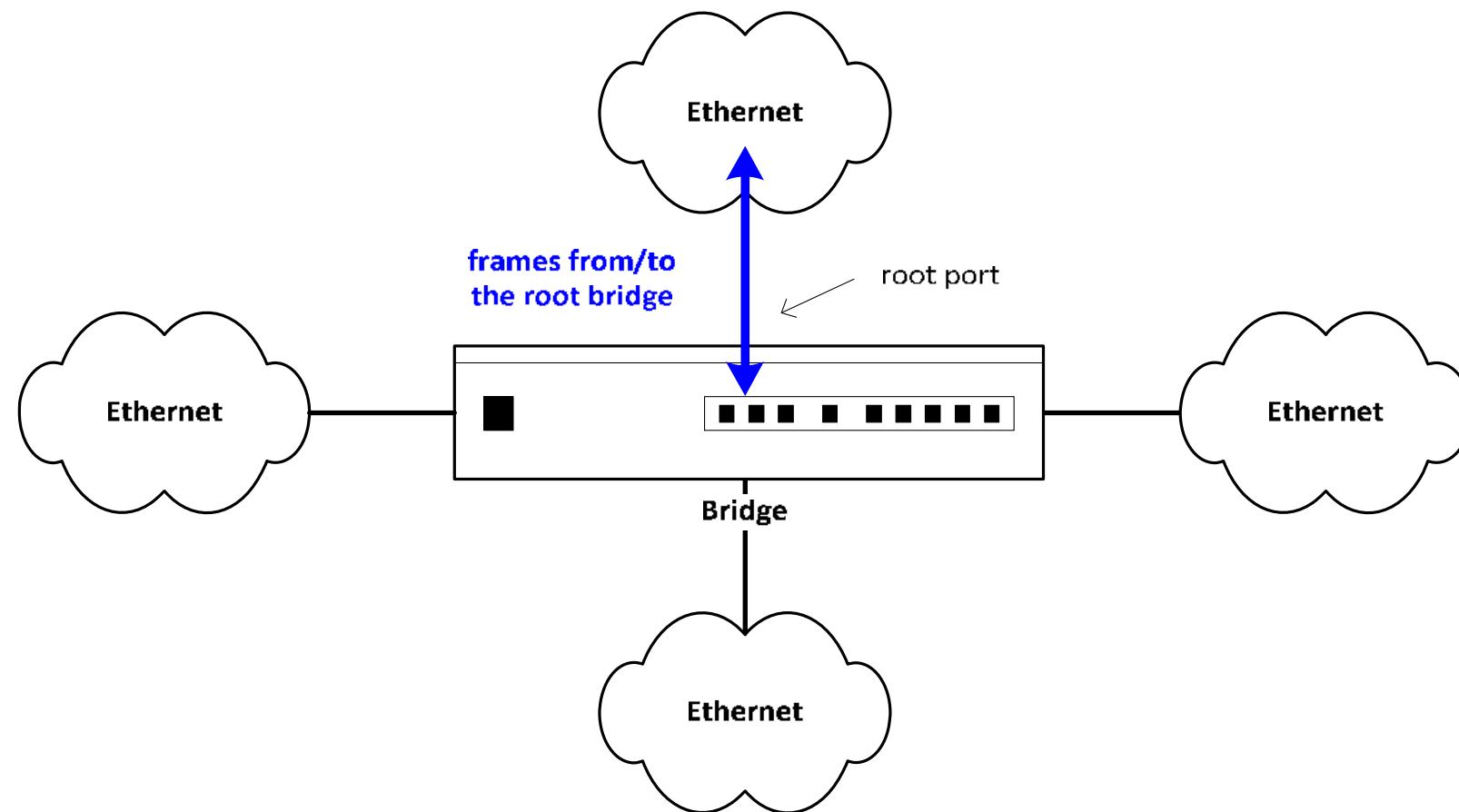
## Basic Spanning Tree Concepts

- Bridge ID – each bridge is identified by an address containing:
  - 2 priority bytes – configurable by the network manager
  - 6 fixed bytes (one of the bridge ports MAC addresses, or any other 48 bit unique address)

The priority part has precedence over the fixed bytes.
- Root Bridge – bridge in the root of the spanning tree; bridge with the **lower** Bridge ID

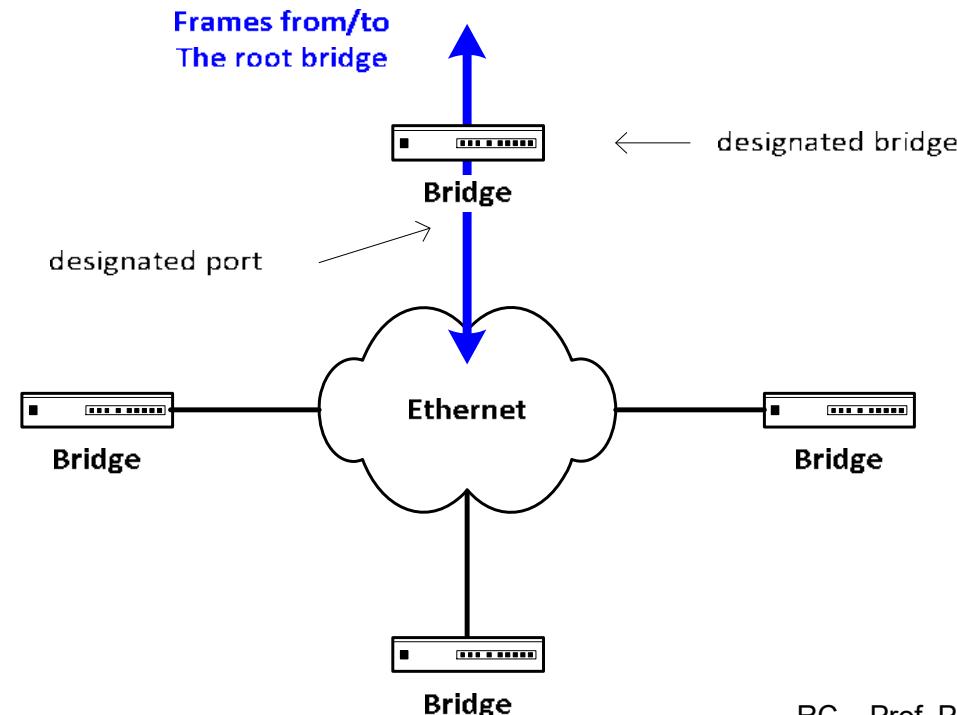
# Basic Spanning Tree Concepts

- *Root Port* – port which, in a bridge, is responsible for the reception/transmission of frames from/to the root bridge



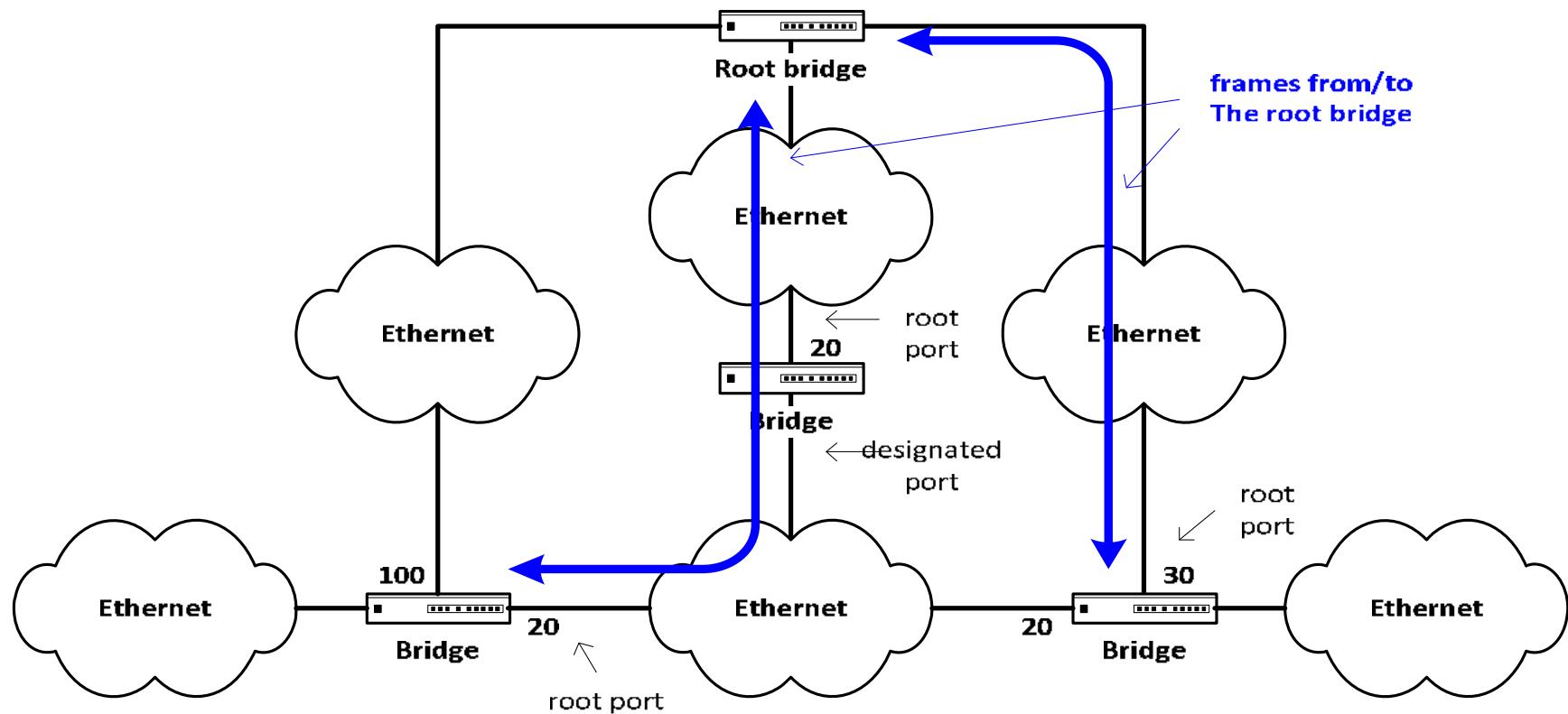
# Basic Spanning Tree Concepts

- *Designated Bridge* – bridge which, for a given LAN, is responsible for sending frames from that LAN to the root bridge and vice-versa; (the root bridge is the designated bridge in all the LANs to which it is connected);
- *Designated Port* – port which, for a given LAN, is responsible for sending frames from that LAN to the root bridge and vice-versa (it is the ports of the designated bridge connected to the LAN)



# Basic Spanning Tree Concepts

- Each bridge has an associated cost of the path to the root bridge (*Root Path Cost*), equal to the sum of the costs from the ports that transmit frames towards the root bridge (root ports), in the least cost path to the root bridge



## *Spanning-Tree Protocol*

- Shortest path is based on cumulative link costs.
- Link costs are based on the speed of the link:

**Table 17-3—Port Path Cost values**

Link Speed	Recommended value	Recommended range	Range
<=100 Kb/s	200 000 000*	20 000 000–200 000 000	1–200 000 000
1 Mb/s	20 000 000 <sup>a</sup>	2 000 000–200 000 000	1–200 000 000
10 Mb/s	2 000 000 <sup>a</sup>	200 000–20 000 000	1–200 000 000
100 Mb/s	200 000 <sup>a</sup>	20 000–2 000 000	1–200 000 000
1 Gb/s	20 000	2 000–200 000	1–200 000 000
10 Gb/s	2 000	200–20 000	1–200 000 000
100 Gb/s	200	20–2 000	1–200 000 000
1 Tb/s	20	2–200	1–200 000 000
10 Tb/s	2	1–20	1–200 000 000

\*Bridges conformant to IEEE Std 802.1D, 1998 Edition, i.e., that support only 16-bit values for Path Cost, should use 65 535 as the Path Cost for these link speeds when used in conjunction with Bridges that support 32-bit Path Cost values.

## *Basic Spanning Tree Concepts*

- The root port is, in each bridge, the port which provides the best path (lower cost) to the root
- The designated port is, in each LAN, the port which provides the best path (lower cost) to the root

The **active ports** in each bridge are:  
root port + the designated ports

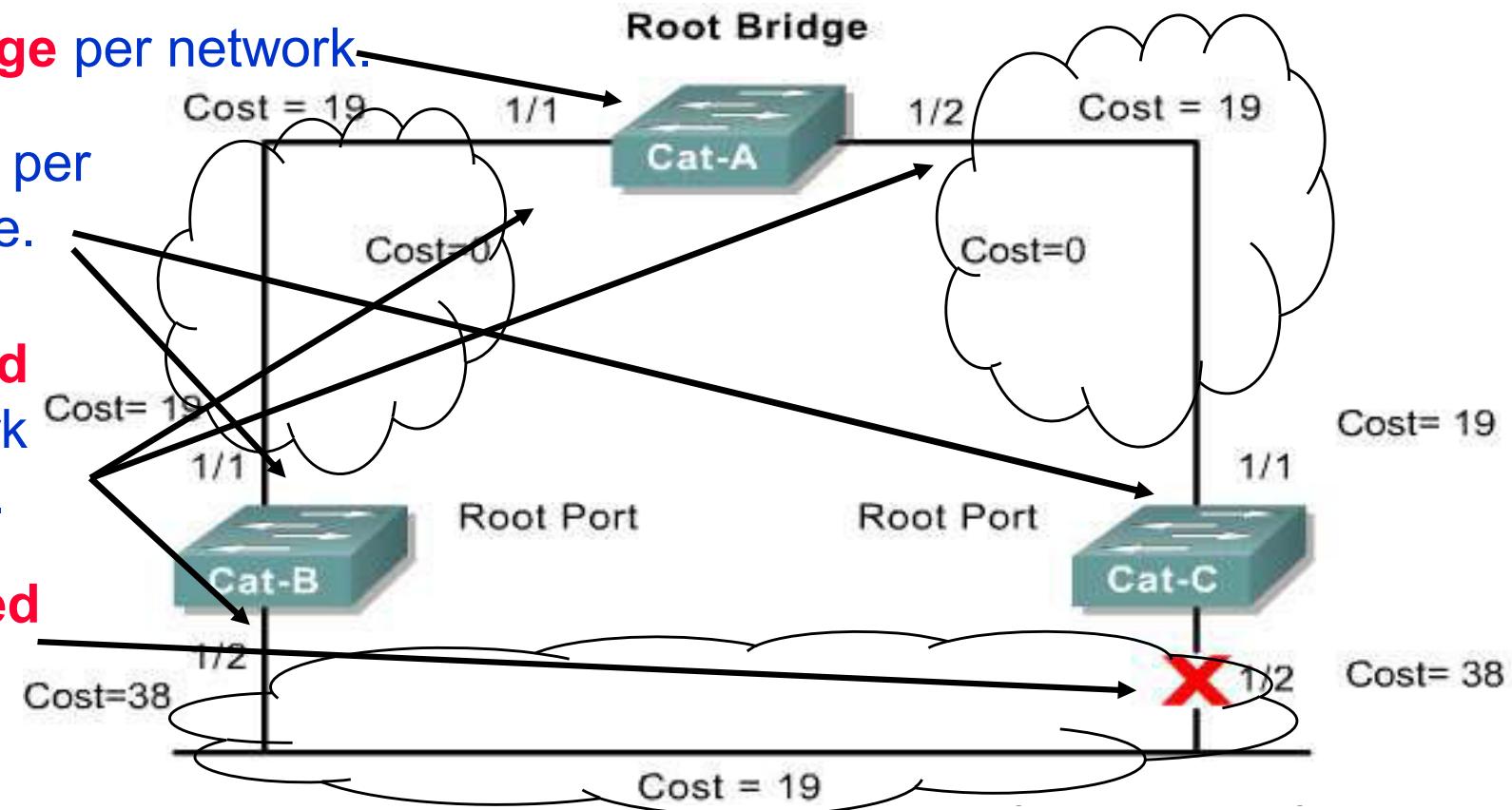
The remaining ports stay **inactive (blocked)**

## Spanning-Tree Operation

When the network has stabilized, it has converged and there is one spanning tree per network.

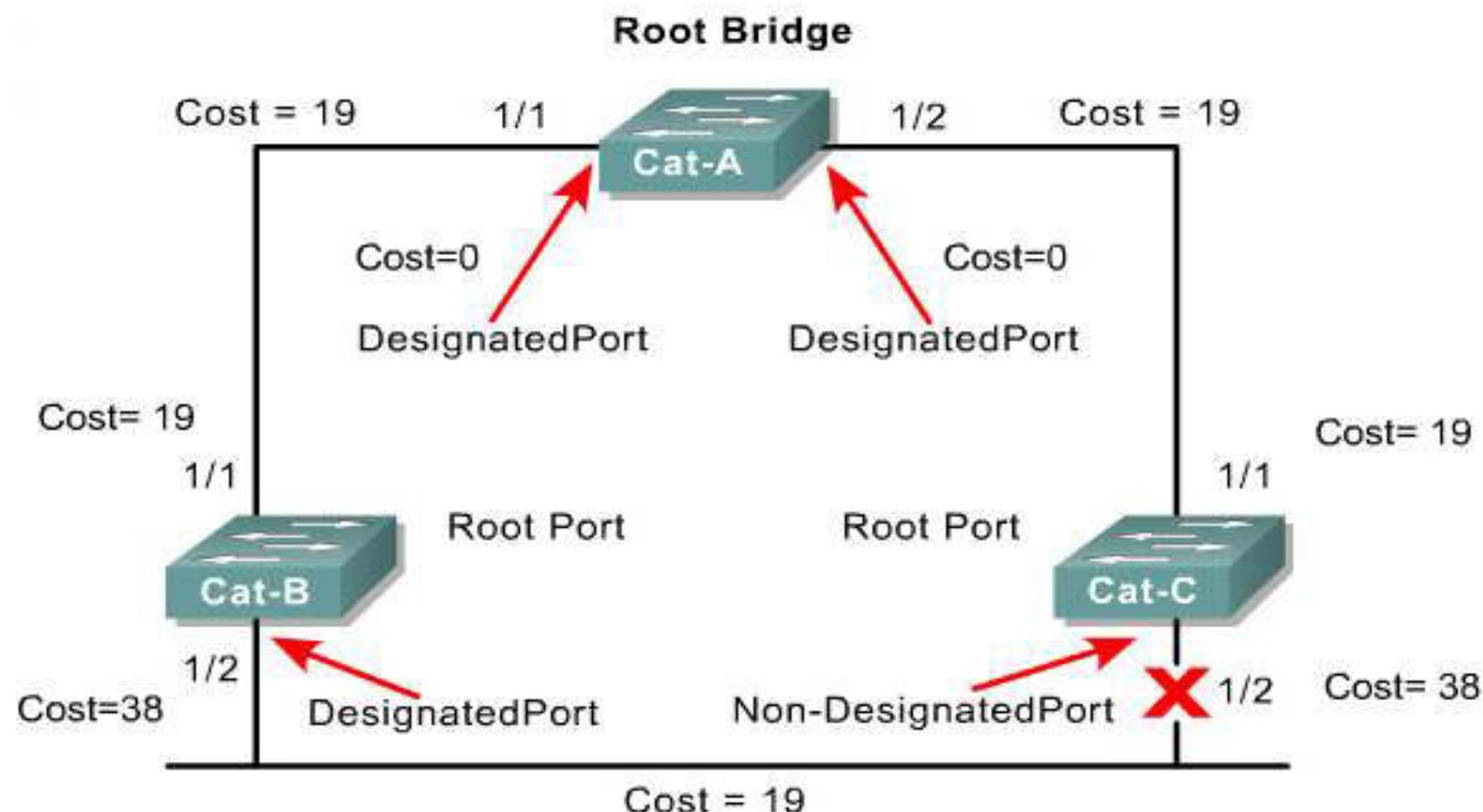
As a result, for every switched network the following elements exist:

- One **root bridge** per network.
- One **root port** per non-root bridge.
- One **designated port** per network (LAN) segment.
- Unused, **blocked ports**.



# Spanning-Tree Protocol

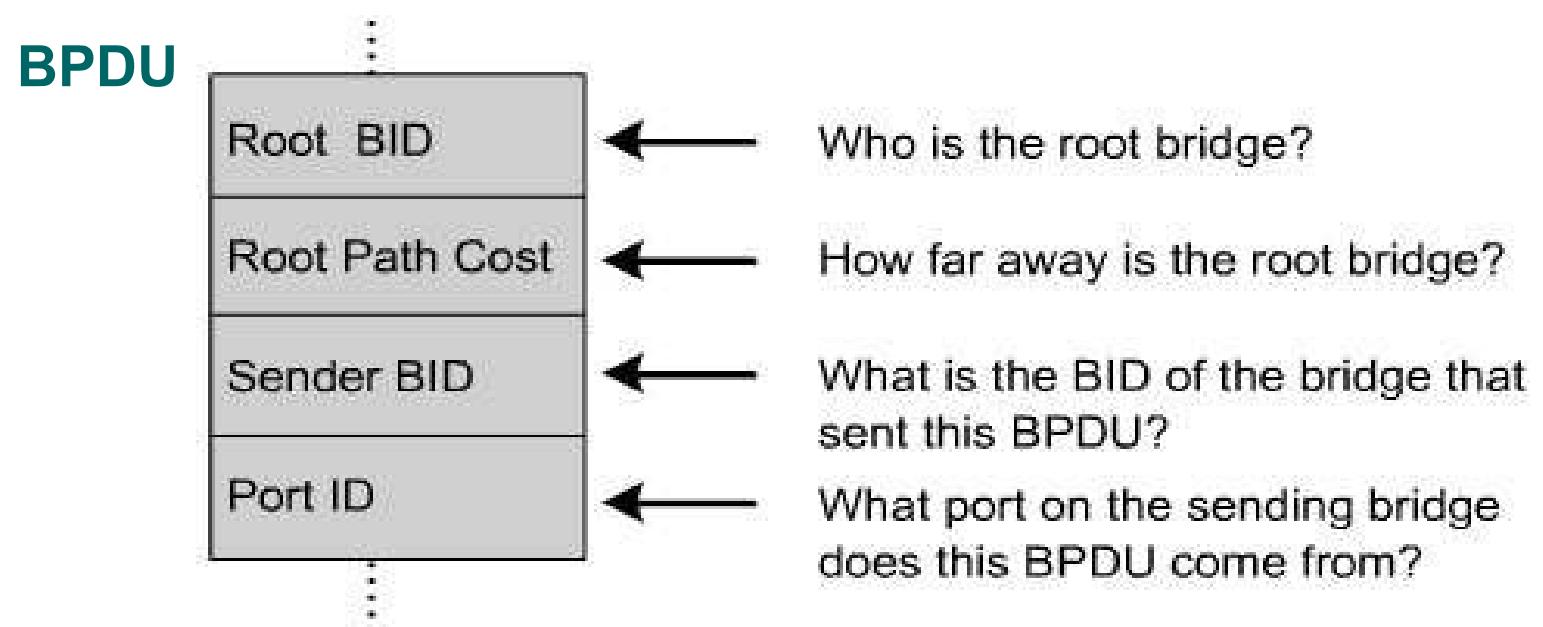
- The Spanning-Tree Protocol requires network devices to exchange messages to detect bridging loops.
- Links that will cause a loop are put into a blocking state.



# Spanning-Tree Protocol

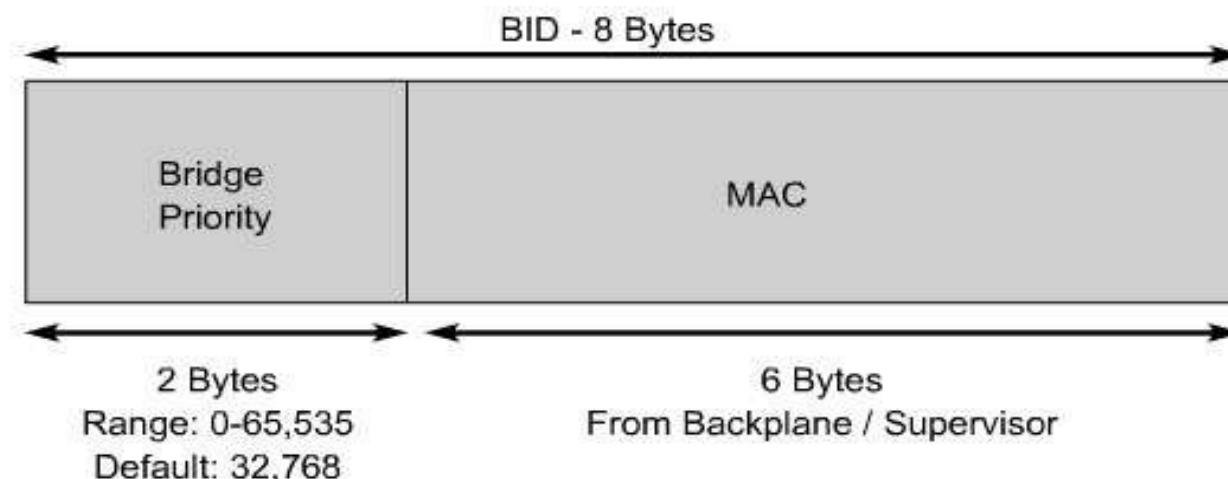
The message that a switch/bridge sends, allowing the formation of a loop free logical topology, is called a **Bridge Protocol Data Unit (BPDU)**.

- BPDUs continue to be received on blocked ports.
- This ensures that if an active path or device fails, a new spanning tree can be calculated.



## Selecting the Root Bridge

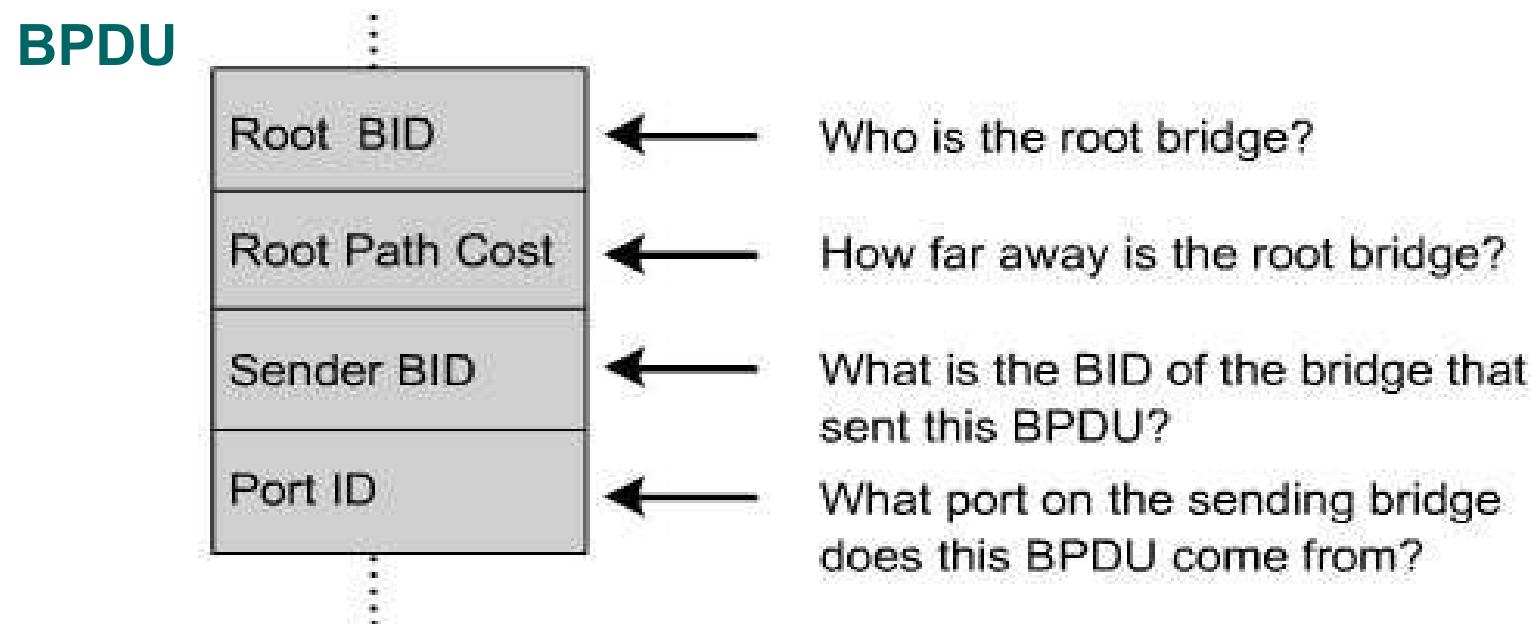
- When a switch (or bridge) is turned on, the spanning-tree algorithm is used to identify the root bridge.
- BPDUs are sent out with the **Sender Bridge ID (BID)**.
- The BID consists of a bridge priority that defaults to 32768 and the switch base MAC address (lowest MAC address of bridge ports).



- Bridge ID (BID) is used to identify each bridge/switch.
- The BID is used in determining the center of the network, in respect to STP, known as the root bridge.

## Selecting the Root Bridge

- When a switch first starts up, it assumes it is the root switch and sends “inferior” BPDUs.
- These BPDUs contain the switch MAC address in both the root and sender BID.
- **By default BPDUs are sent every two seconds.**



## Selecting the Root Bridge

- All switches see the BIDs sent.
- As a switch receives a BPDU with a lower root BID it replaces that in the subsequent PDUs that are sent out.
- All bridges see these and decide that **the bridge with the smallest BID value will be the root bridge.**



MAC-ADDRESS: AAAA.AAAA.AAAA

32768.AAAA.AAAA.AAAA
0
32768.AAAA.AAAA.AAAA
Sending Port ID e.g. 13

Root BID

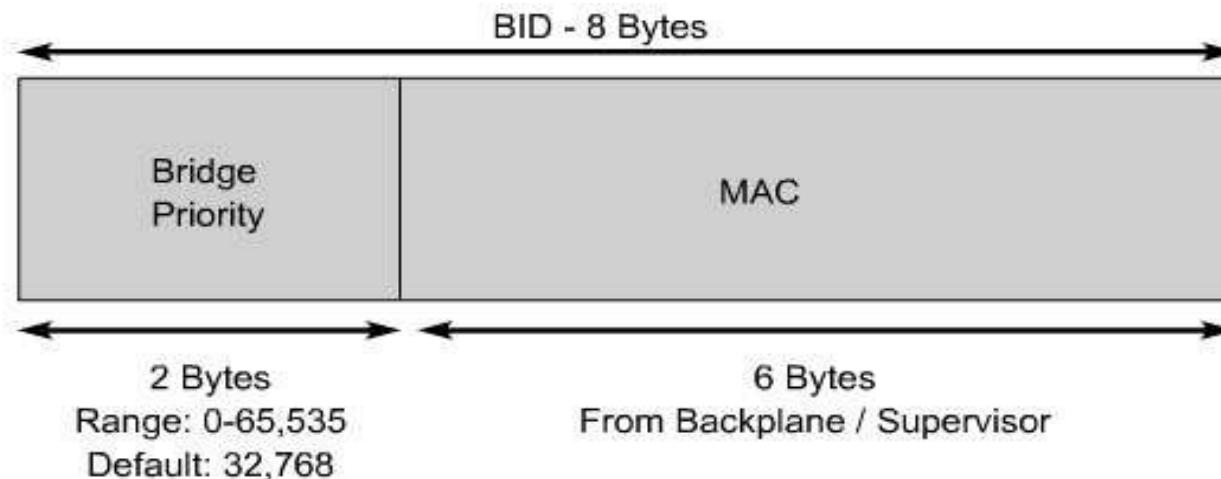
Root Path Cost

Sender BID

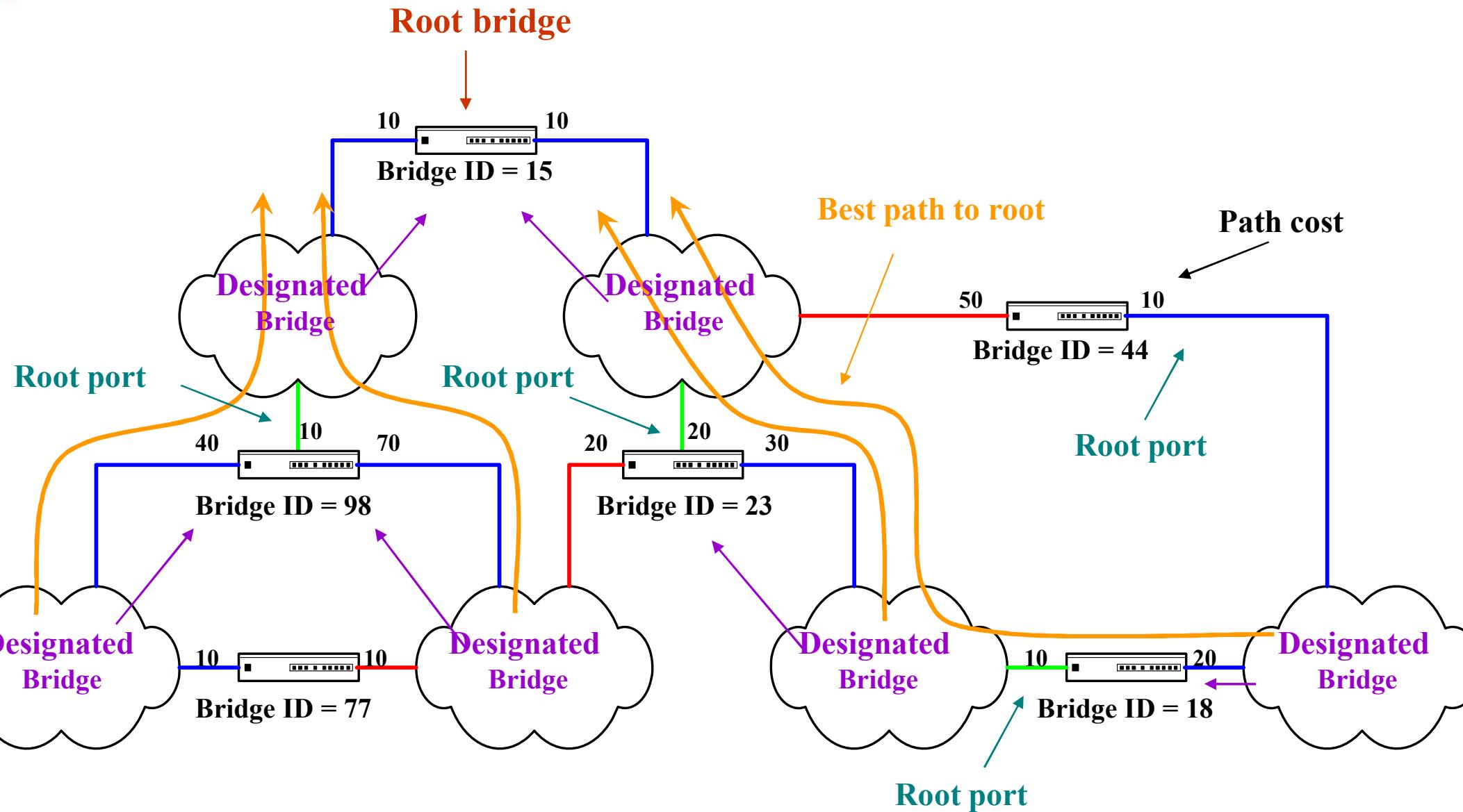
Port ID

## Selecting the Root Bridge

- A network administrator may want to influence the decision by setting the switch priority to a smaller value than the default, which will make the BID smaller.
- This should only be implemented when the traffic flow on the network is well understood.



# Basic Spanning Tree Concepts

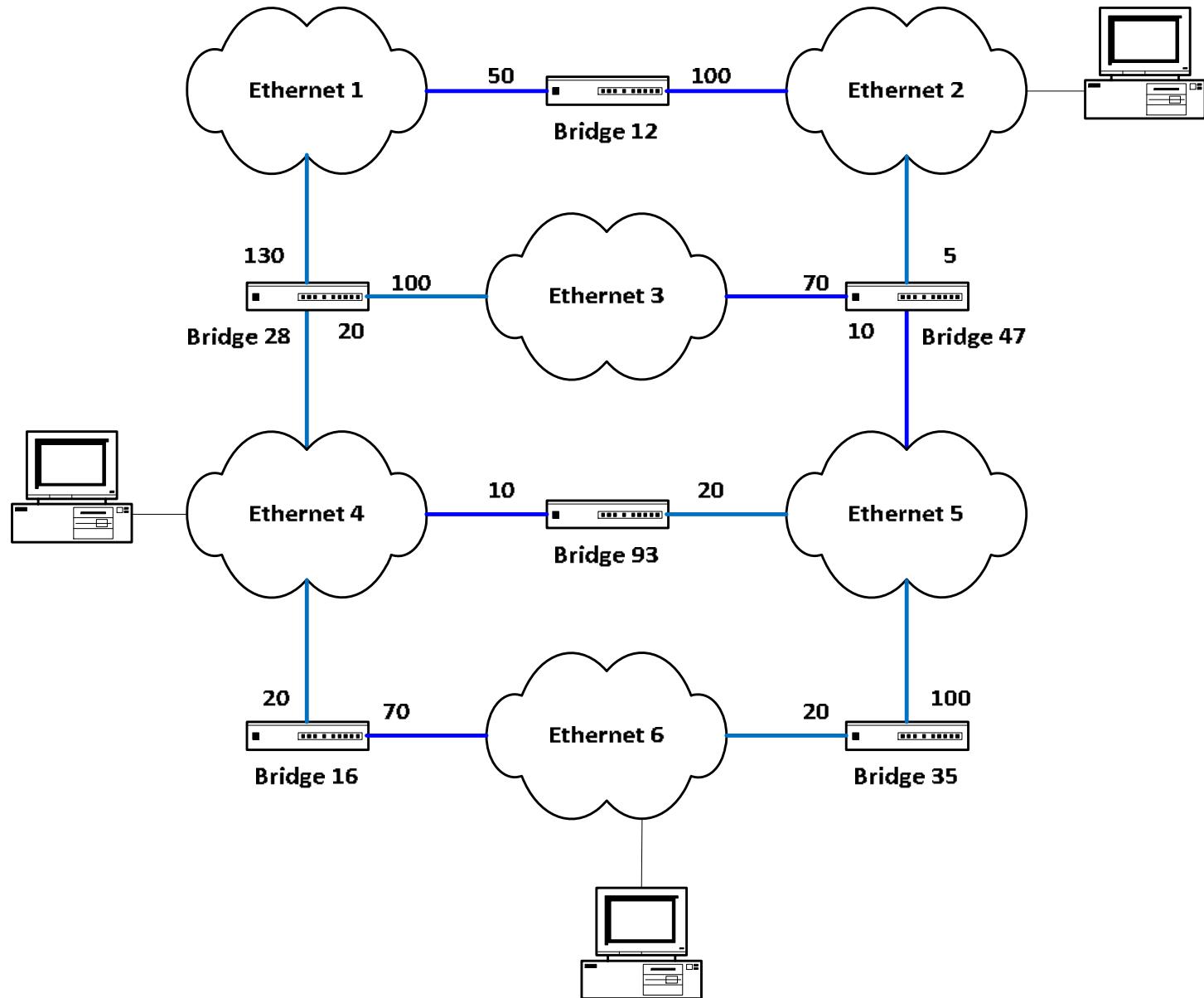


# Spanning Tree Example

Designated bridges

LAN Bridge

Eth1	
Eth 2	
Eth 3	
Eth 4	
Eth 5	
Eth 6	



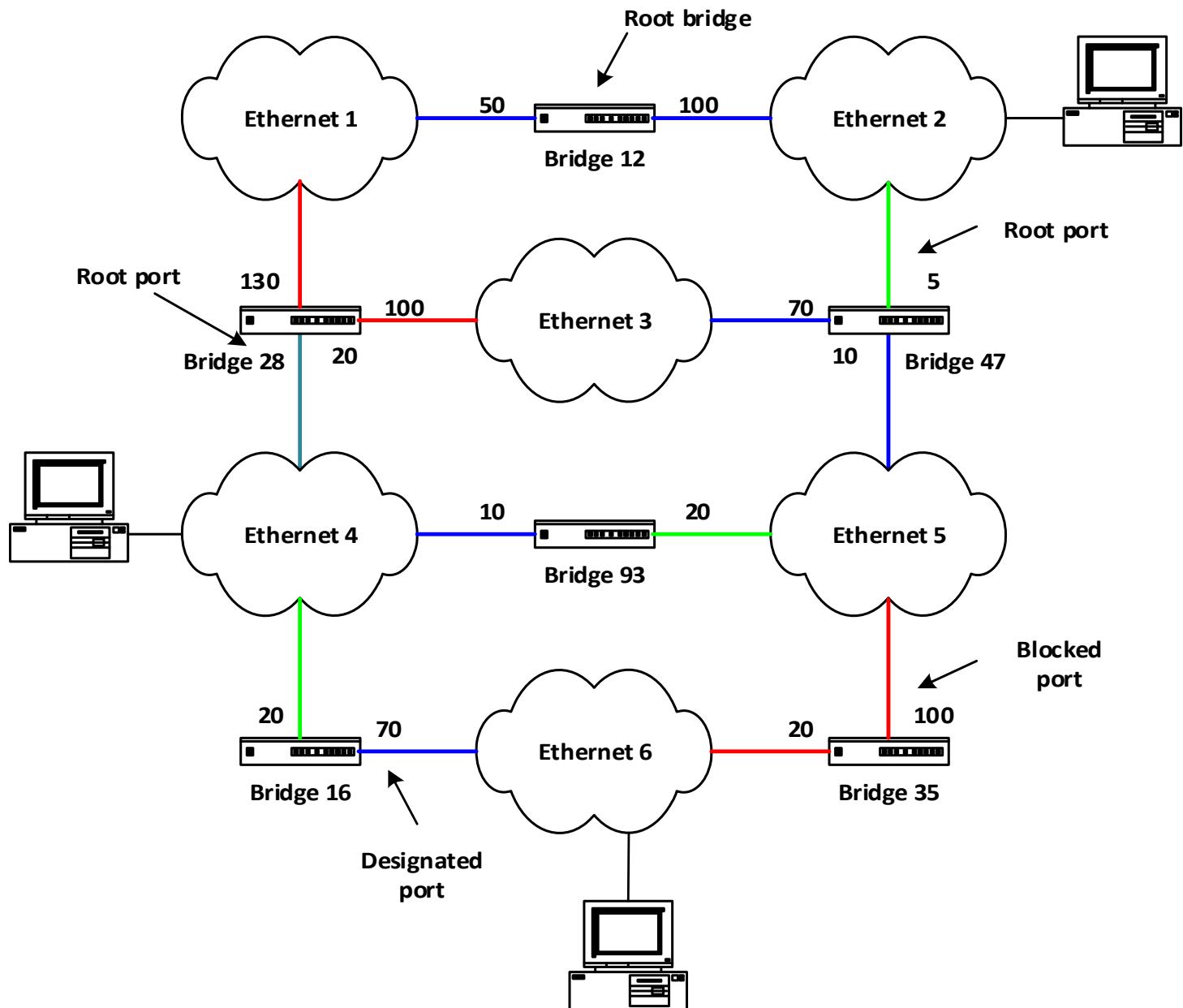


# Spanning Tree Example

Designated bridges

LAN Bridge

Eth1	12
Eth 2	12
Eth 3	47
Eth 4	93
Eth 5	47
Eth 6	16

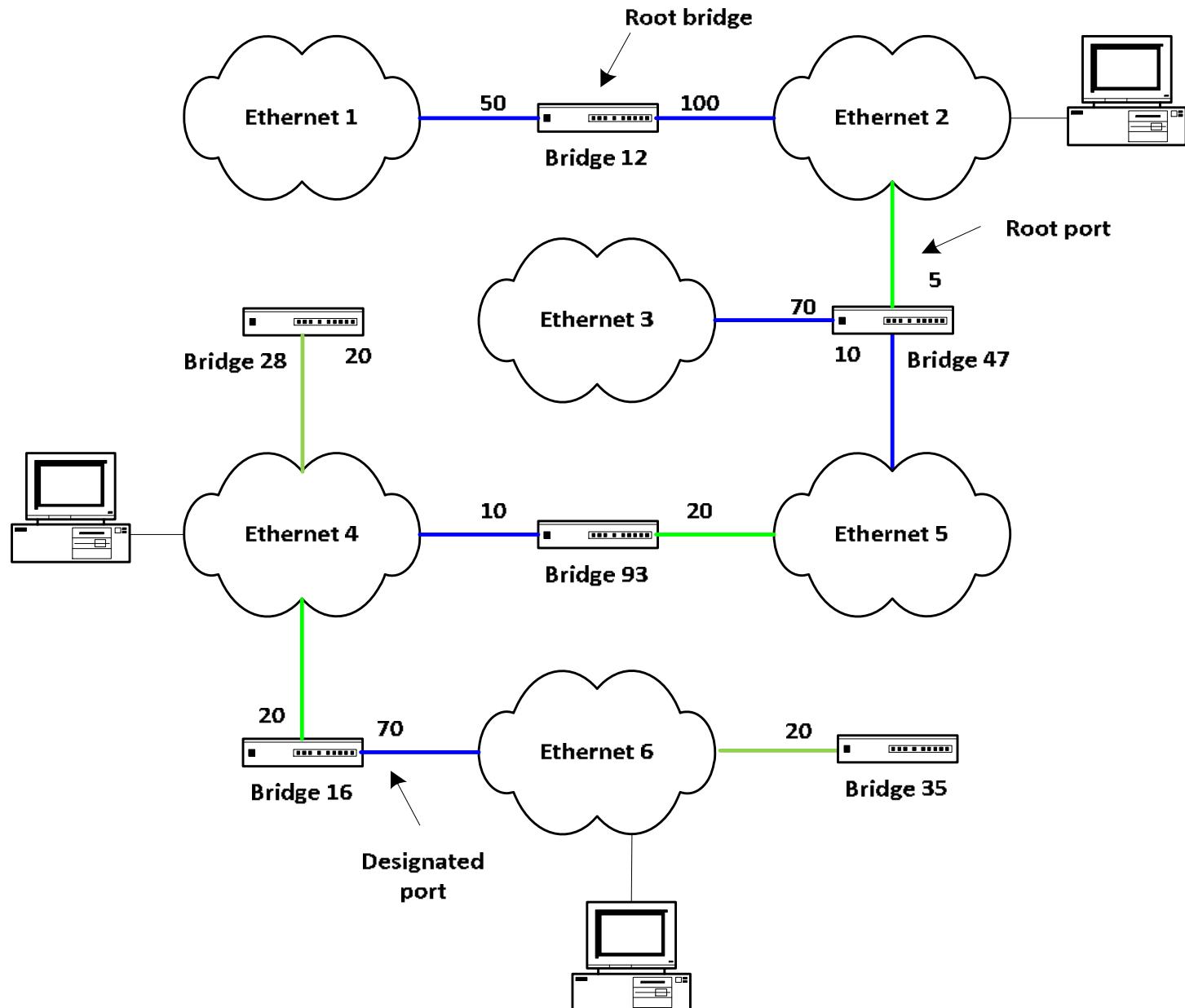


# Spanning Tree Example

Designated bridges

LAN Bridge

Eth1	12
Eth 2	12
Eth 3	47
Eth 4	93
Eth 5	47
Eth 6	16



# BPDUs (Bridge Protocol Data Units)

- To build and maintain the spanning tree, the bridges exchange special messages among them, known as BPDUs
- There exist two such types of messages:
  - ***Configuration*** and ***Topology Change Notification***

[+] 802.2: Address: 00-E0-B0-64-48-77 ---> 01-80-C2-00-00-00  
[+] LLC: Sap 0x42 ---> 0x42 (Command)  
[-] Bridge Protocol Data Unit

- Protocol ID: 0
- Version: 0
- Type: 0 (Configuration)
- Flags: 0x00
- Root ID: 40000800021EA1F1
  - Priority: 0x4000
  - MAC Address: 08-00-02-1E-A1-F1
- Root Path Cost: 100
- Bridge ID: 800000E0B0644876
  - Priority: 0x8000
  - MAC Address: 00-E0-B0-64-48-76
- Port ID: 0x8003
- Message Age: 1
- Max Age: 20
- Hello Time: 2
- Forward Delay: 15
- Frame Padding : (8 bytes)
- Calculate CRC: 0xb209dfee

destination	source	DSAP	SSAP	BPDU
-------------	--------	------	------	------

- Destination – **multicast address assigned to every bridge**
- Source – MAC address of the port sending the BPDU
- DSAP = SSAP = 01000010 = 42 hex

# Configuration BPDU

- The spanning tree configuration is done using Conf - BPDU (configuration messages)

```

+ [ ] 802.2: Address: 00-E0-B0-64-48-77 --->01-80-C2-00-00-00
+ [ ] LLC: Sap 0x42 ---> 0x42 (Command)
- [ ] Bridge Protocol Data Unit
  [ ] Protocol ID: 0
  [ ] Version: 0
  [ ] Type: 0 (Configuration)
  [ ] Flags: 0x00
  [ ] Root ID: 40000800021EA1F1
  [ ] Priority: 0x4000
  [ ] MAC Address: 08-00-02-1E-A1-F1
  [ ] Root Path Cost: 100
  [ ] Bridge ID: 800000E0B0644876
  [ ] Priority: 0x8000
  [ ] MAC Address: 00-E0-B0-64-48-76
  [ ] Port ID: 0x8003
  [ ] Message Age: 1
  [ ] Max Age: 20
  [ ] Hello Time: 2
  [ ] Forward Delay: 15
  [ ] Frame Padding : (8 bytes)
  [ ] Calculate CRC: 0xb209dfee

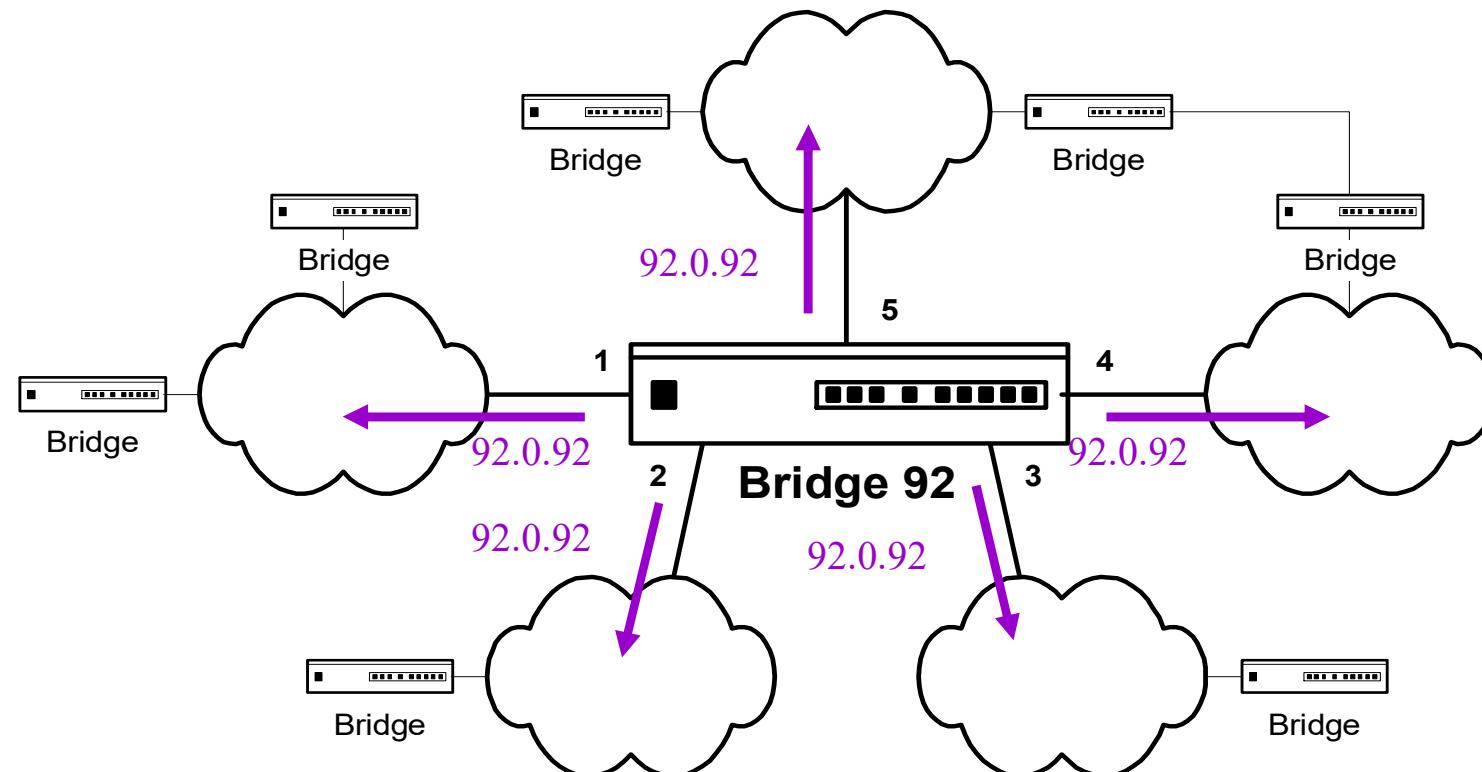
```

- More important fields:

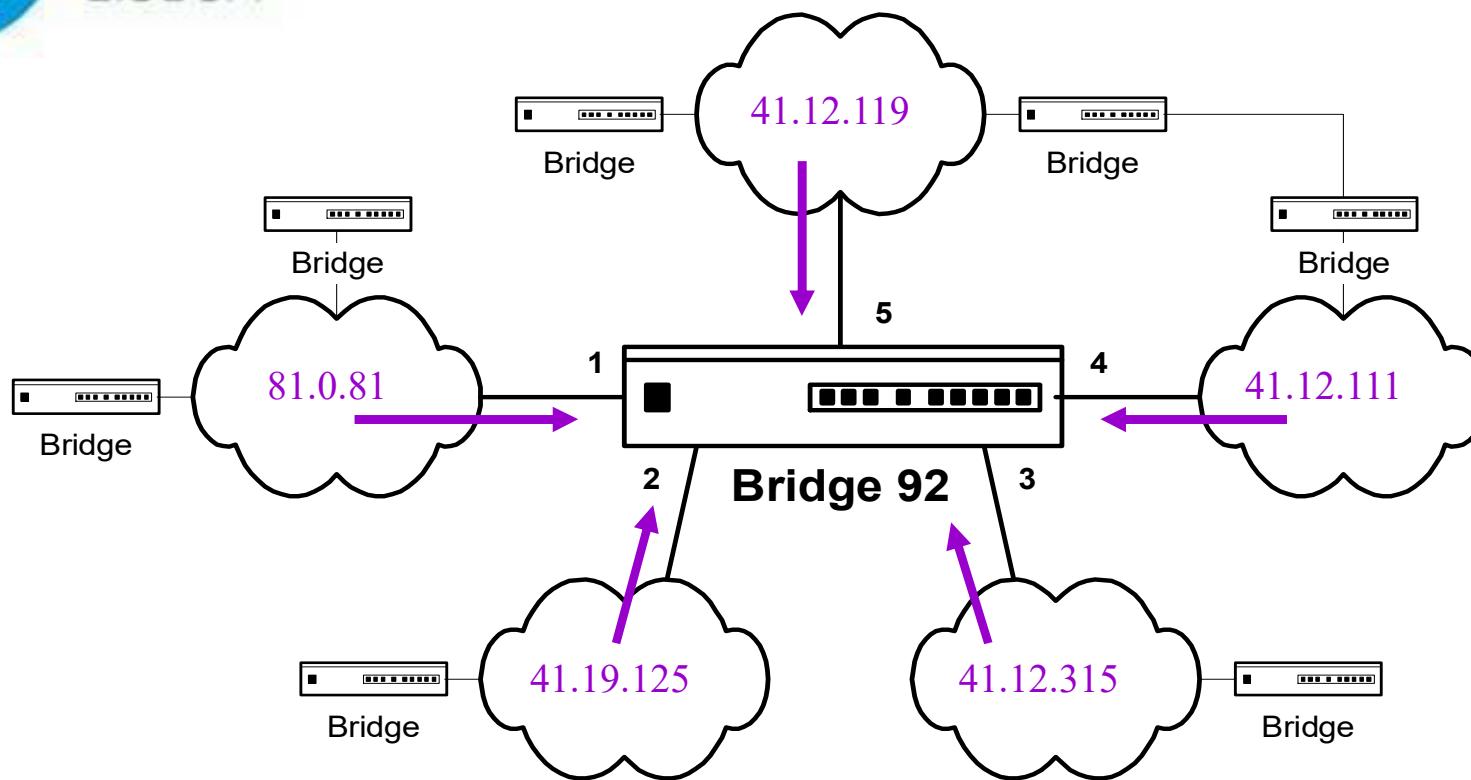
- Root ID: present estimate of the root bridge address
- Root Path Cost: present estimate of the path cost to the root bridge
- Bridge ID: address of the bridge sending the configuration message
- Port ID: address of the port sending the configuration message

# Spanning Tree Construction

- Each bridge initially assumes it is the root bridge (making Root Path Cost = 0); it sends configuration messages in all its ports



# Spanning Tree Construction



Best received messages in bridge 92 so far

Bridge 92 estimates:

- Root bridge = 41
- Root port = 4
- Root path cost = 12 + 1

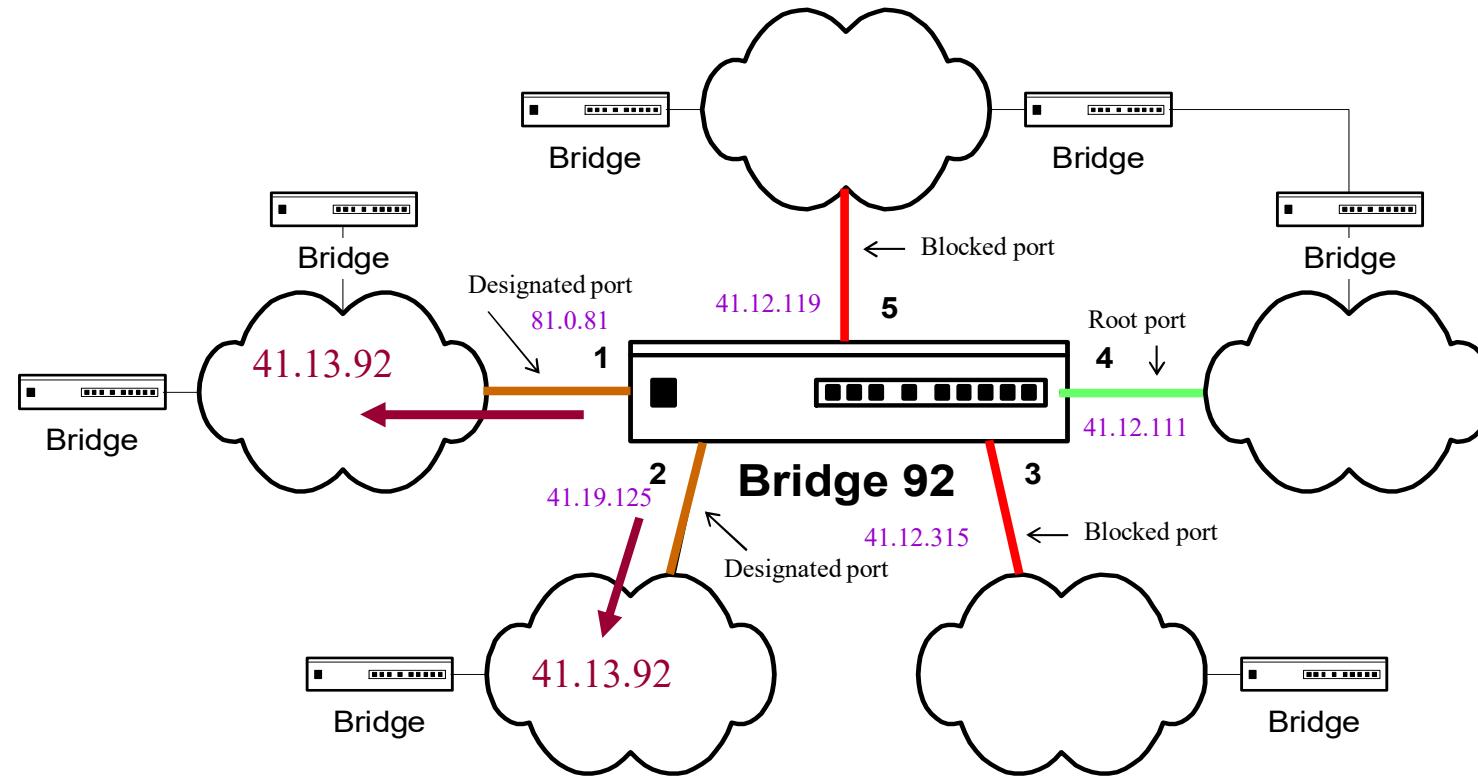
## Configuration Message Order

- A configuration message  $C_1$  is said to be “better” than  $C_2$  if:
  - $C_1$  Root ID is lower than that of  $C_2$
  - if the Root IDs are identical, the Root Path Cost of  $C_1$  is lower than that of  $C_2$
  - being identical the Root ID and Root Path Cost, the Bridge ID of  $C_1$  is lower than that of  $C_2$
  - being identical the Root ID, Root Path Cost and Bridge ID, the Port ID of  $C_1$  is lower than that of  $C_2$

Root ID	Root Path Cost	Bridge ID	Port ID
41	12	111	2
41	12	111	4
41	12	119	1
41	19	125	3
81	0	81	2

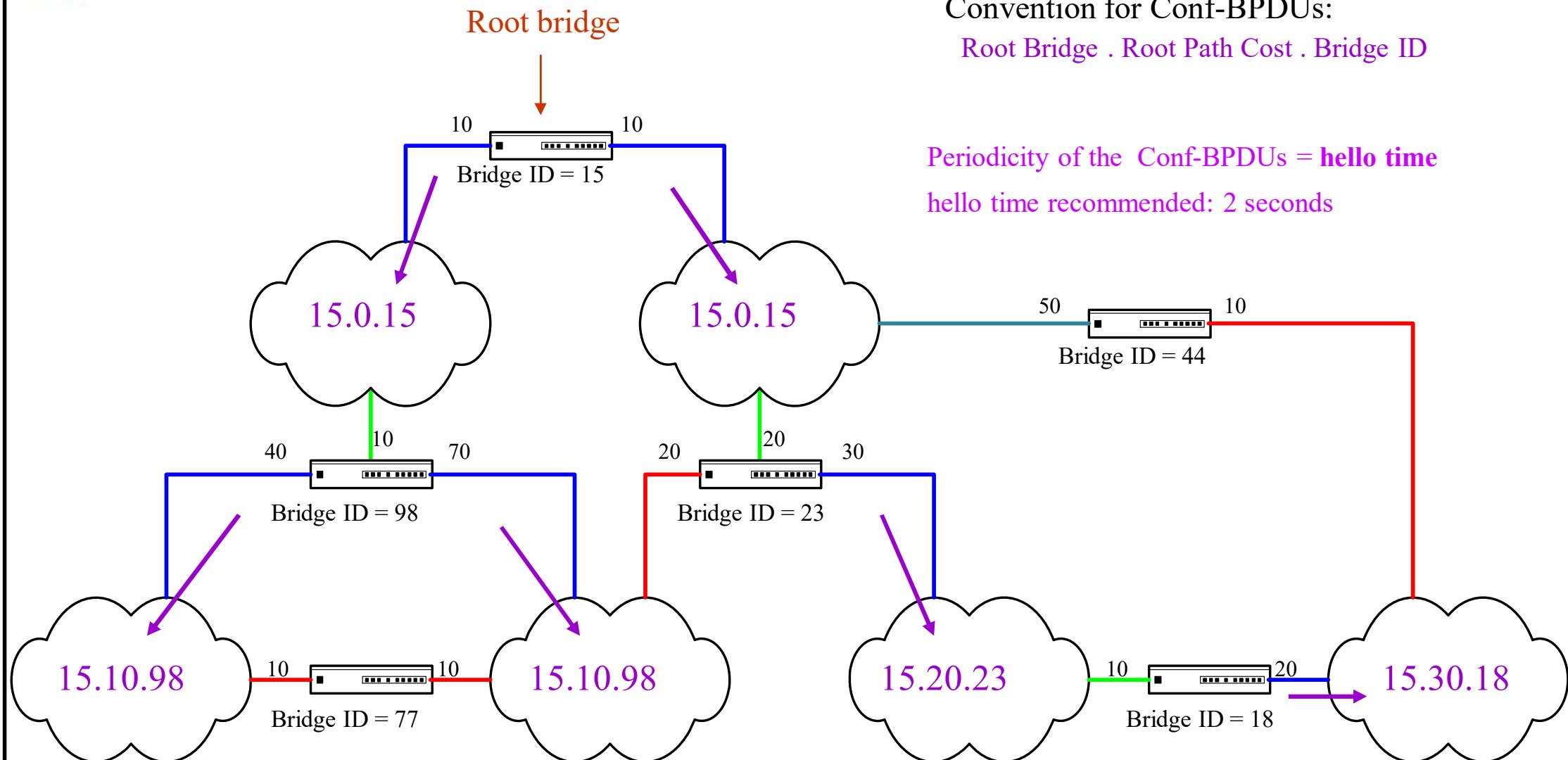
# Spanning Tree Construction

TPC: Prob. 16



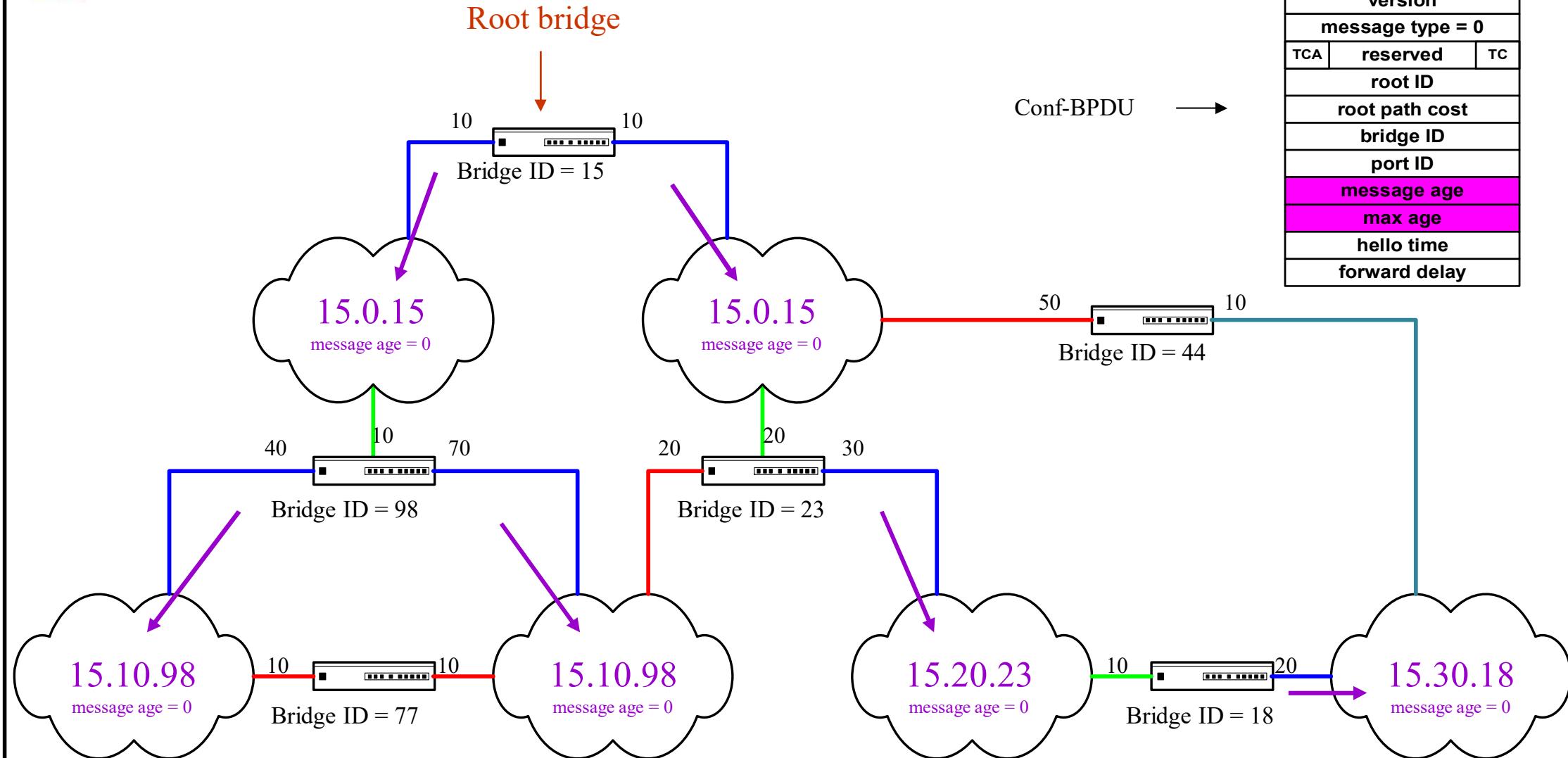
messages sent by Bridge 92 - 41.13.92

# Spanning Tree Maintenance



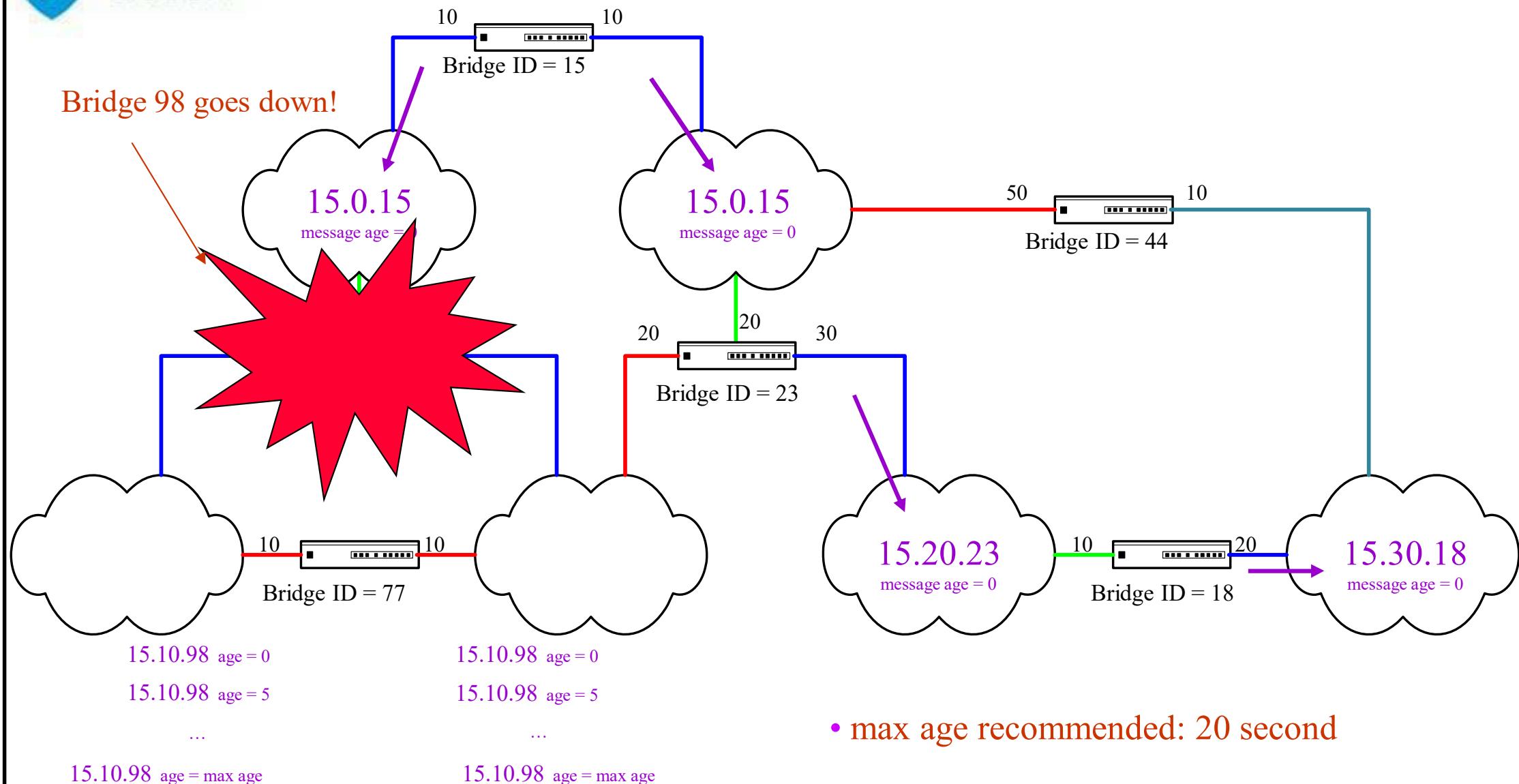


# Problems in Bridges or LANs

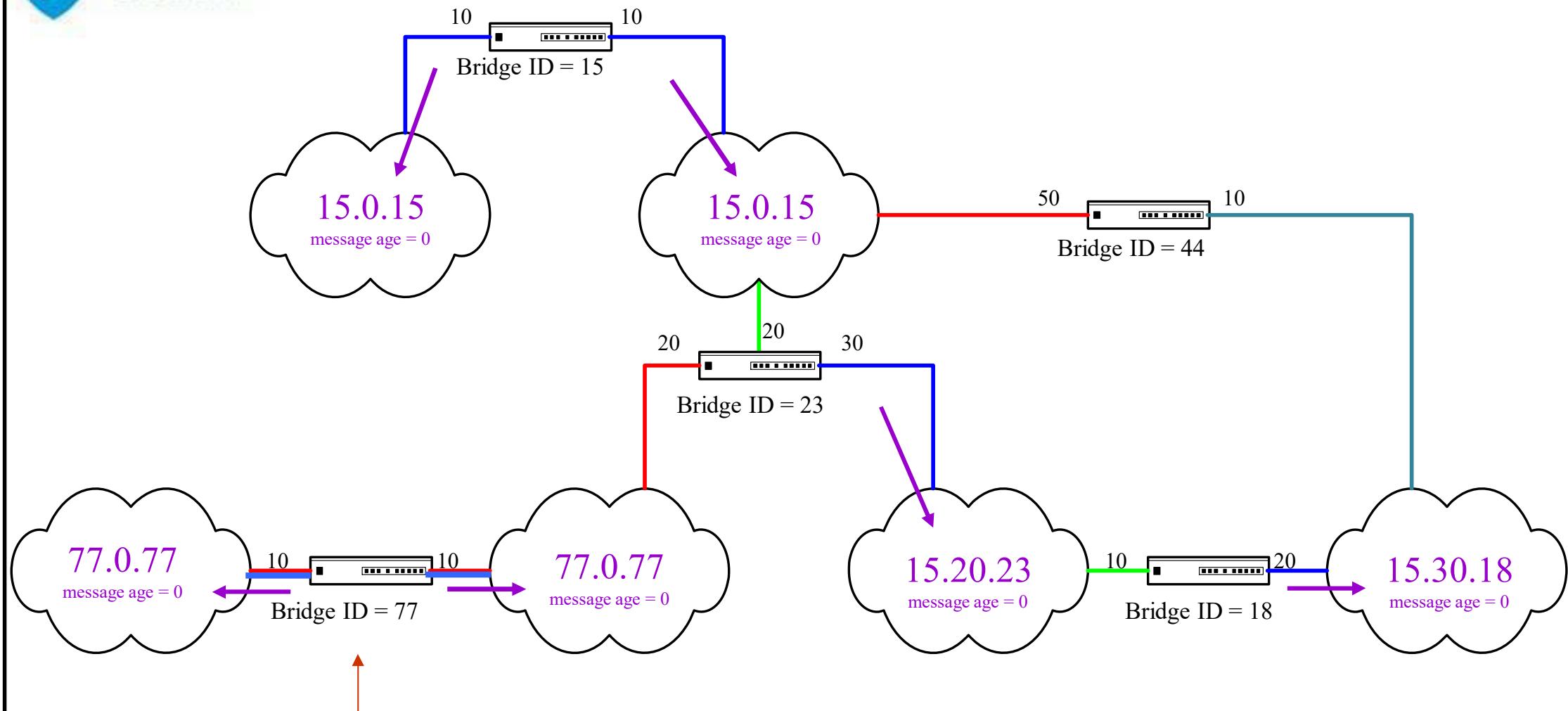


# Problems in Bridges or LANs

Bridge 98 goes down!

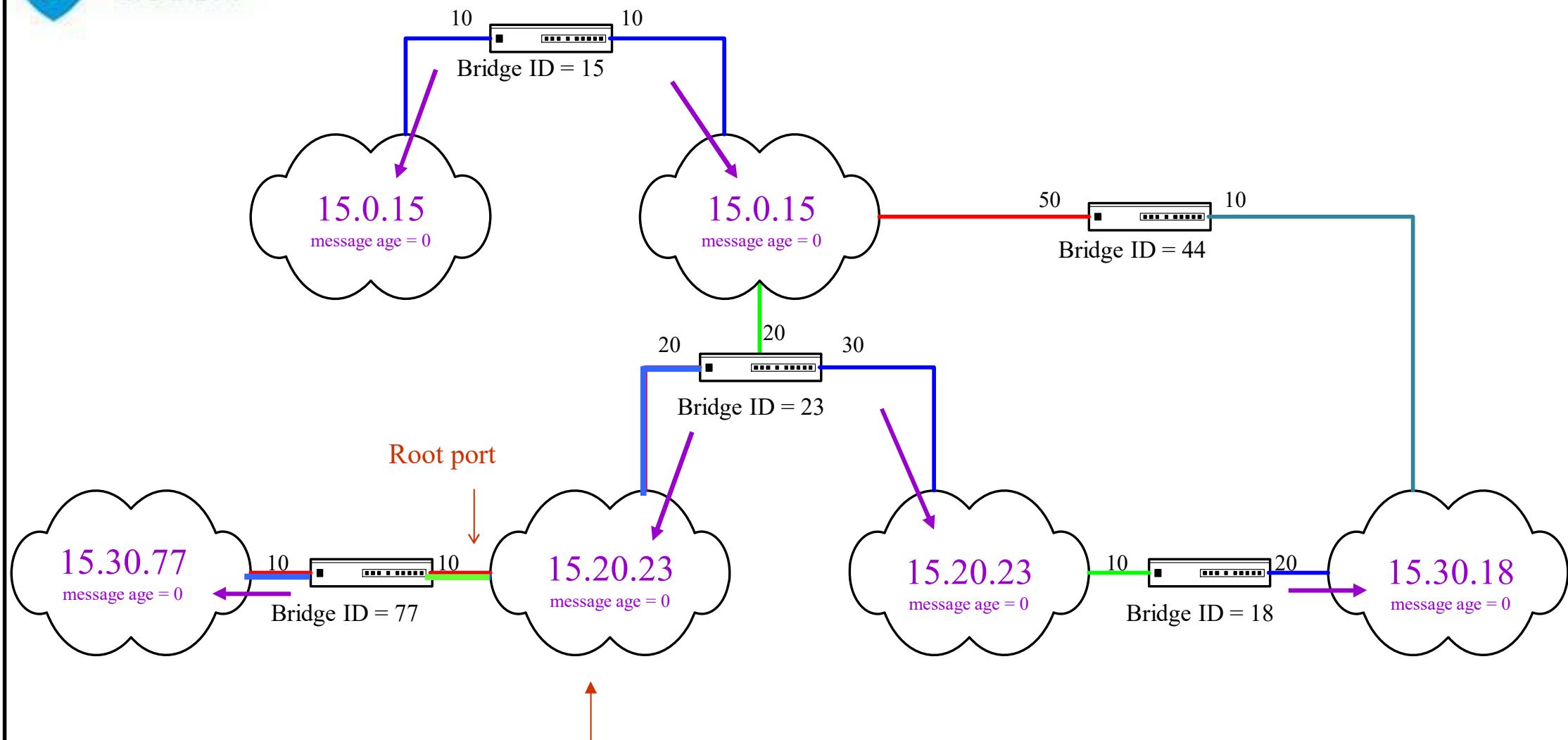


# Problems in Bridges or LANs



Bridge 77 assumes it is the root

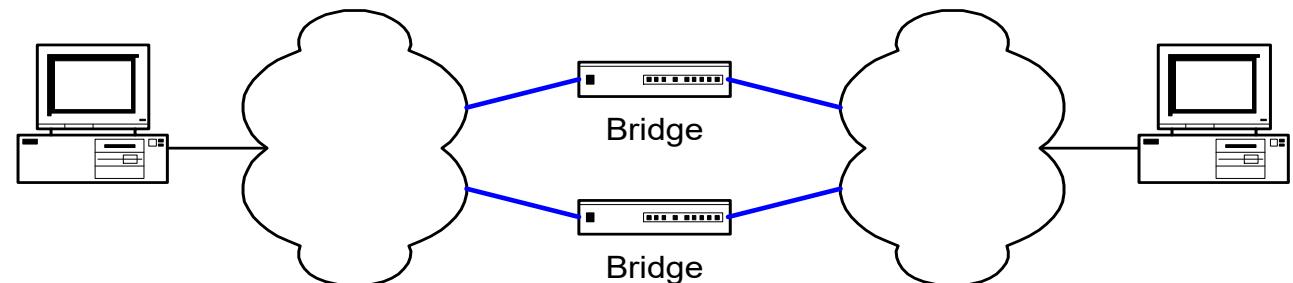
# Problems in Bridges or LANs



Bridge 23 becomes designated in this LAN

# Temporary Cycles/Connectivity Loss

- After the network topology change:
  - There may be a **temporary loss of connectivity** if one previously blocked port has not yet “realized” that it should become active in the new topology
  - There may be **temporary cycles** if a previously active port has not yet “realized” that it should become blocked in the new topology
- To minimize the probability of temporary cycles appearing, the bridges must wait some time before allowing one of their ports moving from the blocked state to an active state; the waiting time is controlled by the parameter: **forward delay**



# Configuration Messages Format

# bytes

2	protocol identifier
1	version
1	message type
1	TCA   reserved   TC
8 root ID	
4	cost of path to root
8	bridge ID
2	port ID
2	message age
2	max age
2	hello time
2	forward delay



An Algorithm for Distributed  
Computation of a Spanning Tree  
in an Extended LAN

Radia Perlman

Digital Equipment Corporation  
1925 Andover St., Tewksbury MA 01876

Abstract

A protocol and algorithm are given in which bridges in an extended Local Area Network of arbitrary topology compute, in a distributed fashion, an acyclic spanning subset of the network.

The algorithm converges in time proportional to the diameter of the extended LAN, and requires a very small amount of memory per bridge, and communications bandwidth per LAN, independent of the total number of bridges or the total number of links in the network.

Algorhyme

*I think that I shall never see  
A graph more lovely than a tree.*

*A tree whose crucial property  
Is loop-free connectivity.*

*A tree which must be sure to span  
So packets can reach every LAN.*

*First the Root must be selected.  
By ID it is elected.*

*Least cost paths from Root are traced.  
In the tree these paths are placed.*

*A mesh is made by folks like me  
Then bridges find a spanning tree.*

Introduction

Local area networks are limited in geography, traffic, and number of stations. A single local area network will often not meet the needs of an organization for these reasons. Conventional LAN interconnection mechanisms, for instance XNS [1], Sytek[2], IBM1[3], IBM2[4], DNA[5] require cooperation from the stations with compatible protocols layered above the protocol necessary to connect to a single LAN.

An approach that is transparent to stations, and thus allows a station to participate in an extended LAN with no modification, is presented in [6],[7], and [8]. In this approach, a bridge connected to two or more links will listen "promiscuously" to all packets transmitted on each of its links, and forward packets received on one of the links onto the others. A bridge also learns of the location of stations relative to itself, so that it will not forward traffic for a station onto a link unnecessarily.

This approach assumes that the topology is a tree (loop-free). However, requiring a topology to be loop-free means there are no backup paths in the case of bridge or LAN failures. Also, because the technology allows network growth so easily, it might be difficult to prevent someone from adding a bridge and creating a loop. A loop in the topology might cause severe performance degradation in the entire extended network due to congestion caused by infinitely circulating packets. It is undesirable to have a network that can be brought down so easily, merely by plugging a cable into the wrong place.



Radia Perlman

## Algoryme

I think that I shall never see  
A graph more lovely than a tree.

A tree whose crucial property  
Is **loop-free connectivity**.

A tree that must be sure to span  
So packets can reach every LAN.

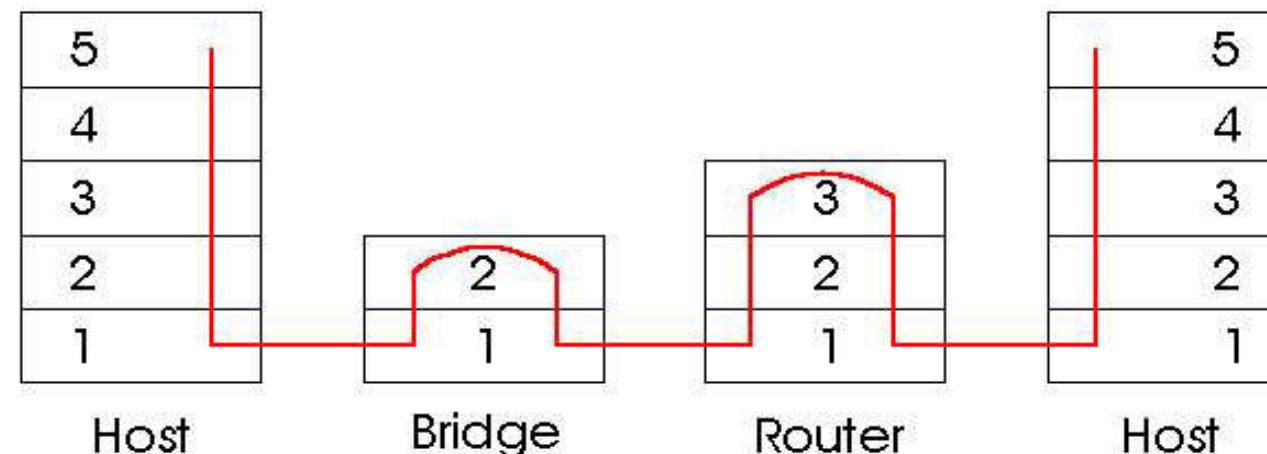
**First the root must be selected.**  
By ID, it is elected.

**Least-cost paths from root** are traced.  
In the tree, these paths are placed.

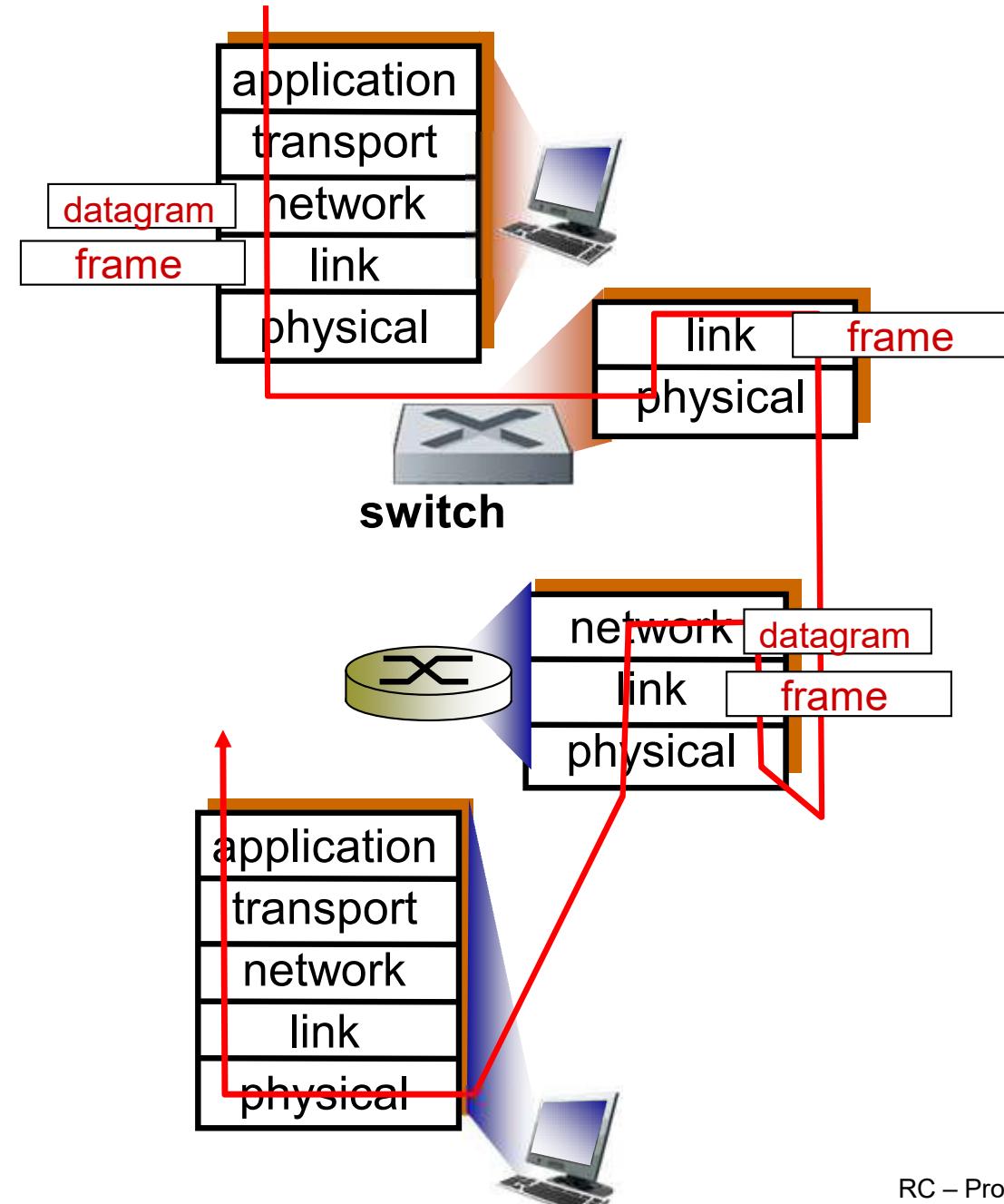
A mesh is made by folks like me,  
Then the bridge finds a **spanning tree**.

# Switches vs. Routers

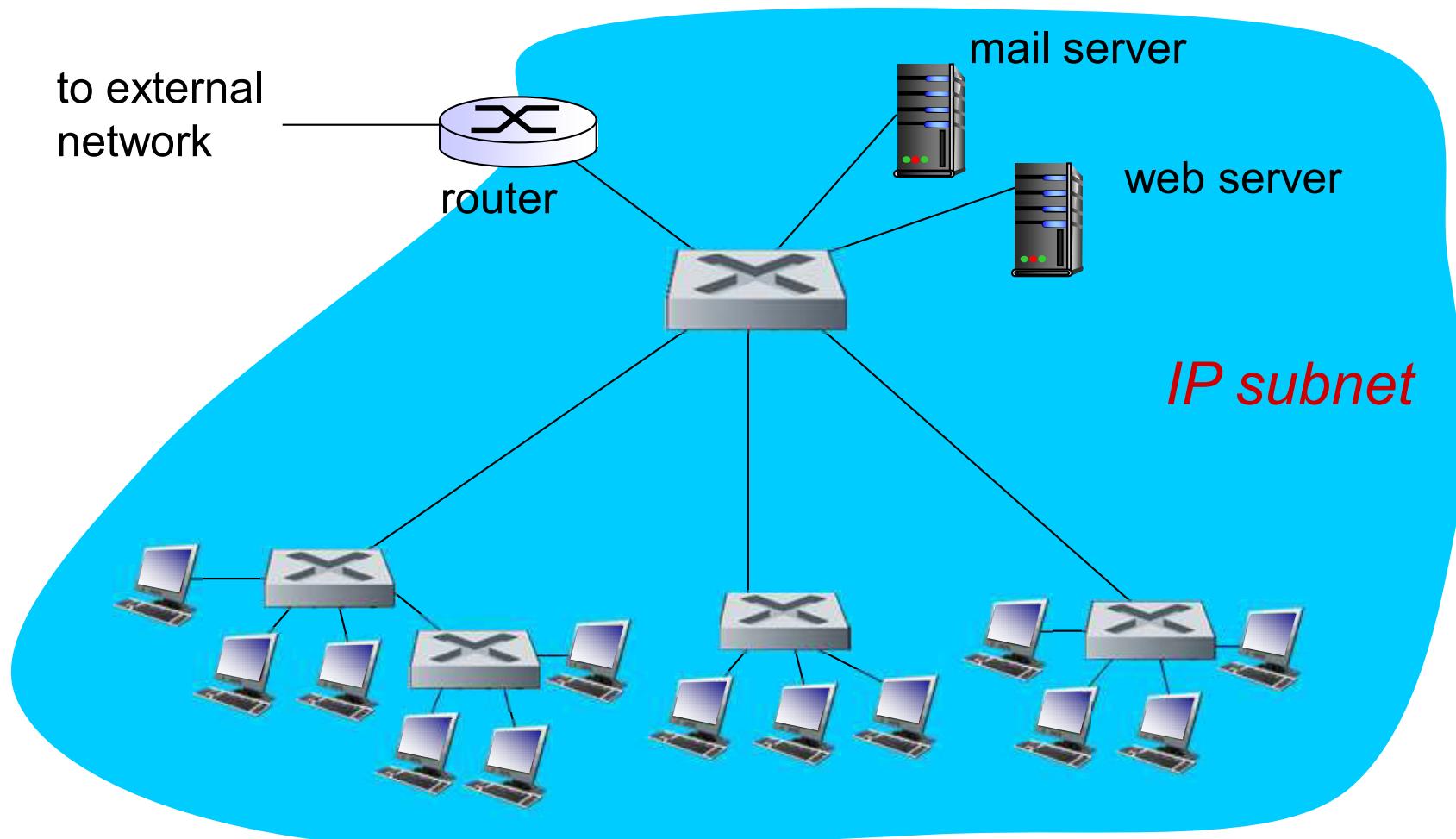
- Both store-and-forward devices:
  - Routers: network layer devices (examine network layer headers);
  - Switches are link layer devices.
- Routers maintain routing tables, implement routing algorithms;
- Switches maintain switch tables, implement filtering, learning algorithms.



# Switches vs. Routers



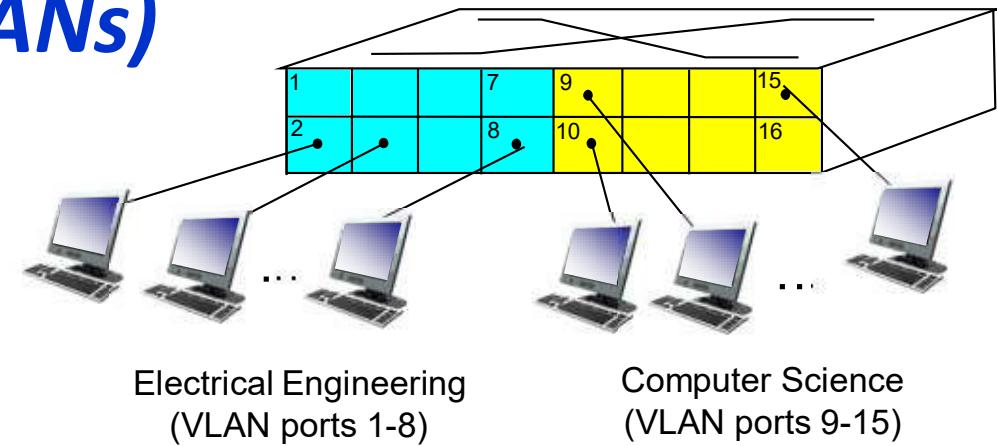
# Institutional Network



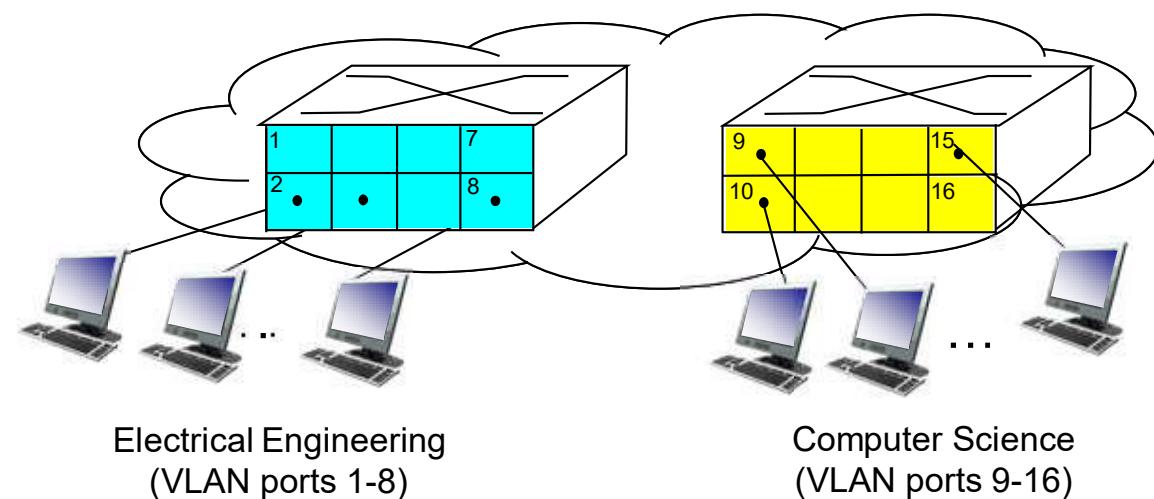
# Virtual LANs (VLANs)

## Virtual Local Area Network

*switch(es) supporting VLAN capabilities can be configured to define multiple **virtual** LANS over a single physical LAN infrastructure.*

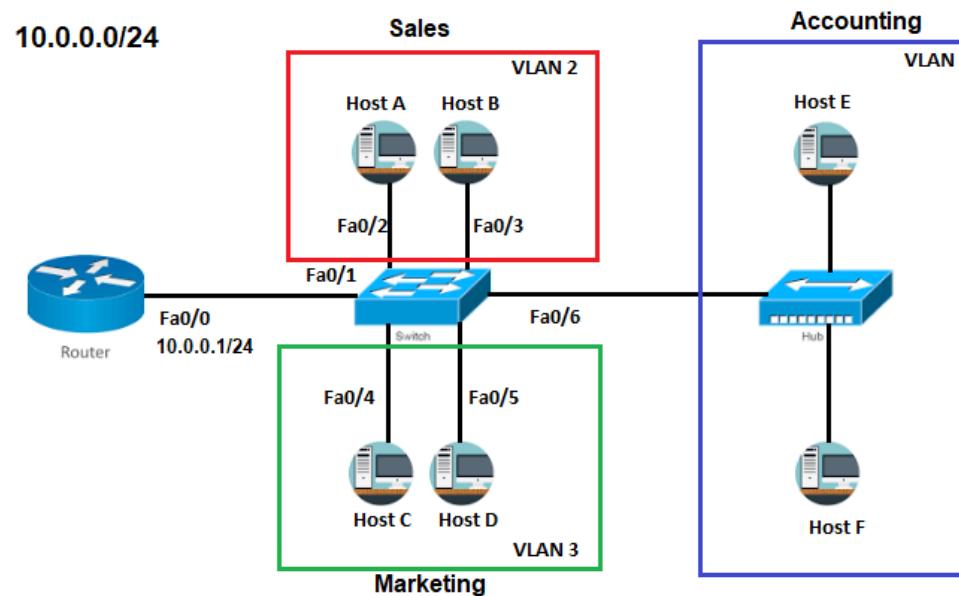


Port-based VLAN: switch ports grouped (by switch management software) so that one single physical switch operates as multiple virtual switches



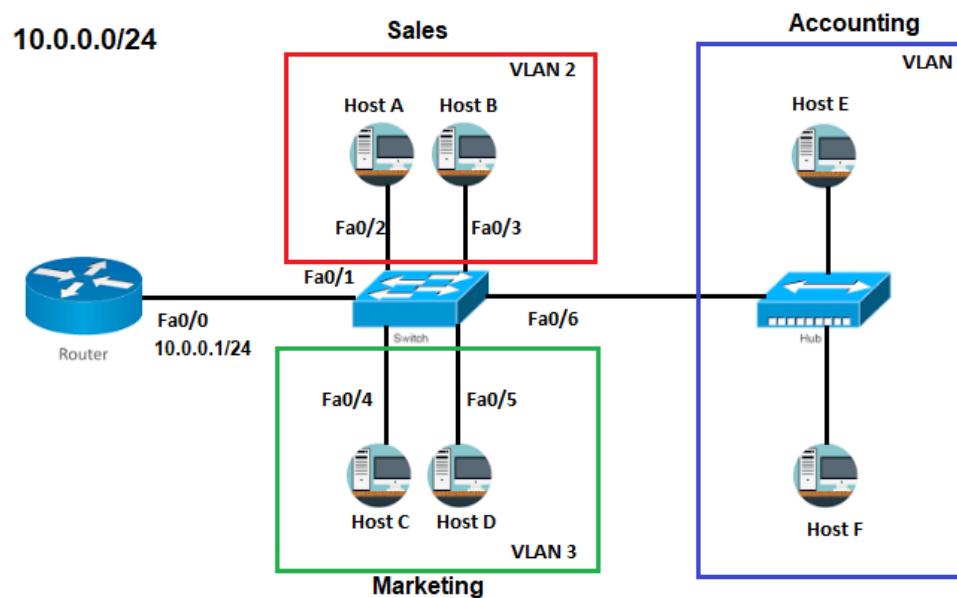
# Virtual LANs (VLANs)

- A VLAN is a type of local area network that does not have its own dedicated physical infrastructure.  
Instead, it uses another LAN physical infrastructure to carry its traffic.
- The traffic is encapsulated so that a **number of logically separate VLANs can be carried by the same physical LAN**.



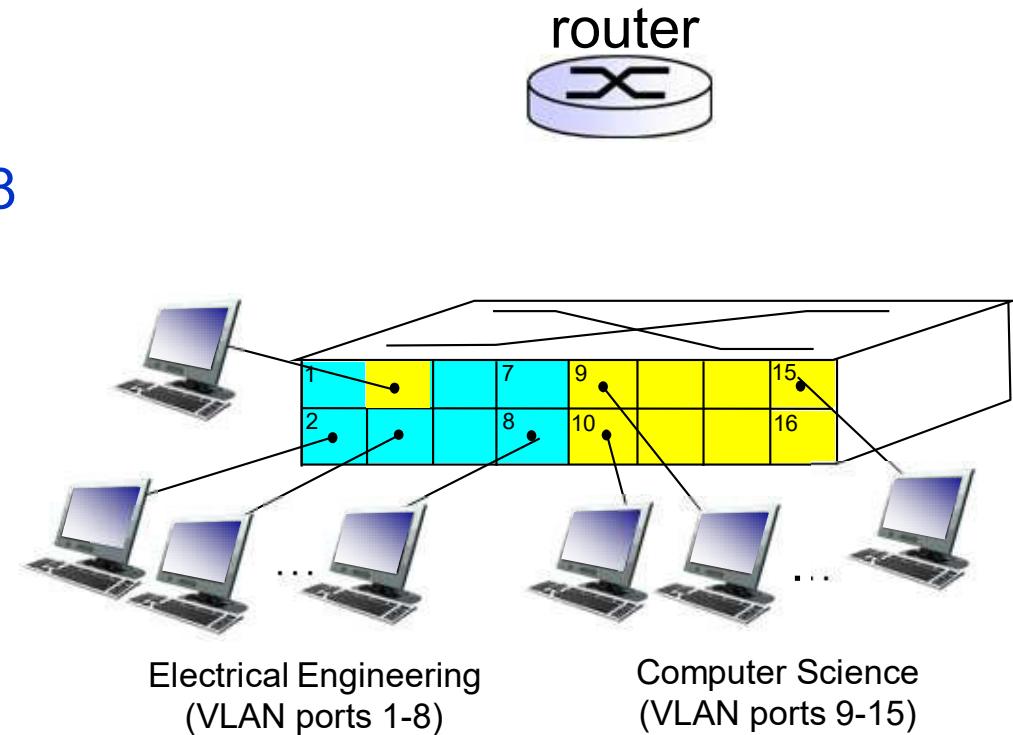
# *Virtual LANs (VLANs)*

- A VLAN divides a physical LAN into parts, and breaks the network into **different broadcast domains** (with a limited, reasonable range).
- Limiting the broadcast can prevent switches from wasting bandwidth to forward messages to unnecessary ports.
- Since **hosts in different VLANs cannot communicate with each other** directly, VLAN can also provide increased security.

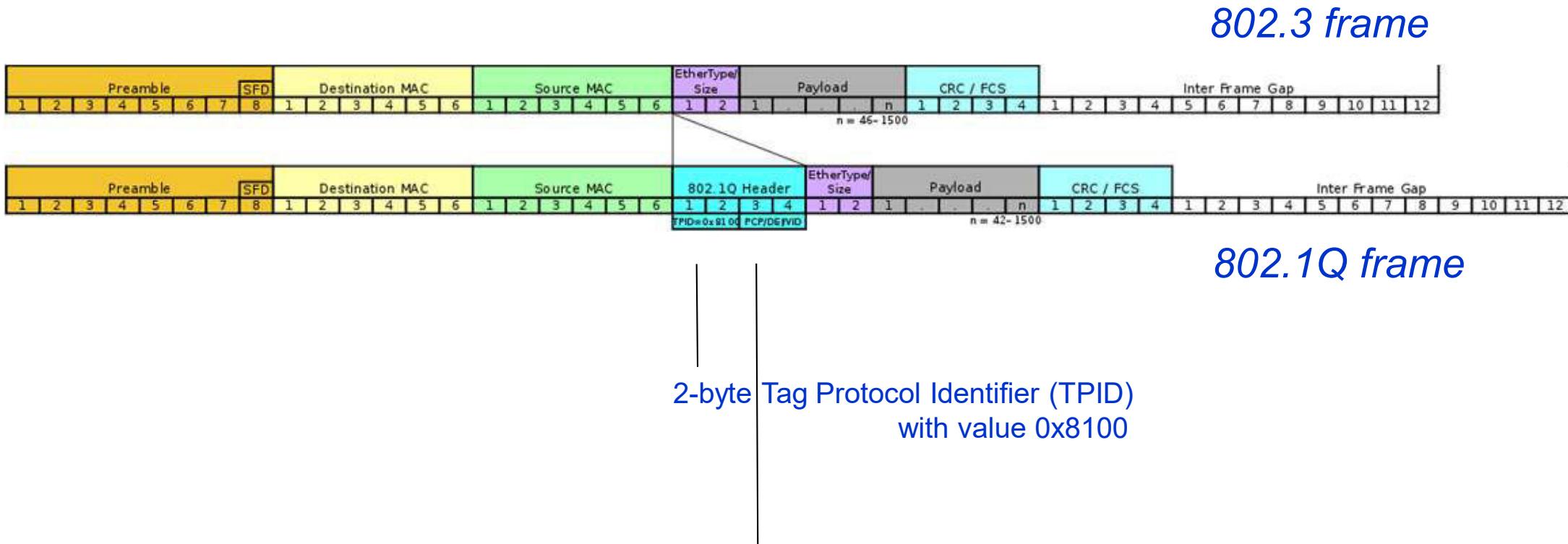


## Port-based VLAN

- **Traffic isolation:** frames to/from ports 1-8 can only reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- **Dynamic membership:** ports can be dynamically assigned among VLANs
- **Forwarding between VLANs:** done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers



# 802.1Q VLAN Frame Format



Tag Control Information (TCI):

- Priority code point (PCP) – 3 bit priority field (like IP TOS)
- Drop eligible indicator (DEI) – 1 bit: eligible to drop if congestion
- VLAN identifier (VID) – **12 bit VLAN ID field**

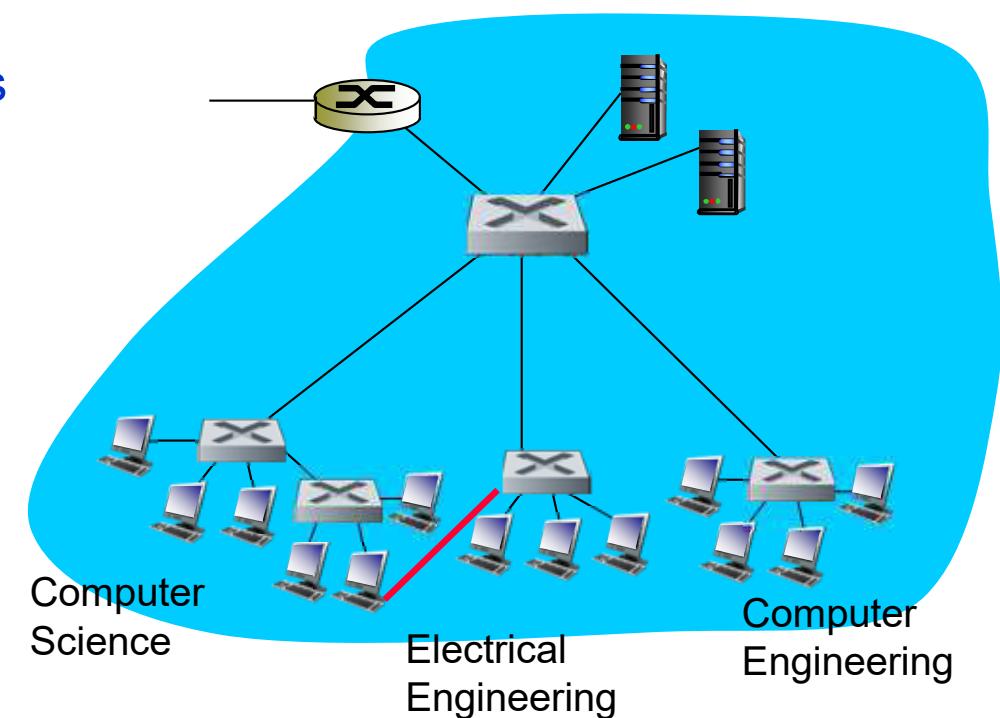
VID = 0x000 indicates that the frame does not carry a VLAN ID

# *Virtual LANs (VLANs)*

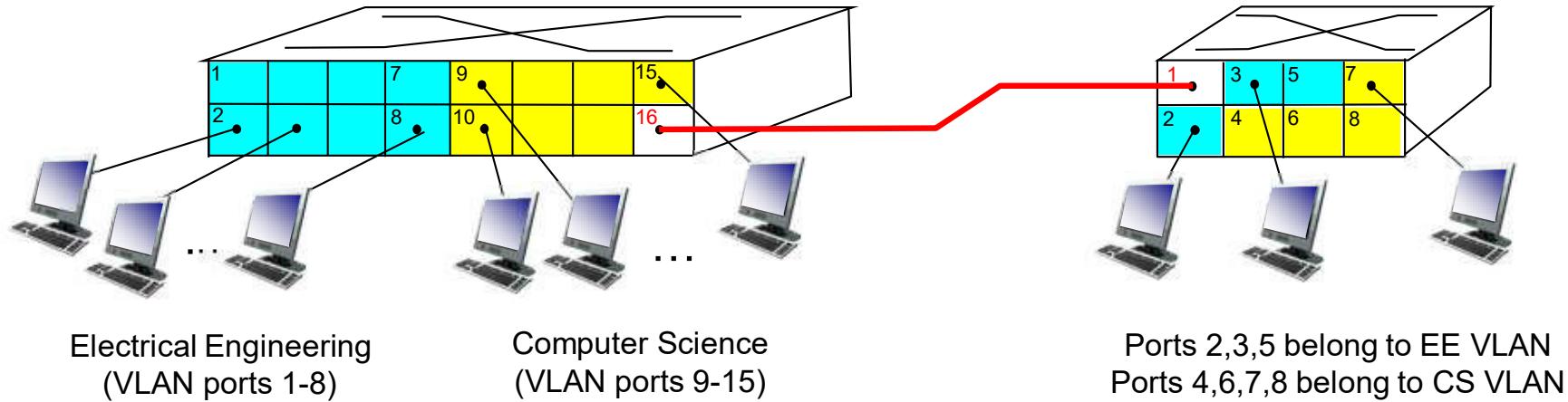
What if CS user moves office to EE, but wants connect to CS switch?

In fact, there should be a single broadcast domain:

- all layer-2 broadcast traffic (ARP, DHCP, broadcast to unknown destinations) must cross the entire LAN
- security/privacy, efficiency issues



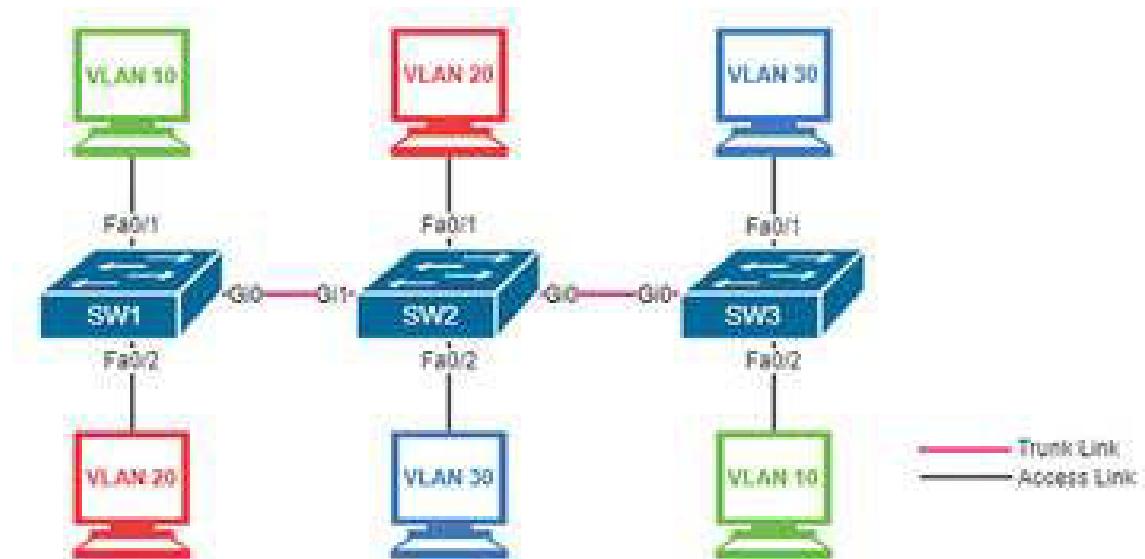
# VLANS Spanning Multiple Switches



- **Trunk port:** carries frames between VLANS defined over multiple physical switches
  - frames forwarded within VLAN between switches can't be regular 802.3 frames (must carry VLAN ID info)
  - 802.1q protocol adds/removes additional header fields for frames forwarded between trunk ports

# VLANs Spanning Multiple Switches

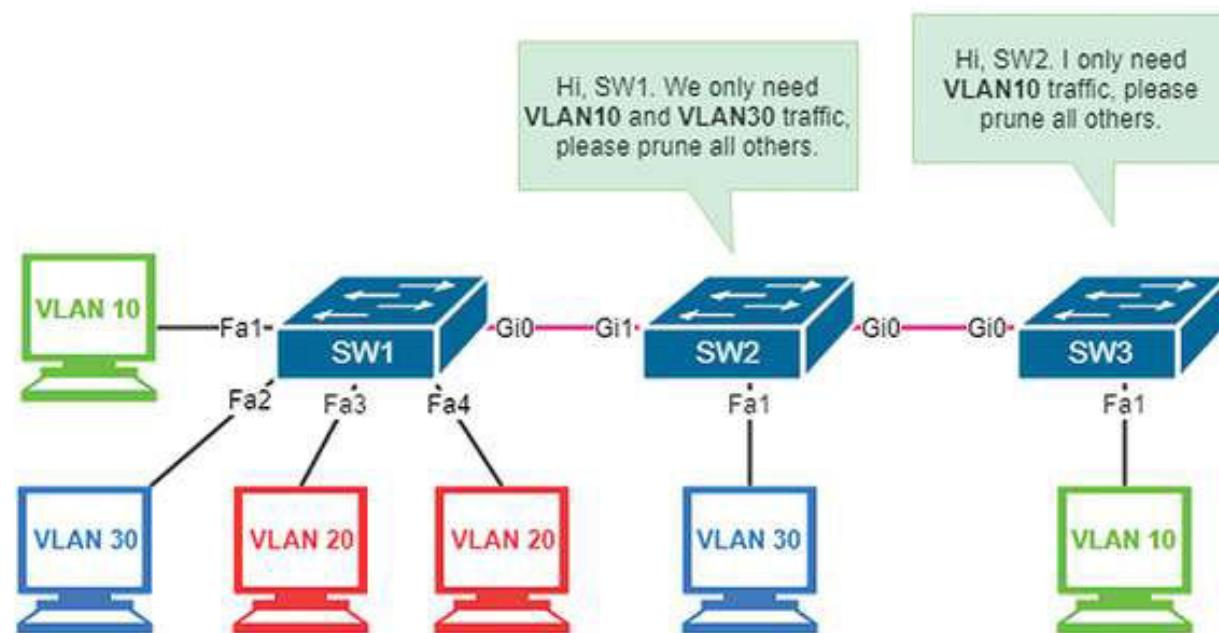
- A host can be assigned to any VLAN regardless of geographic location, as long as the **switches are connected by trunk links**.
- VLANs need to be created on all switches on the path through which VLAN traffic passes.
  - Figure: VLAN20 created on SW1 and SW2 (for red hosts to communicate), VLAN 30 on SW2 and SW3 (for blue hosts to communicate) and VLAN10 on the three switches (for green hosts to communicate).



# VLANS Spanning Multiple Switches

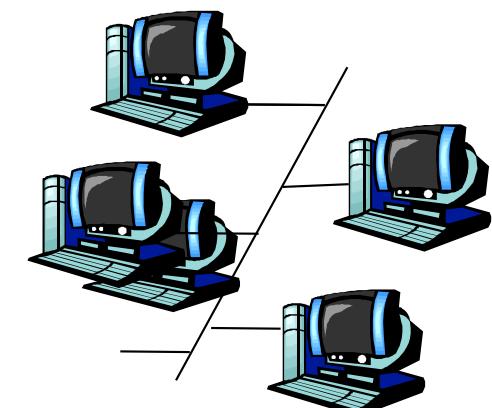
## Pruning:

- The pruning feature can automatically block unnecessary traffic that passes through trunk links, to avoid wasting bandwidth.
  - Figure: SW3 only needs VLAN10 traffic → SW3 notifies SW2 to prune all other VLAN traffic.  
SW2 only needs VLAN30 traffic → it combines with SW3 request and tells SW1 to prune all VLANs other than VLAN10 and VLAN30.

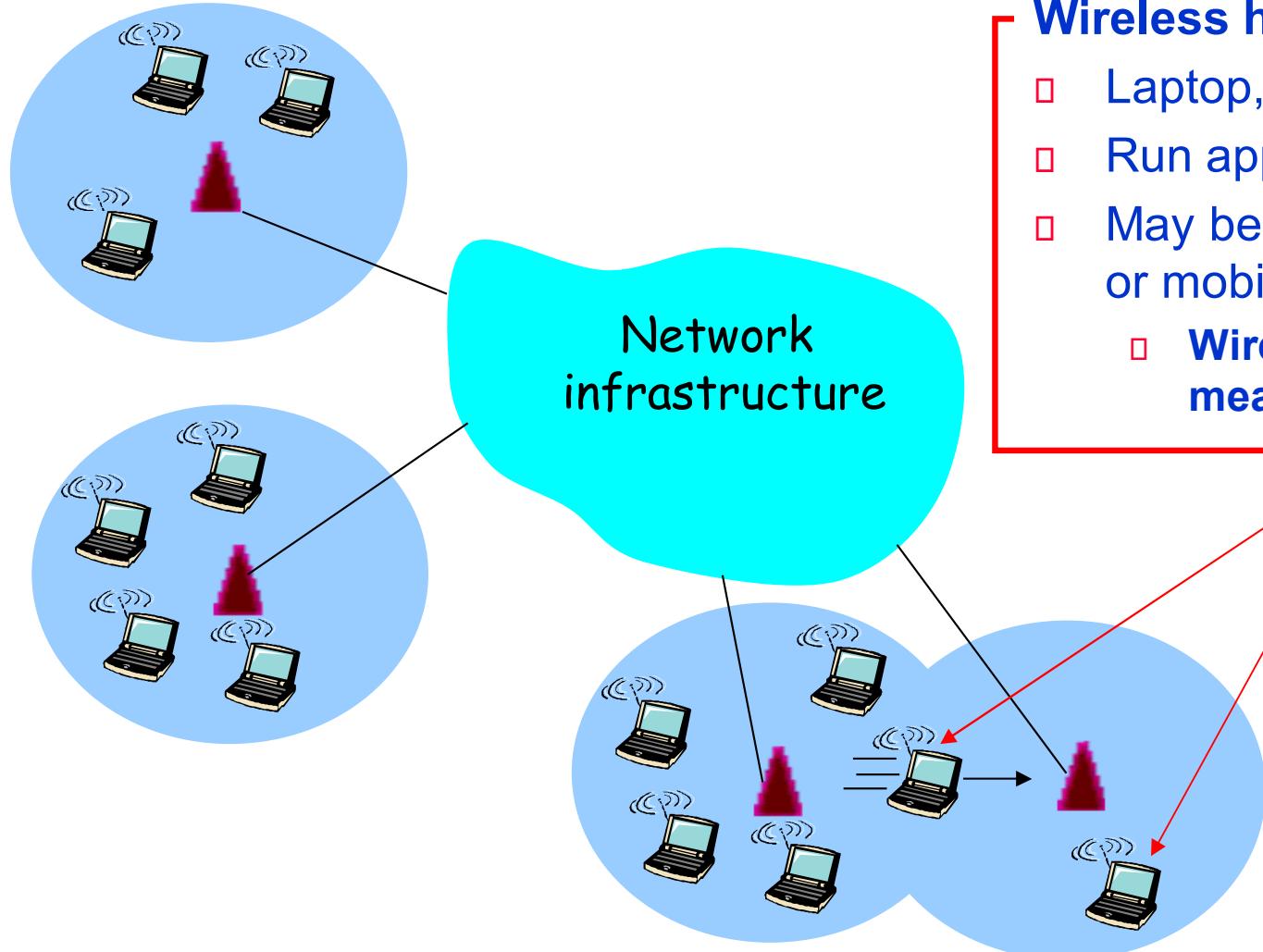


## *Outline*

- Link-layer Addressing
- Introduction and services
- Error detection and correction
- Multiple access protocols
- Ethernet
- Link-layer switches
- IEEE 802.11 Wireless LANs
- Framing



# *Elements of a Wireless Network*

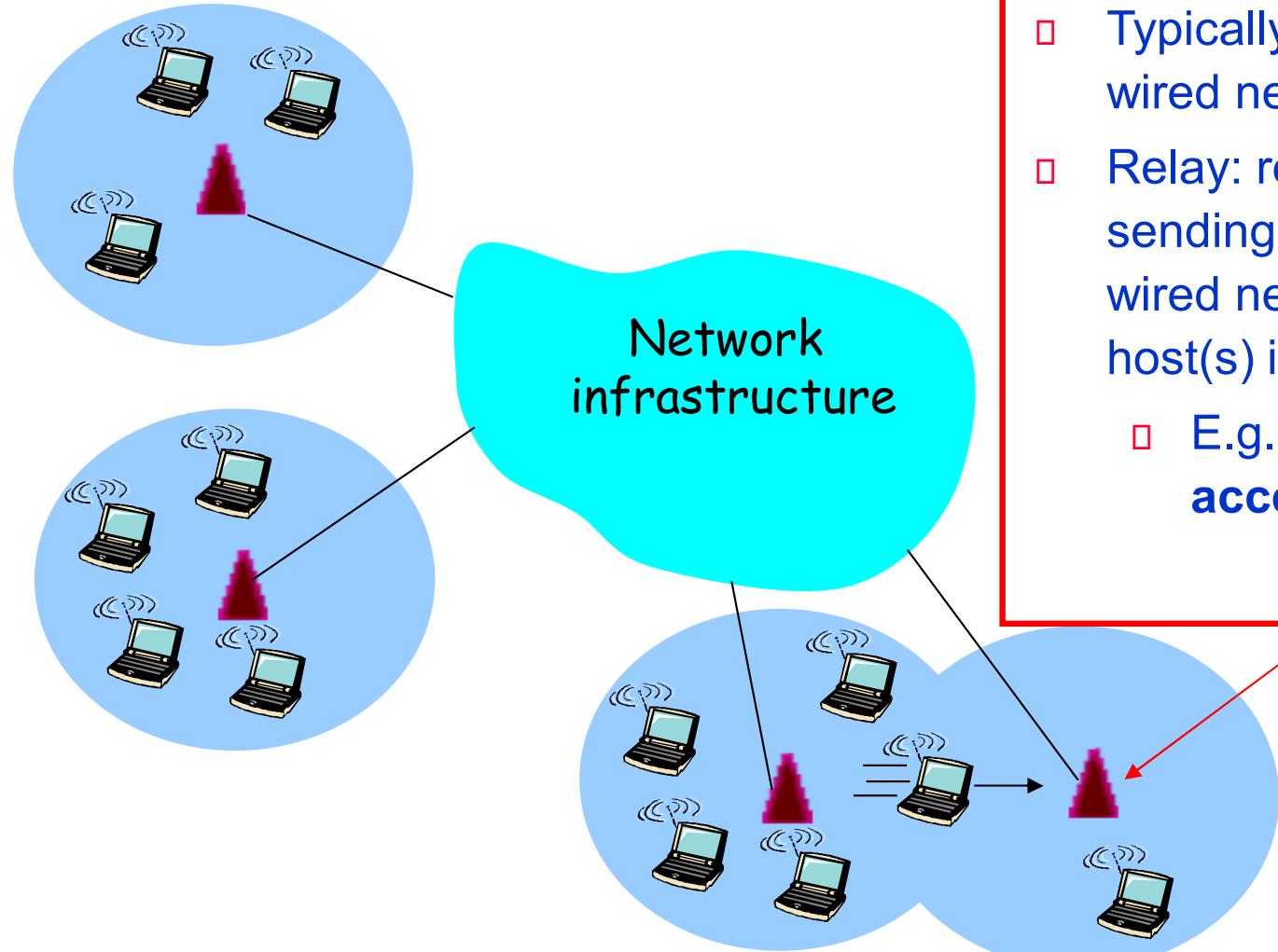


## Wireless hosts:

- Laptop, Smartphone, Tablet, ...;
- Run applications;
- May be stationary (non-mobile) or mobile:
  - **Wireless does *not* always mean mobility!**

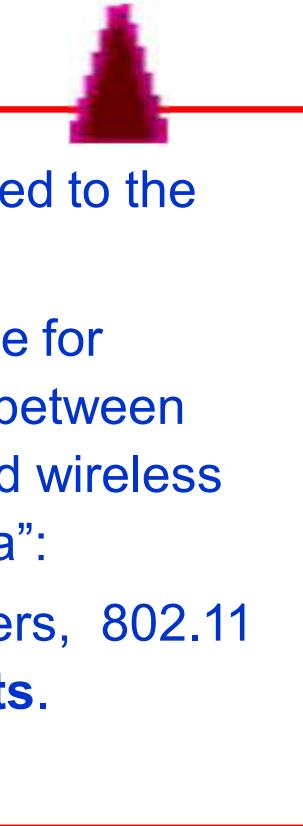


# *Elements of a Wireless Network*

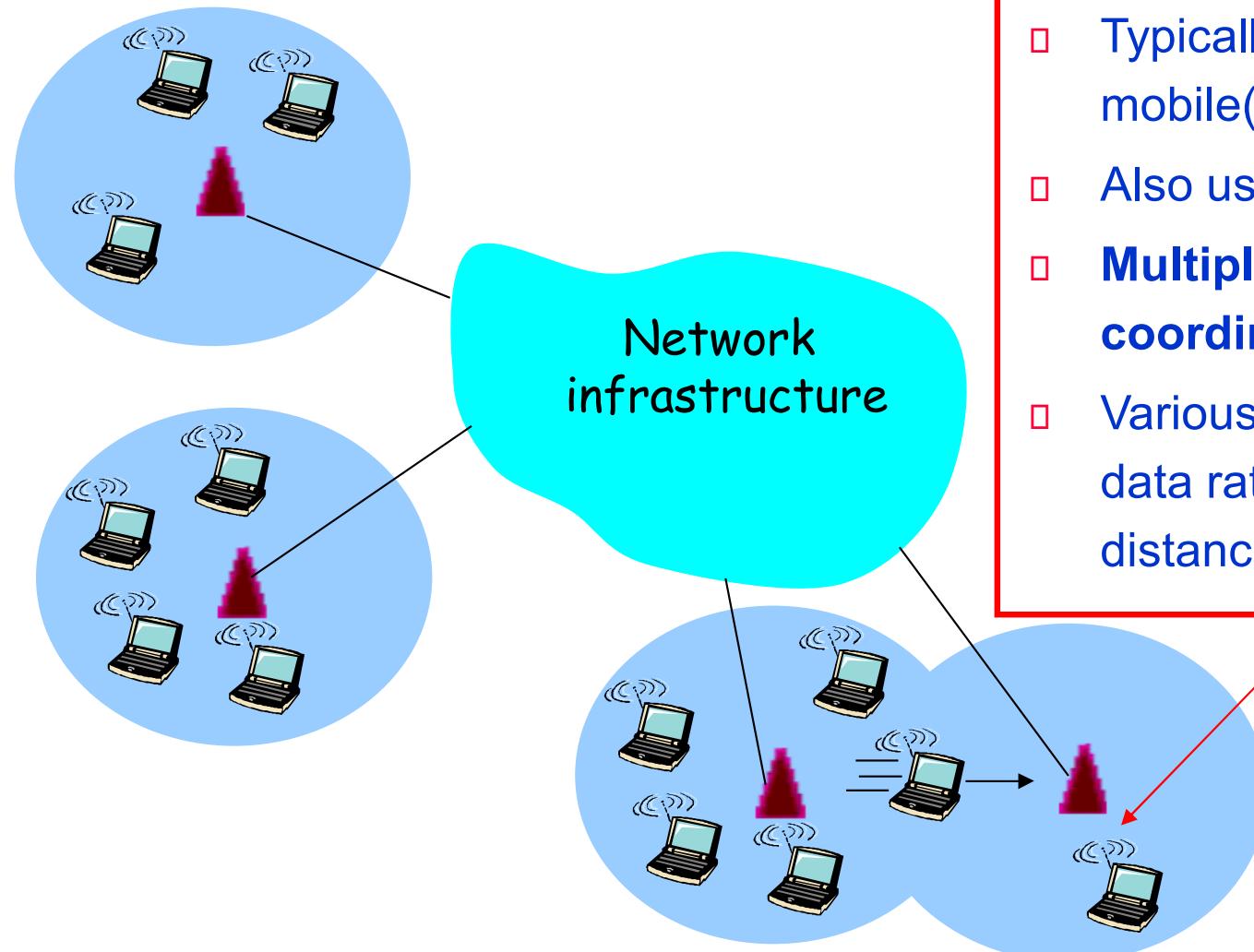


## **Base station:**

- Typically connected to the wired network;
- Relay: responsible for sending packets between wired network and wireless host(s) in its “area”:
  - E.g., cell towers, 802.11 access points.



# *Elements of a Wireless Network*

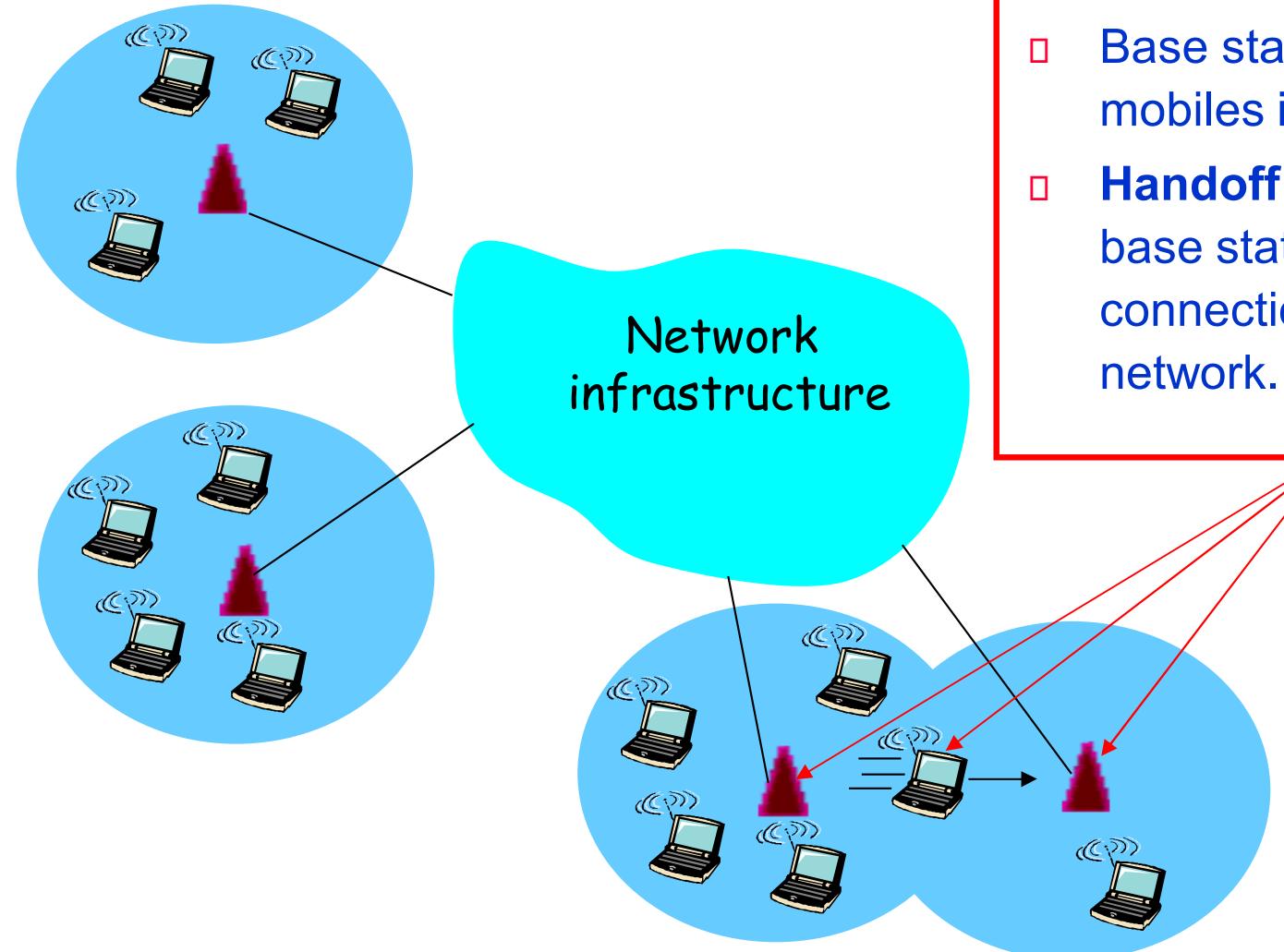


## Wireless link:

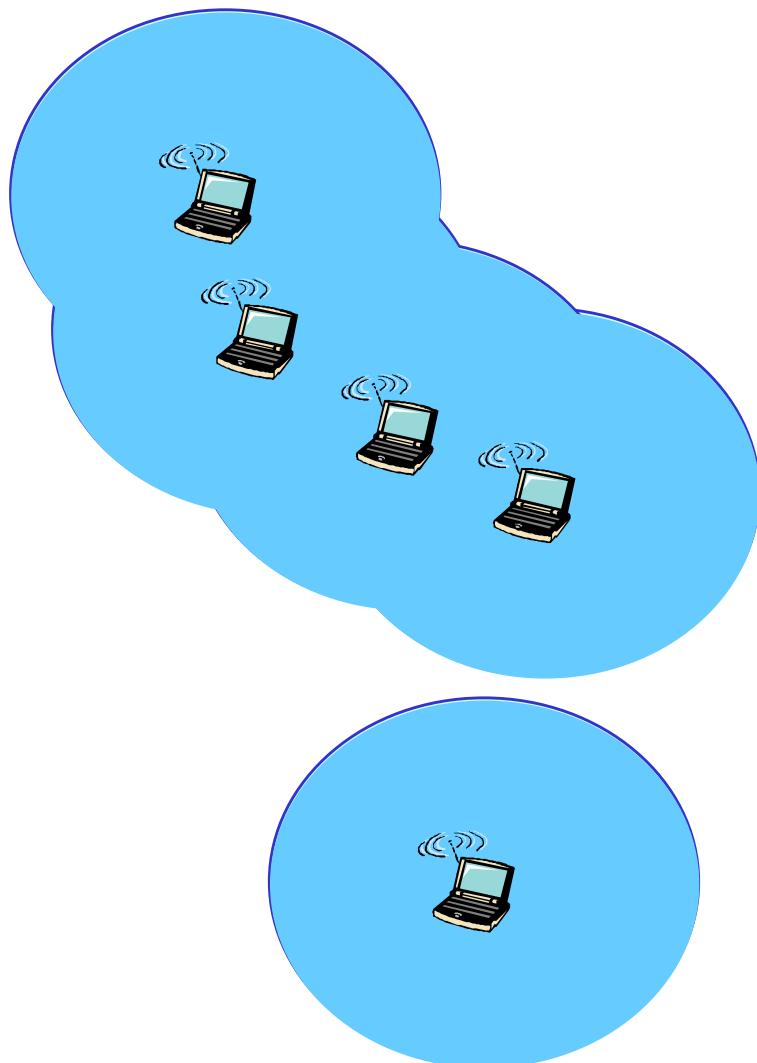
- Typically used to connect mobile(s) to base station;
- Also used as backbone link;
- **Multiple access protocol coordinates link access;**
- Various frequency bands, data rates, transmission distance.



# *Elements of a Wireless Network*



# *Elements of a Wireless Network*



## **Ad hoc mode:**

- **No base stations;**
- Nodes can only transmit to other nodes within link coverage;
- Nodes organize themselves into a network: route among themselves.

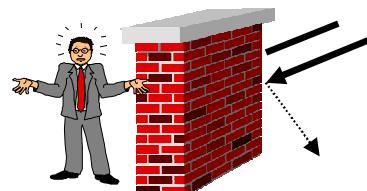
# Wireless Communications

Wireless communications are needed to:

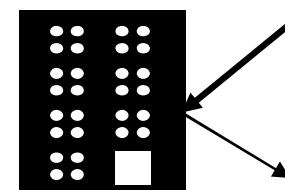
- Allow communications while moving;
- Allow communications in places where it is difficult, or impossible, to implement a cabled infrastructure;
- Allow broadcasting;
- Allow the fast implementation, with low initial cost, of a communications system;

However:

- Less controlled operation environment, more subject to interference, noise, unauthorized detection;
- Often provides lower transmission rates;
- Frequency bands are easier to reuse in guided media.



Shadow areas



Reflexions



Dispersion



Difraction

# Wireless Link Characteristics

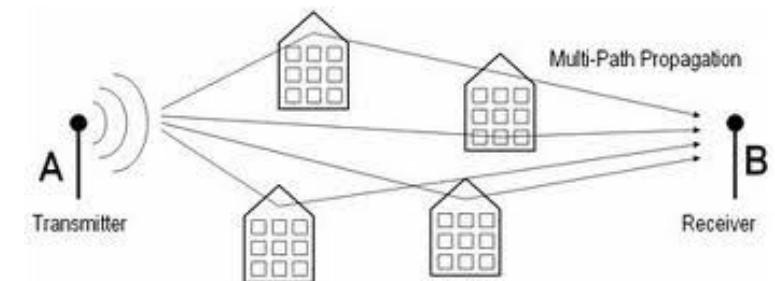
Differences from wired link:

- **Interference from other sources:**
  - Standardized wireless network frequencies (e.g., 2.4 GHz) shared by other devices (e.g., phone);
  - Devices (motors) interfere as well;
- **Multipath propagation:**

Radio signal reflects off objects and ground, arriving at destination at slightly different times;
- **Decreased signal strength:**

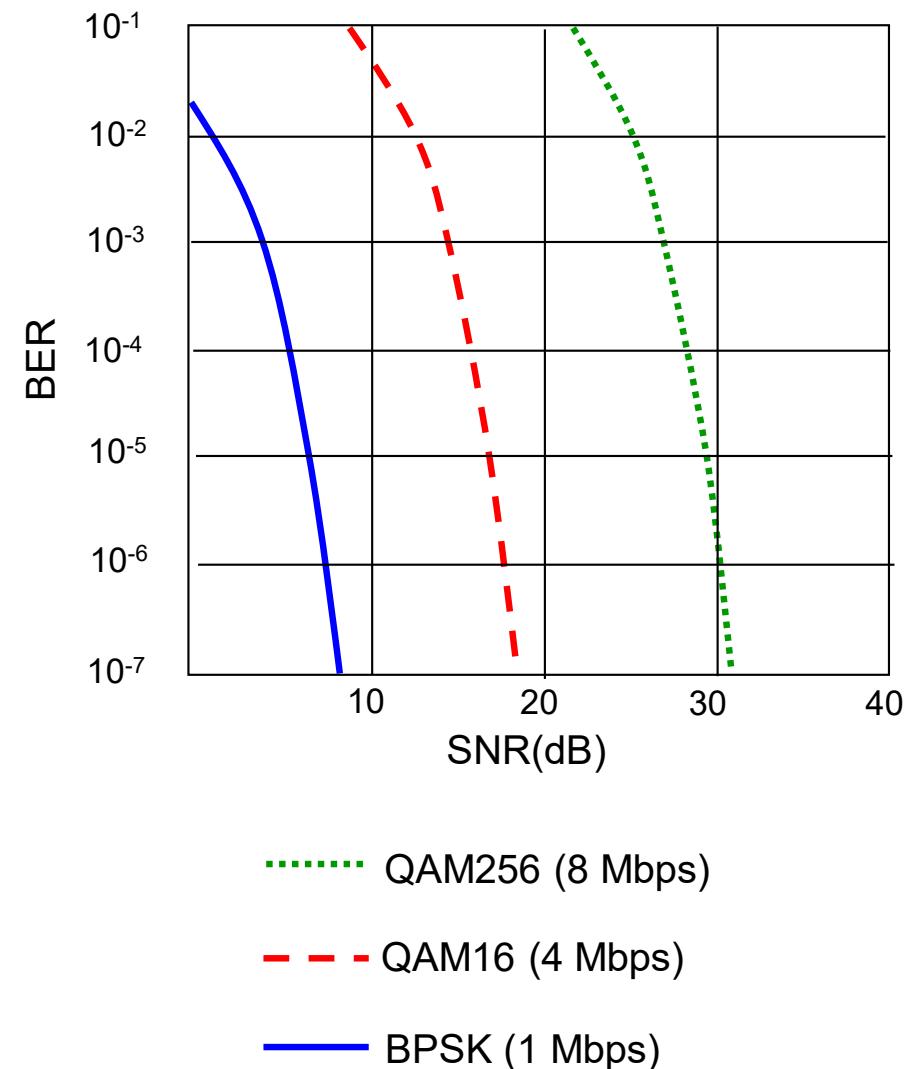
Radio signal attenuates as it propagates through matter (path loss).

.... make communication across (even a point to point) wireless link much more “difficult”!



# Wireless Link Characteristics

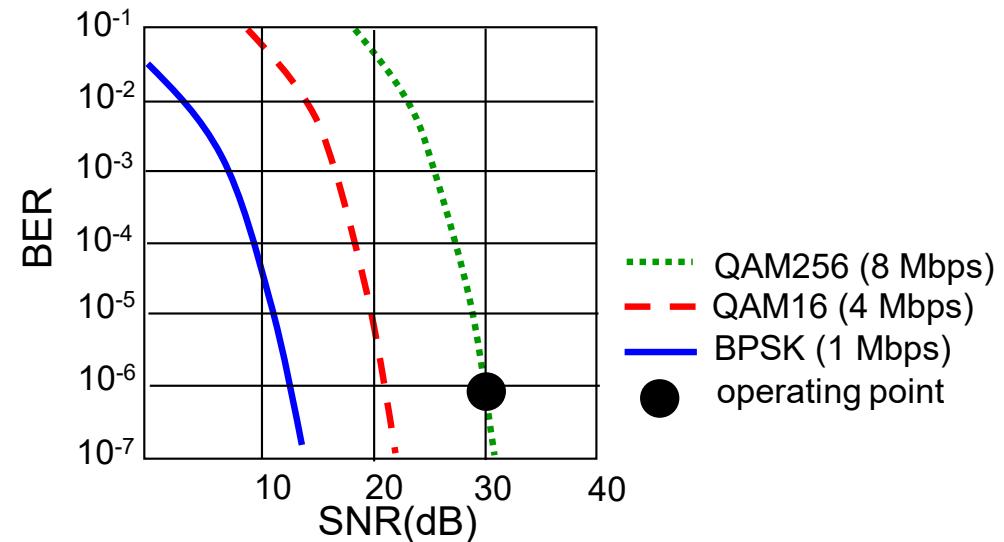
- Signal-to-noise ratio (SNR):
  - Larger SNR – easier to extract signal from noise (a “good thing”);
- ***SNR versus BER tradeoffs:***
  - ***Given physical layer:***  
increase power → increase SNR  
→ decrease bit error rate (BER);
  - ***Given SNR:*** choose physical layer that meets BER requirement, giving highest throughput:
    - SNR may change with mobility:  
**dynamically adapt physical layer (modulation technique, rate).**



# 802.11: Advanced Capabilities

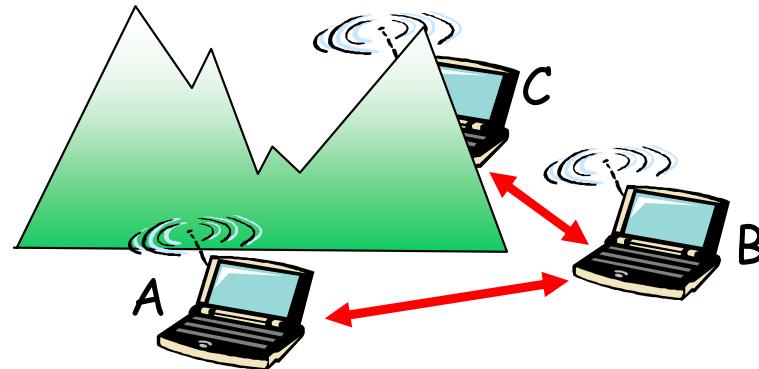
## Rate Adaptation:

- Base station, mobile:
  - Dynamically change transmission rate (physical layer modulation technique) as mobile moves and SNR varies.
  1. SNR decreases, BER increase as node moves away from base station
  2. When BER becomes too high, switch to lower transmission rate but with lower BER



# Wireless Network Characteristics

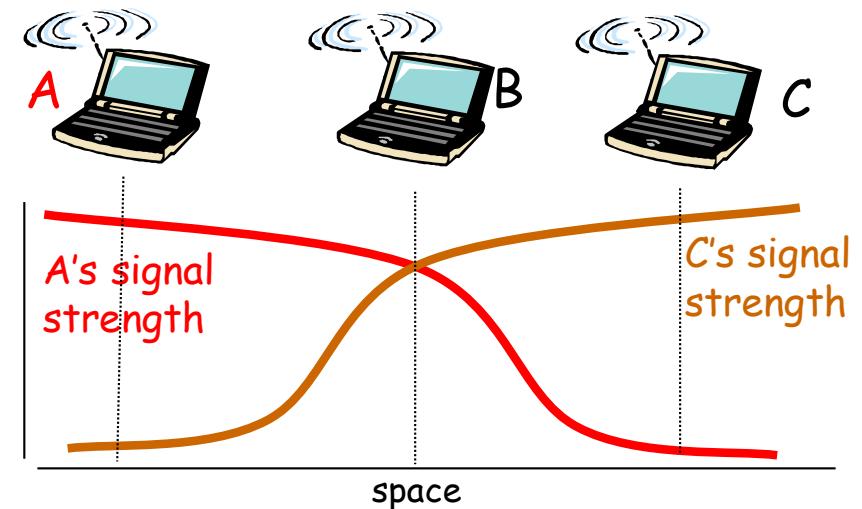
Multiple wireless senders and receivers create additional problems (beyond multiple access):



Hidden terminal problem:

- B, A hear each other;
- B, C hear each other;
- A, C can not hear each other;

→ A, C unaware of their interference at B.



Signal attenuation:

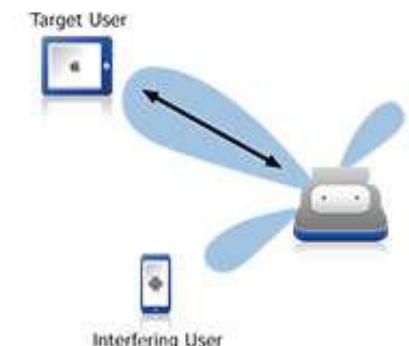
- B, A hear each other;
- B, C hear each other;
- A, C can not hear each other interfering at B.

# IEEE 802.11 Wireless LAN



- 802.11b
    - 2.4 GHz unlicensed spectrum; up to 11 Mbps.
  - 802.11a
    - 5 GHz band; up to 54 Mbps.
  - 802.11g
    - 2.4 GHz band; up to 54 Mbps.
  - 802.11n (multiple antennae)
    - 2.4, 5 GHz band (40 MHz channel); up to 600 Mbps.
  - 802.11ac (multiple antennae)
    - 5 GHz band (160 MHz channel); up to 1.3 Gbps; up to 3.5 Gbps (one user).
  - 802.11ad – WiGig (60 GHz – up to 10 m)
    - 2.4, 5, 60 GHz (80 or 160 MHz channel); up to 6.7 Gbps.
- 
- All use CSMA/CA for multiple access;
  - All have base-station and ad-hoc network versions.

Standard	Maximum Speed	Frequency	Backwards Compatible
802.11a	54 Mbps	5 GHz	No
802.11b	11 Mbps	2.4 GHz	No
802.11g	54 Mbps	2.4 GHz	802.11b
802.11n	600 Mbps	2.4 GHz or 5 GHz	802.11b/g
802.11ac	1.3 Gbps (1300 Mbps)	2.4 GHz and 5.5 GHz	802.11b/g/n
802.11ad	7 Gbps (7000 Mbps)	2.4 GHz, 5 GHz and 60 GHz	802.11b/g/n/ac

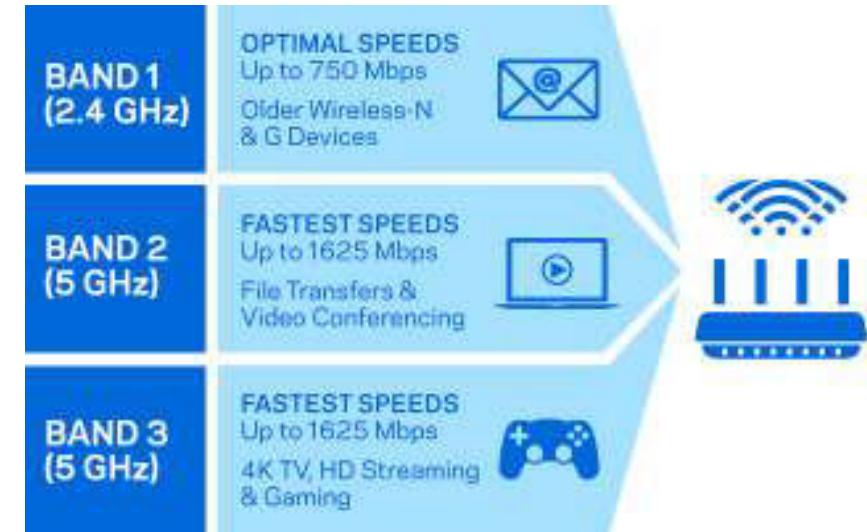
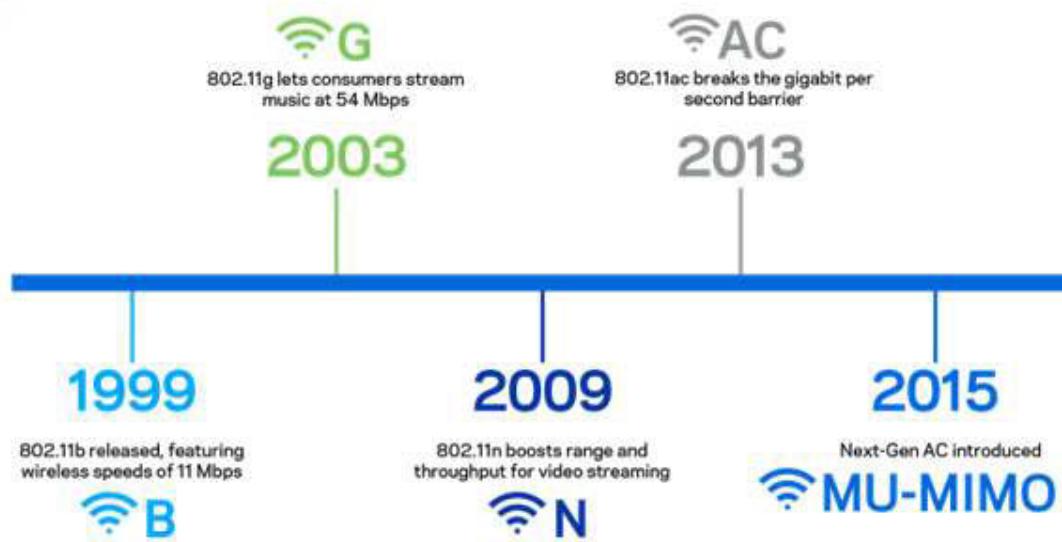


# IEEE 802.11 Wireless LAN



- 802.11ah (HaLow) – May 2017
  - <1, 2.4, 5 GHz; up to 347 Mbps.
- 802.11aj
  - To operate in China in bands 45, 59-64 GHz.
- 802.11ak
  - Emphasis on standardized security and quality-of-service improvements.
- 802.11ax (2019)
  - 2.4, 5 GHz band. High Efficiency WLAN; 4x faster than 802.11 n or ac.
- 802.11ay (2019)
  - Next Generation 60 GHz; > 20 Gbit/s.
- 802.11az (2021)
  - Next Generation Positioning (NGP) – emphasis on determining the absolute and relative position of stations.
- 802.11ba (2020)
  - Wake-Up Radio (WUR) – emphasis on extending battery life.

# IEEE 802.11 Wireless LAN

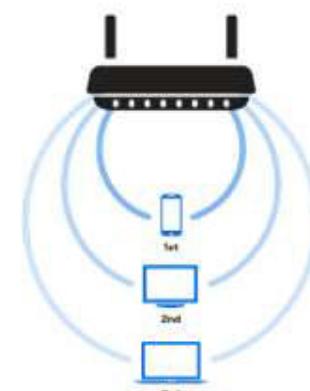


## TRADITIONAL ROUTERS | MAX-STREAM ROUTERS

Single-User MIMO Technology

Multi-User MIMO Technology

Wi-Fi to one device at a time.



VS

Wi-Fi to multiple devices at once, at the same speed.

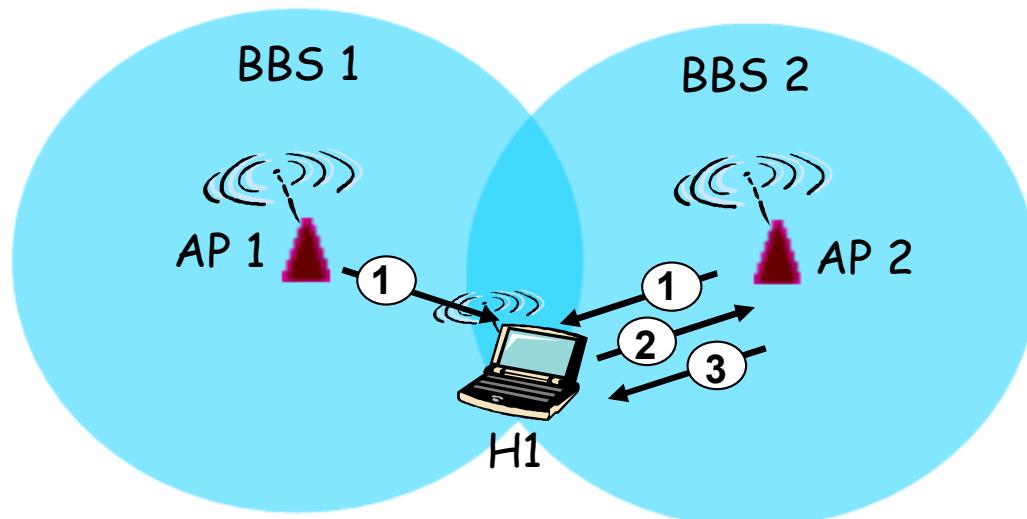


## 802.11: Channels, Association

### 802.11b:

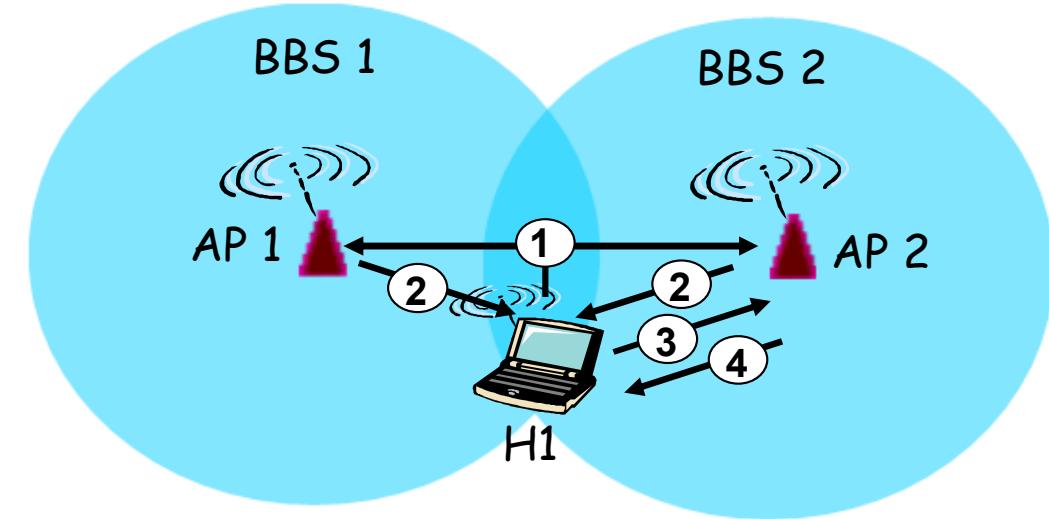
- 2.4GHz - 2.485GHz spectrum divided into 11 channels at different frequencies:
  - AP administrator chooses frequency for AP;
  - Interference possible:  
Channel can be same as that chosen by neighboring AP !
- Host – must *associate* with an AP:
  - Scans channels, listening for *beacon frames* containing APs names (SSID) and MAC addresses;
  - Selects AP to associate with;
  - May perform authentication;
  - Will typically run DHCP to get IP address in AP's subnet.

## 802.11: Passive/Active Scanning



### Passive Scanning:

- (1) Beacon frames sent from APs;
- (2) Association Request frame sent: H1 to selected AP;
- (3) Association Response frame sent: H1 to selected AP.

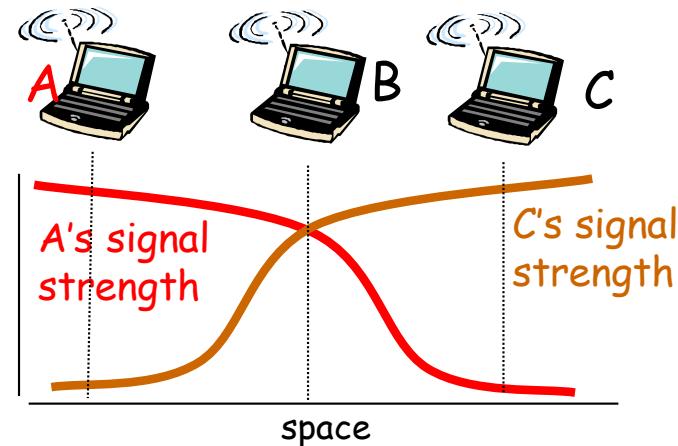
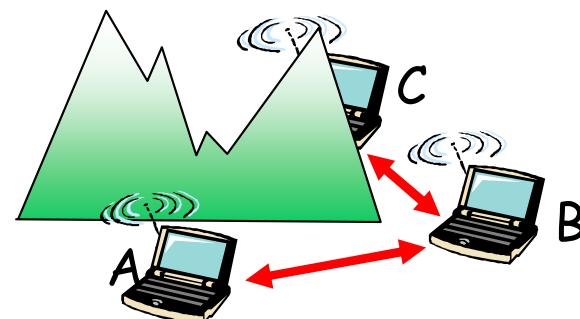


### Active Scanning:

- (1) Probe Request frame broadcast from H1;
- (2) Probe Response frame sent from APs;
- (3) Association Request frame sent: H1 to selected AP;
- (4) Association Response frame sent: H1 to selected AP.

# IEEE 802.11: Multiple Access

- 802.11: CSMA - sense channel before transmitting
  - Don't collide with ongoing transmission by other node.
- 802.11: no collision detection!
  - Difficult to receive (sense collisions) when transmitting due to weak received signals (fading);
  - Can't sense all collisions in any case: hidden terminal, fading;
  - **Goal: avoid collisions: CSMA/CA (Collision Avoidance).**



1. NIC receives datagram from network layer → creates frame;
2. If **channel idle** (96 bit times), starts frame transmission;  
If **channel busy**, waits until channel idle, then transmits
3. If NIC transmits entire frame without detecting another transmission:  
→ success;
4. If NIC detects another transmission while transmitting - **collision**:  
→ aborts and sends jam signal (reinforce collision);
5. After aborting, NIC enters **exponential backoff**:  
→ After  $m^{\text{th}}$  collision, NIC chooses  $K$  at random from  $\{0,1,2,\dots,2^m-1\}$ ;  
→ NIC waits  $K \times 512$  bit times; then returns to Step 2.

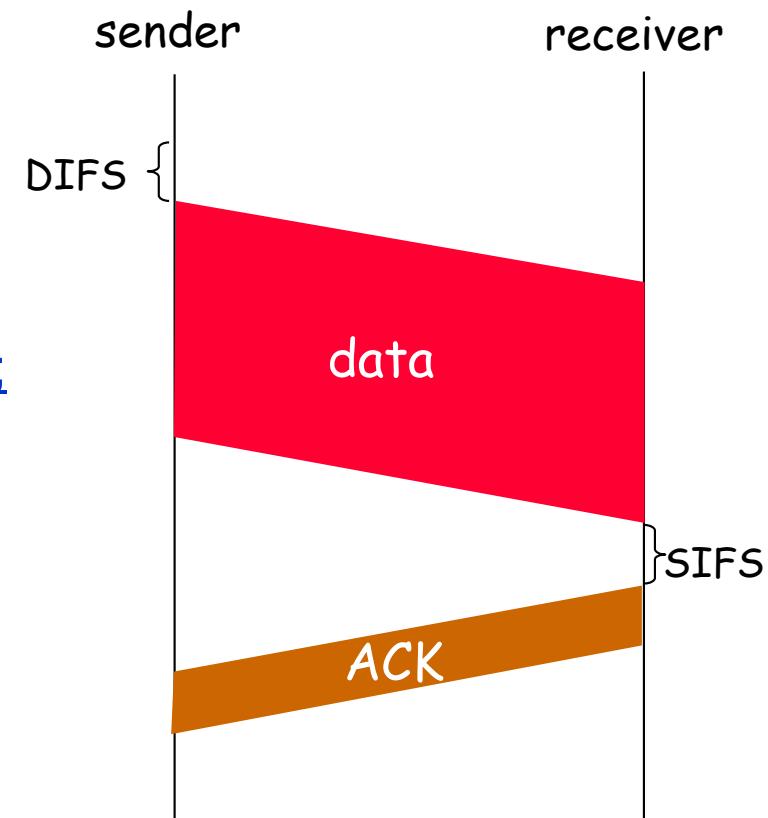
# IEEE 802.11 MAC Protocol: CSMA/CA

## 802.11 sender

1. If sense **channel idle for DIFS** then:
  - Transmit entire frame (no CD);
2. If sense **channel busy** then:
  - Start random backoff time;
  - Timer counts down **only** when channel is idle;
  - Transmit entire frame when timer expires;
  - If no ACK, increase random backoff interval, then repeat from 2.

## 802.11 receiver

- If frame received **OK**:
  - Return **ACK after SIFS**  
*(ACK is needed to confirm the success of a transmission; it also helps with the hidden terminal problem).*



# Collision Avoidance

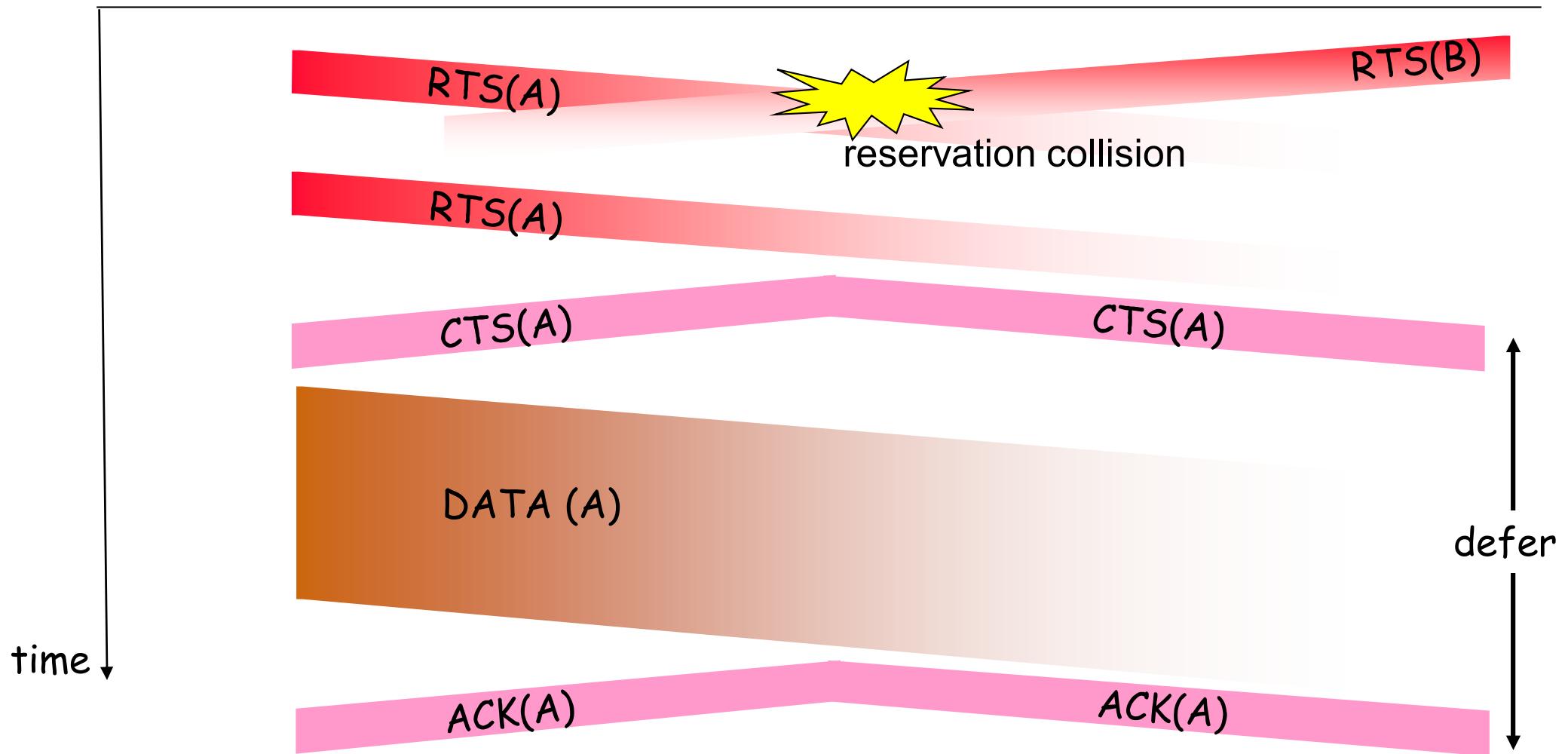
## Adding RTS/CTS

Idea:

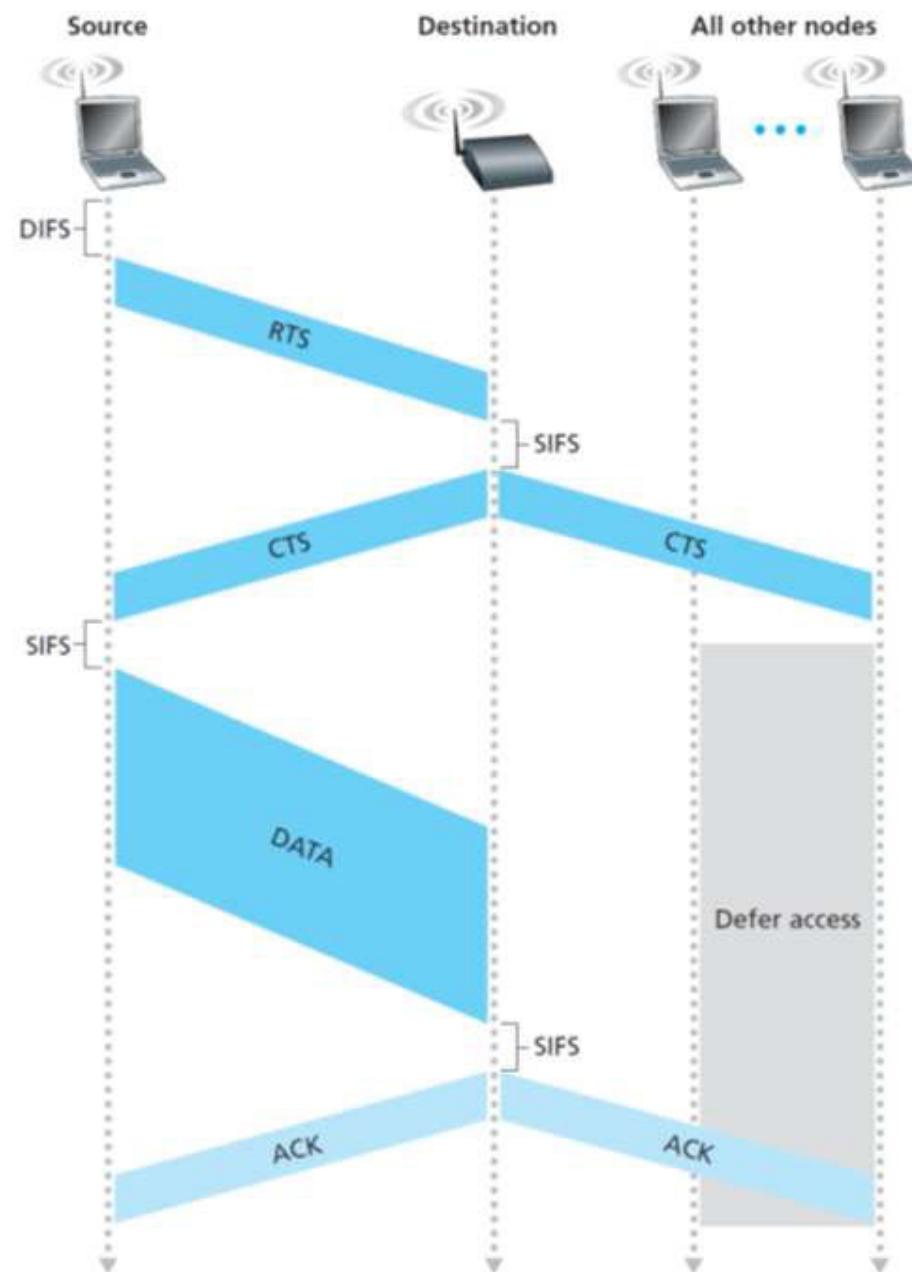
- Allow sender to “**reserve**” **channel** rather than random access of data frames → avoid collisions of long data frames;
- Sender first transmits *small request-to-send (RTS)* packets to BS using CSMA:
    - RTSs may still collide with each other (but they’re short);
  - BS broadcasts **clear-to-send CTS** in response to RTS;
  - CTS heard by all nodes:
    - Sender transmits data frame;
    - Other stations defer transmissions.

Avoid data frame collisions completely  
using small reservation packets!

# Collision Avoidance: Adding RTS-CTS

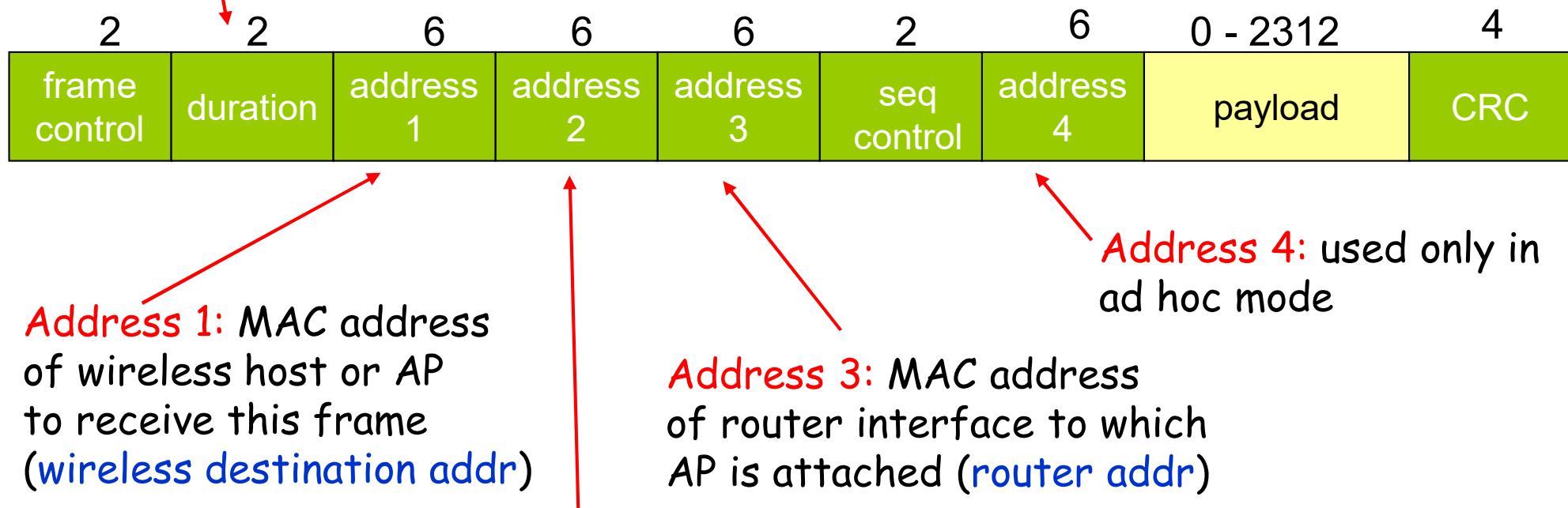


# Collision Avoidance: Adding RTS-CTS



## 802.11 Frame: Addressing

Duration: to “reserve” channel  
(for transmission + ACK)



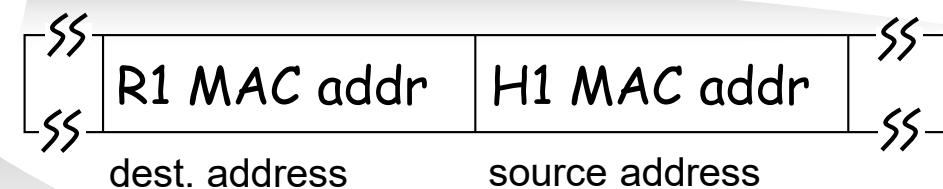
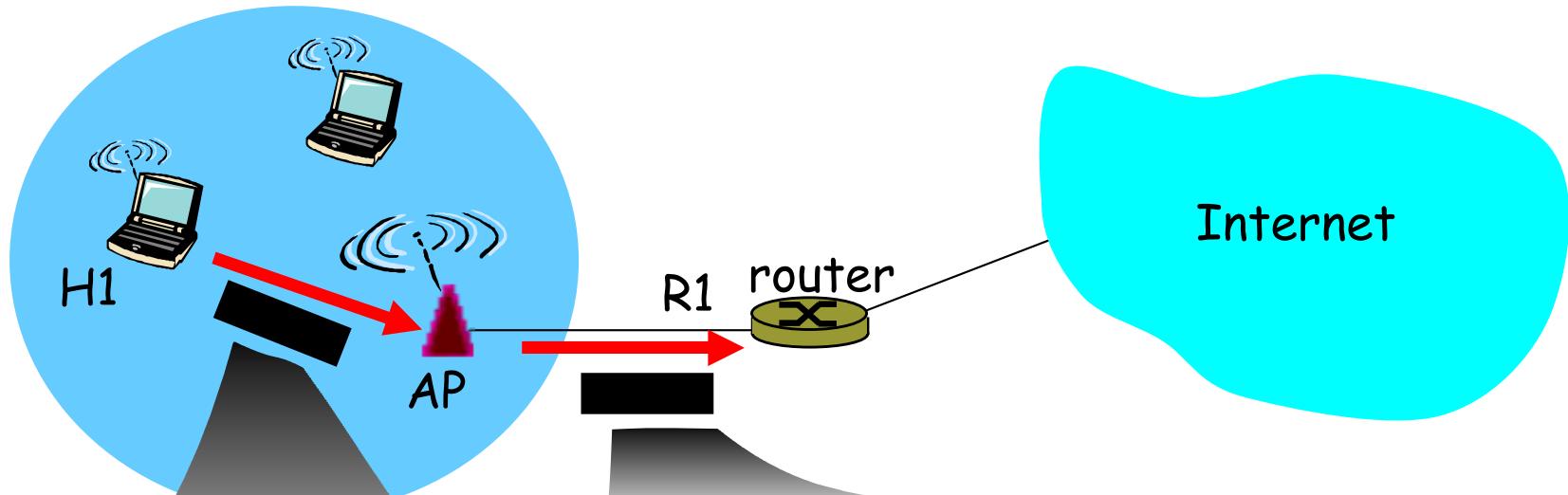
Address 2: MAC address  
of wireless host or AP  
transmitting this frame  
(sender addr)

Address 4: used only in  
ad hoc mode

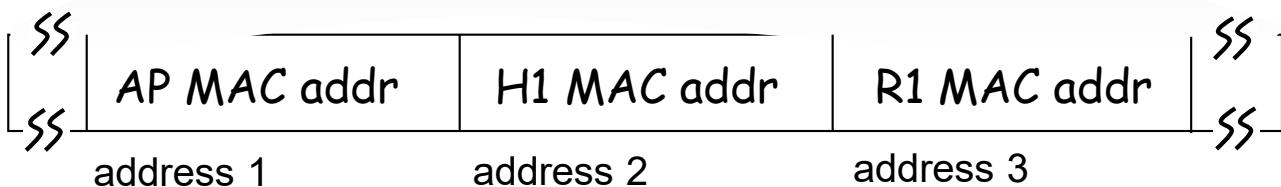
Address 3: MAC address  
of router interface to which  
AP is attached (router addr)

# *802.11 Frame: Addressing*

## TPC: Prob. 17

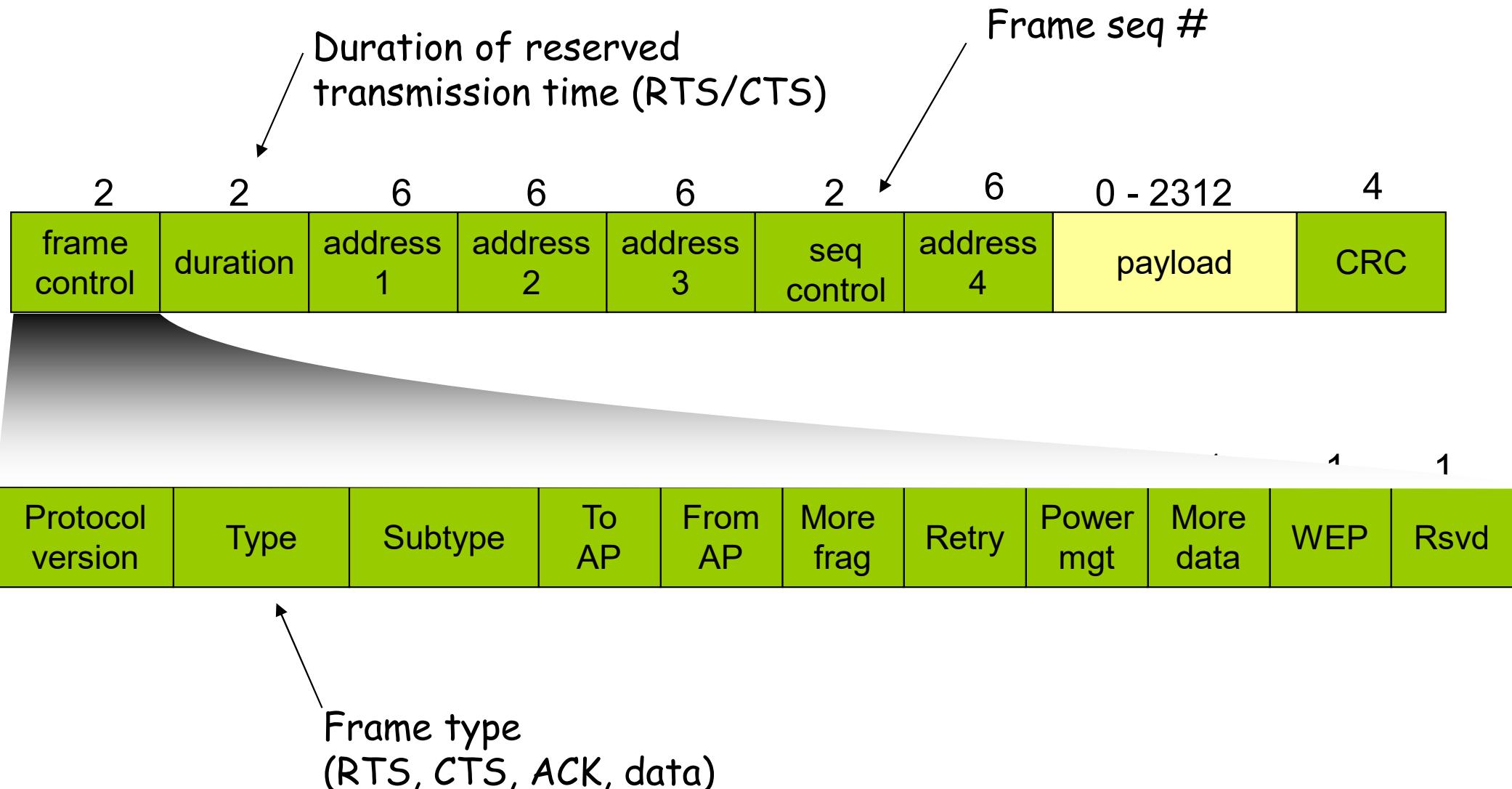


## 802.3 frame



# 802.11 frame

## 802.11 Frame: more



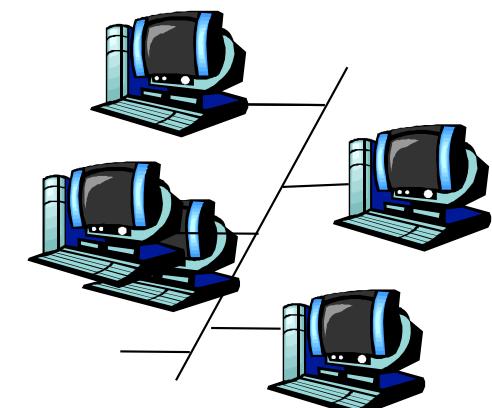
# 802.11: Advanced Capabilities

## *Power Management:*

- Node-to-AP: “I am going to sleep until next beacon frame”
  - AP knows not to transmit frames to this node;
  - Node wakes up before next beacon frame;
- Beacon frame: contains list of mobiles with AP-to-mobile frames waiting to be sent:
  - Node will stay awake if AP-to-mobile frames are to be sent;  
Otherwise sleep again until next beacon frame.

# Outline

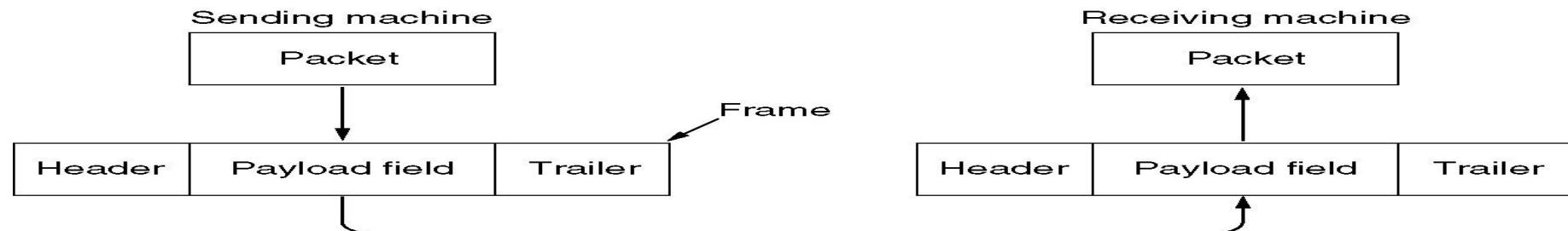
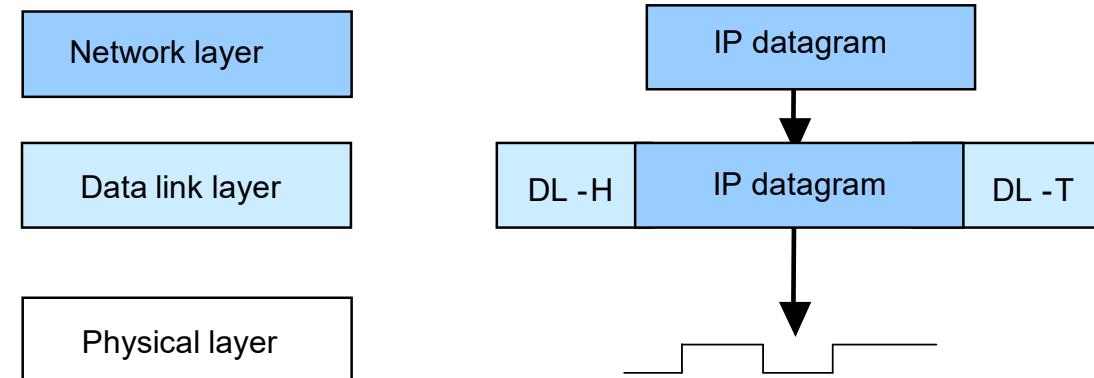
- Introduction and services
- Link-layer Addressing
- Error detection and correction
- Multiple access protocols
- Ethernet
- Link-layer switches
- IEEE 802.11 Wireless LANs
- Framing



# Framing

## Data Link layer:

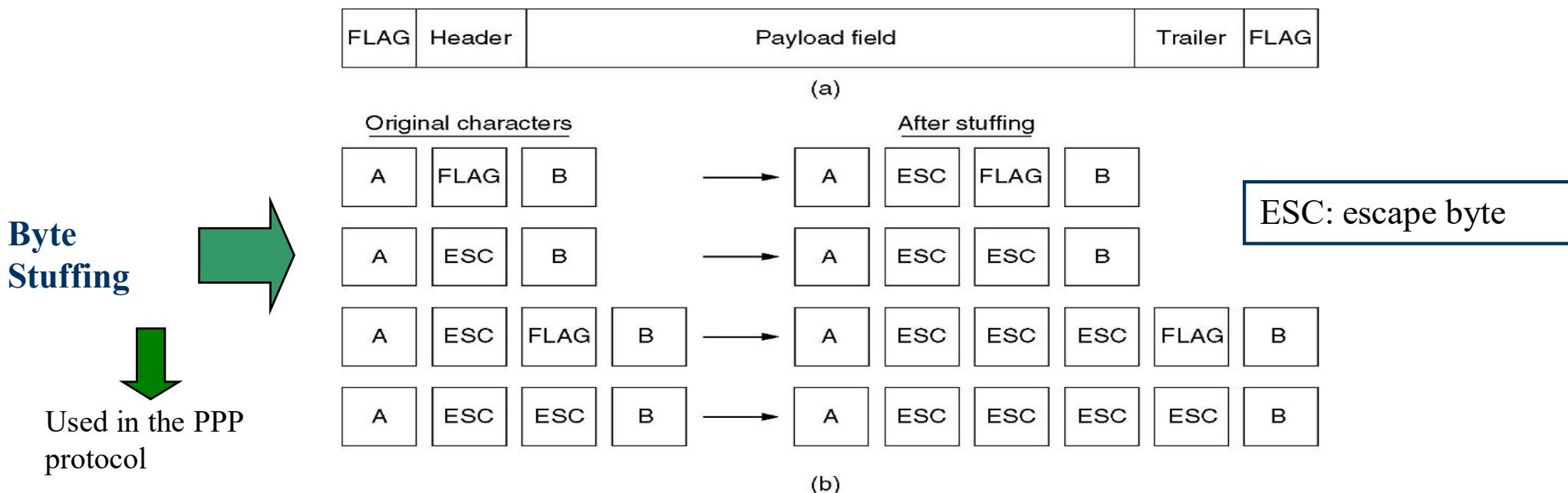
- Frames include a **header** with synchronization, addressing and control (type and frame number) information;
- Frames also include a **trailer** with error control and synchronization information.



## Framing: Byte “Stuffing”

Network layer data may contain binary patterns equal to those used for synchronization (*flags*) purposes in the data link layer:

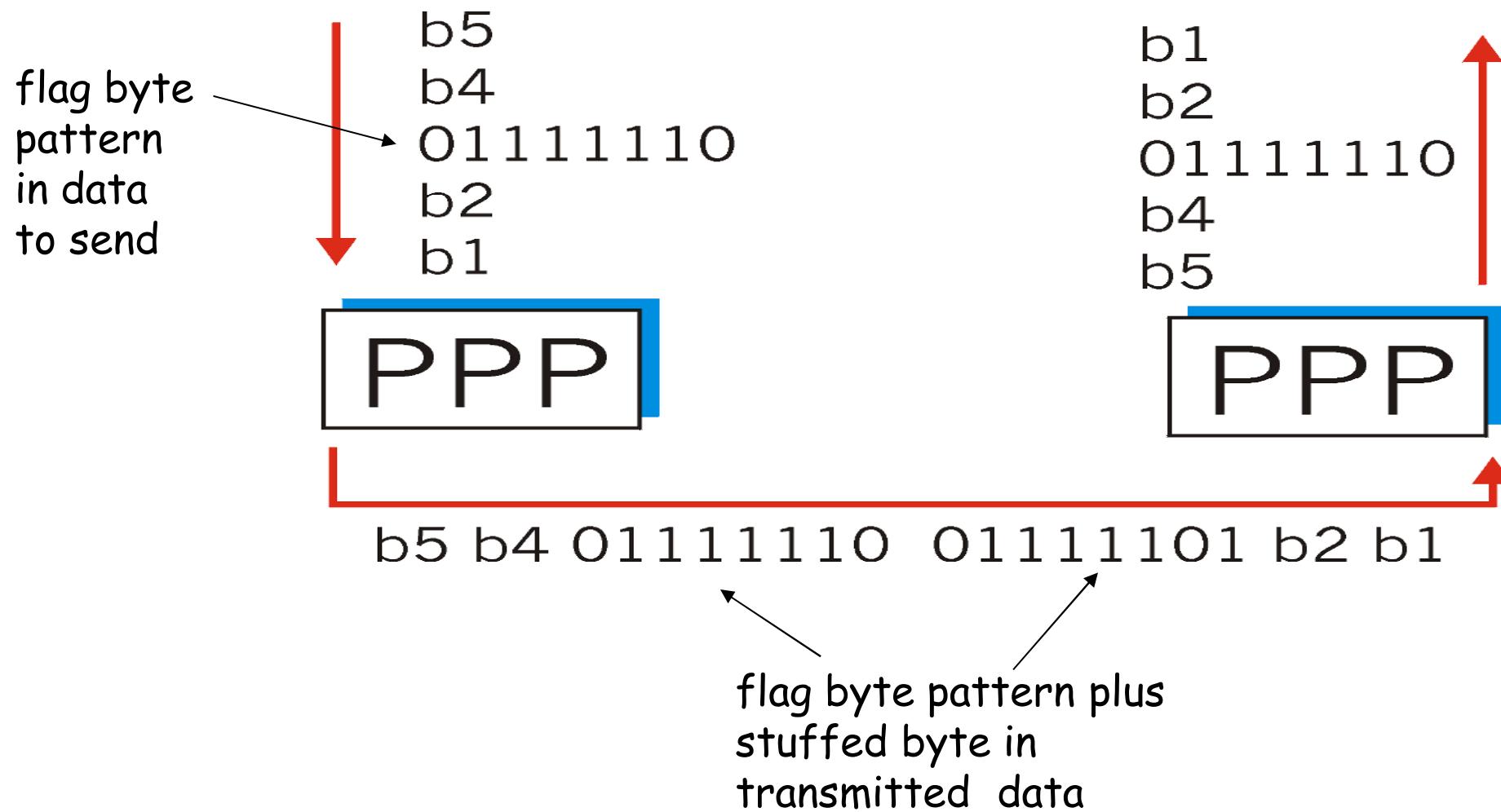
- There is the need to avoid taking those IP data as a data link *flag*;
- To that purpose, if the *flag* pattern appears in the IP data it should be preceded, in the data link frame, by a ***stuffing*** pattern, to avoid any confusion.



# Byte Stuffing

- “Data transparency” requirement: data field must be allowed to include flag pattern <01111110>.
  - Q: is received <01111110> data or flag?
- Sender: adds (“stuffs”) extra <01111110> byte after each <01111110> *data* byte.
- Receiver:
  - Two 01111110 bytes in a row: discard first byte, continue data reception;
  - Single 01111110: flag byte.

## Byte Stuffing



## **Framing: Bit “Stuffing”**

(a) 011011111111111110010

# Bit Stuffing

(b) 0 1 1 0 [1 1 1 1 1] 0 [1 1 1 1 1] 0 [1 1 1 1 1] 0 1 0 0 1 0

↑  
↑  
↑

Stuffed bits      Stuffed bits      Stuffed bits

(c) 011011111111111110010

## Example:

- Frame delimitation using the *flag*: **0111 1110**;
  - **Bit stuffing** consists in:
    - Add an extra 0 (b) after any sequence of five consecutive 1s in the data field coming from the network layer (a), to avoid being taken for the *flag*.
  - In the receptor the inverse operation takes place (c).

## Summary

- Principles behind data link layer services:
  - Link layer addressing;
  - Error detection, correction;
  - Sharing a broadcast channel: **multiple access**.
- Instantiation and implementation of various link layer technologies:
  - Ethernet;
  - IEEE 802.11 Wireless LANs;
  - Switched LANS;

## Summary

- Journey down protocol stack *complete* (except PHY);
- Solid understanding of networking principles and practice;
- ..... could stop here .... but *lots* of interesting topics!
  - Multimedia,
  - Security,
  - Network management, ...

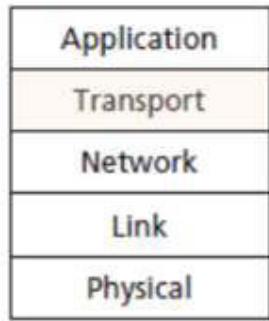
# Redes de Computadores

## LEIC-A

### *3 – Transport Layer (part 2)*

Prof. Paulo Lobato Correia  
*IST, DEEC – Área Científica de Telecomunicações*

# Objectives



- Understand the principles behind transport layer services:
  - Multiplexing/demultiplexing;
  - Reliable data transfer;
  - Flow control;
  - Congestion control.
- Transport layer protocols in the Internet:
  - UDP: connectionless transport;
  - TCP: connection-oriented transport with congestion control;
  - QUIC: Quick UDP Internet Connections.

# Internet Transport Layer Protocols

## □ Reliable, in-order delivery (TCP):

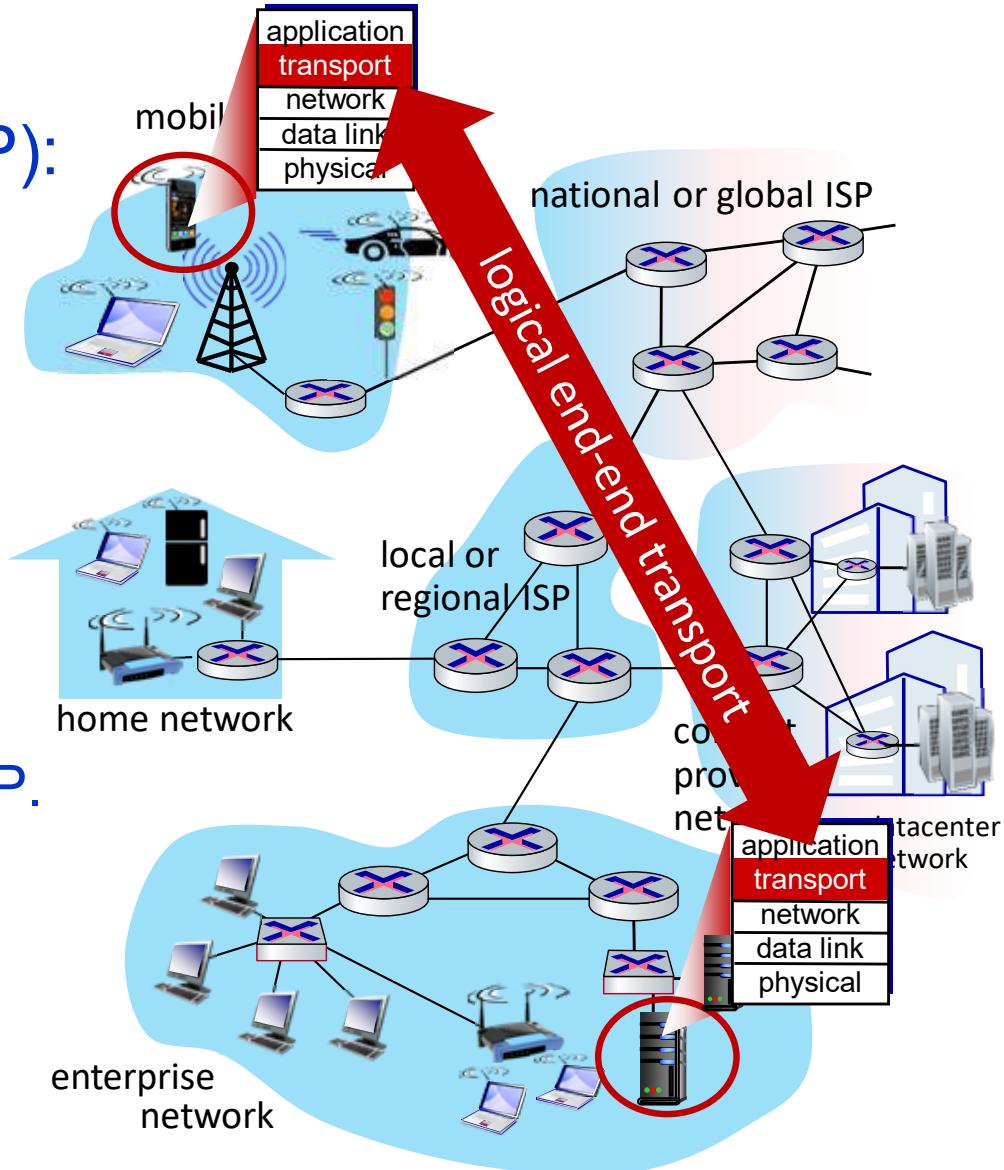
- Connection setup;
- Flow control;
- Congestion control.

## □ Unreliable, unordered delivery (UDP):

- Simple extension of “best-effort” IP.

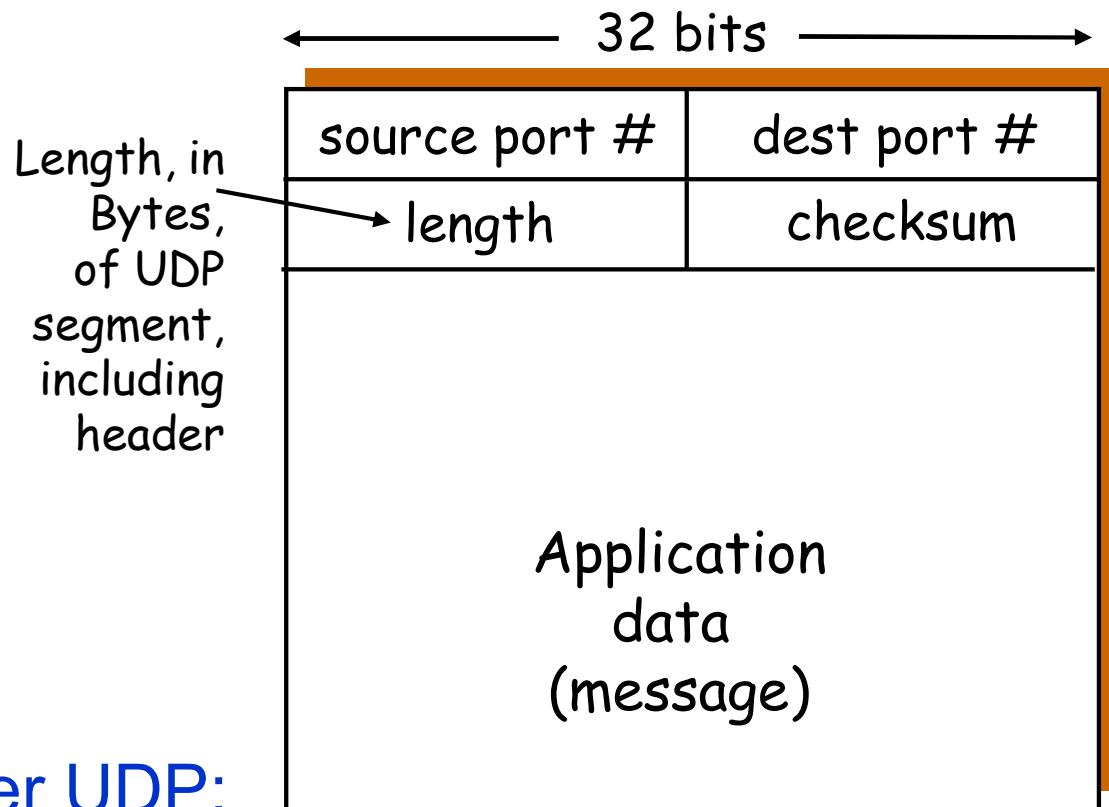
## □ Services not available:

- Delay guarantees;
- Bandwidth guarantees.



# UDP: User Datagram Protocol

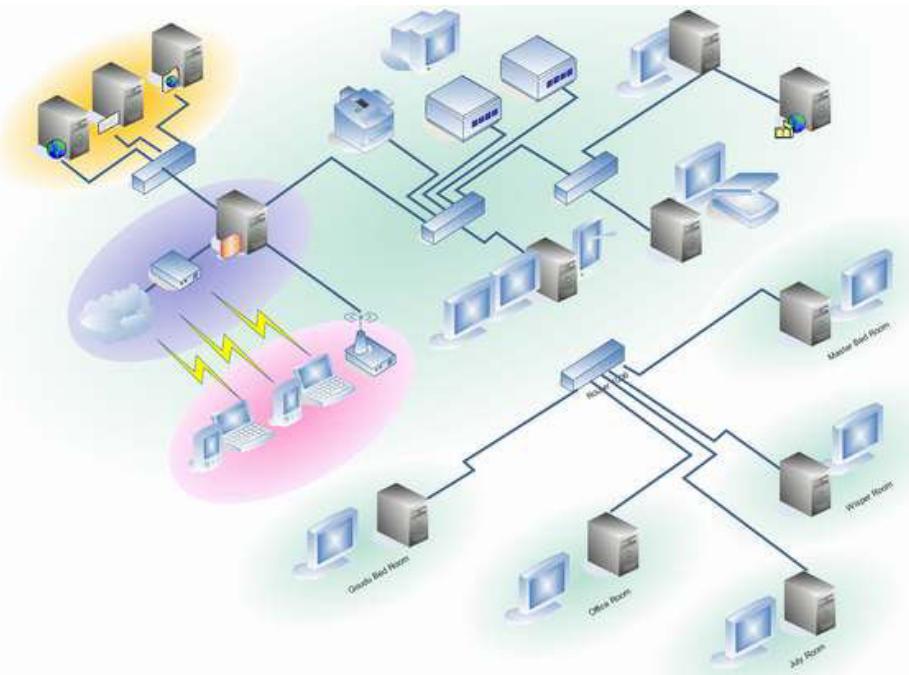
- Often used for streaming multimedia applications:
  - Loss tolerant;
  - Rate sensitive.
- Other UDP uses
  - DNS;
  - SNMP;
  - QUIC.
- To have reliable transfer over UDP:
  - Reliability added at application layer;
  - Application-specific error recovery!



UDP segment format

# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Connection management;
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
- Principles of congestion control
- TCP congestion control
- QUIC



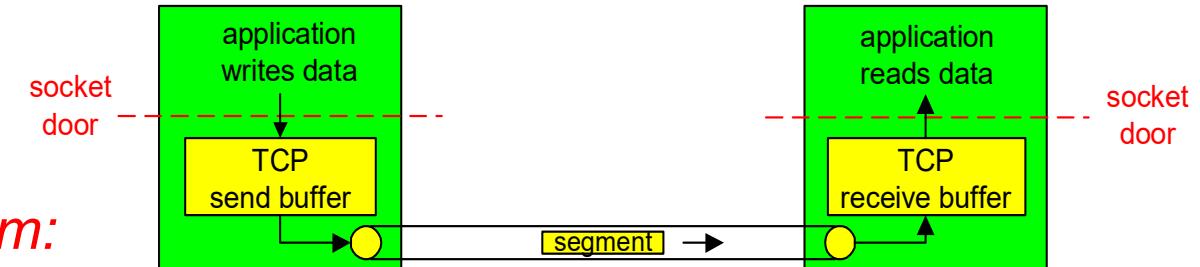
# TCP: Overview

Some RFCs: 675, 793, 1122, 1323, 1379, 1948, 2018, 5681, 6247, 6298, 6824, 7323, 7424, ...

- RFC 793: "Transmission Control Protocol", STD 7 (*Sept 1981*) - fundamental TCP specification document
- RFC 1122: "Requirements for Internet Hosts - Communication Layers" (*Oct 1989*) - updates and clarifies RFC 793, fixes specification bugs and oversights. **Mandates that a congestion control mechanism must be implemented.**
- RFC 5681: "TCP Congestion Control" (*Aug 2009*) - defines **congestion avoidance and control mechanism for TCP**. RFCs 2001 and 2581 are conceptual precursors of RFC 5681.
- RFC 6298: "Computing TCP's Retransmission Timer" (*June 2011*)
- RFC 6691: "TCP Options and Maximum Segment Size (MSS)" (*July 2012*)
- RFC 7323: "TCP Extensions for High Performance" (*Sept 2014*)
- RFC 7414 - A Roadmap for Transmission Control Protocol (TCP) Specification Documents

# TCP: Overview

- Point-to-point:
  - One sender, one receiver;
- Reliable, in-order *byte steam*:
  - No “message boundaries”;
- Sliding window:
  - TCP congestion and flow control set window size;
- Send & receive buffers;
- Full duplex data:
  - Bi-directional data flow in same connection;
  - **MSS**: maximum segment size;
- Connection-oriented:
  - Handshaking (exchange of control messages)  
initializes sender and receiver state before data exchange;
- Flow control:
  - Sender will not overwhelm receiver.



# TCP segment structure

ACK: seq # of next expected byte;  
 A bit: this is an ACK

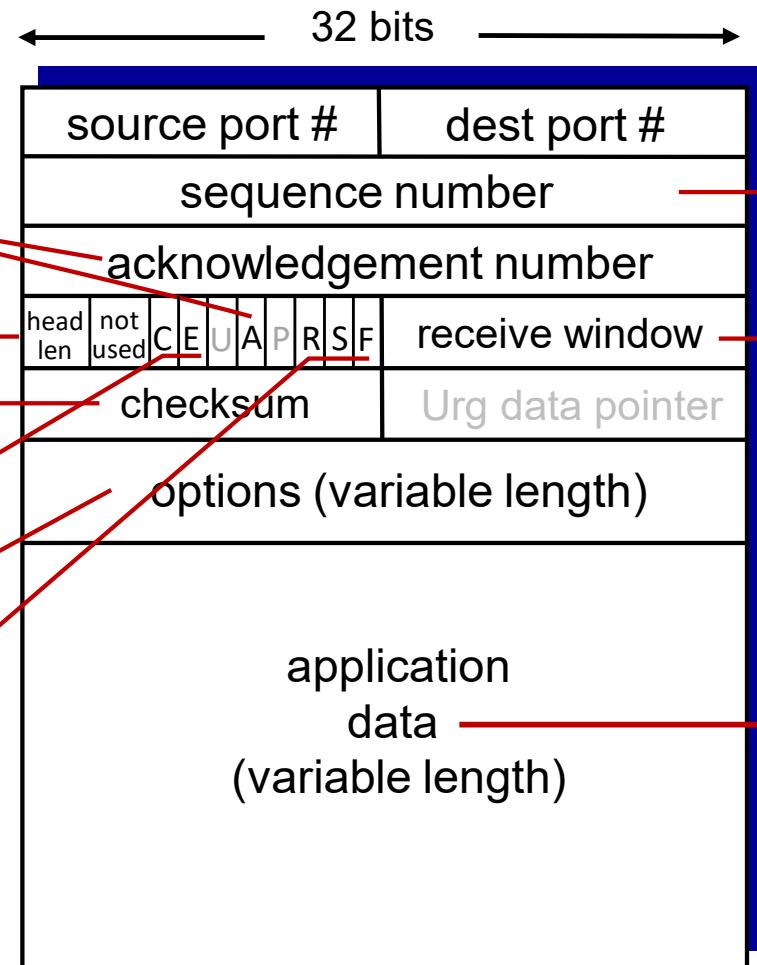
length (of TCP header)

Internet checksum

C, E: congestion notification

TCP options

RST, SYN, FIN: connection management



segment seq #: counting bytes of data into bytestream (not segments!)

flow control: # bytes receiver willing to accept

data sent by application into TCP socket

# TCP Connection Management

## Recall:

- TCP sender and receiver establish “connection” before exchanging data segments;
- Initialize TCP variables: seq. numbers, buffers, flow control information (e.g. **RcvWindow**);
- *Client*: initiates connection;
- *Server*: contacted by client.

## Three-way handshake:

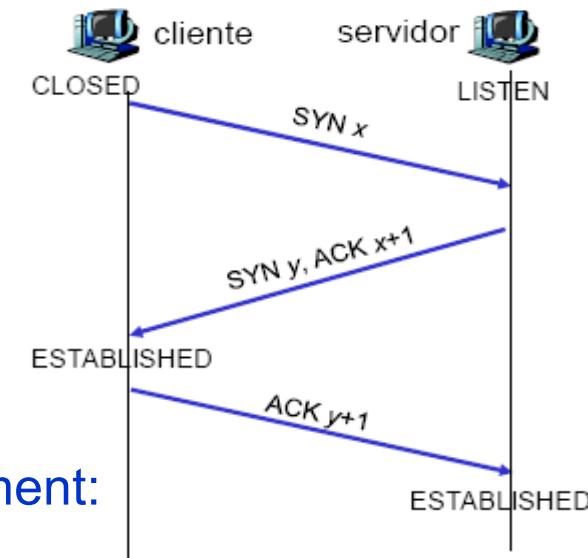
Step 1: Client sends TCP **SYN** segment to server:

- Specifies initial client seq. number (x);
- No data included.

Step 2: Server receives SYN and replies with **SYN ACK** segment:

- Server allocates buffers;
- Specifies server initial seq. number (y).

Step 3: Client receives SYN ACK, replies with **ACK** segment, which **may already contain data**.



# TCP Closing a Connection

Client and server each close their side of connection.

Example: client closes socket (*server could start*) :

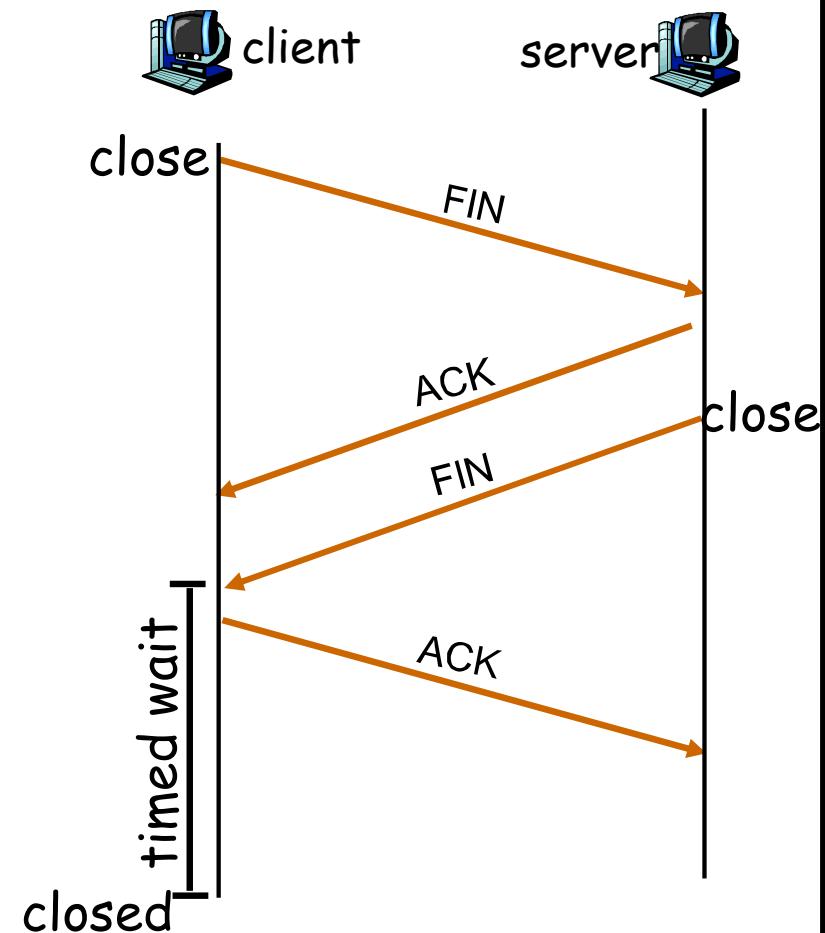
**Step 1:** Client end system sends TCP **FIN** control segment to server;

**Step 2:** Server receives FIN, replies with **ACK**. Closes connection, sends **FIN**.

**Step 3:** Client receives FIN, replies with **ACK**.

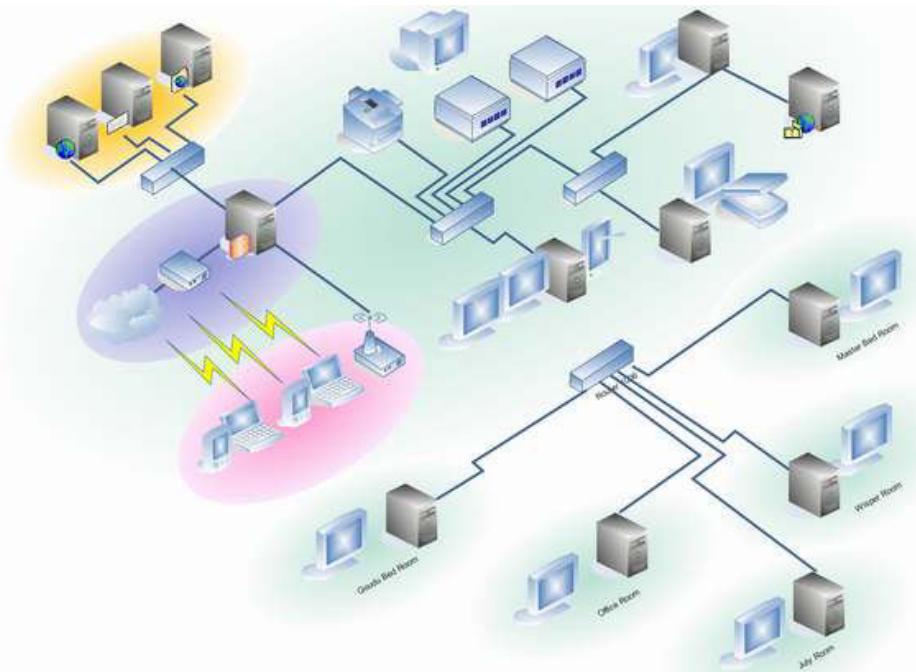
- Enters “**timed wait**” - will respond with ACK to received FINs.

**Step 4:** Server, receives ACK. Connection closed.



# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- **Connection-oriented transport: TCP**
  - Connection management;
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
- Principles of congestion control
- TCP congestion control
- QUIC



# TCP Sequence Numbers, ACKs

## Sequence numbers:

- Byte stream – use “**number of first byte** in segment’s data

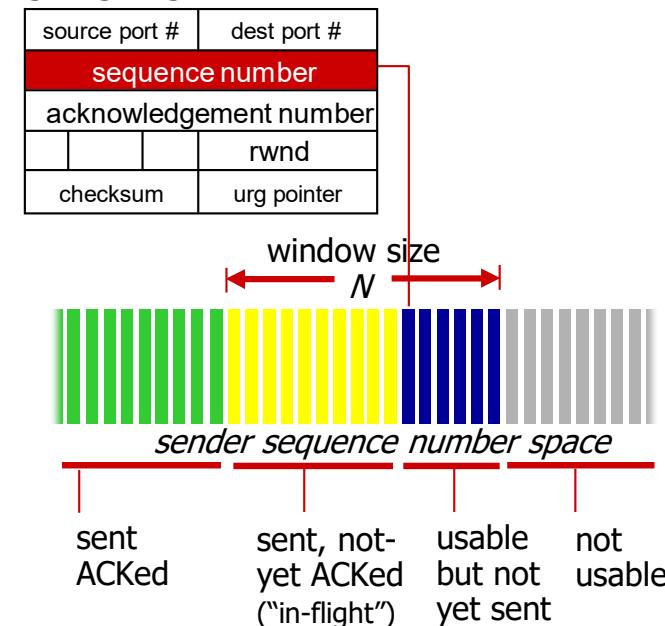
## Acknowledgements:

- Seq # of next byte expected** from other side
- Cumulative ACK

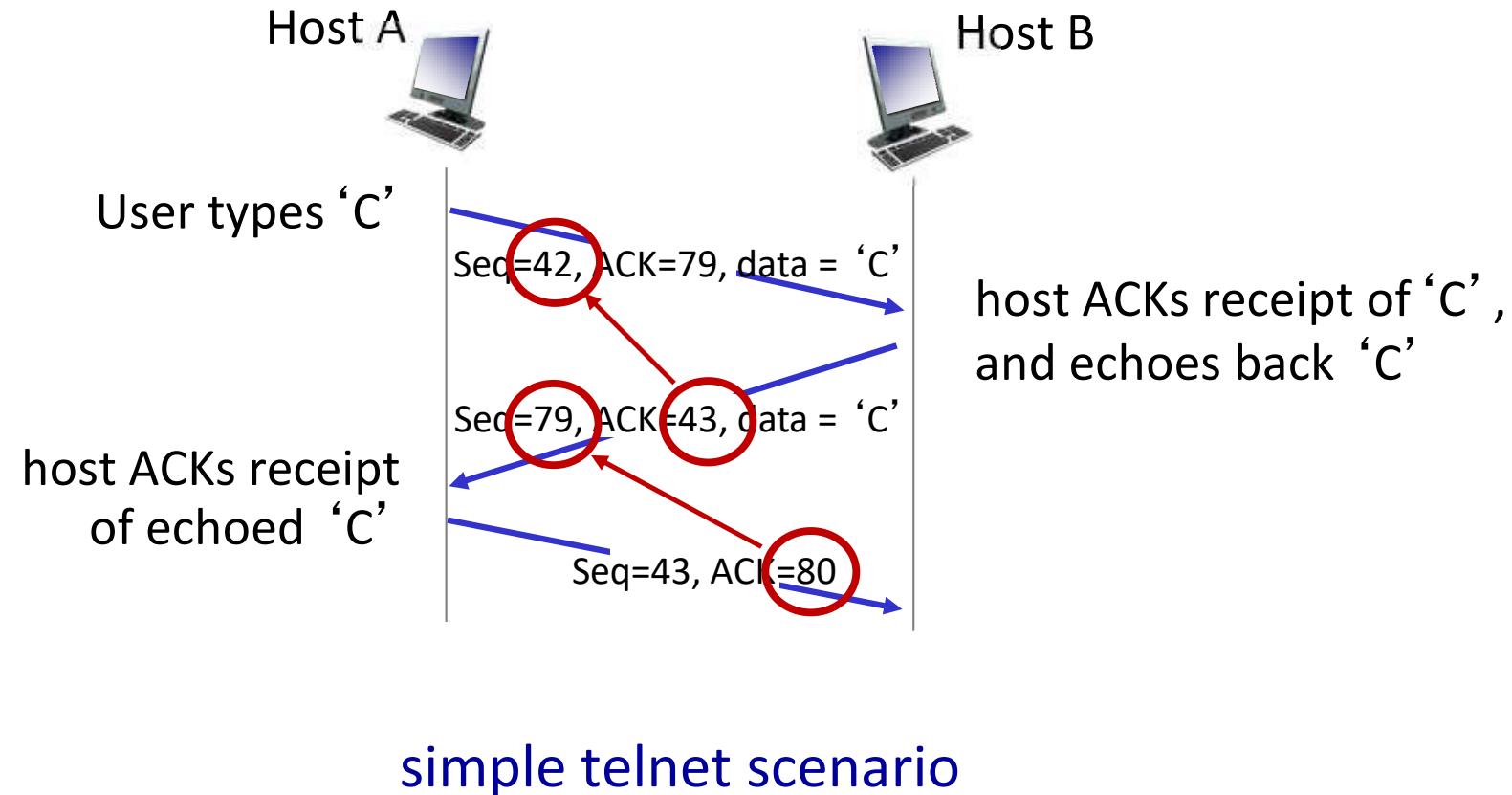
### outgoing segment from receiver

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

### outgoing segment from sender



# TCP Sequence Numbers, ACKs



# TCP Round Trip Time *and* Timeout

## How to set TCP timeout value?

- Longer than RTT:
  - But RTT varies...
- Too short: premature timeout – unnecessary retransmissions.
- Too long: slow reaction to segment loss.

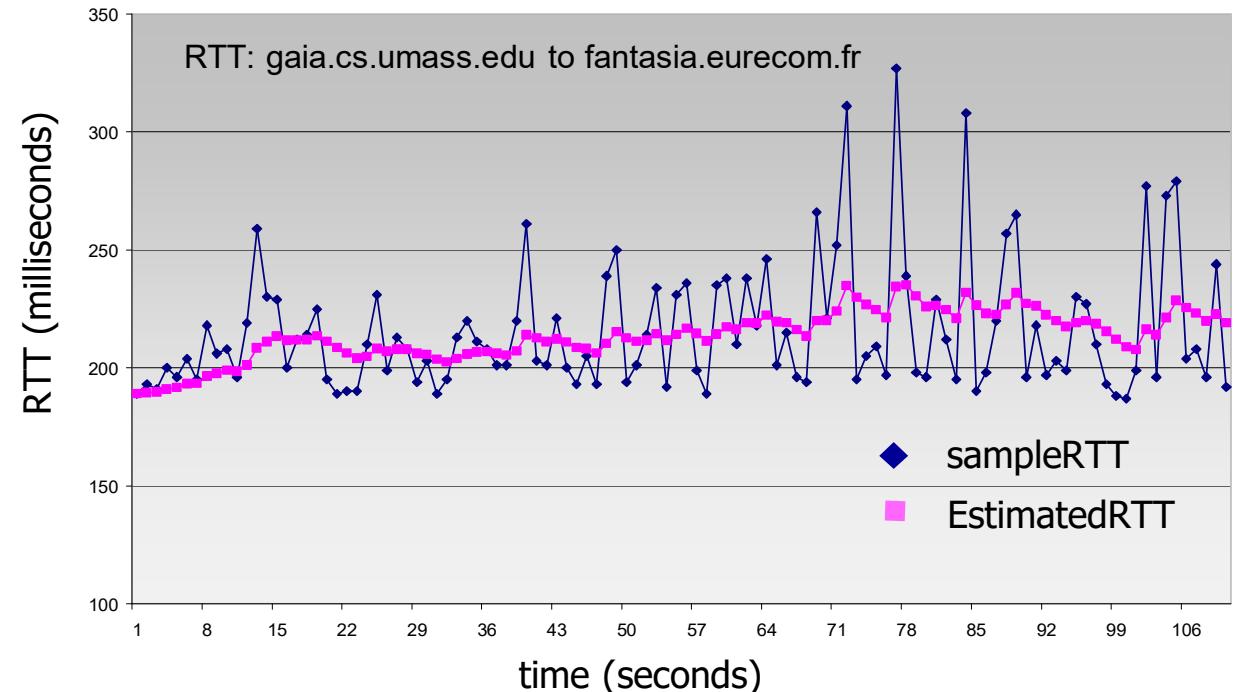
## How to estimate RTT?

- **SampleRTT**: measured time between segment transmission and ACK receipt.
  - Ignore retransmissions;
- **SampleRTT** varies;  
Smoother RTT estimation (**EstimatedRTT**):
  - **Average several recent measurements, not just current SampleRTT**.

# TCP Round Trip Time, Timeout

$$\text{EstimateRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}_d$$

- Exponential weighted moving average (EWMA)
- Influence of past sample decreases exponentially fast
- Typical value:  $\alpha = 0.125$



# *TCP Round Trip Time, Timeout*

- timeout interval: **EstimatedRTT** plus “safety margin”
  - large variation in **EstimatedRTT**: want a larger safety margin

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



estimated RTT

“safety margin”

- **DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

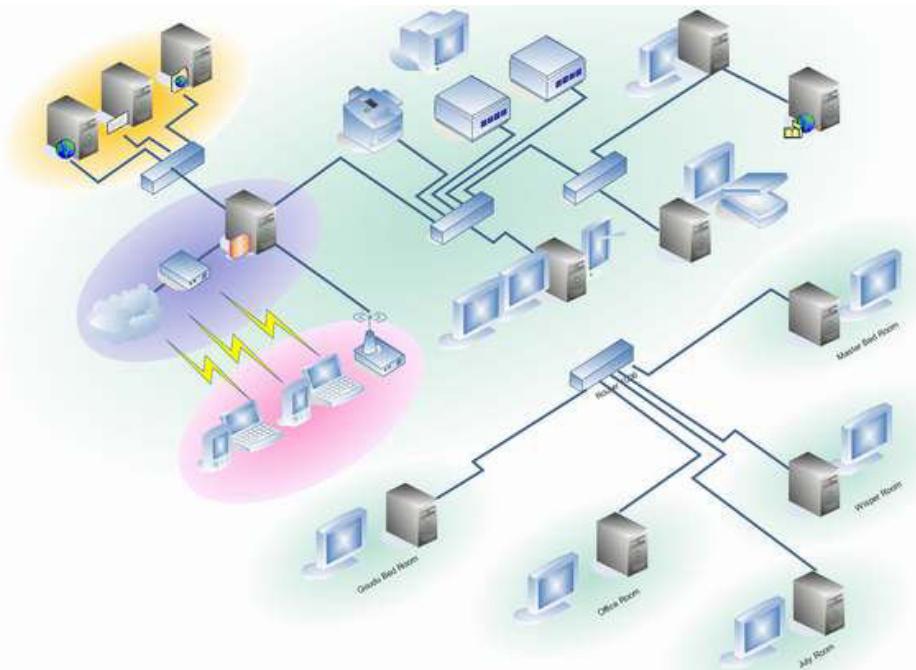
$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically,  $\beta = 0.25$ )

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Connection management;
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
- Principles of congestion control
- TCP congestion control
- QUIC



# TCP Reliable Data Transfer

- TCP creates a reliable data transfer (rdt) service on top of the unreliable service offered by IP;
- TCP uses:
  - Sliding window;
  - Cumulative ACKs;
  - A single **retransmission timer**;
- Retransmissions are triggered by:
  - Timeout events;
  - Duplicate ACKs.

Initially, let's consider a simplified TCP sender:

- Ignore duplicate ACKs
- Ignore flow control and congestion control.

## TCP Sender Events

### Data received from the application layer:

- Create segment;
- Sequence number is the **number of the first data byte** in the segment;
- Start timer (if not yet running – timer is for the oldest unacked segment);
- Expiration interval: **TimeOutInterval**

### Timeout:

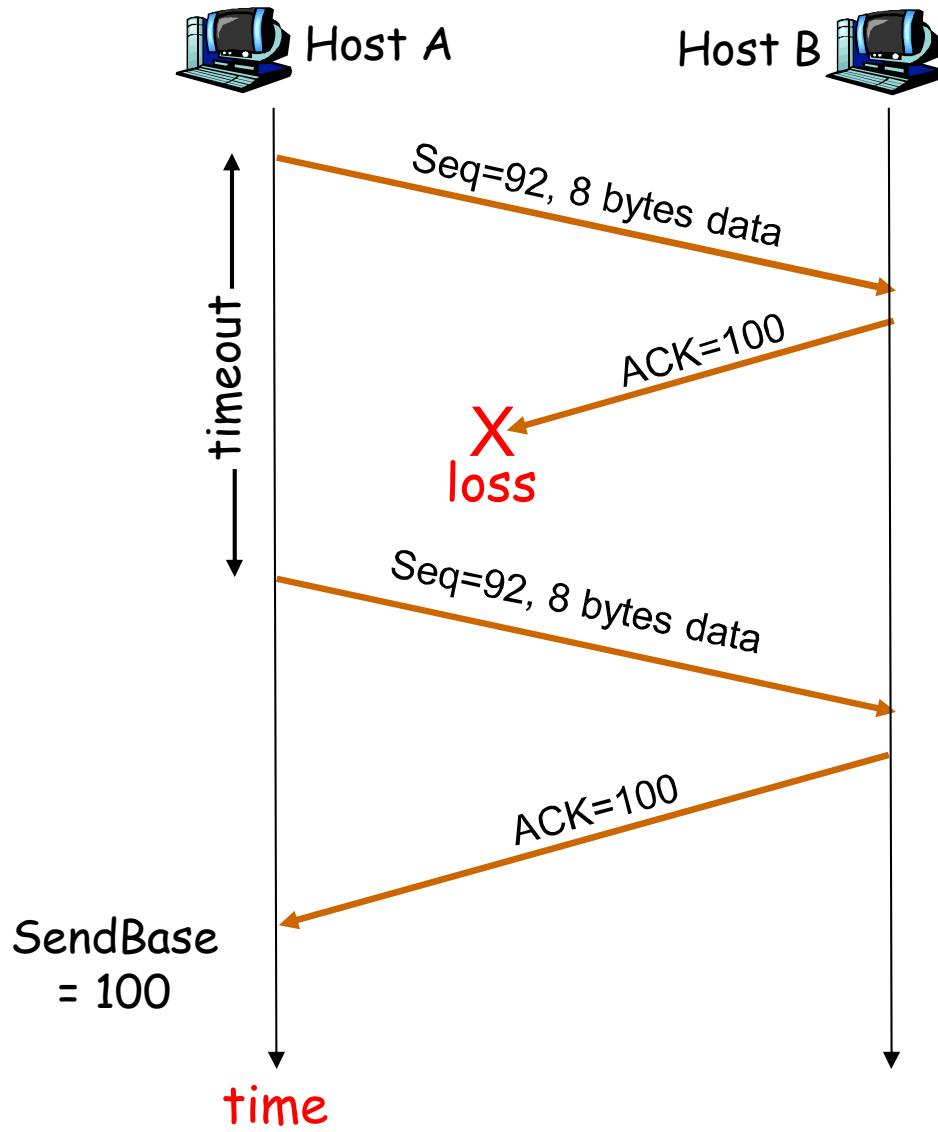
- Retransmit the segment that caused the timeout;
- Restart the timer

### ACK received:

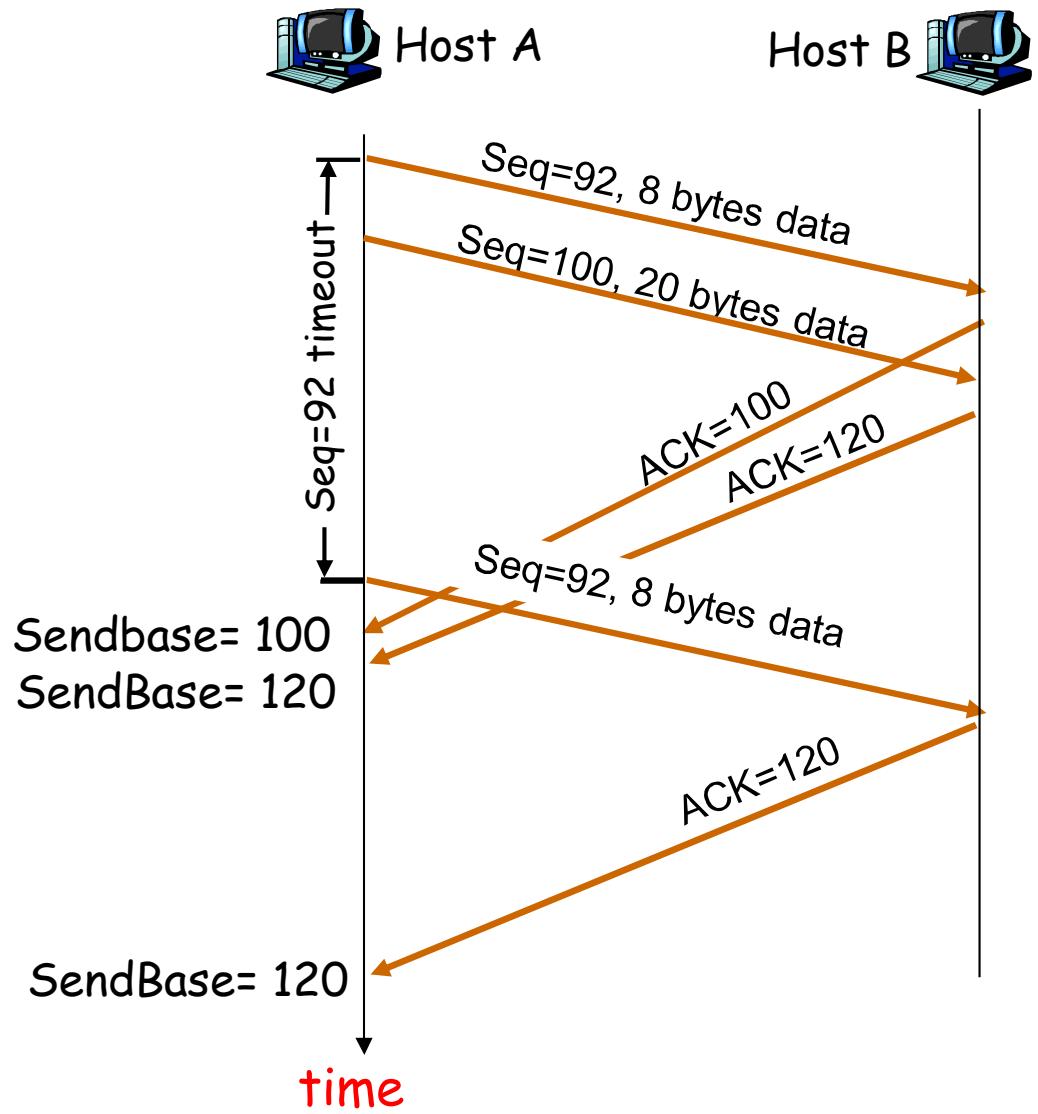
- If previously unacked, segments are acknowledged:
  - Update what is known to be ACKed;
  - Start timer if there are other outstanding segments.

# TCP: Retransmission Scenarios

Lost ACK scenario

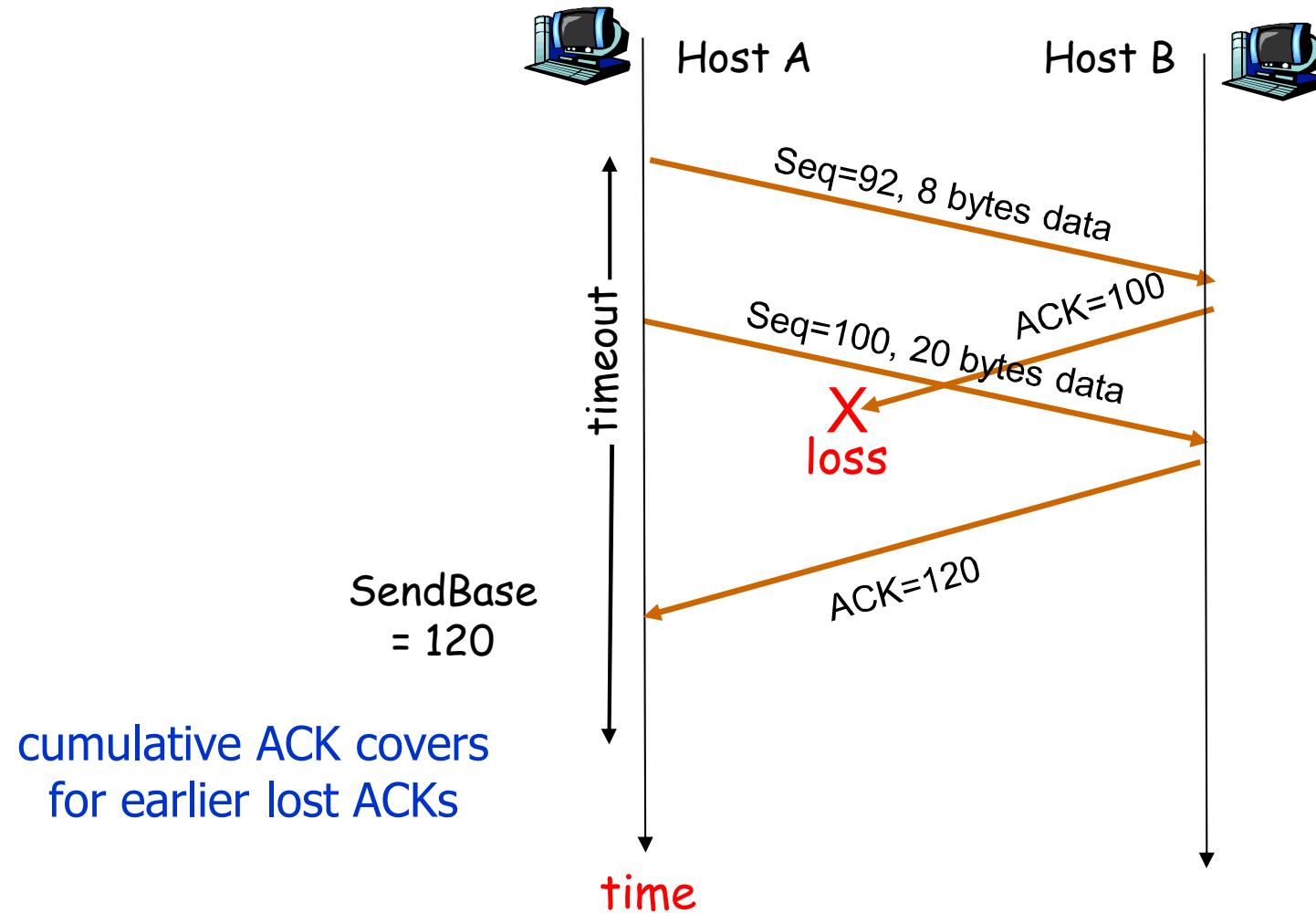


Premature timeout



# TCP: Retransmission Scenarios

Cumulative ACK scenario

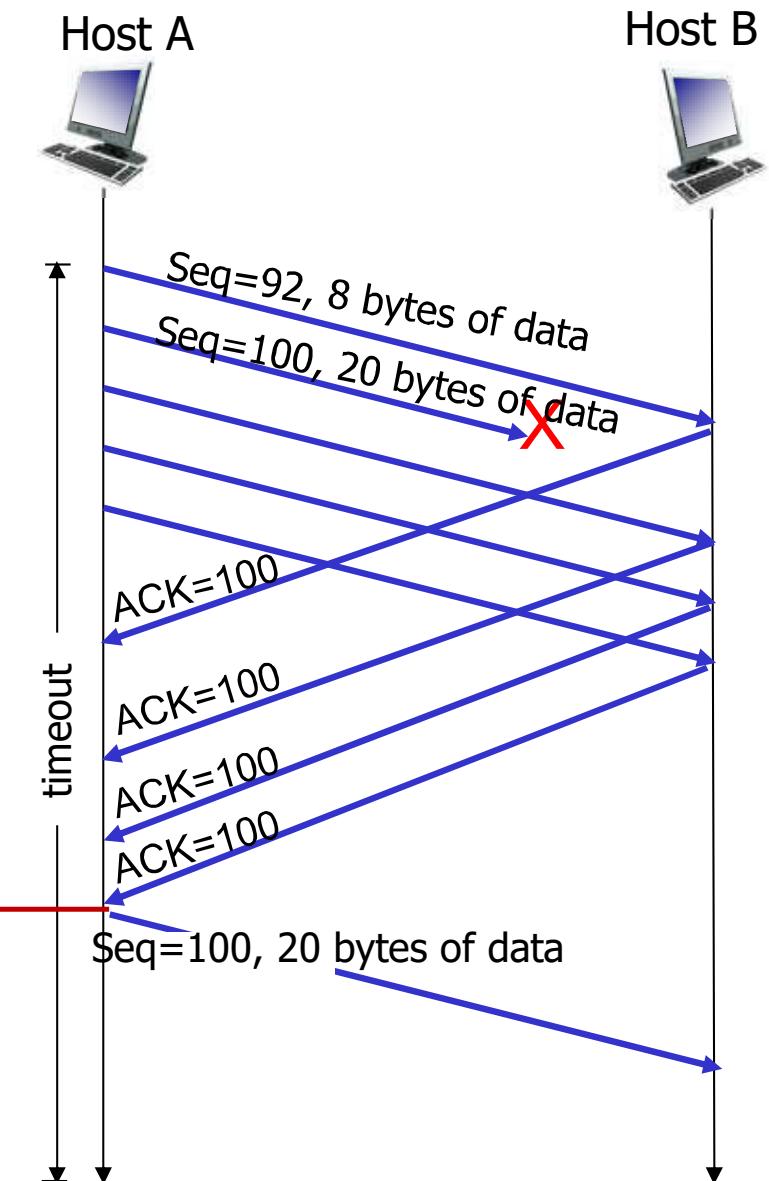


# TCP Fast Retransmit

- Time-out period is relatively long:
- **Detect lost segments via duplicate ACKs.**
  - Sender often sends many data segments;
  - If a segment is lost → many duplicate ACKs.
- If sender receives 3 additional ACKs, assumes the segment (after the ACKed data) was lost:
  - **Fast retransmit:**  
resends segment before timer expiration.



Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. Retransmit!



## TCP ACK Generation [RFC 1122, RFC 2581]

### Event at Receiver

Arrival of **in-order** segment with expected seq #. All data up to expected seq # already ACKed

Arrival of **in-order** segment with expected seq #. One **other segment has ACK pending**

Arrival of **out-of-order** segment higher-than-expect seq. # .  
**Gap detected**

Arrival of segment that **partially or completely fills gap**

### TCP Receiver action

**Delayed ACK.** Wait up to 500ms for next segment. If no next segment, send ACK

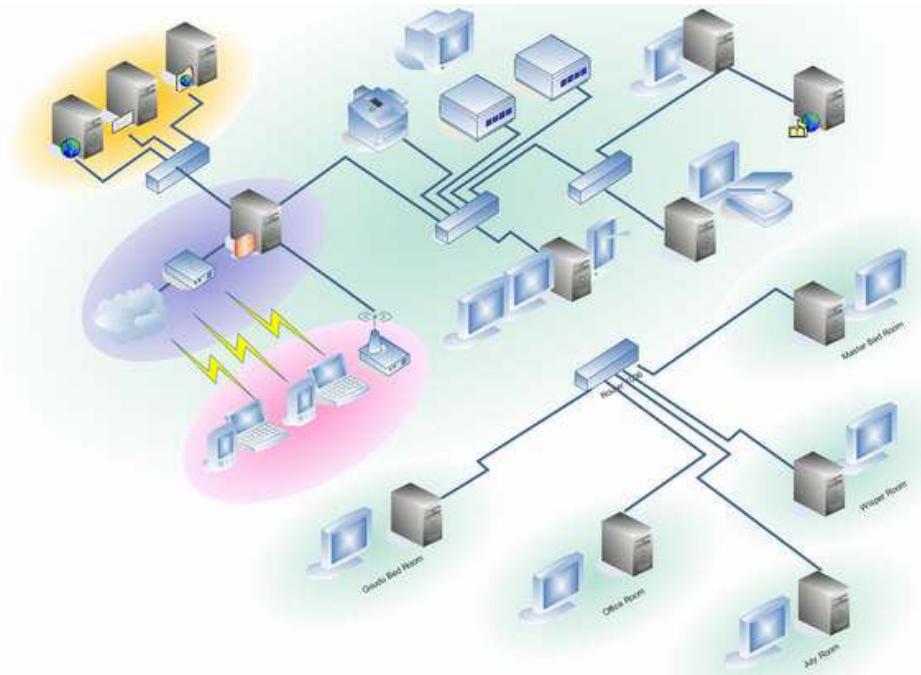
Immediately **send single cumulative ACK**, ACKing both in-order segments

Immediately **send duplicate ACK**, indicating seq. # of next expected byte

Immediately **send ACK**, provided that segment starts at lower end of gap

# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- **Connection-oriented transport: TCP**
  - Connection management;
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
- Principles of congestion control
- TCP congestion control
- QUIC



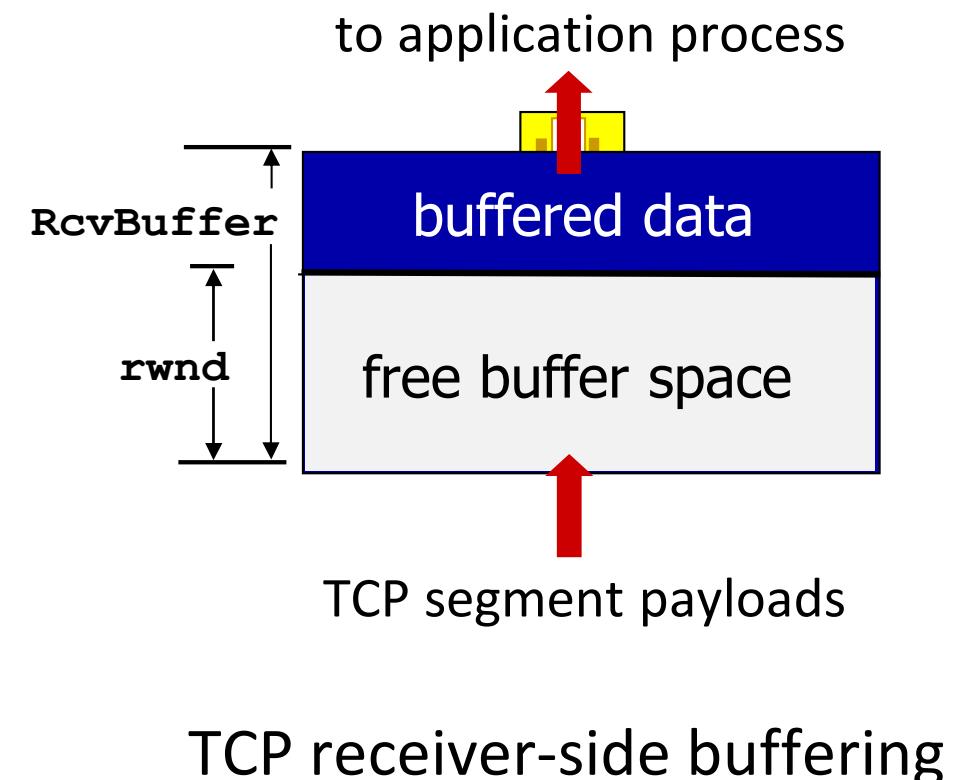
# TCP Flow Control

Goal: ensure that sender won't overflow the receiver's buffer by transmitting too fast.

TCP has a receive buffer:

Application process may be slow at reading from buffer

**Speed-matching**: adjust send rate to the receiving application processing rate



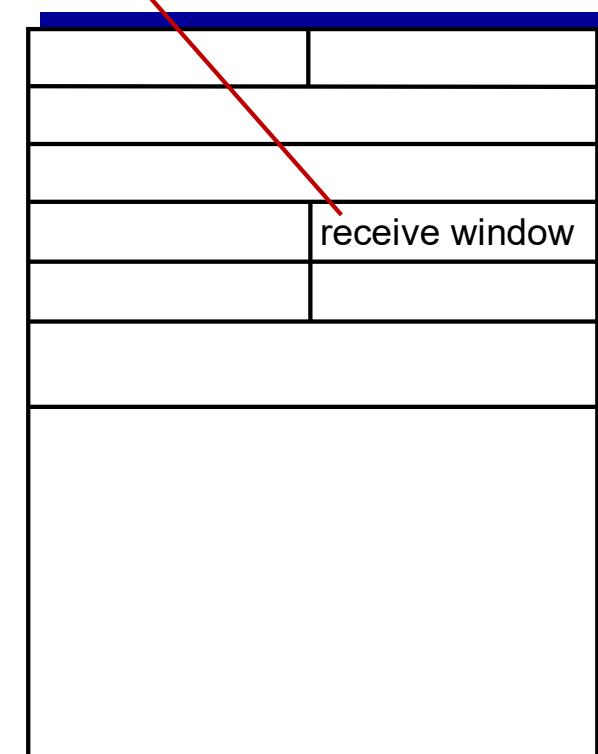
TCP segment payloads

TCP receiver-side buffering

# TCP Flow Control

- TCP receiver “advertises” free buffer space in **rwnd** field in TCP header
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **RcvBuffer**
- Sender limits amount of unACKed (“in-flight”) data to received **rwnd**
  - Guarantees receive buffer will not overflow

flow control: # bytes receiver willing to accept



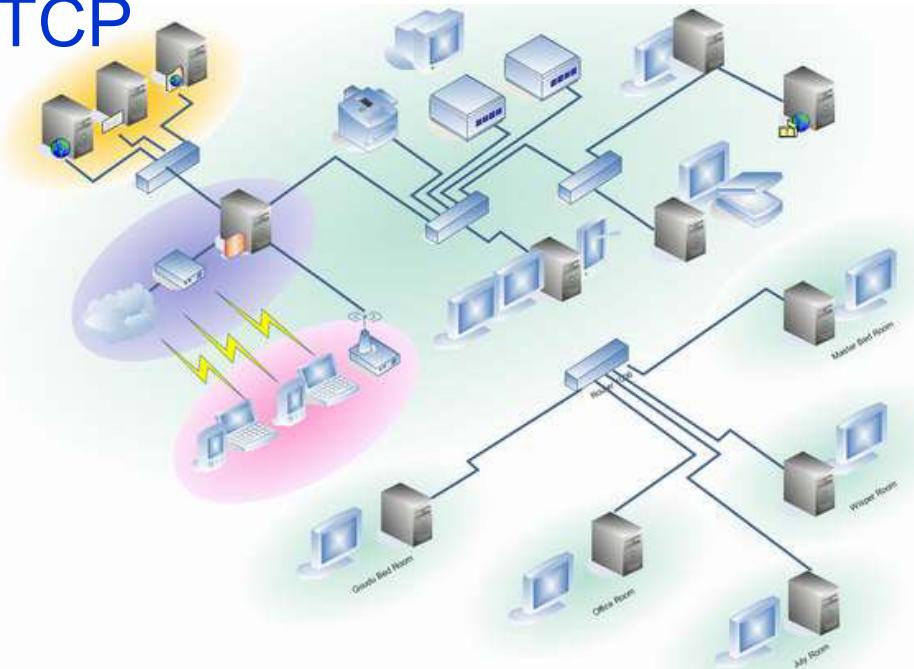
TCP segment format

Test applet at:

[https://media.pearsoncmg.com/aw/ecs\\_kurose\\_compnetwork\\_7/cw/content/interactiveanimations/flow-control/index.html](https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/flow-control/index.html)

# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Connection management;
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
- Principles of congestion control
- TCP congestion control



# Principles of Congestion Control

## Congestion:

- Informally:  
“Too many sources sending **too much data too fast** for the *network* to handle”;
- Different from flow control!
- Manifestations/consequences:
  - **Lost packets** (*buffer overflow at routers*);
  - **Long delays** (*queueing in router buffers*).
- A top-10 problem!

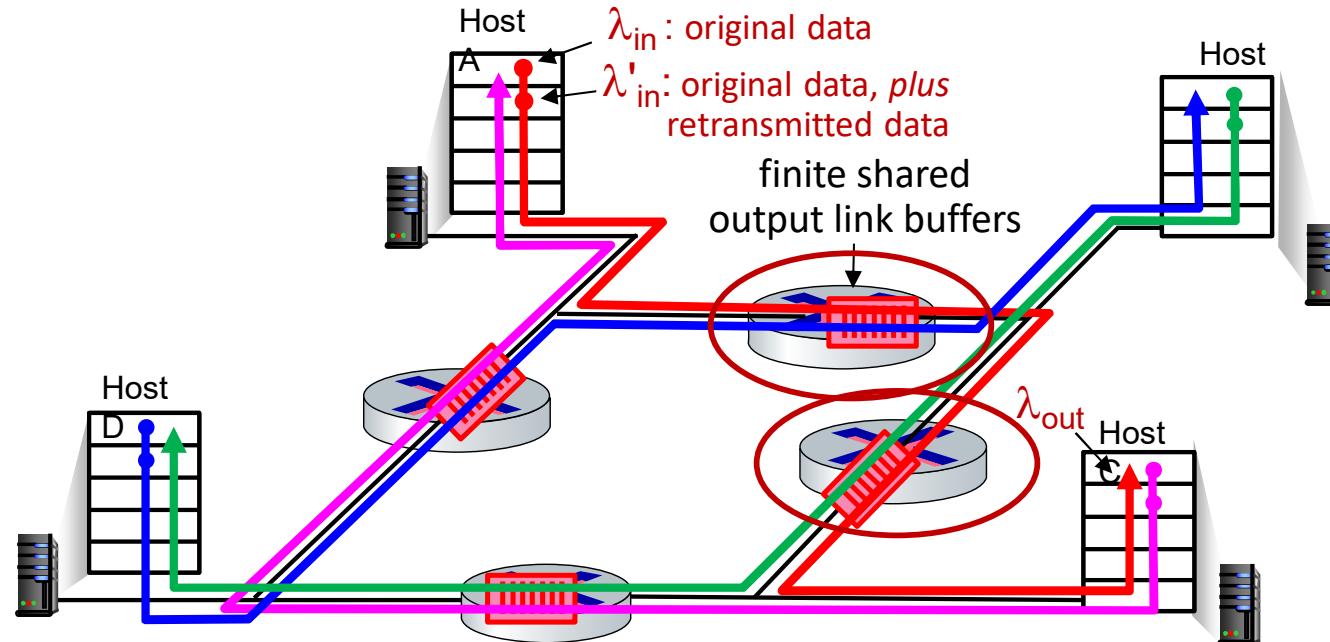


congestion control:  
too many senders,  
sending too fast

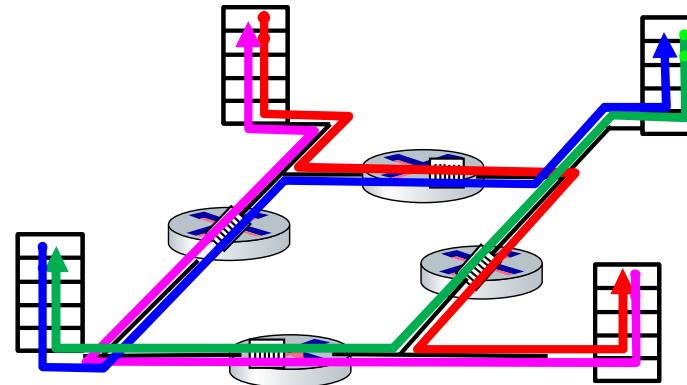
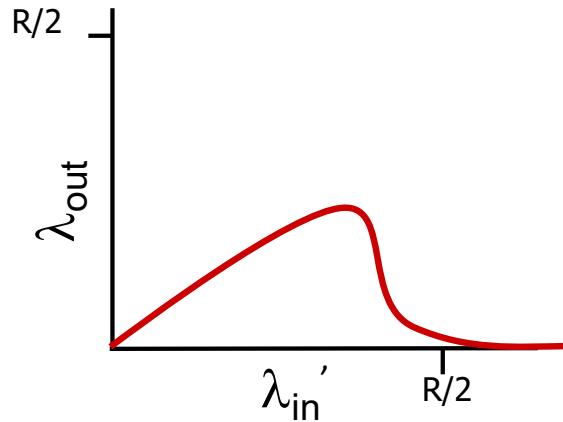
# Causes/Costs of Congestion

- four senders
- multi-hop paths
- timeout/retransmit

As red  $\lambda_{in}'$  increases, all arriving blue pkts at upper queue are dropped, blue throughput  $\rightarrow 0$



# Causes/Costs of Congestion

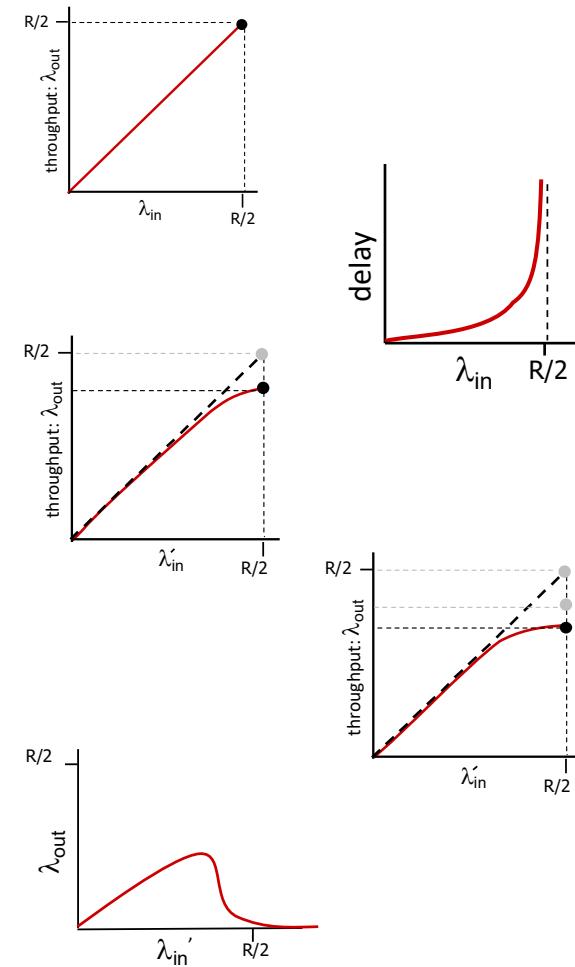


Another “cost” of congestion:

- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

# Causes/costs of Congestion: Insights

- Throughput can never exceed capacity
- Delay increases as capacity approached
- Loss/retransmission decreases effective throughput
- Un-needed duplicates further decreases effective throughput
- Upstream transmission capacity / buffering wasted for packets lost downstream



# *Approaches Towards Congestion Control*

Two approaches towards congestion control:

Network-assisted congestion control:

- Routers provide feedback to end systems:
  - Bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM);
  - Router may inform sender explicitly of supported rate.

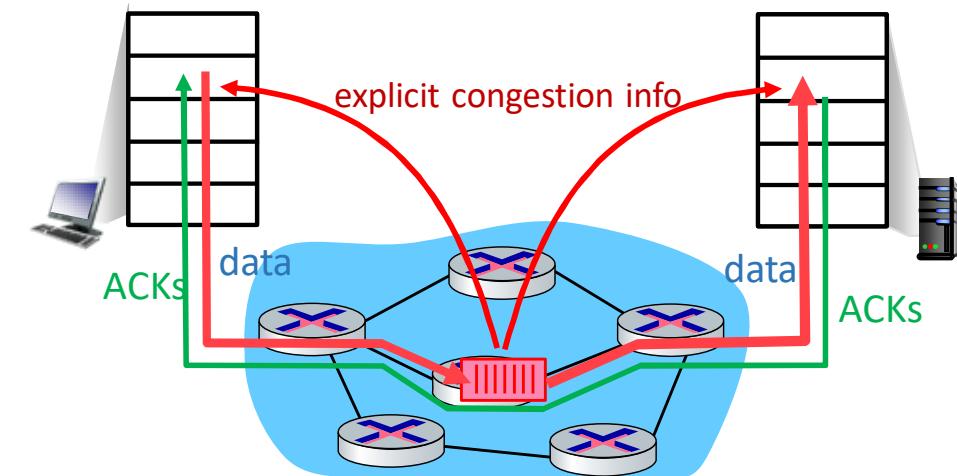
End-end congestion control:

- No explicit feedback from network;
- Congestion inferred from end-system observed loss and delay;
  - It's the approach taken by TCP.

# Approaches Towards Congestion Control

## Network-assisted congestion control:

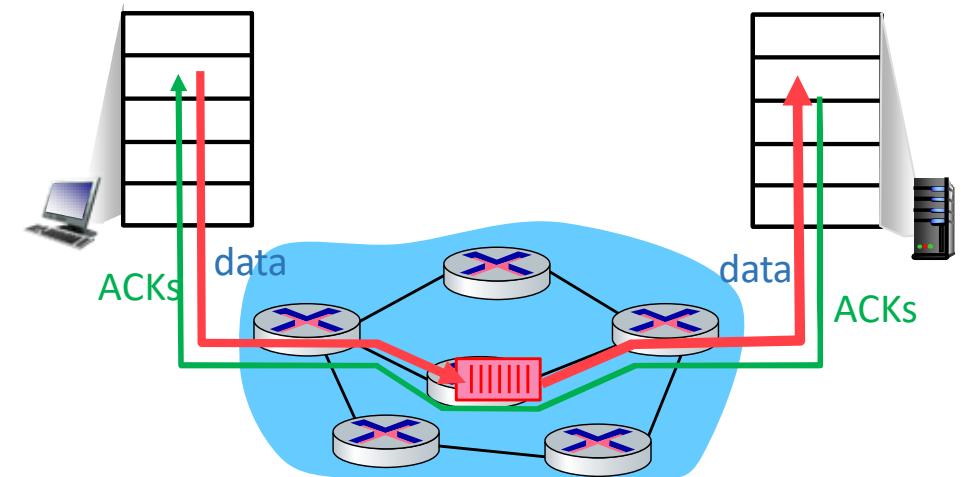
- Routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
  - May indicate congestion level or explicitly set sending rate
- **TCP ECN, ATM, DECbit protocols**



# Approaches Towards Congestion Control

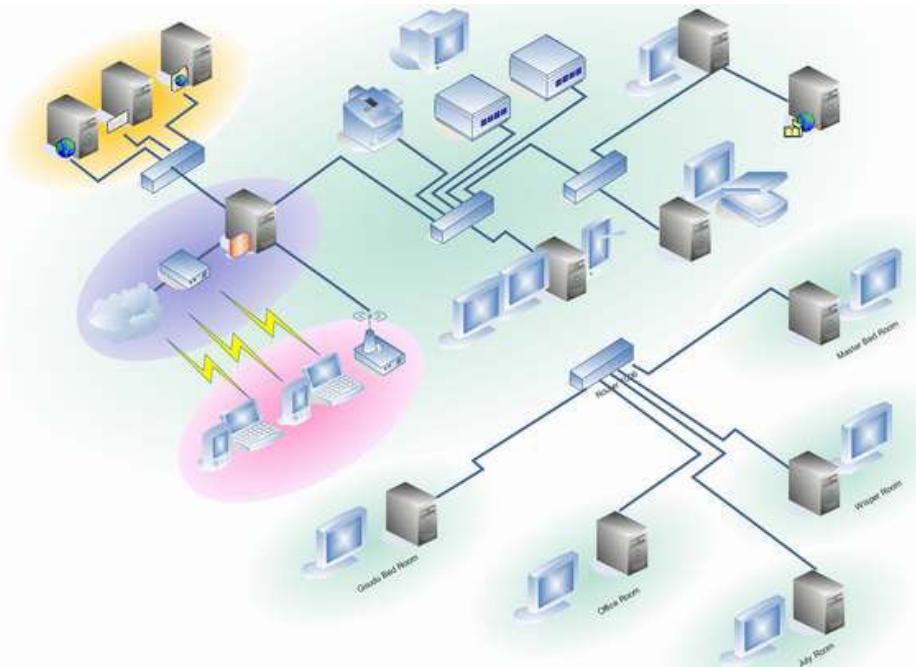
End-end congestion control:

- No explicit feedback from network
- Congestion *inferred* from observed loss, delay
- Approach taken by TCP



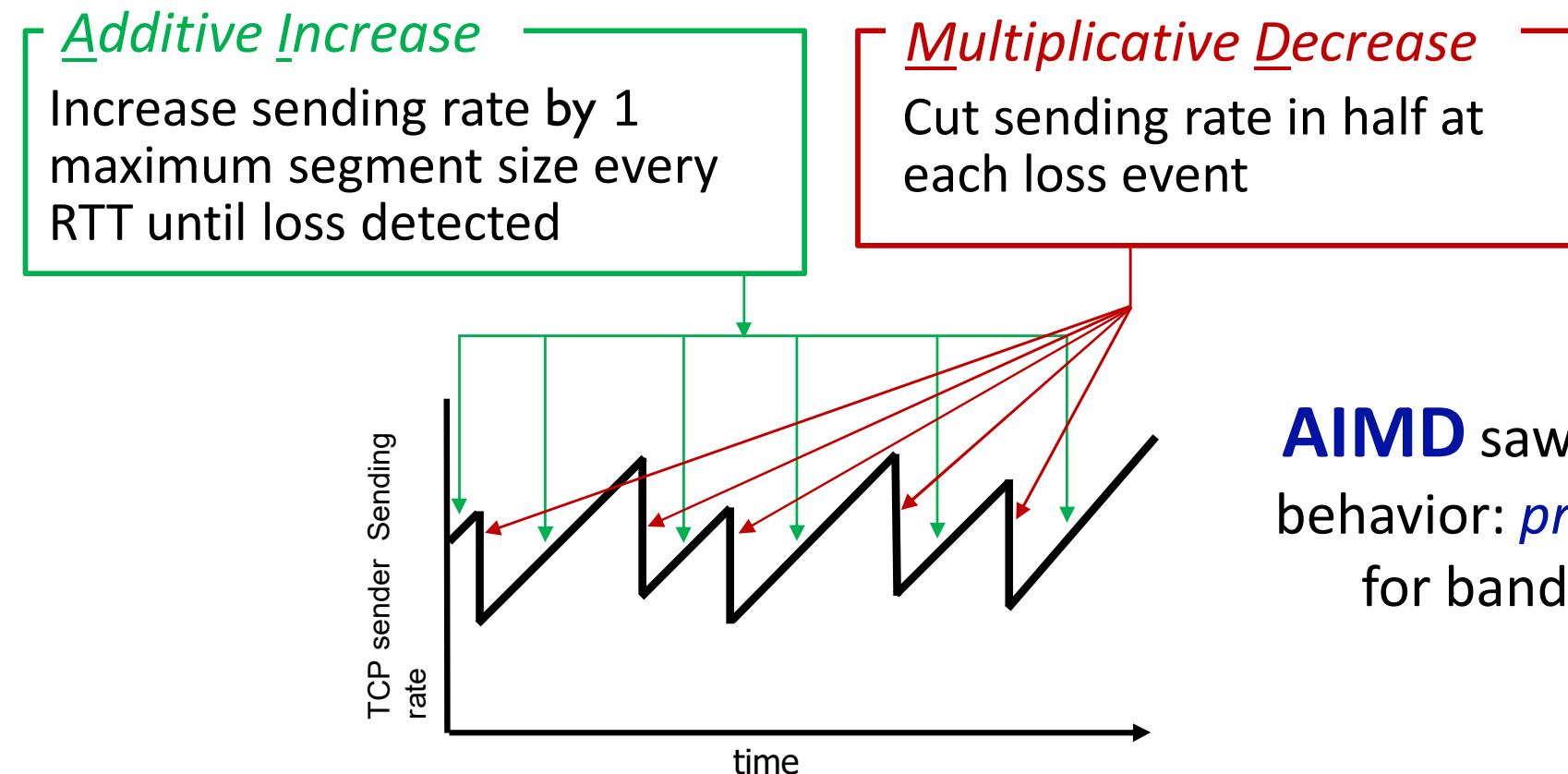
# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Connection management;
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
- Principles of congestion control
- TCP congestion control
- QUIC



# TCP Congestion Control: AIMD

- **Approach:** senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event



## TCP AIMD: more

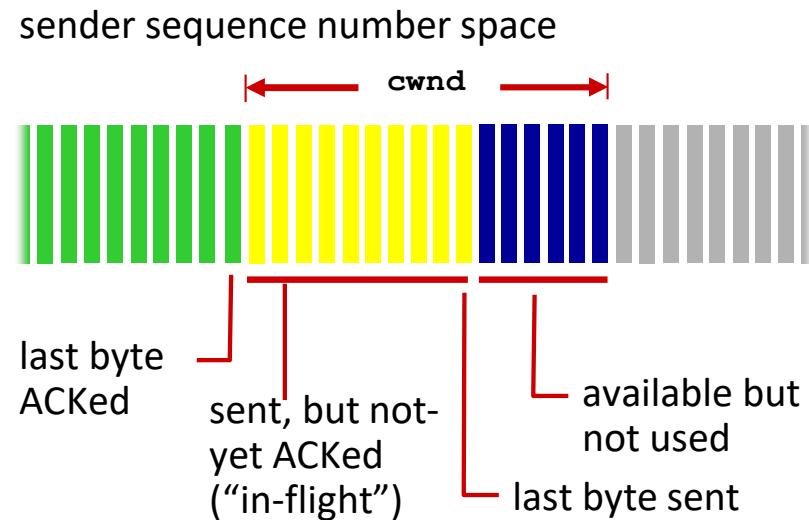
*Multiplicative decrease* detail – sending rate is:

- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

Why AIMD?

- AIMD – a distributed, asynchronous algorithm – has been shown to:
  - Optimize congested flow rates network wide!
  - Have desirable stability properties

# TCP Congestion Control: Details



TCP sending behavior:

- *Roughly*: send  $cwnd$  bytes, wait RTT for ACKS, then send more bytes

$$\text{TCP rate} \approx \frac{cwnd}{RTT} \text{ bytes/sec}$$

- TCP sender limits transmission:  $\text{LastByteSent} - \text{LastByteAcked} \leq cwnd$
- $cwnd$  is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

*typically*  $cwnd < rwnd$

# TCP Congestion Control: Details

## How does sender perceive congestion?

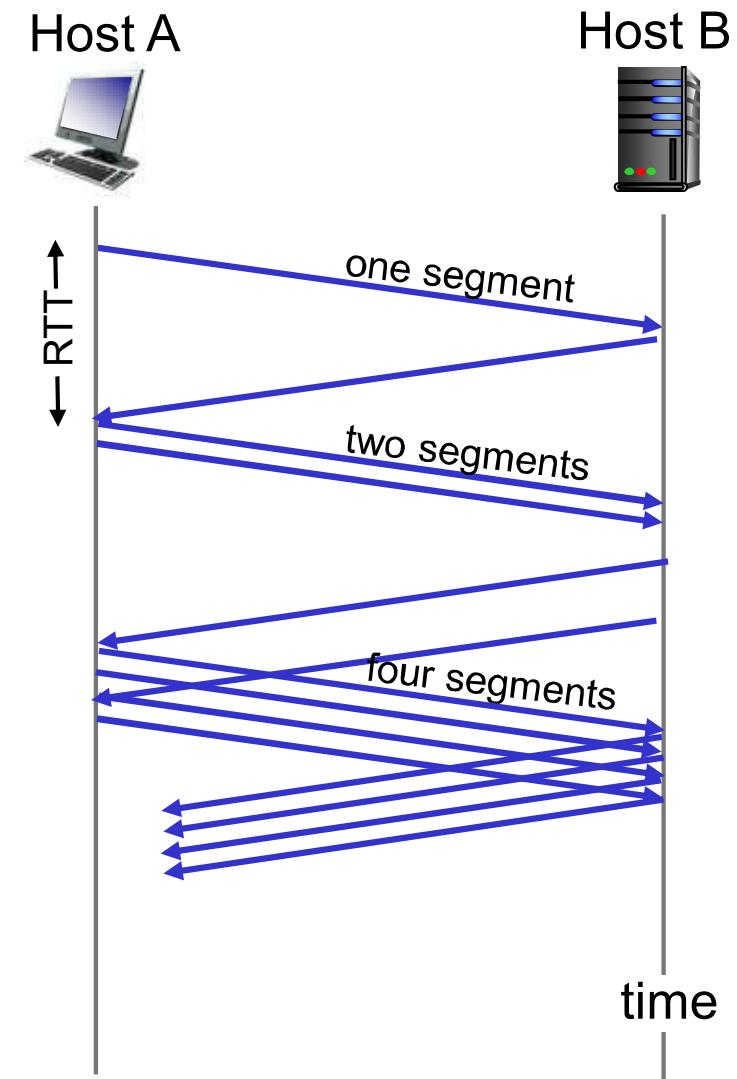
- Loss events:
  - Timeout (TCP Tahoe);
  - 3 duplicate ACKs (TCP Reno).
- TCP sender reduces rate (**cwnd**) after loss event.

## Three mechanisms:

- AIMD (additive increase, multiplicative decrease);
- Slow start;
- Conservative after timeout events.

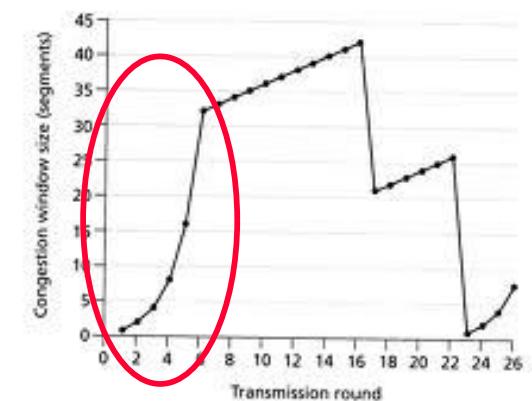
# TCP Slow Start

- When connection begins, rate increases exponentially until first loss event:
  - Initially **cwnd** = 1 MSS
  - Double **cwnd** every RTT;
  - This is done by incrementing **CongWin** of MSS for every ACK received:  
$$\text{cwnd} = \text{cwnd} + \text{MSS}$$
- Slow start: The initial rate is slow but it grows exponentially fast.



## *TCP Slow Start*

- When connection begins,  $cwnd = 1$  MSS
  - Example:  $MSS = 500$  bytes &  $RTT = 200$  msec
  - Initial rate ( $\sim MSS/RTT$ ) = 20 kbps
  - Too slow ...
- Available bandwidth may be  $>>$   $MSS/RTT$ 
  - It is desirable to quickly ramp up to a respectable rate.
- **Slow Start:**
  - When connection begins,  
**rate increases exponentially fast**  
**until first loss event or  $cwnd = Thr$ .**



## Loss Events

- After 3 *duplicate ACKs* (→ **Congestion Avoidance**):
  - **cwnd** is cut in half;
  - Window then grows linearly.
- But after *timeout* (→ **Slow Start**):
  - **cwnd** is set to 1 MSS;
  - The window then grows exponentially up to a threshold, then grows linearly.

Philosophy:

- 3 duplicate ACKs indicate that the network is still capable of delivering some segments;
- A timeout indicates a “more alarming” congestion scenario.

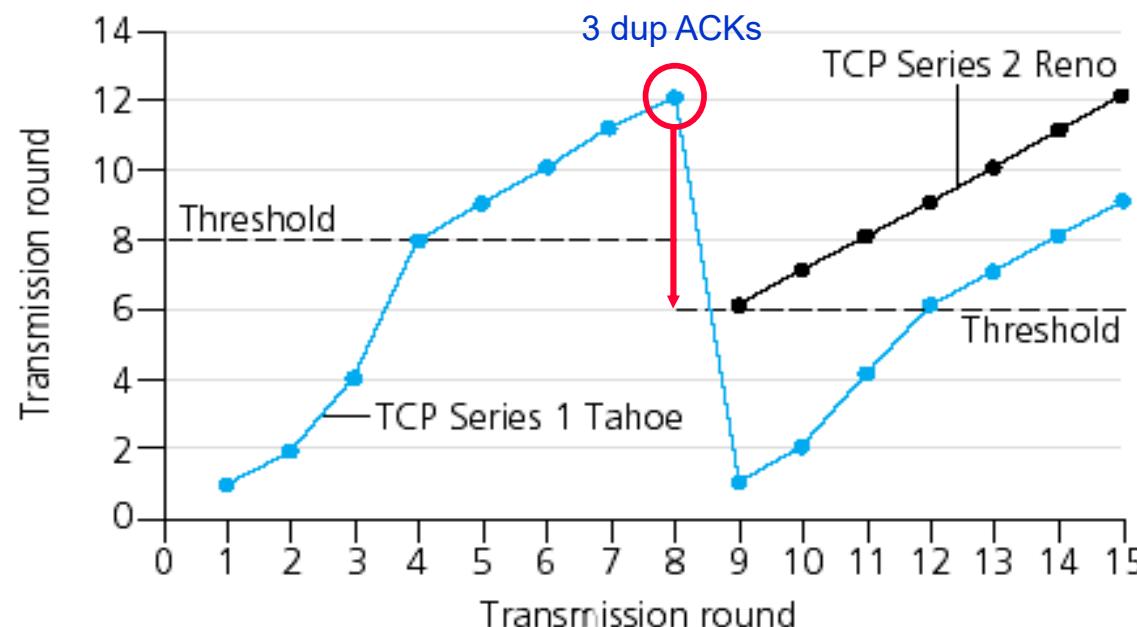
# Loss Events

Q: When should the exponential increase switch to linear?

A: When **cwnd** gets to 1/2 of its value before timeout.

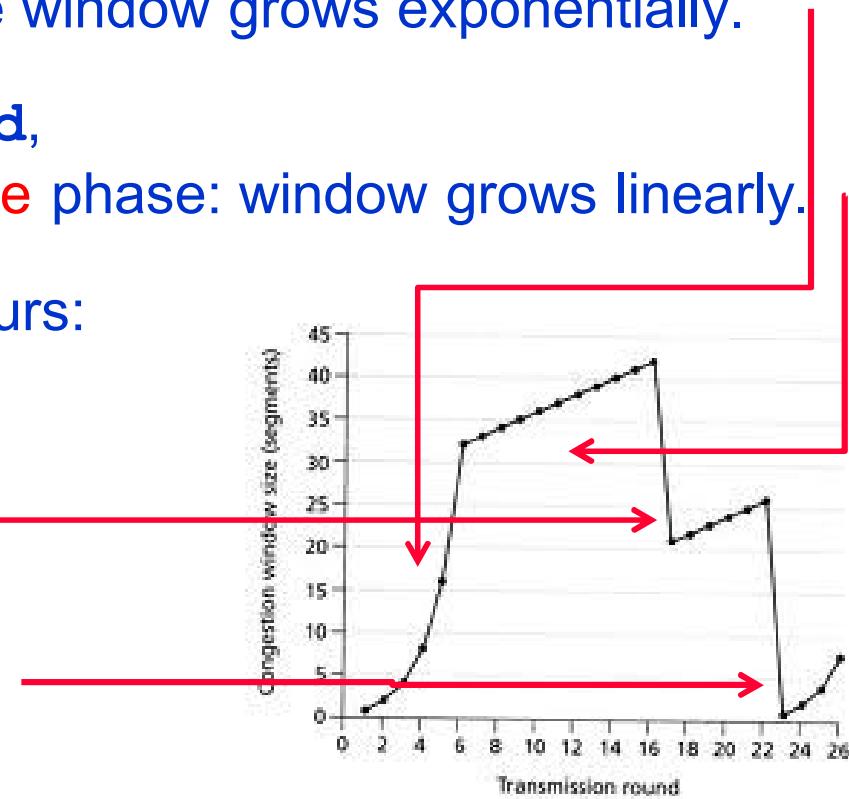
## Implementation:

- Variable **Threshold**;
- At loss event, **Threshold** is set to 1/2 of **cwnd** just before loss event.



# Summary: TCP Congestion Control

- When **cwnd** is below **Threshold**,  
sender is in **slow-start** phase: the window grows exponentially.
- When **cwnd** is above **Threshold**,  
sender is in **congestion-avoidance** phase: window grows linearly.
- When a **triple duplicate ACK** occurs:  
**Threshold** set to **cwnd/2** and  
**cwnd** set to **Threshold**.
- When **timeout** occurs:  
**Threshold** set to **cwnd/2** and  
**cwnd** is set to 1 MSS.



# TCP Sender Congestion Control

## Fast Recovery state

When detecting **three duplicated ACKs** and before moving to congestion avoidance state, *try to speed up recovery* by fast resend of unacknowledged data:

- **cwnd** is increased by 1 MSS for every duplicate ACK received for the missing segment that caused TCP to enter the **fast recovery state**;

When receiving ACK for the missing segment:

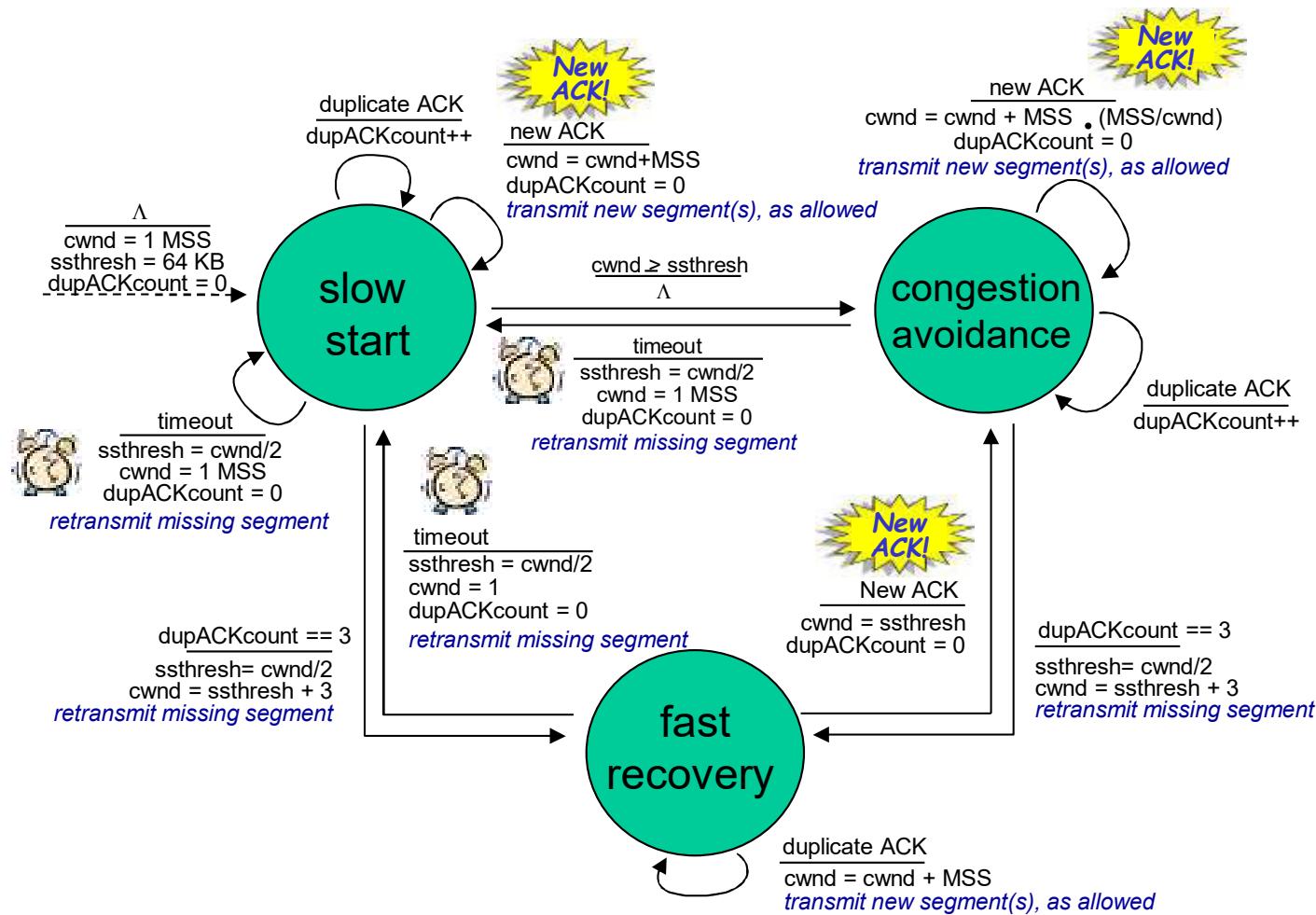
- Move to **congestion avoidance** state (after deflating **cwnd**).

Fast recovery is a recommended, but not required, component of TCP [RFC 5681].

# TCP Sender Congestion Control

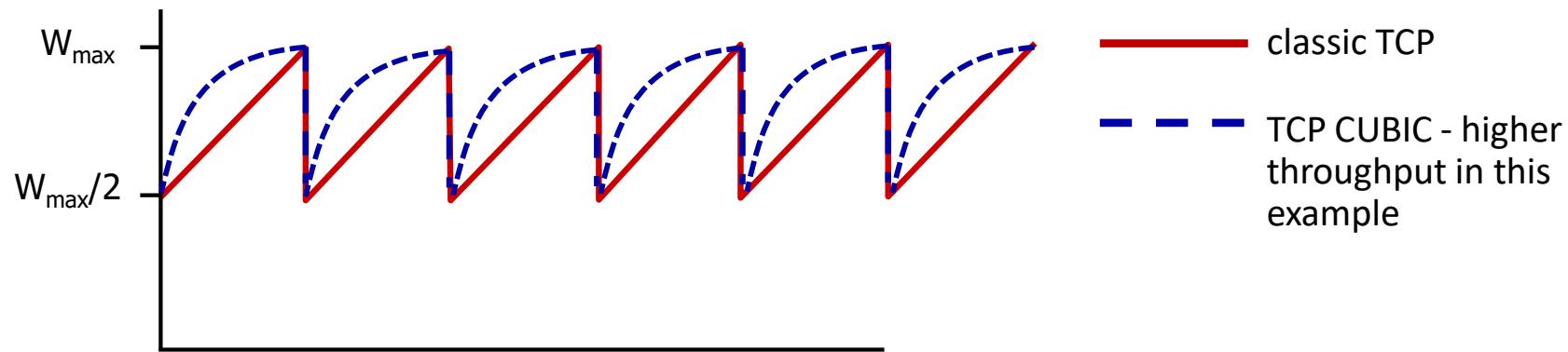
State	Event	TCP Sender Action	Comment
Slow Start (SS)	ACK receipt for previously unacked data	cwnd = cwnd + MSS, If (cwnd > Threshold) state = "Congestion Avoidance"	Double cwnd every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	cwnd = cwnd + MSS*(MSS/cwnd)	Additive increase: cwnd increases 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = cwnd/2, cwnd = Threshold + 3 MSS, State = "Fast Recovery"	Enter fast recovery, implementing multiplicative decrease. cwnd will not drop below 1 MSS.
Fast Recovery (FR)	ACK receipt for the previously unacked data	cwnd = Threshold, dupACKcount=0 State = "Congestion Avoidance"	Exit fast recovery
FR	duplicate ACK	cwnd = cwnd + MSS	FR – increase cwnd
SS, CA or FR	Timeout	Threshold = cwnd/2, <b>cwnd = 1 MSS</b> , State = "Slow Start"	Enter Slow Start
SS or CA	Duplicate ACK	dupACKcount ++	cwnd and Threshold not changed

# *TCP Sender Congestion Control*



# TCP CUBIC

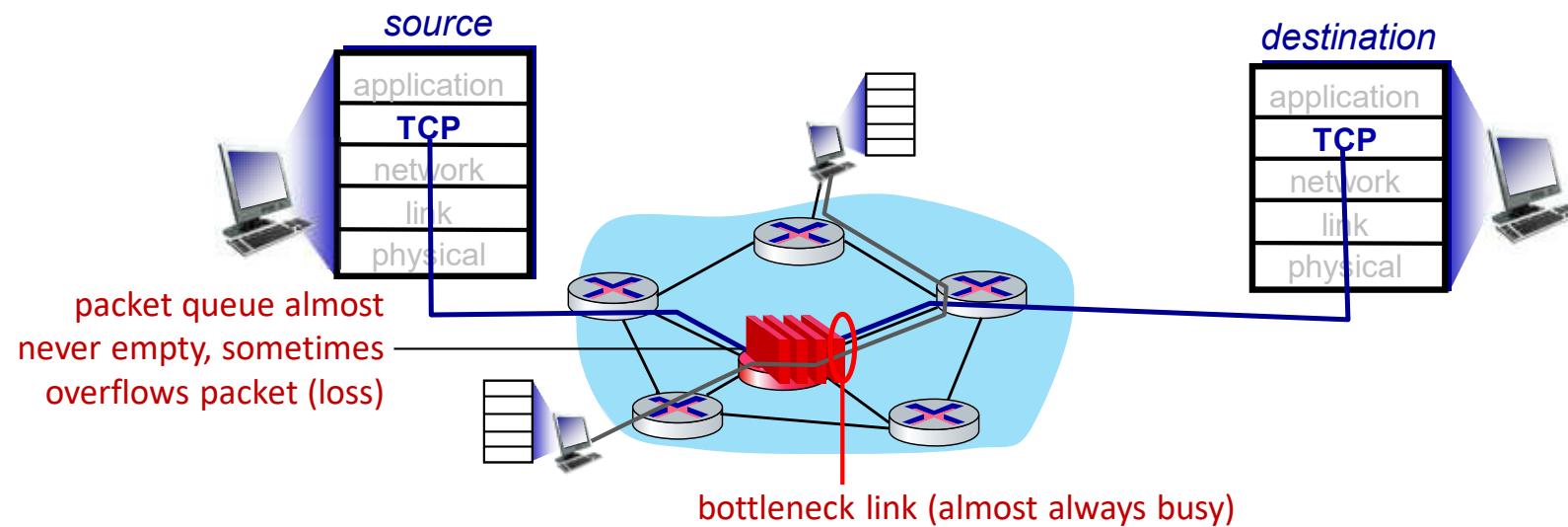
- Is there a better way than AIMD to “probe” for usable bandwidth?
- Insight/intuition:
  - $W_{\max}$ : sending rate at which congestion loss was detected
  - congestion state of bottleneck link probably (?) hasn’t changed much
  - after cutting rate/window in half on loss:
    - start towards  $W_{\max}$  **faster**, but then approach  $W_{\max}$  **slowly**



TCP CUBIC - default in Linux, most popular TCP for popular Web servers

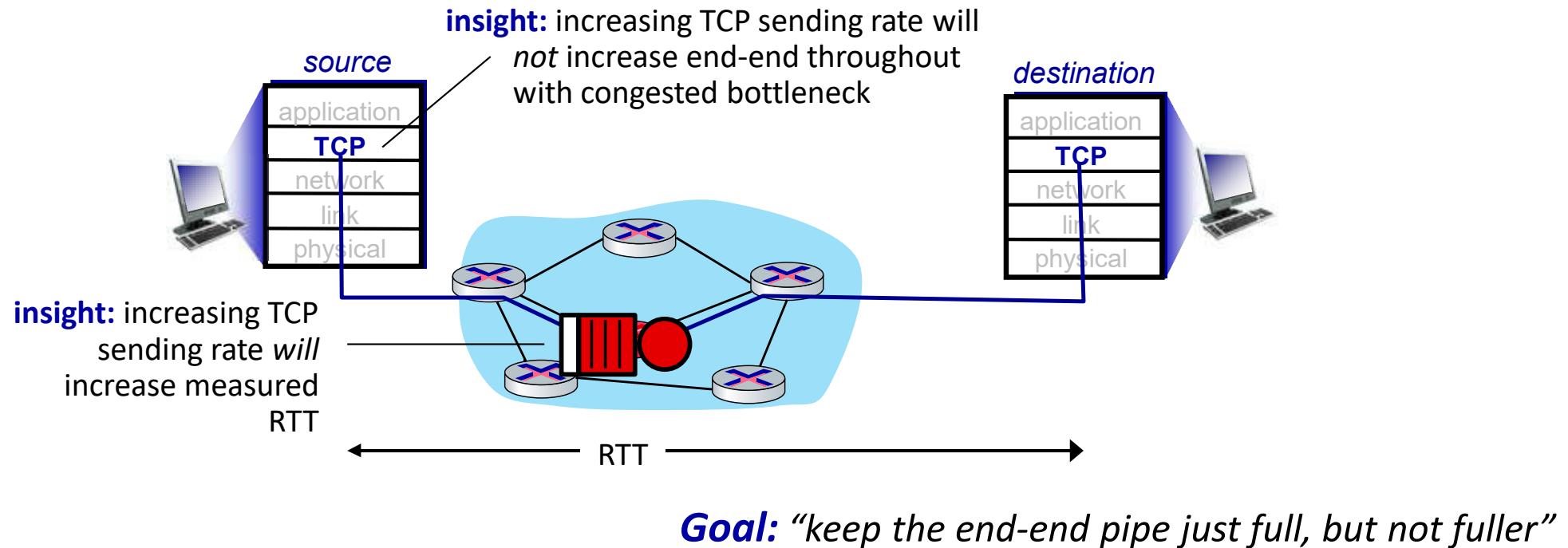
# TCP and Congested “bottleneck link”

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*



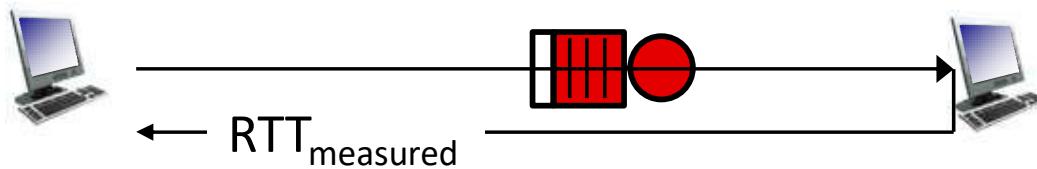
# TCP and Congested “bottleneck link”

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*
- Understanding congestion: useful to focus on congested bottleneck link



# *Delay-based TCP Congestion Control*

Keeping sender-to-receiver pipe “just full enough, but no fuller”: keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\# \text{ bytes sent in last RTT interval}}{\text{RTT}_{\text{measured}}}$$

## Delay-based approach:

- RTT<sub>min</sub> - minimum observed RTT (uncongested path)
  - uncongested throughput with congestion window  $cwnd$  is  $cwnd/RTT_{min}$

if measured throughput “very close” to uncongested throughput  
increase cwnd linearly /\* since path not congested \*/

else if measured throughput “far below” uncongested throughput  
    **decrease cwnd linearly**                   /\* since path is **congested** \*/

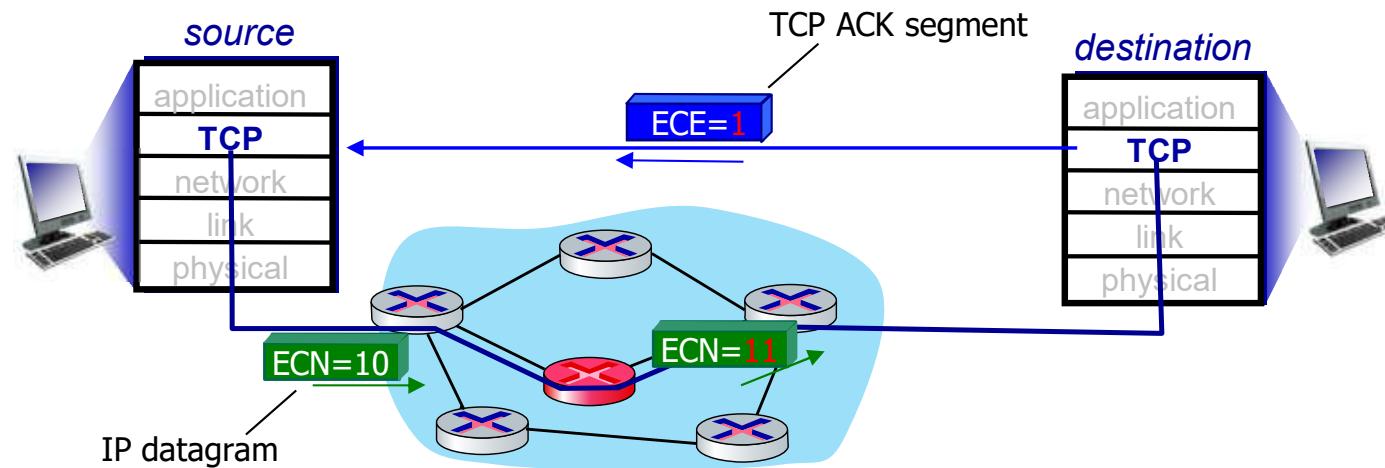
# *Delay-based TCP Congestion Control*

- Congestion control without inducing/forcing loss
- Maximize throughout (“keeping the just pipe full... ”) while keeping delay low (“...but not fuller”)
- A number of deployed TCPs take a delay-based approach:
  - BBR deployed on Google’s (internal) backbone network

# Explicit congestion notification (ECN)

TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



# TCP segment structure

ACK: seq # of next expected byte;  
 A bit: this is an ACK

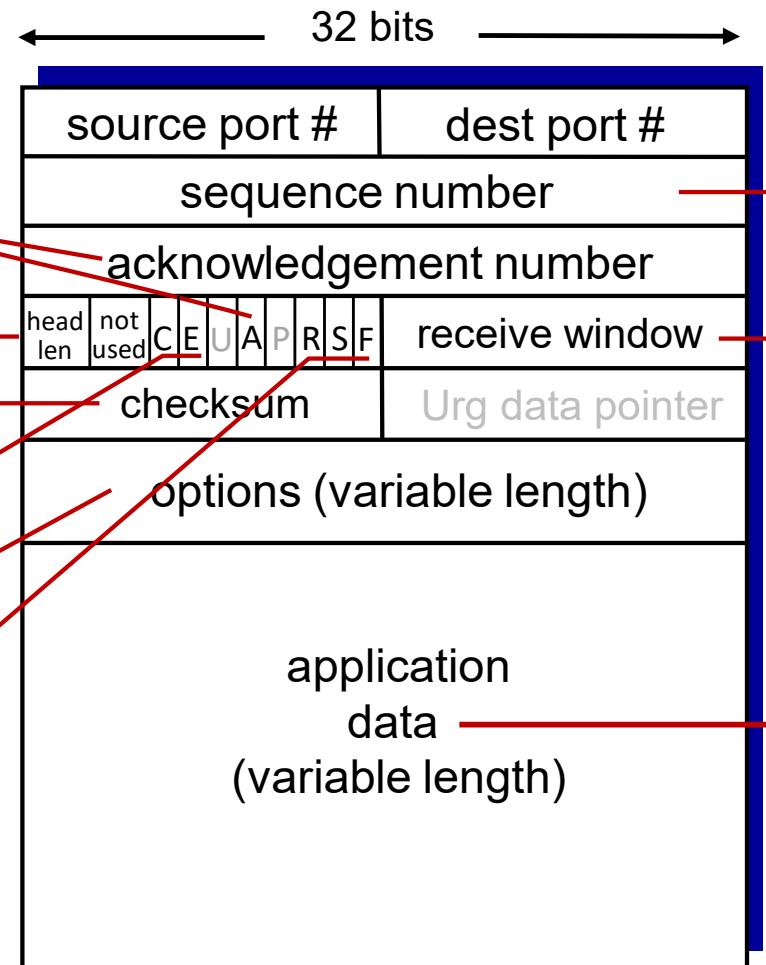
length (of TCP header)

Internet checksum

C, E: congestion notification

TCP options

RST, SYN, FIN: connection management



segment seq #: counting bytes of data into bytestream (not segments!)

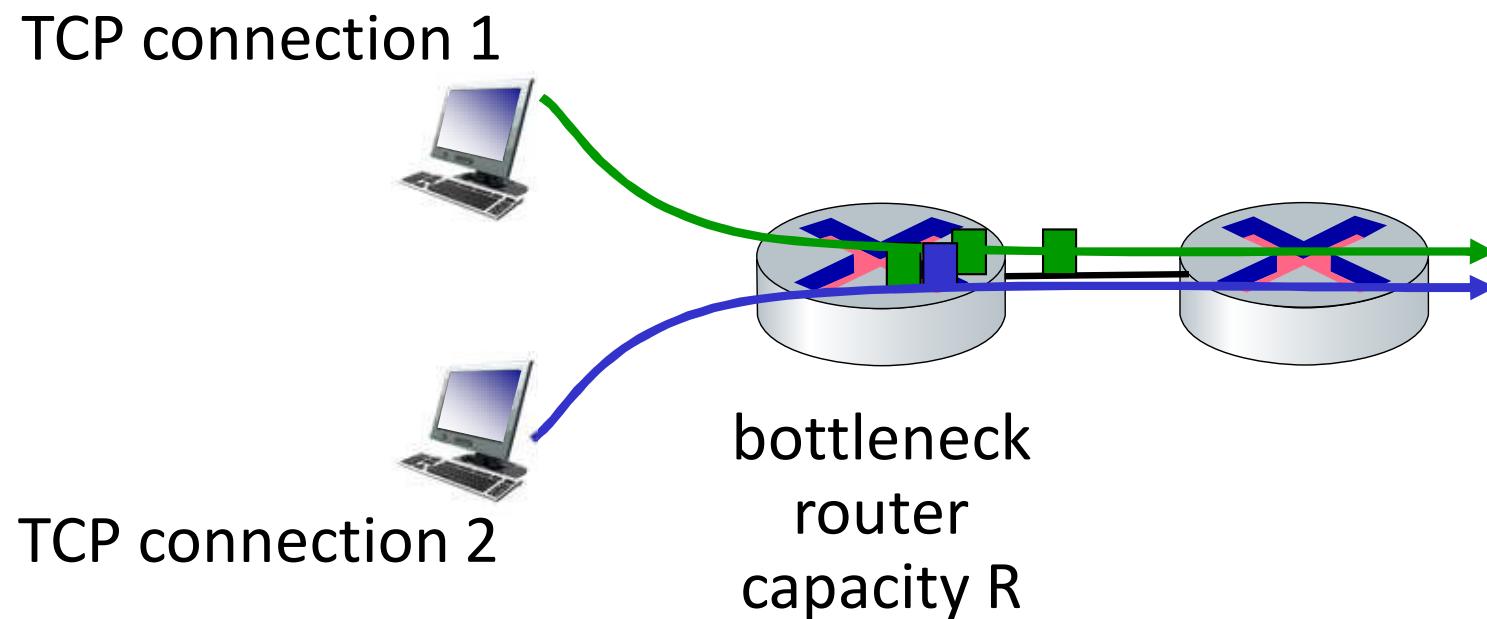
flow control: # bytes receiver willing to accept

data sent by application into TCP socket

# TCP Fairness

Fairness goal:

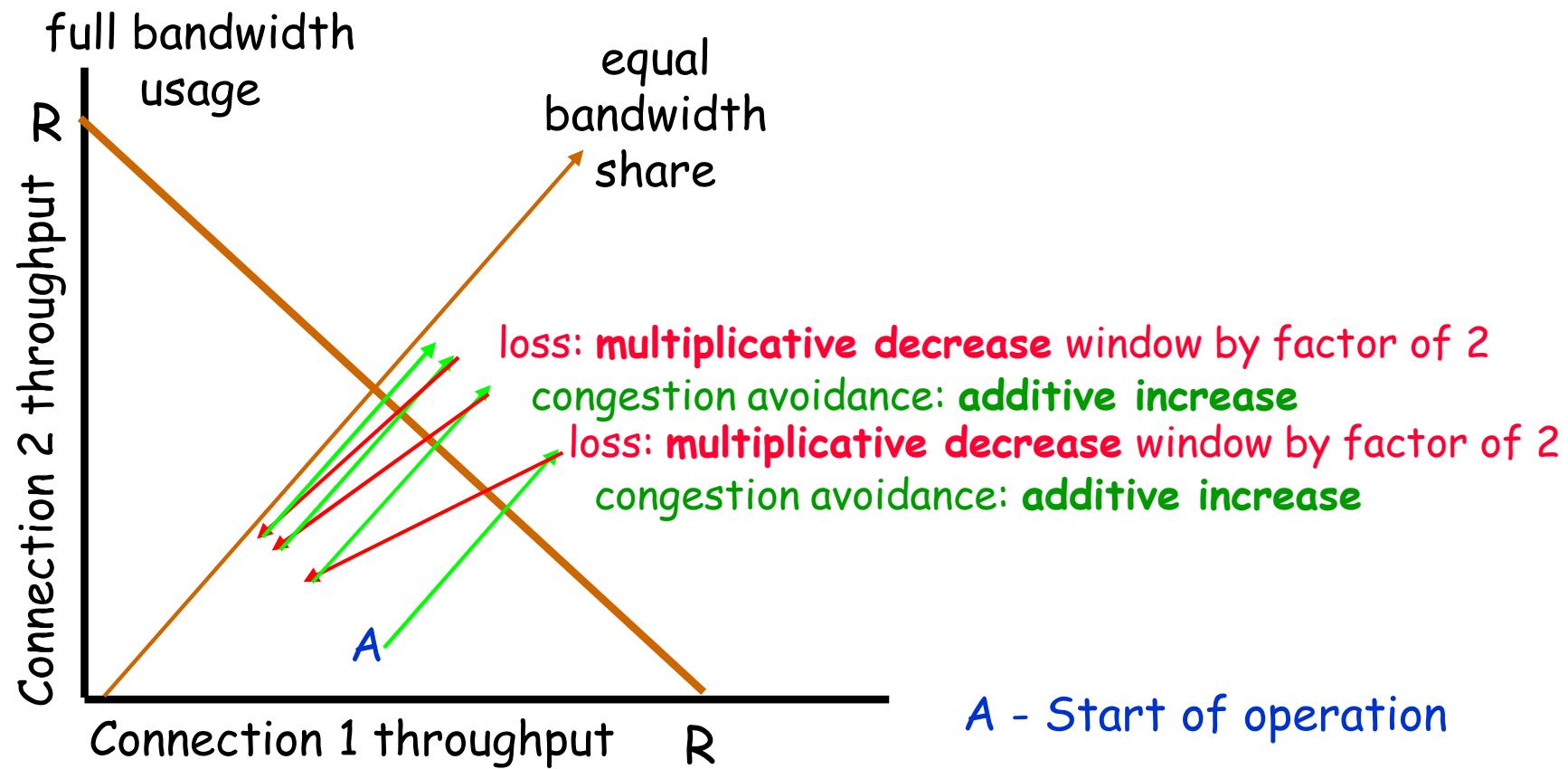
For  $k$  TCP sessions sharing the same bottleneck link, of bandwidth  $R$ , each should have average rate of  $R/k$ .



# Is TCP Fair?

Two competing sessions (both have a large amount of data to send, in CA mode):

- Additive increase gives slope of 1, as throughput increases;
- Multiplicative decrease reduces throughput proportionally.



# Fairness

## Fairness and parallel TCP connections

- Application can open parallel connections between 2 hosts;
- Web browsers do this.
- Example: link of rate R supporting 9 connections;
  - New app asks for 1 TCP, gets rate R/10
  - Another new app asks for 10 TCPs, getting R/2 !

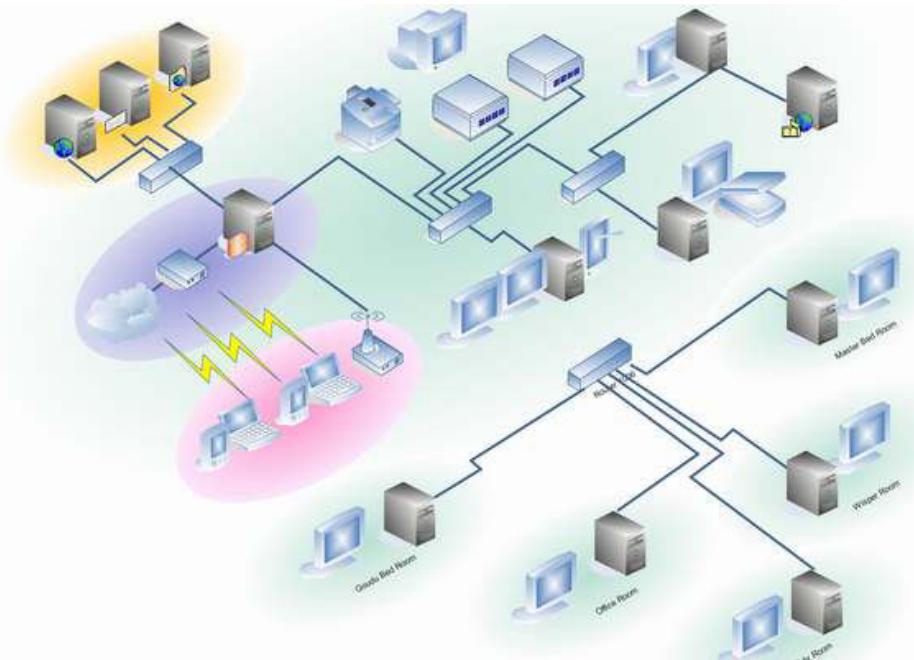
## Fairness and UDP

- Multimedia applications often do not use TCP:
  - Do not want rate restricted by congestion control.
- Instead use UDP:
  - Send audio/video at constant rate and tolerate packet loss.

There is no “Internet police” to check the usage of congestion control.

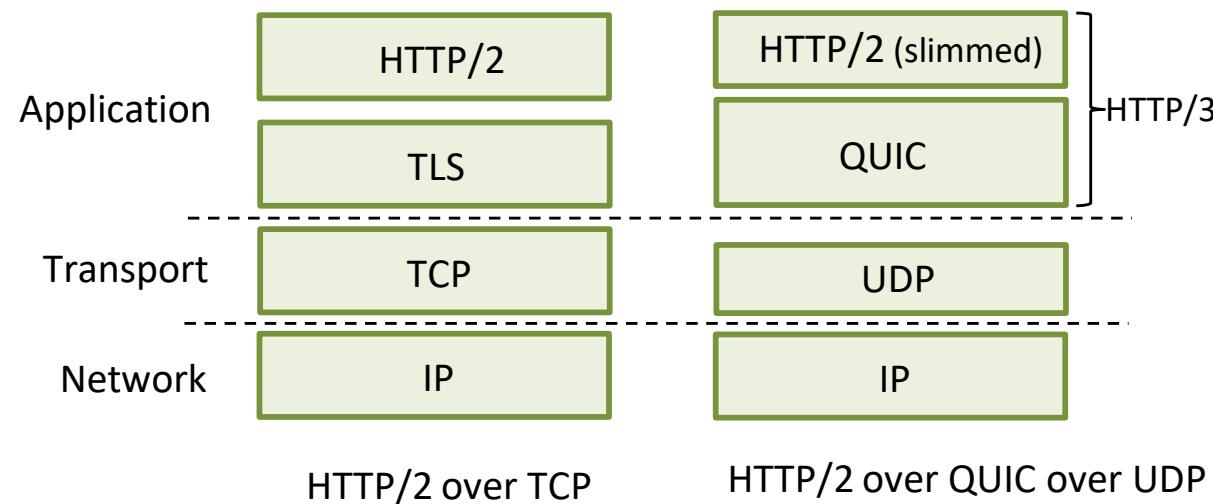
# Outline

- Transport-layer services;
- Multiplexing and demultiplexing;
- Connectionless transport: UDP;
- Principles of reliable data transfer;
- Connection-oriented transport: TCP
  - Connection management;
  - Segment structure;
  - Reliable data transfer;
  - Flow control;
- Principles of congestion control
- TCP congestion control
- QUIC



# QUIC: Quick UDP Internet Connections

- Application-layer protocol, on top of UDP
  - Increase performance of HTTP
  - Deployed on many Google servers, apps (e.g., Chrome, mobile YouTube app)

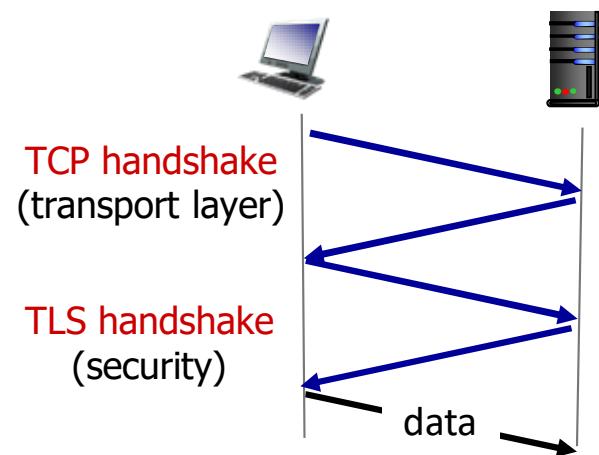


# QUIC: Quick UDP Internet Connections

QUIC:

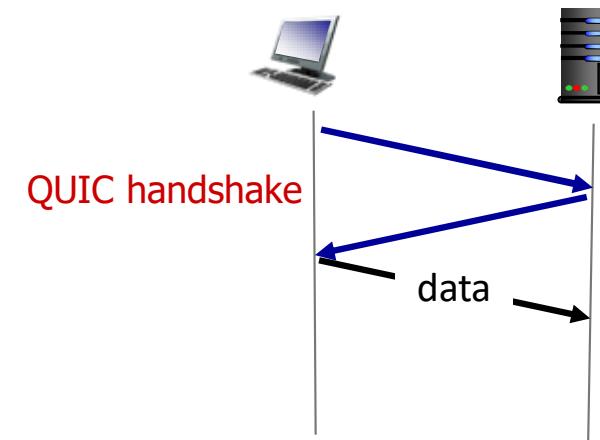
- **Error and congestion control:**  
“Readers familiar with TCP’s loss detection and congestion control will find algorithms here that parallel well-known TCP ones.”  
*[from QUIC specification]*
- **Connection establishment:**  
Reliability, congestion control, authentication, encryption, state – all established in just one RTT
- Multiple application-level “streams” multiplexed over single QUIC connection
  - Separate reliable data transfer, security
  - Common congestion control

# QUIC: Connection Establishment



TCP (reliability, congestion control state)  
+  
TLS (authentication, crypto state)

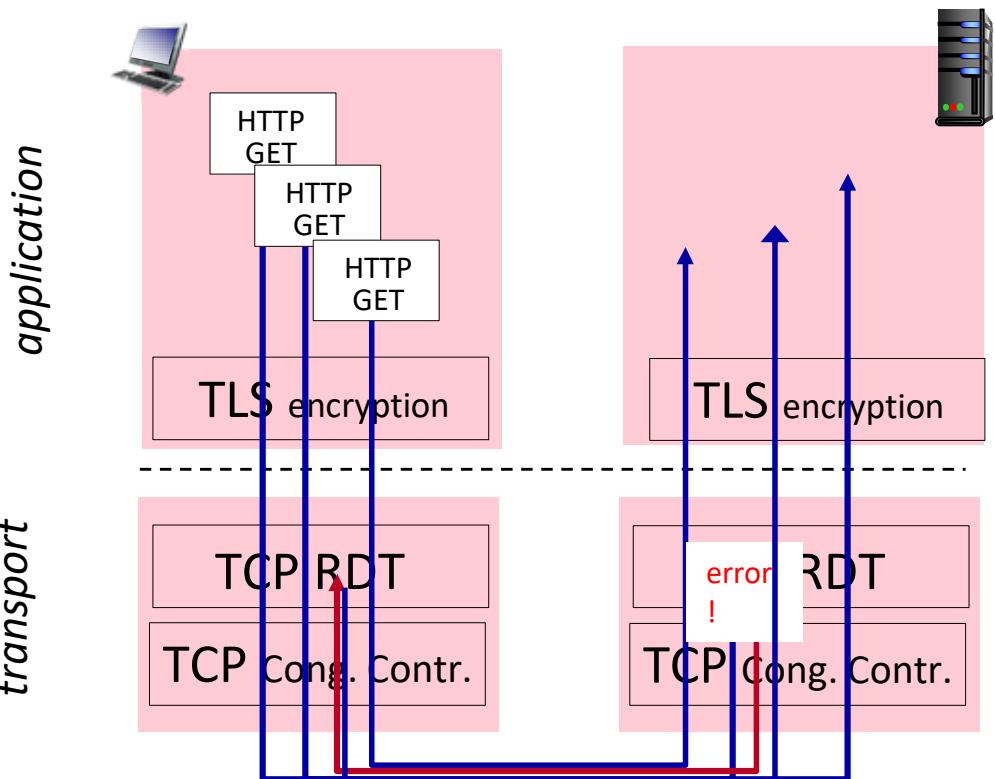
- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

# QUIC Streams: Parallelism, no HOL Blocking



(a) HTTP 1.1

## *Chapter 3: Summary*

- Principles behind transport layer services:
  - Multiplexing and demultiplexing;
  - Reliable data transfer;
  - Flow control;
  - Congestion control.
- Instantiation and implementation in the Internet:
  - UDP;
  - TCP;
  - QUIC.

