

The background features a complex network diagram with numerous nodes (small circles) and connecting lines (edges) in shades of blue, green, and yellow, set against a dark background. The lines and nodes are distributed across the frame, with a denser cluster on the right side.

Lecture Week 2: Networking and Socket Programming

Dr. Hamed Hamzeh

30/01/2025

The Internet: a “nuts and bolts” view



Billions of connected computing *devices*:

- *hosts* = end systems
- running *network apps* at Internet’s “edge”



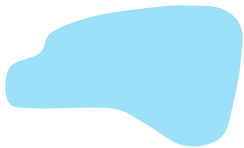
Packet switches: forward packets (chunks of data)

- *routers, switches*



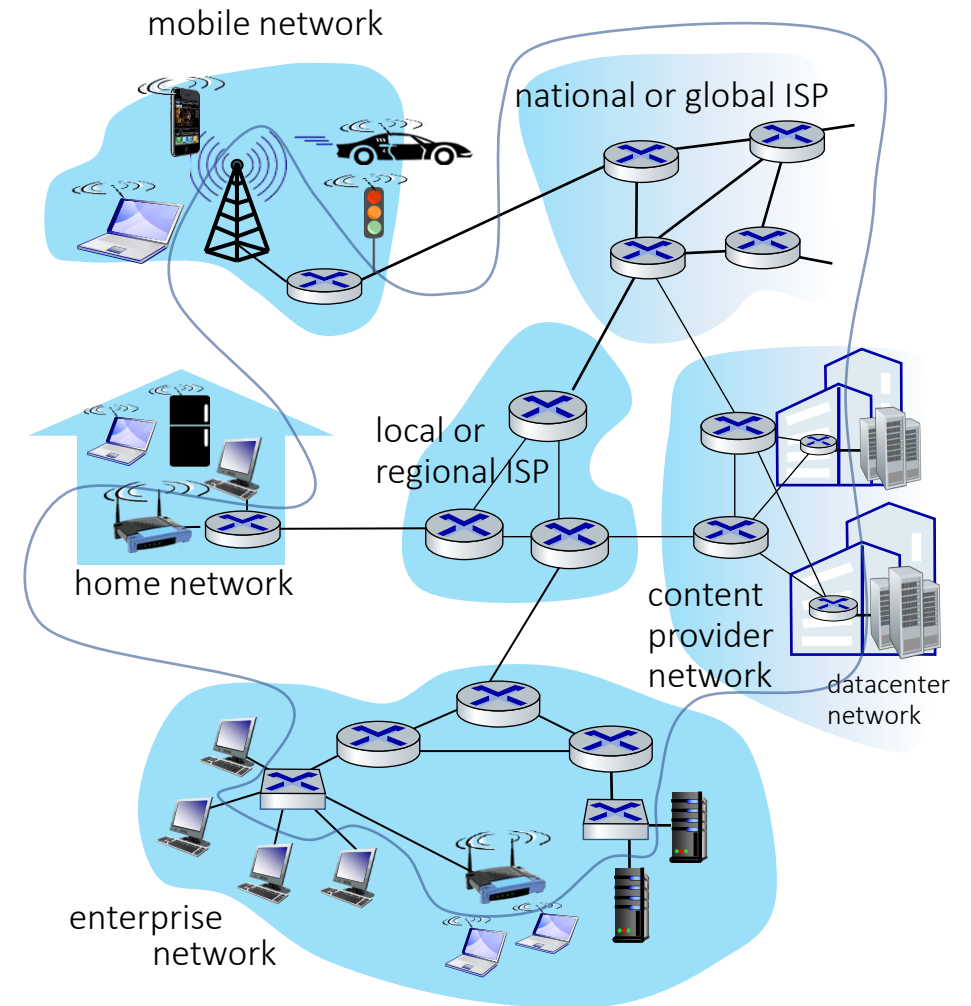
Communication links

- fiber, copper, radio, satellite
- transmission rate: *bandwidth*



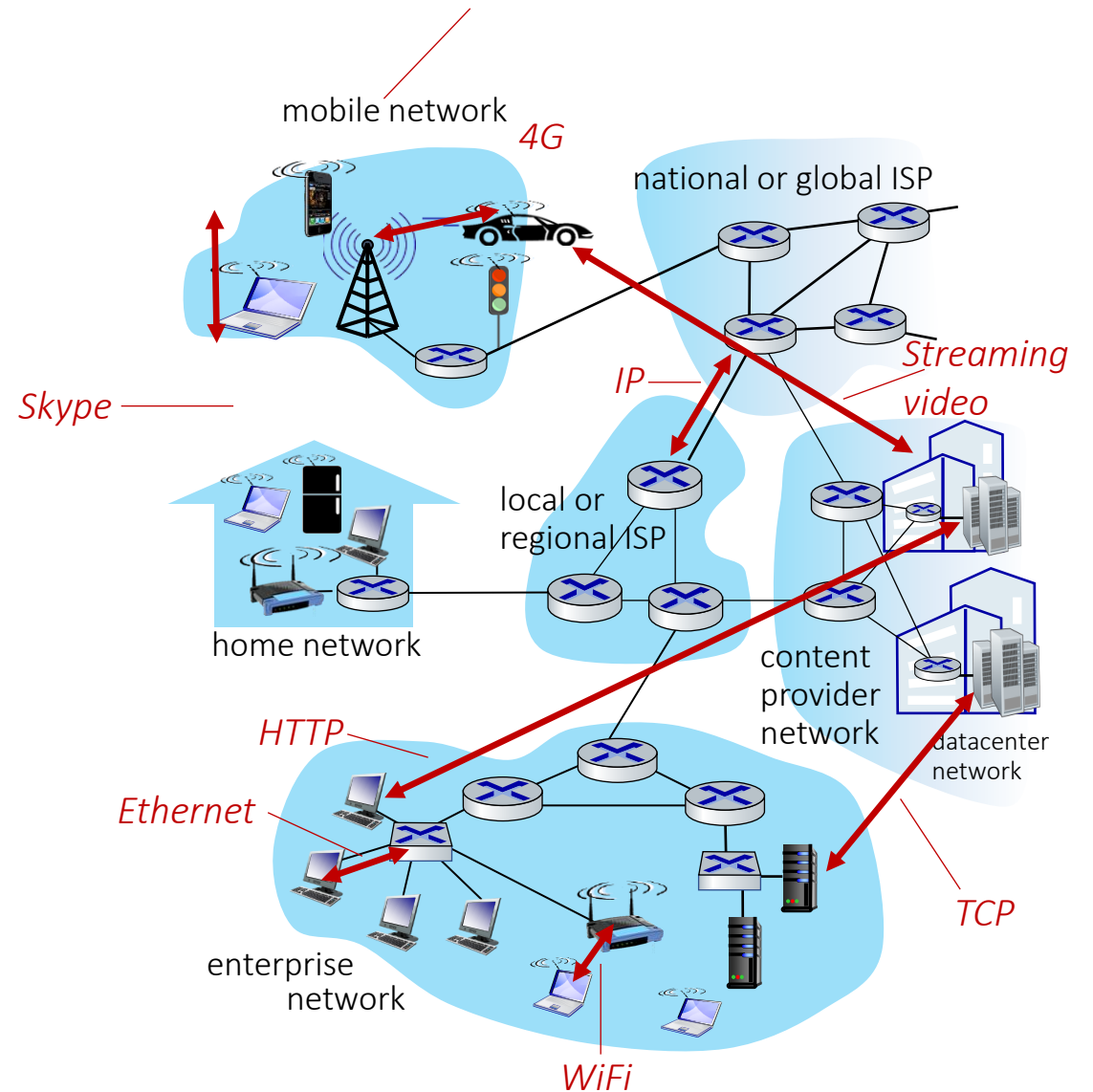
Networks

- collection of devices, routers, links: managed by an organization



The Internet: a “nuts and bolts” view

- Internet: “network of networks”
 - Interconnected ISPs
- protocols are *everywhere*
 - control sending, receiving of messages
 - e.g., HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, Ethernet
- Internet standards
 - RFC: Request for Comments
 - IETF: Internet Engineering Task Force



What's a protocol?

Human protocols:

- “what’s the time?”
- “I have a question”
- introductions

... specific messages sent

... specific actions taken
when message received,
or other events

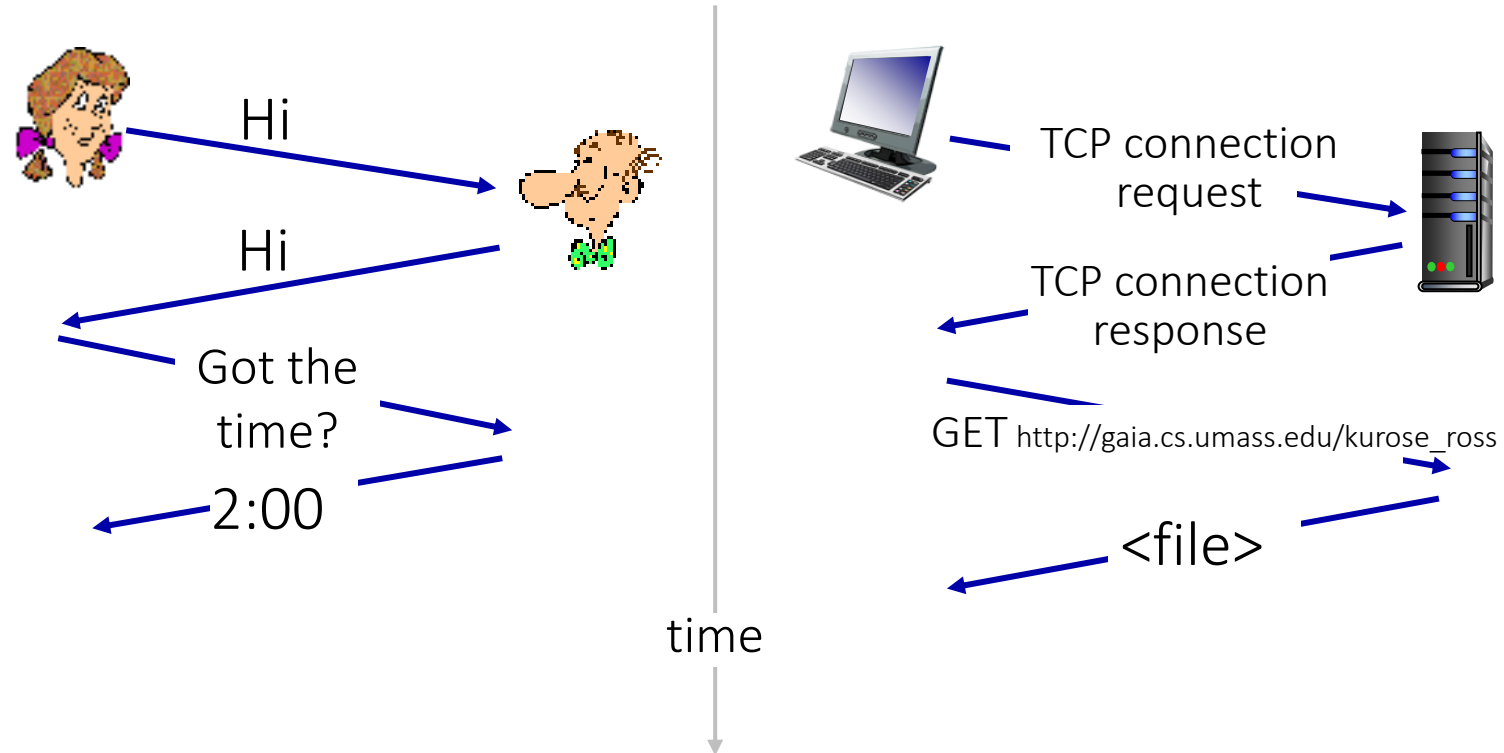
Network protocols:

- computers (devices) rather than humans
- all communication activity in Internet governed by protocols

*Protocols define the **format, order** of **messages sent and received** among network entities, and **actions taken** on msg transmission, receipt*

What's a protocol?

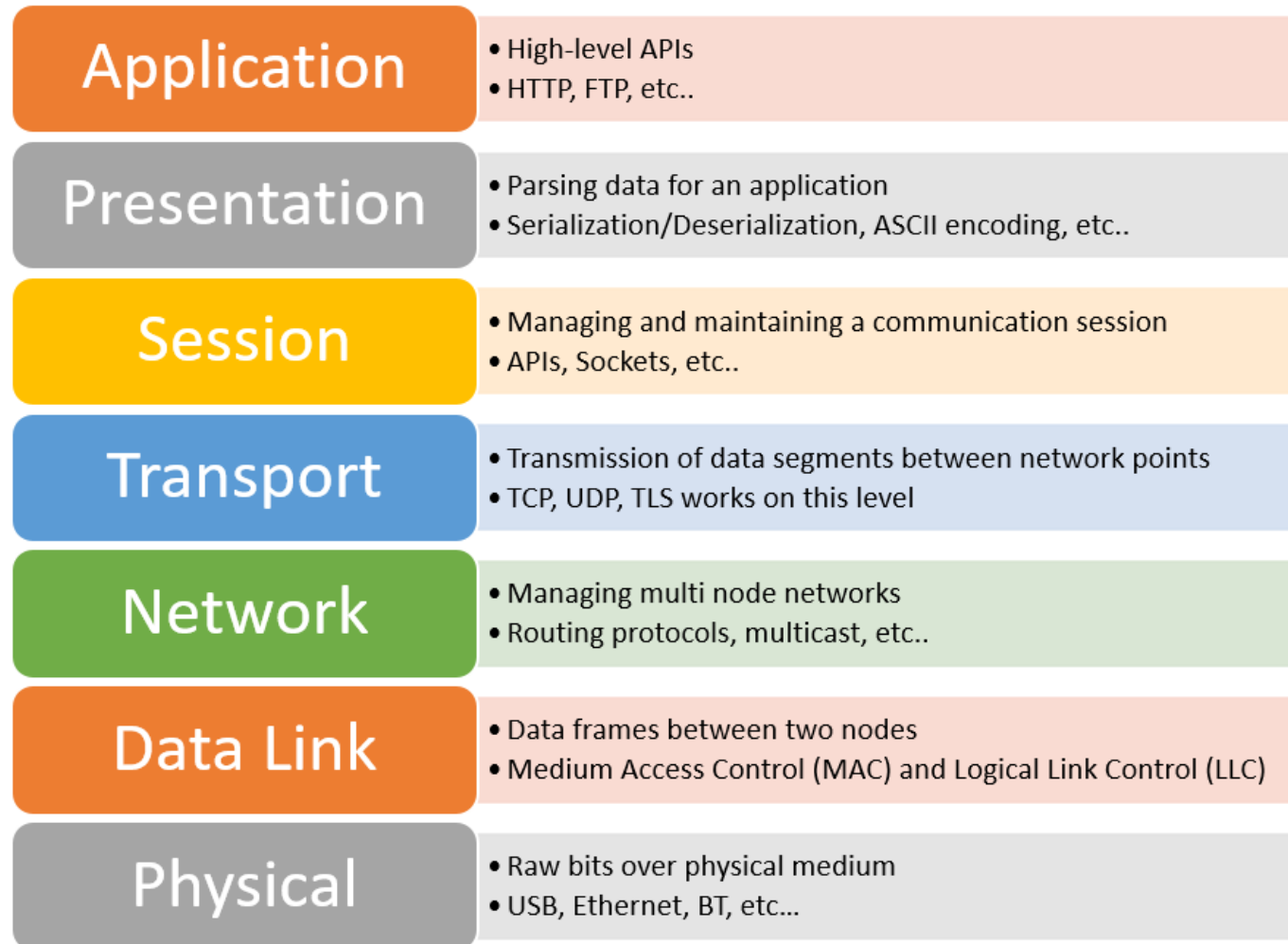
A human protocol and a computer network protocol:



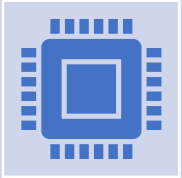
Q: other human protocols?

OSI Model

- OSI Model (Open Systems Interconnection Model) is a reference tool and a reference model for data communications in networks”. Proposed by the International Standards Organization (ISO).



TCP/IP model



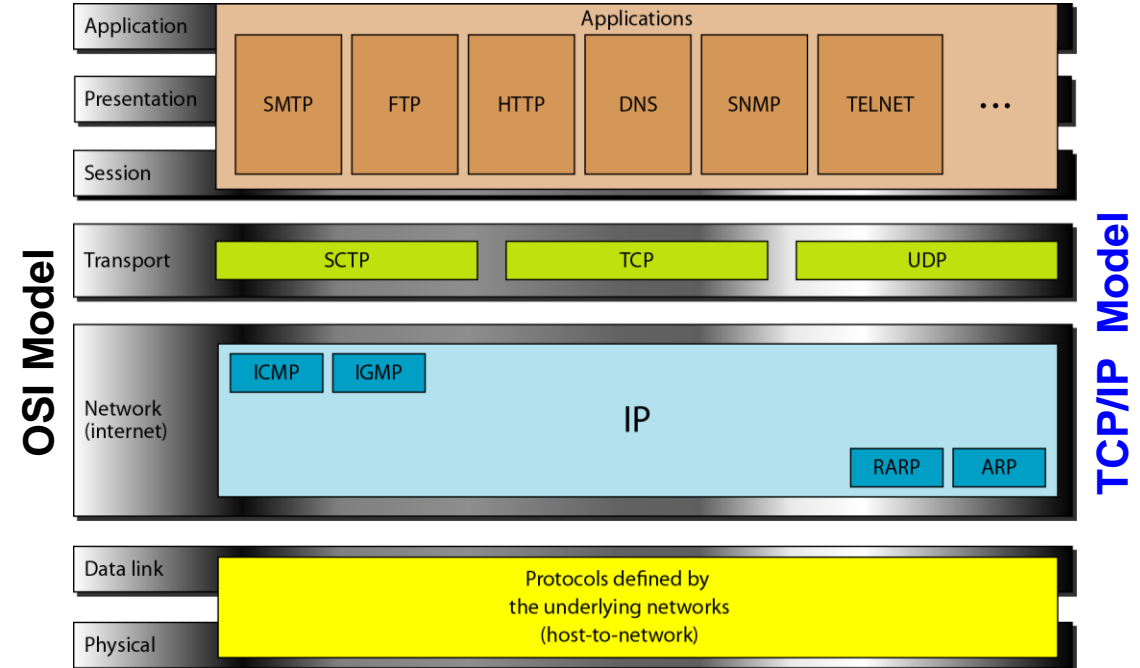
The layers in the TCP/IP protocol suite do not exactly match those in the OSI model.



The original TCP/IP protocol suite was defined as having four layers: host-to-network, internet, transport, and application.



However, when TCP/IP is compared to OSI, we can say that the TCP/IP protocol suite is made of five layers: physical, data link, network, transport, and application.



Physical Address



Also known as a MAC (Media Access Control) address, it is a unique identifier assigned to network interfaces for communication within a network.



Length: MAC addresses are 48 bits long and commonly expressed as six groups of two hexadecimal digits.

```
Select Command Prompt
Microsoft Windows [Version 10.0.18363.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

::\Users\JTP>ipconfig/all

Windows IP Configuration

Host Name . . . . . : DESKTOP-KQ3AJLF
Primary Dns Suffix . . . . . :
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No

Ethernet adapter Ethernet:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Description . . . . . : Realtek PCIe GbE Family Controller
Physical Address. . . . . : C4-65-16-E8-E5-A9
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes

Ethernet adapter VMware Network Adapter VMnet1:

Connection-specific DNS Suffix . :
Description . . . . . : VMware Virtual Ethernet Adapter for VMnet1
Physical Address. . . . . : 00-50-56-C0-00-01
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . : fe80::4c03:d594:5ea3:56d5%20(Preferred)
IPv4 Address. . . . . : 192.168.13.1(Preferred)
Subnet Mask . . . . . : 255.255.255.0
```


Port Address

01

Ports are virtual endpoints for communication in a network.

02

Range: Ports are identified by numbers from 0 to 65535.

03

Types: Well-known ports (0-1023), registered ports (1024-49151), and dynamic or private ports (49152-65535).

IP Class A

Range: 1.0.0.0 to 127.0.0.0

Used for large organizations with many networks and devices

Could support up to 127 networks with 16,777,214 devices each

Used by large companies, government agencies, and universities

IP Class B

Range: 128.0.0.0 to 191.255.0.0

Used for medium-sized organizations with a moderate number of networks and devices

Could support up to 16,384 networks with 65,534 devices each

Used by universities, hospitals, and mid-sized businesses

IP Class C



Range: 192.0.0.0 to 223.255.255.0



Used for small organizations and home networks



Could support up to 2,097,152 networks with 254 devices each



Most commonly used class for home and small business networks

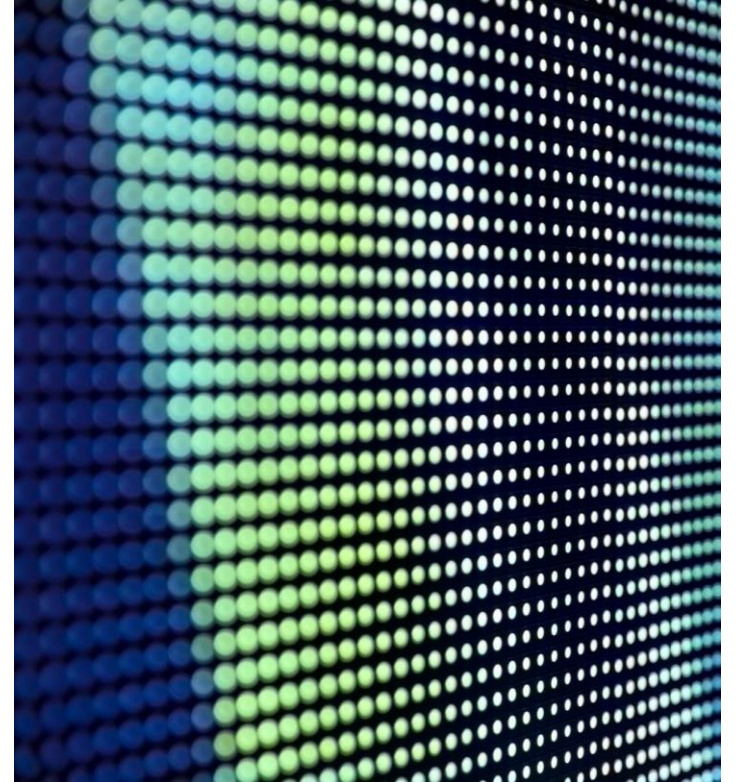
IP Class D and E



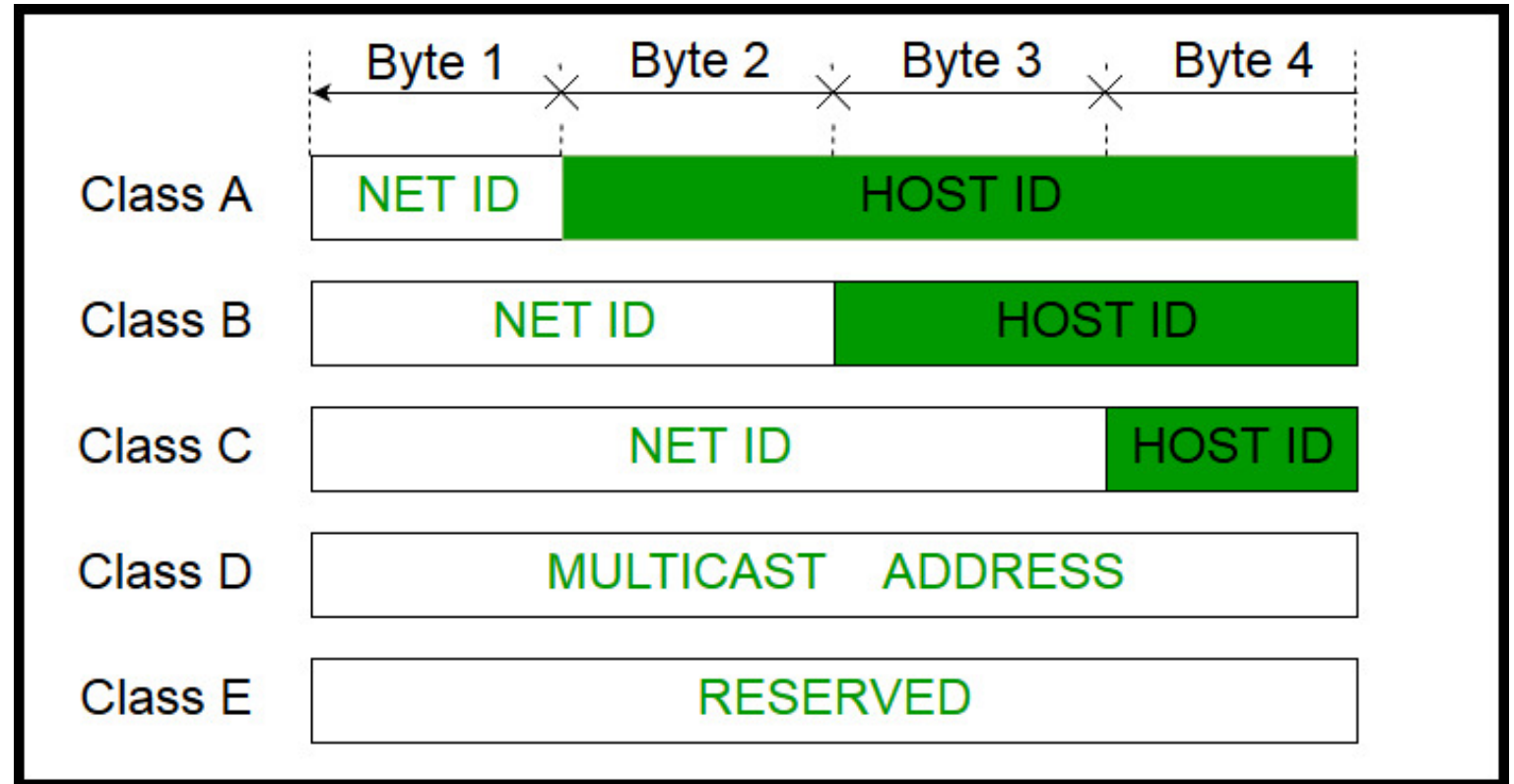
Class D (224.0.0.0 to 239.255.255.255):
Reserved for multicast groups.



Class E (240.0.0.0 to 255.255.255.255):
Reserved for experimental purposes.



IP Classes: Network and Host



Private IP Addresses



Reserved IP addresses for use within a private network.



Range: Examples include addresses from the following ranges:

10.0.0.0 to 10.255.255.255

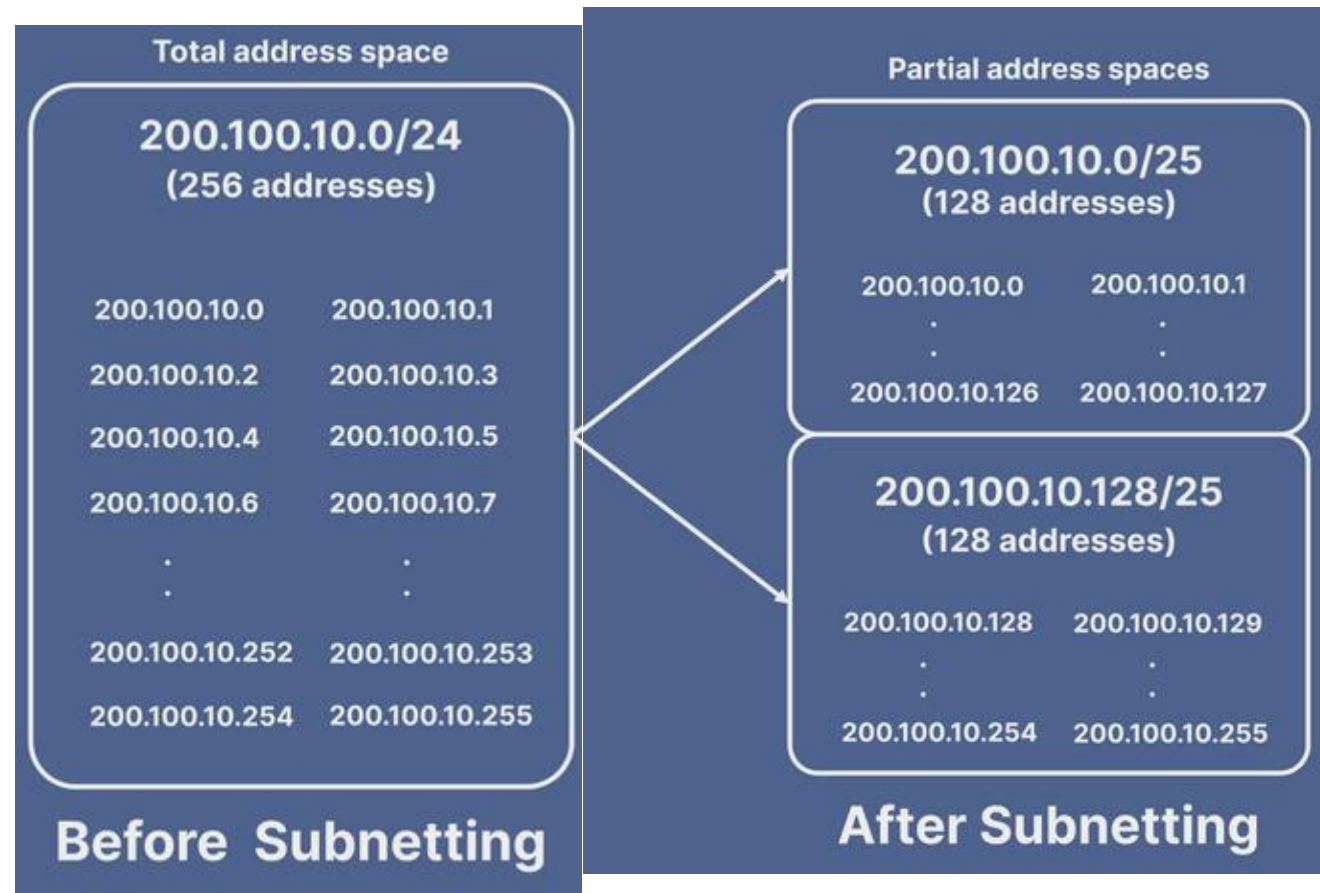
172.16.0.0 to 172.31.255.255

192.168.0.0 to 192.168.255.255



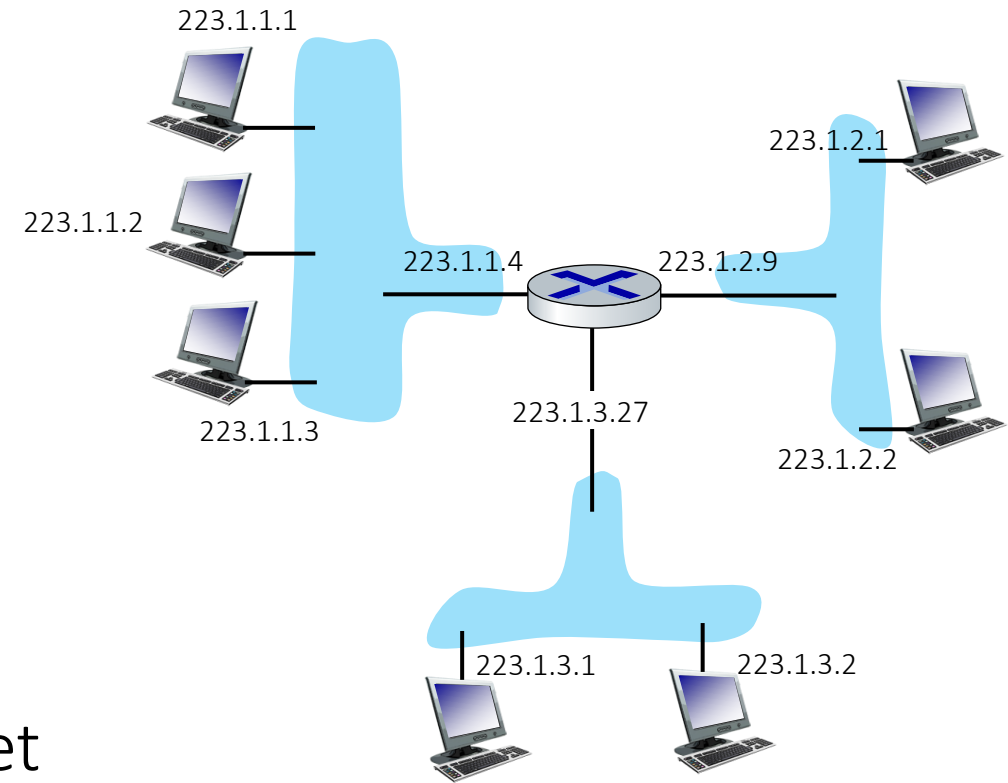
Usage: Ideal for internal network communication within an organization or a home network.

Subnetting



Subnets

- What's a subnet ?
 - Through subnetting, network traffic can travel a shorter distance **without passing through unnecessary routers to reach its destination.**
- IP addresses have structure:
 - **subnet part:** devices in same subnet have common high order bits
 - **host part:** **remaining** low order bits

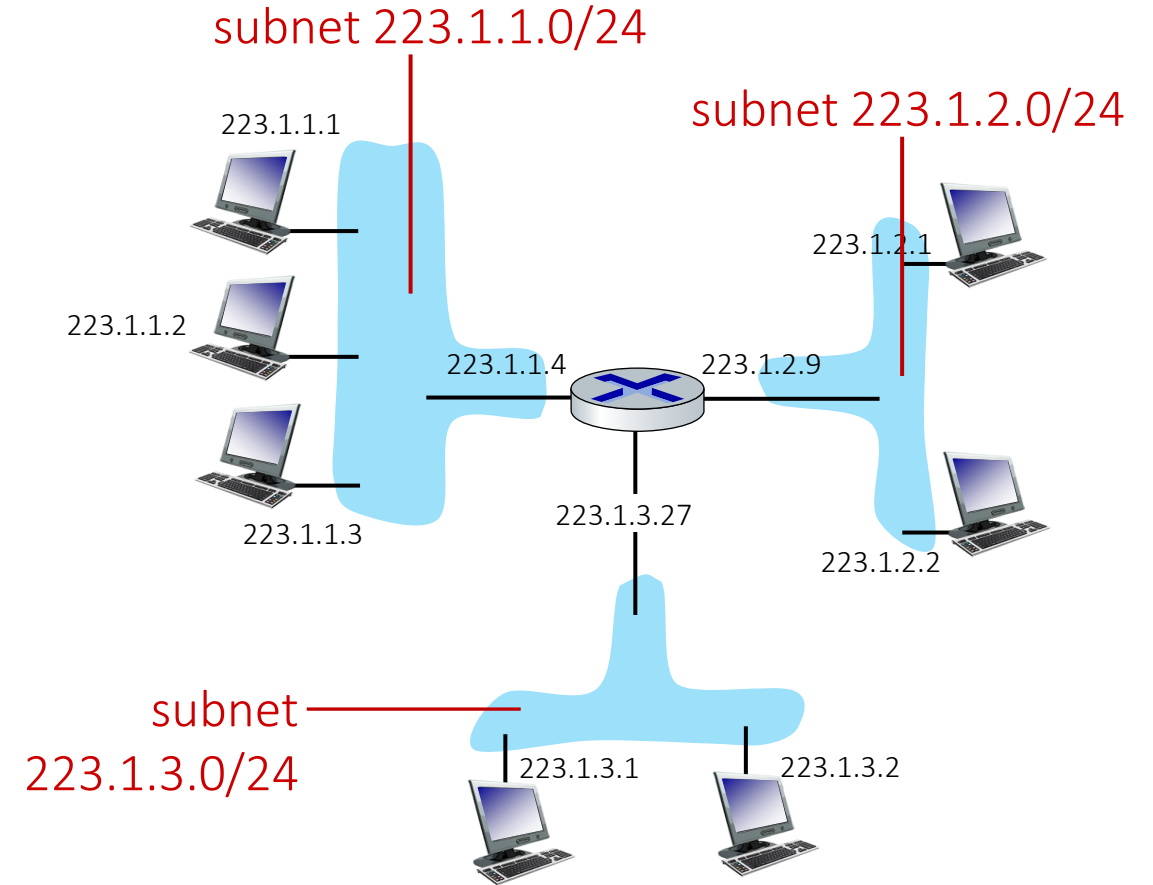


network consisting of 3 subnets

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**

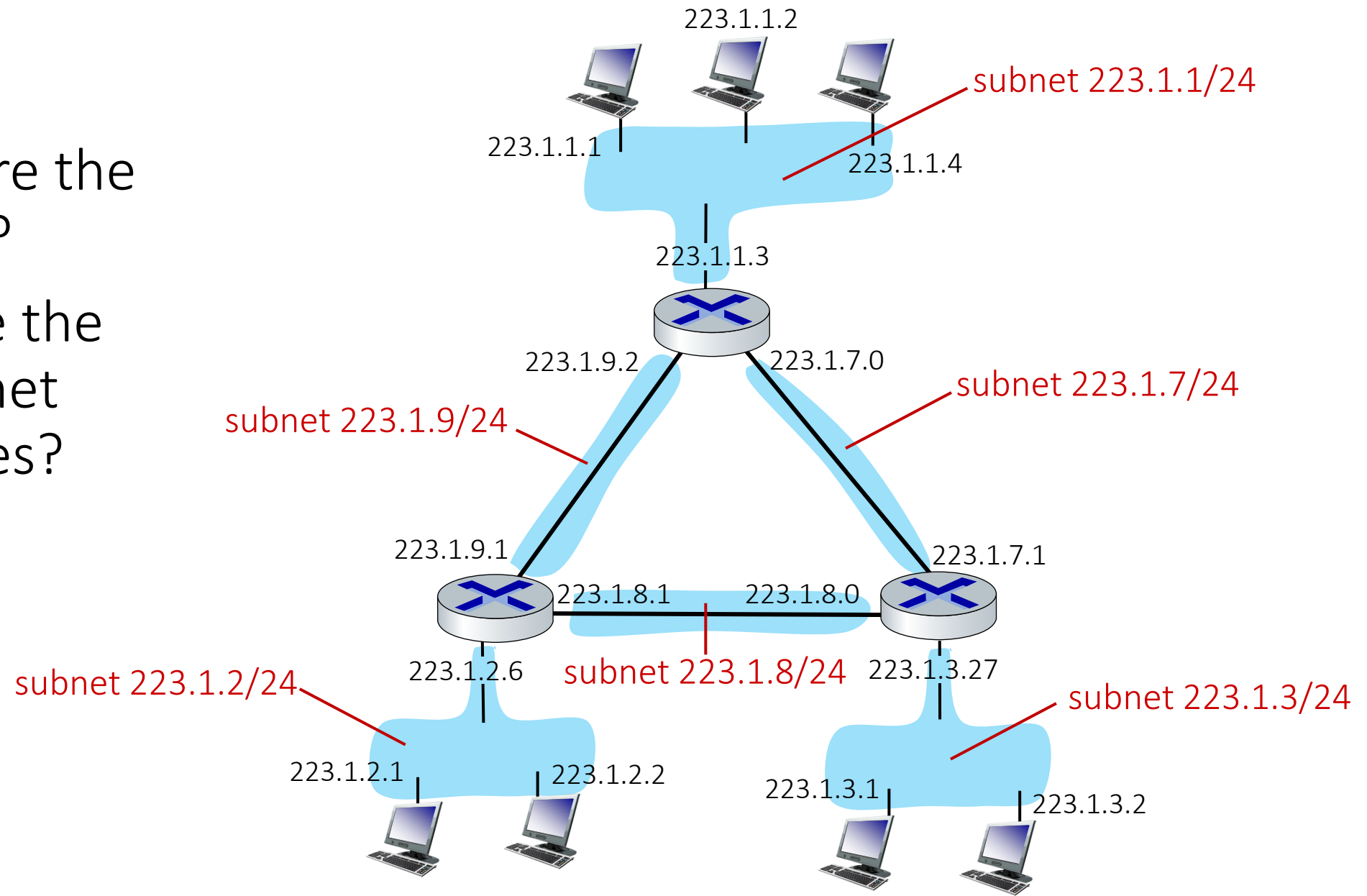


subnet mask: /24

(high-order 24 bits: subnet part of IP address)

Subnets

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



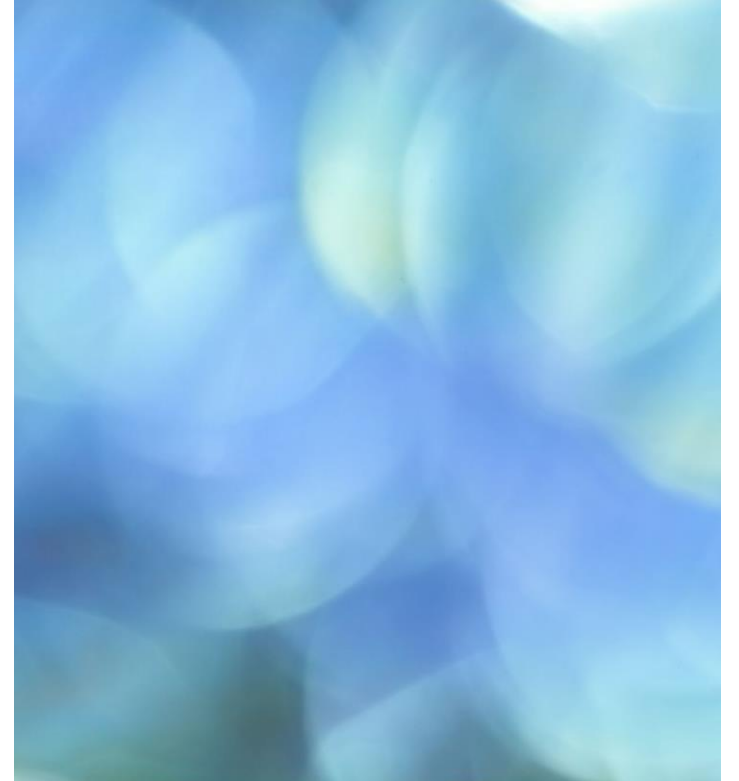
What is a Subnet Mask?

A subnet mask is a 32-bit number that divides an IP address into network and host portions.

- 11111111.11111111.11111111.00000000

This subnet mask is typically shown in the equivalent, more readable form:

- 255.255.255.0



Subnetting

IP Address: 192 . 168 . 100 . 1

IP (Binary): 11000000.10101000.01100100.00000001

Network ID

Host ID

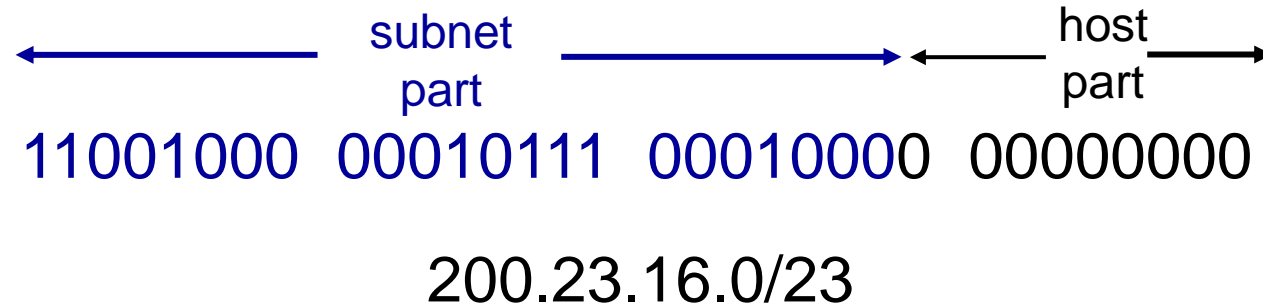
SM (Binary): 11111111.11111111.11111111.00000000


Subnet Mask: 255 . 255 . 255 . 0

IP addressing: CIDR

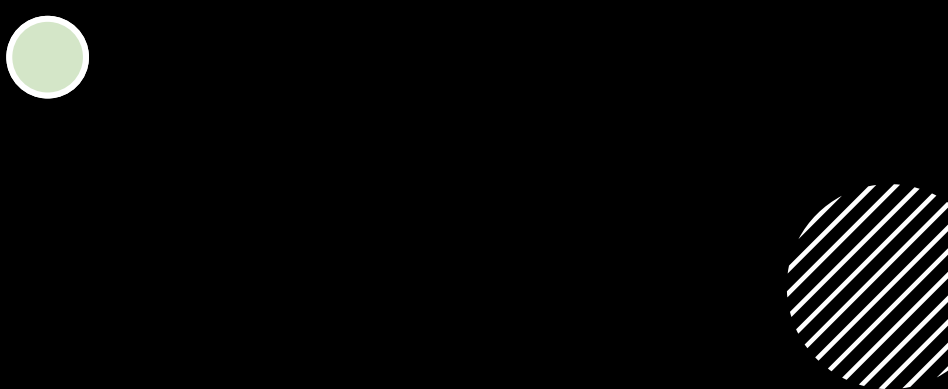
CIDR: Classless InterDomain Routing (pronounced “cider”)

- subnet portion of address of arbitrary length
- address format: **a.b.c.d/x**, where x is # bits in subnet portion of address





Subnetting Example Scenario: Innovate Inc.



Innovate Inc. initially used a single network: 192.168.1.0/24 (254 usable IPs).



Expanded to three departments: Development, Marketing, Sales.



Facing network congestion, security concerns, and IP address exhaustion.

The Solution: Subnetting

- Divide the 192.168.1.0 network into four subnets.
- Use subnet mask 255.255.255.192 (or /26 in CIDR notation).
- One subnet for each department (Development, Marketing, Sales) and one for shared resources.

Subnet	Network Address	Subnet Mask	Usable IP Range	Purpose
Subnet 1	192.168.1.0	255.255.255.192	192.168.1.1 - 192.168.1.62	Development
Subnet 2	192.168.1.64	255.255.255.192	192.168.1.65 - 192.168.1.126	Marketing
Subnet 3	192.168.1.128	255.255.255.192	192.168.1.129 - 192.168.1.190	Sales
Subnet 4	192.168.1.192	255.255.255.192	192.168.1.193 - 192.168.1.254	Shared Resources

Subnetting Calculation

Step 1 - Binary

- Convert Subnet Mask to Binary:
- 255.255.255.192 becomes 11111111.11111111.11111111.11000000

Step 2 - Network & Host Bits:

- Binary Subnet Mask: 11111111.11111111.11111111.11000000
- Network Bits: The 1s in the subnet mask (26 bits in this case: /26).
- Host Bits: The 0s in the subnet mask (6 bits).

Step 3 - Number of Subnets:

- The number of subnets is determined by the number of bits borrowed from the host portion to create the subnet id.
- In this case, we borrowed 2 bits (the two 1s that are in the host portion of the original subnet mask).
- The calculation is 2 raised to the number of borrowed bits: $2^2 = 4$ subnets.

Subnetting Calculation

Step 4 - Hosts per Subnet:

- Formula: $2^{(\text{number of host bits})} - 2$
- Calculation: $2^6 - 2 = 64 - 2 = 62$ usable hosts per subnet. (Subtract 2 for the network address and broadcast address).

Step 5 - Usable IP Range:

- Network Address: 192.168.1.0
- Broadcast Address:
 - Convert last octet of network address to binary: 00000000
 - Set host bits to 1: 00111111
 - Convert back to decimal: 63
- Broadcast Address: 192.168.1.63
- Usable Range: 192.168.1.1 to 192.168.1.62

IP addresses: how to get one?

That's actually **two** questions:

1. Q: How does a *host* get IP address within its network (host part of address)?
2. Q: How does a *network* get IP address for itself (network part of address)?

How does *host* get IP address?

- hard-coded by sysadmin in config file (e.g., /etc/rc.config in UNIX)
- **DHCP**: Dynamic Host Configuration Protocol: dynamically get address from a server

DHCP: Dynamic Host Configuration Protocol

goal: host dynamically obtains IP address from network server when it “joins” network

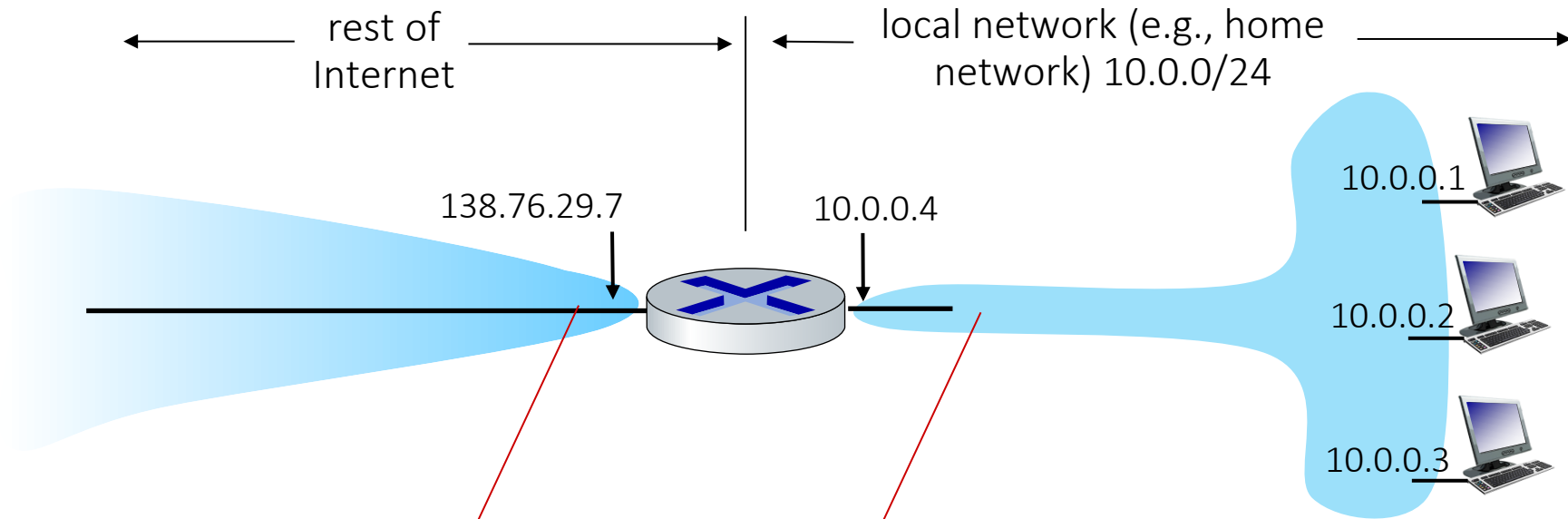
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

DHCP overview:

- host broadcasts **DHCP discover** msg [optional]
- DHCP server responds with **DHCP offer** msg [optional]
- host requests IP address: **DHCP request** msg
- DHCP server sends address: **DHCP ack** msg

NAT: network address translation

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

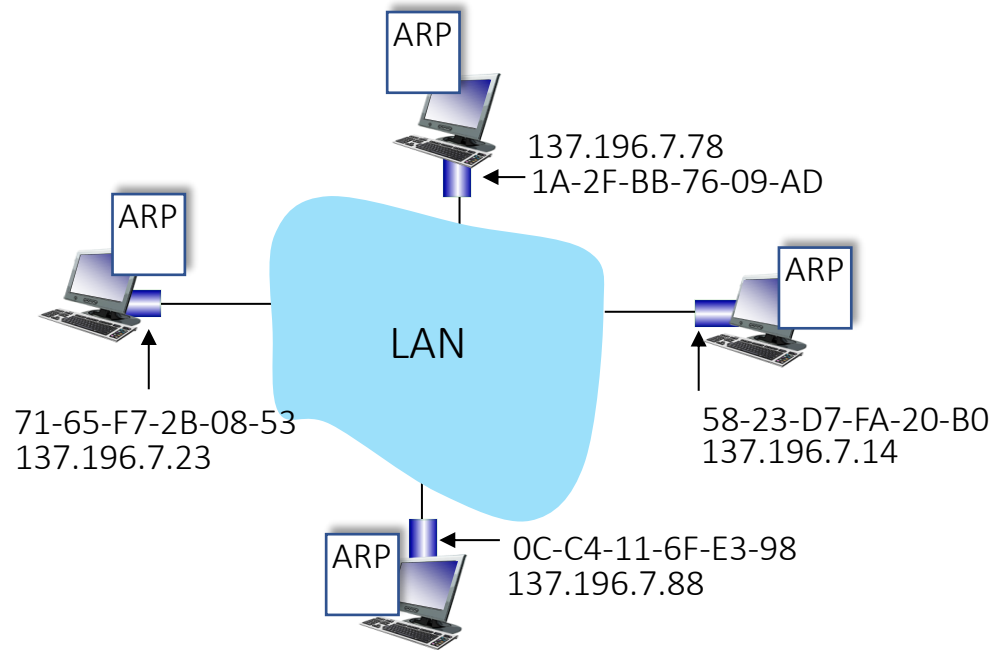


all datagrams **leaving** local network have **same** source NAT IP address: 138.76.29.7, but different source port numbers

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

ARP: address resolution protocol

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



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

- IP/MAC address mappings for some LAN nodes:
< IP address; MAC address; TTL >
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP protocol in action

example: A wants to send datagram to B

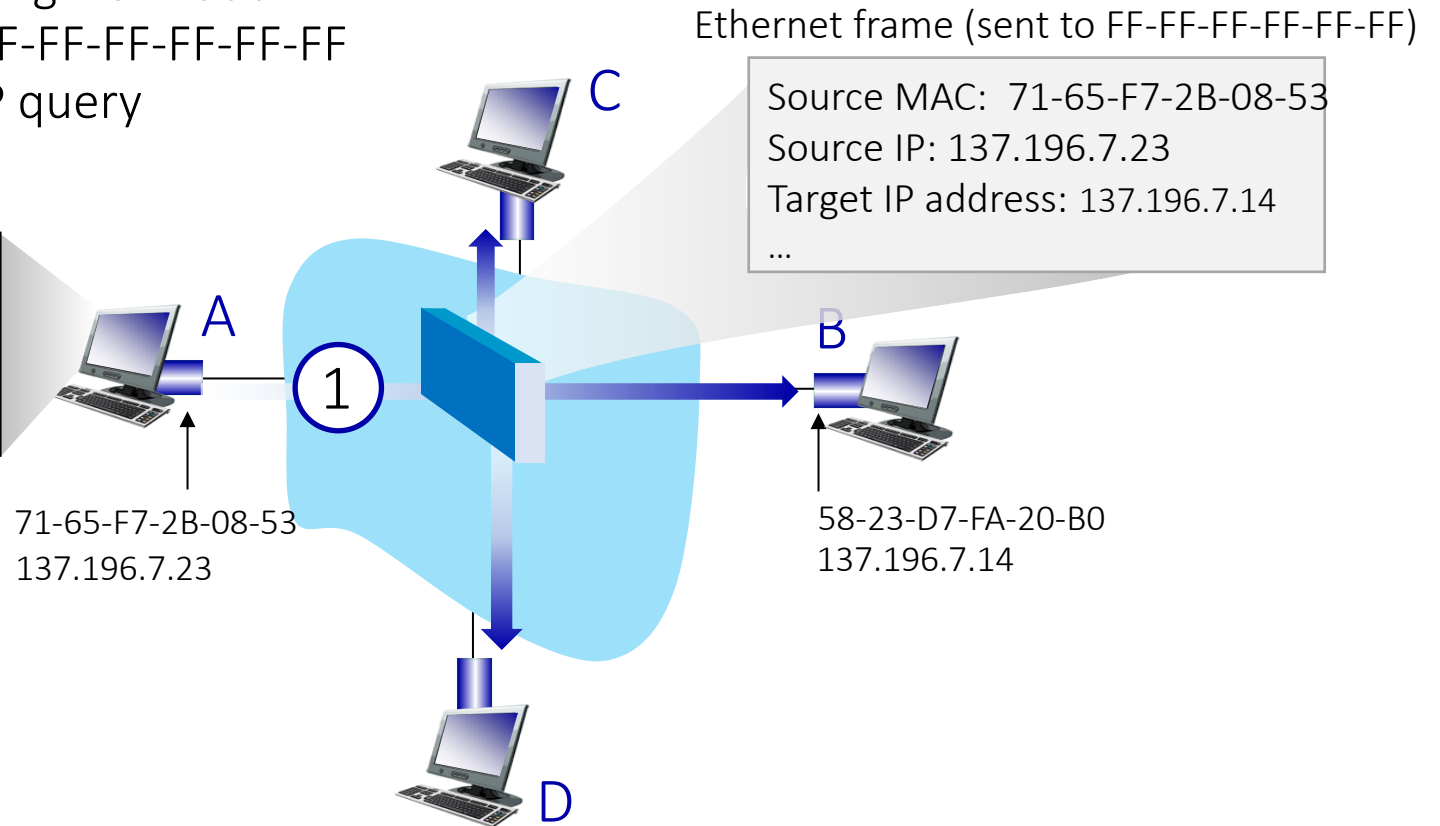
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

A broadcasts ARP query, containing B's IP addr

- ①
- destination MAC address = FF-FF-FF-FF-FF-FF
 - all nodes on LAN receive ARP query

ARP table in A

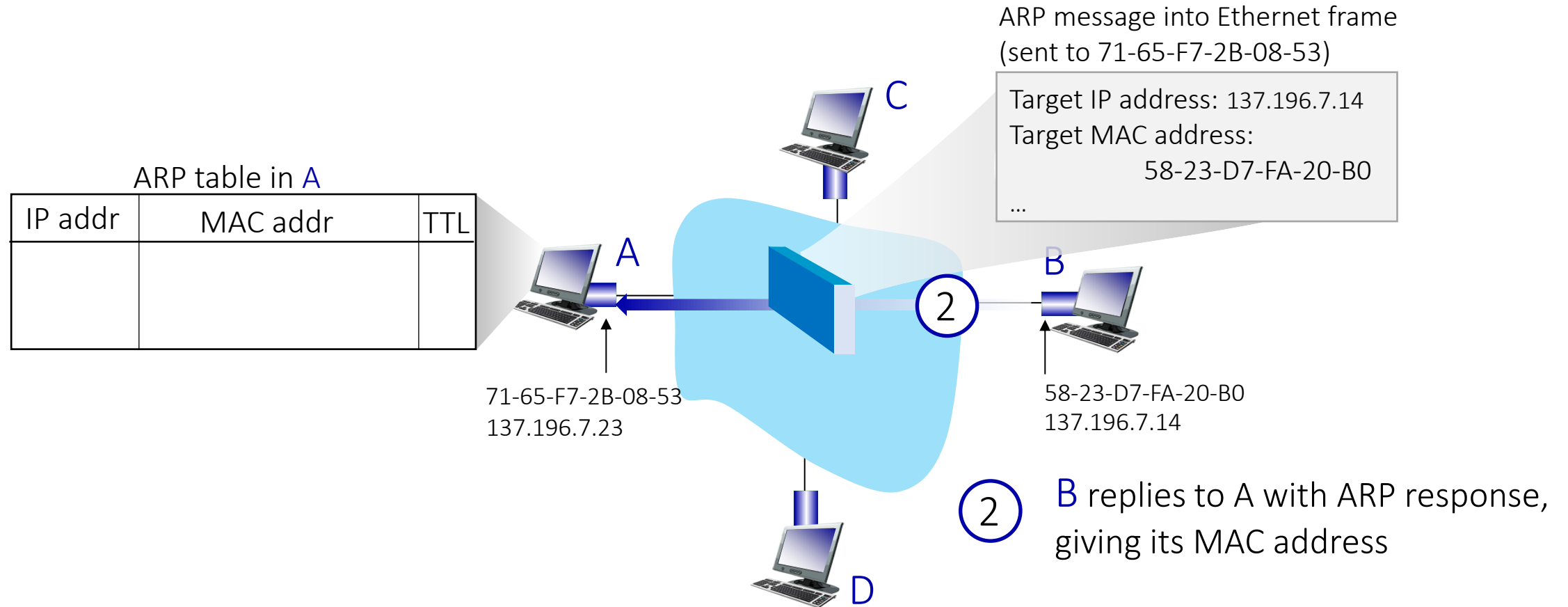
IP addr	MAC addr	TTL



ARP protocol in action

Example: A wants to send datagram to B

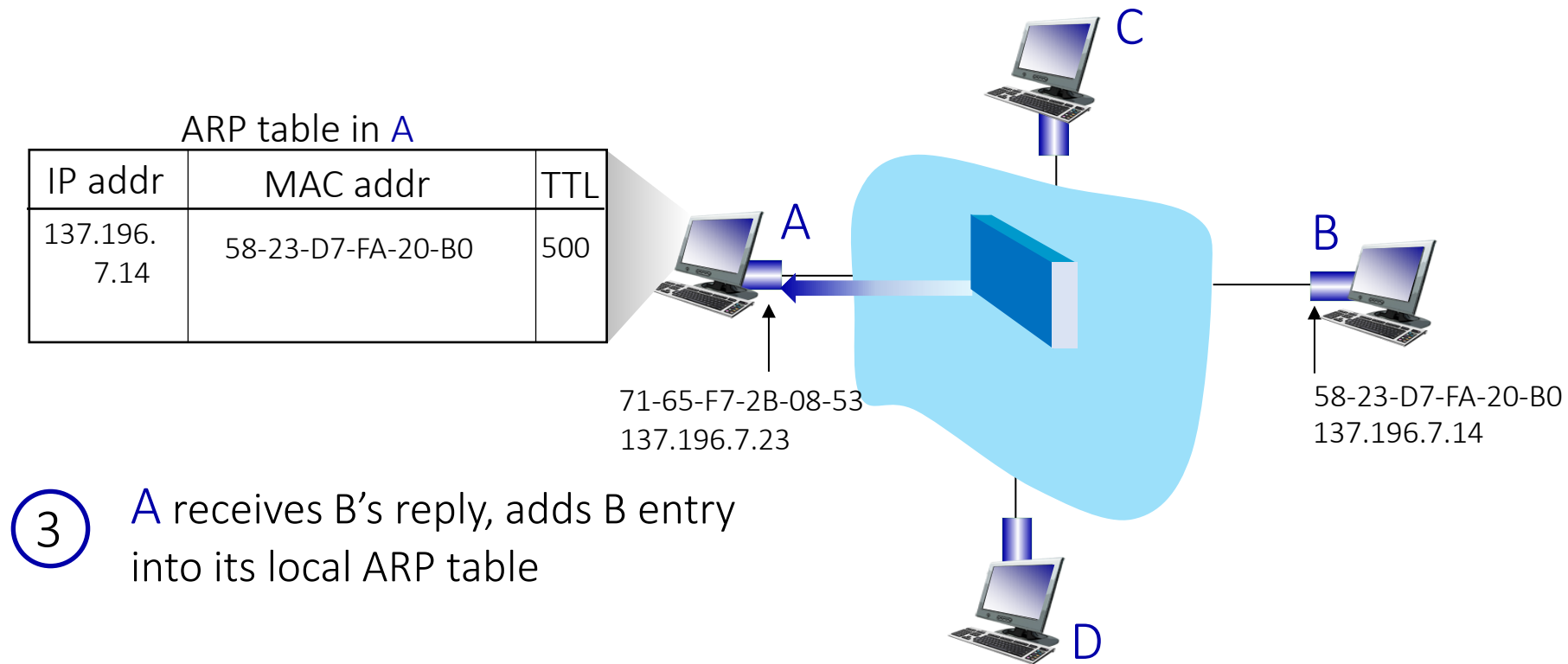
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



ARP protocol in action

example: A wants to send datagram to B

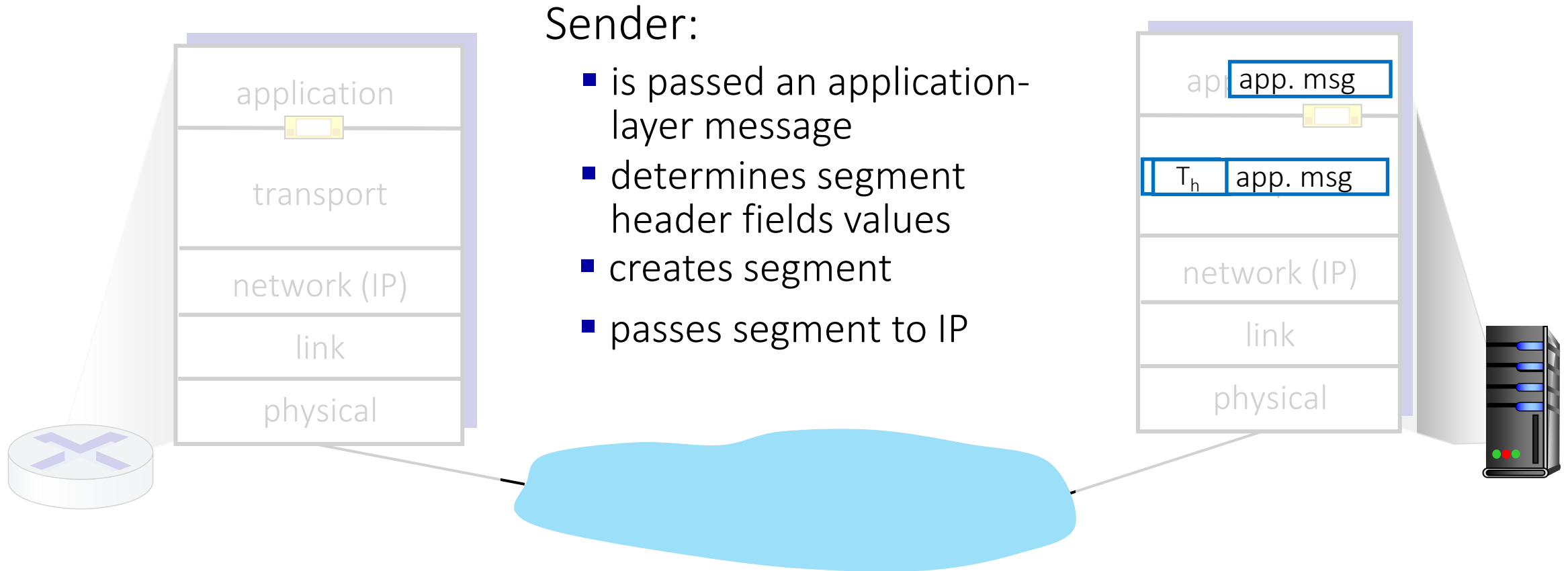
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



TCP: overview

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

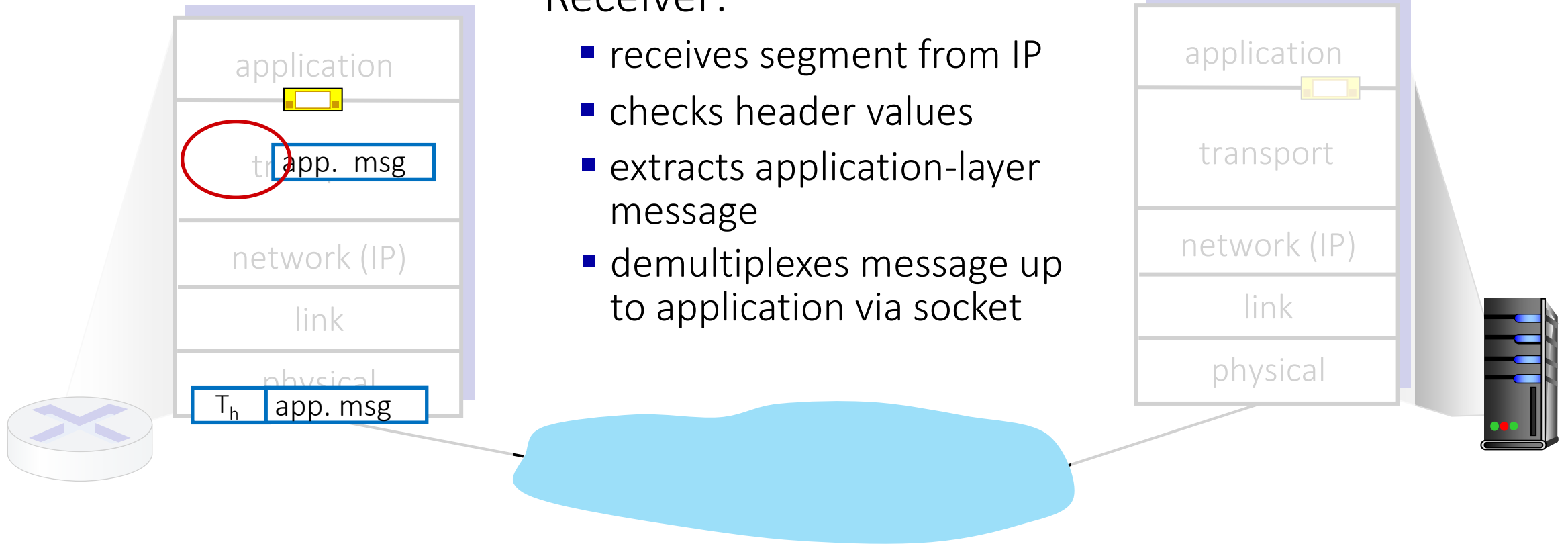
Transport Layer Actions



Transport Layer Actions

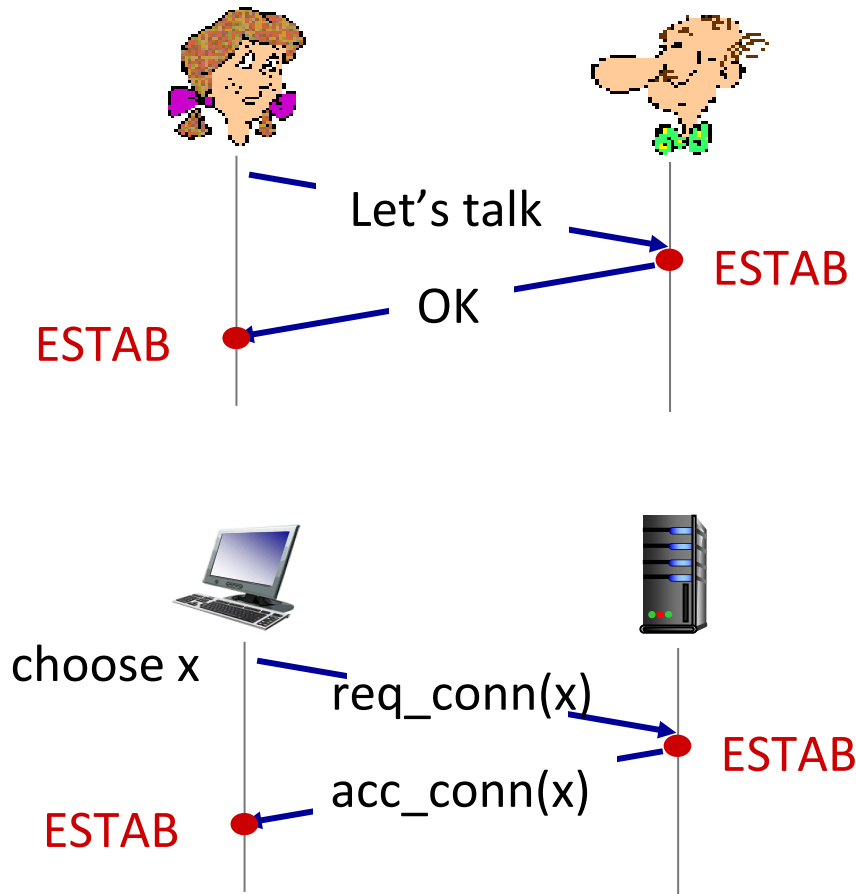
Receiver:

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



Agreeing to establish a connection

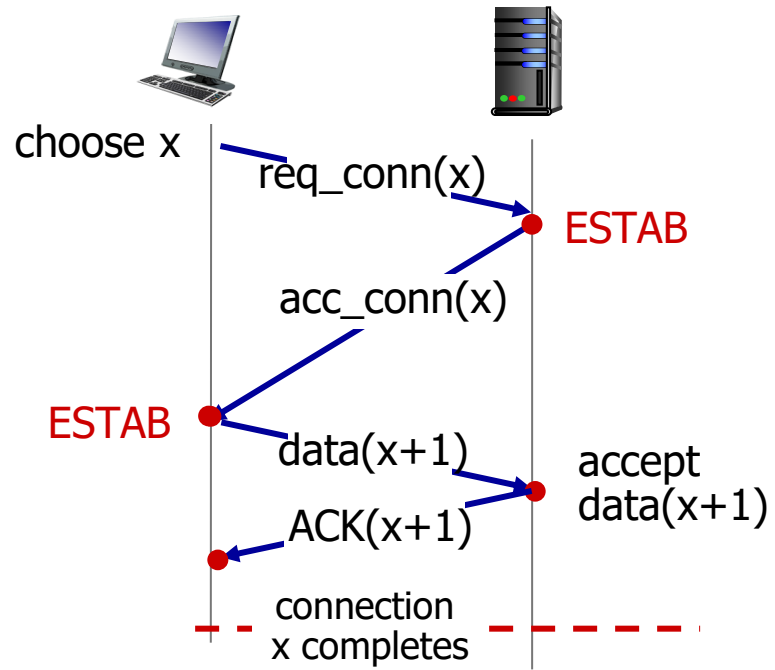
2-way handshake:



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

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

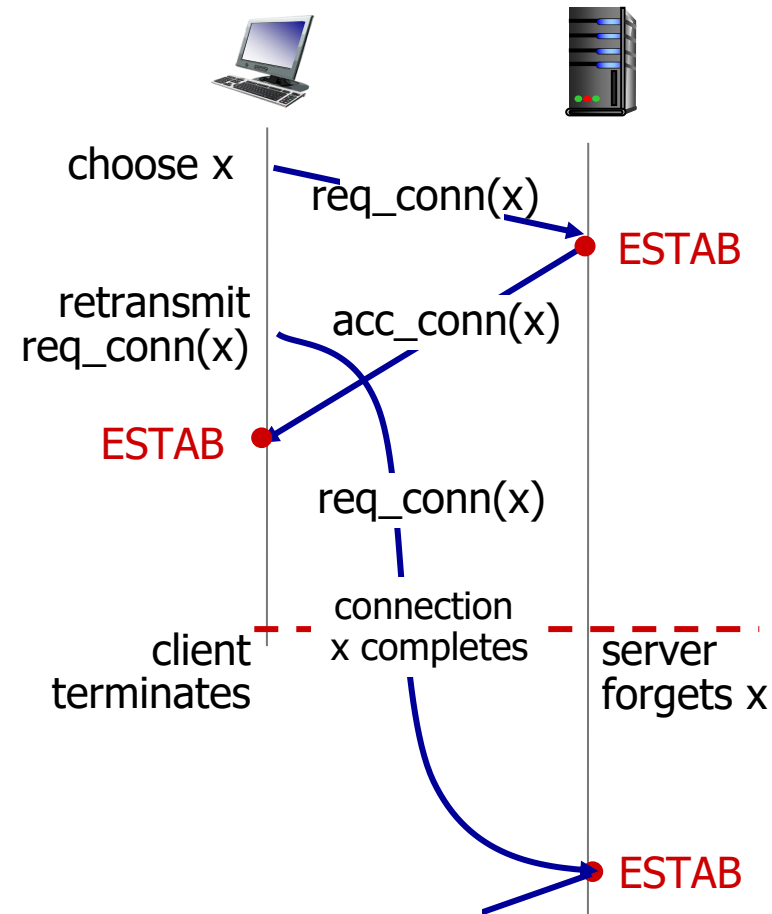
2-way handshake scenarios



No problem!

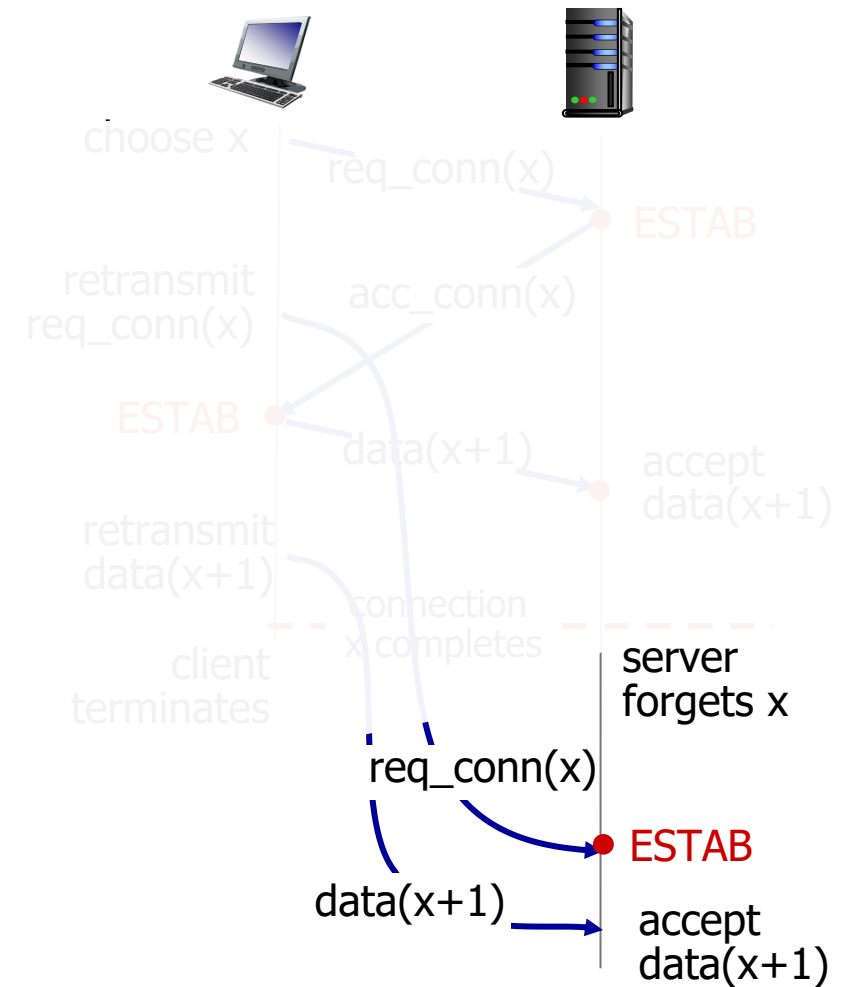


2-way handshake scenarios



Problem: half open connection! (no client)

2-way handshake scenarios



 Problem: dup data accepted!

TCP 3-way handshake

Client state

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

LISTEN

```
clientSocket.connect((serverName, serverPort))
```

SYNSENT

ESTAB

choose init seq num, x
send TCP SYN msg

SYNbit=1, Seq=x

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

Server state

```
serverSocket = socket(AF_INET, SOCK_STREAM)  
serverSocket.bind(('', serverPort))  
serverSocket.listen(1)  
connectionSocket, addr = serverSocket.accept()
```

LISTEN

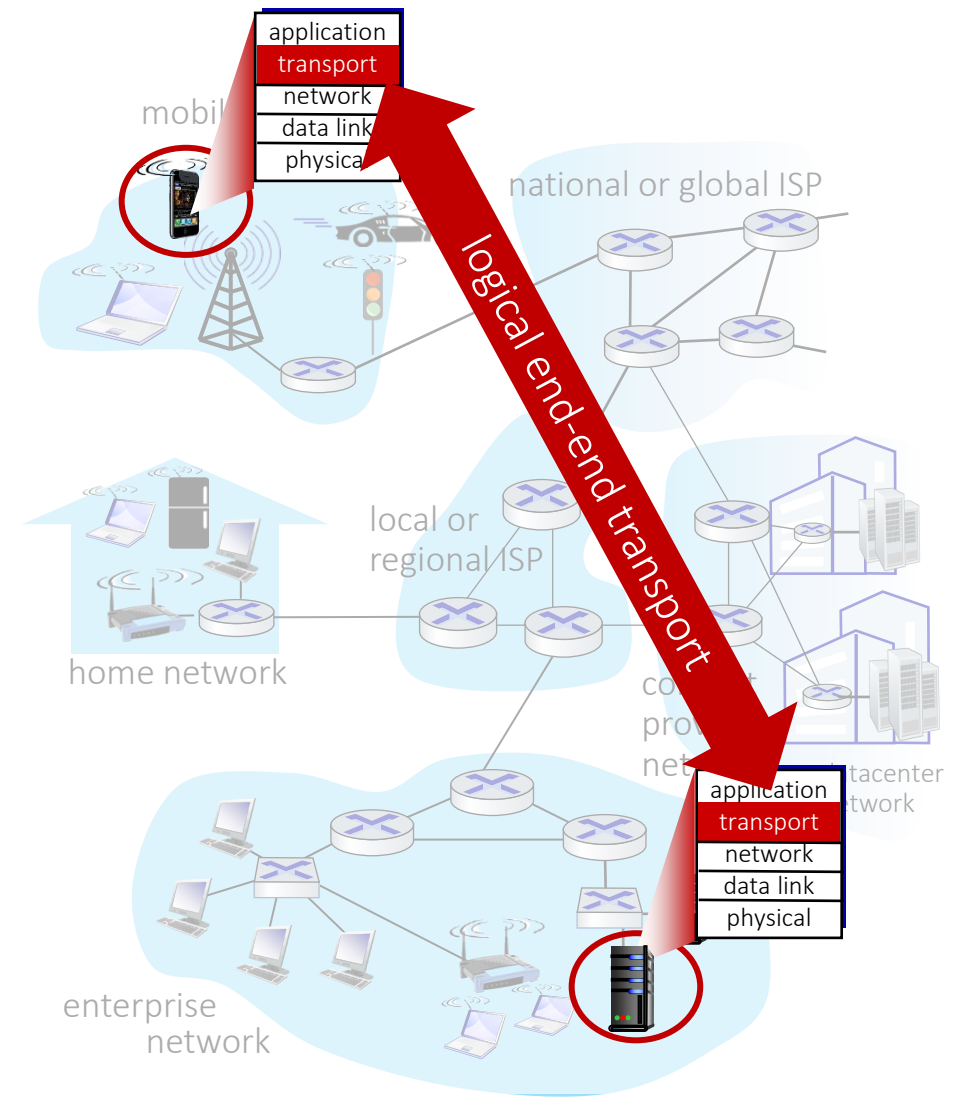
SYN RCVD

ESTAB

choose init seq num, y
send TCP SYNACK
msg, acking SYN

Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
 - reliable, in-order delivery
 - congestion control
 - flow control
 - connection setup
- **UDP:** User Datagram Protocol
 - unreliable, unordered delivery
 - no-frills extension of “best-effort” IP



UDP: User Datagram Protocol

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

Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
 - UDP can blast away as fast as desired!
 - can function in the face of congestion

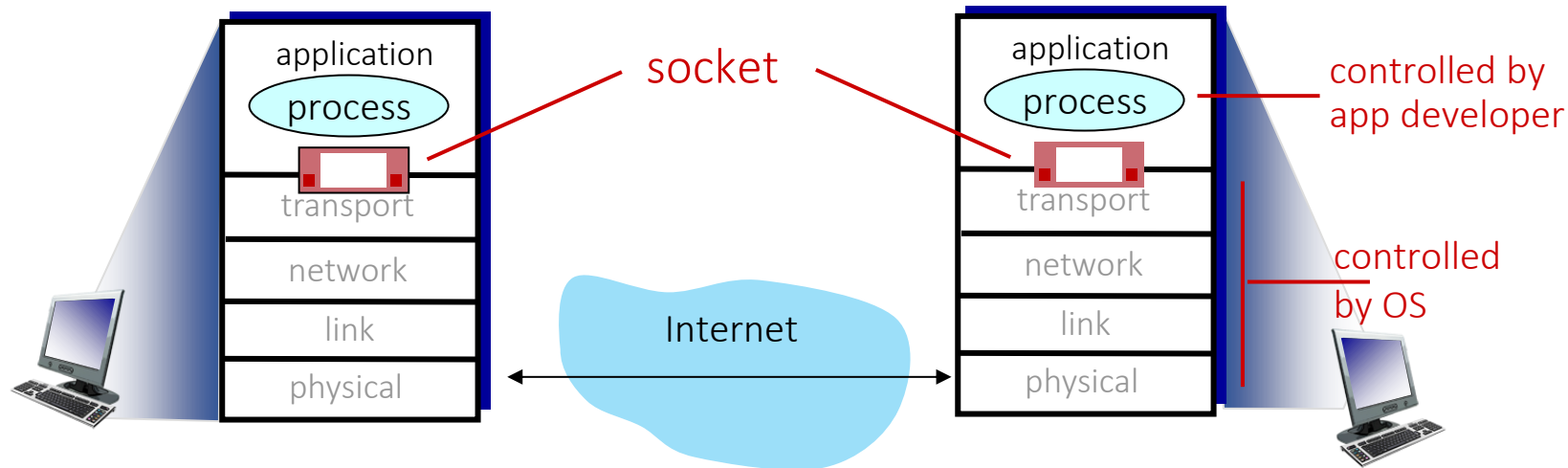
UDP: User Datagram Protocol

- UDP use:
 - streaming multimedia apps (loss tolerant, rate sensitive)
 - DNS
 - SNMP
 - HTTP/3
- if reliable transfer needed over UDP (e.g., HTTP/3):
 - add needed reliability at application layer
 - add congestion control at application layer

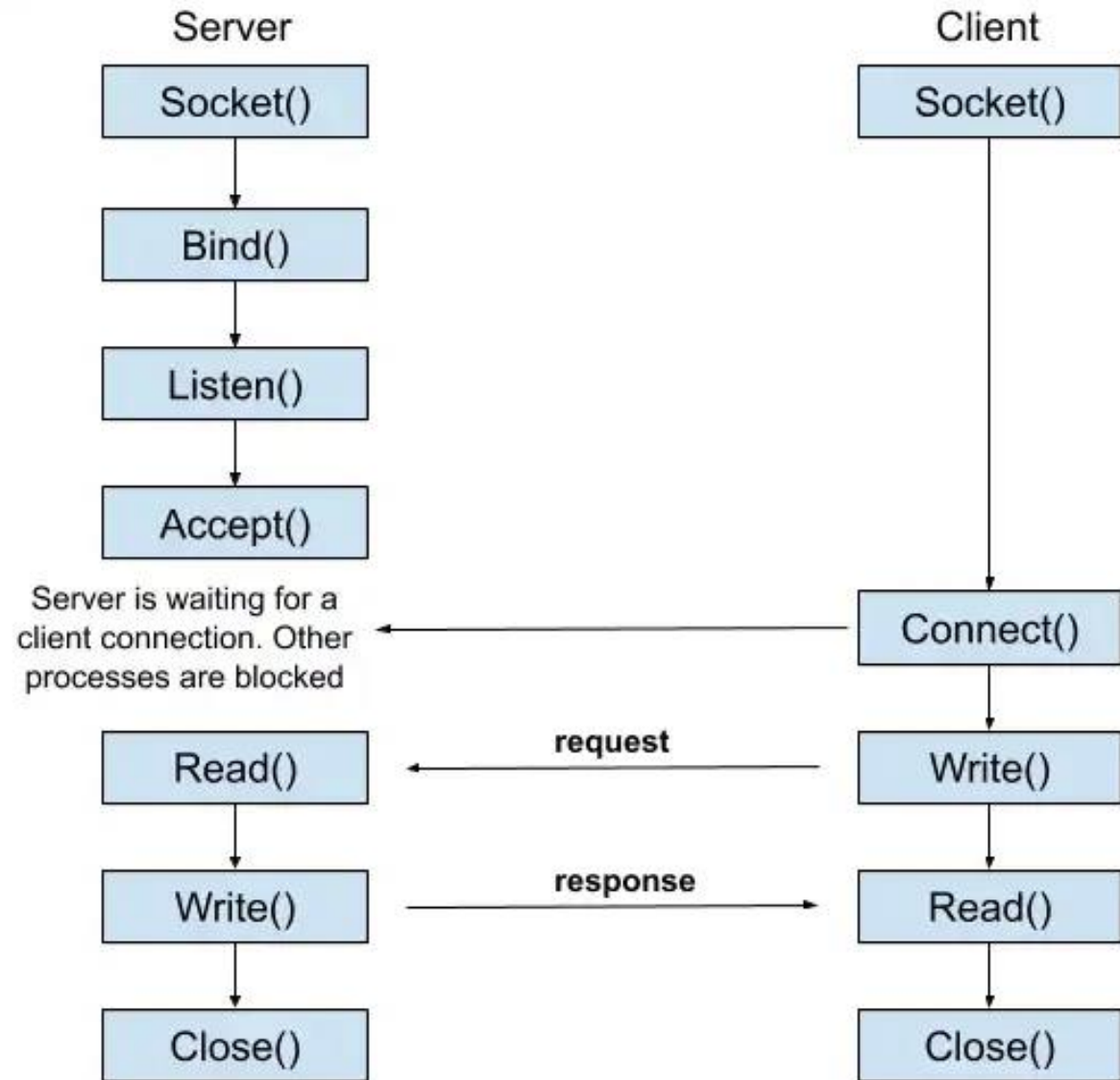
Sockets

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



How do
sockets
work?



Client/server socket interaction: TCP



server (running on `hostid`)

client



create socket,
port=`x`, for incoming
request:
`serverSocket = socket()`

wait for incoming
connection request
`connectionSocket =
serverSocket.accept()`

read request from
`connectionSocket`

write reply to
`connectionSocket`

close
`connectionSocket`

TCP
connection setup

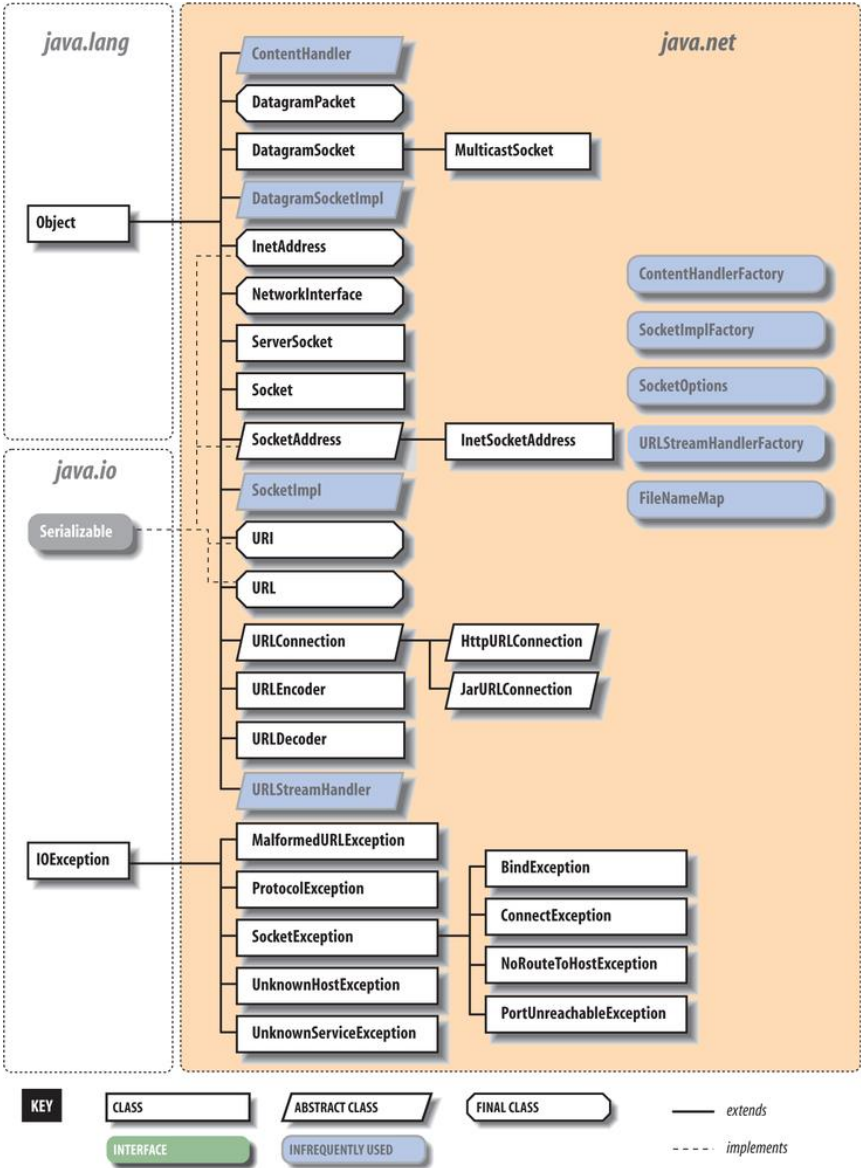
create socket,
connect to `hostid`, port=`x`
`clientSocket = socket()`

send request using
`clientSocket`

read reply from
`clientSocket`

close
`clientSocket`

Java Networking Packages



Modifier	Constructor and Description
	Socket() Creates an unconnected socket, with the system-default type of SocketImpl.
	Socket(InetAddress address, int port) Creates a stream socket and connects it to the specified port number at the specified IP address.
	Socket(InetAddress host, int port, boolean stream) Deprecated. <i>Use DatagramSocket instead for UDP transport.</i>
	Socket(InetAddress address, int port, InetAddress localAddr, int localPort) Creates a socket and connects it to the specified remote address on the specified remote port.
	Socket(Proxy proxy) Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.
protected	Socket(SocketImpl impl) Creates an unconnected Socket with a user-specified SocketImpl.
	Socket(String host, int port) Creates a stream socket and connects it to the specified port number on the named host.
	Socket(String host, int port, boolean stream) Deprecated. <i>Use DatagramSocket instead for UDP transport.</i>
	Socket(String host, int port, InetAddress localAddr, int localPort) Creates a socket and connects it to the specified remote host on the specified remote port.

Socket Programming

- Socket programming enables communication between computers over a network.
- Java provides the **java.net** package for networking.
- Two types of sockets: ServerSocket (listens for incoming connections) and Socket (initiates a connection).

```
// Importing necessary packages
import java.net.ServerSocket;
import java.net.Socket;

// Creating a ServerSocket
ServerSocket serverSocket = new ServerSocket(8080);
```

Server Side - Accepting Connections



The ServerSocket accepts incoming client connections.



Upon accepting a connection, a new Socket is created for communication with that client.

```
// Accepting a client connection  
Socket clientSocket = serverSocket.accept();
```

Server Side - Reading and Writing Data

- Input and output streams are used for reading from and writing to the socket.
- **BufferedReader** and **PrintWriter** simplify reading and writing text-based data.

```
// Creating input and output streams
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

// Reading data from the client
String clientMessage = in.readLine();

// Writing data to the client
out.println("Hello, client!");
```

Client Side - Connecting to a Server

- The client creates a Socket and connects it to the server's IP address and port.
- Communication follows a similar pattern as on the server side.

```
// Creating a socket and connecting to the server  
Socket socket = new Socket("localhost", 8080);
```

Closing Connections

- Properly closing connections is essential to release resources.
- The close() method is used for both ServerSocket and Socket.

```
// Closing connections  
serverSocket.close();  
clientSocket.close();
```

```
import java.io.*;  
import java.net.*;
```

Importing Packages

```
public class Client {  
    public static void main(String[] args) {  
        try {
```

```
            Socket socket = new Socket("localhost", 12345);
```

Create a socket to connect to the server

```
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
```

```
            // Send a message to the server  
            writer.println("Hello, server!");
```

```
            // Receive and print the response from the server  
            String serverResponse = reader.readLine();  
            System.out.println("Server says: " + serverResponse);
```

Create input and output streams for communication with the server

```
            // Close resources
```

```
            reader.close();  
            writer.close();  
            socket.close();
```

Close Resources

```
        } catch (IOException e) {  
            e.printStackTrace();
```

```
        }
```

```
    }
```

```
}
```

