

Introduction to Computer Networks and the Internet

Introduction

- Goal:
 - get “feel” and terminology
 - more depth, detail later in course
- Approach:
 - use Internet as example

Introduction

- **What's the Internet?**
- What's a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching Internet structure
- Internet structure
- protocol layers, service models
- performance: loss, delay, throughput
- security
- history

What's the Internet: “Nuts and Bolts”

View



- billions of connected computing devices:
 - **Hosts / end systems**
 - running **network apps**

Fun Internet-connected devices



IP picture frame
<http://www.ceiva.com/>



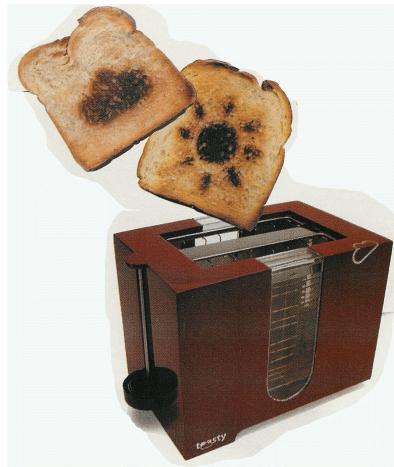
Internet refrigerator



Slingbox: watch,
control cable TV remotely



sensorized,
bed
mattress



Web-enabled toaster +
weather forecaster



Tweet-a-watt:
monitor energy use

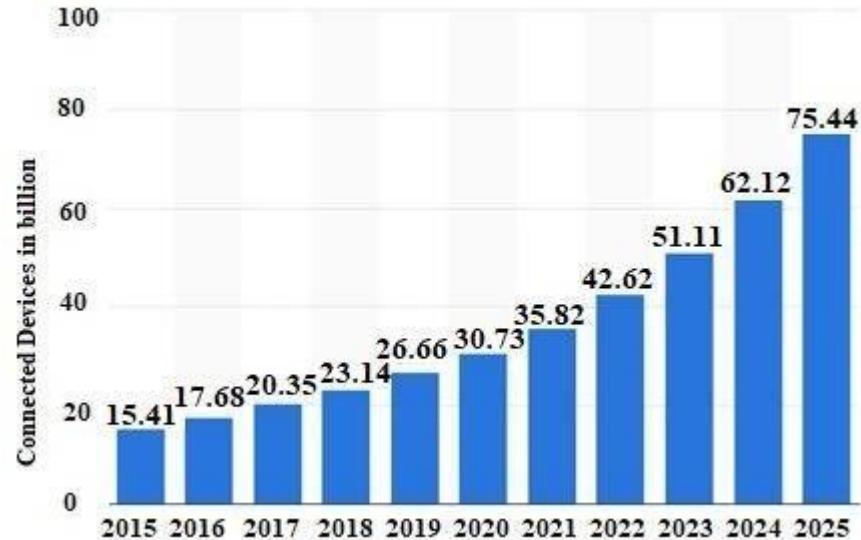


Internet phones

What's the Internet: "Nuts and Bolts"

View

Number of devices connected to Internet in billions



<https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>

What's the Internet: “Nuts and Bolts”

View



- billions of connected computing devices:
 - **Hosts / end systems**
 - running **network apps**



- **communication links**
 - fiber, copper, radio, satellite

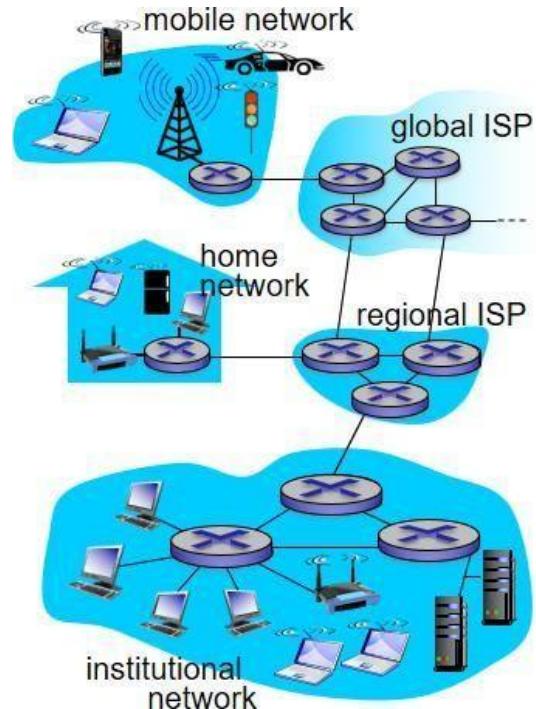


- **packet switches:** forward packets (chunks of data)
 - **routers and switches**

What's the Internet: “Nuts and Bolts”

View

- End systems access the Internet through Internet Service Providers (**ISPs**)
- internet: “network of networks”
 - Interconnected ISPs
- **protocols** control sending, receiving of messages
 - e.g., TCP, IP, HTTP, Skype, 802.11
- **Internet standards**
 - IETF: Internet Engineering Task Force
 - RFC (Request for comments): The IETF standards documents
 - The IEEE 802 standard Committee
 - Ethernet and wireless WiFi standards

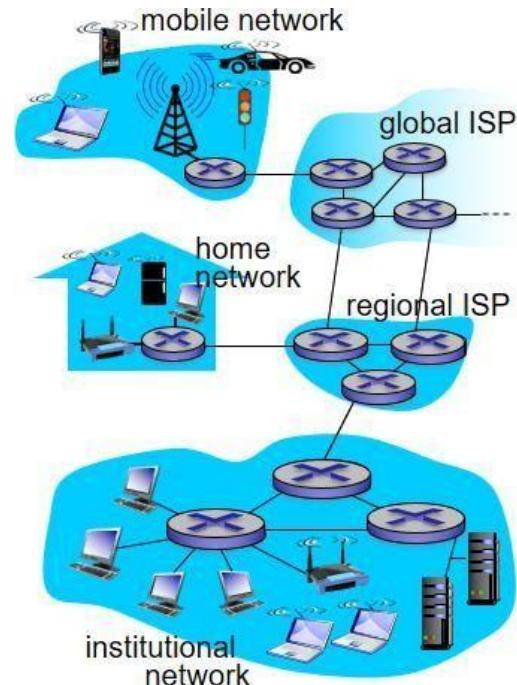


What's the Internet: A Service

View

infrastructure that provides services to the applications:

- Web, VoIP, email, games, e-commerce, social nets, ...
 - Distributed applications
 - Runs on end systems
- provides **programming interface** (socket interfaces) to apps
 - A set of rules that the sending and receiving programs must follow so that the Internet can deliver the data
 - provides service options, analogous to postal service

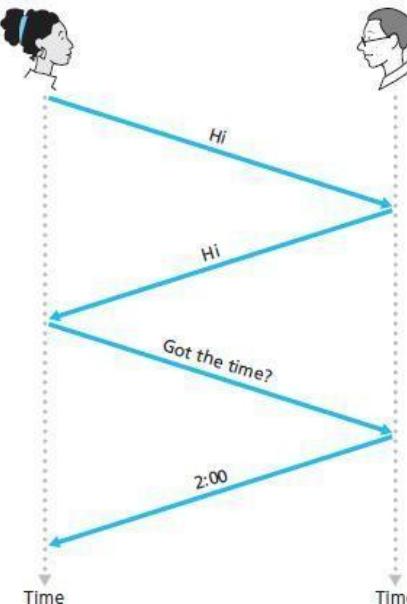


Introduction

- What's the Internet?
- **What's a protocol?**
- network edge; hosts, access net, physical media
- network core: packet/circuit switching Internet structure
- Internet structure
- protocol layers, service models
- performance: loss, delay, throughput
- security
- history

What's a protocol?

A human protocol

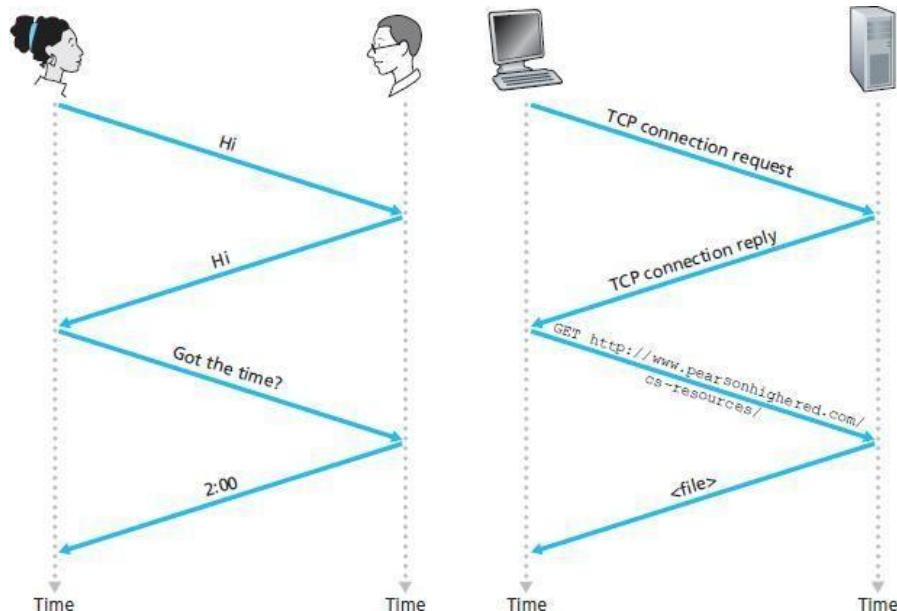


What's a protocol?

- human protocols:
 - ... specific messages sent
 - ... specific actions taken when messages received, or other events (no reply)
- network protocols:
 - machines rather than humans
 - all communication activity in Internet governed by protocols
- protocols define **format, order of messages sent and received** among network entities, and **actions taken** on message transmission, receipt

What's a protocol?

A human protocol and a computer network protocol:

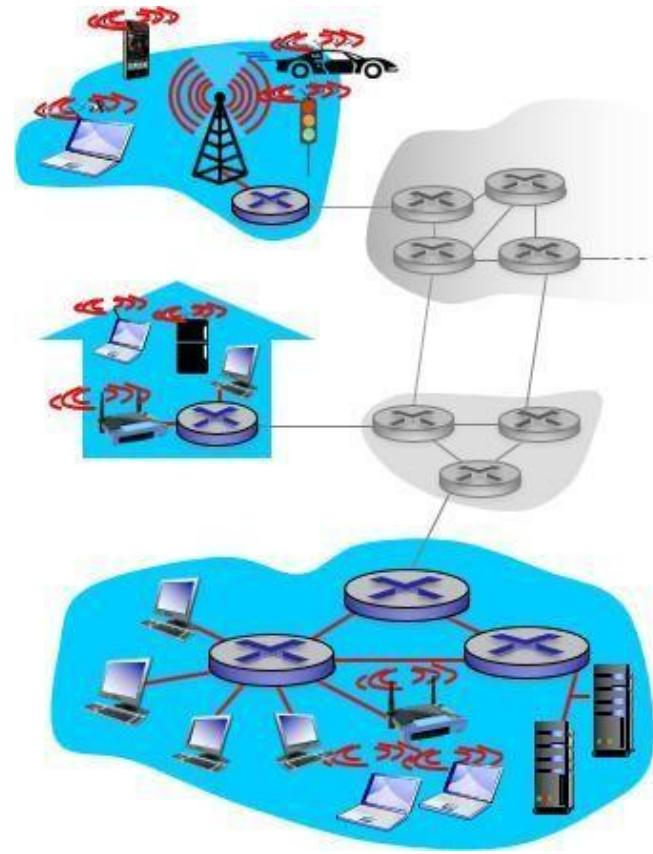


Introduction

- What's the Internet?
- What's a protocol?
- **network edge; hosts, access net, physical media**
- network core: packet/circuit switching Internet structure
- Internet structure
- protocol layers, service models
- performance: loss, delay, throughput
- security
- history

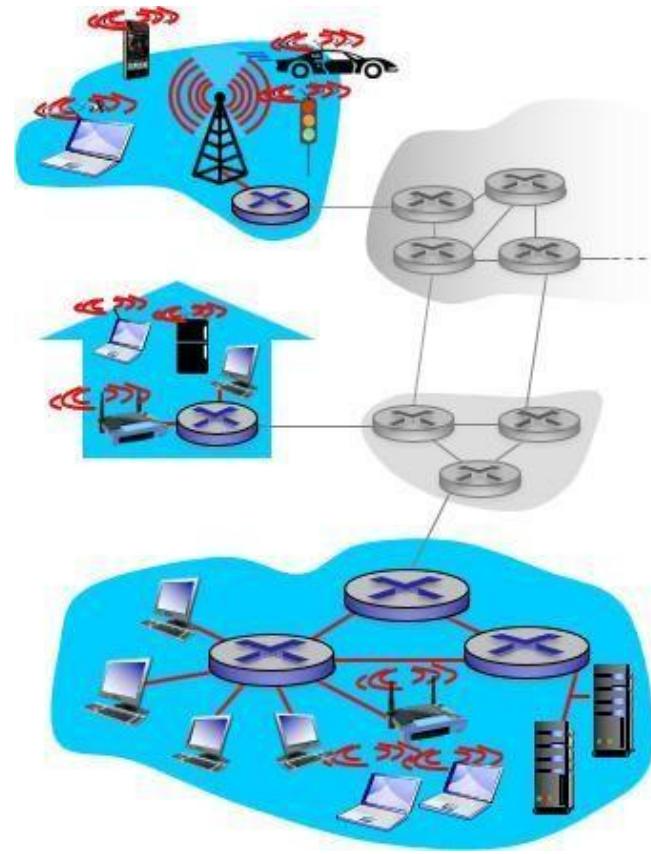
A Closer Look at Network Structure: Network Edge

- Network edge
- Network core



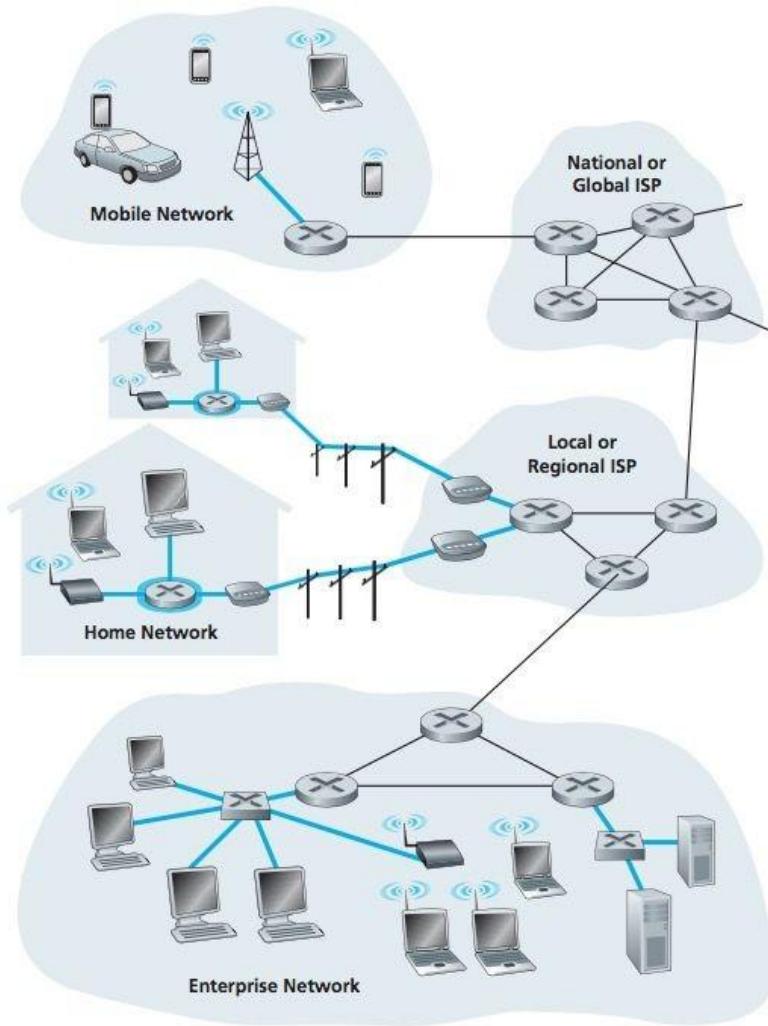
Network Edge

- End systems/Host
 - Smartphones, computers, desktop computers, servers, mobile devices, things..
 - They sit at the edge of the Internet
 - They run application programs such as a Web browser, web server, email client or server
 - clients/servers: servers often in data centres



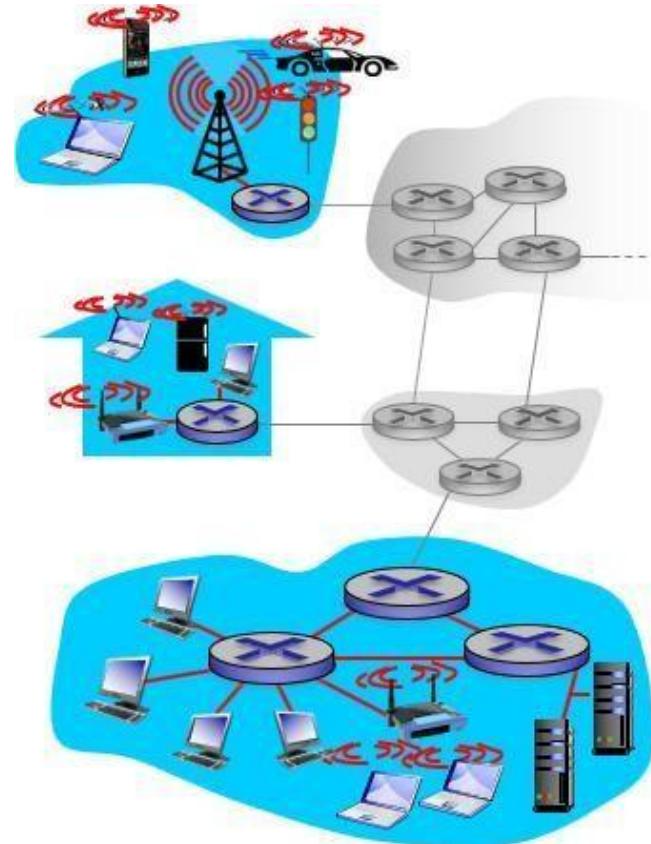
Network Edge

- Access networks:
 - the network that physically connects an end system to the first router on a path from the end system to any distant end system
- Network access technologies
 - physical media: wired, wireless communication links

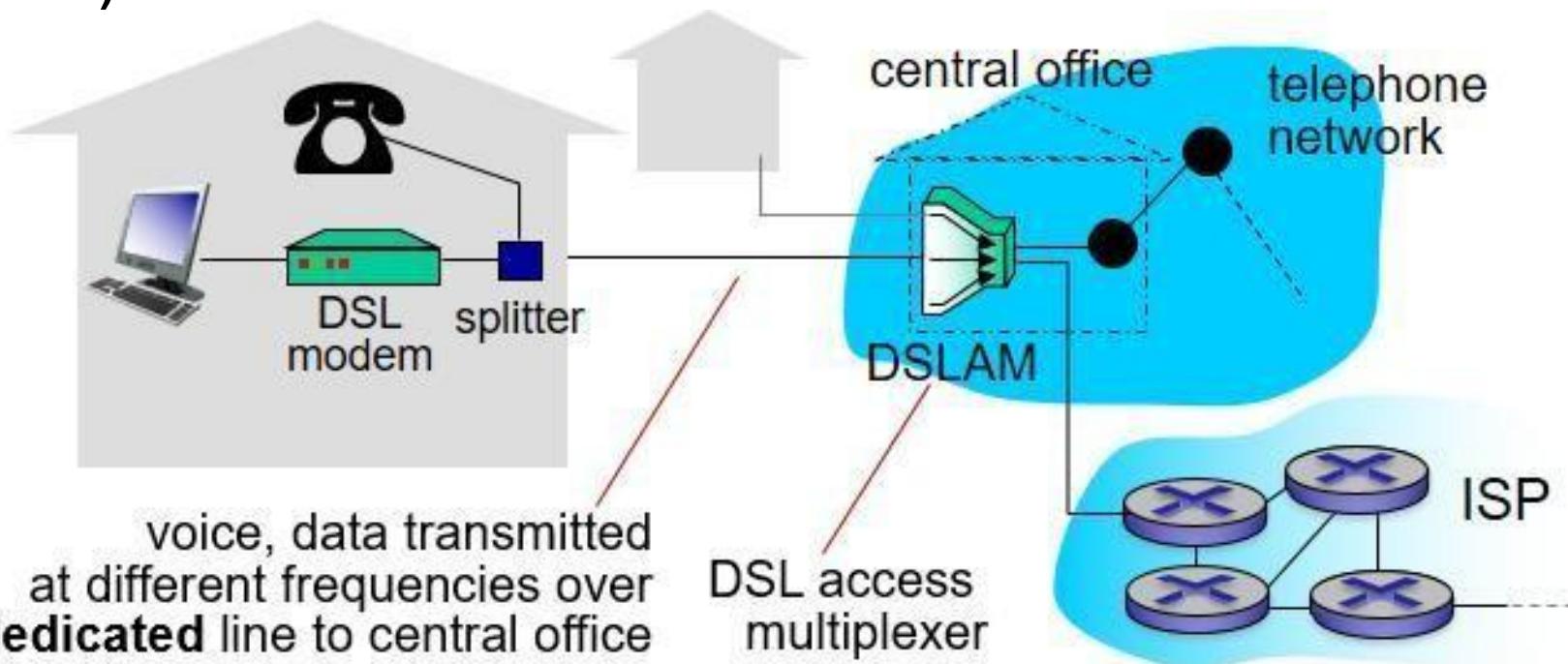


Access Networks and Physical Media

- Q: How to connect end systems to edge router?
 - residential access nets
 - institutional access networks (school, company)
 - mobile access networks
- keep in mind:
 - bandwidth (bits per second) of access network?
 - shared or dedicated?



Access Network: Digital Subscriber Line (DSL)

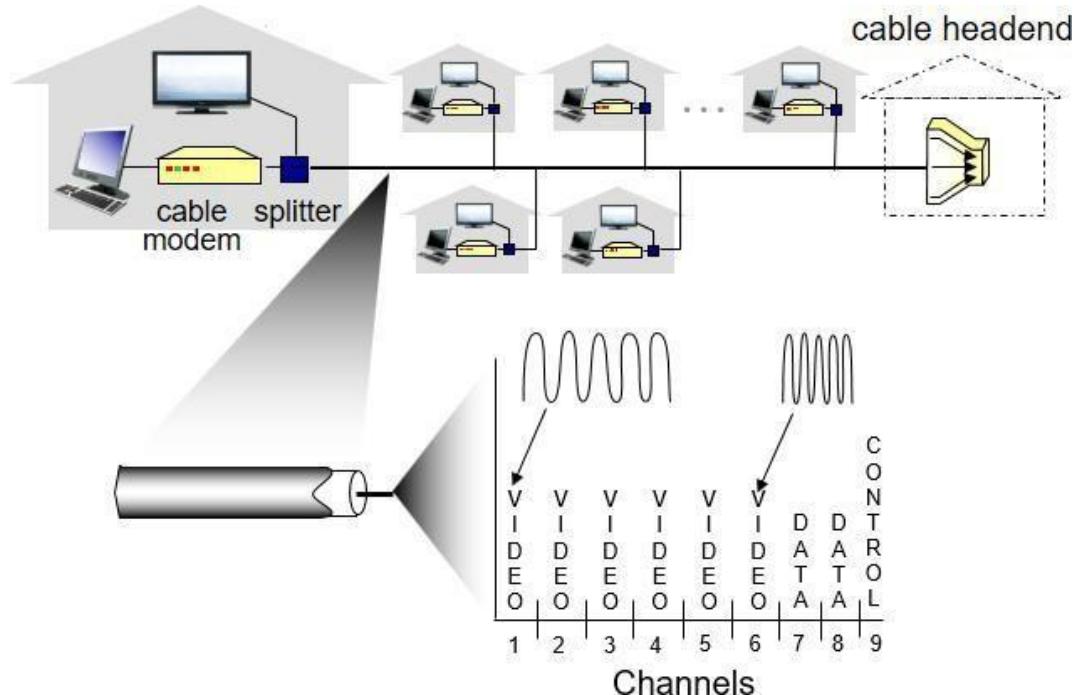


Access Network: Digital Subscriber Line (DSL)

- use existing **telephone line** to central office DSLAM (Digital subscriber line access multiplexer)
- data and traditional telephone signals are encoded at different frequency (FDM)
 - A high-speed downstream channel, in the 50 KHz to 1 MHz band
 - A medium-speed upstream channel, in the 4 KHz to 50 kHz band
 - An ordinary two-way telephone channel, in the 0 to 4 KHz band
- data over DSL phone line goes to Internet
- voice over DSL phone line goes to telephone net
- 3.5-16 Mbps upstream transmission rate
- 24-52 Mbps downstream transmission rate
 - 100 Mbps to 1 Gbps has been reached

Access Network: Cable Internet Access

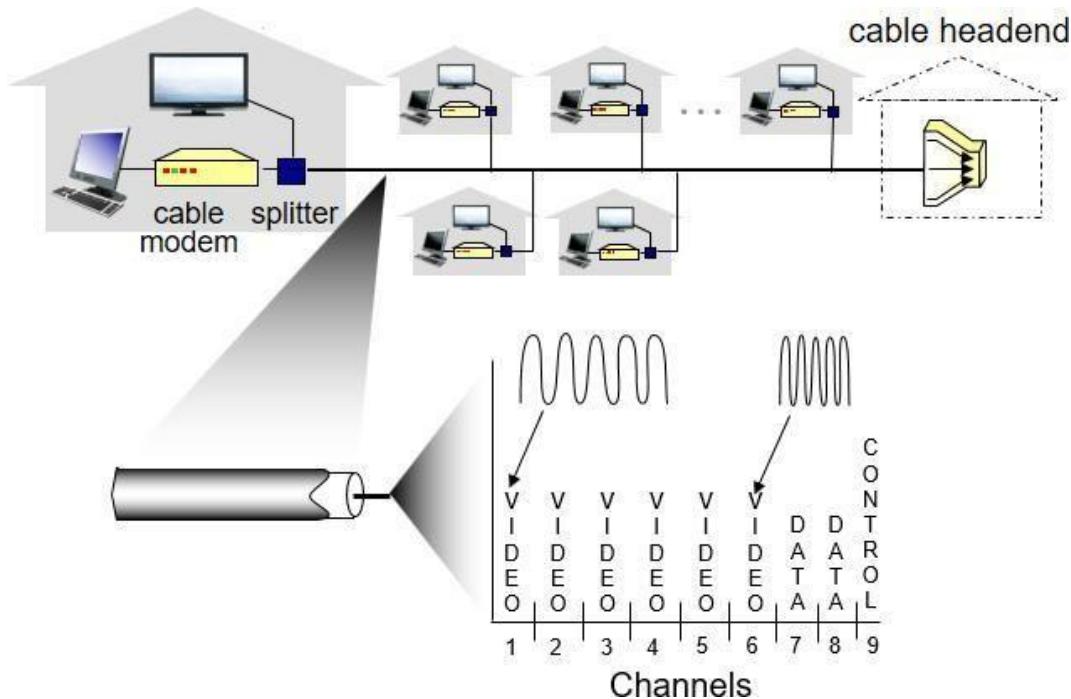
- Use of the cable television company
- cable modem are needed (similar to DSL modem)
- At the cable headend, there is Cable Modem Termination System (**CMTS**): similar to DSLAM
 - turn the analog signal sent from the cable modem into digital format
- Two channels: downstream and upstream



Access Network: Cable Network

frequency division

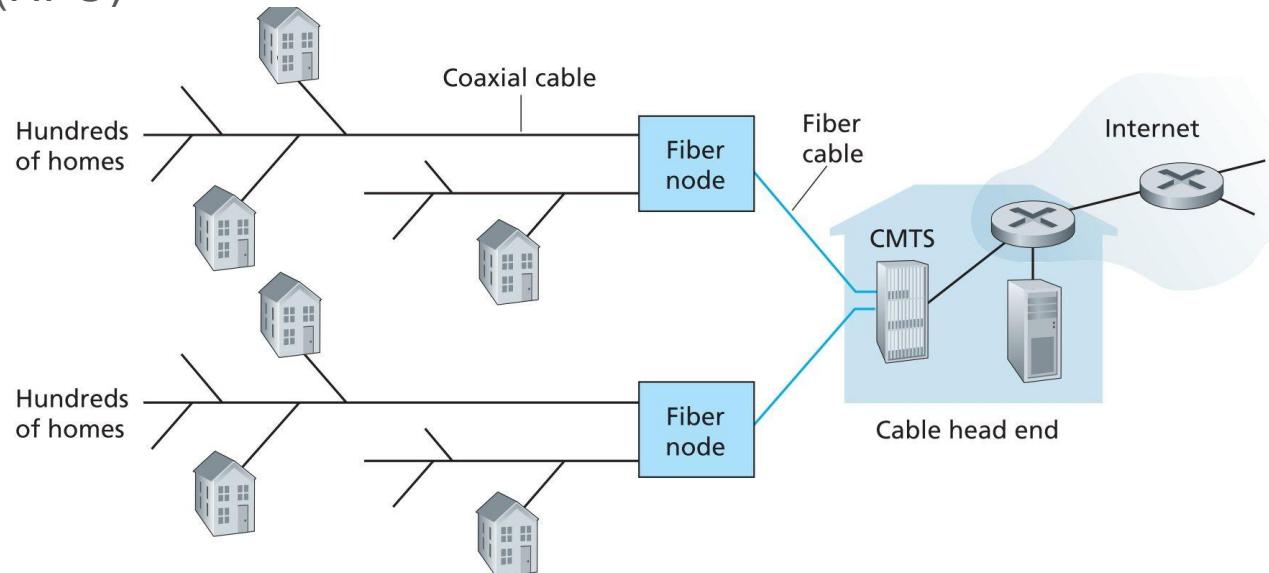
multiplexing: different channels transmitted in different frequency bands



Access Network: Cable Network

- Cable Modem Termination System (CMTS)
 - turns the analog signal sent from the cable modems in many downstream homes back into digital format
- Hybrid Fiber Coax (HFC)

Data, TV transmitted at different frequencies over shared cable distribution network



Access Network: Cable Network

- HFC: hybrid fiber coax
 - asymmetric: up to 1.2 Gbps downstream transmission rate, 100Mbps upstream transmission rate
- network of cable, fiber attaches homes to ISP router
 - **shared broadband medium:** homes share access network to cable headend
 - unlike DSL, which has dedicated access to central office

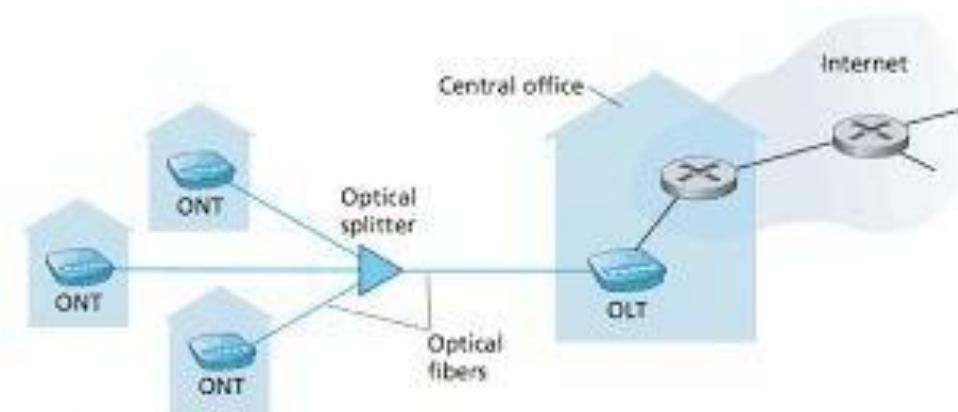
Question

Is HFC transmission rate dedicated or shared among users? Are collisions possible in a downstream HFC channel? Why or why not?

Answer: Yes, it is shared. There is no collision possible in a downstream channel since all packets emanate from a single source

Access Network: Fiber to the home (FTTH)

- An optical fiber path from the telco directly to the home
 - direct fibre
 - shared fibre
 - Active optical networks (AONs)
 - Passive Optical Networks (PONs)
 - rates in the gigabits per second

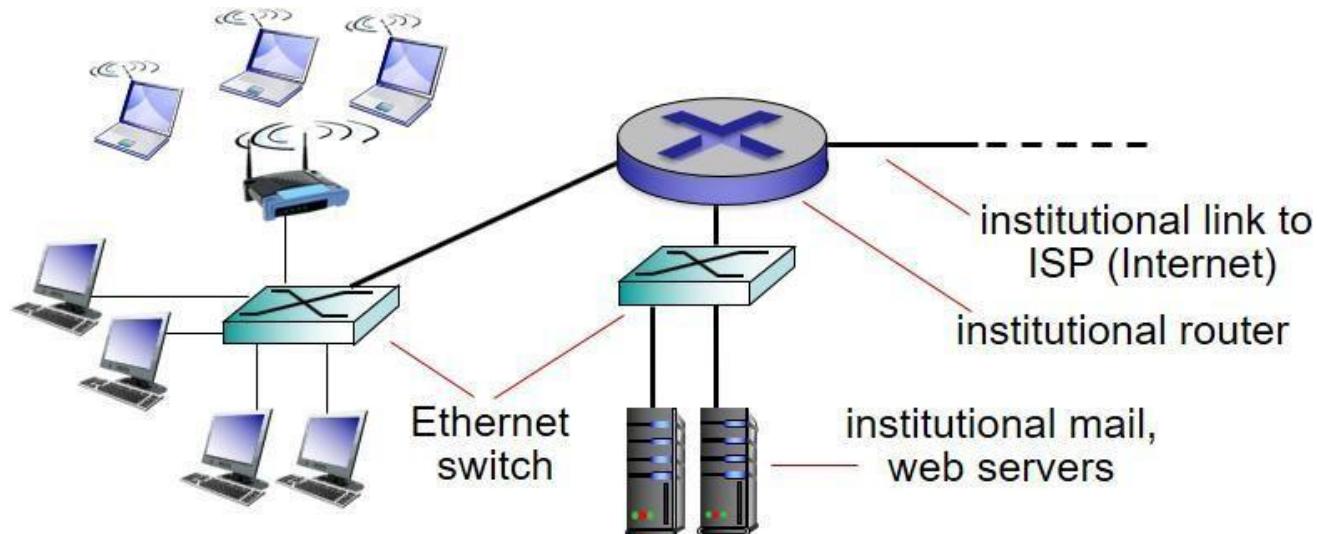


Other Access Network

- Satellite links
 - speeds of more than 1 Mbps
- Dial-up access over traditional phone lines
 - similar to DSL: a home model connects over a phone line to a modem in the ISP
 - slow rate: 56 kbps

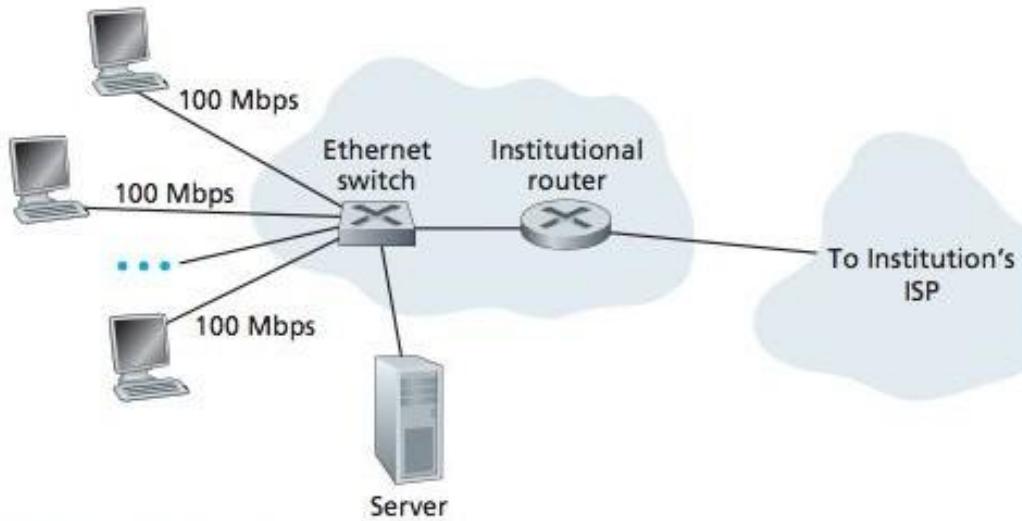
Enterprise Access Networks (Ethernet)

- typically used in companies, universities, etc.
- 10 Mbps, 100M bps, 1Gbps, 10Gbps, 100Gbps transmission rates
- end systems typically connect into Ethernet switch



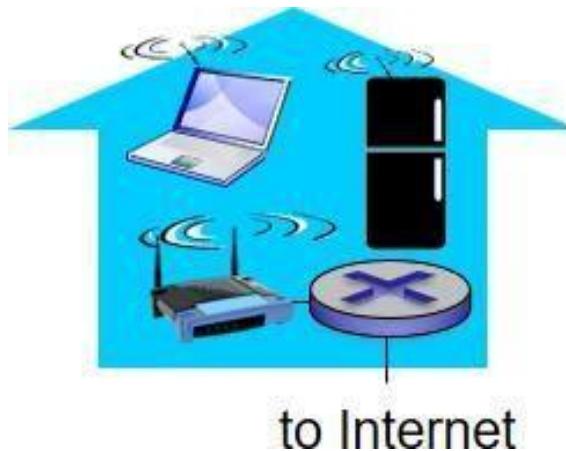
Enterprise Access Networks (Ethernet)

- typically used in companies, universities, etc.
- 10 Mbps, 100M bps, 1Gbps, 10Gbps, 100Gbps transmission rates
- today, end systems typically connect into Ethernet switch

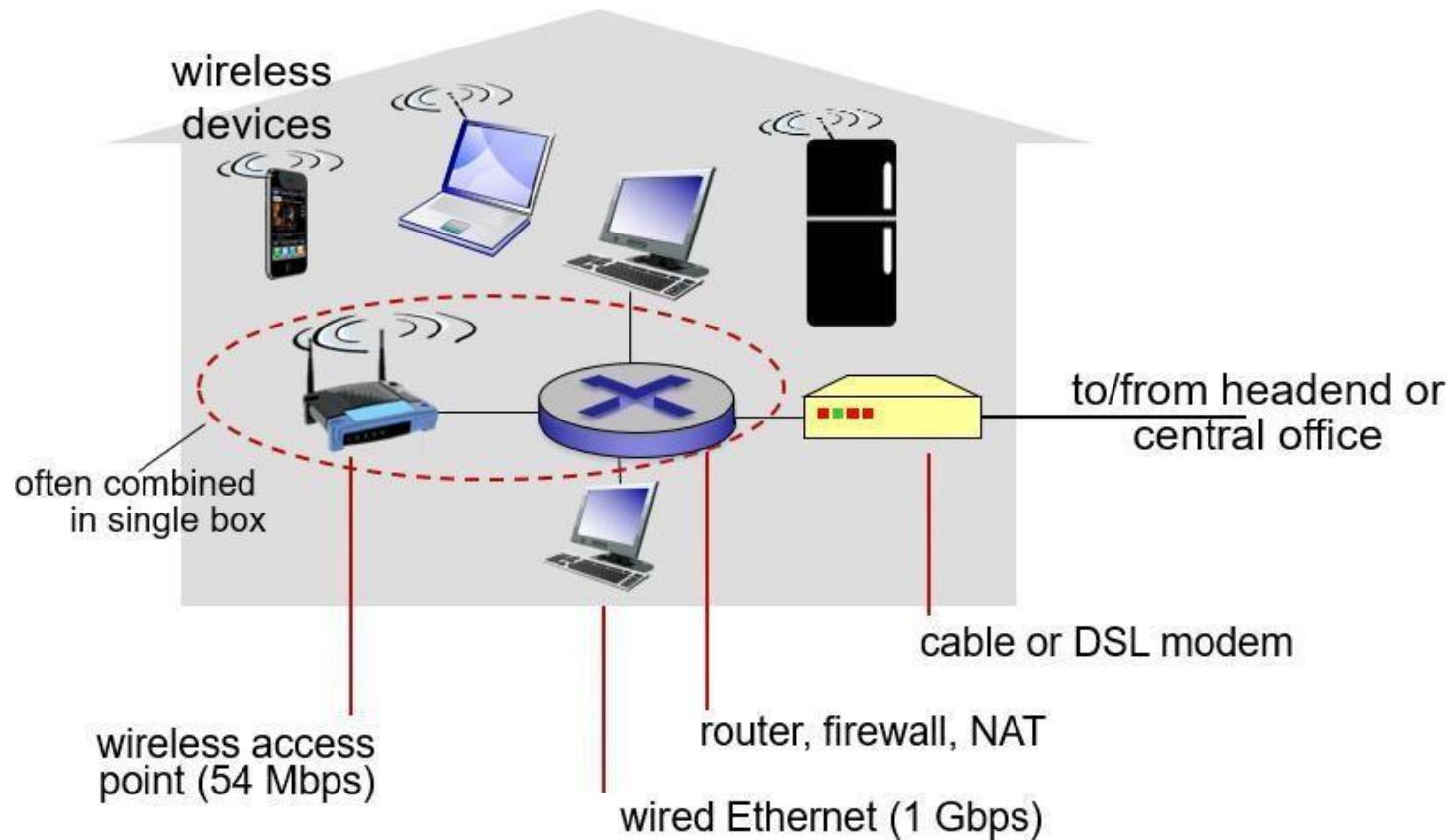


Wireless Access Networks

- shared wireless access network connects end system to router
 - via base station aka “access point”
- wireless LANs:
 - A wireless lan user must typically be within a few tens of meters of the access point
 - 802.11b/g/n (WiFi): 11, 54, 450 Mbps transmission rate



Access Network: Home Network



Wireless Access Networks

- wide-area wireless access
 - provided by telco (cellular) operator
 - Employ the same wireless infrastructure used for cellular phone
 - A user needs to be within a few tens of kilometers of the base station
 - 3G: speed > 1Mbps
 - LTE: speed > 10 Mbps
 - 3G, 4G, 5G



Access Networks Summary

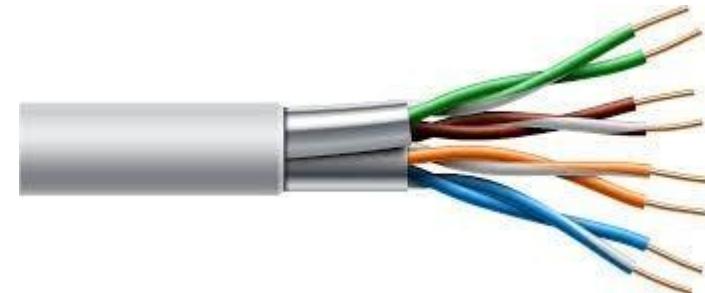
- Dial up modems: up to 56 kbps, bandwidth is dedicated
- ADSL: up to 52 Mbps downstream and 16 Mbps upstream, bandwidth is dedicated
- HFC: downstream rates up to 1.2 Gbps and upstream rates of up to 100 Mbps, bandwidth is shared
- FTTH: order of Gbps download, bandwidth is not shared

Physical Media

- For each transmitter/receiver pair:
 - Bit is sent by propagating electromagnetic waves or optical pulses across a physical medium
- **physical link:** what lies between transmitter & receiver
- **guided media:**
 - electromagnetic signals propagate in solid media: copper, fiber, coax
- **unguided media:**
 - signals propagate freely and in atmosphere, e.g., radio
 - in wireless lan or a satellite channel

Physical Media

- **Twisted pair (TP)**
 - The least expensive and most commonly used medium
 - two insulated copper wires, twisted together to reduce the electrical interference from similar pairs close by.
 - data rate depends on the thickness of the wire and distance between transmitter and receiver
 - Category 5: 100 Mbps, 1 Gbps Ethernet
 - Category 6: 10G bps for distances up to a hundred meters
- Used in high-speed LAN networking
- Used in dial-up modem and DSL

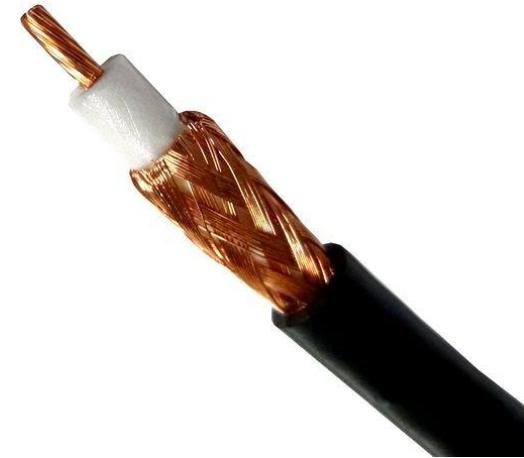


https://higherlogicdownload.s3.amazonaws.com/CEDIA/UploadedImages/5092830d-729a-4c31-bddd-6bff0f2af4ff/Cable_FUTP.png

Physical Media: Coaxial cable

- **Coaxial Cable:**

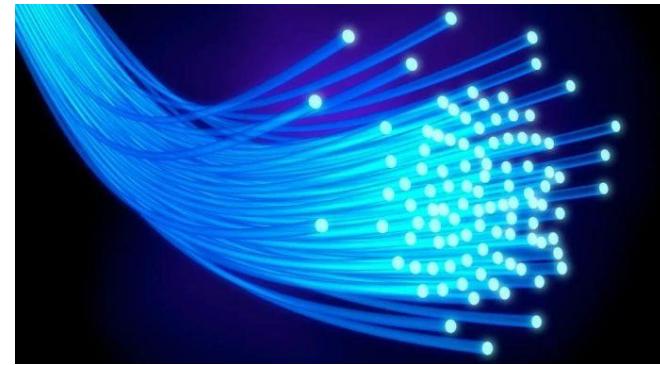
- two concentric copper conductors
- Higher transmission rate
- Used in Cable television and Cable Internet Access
- multiple channels on cable (shared)
- guided **shared** medium



<https://cdn3.volusion.com/rcmqs.fsseg/v/vspfiles/photos/DAVISRG8X-2.jpg>

Physical Media: Fiber Optical cable

- Fiber Optic Cable:
 - thin, flexible medium
 - carrying light pulses, each pulse a bit
 - high-speed operation:
 - high-speed point-to-point transmission (e.g., 10's-100's Gbps transmission rate)
 - low error rate:
 - repeaters spaced far apart
 - immune to electromagnetic noise
 - good for long distance transmission
 - long distance telephone network, overseas links
 - backbone of the Internet
- Problems
 - high costs of optical devices: transmitter, receiver, and switches
 - Not a good option for inside the LAN or residential access network



https://cdn.shopify.com/s/files/1/0020/4433/0057/articles/fiber-optic-cable_1024x1024.png?v=1555004061

Physical Media: Wireless Radio

- signal carried in electromagnetic spectrum
- no physical “wire”
- The characteristics depends on
 - propagation environment
 - Distance over which a signal is to be carried
 - The signal strength is decreased as it travels over a distance and around/through obstructing objects
 - Reflection: signal reflection off of interfering objects
 - Interference (due to other transmission and electromagnetic signals)

Physical Media: Radio link types

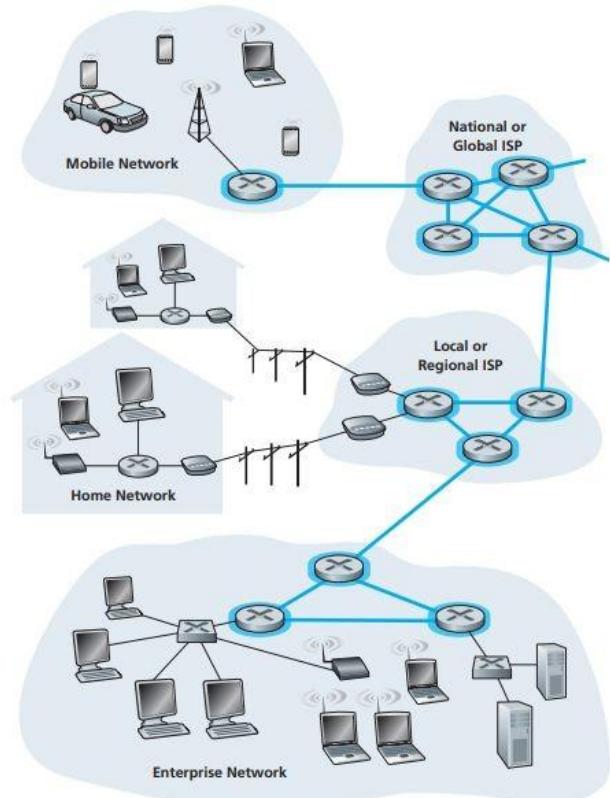
- Radio link types
 - Bluetooth:
 - short distances, limited rates
 - Wireless LAN (WiFi)
 - 10-100's Mbps; 10's of meters
 - wide-area (e.g., 4G cellular)
 - 10's Mbps over ~10 Km
 - Operate in a wide area (tens of kilometers)
 - Cellular access technology
 - terrestrial microwave
 - point-to-point; 45 Mbps channels
 - satellite
 - up to 45 Mbps per channel
 - 270 msec end-end delay

Introduction

- What's the Internet?
- What's a protocol?
- network edge; hosts, access net, physical media
- **network core: packet/circuit switching Internet structure**
- Internet structure
- protocol layers, service models
- performance: loss, delay, throughput
- security
- history

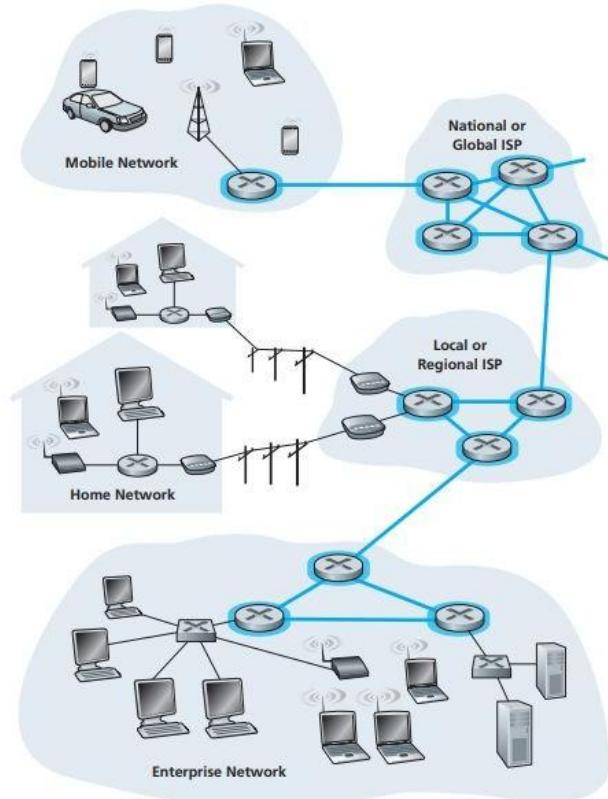
A Closer Look at Network Structure:

- network core:
 - interconnected routers
- Two fundamental approaches to moving data through a network of links and switches
 - **Circuit Switching**
 - **Packet Switching**



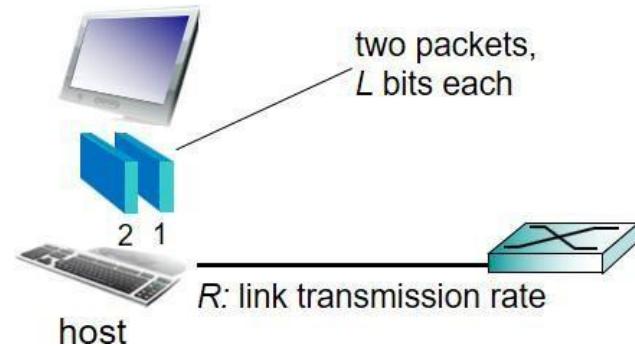
Packet Switching: How

- **packet-switching:**
 - **hosts**: break application messages into packets
 - **packet-switch**: forward packets from one router to the next, across links on path from source to destination
 - each packet transmitted at **full link capacity**
 - if a source is sending a packet of L bits over a link with transmission rate R bits/sec, then the time to transmit the packet is L/R seconds



Packet Switching: Characteristics

- host sending function:
 - takes application message
 - breaks into smaller chunks, known as packets, of length L bits
 - transmits packet into access network



Packet Switching: Characteristics

- **Store-and-Forward Transmission**

- packet switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link.

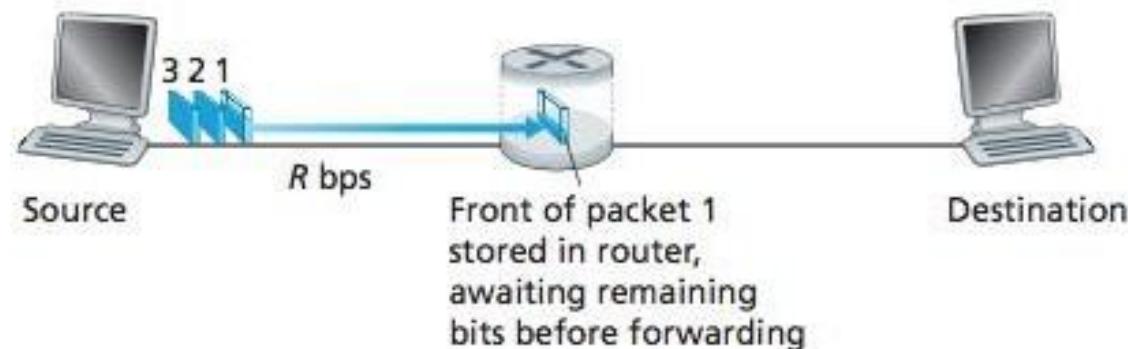
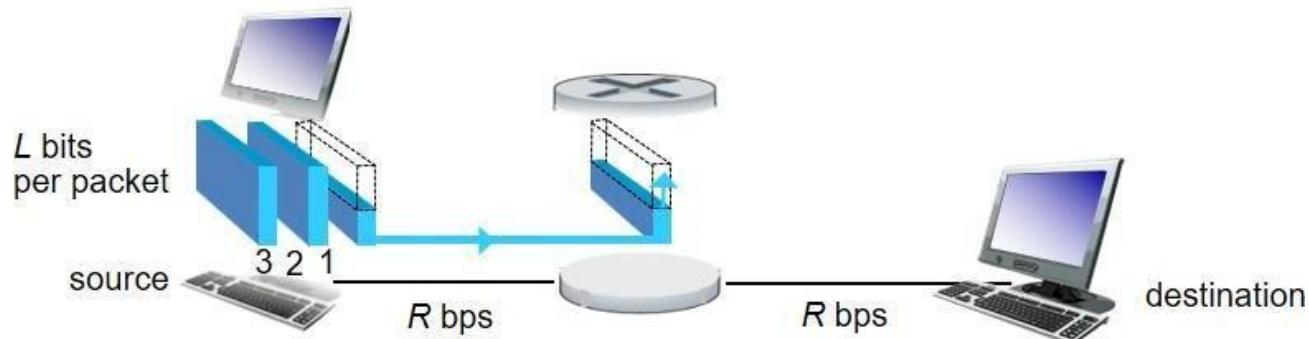


Figure 1.11 • Characteristics of Store-and-Forward

Packet Switching

- **Store-and-Forward Transmission**

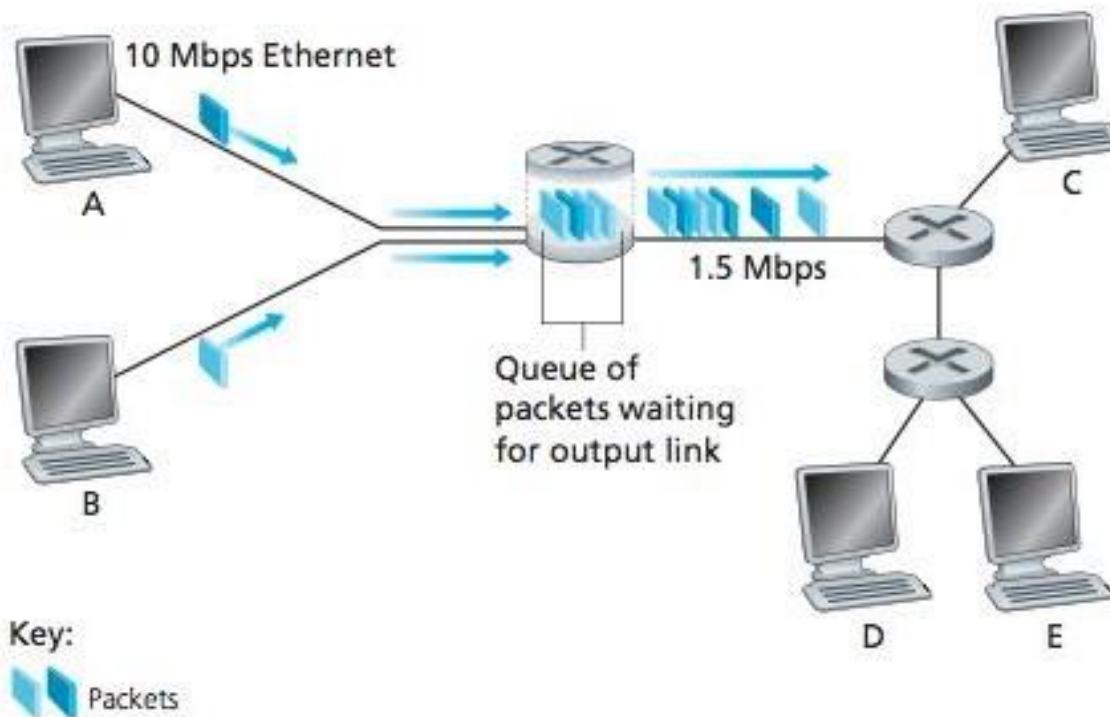
- packets in a packet-switched router are transmitted only after all the packets that have arrived before it have been transmitted.
- **causes delay**
- What is the time needed to send one packet of L bits from source to destination?
- Sources takes L/R seconds to transmit (push out) L -bit packet into link at R bps
- store and forward: entire packet must arrive at router before it can be transmitted on next link
- End-to-end delay = $2L/R$ (assuming zero **propagation delay**)



Packet Switching

- **Queueing Delay and Packet Loss**
 - For each attached link, the packet switch has an **output buffer** (also called an output queue)
 - stores packets that the router is about to send into that link
 - **queuing delays:** depend on the level of congestion in the network
 - **packet loss** will occur
 - the amount of buffer space is limited.
 - an arriving packet may find that the buffer is completely full with other packets waiting for transmission,
 - —either the arriving packet or one of the already-queued packets will be dropped

Packet Switching

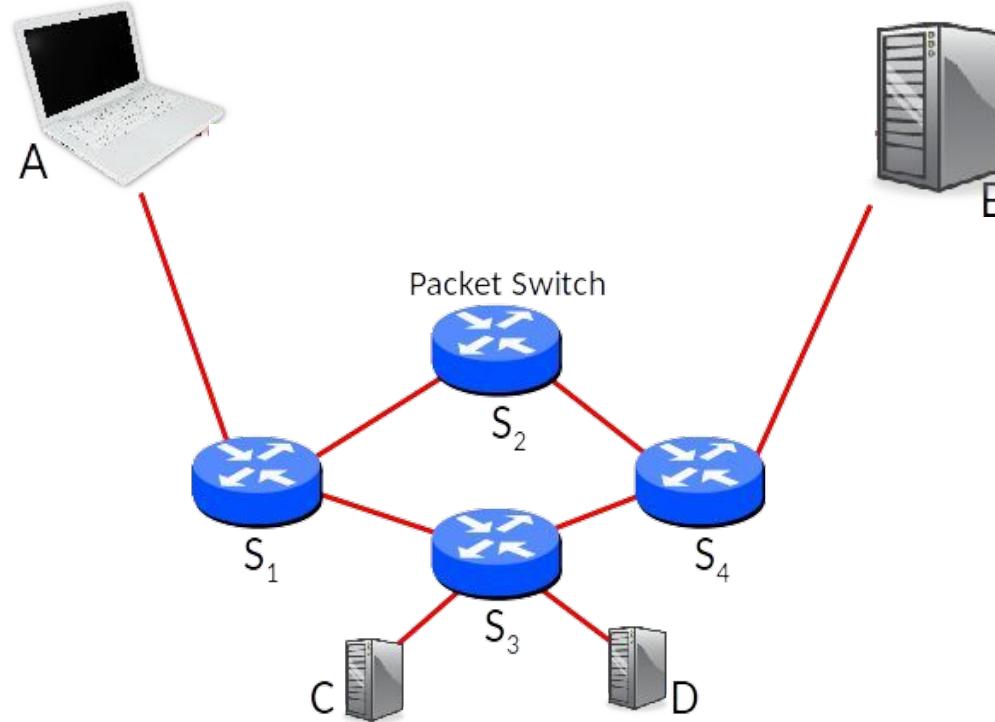


Packet Switching

- **Forwarding Tables and Routing Protocols**
 - how a router forwards the packet?

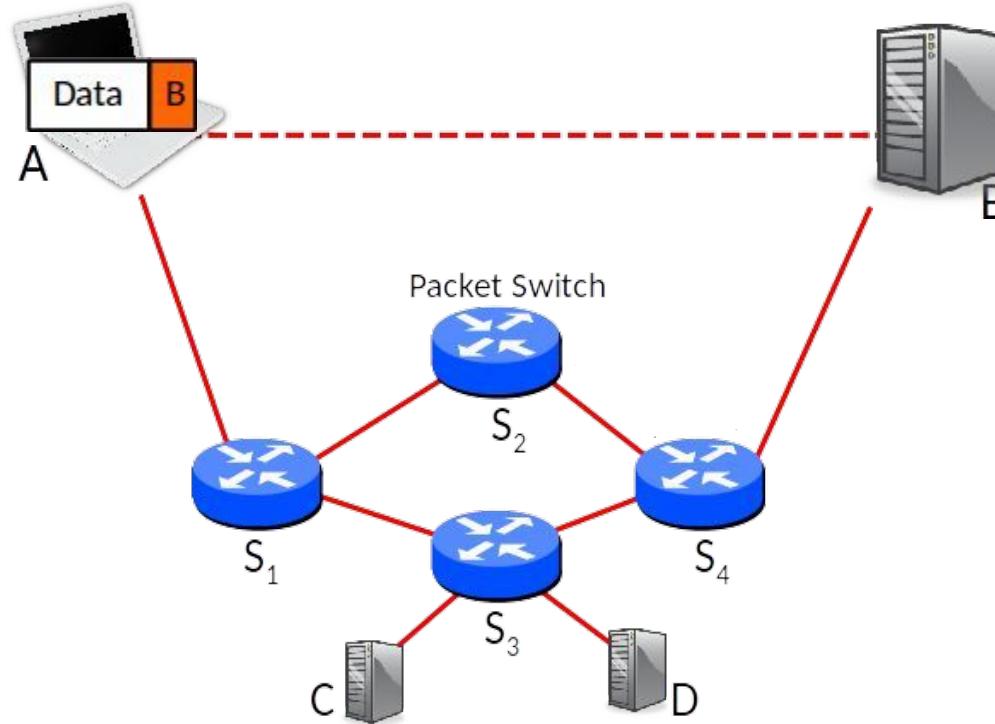
Packet Switching

- Forwarding Tables and Routing Protocols



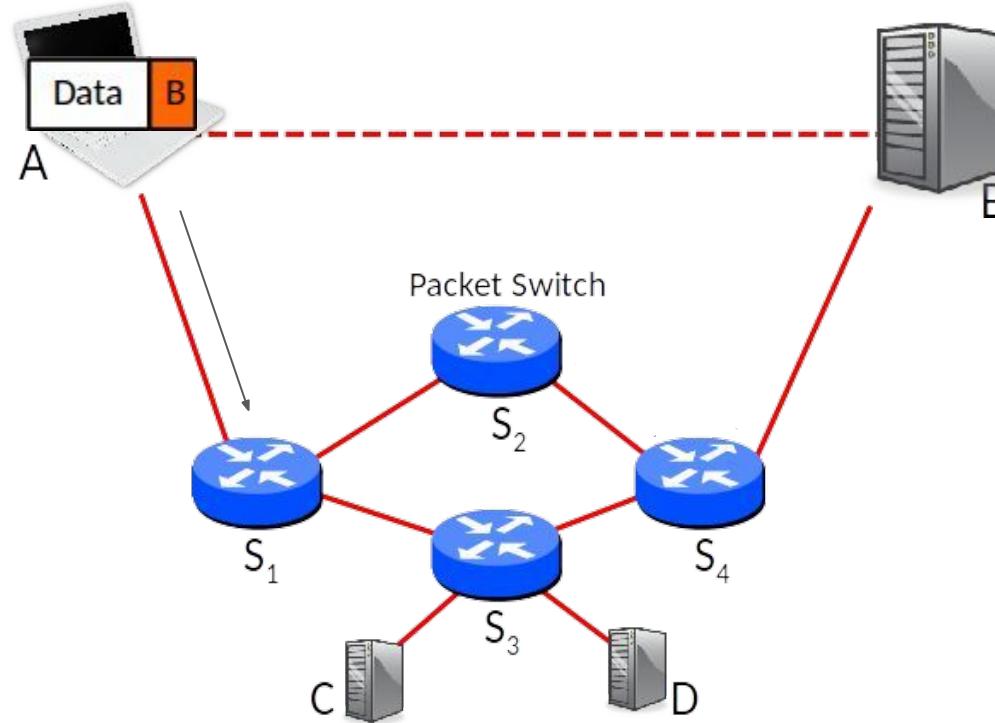
Packet Switching

- Forwarding Tables and Routing Protocols



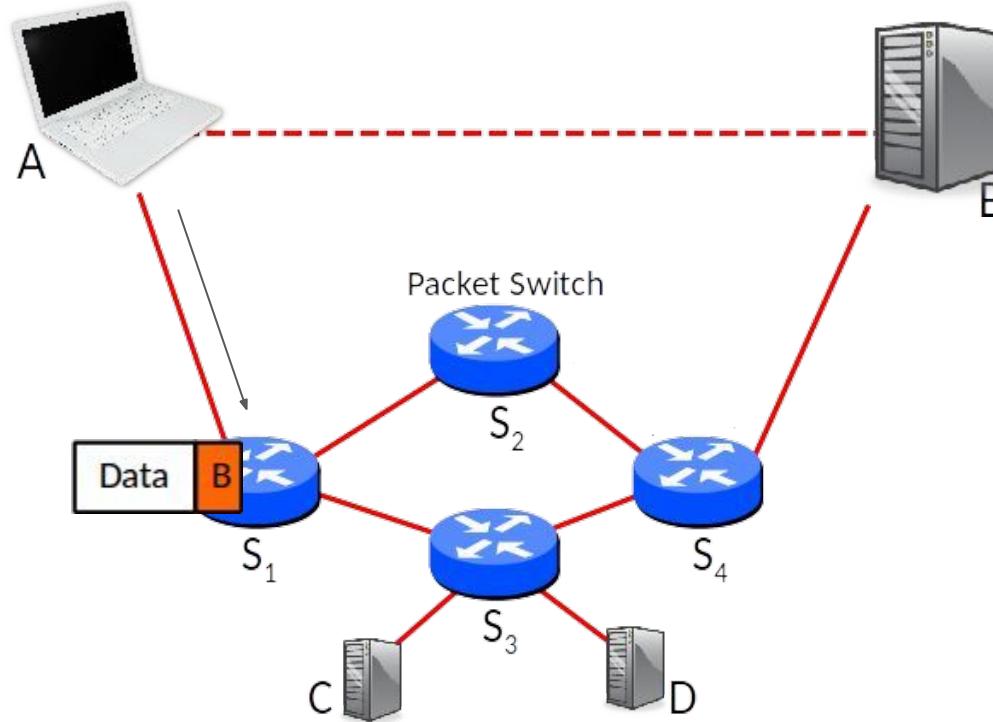
Packet Switching

- Forwarding Tables and Routing Protocols



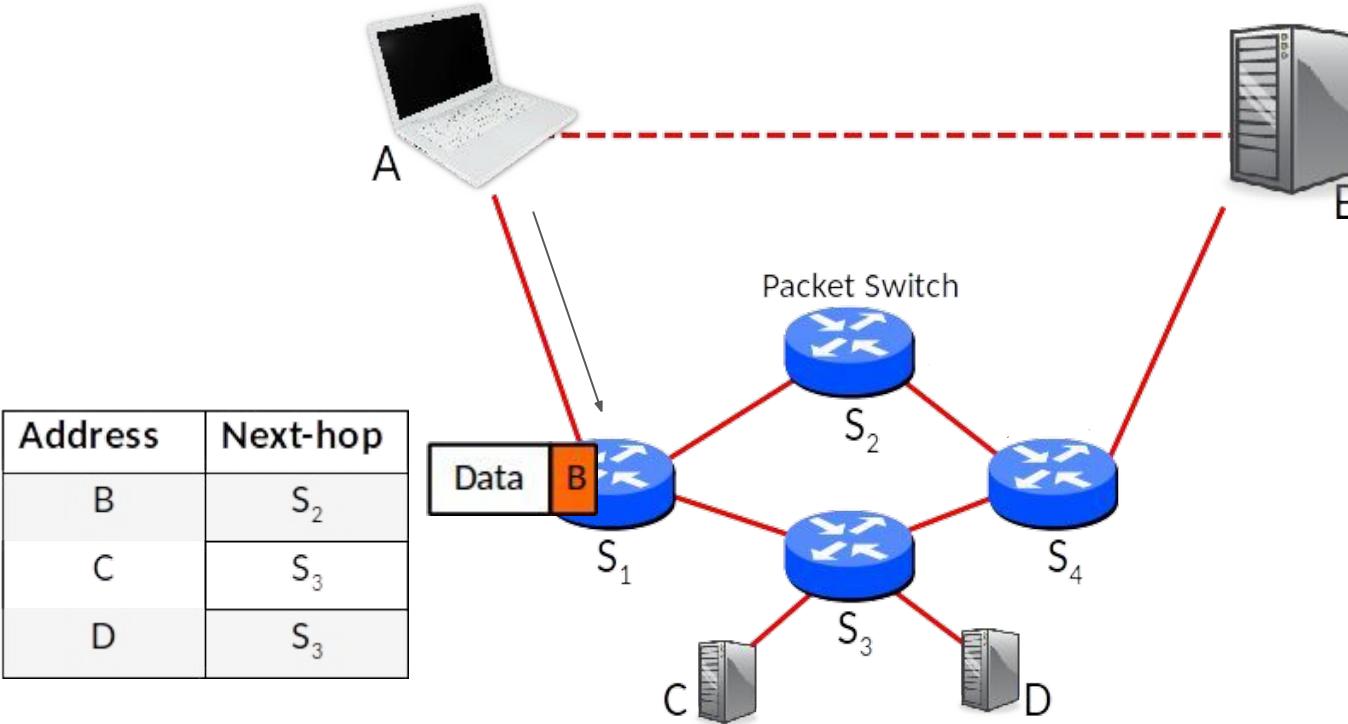
Packet Switching

- Forwarding Tables and Routing Protocols



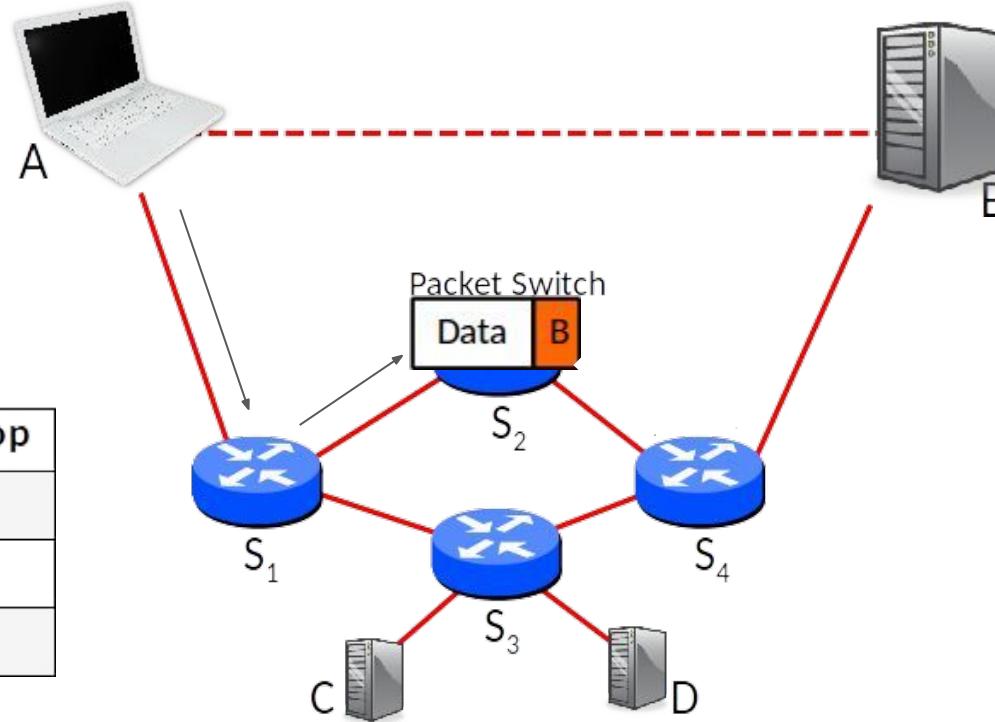
Packet Switching

- Forwarding Tables and Routing Protocols



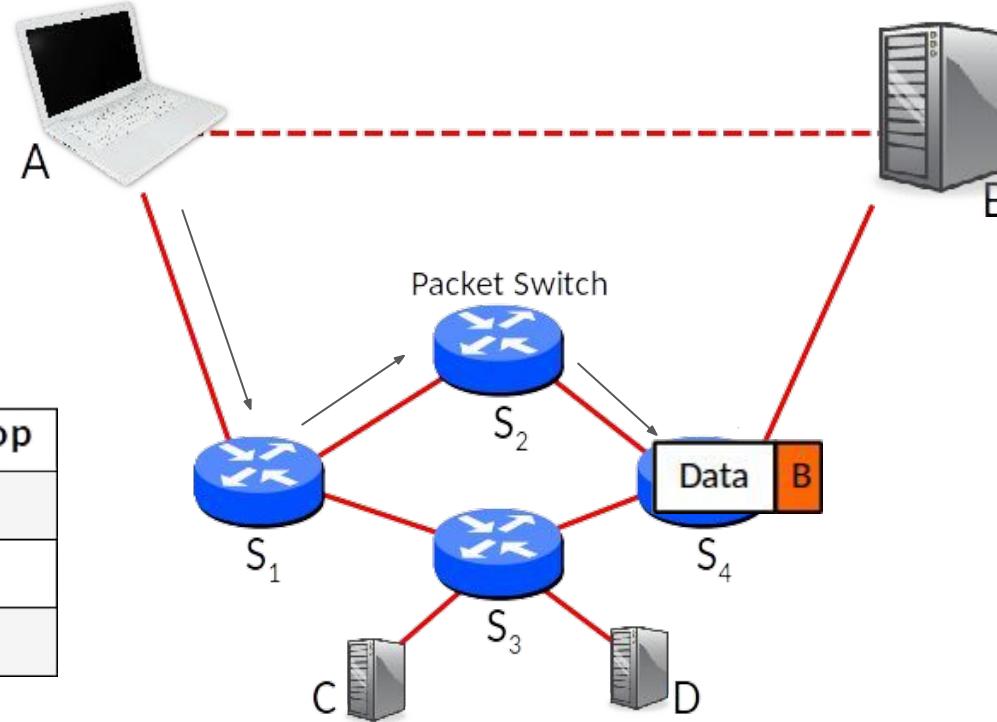
Packet Switching

- Forwarding Tables and Routing Protocols



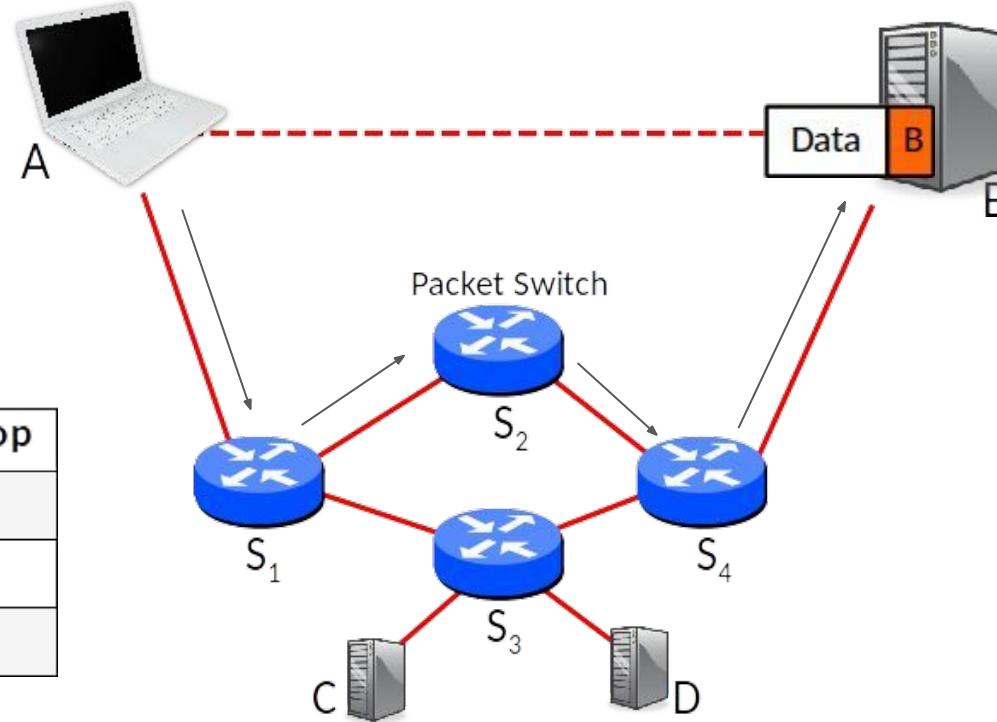
Packet Switching

- Forwarding Tables and Routing Protocols



Packet Switching

- Forwarding Tables and Routing Protocols



end-to-end routing in packet-switches: Analogy

Packet Switching

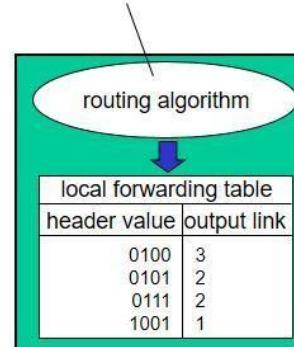
- **Forwarding Tables and Routing Protocols**
 - how a router forwards the packet?
 - using forwarding table
 - How the forwarding table are put into packet switches
 - using routing protocols
 - example: The routing protocol may find the shortest path from each source to each destination
 - traceroute

Packet Switching

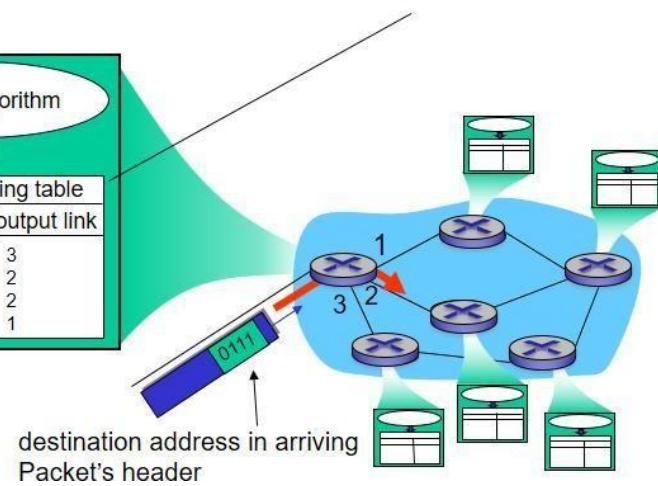
- send data anytime in packets (having a header containing the address)
 - like an envelope tells the post office where to send the letter.

routing: determines source-destination route taken by packets

- routing algorithms



forwarding: move packets from router's input to appropriate router output



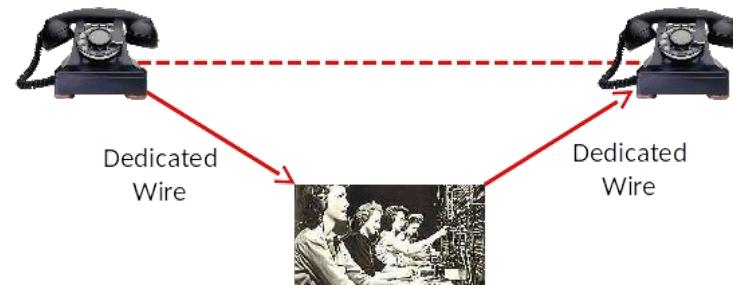
Circuit Switching

- Another approach to moving data through a networks of links and switches
 - the resources needed along a path to provide for communication between end systems are reserved for the duration of the communication between the end systems
 - buffers, links transmission rate
 - In packet-switches the resources are not reserved, but used on-demand that's why there is wait
 - Restaurant analogy

Circuit Switching

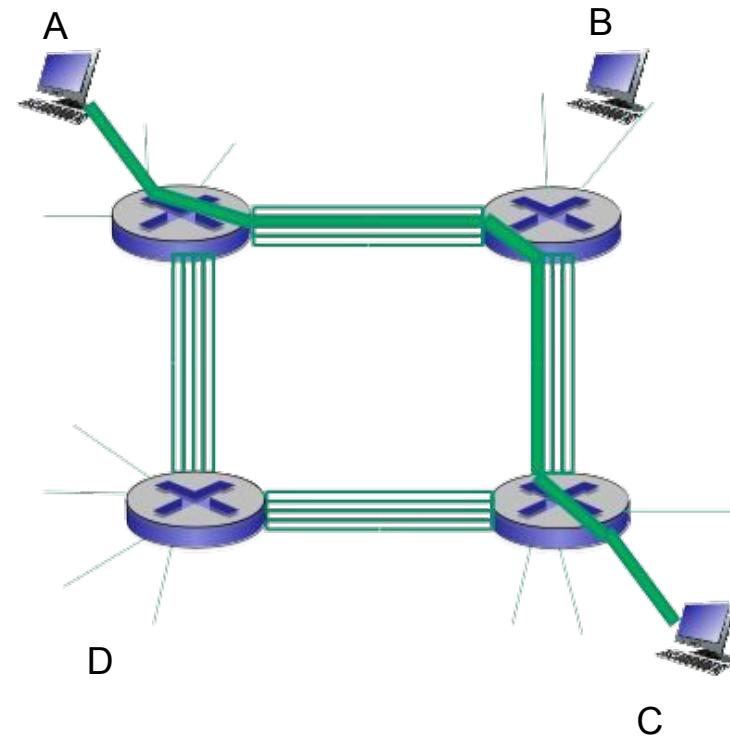
- Traditional telephone networks
 - Each call has its own private, guaranteed, isolated data rate from end-to-end.
 - end-end resources allocated to, reserved for “call” between source & dest:
 - A call has three phases:
 - Establish circuit from end-to-end (“dialing”)
 - Communicate
 - Close circuit (“tear down”)

Circuit Switching



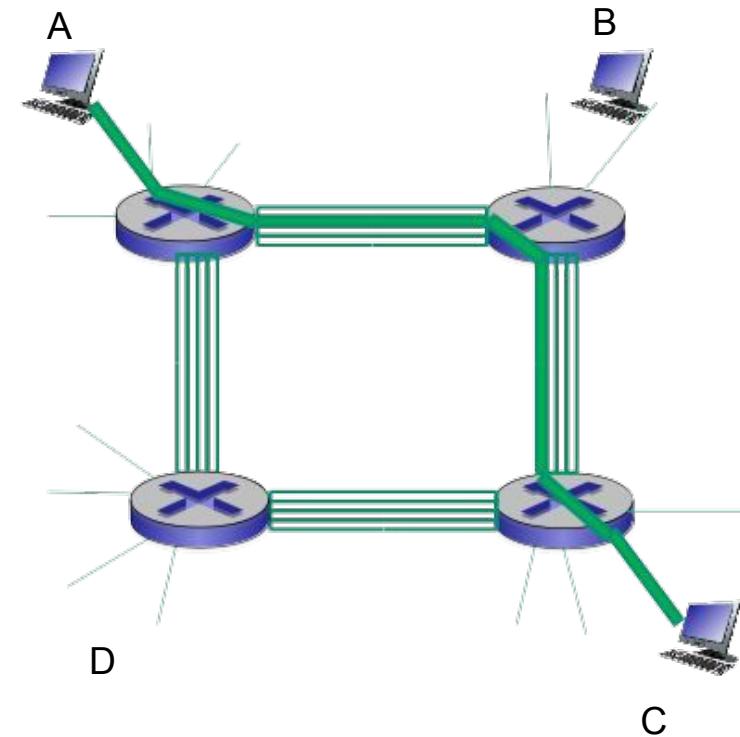
Circuit Switching Characteristics

- A circuit switched network:
 - The four circuit switches are connected by four links
 - Each link has four circuits, so it can support four simultaneous connections
 - When hosts A and C want to communicate, the network must first reserve one circuit on each of the two links.
 - because each link has 4 circuits, for each link used by the end-to-end connection, the connection gets one fourth of the link's total transmission capacity for the duration of the connection



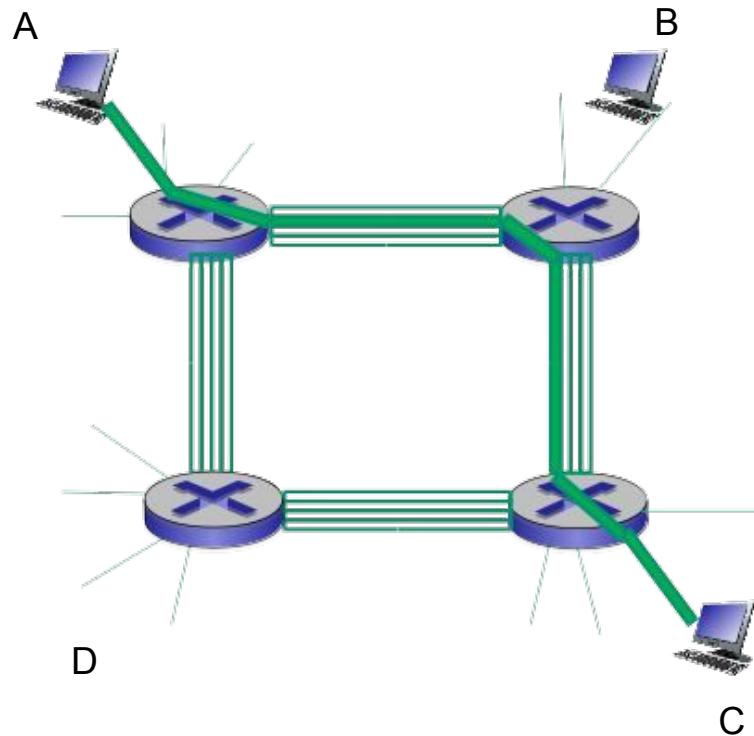
Circuit Switching Characteristics

- dedicated resources: no sharing
- guaranteed performance
- circuit segment idle if not used by call (no sharing)



Circuit Switching Characteristics

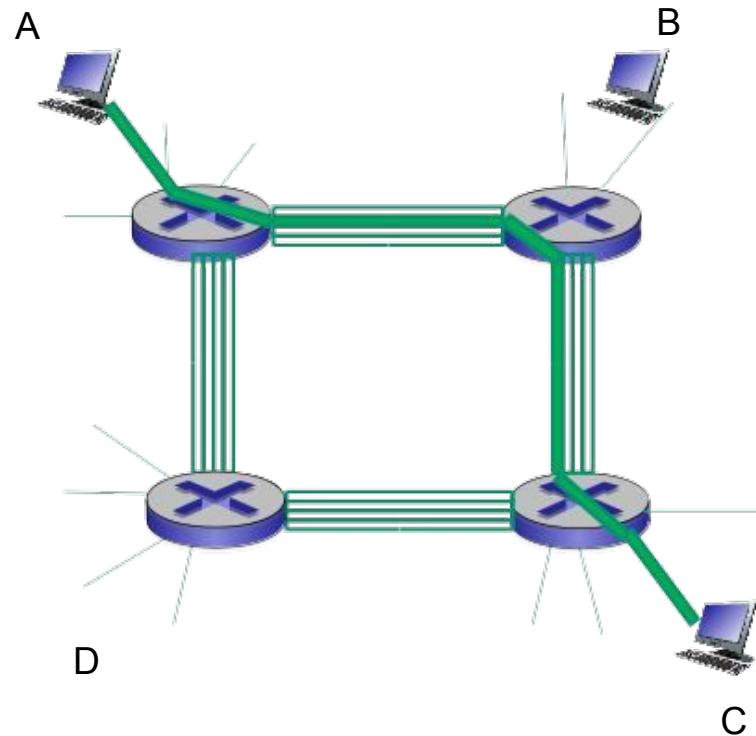
- **Question:**
 - if each link has 1 Mbps transmission rate,
what is the transmission rate of an end-to-end
circuit switch connection between A and C?
- A. 250 kbps
B. 1 Mbps
C. 4 Mbps



Circuit Switching Characteristics

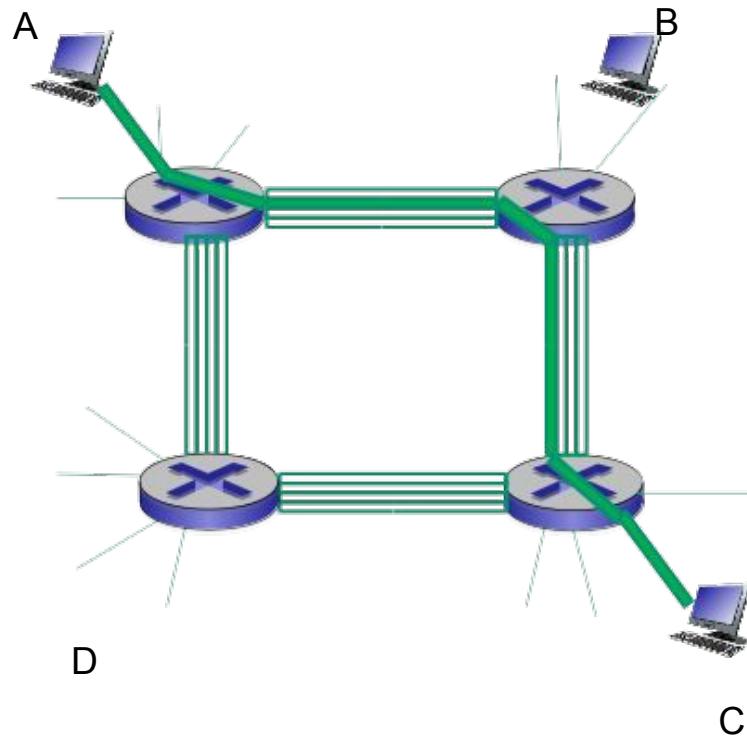
- **Question:**
 - if each link has 1 Mbps transmission rate, what is the transmission rate of an end-to-end circuit switch connection between A and C?

- A. 250 kbps
- B. 1 Mbps
- C. 4 Mbps



Circuit Switching Characteristics

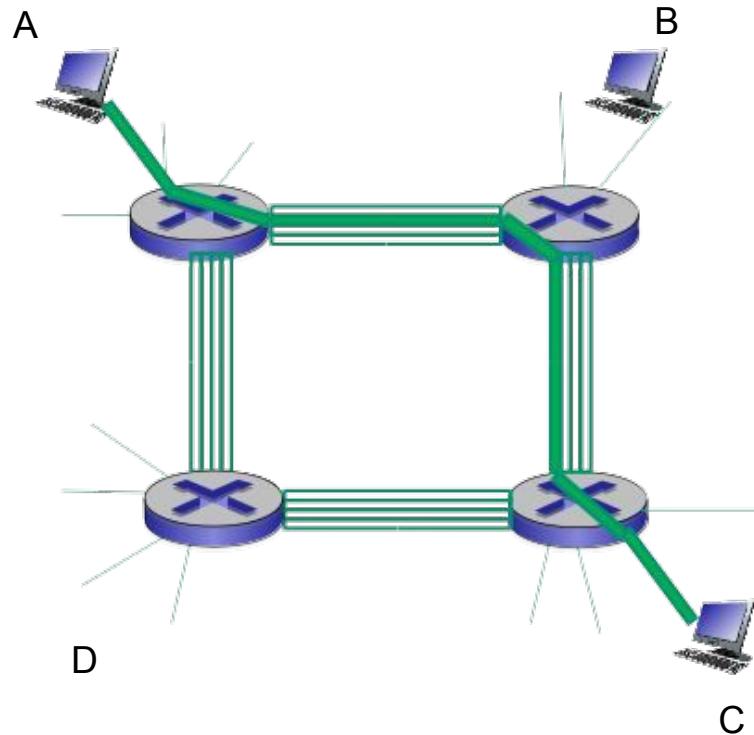
- **Question:**
 - What is the maximum number of simultaneous connections that can be in progress at any one time in this network?
- A. 4
B. 8
C. 16
D. 32



Circuit Switching Characteristics

- **Question:**
 - Suppose that all connections are between switches A and C. What is the maximum number of simultaneous connections that can be in progress?

- A. 4
- B. 8
- C. 16
- D. 32

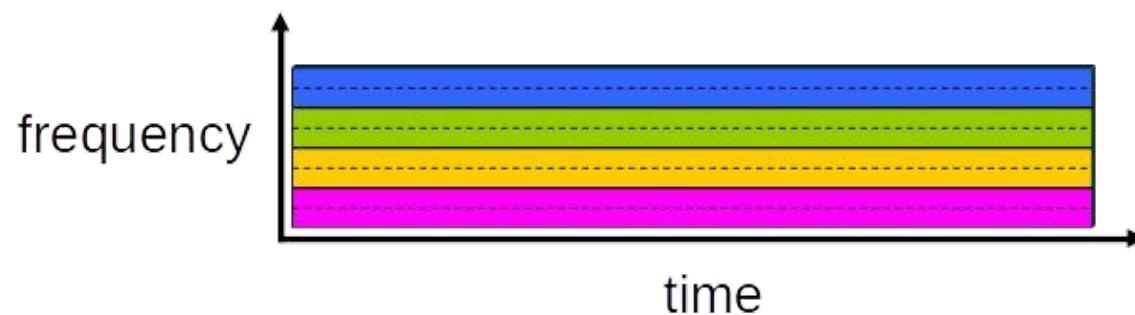


Circuit Switching: How to implement

- **FDM: Frequency Division multiplexing**
 - frequency spectrum of a link is divided up among the connections established across the link
 - The link dedicates a frequency band to each connection

Example:

4 users



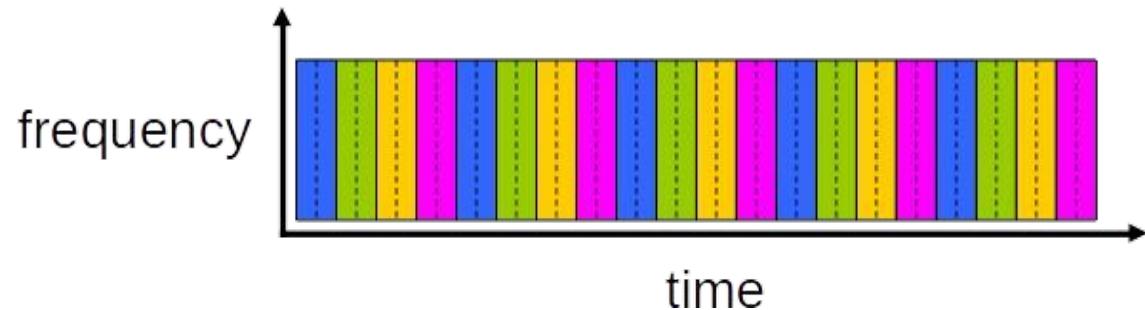
Circuit Switching: How to implement

- **TDM: Time Division Multiplexing**

- Time is divided into frames of fixed duration
- Each frame is divided into a fixed number of time slots
- When the network establishes a connection across a link, one time slot in every frame is assigned to a connection
- These slots are dedicated for the sole use of the connection

Example:

4 users



Circuit Switching: FDM Versus TDM

FDM: each circuit continuously gets a fraction of the bandwidth

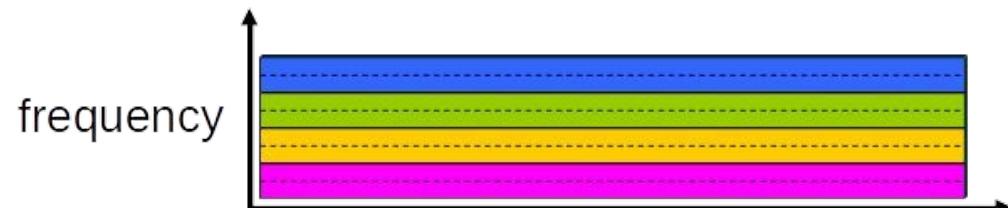
TDM: each circuit gets all of the bandwidth periodically during brief intervals of time (during a slot)

Example:

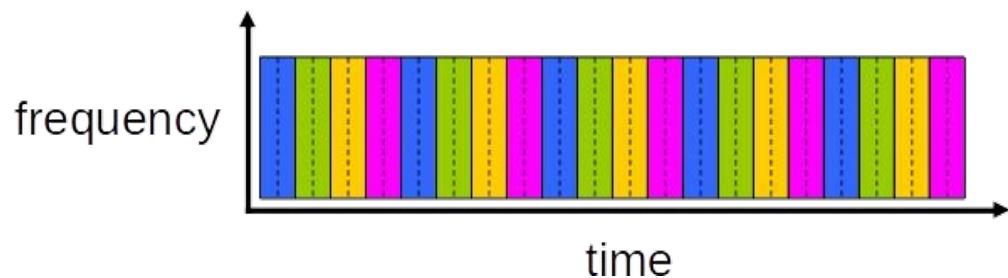
4 users



FDM



TDM



Circuit Switching: Numerical Activity

- **Question:** How long does it take to send a file of 640000 bits from Host A to Host B over a circuit-switched network?
 - All the links in the network use TDM with 24 slots
 - All links have a bit rate of 1.536 Mbps
 - It takes 50 msec to establish an end-to-end circuit before Host A can begin to transmit the file.

Hint: $1.536/24 = 64$

- A. 10 seconds
- B. 10.05 seconds
- C. 10000 seconds

Circuit Switching vs. Packet Switching

- Packet switching advantages

- Efficient use of expensive links
 - great for bursty data: Computer communication tends to be very **bursty** e.g. typing over an ssh connection, or viewing a sequence of web pages. If each communication has a dedicated circuit, it will be used very inefficiently.
- better sharing of transmission capacity
- Simpler, no circuit setup
- Resilience to failure of links & routers
- No per-communication state management.
- better service for end system with different rates
- Circuit switching **pre-allocates** use of the transmission link regardless of demand
 - allocated but unneeded link time going unused,
 - while Packet switching allocates link use on demand
- The trend in today's telecommunication networks is towards packet switching

Circuit Switching vs. Packet Switching

- Packet Switching disadvantages

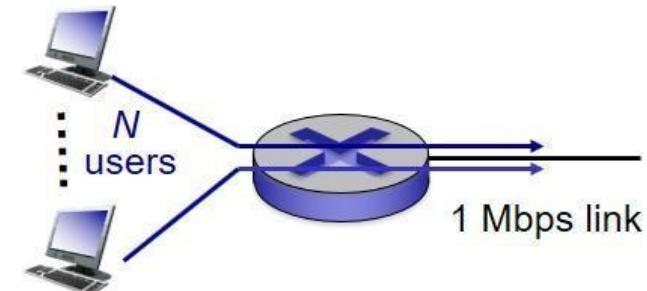
- Excessive congestion possible: packet delay and loss
 - protocols needed for reliable data transfer, congestion control
- Not suitable for real-time services (telephone calls and video conference calls) because of unpredictable end-to-end delay.
 - bandwidth guarantees needed for audio/video apps
 - Workaround

Packet Switching Versus Circuit Switching: analogy

- going from waterloo to Ottawa
- Circuit switching
 - use a map and know all the path beforehand
- Packet switching
 - ask for direction along the way
 - you go to the closest gas station and ask hot to get to the parliament
 - They direct you to 401 highway
 - You may ask for help before or not
 - On the way you may check with another gas station and ask for ottawa
 - They direct you to exit from 401 and take the other highway

Packet Switching Versus Circuit Switching: Example

- Is packet switching more **efficient**? Why?
 - Yes, it is more efficient because it packet switching allows more users to use network!
 - Example:
 - Uses share a 1Mb/s link
 - each user:
 - 100 kbps when “active”
 - active 10% of time
 - **circuit-switching**
 - 10 users can share the link
 - 100 kbps must be reserved for each user at all times
 - example: A TDM where one second frame is divided into 10 time slots of 100 ms

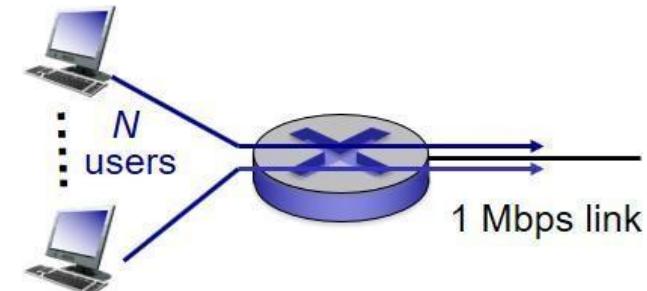


Packet Switching Versus Circuit Switching:

Example 1

- **Packet-switching:**

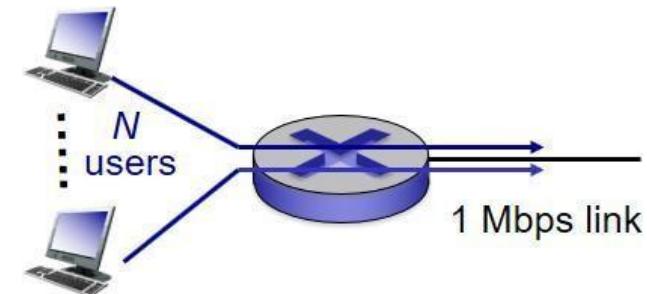
- 10 or fewer simultaneously active users (with probability 0.9996)
 - the data rate is less than 1 Mbps
 - packets flow through the link without delay (similar to circuit switching)
- more than 10 users
 - incoming rate is larger than output link rate
 - the output queue will begin to grow.
 - as the number of users increase, the delay increase
 - with 35 users, the probability of having more than 10 active users simultaneously is less than .0004



packet switching has the same performance as circuit switching
with a larger number of users

Packet Switching Versus Circuit Switching: Example 1

- Q: how did we get value 0.0004?
 - What is the probability that a given user is transmitting?
 - What is the probability that at any given time, exactly 10 users are transmitting simultaneously? (assume there are 35 users)
 - What is the probability that 11 users or more are transmitting simultaneously



Packet Switching Versus Circuit Switching:

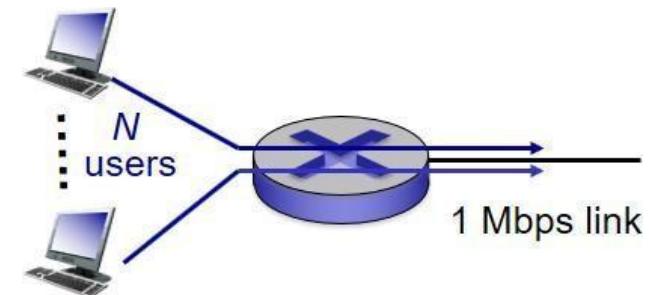
Example 1

- Q: what happens if > 35 users ?

Packet Switching Versus Circuit Switching: Example2

- There are 10 users
- One user generates **one thousand 1,000 bits** packet, while other users are idle
- TDM circuit switching: each frame having 10 slots of 1ms
 - link capacity: 1 Mbps → 1000 bits are transferred in one slot → one thousand 1000 bits will be transferred in 10 seconds
- Packet switching
 - the user can send at 1 Mbps
 - no other user is generating packet

Packet switching has better performance



Introduction

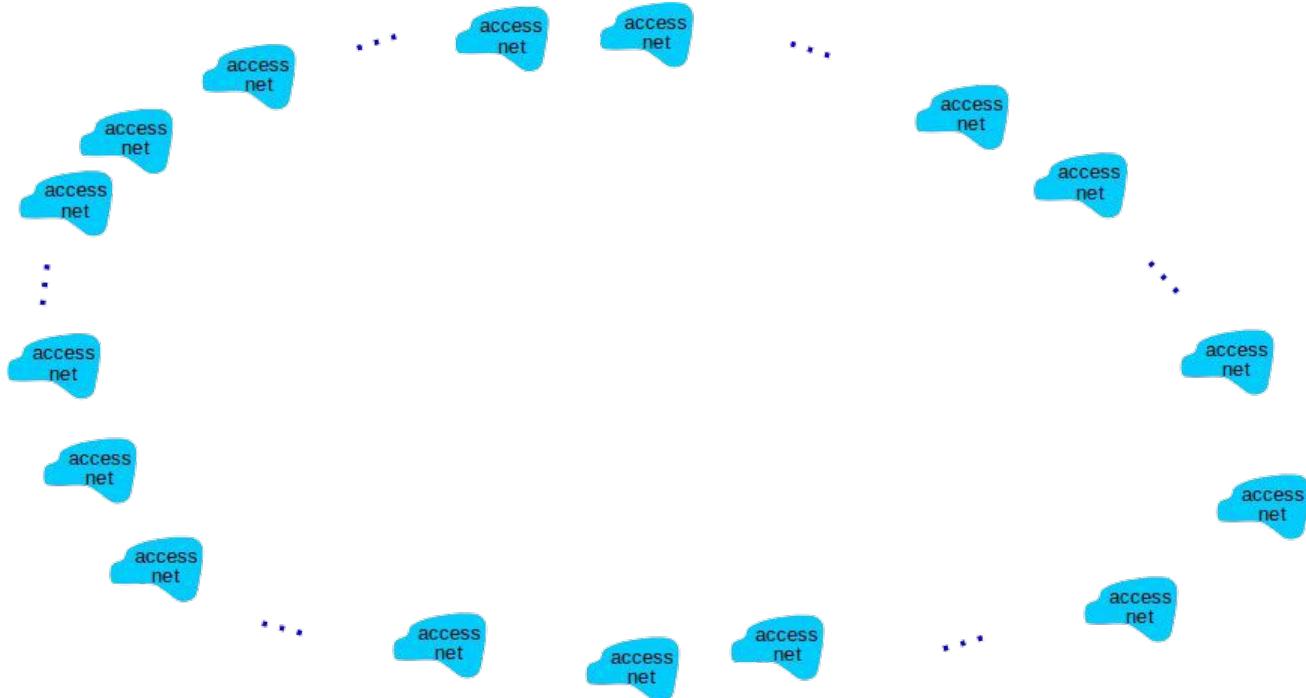
- What's the Internet?
- What's a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching Internet structure
- **Internet structure**
- protocol layers, service models
- performance: loss, delay, throughput
- security
- history

Internet structure: network of networks

- End systems connect to Internet via access networks (ISPs:Internet Service Providers)
 - residential, company and university ISPs
- Access ISPs in turn must be interconnected.
 - to make it possible for any two hosts to send packets to each other
 - This is done by creating a network of networks which is very complex
 - evolution was driven by economics and national policies
- Let's take a stepwise approach to describe current Internet structure

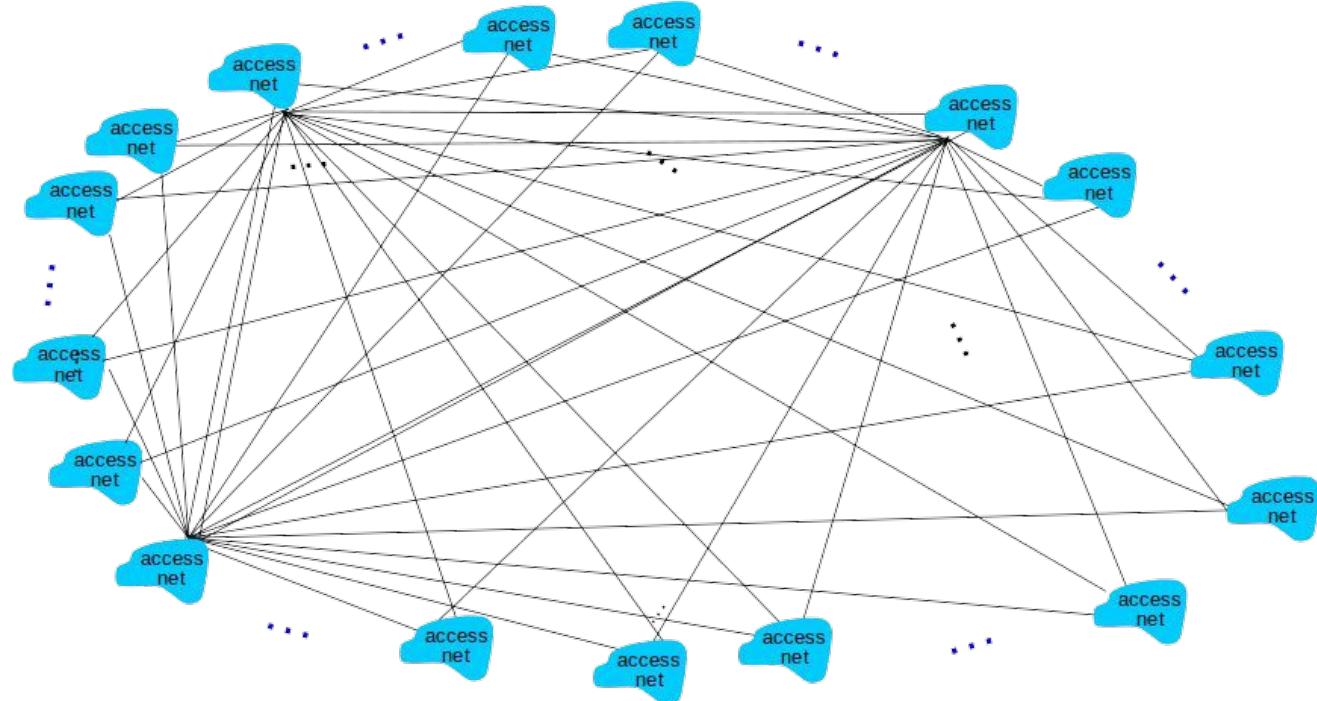
Internet structure: network of networks

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



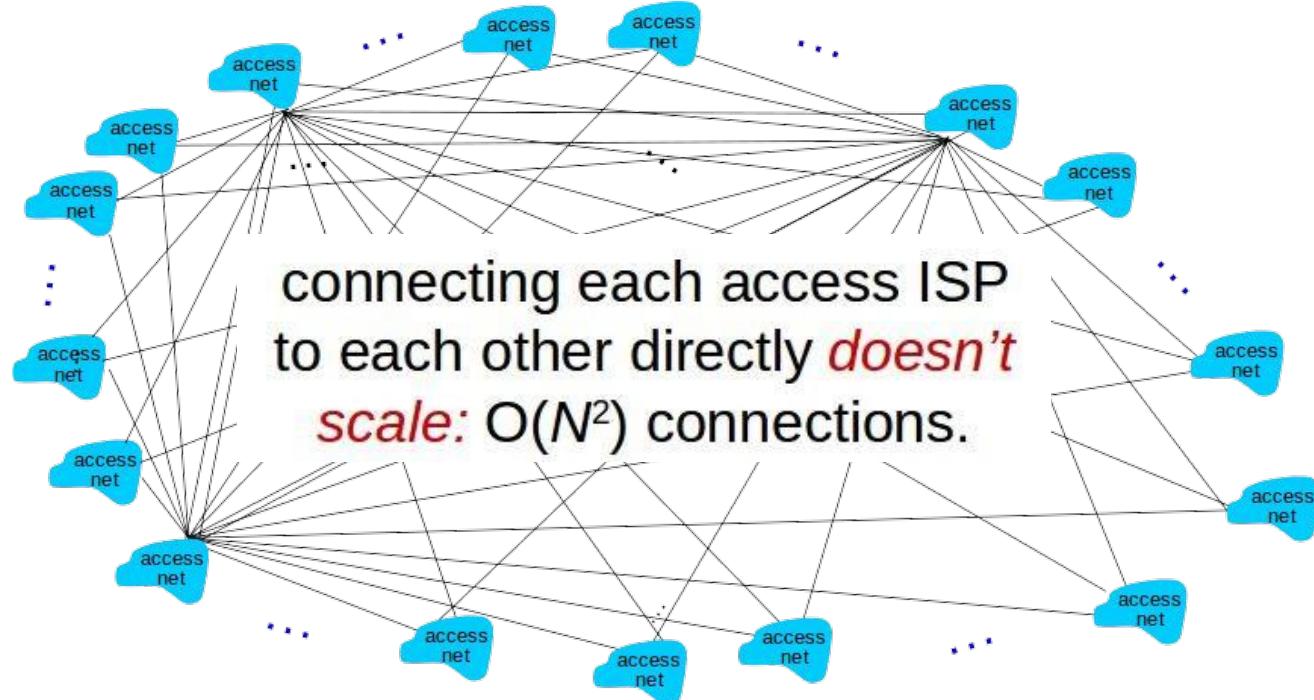
Internet structure: network of networks

Option: connect each access ISP to every other access ISP?



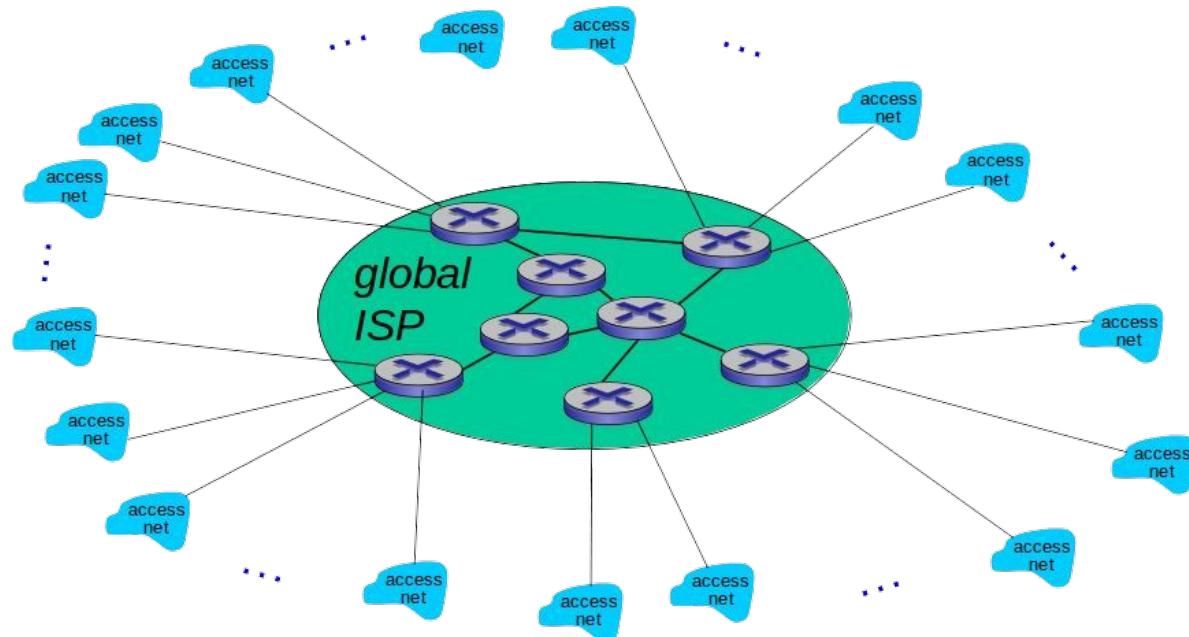
Internet structure: network of networks

Option: connect each access ISP to every other access ISP?



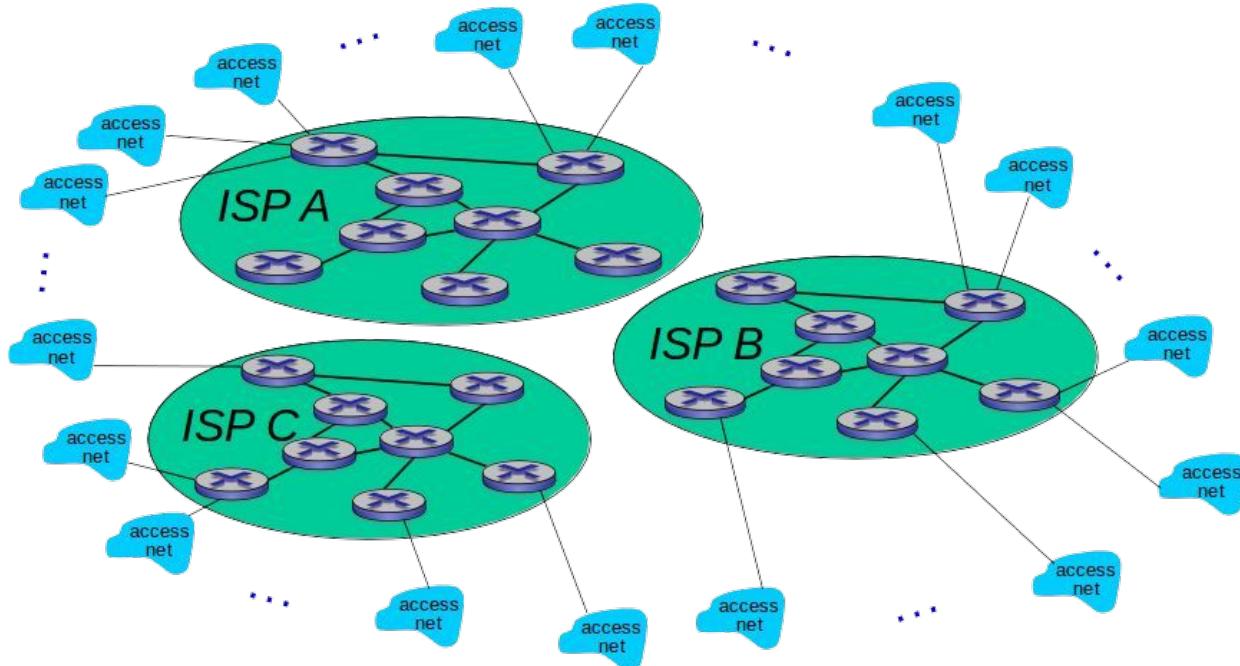
Internet structure: network of networks

- Option: connect each access ISP to one global transit ISP?
- Customer and provider ISPs have economic agreement.

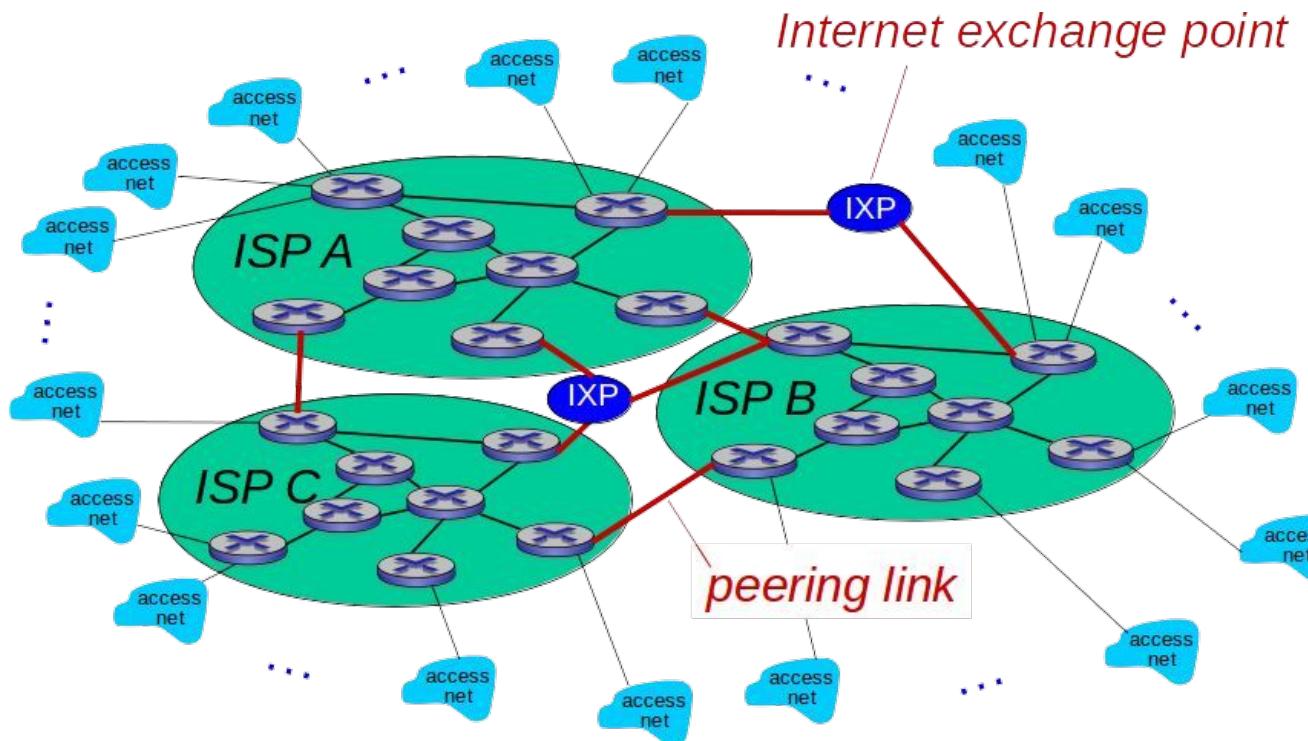


Internet structure: network of networks

But if one global ISP is viable business, there will be competitors which must be interconnected

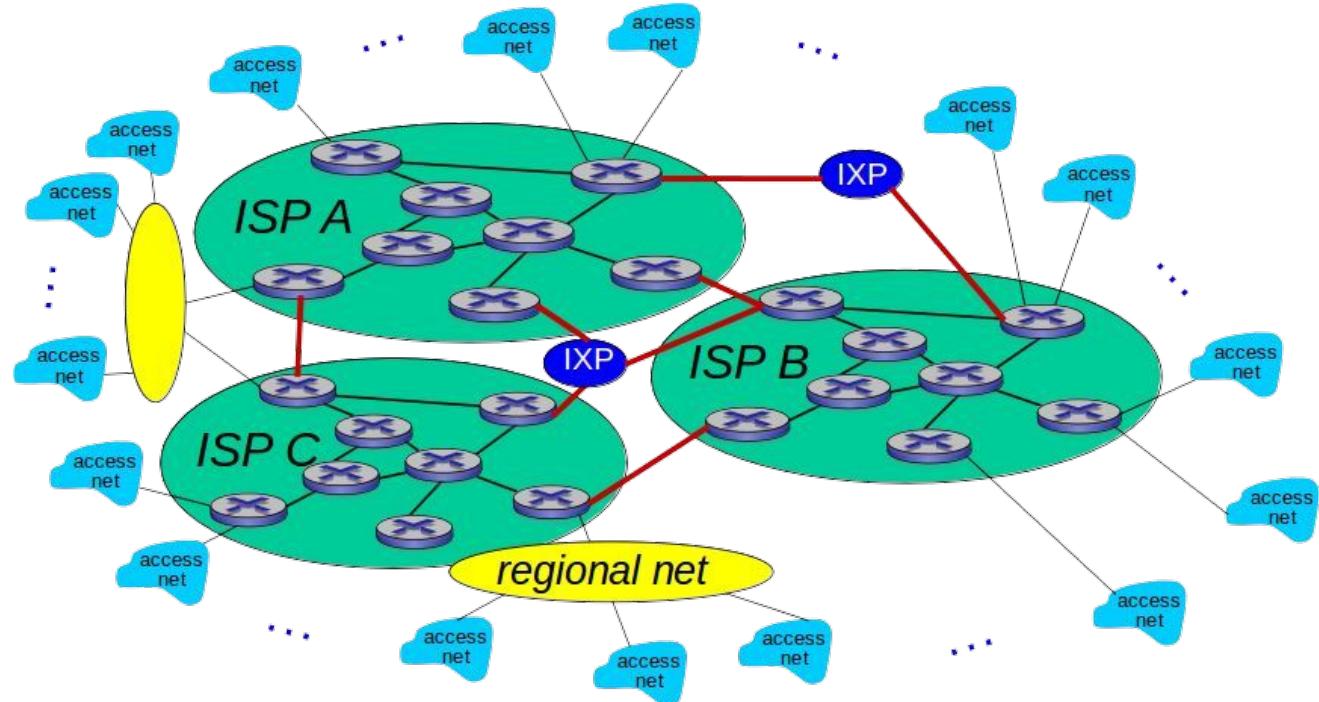


Internet structure: network of networks



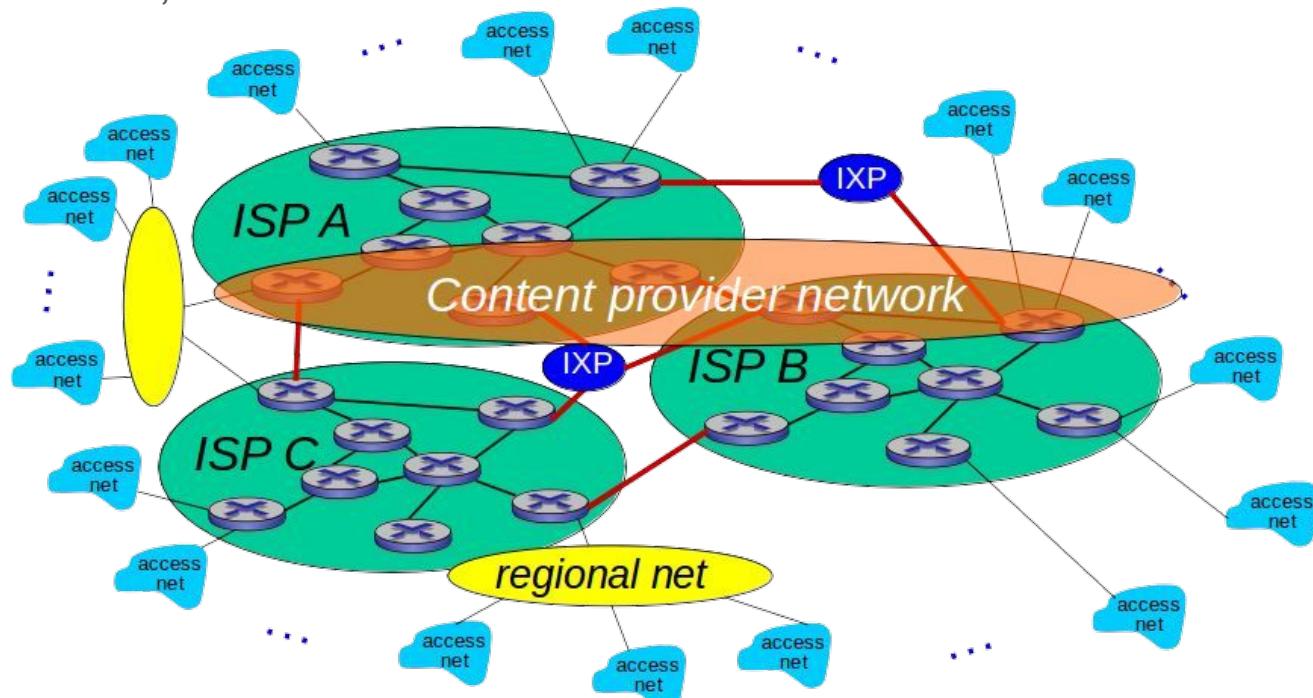
Internet structure: network of networks

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



Internet structure: 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: network of networks

- Hierarchies of ISPs
 - tier-1 ISPs:
 - Larger regional ISP:
 - smaller regional ISP
 - Larger regional ISP connects to tier-1 access ISPs
 - Example:
 - Canada
 - China: access ISPs in each city, provincial ISPs, national ISPs, tier-1 ISPs
- Increasing connectedness/reducing the cost
 - points of presence (PoPs)
 - Multi-home
 - Peer
 - Internet Exchange Point (IXP)
 - Content-provider networks

Internet structure: network of networks

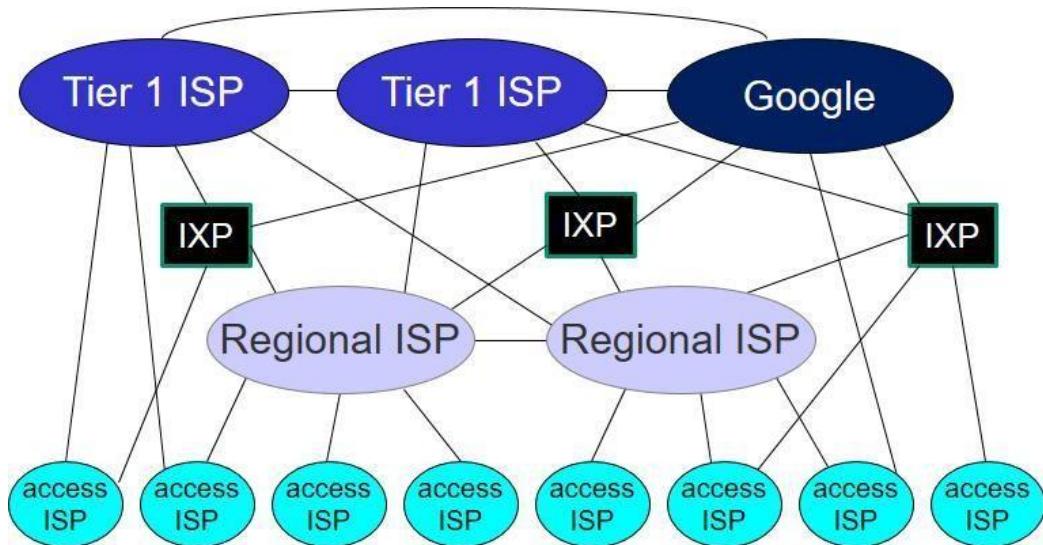
- points of presence (PoPs)
 - a group of one or more routers in the provider's network where customer ISPs can connect into the provider ISP:
 - customer network lease a high-speed link from a third-party telecommunication provider to directly connect one of its routers to a router at the PoP
- Multi-home
 - Any ISP (except for tier-1 ISPs) connecting to two or more provider ISP
 - Example: an access ISP may multi-home with two regional ISPs; multi-home with two regional ISPs and with a tier-1 ISP; A regional ISP may multi-home with multiple tier-1 ISPs
 - The ISP can continue to send and receive packets into the Internet even if one of its providers has a failure

Internet structure: network of networks

- Peer
 - ISPs at the same level of the hierarchy directly connects their network together
 - All the traffic between the two ISPs passes over the direct communication link
 - to reduce the cost of payment to provide ISP
- Internet Exchange Point (IXP)
 - third-party companies: a meeting point where multiple ISPs can peer together
 - a standalone building with its own switches
- Content-provider networks
 - bypass the upper tiers of the Internet by peering with lower-tier ISP
 - connecting directly
 - connecting at IXPs
 - Still are connected to tier-1 ISPs
 - save money by sending less traffic into provider network
 - more control over how the services are delivered to the end user

Internet structure: 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 network (e.g., Google): private network that connects its data centers to Internet, often bypassing tier-1, regional ISPs



Introduction

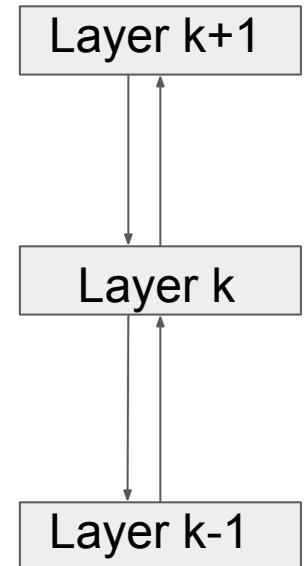
- What's the Internet?
- What's a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching Internet structure
- Internet structure
- **protocol layers, service models**
- performance: loss, delay, throughput
- security
- history

Protocol “layers”

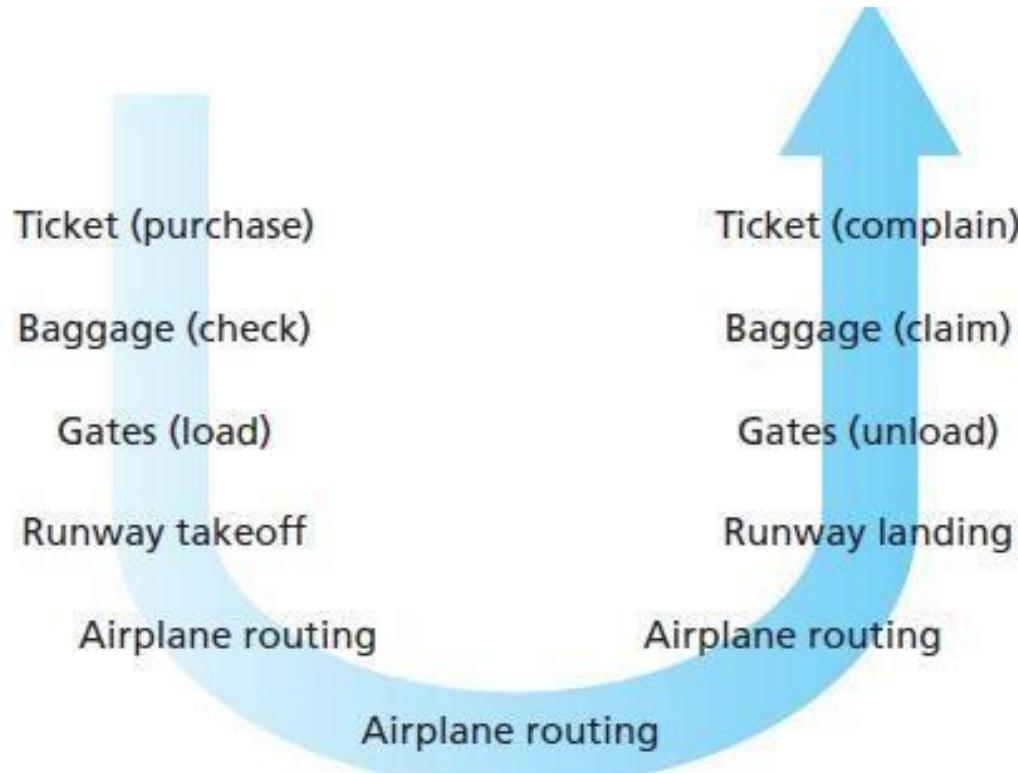
- Networks are complex, with many “pieces”:
 - Hosts
 - Routers
 - links of various media
 - Applications
 - Protocols
 - hardware, software
- Question: is there any hope of organizing structure of network?
- or at least our discussion of networks?

Layering

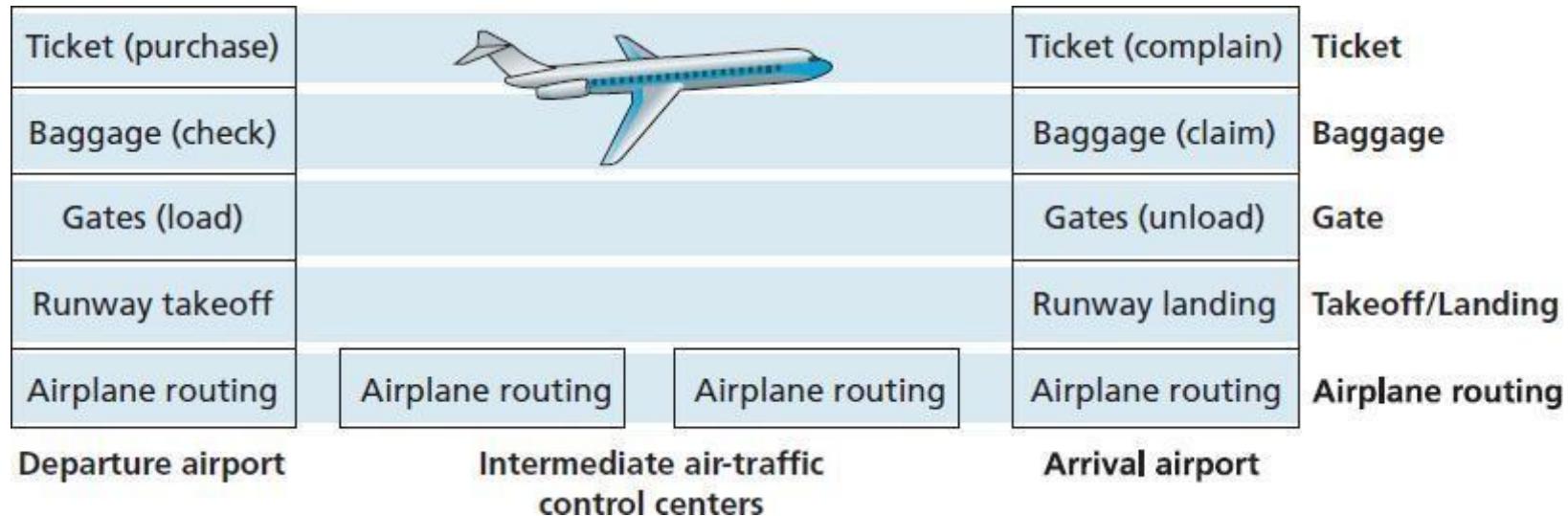
- Layering: organization of a system into a number of separate functional components or layers
 - Each layer provides a well-defined service to the layer above, using the services provided by layers below and its own private processing.
 - Layers are functional components
 - Layers communicate sequentially with the layers above and below.
- Examples:
 - Postal service
 - airplane service



Organization of Air Travel



Layering of airline functionality

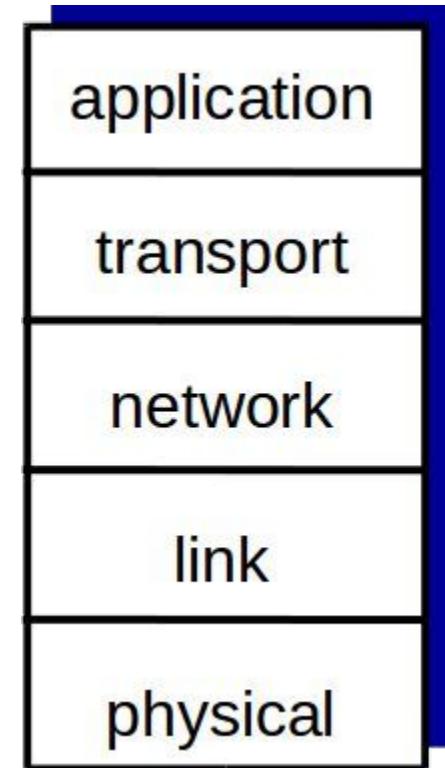


Why Layering?

- dealing with complex systems:
 - explicit structure allows identification, relationship of complex system's pieces
 - discuss a well-defined specific part of a large and complex system
 - modularization eases maintenance, updating of system
 - change of implementation of layer's service transparent to rest of system
 - layering considered harmful?
 - one layer may duplicate lower-layer functionality
 - functionality at one layer may need information that is present only in another layer

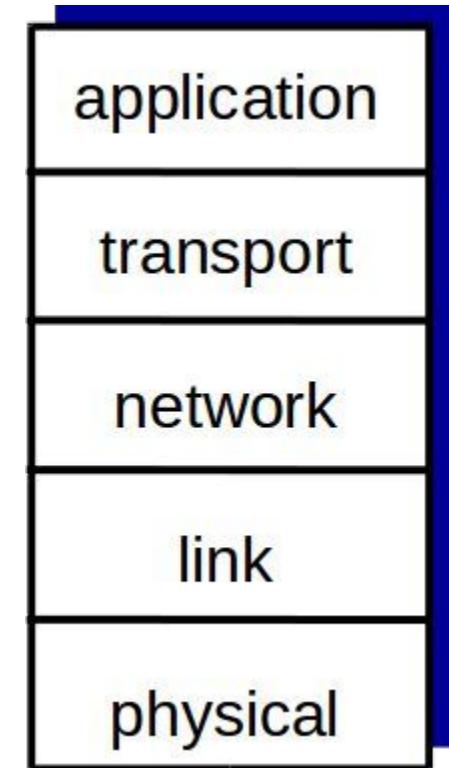
Internet Protocol Stack

- The protocols of the various layers taken together
- **service model:** each layer provides some services to the layer above
 - performing certain action within that layer
 - using the services of the layer below it
 - e.g; layer n may provide reliable delivery of messages
 - by using an unreliable message delivery service of layer n-1
- A protocol in a layer can be implemented in software/hardware/both
- Each layer protocol is distributed



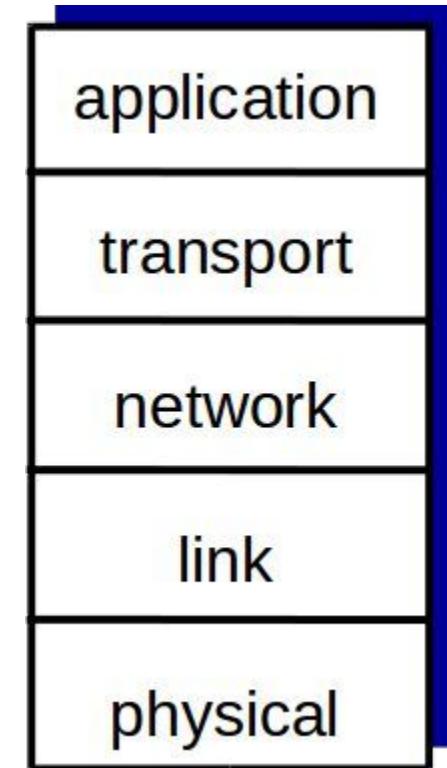
Internet Protocol Stack

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



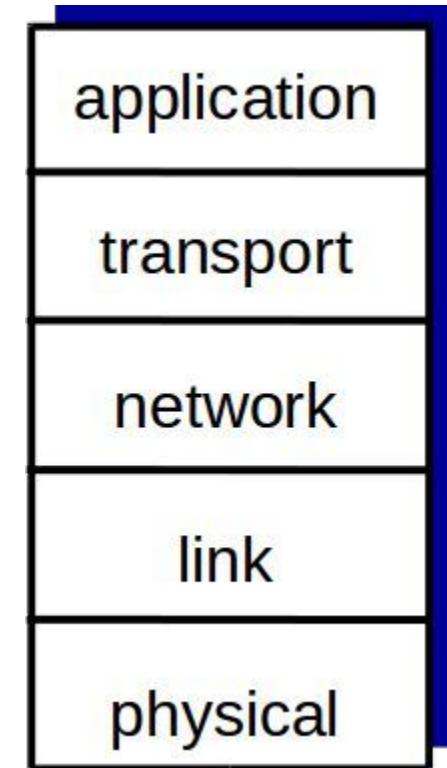
Internet Protocol Stack

- **application:** supporting network applications
 - network application and their application-layer protocol reside
 - HTTP, SMTP, FTP, DNS
 - Application-layer protocol:
 - distributed over multiple end systems
 - Application-layer packets of information: ***message***



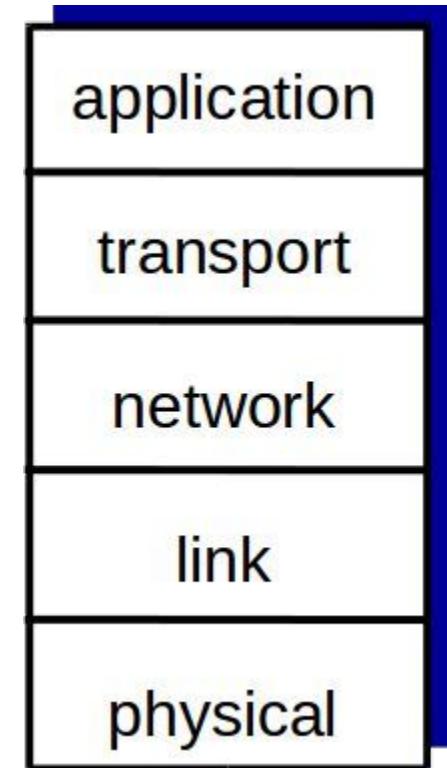
Internet Protocol Stack

- transport application-layer messages between application endpoints
- transport: process-process data transfer
- TCP
 - connection-oriented service
 - guaranteed delivery of application-layer messages
 - control flow (sender/receiver speed matching)
 - congestion control
- UDP
 - no reliability
 - no flow control
 - no congestion control
- transport-layer packet: **segment**



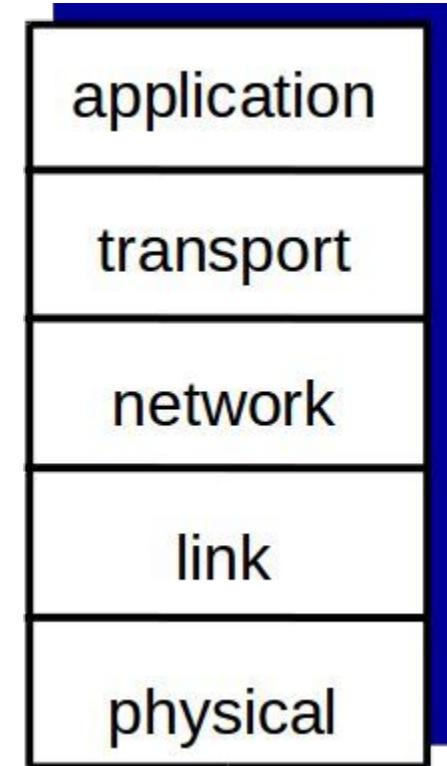
Internet Protocol Stack

- **network:** routing of *datagrams* from source to destination
 - IP
 - defines the fields in the datagram
 - how the end systems and routers act on these fields
 - routing protocols



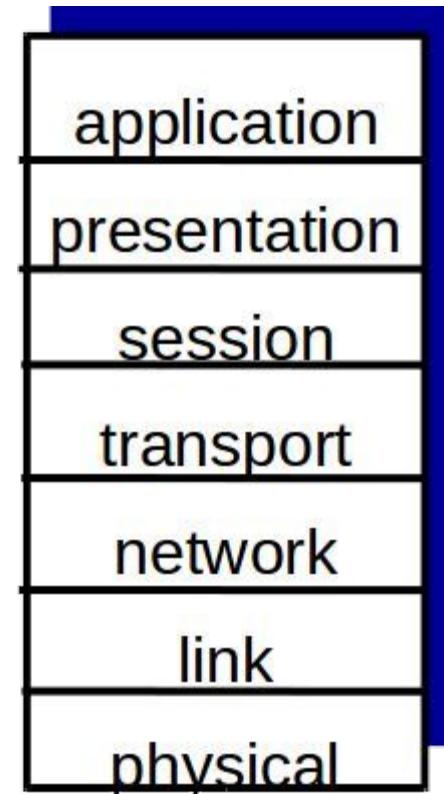
Internet Protocol Stack

- **link:** data transfer between neighboring network elements
 - Ethernet, 802.111 (WiFi), cable access network's DOCSIS
 - Different link-layer protocols along the way
 - A datagram may be handled by different link layer protocol at different links along its route
 - frames: link-layer packets
- **physical:** move bits within the frame from one node to the next
 - link-dependent

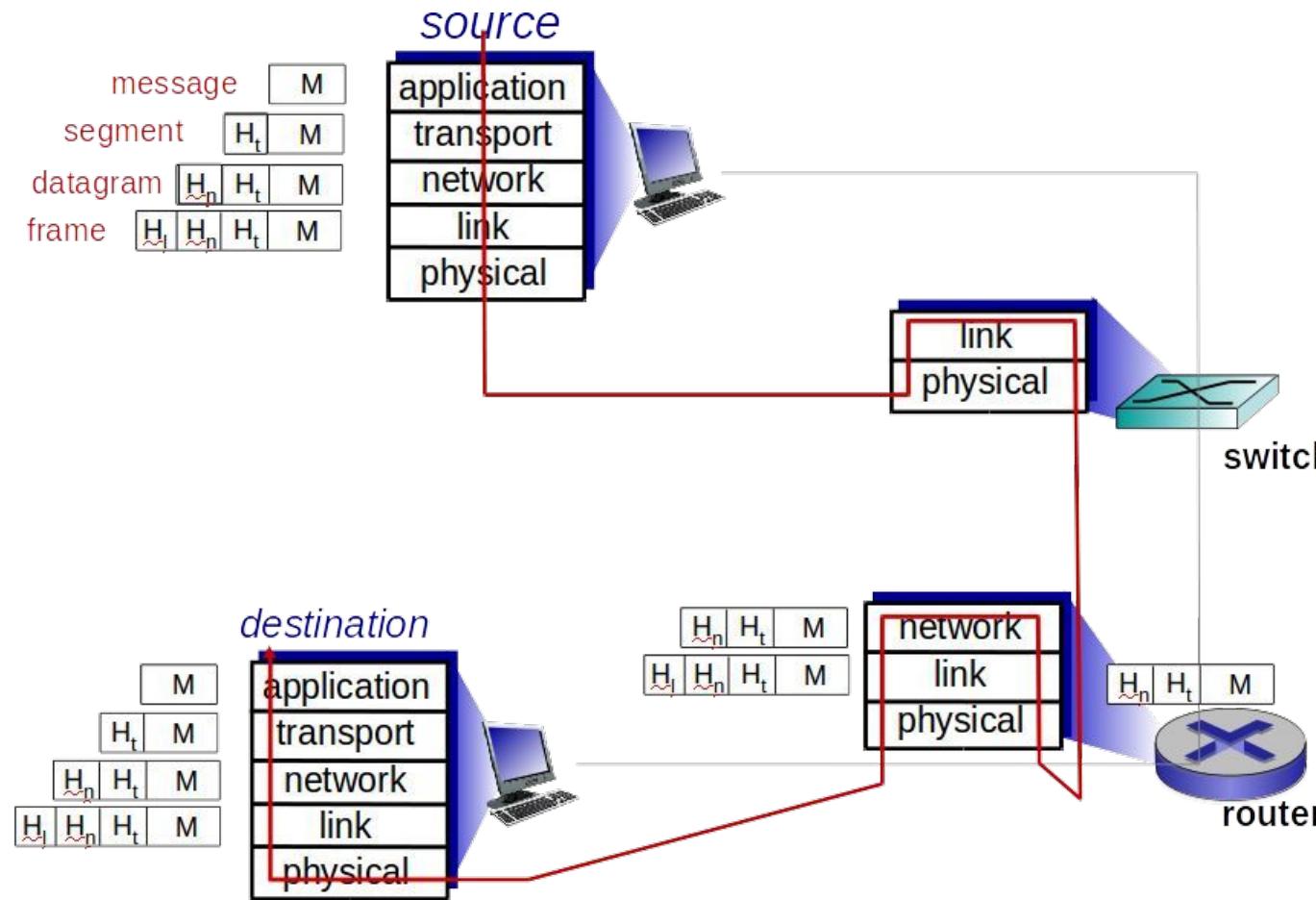


ISO/OSI reference model

- **Open System Interconnection**
 - **presentation:** provide services for applications to interpret meaning of data, e.g., encryption, compression, data description
 - applications do not need to worry about the format of data in different computer systems.
 - Examples: image/jpeg
 - **session:** provides services for synchronization of application
- Internet protocol stack “missing” these layers!
- these services, if needed, must be implemented in application



Encapsulation Decapsulation



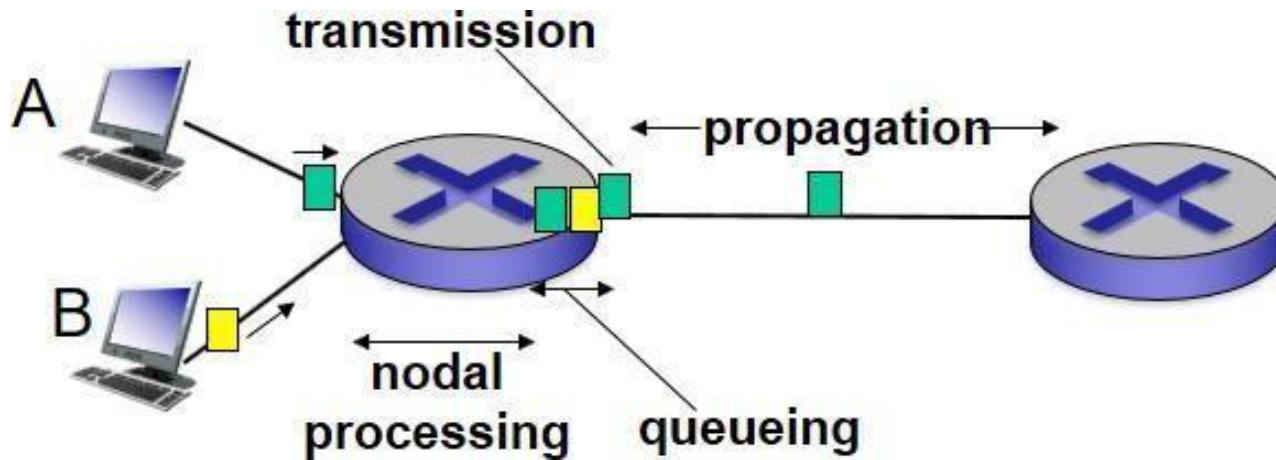
Introduction

- What's the Internet?
- What's a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching Internet structure
- Internet structure
- protocol layers, service models
- **performance: loss, delay, throughput**
- security
- history

Delay, Loss and Throughput in packet switched Networks

- Computer networks move data with delay and lose packets.
- A packet can be transmitted on a link only if there is no other packet currently being transmitted on the link and if there are no other packets preceding it in the queue

Four Sources of Packet Delay



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

The nodal processing delay

d_{proc} : nodal processing

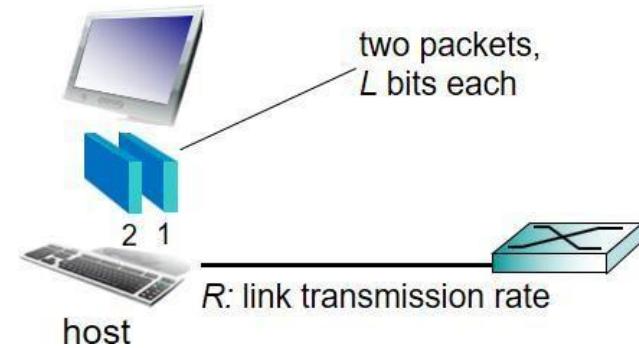
- check bit-level errors in the packet that occurred in transmitting the packet's bits to router
- determine output link
- examine the packet's header
- in high speed routers, on the order of microseconds

Transmission delay

- The amount of time required to push all of the packet's bits into the link
 - L : packet length (bits)
 - R : transmission rate for a link (aka **bandwidth**, aka **capacity**)
 - How fast we can put bits onto the link
 - d_{trans} : **transmission delay**
 - time needed to transmit L bits packet into the link:
- typically on the order of microseconds to milliseconds in practice

Transmission delay

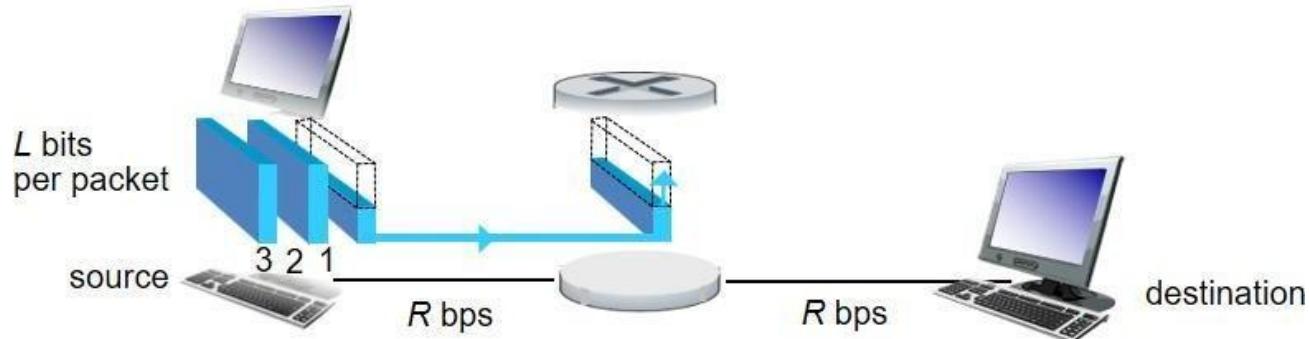
- Example:
 - one hop
 - Host transmits packet into access network at **transmission rate R**
 - $L = 7.5 \text{ Mbits}$
 - $R = 1.5 \text{ Mbps}$
 - one-hop transmission delay = 5 sec



End-to-end transmission delay

- Packet-switched: store-and-forward

- **Question:** What is the end-to-end transmission delay if the source wants to send L bits to destination?
- Sources takes L/R seconds to transmit (push out) L -bit packet into link at R bps
- store and forward: entire packet must arrive at router before it can be transmitted on next link
- End-to-end delay = $2L/R$



End-to-end transmission

delay

- 1 packets over link consisting of N-1 router
- p packet over a link consisting of N-1 router

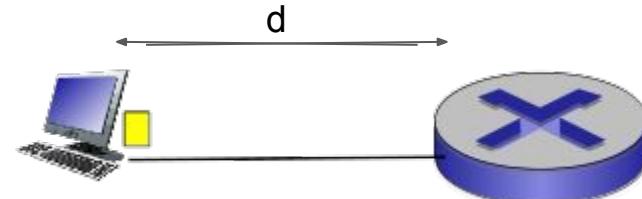
Propagation delay

- The time it takes a single bit to travel over a link at propagation speed s.
- The bits propagates at the propagation speed of the link.
 - Example: A bit takes 5ms to travel 1,000km in an optical fiber with propagation speed 2×10^8 m/s.
 - The propagation speed depends on the physical medium of the link
- on order of milliseconds

$$d_{\text{prop}}: \text{propagation delay} = \frac{d}{s}$$

d: length of physical link

s: propagation speed

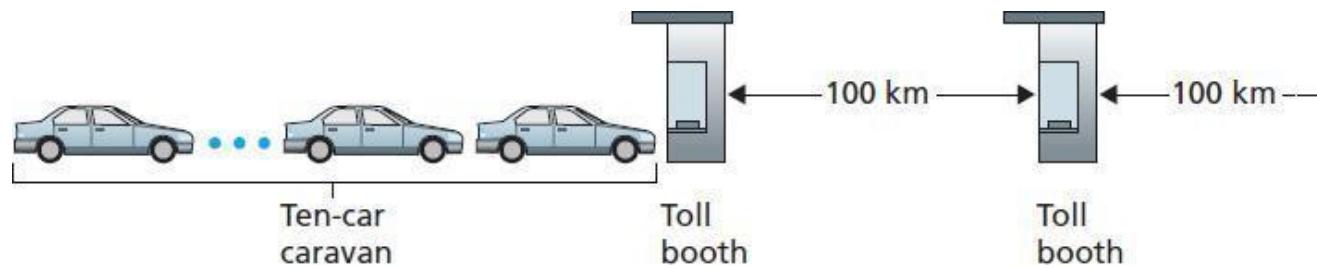


transmission delay vs propagation delay

- transmission delay
 - packet length
 - transmission rate of the link
- propagation delay
 - distance between the two routers
 - medium used for communication

transmission delay vs propagation delay: Caravan Analogy

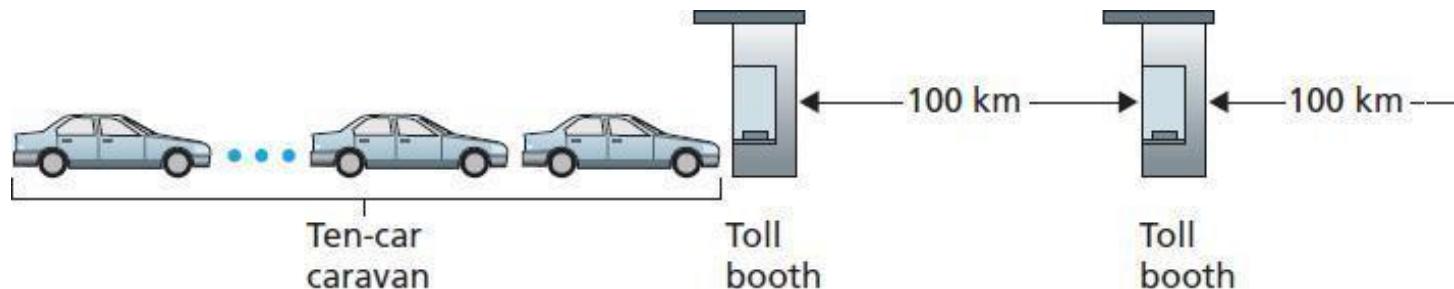
- Network ~ a Highway that has tollbooth every 100 kilometers
- Tollbooth ~ router
- cars propagate at 100 km/h
- toll booth takes 12 sec to service car (bit transmission time)
- car ~ bit; caravan ~ packet
- 10 cars traveling as a caravan, follow each other in fixed order
- **Q: How long until caravan is lined up before 2nd toll booth?**



Caravan Analogy

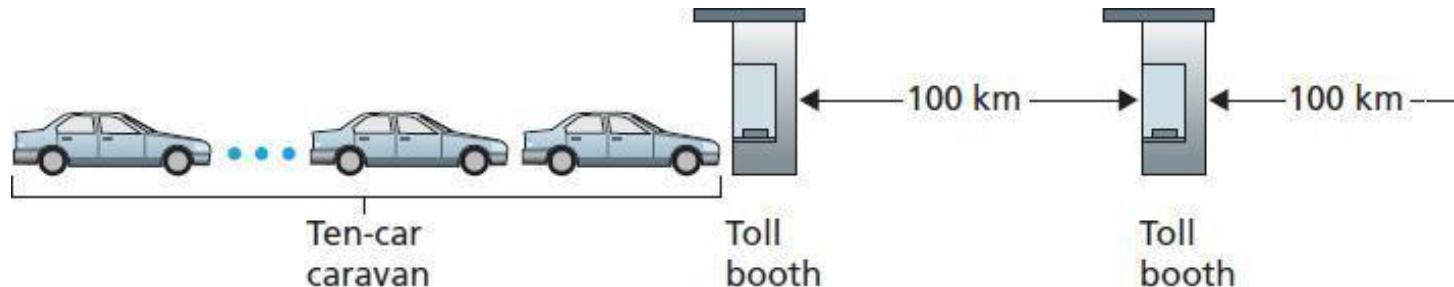
- Q: How long until caravan is lined up before 2nd toll booth?
 - time to “push” entire caravan through toll booth onto highway = $12 \times 10 = 120$ sec
 - time for last car to propagate from 1st to 2nd toll both:
- A: 62 minutes

$$\frac{100\text{km}}{\left(\frac{100\text{km}}{\text{hr}}\right)} = 1\text{hr}$$



Caravan Analogy

- suppose cars now “propagate” at 1000 km/hr and suppose toll booth now takes one minute to service a car
- **Q: Will cars arrive to 2nd booth before all cars serviced at first booth?**
- **A: Yes!** after 7 min, first car arrives at second booth; three cars still at first booth



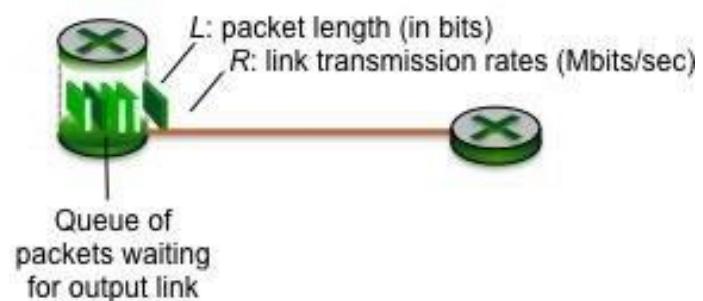
transmission delay vs propagation delay

Animation

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/transmission-vs-propogation-delay/transmission-propagation-delay-ch1/index.html

Computing the one-hop transmission delay

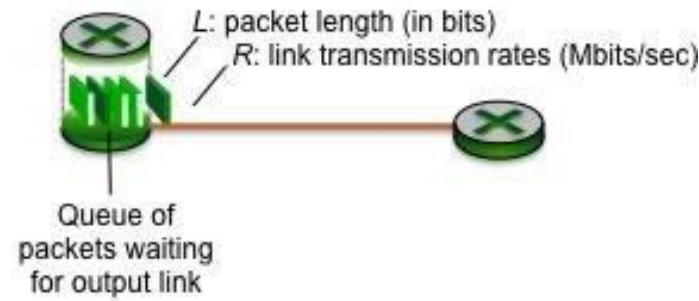
- **Question:** Consider the figure below, in which a single router is transmitting packets, each of length L bits, over a single link with transmission rate R Mbps to another router at the other end of the link.
 - packet length: $L = 16000$ bits
 - the link transmission rate along the link to router on the right is $R = 10$ Mbps.
- What is the transmission delay (the time needed to transmit all of a packet's bits into the link)?



- A. 1600 msec
- B. 1.6 sec
- C. 1600 msec
- D. 1.6 msec

Computing the one-hop transmission delay

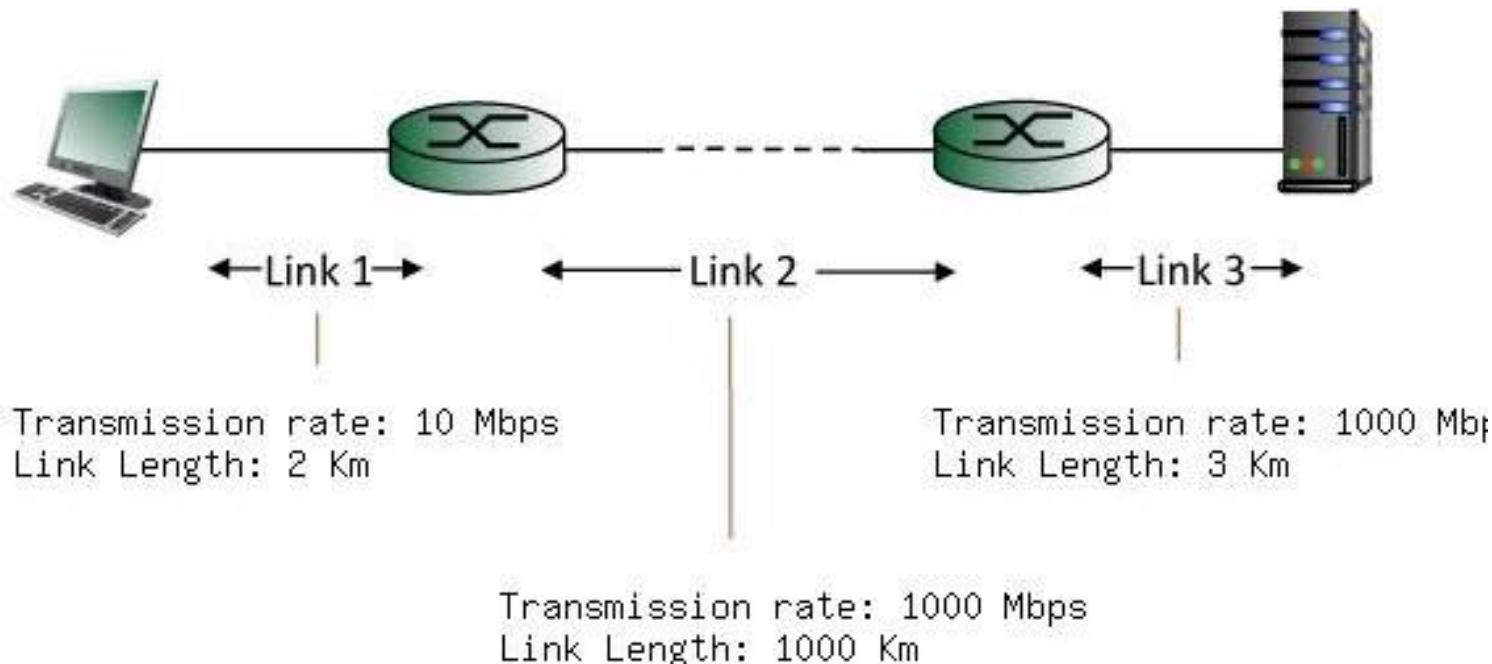
- **Question:** Consider the figure below, in which a single router is transmitting packets, each of length L bits, over a single link with transmission rate R Mbps to another router at the other end of the link.
 - packet length: $L = 16000$ bits
 - the link transmission rate along the link to router on the right is $R = 10$ Mbps.
- what is the maximum number of packets per second that can be transmitted by the link?



- A. 2000
- B. 625
- C. 1600

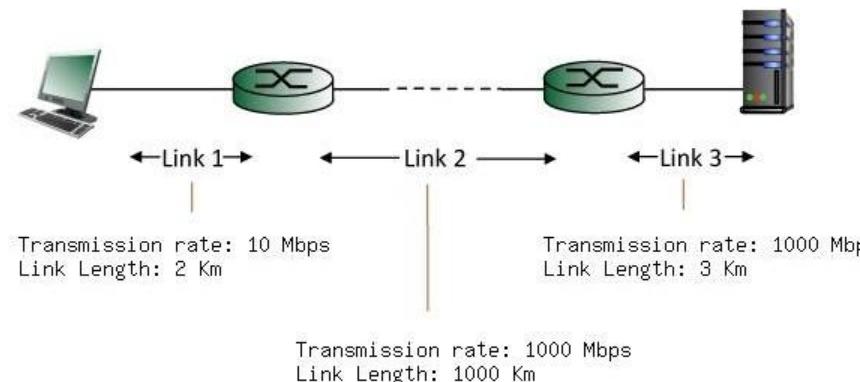
end-to-end delay (transmission and propagation delay)

- **Question:** Find the end-to-end delay



end-to-end delay (transmission and propagation delay)

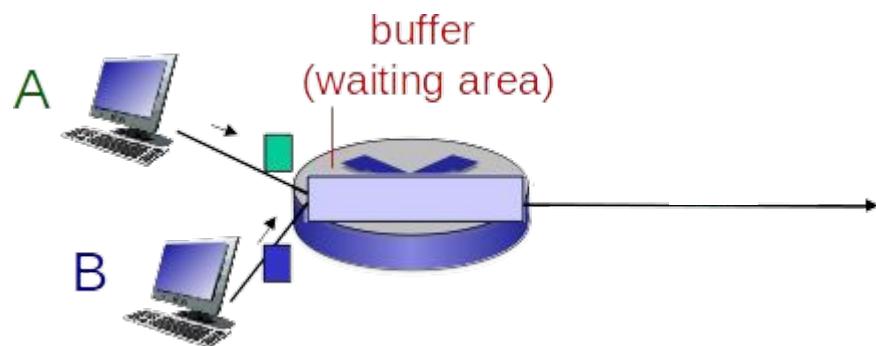
- **Question:** Find the end-to-end delay
 - transmission delays and propagation delays on each of the three links from when the left host begins transmitting the first bit of a packet to the time when the last bit of that packet is received at the server at the right.
 - propagation delay on each link is 3×10^{-8} m/sec.
 - transmission rates are in Mbps and the link distances are in Km.
 - packet length: 4000 bits. Give your answer in milliseconds.



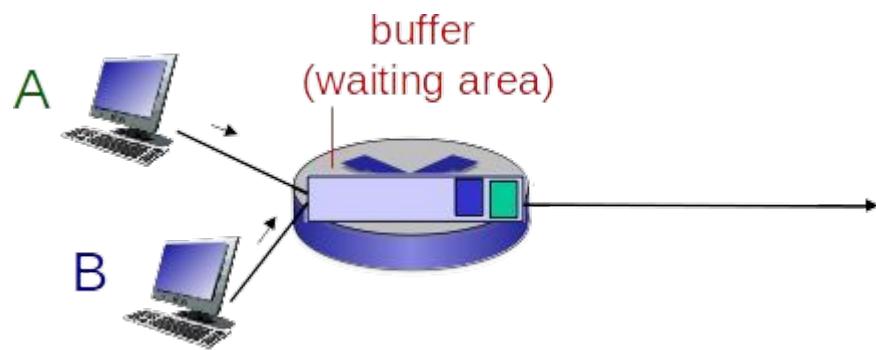
end-to-end delay (transmission and propagation delay)

- **Answer:**
 - Link 1 transmission delay = $L/R = 4000 \text{ bits} / 10 \text{ Mbps} = 0.400000 \text{ msec.}$
 - Link 1 propagation delay = $d/s = 2 \text{ Km} / 3*10^{**8} \text{ m/sec} = 0.006667 \text{ msec.}$
 - Link 2 transmission delay = $L/R = 4000 \text{ bits} / 1000 \text{ Mbps} = 0.004000 \text{ msec.}$
 - Link 2 propagation delay = $d/s = 1000 \text{ Km} / 3*10^{**8} \text{ m/sec} = 3.333333 \text{ msec.}$
 - Link 3 transmission delay = $L/R = 4000 \text{ bits} / 1000 \text{ Mbps} = 0.004000 \text{ msec.}$
 - Link 3 propagation delay = $d/s = 3 \text{ Km} / 3*10^{**8} \text{ m/sec} = 0.010000 \text{ msec.}$
 - Thus, the total end-to-end delay is the sum of these six delays: 3.758000 msec.

Queueing delay

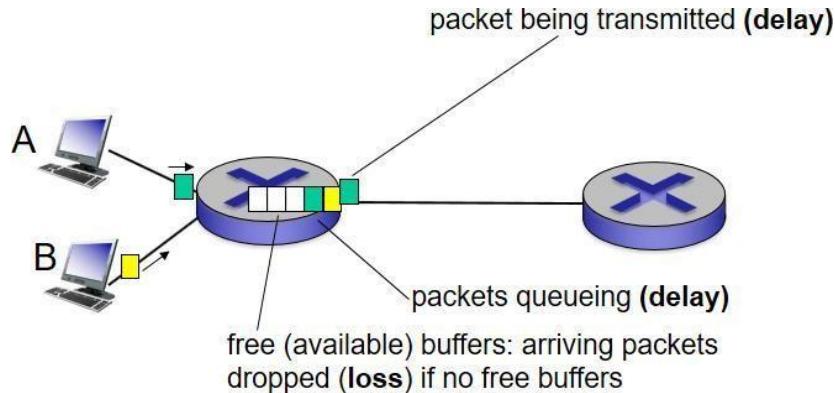


Queueing delay



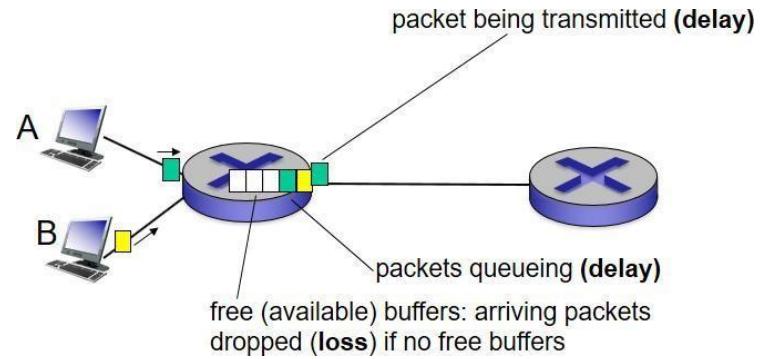
Queueing delay

- packets queue in router buffers
- If packet arrival rate to link (temporarily) exceeds output link capacity
 - packets queue, wait for turn



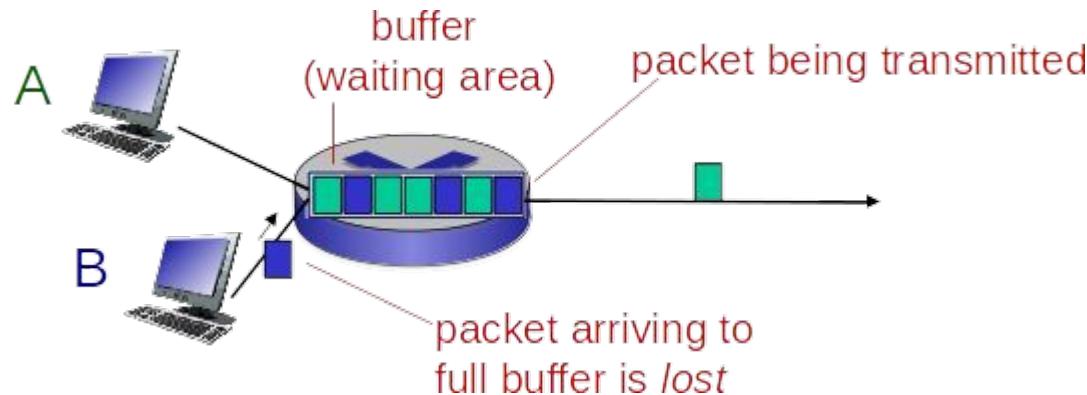
Queueing delay

- d_{queue} : queueing delay
- time waiting at output link for transmission
 - depends on congestion level of router (non-deterministic)
 - delay is zero:
 - if the queue is empty and no other packet is being transmitted, delay is zero
 - delay is high:
 - if the traffic is heavy and many other packets are also waiting to be transmitted
- in order of microseconds to milliseconds in practice



Packet Loss

- queue (buffer) preceding link in buffer has finite capacity
- packet arriving to full queue dropped (lost)
- lost packet may be retransmitted by previous node, by source end system, or not at all



Queueing delay

Animation

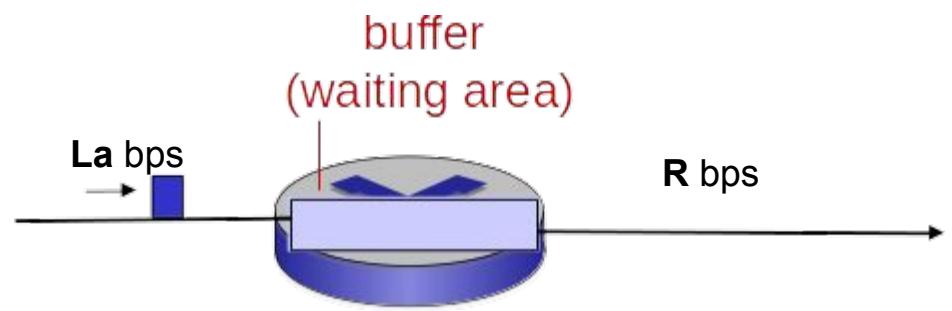
https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/queuing-loss-applet/index.html

Queueing Delay

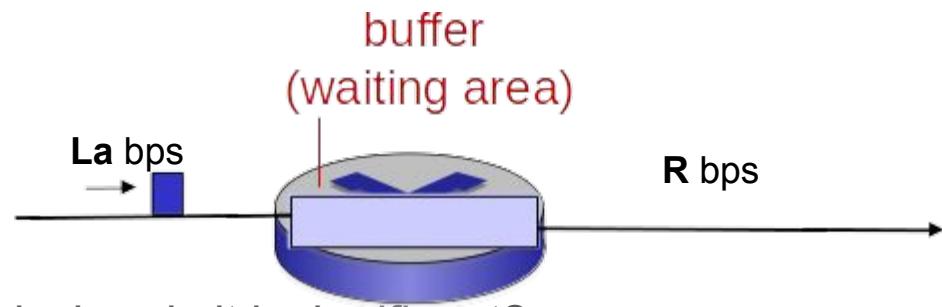
- **Q: Is queueing delay the same for all packets?**
 - No, it can vary from packet to packet
 - If 10 packets arrive at an empty queue at the same time no delay experienced by the first packet, while the last one suffers a huge delay
- How to specify queuing delay?
 - It is characterized using statistical measures such as:
 - average queueing delay
 - probability of queueing delay > value

Queueing delay

- Example1:
 - One packet arrives every L/R seconds
 - What is the queueing delay of the n -th packet?
 - Queueing delay: 0
- Example2:
 - Traffic comes periodically but in burst:
 - N packets arrive simultaneously every $(L/R)N$ seconds
 - L : size of each packet in bits
 - What is the queueing delay of the n th packet?
 - What is the average queueing delay?



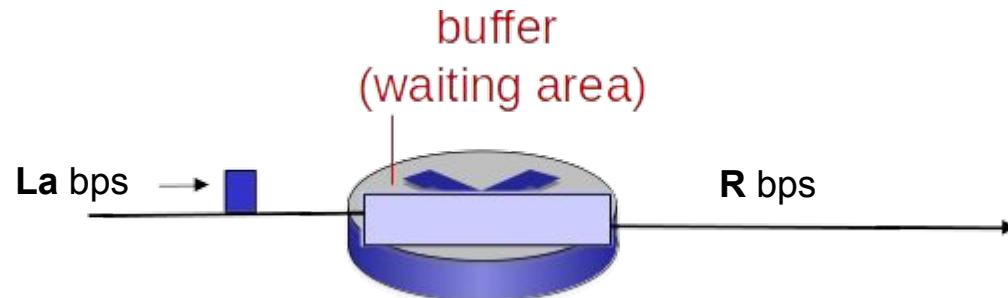
Queueing delay



- When is the queueing delay large and when is it insignificant?
 - R : output link bandwidth (bps): The transmission rate at which packets leave the buffer
 - L : packet length (bits)
 - a : average packet arrival rate
 - La : average rate at which bits arrive at the queue (bits/s)
 - assumption: queue can hold an infinite number of bits
 - La/R : traffic intensity
 - $La/R > 1$
 - more work arriving than can be serviced, the queue will tend to increase without bound, avg delay infinite
 - $La/R \leq 1$
 - queueing delay depend on the nature of arriving traffic impact the delay
 - Periodically
 - In burst

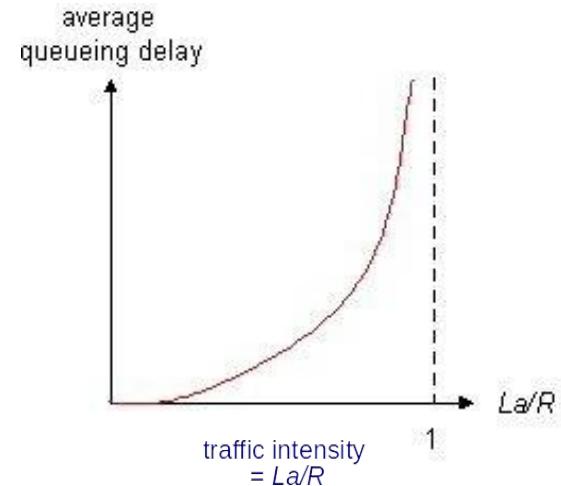
Queueing delay

- When is the queueing delay large and when is it insignificant?
 - In reality, arrival process to a queue is random
 - Arrivals do not follow any patterns
 - Packets are spaced apart by random amounts of time
 - L_a/R not sufficient to fully characterize the queueing delay statistics
 - Gives an intuitive understanding of the extent of the queueing delay



Queueing delay

- assumption: queue can hold an infinite number of bits
- $La/R \sim 0$: avg. queueing delay small
- $La/R \sim 1$: avg. queueing delay large
- $La/R > 1$: more work arriving than can be serviced, the queue will tend to increase without bound, avg delay infinite

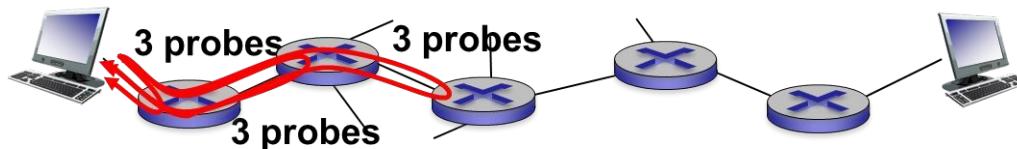


$$\frac{La}{R} \sim 0$$

$$\frac{La}{R} - > 1$$

“Real” Internet Delays and Routes

- what do “real” Internet delay & loss look like?
- traceroute program: provides delay measurement from source to routers along end-end Internet path towards destination.
- For all n:
 - sends three packets that will reach router n on path towards destination
 - router n will return packets to sender
 - sender times interval between transmission and reply.



“Real” Internet Delays and Routes

Traceroute: from `gaia.cs.umass.edu` to
www.eurecom.fr

3 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

trans-oceanic link



* means no response (probe lost, router not replying)

Throughput

- **throughput:** rate (bits/time unit) at which bits transferred between sender/receiver
 - **instantaneous throughput:** rate at given point in time
 - rate at which host B is receiving file at any instant of time
 - **average:** rate over longer period of time
 - file size: F bits
 - transfer time: T seconds
 - average throughput: F/T

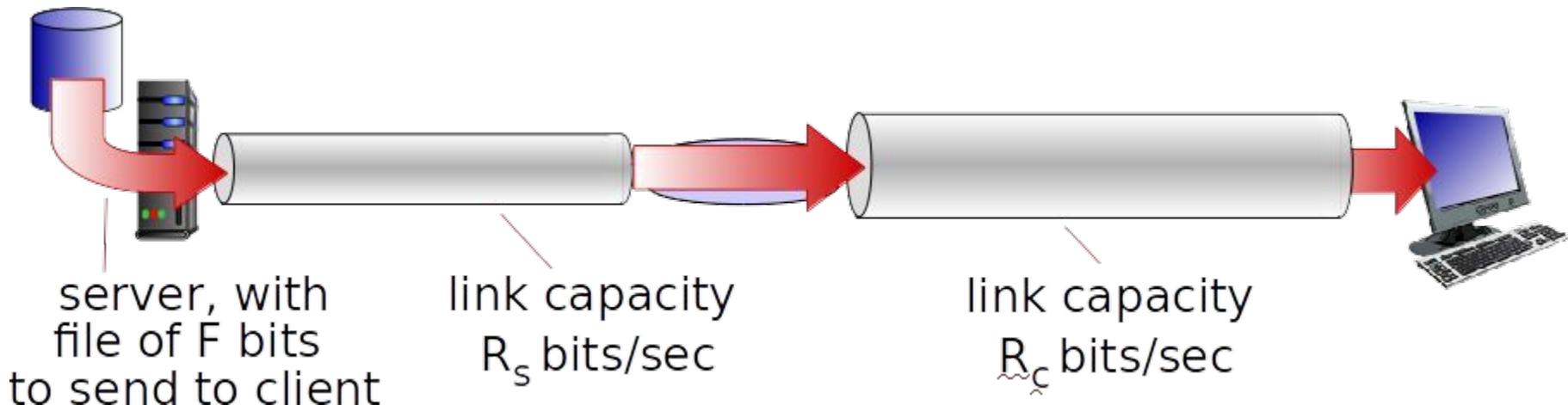


A

B

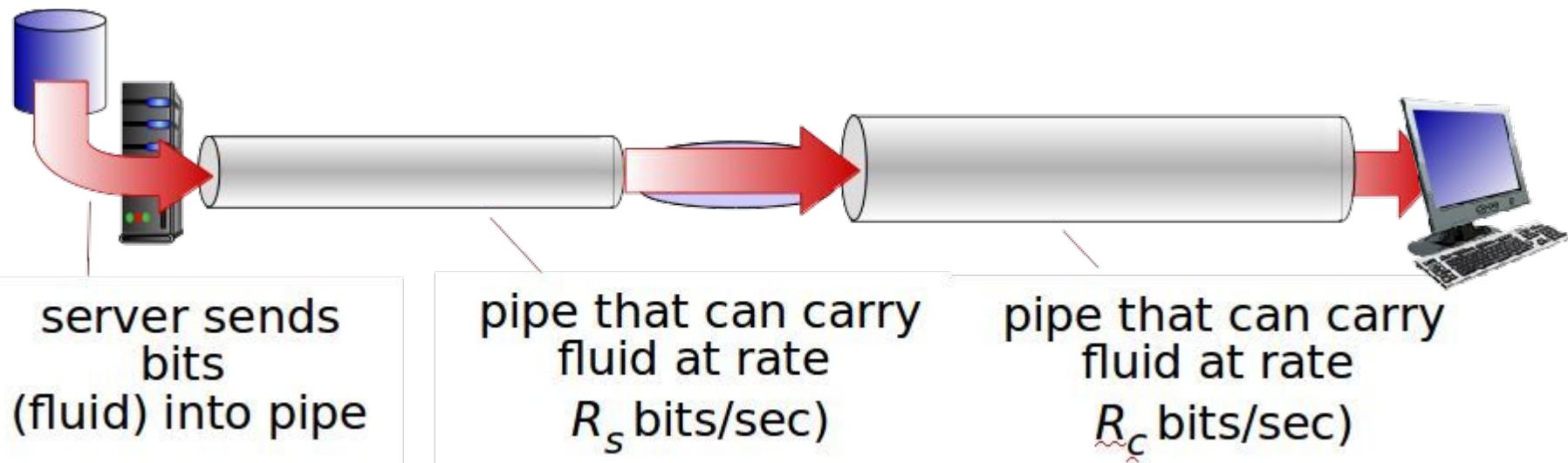
Throughput

- What is the average end-to-end throughput for a file transfer from the server to the client?
 - R_s : the rate of the link between the server and the router
 - R_c : the rate of the link between the router and the client
 - only bits being sent are those from the server to the client



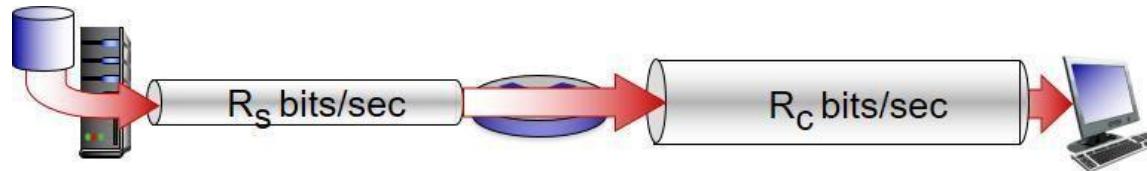
Throughput

Analogy: bits as fluid; communication links as pipes

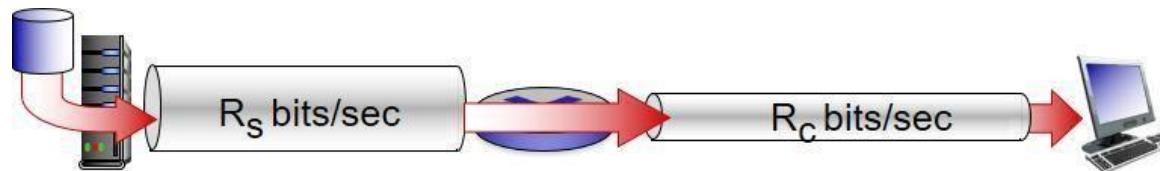


Throughput

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

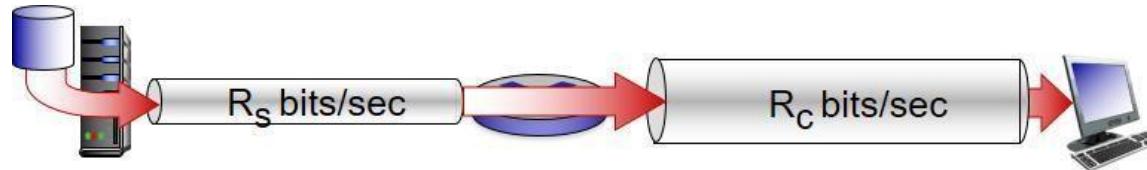


- $R_s > R_c$ What is average end-end throughput?

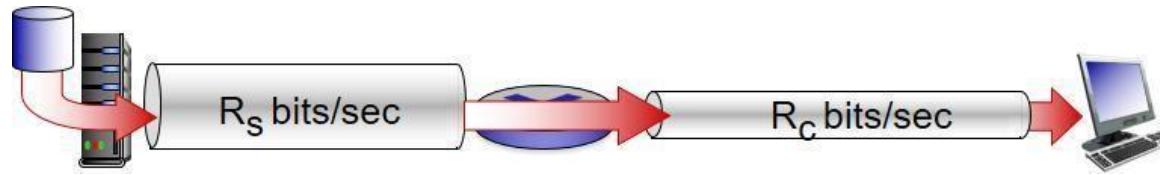


Throughput

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



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



- **throughput: $\min\{R_s, R_c\}$**
- transmission rate of **bottleneck** link
 - link on end-end path that constrains end-end throughput

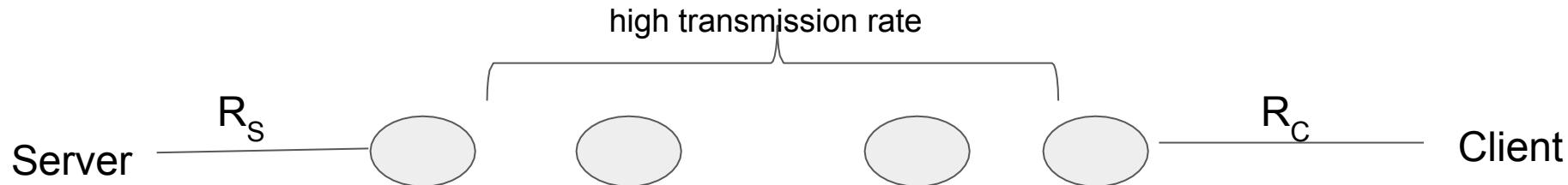
Throughput

- What is the throughput in the following scenario?
 - throughput is the minimum of transmission rate between client and server
 - throughput is the minimum of $\{R_1, R_2, R_3, \dots, R_N\}$
 - throughput is the transmission rate of **bottleneck link**



Throughput

- What is the throughput in the following scenario?
 - all the links in the core of the communication network have very high transmission rates
 - only bits being sent are from the server to the client
 - throughput is the minimum of $\{R_s, R_c\}$
 - constraining factor is the access network

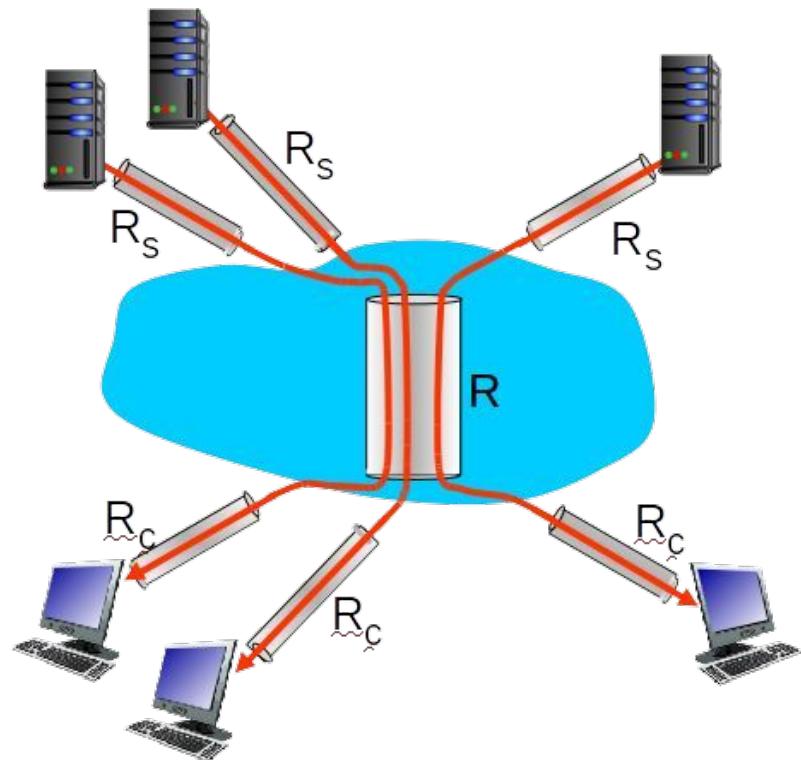


Throughput

- What is per-connection end-end throughput:
 - If R is much larger than R_s and R_c , throughput is $\min(R_s, R_c)$
 - in practice: R_c or R_s is often bottleneck
 - if R is the same order of R_s and R_c throughput is $\min(R_c, R_s, R/10)$

throughput depends:

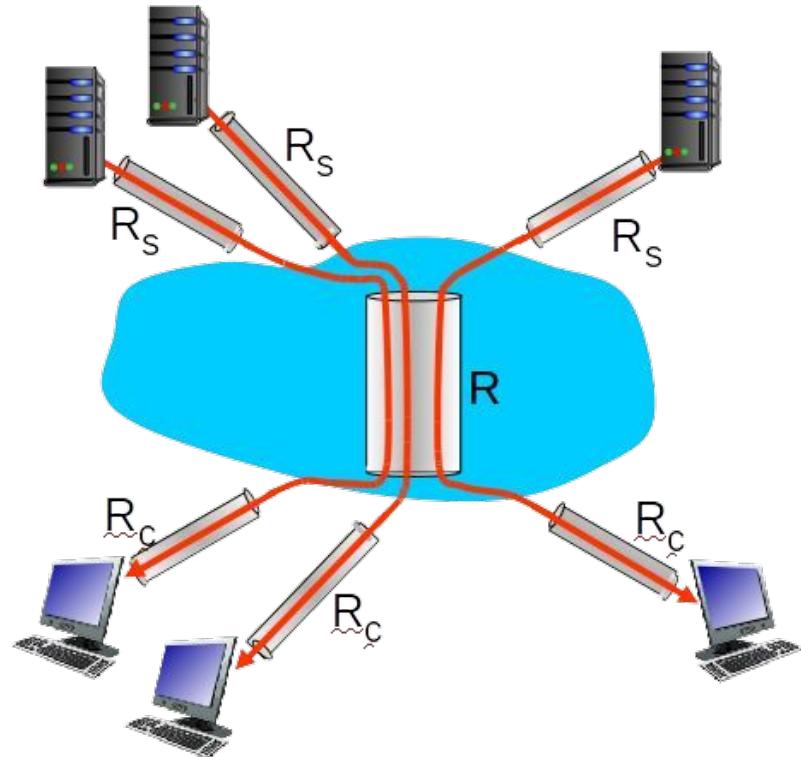
- transmission rate of the links along the path
- intervening traffic



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

Throughput

- throughput depends on the transmission rate of the links along the path
 - No intervening traffic
 - throughput is the minimum transmission rate along the path
 - intervening traffic

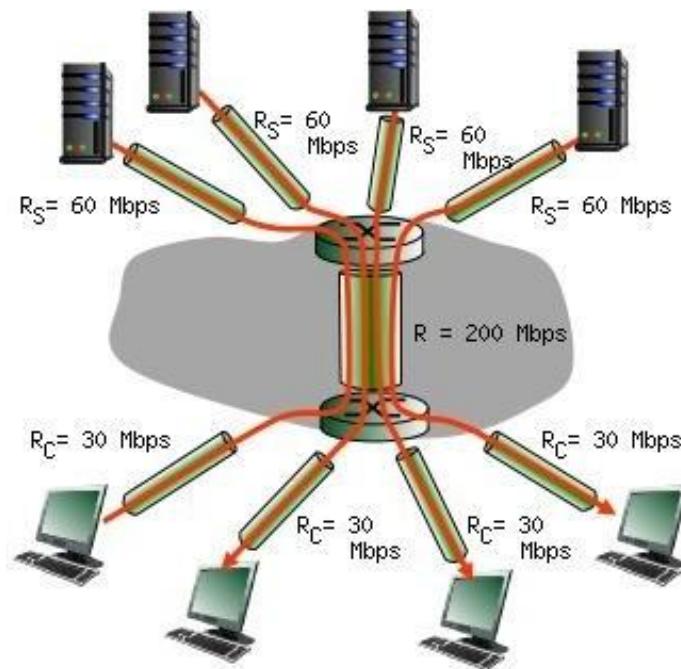


10 connections (fairly) share
backbone bottleneck link R bits/sec

Question: End-to-End Throughput and Bottleneck Links

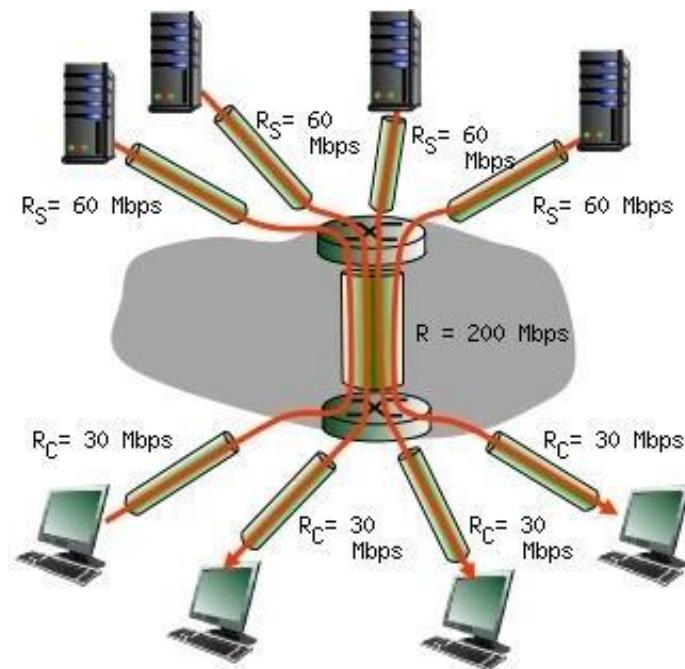
What is the maximum achievable end-end throughput (in Mbps) for each of four client-to-server pairs, assuming that the middle link is fair-shared (i.e., divides its transmission rate equally among the four pairs)?

- A. 30
- B. 60
- C. 200
- D. 50



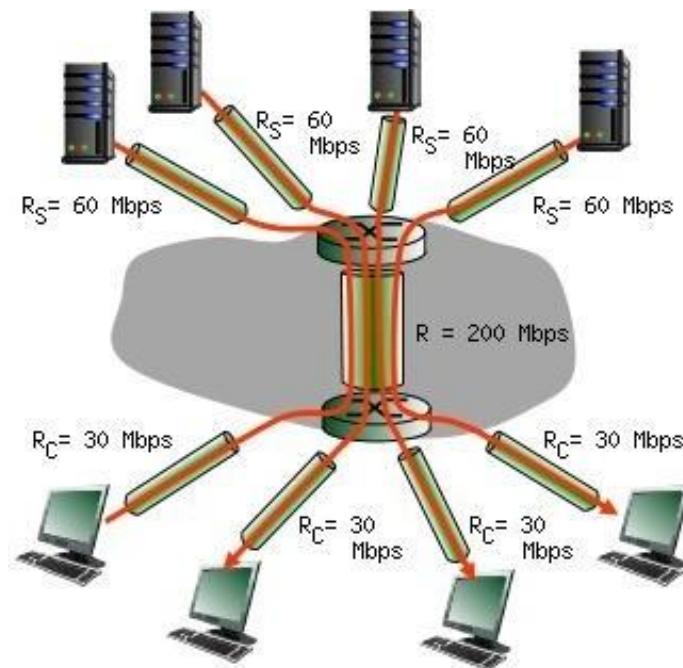
Question: End-to-End Throughput and Bottleneck Links

Which link is the bottleneck link for each session?



Question: End-to-End Throughput and Bottleneck Links

Assuming that the senders are sending at the maximum rate possible, what are the link utilizations for the sender links (R_S), client links (R_C), and the middle link (R)?



Introduction

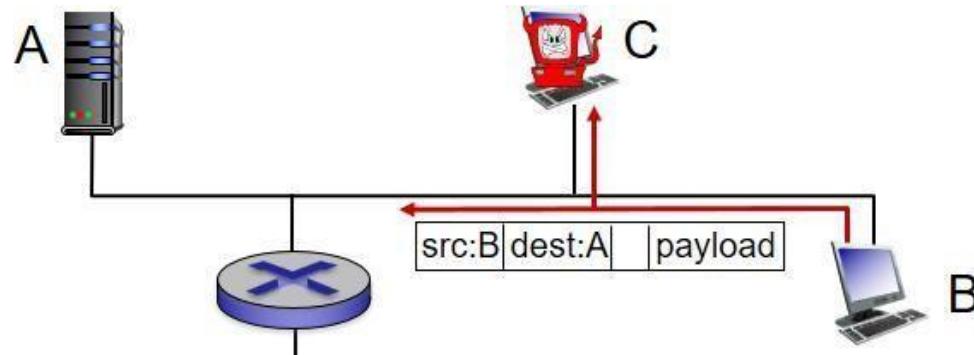
- What's the Internet?
- What's a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching Internet structure
- Internet structure
- protocol layers, service models
- performance: loss, delay, throughput
- **security**
- history

Network security

- Internet not originally designed with (much) security in mind
 - original vision: “a group of mutually trusting users attached to a transparent network”
 - Internet protocol designers playing “catch-up”
 - security considerations in all layers!
- field of network security:
 - how bad guys can attack computer networks
 - how we can defend networks against attacks
 - how to design architectures that are immune to attacks

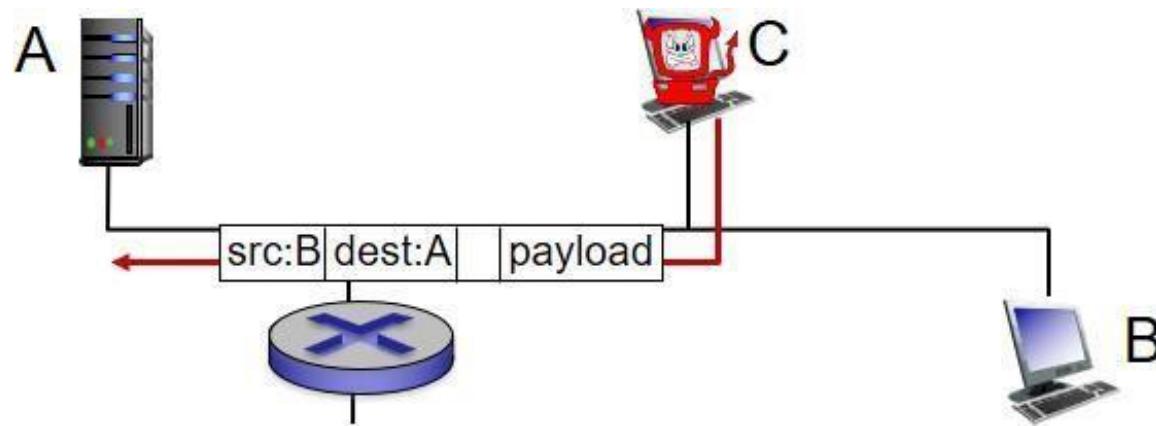
Bad guys can sniff packets

- packet “sniffing”:
 - broadcast media (shared Ethernet, wireless)
 - promiscuous network interface reads/records all packets (e.g., including passwords!) passing by
- wireshark software is a (free) packet-sniffer



Bad guys can use fake addresses

- **spoofing**: when a hacker impersonates another device or user on a network in order to steal data, spread malware, or bypass access controls.
- IP spoofing: send packet with false source address
- **email** spoofing: creation of email addresses with a forged sender address
- Solution: end-point authentication



Bad guys: attack server, network infrastructure

Distributed Denial of Service (DoS):



Bad guys: attack server, network infrastructure

Distributed Denial of Service (DoS):

1. select target



Bad guys: attack server, network infrastructure

Distributed Denial of Service (DoS):

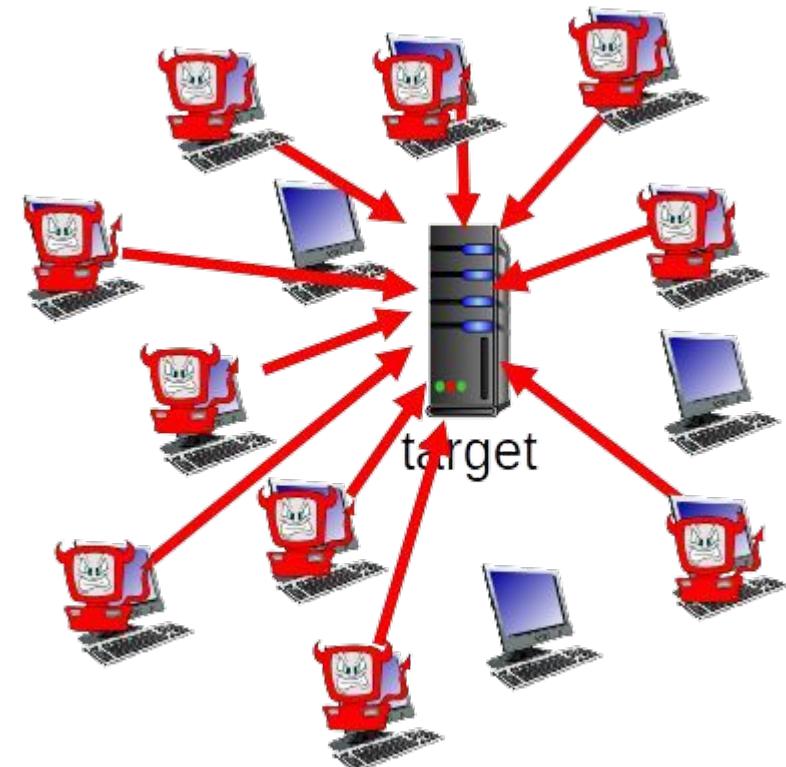
1. select target
2. break into hosts around the network
(botnet)



Bad guys: attack server, network infrastructure

Distributed Denial of Service (DoS):

1. select target
2. break into hosts around the network
(botnet)
3. send packets to target from
compromised hosts



List of defense

- **authentication:** proving you are who you say you are
 - cellular networks provides hardware identity via SIM card; no such hardware assist in traditional Internet
- **confidentiality:** via encryption
- **integrity checks:** digital signatures prevent/detect tampering
- **access restrictions:** password-protected VPNs
- **firewalls:** specialized “middleboxes” in access and core networks:
 - off-by-default: filter incoming packets to restrict senders, receivers, applications
 - detecting/reacting to DOS attacks

Introduction

- What's the Internet?
- What's a protocol?
- network edge; hosts, access net, physical media
- network core: packet/circuit switching Internet structure
- Internet structure
- protocol layers, service models
- performance: loss, delay, throughput
- security
- **history**

Internet History

1960s: circuit switching network

- telephone networks were dominants
- used circuit switching
 - appropriate choice for voice
- increasing importance of computers
- advent of time-shared computer
 - how to share computers among geographically distributed users
- The traffics were bursty

Internet History

1961-1972: Early packet-switching principles

- **1961:** Kleinrock - queueing theory shows effectiveness of packet-switching for bursty traffic
- **1964:** Baran - packet-switching in military nets
- **1967:** ARPAnet conceived by Advanced Research Projects Agency
- **1969:** ~~the first packet switch were installed at UCLA shortly after~~ **first ARPANet node had four nodes** three additional packet switch was installed at SRI (Stanford Research Institute), UC Santa Barbara and the university of Utah
 - first use of the network: remote login
- **1972:**
- ARPAnet public demo
- NCP (Network Control Protocol) first host-host protocol (RFC 001)
- first e-mail program
- ARPAnet has 15 nodes



The ARPANET in December 1969

Taken from <https://www.vox.com/a/internet-maps>

Internet History

1972-1980: Internetworking, new and proprietary nets

- 1970: ALOHAnet satellite network in Hawaii
- 1974: Cerf and Kahn - architecture for interconnecting networks
 - all networks able to communicate using shared standards
- 1976: Ethernet at Xerox PARC
- Late 70's: proprietary architectures: DECnet, SNA, XNA
 - late 70's: switching fixed length packets (ATM precursor)
- 1979: ARPAnet has 200 nodes



Cerf and Kahn's internetworking principles specified the basic format of data packets transmitted across the internet

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

define today's Internet architecture

Internet History

1980-1990: new protocols, a proliferation of networks

- **1983:** deployment of TCP/IP
- **1982:** smtp e-mail protocol defined
- **1983:** DNS defined for name-to-IP-address translation
- **1985:** ftp protocol defined
- **1988:** TCP congestion control
- new national networks: CSnet, BITnet, NSFnet, Minitel
- 100,000 hosts connected to confederation of networks

Internet History

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

- **early 1990's:** ARPAnet decommissioned
- **1991:** NSF lifts restrictions on commercial use of NSFnet
 - NFS decommissioned in 1995
 - was replaced by commercial Internet Service Providers: UUNet, AT&T, Sprint, and Level 3.
- **early 1990s:** World Wide Web
 - a platform for enabling and deploying many applications: Google Search, Amazon, ebay, social networks
 - hypertext [Bush 1945, Nelson 1960's]
 - HTML, HTTP, web server, browser
 - **1994:** Mosaic, later Netscape: GUI-based web browsers
- late 1990's – 2000's:
 - est. 50 million host, 100 million+ users
 - E-mail, including attachment and Web-accessible e-mail
 - instant messaging, P2P file sharing
 - network security to forefront
 - backbone links running at Gbps
 - video applications:
 - distribution of user-generated videos (youtube)
 - on-demand streaming of movies and TV shows
 - multi-person video conference

Internet History

2005-present: scale, SDN, mobility, cloud

- aggressive deployment of broadband home access (10-100's Mbps)
- Rise of smartphones: more mobile than fixed devices on the Internet
- ~26B devices attached to Internet (2019)
 - Smartphones, tablets, things
- service providers (Google, FB, Microsoft) create their own networks
 - bypass commercial Internet to connect "close" to end user, providing "instantaneous" access to social media, search, video content, ...
- increasing ubiquity of high-speed wireless access: wifi, 4G/5G networks
- emergence of online social networks:
 - Facebook: ~ one billion users
- service providers (Google, Microsoft) create their own networks
 - bypass Internet, providing "instantaneous" access to search, video content, email, etc.
- e-commerce, universities, enterprises running their services in "cloud" (e.g., Amazon EC2, Microsoft Azure)

Summary

- covered a “ton” of material!
 - Internet overview
 - What’s a protocol?
 - network edge, core, access network
 - packet-switching versus circuit-switching
 - performance: loss, delay, throughput
 - Internet structure
 - layering, service models
 - security
 - history
- you now have:
 - context, overview, “feel” of networking
 - more depth, detail to follow!

Application Layer

Some network apps

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

Outline

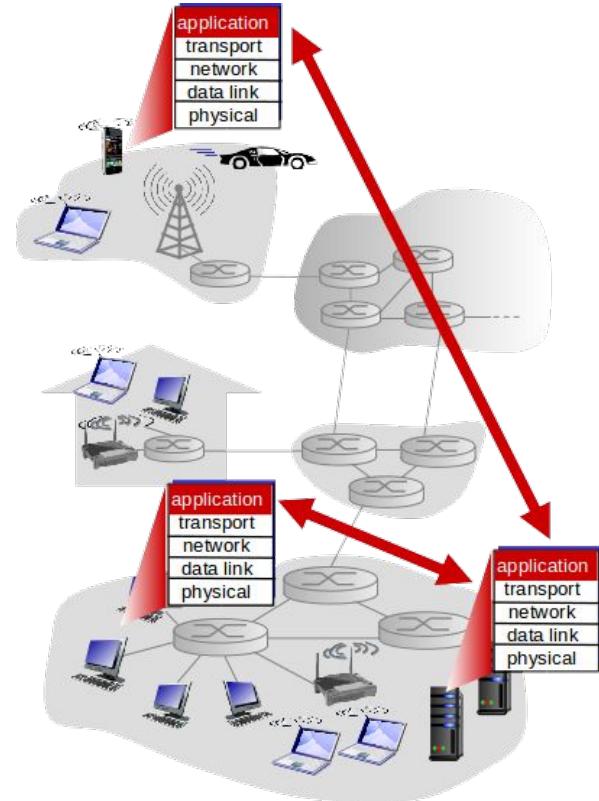
- **principles of network applications**
- socket programming with UDP and TCP
- Web and HTTP
- electronic mail
 - SMTP, POP3, IMAP
- DNS
- P2P applications
- video streaming and content distribution networks

Goal

- our goals:
 - Conceptual and implementation aspects of network application protocols
 - Concepts:
 - Network services required by applications
 - Transport-layer service models
 - Clients and servers
 - Applications:
 - Web, email, DNS, P2P file distribution and video streaming
 - HTTP, SMTP / POP3 / IMAP, DNS
 - Network applications development
 - socket API
 - Using both TCP and UDP
 - Developing applications in python

Creating a network app

- write programs that:
 - run on (different) end systems
 - communicate over network
 - E.g., Web application:
 - web server software communicates with browser software
- no need to write software for network-core devices
 - network-core devices do not run user applications
 - Limiting the applications developing to end systems (hosts) facilitated rapid app development and propagation



Application architectures

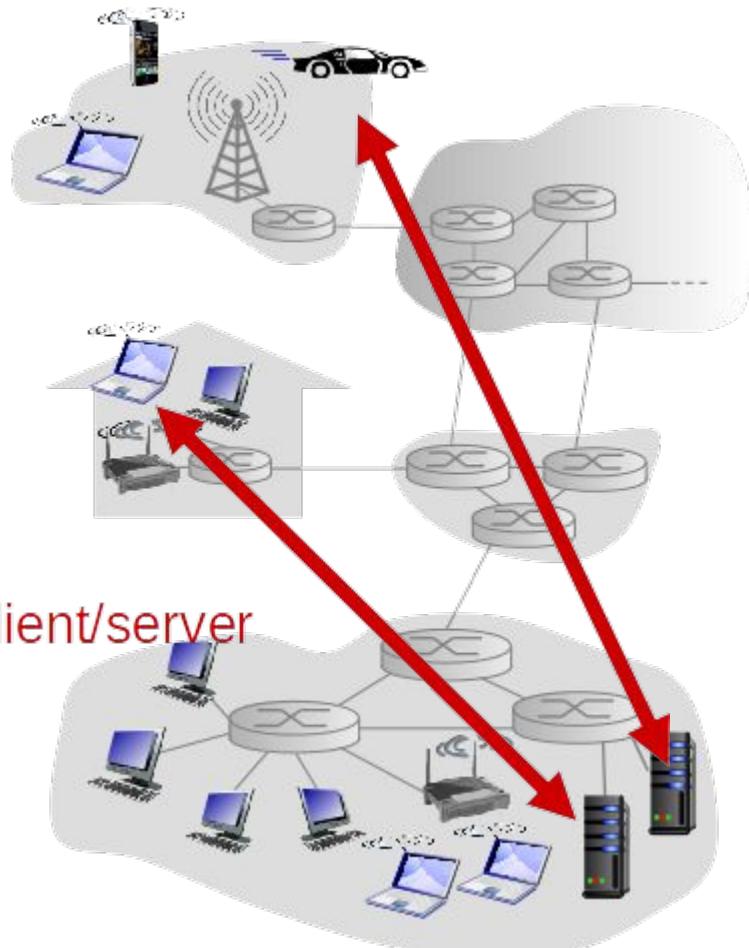
possible structure of applications:

- client-server
- peer-to-peer (P2P)

Question: Is application architecture the same as Network architecture?

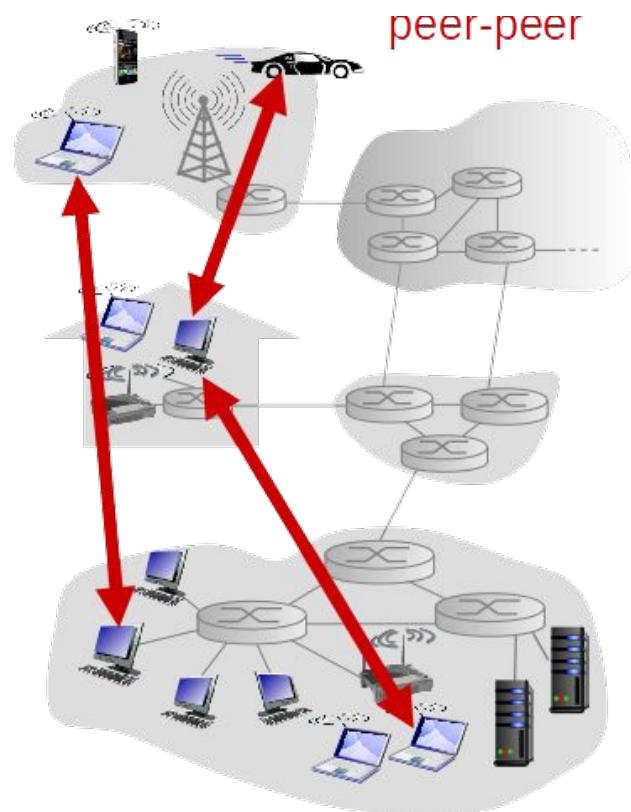
Client-server architecture

- server:
 - always-on host
 - permanent IP address
 - data centers for scaling
- Clients:
 - communicate with server
 - may be intermittently connected
 - may have dynamic IP addresses
 - do not communicate directly with each other
- Example applications:
 - Web application, FTP, Telnet, email



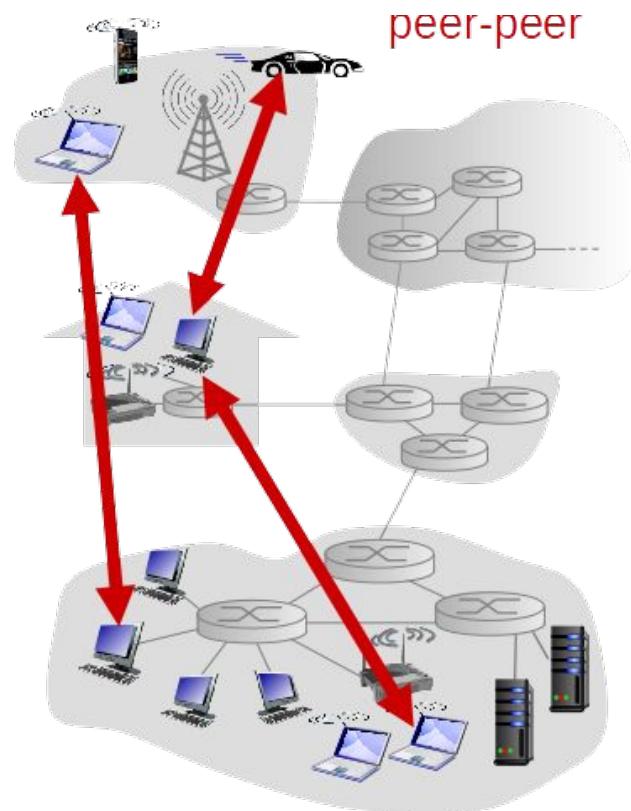
P2P architecture

- no always-on server
- arbitrary end systems (peer) directly communicate
 - peers are not owned by service providers, desktops and laptops controlled by users
 - peers request service from other peers, provide service in return to other peers



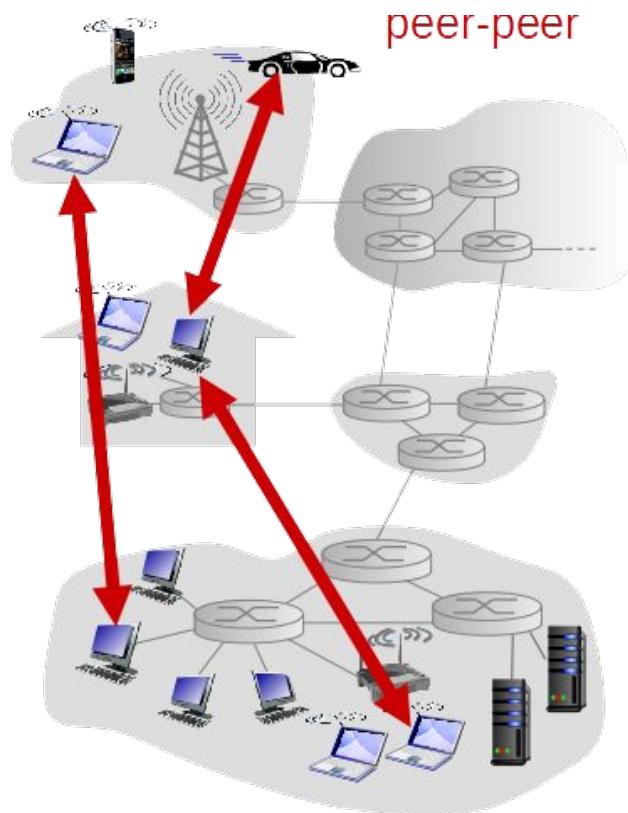
P2P architecture

- Examples
 - file sharing (BitTorrent)
 - Internet telephony and video conference (skype)
 - applications with **hybrid architecture**
 - instant messaging applications
 - Servers track the IP addresses of users
 - user-to-user messages are sent directly between user hosts



P2P architecture

- **self scalability**
 - new peers bring new service capacity, as well as new service demands
 - in P2P file sharing application
 - each peer generate workload
 - each peer add service capacity to the system
- **cost effective**
 - don't require significant server infrastructure and server bandwidth
- **Challenges**
 - security, performance, and reliability
 - peers are intermittently connected and change IP addresses
 - complex management



Processes communication

- process: program running within a host
 - within same host, two processes communicate using **inter-process communication** (defined by OS)
 - processes in different hosts communicate by exchanging **messages across the computer network**
- For each **pair of communicating processes**
 - **client process**: process that initiates communication
 - **server process**: process that waits to be contacted
 - Client-server architecture:
 - Web application: browser process initializes contact with a web server process
 - P2P architecture:
 - in any **communication session between a pair of processes**
 - one process is labelled as client, one process is labelled as a server

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door (process is analogous to a house)
 - sending process sends message out door
 - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process

Sockets

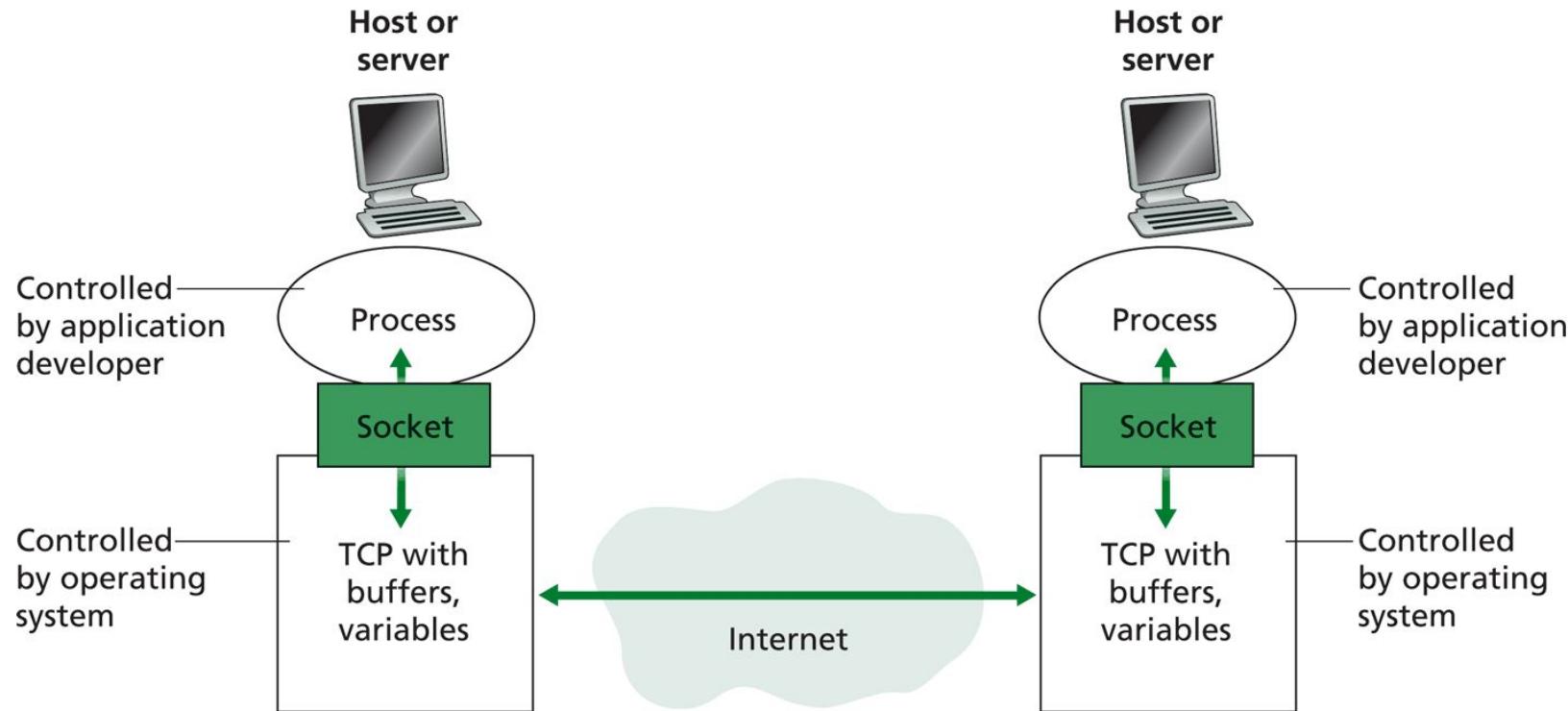


Figure 2.3 ♦ Application processes, sockets, and underlying transport

Addressing processes

- to receive messages, process must have **identifier**
- host device has unique 32-bit IP address
 - **Q:** does IP address of host on which process runs suffice for identifying the process?
 - **A:** no, many processes can be running on same host
- **identifier: IP address and port numbers** associated with process on host.
 - example port numbers:
 - HTTP server: 80
 - mail server: 25
- to send HTTP message to www.wlu.ca web server:
 - IP address: 216.249.48.130
 - port number: 80

What transport service does an app need?

- Reliable data transfer (reliability)
 - some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
 - other apps (e.g., audio) can tolerate some loss
- Throughput
 - some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
 - Bandwidth-sensitive applications
 - other apps (“elastic apps”) make use of whatever throughput they get
 - Elastic applications
- Timing
 - some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”
- Security
 - encryption, data integrity, ...

Question

- **Question:** None of the applications, requires both no data loss and timing.
Name an application that requires no data loss and is highly sensitive
- **Answer:** Online word processors

Transport service requirements: common apps

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
text messaging	no loss	elastic	yes and no

Internet transport protocols services

- TCP service:
 - reliable transport between sending and receiving process
 - connection-oriented: setup required between client and server processes
 - flow control: sender won't overwhelm receiver
 - congestion control: throttle sender when network overloaded
 - does not provide: timing, minimum throughput guarantee, security
- UDP service:
 - unreliable data transfer between sending and receiving process
 - does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,
- Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (<u>e.g.</u> , YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (<u>e.g.</u> , Skype)	TCP or UDP

Securing TCP

- TCP & UDP
 - no encryption
 - cleartext passwords sent inot sockets traverse Internet in cleartext
- SSL
 - Provides encrypted TCP connection
 - Data integrity
 - End-point authentication
- SSL is at app layer
 - apps use SSL libraries, that “talk” to TCP
- SSL socket API
 - cleartext passwords sent into socket traverse the Internet encrypted

Question

Suppose you wanted to do a transaction from a remote client to a server as fast as possible. Would you use UDP or TCP? Why?

- A. TCP
- B. UDP

App-layer protocol defines

- **types of messages exchanged**
 - e.g., request, response
- **message syntax**
 - what fields in messages & how fields are delineated
- **message semantics**
 - meaning of information in fields
- **rules** for when and how processes send & respond to messages
- **open protocols:**
 - defined in RFCs
 - allows for interoperability
 - e.g., HTTP, SMTP
- **proprietary protocols:**
 - e.g., Skype

Network Applications vs. Application Layer Protocol

- An application-layer protocol is only one piece of a network applications
 - Web: a client-server application
 - standard for document format (HTML)
 - Web browsers
 - Web Servers
 - Application layer protocol: HTTP
 - Internet email application
 - mail servers
 - mail clients (outlook)
 - standard for defining the structure of an email message
 - application-layer protocols (SMTP): defines how messages are passed between servers
 - how messages are passed between servers and mail clients
 - how the contents of message headers are to be interpreted --SMTP (RFC 5321)

Outline

- principles of network applications
- socket programming with UDP and TCP
- **Web and HTTP**
- electronic mail
 - SMTP, POP3, IMAP
- DNS
- P2P applications
- video streaming and content distribution networks

Web and HTTP

- Web page consists of
 - Base html file
 - Objects, each of which can be stored on a different web server
 - object can be HTML file, JPEG image, audio file, ...
 - web page consists of **base HTML-file** which includes **several referenced objects**
 - each object is addressable by a URL, e.g.,

`http://www.funwebdev.com/index.php?page=17#article`

Protocol

Domain

Path

Query String

Fragment

HTTP Overview

- HTTP: hypertext transfer protocol
 - Web's application layer protocol
 - client/server model
 - client: browser that requests, receives, (using HTTP protocol) and “displays” Web objects
 - server: Web server sends (using HTTP protocol) objects in response to requests
- RFC7231(HTTP 1.1)



HTTP Overview

- uses TCP:
 - client initiates TCP connection (creates socket) to server, port 80
 - server accepts TCP connection from client
 - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
 - TCP connection closed

HTTP Connection

non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection then closed
 - downloading multiple objects required multiple connections

default mode in HTTP/1.0

HTTP Connection

non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection then closed
 - downloading multiple objects required multiple connections

default mode in HTTP/1.0

persistent HTTP

1. TCP connection opened
2. multiple objects can be sent over single TCP connection between client, server
3. TCP connection then closed
 - Default mode in HTTP/1.1

Default mode in HTTP/1.1

Non-persistent HTTP: Example



user enters URL:www.someSchool.edu/someDepartment/home.index
(contains text, references to 10 jpeg images)



1a. HTTP client initiates TCP connection to
HTTP server (process) at
www.someSchool.edu on port 80

2. HTTP client sends HTTP request message
(containing URL) into TCP connection socket.
Message indicates that client wants object
someDepartment/home.index

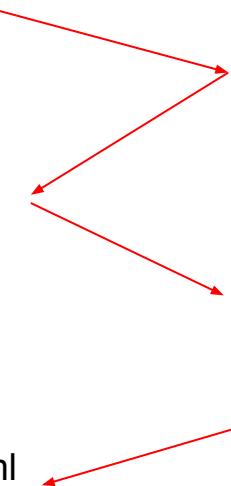
5. HTTP client receives response message
containing html file, displays html. Parsing html
file, finds 10 referenced jpeg objects

1b. HTTP server at host
www.someSchool.edu waiting for TCP
connection at port 80. “accepts” connection,
notifying client

3. HTTP server receives request message,
forms response message containing
requested object, and sends message into
its socket

4. HTTP server closes TCP connection.

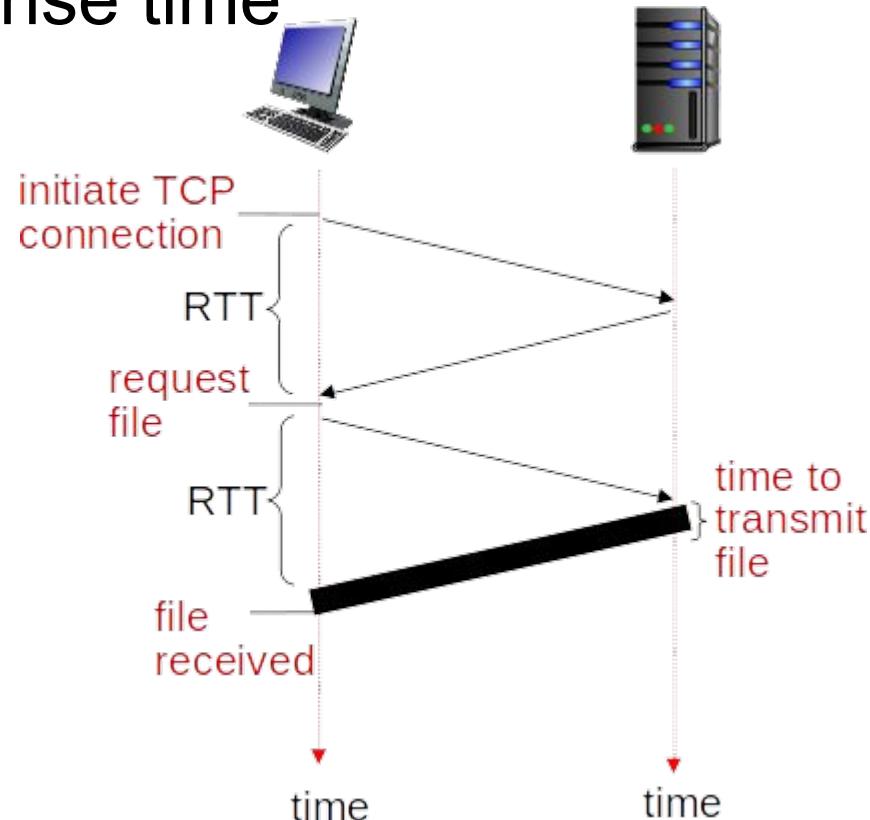
6. Steps 1-5 repeated for each of 10 jpeg objects



Non-persistent HTTP: response time

- **RTT (definition):** time for a small packet to travel from client to server and back
- **HTTP response time:**
 - one RTT to initiate TCP connection
 - one RTT for HTTP request and first few bytes of HTTP response to return
 - file transmission time

**non-persistent HTTP response time =
2RTT + file transmission time**



Persistent HTTP

- non-persistent HTTP issues:
 - requires 2 RTTs per object
 - browsers often open parallel TCP connections to fetch referenced objects
 - OS overhead for each TCP connection
- persistent HTTP:
 - server leaves connection open after sending response
 - subsequent HTTP messages between same client/server sent over open connection
 - without waiting for replies to pending request (pipelining)
 - client sends requests as soon as it encounters a referenced object
 - as little as one RTT for all the referenced objects (**cutting response time in half**)
 - HTTP server closes a connection when it isn't used for a certain time
 - Example: http1.1

Question

A user request a web page that consists of some text and three images. For this page, the client will send one request message and receive four response messages

- A. True
- B. Flase

HTTP Messages

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

HTTP Request Message

request line
(GET, POST,
HEAD commands)

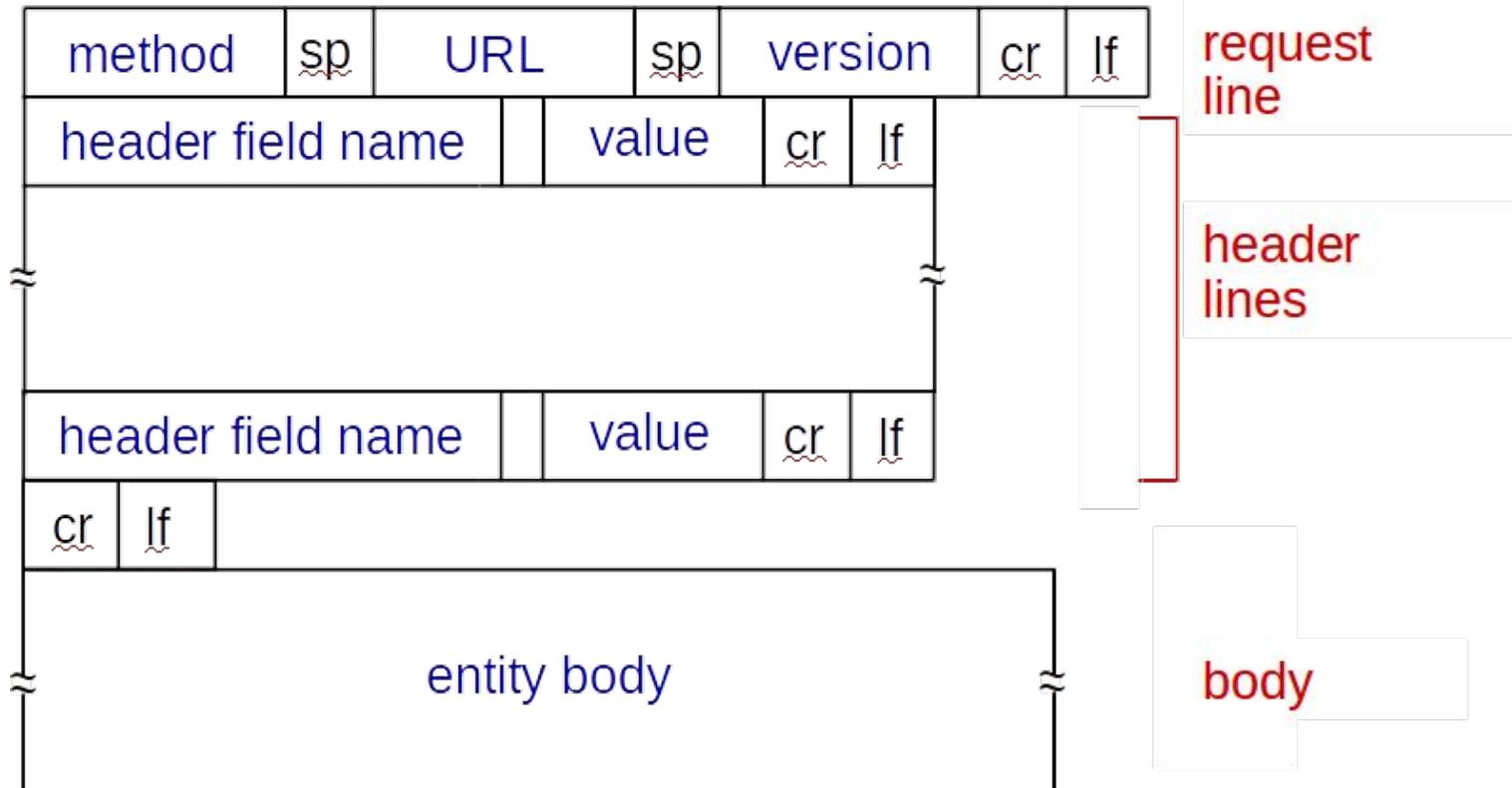
header
lines

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

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

carriage return character
line-feed character

HTTP Request Message: general format



HTTP Request messages

- POST method:
 - web page often includes form input. input is uploaded to server in entity body of an http message
- GET method:
 - Request an object. Input may be uploaded in URL field of request line (the part after question mark in the following example):
 - www.somesite.com/animalsearch?monkeys&banana
- HEAD method:
 - requests headers (only) that would be returned if specified URL were requested with an HTTP GET method.
- PUT method:
 - completely replaces file that exists at specified URL with content in entity body of POST HTTP request message
- DELETE
 - deletes file specified in the URL field

Method types

- HTTP/1.0:
 - GET
 - POST
 - HEAD
- HTTP/1.1:
 - GET
 - POST
 - HEAD
 - PUT
 - DELETE

There are different method types in different versions of http

HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS)\r\nLast-Modified: Tue, 30 Oct 2007 17:00:02\r\nGMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-\r\n1\r\n\r\ndata data data data data ...
```

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:
 - 200 OK
 - request succeeded, requested object later in this msg
 - 301 Moved Permanently
 - requested object moved, new location specified later in this msg (Location:)
 - 400 Bad Request
 - request msg not understood by server
 - 404 Not Found
 - requested document not found on this server
 - 505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

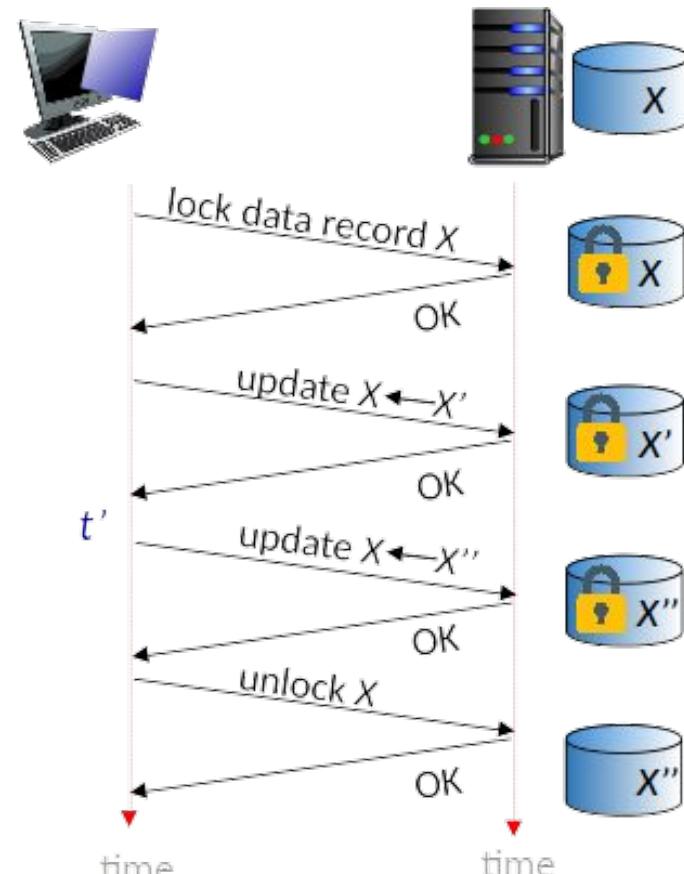
- 1. Telnet to your favorite Web server:
 - telnet www.wlu.ca 80
 - opens TCP connection to port 80 (default HTTP server port) at wlu.ca
 - anything typed in will be sent to port 80 at wlu.ca
- 2. type in a GET HTTP request:
 - GET /index.php HTTP/1.1 Host: wlu.ca
 - by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server
- 3. look at response message sent by HTTP server!
 - (or use Wireshark to look at captured HTTP request/response)

Maintaining user-server state: cookies

- HTTP is “stateless”
 - All http requests are independent of each other.
 - server maintains no information about ongoing client requests
 - no notion of multi-step exchanges of HTTP messages to complete a Web “transaction”
 - no need for client/server to track “state” of multi-step exchange
 - no need for client/server to “recover” from a partially-completed-but-never-completely-completed transaction
 - simplifies server design

Maintaining user-server state: cookies

- a **stateful** protocol: client makes two changes to X, or none at all
- Q: what happens if network connection or client crashes at t' ?
 - past history (state) must be maintained
 - if server/client crashes, their views of “state” may be inconsistent, must be reconciled



Maintaining user-server state: cookies

- HTTP is “stateless”
 - All http requests are independent of each other.
 - server maintains no information about past client requests
 - no notion of multi-step exchanges of HTTP messages to complete a Web “transaction”
 - no need for client/server to track “state” of multi-step exchange
 - no need for client/server to “recover” from a partially-completed-but-never-completely-completed transaction
 - simplifies server design
 - high performance web servers are developed that can handle thousands of simultaneous TCP connections
 - protocols that maintain “state” are complex!
 - how to keep “state”:
 - protocol endpoints: maintain state at sender/receiver over multiple transactions
 - cookies: http messages carry state

Maintaining user-server state: cookies

Web sites and client browser use **cookies** to maintain some state between transactions

four components:

1. **set-cookie** header line of HTTP **response** message
2. **cookie** header line in next HTTP **request** message
3. cookie file kept on user's host, managed by user's browser
4. back-end database at Web site

Maintaining user-server state: cookies

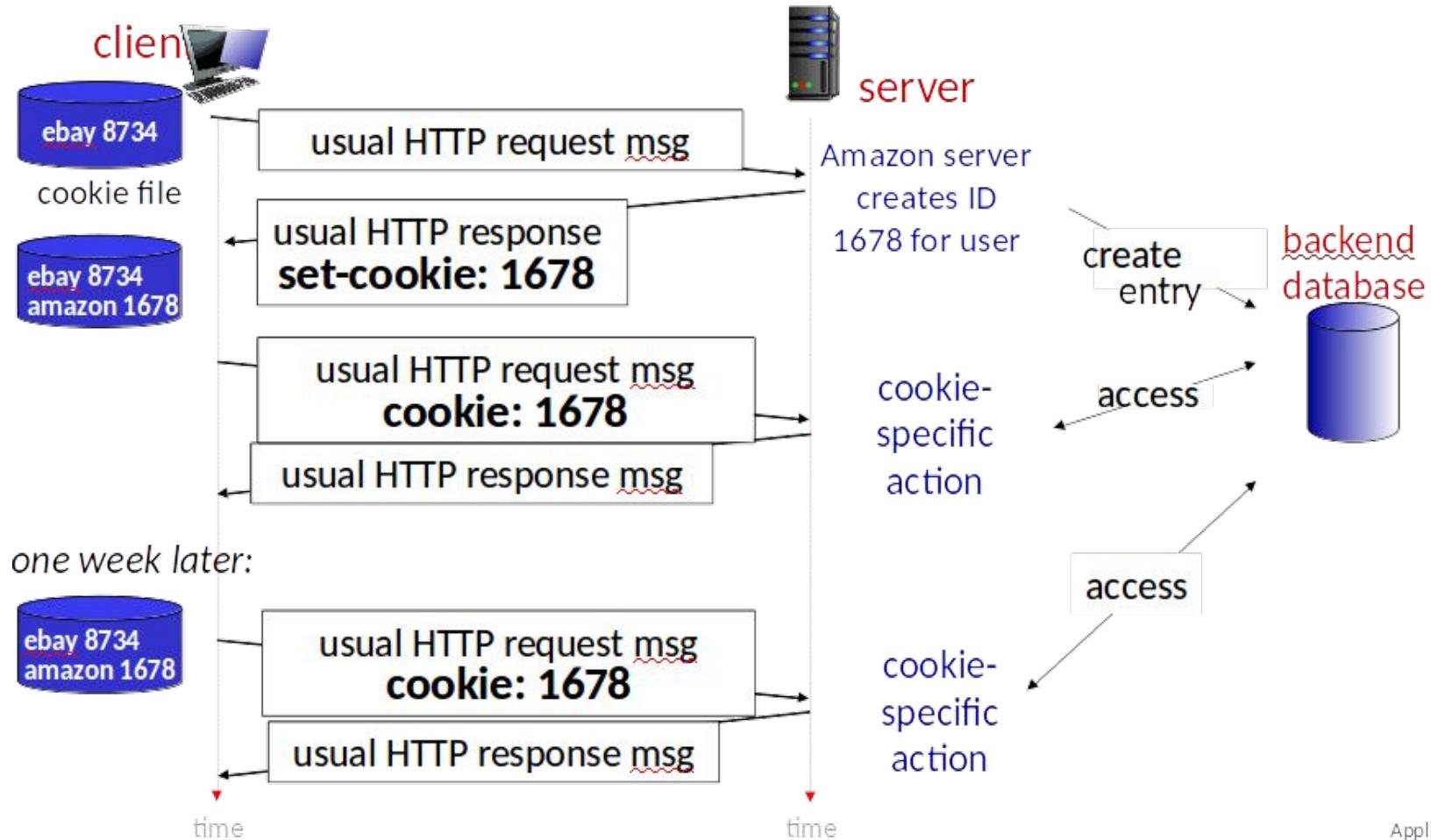
Web sites and client browser use **cookies** to maintain some state between transactions

four components:

1. **set-cookie** header line of HTTP **response** message
2. **cookie** header line in next HTTP **request** message
3. cookie file kept on user's host, managed by user's browser
4. back-end database at Web site

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
 - unique ID (aka "cookie")
 - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan



HTTP Cookies

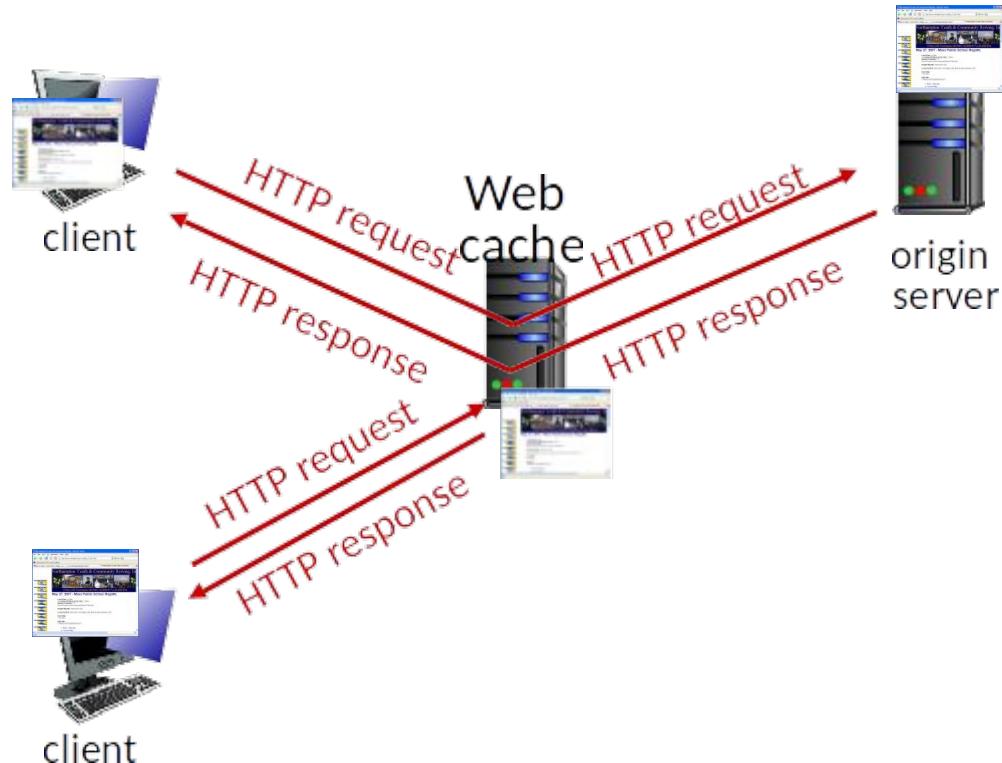
- what cookies can be used for:
 - Authorization
 - shopping carts: maintains a list of all of intended purchases
 - Recommendations: user session state (Web e-mail)
 - the first time user visit a site provide their identification
 - during subsequent session, the browser passes a cookie header to the server to identify the user to the server
- cookies and privacy:
 - cookies permit sites to learn a lot about you
 - you may supply name and e-mail to sites
 - cookie controversy : <http://www.cookiecentral.com/>
 - <https://gdpr.eu/what-is-gdpr/>
 -

Outline

- principles of network applications
- socket programming with UDP and TCP
- **Web and HTTP**
- electronic mail
 - SMTP, POP3, IMAP
- DNS
- P2P applications
- video streaming and content distribution networks

Web caches (proxy server)

- goal: satisfy client request without involving origin server
 - user sets browser: Web accesses via cache
 - browser sends all HTTP requests to cache
 - object in cache: cache returns object
 - else cache requests object from origin server, cache it, then returns object to client



More about Web caching

- Web cache acts as both client and server
 - server for original requesting client
 - client to origin server
- server tells cache about object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

- typically cache is installed by ISP (university, company, residential ISP)

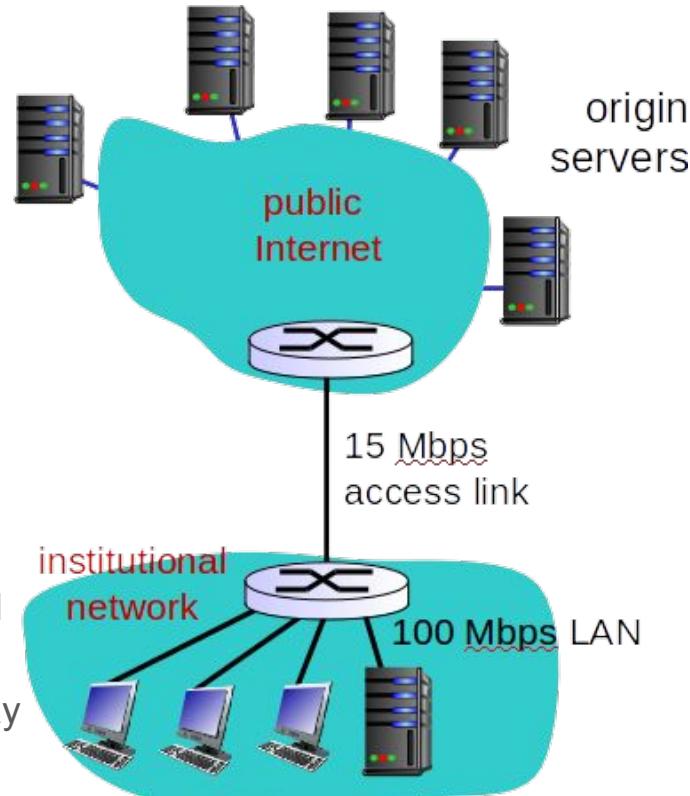
Why Web Caching?

- reduce response time for client request
 - especially if the bottleneck bandwidth between client and the origin server is much less than the bottleneck bandwidth between client and the cache
- reduce traffic on an institution's access link to the Internet
 - does not have to upgrade bandwidth as quickly
- reduce web traffic in the Internet as a whole
 - Because the origin server is not sending content so frequently.

Caching example:

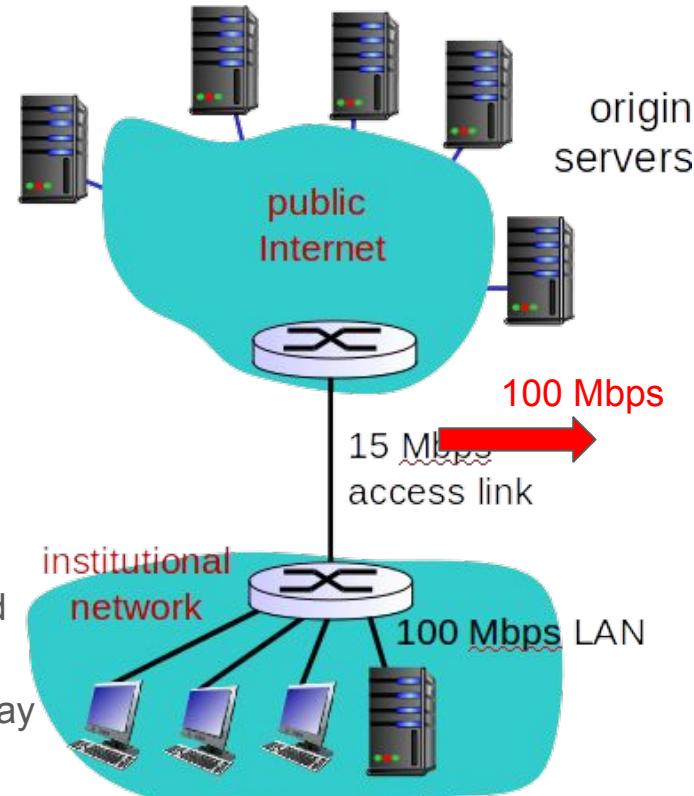
- Assumptions:
 - access link rate: 15 Mbps
 - avg object size: 1 Mbits
 - avg request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 15 Mbps
 - RTT from router on the public Internet to any origin server: 2 sec (Internet delay)
- Consequences:
 - Access link utilization: 1
 - LAN utilization: 0.15
 - Traffic density of LAN: $15 * 1 / 100 = 0.15$
 - Traffic density on the access link (between the institution and the Internet router) : $15 * 1 / 15 = 1$
 - end-to-end delay = Internet delay + access delay + LAN delay

$$= 2 \text{ sec} + \text{minutes} + \text{milliseconds} (0.01 \text{ s})$$



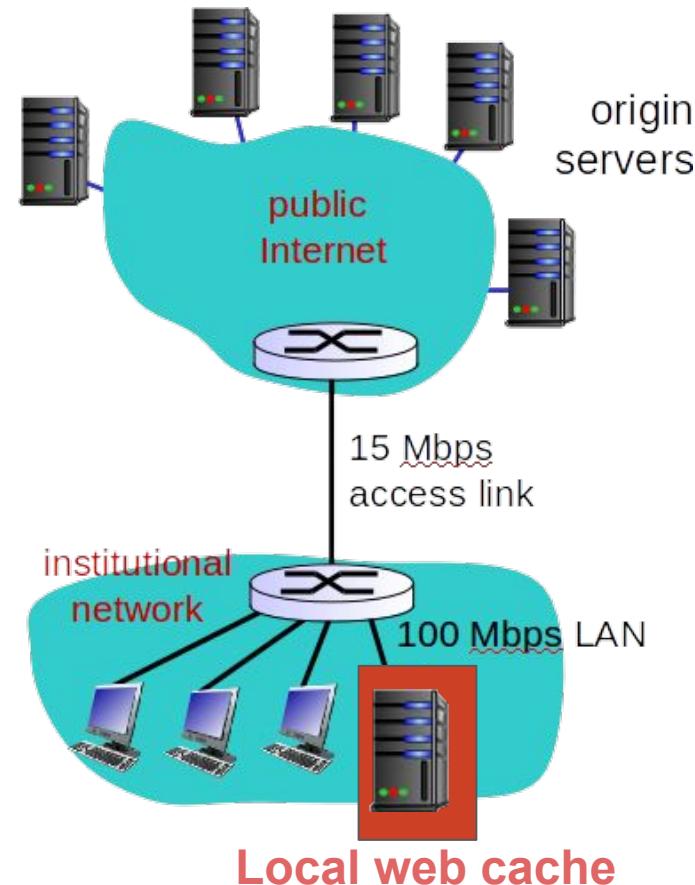
Caching example: replacing access link

- Assumptions:
 - access link rate: 15 Mbps
 - avg object size: 1 Mbits
 - avg request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 15 Mbps
 - RTT from router on the public Internet to any origin server: 2 sec (Internet delay)
- Consequences:
 - Access link utilization: 15 Mbps
 - LAN utilization: 0.15
 - Traffic density of LAN: $15 * 1 / 100 = 0.15$
 - Traffic density on the access link (between the institution and the Internet router) : $15 * 1 / 100 = 0.15$
 - end-to-end delay = Internet delay + access delay + LAN delay
 $= 2 \text{ sec} + \text{minutes} + \text{milliseconds} (0.01 \text{ s})$
milliseconds



Caching example: Install local cache

- Assumptions:
 - access link rate: 15 Mbps
 - avg object size: 1 Mbits
 - avg request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 15 Mbps
 - RTT from router on the public Internet to any origin server: 2 sec (Internet delay)
- Consequences:
 - Access link utilization: ?
 - LAN utilization: ?
 - Average end-to-end delay = ?

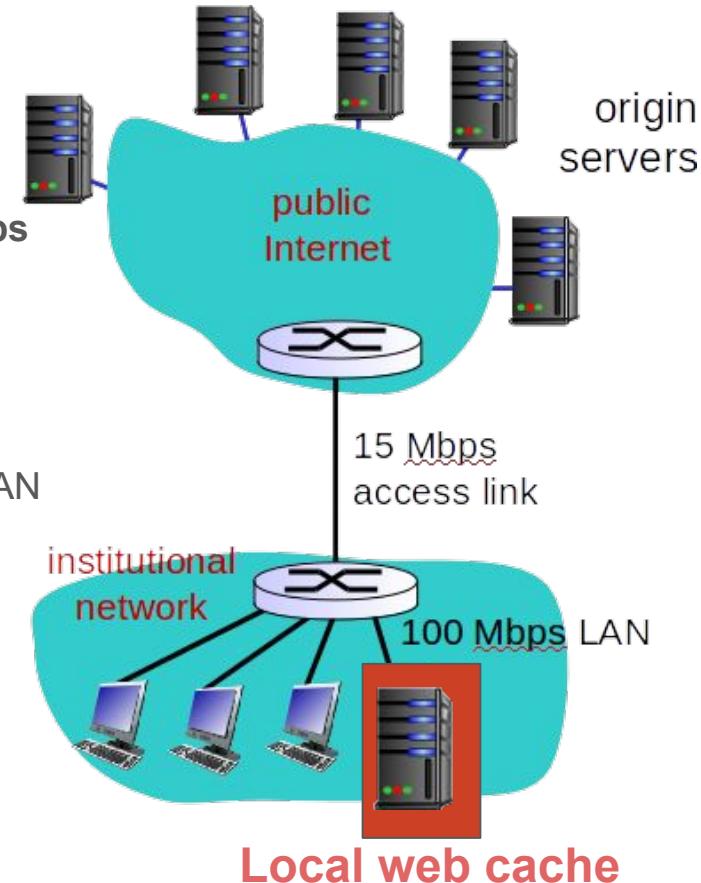


Caching example: Install local cache

- suppose cache hit rate is 0.4:

- 40% requests satisfied at cache, with low delay (msec)
- 60% requests satisfied at origin, use access link
 - Rate to browser over access link = $15 \text{ Mbps} * 0.6 = 9 \text{ Mbps}$
 - Access link utilization = $9 / 15 = 0.6$
 - Low (msec) queueing delay at access link
 - traffic density on access link: 0.6
 - a traffic intensity < 0.8 means a small delay
 - Average end-to-end delay = Internet delay + access delay + LAN delay
 - total delay with cache = $0.4 * 0.01 + 0.6 * 2.01 = 1.2 \text{ seconds}$

lower average end-end delay than with 15 Mbps link
(and cheaper too!)



Question

Will web caching reduce the delay for all objects (cached+non-cached) requested by a user or for only some of the objects (cached objects)?

- A. All of the objects
- B. Some of the objects

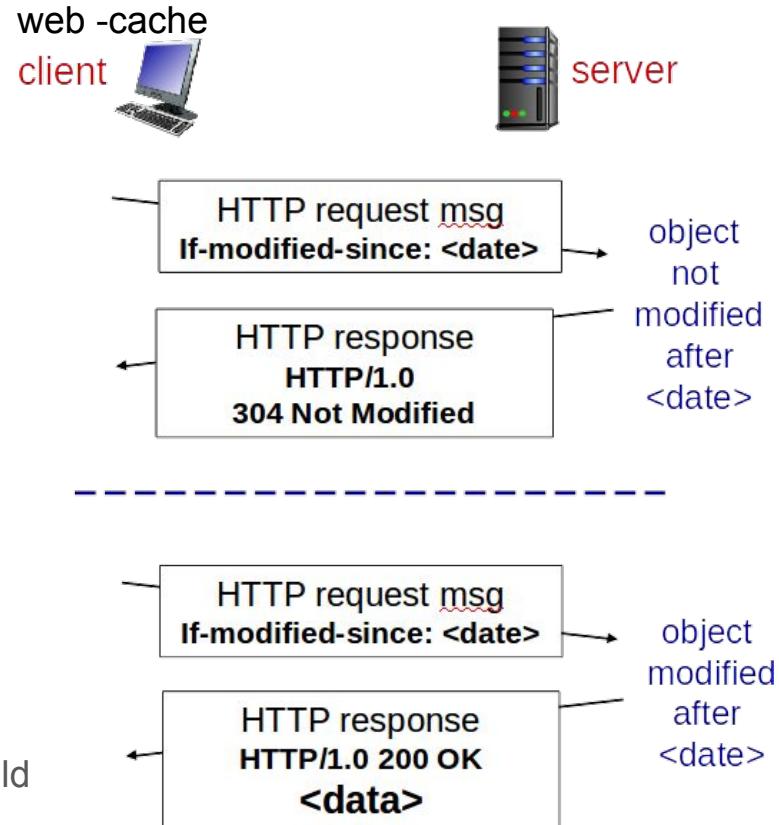
Caching

- Caching is also used on browser
- **Problem**
 - The objects in cache may be stale
 - Objects in a cache may have been modified since the copy was cached
- **Solution:** conditional get in HTTP
 - verify that objects are up to date

Conditional GET

- Goal: don't send object if cache has up-to-date cached version
 - no object transmission delay
 - lower link utilization
- **cache:** specify date of cached copy in HTTP request
 - **If-modified-since: <date>**
- **server:** response contains no object if cached copy is up-to-date:
 - **HTTP/1.0 304 Not Modified**

Note: **<date>** is the value of `last-modified` header field when the object was first received by the web cache



HTTP/2

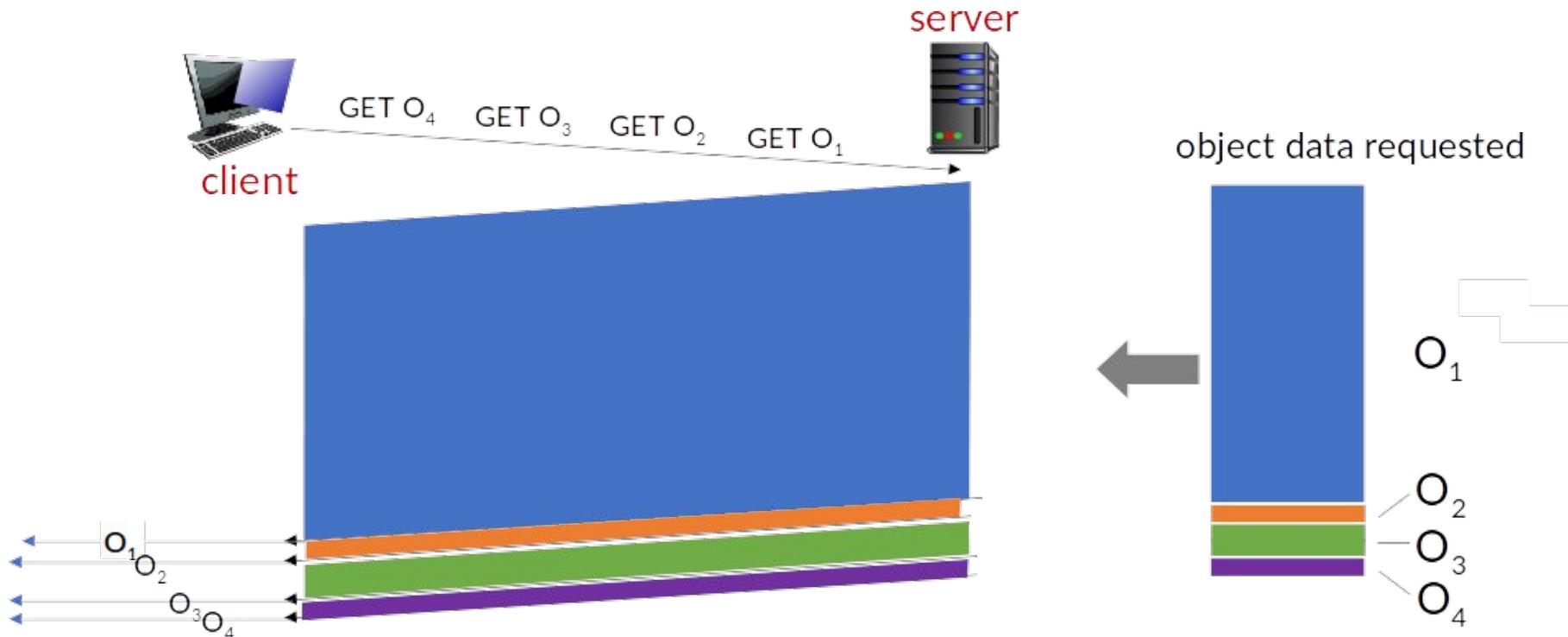
- **Key goal:** decreased delay in multi-object HTTP requests
- **HTTP1.1:** introduced **multiple, pipelined GETs** over single TCP connection
 - server responds in-order (FCFS: first-come-first-served scheduling) to GET requests
 - with FCFS, small object may have to wait for transmission (**head-of-line (HOL) blocking**) behind large object(s)
 - loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

- **Key goal:** decreased delay in multi-object HTTP requests
- **HTTP/2:** [[RFC 7540, 2015](#)] increased flexibility at server in sending objects to client:
 - methods, status codes, most header fields unchanged from HTTP 1.1
 - transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
 - push unrequested objects to client
 - divide objects into frames, schedule frames to mitigate HOL(Head of the line) blocking

HTTP/2: mitigating HOL blocking

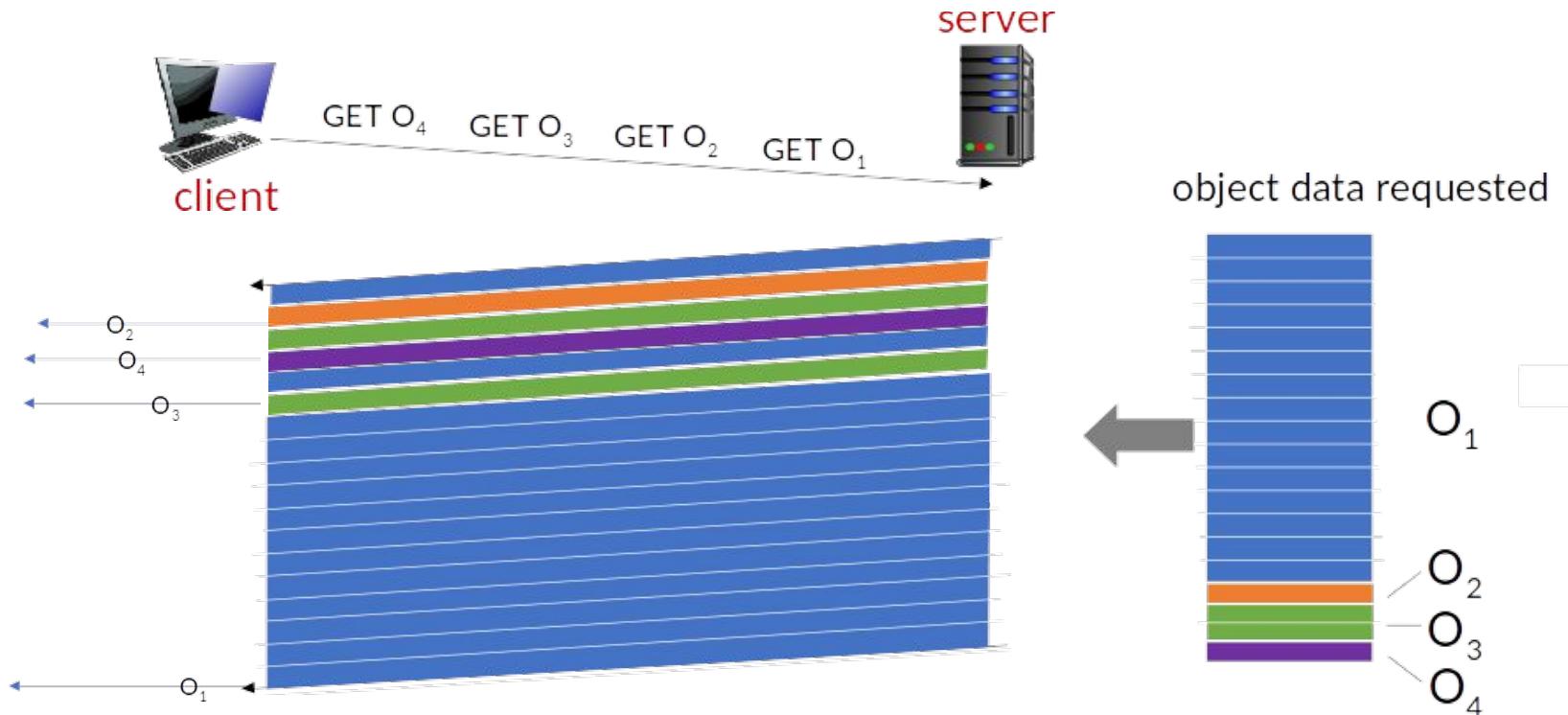
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



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

HTTP/2: mitigating HOL blocking

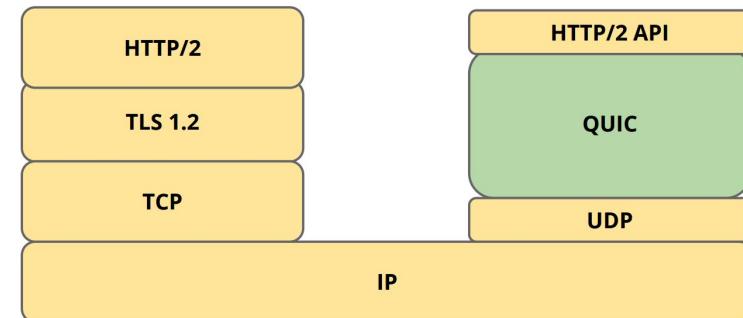
HTTP/2: objects divided into frames, frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/2 to HTTP/3

- HTTP/2 over single TCP connection means:
 - recovery from packet loss still stalls all object transmissions
 - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
 - no security over vanilla TCP connection,
- HTTP/3: adds security, per object error- and congestion-control (more pipelining) over UDP
 - more on HTTP/3 in transport layer
 - includes TLS/encryption in the connection establishment
 - All traffic encrypted (mostly)



Outline

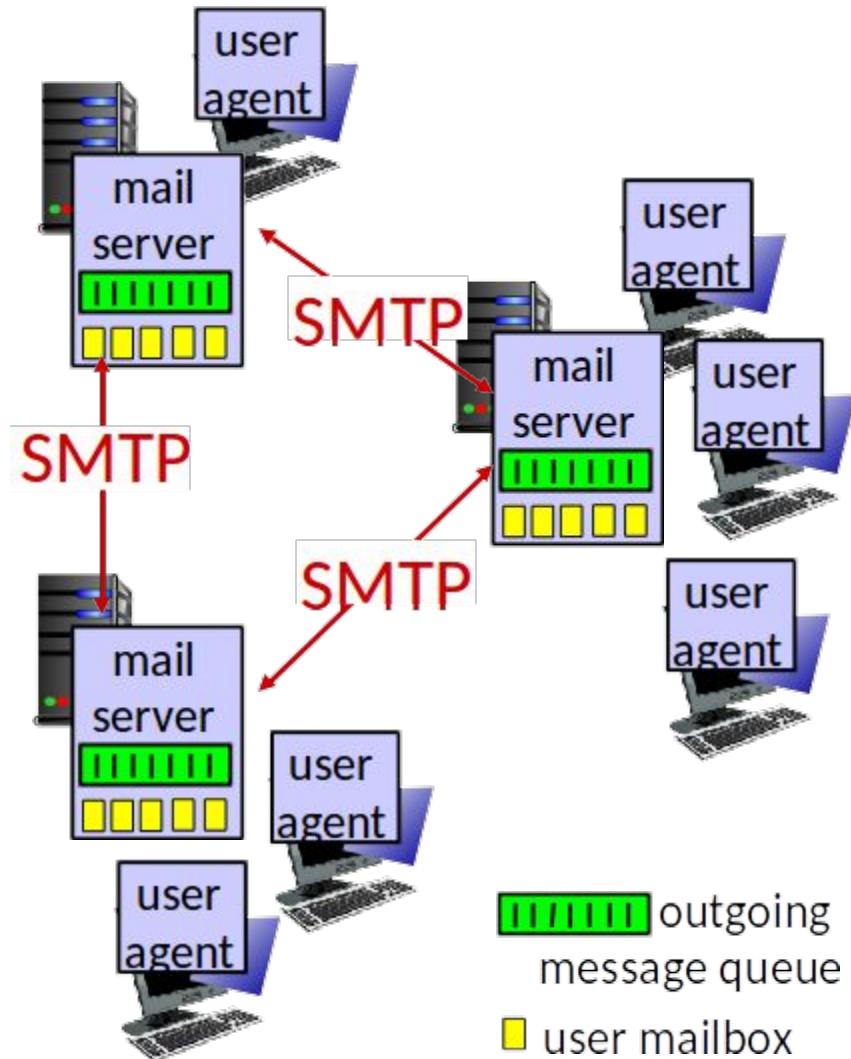
- principles of network applications
- Web and HTTP
- **electronic mail**
 - SMTP, POP3, IMAP
- DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

Electronic Email

- Infrastructure:
 - user agents
 - Servers
 - mailboxes
- Protocols:
 - SMTP: Simple Mail Transfer Protocol
 - POP
 - IMAP

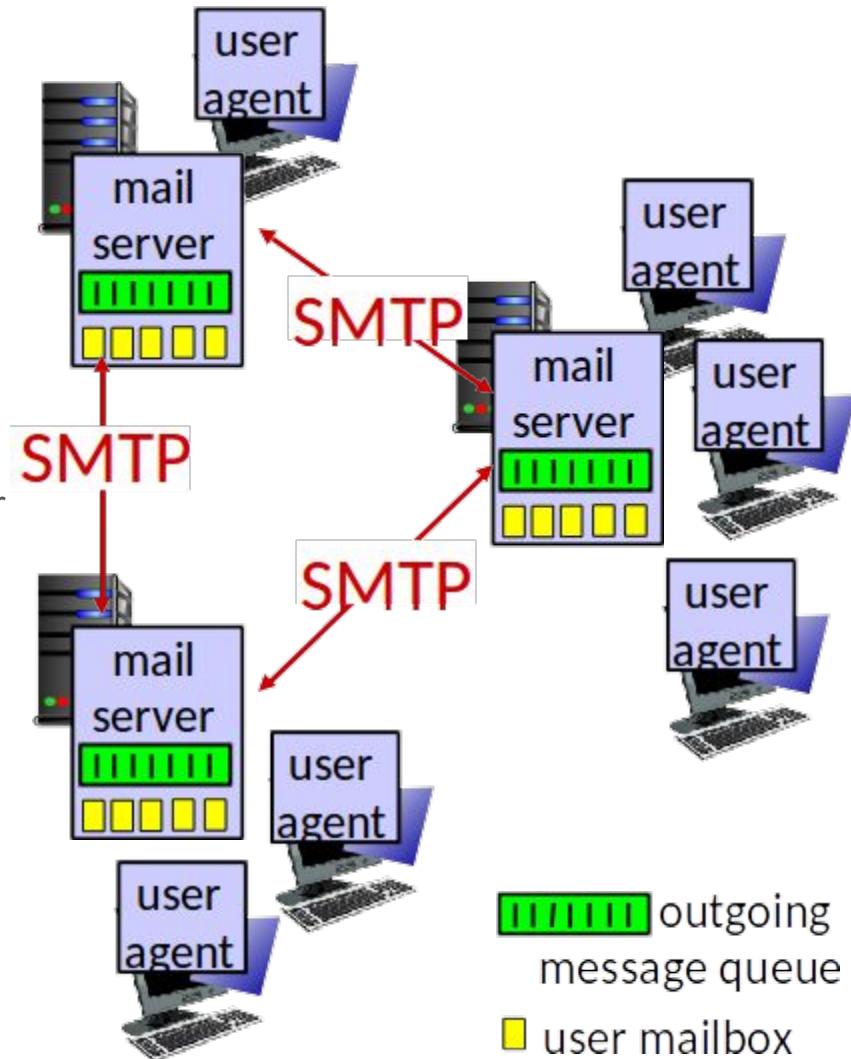
Electronic mail

- Three major components:
 - user agents
 - mail servers
 - simple mail transfer protocol: SMTP



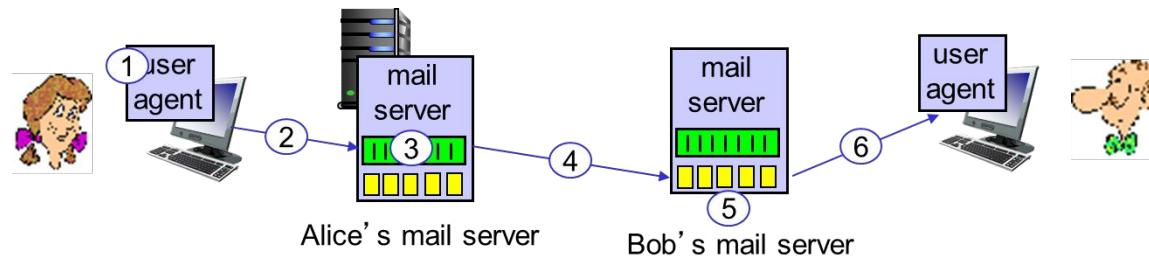
Electronic mail

- User Agent
 - a.k.a. “mail reader”
 - composing, editing, reading mail messages
 - e.g., Outlook, Thunderbird, iPhone mail client
 - outgoing, incoming messages stored on server
- 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: user agent/ sending mail server
 - “server”: receiving mail server



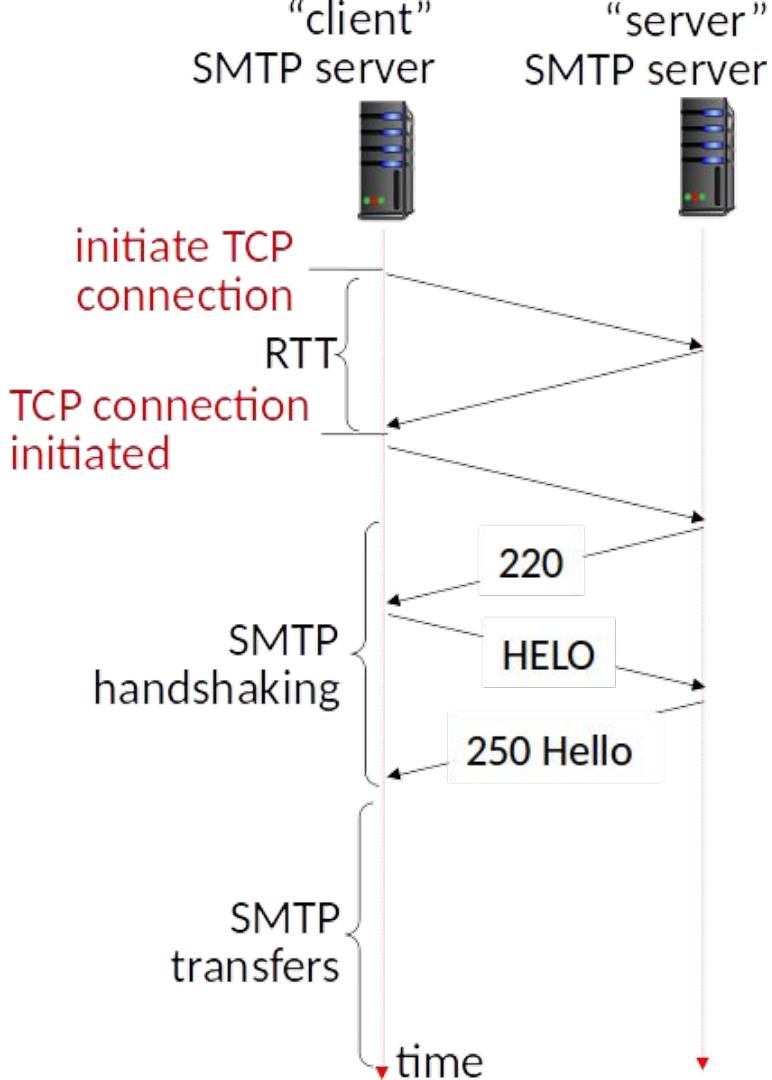
Scenario: Alice Sends Message to Bob

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



SMTP [RFC 5321]

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



Sample SMTP Interaction

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

Try SMTP Interaction for Yourself

telnet servername 25

unsecured version

telnet smtp.gmail.com 587

see 220 reply from server

enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

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

SMTP: observations

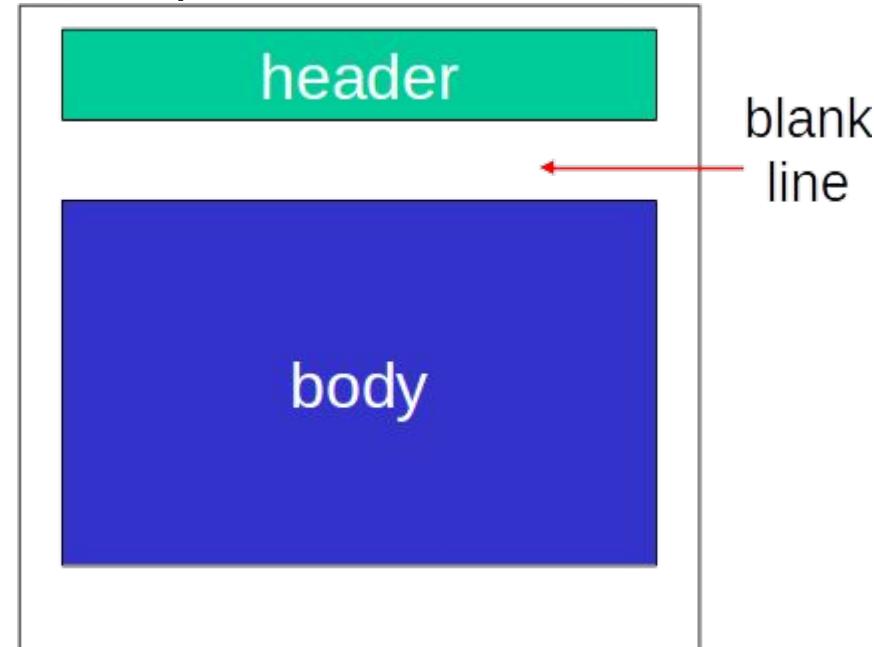
- Comparison with HTTP
 - HTTP: **client pull**: the machine that wants to receive the file initiates the TCP connection
 - SMTP: **client push**: the machine that wants to send the file initiates the TCP connection
 - both have ASCII command/response interaction, status codes
 - SMTP requires each message, including the body of each message, to be in 7-bit ASCII format, while HTTP does not
 - For example an image should be encoded into 7-bit ASCII
 - In SMTP multiple objects sent in one message while in HTTP each object encapsulated in its own response message
- SMTP uses persistent connection
- SMTP server uses CRLF.CRLF to determine end of message

Mail Message Format

- SMTP (RFC 5321): protocol for exchanging email messages
 - Like RFC 7231 defining HTTP
- RFC 2822: standard for email message format:
 - Like HTML standard defining syntax for web documents

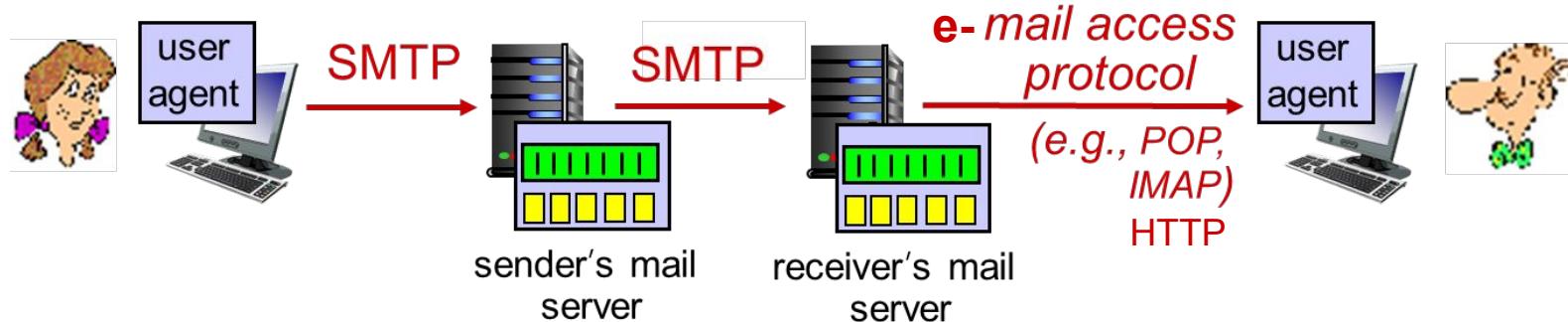
Mail Message Format (RFC 2822)

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



Mail Access Protocols

- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
 - POP: Post Office Protocol [RFC 1939]
 - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
 - HTTP: gmail, Hotmail, Yahoo! Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages



Web-Based Email

- user agents are web browser
- user communicate with its remote mailbox using HTTP
- communication between mail servers is done through SMTP

Outline

- principles of network applications
- Web and HTTP
- electronic mail
 - SMTP, POP3, IMAP
- **DNS**
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

DNS: Domain Name System

- people: many identifiers:
 - SSN, name, passport #
- Internet hosts, routers:
 - “name”, e.g., www.yahoo.com - used by humans
 - difficult to process by router
 - provide little information about the location of the host
 - IP address (32 bit) - used for addressing datagrams
 - four bytes (fixed length)
 - hierarchical structure
- Q: how to map between **name** and **address** (**Address resolution**)?
 - Naive solution: a centralized file

Before the DNS - HOSTS.TXT

- HOSTS.TXT: a file hosted on a machine at the NIC (Network Information Center)
- RFC952
- Hosts periodically used a file transfer protocol to download new version
- Names were initially flat, became hierarchical (e.g., lcs.mit.edu) ~85
- Not manageable or efficient as the ARPANET grew ...

Domain Name System

- **DNS Servers**
 - **distributed database** implemented in hierarchy of many name servers
- **DNS Protocol**
 - **application-layer protocol**: hosts, name servers communicate to resolve names (address/name translation)
 - allows hosts to query the distributed database
- note: core Internet function, implemented as application-layer protocol
 - complexity at network's "edge"

DNS: services

- hostname to IP address translation
- host aliasing
 - canonical, alias names
 - alias hostname are more mindful
- mail server aliasing
 - a company web server and mail server can have identical (aliased) hostname
- load distribution
 - replicated Web servers: many IP addresses correspond to one name
 - server rotates the ordering of the addresses

DNS Structure

- why not centralize DNS?
 - single point of failure
 - traffic volume
 - distant centralized database
 - maintenance
- A: doesn't scale!
 - Comcast DNS servers alone: 600B DNS queries/day
 - Akamai DNS servers alone: 2.2T DNS queries/day

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

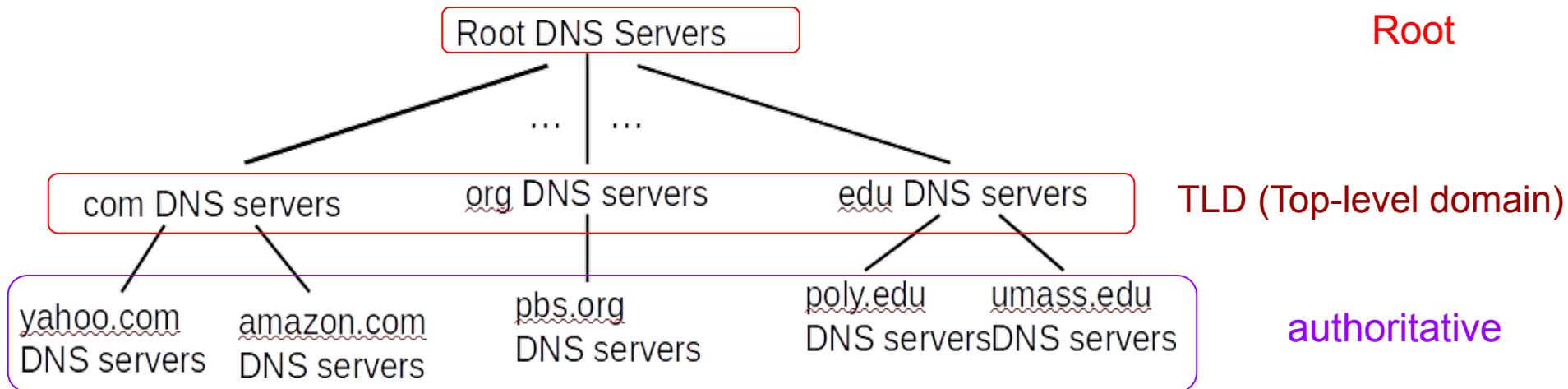
“bulletproof”: reliability, security



DNS: Example

- Example: a browser on some user's host requests the URL
`www.someschool.edu/index.htm`
 - The same user machine runs the client side of the DNS application
 - The browser extracts hostname from the URL, and passes it to the client side of the DNS application
 - The DNS client sends a query to a DNS server
 - The DNS client receives a reply: IP address of hostname
 - all DNS queries and reply messages are sent with UDP port 53
 - `gethostbyname()` function call
 - The browser initiates a TCP connection to HTTP server
- DNS adds an additional delay
 - the desired IP address is often cached in a nearby DNS server
 - reduce network traffic and average DNS delay

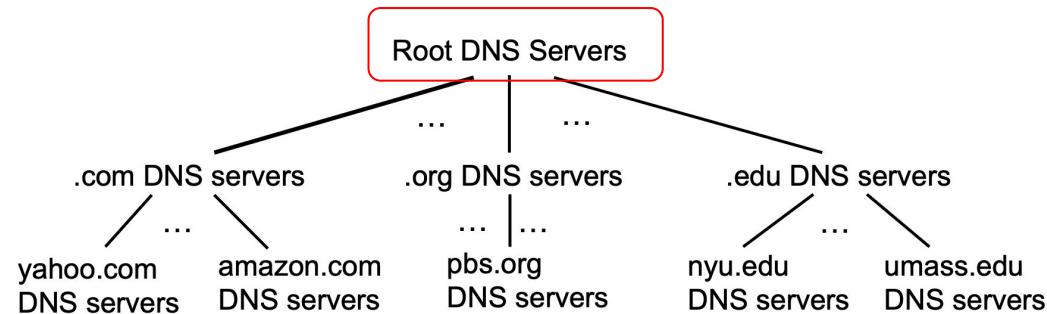
DNS: a distributed, hierarchical database



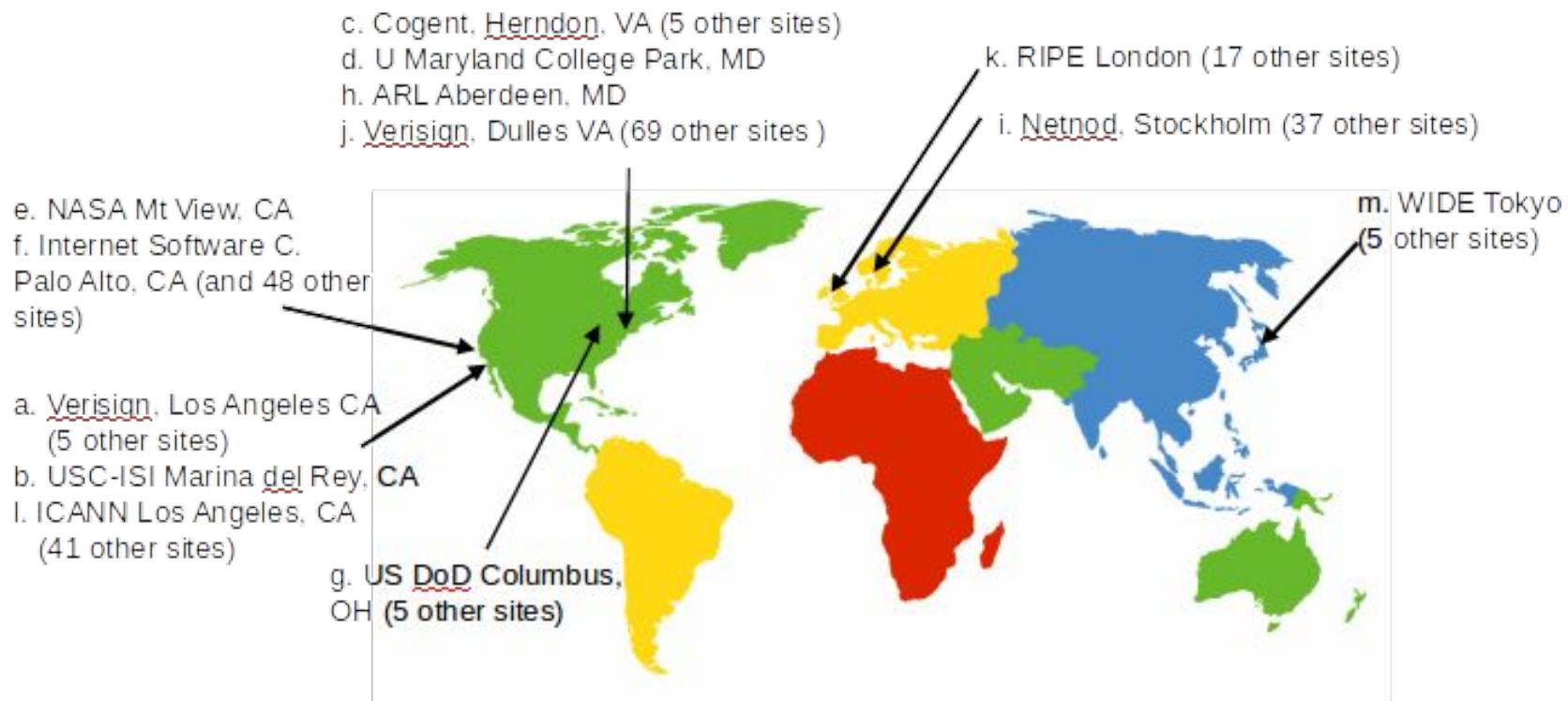
- client wants IP for www.amazon.com; 1st approximation:
 - client queries root server to find com DNS server
 - client queries .com DNS server to get amazon.com DNS server
 - client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
 - contacted by local name server that can not resolve name
 - root name server:
- incredibly important Internet function
 - Internet couldn't function without it!
 - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain



DNS: root name servers: <https://root-servers.org/>

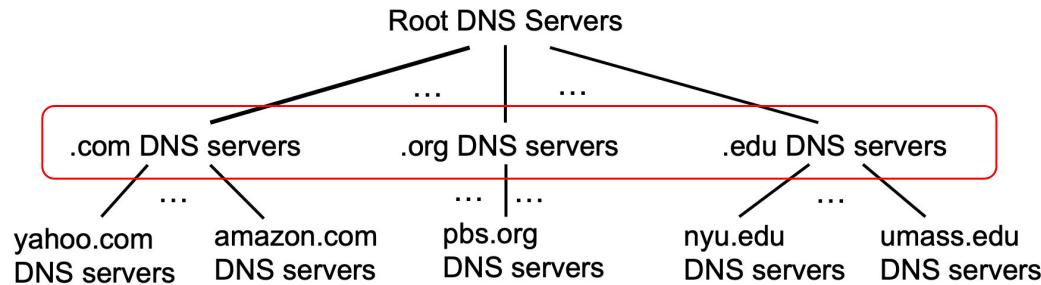


DNS: root name servers

- 13 logical root name “servers” worldwide
 - a.root-servers.net to m.root-servers.net
- each “server” replicated many times: 1368 physical servers
- All nameservers need root IP addresses
 - Handled via configuration file
- Most servers are reached by IP anycast
 - Multiple locations advertise same IP! Routes take client to the closest one.

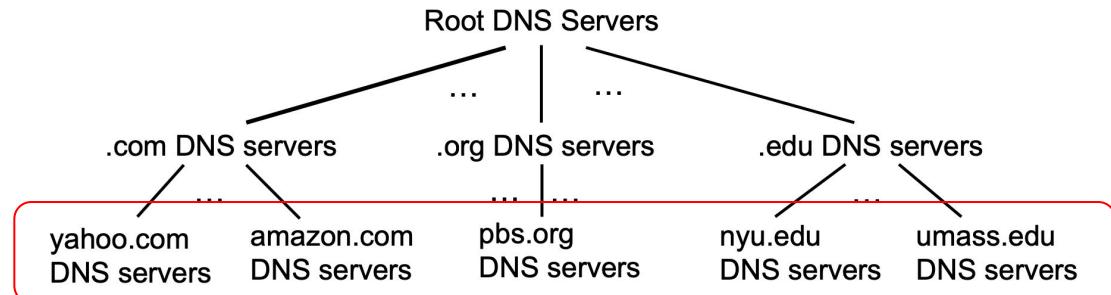
DNS: TLD (Top-Level Domains)

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



DNS: authoritative servers

- keep DNS record mapping an organization name to its IP
- Two ways to implement it
 - Organization's own DNS server(s)
 - Most universities and large companies implement and maintain their own authoritative DNS server
 - using authoritative DNS server of some service provider



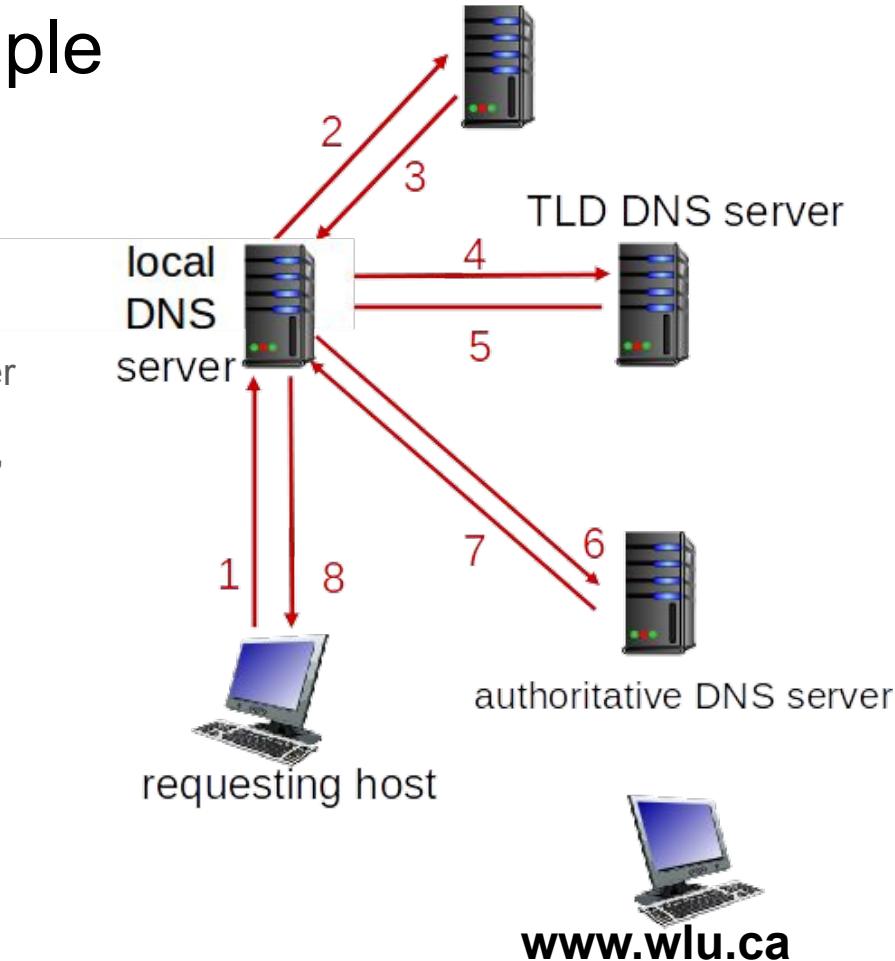
Local DNS name server

- Local DNS does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
 - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
 - has local cache of recent name-to-address translation pairs (but may be out of date!)
 - acts as proxy, forwards query into hierarchy
- What is your local DNS server
 - Linux: nm-tool
 - Window: ipconfig /all
 - Mac: scutil --dns
 - network status window in Windows

root DNS server

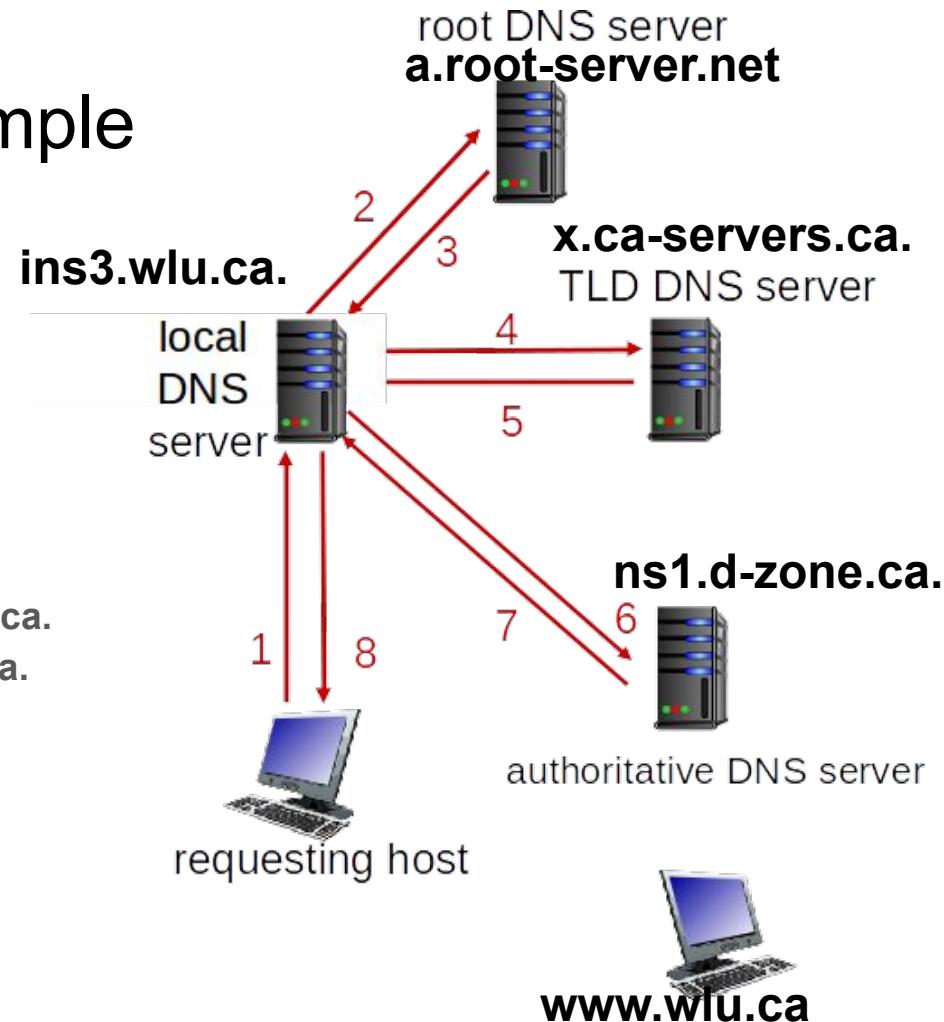
DNS name resolution example

- host at cs.uwaterloo.ca wants IP address for www.wlu.ca
- iterative query:**
 - contacted server replies with name of server to contact
 - "I don't know this name, but ask this server"



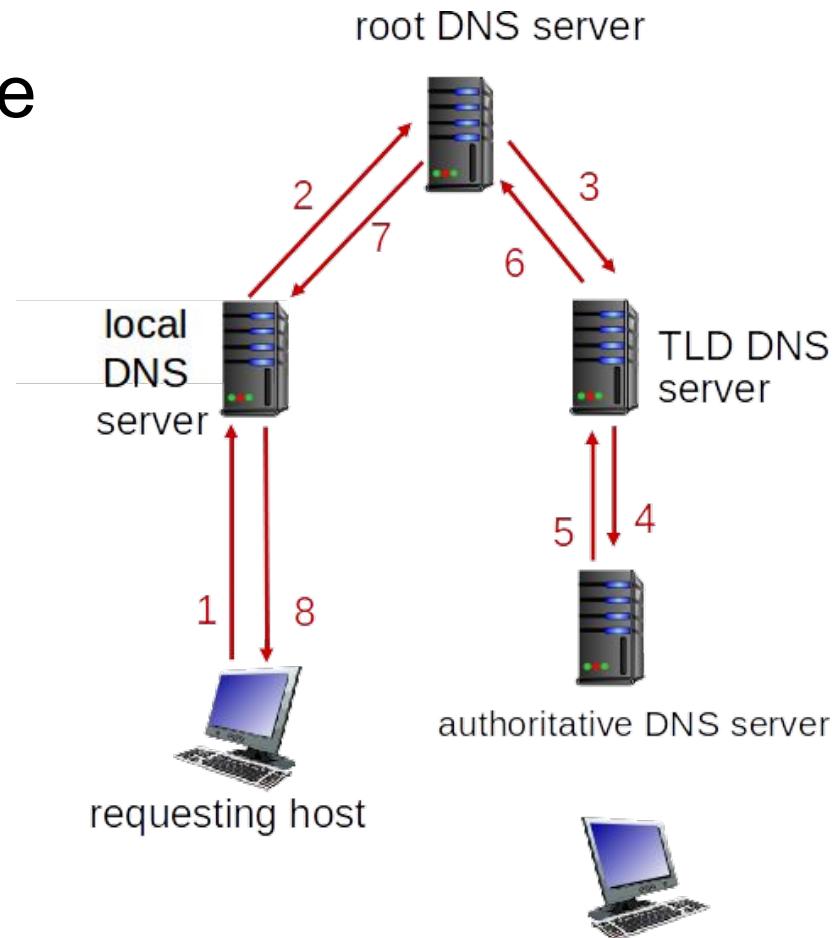
DNS name resolution example

- host at cs.uwaterloo.ca wants IP address for www.wlu.ca
- Command-line tool: dig
- Method 1
 - dig +norec www.wlu.ca @a.root-servers.net
 - dig +norec www.wlu.ca @x.ca-servers.ca.
 - dig +norec www.wlu.ca @ns1.d-zone.ca.
- Method 2
 - dig @8.8.8.8 +trace www.wlu.ca



DNS name resolution example

- host at cs.uwaterloo.ca wants IP address for www.wlu.ca
- recursive query:**
 - puts burden of name resolution on contacted name server
 - heavy load at upper levels of hierarchy



DNS: caching

- once (any) name server learns mapping, it **caches** mapping and immediately returns a cached mapping in response to a query
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers
 - thus root name servers not often visited
- cached entries may be out-of-date
 - if named host changes IP address, may not be known Internet-wide until **all** TTLs expire!
 - best-effort name-to-address translation!
 - update/notify mechanisms proposed IETF standard: RFC 2136
- Why caching
 - caching improves response time and reduce delays
 - reduce the number of DNS messages

DNS records

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

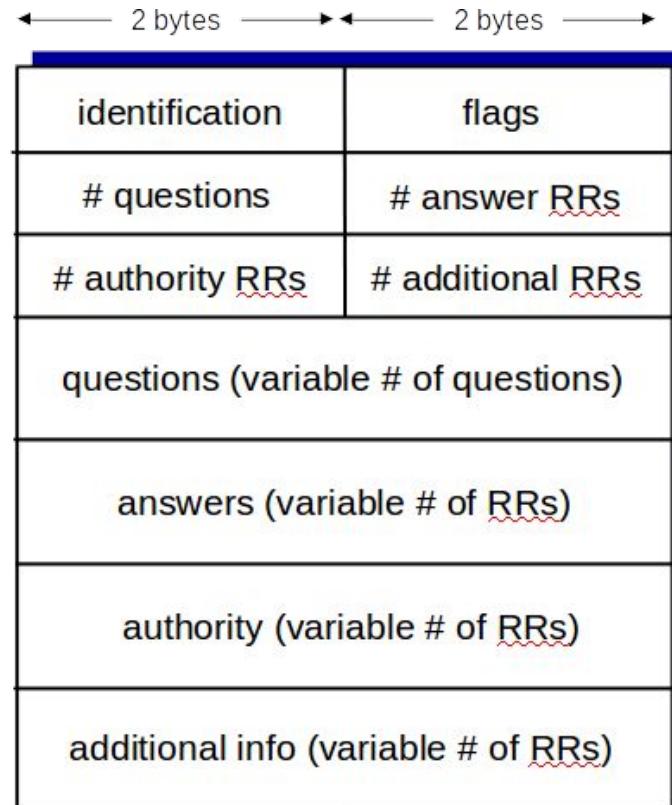
- DNS: distributed database storing resource records (RR)
- type=A
 - **name** is hostname, **value** is IP address
- type=NS
 - **name** is domain (e.g., foo.com)
 - **value** is hostname of authoritative name server for this domain
- type=CNAME
 - **name** is alias name for some “canonical” (the real) name
 - [www.wlu.ca](#) is really wwwprod.wlu.ca
 - **value** is canonical name
- type=MX
 - **value** is name of mailserver associated with **name**

```
dig www.wlu.ca
dig ns wlu.ca
nslookup -type=NS wlu.ca
dig cname www.wlu.ca
dig mx wlu.ca
nslookup -type=MX wlu.ca
```

DNS protocol messages

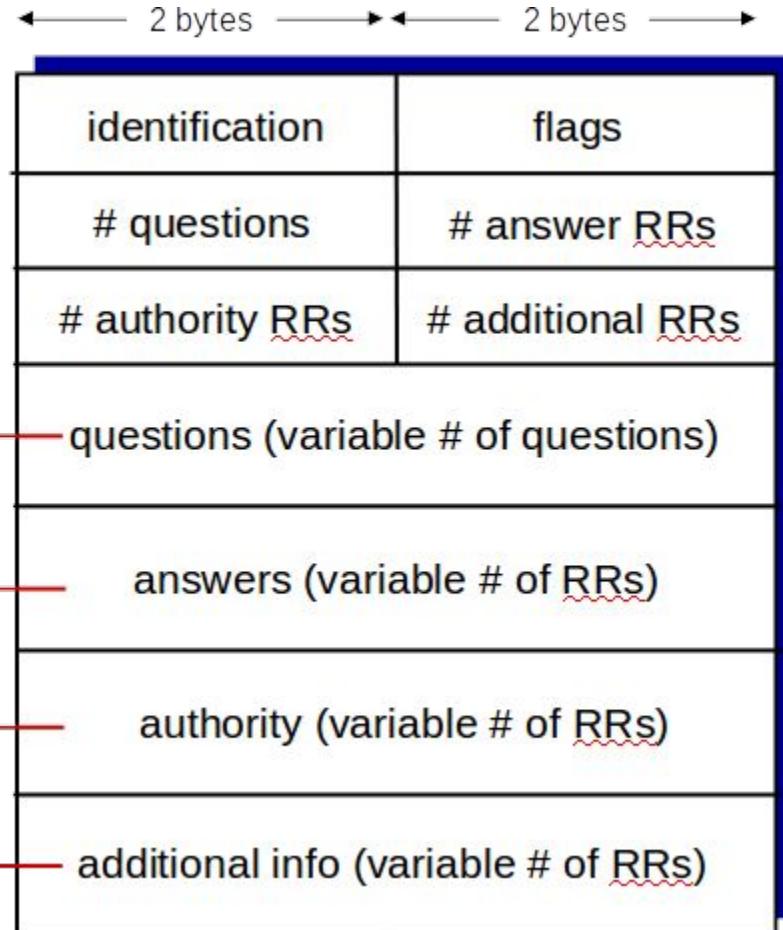
- **query** and **reply** messages, both with same message format
- message header
 - **identification**: 16 bit # for query, reply to query uses same #
 - **flags**:
 - query or reply
 - recursion desired
 - recursion available
 - reply is authoritative

Message format



DNS protocol, messages

- name, type fields for a query
- RRs in response to query
- records for authoritative servers
- additional “helpful” info that may be used



Inserting records into DNS

- example: new startup “Network Utopia” (networkutopia.com)
 - register name networkutopia.com at DNS registrar (e.g., Network Solutions)
 - Registrar verifies the uniqueness of the domain name
 - enters the domain name into the DNS database
 - Enters domain name into whois database
 - collects a small fee for its service
 - List of registrar: <https://www.internic.net/>
 - accredited by ICANN(Internet Corporation for Assigned Name and numbers

Inserting records into DNS

- example: new startup “Network Utopia” (networkutopia.com)
 - What records are inserted into into DNS database
 - Your provide names, IP addresses of authoritative name server (primary and secondary) to registrar
 - registrar inserts two RRs into .com TLD server:
 - (`networkutopia.com, dns1.networkutopia.com, NS`)
 - (`dns1.networkutopia.com, 212.212.212.1, A`)
 - create authoritative server type A record for `www.networkutopia.com`; type MX record for `networkutopia.com`
 - (`networkutopia.com, 212.212.71.4, A`)
 - (`networkutopia.com, mail.networkutopia.com, MX`)

Attacking DNS

- DDoS attacks
 - bombard root servers with traffic (e.g; ICMP ping messages)
 - not successful to date
 - traffic filtering
 - local DNS servers cache IPs of TLD servers, allowing root server bypass
 - bombard TLD servers
 - bombard TLD servers with a deluge of **DNS queries**
 - potentially more dangerous
- Spoofing attacks
 - intercept DNS queries, returning bogus replies
 - DNS cache poisoning
 - RFC 4033: DNSSEC authentication services

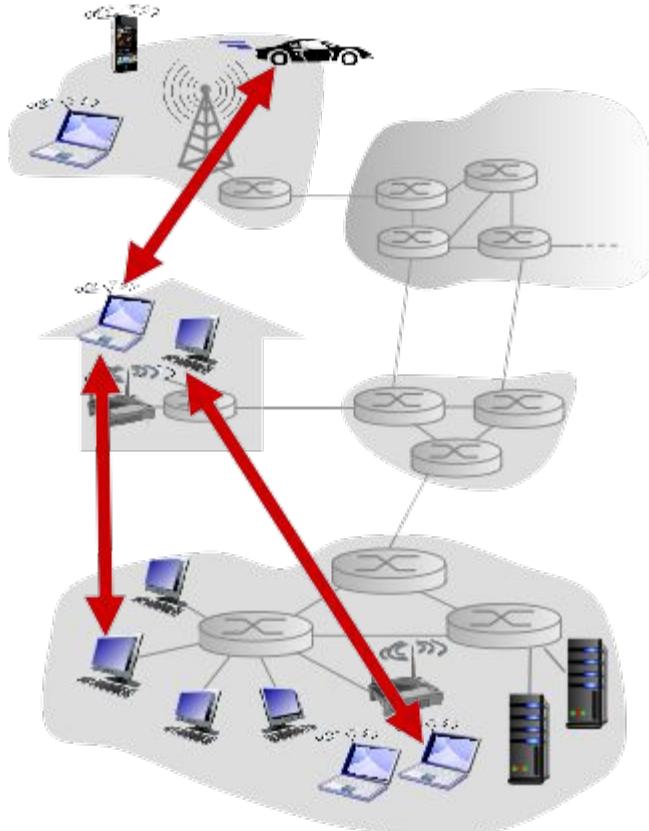
Why does DNS use UDP and not TCP?

Outline

- principles of network applications
- socket programming with UDP and TCP
- Web and HTTP
- electronic mail
 - SMTP, POP3, IMAP
- DNS
- **P2P applications**
- video streaming and content distribution networks

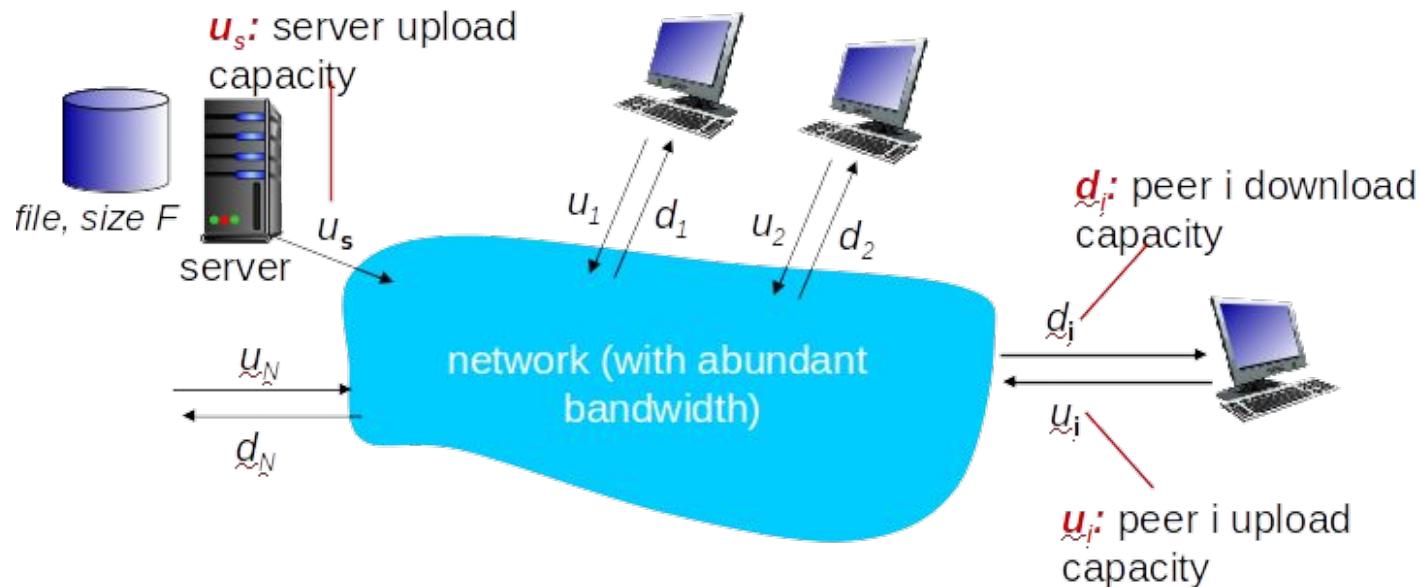
Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- Examples:
 - file distribution (BitTorrent)
 - Streaming (KanKan)
 - VoIP (Skype)



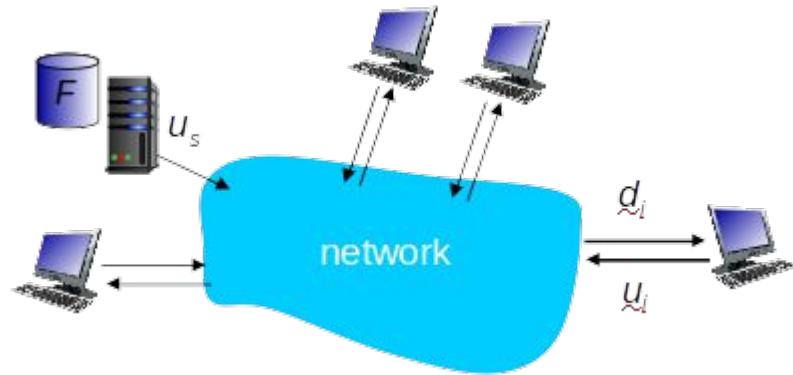
File distribution: client-server vs P2P

- Question: how much time to distribute file (size F) from one server to N peers?
 - peer upload/download capacity is limited resource



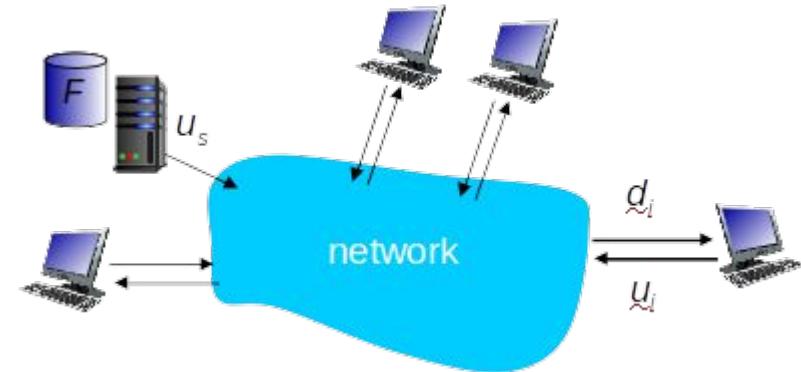
File distribution time: client-server

- Question: how much time to distribute file (size F) from one server to N peers?
 - simplifying assumption
 - All of the bottlenecks are in access networks
 - The server and client are not participating in any other network applications



File distribution time (D_{cs}): client-server

- **server transmission:** must sequentially send (upload) N file copies:
 - time to send one copy: F/u_s
 - time to send N copies: NF/u_s
- **client:** each client must download file copy
 - d_{min} = min client download rate
 - min client download time: F/d_{min}



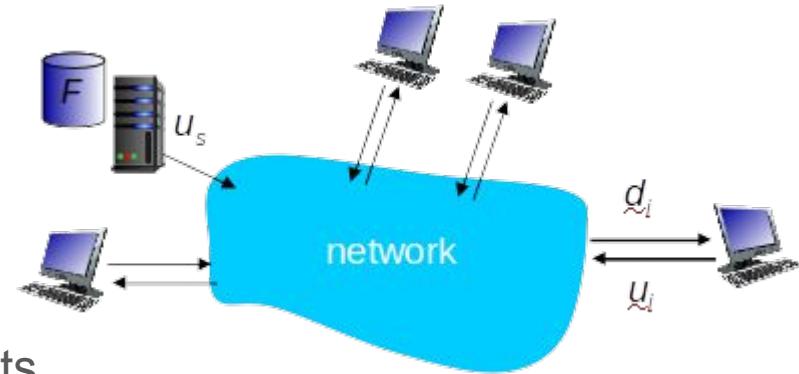
*time to distribute F
to N clients using
client-server approach*

$$D_{cs} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time (D_{cs}): P2P

- **server transmission:** must upload at least one copy
 - time to send one copy: F/u_s



- **client:** each client must download file copy
 - min client download time: F/d_{min}

- **clients:** as aggregate must download **NF** bits
 - max upload rate (limiting max download rate) is $u_s + \sum u_i$

time to distribute F to N clients using P2P approach

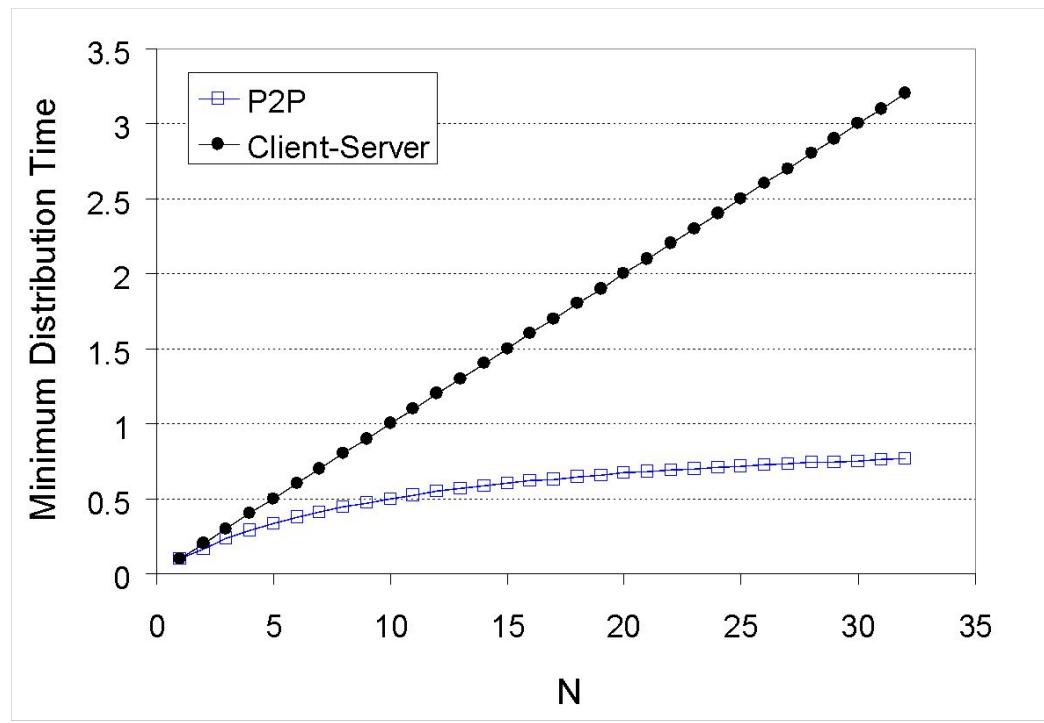
$$D_{P2P} > \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

increases linearly in N ...

... but so does this, as each peer brings service capacity

Client-server vs. P2P: example

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

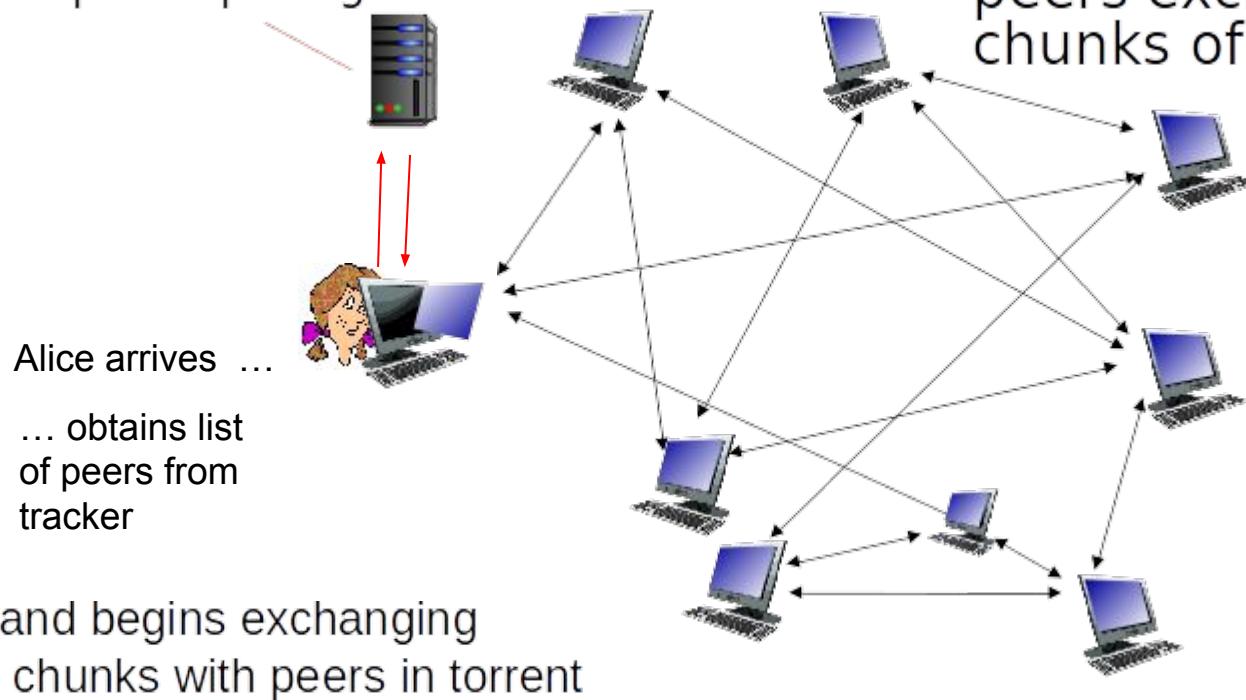


P2P file distribution: BitTorrent

- Torrent: the collection of all peers participating in the distribution of a particular file
- file divided into 256KBytes chunks
- peers in torrent send/receive file chunks

P2P file distribution: BitTorrent

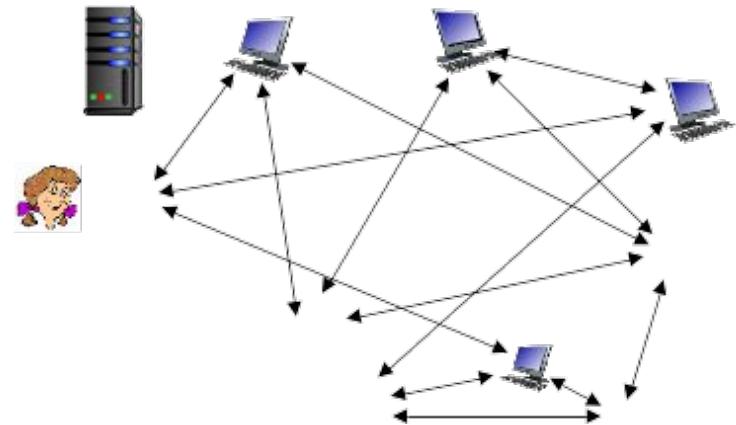
tracker: tracks peers participating in torrent



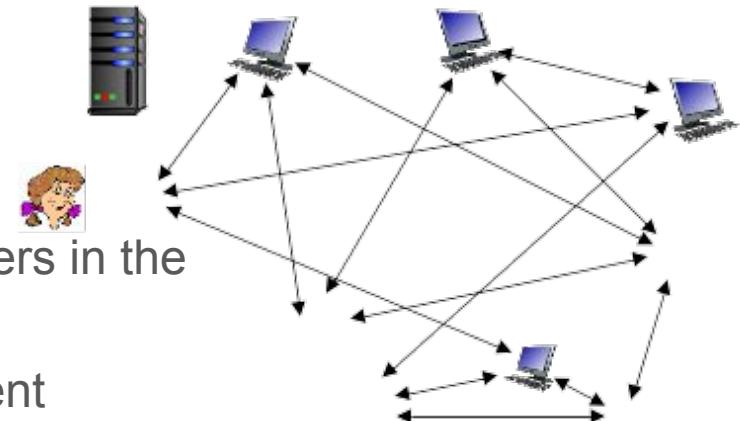
torrent: group of peers exchanging chunks of a file

P2P file distribution: BitTorrent

- peer joining torrent:
 - has no chunks, but will accumulate them over time from other peers
 - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- **churn**: peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent



P2P file distribution: BitTorrent



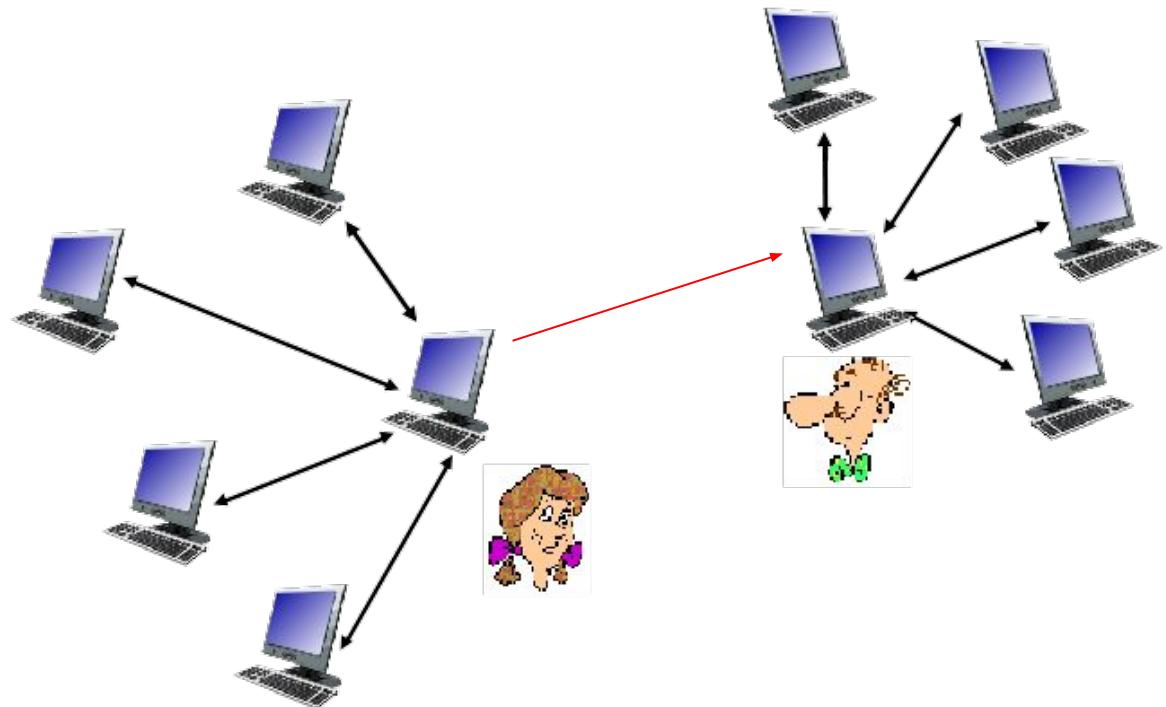
- Alice makes TCP connections with all the peers in the list
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- L: list of chunks from L neighbors
- Request algorithm
 - Which chunks should she request from her neighbors?
- Response algorithm
 - To which of her neighbors should she send requested chunks?

BitTorrent: requesting, sending file chunks

- requesting chunks:
 - Alice requests missing chunks from peers, **rarest first**
 - The chunk with fewest repeated copies among her neighbors
 - The rarest chunks get more quickly redistributed
 - roughly equalize the numbers of copies of each chunk in the torrent
- sending chunks: tit-for-tat
 - Alice sends chunks to those four peers currently sending her chunks **at highest rate**
 - other peers are choked by Alice (do not receive chunks from her)
 - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 - newly chosen peer may join top 4
 - allows new peers to get chunks, so they have something to trade

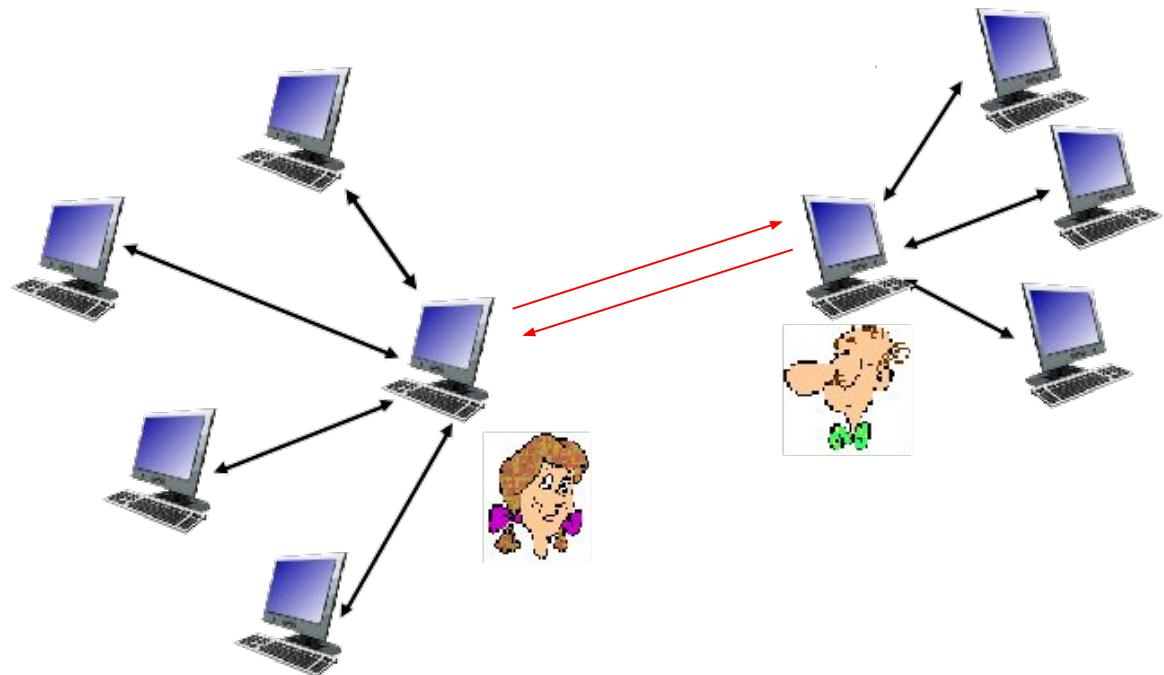
BitTorrent: tit-for-tat

1. Alice “optimistically unchoke” Bob



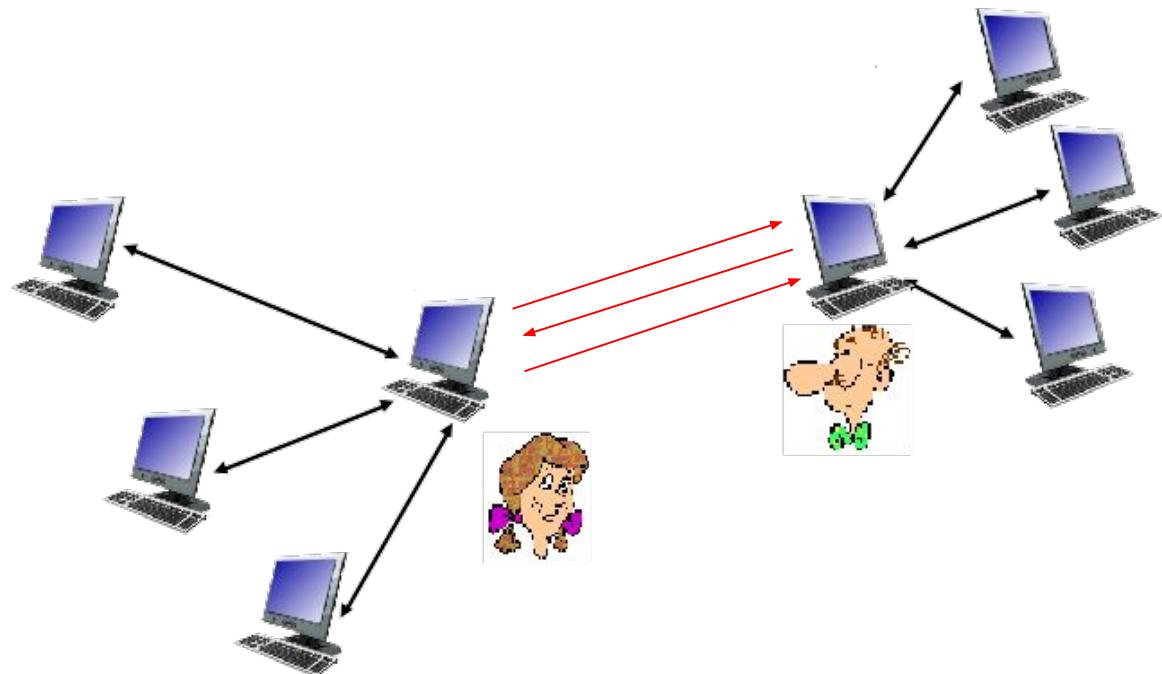
BitTorrent: tit-for-tat

1. Alice “optimistically unchoke” Bob
2. Alice may become one of Bob’s top-four providers; Bob reciprocates



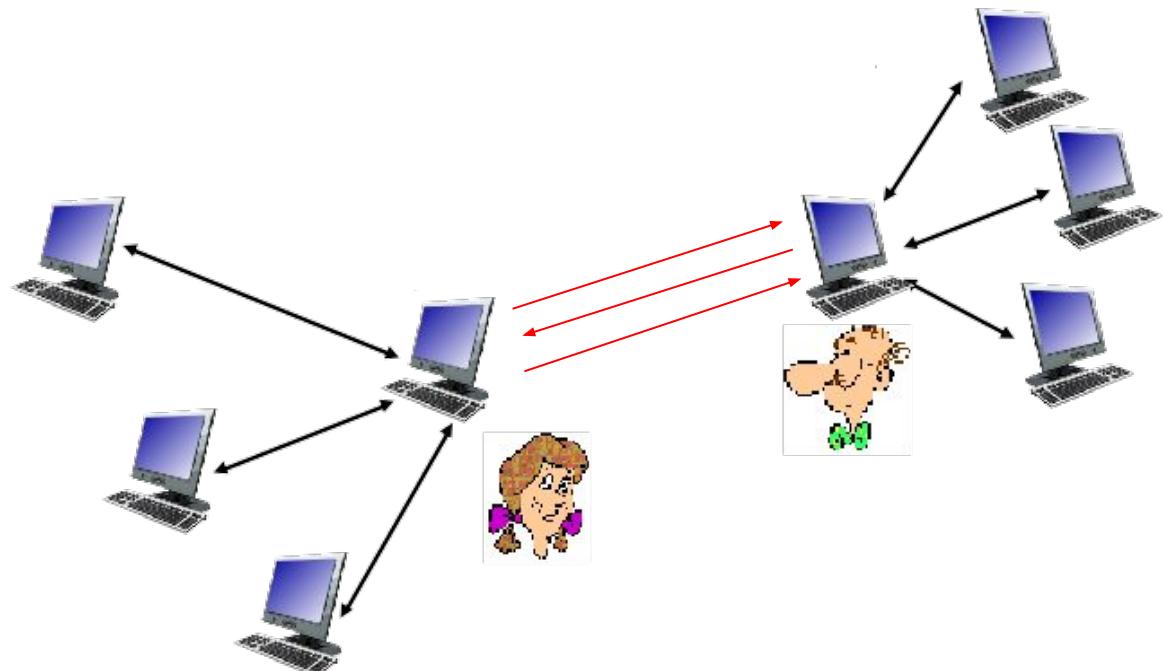
BitTorrent: tit-for-tat

1. Alice “optimistically unchoke” Bob
2. Alice may become one of Bob’s top-four providers; Bob reciprocates
3. Bob may become one of Alice’s top-four providers



BitTorrent: tit-for-tat

1. Alice “optimistically unchoke” Bob
2. Alice may become one of Bob’s top-four providers; Bob reciprocates
3. Bob may become one of Alice’s top-four providers



higher upload rate: find better trading partners, get file faster !

Question

Bob joins a BitTorrent torrent, but he does not want to upload any data to any other peer (free-riding). Can he receive a complete copy of the file that is shared by the torrent

True

Flase

Outline

- principles of network applications
- socket programming with UDP and TCP
- Web and HTTP
- electronic mail
 - SMTP, POP3, IMAP
- DNS
- P2P applications
- **video streaming and content distribution networks**

Video Streaming and CDNs: context

- Stream video traffic: major consumer of Internet bandwidth
 - Netflix, YouTube, Amazon Prime:
 - 80% of residential ISP traffic (2020)
 - ~2B YouTube users, ~193M Netflix users
- challenge: scale - how to reach ~2B users?
 - single mega-video server won't work (why?)
- challenge: heterogeneity
 - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
 - packet loss, delay due to congestion result in poor video quality
- solution: distributed application-level infrastructure



Multimedia: video

- video: sequence of images displayed at constant rate
 - e.g., 24 images/sec
- digital image: array of pixels
 - each pixel represented by bits
- coding: use redundancy **within** and **between** images to decrease # bits used to encode image
 - spatial (within image)
 - temporal (from one image to next)



frame *i*

spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i

frame *i+1*

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- Examples:
 - MPEG 1 (CD-ROM) 1.5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4 (often used in Internet, 64Kbps – 12 Mbps)

spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (purple) and number of repeated values (N)



frame i



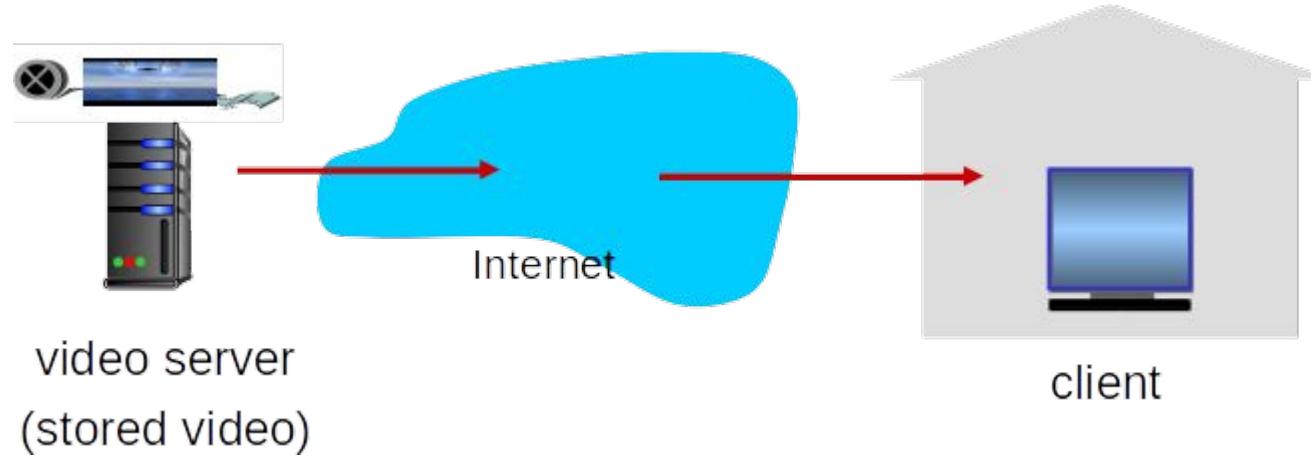
temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i

frame $i+1$

Multimedia: video

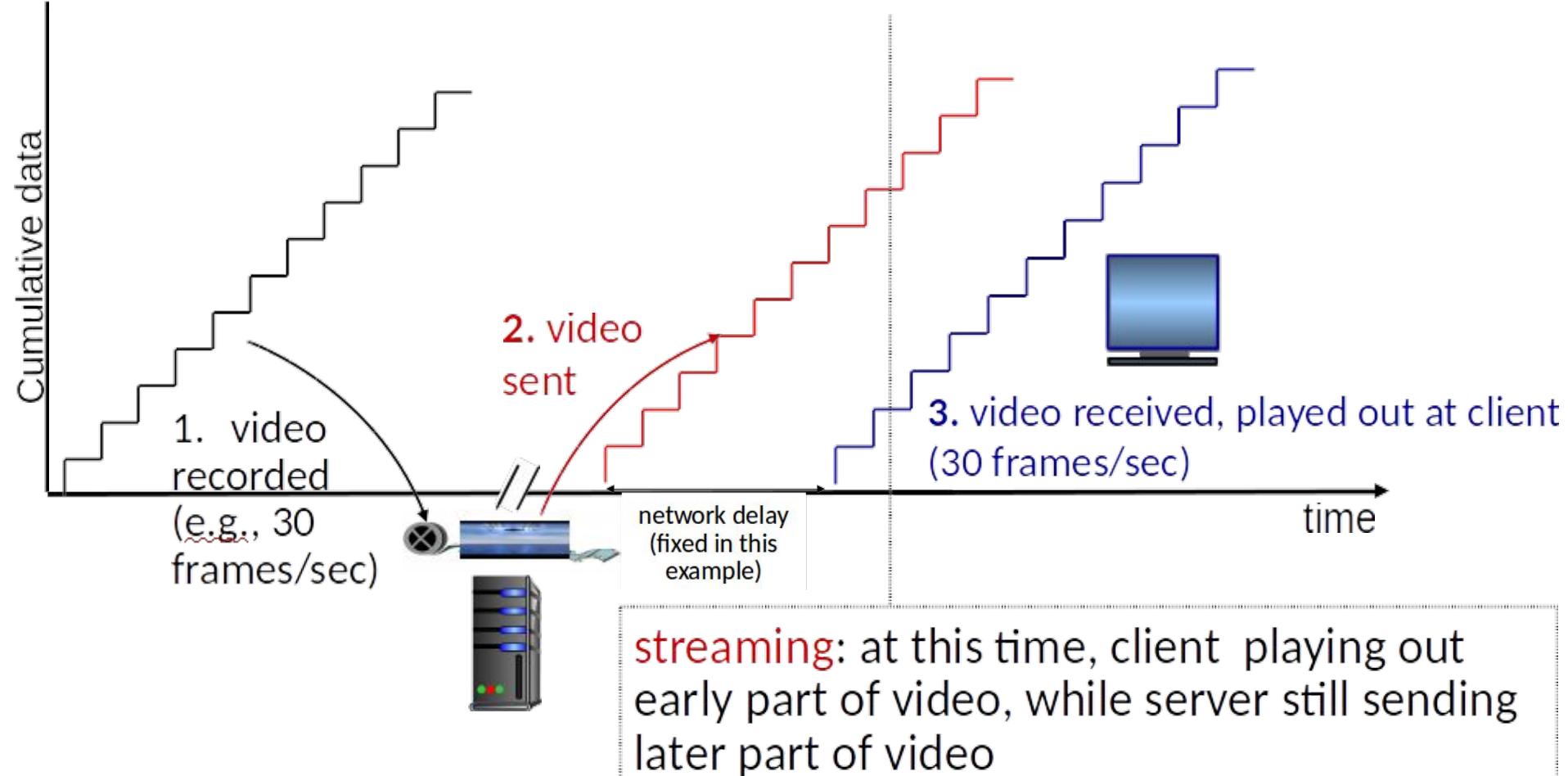
- Bit rate of a video
 - Compressed Internet video
 - 100 kbps: low-quality video
 - Over 3 Mbps for streaming high-definition movie
 - 4K streaming...10Mbps
- This can translate to huge amount of traffic and storage
 - A single 2 Mbps video with a duration of 67 minutes..... 1 gigabyte of data

Streaming multimedia: simple scenario



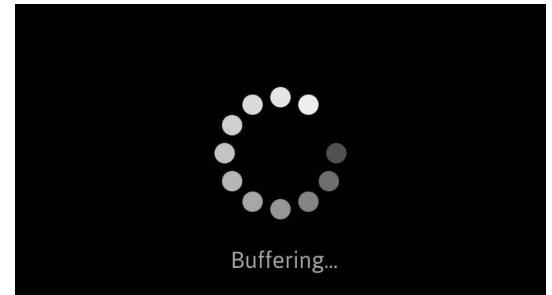
- Main challenges:
 - packet loss, delay due to congestion will delay playout, or result in poor video quality
 - server-to-client bandwidth will **vary** over time, with changing network congestion levels (in house, access network, network core, video server)

Streaming stored video

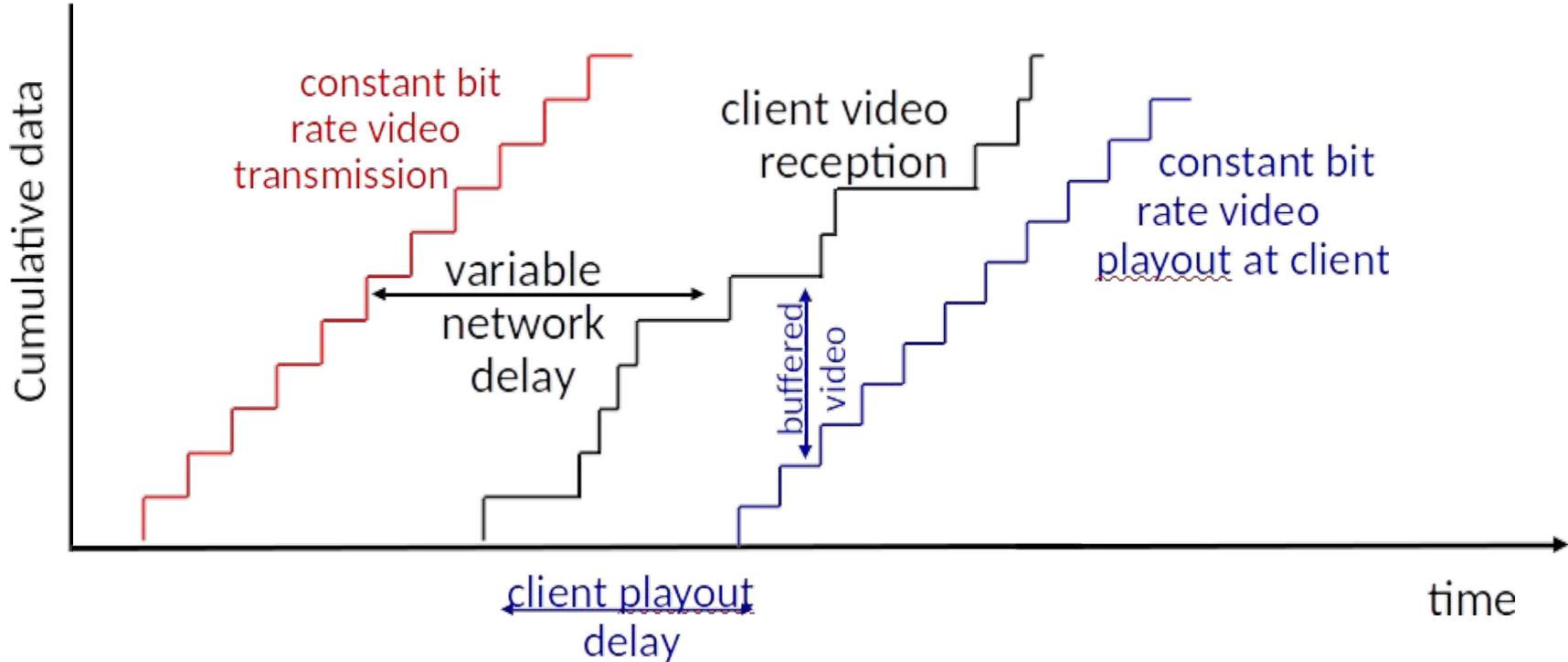


Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
 - ... but **network delays are variable** (jitter), so will need **client-side buffer** to match continuous playout constraint
- other challenges:
 - client interactivity: pause, fast-forward, rewind, jump through video
 - video packets may be lost, retransmitted

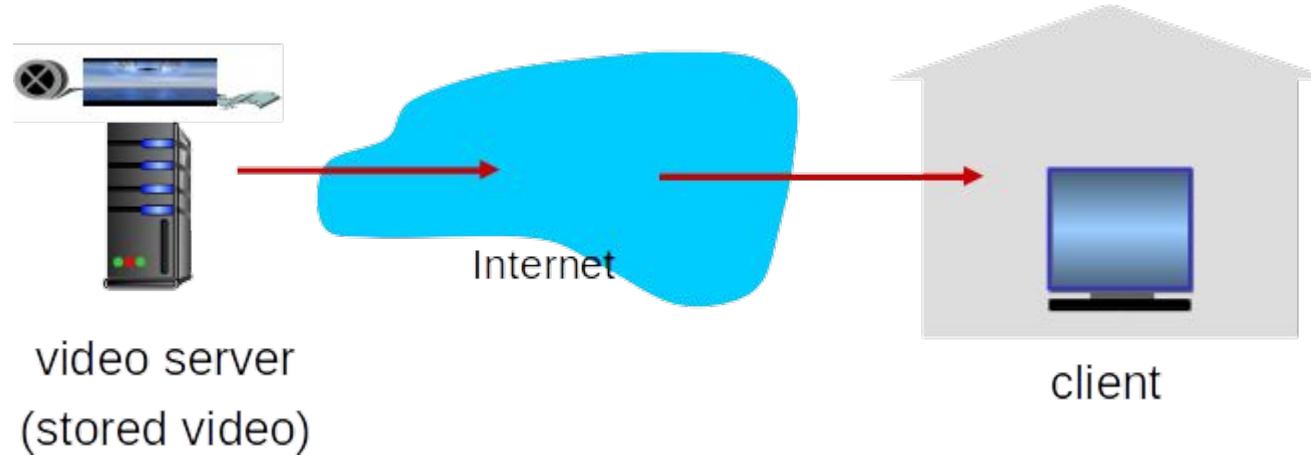


Streaming stored video: playout buffering



client-side buffering and playout delay: compensate for network-added delay, delay jitter

Streaming multimedia: simple scenario



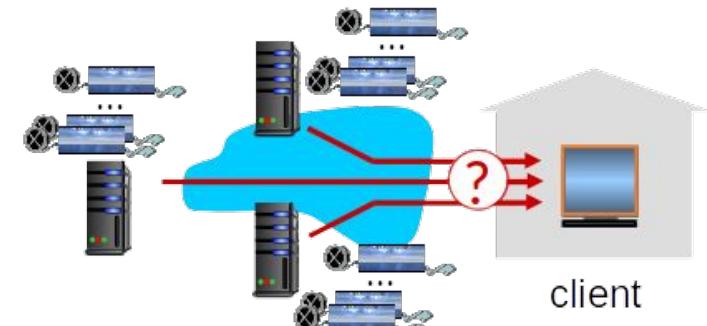
- Main challenges:
 - packet loss, delay due to congestion will delay playout, or result in poor video quality
 - server-to-client bandwidth will **vary** over time, with changing network congestion levels (in house, access network, network core, video server)

Streaming multimedia: simple scenario

- The video is stored at an HTTP server as an ordinary file with a specific URL
 - user send and HTTP GET request for the URL
 - server sends the file within an HTTP response
 - Problem:
 - all clients receive the same encoding of the video
 - Large variation in the amount of bandwidth available to a client
 - across different client
 - over time for the same client
 - Solutions
 - use compression to create multiple version of the same video
 - each version having a different quality level
 - three versions of the same video, at rates of 300 kbps, 1 Mbps, and 3 Mbps
 - users can then decide which version they want to watch
 - DASH: Dynamic, Adaptive Streaming over H T T P

Streaming multimedia: DASH

- **DASH: Dynamic, Adaptive Streaming over HTTP**
- Server:
 - divides video file into multiple chunks
 - each chunk encoded at multiple different rates
 - different rate encodings stored in different files
 - files replicated in various CDN nodes
 - **manifest file**: provides URLs for different chunks
- Client:
 - periodically estimates server-to-client bandwidth
 - consulting manifest, requests one chunk at a time
 - chooses maximum coding rate sustainable given current bandwidth
 - can choose different coding rates at different points in time (depending on available bandwidth at time)



Streaming multimedia: DASH

- DASH: Dynamic, Adaptive Streaming over HTTP
- “intelligence” at client: client determines
 - **when** to request chunk (so that buffer starvation, or overflow does not occur)
 - **what encoding** rate to request (higher quality when more bandwidth available)
 - **where** to request chunk (can request from URL server that is “close” to client or has high available bandwidth)

Streaming video = encoding + DASH + playout buffering

Content distribution networks

- **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?
- **option 1:** single, large “mega-server”
 - single point of failure
 - point of network congestion
 - long path to distant clients
 - multiple copies of video sent over outgoing link

....quite simply: this solution doesn't scale

Content distribution networks

- **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?
- **option 2:** store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
 - manages servers in multiple geographically distributed locations
 - Stores copies of the videos (images, audio) in its servers
 - And direct each request to a CDN location that will provide the best user experience
 - **Private CDN**
 - **Third-party CDN**

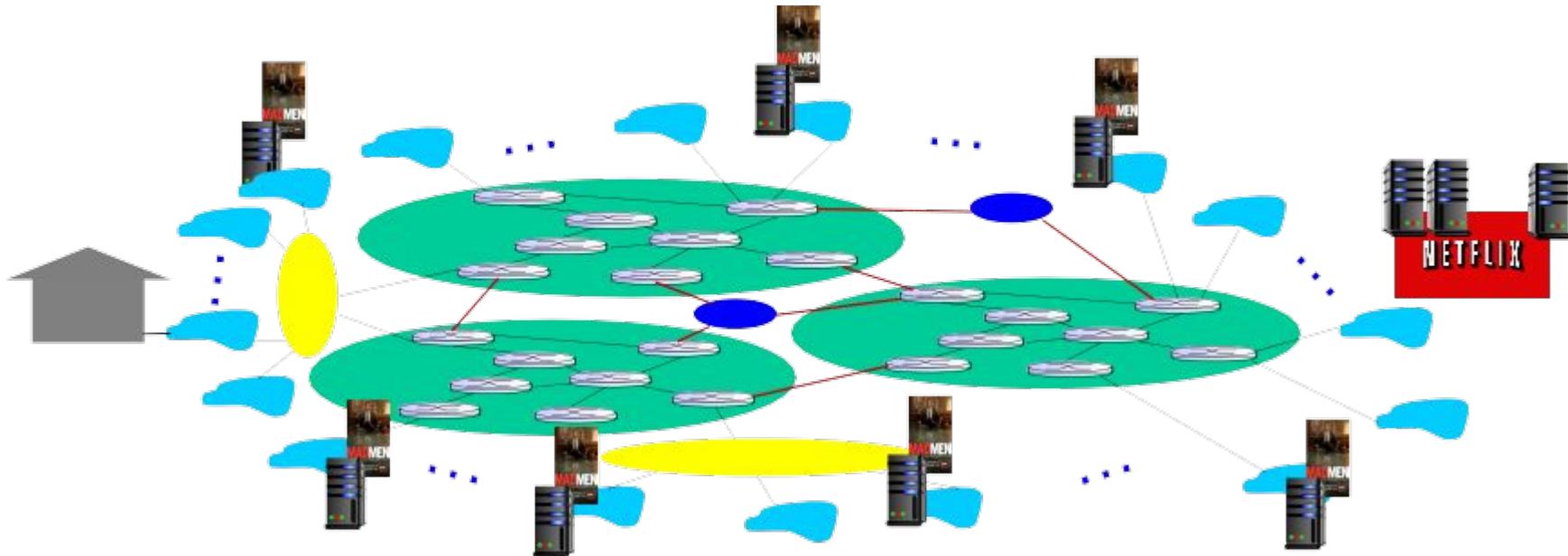
Content Distribution Networks (CDNs)

- Server Placement Philosophy
 - **enter deep:** push CDN servers deep into many access networks
 - close to users
 - used by Akamai, 1700 locations
 - **bring home:** smaller number (10's) of larger clusters in IXPs
 - used by Limelight

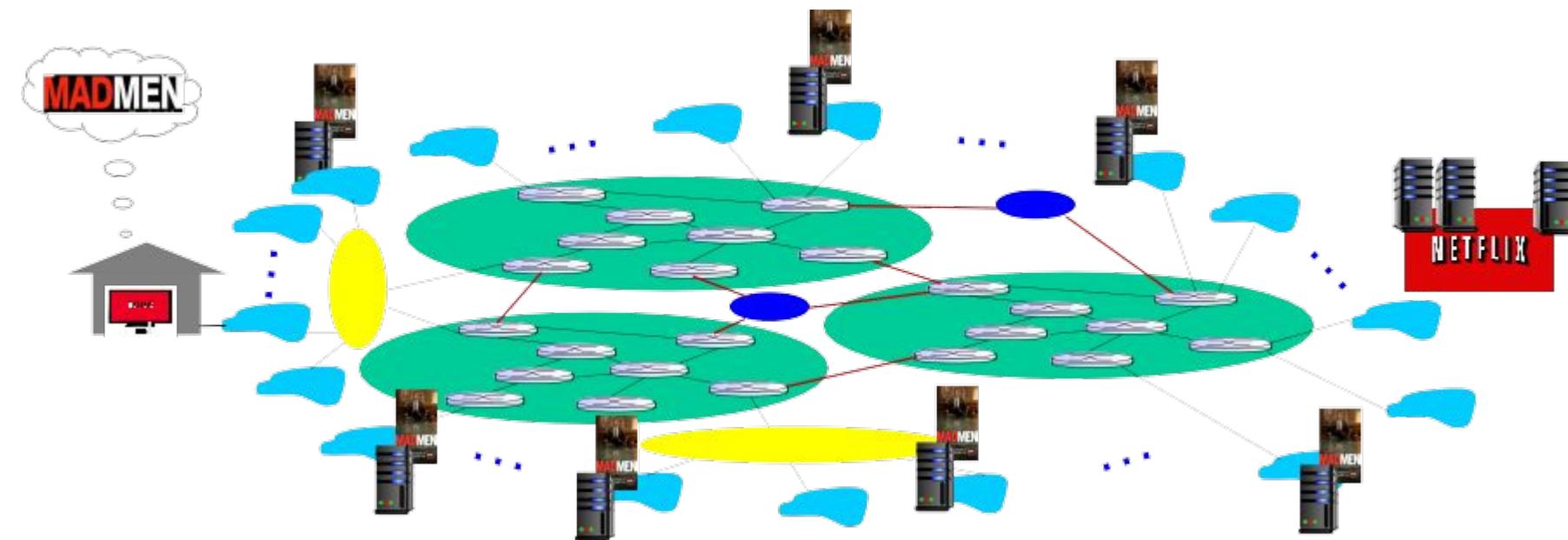
Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
 - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
 - service provider returns manifest
 - using manifest, client retrieves content at highest supportable rate
 - directed to nearby copy, retrieves content
 - may choose different copy if network path congested

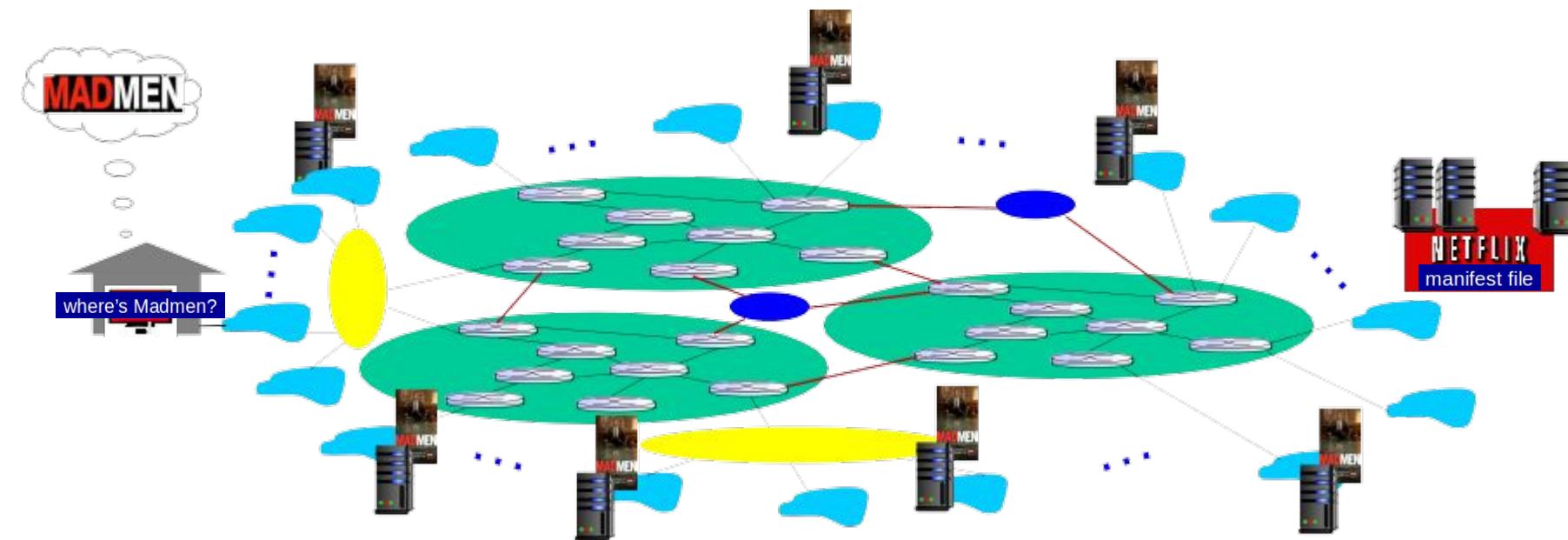
Content Distribution Networks (CDNs)



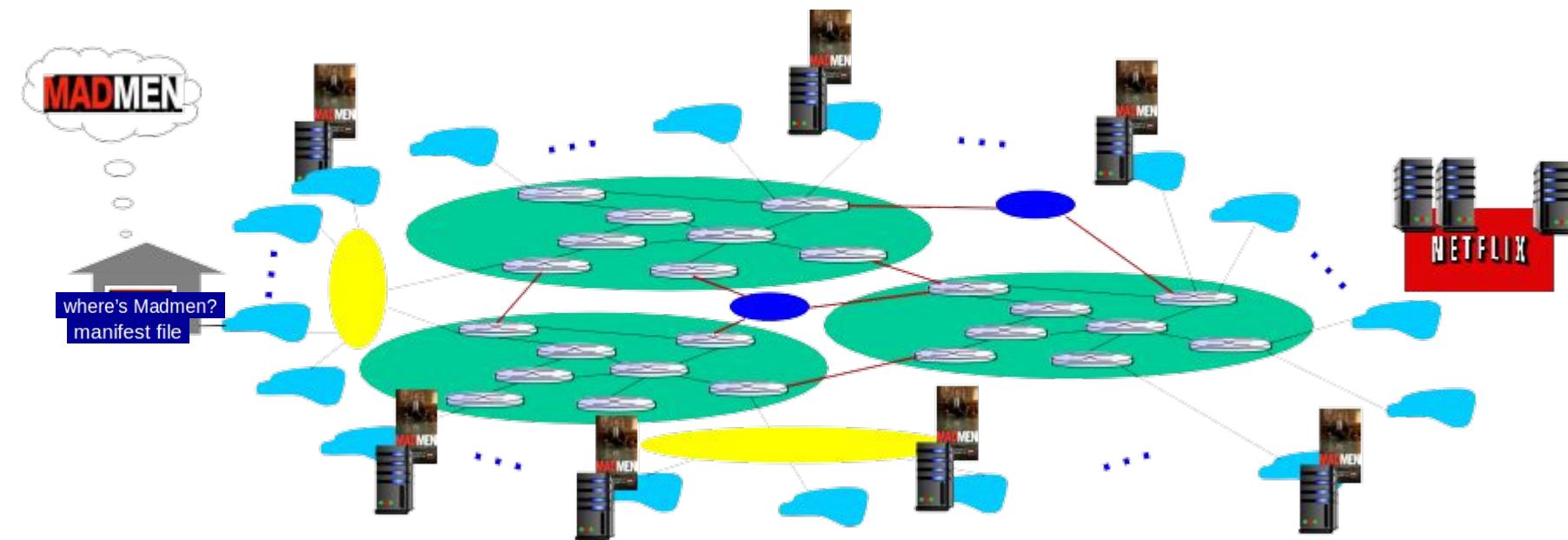
Content Distribution Networks (CDNs)



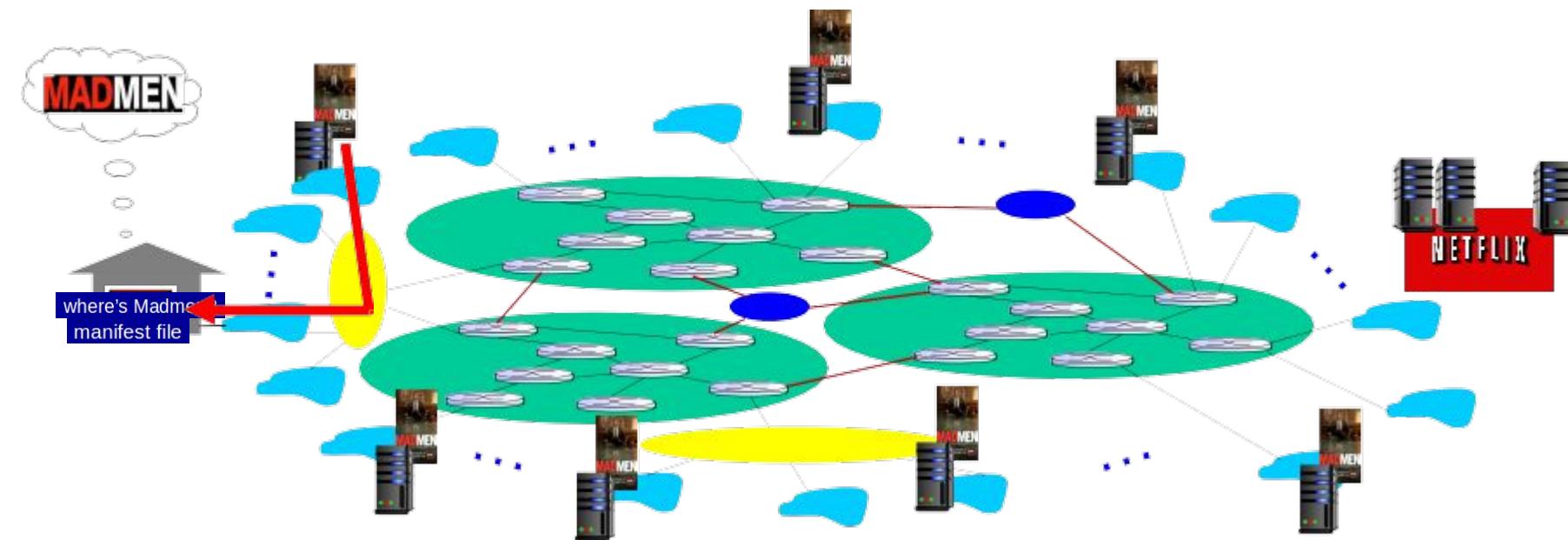
Content Distribution Networks (CDNs)



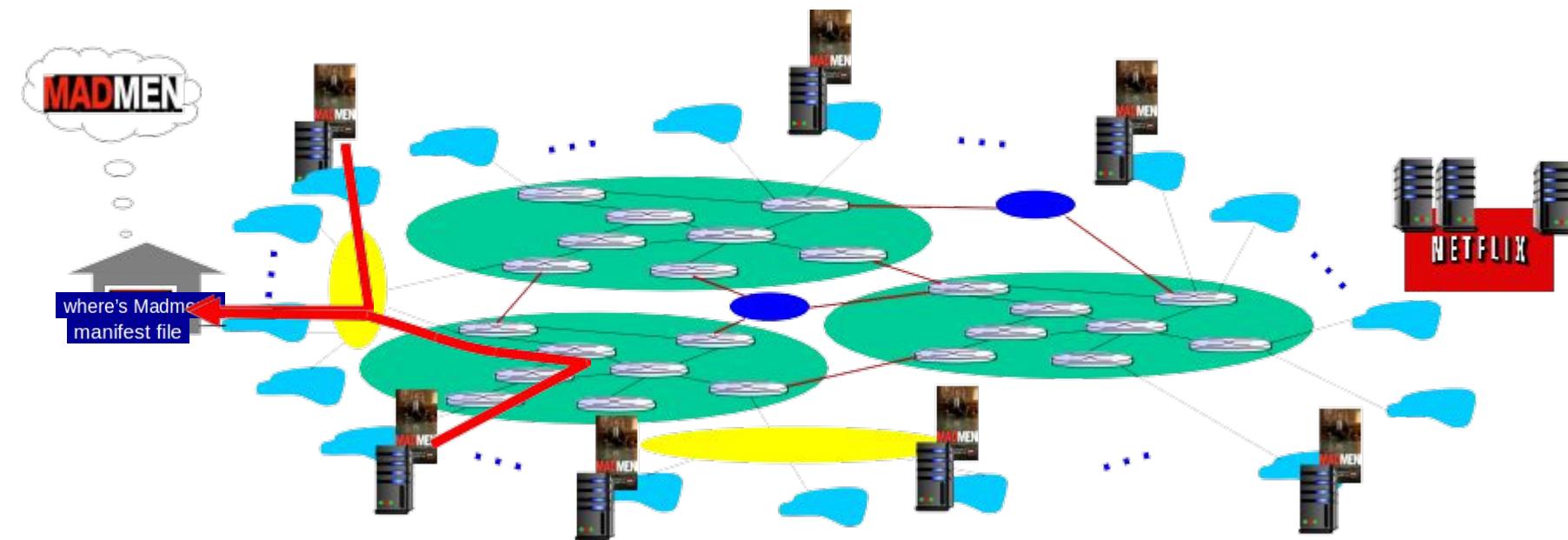
Content Distribution Networks (CDNs)



Content Distribution Networks (CDNs)

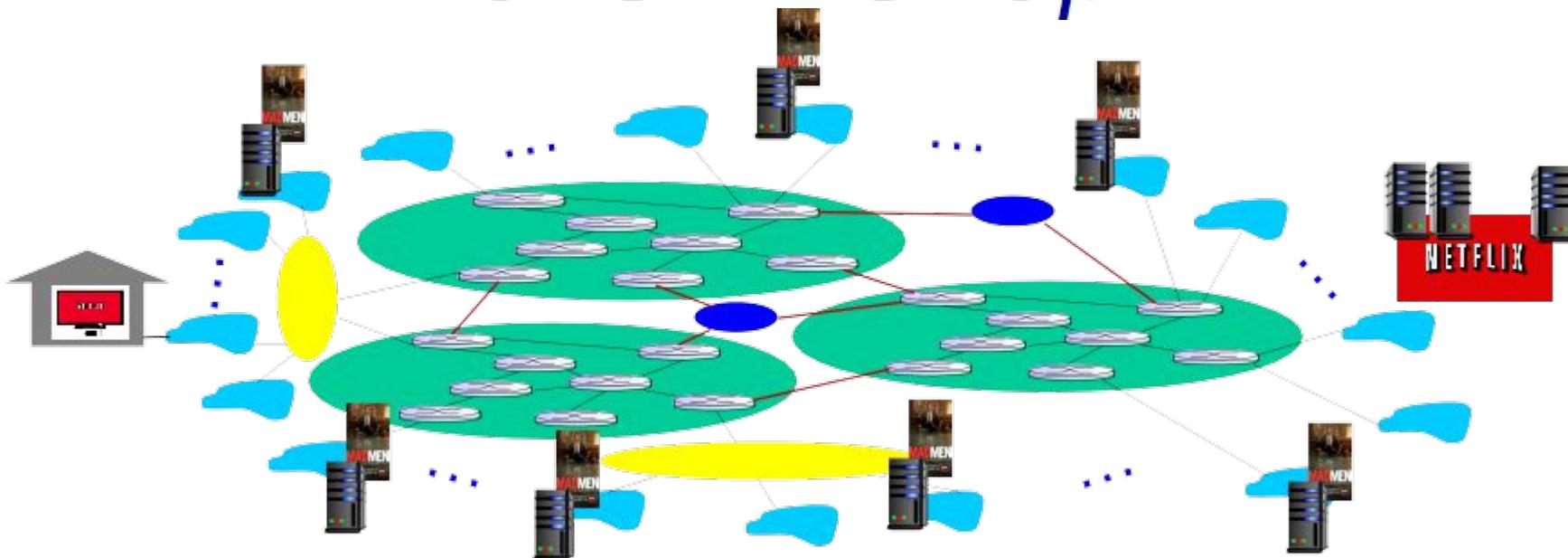


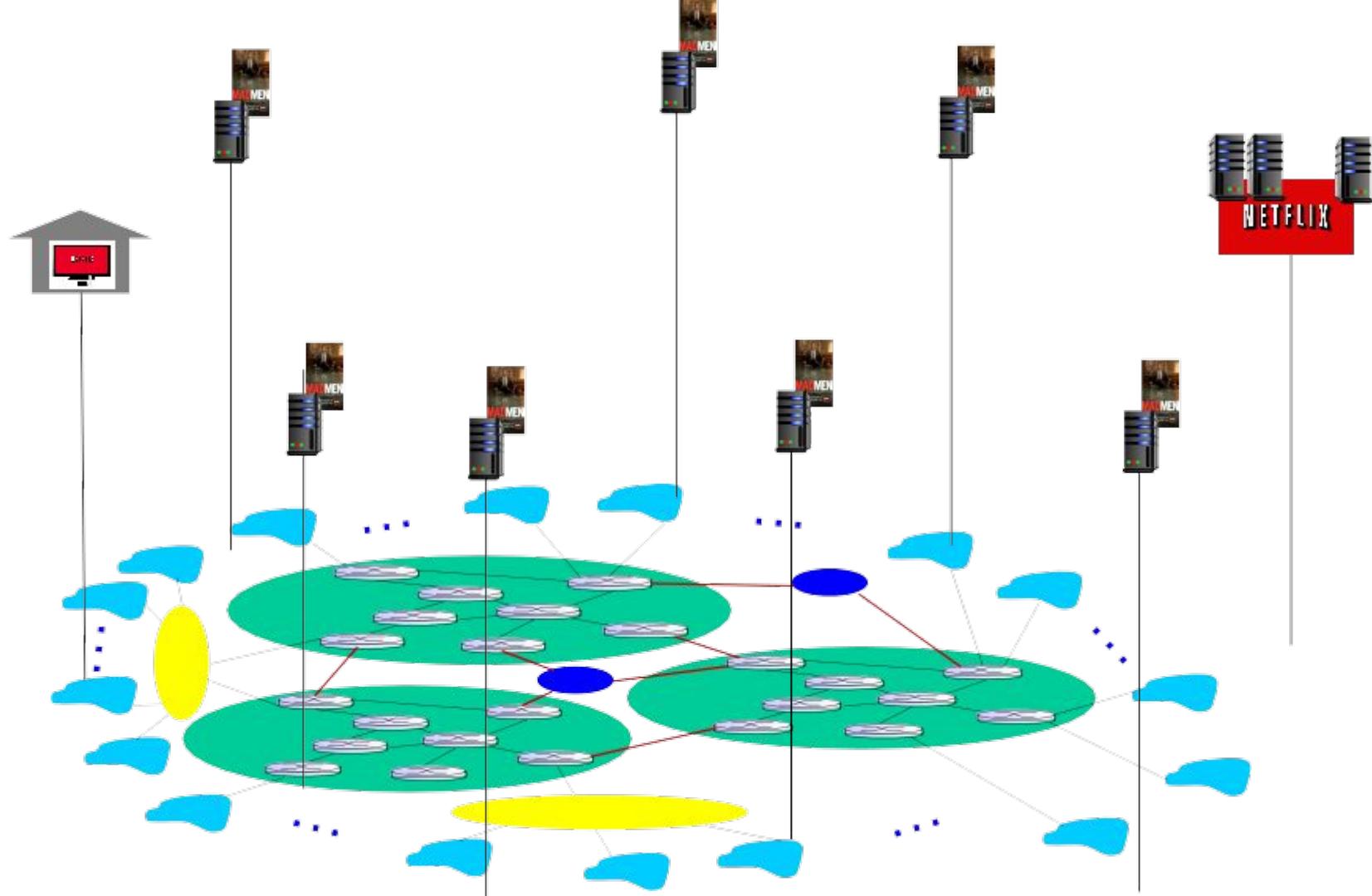
Content Distribution Networks (CDNs)

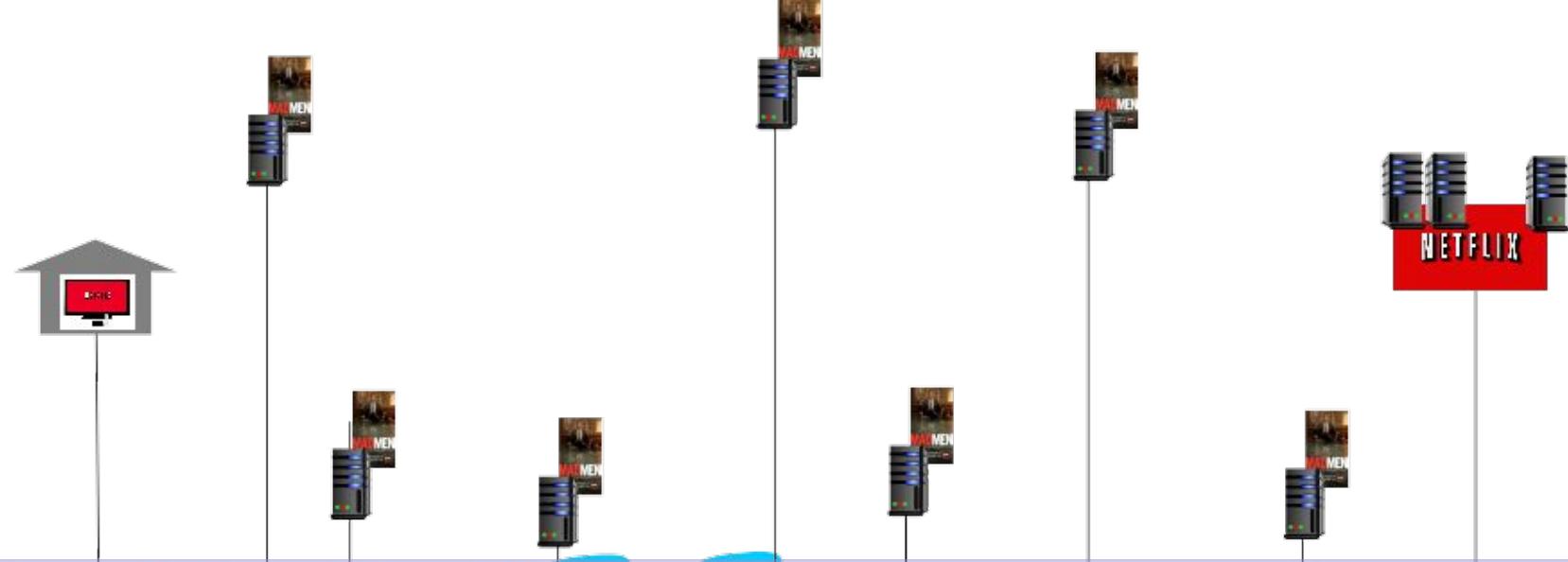


Content Distribution Networks (CDNs)

“over the top”







Internet host-host communication as a service

OTT challenges: coping with a congested Internet

- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

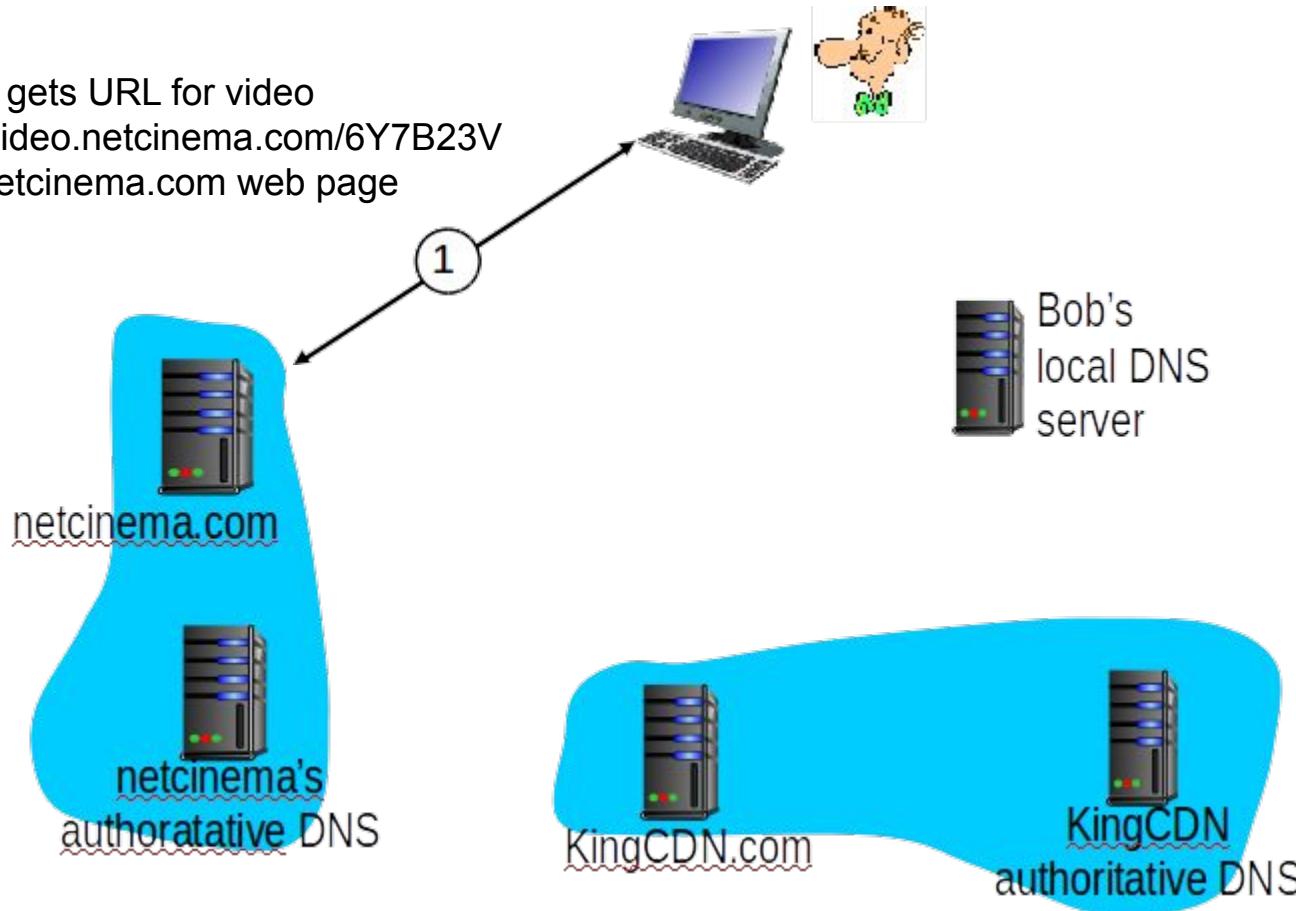
CDN content access: a closer look

- How does CDN operate?
 - Determine a suitable CDN server cluster for that client at that time
 - Redirect the client's request to a server in that cluster
- Solution:
 - Using DNS
 - Just specify CDN in the manifest file
- Example: DNS is used to redirect the client's request
 - Bob (client) requests video <http://netcinema.com/>

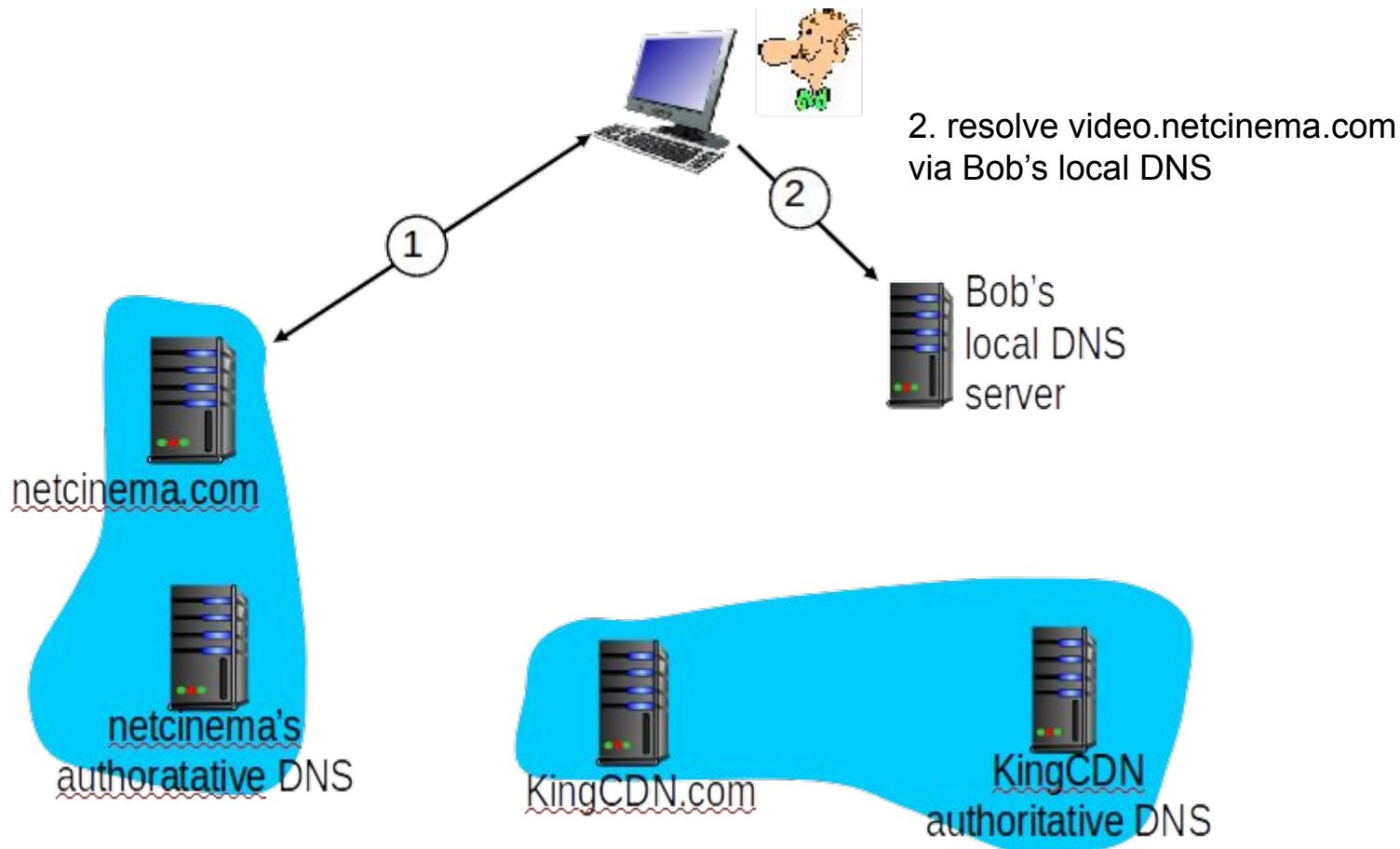
CDN content access: a closer look

1. Bob gets URL for video

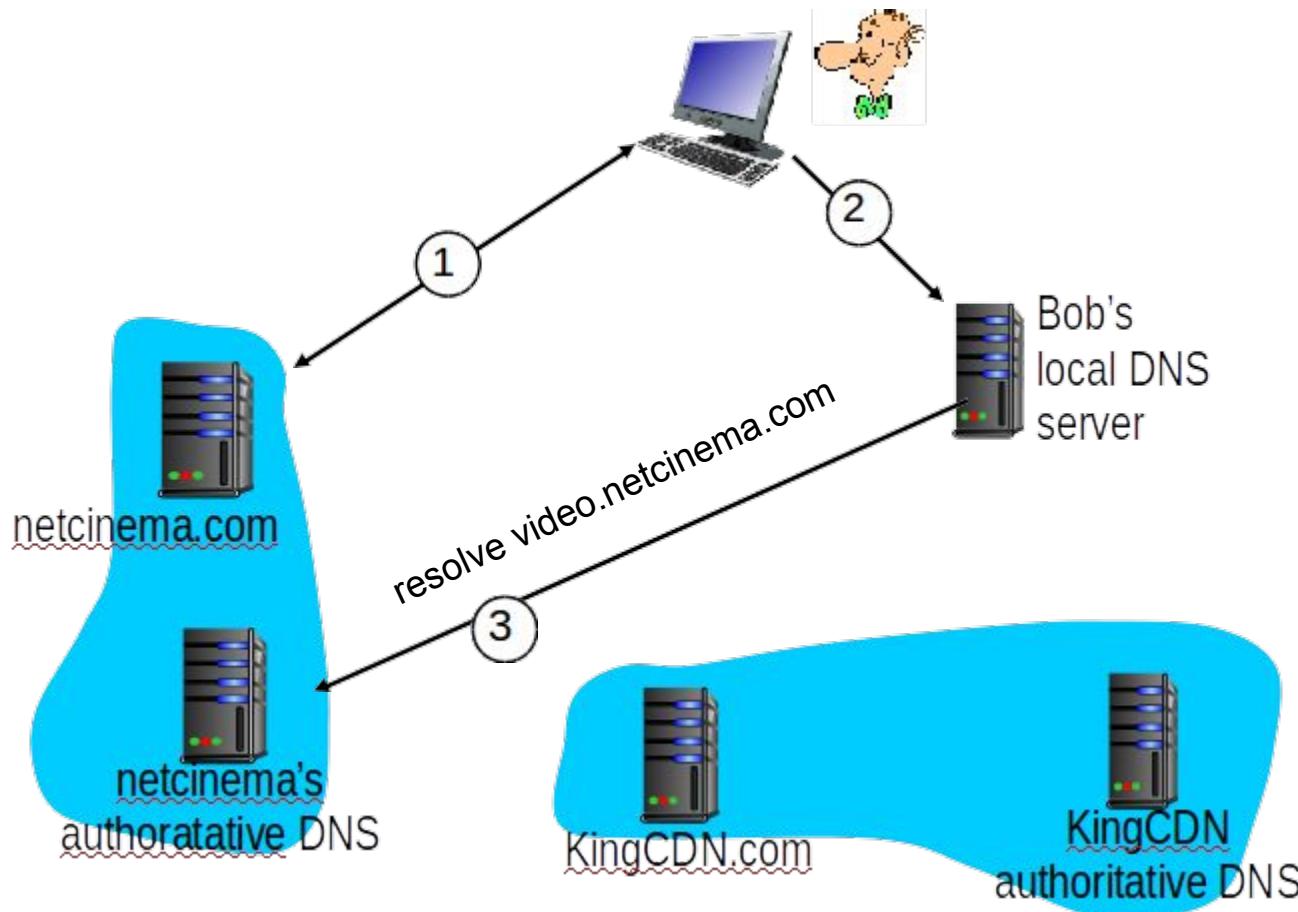
<http://video.netcinema.com/6Y7B23V>
from netcinema.com web page



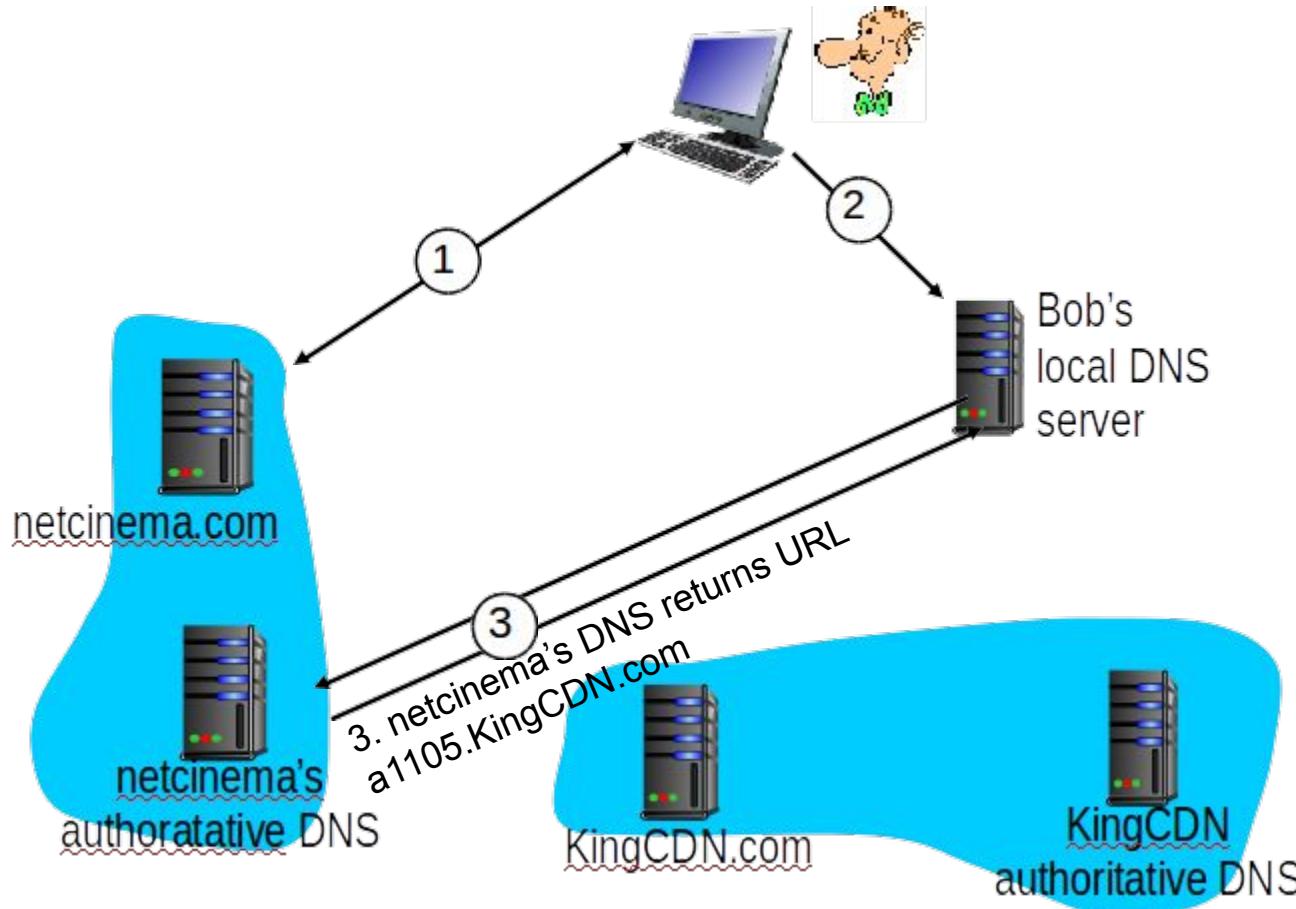
CDN content access: a closer look



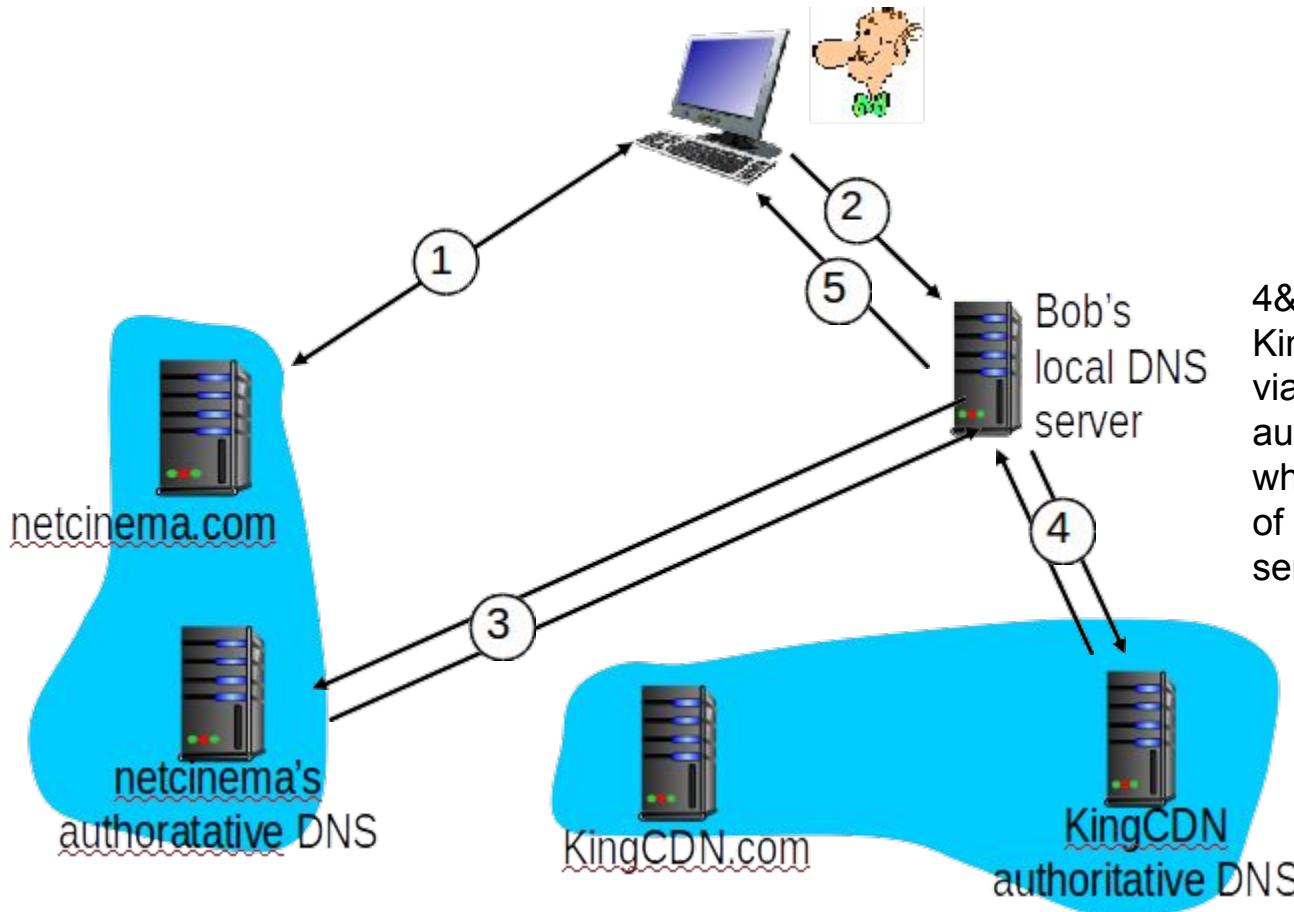
CDN content access: a closer look



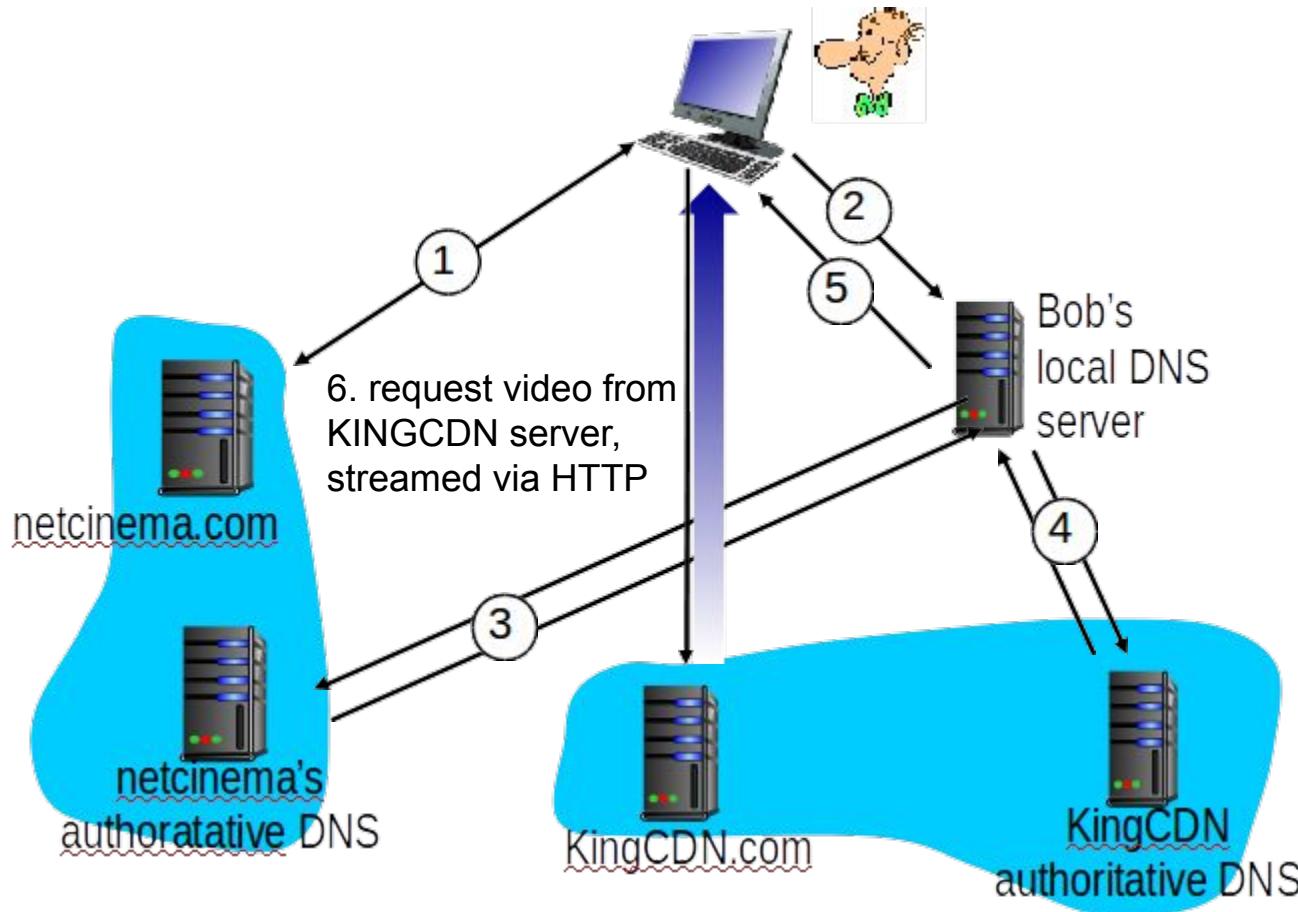
CDN content access: a closer look



CDN content access: a closer look



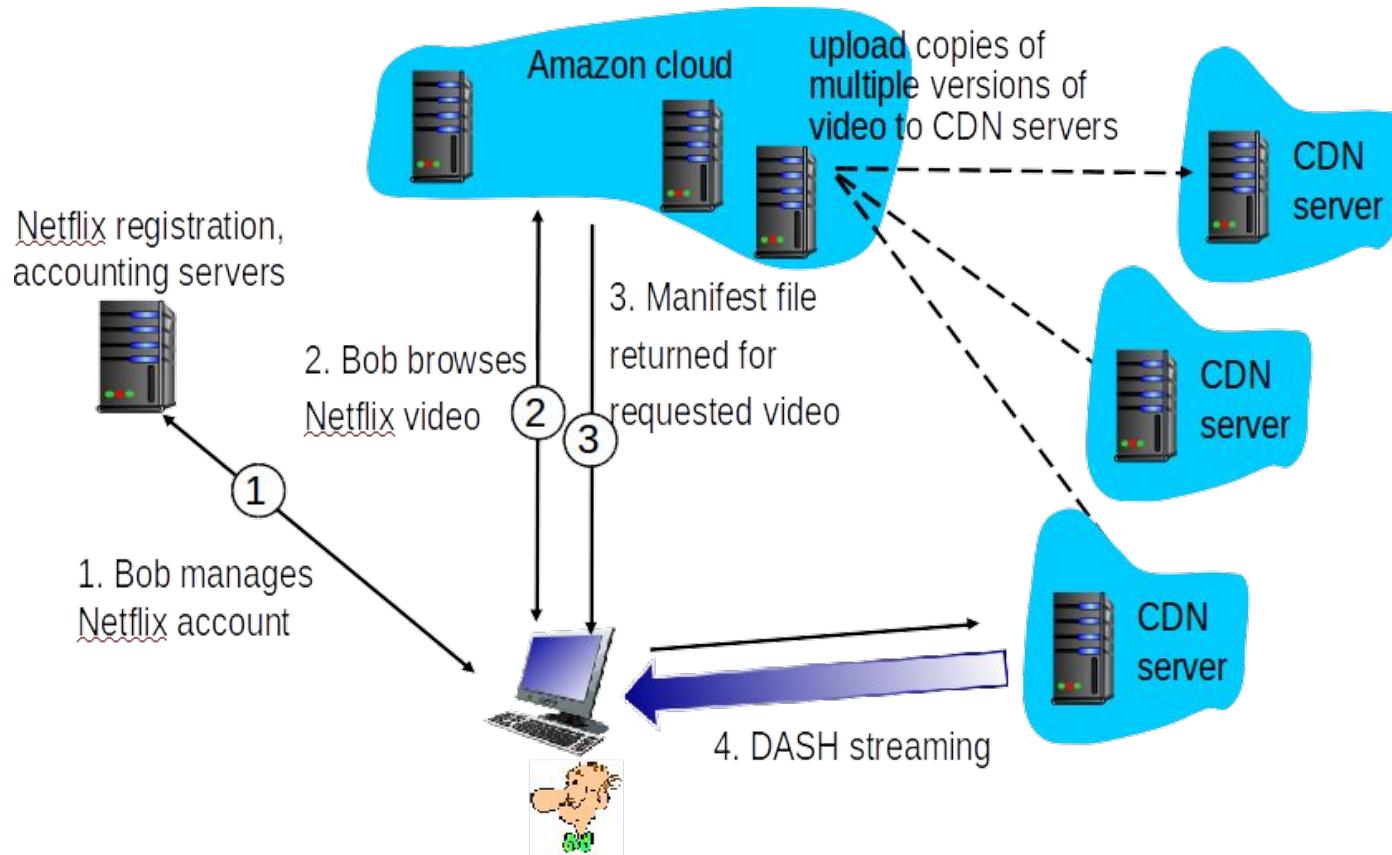
CDN content access: a closer look



CDN Content Access: Cluster selection strategies

- CDN employ proprietary cluster selection
- **geographically closest**
 - select a cluster based on the IP address of local DNS
 - Pros: choose the closest cluster; good for large fraction of the clients
 - Cons
 - may not be the closest cluster in terms of number of hops
 - some end-users use remotely located local DNSs
 - always assigning the same cluster to a particular client
 - ignores the variation in delay and available bandwidth over time of Internet paths,
- **determine cluster based on the current traffic**
 - based on periodic real-time measurement of delay and loss between clusters and clients
 - each cluster of a CDN sending probes to all of the LDNS around the world

Case study: Netflix



Application Layer: Summary

- application architectures
 - Client-server
 - P2P
- application service requirements:
 - reliability, bandwidth, delay
- Internet transport service model
 - connection-oriented, reliable: TCP
 - unreliable, datagrams: UDP
- specific protocols: HTTP, SMTP, POP, IMAP, DNS, P2P: BitTorrent
- video streaming, CDNs
- socket programming: TCP, UDP sockets

Application Layer: Summary

- most importantly: learned about protocols!
 - typical request/reply message exchange:
 - client requests info or service
 - server responds with data, status code
- message formats:
 - headers: fields giving info about data
 - data: info(payload) being communicated
- important themes:
 - centralized vs. decentralized
 - stateless vs. stateful
 - scalability
 - reliable vs. unreliable message transfer
 - “complexity at network edge”

Transport Layer

Transport Layer

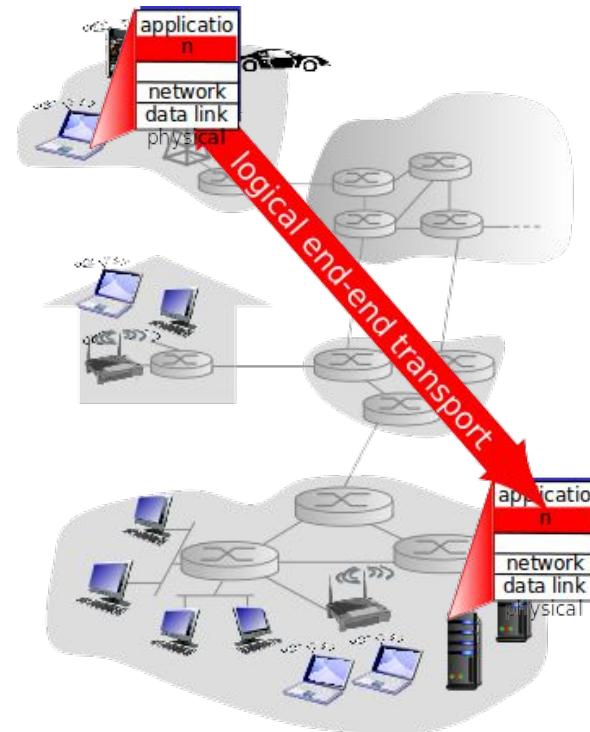
- our goals:
 - understand principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- learn about Internet transport layer protocols:
 - UDP: connectionless transport
 - TCP: connection-oriented reliable transport
 - TCP congestion control

outline

- **transport-layer services**
- multiplexing and demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
- principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Transport services and protocols

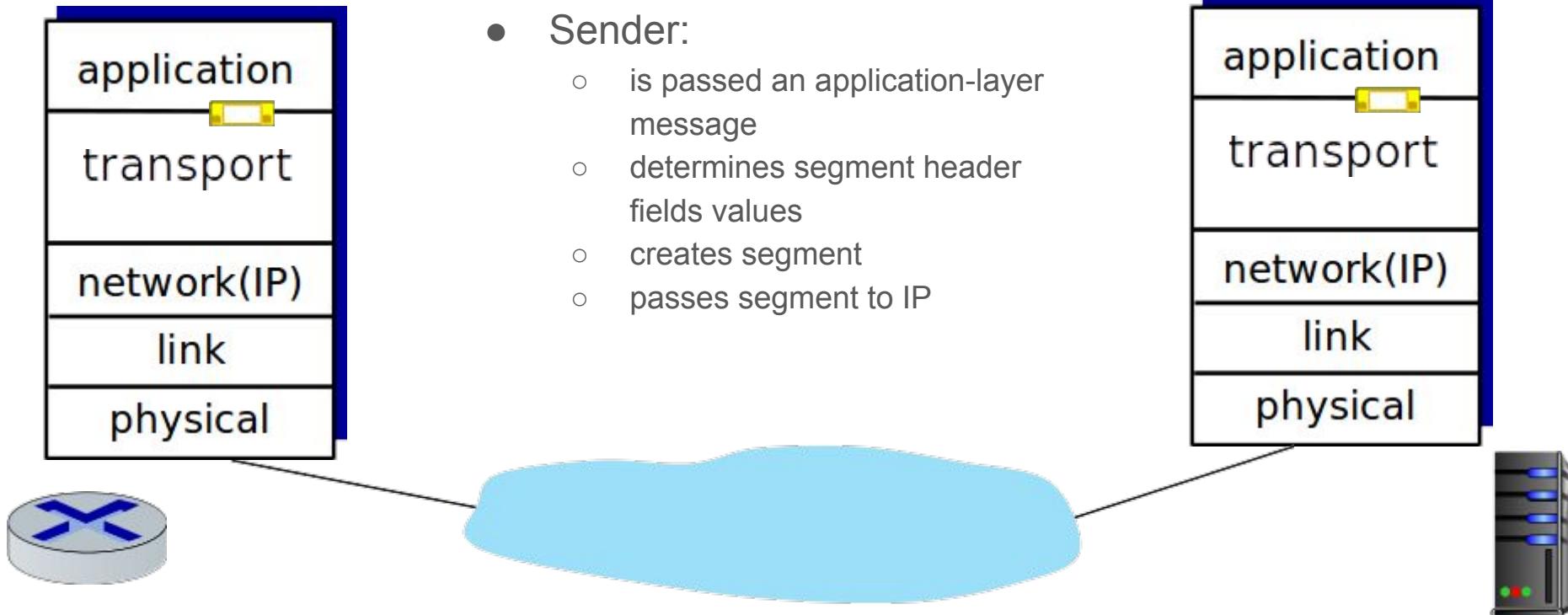
- provide **logical communication** between app processes running on different hosts
- transport protocols run in end systems
 - sender: breaks app messages into **segments**, passes to network layer
 - receiver: reassembles segments into messages, passes to app layer
- more than one transport protocol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



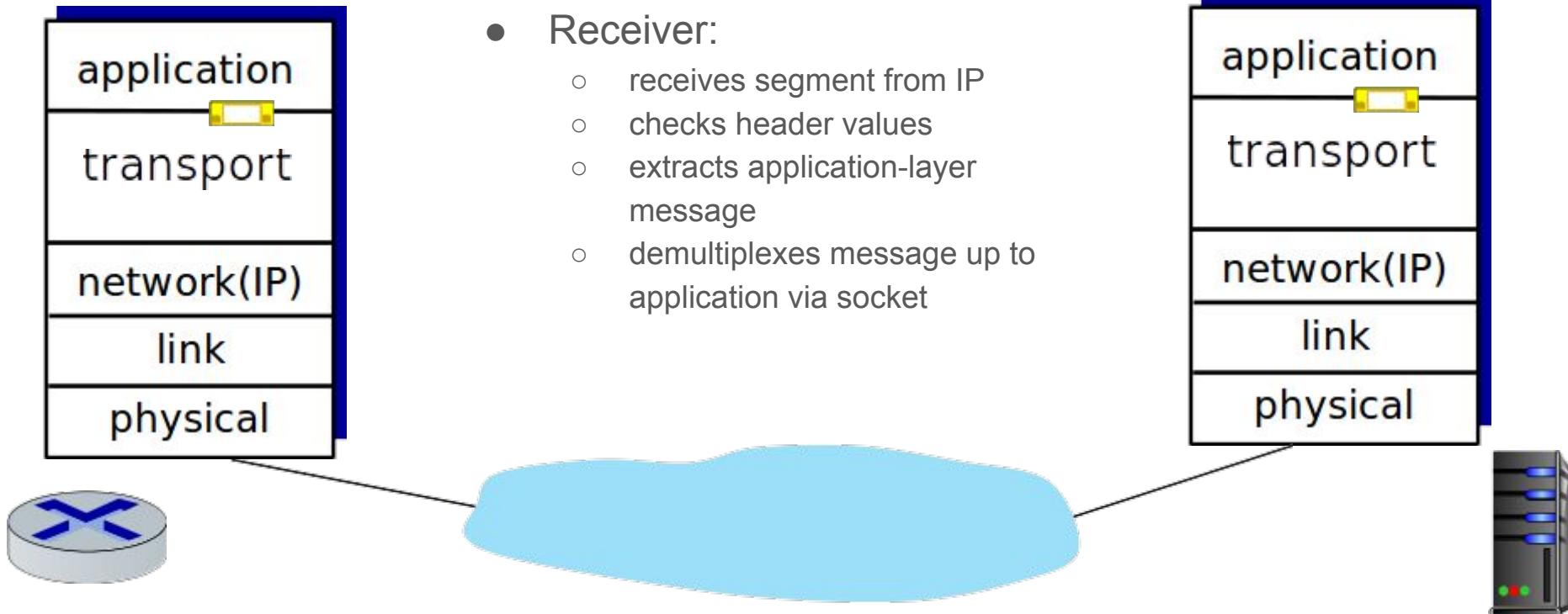
Transport vs. network layer services and protocols

- **network layer:** logical communication between **hosts**
- **transport layer:** logical communication between **processes**
 - relies on, enhances, network layer services
- Household analogy
 - 12 kids in Ann's house sending letters to 12 kids in Bill's house:
 - processes = kids
 - app messages = letters in envelopes
 - transport protocol = Ann and Bill who demux to in-house siblings
 - hosts = houses
 - network-layer protocol = postal service

Transport Layer Actions



Transport Layer Actions



Internet transport-layer protocols

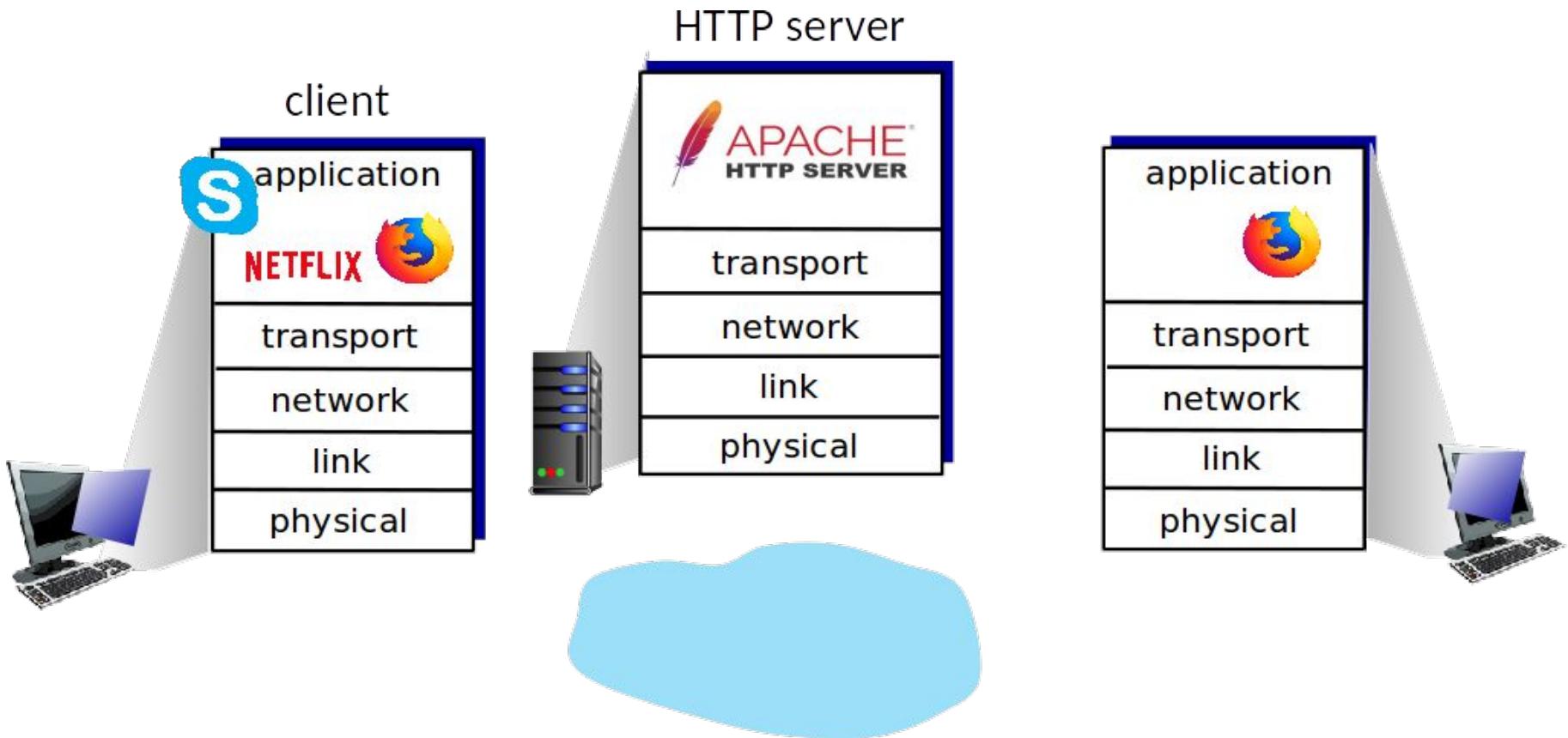
- IP provides logical communication between hosts
- IP service model:
 - best-effort delivery service
 - Unreliable service

Internet transport-layer protocols

- TCP: Transmission Control Protocol
 - reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup
- UDP: User Datagram Protocol
 - unreliable, unordered delivery: UDP
 - no-frills extension of “best-effort” IP
- services provided by both TCP and UDP
 - extending IP’s layer host-to-host delivery to process-to-process delivery
 - multiplexing and de-multiplexing
 - error checking
- services not available:
 - delay guarantees, bandwidth guarantees

outline

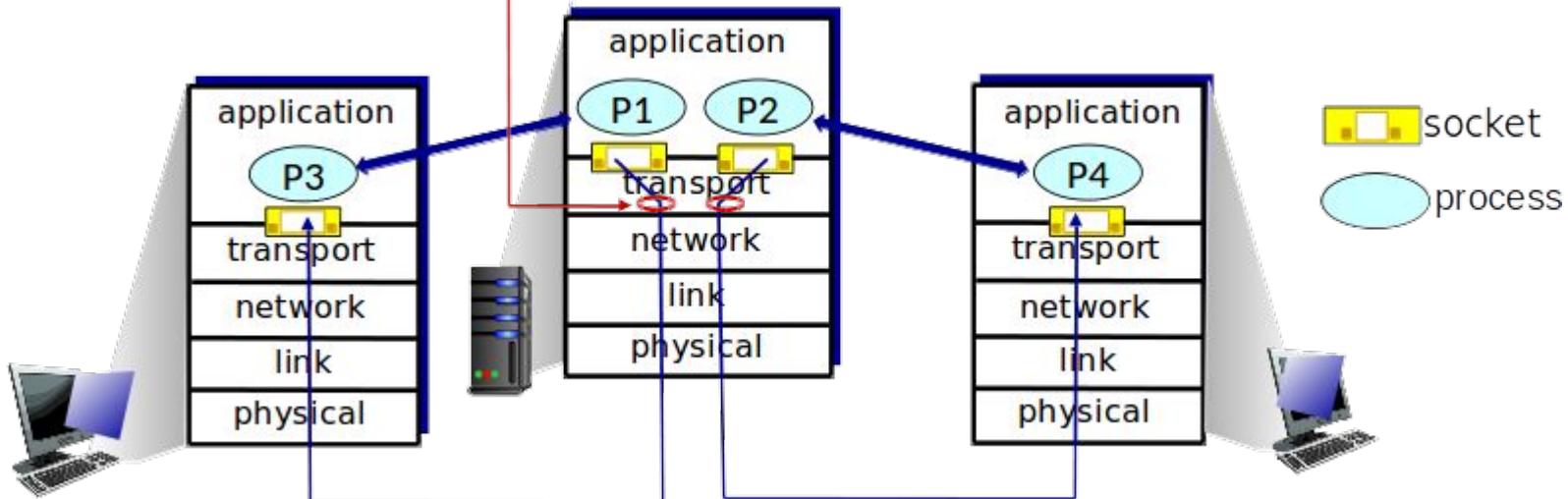
- transport-layer services
- **multiplexing and demultiplexing**
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
- principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality



Multiplexing/demultiplexing

multiplexing at sender:

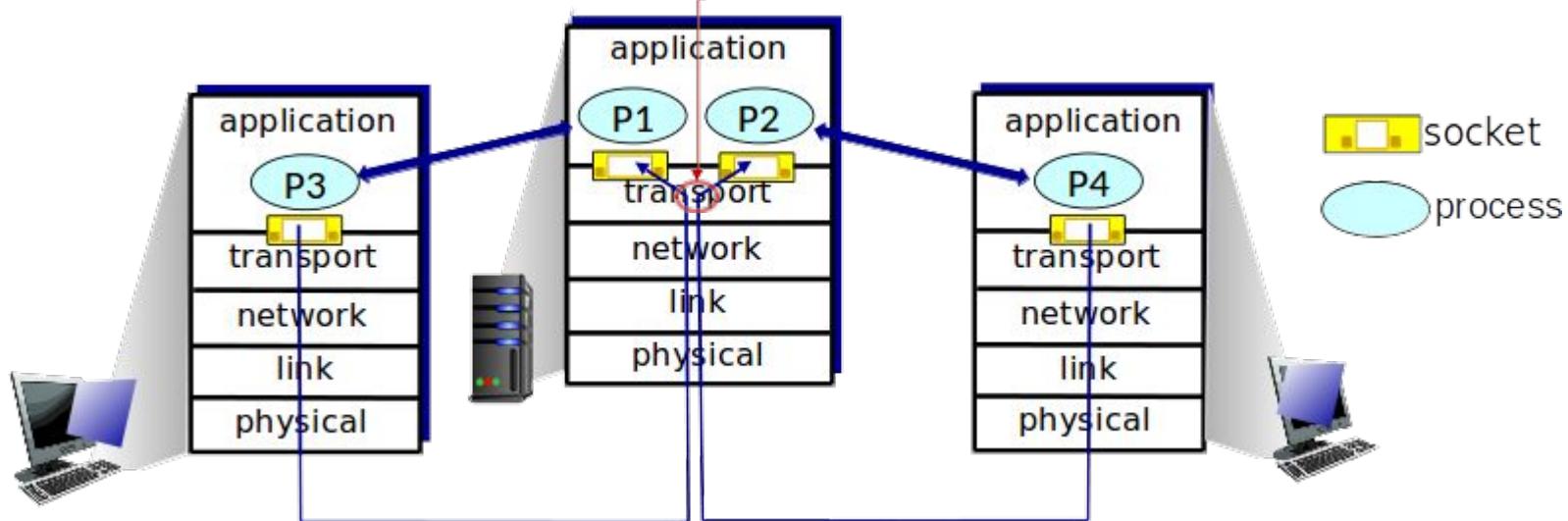
handle data from multiple sockets, add transport header
(later used for demultiplexing)



Multiplexing/demultiplexing

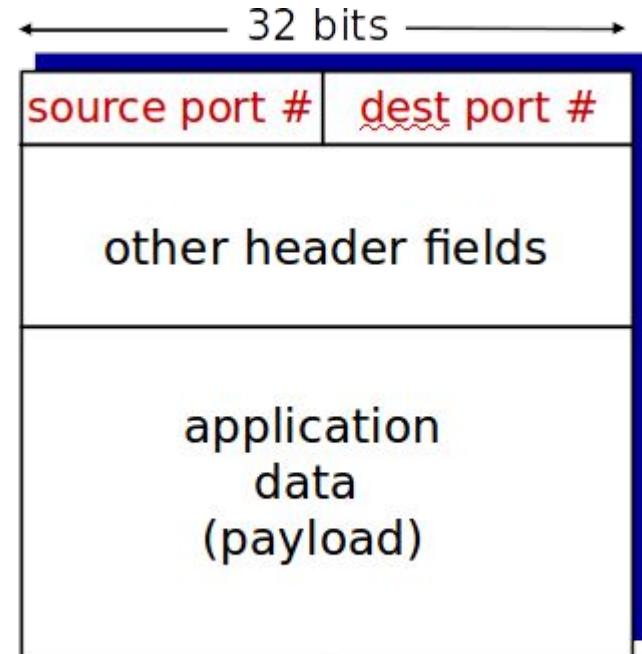
demultiplexing at receiver:

use header info to deliver received segments to correct socket



How demultiplexing works

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

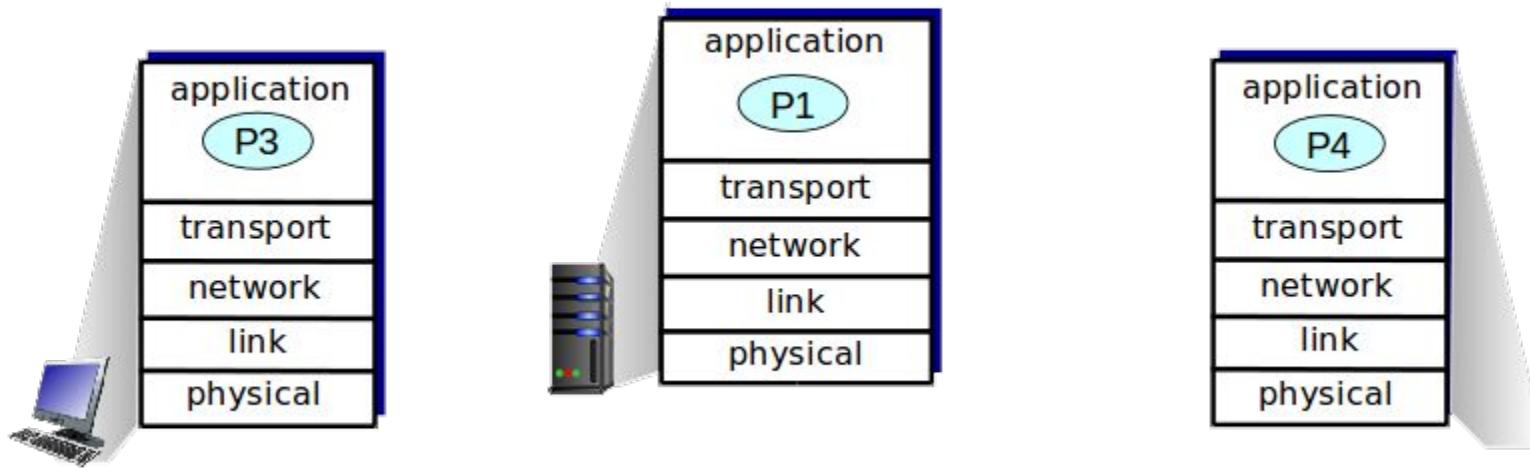


TCP/UDP segment format

Connectionless demultiplexing

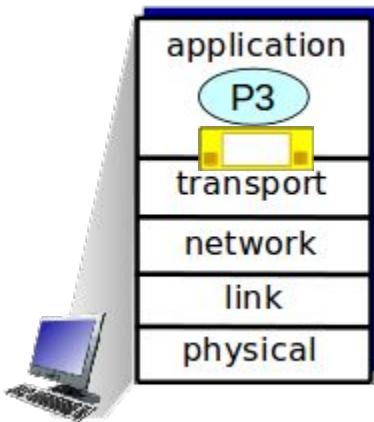
- **recall:** created socket has **host-local** port #:
- DatagramSocket mySocket1 = new DatagramSocket(12534);
 - socket = socket(AF_INET, SOCK_DGRAM)
 - Server: socket.bind("", serverPort))
 - Client: The socket for client is specified by the OS
- **recall:** when creating datagram to send into UDP socket, must specify
 - destination IP address
 - destination port #
- When host receives UDP segment:
 - checks destination port # in segment
 - directs UDP segment to socket with that port #
- IP datagrams with **same dest. port #**, but different source IP addresses and/or source port numbers will be directed to **same socket** at dest

Connectionless demux: example

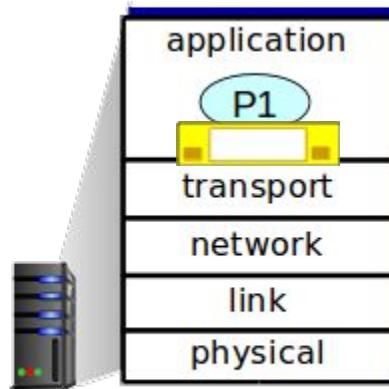


Connectionless demux: example

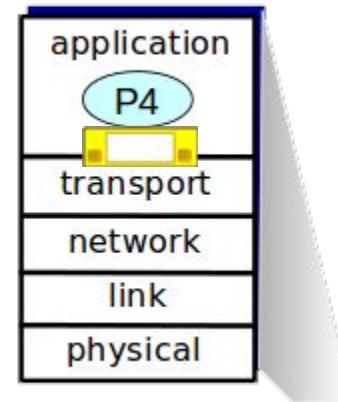
DatagramSocket mySocket2 =
new DatagramSocket (**9157**);



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

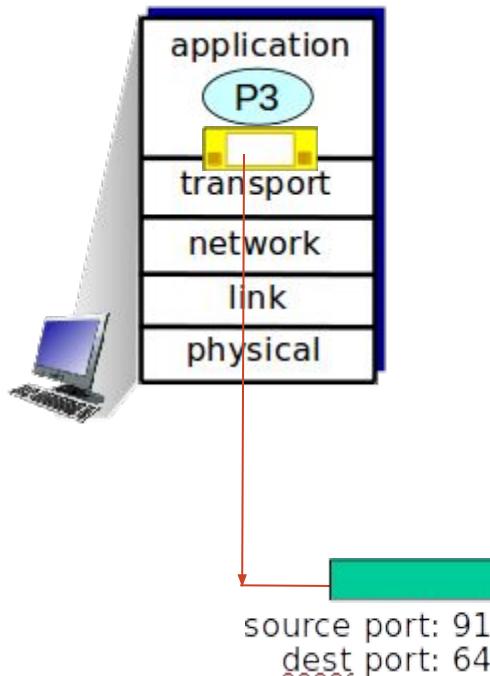


DatagramSocket mySocket1 =
new DatagramSocket (**5775**);

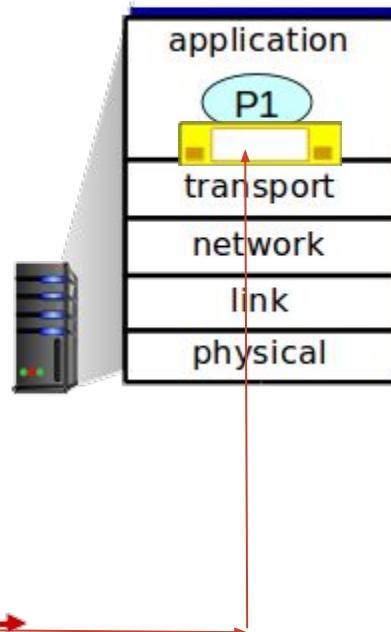


Connectionless demux: example

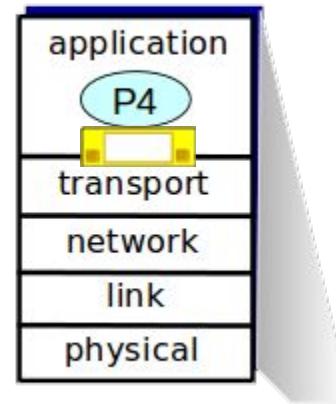
DatagramSocket mySocket2 =
new DatagramSocket (**9157**);



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

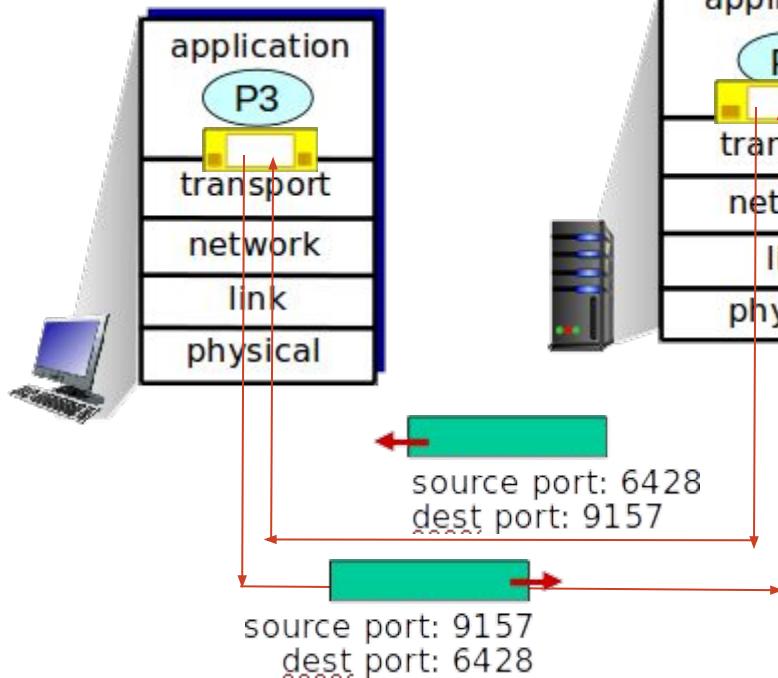


DatagramSocket mySocket1 =
new DatagramSocket (**5775**);

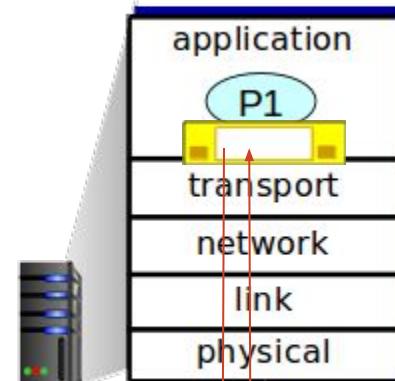


Connectionless demux: example

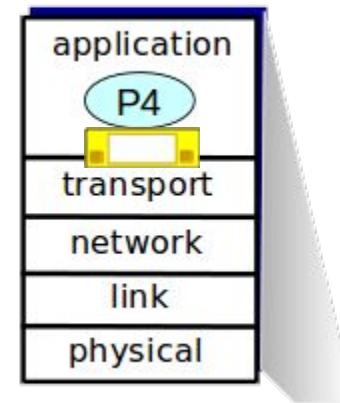
DatagramSocket mySocket2 =
new DatagramSocket (**9157**);



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



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

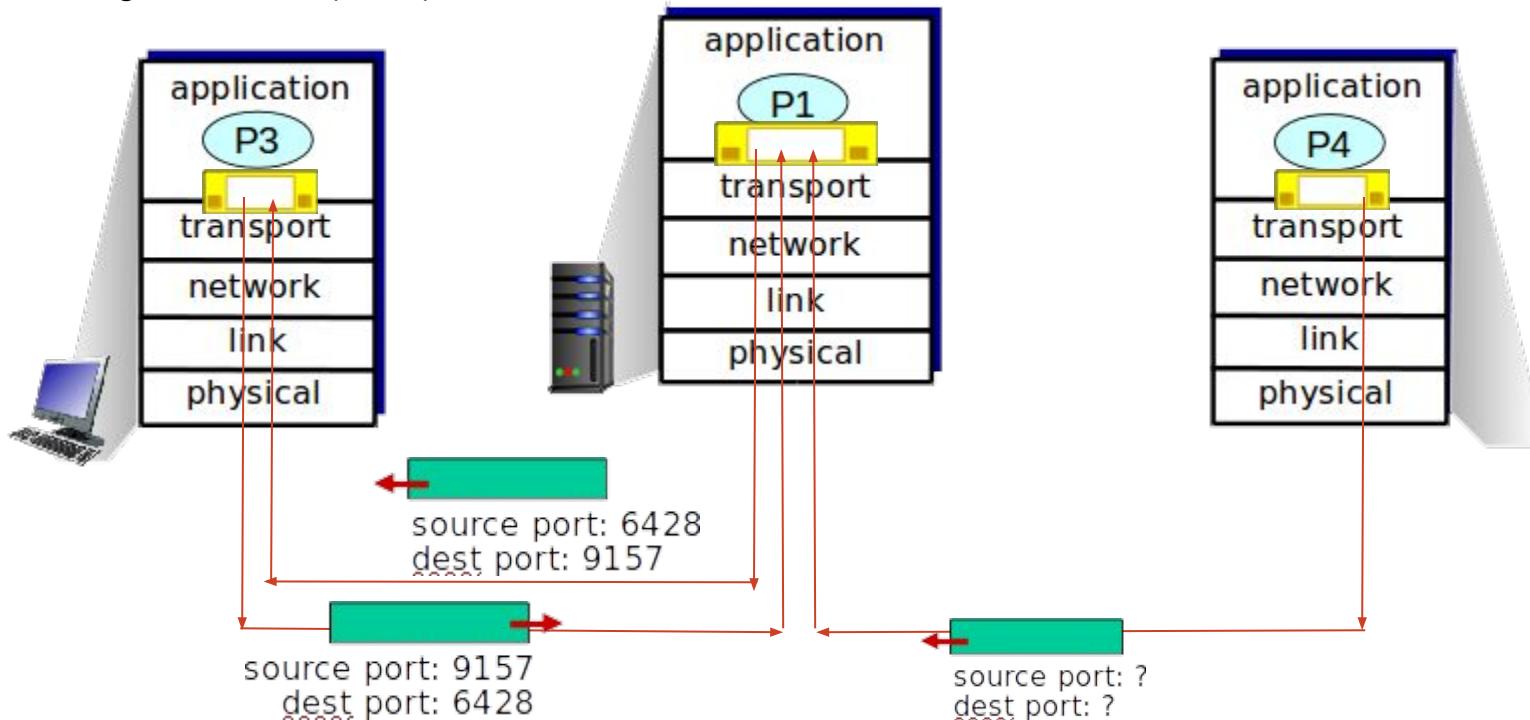


Connectionless demux: example

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

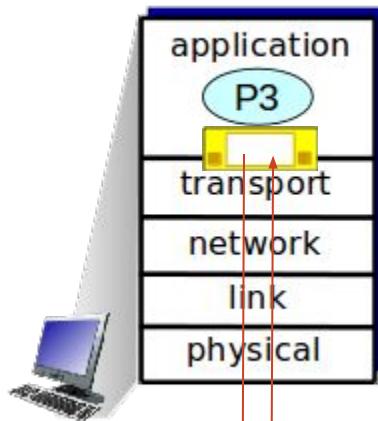
DatagramSocket serverSocket =
new DatagramSocket (**6428**);

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

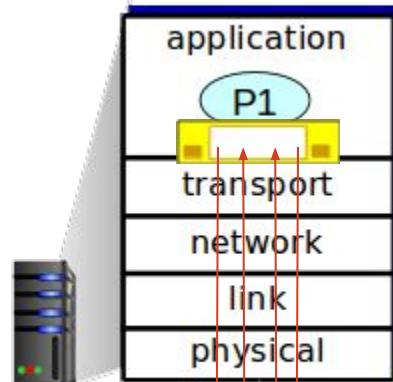


Connectionless demux: example

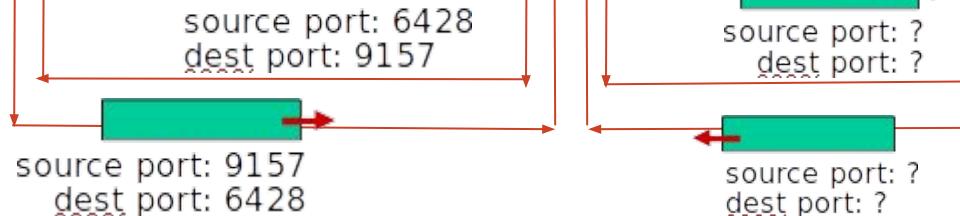
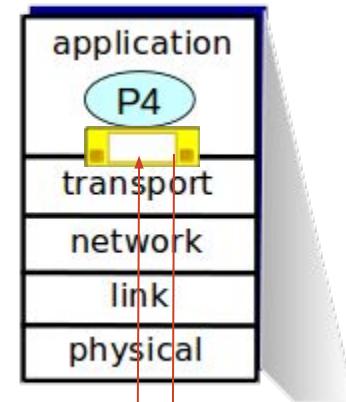
DatagramSocket mySocket2 =
new DatagramSocket (**9157**);



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



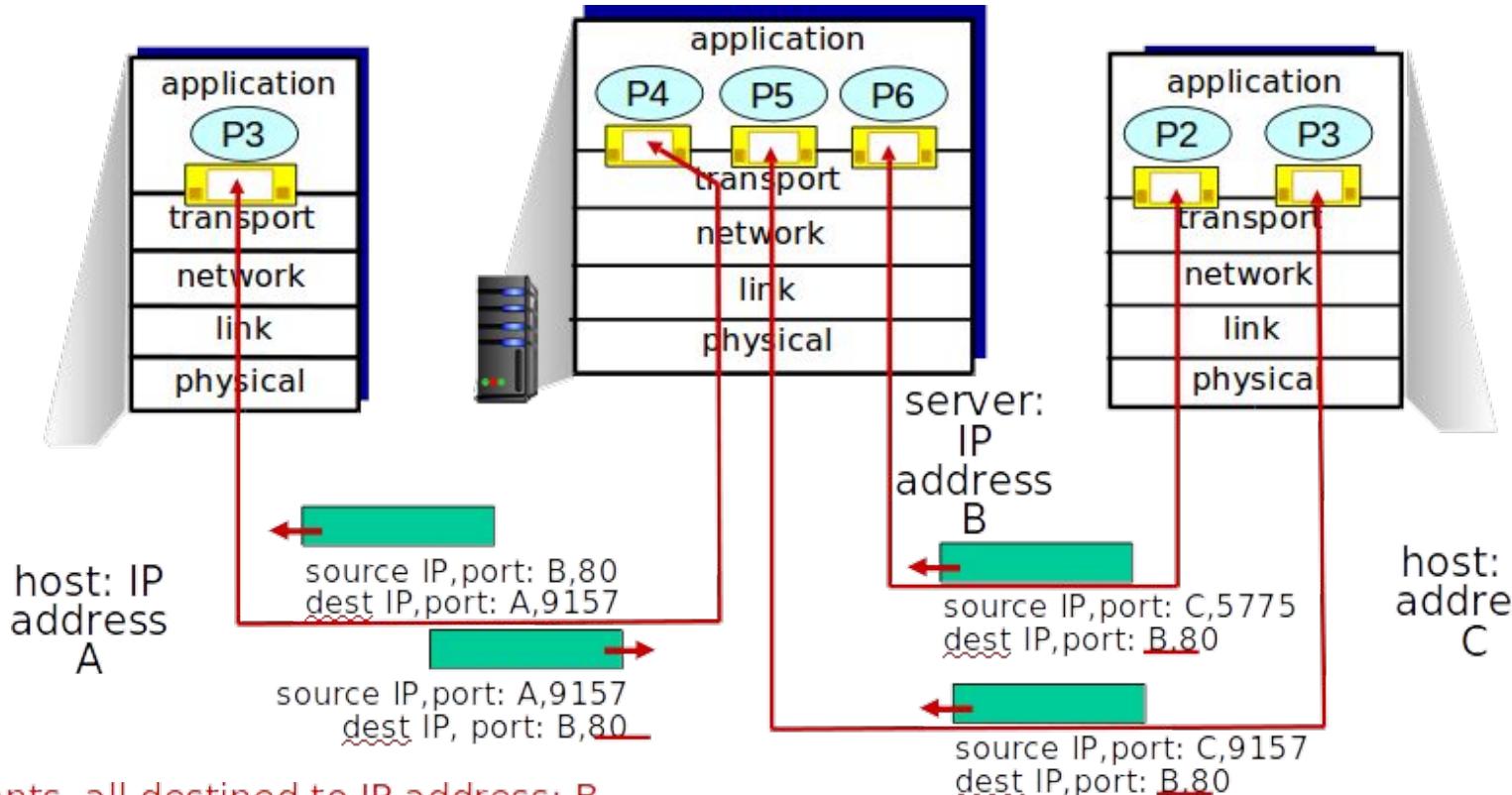
DatagramSocket mySocket1 =
new DatagramSocket (**5775**);



Connection-oriented demux

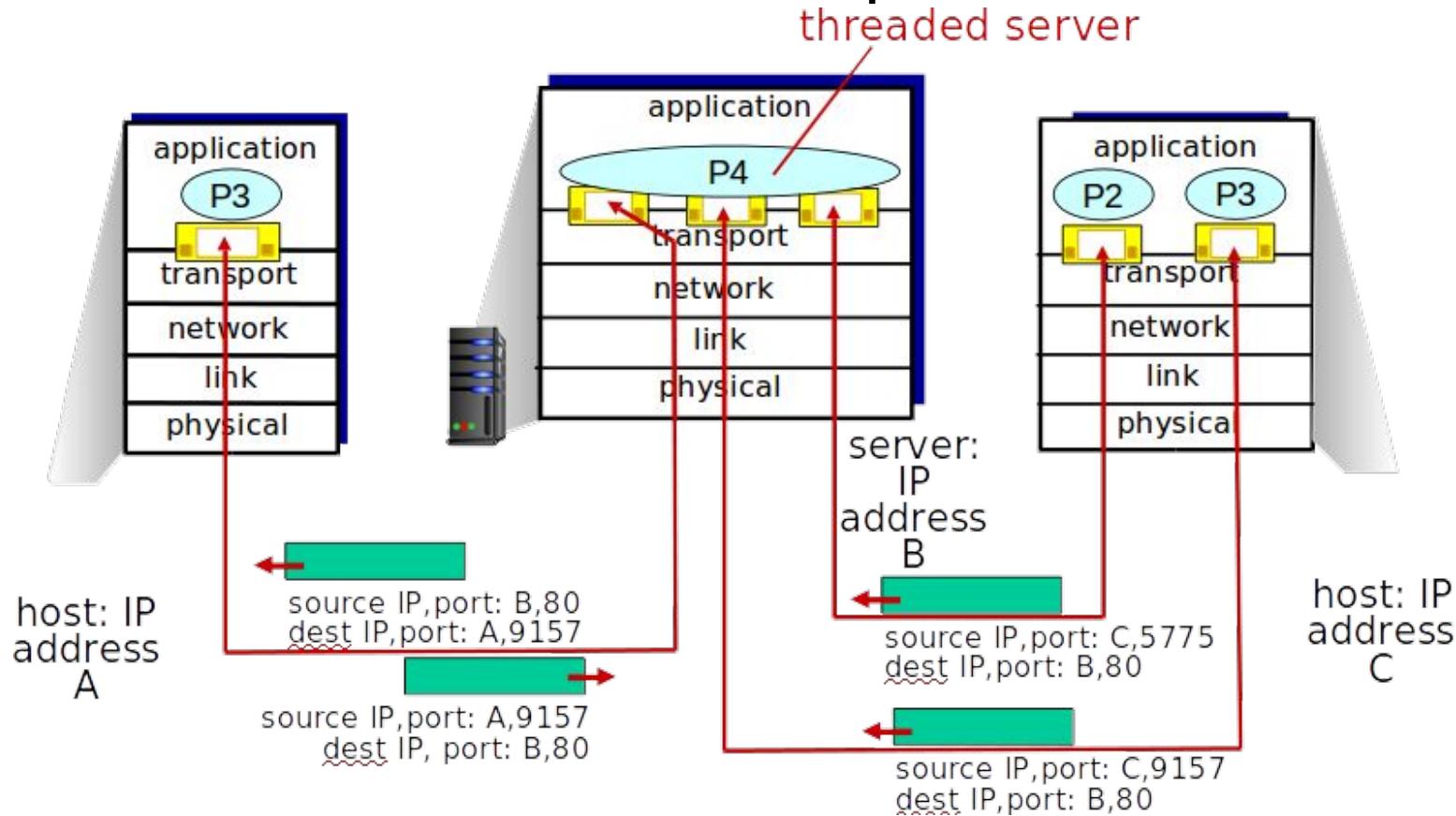
- TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- demux: receiver uses **all four values** to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple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

Connection-oriented demux: example



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

Connection-oriented demux: example



Question

Suppose Client A initiates a Telnet session with Server S. At about the same time, Client B also initiates a Telnet session with Server S. Which one is a possible source and destination port numbers for the segments sent from A to S. Telnet runs on port 23.

- A. Source port#:23 destination port#:23
- B. Source port#:468 destination port#:23
- C. Source port#:23 destination port#:468
- D. Source port#:468 destination port#:468

Question

Suppose Client A initiates a Telnet session with Server S. At about the same time, Client B also initiates a Telnet session with Server S.

If A and B are different hosts, is it possible that the source port number in the segments from A to S is the same as that from B to S?

- A. Yes
- B. No

Multiplexing/demultiplexing: Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- UDP: demultiplexing using destination port number (only)
- TCP: demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at all layers

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]

- “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 datagram handled independently of others
 - No connection state such as receive and send buffers, congestion-control parameters, etc.
 - A server can support more active clients when running over UDP
- Small packet header overhead
 - TCP segment has 20 bytes of header overhead; UDP had 8 bytes of overhead
- Finer application-level control over what data is sent, and when
 - TCP has congestion control, retransmission, etc.

UDP: User Datagram Protocol

- 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 [RFC 768]

- UDP use:
 - streaming multimedia apps
 - loss tolerant
 - rate sensitive: react poorly to TCP's congestion control
 - Controversial
 - UDP packets drops
 - TCP senders decrease their rates
 - DNS
 - SNMP: must often run when the network is in a stressed state; reliable, congestion-controlled data transfer is difficult to achieve
 - HTTP/3
 - reliable transfer over UDP (for example HTTP/3):
 - add reliability at application layer
 - add congestion control at application layer

UDP: User Datagram Protocol [RFC 768]



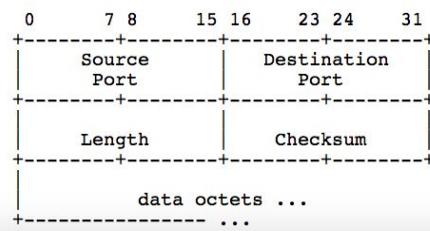
User Datagram Protocol

Introduction

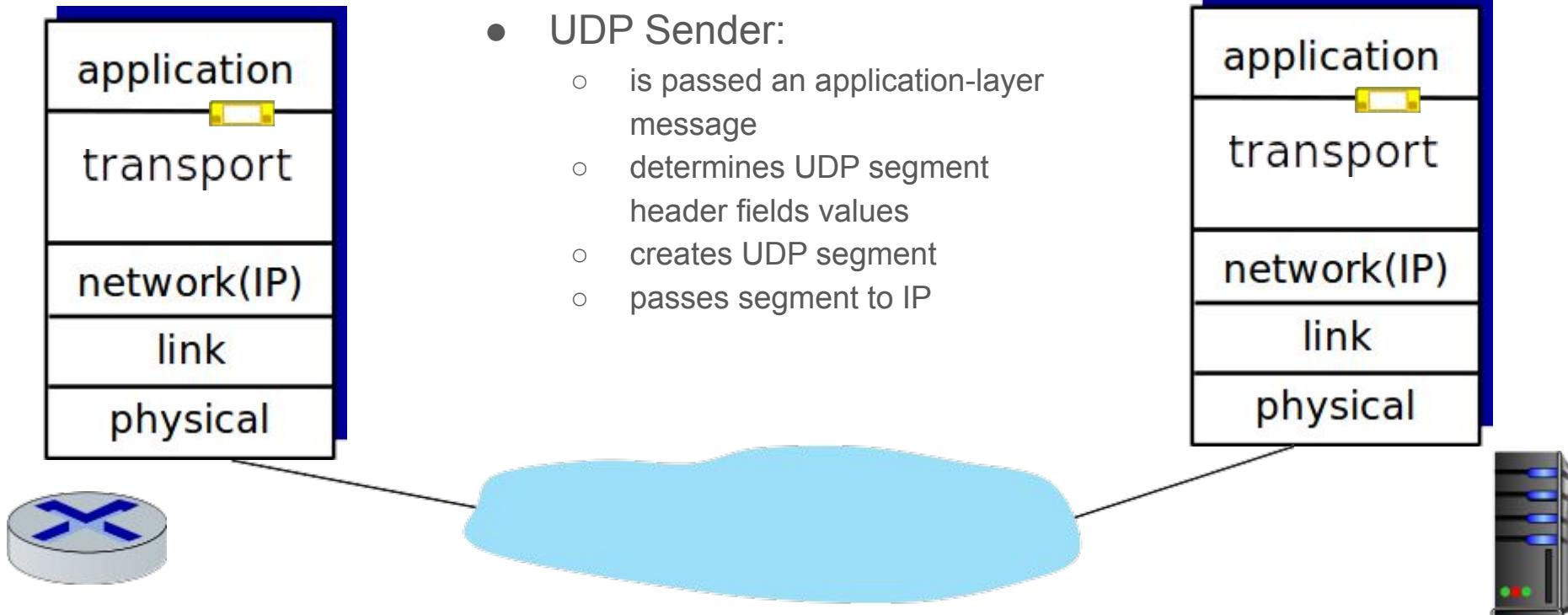
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

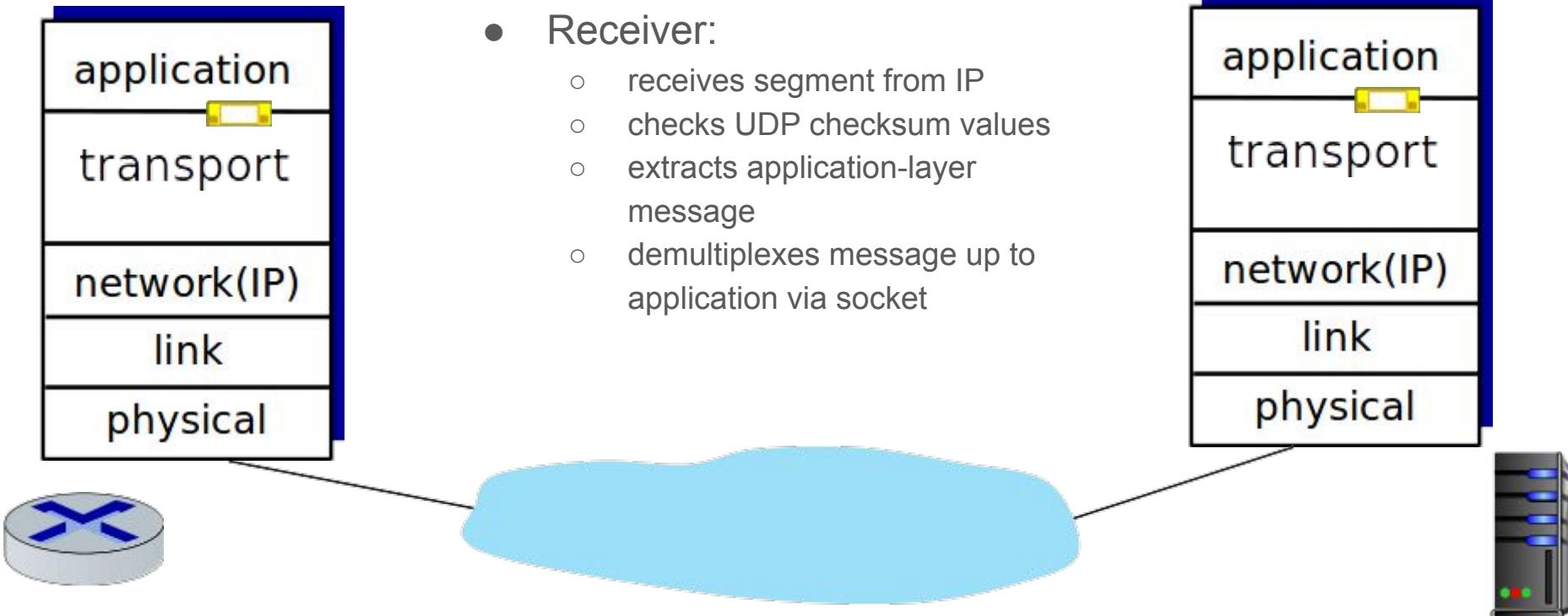
Format



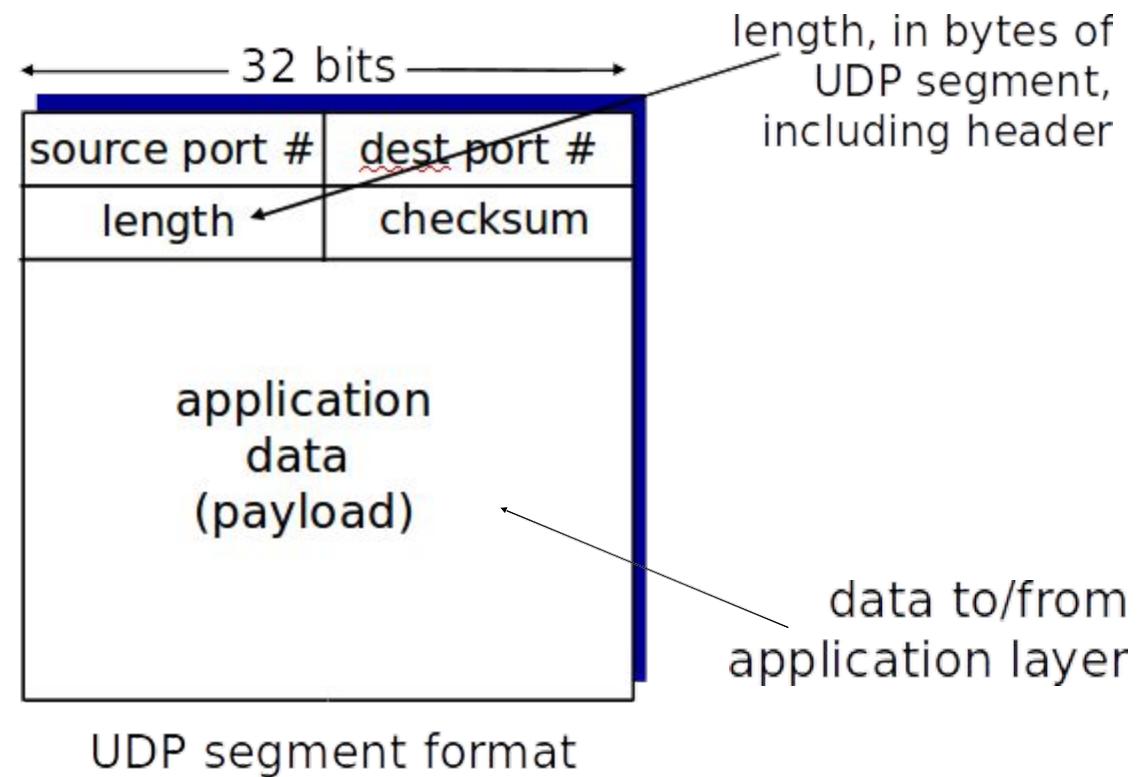
UDP: Transport Layer Actions



UDP: Transport Layer Actions



UDP: segment header



UDP checksum

- **Goal:** detect “errors” (e.g., flipped bits) in transmitted segment
- **Sender:**
 - treat segment contents, including header fields, as sequence of 16-bit integers
 - **checksum:** addition (one's complement sum) of segment contents
 - checksum value put into UDP checksum field
- **Receiver:**
 - compute checksum of received segment
 - check if computed checksum equals checksum field value:
 - by adding the checksum to the data: if the result is all one → no error; if one the bits is a 0→ error
 - NO - error detected
 - YES - no error detected. **But maybe errors nonetheless?** More later

UDP checksum

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

	1 st number	2 nd number	sum
Transmitted:	5	6	11
Received:	4	6	11

Diagram illustrating UDP checksum detection:

- A pink downward arrow points from the transmitted row to the received row.
- The "Received" row shows a receiver-computed checksum (4 + 6 = 10) and a sender-computed checksum (as received) of 11.
- The receiver-computed checksum (10) does not match the sender-computed checksum (11), indicated by a red circle with a white exclamation mark (!).

Internet Checksum: Example

example: calculate checksum for a packet with 3 16 bits data

0110011001100000	→	0110011001100000
0101010101010101	→	0101010101010101
1000111100001100		-----
		1011101110110101

		1000111100001100

1011010100111101	← 1s complement	0100101011000010
		checksum

Internet checksum: example

example: add two 16-bit integers

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

wraparound 
$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

sum	$\begin{array}{r} 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \end{array}$
checksum	$\begin{array}{r} 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \end{array}$

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

Internet checksum: example

example: add two 16-bit integers

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \\ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \\ \hline \end{array}$$

0 1
1 0
1 0

wraparound

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

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

sum 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0

checksum 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1

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

Why UDP provides a Checksum?

- link-by-link is not guaranteed
 - No guarantee that all the links between source and destination provide error checking
- in-memory errors
 - bit errors could be introduced when a segment is stored in a router's memory

Summary: UDP

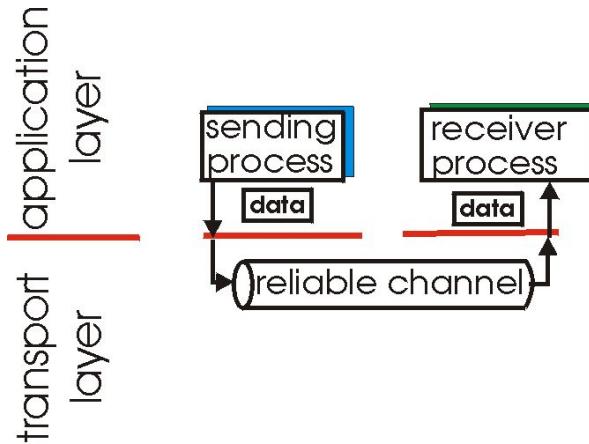
- “no frills” protocol:
 - segments may be lost, delivered out of order
 - best effort service: “send and hope for the best”
- UDP has its plusses:
 - no setup/handshaking needed (no RTT incurred)
 - can function when network service is compromised
 - helps with reliability (checksum)
- build additional functionality on top of UDP in application layer (e.g., HTTP/3)

outline

- transport-layer services
- multiplexing and demultiplexing
- connectionless transport: UDP
- **principles of reliable data transfer**
- connection-oriented transport: TCP
- principles of congestion control
- TCP congestion control

Principles of reliable data transfer

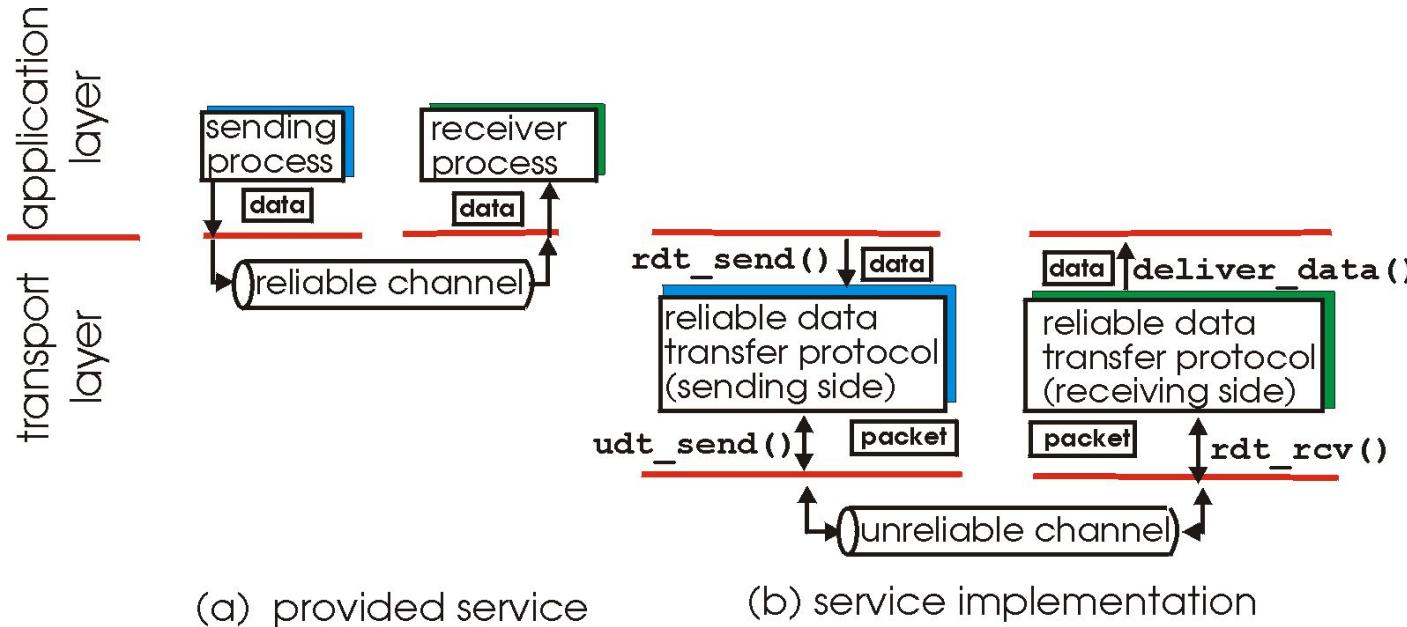
- important in application, transport, link layers
- top-10 list of important networking topics!



(a) provided service

Principles of reliable data transfer

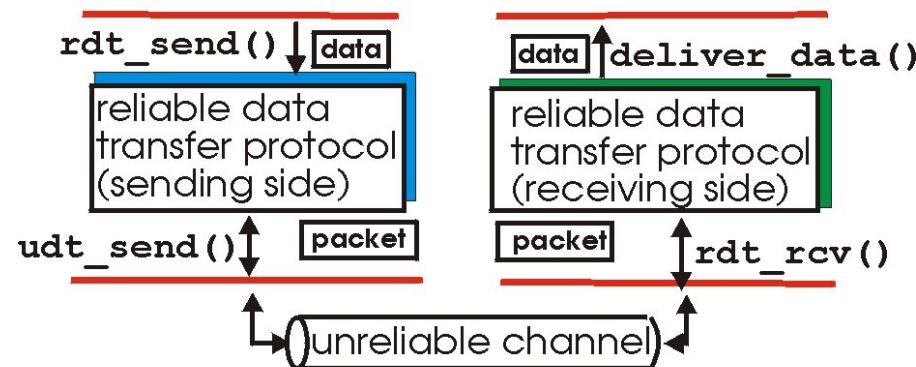
- important in application, transport, link layers
- top-10 list of important networking topics!



Principles of reliable data transfer

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

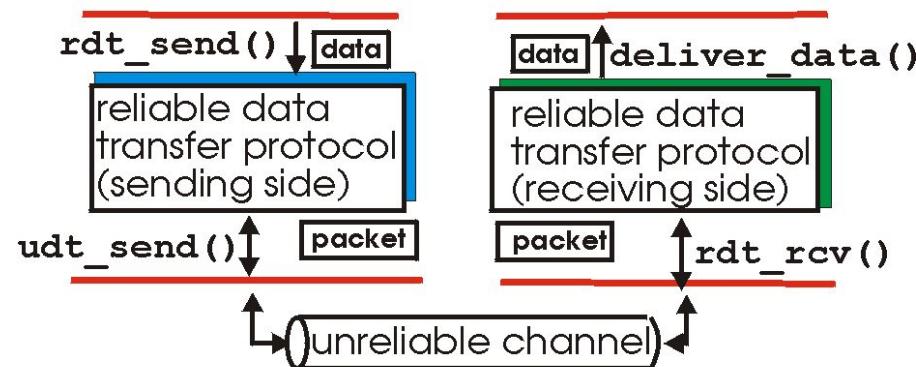
- Level of unreliability



Reliable service implementation

Principles of reliable data transfer

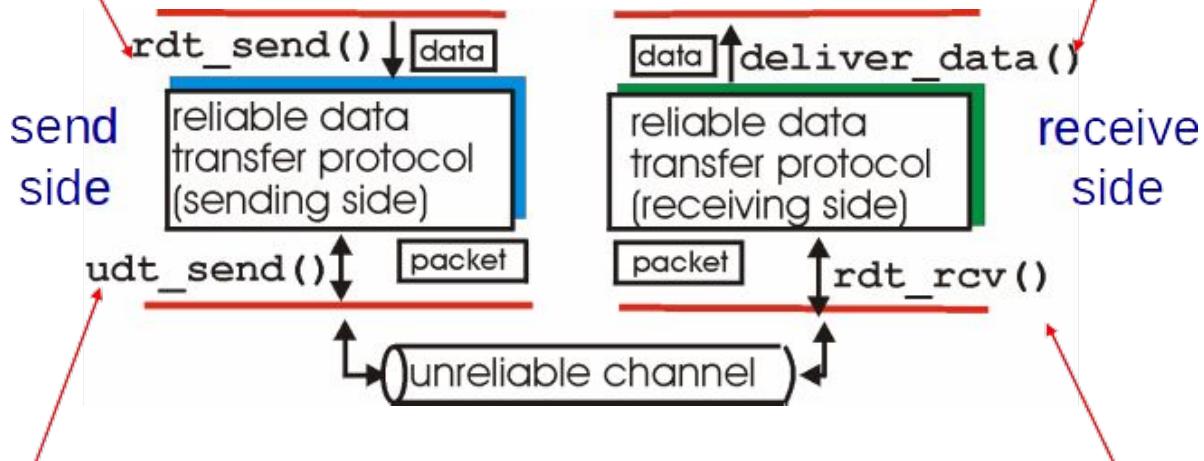
- Sender, receiver do not know the “state” of each other, e.g., was a message received?
 - Unless communicated via a message



Reliable data transfer protocol (rdt): interfaces

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

deliver_data(): called by **rdt** to deliver data to upper



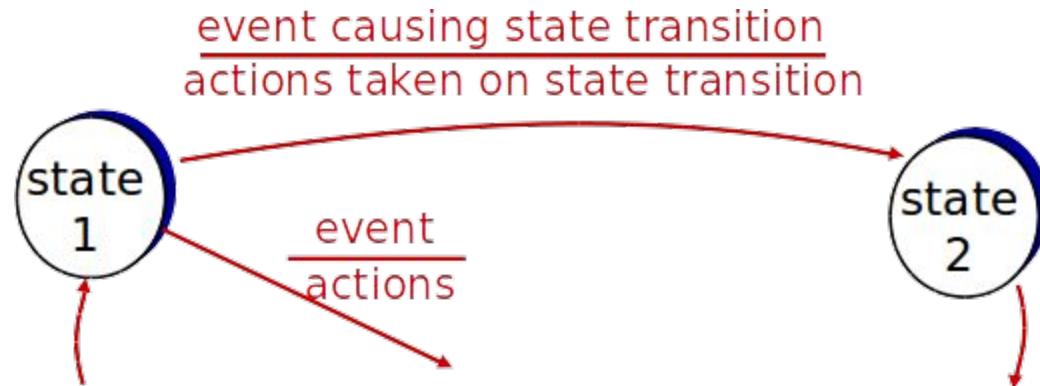
udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

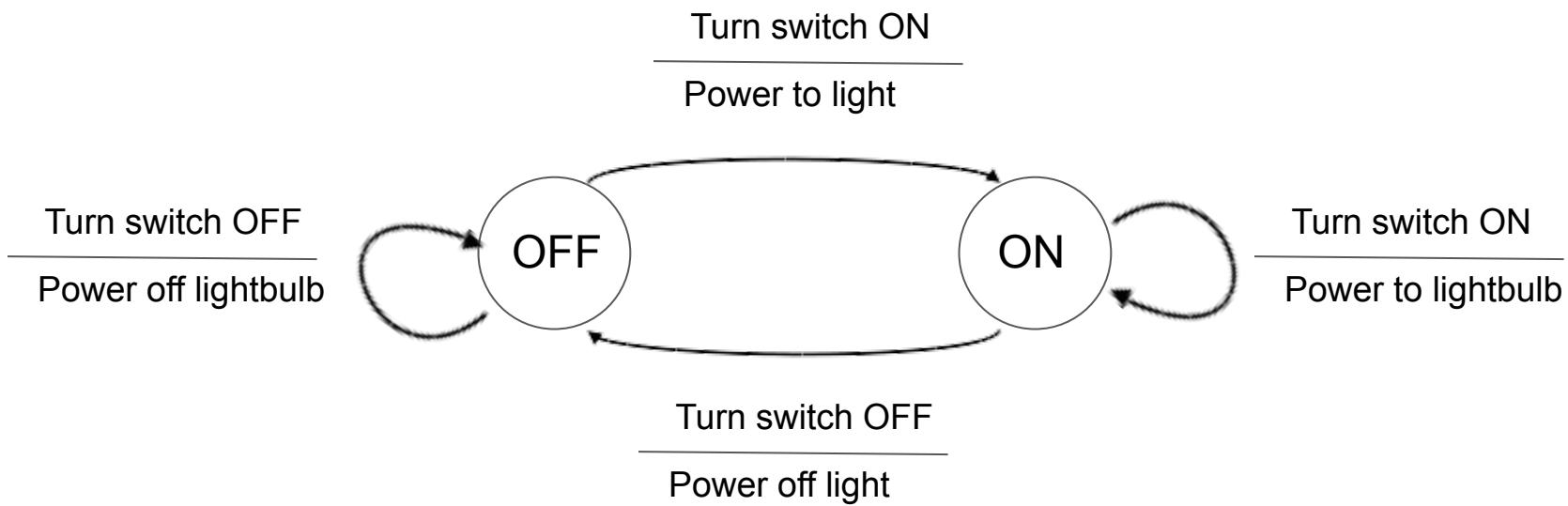
rdt_rcv(): called when packet arrives on rcv-side of channel

Reliable data transfer: getting started

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

state: when in this "state" next state uniquely determined by next event



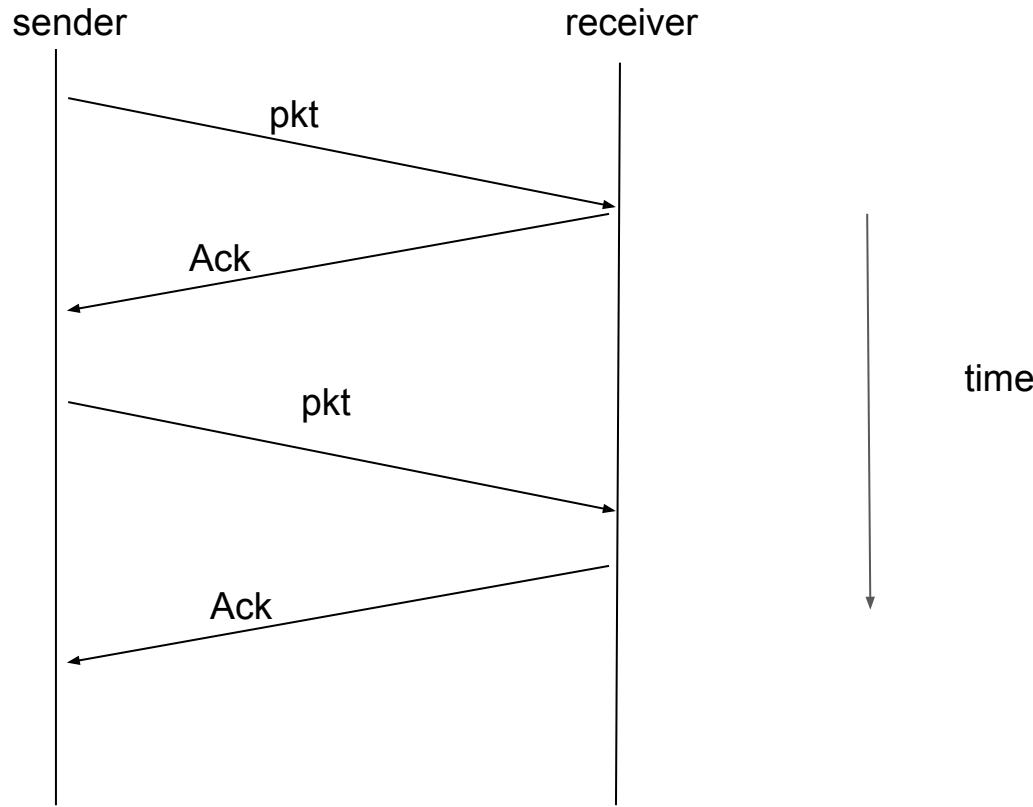


rdt1.0: reliable transfer over a reliable channel

- 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



rdt1.0: reliable transfer over a reliable channel



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:

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:

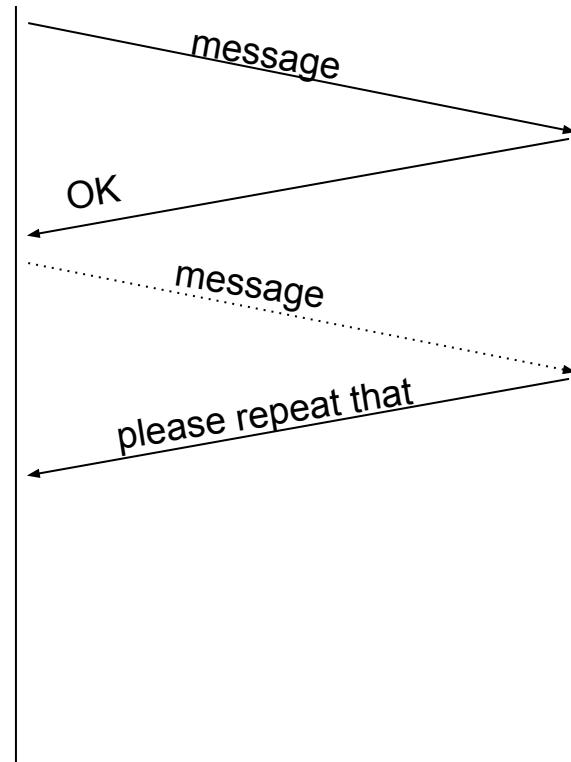
*How do humans recover from “errors”
during conversation?*

rdt2.0: channel with bit errors

- human analogy?
- positive ack
 - ok
- negative ack
 - please repeat that

corrupt packet

sender receiver



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

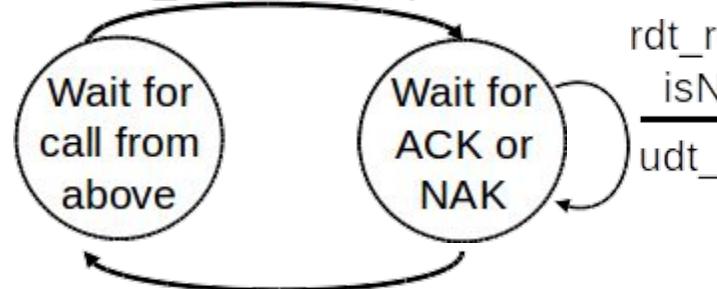
stop and wait: sender sends one packet, then waits for receiver response

rdt2.0: FSM specification

sender

rdt_send(data)

snkpkt = make_pkt(data, checksum)
udt_send(sndpkt)



Λ

receiver

rdt_rcv(rcvpkt) && corrupt(rcvpkt)

udt_send(NAK)



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)

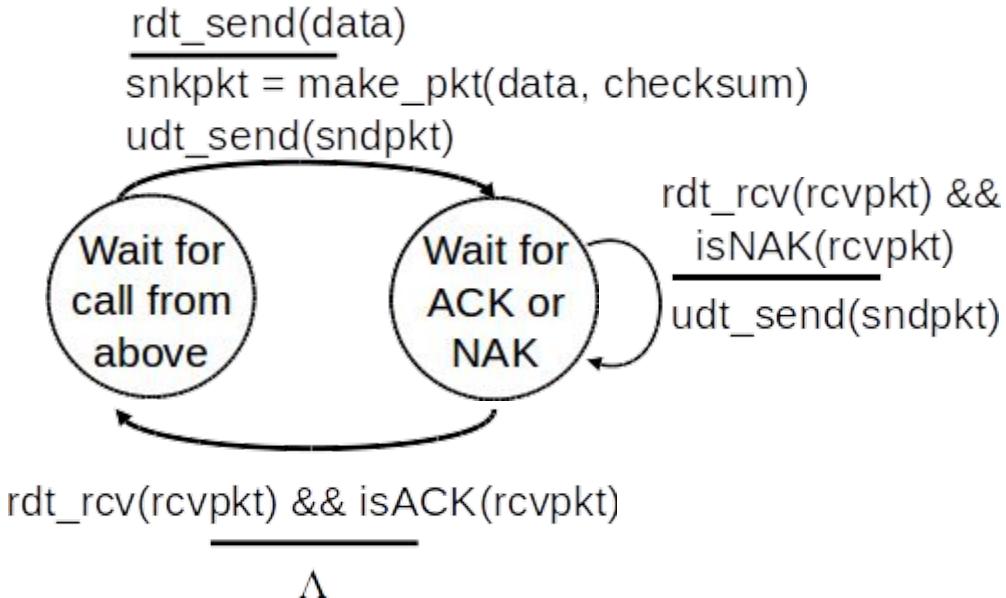
extract(rcvpkt,data)

deliver_data(data)

udt_send(ACK)

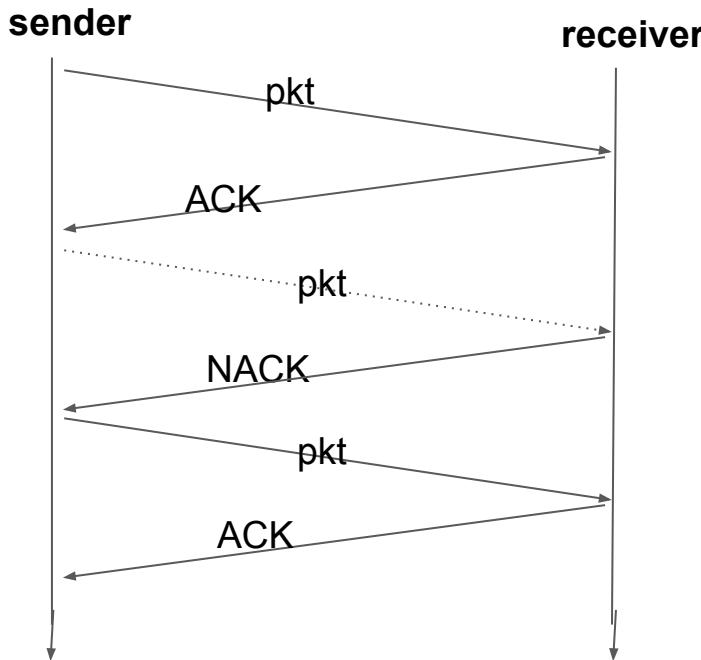
rdt2.0: FSM specification

sender



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: channel with bit errors



stop and wait
sender sends one packet, then
waits for receiver response

rdt2.0: operation with no errors

sender

rdt_send(data)

 sndpkt = make_pkt(data, checksum)

 udt_send(sndpkt)



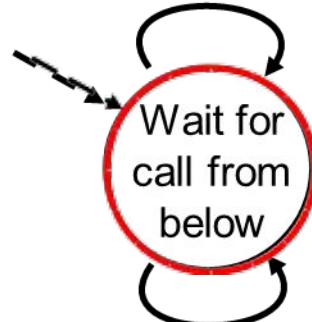
rdt_rcv(rcvpkt) && isACK(rcvpkt)

Λ

receiver

rdt_rcv(rcvpkt) && corrupt(rcvpkt)

 udt_send(NAK)



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)

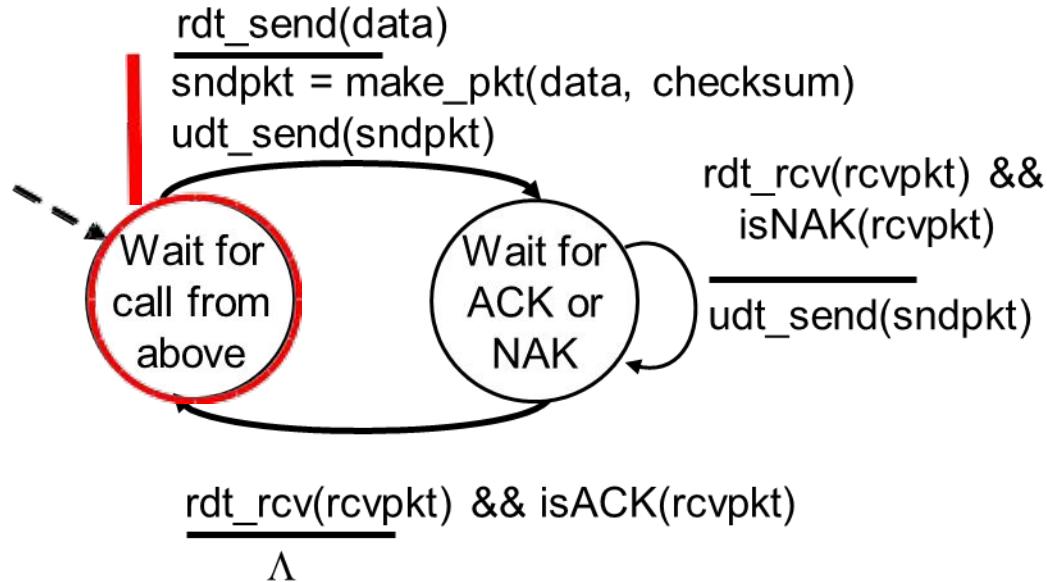
 extract(rcvpkt,data)

 deliver_data(data)

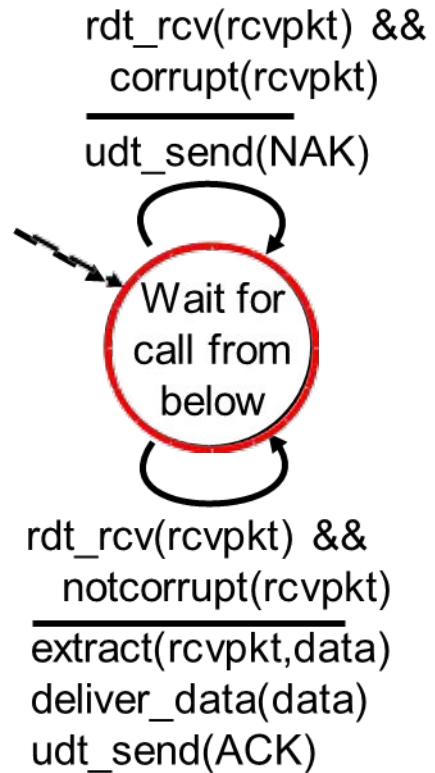
 udt_send(ACK)

rdt2.0: operation with no errors

sender



receiver



rdt2.0: operation with no errors

sender

rdt_send(data)

$\text{sndpkt} = \text{make_pkt}(\text{data}, \text{checksum})$

udt_send(sndpkt)



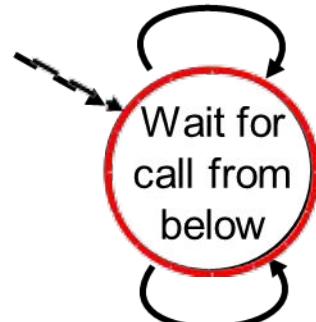
rdt_rcv(rcvpkt) && isACK(rcvpkt)

Λ

receiver

rdt_rcv(rcvpkt) && corrupt(rcvpkt)

udt_send(NAK)



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)

extract(rcvpkt,data)

deliver_data(data)

udt_send(ACK)

rdt2.0: operation with no errors

sender

rdt_send(data)

 sndpkt = make_pkt(data, checksum)

 udt_send(sndpkt)



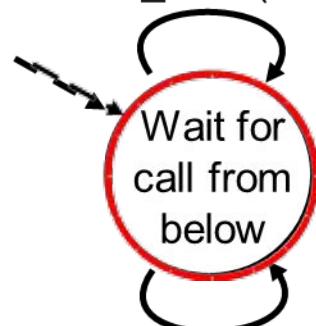
rdt_rcv(rcvpkt) && isACK(rcvpkt)

Λ

receiver

rdt_rcv(rcvpkt) && corrupt(rcvpkt)

 udt_send(NAK)



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)

 extract(rcvpkt,data)

 deliver_data(data)

 udt_send(ACK)

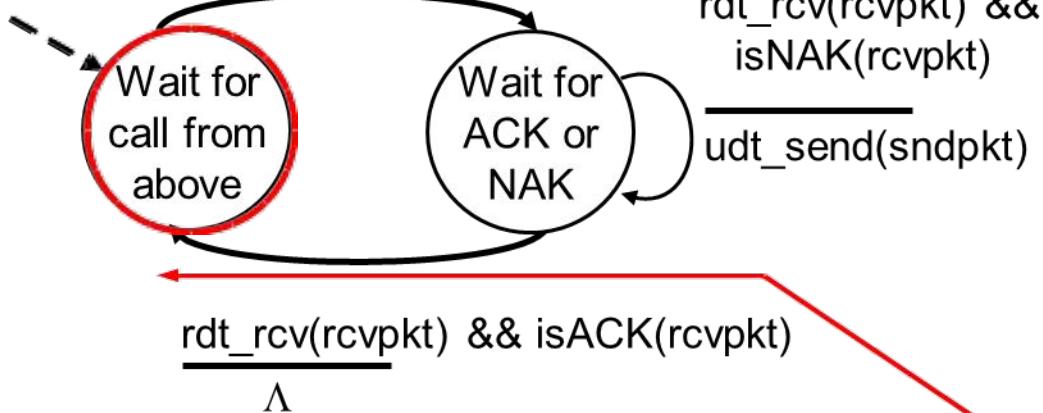
rdt2.0: operation with no errors

sender

rdt_send(data)

 sndpkt = make_pkt(data, checksum)

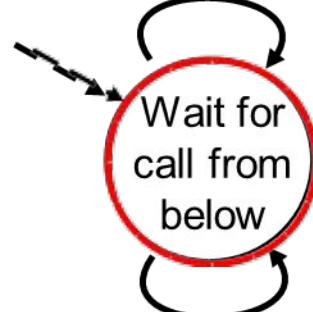
 udt_send(sndpkt)



receiver

rdt_rcv(rcvpkt) && corrupt(rcvpkt)

 udt_send(NAK)



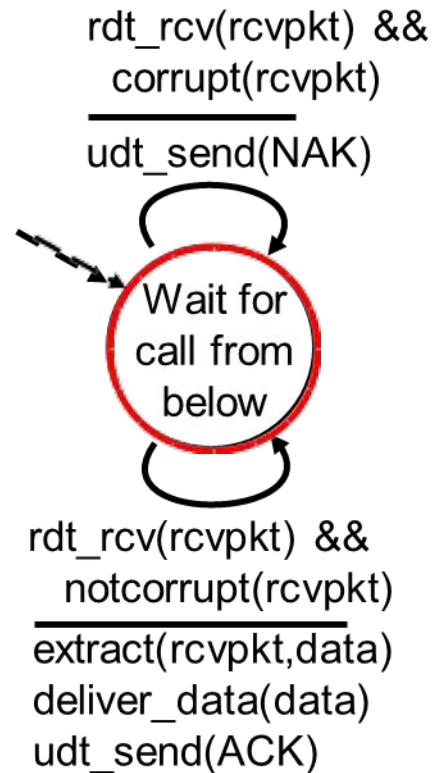
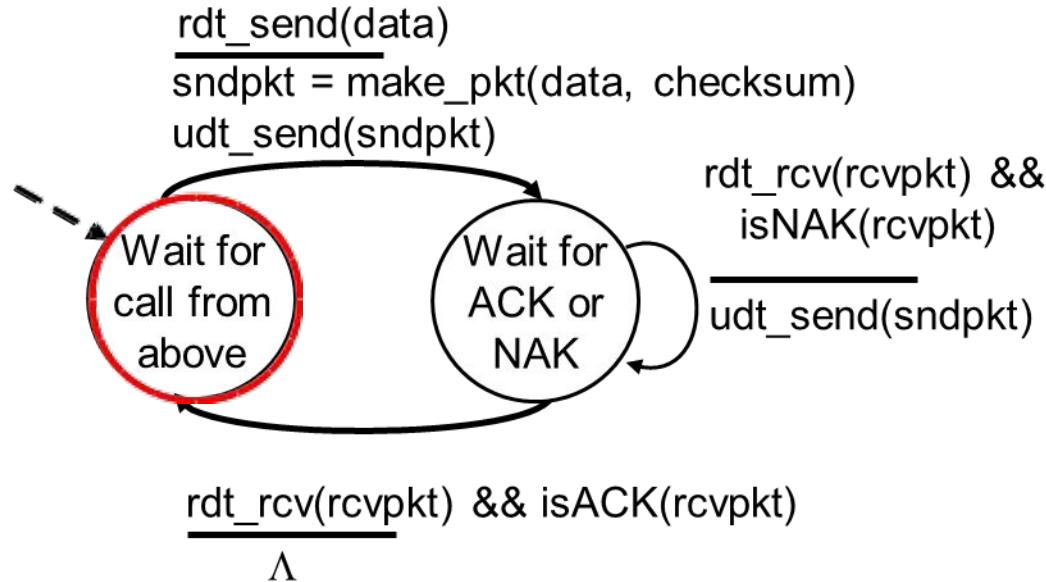
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)

 extract(rcvpkt,data)

 deliver_data(data)

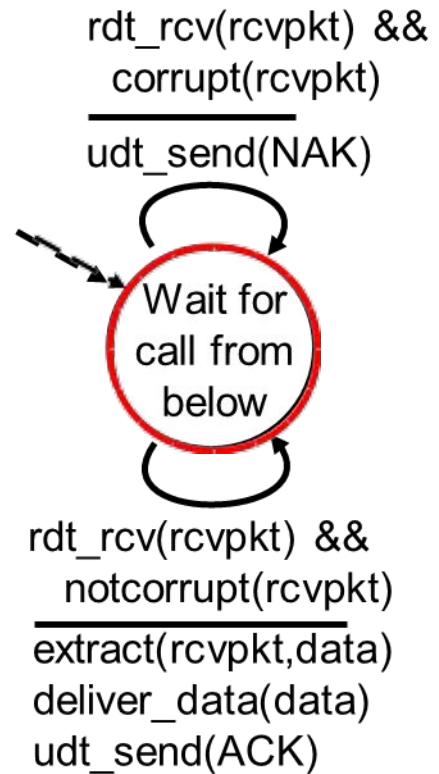
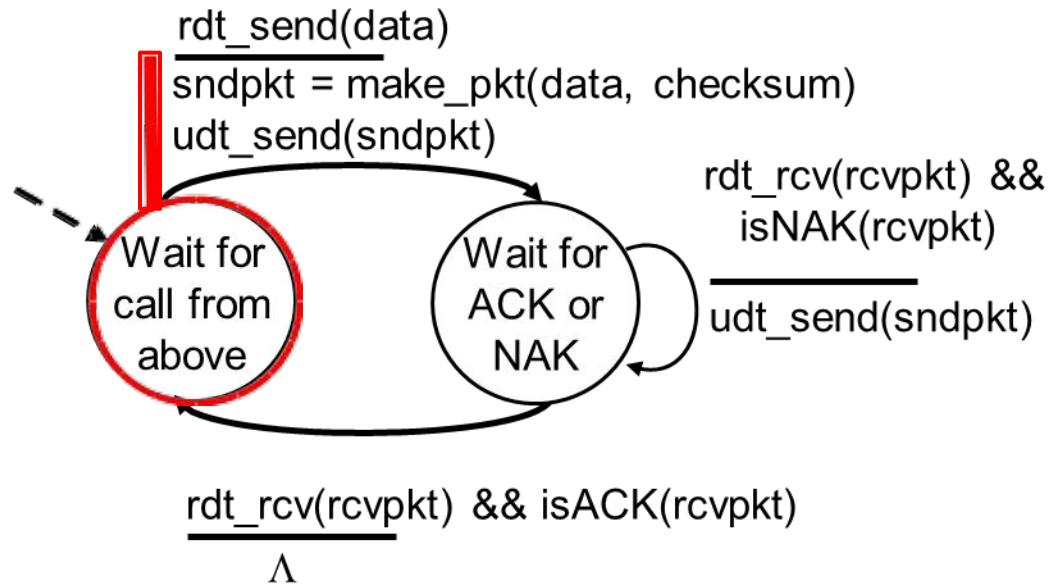
 udt_send(ACK)

rdt2.0: error scenario: corrupted packet **receiver** **sender**

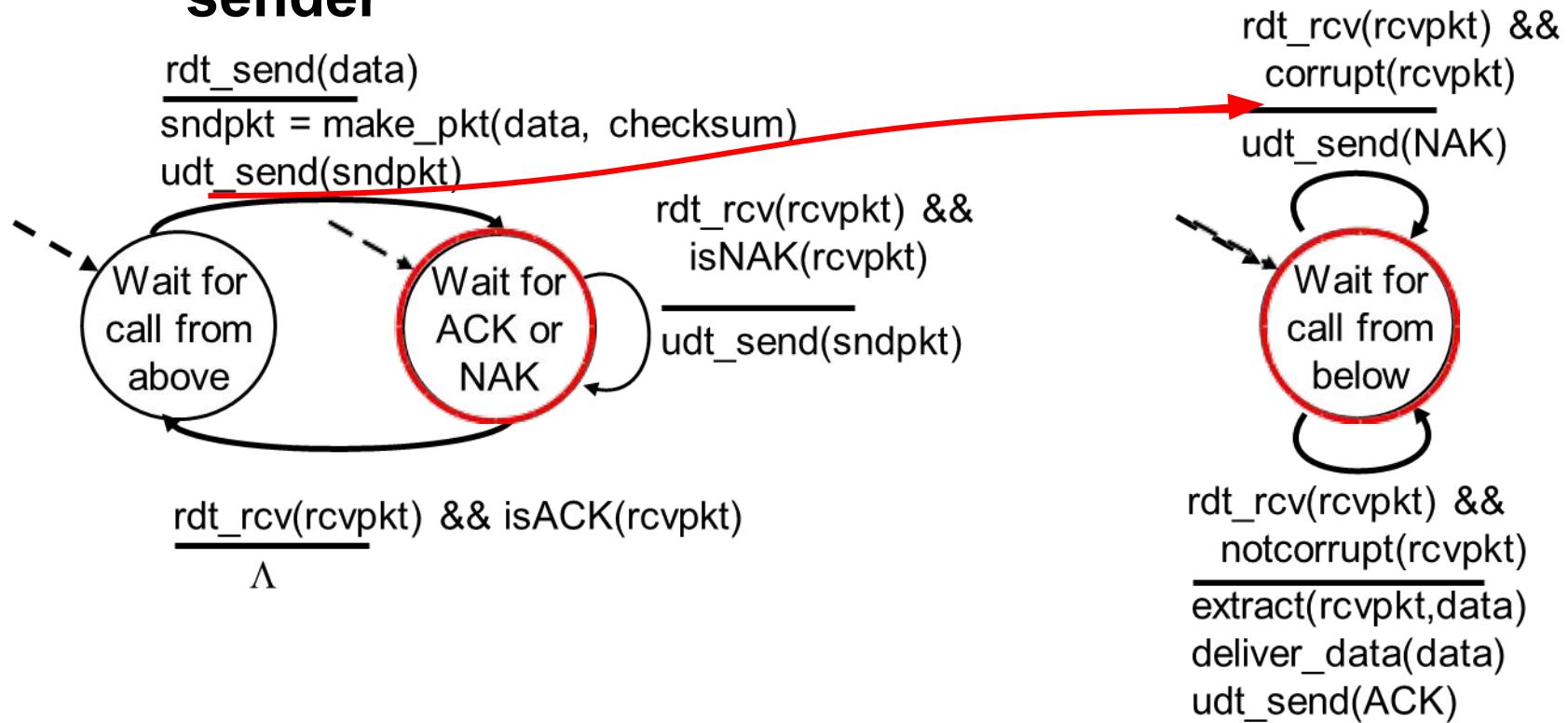


rdt2.0: error scenario: corrupted packet

receiver
sender



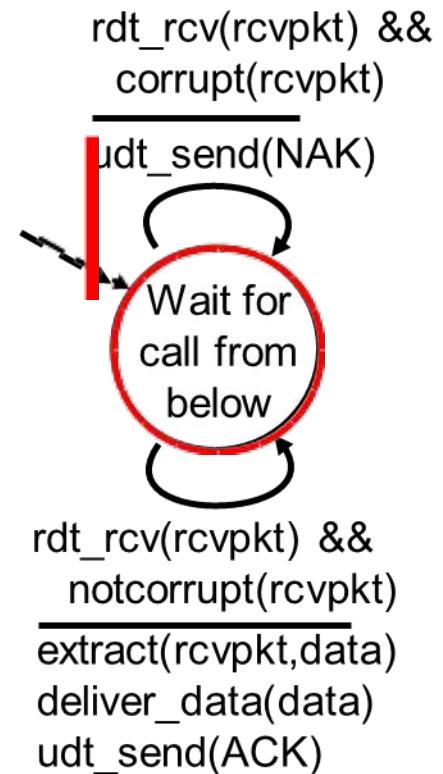
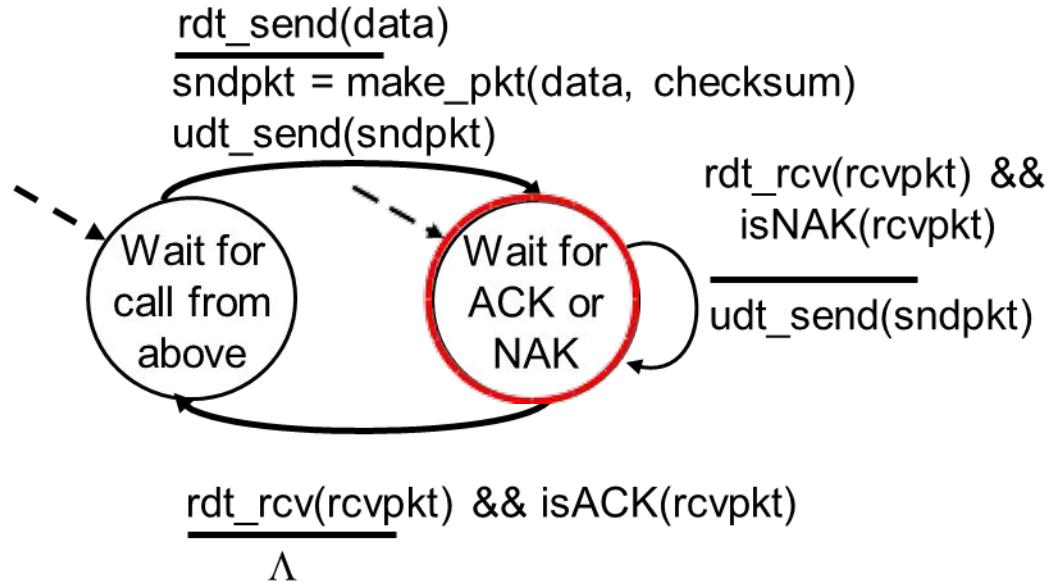
rdt2.0: error scenario: corrupted packet **receiver** **sender**



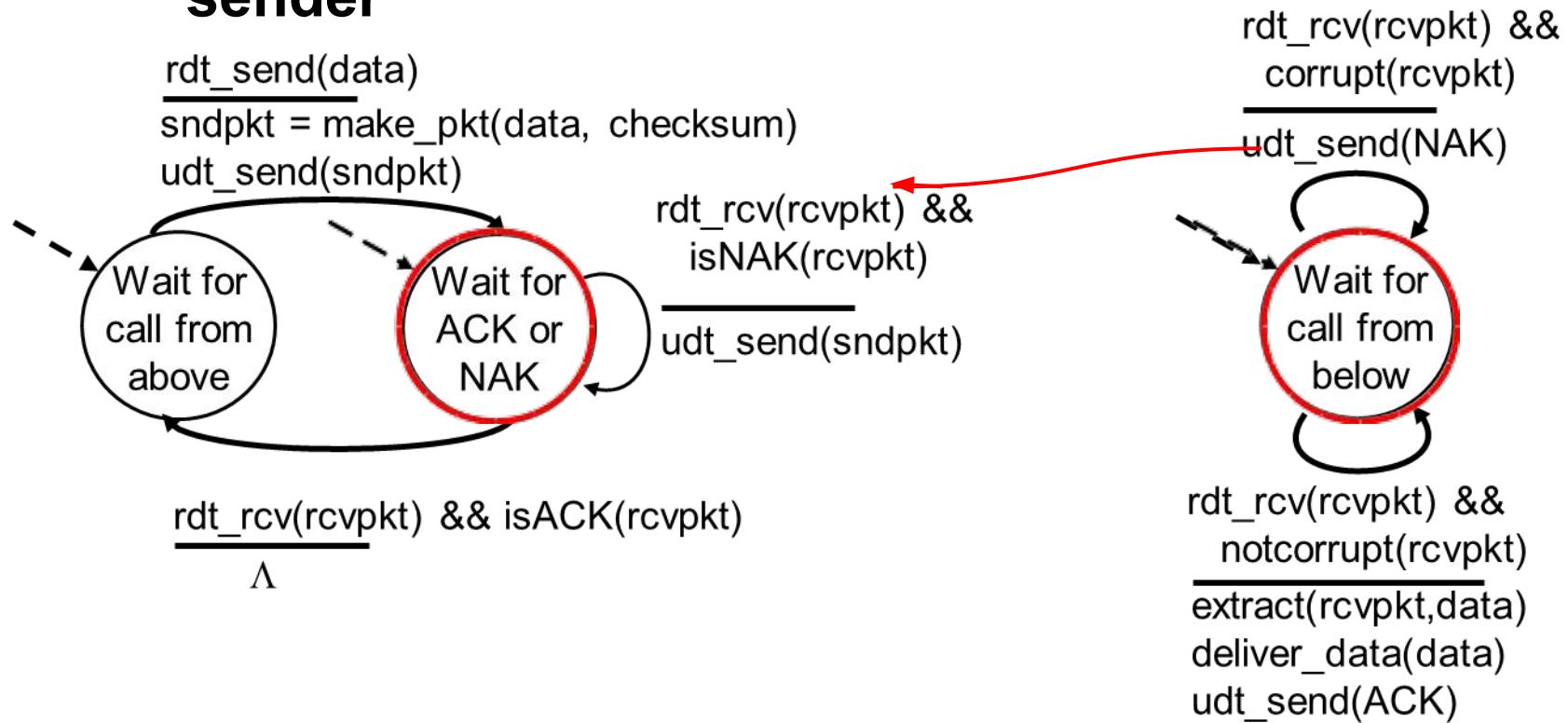
rdt2.0: error scenario: corrupted packet

receiver

sender

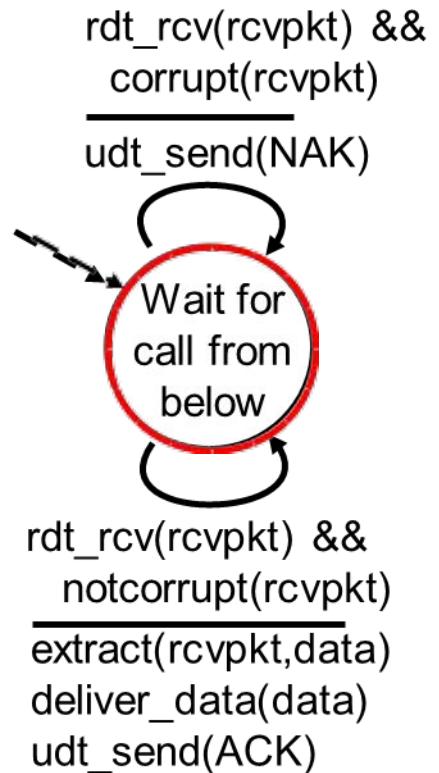
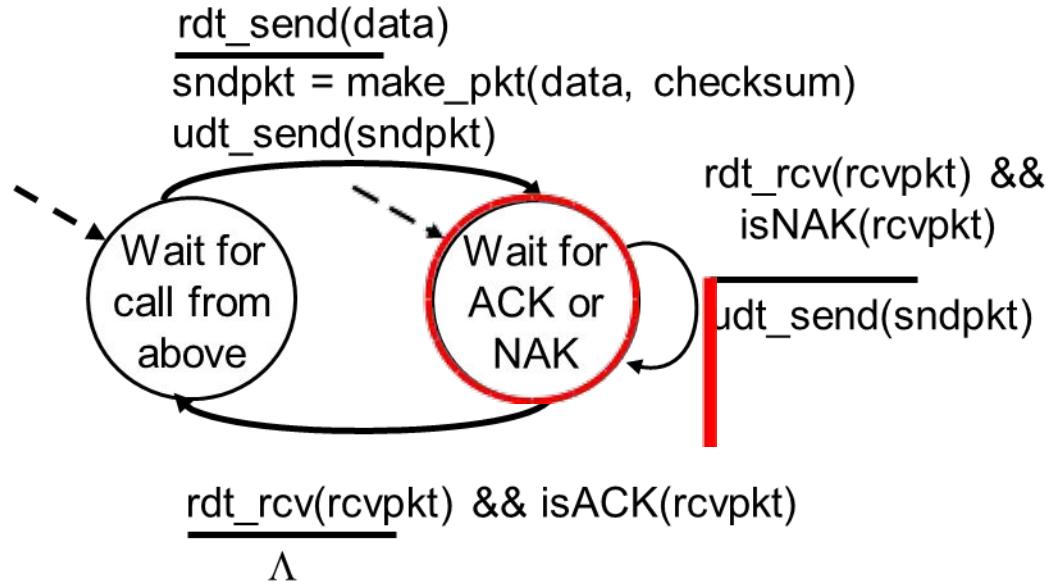


rdt2.0: error scenario: corrupted packet **receiver** **sender**



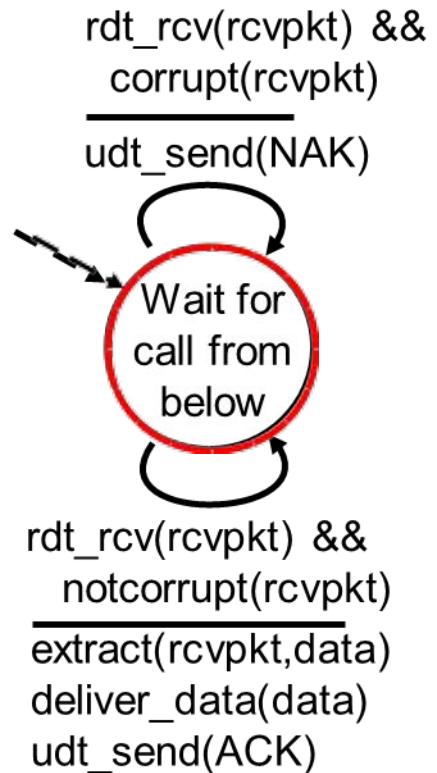
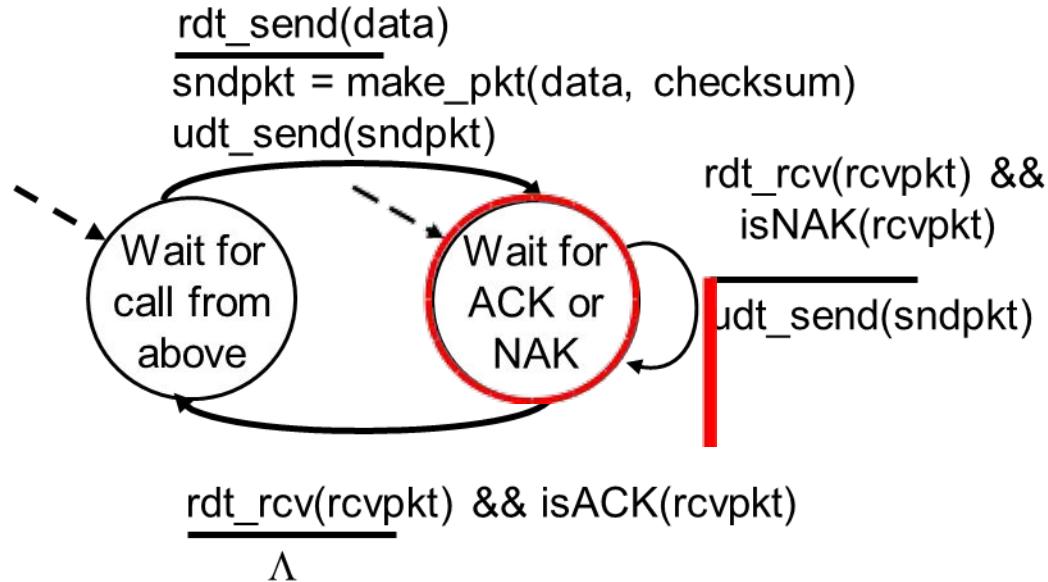
rdt2.0: error scenario: corrupted packet

receiver
sender

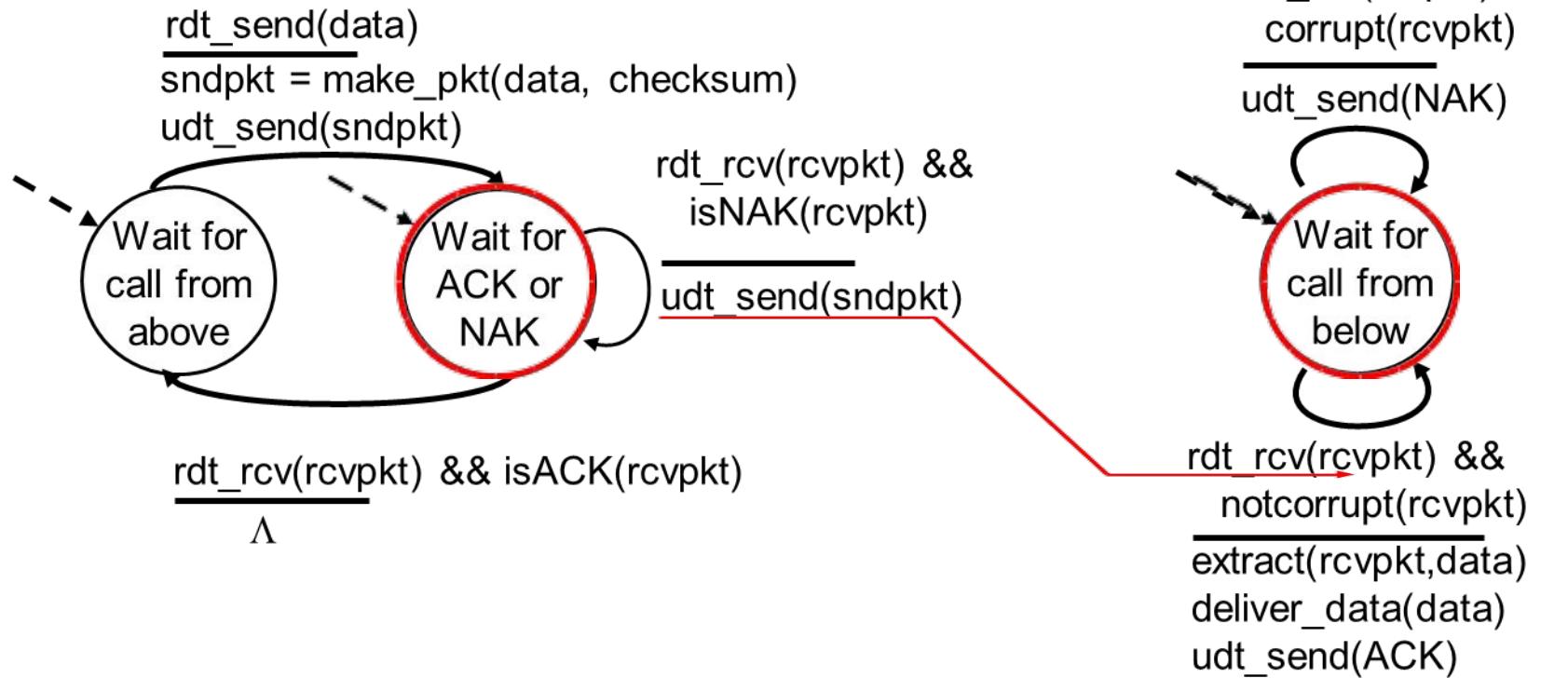


rdt2.0: error scenario: corrupted packet

receiver
sender

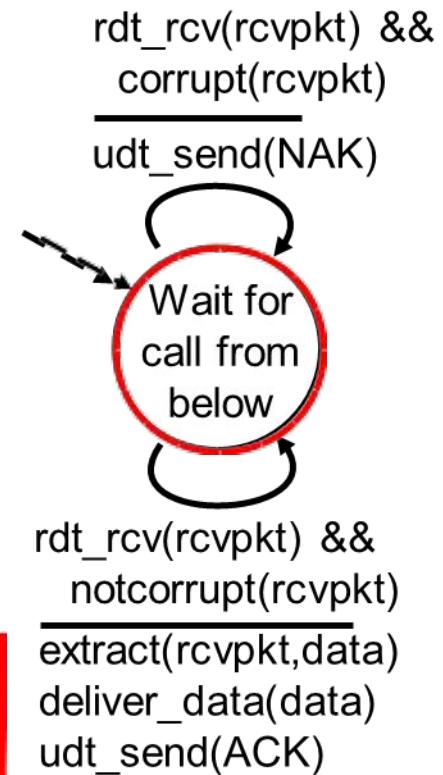
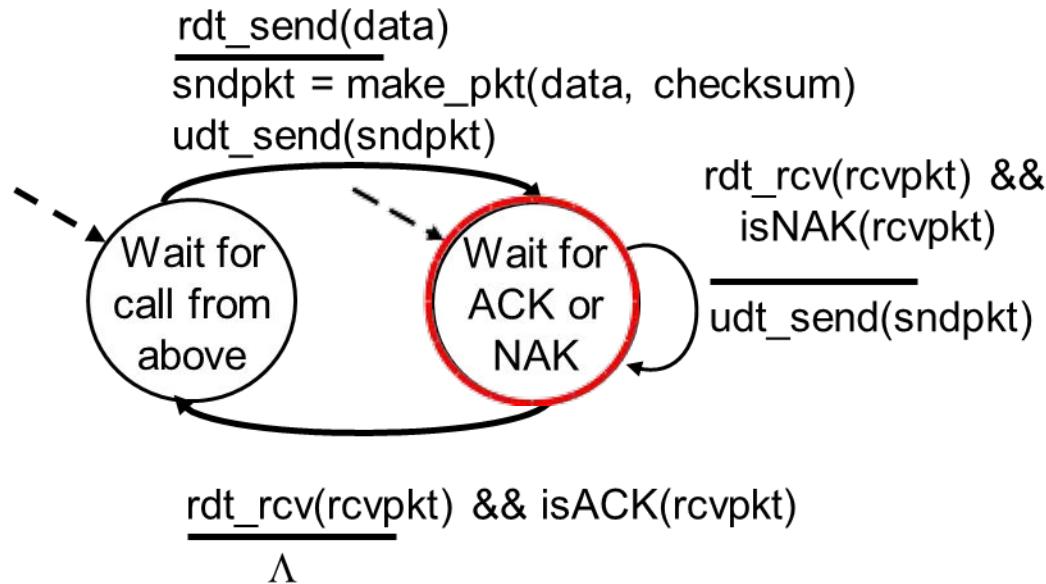


rdt2.0: error scenario: corrupted packet **receiver** **sender**

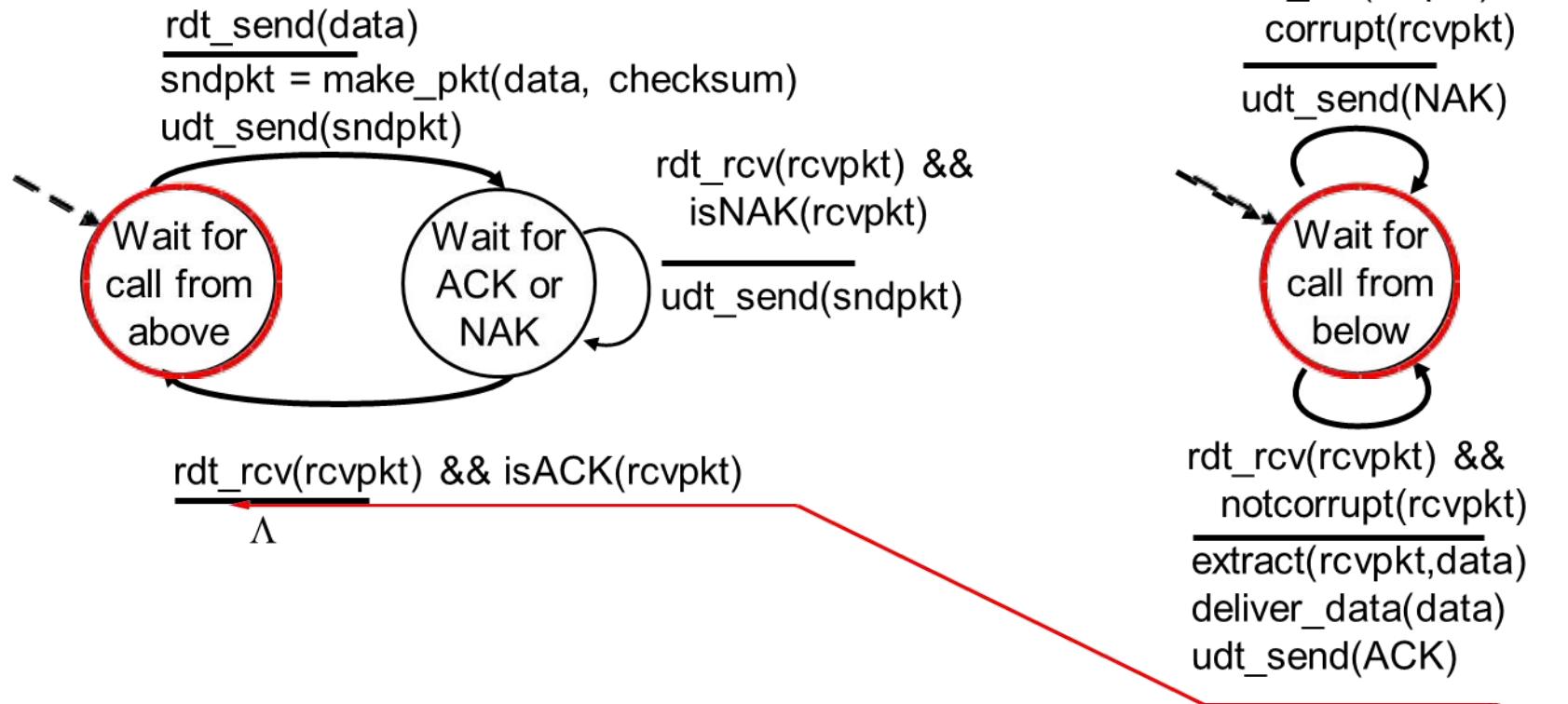


rdt2.0: error scenario: corrupted packet

receiver
sender



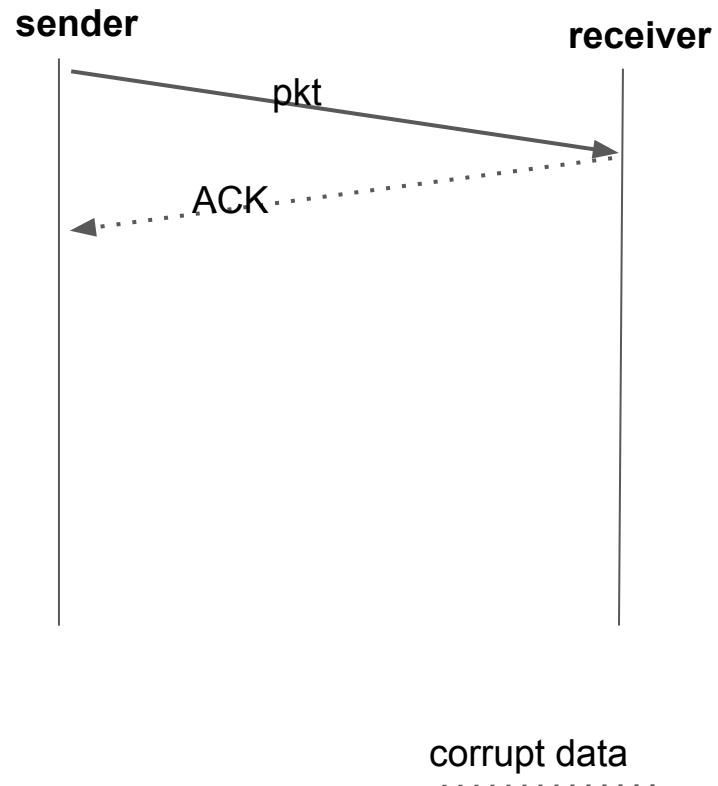
rdt2.0: error scenario: corrupted packet **receiver** **sender**



rdt2.0 has a fatal flaw!

what happens if ACK / NAK corrupted?

sender doesn't know what happened at receiver!

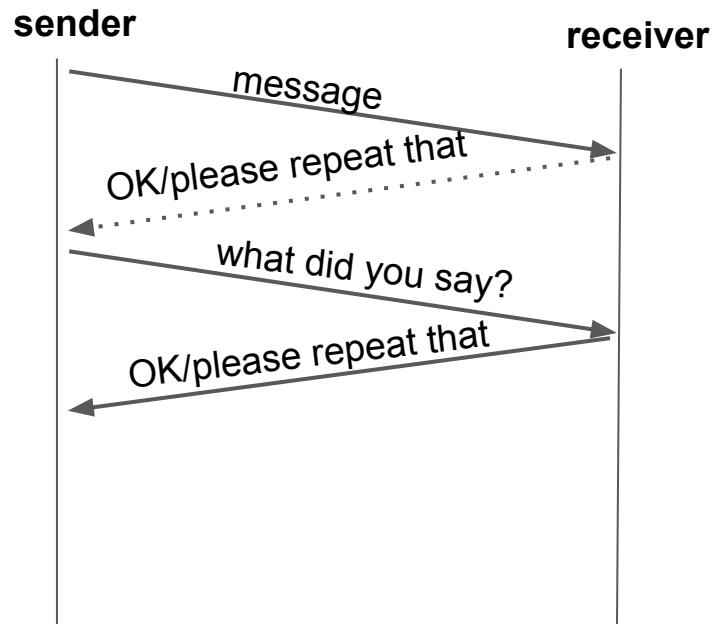


rdt2.0 has a fatal flaw!

what happens if ACK / NAK corrupted?

sender doesn't know what happened at receiver!

human analogy?



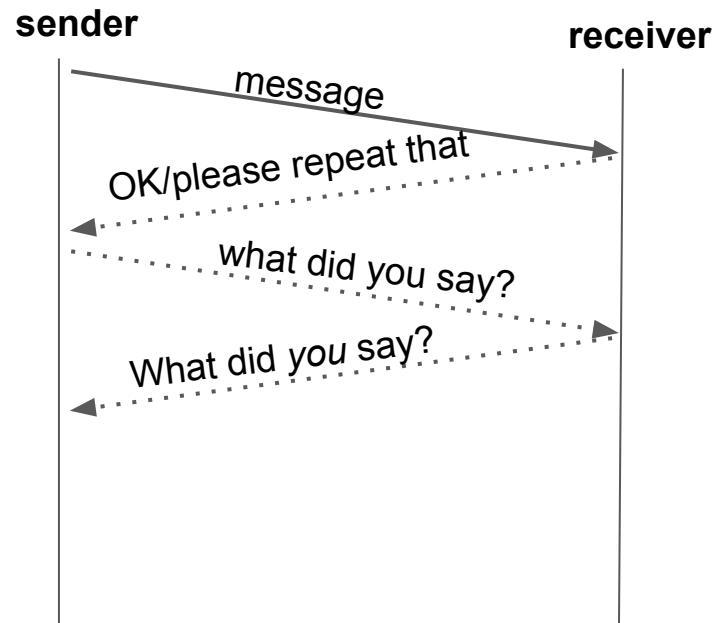
rdt2.0 has a fatal flaw!

what happens if ACK / NAK corrupted?

sender doesn't know what happened at receiver!

human analogy?

receiver has no idea whether the garbled sentence was part of the message or a request to repeat the last reply, probably respond with “what did you say”, that response may be garbled



rdt2.0 has a fatal flaw!

what happens if ACK / NAK corrupted?

Possible Solution: send enough checksum bits to allow the sender not only to detect, but also to recover from bit errors.

NO

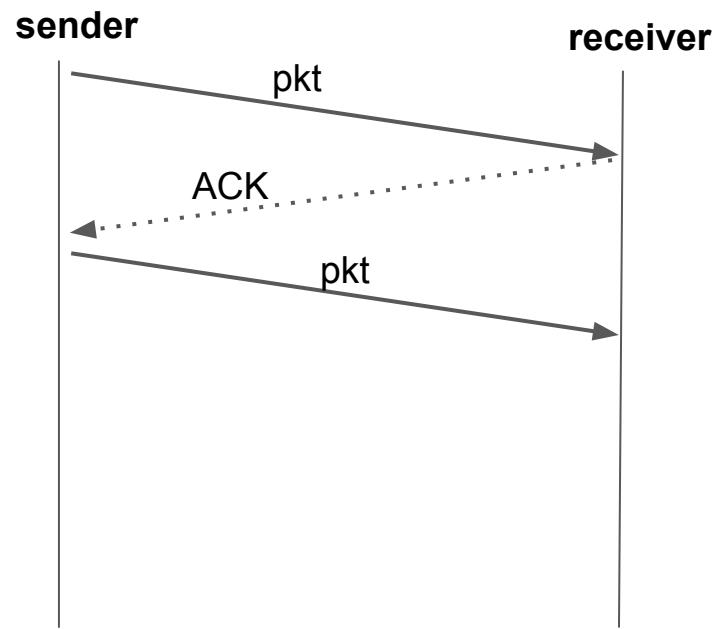
Problem: does not work if the channel does lose the packet

rdt2.0 has a fatal flaw!

what happens if ACK / NAK corrupted?

Possible Solution: re-send current packet when a garbled ACK/NACK is received.

Problem: Can't just retransmit: possible **duplicate**. The receiver does not know the new packet is a new data or retransmission



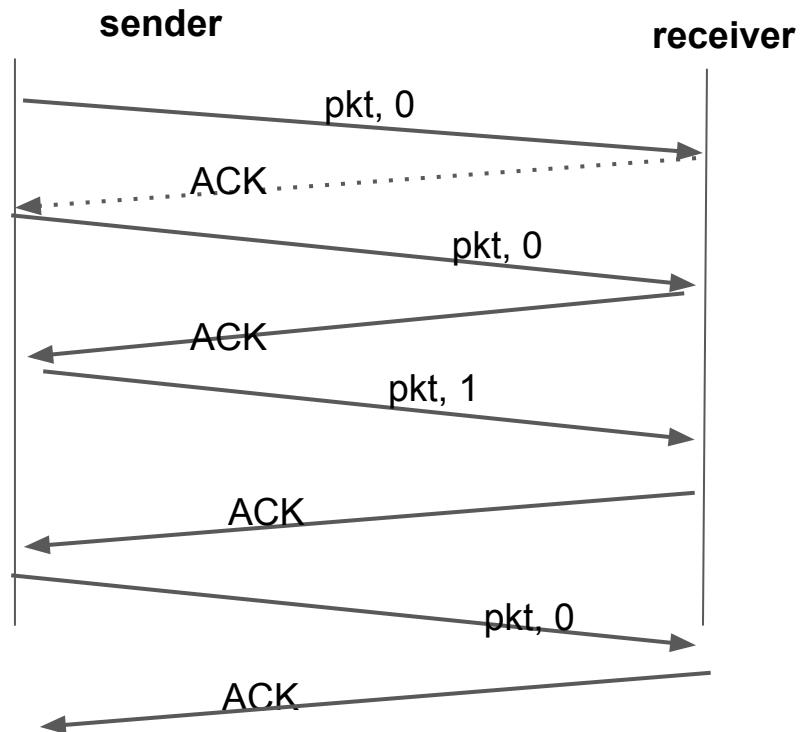
rdt2.0 has a fatal flaw!

what happens if ACK / NAK corrupted?

Solution: re-send current packet when a garbled ACK/NACK is received. The sender adds a **sequence number** to its data packet

if the sequence number of the received packet has the same sequence number as the most recently received packet → retransmitted packet

the sequence number has changed → new packet



rdt2.0 has a fatal flaw!

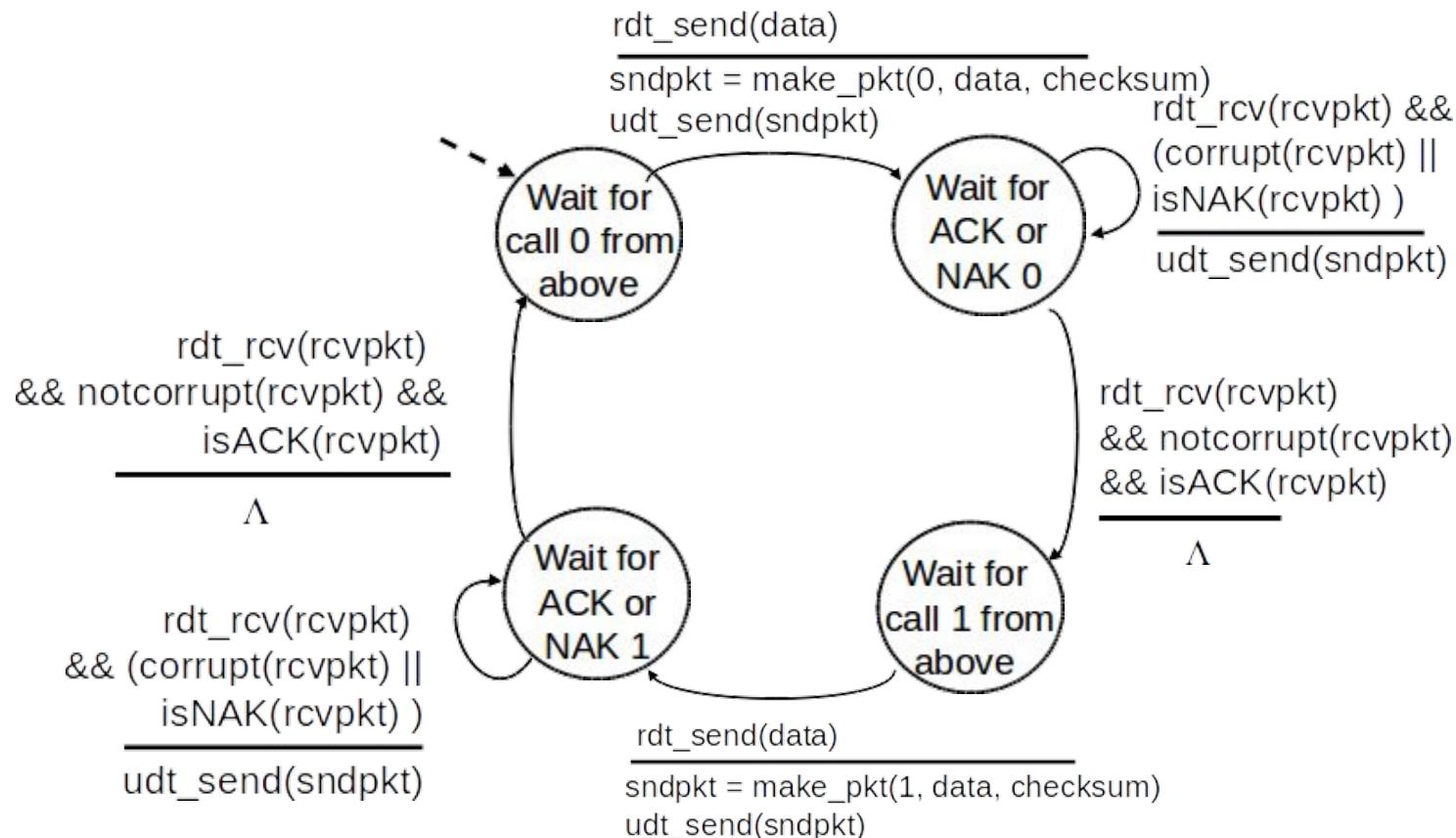
- **what happens if ACK / NAK corrupted?**
 - **Retransmission** and sequence number
 - How sequence number makes it possible to differentiate retransmitted packet and new packets?
 - if the sequence number of the received packet has the same sequence number as the most recently received packet → retransmitted packet
 - If the sequence number has changed → new packet
 - ACK and NACK do not need to indicate the sequence number of packet they are acknowledging since the channel does not lose packet. The sender knows that a received ACK or NACK was generated in response to its most recently transmitted data

rdt2.0 has a fatal flaw!: Summary

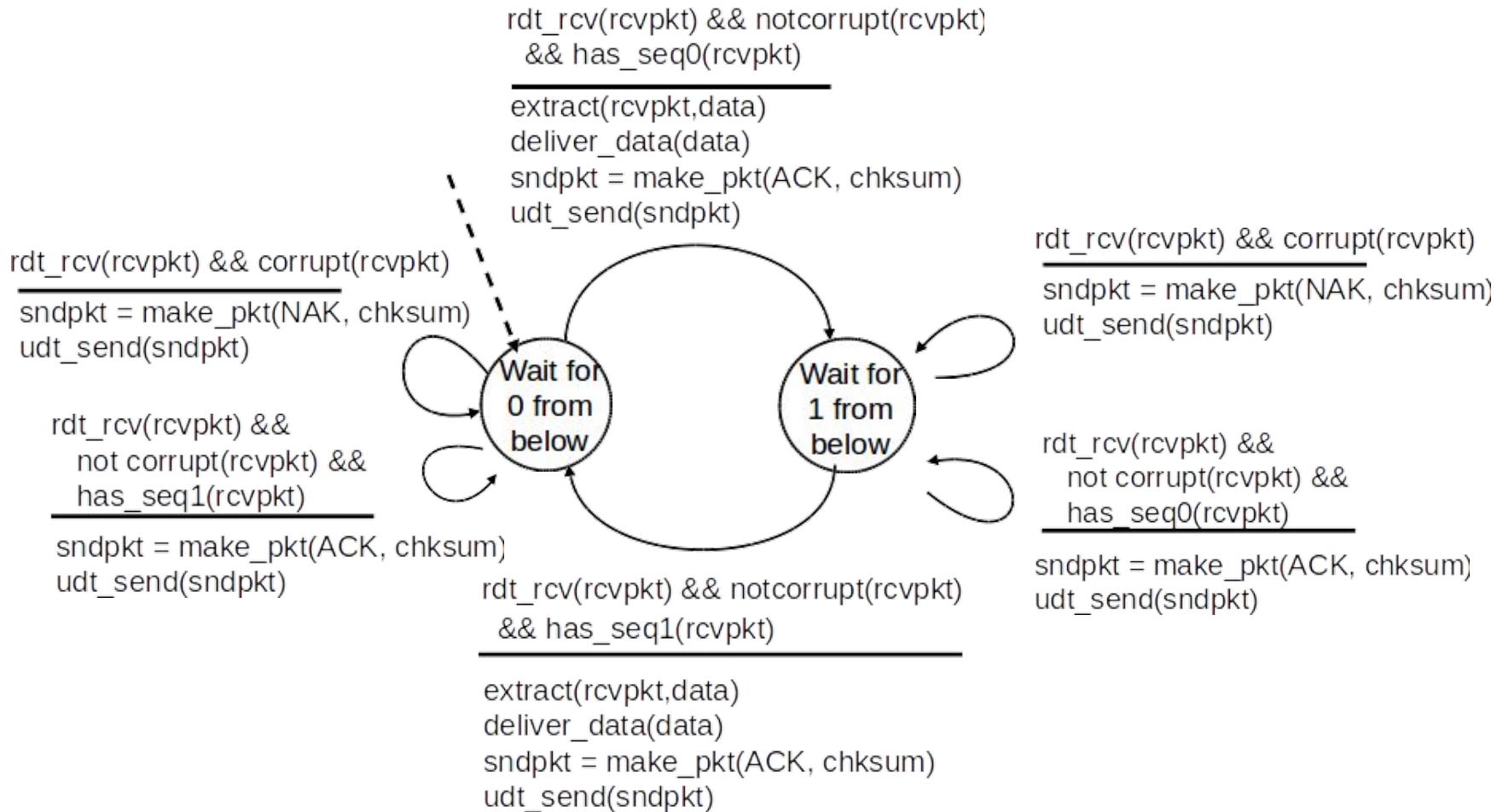
- **what happens if ACK/NAK corrupted?**
 - sender doesn't know what happened at receiver!
 - can't just retransmit: possible duplicate
- **handling duplicates:**
 - sender **retransmits** current pkt if ACK/NAK corrupted
 - sender adds **sequence number** to each pkt
 - receiver discards (doesn't deliver up) duplicate pkt

stop and wait: sender sends one packet, then waits for receiver response

rdt2.1: sender, handles garbled ACK/NACKs



rdt2.1: receiver, handling garbled ACK/NAKs



rdt2.1: discussion

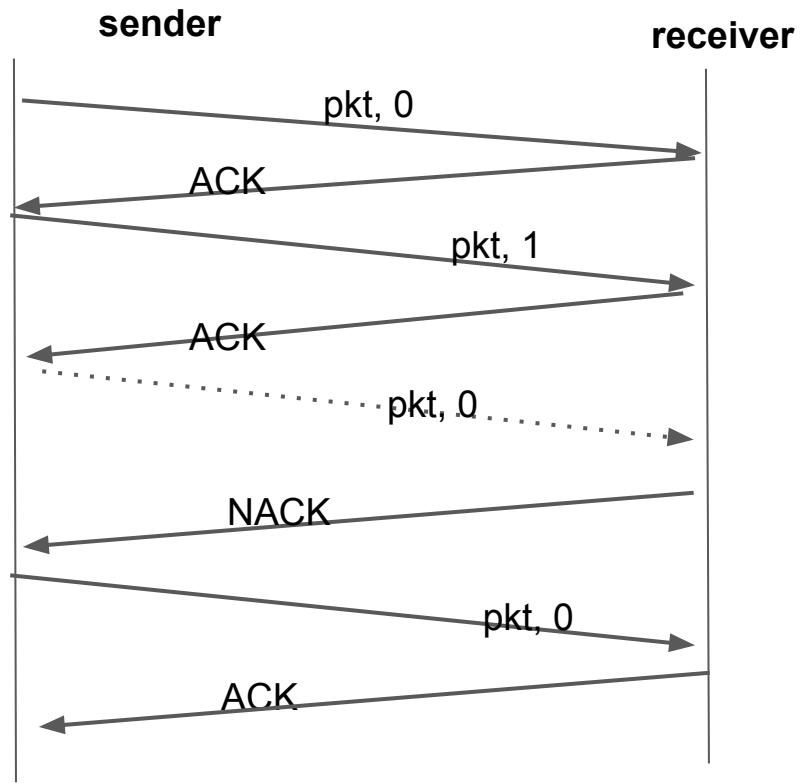
- Sender:
 - seq # added to pkt
 - two seq. #'s (0,1) will suffice. Why?
 - must check if received ACK/NAK corrupted
 - twice as many states
 - state must “remember” whether “expected” pkt should have seq # of 0 or 1
- receiver:
 - must check if received packet is duplicate
 - state indicates whether 0 or 1 is expected pkt seq #
 - note: receiver can not know if its last ACK/NAK received OK at sender

rdt2.1: A NAK-based Protocol

Can we change it to a NACK-free protocol?

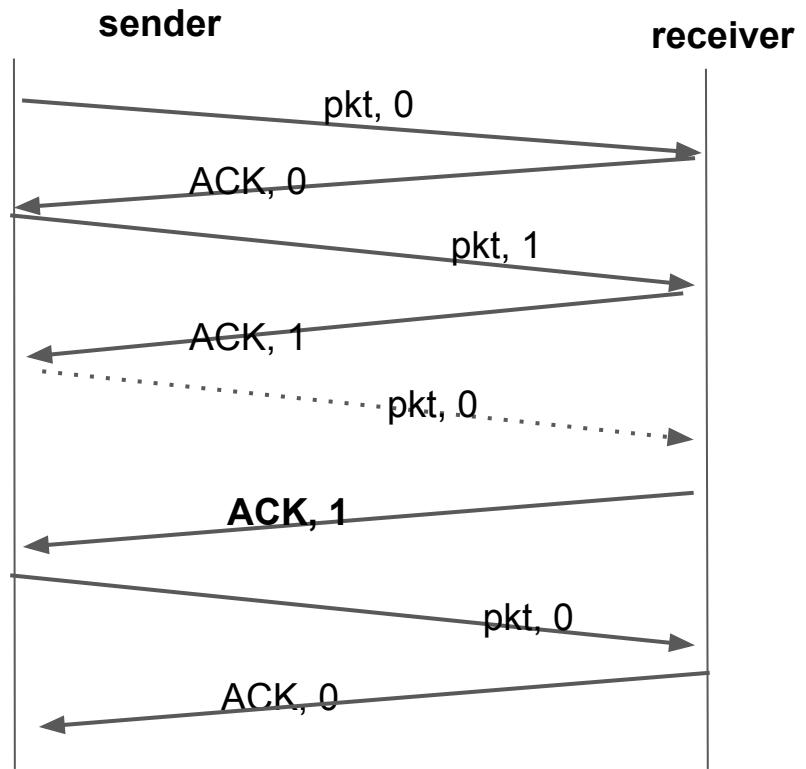
Yes, Instead of sending a NACK,
we send an ACK for **the last
correctly received packet**.

**receiver must explicitly include
seq # of pkt being ACKed**



rdt2.2: A NAK-free Protocol

A sender that receives two ACKs for the same packet (**duplicate ACKs**) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice.

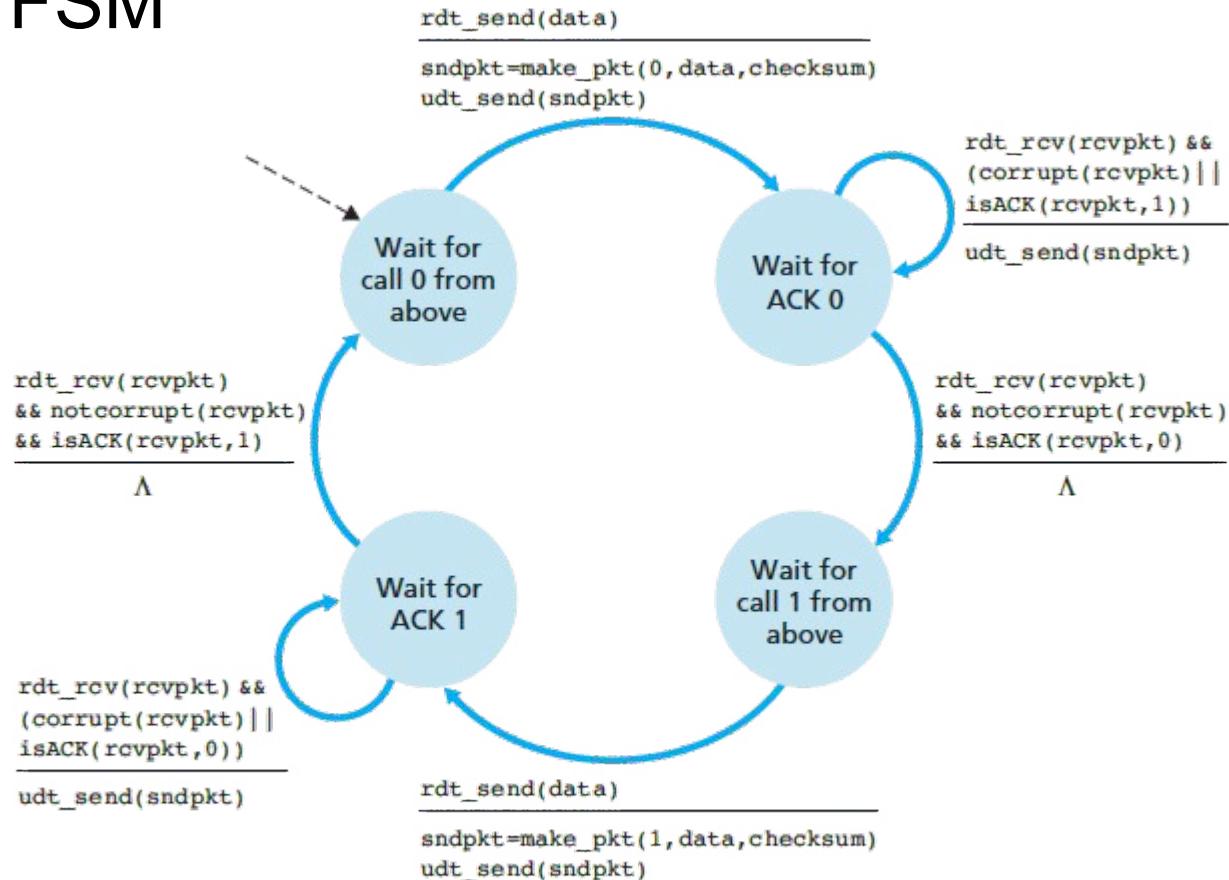


rdt2.2: a NAK-free protocol

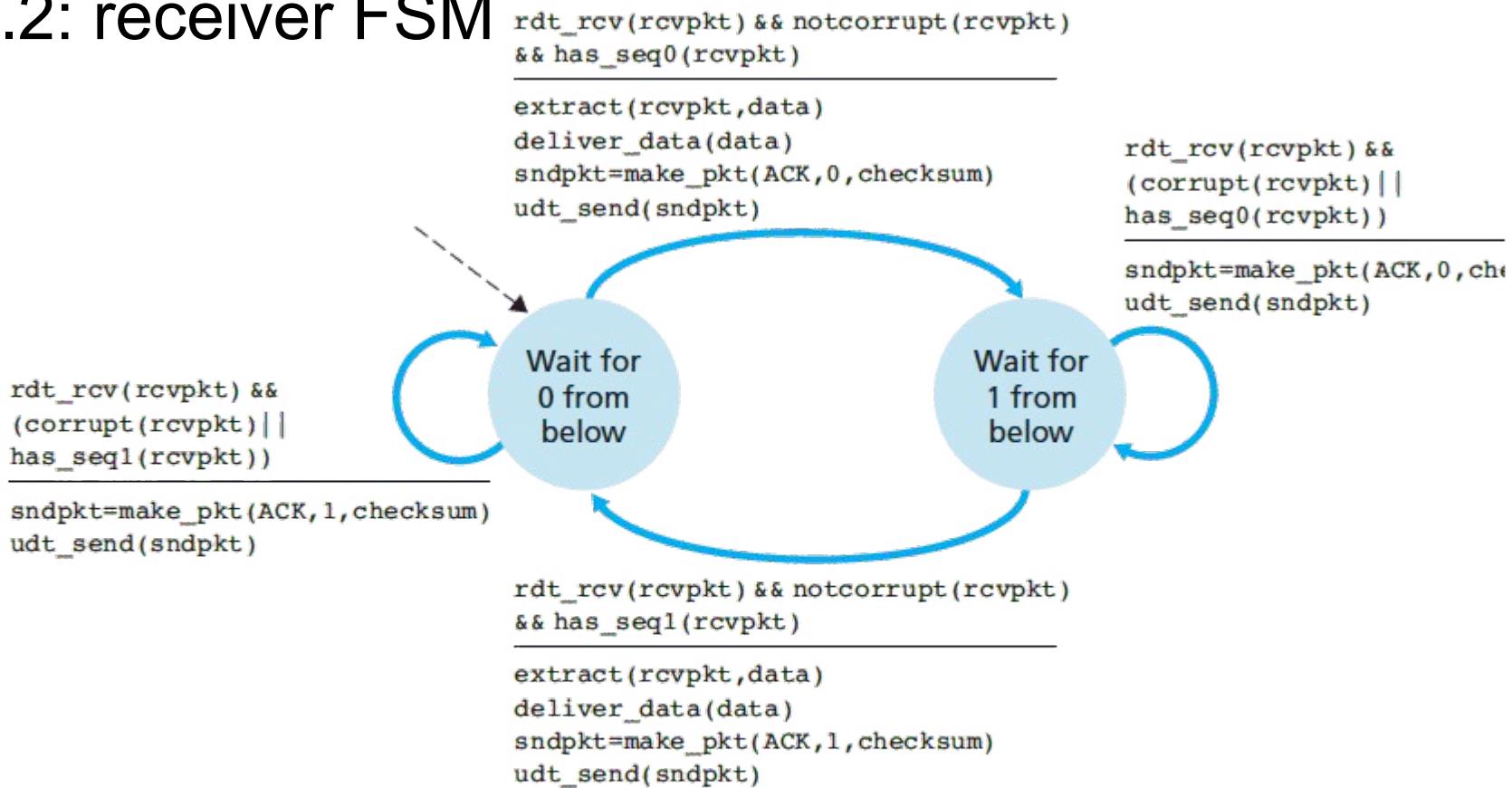
- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
 - receiver must **explicitly** include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: **retransmit current pkt**

As we will see, TCP uses this approach to be NAK-free

rdt2.2: sender FSM



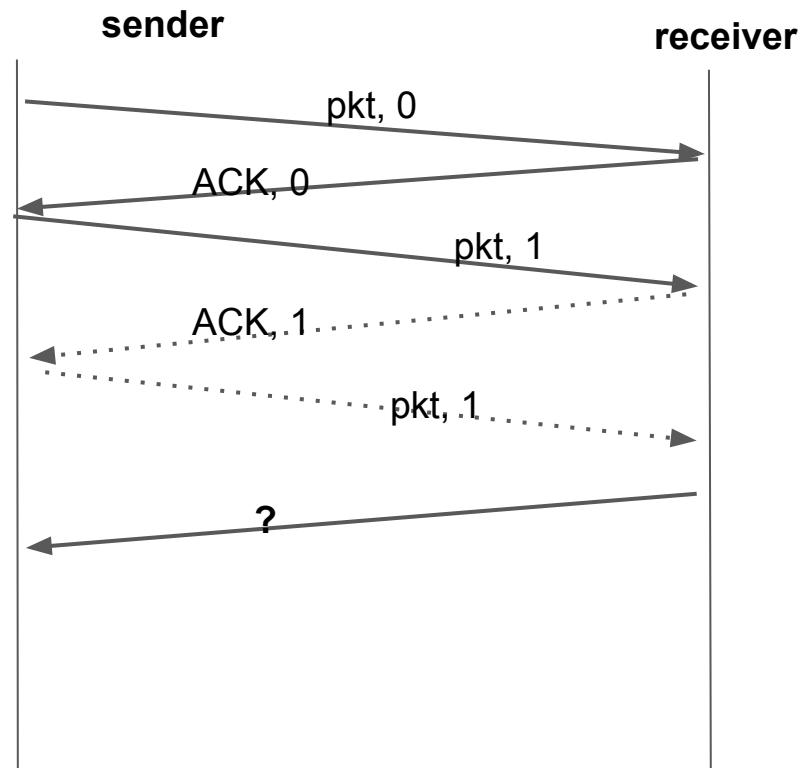
rdt2.2: receiver FSM



Question

The diagram on the right shows the operation of rdt2.2? What should go in place of “?” ?

- A. ACK, 1
- B. ACK, 0
- C. ACK



rdt3.0: channels with errors and loss

- **new channel assumption:** underlying channel can also lose packets (data, ACKs)
 - How to detect packet loss
 - What to do when packet loss occurs
 - checksum, seq. #, ACKs, retransmissions will be of help ... but not enough for detecting loss

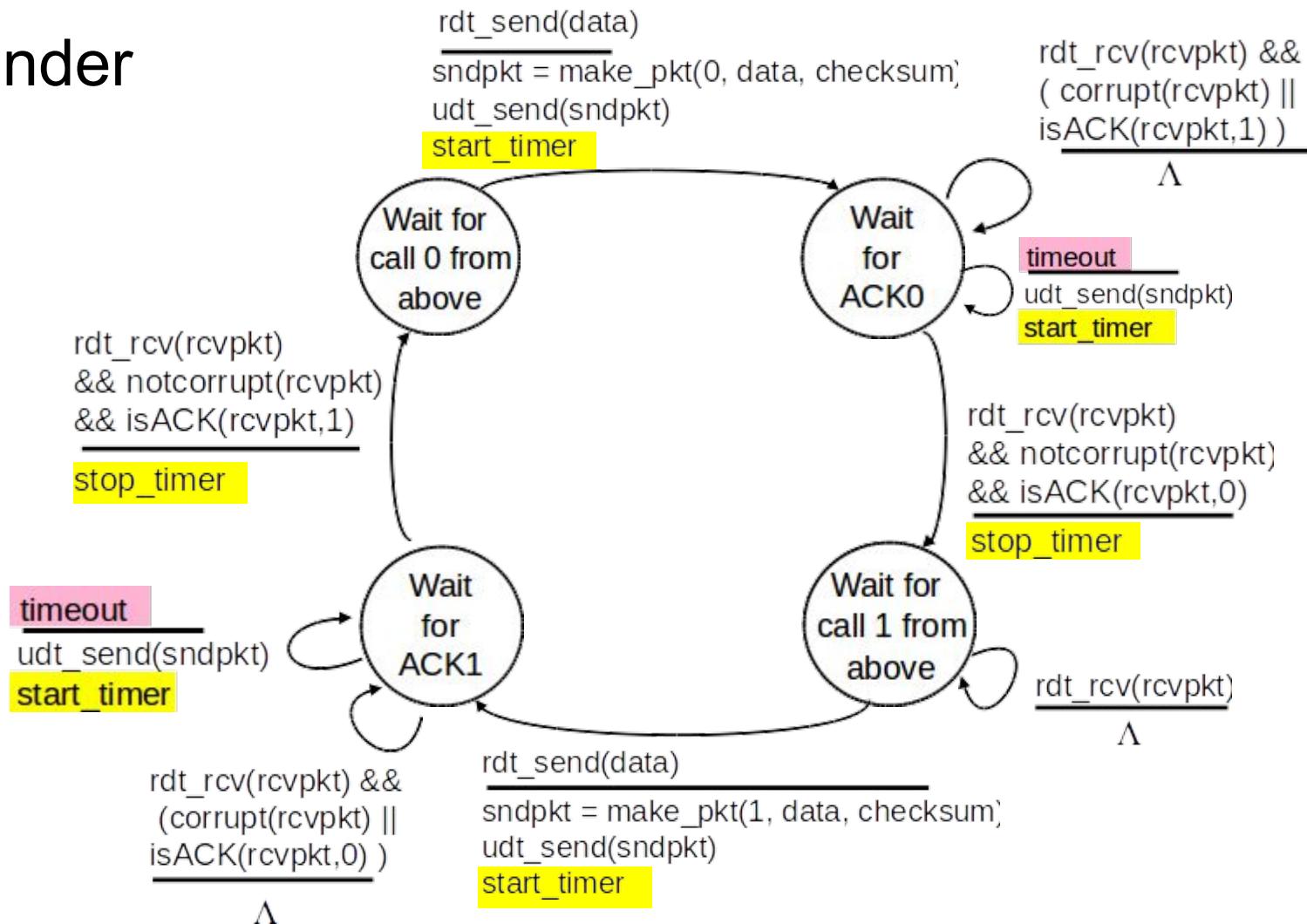
Q: How do humans handle lost sender-to-receiver words in conversation?

rdt3.0: channels with errors and loss

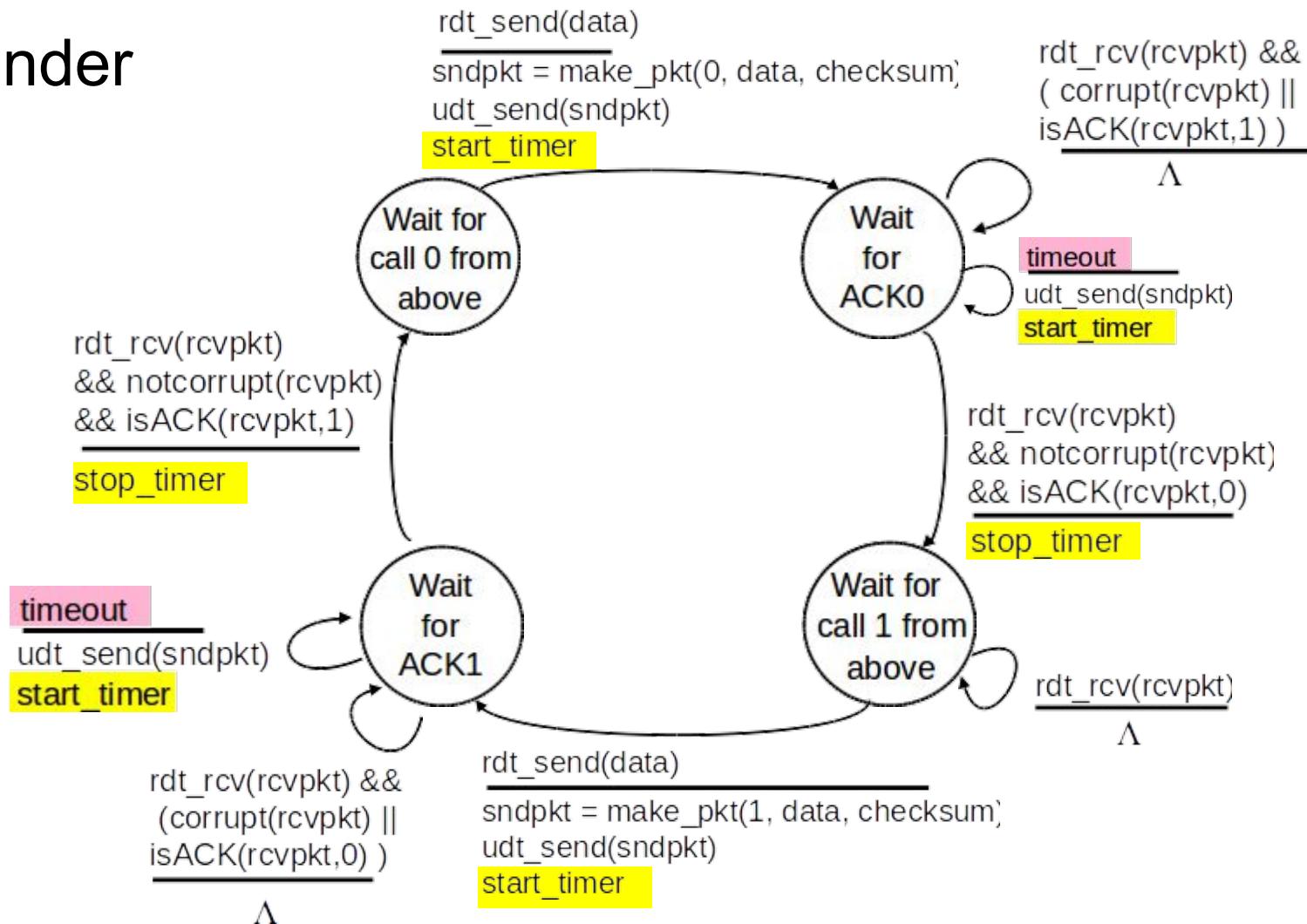
- **Approach:** sender waits “reasonable” amount of time for ACK
 - retransmits if no ACK received in this time
 - if pkt (or ACK) just delayed (not lost):
 - retransmission will be duplicate, but seq. #'s already handles this
 - receiver must specify seq # of pkt being ACKed
 - use countdown timer to interrupt after “reasonable” amount of time



rdt3.0 sender



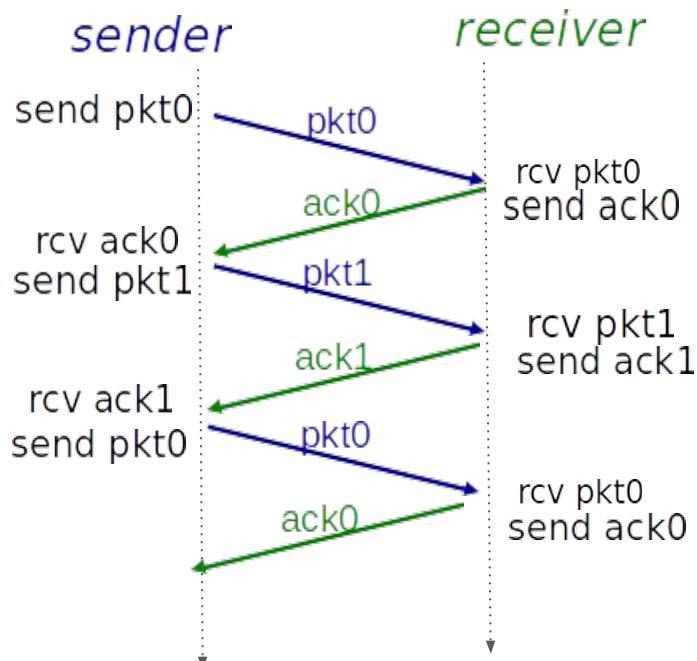
rdt3.0 sender



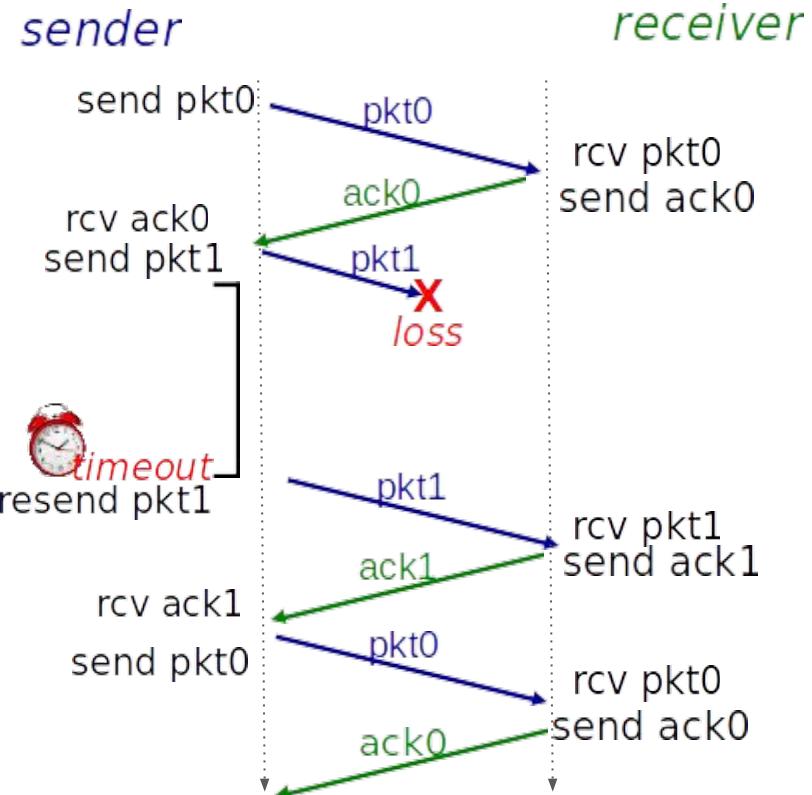
Rdt3.0 receiver

The same as rdt2.2 receiver

rdt3.0 in action

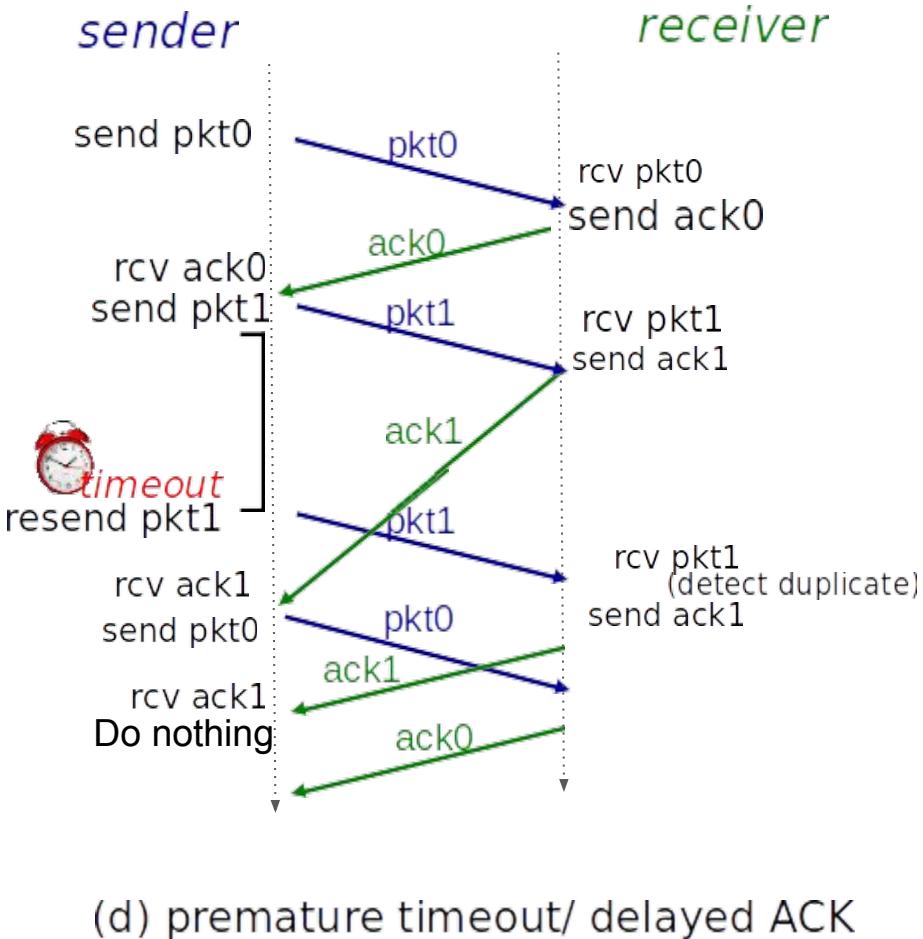
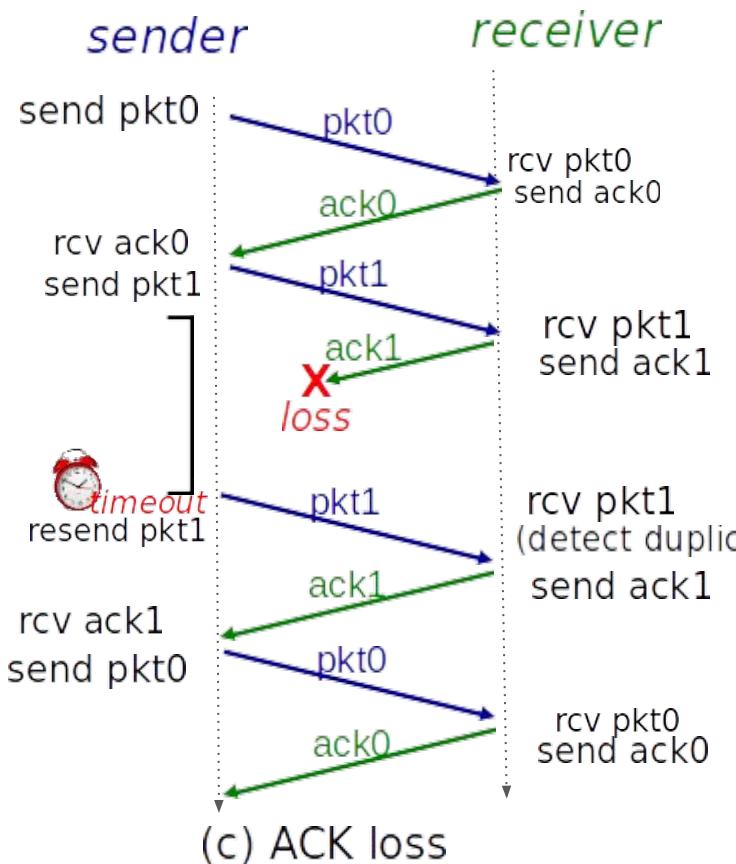


(a) no loss



(b) packet loss

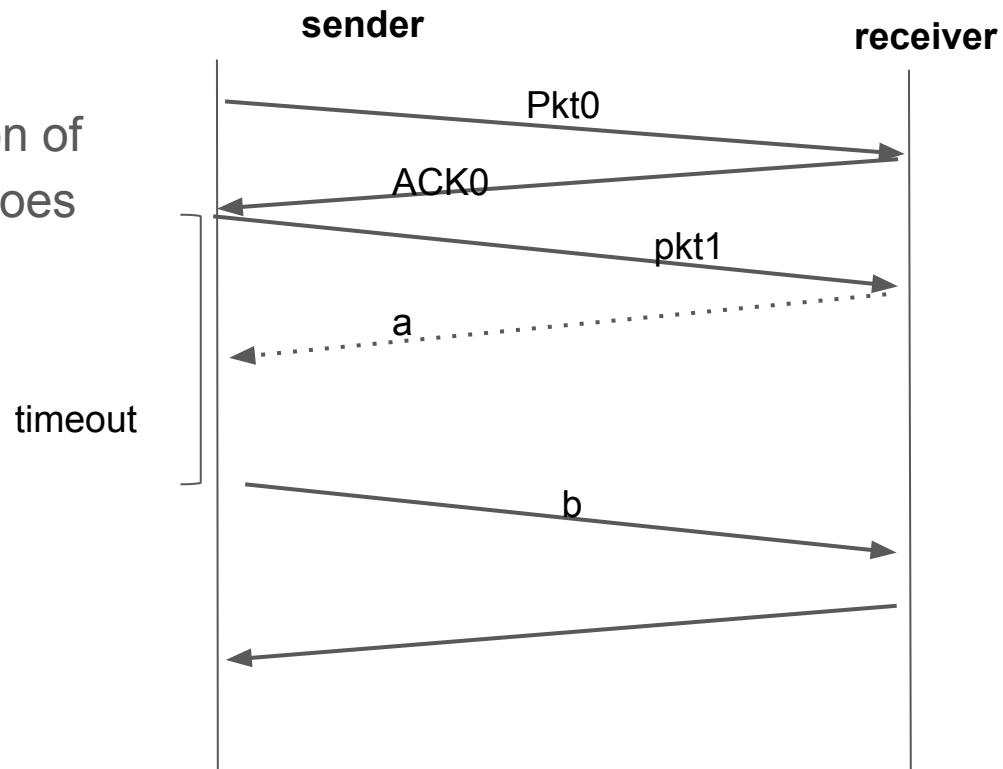
rdt3.0 in action



Question

In the diagram showing the operation of rdt3.0 in case of corrupt ack, what goes in place of a and b respectively?

- A. Ack1 , pkt1
- B. Ack0, pkt0
- C. Ack1, pkt0
- D. Ack0, pkt1



outline

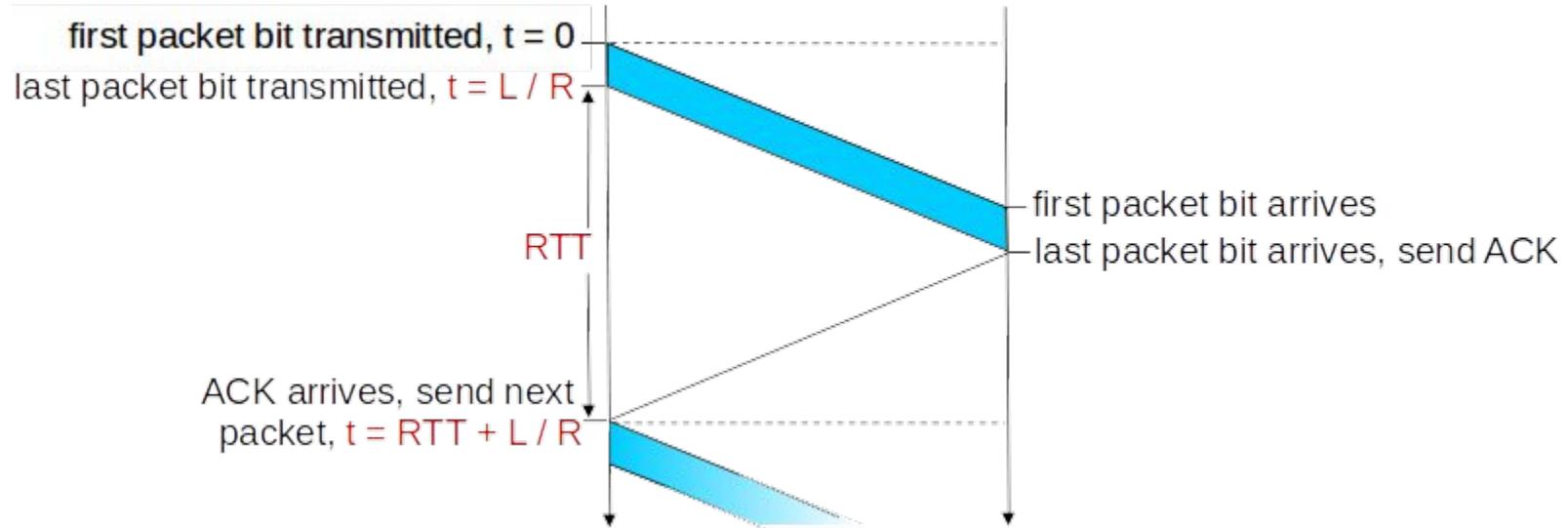
- transport-layer services
- multiplexing and demultiplexing
- connectionless transport: UDP
- **principles of reliable data transfer (continued)**
- connection-oriented transport: TCP
- principles of congestion control
- TCP congestion control

Stop and Wait

At most one packet in flight at any time

- Sender sends one packet
- Receiver sends acknowledgment packet when it receives data
- On receiving acknowledgment, sender sends new data
- On timeout, sender resends current data
- Use 1-bit counter to detect duplicates

rdt3.0: stop-and-wait operation



U_{sender} : utilization – fraction of time
sender busy sending

$$U_{\text{sender}} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}}$$

Performance of rdt3.0

- rdt3.0 is correct, but performance stinks

- example: 1 Gbps link;
- 15 ms prop. Delay,
- RTT = 30 msec;
- packet size = 1 KB = 8000 bit

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits / packet}}{10^9 \text{ bits / sec}} = 8 \text{ microseconds}$$

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

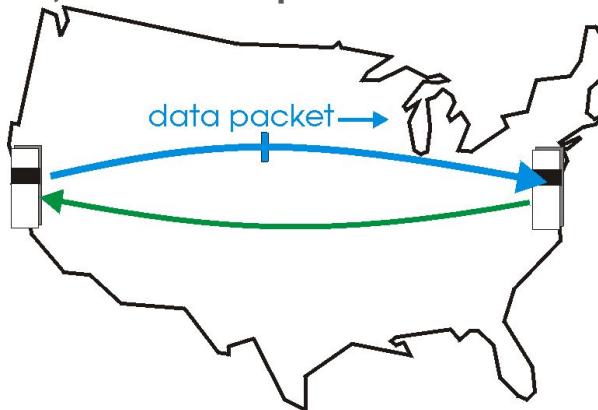
- If RTT=30 msec, 1KB pkt every 30 msec: 33KB/sec thruput over 1 Gbps link,
- Protocol limits performance of underlying infrastructure (channel)**

Throughout: $\frac{1000 \text{ bytes}}{30.008 \text{ msec}} = 267 \text{ kbps}$

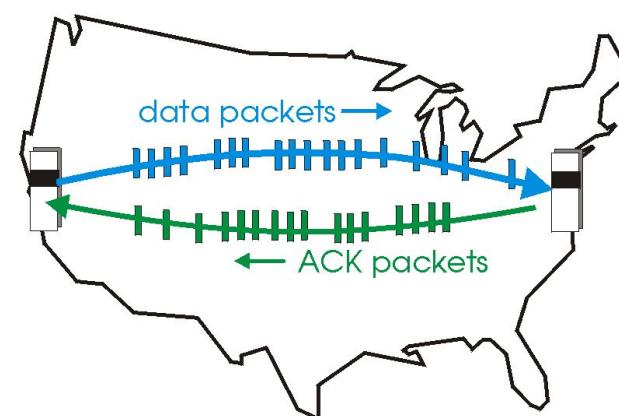
$$267 \text{ kbps} = 33 \text{ kB/sec}$$

Pipelined protocols

- pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts
 - range of sequence numbers must be increased
 - buffering at sender and/or receiver
- two generic forms of pipelined protocols:
 - go-Back-N, selective repeat

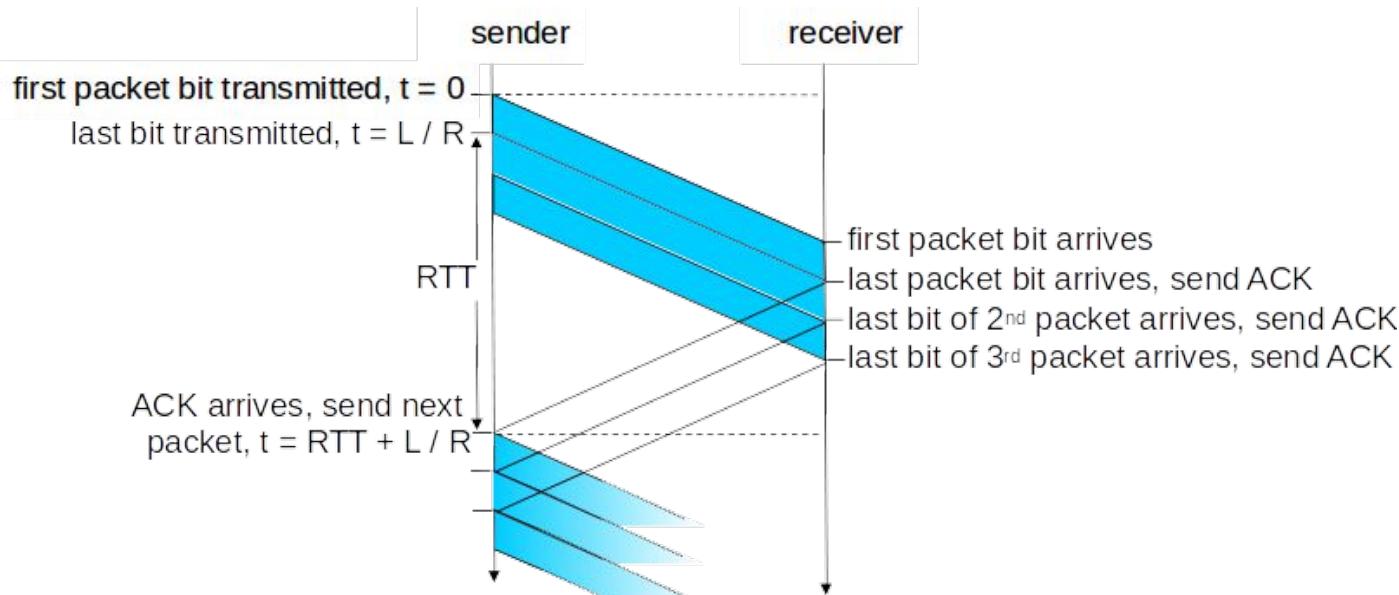


(a) a stop-and-wait protocol in operation



(b) a pipelined protocol in operation

Pipelining: increased utilization



$$U_{\text{sender}} = \frac{\frac{3L}{R}}{RTT + \frac{L}{R}} = \frac{.0024}{30.008} = 0.00081$$

3-packet pipelining increases utilization by a factor of 3!

Question

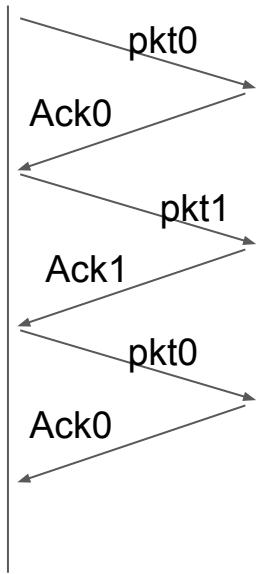
In the previous example, how big would the window size have to be for the channel utilization to be greater than 98 percent?

Packet size = 1500 bytes

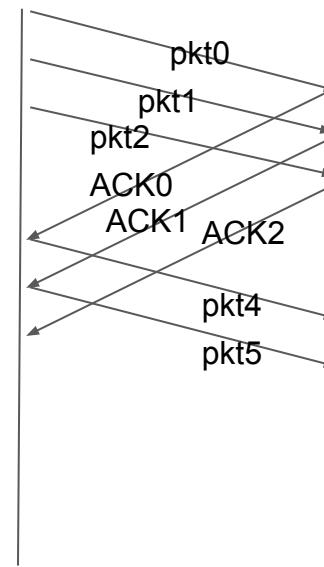
- A. 245
- B. 2451
- C. 3675
- D. 3676

stop-and-wait vs sliding window

sender

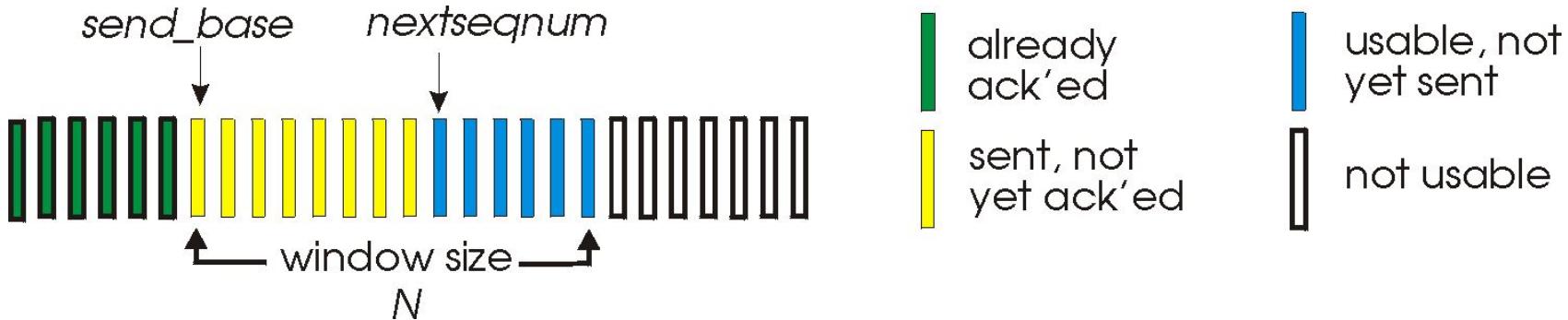


receiver



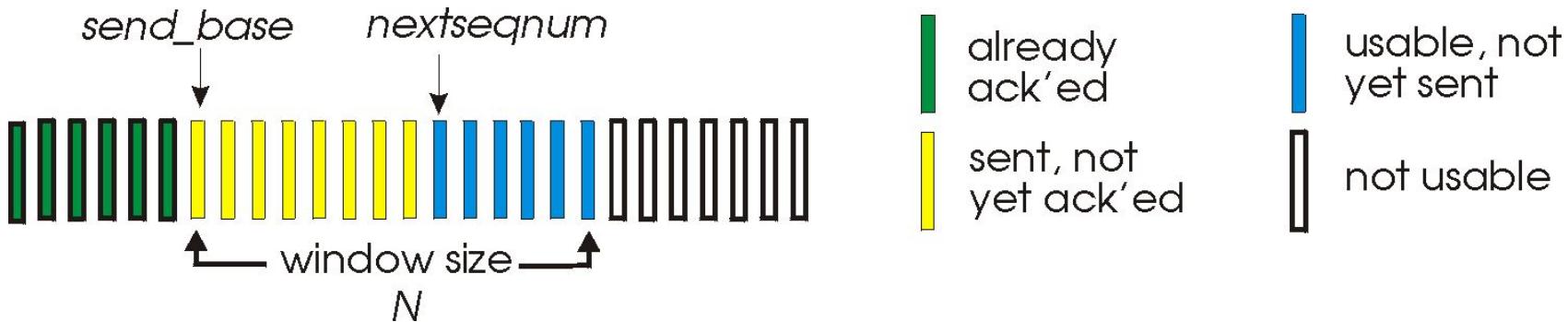
Go-Back-N: Sender

- sender can have up to N unacked packets in pipeline (window size, buffer size)
- **sliding-window protocol**: As protocol operates this window slides forward over the sequence number space
- **base**: the sequence number of the oldest unacknowledged packet
- **nextseqnum**: smallest unused sequence number: sequence number of the next packet to be sent
 - At any time: $\text{nextseqnum} - \text{base} \leq \text{window size}$



Go-Back-N: Sender

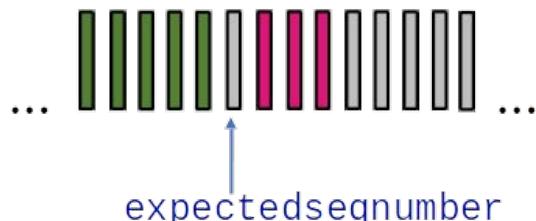
- **cumulative ACK: ACK(n)**: ACKs all pkts up to, including seq # n
 - on receiving ACK(n): move window forward to begin at n+1
 - Sender may receive duplicate ACKs
- timer for oldest in-flight pkt
 - **timeout(n)**: retransmit packet n and all higher seq # pkts in window



GBN: Receiver

- ACK-only: always send ACK for correctly-received packet with highest **in-order seq #**
 - send **cumulative acks**: if received 1,2,3, 5, acknowledges three
 - may generate duplicate ACKs
 - need only remember **expectedseqnum**: sequence number of the next in-order packet
- On receipt of out-of-order pkt:
 - can discard (don't buffer) or buffer: an implementation decision
 - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:



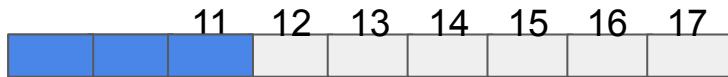
received and ACKed

Out-of-order: received but not ACKed

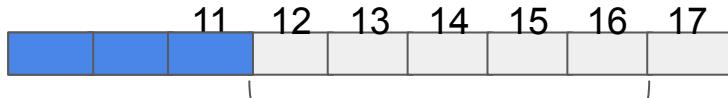
Not received

Go-Back-N: example

sender



Window size = 5



Window size = 5



Window size = 5

receiver



expectedseqnumber



expectedseqnumber



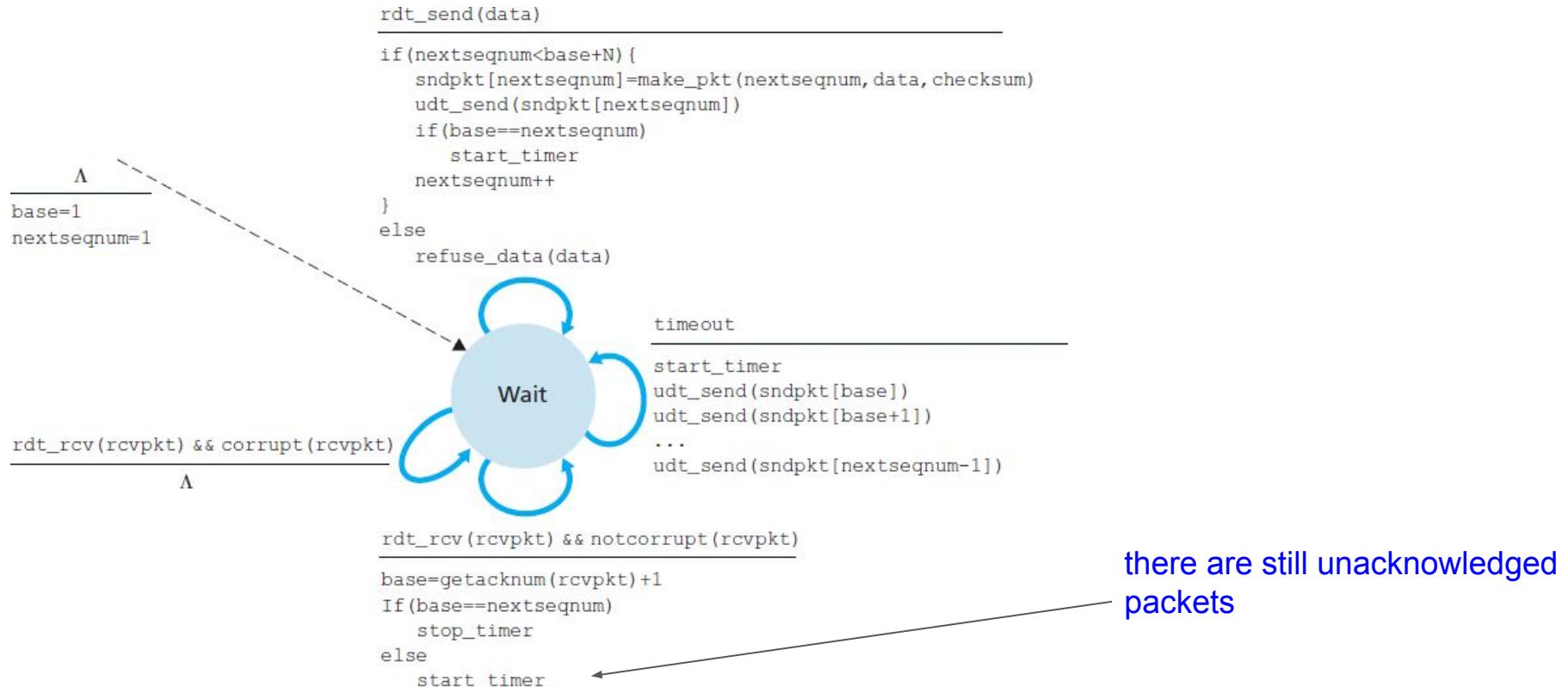
expectedseqnumber

Go-Back-N

Animation:

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html

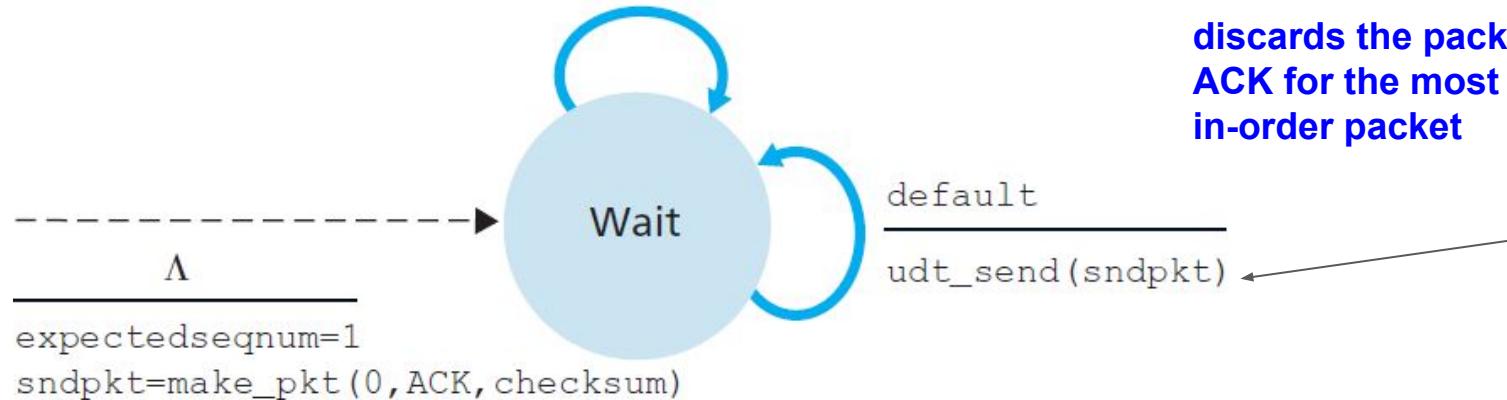
GBN: sender extended FSM



GBN: receiver extended FSM

```
rdt_rcv(rcvpkt)
  && notcorrupt(rcvpkt)
  && hasseqnum(rcvpkt, expectedseqnum)
```

```
extract(rcvpkt, data)
deliver_data(data)
sndpkt=make_pkt(expectedseqnum, ACK, checksum)
udt_send(sndpkt)
expectedseqnum++
```



discards the packets and resend an
ACK for the most recently received
in-order packet

GBN in Action

sender window (N=4)

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)



pkt 2 timeout

send pkt2
send pkt3
send pkt4
send pkt5

receiver

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, discard,
(re)send ack1

receive pkt4, discard,
(re)send ack1

receive pkt5, discard,
(re)send ack1

rcv pkt2, deliver, send ack2
rcv pkt3, deliver, send ack3
rcv pkt4, deliver, send ack4
rcv pkt5, deliver, send ack5

GBN: Receiver

- re-ACK pkt with highest in-order seq # that is correctly received
- out-of-order pkt:
 - discard (don't buffer): **no receiver buffering!**
 - cons:
 - Waste
 - the re-transmitted packet might get lost or garbled
 - Pros:
 - simplicity of receiver buffering
 - No buffering of out-of-order packets

Question

With GBN, it is possible for the sender to receive an ACK for a packet that fall outside of its current window

- True
- False

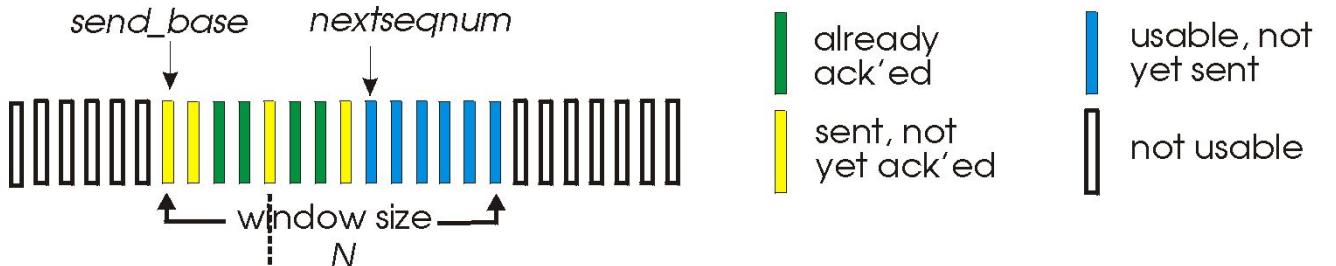
GBN

- avoid the channel utilization problem with stop-and-wait protocol
- **performance problems**
 - window size and bandwidth delay product are large
 - a single packet error causes GBN to retransmit a large number of packets, many unnecessarily
- **Solution:** To avoid retransmission problem
 - selective repeat
 - a sliding window protocol

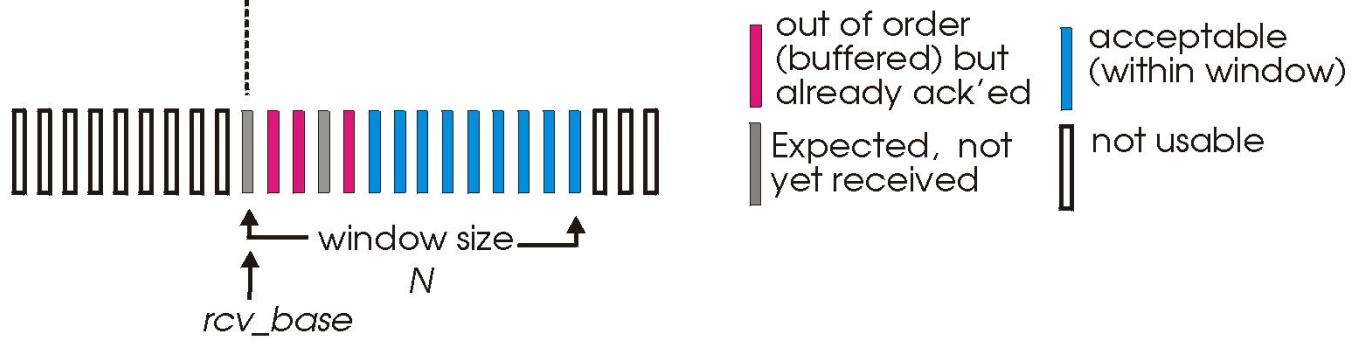
Selective repeat

- receiver **individually** acknowledges all correctly received pkts
 - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
- sender maintains timer for each unACKed pkt
- sender window
 - N consecutive seq #'s
 - limits seq #'s of sent, unACKed pkts

Selective repeat: sender, receiver windows



(a) sender view of sequence numbers



(b) receiver view of sequence numbers

Selective repeat: sender

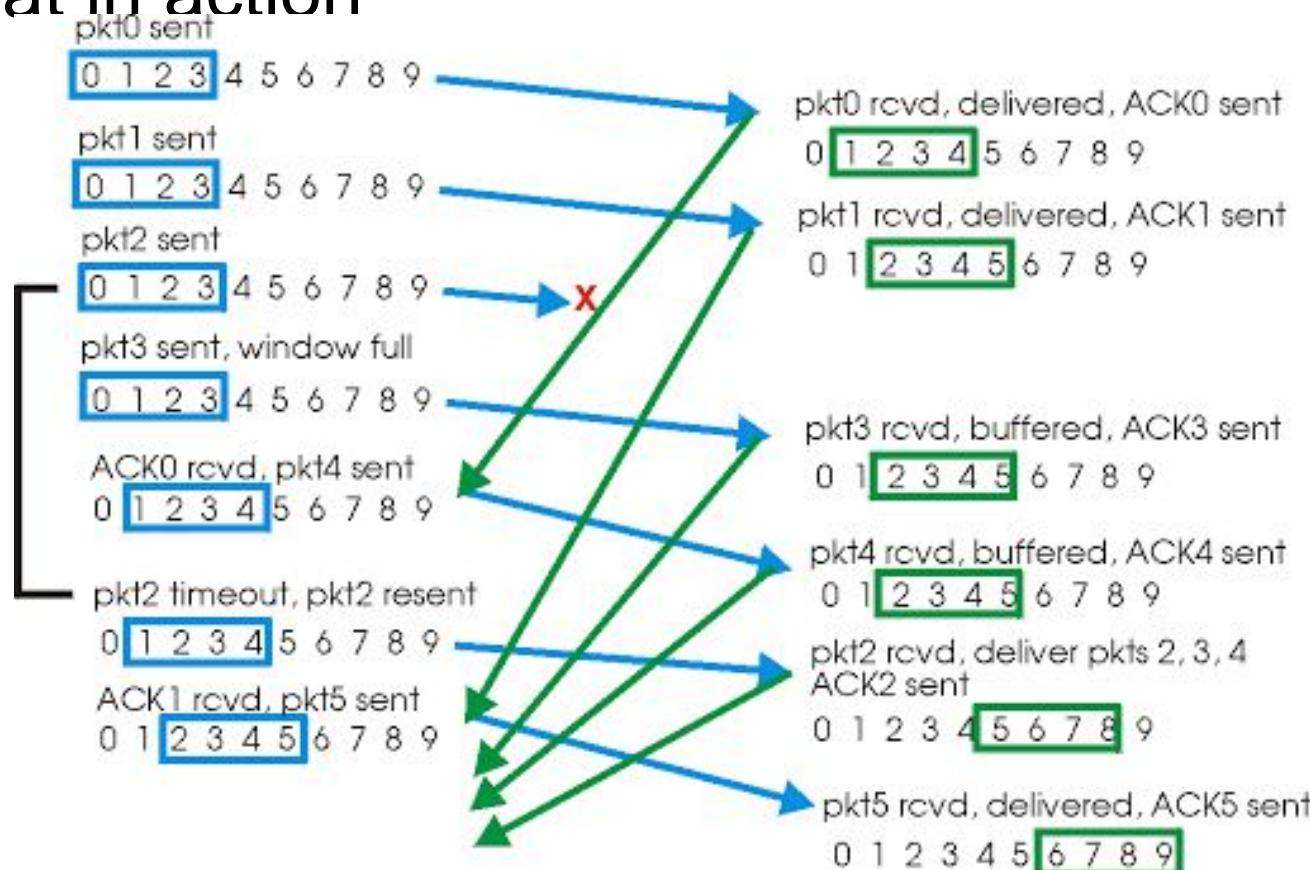
- **data from above:**
 - if next available seq # in window, send pkt
- **timeout(n):**
 - resend pkt n, restart timer
- **An acknowledgement received with a sequence number n in [sendbase,sendbase+N]:**
 - packet with sequence number n is marked received
 - if n is the smallest unACKed packet, advance window base to next unACKed seq #

Selective repeat: Receiver

- **packet with sequence number n is received**
 - **n is in [rcvbase, rcvbase+N-1]**
 - send ACK(n)
 - out-of-order: buffer
 - in-order: deliver (also deliver buffered, in-order packets), advance window by the number of packets delivered to the upper layer
 - **pkt n in [rcvbase-N, rcvbase-1]**
 - ACK(n)
 - Ack is sent even though this is a packet acknowledged before
 - **otherwise:**
 - ignore

Selective repeat in action

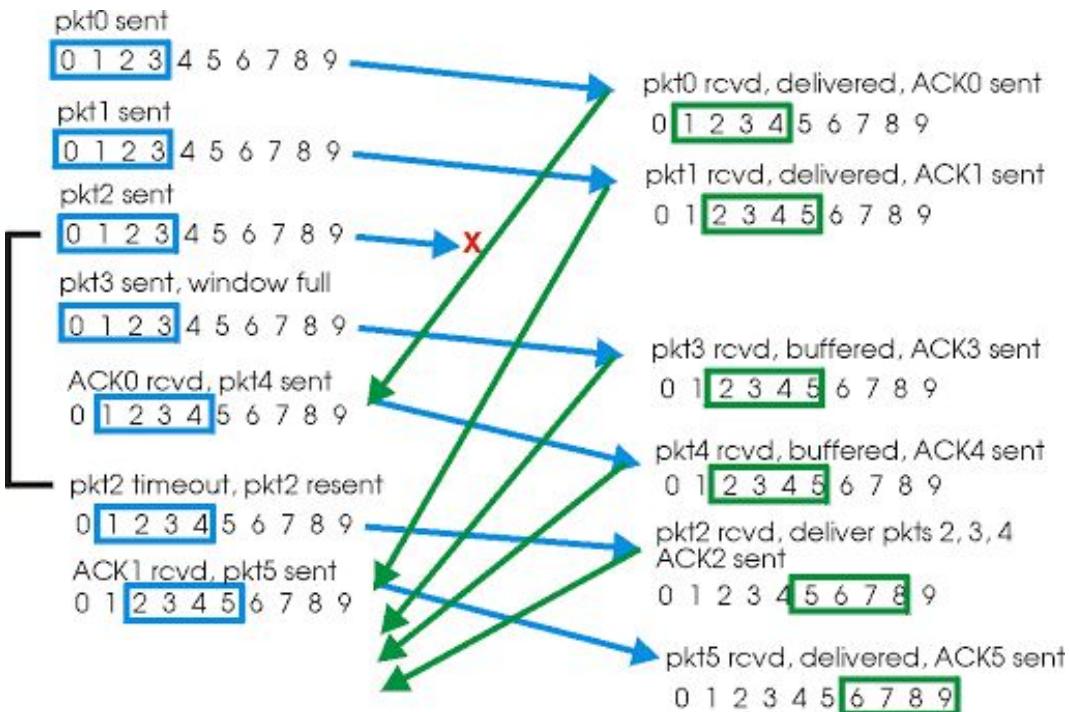
Note: the sender and receiver's window are not synchronized at any time



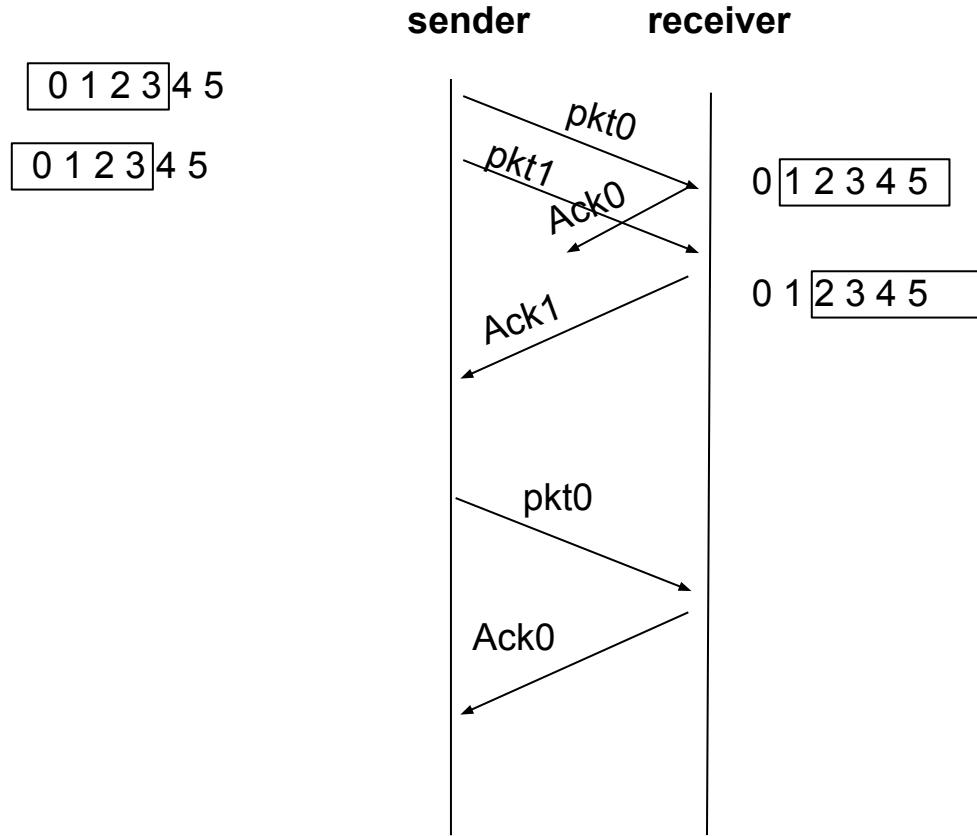
Question

What happens when Ack2 arrives?

- A. Sender moves its window forward
- B. Sender retransmit packet with seq#2
- C. Sender sends packet with seq#3
- D. Sender ignores that



Selective repeat: Receiver



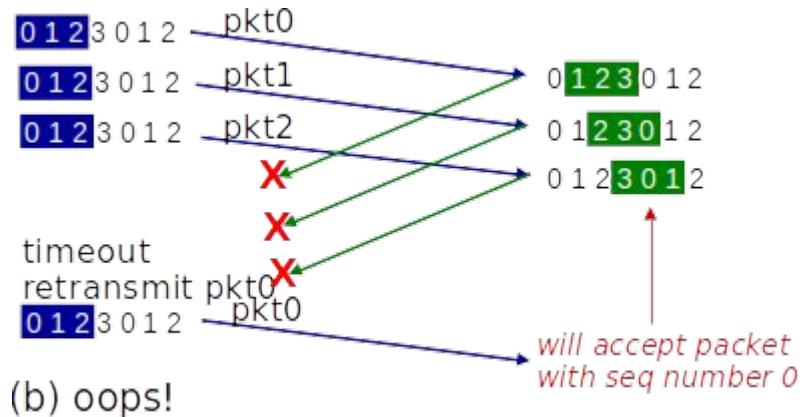
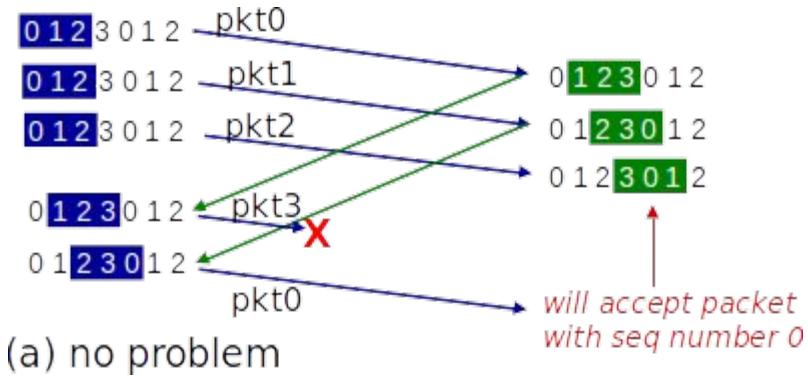
Another example of the lack of synchronization between sender and receiver in selective repeat protocol

pkt n in [rcvbase-N,rcvbase-1]
is received

Lack of synchronization between sender and receiver is problematic when the range of sequence numbers are limited.

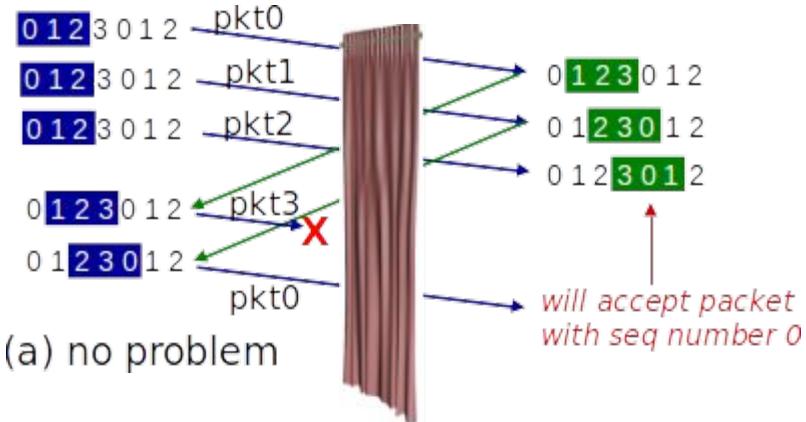
Selective repeat: dilemma

- Example:
 - seq #'s: 0, 1, 2, 3
 - window size=3
- Two scenarios: (a) , (b)

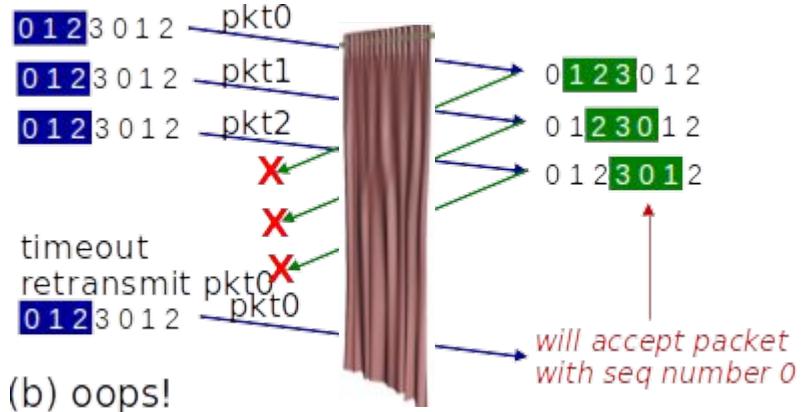


Selective repeat: dilemma

- Example:
 - seq #'s: 0, 1, 2, 3
 - window size=3
- Two scenarios: (a) , (b)
 - receiver sees no difference in two scenarios!
 - duplicate data accepted as new in (b)
- Q: what relationship between seq # size and window size to avoid problem in (b)?



*receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!*



Range of seq# and window size(N)

- The range of sequence number must be large enough to fit the entire receiver window and the entire sender window without overlap
- Receiver window: $[m, m+N-1]$
 - Packet $m-1$ and the $N-1$ packets before that are ACKed
 - These packets may have not been received by the senders
 - The sender's window could be $[m-N, m-1]$
- The sequence number range must be big enough to accommodate $2N$ sequence number:
 - $m+N-1 - (m-N) + 1 = 2N$
- If k bits are used to represent the sequence number then $2^k \geq 2N$

Selective repeat: Animation

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html

Pipelined protocols: summary

- **Go-back-N:**
 - sender can have up to N unacked packets in pipeline
 - receiver only sends **cumulative ack**
 - doesn't ack packet if there's a gap
 - sender has timer for oldest unacked packet
 - when timer expires, retransmit all unacked packets
- **Selective Repeat:**
 - sender can have up to N unacked packets in pipeline
 - receiver sends **individual ack** for each packet
 - sender maintains timer for each unacked packet
 - when timer expires, retransmit only that unacked packet

Unreliable Channels: re-ordering

- Reordering packets is solved by buffering
- **Challenge:**
 - A channel that reorder packets is equivalent to a channel that keeps the packet and spontaneously emit them at **any** point in the future.
 - old copies of a packet with a sequence or acknowledgement number of x can appear, even though neither the sender's nor the receiver's window contains x
 - sequence number may be reused
- **Solution:** a sequence number is used only if the sender is sure that any previously sent packets with sequence number x are no longer in the network
 - How: A maximum packet lifetime is defined
 - each packet cannot live longer than that
 - In TCP it is 3 minutes

Reliable data transfer mechanisms and their use

- **Checksum:** detect bit errors in a transmitted packet
- **Timer:** timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Duplicate copies of a packet may be received by receiver
- **Sequence number:**
 - gaps in the sequence number of received packets indicate a lost packets. Duplicate sequence numbers allow the receiver to detect duplicate copies of a packet
- **Acknowledgement:**
 - carry the sequence number of the packet(s) being acknowledged. Could be individual or cumulative
- **Negative acknowledgement**
- **Window, pipelining**

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

outline

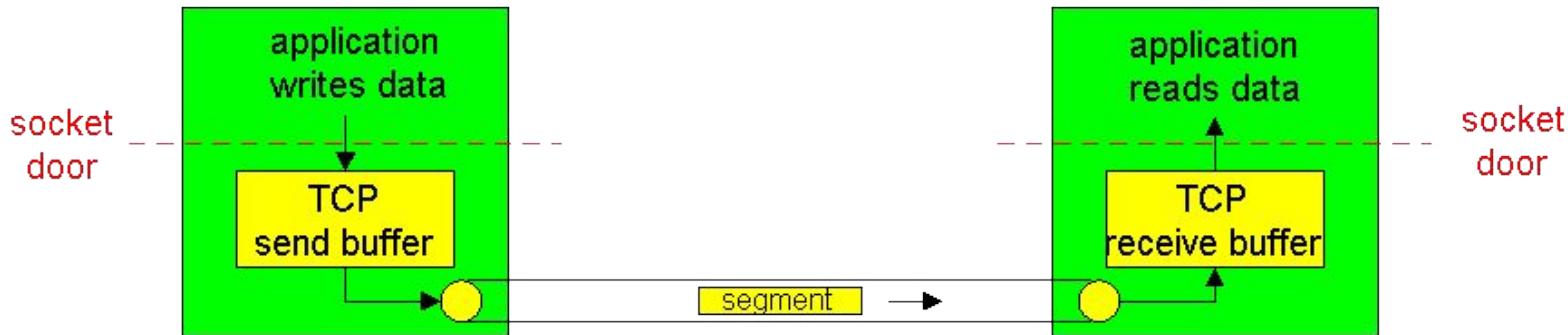
- transport-layer services
- multiplexing and demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- **connection-oriented transport: TCP**
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- principles of congestion control
- TCP congestion control

TCP: RFCs: 793, 1122, 2018, 5681, 7323

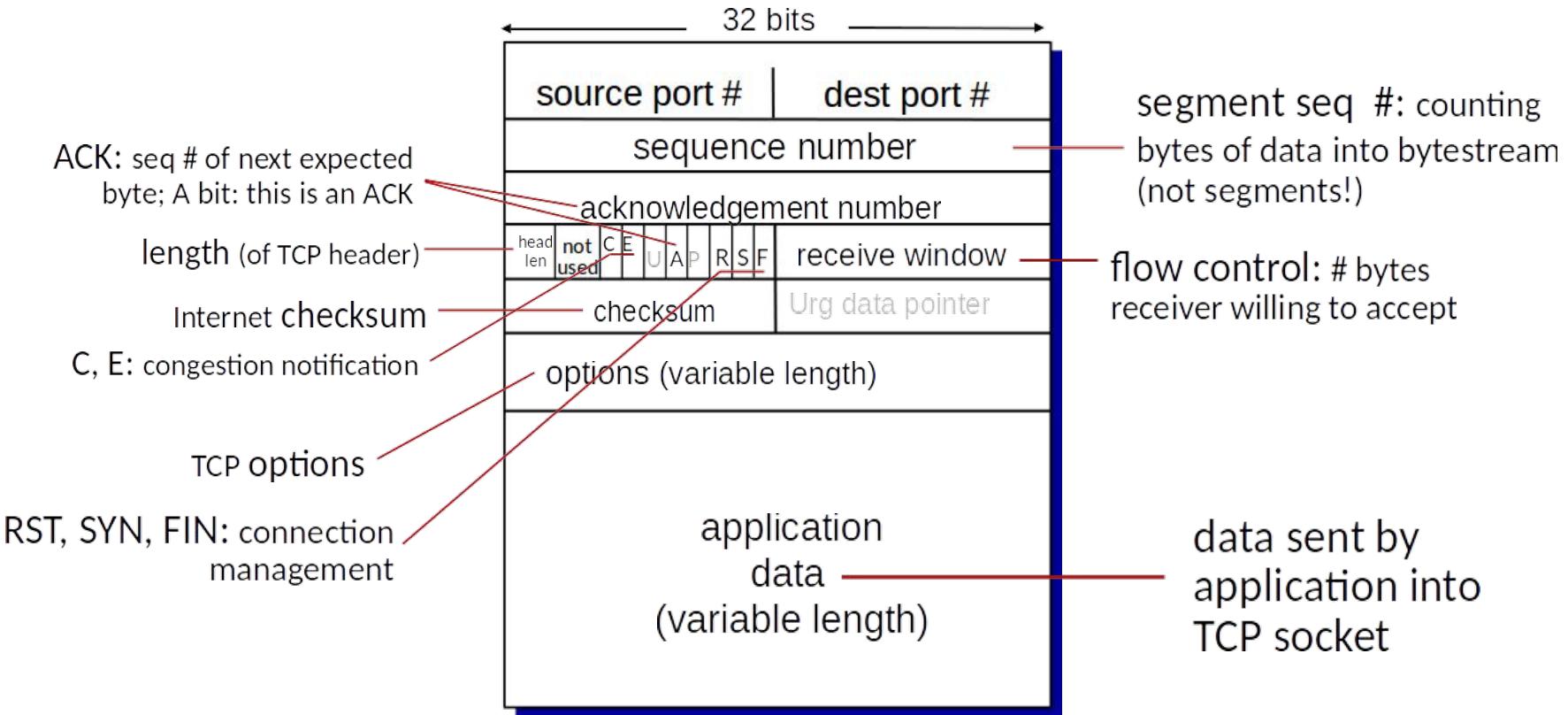
- **Point-to-point:**
 - one sender, one receiver
 - multicasting is not possible with TCP
- **reliable, in-order byte stream:**
 - no “message boundaries”
- **full duplex data:**
 - bi-directional data flow in same connection
- **Cumulative ACKs**
- **Pipelined**
 - TCP congestion and flow control set window size
- **connection-oriented:**
 - handshaking (exchange of control msgs) initializes sender, receiver state before data exchange
 - logical connection
- **flow controlled:** sender will not overwhelm receiver

TCP Connection

- Three-way handshake: `clientSocket.connect`, `serverSocket.accept`
- send and receive buffer are set during the initial three-way handshake
 - MSS: maximum segment size
 - MSS is the maximum amount of application layer data in the segment
 - MTU: the largest link-layer frame that can be sent
 - $\text{MSS} + \text{TCP headers} \leq \text{MTU}$
 - Note: MSS is not the maximum size of the TCP segment

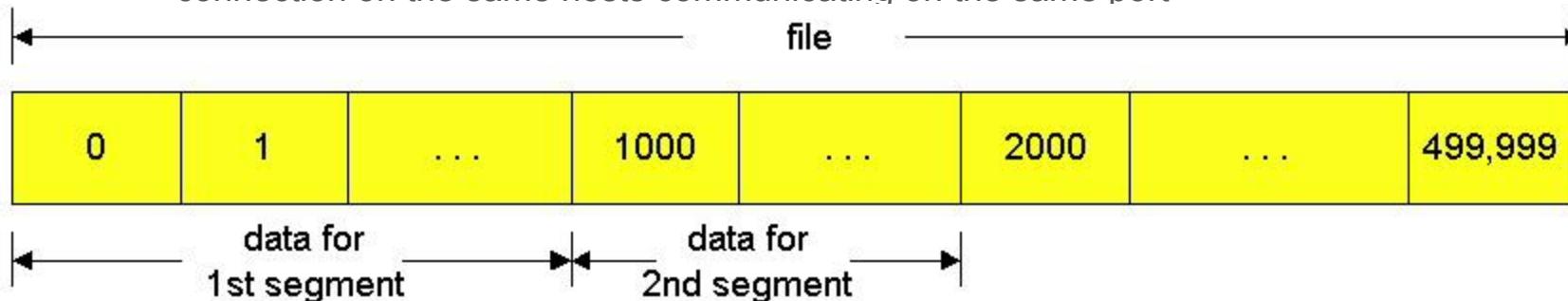


TCP Segment Structure



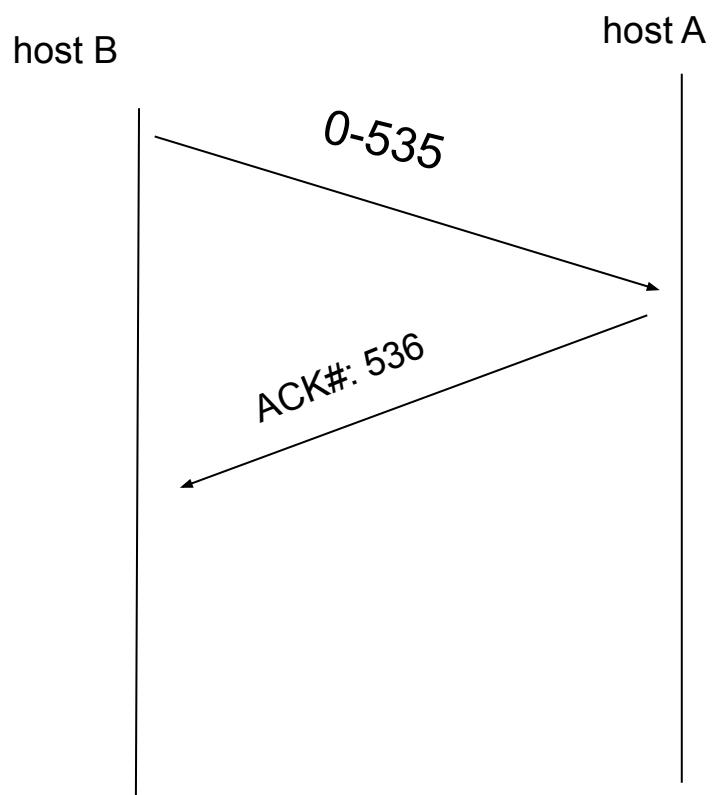
TCP sequence numbers

- sequence number for a segment is the byte number of the first byte in the segment
- Example:
 - data stream is a file consisting of 500,000 bytes; MSS=1000 bytes; 500 segments
 - The first segment get sequence number 0
 - The second segment get sequence number 1000
 - initial sequence number is randomly chosen to avoid numbers overlap between two connection on the same hosts communicating on the same port



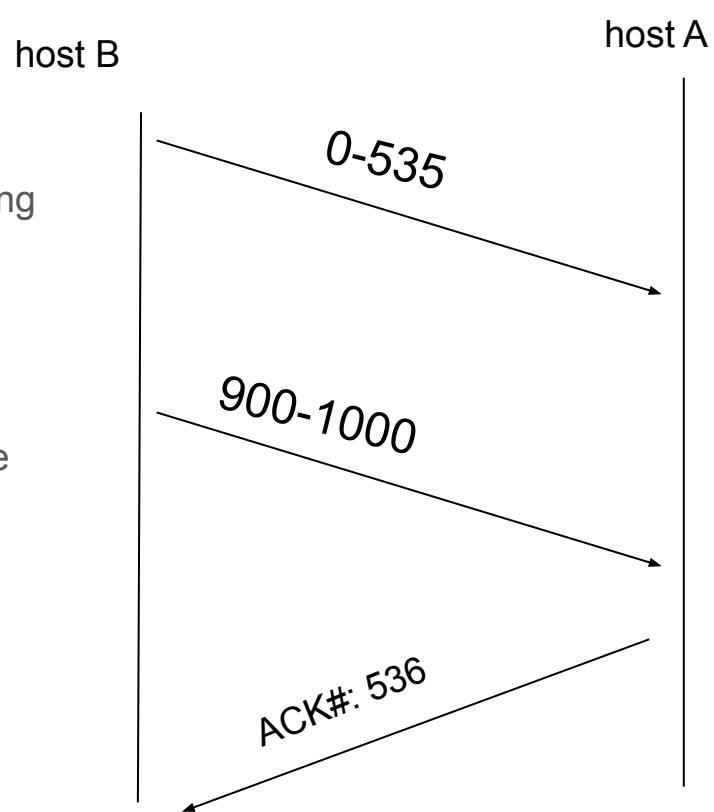
TCP Acknowledgement Numbers

- it is the sequence number of the next byte expecting
- Example
 - Host A has received all bytes numbered 0-535 from B
 - Host A is waiting for byte 536 and all the subsequent bytes
 - Host A puts 536 in the acknowledgement number



TCP Acknowledgement Numbers

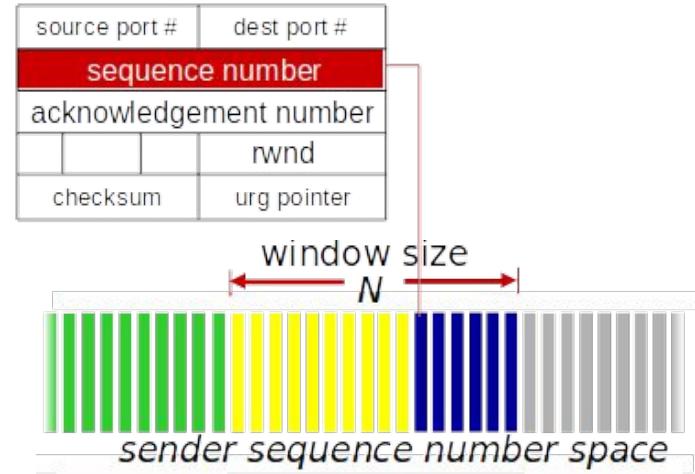
- Cumulative acknowledgement
- Example
 - Host A has received one segment from B containing bytes 0-535
 - Host A has received another segment from B containing bytes 900-1,000
 - Host A has not yet received bytes 536-899
 - Hosts A's next segment to B will contain 536 in the acknowledgement number field
 - cumulative acknowledgement: TCP only acknowledges bytes up to the first missing byte in the stream



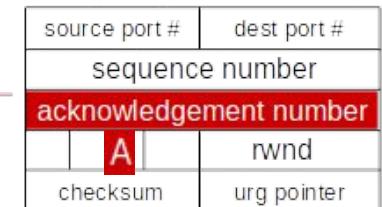
TCP seq. numbers, ACKs

- **sequence numbers:**
 - byte stream “number” of first byte in segment’s data
- **Acknowledgements:**
 - seq # of next byte expected from other side
 - cumulative ACK
- Q: how receiver handles out-of-order segments
- A: TCP spec doesn’t say, - up to implementor
 - They are buffered in practice

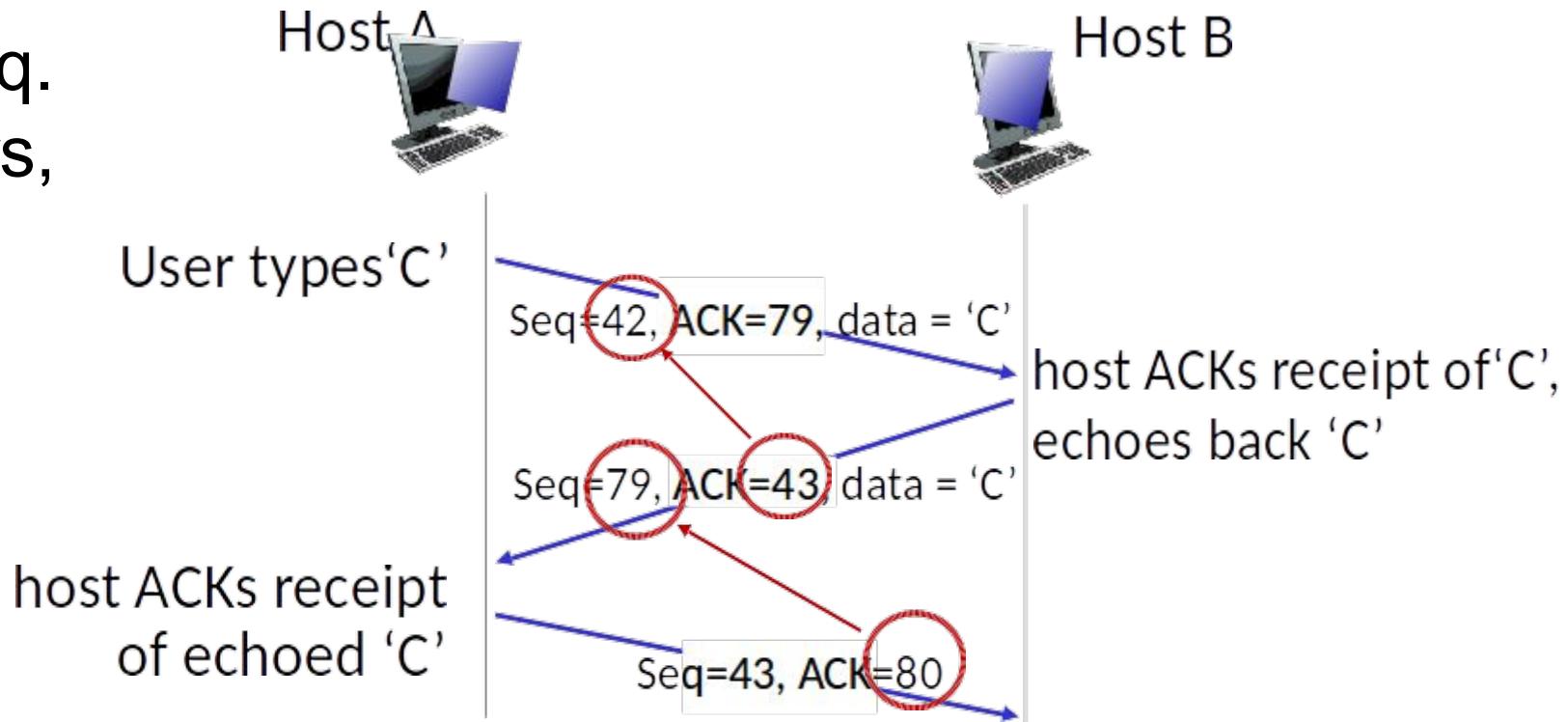
outgoing segment from sender



outgoing segment from receiver



TCP seq. numbers, ACKs



simple telnet scenario

Question

Suppose host A is sending a large file to host B over a TCP connection. If the sequence number for a segment of this connection is m , then the sequence number for the subsequent segment will necessarily be $m+1$

- True
- False

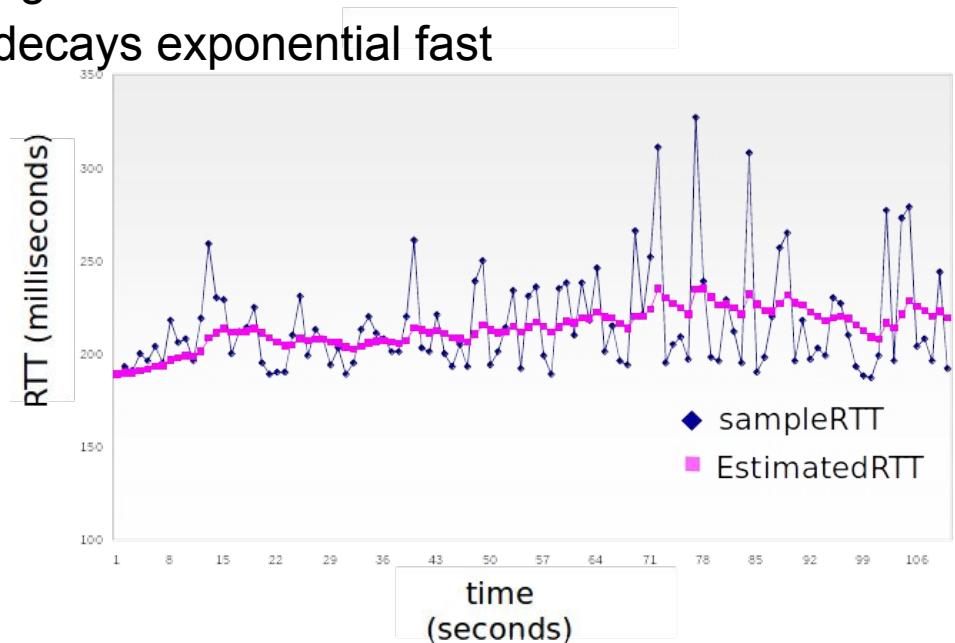
TCP round trip time, timeout

- Q: how to set TCP timeout value?
 - longer than RTT
 - too short: premature timeout, unnecessary retransmissions
 - too long: slow reaction to segment loss
 - RTT varies
- Q: how to estimate RTT?
 - **SampleRTT**: measured time from segment transmission until ACK receipt
 - Once in a while measured for only one of the transmitted but unacknowledged segment
 - ignore retransmissions. Why?
 - SampleRTT will vary, want estimated RTT “smoother”
 - **EstimatedRTT**: average several *recent* measurements, not just current SampleRTT

TCP round trip time, timeout

$$\text{EstimatedRTT} = (1-\alpha) \text{ EstimatedRTT} + \alpha \text{ SampleRTT}$$

- exponential weighted moving average
- The weight of a given sampleRTT decays exponential fast as the updates proceed
- typical value:
- Typical value: $\alpha = 0.125$



TCP round trip time, timeout

- estimate of the variability of RTT
- DevRTT: an estimate of how much SampleRTT deviates from EstimatedRTT:
 - EWMA of SampleRTT deviation from EstimatedRTT:

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

- If sampleRTT values have little fluctuation, then DevRTT will be small
- If there is a lot of fluctuation, DevRTT will be large

TCP round trip time, timeout

- timeout interval: **EstimatedRTT** plus “safety margin”
 - large variation in **EstimatedRTT** -> larger safety margin

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

↑ ↑
estimated RTT “safety margin”

- Initial **TimeoutInterval**=1 sec (RFC 6298)
 - When a timeout occurs, the value of TimeoutInterval is doubled to avoid a premature timeout occurring for a subsequent segment that will soon be acknowledged
 - As soon as a segment is received and EstimatedRTT is updated, the TimeoutInterval is again computed using the formula

Question

Suppose that the last `sampleRTT` in a TCP connection is equal to 1 sec. The current value of `TimeoutInterval` for the connection will necessarily be ≥ 1 sec.

- True
- False

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

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
- IP service
 - does no guarantee delivery
 - no guarantee on in-order delivery
 - does not guarantee the integrity of the data in the datagram
 - datagrams can overflow router buffers and never reach their destination

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
 - pipelined segments
 - cumulative acks
 - **single** retransmission timer
- retransmissions triggered by:
 - timeout events
 - duplicate acks

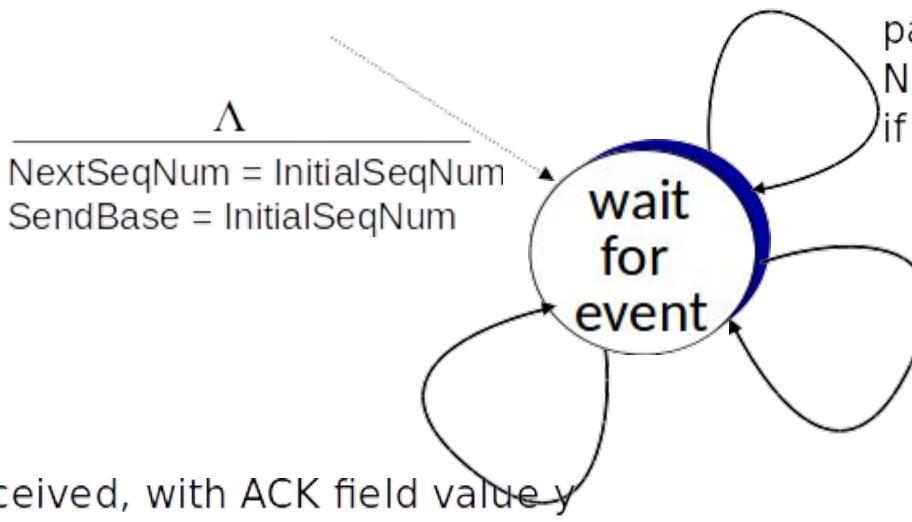
let's initially consider simplified TCP sender:

- ignore duplicate acks
- ignore flow control, congestion control

TCP sender events:

- **data rcvd from application layer:**
 - create segment with seq # NextSeqNum
 - seq # is byte-stream number of first data byte in segment
 - start timer if not already running
 - think of timer as for oldest unacked segment; expiration interval: **TimeOutInterval**
- **Timeout:**
 - retransmit the segment that caused the timeout:
 - not-yet-acknowledged segment with smallest sequence number
 - restart timer
- **ack rcvd:**
 - if ack acknowledges previously unacked segments
 - update what is known to be ACKed
 - start timer if there are still unacked segments

TCP Sender (simplified)



```
if ( $y > \text{SendBase}$ ) {  
     $\text{SendBase} = y$   
    /*  $\text{SendBase}-1$ : last cumulatively ACKed byte */  
    if (there are currently not-yet-acked segments)  
        start timer  
    else stop timer  
}
```

data received from application above
create segment, seq. #: NextSeqNum
pass segment to IP (i.e., "send")
 $\text{NextSeqNum} = \text{NextSeqNum} + \text{length(data)}$
if (timer currently not running)
 start timer

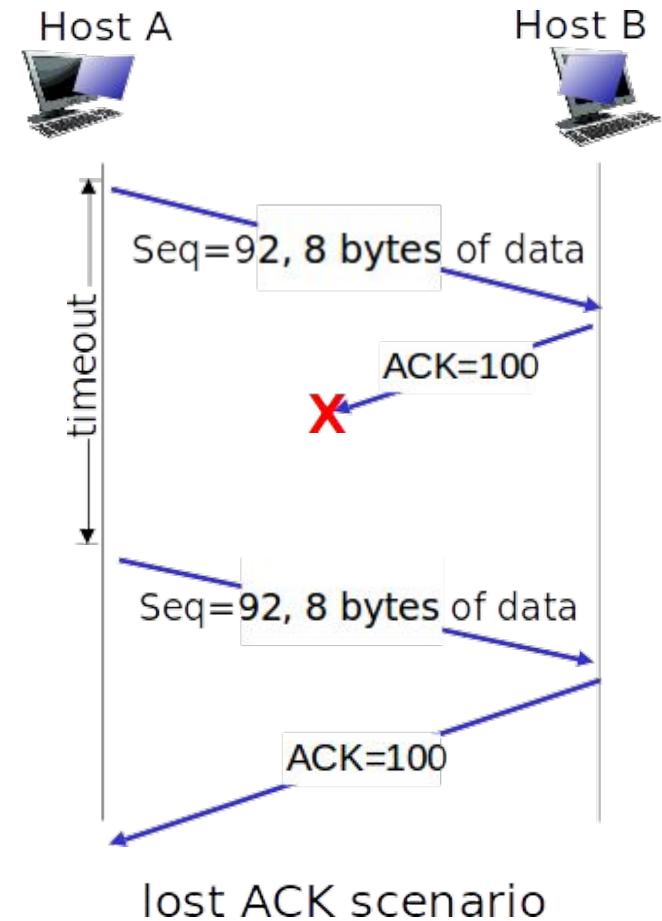
timeout
retransmit not-yet-acked
segment with smallest seq. #
start timer

TCP ACK generation: RFC 1122, RFC 2581

<i>event at receiver</i>	<i>TCP receiver action</i>
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. #. Gap detected	immediately send <i>duplicate ACK</i> , indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

TCP: Retransmission Scenarios

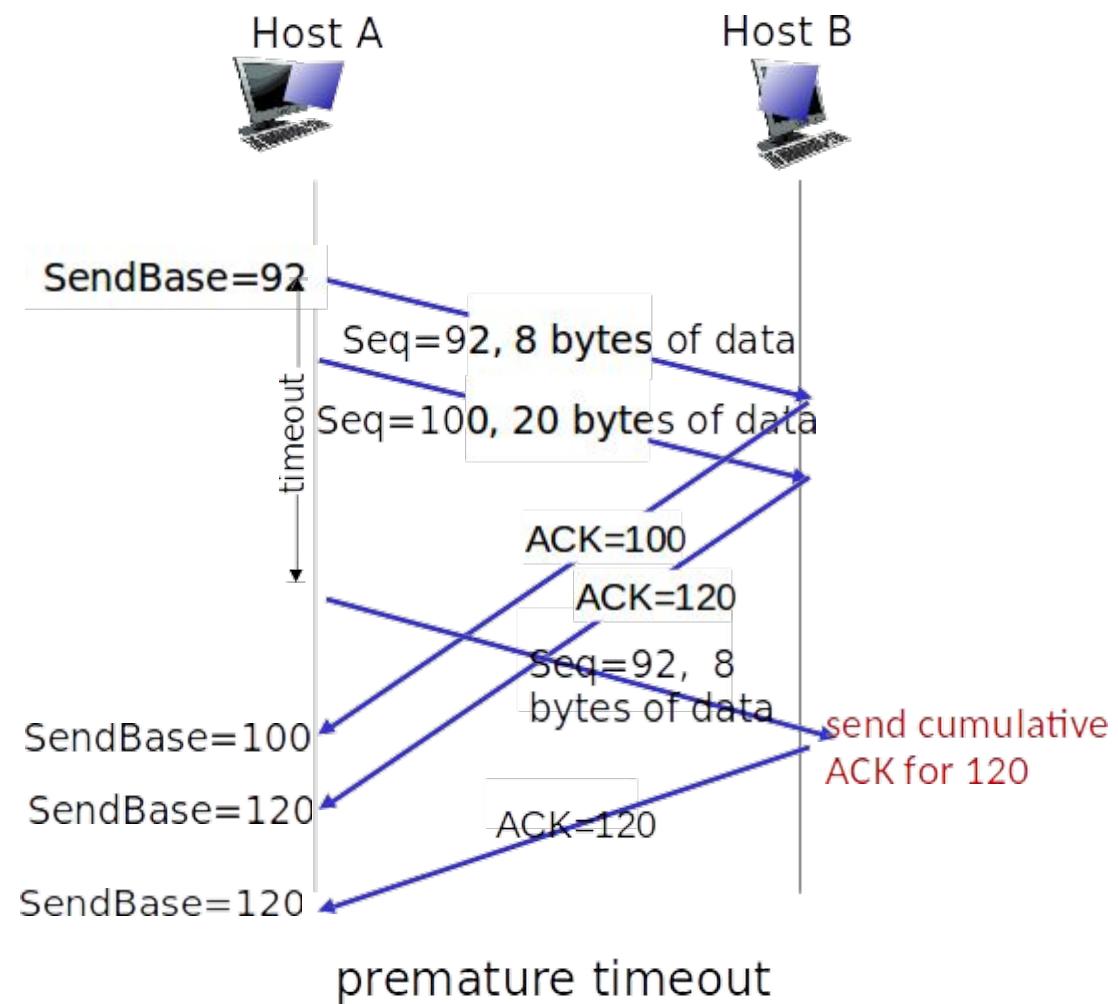
Retransmission due to a lost acknowledgement



TCP: Retransmission Scenarios

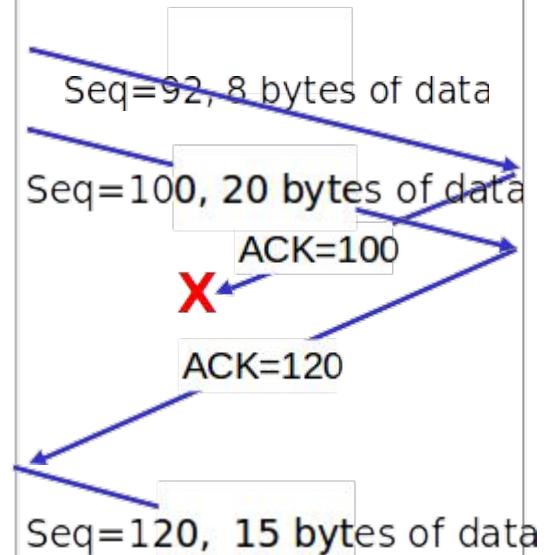
Premature timeout

segment 100 not
retransmitted



TCP: Retransmission Scenarios

A cumulative acknowledgement
avoids retransmission of the first
segment



cumulative ACK
covers for earlier
lost ACK

TCP: Doubling the Timeout Interval

- In case of a timeout
 - TCP retransmits the not-yet-acknowledged segment with the smallest sequence number
 - Each time TCP retransmits. It sets the next timeout interval to twice the previous value
 - Example: TimeoutInterval for the oldest not yet acknowledge segment: 0.75 sec
 - When the timer first expires, TCP sets the new expiration time to 1.5 sec
 - If the time expires again: TCP will retransmit this segment and set the expiration time to 3.0 sec
 - When acknowledgement is received, the TimeoutInterval is derived from the most recent values of EstimatedRTT and DevRTT
 - A limited form of congestion control

TCP fast retransmit

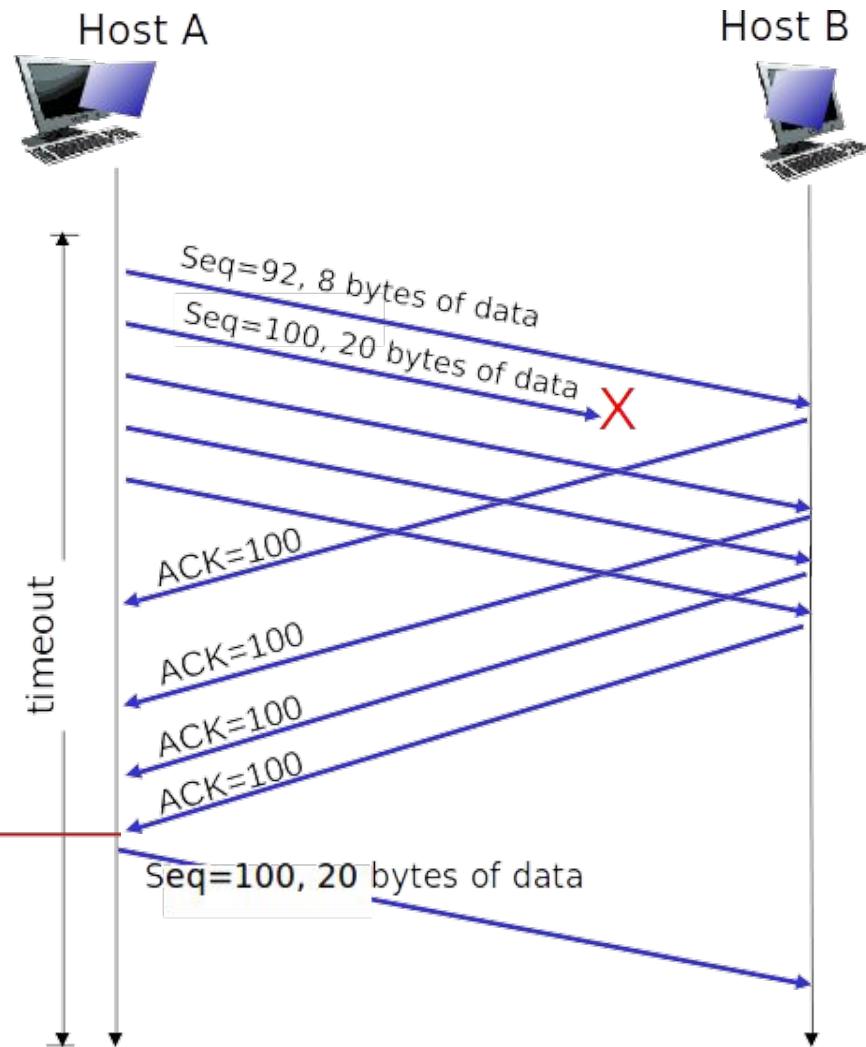
- time-out period often relatively long:
 - long delay before resending lost packet
- detect lost segments via **duplicate ACKs.**
 - sender often sends many segments back-to-back
 - if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

- if sender receives 3 ACKs for same data (“triple duplicate ACKs”), resend unacked segment with smallest seq #
 - likely that unacked segment lost, so don’t wait for timeout



Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



TCP fast retransmit

- Why TCP waits for three ACKs?
 - a tradeoff between triggering a quick retransmission when needed, but not retransmitting prematurely in the face of packet reordering

TCP: Go-Back-N or Selective Repeat?

- similarities with Go-Back-N
 - cumulative acknowledgement
 - acknowledging the last correctly received in-order segment
 - correctly received but out-of-order segments are not individually acked by the receiver
 - windows size(N), **base**: the sequence number of the oldest unacknowledged packet, and **nextseqnum**: smallest unused sequence number
- differences with Go-Back-N
 - correctly received but out-of-order packets are buffered in many TCP implementations
 - TCP transmits only the lost packet, not all the packets in the window
 - selective acknowledgement (proposed in 2018)
 - acknowledges out-of-order segments selectively rather than just cumulatively

outline

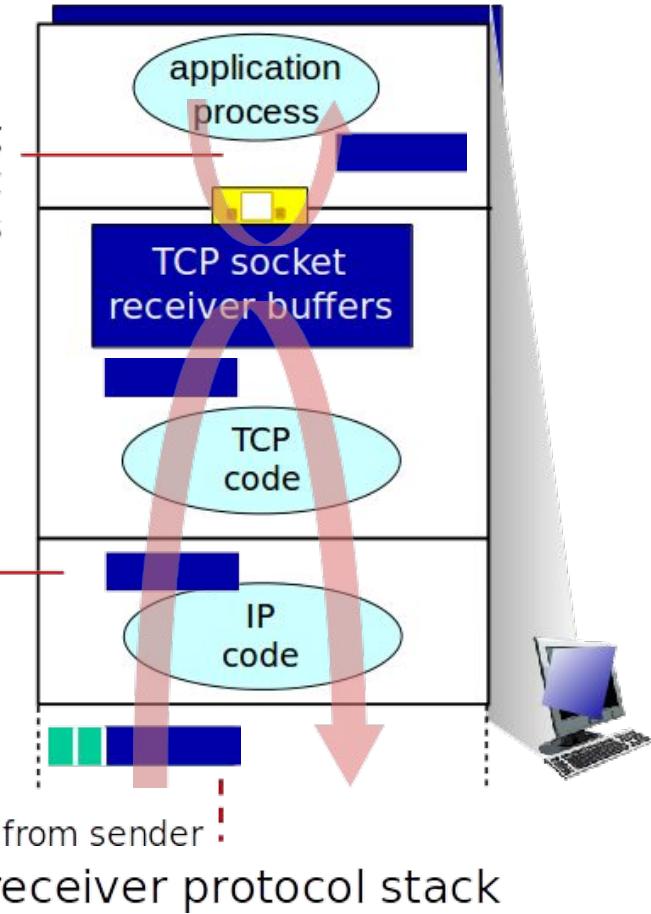
- transport-layer services
- multiplexing and demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - **flow control**
 - connection management
- principles of congestion control
- TCP congestion control

TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



Application removing data from TCP socket buffers



TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

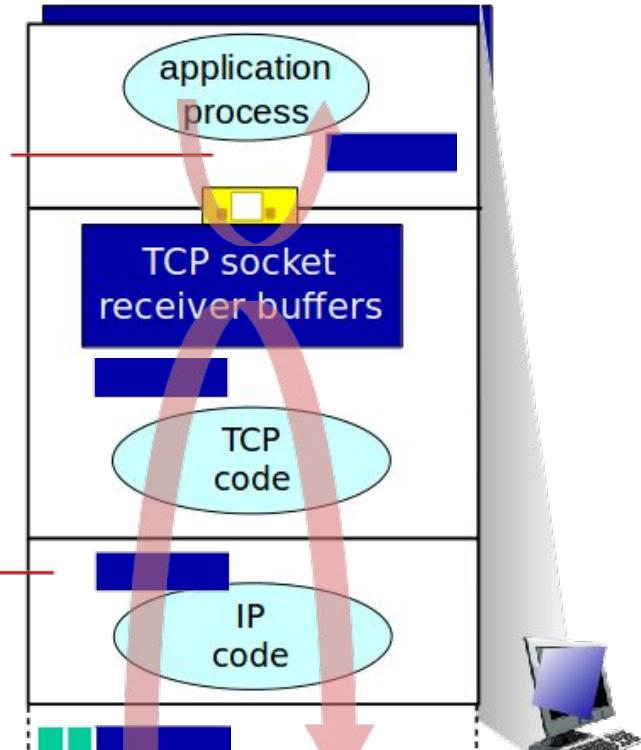
Flow control

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

Application removing data from TCP socket buffers

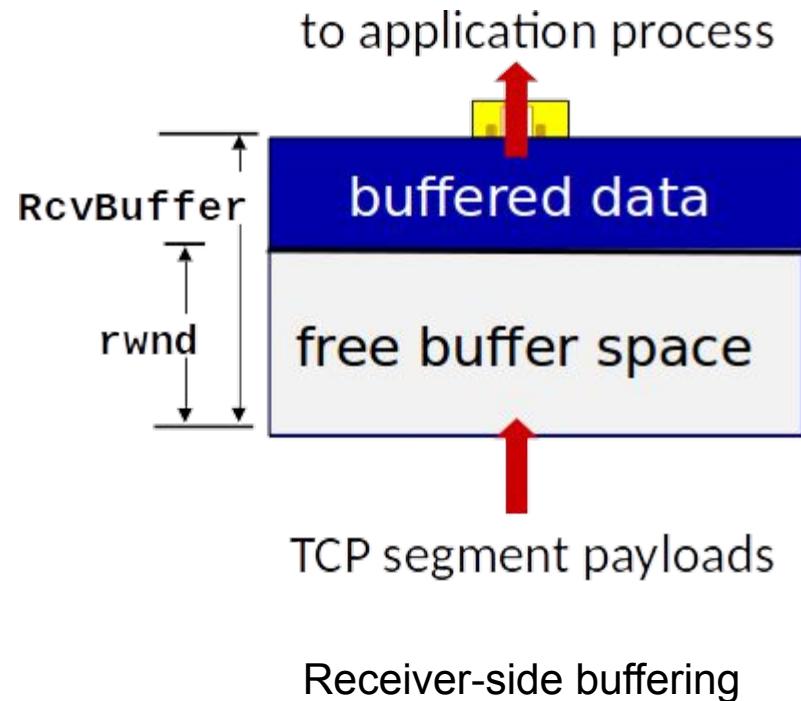
Network layer delivering IP datagram payload into TCP socket buffers

from sender
receiver protocol stack



TCP flow control

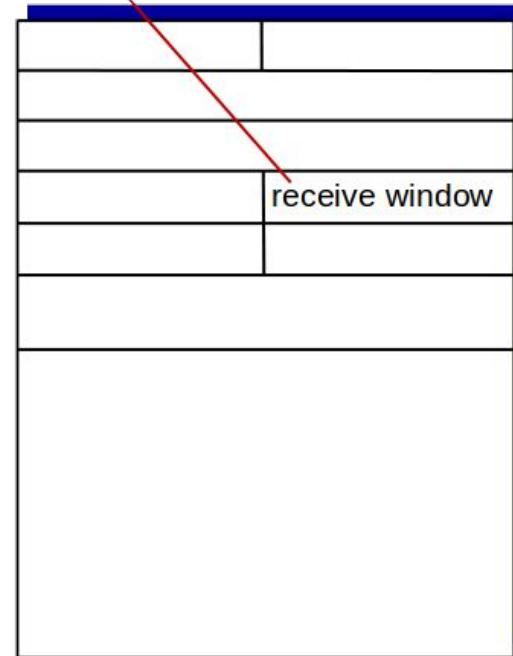
- receiver “advertises” free buffer space by including **rwnd** value in TCP header
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - many operating systems autoadjust **RcvBuffer**
- sender limits amount of unacked (“in-flight”) data to receiver’s **rwnd** value
- guarantees receive buffer will not overflow



TCP flow control

- receiver “advertises” free buffer space by including **rwnd** value 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 receiver’s **rwnd** value
- guarantees receive buffer will not overflow

flow control: # bytes receiver willing to accept



TCP segment format

TCP flow control: animation

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/flow-control/index.html

Question

Suppose Host A is sending Host B a large file over a TCP connection. The number of unacknowledged bytes that A sends cannot exceed the size of the receive buffer

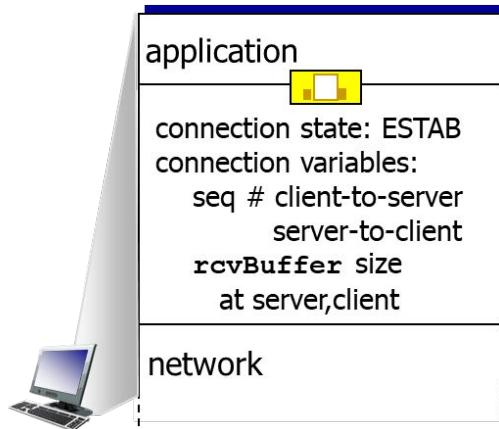
- True
- False

outline

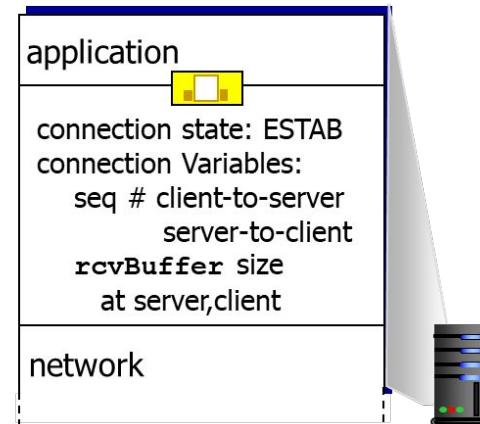
- transport-layer services
- multiplexing and demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - **connection management**
- principles of congestion control
- TCP congestion control

Connection Management

- before exchanging data, sender/receiver “handshake”:
 - agree to establish connection (each knowing the other willing to establish connection)
 - agree on connection parameters

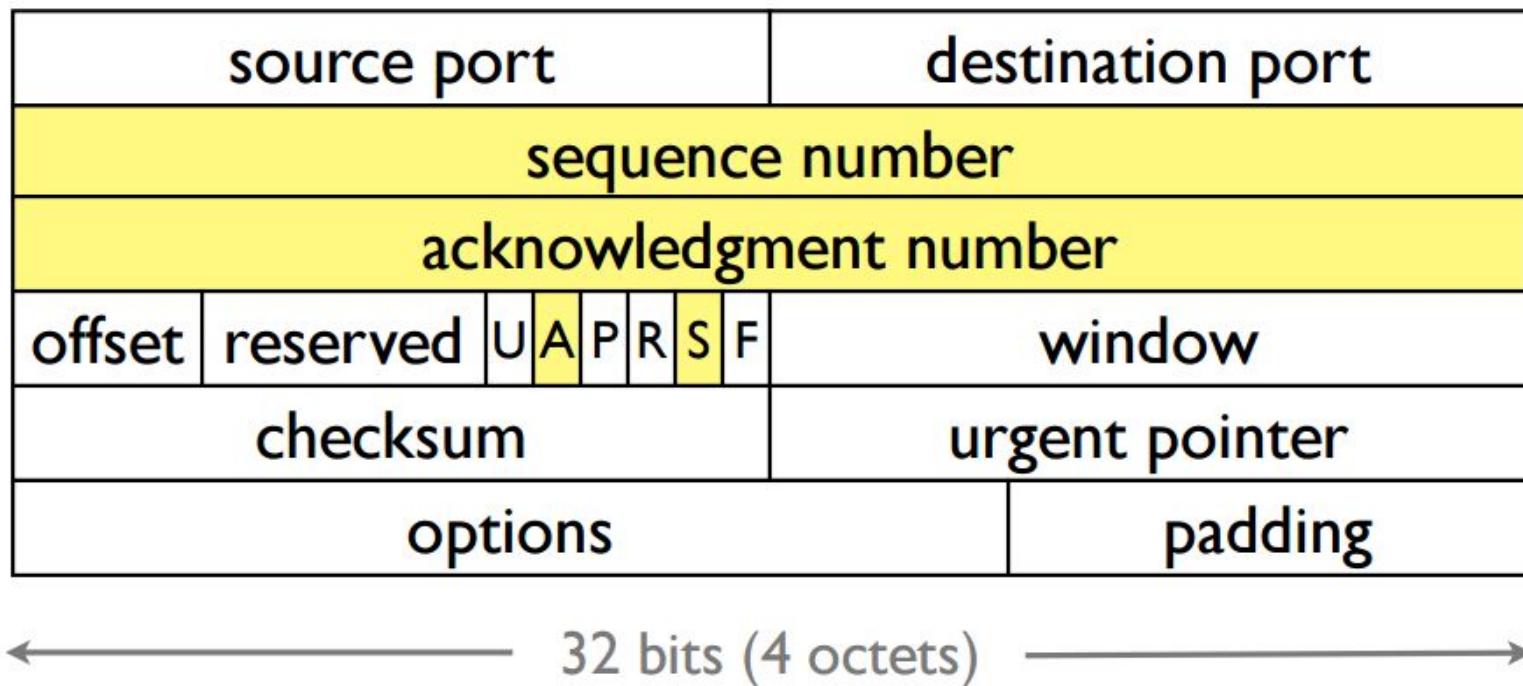


```
Socket clientSocket =  
    newSocket("hostname", "port  
number");
```



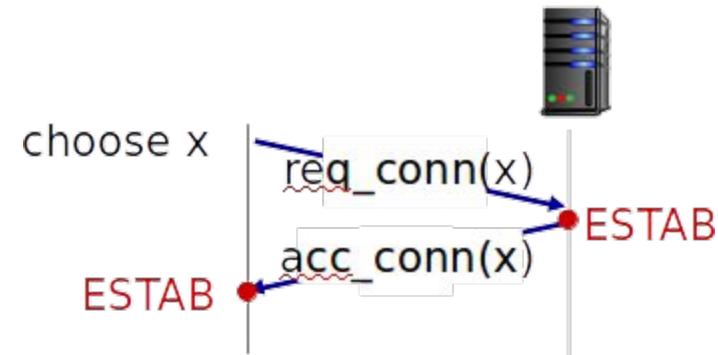
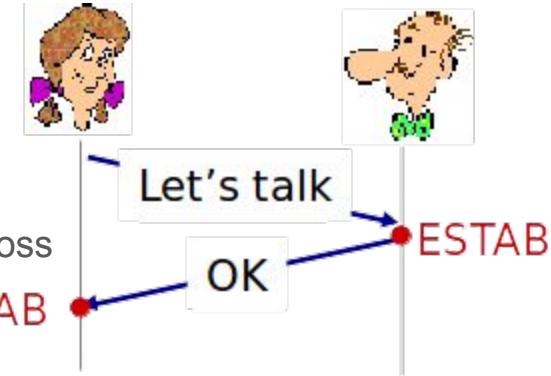
```
Socket connectionSocket =  
    welcomeSocket.accept();
```

Connection Management

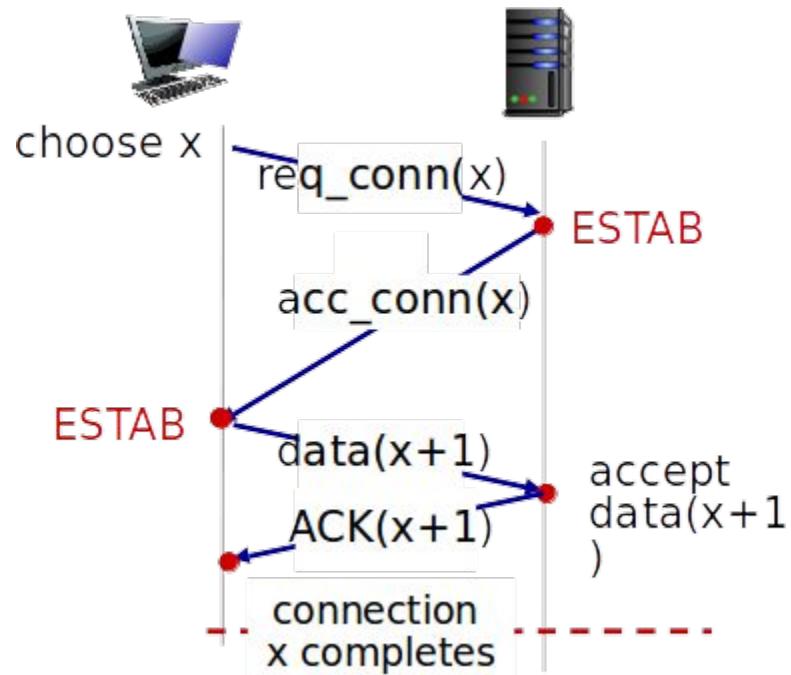


Agreeing to establish a connection

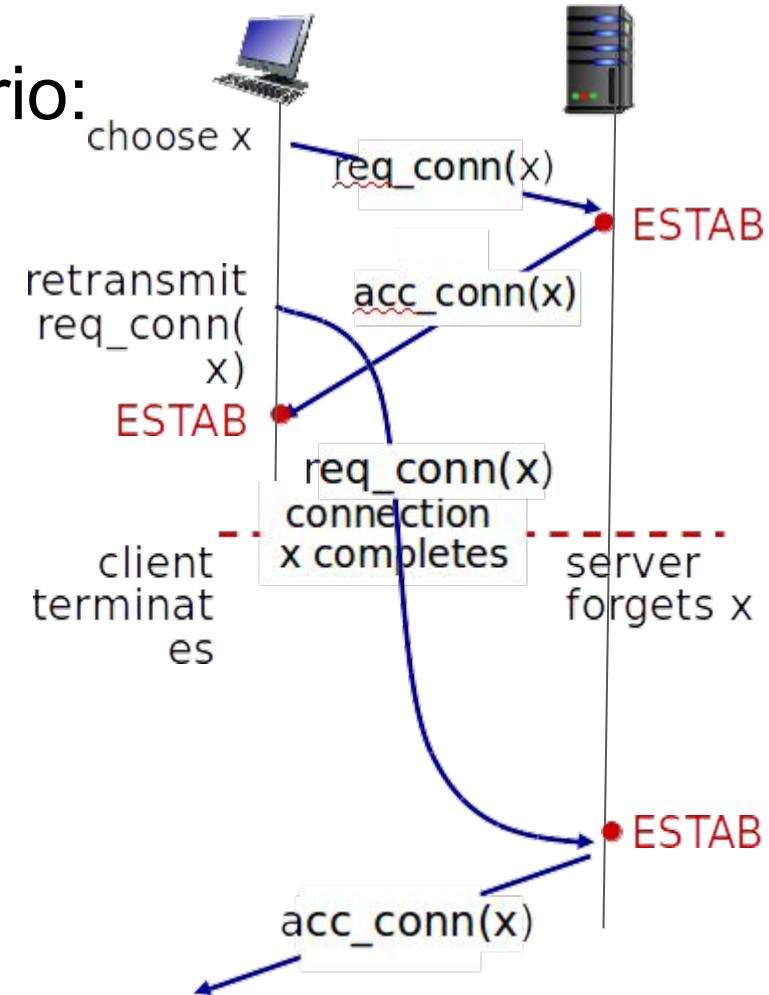
- 2-way handshake:
- Q: will 2-way handshake always work in network?
 - variable delays
 - retransmitted messages (e.g. `req_conn(x)`) due to message loss
 - message reordering
 - can't "see" other side



2-way handshake scenarios: One

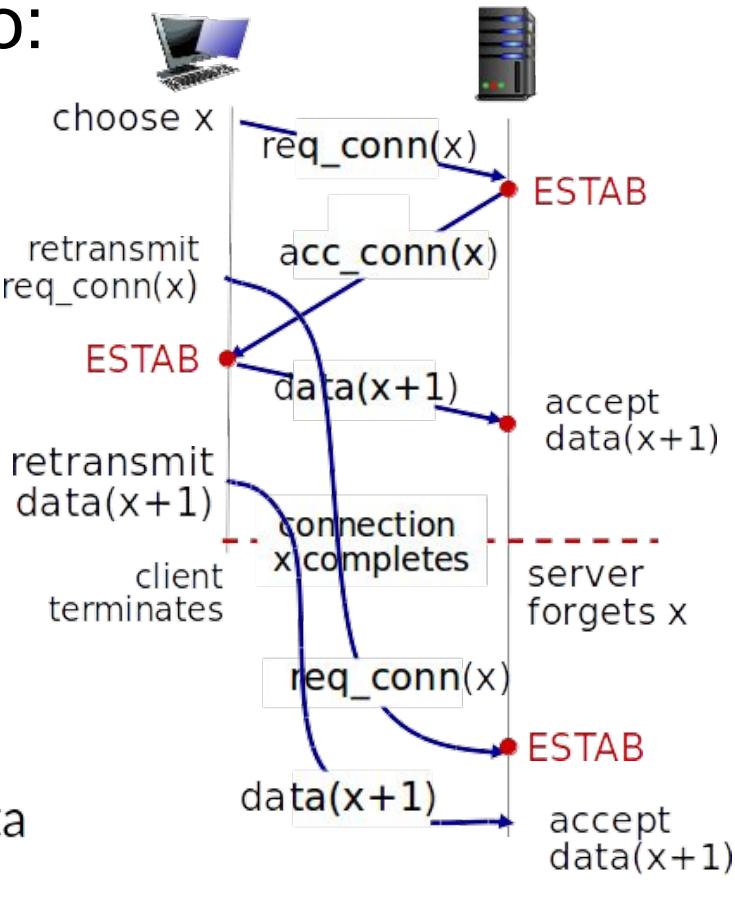


2-way handshake failure scenario: Second scenario



Problem: half open
connection! (no client)

2-way handshake failure scenario: Third scenario



Problem: dup data
accepted!

TCP 3-way handshake

Client state

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

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

SYNSENT

choose init seq num, x
send TCP SYN msg



SYNbit=1, Seq=x

ESTAB

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

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

ACKbit=1, ACKnum=y+1

Server state

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

LISTEN

SYN RCVD

ESTAB

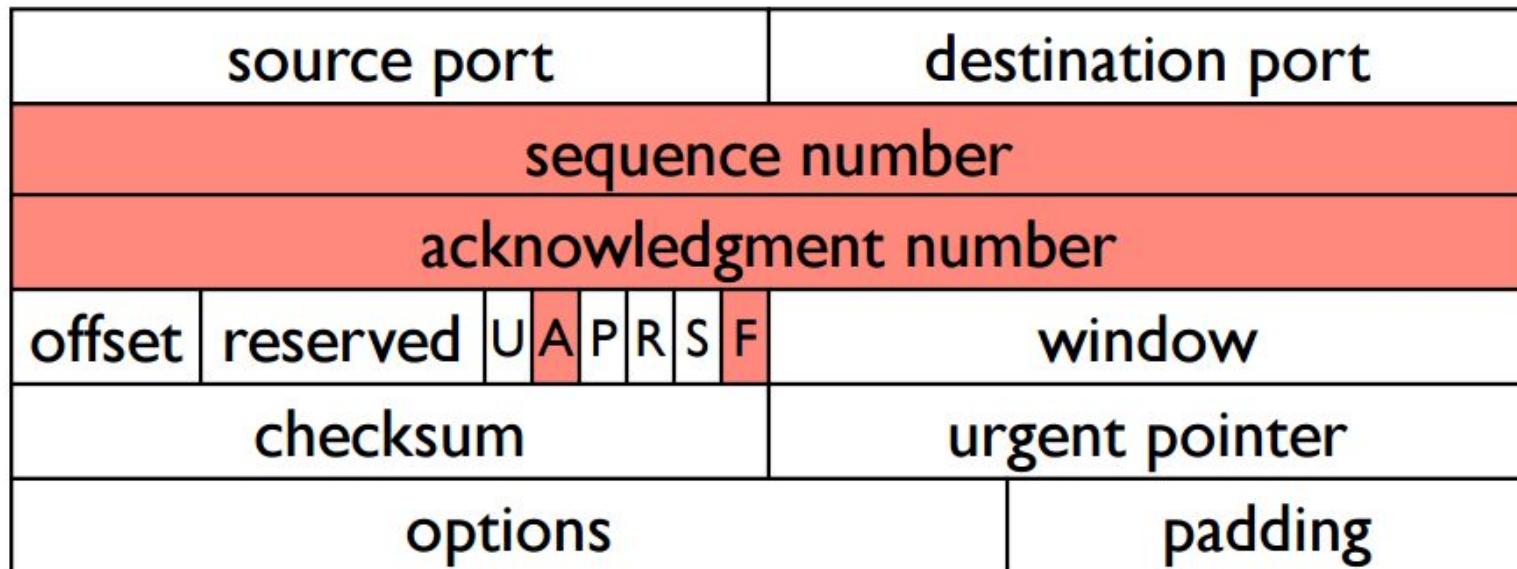
choose init seq num, y
send TCP SYNACK
msg, acking SYN

received ACK(y)
indicates client is live

TCP: closing a connection

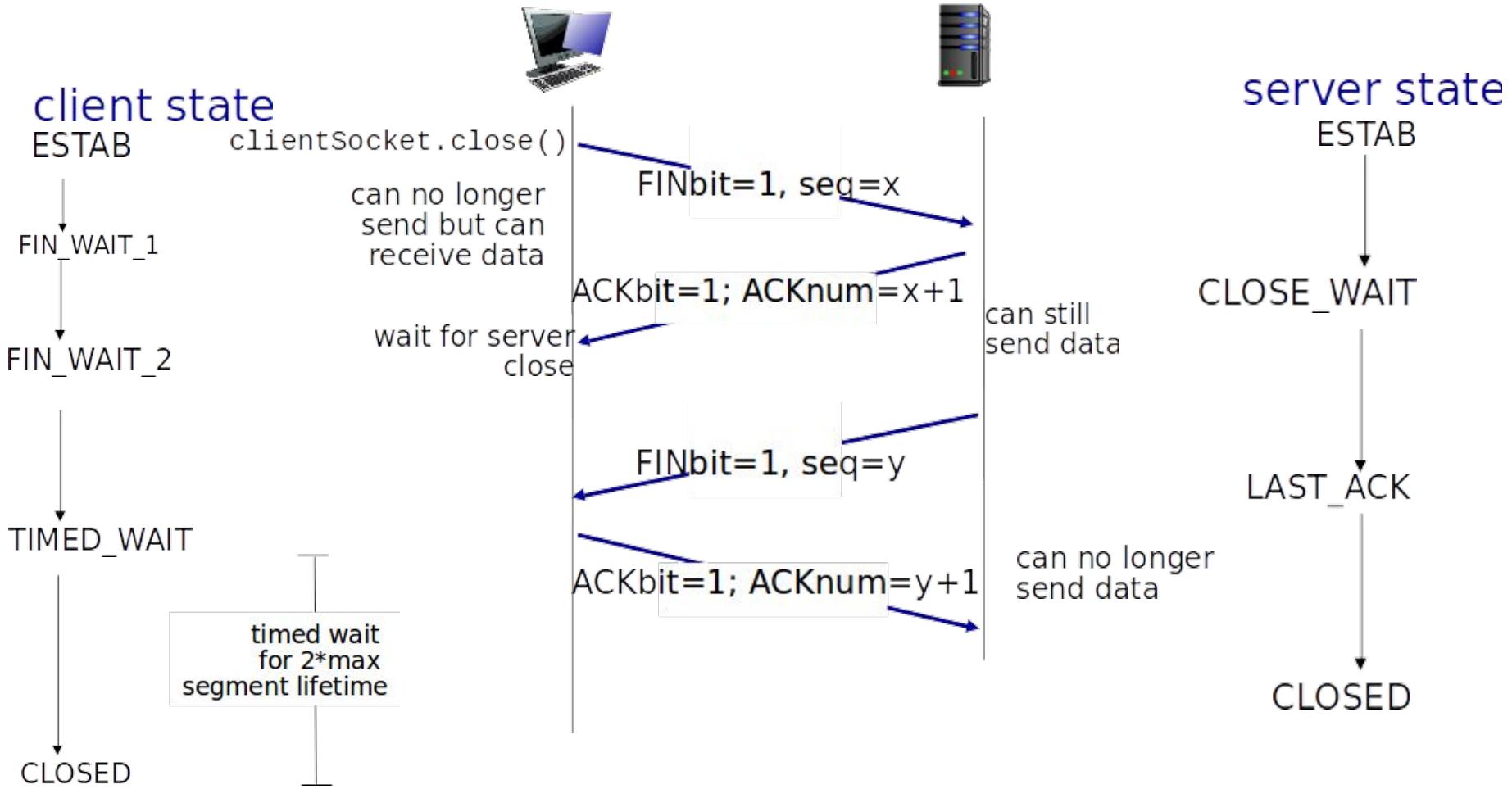
- client, server each close their side of connection
 - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
 - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

TCP: closing a connection



32 bits (4 octets)

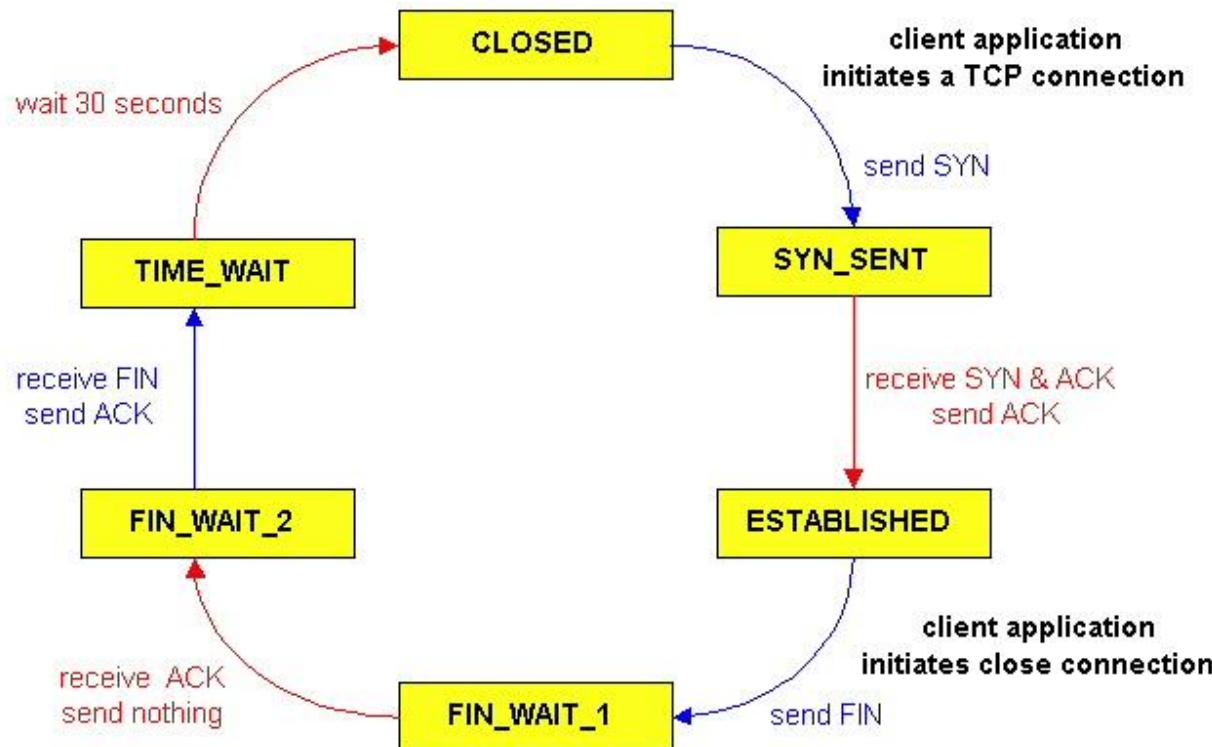
TCP: closing a connection



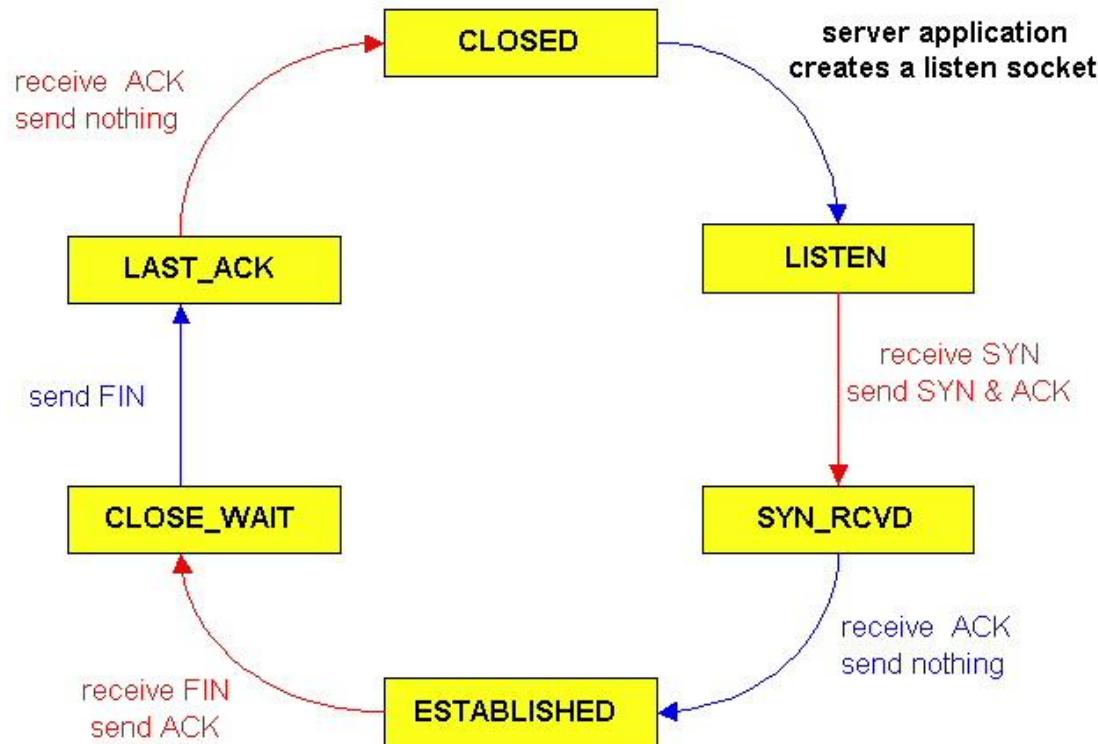
TCP: Closing a Connection

- Time_wait state:
 - Wait a long time after sending all the segments and before completing the close.
 - $2 * \text{maximum segment lifetime}$
- Why?
 - Delayed segments from one connection being misinterpreted as being part of a subsequent connection.
 - ACK might have been lost, in which case FIN will be resent for an orderly close

States visited by a client TCP



States visited by a server TCP



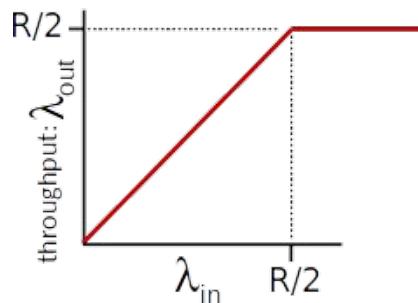
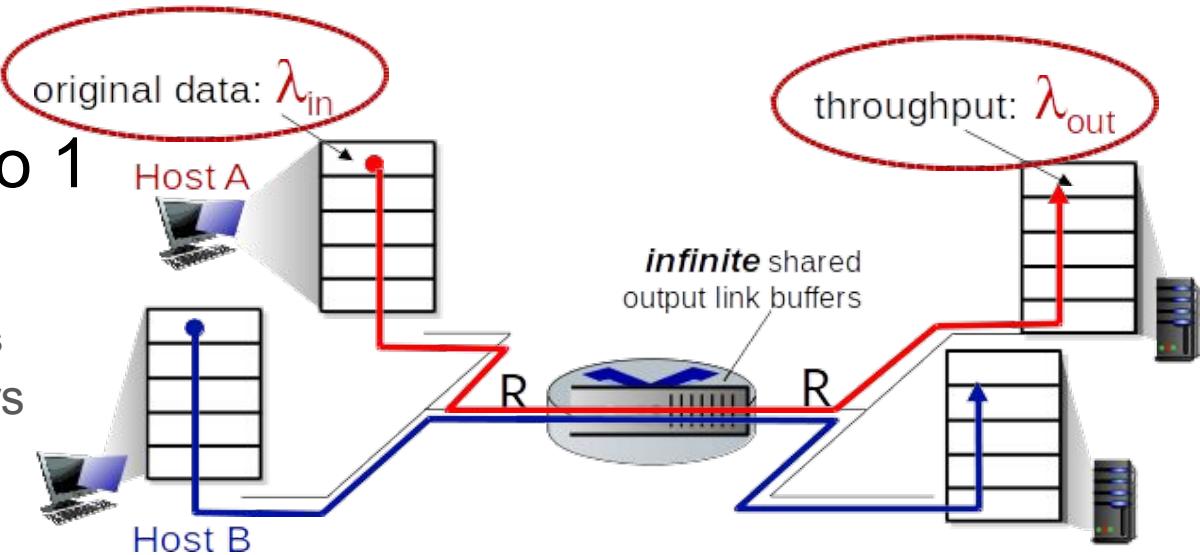
principles of congestion control

- Congestion:
 - informally: “too many sources sending too much data too fast for network to handle”
 - Causes: buffers get full
 - Manifestations/Costs:
 - lost packets (buffer overflow at routers)
 - long delays (queueing in router buffers)
 - a top-10 problem!
 - Different from flow control

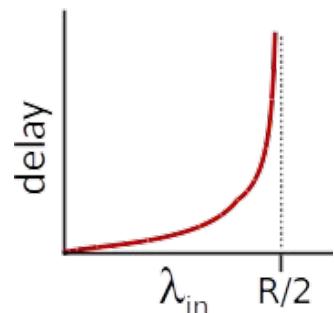


Causes/Costs of Congestion: Scenario 1

- Two flows: red and blue
 - two senders, two receivers
 - one router, **infinite** buffers
 - No retransmission
 - output link capacity: R
-
- What happens as arrival rate λ_{in} approaches R/2?



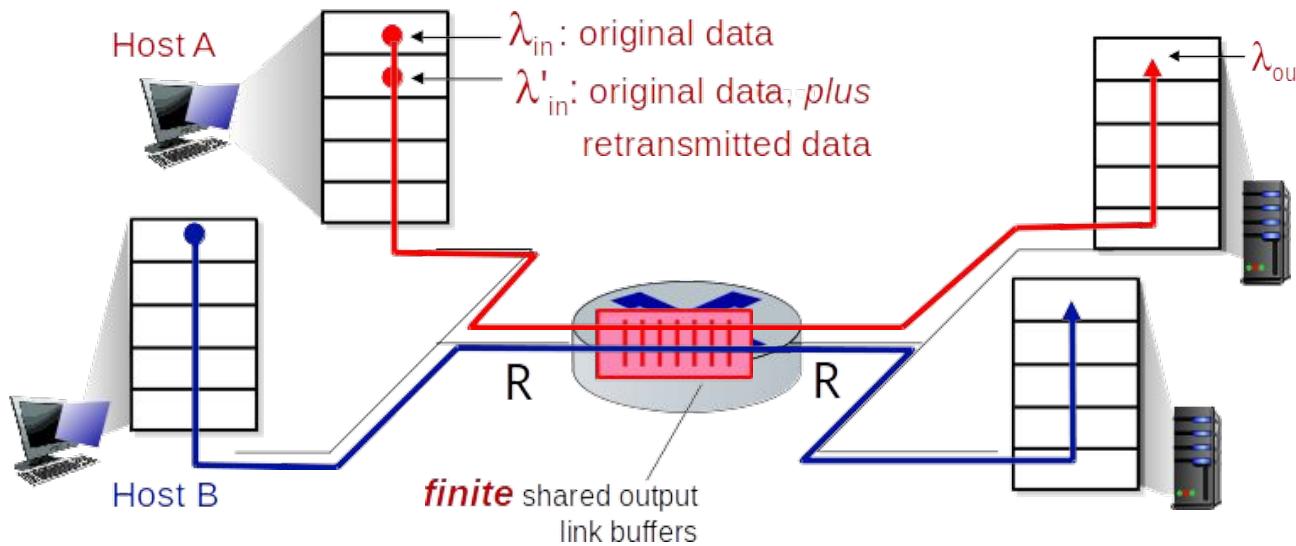
maximum per-connection throughput: $R/2$



large delays as arrival rate λ_{in} approaches capacity

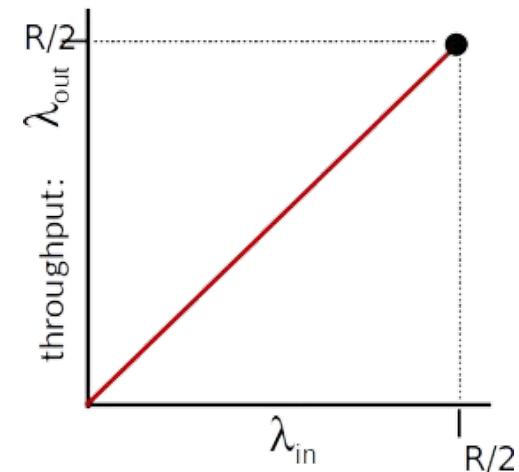
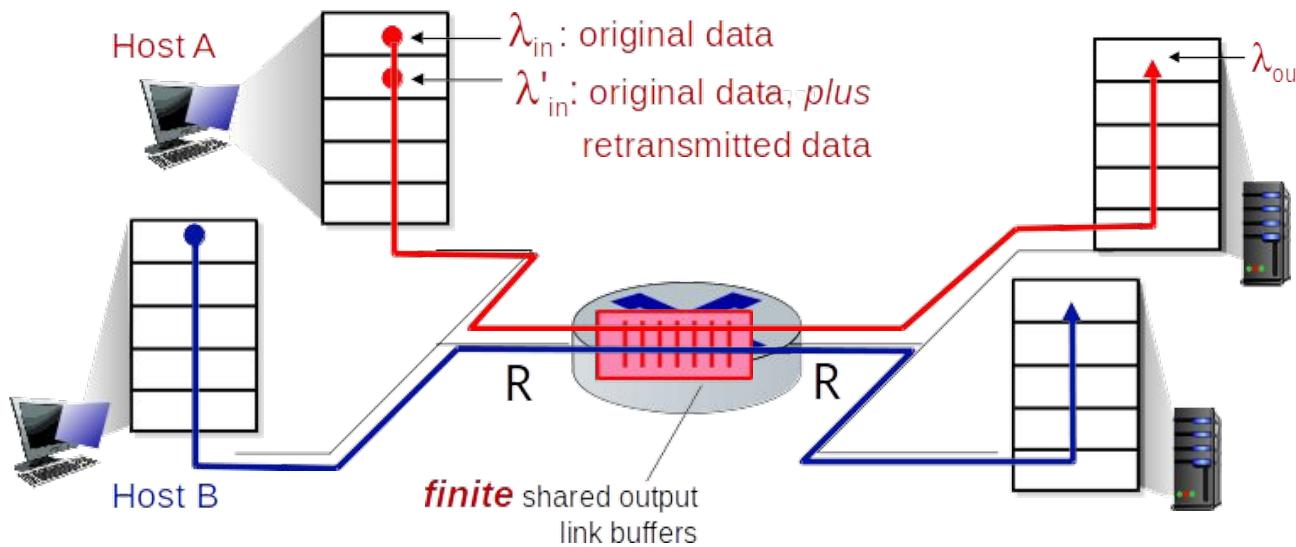
Causes/Costs of Congestion: Scenario 2

- one router, **finite** buffers
- sender retransmits lost, timed-out packet
 - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
 - transport-layer input includes retransmissions : $\lambda'_{in} \geq \lambda_{in}$



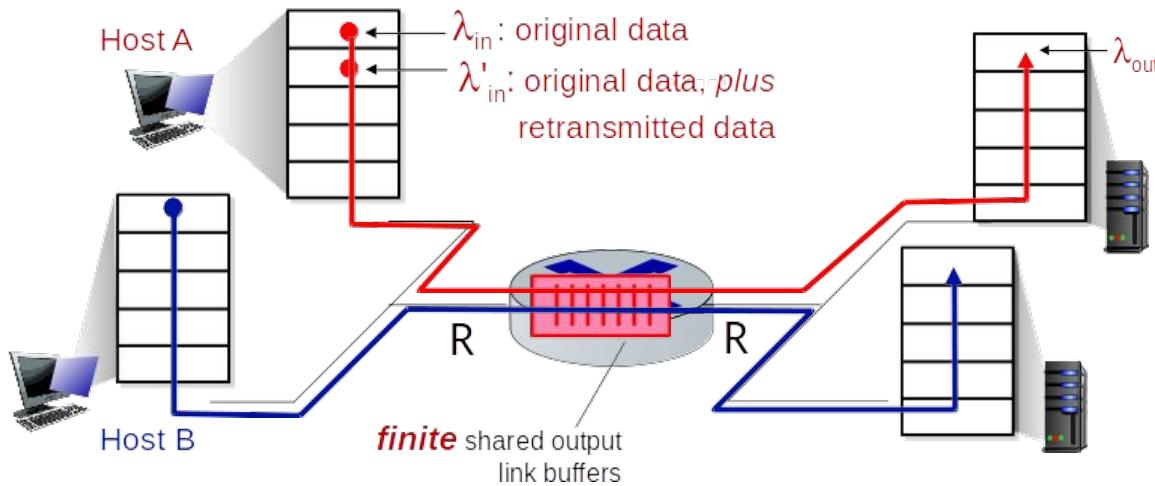
Causes/Costs of Congestion: Scenario 2: case 1

- **Idealization:** perfect knowledge
 - sender sends only when router buffers available



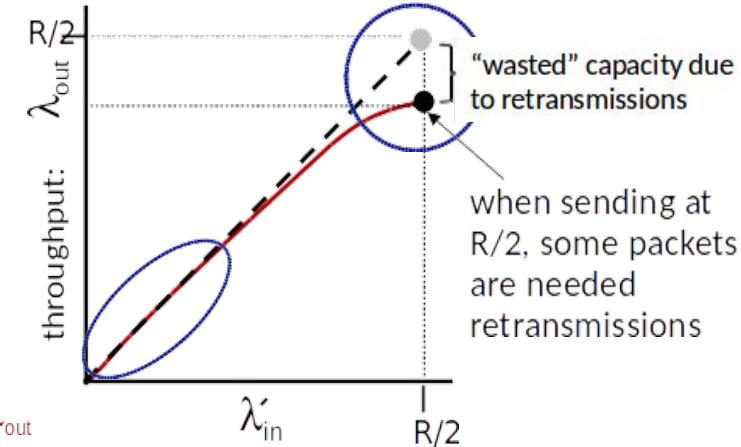
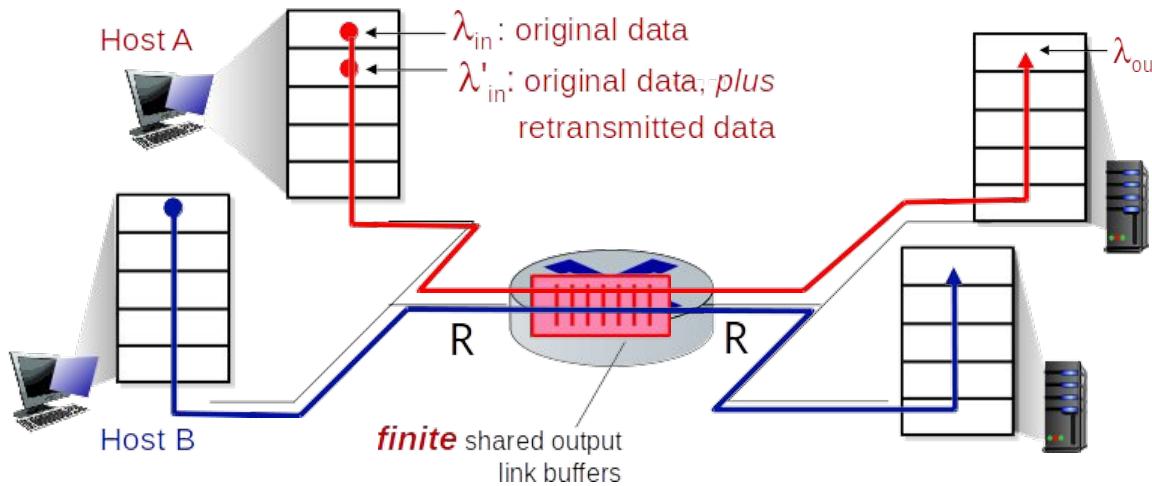
Causes/Costs of Congestion: Scenario 2: case 2

- **Idealization: Some** perfect knowledge:
 - packets can be lost, dropped at router due to full buffers
 - sender knows when packet has been dropped
 - sender only re-sends if packet **known** to be lost



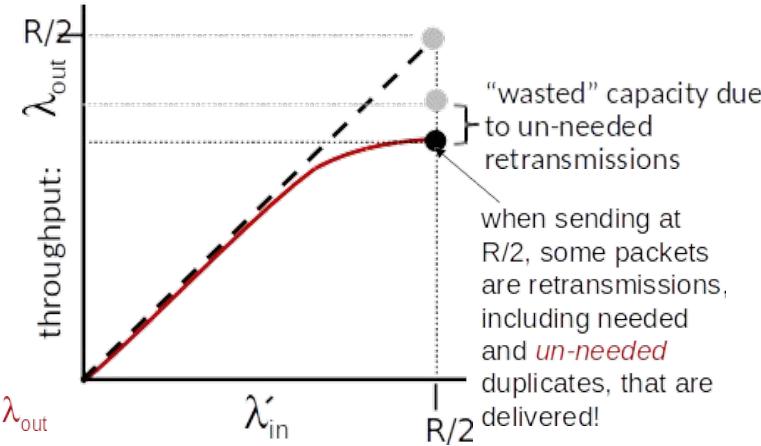
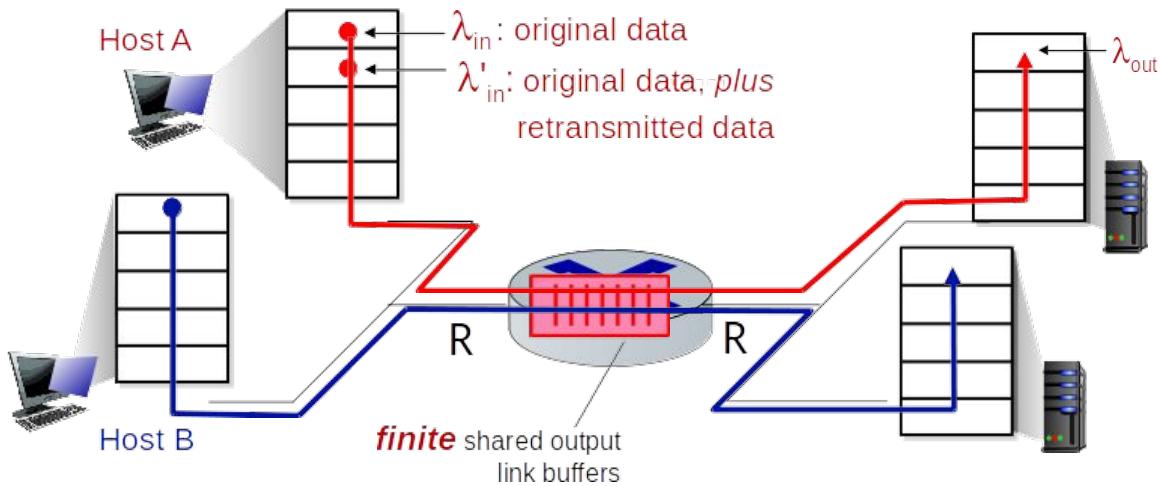
Causes/Costs of Congestion: Scenario 2: case 2

- **Idealization: Some** perfect knowledge:
 - packets can be lost, dropped at router due to full buffers
 - sender knows when packet has been dropped
 - sender only re-sends if packet **known** to be lost



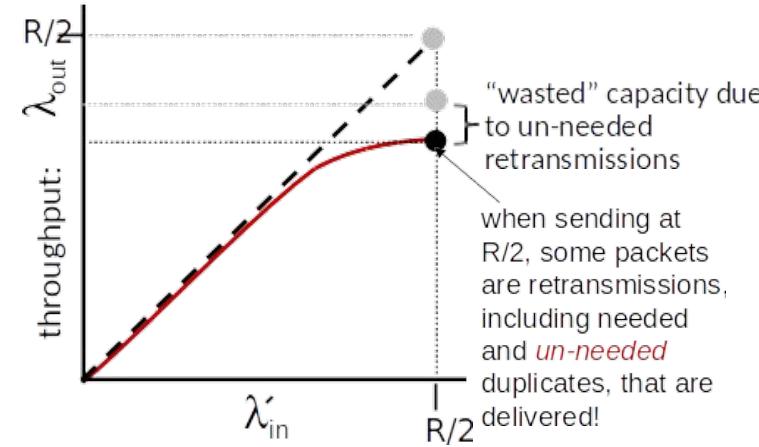
Causes/Costs of Congestion: Scenario 2: case 3

- **Realistic scenario:** un-needed duplicates
 - packets can be lost, dropped at router due to full buffers - requiring retransmissions
 - sender times out prematurely, sending **two** copies, **both** of which are delivered



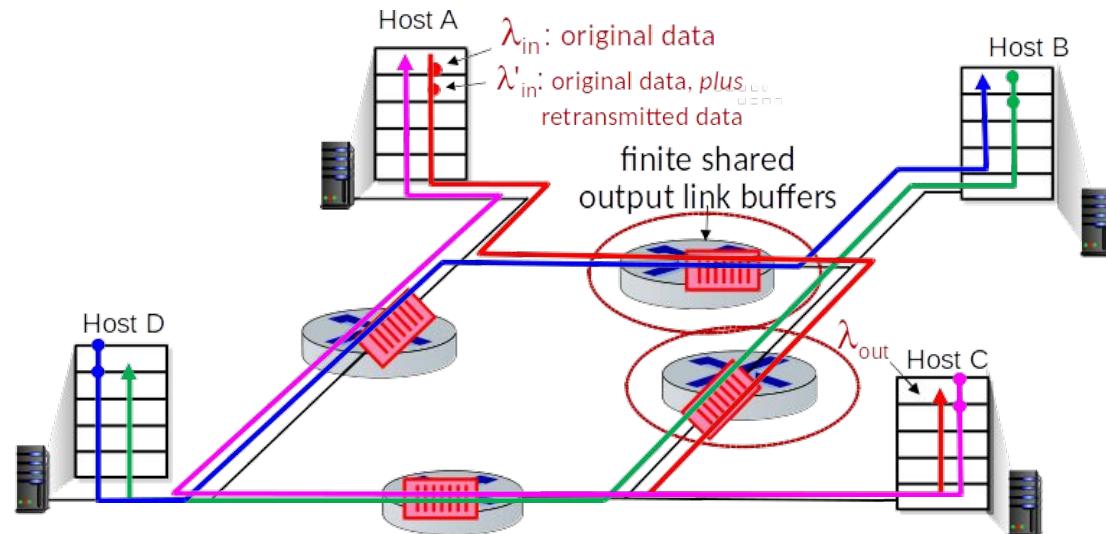
Causes/Costs of Congestion: Scenario 2: case 3

- **Realistic scenario:** un-needed duplicates
 - packets can be lost, dropped at router due to full buffers - requiring retransmissions
 - sender times out prematurely, sending **two** copies, **both** of which are delivered
- **“costs” of congestion:**
 - more work (retransmission) for given receiver throughput
 - unneeded retransmissions: link carries multiple copies of a packet
 - decreasing maximum achievable throughput



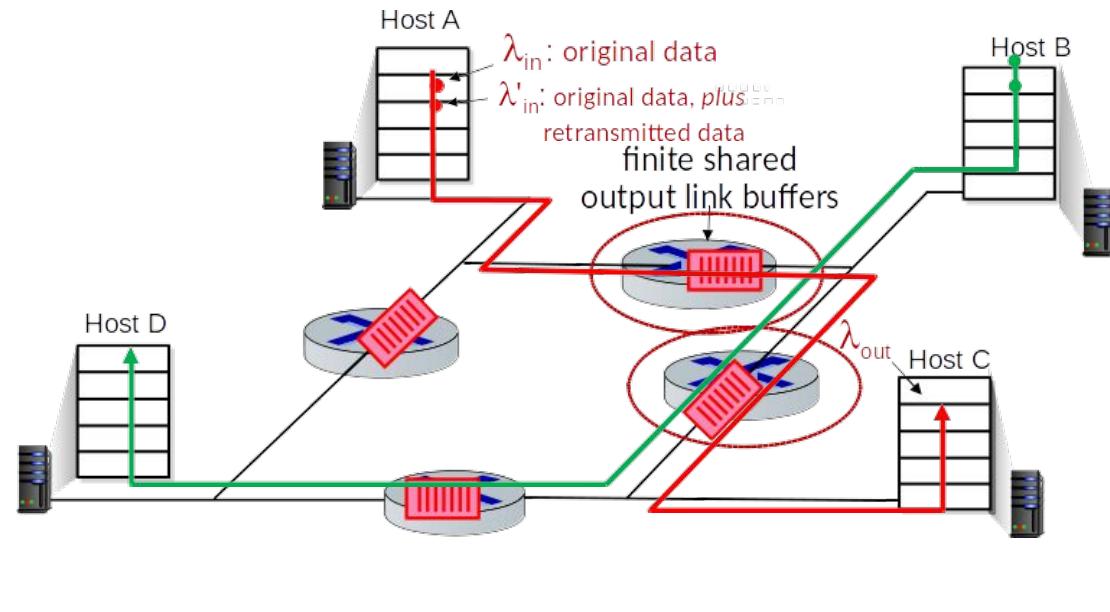
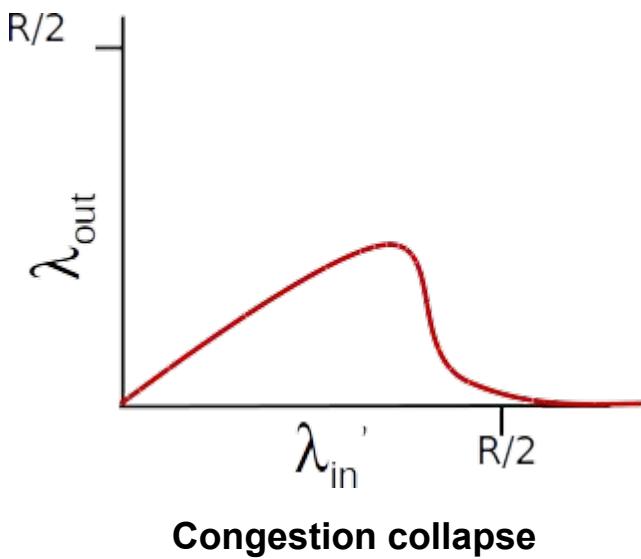
Causes/Costs of Congestion: Scenario 3

- four senders
- multihop paths
- timeout/retransmit
- Q: what happens as λ_{in} and λ'_{in} increase ?
- A: as red λ'_{in} increases, all arriving blue pkts at upper queue are dropped, blue throughput will be 0



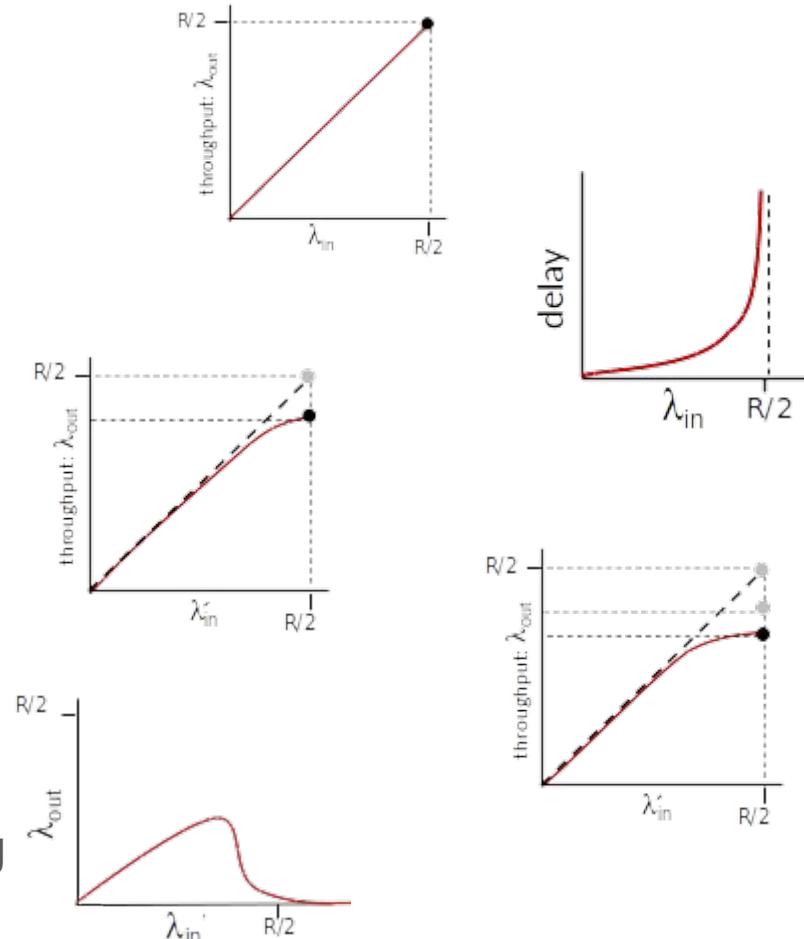
Causes/Costs of Congestion: Scenario 3

- another “cost” of congestion:
 - when packet dropped, any “upstream” transmission capacity 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



Causes and costs of congestion

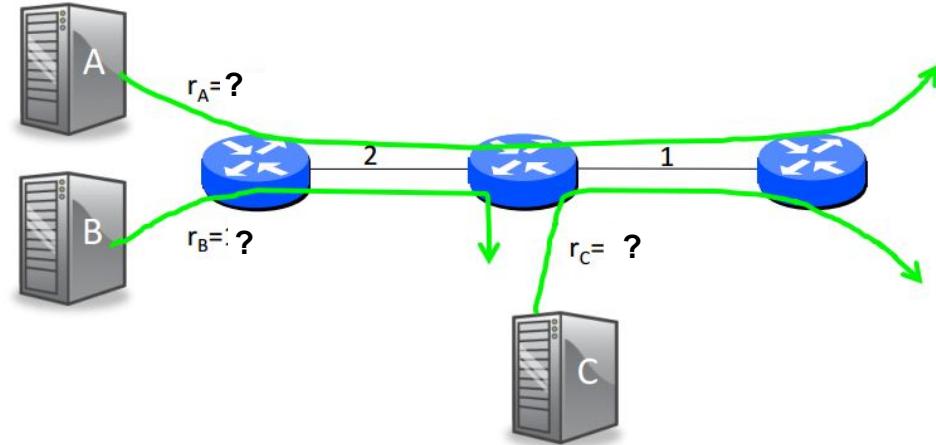
- What causes congestion:
 - buffers get full and drop packets
- Costs of congestions
 - large delays
 - If packets are dropped then retransmissions can make congestion even worse
 - unneeded retransmissions in case of premature timeout
 - When packets are dropped, they waste resources upstream before they were dropped
- Congestion control
 - how a network protocol or system tries to not overload the network between two endpoints.
 - by **limiting the rate** of data on a link

Goals of Congestion Control

- High throughput: keep links busy and flows fast
- fairness
- respond quickly to changing network conditions
- distributed control

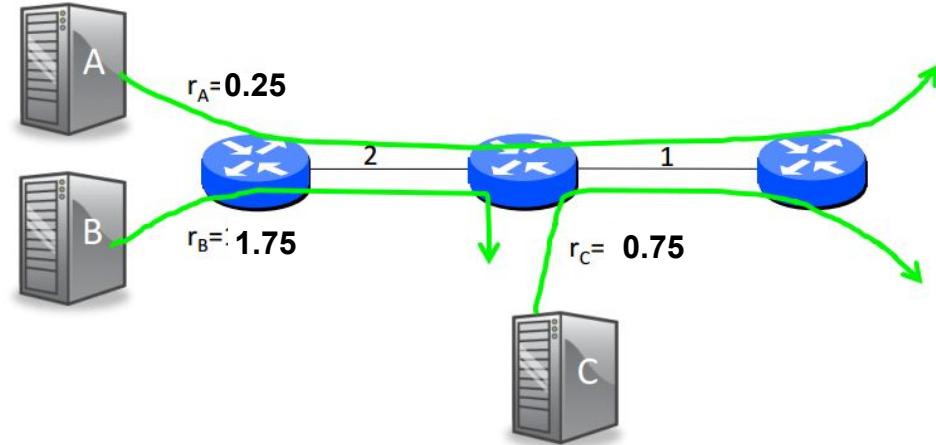
Congestion control

Bandwidth allocation



Congestion control

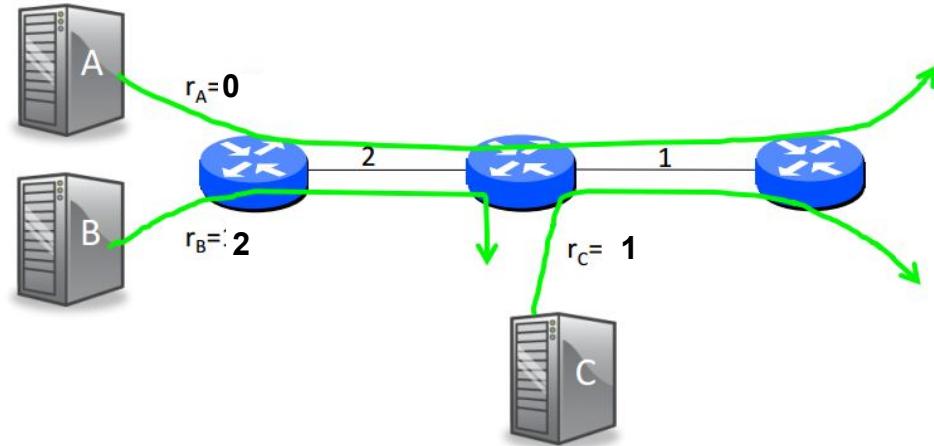
Bandwidth allocation



- $r_A = 0.25 \quad r_B = 1.75 \quad r_C = 0.75 \rightarrow \text{Total throughput} = 2.75$

Congestion control

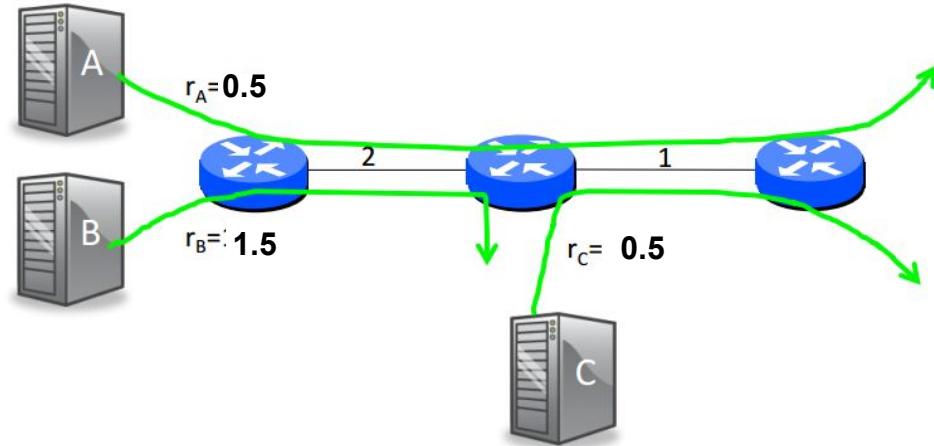
Bandwidth allocation



- $r_A = 0.25 \quad r_B = 1.75 \quad r_C = 0.75 \rightarrow \text{Total throughput} = 2.75$
- $r_A = 0 \quad r_B = 2 \quad r_C = 1 \rightarrow \text{Total throughput} = 3$

Congestion control

Bandwidth allocation



- $r_A = 0.25 \quad r_B = 1.75 \quad r_C = 0.75 \rightarrow \text{Total throughput} = 2.75$
- $r_A = 0 \quad r_B = 2 \quad r_C = 1 \rightarrow \text{Total throughput} = 3$
- $r_A = 0.5 \quad r_B = 1.5 \quad r_C = 0.5 \rightarrow \text{Total throughput} = 2.5$

Last allocation is fairer. How can we define fair allocation?

Congestion control: Bandwidth allocation

- Good allocation is both efficient and fair
 - **Efficient** means most capacity is used but there is no congestion
 - **Fair** means every sender gets a reasonable share of the network

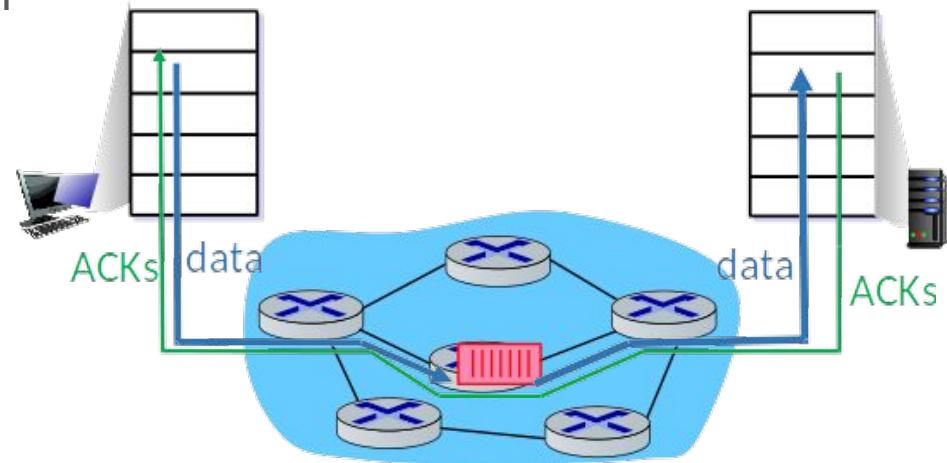
Approaches to Congestion Control

- At the end hosts
 - End-end congestion control
- At the network
 - **Network-assisted congestion control:**
 - Routers provide clear signal to the hosts
 - Congestion is detected early, before loss happening

Approaches to Congestion Control

End-end congestion control

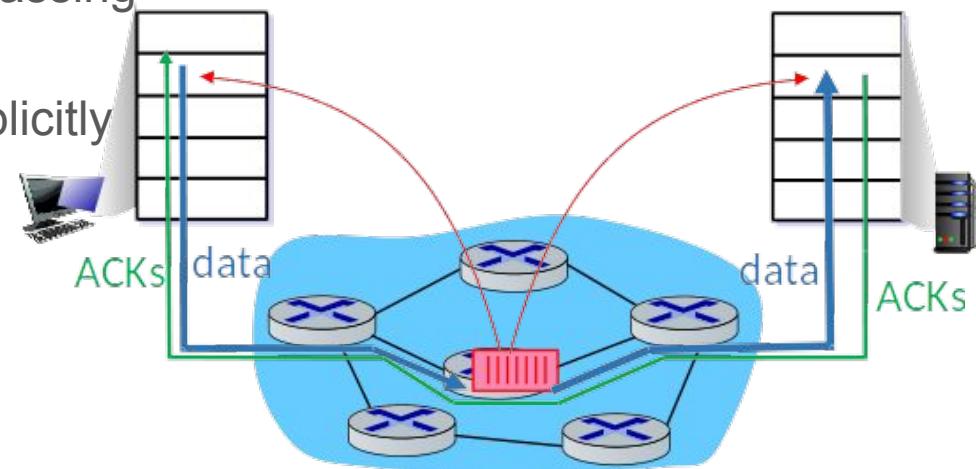
- No explicit feedback from network
- Congestion **inferred** from end-system observed loss, delay
- Approach taken by TCP



Approaches to 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



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**

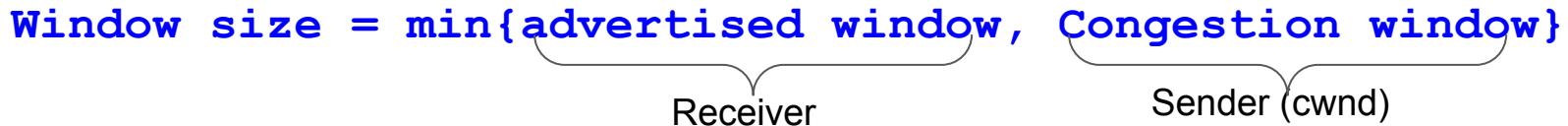
TCP Congestion Control

- Classic TCP:
 - loss-based, end-to-end
 - Additive increase, multiplicative decrease / Slow start
 - TCP Tahoe, TCP Reno, TCP CUBIC
- Enhanced TCPs:
 - Delay-based congestion control
 - Explicit congestion notification
- TCP fairness

Congestion Control in TCP

- Limit send rate when network is congested
 - How to perceive congestion?
 - Implicit end-to-end feedback?
 - ACK received?
 - Ack not received?
 - How to limit/change send rate?
- Reacts to events observable at the end host (e.g. packet loss).
- Limit the sending rate by exploiting TCP's sliding window

Window size = min{advertisised window, Congestion window}



The diagram illustrates the formula for calculating the window size. It shows two components: 'advertisised window' underlined and 'Congestion window' underlined. A bracket labeled 'Receiver' spans both underlined terms. Another bracket labeled 'Sender (cwnd)' spans the 'Congestion window' term.

- How does TCP decide the value of cwnd?

TCP Congestion Control: Additive Increase Multiplicative Decrease

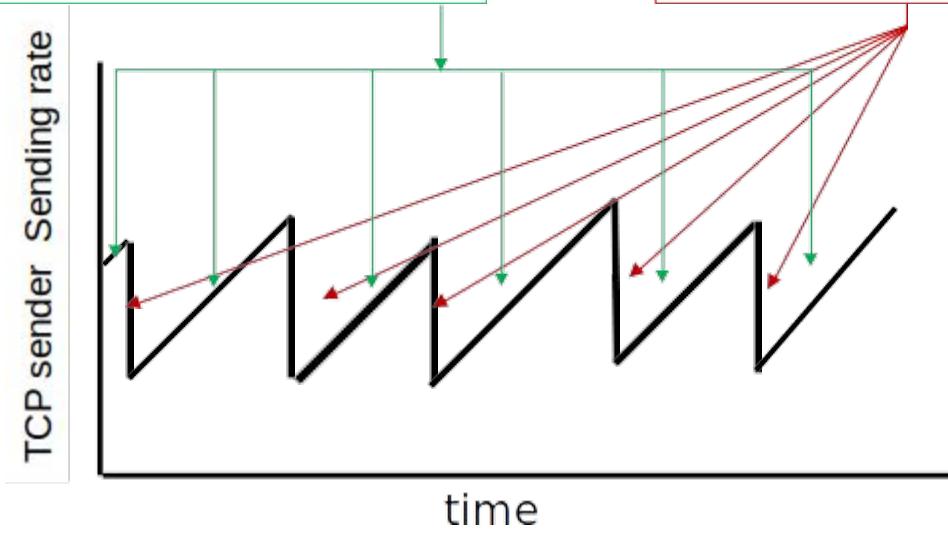
- **approach:** sender increases sending rate (window size), probing for usable bandwidth, until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

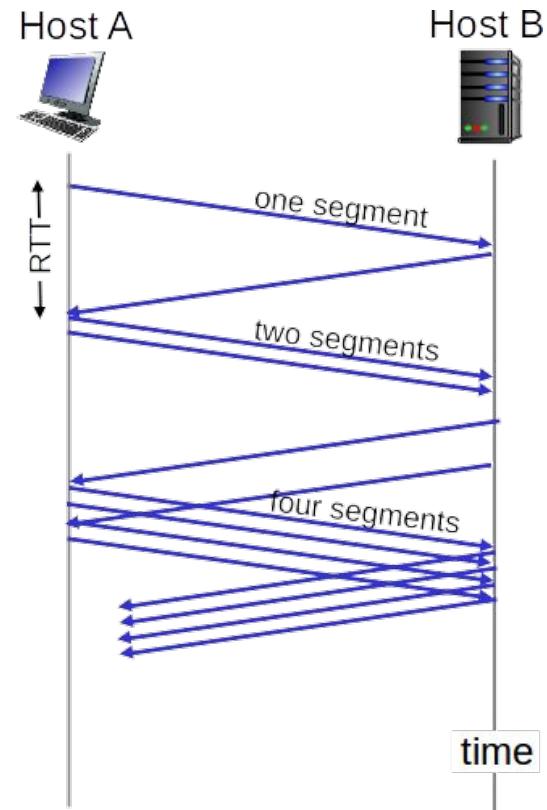
cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP Congestion control: TCP slow start

- when connection begins, increase rate exponentially until first loss event:
 - initially **cwnd** = 1 MSS
 - double **cwnd** every RTT
 - done by incrementing cwnd for every ACK received:
 - $cwnd += \text{MSS} * (\text{MSS}/\text{cwnd})$
 - **summary:** initial rate is slow but ramps up exponentially fast
- When should this exponential growth end?
 - If there is a loss event
 - When **cwnd == ssthresh**



TCP Slow Start

- benefit of slow start: initial rate is slow but ramps up exponentially fast to sense network capacity
- Exponentially increase congestion window to sense network capacity
- “Slow” compared to prior approach

MSS=500 bytes

RTT= 200 msec

initial sending rate= $500 * 8 / (20 * 10^{-3}) = 20 \text{ kbps}$

TCP Congestion Control: detecting, reacting to loss

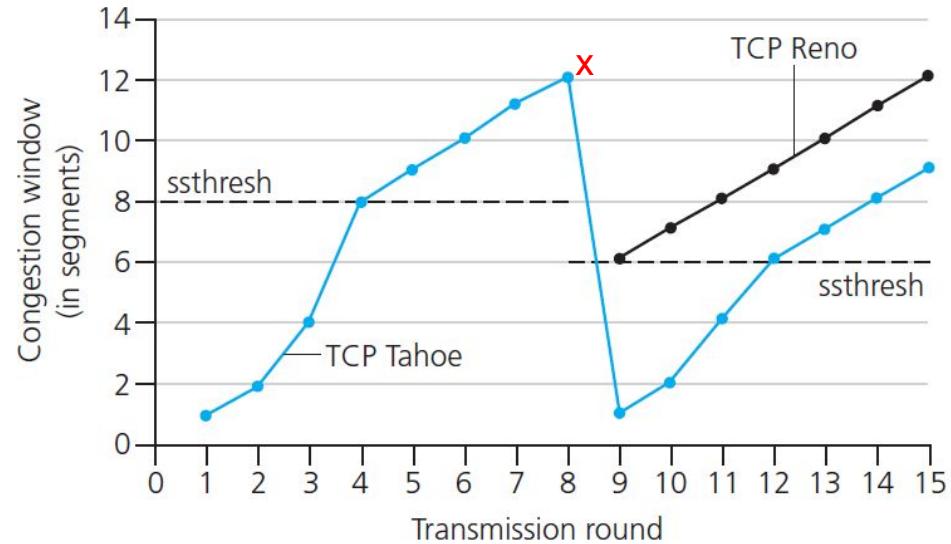
- loss indicated by timeout:
 - **cwnd** set to 1 MSS;
 - window then grows exponentially (as in slow start) to threshold, then grows linearly
 - When growing linearly, the cwnd is increased by MSS (1 segment) each round trip time
- loss indicated by 3 duplicate ACKs: **TCP RENO**
 - dup ACKs indicate network capable of delivering some segments
 - **cwnd** is cut in half window then grows linearly
- **TCP Tahoe** always sets cwnd to 1 (loss due to timeout or 3 duplicate acks)

TCP: switching from slow start to Congestion Avoidance

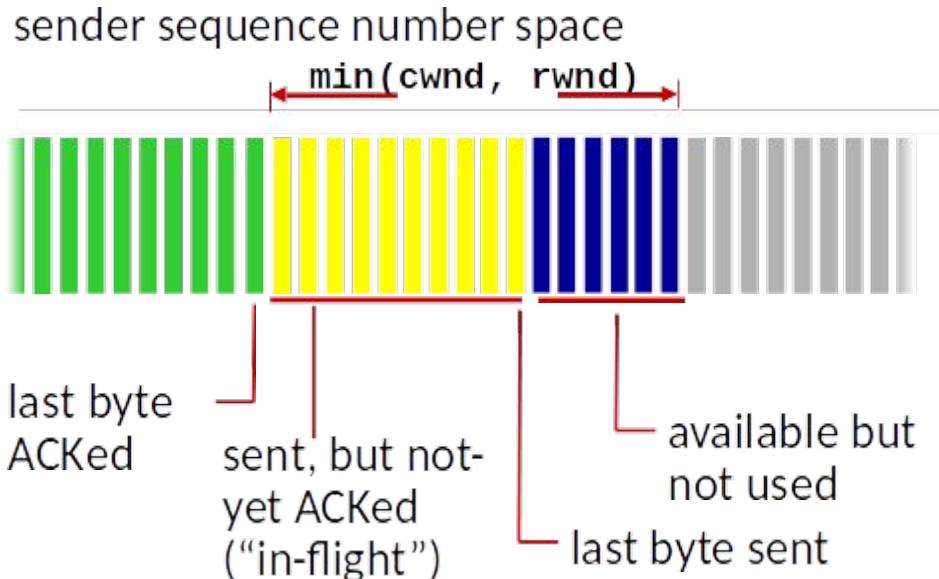
Q: when should the exponential increase switch to linear?

A: when **cwnd** gets equal to **ssthresh**

- Initial value of **ssthresh** should be high (<https://tools.ietf.org/html/rfc5681>)
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event



TCP Congestion Control: details



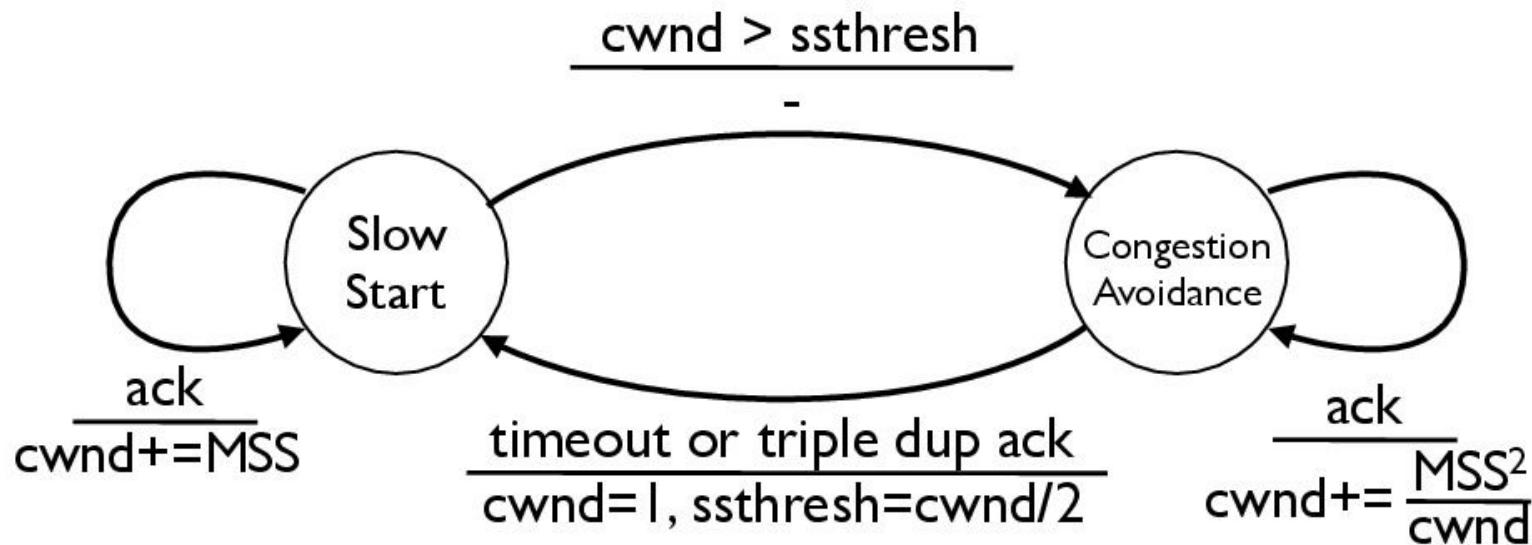
TCP sending behavior:

- roughly: send cwnd bytes, wait RTT for ACKS, then send more bytes

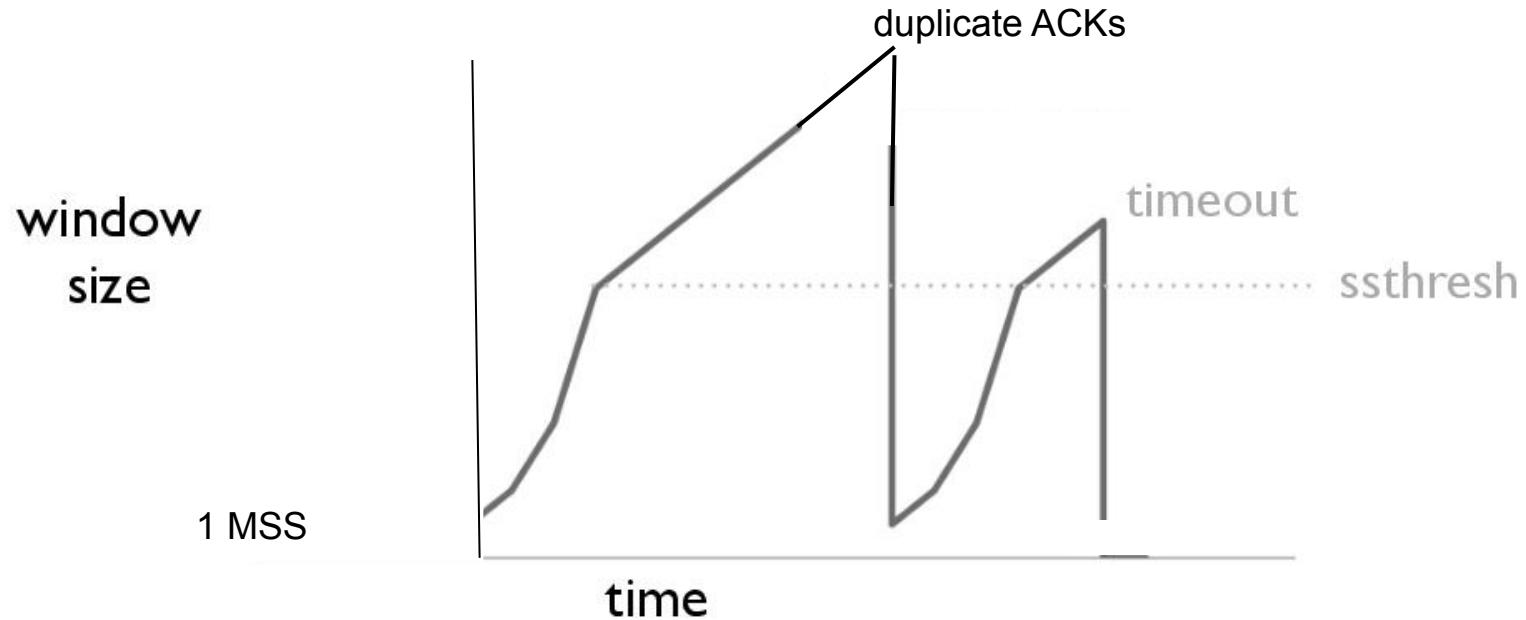
$$\text{TCP rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- TCP sender limits transmission: $\text{LastByteSent} - \text{LastByteAcked} \leq \min(\text{cwnd}, \text{rwnd})$
- cwnd is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

TCP Tahoe FSM



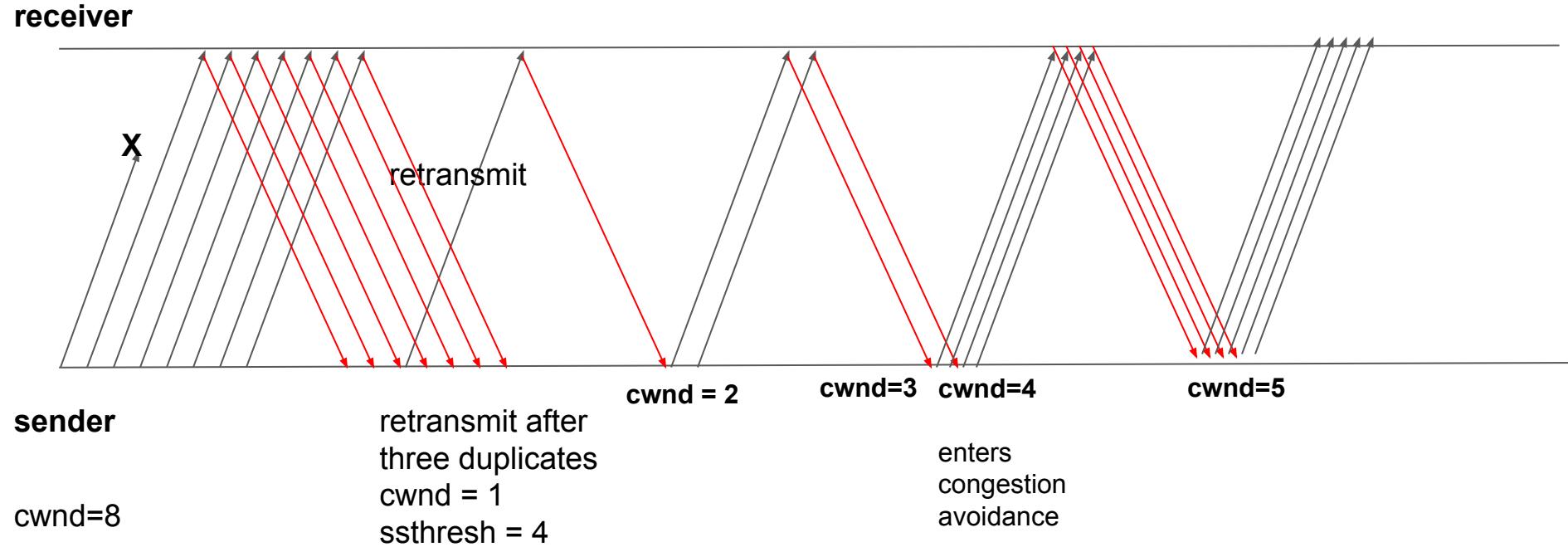
TCP Tahoe Behavior



TCP Congestion Control

- Example: TCP Tahoe
 - When in congestion avoidance step, the window size is increased linearly (additive increase)
 - increase by MSS (1 segment) each round trip time or $MSS^2/cwnd$ for each ack
 - example:
 - MSS=1460 bytes, cwnd=14600
 - 10 segments are sent within an RTT
 - Each arriving ack increases the congestion windows size by 1/10 MSS
 - the value of the congestion window will have increased by one MSS after 10 ACKs when all 10 segments have been received

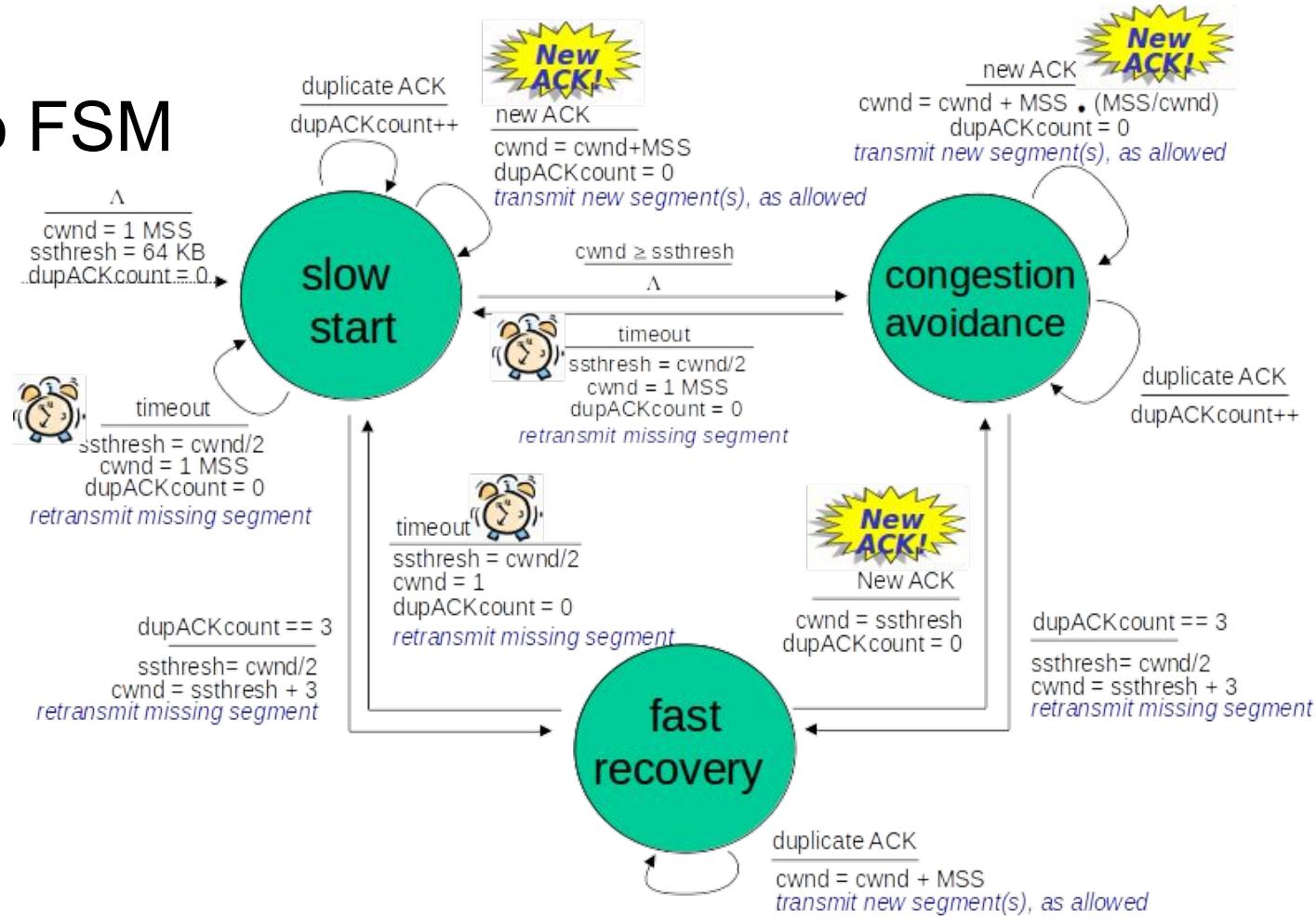
TCP Tahoe Walkthrough



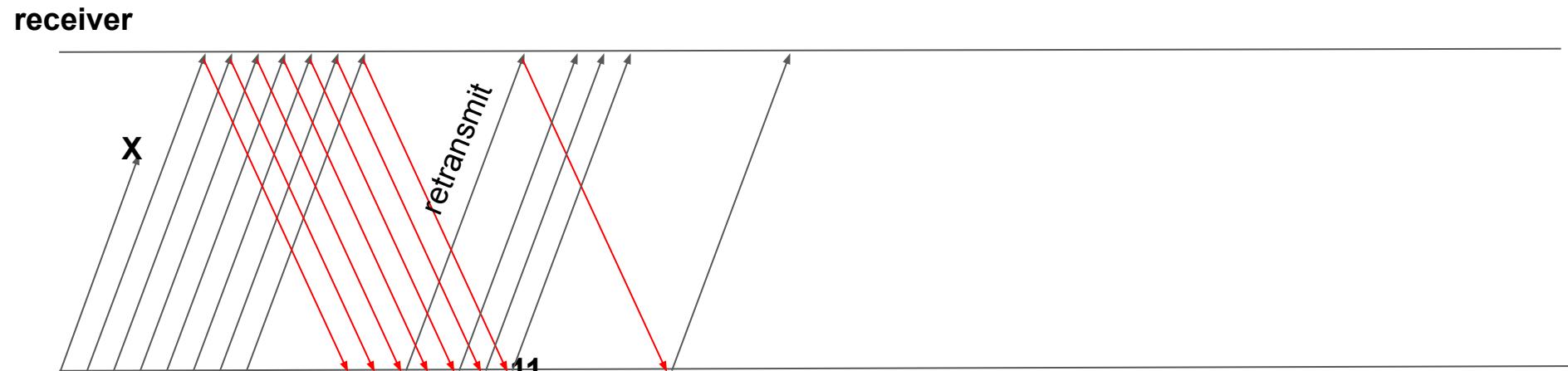
TCP Reno

- Same as TCP Tahoe on timeout
- Introduce another states: **Fast recovery**
- Start in slow start state
- loss indicated by 3 duplicate ACKs: different behavior from Tahoe
 - dup ACKs indicate network capable of delivering some segments
 - **ssthresh = cwnd/2**
 - **cwnd = cwnd/2 + 3 MSS**
- loss indicated by timeout: same behavior as Tahoe
 - **cwnd = 1 MSS**
 - window then grows exponentially (as in slow start) to threshold, then grows linearly

TCP Reno FSM



TCP Reno Walkthrough



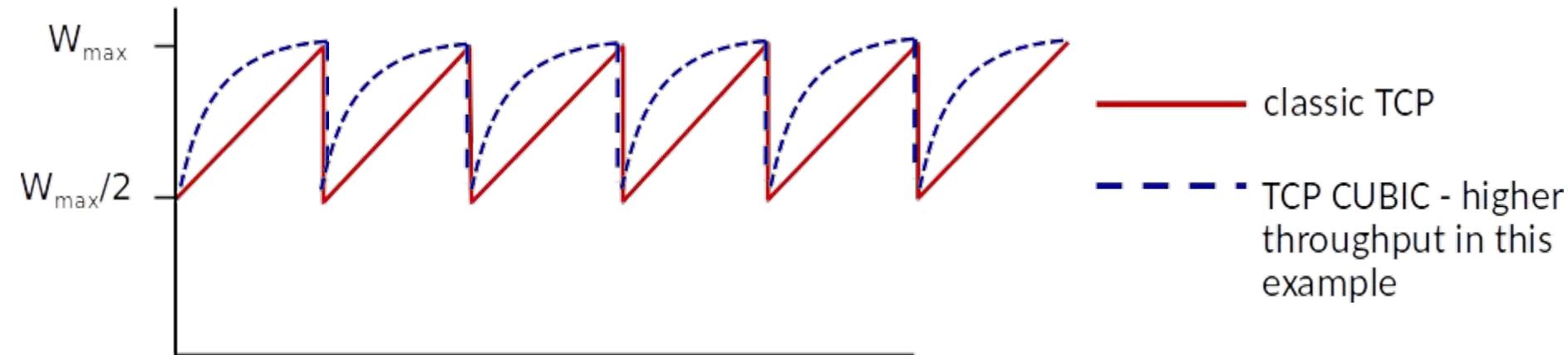
sender

cwnd=8

- fast recovery
- cwnd = $8/2 + 3 = 7$**
- ssthresh = 4
- retransmit after three duplicates

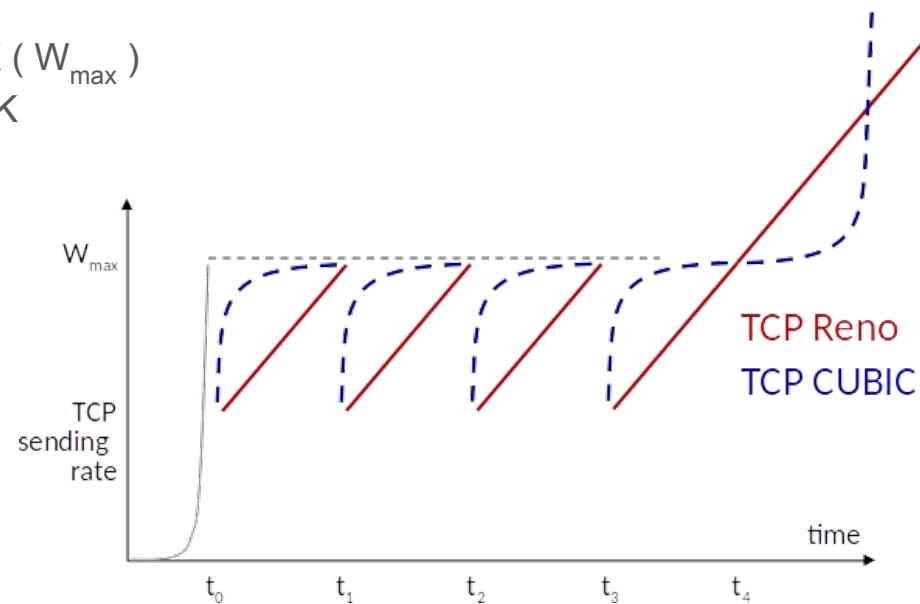
TCP CUBIC (RFC8312)

- 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, initially ramp to W_{\max} faster, but then approach W_{\max} more slowly



TCP CUBIC

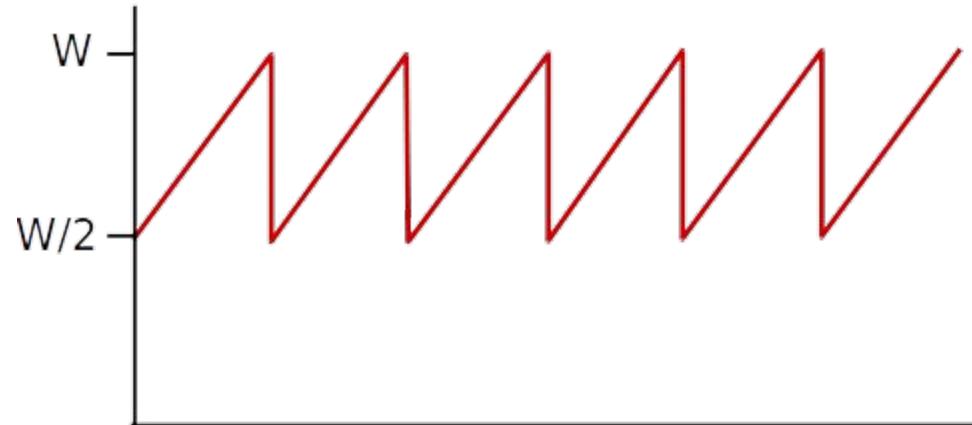
- K : point in time when TCP window size will reach W_{\max}
 - K itself is tuneable
- increase W as a function of the **cube** of the distance between current time and K
 - larger increases when further away from K (W_{\max})
 - smaller increases (cautious) when nearer K
- TCP CUBIC default in Linux, most popular TCP for popular Web servers



TCP throughput

- avg. TCP throughput as function of window size and RTT?
 - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
 - avg. window size (# in-flight bytes) is $\frac{3}{4}W$
 - avg. thruput is $\frac{3}{4}W$ per RTT

$$\text{avg TCP thruput} = \frac{3}{4} \frac{W}{\text{RTT}} \text{ bytes/sec}$$



TCP throughput in terms of loss

- Average throughput of TCP in terms of loss
- The loss rate: L
 - the number of packets lost over the number of packets sent
- The number of packets sent in a cycle is: $\frac{3}{8}W^2 + \frac{3}{4}W$
- 1 packet is lost in a cycle, thus the loss rate is:
 - $1 / (\frac{3}{8}W^2 + \frac{3}{4}W)$

$$\begin{aligned}\frac{W}{2} + \left(\frac{W}{2} + 1\right) + \dots + W &= \sum_{n=0}^{W/2} \left(\frac{W}{2} + n\right) \\ &= \left(\frac{W}{2} + 1\right) \frac{W}{2} + \sum_{n=0}^{W/2} n \\ &= \left(\frac{W}{2} + 1\right) \frac{W}{2} + \frac{W/2(W/2 + 1)}{2} \\ &= \frac{W^2}{4} + \frac{W}{2} + \frac{W^2}{8} + \frac{W}{4} \\ &= \frac{3}{8}W^2 + \frac{3}{4}W\end{aligned}$$

$$L = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

TCP throughput in terms of loss

$$\text{avg TCP thruput} = \frac{3}{4} \frac{W}{RTT} \text{ bytes / sec}$$

For W large, $\frac{3}{8}W^2 \gg \frac{3}{4}W$. Thus $L \approx 8/3W^2$ or $W \approx \sqrt{\frac{8}{3L}}$. From the text, we

therefore have

$$\text{average throughput} = \frac{3}{4} \sqrt{\frac{8}{3L}} \cdot \frac{MSS}{RTT}$$

$$= \frac{1.22 \cdot MSS}{RTT \cdot \sqrt{L}}$$

TCP Futures: TCP over high-bandwidth paths

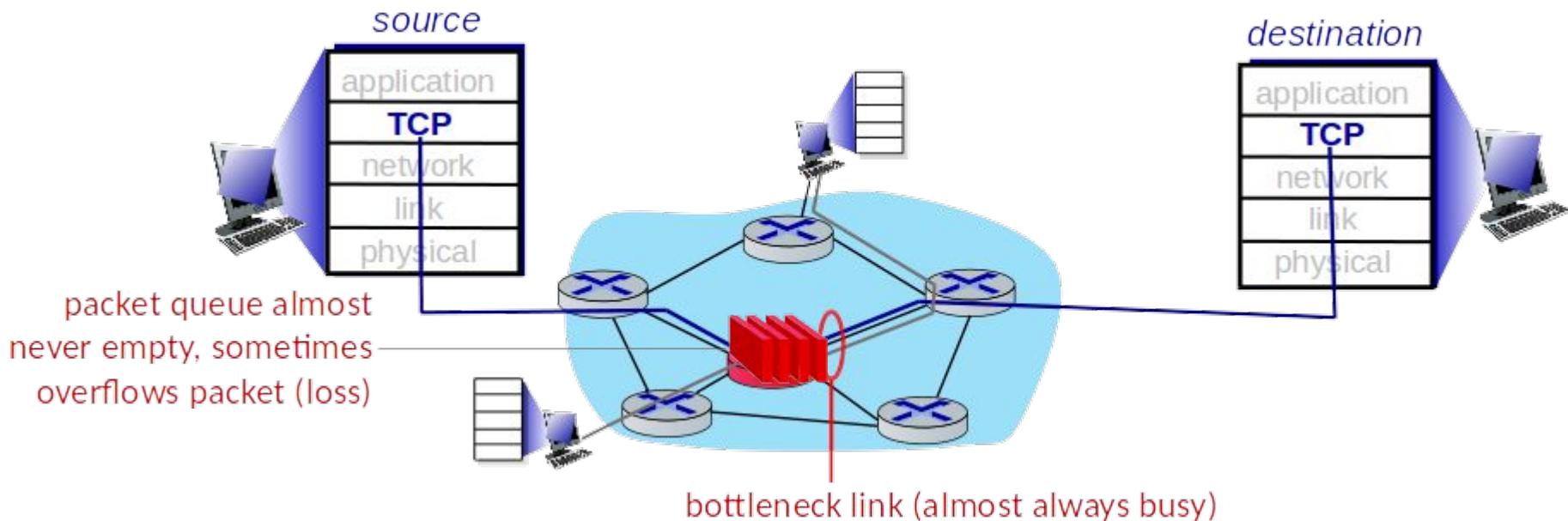
- TCP has evolved over the years
 - example: a TCP connection, **MSS=1500** bytes, **RTT=100ms**, desired throughput: **10 Gbps**
 - requires $w = 83,333$ in-flight segments
- What would happen in the case of a loss?
 - What fraction of the transmitted segments could be lost that would allow the TCP congestion-control algorithm achieve the desired 10 Gbps?
 - throughput in terms of segment loss probability L

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

- to achieve 10 Gbps throughput, need a loss rate of $L = 2 \cdot 10^{-10}$ – a very small loss rate! One loss over 5,000,000,000 segments
- new versions of TCP for high-speed

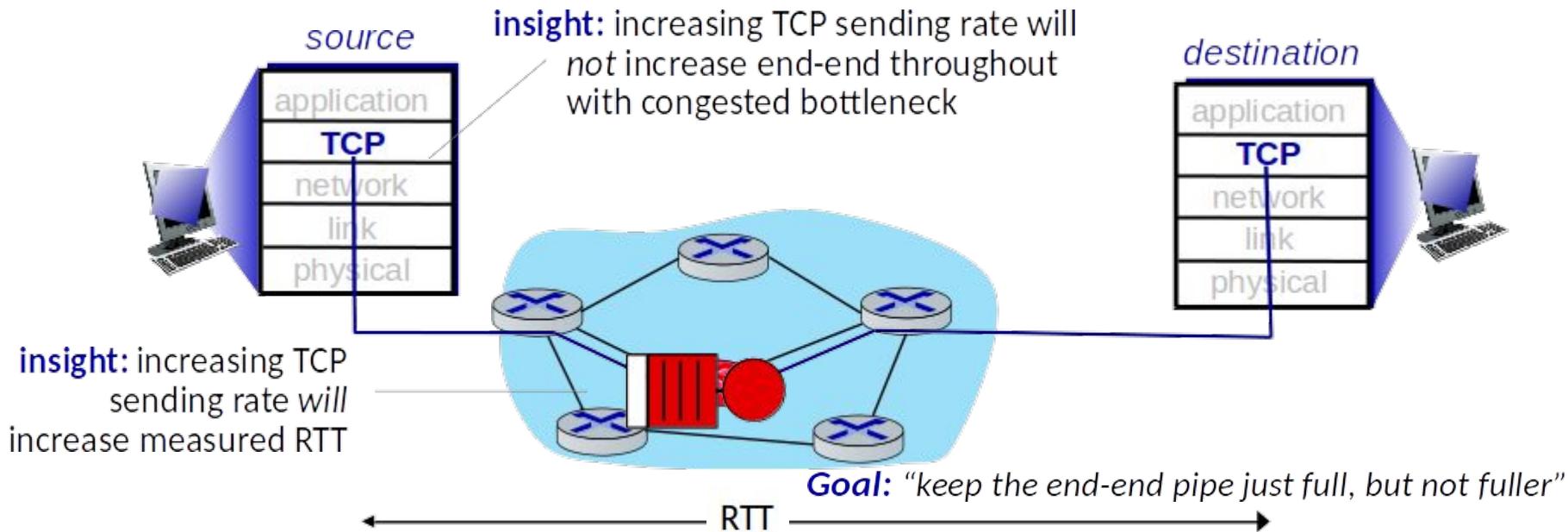
TCP and the congested “bottleneck link”

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the **bottleneck link**



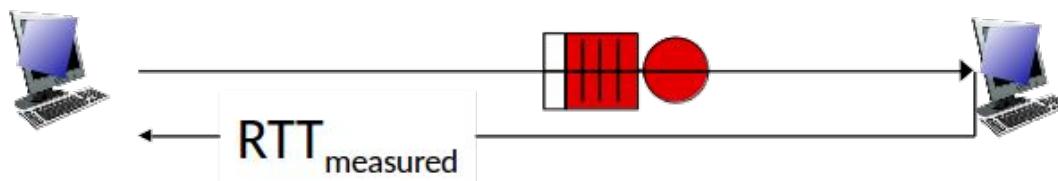
TCP and the congested “bottleneck link”

- 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:

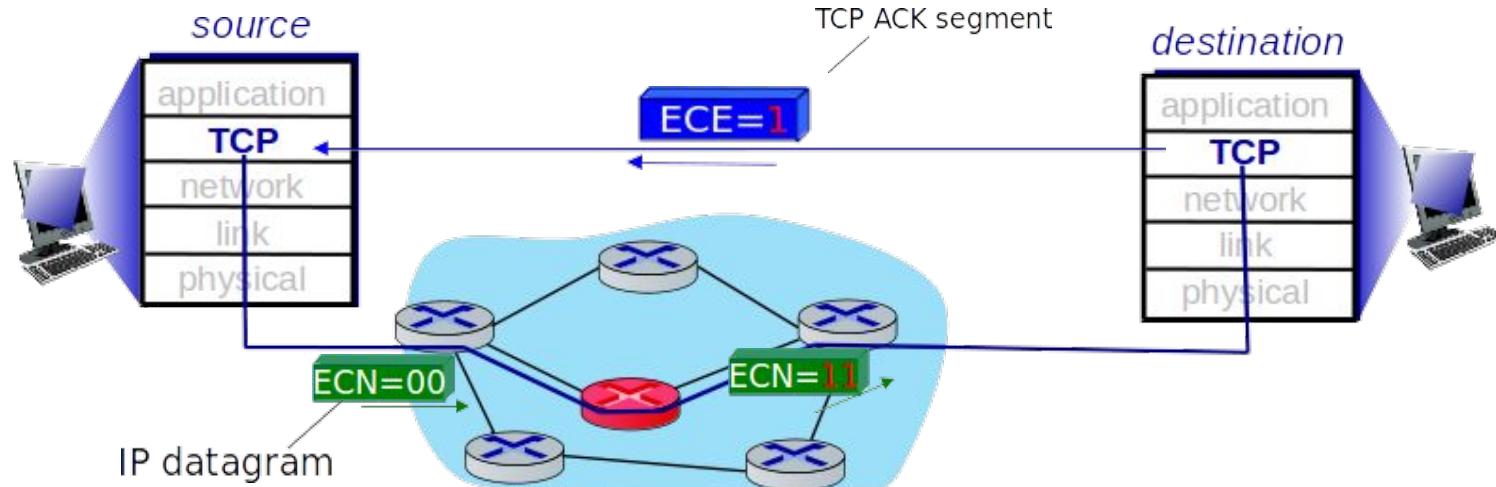
- RTT_{\min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window cwnd is $\text{cwnd}/\text{RTT}_{\min}$
 - if measured throughput “very close” to uncongested throughput
increase cwnd linearly /* since path not congested */
 - else if measured throughput “far below” uncongested throughput
decrease cwnd linearly /* since path is congested */

Delay-based TCP congestion control

- congestion control without inducing/forcing loss
- maximizing throughput (“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

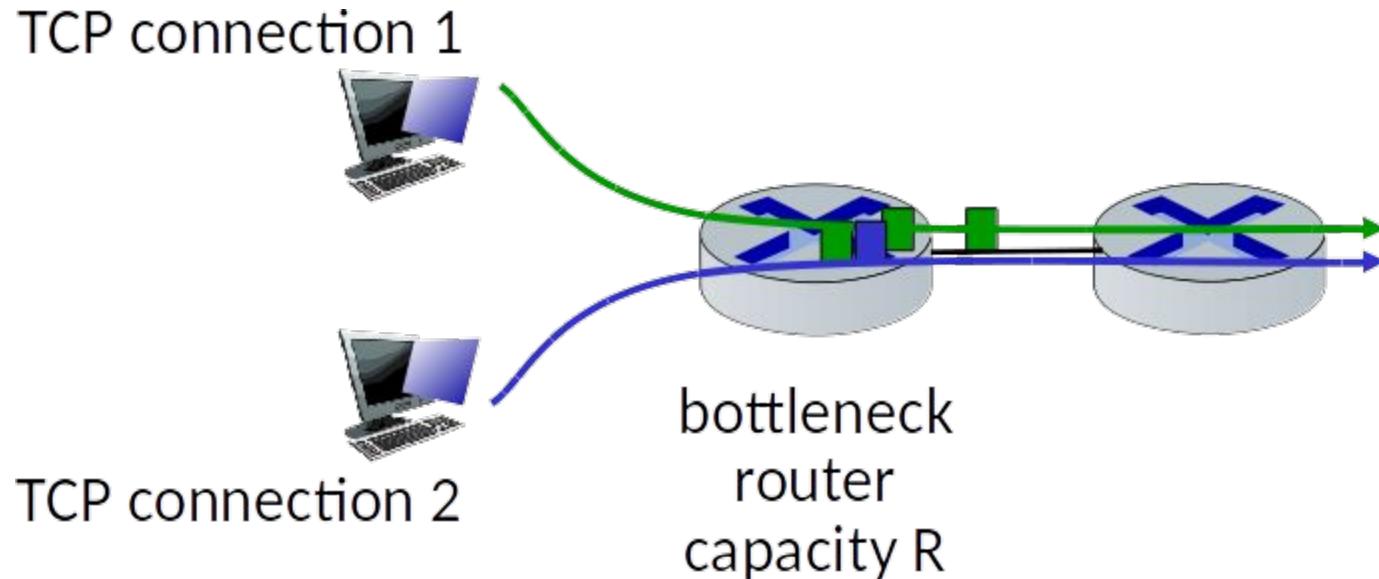
Explicit Congestion Notification (ECN): RFC 3168

- network-assisted congestion control:
- 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 receiving host
 - receiver (seeing congestion indication in IP datagram) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion



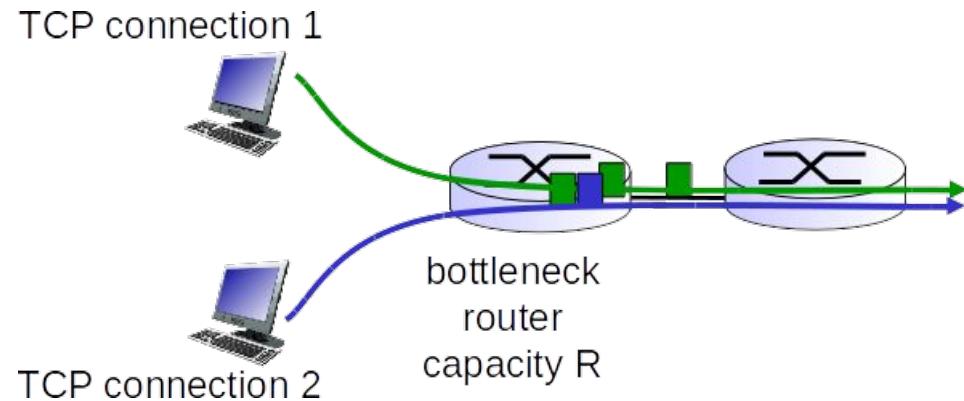
TCP Fairness

fairness goal: if K TCP sessions share same bottleneck link of bandwidth R , each should have average rate of R/K



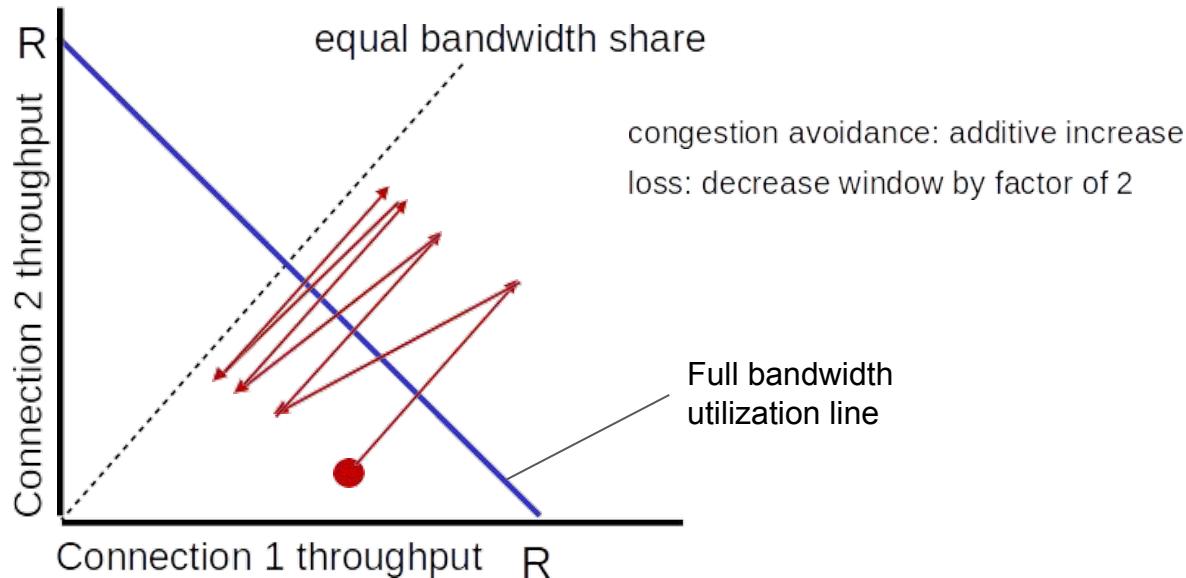
TCP Fairness

- Is TCP fair?
 - Assumption:
 - Two connections have the same MSS and RTT → if they have the same congestion window size, then they have the same throughput
 - Two connections have a large amount of data to send
 - No other TCP connection or UDP datagram traverses this shared link
 - Ignore the slow-start phase of TCP



Why is TCP fair?

- Example: two competing sessions:
 - additive increase gives slope of 1, as throughput increases
 - multiplicative decrease decreases throughput proportionally



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

Question

When multiple connections share a common bottleneck, those with a smaller RTT are able to grab the available bandwidth at that link more quickly as it becomes free.

TRUE

FALSE

Fairness

- **Fairness and UDP**
 - multimedia apps often do not use TCP
 - do not want rate throttled by congestion control
 - instead use UDP:
 - send audio/video at constant rate, tolerate packet loss
 - Not fair from TCP perspective
- **Fairness, parallel TCP connections**
 - application can open multiple parallel connections between two hosts
 - web browsers do this
 - e.g., link of rate R with 9 existing connections:
 - new app asks for 1 TCP, gets rate $R/10$
 - new app asks for 11 parallel TCPs, gets more than $R/2$

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
- **Evolution of transport-layer functionality**

Evolving transport-layer functionality

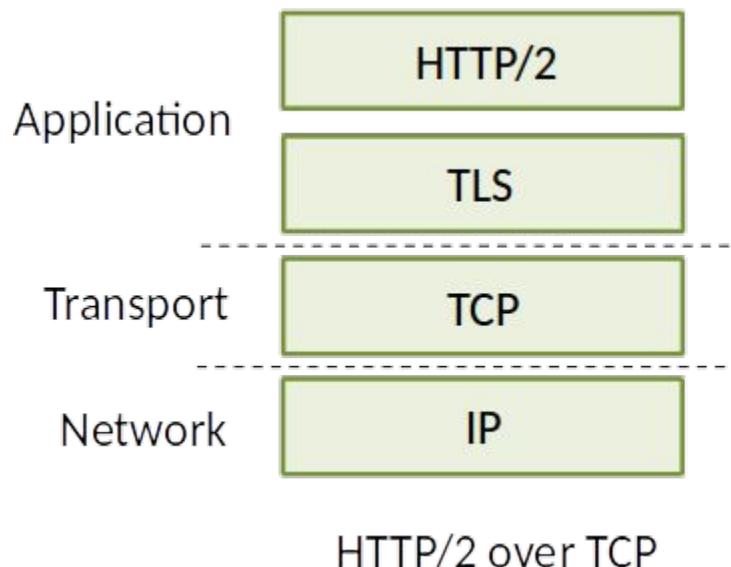
- TCP, UDP: principal transport protocols for 40 years
- different “flavors” of TCP developed, for specific scenarios:

Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets “in flight”; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, “background” TCP flows

- moving transport-layer functions to application layer, on top of UDP
- HTTP/3: QUIC

QUIC: Quick UDP Internet Connections

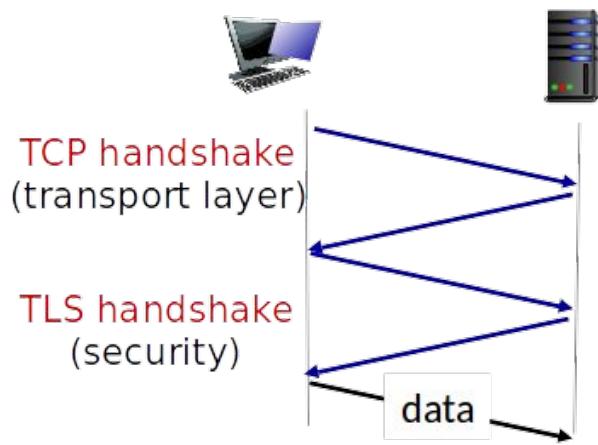
- application-layer protocol, on top of UDP
 - increase performance of HTTP
 - deployed on many Google servers, apps (Chrome, mobile YouTube app)



QUIC: Quick UDP Internet Connections

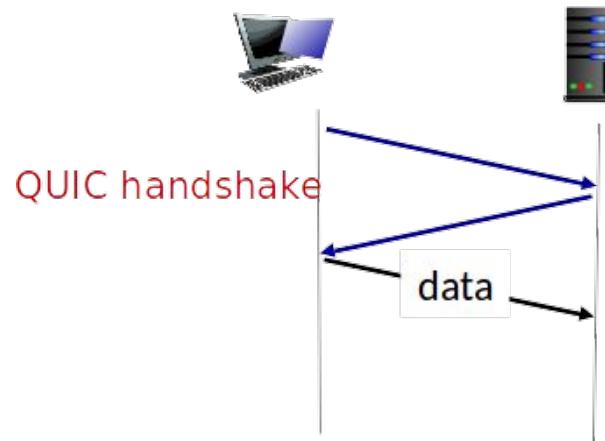
- adopts approaches we've studied in this chapter for connection establishment, error control, congestion control
 - **error and congestion control:** "Readers familiar with TCP's loss detection and congestion control will find algorithms here that parallel well-known TCP ones." [from QUIC specification]
- **connection establishment:** reliability, congestion control, authentication, encryption, state established in one RTT
- multiple application-level "streams" multiplexed over single QUIC connection
 - separate reliable data transfer, security
 - common congestion control

QUIC: Connection establishment



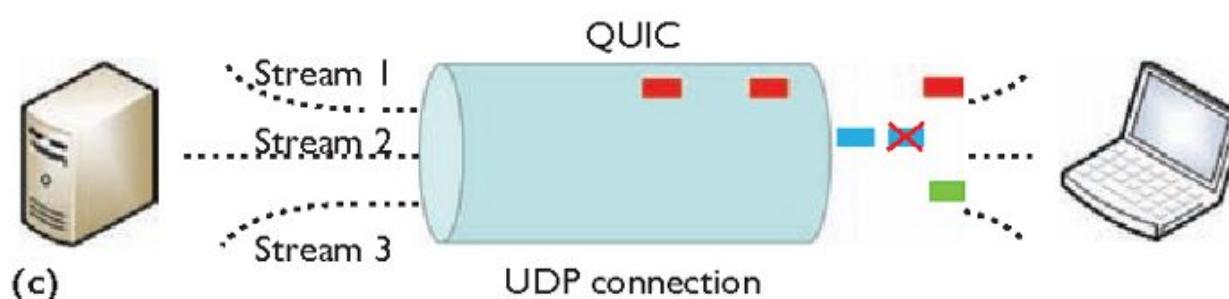
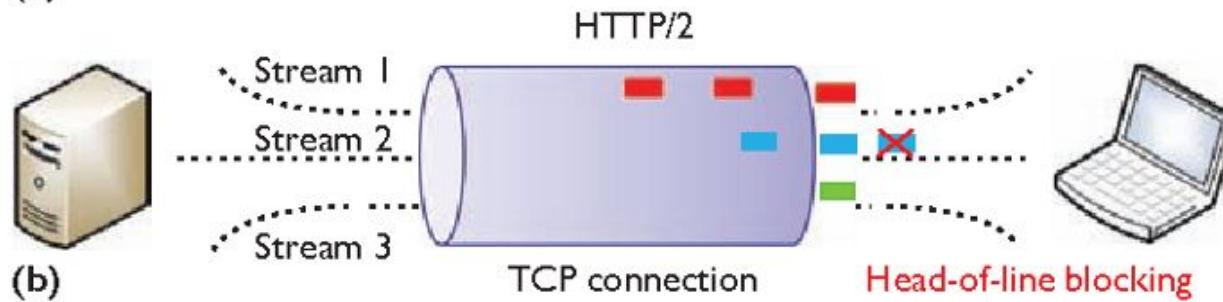
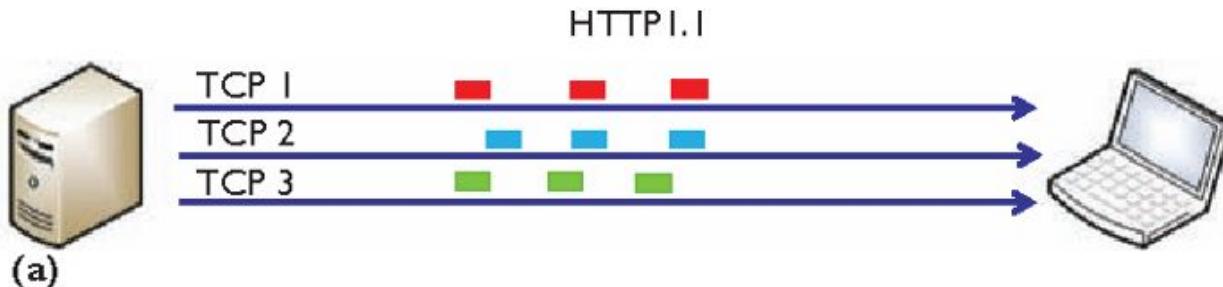
TCP (reliability, congestion control state) + TLS (authentication, crypto state)

- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake



Summary

- principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- instantiation, implementation in the Internet
 - UDP
 - TCP
- Next:
 - leaving the network “edge” (application, transport layers)
 - into the network “core”
 - two network layer chapters:
 - data plane
 - control plane

Socket Programming

Socket Programming

- **goal:** learn how to build client/server applications that communicate using socket API
 - **Socket API(Application Programming Interface):**
 - The programming interface between the application and the network
 - Part of all major OSes and languages; originally Berkeley (Unix) ~1983
 - Application developer has control of everything on the application-layer side of the socket
 - little control of the transport-layer side of the socket
 - the choice of transport protocol
 - the ability to fix a few transport layer parameters such as maximum buffer and maximum segment size

Sockets

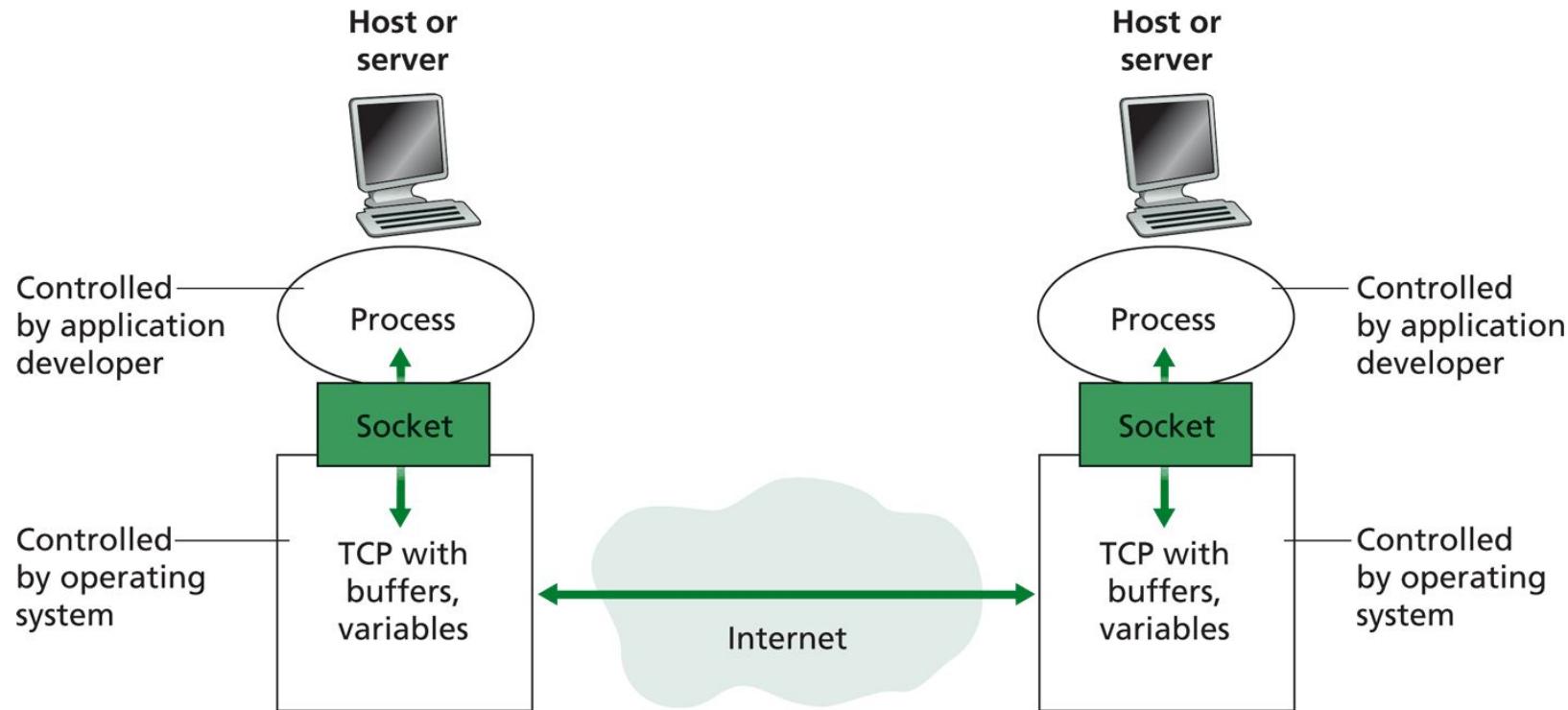


Figure 2.3 ♦ Application processes, sockets, and underlying transport

Network Applications

- General format for many network applications
 - Simple Client
 - Send requests to the server
 - Wait for a reply
 - Extract the information from the reply
 - Continue...
 - Simple Server
 - Wait to be contacted by the server
 - Server handles the Client requests
 - Multi-threaded (multi-process)
- Example:
 - File transfer: send name, get file
 - Web browsing: send URL, get page

Socket API

- Two kinds of sockets
 - Streams (TCP): reliably send a stream of bytes
 - Datagrams (UDP): unreliablely send separate messages

Socket API: UDP Socket

Client:

socket()

sendto()

recvfrom()

close()

Server

socket()

bind()

recvfrom()

sendto()

close()

Client/Server Socket Interaction: UDP

server (running on serverIP)

```
create socket, port= x:  
serverSocket =  
socket(AF_INET,SOCK_DGRAM)
```

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

```
create socket:  
clientSocket =  
socket(AF_INET,SOCK_DGRAM)
```

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket
close
clientSocket

Socket Programming with UDP

- **UDP: no “connection” between client & server**
 - no handshaking before sending data
 - sender explicitly attaches IP destination address and port # to each packet
 - receiver extracts sender IP address and port# from received packet
- **UDP: transmitted data may be lost or received out-of-order**
- **Application viewpoint:**
 - UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server

Socket Programming: example

- Application Example:
 - client reads a line of characters (data) from its keyboard and sends data to server
 - server receives the data and converts characters to uppercase
 - server sends modified data to client
 - The source codes are provided in MLS

Example App: UDP Client

Python UDPClient

```
include Python's socket  
library → from socket import *  
  
create UDP socket → clientSocket = socket(AF_INET,  
                                         SOCK_DGRAM)  
  
get user keyboard  
input → message = raw_input('Input lowercase sentence:')  
  
Attach server name, port to  
message; send into socket → clientSocket.sendto(message.encode(),  
                                                (serverName, serverPort))  
  
modifiedMessage, serverAddress =  
                                clientSocket.recvfrom(2048)  
  
print out received string → print modifiedMessage.decode()  
and close socket → clientSocket.close()
```

Example App: UDP Server

Python UDPServer

```
from socket import *
serverPort = 12000
create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port → serverSocket.bind(("", serverPort))
number 12000
print ("The server is ready to receive")
loop forever → while True:
Read from UDP socket into → message, clientAddress = serverSocket.recvfrom(2048)
message, getting client's
address (client IP and port) → modifiedMessage = message.decode().upper()
send upper case string → serverSocket.sendto(modifiedMessage.encode(),
back to this client
                                clientAddress)
```

Socket Programming with TCP

- Connection-oriented
 - Client and sever first need to handshake and establish a connection
 - client contacts server by creating TCP socket, specifying IP address, port number of server process and client process
 - To send data, client just drop data into the TCP connection via its socket
 - In UDP, the server must attach a destination address to the packet before dropping it into the socket
- client must contact the server:server process must first be running
 - server must have created socket (door) that welcomes client's contact welcoming socket
 - when contacted by client, server TCP creates new socket for server process to communicate with that particular client
- application viewpoint:
 - TCP provides reliable, in-order byte-stream transfer (“pipe”) between client and server

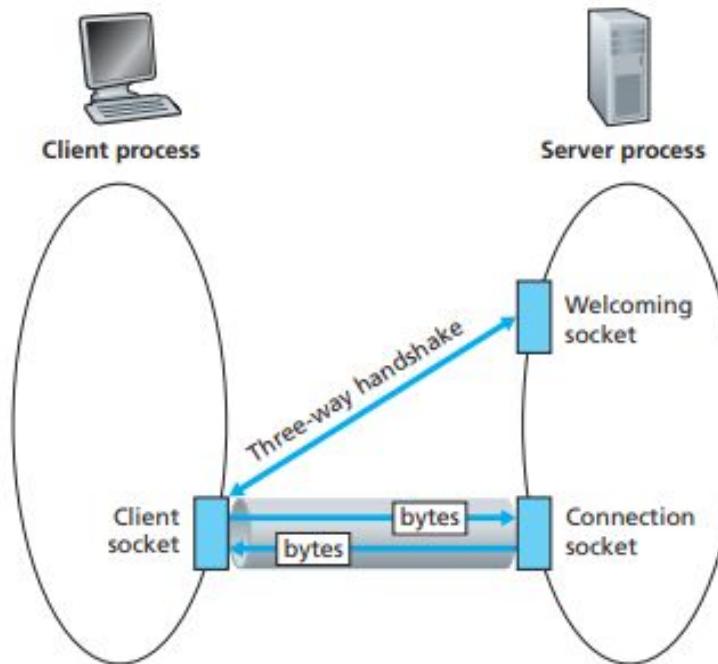


Figure 2.29 ♦ The TCP Server process has two sockets

Socket API: TCP Socket

Client:

socket()

connect()

send()

receive

close()

Server

socket()

bind()

listen()

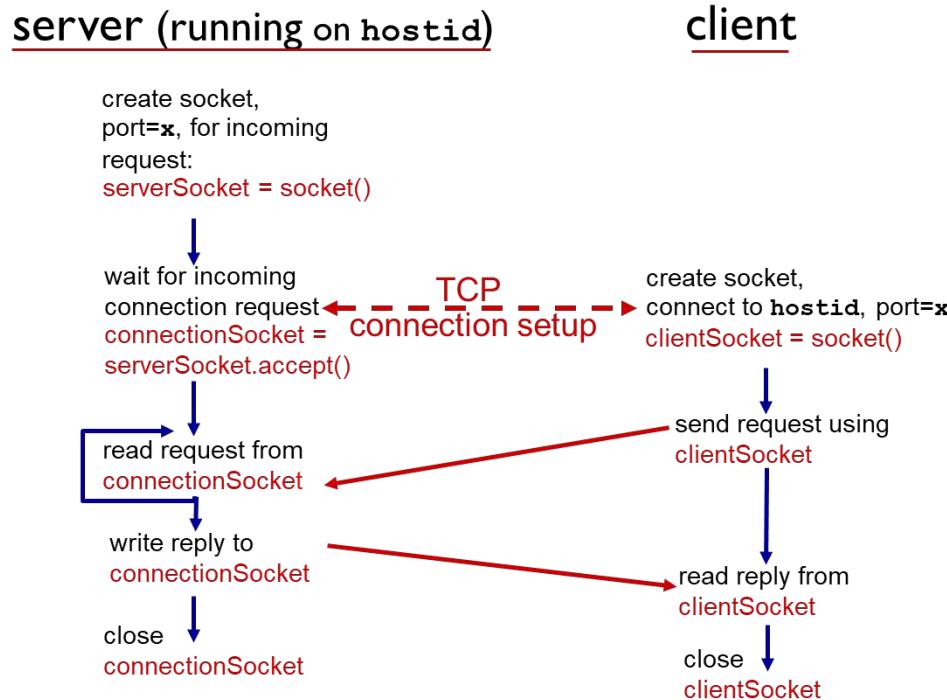
accept()

receive()

send()

close()

Client/Server Socket Interaction: TCP



Example App: TCP Client

Python TCP Client

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence.')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for
server, remote port 12000

No need to attach server
name, port



Example App: TCP Server

Python TCP Server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(("",serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
```

create TCP welcoming
socket → serverSocket = socket(AF_INET,SOCK_STREAM)

server begins listening for
incoming TCP requests → serverSocket.bind(("",serverPort))
serverSocket.listen(1)

loop forever → print 'The server is ready to receive'

server waits on accept()
for incoming requests, new
socket created on return → while True:

read bytes from socket (but
not address as in UDP) → connectionSocket, addr = serverSocket.accept()

close connection to this
client (but *not* welcoming
socket) → sentence = connectionSocket.recv(1024).decode()
capitalizedSentence = sentence.upper()
connectionSocket.send(capitalizedSentence.
encode())
connectionSocket.close()

Question

For the client-server application over TCP, does the server program have to be executed before the client program?

- A. True
- B. False

Question

For the client-server application over UDP, does the server program have to be executed before the client program?

- A. True
- B. False

Network Layer

Forwarding

outline

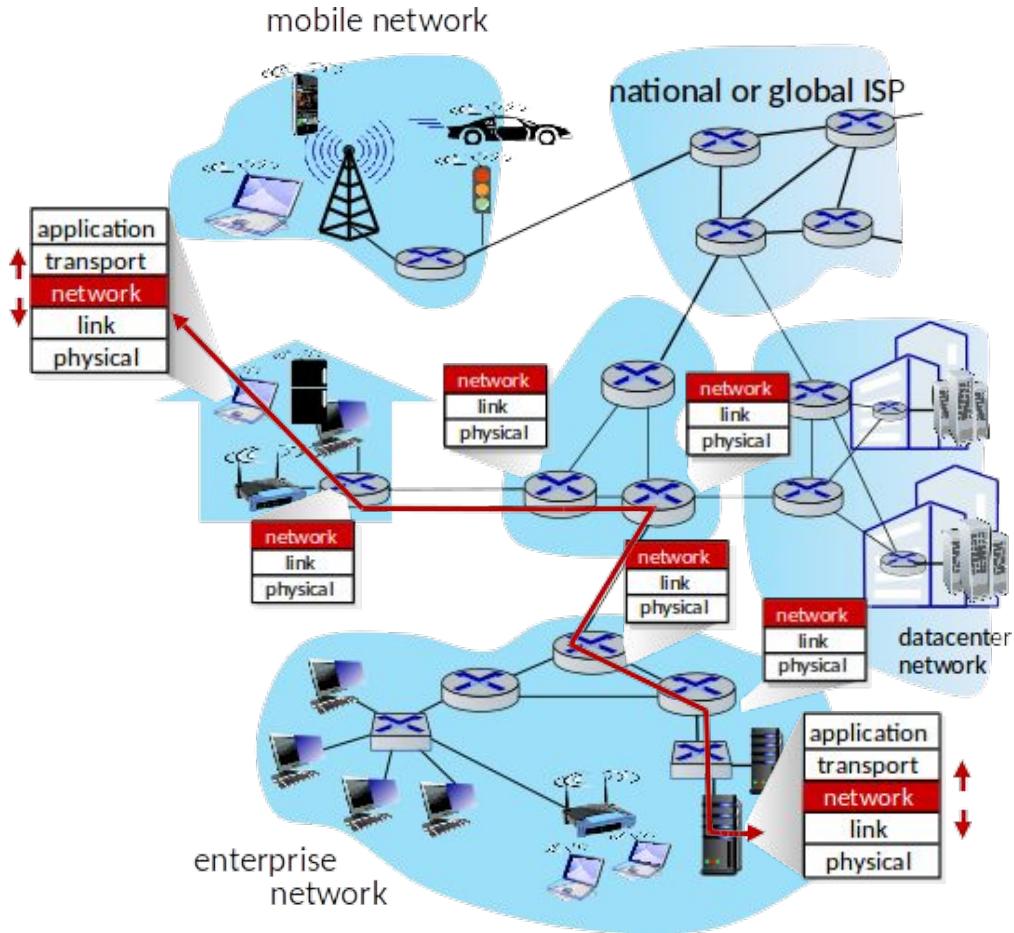
- **Overview of Network layer**
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - datagram format
 - IPv4 addressing
 - network address translation
 - IPv6
- Traditional IP forwarding and Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow: examples of match-plus-action in action

Network Layer: data plane: Goals

- Understand principles behind network layer services, focusing on data plane:
 - network layer service models
 - forwarding versus routing
 - how a router works
 - generalized forwarding
- instantiation, implementation in the Internet

Network layer services and protocols

- transport segment from sending to receiving host
 - **sender**: encapsulates segments into datagrams, passes to link layer
 - **receiver**: delivers segments to transport layer
- network layer protocols in **every Internet device**: hosts, routers
- **routers**:
 - examines header fields in all IP datagrams passing through it
 - moves datagrams from input ports to output ports to transfer datagrams along end-end path



Two key network-layer functions

- **forwarding:** move packets from router's input to appropriate router output
- **routing:** determine route taken by packets from source to destination
 - routing algorithms
- **analogy: taking a trip**
 - **forwarding:** process of getting through single interchange
 - **routing:** process of planning trip from source to destination



forwarding

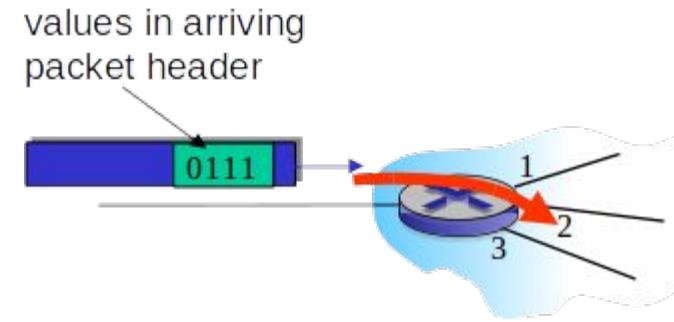


routing

Network layer: data plane, control plane

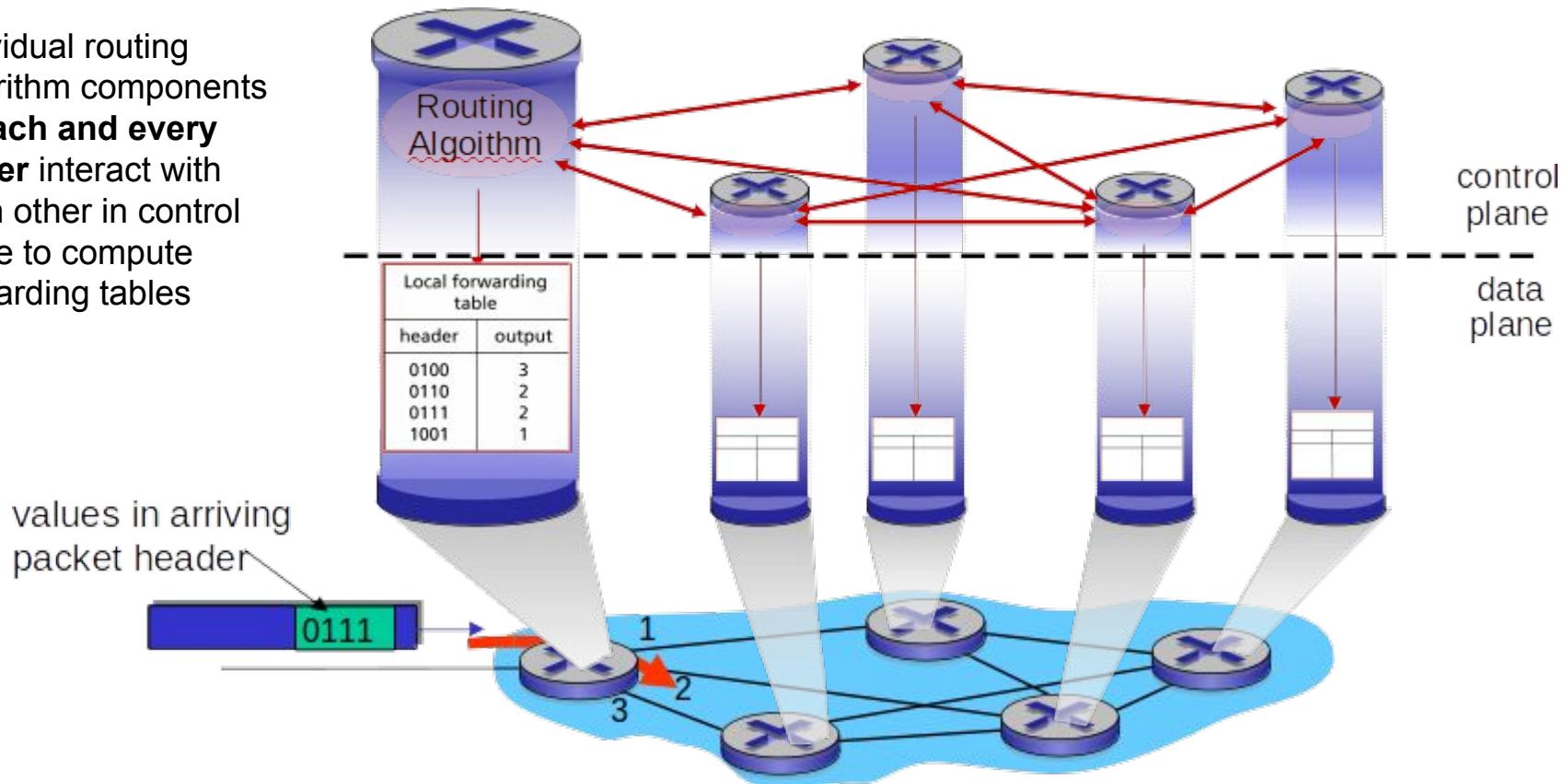
- **Data plane**
 - local, per-router function
 - determines how datagram arriving on router input port is forwarded to router output port
 - forwarding function

- **Control plane**
 - **network-wide** logic
 - determines how datagram is routed among routers along end-end path from source host to destination host
 - two control-plane approaches:
 - **traditional routing algorithms**: implemented in routers
 - **software-defined networking (SDN)**: implemented in (remote) servers



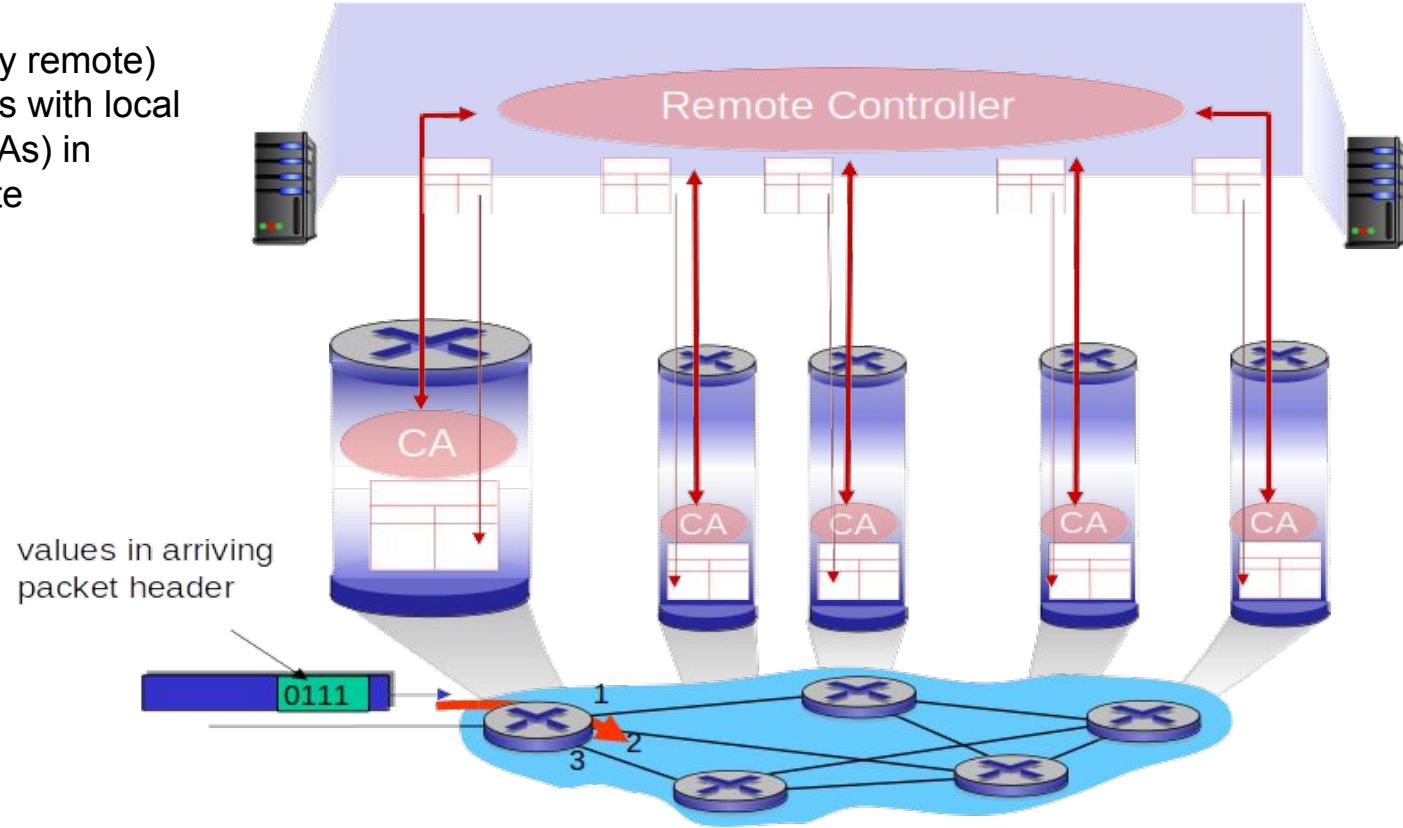
Per-router control plane

Individual routing algorithm components **in each and every router** interact with each other in control plane to compute forwarding tables



Software-Defined Networking (SDN) control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



Network service model

- Q: What **service model** for “channel” transporting datagrams from sender to receiver?
- **example services for individual datagrams:**
 - guaranteed delivery
 - guaranteed delivery with less than 40 msec delay
- **example services for a flow of datagrams:**
 - in-order datagram delivery
 - guaranteed minimum bandwidth to flow
 - security

Network-layer service models:

Network Architecture	Service Model	Guarantees ? Bandwidth	Guarantees ? Loss	Guarantees ? Order	Guarantees ? Timing	Congestion feedback
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

Network-layer service models:

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

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

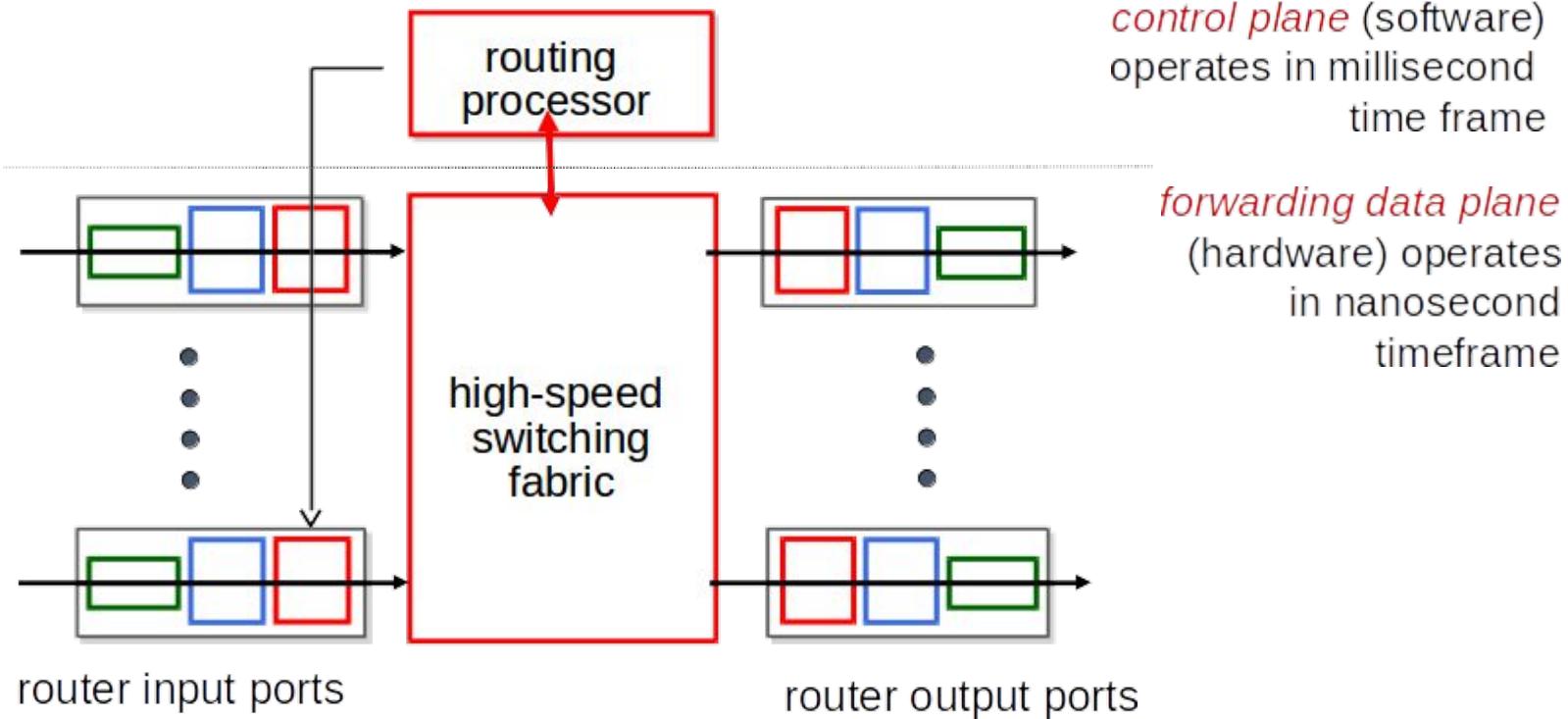
It's hard to argue with success of best-effort service model

outline

- Overview of Network layer
 - data plane
 - control plane
- **What's inside a router?**
 - input ports, switching, output ports, buffer management, scheduling
- IP: Internet Protocol
 - datagram format
 - IPv4 addressing
 - network address translation
 - IPv6
- Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow examples of match-plus-action in action

Router architecture overview

high-level view of generic router architecture:



Router architecture overview

- **Input ports**
 - Physical layer function of terminating an incoming physical link
 - Link-layer function needed to interoperate with the link layer at the other side of the incoming link
 - Lookup function
 - Consulting forwarding table to determine the router output port to which an arriving packet will be forwarded via the switching fabric
 - Control packets, e.g., carrying routing information) are forwarded to the routing processor
 - The number of ports could range from small number in enterprise routers, to hundreds of 10Gbps ports in a router at an ISP' edge

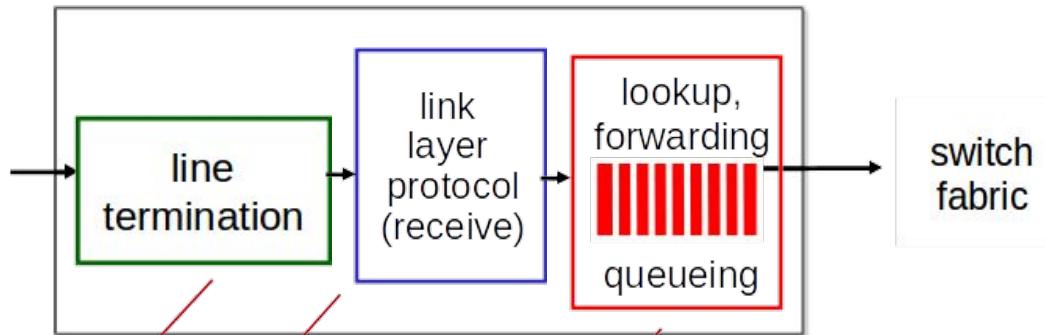
Router architecture overview

- **Switching fabrics**
 - Connects the router's input ports to its output ports
- **Output ports**
 - Stores packets received from the switching fabric and transmits these packets on the outgoing link
 - Performing the necessary link layer and physical layer functions
- **Routing processors**
 - Performs control-plane functions
 - Traditional routers: executes the routing protocols, maintains routing tables and attached link state information, computes the forwarding table for the router
 - SDN routers: responsible for communicating with the remote controller; receive forwarding table entries computed by the remote controller; install these entries in the router's input ports
 - Performs network management functions

How router parts are implemented?

- **Data plane: Input ports, output ports, and switching fabrics**
 - Always implemented in hardware. Why?
 - Example: 10Gbps input link and a 64-byte IP datagram
 - The input port has only 51.2 ns to process the datagram before another datagram may arrive
 - $64*8/10,000,000,000 = 51.2 \text{ ns}$
 - If N ports are combined on a line card, the datagram-processing pipeline must operate N times faster
- **Control functions:**
 - Implemented in software
 - Executed on the routing processor

Input port functions



physical layer:

bit-level reception

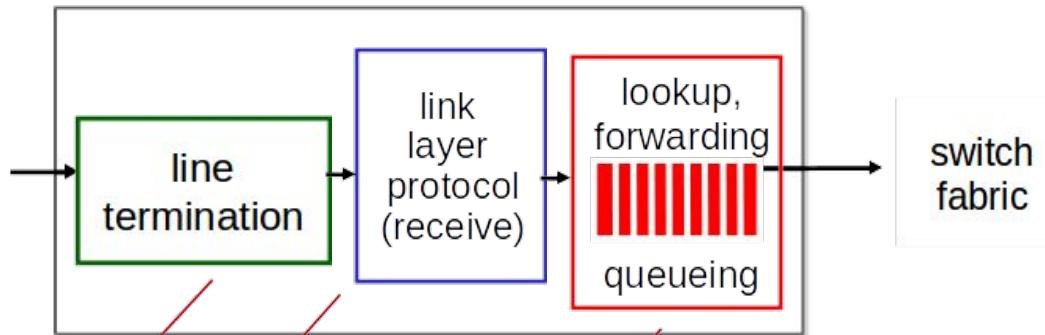
link layer:

e.g., Ethernet
(chapter 6)

decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory (“*match plus action*”)
- goal: complete input port processing at ‘line speed’
- **input port queuing:** if datagrams arrive faster than forwarding rate into switch fabric

Input port functions



physical layer:

bit-level reception

link layer:

e.g., Ethernet
(chapter 6)

decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory (“*match plus action*”)
- **destination-based forwarding**: forward based only on destination IP address (traditional)
- **generalized forwarding**: forward based on any set of header field values

Input port functions: Forwarding

- Destination-based forwarding
- Generalized forwarding

Input port function: Destination-based forwarding

- In the case of 32-bit IP addresses
 - A brute-force implementation of the forwarding table would have one entry for every possible destination address

<i>forwarding table</i>	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Input port function: Destination-based forwarding

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00010000 00000100 through 11001000 00010111 00010000 00000111	3
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Input port function: Destination-based forwarding

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

Destination Address Range	Link interface
11001000 00010111 00010**** *****	0
11001000 00010111 000110000 ****	1
11001000 00010111 00011**** *****	2
otherwise	3

- Examples:
 - DA: 11001000 00010111 00010110 10100001 which interface?
 - DA: 11001000 00010111 00011000 10101010 which interface?

Input port function: Destination-based forwarding

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

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

- Examples:
 - DA: 11001000 00010111 00010110 10100001 which interface?
 - DA: 11001000 00010111 00011000 10101010 which interface?

Input port function: Destination-based forwarding

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

Destination Address Range	Link interface
11001000 00010111 00010*****	0
11001000 00010111 000110000*****	1
11001000 00010111 00011*****	2
otherwise	3

- Examples:
 - DA: 11001000 00010111 00010110 10100001 which interface?
 - DA: 11001000 00010111 00011000 10101010 which interface?

Input port function: Destination-based forwarding

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

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

- Examples:
 - DA: 11001000 00010111 00010110 10100001 which interface?
 - DA: 11001000 00010111 00011000 10101010 which interface?

Input port function: Destination-based forwarding

- we'll see **why** longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
 - **content addressable**: present address to TCAM: retrieve output port in one clock cycle, regardless of table size
 - Cisco Catalyst: ~1M routing table entries in TCAM
- What is next?
 - Once the output port has been determined via the lookup, the packet can be sent into the switching fabric

Other input port functions

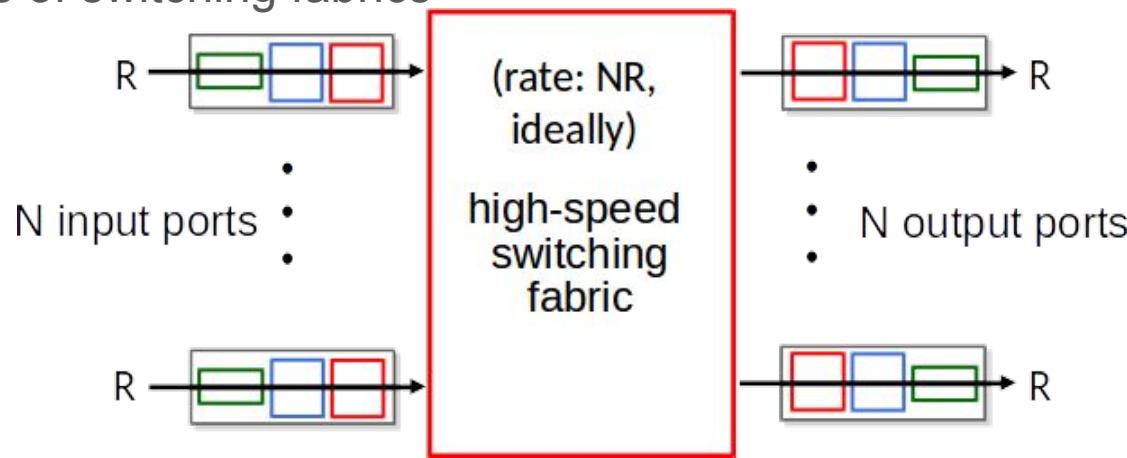
- Looking up a destination IP address (“match”) and then sending the packet into the switching fabric to the specified output port (“action”)
- the packet’s version number, checksum and time-to-live field must be checked and the latter two must be rewritten
- counters used for network management (such as the number of IP datagram received) must be updated

Other input port functions:

- Looking up a destination IP address (“match”) and then sending the packet into the switching fabric to the specified output port (“action”) is a special case of “Match plus action” abstraction in network devices
 - In link-layer switches, link-layer destination addresses are looked up
 - In firewalls (devices that filter out selected incoming packet), a packet may be dropped
 - NAT router

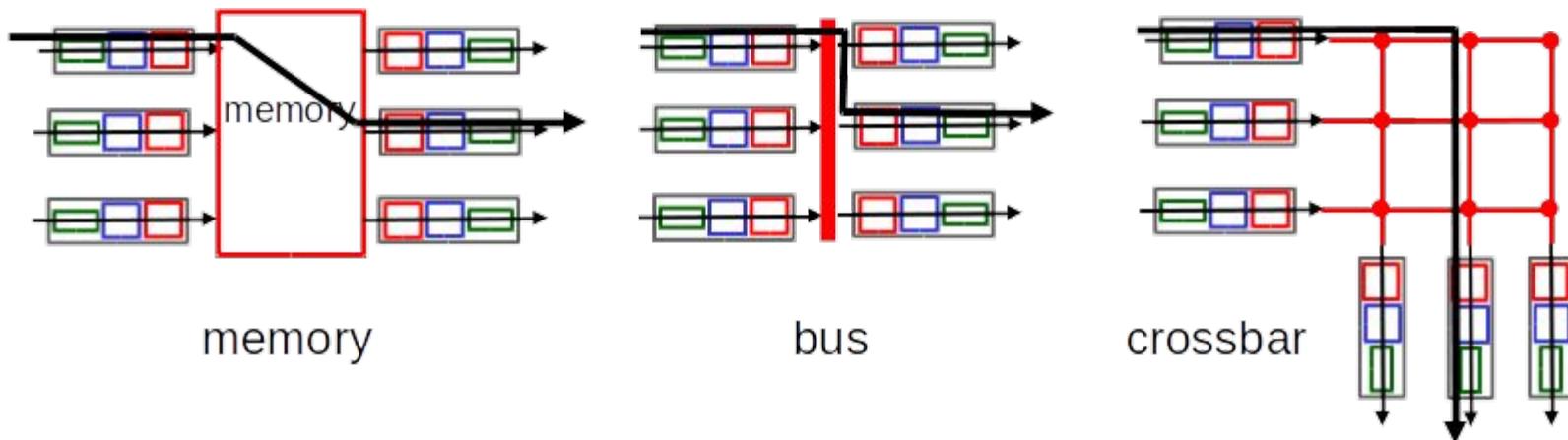
Switching fabrics

- transfer packet from input buffer to appropriate output buffer that is determined by longest prefix matching
- **switching rate:** rate at which packets can be transferred from inputs to outputs
 - often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate desirable: ideal scenario:
- three types of switching fabrics



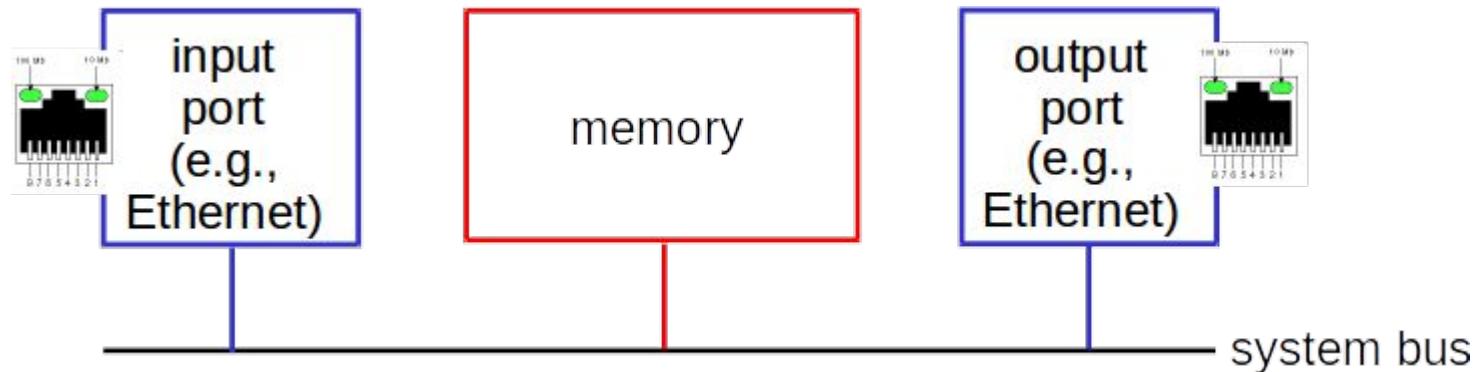
Switching fabrics

- transfer packet from input buffer to appropriate output buffer
- three types of switching fabrics



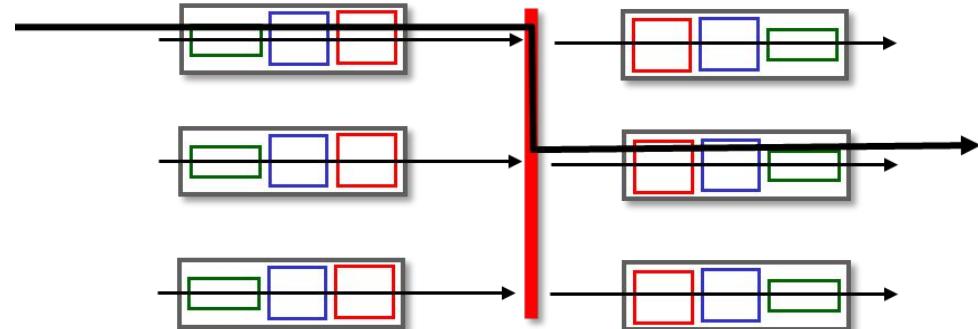
Switching via memory

- first generation routers:
 - traditional computers with switching done under direct control of CPU
 - packet copied to system's memory
 - speed limited by memory bandwidth (2 bus crossings per datagram)
 - Two packets cannot be forwarded at the same time, even if they have different destination port, since only one memory read/write can be done at a time over the shared bus



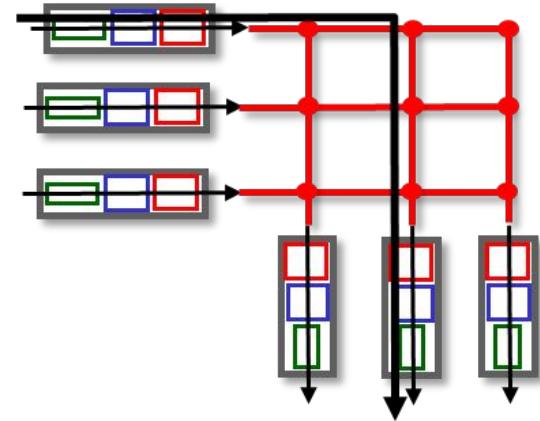
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
 - the input port attach a label containing the output port
 - All output ports receive the packet, but only the one that matches the label will keep that
- **bus contention:** switching speed limited by bus bandwidth
 - only one packet can cross the bus at a time
 - Blocking: packets destined for different output ports should wait
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers



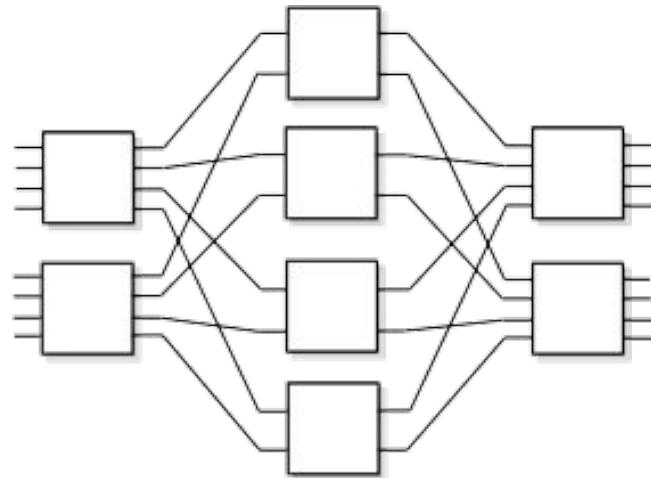
Switching via interconnection network

- overcome bus bandwidth limitations
- an interconnection network consisting of $2N$ buses
 - initially developed to connect processors in multiprocessor
- multistage switch:
 - $n \times n$ switch from multiple stages of smaller switches
- Non-blocking: packets destined to **different output port** need not to wait



Switching via interconnection network

- Made up by connecting smaller size switch elements serially and in parallel
 - Example: is made of four 4x4 and four 2x2 switches
- Capable of forwarding multiple packets in parallel since have parallel path from input side to output side
 - packets from different input port proceeds towards the **same output port** at the same time
 - fragment datagram into fixed length cells on entry, switch cells through the fabric, reassemble datagram at exit
 - Cisco 12000: switches 60 Gbps through the interconnection network



8x8 multistage switch
built from smaller-sized switches

Question

Suppose two packets arrive to two different input ports of a router at exactly the same time. Also suppose there are no other packets anywhere in the router

Suppose the two packets are to be forwarded to two different output ports. Is it possible to forward the two packets through the switch fabric at the same time when the fabric uses a shared bus / uses switching via memory / uses a crossbar?

Yes

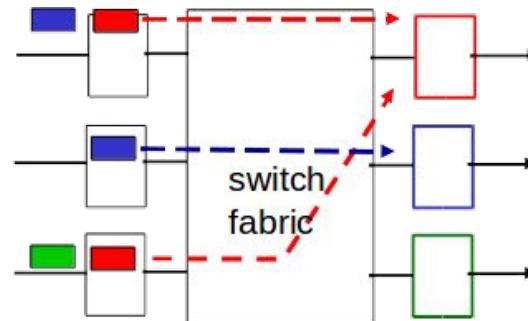
No

outline

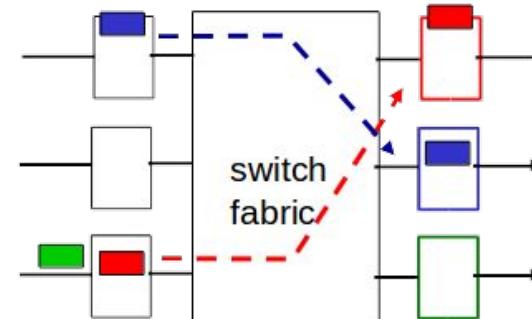
- Overview of Network layer
 - data plane
 - control plane
- **What's inside a router?**
 - input ports, switching, output ports, **buffer management, scheduling**
- IP: Internet Protocol
 - datagram format
 - Fragmentation
 - IPv4 addressing
 - network address translation
 - IPv6
- Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow examples of match-plus-action in action

Input port queuing

- If switch fabric slower than input ports combined -> queueing may occur at input queues
 - queueing delay and loss due to input buffer overflow!
- **Head-of-the-Line (HOL) blocking:** queued datagram at front of queue prevents others in queue from moving forward

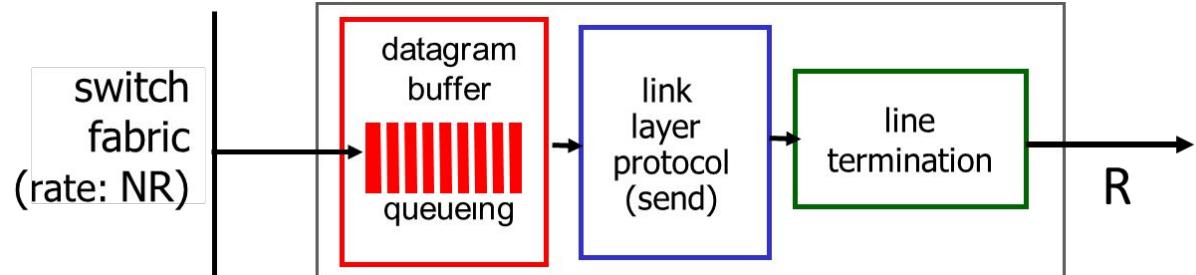


output port contention:
only one red datagram can be
transferred.
lower red packet is blocked



one packet time later:
green packet
experiences HOL
blocking

Output ports



- **buffering** required when datagrams arrive from fabric faster than the transmission rate
- Datagram (packets) can be lost due to congestion, lack of buffers
- **Drop policy:** which datagrams to drop if no free buffers?
- **scheduling discipline** chooses among queued datagrams for transmission
 - Priority scheduling – who gets best performance, network neutrality

How much buffering?

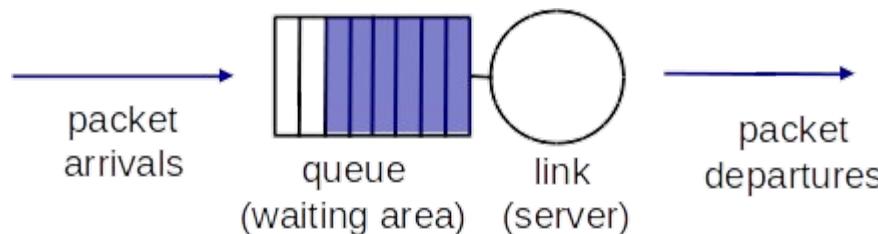
- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
 - e.g., C = 10 Gpbs link: 2.5 Gbit buffer
- recent recommendation: with N flows, buffering equal to

$$\frac{\text{RTT} \cdot C}{\sqrt{N}}$$

- but too much buffering can increase delays (particularly in home routers)
 - long RTTs: poor performance for realtime apps, sluggish TCP response
 - recall delay-based congestion control: “keep bottleneck link just full enough (busy) but no fuller”

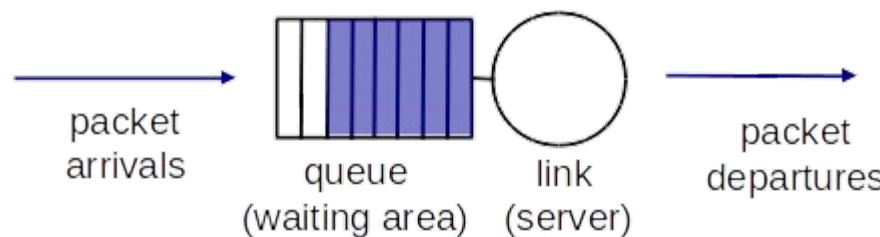
Buffer Management

- **Buffer Management**
 - **discard policy:** if packet arrives to full queue, which packet to discard?
 - **tail drop:** drop arriving packet
 - **priority:** drop/remove on priority basis
 - **random:** drop/remove randomly
 - **Marking:**
 - which packets to mark to signal congestion (ECN)



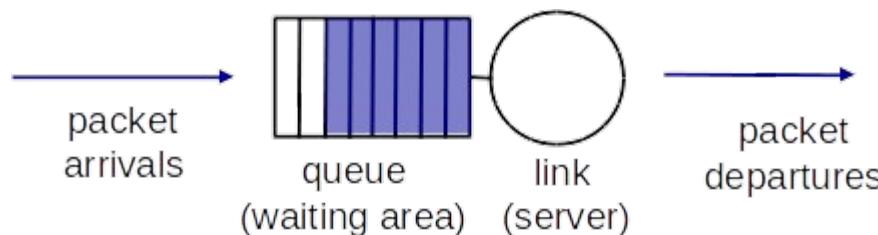
Scheduling policies

- **Packet scheduling:** deciding which packet to send next on link
 - first come, first served
 - priority
 - round robin
 - weighted fair queueing



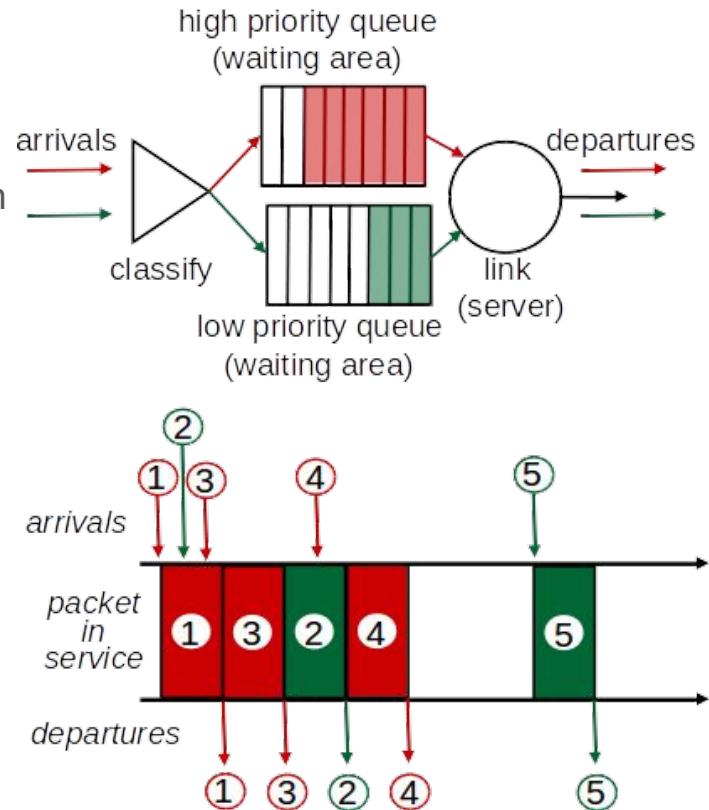
Scheduling policies: FCFS

- **FIFO (first in first out) scheduling:** send packets in order of arrival to output port
 - also known as: First-in-first-out (FIFO)
 - real-world example?
 - People in line in a supermarket



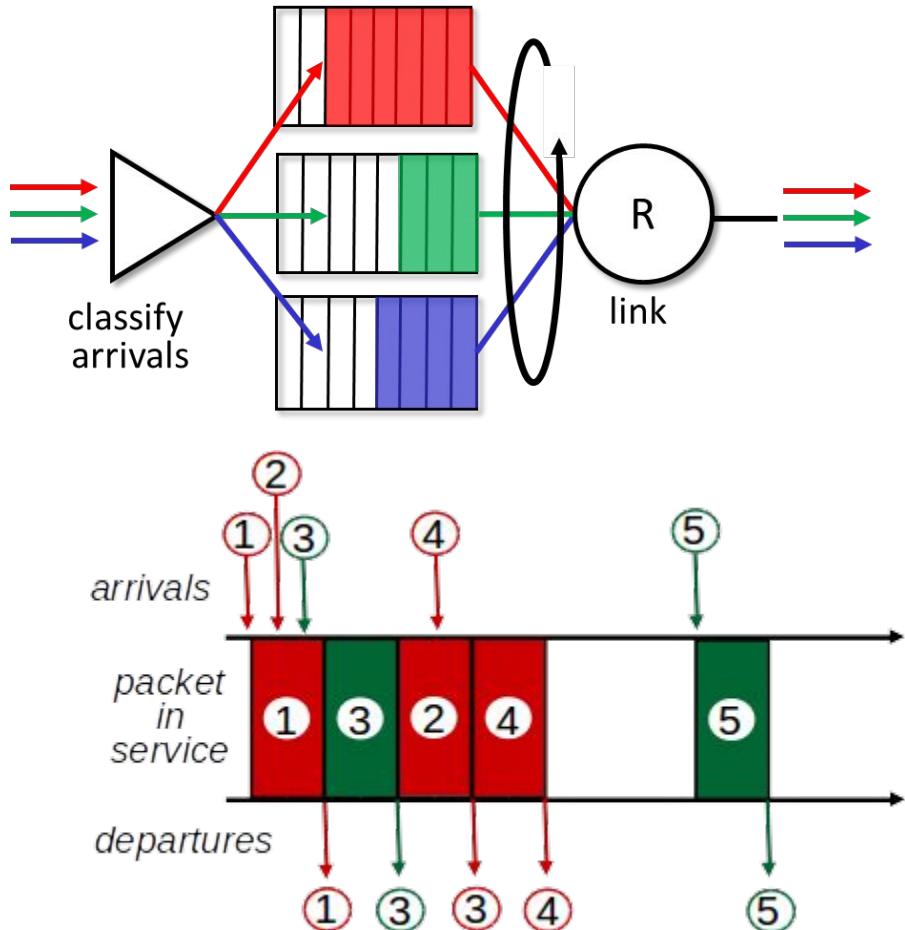
Scheduling policies: Priority

- priority scheduling
 - arriving traffic classified, queued by class
 - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
 - FCFS within a priority class
 - real world example?
 - Boarding at airports
- How priority classes are determined?
 - the network operator or ISP: Network management traffic / voice over ip, by looking at transport layer port number the router can figure out what type of traffic is carried by the packet..
 - Source or destination



Scheduling policies

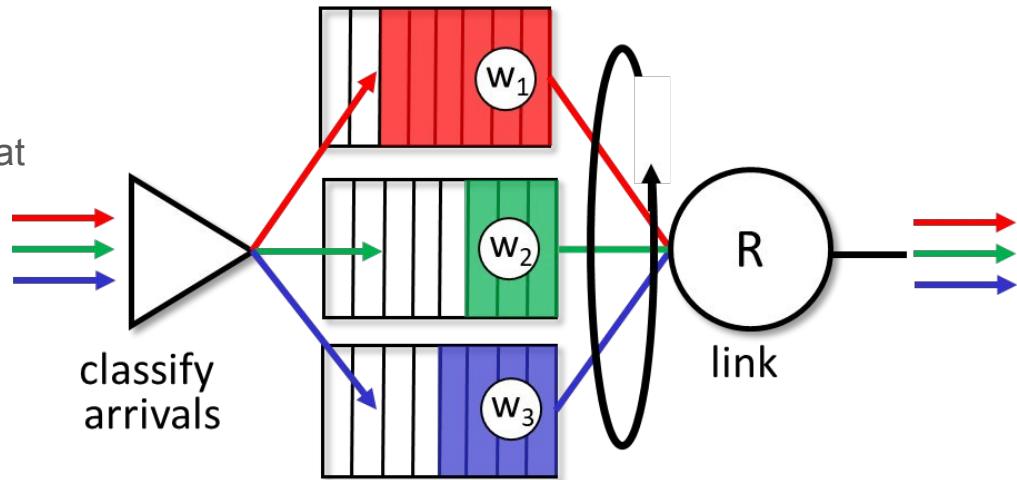
- Round Robin (RR) scheduling:
 - arriving traffic classified, queued by class
 - any header fields can be used for classification
 - multiple classes
 - cyclically scan class queues, sending one complete packet from each class (if available)
 - Example:
 - CPU assigned to processors



Scheduling policies

- Weighted Fair Queuing (WFQ):
 - generalized Round Robin
 - each class, i , has weight, w_i , and gets weighted amount of service in each cycle
 - minimum bandwidth guarantee (per-traffic-class)
 - For a link with transmission rate R , class i will achieve a throughput of at least:

$$R \times \frac{w_i}{\sum_j w_j}$$

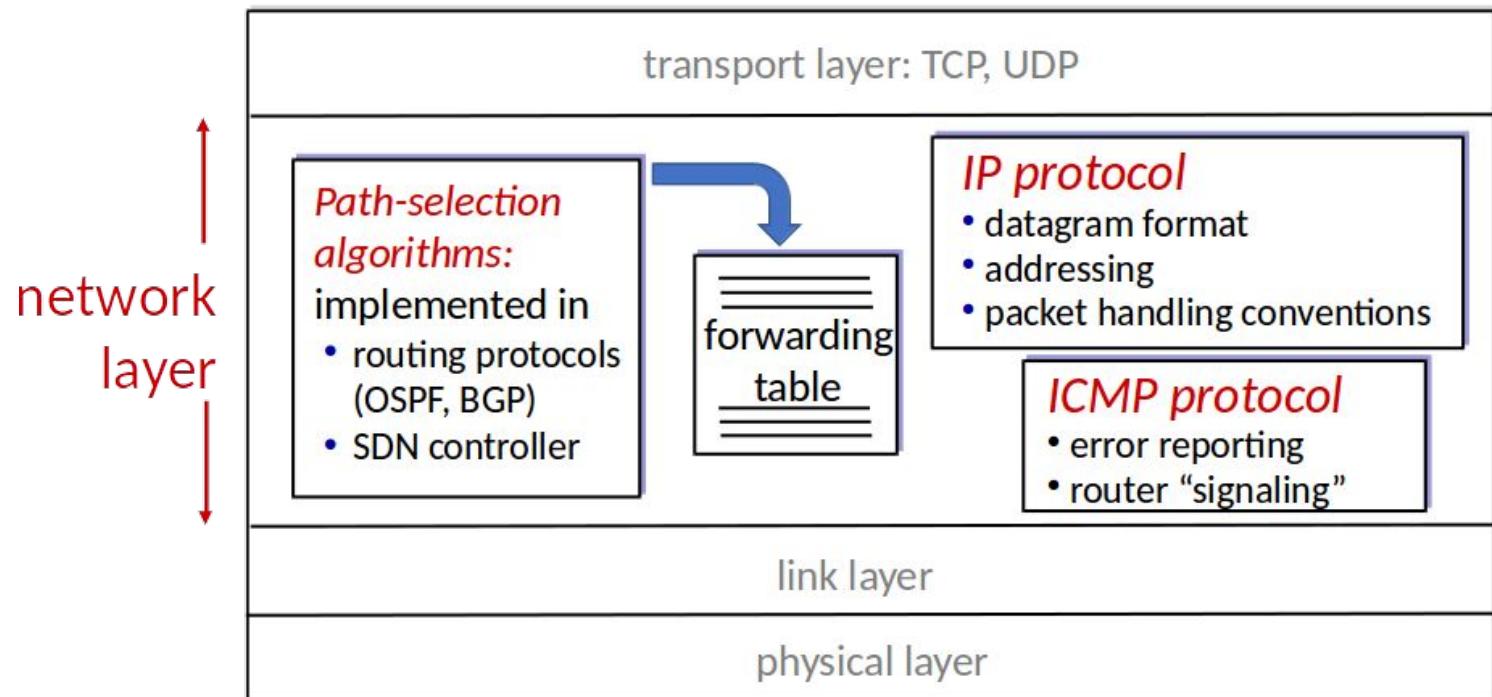


outline

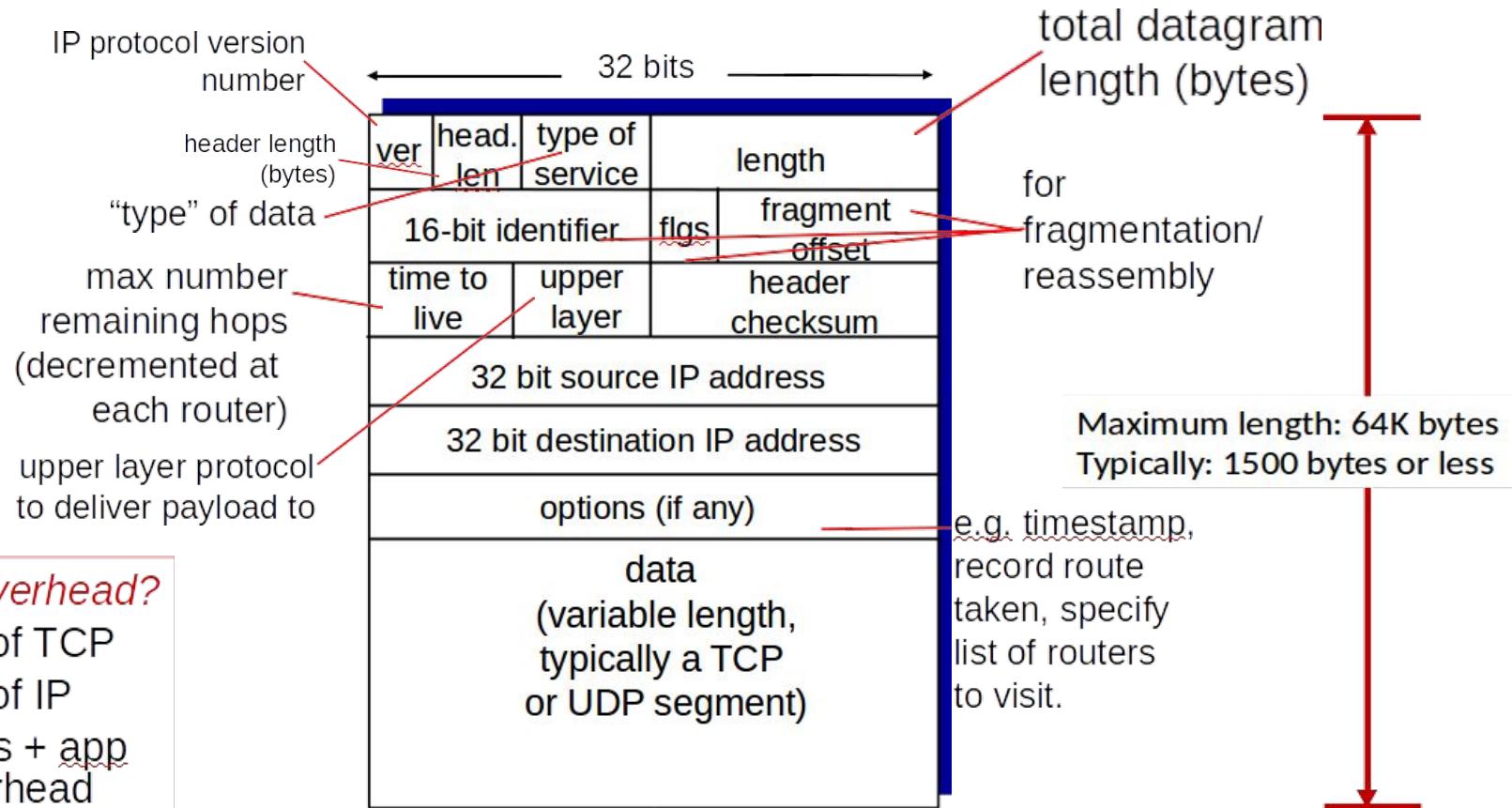
- Overview of Network layer
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - **datagram format**
 - IPv4 addressing
 - network address translation
 - IPv6
- Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow examples of match-plus-action in action

The Internet network layer

host, router network layer functions:



IP datagram format



how much overhead?

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

Question

suppose host A sends host B a TCP segment encapsulated in an IP datagram. When Host B receives the datagram, how does the network layer in Host B know it should pass the segment (that is, the payload of the datagram) to TCP rather than to UDP or to some other upper-layer protocol?

Answer:

- The 8-bit **protocol** field in the IP datagram contains information about which transport layer protocol the destination host should pass the segment to.

outline

- Overview of Network layer
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - datagram format
 - **IPv4 addressing**
 - network address translation
 - IPv6
- Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow examples of match-plus-action in action

outline

- Overview of Network layer
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - datagram format
 - **IPv4 addressing**
 - network address translation
 - IPv6
- Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow examples of match-plus-action in action

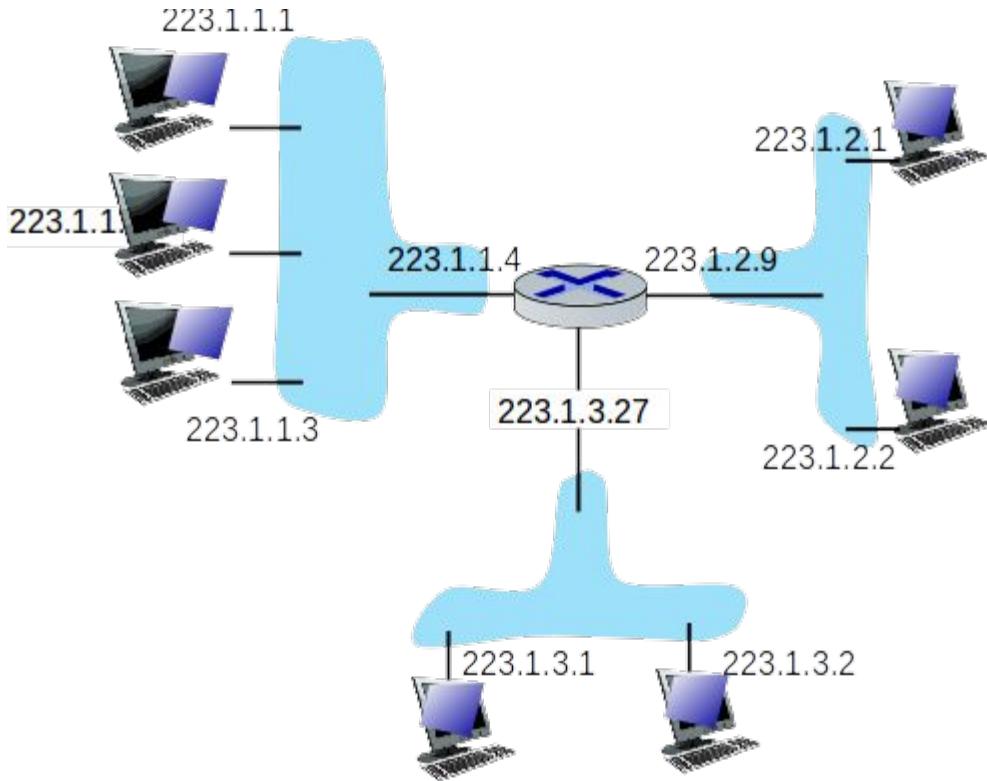
IP Addresses

- IPv4 uses 32-bit addresses
 - Later we'll see IPv6, which uses 128-bit addresses
- Written in “dotted decimal” notation
 - Four 8-bit numbers separated by dots



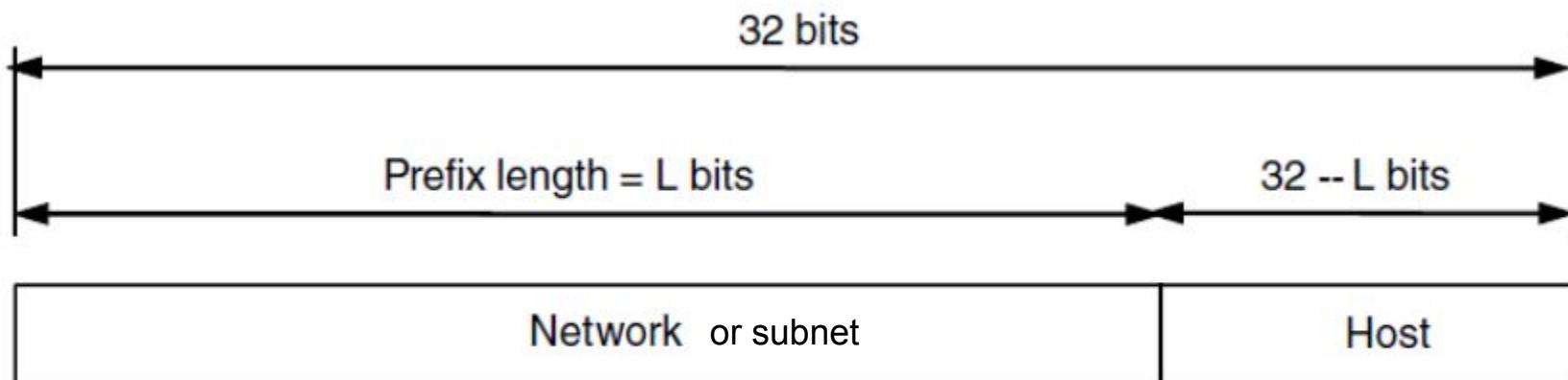
IP addressing

- **IP address:** associated with each **interface** in a host or router
- **interface:** connection between host/router and physical link
 - router's typically have multiple interfaces
 - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)



IP Addresses

- Addresses have two parts:
 - an L-bit prefix
 - There are 2^{32-L} addresses for an L bit prefix



IP Addresses

- Written in “**address/length**” notation
 - address** is lowest address in the prefix, **length** is prefix bits
 - Example: 128.13.0.0/16 is 128.13.0.0 to 128.13.255.255
 - So a /24 (“slash 24”) is 256 addresses, and a /32 is one address

000100100001111100000000xxxxxxx ↔

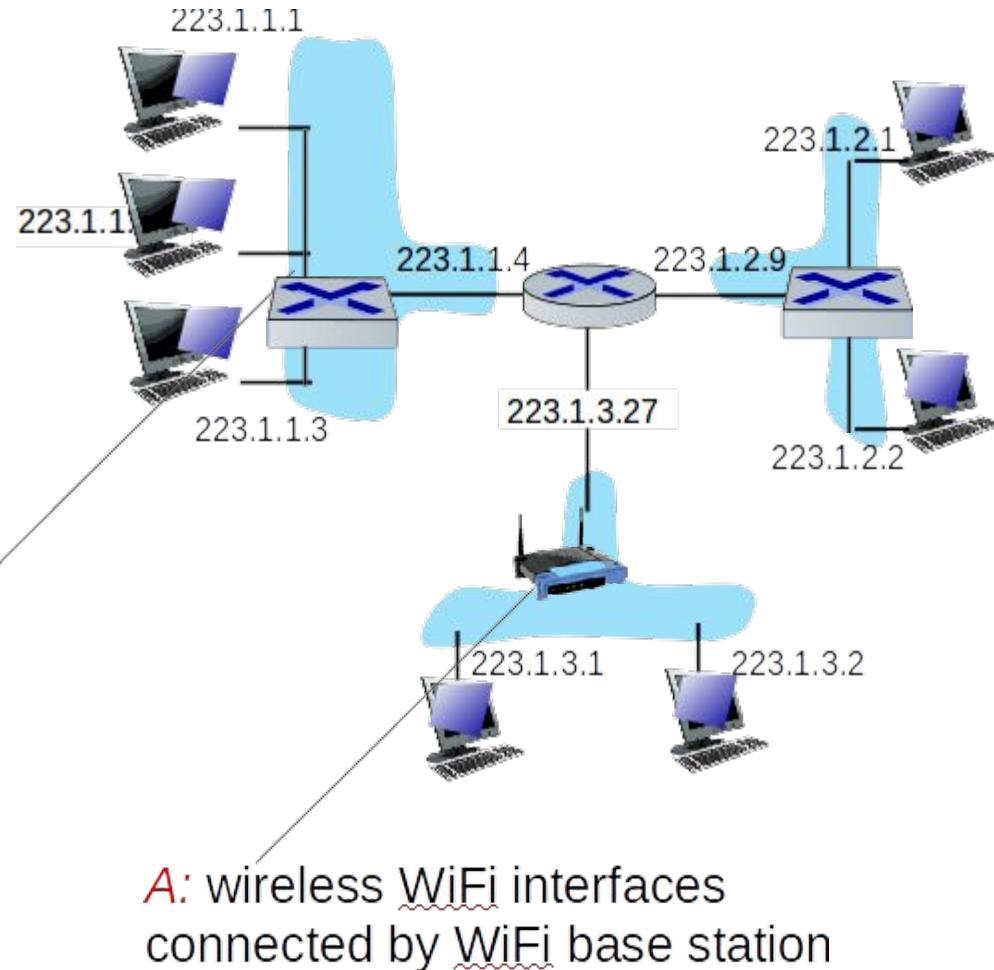
↔ 128.13.0.0/16

IP addressing

- **Q:** how are interfaces actually connected?
- **A:** we'll learn about that when studying link layer

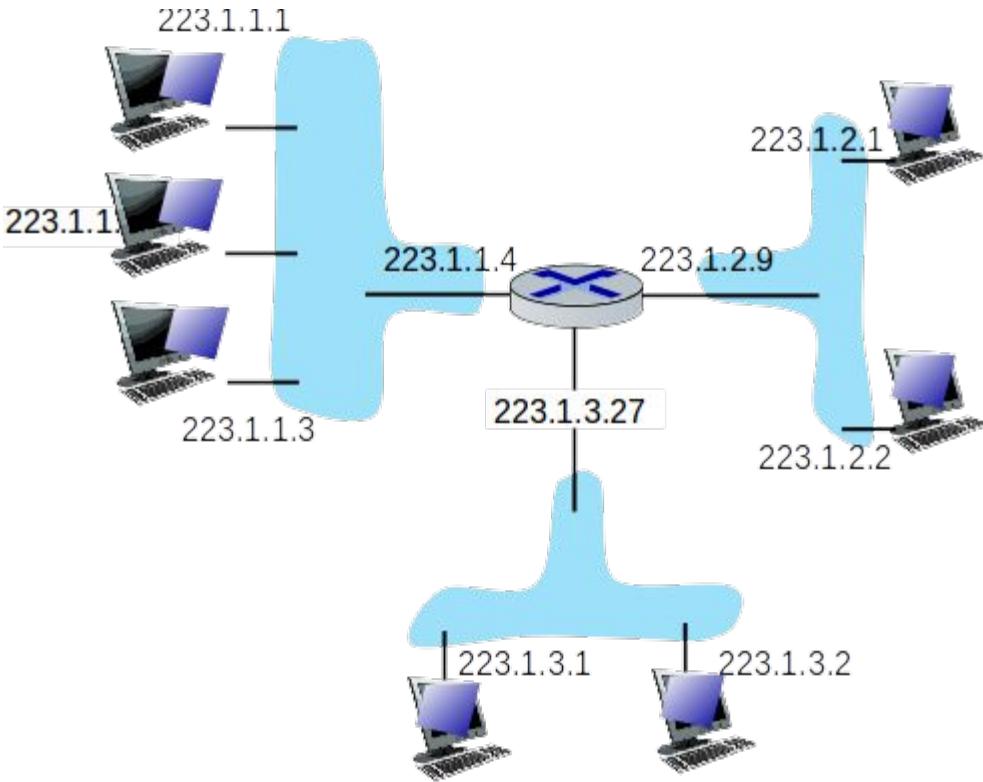
A: wired Ethernet interfaces connected by Ethernet switches

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



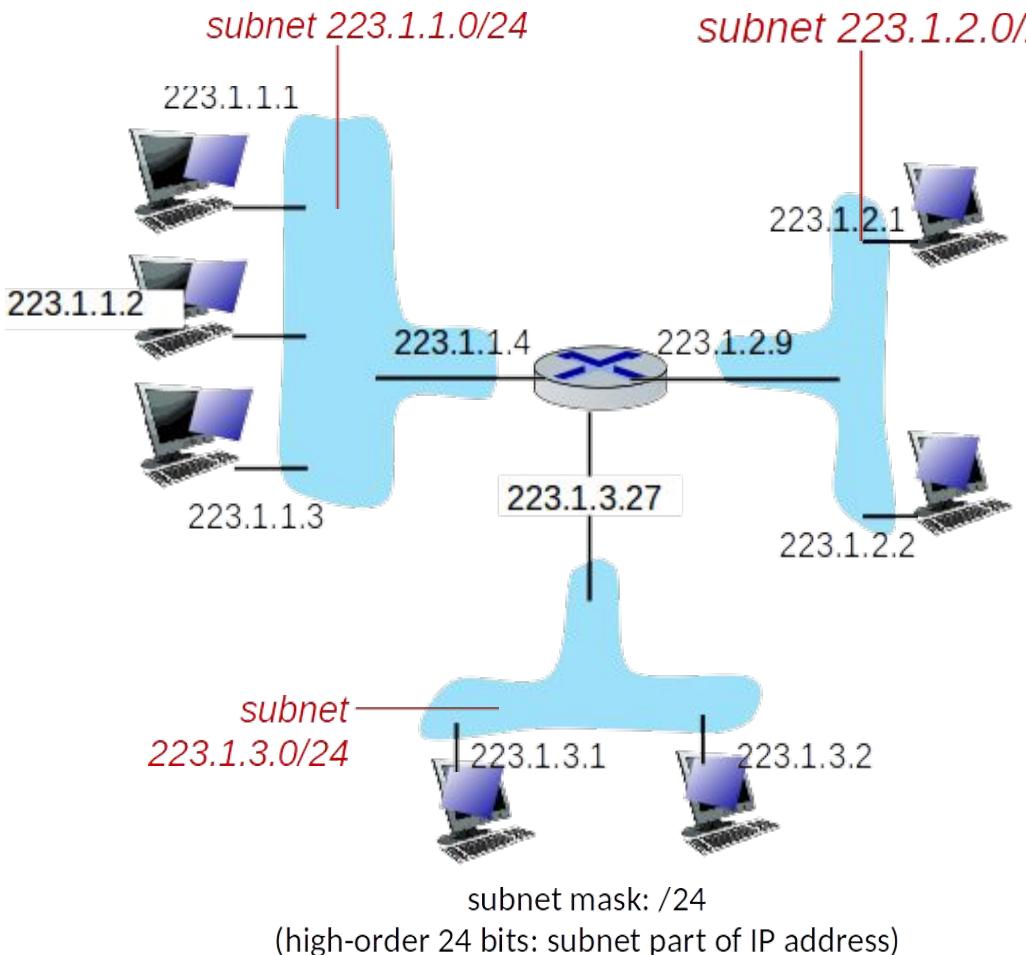
Subnets

- what's a subnet ?
 - device interfaces with same subnet part of IP address
 - can physically reach each other **without intervening router**
 - Subnet is an IP terminology
 - A subnet is simply a network in the Internet literature
- IP address:
 - **subnet part** - high order bits
 - Devices in same subnet have common high order bits
 - **host part** - low order bits



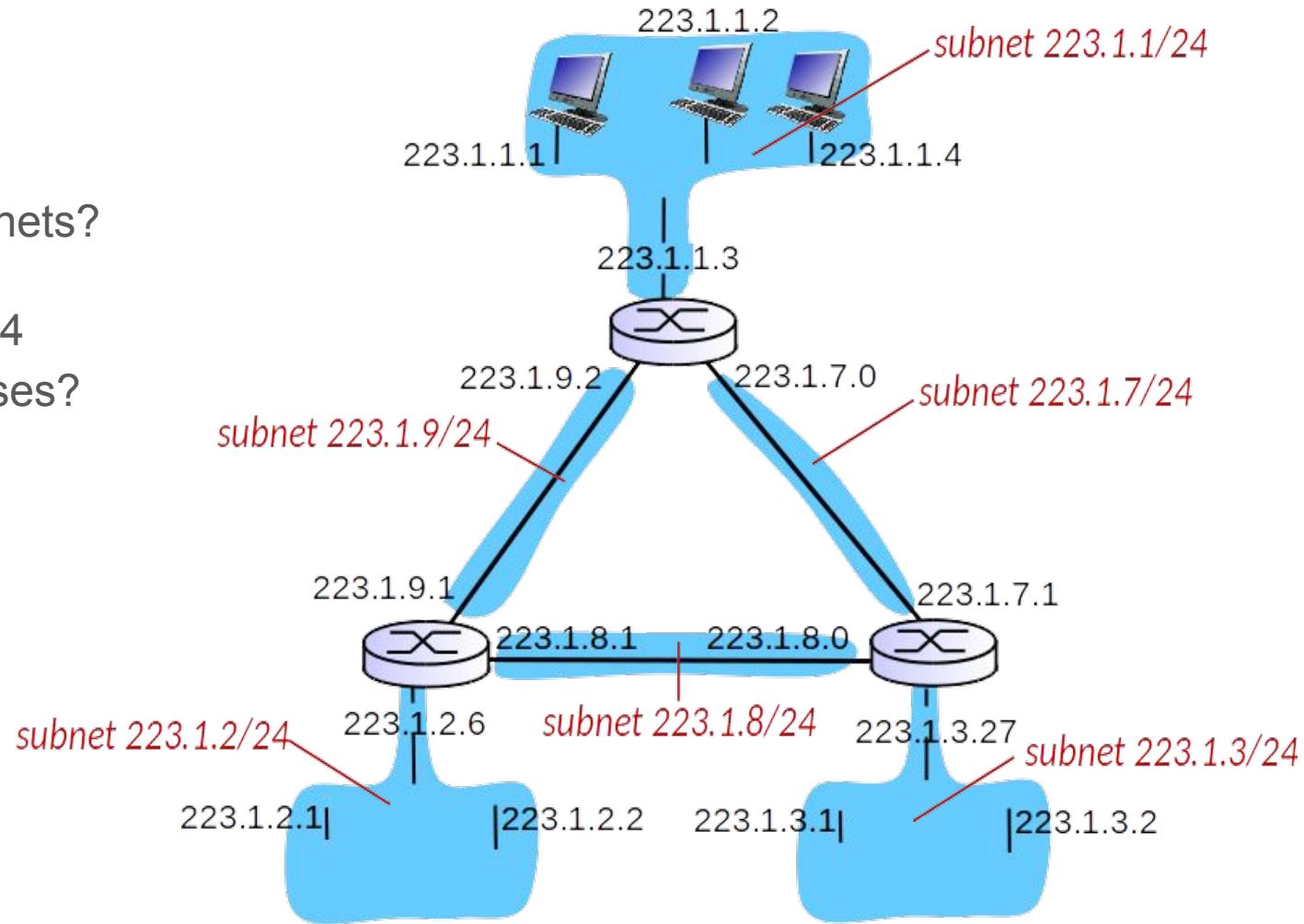
Subnets

- IP addressing assigns an address to the subnet
- Recipe for defining subnets
 - detach each interface from its host or router, creating islands of isolated networks
 - each isolated network is called a **subnet**



Question

- how many subnets?
 - 6
- what are the /24 subnet addresses?



Question

- Suppose there are three routers between a source host and a destination host. an IP datagram sent from the source host to the destination host will travel over how many interfaces?
 - 8
- How many forwarding tables will be indexed to move the datagram from source to destination
 - 3

IP Addressing: CIDR

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



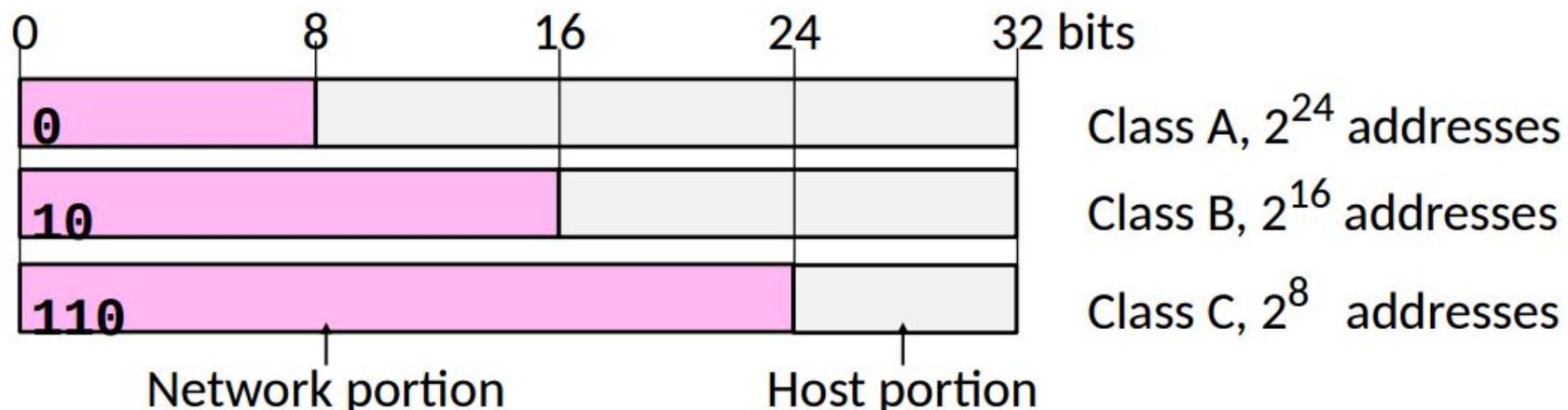
11001000 00010111 00010000 00000000

200.23.16.0/23

-
- How many addresses? $2^9 - 2$
- 2 is subtracted because one address is broadcast address, the other is subnet ID

Classful IP Addressing

- Originally, IP addresses came in fixed size blocks with the class/size encoded in the high-order bits
 - They still do, but the classes are now ignored



IP broadcast address

255.255.255.255

is delivered to all hosts on the same subnet

Routers may forward the message into neighboring subnets

IP addresses: how to get one?

That's actually three 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)
3. Q: How does an ISP get block of addresses?

IP addresses: how to get one?

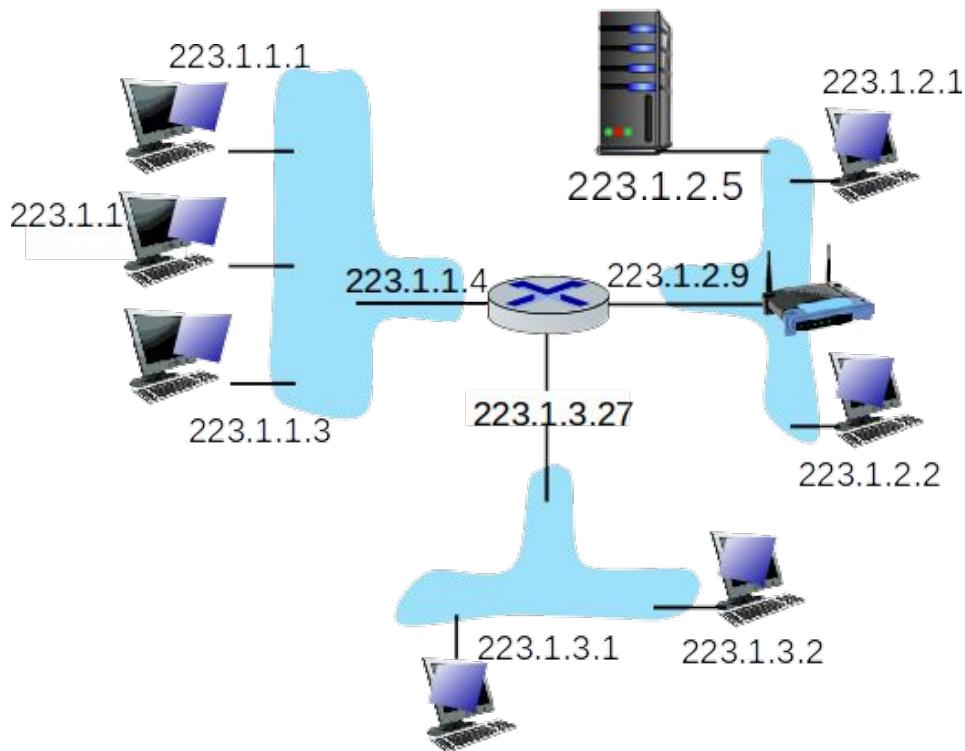
- **Q: How does a host get IP address?**
 - hard-coded by system admin in a file
 - Windows: control-panel->network->configuration->tcp/ip->properties
 - UNIX: /etc/rc.config
 - **DHCP: Dynamic Host Configuration Protocol:** dynamically get address from a server
 - “plug-and-play”

DHCP: Dynamic Host Configuration Protocol

- **goal:** allow host to dynamically obtain its 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 want to join network (more shortly)
- **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

DHCP client-server scenario

DHCP server

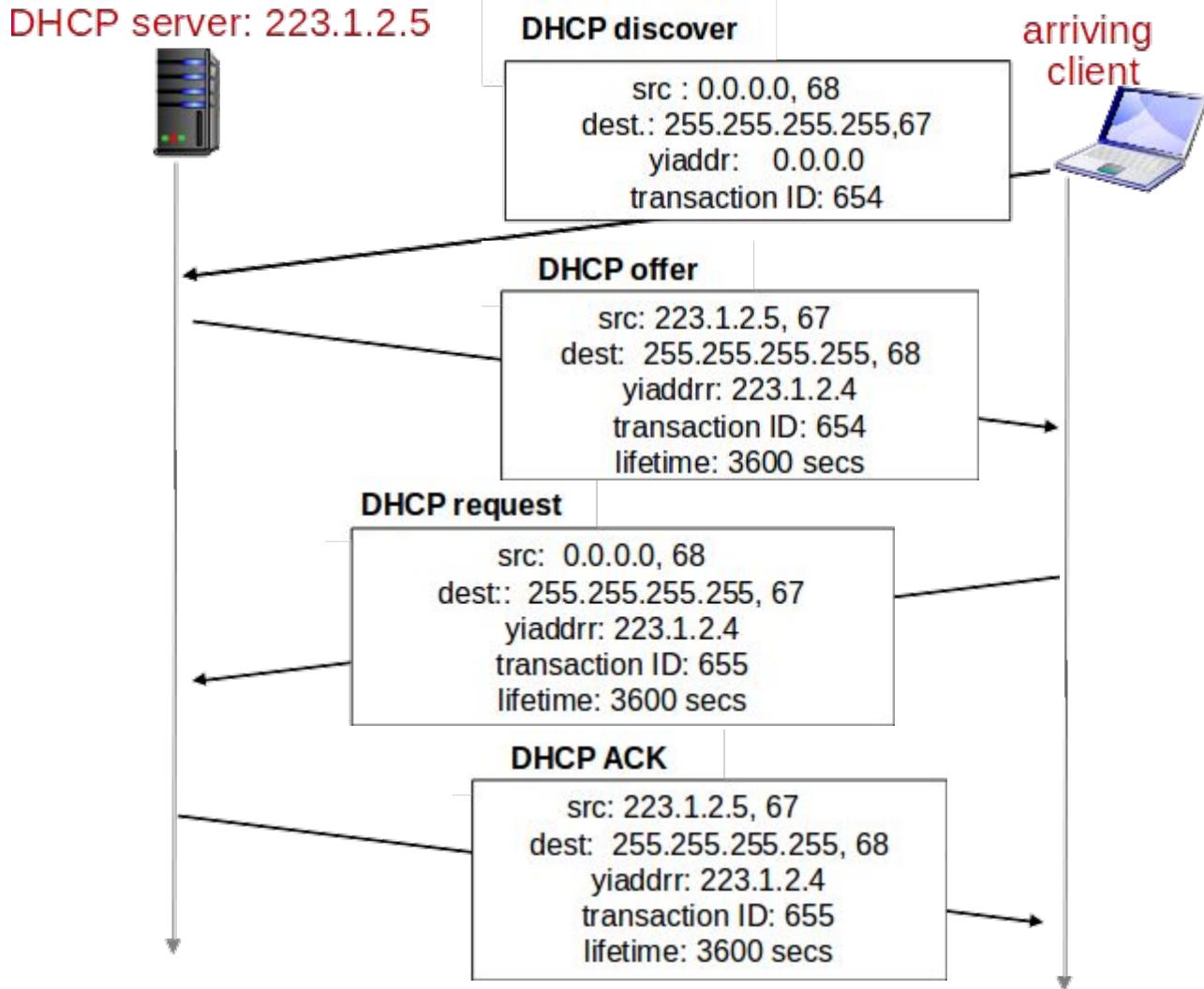


Typically, DHCP server will be co-located in router, serving all subnets to which router is attached



arriving **DHCP client** needs address in this network

DHCP client-server scenario

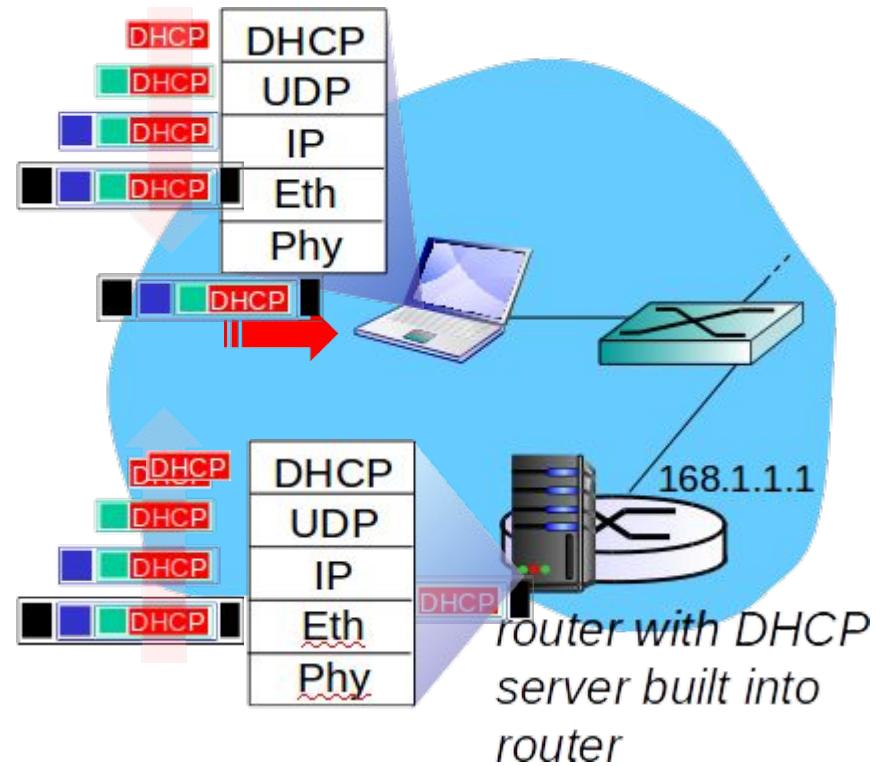


DHCP: more than IP addresses

- DHCP can return more than just allocated IP address on subnet:
 - address of first-hop router for client
 - name and IP address of DNS server
 - network mask (indicating network versus host portion of address)

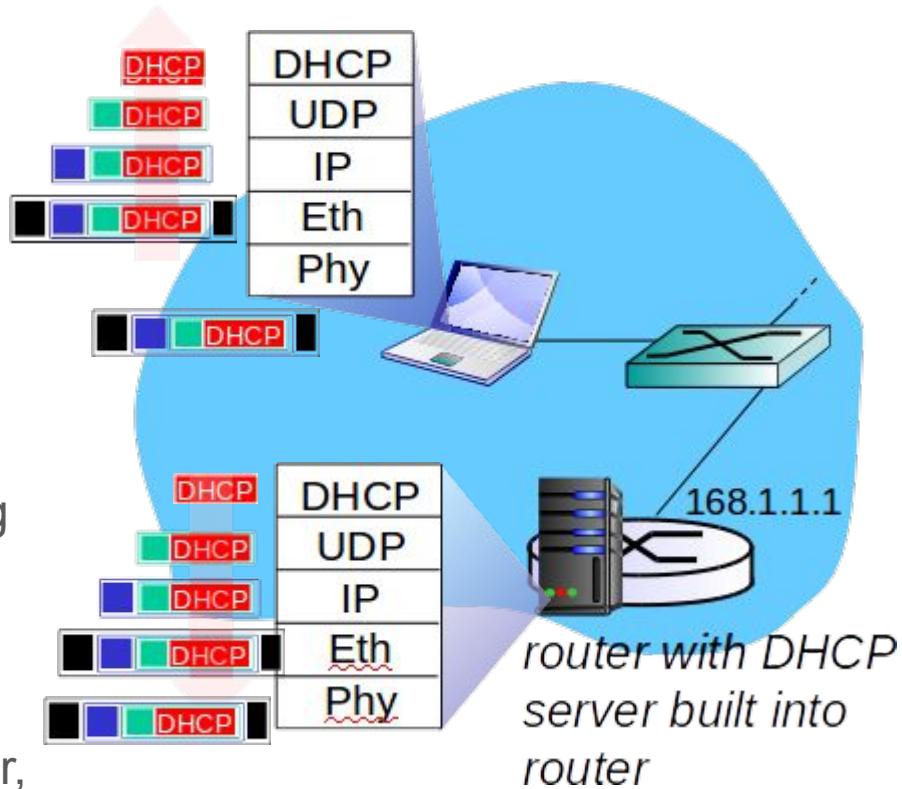
DHCP: example

- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP



DHCP: example

- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router



IP addresses: how to get one?

Q: how does *network* get subnet part of IP addr?

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

ISP's block 11001000 00010111 00010000 00000000 200.23.16.0/20

ISP can then allocate out its address space in 8 blocks:

Organization 0 11001000 00010111 00010000 00000000 200.23.16.0/23

Organization 1 11001000 00010111 00010010 00000000 200.23.18.0/23

Organization 2 11001000 00010111 00010100 00000000 200.23.20.0/23

...

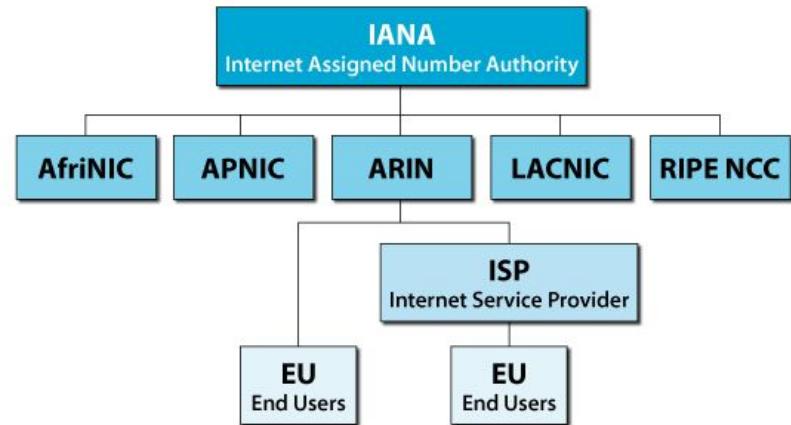
Organization 7 11001000 00010111 00011110 00000000 200.23.30.0/23

IP addresses: how to get one?

- **Q:** how does an ISP get block of addresses?
- **A:** ICANN: Internet Corporation for Assigned Names and Numbers

<http://www.icann.org/>

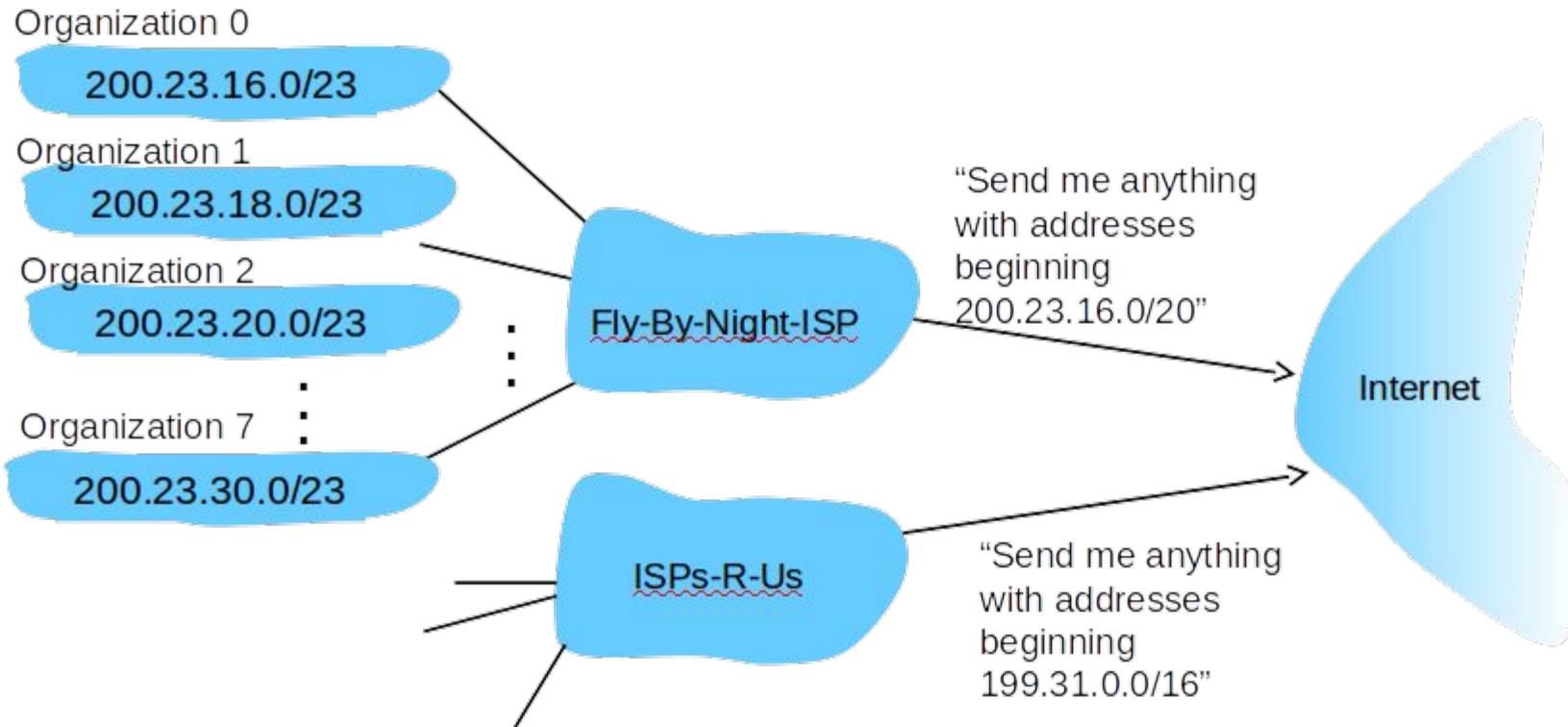
- allocates IP addresses, through **5 regional registries (RRs)** (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management
- assigns domain names, resolves disputes



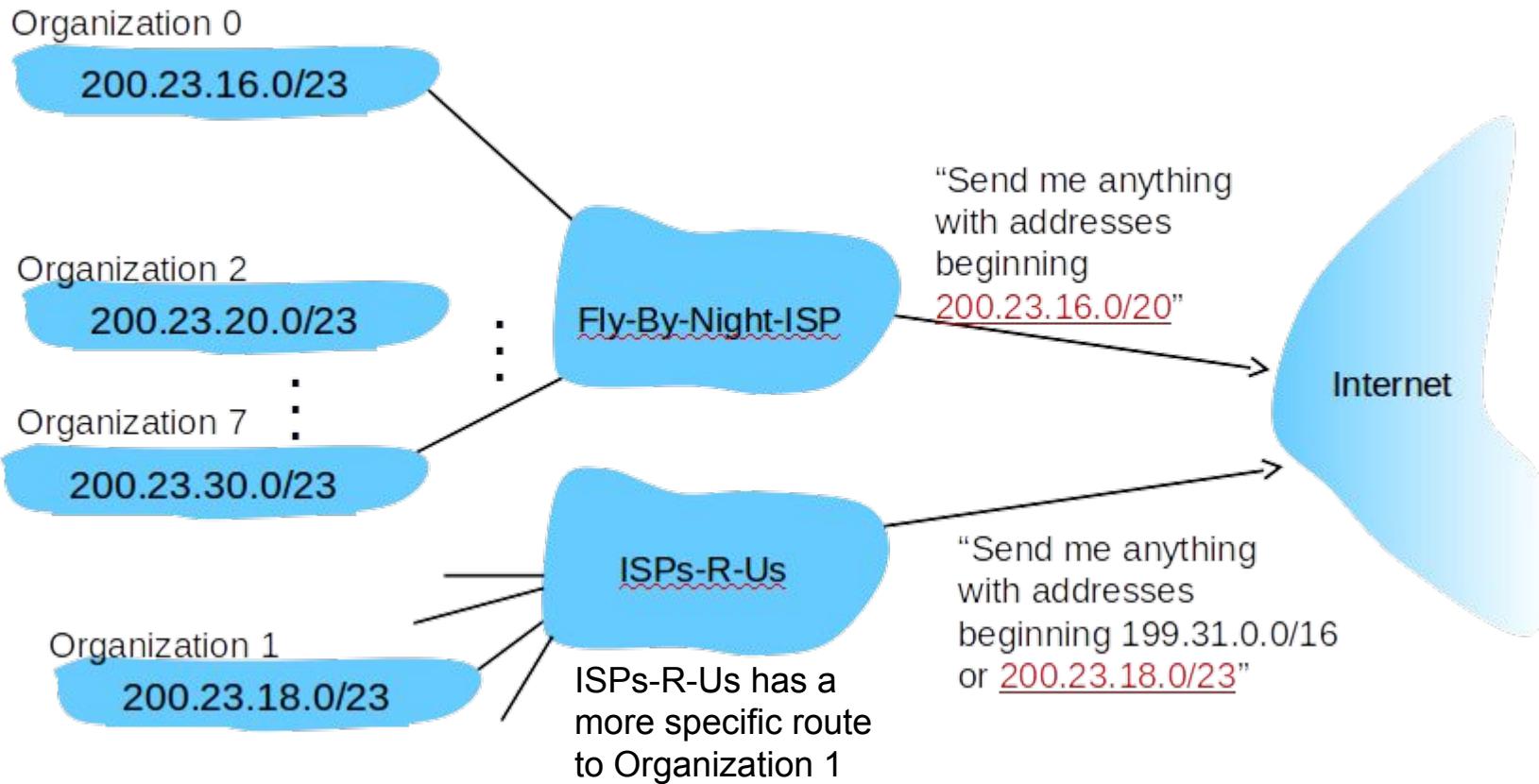
Hierarchical addressing: route aggregation

- hierarchical addressing allows efficient advertisement of routing Information:
- **Route aggregation/address aggregation/route summarization**
 - The ability to use a single prefix to advertise multiple networks
 - Scale routing by adjusting the size of IP prefixes
- Routers can change prefix lengths without affecting hosts

Hierarchical addressing: route aggregation



Hierarchical addressing: more specific routes



Question

consider a router that interconnects three subnets: subnet 1, subnet 2, and subnet 3. Suppose all of the interfaces in each of these three subnets are required to have the prefix 223.1.17/24. Also, suppose that Subnet 1 is required to support at least 60 interfaces, Subnet 2 is to support at least 90 interfaces, and Subnet 3 is to support at least 12 interfaces. Provide three network addresses (of the form a.b.c.d/x) that satisfy these requirements.

Answer: **subnet 1: 223.1.17.0/26**

Subnet 2: 223.1.17.128/25

Subnet 3: 223.1.17.192/28

IP addresses: Are there enough 32-bit addresses

- Q: are there enough 32-bit IP addresses?
 - ICANN allocated last chunk of IPv4 addresses to RRs in 2011
 - NAT helps IPv4 address space exhaustion
 - IPv6 has 128-bit address space
 - https://www.youtube.com/watch?v=Jq9b89MseAU&ab_channel=chriscxd

outline

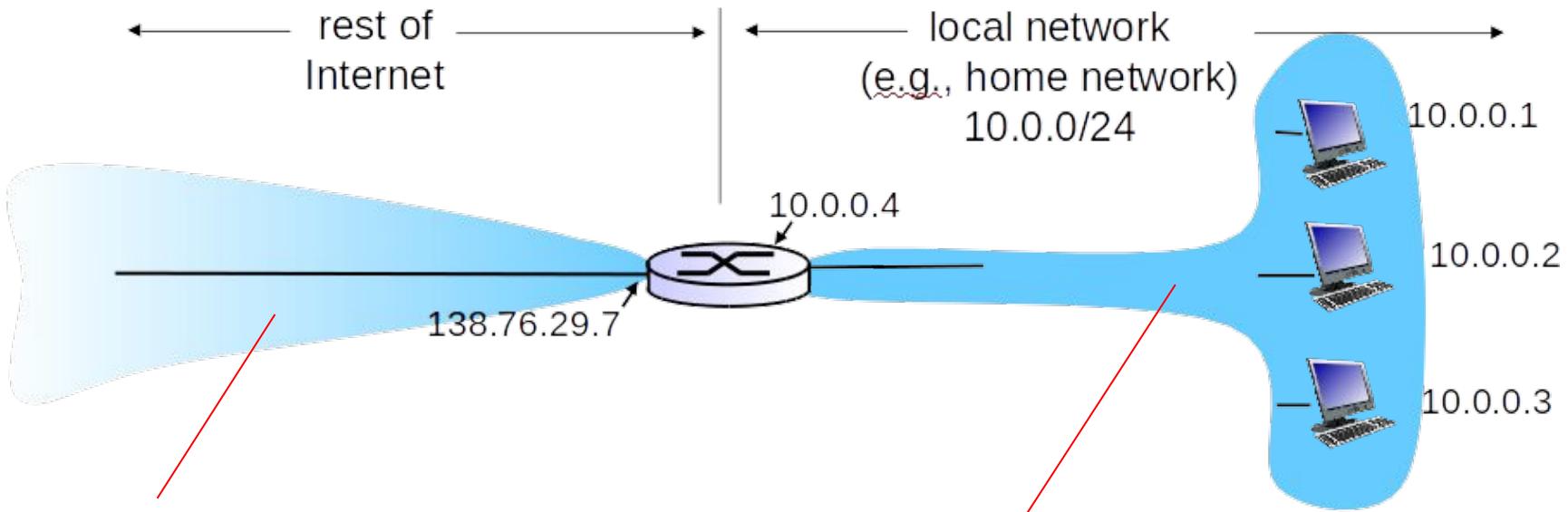
- Overview of Network layer
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - datagram format
 - Fragmentation
 - IPv4 addressing
 - **network address translation**
 - IPv6
- Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow examples of match-plus-action in action

NAT: Network Address Translation

- Problem: Internet Growth
 - Many billions of Hosts
 - And we're using 32-bit addresses!
- Solution:
 - Basic idea: Map many “Private” IP addresses to one “Public” IP.
 - Allocate IPs for private use (192.168.x, 10.x)

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** single source NAT IP
address: example: 138.76.29.7,different
source port numbers

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

NAT: network address translation: advantages

- all devices in local network have 32-bit addresses in a “private” IP address space (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 prefixes) that can only be used in local network
- Advantages:
 - just one IP address needed from provider ISP for all devices
 - can change addresses of host in local network without notifying outside world
 - can change ISP without changing addresses of devices in local network
 - security: devices inside local net not directly addressable, visible by outside world

Question

- If all datagrams arriving at the NAT router have the same destination IP address, how does the router know the internal host to which it should forward a given datagram?
 - By using a NAT translation table at the NAT router
 - Include port numbers as well as IP addresses in the table entries

NAT: network address translation

implementation: NAT router must (transparently):

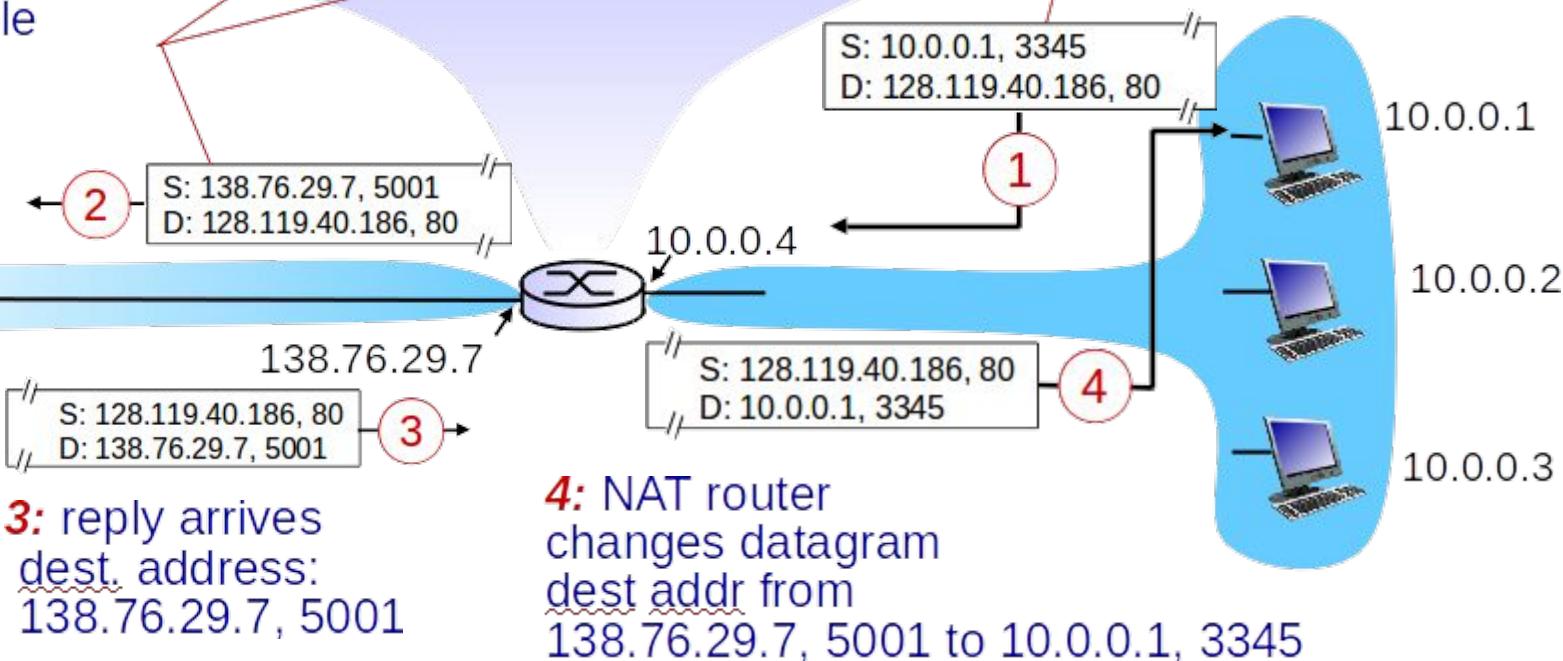
- **outgoing datagrams: replace** (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
 - remote clients/servers will respond using (NAT IP address, new port #) as destination address
- **remember (in NAT translation table)** every (source IP address, port #) to (NAT IP address, new port #) translation pair
- **incoming datagrams: replace** (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

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

NAT translation table

WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
.....

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80



NAT: network address translation

- 16-bit port-number field:
 - 60,000 simultaneous connections with a single LAN-side address!
- NAT in action
 - Ifconfig
 - What is my IP?

NAT: network address translation

- NAT has been controversial:
 - routers should only process up to layer 3
 - address “shortage” should be solved by IPv6
 - violates end-to-end argument (port # manipulation by network-layer device)
 - NAT traversal: what if client wants to connect to server behind NAT?
 - NAT traversal tools: RFC 5389
- but NAT is here to stay:
 - extensively used in home and institutional nets, 4G/5G cellular nets

outline

- Overview of Network layer
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - datagram format
 - Fragmentation
 - IPv4 addressing
 - network address translation
 - **IPv6**
- Generalized Forward and SDN
 - Match
 - Action
 - OpenFlow examples of match-plus-action in action

IPv6: motivation

- **initial motivation:**
 - Problem: Internet growth
 - Many billions of hosts and we're using 32-bit addresses
 - Need for a larger IP address space
 - 32-bit IPv4 address space would be completely allocated
- **additional motivation:**
 - header format helps speed processing/forwarding
 - header changes to facilitate QoS
 - Flow label is added in the header field
 - enable different network-layer treatment of “flows”

IPV6 address format

- Features large addresses
 - 128 bits, most of header
- New notation
 - 8 groups of 4 hex digits (16 bits)
 - Omit leading zeros, groups of zeros

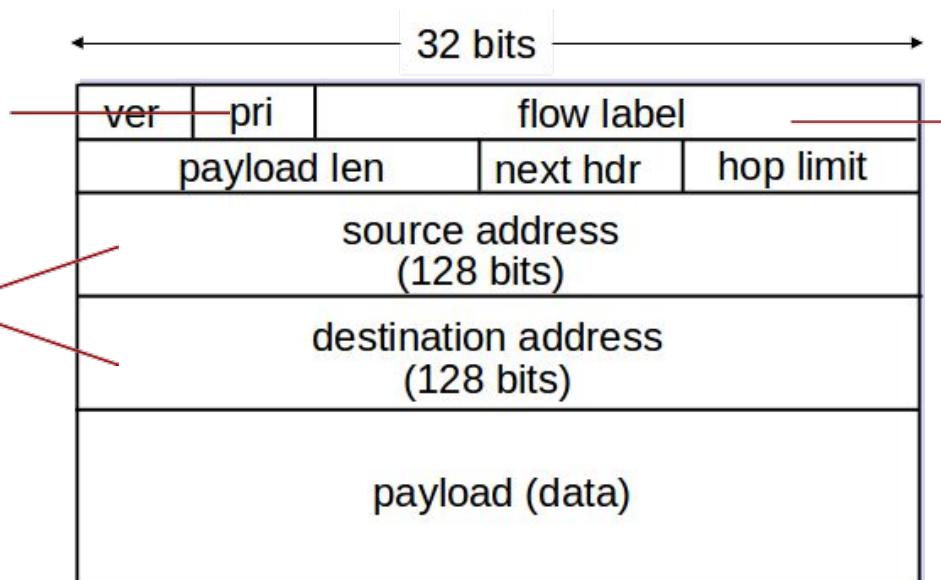
Example: **2001:0db8:0000:0000:0000:ff00:0042:8329**

2001:db8::ff00:42:8329

IPv6 datagram format

priority: identify priority among datagrams in flow

128-bit IPv6 addresses



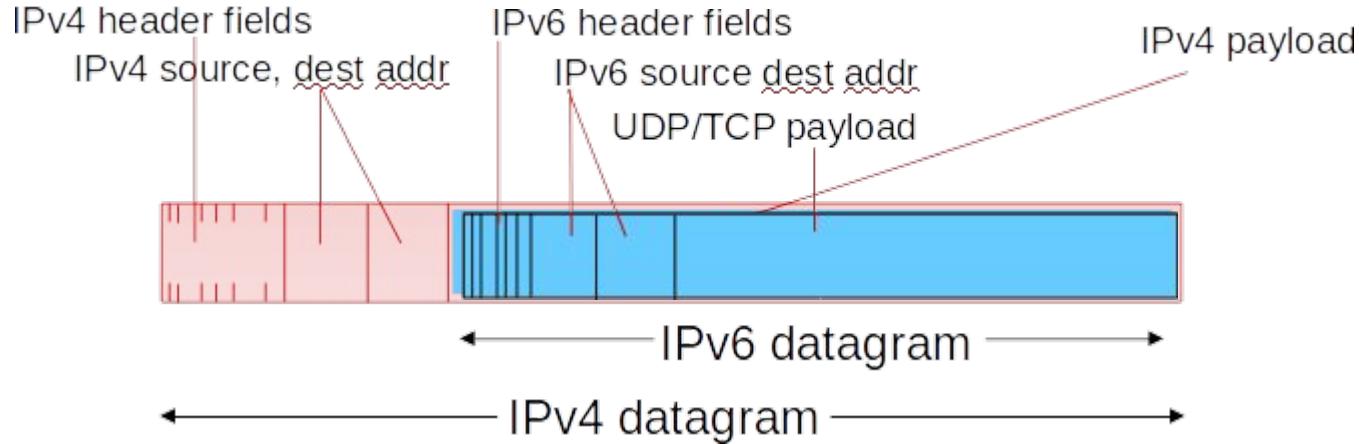
flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

IPv6 datagram format: changes from IPv4

- **No checksum:** to speed processing at routers
 - fast processing of IP packets was a main concern
- **No Options:**
 - Still allowed, by “Next Header” field pointing to an option field
- **No fragmentation/reassembly at intermediate routers**
 - only by source and destination
 - What if a router receive a large IPv6 datagram
 - drops the datagram
 - Sends an ICMP error message (“Packet too big”) back to the sender
 - result: speed up IP forwarding within the network

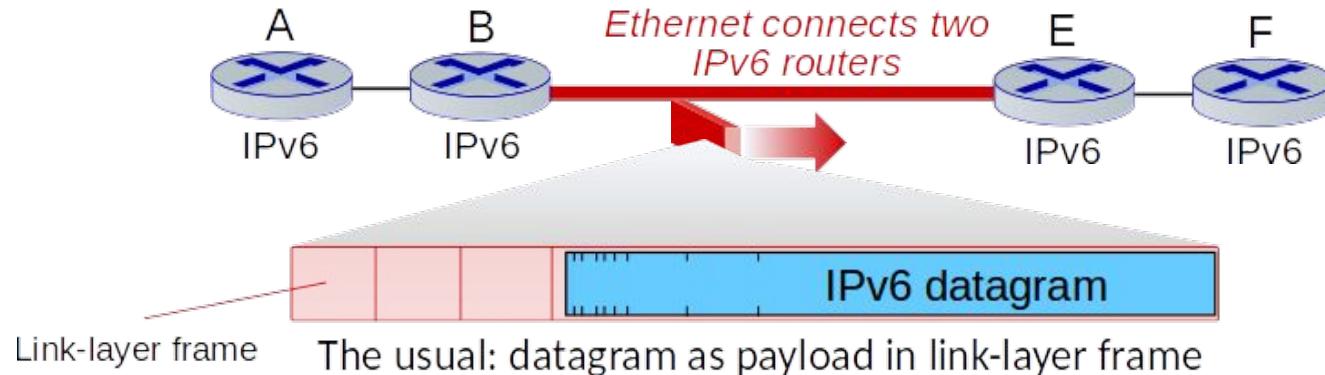
IPV6 Transition

- How to deploy IPv6?
 - not all routers can be upgraded simultaneously
 - no “flag days”
 - how will network operate with mixed IPv4 and IPv6 routers?
- **tunneling:** IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers
 - tunneling used extensively in other contexts (4G/5G)

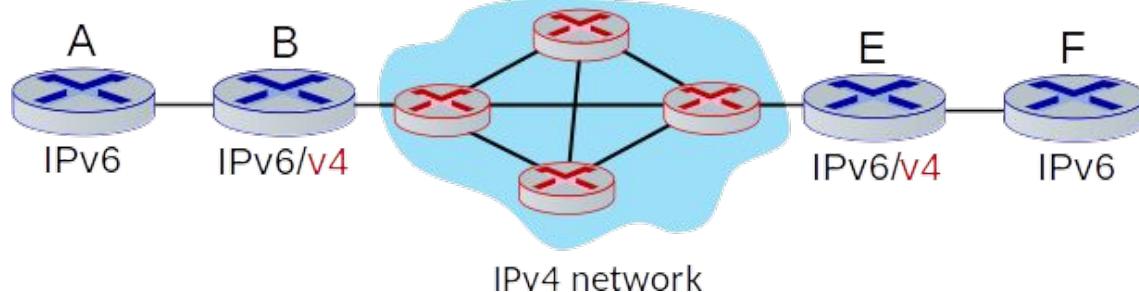


Tunneling and encapsulation

Ethernet connecting
two IPv6 routers:

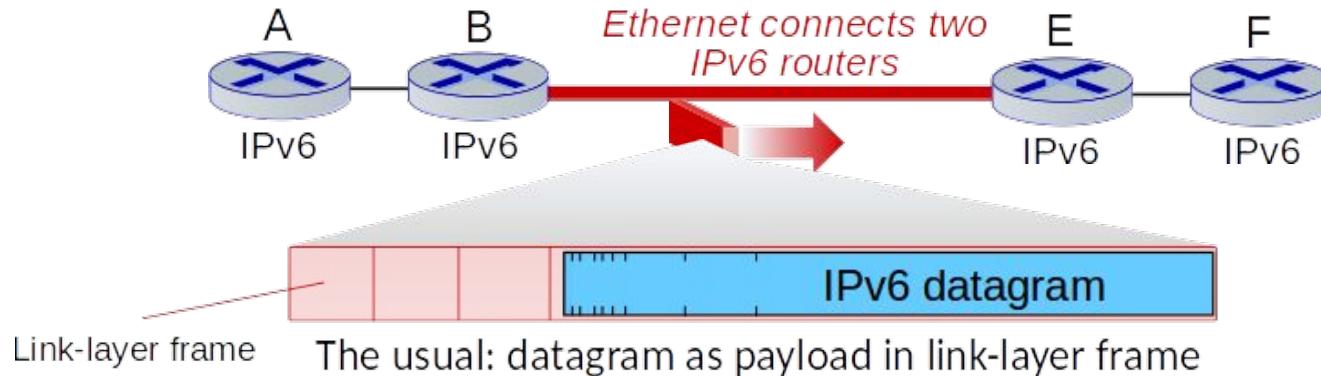


IPv4 network
connecting two
IPv6 routers

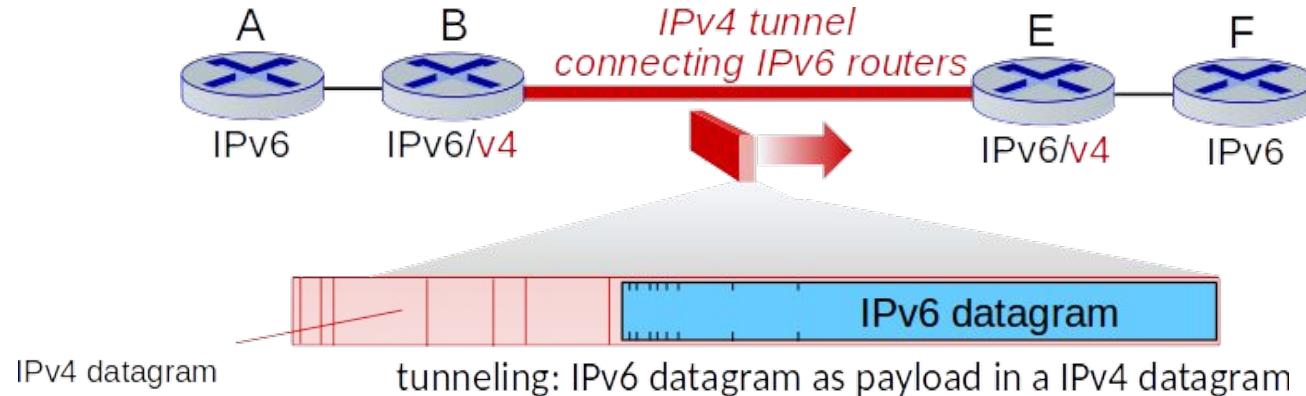


Tunneling and encapsulation

Ethernet connecting
two IPv6 routers:

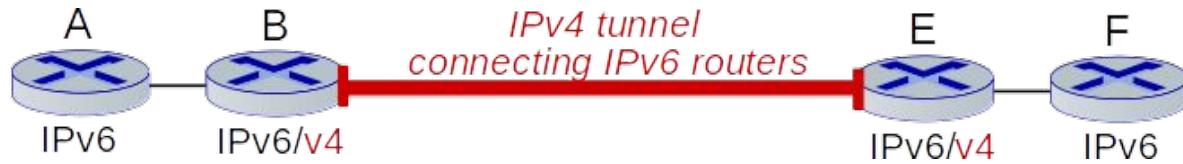


IPv4 network
connecting two
IPv6 routers

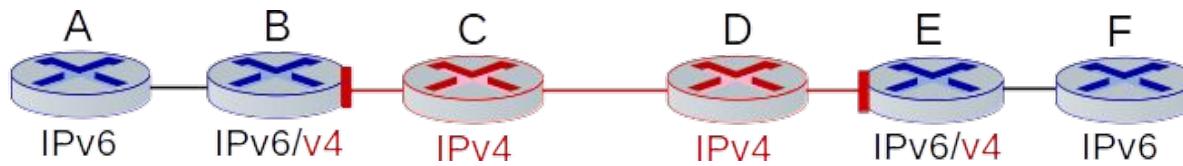


Tunneling

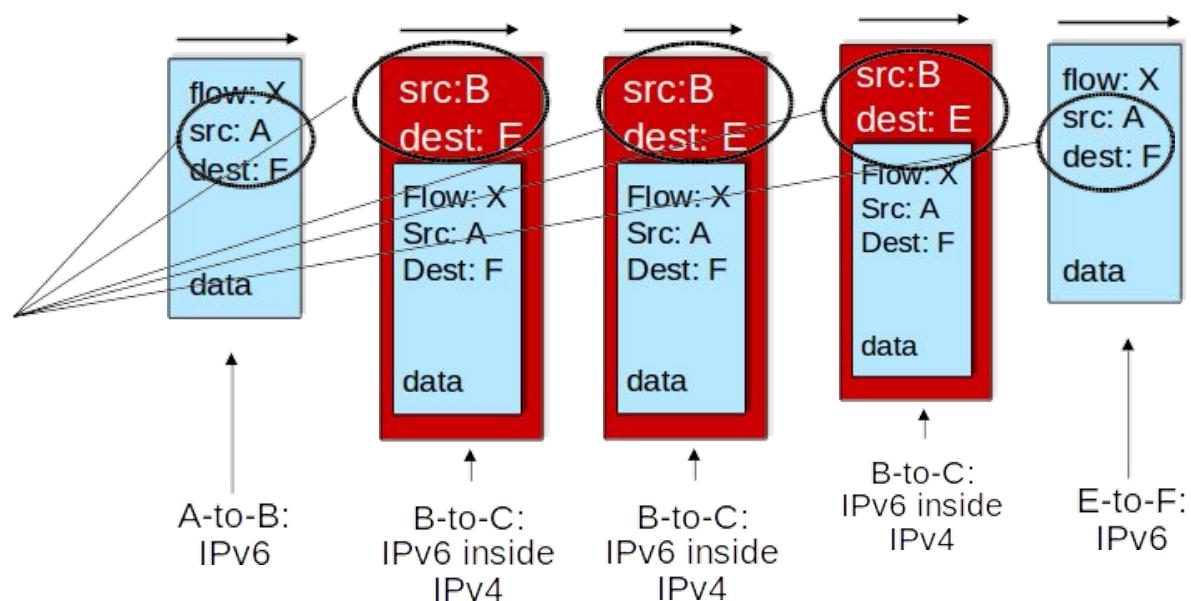
logical view:



physical view:

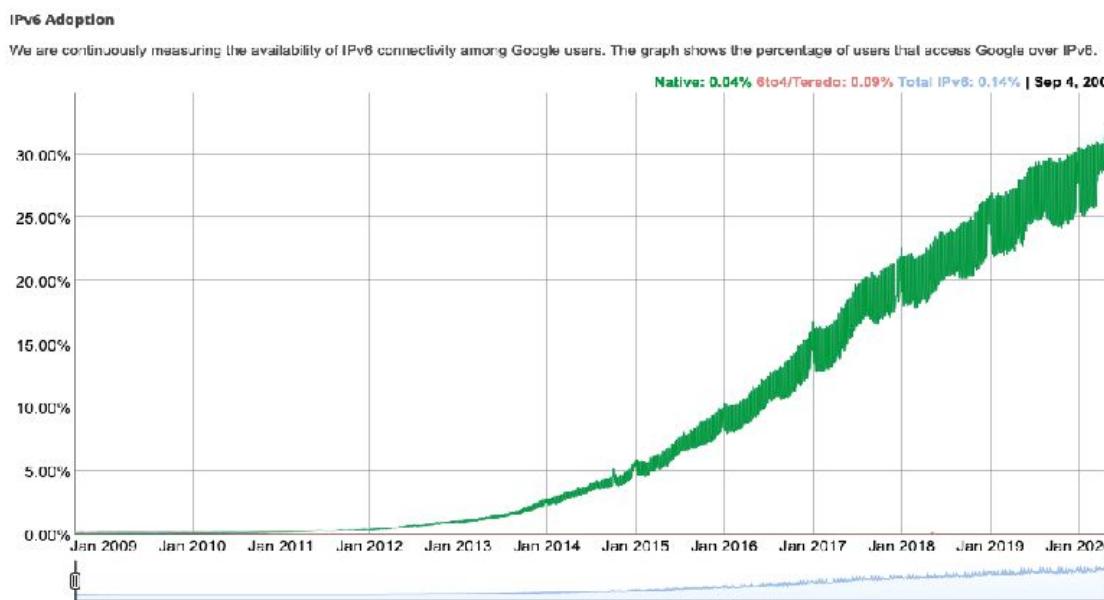


Note source and destination addresses!



IPv6: adoption

- Google: Approximately 30% of clients access services via IPv6
 - <https://www.google.com/intl/en/ipv6/statistics.html>
- NIST: 1/3 of all US government domains are IPv6 capable



1

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

IPv6: adoption

- Google: Approximately 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable
- Long (long!) time for deployment, use
 - Introducing new protocols in the network layer is like replacing the foundation of a house
 - think of application-level changes in last 20 years: WWW, Social media, streaming media, gaming, ...
 - Like adding a new layer of paint to a house
 - Changes will occur on a time scale that is much slower than the changes that will occur at the application layer

outline

- Overview of Network layer
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - datagram format
 - Fragmentation
 - IPv4 addressing
 - network address translation
 - **IPv6**
- **Generalized Forward and SDN**
 - Match+ Action abstraction
 - OpenFlow

outline

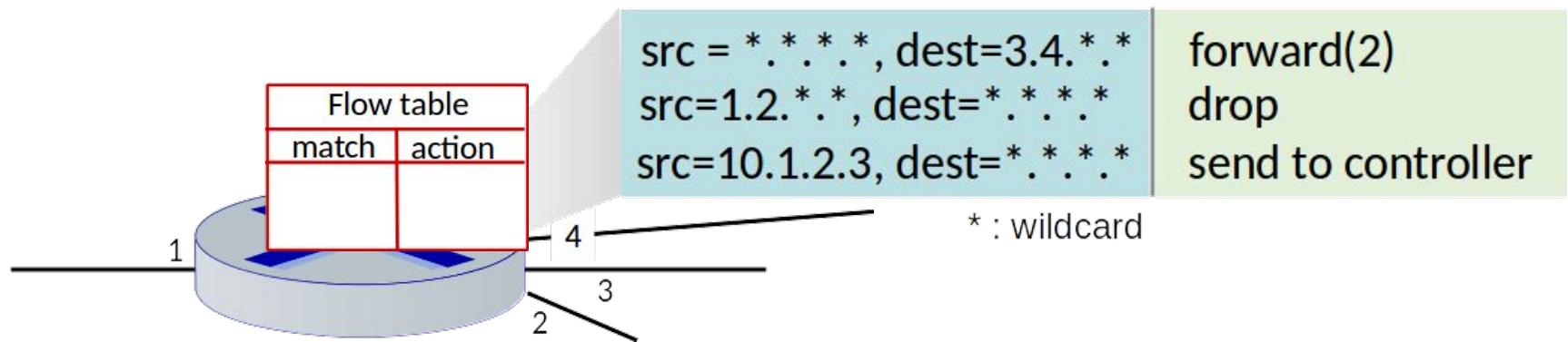
- Overview of Network layer
 - data plane
 - control plane
- What's inside a router?
- IP: Internet Protocol
 - datagram format
 - Fragmentation
 - IPv4 addressing
 - network address translation
 - **IPv6**
- **Generalized Forward and SDN**
 - Match+ Action abstraction
 - OpenFlow

Generalized forwarding

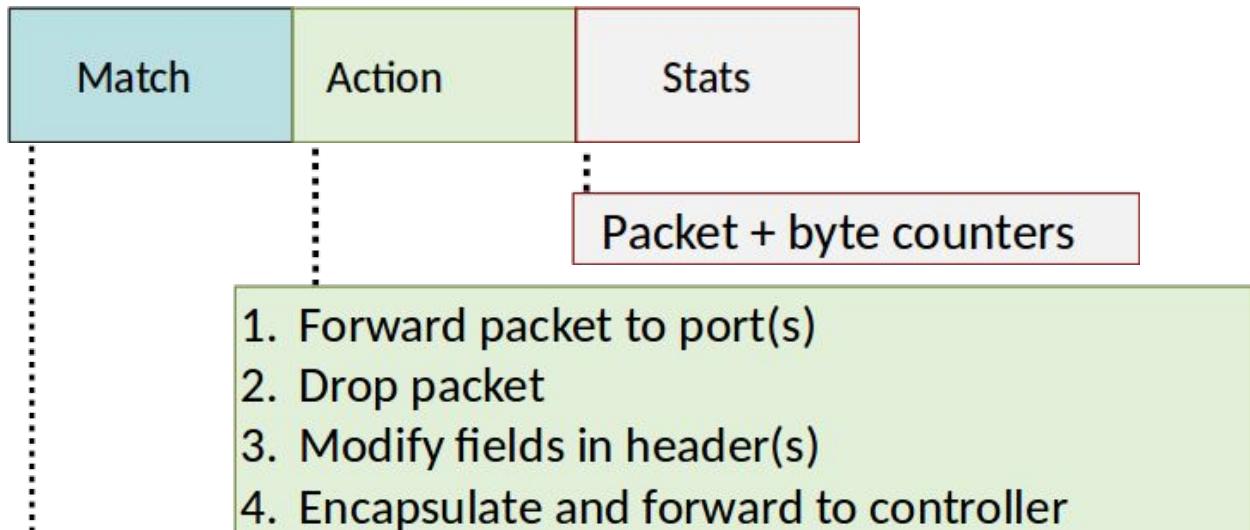
- Review: each router contains a forwarding table
 - “match plus action” abstraction: match bits in arriving packet, take action
 - **Destination-based forwarding:** forward based on dest. IP address
 - **Match:** looking up a destination IP
 - **Action:** sending the packet into the switching fabric to the specified output port
 - **Generalized forwarding**
 - **Match**
 - is made **over multiple header fields** associated with different protocols at different layers in the protocol stack
 - **Action**
 - forwarding, load balancing, rewriting header values, blocking/dropping a packet, sending the packet into the switching fabric to the specified output port

OpenFlow data plane abstraction

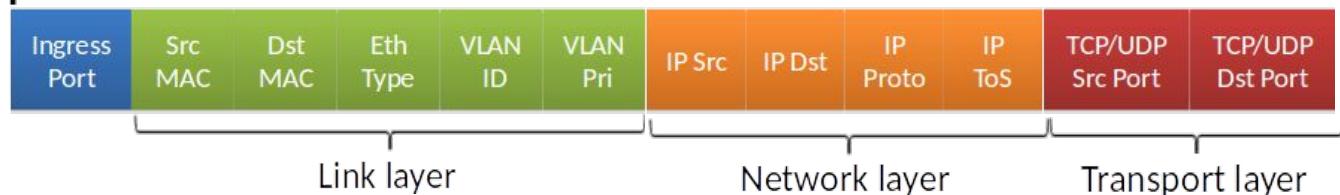
- **flow:** defined by header field values
- **generalized forwarding:** simple packet-handling rules
 - **match:** pattern values in packet header fields
 - **actions:** for matched packet: drop, forward, modify, matched packet or send matched packet to SDN controller
 - **Priority:** disambiguate overlapping patterns
 - **Counters:** #bytes and #packets



OpenFlow: Flow Table Entries



Header fields to match:



OpenFlow: Examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	128.119.1.1	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

OpenFlow: Examples

Layer 2 destination-based forwarding:

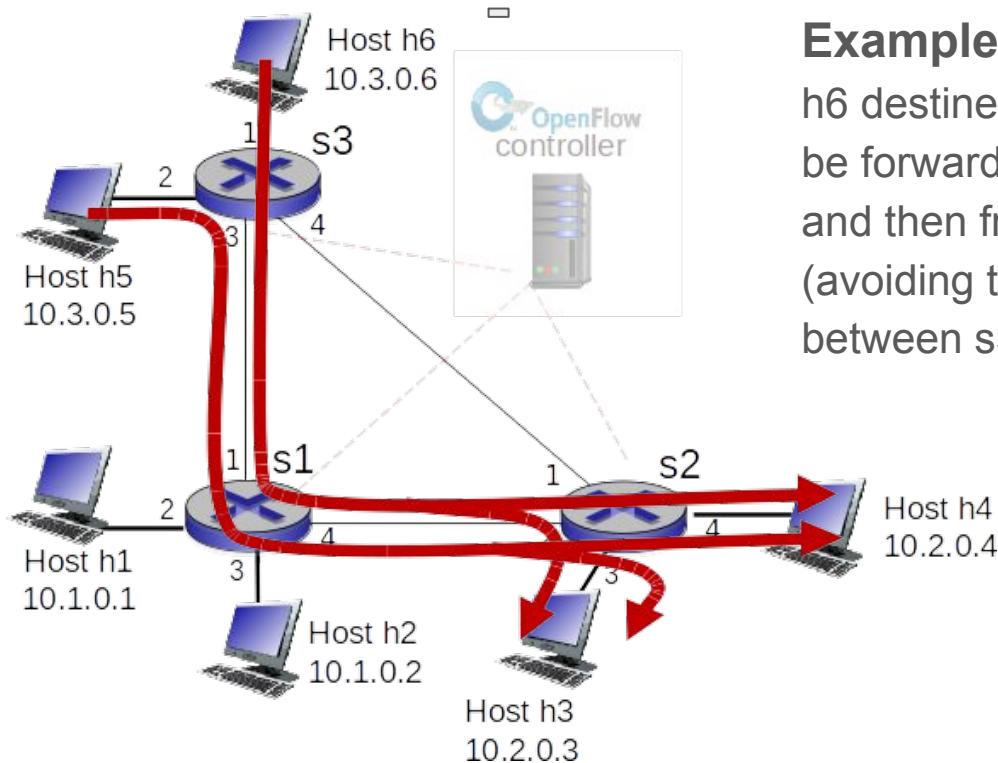
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

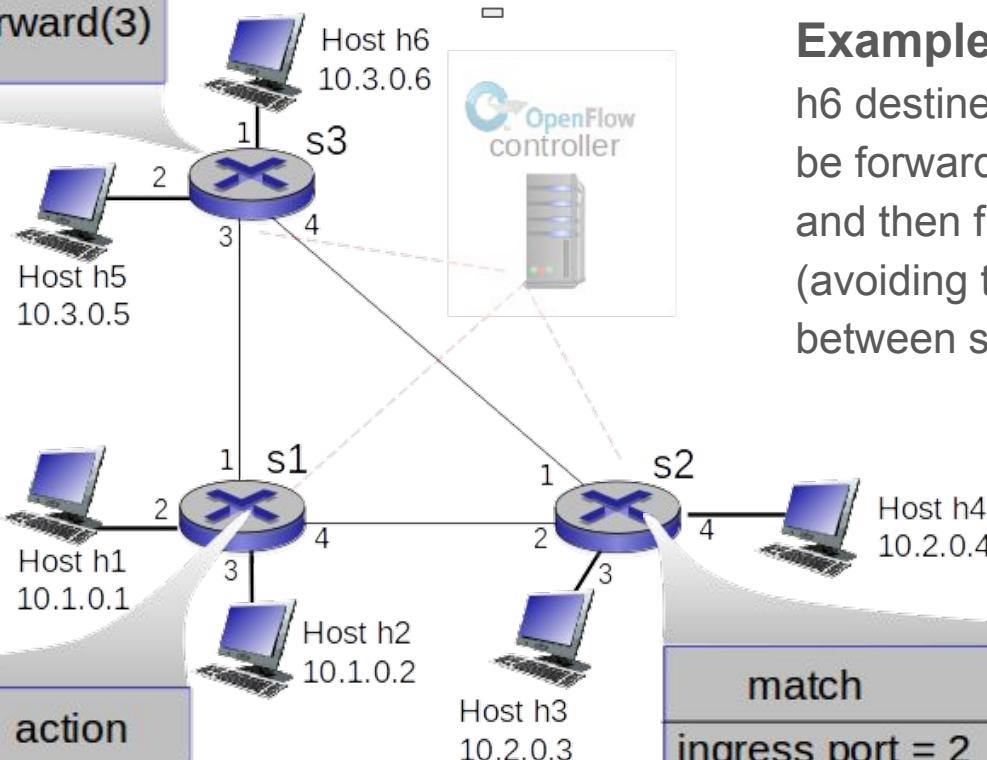
OpenFlow abstraction

- **match+action:** abstraction unifies different kinds of devices
 - Router
 - **match:** longest destination IP prefix
 - **action:** forward out a link
 - Switch
 - **match:** destination MAC address
 - **action:** forward or flood
 - Firewall
 - **match:** IP addresses and TCP/UDP port numbers
 - **action:** permit or deny
 - NAT
 - **match:** IP address and port
 - **action:** rewrite address and port

Orchestrated tables can create network-wide behavior, **Example**: packets from h5 or h6 destined to h3 or h4 are to be forwarded from s3 to s1, and then from s1 to s2 (avoiding the use of the link between s3 and s2)



match	action
IP Src = 10.3.*.*	
IP Dst = 10.2.*.*	forward(3)



Orchestrated tables can create network-wide behavior, **Example:** packets from h5 or h6 destined to h3 or h4 are to be forwarded from s3 to s1, and then from s1 to s2 (avoiding the use of the link between s3 and s2)

match	action
ingress port = 1	
IP Src = 10.3.*.*	forward(4)
IP Dst = 10.2.*.*	

match	action
ingress port = 2	
IP Dst = 10.2.0.3	forward(3)
ingress port = 2	
IP Dst = 10.2.0.4	forward(4)

Generalized forwarding: summary

- “match plus action” abstraction: match bits in arriving packet header(s) in any layers, take action
 - matching over many fields (link-, network-, transport-layer)
 - local actions: drop, forward, modify, or send matched packet to controller
 - “program” network-wide behaviors
- SDN
 - Provide a unified approach towards providing many of these network-layer functions, and certain link-layer functions that are done with middleboxes
- simple form of “network programmability”
 - P4: programming language
 - [https://en.wikipedia.org/wiki/P4_\(programming_language\)](https://en.wikipedia.org/wiki/P4_(programming_language))

Network Layer

Routing

Chapter 5: network layer control plane

- Goals: understand principles behind network control plane
 - traditional routing algorithms
 - SDN controllers
 - network management, configuration
- Instantiation, implementation in the Internet:
 - OSPF, BGP
 - OpenFlow, ODL and ONOS controllers
 - ICMP: Internet Control Message Protocol
 - SNMP

Outline

- introduction
- routing protocols
 - link state
 - distance vector
- intra-ISP routing in the Internet: OSPF
- routing among the ISPs: BGP
- SDN control plane
- ICMP: The Internet Control Message Protocol
- Network management, configuration
 - SNMP

Network-layer functions

Recall: two network-layer functions:

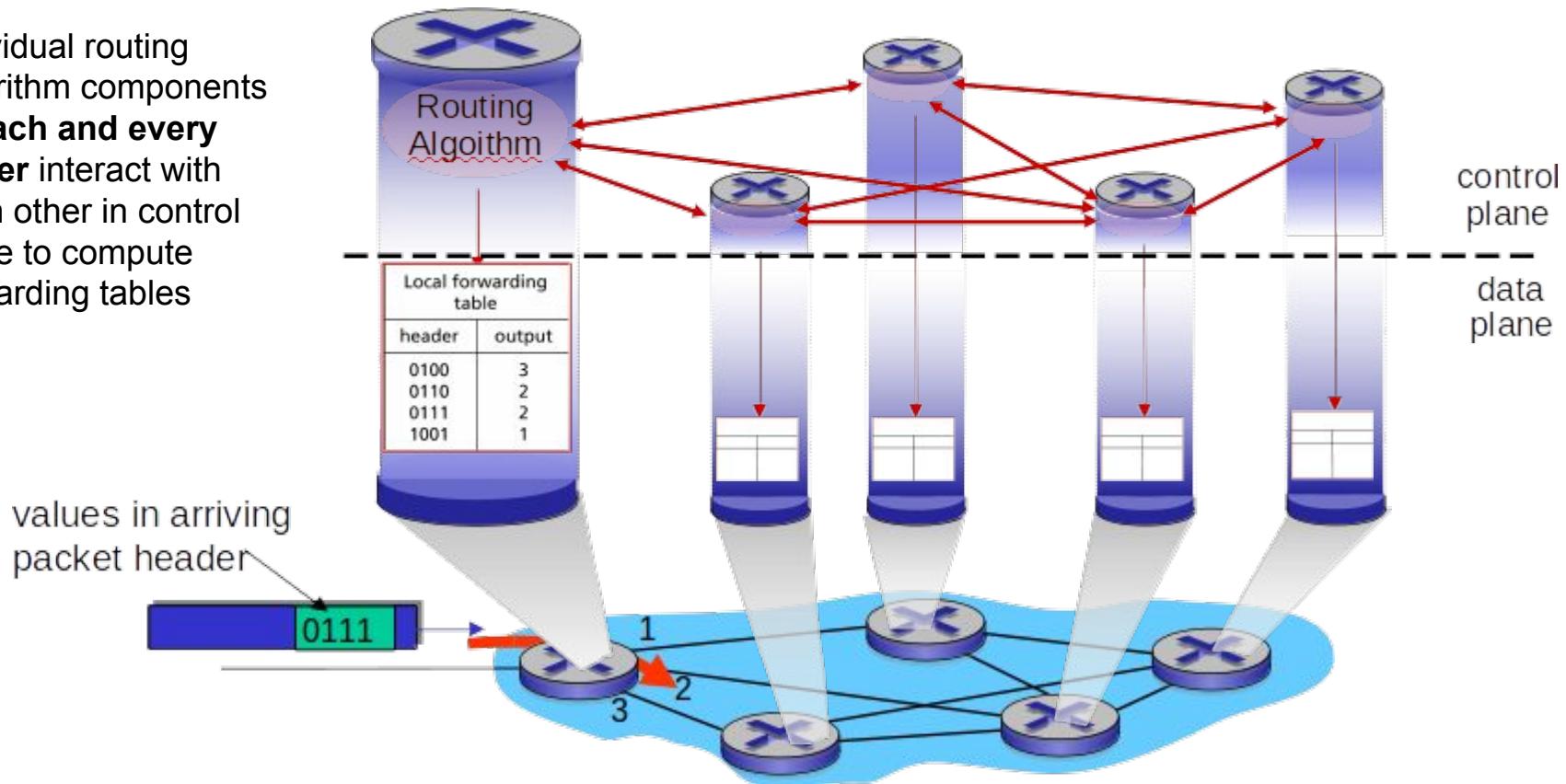
- **forwarding**: move packets from router's input to appropriate router output *data plane*
- **routing**: determine route taken by packets from source to destination *control plane*

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

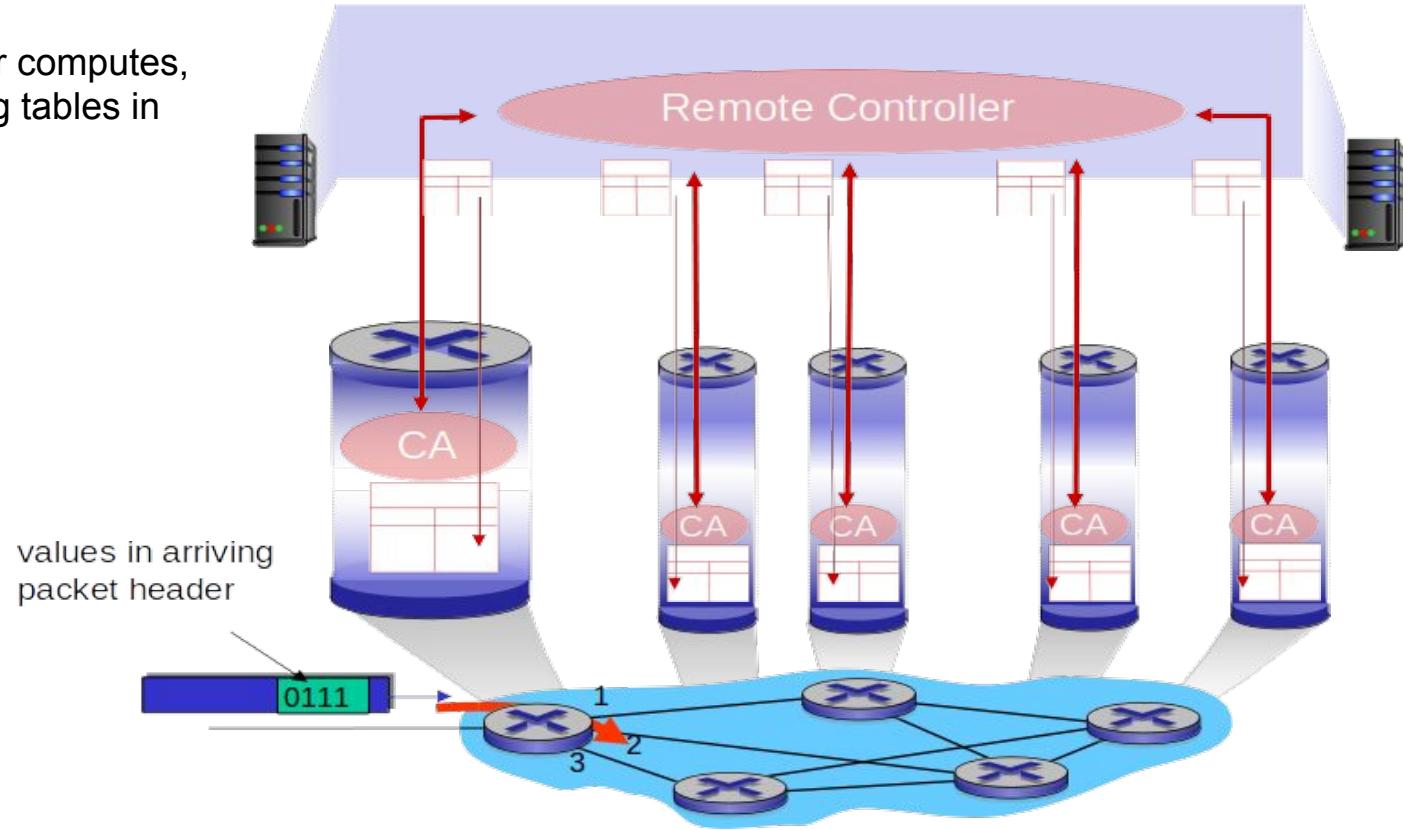
Per-router control plane

Individual routing algorithm components **in each and every router** interact with each other in control plane to compute forwarding tables



Software-Defined Networking (SDN) control plane

Remote controller computes,
installs forwarding tables in
routers



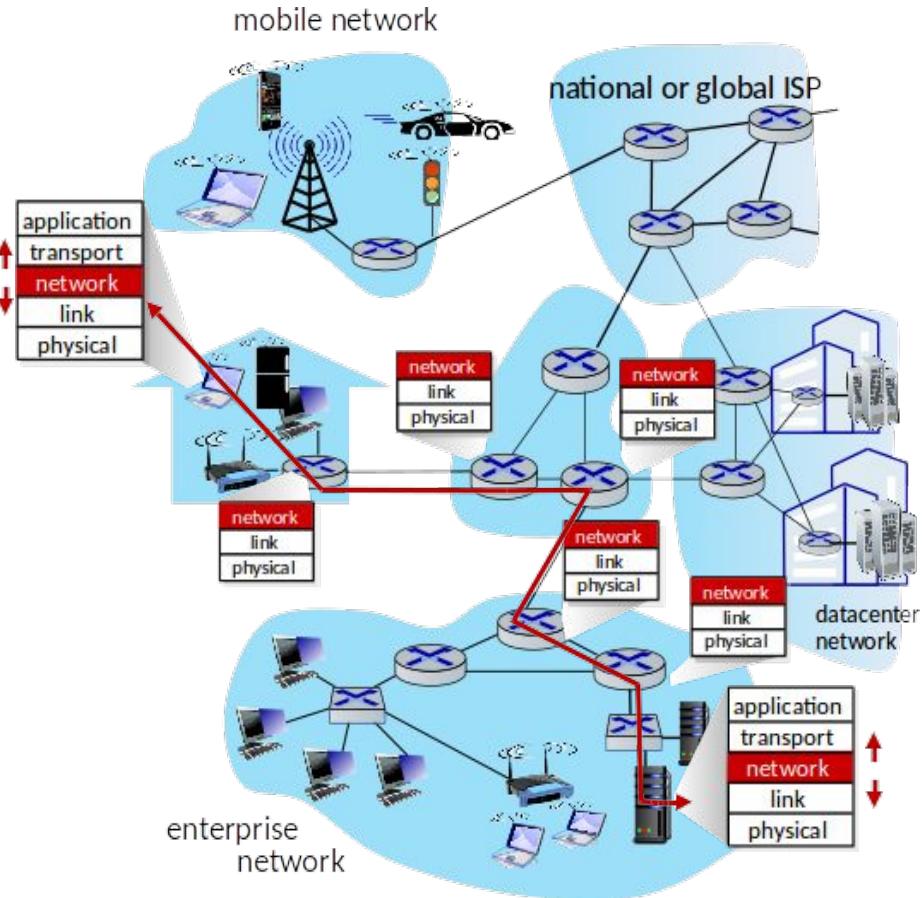
Outline

- introduction
- **routing protocols**
 - link state
 - distance vector
- intra-AS routing in the Internet: OSPF
- routing among the ISPs: BGP
- The SDN control plane
- ICMP: The Internet Control Message Protocol
- Network management and SNMP

Routing protocols

Routing protocol goal: determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets will traverse in going from given initial source host to final destination host
- “**good**”: least “cost”, “fastest”, “least congested”
- routing: a “top-10” networking challenge!



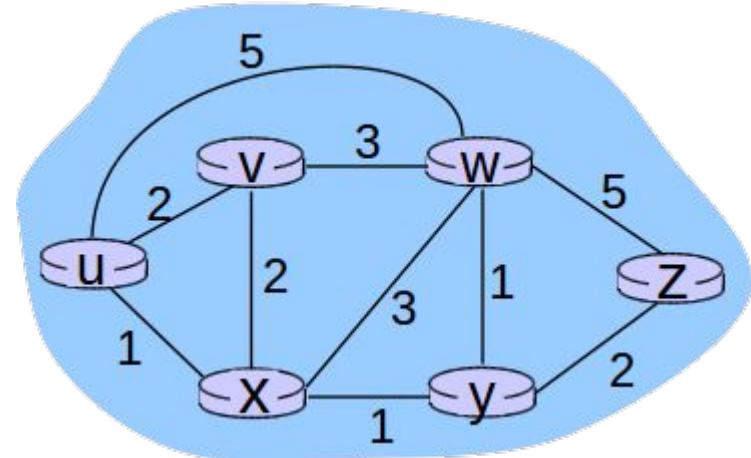
Graph abstraction of the network

A graph is used to formulate routing problem

graph: $G = (N, E)$

$N = \text{set of routers} = \{ u, v, w, x, y, z \}$

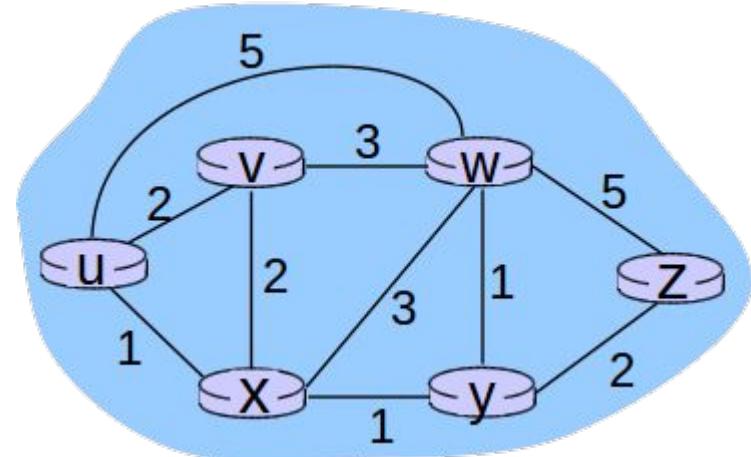
$E = \text{set of links} = \{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$



- aside: graph abstraction is useful in other network contexts, e.g., P2P, where N is set of peers and E is set of TCP connections

Graph abstraction: costs

- $c_{a,b}$ = cost of **direct** link connecting a and b
e.g., $c_{w,z} = 5$, $c_{u,z} = \text{infty}$
- Cost defined by network operator
 - cost could always be 1, or inversely related to bandwidth, or inversely related to congestion



Cost of path $(x_1, x_2, x_3, \dots, x_p) = c_{x_1,x_2} + c_{x_2,x_3} + c_{x_3,x_4} + \dots + c_{x_{p-1},x_p}$

key question: what is the least-cost path between u and z ?

routing algorithm: algorithm that finds that least cost path

Routing algorithm classification

- Q: global or decentralized information?
 - **Global:**
 - all routers have complete topology, link cost info
 - “**link state**” algorithms
 - **decentralized:**
 - router knows physically-connected neighbors, link costs to neighbors
 - iterative process of computation and exchange of info with neighbors
 - “**distance vector**” algorithms
- Q: static or dynamic?
 - **static:** routes change slowly over time
 - **dynamic:** routes change more quickly
 - periodic updates
 - Updates in response to link cost changes or topology change

Outline

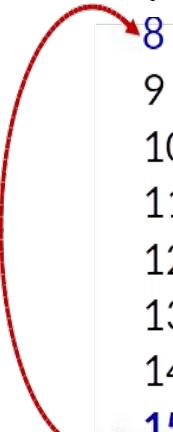
- introduction
- routing protocols
 - link state
 - distance vector
- intra-AS routing in the Internet: OSPF
- routing among the ISPs: BGP
- The SDN control plane
- ICMP: The Internet Control Message Protocol
- Network management and SNMP

A link-state routing algorithm

- **Dijkstra's algorithm**
 - Centralized: network topology, link costs known to all nodes
 - accomplished via “link state broadcast”
 - all nodes have same info
 - computes least cost paths from one node (“source”) to all other nodes
 - gives **forwarding table** for that node
 - **iterative**: after k iterations, know least cost path to k destinations
- Notation
 - $c_{x,y}$: direct link cost from node x to y; $= \infty$ if not direct neighbors
 - $D(v)$: current estimate of cost of least-cost-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

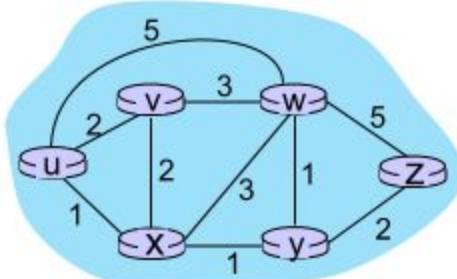
Dijkstra's link-state routing algorithm

```
1 Initialization:  
2    $N' = \{u\}$                                 /* compute least cost path from u to all other nodes */  
3   for all nodes v  
4     if v adjacent to u          /* u initially knows direct-path-cost only to direct neighbors */  
5       then  $D(v) = c_{u,v}$       /* but may not be minimum cost! */  
6     else  $D(v) = \infty$   
7  
8 Loop  
9   find w not in  $N'$  such that  $D(w)$  is a minimum  
10  add w to  $N'$   
11  update  $D(v)$  for all v adjacent to w and not in  $N'$  :  
12     $D(v) = \min(D(v), D(w) + c_{w,v})$   
13  /* new least-path-cost to v is either old least-cost-path to v or known  
14  least-cost-path to w plus direct-cost from w to v */  
15 until all nodes in  $N'$ 
```



Dijkstra's algorithm: an example

Step	N'	v	w	x	y	z
0	u	$2,u$	$5,u$	$1,u$	∞	∞
1						
2						
3						
4						
5						

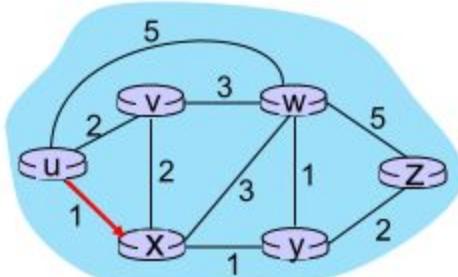


Initialization (step 0):

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

Dijkstra's algorithm: an example

Step	N'	v	w	x	y	z
0	u					
1	ux	2,u	5,u	1,u	∞	∞
2						
3						
4						
5						

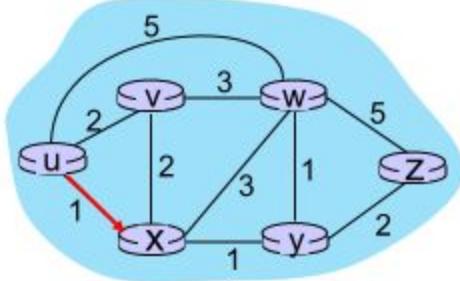


8 Loop

- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'

Dijkstra's algorithm: an example

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



8 Loop

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

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

$$D(v) = \min(D(v), D(x) + c_{x,v}) = \min(2, 1+2) = 2$$

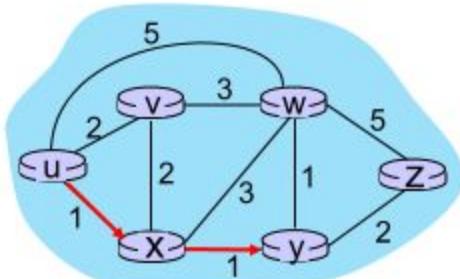
$$D(w) = \min(D(w), D(x) + c_{x,w}) = \min(5, 1+3) = 4$$

$$D(y) = \min(D(y), D(x) + c_{x,y}) = \min(\infty, 1+1) = 2$$



Dijkstra's algorithm: an example

Step	N'	v	w	x	y	z
0	u	$2,u$	$5,u$	$1,u$	∞	∞
1	ux	$2,u$	$4,x$	$2,x$	∞	∞
2	uxy					
3						
4						
5						

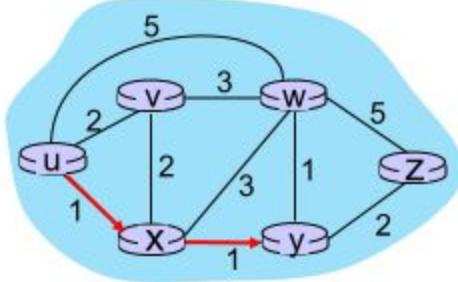


8 Loop

- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'

Dijkstra's algorithm: an example

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



8 Loop

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

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

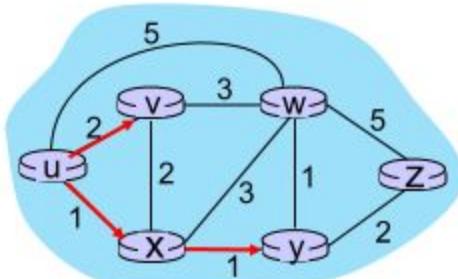
$$D(w) = \min(D(w), D(y) + c_{x,w}) = \min(4, 2+1) = 3$$

$$D(z) = \min(D(z), D(y) + c_{y,x}) = \min(\infty, 2+2) = 4$$

NEW!

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	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv					
4						
5						

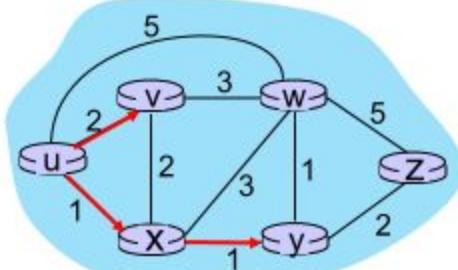


8 Loop

- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to 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	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4						
5						



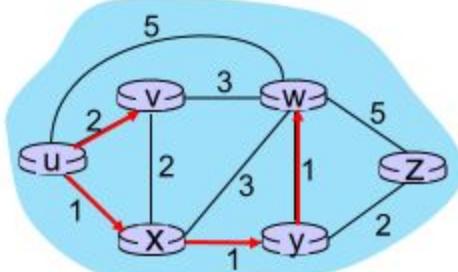
8 Loop

- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 update $D(b)$ for all b adjacent to a and not in N' :
$$D(b) = \min (D(b), D(a) + c_{a,b})$$

$$D(w) = \min (D(w), D(v) + c_{v,w}) = \min (3, 2+3) = 3$$

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	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyw					
5						

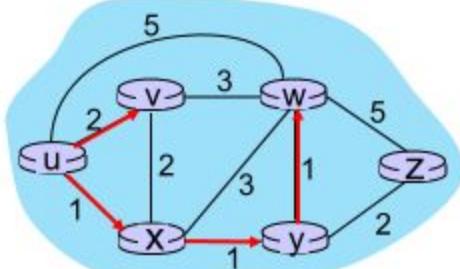


8 Loop

- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'

Dijkstra's algorithm: an example

Step	N'	v $D(v), p(v)$	w $D(w), p(w)$	x $D(x), p(x)$	y $D(y), p(y)$	z $D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5						



8 Loop

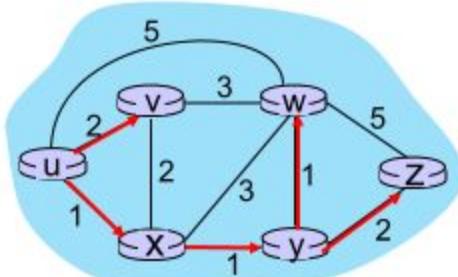
- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 update $D(b)$ for all b adjacent to a and not in N' :

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

$$D(z) = \min (D(z), D(w) + c_{w,z}) = \min (4, 3+5) = 4$$

Dijkstra's algorithm: an example

Step	N'	v	w	x	y	z
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



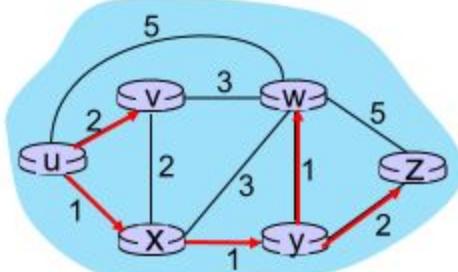
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's algorithm: an example

Step	N'	v $D(v), p(v)$	w $D(w), p(w)$	x $D(x), p(x)$	y $D(y), p(y)$	z $D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



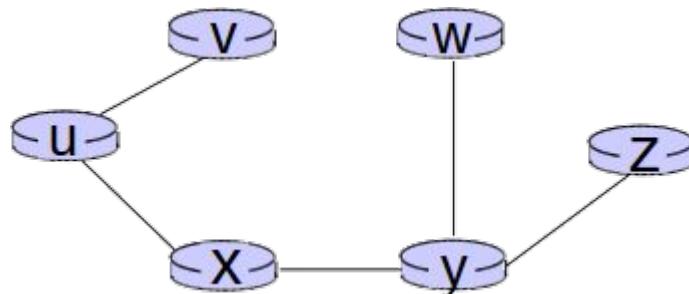
8 Loop

- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 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: example

resulting shortest-path tree from u:



resulting forwarding table in u:

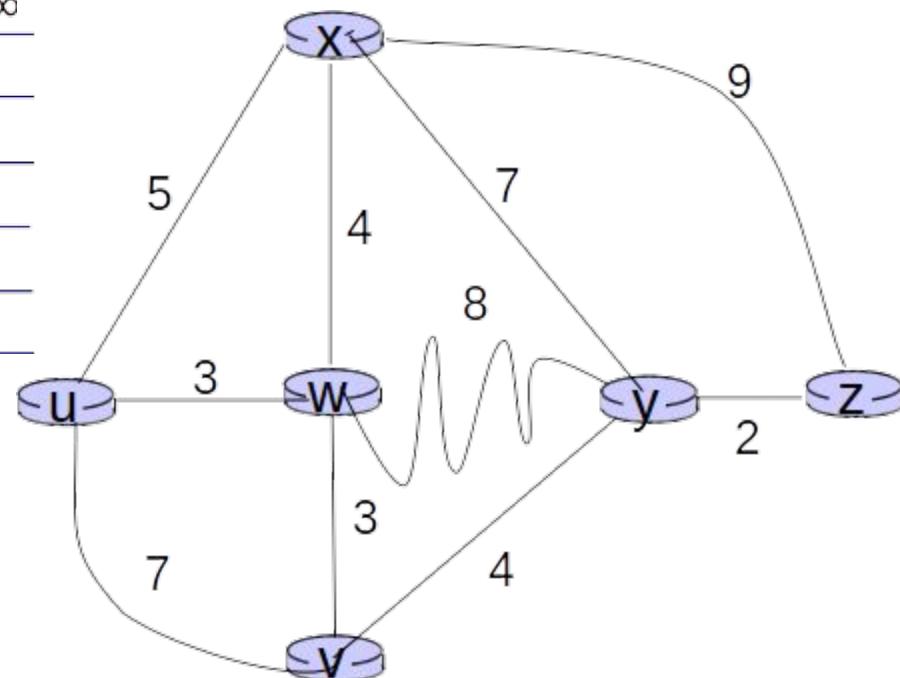
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

route from u to v directly

route from u to all other destinations via x

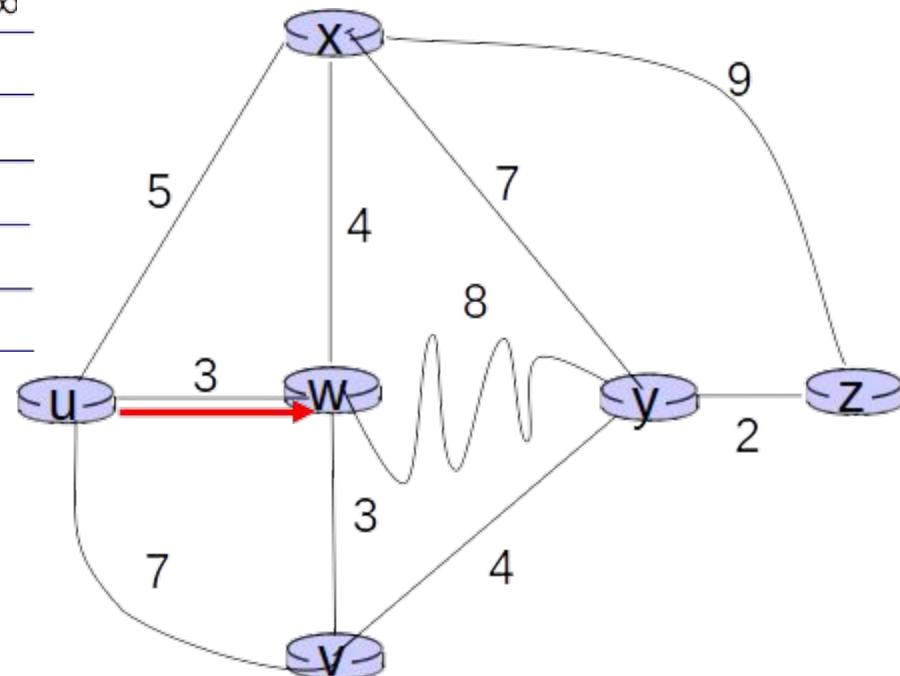
Dijkstra's algorithm: example

Step	N'	D(v)	D(w)	D(x)	D(y)	D(z)
		p(v)	p(w)	p(x)	p(y)	p(z)
0	u	7,u	3,u	5,u	∞	∞
1						
2						
3						
4						
5						



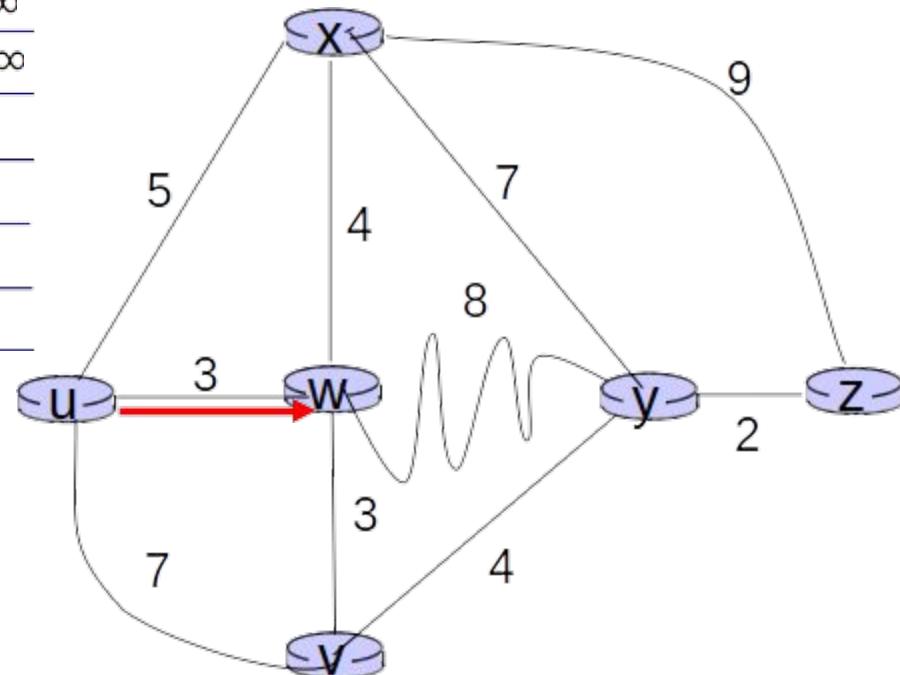
Dijkstra's algorithm: example

Step	N'	D(v)	D(w)	D(x)	D(y)	D(z)
		p(v)	p(w)	p(x)	p(y)	p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw					
2						
3						
4						
5						



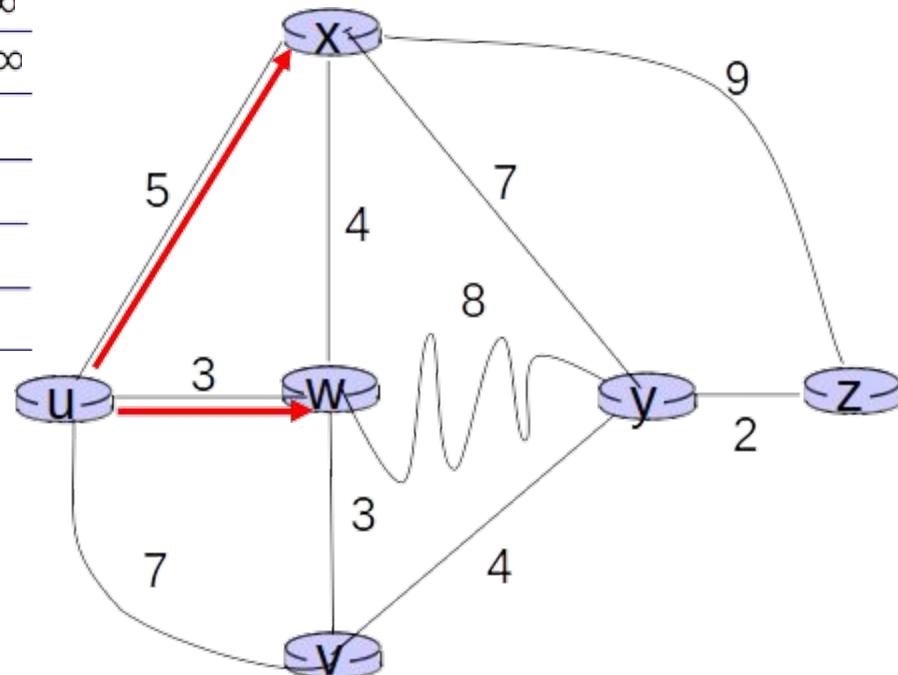
Dijkstra's algorithm: example

Step	N'	$D(v)$ $p(v)$	$D(w)$ $p(w)$	$D(x)$ $p(x)$	$D(y)$ $p(y)$	$D(z)$ $p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2						
3						
4						
5						



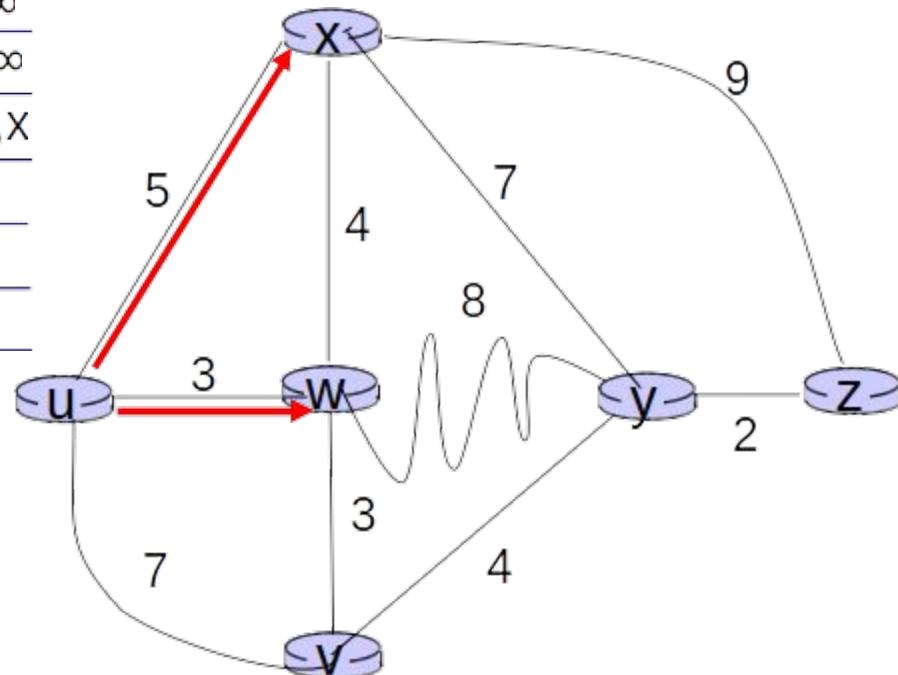
Dijkstra's algorithm: example

Step	N'	$D(v)$ $p(v)$	$D(w)$ $p(w)$	$D(x)$ $p(x)$	$D(y)$ $p(y)$	$D(z)$ $p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,W	∞
2	uwx					
3						
4						
5						



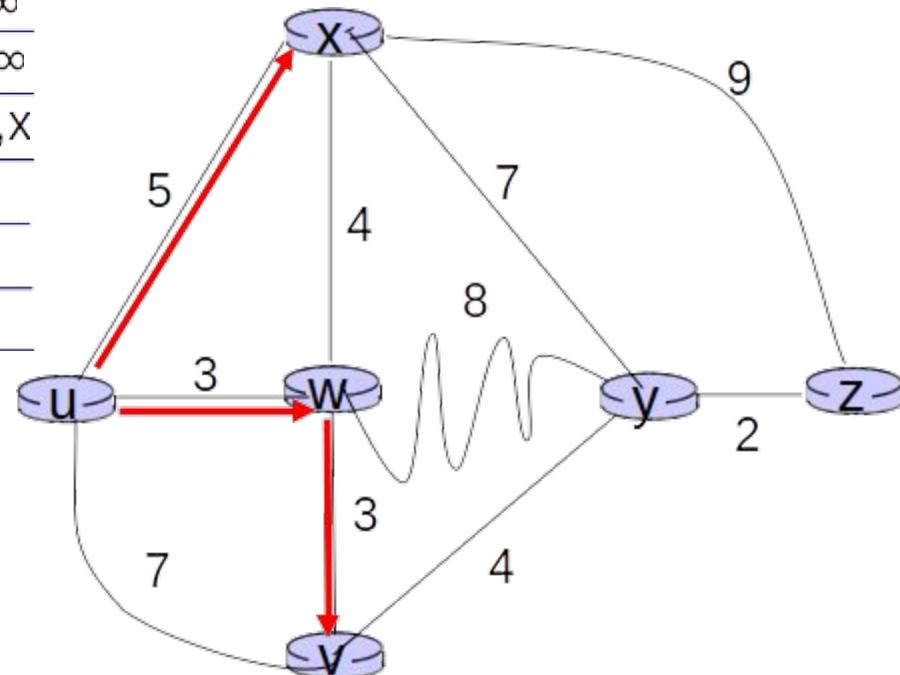
Dijkstra's algorithm: example

Step	N'	$D(v)$	$D(w)$	$D(x)$	$D(y)$	$D(z)$
		$p(v)$	$p(w)$	$p(x)$	$p(y)$	$p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3						
4						
5						



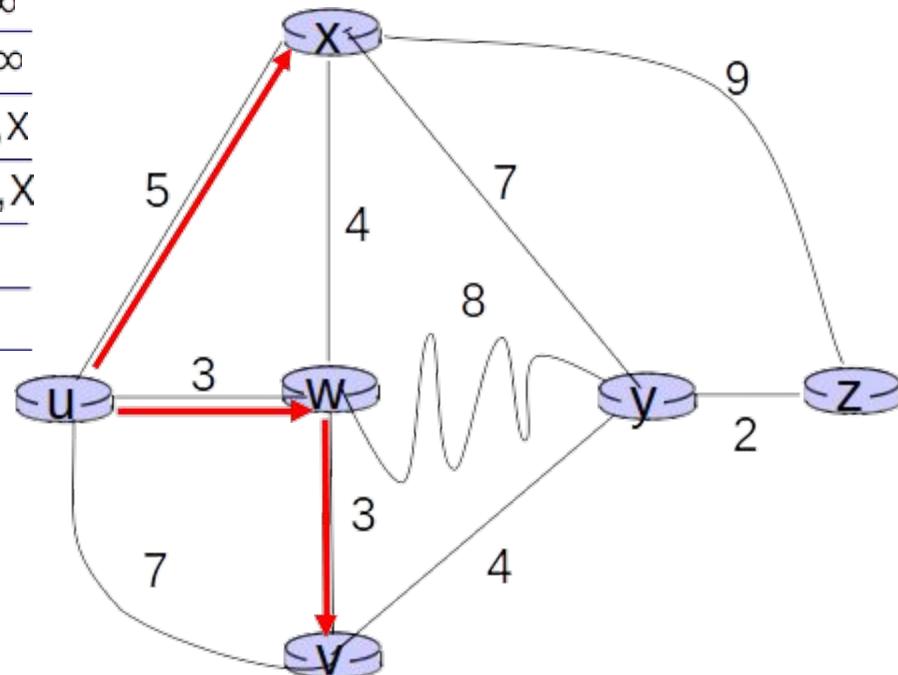
Dijkstra's algorithm: example

Step	N'	$D(v)$ $p(v)$	$D(w)$ $p(w)$	$D(x)$ $p(x)$	$D(y)$ $p(y)$	$D(z)$ $p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv					
4						
5						



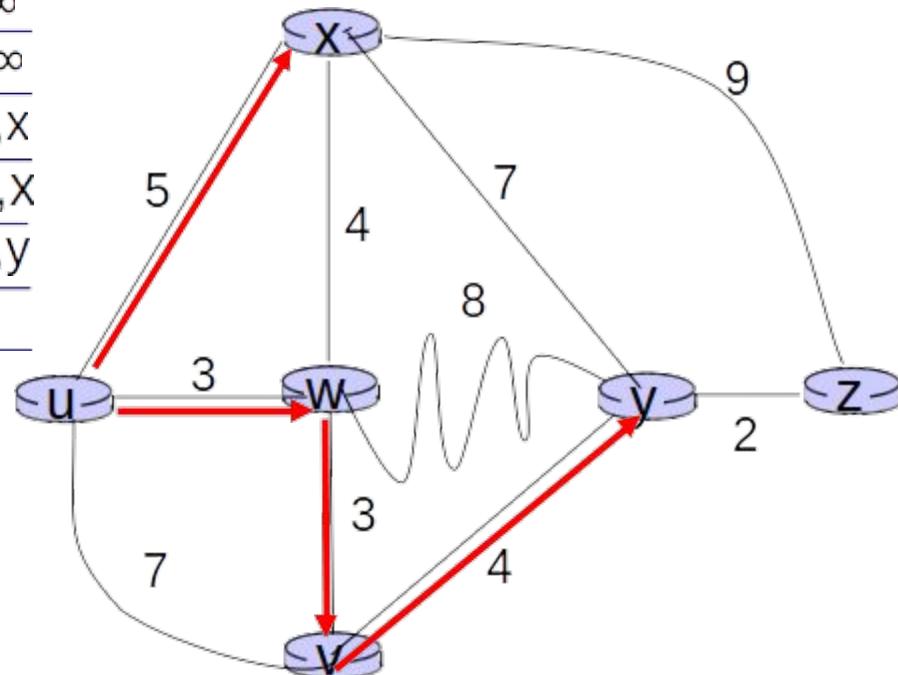
Dijkstra's algorithm: example

Step	N'	$D(v)$	$D(w)$	$D(x)$	$D(y)$	$D(z)$
		$p(v)$	$p(w)$	$p(x)$	$p(y)$	$p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4						
5						



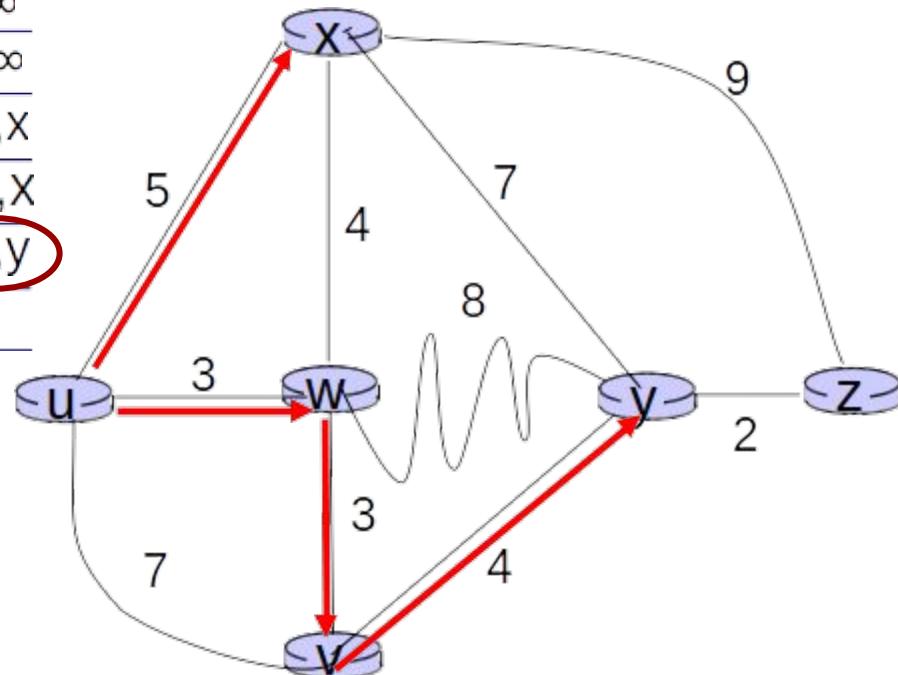
Dijkstra's algorithm: example

Step	N'	$D(v)$ $p(v)$	$D(w)$ $p(w)$	$D(x)$ $p(x)$	$D(y)$ $p(y)$	$D(z)$ $p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w		11,w	14,x	
3	uwxv			10,v	14,x	
4	uwxvy				12,y	
5						



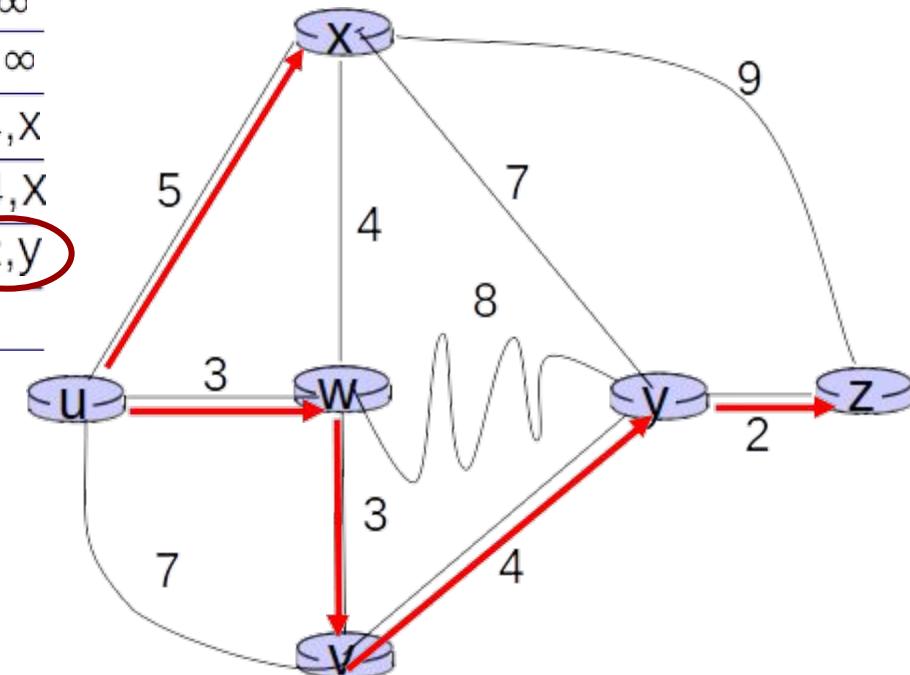
Dijkstra's algorithm: example

Step	N'	$D(v)$ $p(v)$	$D(w)$ $p(w)$	$D(x)$ $p(x)$	$D(y)$ $p(y)$	$D(z)$ $p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w		11,w	14,x	
3	uwxv			10,v	14,x	
4	uwxvy				12,y	
5	uwxvzy					



Dijkstra's algorithm: example

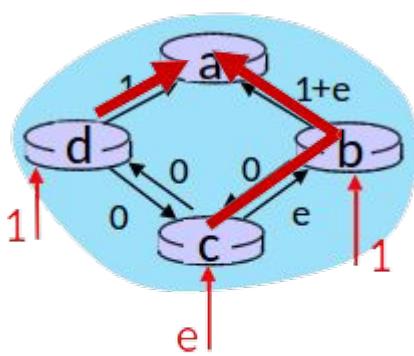
Step	N'	$D(v)$	$D(w)$	$D(x)$	$D(y)$	$D(z)$
		$p(v)$	$p(w)$	$p(x)$	$p(y)$	$p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w		11,w	14,x	
3	uwxv			10,v	14,x	
4	uwxvy				12,y	
5	uwxvyz					



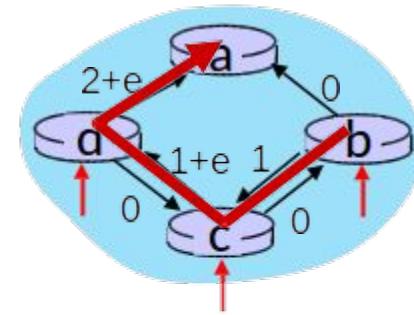
- Notes:
 - construct shortest path tree by tracing predecessor nodes
 - ties can exist (can be broken arbitrarily)

Dijkstra's algorithm: Oscillation problem

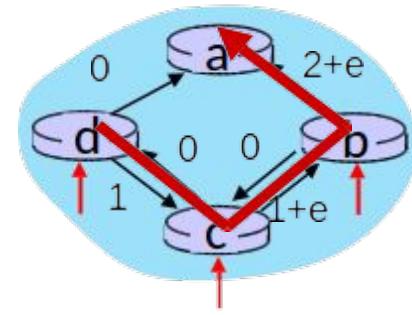
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
 - routing to destination a, traffic entering at d, c, b with rates 1, e (<1), 1
 - link costs are directional, and volume-dependent



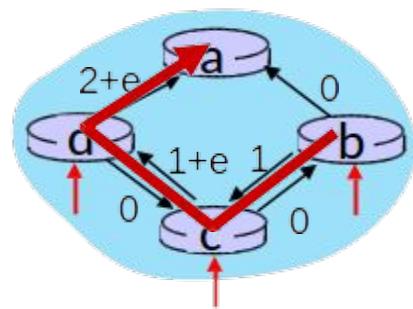
initially



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

Dijkstra's algorithm: Oscillation problem

- How to prevent oscillation
 - mandate the link cost not depend on the amount of traffic carried (not acceptable)
 - Not all routers run the LS algorithm at the same time
 - start at different time but the same period
 - routers self-synchronize
 - each router randomize the time it sends out a link advertisement

Outline

- introduction
- routing protocols
 - link state
 - **distance vector**
- intra-AS routing in the Internet: OSPF
- routing among the ISPs: BGP
- The SDN control plane
- ICMP: The Internet Control Message Protocol
- Network management and SNMP

Distance vector algorithm

Based on Bellman-Ford 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

Bellman-Ford Example

Bellman-Ford equation says:

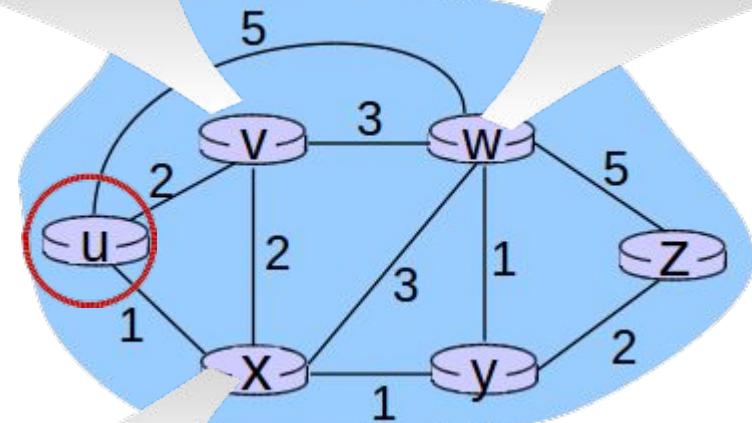
$$D_u(z) = \min \{ c_{u,v} + D_v(z), c_{u,x} + D_x(z), c_{u,w} + D_w(z) \}$$

$$= \min \{ 2 + 5, 1 + 3, 5 + 3 \} = 4$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

$$D_v(z) = 5$$

$$D_w(z) = 3$$



$$D_x(z) = 3$$

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

Distance vector algorithm

$D_x(y)$ = estimate of least cost from x to y

node x :

- knows cost to each neighbor v : $c_{x,v}$
- x maintains distance vector $D_x = [D_x(y) : y \in N]$
- maintains its neighbors' distance vectors. For each neighbor v , x maintains

$$D_v = [D_v(y) : y \in N]$$

node x table

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
z		∞	∞	∞

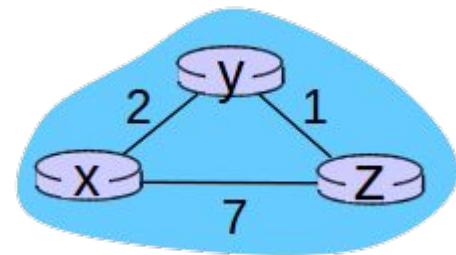
node y table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
z		∞	∞	∞

node z table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
z		7	1	0

→ time



**node x
table**

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

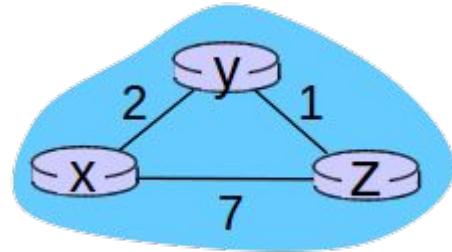
**node y
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

**node z
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

→ time



$$\begin{aligned}
 D_x(y) &= \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \\
 &= \min\{2+0, 7+1\} = 2
 \end{aligned}$$

**node x
table**

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
Z		∞	∞	∞

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
Z		7	1	0

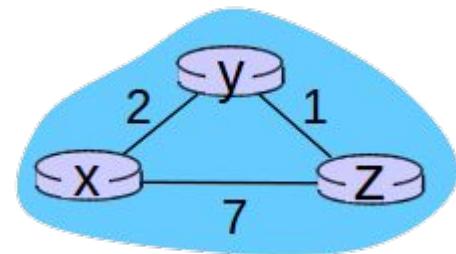
**node y
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
Z		∞	∞	∞

**node z
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
Z		7	1	0

time



$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x table

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

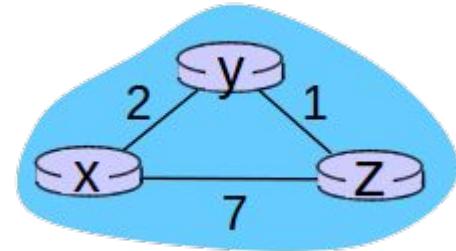
node y table

from	x	y	z
x	∞	∞	∞
y	∞	0	1
z	∞	∞	∞

node z table

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0



time

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x table

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

node y table

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

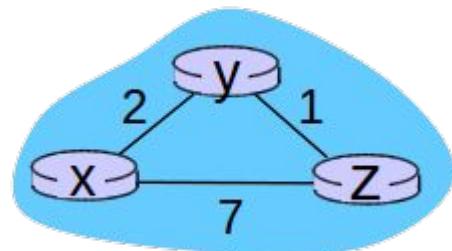
	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

→ time



$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x table

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

node y table

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

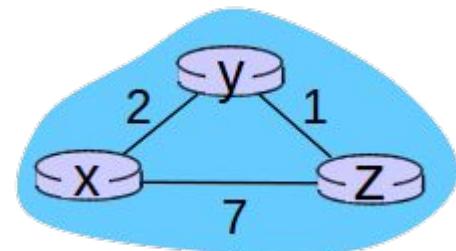
	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



time

Distance vector algorithm

- Distributed:
 - each node notifies neighbors *only* when its DV changes
 - neighbors then notify their neighbors if necessary
 - no notification received, no actions taken!
- Iterative
 - each local iteration caused by:
 - local link cost change
 - DV update message from neighbor
 - Continues until no more information is exchanged between neighbors
 - self-terminating
- Asynchronous:
 - It does not require all of the nodes to operate in lockstep with each other

each node:

```
graph TD; A[each node:] --> B[wait for (change in local link cost or msg from neighbor)]; B --> C[recompute DV estimates using DV received from neighbor]; C --> D;if DV to any destination has changed, notify neighbors;
```

wait for (change in local link cost or msg from neighbor)

recompute DV estimates using DV received from neighbor

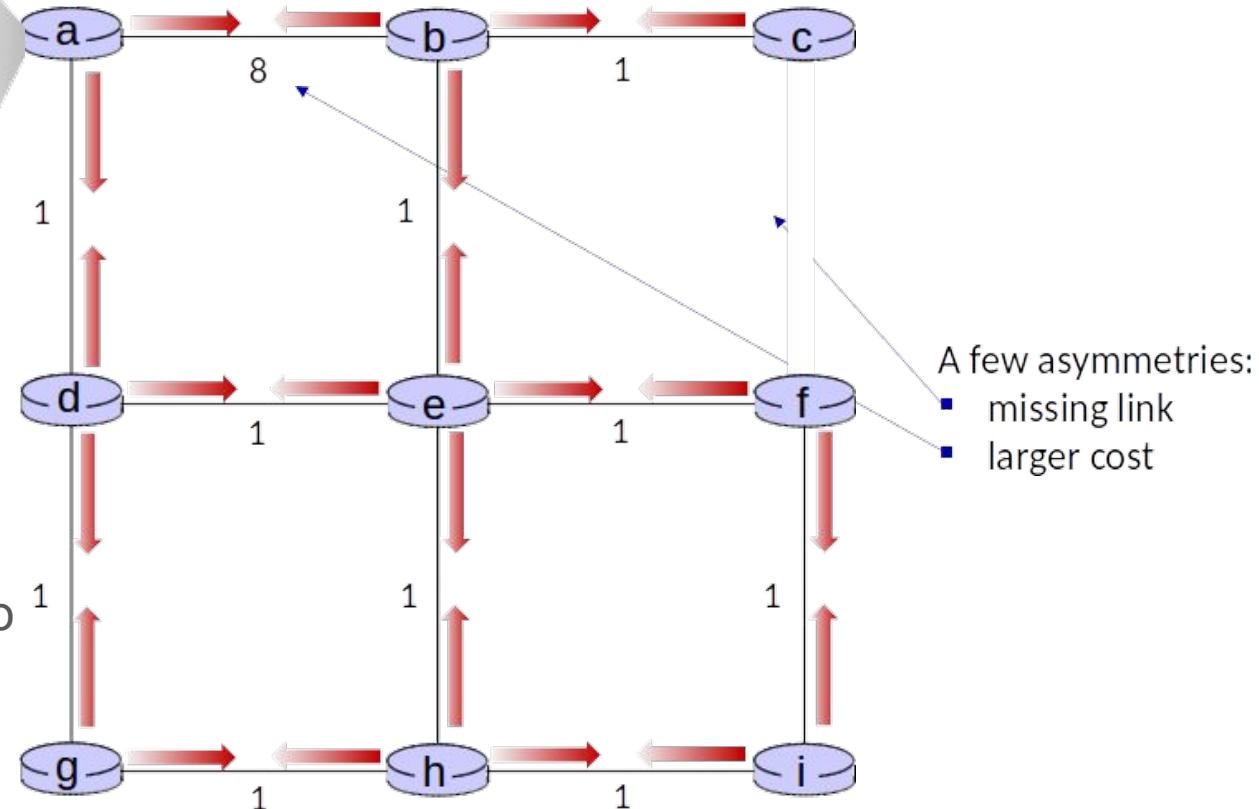
if DV to any destination has changed, *notify* neighbors



$t=0$

DV in a:

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

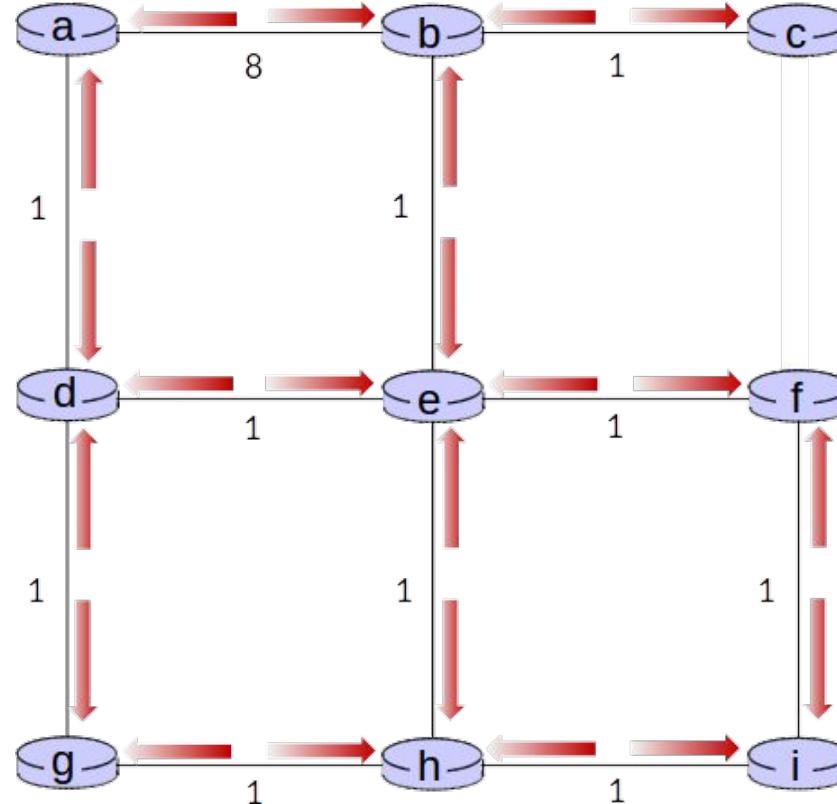


- All nodes send their local distance vector to their neighbors



$t=1$

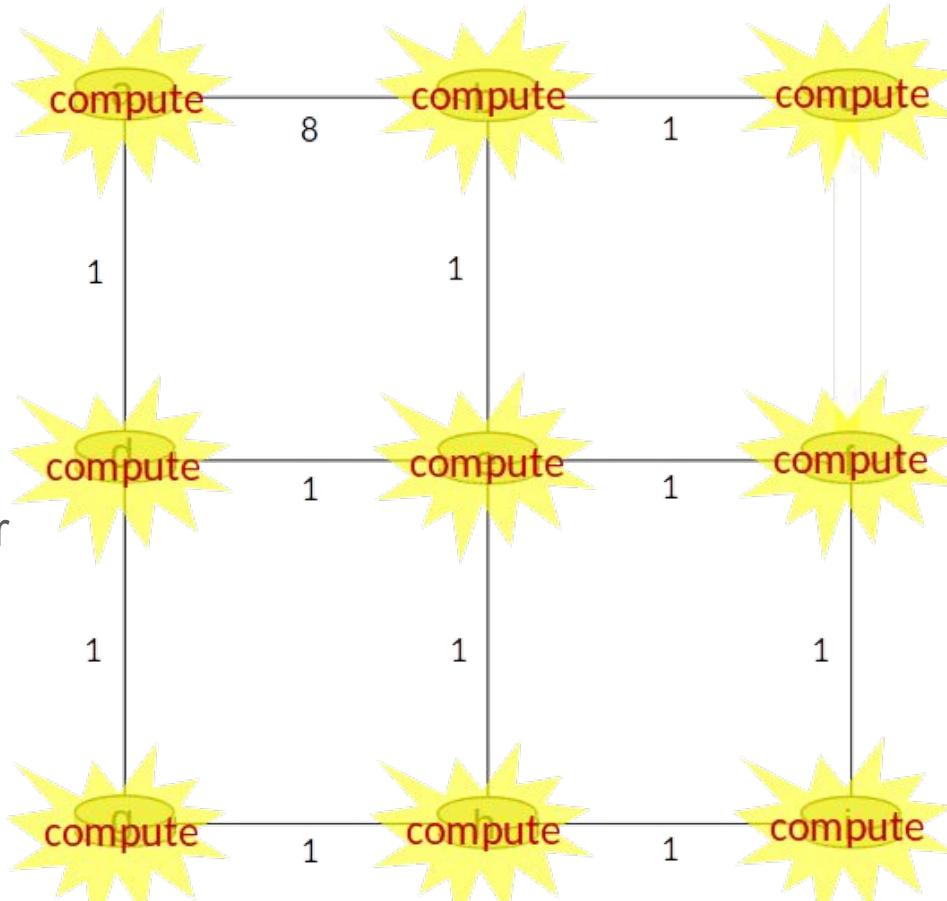
- receive distance vectors from neighbors





$t=1$

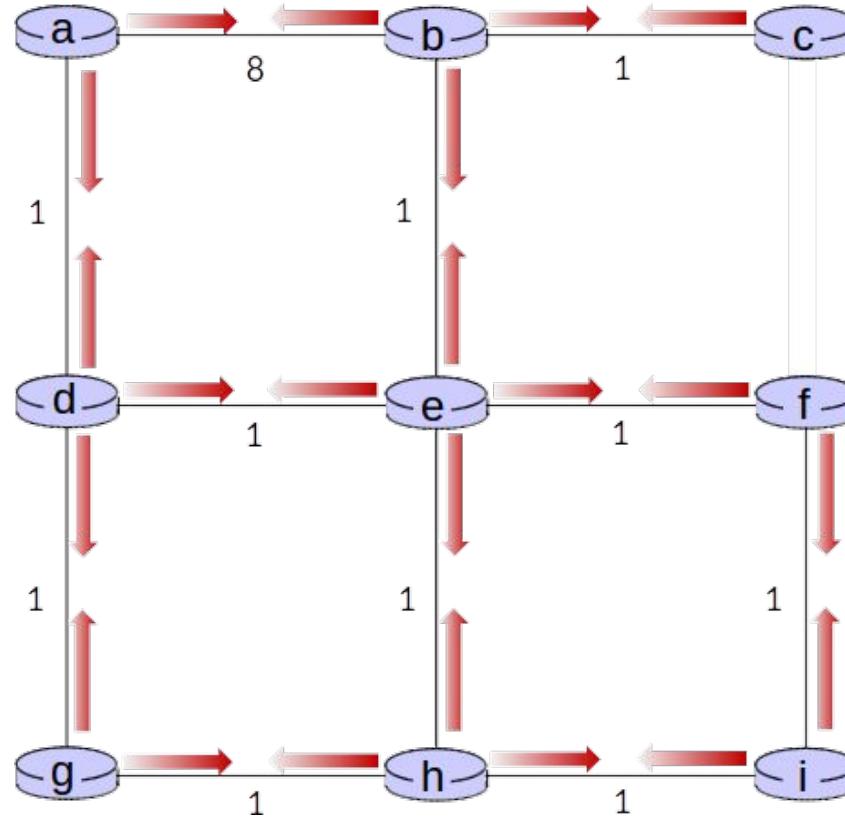
- compute their new local distance vector





$t=1$

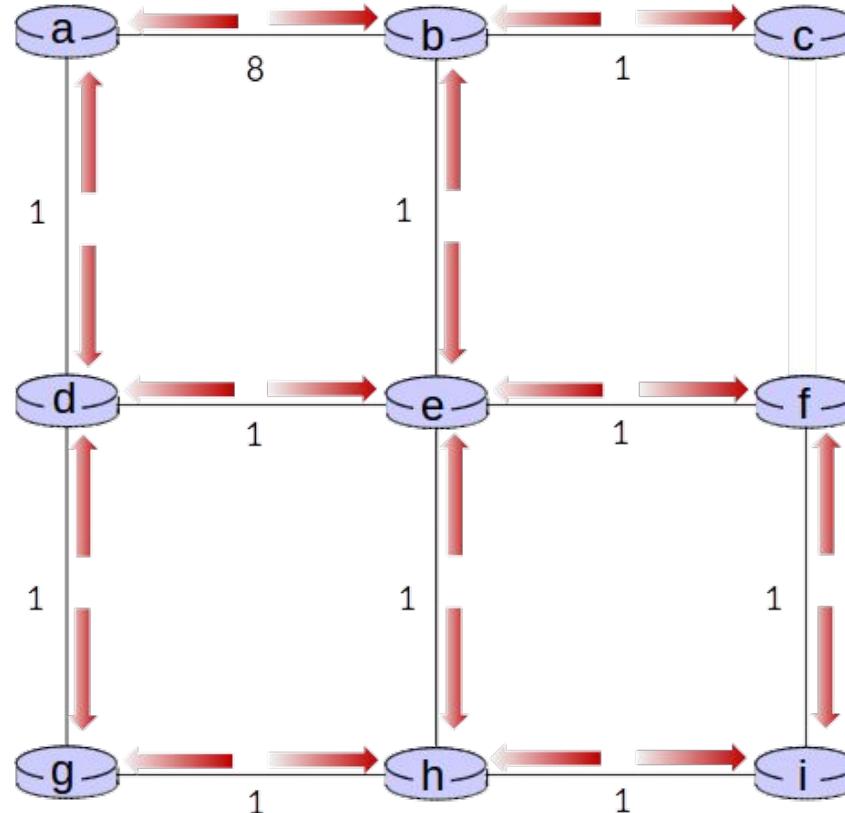
- send their new local distance vector to neighbors





$t=2$

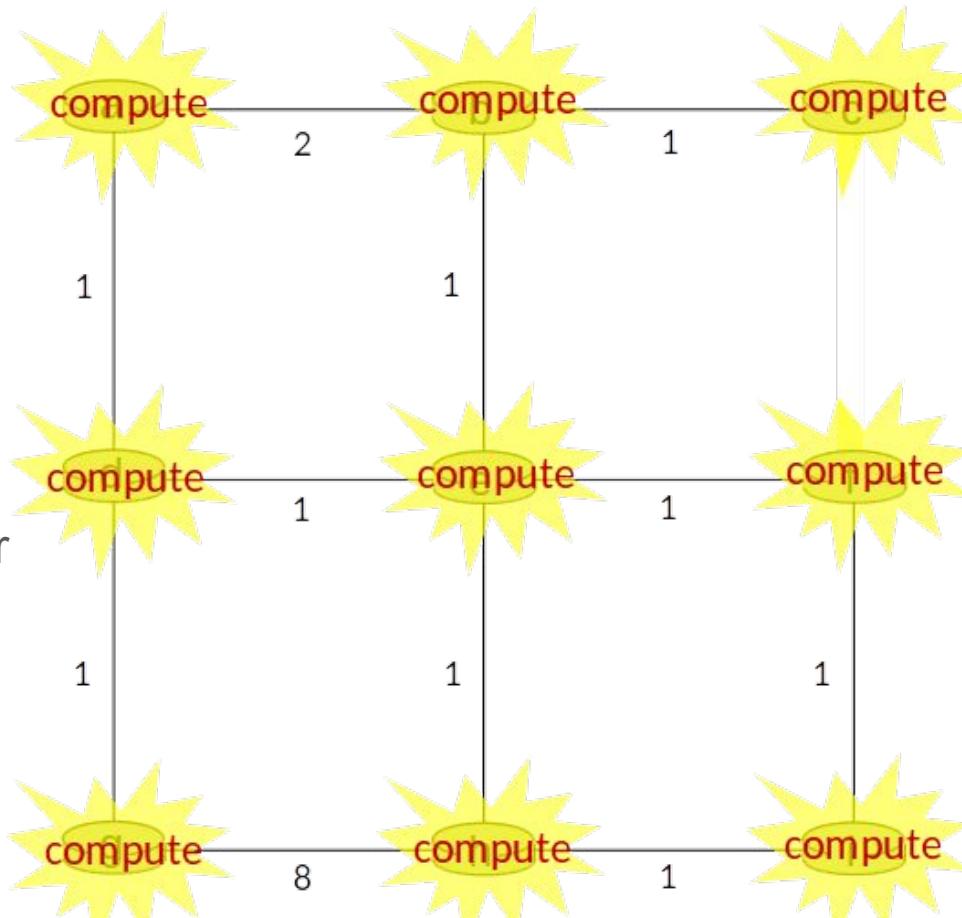
- receive distance vectors from neighbors





t=2

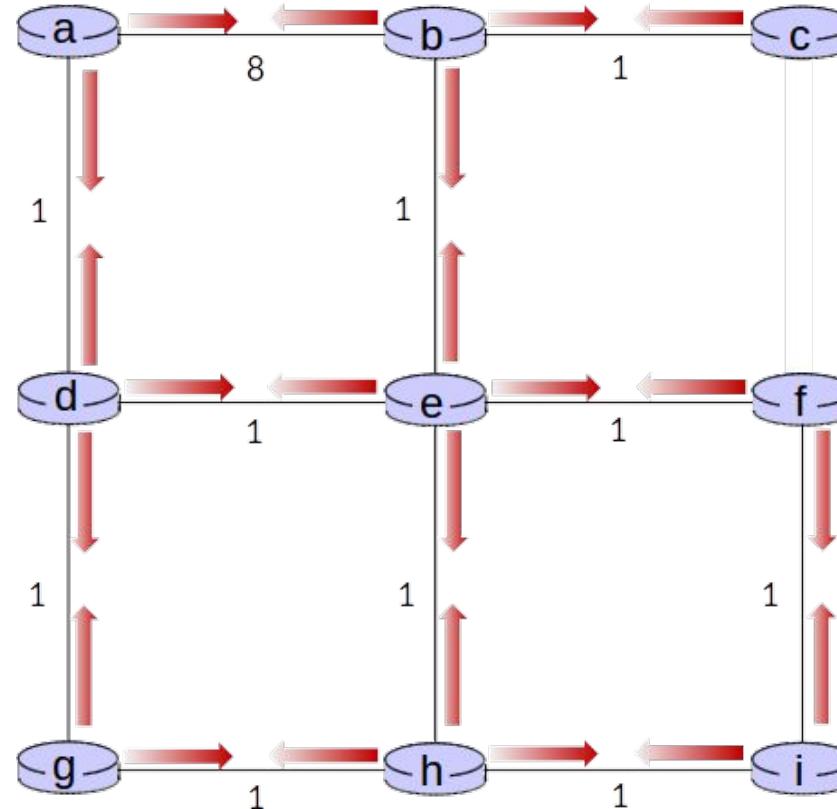
- compute their new local distance vector





$t=2$

- send their new local distance vector to neighbors



Distance vector example: iteration

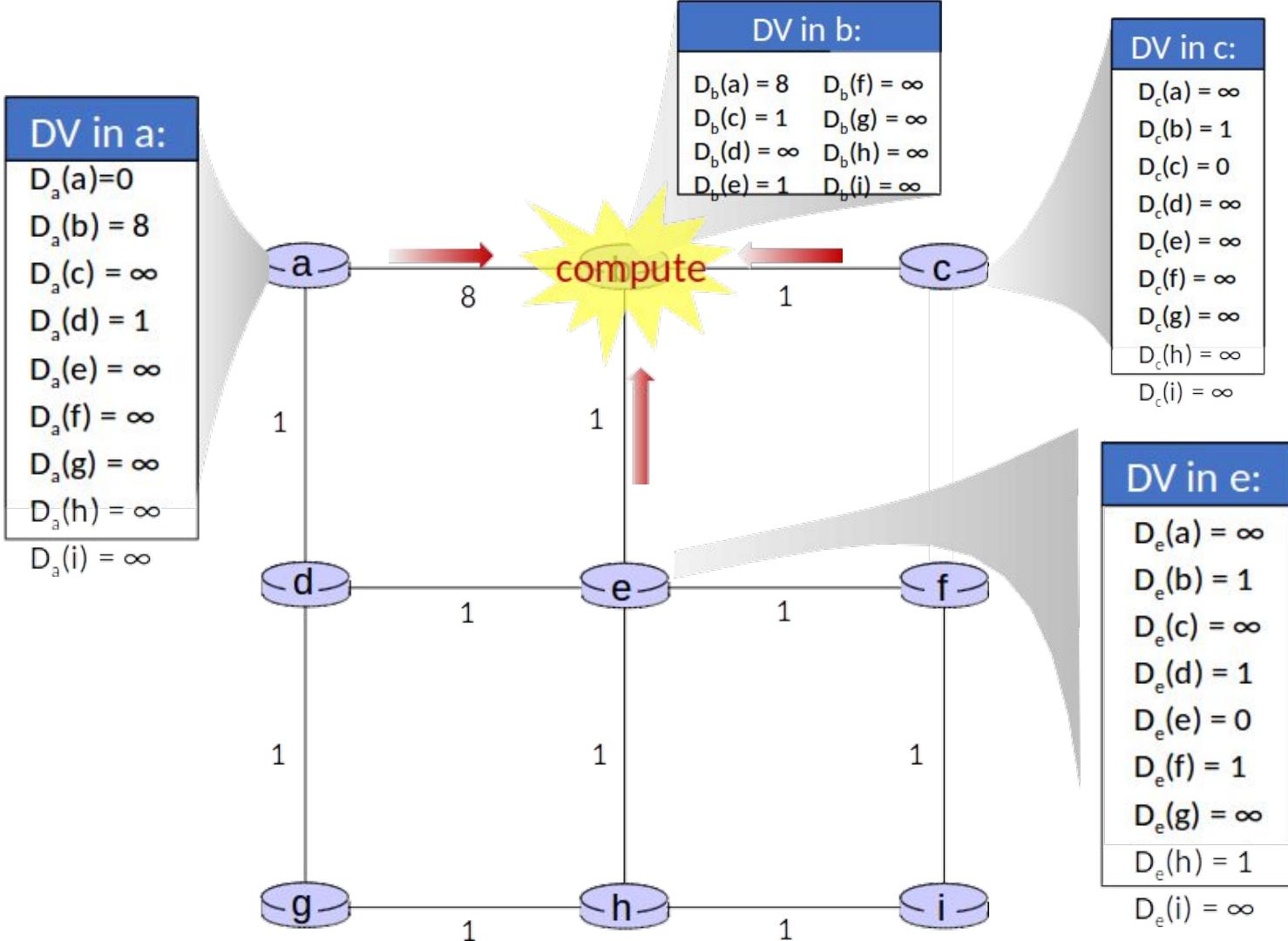
.... and so on

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



$t=1$

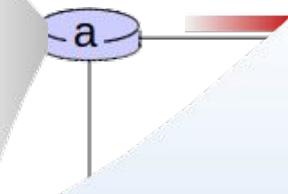
- b receives DVs from a, c, e



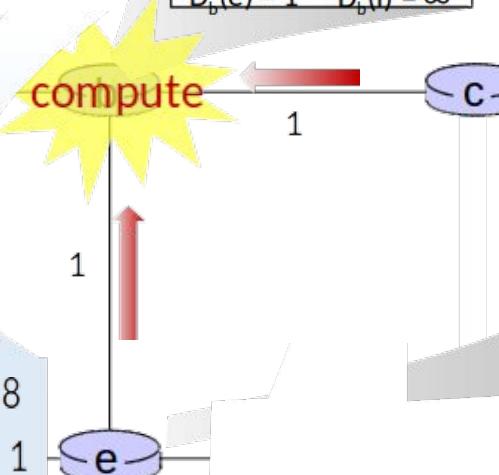


$t=1$

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



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

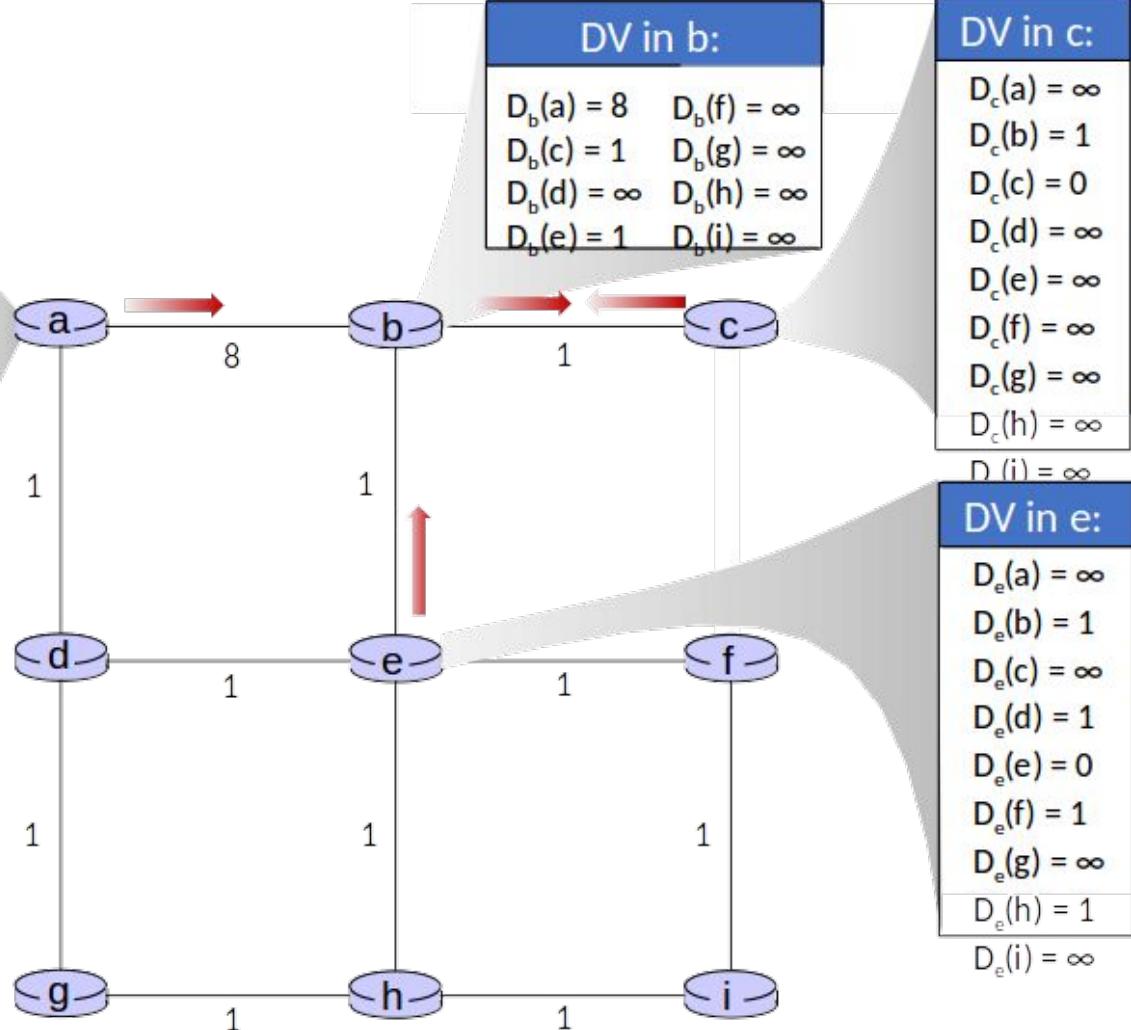
$$\begin{aligned}
 D_b(a) &= \min\{c_{b,a} + D_a(a), c_{b,c} + D_c(a), c_{b,e} + D_e(a)\} = \min\{8, \infty, \infty\} = 8 \\
 D_b(c) &= \min\{c_{b,a} + D_a(c), c_{b,c} + D_c(c), c_{b,e} + D_e(c)\} = \min\{\infty, 1, \infty\} = 1 \\
 D_b(d) &= \min\{c_{b,a} + D_a(d), c_{b,c} + D_c(d), c_{b,e} + D_e(d)\}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



$t=1$

- c receives DVs from b

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





$t=1$

- c receives DVs from b computes:

$$D_c(a) = \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9$$

$$D_c(b) = \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1$$

$$D_c(d) = \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty$$

$$D_c(e) = \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2$$

$$D_c(f) = \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty$$

$$D_c(g) = \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty$$

$$D_c(h) = \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty$$

$$D_c(i) = \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty$$

DV in b:

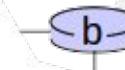
$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

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 c:

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



compute



$t=1$

- 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 b:

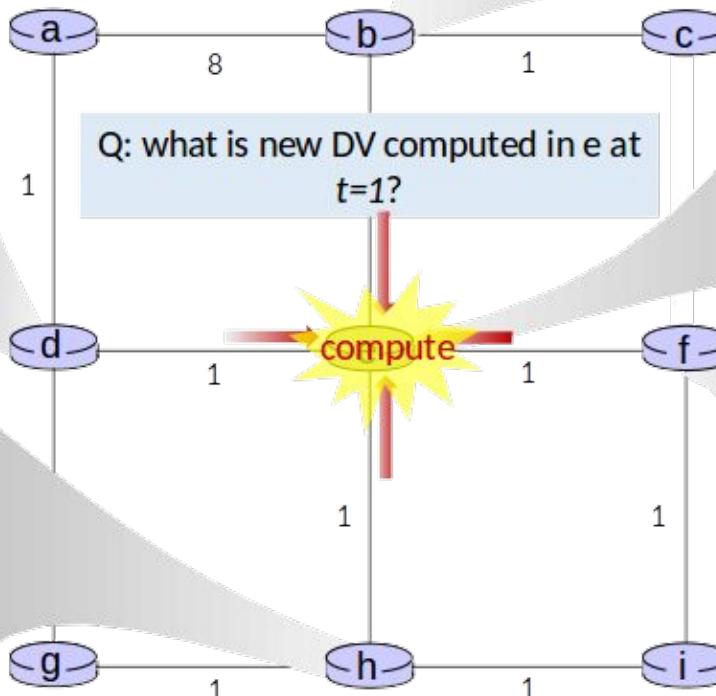
$D_b(a) = 8$ $D_b(f) = \infty$
 $D_b(c) = 1$ $D_b(g) = \infty$
 $D_b(d) = \infty$ $D_b(h) = \infty$
 $D_b(e) = 1$ $D_b(i) = \infty$

DV in e:

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

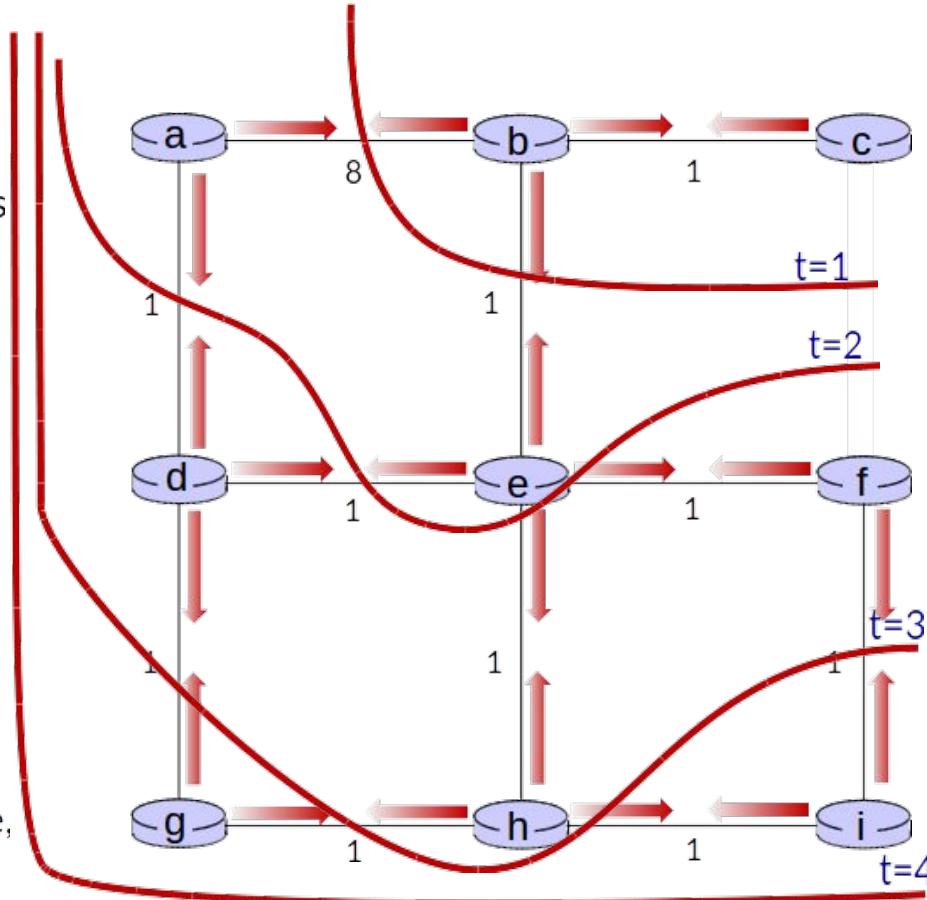
DV in f:

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



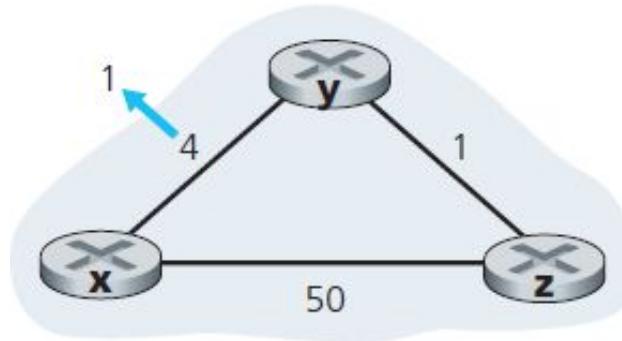
Distance vector: state information diffusion

-  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



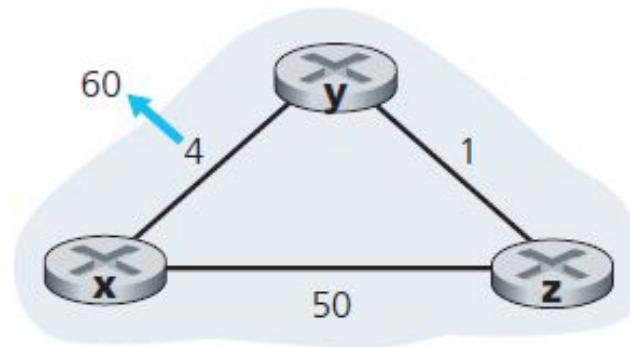
Distance vector: link cost changes

- Initially: $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, $D_z(x) = 5$
- node detects local link cost change
 - updates routing info, recalculates local DV
 - if DV changes, notify neighbors
- “good news travels fast”
 - t_0 : **y detects link-cost change, updates its DV, informs its neighbors.**
 - $D_y(x) = \min \{c_{y,x} + D_x(x), c_{y,z} + D_z(x)\} = \min \{1 + 0, 1+5\} = 1$
 - Node z also receives notification from x but its DV does not change
 - t_1 : **z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV.**
 - $D_z(x) = \min \{c_{z,x} + D_x(x), c_{z,y} + D_y(x)\} = \min \{50 + 0, 1+1\} = 2$
 - t_2 : **y receives z's update, updates its distance table. y's least costs do not change, so y does not send a message to z.**
 - X also receives z's update but its DV does not change



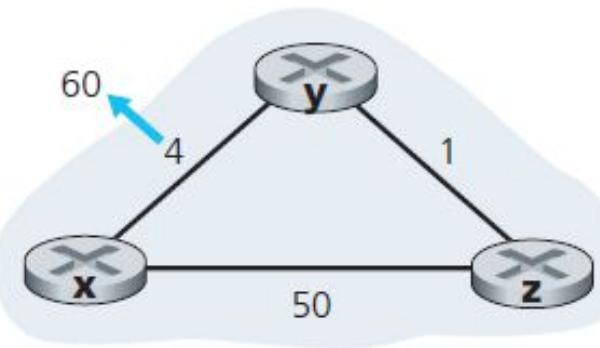
Distance vector: link cost changes

- Initially: $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, $D_z(x) = 5$
- Node detect local link cost change: update
 - y sees direct link to x has new cost 60, but z has said it has a path at cost of 5. So y computes “my new cost to x will be 6, via z); notifies z of new cost of 6 to x.
 - $D_y(x) = \min \{c_{y,x} + D_x(x), c_{y,z} + D_z(x)\} = \min \{60 + 0, 1+5\} = 6$
 - notify z: z updates its distance vector
 - z learns that path to x via y has new cost 6, so z computes “my new cost to x will be 7 via y), notifies y of new cost of 7 to x.
 - $D_z(x) = \min\{ 50+0, 1+6 \} = 7$
 - y learns that path to x via z has new cost 7, so y computes “my new cost to x will be 8 via y), notifies z of new cost of 8 to x.
 - z learns that path to x via y has new cost 8, so z computes “my new cost to x will be 9 via y), notifies y of new cost of 9 to x.
 - 44 iterations before algorithm stabilizes
 - A routing loop: in order to get to x, y routes through z, z routes through y



Distance vector: link cost changes

- **bad news travels 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)



Comparison of LS and DV algorithms

- **message complexity**
 - LS: Each node needs to know the cost of each link in the network
 - with n nodes, E links, $O(nE)$ msgs sent
 - A link cost change is sent to all nodes
 - DV: exchange between neighbors only
 - convergence time varies
 - The link cost change is propagated only if the new link cost results in a changed least-cost path for one of the nodes attached to that link
- **speed of convergence**
 - LS: $O(n^2)$ algorithm requires $O(nE)$ msgs
 - may have oscillations
 - DV: convergence time varies
 - may have routing loops
 - count-to-infinity problem

Comparison of LS and DV algorithms

- **robustness:** what happens if router malfunctions?
 - LS:
 - a router could advertise incorrect link cost
 - a router could corrupt or drop any packets it received as part of an LS broadcast
 - each node computes only its own table
 - router calculations are somewhat separated, providing more robustness
 - DV:
 - DV node can advertise incorrect path cost to any or all destinations
 - “I have a really low-cost path to everywhere”
 - “I have a zero-cost path to major ISP”
 - each node’s table used by others
 - error propagate thru network

Outline

- introduction
- routing protocols
 - link state
 - distance vector
- **intra-AS routing in the Internet: OSPF**
- routing among the ISPs: BGP
- The SDN control plane
- ICMP: The Internet Control Message Protocol
- Network management and SNMP

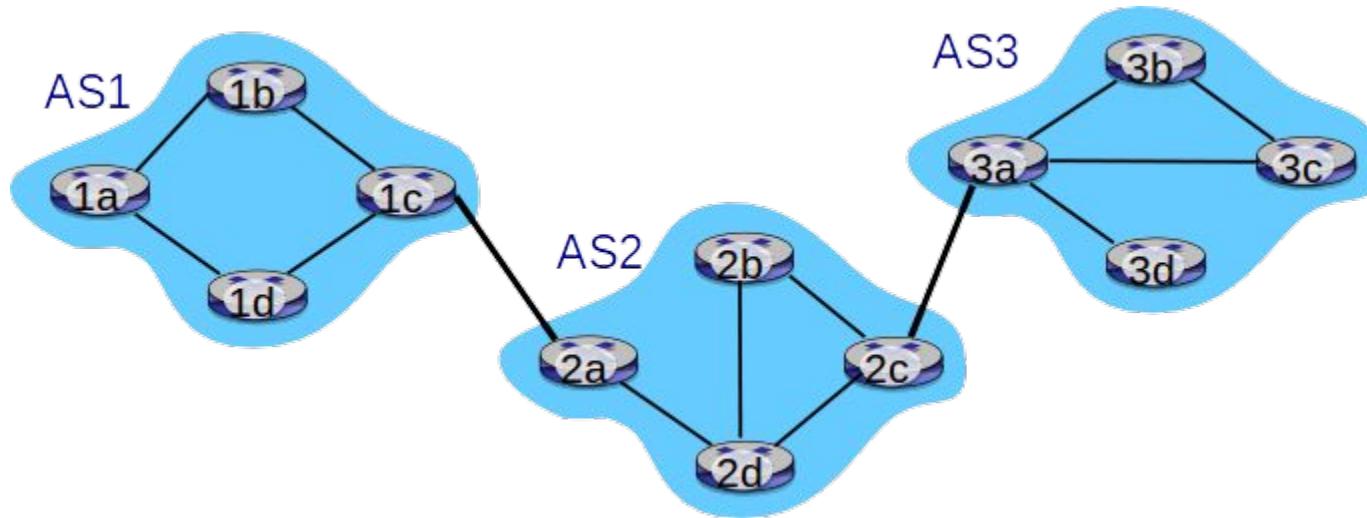
Making routing scalable

- our routing study thus far was idealized
 - all routers identical
 - network “flat”
- ... not true in practice
- **scale:** with billions of destinations:
 - can't store all destinations in routing tables!
 - The overhead of routing table exchange (broadcast connectivity and link states) would be huge
- **administrative autonomy**
 - internet = network of networks
 - each network admin may want to control routing in its own network

Internet approach to scalable routing

- aggregate routers into regions known as “**autonomous systems**” (AS) (a.k.a. “**domains**”)
 - each AS consisting of a group of routers that are under the same administrative control
 - An autonomous system is identified by its globally unique autonomous system number (ASN)
 - AS numbers, like IP addresses are assigned by ICANN registries

Internet approach to scalable routing



gateway router: a router on the edge of an AS that directly connects to one or more routers in other ASs.

internal router connects only to hosts and routers within its own AS.

In AS1, router 1c is a gateway router; routers 1a, 1b, and 1d are internal routers

Internet approach to scalable routing

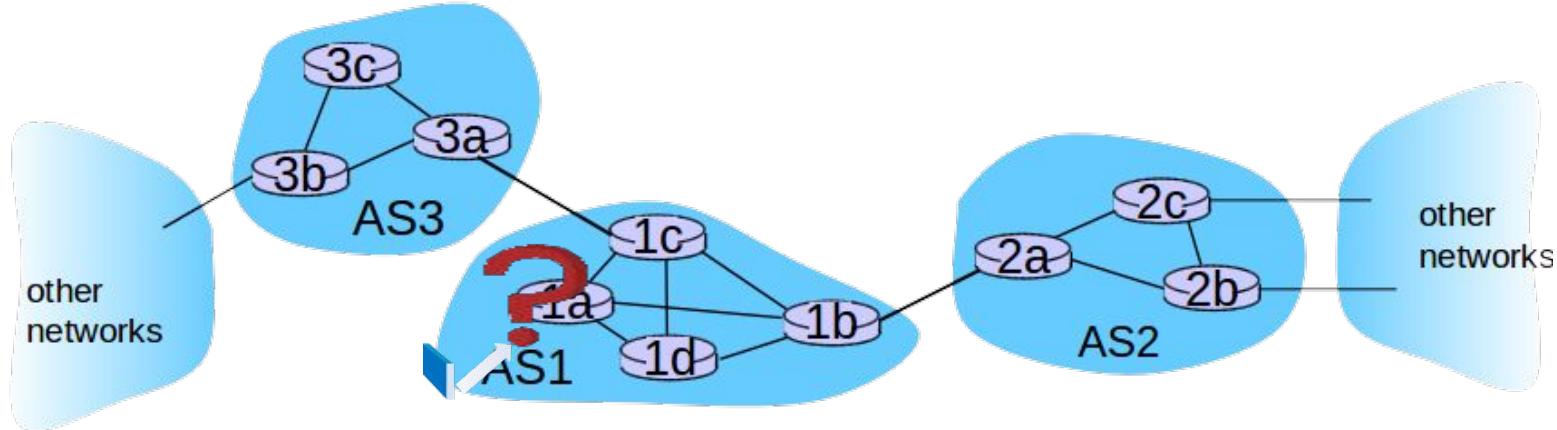
- intra-AS routing
 - routing among hosts, routers in same **AS**
 - all routers in AS must run same intra-domain routing protocol
 - routers in different AS can run different intra-domain routing protocol
- inter-AS routing
 - routing **among** AS'es
 - **gateways** perform inter-domain routing as well as intra-domain routing

Solves the issue of **autonomy** and **scale**

Inter-AS tasks



- suppose router in AS1 receives datagram destined outside of AS1:
- router should forward packet to gateway router, but which one?
- **AS1 must:**
 - learn which destinations are reachable through AS2, which through AS3
 - propagate this reachability info to all routers in AS1
 - job of inter-AS routing!



Interconnected ASes

- forwarding table
configured by both intra-
and inter-AS routing
algorithm
 - intra-AS routing determine
entries for destinations
within AS
 - inter-AS and intra-A
determine entries for
external destinations



Intra-AS Routing

- most common **intra-AS** routing protocols:
 - RIP: Routing Information Protocol [RFC 1723]
 - classic DV: DVs exchanged every 30 secs
 - no longer widely used
 - EIGRP: Enhanced Interior Gateway Routing Protocol
 - DV based
 - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
 - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF)
 - link-state routing
 - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

OSPF (Open Shortest Path First) (RFC2328)

- “open”: routing protocol specification is publicly available
- uses link-state algorithm
 - router floods OSPF link-state advertisements to all other routers in **entire AS**
 - carried in OSPF messages directly over IP (rather than TCP or UDP)
 - *upper-layer protocol* field in IP datagram is 89 for OSPF
 - each router has full topology, uses Dijkstra’s algorithm to compute forwarding table
 - multiple link costs metrics possible: bandwidth, delay
 - Link costs is set by admin

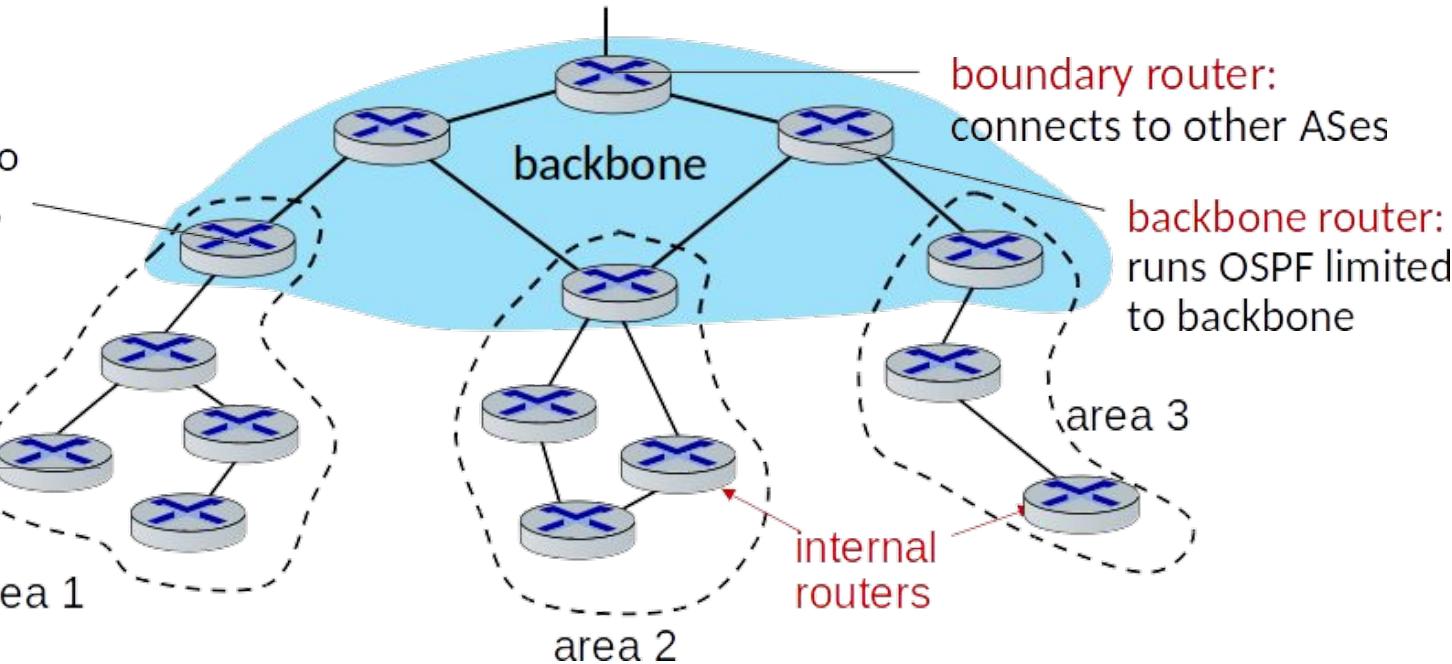
OSPF “advanced” features

- **security:** all OSPF messages authenticated (to prevent malicious intrusion)
- **hierarchical OSPF in large domains.**
 - An OSPF autonomous system can be configured hierarchically into several areas

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

area border routers:
“summarize” distances to
destinations in own area,
advertise in backbone



Outline

- introduction
- routing protocols
 - link state
 - distance vector
- intra-AS routing in the Internet: OSPF
- **routing among the ISPs: BGP**
- The SDN control plane
- ICMP: The Internet Control Message Protocol
- Network management and SNMP

Internet inter-AS routing: BGP

- **BGP (Border Gateway Protocol)**: the de facto inter-domain routing protocol
 - “glue that holds the Internet together”
 - decentralized and asynchronous protocol similar to distance-vector routing
- Two main tasks of BGP:
 - Finding least-cost path from source to destination
 - How BGP allows network operator to implement routing policy
 - To control how packets that are routed to/from customer's network are controlled and how a network handle control transit.
 - BGP is as much about policy as it is about finding the best path

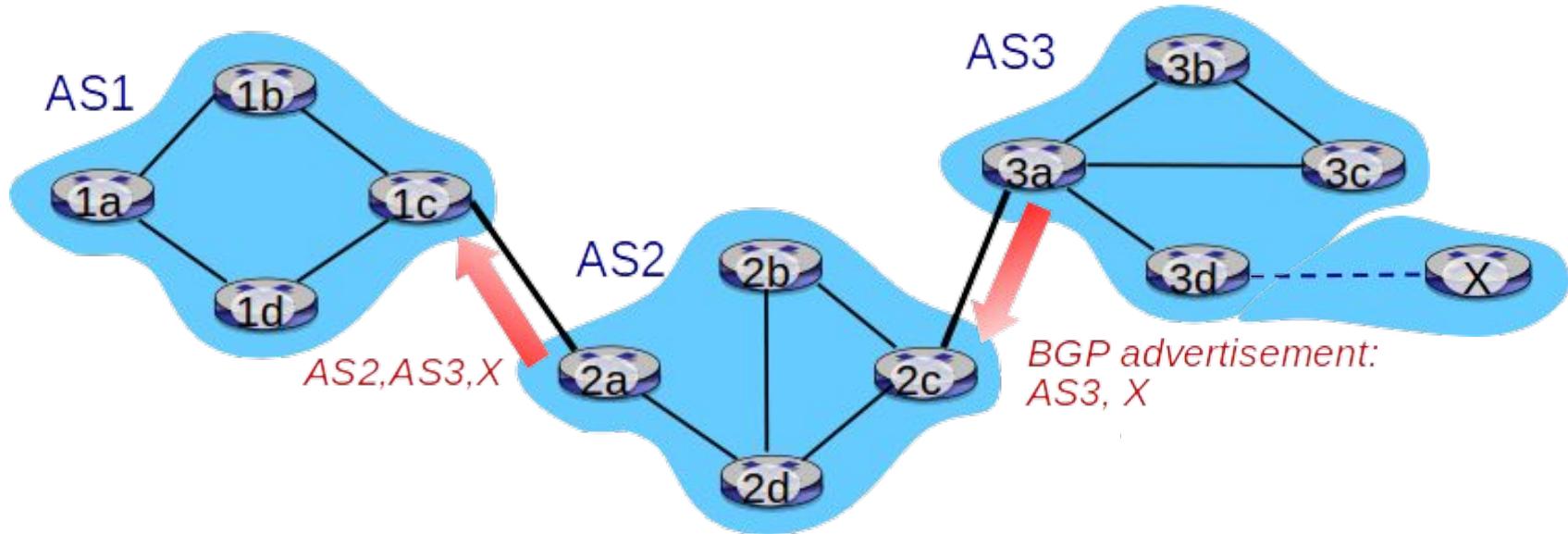
Internet inter-AS routing: BGP

- **BGP (Border Gateway Protocol)**: the de facto inter-domain routing protocol
 - “glue that holds the Internet together”
 - decentralized and asynchronous protocol similar to distance-vector routing
- BGP overview:
 - **Path advertisement**
 - Each AS (gateway routers in the subnet) **advertises** its existence, and the destinations it can reach to the rest of the Internet.
 - “I am here, here is who I can reach, and how”
 - BGP makes sure all the routes in the Internet know about this subnet
 - Each AS obtain destination network reachability information from neighboring ASes
 - **determine “good” routes to other networks based on reachability information and policy**
 - propagate reachability information to all AS-internal routers (within its network)
 - **advertise** (to neighboring networks) destination reachability information

BGP

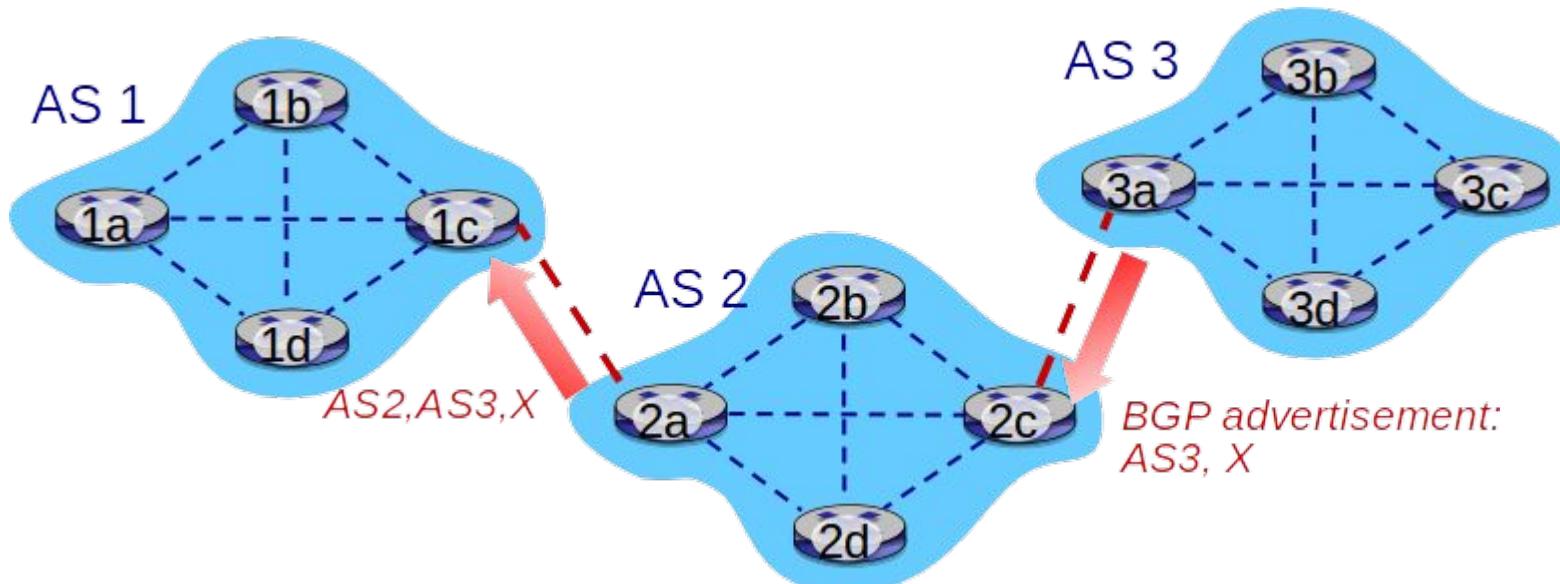
- Two main phases in BGP are
 - path advertisement
 - determining best route

BGP Path advertisement



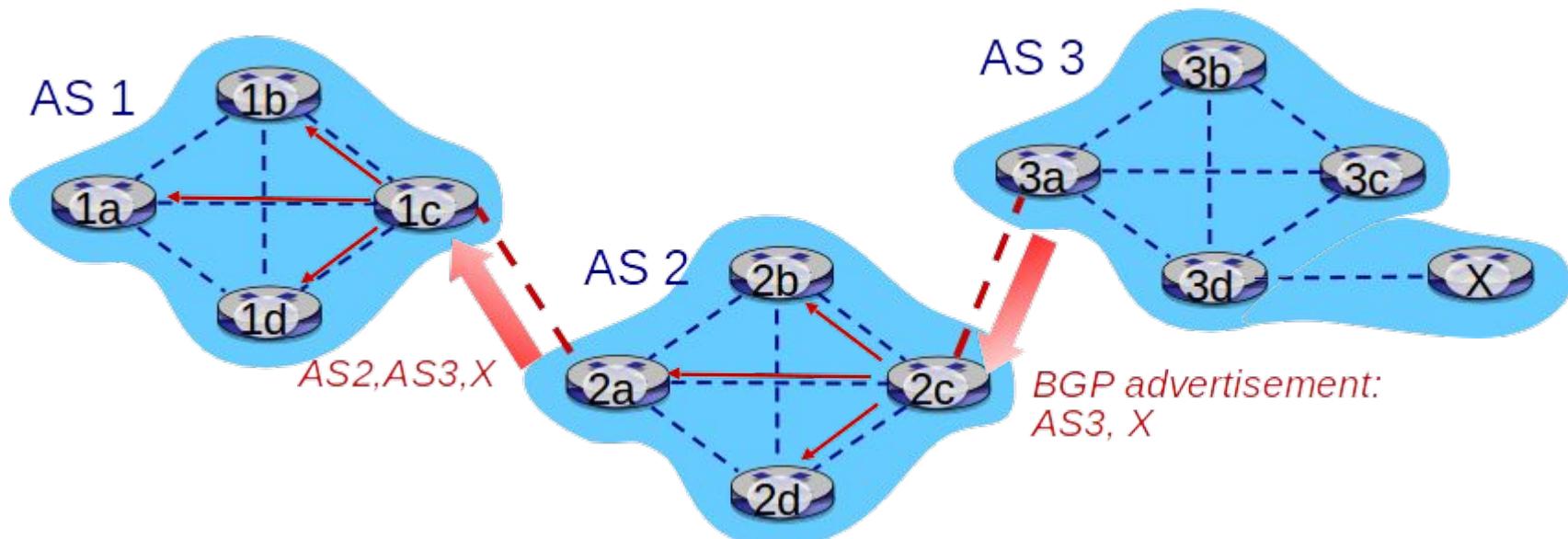
BGP Path advertisement

- In BGP, routers exchange BGP messages over **BGP sessions**
 - **BGP sessions** is a TCP connection using port 179 along with all the BGP messages sent over the connection
 - **eBGP**: a BGP connection that spans two ASs is called an **external BGP (eBGP)** connection
 - **iBGP**: a BGP session between routers in the same AS is called an **internal BGP** connection



BGP Path advertisement

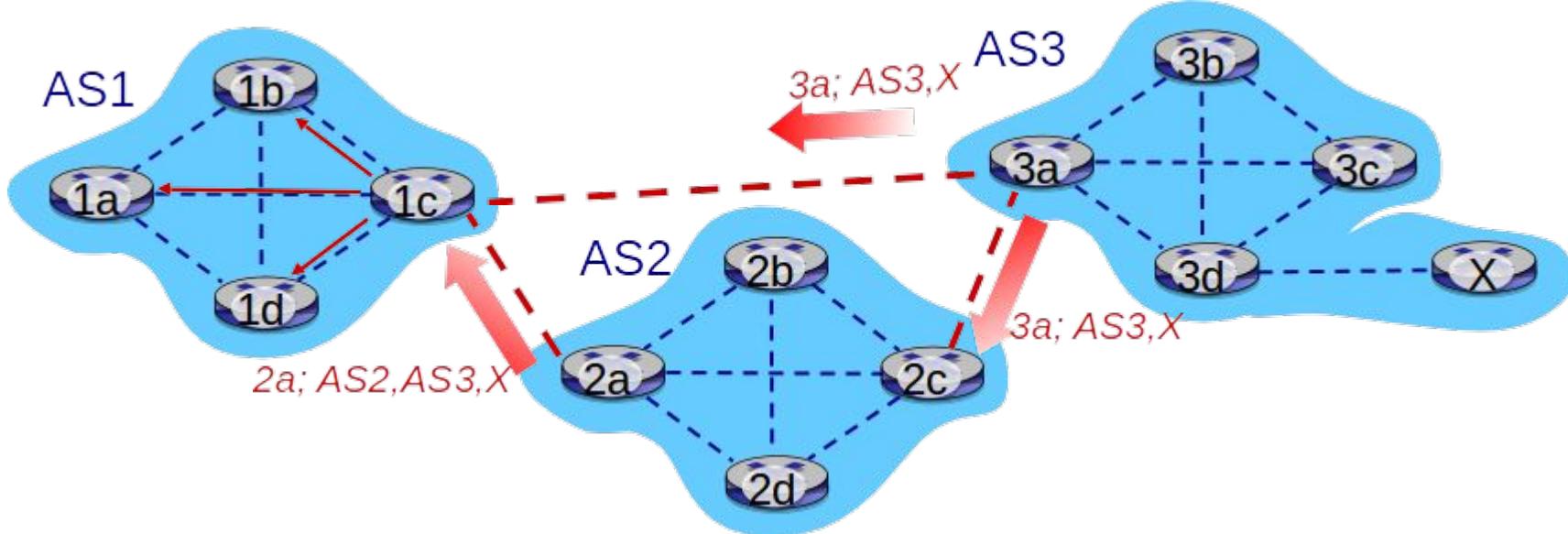
- when AS3 gateway router 3a advertises path **AS3,X** to AS2 gateway router 2c:
 - AS3 promises to AS2 it will forward datagrams towards X



BGP Path advertisement: Path attributes and BGP routes

- BGP advertised route includes:
 - **prefix + attributes = “route”**
- 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 **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 path advertisement



- gateway routers 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 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

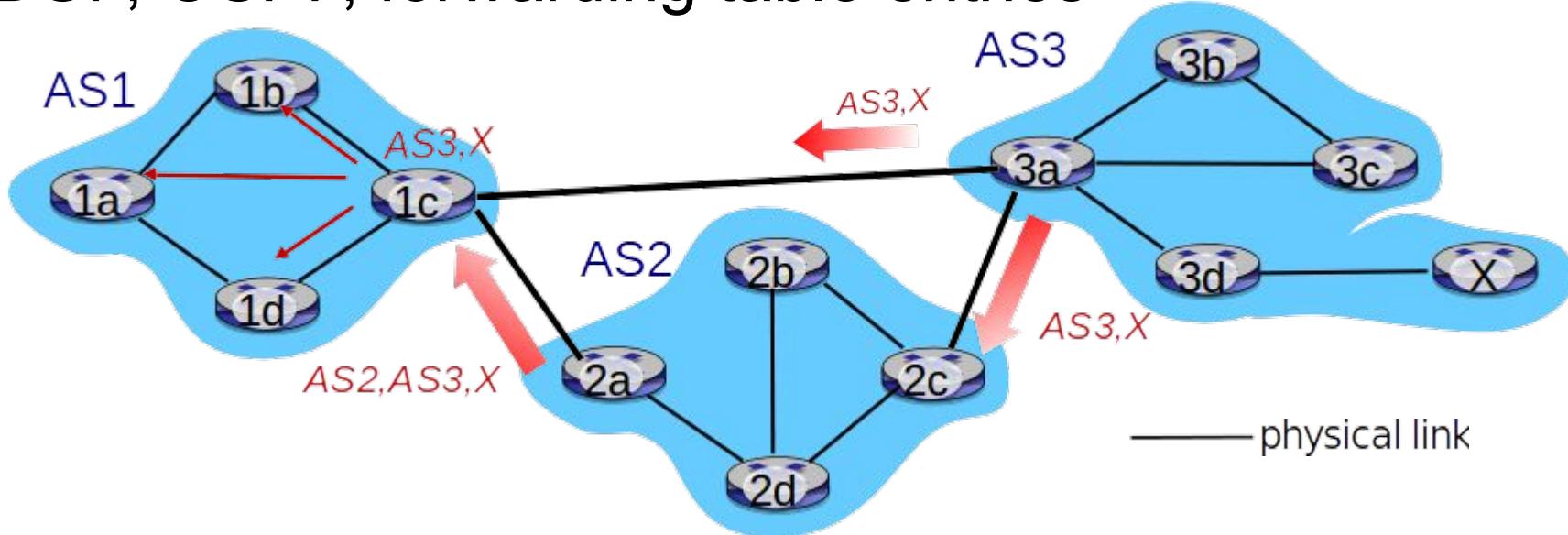
BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

Internet inter-AS routing: BGP

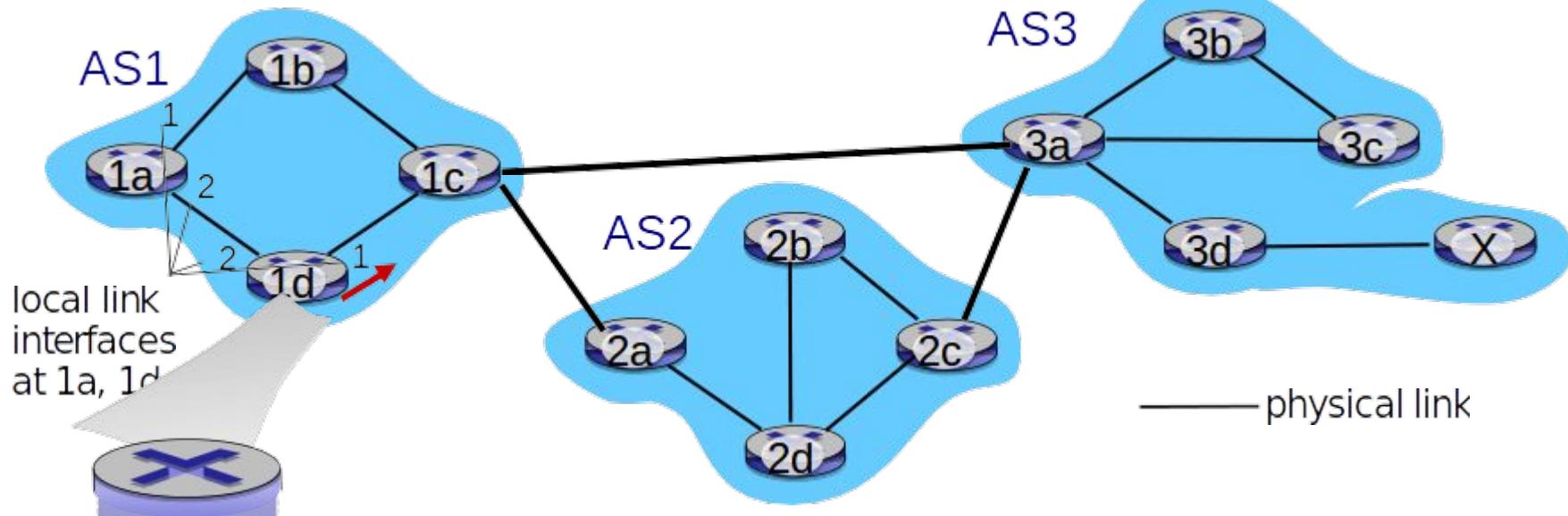
- **BGP (Border Gateway Protocol):**
 - In BGP, packets are *not* routed to a specific destination address, but instead to CIDRized prefixes, with each prefix representing a subnet or a collection of subnets
 - Example: a destination could be **138.16.68/22** which mean **1024** IP addresses
 - A router's forwarding table have entries of the form **(x,I)**
 - x: 138.16.68/22
 - I: one of router's interface numbers

BGP, OSPF, forwarding table entries



- recall: 1a, 1b, 1d learn about dest X via iBGP from 1c: “path to X goes through 1c”

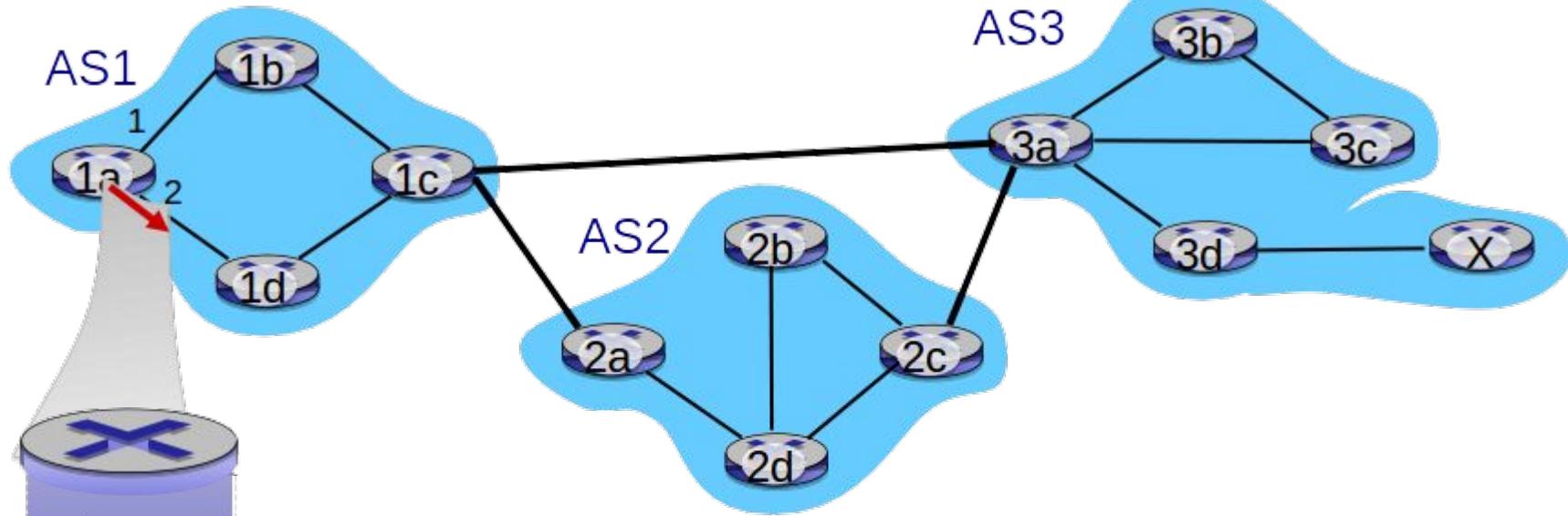
BGP, OSPF, forwarding table entries



dest	interface
...	...
1c	1
X	1
...	...

- recall: 1a, 1b, 1d learn about dest X 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, OSPF, forwarding table entries



dest	interface
...	...
1c	2
X	2
...	...

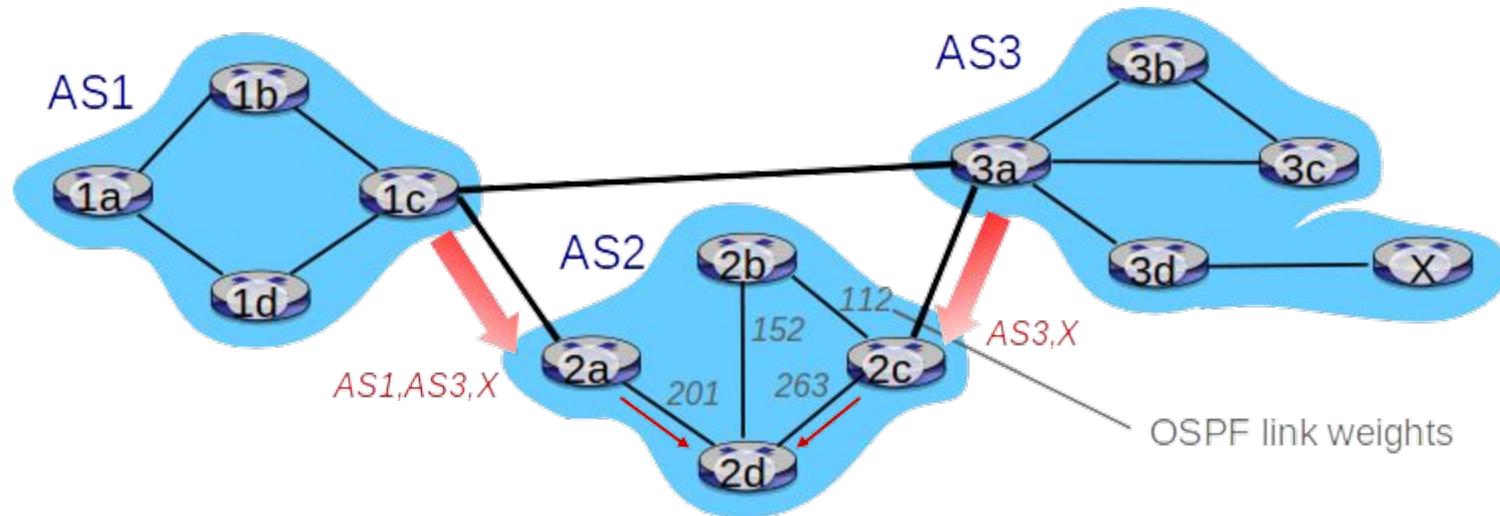
- recall: 1a, 1b, 1d learn about dest X via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1
- at 1a: OSPF intra-domain routing: to get to 1c, use interface 2
- at 1a: to get to X, use interface 2

BGP

- Two main phases in BGP are
 - path advertisement
 - determining best route

Hot Potato Routing

- 2d learns (via iBGP) it can route to X via 2a or 2c
- **hot potato routing:** choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!



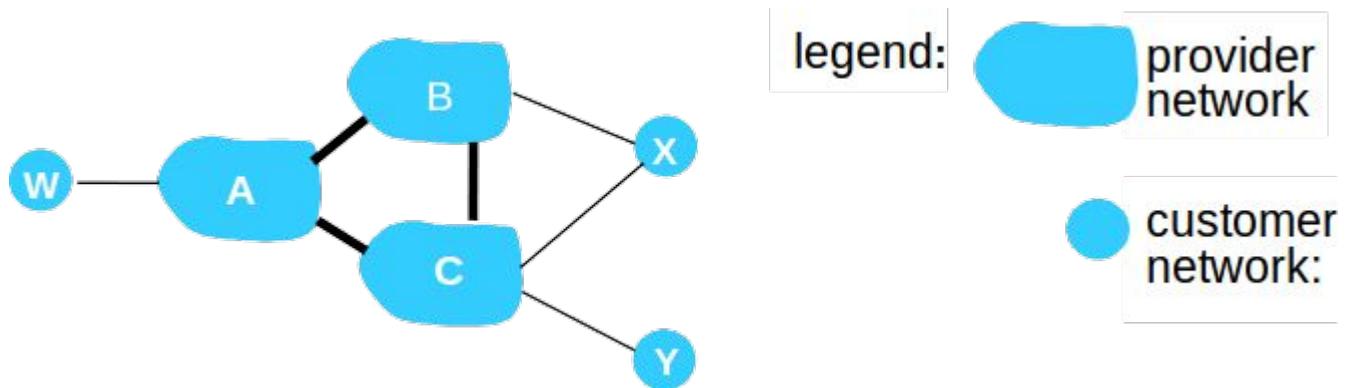
BGP Route Selection

router may learn about more than one route to destination AS, selects route based on:

- 1) local preference value attribute: policy decision
 - a) assigned to a route by the AS system admin
 - b) The routes with the highest local preference values are selected
- 2) shortest AS-PATH
- 3) Route with closest NEXT-HOP router: hot potato routing
- 4) additional criteria, such as BGP identifiers are used.

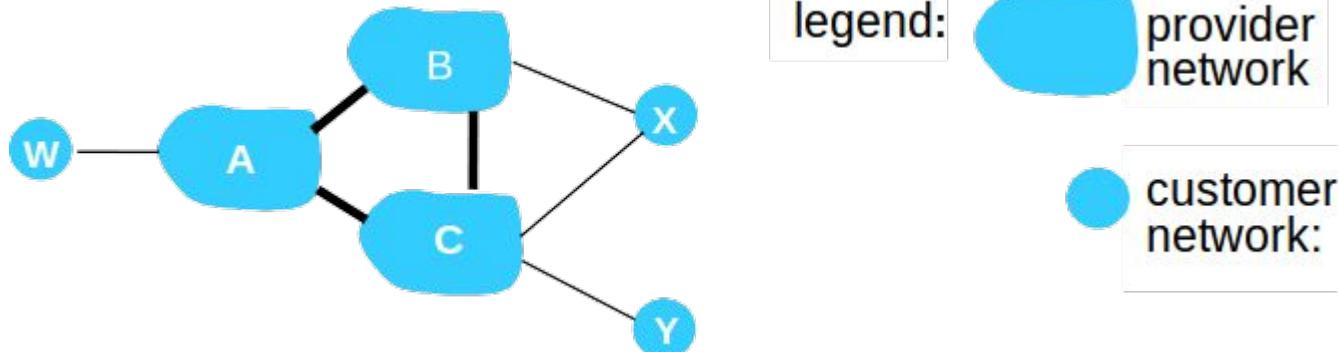
BGP: achieving policy via advertisements

- Policy: an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)
 - A,B,C are **provider networks** (They are Autonomous Systems not routers)
 - X,W,Y are customers of provider networks
 - X is **dual-homed**: attached to two networks
 - **policy to enforce**: X does not want to route from B to C via X
 - so X will not advertise to B a route to C



BGP: achieving policy via advertisements

- Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)
 - B does not want to carry traffic between A and C
 - A advertises path Aw to B and to C
 - B **chooses not to advertise** BAw to C:
 - B gets no “revenue” for routing CBAW, since none of C, A, w are B’s customers
 - C does not learn about CBAW path
 - C will route CAW (not using B) to get to w



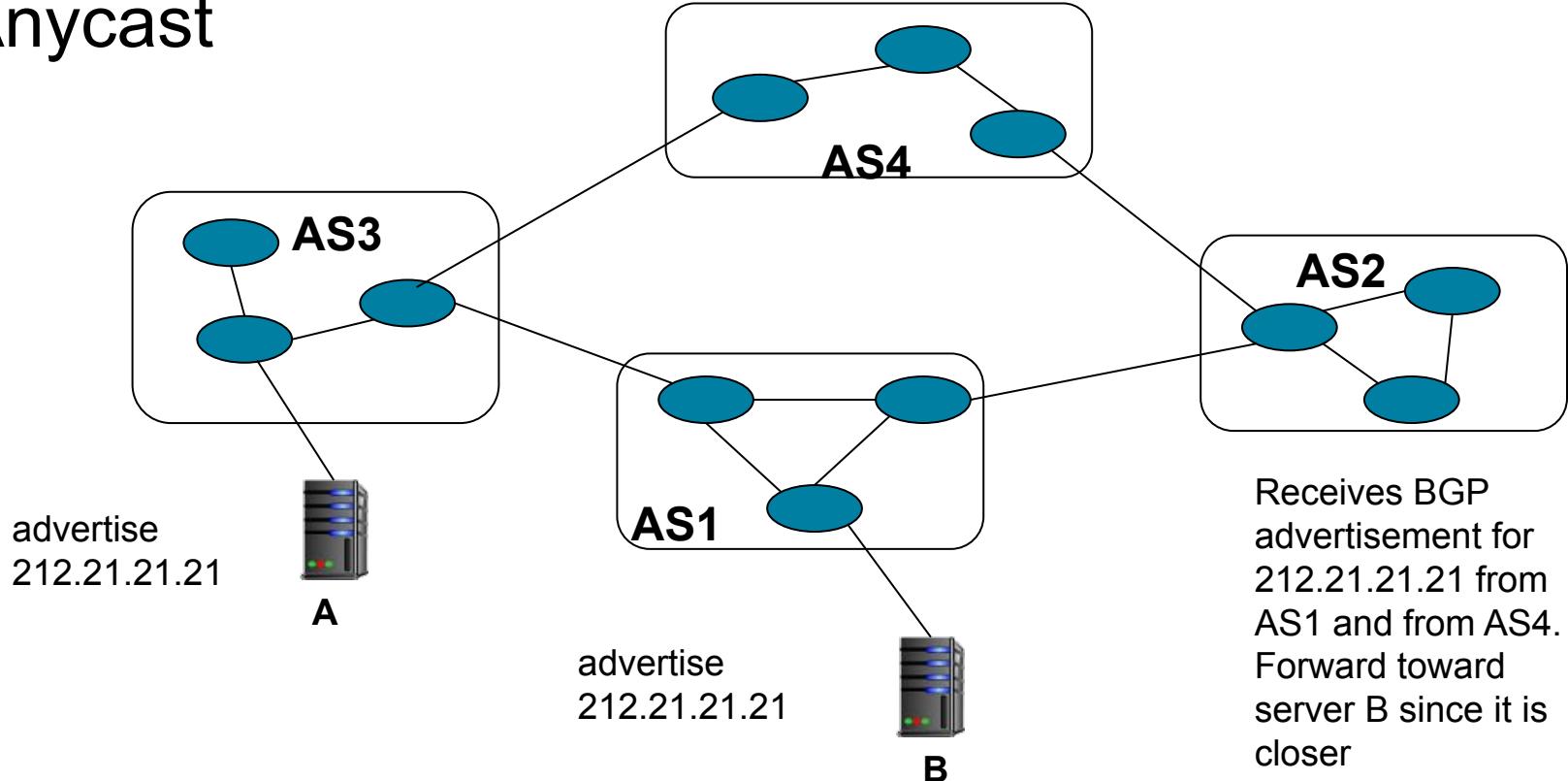
Why different Intra-AS and Inter-AS routing ?

- policy:
 - inter-AS: admin wants control over how its traffic routed, who routes through its net.
 - intra-AS: single admin, so no policy decisions needed
- Scale:
 - hierarchical routing saves table size, reduced update traffic
- performance:
 - intra-AS: can focus on performance
 - inter-AS: policy may dominate over performance

IP-Anycast

- Multiple hosts accept traffic on a single IP address.
- Motivation for IP-anycast:
 - In many applications, we want to replicate the same content on different servers in many different dispersed geographical locations
 - We want each user access the content from the server that is closest
 - Example applications:
 - CDN
 - A CDN may replicate videos and other objects on servers in different countries
 - DNS
 - The DNS system can replicate DNS records on DNS server throughout the world. When a user wants to access this replicated content, we want the user to be served from the “nearest” server

IP-Anycast

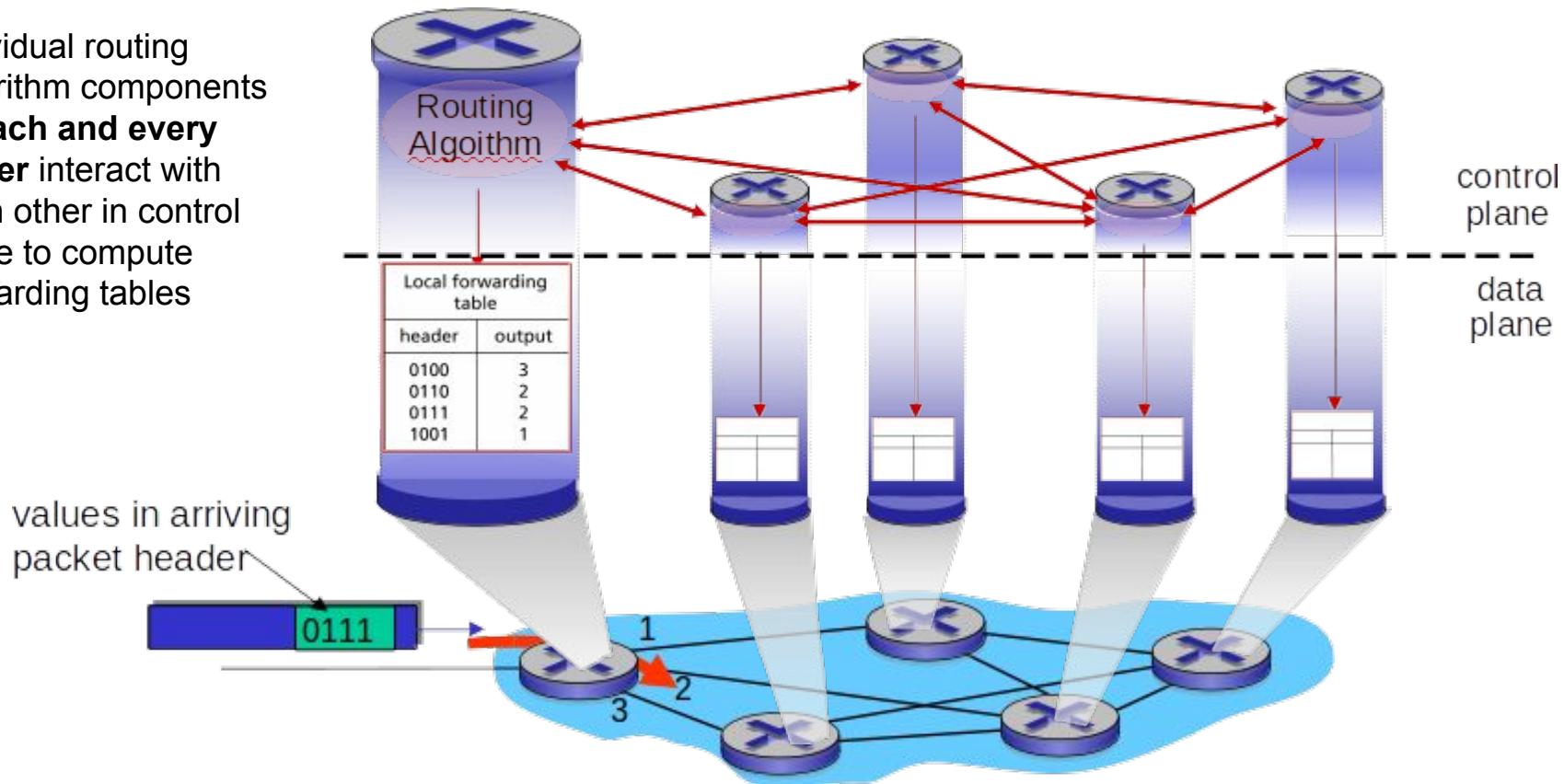


Outline

- introduction
- routing protocols
 - link state
 - distance vector
- intra-AS routing in the Internet: OSPF
- routing among the ISPs: BGP
- **The SDN control plane**
- ICMP: The Internet Control Message Protocol
- Network management and SNMP

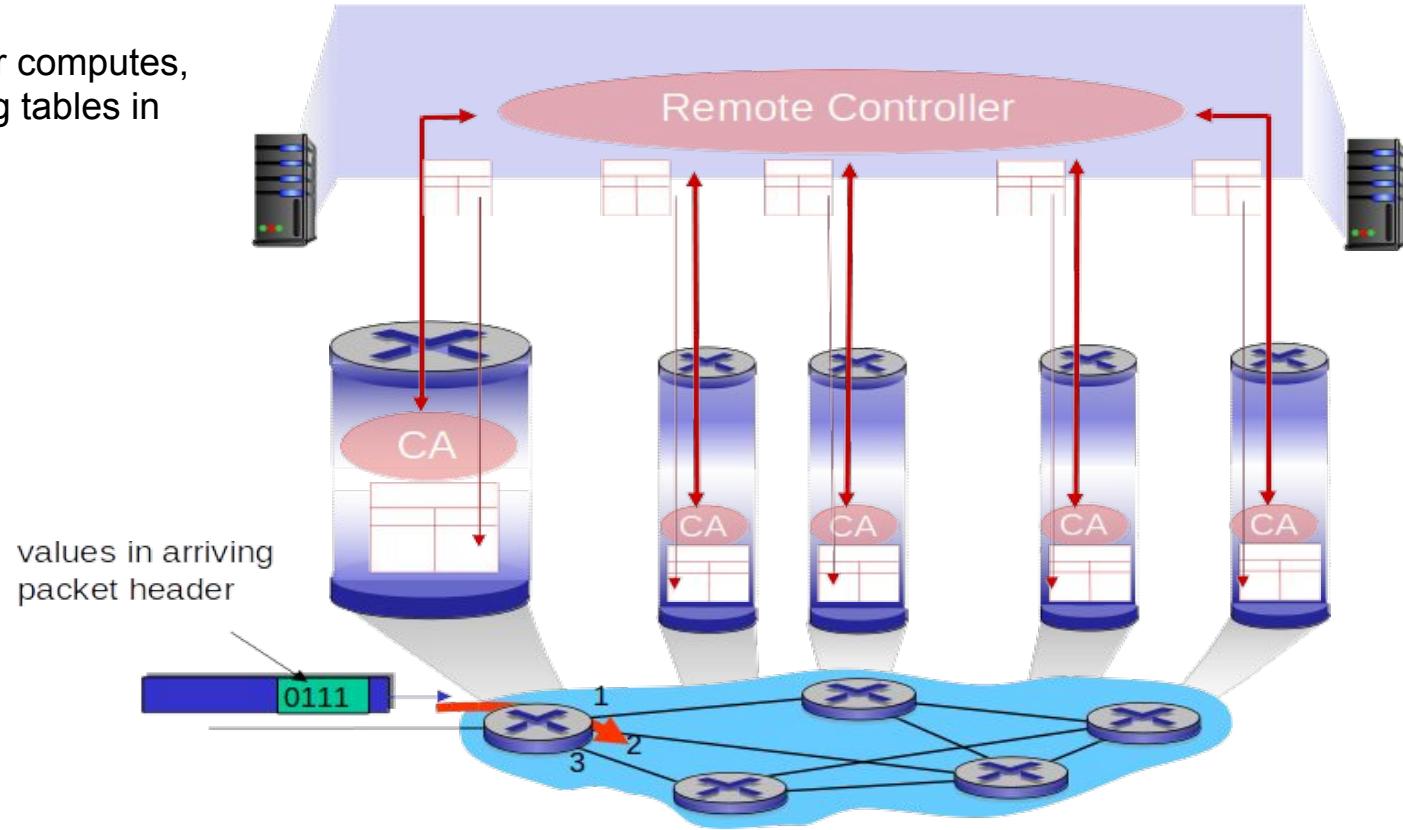
Per-router control plane

Individual routing algorithm components **in each and every router** interact with each other in control plane to compute forwarding tables



Software-Defined Networking (SDN) control plane

Remote controller computes,
installs forwarding tables in
routers



Software defined networking (SDN)

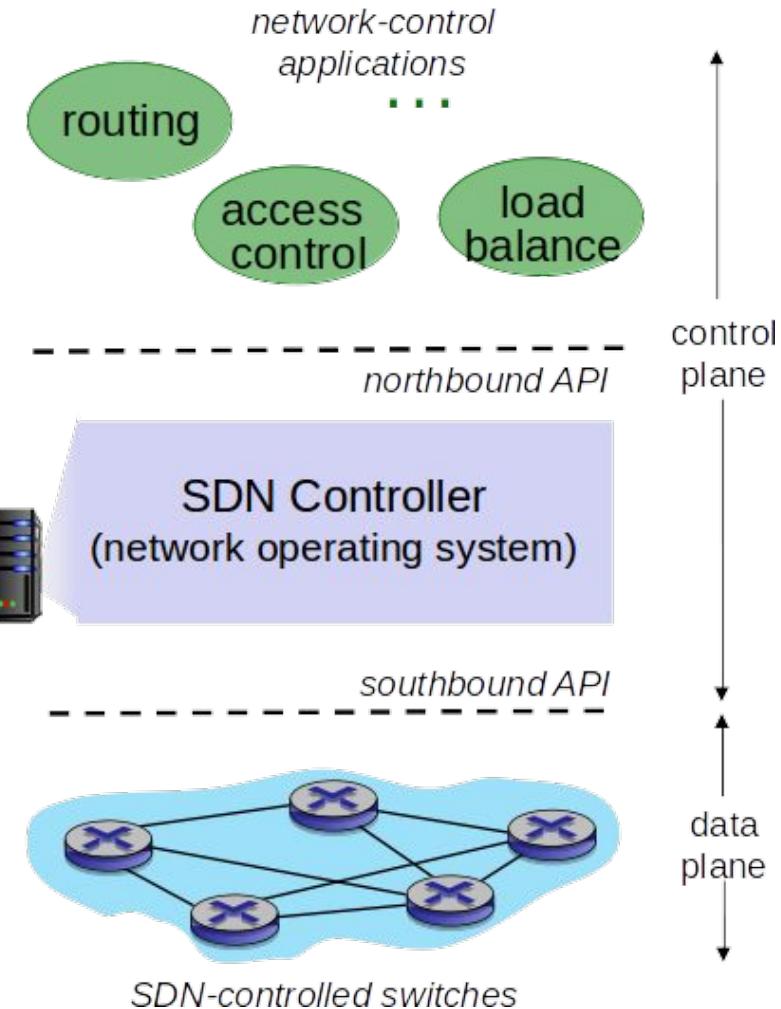
Why a logically centralized control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows “programming” routers
 - centralized “programming” easier: compute tables centrally and distribute
 - distributed “programming”: more difficult: compute tables as result of distributed algorithm (protocol) implemented in each and every router
- open (non-proprietary) implementation of control plane

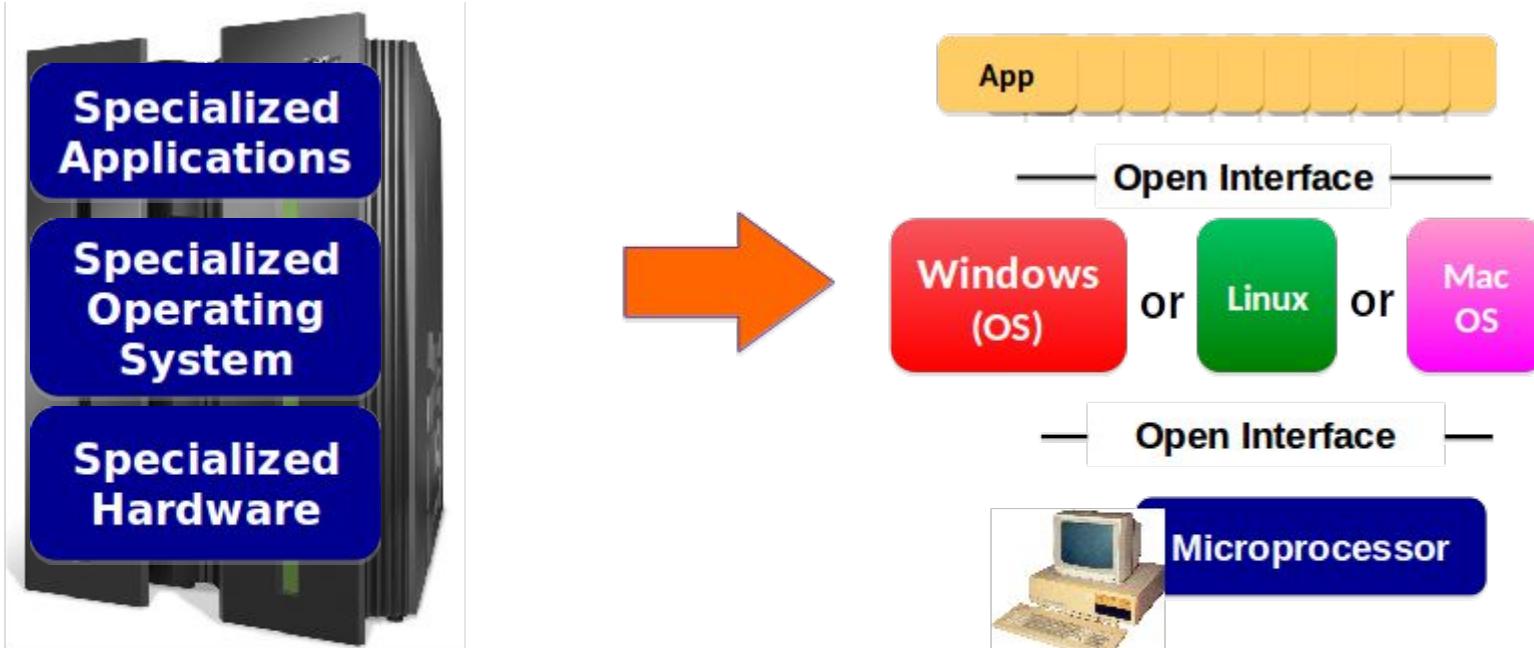
Software defined networking (SDN)

- Internet network layer: historically has been implemented via distributed, per-router approach
 - **monolithic** router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
 - different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane
- SDN represents a significant unbundling of network functionality:
 - data plane switches
 - SDN controllers
 - network-control applications
 - These are separate entities; each may be provided by different vendors and organization

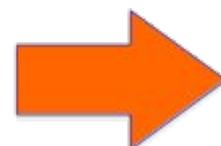
SDN perspective



Analogy: mainframe to PC evolution



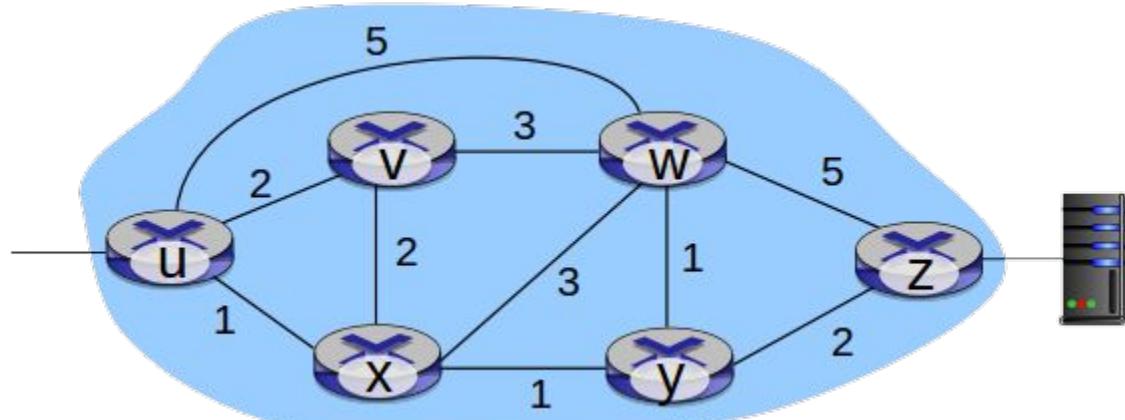
Vertically integrated
Closed, proprietary
Slow innovation
Small industry



Horizontal
Open interfaces
Rapid innovation
Huge industry

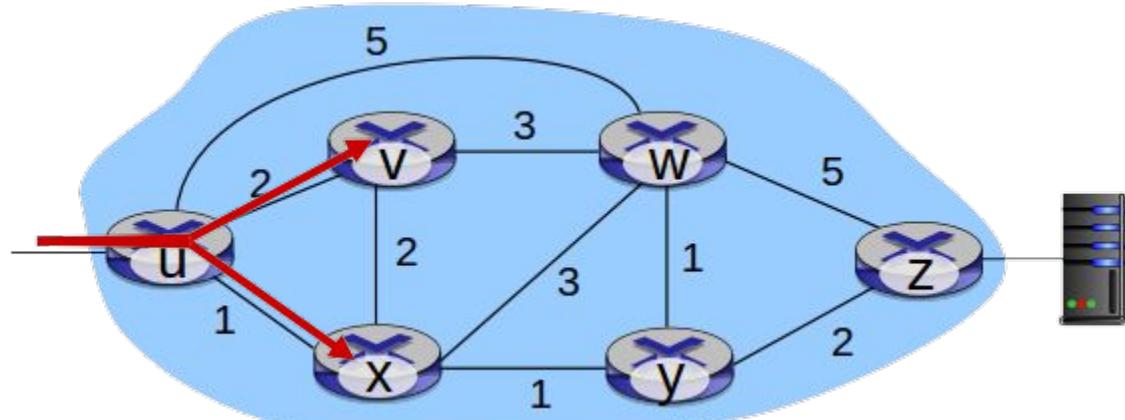
Traffic engineering: difficult traditional routing

- Q: what if network operator wants u-to-z traffic to flow along uvwz, x-to-z traffic to flow xwyz?
- A: need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!
- **Link weights are only control “knobs”: wrong!**



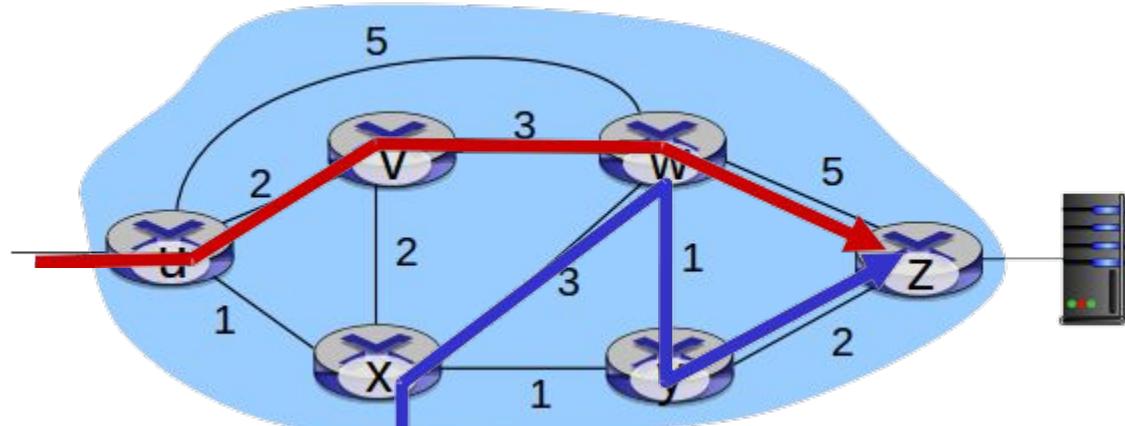
Traffic engineering: difficult traditional routing

- Q: what if network operator wants to split u-to-z traffic along uvwz and uxzy (load balancing)?
- A: can't do it (or need a new routing algorithm)



Traffic engineering: difficult traditional routing

- Q: what if w wants to route blue and red traffic differently?
- A: can't do it with destination based forwarding, and LS, DV routing



Software defined networking (SDN)

4. programmable control —
applications

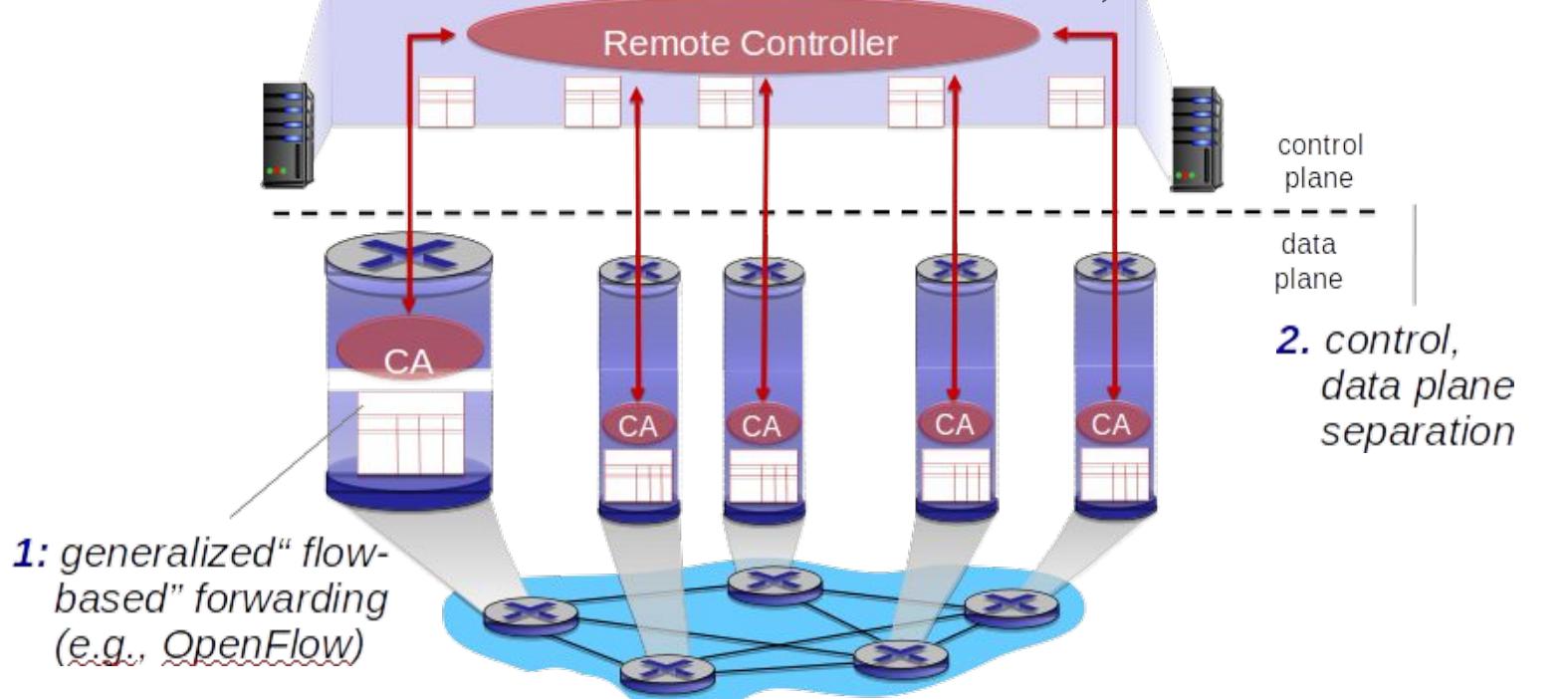
routing

access
control

...

load
balance

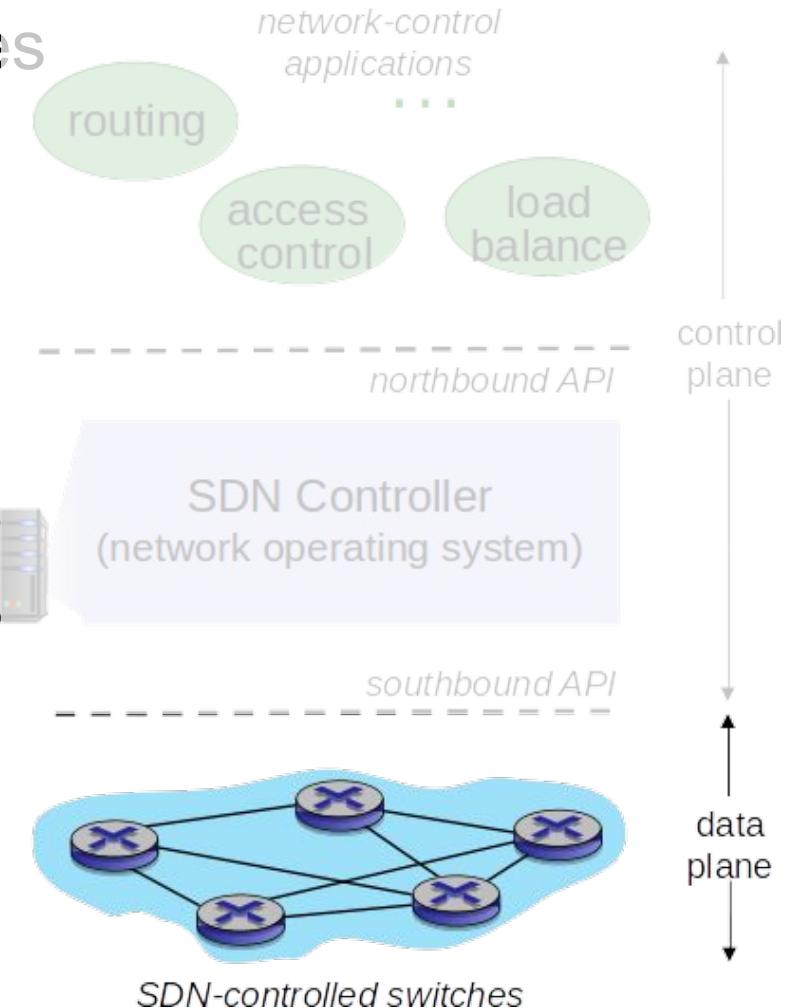
3. control plane
functions
external to data-
plane switches



SDN perspective: data plane switches

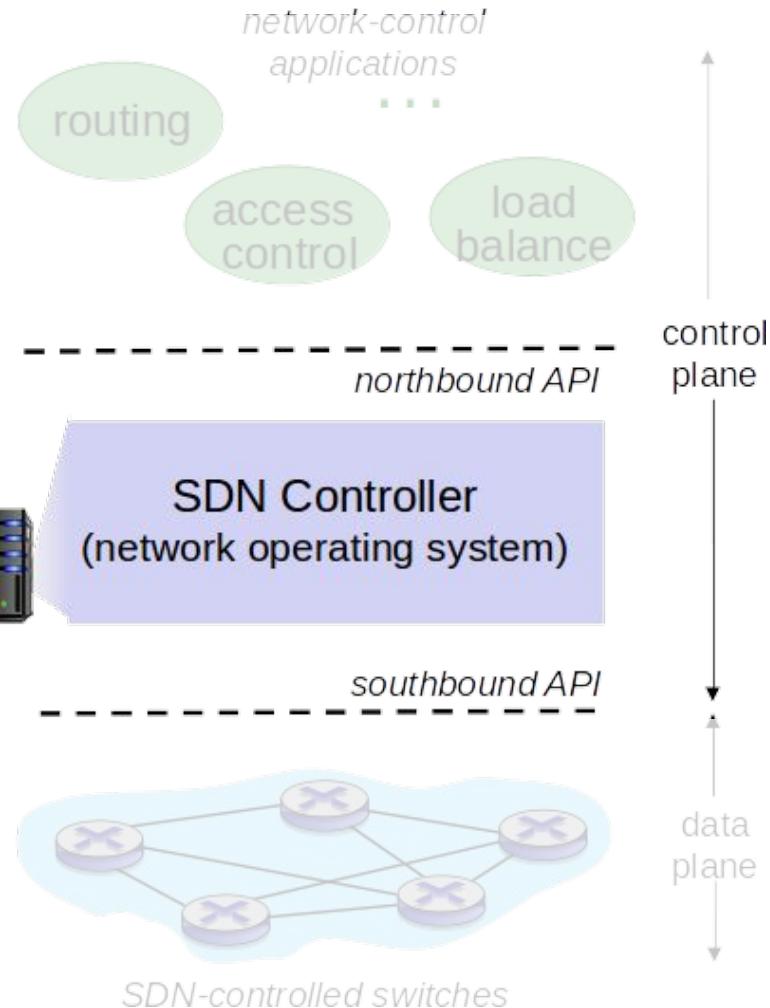
- **Data plane switches**

- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)



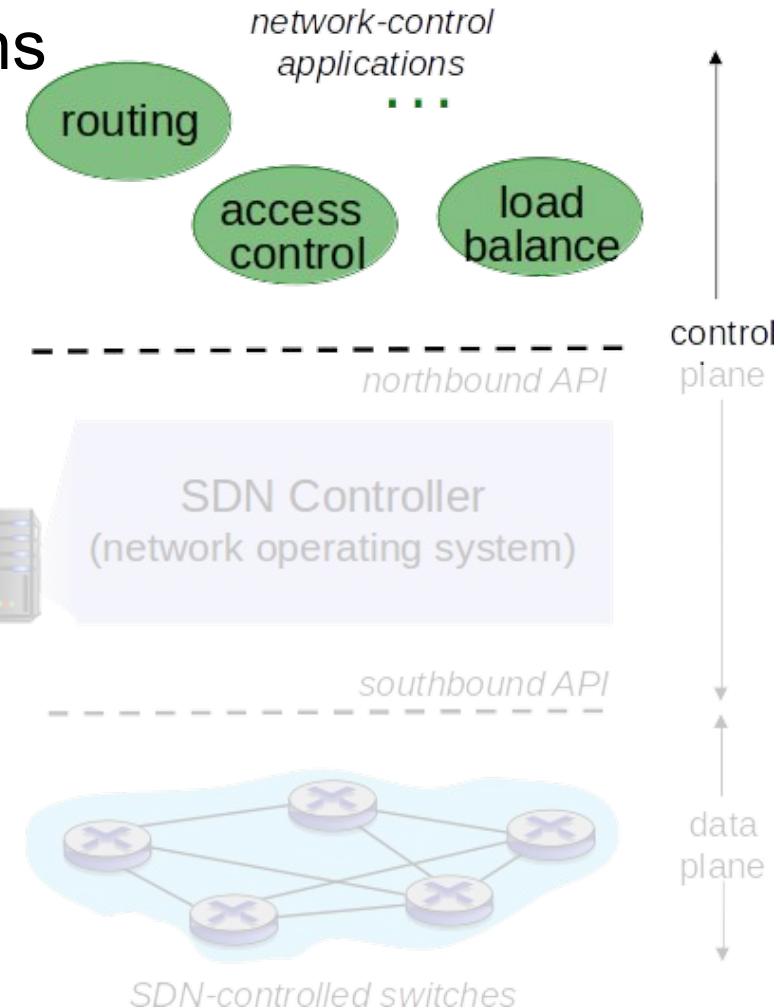
SDN perspective: SDN Controller

- **SDN controller (network OS)**
 - maintain network state information
 - interacts with network control applications “above” via northbound API
 - interacts with network switches “below” via southbound API
 - implemented as distributed system for performance, scalability, fault-tolerance, robustness



SDN perspective: control Applications

- **network-control apps**
 - “brains” of control: implement control functions using lower-level services, API provided by SDN controller
 - **unbundled**: can be provided by 3rd party: distinct from routing vendor, or SDN controller

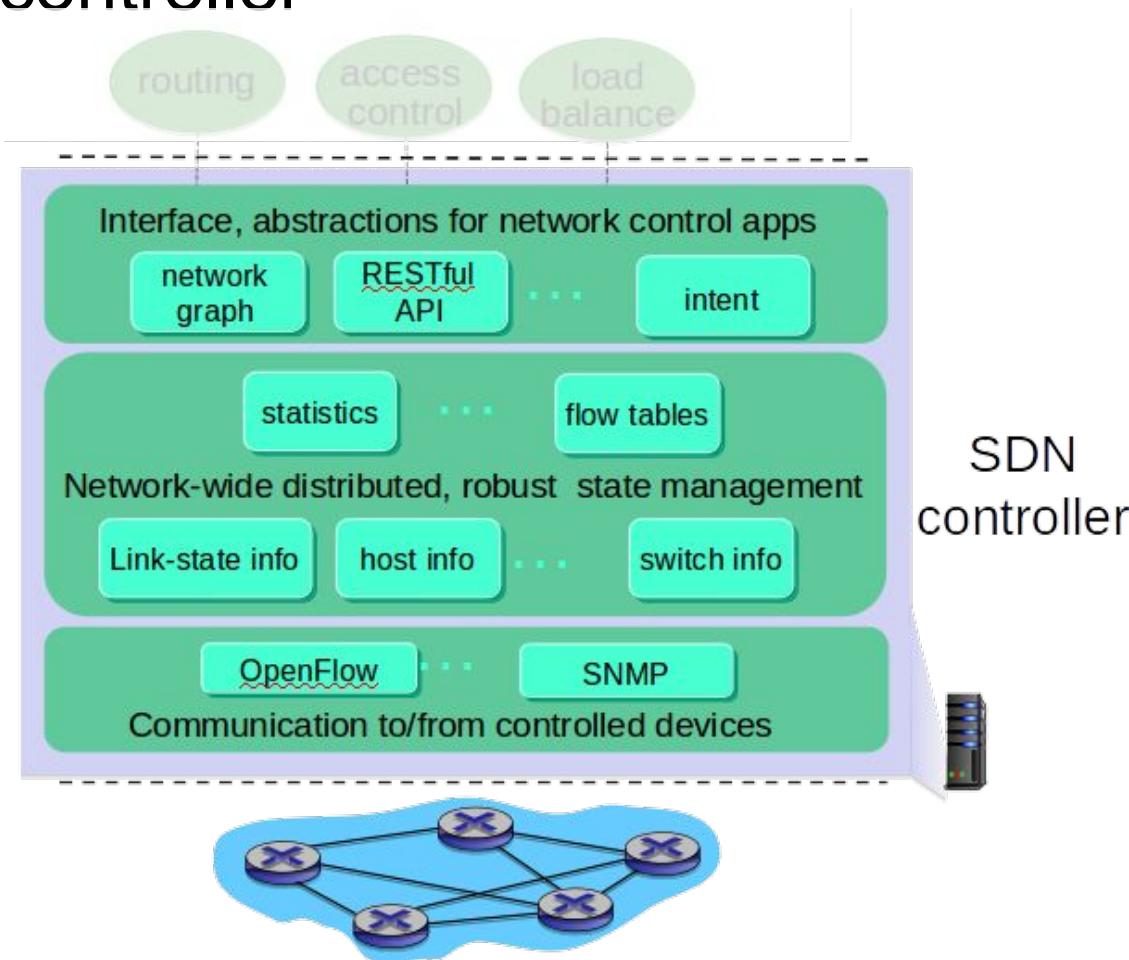


Components of SDN controller

Interface layer to network control apps: abstractions API

Network-wide state management layer: state of networks links, switches, services: a distributed database

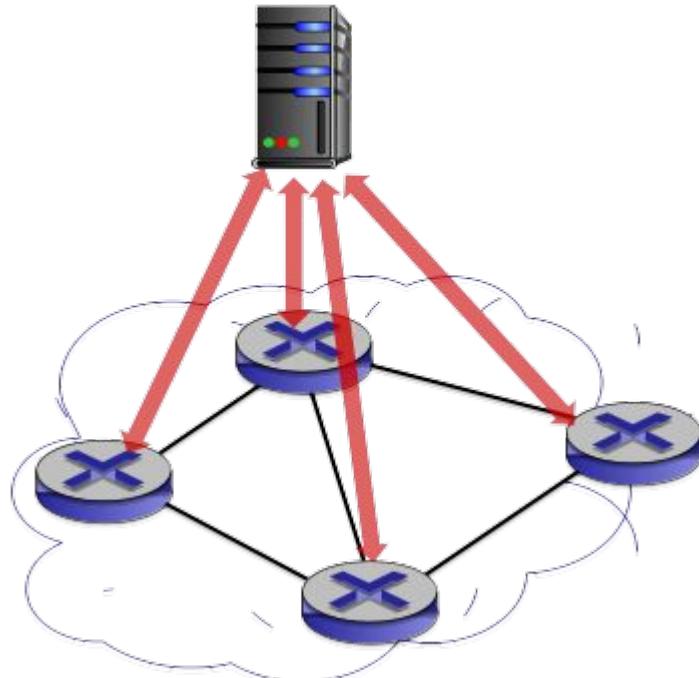
communication layer:
communicate between SDN controller and controlled switches



OpenFlow protocol

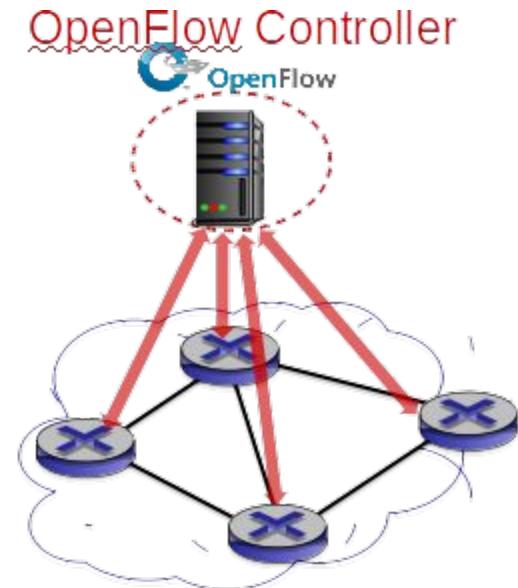
- operates between controller and switches
- TCP used to exchange messages
 - optional encryption
- three classes of OpenFlow messages:
 - Controller-to-switch
 - asynchronous (switch to controller)
 - symmetric
- distinct from OpenFlow API
 - API used to specify generalized forwarding actions

OpenFlow Controller



OpenFlow: controller-to-switch messages

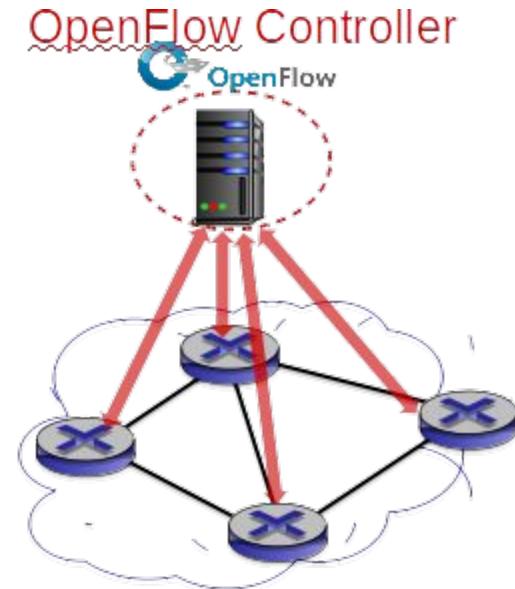
- Key controller-to-switch messages
 - **configure**: controller queries/sets switch configuration parameters
 - **Read-State**: controller collects statistics and counter values from the switch's flow table and ports
 - **modify-state**: add, delete, modify flow entries in the switches flow tables
 - **Send-packet**: controller can send this packet out of a specific port at the controlled switch.



OpenFlow: switch-to-controller messages

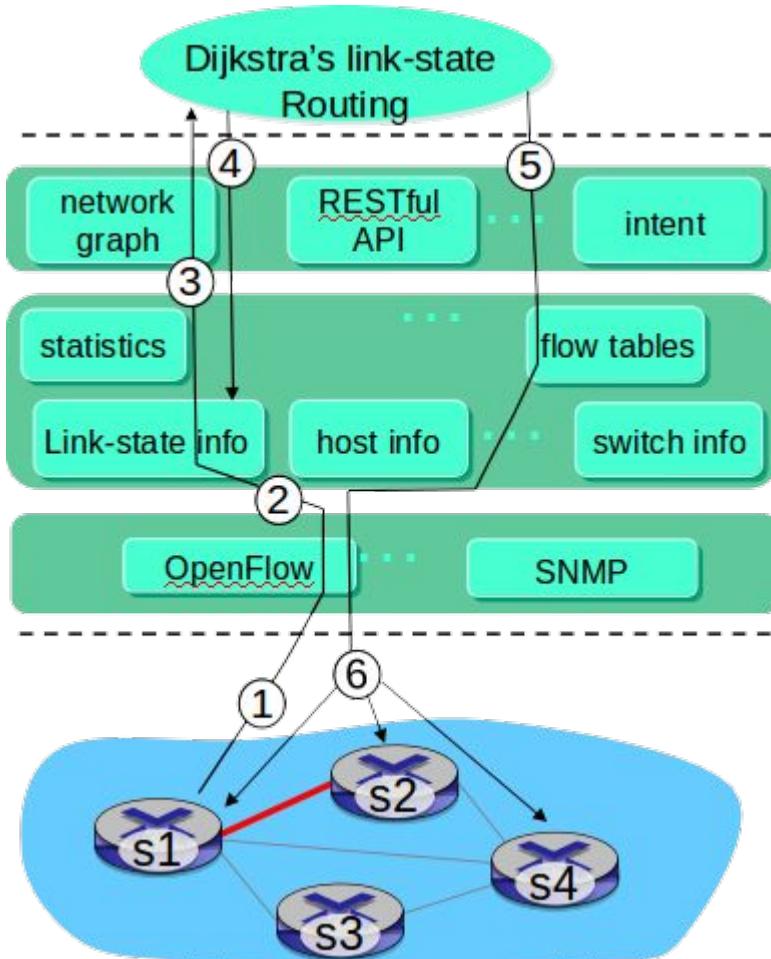
Key switch-to-controller messages

- **flow-removed**: this message informs the controller that a flow table entry has been removed (e.g., by a timeout)
- **port status**: inform controller of a change in a port status
- **packet-in**: transfer incoming packet to the controller.
 - E.g., if the packet doesn't match any entries in the flow table or if the action is to send the packet to the controller



SDN: control/data plane interaction example

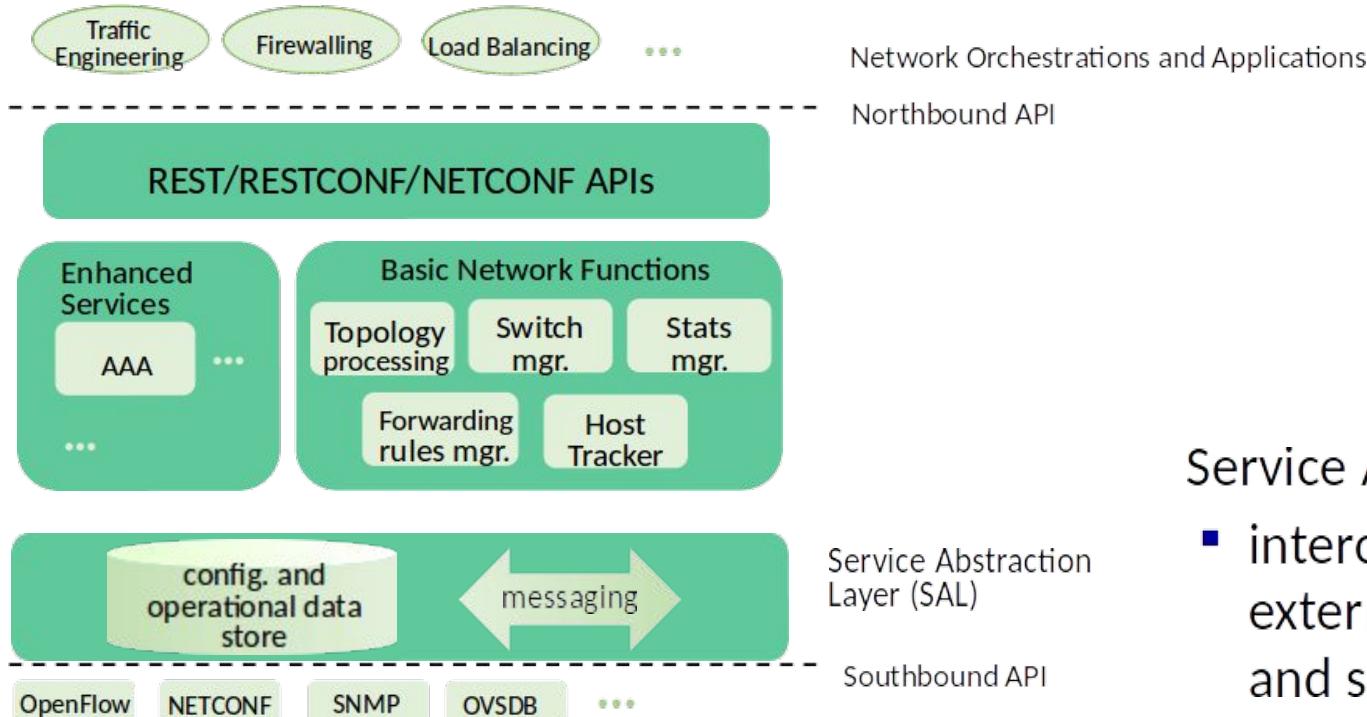
1. S1, experiencing link failure uses OpenFlow port status message to notify controller
2. SDN controller receives OpenFlow message, updates link state databases
3. Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It receives the notification of link-state change
4. Dijkstra's routing algorithm interact with the link state manager to get updated link cost; it might also consult other components in state-management layer. It then computes the new least-cost path
5. link state routing app interacts with the flow table manager, which determines the flow table to be updated
6. Controller uses OpenFlow to update flow table entries at affected switches



SDN Controller Case Studies

- OpenDaylight (ODL) controller
- ONOS controller

OpenDaylight (ODL) controller

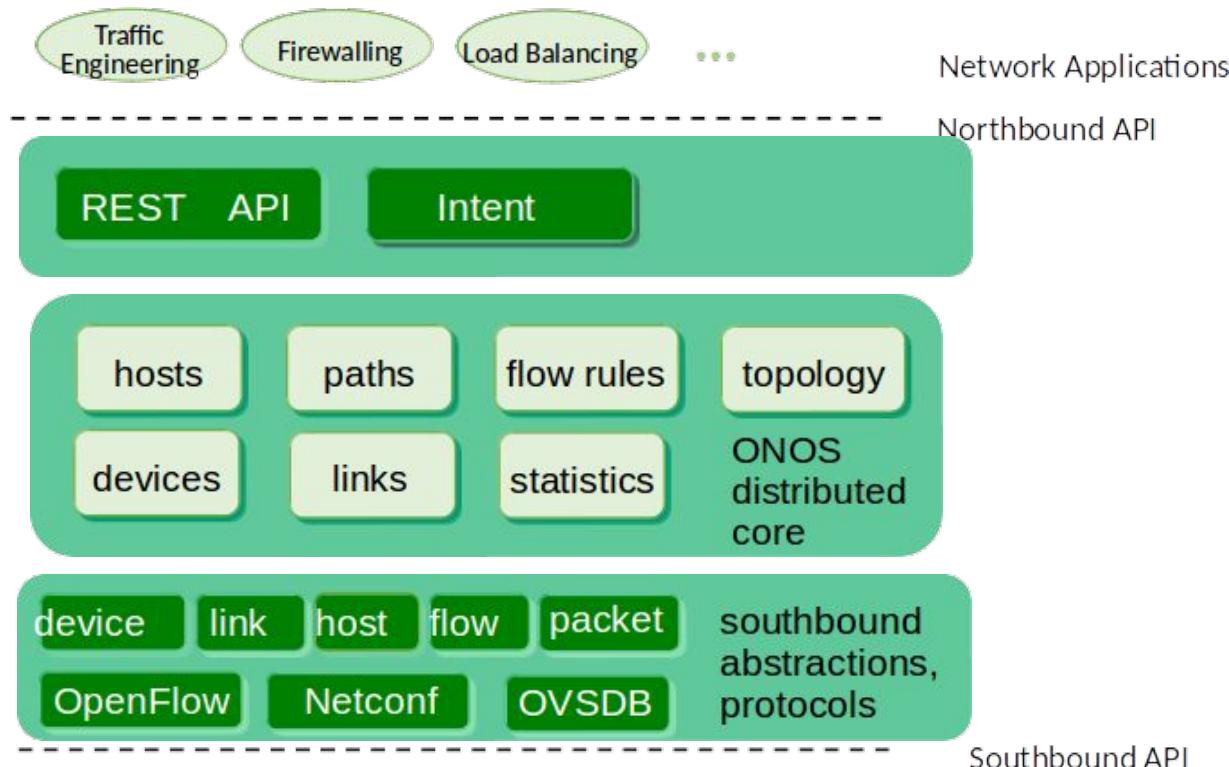


Service Abstraction Layer:

- interconnects internal, external applications and services

ONOS controller

- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling



SDN: selected challenges

- hardening the control plane: We need a dependable, reliable, performance-scalable, secure distributed system
 - robustness to failures: leverage strong theory of reliable distributed system for control plane
 - dependability, security: “baked in” from day one?
- Networks and protocols meeting mission-specific requirements
 - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling
 - An important research area is to extend SDN concepts from the intra-AS settings to the inter-AS settings
 - beyond a single AS

SDN and the future of traditional network protocols

- SDN-computed versus router-computer forwarding tables:
 - just one example of logically-centralized-computed versus protocol computed
- one could imagine SDN-computed congestion control:
 - controller sets sender rates based on router-reported (to controller) congestion levels

Outline

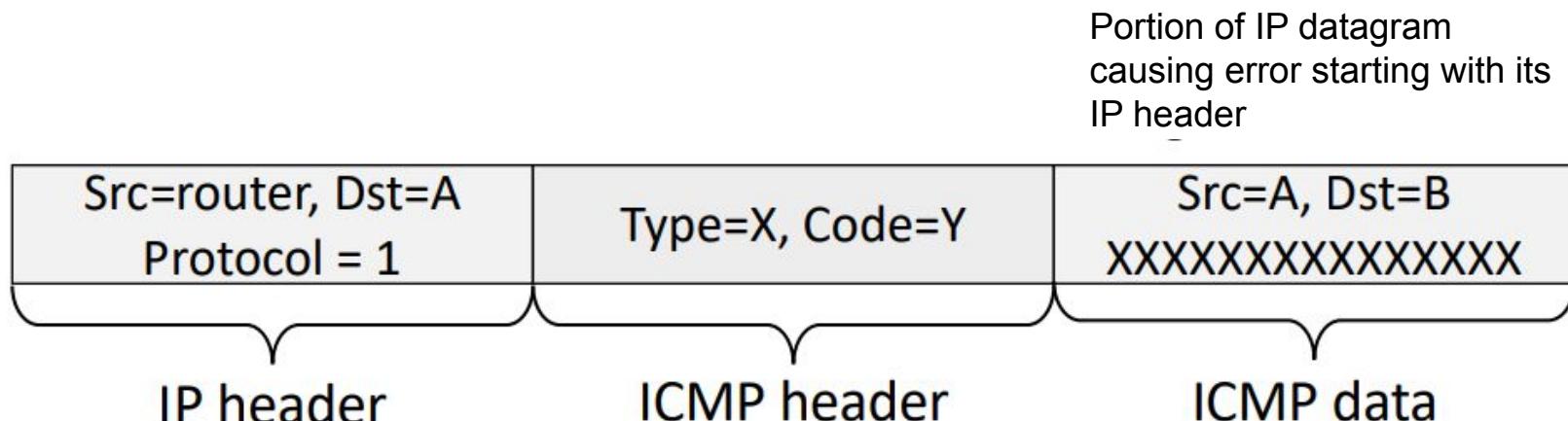
- introduction
- routing protocols
 - link state
 - distance vector
- intra-AS routing in the Internet: OSPF
- routing among the ISPs: BGP
- The SDN control plane
- **ICMP: The Internet Control Message Protocol**
- Network management and SNMP

ICMP: internet control message protocol

- used by hosts & routers to communicate network-level information
 - error reporting: unreachable host, network, port, protocol
 - echo request/reply (used by ping)
- network-layer “above” IP:
 - ICMP messages carried in IP datagrams
 - Protocol number: 1

ICMP Message Format

- Each ICMP message has a Type, Code, plus first 8 bytes of IP datagram causing ICMP message to be generated
 - Example: first 8 bytes of datagram whose TTL was reached zero
- Each message is carried in an IP packet



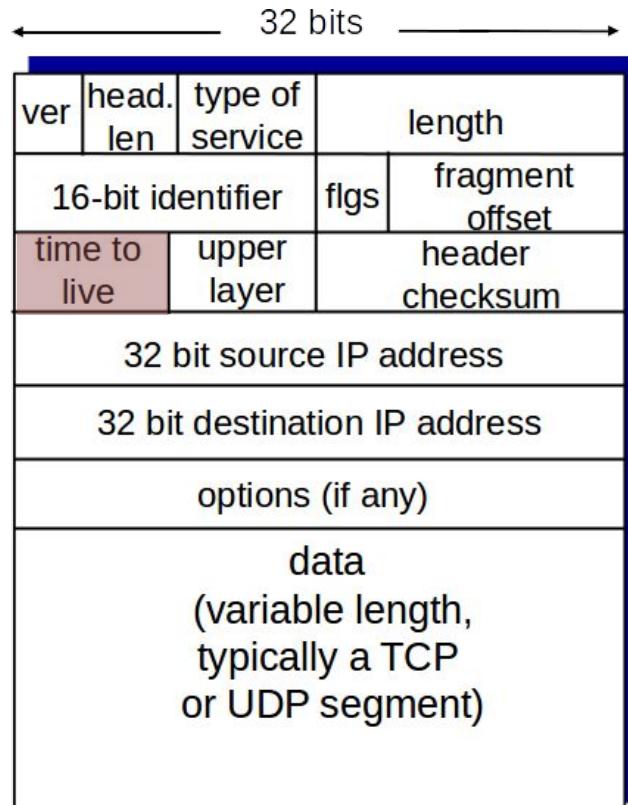
Example ICMP Messages

- Selected ICMP message types
- ICMP messages are used not only for signalling error conditions.
 - Example: it is used for implementing ping programs.
 - The program sends an ICMP echo request (type 8 code 0) message to the specified host. The host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply.

ICMP Type	Code	Description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

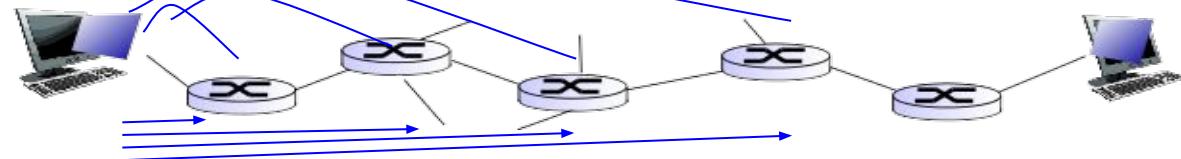
Traceroute

- Traceroute program allows us to trace a route from a host to any other host in the world
 - It specifies the names and addresses of the routers between source and destination.
- Traceroute is implemented with ICMP message.
- IP header contains TTL (Time to live) field
 - Decremented every router hop, with ICMP error at zero
 - Protects against forwarding loops
- Traceroute uses TTL and ICMP functionality



Traceroute and ICMP

- source sends sets of UDP segments to destination with increasing TTL starting from 1
 - first set has TTL =1; second set has TTL=2, etc.
 - The packet has an **unlikely port number**
- when datagram in **n-th** set arrives to **n-th** router:
 - router discards datagram and sends source ICMP message (type 11, code 0)
 - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: records RTTs
- stopping criteria:
 - UDP segment eventually arrives at destination host
 - destination returns ICMP “port unreachable” message (type 3, code 3)
 - source stops



Outline

- introduction
- routing protocols
 - link state
 - distance vector
- intra-AS routing in the Internet: OSPF
- routing among the ISPs: BGP
- The SDN control plane
- ICMP: The Internet Control Message Protocol
- **Network management and SNMP**

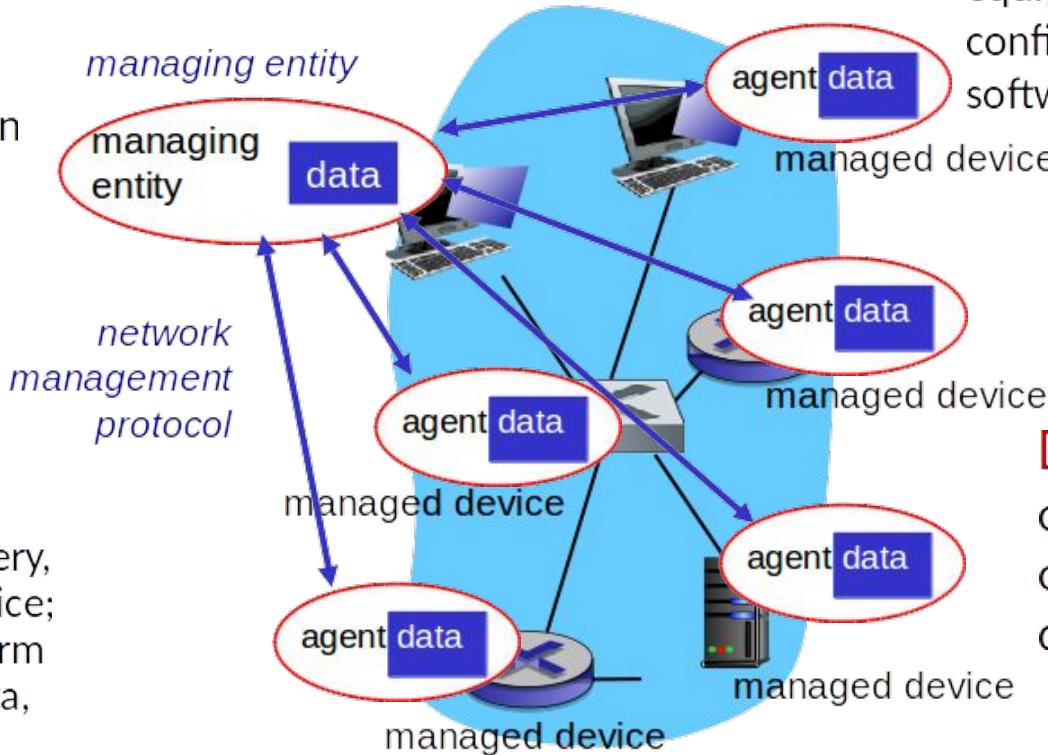
What is network management?

- autonomous systems (aka “network”): 1000s of interacting hardware/software components
- other complex systems requiring monitoring, control:
 - jet airplane
 - nuclear power plant
 - others?
- "**Network management** includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

Infrastructure for network management

Managing server: application, typically with network managers (humans) in the loop

Network management protocol: used by managing server to query, configure, manage device; used by devices to inform managing server of data, events.



Managed device: equipment with manageable, configurable hardware, software components

Data: device “state” configuration data, operational data, device statistics

Infrastructure for network management

- Key components of network management
 - The **managing server**:
 - is an application, typically with a human in the loop, running in a centralized network management station in the network operations center.
 - It controls the collection, processing, analysis, and/or display of network management information
 - Controlling network behavior is initiated here
 - Human network admin interacts with the network's device
 - A **managed device**
 - A piece of network equipment (including its software) that resides on a managed network: host, router, switch, middlebox, modem, or other network-connected device
 - **managed object**: actual piece of hardware within the managed device: e.g., a network interface card in a host or router, configuration parameters for these hardware and software

Infrastructure for network management

- Key components of network management
 - **Data: Management Information Base (MIB)**
 - The information from managed objects is collected into Management Information Base (MIB) and are available to managing server.
 - Example of an MIB object: a counter, such as the number of IP datagram discarded at a router due to errors in an IP datagram header; the number of UDP segments received at a host; status information about the functioning of a device
 - **network management agent**
 - a process running in the managed device that communicates with the managing server, taking local actions at the managed device under the command and control of the managing server

Infrastructure for network management

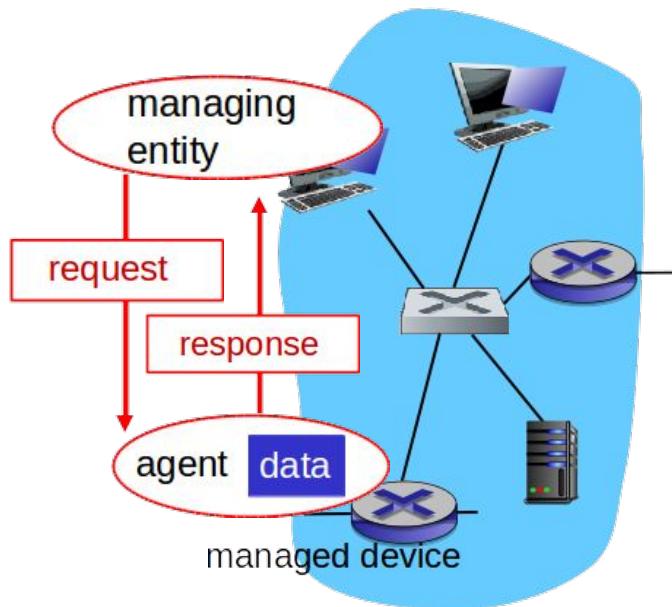
- Key components of network management
 - **network management protocol**
 - The protocol running between the managing server and the managed device
 - Allow the managing server to query the status of managed devices and take actions at the devices via its agents
 - Agents use the network management protocol to inform the managing server of events such as component failures
 - does not manage network itself
 - provide capabilities that a network admin can use to manage network

Network operator approaches to management

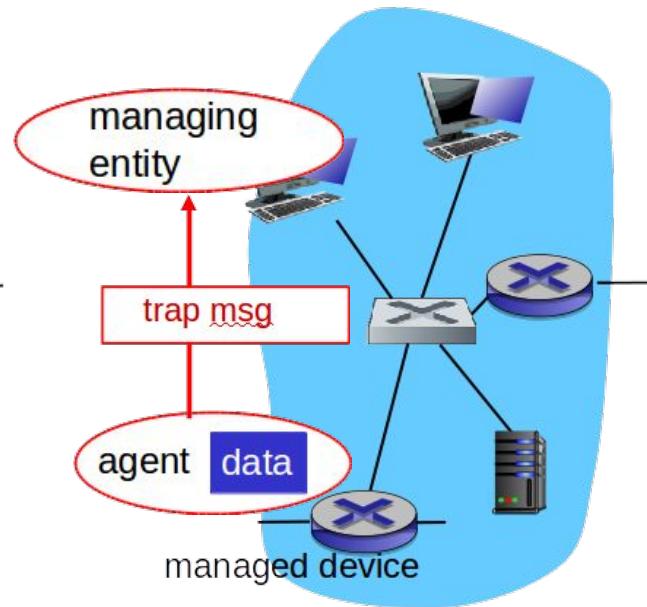
- CLI (Command Line Interface)
 - operator issues (types, scripts) direct to individual devices (e.g., via ssh)
- SNMP/MIB
 - operator queries/sets devices data (MIB) using Simple Network Management Protocol (SNMP)

SNMP protocol

Two ways to convey MIB info, commands:



request/response mode



trap mode

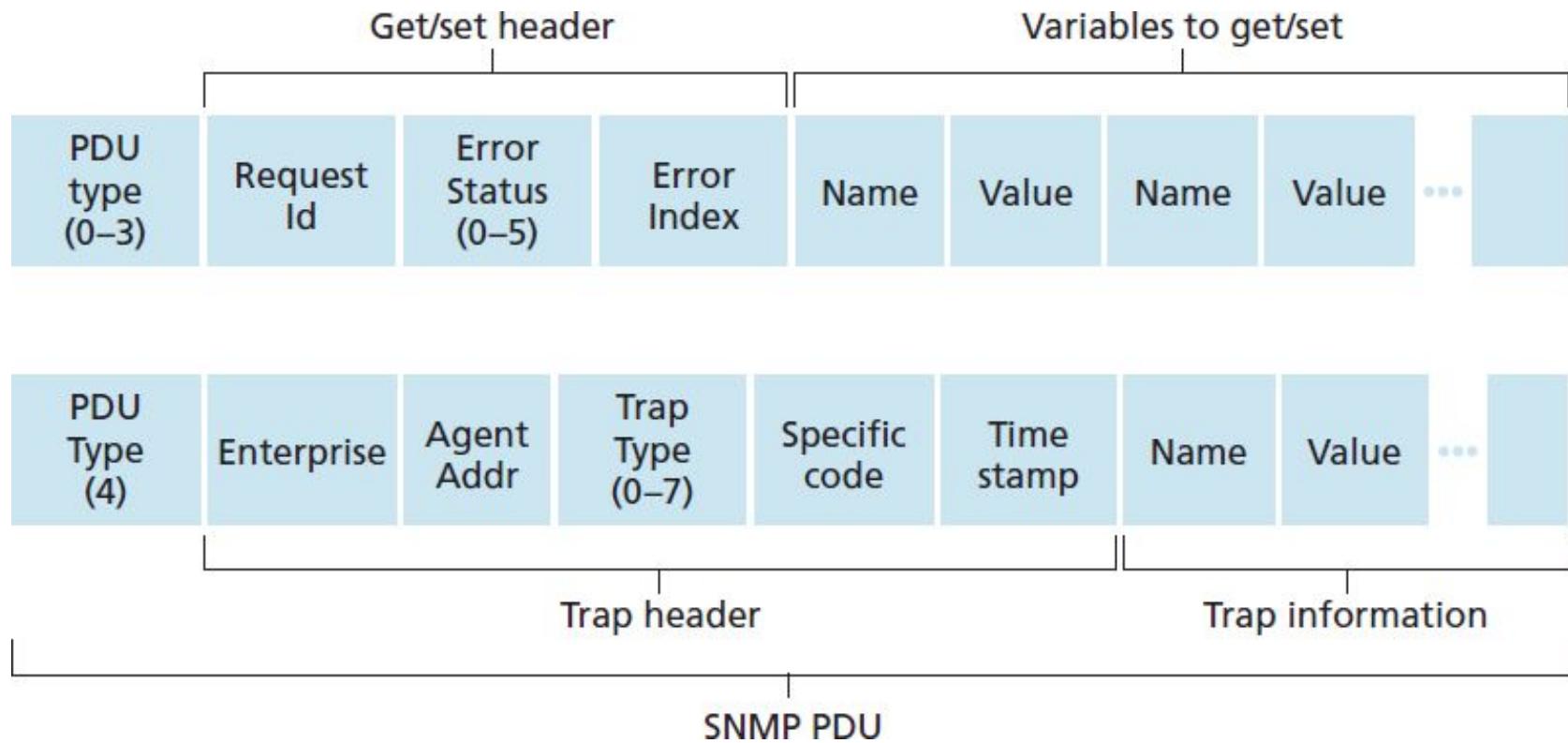
SNMP protocol

- The Simple Network Management Protocol is an application-layer protocol used to convey network-management control and information messages between a managing server and an agent executing on behalf of that managing server.
- Two common usage of SNMP:
 - A request-response mode: an SNMP managing server sends a request to an SNMP agent. The agent receives the request, performs some action and sends a response. The request could be to query(retrieve) or modify(set) MIB object values associated with a managed device.
 - An agent uses SNMP to send an unsolicited message, a trap message, to a managing server. Trap messages used to notify a managing server of an exceptional situation that has resulted in changes to MIB object values

SNMP protocol: message types

Message type	Function
GetRequest	manager-to-agent: “get me data”
GetNextRequest	(data instance, next data in list, block of data)
GetBulkRequest	
InformRequest	manager-to-manager: here's MIB value
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

SNMP protocol: message formats



SNMP Protocol

- SNMP version 2 define seven types of messages. The message are known as PDUs (Protocol Data Units). The message are shown above and the format of the PDU is shown in the next slide.
- **GetRequest; GetNextRequest; GetBulkRequest:** all sent from a managing server to an agent to request the value of one or more MIB objects at the agent's managed device. They differ in granularity of their data request.
- The SNMP PDUs are typically carried in the payload of a UDP datagram. UDP is an unreliable protocol, so there is no guarantee that a request, or its response, will be received at the intended destination.
- The managing server uses the **request ID** field of the PDU to number its requests to an agent; the agent's response takes it request id from that of the received request. So the managing server can find out if a request or response is lost. It is up to the managing server to retransmit the request if it is lost. SNMP standard does not specify that.
- SNMP version 3 adds security and administration. SNMPv2 is not secure.

summary

we've learned a lot!

- approaches to network control plane
 - per-router control (traditional)
 - logically centralized control (software defined networking)
- traditional routing algorithms
 - implementation in Internet: OSPF, BGP
- SDN controllers
 - implementation in practice: ODL, ONOS
- Internet Control Message Protocol
- network management

Outline

- introduction
- routing protocols
 - link state
 - distance vector
- intra-AS routing in the Internet: OSPF
- routing among the ISPs: BGP
- The SDN control plane
- ICMP: The Internet Control Message Protocol
- Network management and SNMP

Link Layer

goals

- understand principles behind link layer services:
 - error detection, correction
 - sharing a broadcast channel: multiple access
 - link layer addressing
 - local area networks: Ethernet, VLANs
- datacenter networks
- instantiation, implementation of various link layer technologies

Outline

- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - addressing, ARP
 - Ethernet
 - Switches
 - VLANs
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

What is the link?

Communication channel that connects adjacent nodes along the communication path

- wired links
- wireless links
- LANs

Communication Medium



Types of links is the link?

- Point-to-point (PPP) communication links
 - The link connecting two routers by a long-distance link
 - point-to-point link between Ethernet switch and a user's office computer
- broadcast (shared medium): connects multiple hosts
 - Wired: Wired LAN (Ethernet)
 - Connect multiple hosts in wireless LANs, satellite network: wireless LAN (802.11)



shared wire (e.g.,
cabled Ethernet)



shared RF
(e.g., 802.11 WiFi)



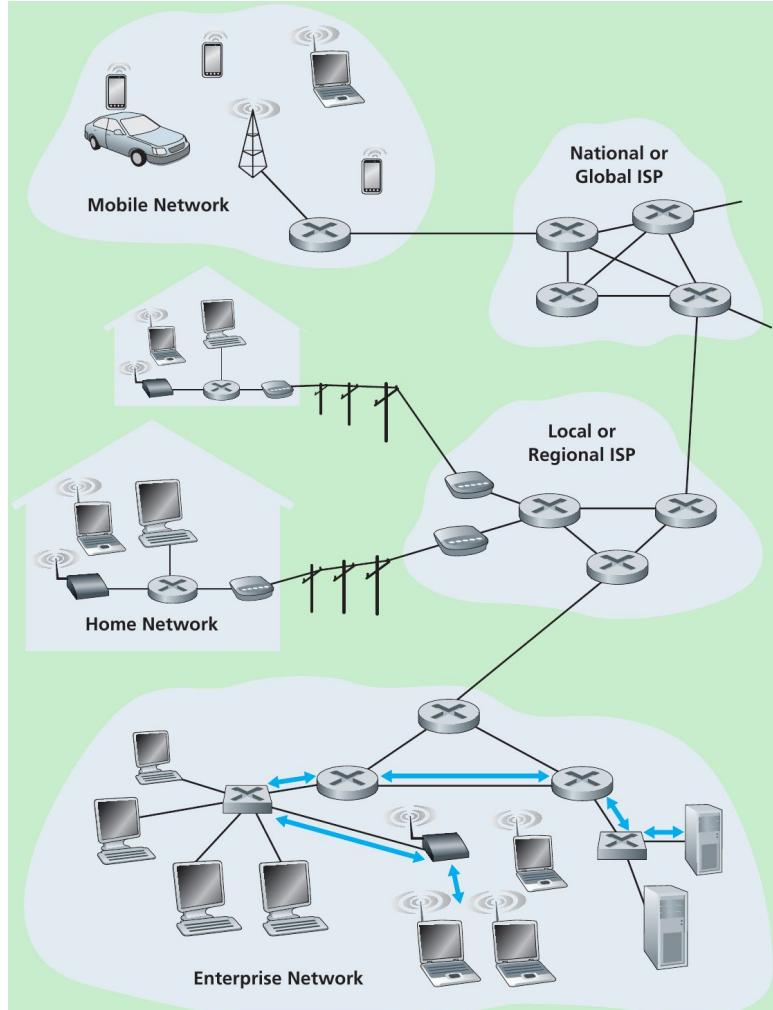
shared RF
(satellite)



humans at a
cocktail party
(shared air, acoustical)

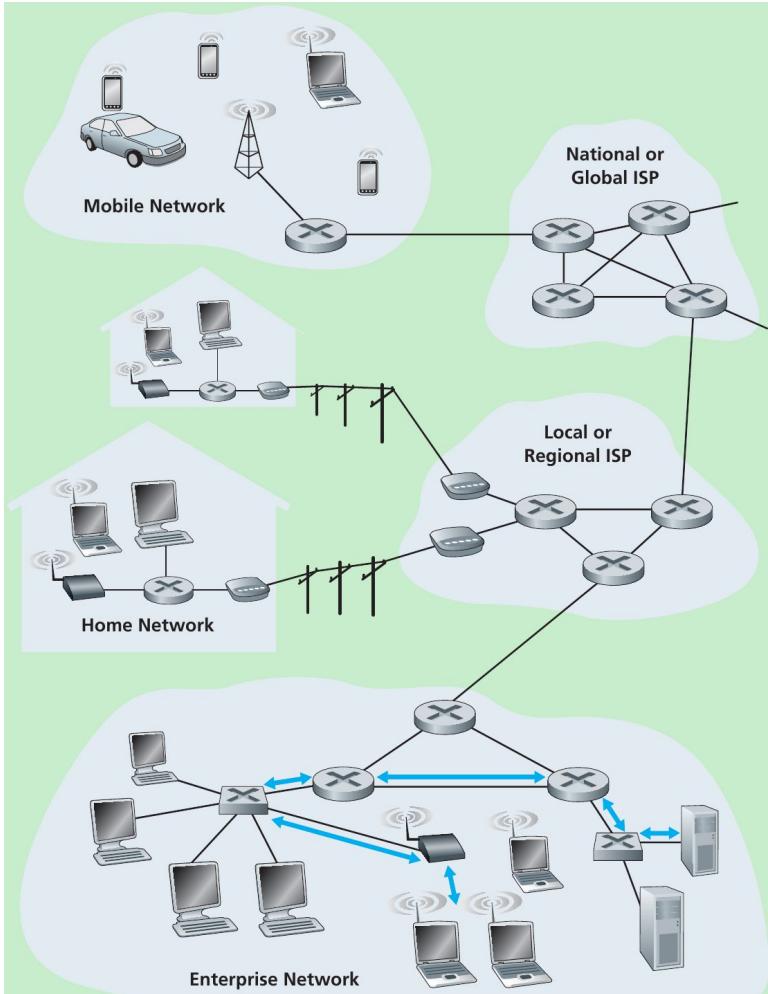
Link Layer

- **Nodes:** hosts and routers
 - **Links:** communication channels that connect adjacent nodes along communication path
 - wired links
 - wireless links
 - LANs
- **Frame:** link-layer packet, encapsulates datagram
 - **link layer** has responsibility of transferring datagram from one node to **physically adjacent node** over a link



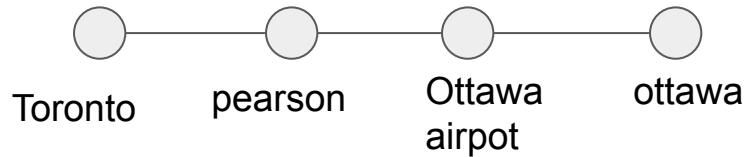
Link Layer: context

- datagram transferred by different link protocols over different links:
 - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- each link protocol provides different services
 - e.g., may or may not provide reliable data transfer over link



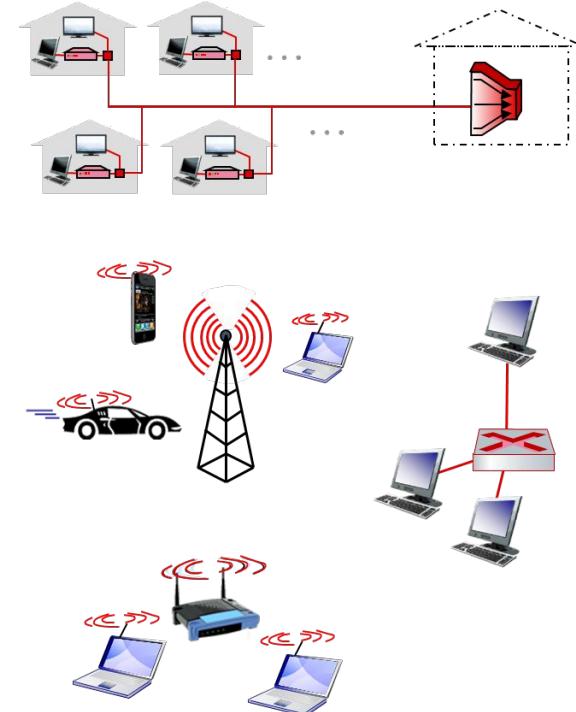
Link Layer: context

- transportation analogy: Link Layer and Network Layer
 - trip from Toronto to Ottawa
 - limo: Toronto to Pearson
 - plane: Pearson to Ottawa Airport
 - train: Ottawa Airport to Ottawa center
- tourist = datagram
- transport segment = communication link
- transportation mode = link layer protocol
- travel agent = routing algorithm



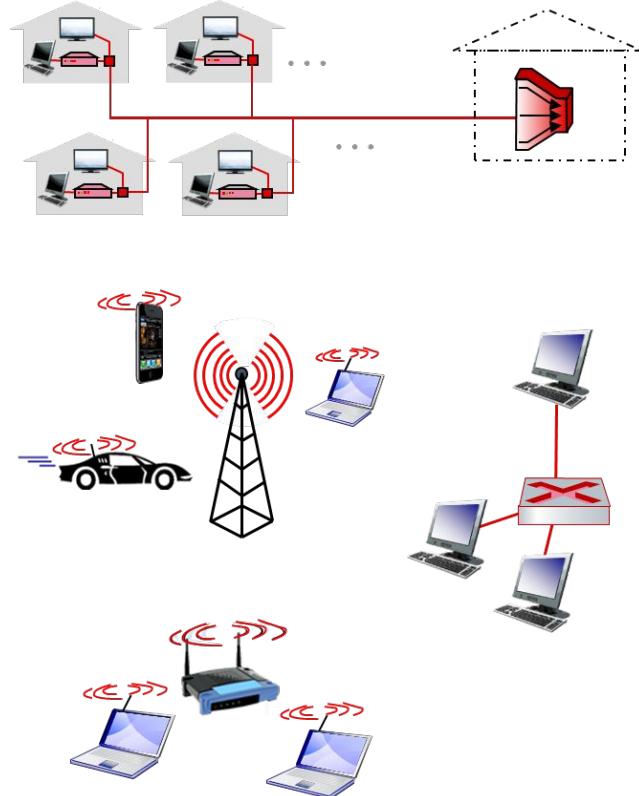
Link Layer Services

- **Framing**
 - encapsulate datagram into frame, adding header, trailer
- **Link access:**
 - A multiple access protocol specifies the rules by which a frame is transmitted onto the link
 - For point-to-point: no MAC protocol, sender sends a frame whenever the link is idle
 - For shared channel
 - “MAC” addresses used in frame headers to identify source, destination
 - different from IP address!
- **reliable delivery between adjacent nodes**
 - we learned how to do this already
 - seldom used on low bit-error link (fiber, some twisted pair)
 - wireless links: high error rates
 - Q: why both link-level and end-end reliability? Remember, many errors happen in the routers



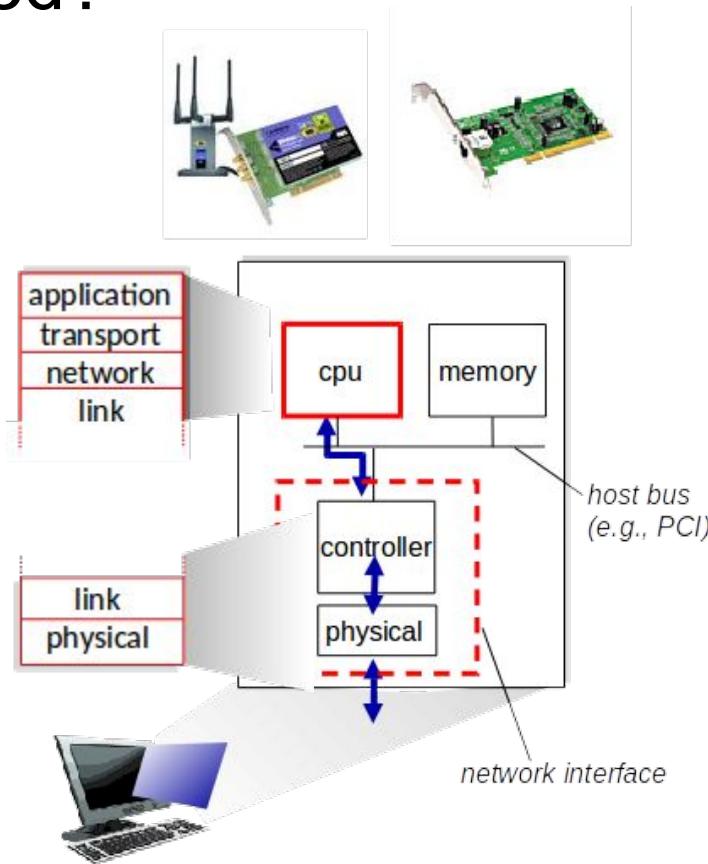
Link Layer Services

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



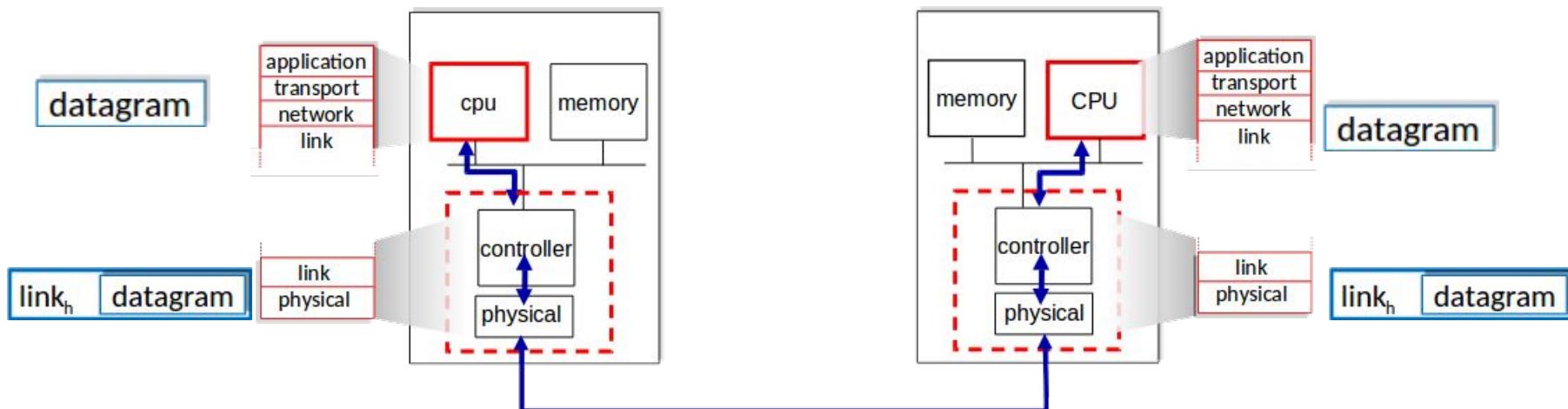
Where is the link layer implemented?

- in each and every node
- link layer implemented in
 - “adaptor” (aka network interface card NIC)
 - Controller
 - Framing, link access, error detection, etc.
 - implements link, physical layer
 - Ethernet card, 802.11 card;
 - Used to be separate cards (PCMCIA card)
 - Attached to host's bus
 - Integrated onto the host's motherboard
 - Software
 - Assembling link-layer addressing information
 - Activating the controller hardware
 - combination of hardware, software



Adaptors communicating

- Sending side
 - Encapsulates datagram in a frame
 - Adds error checking bits, reliable data transfer, flow control, etc.
- Receiving side
 - Looks for errors, reliable data transfer, flow control, etc.
 - Extracts datagram and passes to upper layer at receiving node

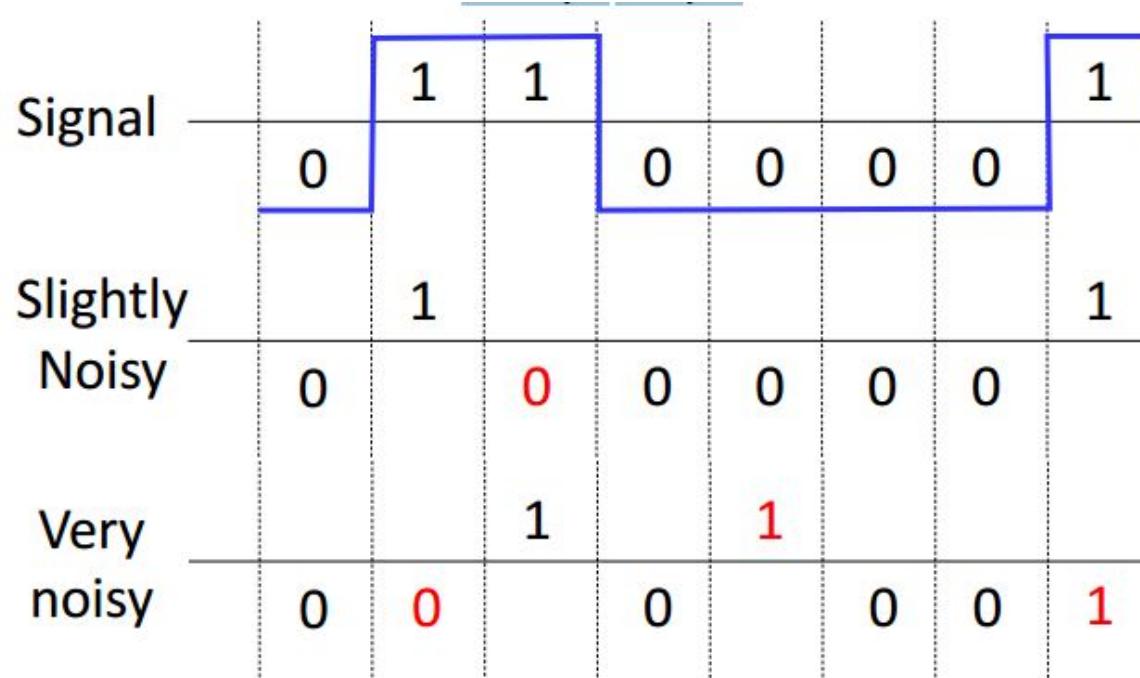


Outline

- introduction, services
- **error detection, correction**
- multiple access protocols
- LANs
 - addressing, ARP
 - Ethernet
 - Switches
 - VLANs
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

Error detection and correction

Problem: Noise may Flip Received Bits



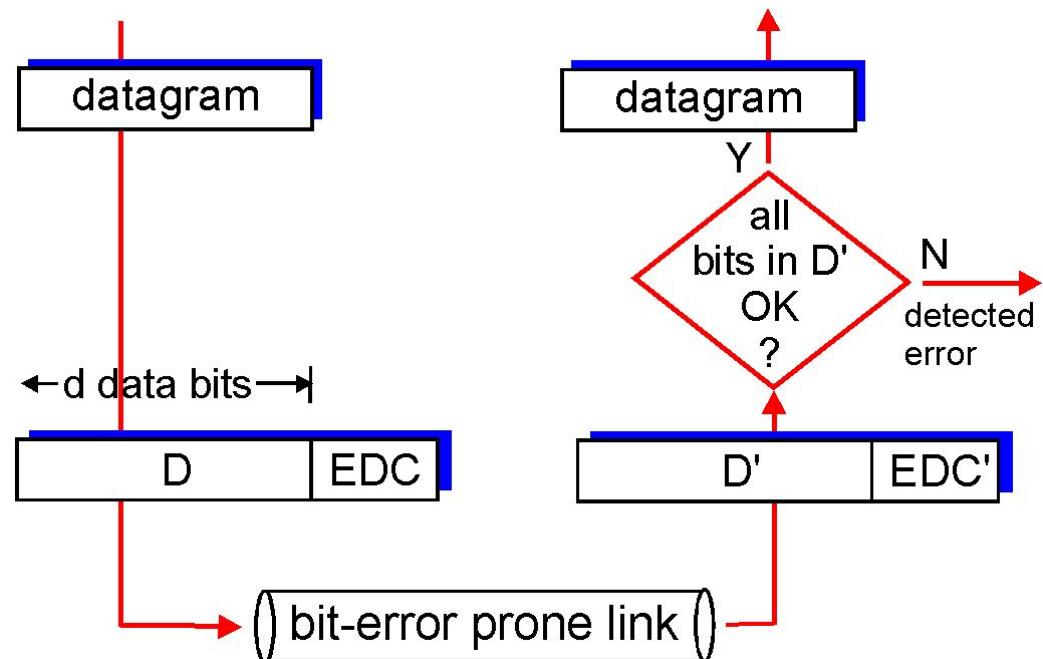
Link Layer: Error detection and correction

Problem: Noise may flip received bits

- Solutions: Link layers provides some protection
 - Retransmit lost frames
 - Add Redundancy
 - Error detection codes: Add check bits to the message bits to let some errors be detected
 - Error correction codes: Add more check bits to let some errors be corrected

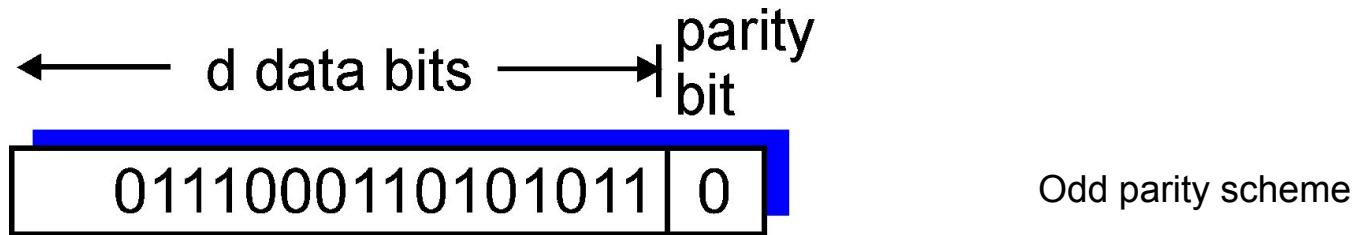
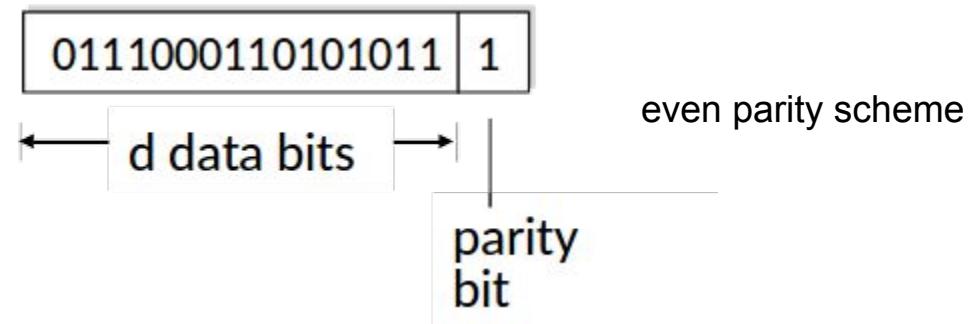
Error detection and correction

- D = Data protected by error checking, may include header fields
- EDC = Error Detection and Correction bits (redundancy)
 - Error detection not 100% reliable!
 - protocol may miss some errors
 - larger EDC field yields better detection and correction



Error detection and correction: Parity checks

- **single bit parity:**
 - detect single bit errors



Error detection and correction: Parity Checks

- Simple Error Detection – **Parity Bit**
 - Take D data bits, add 1 check bit
 - Sender:
 - Even-parity scheme
 - Set parity bit so that there is an even number of 1s in d+1 bits
 - Odd-parity scheme
 - Set parity bit so that there is an odd number of 1s in d+1 bits
 - Receiver
 - Odd-number of 1-valued bits in an **even-parity** scheme → at least one error
 - knows some odd numbers of bit errors have occurred
 - What if an even number of bit errors occur?
 - Undetected errors (no errors or possibly an even number of errors)
 - Errors often are clustered in bursts
 - The probability of error can approach 50 percent

Parity checks

- **Two-dimensional bit parity:**
 - Data is divided into i rows and j columns
 - A parity value is computed for each row and for each column
- In case of a **single** bit error
 - **Detect** error and **correct** error
 - The parity of both the column and the row containing the flipped bit will be in error
- Can detect **two** bit errors
 - But **cannot** fix them
- Example: 2-dimensional even parity

				row parity
				$d_{1,j+1}$
				$d_{2,j+1}$
				...
				$d_{i,j+1}$
column parity				$d_{i+1,j+1}$
				$d_{i+1,j+1}$
				$d_{i+1,j+1}$

10101 1
11110 0
01110 1
0.01010

no errors

10101 1
10110 0
01110 1
0.01010

parity error

*correctable
single bit error*

Error detection and correction

- Two-dimensional parity check is an example of Forward error checking (FEC) techniques
 - Decrease the number of retransmissions by sender
 - Immediate correction of errors at the receiver
 - Receiver avoid having to wait for the round-trip propagation delay needed for the sender to receive a NACK packet and for the retransmitted packet to propagate back to the receiver
 - A big advantage for real-time network applications

Error detection and correction

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

- Sender:
 - treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
 - checksum: addition (one's complement sum) of segment content
 - checksum value put into UDP checksum field
- Receiver:
 - compute checksum of received segment
 - check if computed checksum equals checksum field value:
 - not equal - error detected
 - equal - no error detected. *But maybe errors nonetheless? More later*

Error detection and correction

Checksums

- Stronger protection (error detection) than parity
- Widely used in, e.g., TCP/IP/UDP
 - Implemented in software
 - Simple and fast
 - Little packet overhead
- Weaker than some of the techniques in link layer, such as CRC.

Error detection and correction: CRC

Cyclic Redundancy Check (CRC)

- more powerful error-detection coding

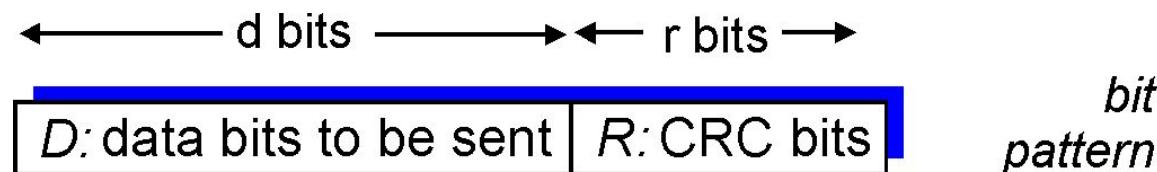
Link Layer: Error detection and correction

Cyclic Redundancy Check (CRC)

- We work with binary values and operate using modulo 2 arithmetic
 - No carries in addition or borrows in subtraction
 - Addition and subtraction are equivalent to the bitwise exclusive-or (XOR) of the operands
 - $1011 \text{ XOR } 0101 = 1110$ is equivalent to $1011 - 0101 = 1110$
 - $1001 \text{ XOR } 1101 = 0100$ is equivalent to $1001 + 1101 = 0100$
 - Multiplication and division are the same as in base-2 arithmetic except that any required addition or subtraction is done without carries or borrows

Error detection and correction: CRC

- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of $r+1$ bits (given)
 - **Sender:** Given d data bits, D , **generate r check bits, R** , such that the $d+r$ bits exactly divisible by a generator G
 - **receiver:** knows G , divides $d+r$ bits received by G . If non-zero remainder: error detected!
 - can detect all burst errors less than $r+1$ bits
 - widely used in practice (Ethernet, 802.11 WiFi)



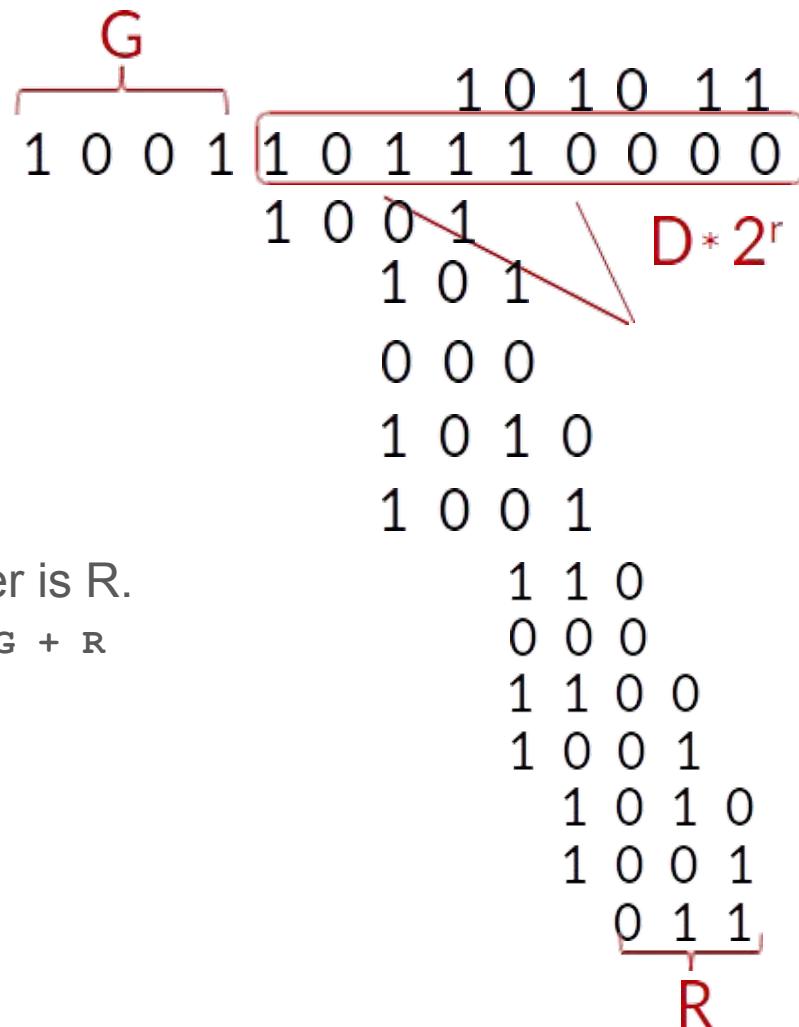
$$D * 2^r \text{ xor } R = nG$$

*mathematical
formula*

CRC: Example

- Sender wants to compute R such that:
 - $D \cdot 2^r \text{ XOR } R = nG$
- Equivalently:
 - $D \cdot 2^r = nG \text{ XOR } R$
- equivalently:
 - if we divide $D \cdot 2^r$ by G, then remainder is R.
 - $D \cdot 2^r = nG \text{ XOR } R$ is equivalent to $D \cdot 2^r = nG + R$

$$R = \text{remainder} \frac{D \cdot 2^r}{G}$$



CRC: Example

Data bits: 1101011111

G = 10011

r = 4 bits

1 0 0 1 1	1 1 0 1 0 1 1 1 1 1
-----------	---------------------

G has r+1 bits

1	0	0	1	1	/	1	1	0	0	0	0	1	1	1	1	0	← Quotient (thrown away)							
						1	1	0	1	0	1	1	1	1	1	0	0	0	0	← Frame with four zeros appended				
						1	0	0	1	1														
						1	0	0	1	1														
						1	0	0	1	1														
						0	0	0	0	1														
						0	0	0	0	0														
						0	0	0	1	1														
						0	0	0	0	0														
						0	0	1	1	1														
						0	0	0	0	0														
						0	1	1	1	1														
						0	0	0	0	0														
						1	1	1	1	0														
						1	0	0	1	1														
						1	1	0	1	0														
						1	0	0	1	1														
						0	0	0	1	0														
						0	0	0	0	0														
																1	0	← Remainder						

Transmitted frame: 1 1 0 1 0 1 1 1 1 0 0 1 0 ← Frame with four zeros appended minus remainder

Outline

- introduction, services
- error detection, correction
- **multiple access protocols**
- LANs
 - addressing, ARP
 - Ethernet
 - Switches
 - VLANs
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

Multiple access links, protocols

two types of “links”:

- Point-to-point
 - point-to-point link between Ethernet switch, host
 - PPP: a link-layer protocol for point-to-point links: for dial-up access
- Broadcast link: multiple sending and receiving nodes connected to the same channel
 - (shared wire or medium)
 - old-fashioned Ethernet
 - upstream HFC in cable-based access network
 - 802.11 wireless LAN, 4G/4G, satellite



shared wire (e.g.,
cabled Ethernet)



shared radio: 4G/5G



shared radio: WiFi



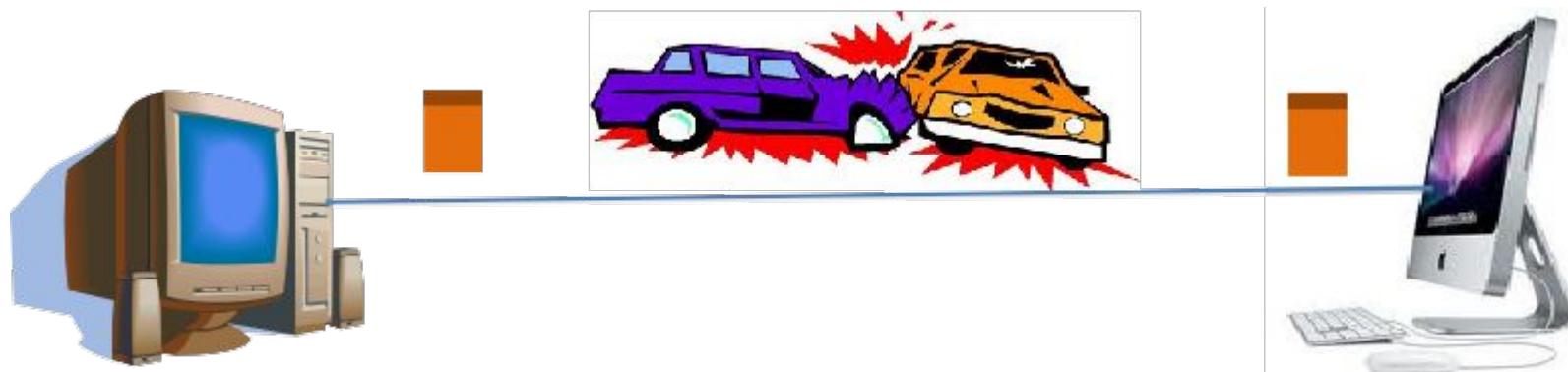
shared radio: satellite



humans at a cocktail party
(shared air, acoustical)

Multiple Access 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
 - Avoid having multiple nodes transmit at once
 - Otherwise, collisions lead to garbled data
 - Human analogy



Multiple Access Protocols

Multiple Access Protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
 - no out-of-band channel for coordination

Multiple Access Protocols

given: broadcast channel of rate R bps

- Goals:
 - High utilization of the shared channel
 - when one node wants to transmit, it can send at rate R .
 - Fair among nodes
 - when M nodes want to transmit, each can send at average rate R/M
 - fully decentralized:
 - no special node to coordinate transmissions
 - no synchronization of clocks, slots
 - Simple and low cost to implement

MAC Protocols

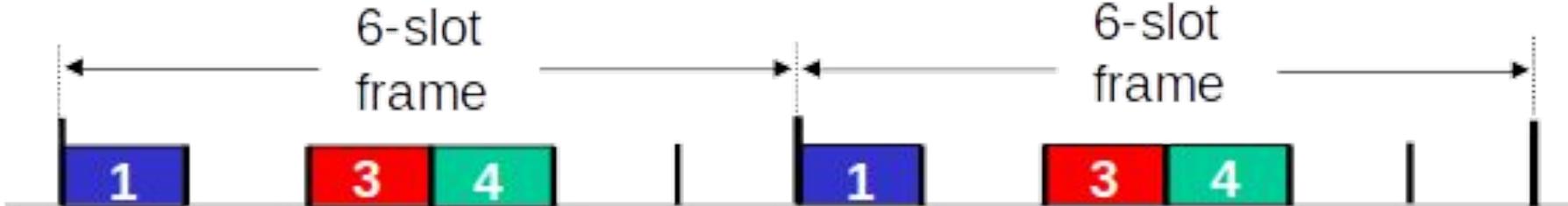
Three broad classes:

- **channel partitioning**
 - divide channel into smaller “pieces” (time slots, frequency, code)
 - allocate piece to node for exclusive use
 - TDM, FDM
- **random access**
 - channel not divided, allow collisions
 - “recover” from collisions
 - Ethernet (IEEE 802.3) , Aloha
- **“taking turns”**
 - nodes take turns, but nodes with more to send can take longer turns
 - Token Ring (IEEE 802.5)

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

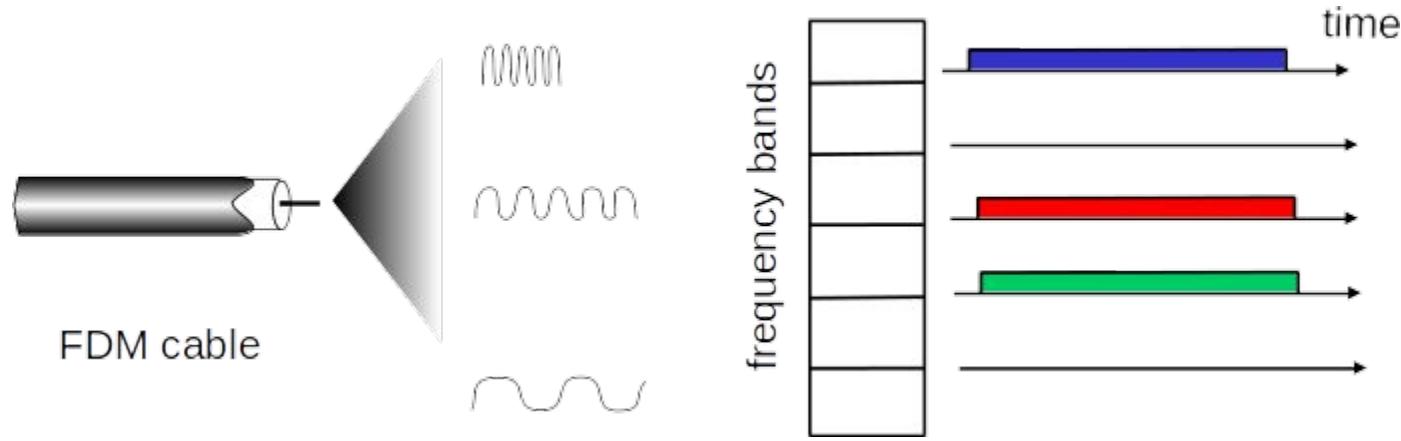
- access to channel in "rounds"
- each node gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-node LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

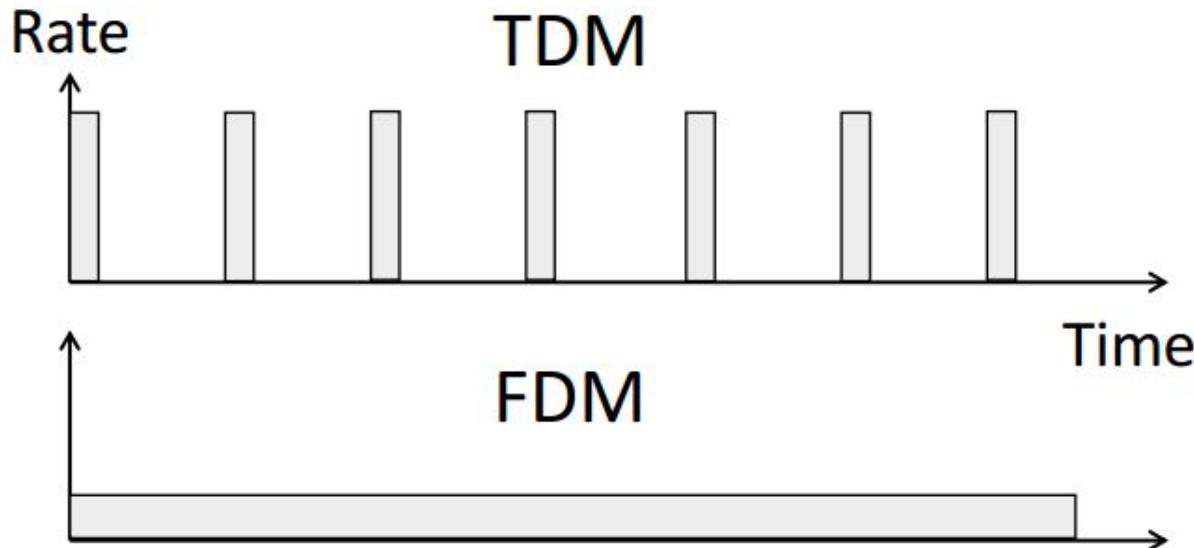
- channel spectrum divided into frequency bands
- each node assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-node LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



TDM versus FDM

In TDM a user sends at a high rate a fraction of the time;

in FDM, a user sends at a low rate all the time



MAC Protocols

Three broad classes:

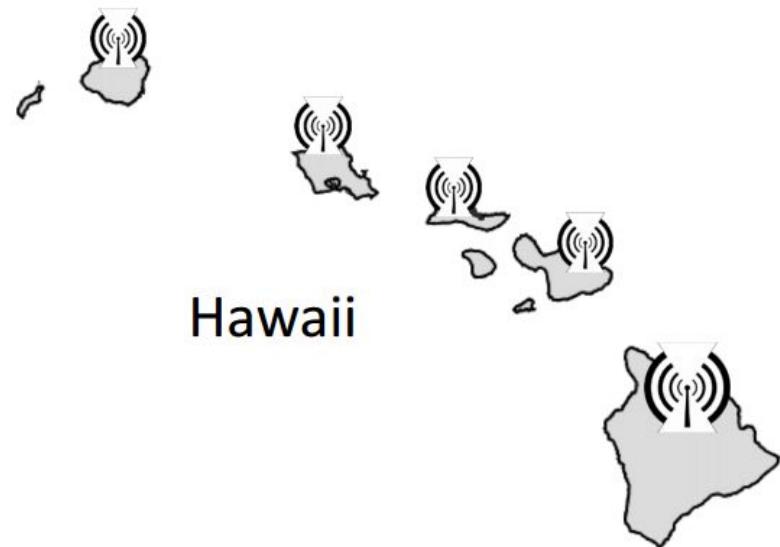
- **channel partitioning**
 - divide channel into smaller “pieces” (time slots, frequency, code)
 - allocate piece to node for exclusive use
 - TDM, FDM
- **random access**
 - channel not divided, allow collisions
 - “recover” from collisions
 - Ethernet (IEEE 802.3) , Aloha
- **“taking turns”**
 - nodes take turns, but nodes with more to send can take longer turns
 - Token Ring (IEEE 802.5)

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 sending nodes → “collision”,
- **random access MAC protocol** specifies:
 - When to send
 - how to detect collisions
 - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
 - slotted ALOHA
 - ALOHA
 - CSMA, CSMA/CD, CSMA/CA

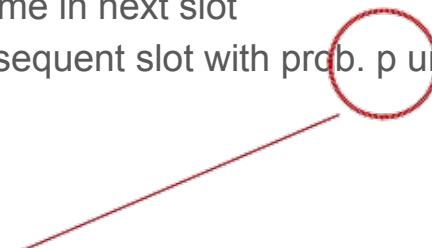
Random Access Protocols: ALOHA

- A satellite-based computer network connecting the Hawaiian islands in the late 1960s
 - A new protocol was devised by Norm Abramson
- **Innovating random access:**
 - **Allow collision to happen** (and then recover via retransmission)
 - **Use randomization** in choosing when to retransmit



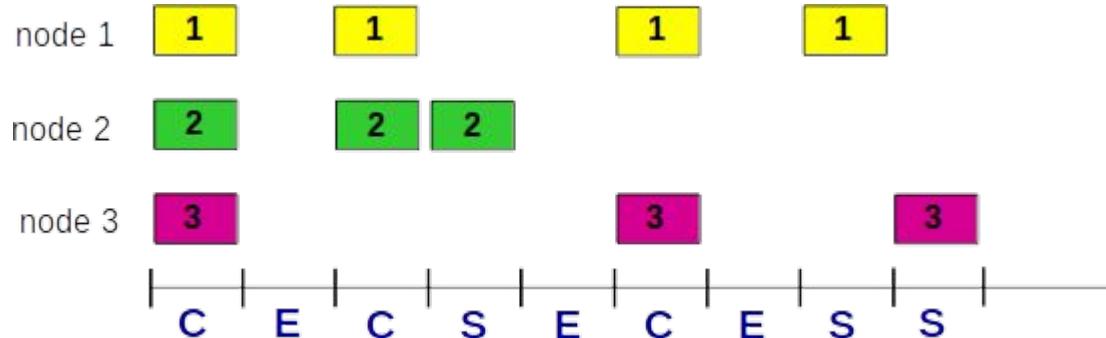
Slotted ALOHA

- Assumptions:
 - all frames same size
 - time divided into equal size slots (time to transmit 1 frame)
 - nodes start to transmit only at beginning of a slot
 - nodes are synchronized so each node knows when the slot begins
 - if 2 or more nodes transmit in slot, all nodes detect collision
- Operation:
 - when node has frame to send, transmits in next slot
 - **if no collision:** success, node can send new frame in next slot
 - **if collision:** node retransmits frame in each subsequent slot with prob. p until success



randomization - why?

Slotted ALOHA



- Pros:
 - single active node can continuously transmit at full rate of channel
 - highly decentralized: only slots in nodes need to be in sync
 - simple
- Cons:
 - collisions, wasting slots
 - idle slots, wasting
 - clock synchronization

Slotted ALOHA: Analysis of efficiency

- **efficiency:** long-run fraction of successful slots (many nodes, all with many frames to send)
 - Assumption: N nodes with many frames to send, each transmits in slot with probability p (a little modification from slotted ALOHA)
 - prob that given node has success in a slot = $p(1-p)^{N-1}$
 - prob that *any* node has a success = $Np(1-p)^{N-1}$
 - Maximum efficiency is obtained when $p=1/N$: the value that maximizes $Np(1-p)^{N-1}$
 - Maximum efficiency for long-run fraction of successful slots (many nodes, all with many frames to send)
 - take limit of $Np(1-p)^{N-1}$ as N goes to infinity and $p=1/N$ gives:
 - **max efficiency = $1/e = .37$**
- When a large number of nodes have many frames to transmit, at best only 37% of slots do useful work!
- Active transmission rate for a channel is not R bps but 0.37 R bps

Pure ALOHA Protocol

- Simple idea:
 - Node just sends when it has traffic.
 - If there was a collision (no ACK received) then wait a random time and resend
- That's it!

Pure ALOHA Protocol

- Simple, decentralized protocol that works well under low load!
 - Not efficient under high load
 - Analysis shows at most 18% efficiency

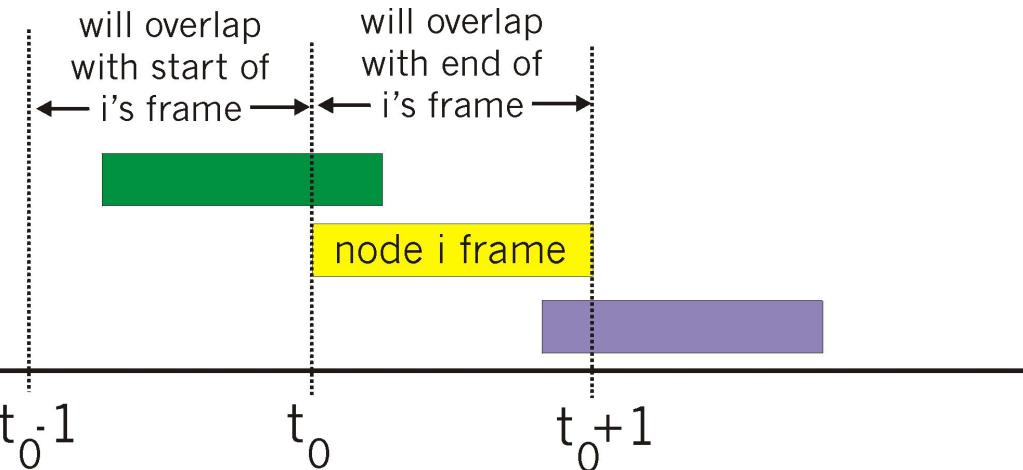
Pure ALOHA Protocol: Analysis

Assumption: Frame transmission time: the unit of time
frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$
 $P(\text{success by given node}) =$

$$\begin{aligned} & P(\text{node transmits}) \times P(\text{no other node transmits in } [t_0-1, t_0]) \times P(\text{no other node transmits in } [t_0, t_0+1]) \\ &= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1} = p \cdot (1-p)^{2(N-1)} \end{aligned}$$

$$P(\text{success by any node}) = N \cdot p \cdot (1-p)^{2(N-1)}$$

... choosing optimum p and then letting $n \rightarrow \infty$
 $= 1/(2e) = .18$

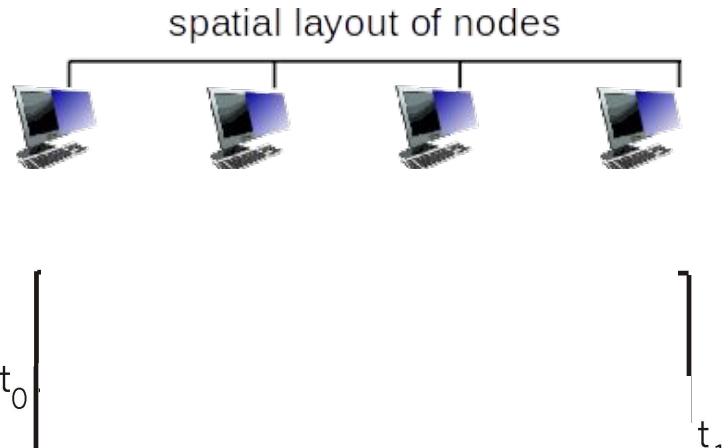


CSMA (Carrier Sense Multiple Access)

- Listen for activity (Carrier Sense) before sending a packet
 - if channel sensed idle: transmit entire frame
 - if channel sensed busy: defer transmission (called random backoff)
 - human analogy: don't interrupt others!
- CSMA/CD: CSMA with **collision detection**
 - collisions detected within short time
 - colliding transmissions aborted, reducing channel wastage
 - collision detection easy in wired, difficult with wireless
 - human analogy: the polite conversationalist
- So does this eliminate collisions?

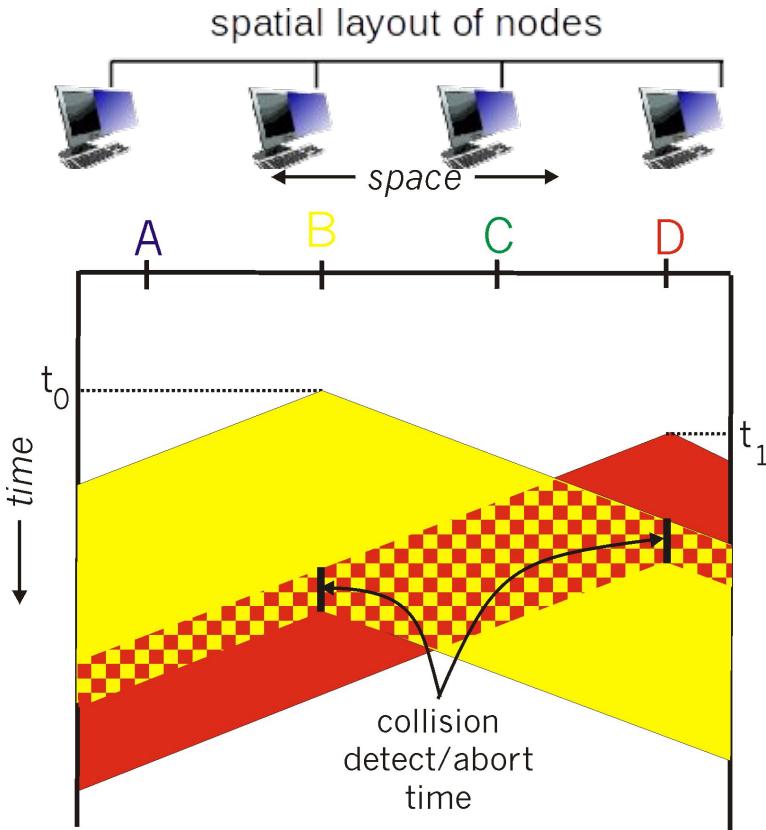
CSMA: Collision

- **collisions can still occur with carrier sensing:**
 - propagation delay means two nodes may not hear each other's just-started transmission
- **collision:** entire packet transmission time wasted
 - distance & propagation delay play role in determining collision probability



CSMA/CD

- CSMA with collision detection.
- CSMA/CS reduces the amount of time wasted in collisions
 - transmission aborted on collision detection



CSMA/CD in Ethernet

All hosts transmit and receive on one channel; frames are of variable size

When a host has a frame to transmit:

1. Carrier Sense:
 - o if **idle**: start frame transmission.
 - o if **busy**: wait until channel idle, then transmit
2. If transmits entire frame without collision, it is done with frame !
3. If collision is detected, stop transmitting (abort)
4. After aborting, the host waits a **random time**, then return to step 1.


Binary exponential backoff

after m -th collision, the host chooses K at random from $\{0, 1, 2, \dots, 2^m - 1\}$.

In ethernet, a node waits $K \cdot 512$ bit times (K times the amount of time needed to send 512 bits into the Ethernet)
longer backoff interval with more collision

The value of m is at most 10 in the Ethernet

CSMA/CD in Ethernet

- Example:
 - A node transmits a frame for the first time and detects a collision
 - The node chooses K from {0, 1}.
 - If K=0, immediately starts sensing the channel
 - If K=1, it waits 512 bit times (e.g., 5.12 microseconds for a 100 Mbps Ethernet) before beginning the sense-and-transmit-when-idle cycle.
 - After a second collision, K is chosen with equal probability from {0,1,2,3}.
 - After three collision, K is chosen from {0,1,2,3,4,5,6,7}.
 - After 10 or more collision, K is chosen from {0, 1,2,...,1023}
 - The size of the sets from which K is chosen grows exponentially with the number of collision;
That's why the algorithm is referred to as binary exponential backoff

CSMA/CD efficiency

- Efficiency of CSMA/CD:
 - When there is a large number of active nodes, with each node having a large number of frames to send, what percentage of link capacity used for transmitting data and not wasted
 - When only one node has a frame, the node can transmit at the full channel rate
 - If many nodes have frames to transmit, the effective transmission rate of the channel can be much less
 - t_{prop} = max prop delay between 2 nodes in LAN
 - t_{trans} = time to transmit max-size frame
 - Efficiency goes to 1
 - as t_{prop} goes to 0 → efficiency goes to 1
 - as t_{trans} goes to infinity → efficiency goes to 1
- better performance than ALOHA; and simple, cheap, decentralized!

$$\text{efficiency} = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

Question

Suppose two nodes start to transmit at the same time a packet of length P over a broadcast channel of rate R . The propagation delay between the two nodes is d_{prop} . Will there be a collision if $P/R > d_{\text{prop}}$?

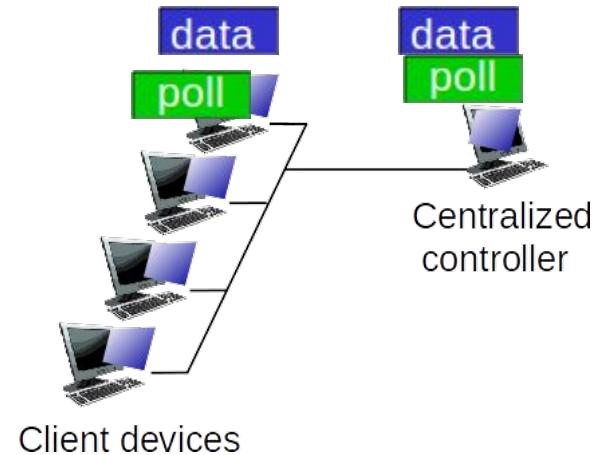
- True
- False

“Taking turns” MAC protocols

- channel partitioning MAC protocols:
 - share channel *efficiently* and *fairly* at high load
 - inefficient at low load: delay in channel access, $1/N$ bandwidth allocated even if only 1 active node!
- random access MAC protocols
 - efficient at low load: single node can fully utilize channel
 - high load: collision overhead
- “**taking turns**” protocols
 - look for best of both worlds!
 - Channel allocated explicitly (no collisions)
 - Nodes won’t hold channel for long if nothing to send
 - Two approaches: **polling, token passing**

“Taking turns” MAC protocols

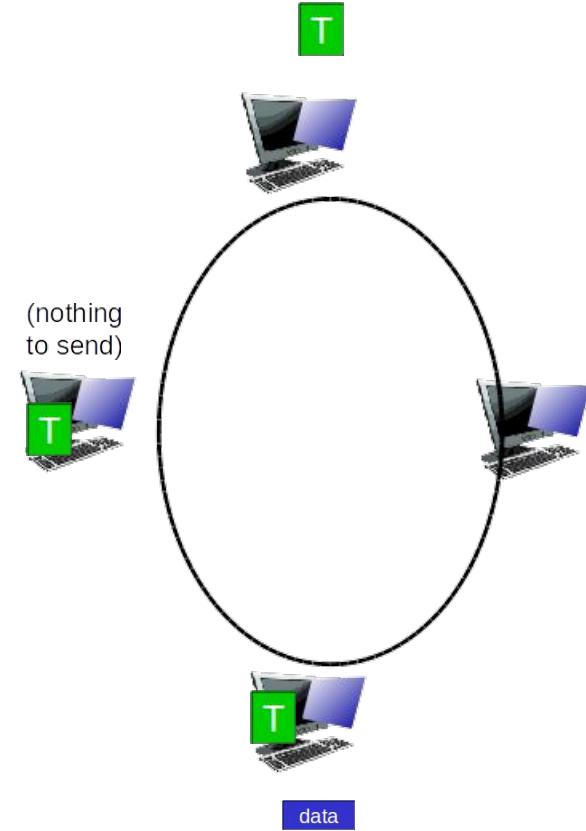
- Polling:
 - Centralized controller uses polling messages to “invite” client nodes to transmit in turn
 - Protocol needed for client devices to join/leave network
 - Pros:
 - Eliminates collisions and empty slots → higher efficiency
 - No collision
 - Cons:
 - polling overhead:
 - Only one active node
 - Access latency (wait for turn)
 - single point of failure (master)
 - 802.15 (wireless) protocol and Bluetooth protocol



“Taking turns” MAC protocols

token passing:

- control token passed from one node to next sequentially.
- Transmit while holding token
- Pros:
 - Decentralized
 - Highly efficient
- Cons:
 - Failure of a node → crash the entire channel
 - If a node does not release a token → recovery mechanisms
 - Access latency
- IEEE 802.5, FDDI (fiber distributed data interface)



“Taking turns” MAC protocols

- channel partitioning MAC protocols:
 - share channel *efficiently* and *fairly* at high load
 - inefficient at low load: delay in channel access, $1/N$ bandwidth allocated even if only 1 active node!
- random access MAC protocols
 - efficient at low load: single node can fully utilize channel
 - high load: collision overhead
- “**taking turns**” protocols
 - look for best of both worlds!
 - Channel allocated explicitly (no collisions)
 - Nodes won’t hold channel for long if nothing to send
 - Two approaches: **polling, token passing**

Question

Is the token-ring protocol inefficient if a LAN had a very large perimeter?

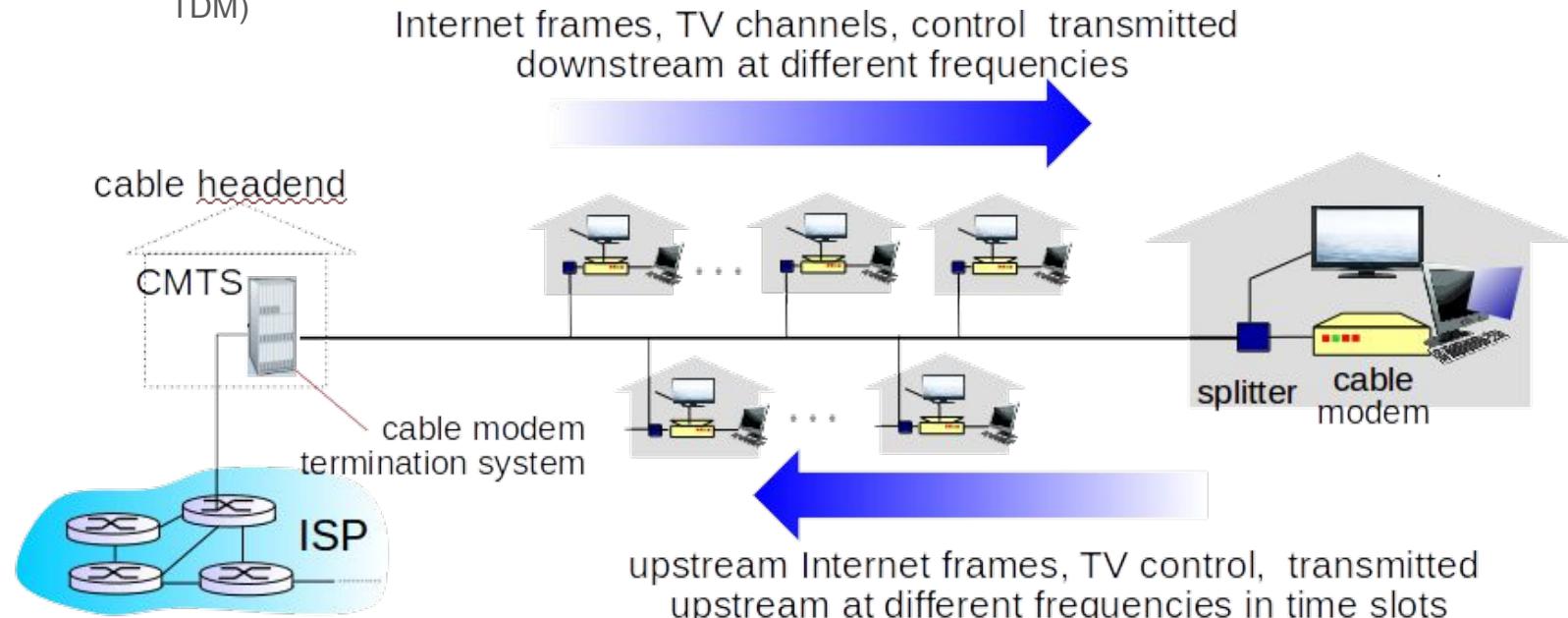
- True
- False

Summary of MAC protocols

- channel partitioning, by time, frequency
 - Time Division, Frequency Division
- 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
 - CSMA/CA used in 802.11
- taking turns
 - polling from central site, token passing
 - Bluetooth, FDDI, token ring

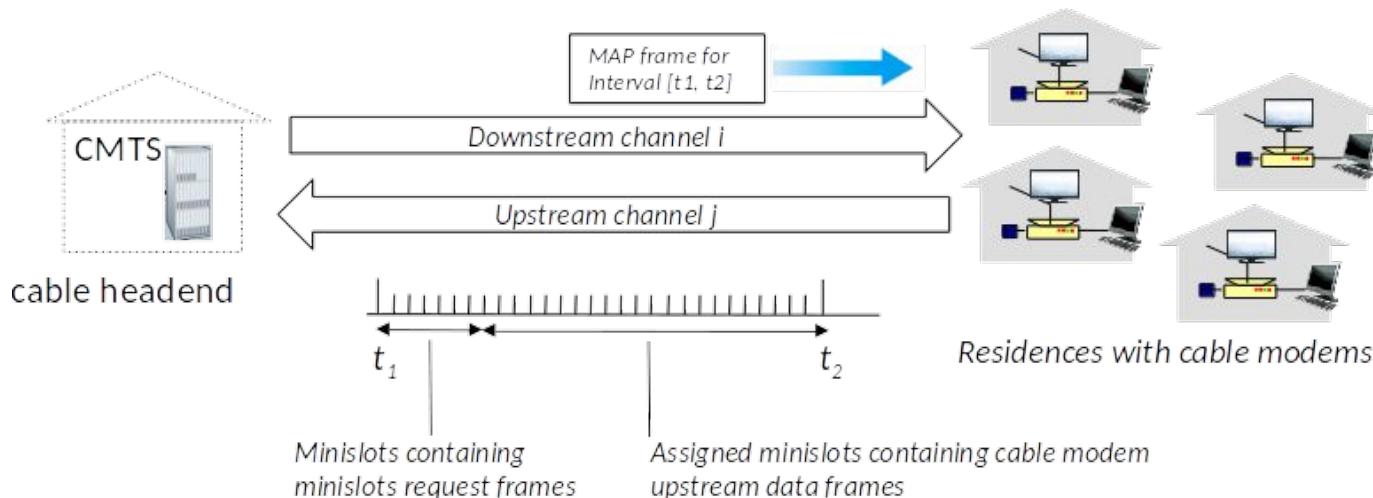
Case study: Cable access network

- **DOCSIS: Data Over Cable Service Interface Specification**
- **multiple** downstream (broadcast) channels (FDM)
 - single CMTS transmits into channels
- multiple upstream channels (FDM)
 - **multiple access:** all users contend (random access) for certain upstream channel time slots (others assigned TDM)



Case study: Cable access network

- FDM over upstream, downstream frequency channels
 - TDM upstream: some slots assigned, some have contention
 - downstream MAP frame: assigns upstream slots
 - request for upstream slots (and data) transmitted random access (binary backoff) in selected slots



Outline

- introduction, services
- error detection, correction
- multiple access protocols
- **LANs**
 - Ethernet
 - Switches
 - addressing, ARP
 - VLANs
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

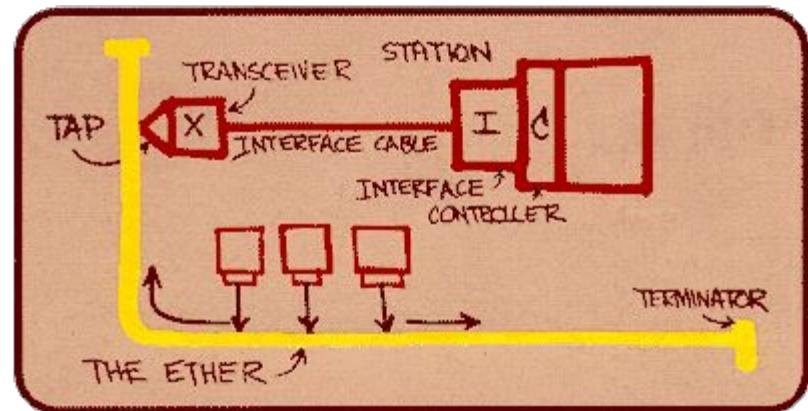
LAN Local Area Networks

- A LAN is a data communication system that allows a number of independent devices to communicate **directly** with each other in a limited geographic area.
 - Area diameter: few km
 - Owned by a single organization
- LANs are broadcast networks.
 - Shared medium
 - The medium access control (MAC) protocols are to coordinate the access to the channel
- Example of LAN protocols
 - Ethernet (802.3)
 - Token bus (802.4)
 - Token ring (802.5)
 - Wireless LANs (802.11)

Ethernet

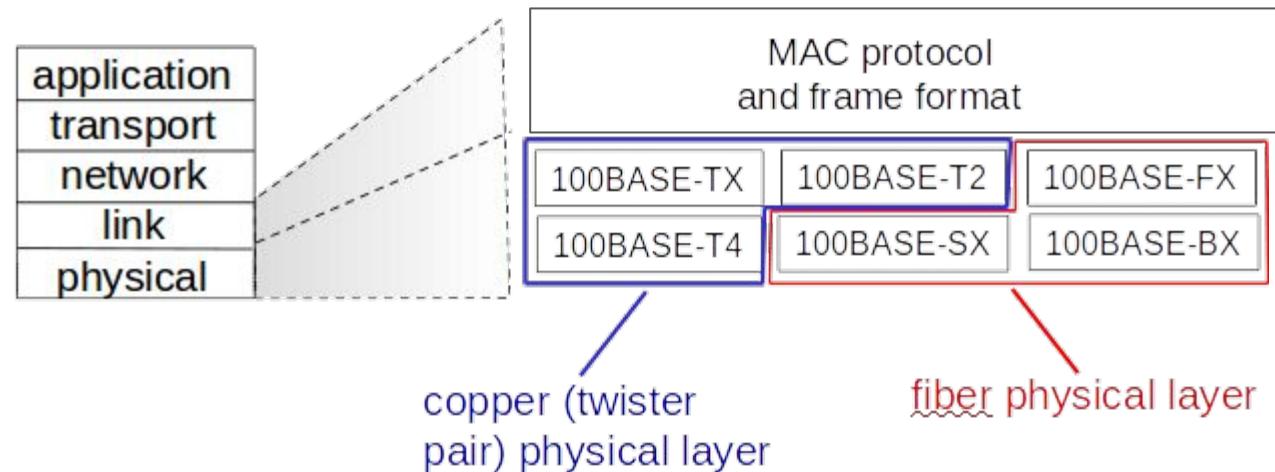
- “dominant” wired LAN technology:
- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 10 Gbps

Original pictures drawn by Bob Metcalfe,
co-inventor of Ethernet (1972–Xerox PARC)



Ethernet

- Ethernet is both a **link-layer** and a **physical layer** specification
- There are many different Ethernet standards
 - Common MAC protocol (CSMA/CS) and frame format
 - different physical layer media: cable (Coaxial cable, copper wire), fiber
 - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps



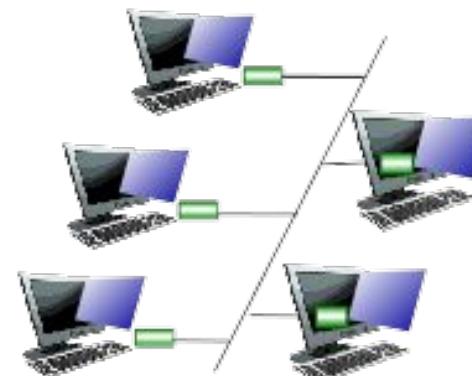
Ethernet standards acronyms

- Ethernet technologies standardized by IEEE 802.3 CSMA/CD working group
- **X-BASE-Y**
- **X**: speed of the standard: 10, 100, 1000, 10G, 10 Megabit, 100 Megabit, Gigabit, 10 Gigabit, 100 Gigabit, 40 Gigabit
- **BASE**: baseband ethernet:
 - the physical media carries only ethernet traffic: all of 802.3 standards are for baseband ethernet
- **Y**: physical media itself
 - 5: 10BASE-5: original ethernet: large thick coaxial cable
 - 2: 10BASE-2: Thin coaxial cable version
 - T: 10BASE-T: twisted-pair copper wires
 - F: 10BASE-F: Two optical fibers in a single cable

Ethernet Evolution: Physical topology: Bus

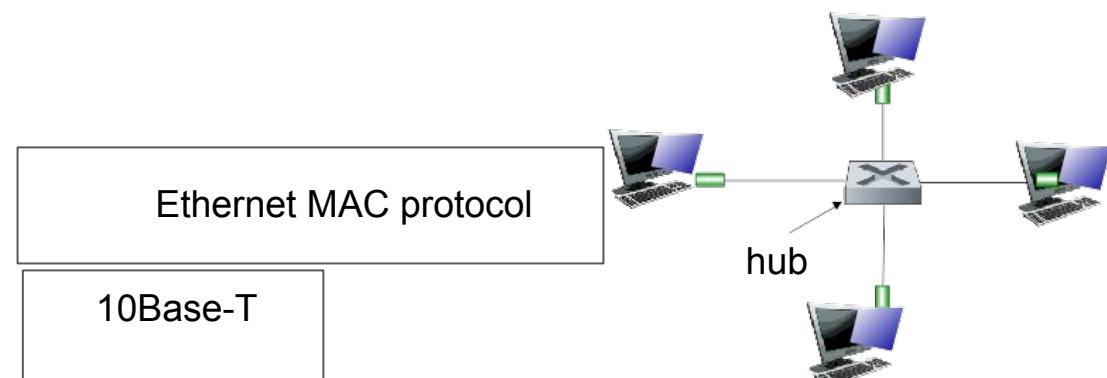
- **bus**: popular through mid 90s
 - all nodes in same collision domain (can collide with each other)
 - CSMA/CD was used for sharing access to the link

Ethernet MAC protocol (CSMA/CD)	
10Base-5	10Base-2



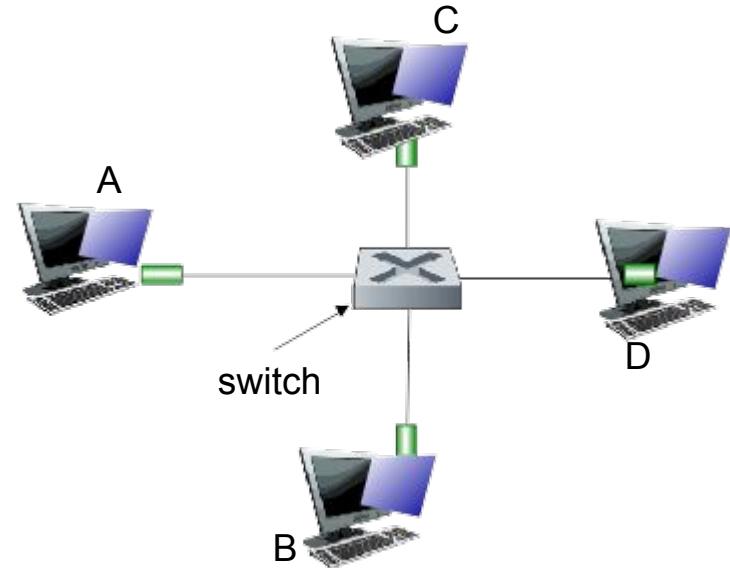
Ethernet Evolution: Hub

- By the late 1990, bus topology was replaced with a hub-based star topology
- A hub is a physical-layer device that acts on individual bits rather than frames.
 - Whenever a hub receives a bit from one of its interfaces, it re-creates the bit, boost its energy, transmit it over all the other interfaces: repeater
- A broadcast LAN
 - Collisions: If a hub receives frames from two different interfaces at the same time, a collision occurs and the nodes that created the frame must retransmit



Ethernet Evolution: Hub replaced with switch

- **switched:** prevails today
 - active link-layer 2 switch in center
 - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)
- Elimination of collisions
 - Multiple concurrent communication (full-duplex links)
 - Never transmits more than one frame on a segment
 - Maximum aggregated throughput of a switch is the sum of all the switch interface rates
- Heterogenous links
 - Different links can have different speed and run over different media
- Management
 - A cable cut disconnects only the host using the cable
- Store and forward ethernet frame



Ethernet Frame Structure



1. **preamble:** 7 bytes with pattern 10101010 followed by one byte with pattern 10101011 used to synchronize receiver, sender clock rates
2. Addresses (**DA, SA**): 6 byte source, destination globally unique MAC addresses assigned by manufacturer: if adapter receives frame with matching destination address, or with broadcast address, it passes data in frame to network layer protocol otherwise, adapter discards frame
3. **type:** indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX)
4. **Pad:** zeroes used to ensure minimum frame length
5. **CRC:** cyclic redundancy check: check sequence to detect bit errors: frame is dropped

Question

What is the difference between frame structure for 10BASE-T, 100BASE-T and gigabit ethernet?

- A. 10BASE-T has smaller frame size than 100BASE-T
- B. All have identical frame structure
- C. 10BASE-T use MAC addresses but 100BASE-T does not

Ethernet: unreliable, connectionless

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

Outline

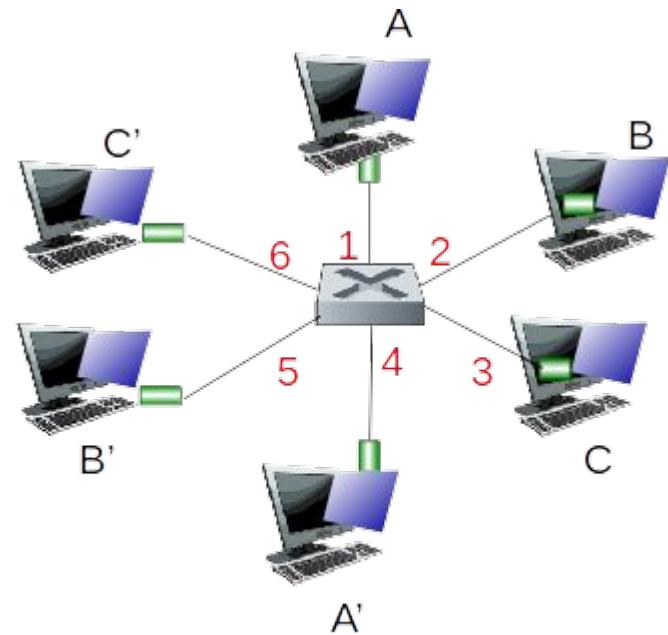
- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - Ethernet
 - **Switches**
 - addressing, ARP
 - VLANs
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

Ethernet switch

- Switch is a **link-layer** device: takes an **active** role
 - store, forward Ethernet frames
 - examine incoming frame's MAC address, **selectively** forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- **transparent**: hosts unaware of presence of switches
- **plug-and-play, self-learning**
 - switches do not need to be configured

Switch: 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
- **Multiple simultaneous transmissions:**
 - A-to-A' and B-to-B' can transmit simultaneously, without collisions



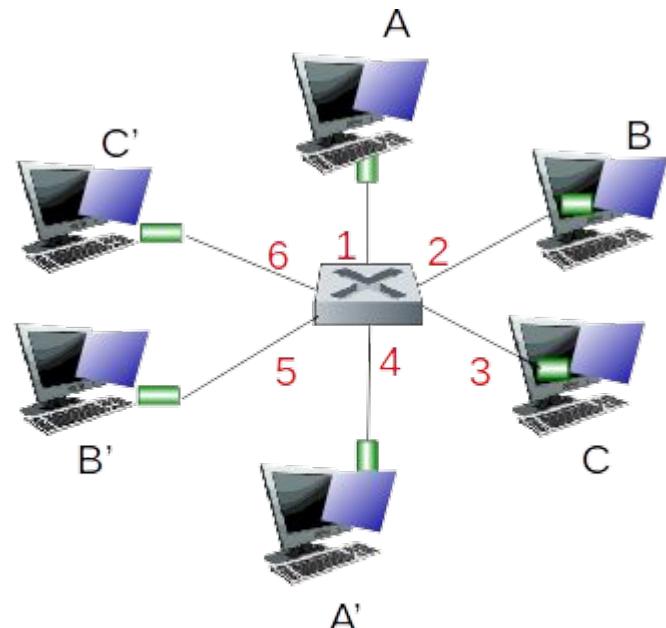
*switch with six interfaces
(1,2,3,4,5,6)*

Switch: forwarding

- Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?
- A: each switch has a **switch table**, each entry:
 - (MAC address of host, interface to reach host, time stamp)
 - looks like a routing table!

MAC <u>addr</u>	interface	TTL

*Switch table
(initially empty)*



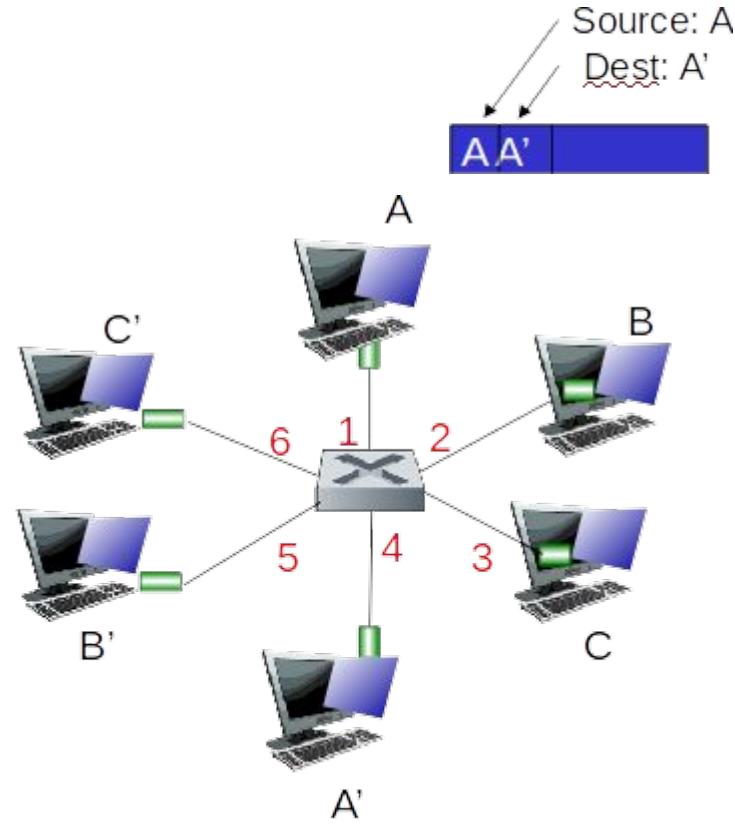
*switch with six interfaces
(1,2,3,4,5,6)*

Switch: self-learning

- Q: how are entries created, maintained in switch table?
 - something like a routing protocol?
- **switch learns** which hosts can be reached through which interfaces
 - when frame received, switch “learns” location of sender: incoming LAN segment
 - records sender/location pair in switch table

MAC <u>addr</u>	interface	TTL

*Switch table
(initially empty)*



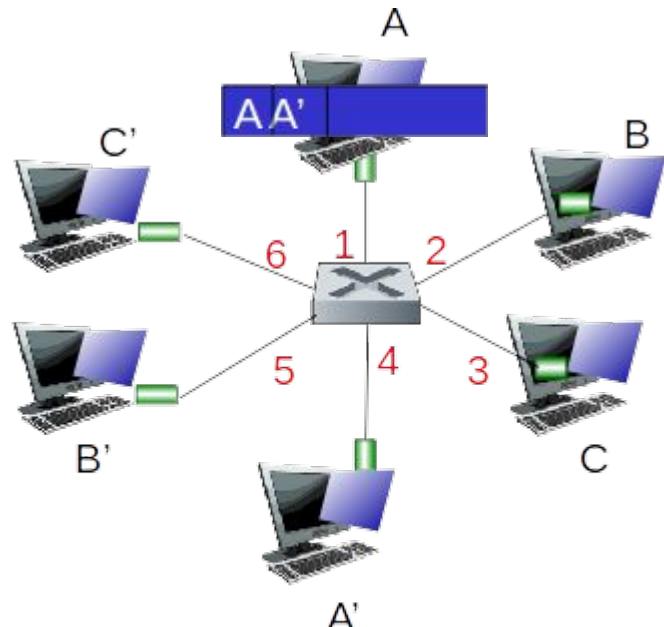
*switch with six interfaces
(1,2,3,4,5,6)*

Self-learning, forwarding: example

- frame destination, A', location unknown:
 - Flood
- destination A location known:
 - Selectively send on just one link

*switch table
(initially empty)*

MAC <u>addr</u>	interface	TTL
A	1	60
A'	4	60



*switch with six interfaces
(1,2,3,4,5,6)*

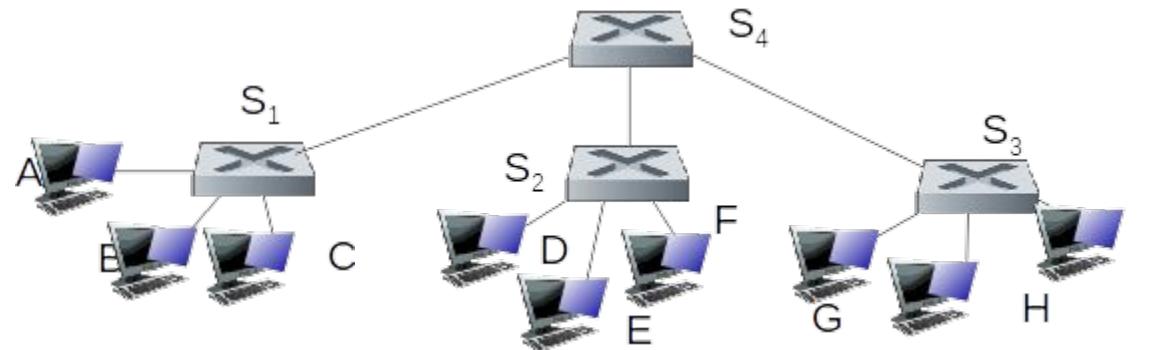
Switch: frame filtering/forwarding

when frame received at switch:

```
record incoming link, MAC address of sending host
index switch table using MAC destination address
if entry found for destination then {
    if destination on segment from which frame arrived then
        drop frame
    else
        forward frame on interface indicated by entry
}
else
    flood /* forward on all interfaces except arriving interface */
```

Interconnecting switches

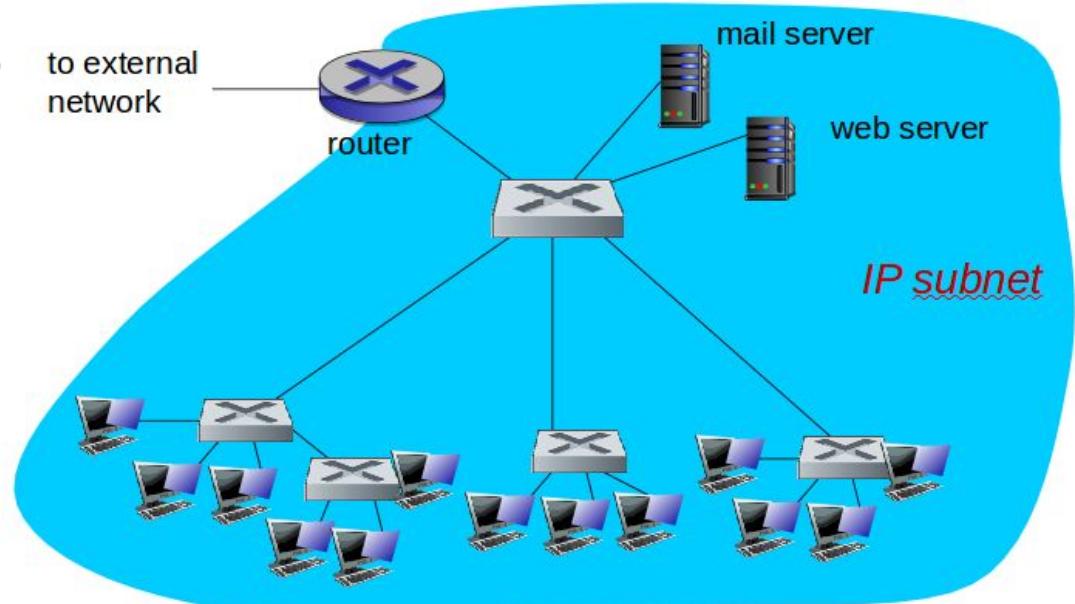
self-learning switches can be connected together:



- Q: sending from A to G - how does S1 know to forward frame destined to G via S4 and S3?
- A: self learning! (works exactly the same as in single-switch case!)

Institutional network

- Ethernet switches can connect many hosts
 - There are switches available with hundreds of ports.



Outline

- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - Ethernet
 - Switches
 - **addressing, ARP**
 - VLANs
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

MAC Addresses (LAN Addresses)

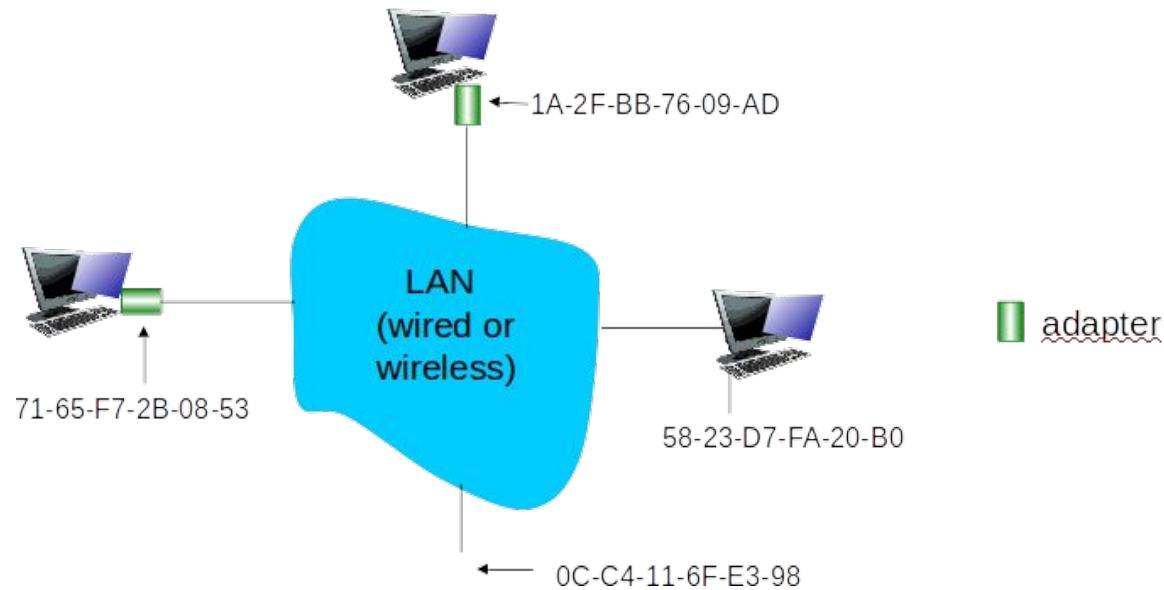
- 48 bit address:
 - 2^{48} possible MAC addresses
 - e.g.: 1A-2F-BB-76-09-AD
 - Expressed in hexadecimal (base 16) notation (each “numeral” represents 4 bits)
 - MAC address burned in NIC ROM, also sometimes software settable
- **MAC (or LAN or physical or Ethernet) address:**
 - function: used ‘locally’ to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)

LAN addresses (more)

- MAC addresses are unique
- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- MAC flat address → portability
 - can move LAN card from one LAN to another
- IP hierarchical address not portable
 - address depends on IP subnet to which node is attached
- Analogy:
 - MAC address: like Social Security Number
 - IP address: like postal address

LAN addresses

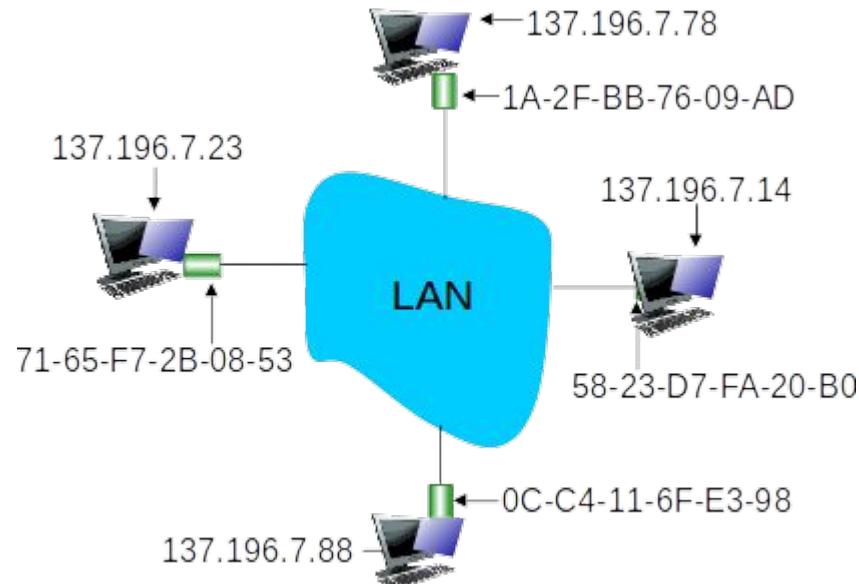
- each interface on LAN
 - has unique MAC address
 - has a locally unique 32-bit IP address (as we've seen)
- **Link layer switches do not have link layer addresses**



ARP: Address Resolution Protocol

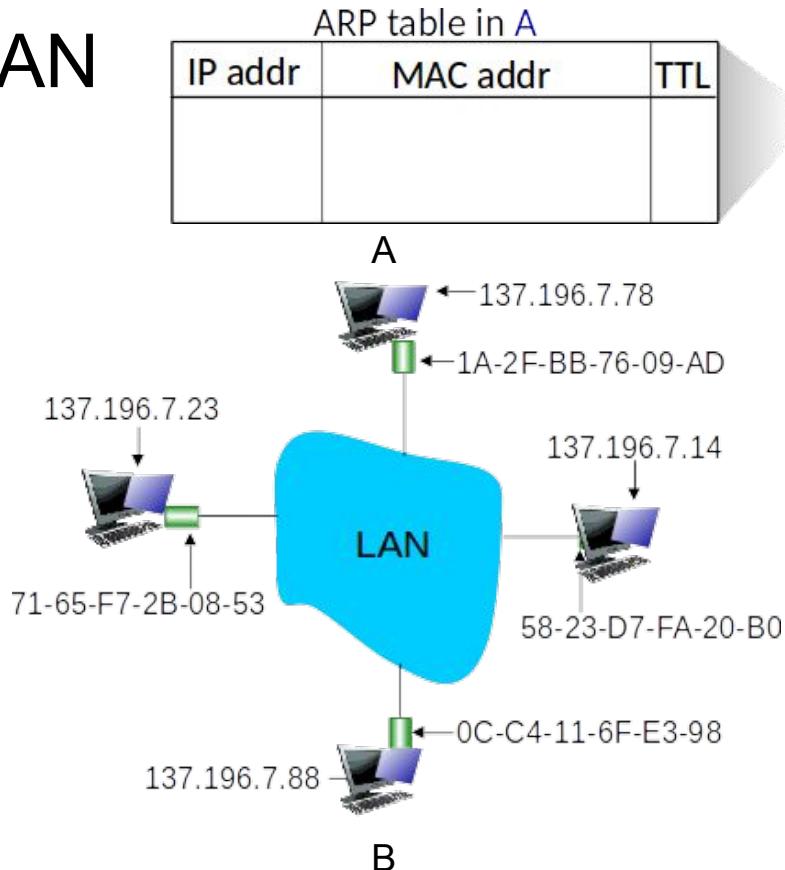
Question: how to determine interface's MAC address, knowing its IP address?

- **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)
 - soft state: information that times out (goes away) unless refreshed



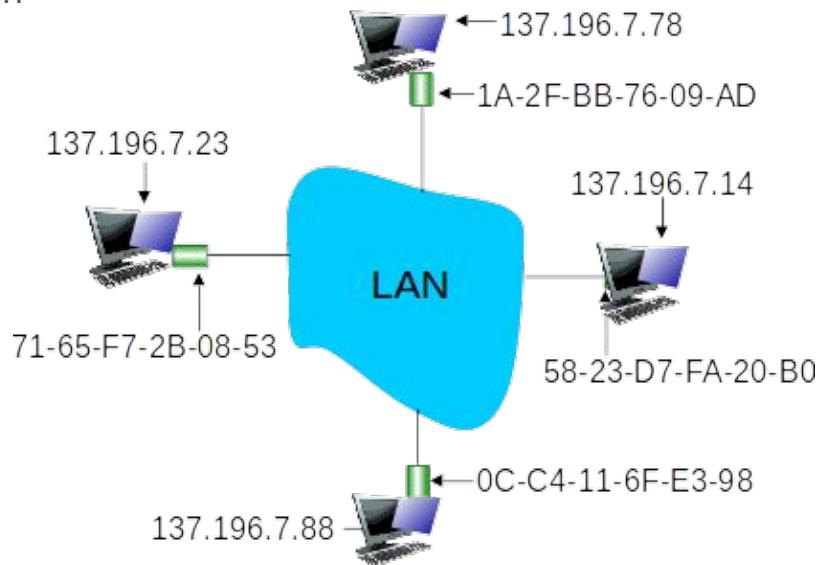
ARP protocol in action: same LAN

- A wants to send datagram to B (MAC address of B not in ARP table of A)
 - A broadcasts **ARP query** packet, containing B's IP address
 - destination MAC address = FF-FF-FF-FF-FF-FF
 - all nodes on LAN receive ARP query
 - B receives ARP packet, replies to A with its (B's) MAC address
 - frame sent to A's MAC address (unicast)
 - A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)

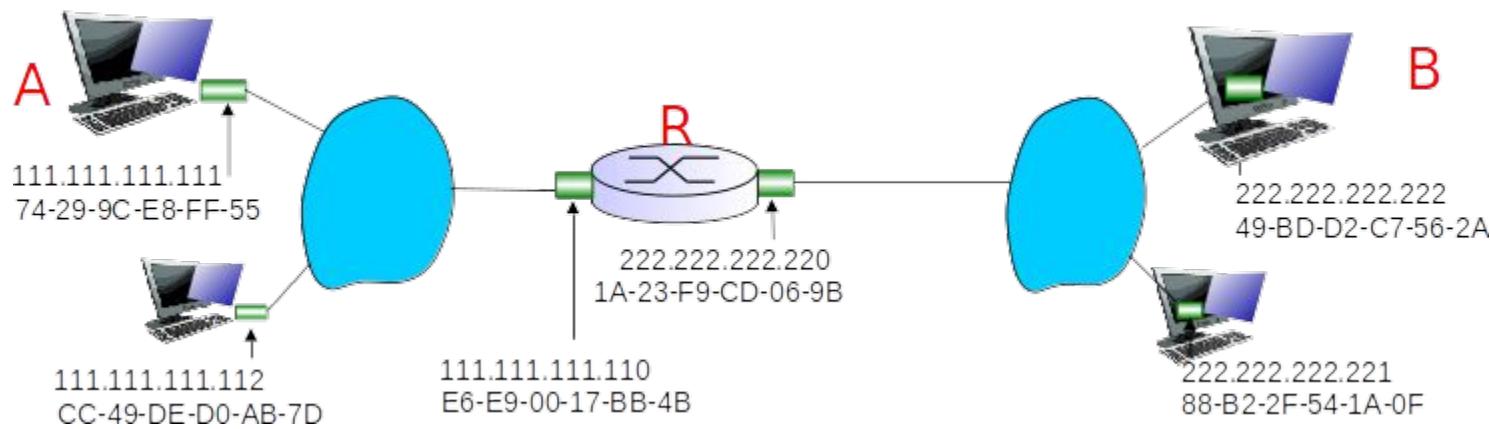


ARP protocol: same LAN

- ARP is “plug-and-play”:
 - nodes create their ARP tables without intervention from net administrator



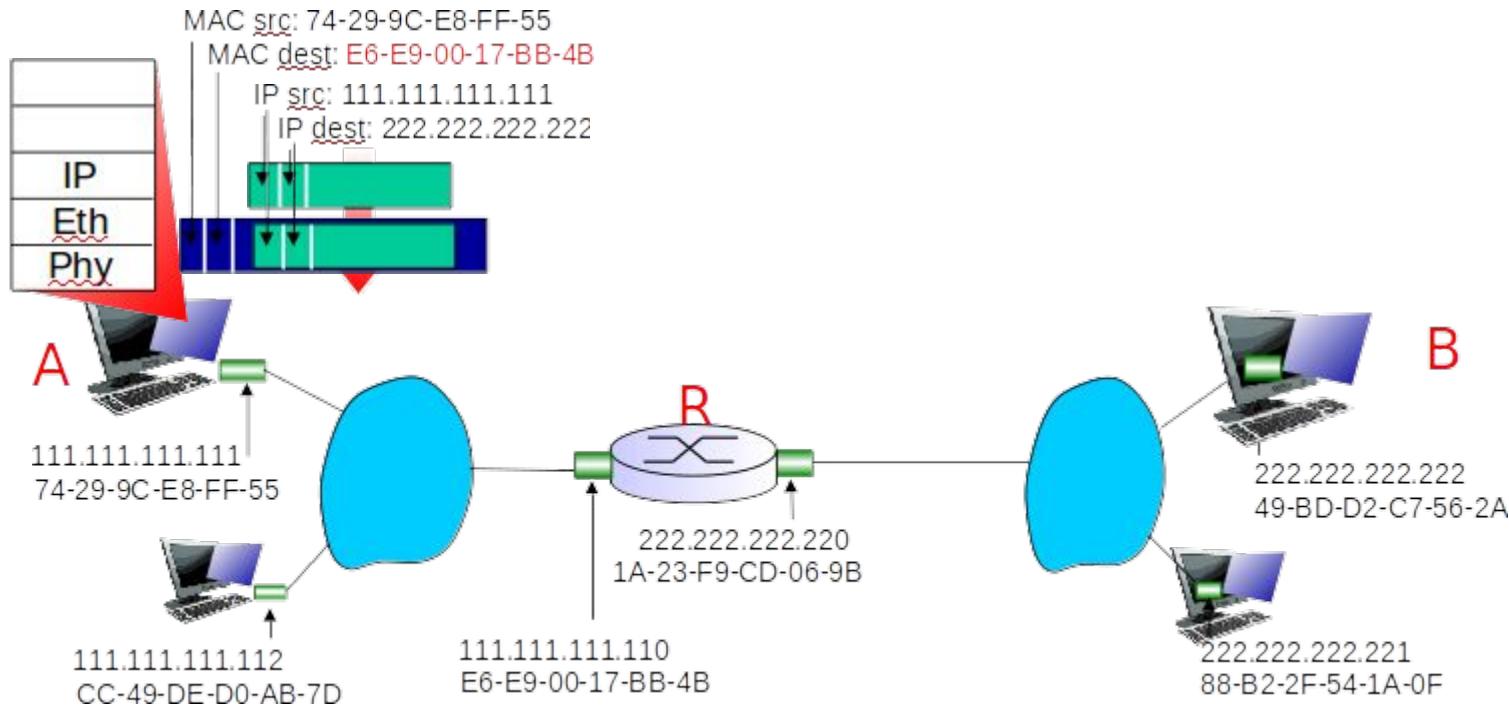
Addressing: routing to another subnet



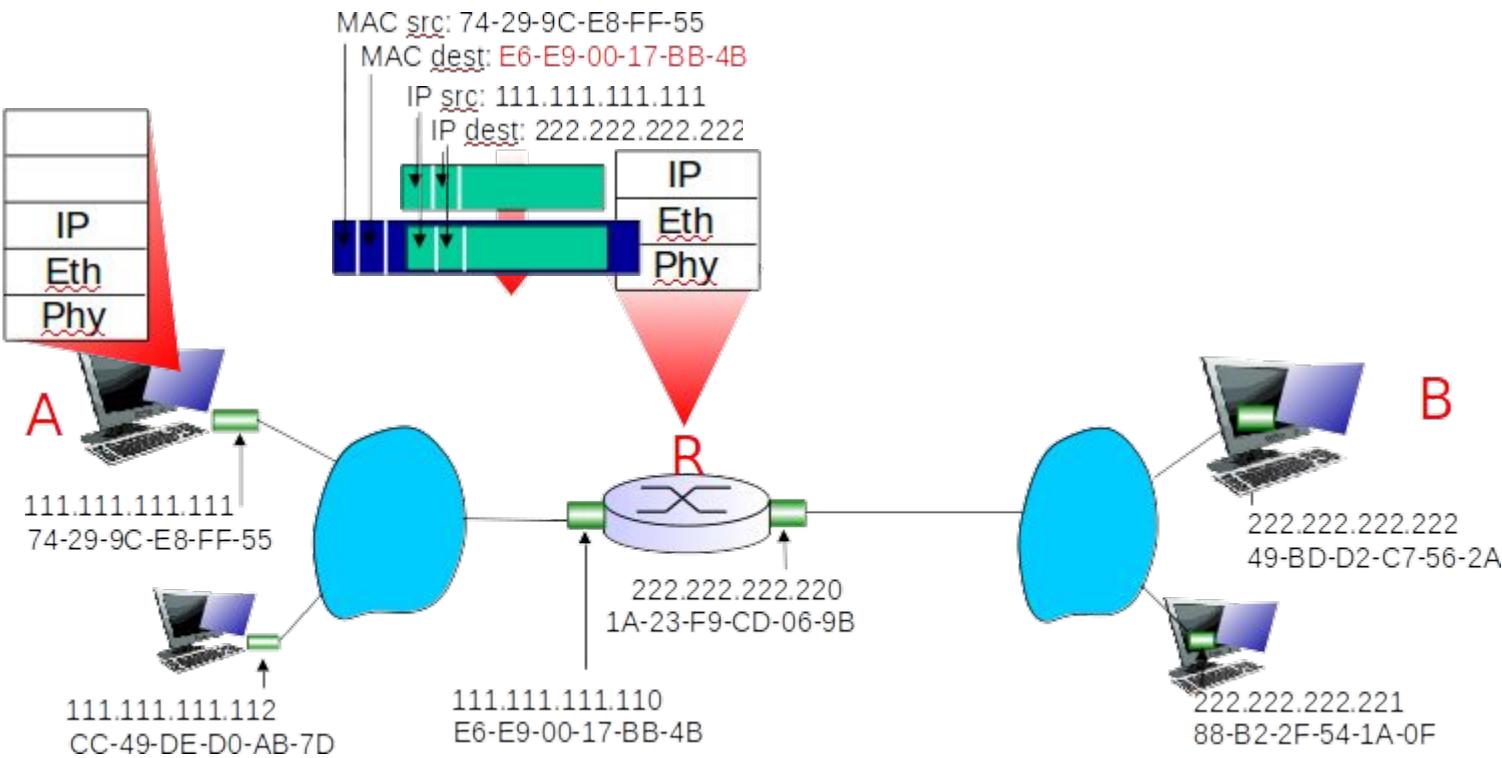
Addressing: routing to another LAN

- walkthrough: send datagram from A to B via R
 - focus on addressing – at IP (datagram) and MAC layer (frame)
 - 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?)
 - DHCP reply contained the IP address of the gateway
 - An ARP request is done to get the MAC oaddress of the router

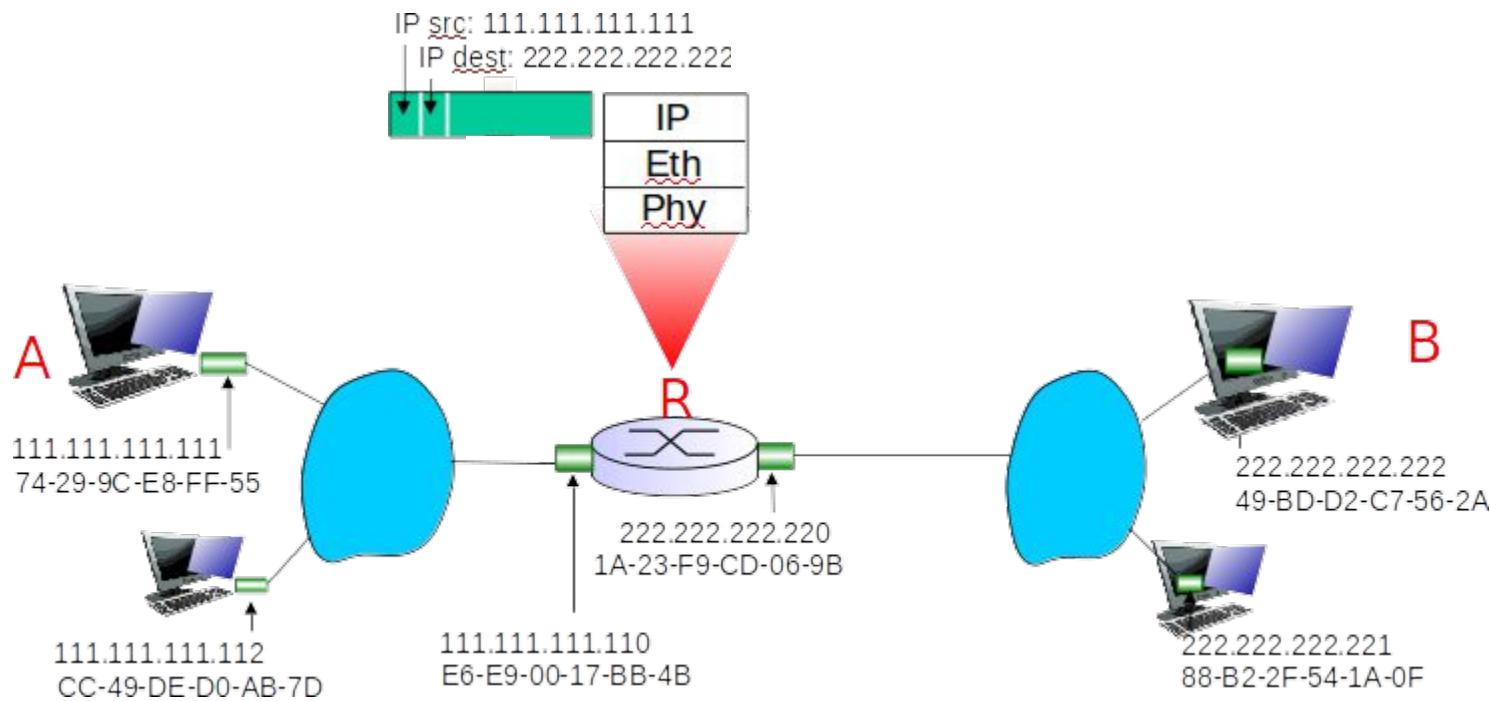
- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram



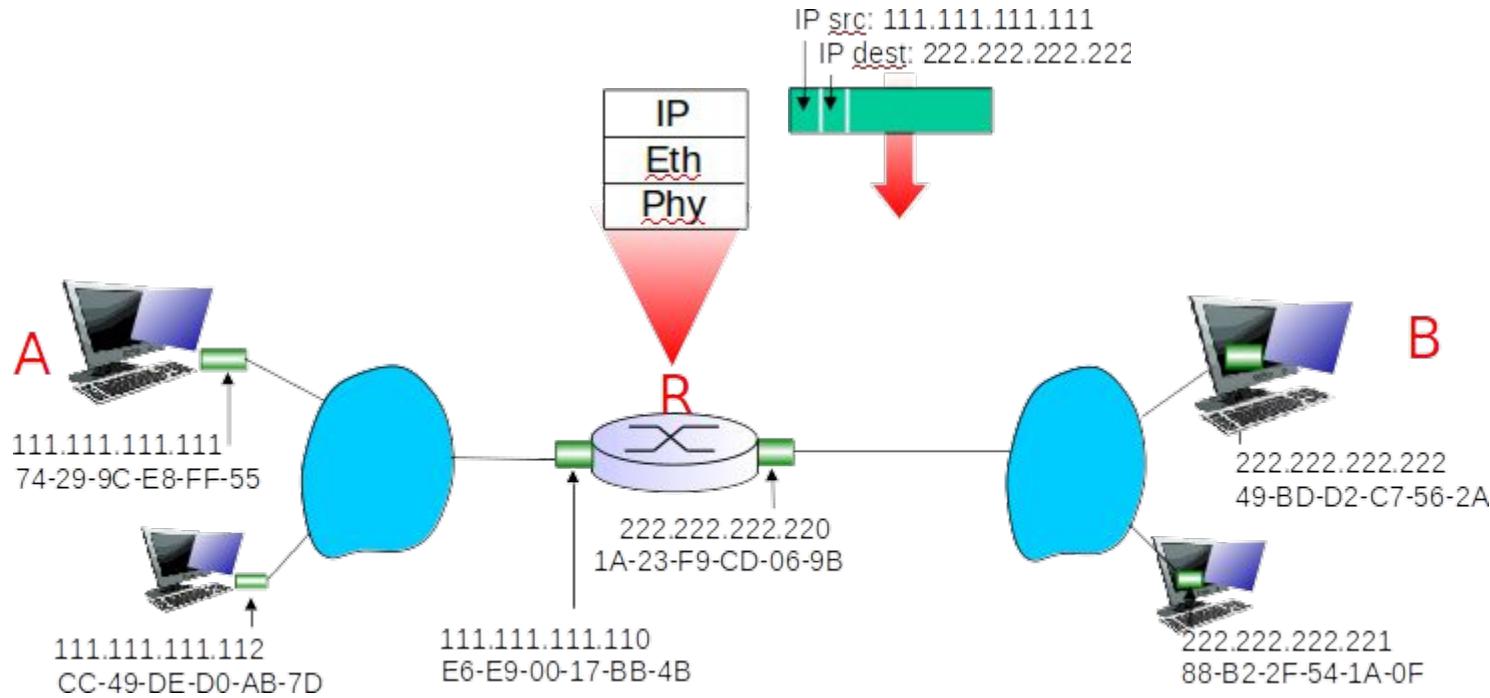
- Frame sent from A to R



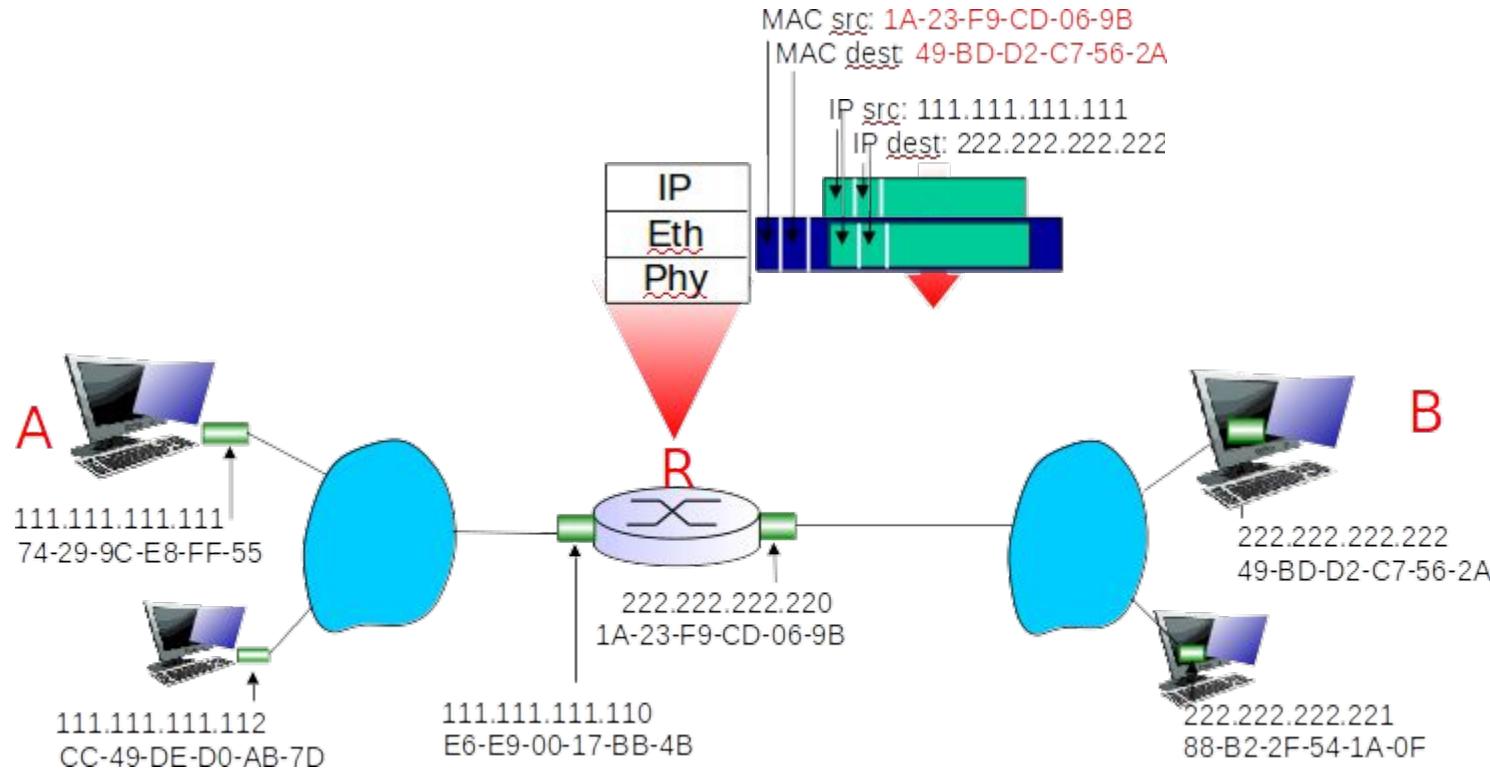
- Frame received at R, datagram removed, passed up to IP



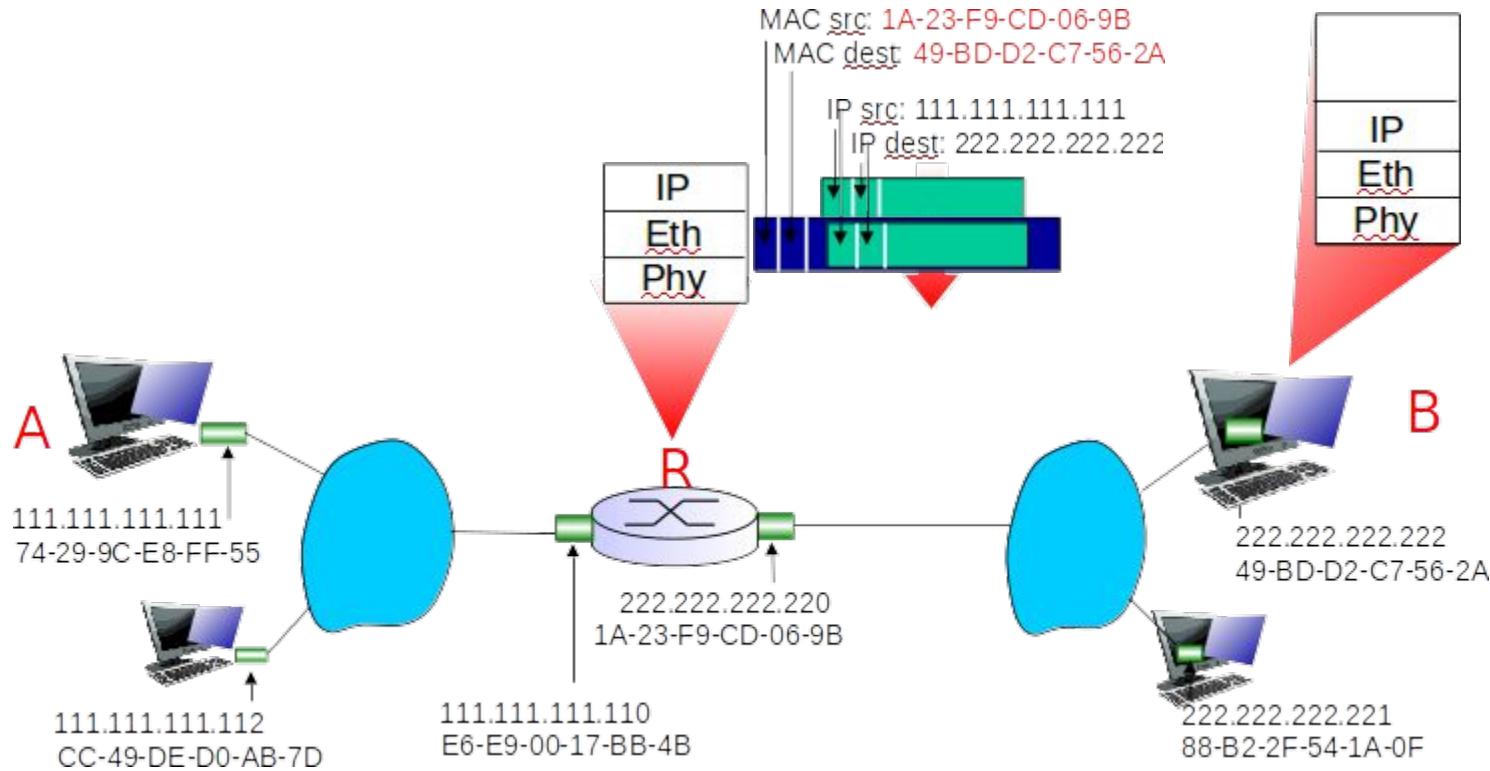
- R forwards datagram with IP source A, destination B



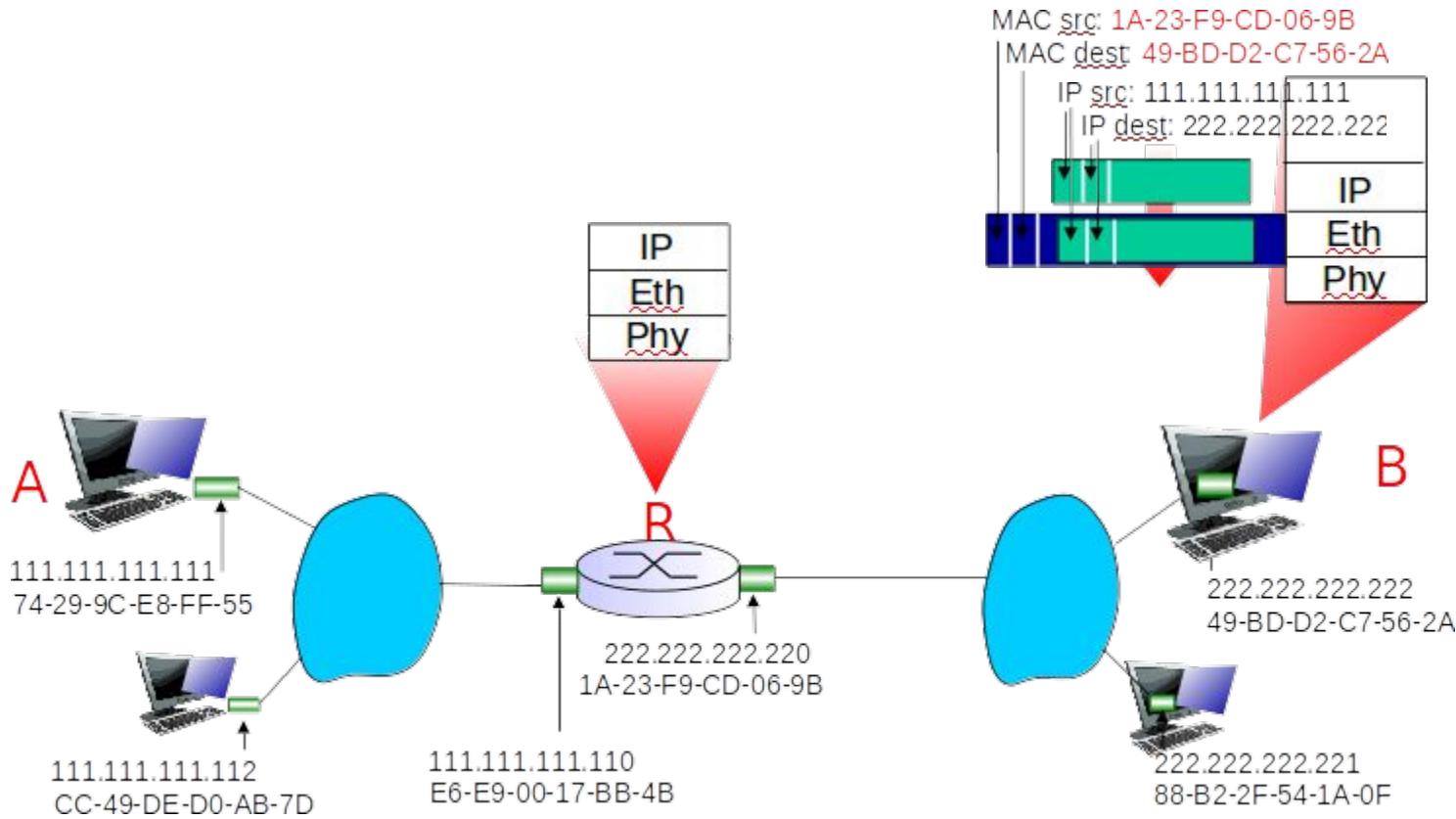
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



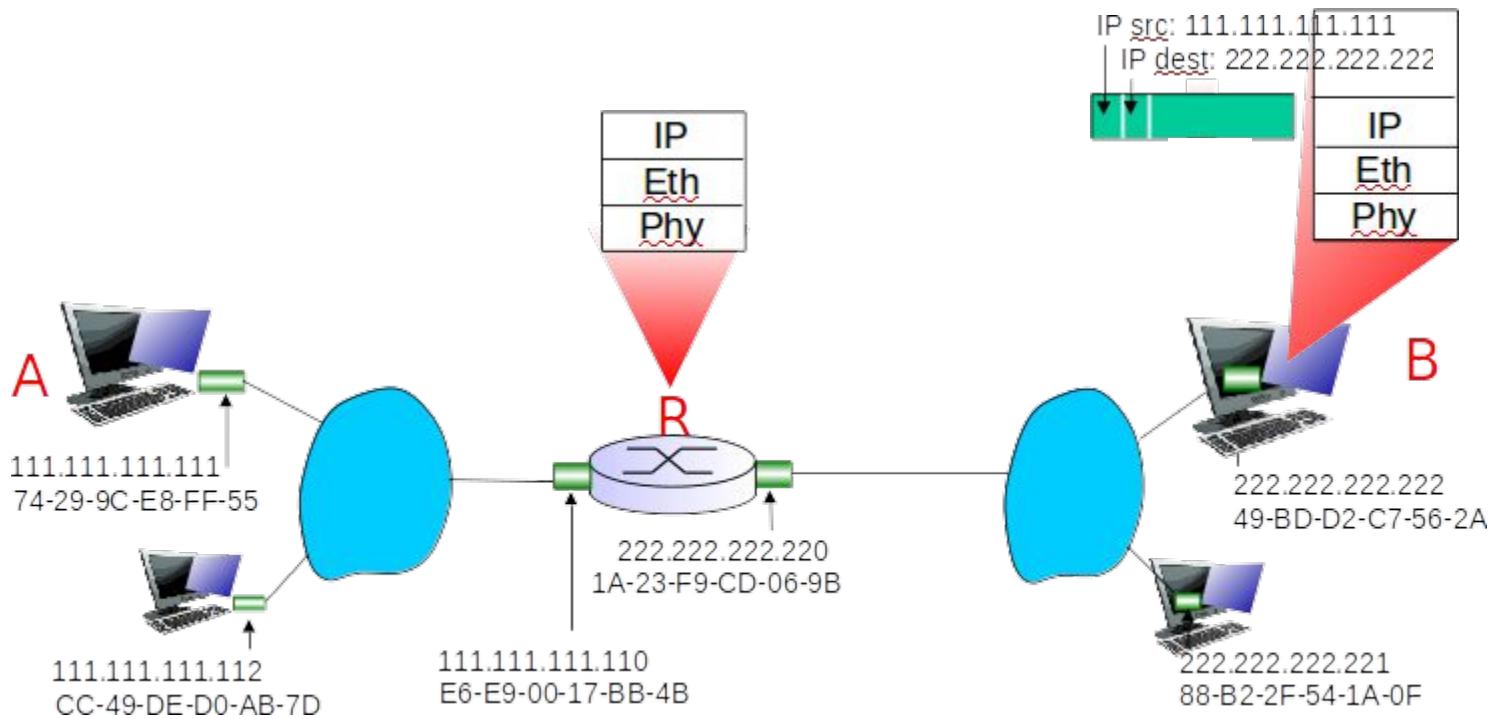
- Frame sent from R to B



- frame received at B

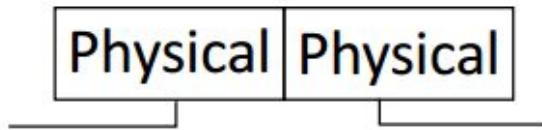


- frame received at B, datagram removed, passed up to IP

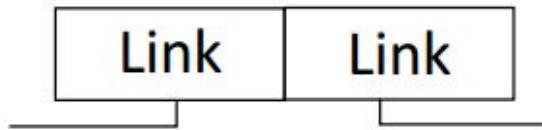


Hub vs switch vs. router

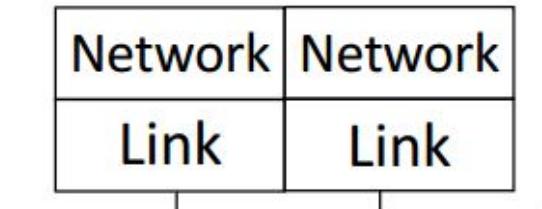
Hub, or
repeater



Switch



Router

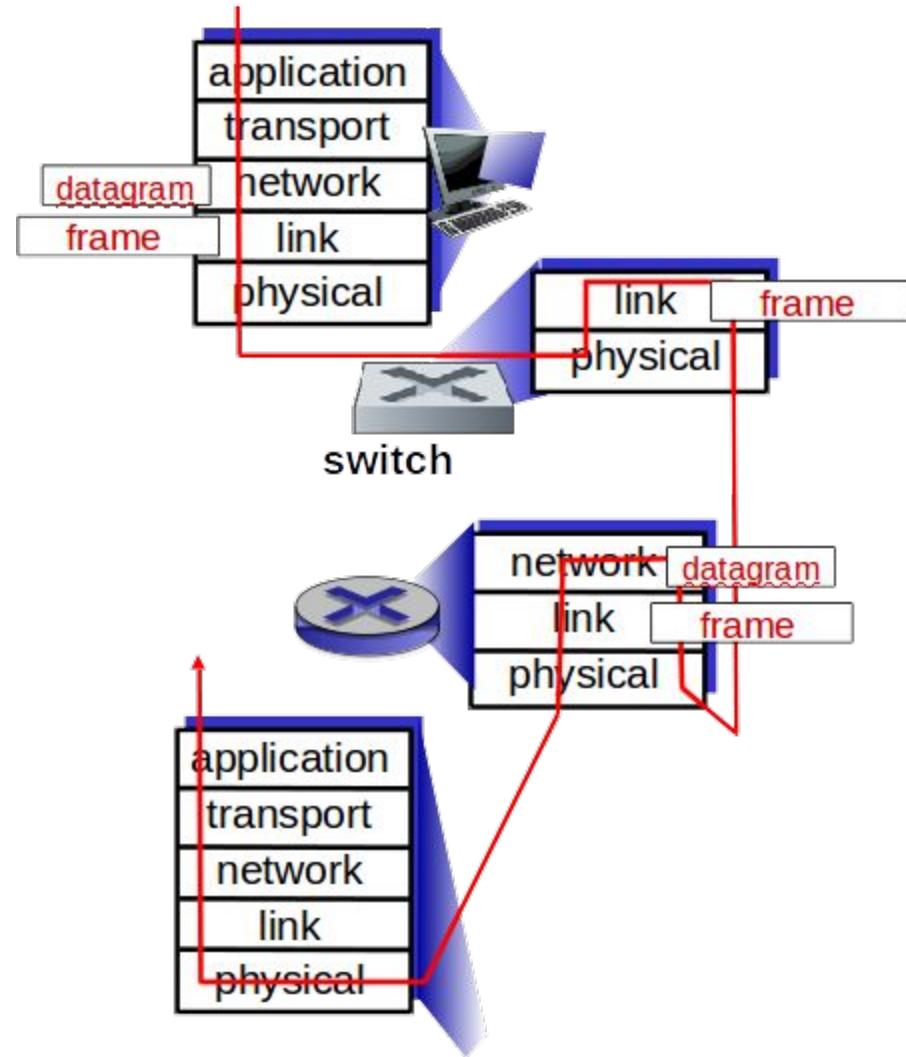


All look like this:



Switches vs. routers

- both are store-and-forward:
 - routers: network-layer devices (examine network-layer headers)
 - switches: link-layer devices (examine link-layer headers)
- both have forwarding tables:
 - routers: compute tables using routing algorithms, IP addresses
 - switches: learn forwarding table using flooding, learning, MAC addresses



Switches

- Pros:
 - plug-and -play devices
 - Process frames only up through layer 2, whereas routers have to process datagrams up through layer 3
- Cons:
 - The active topology of a switched network is restricted to a spanning tree
 - To prevent the cycling of frames
 - Large switch network → large ARP table → large ARP traffic
 - Prone to broadcast storms
 - If one host transmits an endless stream of ethernet broadcast frames, the switches will forward all of these frames→ the entire network collapse

Routers

- Pros:
 - Packets do not normally cycle through routes even when the network has redundant paths
 - Because network addressing is hierarchical (not flat as in MAC addresses)
 - Unless routers are misconfigured
 - TTL field in IP limits cycling
 - Packets are not restricted to spanning tree and can use the best path
 - Firewall protection against broadcast storm
- Cons:
 - Not a plug-and-play device
 - The routers and the host that connect to them need their IP addresses to be configured
 - Larger per-packet processing time than switches

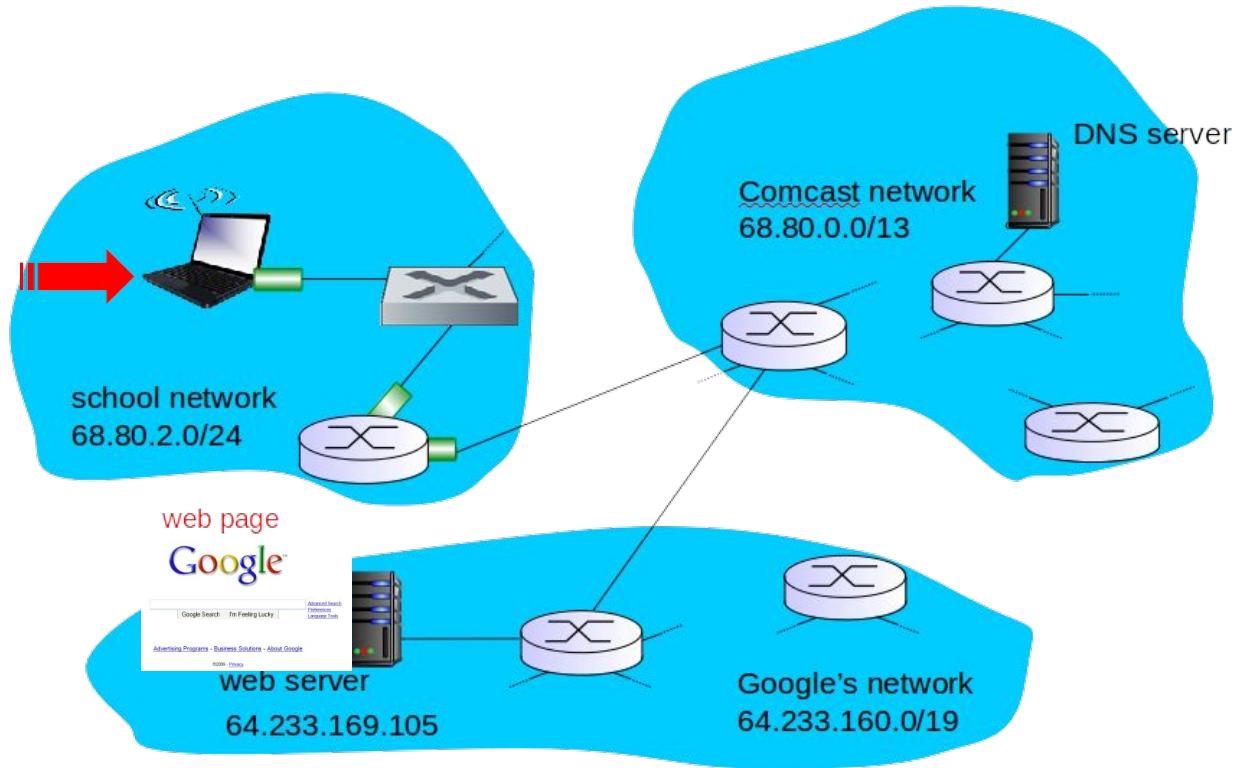
Outline

- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - Ethernet
 - Switches
 - addressing, ARP
 - VLANs
- **a day in the life of a web request**
- link virtualization: MPLS
- data center networking

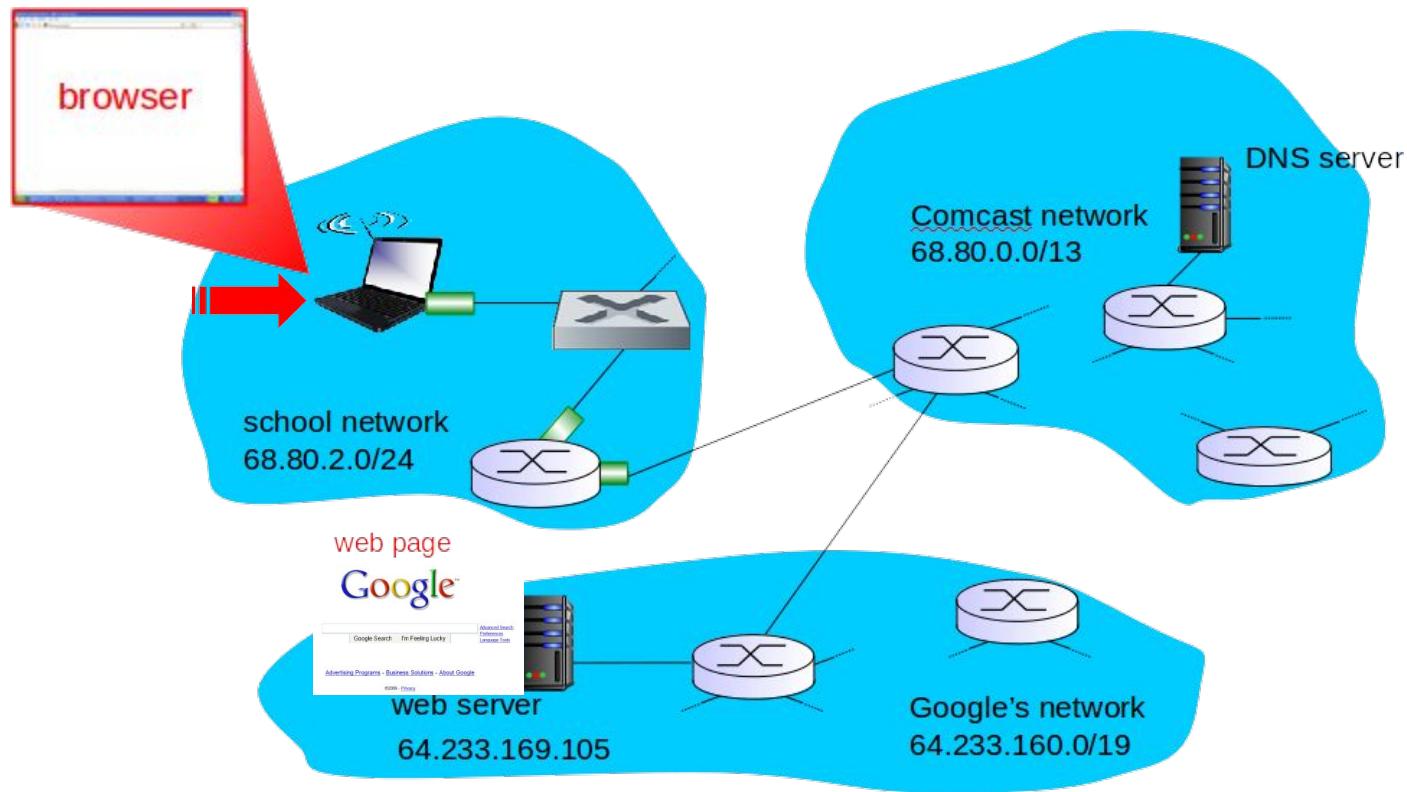
Synthesis: a day in the life of a web request

- journey down protocol stack complete!
 - application, transport, network, link
- putting-it-all-together: synthesis!
 - **goal:** identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
 - **scenario:** student attaches laptop to campus network, requests/receives www.google.com

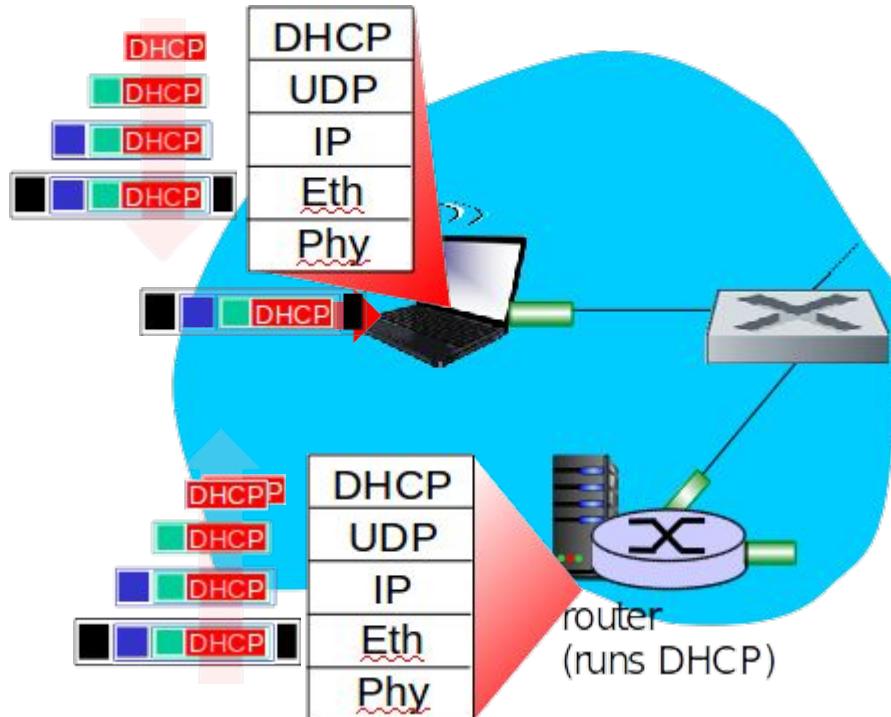
A day in the life: scenario



A day in the life: scenario

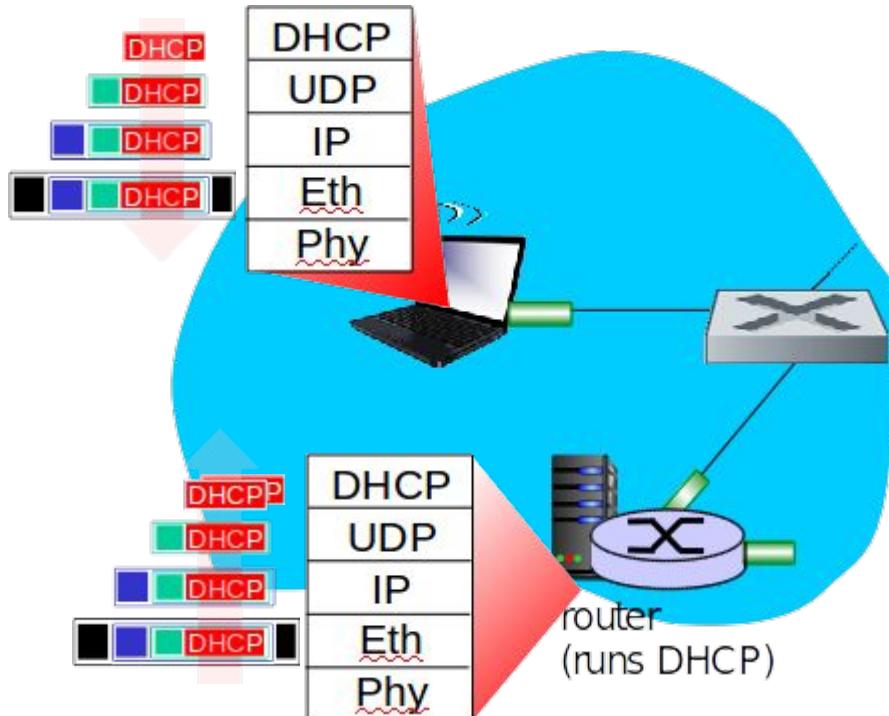


A day in the life... connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use **DHCP**
- DHCP request encapsulated in **UDP**, **encapsulated** in **IP**, encapsulated in **802.3** Ethernet
- Ethernet frame **broadcast** (dest: FFFFFFFFFFFF) on LAN, received at router running **DHCP** server
- Ethernet **demuxed** to IP demuxed, UDP demuxed to DHCP

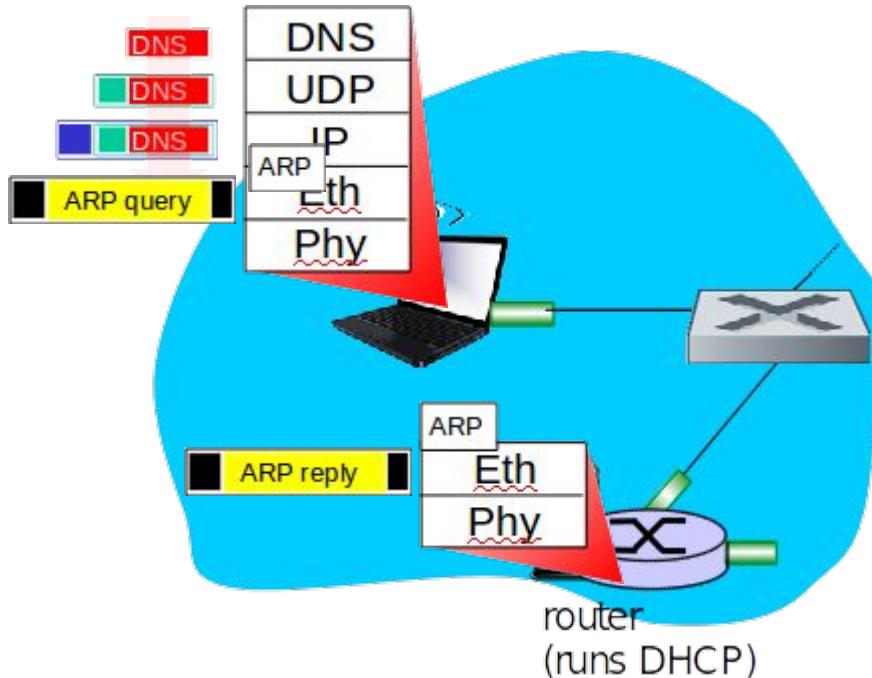
A day in the life... connecting to the Internet



- DHCP server formulates **DHCP ACK** containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (**switch learning**) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

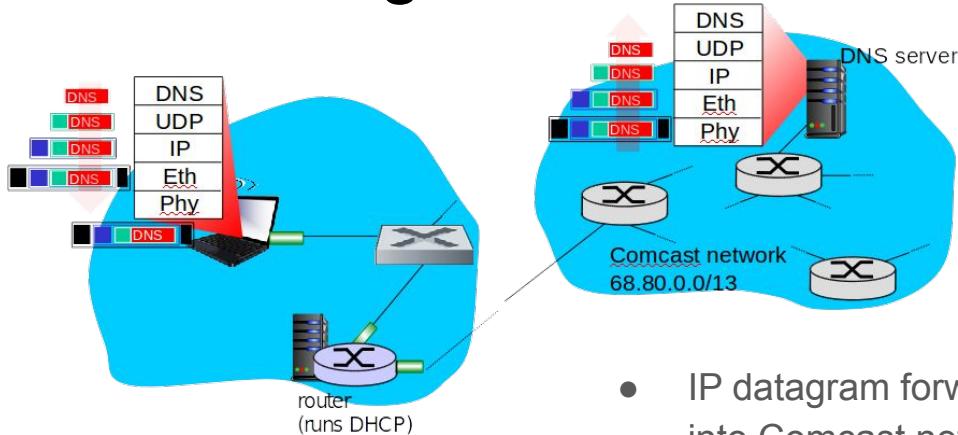
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life... connecting to the Internet



- before sending **HTTP** request, need IP address of www.google.com: **DNS**
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: **ARP**
- **ARP query** broadcast, received by router, which replies with **ARP reply** giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

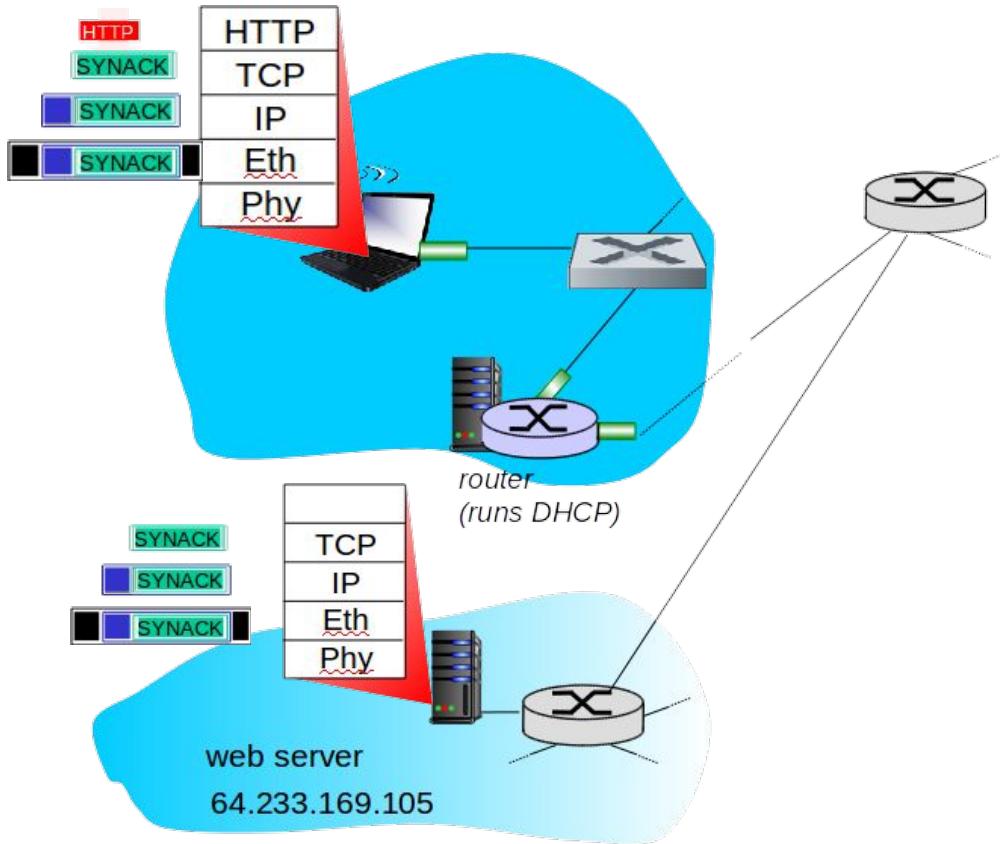
A day in the life... using DNS



IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

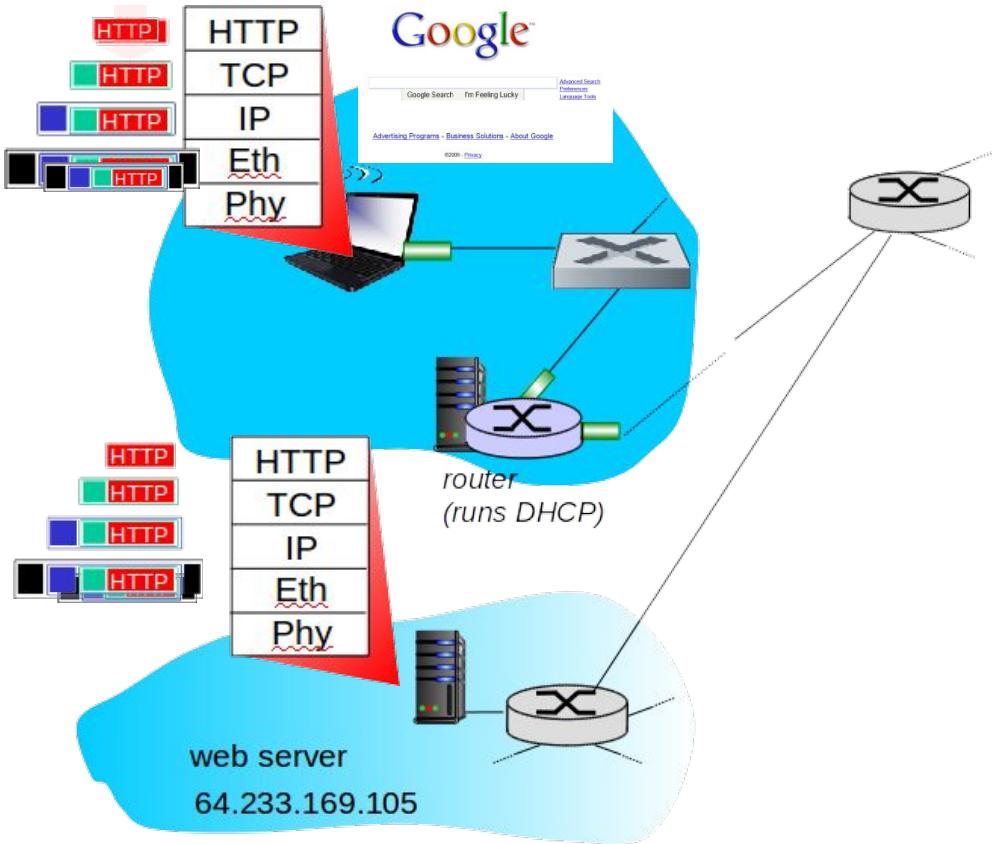
- IP datagram forwarded from campus network into Comcast network, routed (tables created by **RIP**, **OSPF**, **IS-IS** and/or **BGP** routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com

A day in the life...TCP connection carrying HTTP



- to send HTTP request, client first opens **TCP socket** to web server
- TCP **SYN segment** (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with **TCP SYNACK** (step 2 in 3-way handshake)
- TCP **connection established!**

A day in the life... HTTP request/reply



- **HTTP request** sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with **HTTP reply** (containing web page)
- IP datagram containing HTTP reply routed back to client

let's take a breath

- journey down protocol stack complete (except PHY)
- solid understanding of networking principles, practice
- could stop here but lots of interesting topics!
 - Wireless
 - security

Outline

- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - Ethernet
 - Switches
 - addressing, ARP
 - **VLANS**
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

Outline

- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - Ethernet
 - Switches
 - addressing, ARP
 - VLANs
- link virtualization: MPLS
- **data center networking**
- a day in the life of a web request

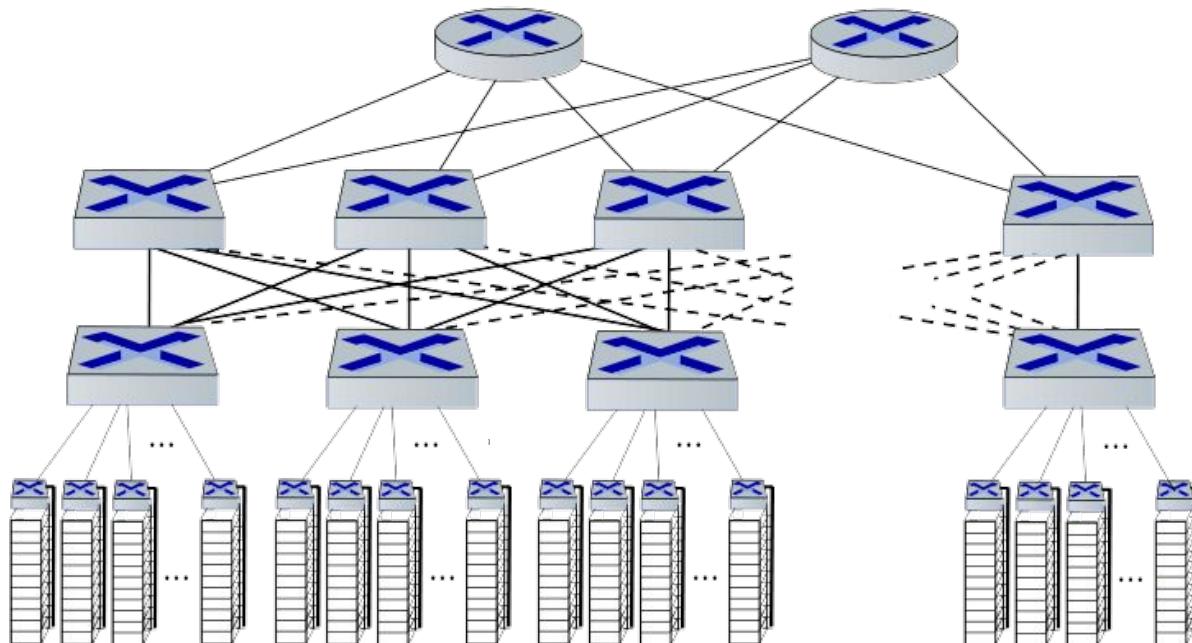
Datacenter networks

- 10's to 100's of thousands of hosts, often closely coupled, in close proximity:
 - e-business (e.g. Amazon)
 - content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
 - search engines, data mining (e.g., Google)
- Challenges:
 - multiple applications, each serving massive numbers of clients
 - Reliability
 - managing/balancing load, avoiding processing, networking, data bottlenecks



Inside a 40-ft Microsoft container, Chicago data center

Data center: network elements



Border routers

- connections outside datacenter

Tier-1 switches

- connecting to ~16 T-2s below

Tier-2 switches

- connecting to ~16 TORs below

Top of Rack (TOR) switch

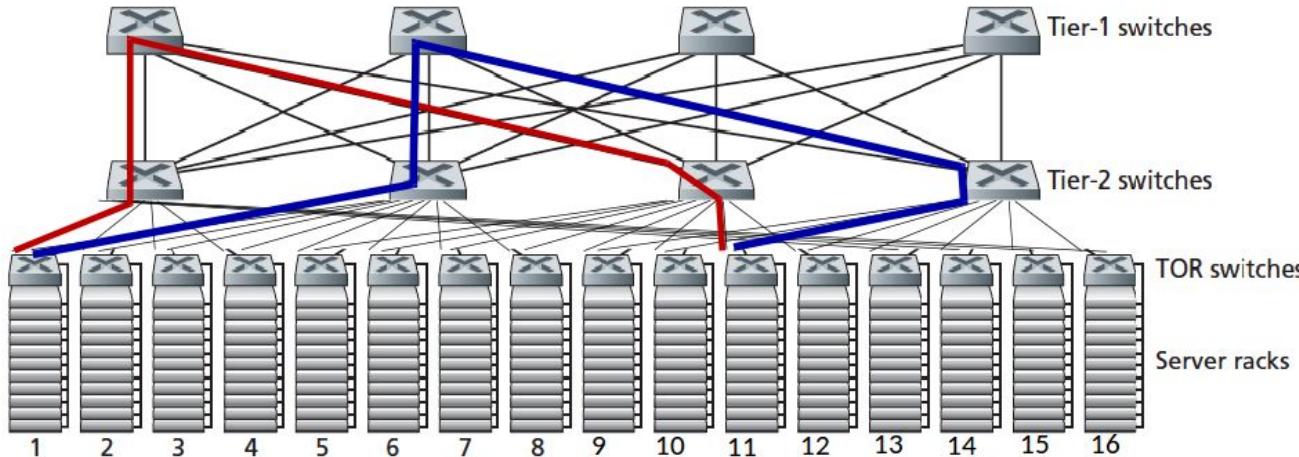
- one per rack
- 40-100Gbps Ethernet to blades

Server racks

- 20- 40 server blades: hosts

Datacenter networks: multipath

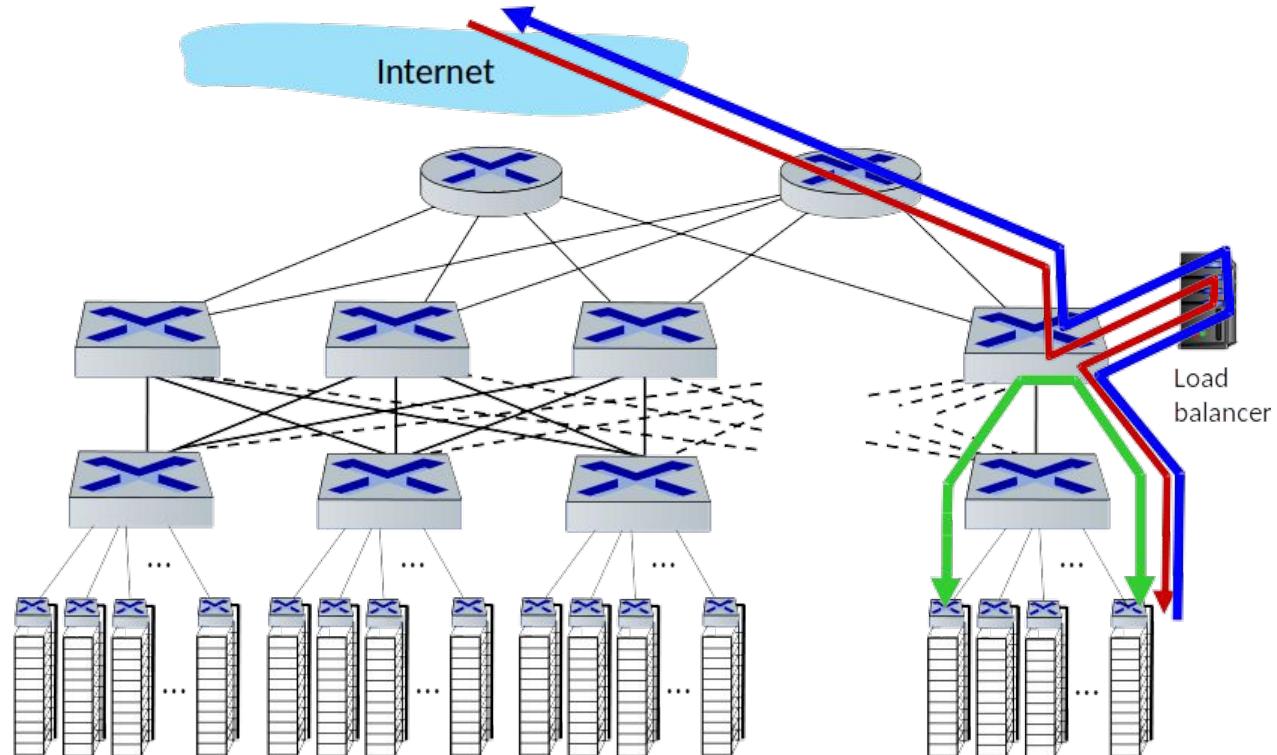
- rich interconnection among switches, racks:
 - increased throughput between racks (multiple routing paths possible)
 - increased reliability via redundancy



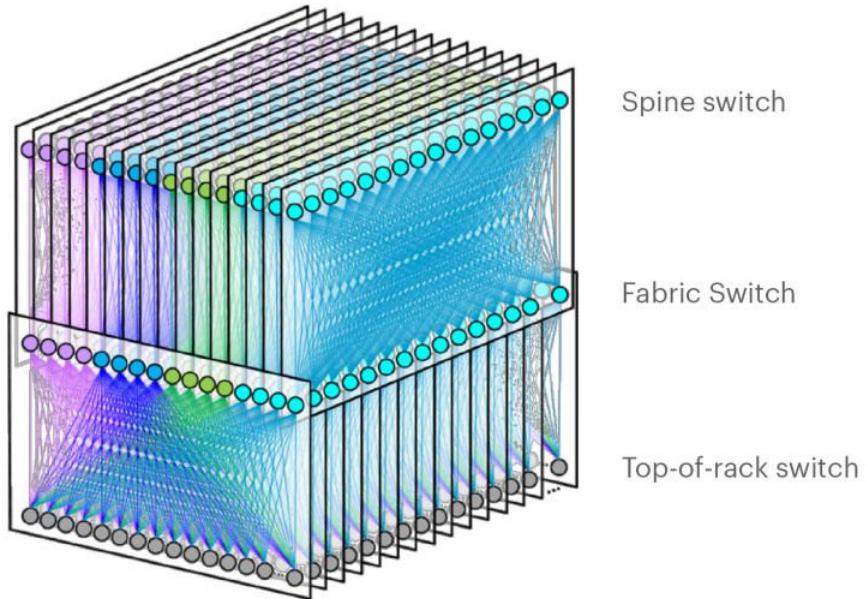
two **disjoint** paths highlighted between racks 1 and 11

Data center networks

- load balancer:
application-layer
routing
 - receives external client
requests
 - directs workload within
data center
 - returns results to
external client (hiding
data center internals
from client)



Facebook F16 data center network topology:



<https://engineering.fb.com/2019/03/14/data-center-engineering/f16-minipack/>

Datacenter networks: protocol innovations

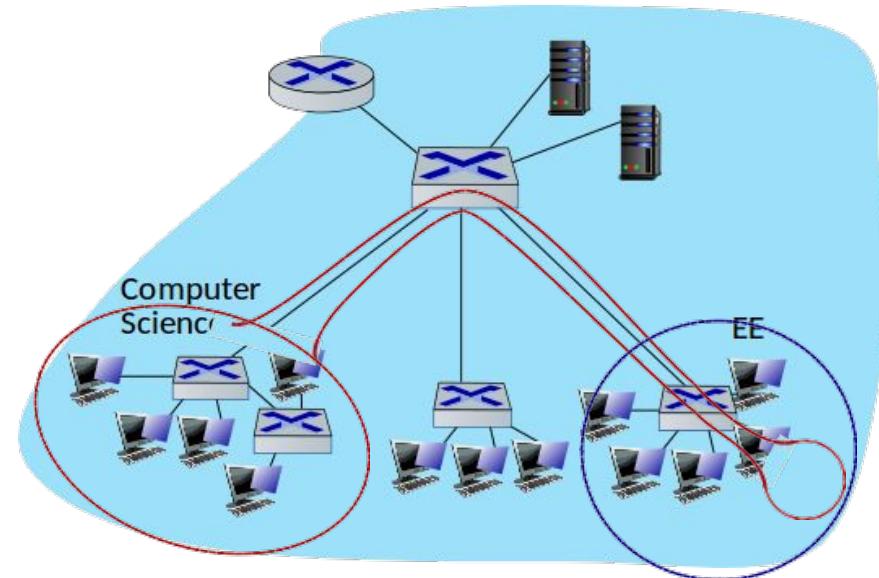
- link layer:
 - RoCE: remote DMA (RDMA) over Converged Ethernet
- transport layer:
 - ECN (explicit congestion notification) used in transport-layer congestion control (DCTCP, DCQCN)
 - experimentation with hop-by-hop (backpressure) congestion control
- routing, management:
 - SDN widely used within/among organizations' datacenters
 - place related services, data as close as possible (e.g., in same rack or nearby rack) to minimize tier-2, tier-1 communication

Outline

- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - Ethernet
 - Switches
 - addressing, ARP
 - **VLANS**
- link virtualization: MPLS
- data center networking
- a day in the life of a web request

Virtual LANs (VLANs): motivation

- Q: what happens as LAN sizes scale, users change point of attachment?
 - single broadcast domain:
 - **scaling**: all layer-2 broadcast traffic (ARP, DHCP, unknown MAC) must cross entire LAN
 - efficiency, security, privacy issues
 - administrative issues:
 - CS user moves office to EE - **physically** attached to EE switch, but wants to remain **logically** attached to CS switch



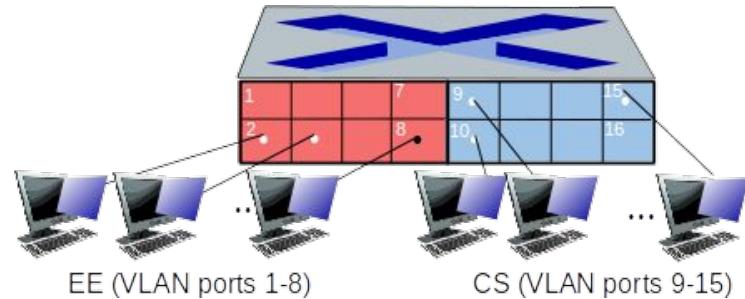
Virtual LAN

- Virtual Local Area Network
 - Defining single/multiple virtual local area network infrastructure over a single/multiple physical local area network
- Why?
 - Traffic isolation
 - Limit LAN traffic for better privacy/security
 - Efficient use of switches
 - No need to assign a single 96-port switch to each group (department) in each institution
 - Managing users
 - An employee can move between groups without the need to change physical cabling

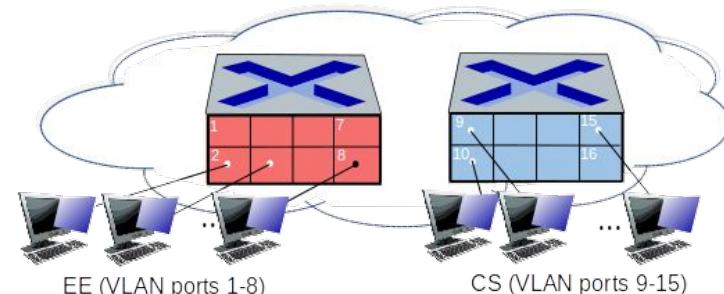
Port-based VLANs

- Virtual Local Area Network (VLAN)
 - switch(es) supporting VLAN capabilities can be configured to define multiple virtual LANs over single *physical* LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch

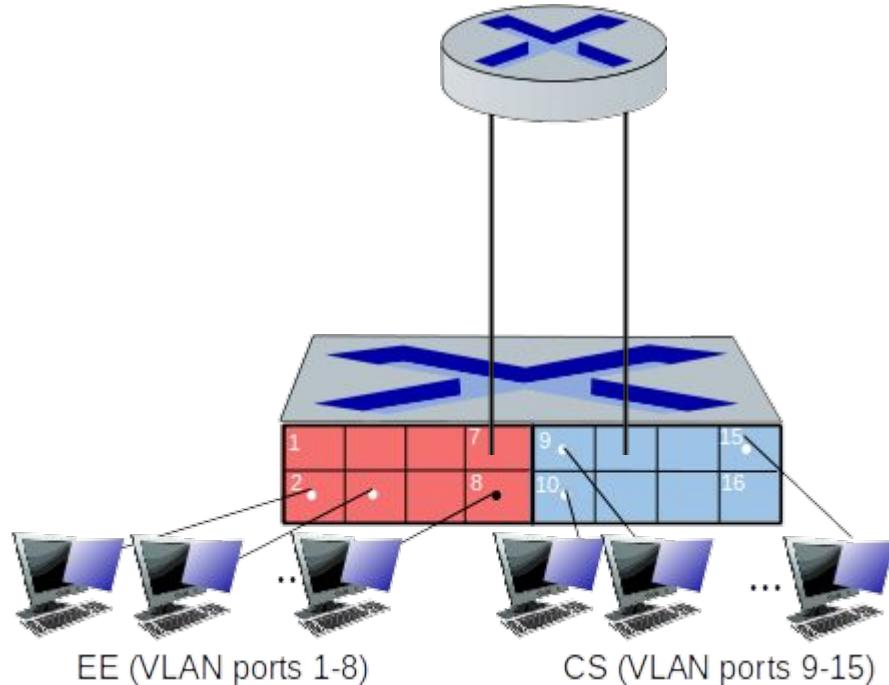


... operates as **multiple** virtual switches



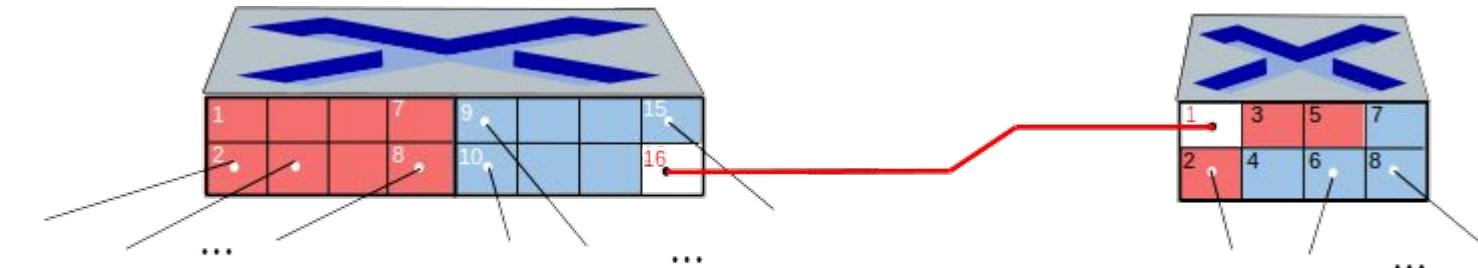
Port-based VLANs

- **traffic isolation:** frames to/from ports 1-8 can only reach ports 1-8
- **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



VLANS spanning multiple switches

- **trunk port:** carries frames between VLANS defined over multiple physical switches
 - frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
 - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

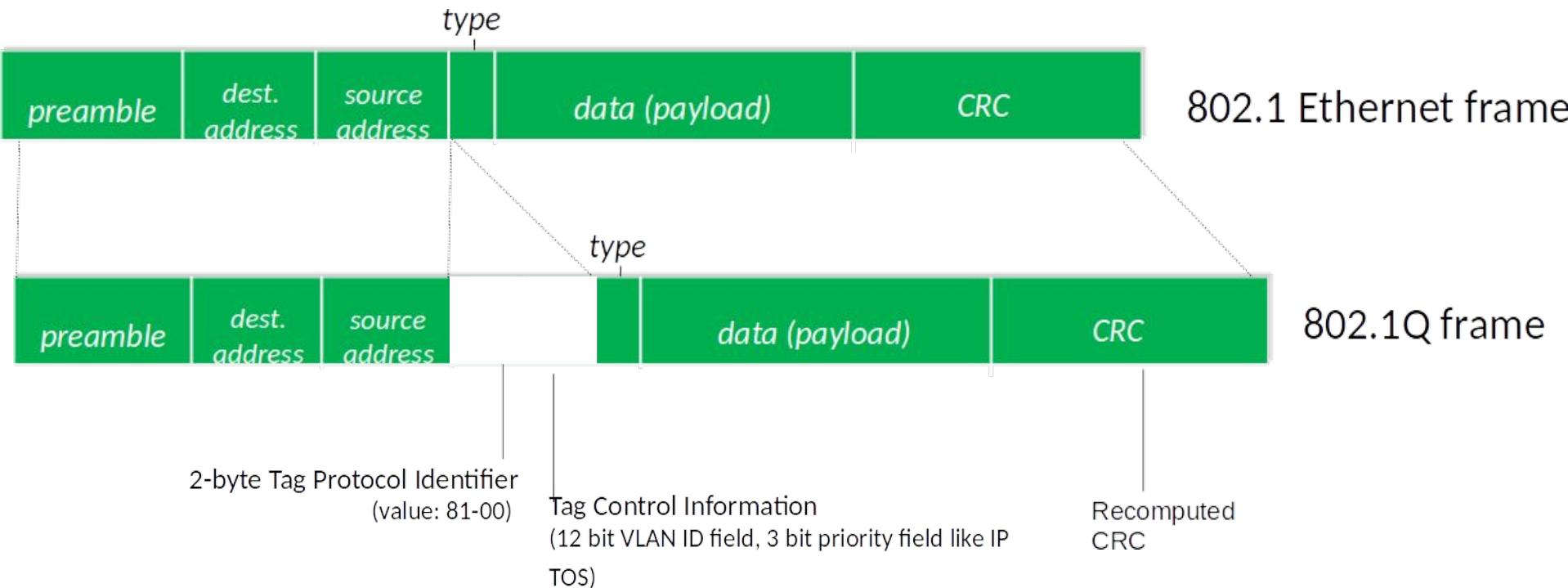


EE (VLAN ports 1-8)

CS (VLAN ports 9-15)

Ports 2,3,5 belong to EE VLAN
Ports 4,6,7,8 belong to CS VLAN

802.1Q VLAN frame format

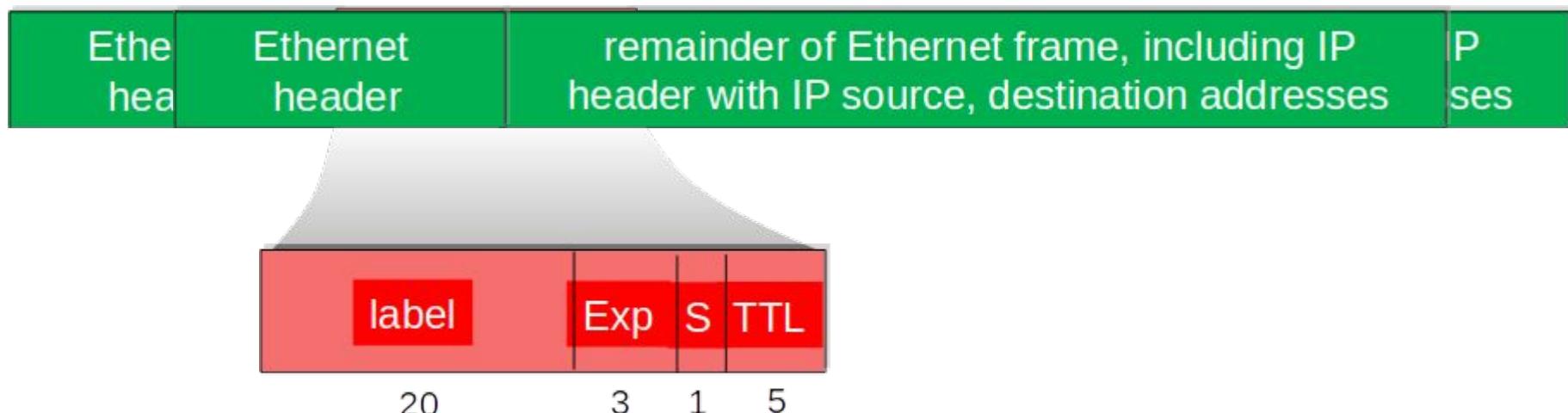


Outline

- introduction, services
- error detection, correction
- multiple access protocols
- LANs
 - Ethernet
 - Switches
 - addressing, ARP
 - VLANs
- **link virtualization: MPLS**
- data center networking
- a day in the life of a web request

Multiprotocol label switching (MPLS)

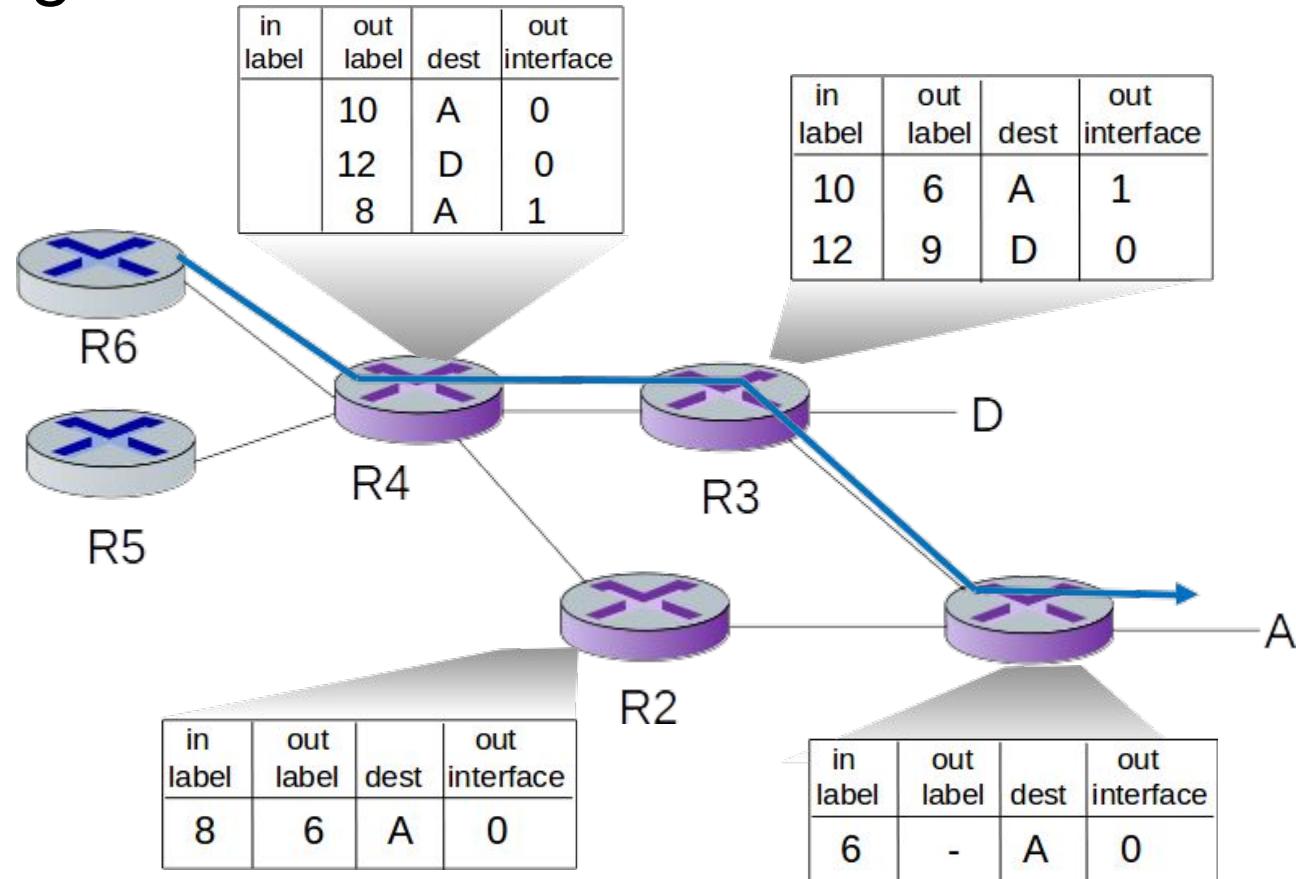
- **goal:** high-speed IP forwarding among network of MPLS-capable routers, using **fixed length label** (instead of shortest prefix matching)
 - faster lookup using fixed length identifier
 - borrowing ideas from Virtual Circuit (VC) approach
 - but IP datagram still keeps IP address!



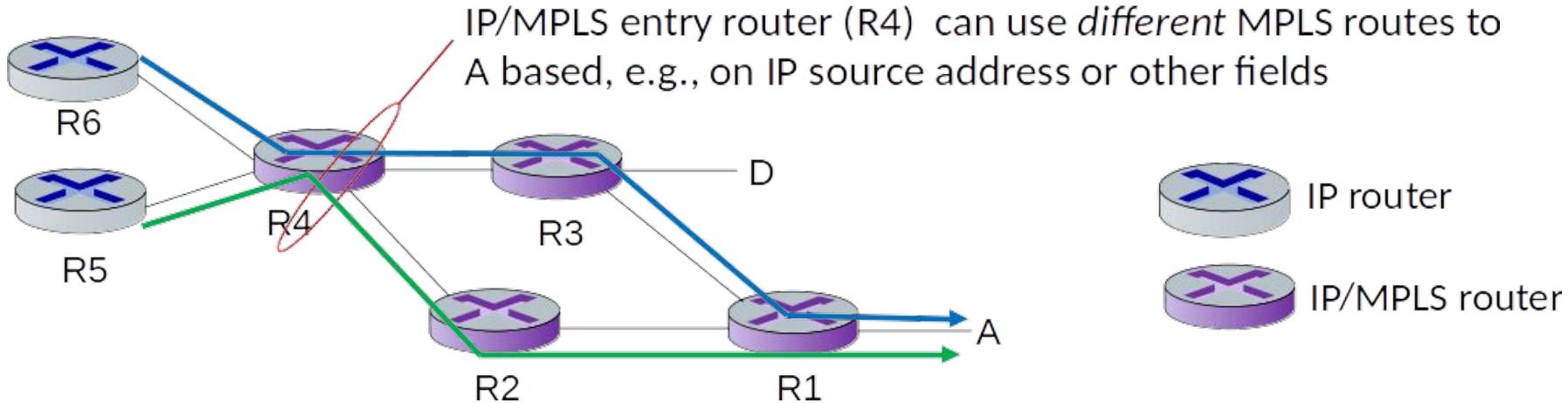
MPLS capable routers

- a.k.a. label-switched router
- forward packets to outgoing interface based only on label value (don't inspect IP address)
 - MPLS forwarding table distinct from IP forwarding tables
- **flexibility:** MPLS forwarding decisions can differ from those of IP
 - use destination and source addresses to route flows to same destination differently (traffic engineering)
 - re-route flows quickly if link fails: pre-computed backup paths

MPLS Forwarding tables



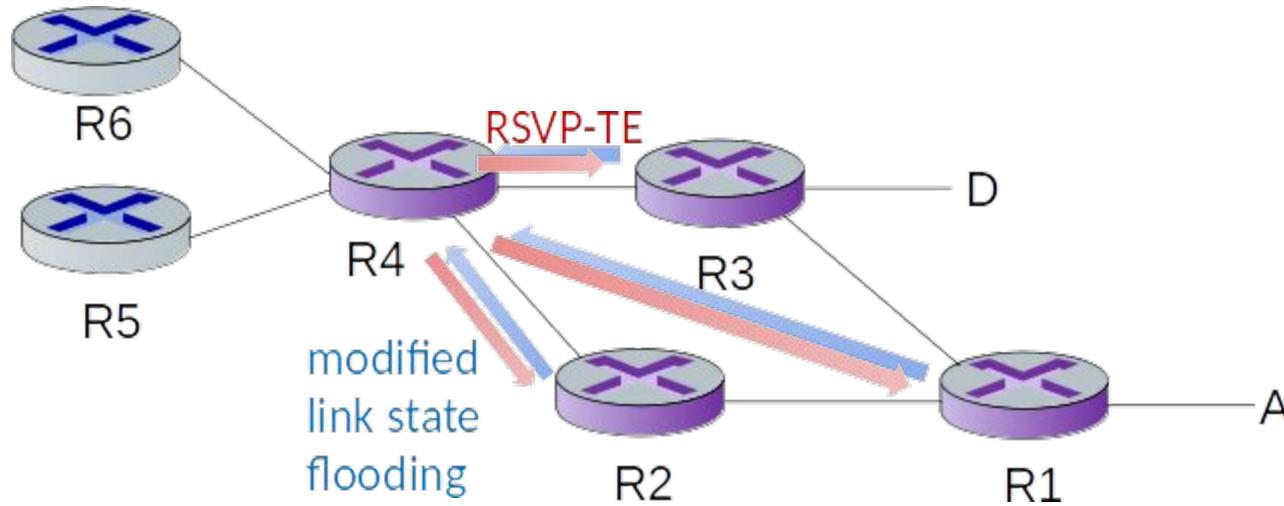
MPLS versus IP paths



- **IP routing:** path to destination determined by destination address alone
- **MPLS routing:** path to destination can be based on source and destination address
 - flavor of generalized forwarding (MPLS 10 years earlier)
 - **fast reroute:** precompute backup routes in case of link failure

MPLS signaling

- modify OSPF, IS-IS link-state flooding protocols to carry info used by MPLS routing:
 - e.g., link bandwidth, amount of “reserved” link bandwidth
- entry MPLS router uses RSVP-TE signaling protocol to set up MPLS forwarding at downstream routers



Summary: Link Layer

- principles behind data link layer services:
 - error detection, correction
 - sharing a broadcast channel: multiple access
 - link layer addressing
- instantiation and implementation of various link layer technologies
 - Ethernet
 - switched LANS, VLANs
 - virtualized networks as a link layer: MPLS
- synthesis: a day in the life of a web request

Security

Network Security

goals:

- understand principles of network security:
 - confidentiality
 - Authentication
 - message integrity
 - Availability
- Techniques to make networks secure
 - cryptography and its many uses beyond confidentiality
- security in practice:
 - firewalls and intrusion detection systems
 - security in application, transport, network, link layers

outline

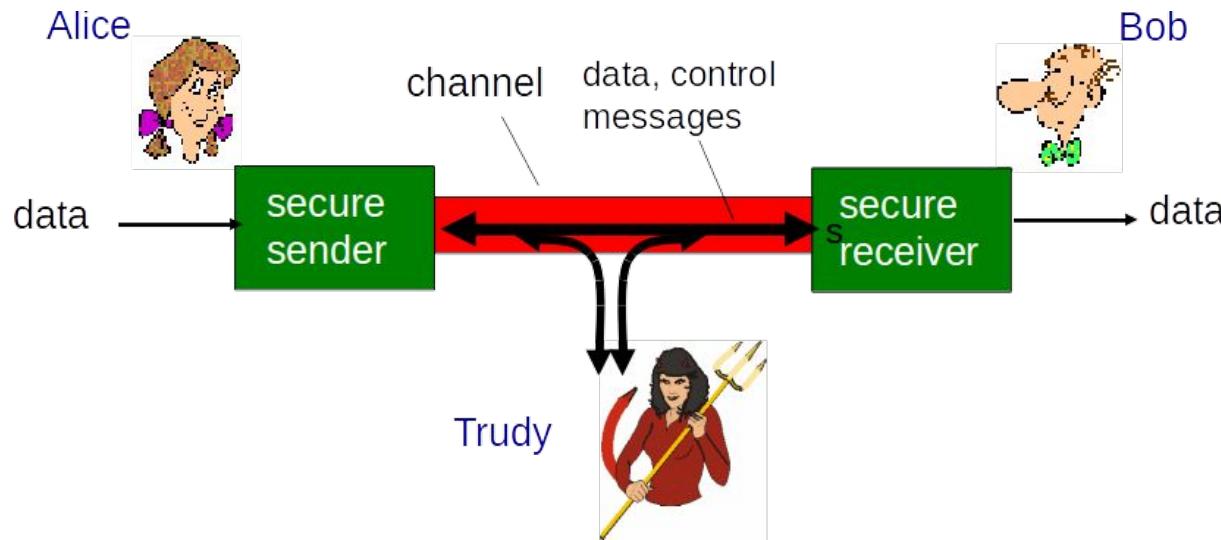
- **What is network security?**
- Principles of cryptography
- Message integrity
- authentication
- Securing e-mail
- Securing TCP connections: SSL
- Network layer security: IPsec
- Securing wireless LANs
- Operational security: firewalls and IDS

What is network security?

- **confidentiality**: only sender, intended receiver should “understand” message contents
 - sender encrypts message
 - receiver decrypts message
- **authentication**: sender, receiver want to confirm identity of each other
- **message integrity**: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection
- **access and availability**: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Who might Bob, Alice be?

- ... well, real-life Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates

There are bad guys (and girls) out there!

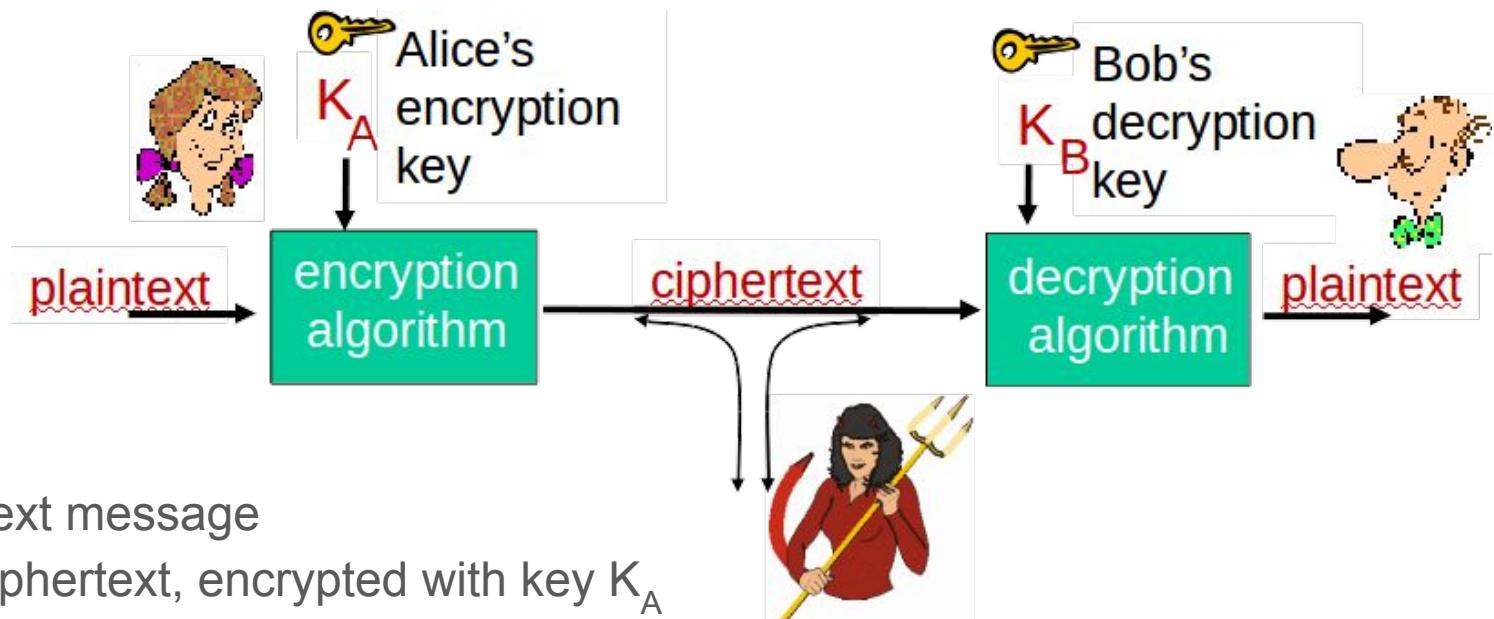
Q: What can a “bad guy” do? **A:** A lot!

- **eavesdrop**: intercept messages
- actively **insert** messages into connection
- **impersonation**: can fake (spoof) source address in packet (or any field in packet) -- IP spoofing
- **hijacking**: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service**: prevent service from being used by others (e.g., by overloading resources)

outline

- What is network security?
- **Principles of cryptography**
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: SSL
- Network layer security: IPsec
- Securing wireless LANs
- Operational security: firewalls and IDS

The language of cryptography

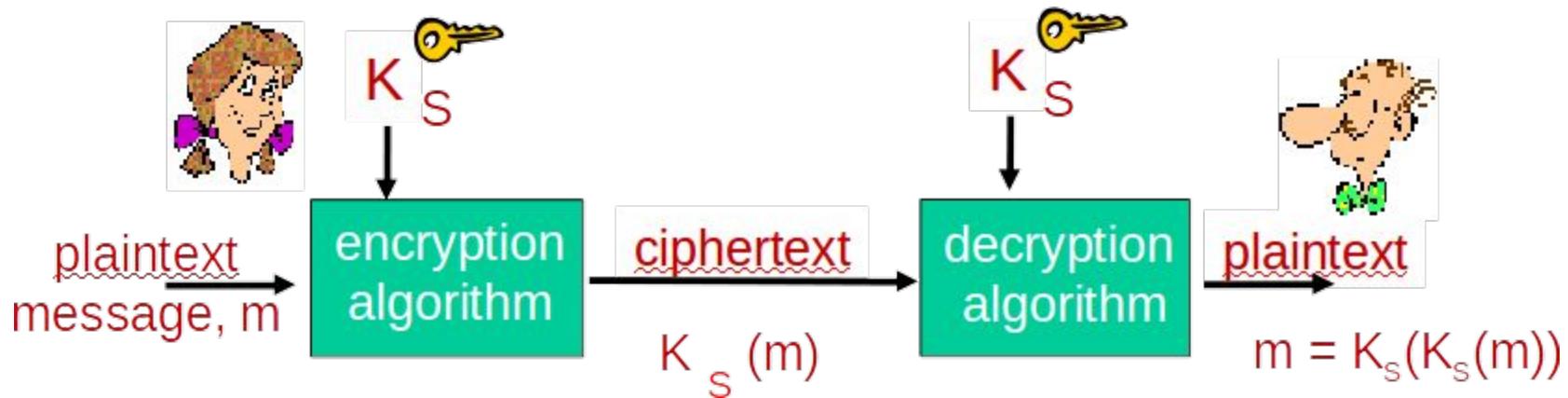


Encryption/Decryption

- Two main kinds of encryption:
 - Symmetric key encryption: e.g., AES
 - Alice and Bob share secret key
 - Public key encryption: e.g., RSA
 - Alice and Bob each have a key in two parts: a public part (widely known), and a private part (only owner knows)

Symmetric key cryptography

- **symmetric key crypto:** Bob and Alice share same (symmetric) key: K_S
 - Q: how do Bob and Alice agree on key value?



Simple encryption scheme

- **substitution cipher:** substituting one thing for another
 - **Caesar ciphers:** each letter is replaced with a letter that is K letters later
 - example: if $k=3$, then **a** is replaced with **d**, **z** is replaced with **c**.
 - e the value of k is the key.
 - **monoalphabetic cipher:** substitute one letter for another

plaintext: abcdefghijklmnopqrstuvwxyz

ciphertext: mnbvvcxzasd^gfghjk^lpoiuytrewq

e.g.: Plaintext: bob. i love you. alice

ciphertext: nkn. s gk^tc wky. mgsbc



Encryption key:

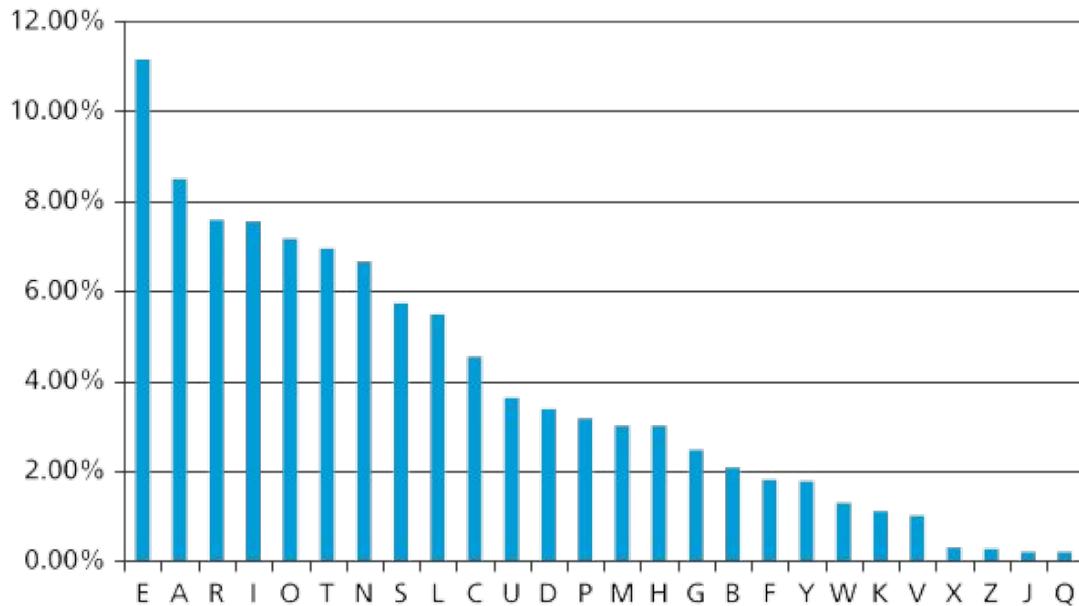
mapping from set of 26 letters to set of 26 letters

A more sophisticated encryption approach

- n substitution ciphers, M_1, M_2, \dots, M_n along with a cycling pattern:
 - for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
 - e.g., n=4: $M_1, M_3, M_4, M_3, M_2; M_1, M_3, M_4, M_3, M_2; \dots$
 - dog: d from M_1 , o from M_3 , g from M_4
 - **Encryption key:**
 - n substitution ciphers, and cyclic pattern

Breaking an encryption scheme

- These encryption schemes could be broken using statistical analysis.
- The caesar cipher
 - Vulnerable to attack
- More complex version
 - Each letter switching in one of 26 ways
 - Ciphertext open to frequency analysis



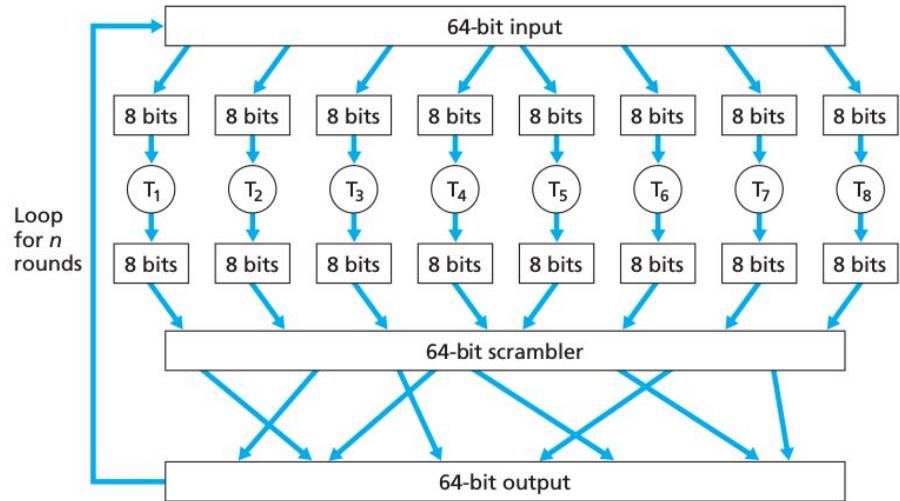
Letter frequency in the English alphabet using Oxford English Dictionary

Breaking an encryption scheme

- **Ciphertext-only attack:** Trudy has ciphertext she can analyze
 - two approaches:
 - brute force: search through all keys
 - statistical analysis
- **known-plaintext attack:** Trudy has plaintext corresponding to ciphertext
 - e.g., in monoalphabetic cipher, if Trudy knows that “bob” and “alice” appeared in the ciphertext message, she could determine pairings for letters a, l, i, c, e, b, o
- **chosen-plaintext attack:** Trudy can choose the plaintext message and obtains its corresponding ciphertext

Symmetric key Encryption

- Block Ciphers
 - Encrypt and decrypt messages using an iterative replacing of a message with another scrambled message using 64 or 128 bits at a time (the **block**)
 - **The resulting letter frequency of the cipher text is almost flat, therefore these techniques are** not vulnerable to classic crypt-analysis



Key: the eight permutation tables (T_1 to T_8)
The scrambling function is known publicly

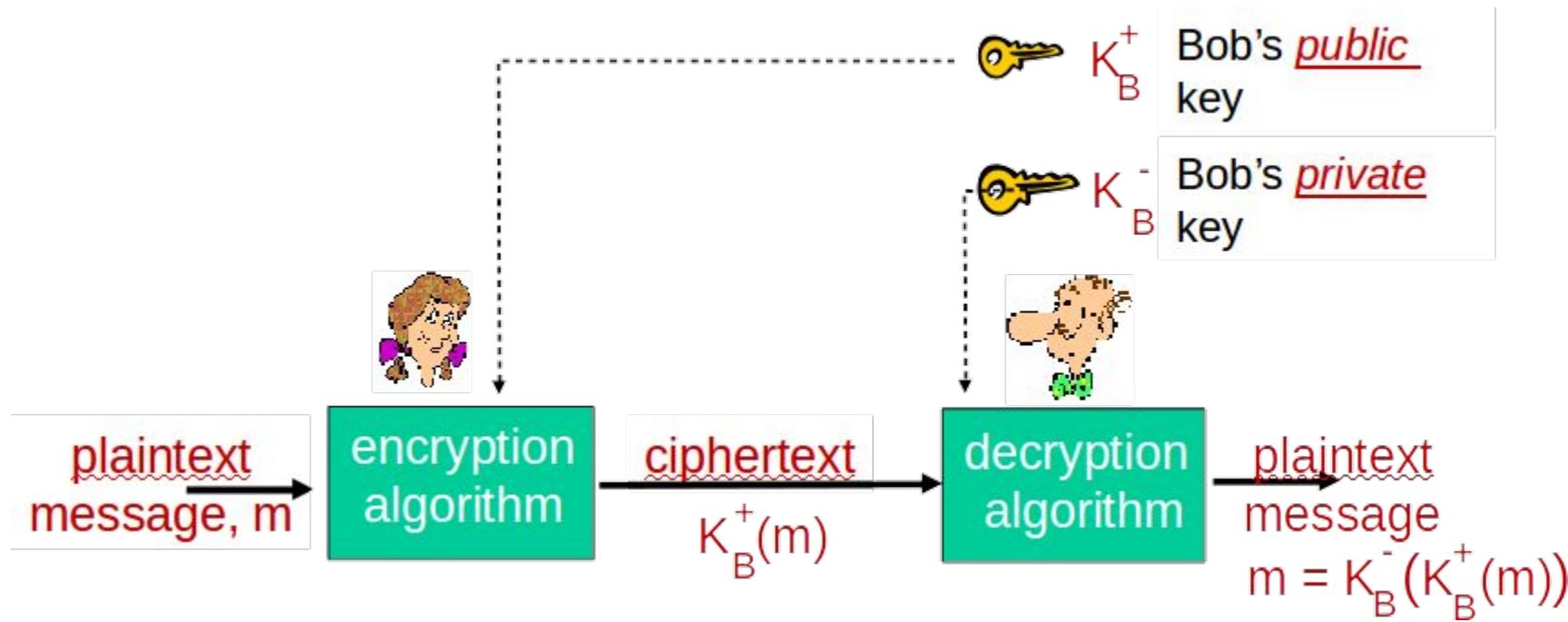
Symmetric key Encryption

- **DES: Data Encryption Standard**
 - US encryption standard [NIST 1993]
 - 56-bit symmetric key (Instead of the tables in previous figure), 64-bit plaintext input
 - **block cipher with cipher block chaining** (If two blocks are the same, their cipher are not the same)
 - how secure is DES?
 - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
 - no known good analytic attack
 - making DES more secure: 3DES: encrypt 3 times with 3 different keys
- **AES: Advanced Encryption Standard**
 - symmetric-key NIST standard, replaced DES (Nov 2001)
 - processes data in 128 bit blocks
 - 128, 192, or 256 bit keys
 - brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

Public Key Cryptography

- symmetric key crypto
 - requires sender, receiver know shared secret key
 - **Q: how to agree on key in first place (particularly if never “met”)?**
- public key crypto
 - radically different approach [Diffie-Hellman76, RSA78]
 - sender, receiver do **not** share secret key
 - **public** encryption key known to all
 - **private** decryption key known only to receiver

Public key cryptography



Public key encryption algorithms

requirements:

- ① need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that
$$K_B^-(K_B^+(m)) = m$$
- ② given public key K_B^+ , it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

Prerequisite: modular arithmetic

- $x \bmod n$ = remainder of x when divide by n
- Facts:
 - $[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$
 - $[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$
 - $[(a \bmod n) * (b \bmod n)] \bmod n = (a*b) \bmod n$
- thus
- $(a \bmod n)^d \bmod n = a^d \bmod n$
- example:
 - $x=14, n=10, d=2:$
 - $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
 - $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

RSA: getting ready

- **message:** just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number
- Example:
 - $m = 10010001$. This message is uniquely represented by the decimal number 145.
 - to encrypt m , we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA: Creating public/private key pair

1. choose two **large prime numbers** p, q . (e.g., 1024 bits each)
2. compute $n = pq$, $z = (p-1)(q-1)$
3. choose e (with $e < n$) that has no common factors with z (e, z are “relatively prime”).
4. choose d such that $ed - 1$ is exactly divisible by z .

(in other words: $ed \bmod z = 1$).

1. public key is (n, e) . private key is (n, d) .

$$\underbrace{(n, e)}_{K_B^+} \quad \underbrace{(n, d)}_{K_B^-}$$

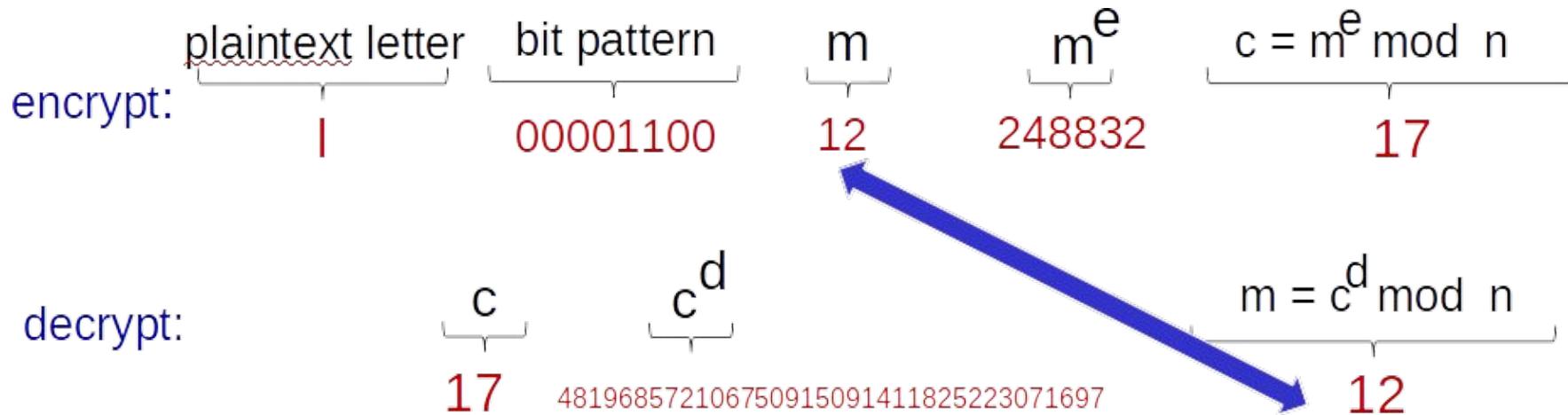
RSA: encryption, decryption

0. given (n, e) and (n, d) as computed above
1. to encrypt message $m (< n)$, compute $c = m^e \bmod n$
2. to decrypt received bit pattern, c , compute $m = c^d \bmod n$

Magic happens! $m = (m^e \bmod n)^d \bmod n$

RSA example:

- Bob chooses $p=5$, $q=7$. Then $n=35$, $z=24$.
 - $e=5$ (so e , z relatively prime).
 - $d=29$ (so $ed-1$ exactly divisible by z).
- encrypting 8-bit messages.



Why does RSA work?

must show that $c^d \bmod n = m$ where $c = m^e \bmod n$

fact: for any x and y : $x^y \bmod n = x^{(y \bmod z)} \bmod n$

where $n = pq$ and $z = (p-1)(q-1)$

thus,

$$\begin{aligned} c^d \bmod n &= (m^e \bmod n)^d \bmod n \\ &= m^{ed} \bmod n \\ &= m^{(ed \bmod z)} \bmod n \\ &= m^1 \bmod n \\ &= m \end{aligned}$$

$$(a \bmod n)^d \bmod n = a^d \bmod n$$

Replace $a = m^e$

RSA: another important property

The following property will be **very** useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

use public key
first, followed
by private key use private key
first, followed
by public key

*result is the
same!*

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$m = c^d \bmod n \text{ and } c = m^e \bmod n$$

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n = m^{de} \bmod n = (m^d \bmod n)^e \bmod n$$

RSA: Creating public/private key pair

1. choose two **large prime numbers** p, q . (e.g., 1024 bits each)
2. compute $n = pq$, $z = (p-1)(q-1)$
3. choose e (with $e < n$) that has no common factors with z (e, z are “relatively prime”).
4. choose d such that $ed - 1$ is exactly divisible by z .

(in other words: $ed \bmod z = 1$).

1. public key is (n, e) . private key is (n, d) .

$$\underbrace{(n, e)}_{K_B^+} \quad \underbrace{(n, d)}_{K_B^-}$$

Why is RSA secure?

- suppose you know Bob's public key (n, e). How hard is it to determine d ?
- essentially need to find factors of n without knowing the two factors p and q
 - fact: factoring a big number is hard, there are no known algorithms for quickly factoring a number (in this case the public value n) into the primes p and q . If one knew p and q , then given the public value e , one could easily compute the secret key, d .
 - On the other hand, it is not known whether or not there **exist** fast algorithm for factoring a number, and in this sense , the security of RSA is not guaranteed.

1. choose two **large prime numbers** p, q . (e.g., 1024 bits each)
2. compute $n = pq$, $z = (p-1)(q-1)$
3. choose e (with $e < n$) that has no common factors with z (e, z are “relatively prime”).
4. choose d such that $ed-1$ is exactly divisible by z .
5. public key is (n, e) . private key is (n, d) .

RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

session key, K_s

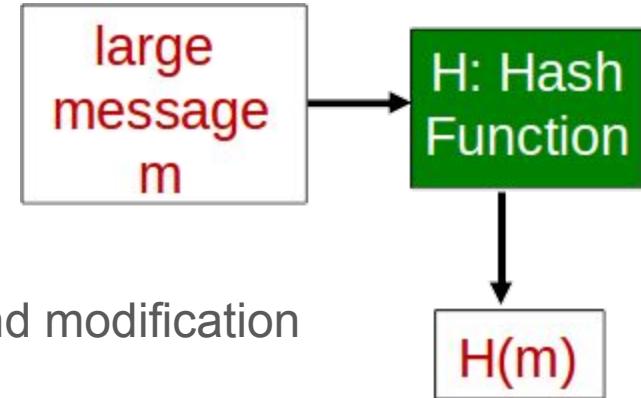
- Bob and Alice use RSA to exchange a symmetric key K_s
- once both have K_s , they use symmetric key cryptography

outline

- What is network security?
- Principles of cryptography
- **Message integrity**
- Authentication
- Securing e-mail
- Securing TCP connections: SSL
- Network layer security: IPsec
- Securing wireless LANs
- Operational security: firewalls and IDS

Message digests

- Providing confidentiality is not enough
- **Integrity:** protecting messages from tampering and modification



Message digests:

- apply **hash function H** to m , get fixed size **message digest, $H(m)$** .
- **Cryptographic Hash function properties**
 - Many-to-1
 - produces fixed-size message digest
 - it is computationally infeasible to find $x \neq y$ such that $H(x) = H(y)$
 - many such collisions exist, but no-one has been able to find one, even after analyzing the algorithm for years
 - MD5 or SHA functions (SHA-256 or SHA-512 today (SHA-2) or SHA-3) are examples of cryptographic hash functions.

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

<u>message</u>	<u>ASCII format</u>
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42

<u>message</u>	<u>ASCII format</u>
I O U <u>9</u>	49 4F 55 <u>39</u>
0 0 . <u>1</u>	30 30 2E <u>31</u>
9 B O B	39 42 D2 42

B2 C1 D2 AC different messages B2 C1 D2 AC
but identical checksums!

Hash function algorithms

- MD5 hash function widely used (RFC 1321)
 - converts any arbitrary sequence of bytes, of any length, into a 128-bit digest.
 - $128\text{-bit} = 2^{128}$ distinct condensations
 - appears difficult to construct msg m whose MD5 hash is equal to x
- SHA-1 (Secure Hash Algorithm) is also used
 - US standard [NIST, FIPS PUB 180-1]
 - 160-bit message digest

Message Authentication Code (MAC)

Can hash functions provide **integrity**?

1. Alice creates message m and $H(m)$
2. Alice sends $(m, H(m))$ to Bob
3. Bob receives (m, h) and calculates $H(m)$. if $H(m) == h$, Bob concludes everything is fine

Wrong! Trudy creates m' , sends $(m', H(m'))$. When Bob receives the message, everything looks good!

Message Authentication Code (MAC)

Can hash functions provide **integrity**?

1. Alice creates message m , creates $\mathbf{m + s}$, calculates $H(m+s)$
2. Alice sends $(m, H(m+s))$ to Bob
3. Bob receives (m, h) and **knowing s** , calculates $H(m+s)$. if $H(m+s) == h$, Bob concludes everything is fine

Shared secret

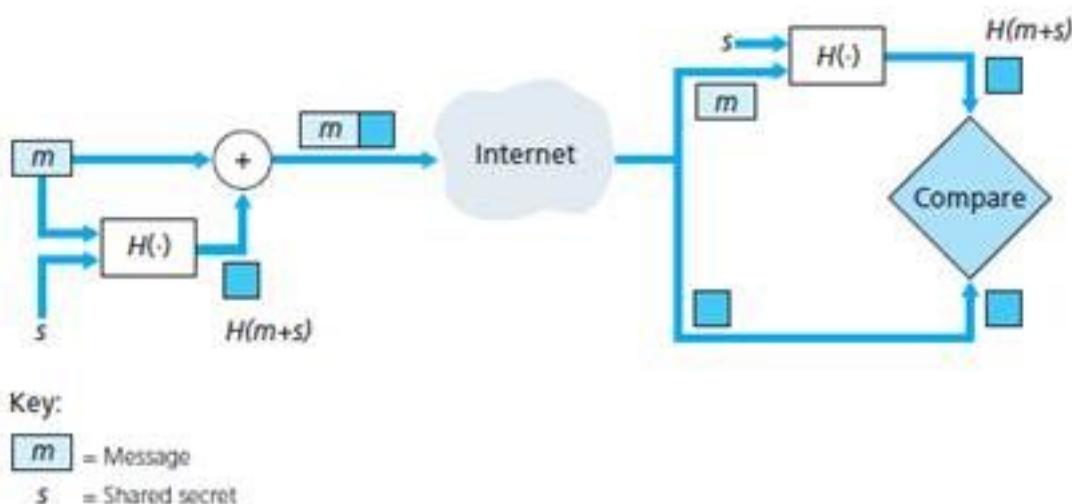
Message authentication code

Threat: Trudy creates m' , sends $(m', H(m'))$.

Bob calculates $H(m'+s)$. It is not likely that a different message generates the same digest because of the properties of cryptographic hash functions. Therefore $H(m'+s)$ is not equal to h and Bob finds out the message has been modified.

Message Authentication Code

- MAC does not require an encryption algorithm
- HMAC: The most popular standard for MAC:
- Can be used either with MD5 or SHA-1



Digital signatures

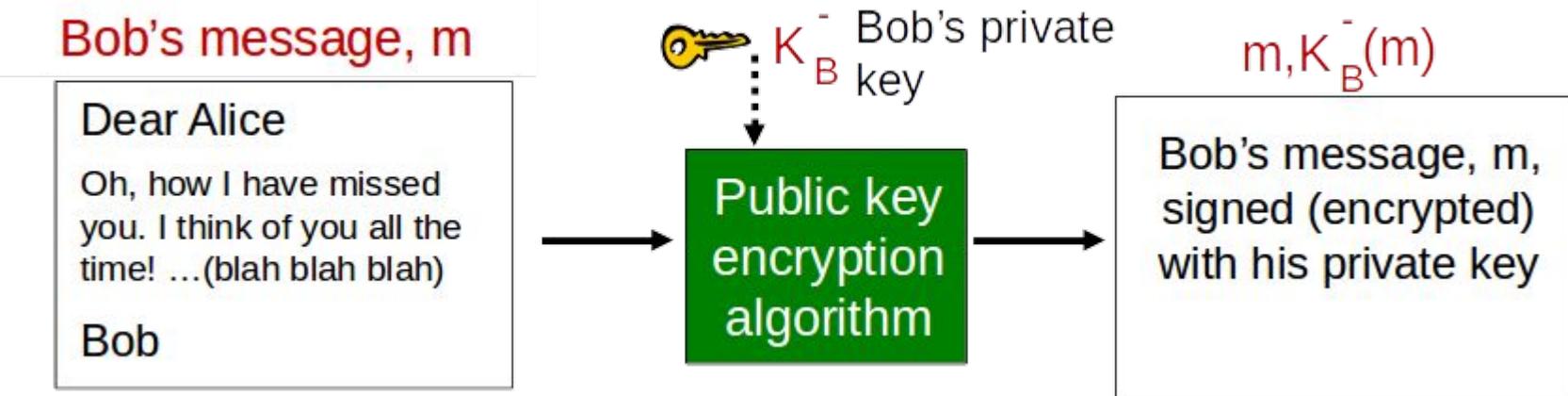
cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs the document, establishing he is document owner/creator.
- **verifiable, nonforgeable:**
 - Verifiable: A document signed by an individual was indeed signed by that individual
 - Non-forgeable: only that individual could have signed the document.
 - recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital signatures

simple digital signature for message m :

- Bob signs m by encrypting with his private key K_B^- , creating “signed” message, $K_B^-(m)$

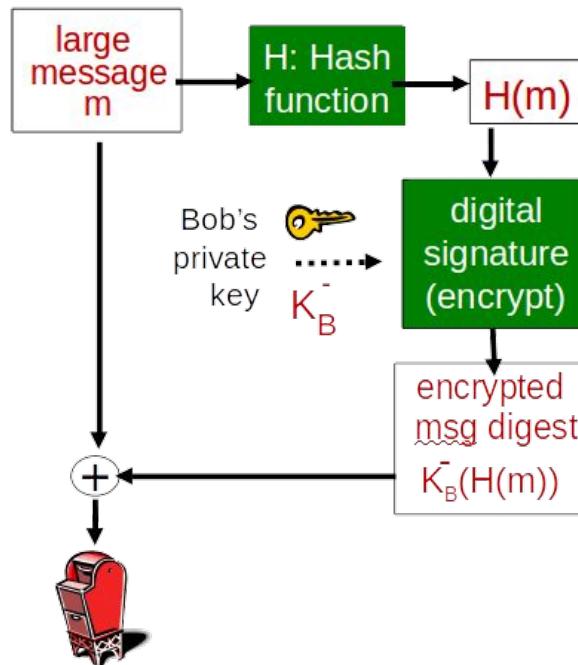


Digital signatures

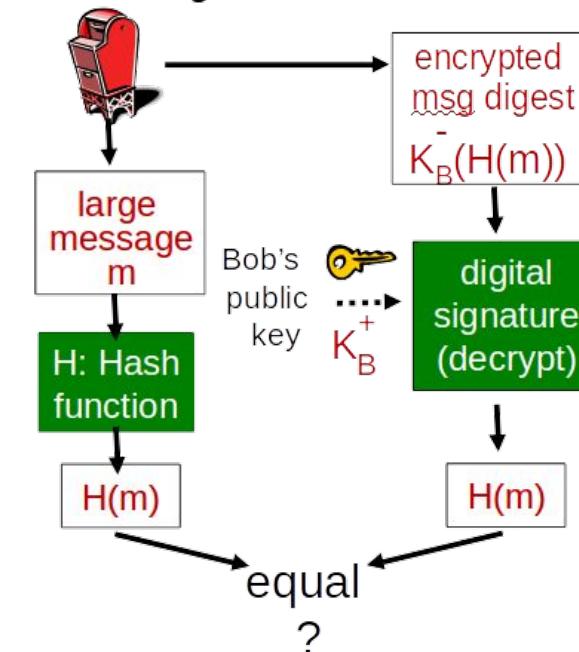
- suppose Alice receives msg m , with signature: $m, K_B^-(m)$
 - Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$
 - If $K_B^+ (K_B^-(m)) = m$, whoever signed m must have used Bob's private key.
- Why this technique work? Because Alice can verify that:
 - Bob signed m (verifiable)
 - no one else signed m (nonforgeable)
 - Bob signed m and not m'
 - If the original document, m is modified to m' , the signature will not be valid for m' since
 - $K_B^+ (K_B^-(m))$ is not equal to m'

Digital signature = signed message digest

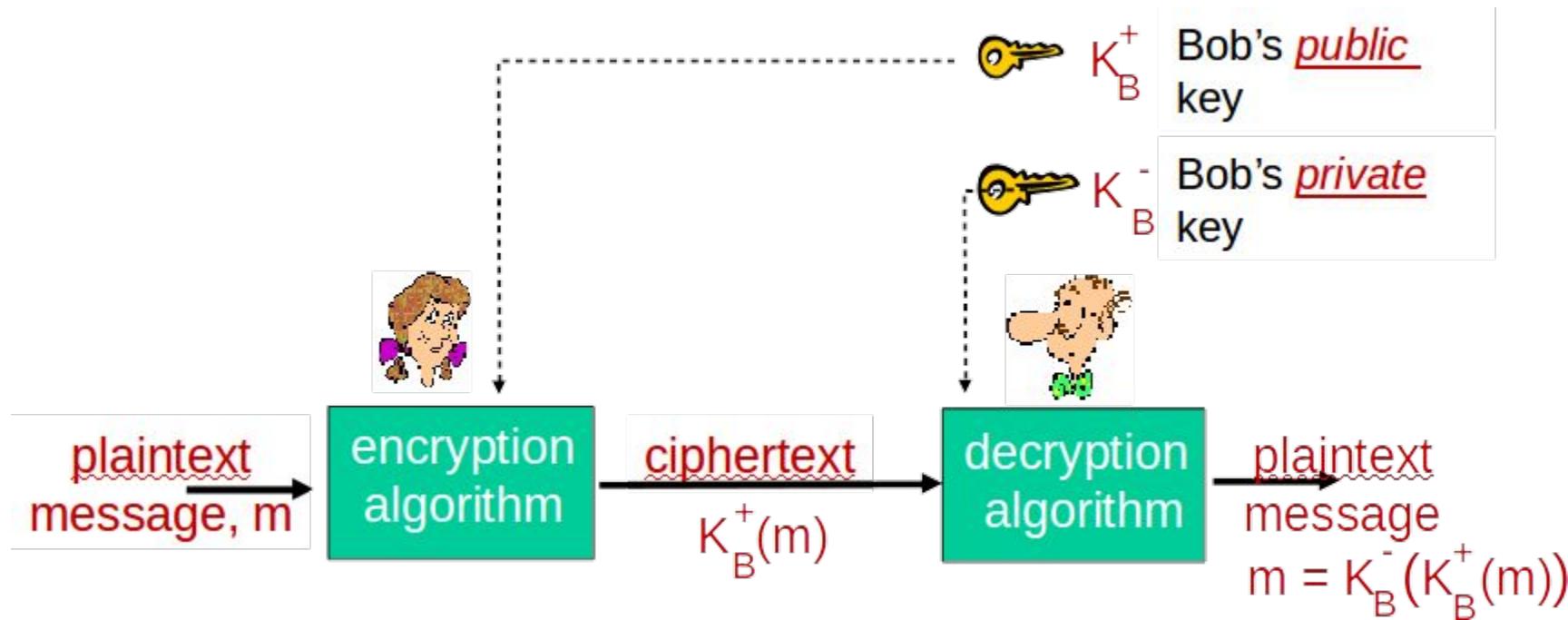
Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:



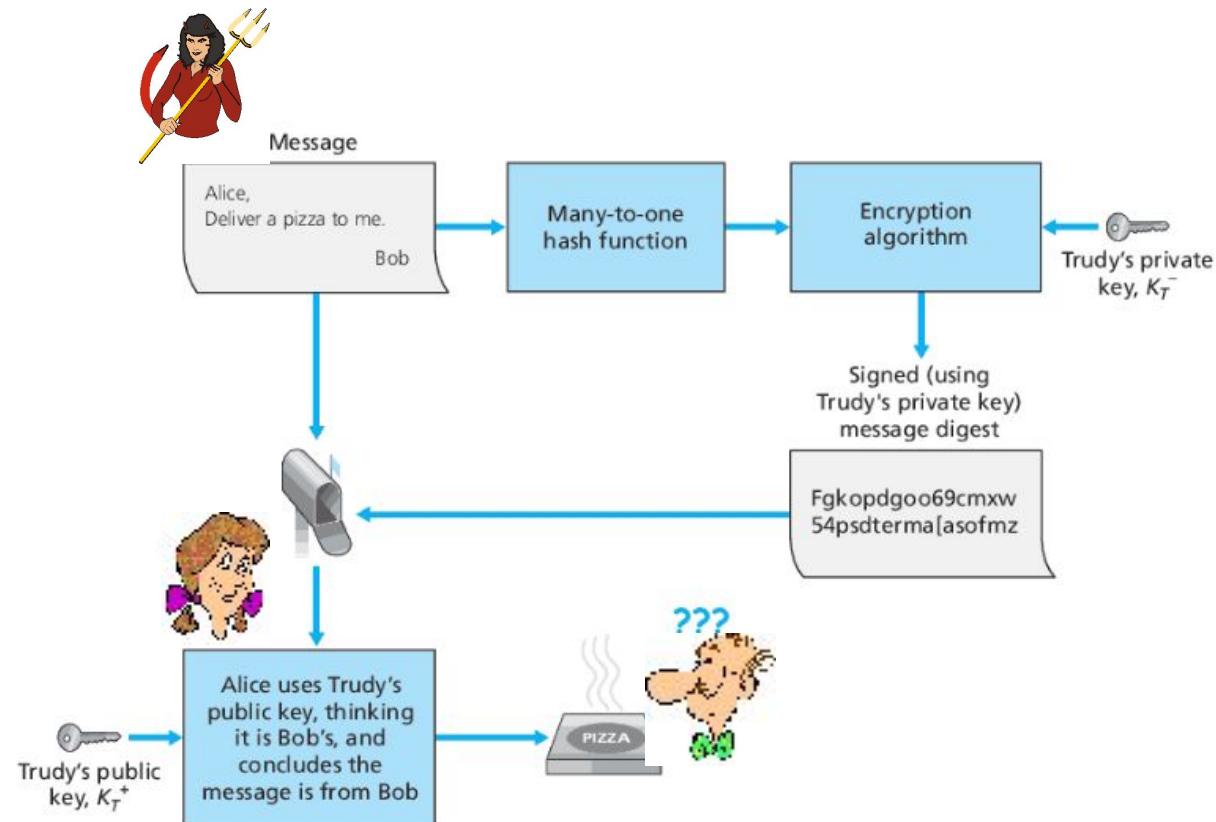
Public-key certification: Public-key cryptography



Public-key certification

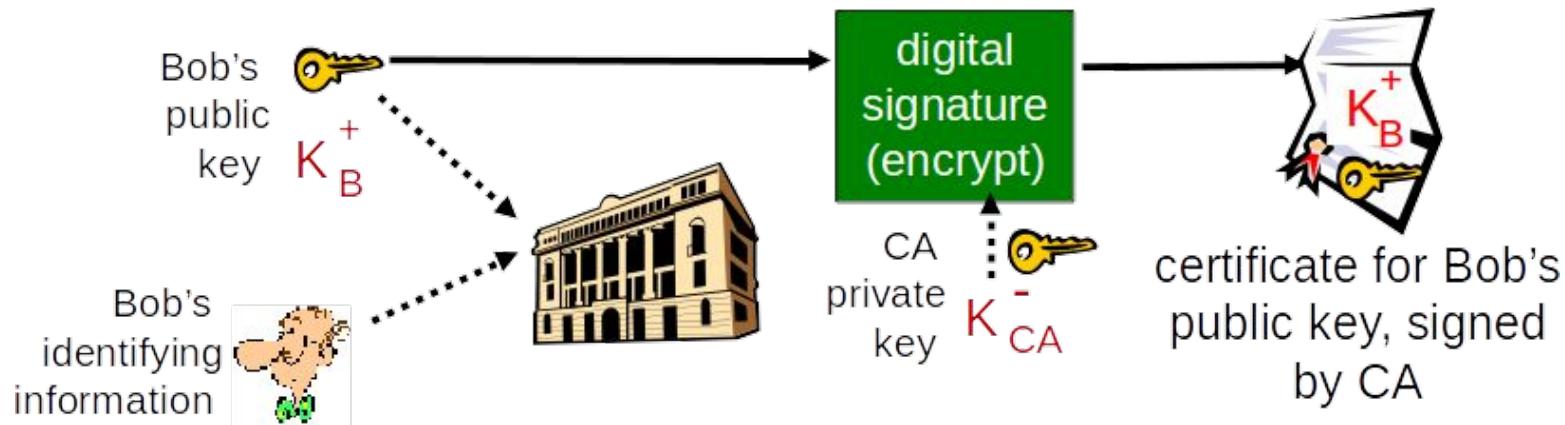
- motivation: Trudy plays pizza prank on Bob
 - Trudy creates e-mail order:
 - Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
 - Trudy signs order with her private key
 - Trudy sends order to Pizza Store
 - **Trudy sends to Pizza Store her public key, but says it's Bob's public key**
 - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
 - Bob doesn't even like pepperoni

Public-key certification



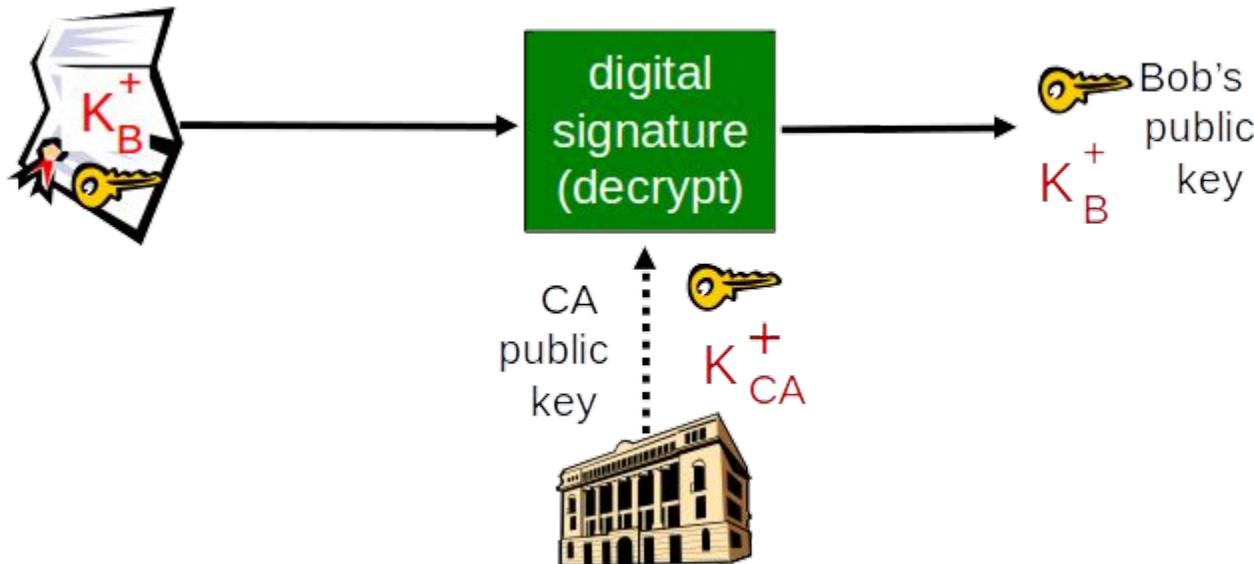
Certification authorities

- **certification authority (CA)**: binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
 - E provides “proof of identity” to CA.
 - CA creates certificate binding E to its public key.
 - certificate containing E’s public key digitally signed by CA – CA says “this is E’s public key”



Certification authorities

- when Alice wants Bob's public key:
 - gets Bob's (or elsewhere) certificate (it includes the public key of Bob)
 - apply CA's public key to Bob's certificate, get Bob's public key



outline

- What is network security?
- Principles of cryptography
- Message integrity
- **Authentication**
- Securing e-mail
- Securing TCP connections: SSL
- Network layer security: IPsec
- Securing wireless LANs
- Operational security: firewalls and IDS

Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”



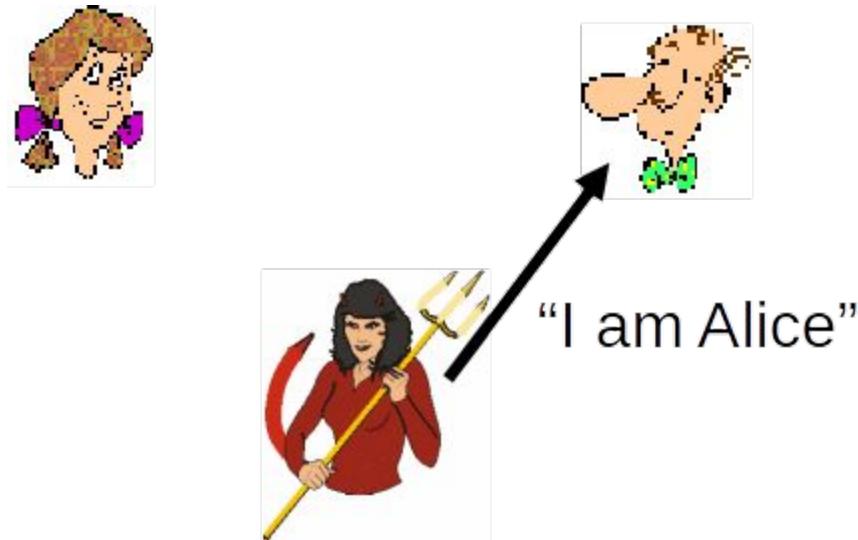
Failure scenario??



Authentication

Goal: Bob wants Alice to “prove” her identity to him

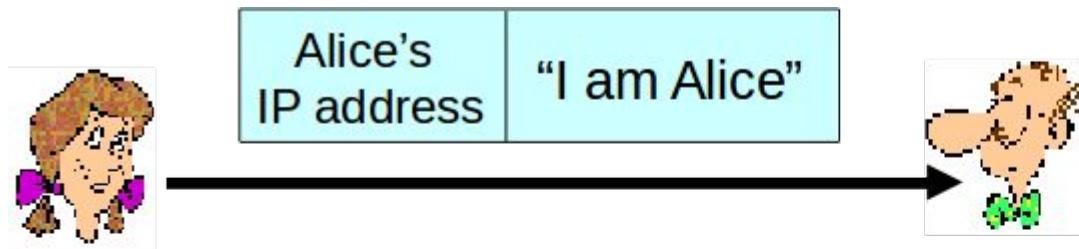
Protocol ap1.0: Alice says “I am Alice”



in a network,
Bob can not “see” Alice,
so Trudy simply declares
herself to be Alice

Authentication: another try

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address

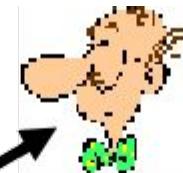


Authentication: another try

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



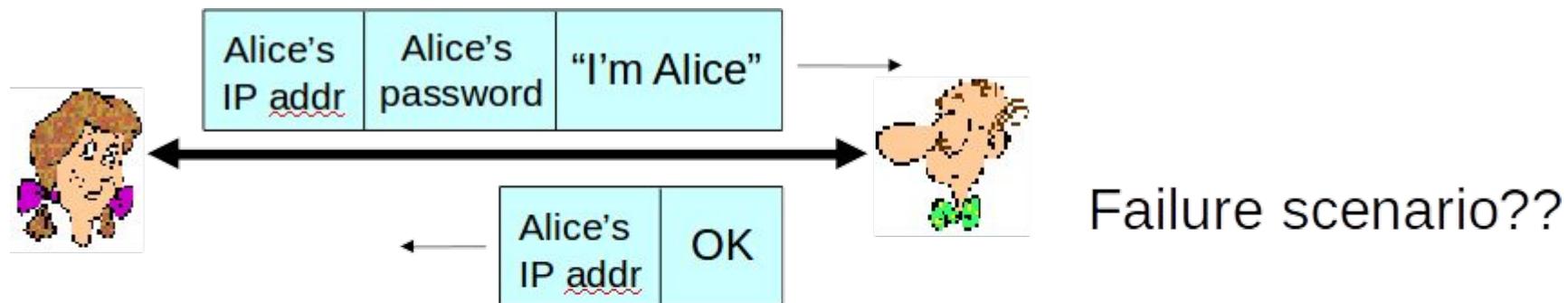
Alice's IP address	“I am Alice”
--------------------	--------------



Trudy can create
a packet “spoofing”
Alice’s address

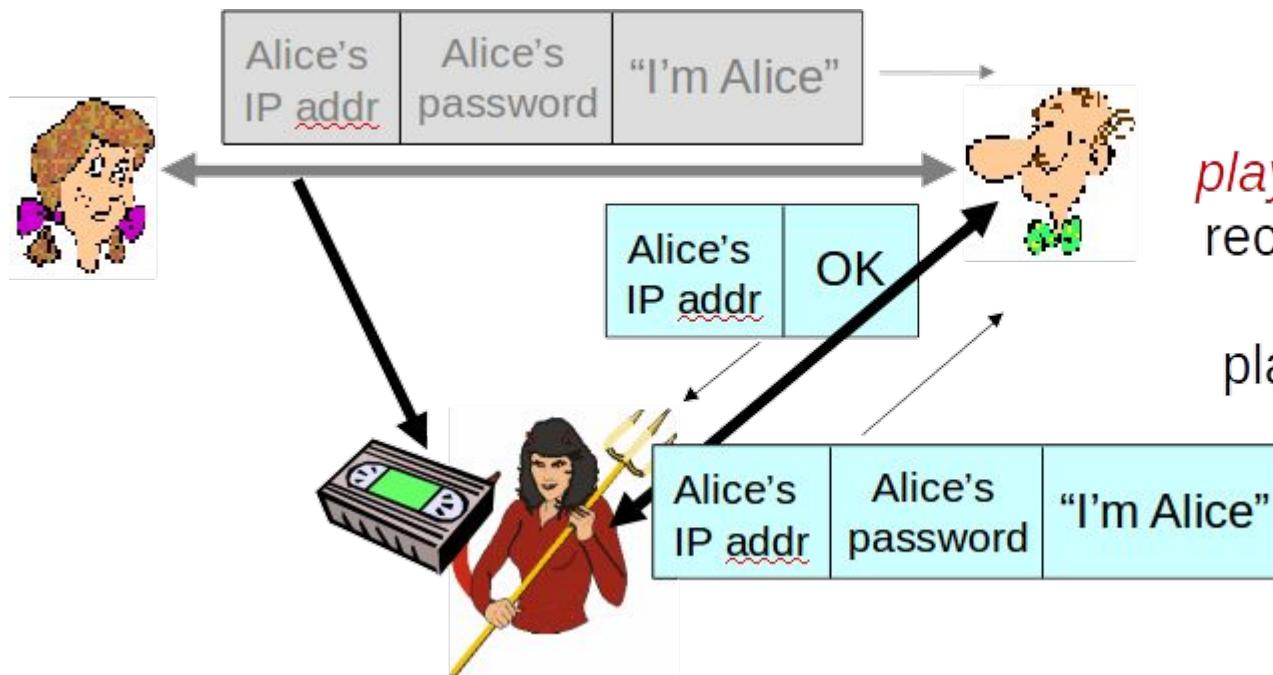
Authentication: another try

Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.



Authentication: another try

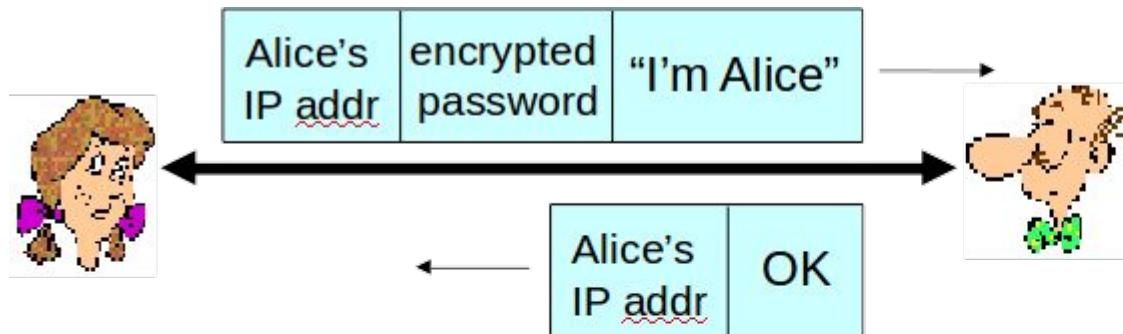
Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.



playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: another try

Protocol ap3.1: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.

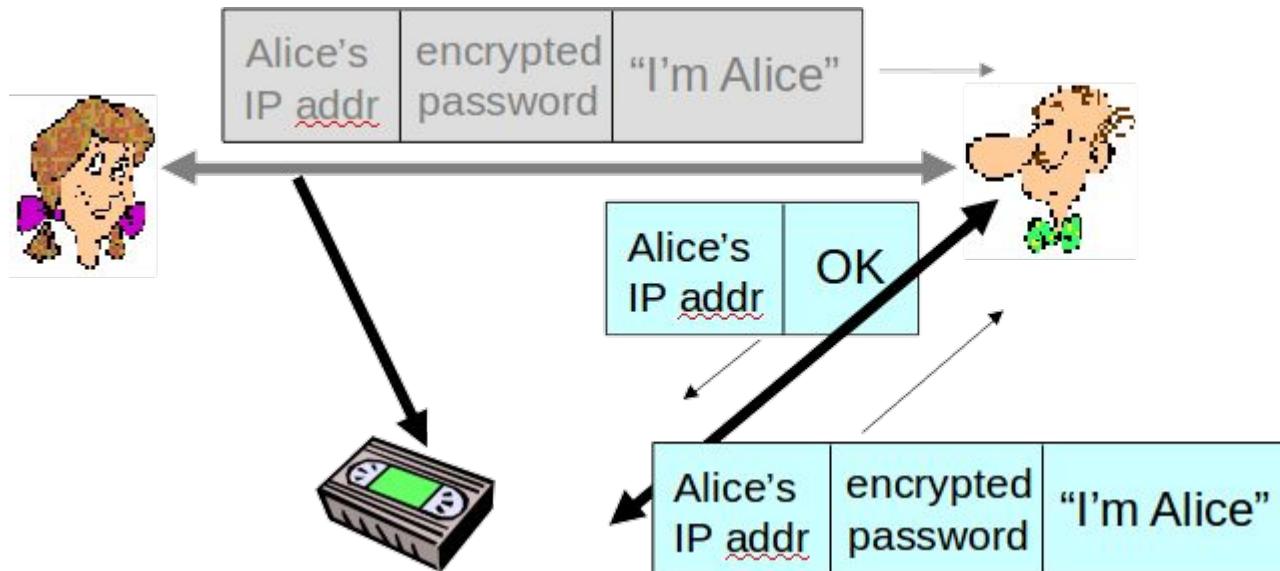


Failure scenario??



Authentication: another try

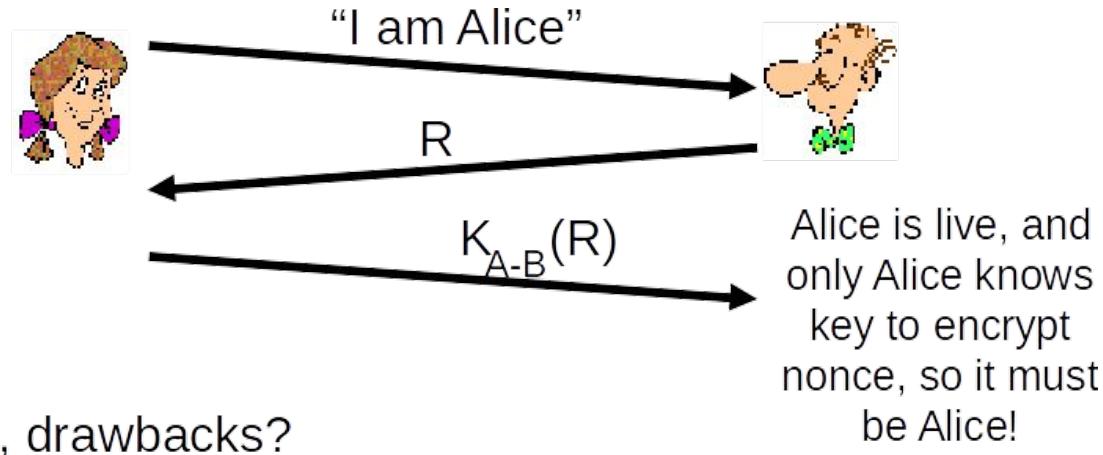
Protocol ap3.1: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.



record
and
playback
still works!

Authentication: another try

- **Goal:** avoid playback attack
- **nonce:** number (R) used only once-in-a-lifetime
- **ap4.0:** to prove Alice “live”, Bob sends Alice **nonce**, R . Alice must return R , encrypted with **shared secret key**



Failures, drawbacks?

outline

- What is network security?
- Principles of cryptography
- Authentication
- Message integrity
- **Securing e-mail**
- Securing TCP connections: SSL
- Network layer security: IPsec
- Securing wireless LANs
- Operational security: firewalls and IDS

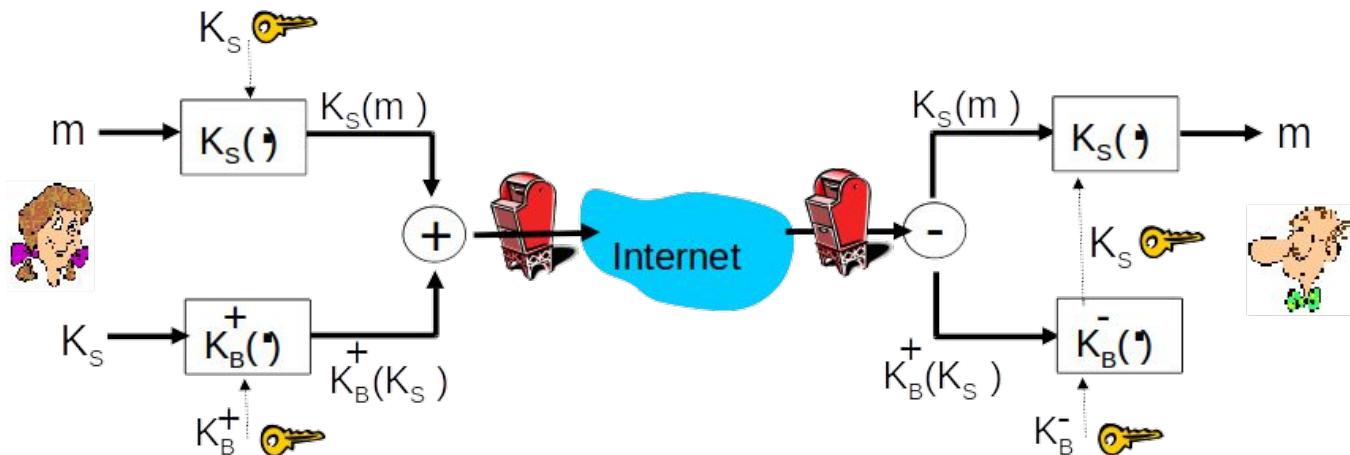
Secure email

Alice wants to send e-mail, m , to Bob.

- Confidentiality
- Sender authentication
 - The receiver (Bob) wants to be sure that the message came from the claimed sender (Alice)
- Message integrity
- Receiver authentication
 - Alice wants to make sure that she is indeed sending the letter to Bob and not someone else who is impersonating Bob

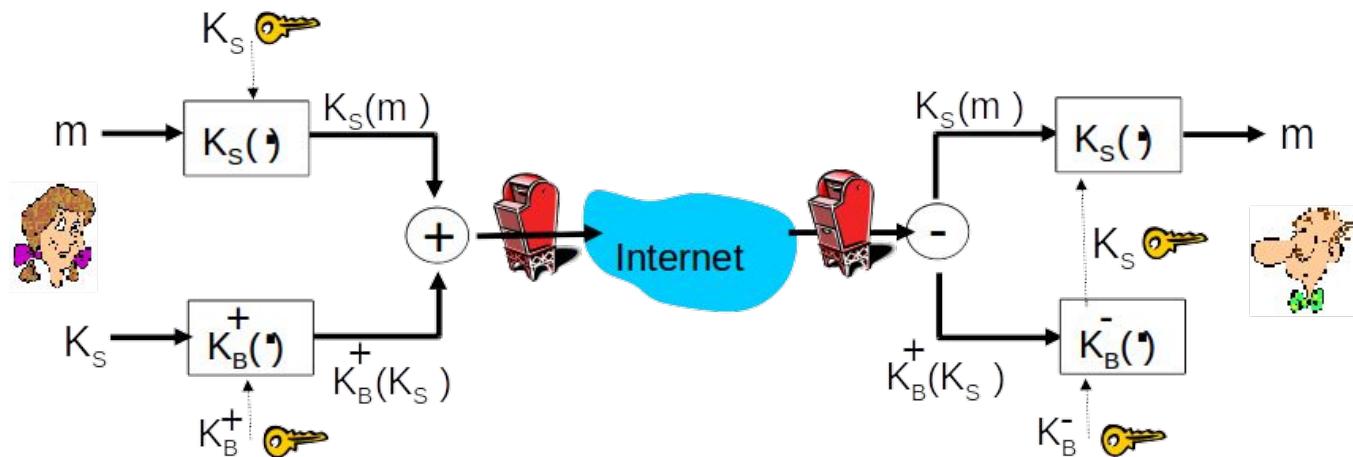
Secure e-mail

- Alice wants to send confidential e-mail, m , to Bob.
- Alice:
 - generates random symmetric private key, K_s
 - encrypts message with K_s (for efficiency)
 - also encrypts K_s with Bob's public key
 - sends both $K_s(m)$ and $K_B^+(K_s)$ to Bob



Secure e-mail

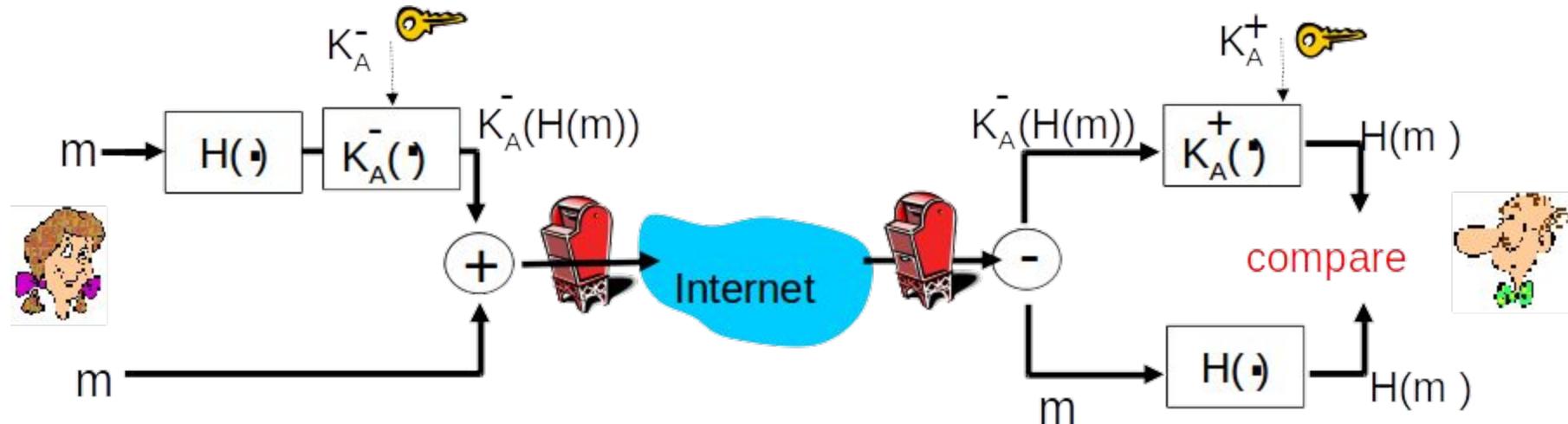
- Alice wants to send confidential e-mail, m , to Bob.
- Bob:
 - uses his private key to decrypt and recover K_s
 - uses K_s to decrypt $K_s(m)$ to recover m



Secure e-mail

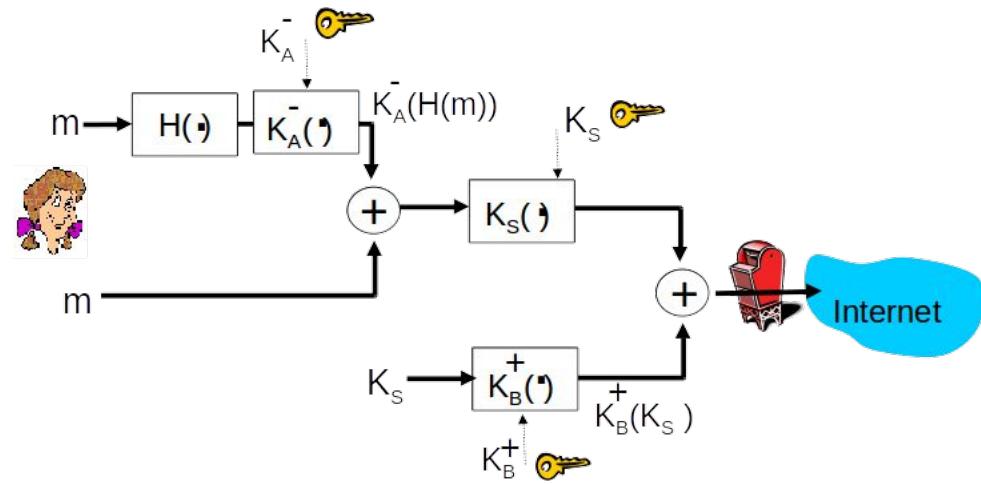
Alice wants to provide sender authentication **message integrity**

- Alice digitally signs message
- sends both message (in the clear) and digital signature



Secure e-mail (continued)

Alice wants to provide secrecy, sender authentication, message integrity.



Alice uses three keys: her private key, Bob's public key, newly created symmetric key

outline

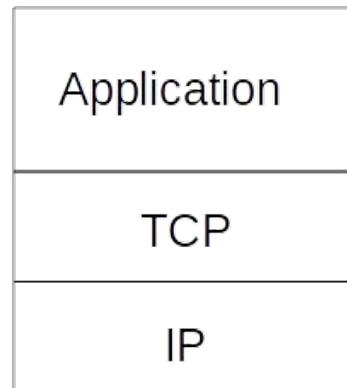
- What is network security?
- Principles of cryptography
- Message integrity
- Authentication
- Securing e-mail
- **Securing TCP connections: SSL**
- Network layer security: IPsec
- Securing wireless LANs
- Operational security: firewalls and IDS

SSL: Secure Sockets Layer

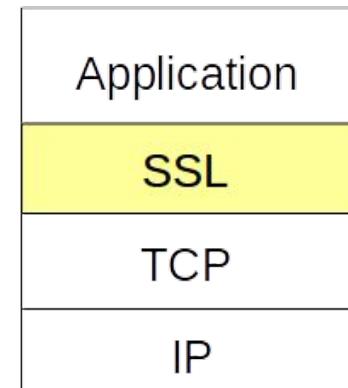
- TLS 1.3 and 1.2 are the most recent versions of TLS
 - TLS: transport layer security, RFC 2246
 - Started as Secure Socket Layer (SSL) by Netscape
- widely deployed security protocol
 - supported by almost all browsers, web servers
 - HTTPS: HTTP protocol running on top of the Transport Layer Security (TLS)
- Provides
 - **confidentiality:** via *symmetric encryption*
 - **integrity:** via *cryptographic hashing*
 - **authentication:** via *public key cryptography*

SSL and TCP/IP

- SSL is part of the the application layer
- available to all TCP applications
 - secure socket interface
- SSL provides application programming interface (API) to applications
 - C and Java SSL libraries/classes readily available



normal application



application with SSL

Toy SSL: a simple secure channel

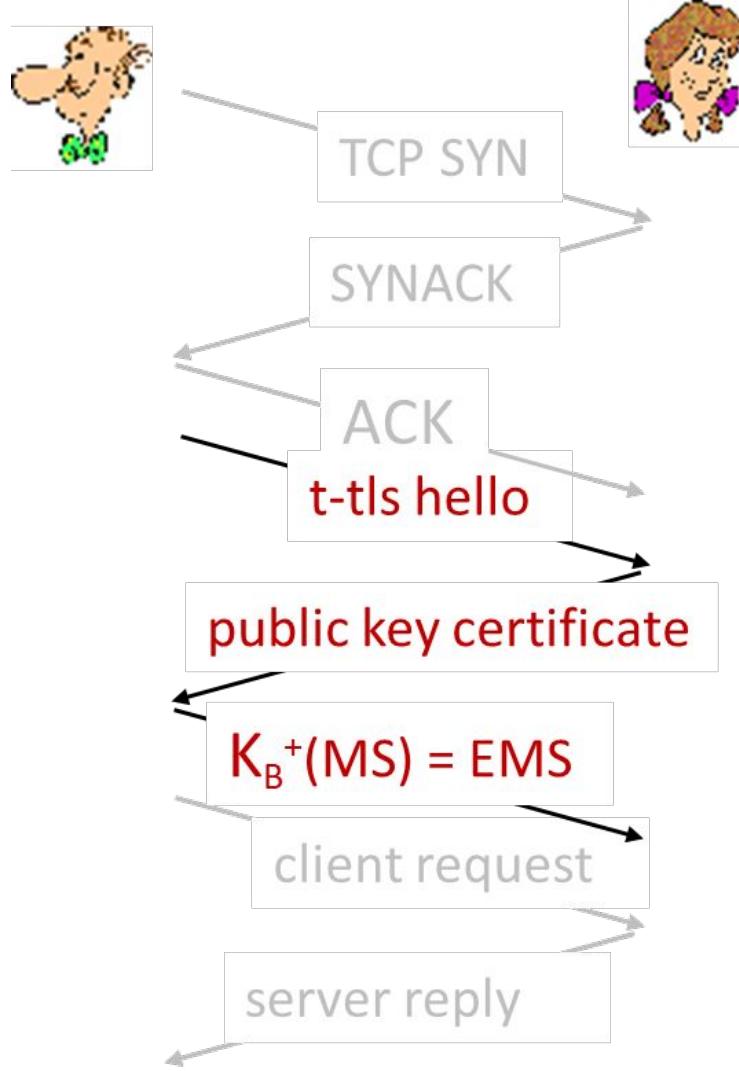
- **handshake**: Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret
- **key derivation**: Alice and Bob use shared secret to derive set of keys
- **data transfer**: data to be transferred is broken up into series of records
- **connection closure**: special messages to securely close connection

Toy SSL : a simple handshake

- Toy SSL handshake phase:
 - Bob establishes TCP connection with Alice
 - Bob verifies that Alice is really Alice
 - Bob sends Alice a master secret key (MS), used to generate all other keys for TLS session
- potential issues:
 - 3 RTT before client can start receiving data (including TCP handshake)

MS: master secret

EMS: encrypted master secret

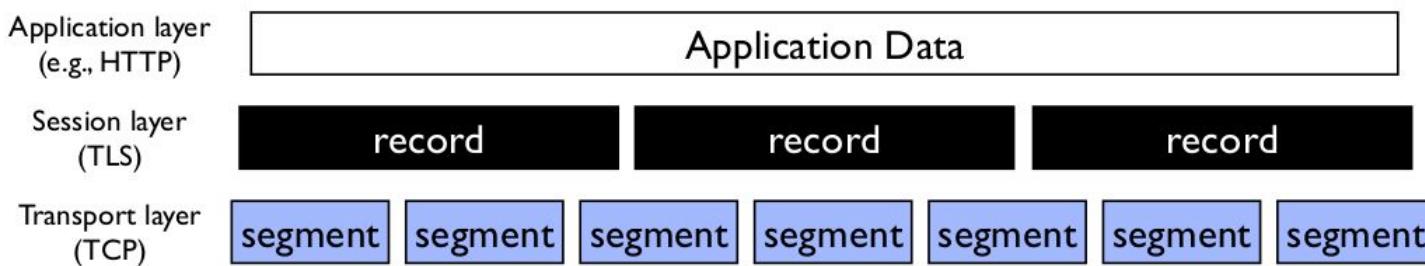


Toy SSL: key derivation

- considered bad to use same key for more than one cryptographic operation
 - use different keys for message authentication code (MAC) and encryption
- four keys:
 - E_B = session encryption key for data sent from Bob to Alice
 - M_B = session MAC key for data sent from Bob to Alice
 - E_A = session encryption key for data sent from Alice to Bob
 - M_A = MAC key for data sent from Alice to Bob
- keys derived from key derivation function (KDF)
 - takes master secret and (possibly) some additional random data and creates the keys

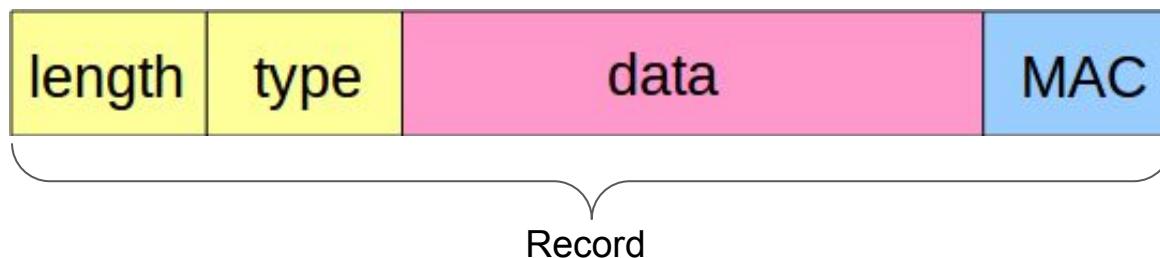
Toy SSL: data transfer

- TLS breaks stream of data from application into records
- Records sent over TCP stream (broken into segments, etc.)



Toy SSL: data records

- why not encrypt data in constant stream as we write it to TCP?
 - where would we put the MAC? If at end, no message integrity until all data processed.
 - We don't want to wait until the end of the TCP session to verify the integrity of all of Bob's data that was sent over the session
- instead, break stream in series of records
 - each record carries a MAC
 - receiver can act on each record as it arrives
- issue: in record, receiver needs to distinguish MAC from data
 - want to use variable-length records

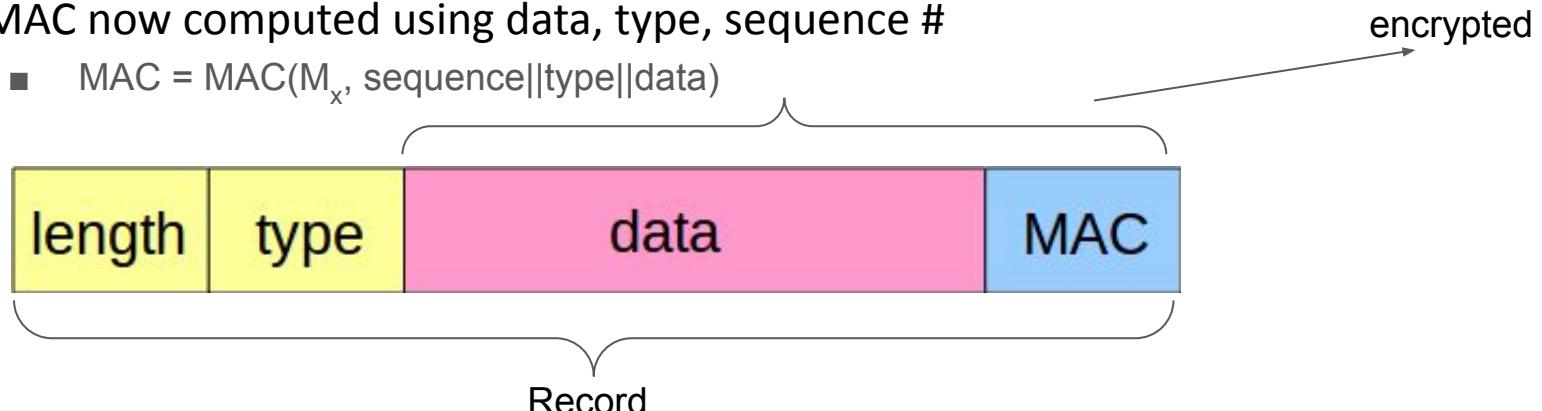


Toy SSL: sequence numbers

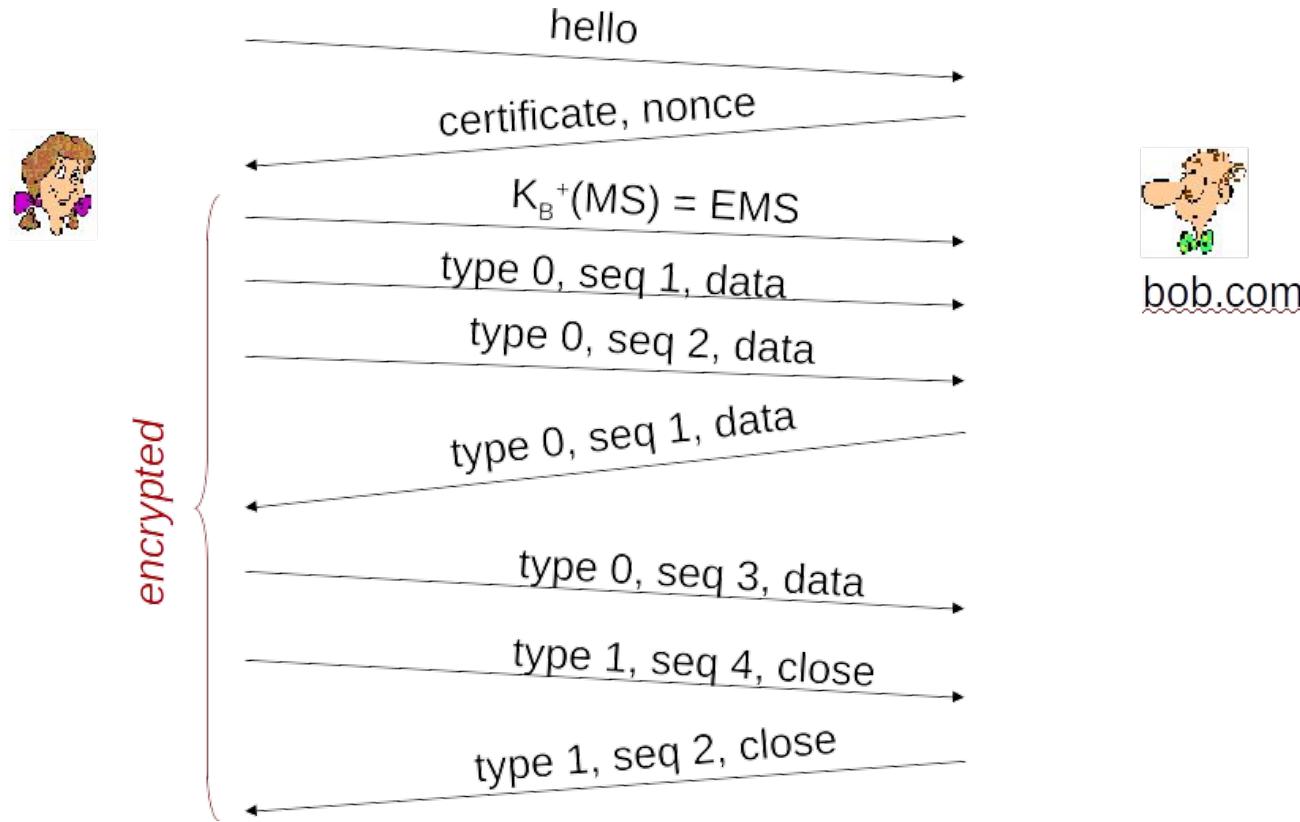
- **possible attacks on data stream?**
 - re-ordering: man-in middle intercepts TCP segments and reorders records
 - (manipulating sequence #s in unencrypted TCP header)
 - replay
- **solution:** put sequence number into MAC:
 - use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)
 - $\text{MAC} = \text{MAC}(M_x, \text{sequence} \parallel \text{data})$
 - note: no sequence number field
- **problem:** attacker could replay **all** records in a TCP session. The above solution works if attacker reorder one record
- **solution:** use *nonce* for each TCP session

Toy SSL: Connection Closure

- **Question:** How to end an SSL connection?
 - Sending TCP FIN segment → leads to **truncation attack**
 - **Truncation attack:** attacker forges TCP connection close segment
 - one or both sides thinks there is less data than there actually is.
- **solution:** record types, with one type for closure
 - type 0 for data; type 1 for closure
 - MAC now computed using data, type, sequence #
 - $\text{MAC} = \text{MAC}(M_x, \text{sequence} \parallel \text{type} \parallel \text{data})$



Toy SSL: summary



Toy SSL isn't complete

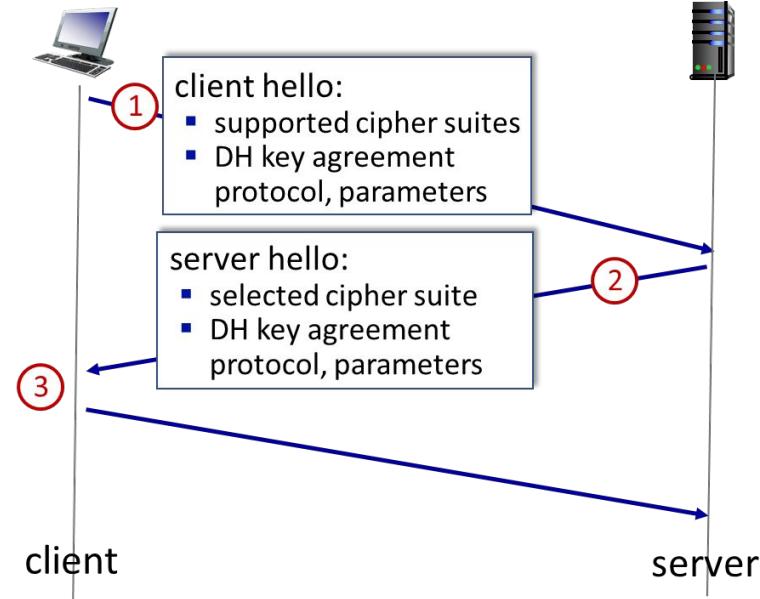
- how long are fields?
- which encryption protocols?
- negotiation
 - allow client and server to support different encryption algorithms
 - allow client and server to choose together specific algorithm before data transfer

SSL cipher suite

- **cipher suite:** algorithms that can be used for key generation, encryption, MAC, digital signature
 - Exchanged between client and server
- SSL supports several cipher suites
- negotiation: client, server agree on cipher suite
 - client offers choice
 - server picks one
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
 - only 5 choices, rather than 37 choices
 - requires Diffie-Hellman (DH) for key exchange, rather than DH or RSA
 - combined encryption and authentication algorithm (“authenticated encryption”) for data rather than serial encryption, authentication
 - 4 based on AES
 - HMAC uses SHA (256 or 284) cryptographic hash function

Real SSL: handshake

1. client sends list of algorithms it supports, along with client nonce
2. server chooses algorithms from list; sends back: choice + certificate + server nonce
3. client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages

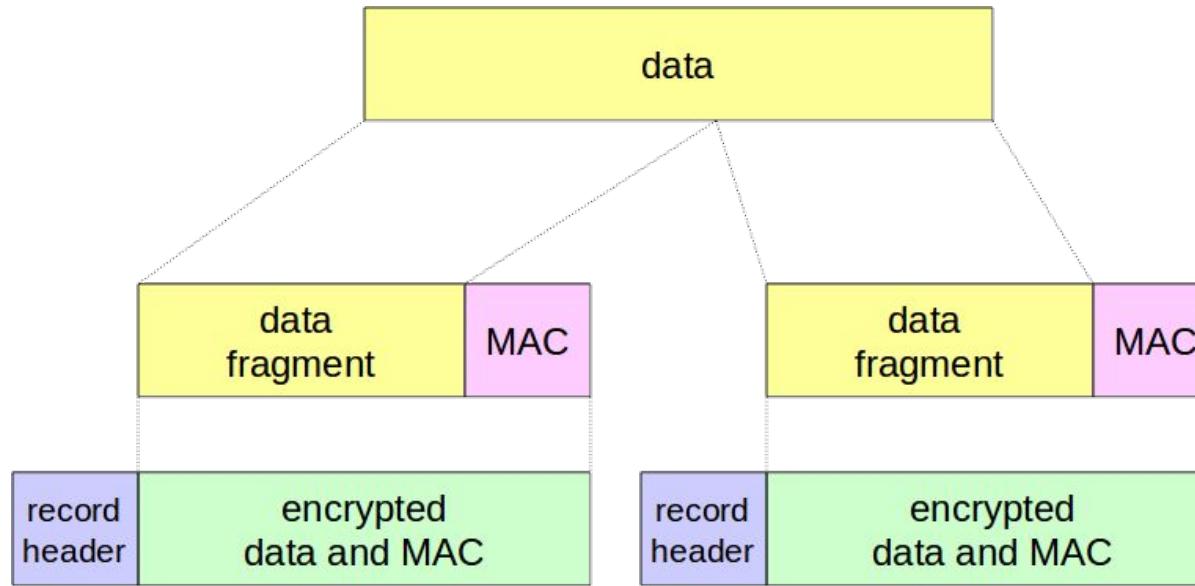


Real SSL: handshaking

last 2 steps protect handshake from tampering

- client typically offers range of algorithms, some strong, some weak
- man-in-the middle could delete stronger algorithms from list
- last 2 steps prevent this
 - last two messages are encrypted

SSL record protocol



record header: content type; version; length

MAC: includes sequence number, MAC key M_x

fragment: each SSL fragment 2^{14} bytes (~ 16 Kbytes)

outline

- What is network security?
- Principles of cryptography
- Message integrity
- Authentication
- Securing e-mail
- Securing TCP connections: SSL
- **Network layer security: IPsec**
- Securing wireless LANs
- Operational security: firewalls and IDS

What is network-layer security?

between two network entities:

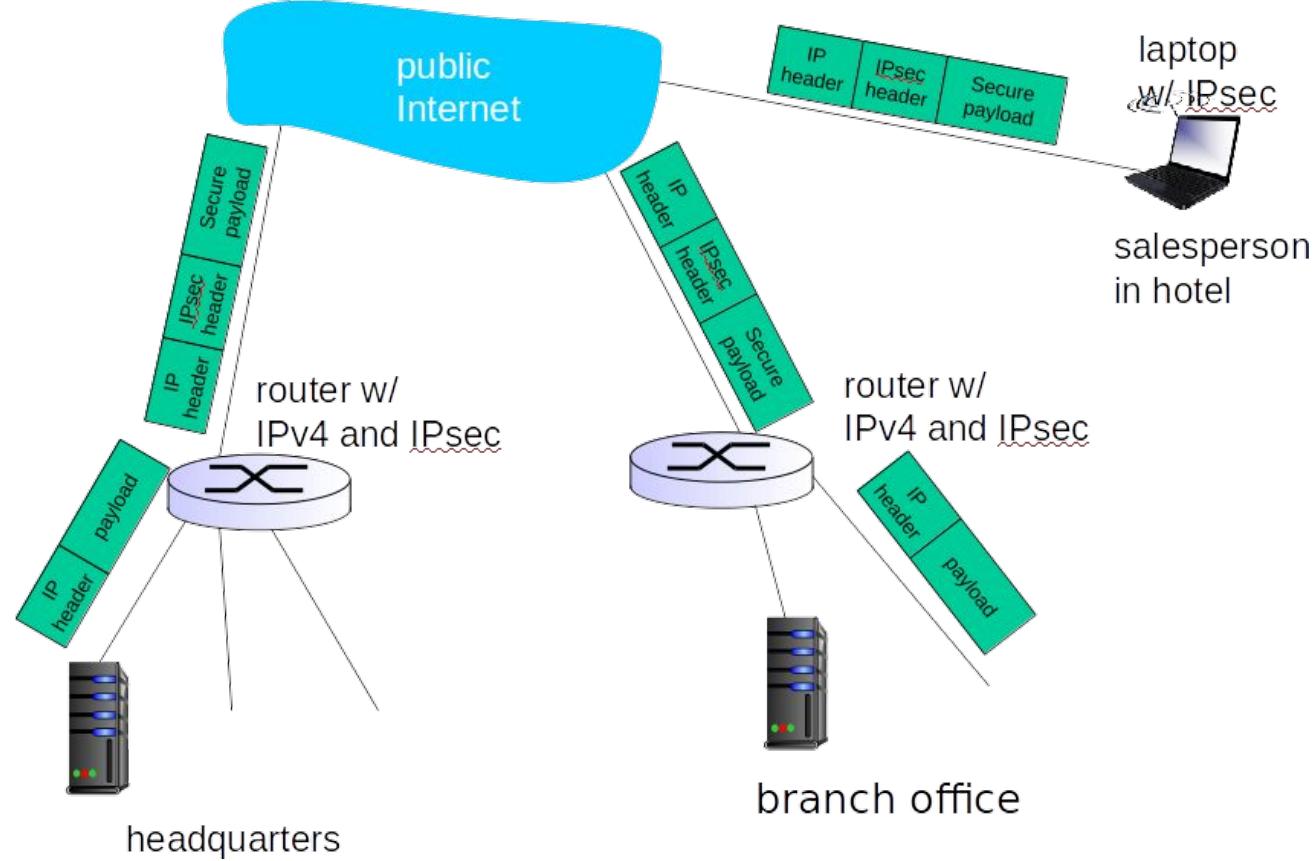
- sending entity encrypts datagram payload, payload could be:
 - TCP or UDP segment, ICMP message, OSPF message
- all data sent from one entity to other would be hidden:
 - web pages, e-mail, P2P file transfers, TCP SYN packets ...
- “blanket coverage”

Virtual Private Networks (VPNs)

motivation:

- institutions often want private networks for security.
 - costly: separate routers, links, DNS infrastructure.
- VPN: institution's inter-office traffic is sent over public Internet instead
 - encrypted before entering public Internet
 - logically separate from other traffic

Virtual Private Networks (VPNs)

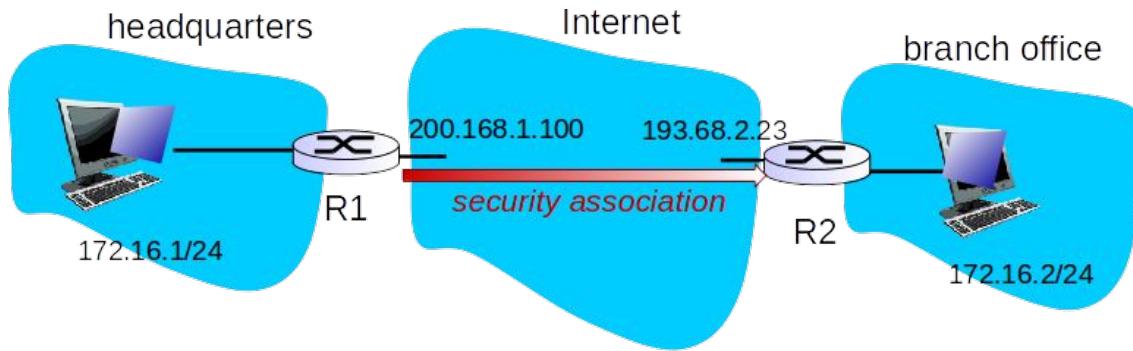


IPsec services

- IPsec: IP security protocol: provides security at the network layer
 - data integrity
 - origin authentication
 - replay attack prevention
 - confidentiality

IPSec

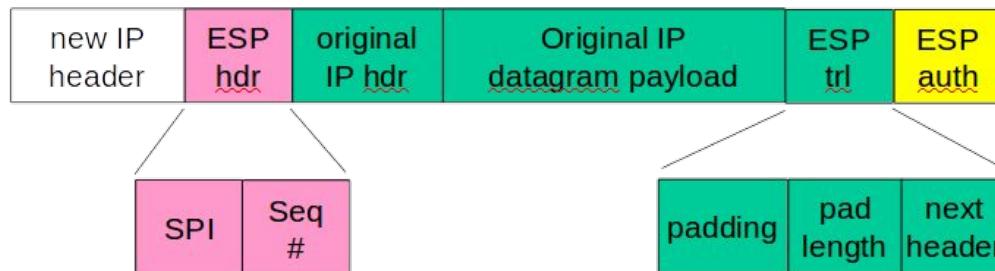
Logical connection



← “enchilada” authenticated →

← encrypted →

IPSec datagram format

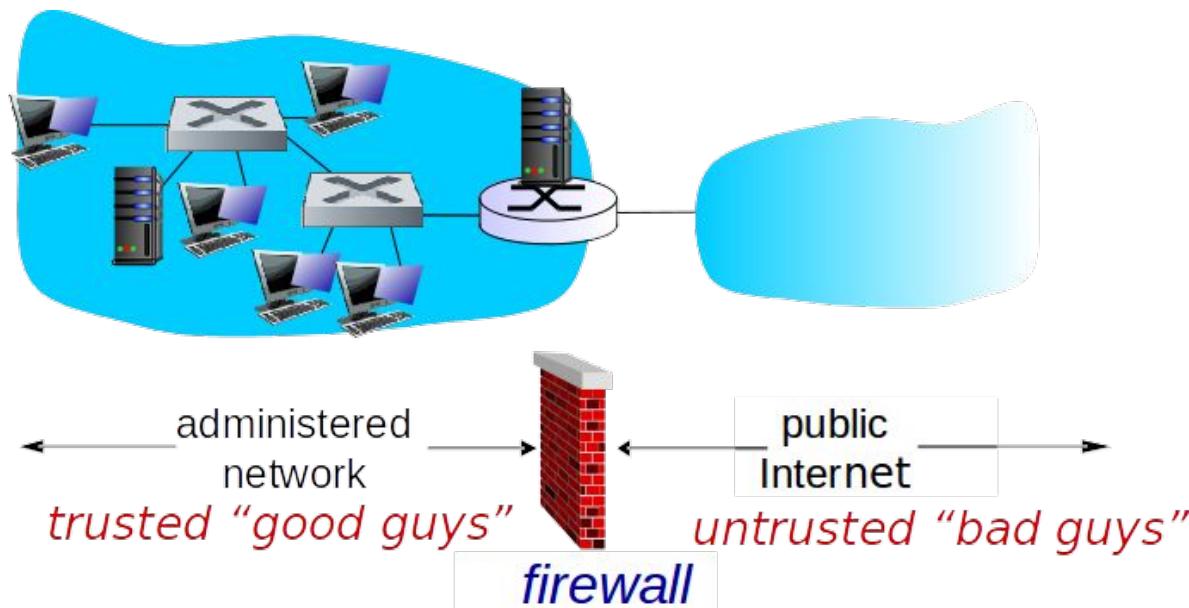


outline

- What is network security?
- Principles of cryptography
- Authentication
- Message integrity
- Securing e-mail
- Securing TCP connections: SSL
- Network layer security: IPsec
- Securing wireless LANs
- **Operational security: firewalls and IDS**

Firewalls

- Firewalls:
 - isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others

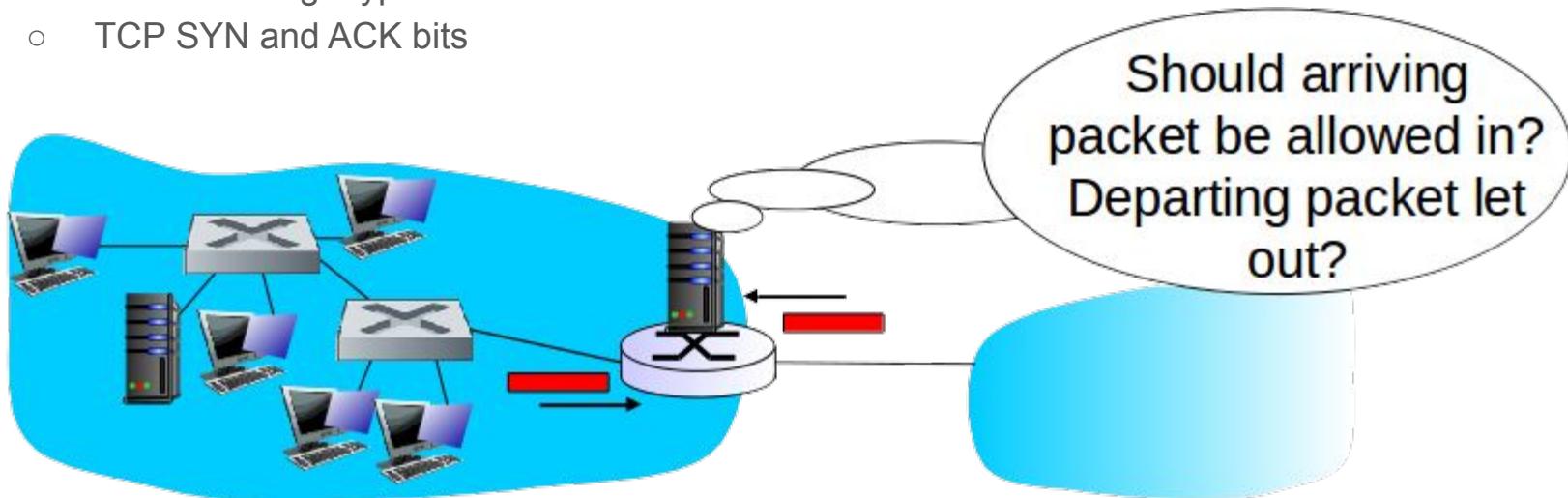


Firewalls: why

- prevent denial of service attacks:
 - SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections
- prevent illegal modification/access of internal data
- allow only authorized access to inside network
 - set of authenticated users/hosts
- three types of firewalls:
 - stateless packet filters
 - stateful packet filters
 - application gateways

Stateless packet filtering

- internal network connected to Internet via **router firewall**
- router **filters packet-by-packet**, decision to forward/drop packet based on:
 - source IP address, destination IP address
 - TCP/UDP source and destination port numbers
 - ICMP message type
 - TCP SYN and ACK bits



Stateless packet filtering: example

- **example 1:** block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
 - **result:** all incoming, outgoing UDP flows and telnet connections are blocked
- **example 2:** block inbound TCP segments with ACK=0.
 - **result:** prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.
 - Recal: The first segment in every TCP connection has the ACK bit set to 0, whereas all the other segments in the connection have the ACK bit set to 1.

Stateless packet filtering: more examples

<i>Policy</i>	<i>Firewall Setting</i>
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Access Control Lists

- Firewall rules are implemented in routers with **access control lists**, with each router interface having its own list.
- **ACL**: table of rules, applied top to bottom to incoming packets: (action, condition) pairs: looks like OpenFlow forwarding

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

Stateful packet filtering

- **stateless packet filter:** filtering decisions are made on each packet in isolation
 - Example rule admits packets with dest port = 80, ACK bit set, even though no TCP connection established:
- **stateful packet filter:** track status of every TCP connection
 - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets belong to a connection
 - timeout inactive connections at firewall: no longer admit packets

action	source address	dest address	protocol	source port	dest port	flag bit
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK

Stateful packet filtering

ACL augmented to indicate need to check connection state table before admitting packet

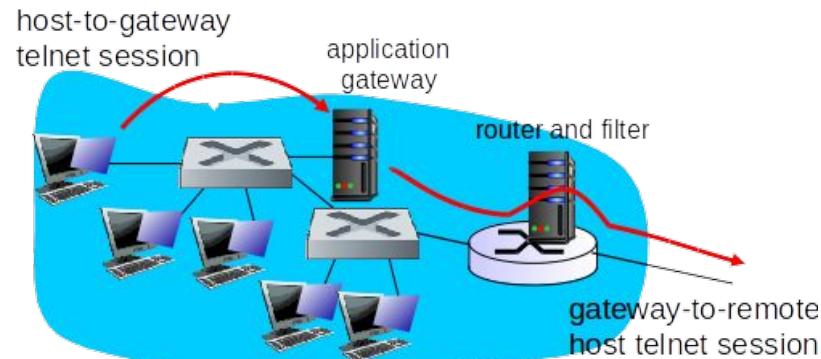
action	source address	dest address	proto	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

Application gateways

filter packets on application data as well as on IP/TCP/UDP fields.

example: allow select internal users to telnet outside

1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway.



Limitations of firewalls, gateways

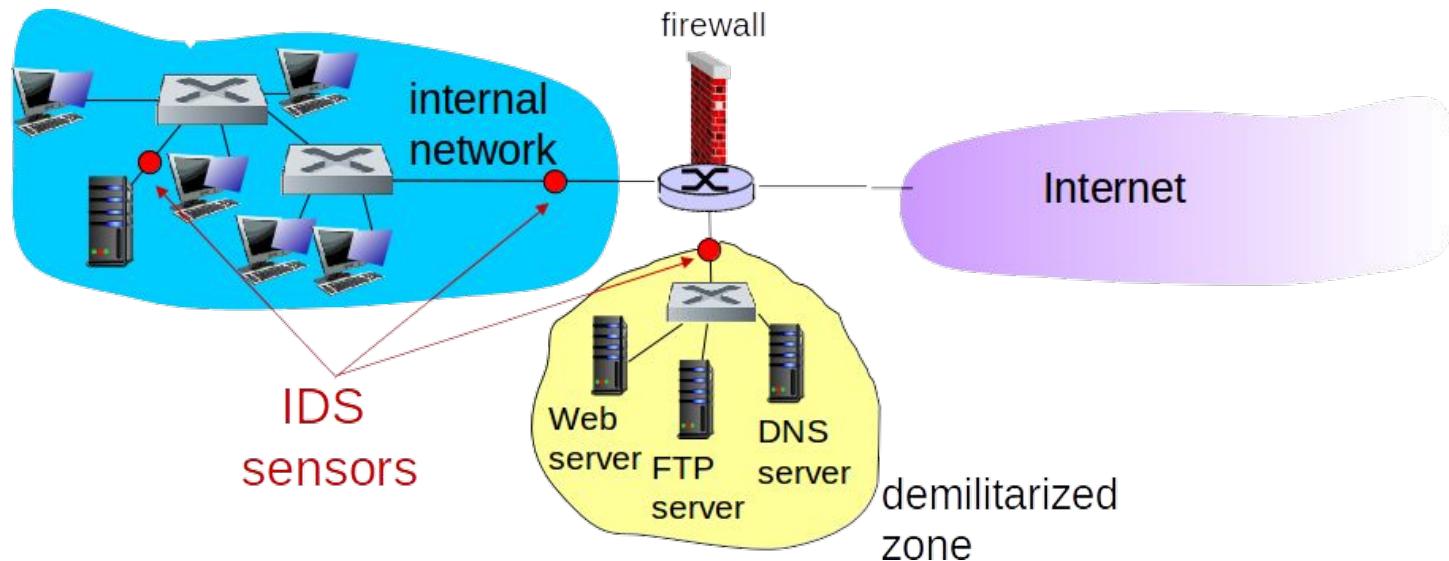
- **IP spoofing:** router can't know if data “really” comes from claimed source
- if multiple applications need special treatment, each has own application gateway
- client software must know how to contact gateway.
 - e.g., must set IP address of proxy in Web browser
- **tradeoff:** degree of communication with outside world, level of security

Intrusion detection systems

- packet filtering:
 - operates on TCP/IP headers only
 - no correlation check among sessions
- IDS: intrusion detection system
 - **deep packet inspection**: look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
 - **examine correlation** among multiple packets
 - port scanning
 - network mapping
 - DoS attack

Intrusion detection systems

- Multiple IDSs: different types of checking at different locations
- Because they do more inspection, putting them in one location reduce the performance of the network



Network Security (summary)

- basic techniques.....
 - cryptography (symmetric and public)
 - message integrity
 - end-point authentication
- used in many different security scenarios
 - secure email
 - secure transport (SSL)
 - IP sec
- operational security: firewalls and IDS