UMBC

AN HONORS UNIVERSITY IN MARYLAND

# IS 450/IS 650–
# Data Communications and Networks

# Course Review
# Final Exam

Nirmalya Roy
Department of Information Systems
University of Maryland Baltimore County

# Final Exam

- When: Tuesday (5/19) 3:30pm - 5:30pm
- Where: In Class

- Closed book, Closed notes
- Transport Layer (Chapter 3) and Network Layer (Chapter 4)

- Materials for preparation:
  - Lecture Slides
  - **Quiz 3 and Homework 2**
  - Textbook
    - Computer Networking: A Top Down Approach

# Course Overview

- Transport Layer (Chapter 3)
  - Reliable Data Transfer
    - Pipelined Reliable Data Transfer
    - Go-Back-N (GBN)
    - Selective Repeat (SR)
  - TCP Congestion Control, Flow Control and RTT Estimation

- Network Layer (Chapter 4)
    - Network layer services
    - IPv4 addressing (subnet, DHCP etc.)
    - Routing algorithms (link state, distance vector etc.)

# Chapter 3 Outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 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 segment handled independently of others

Why is there a UDP?

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

# Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

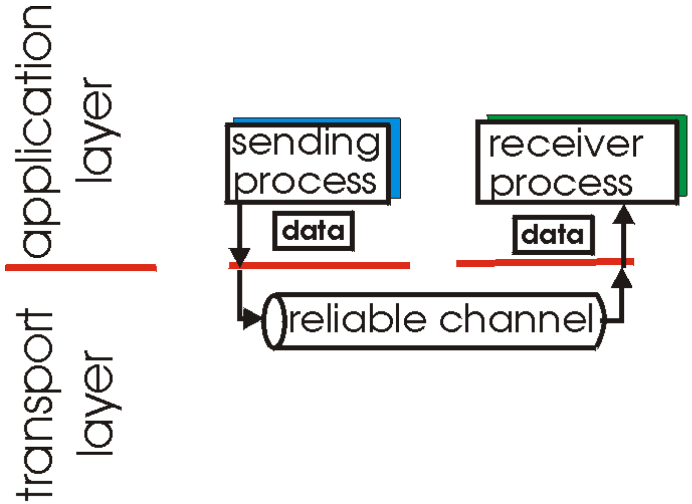
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# Principles of reliable data transfer

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

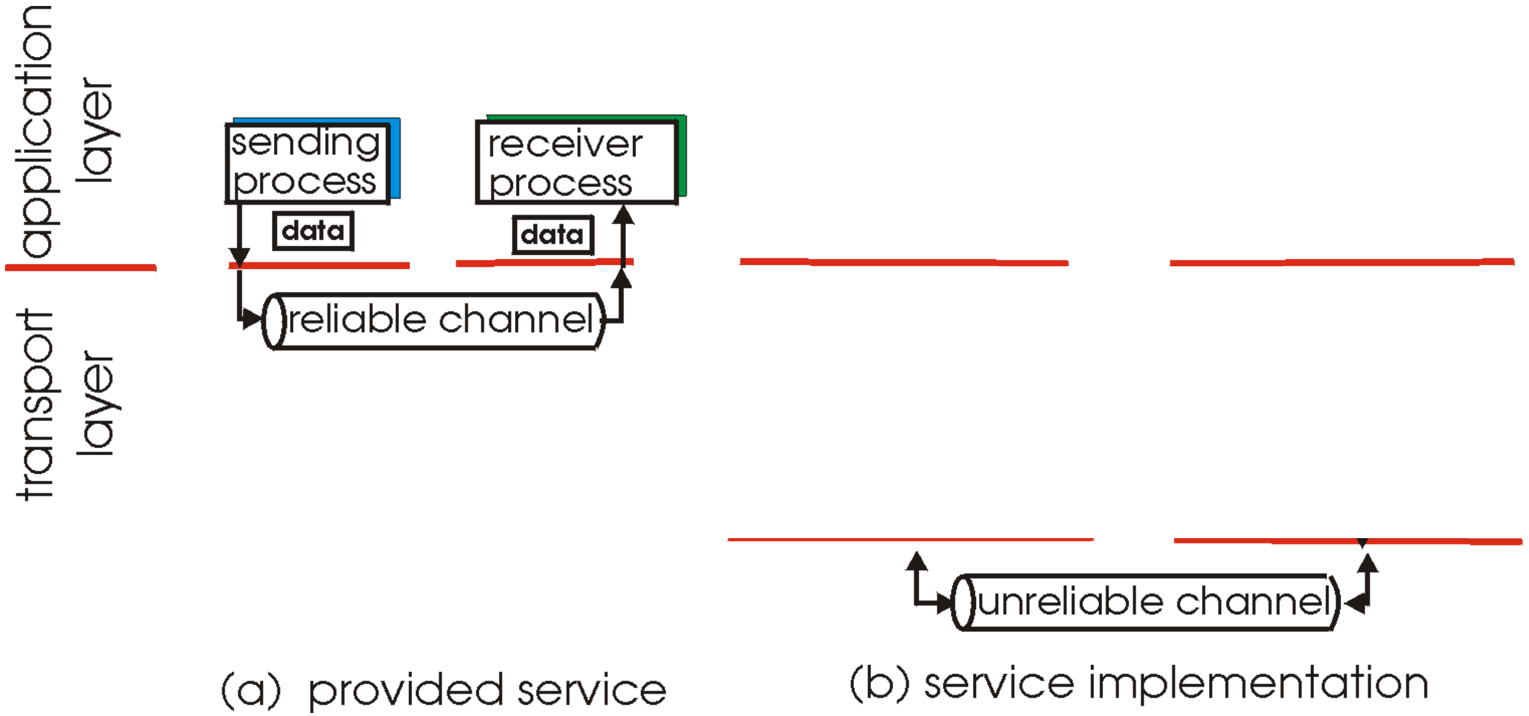



(a) provided service

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

# Principles of reliable data transfer

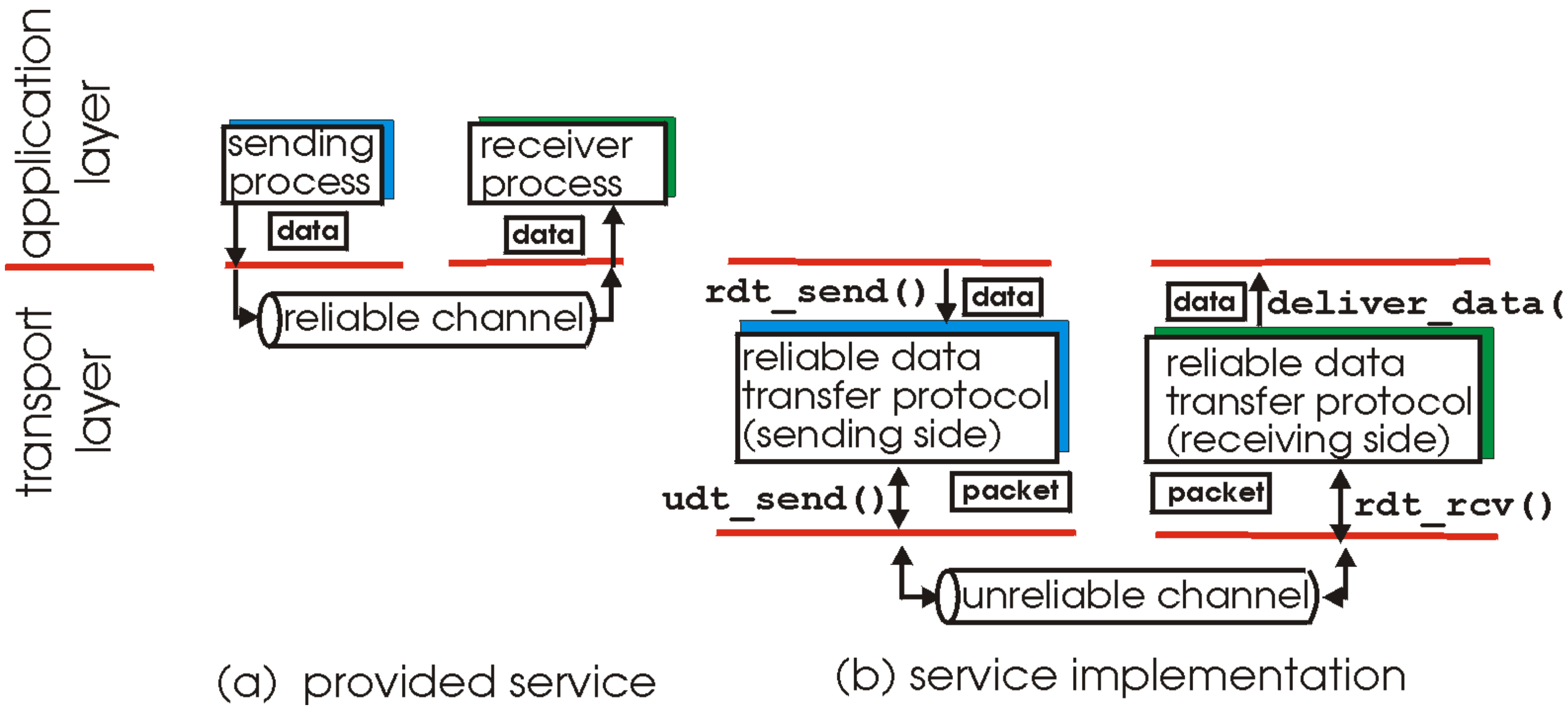
- important in application, transport, link layers
  - top-10 list of important networking topics!




- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

# Principles of reliable data transfer

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



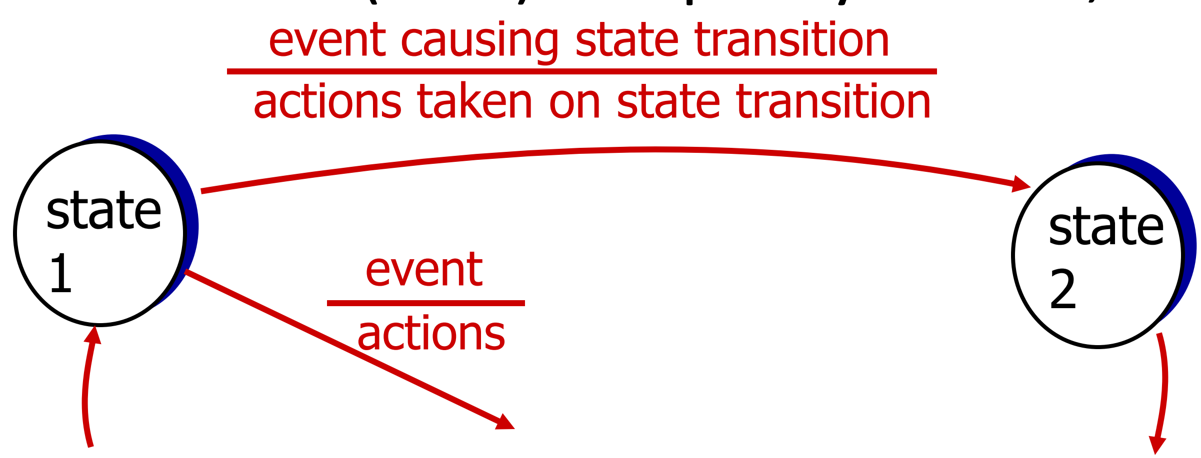

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

# Reliable data transfer: getting started

We' ll:

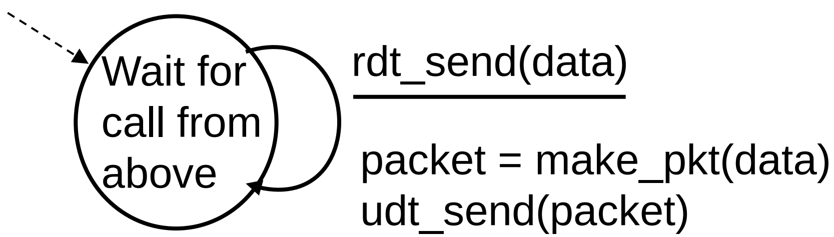
- incrementally develop sender, receiver sides of <u>r</u>eliable <u>d</u>ata <u>t</u>ransfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver

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

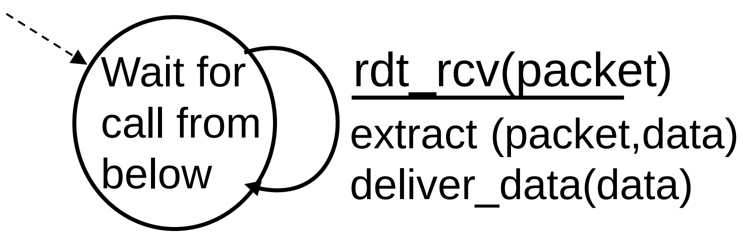


# rdt1.0: reliable transfer over a reliable channel

- 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




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:

*How do humans recover from "errors" during conversation?*

# rdt2.0 has a fatal flaw!

what happens if ACK/NAK corrupted?

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

**handling duplicates:**

- sender retransmits current pkt if ACK/NAK corrupted
- sender adds ***sequence number*** to each pkt
- receiver discards (doesnt deliver up) duplicate pkt

stop and wait

sender sends one packet, then waits for receiver response

# rdt3.0: channels with errors *and* loss

new assumption:

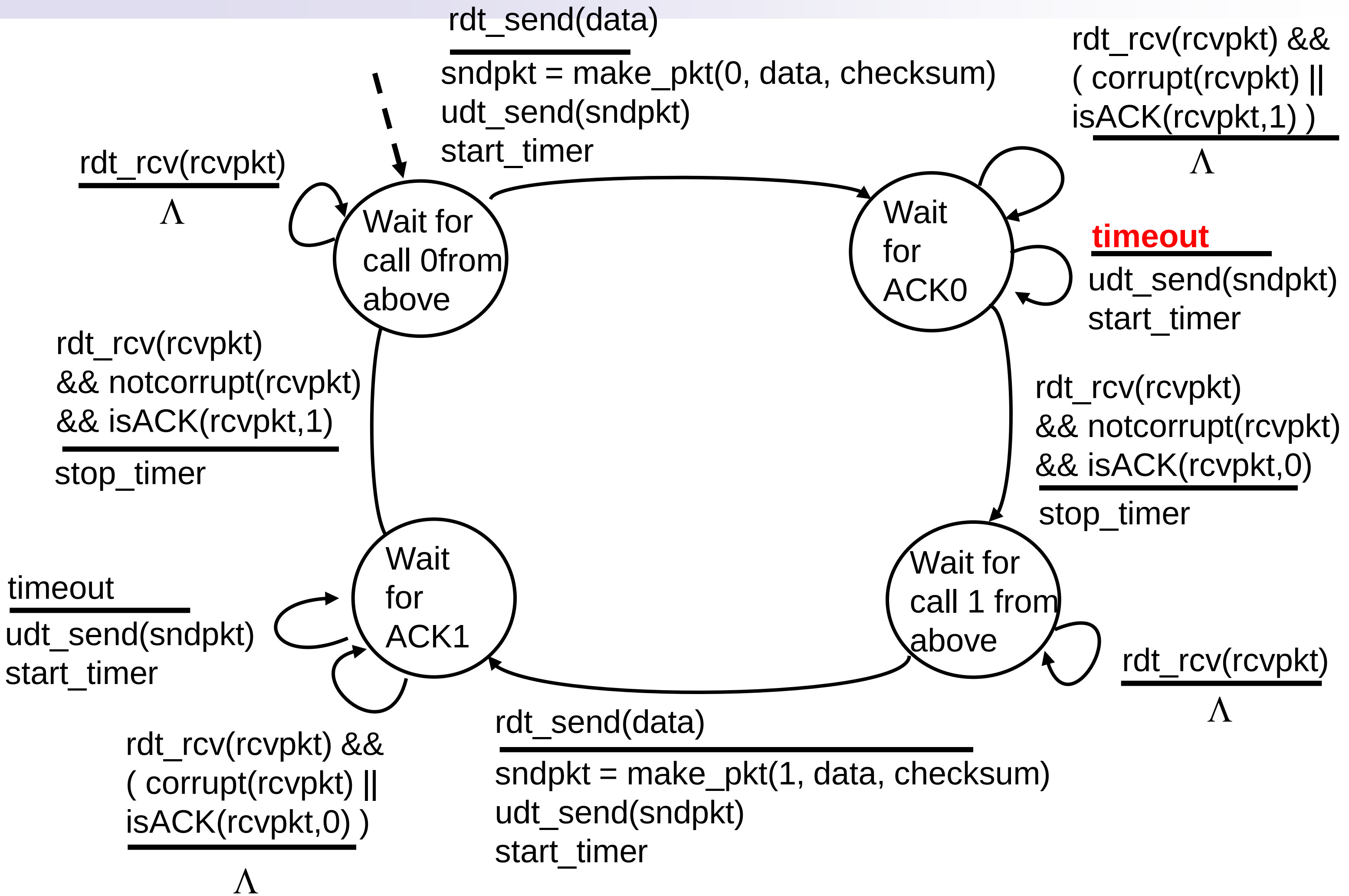
underlying channel can **also lose packets** (data, ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help … but not enough

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
- **requires countdown timer**

# rdt3.0 sender

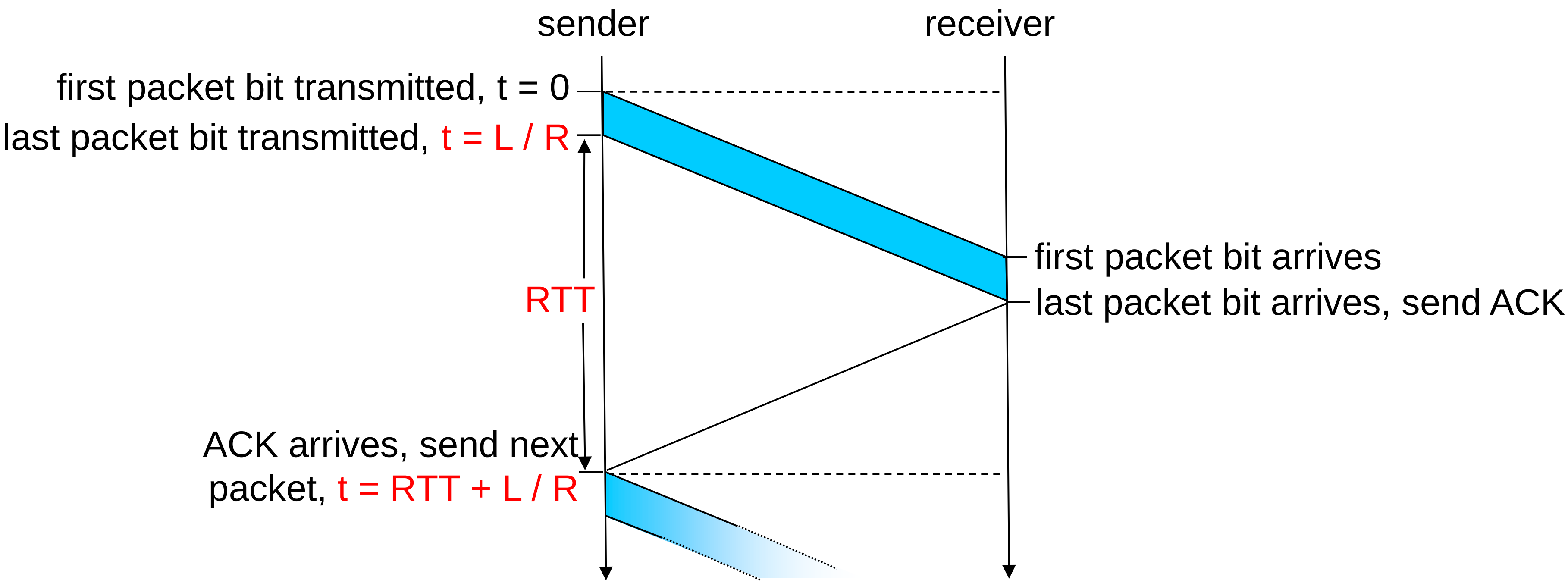


# Some transmission methods

- Reliable data transfer over a channel with Bit Errors
  - Positive acknowledgement (ACK)
  - Negative acknowledgement (NAK)
  - ARQ (Automatic Repeat reQuest) protocols
    - Error detection
    - Receiver feedback
    - Retransmission

- Stop & Wait
- Pipelined
  - Go Back N
  - Selective Repeat

# Stop-and-wait operation

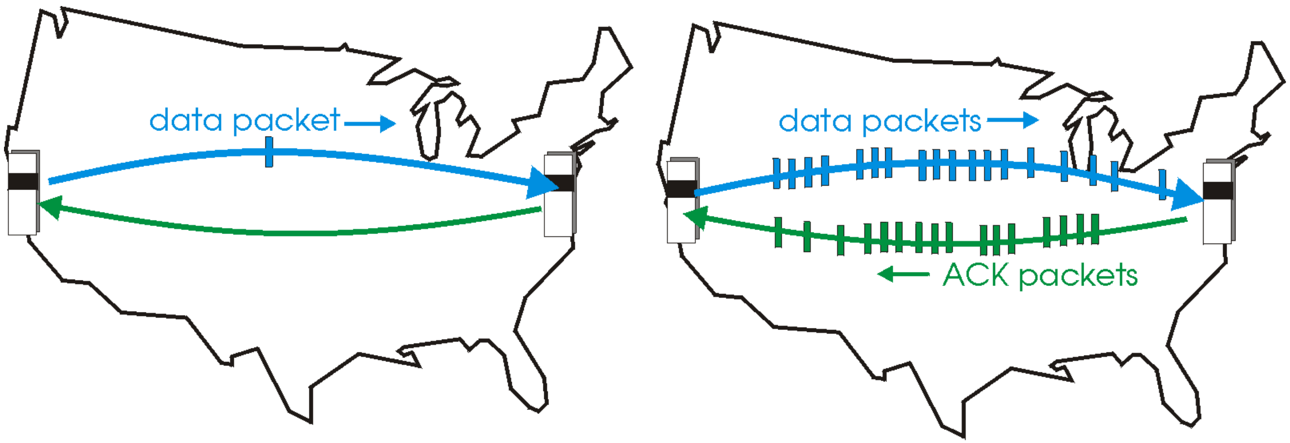



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

# Pipelined protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

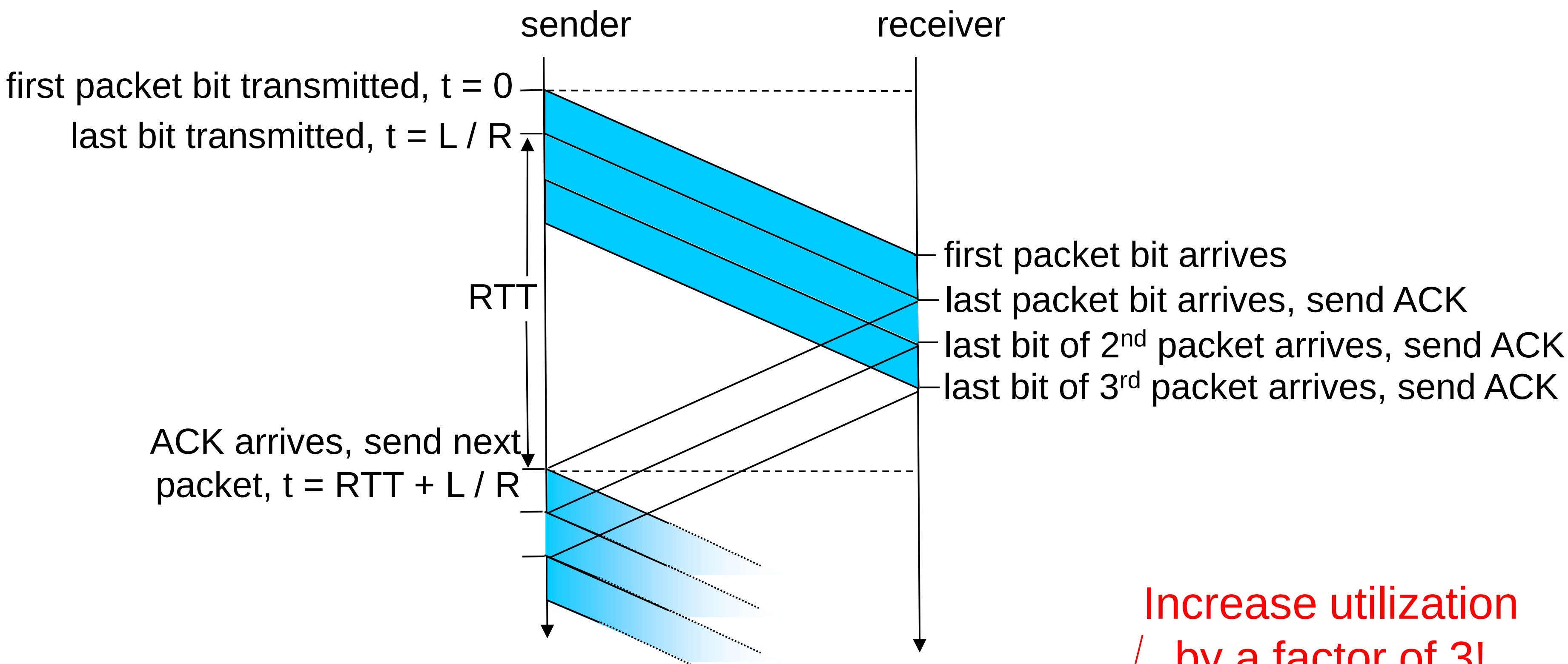
- range of sequence numbers must be increased
- buffering at sender and/or receiver




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

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

# Pipelining: increased utilization



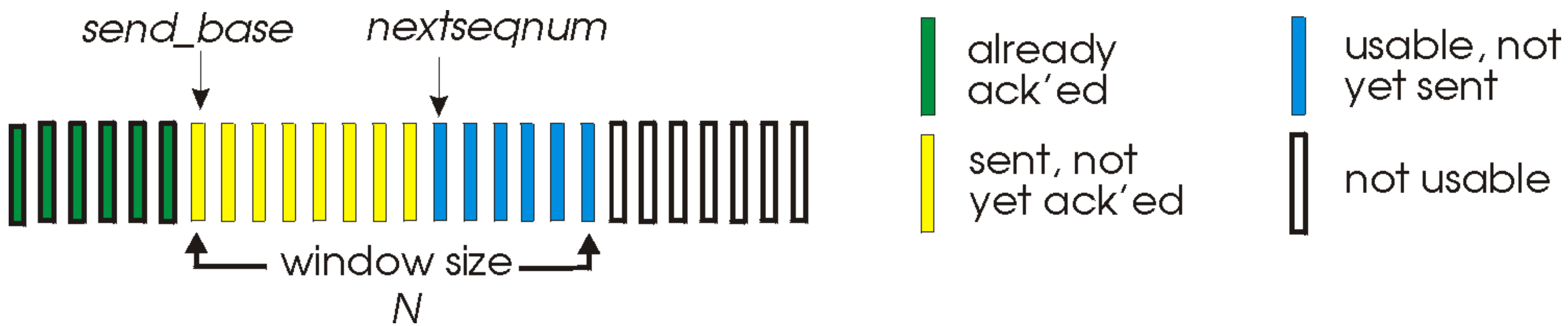

Increase utilization by a factor of 3!

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

# Go-Back-N

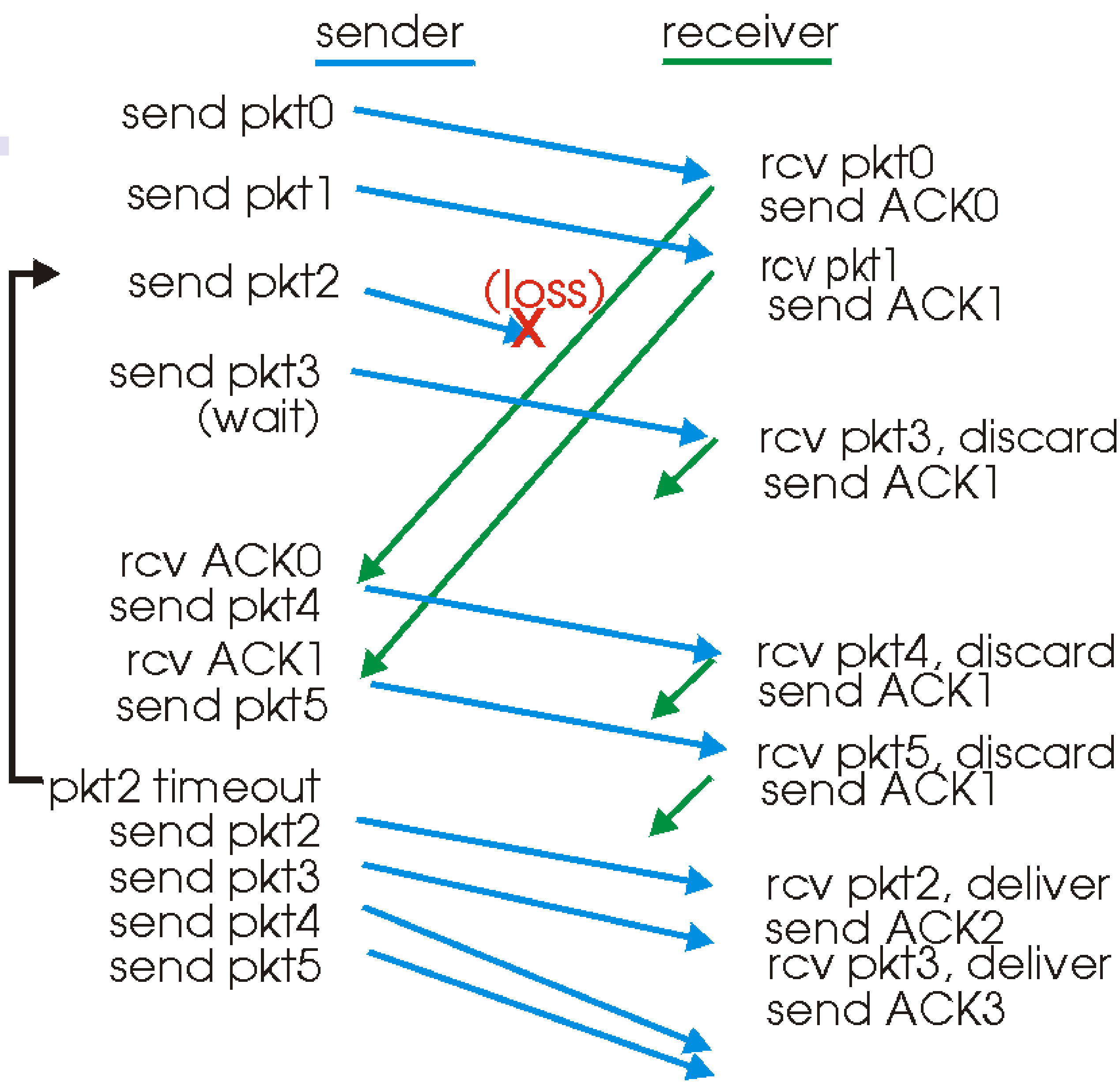
Sender:

- k-bit seq # in pkt header
- “window” of up to N, consecutive unack’ed pkts allowed




- ACK(n): ACKs all pkts up to, including seq # n - “cumulative ACK”
  - may receive duplicate ACKs
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window
- N referred to as the window size and GBN as a sliding-window protocol
- Why we limit N? (flow control, TCP congestion control)

# GBN in action



# Selective Request

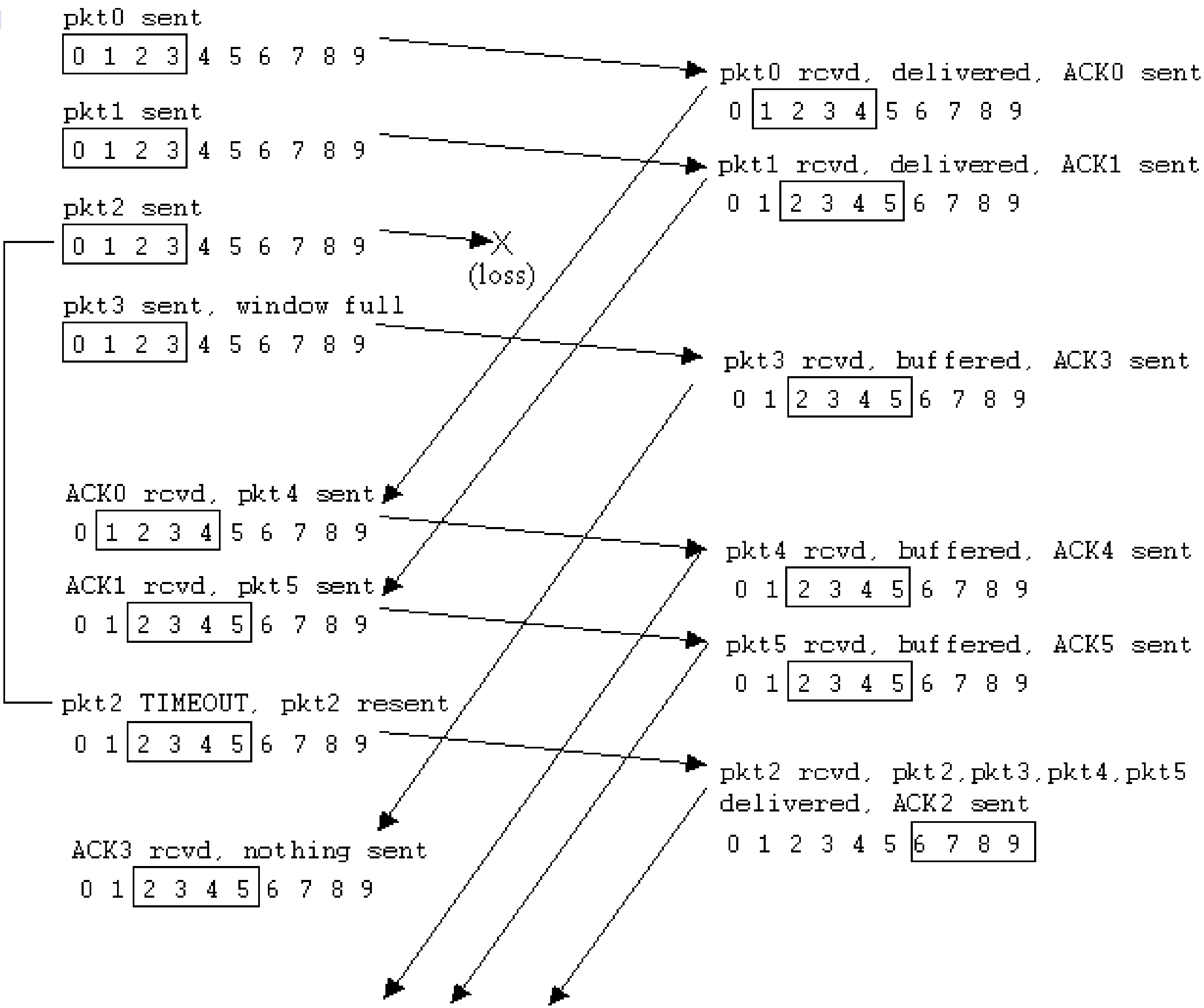
- Makes sense to transmit only the lost packets
  - But this is true under what assumption ?
    - GBN suffers from performance  problems
      - Window size and bandwidth-delay product are both large, many pkts in pipeline
      - Single pkt error cause GBN to retransmit a large # of pkts, many unnecessarily
      - Probability of channel error increases, the pipeline can become filled with these unnecessary transmissions
  - Can you say a case in which Go-BACK-N might be better
    - GBN protocol allows the sender to potentially “fill the pipeline” with packets
      - Increase the channel utilization than stop-and-wait protocols
  - SR protocols avoid unnecessary retransmissions
    - Sender only retransmits pkts that are received in error at receiver (lost/corrupted)

# Selective Repeat (SR)

- SR receiver acknowledge a correctly received packet whether or not it is in order

- Out-of-order pkts are buffered
  - If any missing pkts (with lower seq #) are received, a batch of pkts can be delivered in order to the upper layer

# Selective repeat in action



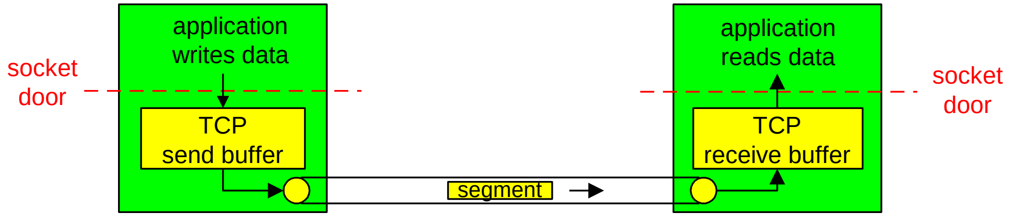
# Agenda

- Reliable Transport
  - Stop-and-wait
  - Pipelined
    - Go back N
    - Selective Request

- TCP
  - Congestion Control
  - Flow Control

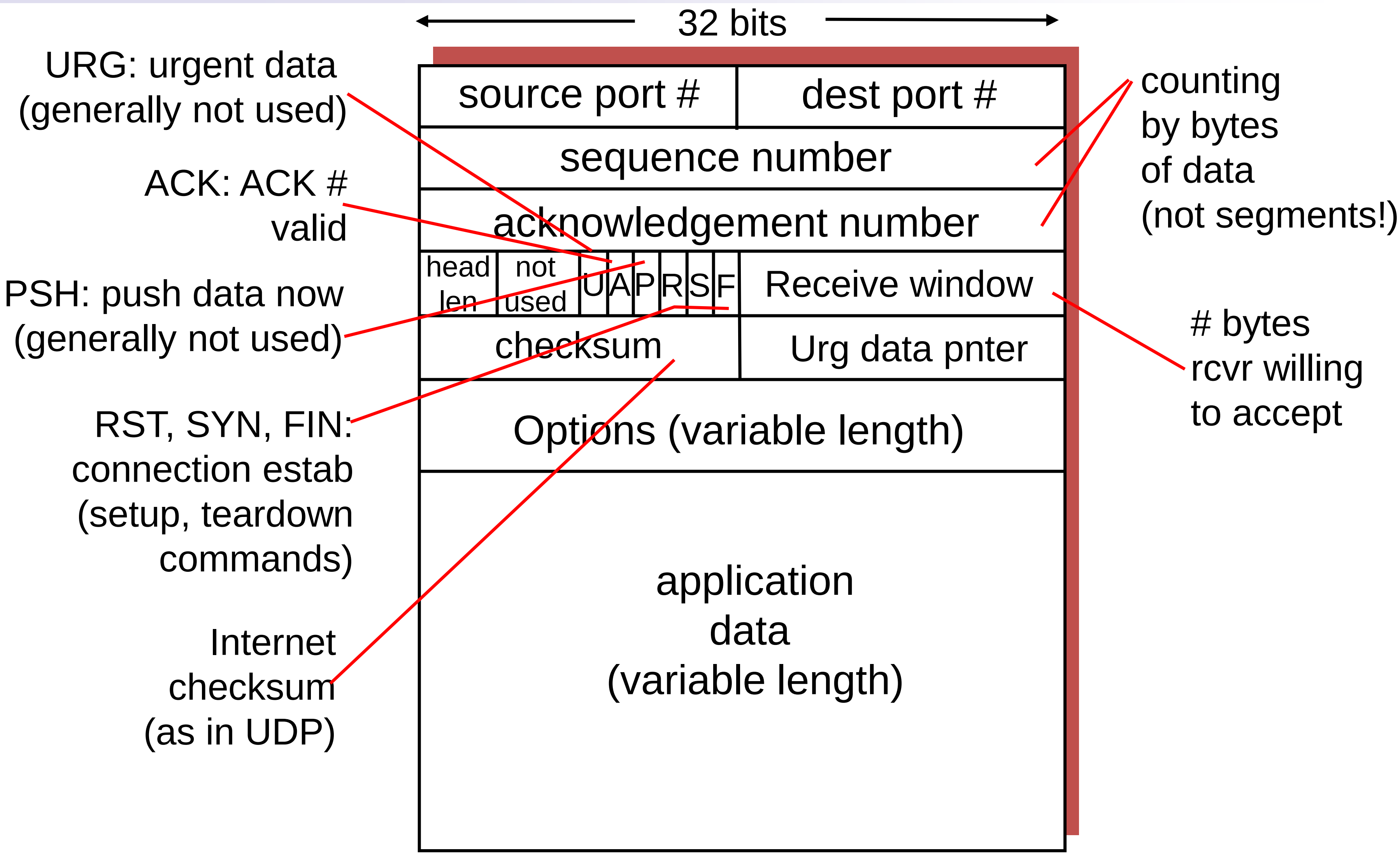
# TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no “message boundaries”
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*




- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init’s sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

# TCP segment structure



# TCP: Connection-Oriented Transport

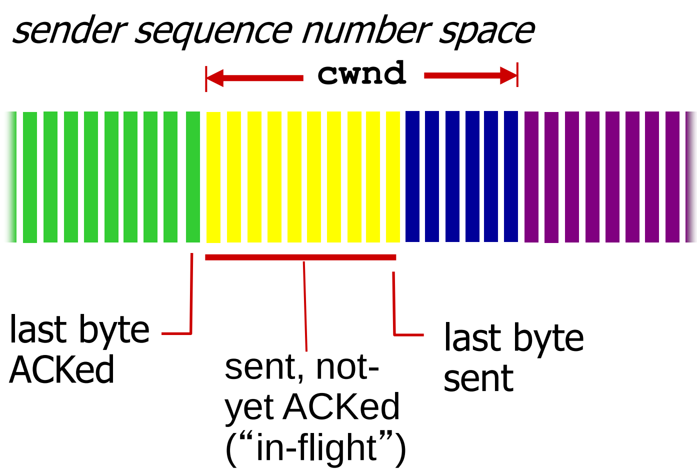
- TCP has 3 main components
  - Reliable transmission
  - Congestion Control
  - Flow Control

# Congstion Control

# TCP Congestion Control

- Problem Definition
  - How much data should I pump into the network to ensure
    - Intermediate router queues not filling up
    - Fairness achieved among multiple TCP flows
- Why is this problem difficult?
  - TCP cannot have information about the network
  - Only TCP receiver can give some feedbacks
- Approach: sender limit the rate of sending traffic as a function of perceived network congestion
  - How does a TCP sender limit the rate?
  - How does TCP sender perceive that there is congestion?
  - What algorithm should the sender use?

# TCP Congestion Control: details




- sender limits transmission:

$$\boxed{\texttt{LastByteSent-LastByteAcked} \leq \texttt{cwnd}}$$

- **cwnd** is dynamic, function of perceived network congestion

*TCP sending rate:*

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

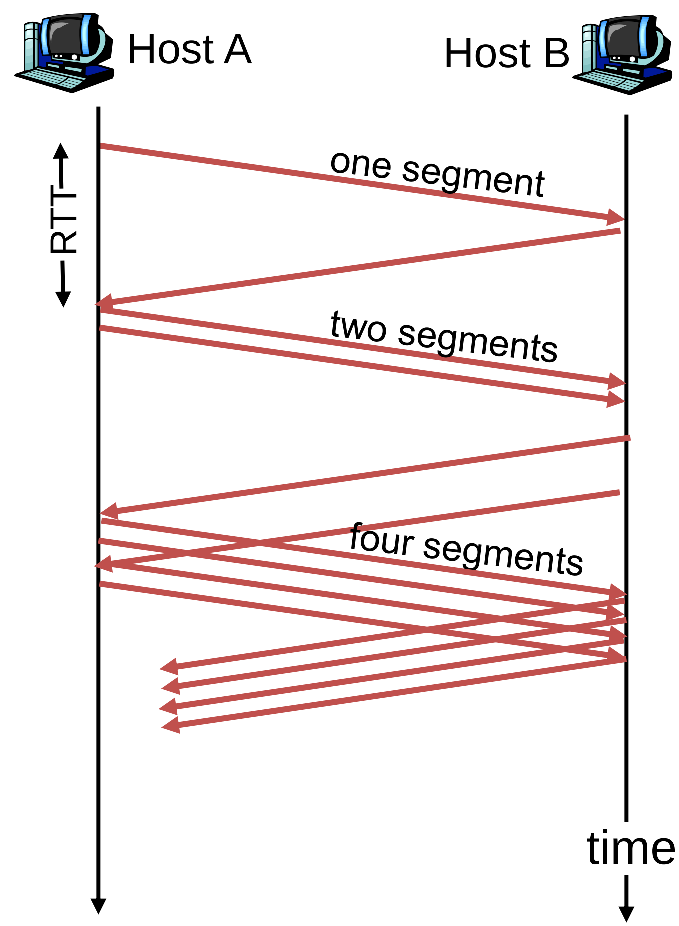
$$\boxed{\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}}$$

# TCP Congestion Control Algorithm

- A loss segment implies congestion
  - TCP sender's rate should be decreased
- An ACK indicates that network is delivering the sender's segment to the receiver
  - TCP sender's rate can be increased
- Bandwidth probing
  - TCP sender increases transmission rate to probe when congestion onset begins
  - backs off from that rate and then begins probing to see if congestion rate has changed
- Jacobson 1988
  - Slow start; congestion avoidance; fast recovery

# TCP Slow Start

- When connection begins, increase rate exponentially until first loss event:
  - double **`CongWin`** every RTT
  - done by incrementing **`CongWin`** for every ACK received
- <u>Summary:</u> initial rate is slow but ramps up exponentially fast
- When this exponential growth rate should end?



# TCP Slow Start (more)

- If there is  a loss event (i.e., congestion) indicated by a timeout
  - TCP sender sets the value of `cwnd` to 1
  - Begin the slow start process anew
  - Sets the value of 2nd state variable `ssthresh` (slow start threshold) to `cwnd/2`
  - Slow start ends when  `cwnd = ssthresh`
- TCP transitions into congestion avoidance (CA) mode
  - TCP increases `cwnd`  more cautiously when in CA mode
  - Final end of slow start happens if 3 duplicate ACKs are detected
  - TCP performs a fast retransmit
  - Enters a fast recovery state

# TCP Slow Start (more)

- Congestion avoidance state
  - value of `cwnd` is approx. half its value when congestion was last detected
  - Rather than doubling the value of `cwnd` every RTT TCP adopts a more conservative approach
  - Increase the value of `cwnd` by just a single MSS

- TCP performs a fast retransmit
  - Enters a fast recovery state

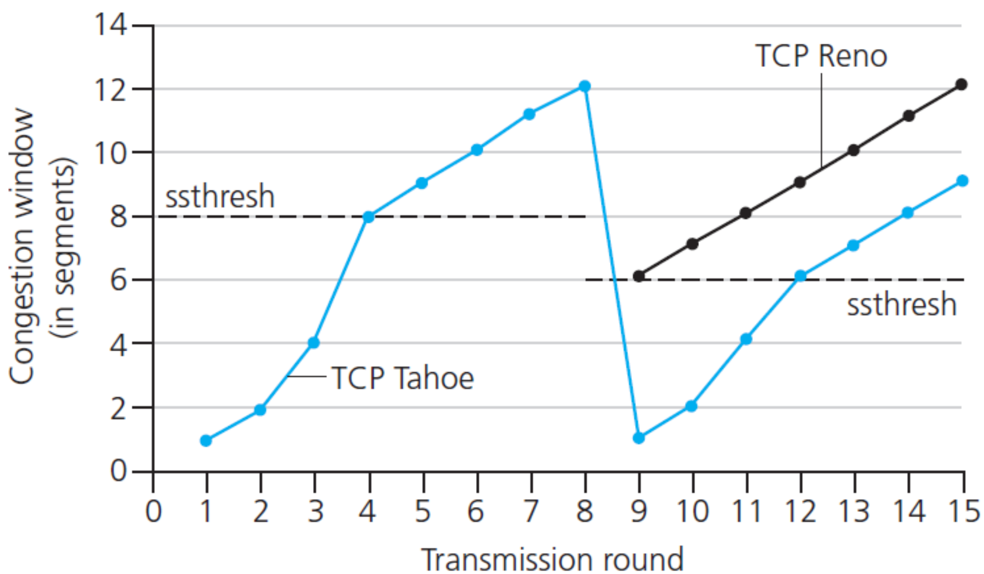
# TCP: detecting, reacting to loss

- loss indicated by timeout:
  - **cwnd** set to 1 MSS
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO (newer version)
  - dup ACKs indicate network capable of delivering some segments
  - **cwnd** is cut in half window then grows linearly
- TCP Tahoe (earlier version) always sets **cwnd** to 1 (timeout or 3 duplicate acks)

# TCP: switching from slow start to CA

Q: When should the exponential increase switch to linear?

A: When **cwnd** gets to 1/2 of its value before timeout.

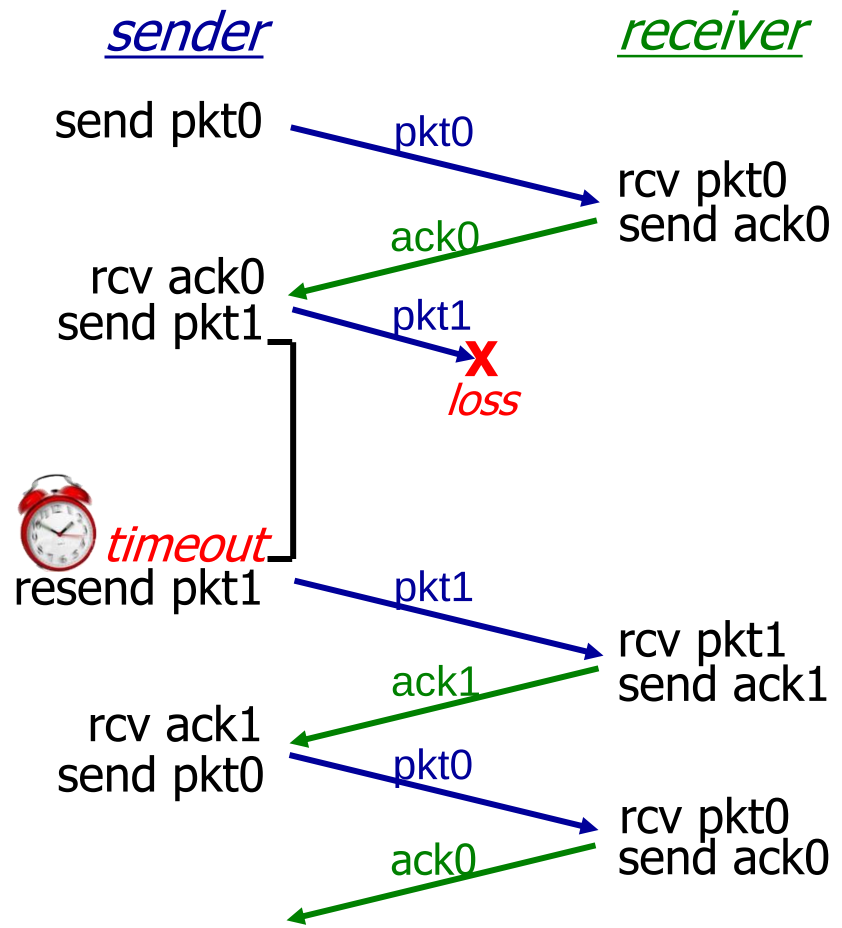



<u>Implementation:</u>

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

# Next Step

- We talked about the congestion window
  - Setting up the congestion window size
- **What about RTT and Retransmission Timeout?**
  - How to determine the value of RTT/RTO?




(b) packet loss

# Timeout -- function of RTT

**Q:** how to set TCP timeout value?

- longer than RTT
  - but RTT varies
  - How much larger?
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss
- How should the RTT be estimated in first place?
- Should a timer be associated with each and every unacknowledged segment? [TCP work by Jacobson 1988]

**Q:** how to estimate RTT?

- `SampleRTT`: measured time from segment transmission until ACK receipt
- One of the transmitted but currently unacknowledged segment
- Vary due to congestion in the routers and varying load on the end systems

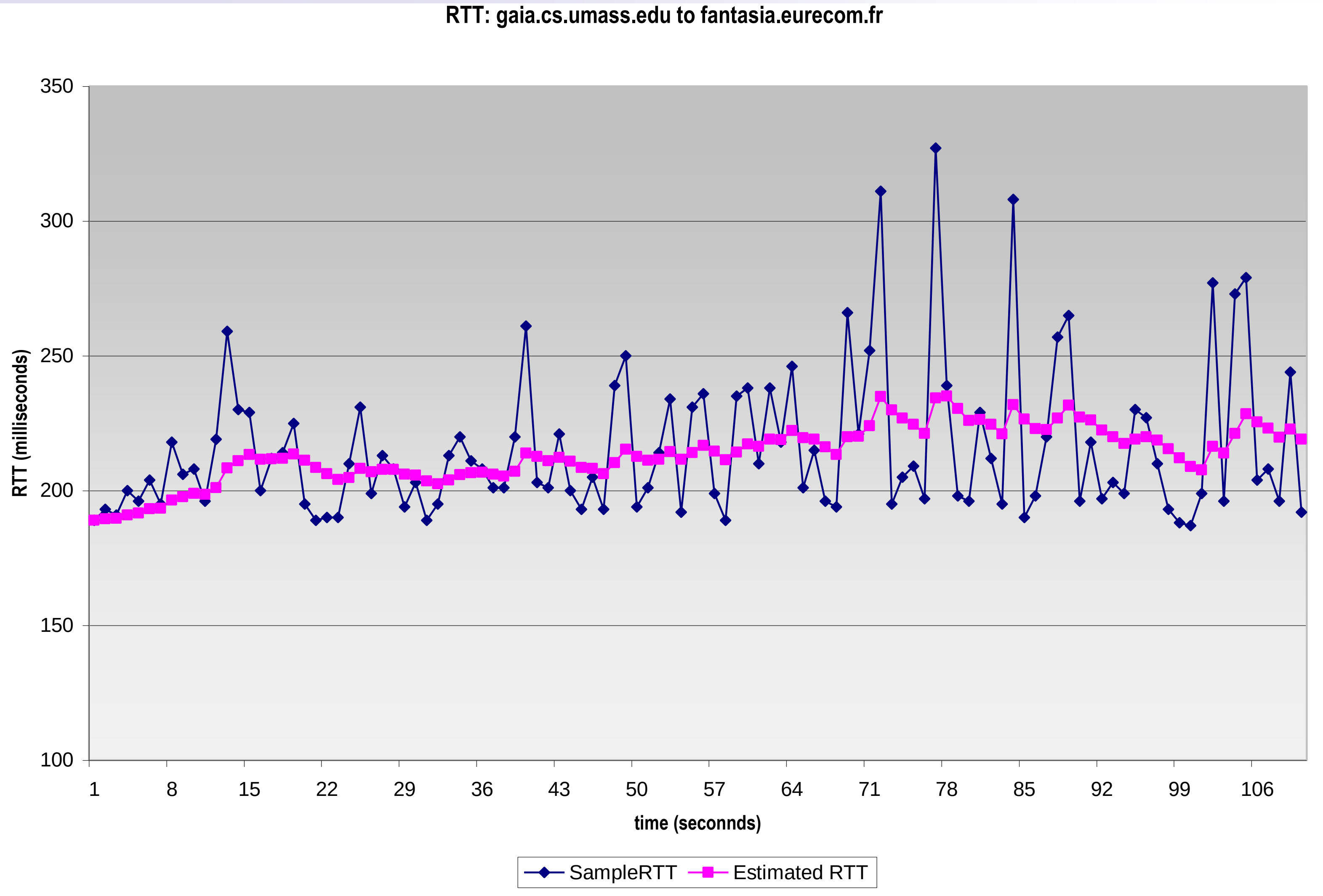
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current **`SampleRTT`**

# TCP Round Trip Time and Timeout

**`EstimatedRTT = (1-` $\alpha$`)*EstimatedRTT +` $\alpha$`*SampleRTT`**

- **Exponential weighted moving average (EWMA)**
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$

# Example RTT estimation:



# TCP Round Trip Time and Timeout

Setting the timeout

- **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT ->`** larger safety margin
- first estimate of how much SampleRTT deviates from EstimatedRTT:

**`DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|`**

**`(typically, β = 0.25)`**

Then set timeout interval:

- Interval should be greater than or equal to **`EstimatedRTT`**, shouldn't be too large
  - Unnecessary retransmissions would be sent or TCP would not quickly retransmit
  - **`EstimatedRTT`** + Margin

**`TimeoutInterval = EstimatedRTT + 4*DevRTT`**

# TCP: Connection-Oriented Transport

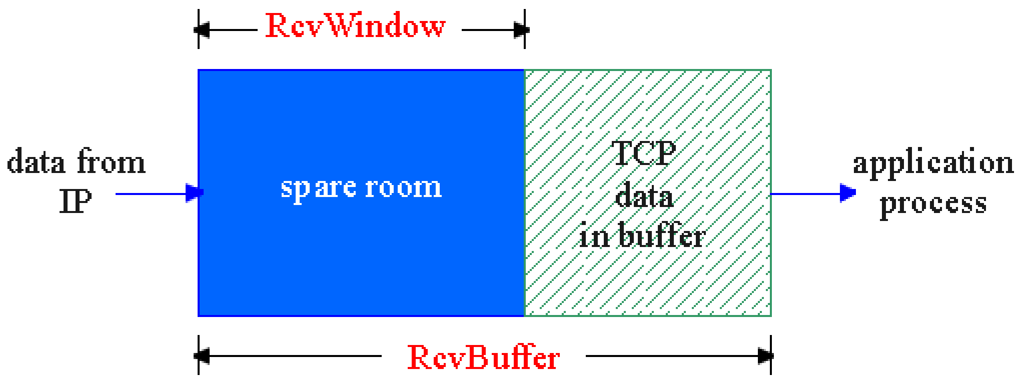
- TCP has 3 main components
  - Reliable transmission
  - Congestion Control
  - Flow Control

# TCP Flow Control

- Problem Definition
  - The receiver has limits on buffer
  - If many nodes transmitting to same receiver
    - Losses may happen at receiver
  - Need to avoid such losses

- Solution
  - Receiver tells transmitter how much space left
  - Transmitter chooses its congestion window accordingly

# TCP Flow Control: how it works




(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer

**= `RcvWindow`**

**= `RcvBuffer-[LastByteRcvd - LastByteRead]`**

- Rcvr advertises spare room by including value of **`RcvWindow`** in segments
- Sender limits unACKed data to **`RcvWindow`**
  - guarantees receive buffer doesn't overflow

# Chapter 3: Transport Layer Focus

- Principles behind transport layer services:
  - Multiplexing, demultiplexing
  - Reliable data transfer
  - Congestion control
  - TCP slow start and CA
  - Flow control
  - TCP RTT and Timeout estimate

# Chapter 4: Network Layer

- 4. 1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What's inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP

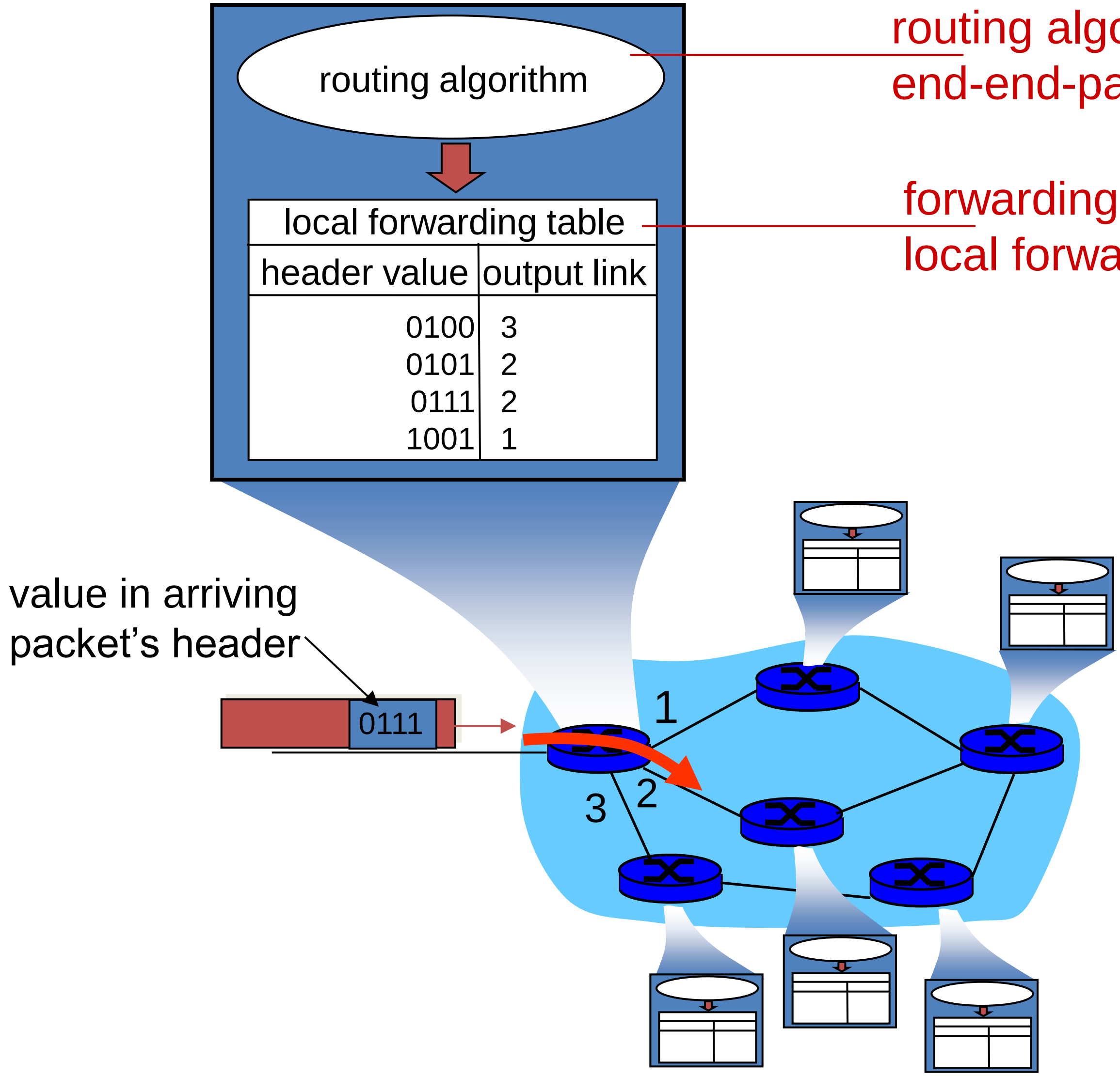
# Key Network-Layer Functions

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

<u>analogy:</u>

- routing: process of planning trip from source to dest
- forwarding: process of getting through actual traffic intersections

# Interplay between routing and forwarding




routing algorithm determines end-end-path through network

forwarding table determines local forwarding at this router

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

# Two types of Network Architecture

- *Connection-Oriented and Connection-Less*

Virtual Circuit Switching

Example: ATM, X.25
Analogy: Telephone

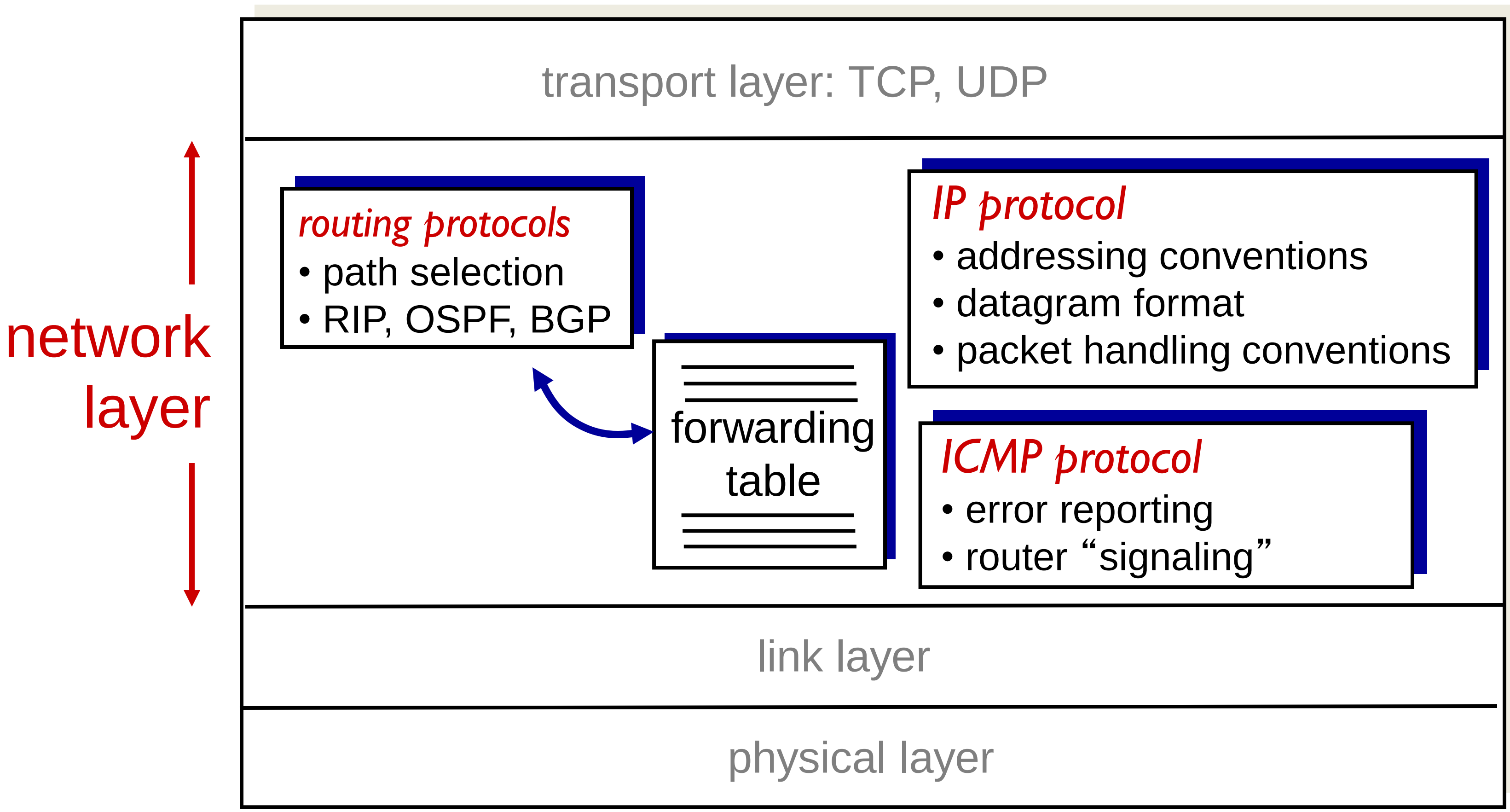
Datagram forwarding

Example: IP networks
Analogy: Postal service

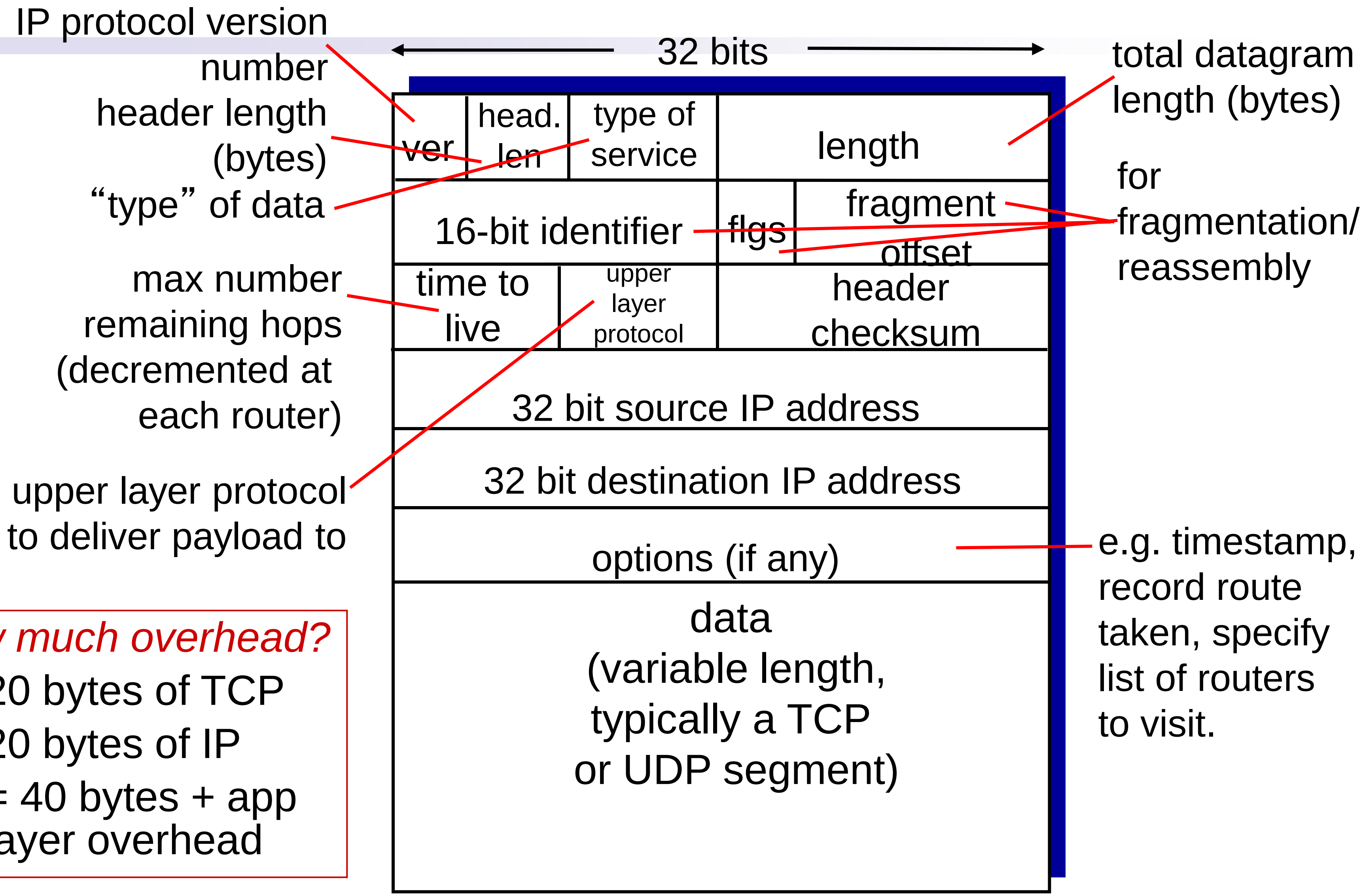
# Chapter 4: outline

# The Internet network layer

host, router network layer functions:



# IP datagram format

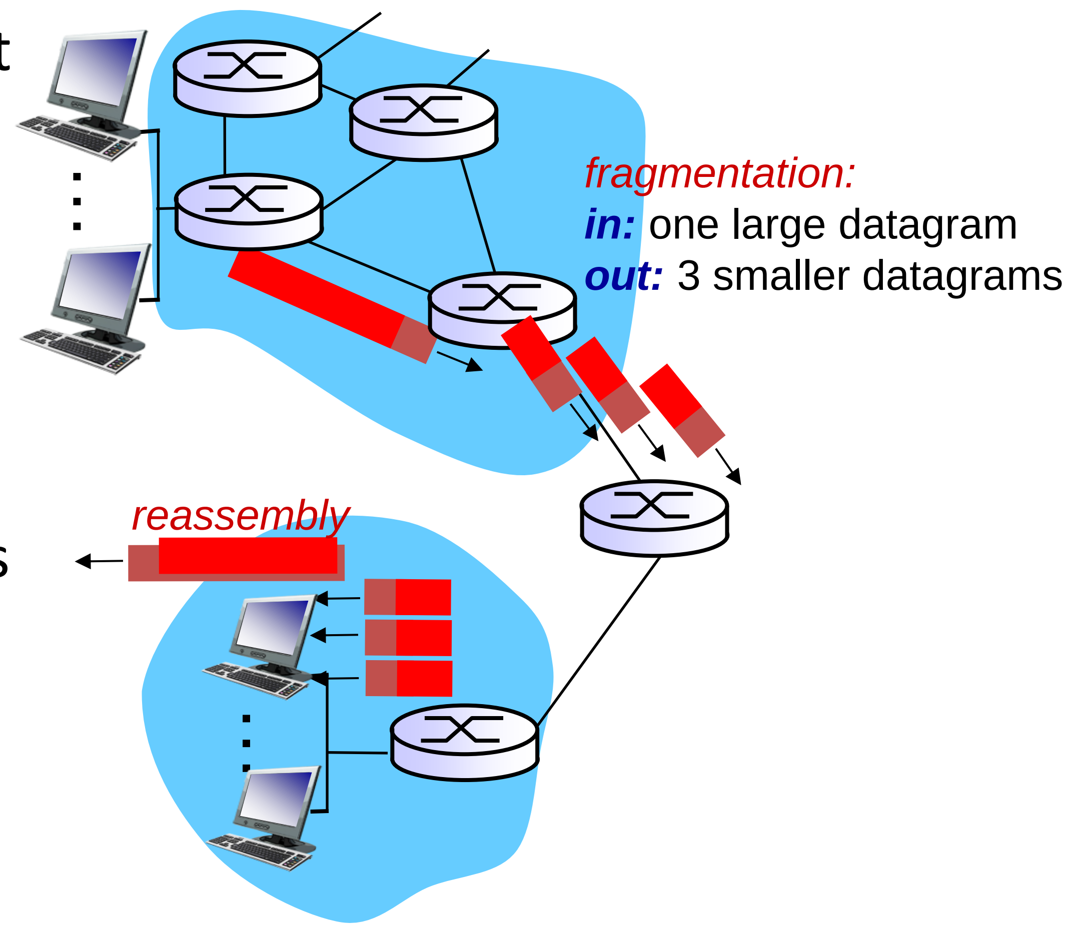



*how much overhead?*

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

# IP fragmentation, reassembly

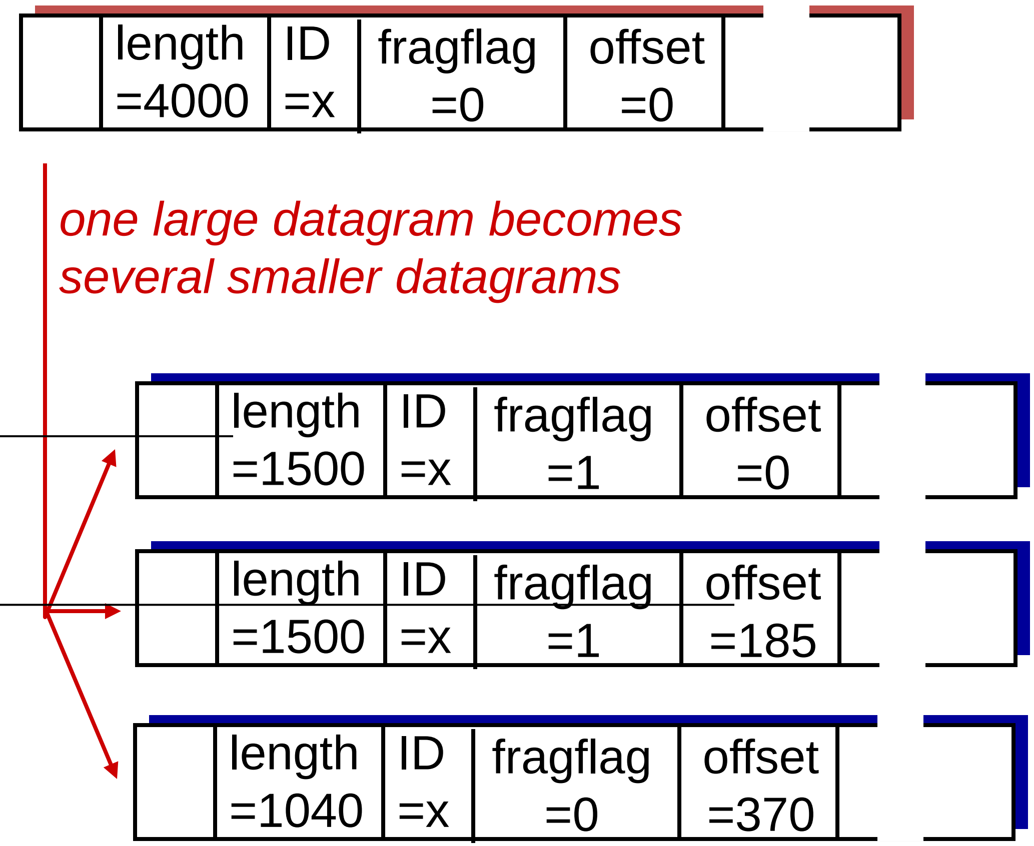
- network links have MTU (max.transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order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



# IP fragmentation, reassembly

*example:*

- 4000 byte datagram
- MTU = 1500 bytes



# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

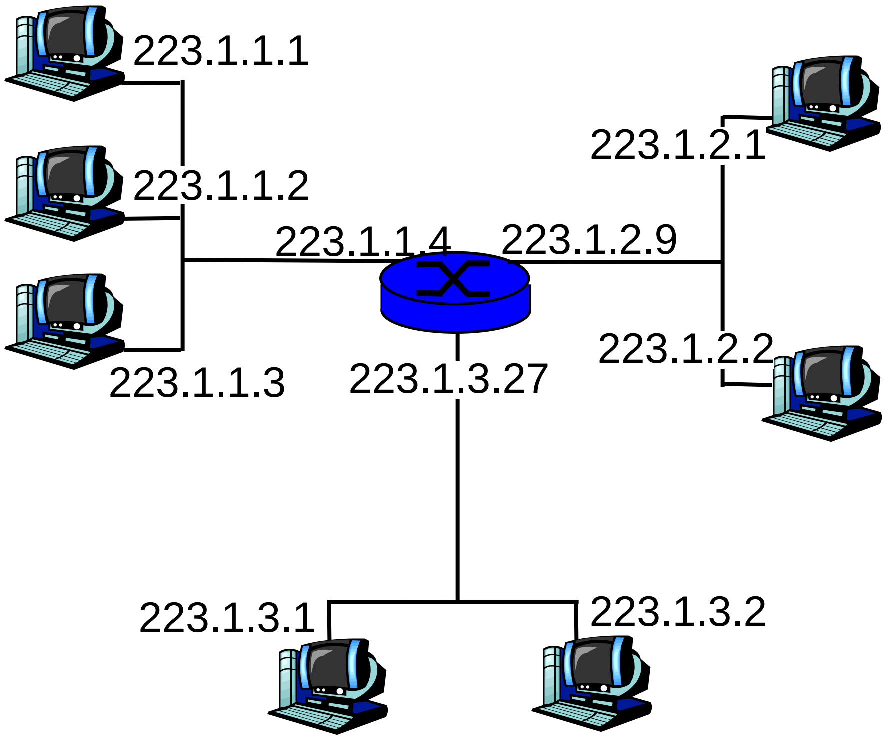
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

# IP Addressing: introduction

- IP address: 32-bit identifier for host, router *interface*
- *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 associated with each interface



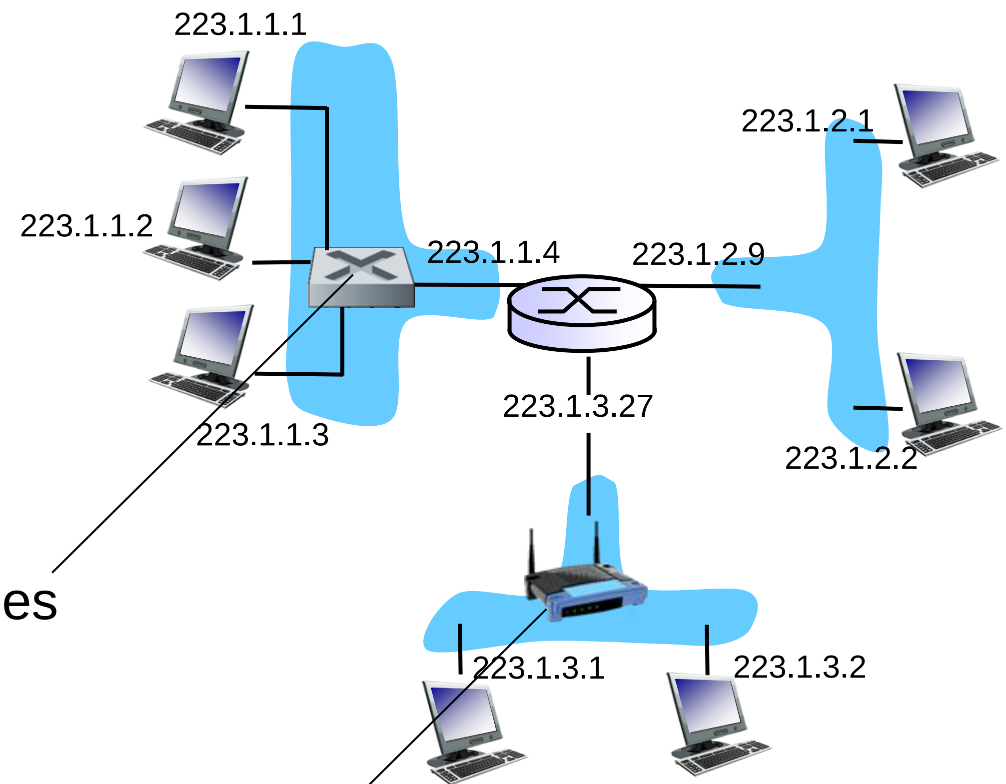

$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

# IP addressing: introduction

*Q: how are interfaces actually connected?*

*A:* wired Ethernet interfaces connected by Ethernet switches

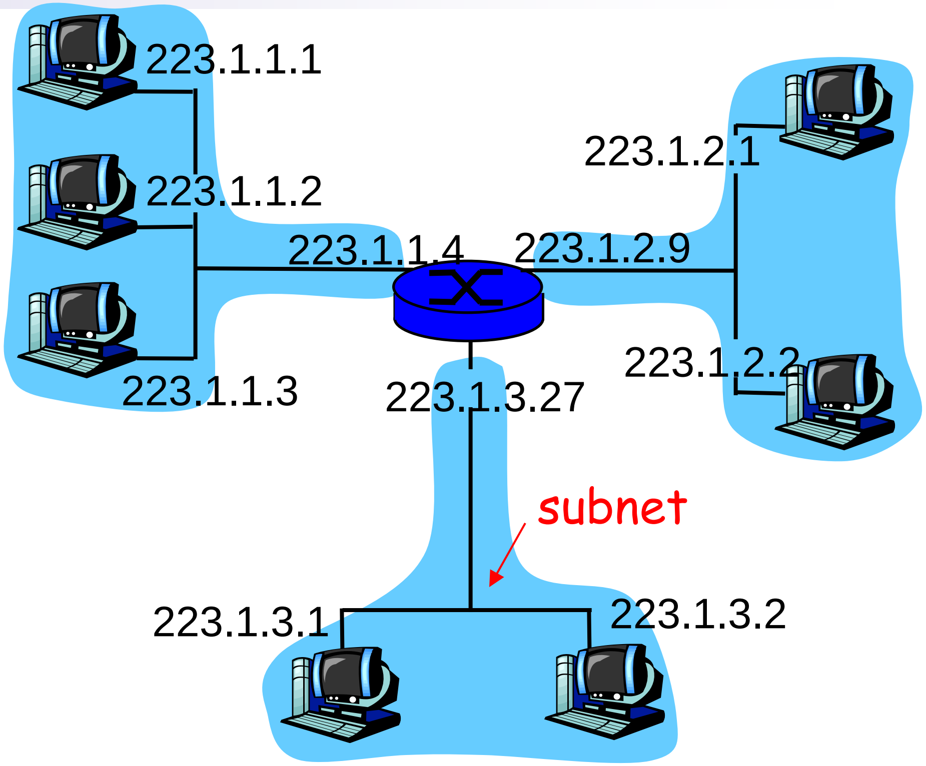
*For now:* don't need to worry about how one interface is connected to another (with no intervening router)




*A:* wireless WiFi interfaces connected by WiFi base station

# Subnets

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

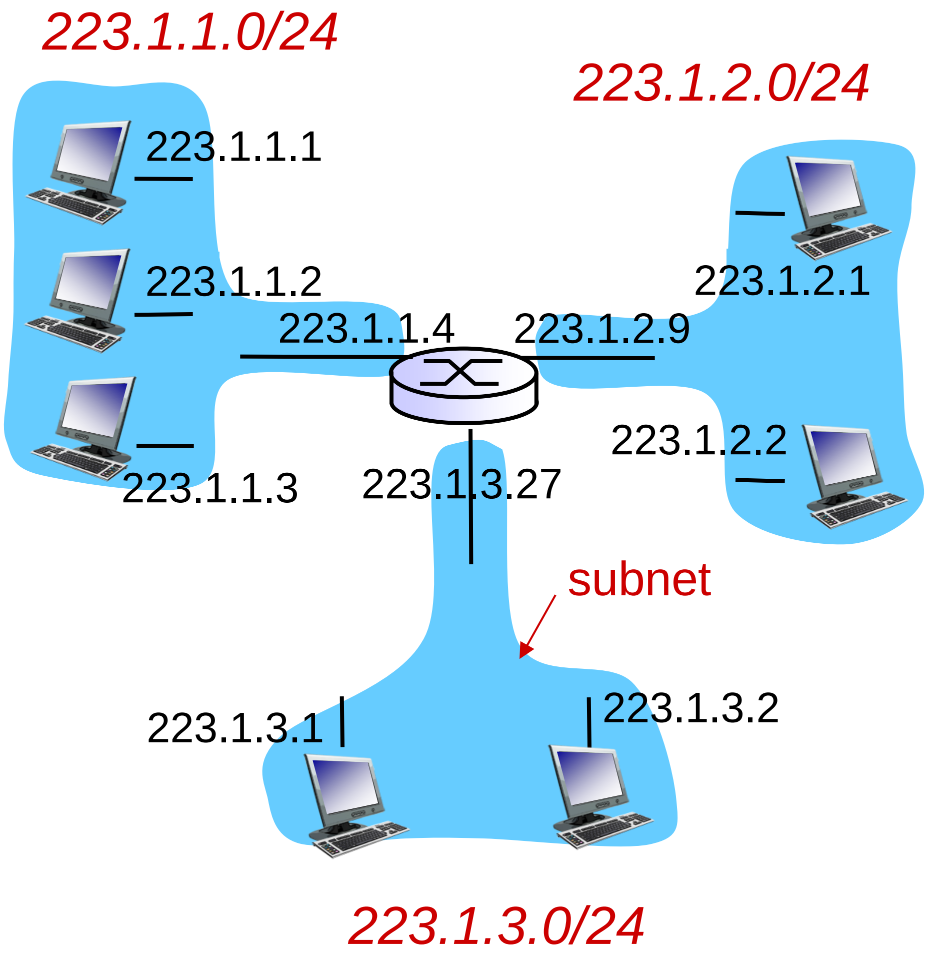



network consisting of 3 subnets

# Subnets

*recipe*

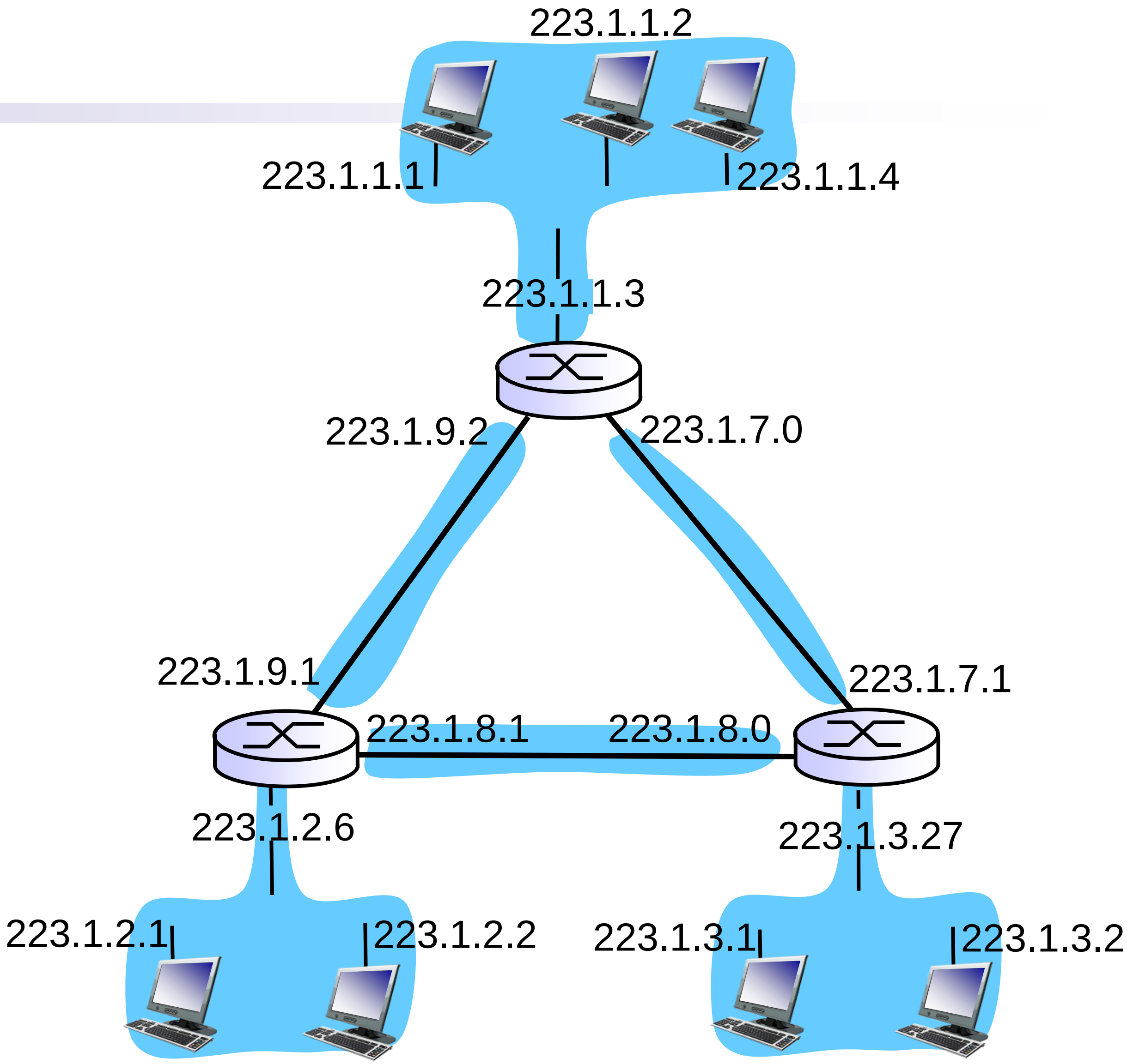
- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*




subnet mask: /24

# Subnets

how many?



# 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

subnet part | host part

11001000 00010111 00010000 00000000

200.23.16.0/23

# 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 as server
  - "plug-and-play"

# DHCP: Dynamic Host Configuration Protocol

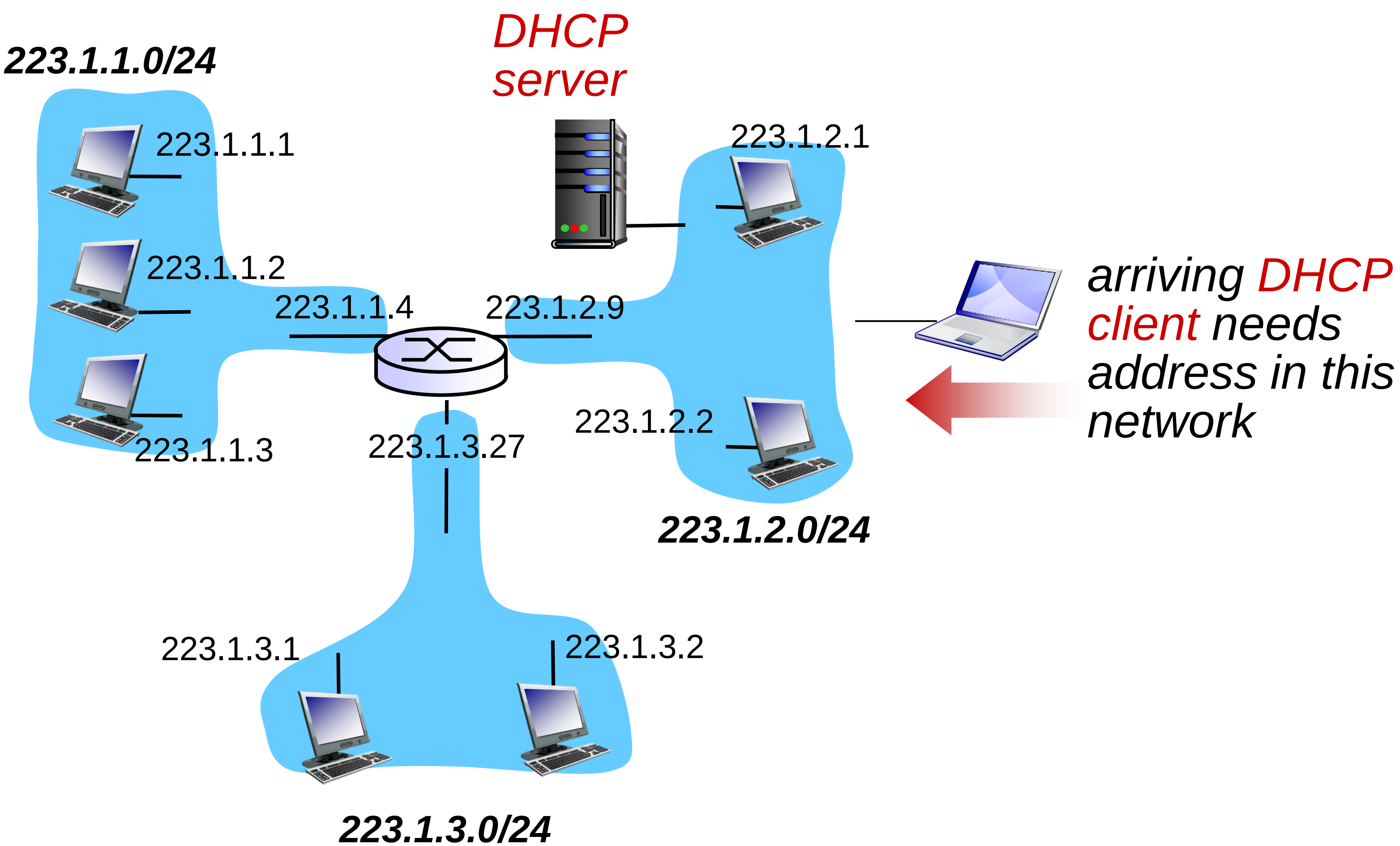
*goal:* allow host to *dynamically* obtain its IP address from network server when it joins network

- 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)

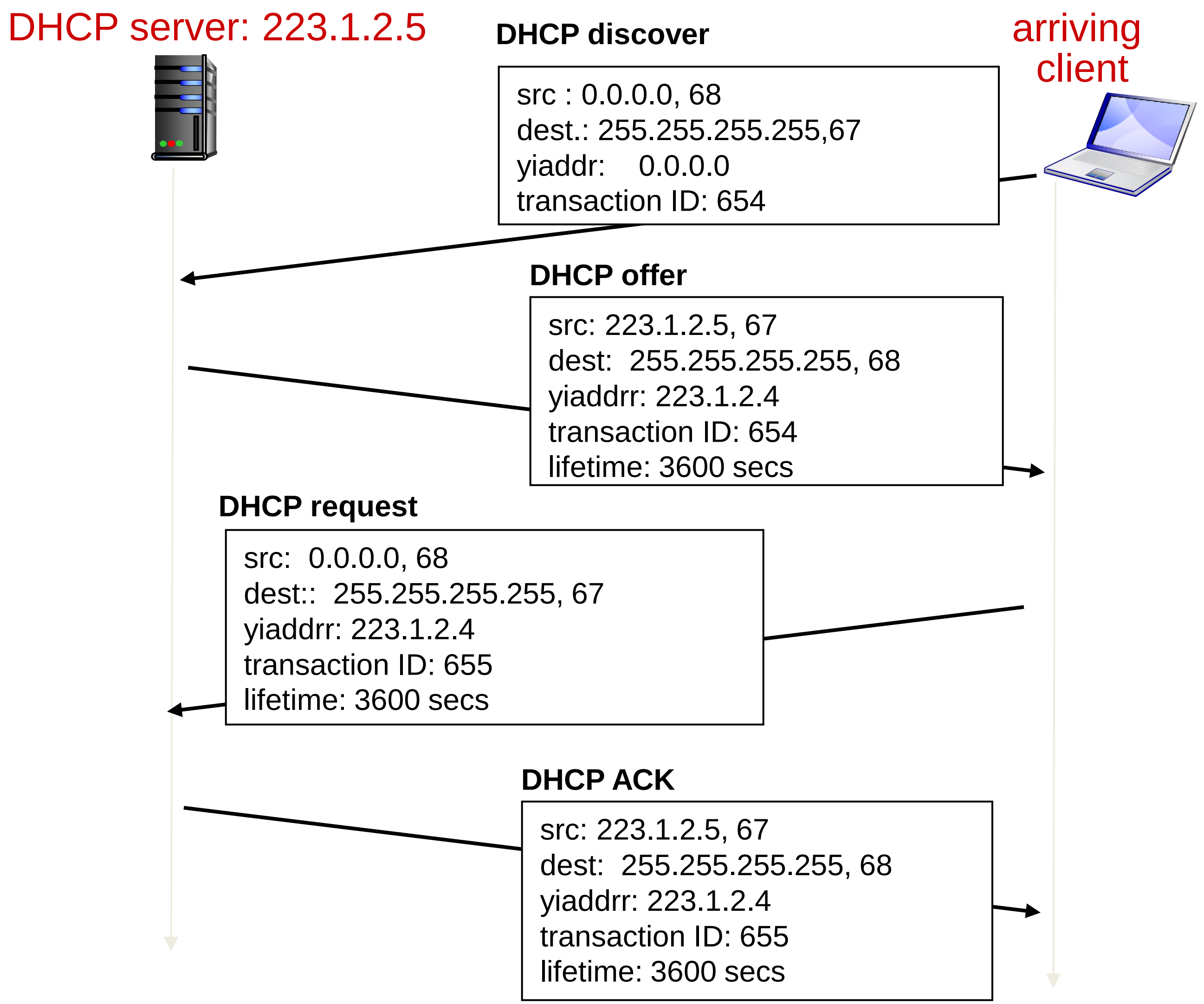
*DHCP overview:*

- 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

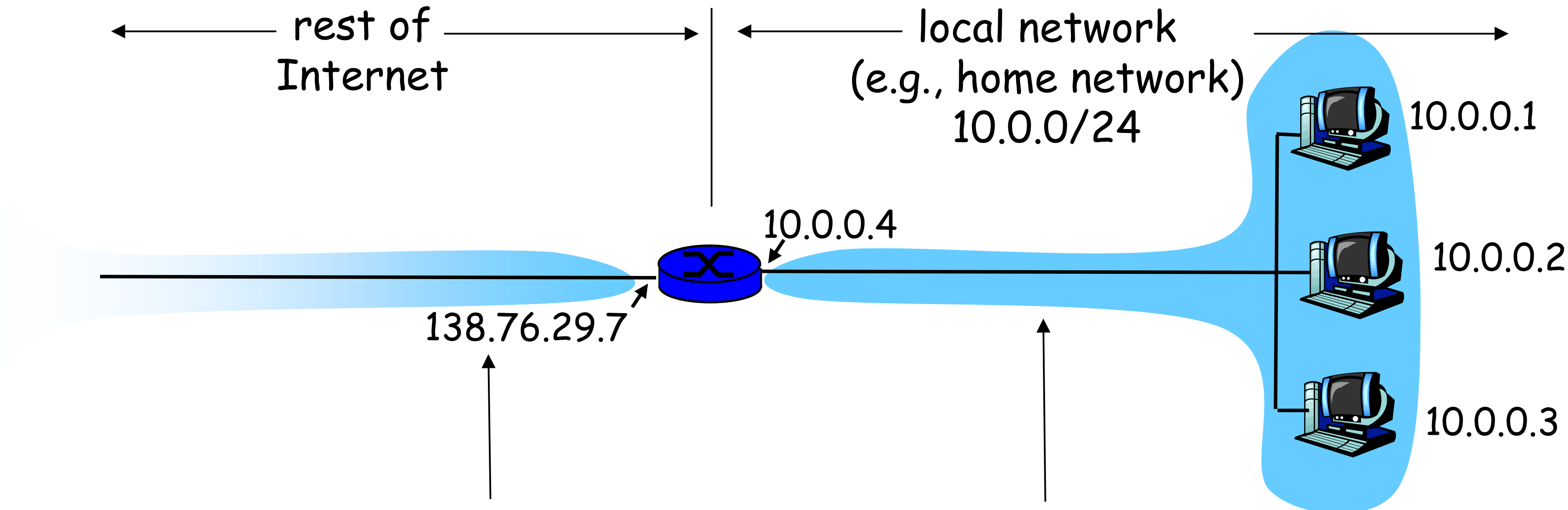


# DHCP client-server scenario



# Network Address Translation

# NAT: Network Address Translation



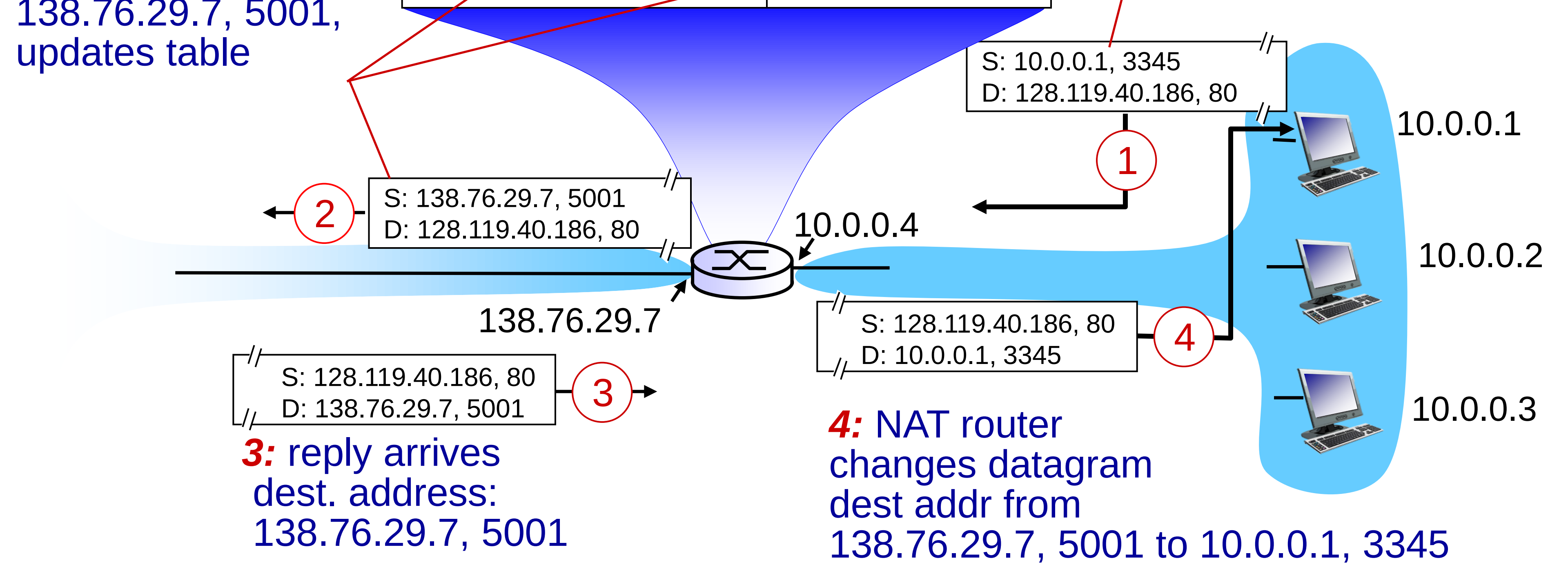

*All* datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7, different source port numbers

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

# NAT: network address translation

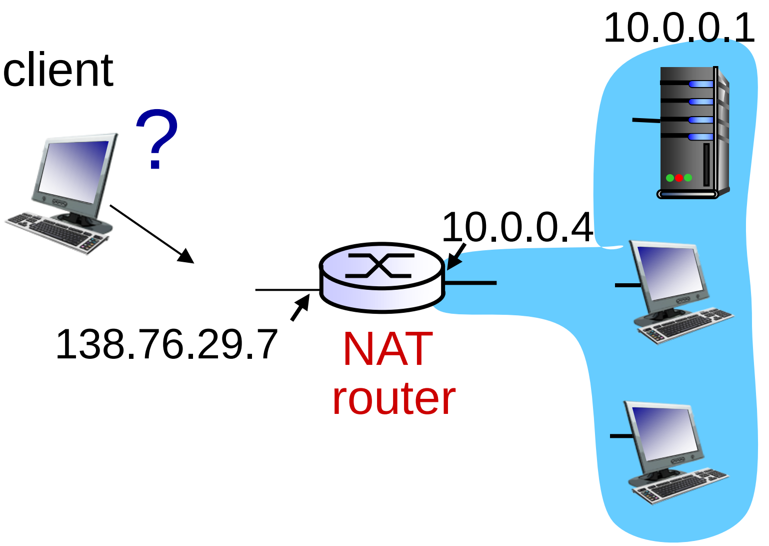
NAT translation table

| WAN side addr | LAN side addr |
|---|---|
| 138.76.29.7, 5001 | 10.0.0.1, 3345 |
| …… | …… |



# NAT traversal problem

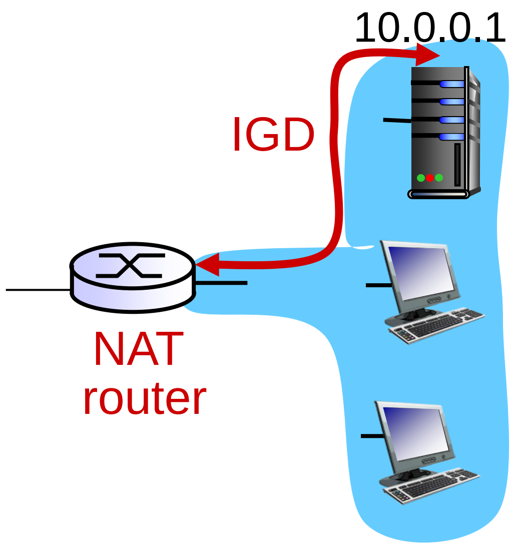
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7
- *solution1:* statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



# NAT traversal problem

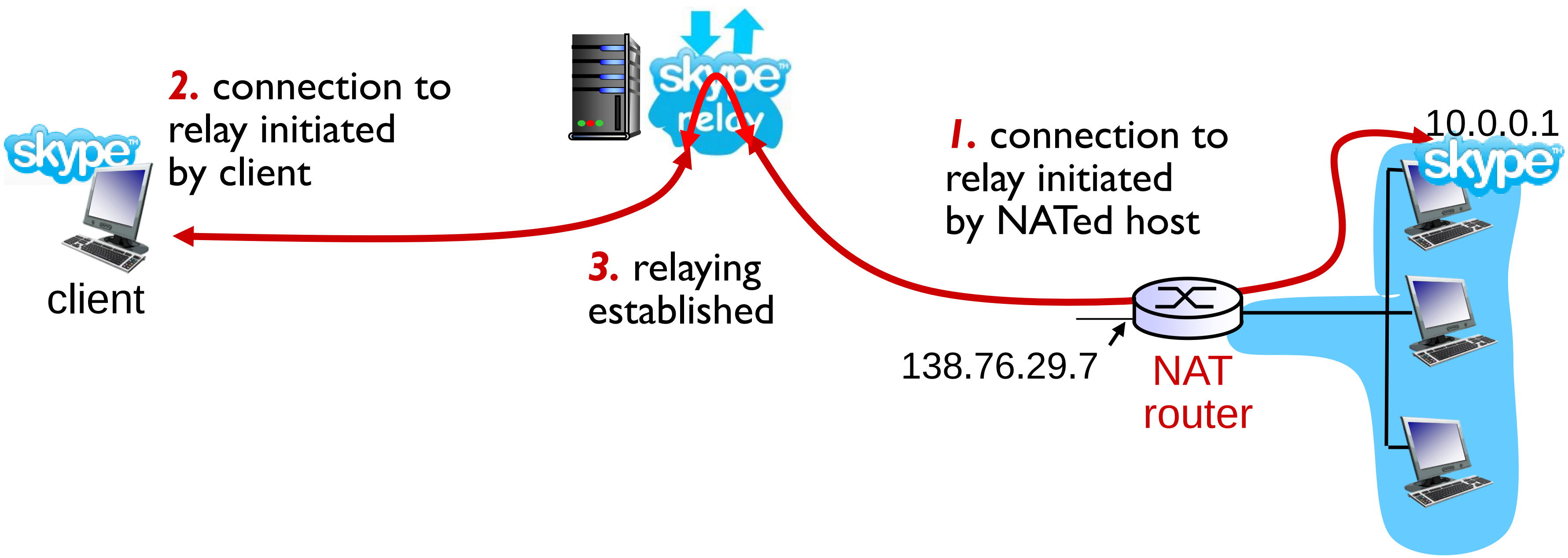
- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - learn public IP address (138.76.29.7)
  - add/remove port mappings (with lease times)

i.e., automate static NAT port map configuration



# NAT traversal problem

- *solution 3:* relaying (used in Skype)
  - NATed client establishes connection to relay
  - external client connects to relay
  - relay bridges packets between to connections



# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what’s inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

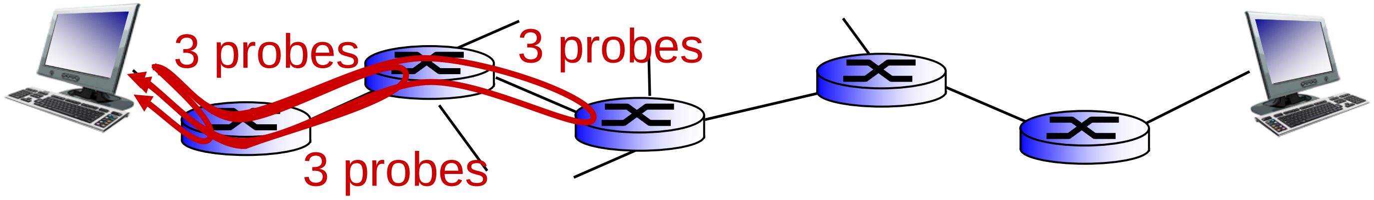
# ICMP: internet control message protocol

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

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

# Traceroute and ICMP

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




- when ICMP messages arrives, source records RTTs

*stopping criteria:*

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

# IPv6: motivation

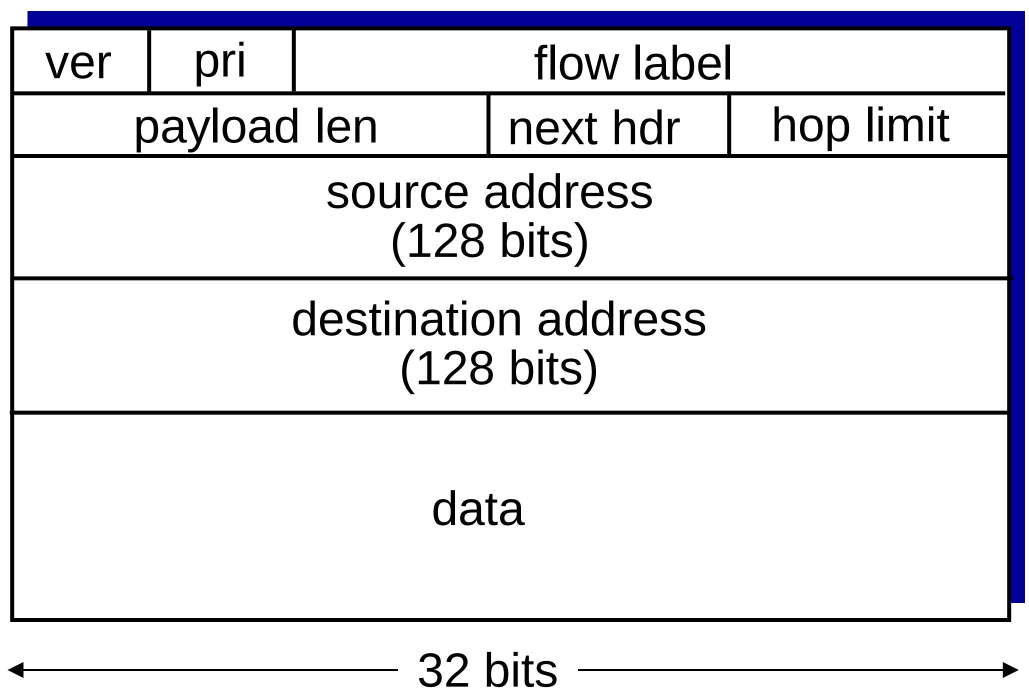
- *initial motivation:* 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

*IPv6 datagram format:*

- fixed-length 40 byte header
- no fragmentation allowed

# IPv6 datagram format

*Priority/traffic class:* identify priority among datagrams in flow
*flow Label:* identify datagrams in same “flow.”
(concept of“flow” not well defined).
*next header:* identify upper layer protocol for data

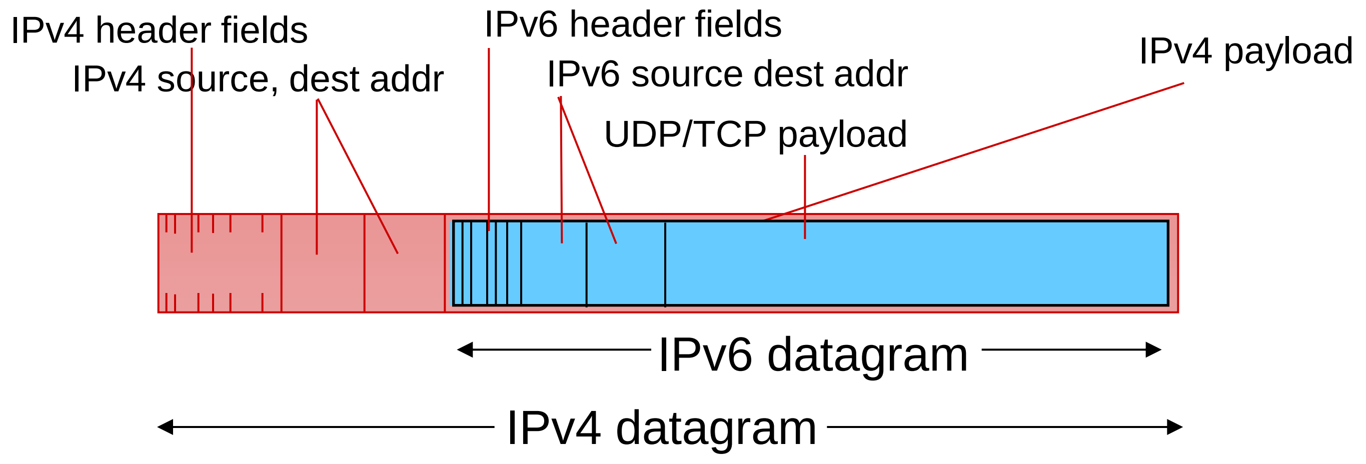


# Other changes from IPv4

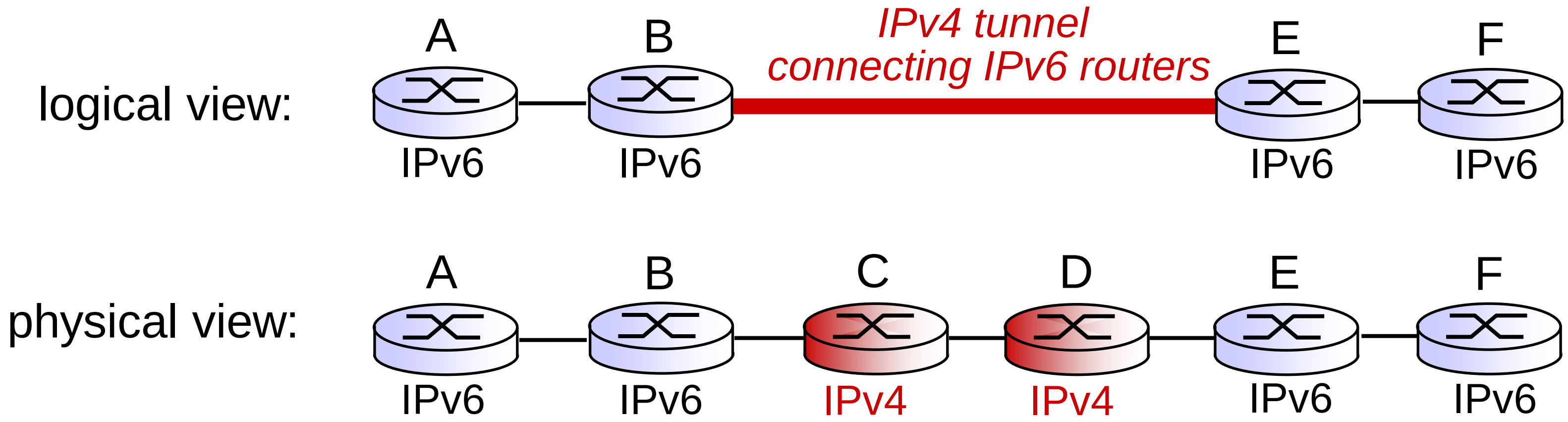
- *checksum*: removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by "Next Header" field
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
  - multicast group management functions

# Transition from IPv4 to 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

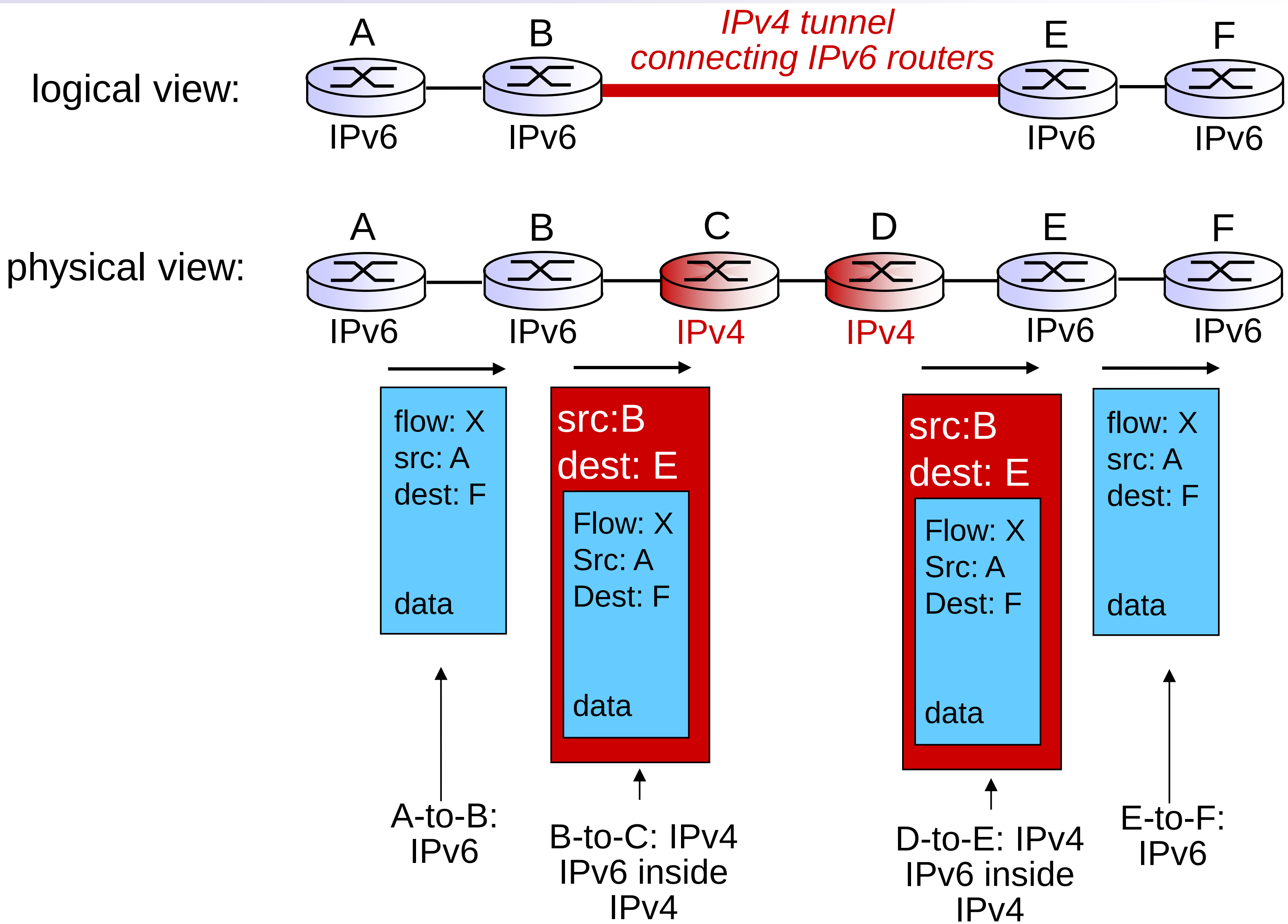


# Tunneling



logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6

# Tunneling



logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6
flow: X
src: A
dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
flow: X
src: A
dest: F
data
A-to-B:
IPv6
B-to-C: IPv4
IPv6 inside
IPv4
D-to-E: IPv4
IPv6 inside
IPv4
E-to-F:
IPv6

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet
- RIP
- OSPF
- BGP

# 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, exchange of info with neighbors
- "distance vector" algorithms

*Q: static or dynamic?*

*static:*

- routes change slowly over time

*dynamic:*

- routes change more quickly
  - periodic update
  - in response to link cost changes

# A Link-State Routing Algorithm

*Dijkstra's algorithm*

- net 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 dest.'s

*notation:*

- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

# Dijsktra's Algorithm

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
      D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Notation:

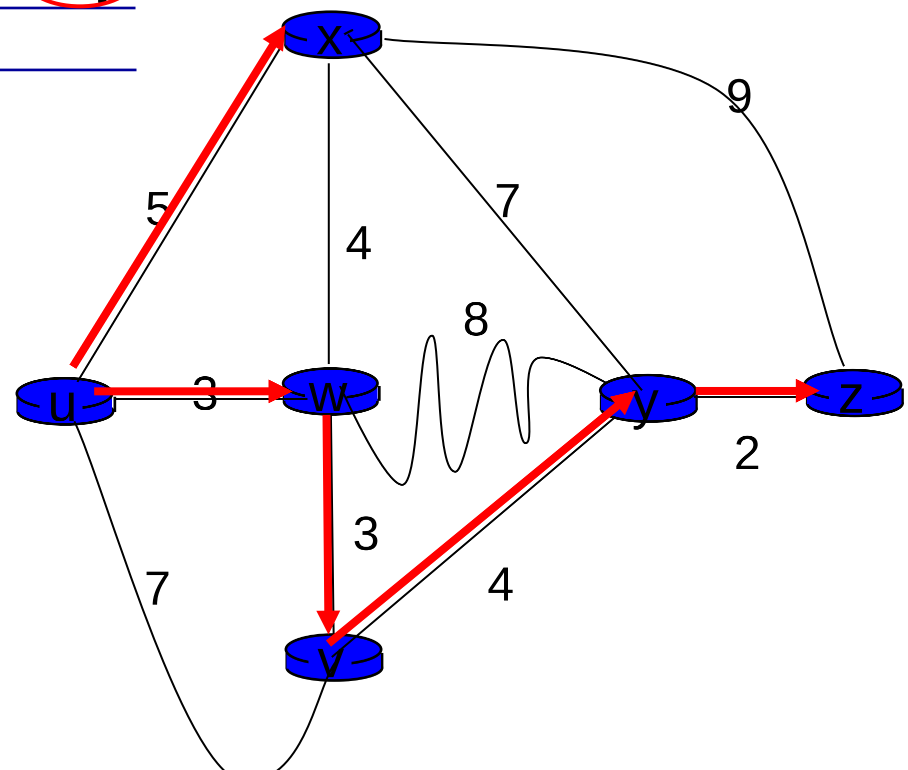
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v

# 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 | uwxvyz | | | | | |

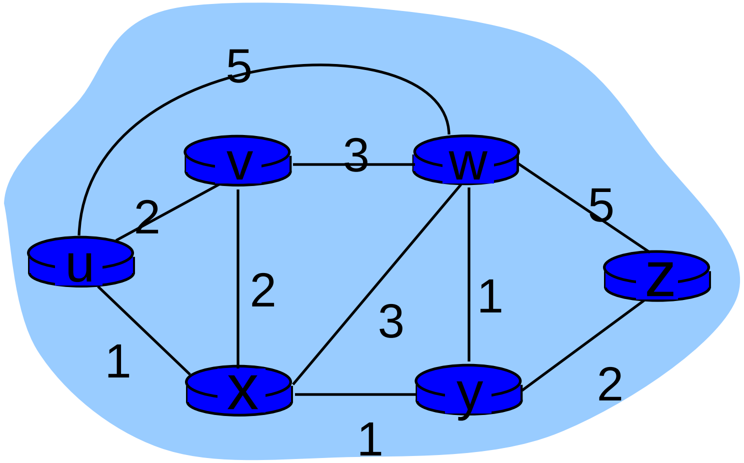
*notes:*

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



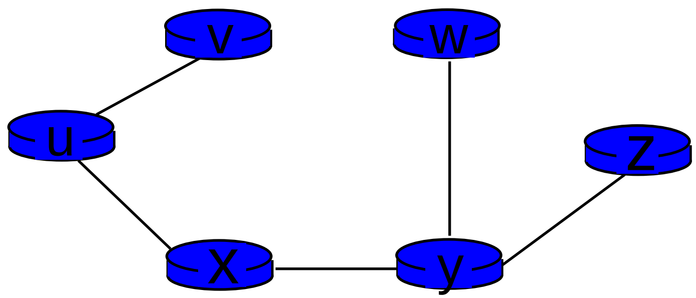
# Dijkstra's algorithm: another example

| 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 | uxyvw | | | | | 4,y |
| 5 | uxyvwz | | | | | |



# Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:




resulting forwarding table in u:

| destination | link |
|---|---|
| v | (u,v) |
| x | (u,x) |
| y | (u,x) |
| w | (u,x) |
| z | (u,x) |

# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

# Distance vector algorithm

*Bellman-Ford equation (dynamic programming)*

let

$d_x(y)$ := cost of least-cost path from x to y

then

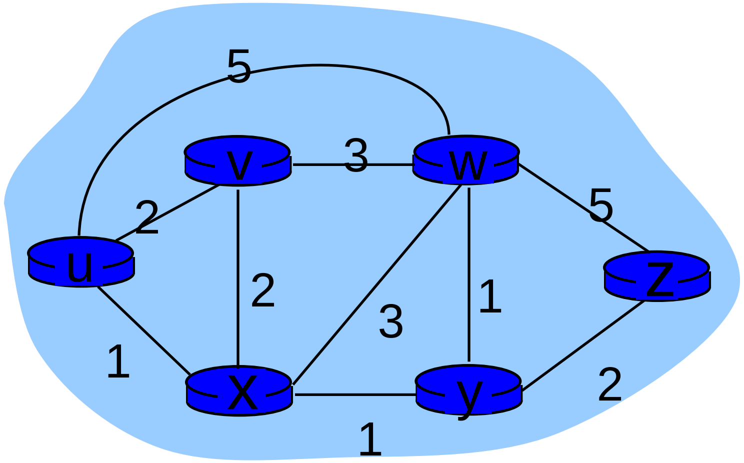
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

*min* taken over all neighbors v of x

# Bellman-Ford example




clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$
\begin{aligned}
d_u(z) &= \min \{c(u,v) + d_v(z), \\
&\qquad c(u,x) + d_x(z), \\
&\qquad c(u,w) + d_w(z)\} \\
&= \min \{2 + 5, \\
&\qquad 1 + 3, \\
&\qquad 5 + 3\} = 4
\end{aligned}
$$

node achieving minimum is next
hop in shortest path, used in forwarding table

$$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\} = \min\{2+0\ ,\ 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\} = \min\{2+1\ ,\ 7+0\} = 3$$

**node x table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

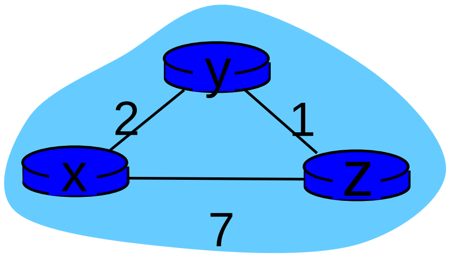
| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

**node y table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

**node z table**

| from \ cost to | x | y | z |
|---|---|---|---|
| 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 \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

**node y table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

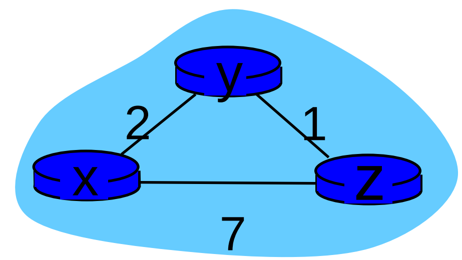
**node z table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | ∞ | ∞ | ∞ |
| z | 7 | 1 | 0 |

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

time



# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

# Hierarchical routing

our routing study thus far - idealization

- all routers identical
- network "flat"

… *not* true in practice

*scale:* with 600 million destinations:

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

*administrative autonomy*

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

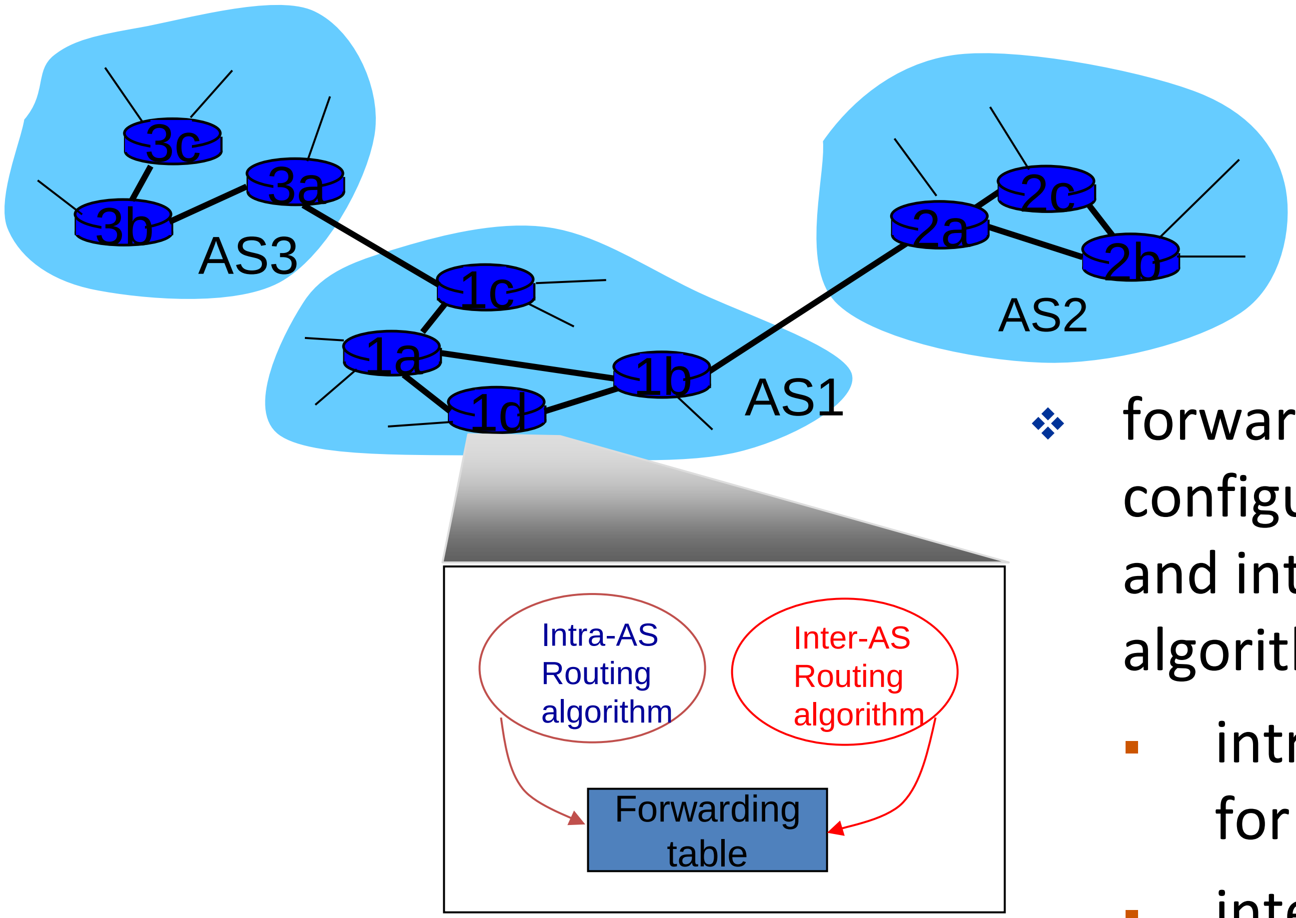
# Hierarchical routing

- aggregate routers into regions, "autonomous systems" (AS)
- routers in same AS run same routing protocol
  - "intra-AS" routing protocol
  - routers in different AS can run different intra-AS routing protocol

*gateway router:*

- at "edge" of its own AS
- has link to router in another AS

# Interconnected ASes

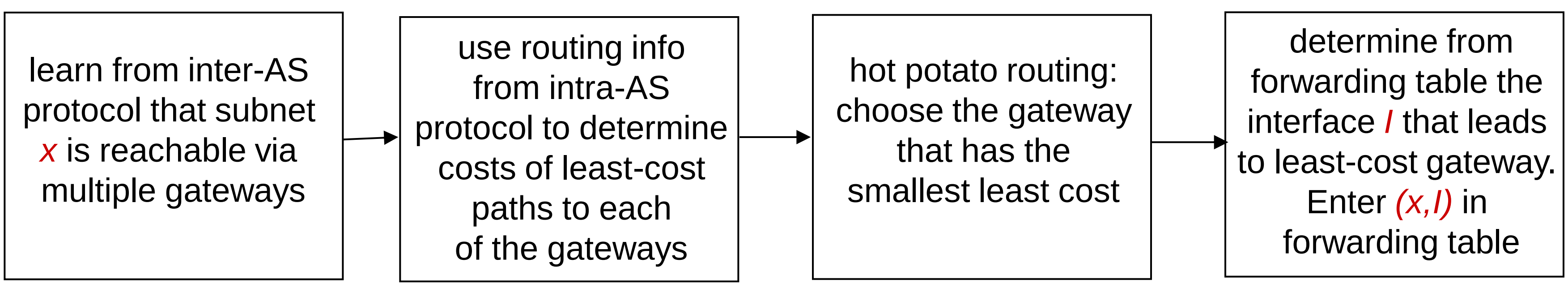



- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS sets entries for internal dests
  - inter-AS & intra-AS sets entries for external dests

# Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest x
  - this is also job of inter-AS routing protocol!

- *hot potato routing:* send packet towards closest of two routers.



# Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

# Intra-AS Routing

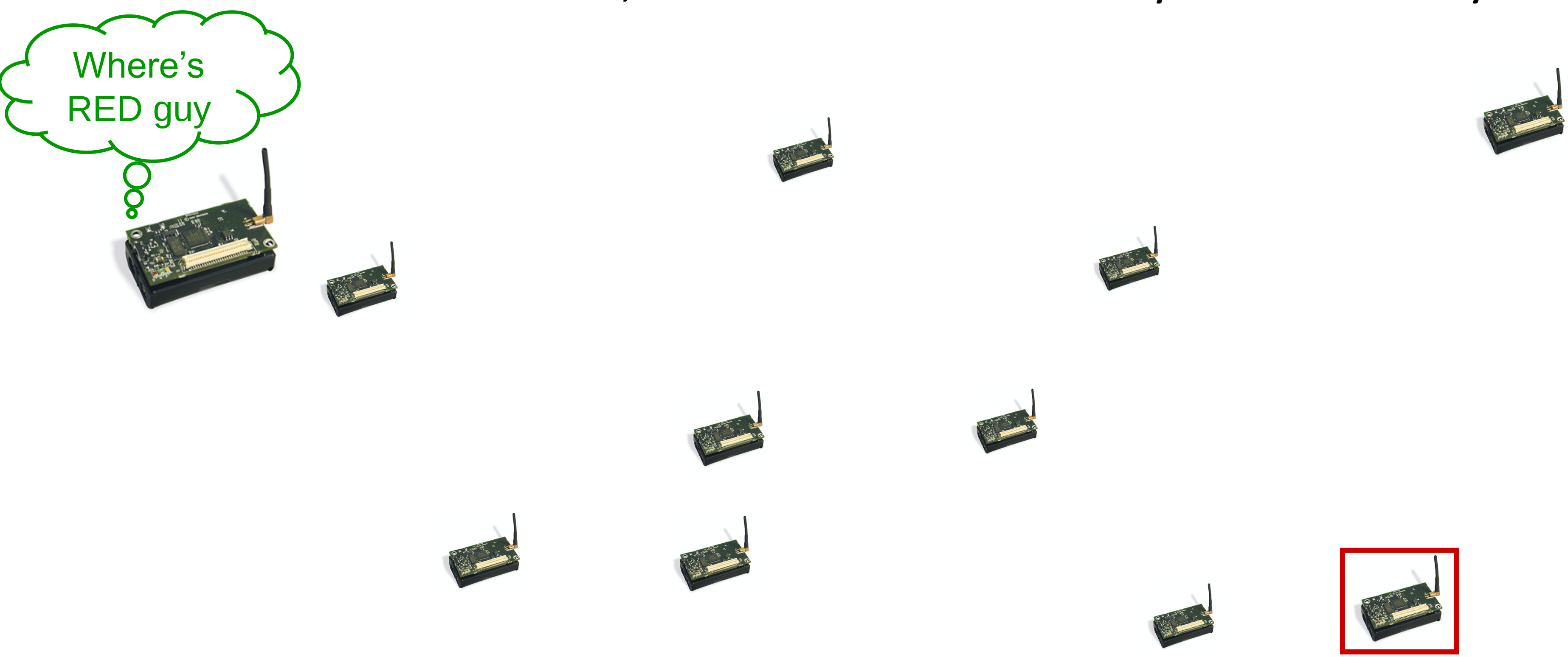
- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary)

# Internet inter-AS routing: BGP

- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASs.
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and policy.
- allows subnet to advertise its existence to rest of Internet: *"I am here"*

# Routing in Wireless Mobile Networks

- Imagine hundreds of hosts moving
  - Routing algorithm needs to cope up with varying wireless channel, error and node mobility and discovery



# Chapter 4: Network Layer Focus

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what’s inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state, distance vector, hierarchical routing

4.6 routing in the Internet

- RIP, OSPF, BGP

Practice the Homework 2 and Quiz 3

# Tentative Final Exam Structure

- Multiple Choice Questions = 18 points
- Chapter 3: (Reliable data transfer, 15+12 = 27 points

TCP congestion control, flow control, & RTT estimation etc.)

- Chapter 3: (TCP slow start, = 15 points
  congestion avoidance etc.)
- Chapter 4: (subnet, routing etc.) 10 + 20 = 30 points
- Chapter 4: General Network Concept = 10 points

100 points

  - When: Tuesday (5/19) 3:30pm – 5:30pm
  - Where: In Class

# Conclusion

Please take a few minutes to complete the online course evaluations.

Thank you for taking IS 450/650. Enjoy your Summer!!

Good Luck!