# Computer Communication Networks

# Network


ICEN/ICSI 416 – Fall 2017

Prof. Dola Saha

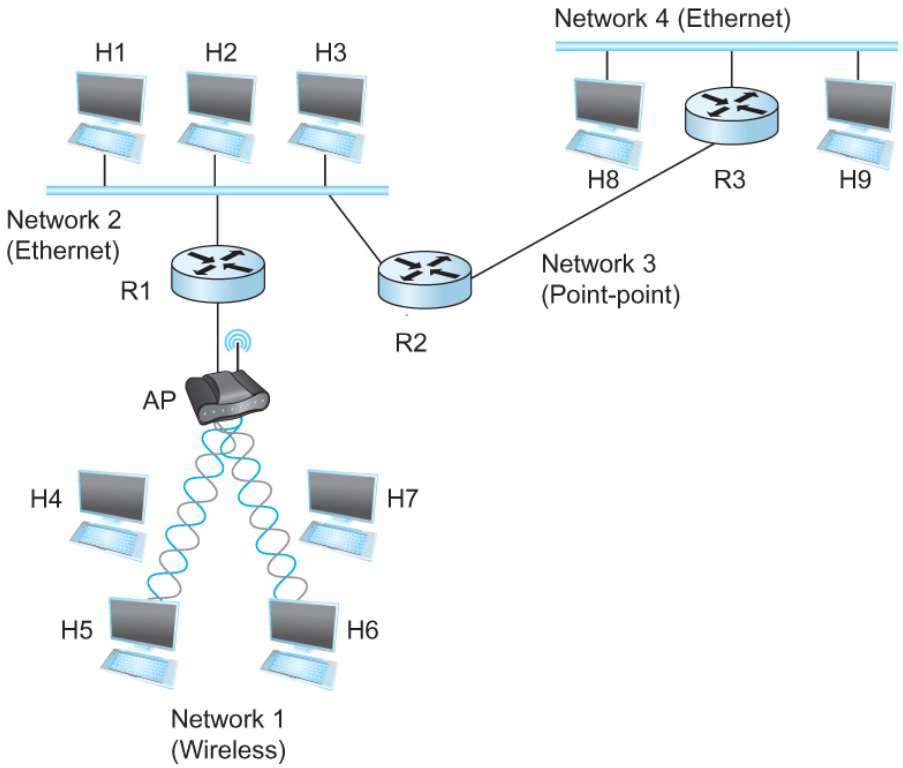
# Network Layer

*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

# Internetworking

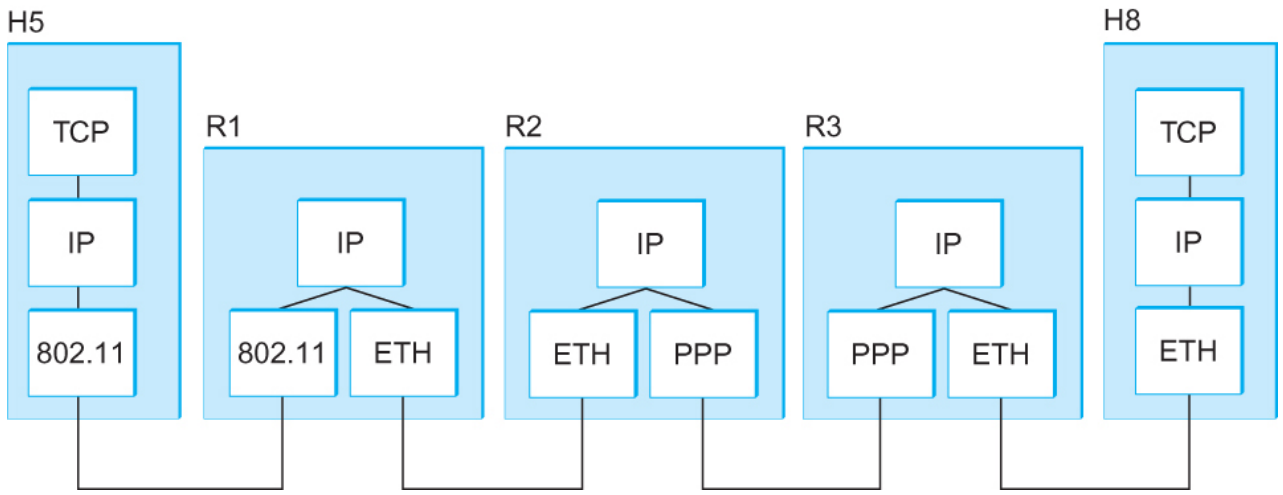
- What is internetwork
  - An arbitrary collection of networks interconnected to provide some sort of host-host to packet delivery service




A simple internetwork where H represents hosts and R represents routers

# Internetworking

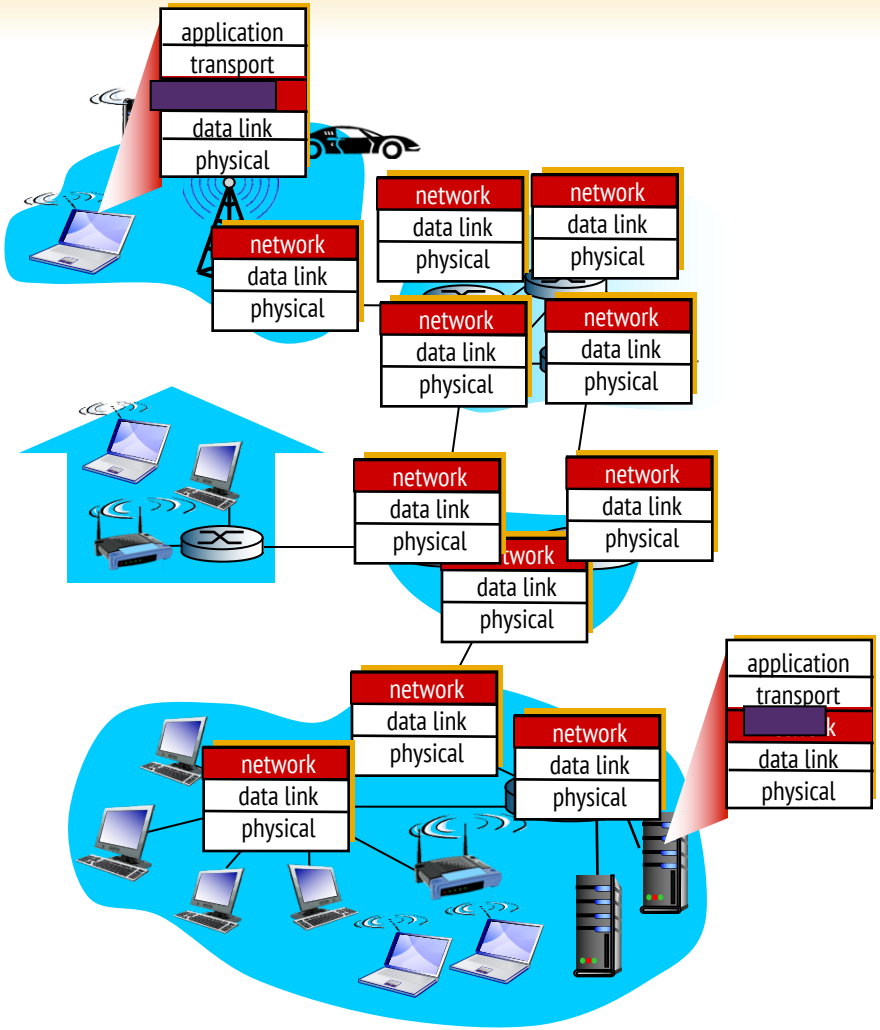
- What is IP
  - IP stands for Internet Protocol
  - Key tool used today to build scalable, heterogeneous internetworks
  - It runs on all the nodes in a collection of networks and defines the infrastructure that allows these nodes and networks to function as a single logical internetwork



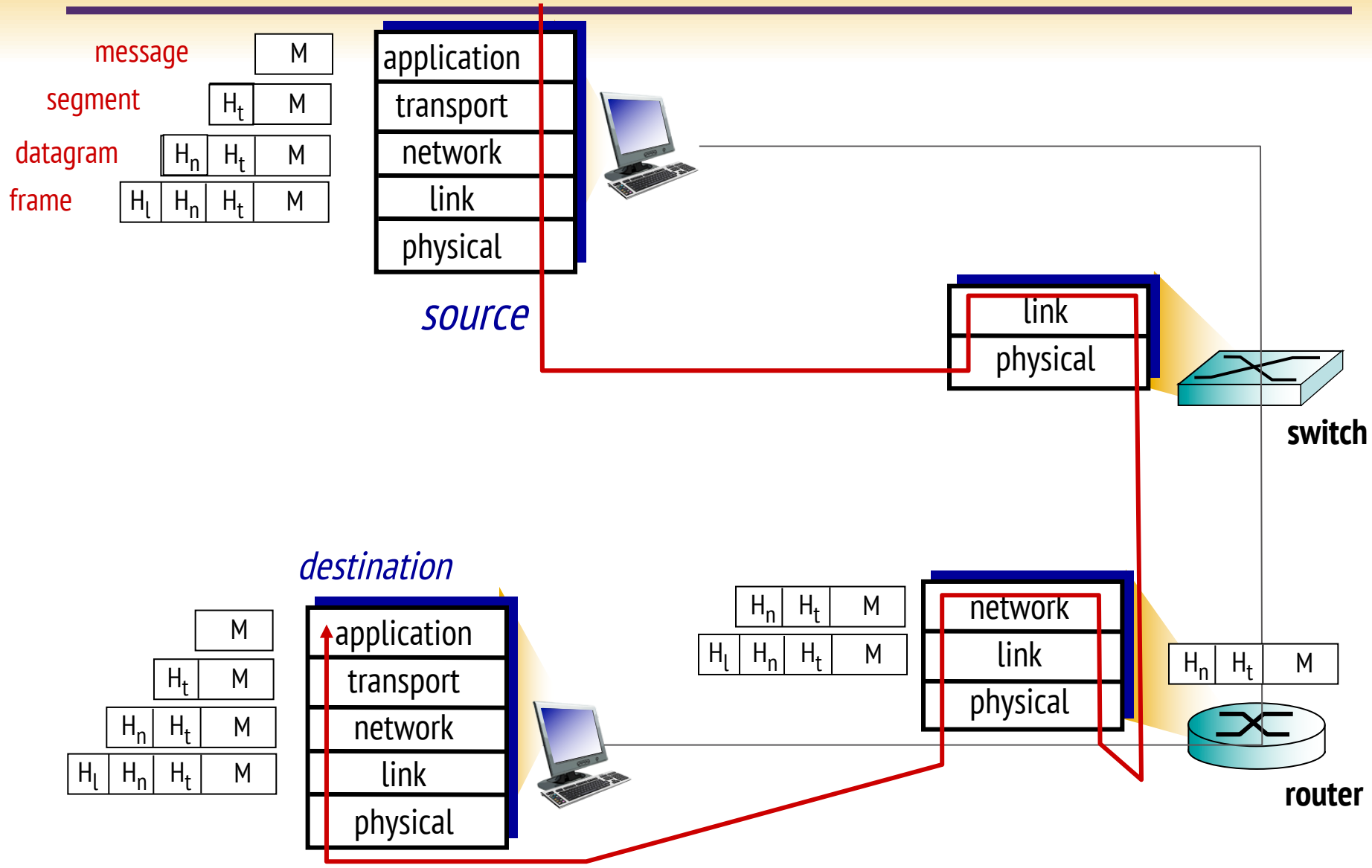

A simple internetwork showing the protocol layers

# Network layer

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

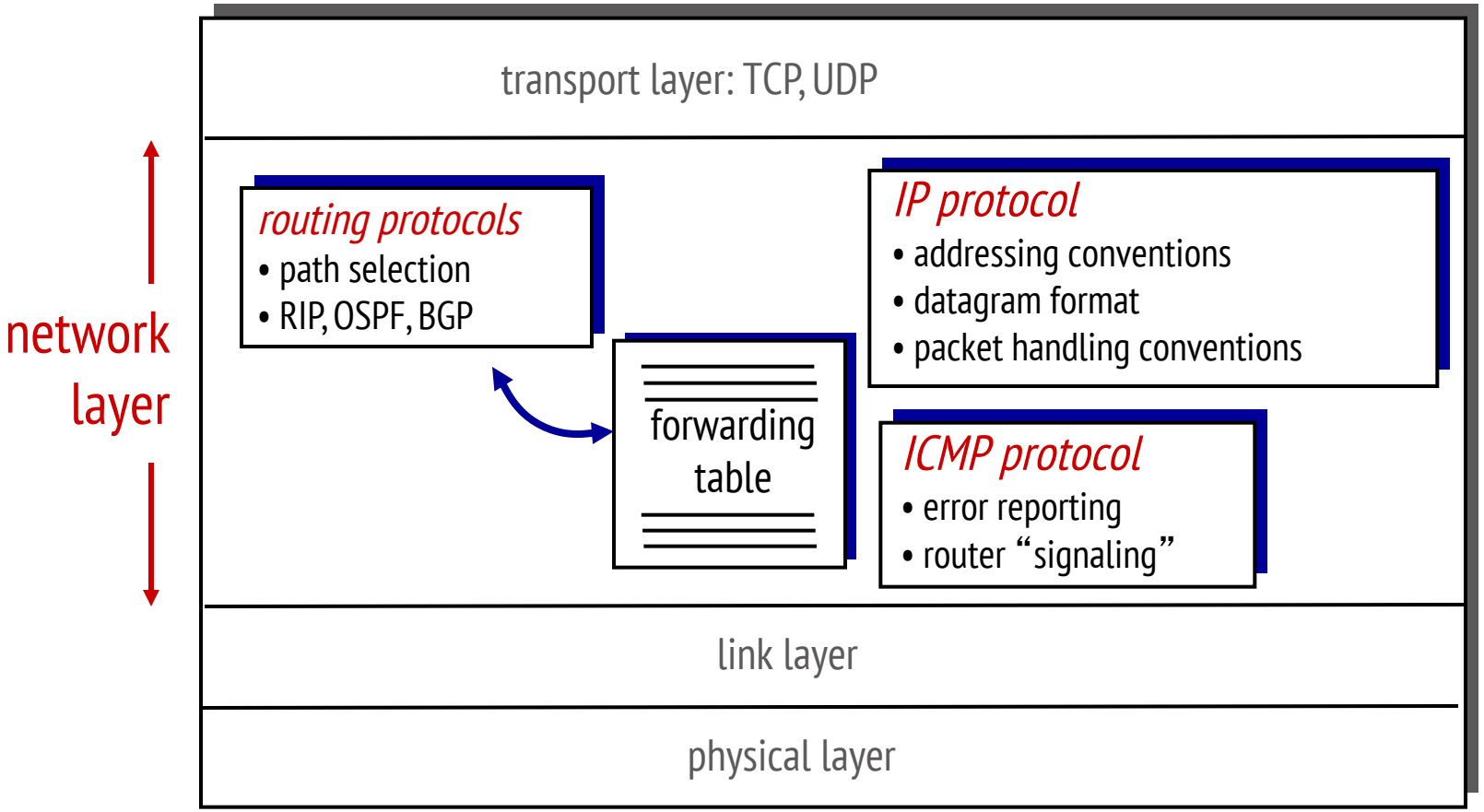


# Encapsulation

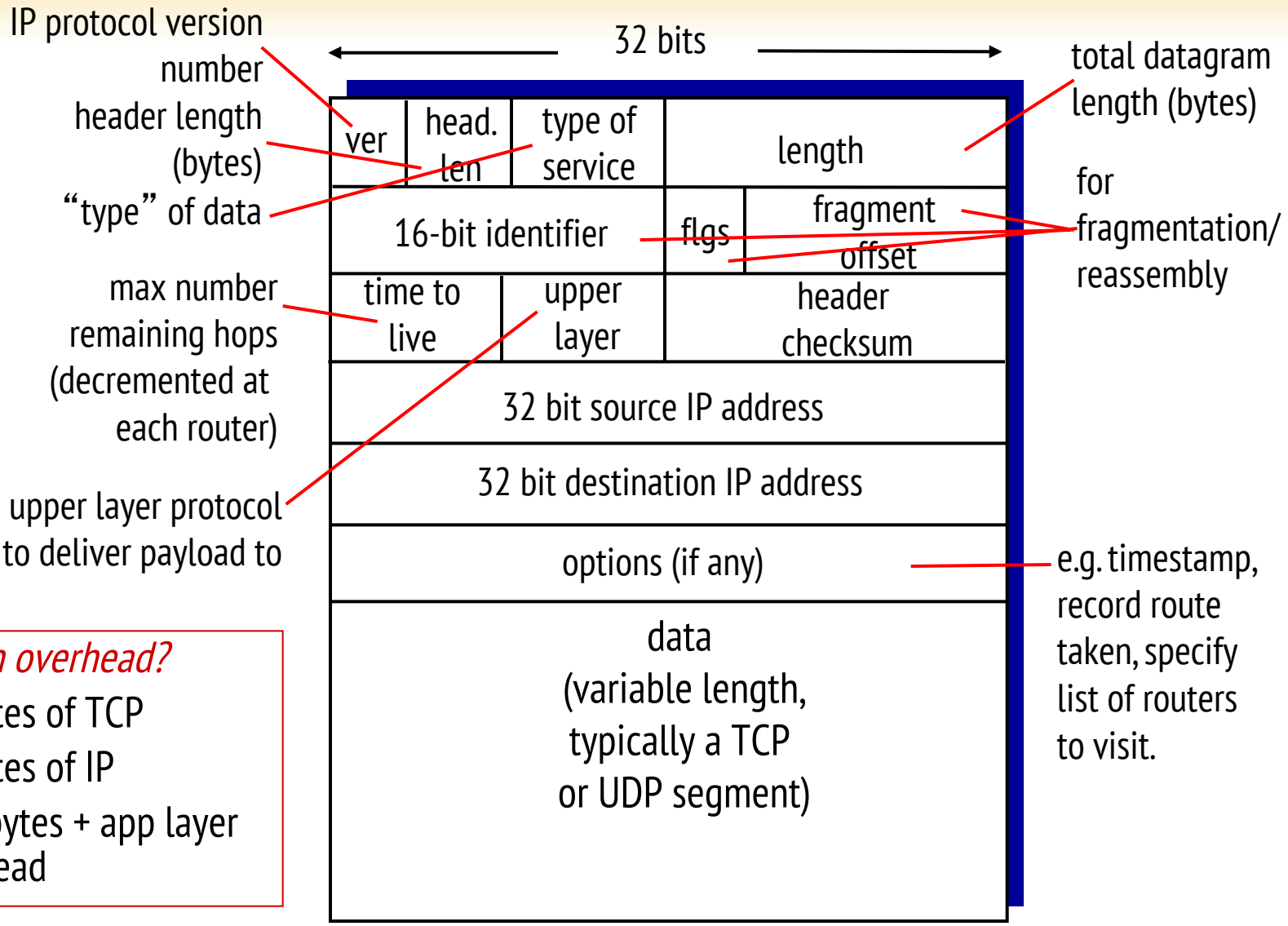


# The Internet network layer

host, router network layer functions:



# IP datagram format




*how much overhead?*

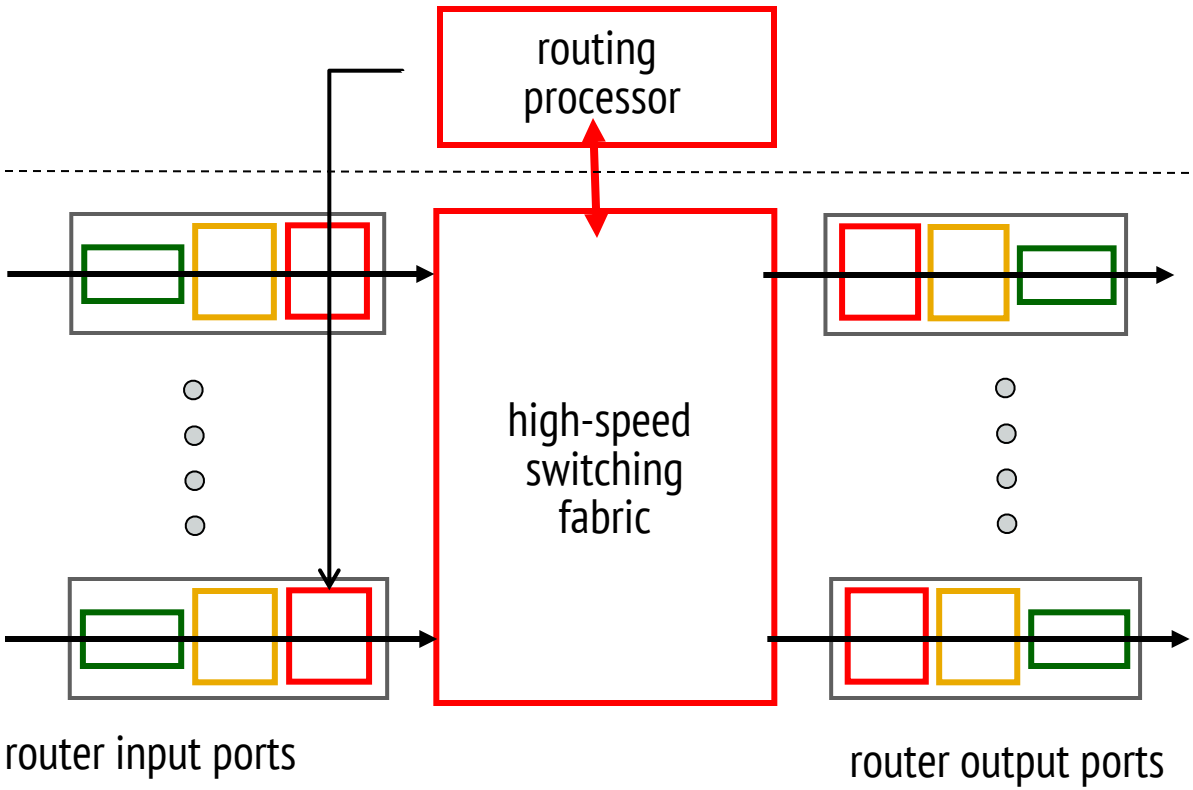
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

# Two key network-layer functions

| *network-layer functions:* | *analogy: taking a trip* |
| --- | --- |
| ➢ *forwarding:* move packets from router's input to appropriate router output | ➢ *forwarding:* process of getting through single interchange |
| ➢ *routing:* determine route taken by packets from source to destination<br>▪ *routing algorithms* | ➢ *routing:* process of planning trip from source to destination |

# Router architecture overview

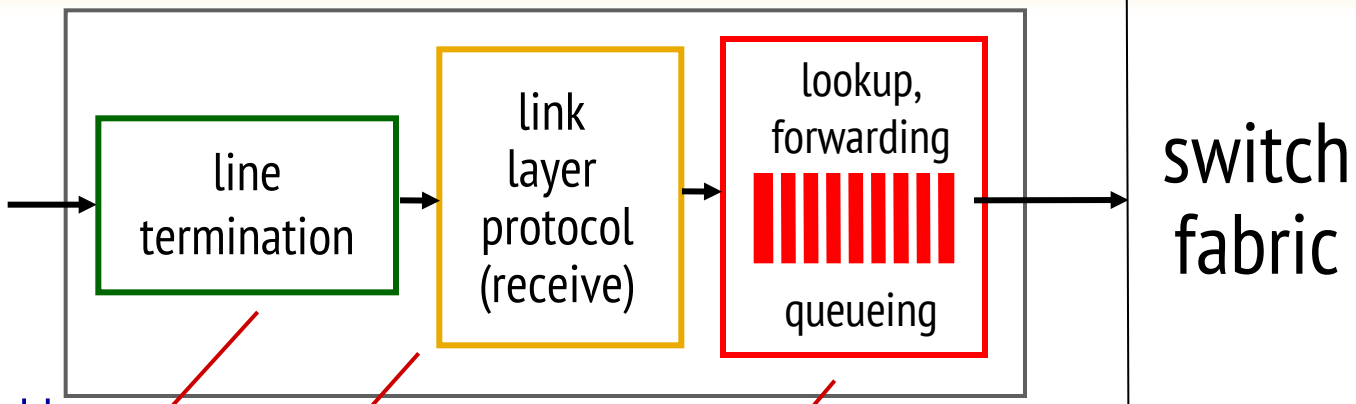
- high-level view of generic router architecture:




*routing, management control plane* (software) operates in millisecond time frame

*forwarding data plane* (hardware) operttes in nanosecond timeframe

# Input port functions



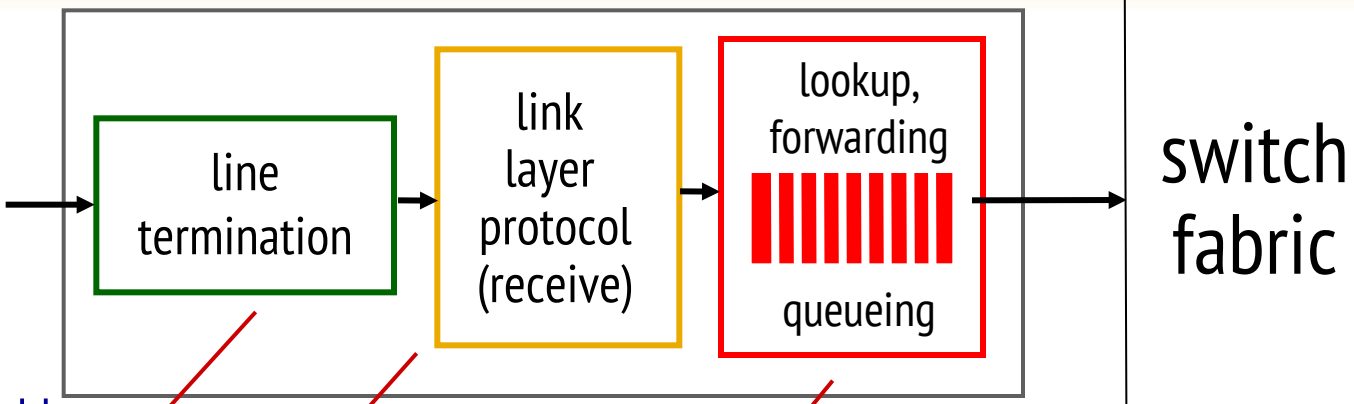

physical layer:
bit-level reception

data link layer:
e.g., Ethernet

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'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

# Input port functions




physical layer:
bit-level reception

data link layer:
e.g., Ethernet

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

# Destination based forwarding

## Forwarding Table

| Destination Address Range | Link Interface |
|---|---|
| **11001000 00010111 0001**0000 00000000<br>through<br>**11001000 00010111 0001**0111 11111111 | 0 |
| **11001000 00010111 0001**1000 00000000<br>through<br>**11001000 00010111 0001**1000 11111111 | 1 |
| **11001000 00010111 0001**1001 00000000<br>through<br>**11001000 00010111 0001**1111 11111111 | 2 |
| otherwise | 3 |

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

# Longest prefix matching

*longest prefix matching*

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

| Destination Address Range | Link Interface |
|---|---|
| 11001000 00010111 00010*** ********* | 0 |
| 11001000 00010111 00011000 ********* | 1 |
| 11001000 00010111 00011*** ********* | 2 |
| otherwise | 3 |

examples:

DA: 11001000 00010111 00010110 10100001 — which interface?

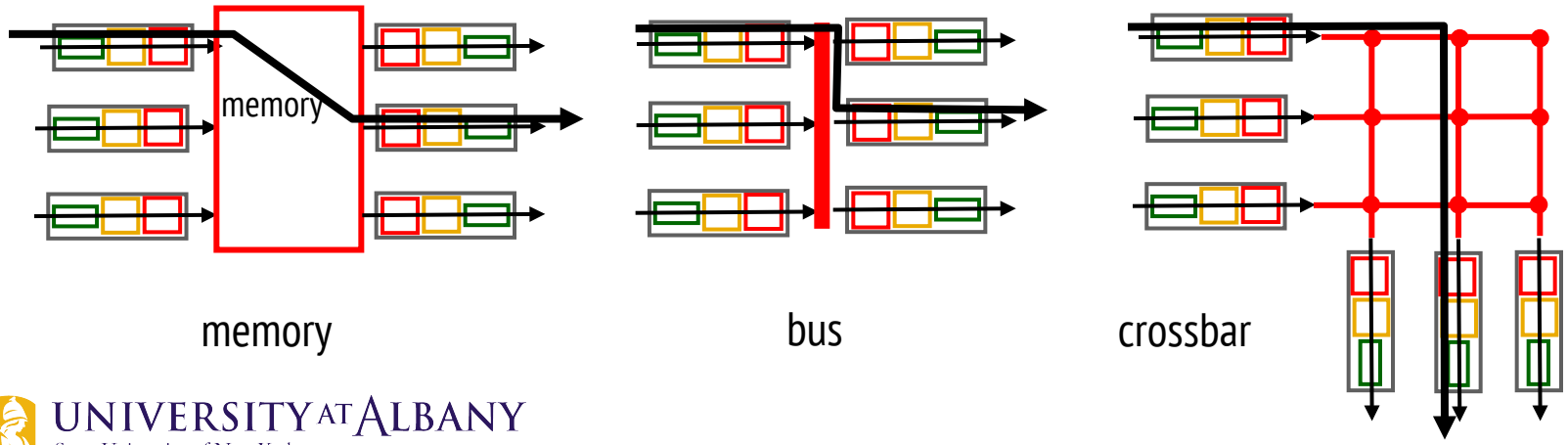
DA: 11001000 00010111 00011000 10101010 — which interface?

# Longest prefix matching

- 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 address in one clock cycle, regardless of table size
  - Cisco Catalyst: can up ~1M routing table entries in TCAM

# Switching fabrics

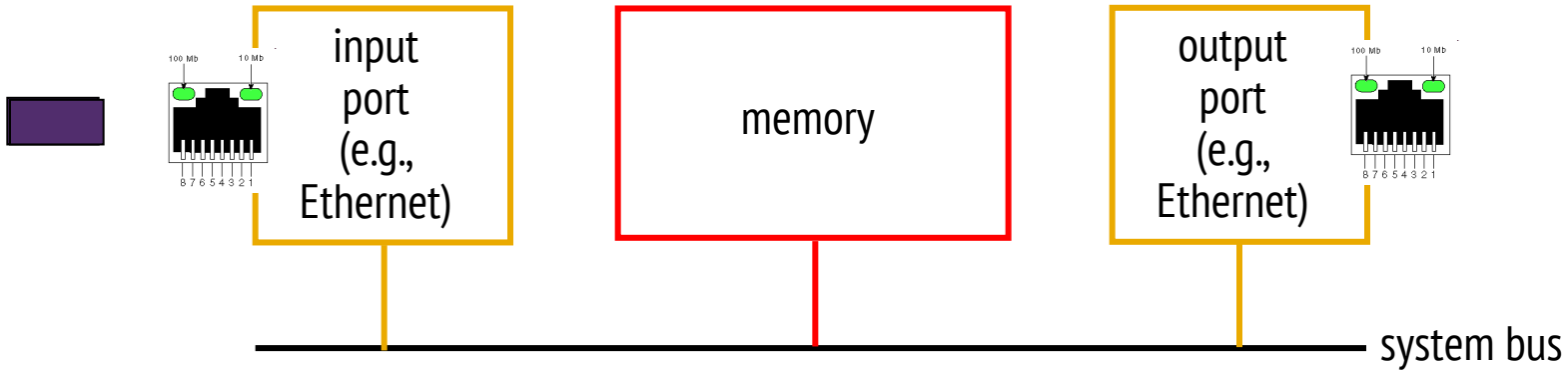
- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics



# Switching via memory

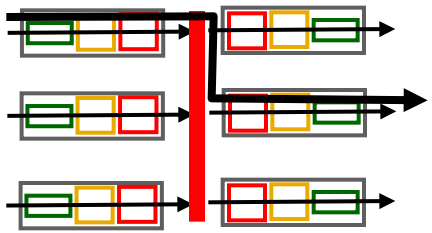
*first generation routers:*

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)
- Forwarding rate < B/2
- Two packets cannot be forwarded at the same time (read and write cannot be done in same cycle.)



# Switching via a bus

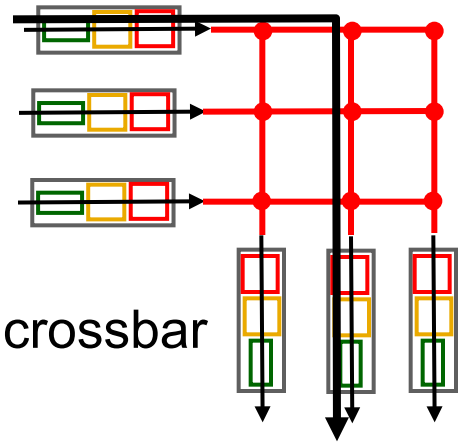
- datagram from input port memory to all output ports memory via a shared bus
- Each packet is attached an internal label, which output port checks to forward
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers



bus

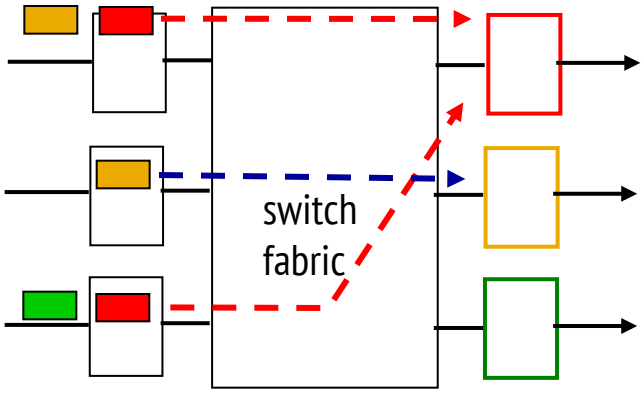
# Switching via interconnection network

- overcome  bus bandwidth limitations
- Switch fabric controller changes conn.
- It is non-blocking – forwarding to an output port happens if another packet is not forwarded to the same output port.
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network

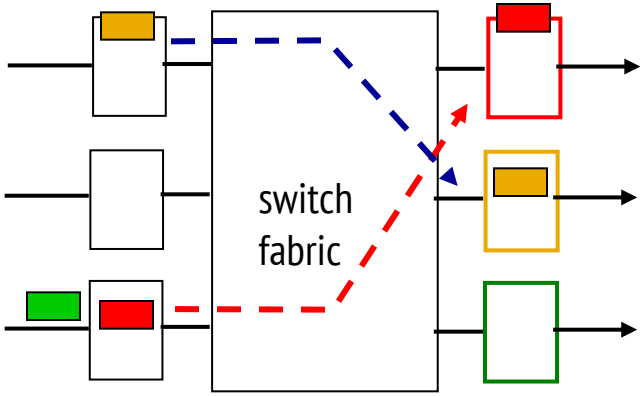


# Input port queuing

- 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

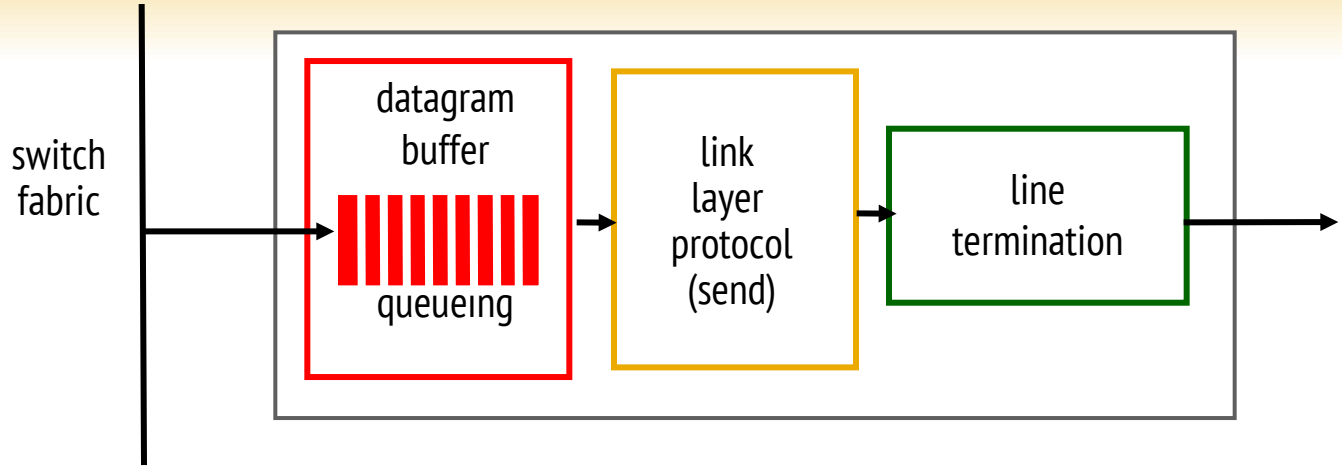



output port contention:
only one red datagram can be transferred.
*lower red packet is blocked*




one packet time later:
green packet experiences HOL blocking

# Output ports



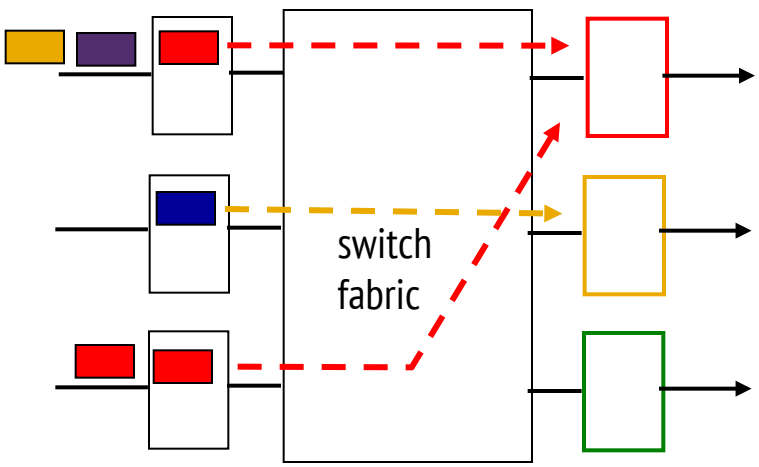

- *buffering* required when datagrams arrive from fabric faster than the transmission rate

Datagram (packets) can be lost due to congestion, lack of buffers

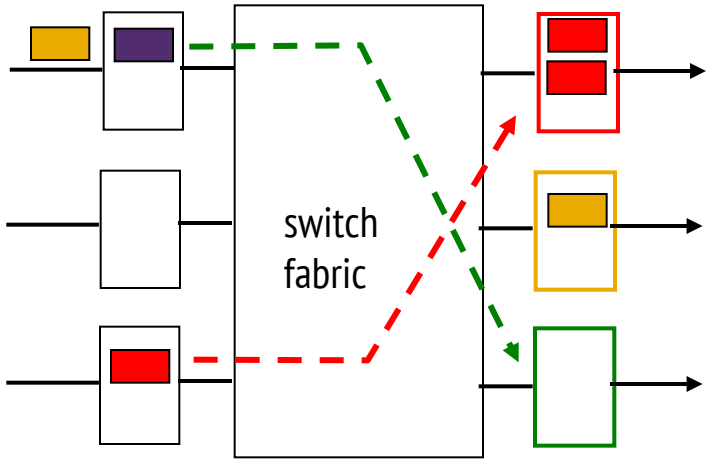
- *scheduling discipline* chooses among queued datagrams for transmission

Priority scheduling – who gets best performance, network neutrality

# Output port queueing




at *t*, packets more from input to output




one packet time later

- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

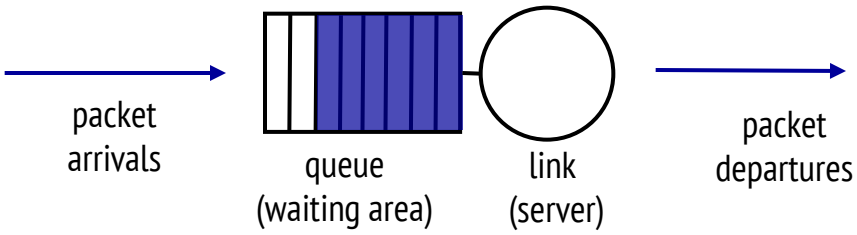
# How much buffering?

- 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

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

# Scheduling mechanisms

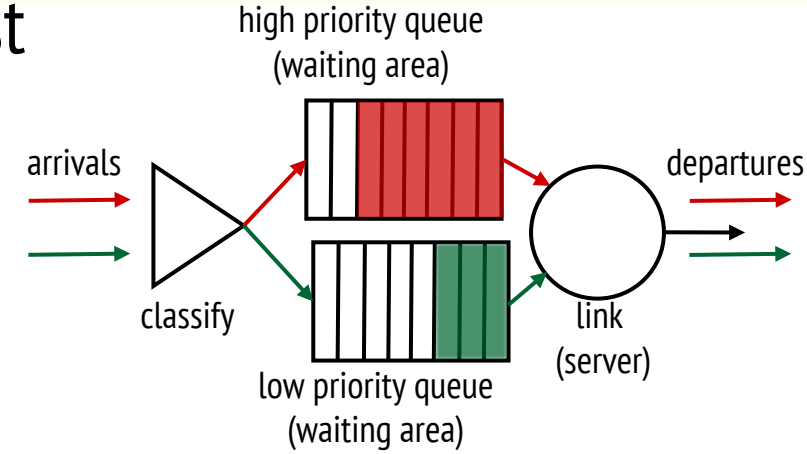
- *scheduling:* choose next packet to send on link
- *FIFO (first in first out) scheduling:* send in order of arrival to queue
  - real-world example?
  - *discard policy:* if packet arrives to full queue: who to discard?
    - *tail drop:* drop arriving packet
    - *priority:* drop/remove on priority basis
    - *random:* drop/remove randomly

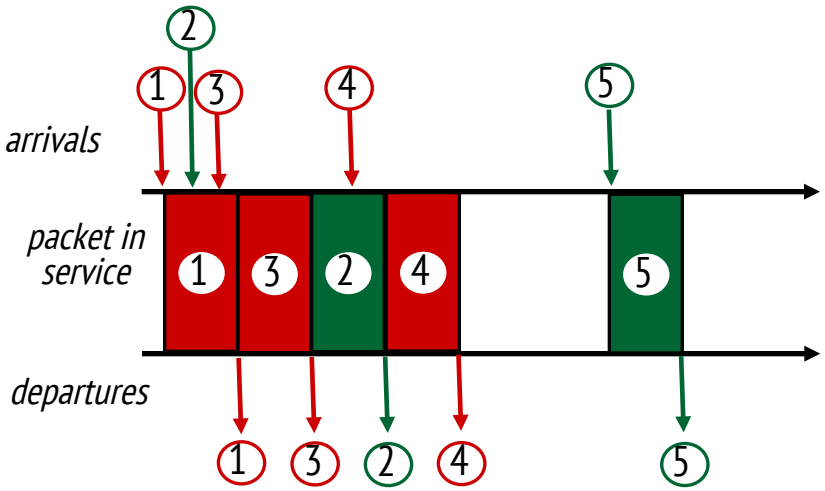


# Scheduling policies: priority

*priority scheduling:* send highest priority queued packet

- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?

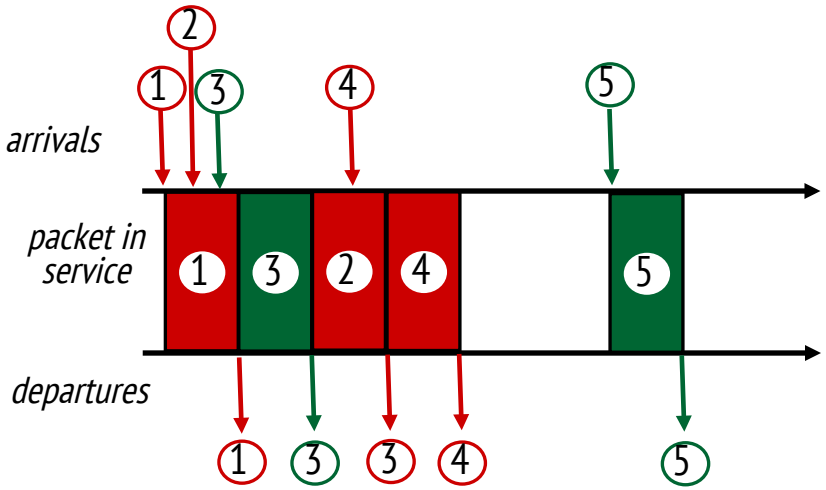


# Scheduling policies: still more

*Round Robin (RR) scheduling:*

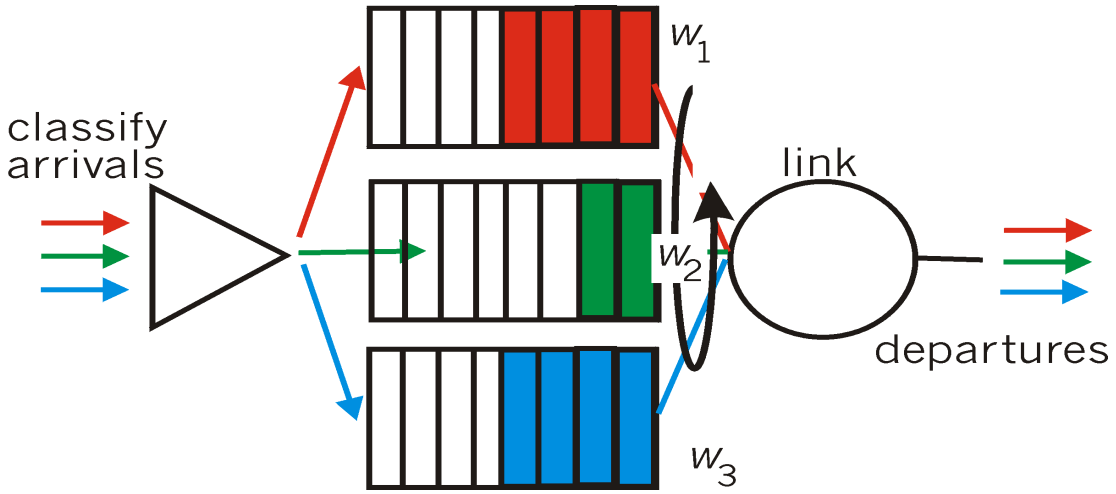
- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
- real world example?



# Scheduling policies: still more

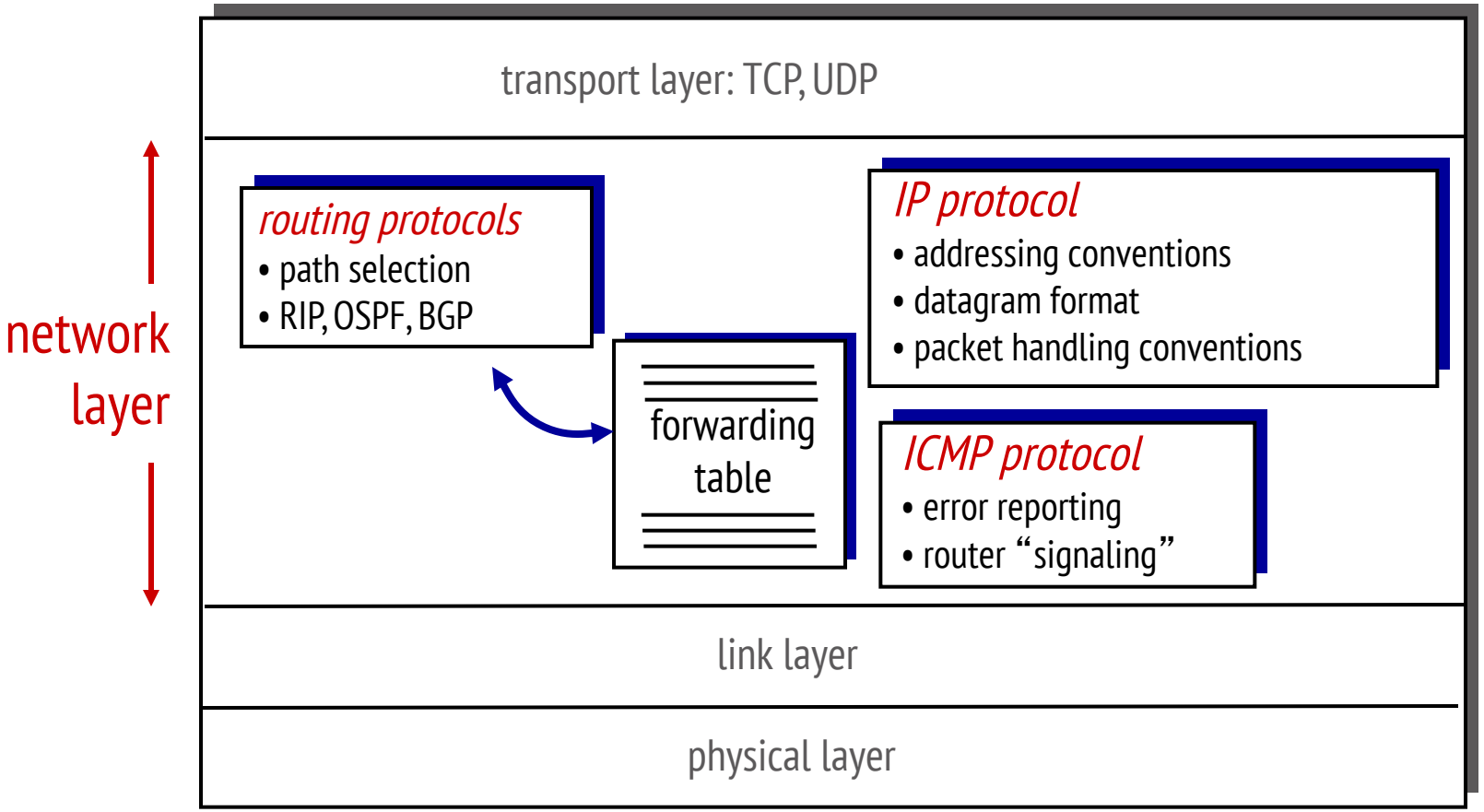
*Weighted Fair Queuing (WFQ):*

- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?

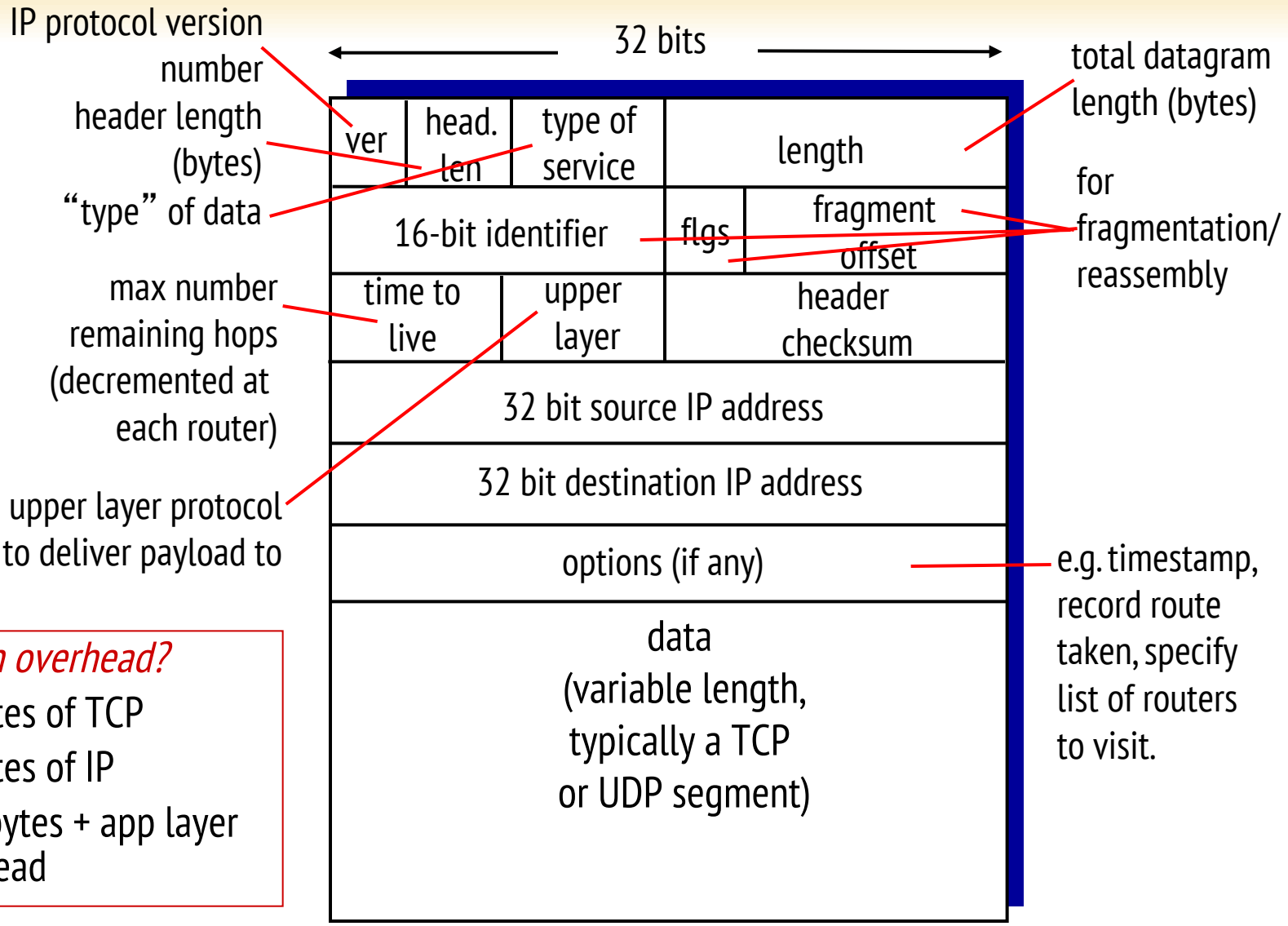


# The Internet network layer

host, router network layer functions:



# IP datagram format

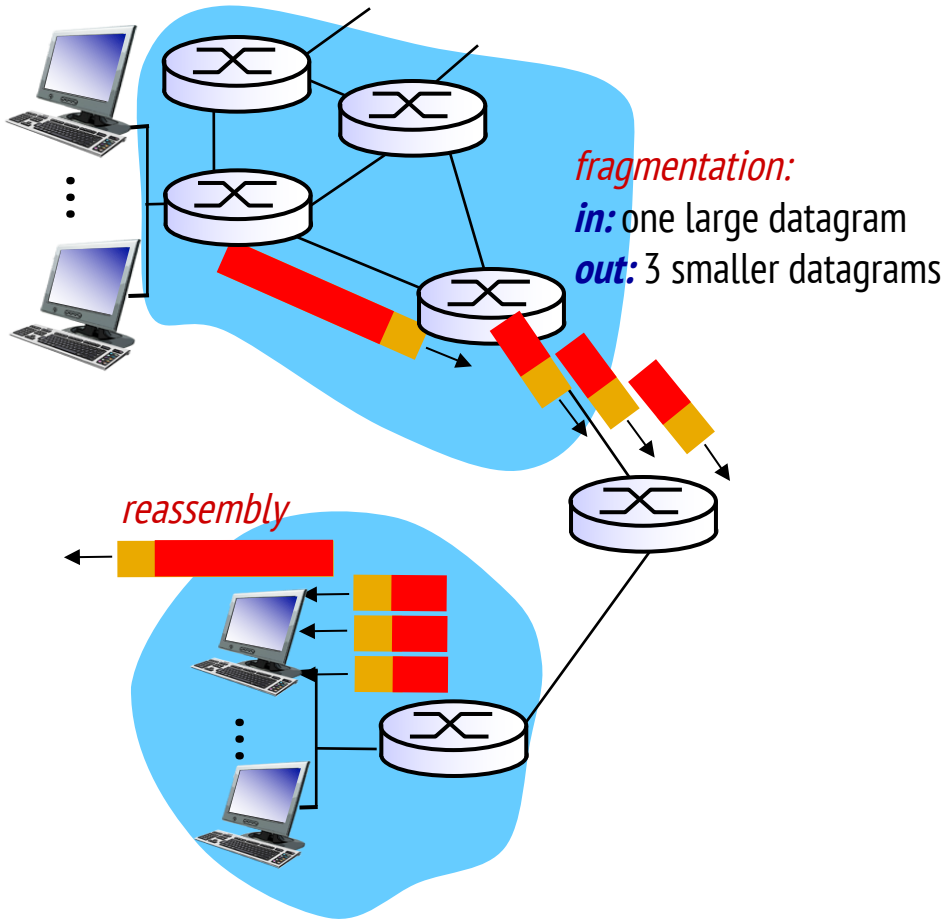



*how much overhead?*

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

# IP fragmentation, reassembly

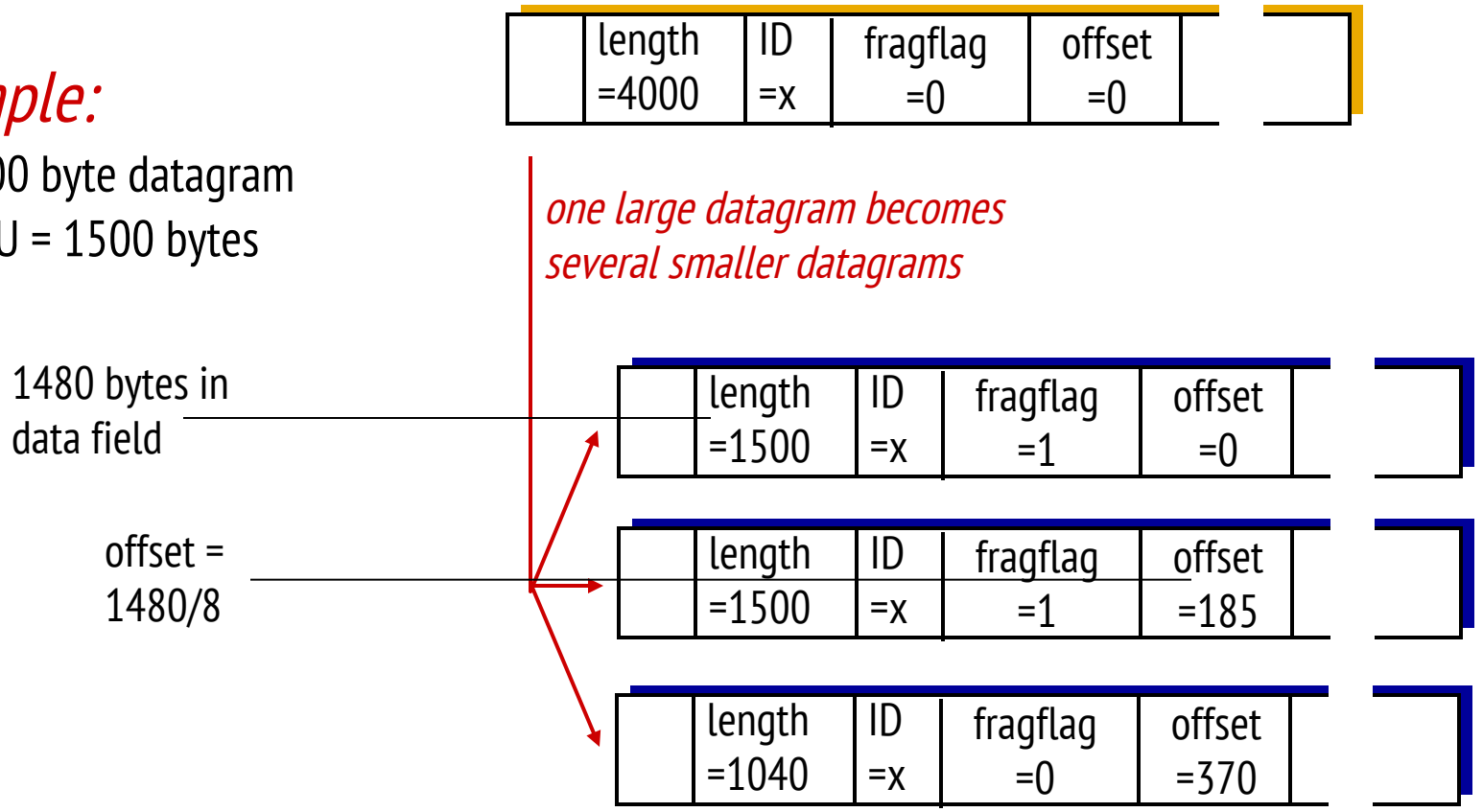
- network links have MTU (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify,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



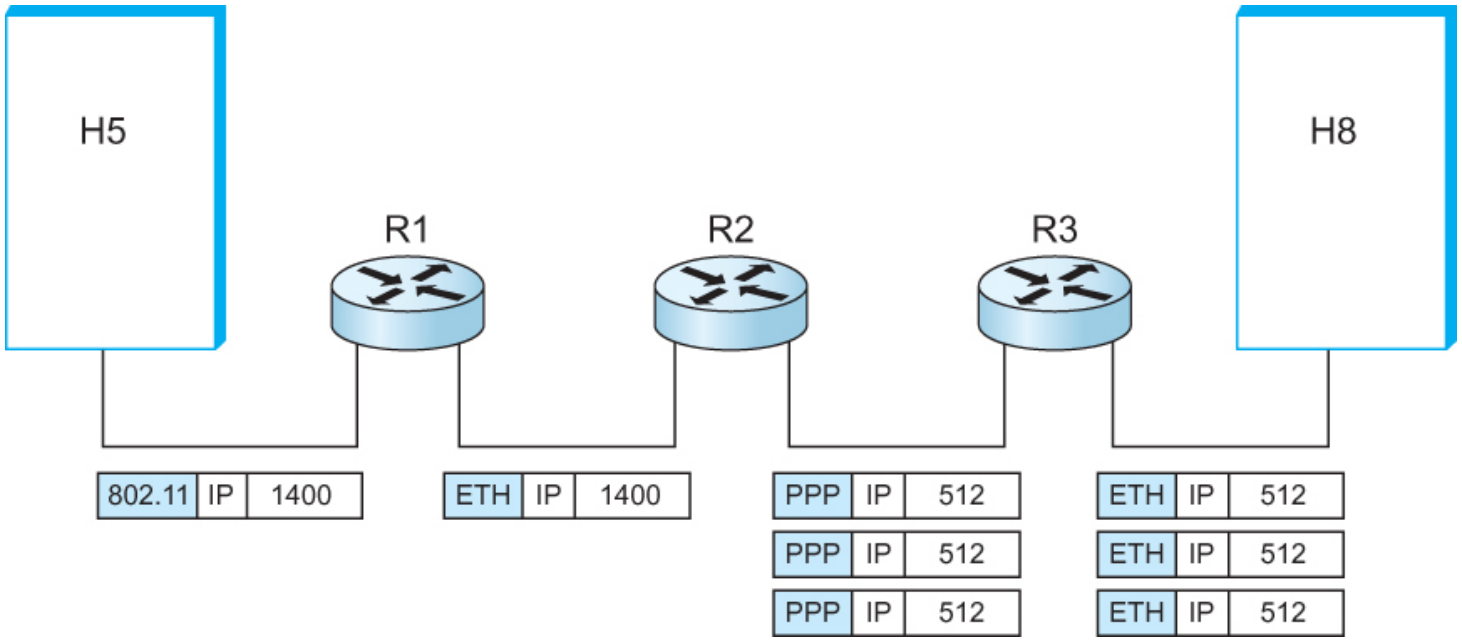
# IP fragmentation, reassembly

*example:*

- 4000 byte datagram
- MTU = 1500 bytes

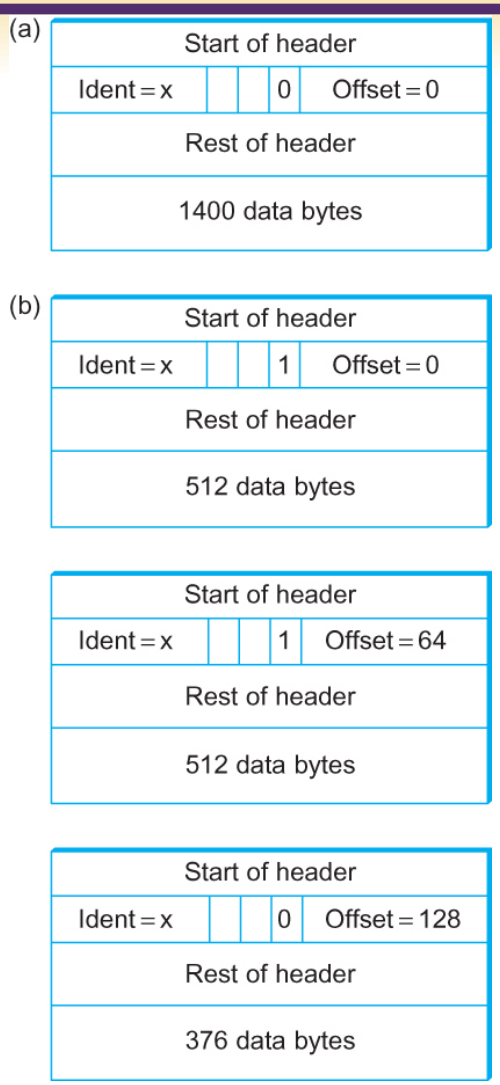


# IP Fragmentation and Reassembly




IP datagrams traversing the sequence of physical networks

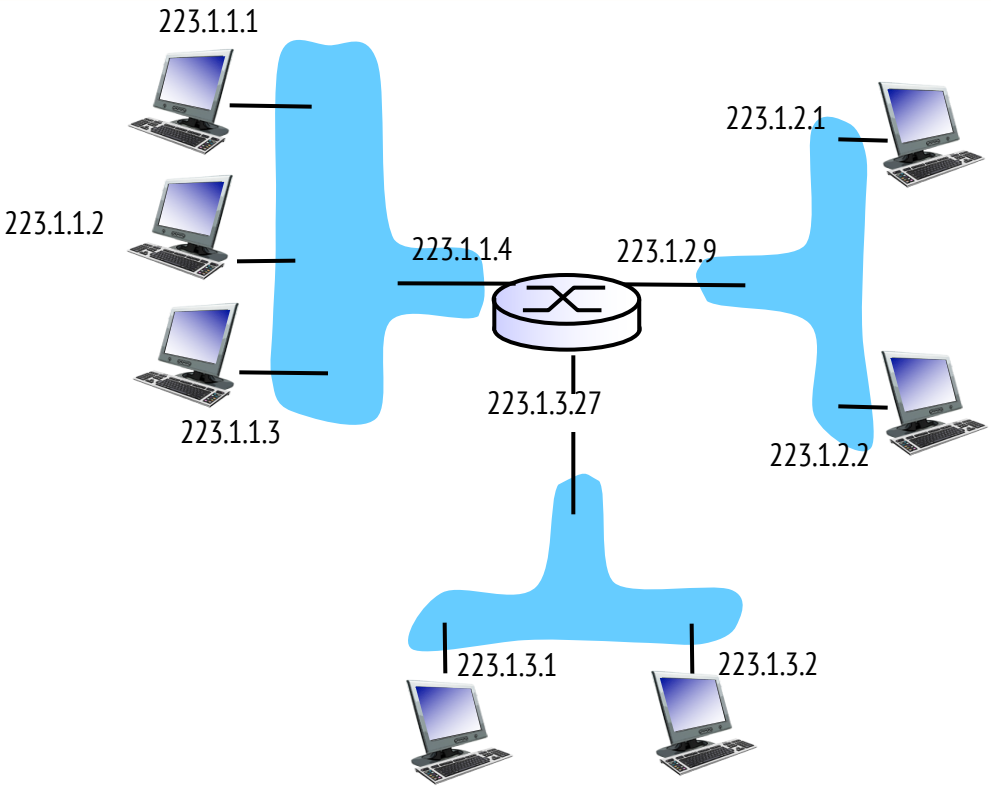
# IP Fragmentation and Reassembly




Header fields used in IP fragmentation. (a) Unfragmented packet; (b) fragmented packets.

# 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*



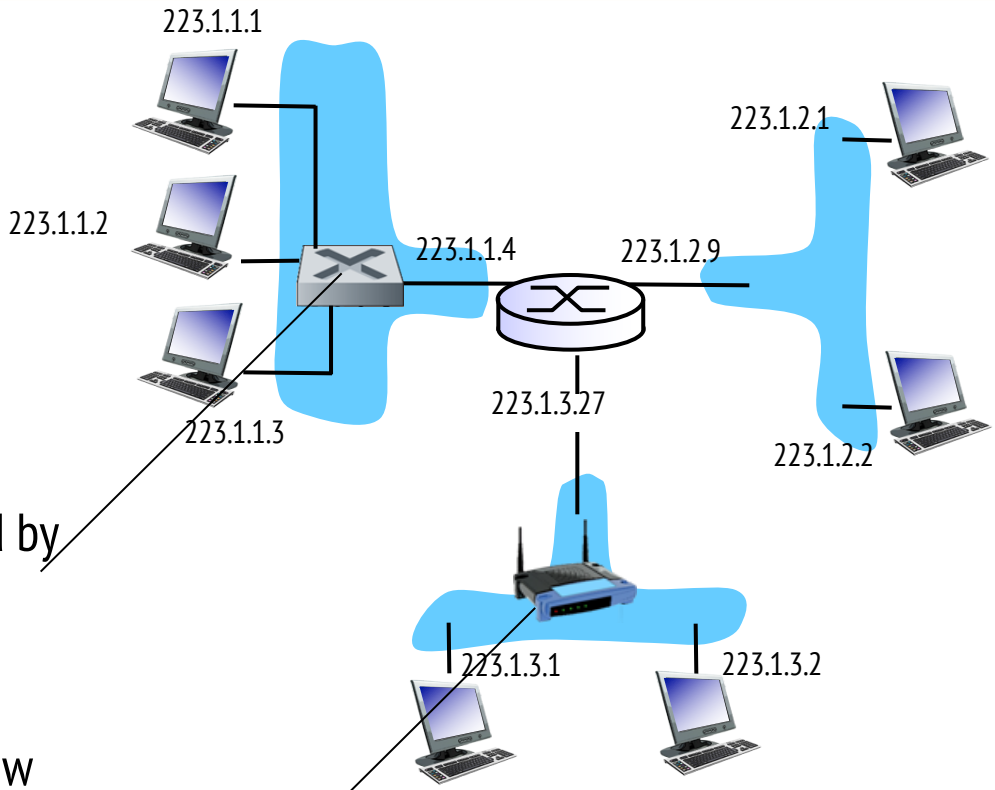

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

# IP addressing: introduction

*Q: how are interfaces actually connected?*

*A: we'll learn about that later.*



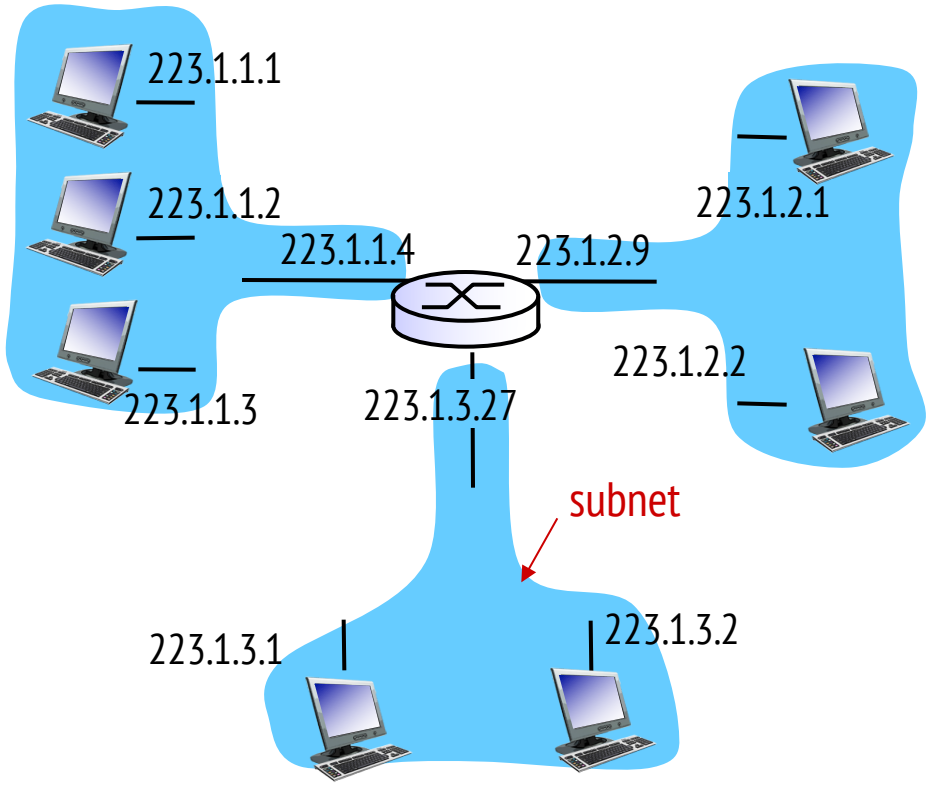

*A:* wired Ethernet interfaces connected by Ethernet switches

*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*

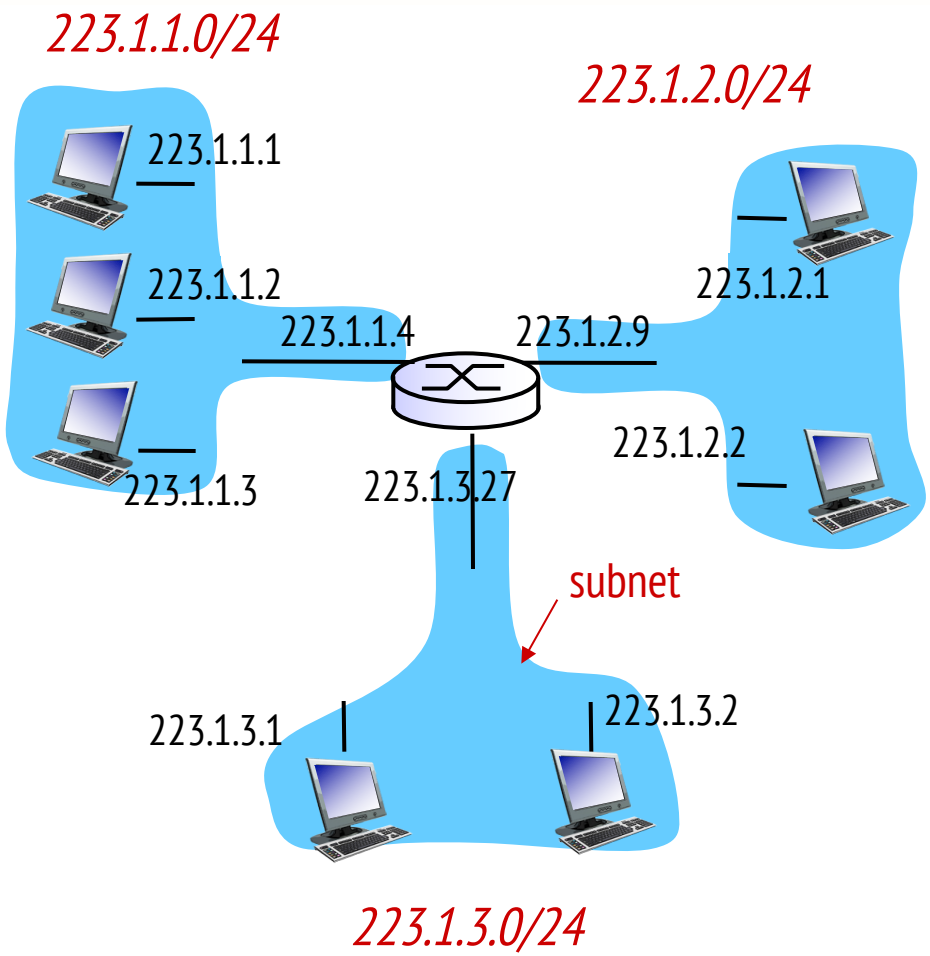



network consisting of 3 subnets

# Subnets

*recipe*

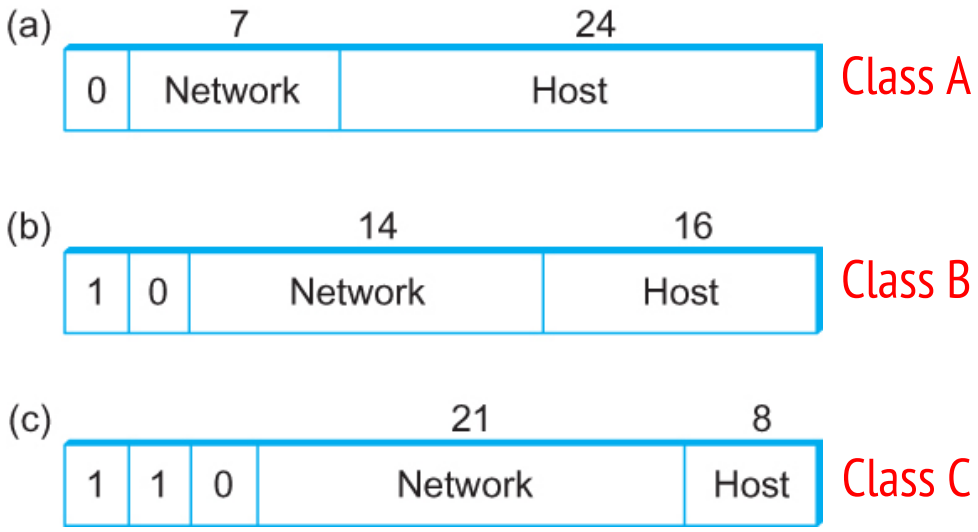
- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*
- *Slash notation or subnet mask: leftmost n bits for subnet*




subnet mask: /24

# Global Addresses

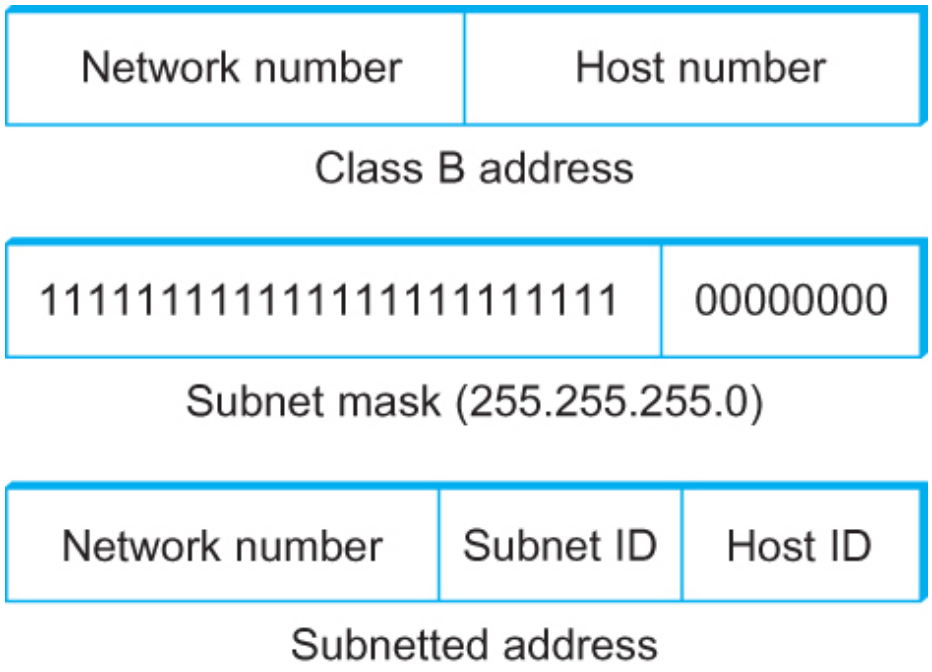
- Properties
  - globally unique
  - hierarchical: network + host
  - 4 Billion IP address, half are A type, ¼ is B type, and 1/8 is C type
- Format



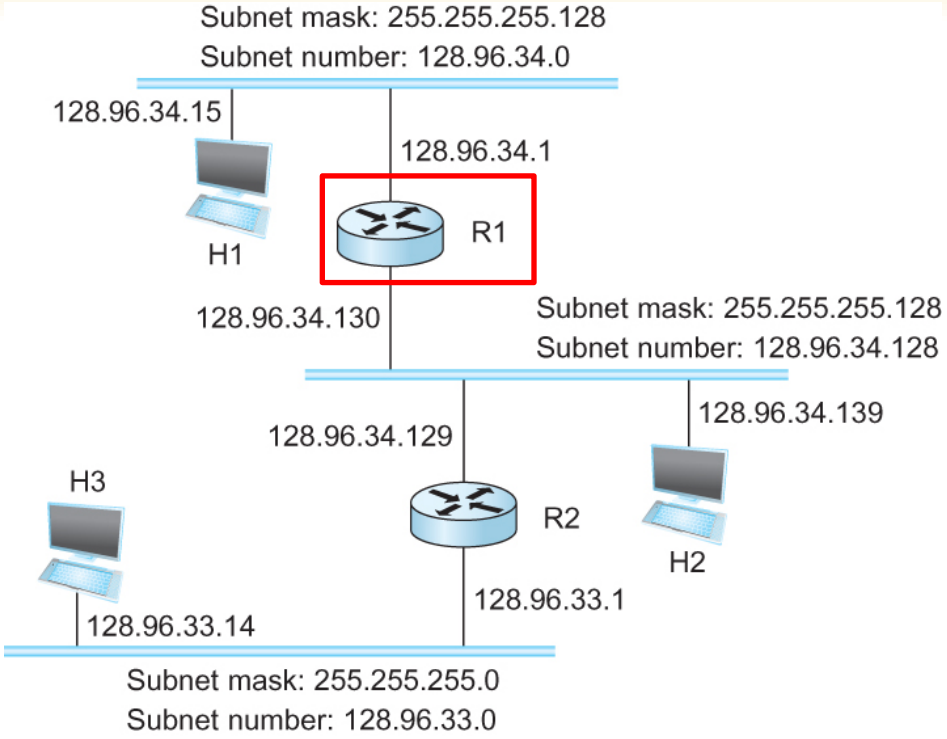

- Dot notation
  - 10.3.2.4
  - 128.96.33.81
  - 192.12.69.77

# Subnetting

- Add another level to address/routing hierarchy: *subnet*
- *Subnet masks* define variable partition of host part of class A and B addresses
- Subnets visible only within site



# Subnetting




- Forwarding Table at Router R1

| SubnetNumber | SubnetMask | NextHop |
|---|---|---|
| 128.96.34.0 | 255.255.255.128 | Interface 0 |
| 128.96.34.128 | 255.255.255.128 | Interface 1 |
| 128.96.33.0 | 255.255.255.0 | R2 |

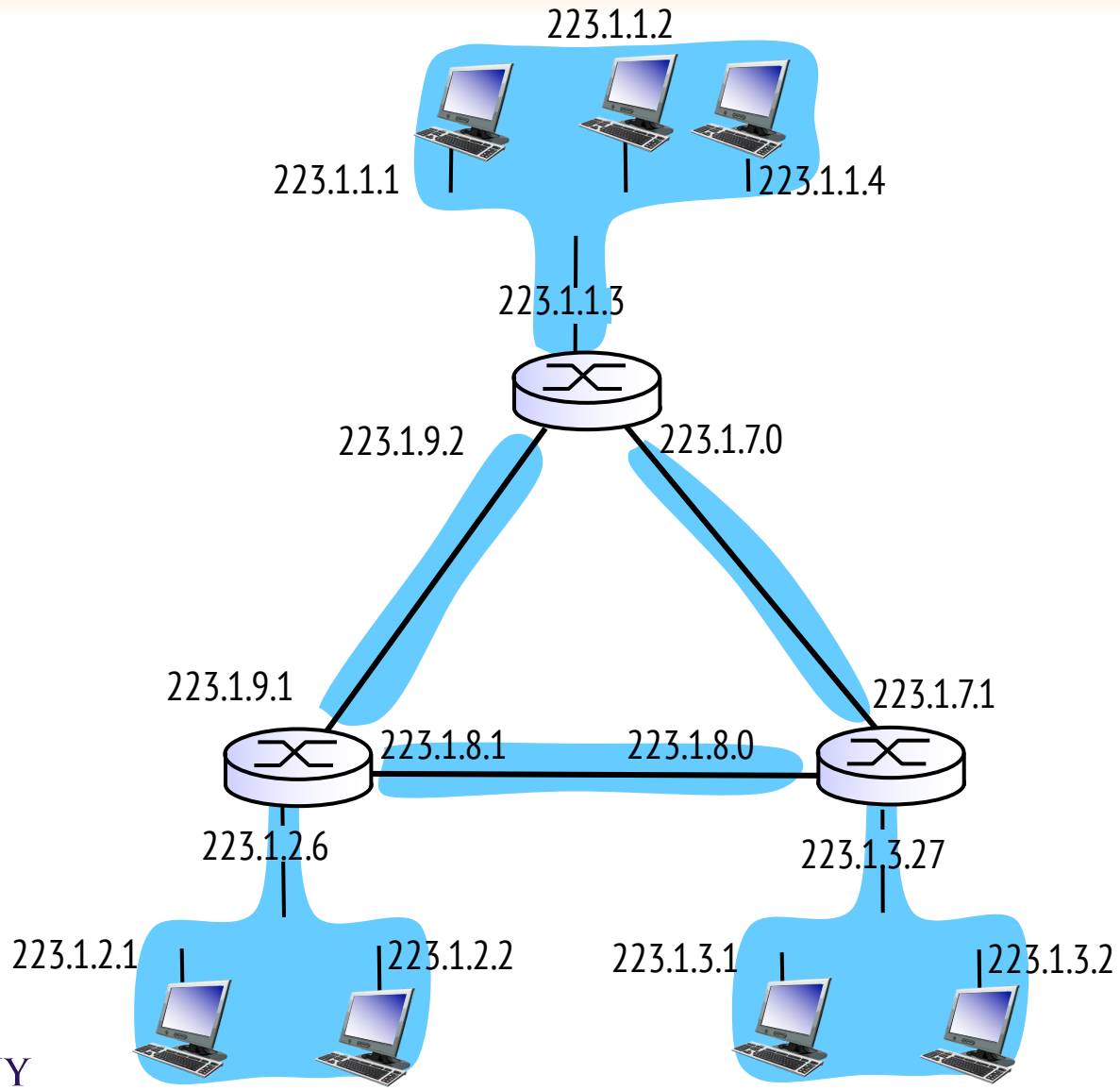
# Subnetting

## Forwarding Algorithm

```
D = destination IP address
for each entry < SubnetNum, SubnetMask, NextHop>
 D1 = SubnetMask & D
 if D1 = SubnetNum
  if NextHop is an interface
     deliver datagram directly to destination
  else
     deliver datagram to NextHop (a router)
```

# Subnets

how many?



# Classless Addressing

## ➢ Classless Inter-Domain Routing

- A technique that addresses two scaling concerns in the Internet
  - The growth of backbone routing table as more and more network numbers need to be stored in them
  - Potential exhaustion of the 32-bit address space

- Address assignment efficiency
  - Arises because of the IP address structure with class A, B, and C addresses
  - Forces us to hand out network address space in fixed-size chunks of three very different sizes
    - ✓ A network with two hosts needs a class C address
      - Address assignment efficiency = 2/255 = 0.78
    - ✓ A network with 256 hosts needs a class B address
      - Address assignment efficiency = 256/65535 = 0.39

# Classless Addressing

- Exhaustion of IP address space centers on exhaustion of the class B network numbers
- Solution
  - Say "NO" to any Autonomous System (AS) that requests a class B address unless they can show a need for something close to 64K addresses
  - Instead give them an appropriate number of class C addresses
  - For any AS with at least 256 hosts, we can guarantee an address space utilization of at least 50%
- What is the problem with this solution?

# Classless Addressing

- Problem with this solution
  - Excessive storage requirement at the routers.
- If a single AS has, say 16 class C network numbers assigned to it,
  - Every Internet backbone router needs 16 entries in its routing tables for that AS
  - This is true, even if the path to every one of these networks is the same
- If we had assigned a class B address to the AS
  - The same routing information can be stored in one entry
  - Efficiency = 16 × 255 / 65, 536 = 6.2%

# Classless Addressing

- CIDR tries to balance the desire to *minimize the number of routes that a router needs to know* against the need to hand out addresses efficiently.

- CIDR uses aggregate routes
  - Uses a single entry in the forwarding table to tell the router how to reach a lot of different networks
  - Breaks the rigid boundaries between address classes

# Classless Addressing

- Consider an AS with 16 class C network numbers.
- Instead of handing out 16 addresses at random, hand out a block of contiguous class C addresses
- Suppose we assign the class C network numbers from 192.4.16 through 192.4.31
- Observe that top 20 bits of all the addresses in this range are the same (11000000 00000100 0001)
  - We have created a 20-bit network number (which is in between class B network number and class C number)
- Requires to hand out blocks of class C addresses that share a common prefix

# Classless Addressing

- Requires to hand out blocks of class C addresses that share a common prefix
- The convention is to place a /X after the prefix where X is the prefix length in bits
- For example, the 20-bit prefix for all the networks 192.4.16 through 192.4.31 is represented as 192.4.16/20

- By contrast, if we wanted to represent a single class C network number, which is 24 bits long, we would write it 192.4.16/24

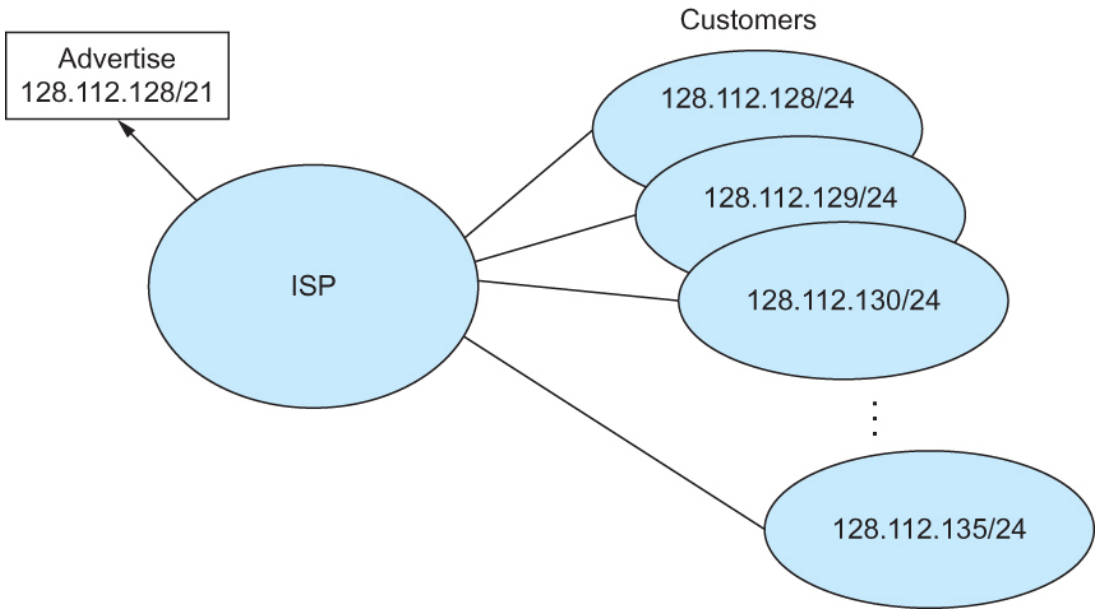
# Classless Addressing

- How do the routing protocols handle this classless addresses
  - It must understand that the network number may be of any length
- Represent network number with a single pair

  **<length, value>**

- All routers must understand CIDR addressing

# Classless Addressing




Route aggregation with CIDR

# IP Forwarding Revisited

- IP forwarding mechanism assumes that it can find the network number in a packet and then look up that number in the forwarding table
- We need to change this assumption in case of CIDR
- CIDR means that prefixes may be of any length, from 2 to 32 bits

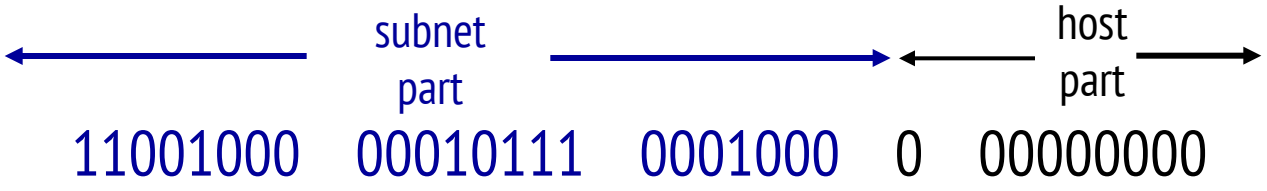
# IP Forwarding Revisited

- It is also possible to have prefixes in the forwarding tables that overlap
  - Some addresses may match more than one prefix
- For example, we might find both 171.69 (a 16 bit prefix) and 171.69.10 (a 24 bit prefix) in the forwarding table of a single router
- A packet destined to 171.69.10.5 clearly matches both prefixes.
  - The rule is based on the principle of "longest match"
    - 171.69.10 in this case
- A packet destined to 171.69.20.5 would match 171.69 and not 171.69.10

# 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




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 server
  - "plug-and-play"

# DHCP: Dynamic Host Configuration Protocol

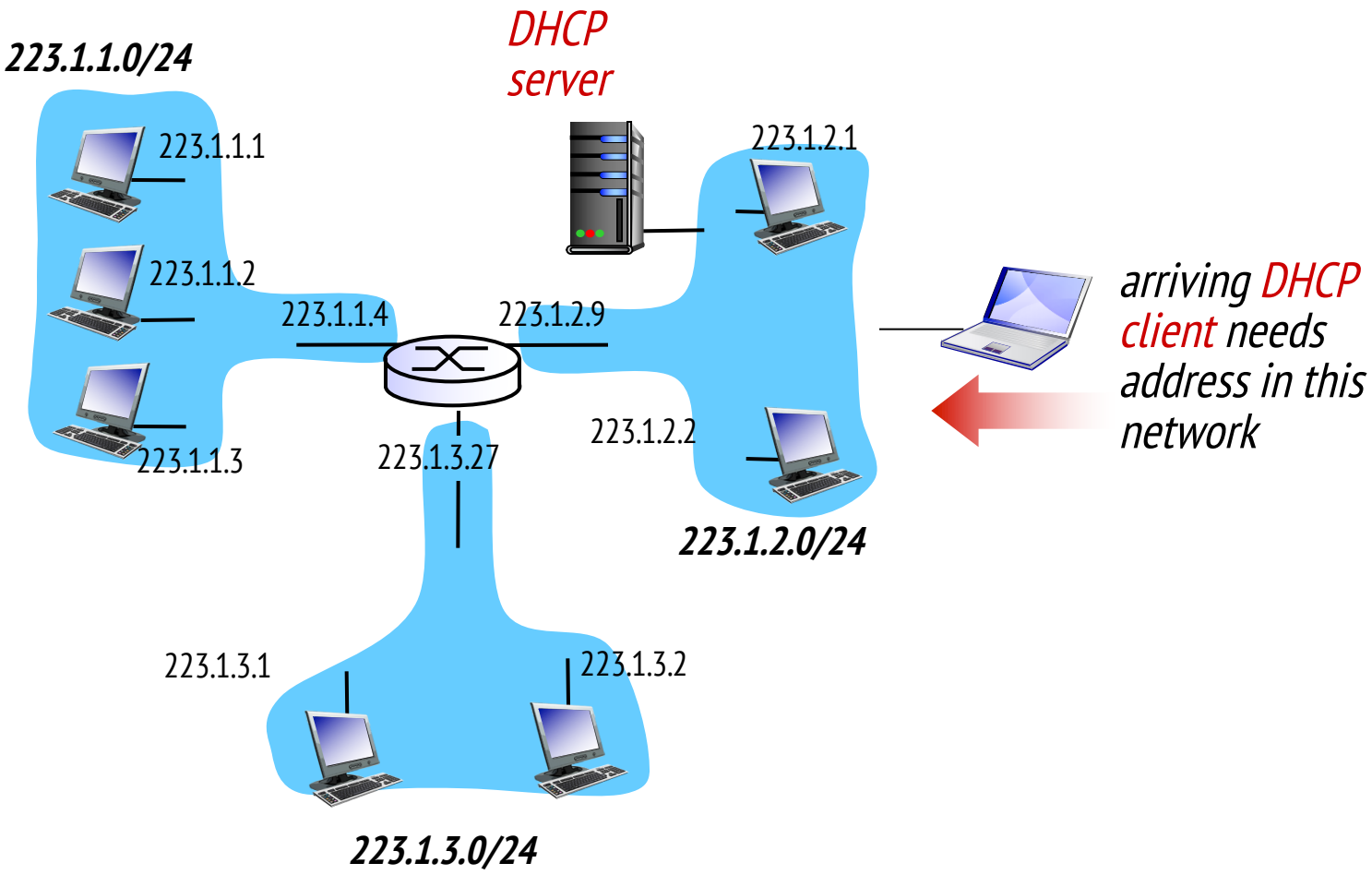
*goal:* allow host to *dynamically* obtain its IP address from network server when it joins network

- 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)

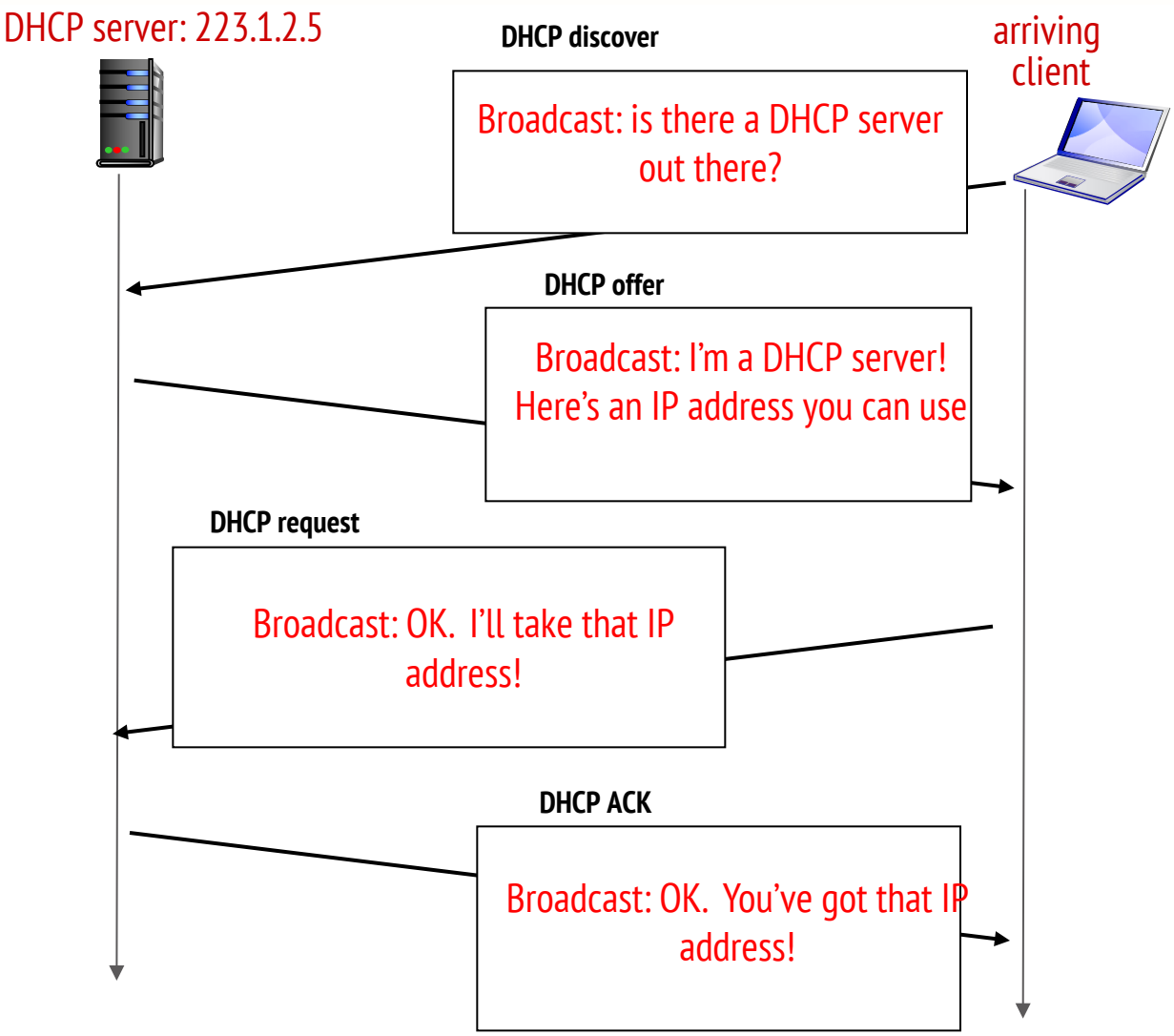
*DHCP overview:*

- 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

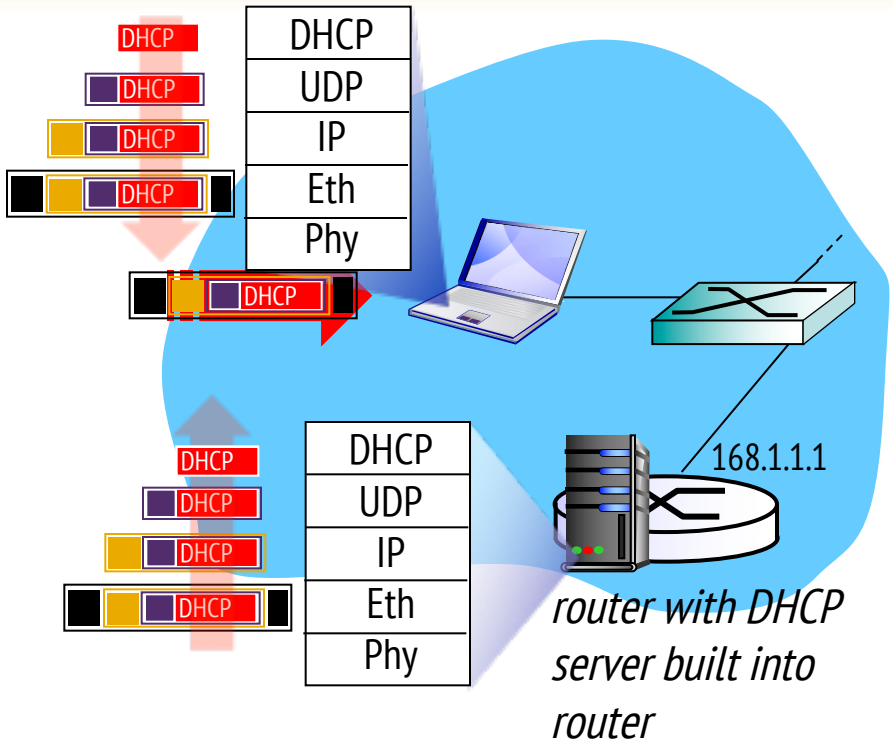


# DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

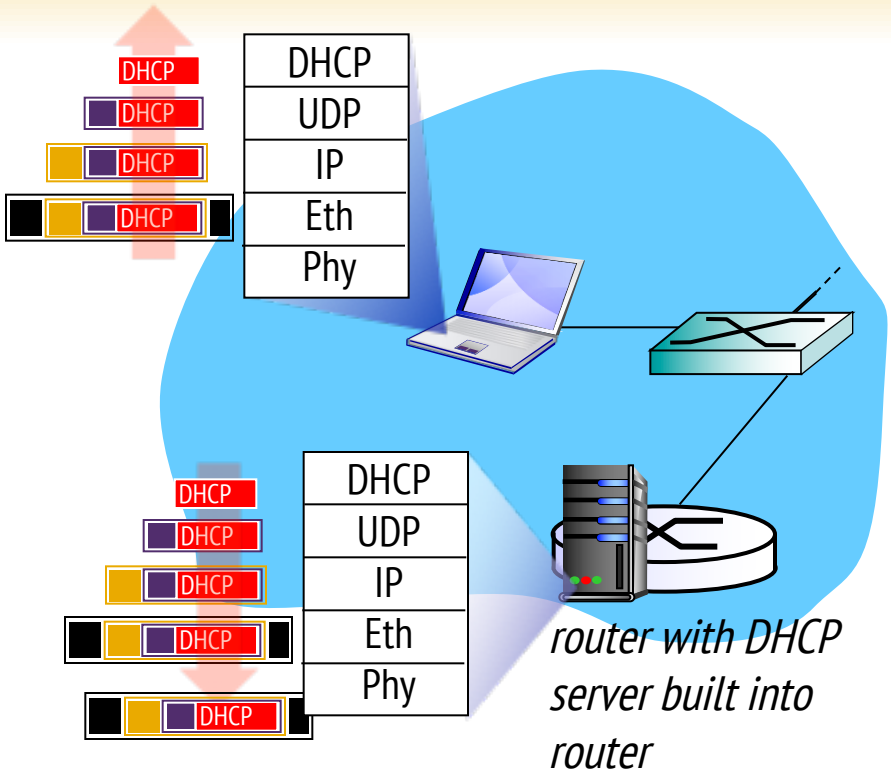
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# 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 DSN 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 |
| 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 |

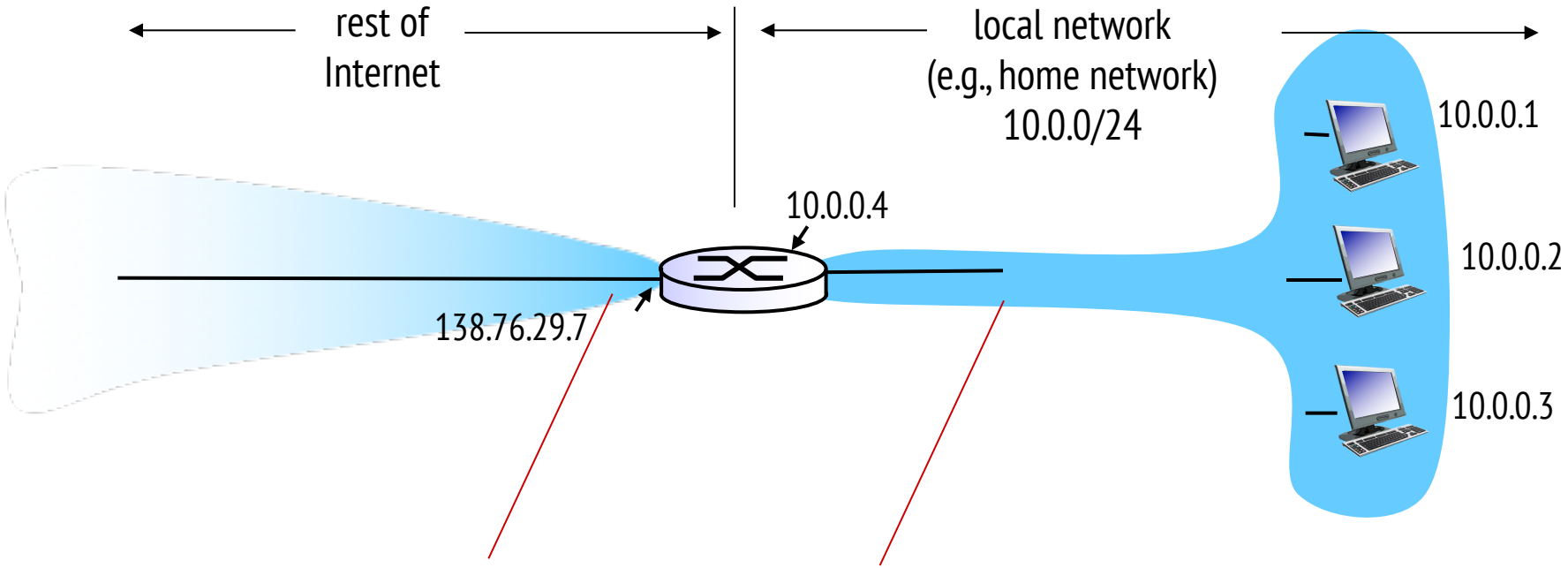
# IP addressing: the last word...

*Q:* how does an ISP get block of addresses?

*A:* ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

# 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

*motivation:* local network uses just one IP address as far as outside world is concerned:

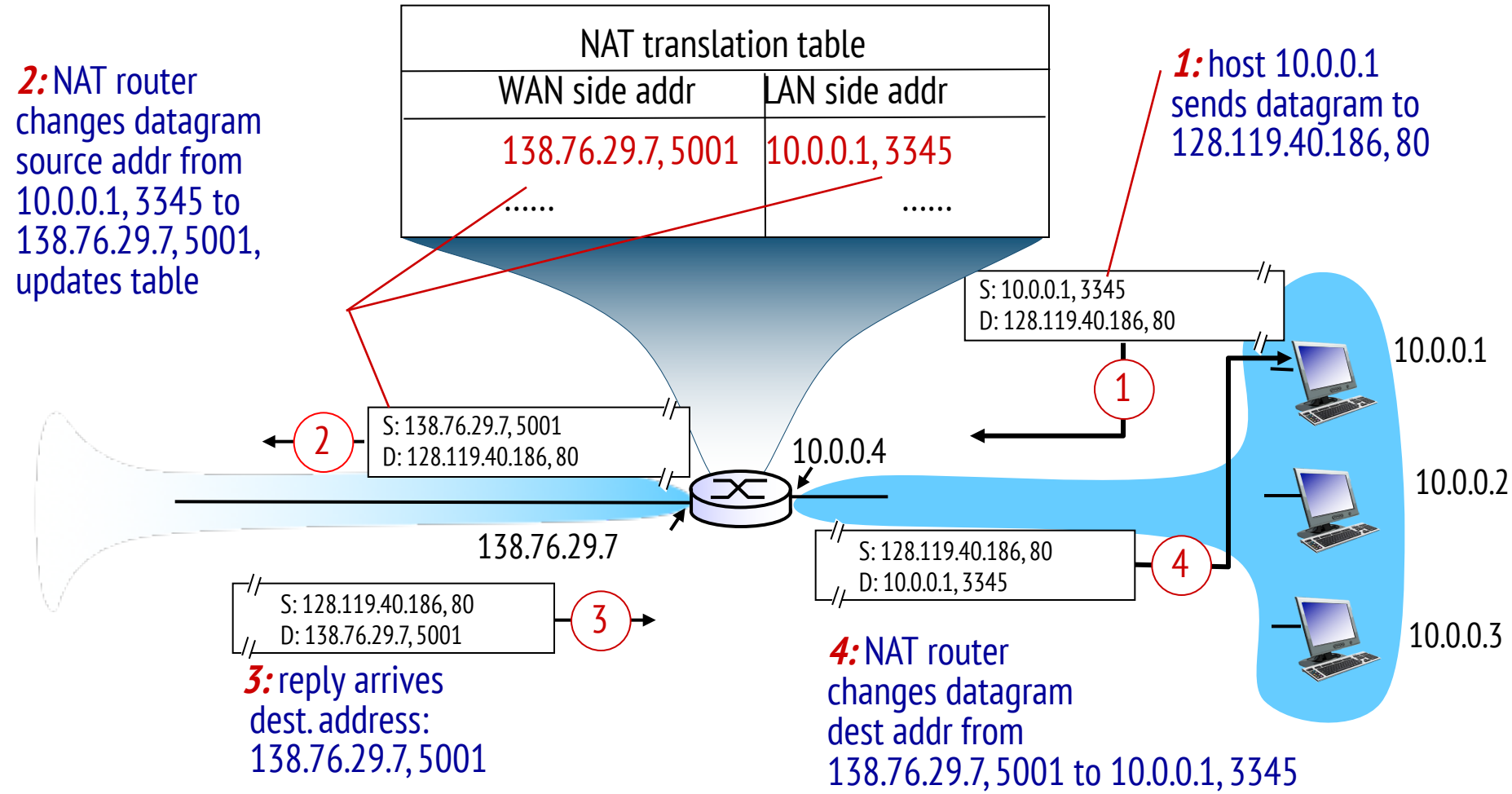
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

# NAT: network address translation

*implementation*: NAT router must:

- *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 addr
- *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

# NAT: network address translation



# NAT: network address translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - address shortage should be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications

# 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:*

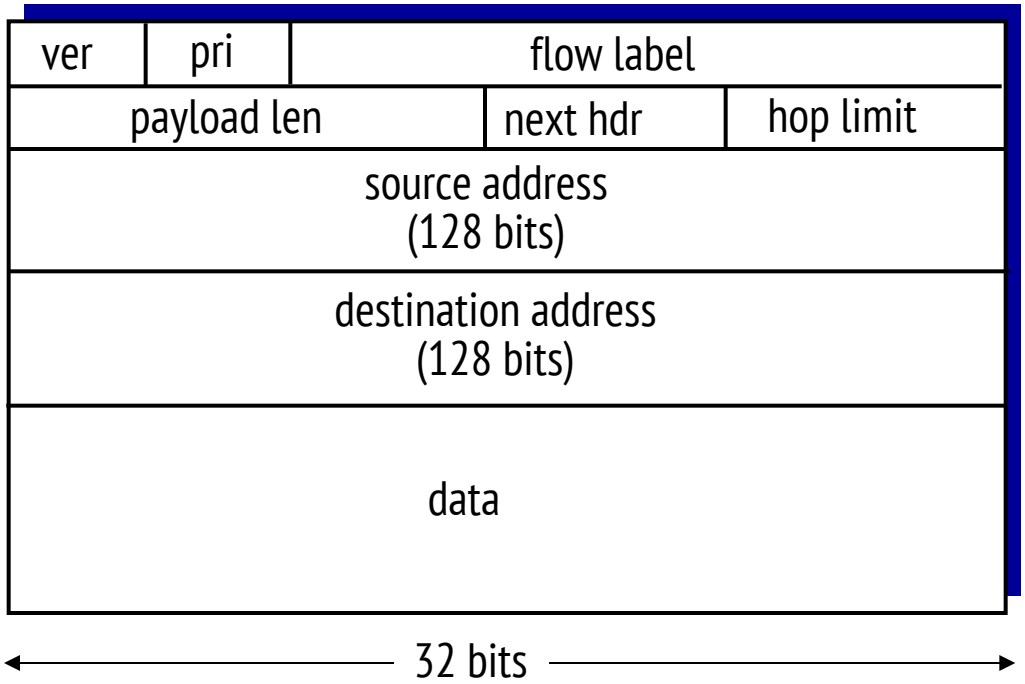
- fixed-length 40 byte header
- no fragmentation allowed

# IPv6 datagram format

*priority:* 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

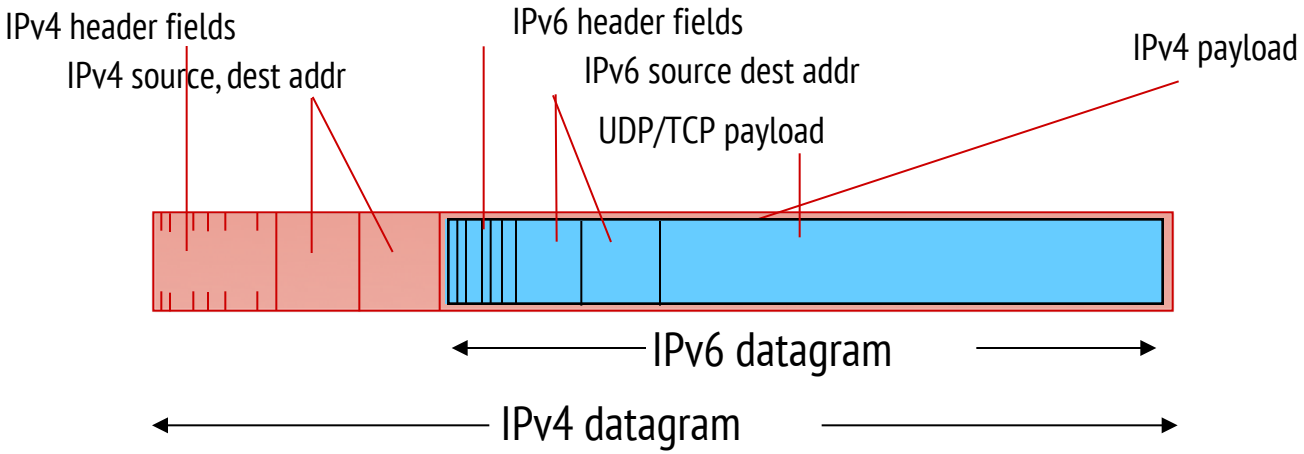


# Other changes from IPv4

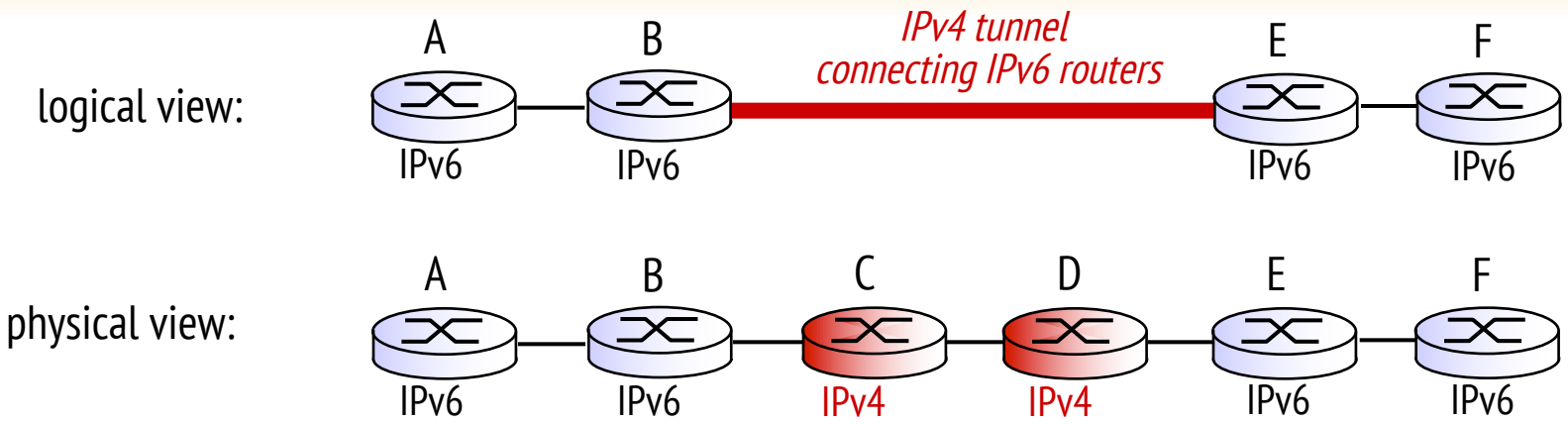
- *checksum*: removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by "Next Header" field
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
  - multicast group management functions

# Transition from IPv4 to IPv6

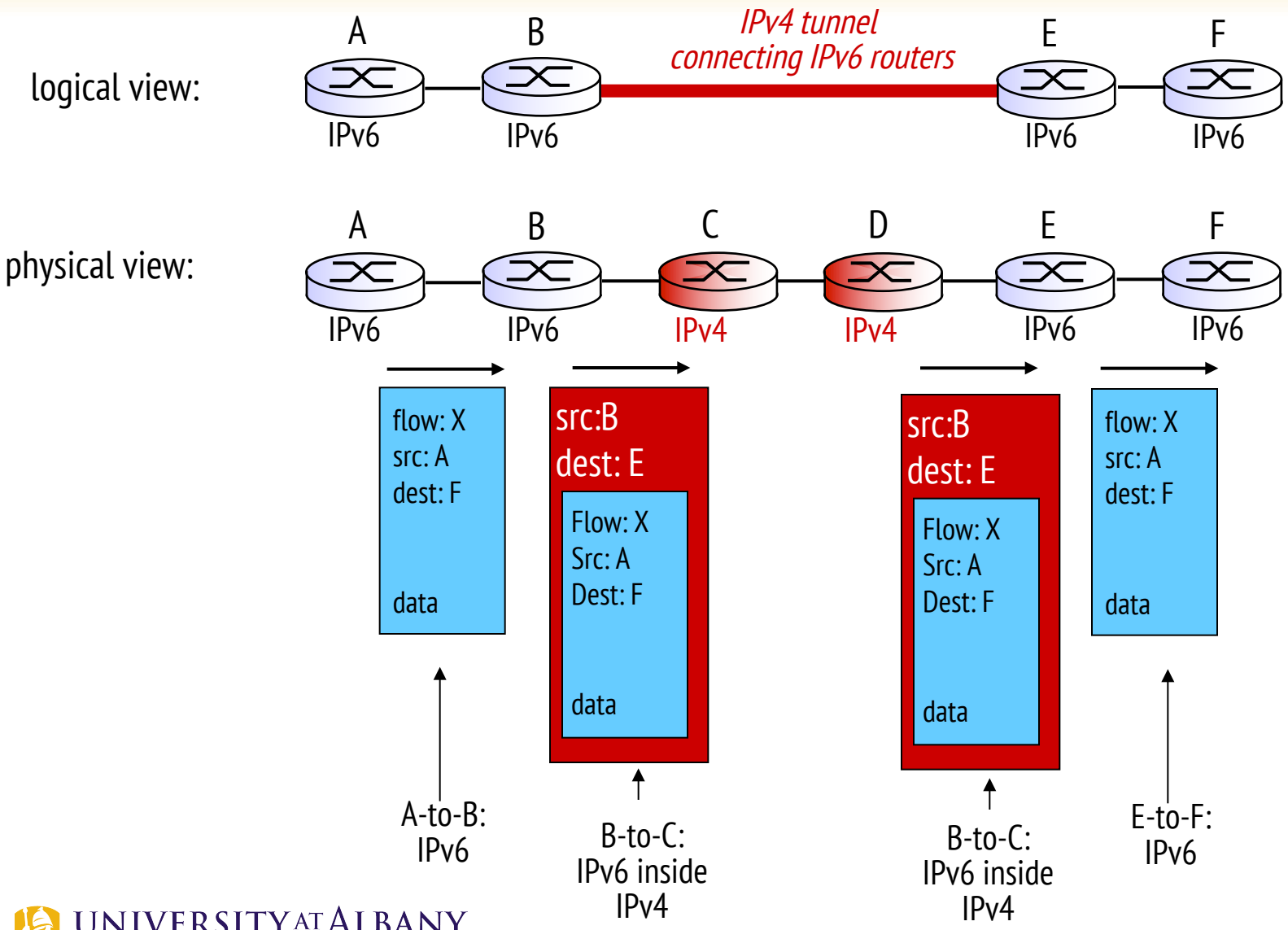
- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- *tunneling:* IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



# 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:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6

# IPv6: adoption

- Google: 11% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

- *Long (long!) time for deployment, use*
  - More than 20 years and counting! [IETF initiated standardization of IPv6 in 1994]
  - think of application-level changes in last 20 years: WWW, Facebook, streaming media, Skype, …

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

# Internet Control Message Protocol (ICMP)

- Defines a collection of error messages that are sent back to the source host whenever a router or host is unable to process an IP datagram successfully
  - Destination host unreachable due to link /node failure
  - Reassembly process failed
  - TTL had reached 0 (so datagrams don't cycle forever)
  - IP header checksum failed

- ICMP-Redirect
  - From router to a source host
  - With a better route information

# Internet Control Message Protocol (ICMP)

- Defines a collection of error messages that are sent back to the source host whenever a router or host is unable to process an IP datagram successfully
  - Destination host unreachable due to link /node failure
  - Reassembly process failed
  - TTL had reached 0 (so datagrams don't cycle forever)
  - IP header checksum failed

- ICMP-Redirect
  - From router to a source host
  - With a better route information

# 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 |

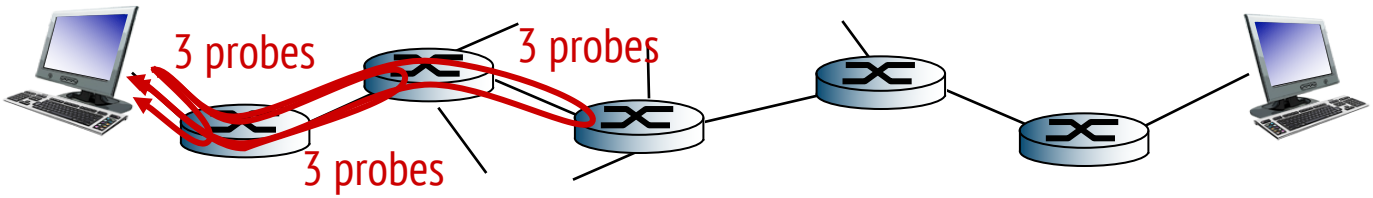
# Traceroute and ICMP

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

when ICMP message 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

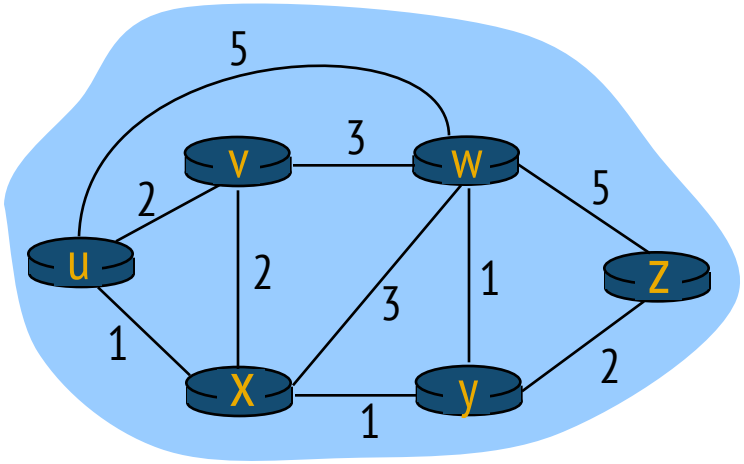


# Routing protocols

*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 given final destination host
- "good": least "cost", "fastest", "least congested"
- routing: a "top-10" networking challenge!

# Graph abstraction of the network



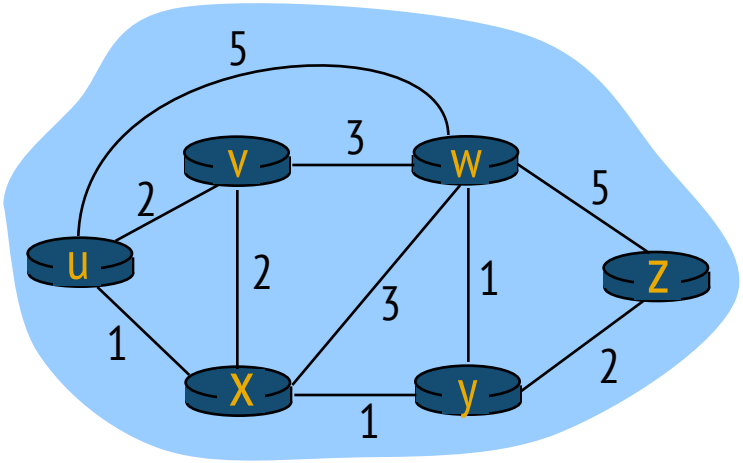

graph: G = (N,E)

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

E = 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(x,x') = cost of link (x,x')
e.g., c(w,z) = 5

cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

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

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

*Q: static or dynamic?*

*static:*

- routes change slowly over time
  - It does not deal with node or link failures
  - It does not consider the addition of new nodes or links
  - It implies that edge costs cannot change

*dynamic:*

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

# Link State Routing Algorithm

# 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

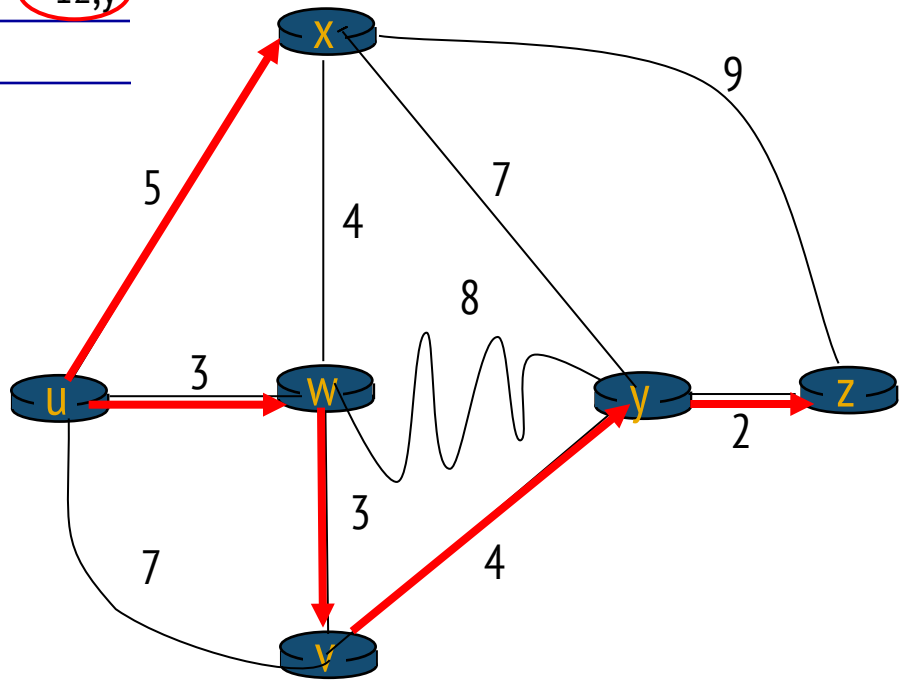
```
1  Initialization:
2   N' = {u}
3   for all nodes v
4     if v adjacent to u
5         then D(v) = c(u,v)
6     else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N' :
12      D(v) = min( D(v), D(w) + c(w,v)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

# 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 | | | | | |

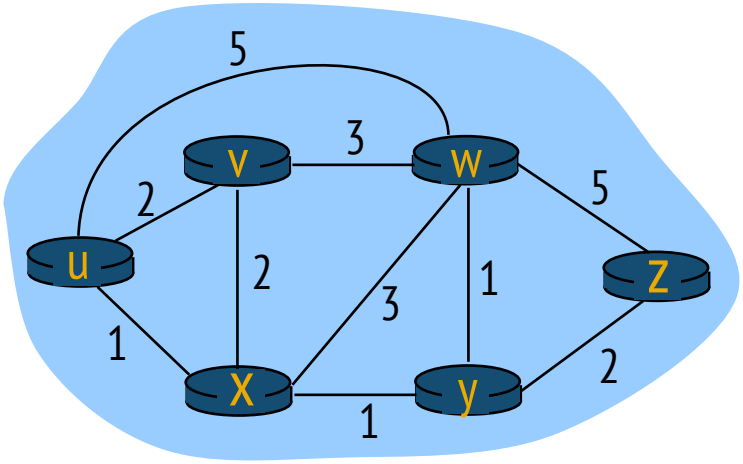
*notes:*

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



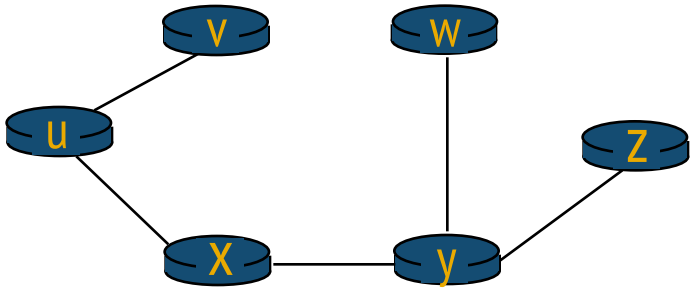
# Dijkstra's algorithm: another example

| 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) |

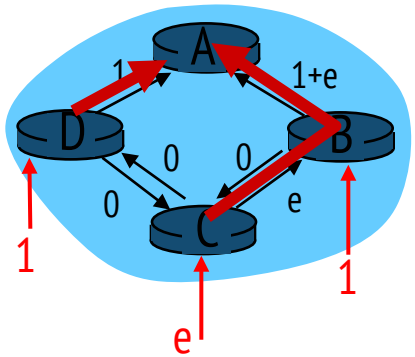
# Dijkstra's algorithm, discussion

*algorithm complexity:* n nodes

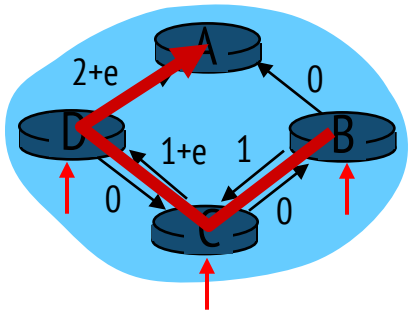
- each iteration: need to check all nodes, w, not in N
- $n(n+1)/2$ comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

*oscillations possible:*

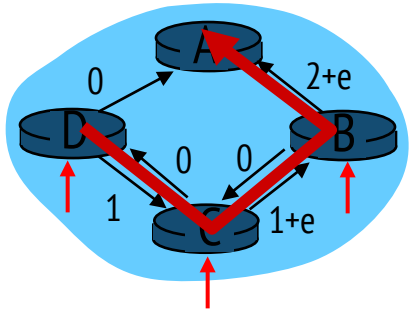
- e.g., support link cost equals amount of carried traffic:



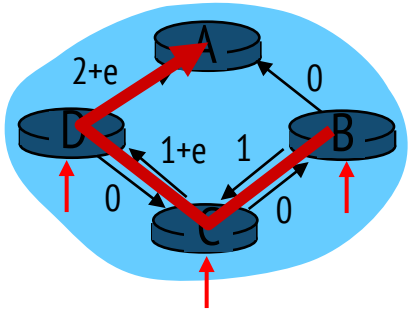

initially




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

*Execute the algorithm at same period, but start at different times.*

# Link State Routing

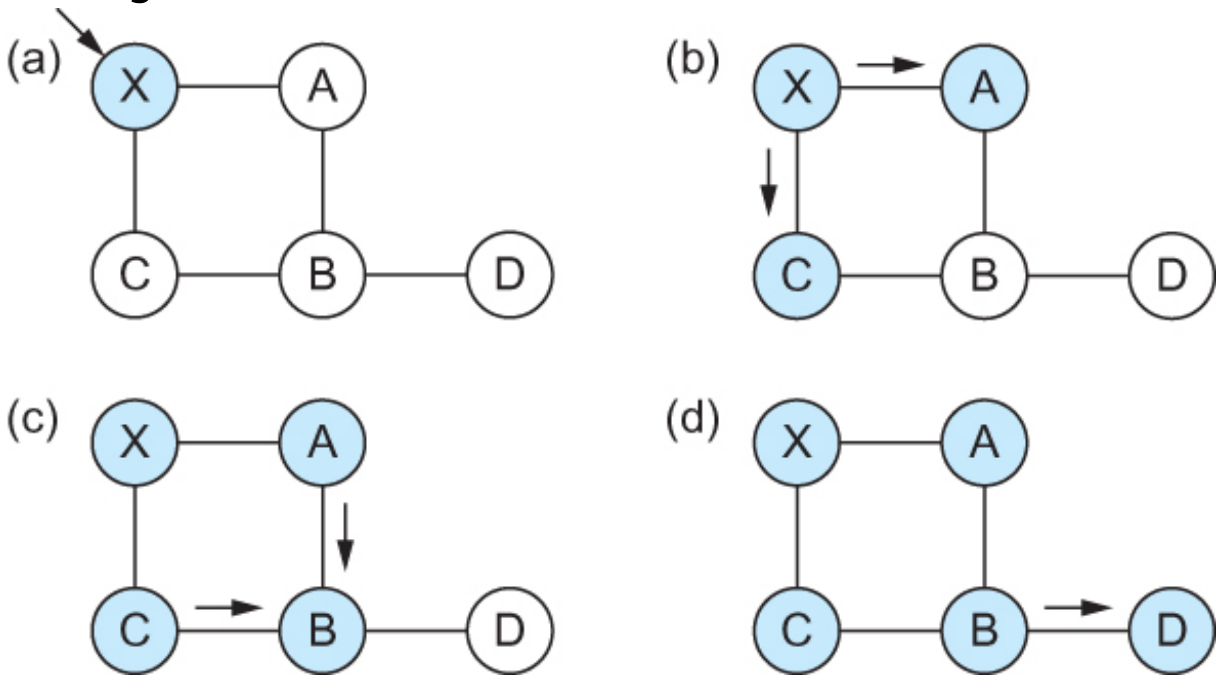
Strategy: Send to all nodes (not just neighbors) information about directly connected links (not entire routing table).

- Link State Packet (LSP)
  - id of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet

- Reliable Flooding
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - generate new LSP periodically (timer); increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP; discard when TTL=0

# Link State

## Reliable Flooding




Flooding of link-state packets.

(a) LSP arrives at node X;

(b) X floods LSP to A and C;

(c) A and C flood LSP to B (but not X);

(d) flooding is complete

# Distance Vector Routing Algorithm

# Distance vector algorithm

*Bellman-Ford equation (dynamic programming)*

let

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

then

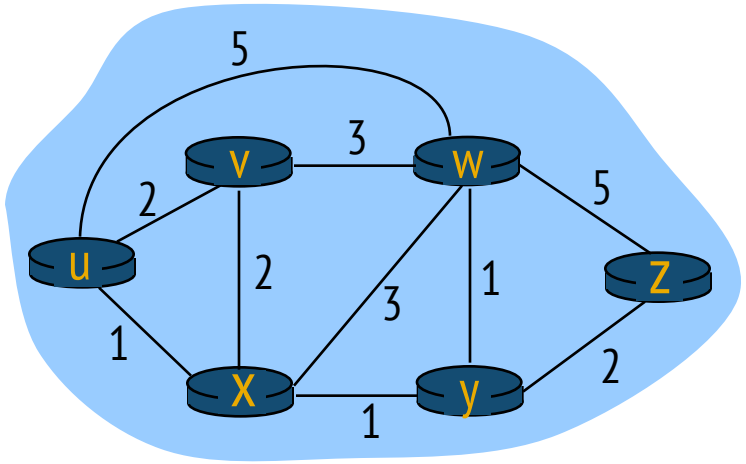
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

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:

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

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

# Distance vector algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- node x:
  - knows cost to each neighbor v: $c(x,v)$
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

# 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 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

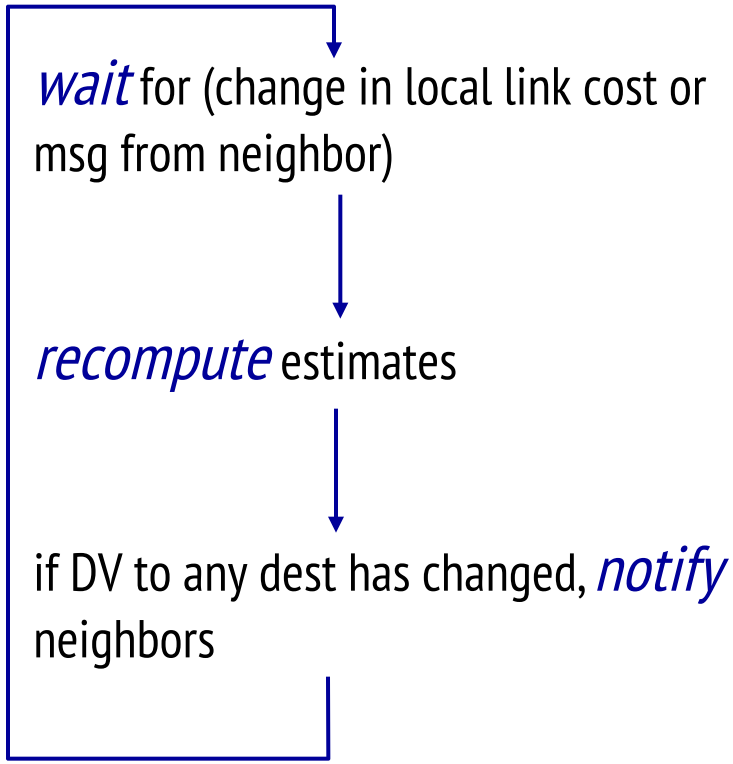
*iterative, asynchronous:* each local iteration caused by:

- local link cost change
- DV update message from neighbor

*distributed:*

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

*each node:*



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

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

**node x table**

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

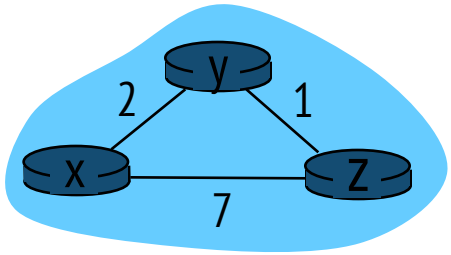
| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

**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 |

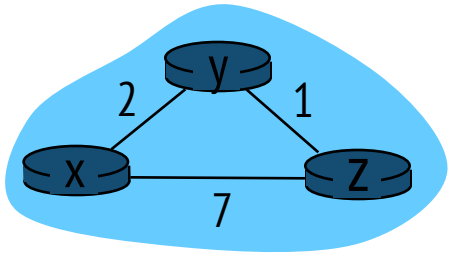
**node z table**

| 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 |

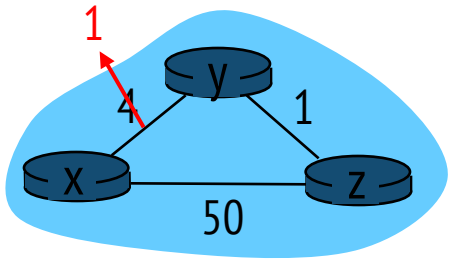
time



# Distance vector: link cost changes

*link cost changes:*

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors




"good news travels fast"

$t_0$: $y$ detects link-cost change, updates its DV, informs its neighbors.

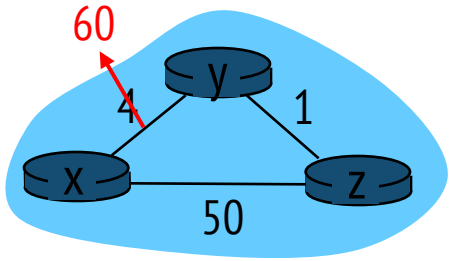
$t_1$: $z$ receives update from $y$, updates its table, computes new least cost to $x$, sends its neighbors its DV.

$t_2$: $y$ receives $z$'s update, updates its distance table. $y$'s least costs do *not* change, so $y$ does *not* send a message to $z$.

# Distance vector: link cost changes

*link cost changes:*

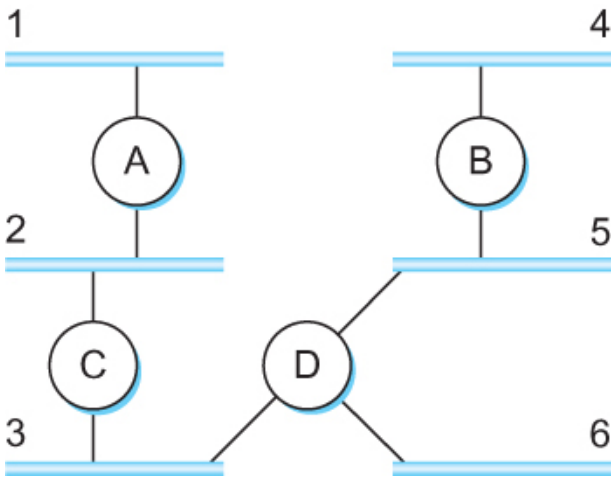
- node detects local link cost change
- *bad news travels slow* - "count to infinity" problem!
- 44 iterations before algorithm stabilizes



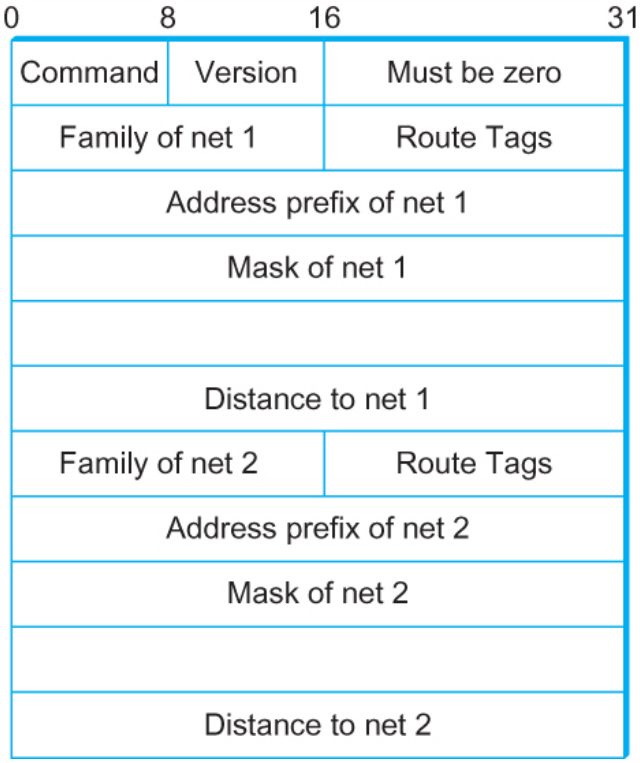

*poisoned reverse:*

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

# Routing Information Protocol (RIP)




Example Network running RIP




RIPv2 Packet Format

An example Distance Vector Protocol

# Comparison of LS and DV algorithms

## *message complexity*

- ***LS:*** with n nodes, E links, O(nE) msgs sent
- ***DV:*** exchange between neighbors only
  - convergence time varies

## *speed of convergence*

- ***LS:*** $O(n^2)$ algorithm requires O(nE) msgs
  - may have oscillations
- ***DV:*** convergence time varies
  - may be routing loops
  - count-to-infinity problem

## *robustness:* what happens if router malfunctions?

***LS:***

- node can advertise incorrect *link* cost
- each node computes only its *own* table

***DV:***

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagates thru network

# Making routing scalable

our routing study thus far - idealized

- all routers identical
- network "flat"

… *not* true in practice

*scale:* with billions of destinations:

- can't store all destinations 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

# Internet approach to scalable routing

aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")

## intra-AS routing

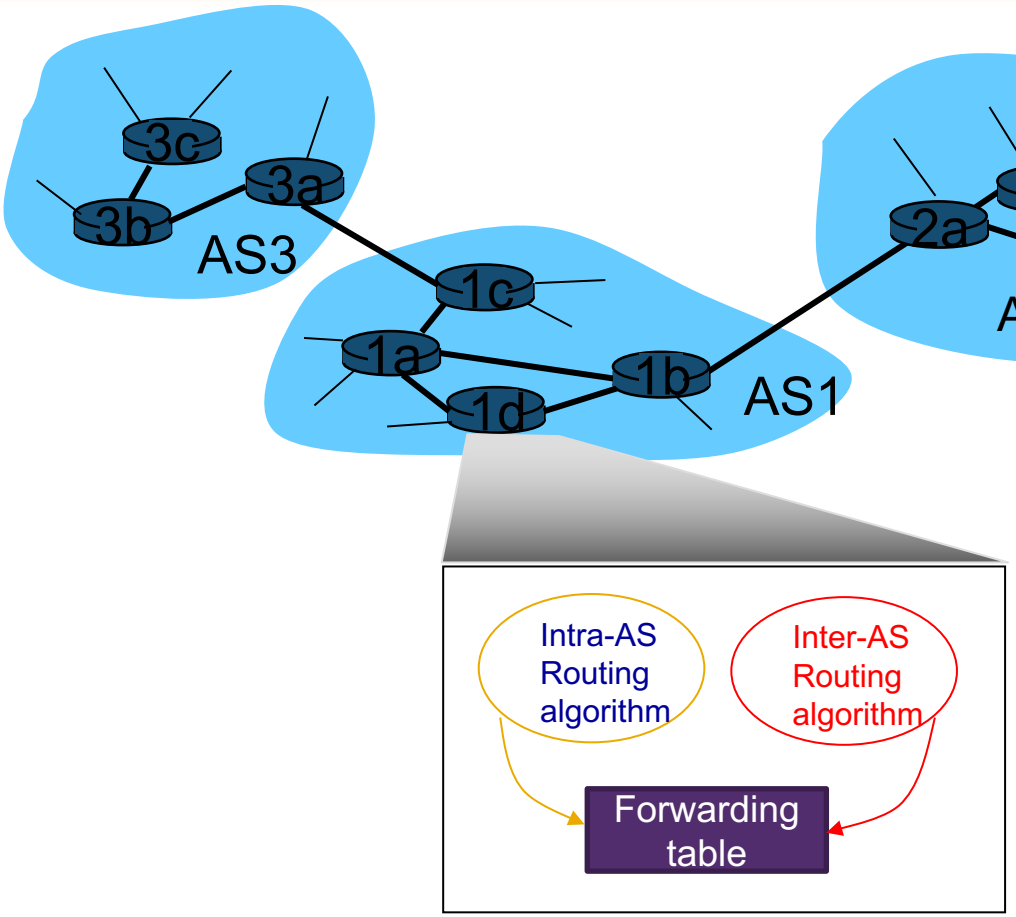
- routing among hosts, routers in same AS ("network")
- all routers in AS must run *same* intra-domain protocol
- routers in *different* AS can run *different* intra-domain routing protocol
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

## inter-AS routing

routing among AS'es

gateways perform inter-domain routing (as well as intra-domain 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 & intra-AS determine entries for external destinations

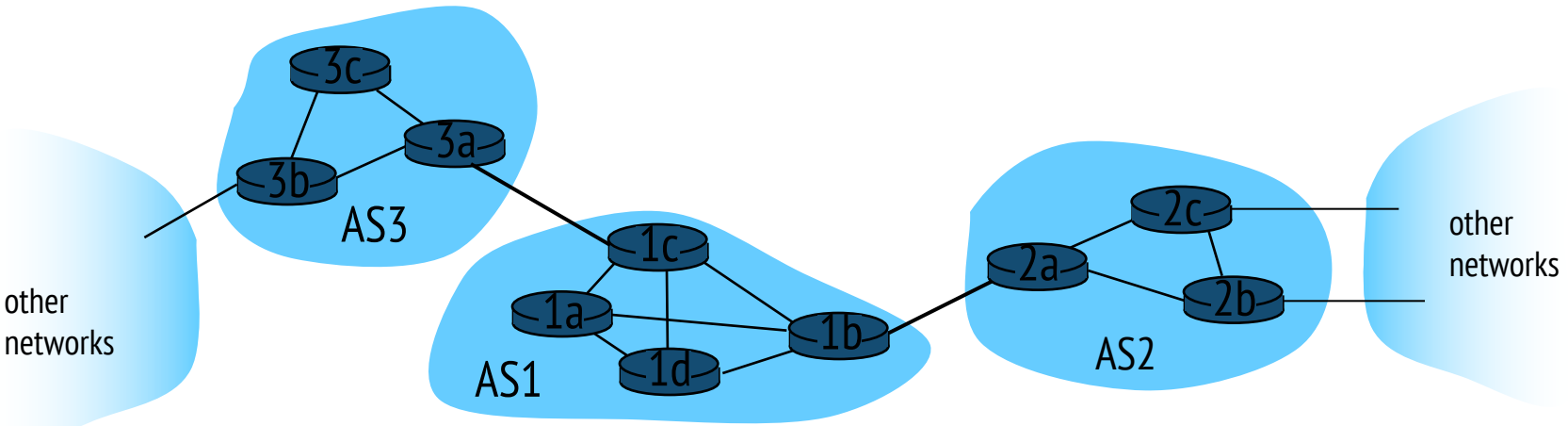
# Inter-AS tasks

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router, but which one?

*AS1 must:*

1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

*job of inter-AS routing!*



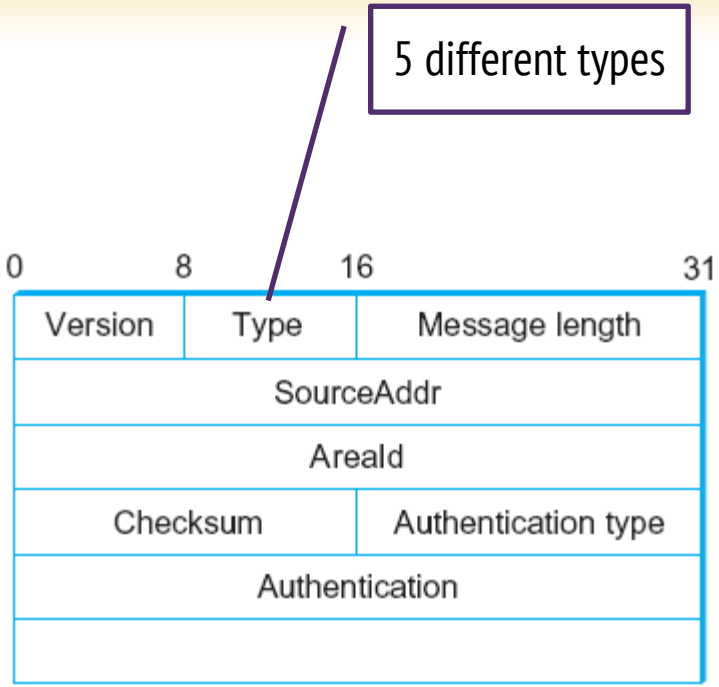
# Intra-AS Routing

- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF)
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)

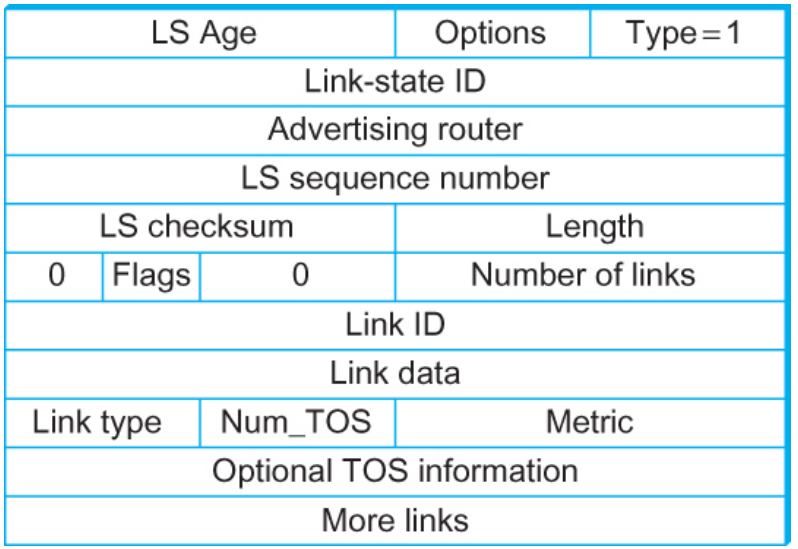
# OSPF (Open Shortest Path First)

- "open": publicly available
- uses link-state algorithm
  - link state packet dissemination
  - topology map at each node
  - route computation using Dijkstra's 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)
  - link state: for each attached link
- *IS-IS routing* protocol: nearly identical to OSPF

# Open Shortest Path First (OSPF)




OSPF Header Format

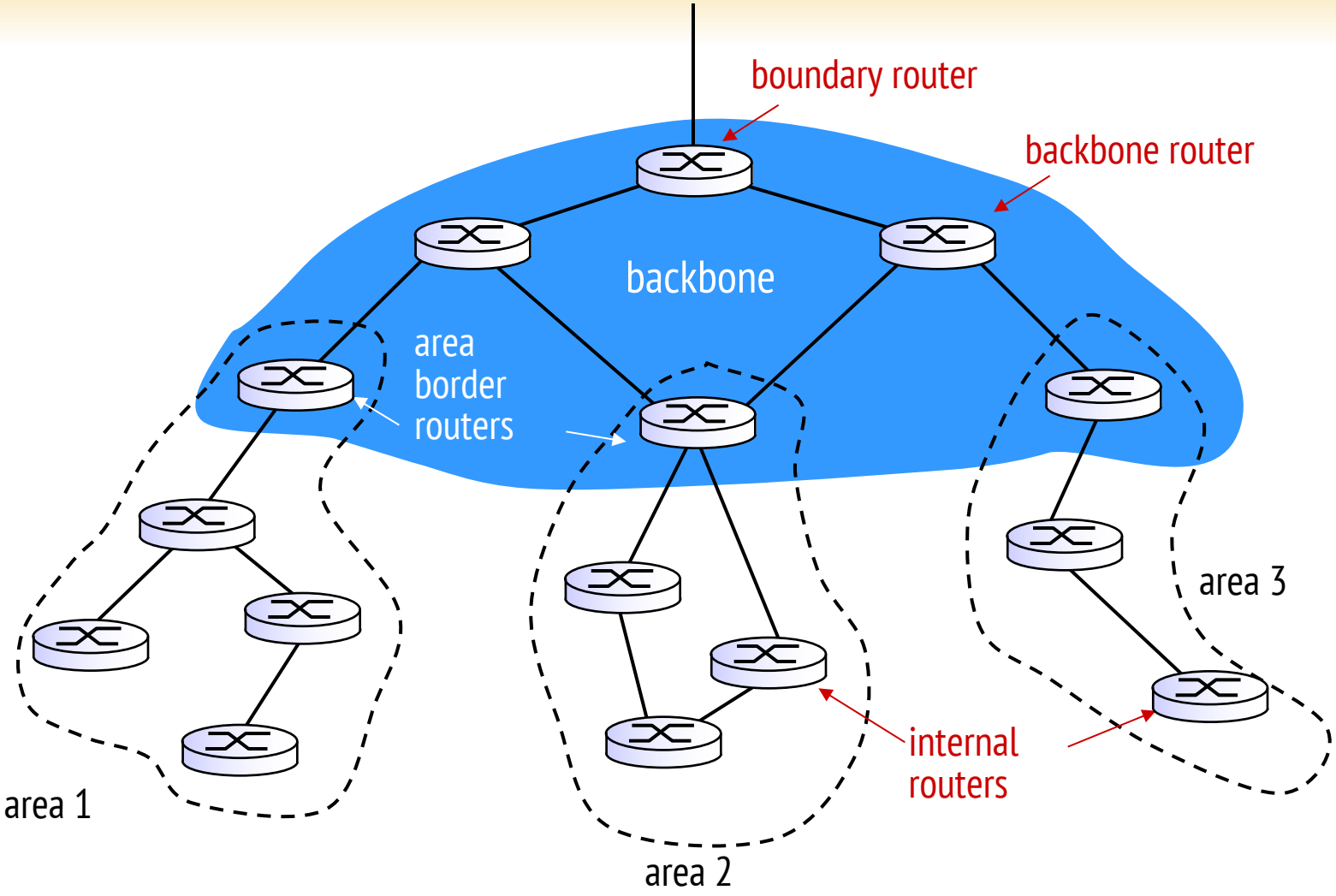



OSPF Link State Advertisement

# OSPF "advanced" features

- *security:* all OSPF messages authenticated (to prevent malicious intrusion)
- multiple same-cost paths allowed (only one path in RIP)
- for each link, multiple cost metrics for different TOS (e.g., satellite link cost (more delay involved) set low for best effort ToS; high for real-time ToS)
- integrated uni- and multi-cast support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- hierarchical OSPF in large domains.

# Hierarchical OSPF



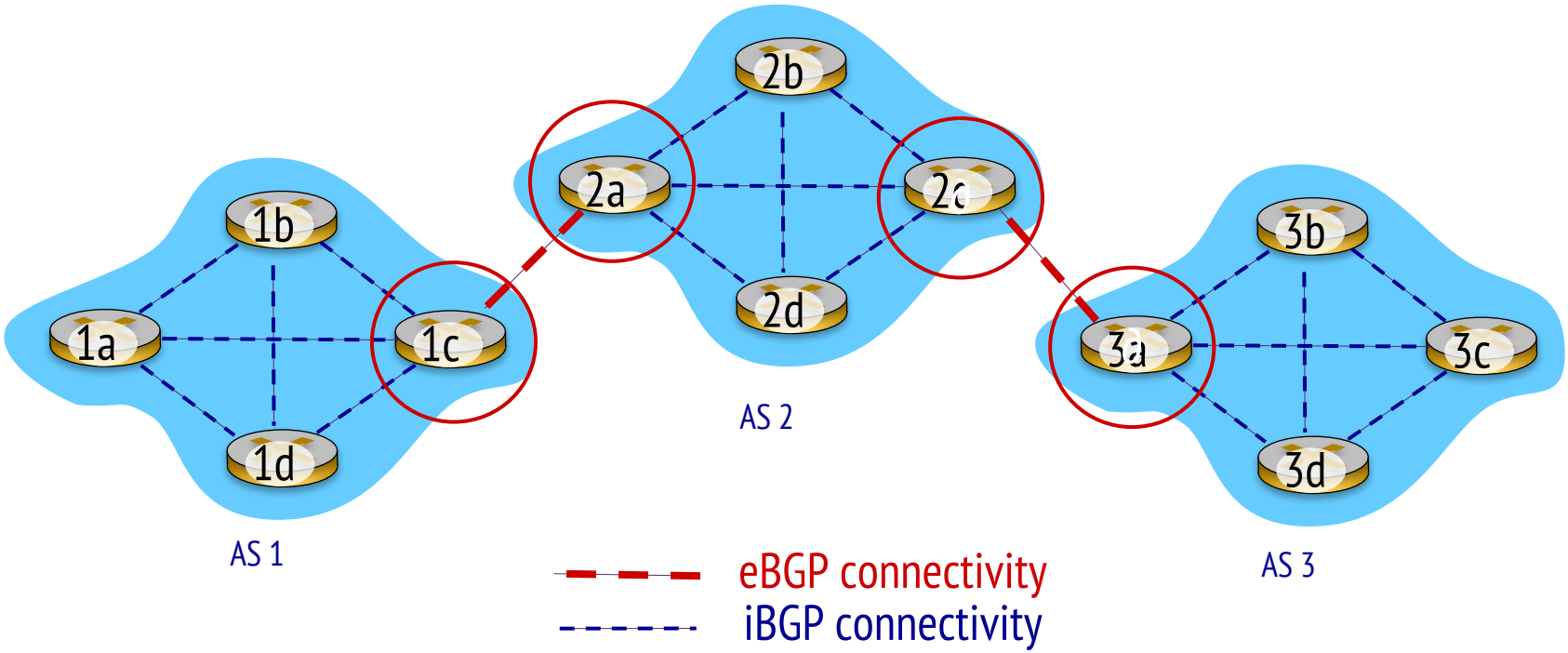
# Hierarchical OSPF

- *two-level hierarchy:* local area, backbone.
  - link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- *area border routers:* "summarize" distances to nets in own area, advertise to other Area Border routers.
- *backbone routers:* run OSPF routing limited to backbone.
- *boundary routers:* connect to other AS'es.

# 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 ASes
  - 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"*

# eBGP, iBGP connections

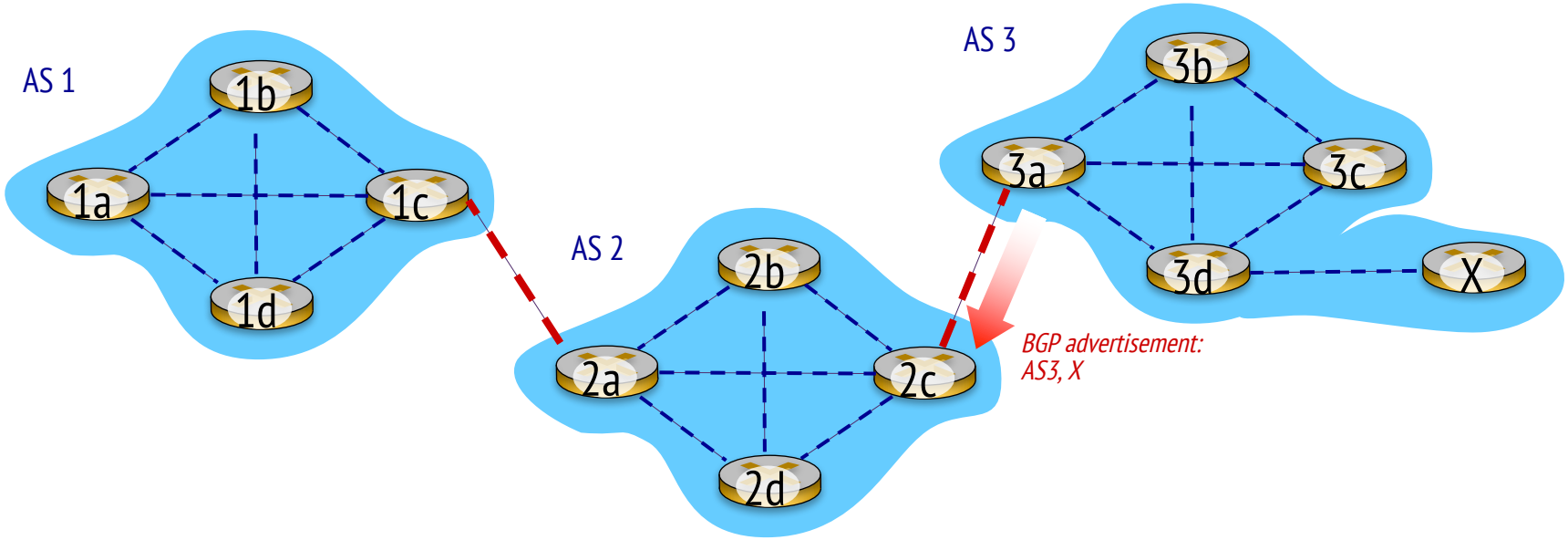




gateway routers run both eBGP and iBGP protools

# BGP basics

- BGP session: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection (port 179):
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)

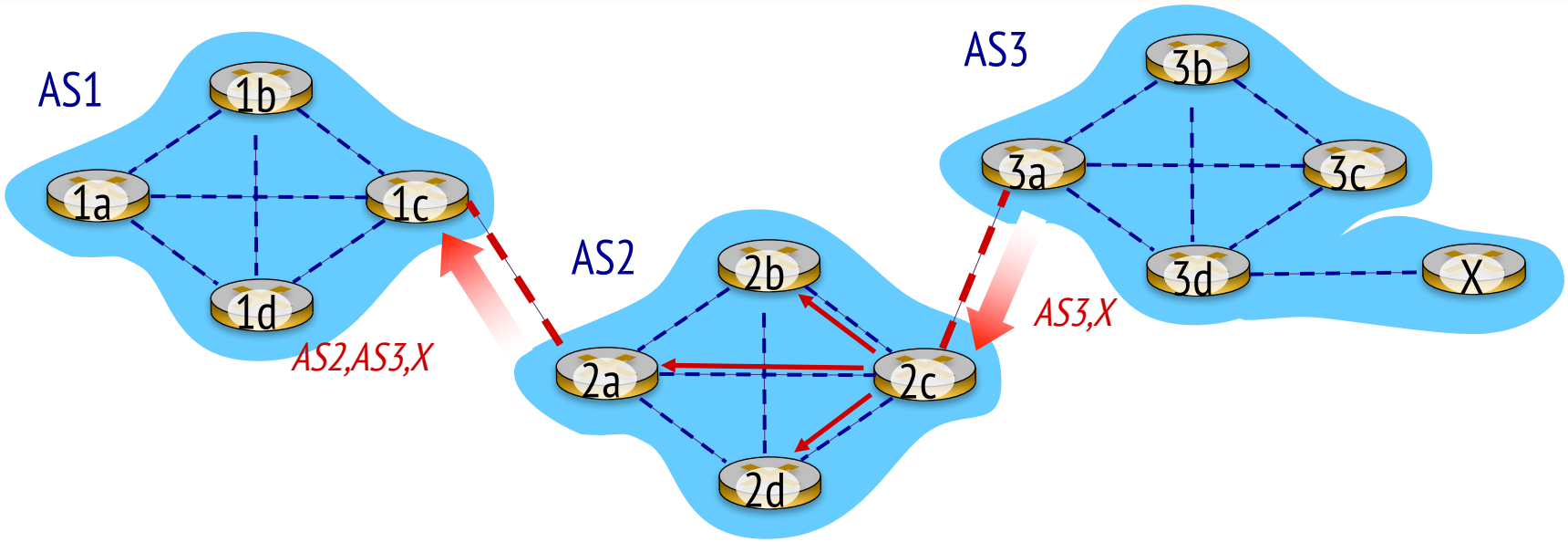



- when AS3 gateway router 3a advertises path AS3,X to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X

# Path attributes and BGP routes

- advertised prefix includes BGP attributes
  - 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 *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

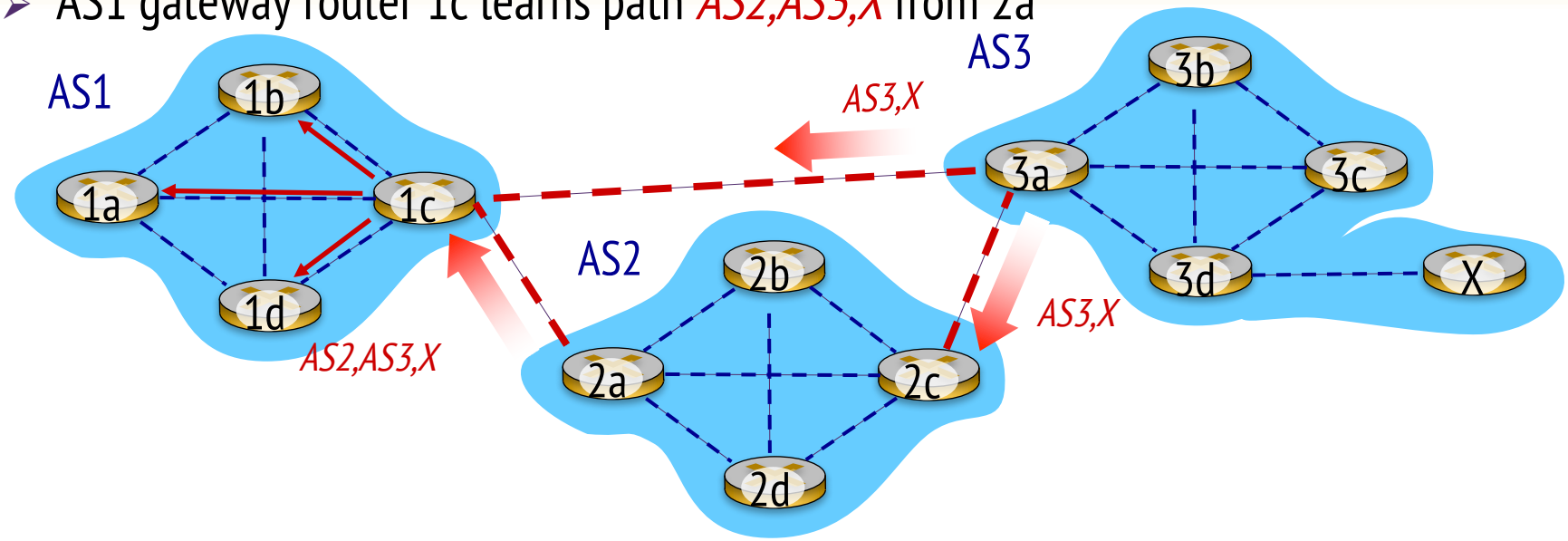
# BGP path advertisement




- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
  - Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c

# BGP path advertisement

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a




gateway router may learn about multiple paths to destination:

- 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*

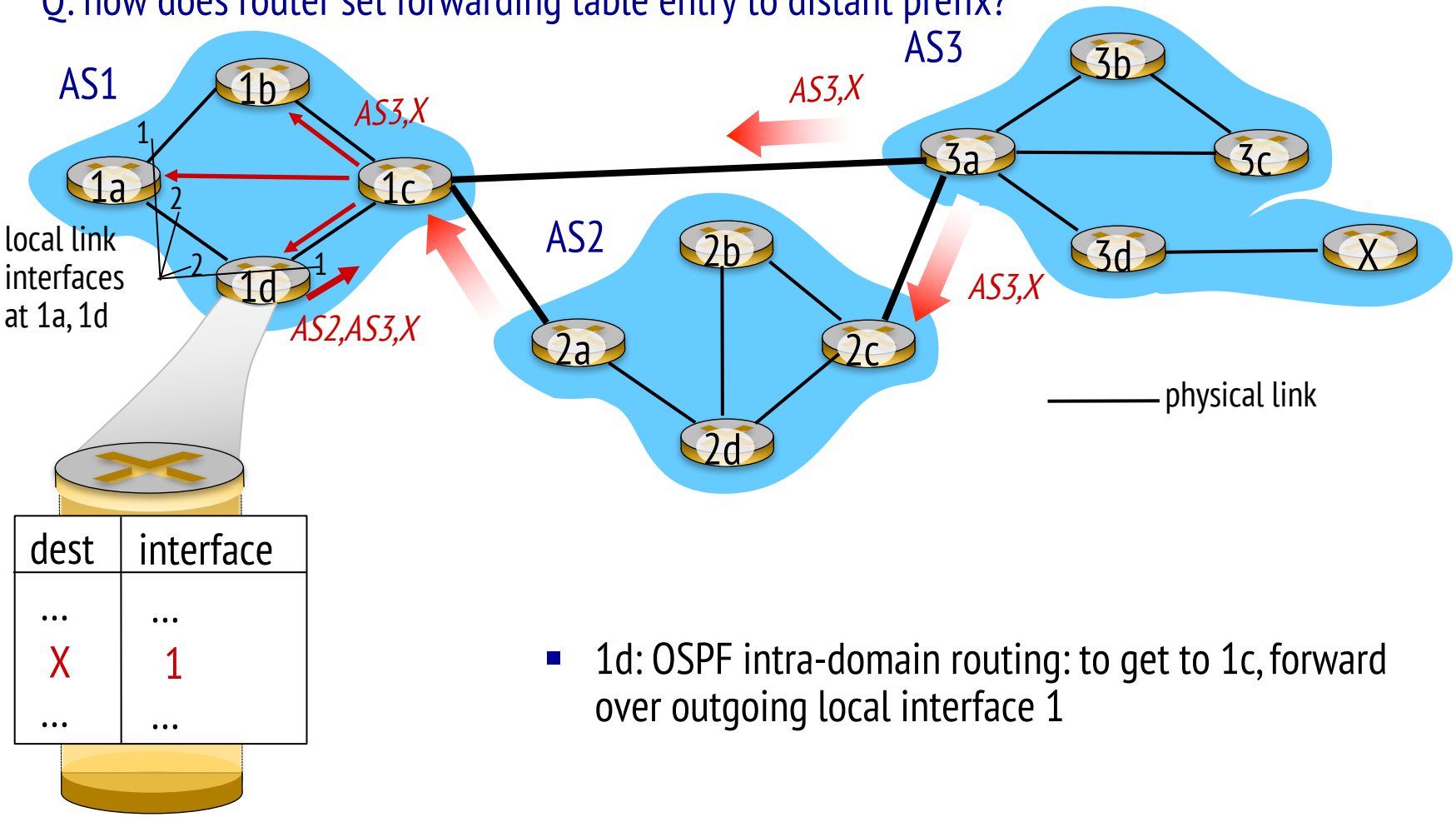
# BGP messages

- 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

- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: "path to X goes through 1c"

Q: how does router set forwarding table entry to distant prefix?

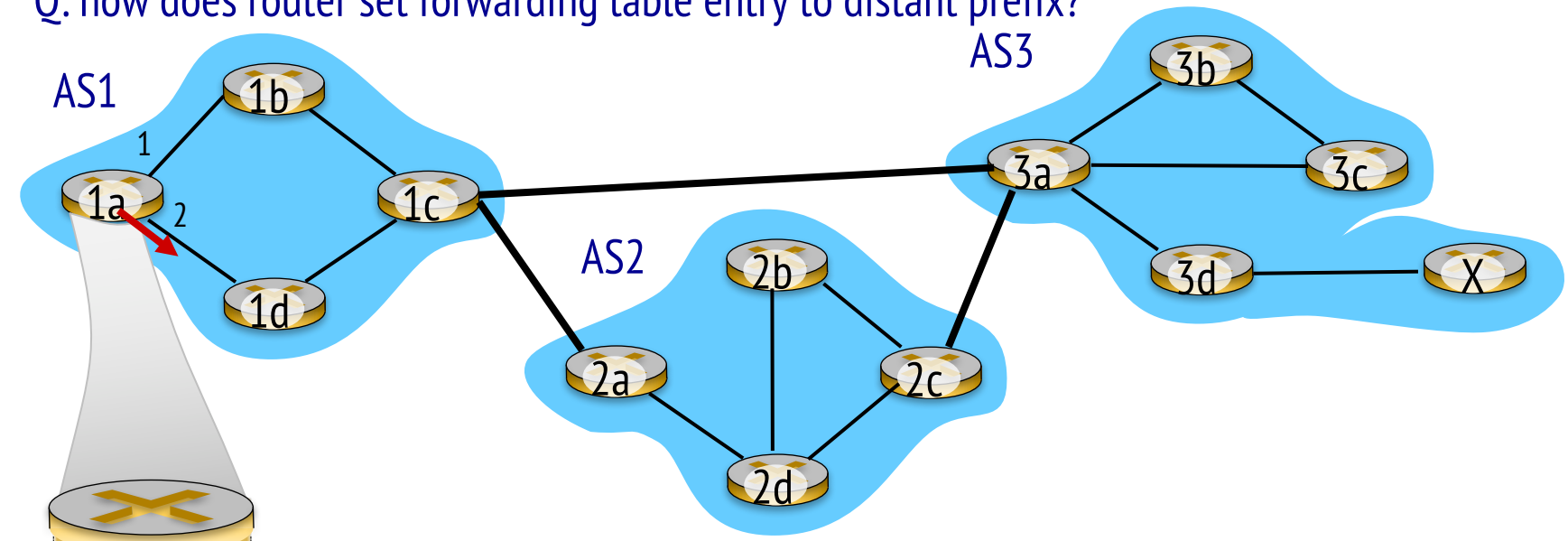



- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1

# BGP, OSPF, forwarding table entries

- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: "path to X goes through 1c"

Q: how does router set forwarding table entry to distant prefix?



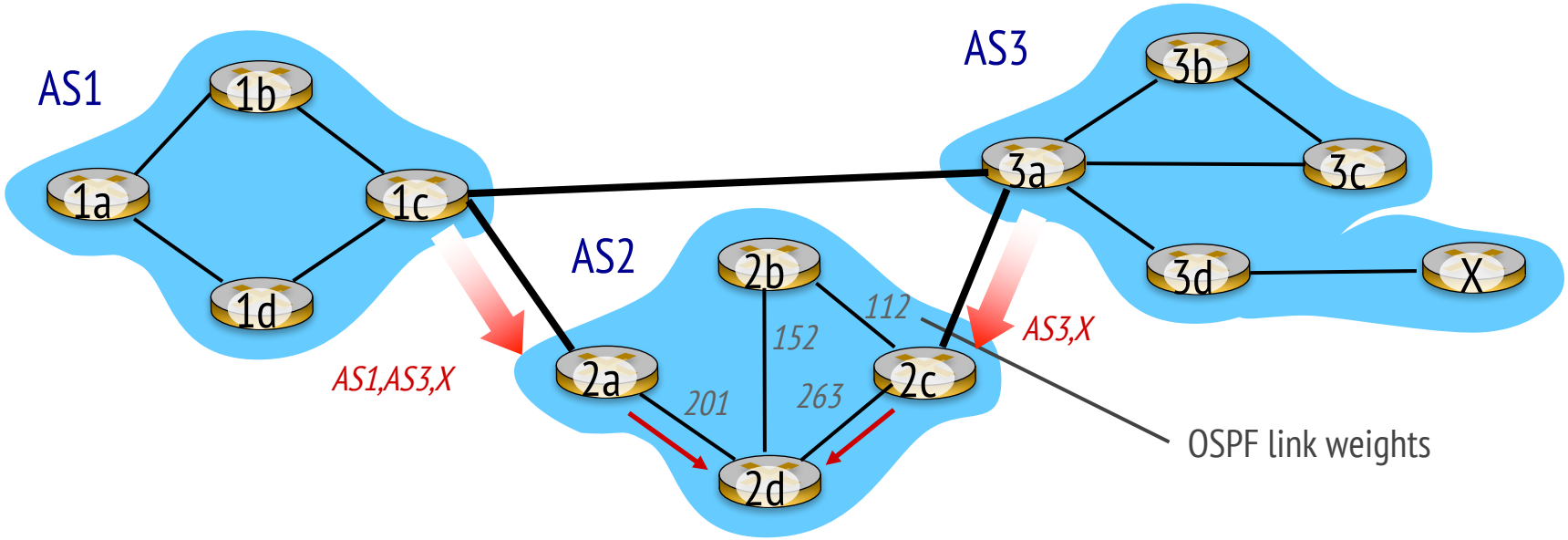

| dest | interface |
|---|---|
| ... | ... |
| X | 2 |
| ... | ... |

- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2

# BGP route selection

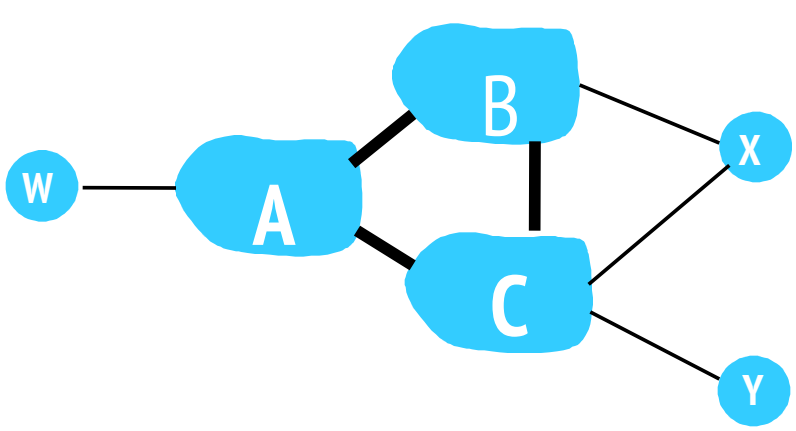
- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

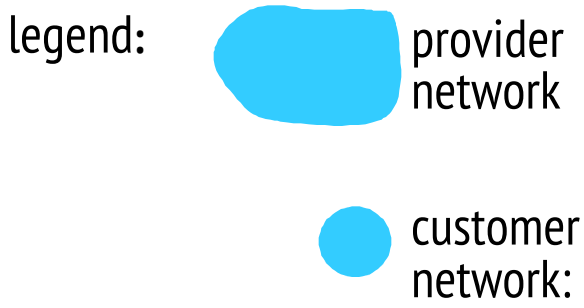
# Hot Potato Routing




- 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: achieving policy via advertisements

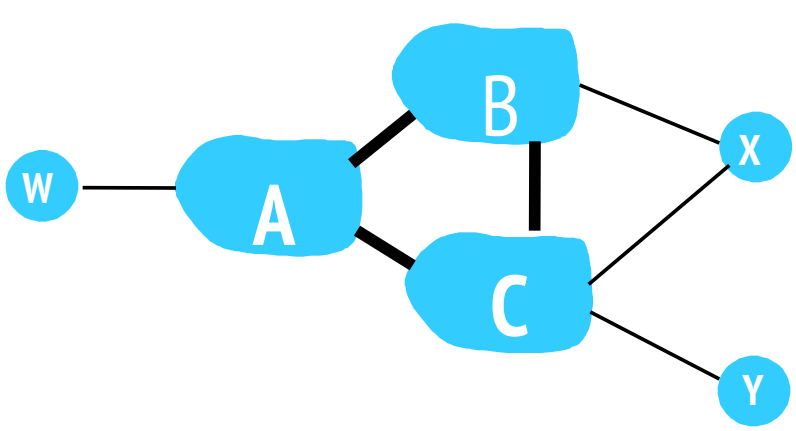


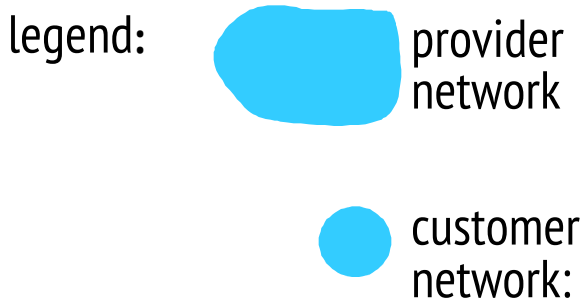




Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- 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

# 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)

- A,B,C are *provider networks*
- X,W,Y are customer (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

# Why different Intra-, 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:*

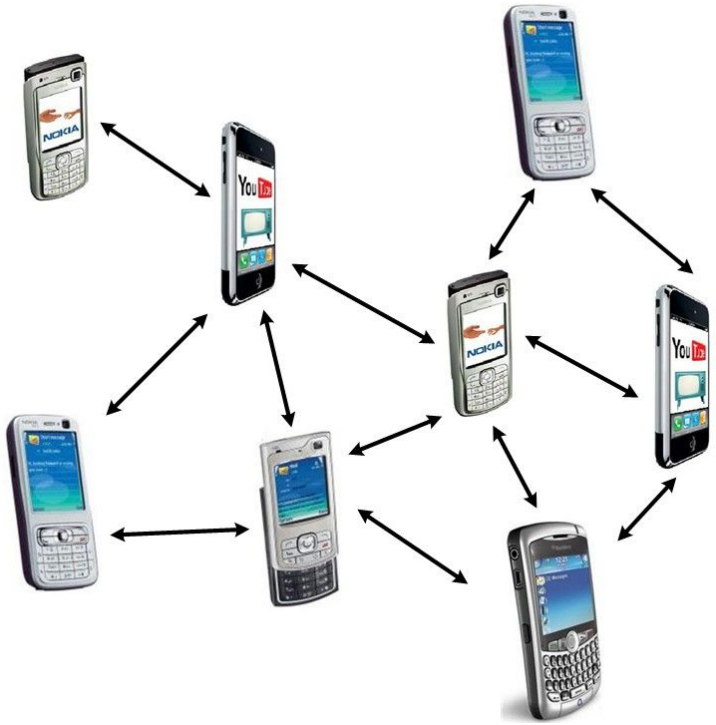
- hierarchical routing saves table size, reduced update traffic

*performance:*

- intra-AS: can focus on performance
- inter-AS: policy may dominate over performance

# Mesh Network / MANET (Mobile Ad Hoc Network)

- Mobile nodes, wireless links
- Infrastructure-less: by the nodes
- Multi-hop routing
- Example scenarios for MANETs
  - Meetings
  - Emergency or disaster relief situations
  - Military communications
  - Wearables
  - Sensor networks



# What's unique about MANET?

- Moving nodes → ever changing topology
- Wireless links
  - → various and volatile link quality
- Pervasive (cheap) devices
  - → Power constraints
- Security
  - Confidentiality, other attacks

# Challenges

- Need dynamic routing
  - Frequent topological changes possible.
  - Very different from dynamic routing in the Internet.
  - Potential of network partitions.

- Routing overhead must be kept minimal
  - Wireless → low bandwidth
  - Mobile → low power
  - Minimize # of routing control messages
  - Minimize routing state at each node

# Routing Protocols

- Reactive (On-demand) protocols
  - Discover routes when needed
  - Source-initiated route discovery
- Proactive protocols
  - Traditional distributed shortest-path protocols
  - Based on periodic updates. High routing overhead
- Tradeoff
  - State maintenance traffic vs. route discovery traffic
  - Route via maintained route vs. delay for route discovery

# Reactive Routing

- Key Goal: Reduction in routing overhead
  - Useful when number of traffic sessions is much lower than the number of nodes.
- No routing structure created *a priori*. Let the structure emerge in response to a need
- Two key methods for route discovery
  - source routing
  - backward learning (similar to intra-AS routing)
- Introduces delay

# Reactive (on demand) routing

Routing only when needed

Advantages:

- eliminate periodic updates
- adaptive to network dynamics

Disadvantages:

- high flood-search overhead with
  - mobility, distributed traffic
- high route acquisition latency

# Reactive Routing – Source Initiated

- Source floods the network with a *route request* packet when a route is required to a destination
  - Flood is propagated outwards from the source
  - Pure flooding = every node transmits the request only once

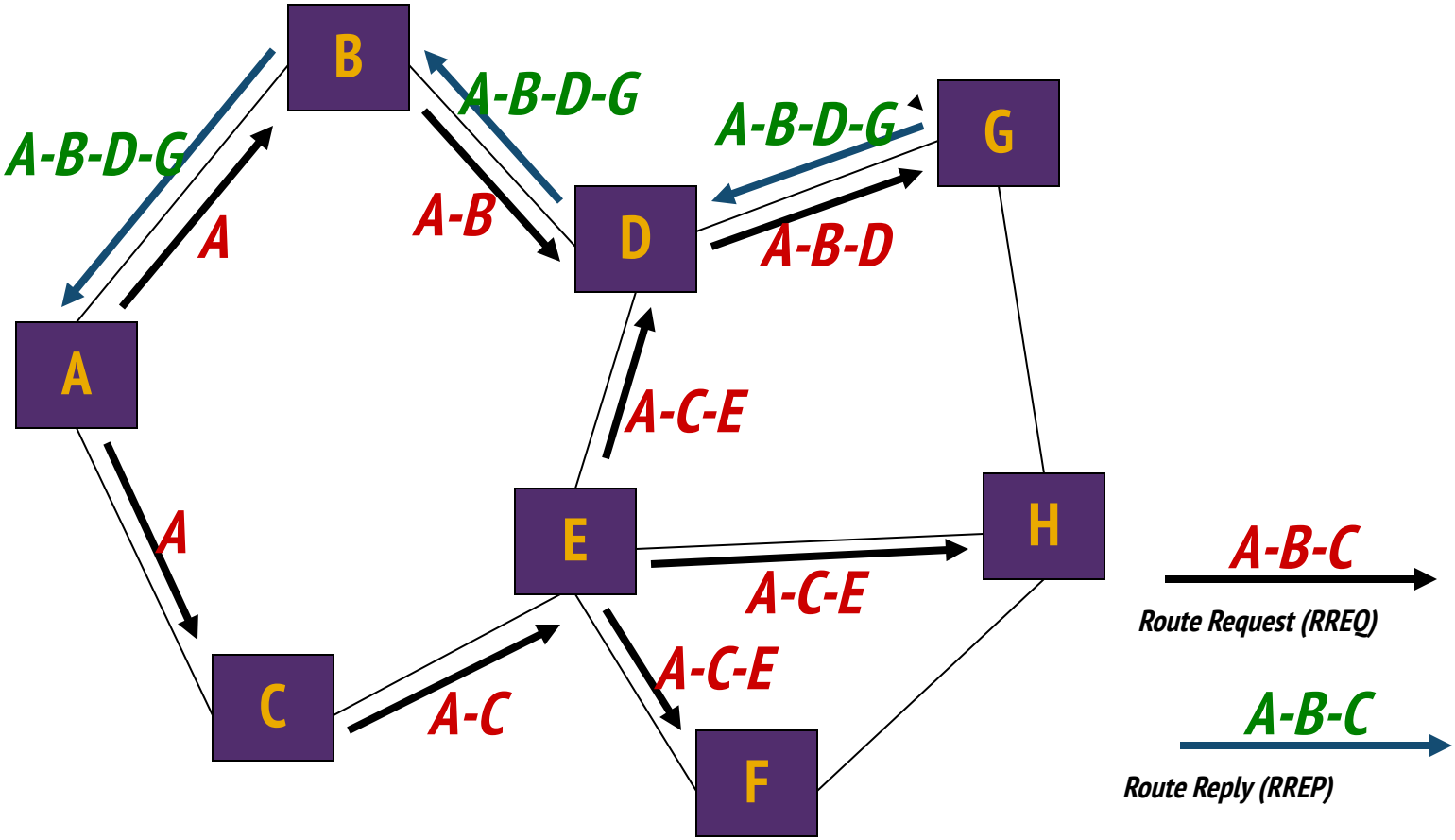
- Destination *replies* to request
  - Reply uses reversed path of route request
  - sets up the forward path

- Two key protocols: DSR and AODV

# Dynamic Source Routing (DSR)

- Cooperative nodes
- Relatively small network diameter (5-10 hops)
- Detectable packet error
- Unidirectional or bidirectional link
- Promiscuous mode (optional)

# Route Discovery



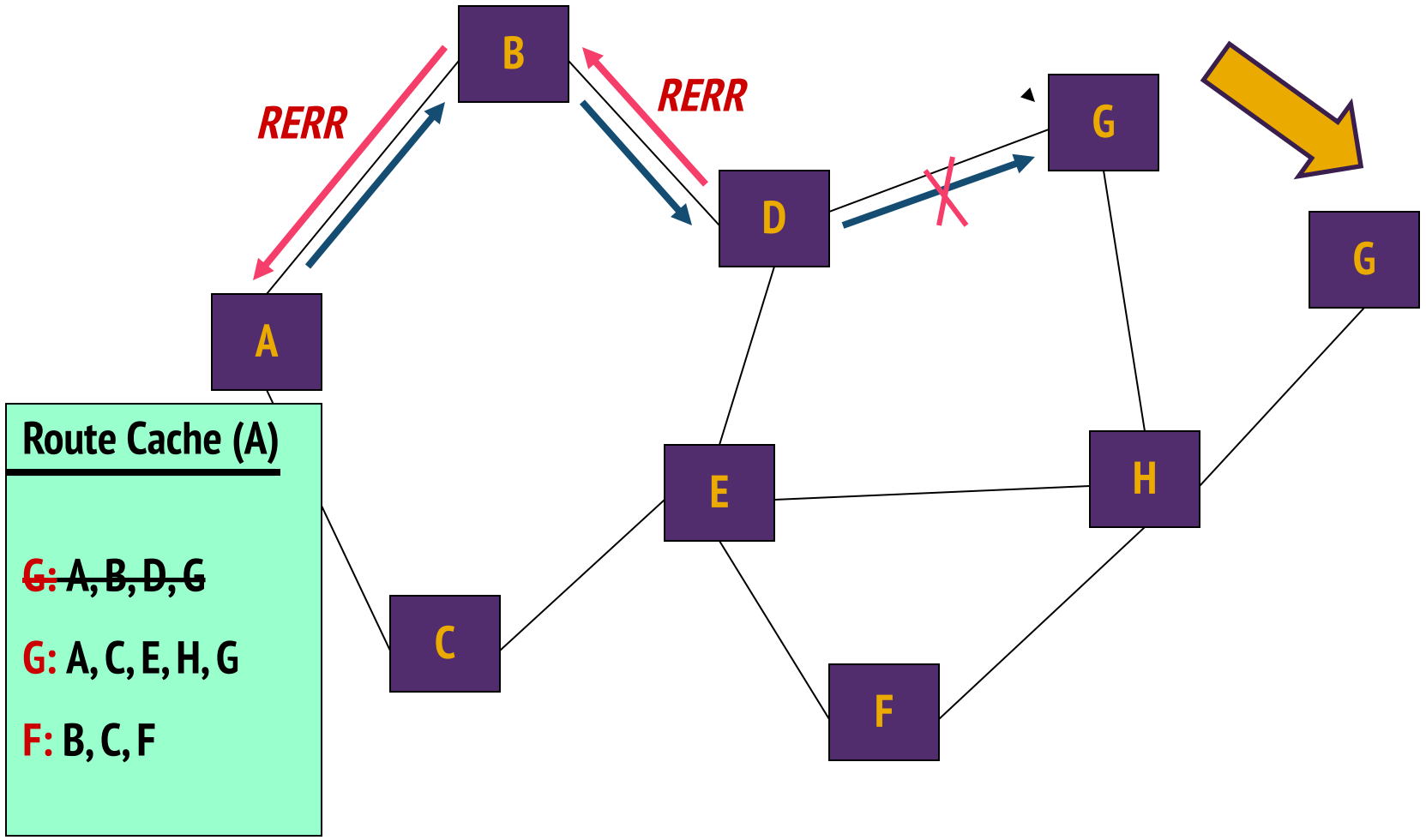
# DSR – Route Discovery

- *Route Reply* message containing path information is sent back to the source either by
  - the destination, or
  - intermediate nodes that have a route to the destination
  - Reverse the order of the route record, and include it in Route Reply.
  - Unicast, source routing
- Each node maintains a *Route Cache* which records routes it has learned and overheard over time

# Route Maintenance

- Route maintenance performed only while route is in use
- Error detection:
  - Monitors the validity of existing routes by *passively* listening to data packets transmitted at neighboring nodes
  - Lower level acknowledgements
- When problem detected, send *Route Error* packet to original sender to perform new route discovery
  - Host detects the error and the host it was attempting;
  - *Route Error* is sent back to the sender the packet – original src

# Route Maintenance



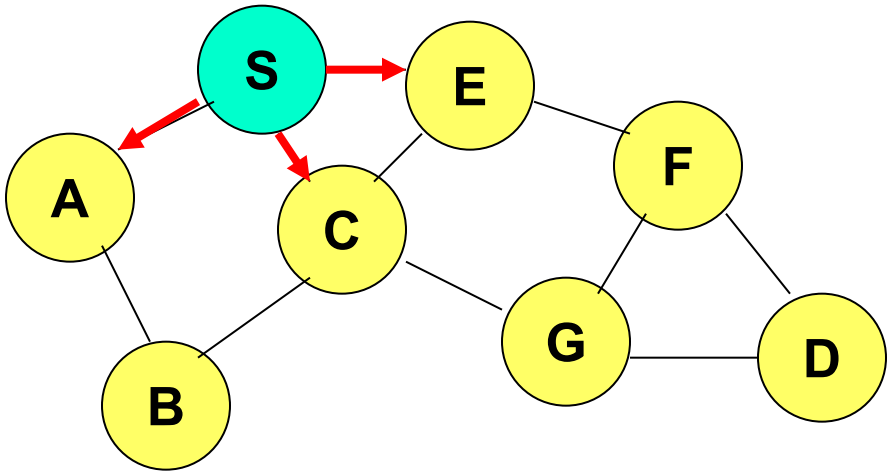
# DSR Summary

- Pros:
  - On-demand, potentially zero control message overhead
  - Trivially loop-free with source routing
  - Supports unidirectional and bidirectional links

- Cons:
  - High packet delays/jitters associated with on-demand routing
  - Space overhead in packets and route caches
  - Promiscuous mode operations consume excessive amount of power

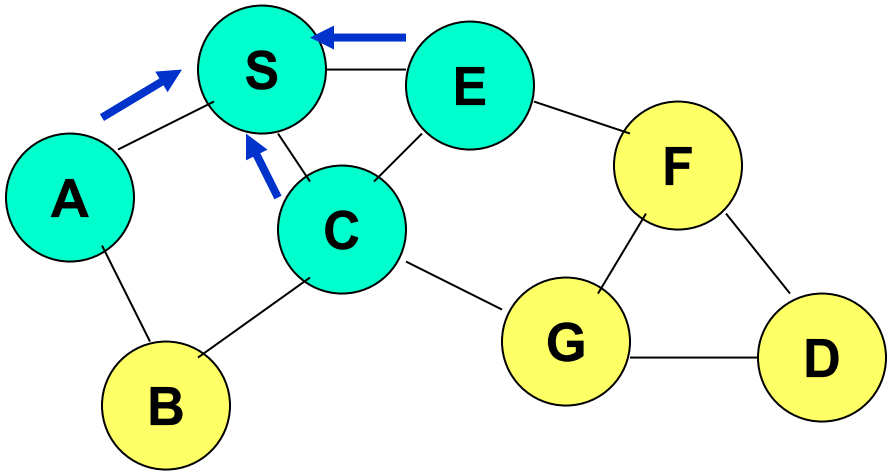
# AODV Routing Protocol

- AODV = Ad Hoc On-demand Distance Vector
- Source floods route request in the network.
- Reverse paths are formed when a node hears a route request.
- Each node forwards the request only once (pure flooding).



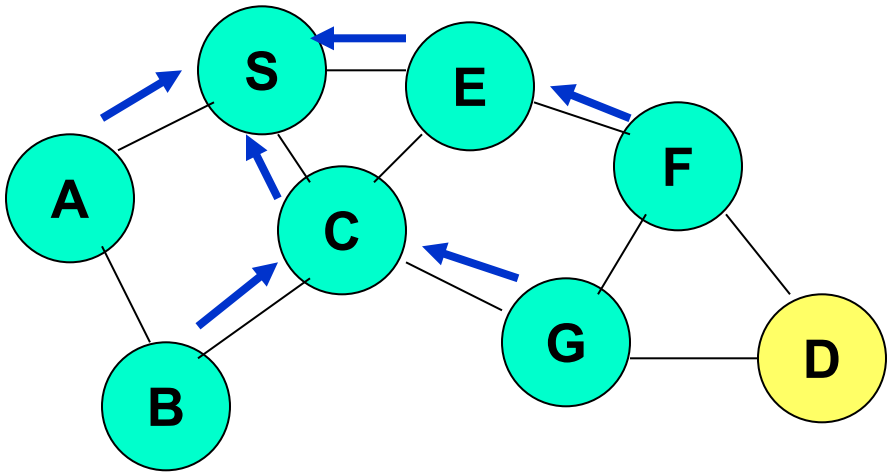
# AODV Route Discovery

- Source floods route request in the network.
- Each node forwards the request only once (pure flooding).



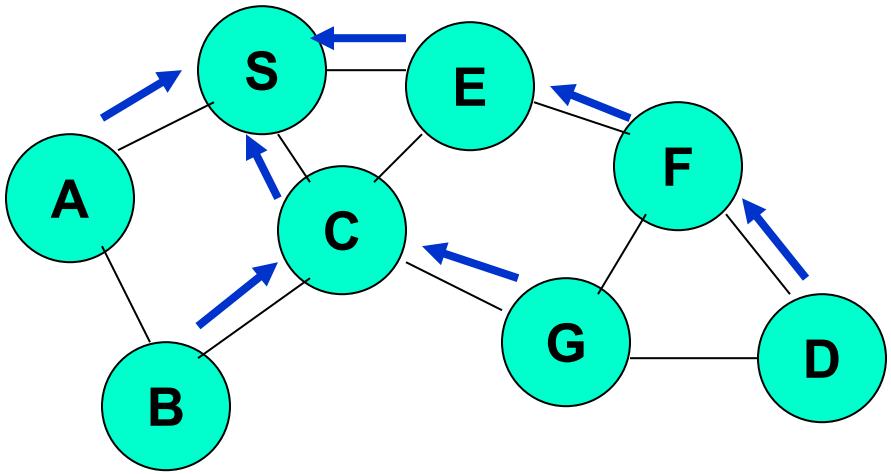
# AODV Route Discovery

- Uses hop-by-hop routing.
- Each node forwards the request only once (pure flooding).
- Reverse paths are formed when a node hears a route request.



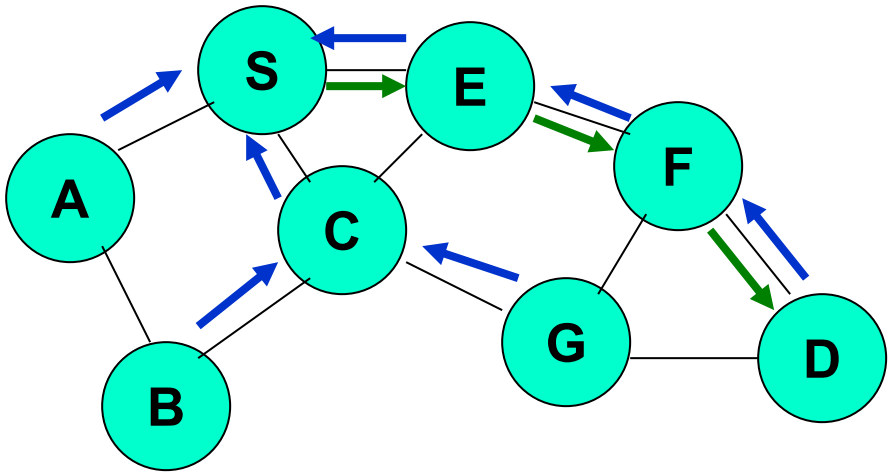
# AODV Route Discovery

- Route Reply forwarded via the reverse path



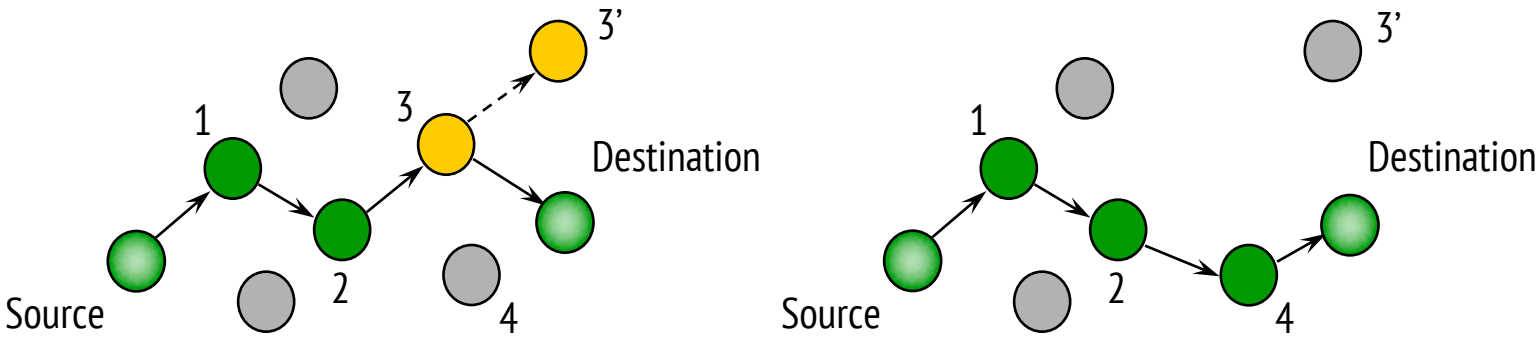
# AODV Route Discovery

- Route reply is forwarded via the reverse path … thus forming the forward path.
- The forward path is used to route data packets.



# AODV Route Maintenance / Expiry

- Unused paths expire based on a timer.
- Movement not along active path triggers no action
  - If source moves, reinitiate route discovery
- When destination or intermediate node moves
  - upstream node of break broadcasts *Route Error* (RERR)
  - RERR contains list of all destinations no longer reachable due to link break
  - RERR propagated until node with no precursors for destination is reached



# AODV Summary

- At most one route per destination maintained at each node
  - After link break, all routes using the failed link are erased.
- Expiration based on timeouts.
- Use of sequence numbers to prevent loops.
- Optimizations
  - Routing tables instead of storing full routes.
  - Control flooding (incrementally increase 'region')

# Few MANET Routing Protocols

- Dynamic Source Routing (DSR)
- Associativity-Based Routing (ABR)
- Ad-hoc On-demand Distance Vector (AODV)
- Temporarily Ordered Routing Algorithm (TORA)
- Zone Routing Protocol (ZRP)
- Signal Stability Based Adaptive Routing (SSA)
- On Demand Multicast Routing Protocol (ODMRP)

# Summary

- IP
  - Datagram, Fragmentation, IPv4, IPv6
- Router Architecture
- Routing
  - Link State
  - Distance Vector
  - Intra- and Inter-AS Routing
  - Routing in MANET