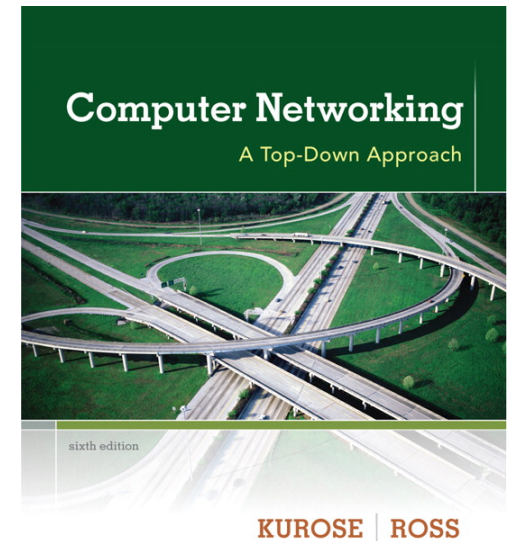


# Chapter 4

## Network Layer

Reti degli Elaboratori  
Canale ALProf.ssa Chiara  
Petrioli  
a.a. 2019/2020

We thank for the support material Prof. Kurose-Ross  
All material copyright 1996-2012  
© J.F Kurose and K.W. Ross, All Rights Reserved



*Computer  
Networking: A Top  
Down Approach*  
6<sup>th</sup> edition  
Jim Kurose, Keith Ross  
Addison-Wesley  
March 2012

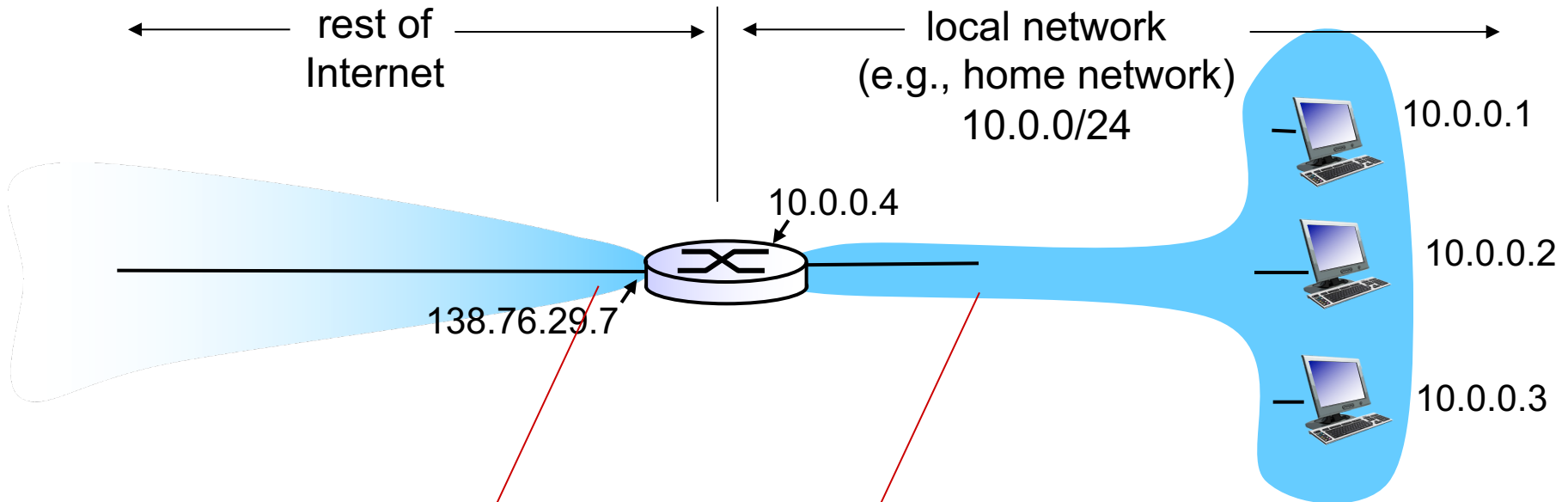
# IP addressing: the last word...

**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:

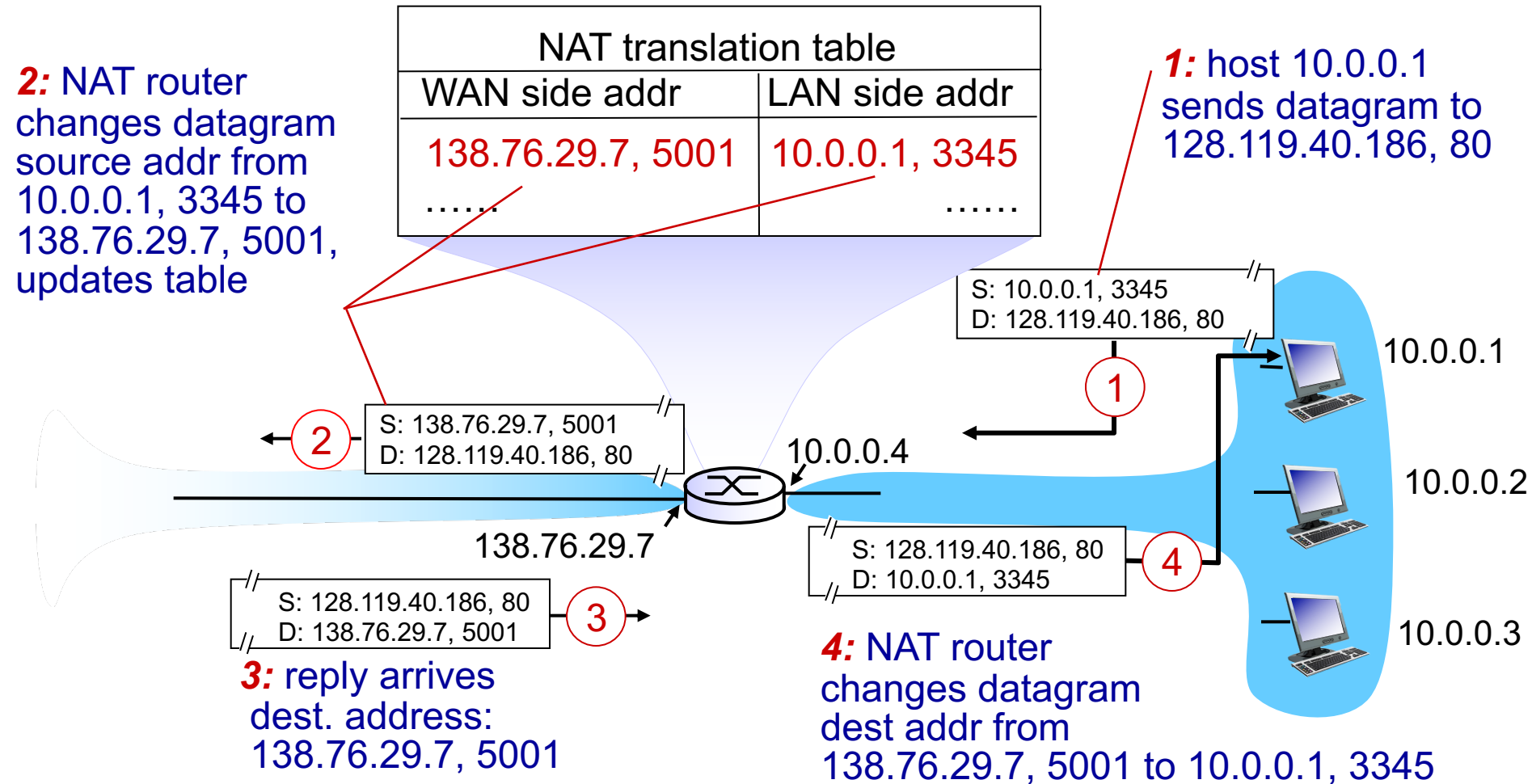
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside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

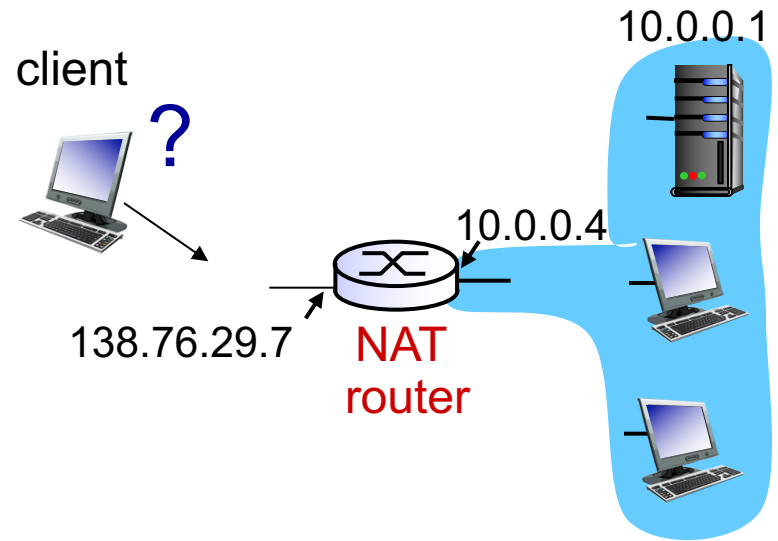


# NAT: network address translation

- ❖ 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
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - address shortage should instead be solved by IPv6
  - what if hosts behind the NAT are servers?

# 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
- ❖ **solution 1:** statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



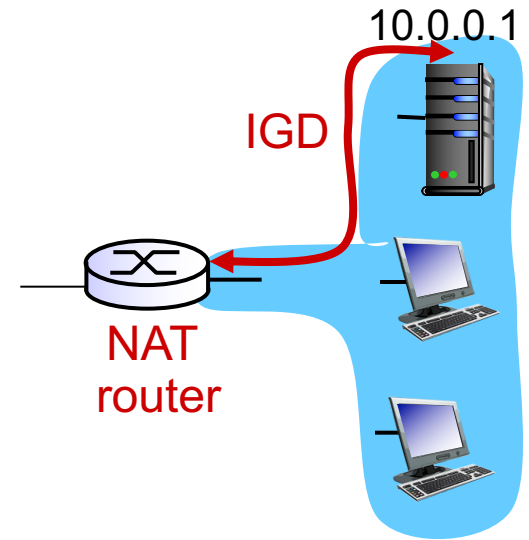


# NAT traversal problem

❖ *solution 2*: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:

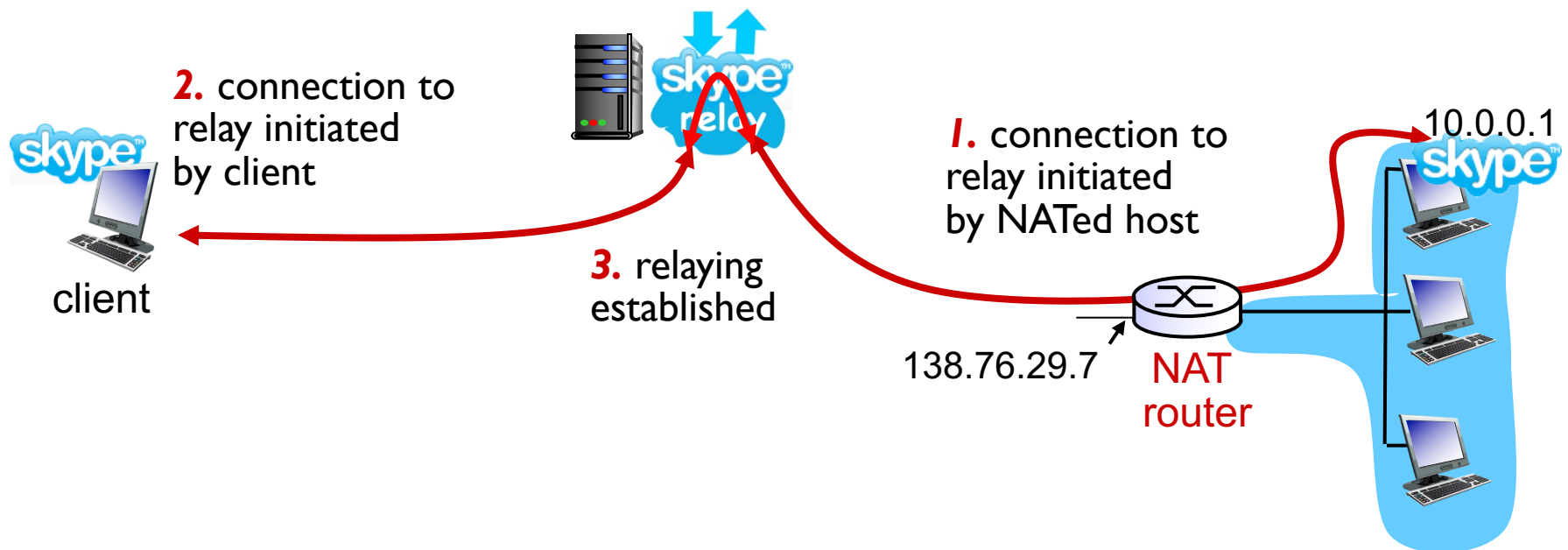
- ❖ learn public IP address (138.76.29.7)
- ❖ add/remove port mappings (with lease times)

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

# 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

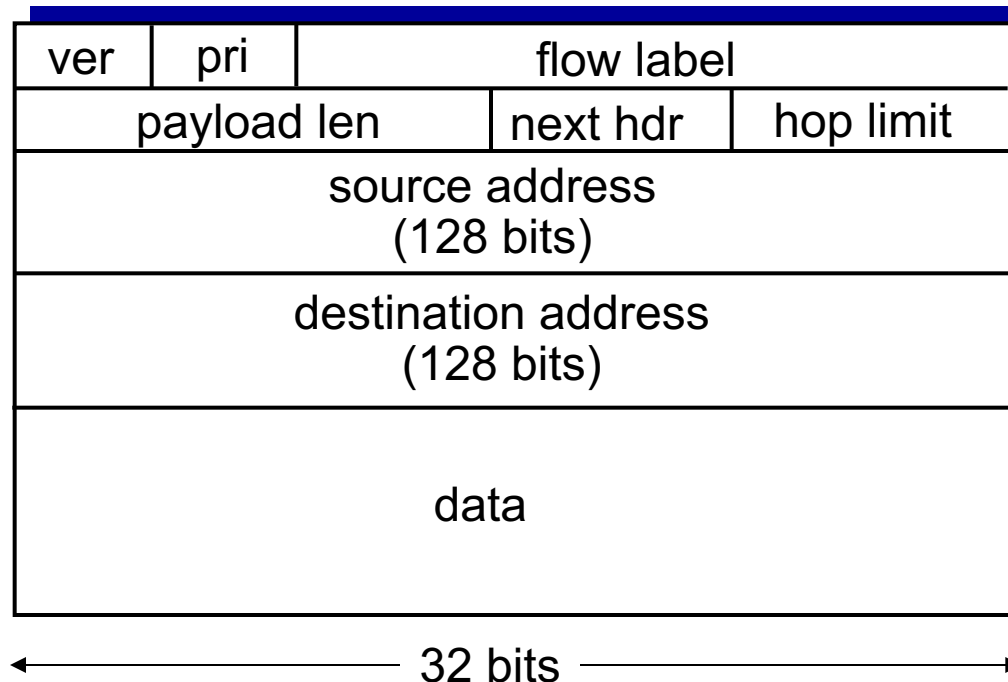
# IPv6 datagram format

*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

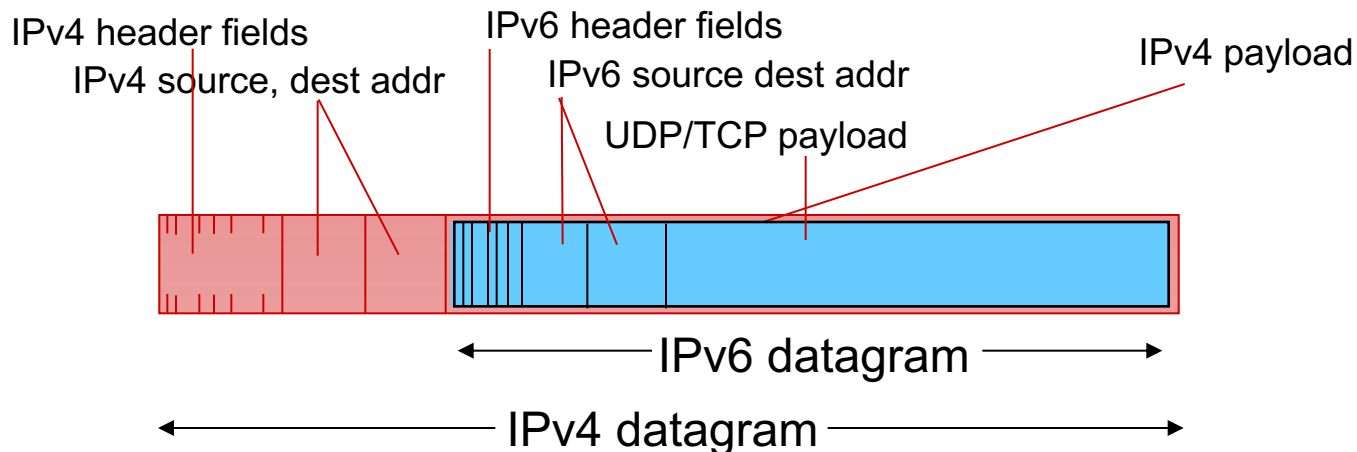


# Other changes from IPv4

- ❖ *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

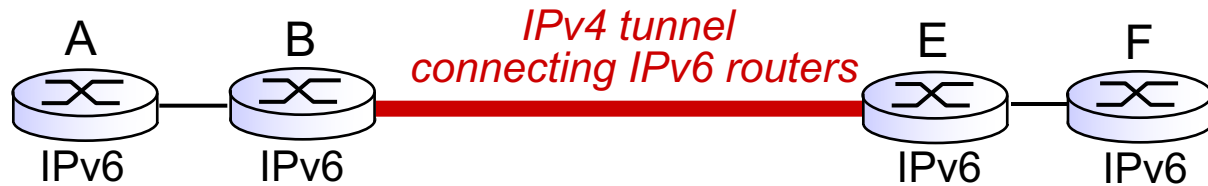
# Transition from IPv4 to IPv6

- ❖ 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

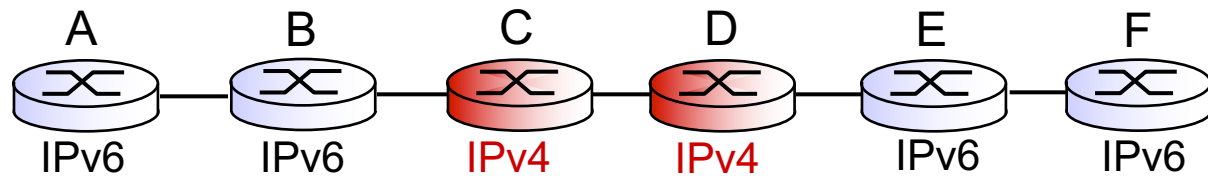


# Tunneling

logical view:



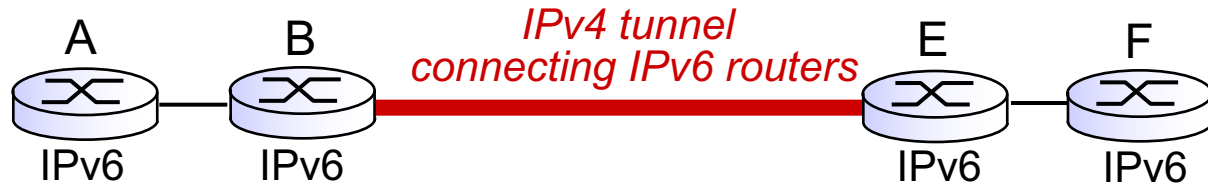
physical view:



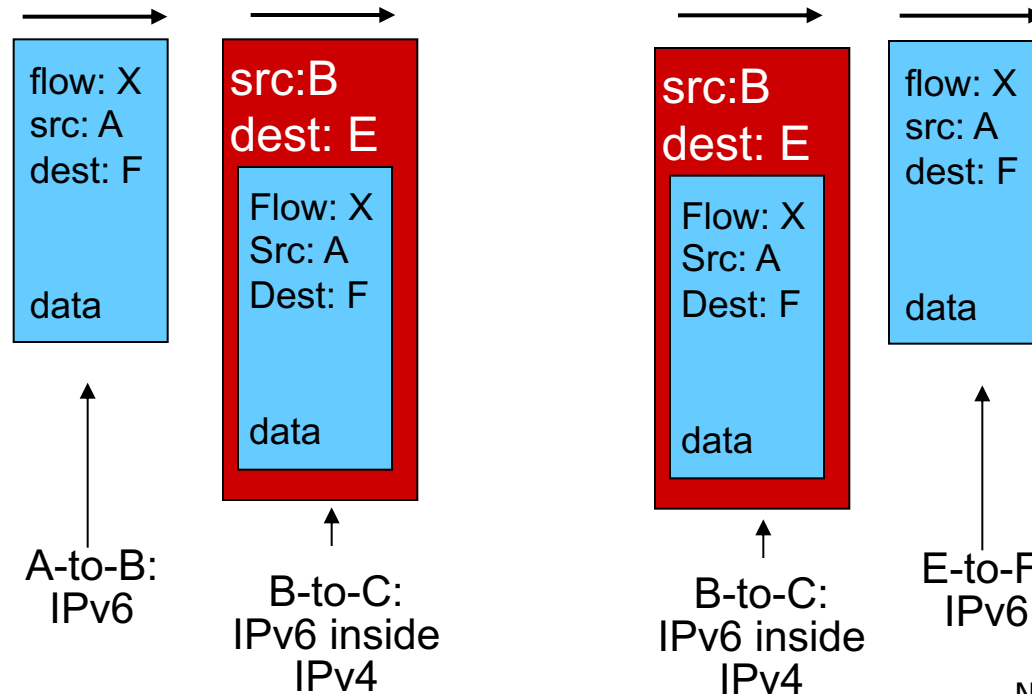
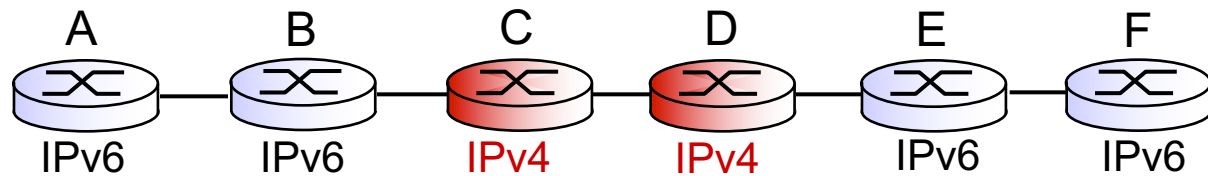


# Tunneling

logical view:



physical view:



# 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:

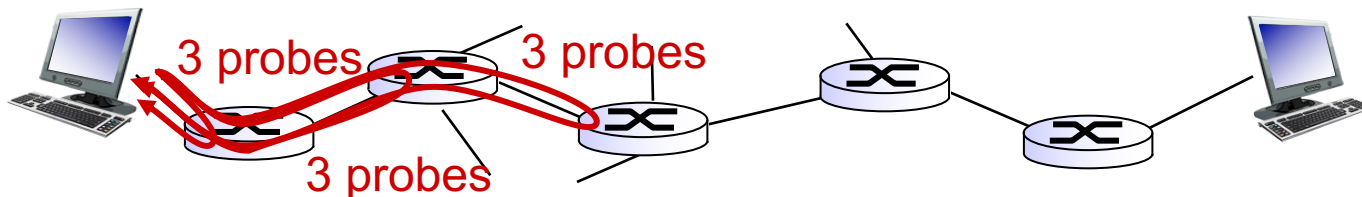
- ICMP msgs carried in IP datagrams

- ❖ **ICMP message:** type, code plus first 8 bytes of IP datagram causing error

<u>Type</u>	<u>Code</u>	<u>description</u>
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  $n$ th 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



# 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

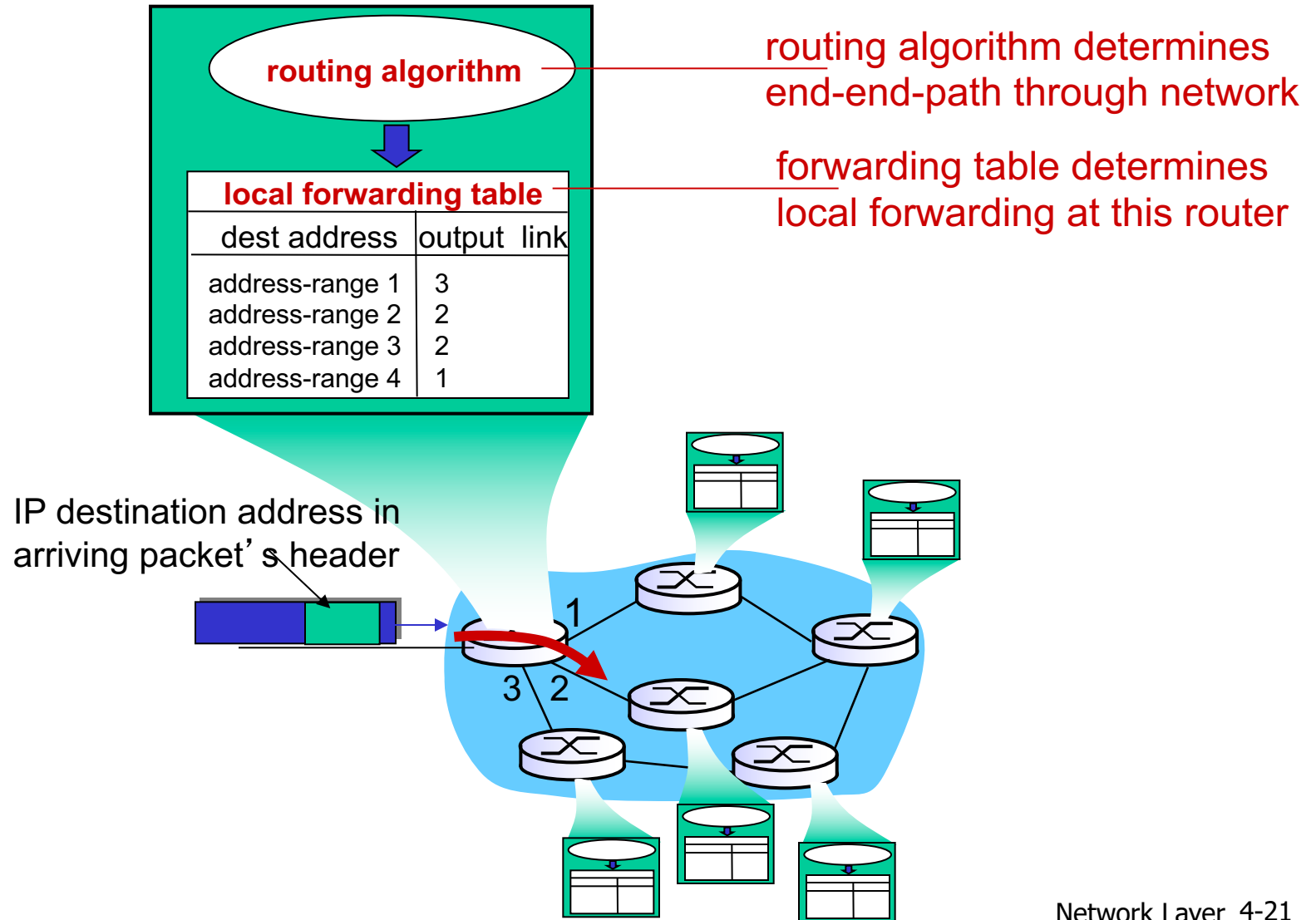
- link state
- distance vector
- hierarchical routing

## 4.6 routing in the Internet

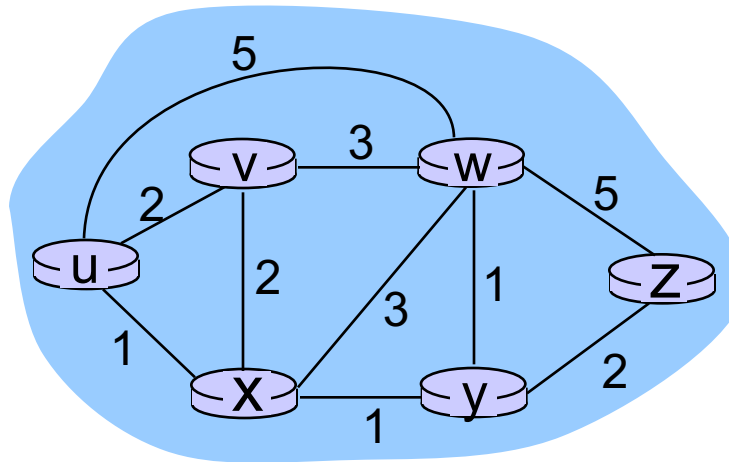
- RIP
- OSPF
- BGP

## 4.7 broadcast and multicast routing

# Interplay between routing, forwarding



# Graph abstraction



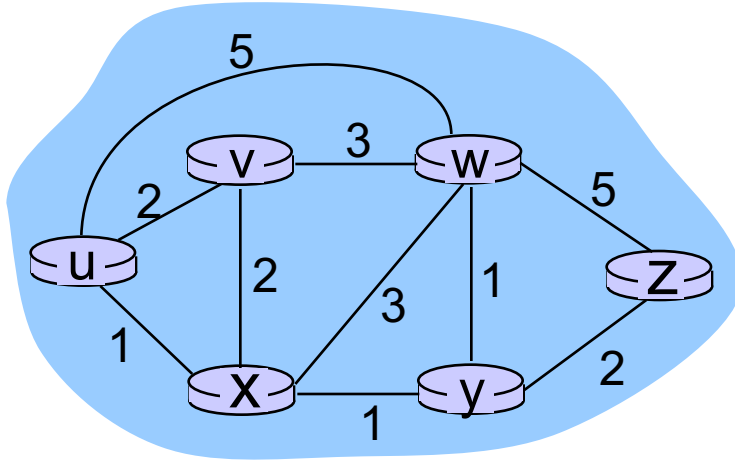
graph:  $G = (N, E)$

$N$  = 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



$c(x, x') = \text{cost of link } (x, x')$   
e.g.,  $c(w, z) = 5$

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

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

**key question:** what is the least-cost path between u and z ?  
**routing algorithm:** algorithm that finds that least cost path

# Routing algorithm classification

*Q: global or decentralized information?*

*global:*

- ❖ all routers have complete topology, link cost info
- ❖ “link state” algorithms

*decentralized:*

- ❖ router knows physically-connected neighbors, link costs to neighbors
- ❖ iterative process of computation, 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



# 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

# 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;  $= \infty$  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

# Dijkstra'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  $D(v) = \infty$

7

8 **Loop**

9 find  $w$  not in  $N'$  such that  $D(w)$  is a minimum

10 add  $w$  to  $N'$

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$  :

12  **$D(v) = \min( D(v), D(w) + c(w,v) )$**

13 /\* new cost to  $v$  is either old cost to  $v$  or known

14 shortest path cost to  $w$  plus cost from  $w$  to  $v$  \*/

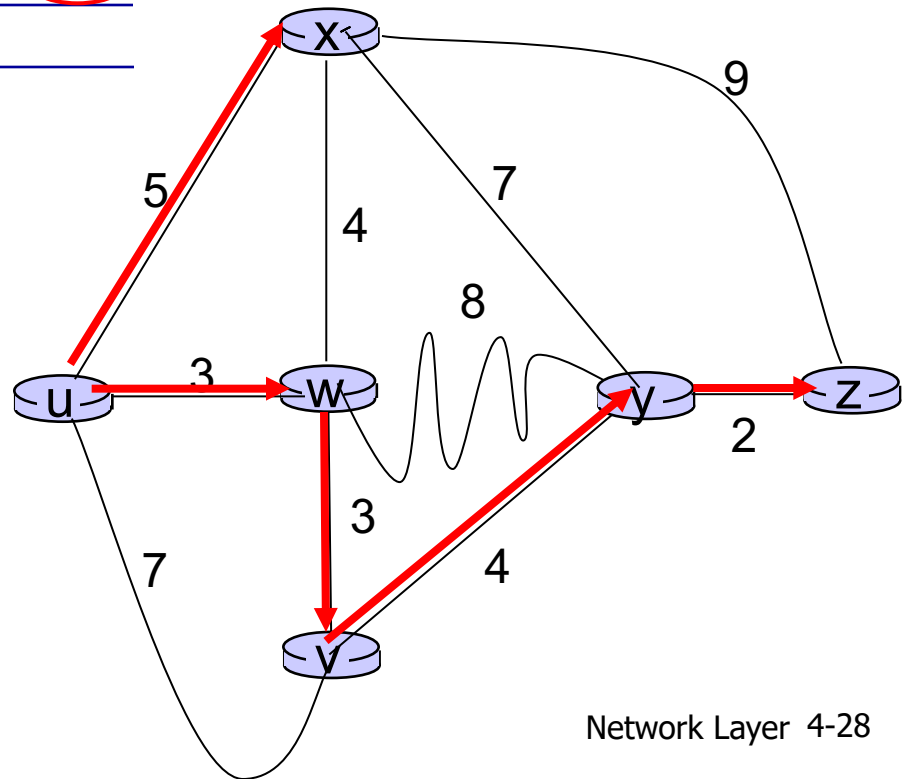
15 **until all nodes in  $N'$**

# Dijkstra's algorithm: example

Step	N'	D( <b>v</b> ) p(v)	D( <b>w</b> ) p(w)	D( <b>x</b> ) p(x)	D( <b>y</b> ) p(y)	D( <b>z</b> ) p(z)
0	u	7,u	<b>3,u</b>	5,u	$\infty$	$\infty$
1	uw	6,w		<b>5,u</b>	11,w	$\infty$
2	uwx	<b>6,w</b>			11,w	14,x
3	uwxv				<b>10,v</b>	14,x
4	uwxvy					<b>12,y</b>
5	uwxvyz					

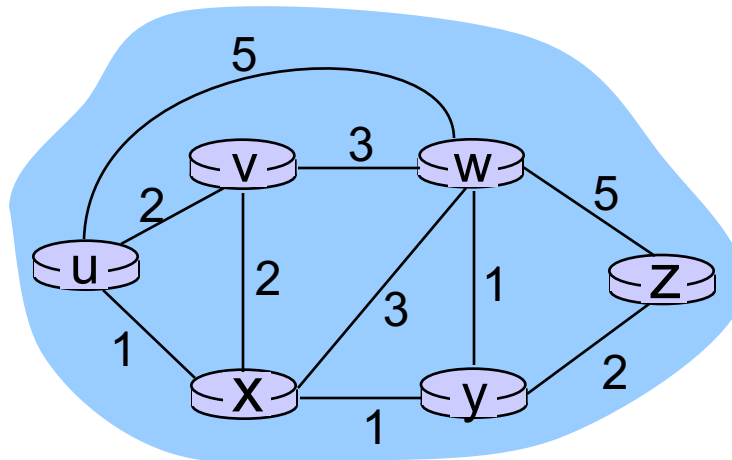
## notes:

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



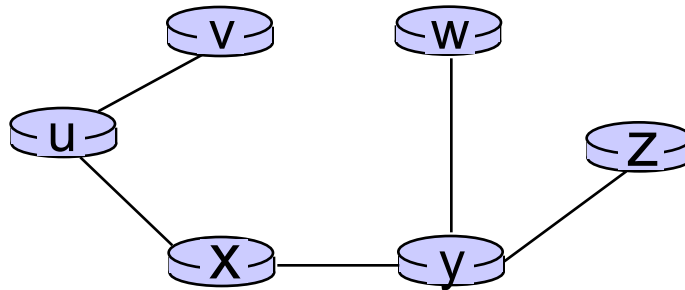
# Dijkstra's algorithm: another example

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	$\infty$	$\infty$
1	ux	2,u	4,x		2,x	$\infty$
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)

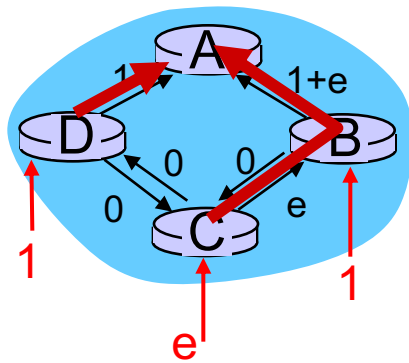
# Dijkstra's algorithm, discussion

*algorithm complexity:* n nodes

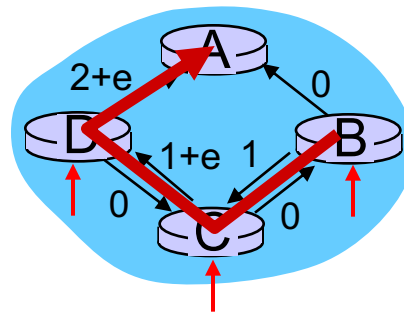
- ❖ each iteration: need to check all nodes, w, not in N
- ❖  $n(n+1)/2$  comparisons:  $O(n^2)$
- ❖ more efficient implementations possible:  $O(n \log n)$

*oscillations possible:*

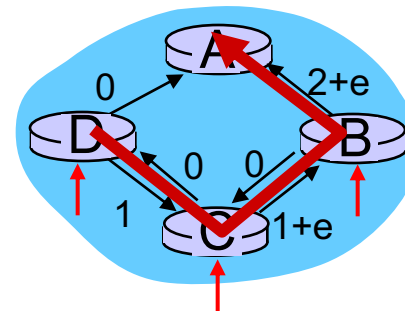
- ❖ e.g., support link cost equals amount of carried traffic:



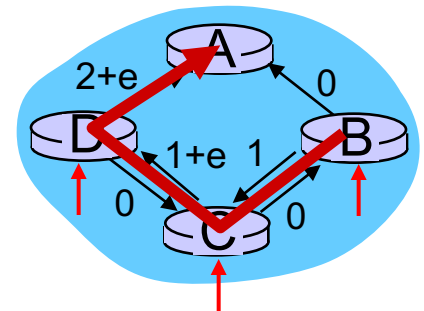
initially



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

# 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

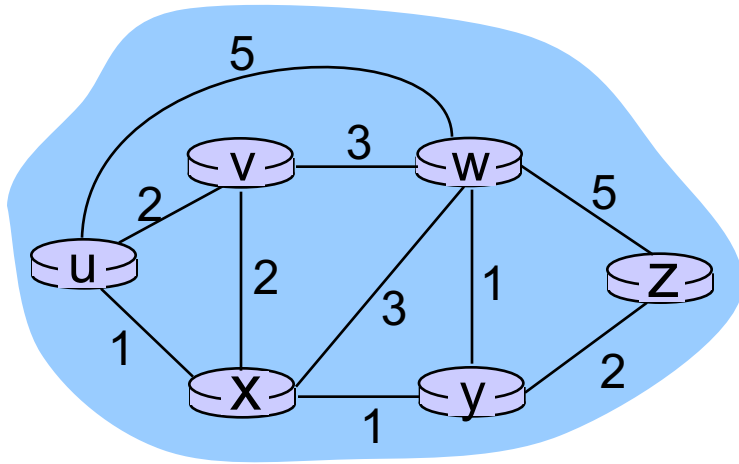
$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

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:

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

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

- ❖ Cosa stiamo dicendo e perchè ci permette di ottenere un cammino minimo
- ❖  $d_x(y)^{h+1} = \min \{ d_x(y)^h, \min \{ c(x,v) + d_v(y)^h \} \}$

# 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  $x$

# 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:*

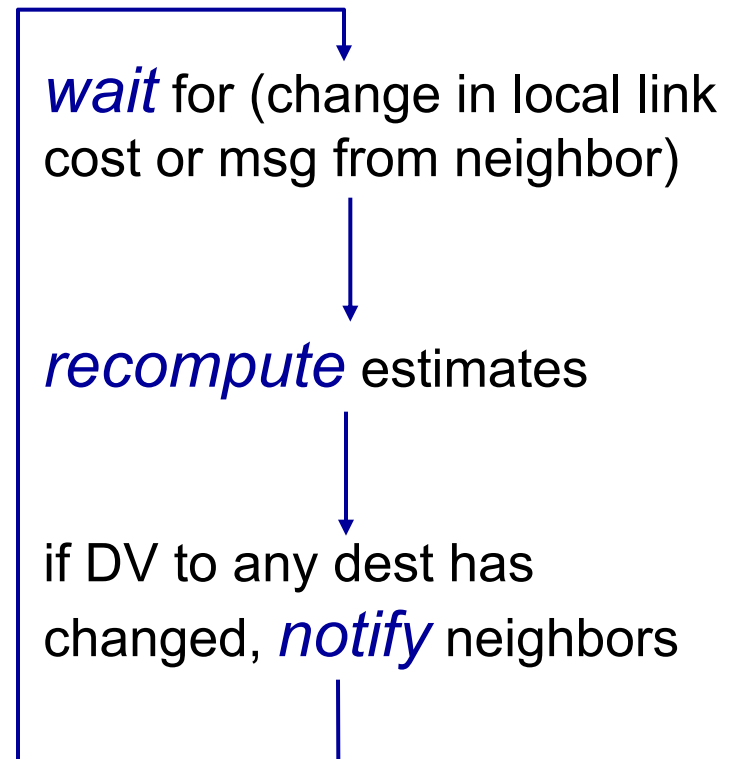
each local iteration  
caused by:

- ❖ 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**

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

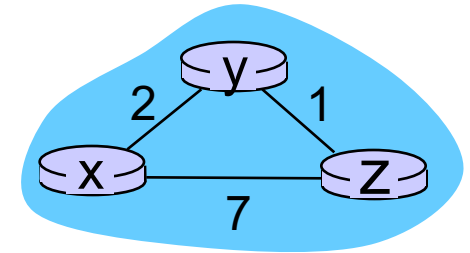
		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

**node y  
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

**node z  
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0



time



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

$$= \min\{2+0, 7+1\} = 2$$

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

$$= \min\{2+1, 7+0\} = 3$$

**node x  
table**

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

**node y  
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

**node z  
table**

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

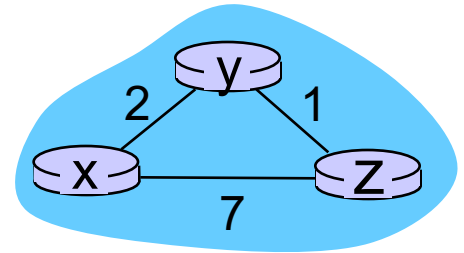
		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

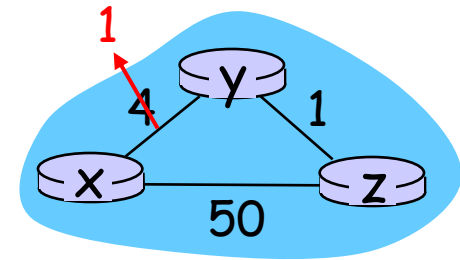


time

# 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.

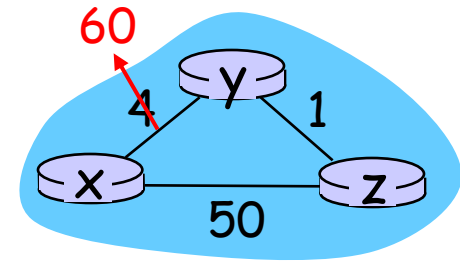
$t_1$ : z receives update from y, updates its table, computes new least cost to x, sends its neighbors its DV.

$t_2$ : 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: see text



## *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?

# Distributed Bellman Ford-Count to Infinity (we will now use a slightly different notation- lightweigh)

## Distance Table data structure

each node has its own

- ❖ row for each possible destination
- ❖ column for each directly-attached neighbor to node
- ❖ example: in node X, for dest. Y via neighbor Z:

Cost associated to the (X,Z) link

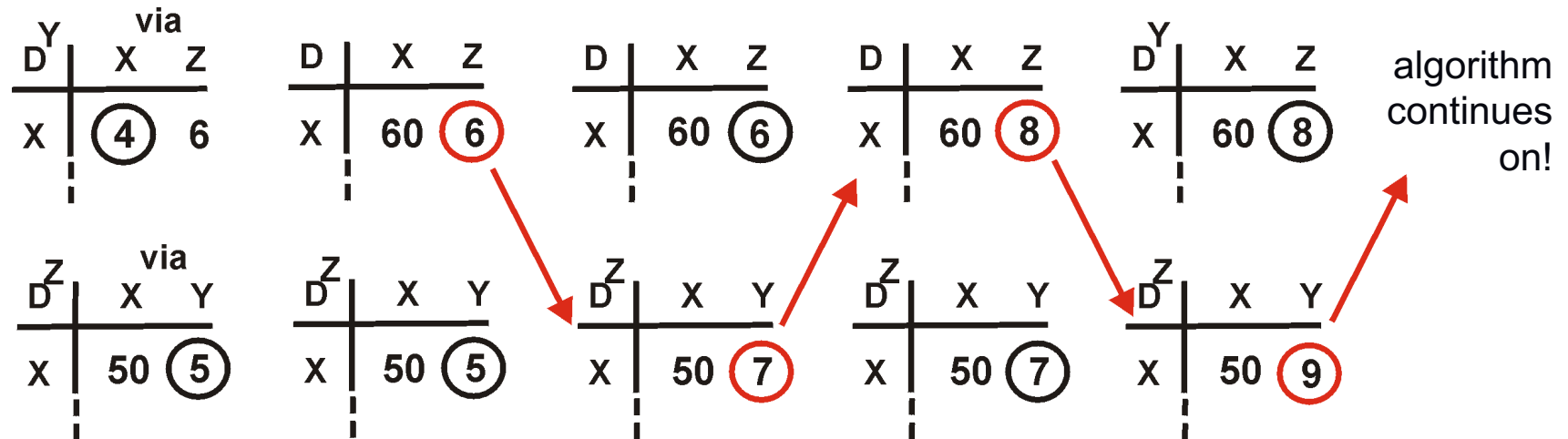
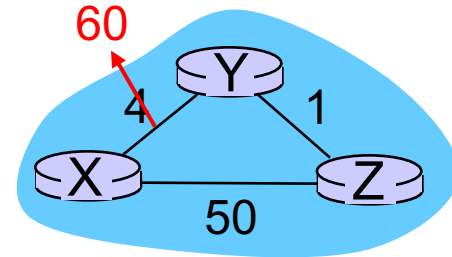
$$\begin{aligned} D^X_{(Y,Z)} &= \text{distance from X to Y, via Z as next hop} \\ &= c(X,Z) + \min_w \{D^Z(Y,w)\} \end{aligned}$$

Info maintained at Z. Min must be communicated

# Distance Vector: link cost changes

## Link cost changes:

- good news travels fast
- **bad news travels slow** - “count to infinity” problem!



algorithm continues on!

$c(X,Y)$   
change

time

$t_0$   $t_1$   $t_2$   $t_3$   $t_4$

Y detects link cost  
Increase but think can  
Reach X through Z at a  
total cost of 6 (wrong!!)

**The path is Y-Z-Y-X**

# Count-to-infinity – an everyday life example

*Which is the problem here?*

**the info exchanged by the protocol!! ‘the best route to X I have has the following cost...’ (no additional info on the route)**

A Roman example...

-assumption: there is only one route going from Colosseo to Altare della Patria: Via dei Fori Imperiali. Let us now consider a network, whose nodes are Colosseo., Altare della Patria, Piazza del Popolo



# Count-to-infinity – everyday life example (2/2)



The Colosseo. and Alt. Patria nodes exchange the following info

- Colosseo says ‘the shortest route from me to P. Popolo is 2 Km’
- Alt. Patria says ‘the shortest path from me to P. Popolo is 1Km’

**Based on this exchange from Colosseo you go to Al. Patria, and from there to Piazza del Popolo OK** Now due to the big dig they close **Via del Corso (Al. Patria—P.Popolo)**

- Al. Patria thinks ‘I have to find another route from me to P.Popolo.

Look there is a route from Colosseo to P.Popolo that takes 2Km, I can be at Colosseo in 1Km → I have found

a 3Km route from me to P.Popolo!!’ Communicates the new cost to Colosseo that updates ‘OK I can go to P.Popolo via Al. Patria in 4Km’

**VERY WRONG!! Why is it so? I didn’t know that the route from Colosseo to P.Popolo was going through Via del Corso from Al.Patria to P.Popolo (which is closed)!!**

# 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 propagate thru network



# 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

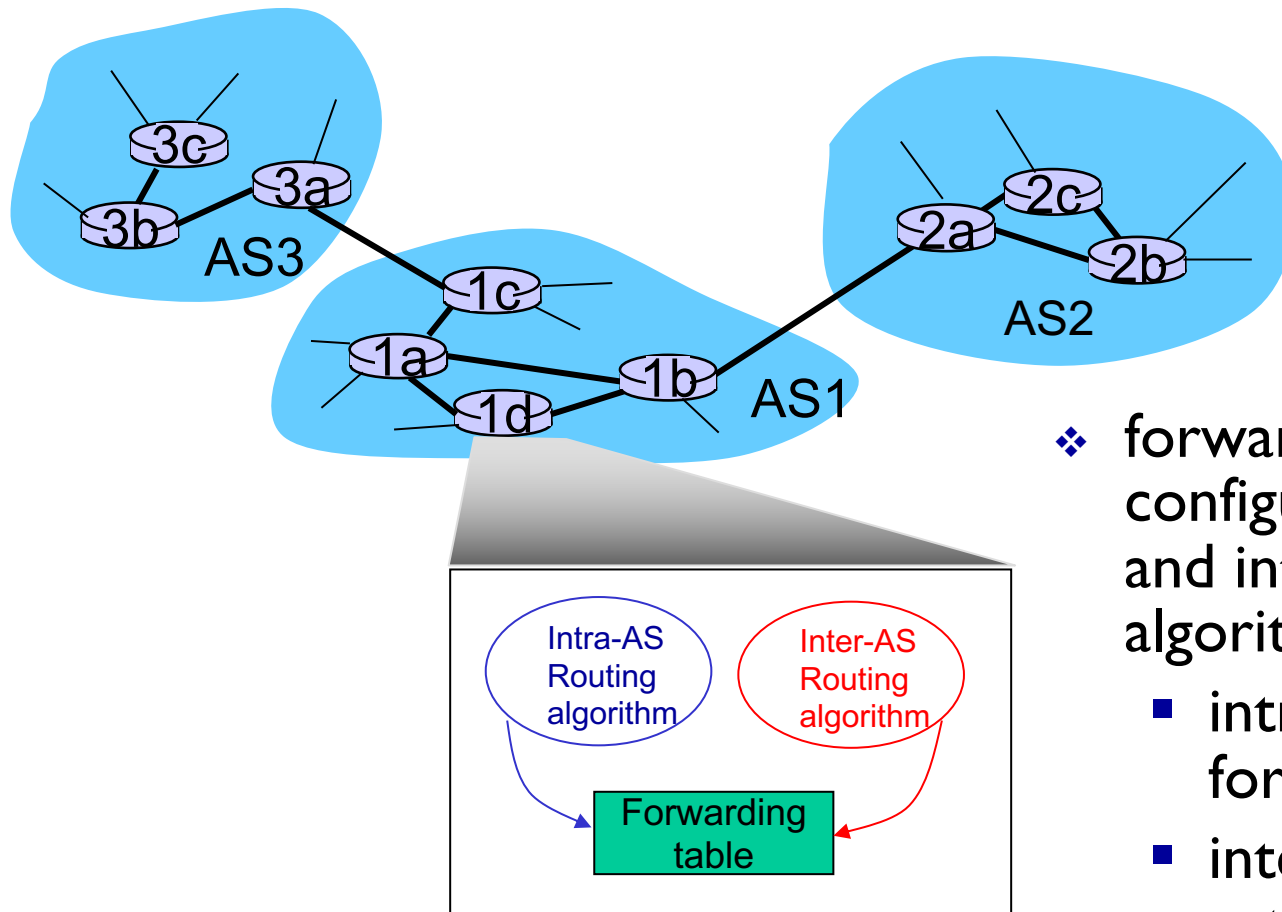
# Hierarchical routing

- ❖ 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

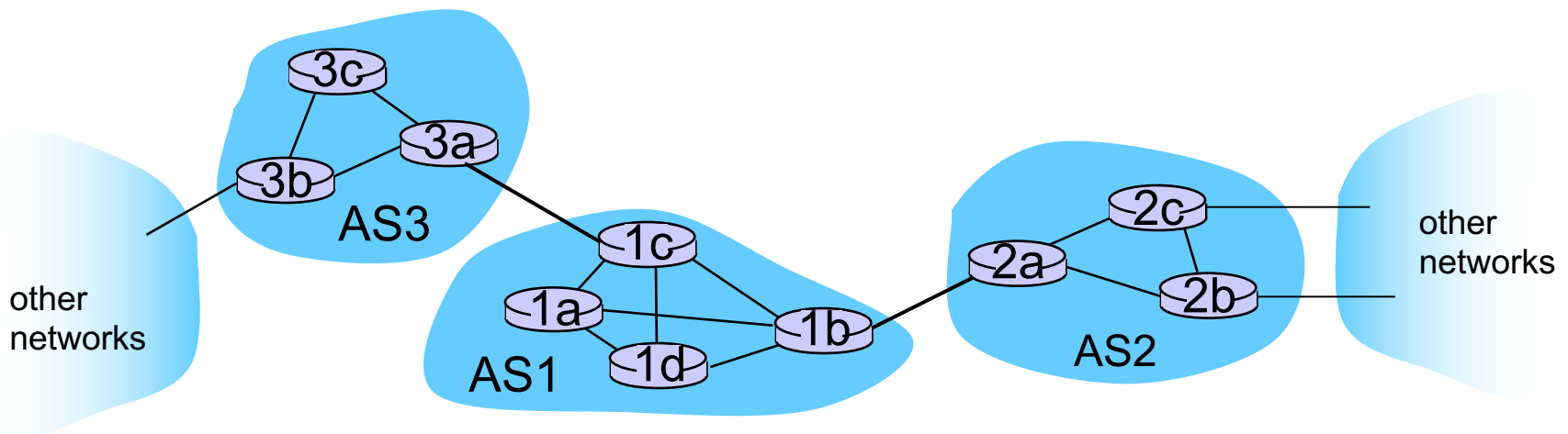
# Inter-AS tasks

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

*AS1 must:*

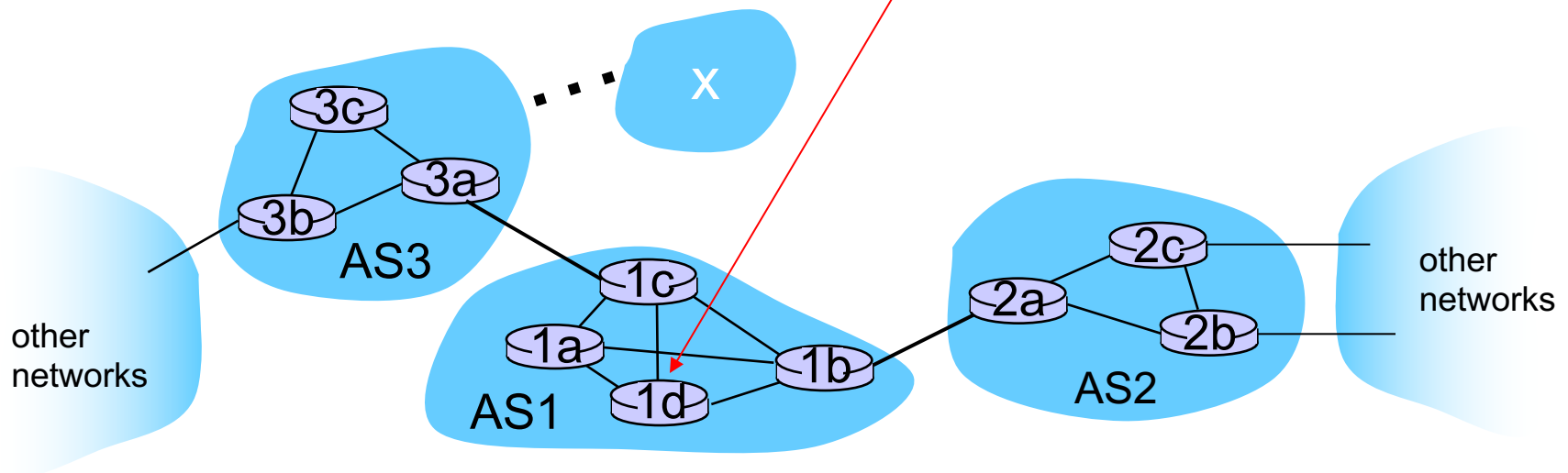
1. learn which destds are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

*job of inter-AS routing!*



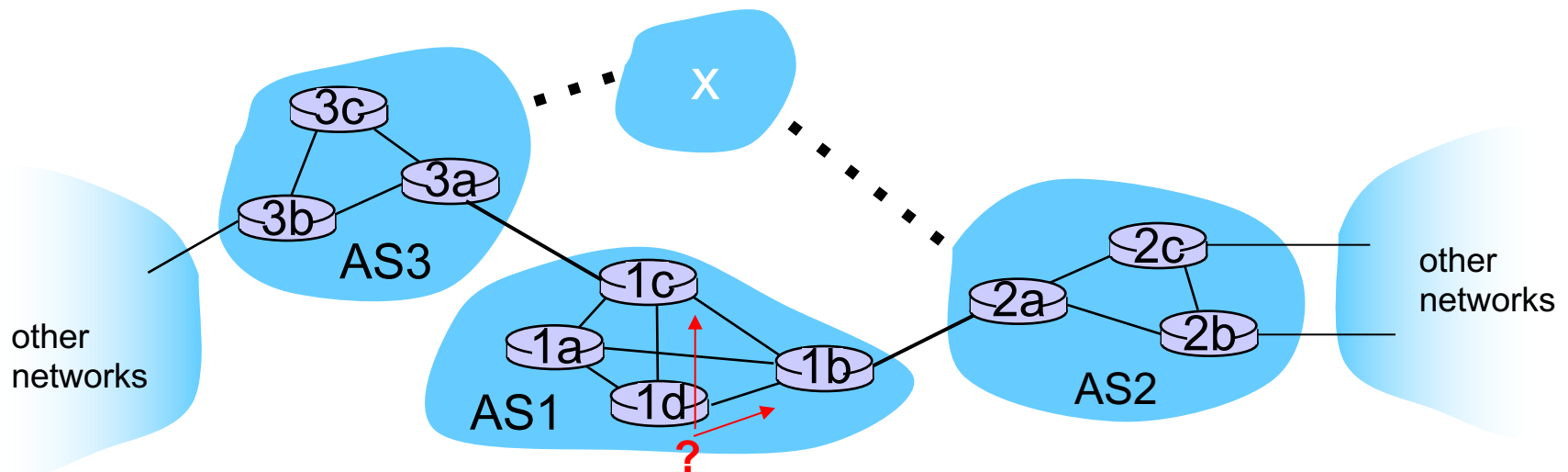
# Example: setting forwarding table in router 1d

- ❖ suppose AS1 learns (via inter-AS protocol) that subnet **x** reachable via AS3 (gateway 1c), but not via AS2
  - inter-AS protocol propagates reachability info to all internal routers
- ❖ router 1d determines from intra-AS routing info that its interface **l** is on the least cost path to 1c
  - installs forwarding table entry **(x,l)**



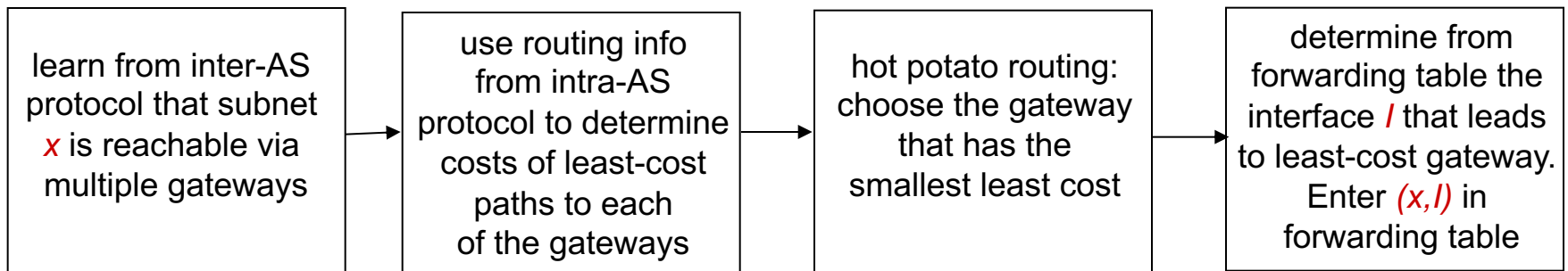
# Example: choosing among multiple ASes

- ❖ 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 which gateway it should forward packets towards for dest **x**
  - this is also job of inter-AS routing protocol!



# 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

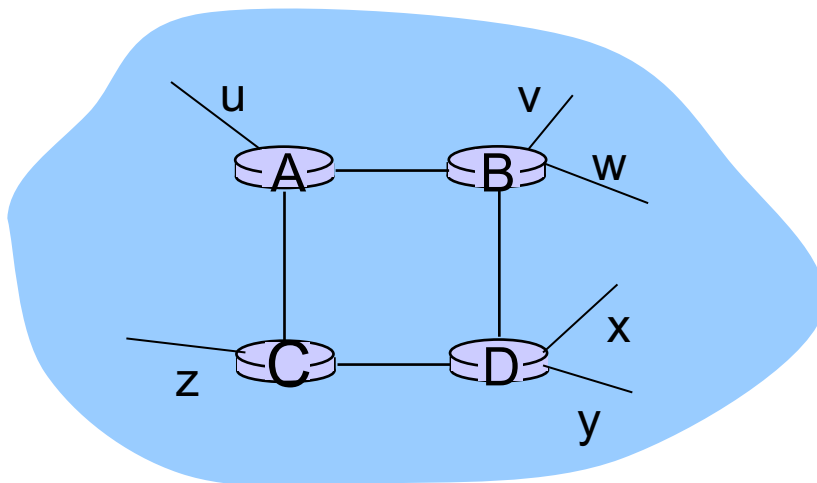
## 4.7 broadcast and multicast routing

# 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)

# RIP ( Routing Information Protocol)

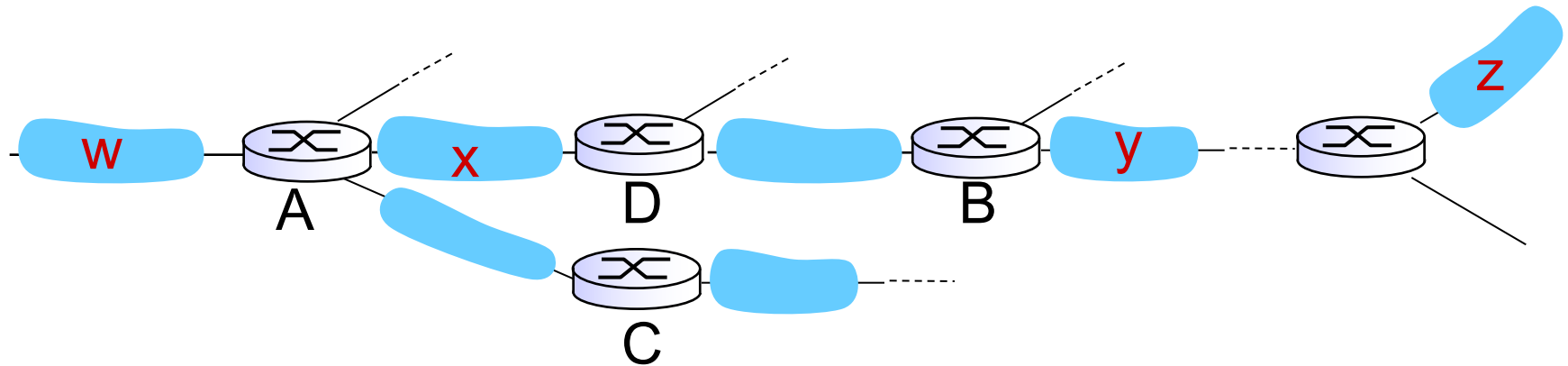
- ❖ included in BSD-UNIX distribution in 1982
- ❖ distance vector algorithm
  - distance metric: # hops (max = 15 hops), each link has cost 1
  - DVs exchanged with neighbors every 30 sec in response message (aka **advertisement**)
  - each advertisement: list of up to 25 destination **subnets** (in IP addressing sense)



from router A to destination **subnets**:

<u>subnet</u>	<u>hops</u>
u	1
v	2
w	2
x	3
y	3
z	2

# RIP: example



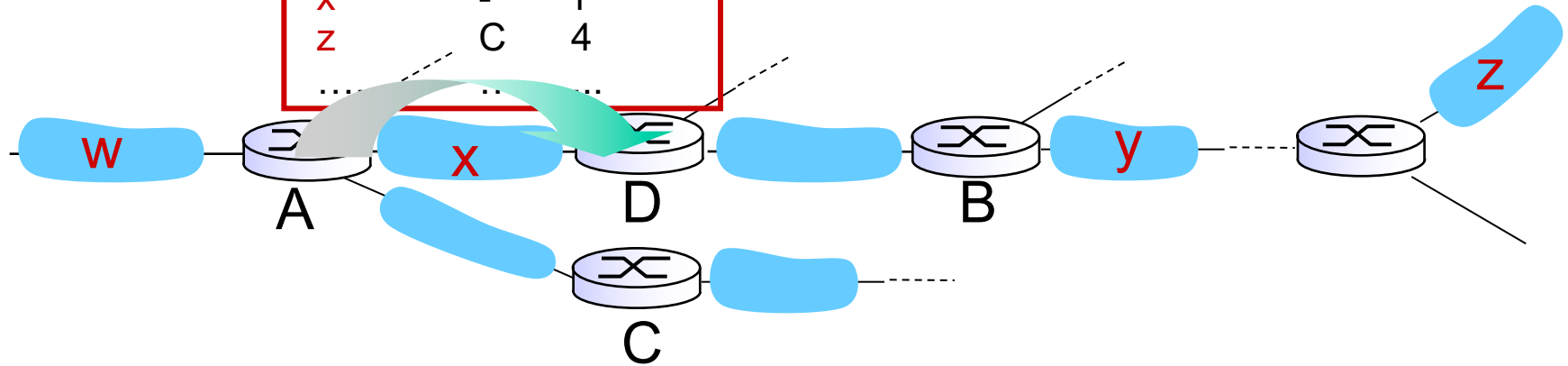
routing table in router D

destination subnet	next router	# hops to dest
w	A	2
y	B	2
z	B	7
x	--	1
....	....	....

# RIP: example

A-to-D advertisement

dest	next	hops
W	-	1
X	-	1
Z	C	4
...	...	...



routing table in router D

destination subnet	next router	# hops to dest
W	A	2
y	B	2
Z	<del>B</del> A	<del>7</del> 5
X	--	1
....	....	....

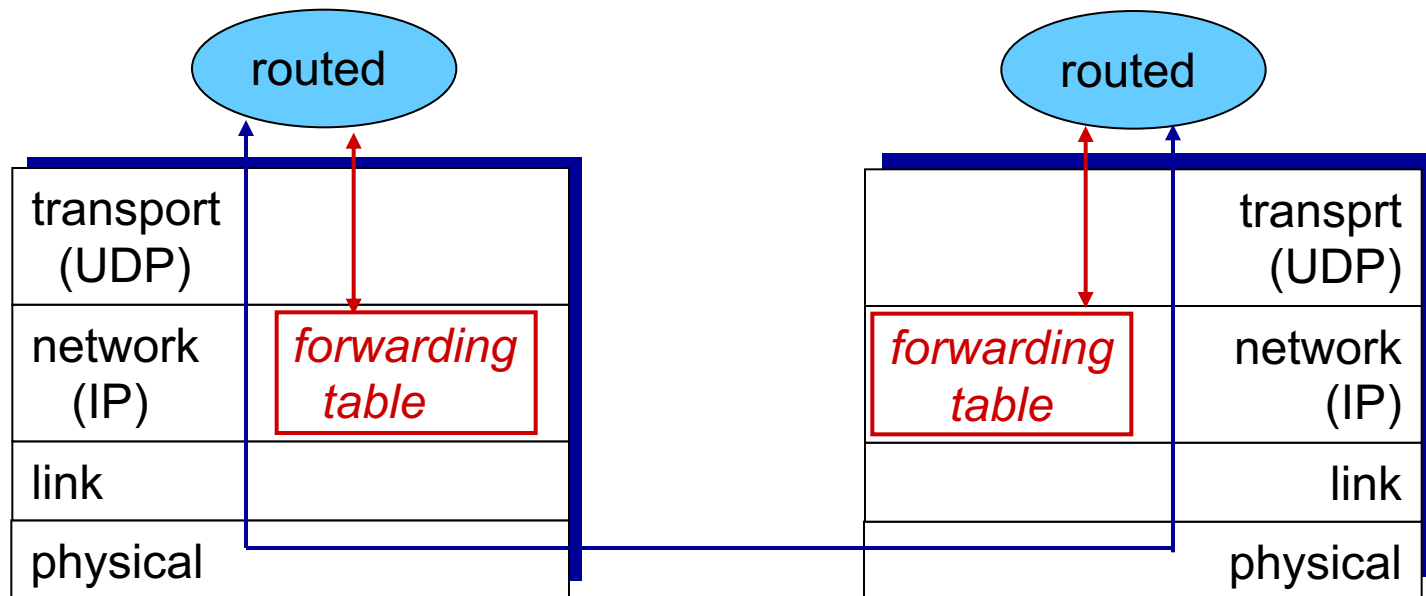
# RIP: link failure, recovery

if no advertisement heard after 180 sec -->  
neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly (?) propagates to entire net
- *poison reverse* used to prevent ping-pong loops (infinite distance = 16 hops)

# RIP table processing

- ❖ RIP routing tables managed by *application-level* process called route-d (daemon)
- ❖ advertisements sent in UDP packets, periodically repeated



# OSPF (Open Shortest Path First)

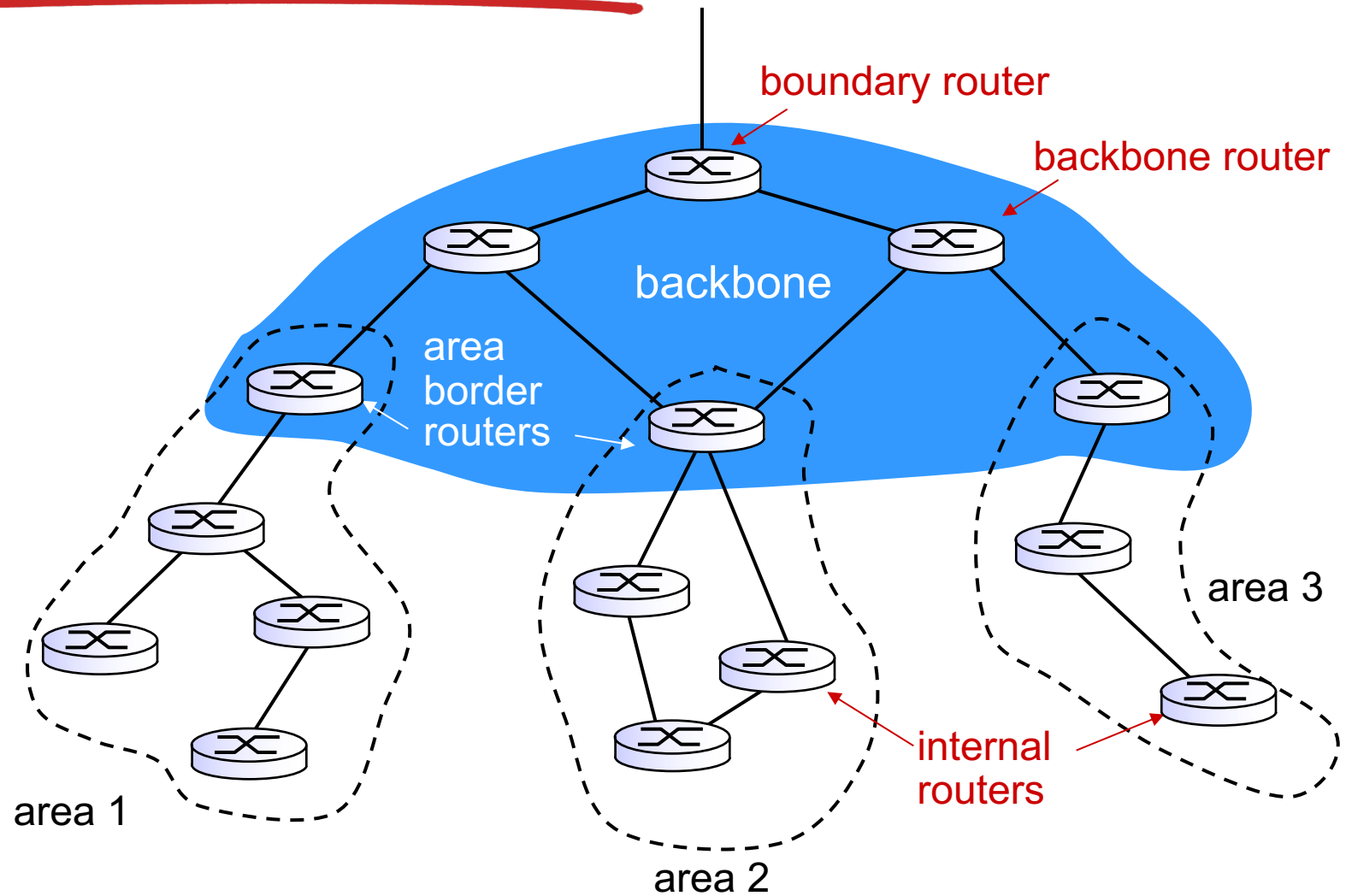
- ❖ “open”: publicly available
- ❖ uses link state algorithm
  - LS packet dissemination
  - topology map at each node
  - route computation using Dijkstra’s algorithm
- ❖ OSPF advertisement carries one entry per neighbor
- ❖ advertisements flooded to *entire* AS
  - carried in OSPF messages directly over IP (rather than TCP or UDP)
- ❖ *IS-IS routing* protocol: nearly identical to OSPF



# OSPF “advanced” features (not in RIP)

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

# Hierarchical OSPF



# Hierarchical OSPF

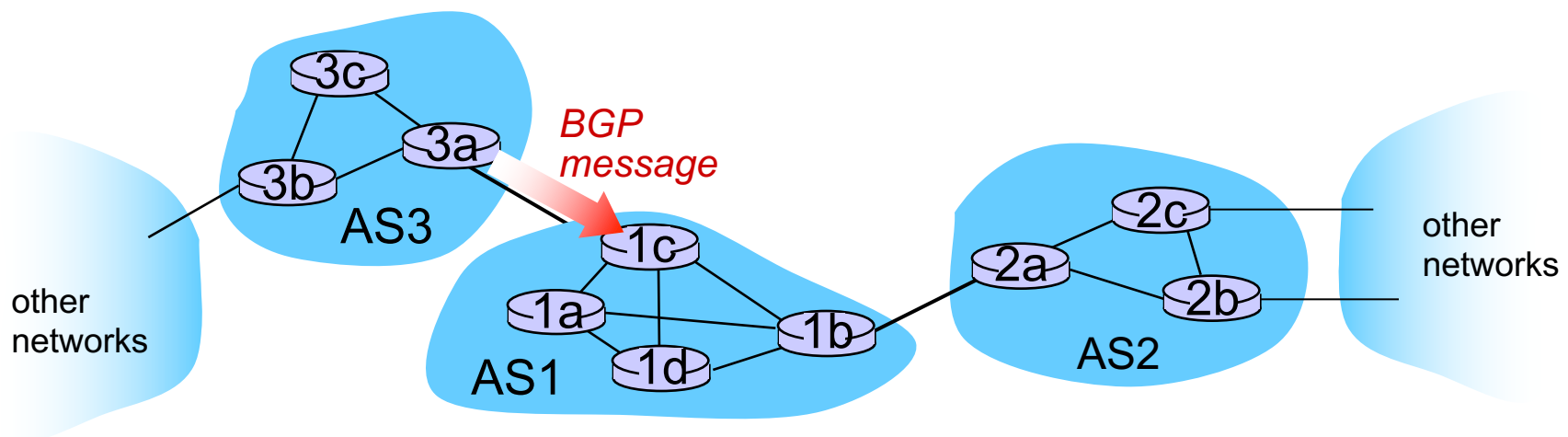
- ❖ *two-level hierarchy*: local area, backbone.
  - link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- ❖ *area border routers*: “summarize” distances to nets in own area, advertise to other Area Border routers.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

# 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*”

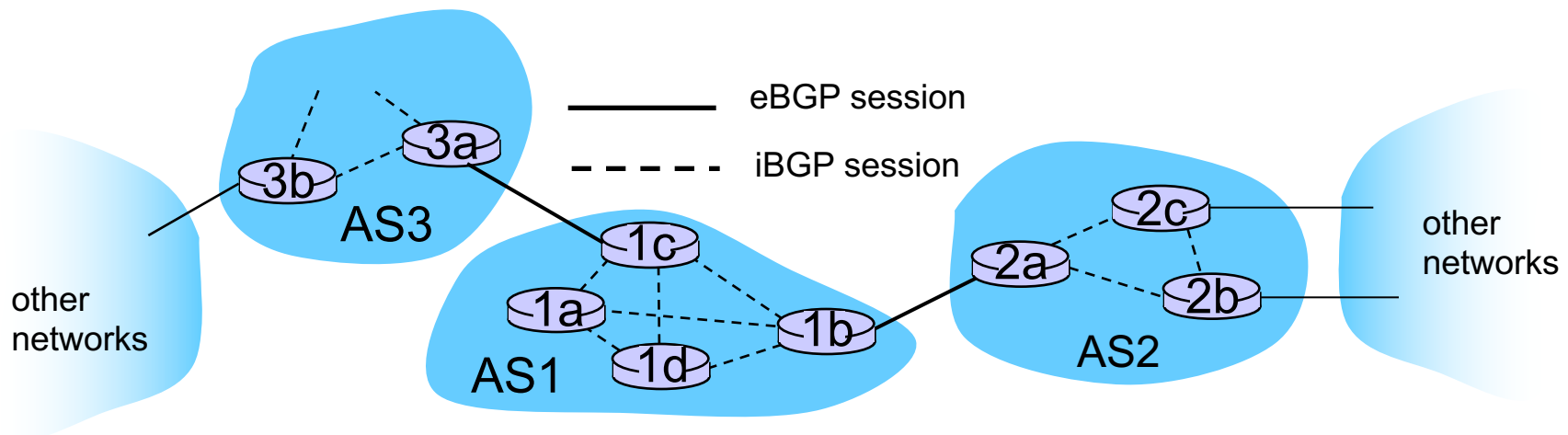
# BGP basics

- ❖ **BGP session:** two BGP routers (“peers”) exchange BGP messages:
  - advertising *paths* to different destination network prefixes (“path vector” protocol)
  - exchanged over semi-permanent TCP connections
- ❖ when AS3 advertises a prefix to AS1:
  - AS3 *promises* it will forward datagrams towards that prefix
  - AS3 can aggregate prefixes in its advertisement



# BGP basics: distributing path information

- ❖ using eBGP session between 3a and 1c, AS3 sends prefix reachability info to AS1.
  - 1c can then use iBGP to distribute new prefix info to all routers in AS1
  - 1b can then re-advertise new reachability info to AS2 over 1b-to-2a eBGP session
- ❖ when router learns of new prefix, it creates entry for prefix in its forwarding table.



# Path attributes and BGP routes

- ❖ advertised prefix includes BGP attributes
  - prefix + attributes = “route”
- ❖ two important attributes:
  - **AS-PATH**: contains ASs through which prefix advertisement has passed: e.g., AS 67, AS 17
  - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS. (may be multiple links from current AS to next-hop-AS)
- ❖ gateway router receiving route advertisement uses **import policy** to accept/decline
  - e.g., never route through AS x
  - *policy-based* routing

# BGP route selection

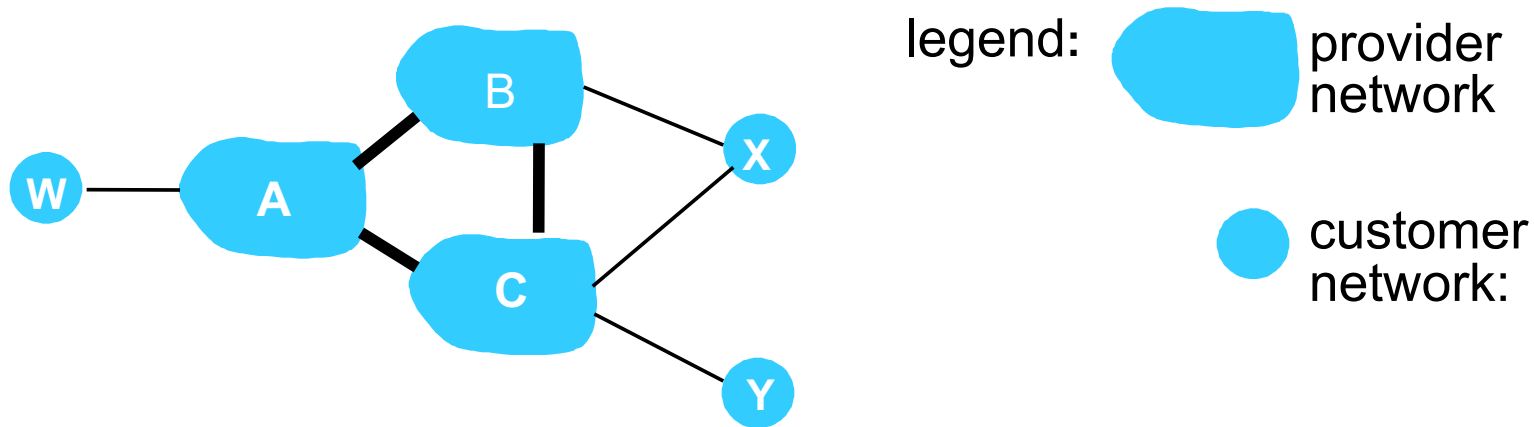
- ❖ router may learn about more than 1 route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria



# BGP messages

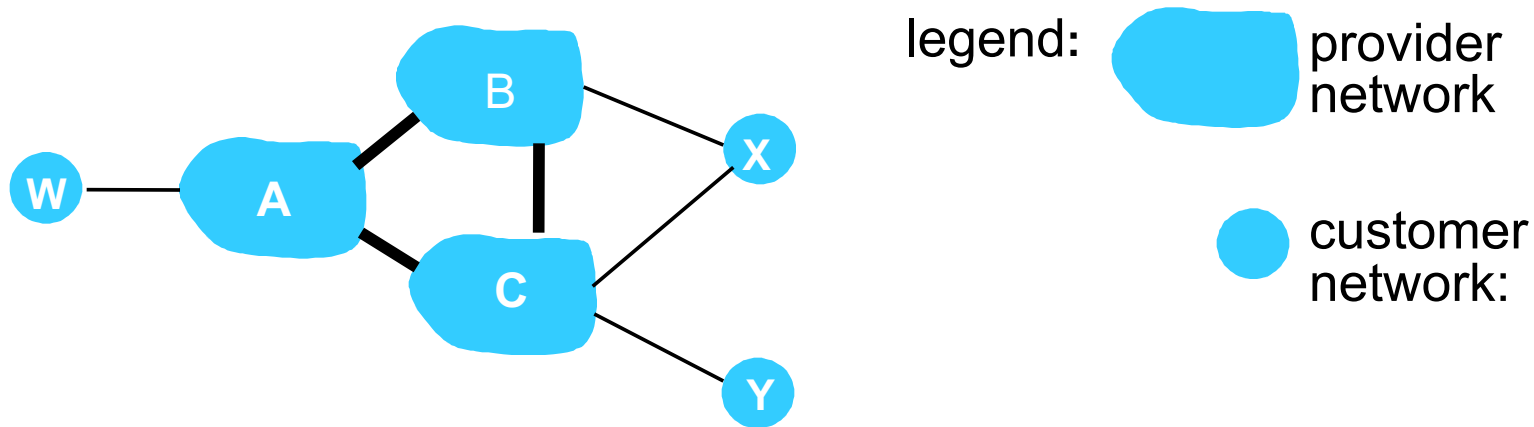
- ❖ BGP messages exchanged between peers over TCP connection
- ❖ BGP messages:
  - **OPEN**: opens TCP connection to peer and authenticates sender
  - **UPDATE**: advertises new path (or withdraws old)
  - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - **NOTIFICATION**: reports errors in previous msg; also used to close connection

# BGP routing policy



- ❖ A,B,C are *provider networks*
- ❖ X,W,Y are customer (of provider networks)
- ❖ X is *dual-homed*: attached to two networks
  - X does not want to route from B via X to C
  - .. so X will not advertise to B a route to C

## BGP routing policy (2)



- ❖ A advertises path AW to B
- ❖ B advertises path BAW to X
- ❖ Should B advertise path BAW to C?
  - No way! B gets no “revenue” for routing CBAW since neither W nor C are B’s customers
  - B wants to force C to route to w via A
  - B wants to route *only* to/from its customers!

# 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

# 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

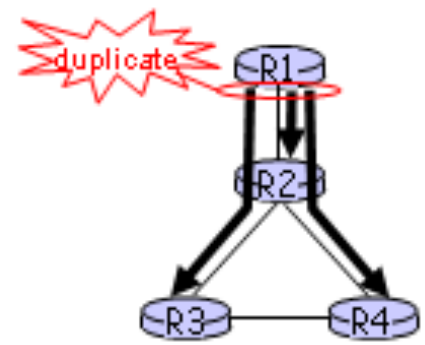
- link state
- distance vector
- hierarchical routing

## 4.6 routing in the Internet

- RIP
- OSPF
- BGP

## 4.7 broadcast and multicast routing

# Unicast ad N vie



## ❖ Inefficiente

- Un singolo collegamento attraversato da N copie del messaggio se il nodo origine è connesso al resto della rete tramite un unico collegamento

## ❖ Indirizzi di tutte le destinazioni devono essere noti al mittente

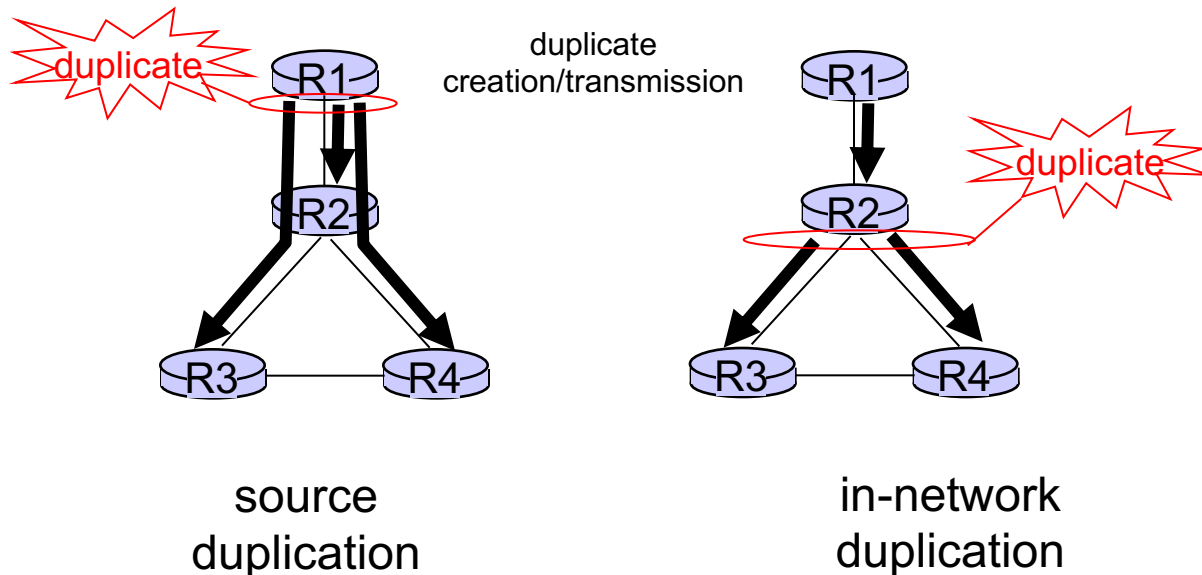
- altri meccanismi protocollari sono richiesti

## ❖ Broadcast può essere usato per inoltrare informazioni di topologia in una situazione in cui le rotte non sono ancora note

- es. OSPF

# Broadcast Routing

- ❖ deliver packets from source to all other nodes
- ❖ source duplication is inefficient:



- ❑ source duplication: how does source determine recipient addresses?

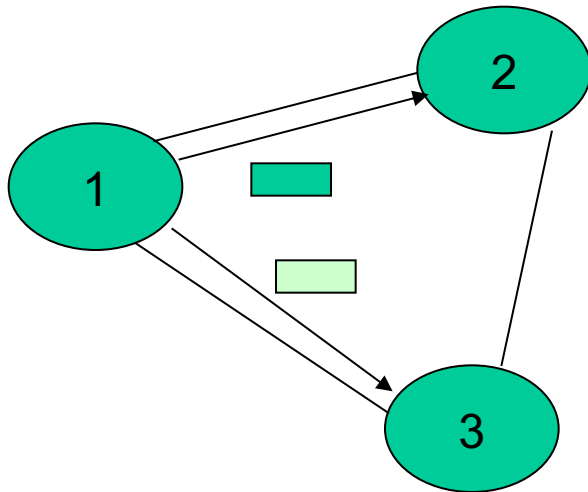
# In-network duplication

- ❖ *flooding*: when node receives broadcast packet, sends copy to all neighbors
  - problems: cycles & broadcast storm
- ❖ *controlled flooding*: node only broadcasts pkt if it hasn't broadcast same packet before
  - node keeps track of packet ids already broadcasted
  - or reverse path forwarding (RPF): only forward packet if it arrived on shortest path between node and source
- ❖ *spanning tree*:
  - no redundant packets received by any node



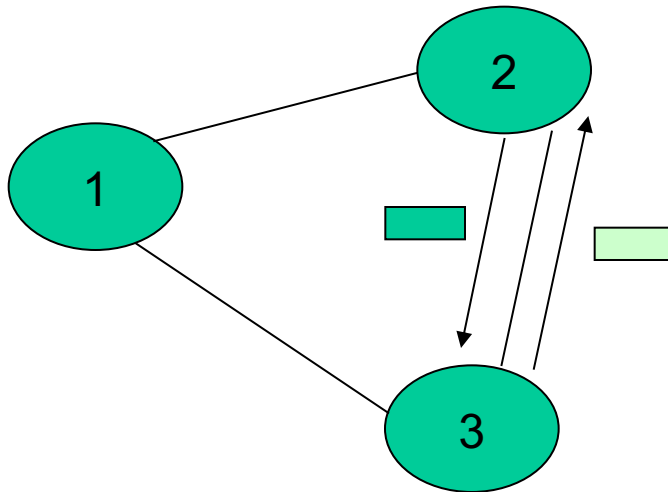
# In-network duplication

- ❖ flooding: when node receives brdcst pckt, sends copy to all neighbors EXCEPT the one from which the pckt was received
  - Problems: [cycles](#) & broadcast storm



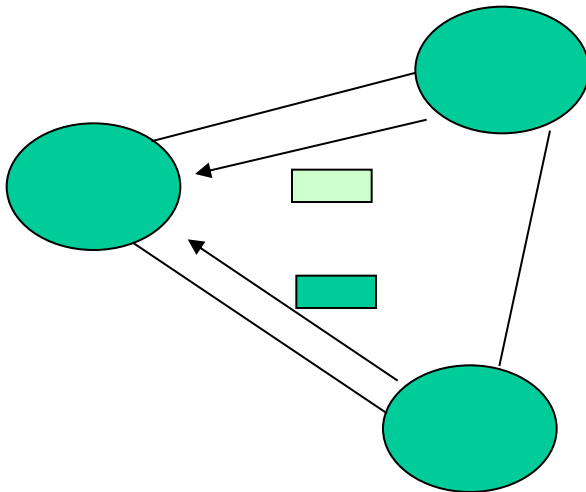
# In-network duplication

- ❖ flooding: when node receives brdcst pckt, sends copy to all neighbors
  - Problems: [cycles](#) & broadcast storm



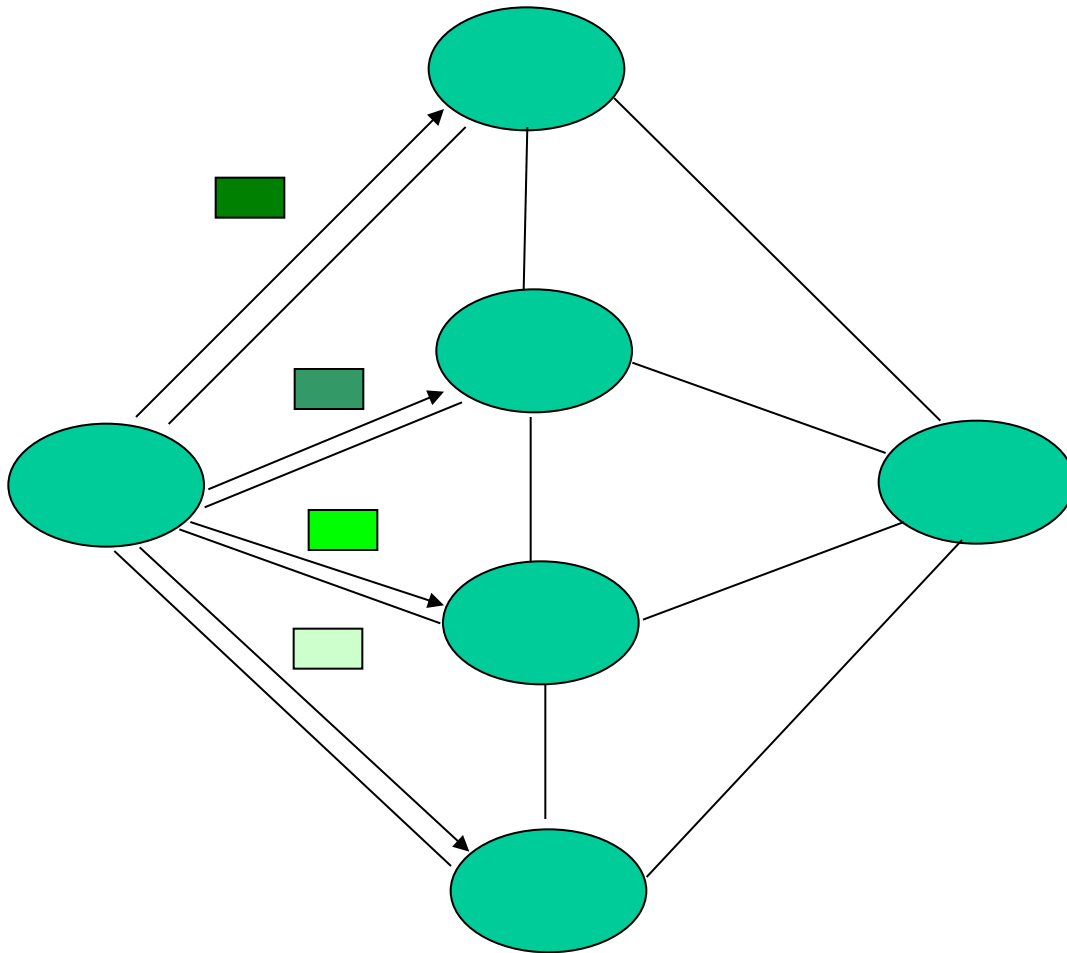
# In-network duplication

- ❖ flooding: when node receives brdcst pkt, sends copy to all neighbors
  - Problems: cycles & broadcast storm

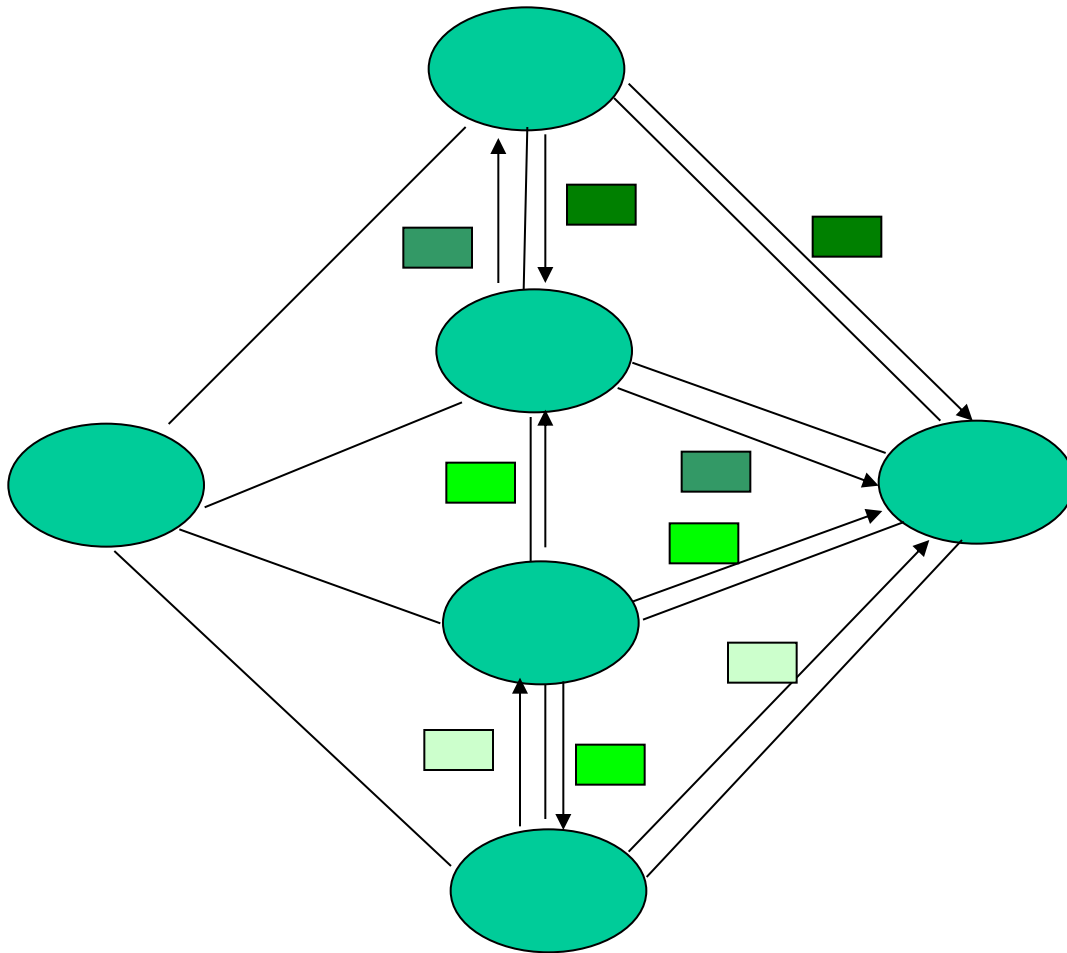


E ricominciamo come nella prima situazione  
Bisogna saper distinguere tra quando  
mandiamo un nuovo messaggio e quando  
stiamo ritrasmettendo qualcosa che  
abbiamo già visto  
→ Sequence numbers!

# Broadcast storm

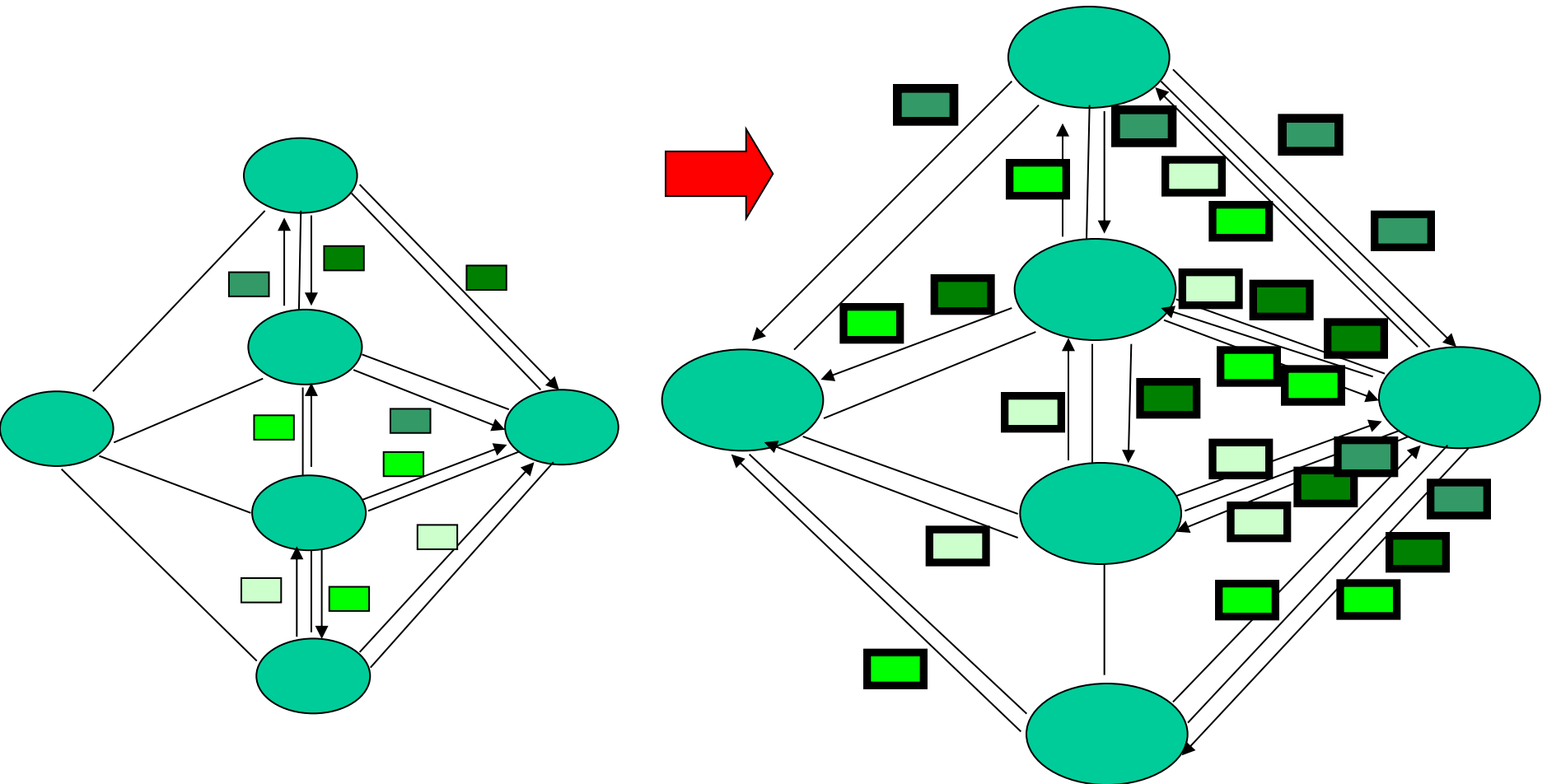


# Broadcast storm



# Broadcast storm

**Il numero di pacchetti in rete  
cresce significativamente!!**



# Controlled flooding

- ❖ Il nodo origine pone il proprio indirizzo ed il numero di sequenza nei pacchetti che invia in broadcast
- ❖ Ciascun nodo mantiene una lista di ID origine, SEQN per i broadcast ricevuti, trasmesso o inoltrato
- ❖ Se riceve un pacchetto broadcast per prima cosa verifica se  $\langle \text{ID}, \text{SEQN} \rangle$  compare nella lista dei pacchetti già gestiti
  - Se si scarta
  - Altrimenti riinvia su tutte le interfacce tranne quella da cui ha ricevuto

# In-network duplication

- ❖ *flooding*: when node receives broadcast packet, sends copy to all neighbors
  - problems: cycles & broadcast storm
- ❖ *controlled flooding*: node only broadcasts pkt if it hasn't broadcast same packet before
  - node keeps track of packet ids already broadcasted
  - or reverse path forwarding (RPF): only forward packet if it arrived on shortest path between node and source
- ❖ *spanning tree*:
  - no redundant packets received by any node

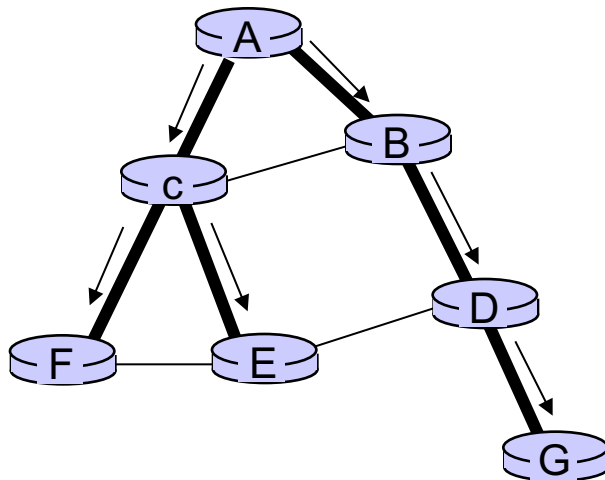


# Controlled flooding, altre opzioni

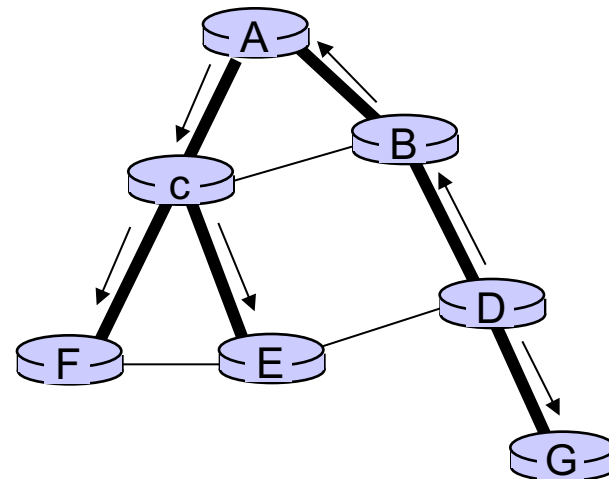
- ❖ Reverse path forwarding (RPF): only forward pckt (on all links but the one from which the packet was received) if it arrived on shortest path between node and source

# Spanning Tree

- ❖ First construct a spanning tree
- ❖ Nodes forward copies only along spanning tree



(a) Broadcast initiated at A



(b) Broadcast initiated at D

# Minimum spanning tree- Prim's Algorithm

❖ Prim's algorithm:

let  $T$  be a single vertex  $x$

while ( $T$  has fewer than  $n$  vertices) {

    Find the smallest edge connecting  $T$  to  $G-T$

    Add it to  $T$

}

# Minimum spanning tree--Kruskal algorithm

## ❖ Kruskal's algorithm:

Sort the edges of  $G$  in increasing order of weight

Keep a subgraph  $S$  of  $G$ , initially empty

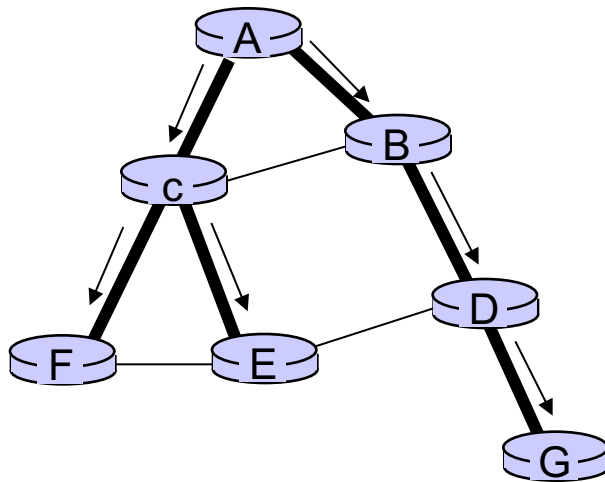
For each edge  $e$  in sorted order

    If the endpoints of  $e$  are  
disconnected                      in  $S$  then add  $e$  to  $S$

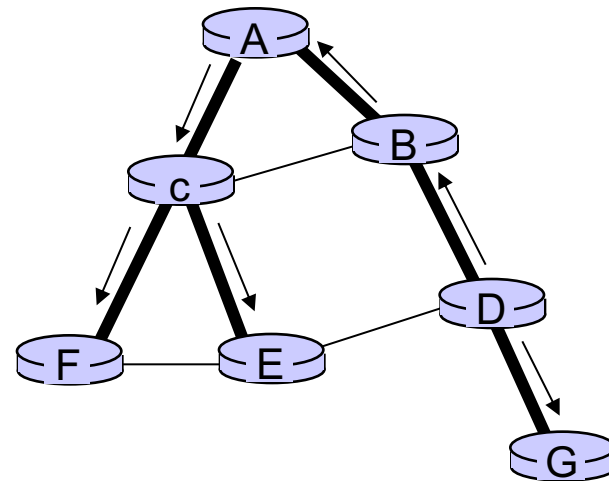
Return  $S$

# Spanning tree

- ❖ first construct a spanning tree
- ❖ nodes then forward/make copies only along spanning tree



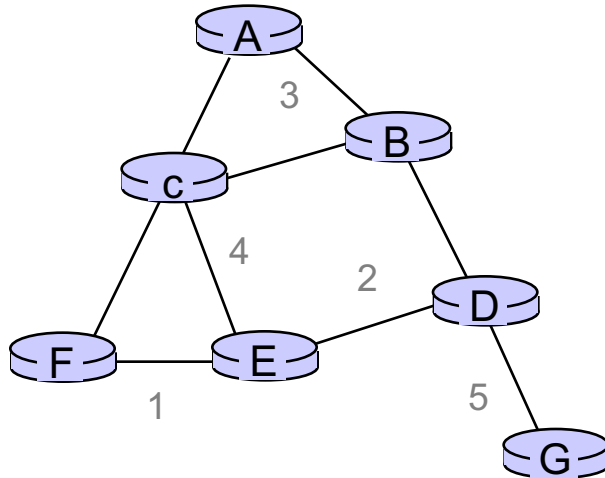
(a) broadcast initiated at A



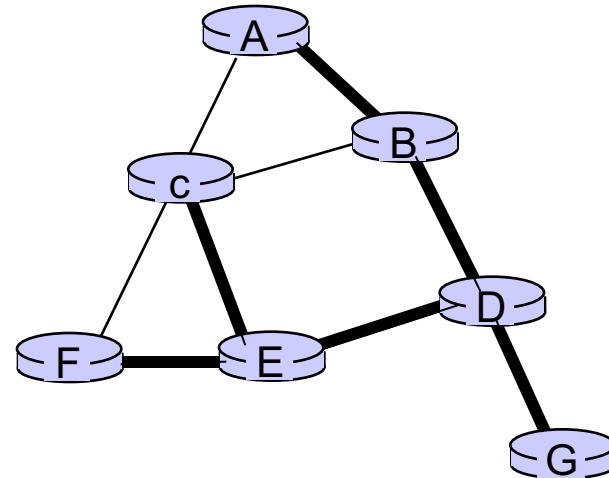
(b) broadcast initiated at D

# Spanning tree: creation

- ❖ center node
- ❖ each node sends unicast join message to center node
  - message forwarded until it arrives at a node already belonging to spanning tree



(a) stepwise construction of spanning tree (center: E)



(b) constructed spanning tree

# 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

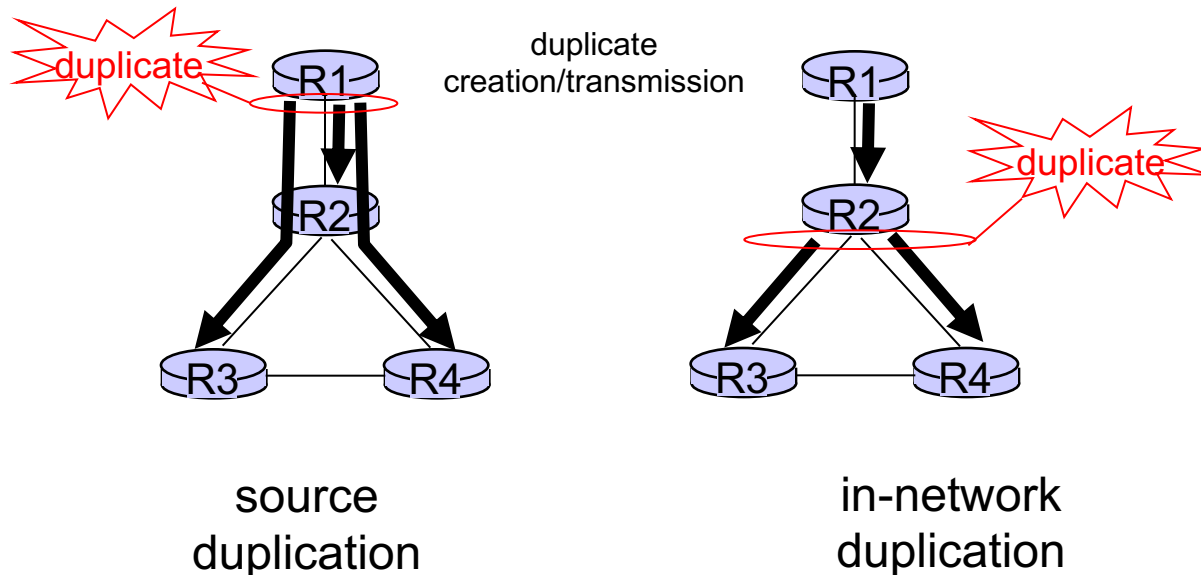
## 4.6 routing in the Internet

- RIP
- OSPF
- BGP

## 4.7 broadcast and multicast routing

# Broadcast routing

- ❖ deliver packets from source to all other nodes
- ❖ source duplication is inefficient:



- ❖ source duplication: how does source determine recipient addresses?

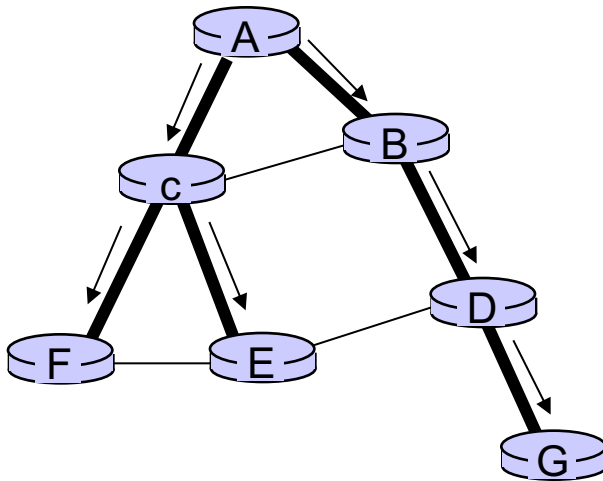


# In-network duplication

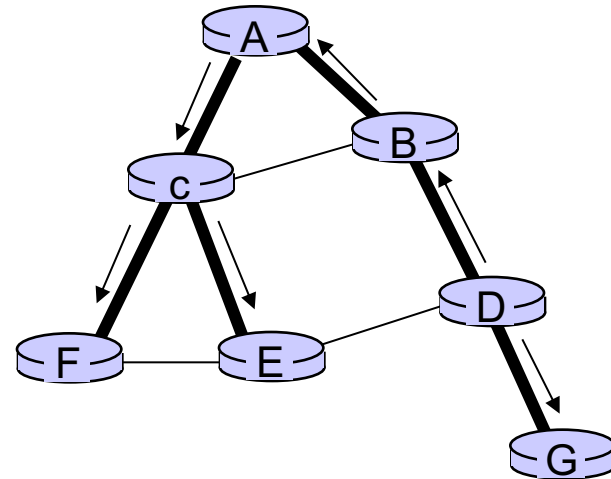
- ❖ *flooding*: when node receives broadcast packet, sends copy to all neighbors
  - problems: cycles & broadcast storm
- ❖ *controlled flooding*: node only broadcasts pkt if it hasn't broadcast same packet before
  - node keeps track of packet ids already broadcasted
  - or reverse path forwarding (RPF): only forward packet if it arrived on shortest path between node and source
- ❖ *spanning tree*:
  - no redundant packets received by any node

# Spanning tree

- ❖ first construct a spanning tree
- ❖ nodes then forward/make copies only along spanning tree



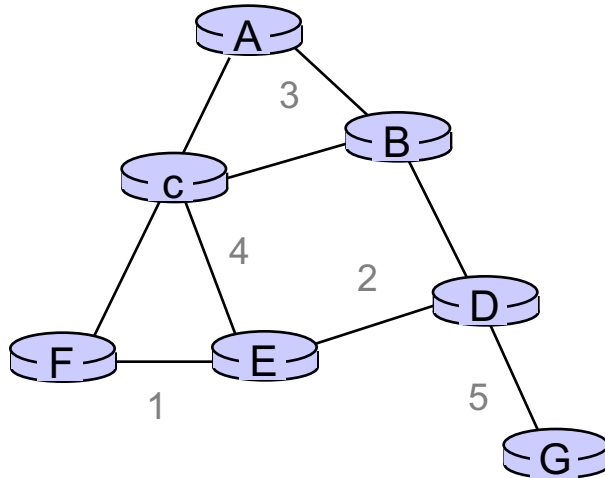
(a) broadcast initiated at A



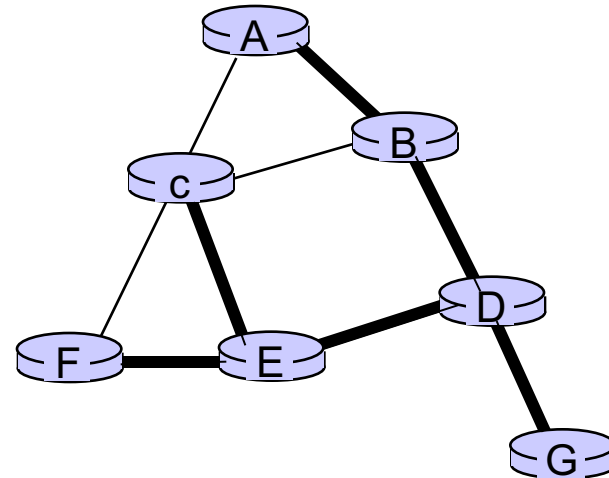
(b) broadcast initiated at D

# Spanning tree: creation

- ❖ center node
- ❖ each node sends unicast join message to center node
  - message forwarded until it arrives at a node already belonging to spanning tree



(a) stepwise construction of spanning tree (center: E)



(b) constructed spanning tree

# Multicast routing: problem statement

*goal:* find a tree (or trees) connecting routers having local mcast group members

- ❖ *tree:* not all paths between routers used
- ❖ *shared-tree:* same tree used by all group members
- ❖ *source-based:* different tree from each sender to rcvrs

*legend*



*group member*



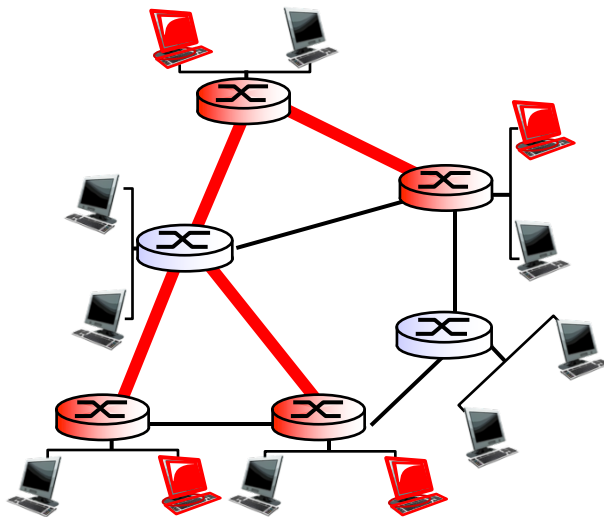
*not group member*



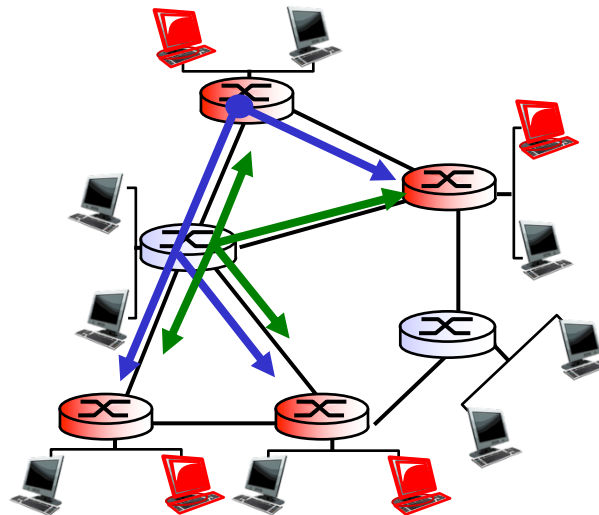
*router with a group member*



*router without group member*



shared tree



source-based trees

# Approaches for building mcast trees

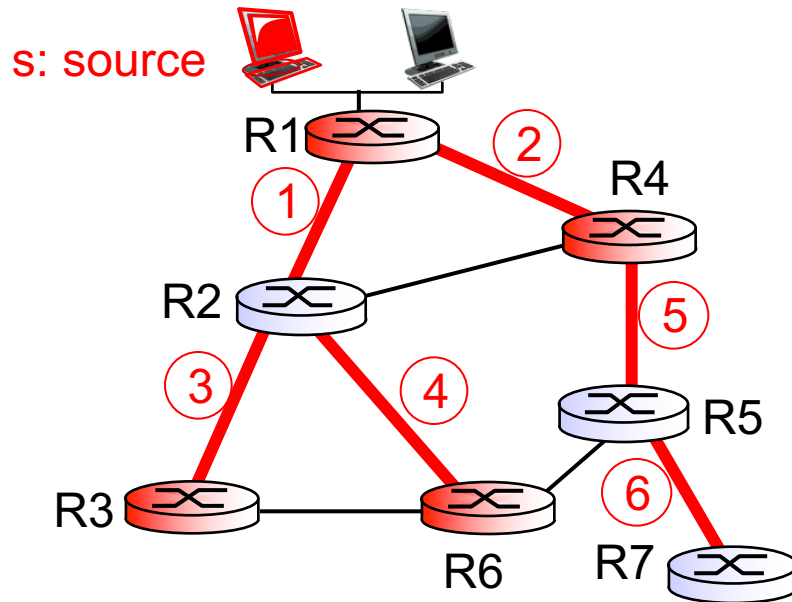
approaches:

- ❖ *source-based tree*: one tree per source
  - shortest path trees
  - reverse path forwarding
- ❖ *group-shared tree*: group uses one tree
  - minimal spanning (Steiner)
  - center-based trees

...we first look at basic approaches, then specific protocols adopting these approaches

# Shortest path tree

- ❖ mcast forwarding tree: tree of shortest path routes from source to all receivers
  - Dijkstra's algorithm



## LEGEND



router with attached group member



router with no attached group member



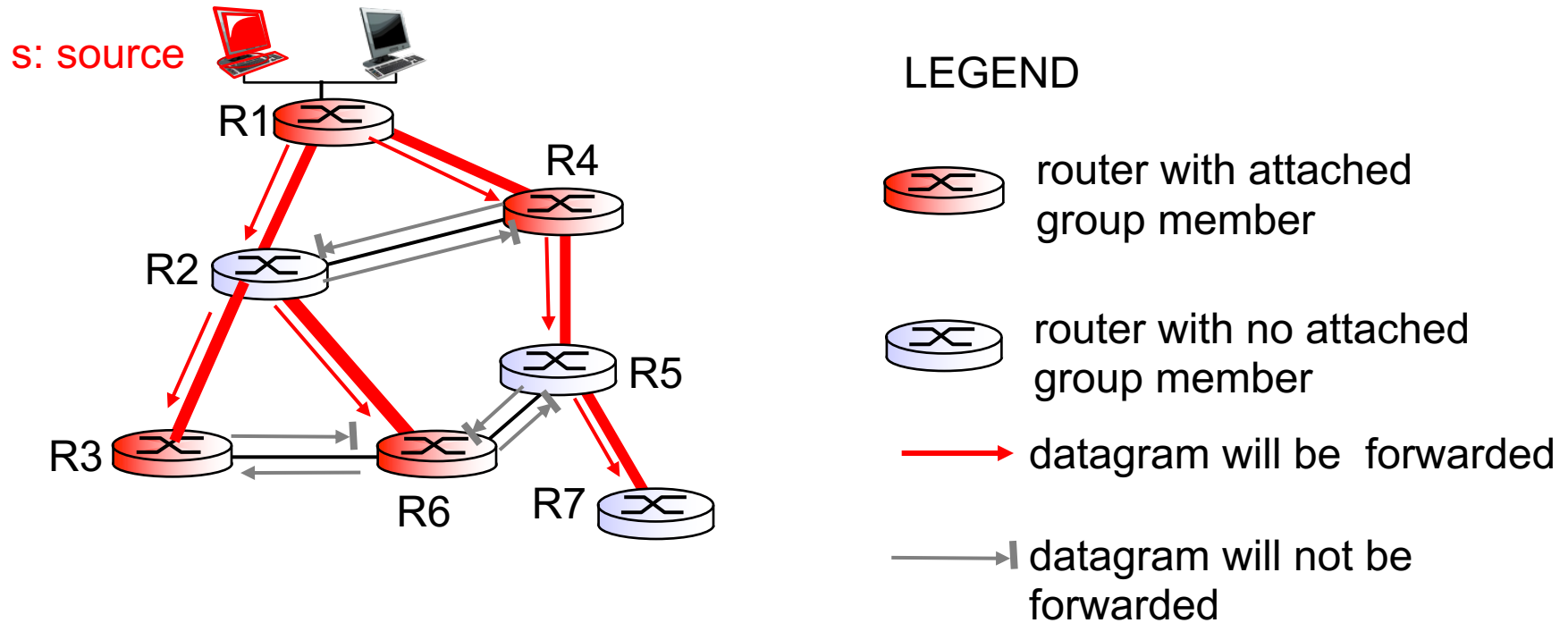
link used for forwarding, i indicates order link added by algorithm

# Reverse path forwarding

- ❖ rely on router's knowledge of unicast shortest path from it to sender
- ❖ each router has simple forwarding behavior:

***if*** (mcast datagram received on incoming link on shortest path back to center)  
***then*** flood datagram onto all outgoing links  
***else*** ignore datagram

# Reverse path forwarding: example

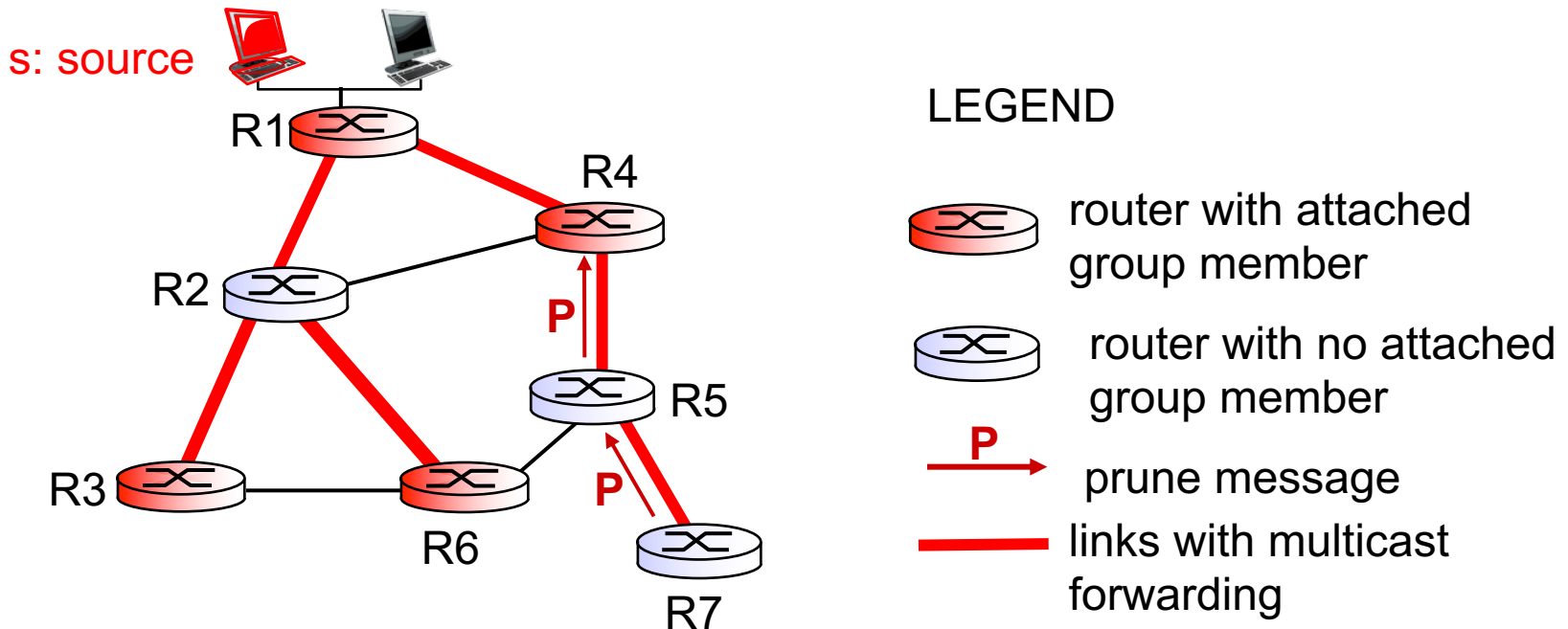


- ❖ result is a source-specific *reverse* SPT
  - may be a bad choice with asymmetric links



# Reverse path forwarding: pruning

- ❖ forwarding tree contains subtrees with no mcast group members
  - no need to forward datagrams down subtree
  - “prune” msgs sent upstream by router with no downstream group members



# Shared-tree: steiner tree

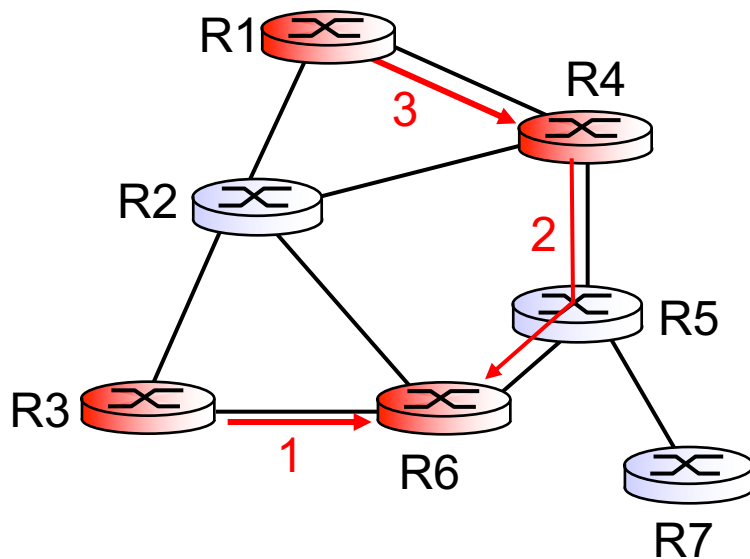
- ❖ *steiner tree*: minimum cost tree connecting all routers with attached group members
- ❖ problem is NP-complete
- ❖ excellent heuristics exists
- ❖ not used in practice:
  - computational complexity
  - information about entire network needed
  - monolithic: rerun whenever a router needs to join/leave

# Center-based trees

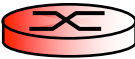


- ❖ single delivery tree shared by all
- ❖ one router identified as “*center*” of tree
- ❖ to join:
  - edge router sends unicast *join-msg* addressed to center router
  - *join-msg* “processed” by intermediate routers and forwarded towards center
  - *join-msg* either hits existing tree branch for this center, or arrives at center
  - path taken by *join-msg* becomes new branch of tree for this router

# Center-based trees: example

suppose R6 chosen as center:



## LEGEND

-  router with attached group member
-  router with no attached group member
-  path order in which join messages generated

# Internet Multicasting Routing: DVMRP

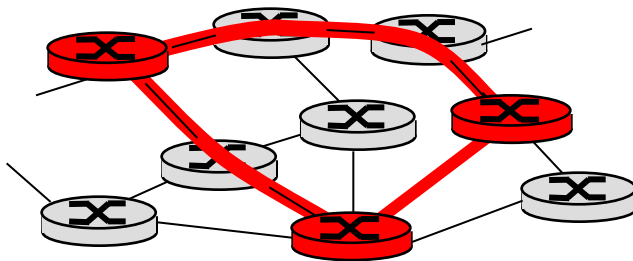
- ❖ **DVMRP**: distance vector multicast routing protocol, RFC1075
- ❖ *flood and prune*: reverse path forwarding, source-based tree
  - RPF tree based on DVMRP's own routing tables constructed by communicating DVMRP routers
  - no assumptions about underlying unicast
  - initial datagram to mcast group flooded everywhere via RPF
  - routers not wanting group: send upstream prune msgs

# DVMRP: continued...

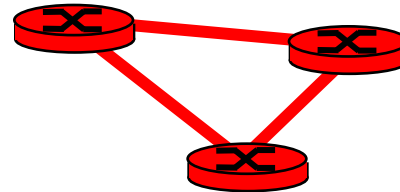
- ❖ *soft state*: DVMRP router periodically (1 min.) “forgets” branches are pruned:
  - mcast data again flows down unpruned branch
  - downstream router: reprune or else continue to receive data
- ❖ routers can quickly regraft to tree
  - following IGMP join at leaf
- ❖ odds and ends
  - commonly implemented in commercial router

# Tunneling

**Q:** how to connect “islands” of multicast routers in a “sea” of unicast routers?



physical topology



logical topology

- ❖ mcast datagram encapsulated inside “normal” (non-multicast-addressed) datagram
- ❖ normal IP datagram sent thru “tunnel” via regular IP unicast to receiving mcast router (recall IPv6 inside IPv4 tunneling)
- ❖ receiving mcast router unencapsulates to get mcast datagram

# PIM: Protocol Independent Multicast

- ❖ not dependent on any specific underlying unicast routing algorithm (works with all)
- ❖ two different multicast distribution scenarios :

## *dense:*

- ❖ group members densely packed, in “close” proximity.
- ❖ bandwidth more plentiful

## *sparse:*

- ❖ # networks with group members small wrt # interconnected networks
- ❖ group members “widely dispersed”
- ❖ bandwidth not plentiful



# Consequences of sparse-dense dichotomy:

## *dense*

- ❖ group membership by routers *assumed* until routers explicitly prune
- ❖ *data-driven* construction on mcast tree (e.g., RPF)
- ❖ bandwidth and non-group-router processing *profligate*

## *sparse:*

- ❖ no membership until routers explicitly join
- ❖ *receiver-driven* construction of mcast tree (e.g., center-based)
- ❖ bandwidth and non-group-router processing *conservative*

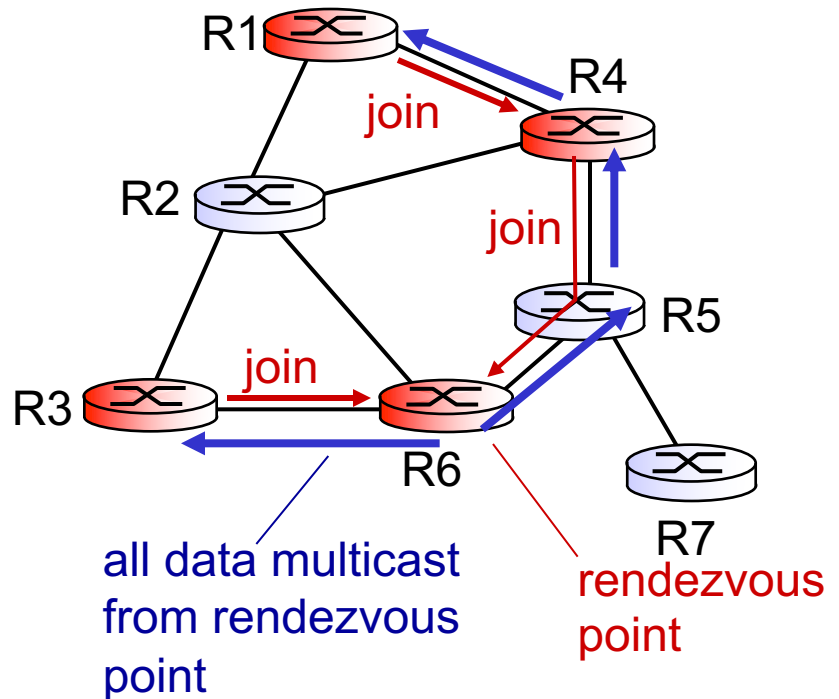
# PIM- dense mode

**flood-and-prune RPF:** similar to DVMRP but...

- ❖ underlying unicast protocol provides RPF info for incoming datagram
- ❖ less complicated (less efficient) downstream flood than DVMRP reduces reliance on underlying routing algorithm
- ❖ has protocol mechanism for router to detect it is a leaf-node router

# PIM - sparse mode

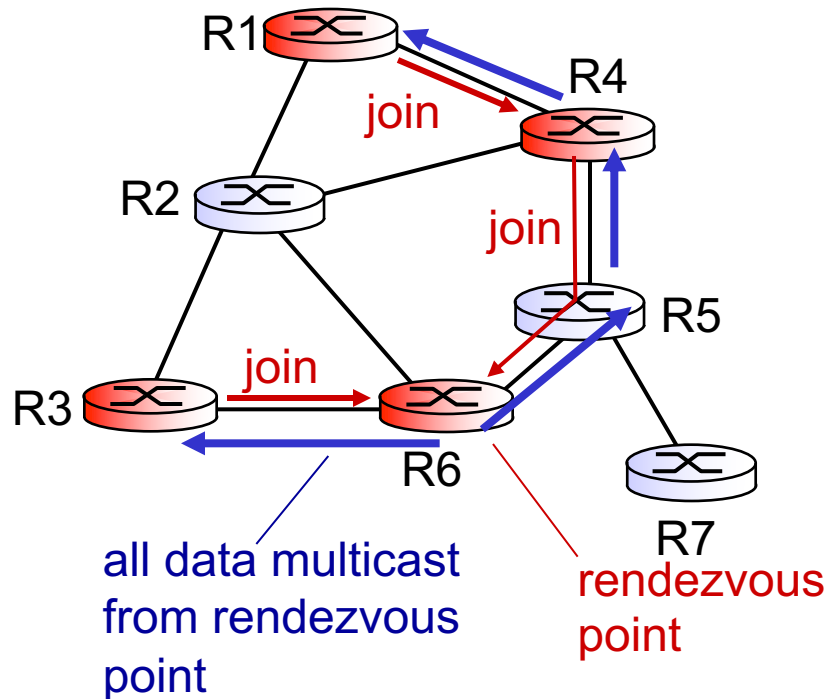
- ❖ center-based approach
- ❖ router sends *join* msg to rendezvous point (RP)
  - intermediate routers update state and forward *join*
- ❖ after joining via RP, router can switch to source-specific tree
  - increased performance: less concentration, shorter paths



# PIM - sparse mode

## *sender(s):*

- ❖ unicast data to RP, which distributes down RP-rooted tree
- ❖ RP can extend mcast tree upstream to source
- ❖ RP can send *stop* msg if no attached receivers
  - “no one is listening!”



# Chapter 4: *done!*

## 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

- ❖ understand principles behind network layer services:
  - network layer service models, forwarding versus routing  
how a router works, routing (path selection), broadcast, multicast
- ❖ instantiation, implementation in the Internet