Chapter 22

Network Layer: Delivery, Forwarding, and Routing

- **Delivery** refers to the process of getting a packet from a source to its destination under the control of the network layer.

- **Forwarding** refers to the way of moving a packet to the next station or router along its path.

- **Routing** refers to the way routing tables are created to determine the best path to help in forwarding.

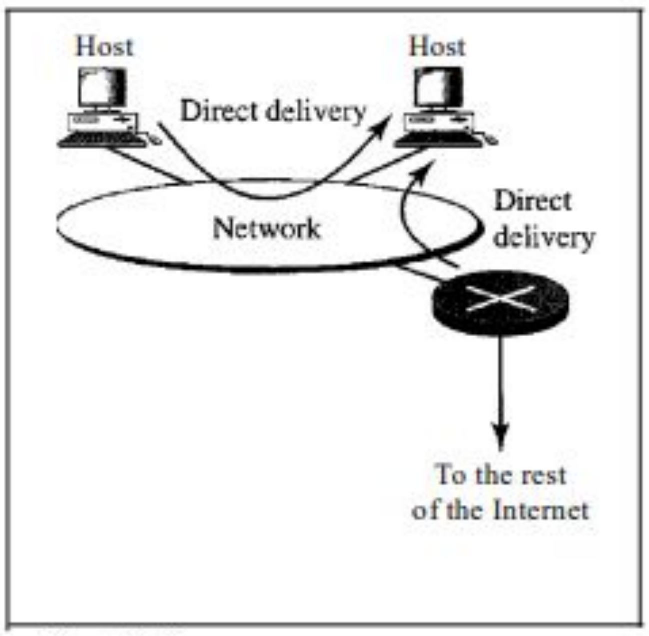
Direct Versus Indirect Delivery

- **Direct Delivery**
- In a direct delivery, the final destination of the packet is a host connected to the same physical network as the deliverer.
- Direct delivery occurs when the source and destination of the packet are located on the same physical network or when the delivery is between the last router and the destination host.

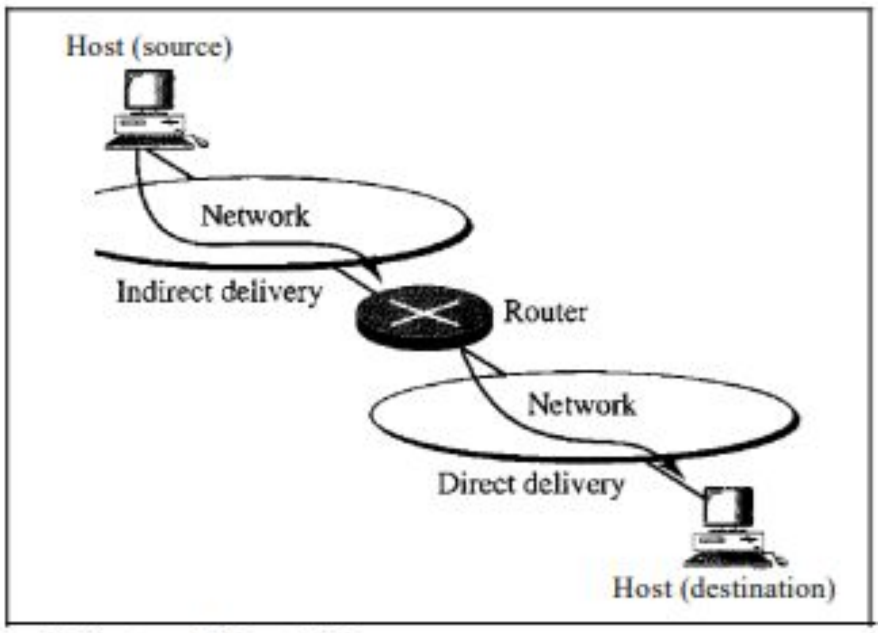
Indirect Delivery

- If the destination host is not on the same network as the deliverer, the packet is delivered indirectly.

- In an indirect delivery, the packet goes from router to router until it reaches the one connected to the same physical network as its final destination.



a. Direct delivery



b. Indirect and direct delivery

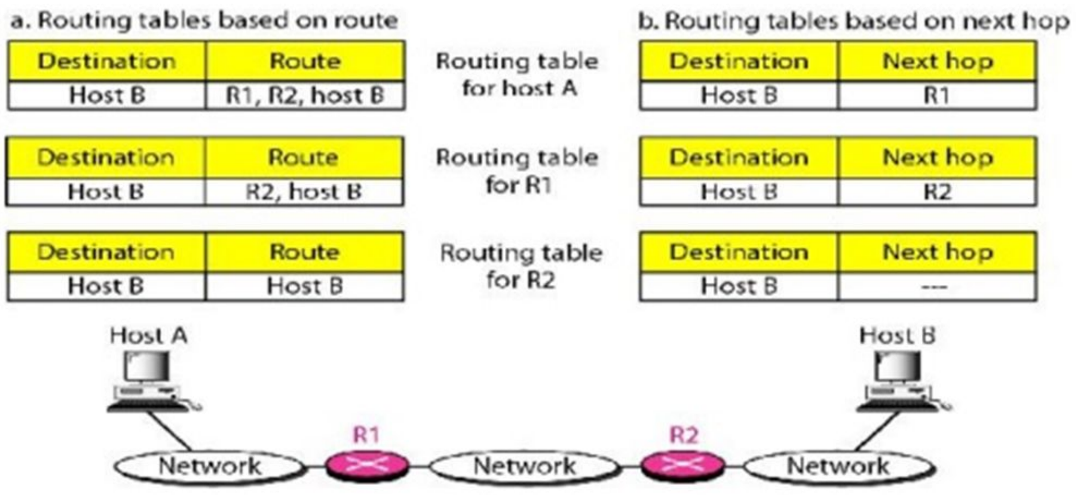
22.5

Forwarding

- Forwarding means to place the packet in its route to its destination.
- Forwarding requires a host or a router to have a routing table.
- When a host has a packet to send or when a router has received a packet to be forwarded, it looks at this table to find the route to the final destination.

Forwarding Techniques

- Next-Hop Method Versus Route Method



a. Routing tables based on route

Destination	Route
Host B	R1, R2, host B

Routing table for host A

Destination	Next hop
Host B	R1

b. Routing tables based on next hop

Destination	Route
Host B	R2, host B

Routing table for R1

Destination	Next hop
Host B	R2

Destination	Route
Host B	Host B

Routing table for R2

Destination	Next hop
Host B	---

Figure 3.40 Route method versus next-hop method

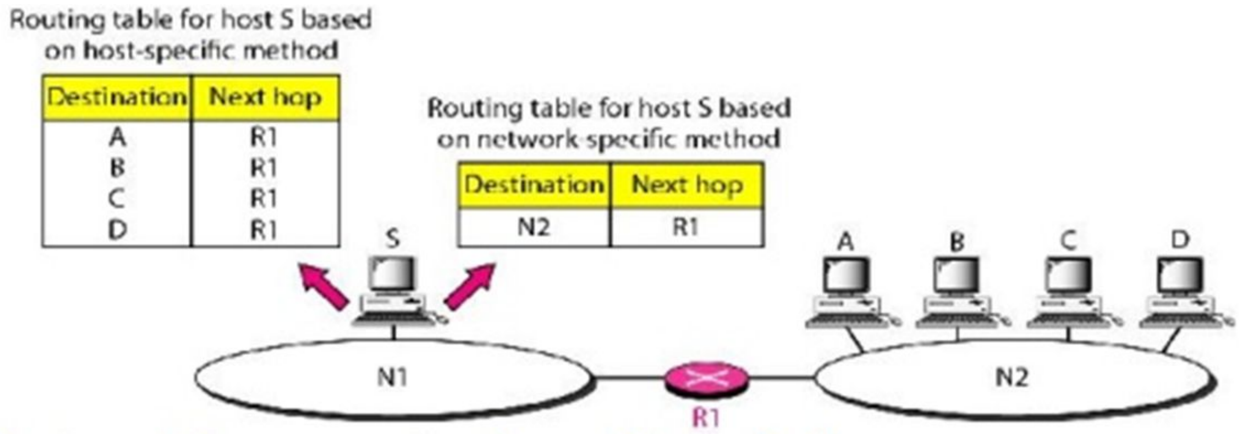
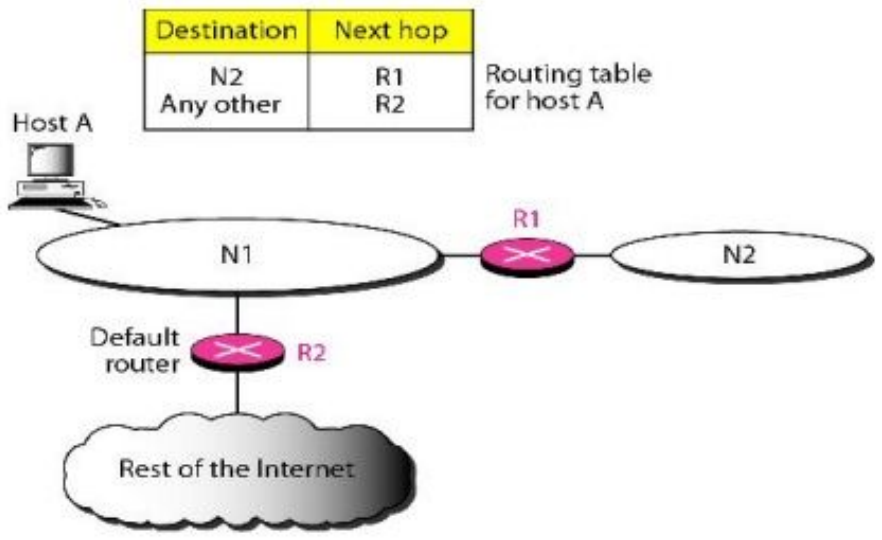

Routing table for host S based on host-specific method
Destination
Next hop
A
R1
B
R1
C
R1
D
R1
Routing table for host S based on network-specific method
Destination
Next hop
N2
R1
S
A
B
C
D
N1
R1
N2

Figure 3.41 Host-specific versus network-specific method

22.8

Default Method

- Another technique to simplify routing is called the default method.

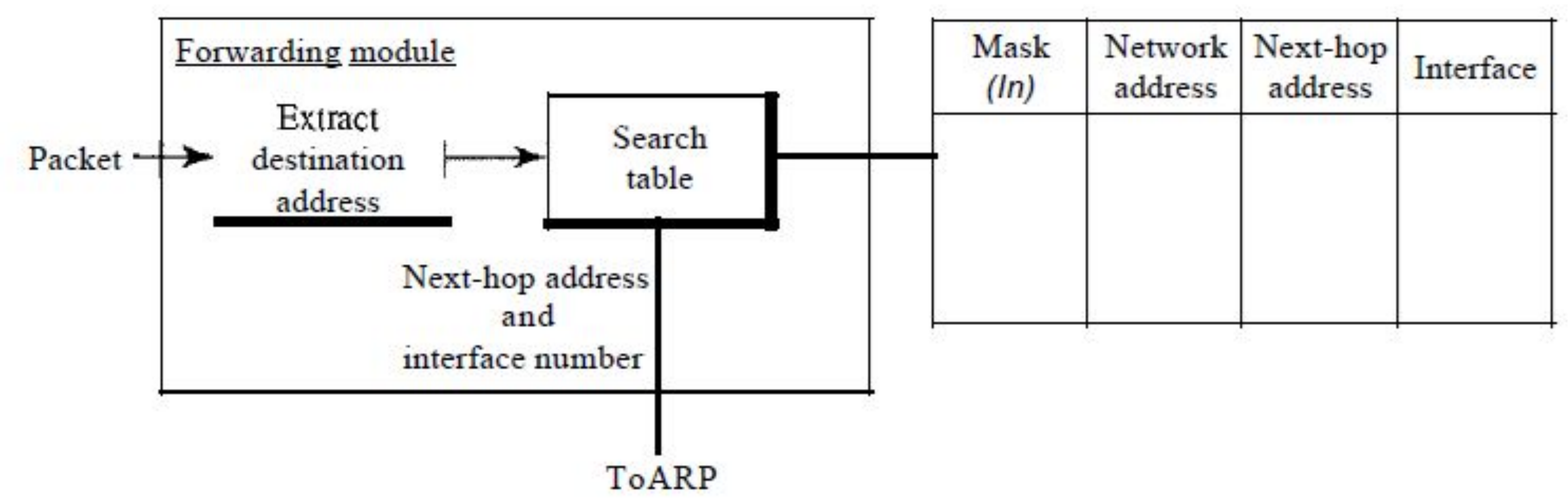


Destination	Next hop
N2	R1
Any other	R2

Figure 3.42 Default method

Forwarding Process

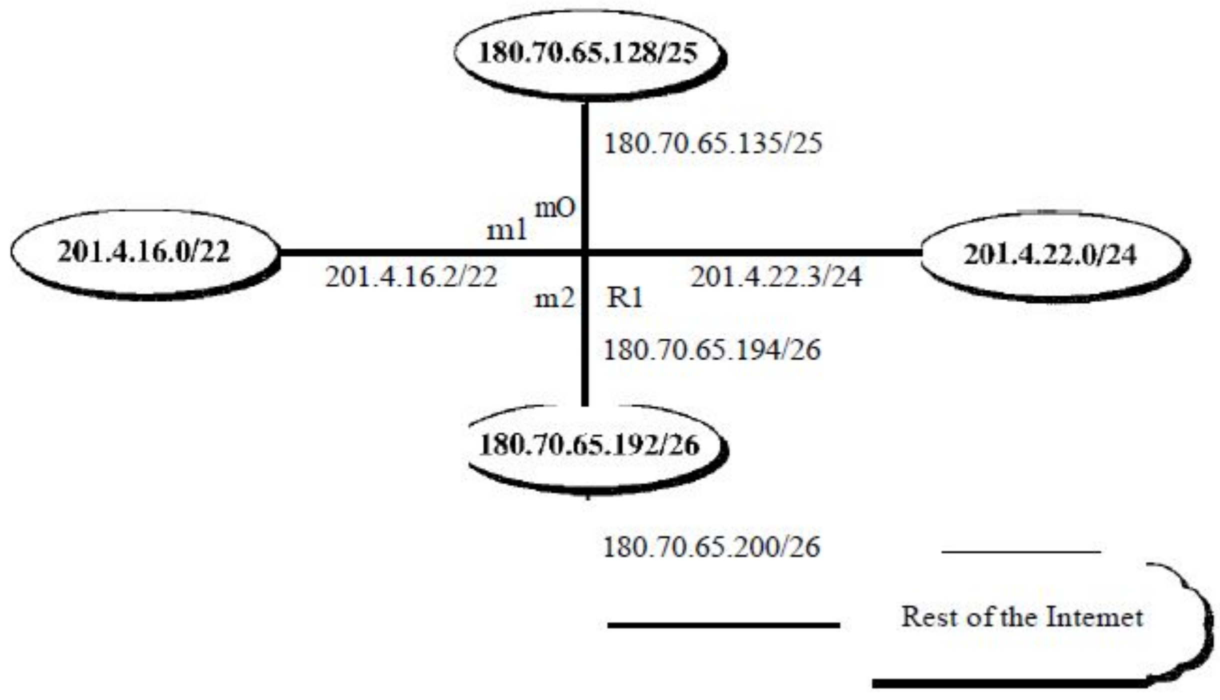
- Simplified forwarding module in classless address.



In classless addressing, we need at least four columns in a routing table.

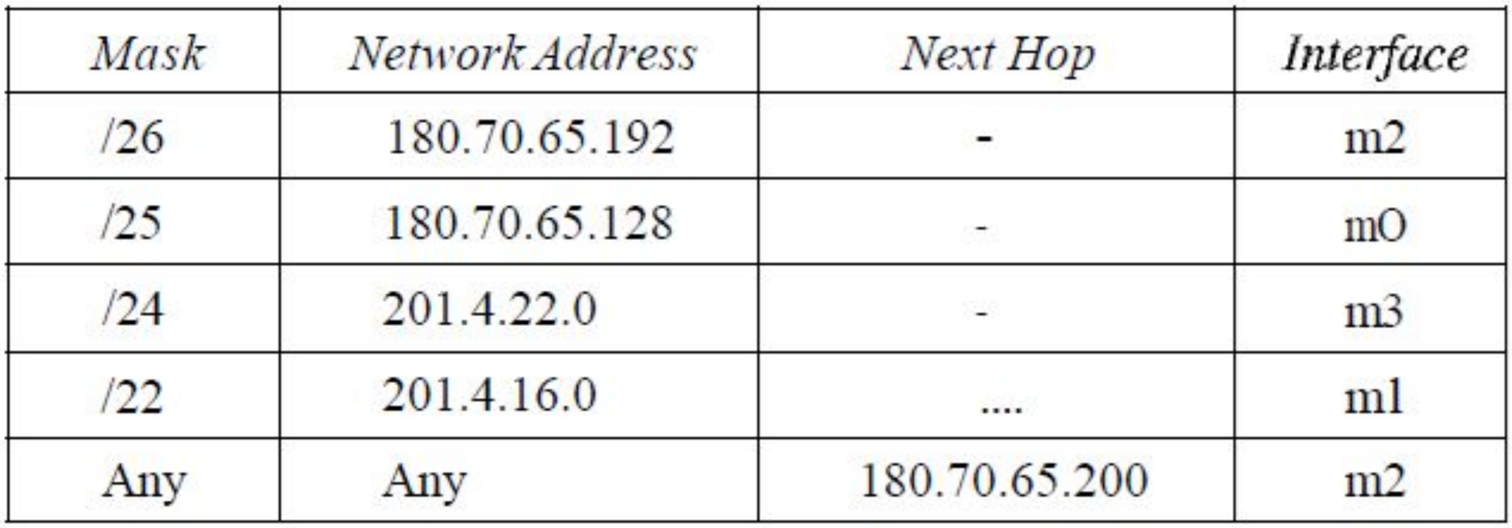
Example 22.1

- Make a routing table for router Rl, using the configuration in Figure 22.6.



Solution

Table 22.1 *Routing table for router RI in Figure 22.6*



Mask	*Network Address*	*Next Hop*	*Interface*
/26	180.70.65.192	-	m2
/25	180.70.65.128	-	mO
/24	201.4.22.0	-	m3
/22	201.4.16.0	....	ml
Any	Any	180.70.65.200	m2

22-3 UNICAST ROUTING PROTOCOLS

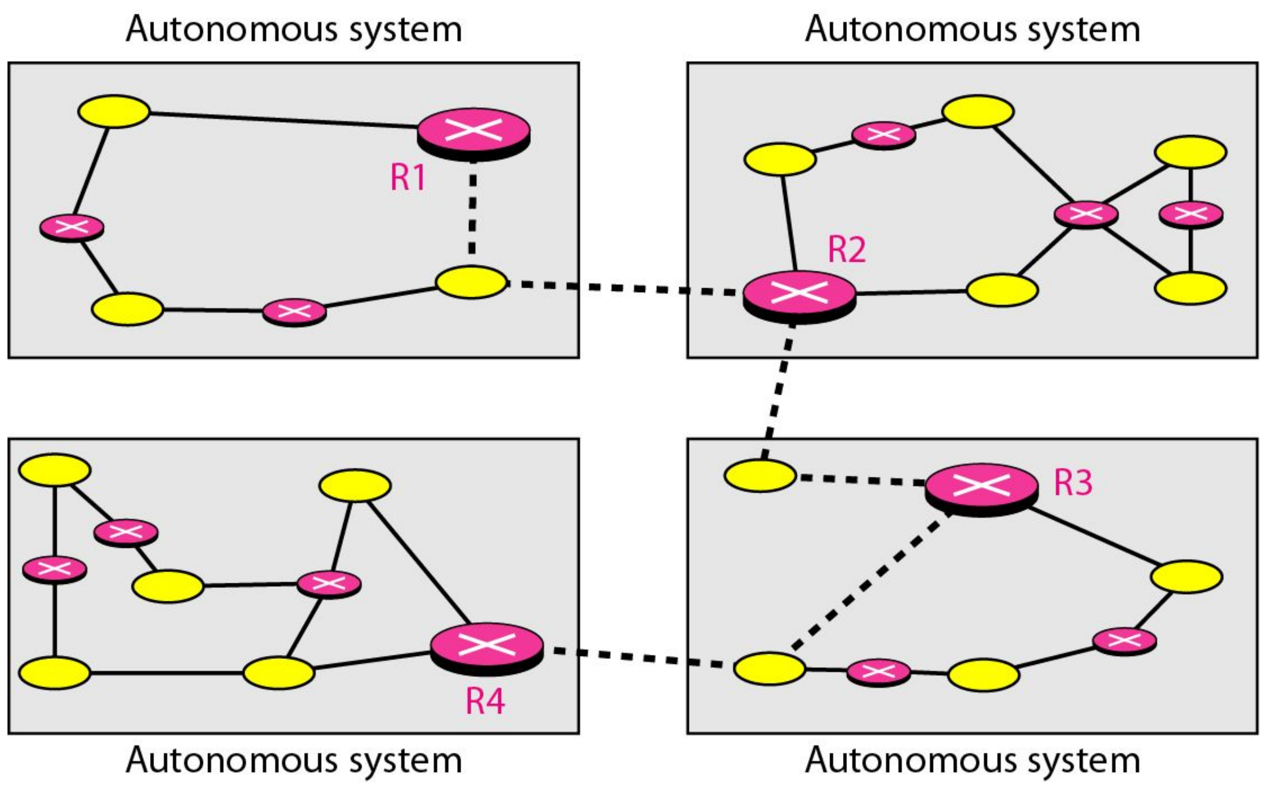
A routing table can be either static or dynamic. A static table is one with manual entries. A dynamic table is one that is updated automatically when there is a change somewhere in the Internet. A routing protocol is a combination of rules and procedures that lets routers in the Internet inform each other of changes.

Topics discussed in this section:

Optimization
Intra- and Interdomain Routing
Distance Vector Routing and RIP
Link State Routing and OSPF
Path Vector Routing and BGP

- Desirable properties of Routing Algorithms:
  - Correctness (applicable to all)
  - Simplicity (applicable to all)
  - Robustness: able to cope up with
    - changes in topology, load.
    - hardware and software failures
  - Stability (hard to achieve)
    - Converge to equilibrium
  - Fairness (conflicting)
  - Optimality (conflicting) see next **fig**.
- Types of Routing Algorithms:
  - **~~Non-Adaptive:~~ Static. Routing decisions computed in advance, off-line and downloaded.**
  - Adaptive: Dynamic. Adaptive to the changes in topology **and load. Issue here is how to get the information? Locally, From adjacent routers, from all routers?**

Figure 22.12 *Autonomous systems*

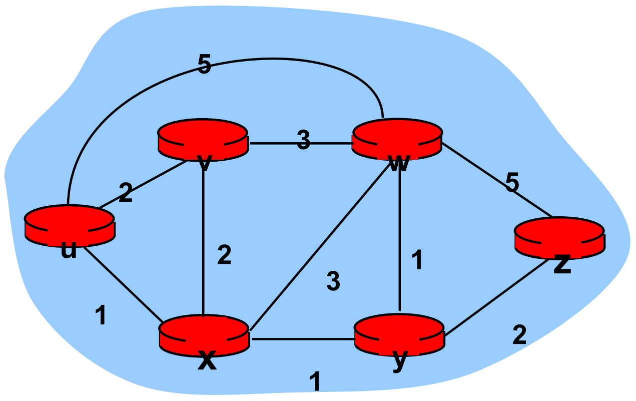


Routing protocols

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)

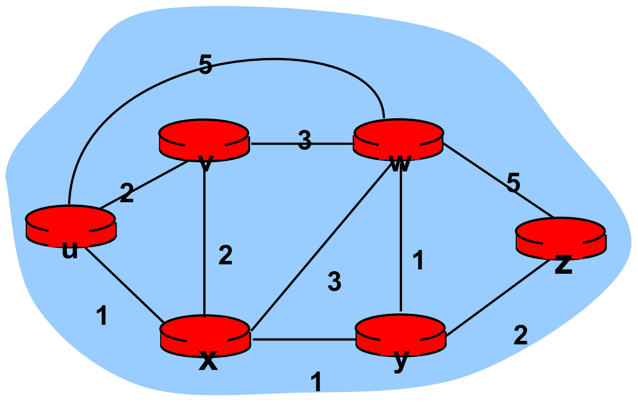
N = set of routers = { u, v, w, x, y, z }

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

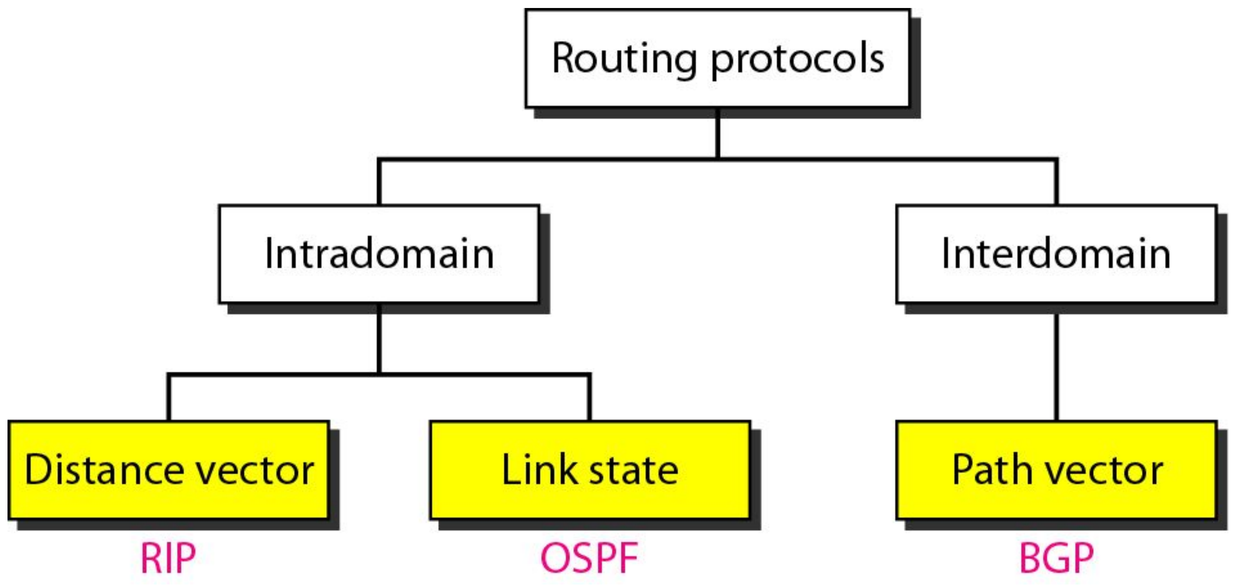
cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

cost of path (x1, x2, x3,…, xp) = c(x1,x2) + c(x2,x3) + … + c(xp-1,xp)

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

routing algorithm: algorithm that finds that least cost path

Figure 22.13 *Popular routing protocols*



Distance Vector Routing:

- So far we have studied Static Routing Algorithms.
- But practically dynamic Routing Algorithms are used.
- Following two are Dynamic Routing Algorithms:
  - 1. Distance Vector Routing Algorithm.
  - 2. Link State Routing Algorithm.
- **Distance Vector Routing Algorithm:**
- At each step within a router:
  - Get routing tables from neighbours
  - Compute distance to neighbours
  - Compute new routing table

1. Router transmits its *distance vector* to each of its neighbors.
2. Each router receives and saves the most recently received *distance vector* from each of its neighbors.
3. A router **recalculates** its distance vector when:
   a. It receives a *distance vector* from a neighbor containing different information than before.
   b. It discovers that a link to a neighbor has gone down (i.e., a topology change).

The DV calculation is based on minimizing the cost to each destination.

The distance vector routing algorithm is sometimes called by other names, the **distributed Bellman-Ford** routing algorithm and the **Ford-Fulkerson algorithm.**

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

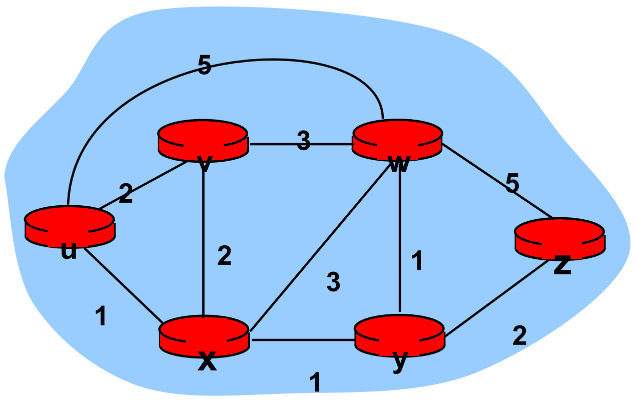
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

***min* taken over all neighbors v of x**

Bellman-Ford example



clearly, dv(z) = 5, dx(z) = 3, dw(z) = 3

B-F equation says:

du(z) = min { c(u,v) + dv(z),
c(u,x) + dx(z),
c(u,w) + dw(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:

$$Dx(y) \leftarrow minv\{c(x,v) + Dv(y)\} \quad \text{for each node } y \in N$$

- **under minor, natural conditions, the estimate *Dx(y) converge to the actual least cost* dx(y)**

Distance vector algorithm

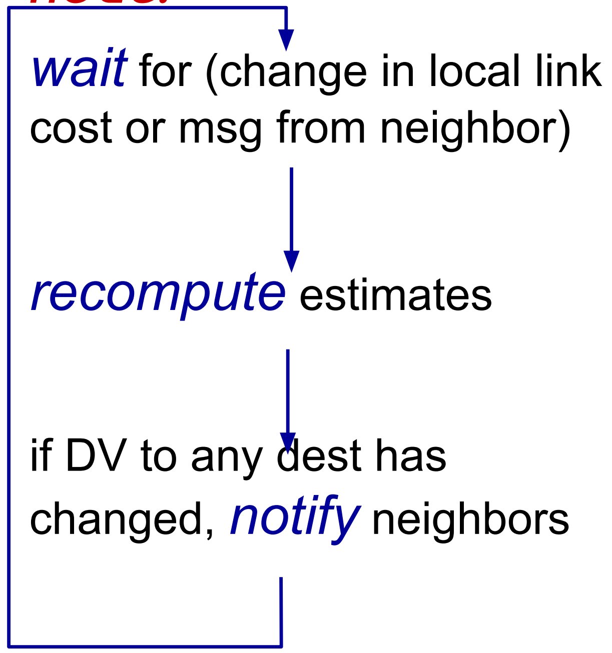
iterative, asynchronous: each local iteration caused by:

- 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:



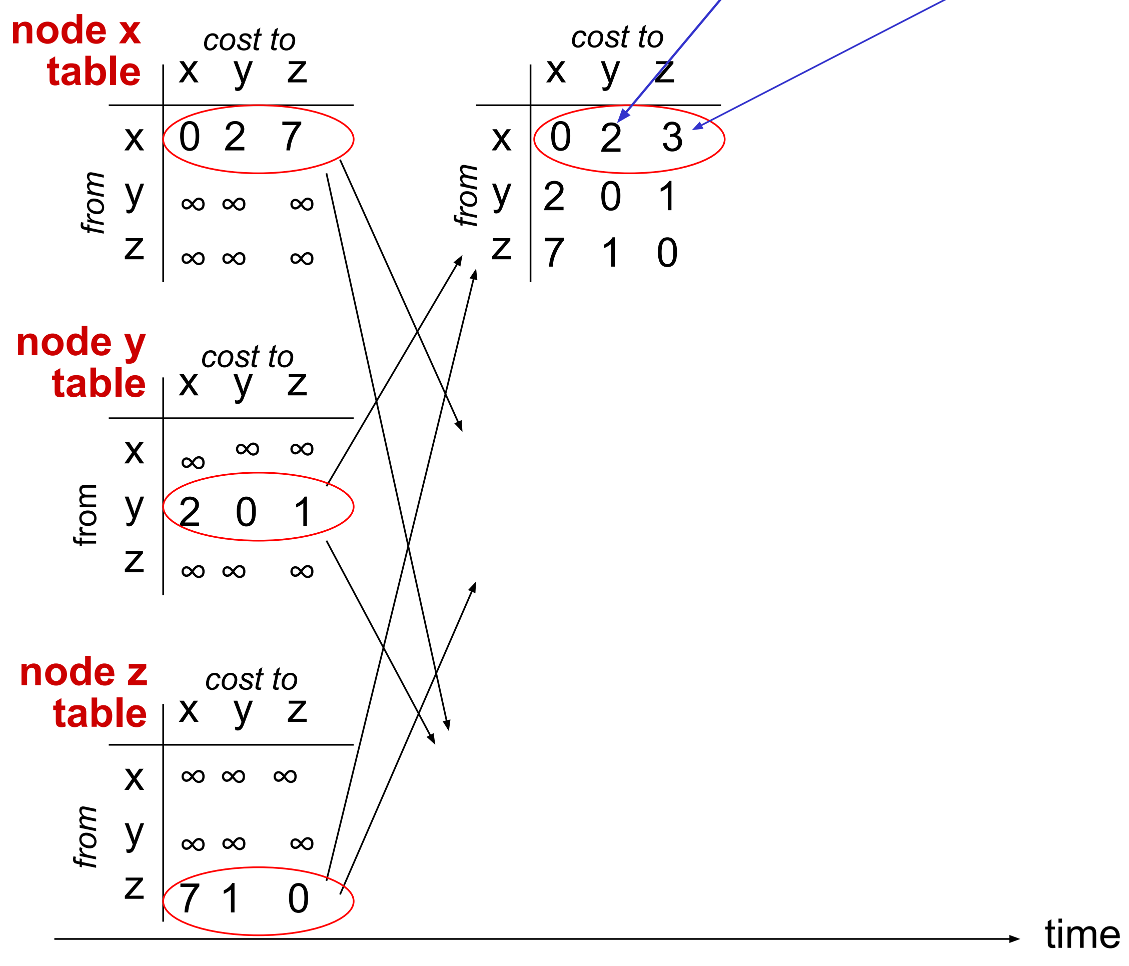
wait for (change in local link cost or msg from neighbor)

recompute estimates

if DV to any dest has changed, *notify* neighbors

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

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



node x table

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

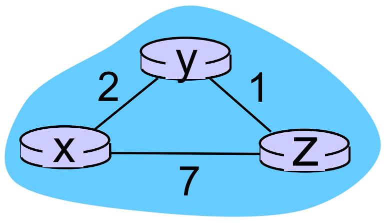
from \ *cost to*	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

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

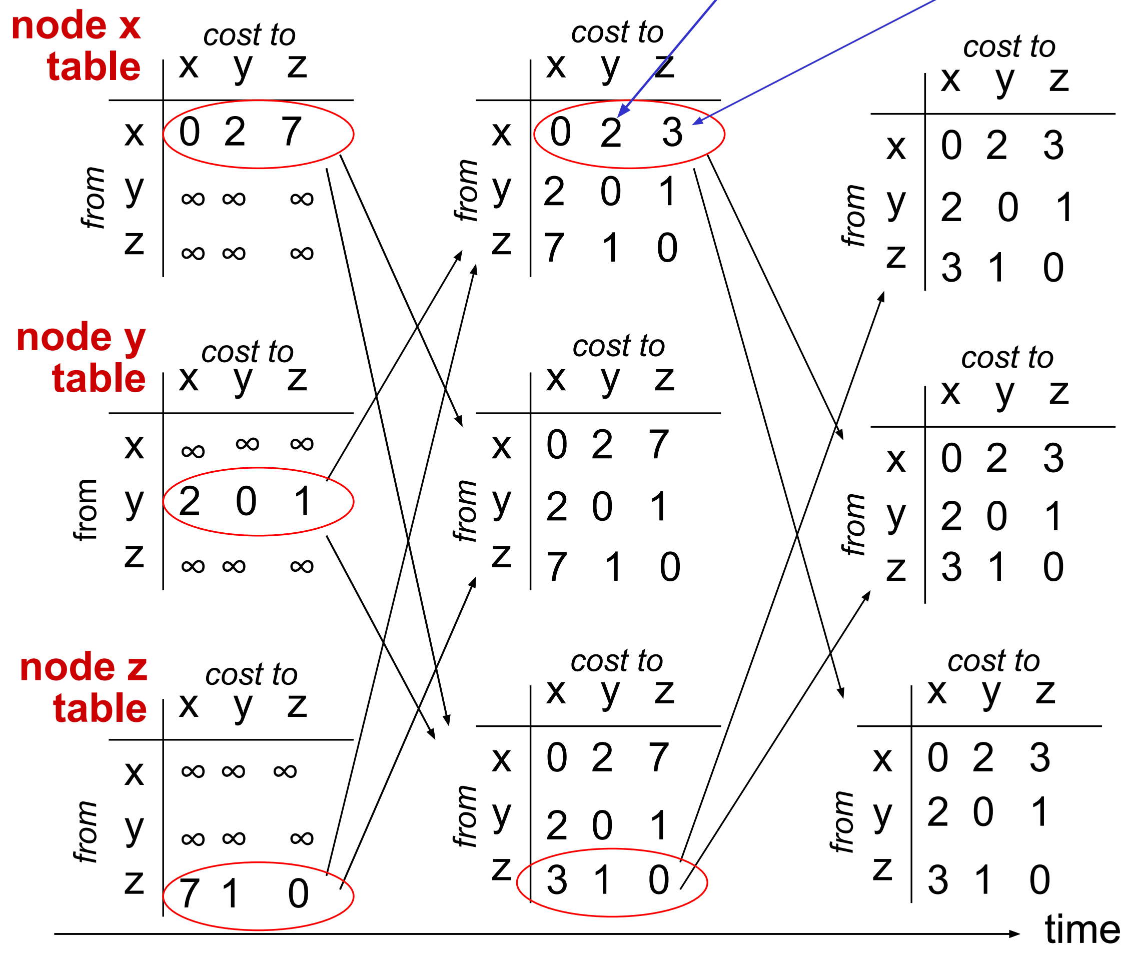
node z table

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



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

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



node x table

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

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

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

node y table

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

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

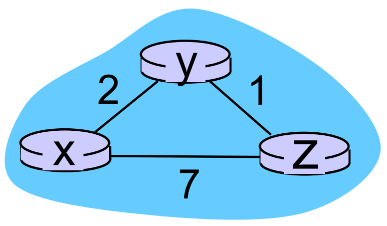
	cost to		
from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

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

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

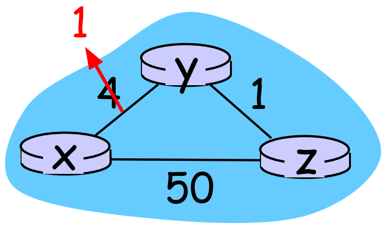
	cost to		
from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



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.

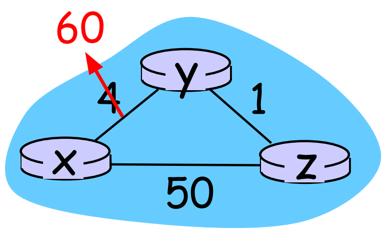
t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

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?

- Drawback of Distance Vector Routing:
  - Count to Infinity Problem:
    - It reacts rapidly to good news,
    - But, leisurely to bad news.
    - Updates value fast when neighbor is down, but not when neighbor is again up. How?
    - Lie to neighbour about distance
      if routing via neighbour
    - The core of the problem is that when X tells Y that it has a path somewhere, Y has no way of knowing whether it itself is on the path? This is how problem is created.
  - It does not take bandwidth into account.
  - Take too long to converge changes in one node to all other nodes.
- Solution?
- Link State Routing.

Link State Routing

Each router must do the following:

1. Discover its neighbors, learn their network address.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

A complete topology is developed. Then Dijkstra's Algorithm can be used to compute the shortest path.

Following 5 steps are followed to implement it.

1. Learning about the Neighbors
2. Measuring Line Cost.
3. Building Link State Packets.
4. Distributing the Link State Packets.
5. Computing the New Routes.

A link-state routing algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.'s

notation:

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

Dijsktra's algorithm

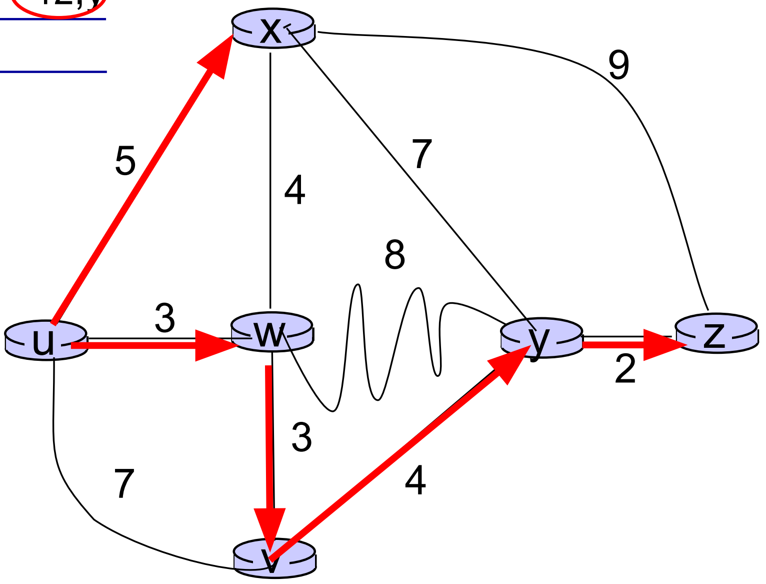
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then $D(v) = c(u,v)$
    else $D(v) = \infty$

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

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,y	14,x
4	uwxvy					12,y
5	uwxvyz					

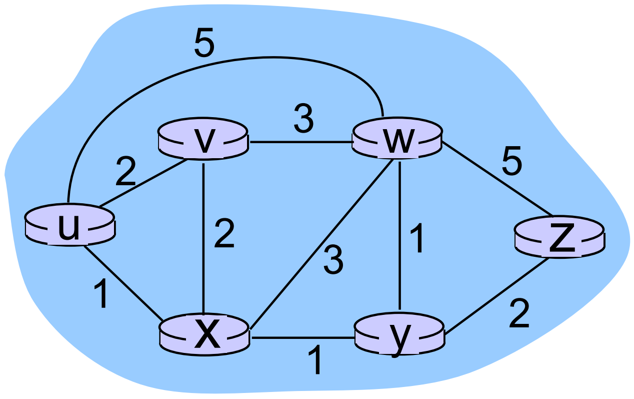
notes:

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



Dijkstra's algorithm: another example

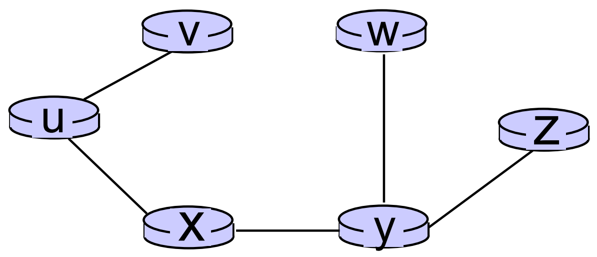
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

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)

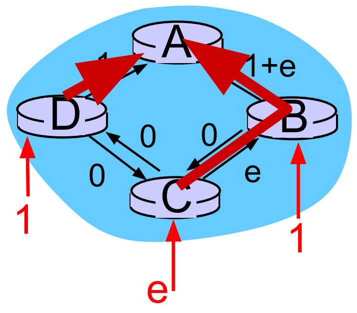
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

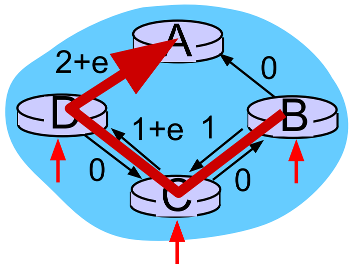
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: O(n^2)
- more efficient implementations possible: O(nlogn)

oscillations possible:

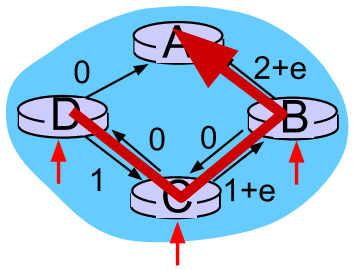
- e.g., support link cost equals amount of carried traffic:



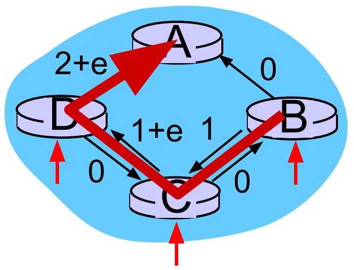
initially



given these costs, find new routing…. resulting in new costs

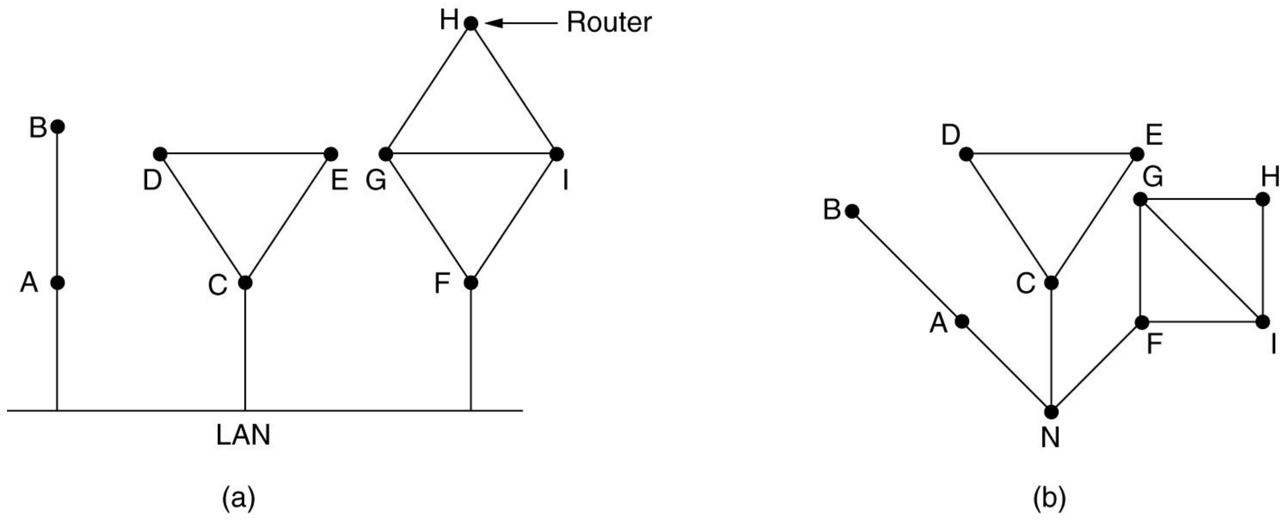


given these costs, find new routing…. resulting in new costs



given these costs, find new routing…. resulting in new costs

Learning about the Neighbors

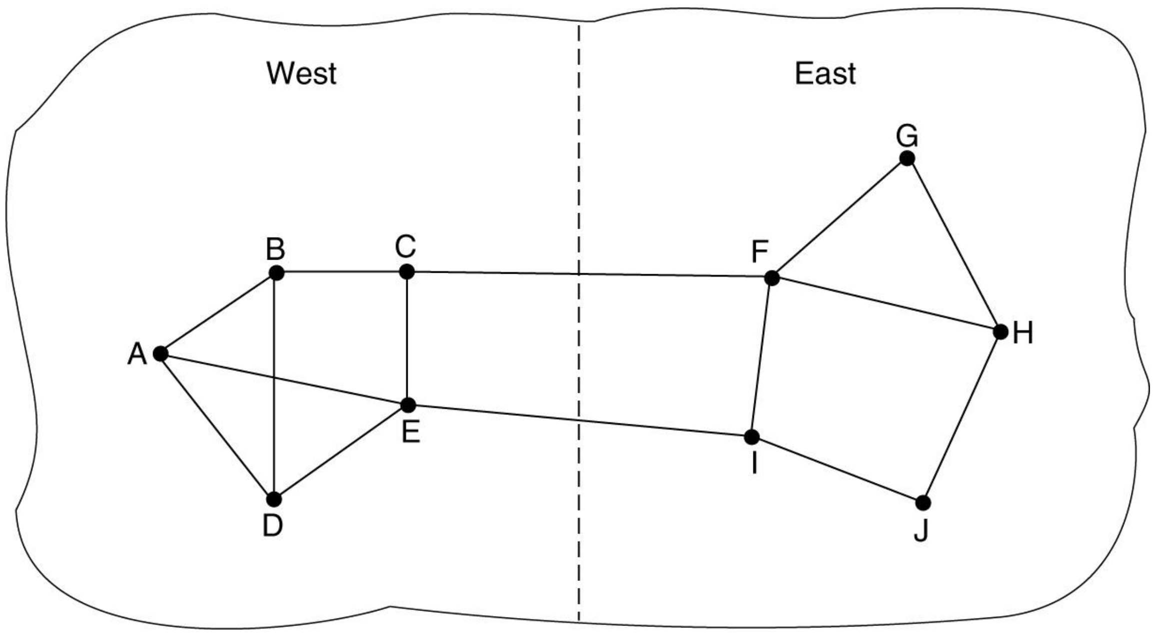


(a) Nine routers and a LAN. (b) A graph model of (a).

- Step 1: Learning about the Neighbours:
  - Upon boot of router,
    - Send HELLO packet on each point-to-point line
    - Routers are supposed to send reply with a globally unique name
- Step 2: Measuring the Line Cost:
  - Measure round-trip delay using ECHO Packet and wait for its reply
  - Take load into account? Yes. Arguments both ways: when choice is given to router having same number of hops from S to D.
    - Yes! preference for unloaded line as shortest path.

Measuring Line Cost

A subnet in which the East and West parts are connected by two lines.



- Step 3: Building Link State Packets:
  - Packet containing:
    - Identity of sender
    - Sequence number + age
    - For each neighbour:
    - name + distance
  - When to build the link state packets?
    - Periodically
    - when significant events occur
- See next figure.

- Step 4: Distributing Link State Packets:
- Distributing link state packets
  - Trickiest part of algorithm
    - Arrival time for packets different
    - How to keep consistent routing tables?
  - Basic algorithm
    - Flooding +
    - Sequence number (in each packet) to limit duplicates.
  - Manageable problems
    - Wrap around of sequence numbers results to wrong data. Solution? Use 32 bit sequence number.
    - Wrong sequence number used in case of :
      - lost in case of crash
      - Corrupted data transmitted.
  - Solution? include the age of each packet after the sequence number and decrement it once per second. When the age hits zero, the information from that router is discarded.
      - duplicates are discarded
      - Old packets are thrown out

Step 5: Computing new routes:

  - With a full set of link state packets, a router can:
    - Construct the entire subnet graph
    - Run Dijkstra’s algorithm to compute the shortest path to each destination
  - Problems for large subnets
    - Memory to store data
    - Compute time for developing these tables.
- Usage:
  - IS-IS protocol (Intermediate System, Intermediate System)
    - Designed for DECnet(digital equipment corporation network protocol suite), adopted by ISO(international standardization organization), used still in internet.
    - Supports multiple network layer protocols
  - OSPF(Open Shortest Path First) protocol used in Internet
  - Common features:
    - Self-stabilizing method of flooding link state updates
    - Concept of a designated router on a LAN
    - Method of computing and supporting path splitting and multiple metrics.
    - Useful in Multi Protocol Environment.

Link State	**Distance Vector**
link states algorithm is an algorithm **using global information**	the distance vector algorithm is **iterative, asynchronous, and distributed**
each node **talks with all other nodes**, but tell them only the cost of it's directly comparison of some of their attribute	each node **talks to only its directly connected neighbors**, but provides its neighbor with least cost estimates from itself to all the nodes.
Message complexity: With link state, every node has to keep the information about the cost of each link within the network.	**Message complexity**: with distance vector algorithm, message is exchanged between two hosts which are directly connected to each other.
very times, if any of the link cost is changed, all the nodes are **updated.**	change of cost in the link which is belong to the least cost path for one of the nodes, the DV algorithm will update the new value. But if the change doesn't belong to the least cost part between 2 hosts, there will **no updating**.
Speed of convergence: can converge faster in comparison of later.	**Speed of convergence**: can converge slowly and have routing loops while the algorithm is converging.
Such probability is less.	DV algorithm also suffers from the **count to infinity** problem.
Robustness: For LS, when a router is down, it can broadcast a wrong cost for the closest one. LS node is computing for its own forwarding table and other node do the calculation for themselves. **Better than DV.**	**Robustness:** DV, the wrong least cost path can be passed to more than one or all of the node so the wrong calculation will be process in the entire net work. This problem of DV is much **worse than LS algorithm**.

What does it mean to be the shortest (or optimal) route ?

a. <u>Minimize</u> mean packet <u>delay</u>
b. <u>Maximize</u> the network <u>throughput</u>
c. <u>Minimize</u> the <u>number of hops</u> along the path

Dijkstra algorithm:

- Each node is labeled (in parentheses) with its distance from the source node along the best known path.
- Initially, no paths are known, so all nodes are labeled with infinity.
- As the algorithm proceeds and paths are found, the labels may change, reflecting better paths.
- A label may be either **tentative** or **permanent**.
- Initially, all labels are tentative.
- When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

*Now lets see the **<u>algorithm</u>** which describes this procedure to develop shortest path from source to destination.*