CSC8004 Computer Networks

The Network Layer (Routing)

Routing

- Routing algorithms are responsible for deciding which output line an incoming packet should be transmitted on.
- Routing has two distinct phases
  1. Routing algorithm: Routers exchange information to decide on best routes and store routes in tables
  2. Forwarding: Routers forward packets according to what the tables say

Routing Choice

- Most hosts have many outbound lines, so the routing algorithm must pick the 'best' among them.
- This depends upon
  - speed of interconnections (avoid slow lines)
  - apparent load of target host (avoid slow machines)
  - congestion (avoid busy routes)

Routing Choice

- The selection of route depends on some performance criterion.
- The simplest criterion is to chose the minimum hop route, i.e. the route with the fewest intermediate nodes, should two or more such routes exist a probabilistic choice is made.
- A generalisation of this criterion is least-cost routing: each hop has an associated cost, the route chosen will be the one whose cumulative hops cost the least.

Routing Classification

Routing algorithms can be classified as follows:

- <u>Adaptive algorithms</u>, which attempt to adjust to traffic changes. Measurements are made of the network traffic during network operation, and changes are made to the algorithm if necessary. These algorithms are also called <u>dynamic</u>.

- <u>Non-adaptive algorithms</u> which do not adjust to traffic changes. In this case, some strategy is worked out beforehand, and it is not changed during network operation. Such algorithms are also called <u>static</u>.

Routing strategies – Fixed Routing

In a **fixed** strategy there is a single route between each pair of nodes.

- Each node has a table which maps the destination of a datagram to the next node on the route.
- Alternatively the same information can be stored in a single matrix held centrally.
- A fixed strategy works well for a reliable network with a steady workload.
- In such a strategy there is no difference between a datagram and virtual circuit approach.

Routing strategies-Flooding

- The source sends a packet to every one of its neighbours, each of which sends the packet on to every one of its neighbours (except the source, and so on.
- Eventually the destination will receive at least one copy of the packet.
- In order for the flood to stop a node must remember every packet it has seen, so if it receives another copy it will not pass it on again.
- Although flooding clearly uses a lot of network resources it does satisfy the routing requirements, also because every possible route will be tried, the shortest path will be found.

Routing strategies-Random routing

- Has a similar simplicity to flooding, but uses far less resources.
- A node simply selects the next node at random (or in rotation) from all its neighbours (excluding the previous node in the route);
- the probabilities of the choice may be weighted such that there is a preferred next node.
- A random strategy is good at distributing load around a network, but it is unlikely that such a strategy will deliver a packet by the best possible route.

Routing Strategies - Adaptive routing

- One of the strengths of datagram routing is the ability to react to altering network conditions, thus the outcome of the performance criterion can depend on both the time and location of the decision.
- Virtually all packet switched networks employ some kind of adaptive routing where the decisions are made subject to the presence of failures and congestion. For such a strategy to work information about the state of the network must be exchanged between the nodes.

Disadvantages of adaptive routing

- The decision process is more complex;
- Propagating information increases network load;
- Too sensitive adaptation may produce oscillating reactive effects.
- Temporary fluctuations may produce long term route changes that are difficult to predict.

Link State Routing (LSR)
(Using Dijstra’s Algorithm)

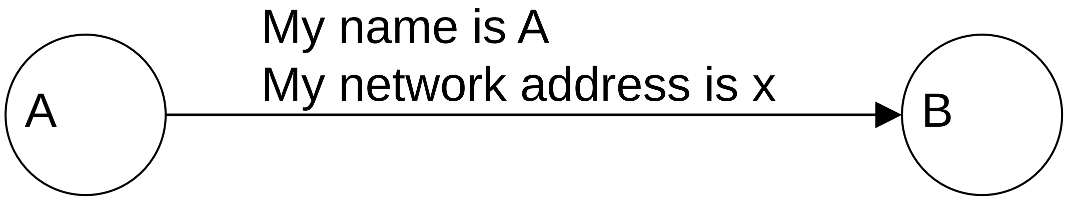
Link State Routing (LSR)

Each router must do the following:

1. Discover its neighbours, learn their network address.
2. Measure the delay to each of its neighbours.
3. Construct a packet telling all it has just learned.
4. Distribute this packet to all other routers.
5. Compute the best route (minimum delay) to every other router using a static algorithm.

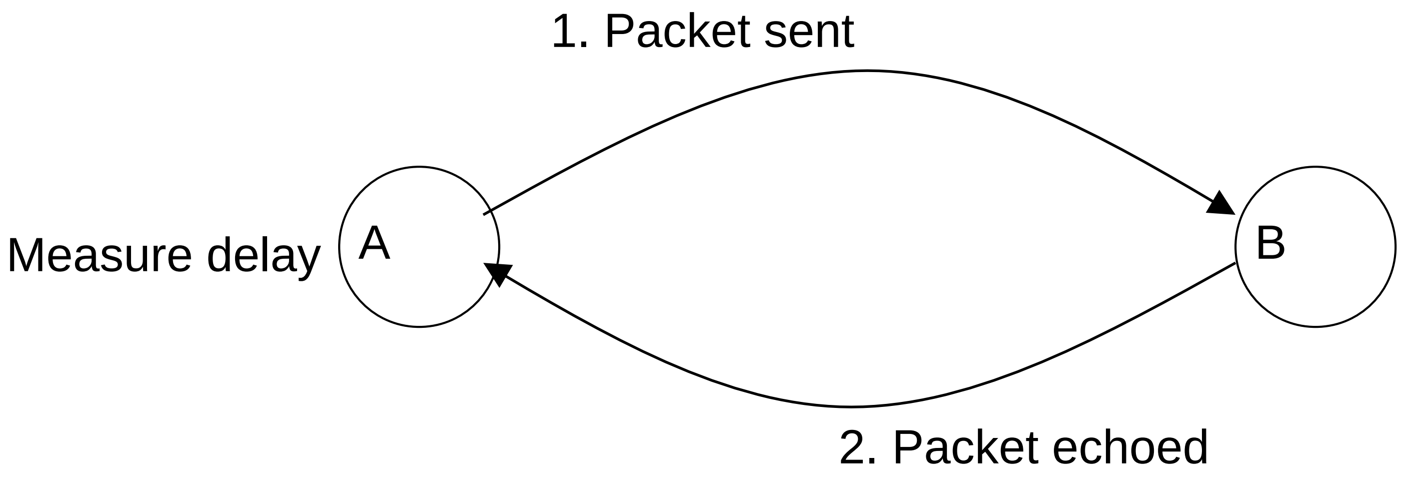
- Can be run periodically but is **rather static** (the algorithm must run to completion before new routes are known).

LSR – 1. Discover neighbours



Every router says hello to its neighbours and tells them something about itself.

LSR – 2. Measure line delay

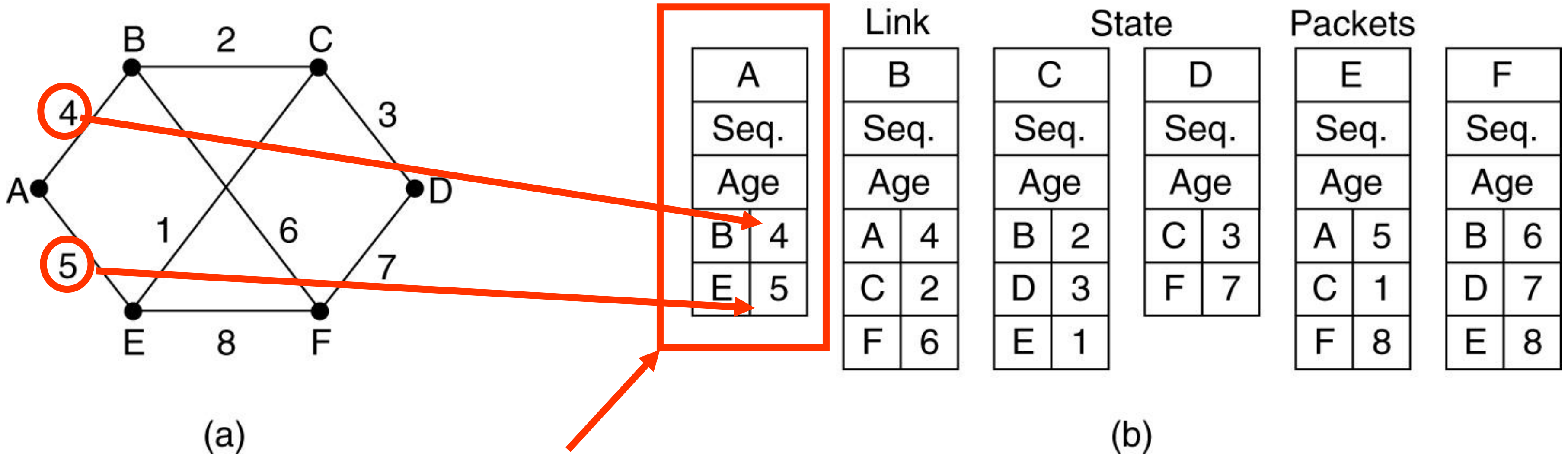


Every router sends an echo packet to its neighbours.

Received echo packets are returned from where they came.

Measure the delay on the echo.

LSR – 3. Build link state packets



A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

The state packet of A is
Measured delay between
A and each of its neighbours

LSR – 4 Distribute packets

Uses flooding to distribute link state packets but needs algorithm to avoid distributing duplicates.

LSR – 5. Compute best route

- Use a static algorithm, such as shortest path (Dijkstra) to calculate best route to all routers based on information in link state packets.

Shortest Path Routing (Dijkstra’s Algorithm)

- The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line.
- Each arc is labelled with cost/distance.
- To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.
- There are several metrics that can be used to measure the path *length*: number of hops, geographical distance in kilometres, mean queuing delay, transmission delay for some standard test packet (eg LSR), or a combination of any of them.
- Algorithm can be run periodically but is rather static

Shortest Path Routing (2)

- Eventually, each node will be labelled with its distance from the source node along the best known path.
- Initially, no paths are known, so all nodes are labelled with *infinity*.
- As the algorithm progresses and paths are found, the labels may change, reflecting the best paths.
- The labels are either <u>tentative</u> or <u>permanent</u>. 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.

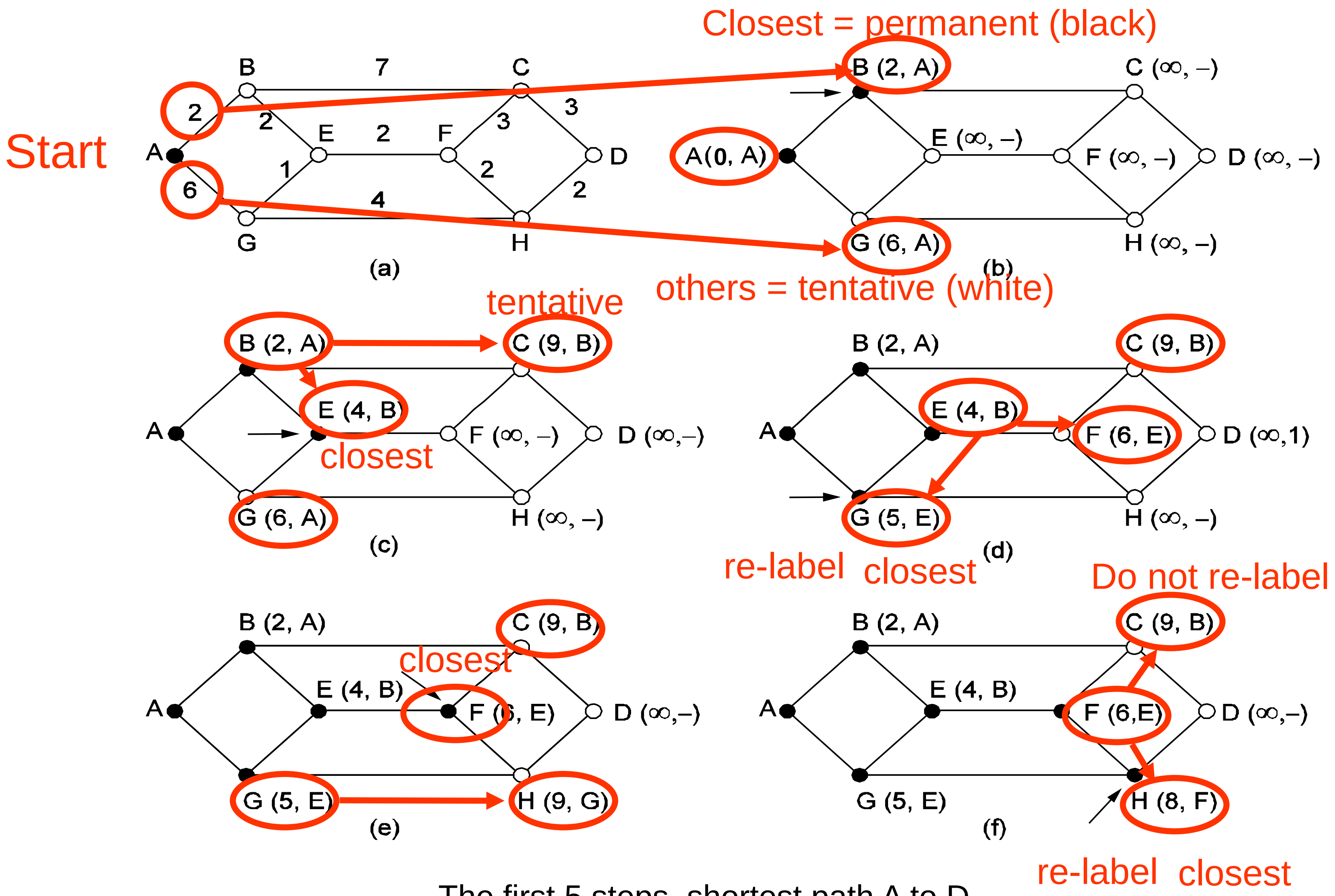
Shortest Path Routing (3)

Initially, no paths are known so label all routers with infinity and mark as tentative. Router A becomes the start router for the algorithm – mark as permanent.

```
start of loop
    for each of the start router’s tentative neighbours:
        Calculate that neighbour’s distance from A via the start router by adding
        the distance between the neighbour and the start router to the distance
        contained in the label on the start router.
        If the new calculated distance from A is less than that on the neighbour’s
        existing label, re-label the neighbour with the new distance from A and
        the name of the start router.
    Examine all tentative labels, choose the one labelled with the minimum
    distance to A and make that router the new start.
Repeat loop until all routers are labelled permanent.
```

Trace the shortest path by following the labels from the destination to the source.

Shortest Path Routing (Example 1)

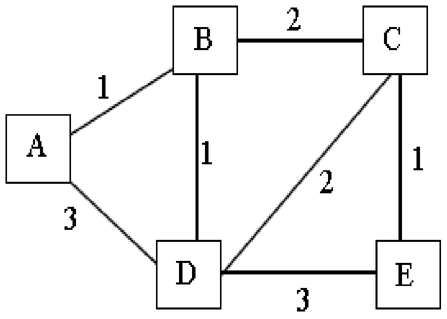


The first 5 steps, shortest path A to D

Shortest Path Routing (Example 2) (Building the Routing Table)

Routing algorithms example

Consider the following network topology. The boxes indicate nodes and the integers next to the arcs are the link cost. All links are bi-directional.



1. Use Dijkstra's algorithm to compute the least cost routes from node A to all other nodes. Show your working in a set of tables with columns "Step" (number of times in the loop), N, and "D(X), p(X)" for all X not equal to A.

Dijkstra's algorithm is a centralised algorithm, so all the topology is known.

The routing table looks like this:

Step	N	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)
0	{A}	**1,A**	∞	**3,A**	∞
1					

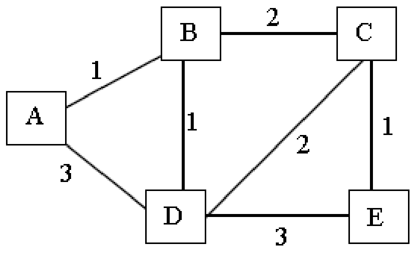
Step is the iteration number of the algorithm.
c(i,j): link cost from node i to j. cost infinite if not direct neighbours
D(v): current value of cost of path from source to destination V
p(v): predecessor node along path from source to v, that is next v
N: set of nodes whose least cost path definitively known

Infinite entries mean no route has yet been found. Bold entries are new.

Shortest Path Routing (Example 2) (Building the Routing Table)

Routing algorithms example

Consider the following network topology. The boxes indicate nodes and the integers next to the arcs are the link cost. All links are bi-directional.



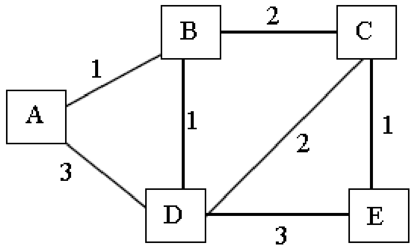
So here's the answer….

Step	N	D(B), p (B)	D(C), p (C)	D(D), p (D)	D(E), p (E)	Comments
0	{A}	1,A	∞	3,A	∞	Start at A
1						
2						
3						
4						

Shortest Path Routing (Example 2) (Building the Routing Table)

Routing algorithms example

Consider the following network topology. The boxes indicate nodes and the integers next to the arcs are the link cost. All links are bi-directional.



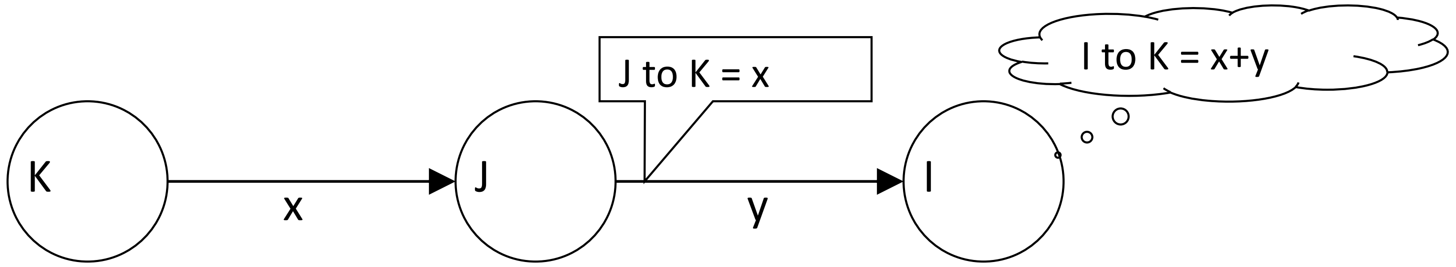
So here's the answer….

Step	N	D(B), p (B)	D(C), p (C)	D(D), p (D)	D(E), p (E)	Comments
0	{A}	**1,A**	∞	**3,A**	∞	Start at A
1	{A,B}	1,A	**3,B**	**2,B**	∞	AB is least cost, so add B to N. D(B) and p(B) will not change hereafter. Still can't get to E from any node in N.
2	{A,B,D}		3,B	2,B	**5,D**	ABD is least cost so add D to N. Can now see E (via ABDE). Cost of C doesn't change.
3	{A,B,D,C}		3,B		**4,C**	ABC is least cost so add C to N. Cost of ABCE is less than ABDE, so change.
4	{A,B,D,C,E}				4,C	Add E to N. All nodes now in N so end loop.

Distance Vector Routing

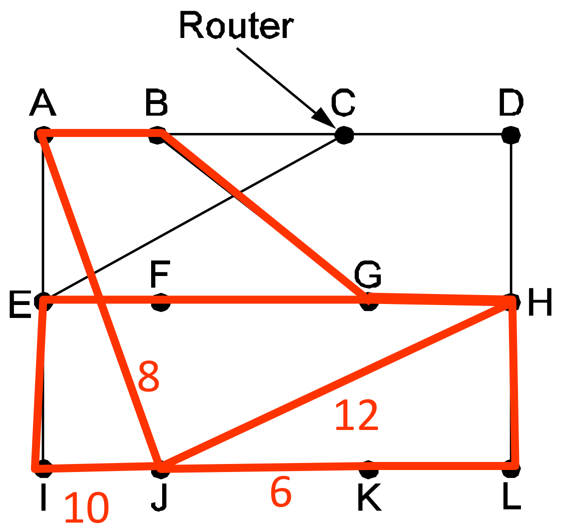
Distance Vector Routing

- Each router maintains a table giving the best (currently known) distance to each destination, and which line must be used to get there.
- These tables are updated periodically by exchanging information with the neighbouring routers so routes adapt dynamically to network changes.
  - If a node I receives a table from its neighbour J saying the ***delay to node K (from J) is x***, and if the ***delay from I to J is y***, then the ***delay from I to K via J is x+y***.



- A major concern is the Count-to-Infinity problem.

Distance Vector Routing (2)



Network

To estimate J-G delay calculate
J-A-G (18 + 8 = 26)
J-I-G (31 + 10 = 41)
J-H-G (6 + 12 = 18)
J-K-G (31 + 6 = 37)
And chose smallest (via H=18)

Vectors received from J's four neighbours

New estimated Delay from J

To	A	I	H	K	New estimated Delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K

	JA delay is	JI delay is	JH delay is	JK delay is
	8	10	12	6
	26	41	18	37

New routing table for J

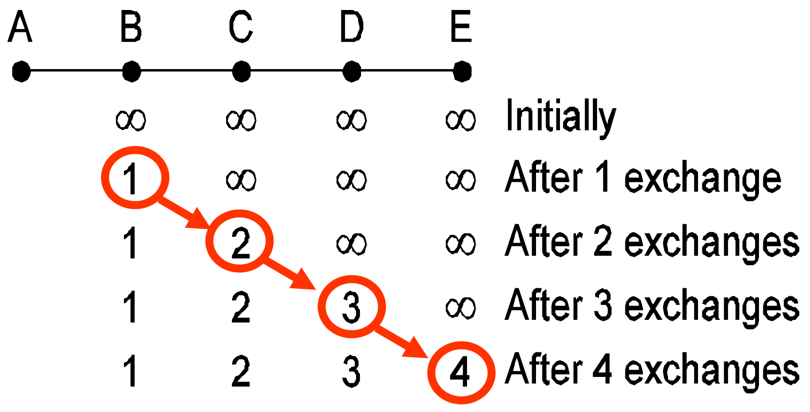
Distance Vector Routing (3)

The Count-to-Infinity (C2I) problem:

- Distance Vector Routing algorithm reacts rapidly to good news (new better route) but slowly to bad news (best route goes down).
- When a router becomes unreachable (or dies), the distance from all routers to it increases slowly to *infinity* after each routing table exchange cycle.
- It will take lots of exchange cycles before a router is declared unreachable. This is also called convergence time.
- One solution is to set *infinity* to longest path + 1.

Distance Vector Routing (4)

The Count-to-Infinity (C2I) problem:

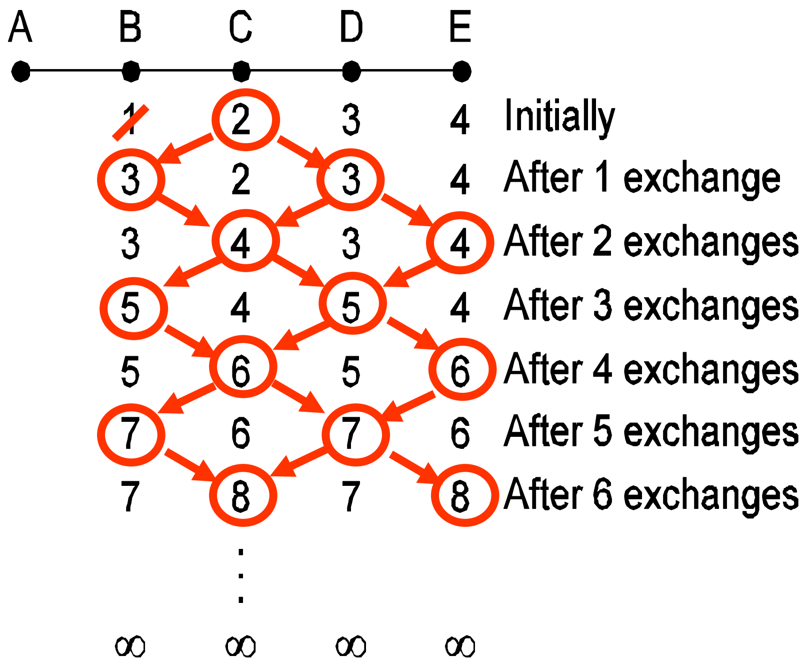


A	B	C	D	E	
	∞	∞	∞	∞	Initially
	1	∞	∞	∞	After 1 exchange
	1	2	∞	∞	After 2 exchanges
	1	2	3	∞	After 3 exchanges
	1	2	3	4	After 4 exchanges

Good news travels fast

Initially A off and others on

A is then switched on



A	B	C	D	E	
	~~1~~	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	∞	∞	∞	∞	

Bad news travels slowly

Initially all on

A is then switched off

Least hop routes

- With Dijkstra set all costs to 1 to find the least hop.
- With Distance vector the least hop route is the first to be found for each destination.

Discussion Points

- Strengths of each approach:
  - Dijkstra is easier when the network is small and holding all the knowledge is easy, but becomes computationally costly quickly.
  - It also means somewhere in the network has to know what the status is of all the nodes and links in the networks in order to react to changes. If changes occur then the Dijkstra may need totally recalculating, whereas distance vector can adapt.

Discussion Points

- Strengths of each approach:
  - Distance vector is distributed; local costs can be calculated very quickly and more distant costs take longer to find. This means it can react to local changes quickly.
  - If costs are fairly similar then the number of iterations needed to find the least cost path will be close to the shortest path length for the most distant pair of nodes (shortest path ≈ least cost path).
  - However, if the variation in costs is high, then the number of iterations will tend to the maximum (non-repeating) path length between any 2 nodes