ROUTING PROTOCOLS

Routing vs. Forwarding/Switching

- routing – process by which **routing table** is built
- forwarding/switching –
  - take a packet and look at its destination address
  - consult a routing table
  - send the packet in the direction determined by that table

Network as a graph

- Node – router/switch/host
- Link – network link with associated *cost*

Problem – find the **lowest cost path** between any two nodes

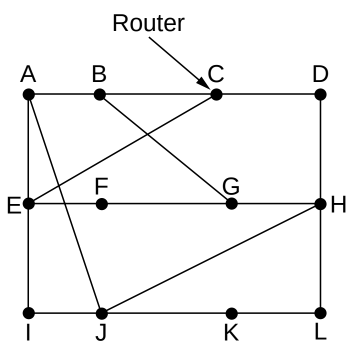
Routing Algorithms/Protocols

- Non-adaptive (static) – do not base routing decisions on measurements/estimates of the current traffic or topology
- Adaptive (dynamic) – change their routing decisions to reflect changes in topology and traffic (distance vector and link state)

⇒ **Routing protocols** that provide distributed & dynamic ways to solve the problem of finding the lowest-cost path in the presence of link/node failures/changing

Distance Vector (DV) Routing

- Used in ARPAnet until 1979
- **Distance Vector** ≡ each node constructs an one-dimensional array (**vector**) containing the costs (**distance**) to all other nodes and distributes that vector to its **immediate neighbors**
- Assumption – each node knows the link cost to each of its **directly connected neighbors** (A–8,I–10,H–12,K–6)
- *Old routing table is not used in calculation*



(a)

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 8	JI delay is 10	JH delay is 12	JK delay is 6	New routing table for J	

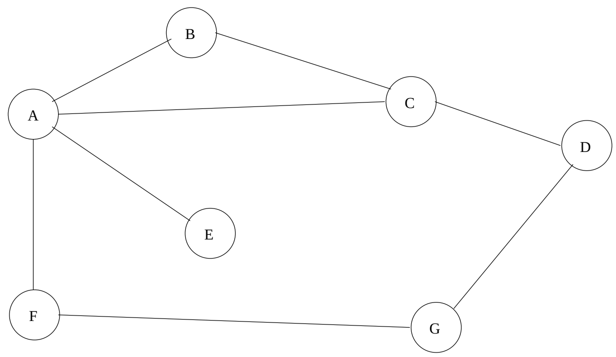
Vectors received from J's four neighbors

(b)

When to send updates

- periodic – once every so often
- triggered updates – whenever routing table changes

Link/Node Failures



- Example 1 – **stable**
  - F detects that link to G has failed
  - F sets distance to G to ∞ and sends update to A
  - A sets distance to G to ∞ since it uses F to reach G
  - A receives periodic update from C with 2-hop path to G
  - A sets distance to G to 3 and sends update to F
  - F decides it can reach G in 4 hops via A
- Example 2 – **unstable**
  - Link from A to E fails
  - A advertises ‘distance to E $\equiv \infty$’
  - B and C advertise ‘distance to E $\equiv$ 2’
  - B decides it can reach E in 3 hops; advertises this to A
  - A decides it can reach E in 4 hops; advertises this to C
  - C decides that it can reach E in 5 hops
  - $\cdots\cdots\cdots$

The Count-to-∞ Problem

react rapidly to good news, but slowly to bad news

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

(a)

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
		⋮			
	∞	∞	∞	∞	

(b)

- Partial solution - **split horizon (with poison reverse)** idea – a node does not send those routes it learned from each neighbor back to that neighbor (actually, it is reported as ∞)
  - A goes down
  - B discovers A is down, C reports '∞ to A' $\rightarrow$ B set ∞ to A
  - C hears A is down from both neighbors $\rightarrow$ C set ∞ to A

Link State (LS) Routing

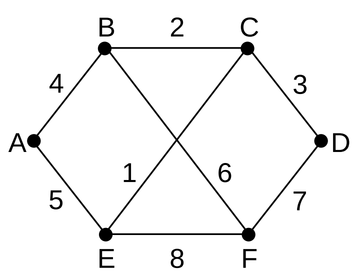
Assumption – each node knows the cost of the link to each of its **directly connected neighbors** (same as DV)

Idea – Send to **all** nodes (not just neighbors) cost information about directly connected links

Each node performs the following 5 steps periodically

1. Discover its neighbors and learn their network addresses via HELLO packet
2. Measure the cost (RTT) to each of its neighbors via ECHO packet
3. Construct LSP (Link State Packet) telling all it learned
4. Distributed LSP to all other R via **reliable flooding**
5. Compute *locally* the **shortest paths** to every other R (complete topology map) via Dijkstra's shortest-path algorithm

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



(a)

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

(b)

- Reliable Flooding
  - store most recent LSP from each node
  - forward LSP to all nodes except the one that sent it
  - generate new LSP periodically – increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP - discard when TTL=0
- Examples – **IS-IS** and **OSPF**

Distance Vector vs. Link State

Strategy

- In DV, each node talks only to its directly connected neighbors, but it tells them everything it has learned (its entire routing table) – distance to all nodes
- In LS, each node talks to all other nodes, but it tells them only what it knows for sure (only the state of its directly connected links)

Metrics – latency, bandwidth, traffic load

- Metric used in DV
  - measured number of packets queued on each link
  - did not take latency or bandwidth into consideration
- Metric used in LS
  - stamp each incoming packet with its arrival time (`AT`)
  - record departure time (`DT`)
  - when link-level ACK arrives, compute

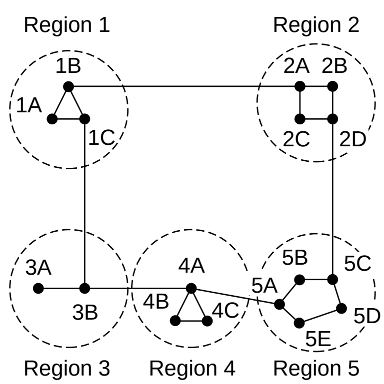
    `Delay = (DT - AT) + Transmit + Latency`
  - if timeout, reset `DT` to departure time for retransmission
  - link cost = average delay over some time period

Hierarchical Routing

Routing table grows **exponentially** – more memory, CPU time, and bandwidth ⇒ cannot afford to have an entry for every router

Routers are divided into **regions**, with each R

- know all the routing details within its own region
- know nothing about the internal structure of other regions



(a)

Full table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Hierarchical table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Router ⊂ Region ⊂ Cluster ⊂ Zone ⊂ Group $\cdots\cdots$
$\Longrightarrow$ **flat** address vs. **hierarchical** address

Penalty – increased path length

Question – how many **levels** should the hierarchy have?