# Chapter 5

## The Network Layer

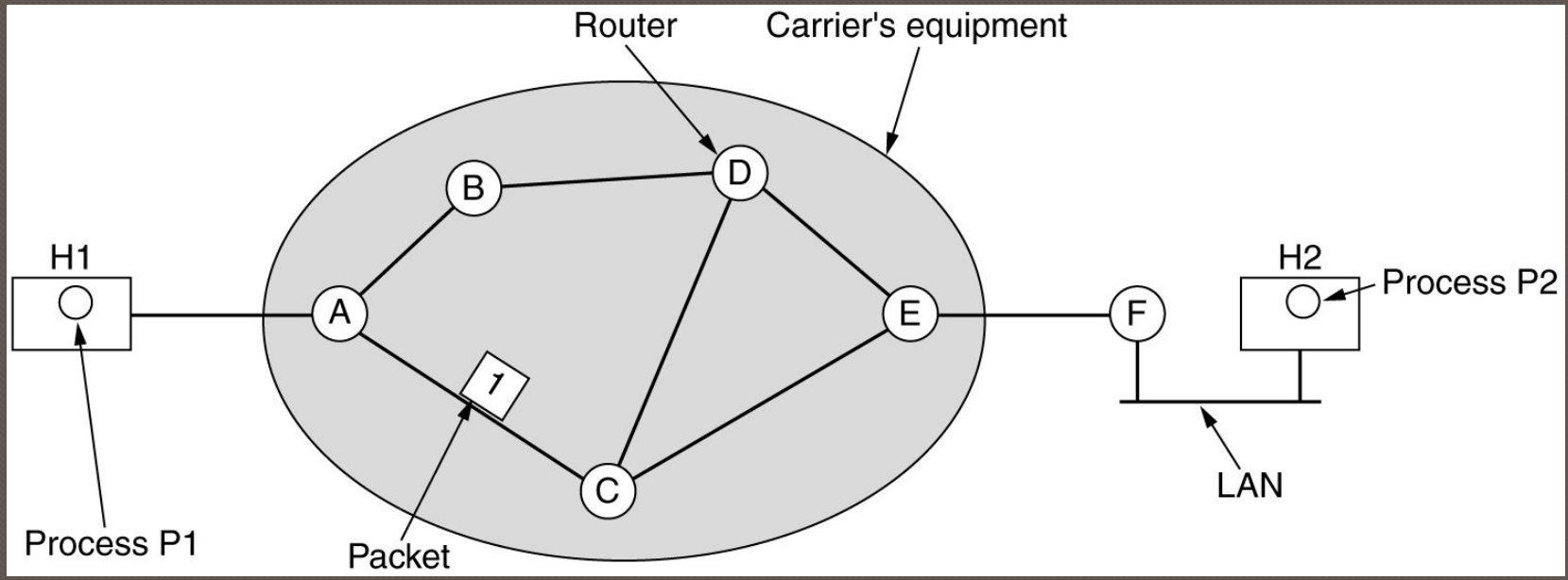
# Network Layer Design Isues

- Store-and-Forward Packet Switching
- Services Provided to the Transport Layer
- Implementation of Connectionless Service
- Implementation of Connection-Oriented Service
- Comparison of Virtual-Circuit and Datagram Subnets

# Store-and-Forward Packet Switching

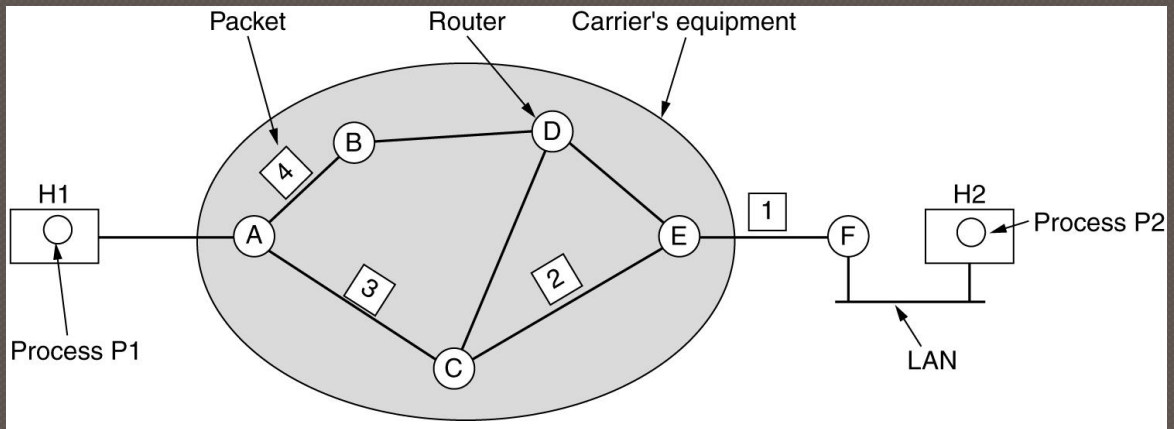
The environment of the network layer protocols.

fig 5-1



# Implementation of Connectionless Service

Routing within a diagram subnet.




A's table

| Dest. | initially | later |
|---|---|---|
| A | – | – |
| B | B | B |
| C | C | C |
| D | B | B |
| E | C | B |
| F | C | B |

C's table

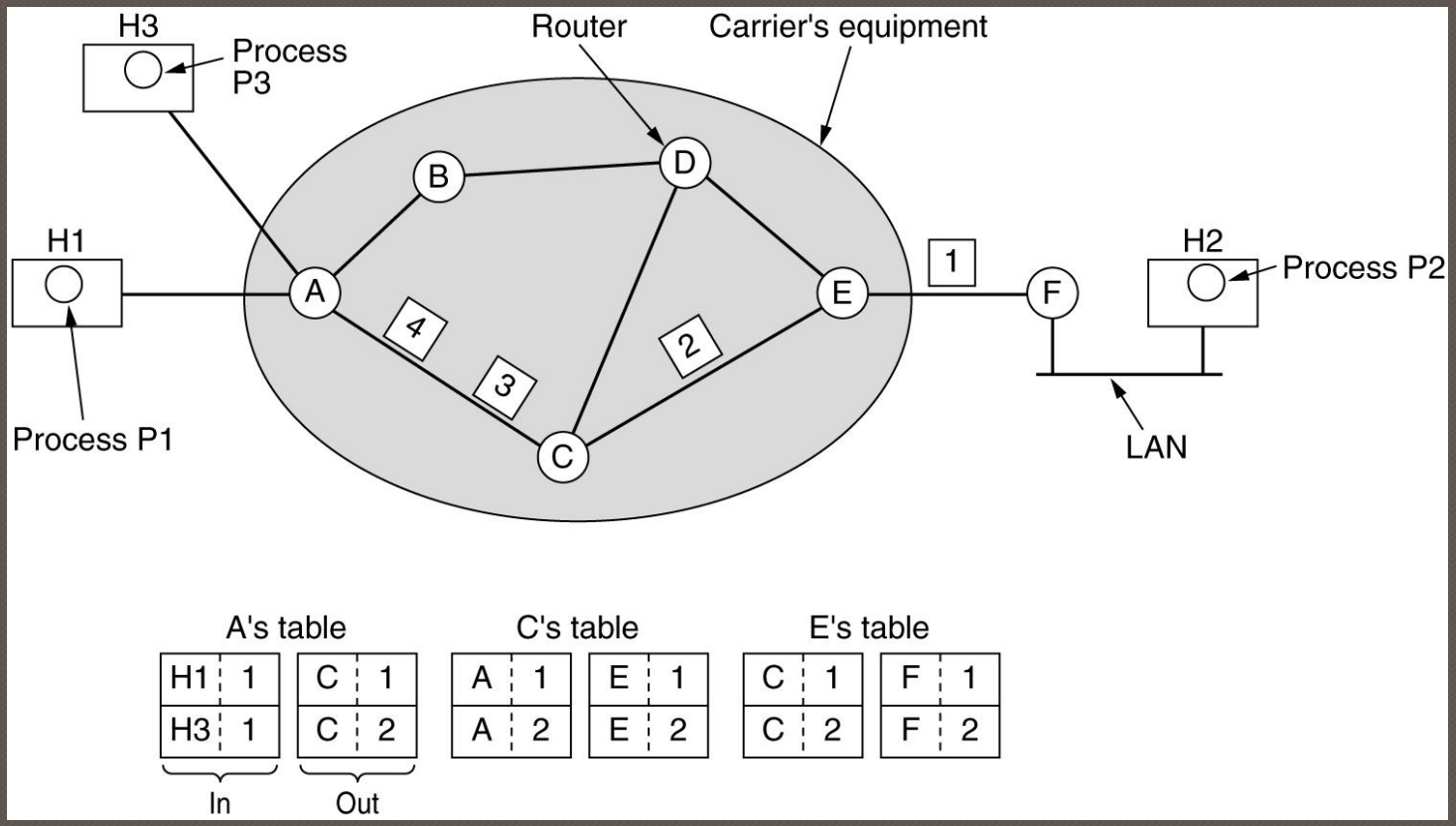
| Dest. | Line |
|---|---|
| A | A |
| B | A |
| C | – |
| D | D |
| E | E |
| F | E |

E's table

| Dest. | Line |
|---|---|
| A | C |
| B | D |
| C | C |
| D | D |
| E | – |
| F | F |

# Implementation of Connection-Oriented Service

Routing within a virtual-circuit subnet.



# Comparison of Virtual-Circuit and Datagram Subnets

| Issue | Datagram subnet | Virtual-circuit subnet |
|---|---|---|
| Circuit setup | Not needed | Required |
| Addressing | Each packet contains the full source and destination address | Each packet contains a short VC number |
| State information | Routers do not hold state information about connections | Each VC requires router table space per connection |
| Routing | Each packet is routed independently | Route chosen when VC is set up; all packets follow it |
| Effect of router failures | None, except for packets lost during the crash | All VCs that passed through the failed router are terminated |
| Quality of service | Difficult | Easy if enough resources can be allocated in advance for each VC |
| Congestion control | Difficult | Easy if enough resources can be allocated in advance for each VC |

# Routing Algorithms

- A Routing Algorithm is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on.
- A session routing occurs when a route remains in force for the entire user session

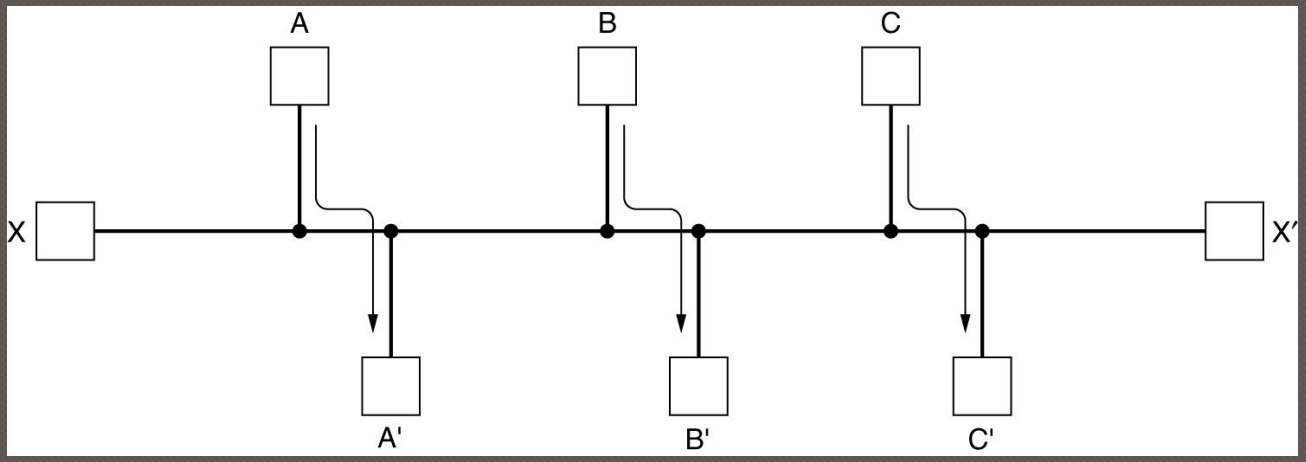
1. Forwarding– decides which outgoing line to use per packet according to the routing tables.
2. Update the routing tables.

# Routing Algorithms

- The Optimality Principle
- Shortest Path Routing
- Flooding
- Distance Vector Routing
- Link State Routing
- Hierarchical Routing
- Broadcast Routing
- Multicast Routing
- Routing for Mobile Hosts
- Routing in Ad Hoc Networks

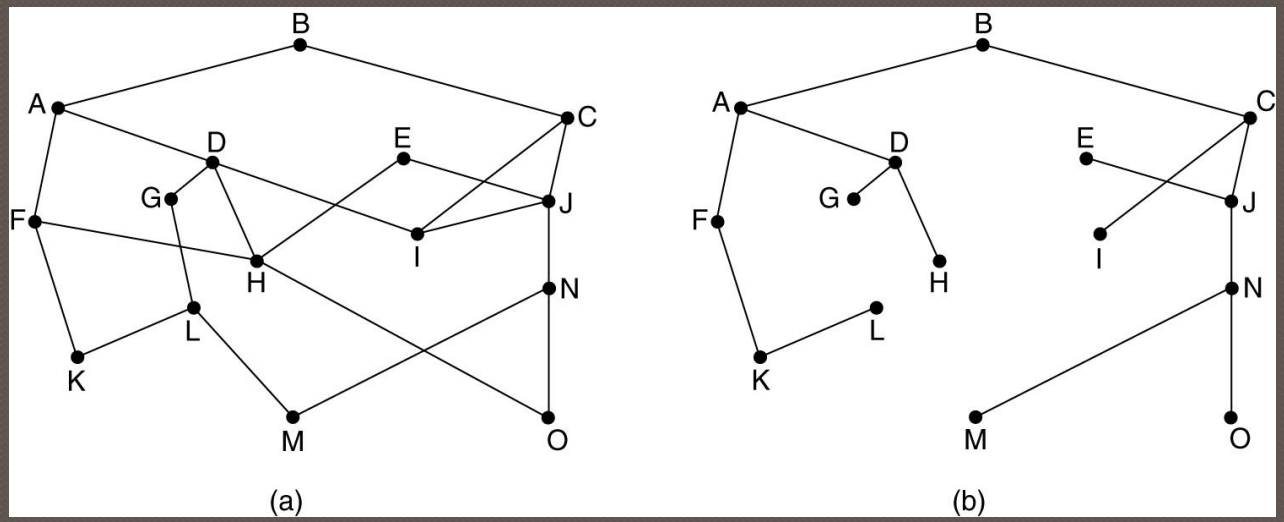
# Routing Algorithms (2)

- Conflict between fairness and optimality.
- Nonadaptative algorithms make routing decisions base on measurements of the current traffic and topology.
- Adaptative algorithms change routing decisions reflecting the current traffic and topology changes.



# The Optimality Principle

***Optimality Principle***: If router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route




(a) A subnet. (b) A sink tree for router B.

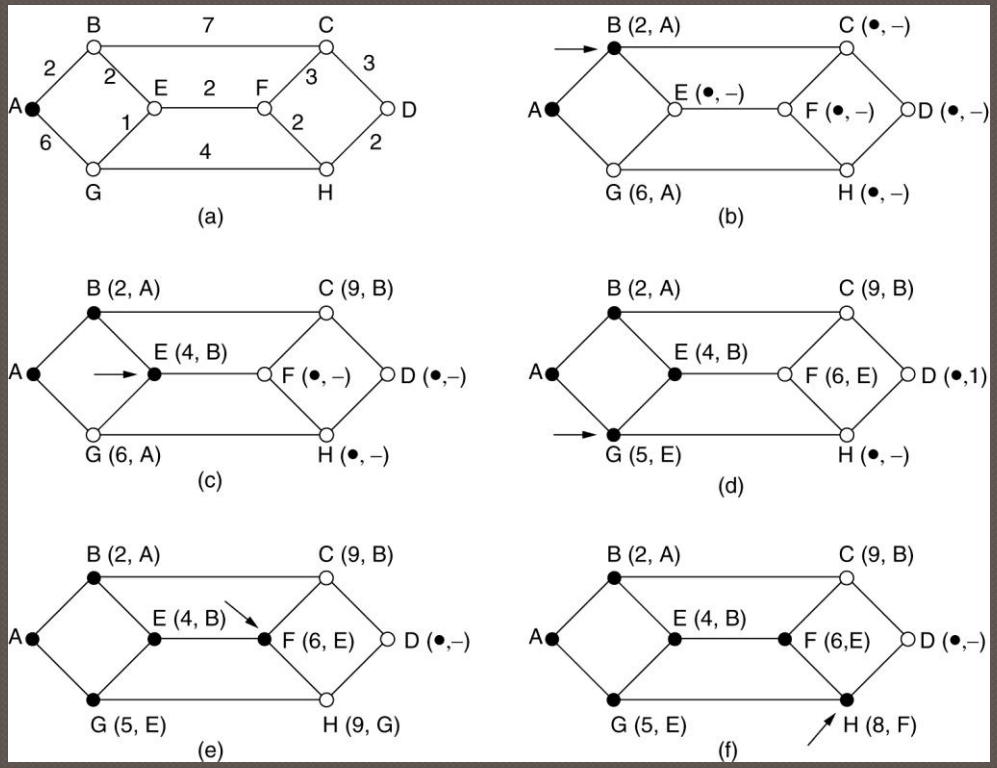
# Shortest Path Routing (SPR)

- How to find the SP from A to J:

1. Label each nodes with the label (∞, -)
2. Mark the initial node
3. Relabel each adjacent nodes to the working node with the distance from the working node and the node from which the probe was made
4. Select the node in the graph with the smallest label and mark it. This is the new working node.
5. Repeat step 3-5 until you reach J.

# Shortest Path Routing

The first 5 steps used in computing the shortest path from A to D.
The arrows indicate the working node.



# Shortest Path Routing

```
#define MAX_NODES 1024                  /* maximum number of nodes */
#define INFINITY 1000000000             /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];/* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                        /* the path being worked on */
      int predecessor;                  /* previous node */
      int length;                       /* length from source to this node */
      enum {permanent, tentative} label; /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *p;

  for (p = &state[0]; p < &state[n]; p++) {  /* initialize state */
      p->predecessor = –1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                                /* k is the initial working node */
```

Dijkstra's algorithm to compute the shortest path through a graph.

# Shortest Path Routing

```
do {                                        /* Is there a better path from k? */
     for (i = 0; i < n; i++)                /* this graph has n nodes */
          if (dist[k][i] != 0 && state[i].label == tentative) {
               if (state[k].length + dist[k][i] < state[i].length) {
                    state[i].predecessor = k;
                    state[i].length = state[k].length + dist[k][i];
               }
          }

     /* Find the tentatively labeled node with the smallest label. */
     k = 0; min = INFINITY;
     for (i = 0; i < n; i++)
          if (state[i].label == tentative && state[i].length < min) {
               min = state[i].length;
               k = i;
          }
     state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0;  k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Dijkstra's algorithm to compute the shortest path through a graph.

# Flooding

- Flooding: every incoming packet is sent out on every outgoing line except the one it arrive on.
  - Generates large number of duplicate packets
- Selective Flooding: every incoming packet is sent out only to those lines that are going approximately in the right direction
  - No need to send a packet east a packet going west

# Distance Vector Routing

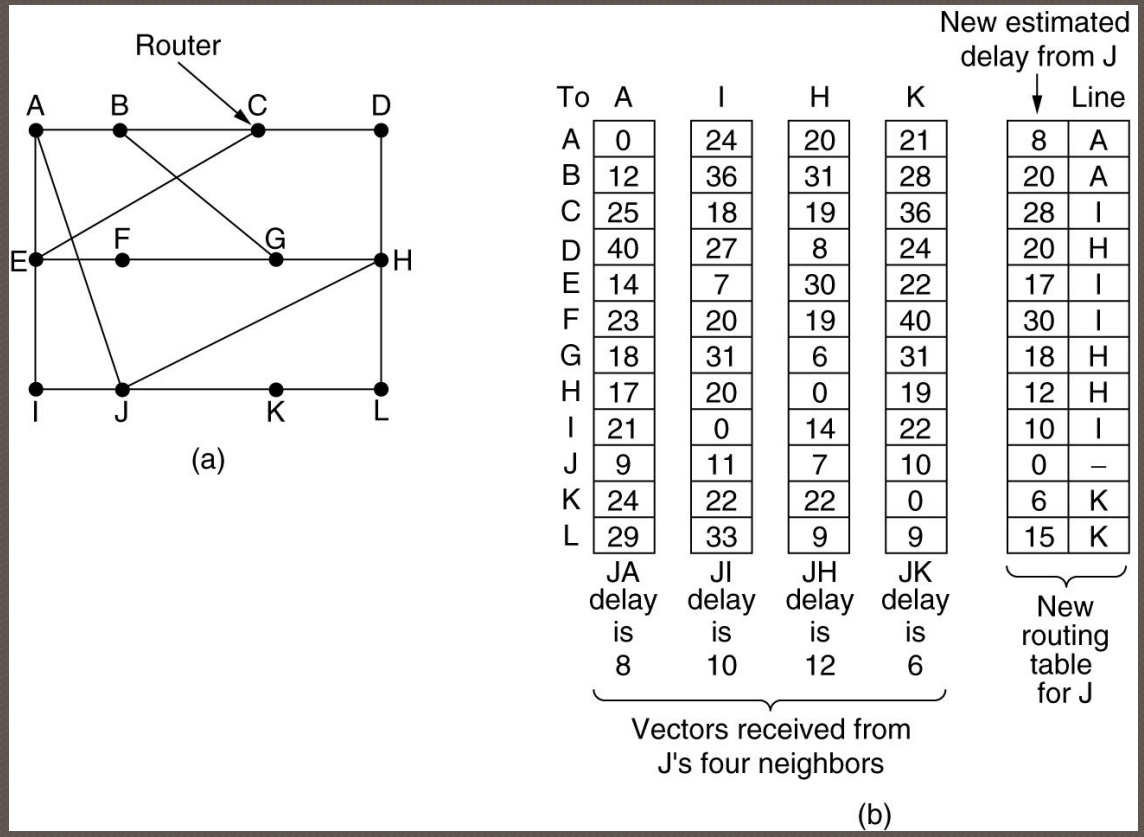
- Also called the Bellman-Ford or Ford-Fulkerson algorithms
- Used in ARPANET

- Idea: - Maintain a routing table (i.e. a vector) for each router containing
  - the best outgoing line for that destination.
  - the distance or time to that destination

- Steps:

1. Every T msec each router sends to each neighbor a list of estimate delays to each destination
2. Each router updates its own table accordingly.

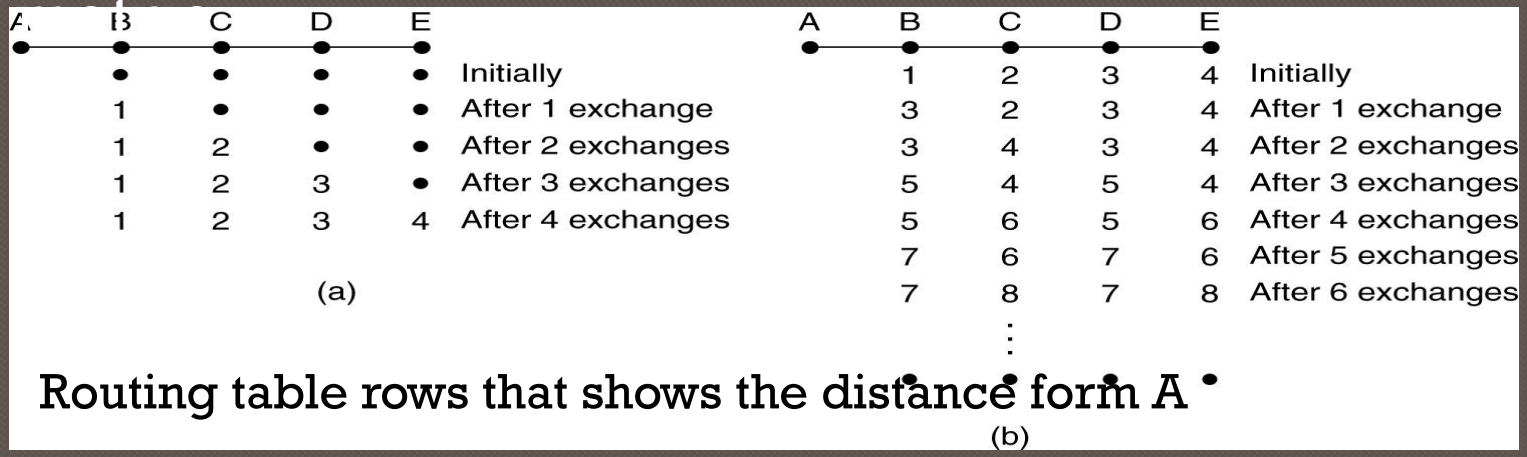
# Distance Vector Routing




(a) A subnet. (b) Input from J's neighbors A, I, H, K, and the new routing table for J.

# The count-to-infinity problem in Distance Vector Routing

- React fast to good news
  - If a neighbor reports a short delay to X, the router switches to the new low value
- News spread slowly
  - If A crashes, the news is spread at the rate of one hop per exchange – the routers work their way up to infinity
- It is wise to set "infinity" to the longest path plus 1 if we count the hops.
- An educated guess upper bound is used if time is the




Routing table rows that shows the distance form A

# Next Time

- More routing algorithms….