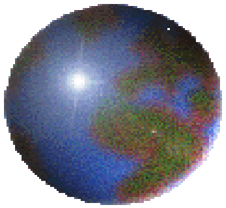


CEN445
Network Protocols & Algorithms

Chapter 5

Network Layer

Prepared by

Dr. Mohammed Amer Arafah

Summer 2008

Network Layer

- 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.

Introduction

- **Network layer** is concerned with *getting packets from the source all the way to the destination*.
- Getting to destination may require making many **hops** at the intermediate nodes.
- To achieve its goal, the network layer must know about the topology of the communication subnet and *choose appropriate paths through it*.
- It must also *choose routes to avoid overloading* some of the communication lines and nodes while leaving the others idle.
- When the source and destination are in different networks, it is up the network layer to deal with these differences (*Internetworking*).

Network Layer Design Issues

1. **Services Provided to the Transport Layer:**

- The network layer services have been designed with the following goals:
  - The service should be **independent** of the subnet technology.
  - The transport layer should be **shielded** from the number, type, and topology of the subnets present.
  - The network addresses made available to the transport layer should use a **uniform numbering plan**, even across LANs or WANs.

- To achieve these goals, the network layer can provide either **connection-oriented service** or **connectionless service**.

Network Layer Design Issues

- The first camp (represented by the **Internet community**) supports connectionless service. It argues that the subnet's job is moving bits around and nothing else. In their view, the subnet is unreliable. Therefore, the host should do **error controls** (error detection and correction) and **flow control** themselves.

- The other camp (represented by the **telephone companies**) argues that the subnet should provide a reasonably reliable, connection-oriented service. In their view, connections should have the following properties:
  - The network layer process on the sending side must set up the connection to its peer on the receiving side.
  - When the connection is set up, the two processes negotiate about the parameters of the service to be provided.
  - Communication in both directions, and packets are delivered in sequence.
  - Flow control is provided.

Conclusion

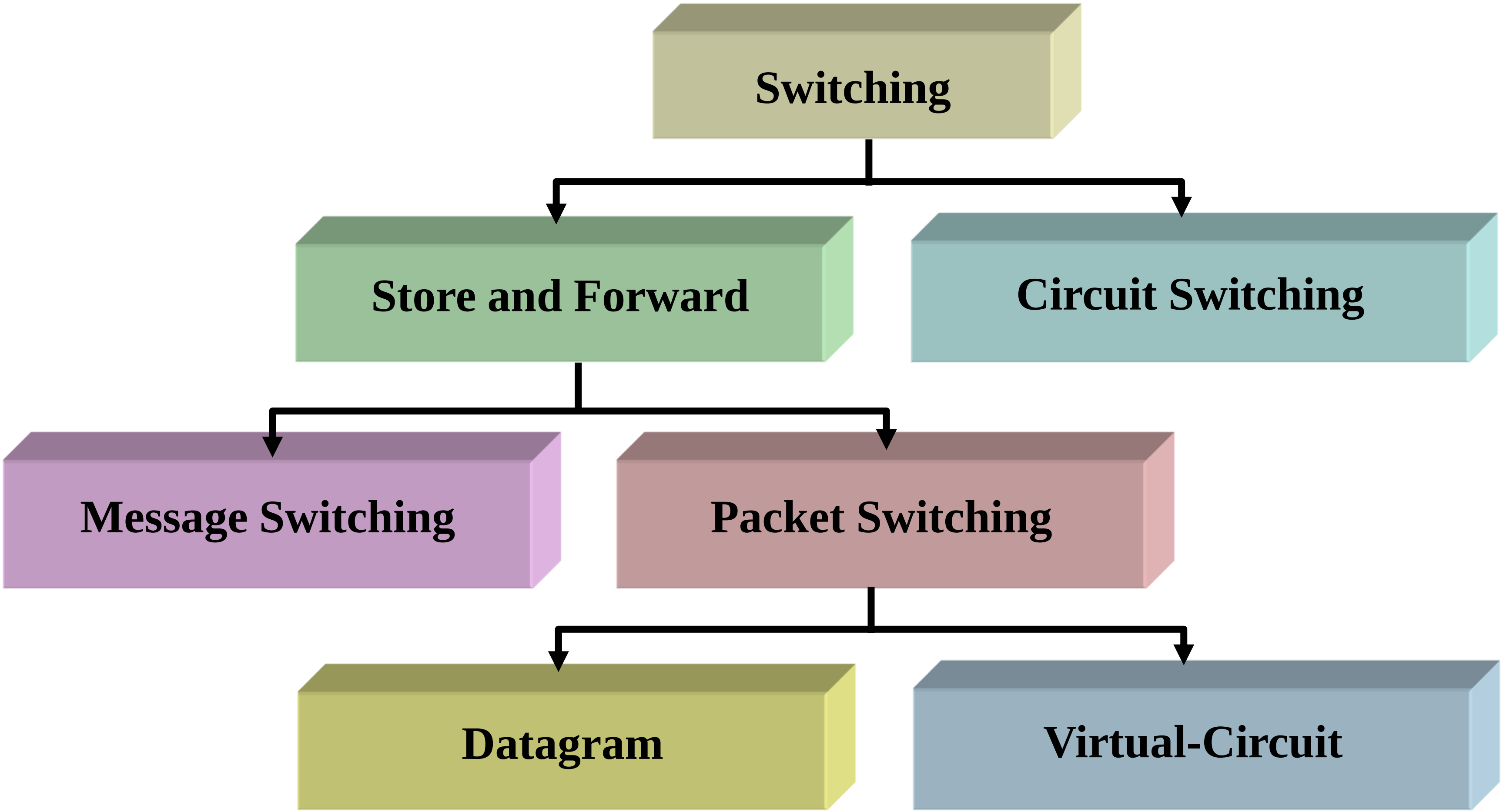
- The argument between connection-oriented and connectionless service really has to do with where to put the **complexity**.
- In the connection-oriented service, it is in the network layer (subnet); in the connectionless service, it is in the transport layer (hosts).
- Supporters of connectionless service say that computing power has become cheap, so that there is no reason to put the complexity in the hosts. Furthermore, it is easier to upgrade the hosts rather than the subnet. Finally, some applications, such as digitized voice and real time data require speedy delivery as much more important than accurate delivery.
- Supporters of connection-oriented service say that most users are not interested in running complex transport layer protocols in their machine. What they want is reliable, trouble free service, and this service can be best provided with network layer connections.

Network Layer Design Issues

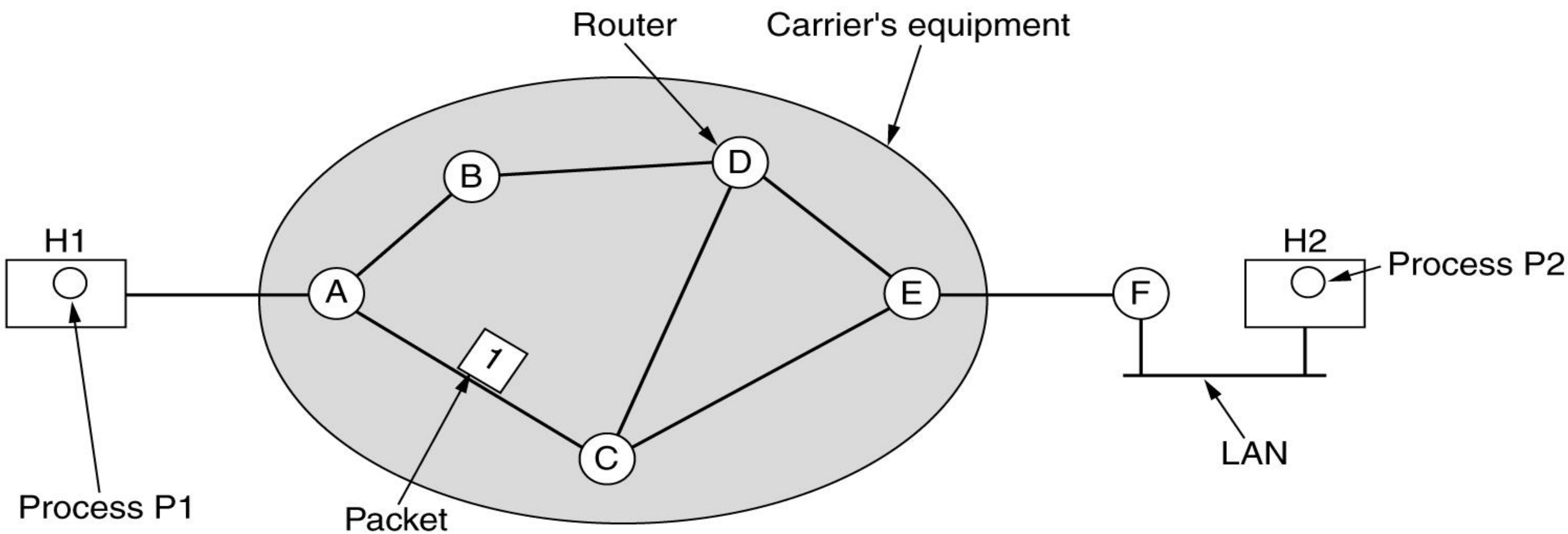
2. Internal Organization of the Network Layer:

- There are basically two different philosophies for organizing the subnet, one using connections (**virtual circuits**) and the other using connectionless (**datagrams**).

Switching

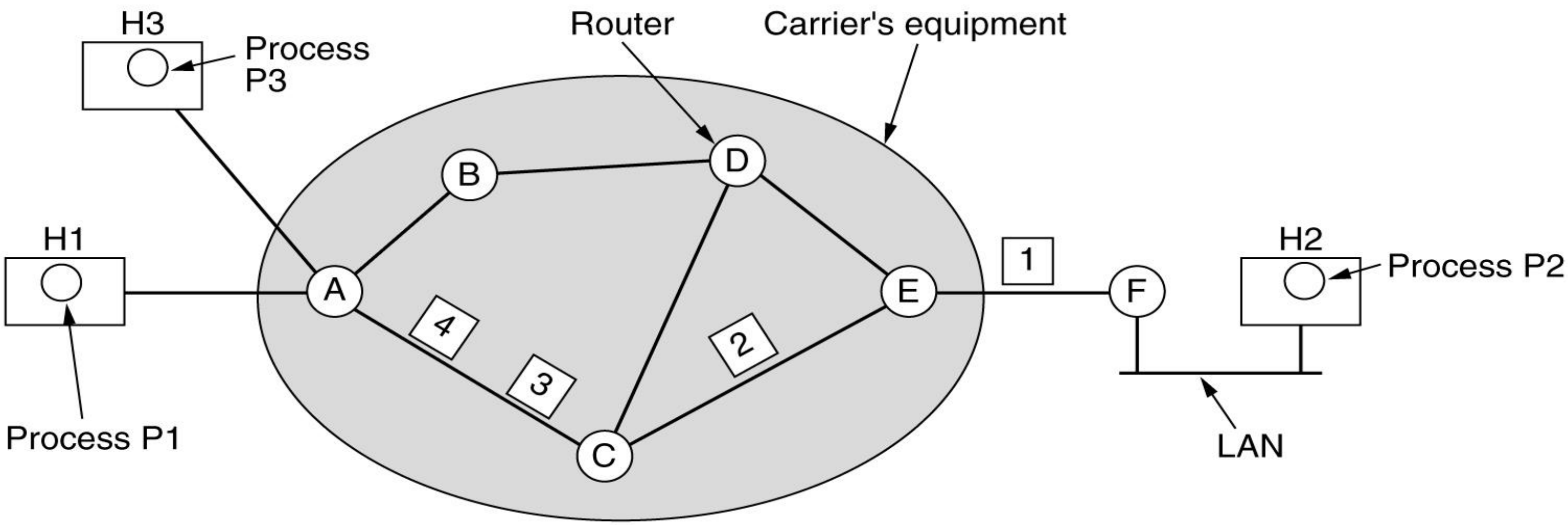

Switching
Store and Forward
Circuit Switching
Message Switching
Packet Switching
Datagram
Virtual-Circuit

Store-and-Forward Packet Switching



The environment of the network layer protocols

Implementation of Connection-Oriented Service



A's table

In		Out	
H1	1	C	1
H3	1	C	2

C's table

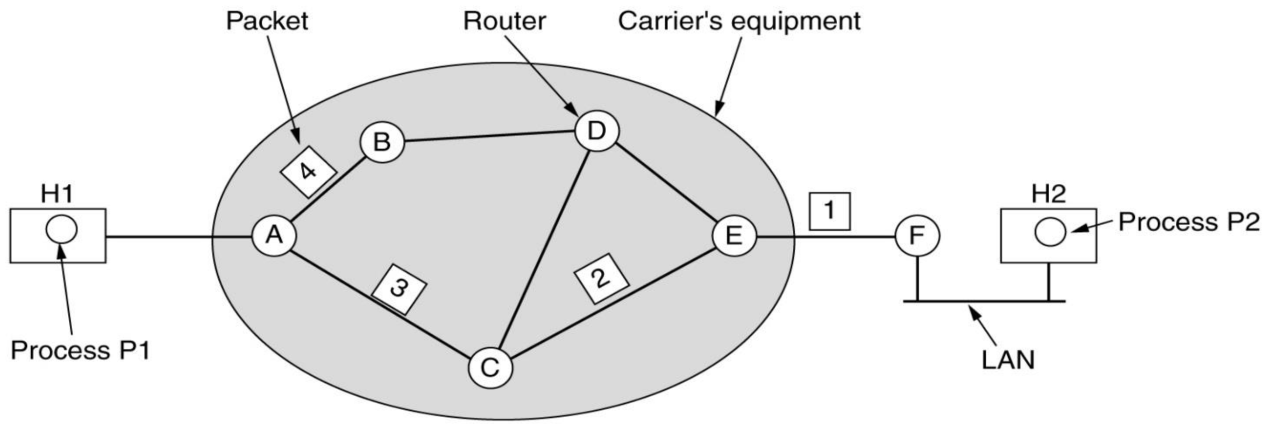
A	1	E	1
A	2	E	2

E's table

C	1	F	1
C	2	F	2

Routing within a virtual-circuit subnet

Implementation of Connectionless Service



Dest.	A's table initially (Line)	A's table later (Line)	C's table (Line)	E's table (Line)
A	–	–	A	C
B	B	B	A	D
C	C	C	–	C
D	B	B	D	D
E	C	B	E	–
F	C	B	E	F

Routing within a diagram subnet

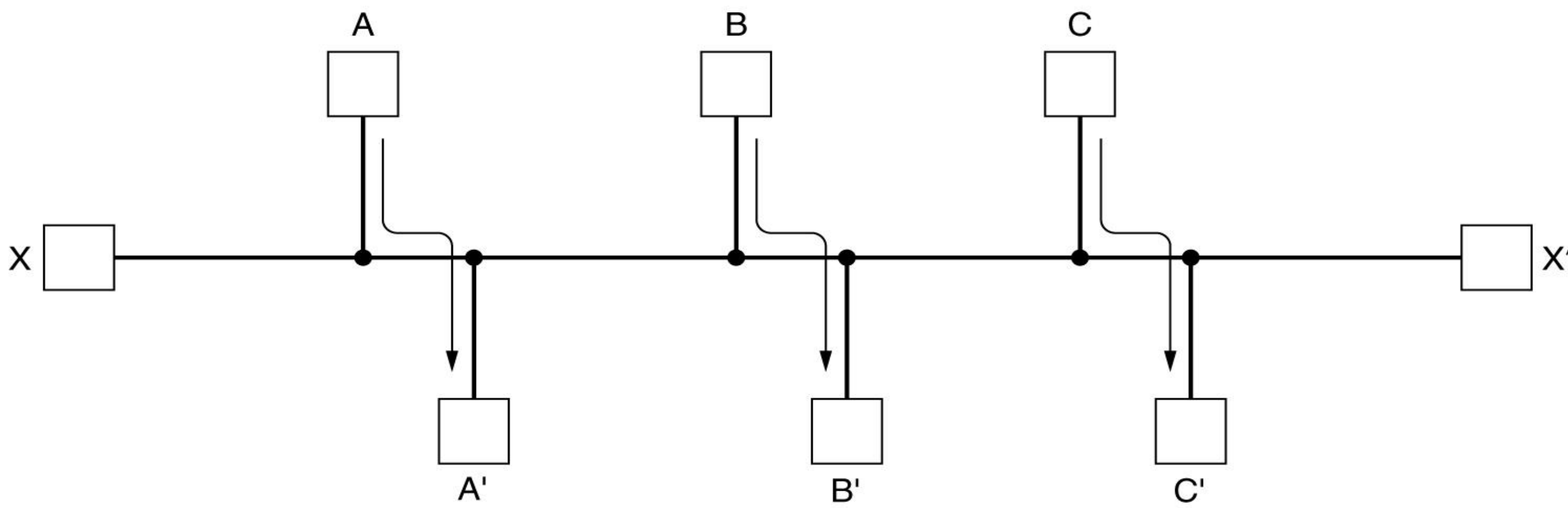
Comparison of Virtual-Circuit and Datagram Subnets

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

- The routing algorithm is the part of the network layer responsible for deciding which output line an incoming packet should be transmitted on.
- If the subnet uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time.
- If the subnet uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Therefore, data packets just follow the previously established route.
- Regardless to the above two schemes, it is desirable for a routing algorithm to have the following properties: **correctness, simplicity, robustness, stability, fairness, and optimality**.

Routing Algorithms



Conflict between fairness and optimality

Routing Algorithms

- Routing algorithms can be grouped into two major classes: **Nonadaptive** and **adaptive algorithms**.

- **Nonadaptive algorithms** do not base their routing decisions on the current traffic or topology. Instead, the route from a source to a destination is computed in advance, off-line, and downloaded to the nodes when the network is booted. This procedure is called **static routing**.

- **Adaptive algorithms**, in contrast, change their routing decisions to reflect changes in the topology and the traffic.

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

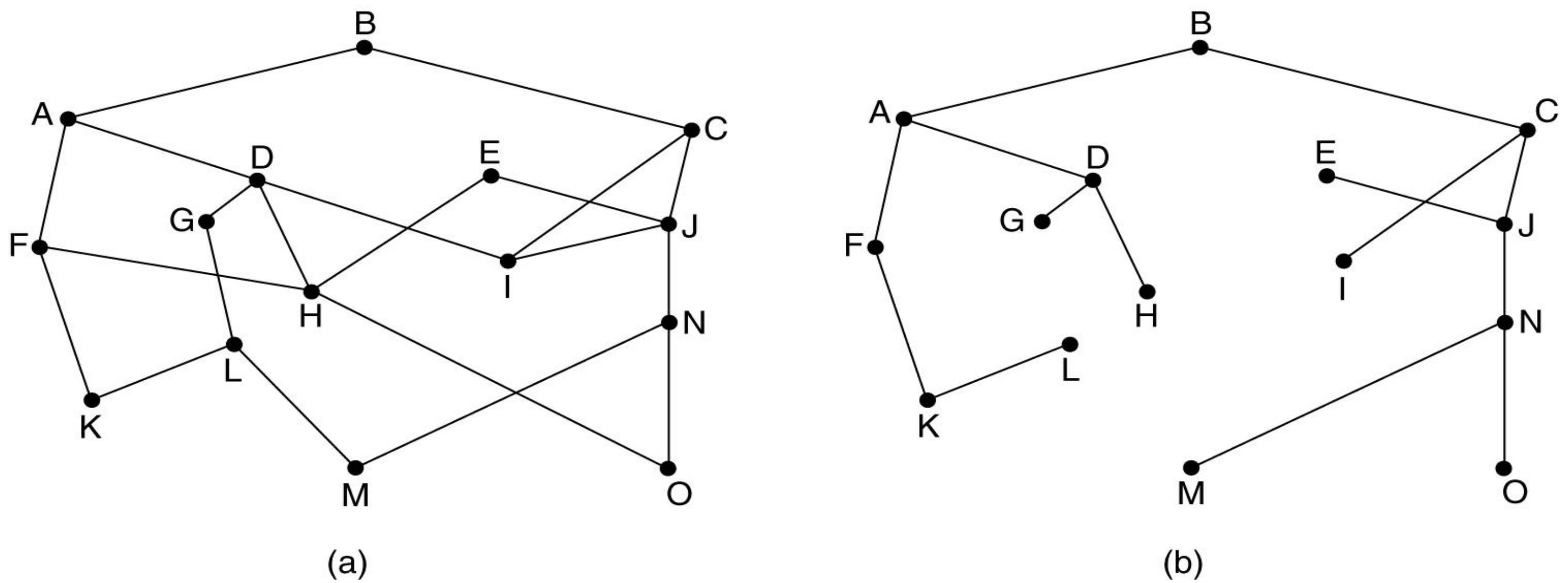
The Optimality Principle

- The **optimality principle** states that if the router ***J*** is on the optimal path from router ***I*** to router ***K***, then the optimal path from ***J*** to ***K*** falls along the same route.

Proof

- If ***r1*** is the part of the route from ***I*** to ***J*** and the rest of the route is ***r2***. If a route better than ***r2*** existed from ***I*** to ***K***, it could be concatenated with ***r1*** to improve the route from ***I*** to ***K***, contradicting our statement that ***r1 r2*** is optimal.

The Optimality Principle



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

The Sink Tree

- As a result from the optimality principle, the optimal routes from all sources to a given destination form a tree rooted at the destination. Such a tree is called **a sink tree**.
- The sink tree does not contain any loop, so each packet will be delivered within a finite and bounded number of hops.

Shortest Path Routing

- The idea is to build a graph for the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line (a link).

- Consider a directed graph $G = (N,A)$ with number of nodes N and number of arcs A, in which each arc (i,j) is assigned some real number dij as the length or distance of the arc.

- The length of any directed path $p = (i,j,k,\ldots,l,m)$ is defined as $dij + djk + \ldots + dlm$.

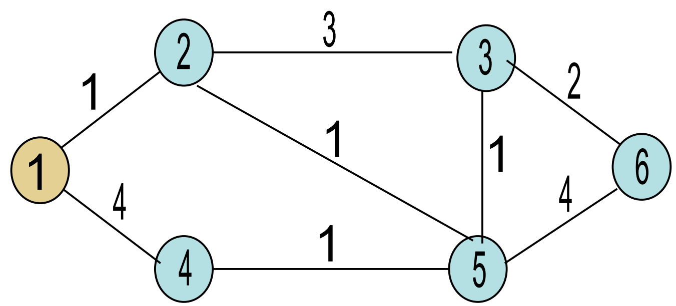
- Given any two nodes i, m of the graph, the shortest path problem is to find a minimum length (*i.e.*, shortest) directed path from i to m.

Bellman-Ford Algorithm

- The Bellman-Ford algorithm finds the shortest path from every node to a certain node (say, node 1).
- Let us denote $d_{ij} = \infty$ if (i,j) is not an arc of the graph.
- Also, we define D_i^h as the length of the shortest walk from the node i to node 1, subject to constraint that the walk contains at most h arcs and goes through node 1 only once.
- By convention, $D_1^h = 0$, for all h. Also, $D_i^0 = \infty$, for all $i \neq 1$.
- D_i^h can be generated by the iteration: $D_i^{h+1} = \min_j [d_{ij} + D_j^h]$, for all $i \neq 1$.
- The algorithm terminates after h iterations if $D_i^h = D_i^{h-1}$, for all i.

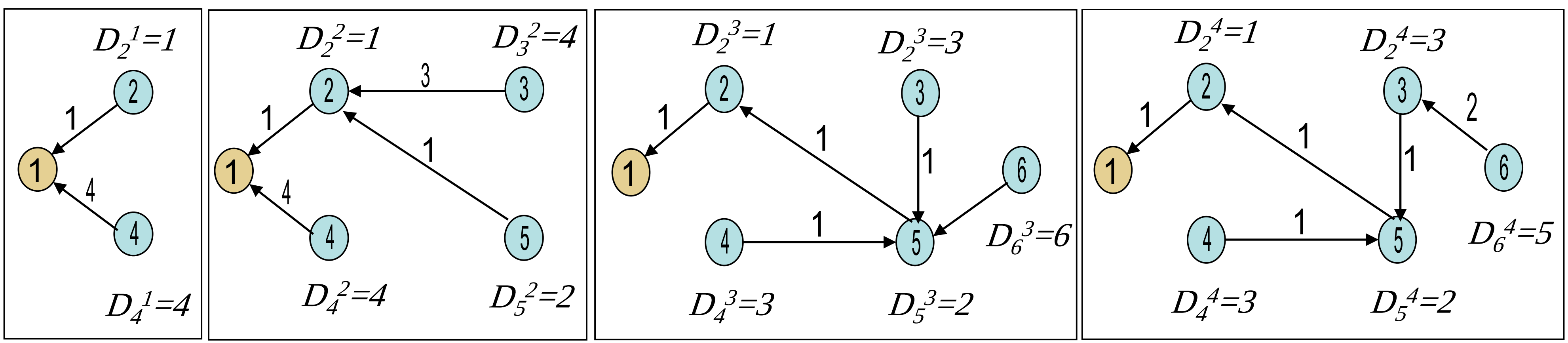
Bellman-Ford Algorithm

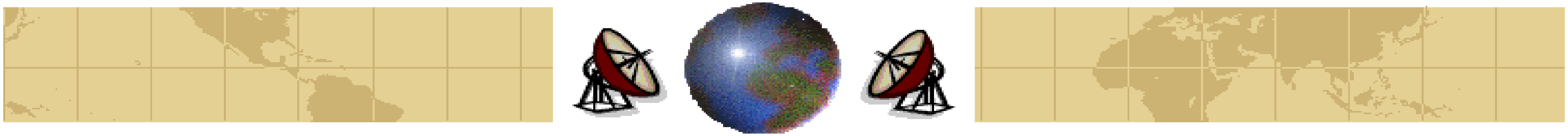
Example:



$d_{ij} = d_{ji}$ for all (i,j)

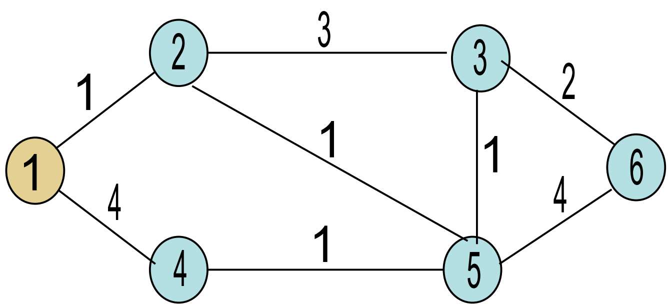
Bellman-Ford





Bellman-Ford Algorithm

Example:



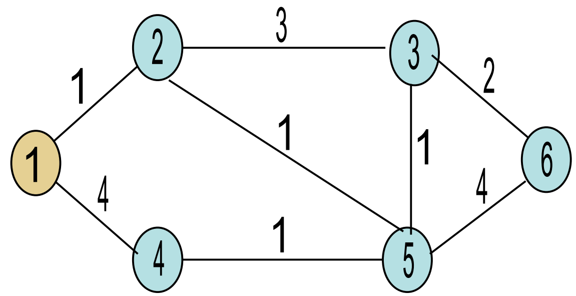
$d_{ij} = d_{ji}$ for all (i,j)

h	1	2	3	4	5	6
0	0	∞	∞	∞	∞	∞
1	0	1	∞	4	∞	∞
2	0	1	4	4	2	∞
3	0	1	3	3	2	6
4	0	1	3	3	2	5
5	0	1	3	3	2	5

Notes:

- Computation Complexity = $O(N3)$.
- The Bellman-Ford algorithm iterates on the number of the arcs in a path.

Bellman-Ford Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

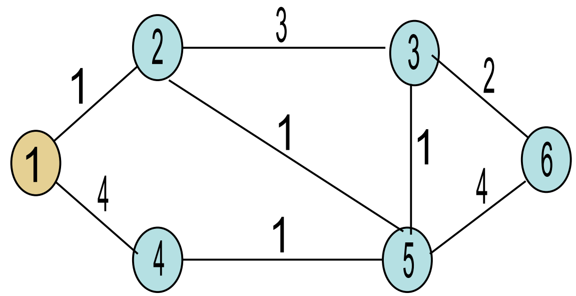
h=0

$D_1^0=0$, $D_2^0=\infty$, $D_3^0=\infty$, $D_4^0=\infty$, $D_5^0=\infty$, $D_6^0=\infty$

h=1

$D_1^1=0$, $D_2^1=1$ *(Direct)*, $D_3^1=\infty$, $D_4^1=4$ *(Direct)*, $D_5^1=\infty$, $D_6^1=\infty$

Bellman-Ford Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

h=2

$D_1^2 = 0$

$D_2^2 = 1$ *(direct)*

$D_3^2 = \min_j [d_{3j} + D_j^1] \quad \Rightarrow D_3^2 = \min_j [d_{32} + D_2^1, d_{35} + D_5^1, d_{36} + D_6^1]$

$\Rightarrow D_3^2 = \min_j [3+1, 1+\infty, 2+\infty] = 4$ *(Through node 2)*

$D_4^2 = \min_j [d_{4j} + D_j^1] \quad \Rightarrow D_4^2 = \min_j [d_{41} + D_1^1, d_{45} + D_5^1]$

$\Rightarrow D_4^2 = \min_j [4+0, 1+\infty] = 4$ *(Direct)*

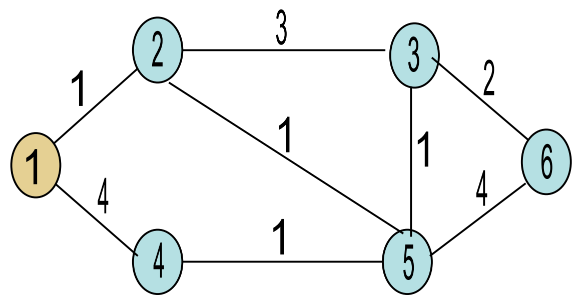
$D_5^2 = \min_j [d_{5j} + D_j^1] \quad \Rightarrow D_5^2 = \min_j [d_{52} + D_2^1, d_{53} + D_3^1, d_{54} + D_4^1, d_{56} + D_6^1]$

$\Rightarrow D_5^2 = \min_j [1+1, 1+\infty, 1+4, 4+\infty] = 2$ *(Through node 2)*

$D_6^2 = \min_j [d_{6j} + D_j^1] \quad \Rightarrow D_6^2 = \min_j [d_{63} + D_3^1, d_{65} + D_5^1]$

$\Rightarrow D_6^2 = \min_j [2+\infty, 4+\infty] = \infty$

Bellman-Ford Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

h=3

$D_1^3=0$

$D_2^3=1$ *(direct)*

$D_3^3=\min_j [d_{3j}+D_j^2] \quad \Rightarrow D_3^3=\min_j [d_{32}+D_2^2, d_{35}+D_5^2, d_{36}+D_6^2]$

$\Rightarrow D_3^3=\min_j [3+1, 1+2, 2+\infty] = 3$ *(Through node 5)*

$D_4^3=\min_j [d_{4j}+D_j^2] \quad \Rightarrow D_4^3=\min_j [d_{41}+D_1^2, d_{45}+D_5^2]$

$\Rightarrow D_4^3=\min_j [4+0, 1+2] = 3$ *(Through node 5)*

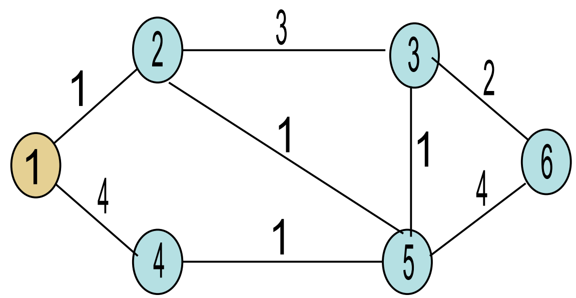
$D_5^3=\min_j [d_{5j}+D_j^2] \quad \Rightarrow D_5^3=\min_j [d_{52}+D_2^2, d_{53}+D_3^2, d_{54}+D_4^2, d_{56}+D_6^2]$

$\Rightarrow D_5^3=\min_j [1+1, 1+4, 1+4, 4+\infty] = 2$ *(Through node 2)*

$D_6^3=\min_j [d_{6j}+D_j^2] \quad \Rightarrow D_6^3=\min_j [d_{63}+D_3^2, d_{65}+D_5^2]$

$\Rightarrow D_6^3=\min_j [2+4, 4+2] = 6$

Bellman-Ford Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

h=4

$D_1^4 = 0$

$D_2^4 = 1$ *(direct)*

$D_3^4 = \min_j [d_{3j} + D_j^3] \quad \Rightarrow D_3^4 = \min_j [d_{32} + D_2^3, d_{35} + D_5^3, d_{36} + D_6^3]$

$\Rightarrow D_3^4 = \min_j [3+1, 1+2, 2+6] = 3$ *(Through node 5)*

$D_4^4 = \min_j [d_{4j} + D_j^3] \quad \Rightarrow D_4^4 = \min_j [d_{41} + D_1^3, d_{45} + D_5^3]$

$\Rightarrow D_4^4 = \min_j [4+0, 1+2] = 3$ *(Through node 5)*

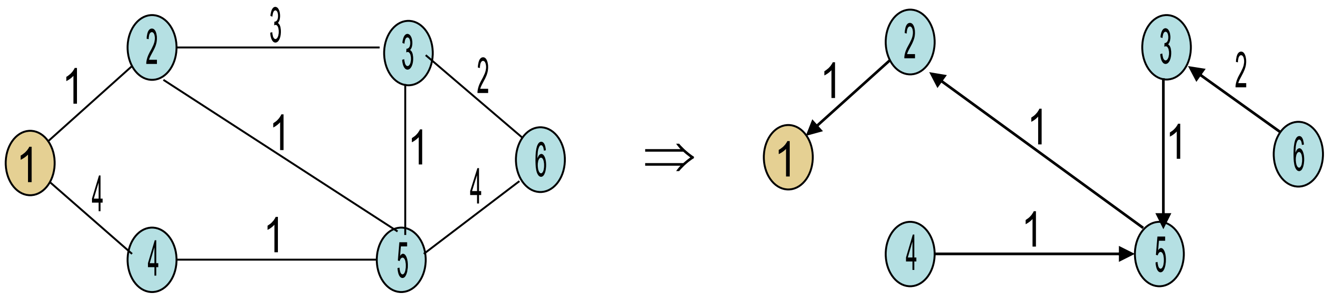
$D_5^4 = \min_j [d_{5j} + D_j^3] \quad \Rightarrow D_5^4 = \min_j [d_{52} + D_2^3, d_{53} + D_3^3, d_{54} + D_4^3, d_{56} + D_6^3]$

$\Rightarrow D_5^4 = \min_j [1+1, 1+4, 1+4, 4+6] = 2$ *(Through node 2)*

$D_6^4 = \min_j [d_{6j} + D_j^3] \quad \Rightarrow D_6^4 = \min_j [d_{63} + D_3^3, d_{65} + D_5^3]$

$\Rightarrow D_6^4 = \min_j [2+3, 4+2] = 5$ *(Through node 3)*

Bellman-Ford Algorithm



h=5

$D_1^5 = 0$

$D_2^5 = 1$ *(direct)*

$D_3^5 = \min_j [d_{3j} + D_j^4] \quad \Rightarrow D_3^5 = \min_j [d_{32} + D_2^4, d_{35} + D_5^4, d_{36} + D_6^4]$

$\Rightarrow D_3^5 = \min_j [3+1, 1+2, 2+6] = 3$ *(Through node 5)*

$D_4^5 = \min_j [d_{4j} + D_j^4] \quad \Rightarrow D_4^5 = \min_j [d_{41} + D_1^4, d_{45} + D_5^4]$

$\Rightarrow D_4^5 = \min_j [4+0, 1+2] = 3$ *(Through node 5)*

$D_5^5 = \min_j [d_{5j} + D_j^4] \quad \Rightarrow D_5^5 = \min_j [d_{52} + D_2^4, d_{53} + D_3^4, d_{54} + D_4^4, d_{56} + D_6^4]$

$\Rightarrow D_5^5 = \min_j [1+1, 1+4, 1+4, 4+6] = 2$ *(Through node 2)*

$D_6^5 = \min_j [d_{6j} + D_j^4] \quad \Rightarrow D_6^5 = \min_j [d_{63} + D_3^4, d_{65} + D_5^4]$

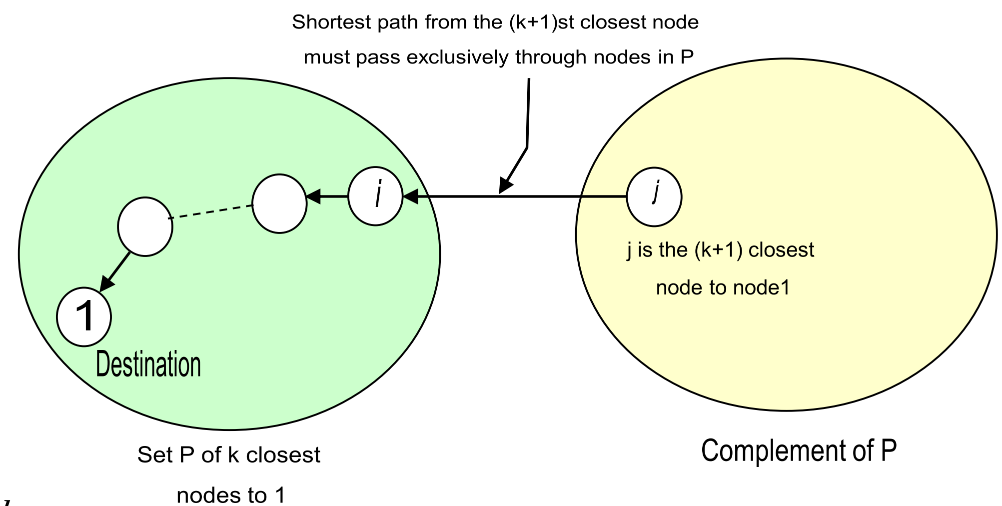
$\Rightarrow D_6^5 = \min_j [2+3, 4+2] = 5$ *(Through node 3)*

Dijkstra's Algorithm

- Dijkstra's algorithm iterates on the length of the path.
- The general idea is to find the shortest paths to a destination (node 1) in order of increasing path length.
- The shortest of the shortest paths to node 1 must be a single arc from the closest neighbor of node 1, since any multiple-arc path cannot be shorter than the first arc length because of nonnegative-length of any arc.
- The next shortest of the shortest paths must either be a single-arc path from the next closest neighbor of 1 or the shortest two-arc path through the previously chosen node, and so on.

Dijkstra's Algorithm

- We define D_i as the **estimate** of the shortest path length from the node i to node 1.
- When the estimate becomes certain, we regard the node as being **permanently labeled** and keep track of this with a set P of permanently labeled nodes.
- The nodes added to P at each step will be the closest to node 1.



Dijkstra's Algorithm

The Detailed Algorithm:

Initially, $P = \{1\}$, $D_1 = 0$, and $D_j = d_{j1}$, for all $j \neq 1$.

Step 1: (Find the next closest node.) Find $i \notin P$ such that

$$D_i = \min D_j, \text{ for all } j \notin P.$$

Set $P := P \cup \{i\}$. If P contains all nodes, then stop; the algorithm is complete.

Step 2: (Updating the labels.) For all $j \notin P$ set

$$D_j = \min [D_j , d_{ji} + D_i]$$

Go to step 1.

Dijkstra's Algorithm

```
#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

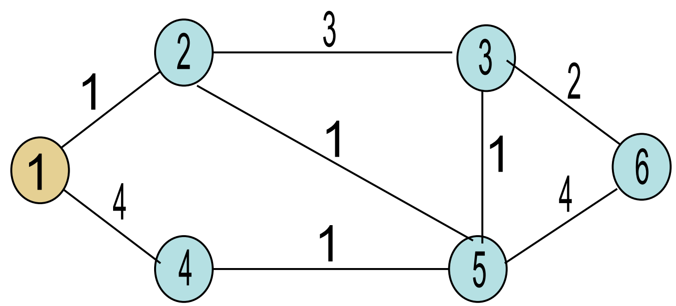
Dijkstra's Algorithm

```
  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

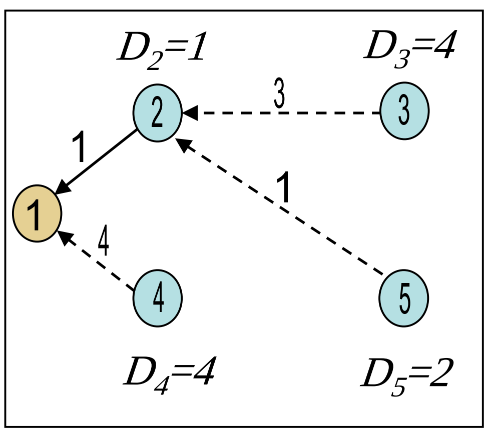
Dijkstra's Algorithm

Example:

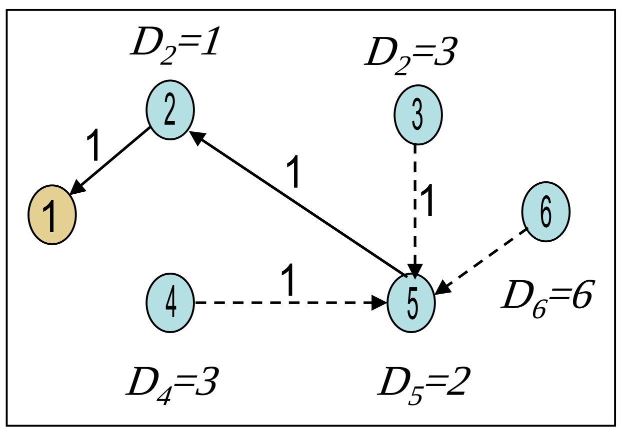


$d_{ij} = d_{ji}$ for all (i,j)

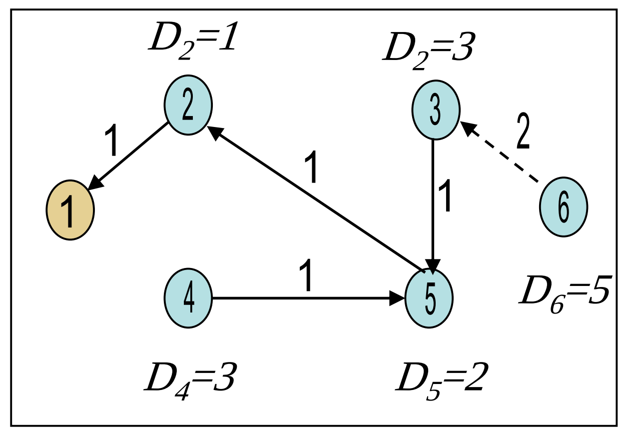
Dijkstra's Algorithm



$P = \{1 , 2\}$

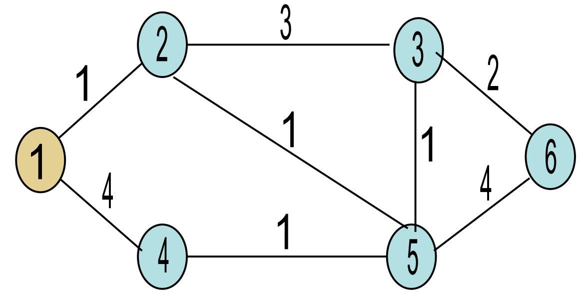


$P = \{1 , 2 , 5\}$



$P = \{1 , 2 , 3 , 4 , 5\}$

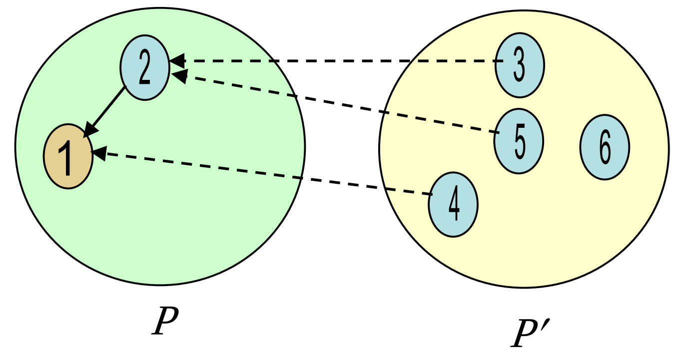
Dijkstra's Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

$P=\{1\}$, $D_1=0$, $D_j=d_{j1}$ for $j\neq 1$.

$\Rightarrow D_2=1$ *(Direct)*, $D_3=\infty$, $D_4=4$ *(Direct)*, $D_5=\infty$, $D_6=\infty$



Step1:

$D_i=\min_{j\notin P}[\,D_j\,] \Rightarrow D_i=\min_{j\in\{2,3,4,5,6\}}=[1, \infty , 4 , \infty, \infty]$

$\Rightarrow D_i=1$ for $i=2 \Rightarrow D_2=1$ *(Direct)*

$\Rightarrow P=\{1 , 2\}$

Step2:

$j\notin P \Rightarrow j\in\{3, 4, 5, 6\}$, $i=2$, $D_2=1$

$$D_j=\min\,[D_j\,,\,d_{j2}+D_2]$$

$j=3$, $D_3=\min[D_3\,,\,d_{32}+D_2] \Rightarrow D_3=\min[\infty,\,3+1] \Rightarrow D_3=4$ *(Through node 2)*

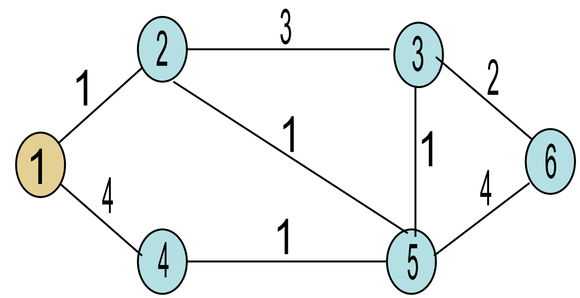
$j=4$, $D_4=\min[D_4\,,\,d_{42}+D_2] \Rightarrow D_4=\min[4,\,\infty+1] \Rightarrow D_4=4$ *(Direct)*

$j=5$, $D_5=\min[D_5\,,\,d_{52}+D_2] \Rightarrow D_5=\min[\infty,\,1+1] \Rightarrow D_5=2$ *(Through node 2)*

$j=6$, $D_6=\min[D_6\,,\,d_{62}+D_2] \Rightarrow D_6=\min[\infty,\,\infty+1] \Rightarrow D_6=\infty$

Go to step 1.

Dijkstra's Algorithm



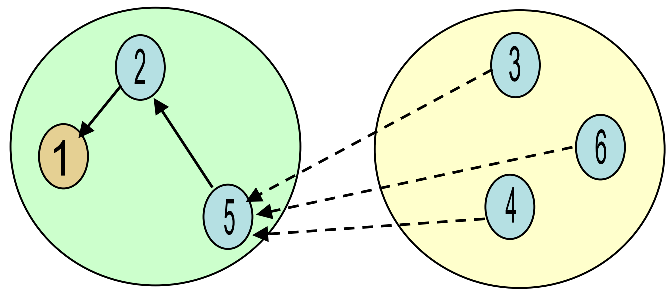
$d_{ij} = d_{ji}$ for all (i,j)

Step1:

$D_i = min_{\,j \notin P}[\, D_j] \Rightarrow D_i = min_{\,j \in \{3,4,5,6\}} = [4, 4 , 2 , \infty]$

$\Rightarrow D_i = 2$ for $i = 5 \Rightarrow D_5 = 2$ *(Through node 2)*

$\Rightarrow P = \{1 , 2 , 5\}$



P $\quad$ P'

Step2:

$j \notin P \Rightarrow j \in \{3, 4, 6\}$, $i=5 , D_5=2$

$$D_j = \min\,[D_j , d_{j5} + D_5]$$

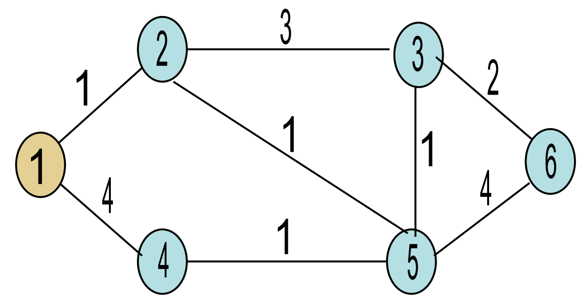
$j=3$, $D_3 = min[D_3 , d_{35} + D_5] \Rightarrow D_3 = min[4, 1+2] \Rightarrow D_3 = 3$ *(Through node 5)*

$j=4$, $D_4 = min[D_4 , d_{45} + D_5] \Rightarrow D_4 = min[4, 1+2] \Rightarrow D_4 = 3$ *(Through node 5)*

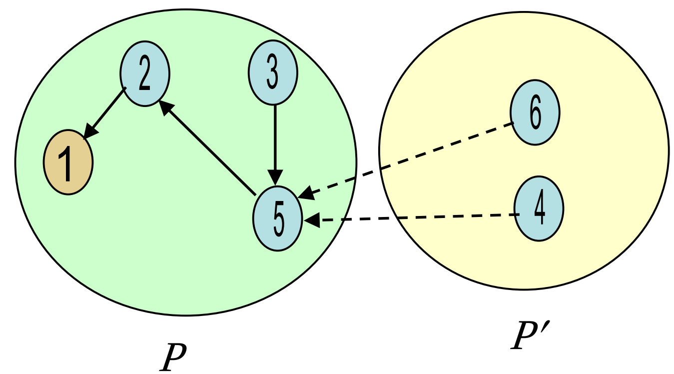
$j=6$, $D_6 = min[D_6 , d_{65} + D_5] \Rightarrow D_6 = min[\infty, 4+2] \Rightarrow D_6 = 6$ *(Through node 5)*

Go to step 1.

Dijkstra's Algorithm



$d_{ij} = d_{ji}$ for all (i,j)



Step1:

$D_i = min_{\,j \notin P}[\,D_j] \Rightarrow D_i = min_{\,j \in \{3,4,6\}} = [3, 3, 6]$

Randomly, pick one, say node 3.

$\Rightarrow D_i = 3$ for $i = 3 \Rightarrow D_3 = 3$ *(Through node 5)*

$\Rightarrow P = \{1, 2, 3, 5\}$

Step2:

$j \notin P \Rightarrow j \in \{4, 6\}$, $i=3$, $D_3=3$

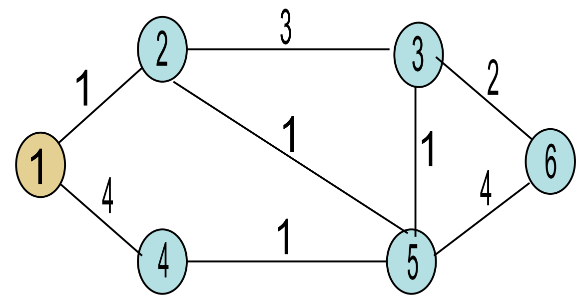
$D_j = \min\,[D_j, d_{j3} + D_3]$

$j=4$, $D_4 = min[D_4, d_{43} + D_3] \Rightarrow D_4 = min[3, \infty + 3] \Rightarrow D_4 = 3$ *(Through node 5)*

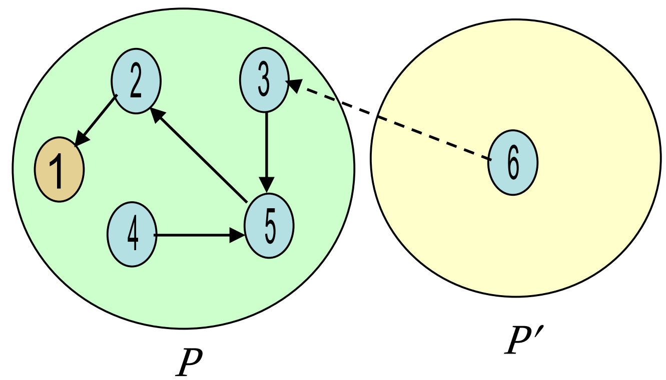
$j=6$, $D_6 = min[D_6, d_{63} + D_3] \Rightarrow D_6 = min[6, 2+3] \Rightarrow D_6 = 5$ *(Through node 3)*

Go to step 1.

Dijkstra's Algorithm



$d_{ij} = d_{ji}$ for all (i,j)



Step1:

$D_i = \min_{j \notin P} [D_j] \Rightarrow D_i = \min_{j \in \{4,6\}} = [3, 5]$

$\Rightarrow D_i = 3$ for $i = 4 \Rightarrow D_4 = 3$ *(Through node 5)*

$\Rightarrow P = \{1, 2, 3, 4, 5\}$

Step2:

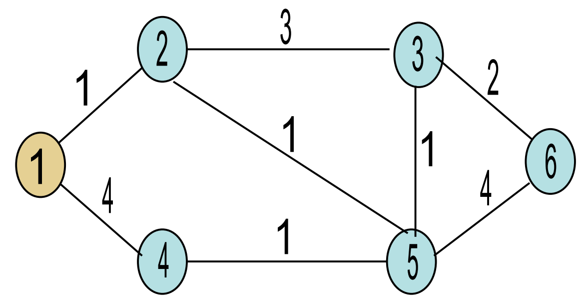
$j \notin P \Rightarrow j \in \{6\}$, $i=4$, $D_4=3$

$$D_j = \min [D_j , d_{j4} + D_4]$$

$j = 6$, $D_6 = \min[D_6 , d_{64} + D_4] \Rightarrow D_6 = \min[5, \infty + 3] \Rightarrow D_6 = 5$ *(Through node 3)*

Go to step 1.

Dijkstra's Algorithm



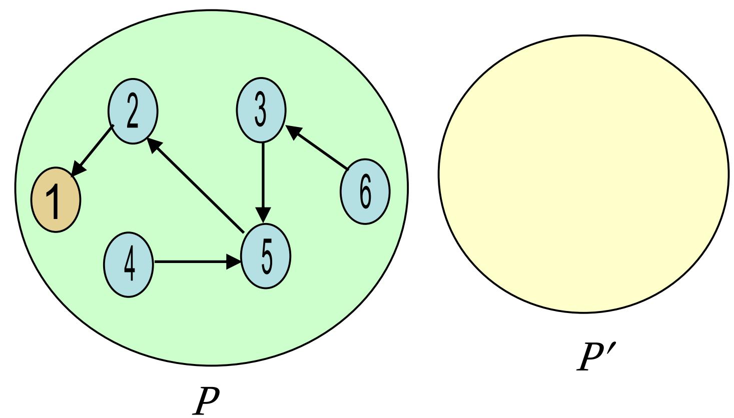
$d_{ij} = d_{ji}$ for all (i,j)

Step1:

$D_i = min_{\,j \notin P} [\, D_j \,] \Rightarrow D_i = min_{\,j \in \{6\}} = [5]$

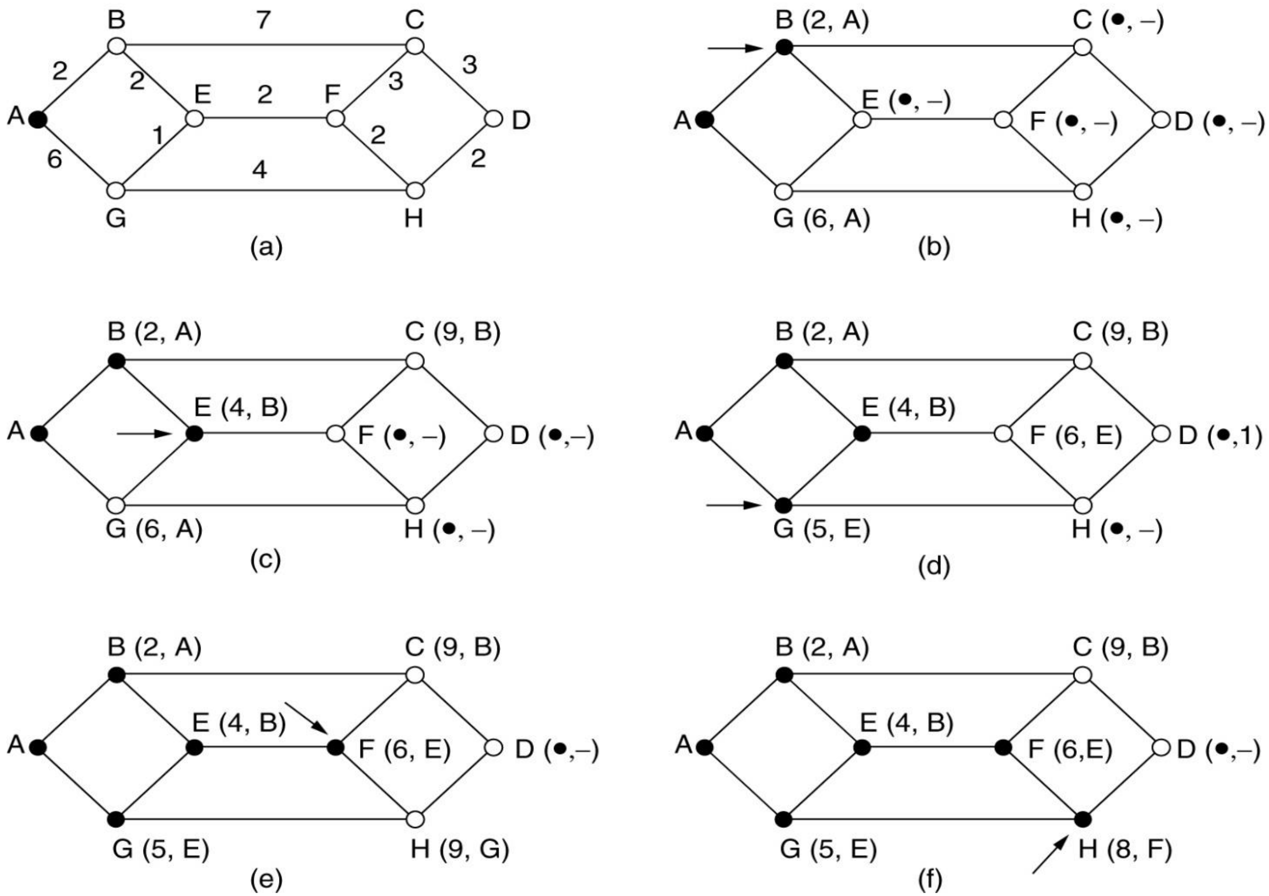
$\Rightarrow D_i = 5$ for $i = 6 \Rightarrow D_6 = 5$ *(Through node 3)*

$\Rightarrow P = \{1 , 2 , 3 , 4 , 5 , 6\}$



P contains all nodes, stop

Dijkstra's Algorithm



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

Floyd-Warshall Algorithm

- The **Floyd-Warshall** algorithm, unlike the previous two, finds the shortest paths between all pairs of nodes together.

- It iterates on the set of nodes that are allowed as intermediate nodes on the paths.

- It starts with single arc distances (*i.e.*, no intermediate nodes) as starting estimates of the shortest path lengths.

- It then calculates the shortest paths under the constraint that only node 1 can be used as intermediate node, and then with constraint that only 1 and 2 can be used, and so forth.

- Let D_{ij}^{n} be the shortest path length from node i to j with the constraint that only nodes 1, 2, …, n can be used as intermediate nodes on paths.

Floyd-Warshall Algorithm

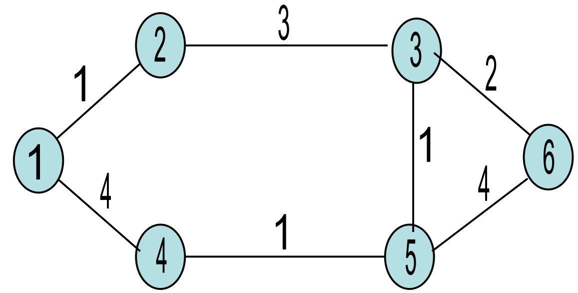
The Detailed Algorithm:

Initially, $D_{ij}^{0} = d_{ij}$, for all i & j, $i \neq j$.

For $n = 0, 1, 2, \ldots, N\text{-}1$,

$$D_{ij}^{n+1} = \min[\, D_{ij}^{n}, D_{i(n+1)}^{n} + D_{(n+1)j}^{n} \,] \quad \text{for all } i \neq j.$$

Floyd-Warshall Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

Example:

Step 0: Direct Link

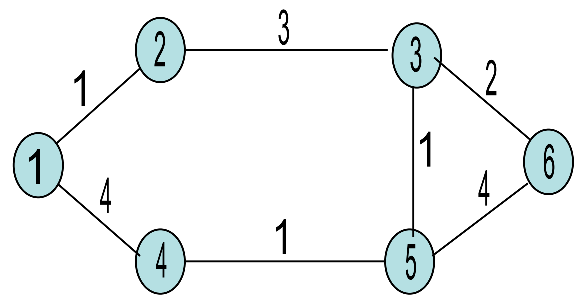
Cost for the direct Link

Last Node Before Target

$$D^{(0)} = \begin{bmatrix} 0 & 1 & \infty & 4 & \infty & \infty \\ 1 & 0 & 3 & \infty & \infty & \infty \\ \infty & 3 & 0 & \infty & 1 & 2 \\ 4 & \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & 1 & 1 & 0 & 4 \\ \infty & \infty & 2 & \infty & 4 & 0 \end{bmatrix}$$

$$\prod^{(0)} = \begin{bmatrix} N & 1 & N & 1 & N & N \\ 2 & N & 2 & N & N & N \\ N & 3 & N & N & 3 & 3 \\ 4 & N & N & N & 4 & N \\ N & N & 5 & 5 & N & 5 \\ N & N & 6 & N & 6 & N \end{bmatrix}$$

Floyd-Warshall Algorithm



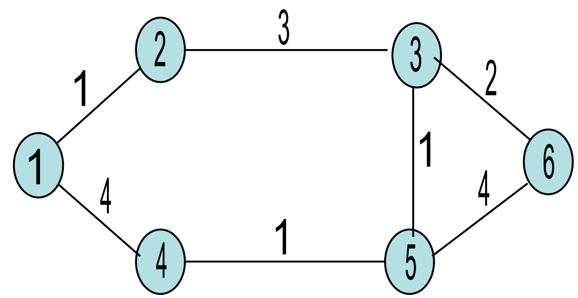
$d_{ij} = d_{ji}$ for all (i,j)

Step 1: Node 1 is the new intermediate node

$$D^{(1)} = \begin{bmatrix} 0 & 1 & \infty & 4 & \infty & \infty \\ 1 & 0 & 3 & 5 & \infty & \infty \\ \infty & 3 & 0 & \infty & 1 & 2 \\ 4 & 5 & \infty & 0 & 1 & \infty \\ \infty & \infty & 1 & 1 & 0 & 4 \\ \infty & \infty & 2 & \infty & 4 & 0 \end{bmatrix}$$

$$\Pi^{(1)} = \begin{bmatrix} N & 1 & N & 1 & N & N \\ 2 & N & 2 & 1 & N & N \\ N & 3 & N & N & 3 & 3 \\ 4 & 1 & N & N & 4 & N \\ N & N & 5 & 5 & N & 5 \\ N & N & 6 & N & 6 & N \end{bmatrix}$$

Floyd-Warshall Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

Step 2: Node 2 is the new intermediate node

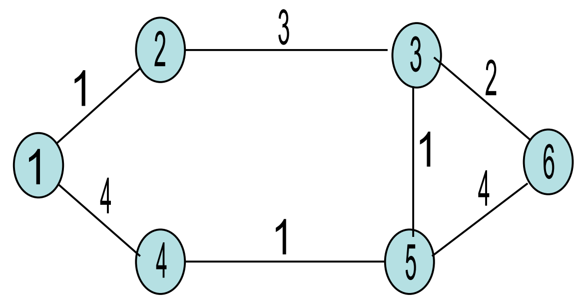
$$D^{(2)} = \begin{bmatrix} 0 & 1 & 4 & 4 & \infty & \infty \\ 1 & 0 & 3 & 5 & \infty & \infty \\ 4 & 3 & 0 & 8 & 1 & 2 \\ 4 & 5 & 8 & 0 & 1 & \infty \\ \infty & \infty & 1 & 1 & 0 & 4 \\ \infty & \infty & 2 & \infty & 4 & 0 \end{bmatrix}$$

$$\Pi^{(2)} = \begin{bmatrix} N & 1 & 2 & 1 & N & N \\ 2 & N & 2 & 1 & N & N \\ 2 & 3 & N & 2 & 3 & 3 \\ 4 & 1 & 2 & N & 4 & N \\ N & N & 5 & 5 & N & 5 \\ N & N & 6 & N & 6 & N \end{bmatrix}$$

$$D_{13}^{(2)} = \min[D_{13}^{(1)}, D_{12}^{(1)} + D_{23}^{(1)}] = [\infty, 1+3] = 4$$

$$D_{34}^{(2)} = \min[D_{34}^{(1)}, D_{32}^{(1)} + D_{24}^{(1)}] = [\infty, 3+5] = 8$$

Floyd-Warshall Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

Step 3: Node 3 is the new intermediate node

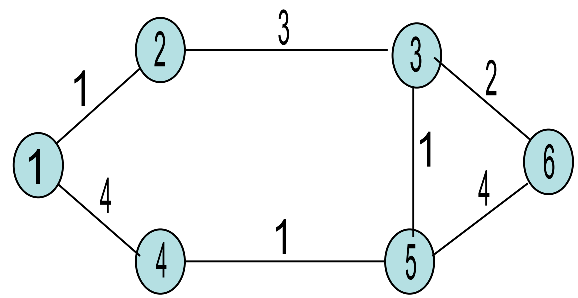
$$D^{(3)} = \begin{bmatrix} 0 & 1 & 4 & 4 & 5 & 6 \\ 1 & 0 & 3 & 5 & 4 & 5 \\ 4 & 3 & 0 & 8 & 1 & 2 \\ 4 & 5 & 8 & 0 & 1 & 10 \\ 5 & 4 & 1 & 1 & 0 & 3 \\ 6 & 5 & 2 & 10 & 3 & 0 \end{bmatrix}$$

$$\prod^{(3)} = \begin{bmatrix} N & 1 & 2 & 1 & 3 & 3 \\ 2 & N & 2 & 1 & 3 & 3 \\ 2 & 3 & N & 2 & 3 & 3 \\ 4 & 1 & 2 & N & 4 & 3 \\ 3 & 3 & 5 & 5 & N & 3 \\ 3 & 3 & 6 & 3 & 3 & N \end{bmatrix}$$

$$D_{15}^{(3)} = \min[D_{15}^{(2)}, D_{13}^{(2)} + D_{35}^{(2)}] = [\infty, 4+1] = 5$$

$$D_{56}^{(3)} = \min[D_{56}^{(2)}, D_{53}^{(2)} + D_{36}^{(2)}] = [4, 1+2] = 3$$

Floyd-Warshall Algorithm



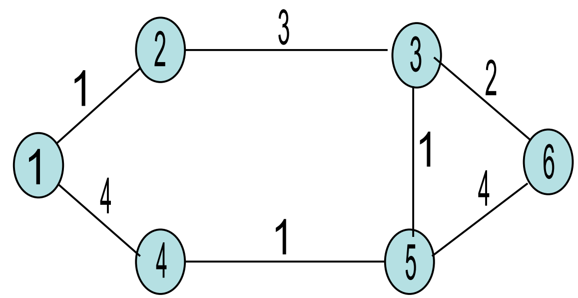
$d_{ij} = d_{ji}$ for all (i,j)

Step 4: Node 4 is the new intermediate node

$$D^{(4)} = \begin{bmatrix} 0 & 1 & 4 & 4 & 5 & 6 \\ 1 & 0 & 3 & 5 & 4 & 5 \\ 4 & 3 & 0 & 8 & 1 & 2 \\ 4 & 5 & 8 & 0 & 1 & 10 \\ 5 & 4 & 1 & 1 & 0 & 3 \\ 6 & 5 & 2 & 10 & 3 & 0 \end{bmatrix}$$

$$\prod^{(4)} = \begin{bmatrix} N & 1 & 2 & 1 & 3 & 3 \\ 2 & N & 2 & 1 & 3 & 3 \\ 2 & 3 & N & 2 & 3 & 3 \\ 4 & 1 & 2 & N & 4 & 3 \\ 3 & 3 & 5 & 5 & N & 3 \\ 3 & 3 & 6 & 3 & 3 & N \end{bmatrix}$$

Floyd-Warshall Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

Step 5: Node 5 is the new intermediate node

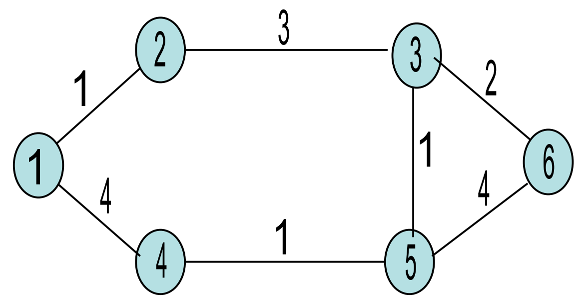
$$D^{(5)} = \begin{bmatrix} 0 & 1 & 4 & 4 & 5 & 6 \\ 1 & 0 & 3 & 5 & 4 & 5 \\ 4 & 3 & 0 & 2 & 1 & 2 \\ 4 & 5 & 2 & 0 & 1 & 4 \\ 5 & 4 & 1 & 1 & 0 & 3 \\ 6 & 5 & 2 & 4 & 3 & 0 \end{bmatrix}$$

$$\prod^{(5)} = \begin{bmatrix} N & 1 & 2 & 1 & 3 & 3 \\ 2 & N & 2 & 1 & 3 & 3 \\ 2 & 3 & N & 5 & 3 & 3 \\ 4 & 1 & 5 & N & 4 & 5 \\ 3 & 3 & 5 & 5 & N & 3 \\ 3 & 3 & 6 & 5 & 3 & N \end{bmatrix}$$

$$D_{34}^{(5)} = \min[D_{34}^{(4)}, D_{35}^{(4)} + D_{54}^{(4)}] = [8, 1+1] = 2$$

$$D_{46}^{(5)} = \min[D_{46}^{(5)}, D_{45}^{(5)} + D_{56}^{(5)}] = [10, 1+3] = 4$$

Floyd-Warshall Algorithm



$d_{ij} = d_{ji}$ for all (i,j)

Step 6: Node 6 is the new intermediate node

$$D^{(6)} = \begin{bmatrix} 0 & 1 & 4 & 4 & 5 & 6 \\ 1 & 0 & 3 & 5 & 4 & 5 \\ 4 & 3 & 0 & 2 & 1 & 2 \\ 4 & 5 & 2 & 0 & 1 & 4 \\ 5 & 4 & 1 & 1 & 0 & 3 \\ 6 & 5 & 2 & 4 & 3 & 0 \end{bmatrix}$$

$$\Pi^{(6)} = \begin{bmatrix} N & 1 & 2 & 1 & 3 & 3 \\ 2 & N & 2 & 1 & 3 & 3 \\ 2 & 3 & N & 5 & 3 & 3 \\ 4 & 1 & 5 & N & 4 & 5 \\ 3 & 3 & 5 & 5 & N & 3 \\ 3 & 3 & 6 & 5 & 3 & N \end{bmatrix}$$

Example:

To get the best path from 1 to 6, consider row 1 of $\Pi^{(6)}$. It shows that the node before the target (6) is 3.

To get the best path from 1 to 3, consider row 1 of $\Pi^{(6)}$. It shows that the node before the target (3) is 2.

To get the best path from 1 to 2, consider row 1 of $\Pi^{(6)}$. It shows that the node before the target (2) is 1.

Therefore, the path from 1 to 6 pass through 2 first, then through 3 before reaching the target 6.

Flooding

- Flooding is a static algorithm, in which every incoming packet is sent out on every outgoing line except the one it arrived on.
- Flooding generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process.
- One such measure is to have a hop counter contained in the header of each packet, which is decremented at each hop, with packet being discarded when the counter reaches zero.
- A variation of flooding is selective flooding. In this algorithm, the routers do not send every incoming packet out every line, only on those lines that are going approximately in the right direction.
- Flooding is used as a metric against which other routing algorithms can be compared. Flooding always chooses the shortest path and can produce the shorter delay.

Flow-Based Routing

- To use this algorithm, we must know the following information in advance:
  - The subnet topology.
  - The traffic matrix (F_{ij}).
  - The Line Capacity matrix (C_{ij}).
  - The tentative routing algorithm.

- In some network, the mean data flow between each pair of nodes is relatively stable and predictable.

- The basic idea is that for a given line, the capacity and average flow is known, it is possible to compute the mean packet delay on the line from **Queueing Theory**. From the mean delays of all the lines, it is easy then to calculate the mean packet delay per line for the entire subnet.

Flow-Based Routing

- We can calculate the mean delay for each line (T_i) using the **Queueing Theory** formula:

$$T_i = \frac{1}{\frac{C_i}{L} - \lambda_i}$$

where L is the mean packet size in bits,

C_i is the capacity in kps of a channel *i*, and

λ_i is the mean flow in packets/sec of a channel *i*.

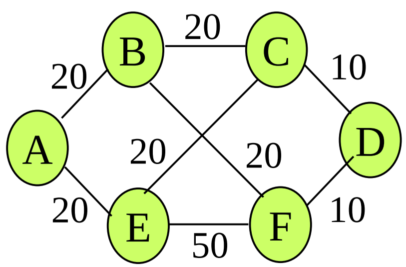
- To compute the mean time (T) for the entire subnet:

$$T = \sum_i T_i W_i$$

where $W_i = \frac{\lambda_i}{\sum_i \lambda_i}$

Flow-Based Routing

Example:



Destinations / Sources

	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA		8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB		3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC		3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD		5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	

(a) A subnet with line capacities shown in kbps.

(b) The traffic in packets/sec and the routing matrix.

Flow-Based Routing

Example: *Continued.*

$$\lambda = \sum_i \lambda_i = 82\, packets/\sec$$

$$\sum_i W_i = 1$$

$$T = \sum_i T_i W_i = 86\, m\sec$$

l	Line	λ_i (pkts/sec)	C_i (kbps)	C_i / L (pkts/sec)	T_i (msec)	Weight
1	AB	14	20	25	91	0.171
2	BC	12	20	25	77	0.146
3	CD	6	10	12.5	154	0.073
4	AE	11	20	25	71	0.134
5	EF	13	50	62.5	20	0.159
6	FD	8	10	12.5	222	0.098
7	BF	10	20	25	67	0.122
8	EC	8	20	25	59	0.098

Analysis of a subnet using a mean packet size of 800 bits.

The reverse traffic (BA, CB, etc.) is the same as the forward traffic.

Queueing Theory

$$T_i = \frac{1}{\frac{C_i}{L} - \lambda_i}$$

Proof:

$\overline{N}$ = Average number of customers in the system.

λ_i = mean flow in packets/sec of a channel *i*.

T = Time spent in the system.

W = Time spent in the Queue.

$\overline{N}_q$ = Average number of customers in the queue.

$\overline{N}_s$ = Average number of customers in the service = ρ, where $0 \leq \rho \leq 1$.

$\overline{x}$ = Channel service time.

μ = Channel service rate.

ρ = Channel Utilization.

P_k = Probability that there are k customers in the system.

P_0 = Probability that the system is idle = $1 - \rho$.

L = mean packet size in bits.

C_i = The capacity in kbps of a channel *i*.

Queueing Theory

Little Theorem:

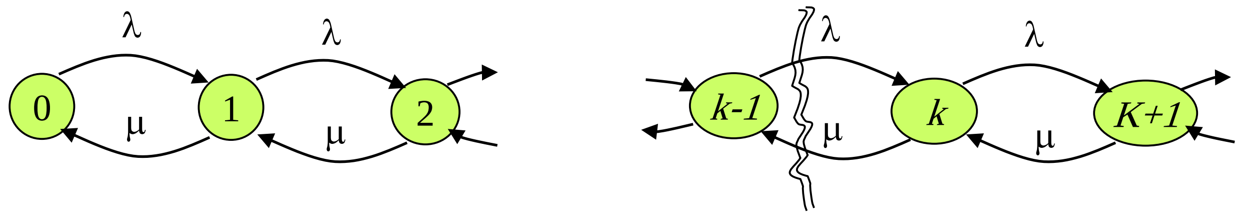
$$\overline{N} = \lambda T$$

$$\overline{N}_q = \lambda W$$

$$\overline{N}_s = \lambda \overline{x}$$

$$\rho = \overline{N}_s \text{ and } \overline{x} = 1/\mu \Rightarrow \rho = \lambda/\mu$$

Queueing Theory



[*M*/*M*/1 Model:]

$$\mu P_k = \lambda P_{k-1}$$

$$P_k = \frac{\lambda}{\mu} P_{k-1}$$

$$P_k = \rho P_{k-1}$$

$$P_k = \rho^2 P_{k-2}$$

$$P_k = \rho^3 P_{k-3}$$

$$P_k = \rho^k P_0$$

$$P_k = \rho^k (1-\rho)$$

Queueing Theory

$$\overline{N} = \sum_{k=0}^{\infty} kP_k$$

$$\overline{N} = \sum_{k=0}^{\infty} k\rho^k (1-\rho)$$

$$\overline{N} = (1-\rho)\sum_{k=0}^{\infty} k\rho^k$$

$$\overline{N} = \rho(1-\rho)\sum_{k=0}^{\infty} k\rho^{k-1}$$

$$\overline{N} = \rho(1-\rho)\sum_{k=0}^{\infty} \frac{d}{d\rho}\rho^k$$

$$\overline{N} = \rho(1-\rho)\frac{d}{d\rho}\sum_{k=0}^{\infty} \rho^k$$

$$\overline{N} = \rho(1-\rho)\frac{d}{d\rho}\left(\frac{1}{1-\rho}\right)$$

$$\overline{N} = \rho(1-\rho)\frac{1}{(1-\rho)^2}$$

$$\overline{N} = \frac{\rho}{(1-\rho)}$$

Queueing Theory

$$\overline{N} = \lambda T$$

$$T = \frac{\overline{N}}{\lambda}$$

$$T = \frac{\rho}{\lambda(1-\rho)}$$

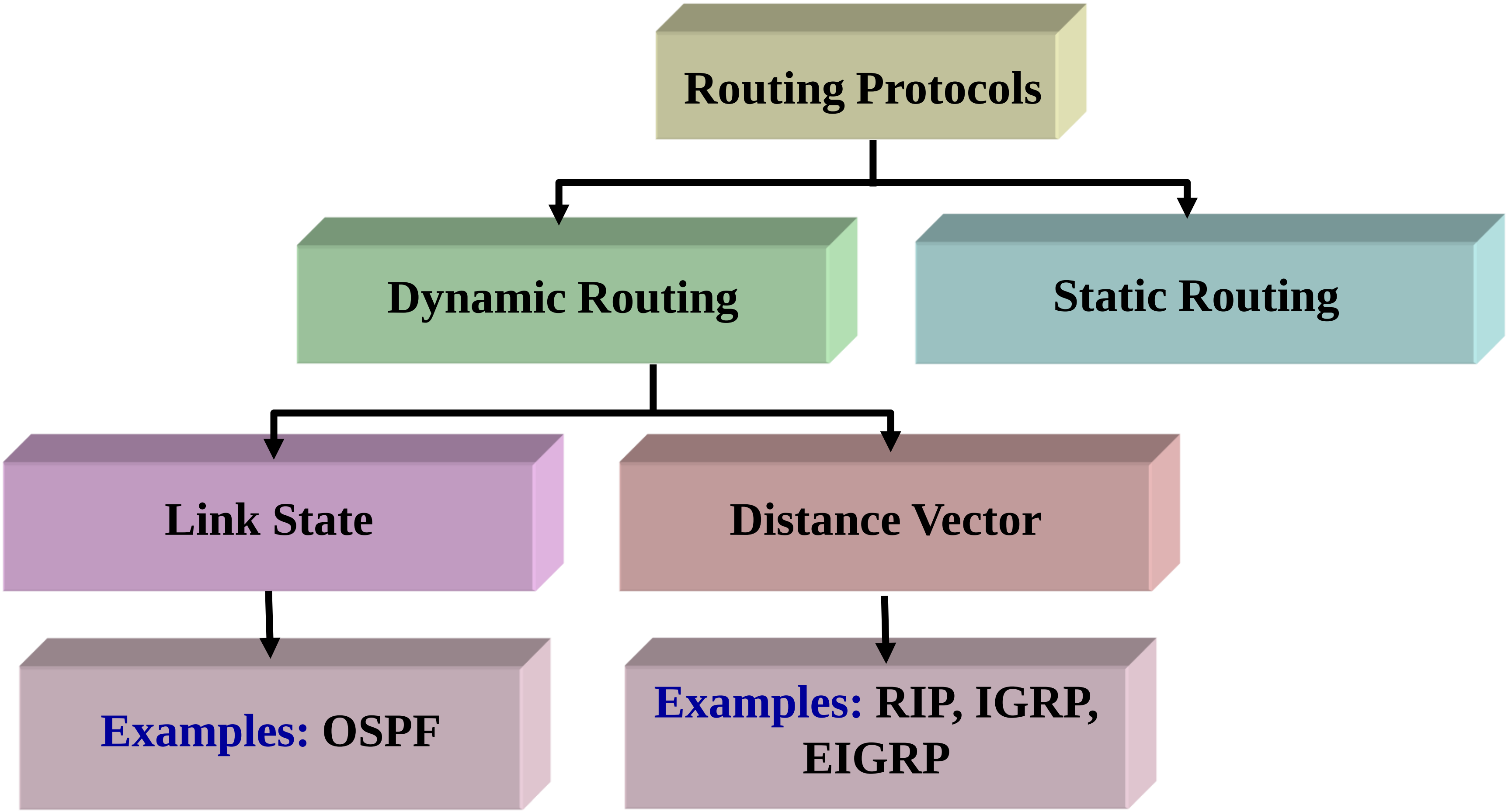
$$T = \frac{\lambda / \mu}{\lambda(1-\lambda / \mu)}$$

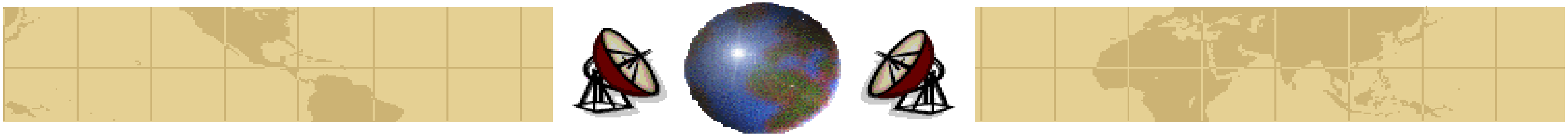
$$T = \frac{1/\mu}{1-\lambda / \mu}$$

$$T = \frac{1}{\mu-\lambda}$$

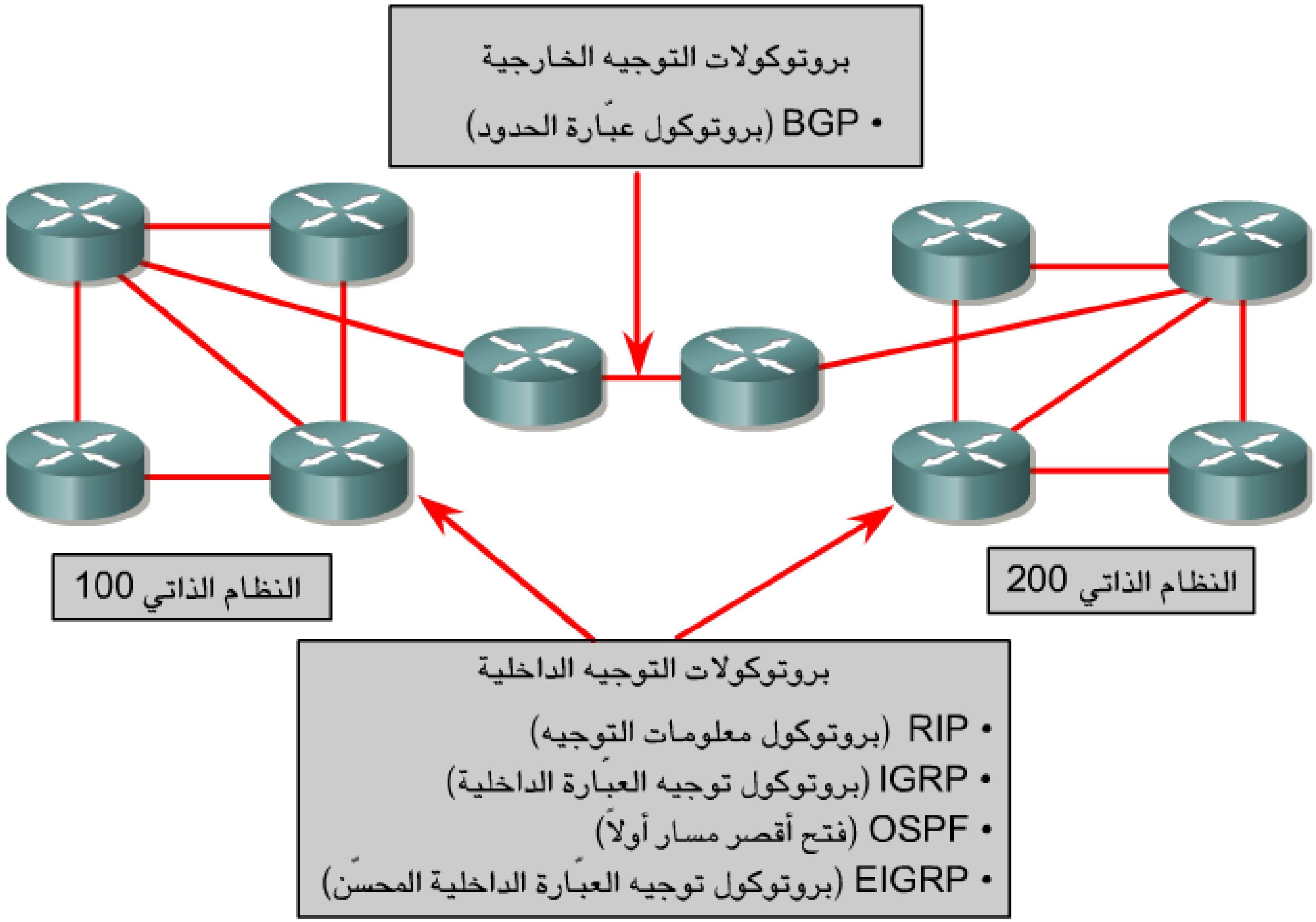
$$T = \frac{1}{\frac{C_i}{L}-\lambda}$$

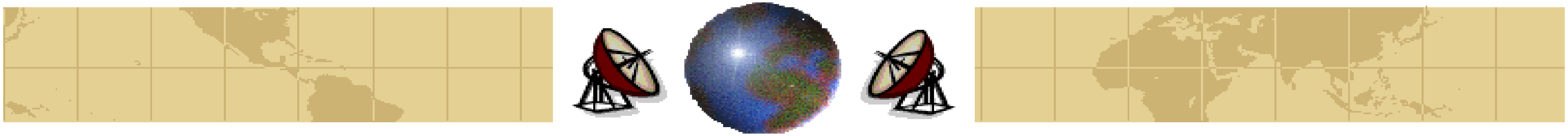
Routing Protocols





Routing Protocols





Routing Protocols

- **RIP (بروتوكول معلومات التوجيه)** – بروتوكول توجيه متجه مسافات داخلي
- **IGRP (بروتوكول توجيه العبّارة الداخلية)** – بروتوكول التوجيه الداخلي لمتجه المسافات خاص بشركة Cisco
- **OSPF (فتح أقصر مسار أولاً)** – بروتوكول توجيه داخلي لحالة الارتباط
- **EIGRP (بروتوكول توجيه العبّارة الداخلية المحسّن)** – بروتوكول التوجيه الداخلي المتقدم لمتجه المسافات الخاص بشركة Cisco
- **BGP (بروتوكول عبّارة الحدود)** – بروتوكول توجيه خارجي لمتجه المسافات

لقد تم تعيين بروتوكول RIP في الأصل في مواصفات RFC (طلب التعليقات) رقم 1058. وتتضمن خصائصه الأساسية ما يلي:

- إنه بروتوكول توجيه لمتجه المسافات.
- يتم استخدام تعداد الخطوات بمثابة القياس لتحديد المسار.
- إذا كان تعداد الخطوات أكبر من 15، يتم تجاهل الحزمة (packet).
- يتم بث تحديثات التوجيه كل 30 ثانية بشكل افتراضي.

يعد IGRP (بروتوكول توجيه العبّارة الداخلية) من البروتوكولات الخاصة التي تمتلكها شركة Cisco وقامت بتطويرها. وفيما يلي بعض خصائص التصميم الأساسية لبروتوكول IGRP:

- إنه بروتوكول توجيه لمتجه المسافات.
- يتم استخدام عرض النطاق الترددي، والتحميل، وفترة التأخير، وإمكانية الموثوقية لإنشاء قياس مركب.
- يتم بث تحديثات التوجيه كل 90 ثانية بشكل افتراضي.

يعد OSPF من بروتوكولات توجيه حالة الارتباط التي لا تملكها جهة معينة. فيما يلي الخصائص الأساسية الخاصة لبروتوكول OSPF:

- إنه بروتوكول توجيه لحالة الارتباط.
- إنه بروتوكول توجيه ذو معيار مفتوح موصوف في RFC رقم 2328.
- يتم استخدام خوارزمية SPF لحساب أدنى تكلفة لوجهة.
- يتم غمر تحديثات التوجيه وفقما تحدث تغييرات بالهيكل.

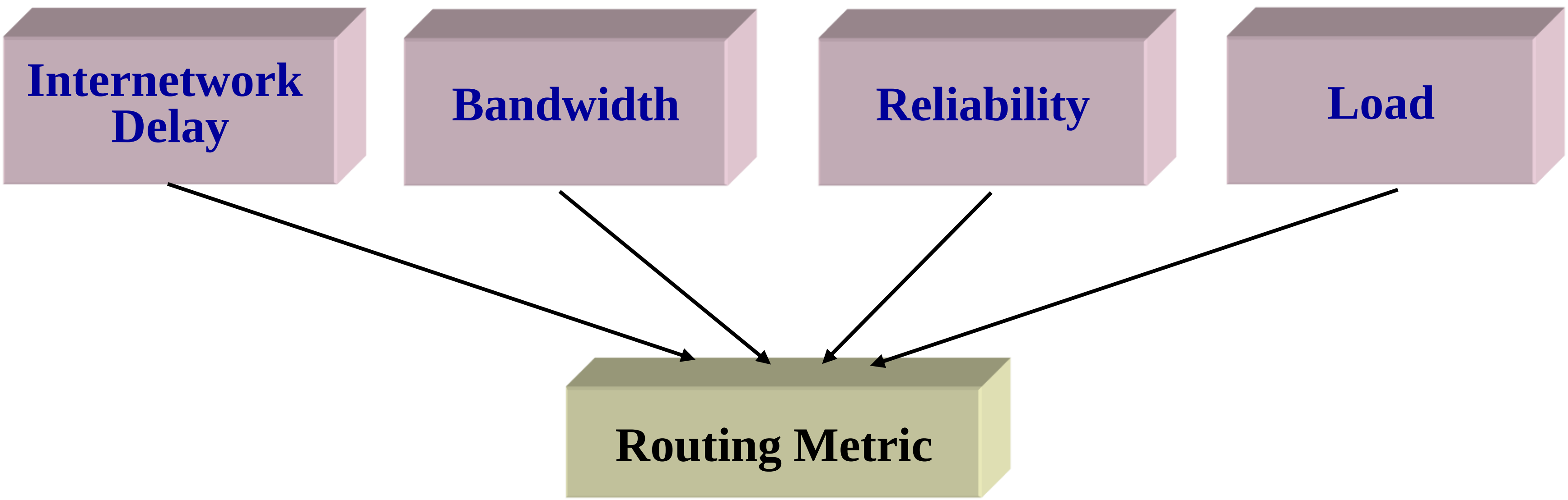
يعد EIGRP (بروتوكول توجيه العبّارة الداخلية المحسّن) بروتوكول توجيه متجه مسافات محسّنًا تملكه شركة Cisco. وفيما يلي الخصائص الأساسية لبروتوكول EIGRP:

- إنه بروتوكول توجيه محسّن لمتجه المسافات.
- يستخدم ضبط موازنة التحميل.
- يستخدم اتحادًا من ميزات متجه المسافات وحالة الارتباط.
- يستخدم خوارزمية تحديث النشر (DUAL) لحساب أقصر مسار.
- تعد تحديثات التوجيه من البث المتعدد ويستخدم 224.0.0.10 كل 30 ثانية أو تبعًا لما يتم تحفيزه من قبل تغييرات الهيكل.

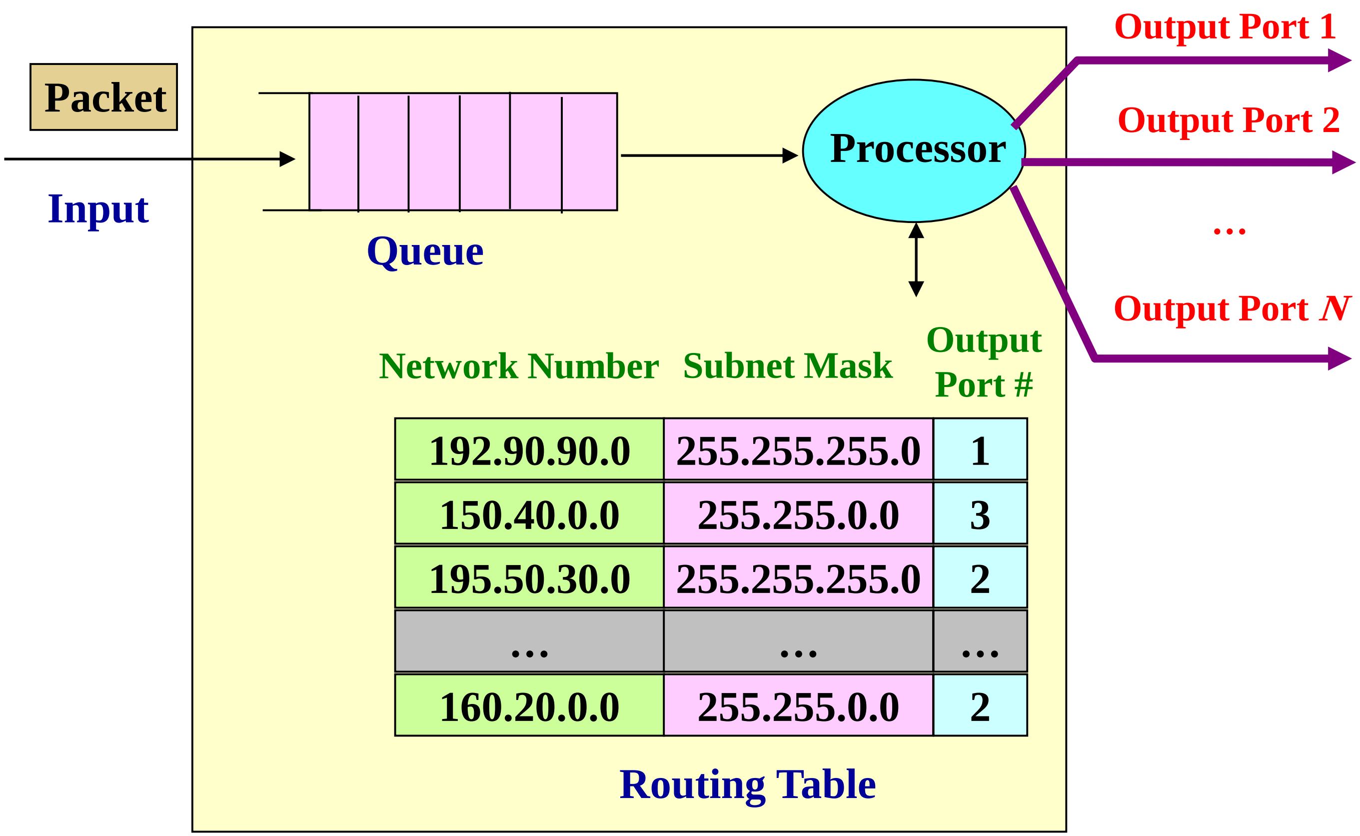
بروتوكول عبّارة الحدود (BGP) هو بروتوكول توجيه خارجي. فيما يلي الخصائص الأساسية لبروتوكول BGP:

- بروتوكول توجيه خارجي لمتجه المسافات.
- يتم استخدامه فيما بين موفري خدمات الإنترنت (ISPs) أو بين موفري خدمات الإنترنت والعملاء.
- يتم استخدامه لتوجيه حركة مرور الإنترنت بين الأنظمة الذاتية.

Routing Metric



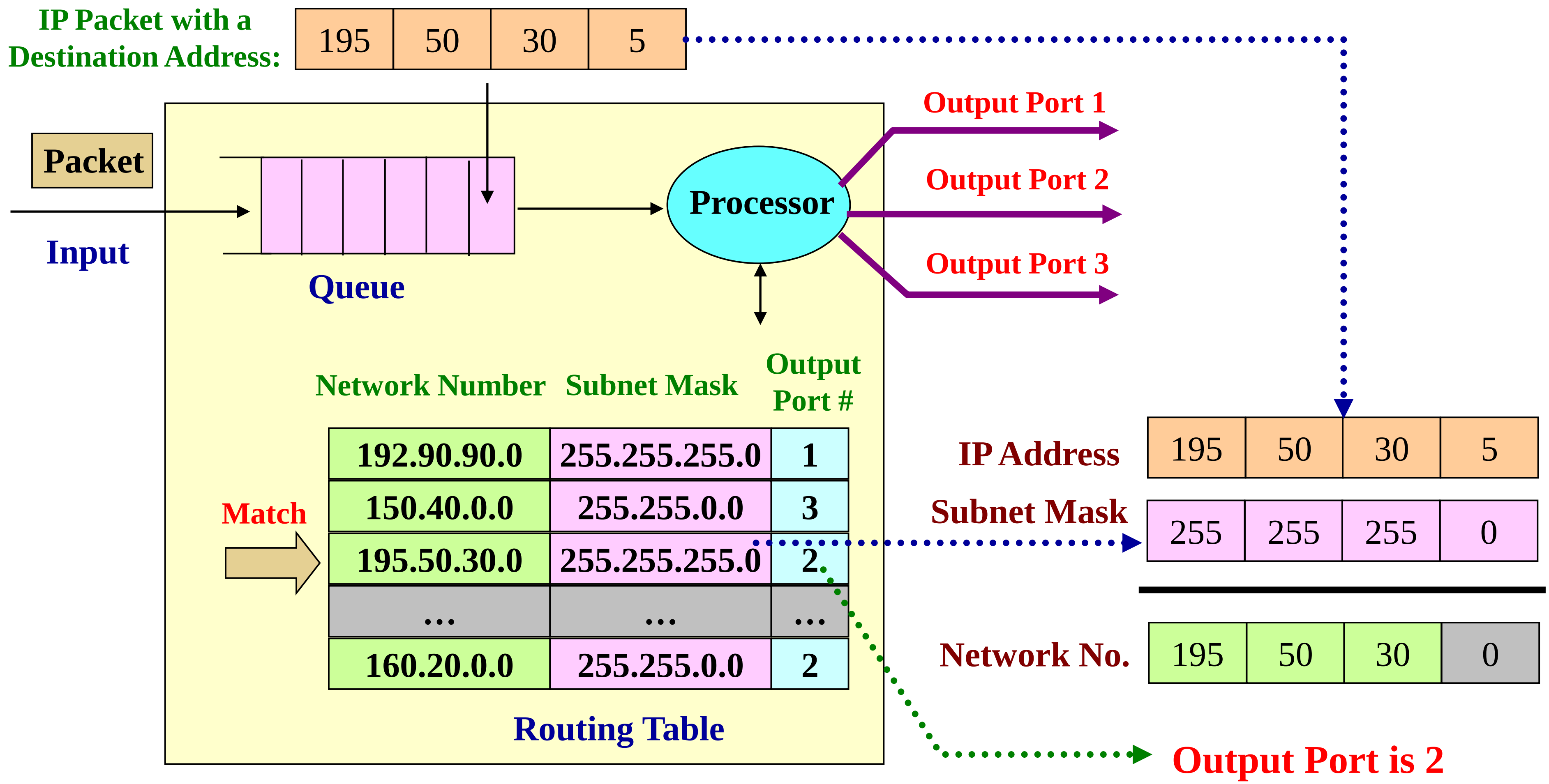
Routed Protocols



Network Number	Subnet Mask	Output Port #
192.90.90.0	255.255.255.0	1
150.40.0.0	255.255.0.0	3
195.50.30.0	255.255.255.0	2
…	…	…
160.20.0.0	255.255.0.0	2

Routing Table

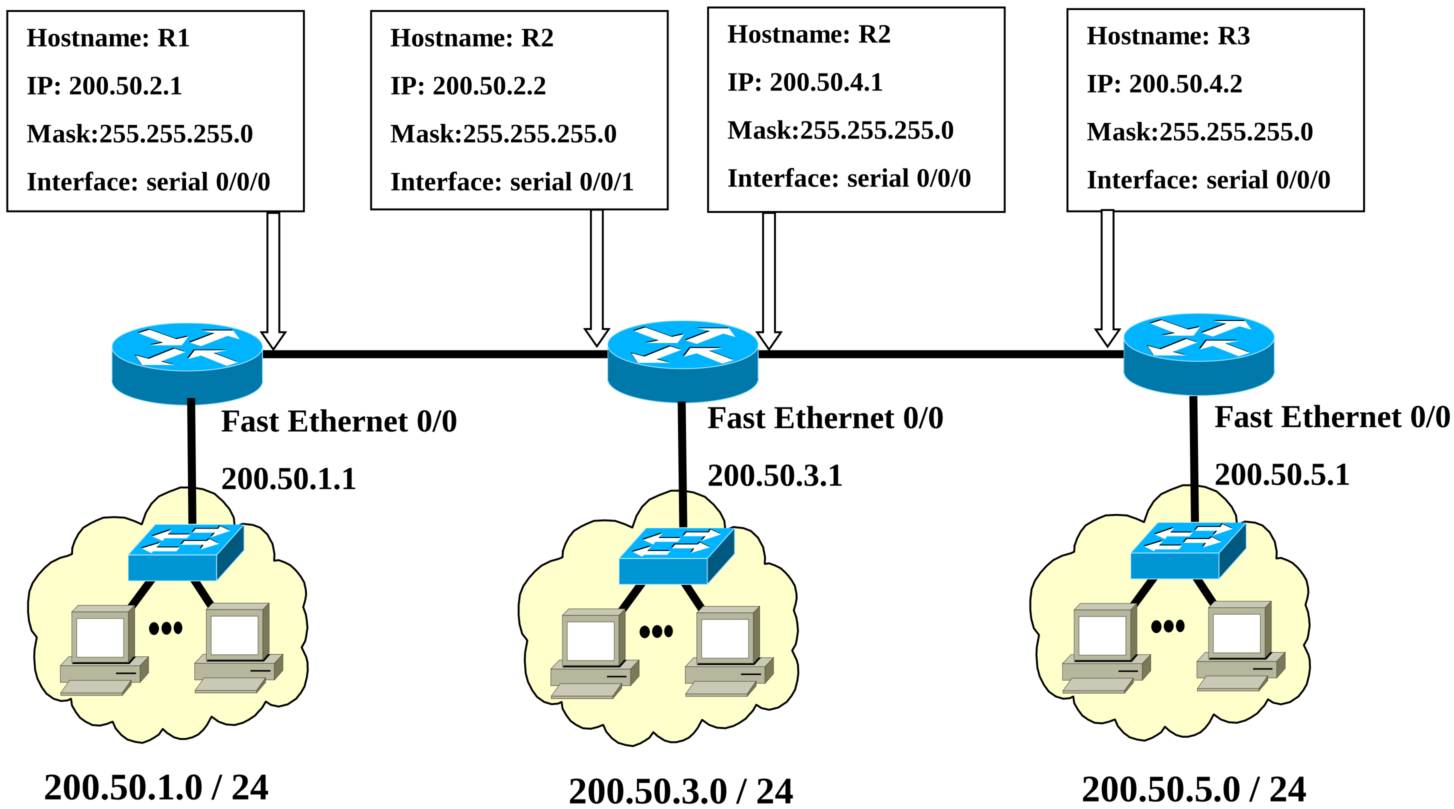
Routed Protocols



Network Number	Subnet Mask	Output Port #
192.90.90.0	255.255.255.0	1
150.40.0.0	255.255.0.0	3
195.50.30.0	255.255.255.0	2
…	…	…
160.20.0.0	255.255.0.0	2

Static Routing

Example 1:



Static Routing

R1’s Configuration:

```
R1# config t
R1(config)#  ip  route  0.0.0.0  0.0.0.0  serial 0/0/0
R1(config)#  exit
R1#  copy run start
```

R2’s Configuration:

```
R2# config t
R2(config)#  ip  route  200.50.1.0  255.255.255.0  serial 0/0/1
R2(config)#  ip  route  200.50.5.0  255.255.255.0  serial 0/0/0
R2(config)#  exit
R2#  copy run start
```

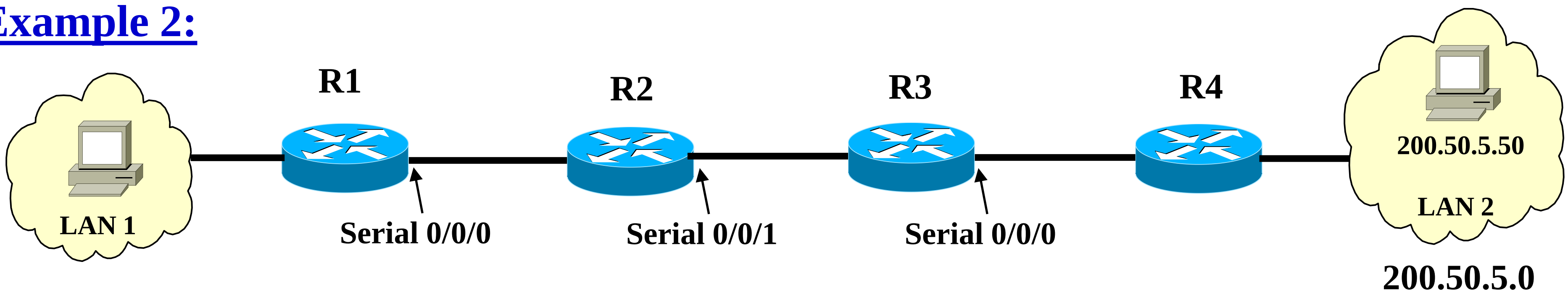
Static Routing

R3's Configuration:

```
R3# config t
R3(config)#  ip  route  0.0.0.0  0.0.0.0  serial 0/0/0
R3(config)#  exit
R3#  copy run start
```

Static Routing

Example 2:




```
R1(config)#  ip  route  200.50.5.0  255.255.255.0  serial 0/0/0
```

```
R2(config)#  ip  route  200.50.5.0  255.255.255.0  serial 0/0/1
```

```
R3(config)#  ip  route  200.50.5.0  255.255.255.0  serial 0/0/0
```

Dynamic Routing

1. Distance Vector Routing
2. Link State Routing

Dynamic Routing

الأمر
`Router(config)#router protocol {options}`

يعرّف بروتوكول توجيه

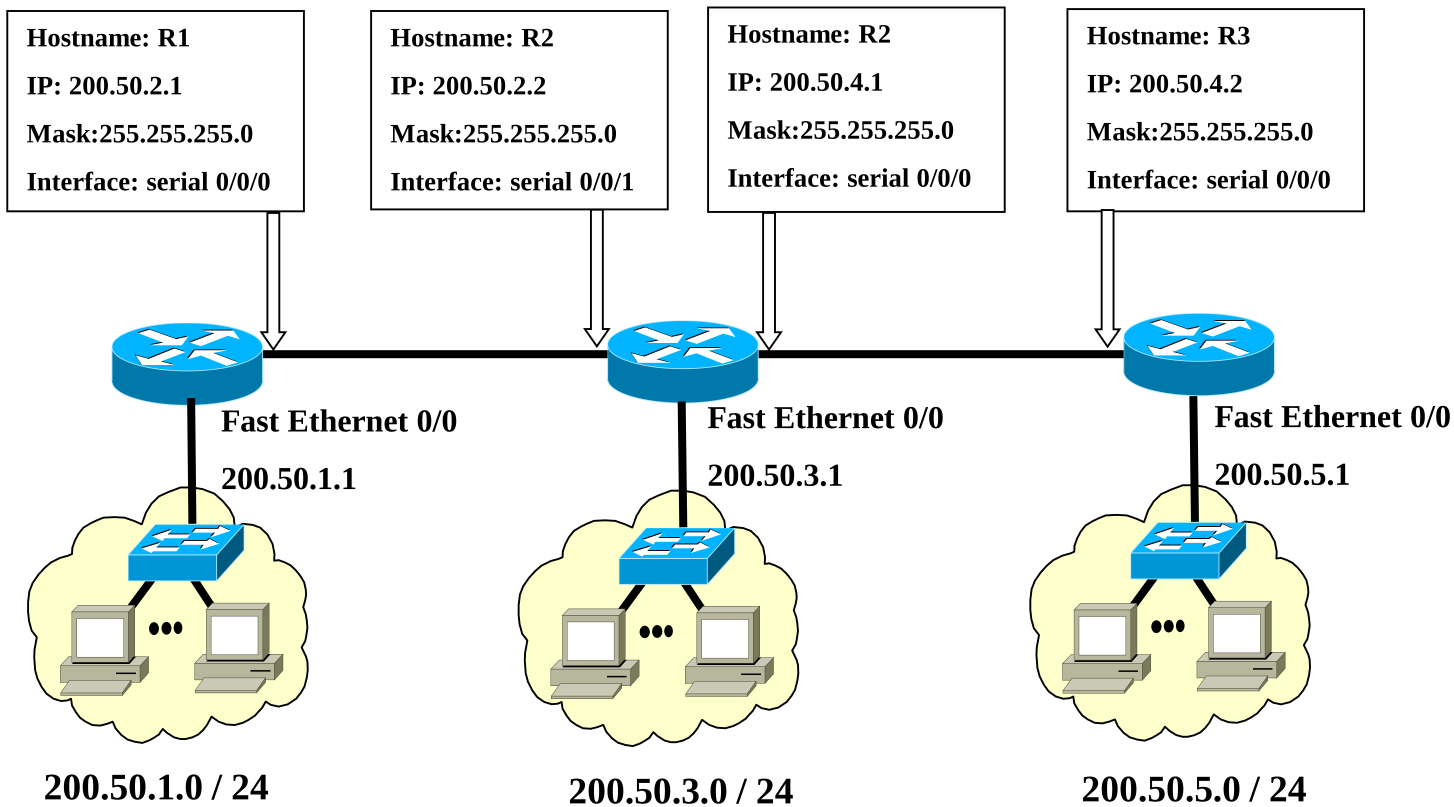
الأمر
`Router(config-router)#network network-number`

أمر الشبكة الفرعي هو أمر تكوين إلزامي لكل عملية توجيه

أمر `router`	الوصف
`protocol`	IGRP(بروتوكول توجيه العبّارة الداخلية)، أو EIGRP (بروتوكول توجيه العبّارة الداخلية المحسّن)، أو OSPF(فتح أقصر مسار أولاً)، أو RIP(بروتوكول معلومات التوجيه)
`options`	يتطلب كل من IGRP (بروتوكول توجيه العبّارة الداخلية) و EIGRP(بروتوكول توجيه العبّارة الداخلية الموسّع) رقماً ذاتياً. تتطلب OSPF (فتح أقصر مسار أولاً) معرّف عملية. بينما لا تتطلب RIP(بروتوكول معلومات التوجيه) ذلك.

Dynamic Routing

Example:



Dynamic Routing

R1's Configuration:

```
R1# config t
R1(config)#  router rip
R1(config-router)#  network  200.50.1.0
R1(config-router)# network  200.50.2.0
R1(config-router)# ^Z
R1#  copy run start
```

R2's Configuration:

```
R2# config t
R2(config)#  router rip
R2(config-router)#  network  200.50.2.0
R2(config-router)# network  200.50.3.0
R2(config-router)# network  200.50.4.0
R2(config-router)# ^Z
R2#  copy run start
```

Dynamic Routing

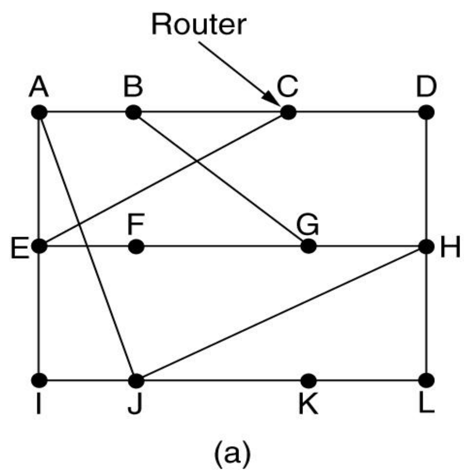
R3's Configuration:

```
R3# config t
R3(config)#  router rip
R3(config-router)#  network  200.50.4.0
R3(config-router)# network  200.50.5.0
R3(config-router)# ^Z
R3#  copy run start
```

Distance Vector Routing (Distributed Bellman-Ford)

- Distance vector routing is a dynamic algorithm.
- The basic idea is that each router maintains a table giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors.
- In distance vector routing, each router maintains a **routing table**, which contains one entry for each router in the network. The entry contains two parts: the preferred outgoing line for a certain destination, and an estimate of time and distance for that destination.
- Once every *T* msec, each router sends to each neighbor a list of its estimated delays to each destination. It also receives similar list from each neighbor.

Distance Vector Routing



(a) A subnet

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

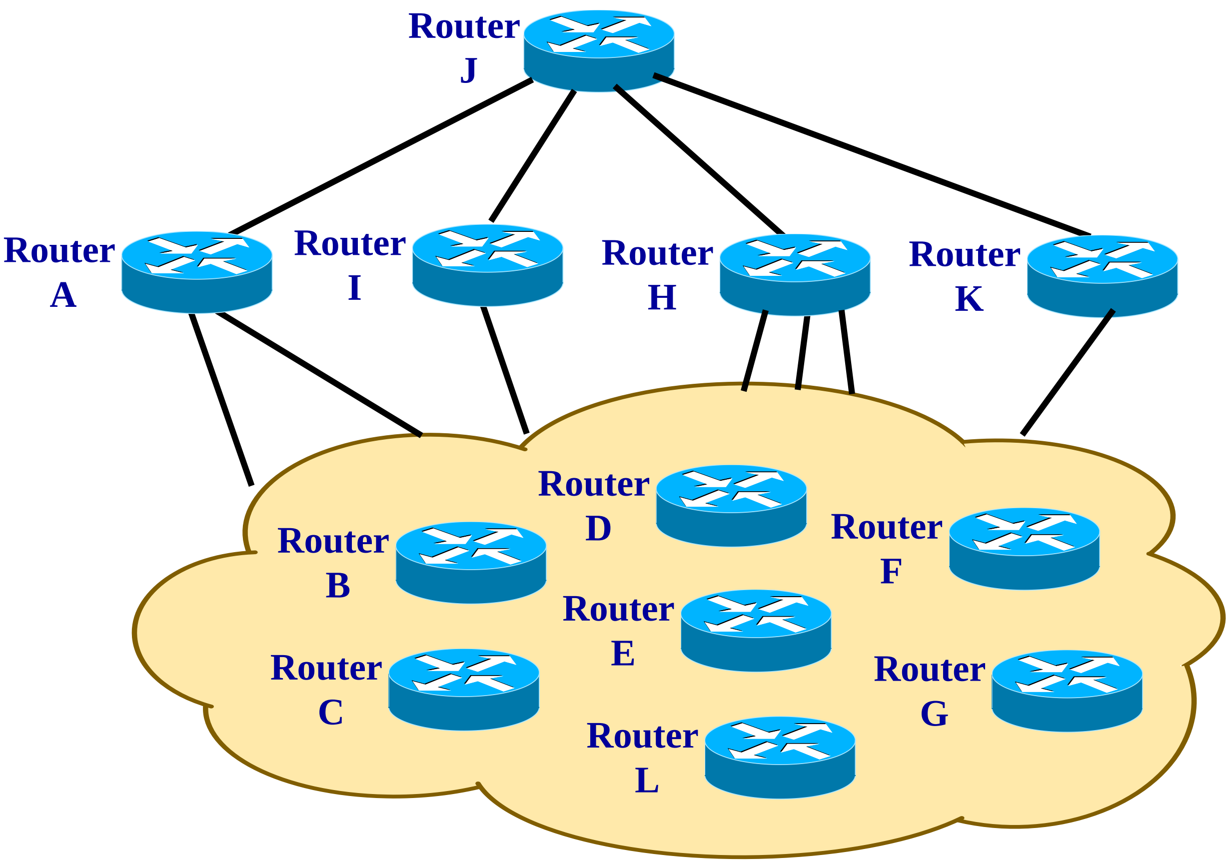
(b)

(b) Input from A, I, H, K,
and the new routing table for J

The router J computes its new route to router G as follows:

$$D_{jG} = \min[D_{JA}+D_{AG}, D_{JI}+D_{IG}, D_{JH}+D_{HG}, D_{JK}+D_{KG}]$$
$$= \min[8+18, 10+31, 12+6, 6+31] = 18 \text{ msec.}$$

Distance Vector Routing



A	0
B	12
C	25
D	40
E	14
F	23
G	18
H	17
I	21
J	9
K	24
L	29

Routing Table A

A	24
B	36
C	18
D	27
E	7
F	20
G	31
H	20
I	0
J	11
K	22
L	33

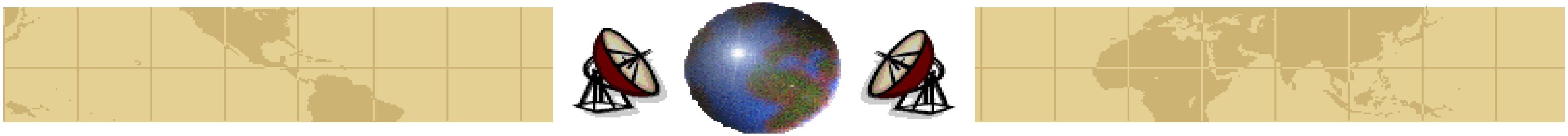
Routing Table I

A	20
B	31
C	19
D	8
E	30
F	19
G	6
H	0
I	14
J	7
K	22
L	9

Routing Table H

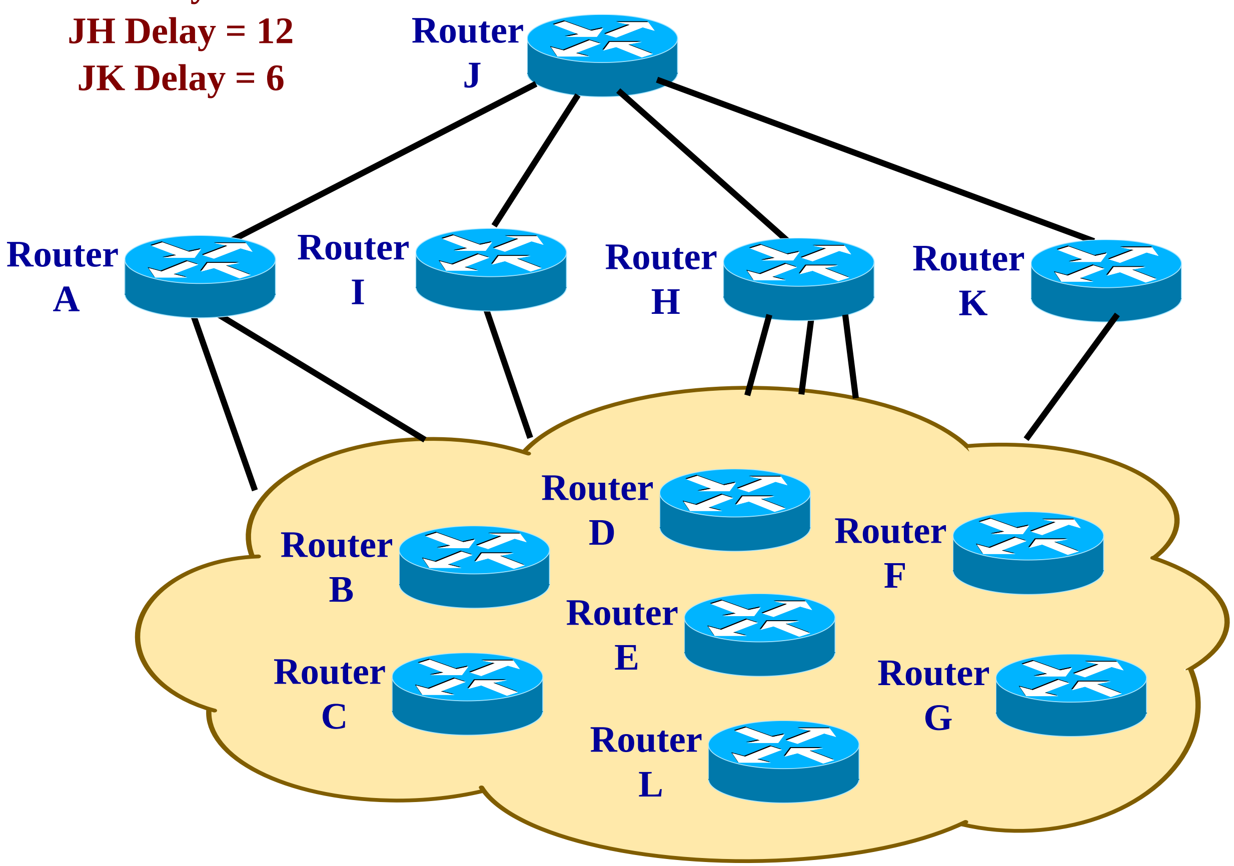
A	21
B	28
C	36
D	24
E	22
F	40
G	31
H	19
I	22
J	10
K	0
L	9

Routing Table K



Distance Vector Routing

JA Delay = 8
JI Delay = 10
JH Delay = 12
JK Delay = 6



A	0
B	12
C	25
D	40
E	14
F	23
G	18
H	17
I	21
J	9
K	24
L	29

Routing Table A

A	24
B	36
C	18
D	27
E	7
F	20
G	31
H	20
I	0
J	11
K	22
L	33

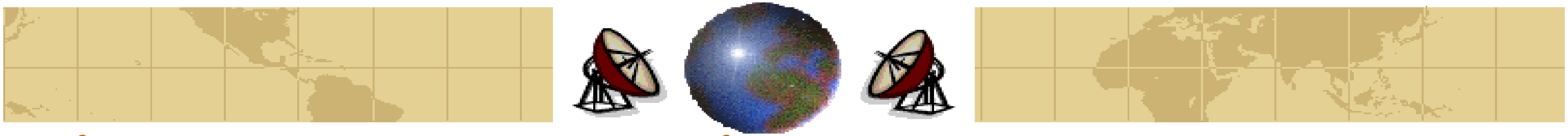
Routing Table I

A	20
B	31
C	19
D	8
E	30
F	19
G	6
H	0
I	14
J	7
K	22
L	9

Routing Table H

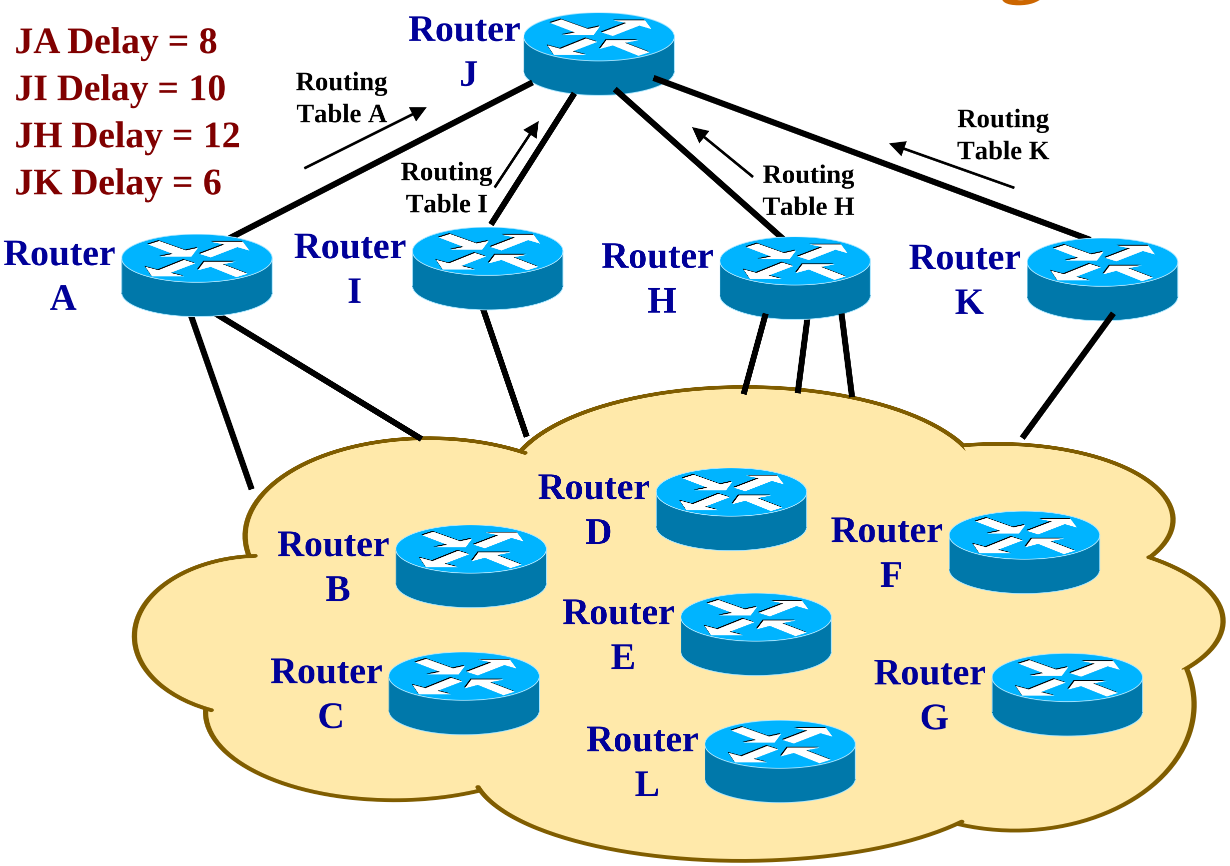
A	21
B	28
C	36
D	24
E	22
F	40
G	31
H	19
I	22
J	10
K	0
L	9

Routing Table K



Distance Vector Routing

JA Delay = 8
JI Delay = 10
JH Delay = 12
JK Delay = 6



A	0
B	12
C	25
D	40
E	14
F	23
G	18
H	17
I	21
J	9
K	24
L	29

Routing Table A

A	24
B	36
C	18
D	27
E	7
F	20
G	31
H	20
I	0
J	11
K	22
L	33

Routing Table I

A	20
B	31
C	19
D	8
E	30
F	19
G	6
H	0
I	14
J	7
K	22
L	9

Routing Table H

A	21
B	28
C	36
D	24
E	22
F	40
G	31
H	19
I	22
J	10
K	0
L	9

Routing Table K

Example: Calculation the cost from J to G

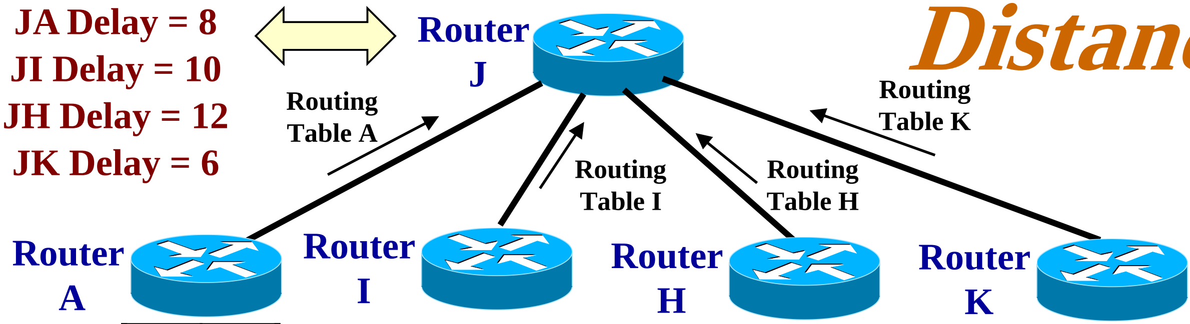
Cost = Minimum(JA+AG , JI+IG , JH+HG , JK+KG)

Cost = Minimum(8+18 , 10+31 , 12+6 , 6+31)

Cost = Minimum(26 , 41 , 18 , 37) = 18 via H

Distance Vector Routing

JA Delay = 8
JI Delay = 10
JH Delay = 12
JK Delay = 6



A	0
B	12
C	25
D	40
E	14
F	23
G	18
H	17
I	21
J	9
K	24
L	29

Routing Table A

A	24
B	36
C	18
D	27
E	7
F	20
G	31
H	20
I	0
J	11
K	22
L	33

Routing Table I

A	20
B	31
C	19
D	8
E	30
F	19
G	6
H	0
I	14
J	7
K	22
L	9

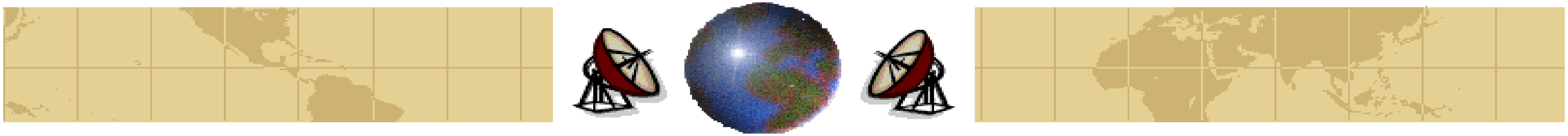
Routing Table H

A	21
B	28
C	36
D	24
E	22
F	40
G	31
H	19
I	22
J	10
K	0
L	9

Routing Table K

Destin-ation	New Cost	Through Neighbor
A	8	A
B	20	A
C	28	I
D	20	H
E	17	I
F	30	I
G	18	H
H	12	H
I	10	I
J	0	-
K	6	K
L	15	K

New Routing Table for J



Count-to-Infinity Problem

- Distance vector routing has a serious drawback in practice: although it converges to the correct answer, it may do so slowly. In particular, it reacts 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)

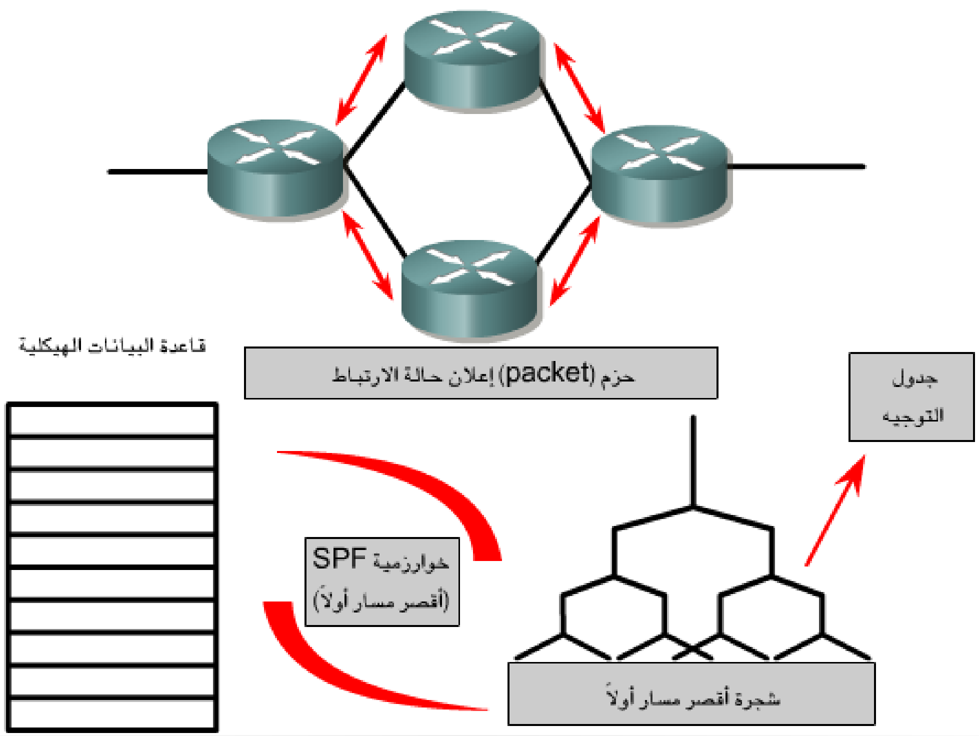
(a) Good News

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)

(a) Bad News

Link State Routing



تقوم أجهزة التوجيه (router) بإرسال بإعلانات LSA (إعلانات حالة الارتباط) إلى الأجهزة المجاورة لها. يتم استخدام إعلانات LSA (إعلانات حالة الارتباط) لبناء قاعدة بيانات هيكلية. ويتم استخدام خوارزمية SPF (أقصر مسار أولا) لحساب شجرة أقصر مسار أولاً التي يكون فيها الجذر جهاز التوجيه (router) الفردي. ثم يتم بعد ذلك إنشاء جدول توجيه.

Link State Routing

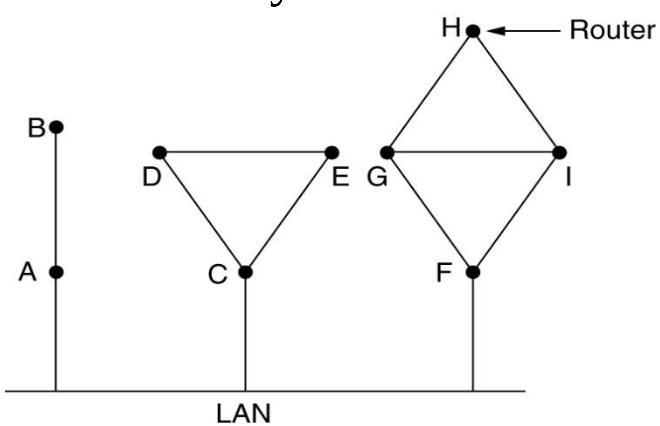
- Distance vector routing was used in the ARPANET until 1979, when it is replaced by **link state routing**.

- Two primary problems of distance vector routing:
  1- Since the delay metric was queue length, it did not take line bandwidth into accounts when choosing routes.
  2- The algorithm often took too long to converge.

- The basic idea is simple and can be stated as *five* parts:
  - Discover its neighbors and learn their network addresses.
  - Measure the delays or cost to each of its neighbors.
  - Construct a packet telling all it has learned.
  - Send this packet to all other routers.
  - Compute the shortest path to every other router.

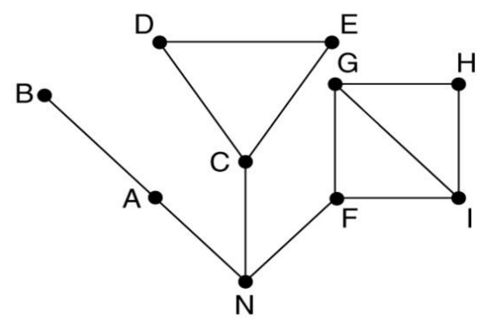
Learning about the Neighbors

- When a router is booted, its first task to *learn who its neighbors are*.
- It accomplishes this goal by sending a special **HELLO** packet on each point-to-point. The router at the other end is expected to send back a reply telling who it is (using a globally unique address).
- When two or more router are connected by a LAN, the situation is slightly more complicated. One way to model the LAN is to consider it as a node itself.



(a)

(a) Nine routers and a LAN.



(b)

(b) A graph model of (a).

Measuring Line Cost

- The link state routing algorithm requires each router to know an estimate of the delay to each of its neighbors.

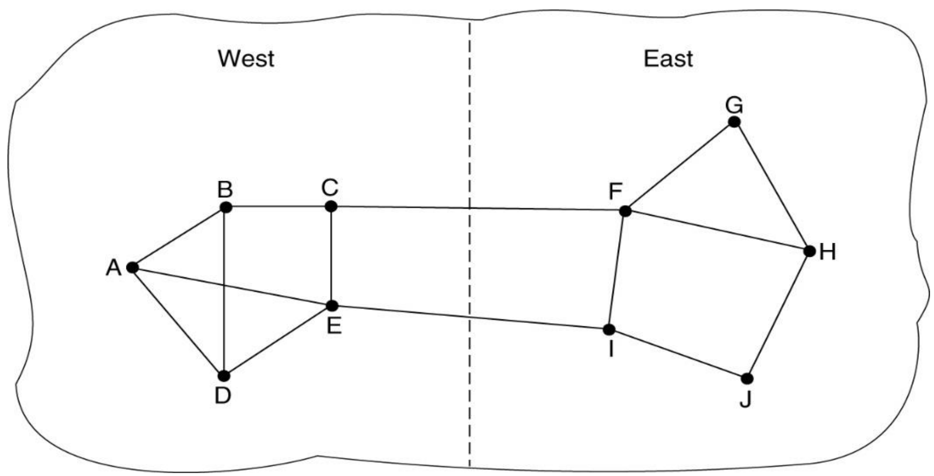
- It sends a special **ECHO** packet over the line that the other side is required to send it back immediately. *By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of delay*. For better results, the test can be conducted several times, and the average is used.

- An interesting issue is whether or not to consider the load when measuring the delay. To factor the load in, the timer must be started when the **ECHO** packet is queued. To ignore the load, the timer should be started when the **ECHO** packet reaches the front of the queue.

Measuring Line Cost

Argument against Including the Load in the delay Calculation

- Consider the given subnet, which is divided into two parts, East and West, connected by two lines, *CF* and *EI*. Suppose the most of the traffic between East and West is using line *CF*. Including the queueing delay in the shortest path calculation will make *EI* more attractive. Then, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate widely.



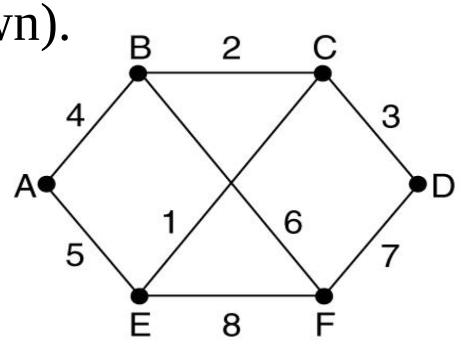
A subnet in which the East and West parts are connected by two lines.

Building Link State Packets

- Each router then builds a packet containing the following data:
  - Identity of the sender.
  - Sequence number.
  - Age.
  - A list of neighbors and their delays from the sender.

When to build the link state packets?

- It can be either periodically or when some significant event occurs (line goes down).



(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)

(a) A subnet. (b) The link state packets for this subnet.

Distributing the Link State Packets

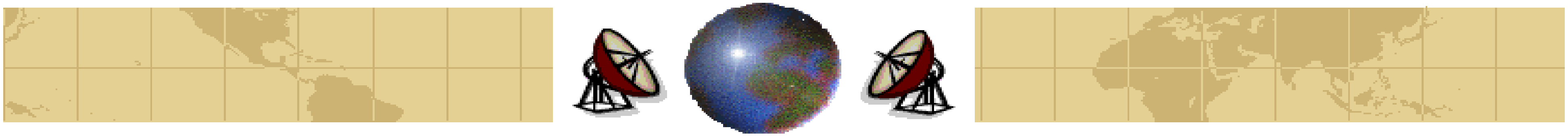
- The fundamental idea is to use **flooding** to distribute the link state packets.
- To manage the flooding operation, each packet contains a *sequence number* that is incremented for each new packet sent. Routers keep track of all the (source router, sequence) pairs they see.
- When a new link state packet comes in, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on.
- If it is a duplicate, It is discarded.
- If a packet with a sequence number lower than the highest one seen, it is rejected as being obsolete.

Distributing the Link State Packets

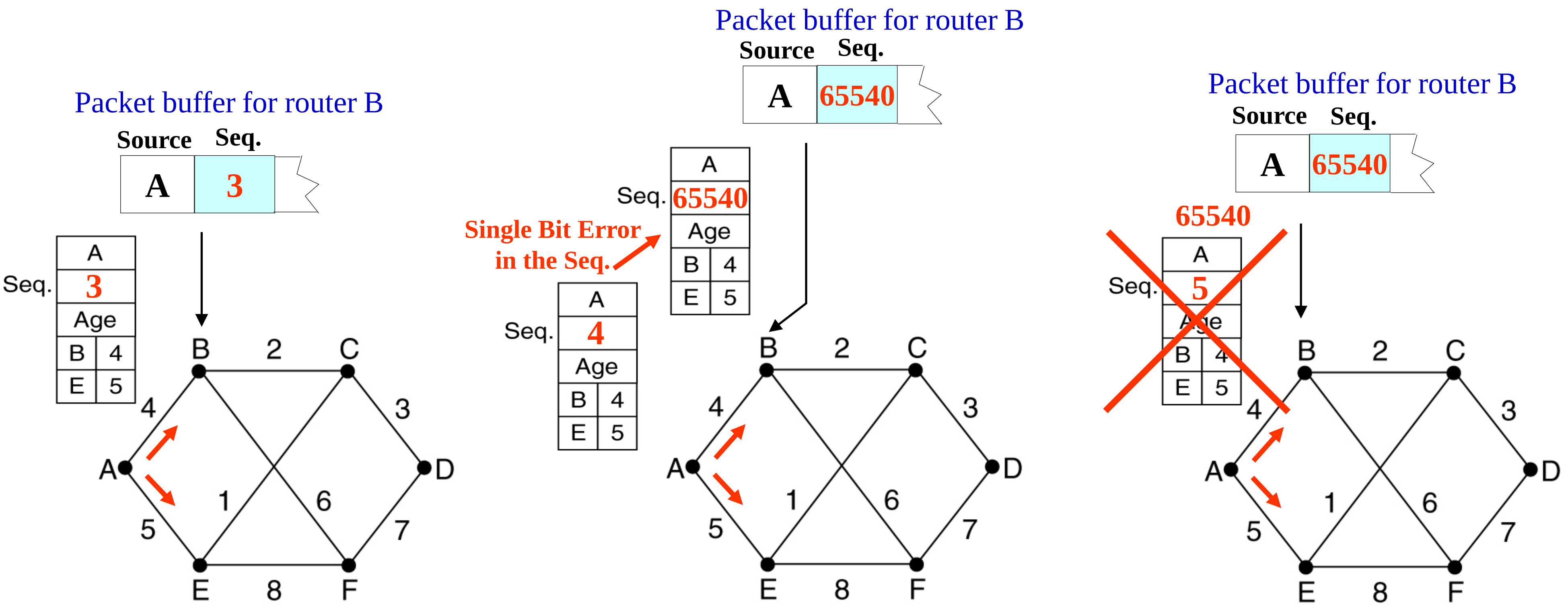
- The algorithm has a few problems:
  - The sequence numbers wrap around. The solution is to use a 32-bit sequence number.
  - If a router crashes, it will lose track of its sequence number.
  - Errors in the sequence numbers.

- The solution of these problems is to the **age** for each packet, and decrement it once per second. When the age hits zero, the information from the router is discarded.

- Some refinements to this algorithm make it more robust. When a link state packet comes in to a router for flooding, it is queued for a short while first. If another link state packet from the same source comes in before it is transmitted, their sequence numbers are compared. If they are equal, the duplicate is discarded. If they are different, the older one is thrown out.

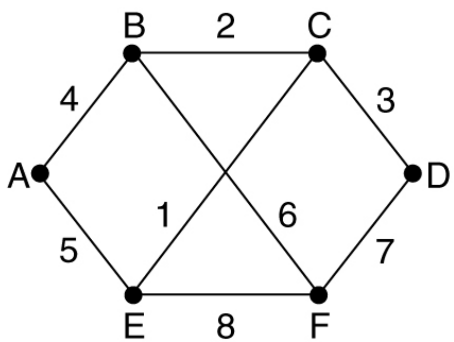


Distributing the Link State Packets



- If a sequence number is corrupted, and 65540 is received instead of 4 (a 1-bit error), packets 5 through 65540 will be rejected as obsolete, since the current sequence number is thought to be 65540.
- The *solution* is to include the **age** of each Link State Packet after the sequence number and decrement it once per second. When the age hits zero, the information from the router is discarded.

Distributing the Link State Packets



Source	Seq.	Age	Send flags A	Send flags C	Send flags F	ACK flags A	ACK flags C	ACK flags F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

The packet buffer for router B

Computing the New Routes

- Once a router has accumulated a full set of link state packets, it can construct the entire subnet graph because every link is represented.

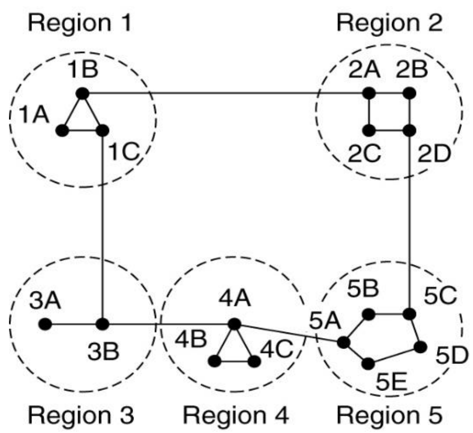
- Next Dijkstra’s algorithm can be run locally to construct the shortest path to all possible destinations.

Hierarchical Routing

- As networks grow in size, the router tables grow proportionally. This causes the following:
  - Router memory is consumed by increasing tables.
  - More CPU time is needed to scan the router tables.
  - More bandwidth is needed to send status reports about them.
- The basic idea of **hierarchical routing** is that routers are divided into **regions**, with each router knowing all the details about how to route packets to destination within its own region, but knowing nothing about the internal structure of other regions.
- For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the *regions* into *clusters*, the clusters into *zones*, the zones into *groups*, and so on.

Hierarchical Routing

- Routing in a two-level hierarchy with five regions.
- When routing is done hierarchically, each router table contains entries for all local routers and a single entry for each region.
- **Drawback:** *The shortest path might not be chosen* (Example Router 5C).



(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)

Broadcast Routing

- Sending a packet to all destinations simultaneously is called **broadcasting**.
- It can be implemented by various methods such as:

1. A source sends a distinct packet to each destination.

- It wastes the bandwidth, and requires the source to have a complete list of all destinations.

2. Flooding.

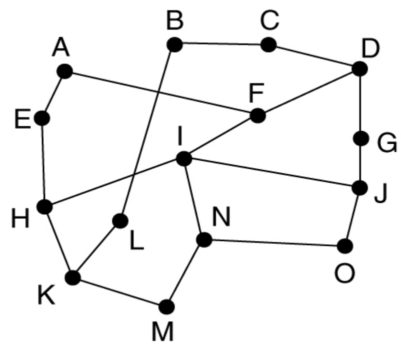
- It is ill-suited for point-to-point communication: too many packets and consumes too much bandwidth.

3. Multidimensional Routing:

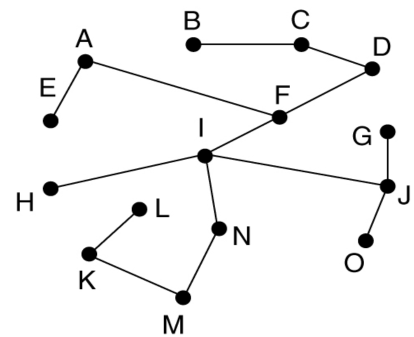
- Each packet contains a bit map indicating the desired destinations.
- When a packet arrives at a router, the router checks all the destination to determine the set of output lines that will be needed. (An output line is needed if it is the best route to at least one of the destinations.)

Broadcast Routing

4. Explicit use of the sink tree for the router initiating the broadcasting.



A subnet

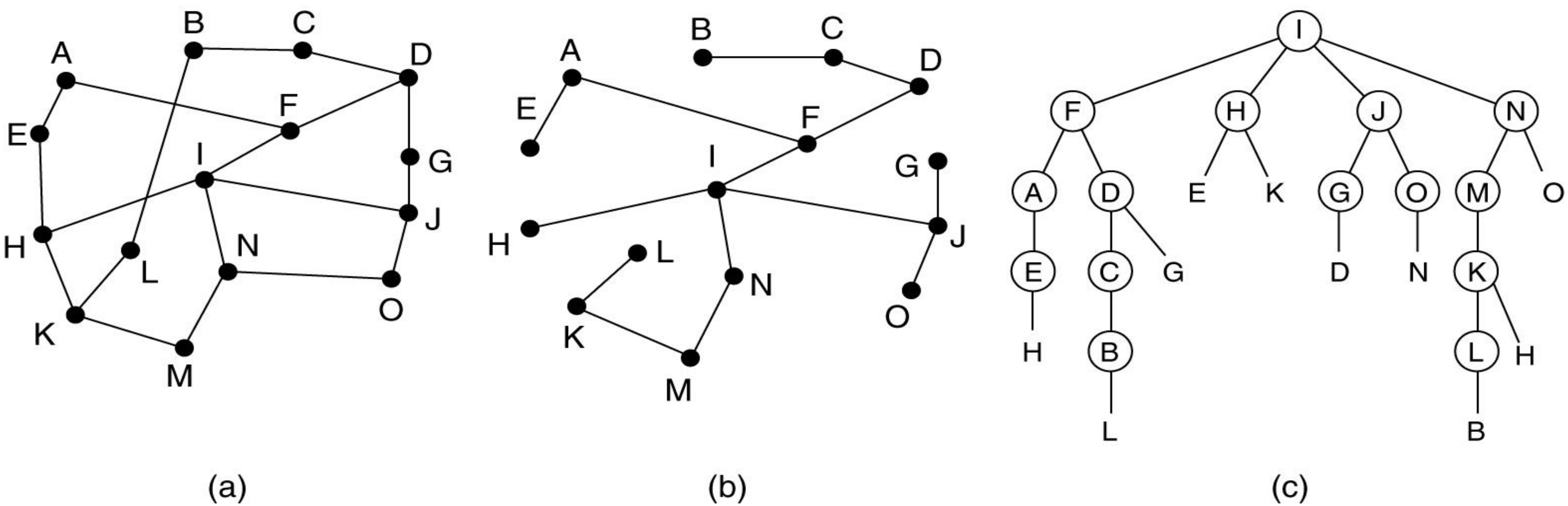


A Sink Tree

5. Reverse path forwarding:

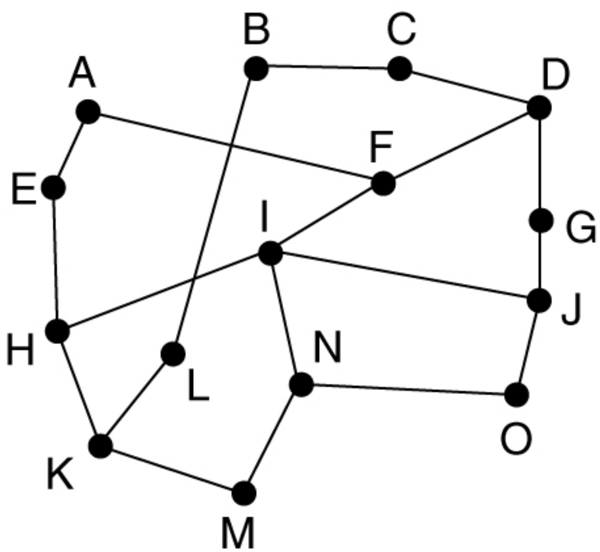
- When a broadcast packet arrives at a router, it checks to see if the packet arrived on the line that is normally used for sending packets to the source of the broadcast.
- If so, the router forwards copies of it to all outgoing lines except the one it arrived on.
- If the broadcast packet arrived on a line other that the preferred one for reaching the source of the broadcast, the packet is discarded.

Broadcast Routing

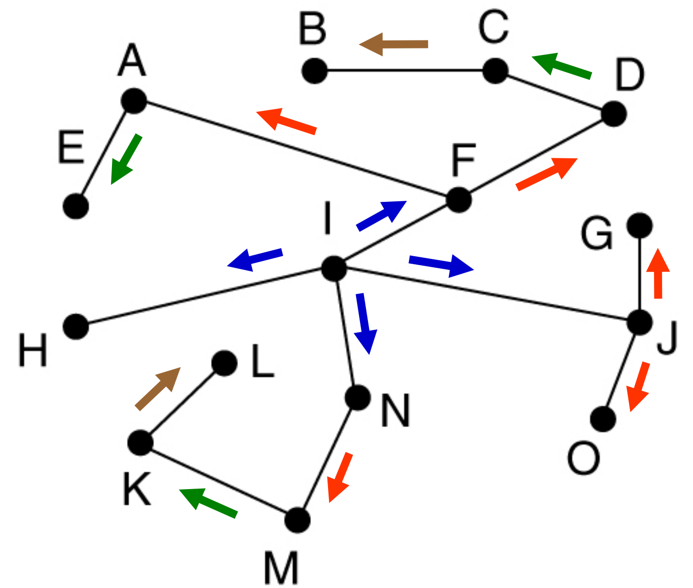


Reverse path forwarding. (a) A subnet. (b) a Sink tree. (c) The tree built by reverse path forwarding.

Broadcast Routing



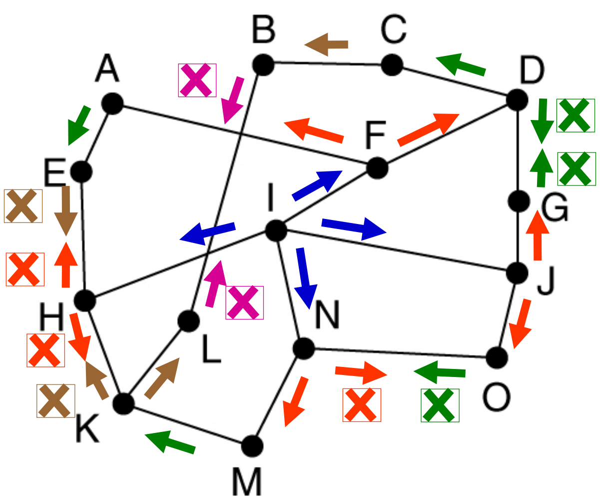
A subnet



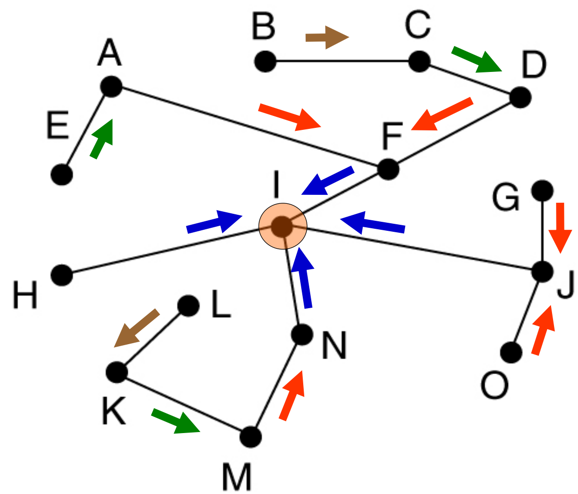
A Sink Tree

Number of copies of the broadcasting packet	14 copies
Maximum number of hops	4 hops

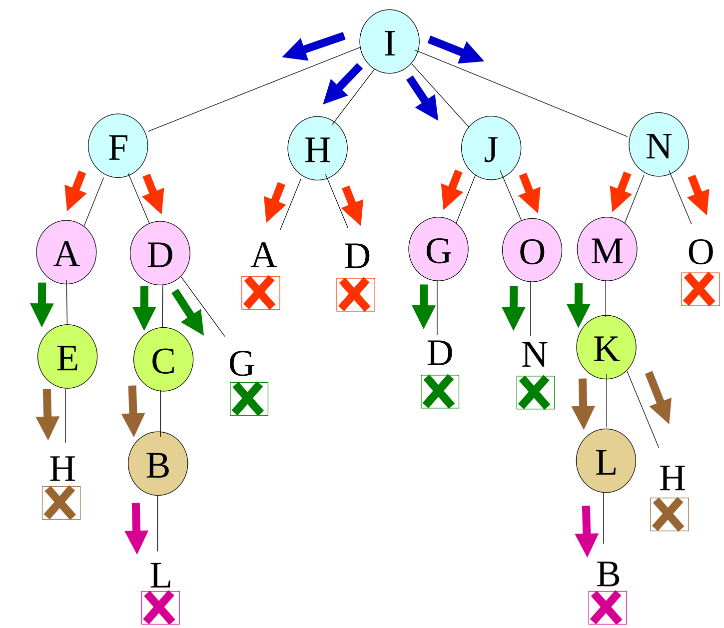
Broadcast Routing



A Subnet



A Sink Tree

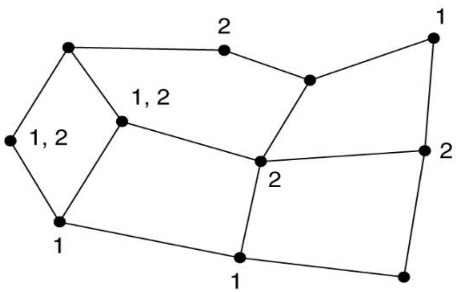


The tree built by reverse path forwarding.

Number of copies of the broadcasting packet	24 copies
Maximum number of hops	5 hops

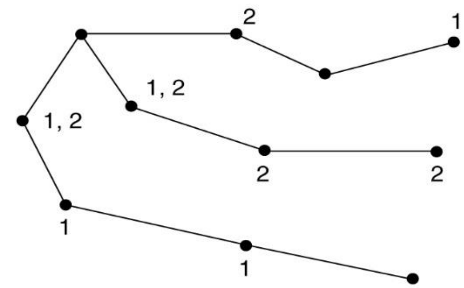
Multicast Routing

- Sending a packet to group (subset of all destinations) is called **Multicasting**.
- To do multicasting, *group management is required*. Some way is needed to *create* and *destroy groups*, and for processes to *join and leave groups*.



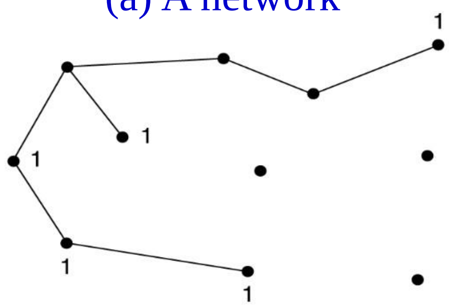
(a)

(a) A network



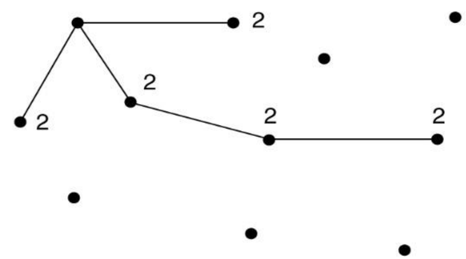
(b)

(b) A spanning tree for the leftmost router.



(c)

(c) A multicast tree for group 1.

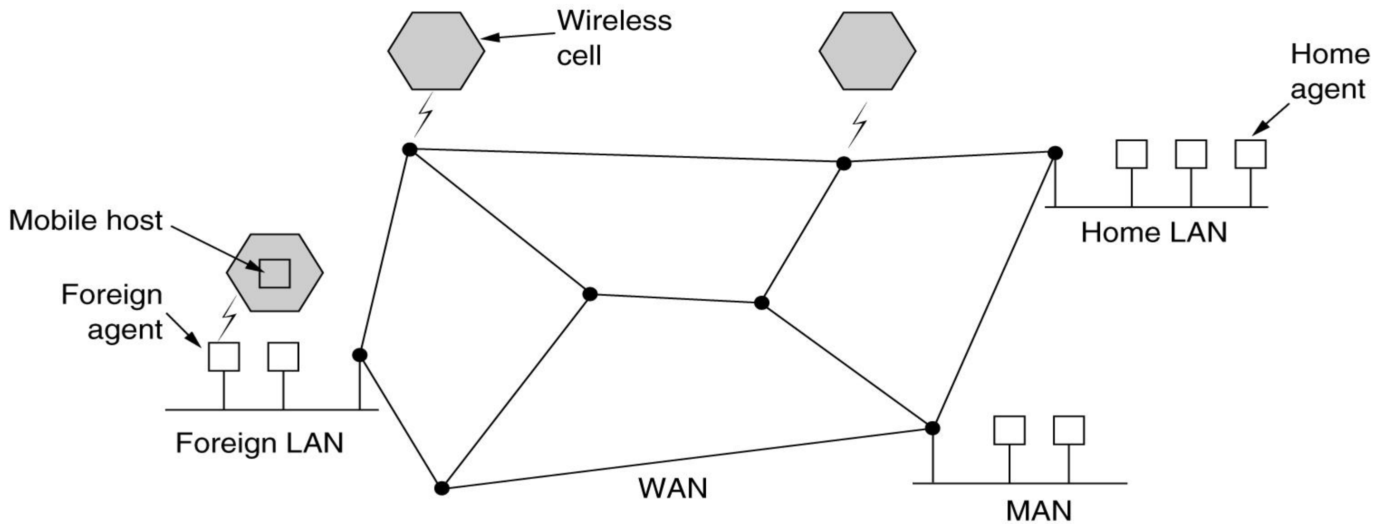


(d)

(d) A multicast tree for group 2.

Routing for Mobile Hosts

- People, who have portable computers, want to read their email and access their regular file systems wherever in the world they may be. These are called **Mobile Hosts**.



A WAN to which LANs, MANs, and wireless cells are attached.

Routing for Mobile Hosts

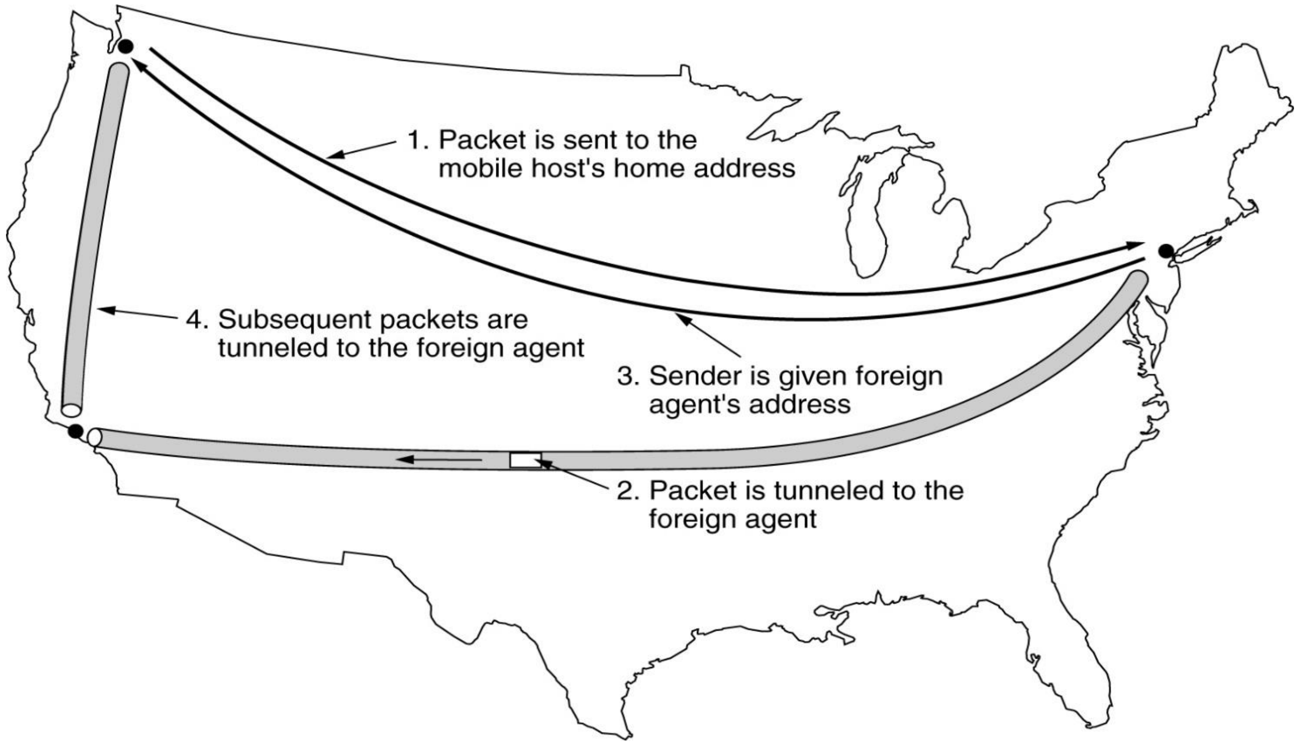
- To route a packet to a mobile host, the network fist has to find it.
- All users are assumed to have a permanent **home location**. Also, Users have a permanent **home address** that can be used to determine their home location.
- The *mobile routing goal* is to send packets to mobile users using their home addresses, and have the packet efficiently reach them wherever they may be.
- The world is divided up (geographically) into small units (typically, a LAN or wireless cell). Each area has one or more **foreign agents**, which keep track of all mobile users visiting the area. Also, each area has a **home agent**, which keep track of users whose home is in this area, but who are currently visiting another area.

Routing for Mobile Hosts

- When a user enters an area, his computer must register itself with the foreign agent there according to the following typical **registration procedure**:
  1. Periodically, each foreign agent broadcasts a packet announcing its existence and address. A newly arrived mobile host may wait for one of these messages, but if none arrives quickly enough, the mobile host can broadcast a packet saying: “Are there any foreign agents around?”
  2. The mobile host registers with the foreign agent, giving its home address and other information.
  3. The foreign host contacts the mobile host’s home agent and says: “One of your hosts is over here.” The message contains the network address of the foreign agent and some security information.
  4. The home agent examines the security information, which contains a timestamp. If it is approved, it informs the foreign agent to proceed.
  5. When the foreign agent gets the acknowledgement from the home agent, it makes an entry in its table and informs the mobile host that it is registered.

Routing for Mobile Hosts

- When a user leaves an area, it should cancel his registration.
- After registration, what does happen when a packet is sent to a mobile user?



Packet routing for mobile users.