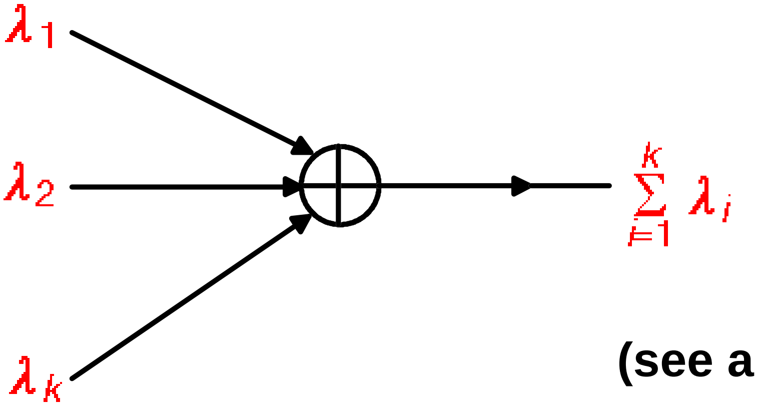
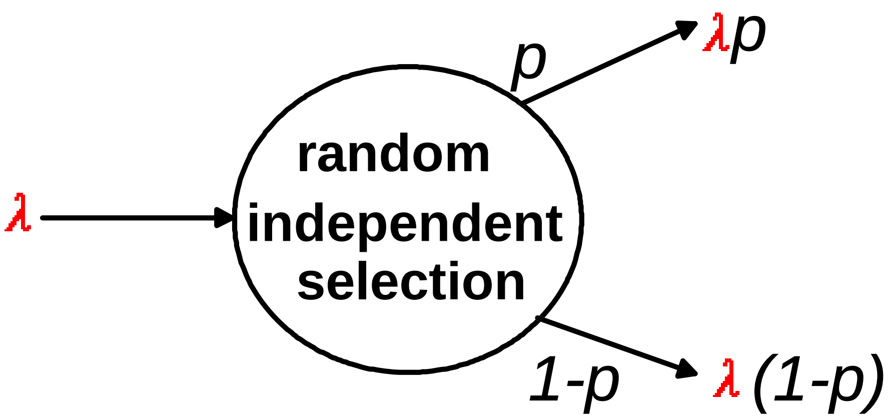
- **If** $A_1(t), A_2(t), \cdots, A_k(t)$ **are independent Poisson processes of rate** $\lambda_1, \lambda_2, \cdots, \lambda_k$**, then** $A_1(t) + A_2(t) + \cdots + A_k(t)$ **is a Poisson process of rate** $\lambda_1 + \lambda_2 + \cdots + \lambda_k$



(see also Ex.3.10c)



If each arrival of a Poisson process is independently sent to system 1 with prob. p **and system 2 with prob.** 1-p**, the arrivals to each system are Poisson and independent.(see also Ex.3.11a)**

Routing in Data Nets

Datagarms: routing decision for every packet.

Virtual Circuits: one routing decision for all the packets of the same session.

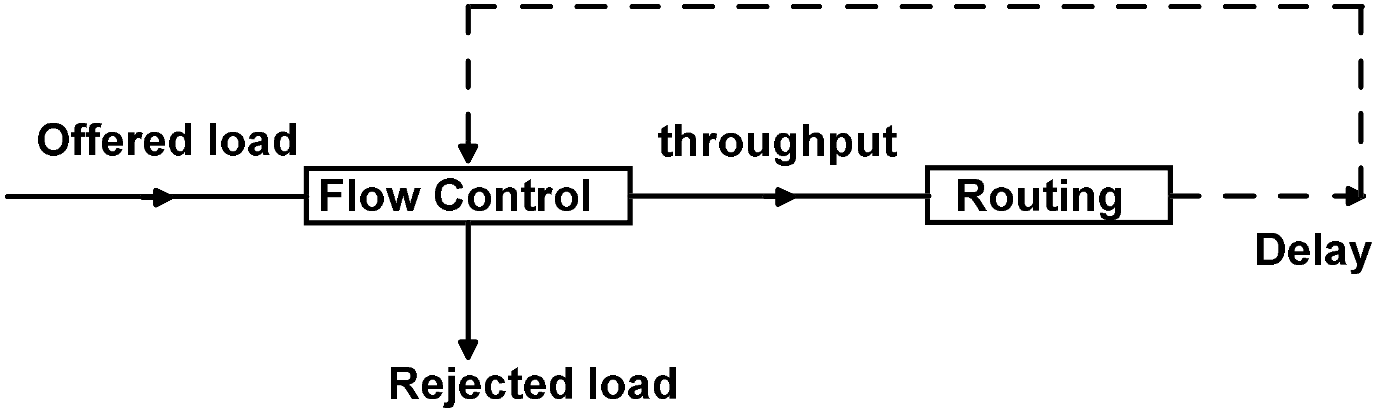
Issues

- selection of paths
- broadcasting of routing-related info.

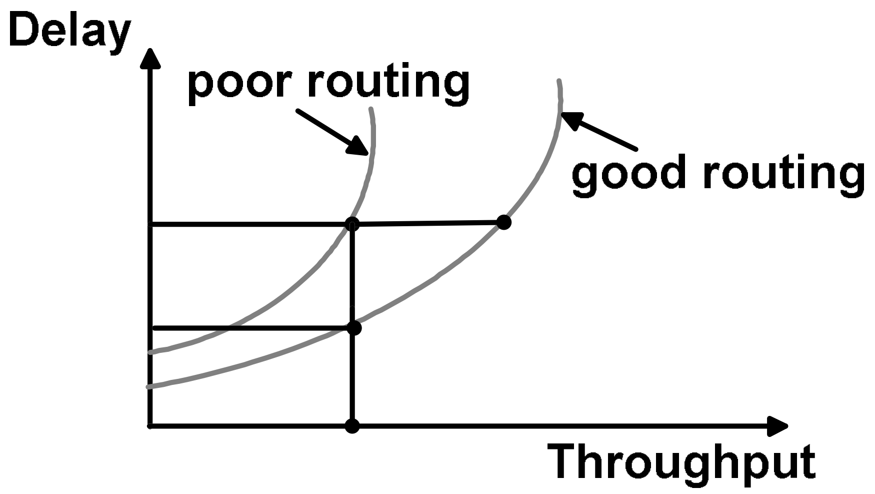
Performance Measures

Throughput (“quantity” of service)

Average packet delay (“quantity” of service)

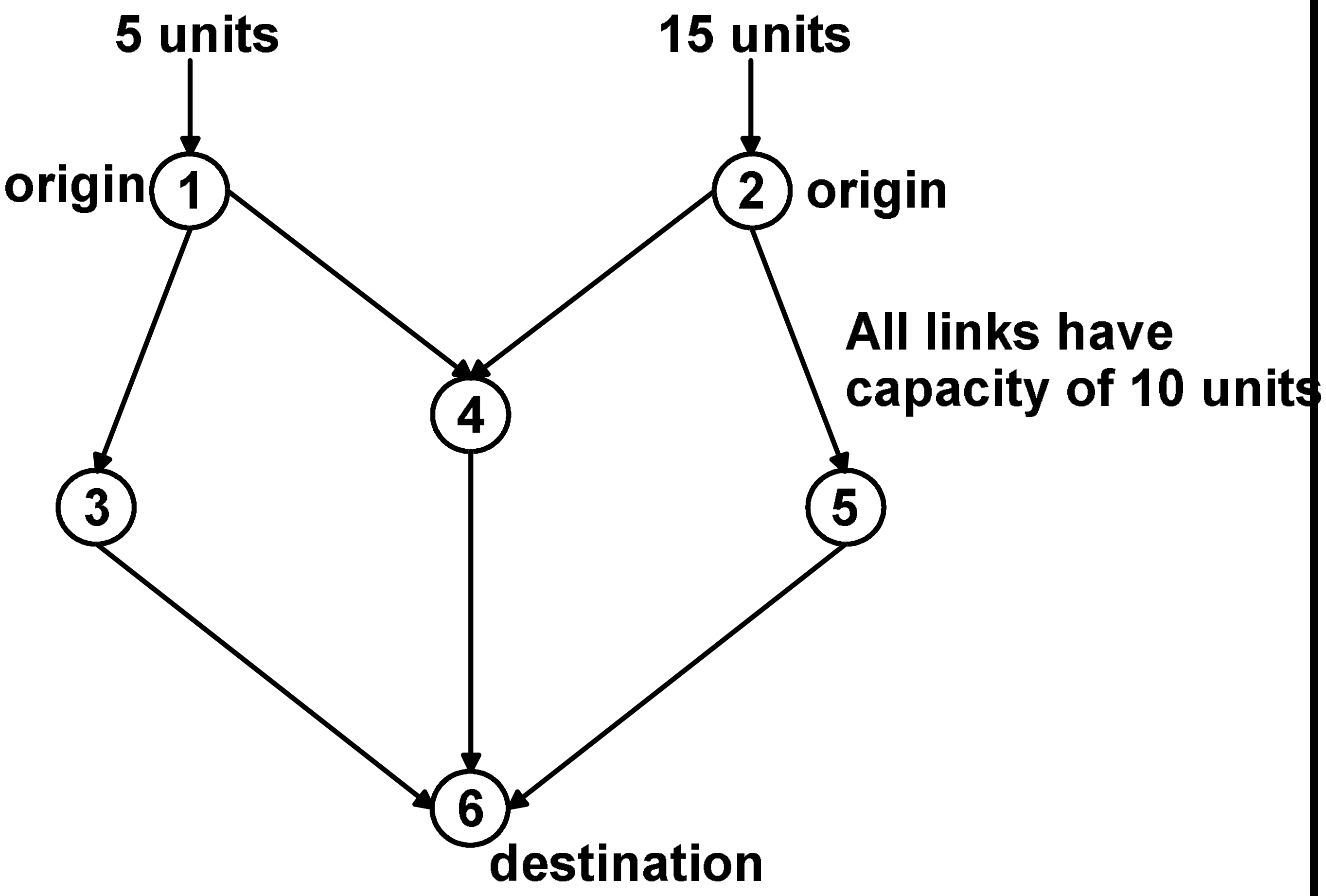


As the routing algorithm succeeds in keeping delay low, the flow control allows more traffic into the network.



Good routing algorithms: higher throughput for the same delay; smaller delay for a given throughput.

Example:



- If everything routed through middle path the delay is large.
- Suppose input traffic at node 2 is increased to 15 units.

  If a single path is used at least 5 units is rejected.

Therefore the delay and the maximum throughput depend on routing.

Routing may be:

1. Centralized or Distributed

- Centralized: all routing choices are made at a central node.
- Distributed: computation of routes is shared among nodes who exchange information if necessary.

2. Static or Dynamic (adaptive)

- Static: path used for an origin-destination pair is fixed.
- Dynamic: path may changes in response to congestion.

Of course even with static algorithms paths will change if nodes or links fail.

Shortest Path problem

Directed graph $G=(N,A) \rightarrow$ **arcs have direction**

$d_{ij} =$ **length of** (i,j)

The length of a directed path $p=\{i,j,k,.....,l,m\}$ **is defined as** $d_{ij} + d_{jk} + \cdots\cdots + d_{lm}$

Given: $G=(N,A)$, d_{ij}*'s* **with no negative length cycles, and a node** *1*

Problem: find the shortest path from every node *i* **to node** *1*

Applications: a) d_{ij}**=delay** *(i,j),* **then "shortest path" corresponds to the "minimum delay" path**

b) If p_{ij} **= prob.** *(i,j)* **is operational. Let define** $d_{ij} = -\ln p_{ij}$ **. Then " shortest path" corresponds to "most reliable" path**

Bellman-Ford algorithm

Let $D_i^h =$**length of shortest path from** i **to 1 that uses** $\le h$ **arcs**

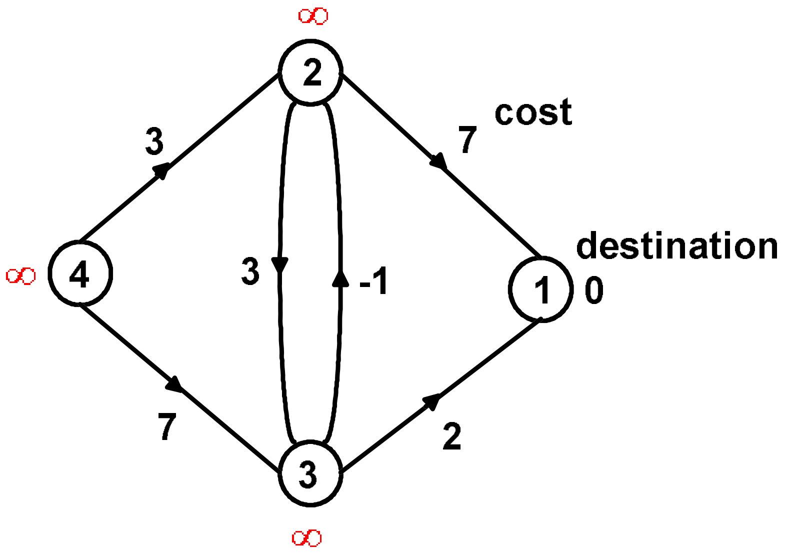
$$D_1^0 = 0$$

$$D_i^0 = \infty \ , \ i \neq 1$$

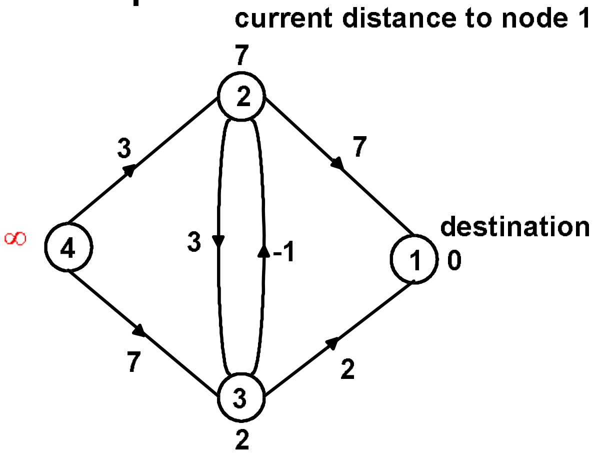
$$D_i^{h+1} = \min_{j \in N(i)} [d_{ij} + D_j^h] \quad *$$

After at most *N-1* **iterations * shortest path is founded (provided that there are no negative length cycles); shortest path distance** $D_i = D_i^{N-1}$**.In fact iteration terminates when after** h **iteration** $D_i^h = D_i^{N-1}$ **for all** i

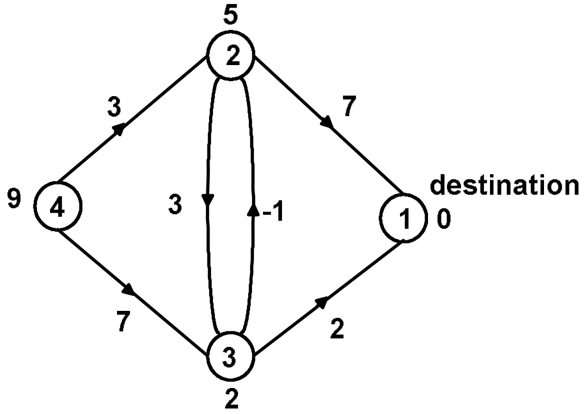
Example: Bellman-Ford (infinite initial conditions)



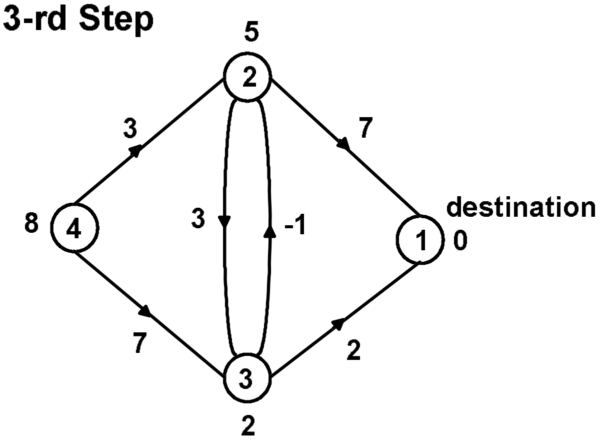
1-st Step



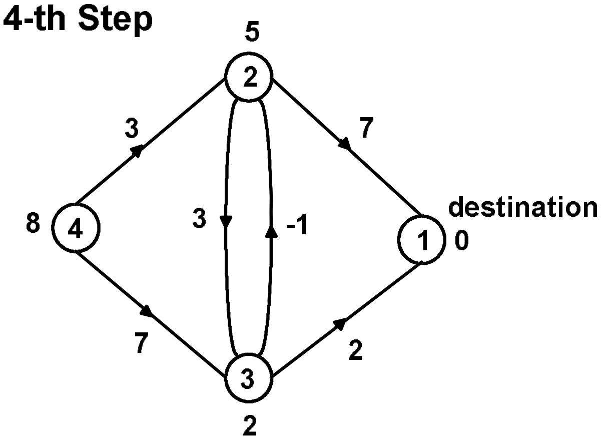
2-nd Step



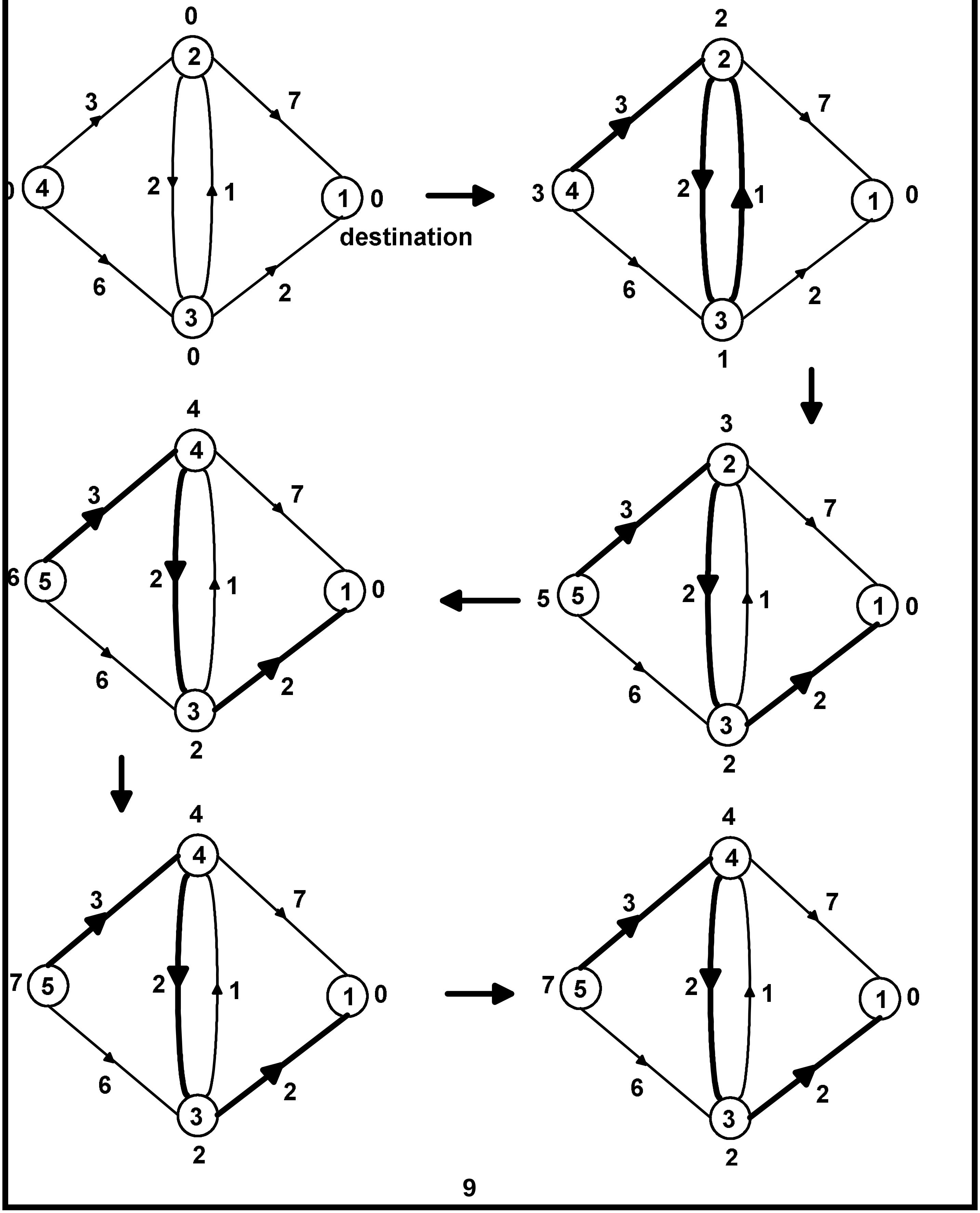
3-rd Step



4-th Step



Example: Bellman-Ford zero initial conditions



Routing in the ARPANET

Uses shortest paths from origin to destination.

1969 algorithm:
Node *i* computes an estimate D_i of its distance to a given node *0*

$$D_i = \min_{j \in N(i)} (d_{ij} + D_j), \ i \neq 0,$$

$N(i)$**: neighbors of *i***
D_j**: obtained by neighbors every 0.62 sec**

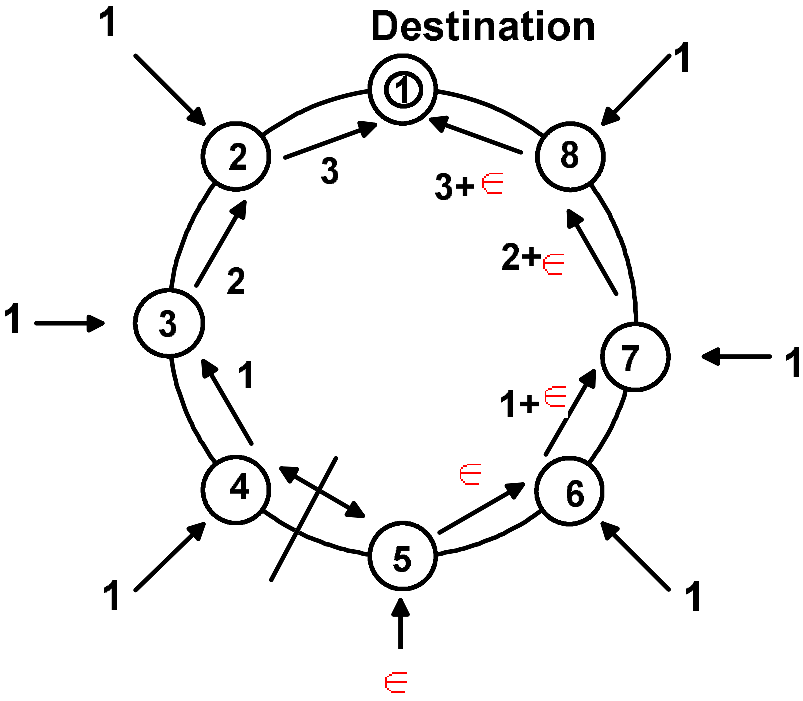
$$D_0 = 0$$

Originally (1969), lengths d_{ij}= number of packet in buffer *(i,j)*

Instability problems (original ARPANET)

Problem: Shortest paths → routes → flow → arc lengths

Feedback here can be unstable.



$D_2=3$

$D_3=5$

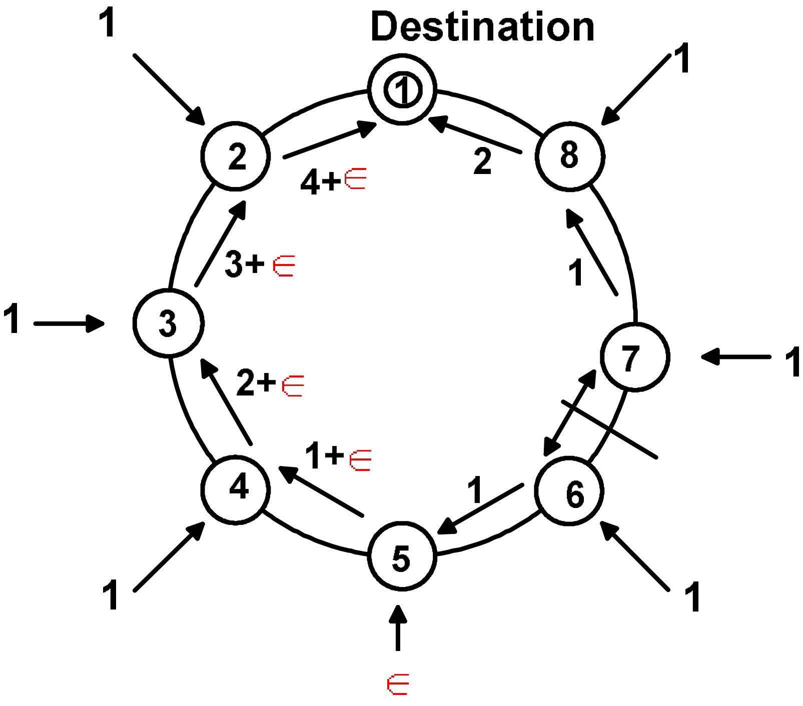
$D_4=6$

$D_5=\cancel{6+4\in}$ 6

$D_6=\cancel{6+3\in}$ 6

$D_7=5+2\in$

$D_8=3+\in$



$D_2=\cancel{3+\in}$ 3

$D_3=\cancel{7+2\in}$ 3

$D_4=\cancel{9+3\in}$ 3

$D_5=\cancel{10+4\in}$ 3

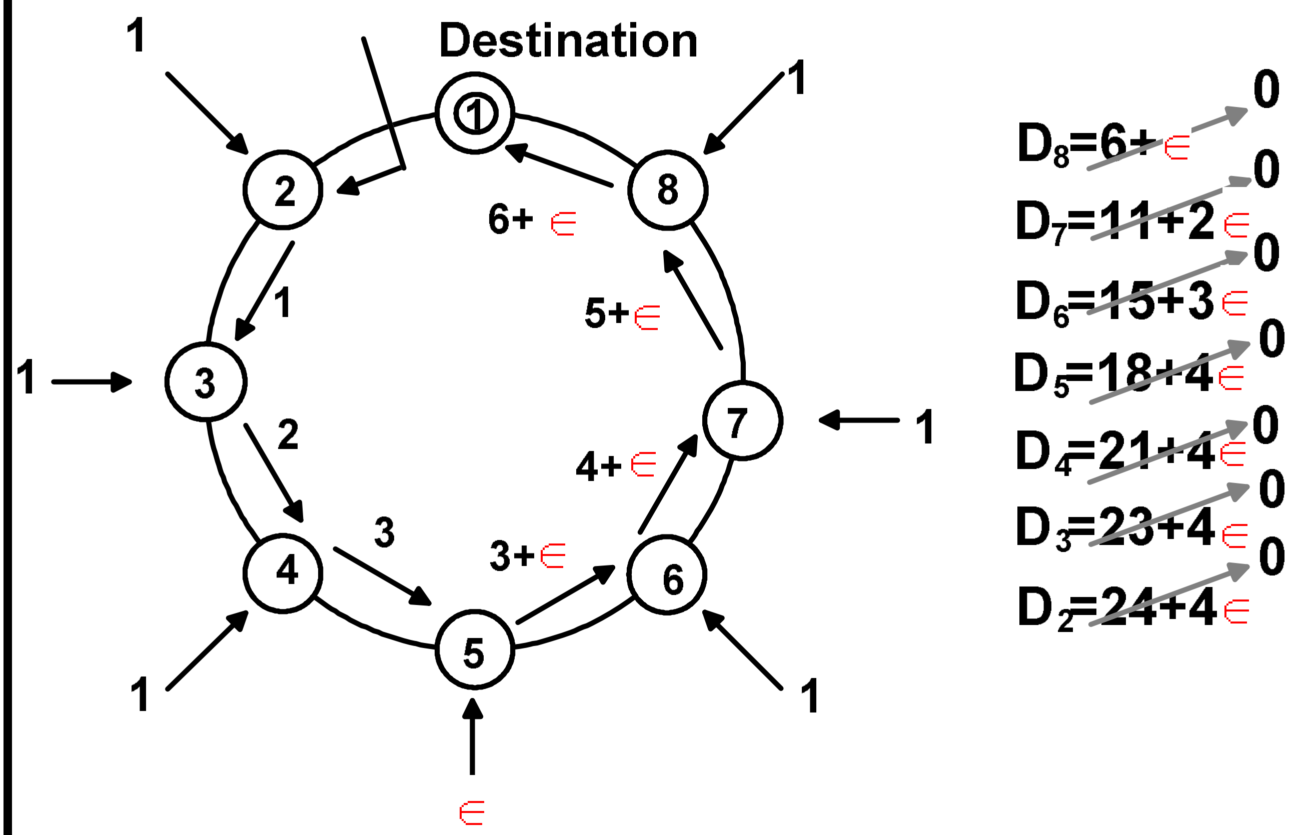
$D_6=\cancel{11+4\in}$ 3

$D_7=3$

$D_8=2$

Assume is equal to flow on (i,j)

Note that next shortest path of node 2 will have a counterclockwise direction.



$D_8=6+\in$ → 0

$D_7=11+2\in$ → 0

$D_6=15+3\in$ → 0

$D_5=18+4\in$ → 0

$D_4=21+4\in$ → 0

$D_3=23+4\in$ → 0

$D_2=24+4\in$ → 0

After updates, everybody will start sending packets clockwise, and so on.....

Having a bias independent of flow in the arc distances helps to prevent this problem (e.g. d_{ij} = constant + number of packets in the buffer)

1979 ARPANET Algorithm

d_{ij} =average delay of packets crossing (i,j) in the last 10 secs. The d_{ij} 's are broadcast every 60 secs to all other nodes, using a flooding mechanism.

Nodes update their shortest paths (asynchronously), using Dijkstra’s algorithm.

Each node keeps track of the first link on the shortest path.

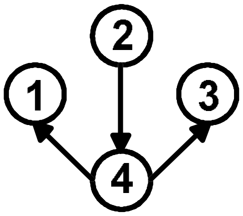


Table of 2	
Dest.	Out. link
1	(2,4)
3	(2,4)
4	(2,4)

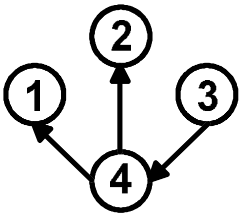


Table of 1	
Dest.	Out. link
2	(1,4)
3	(1,4)
4	(1,4)

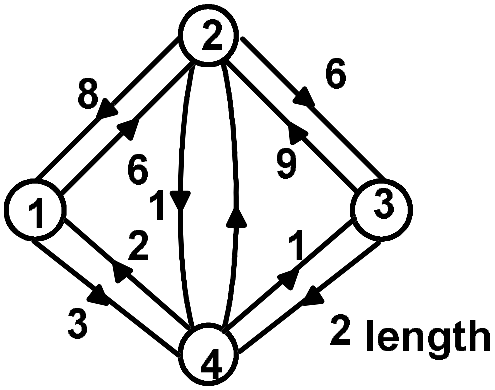


Table of 3	
Dest.	Out. link
1	(3,4)
2	(3,4)
4	(3,4)

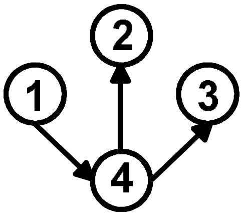
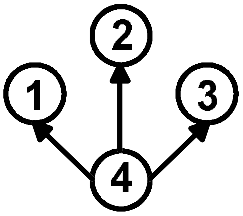
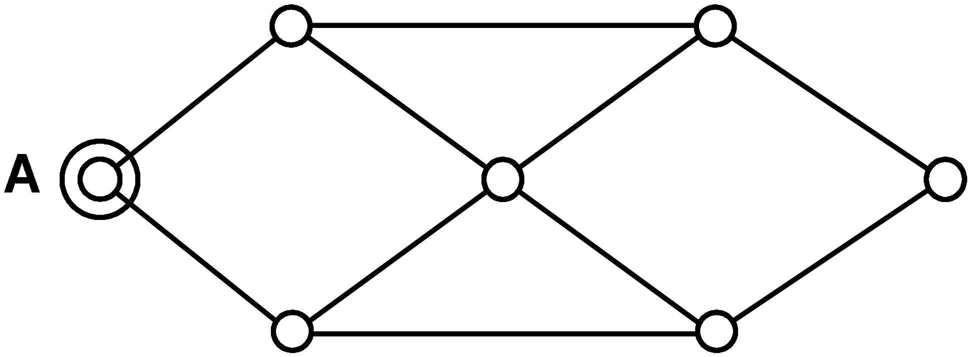


Table of 4	
Dest.	Out. link
1	(4,1)
2	(4,2)
3	(4,3)

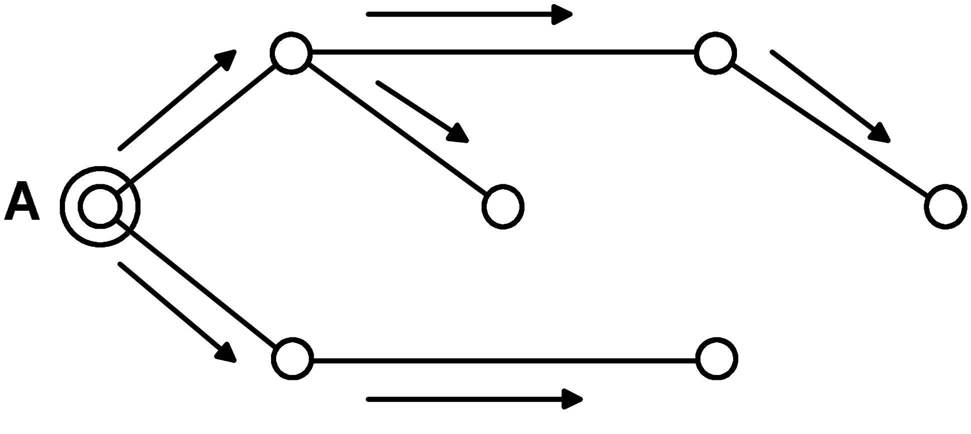


Link lengths are broadcast to all other nodes through flooding algorithm:



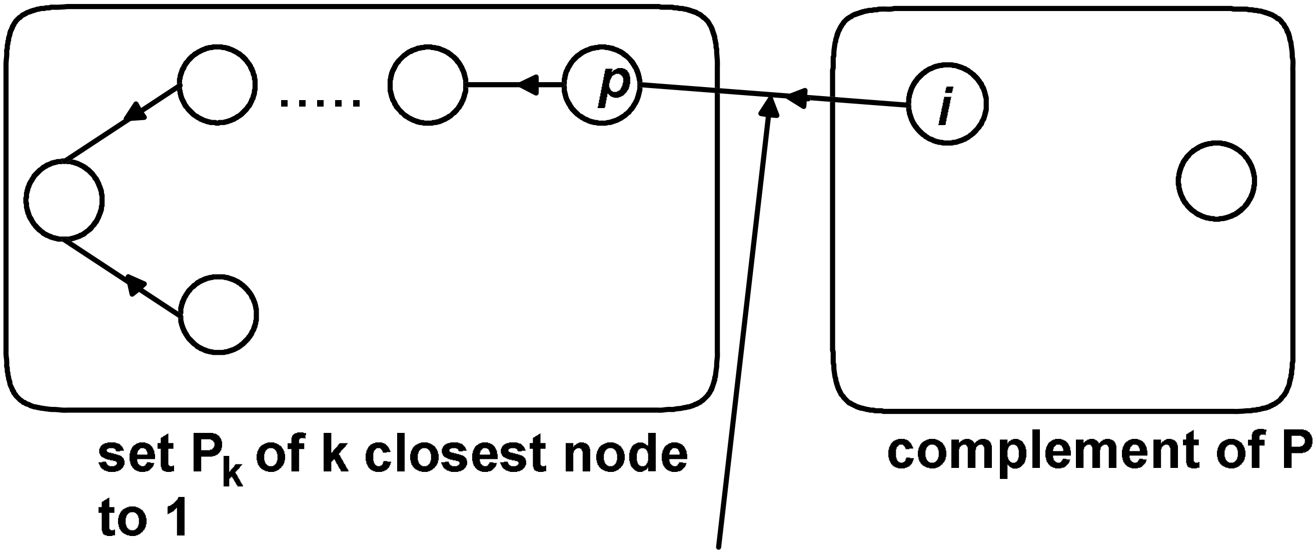
- origin sends information to neighbors
- The neighbors send it to their neighbors (except for node from which they received it)
- Each packet has a SN and a node ID. Using this information, nodes avoid forwarding a packet twice.

Note: other alternatives include broadcasting over a spanning tree



Dijkstra’s Algorithm

Given: *G=(N,A)*, $d_{ij} \geq 0,$ **destination node 1.**
Idea: find the shortest paths in order of increasing path length.



set P_k of k closest node to 1

complement of P_k

shorest path from (k+1)th ckosest node must pass through nodes in P_k

$P_1 = \{1\}, D_1 = 0, D_j = d_j$ **for** $j \neq 1$

1. Find next closest node. Find $i \notin P_{k-1}$ **such that**

$$D_i = \min_{j \notin P_{k-1}} D_j$$

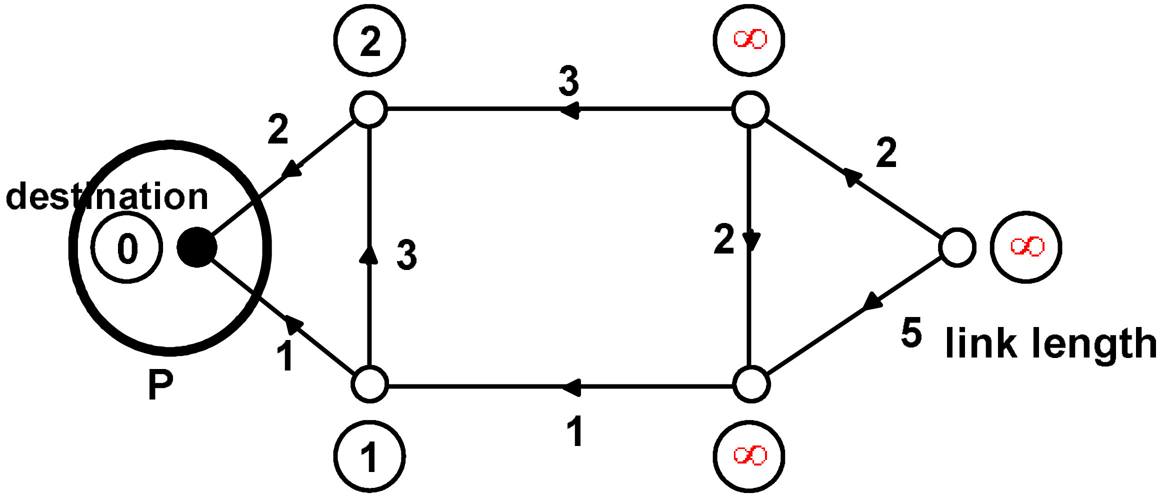
set $P_k = P_{k-1} \cup \{i\}$. **If all nodes covered, stop.**

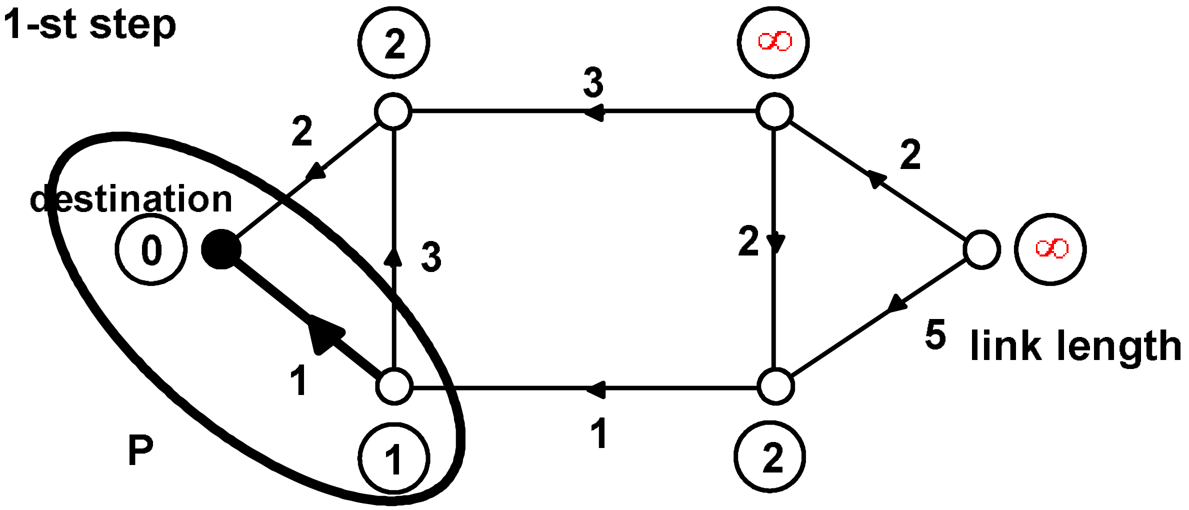
2. Update labels. For all $j \notin P_k$

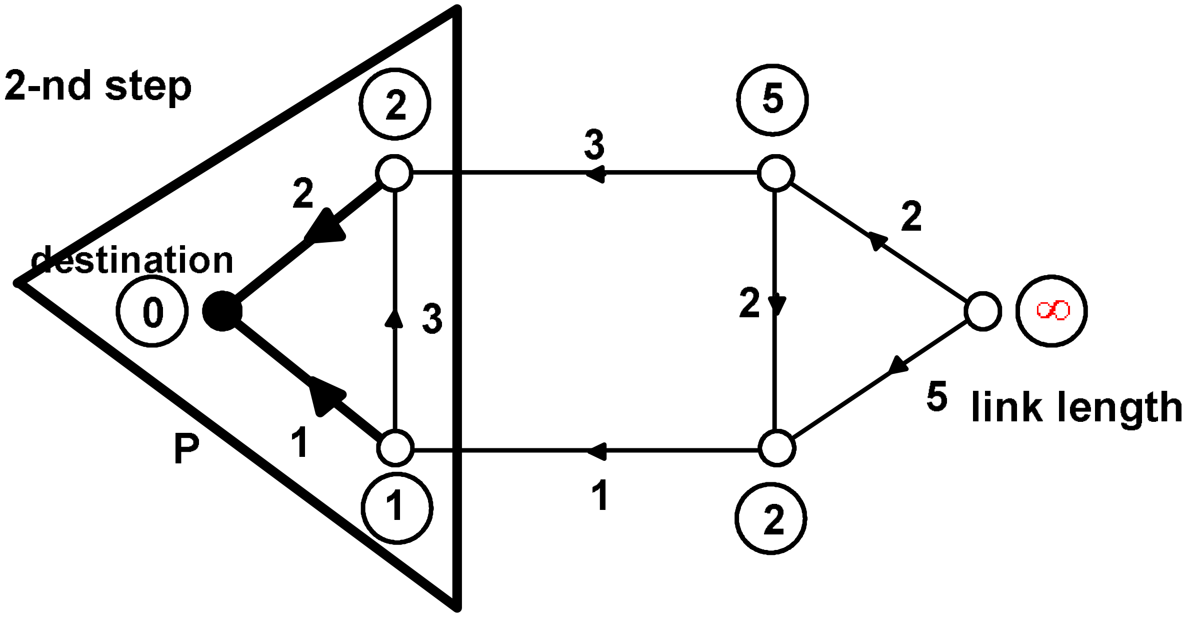
$$D_j = \min_{i \in P_k} \{d_{ji} + D_i\}$$

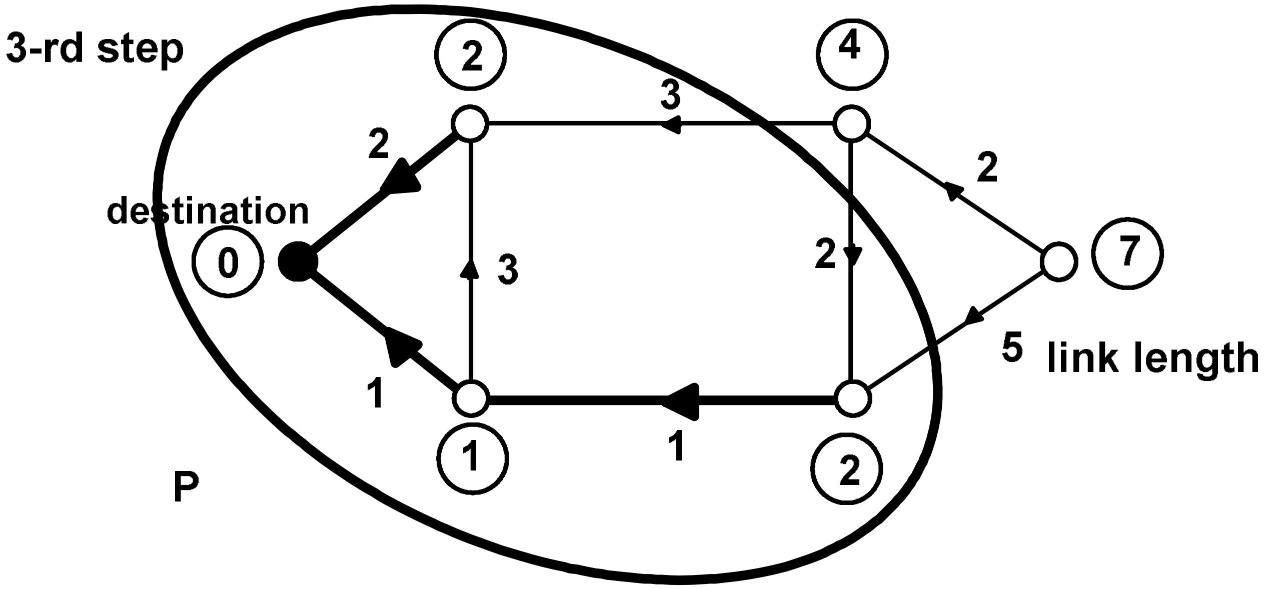
Go to 1.

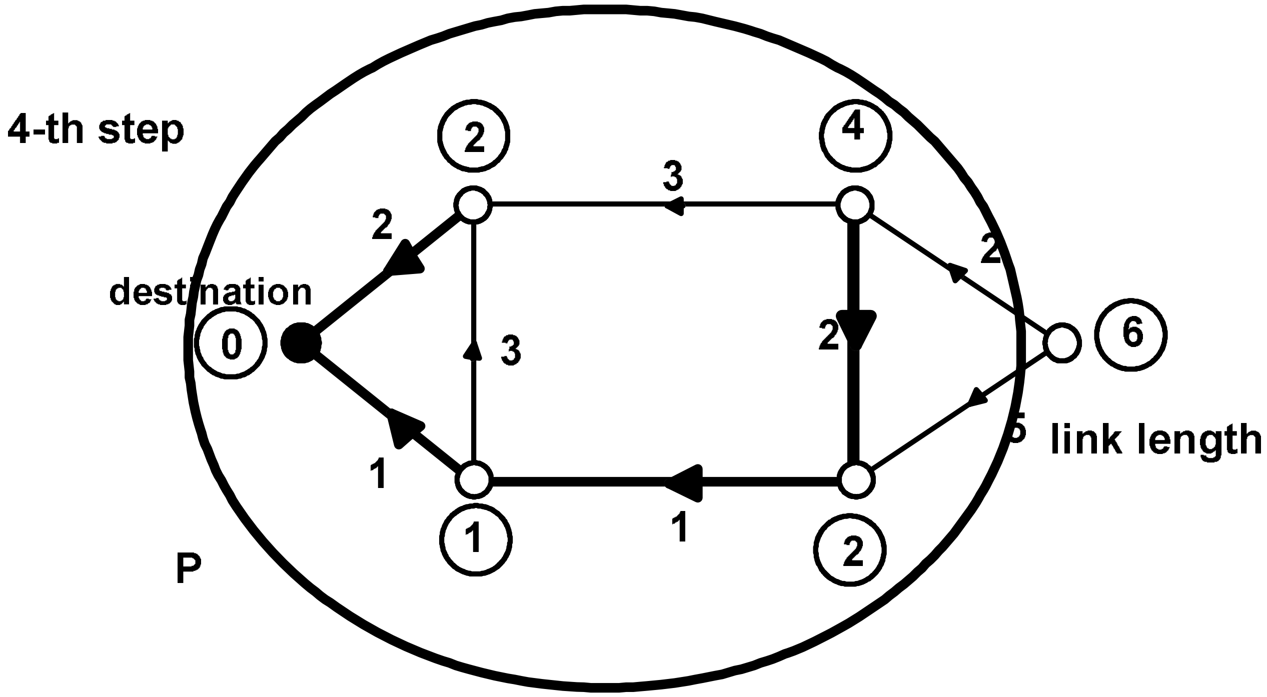
Example: (Dijkstra’s Algorithm)

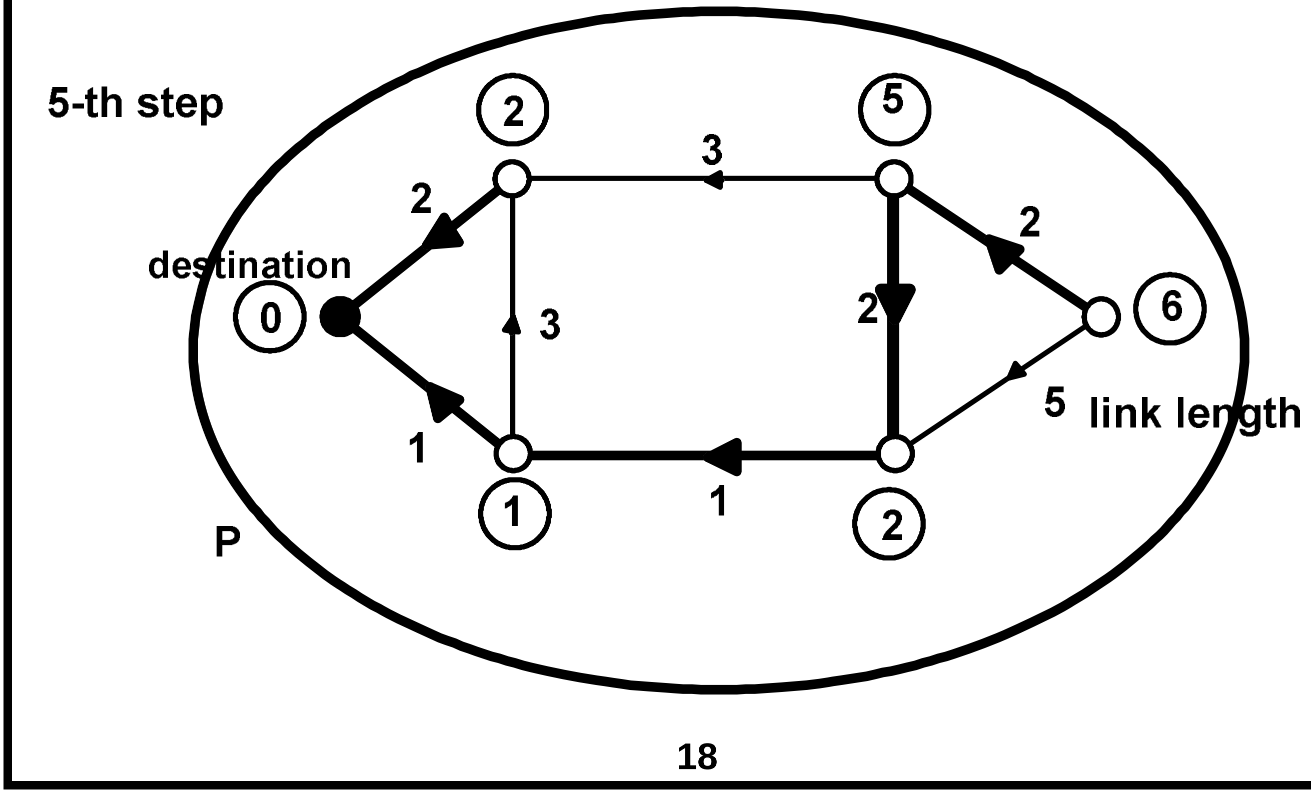





2-nd step
2
5
3
2
2
destination
0
2
3
∞
5 link length
P
1
1
1
2


3-rd step
2
4
3
2
2
destination
0
2
3
7
5 link length
1
1
1
2
P


4-th step
2
4
3
2
2
destination
0
2
6
3
5
link length
1
1
1
2
P


5-th step
2
5
3
2
2
destination
0
2
6
3
5
link length
1
1
1
2
P