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.

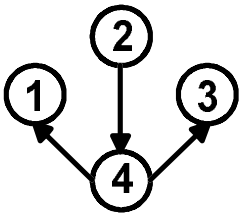


Table of 2	
Dest.	Out. link
1	(2,4)
3	(2,4)
4	(2,4)

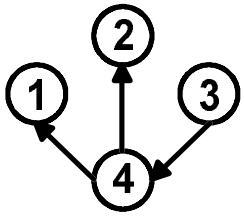


Table of 3	
Dest.	Out. link
1	(3,4)
2	(3,4)
4	(3,4)

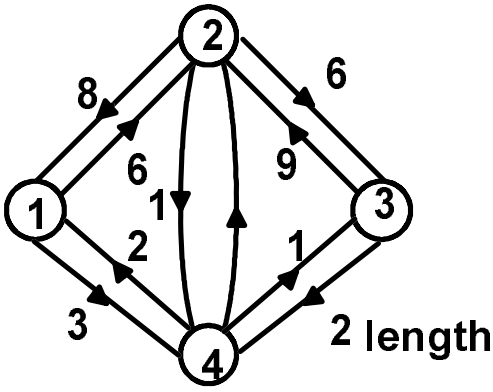


Table of 1	
Dest.	Out. link
2	(1,4)
3	(1,4)
4	(1,4)

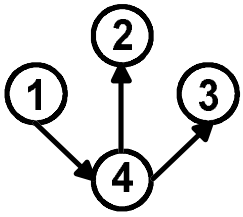
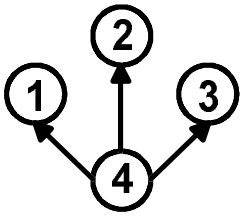
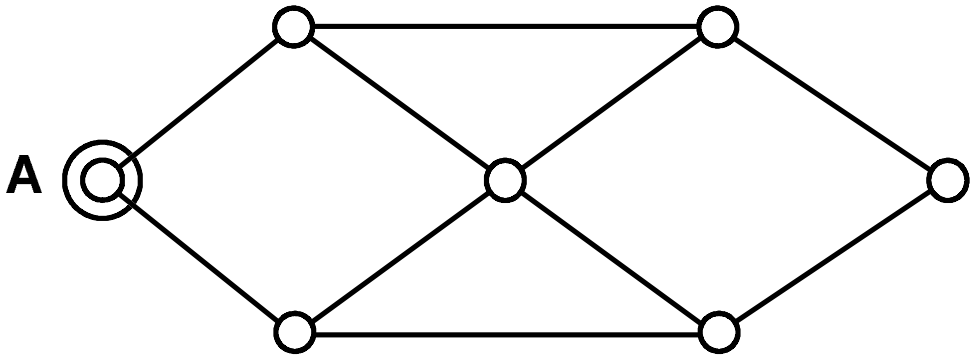


Table of 4	
Dest.	Out. link
1	(4,1)
2	(4,2)
3	(4,3)

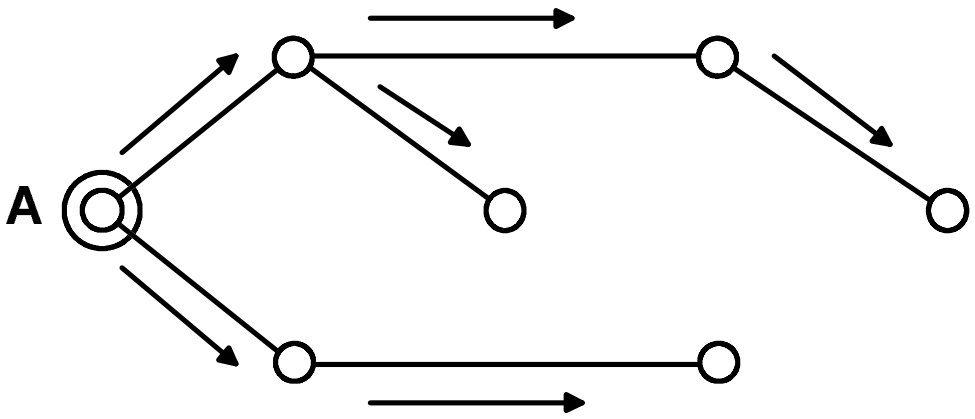


Link lengths are broadcast to all other nodes through flooding algorithm:



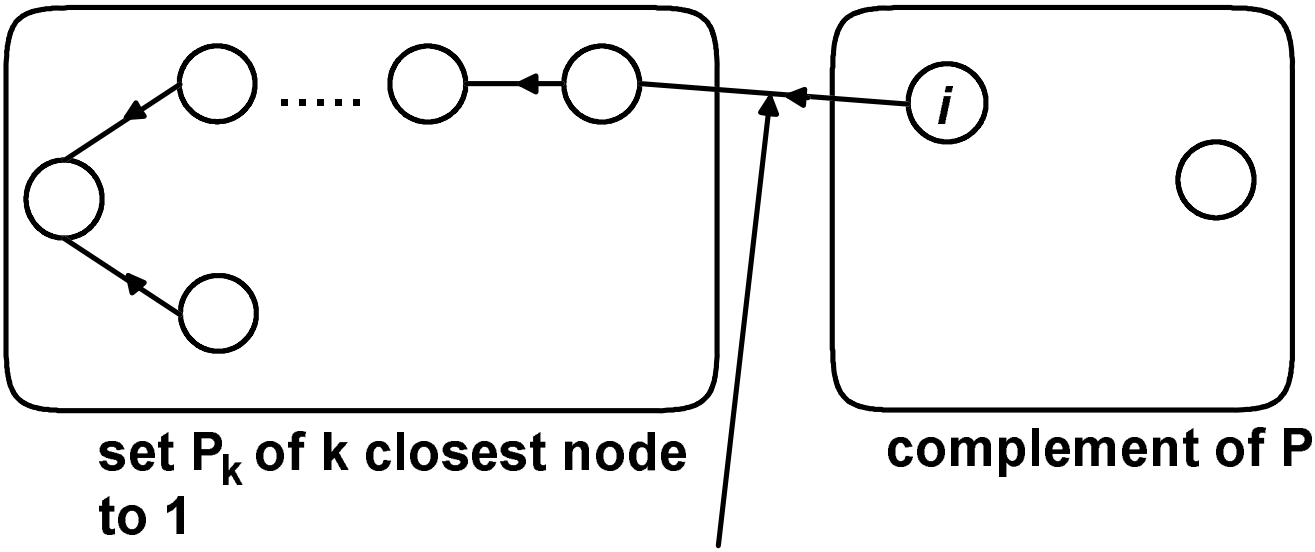
- **origin sends information to neighbors**
- **The neighbors send it to their neighbors (except for node from which they received it)**
- **Each packet has a SN and a node ID. Using this information, nodes avoid forwarding a packet twice.**

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**

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

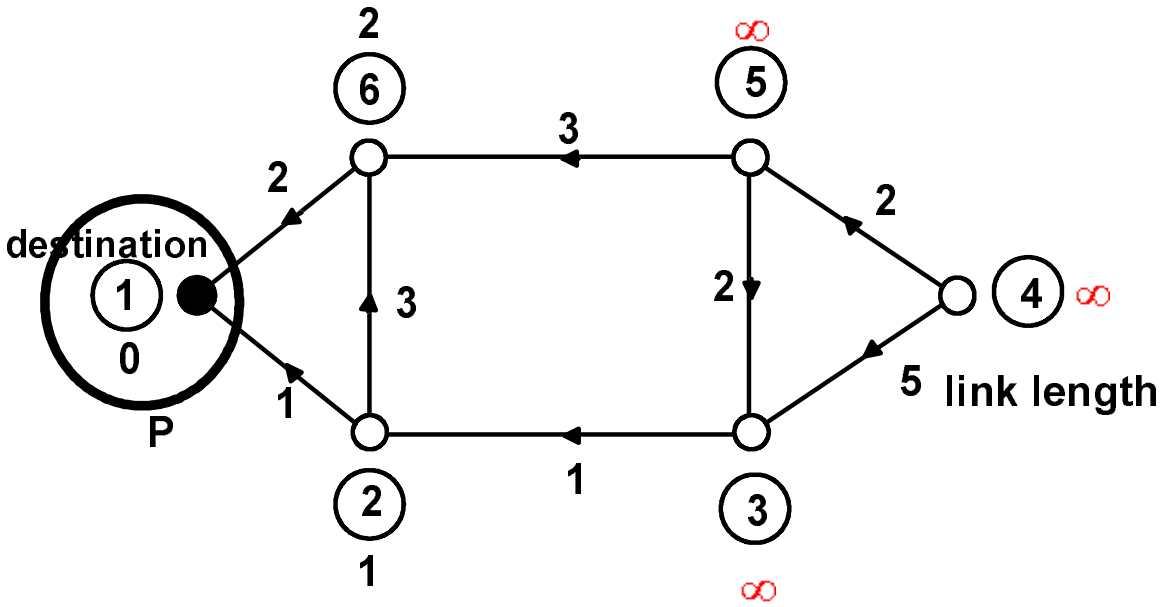
set $P_k = P_{k-1} \cup \{i\}$. **If all nodes covered, stop.**

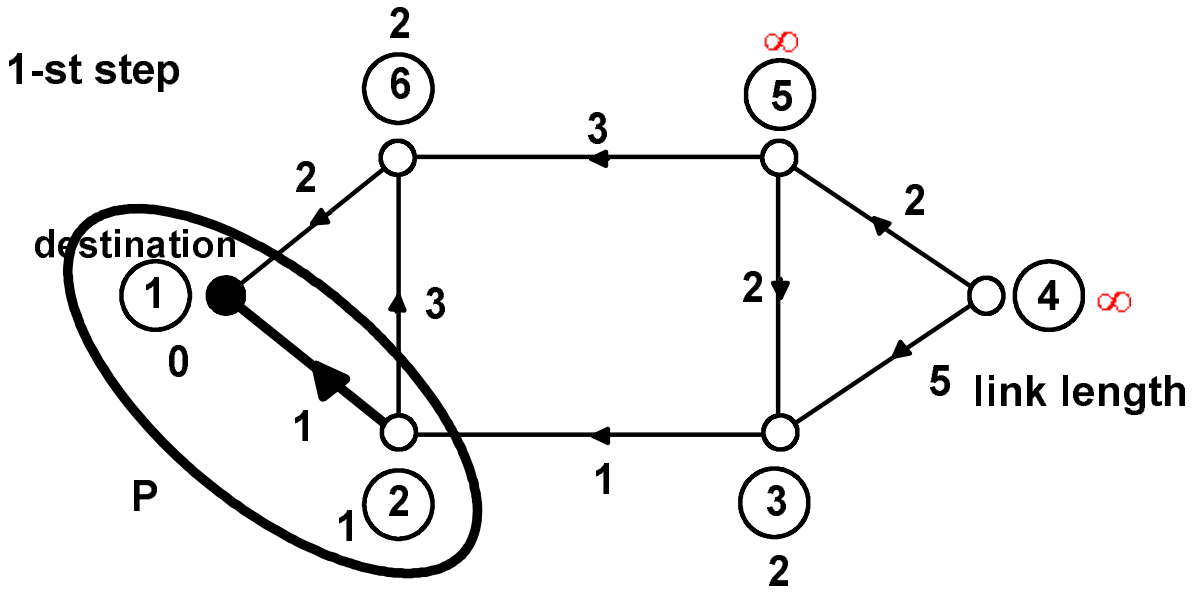
2. Update labels. For all $j \notin P_k$

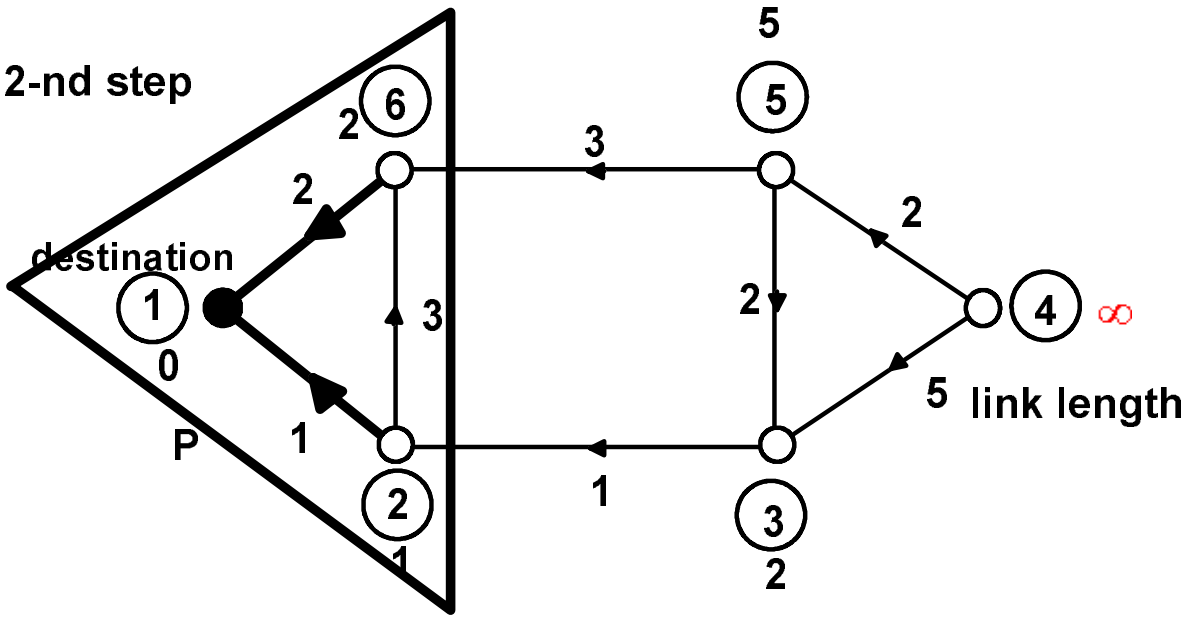
$$D_j = \min_{i \in P_k} \{d_{ji} + D_i\}$$

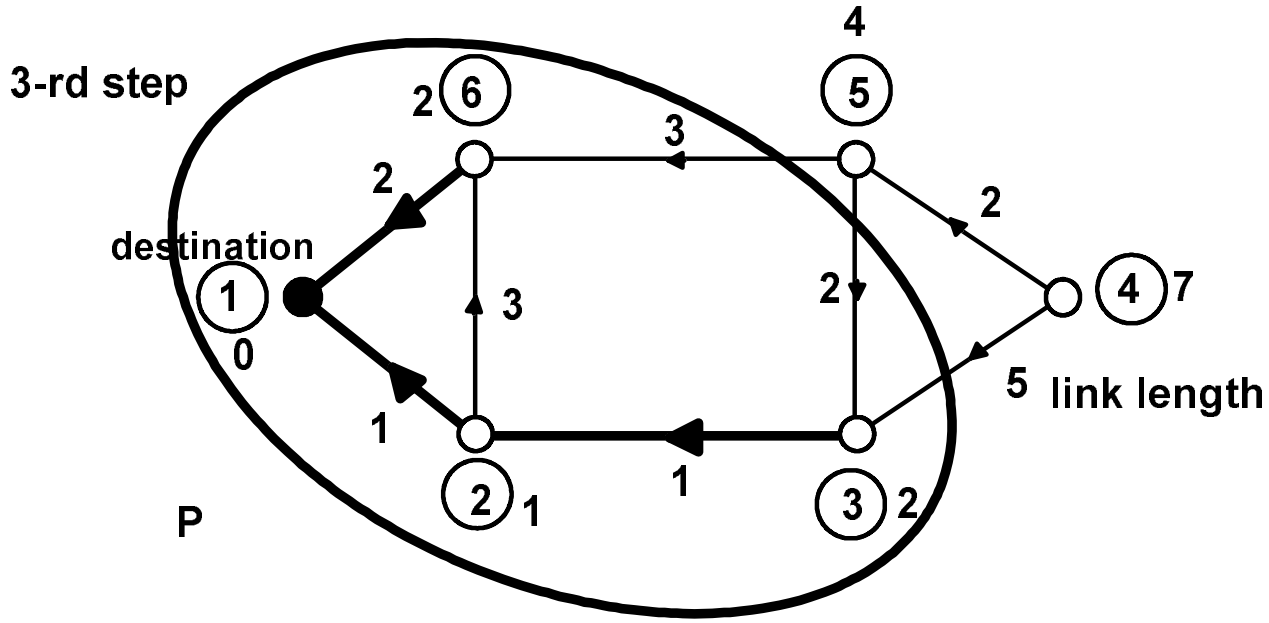
Go to 1.

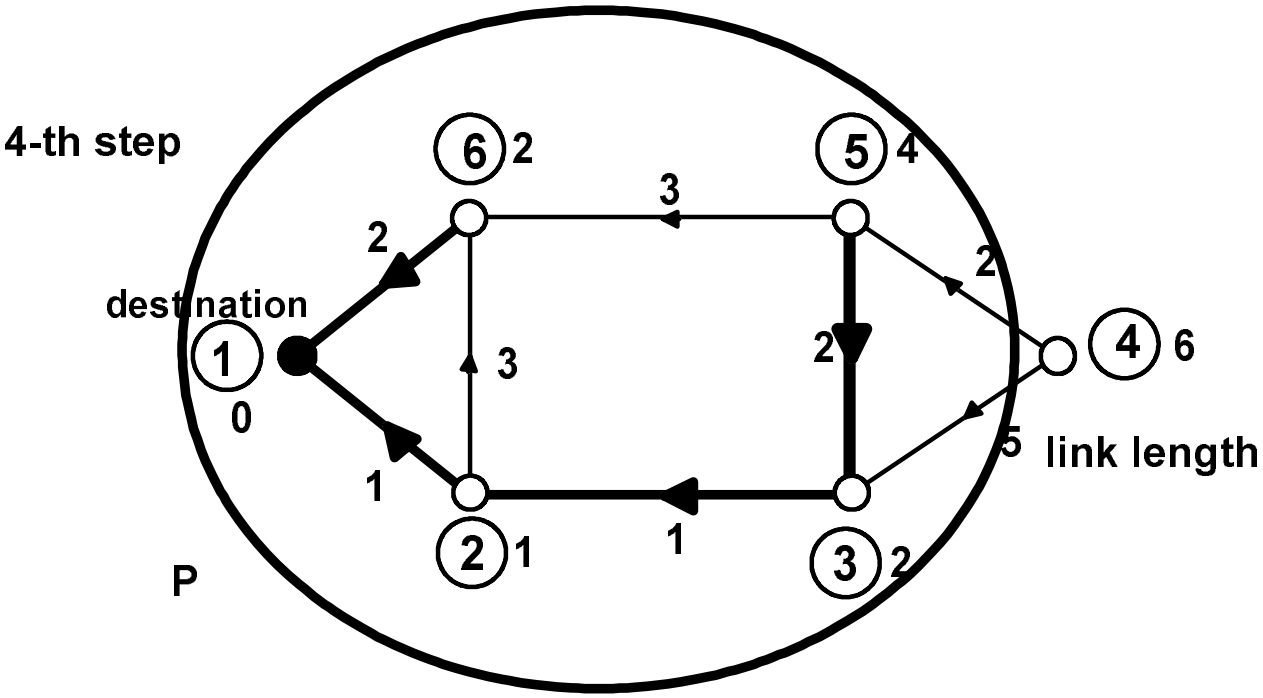
Example: (Dijkstra's Algorithm)

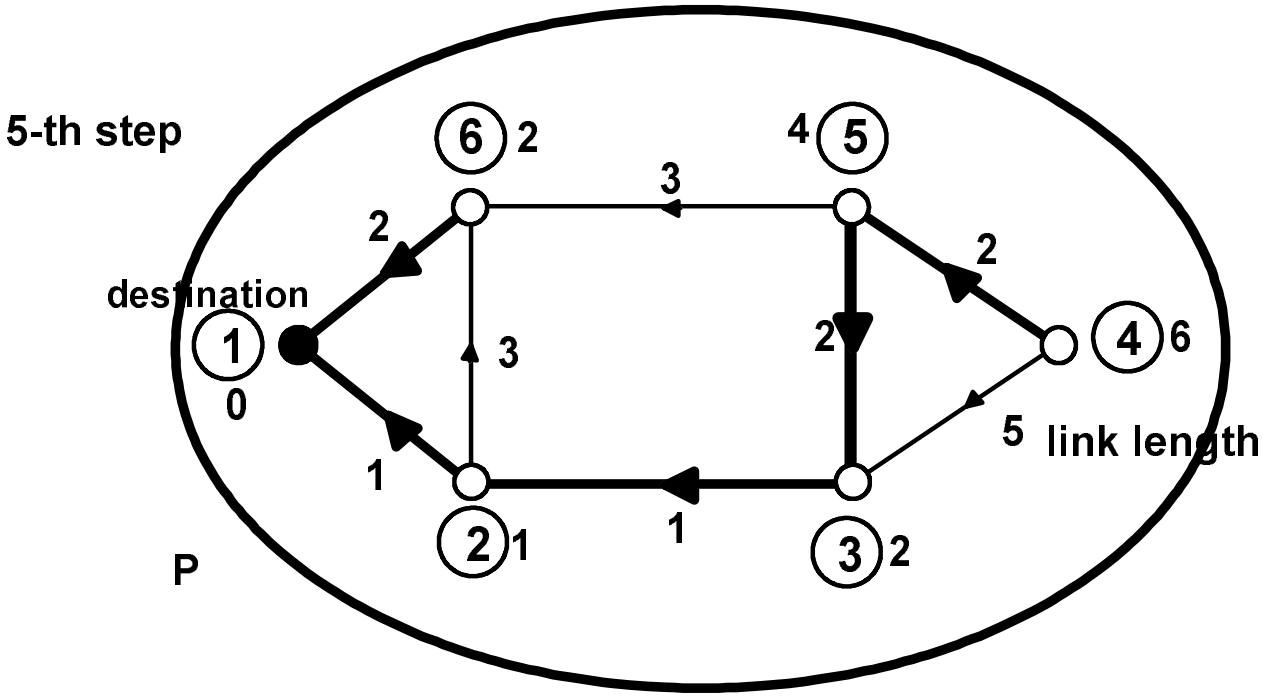





2-nd step
destination
1
0
P
6
2
2
3
5
5
3
2
2
4
∞
5 link length
1
2
1
1
3
2

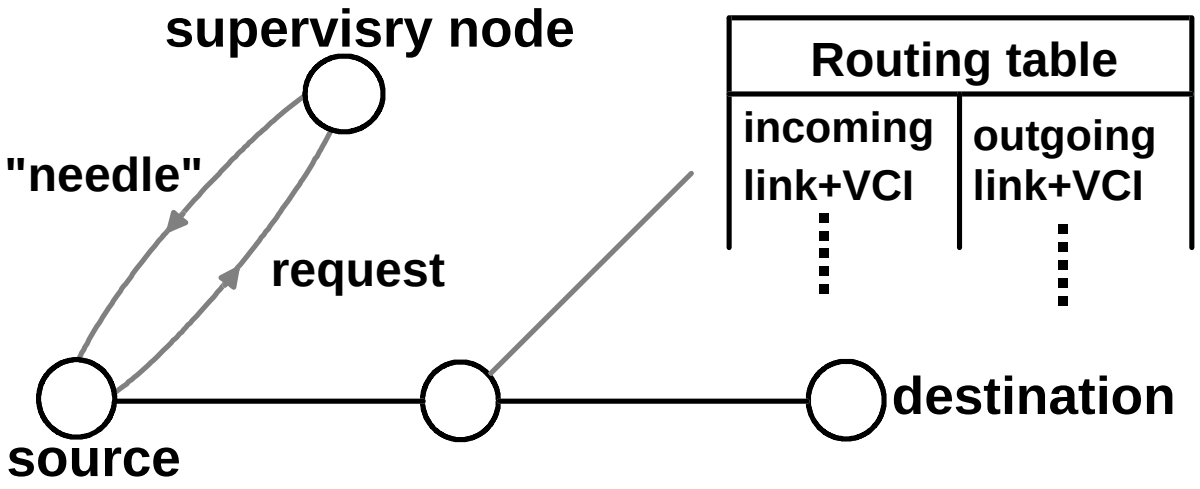

3-rd step
destination
1
0
P
6
2
2
3
4
5
3
2
2
4
7
5 link length
1
2
1
1
3
2


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


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

TYMNET Network (1971 and 1981)

Uses same adaptive shortest path idea, but routing is centralized (virtual circuits)



The supervisory node computes the shortest path and sends a “needle” packet to source that contains the routing information.

TYMNET I: the supervisory node explicitly sets the routing tables at the nodes.

TYMNET II: routing tables are set by a set-up packets that precedes the transmission of data.

Similar idea used in Codex network

Bellman-Ford algorithm

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

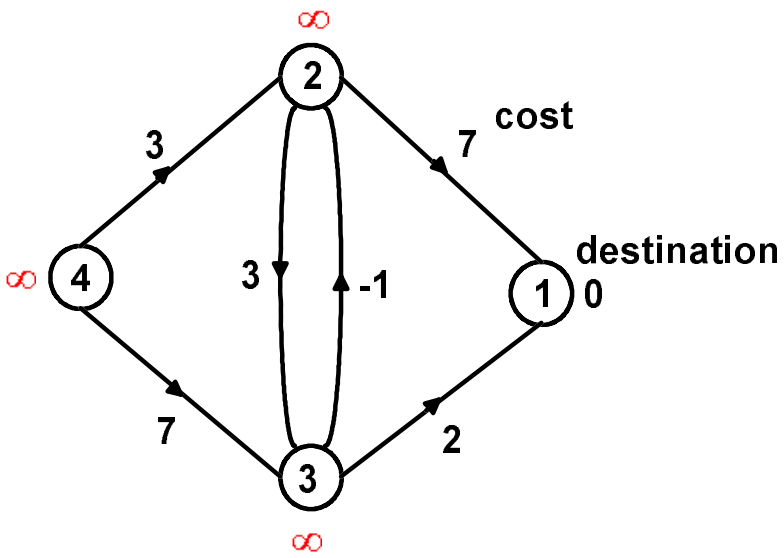
$$D_1^0 = 0$$

$$D_i^0 = \infty \text{ , } i \neq 1$$

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

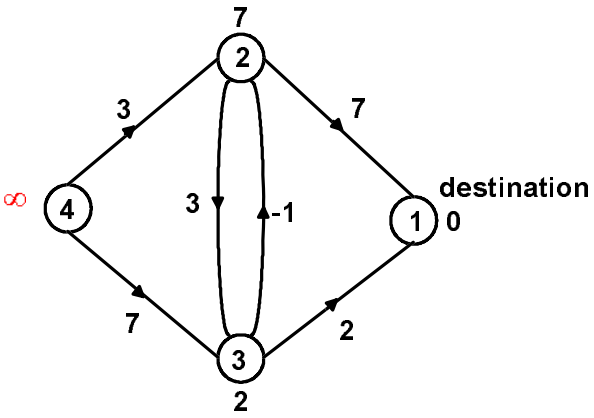
After at most *N-1* **iterations * shortest path is founded (provided that there are no negative length cycles); shortest path distance** $D_i = D_i^{N-1}$**.In fact iteration terminates when after** *h* **iteration** $D_i^h = D_i^{N-1}$ **for all** *i*

Example: Bellman-Ford (infinite initial conditions)

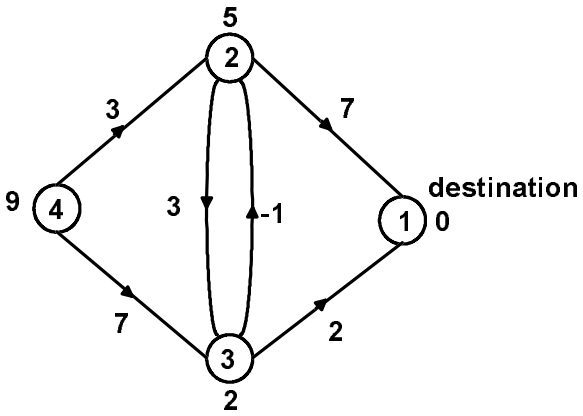


1-st Step

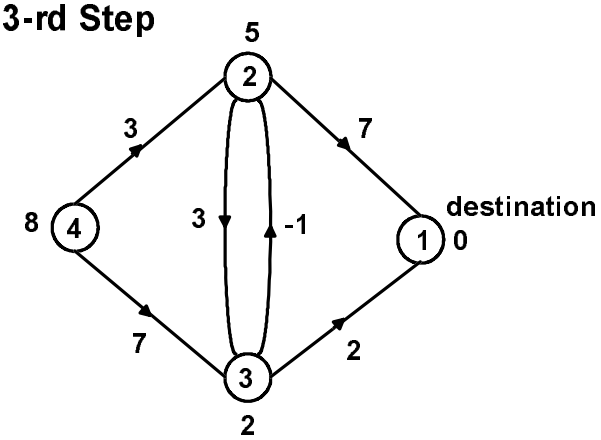
current distance to node 1



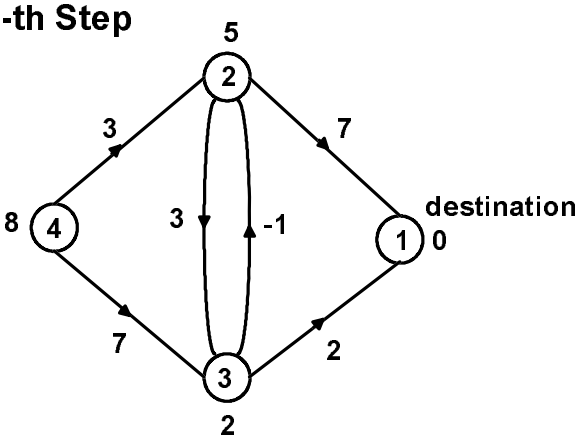
2-nd Step



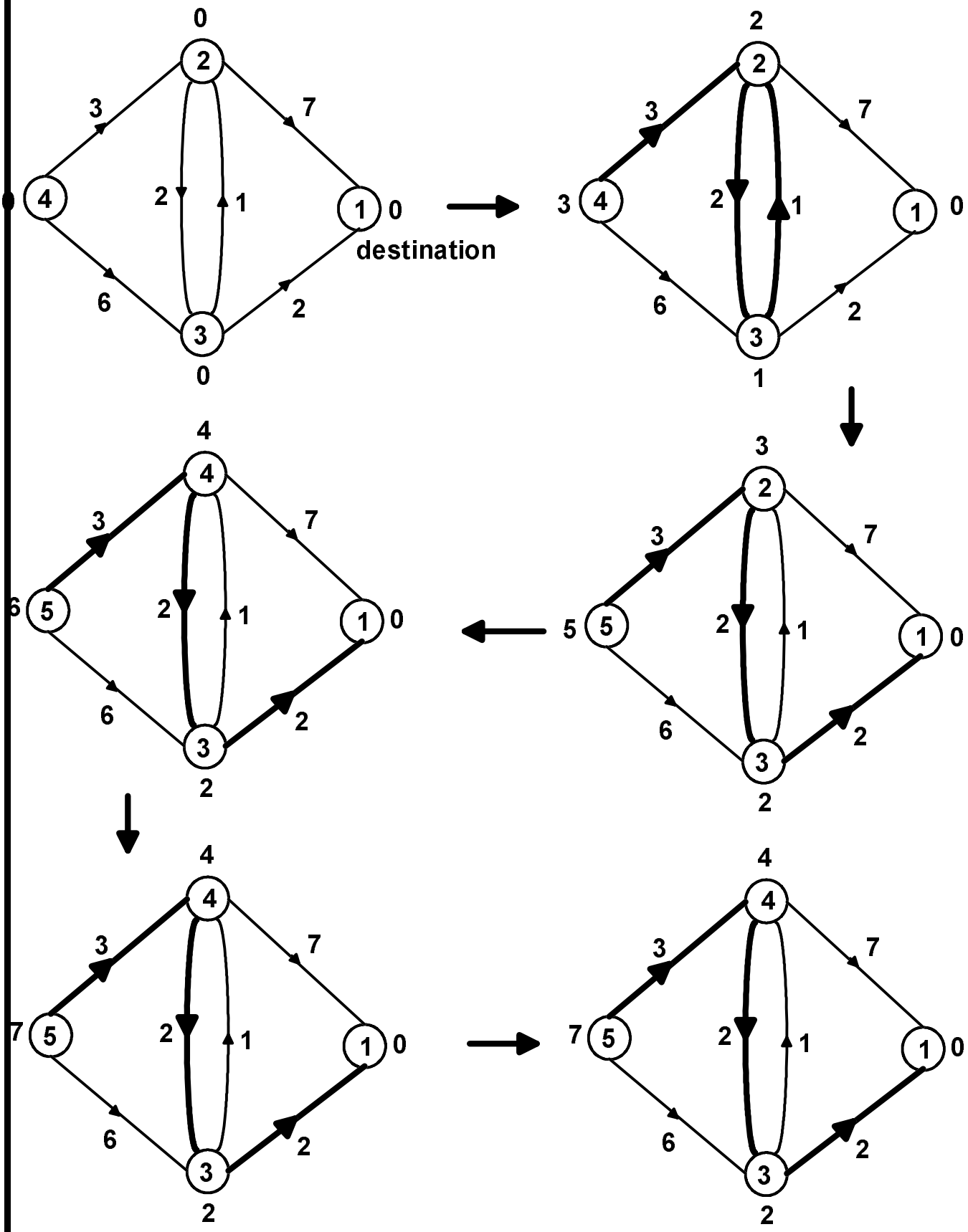
3-rd Step



4-th Step



Example: Bellman-Ford zero initial conditions



Bellman-Ford algorithm (revisited)

Initial conditions: $D_1^{(0)} = 0, \quad D_i^{(0)} = \infty \text{ , } i \neq 1$

$$D_i^{(k+1)} = \min_{j \in N(i)} [d_{ij} + D_j^{(k)}] \quad i \neq 0 \text{ , } D_1^{(k+1)} = 0$$

$D_i^{(k)}$ = **length of shortest path from** i **to 1 using up to** k **links**

Algorithm terminates after at most $N-1$ **steps.**
Can be implemented in a distributed way.

Problem 1: it is often preferably to send distances asynchronously rather than waiting for D_j^k **from all neighbors, before new iteration.**

Solution: update asynchronously at each node $i \neq 1$

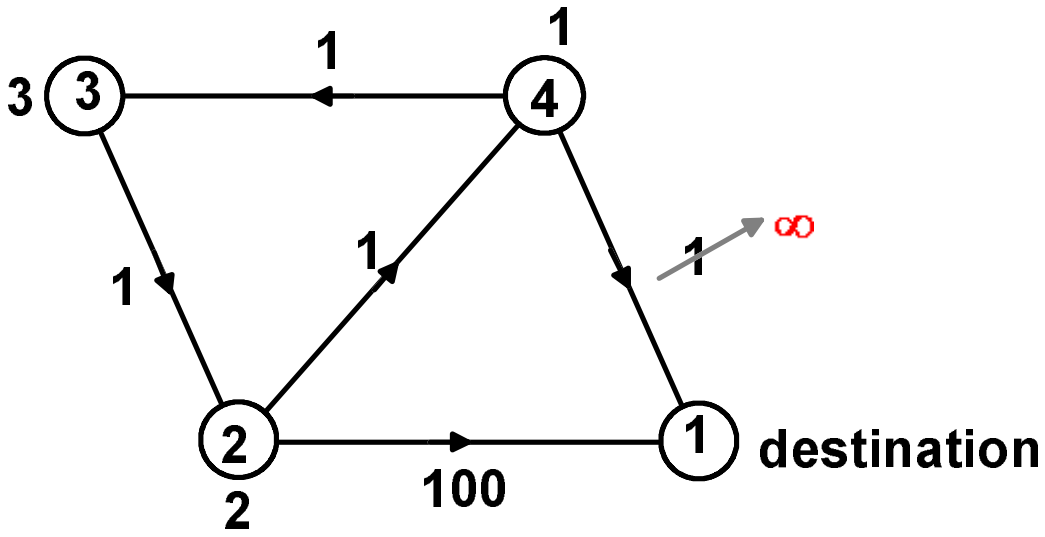
$$D_i = \min_{j \in N(i)} [d_{ij} + D_j]$$

where D_j **is the latest estimate from** j

Reason it works: after each update, D_i **is the distance from** i **to 1 on some path (provided initial conditions are** $D_i = \infty$ **for** $i \neq 1$, $D_1 = 0$**)**

Problem 2: In practice, link lengths d_{ij} occasionally change

Solution: just keep iterating asynchronous algorithm with new lengths $D_i = \min_{j \in N(i)} [d_{ij} + D_j]$



Before link (4,1) breaks, $D_4 = 1, D_2 = 2, D_3 = 3$
After node 4 notices that $d_{41} = \infty$**, iterations start until convergence achieved.**

Note: bad news can travel slowly

Lemma: The asynchronous algorithm

$$D_i = \min_{j \in N(i)} [d_{ij} + D_j] \quad d_{ij}$$ **'s fixed**

converges to the true shortest distances if the initial conditions are:
(a) $D_i = \infty$ **for** $i \neq 1$**,** $D_1 = 0$ **("infinite initial conditions")**
or (b) $D_i = 0$ **for all** i **("zero initial conditions")**

Theorem: consider the asynchronous B-F algorithm
Let D_i **be the value at any time with some arbitrary initial conditions.**
Let $\overline{D_i}$ **be the value that we would have if initial conditions where "infinite"**
Let $\underline{D_i}$ **be the value that we would have if initial conditions where "zero". Then**

$$\underline{D_i}^{(k)} \leq \underline{D_i}^{(k+1)} \leq D_i \leq \overline{D_i}^{(k+1)} \leq \overline{D_i}^{(k)}$$

Since $\underline{D_i}^{(k)} \uparrow$ **and** $\overline{D_i}^{(k)} \downarrow$ **converge to correct distances,** D_i **also converges to correct distances.**

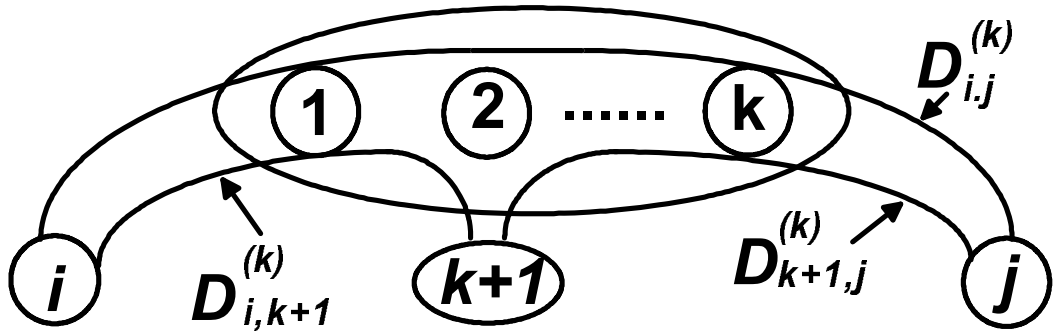
Floyd-Warshall Algorithm

Calculates the shortest distances D_{ij} for all source-destination pairs (i,j)

$D_{i,j}^{(k)}$ = shortest distance from i to j using only nodes in $\{1,2,3,...,K\}$ as intermediate nodes

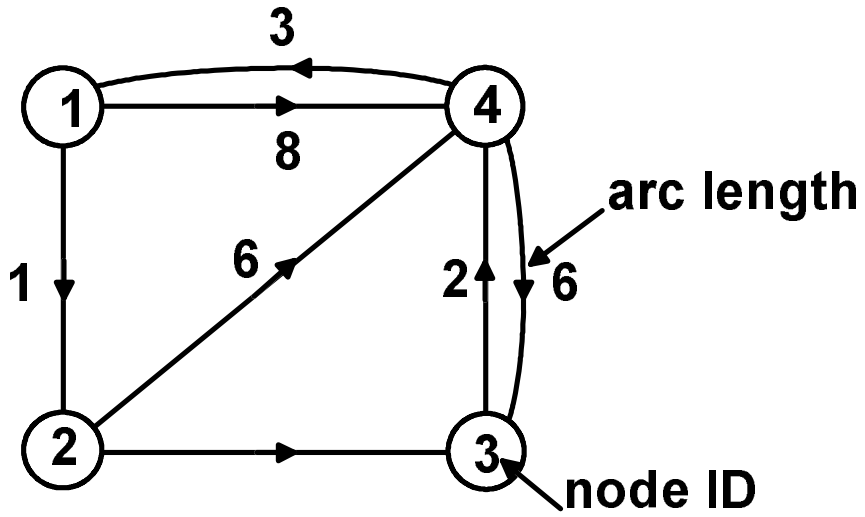
$$D_{i,j}^{(o)} = d_{ij} \quad \forall (i,j), i \neq j$$

$$D_{i,j}^{(k+1)} = \min\{D_{i,j}^{(k)}, D_{i,k+1}^{(k)} + D_{k+1,j}^{(k)}\}, \forall i,j$$



$D_{i,j}^{(N)}$ are the final distances (N # of nodes)

Example (Floyd-Warshall)



0 intermediate nodes

i \ j	1	2	3	4
1	0	1	∞	8
2	∞	0	1	6
3	∞	∞	0	2
4	3	∞	6	0

$D_{i,j}^{(0)}$

$$D_{i,j}^{(k+1)} = \min\{D_{i,j}^{(k)}, D_{i,k+1}^{(k)} + D_{k+1,j}^{(k)}\}$$

intermediate nodes={1}

i \ j	1	2	3	4
1	0	1	∞	8
2	∞	0	1	6
3	∞	∞	0	2
4	3	4	6	0

$D_{i,j}^{(1)}$

$D_{i,j}$ **has changed**

intermediate nodes={1,2}

i \ j	1	2	3	4
1	0	1	2	7
2	∞	0	1	6
3	∞	∞	0	2
4	3	4	5	0

$D_{i,j}^{(2)}$

$$D_{i,j}^{(k+1)} = \min\{D_{i,j}^{(k)}, D_{i,k+1}^{(k)} + D_{k+1,j}^{(k)}\}$$

intermediate nodes={1,2,3}

i \ j	1	2	3	4
1	0	1	2	4
2	∞	0	1	3
3	∞	∞	0	2
4	3	4	5	0

$D_{i,j}^{(3)}$

intermediate nodes={1,2,3,4}

i \ j	1	2	3	4
1	0	1	2	4
2	7	0	1	3
3	5	6	0	2
4	3	4	5	0

$D_{i,j}^{(4)}$ **(final shortest distances)**

Optimal Routing

A rational approach is to try to minimize the expected number of packets in the entire network, By Little's low, this also minimizes the average packet delay.

Changing the routing of a session changes:

- **The congestion in each link, thus changing are lengths in shortest path calculations**
- **The delay seen by other sessions.**

This suggests that:

- **Routing should be as a global problem**
- **Algorithm should make small changes rather than large changes.**

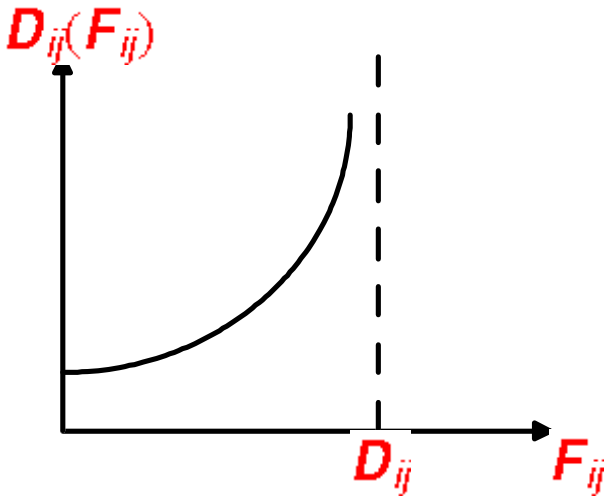
If we use Kleirock independence approximation, the expected number of packets queued or being served at arc *(i,j)*

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij}-F_{ij}} + d_{ij}F_{ij}$$

where F_{ij}= expected data rate on *(i,j)*

C_{ij}= capacity of *(i,j)*

d_{ij}= propagation and processing delay



Our problem then is to choose to minimize $\sum_{i,j} D_{ij}(F_{ij})$ subject to constraints on F_{ij}

Notation

W= **the set of all sessions**

P_w= **the set of all paths for session** $w \in W$

r_w= **the data rate for session** $w \in W$

X_p= **the data rate of session** w **sent over path** $p \in P_w$

Constraints: $X_p \geq 0$ **for all** $p \in P_w$

$$\sum_{p \in P_w} X_p = r_w \text{ for all } w \in W$$