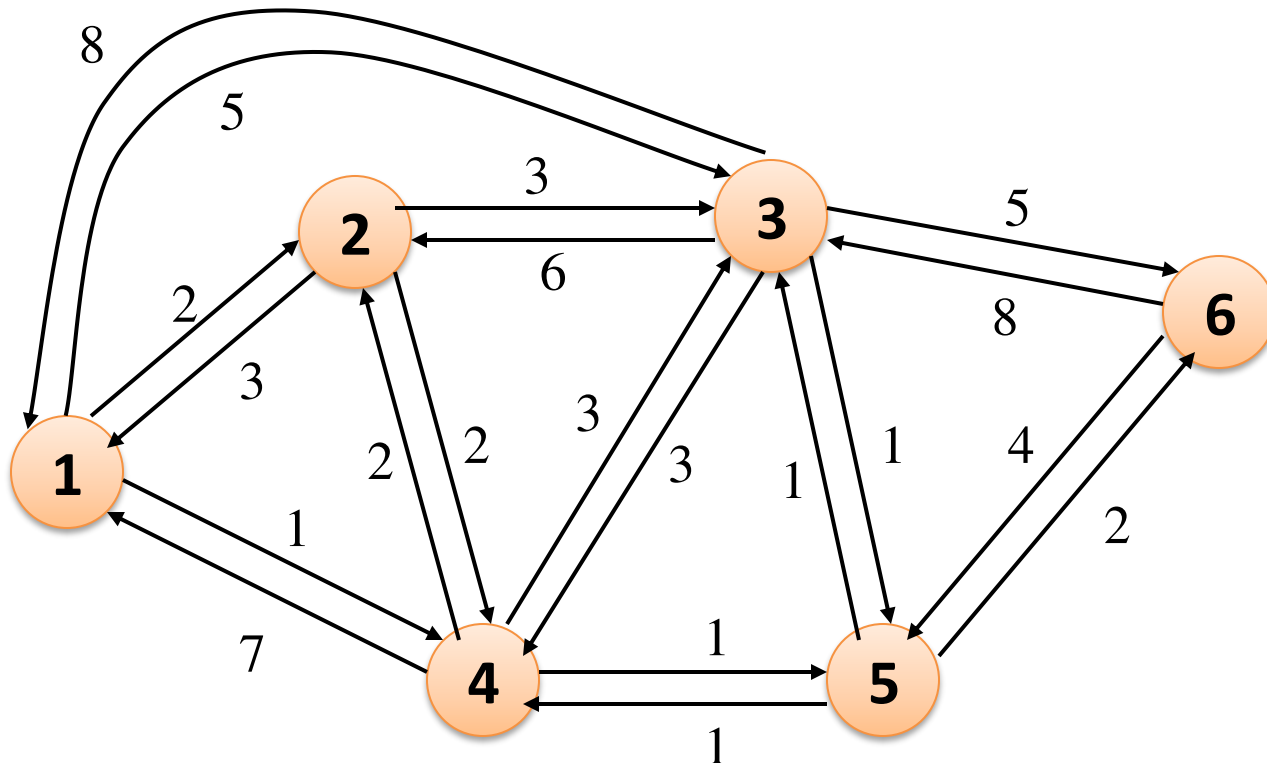
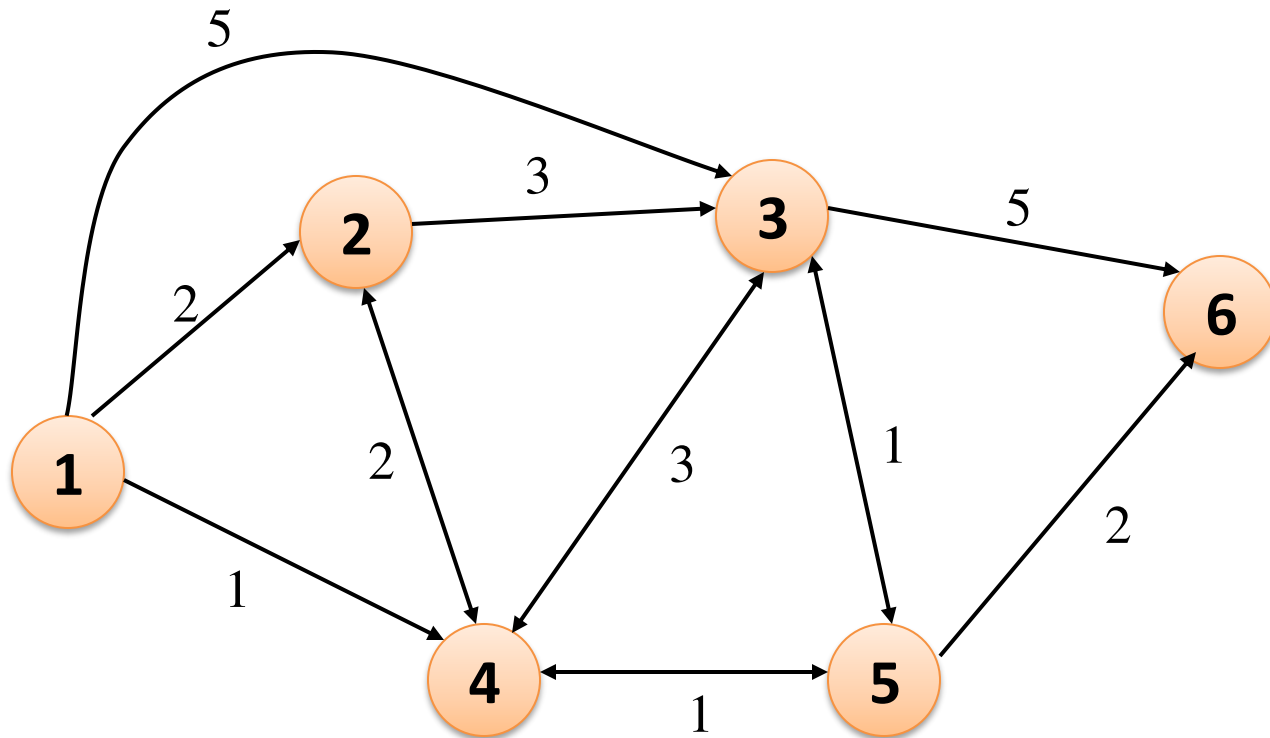


Shortest Path Algorithm

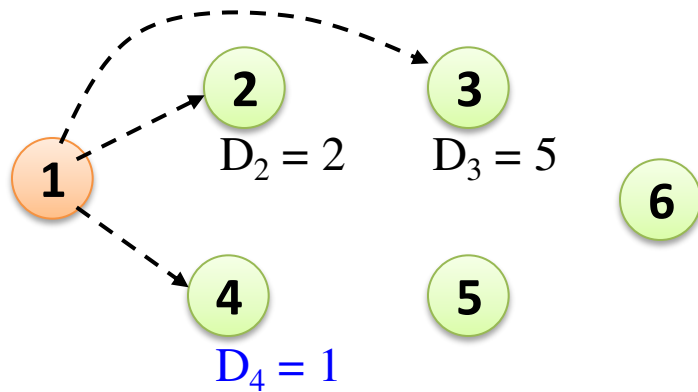
- **Dijkstra's Algorithm**
- **Bellman-Ford Algorithm**



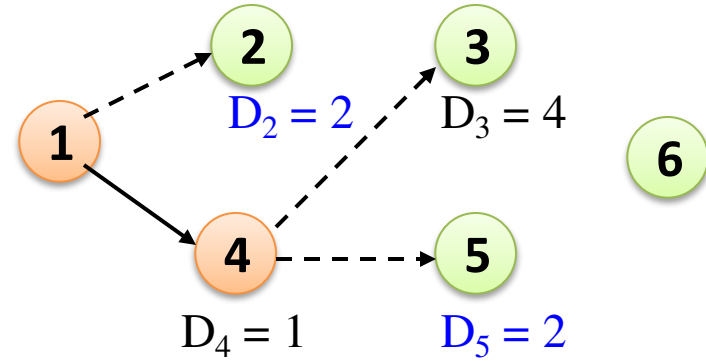
- **Reduced graph**



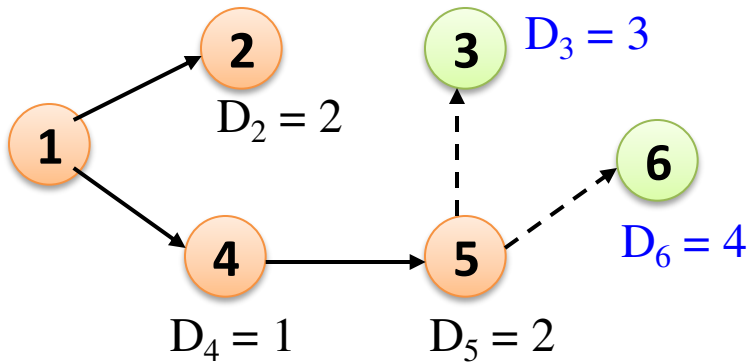
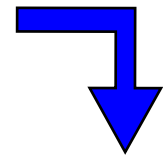
Dijkstra's Algorithm



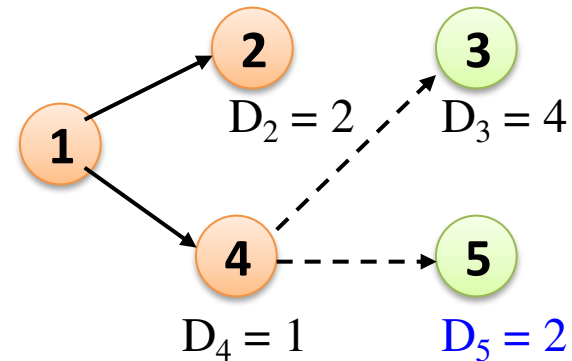
$T = \{1\}$



$T = \{1, 4\}$



$T = \{1, 2, 4, 5\}$



$T = \{1, 2, 4\}$



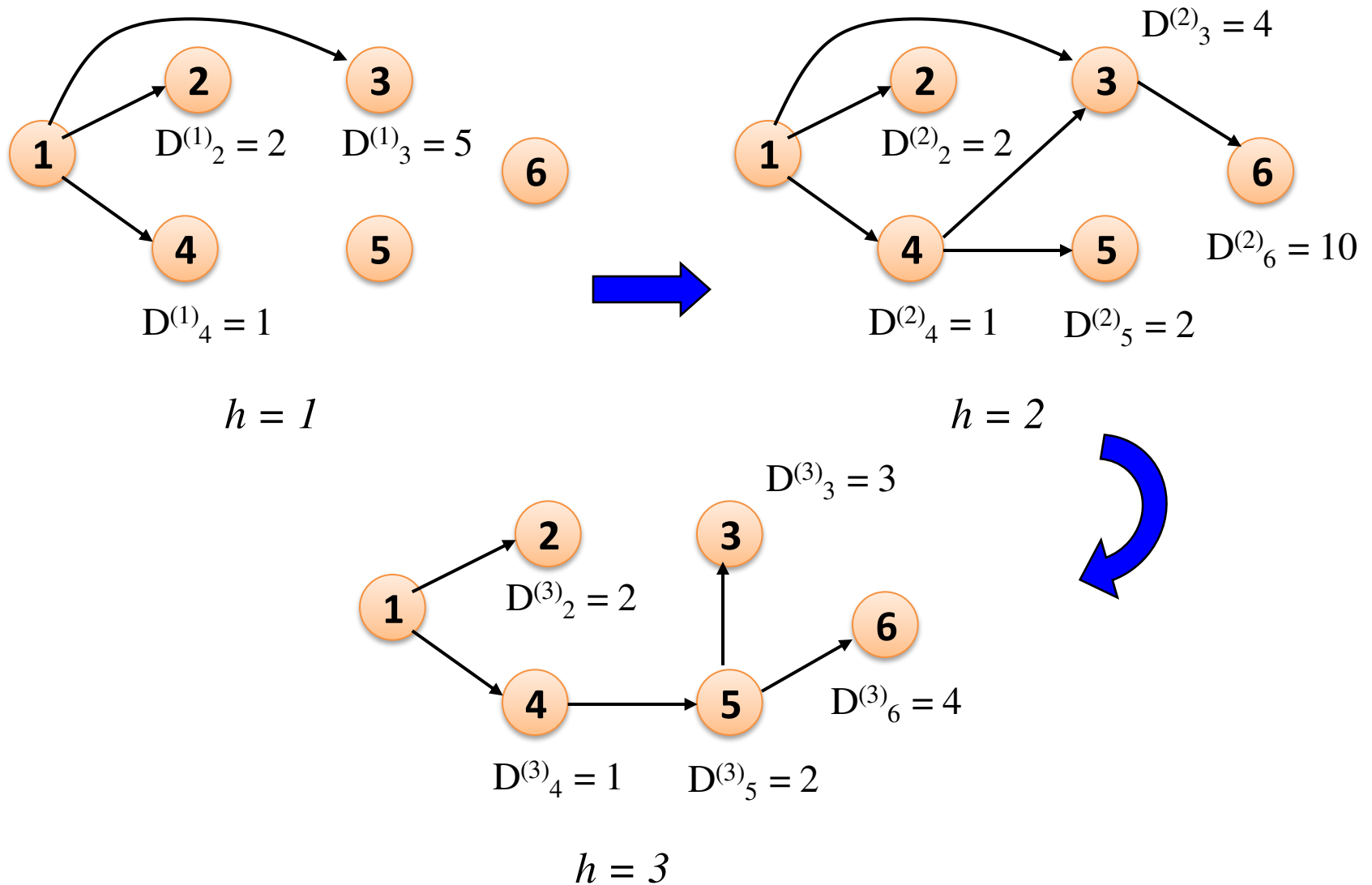
Dijkstra's Algorithm (cont)

		Node				
T	D					
		2	3	4	5	6
	1	2	5	<u>1</u>	∞	∞
	1, 4	<u>2</u>	4	1	2	∞
	1, 4, 2	2	4	1	<u>2</u>	∞
	1, 4, 2, 5	2	<u>3</u>	1	2	4
	1, 4, 2, 5, 3	2	3	1	2	<u>4</u>
	1, 4, 2, 5, 3, 6	2	3	1	2	4

Dijkstra's Algorithm (cont)

- $w(i, j)$ = link cost, $L(n)$ = path cost from node s to n
- 1. *[Initialization]*
 - $T = \{s\}$
 - $L(n) = w(s, n)$ for $n \neq s$
- 2. *[Get next node]*
 - Find $x \notin T$ such that $L(x) = \min_{j \notin T} L(j)$
 - Add x to T
- 3. *[Update Least-Cost Paths]*
 - $L(n) = \min [L(n), L(x) + w(x, n)]$ for all $n \notin T$
 - Go to step 2

Bellman-Ford Algorithm

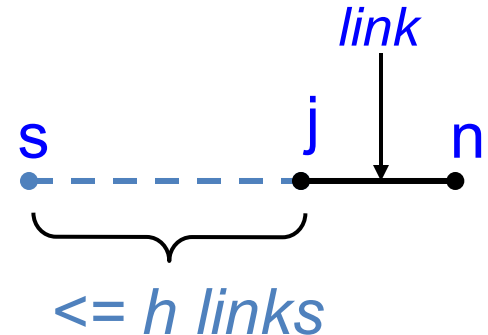


Bellman-Ford Algorithm (cont)

		Node				
		D				
		h				
		2	3	4	5	6
Source = 1	0	∞	∞	∞	∞	∞
	1	2	5	1	∞	∞
	2	2	4	1	2	10
	3	2	3	1	2	4
	4	2	3	1	2	4

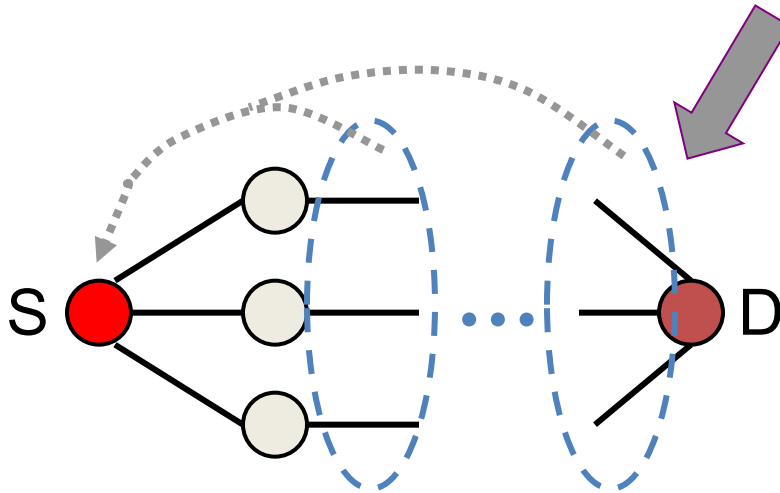
Bellman-Ford Algorithm (cont)

- $L_h(n)$ = path cost from s to n w/ no more than h links
- 1. *[Initialization]*
 - $L_0(n) = \infty$, for all $n \neq s$
 - $L_h(s) = 0$, for all h
- 2. *[Update]*
 - For each successive $h \geq 0$
 - For each $n \neq s$, compute
 - $L_{h+1}(n) = \min_j [L_h(j) + w(j, n)]$

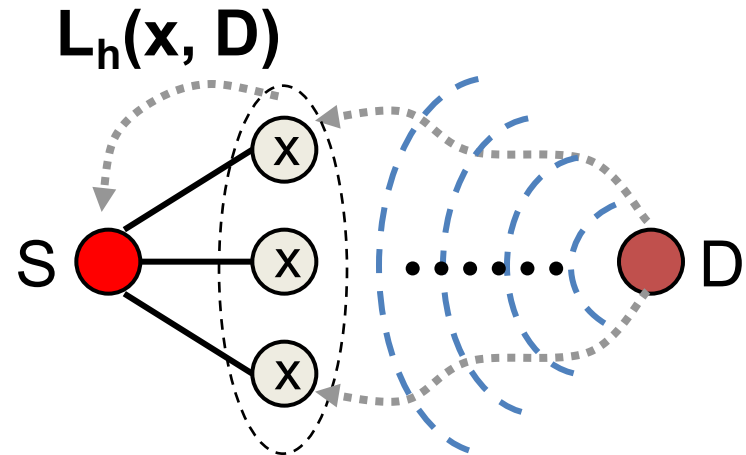


Comparisons

$$L(n) = \min [L(n), L(x) + \underbrace{w(x, n)}]$$



Dijkstra's
(Link State)



Bellman-Ford
(Distance Vector)

Bellman-Ford vs. Dijkstra

- Results from two algorithms agree
- **Bellman-Ford**
 - Calculation for *node n* needs link cost to neighbouring nodes plus total cost to each neighbour from *s*
 - Each node can maintain set of costs and paths for every other node.
 - Can exchange information with direct neighbours
 - Can update costs and paths based on information from neighbours and knowledge of link costs
- **Dijkstra**
 - Each node needs complete topology
 - Must know link costs of all links in network
 - Must exchange information with all other nodes

Routing in ARPANET

- First generation(RIP), 1969
 - Adaptive Routing is adopted
 - Use Bellman-Ford algorithm → Distance Vector
 - Estimated link delay is simply the queue length for that link
 - Every 128ms, each node exchanges its delay vector (routing table) with all its neighbors
 - Information about a change in network condition would gradually ripple through the network

Routing in ARPANET (cont)

- Each node i maintains

- d_{ij} = current estimate of min delay from i to j
- s_{ij} = next node in the current min-delay route from i to j

- Node k updates its vectors as follows

- $d_{kj} = \text{Min} [l_{ki} + d_{ij}]$
 $i \in A$



- $s_{kj} = i$ using i that minimizes the expression above
where

A = set of neighbor nodes for k

l_{ki} = current estimate of delay from k to i

Routing in ARPANET (cont)

- **Major shortcomings of RIP**
 - It did not consider line speed, merely queue length. Higher capacity links were not given the favored status
 - Queue length is an artificial measure of delay
 - The algorithm was not very accurate. It responded slowly to congestion and delay increases.

Routing in ARPANET (cont)

- **Second generation, 1979**
 - Using **Dijkstra's** algorithm
 - **Link State Routing Protocol**
 - **OSPF**: Open Shortest Path First protocol
 - The **delay is measured directly**
 - Every 10 seconds, the node computes the **average delay** on each outgoing link
 - Information of changes in delay is sent to all others nodes using **flooding**

Routing in ARPANET (cont)

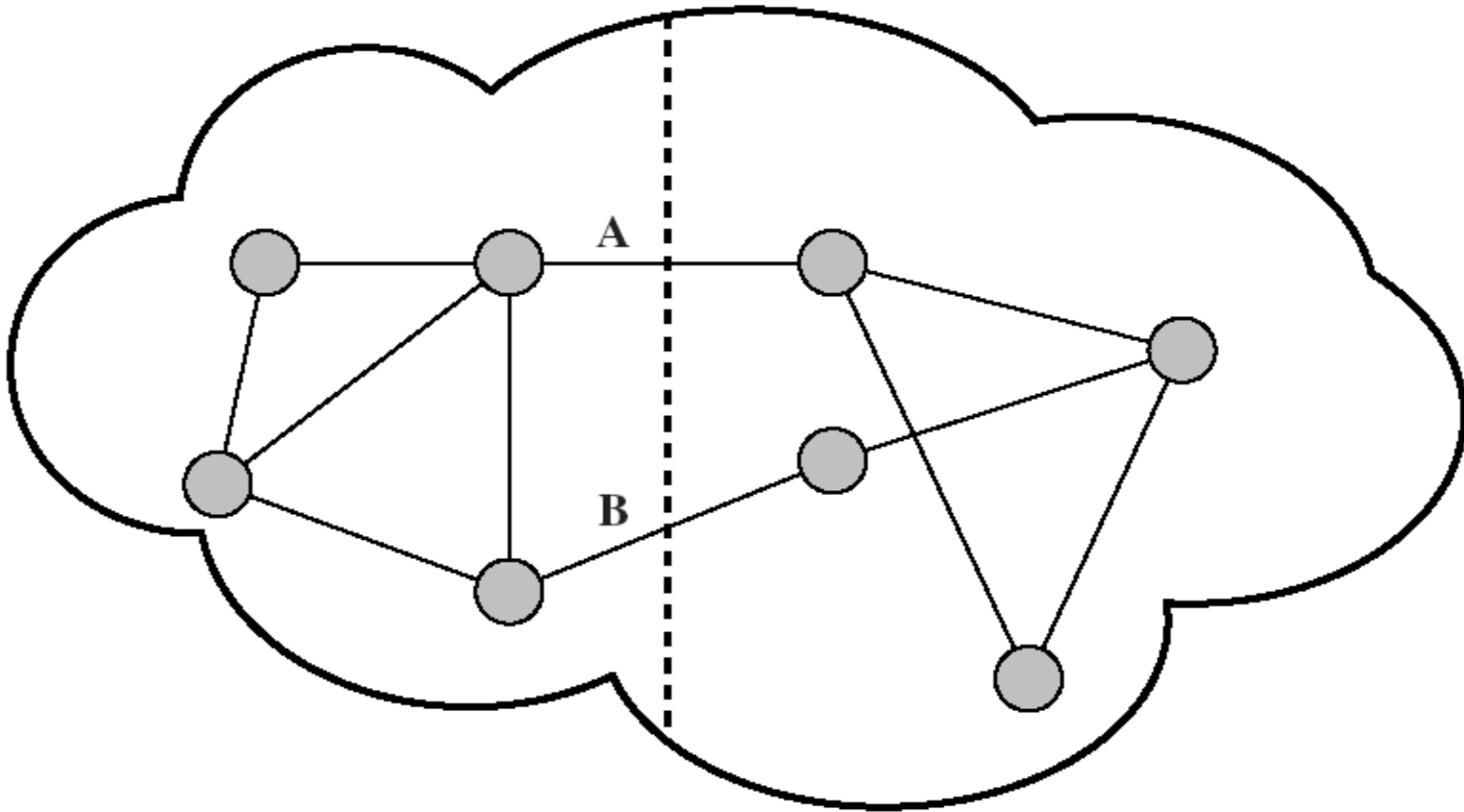


Figure 10.12 Packet-Switching Network Subject to Oscillations

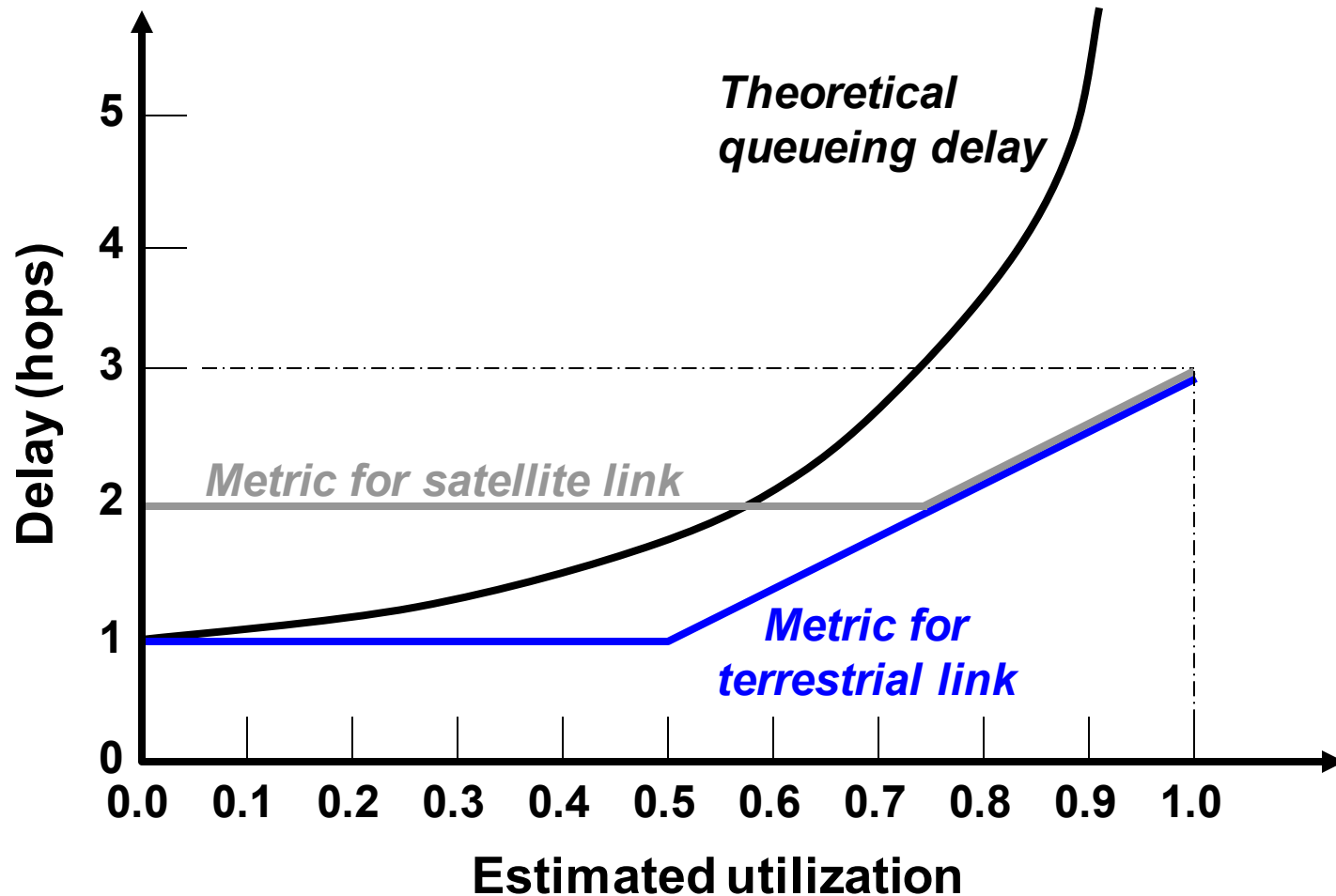
Routing in ARPANET (cont)

- **Third generation, 1987**
 - **Problem**
 - The correlation between the reported values (delay) and those actually experienced after rerouting
 - **Conclusion**
 - Under heavy load, the goal of routing should be to give the average route a good path instead of attempting to give all routes the best path
 - **Solution**
 - Also consider the **average utilization** of links
 - *Revised cost function:*
 - **delay-based** metric **under light loads**
 - **capacity-based** metric **under heavy loads**

Calculate Link Costs

1. Measure the avg. delay over the last 10 sec
2. Using the single-server queuing model (M/G/1), the measured delay is transformed into an estimate of link utilization
3. Average the link utilization $\rho(n+1)$ with the previous estimate of utilization $U(n)$
 - $U(n+1) = 0.5 * \rho(n+1) + 0.5 * U(n)$
4. The link cost is set as a function of average utilization

ARPANET Delay Metrics (3rd)



ARPANET Delay Metrics (3rd)

1. **Delay is normalized** to the value achieved on an idle line
2. The cost value is **kept at the minimum value** until a given level of utilization is reached
 - Reducing routing overhead at low traffic levels
3. Above a certain level of utilization, the cost level is allowed to rise to **a maximum value that is equal to three times the minimum value**
 - To dictate that traffic should not be routed around a heavily utilized line by more than two additional hops