第12章　交换网络中的路由选择

南京大学计算机系　　黄皓教授

2007年9月30日 星期五

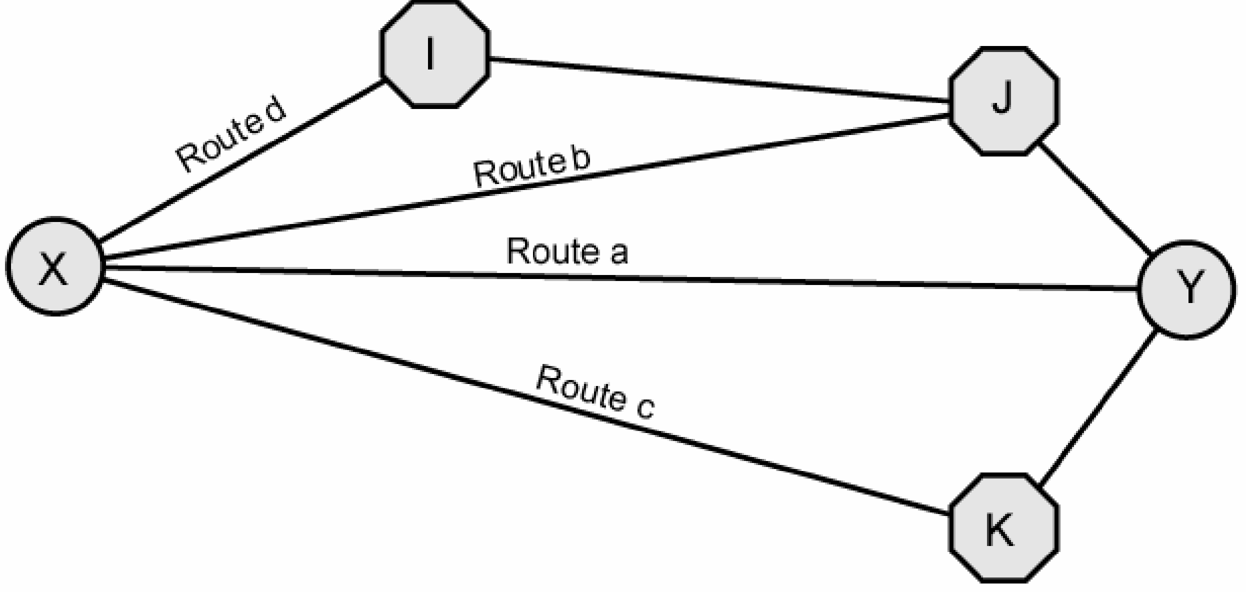
12.1 Routing in Circuit Switched Network

- Many connections will need paths through more than one switch
- Need to find a route
  - Efficiency
  - Resilience
- Static routing
  - Public telephone switches are a tree structure
  - Static routing uses the same approach all the time
- Dynamic routing allows for changes in routing depending on traffic
  - Uses a peer structure for nodes

Alternate Routing

- Possible routes between end offices predefined
- Originating switch selects appropriate route
- Routes listed in preference order
- Different sets of routes may be used at different times

Alternate Routing Diagram



Route a:	X® Y
Route b:	X® J® Y
Route c:	X® K® Y
Route d:	X® I® J® Y

◯ = end office

⯃ = intermediate switching node

(a) Topology

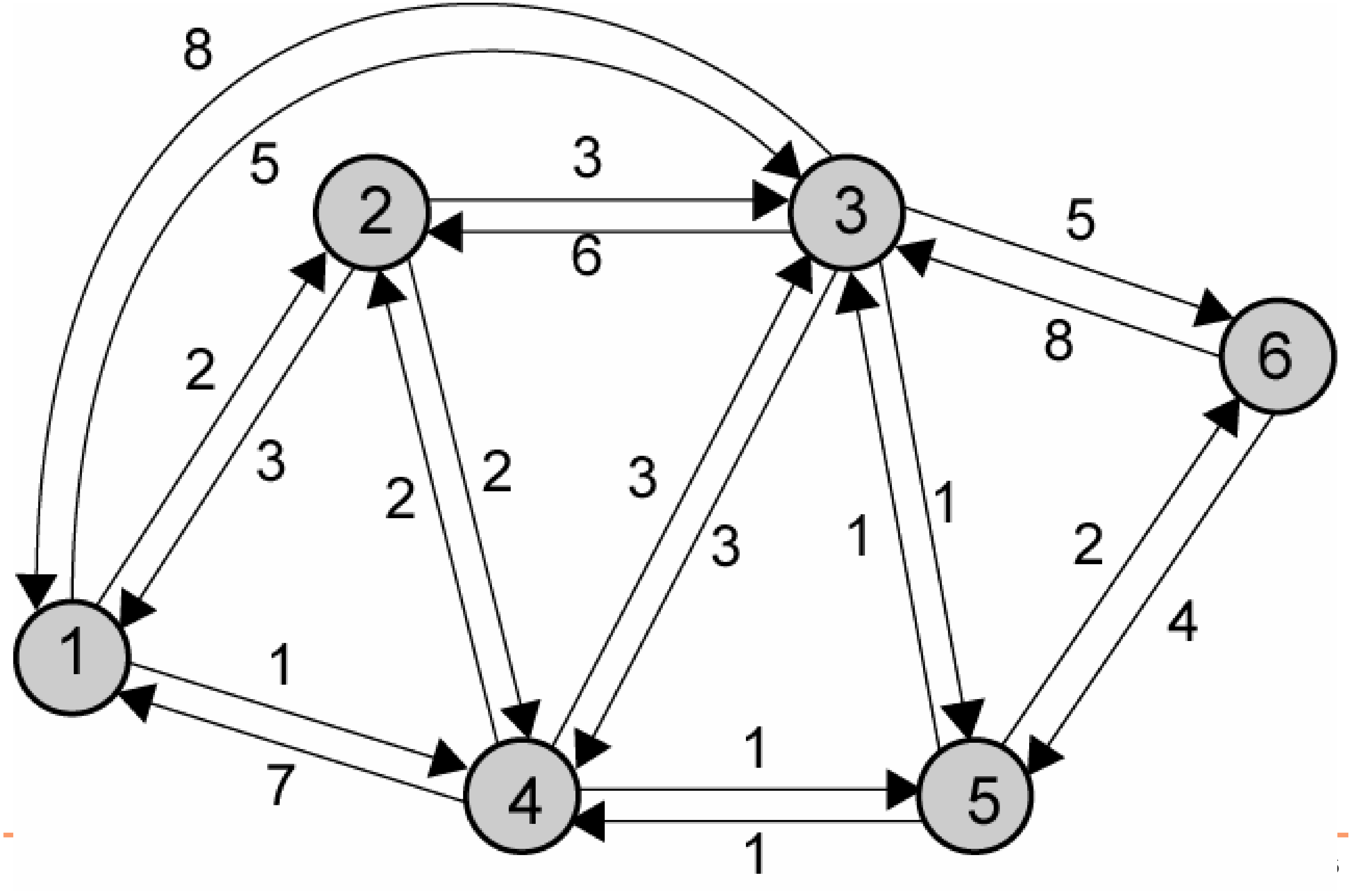
Time Period	First route	Second route	Third route	Fourth and final route
Morning	a	b	c	d
Afternoon	a	d	b	c
Evening	a	d	c	b
Weekend	a	c	b	d

(b) Routing table

12.2 Routing in Packet Switched Network

- Complex, crucial aspect of packet switched networks
- Characteristics required
  - Correctness
  - Simplicity
  - Robustness
  - Stability
  - Fairness
  - Optimality
  - Efficiency

Example Packet Switched Network

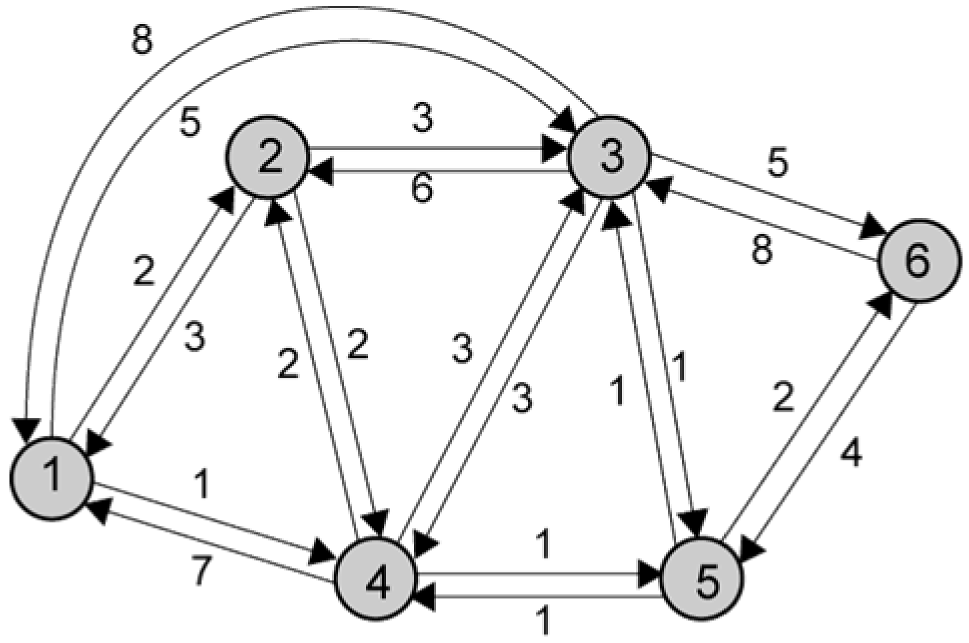


12.2.1 Elements of Routing

- Performance criteria
- Decision time
- Decision place
- Network info source
- Network info update timing

Performance Criteria

- Minimum hop
  - e.g. 1–3–6
- Least cost
  - e.g. 1–4–5–6
- Others
  - Minimum delay
  - Largest throughput



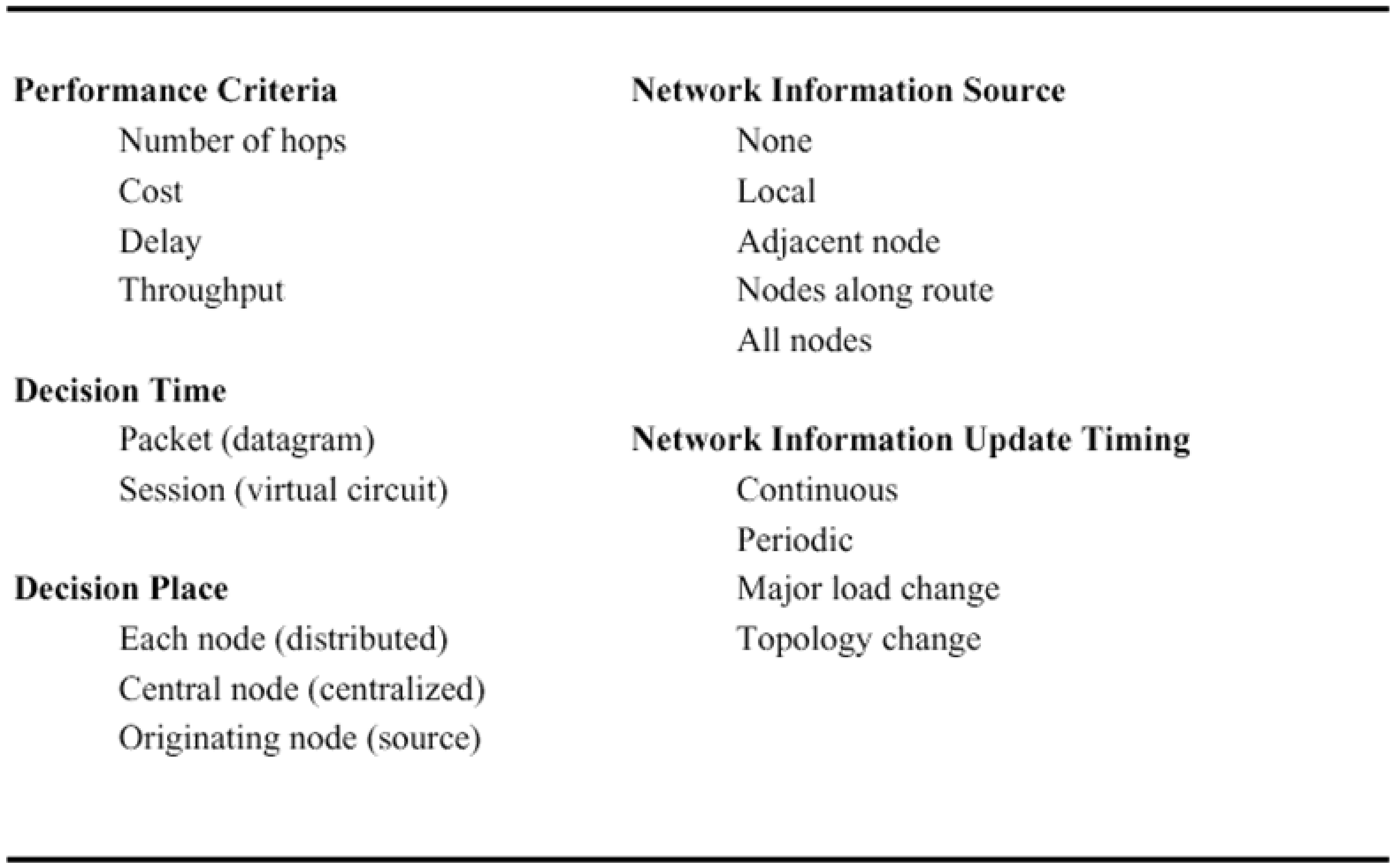
Decision Time and Place

- Time
  - Packet
  - virtual circuit
- Place
  - Distributed
    - Made by each node
  - Centralized
  - Source

Network Information Source and Update Timing

- Routing decisions usually based on knowledge of network (not always)
- Distributed routing
  - Nodes use local knowledge
  - May collect info from adjacent nodes
  - May collect info from all nodes on a potential route
- Central routing
  - Collect info from all nodes
- Update timing
  - When is network info held by nodes updated
  - Fixed - never updated
  - Adaptive - regular updates

Conclusion of Routing Elements



Performance Criteria
- Number of hops
- Cost
- Delay
- Throughput

Decision Time
- Packet (datagram)
- Session (virtual circuit)

Decision Place
- Each node (distributed)
- Central node (centralized)
- Originating node (source)

Network Information Source
- None
- Local
- Adjacent node
- Nodes along route
- All nodes

Network Information Update Timing
- Continuous
- Periodic
- Major load change
- Topology change

12.2.2 Routing Strategies

- Fixed
- Flooding
- Random
- Adaptive

(1) Fixed Routing

- Single permanent route for each source to destination pair
- Determine routes using a least cost algorithm
- Route fixed, at least until a change in network topology

(1) Fixed Routing tables

CENTRAL ROUTING DIRECTORY

		From Node					
		1	2	3	4	5	6
To Node	1	—	1	5	2	4	5
	2	2	—	5	2	4	5
	3	4	3	—	5	3	5
	4	4	4	5	—	4	5
	5	4	4	5	5	—	5
	6	4	4	5	5	6	—

Node 1 Directory

Destination	Next Node
2	2
3	4
4	4
5	4
6	4

Node 2 Directory

Destination	Next Node
1	1
3	3
4	4
5	4
6	4

Node 3 Directory

Destination	Next Node
1	5
2	5
4	5
5	5
6	5

Node 4 Directory

Destination	Next Node
1	2
2	2
3	5
5	5
6	5

Node 5 Directory

Destination	Next Node
1	4
2	4
3	3
4	4
6	6

Node 6 Directory

Destination	Next Node
1	5
2	5
3	5
4	5
5	5

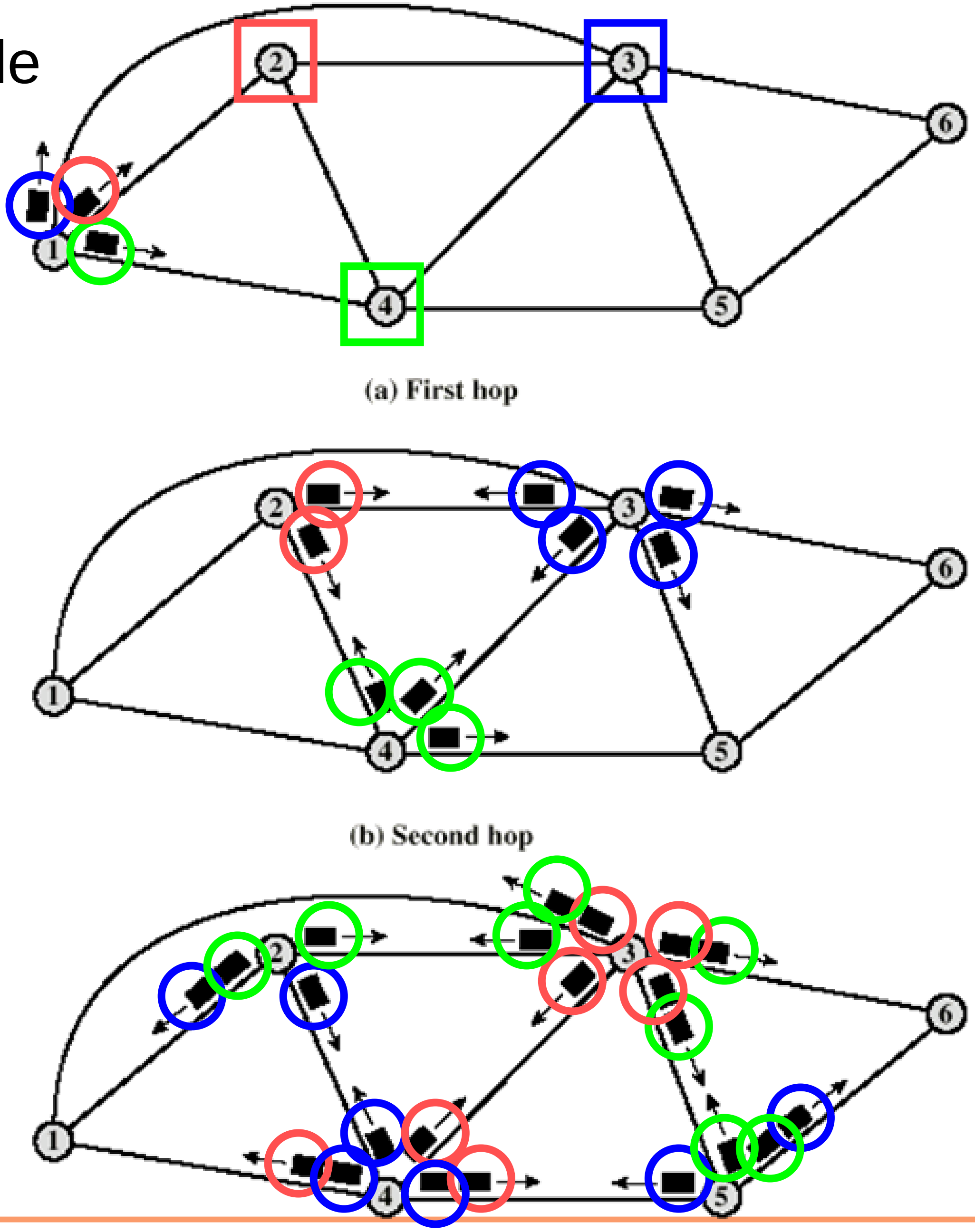
(2) Flooding

- No network info required
- Packet sent by node to every neighbor
  - Incoming packets retransmitted on every link except incoming link
- Eventually a number of copies will arrive at destination
- Each packet is uniquely numbered so duplicates can be discarded
- Nodes can remember packets already forwarded to keep network load in bounds
- Can include a hop count in packets

(2) Flooding Example

Hop count = 3

- Initial
  - 3 packets
- 1st hop
  - 9 packets
- 2nd hop
  - 23 packets



(a) First hop

(b) Second hop

(c) Third hop

(2) Properties of Flooding

- All possible routes are tried
  - Very robust
- At least one packet will have taken minimum hop count route
  - Can be used to set up virtual circuit
- All nodes are visited
  - Useful to distribute information (e.g. routing)

(3) Random Routing

- Node selects one outgoing path for retransmission of incoming packet
- Selection can be random or round robin
- Can select outgoing path based on probability calculation
- No network info needed
- Route is typically not least cost nor minimum hop

(3) Random Routing

Assign Probabilities

- $P_i = R_i / \sum_j R_j$
  - P_i – Probability of selecting out-link i
  - R_i – Cost factor of link i
- Possible cost factor
  - Transmission rate – for throughput
  - Queue size – for delay
- Partly used in adaptive routing

Adaptive Routing

- Used by almost all packet switching networks
- Routing decisions change as conditions on the network change
  - Failure
  - Congestion
- Requires info about network
- Decisions more complex
- Tradeoff between quality of network info and overhead
- Reacting too quickly can cause oscillation
- Too slowly to be relevant

Adaptive Routing - Advantages

- Improved performance
- Aid congestion control
- Complex system
  - May not realize theoretical benefits

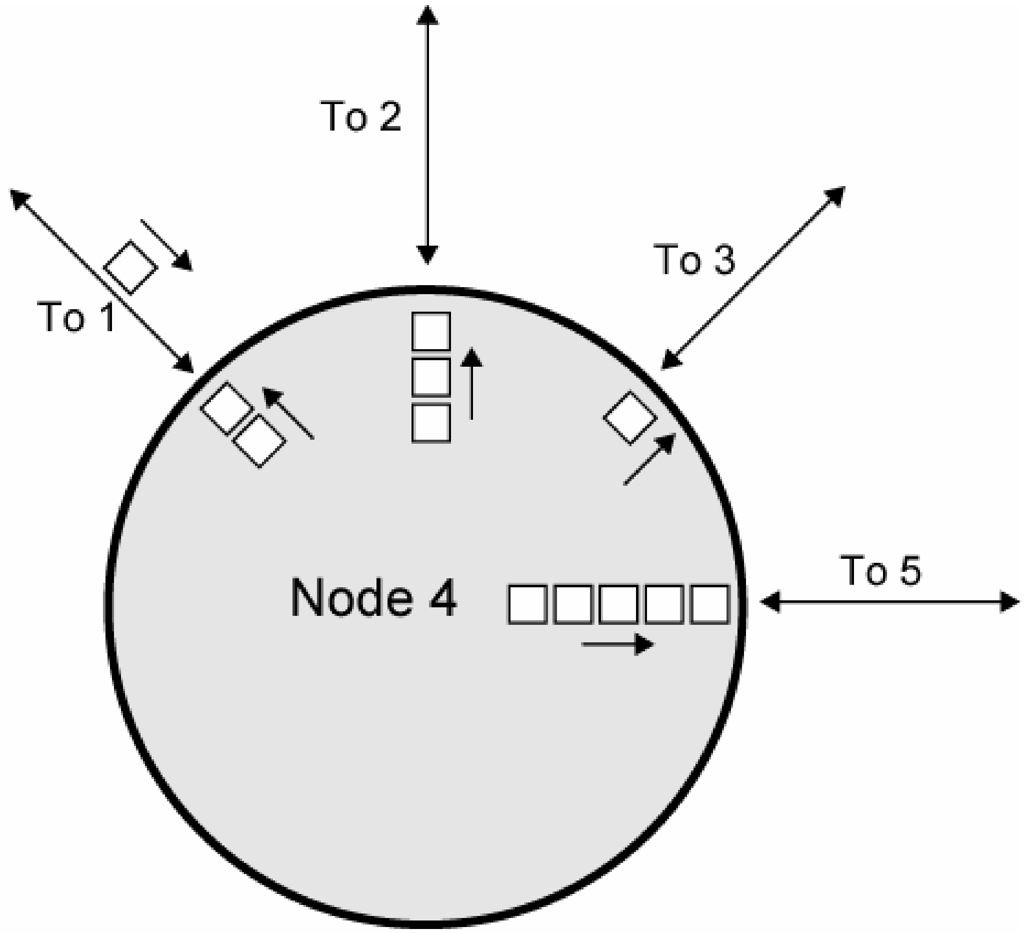
Classification

- Based on information sources
  - Local (isolated)
    - Route to outgoing link with shortest queue
    - Can include bias for each destination
    - Rarely used - do not make use of easily available info
  - Adjacent nodes
  - All nodes

Isolated Adaptive Routing

Node 4's Bias
Table for
Destination 6

Next Node	Bias
1	9
2	6
3	3
5	0



12.3 Least Cost Algorithms

- Given network of nodes connected by bi-directional links
- Each link has a cost in each direction
  - Each link cost 1 – **minimum hop count**
  - Link cost inversely proportional to capacity – **maximum throughput**
- Link costs in different directions may be different
  - e.g. length of packet queue
- Define cost of path between two nodes as sum of costs of links traversed
- For each pair of nodes, find a path with the least cost

2 Algorithms in Adaptive Routing

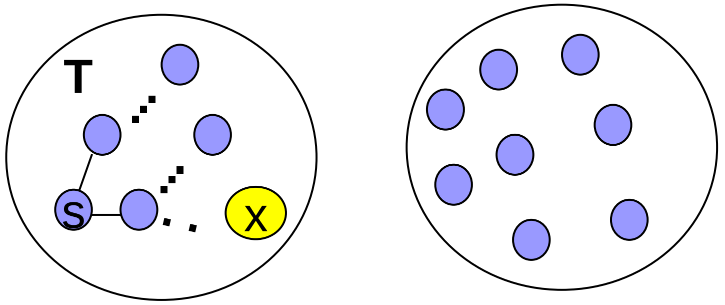
- Dijkstra's Algorithm
- Bellman-Ford Algorithm

Dijkstra's Algorithm

- Find shortest paths from given source to all other nodes
  - Developing paths in order of increasing path cost (length)
- N = set of nodes in the network
- s = source node
- T = set of nodes so far incorporated by the algorithm
- $w(i, j)$ = link cost from node i to node j
  - $w(i, i) = 0$
  - $w(i, j) = \infty$ if the two nodes are not directly connected
  - $w(i, j) \geq 0$ if the two nodes are directly connected
- $L(n)$ = cost of least-cost path from node s to node n currently known
  - At termination, $L(n)$ is cost of least-cost path from s to n

Dijkstra's Algorithm Method

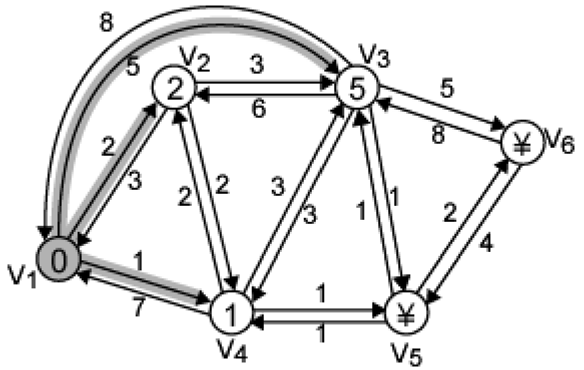
- Step 1 [Initialization]
  - $T = \{s\}$ Set of nodes so far incorporated consists of only source node
  - $L(n) = w(s, n)$ for $n \neq s$
  - Initial path costs to neighboring nodes are simply link costs
- Step 2 [Get Next Node]
  - Find node x not in T with least-cost path from s (i.e. $\min L(x)$)
  - Incorporate node x into T
  - Also incorporate the edge that links x with the node in T that contributes to the path
- Step 3 [Update Least-Cost Paths]
  - $L(n) = \min[L(n), L(x) + w(x, n)]$ for all $n \notin T$
  - If latter term is minimum, path from s to n is path from s to x concatenated with link from x to n
- Algorithm terminates when all nodes have been added to T



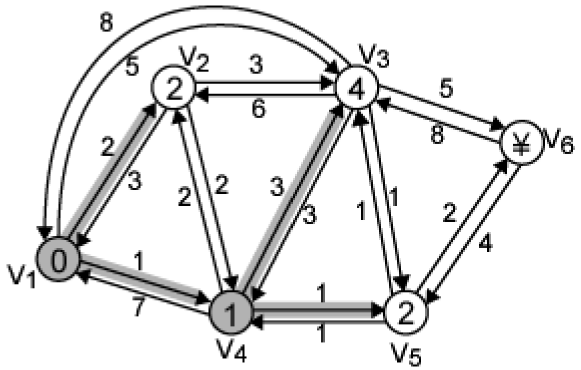
Dijkstra's Algorithm Notes

- One iteration of steps 2 and 3 adds one new node to T
  - Defines least cost path from s to that node
- At termination, value $L(n)$ associated with each node n is the cost (length) of least-cost path from s to n
- In addition, T defines the least-cost path from s to each other node

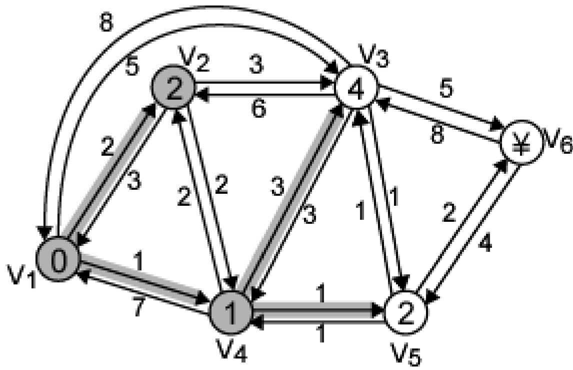
Example of Dijkstra's Algorithm



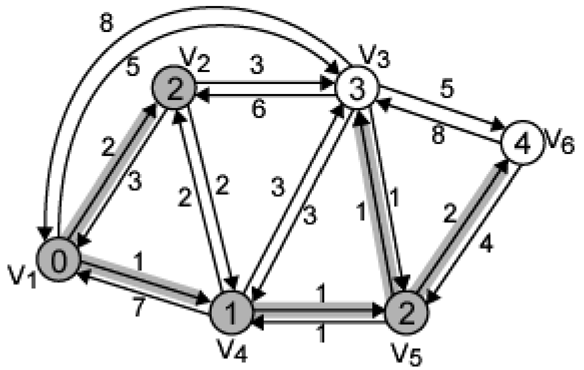
T = {1}



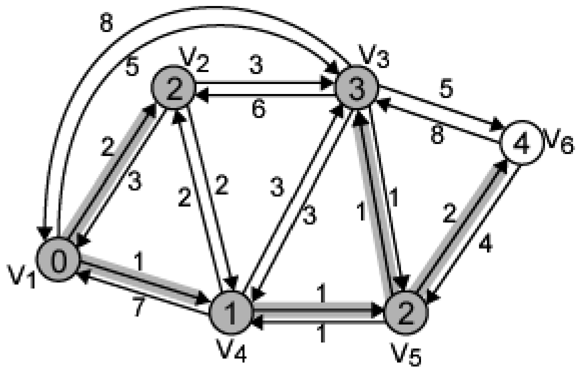
T = {1, 4}



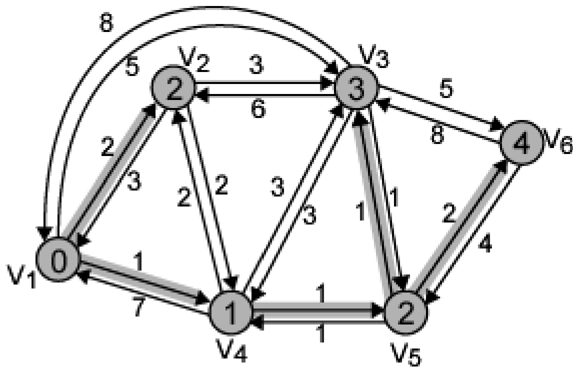
T = {1, 2, 4}



T = {1, 2, 4, 5}



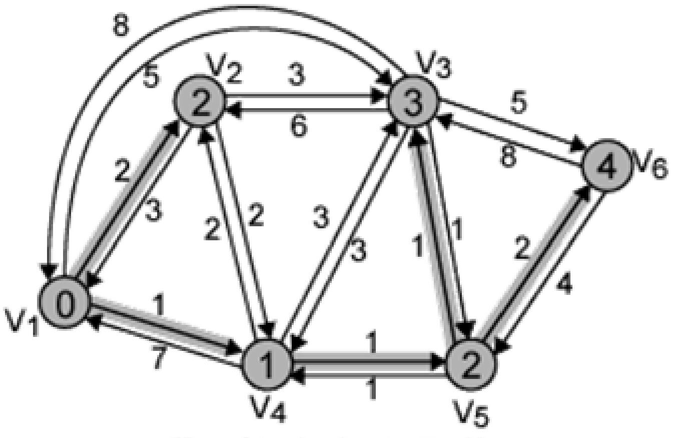
T = {1, 2, 3, 4, 5}



T = {1, 2, 3, 4, 5, 6}

Results of Example Dijkstra's Algorithm

No	T	L(2)	Path	L(3)	Path	L(4)	Path	L(5)	Path	L(6)	Path
1	{1}	2	1-2	5	1-3	1	1-4	∞	–	∞	–
2	{1,4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	–
3	{1, 2, 4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	–
4	{1, 2, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
5	{1, 2, 3, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
6	{1, 2, 3, 4, 5, 6}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6



T = {1, 2, 3, 4, 5, 6}

Destination	Next-Hop	Distance
2	2	2
3	4	3
4	4	1
5	4	2
6	4	4

Bellman-Ford Algorithm Definitions

- Find shortest paths from given node containing at most 1 link
- Find the shortest paths that containing at most 2 links, based on the result of 1 link
- Find the shortest paths of 3 links based on result of 2 links, and so on
- s = source node
- $w(i, j)$ = link cost from node i to node j
  - $w(i, i) = 0$
  - $w(i, j) = \infty$ if the two nodes are not directly connected
  - $w(i, j) \geq 0$ if the two nodes are directly connected
- h = maximum number of links in path at current stage of the algorithm
- $L_h(n)$ = cost of least-cost path from s to n under constraint of no more than h links

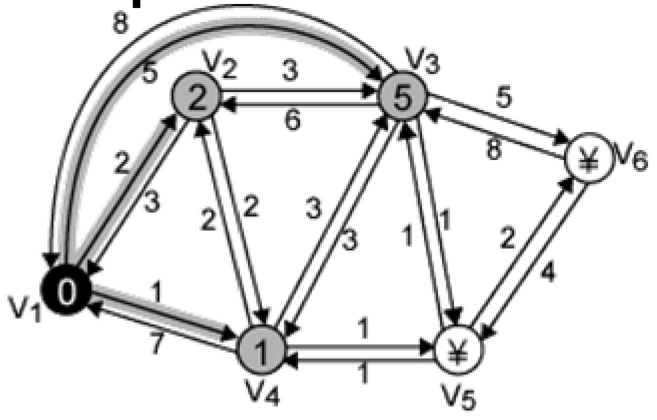
Bellman-Ford Algorithm Method

- Step 1 [Initialization]
  - $L_0(n) = \infty$, for all $n \neq s$
  - $L_1(n) = w(s, n)$
  - $L_h(s) = 0$, for all h
- Step 2 [Update]
  - For each successive $h \geq 0$
    - For each $n \neq s$, compute $L_{h+1}(n) = \min_j[L_h(j)+w(j,n)]$
  - Connect n with predecessor node j that achieves minimum
  - Eliminate any connection of n with different predecessor formed during earlier iterations
  - Path from s to n terminates with link from j to n
- Repeat until no change made to route (convergence)

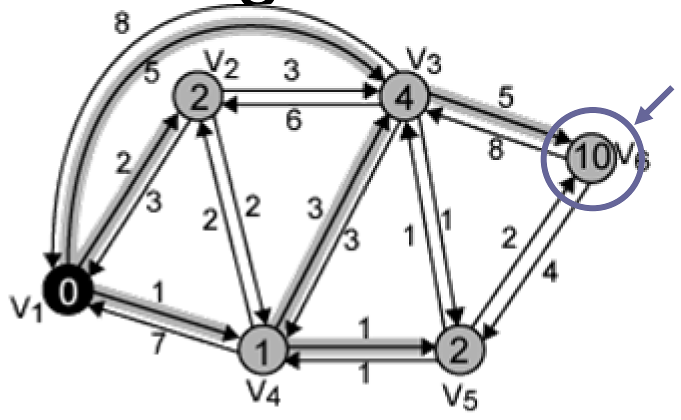
Bellman-Ford Algorithm Notes

- For each iteration with $h=K$ and for each destination node n
  - Compares newly computed paths from s to n of length K with path from previous iteration
- If previous path shorter it is retained
- Otherwise new path is defined

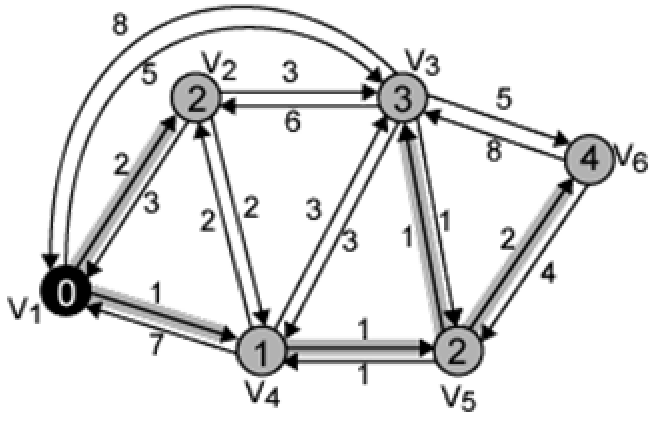
Example of Bellman-Ford Algorithm



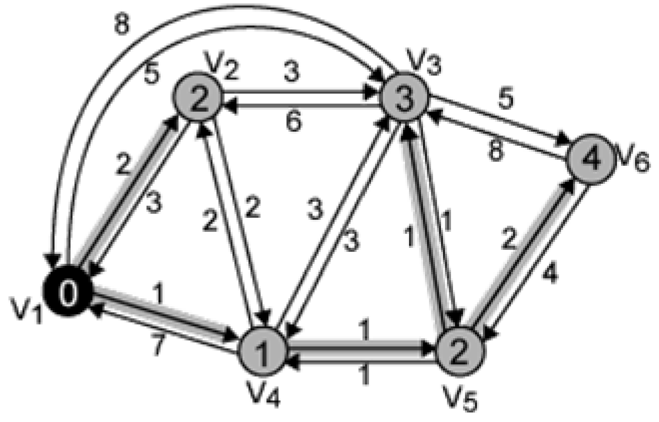
h = 1



h = 2



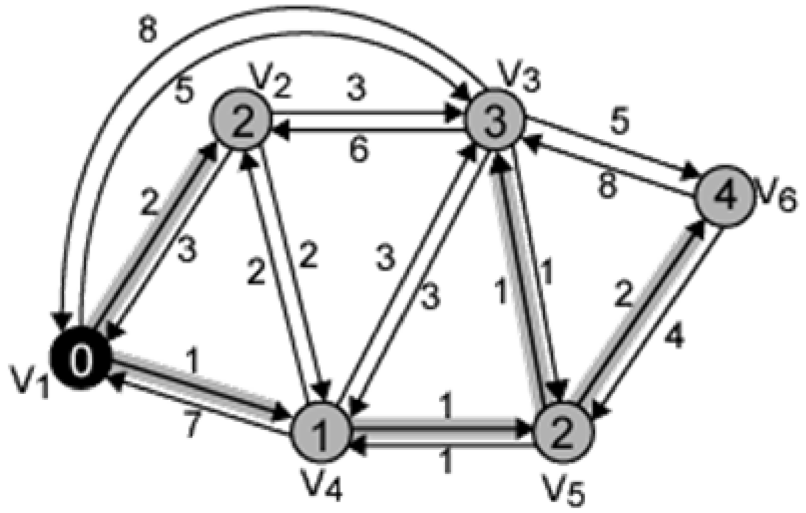
h = 3



h = 4

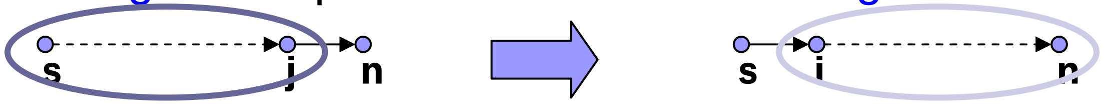
Results of Bellman-Ford Example

h	$L_h(2)$	Path	$L_h(3)$	Path	$L_h(4)$	Path	$L_h(5)$	Path	$L_h(6)$	Path
0	∞	–	∞	–	∞	–	∞	–	∞	–
1	2	1-2	5	1-3	1	1-4	∞	–	∞	–
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
4	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6



Comparison of Info Gathering

- Bellman-Ford
  - Calculation for node n involves knowledge of link cost to all neighbours plus total cost to each neighbour from s



  - Each node can maintain set of costs and paths to every other node
  - Can exchange info with direct neighbours
  - Update costs and paths based on info from neighbours and knowledge of direct link costs
- Dijkstra
  - Each node needs complete topology
  - Must know link costs of all links in network
  - Must exchange information with all other nodes

Comparison of Efficiency

- Evaluation
  - Dependent on processing time of algorithms
  - Dependent on amount of info required from other nodes
- Implementation specific
  - Both converge to same solution under static topology and costs
  - If link costs change, algorithms will attempt to catch up
  - If link costs depend on traffic, which depends on routes chosen, then feedback
  - Both will result in instability

12.4 ARPANET Routing Strategies

- First Generation, 1969
- Second Generation, 1979
- Third Generation, 1987

First Generation

- Method
  - Distributed adaptive, uses Bellman-Ford algorithm
  - Estimated delay using performance criterion per 128ms
  - Node exchanges delay vector with neighbors
  - Update routing table based on incoming info
- Problem
  - Doesn't consider line speed, just queue length (lines speed is not similar now)
  - Queue length not a good measurement of delay
  - Responds slowly to congestion

The 1st Routing Algorithm

Desti-nation	Delay	Next node
1	0	—
2	2	2
3	5	3
4	1	4
5	6	3
6	8	3
	$\underbrace{\qquad}_{D_1}$	$\underbrace{\qquad}_{S_1}$

(a) Node 1's Routing table before update

D_2	D_3	D_4
3	7	5
0	4	2
3	0	2
2	2	0
3	1	1
5	3	3

(b) Delay vectors sent to node 1 from neighbor nodes

Desti-nation	Delay	Next node
1	0	—
2	2	2
3	3	4
4	1	4
5	2	4
6	4	4

$l_{1,2} = 2$

$l_{1,3} = 5$

$l_{1,4} = 1$

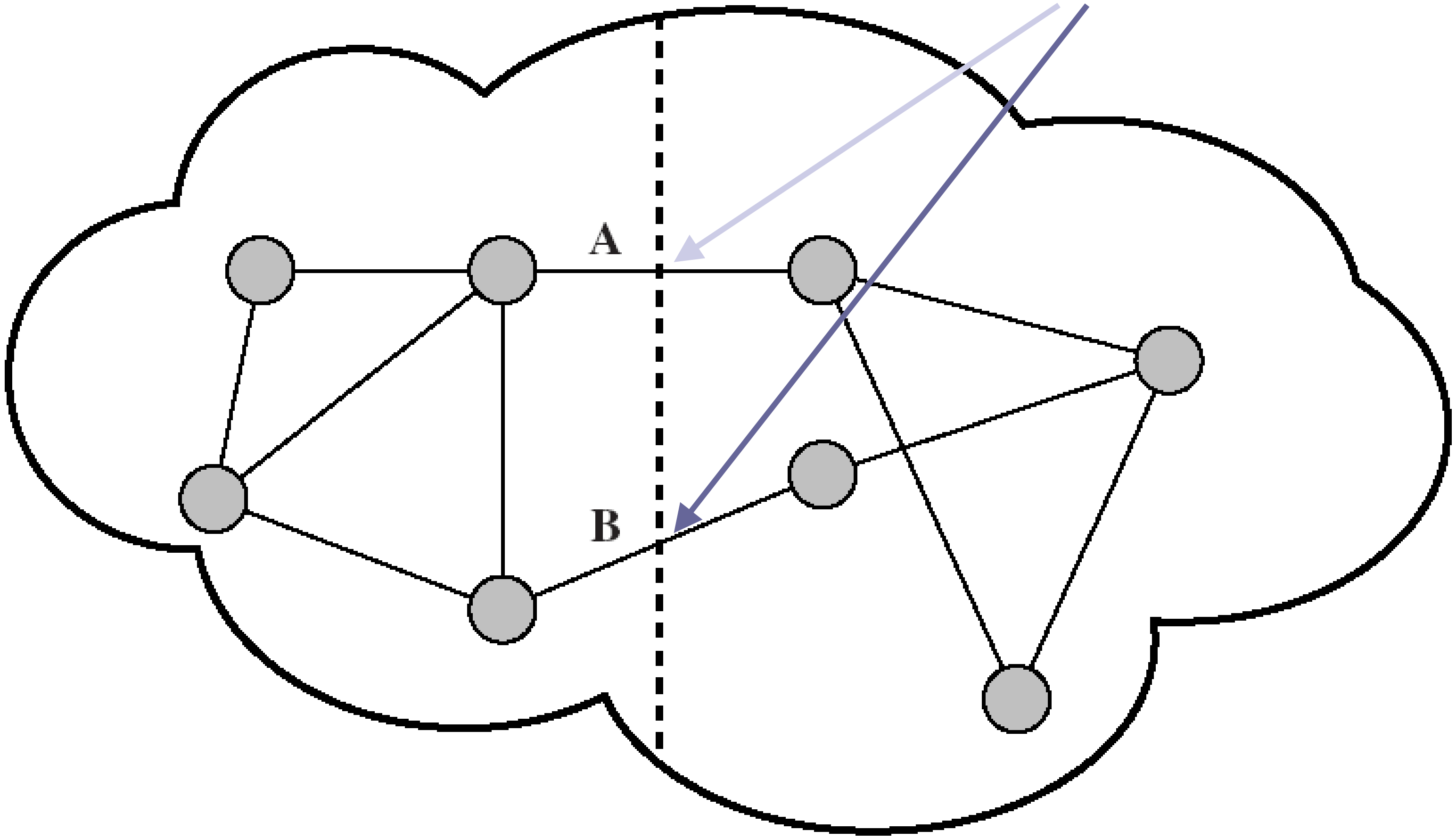
(c) Node 1's routing table after update and link costs used in update

Second Generation

- Method
  - Uses measured delay as performance criterion
  - Per 10s, the node computes the average delay on each outgoing link
    - Time of retransmit - Time of arrive + Transmission time + Transport time
  - Uses Dijkstra’s algorithm
  - The delay info of each link is sent to all other nodes using flooding
- Problem
  - Good under light and medium loads
  - Under heavy loads, little correlation between reported delays and those experienced

The Vibration Problem

Trunks of data vibrate between links A and B



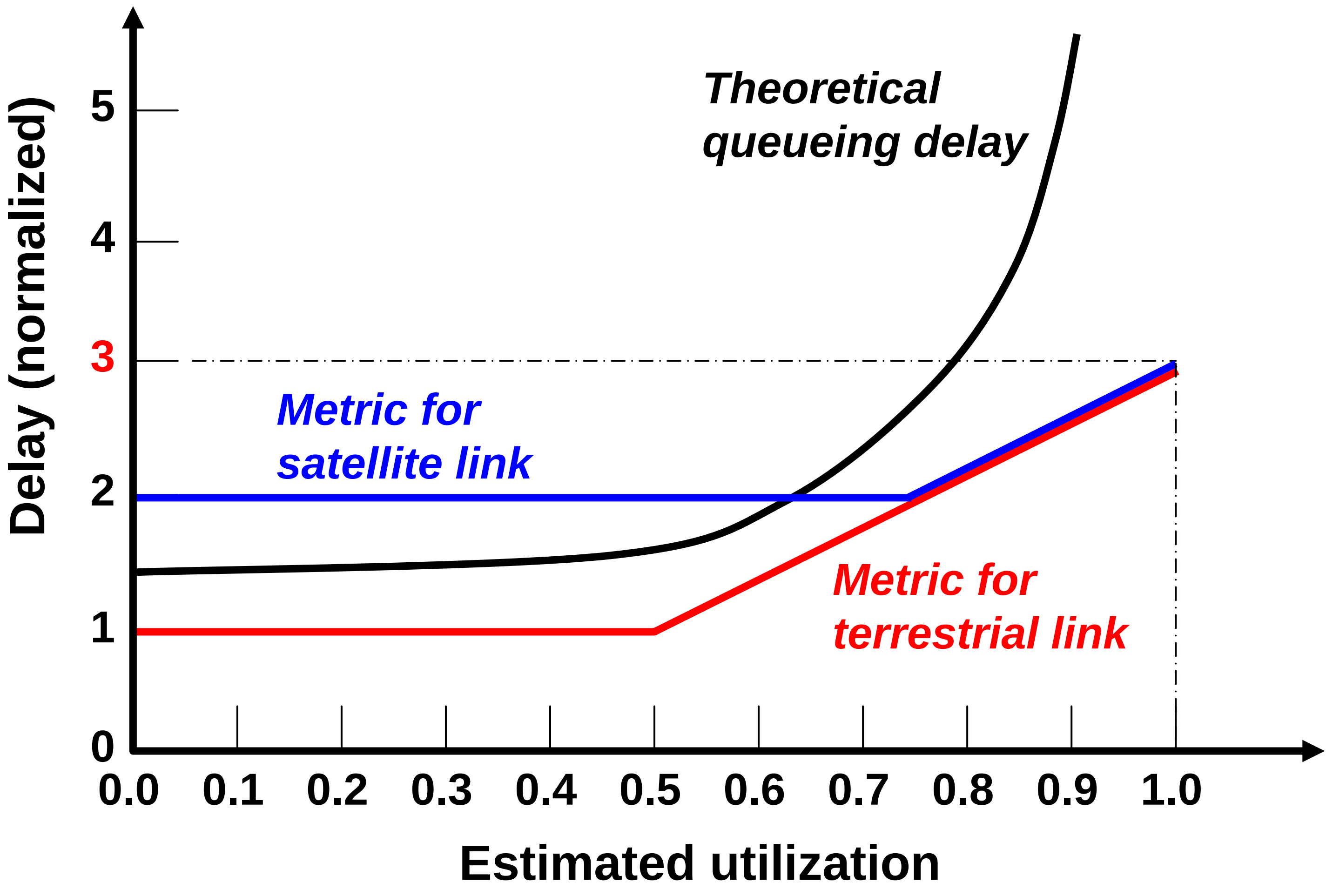
Third Generation

- Goal
  - Let some stay on loaded link to balance and avoid oscillations
  - Under heavy load, should give the majority route a good path instead of every route the best path
- Method
  - Link cost calculations changed
  - Leveling based on current value and previous results
  - Use hop normalized metric to calculate cost

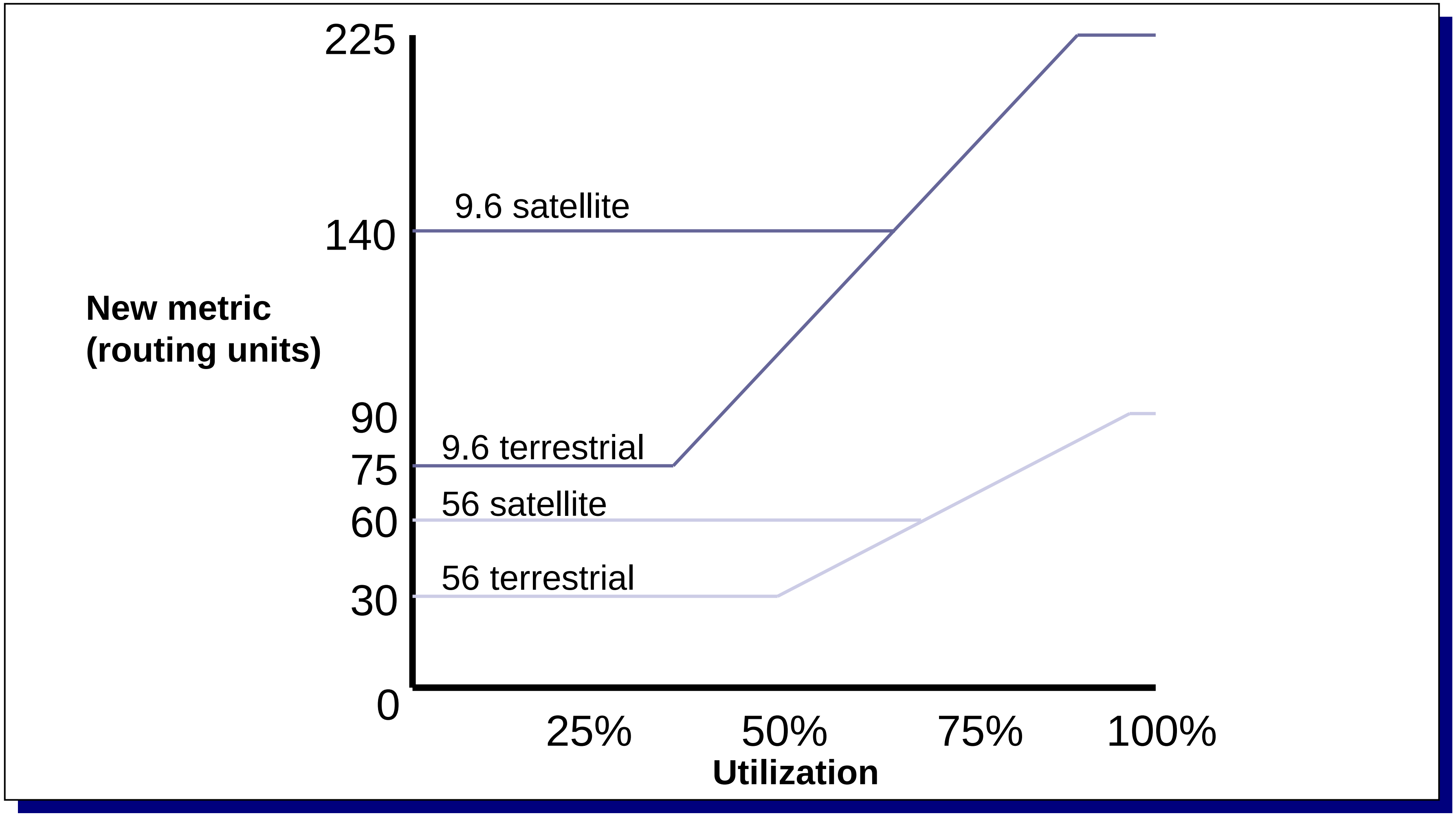
Cost Calculation

- Uses the single-server queuing model, computes link utilization instead of delay
- Step 1. link utilization
  - $\rho = {2(T_S - T)} \Big/ {(T_S - 2T)}$
  - ρ – link utilization
  - T – current measured delay
  - T_s – mean packet length (600 bit) / transmission rate of the link
- Step 2. Leveling
  - $U_{n+1} = \alpha \times \rho_{n+1} + (1-\alpha) \times U_n$
  - U_n – average link utilization at time n
  - α – constant, now set 0.5
- Step 3. Set link cost based on leveled utilization

Link Utilization to Cost (1)

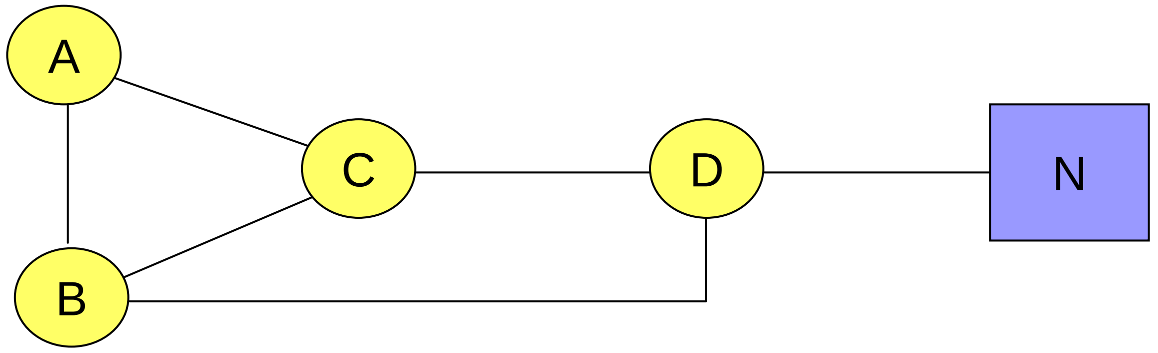


Link Utilization to Cost (2)



12.5 Routing Information Protocol

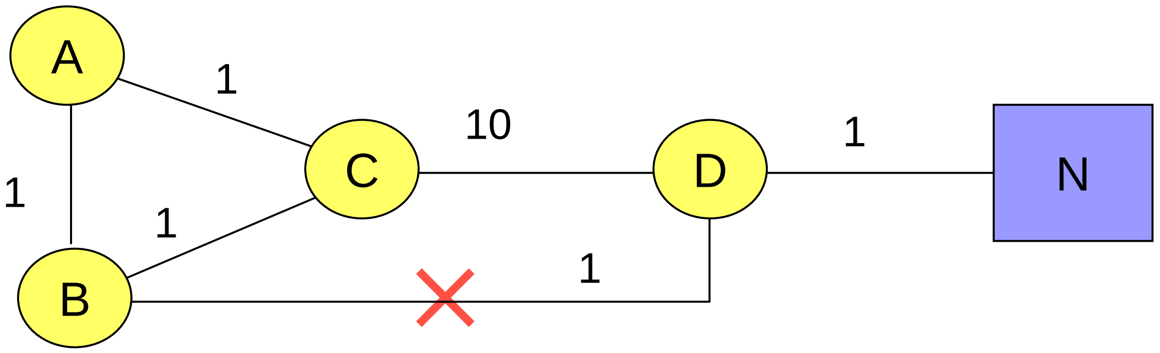
Convergence

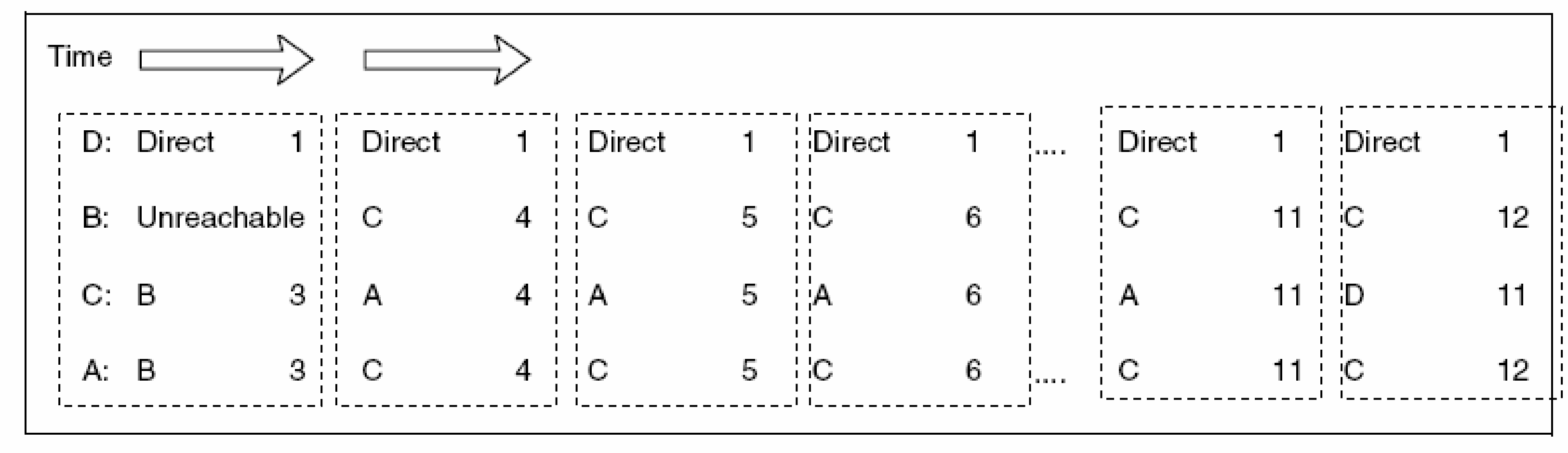


- Router D to the target network: Directly connected network. Metric 1.
- Router B to the target network: Next hop is router D. Metric is 2.
- Router C to the target network: Next hop is router B. Metric is 3.
- Router A to the target network: Next hop is router B. Metric is 3.

counting to infinity

- the link connecting router B and router D fails.

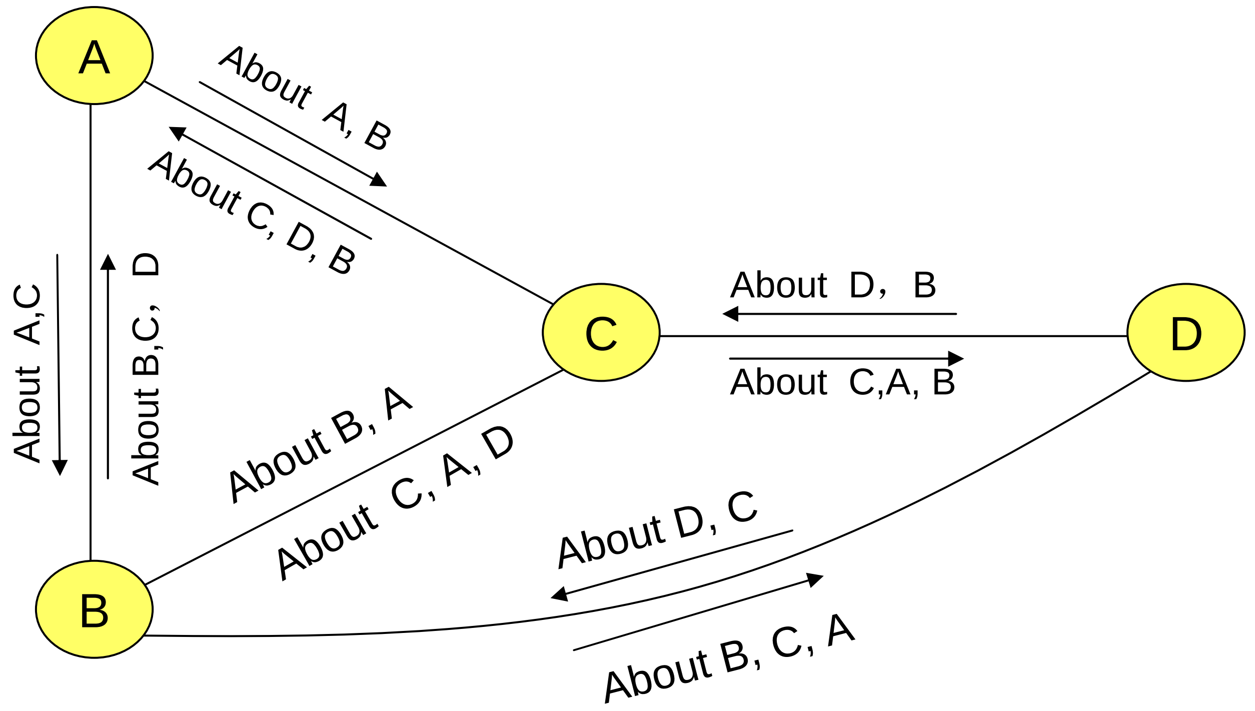



Time

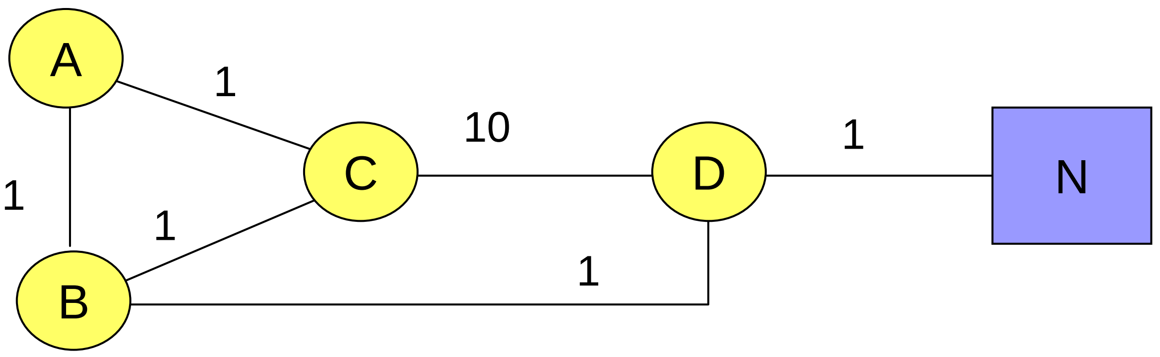
D:	Direct	1	Direct	1	Direct	1	Direct	1	....	Direct	1	Direct	1
B:	Unreachable		C	4	C	5	C	6		C	11	C	12
C:	B	3	A	4	A	5	A	6		A	11	D	11
A:	B	3	C	4	C	5	C	6	....	C	11	C	12

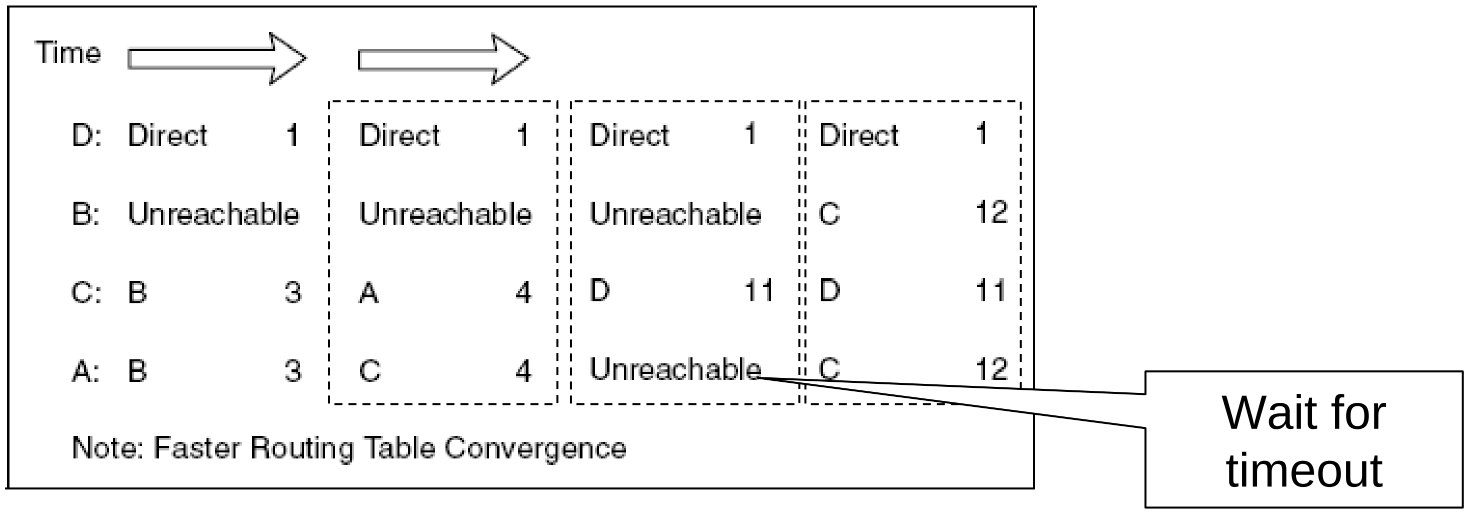
split horizon

- The "simple " scheme omits routes learned from one neighbor in updates sent to that neighbor.



split horizon





Time

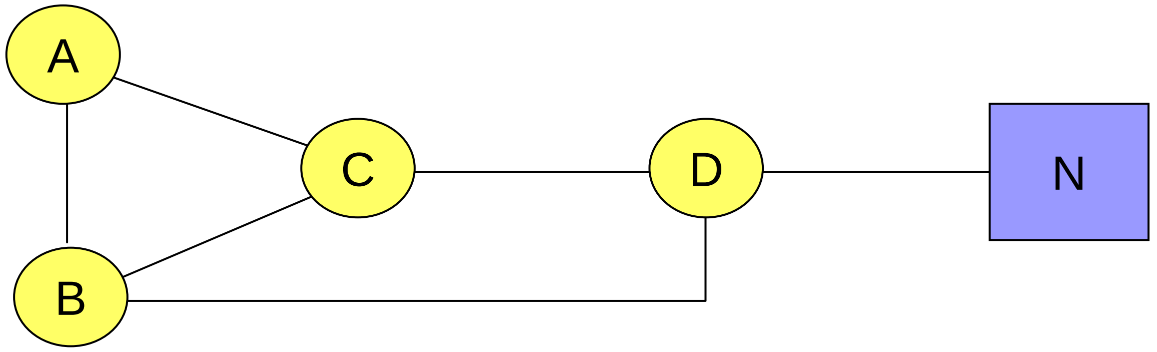
D:	Direct	1	Direct	1	Direct	1	Direct	1
B:	Unreachable		Unreachable		Unreachable		C	12
C:	B	3	A	4	D	11	D	11
A:	B	3	C	4	Unreachable		C	12

Note: Faster Routing Table Convergence

Wait for timeout

Split horizon with poisoned reverse

- "Split horizon with poisoned reverse" includes such routes in updates, but sets their metrics to infinity.
- If A thinks it can get to D via C, its messages to C should indicate that D is unreachable.
- If the route through C is real, then C either has a direct connection to D, or a connection through some other gateway.



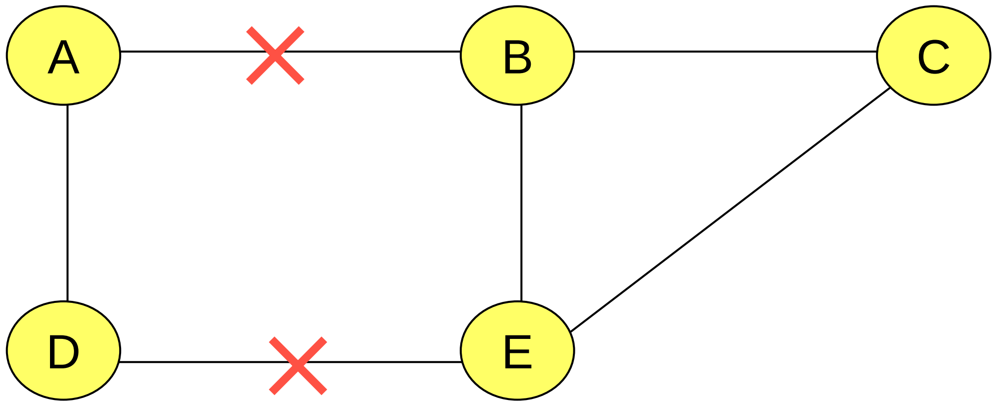
counting to infinity under the Split horizon with poisoned reverse

	距离	下一跳
B→D	2	E
C→D	2	E
E→D	无穷	

	距离	下一跳
B→D	无穷	
C→D	2	E
E→D	无穷	

	距离	下一跳
B→D	3	E
C→D	2	E
E→D	4	B

Unreachable message reached B but not reached C.



Triggered updates

- To get triggered updates, we simply add a rule that whenever a gateway changes the metric for a route, it is required to send update messages almost immediately, even if it is not yet time for one of the regular update message.

12.5 Routing Information Protocol

- RIP is a UDP-based protocol.
- Each host that uses RIP has a routing process that sends and receives datagrams on UDP port number 520.

作业

- 7，8，9，10，12，16