



南京大學

NANJING UNIVERSITY



Computer Networks

Wenzhong Li, Chen Tian

Nanjing University

Material with thanks to James F. Kurose, Mosharaf Chowdhury, and other colleagues.



Chapter 3. Packet Switching Networks

- Switching and Forwarding
- Virtual Circuit and Datagram Networks
- ATM and Cell Switching
- X.25 and Frame Relay
- Routing



Routing Fundamentals



Routing

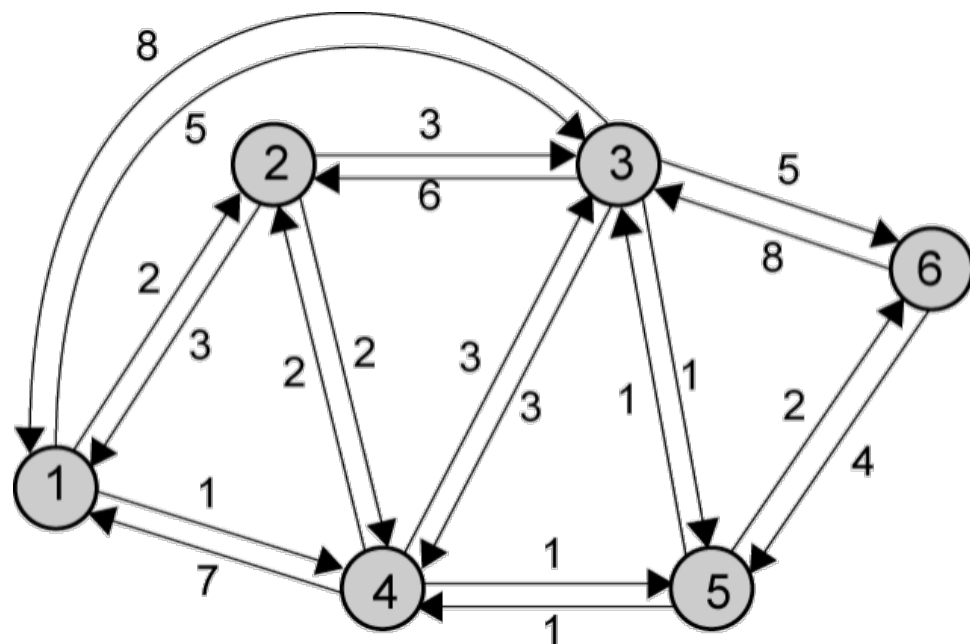
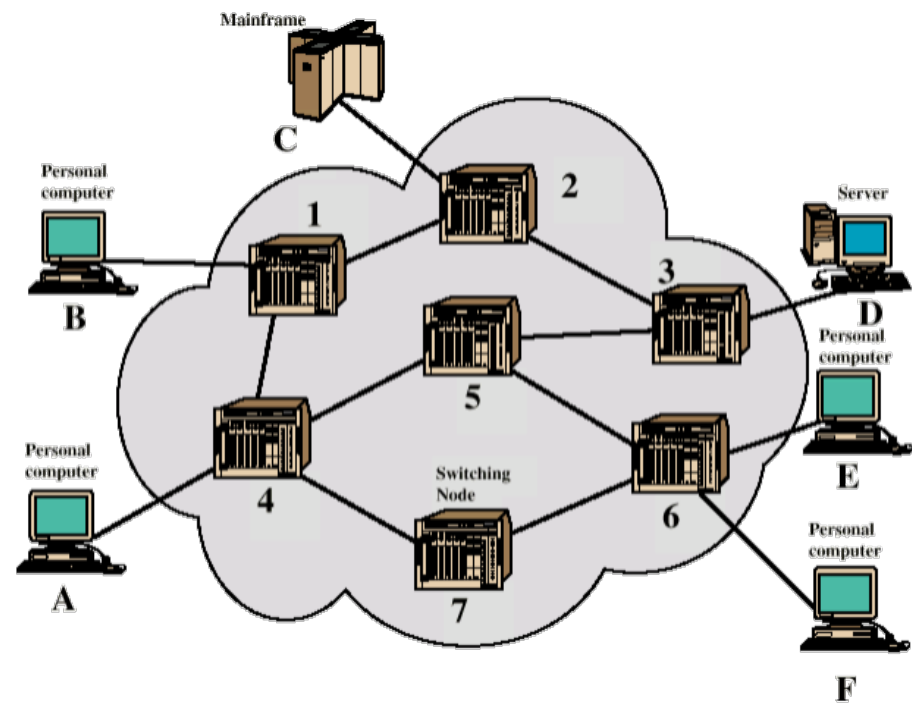
■ Objective

- Build **routing tables** on switches for datagram networks
- Choose paths and build forwarding tables when setting up connections for VC networks

■ Characteristics required

- Efficiency: e.g. smallest possible line or switch
- Resilience: peak load, switch or line failure
- Stability: avoid oscillation

Network Abstraction





Routing Elements

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



Performance Criteria

- Minimum hop

- e.g. 1-3-6

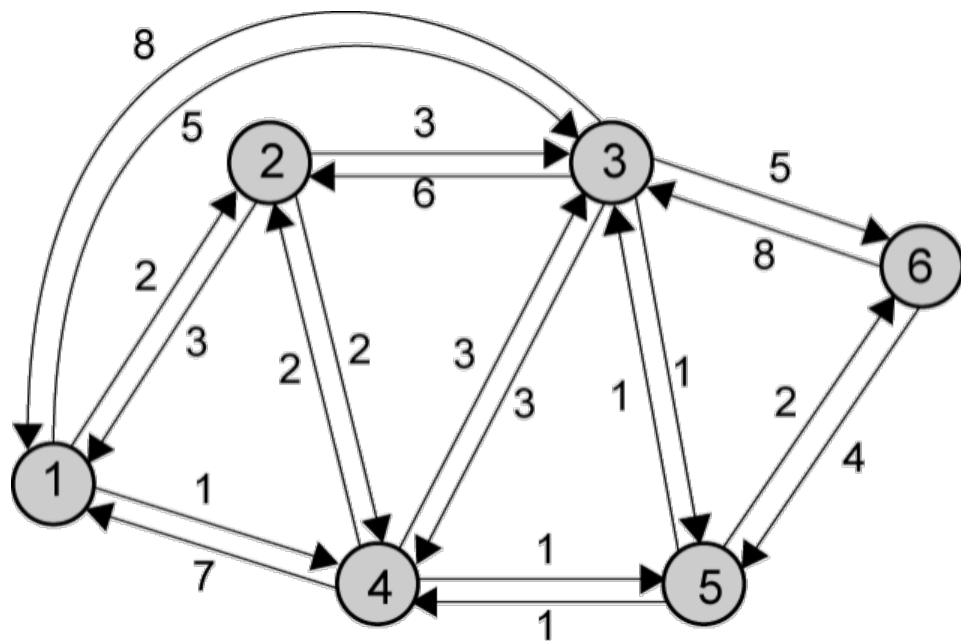
- Least cost

- e.g. 1-4-5-6

- Determine cost

- Minimum delay: queue length

- Largest throughput: reverse of transmission rate





Decision Time and Place

■ Time

- For each packet --- datagram networks
- At the start of each virtual circuit --- VC networks

■ Place

- Centralized
- Source --- source routing
- Distributed --- by each switch node



Network Info Source and Update Timing

■ Info source

- Local information
- Adjacent switches
- All switches in the network

■ Update timing

- Update periodically
- Upon major changes in switches or links
- Fixed (manual configuration)



Different Routing Strategies

- Central (static)
 - Fixed and configured
- Distributed
 - Flooding
 - Random
 - Adaptive

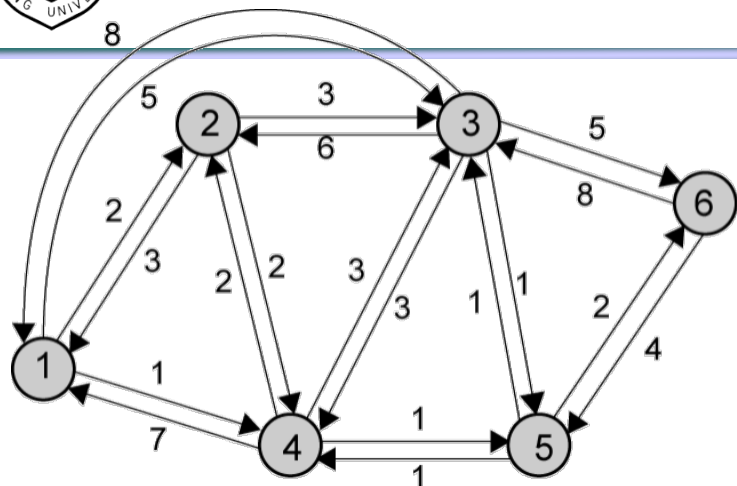


Central Routing

- **Single fixed** route for each source to destination pair
- Determine routes using a **least cost algorithm**
- Routes re-config upon **major changes** in network topology



1 → 6



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	—

Centralized

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

Distributed

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

Fixed
Routing
Tables



Flooding

- No network info required
- Packet sent by switch **to every neighbor**
 - Packets retransmitted on every link except incoming link
- Eventually a number of copies will arrive at destination
- **Duplicates**
 - Many copies of the same packet is created
- **Cycle problem**
 - These copies may circling around the network forever
 - A hop count in packets can handle the problem



Flooding Example

Hop count = 3

- Initial

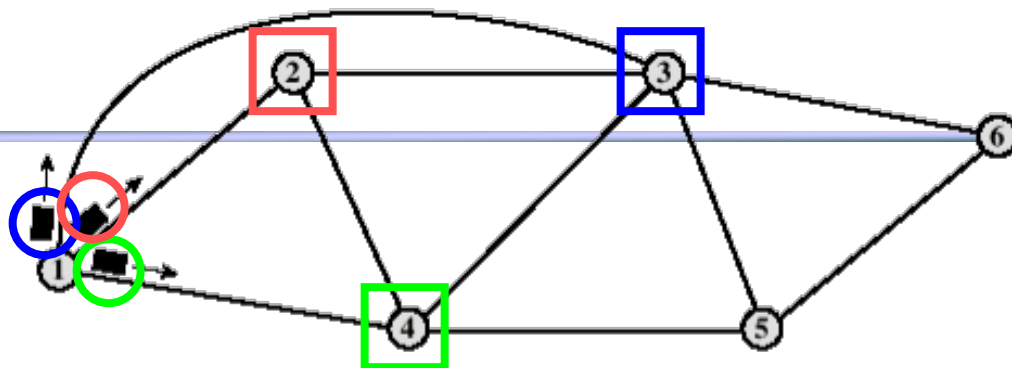
- 3 packets

- 1st hop

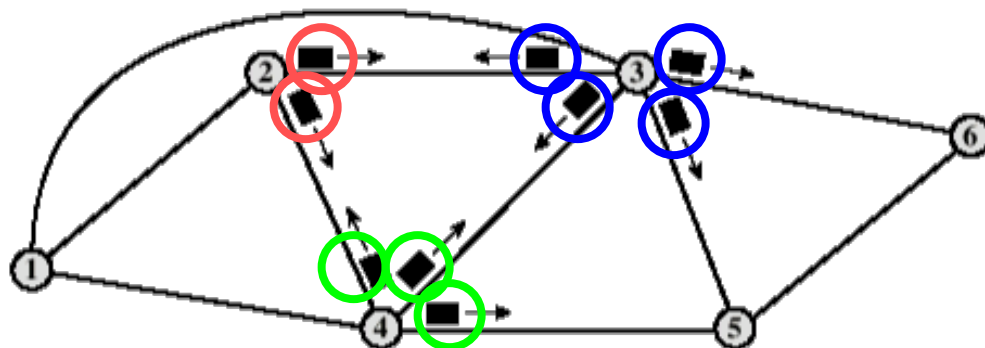
- 9 packets

- 2nd hop

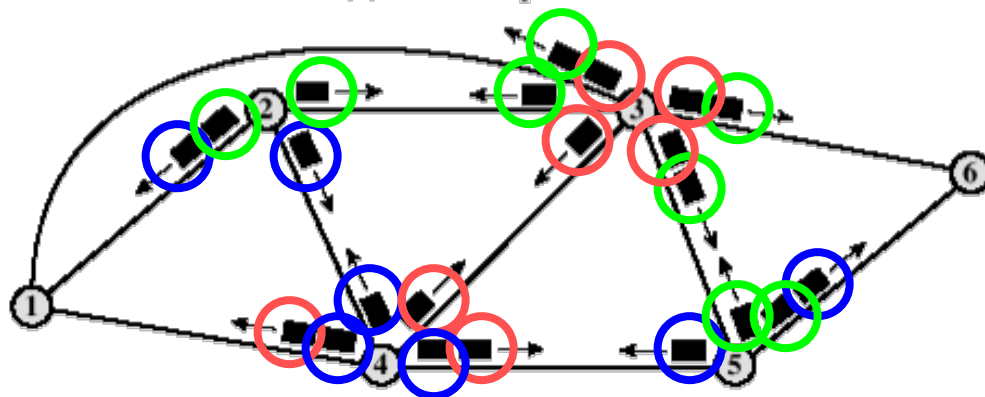
- 23 packets



(a) First hop



(b) Second hop



(c) Third hop



Properties of Flooding

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



Random Routing

- Node selects **one outgoing path** for retransmission of incoming packet
 - Selection can be random or round robin
 - Or based on probability calculation
- No network info needed
- Suitable for **strongly-connected network**
- Route is typically not optimal



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
 - Reverse of queue size – for delay



Adaptive Routing

- Used by almost all packet switching networks
- Routing decisions change as conditions on the network change
- Requires info about network
 - Tradeoff between quality of network info and overhead
- Aid in congestion control



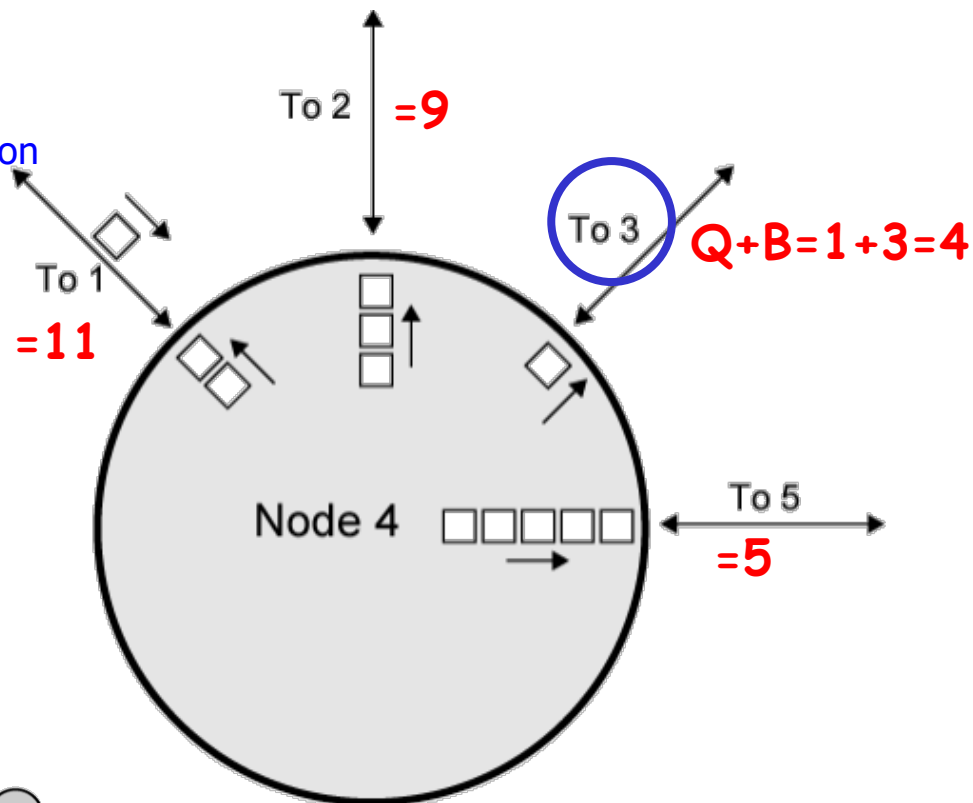
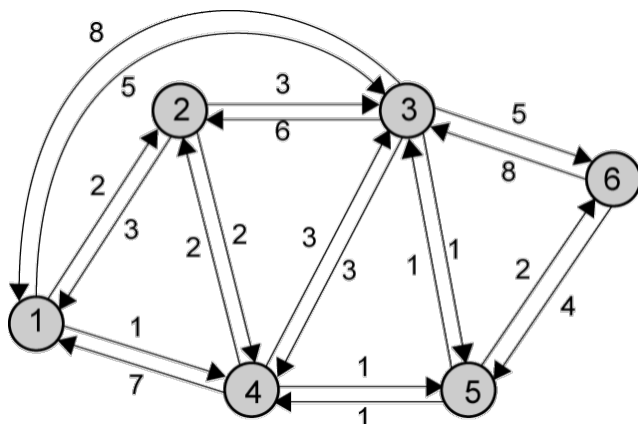
An Isolated Adaptive Routing

- Only local info used
- Strategy 1: route to the outgoing link with shortest queue length Q
 - Pros. Load balancing
 - Cons. May away from the destination
- Strategy 2: take direction into account
 - Each link has a bias B for the destination
 - Route to minimize $Q+B$

Node 4's Bias

Table for
Destination 6

Next Node	Bias
1	9
2	6
3	3
5	0





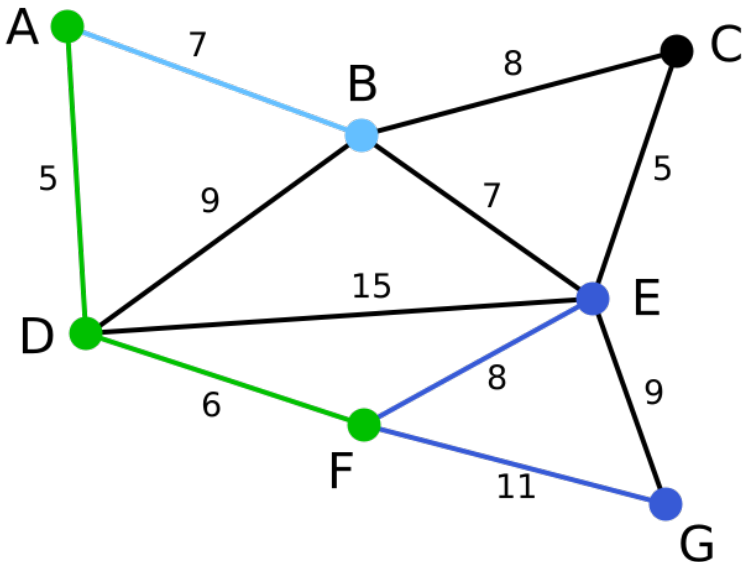
2 Least Cost Algorithms

- For each pair of nodes, find a path with the least cost
- Dijkstra's Algorithm
- Bellman-Ford Algorithm

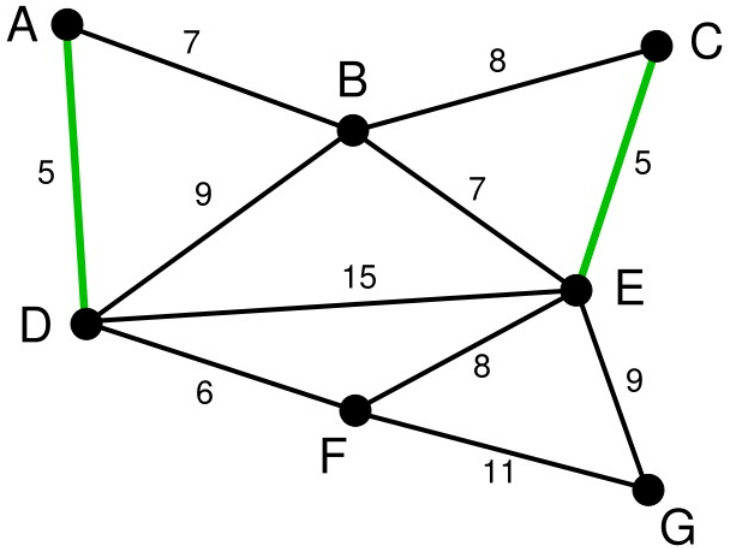


Recap: Prim's and Kruskal's algorithm

- Minimum Spanning Tree Algorithm



Prim's algorithm



Kruskal's algorithm



Dijkstra's Algorithm

- Find shortest paths from given source to all other nodes
 - Developing paths in order of increasing path cost (length)
- Denote
 - N = set of nodes in the network
 - s = the 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
 - $w(i, j) > 0$ if the two nodes are directly connected



Dijkstra's Algorithm Method

- $L(n)$ = **cost of least-cost path** from source s to node n currently known
 - At termination, $L(n)$ is cost of least-cost path from s to n
- Step 1 [**Initialization**]
 - $T = \{s\}$ set of nodes 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



Dijkstra's Algorithm Method

- 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**

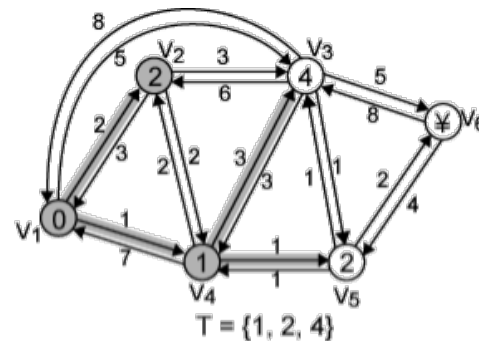
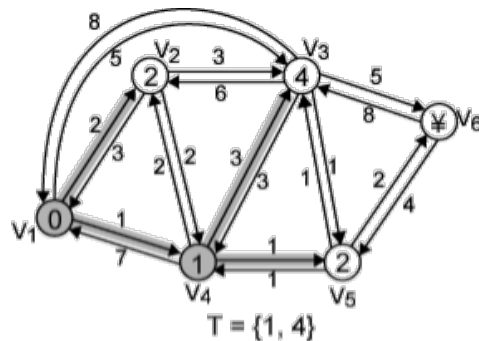
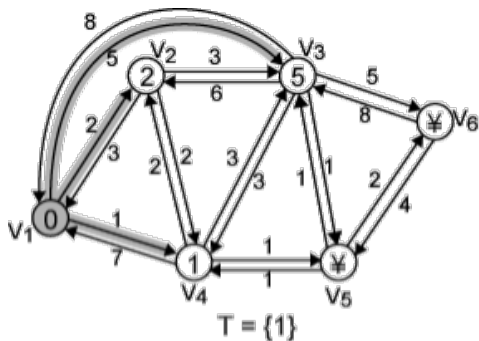


Dijkstra's Algorithm Notes

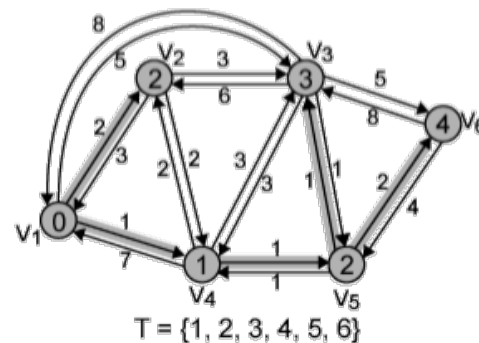
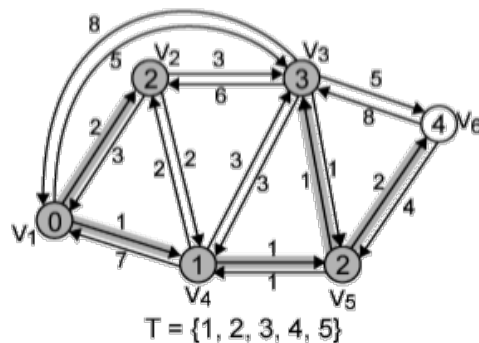
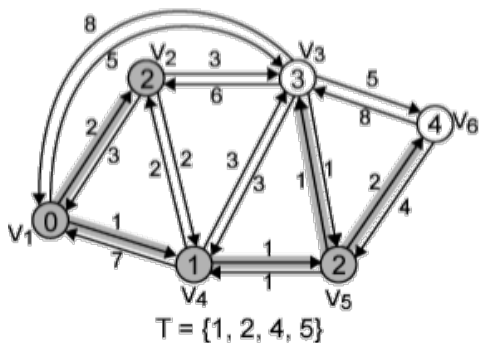
- One iteration of steps 2 and 3 adds one new node to T
 - Defines least cost path from s to that node
- Value $L(n)$ for each node n is the cost (length) of least-cost path from s to n
- At last, T defines the least-cost path from s to each other node



Example of Dijkstra's Algorithm



$$L(3) = \min\{L(3), L(4) + w(4, 3)\} = \min\{5, 1 + 3\} = 4$$

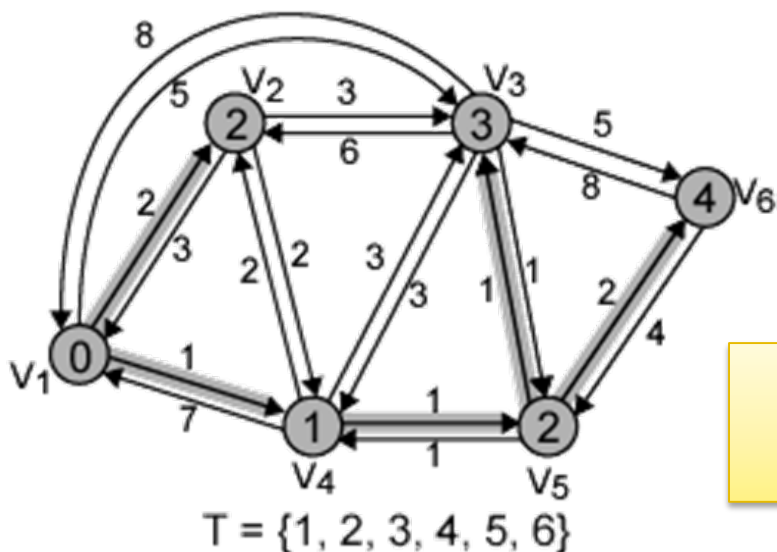


$$L(3) = \min\{L(3), L(5) + w(5, 3)\} = \min\{4, 2 + 1\} = 3$$

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			2	1-4-5	∞	—
3	{1, 2, 4}			4	1-4-3			2	1-4-5	∞	—
4	{1, 2, 4, 5}			3	1-4-5-3					4	1-4-5-6
5	{1, 2, 3, 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



Exercise



$T = T + \{x\}$ // x is the min cost in $N - T$

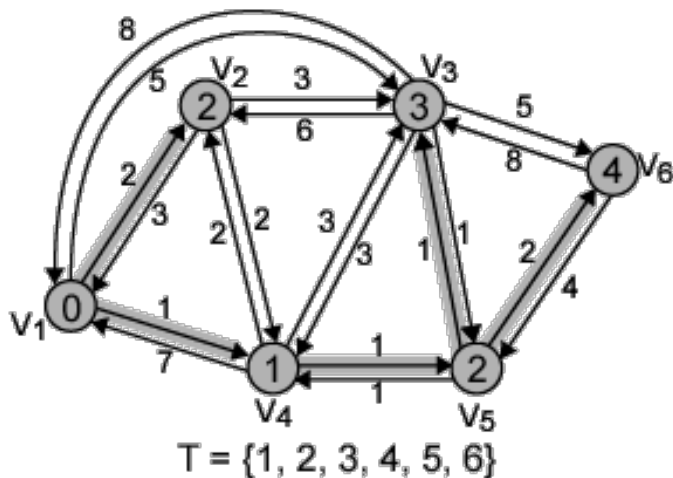
$L(n) = \min[L(n), L(x) + w(x, n)]$ for all $n \notin T$

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			2	1-4-5	∞	—
3											
4											
5											
6											



Results of 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			2	1-4-5	∞	—
3	{1, 2, 4}			4	1-4-3			2	1-4-5	∞	—
4	{1, 2, 4, 5}			3	1-4-5-3					4	1-4-5-6
5	{1, 2, 3, 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



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



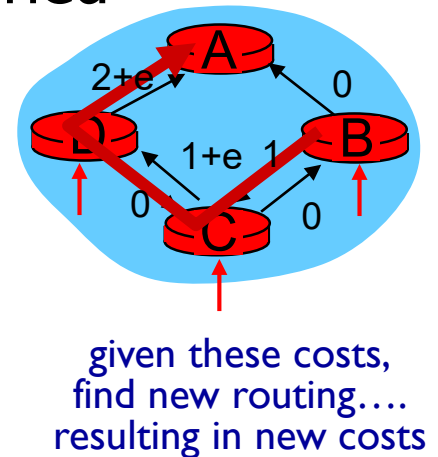
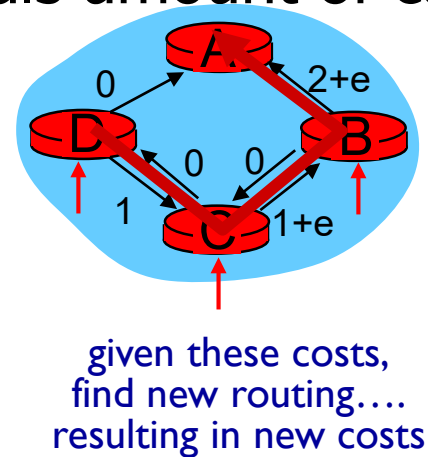
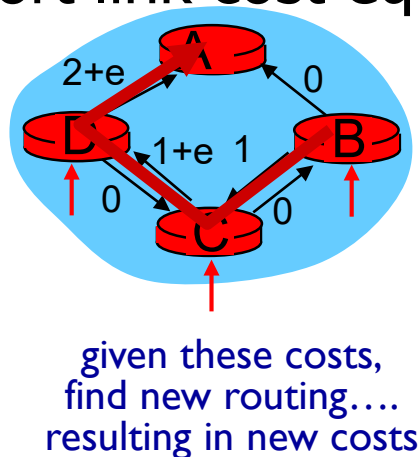
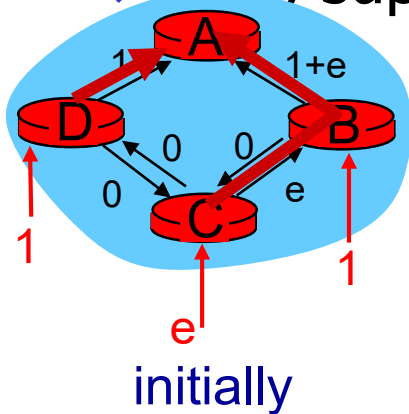
Dijkstra's algorithm discussion

algorithm complexity: n nodes

- ❖ each iteration: need to check all nodes, w, not in N
- ❖ $n(n+1)/2$ comparisons: $O(n^2)$
- ❖ more efficient implementations possible: $O(n \log n)$

oscillations possible:

- ❖ e.g., support link cost equals amount of carried



震荡：解决办法-非同步运行路由算法，链路代价更新随机化



Bellman-Ford Algorithm

- 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 = the 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
 - $w(i, j) > 0$ if the two nodes are directly connected



Bellman-Ford Algorithm Method

- 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
- 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



Bellman-Ford Algorithm Method

- Step 2 [Update]
 - For each successive $h > 0$
 - 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
- Repeat until no change made to route (convergence)

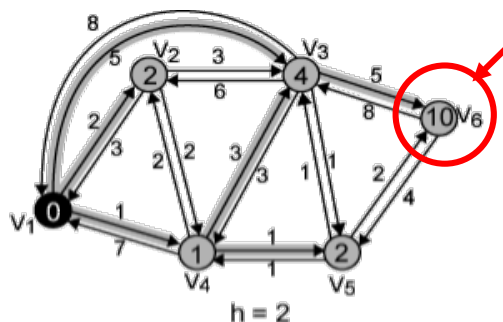
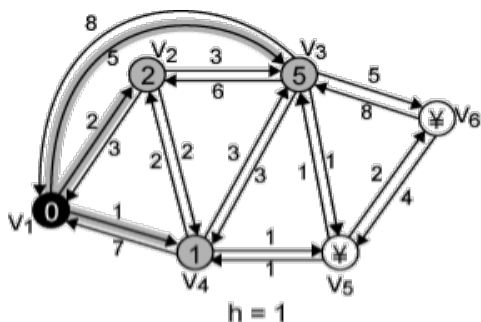


Bellman-Ford Algorithm Notes

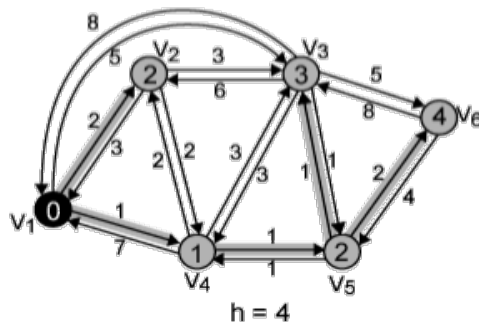
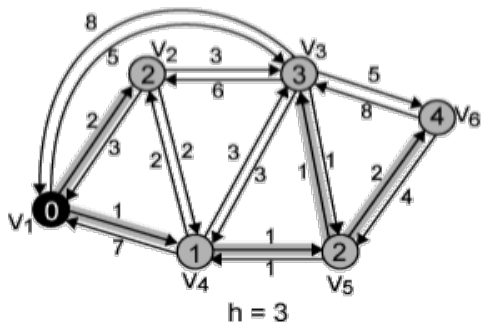
- For each iteration with h and for each destination node n
 - Compares newly computed path from s to n of length h with path from previous iteration ($h-1$)
- If previous path shorter it is retained
- Otherwise new path is defined



Example of Bellman-Ford Algorithm



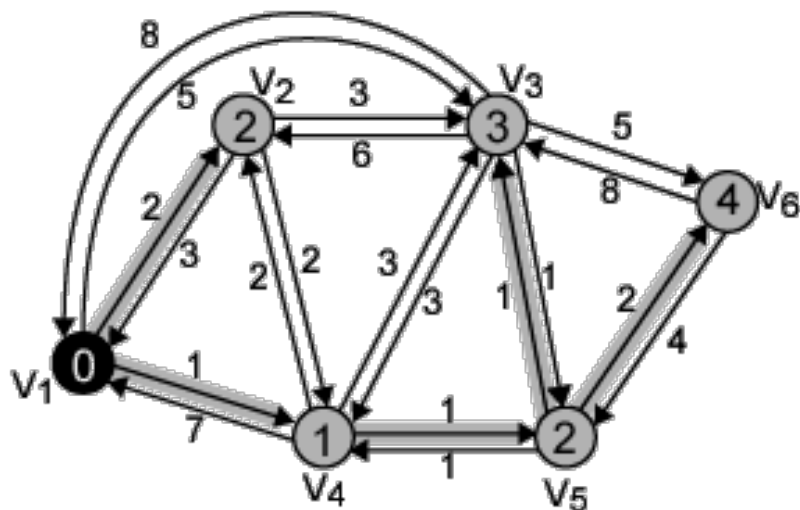
$$L_{h+1}(n) = \min_j [L_h(j) + w(j, n)]$$



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			4	1-4-3			2	1-4-5	10	1-3-6
3			3	1-4-5-3					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



Exercise



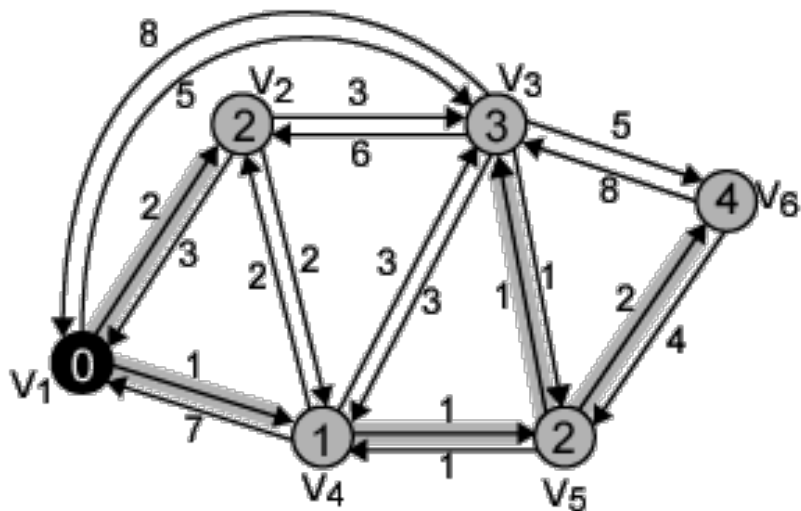
$$L_{h+1}(n) = \min_j [L_h(j) + w(j, n)]$$

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										
3										
4										



Results of Bellman-Ford Algorithm

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			4	1-4-3			2	1-4-5	10	1-3-6
3			3	1-4-5-3					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



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



Link cost changes

link cost changes:

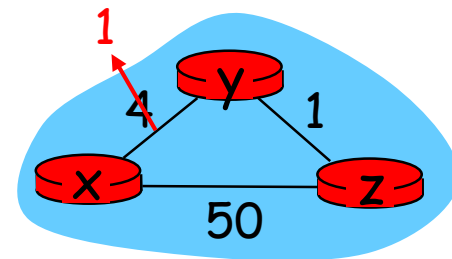
- ❖ node detects local link cost change
- ❖ updates routing info, recalculates distance vector
- ❖ if DV changes, notify neighbors

“good news travels fast”

t_0 : y detects link-cost change, updates its DV, informs its neighbors.

t_1 : z receives update from y , updates its table, computes new least cost to x , sends its neighbors its DV.

t_2 : y receives z 's update, updates its distance table. y 's least costs do *not* change, so y does *not* send a message to z .





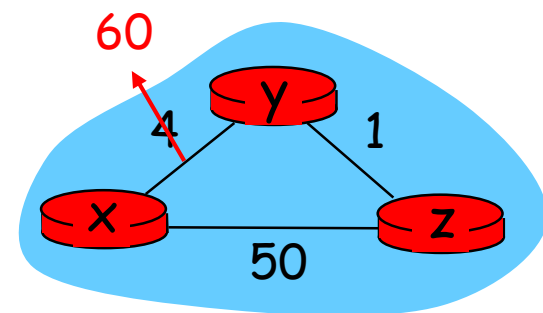
Link cost changes

link cost changes:

- ❖ node detects local link cost change
- ❖ *bad news travels slow* –
“count to infinity” problem!
- ❖ 44 iterations before algorithm stabilizes:
see text

poisoned reverse: (毒性逆转)

- ❖ If Z routes through Y to get to X:
 - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- ❖ will this completely solve count to infinity problem?
 - ❖ No, for 3 or more nodes circle it still exists.



Before:

$D_y(x)=4,$
 $D_y(z)=1, D_z(y)=1, D_z(x)=5$

After *:

$D_y(x)=\min\{60, w(y,z)+D_z(x)\}=6$
 $D_y(z)=1, D_z(y)=1, D_z(x)=5$

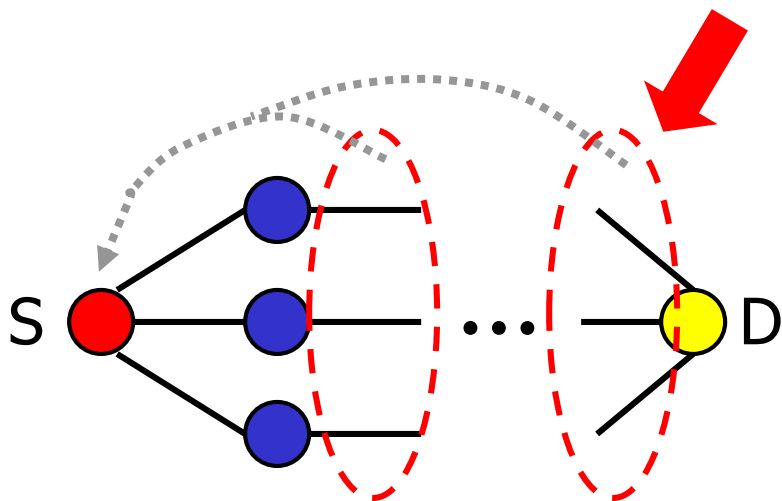
After **::

$D_y(x)=6;$
 $D_y(z)=1$
 $D_z(y)=1,$
 $D_z(x)=\min\{50, w(z,y)+D_y(x)\}=7$

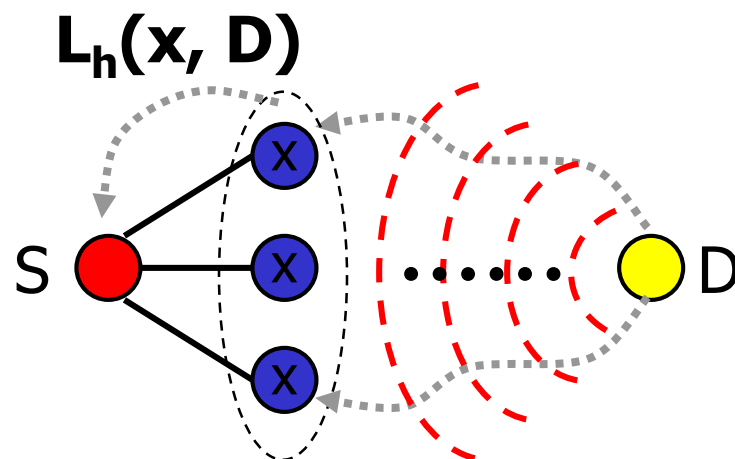
After ***:



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



Dijkstra's
(Link
State)



Bellman-Ford
(Distance
Vector)



Dijkstra vs. Bellman-Ford

- Routing based on **Dijkstra**
 - **Link states** flood to all other nodes
 - Each node will have **complete topology** and build its own routing table
 - Cannot deal with negative weight
- Routing based on **Bellman-Ford**
 - Each node maintain **distance vectors** to other known nodes
 - Vectors exchanged with direct neighbours to update the paths and costs
 - Routing tables built in a distributed way



Dijkstra vs. Bellman-Ford

Message complexity

- **DK**: n nodes, e links, $O(ne)$ messages
- **BF**: Depends on convergence time

Speed of convergence

- **DK**: $O(n^2)$ and quick; May have oscillations
- **BF**: Slow and depends on changes; May contain routing loops

Robustness: what happens if node malfunctions

- **DK**: Advertise incorrect direct links cost; Error range constrained
- **BF**: Error node can exchange incorrect paths cost; Error may propagate through the network



Routing algorithm classification

Q: global or local information?

centralized:

- all routers have complete topology, link cost info
- “link state” algorithms

decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms

Q: static or dynamic?

static:

- ❖ routes change slowly over time

dynamic:

- ❖ routes change more quickly
 - periodic update
 - in response to link cost changes



Determine Link Cost

- 3 stages in ARPANET
- First stage in 1969
 - **Output queue length** is used to define a link cost
 - **Bellman-Ford** algorithm is used for routing
- Second stage in 1979
 - **Measured delay** is used to define a link cost
 - Mix queuing, transmission, and propagation
 - **Time of retransmit – Time of arrive + Transmission time + Propagation time**
 - **Dijkstra's** algorithm is used for routing



Determine Link Cost

- To handle the oscillation problem of Dijkstra
- Let **some stay on loaded links** to balance the traffic
- Apply **Link utilization** to represent a link's state
- Leveling based on previous value and new utilization
- Use **hop normalized metric** to calculate link cost



Calculate Link Cost

- Uses the single-server queuing model
- Link utilization
 - $\rho = 2(T_s - T) / (T_s - 2T)$
 - T – current measured delay
 - T_s – mean packet length (600 bit) / transmission rate of the link
- Leveling
 - $U_n = \alpha \times \rho_n + (1 - \alpha) \times U_{n-1}$
 - U_n – leveled link utilization at time n
 - α – constant, now set 0.5



Summary

- 集中式路由
- 分布式路由：洪泛，随机行走，自适应路由
- 最小代价路由算法及其性能分析
 - Dijkstra Algorithm（集中式、全局信息）
 - Bellman-Ford（分布式、局部信息）
- 链路代价的计算



Homework

- 第四章: R21, P26, P28, P30, P34