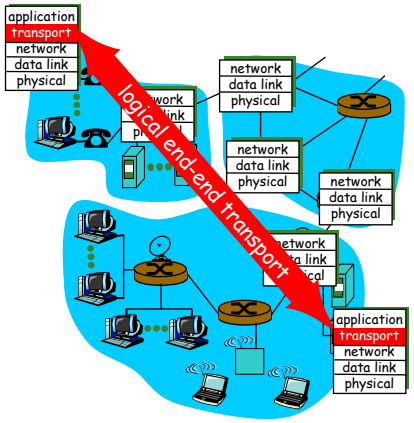
Network Layer I

EECS 122: Lecture 11

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley

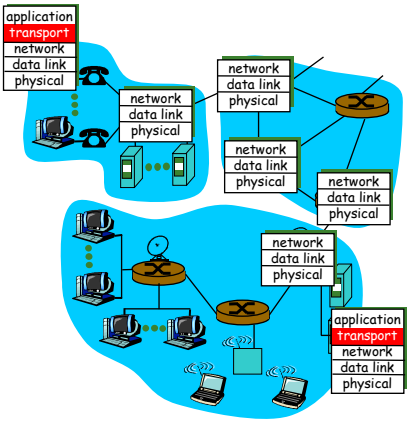
What is the network layer?

- So far we have been treating the network "as a cloud" that "routes packets"



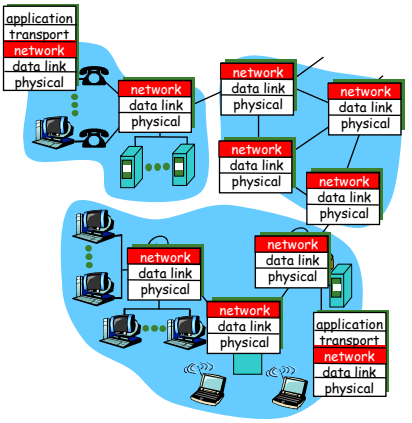
What is the network layer?

- So far we have been treating the network "as a cloud" that "routes packets"
- The Network Layer
1. Chops transport layer messages into IP packets
2. Delivers them to the correct destination(s)
3. Reconstitutes packets into transport layer messages



Network layer

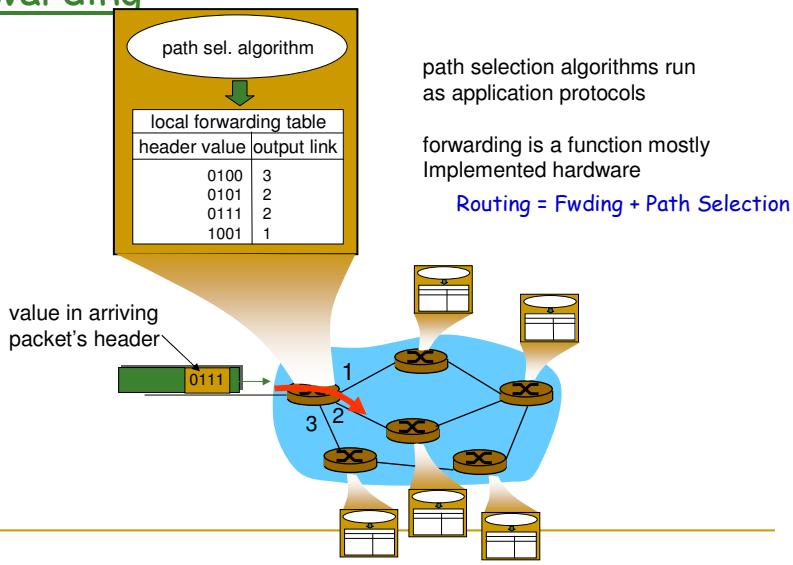
- Network layer protocols must run in *every* host, router
  - In the internet, they all understand IP
- Routers do not examine the transport layer headers of the packets they forward
- But routers must establish routes, and so have to run their own applications e.g. OSPF, BGP etc.



Network Layer Functions

- Control Functions: Ensure that routers are configured to deliver packets correctly to the destination
  - Path Selection (called routing in the book)
  - Connection Setup: required in virtual circuit routing.
- Data Functions: Ensure that arriving packets are forwarded correctly within a router with minimum delay
  - Forwarding

Interplay between path selection and forwarding



path selection algorithms run as application protocols

forwarding is a function mostly Implemented hardware

Routing = Fwding + Path Selection

Network Level Connections

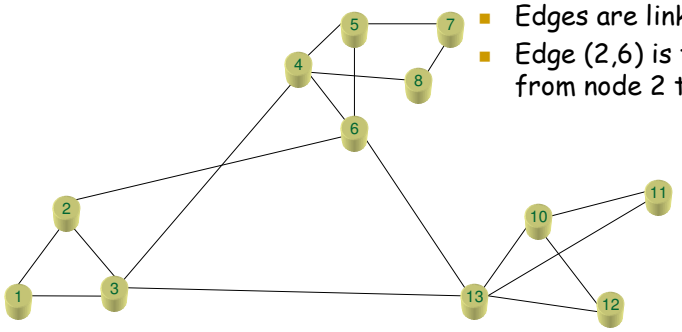
- Important function in *some* network architectures:
  - ATM, frame relay, X.25
- Before data packets flow, two hosts and intervening routers establish virtual connection
  - Routers get involved
- Network and transport layer cnctn service:
  - Network: between two hosts
  - Transport: between two processes
- Note that connection setup is a control function but it is real-time. This makes it difficult to implement in the network layer

Outline of next few lectures

1. Path Selection: Next two lectures
2. Forwarding: One lecture
3. Network Connection Setup (QoS): Two lectures

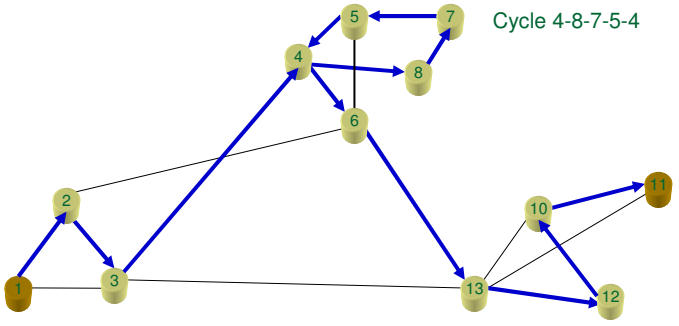
A Graph Model

- Nodes are Routers/Hosts
- Edges are links
- Edge (2,6) is the edge from node 2 to node 6



Walks

A **Walk** from 1 to 11

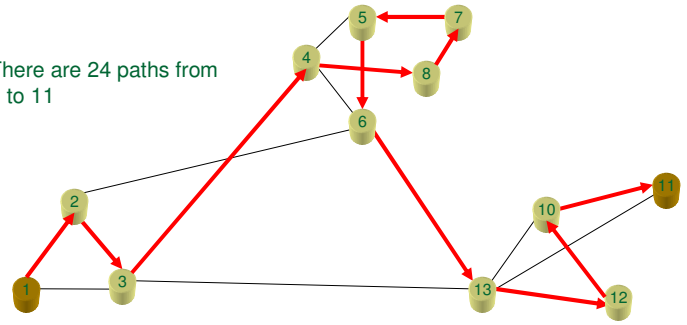


Paths

A **Path** is a Walk with no cycles

Routes are Paths

There are 24 paths from 1 to 11



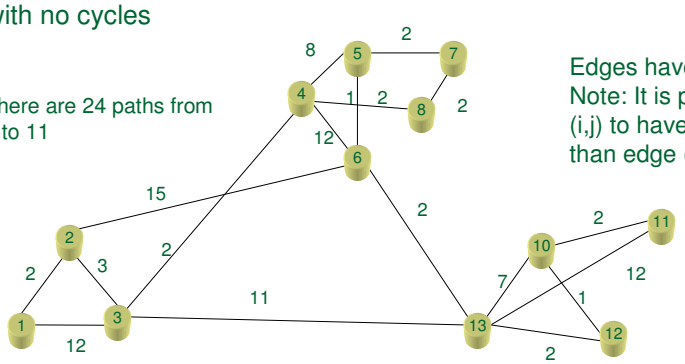
Routes

A **Path** is a Walk with no cycles

Routes are Paths

There are 24 paths from 1 to 11

Edges have weights: c(i,j)
Note: It is possible for edge (i,j) to have a different weight than edge (j,i).



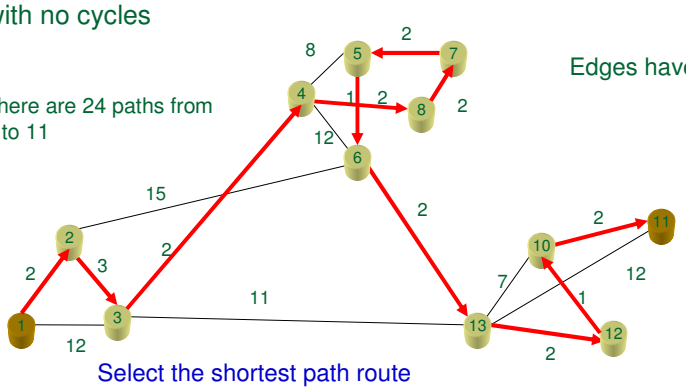
Routing

A **Path** is a Walk with no cycles

Routes are Paths

Edges have weights

There are 24 paths from 1 to 11



Select the shortest path route
Many ways to do this: Path Selection Algorithms

Routing Algorithm classification

Global or decentralized information?

Global:

- all routers have complete topology, link cost info
- "link state" algorithms
  - E.g. OSPF

Decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms
  - E.g. RIP

Static or dynamic?

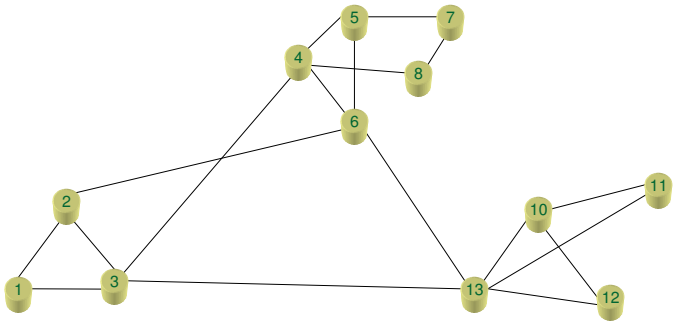
Static:

- routes change slowly over time

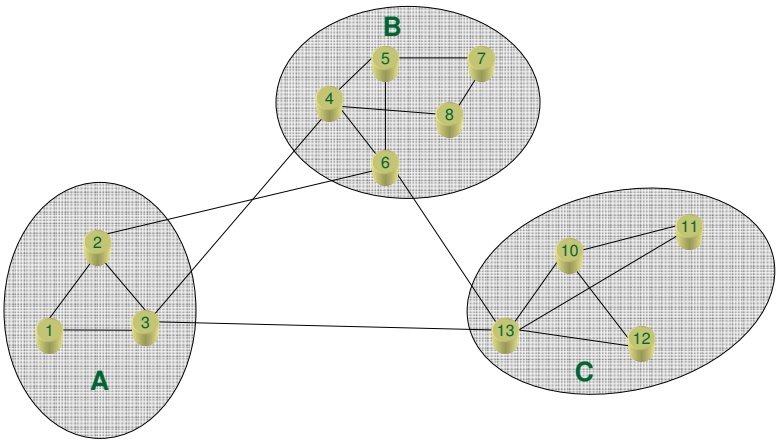
Dynamic:

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

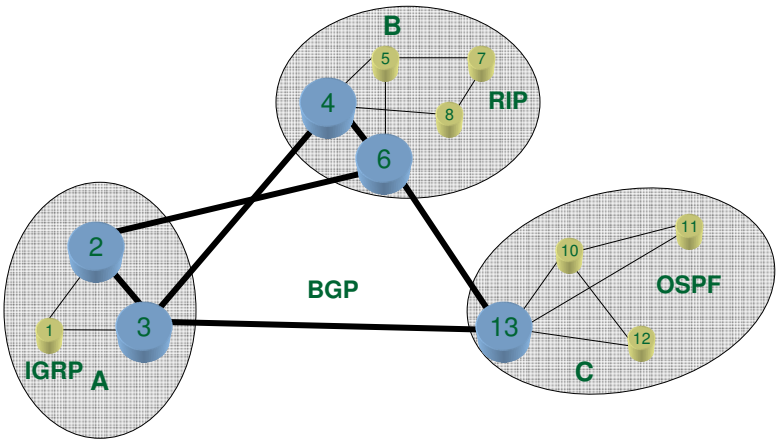
The internet has many Administrative Domains



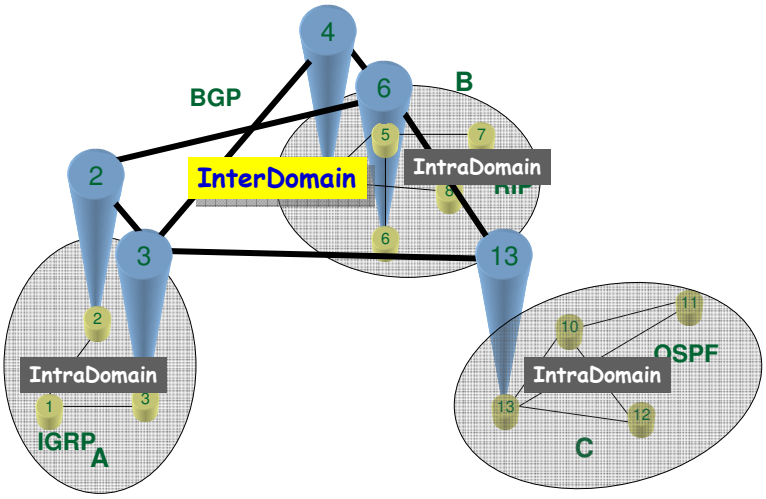
The internet has many Administrative Domains



Border Routers



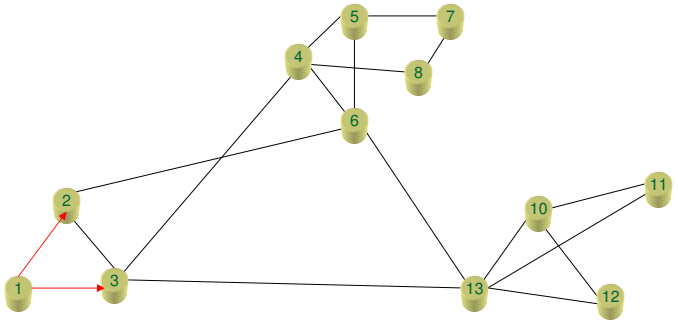
Hierarchical Routing



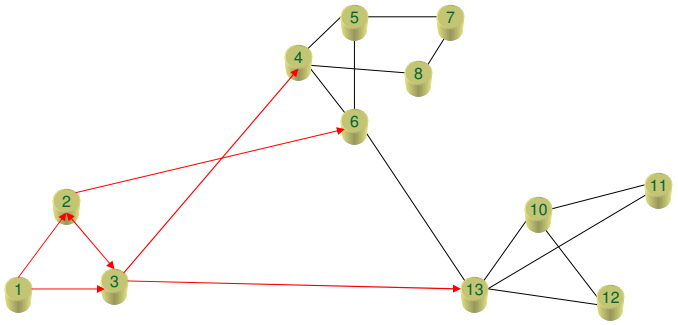
Link State Protocols

1. Every node learns the topology of the network
   - Flooding of Link State Packets (LSP)
2. An efficient shortest path algorithm computes routes to every other node
3. Node updates Forwarding Table

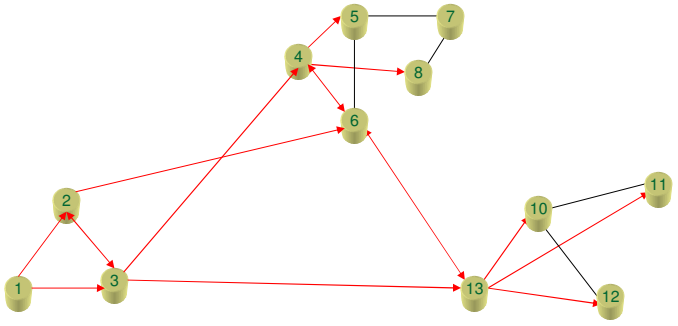
Flooding



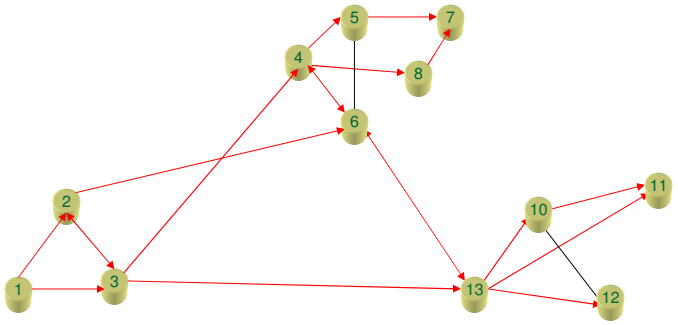
Flooding



Flooding



Flooding



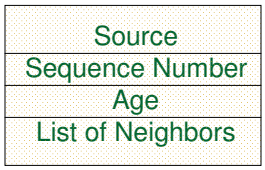
Flooding is trickier than it looks…

- Suppose node i is to flood a list of neighbors.
  - Rule at each of the nodes: upon receiving the packet send to all neighbors except the one you got the packet from…
    - FAILS if there are cycles in the graph
- Don't send the same packet out twice…
  - But what if link states change?
  - Use sequence numbers…
  - Better but not perfect

Some Issues

- What happens if sequence numbers wrap?
  - Hardware errors could cause arbitrary behavior
- What happens when a partitioned network is reconstituted?
- What about security?
- Etc., etc.
- Many lines of code

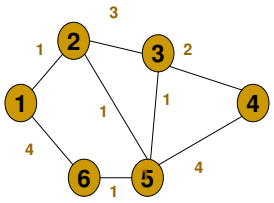
Learning the topology



- Every router sends Link State Packets (LSPs) to all of its neighbors
- LSPs arrive and wait in buffers to be "accepted"
- If node j receives a LSP from node k it compares the sequence numbers. If this is the most recent one from k, send to N(j)-{k}.
  - This way each router can send its LSP to all other routers
- Age starts out at 7. At any router, value is decremented every 8 seconds. At 0 discard.
- As long as sequence don't wrap this works
  - Otherwise things can get ugly

Route Computation: Dijkstra

- Every node knows the graph
  - All link weights are >= 0
- Goal at node 1: Find the shortest paths from 1 to all the other nodes.
- Each node computes the same shortest paths so they all agree on the routes
- Strategy at node 1: Find the shortest paths in order of increasing path length
  - List the nodes in increasing order of (shortest) distance
  - S(k): closest k nodes
  - Iteration k yields S(k) and a way to get there

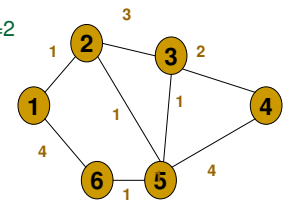


S(1)={1}
S(2)={1,2}
S(3)={1,2,5}

Dijkstra: Shortest Path

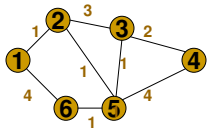
Notation
c(i,j) >=0 :cost of link from (I,j)
D(1,i): Shortest path from 1 to i.
S(k) : the set of nodes k-closest to 1

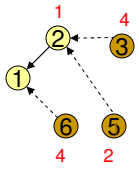
D(1,5)=2



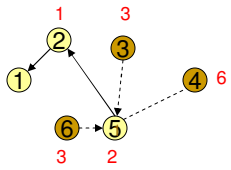
- **IDEA: Given S(k) we can find S(k+1) efficiently:**
- To get S(k+1), observe that
  1. This node cannot be in P(k)
  2. It must be one hop away from some node in P(k)
- Suppose 2 were false. We picked i
  - Node i has no edge into S(k)
  - There must be a node x, not in P(k) such that
    x is one hop away from S(k) and
    D(1,i)=D(1,x)+D(x,i)
  - But then, D(1,x) < D(1,i) and we would have picked x instead.
- Pick node(s) that is one hop away from P(k) that is closest to 1.
- Keep iterating until all nodes are in P

Dijkstra: Shortest Path

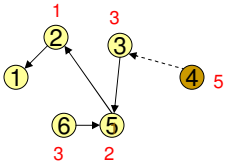




S(2)={1,2}
D(1,2)=1

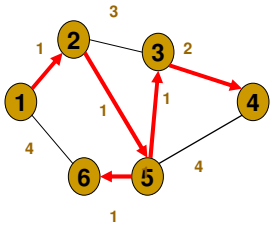


S(3)={1,2,5}
D(1,5)=2



S(4)={1,2,3,5,6}
D(1,3)=3
D(1,6)=3

Dijkstra: Forwarding Table



At node 5

	Outgoing	Cost
1	2	2
2	2	1
3	3	1
4	3	3
6	6	1

Dijkstra's algorithm: Complexity

Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in S()
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)
- Much better complexity for a sparse graph

Distance Vector Algorithms

- Nodes communicate distance estimates to their neighbors, not topology information
- Based on the Bellman Ford Equation:

  Define D(x,y) to the shortest distance from x to y.

  $$D(x,y) = \min_{v \in N(x)} \{c(x,v) + D(v,y)\}$$

  where N(x) are the neighbors of node x.
- Why is this true?

Let D(x,v,y) be the shortest path from x to y where the first node after x is v.

Then D(x,v,y) = c(x,v) + D(v,y).

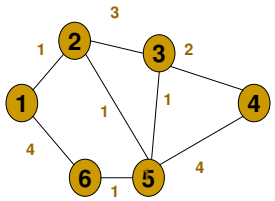
$D(x,y) = \min_v D(x,v,y)$

$= \min_v \{c(x,v) + D(v,y)\}$

Distance Vector Protocols

- Communicate current distance estimates of node to every other node
  - This is called its distance vector: $D_i = (D(i,1),D(i,2),\ldots,D(i,n))$
  - Initially, assume that all distance estimates are c(i,j)
- The nodes do not need to learn the entire topology
  - Just use the distance estimates (vectors) of their neighbors
  - The Bellman Ford equation helps refine estimates over time
- Periodically each node sends its distance vector to all of its neighbors

No communication yet



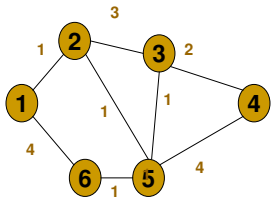
At node 1:
D_1: (0,1,∞,∞, ∞,4)
Neighbor estimates
2: (1,0,∞,∞,∞,∞)
6: (4,∞,∞,∞,∞,0)

At node 6:
D_6: (4,∞,∞,∞,1,0)
Neighbor estimates
1: (0,∞,∞,∞,∞,4)
5: (∞,∞,∞,∞,0,1)

Distance Vector Protocols

- Upon receiving a more recent distance vector from its neighbors, a node, i, stores it and revises D_i:
- New D(i,d):
  - The total cost to send it via neighbor j is the sum of
    - The link cost c(i,j)
    - The stored estimate to reach d from j
  - Pick the lowest sum over all the neighbors
  - $D(i,d) = \min_{j \in N(i)} \{c(i,j) + D(j,d)\}$

Focus on node 6



Node 6 receives
(0,1,∞, ∞,∞,∞,4) from 1
(∞,1,1,4,0,1) from 5

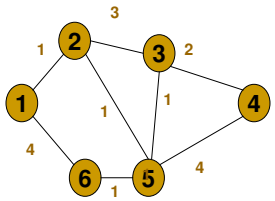
Old Info:
DV: (4,∞,∞,∞,1,0)
Neighbor estimates:
1: (0, ∞,∞,∞,∞,4)
5: (∞,∞,∞,∞,0,1)

Revised:
DV: (**4,2,2,5,1,0**)
Neighbor estimates:
1: (0,1, ∞,∞, ∞,4)
5: **(∞,1,1,4,0,1)**

Send all packets to 2 via 1.

Distance Vector Protocols

- Forwarding Table at 6



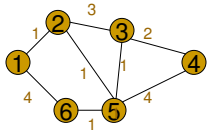
Estimates
DV: (3,2,2,4,1,0)
Neighbor estimates:
1: (0,1,3,5,2,3)
5: (2,1,1,3,0,1)

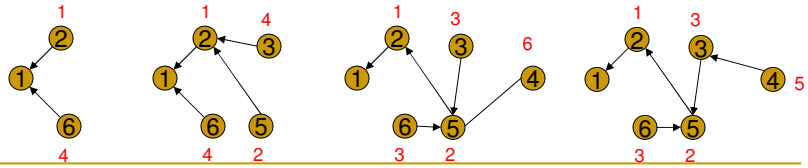
	Node	Cost
1	5	3
2	5	2
3	5	2
4	5	4
5	5	1

Why does this compute shortest paths?

- Suppose in every tick each node sends its distance vector.
- Assume that initial distances are ∞
- At time h, node i has as an estimate of the shortest path to node j that has <= h+1 hops!
- $D^{h+1}(i,j) = \min_{k \in N(i)} \{D^h(k) + c(i,k)\}$

Distances to node 1





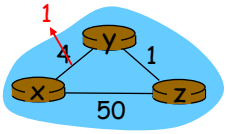
Asynchronous Bellman Ford

- In general, nodes are using different and possibly inconsistent estimates
- If no link changes after some time t, the algorithm will eventually converge to the shortest path
- No synchronization required at all…

Distance Vector: link cost changes

Link cost changes:

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



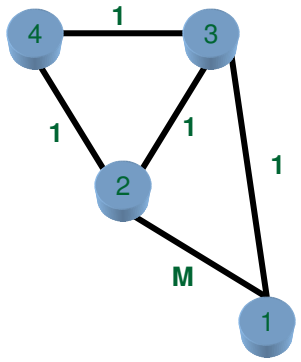
"good news travels fast"

At time t_0, y detects the link-cost change, updates its DV, and informs its neighbors.

At time t_1, z receives the update from y and updates its table. It computes a new least cost to x and sends its neighbors its DV.

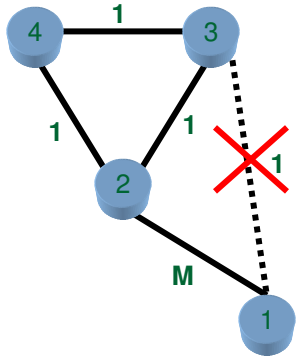
At time t_2, y receives z's update and updates its distance table. y's least costs do not change and hence y does *not* send any message to z.

Bad News Travels Slowly…



D(2,1)=2, D(3,1)=1, D(4,1)=2

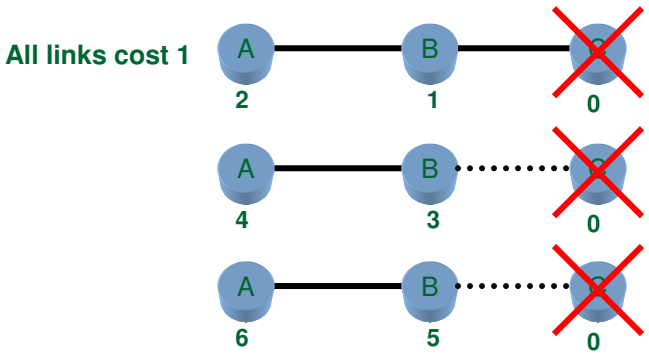
Bad News Travels Slowly…



Node 2 takes about M Iterations to figure out that D(2,1)=M

•Tricks exist to get around these problems but not fool proof

Counting to Infinity



Ping-Pong to Eternity

Oscillations

- Link costs must reflect link speed AND congestion
- Under both LSP and DV routing occurs over a tree
  - The costs of the links of this tree will increase
- The other links will not be congested
  - Their costs will drop
- Routing protocol will shift traffic and create a new tree
- This process of shifting and reshifting can be severe
- Way out: Change congestion costs slowly (exponential averaging) – Route dampening

Comparison of LS and DV algorithms

Message complexity
- LS: with n nodes, E links, O(nE) msgs sent
- DV: exchange between neighbors only
  - convergence time varies

Speed of Convergence
- LS: $O(n^2)$ algorithm requires O(nE) msgs
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:
- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:
- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

Link State vs. Distance Vector
No clear winner

- LS is robust since it each node computes its own routes independently
  - Suffers from the weaknesses of the topology update protocol. Inconsistency etc.
  - Excellent choice for a well engineered network within one administrative domain
  - E. g. OSPF
- DV works well when the network is large since it requires no synchronization and has a trivial topology update algorithm
  - Suffers from convergence delays
  - Very simple to implement at each node
  - Excellent choice for large networks
  - E.g. RIP