Redes de Computadores

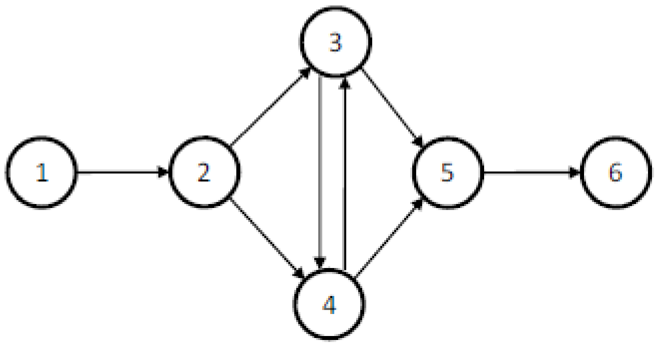
Shortest Paths in Networks

Manuel P. Ricardo, Rui Prior

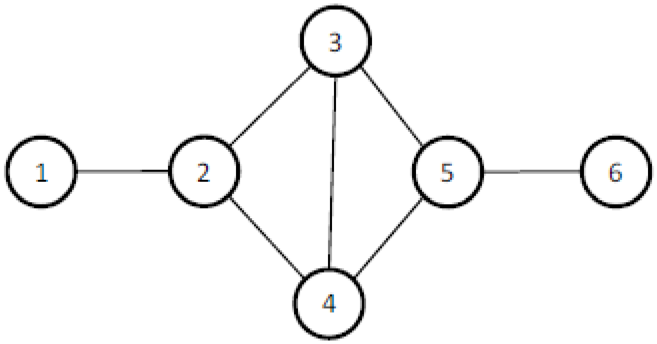
Universidade do Porto

» *What is a graph?*
» *What is a spanning tree?*
» *What is a shortest path tree?*
» *How are paths defined in a network?*
» *How does the Dijkstra algorithm work?*
» *How does a link state routing protocol work?*
» *How does a node learn about neighbours?*
» *How does the Bellman-Ford algorithm work?*
» *How does a distance vector work?*
» *What are the limitations of the layer 2 network of switches?*
» *How does the IEEE spanning tree protocol work?*
» *What is the maximum capacity of a flow network?*

Graph – Directed and Undirected



a) Directed graph



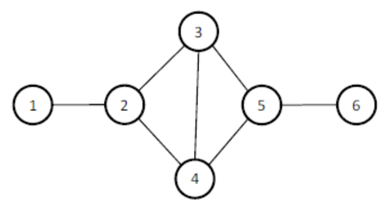
b) Undirected graph

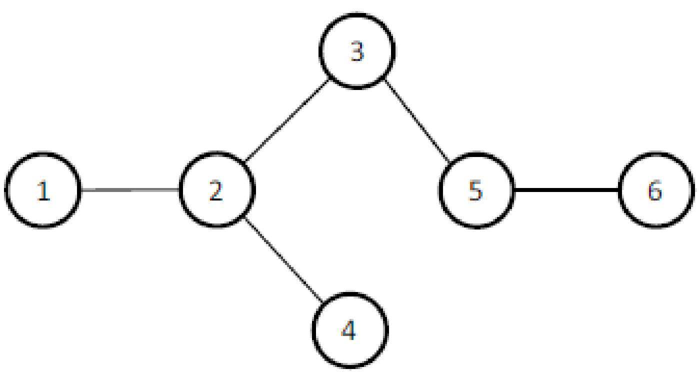
$$\begin{aligned}
G = &\quad (V, E) \\
V = &\quad \{v_1, v_2, v_3, v_4, v_5, v_6\}, & |V| = 6 \\
E = &\quad \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_4), \\
&\quad (v_4, v_3), (v_3, v_5), (v_4, v_5), (v_5, v_6)\}, & |E| = 8
\end{aligned}$$

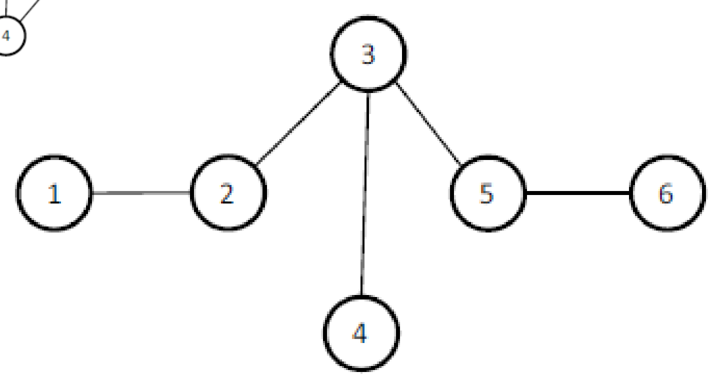
$$\begin{aligned}
G = &\quad (V, E) \\
V = &\quad \{v_1, v_2, v_3, v_4, v_5, v_6\}, & |V| = 6 \\
E = &\quad \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_4), \\
&\quad (v_3, v_5), (v_4, v_5), (v_5, v_6)\}, & |E| = 7
\end{aligned}$$

Tree

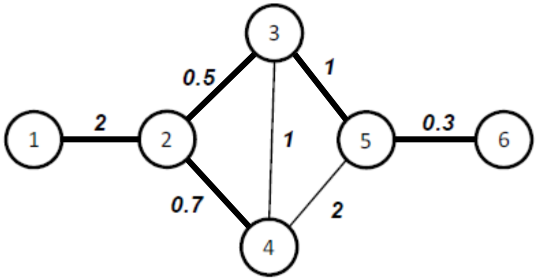
- Tree T = (V,E)
  - connected graph with no cycles
  - number of edges |E| = |V |−1
  - any two vertices of the tree are connected by exactly one path
- A tree T is said to span a graph G = (V,E) (spanning tree) if
  - T = (V,E′) and E′ ⊆ E







Shortest Path Trees

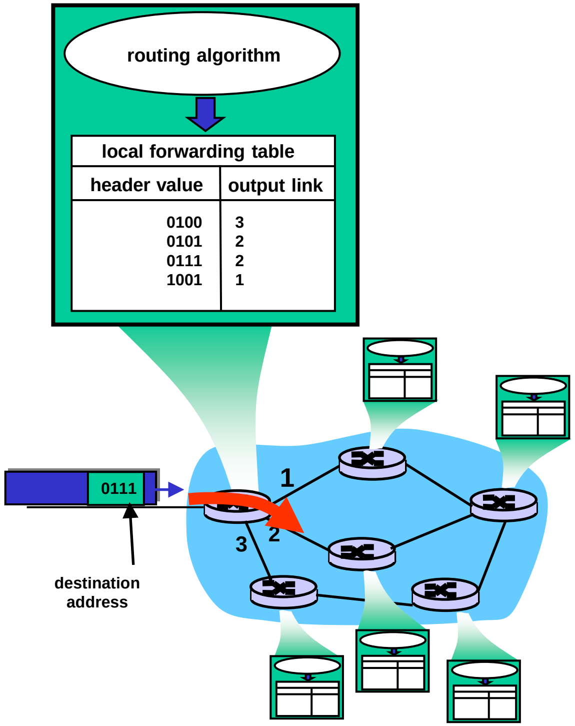


- Graphs and Trees can be weighted
  - » G=(V, E, w)
  - » T=(V, E’, w)
  - » w: E→ℝ
- Total cost of a tree T ➔ $C_{total}(T) = \sum_{i=1}^{|E|} w(e_i)$
- Minimum Spanning Tree T* ➔ $C_{total}(T^*) = \min(C_{total}(T))$
  - » algorithms used to compute MST: Prim, Kruskal
- Shortest Path Tree (SPT) Rooted at Vertex **s**
  - » tree composed by the union of the shortest paths between **s** and each of other vertices of **G**
  - » algorithms used to compute SPT: **Dijkstra, Bellman-Ford**
  - » depends on **s**
- Computer networks use Shortest Path Trees

Routing in Layer 3 Networks

Forwarding, Routing

- Forwarding ➔ data plane
  - » directing packet from input to output link
  - » using a forwarding table
- Routing ➔ control plane
  - » computing paths the packets will follow
  - » routers exchange messages
  - » each router creates its forwarding table



Importance of Routing

- End-to-end performance
  - » path affects quality of service
  - » delay, throughput, packet loss
- Use of network resources
  - » balance traffic over routers and links
  - » avoiding congestion by directing traffic to less-loaded links
- Transient disruptions
  - » failures, maintenance
  - » limiting packet loss and delay during changes

Shortest-Path Routing

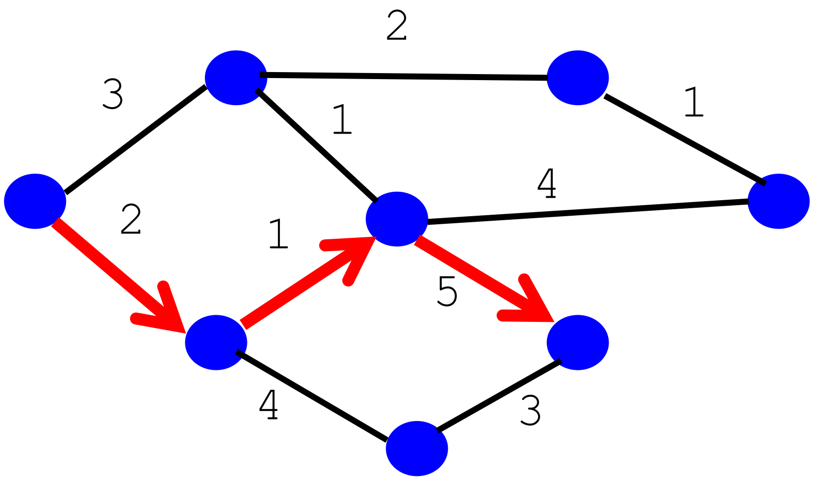
Path-selection model

» Destination-based

» Load-insensitive (e.g., static link weights)

» Minimum sum of link weights

– Special case: minimum hop count (all weights equal)

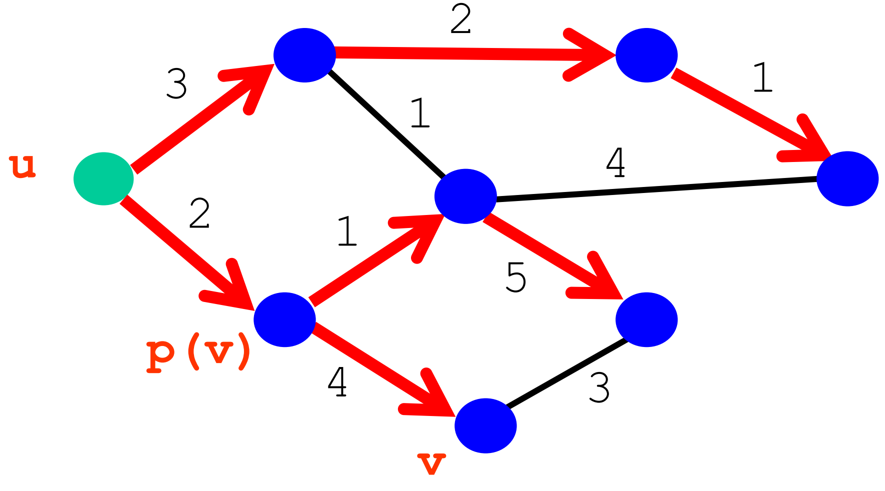


Shortest-Path Problem

- Given a network topology with link costs
  - **c(x,y)** - link cost from node x to node y
  - infinity if x and y are not direct neighbors

- Compute the least-cost paths from source **u** to all nodes

  p(v) - node predecessor of node v in the path to u



Dijkstra's Shortest-Path Algorithm

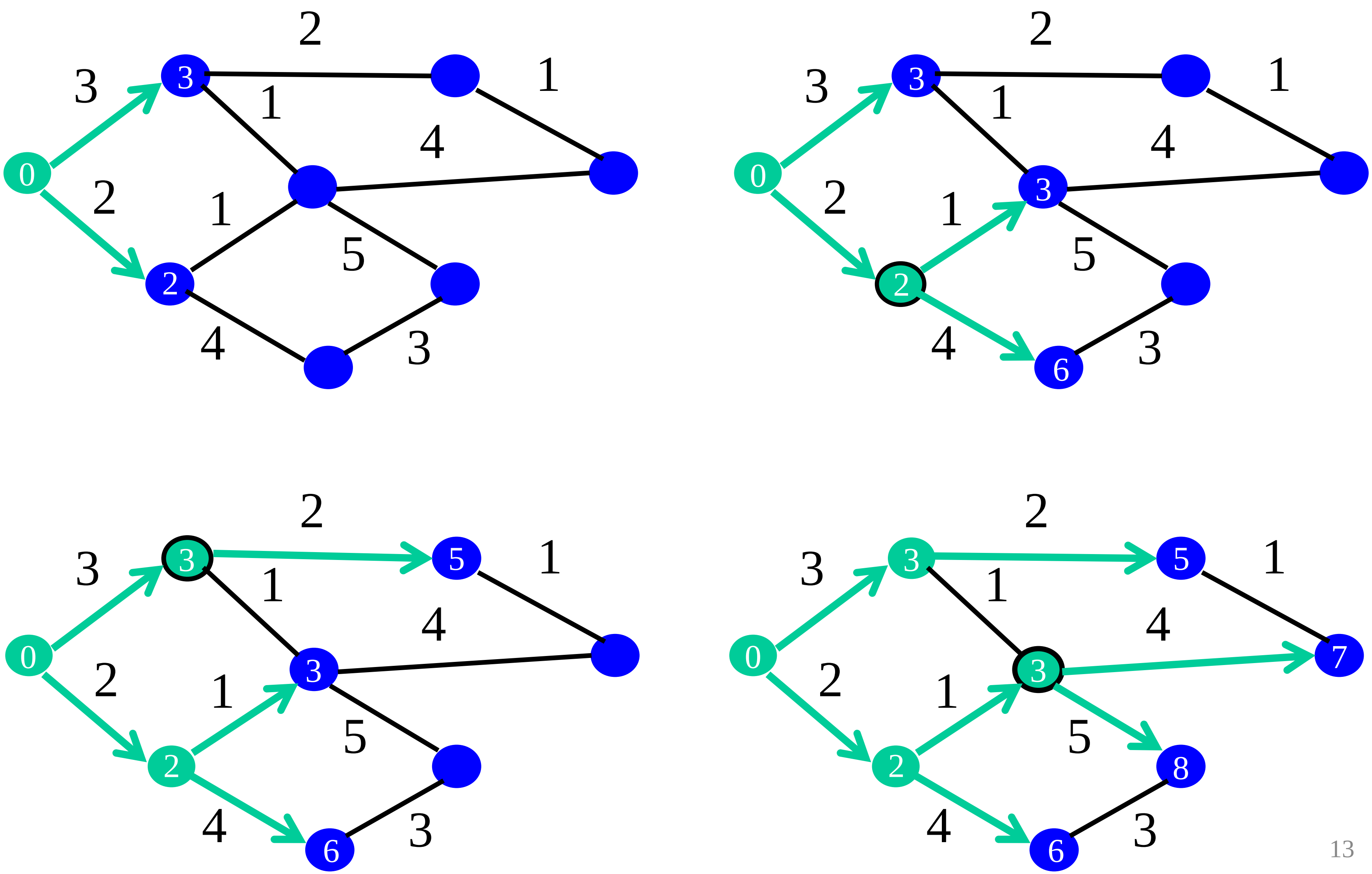
- Iterative algorithm
  - After *k* iterations ➔ known least-cost paths to *k* nodes
- **`S`** ➔ set of nodes for which least-cost path is known
  - Initially, **`S={u}`**, where **`u`** is the source node
  - Add one node to **`S`** in each iteration
- **`D(v)`** ➔ current cost of path from source to node **`v`**
  - Initially
    - **`D(v)=c(u,v)`** for all nodes **`v`** adjacent to **`u`**
    - **`D(v)=∞`** for all other nodes **`v`**
  - Continually update **`D(v)`** when shorter paths are learned

Dijsktra's Algorithm

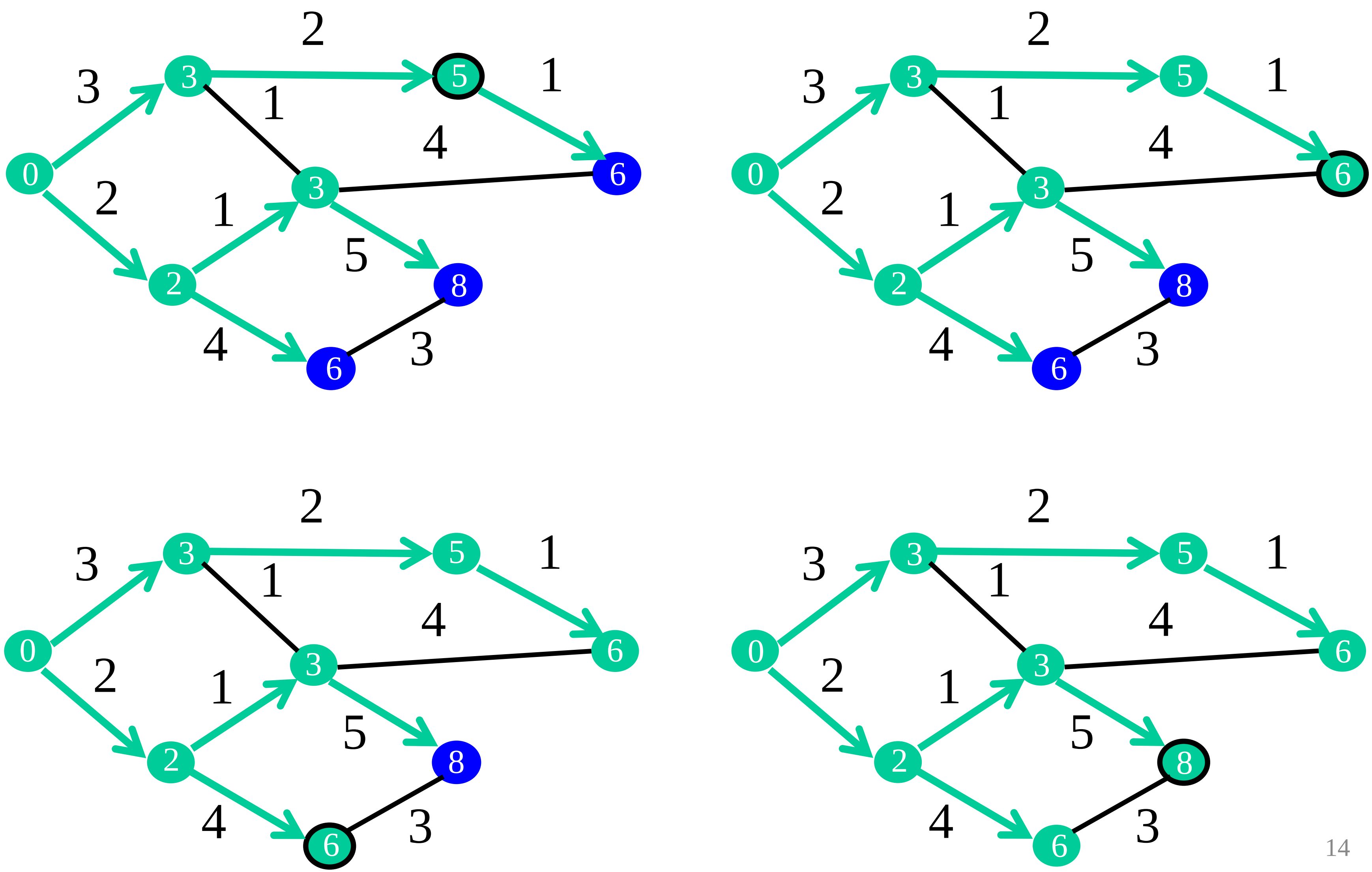
```
Initialization:
  S = {u}
  for all nodes v
    if v adjacent to u
      D(v) = c(u,v)
      p(v) = u
    else D(v) = ∞

Loop:
  find node w not in S with the smallest D(w)
  add w to S
  update D(v), p(v) for all v adjacent to w and not in S:
    if D(w) + c(w,v) < D(v)
      D(v) = D(w) + c(w,v)
      p(v) = w
repeat until all nodes are in S
```

Dijkstra's Algorithm - Example

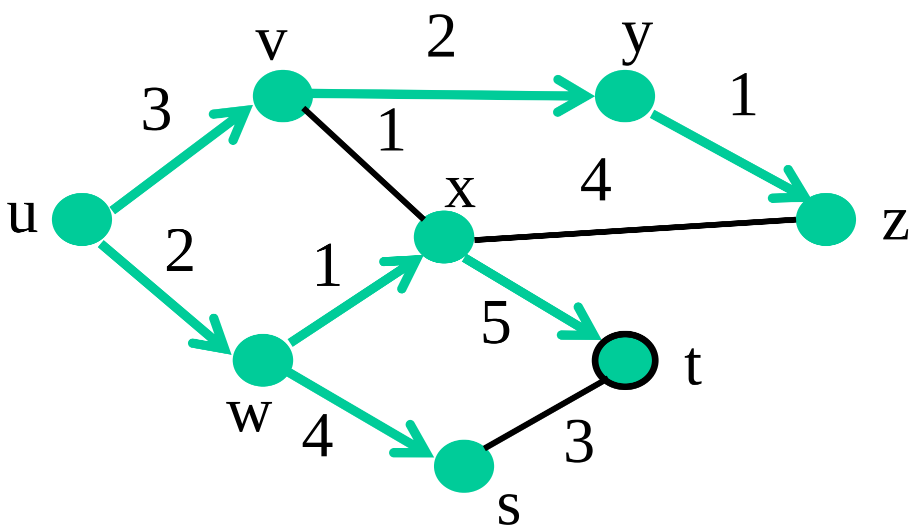


Dijkstra's Algorithm - Example



Shortest-Path Tree

- Shortest-path tree from u



NOTE: keeping track of the predecessors is necessary to build the shortest paths tree

- Forwarding table at u

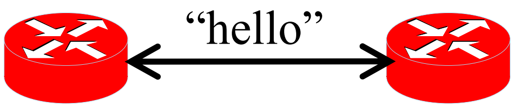
	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Link-State Routing

- Each router keeps track of its incident links
  - » link up, link down
  - » cost on the link
- Each router broadcasts link states
  - » every router gets a complete view of the graph
- Each router runs Dijkstra's algorithm, to
  - » compute the shortest paths
  - » construct the forwarding table
- Example protocols
  - » Open Shortest Path First (OSPF)
  - » Intermediate System – Intermediate System (IS-IS)

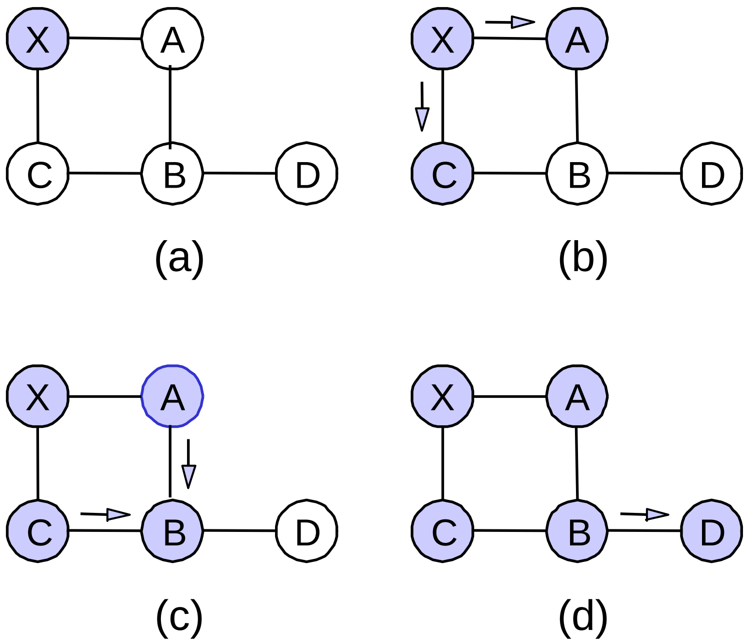
Detection of Topology Changes

- Beacons generated by routers on links
  - » Periodic “hello” messages in both directions
  - » After a few missed “hellos” ➔ link failure detected



Broadcasting the Link State

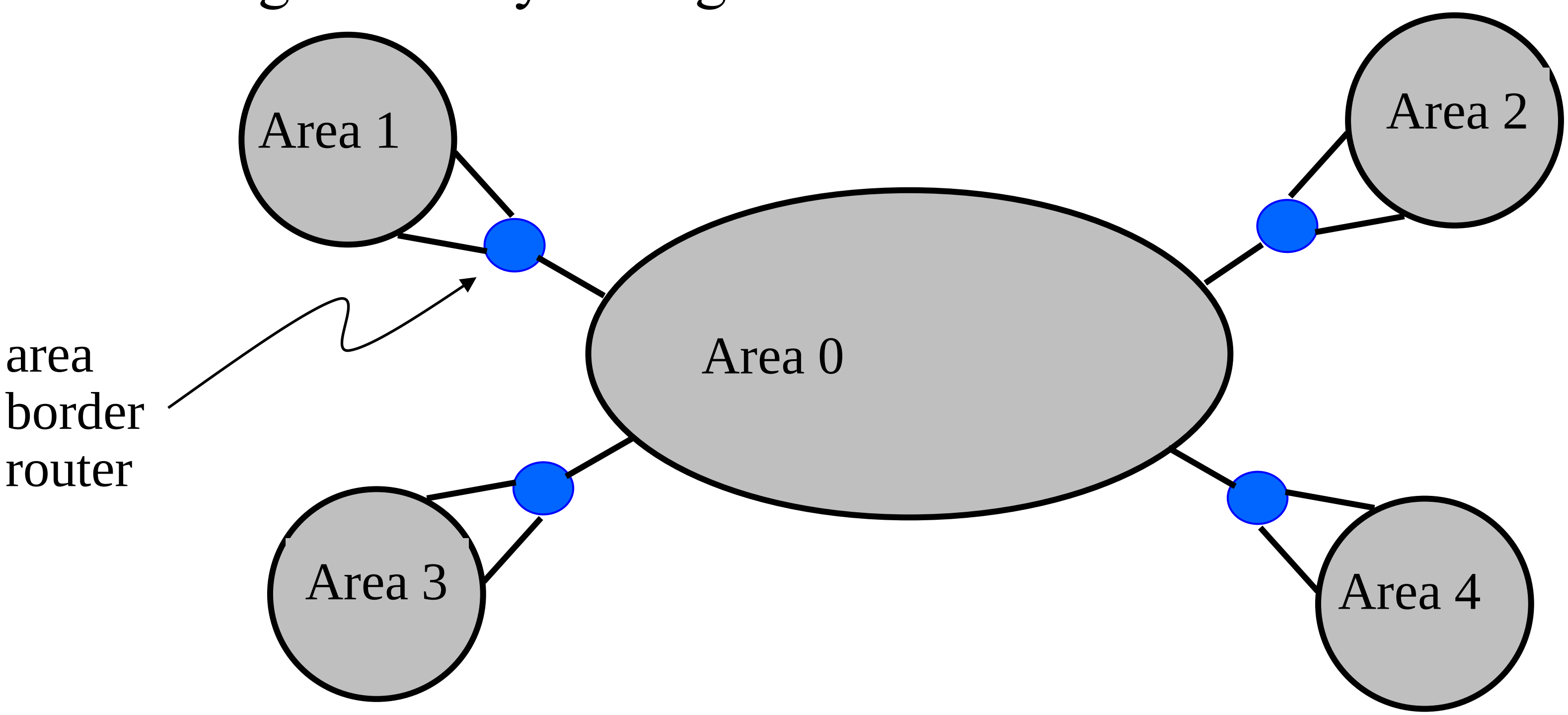
- How to Flood the link state?
  - » every node sends link-state information through adjacent links
  - » next nodes forward that info to all links
    except the one where the information arrived



- When to initiate flooding?
  - » Topology change
    - – link or node failure/recovery
    - – link cost change
  - » Periodically
    - – refresh link-state information
    - – typically 30 minutes

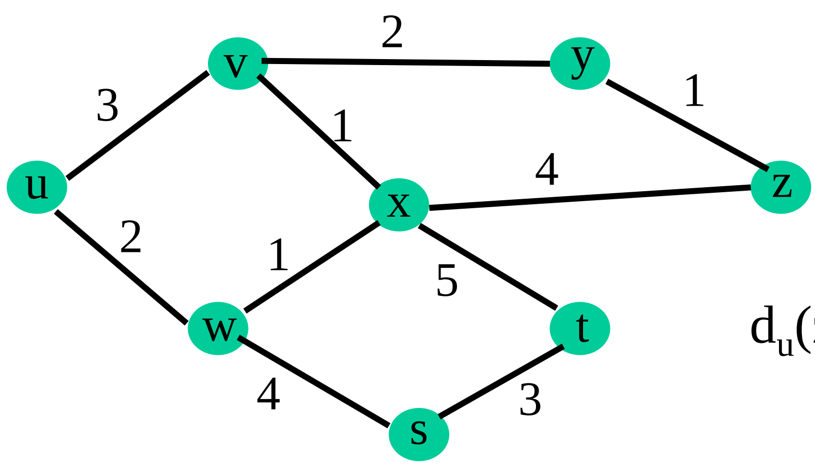
Scaling Link-State Routing

- Overhead of link-state routing
  - » flooding link-state packets throughout the network
  - » running Dijkstra's shortest-path algorithm
- Introducing hierarchy through "areas"



Bellman-Ford Algorithm

- Define distances at each node x
  - » $d_x(y)$ = cost of least-cost path from x to y
- Update distances based on neighbors
  - » $d_x(y) = \min \{c(x,v) + d_v(y)\}$ over all neighbors v



$$d_u(z) = \min\{c(u,v) + d_v(z),\ c(u,w) + d_w(z)\}$$

Distance Vector Algorithm

- $c(x,v)$ = cost for direct link from x to v

  node x maintains costs of direct links $c(x,v)$

- $D_x(y)$ = estimate of least cost from x to y

  node x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$

- Node x maintains also its neighbors' distance vectors

  for each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

- Each node v periodically sends D_v to its neighbors
  - » and neighbors update their own distance vectors
  - » $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ for each node $y \in N$

- Over time, the distance vector D_x converges

Distance Vector Algorithm

- Iterative, asynchronous

  each local iteration caused by:

  - local link cost change
  - distance vector update message from neighbor

- Distributed

  » node notifies neighbors

  only when its DV changes

- Neighbors then

  notify their neighbors,

  if necessary

Each node:

wait for (change in local link cost or message from neighbor)

recompute estimates

if DV to any destination has changed, *notify* neighbors

Distance Vector Example - Step 0

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	(A)	A	4	A
B	4	B	B	0	(B)
C	∞	–	C	∞	–
D	∞	–	D	3	D
E	2	E	E	∞	–
F	6	F	F	1	F

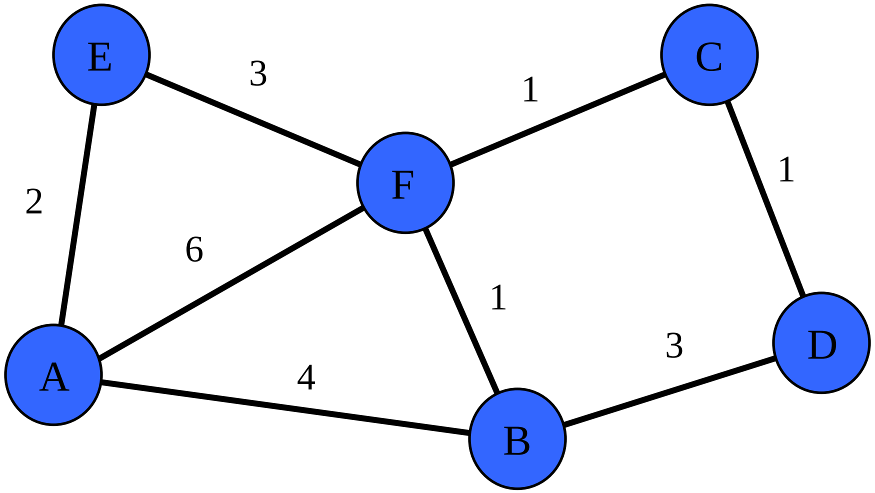


Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	∞	–	A	∞	–	A	2	A	A	6	A
B	∞	–	B	3	B	B	∞	–	B	1	B
C	0	(C)	C	1	C	C	∞	–	C	1	C
D	1	D	D	0	(D)	D	∞	–	D	∞	–
E	∞	–	E	∞	–	E	0	(E)	E	3	E
F	1	F	F	∞	–	F	3	F	F	0	(F)

Distance Vector Example - Step 1

Table for A

Dst	Cst	Hop
A	0	(A)
B	4	B
C	7	F
D	7	B
E	2	E
F	5	E

Table for B

Dst	Cst	Hop
A	4	A
B	0	(B)
C	2	F
D	3	D
E	4	F
F	1	F

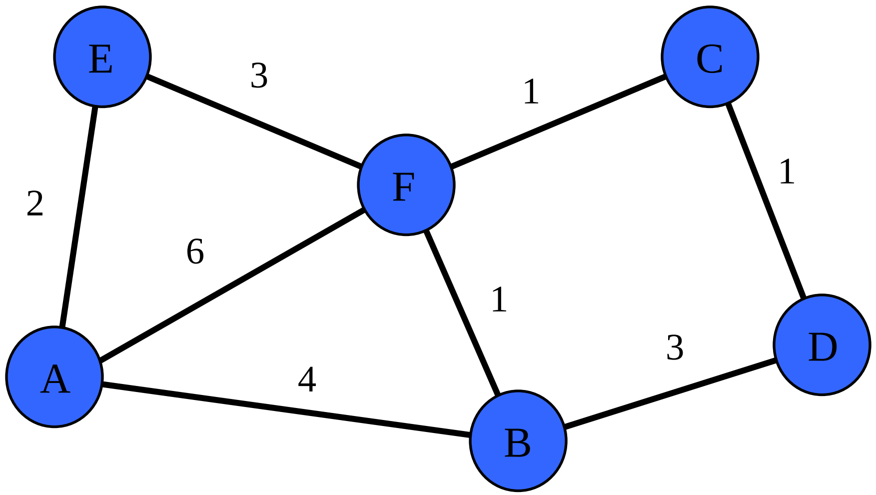


Table for C

Dst	Cst	Hop
A	7	F
B	2	F
C	0	(C)
D	1	D
E	4	F
F	1	F

Table for D

Dst	Cst	Hop
A	7	B
B	3	B
C	1	C
D	0	(D)
E	∞	–
F	2	C

Table for E

Dst	Cst	Hop
A	2	A
B	4	F
C	4	F
D	∞	–
E	0	(E)
F	3	F

Table for F

Dst	Cst	Hop
A	5	B
B	1	B
C	1	C
D	2	C
E	3	E
F	0	(F)

Distance Vector Example - Step 2

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	(A)	A	4	A
B	4	B	B	0	(B)
C	6	E	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F

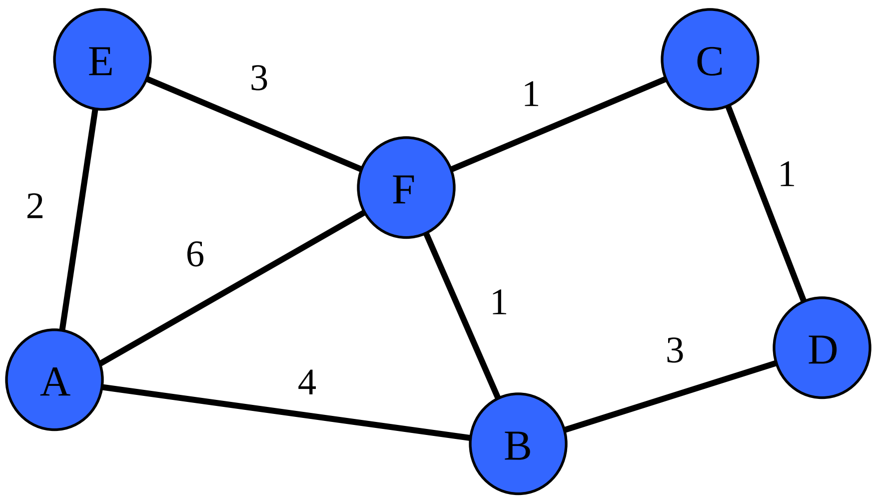


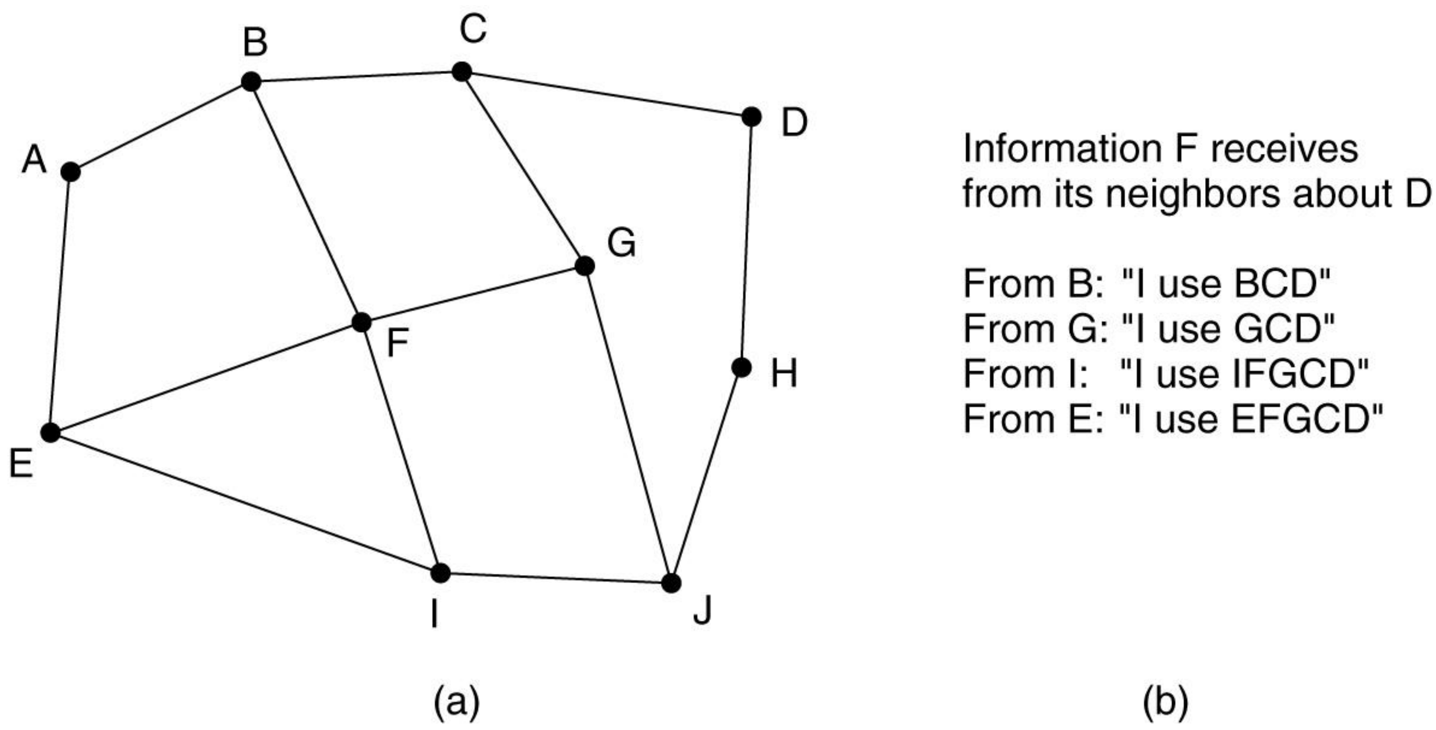
Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	6	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	(C)	C	1	C	C	4	F	C	1	C
D	1	D	D	0	(D)	D	5	F	D	2	C
E	4	F	E	5	C	E	0	(E)	E	3	E
F	1	F	F	2	C	F	3	F	F	0	(F)

Demo: https://www.dcc.fc.up.pt/~rprior/RC/dvrp/

Routing Information Protocol (RIP)

- Distance vector protocol
  - nodes send distance vectors every 30 seconds
  - or, when an update causes a change in routing
- RIP is limited to small networks

BGP – The Exterior Gateway Routing Protocol

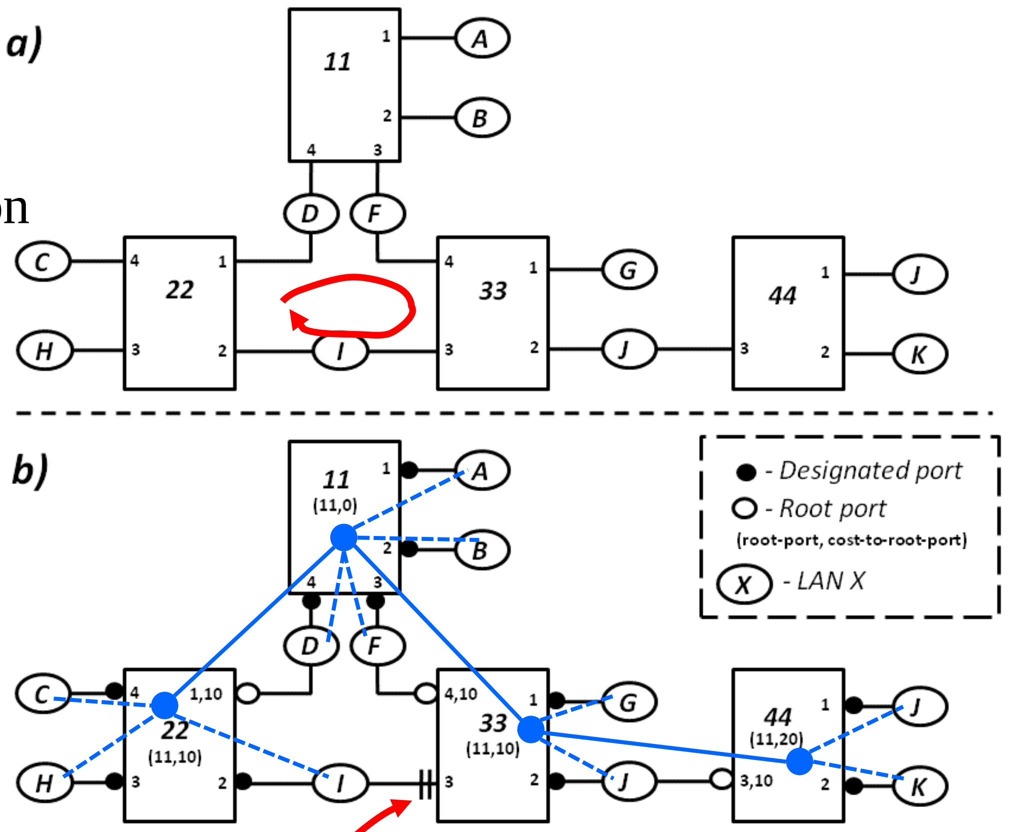


(a) A set of BGP routers. (b) Information sent to F

Unique Spanning Tree in Ethernet Networks

L2 Networking - Single Tree Required

- Ethernet frame
  - *No hop-count*
  - Could *loop* forever
  - Broadcast frame, misconfiguration
- Layer 2 network
  - **Required to have tree topology**
  - Single path between every pair of stations
- Spanning Tree Protocol (STP)
  - Running in bridges
  - Helps building the spanning tree
  - Blocks ports



Constructing a Spanning Tree

Distributed algorithm

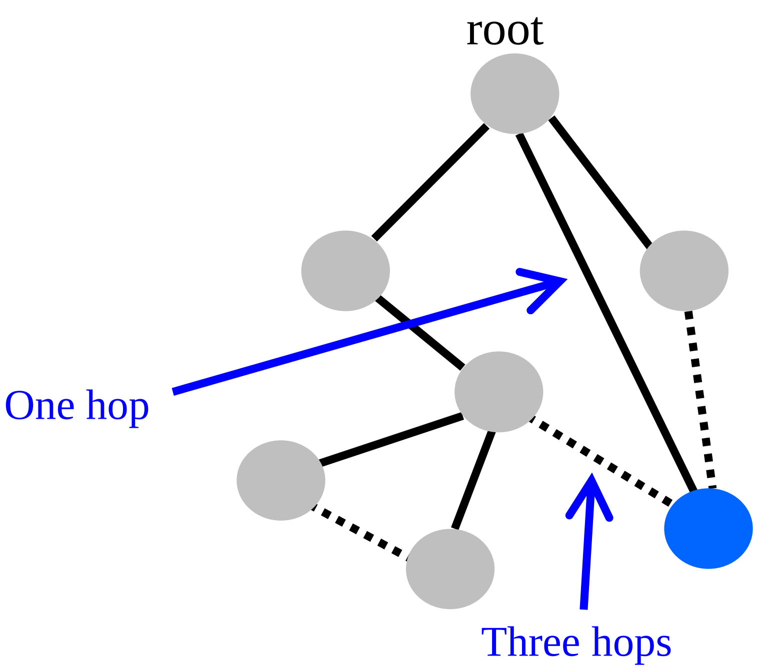
» switches need to elect a "root"

the switch with the smallest identifier

» each switch identifies if its interface is on the shortest path from the root

» messages (Y, d, X)

- from node X
- claiming Y is the root
- and the distance is d

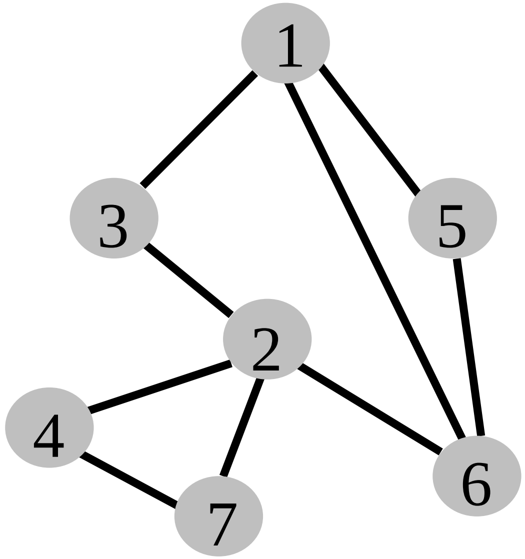


Steps in Spanning Tree Algorithm

- Initially, each switch thinks it is the root
  - switch sends a message out every interface
  - identifying itself as the root with distance 0
  - example: switch X announces (X, 0, X)
- Other switches update their view of the root
  - upon receiving a message, check the root id
  - if the new id is smaller, start viewing that switch as the root
- Switches compute their distance from the root
  - add interface *cost* to the distance received from a neighbor
  - identify interfaces not on a shortest path to the root and exclude them from the spanning tree

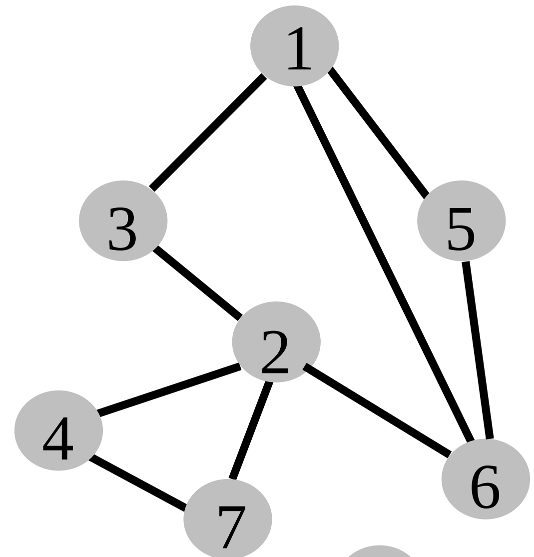
Example - Switch #4's Viewpoint

- Switch #4 thinks it is the root
  - » sends (4, 0, 4) message to 2 and 7
- Then, switch #4 hears from #2
  - » receives (2, 0, 2) message from 2
  - » … and thinks that #2 is the root
  - » and realizes it is just one hop away
- Then, switch #4 hears from #7
  - » receives (2, 1, 7) from 7
  - » and realizes this is a longer path
  - » so, prefers its own one-hop path
  - » and removes 4-7 link from the tree

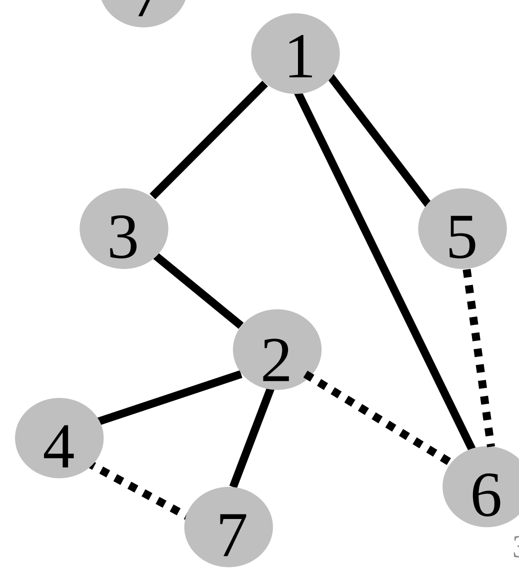


For simplicity, consider all costs = 1

Example - Switch #4's Viewpoint



- Switch #2 hears about switch #1
  - switch 2 hears (1, 1, 3) from 3
  - switch 2 starts treating 1 as root
  - and sends (1, 2, 2) to neighbors
- Switch #4 hears from switch #2
  - switch 4 starts treating 1 as root
  - and sends (1, 3, 4) to neighbors

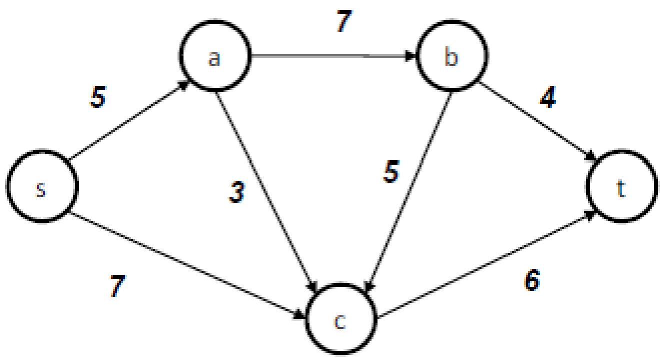


- Switch #4 hears from switch #7
  - switch 4 receives (1, 3, 7) from 7
  - and realizes this is a longer path
  - so, prefers its own three-hop path
  - and removes 4-7 link from the tree

Maximum Flow of a Network

Flow Network Model

- Flow network
  - » source s
  - » sink t
  - » nodes a, b and c
- Edges are labeled with **capacities**
  - » **(e.g. bit/s)**



- Communication networks are not flow networks
  - » they are queue networks
  - » flow networks allow us to determine limit capacity values

Maximum Capacity of a Flow Network

- Max-flow min-cut theorem
  - » maximum amount of flow transferable through a network equals minimum value among all simple cuts of the network

- Cut ➔ split of the nodes V into two disjoint sets S and T
  - » $S \cup T = V$
  - » there are $2^{|V|-2}$ possible cuts

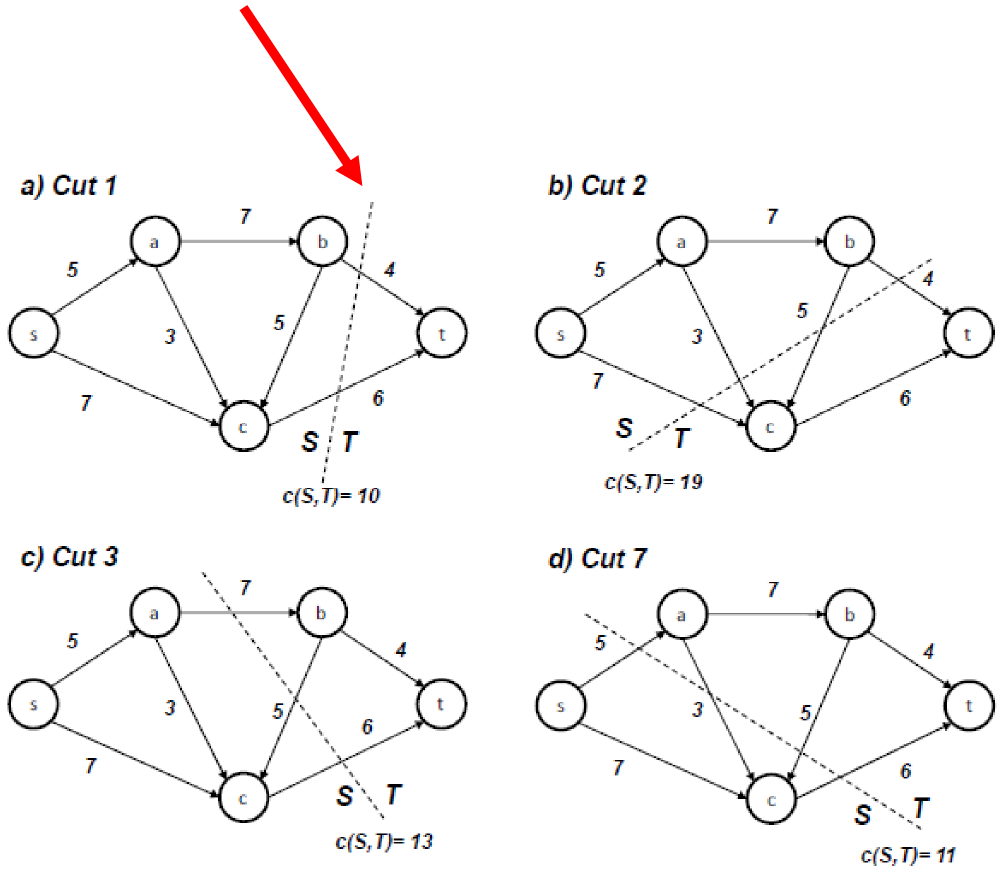
- Capacity of cut (S, T): $c(S,T) = \sum_{(u,v) \mid u \in S, v \in T, (u,v) \in E} c(u,v)$

Max-flow Min-cut - Example

$2^{|5|-2} = 8$ possible cuts

Cut	Vertices					$c(S,T)$	Feasability
	s	a	b	c	t		
1	S	S	S	S	T	10	✓
2	S	S	S	T	T	19	✓
3	S	S	T	S	T	13	✓
4	S	S	T	T	T	17	✓
5	S	T	S	S	T	-	×
6	S	T	S	T	T	-	×
7	S	T	T	S	T	11	✓
8	S	T	T	T	T	12	✓

Maximum flow = 10



Homework

1. Review slides

2. Read from Tanenbaum
   » Section 5.2 – Routing algorithms
   » Section 4.8.3 – Spanning Tree Bridges