Imperial College London

Computer Networks and Distributed Systems

Tutorial – Bridging and Routing Protocols

Course 527 – Spring Term 2015-2016

Emil C Lupu and Daniele Sgandurra

e.c.lupu@imperial.ac.uk, d.sgandurra@imperial.ac.uk

Part 4 – Contents

Spanning Tree Protocol (Layer 2)

Used by switches to turn a redundant topology into a **spanning tree:**

- Avoid and **eliminate loops** in the network by negotiating a loop-free path through a **root bridge.**
- Disables unwanted links by **blocking** ports.
- Ensures that there will be **only one active path** to every destination.

STP defined by **IEEE 802.1d.**

Switches run STP by default – no configuration needed.

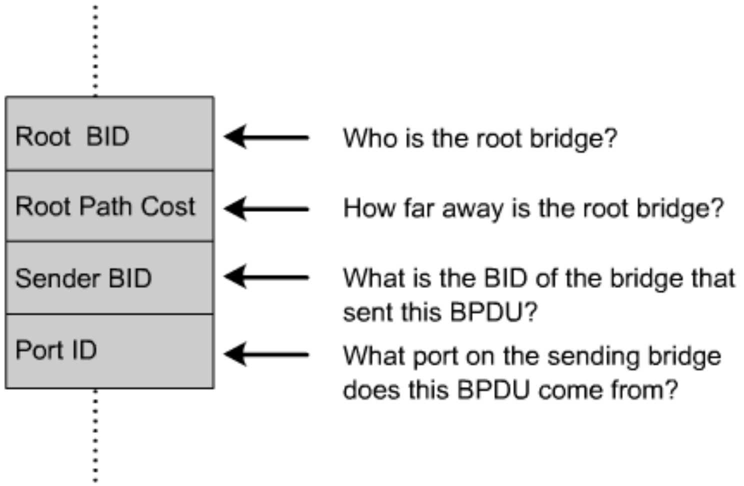
Spanning Tree Algorithm

The switches use this algorithm to decide which ports should be shut down.

1. Choose one switch to be "**root bridge**"
2. Choose a "**root port**" on each other switch
3. Choose a "**designated port**" on each segment.
4. **Close down** all other ports.

See step-by-step example at:
http://www.cisco.com/image/gif/paws/10556/spanning_tree1.swf

BPDU (Bridge Protocol Data Units)



Root and Designated Ports

Root Port: This is the port that is the closest to the root bridge in terms of path cost.

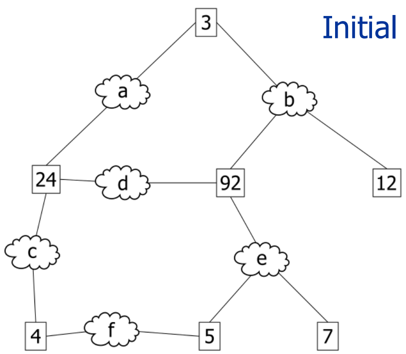
Designated port: it can send the best BPDU on the segment to which it is connected.

- On a **given segment**, there can be **only one path toward the root bridge** otherwise redundant paths would create a bridging loop.

- All bridges connected to a given segment listen to each other's BPDUs and agree on the bridge sending the best BPDU as the designated bridge for the segment.

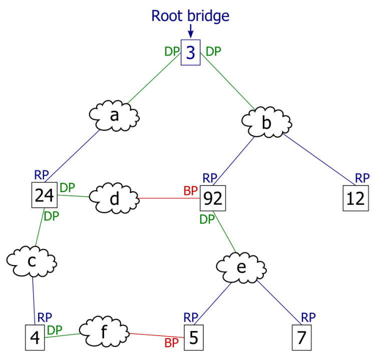
A Root Port can **never** be a Designated Port.

Spanning Tree Example



Initial State

Final State (after STP)



Distance Vector Protocol (Layer 3)

A **distance vector routing** algorithm operates by having each router maintain a table (i.e., vector):

- **Best known distance to each router.**
- **Which link to use to get there.**

Each table is indexed by, and containing one entry for each router in the network:

- Preferred outgoing line for that destination.
- Estimate of the distance to that destination.

Distance Vector Protocol

A router is assumed to know the **distance** (cost) to each of its neighbors.

Once every T_{msec}, each router sends to each neighbor a list of its **estimated delays** to each destination:

- It also receives a similar list from each neighbor.

If R has neighbor X:

- Table from X: X_i= estimated delay to get to router i.
- If delay to X estimated by R is m:
  - R can reach i in $m+X_i$ msec.

Bellman-Ford Algorigthm

$c(x,v)$ = 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 $D_x = [D_x(y): y \in N]$.

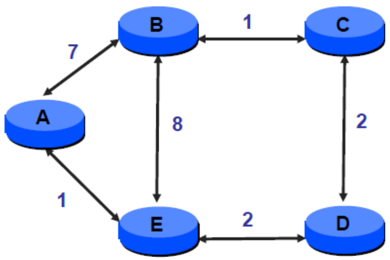
Node x maintains its neighbors' distance vectors:

- For each neighbor v, x maintains $D_v = [D_v(y): y \in N]$.

Each node v periodically sends D_v to its neighbors:

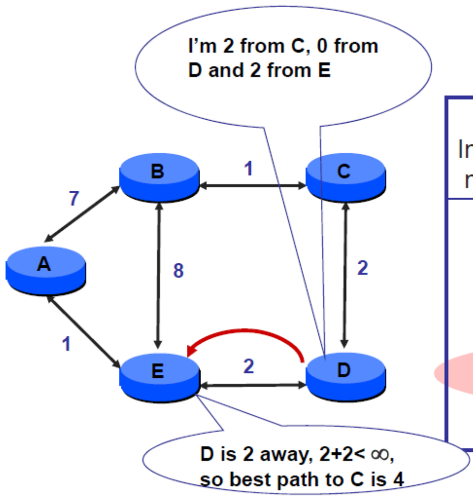
- And neighbors update their own distance vectors.
- $D_x(y) \leftarrow min_v\{c(x,v) + D_v(y)\}$ for each node $y \ \varepsilon \ N$.

Example: Initial State



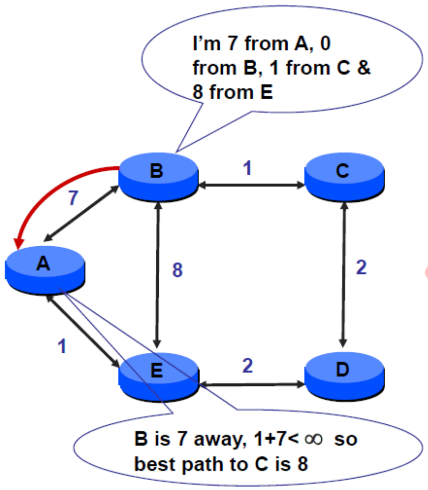
Info at node	Distance to Node A	B	C	D	E
A	0	7	∞	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	∞	2	0

D Sends Vector to E



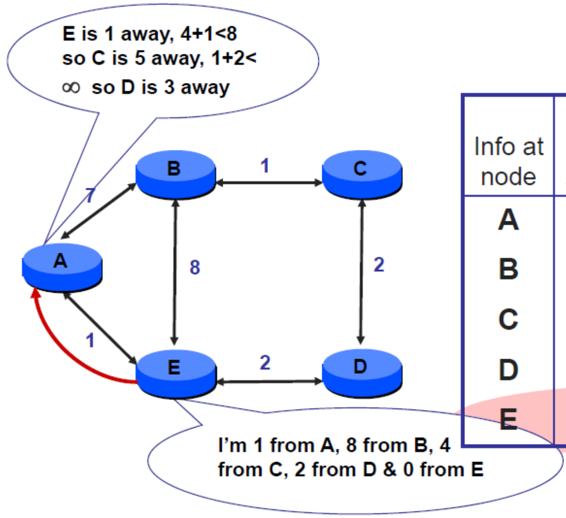
Info at node	Distance to Node A	B	C	D	E
A	0	7	∞	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	4	2	0

B Sends Vector to A



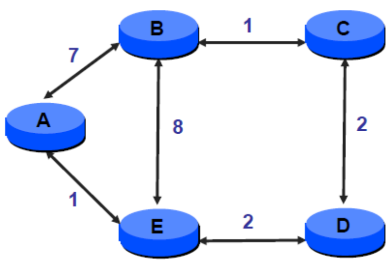
Info at node	Distance to Node A	B	C	D	E
A	0	7	8	∞	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	4	2	0

E Sends Vector to A



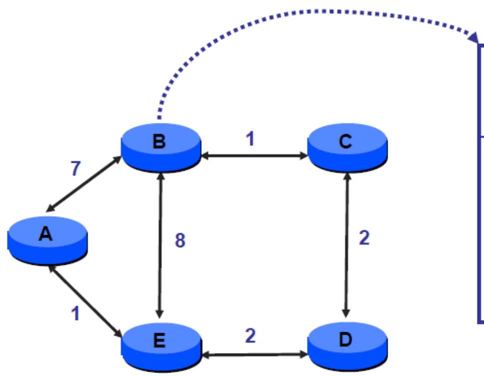
Info at node	Distance to Node A	B	C	D	E
A	0	7	5	3	1
B	7	0	1	∞	8
C	∞	1	0	2	∞
D	∞	∞	2	0	2
E	1	8	4	2	0

Until Convergence



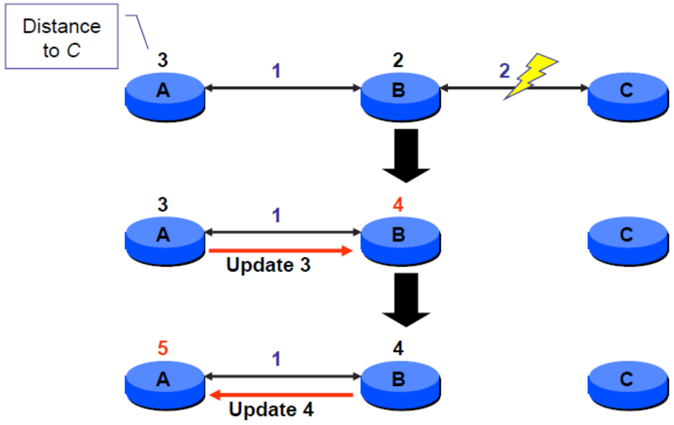
Info at node	Distance to Node A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

Node B's Distance Vector



Dest	Next hop A	E	C
A	7	9	6
C	12	12	1
D	10	10	3
E	8	8	5

Counting to Infinity Problem



Etc…

Link-State Routing (Layer 3)

Same goal as Distance Vector, but different approach:

- Tell **every** node what you know about your **direct neighbors.**

Two phases:

- **Reliable flooding:**
  - Tell all routers what you know about your local topology.
- **Path calculation** (Dijkstra's algorithm):
  - Each router computes best path over complete network.

Reliable-Flooding

Each router transmits a **Link State Packet** on all links:

- Identifier of the sender.
- List of neighbors with their associated costs.

A neighboring router **forwards** out all links except incoming.

Keep a copy locally; don't forward previously-seen LSPs.

Caveats:

- Acknowledgments and retransmissions.
- Sequence numbers.
- Time-to-live for each packet.

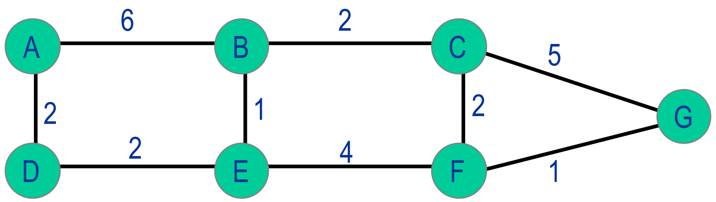
Running Dijkstra Algorithm

Determine the **shortest distance** between a start node and any other node in a graph.

```
function Dijkstra(Graph, source):
   for each vertex v in Graph:          // Initialization
      dist[v] := infinity               // Initial distance from source to v is set to ∞
      previous[v] := undefined          // Previous node in optimal path from source
   dist[source] := 0                    // Distance from source to source
   Q := the set of all nodes in Graph   // All nodes in the graph are unoptimized
   while Q is not empty:                // main loop
      u := node in Q with smallest dist[ ]
      remove u from Q
      for each neighbor v of u:         // where v has not yet been removed from Q.
         alt := dist[u] + dist_between(u, v)
         if alt < dist[v] // Relax (u,v)
            dist[v] := alt
            previous[v] := u
    return previous[ ]
```

Dijkstra's LSR Algorithm

Consider the following network:



Link state database:

A	
B	6
D	2

B	
A	6
C	2
E	1

C	
B	6
F	2
G	5

D	
A	2
E	2

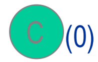
E	
B	1
D	2
F	4

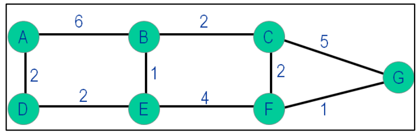
F	
C	2
E	4
G	1

G	
C	5
F	1

Dijkstra's LSR Algorithm

Running Dijkstra on C

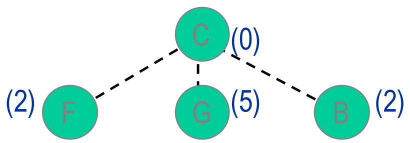


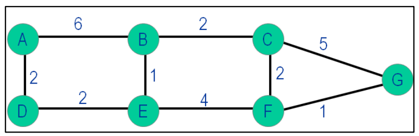


Dijkstra's LSR Algorithm

Examine C's LSP

- Cost to F, G, and B

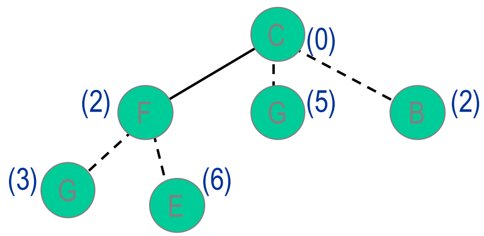


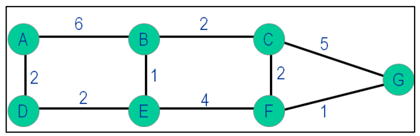


Dijkstra's LSR Algorithm

Pick node in Q with lowest dist: F

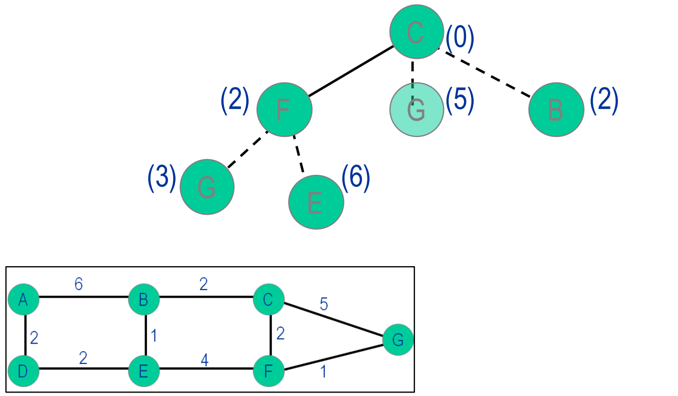
- Its predecessor is determined: C (cannot be changed)





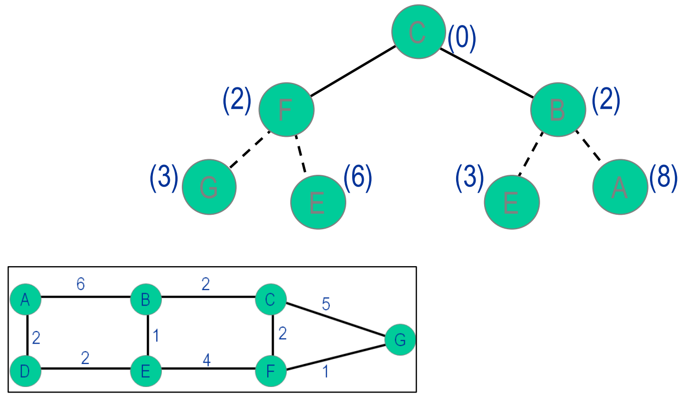
Dijkstra’s LSR Algorithm

Update G cost and predecessor



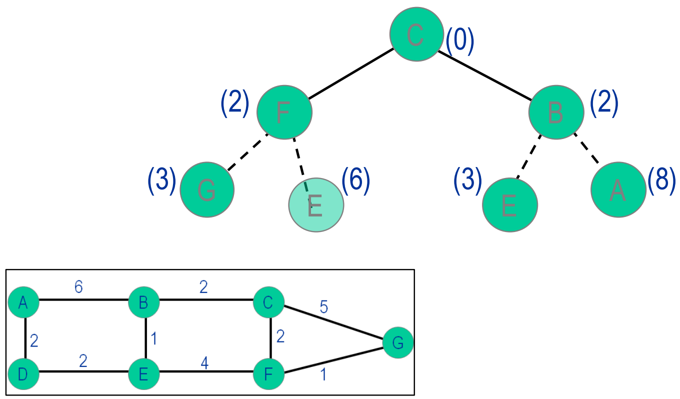
Dijkstra’s LSR Algorithm

Pick node in Q with lowest dist: B



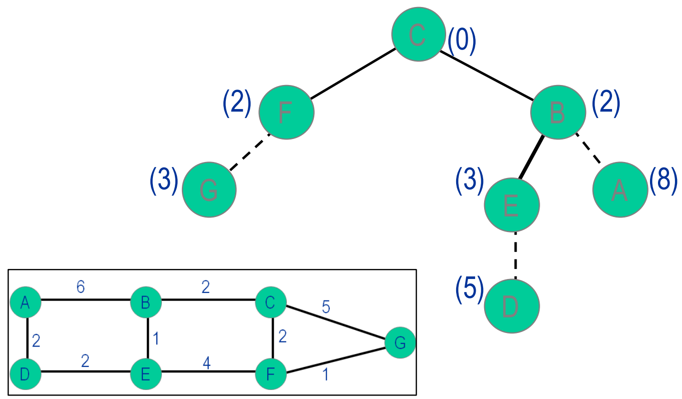
Dijkstra’s LSR Algorithm

Update E cost and predecessor



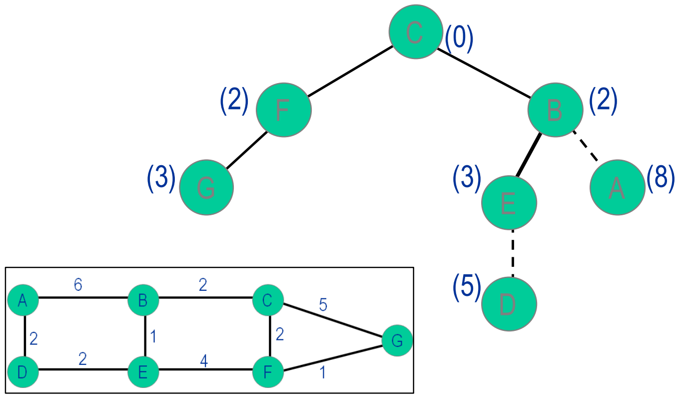
Dijkstra’s LSR Algorithm

Pick node in Q with lowest dist: E



Dijkstra’s LSR Algorithm

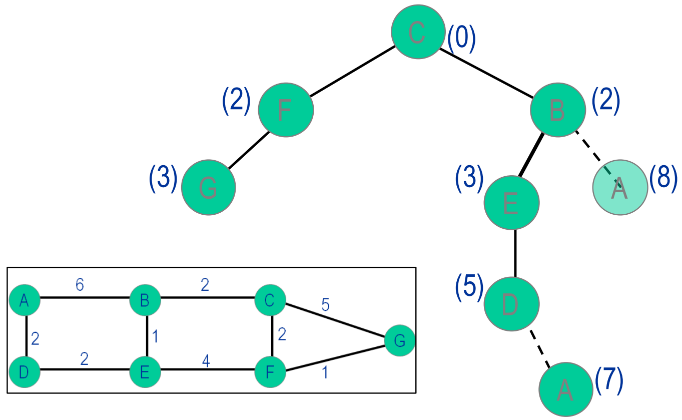
Pick node in Q with lowest dist: G



Dijkstra’s LSR Algorithm

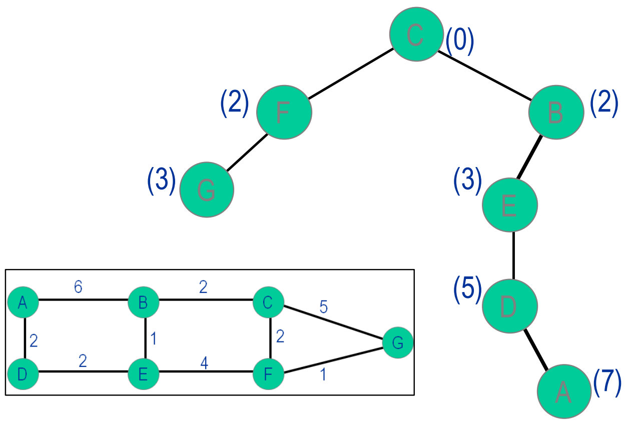
Pick node in Q with lowest dist: D

– Update A cost and predecessor



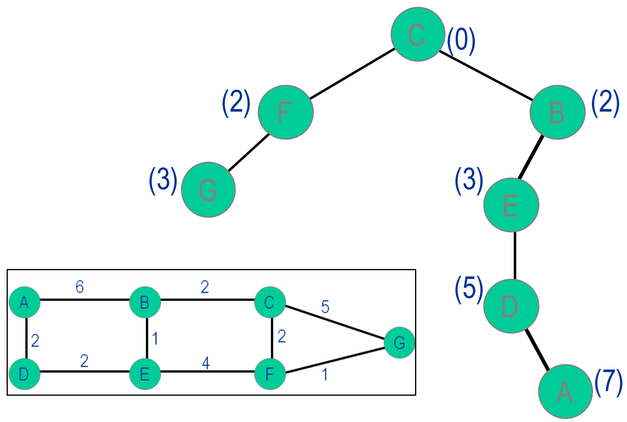
Dijkstra’s LSR Algorithm

Pick node in Q with lowest dist: A



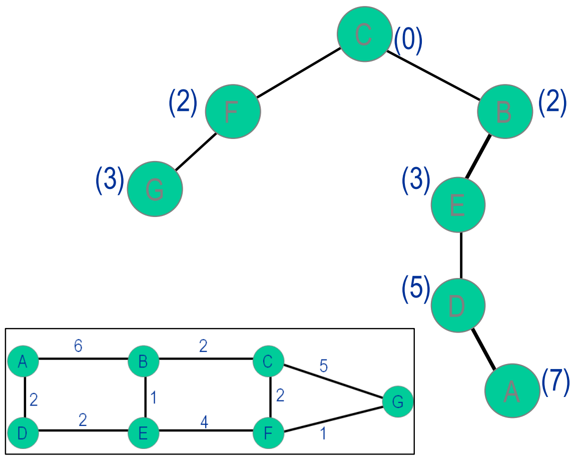
Dijkstra’s LSR Algorithm

Q is empty: DONE



Dijkstra's LSR Algorithm

We can now create a routing (forwarding) table for C:



Routing Table	
Destination	**Port**
C	C
F	F
G	F
B	B
E	B
D	B
A	B