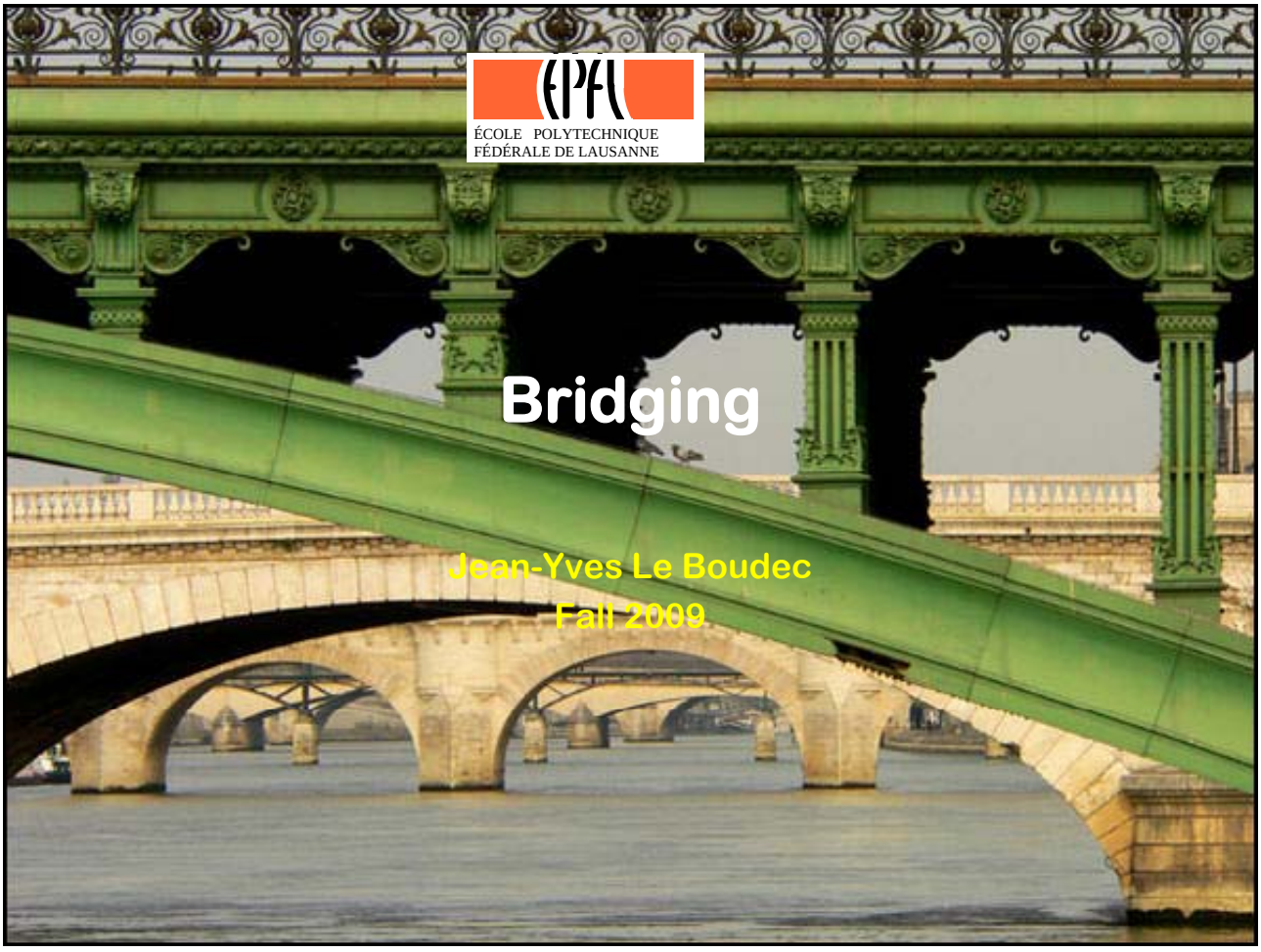

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Bridging
Jean-Yves Le Boudec
Fall 2009

Algorhyme

I think that I shall never see
a graph more lovely than a tree.
A tree whose crucial property
is loop-free connectivity.
A tree that must be sure to span
so packet can reach every LAN.
First, the root must be selected.
By ID, it is elected.
Least-cost paths from root are traced.
In the tree, these paths are placed.
A mesh is made by folks like me,
then bridges find a spanning tree.

Radia Perlman

Contents

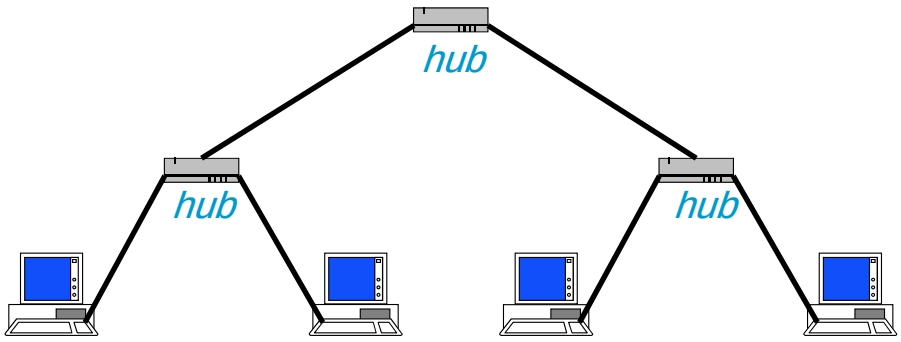
- 1.Transparent bridging
- 2. Spanning Tree Protocol (STP)
    a. specification
    b. an exotic version of Bellman-Ford
    c. the STP protocol

1. Transparent Bridging
Bridging in General

- Bridges are intermediate systems that forward MAC frames to destinations based on MAC addresses
- Interconnect systems beyond one LAN segment, keeping main characteristics of LAN
  - without additional addresses
    - MAC addresses used to identify end systems
  - preserve sequence integrity
- The LAN segments can be of different nature
  - Ex: WiFi and Ethernet
- There are several possible methods, only one is wide-spread: Transparent Bridging

Transparent Bridging (TB)

- End systems ignore that there are transparent bridges
  - bridge is transparent
  - MAC frames not changed by bridges
  - frames not sent *to* bridge, but rather: bridge is promiscuous
    - (listens to all frames)
- Bridges are required to be *plug and play* (i.e. no configuration by system manager)
- Q. Is an Ethernet hub a bridge or a repeater ? What's the difference ? solution



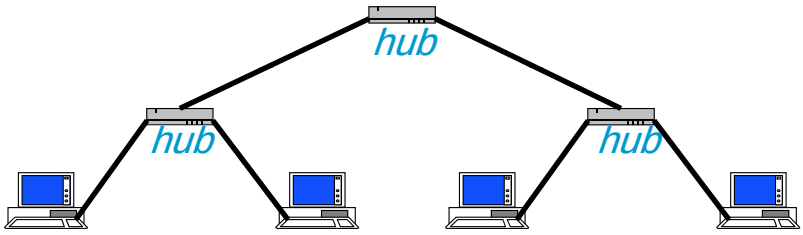
5

Transparent Bridging (TB)

- End systems ignore that there are transparent bridges
  - bridge is transparent
  - MAC frames not changed by bridges
  - frames not sent *to* bridge, but rather: bridge is promiscuous
    - (listens to all frames)
- Q. Is an Ethernet hub a bridge or a repeater ? What's the difference ?
- A. It can be either a bridge or a repeater. "hub" is a product name, not an architecture name. Modern hubs are bridges. Old ones are repeaters.

  The difference is: a repeater is a layer 1 intermediate system (acts on bits) whereas a bridge is a layer 2 intermediate system (acts on entire MAC frames). Also: a bridge separates collision domains, a repeater does not

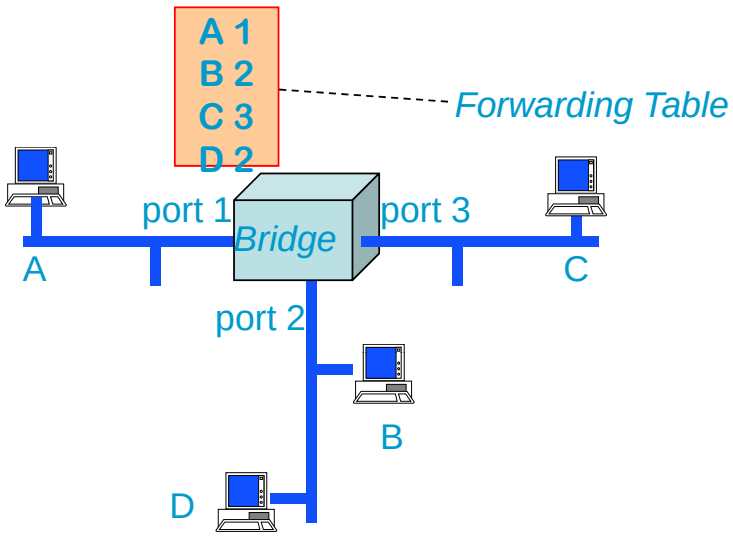
back



6

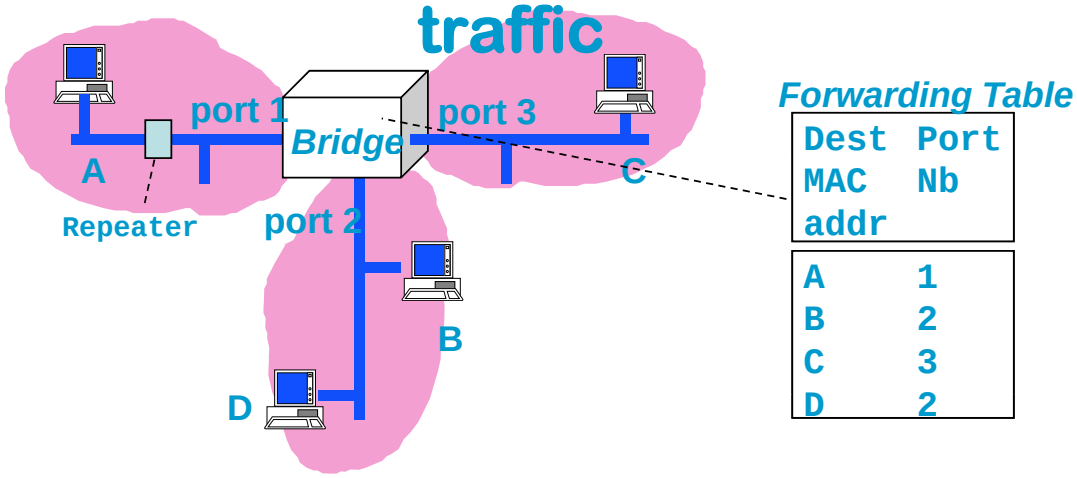
Transparent Bridging uses forwarding tables

- Table maps MAC addresses to port numbers
  - No IP addresses here !



7

Bridges learn addresses by observing traffic

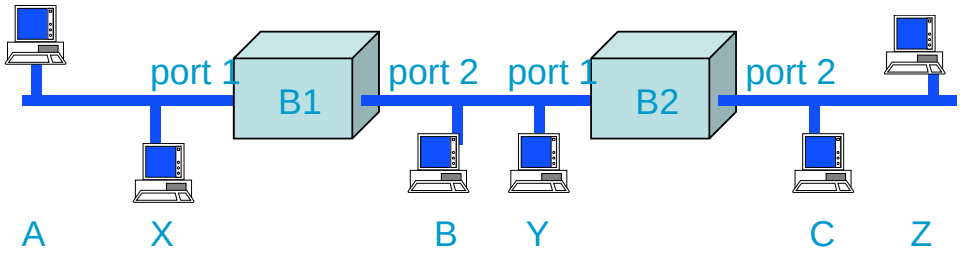


- How can a bridge build its table ?
  - No equivalent to routing protocols, we need a plug and play solution
- Bridge builds routing table by reading all traffic
  - table built by **learning** from SA field in MAC frame
  - learnt addresses times out if not re-learnt
- If destination address not in table broadcast to all ports
  - same for group addresses

8

Can this method of learning addresses be extended to a network of bridges?

- On this example, yes.
- Q. How does B2 see the network?
  solution



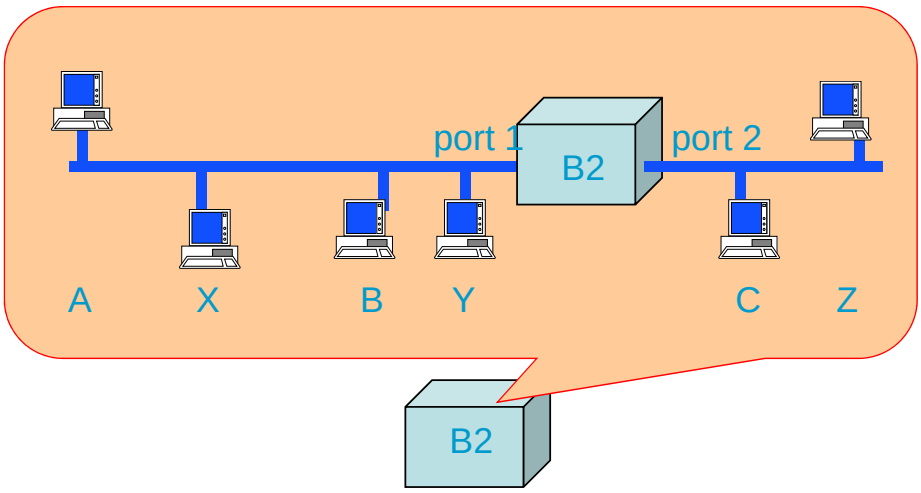
9

Can this method of learning addresses be extended to a network of bridges?

- On this example, yes.
- Q. How does B2 see the network?
  A. B2 sees that A, X, B and Y are on port 1 (B1 is transparent !)
  Its forwarding table is

A 1
B 1
C 2
X 1
Y 1
Z 2

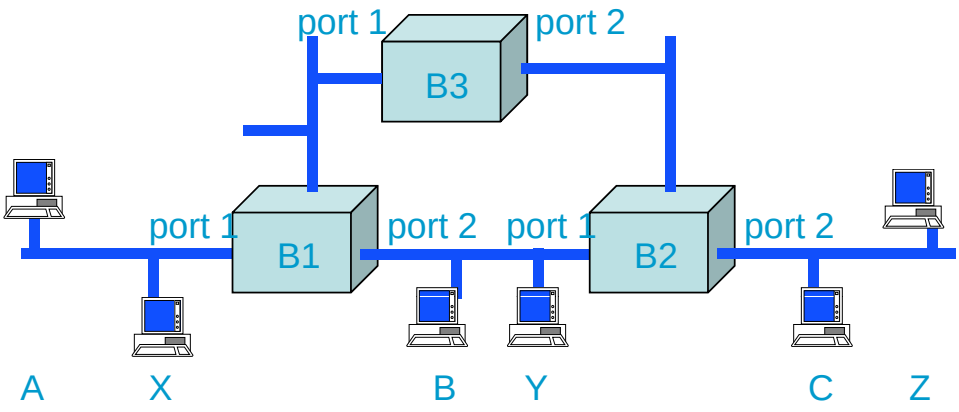
back



10

The method of learning does not work if there are loops in the topology

- Q. What happens when A send a frame to B?
  - assume empty forwarding tables at the beginning
    solution



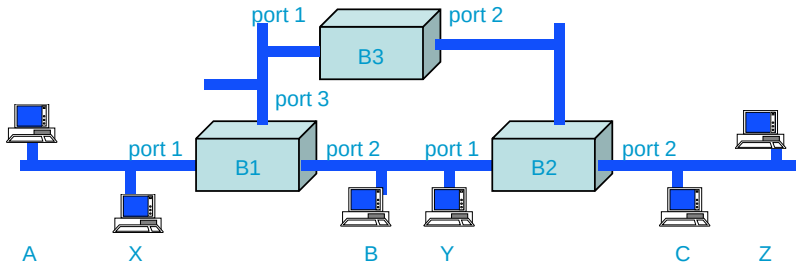
11

The method of learning does not work if there are loops in the topology

- Q. What happens when A send a frame to B?
  - assume empty forwarding tables at the beginning

A. frame is sent by B1 to ports 2 and 3. B1 learns that A is on port 1. B3 sends it to port 2. B2 sends it to ports 1 and 2. B1 now learns that A is on port 2. B1 sends frame to ports 1 and 3 etc… the frame is multiplied a number of times. B receives several copies

back



Transparent Bridges force the Active Topology to be loop-free

- Learning bridge works well if there is no loop in the topology
  - The topology can be represented as a bidirectional graph where vertex = bridge, edge = connection through collision domains (called here: LAN)
  - for such graphs : Loop- free **and** connected $\equiv$ tree
  - On a tree, there is only one path from one host to one bridge. Therefore, a bridge sees a host on exactly one port.
- A network of bridges may have redundant connections (as in previous example). This is good for reliability, but this causes loops in the topology.
- The solution adopted by transparent bridges is: maintain an ***active topology*** that is loop-free
  - i.e. decide that some ports are blocked
  - This should be done automatically, without configuration (plug and play)

13

The Spanning Tree Protocol

- What does it do ?
  - Prevent loops in the active topology
  - Decide which ports should be blocked or opened
    - ports that are allowed to forward frames are said to be “in the forwarding state” or called “forwarding ports”
  - Adapt to changes in the physical topology
- How does it work ?
  - See next section

14

Summary: what a transparent bridge does

Individual PDU forwarding

```
Copy all frames on all forwarding ports

Frame received on port i ->
                    /* port i is forwarding */
If DA is unicast, is in forwarding table with
       port j and j is a forwarding port
       then copy to port j
       else flood all forwarding ports ≠ i
Update forwarding table with (i, SA)
```

Control Method

```
Run the Spanning Tree Protocol
```

2. The Spanning Tree Protocol

We present the Spanning Tree Protocol in 4 steps :

(a) Specification

(b) Design of main algorithm

(c) Main Protocol

(d) Topology Changes and Synchronization with Packet Forwarding

(a) Specification

(a) We now specify the STP method (ie ***what*** it does, in more details than before, not ***how***)

- There are many ways to build a tree on a graph.
  - Minimum Spanning Tree (Kruskal or Prim's algorithms)
  - The STP chose to use the set of shortest paths towards some selected vertex.
- Each bridge has a **bridge label,** based on MAC address + configurable offset. Bridge with smallest label is selected and called "root".

  Each LAN between bridges has a **cost**, by default, decreasing function of bit rate:

Port Type	Duplex	Cost
100BASE-TX / 100BASE-FX (VLT)	Full	5
	Half	12
10BASE-T	Full	6
	Half	700

- ***What:*** The STP computes a tree of shortest paths to the root bridge

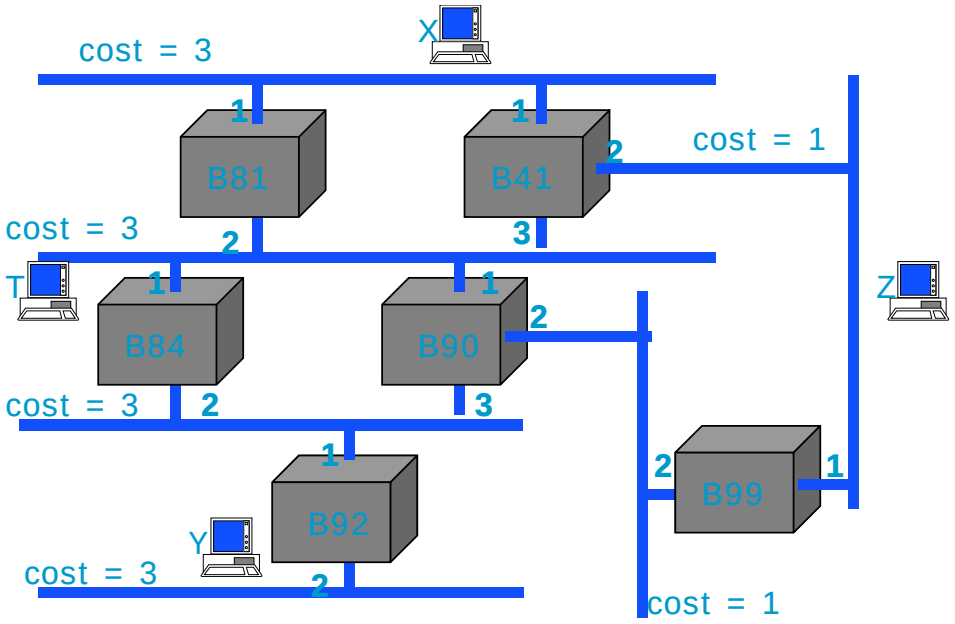
17

Specification of STP (cont'd)

- STP gives a *role* to all ports
  - Root or designated (ports on spanning tree)
  - Blocked (ports not on spanning tree)
- Root ports
  - One per bridge
  - := port towards root along shortest path
  - in case of equal costs, lowest port id chosen
- Designated ports
  - On every LAN (≡ collision domain), choose one *designated bridge*
  - all ports on LAN for which the bridge is designated are designated ports
  - Designated bridge
    - one per LAN
    - defined by : it has the shortest path to root
    - possibly root itself
- Ports other than root or designated are blocking

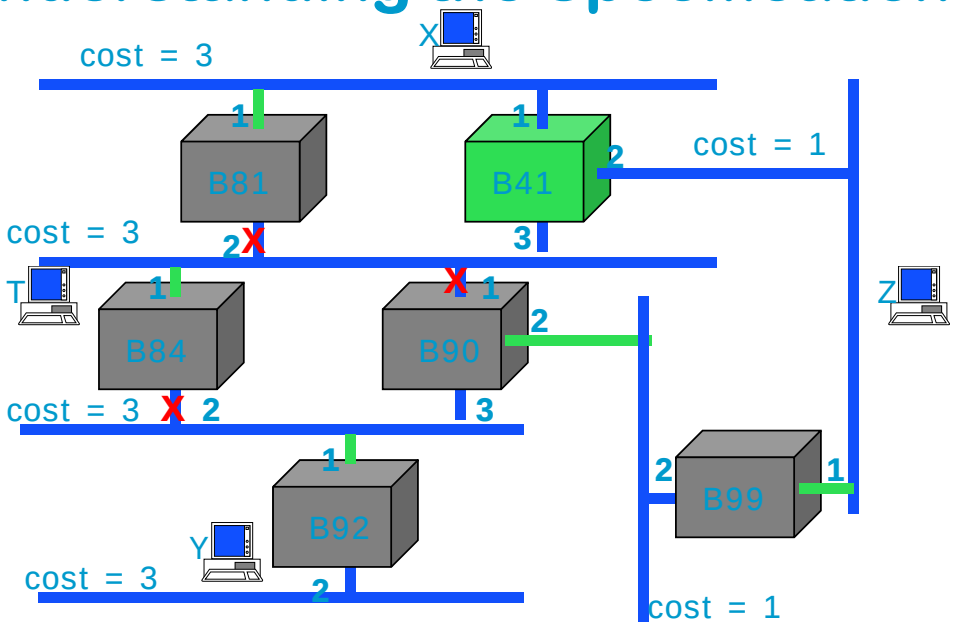
Understanding the Specification

- Q1. find the root, root ports, designated bridges, designated ports and blocking ports
- Q2. find the forwarding table at all bridges
  solution



19

Understanding the Specification



back

root port designated port

blocking port

Forwarding Tables:	
B41 1X 2YZ 3T	B81 1XYZT
B84 1XYZT	B90 2XZT 3Y
B92 1XZT 2Y	B99 1XZT 2Y

20

(b) STP: design of main algorithm

- STP uses a variant of the Bellman-Ford algorithm (see dv.ppt), which we call the *Bellman-Ford algorithm for Bridges*
- Like the original Bellman-Ford algorithm, it is a distributed algorithm, but it is easier to understand it by studying first the centralized version

21

The Centralized Bellman Ford Algorithm for Bridges uses Special Link and Path Attributes

- Represent the network by a graph, where a vertex is a bridge. For this algorithm, we treat the graph as directed.
- We are given *link costs* c(i,j) (costs of LANs, see later for default values)
  - We assume c(i, j) > 0 and c(i,j) = ∞ when i and j are not connected.
- We are also given vertex *labels* l(i)(concatenation of bridge priority and serial number, set by manufaturer)
- **Definition:** a link *attribute* is
  - A(i,j) := [l(j), c(i,j)]
  - *concatenation* of attributes:
    - [l,c] $\oplus$ [l', c'] = [min(l,l'), c+c']
    - the *attribute* of a *path* i1 i2 … ik is the concatenation of the attributes of the links – i.e.: [minimum label, sum of costs]
  - *comparison* of attributes
    - lexicographic: [l, c] $\leq$ [l' c'] iff [(l < l') or (l = l' and c $\leq$ c')]
    - a total order on N x [0, ∞]
  - A path *p* is better than *p'* if *attribute of p* $\leq$ *attribute of p'*

22

Examples of Path Attributes

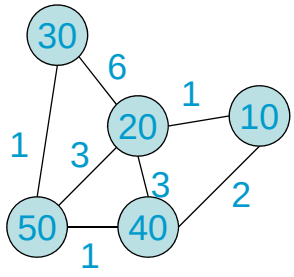
- Q. Q. What are the attribute of the following paths ? Which one is best ?

  $30 \rightarrow 50 \rightarrow 40$

  $30 \rightarrow 20 \rightarrow 10 \rightarrow 40$

  $30 \rightarrow 20 \rightarrow 10$

  $30 \rightarrow 50 \rightarrow 40 \rightarrow 10$



solution

Examples of Path Attributes

- Q. What are the attribute of the following paths ?

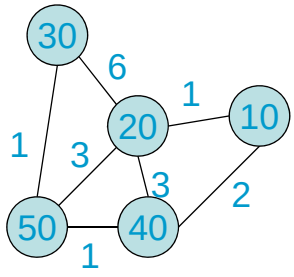
  $30 \rightarrow 50 \rightarrow 40$ [**40**, 2]

  $30 \rightarrow 20 \rightarrow 10 \rightarrow 40$ [**10**, 9]

  $30 \rightarrow 20 \rightarrow 10$ [**10**, 7]

  $30 \rightarrow 50 \rightarrow 40 \rightarrow 10$ [**10**, 4]

  The best path is the last.



back

What are "Best Paths" in this Setting ?

- Assume
  - the graph is fully connected;
  - all vertex labels are different;
  - all link costs are > 0
- A path p that starts at i is ***best*** (among all paths that start at i) iff
  - It goes through the vertex i_0 that has the smallest label in the graph (the minimum label is reached at only one vertex, by hypothesis)
  - It stops at i_0
  - It is a shortest path from i to i_0
- Thus: the best paths in this graph are the shortest paths to the node with the smallest label

The Centralized Bellman-Ford Algorithm for Bridges

- **What:** Given a directed graph with links attributes as above, computes one tree of best paths from any vertex

  let A(i,j):= attribute of link (i,j) =[l(j), c(i,j)]

- **How:** Define $p^k(i)$ as the cost of the best path from i to anywhere in at most k hops.

$$p^0(i) = [l(i), 0] \ \forall i$$

for $k = 1, 2, \ldots$ do

  for all i

$$p^k(i) = \min\left\{[l(i), 0], \min_{j \neq i}\left[A(i,j) \oplus p^{k-1}(j)\right]\right\} \quad (1)$$

until $p^k = p^{k-1}$

- **Theorem**
  1. If the graph is fully connected, the algorithm stops at the latest at k=number of vertices ; at the end, $p^k(i)$ is the attribute of a best path
  2. A best path from *i* is obtained by letting pred(i) = the index (j or i) that achieves the minimum in (1)

     If the min is achieved by the term [l(i),0] then pred[i]=i; this happens only when vertex i has the smallest label

- The algorithm is the same as the classical Bellman-Ford algorithm [dv.ppt], with the following modifications
  - **Exotic algebra** instead of usual algebra: costs are replaced by attributes; addition of costs is replaced by concatenation ($\oplus$) and comparison by the lexicographic order.
  - **All paths** instead of paths to a specific node: add a virtual node 0 such that A(i,0)=[l(i), 0] and A(0,i)=[∞, ∞]. Apply the classical Bellman-Ford to compute the "shortest" (i.e. best) paths from all nodes i to node 0. Remove the final edge from these paths and obtain the best paths we are looking for.

  Indeed, with these modifications, the classical Bellman-Ford becomes

$$p^0(i) = [\infty, \infty] \ \forall i \neq 0$$
$$p^0(0) = [\infty, 0]$$

for $k = 1, 2, \ldots$ do
  $p^k(0) = [\infty, 0]$
  for all $i \neq 0$
    $p^k(i) = \min_{j \neq i} \left[A(i,j) \oplus p^{k-1}(j) \right]$ (2)
until $p^k = p^{k-1}$

One can easily see that (2) is equivalent to (1), given that we set $p^{k-1}(0)$ to [∞, 0], and that the impact of the initialization for $p^0(i)$ disappears after one step.

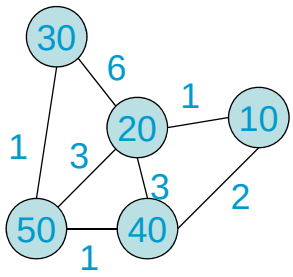
- Note: in the algorithm, "min" is the lexicographic min (derived from the comparison of attributes)
- The proof of the algorithm is similar to the classical case. It relies on the fact that $\oplus$ is associative.

27

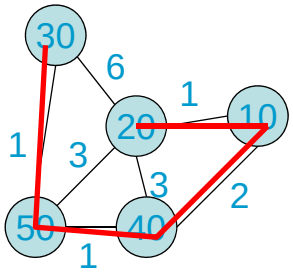
Run the Centralized Bellman Ford Algorithm for Bridges on this Example

- Q. Write $p^k(i)$, pred(i) and draw the spanning tree.

solution



Run the Centralized Bellman Ford Algorithm for Bridges on this Example



$p^k(i)$: (format: (label, cost))

k \ i	10	20	30	40	50
0	10,0	20,0	30,0	40,0	50,0
1	10,0	10,1	20,6	10,2	20,3
2	10,0	10,1	10,7	10,2	10,3
3	10,0	10,1	10,4	10,2	10,3

i	10	20	30	40	50
pred(i)	10	10	50	10	40

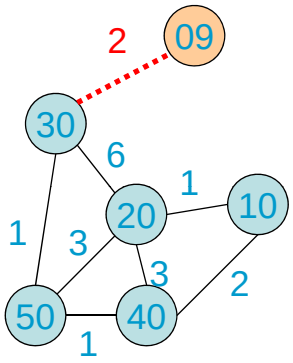
back

Impact of Initial Conditions on the Bellman-Ford Algorithm for Bridges

- The classical Bellman-Ford algorithm continues to work if we take different initial conditions
  - but the interpretation that $p^k(i)$ is the distance from i to 1 in at most k hops is no longer true
- Does this still hold for the Bellman-Ford algorithm for Bridges ?

Q. write $p^k(i)$, pred(i) and draw the spanning tree, with initial conditions as shown. The dotted link does not exist in the current configuration. It existed before, and explains why node 30 starts with these initial conditions.

solution



$p^k(i)$: (format: label, cost)

k \ i	10	20	30	40	50
0		10,0	20,0	09,2	40,0	50,0
1					
2					
3					

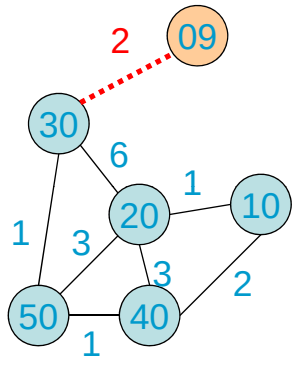
Impact of Initial Conditions on the Bellman-Ford Algorithm for Bridges

Q. write $p^k(i)$, pred(i) and draw the spanning tree, with initial conditions as shown. The dotted links do not exist in the current configuration. They existed before, and explain why nodes 30 and starts with these initial conditions.

A. no, it does not work.

After a few steps, all nodes believe the best label is 09, and start computing the best path towards 09. Then they start a count to infinity (we are computing the usual distance to 09, which is infinite). The algorithm does not converge.

back



$p^k(i)$: (format: label, cost)

k \ i	10	20	30	40	50
0		10,0	20,0	09,2	40,0	50,0
1		10,0	09,8	20,6	10,2	09,3
2		09,9	09,6	09,4	09,4	09,11
3		09,16	09,7	09,12	09,9	09,10
4		09,8	09,12	09,11	09,10	09,10
5		09,12	09,9	09,11	09,10	09,10
6		09,10	09,13	09,11	09,11	09,11

The Bellman-Ford Algorithm for Bridges is sensitive to initial conditions

Theorem

- If the initial conditions in the centralized Bellman-Ford Algorithm for Bridges satisfy:

$$\forall i : \quad p^0(i)=(m_i, c_i) \text{ with } m_i \geq \min_j l(j)$$

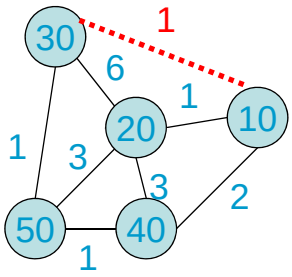
the algorithm converges to the correct value

- else the algorithm diverges
  - with $\lim_{k\to\infty} p^k(i)=(m_0,\infty)$ where $m_0=\min_i m_i$

Proof: first show that the label converges to the minimum of all initial conditions (it can only decrease). Then use the property of Bellman-Ford in the usual algebra (see chapter "distance vector")
Comment: the convergence may be much longer than with the initial conditions in theorem "All-path" variant of Bellman Ford"
Note that there is a condition on the initial label, not on the initial cost.

Example

- Q. write $p^k(i)$, pred(i) and draw the spanning tree, with initial conditions as shown. The dotted link does not exist in the current configuration.
  It existed before, and explains why node 30 starts with these initial conditions.
  Does the algorithm converge to the correct values ?



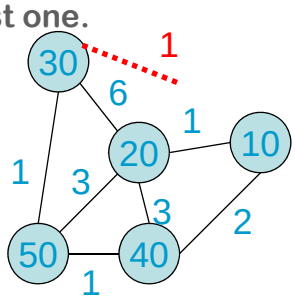
$p^k(i)$: (format: (label, cost))

k \ i	10	20	30	40	50
0	10,0	20,0	10,1	40,0	50,0
1					
2					
3					

solution

Example

- Q. write $p^k(i)$, pred(i) and draw the spanning tree, with initial conditions as shown. The dotted link does not exist in the current configuration.
  It existed before, and explains why node 30 starts with these initial conditions.
  A. The algorithm converges since the initial labels are not below the smallest one.



$p^k(i)$: (format: (label, cost))

k \ i	10	20	30	40	50
0		10,0	20,0	10,1	40,0	50,0
1		10,0	10,1	20,6	10,2	10,2
2		10,0	10,1	10,3	10,2	10,3
3		10,0	10,1	10,4	10,2	10,3
4		10,0	10,1	10,4	10,2	10,3

back

Distributed Bellman-Ford Algorithm for Bridges

- Like the classical Bellman-Ford (i.e. BFD2 in dv.ppt), the Bellman-Ford Algorithm for Bridges can be distributed: It is the algorithm used by STP

every node, say i, maintains an estimate q(i) of p(i), the attribute of a best path from i and of pred(i), the next node on a best path;

initially q(i)=[l(i),0] and pred(i)=i

from time to time, i sends its value q(i) to all its neighbours

when node i receives a value q(j0) from *any neighbour* j0, it sets q(j0) to the received value and updates q(i) by recomputing

eq (3) **if** j0 == pred(i)
then q(i):=min { $A(i,j_0) \oplus q(j_0)$, [l(i),0] }
else q(i) := min{$A(i,j_0) \oplus q(j_0)$, q(i) }

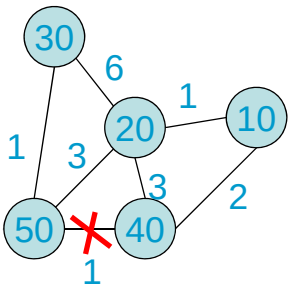
if eq (3) causes q(i) to be modified, pred(i) is set to j_0

if A(i,pred(i)) changes (including if pred(i) stops being a neighbour) then q(i) is set to [l(i),0] and pred(i) is set to i.

35

Sample Run of the Distributed Bellman-Ford Algorithm for Bridges

A possible run :



i	10	20	30	40	50
	10,0	20,0	30,0	40,0	50,0
10 -> 20		10,1			
50 -> 20		10,1			
20 -> 50					10,4
20 -> 30			10,7		
50 -> 40				10,5	
20 -> 40				10,4	
10 -> 40				10,2	
40 -> 50					10,3
50 -> 20					
50 -> 30			10,4		
link breaks	50 does as if received q(40)= (∝,∝); pred(50)=40 thus 50 does q(50)=(50,0); similarly 40 does a new computation but this does not change 40				
				10,2	50,0
50 -> 30			30,0		
20 -> 50					10,4
50 -> 30			10,5		

The Distributed Bellman-Ford Algorithm for Bridges may need to be reset

- Like the centralized algorithm, the distributed algorithm is robust to changes in configuration as long as the node with the smallest label (called "root bridge") is still present and reachable from all bridges.

  If this is not true, the algorithm does not converge to a true value. It needs to be ***reset*** by some additional mechanism.

Q. Compare to the classical distributed Bellman-Ford algorithm.
solution

37

The Distributed Bellman-Ford Algorithm for Bridges may need to be reset

- Like the centralized algorithm, the distributed algorithm is robust to changes in configuration as long as the node with the smallest label (called "root bridge") is still present and reachable from all bridges.

  If this is not true, the algorithm does not converge to a true value. It needs to be ***reset*** by some additional mechanism.

Q. Compare to the classical distributed Bellman-Ford algorithm.
back

A. It does not need to be reset, since it always converges to the true value.

(c) STP – Main Protocol

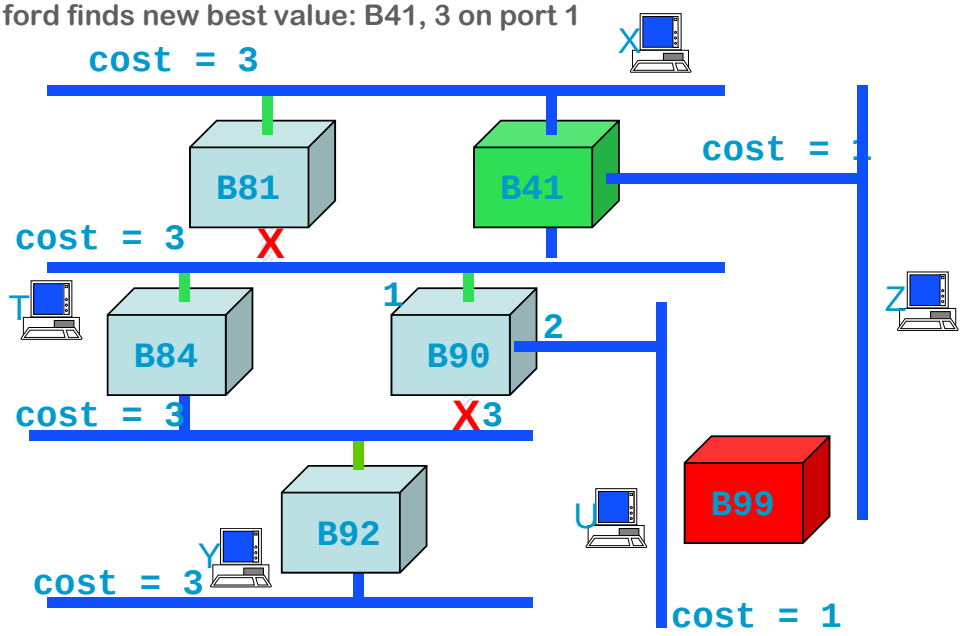
- Standardized by IEEE 802.1D
- All bridges run it
- Implements the Distributed Bellman Ford Algorithm for Bridges
  - Bridge keeps best values received on all ports
  - Bridge periodically sends its values to neighbours, and whenever a change occurs

Topology changes

- Topology changes occur due to
  - changes in configuration
  - failures, recoveries
- It changes occur, the behaviour depends on whether the root bridge is still reachable from all bridges
  - if so, let distributed Bellman Ford do the job
  - else, we need some additional mechanism: STP uses root monitoring for this:
    - root refreshes validity of STP by periodically sending a refresh message every HelloTime (2s)
    - the refresh message is propagated along the spanning tree
    - a bridge that does not receive refresh message for MaxAge restarts STP basic procedure from fresh initial conditions (= reset)

A Topology Change That is Handled by Bellman-Ford

- B99 crashes; focus on **B90**
  - B90 detects absence of B99 (absence of hello, or other mechanism); this is equivalent to receiving (in Bellman-Ford's algorithm) a state information: "from B99: best attribute (∞, ∞)
  - B90 compares all values received so far on all ports
    - Port 1: best = B41, 3; port 2 = ∞, ∞ port 3: best = B90,6
    - Bellman ford finds new best value: B41, 3 on port 1



A Topology Change That is *not* Handled By Bellman-Ford

- Q: If B41 dies, what happens ?
- A: root monitoring at all bridges detect that B41 does not send a refresh message anymore

  all bridges start the STP procedure from fresh initial conditions and converge to a new spanning tree rooted at B81

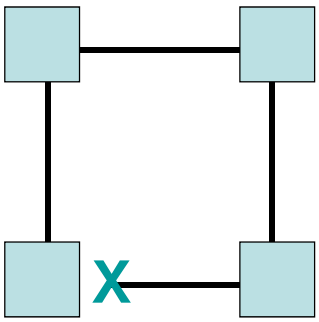
42

Other Bells and Whistles

- Bridges wait for some time after any topology change before declaring the port as « forwarding » (15--45 secs)
  - To avoid loops during transients
- Optimizations of STP (called « Rapid STP », RSTP) avoid the timers in some frequent cases
  - Detects that the change cannot cause a loop
  - See « Rapid Spanning Tree » on www.cisco.com

43

Efficiency ?



Algorhyme

I think that I shall never see
a graph more lovely than a tree.
A tree whose crucial property
is loop-free connectivity.
A tree that must be sure to span
so packet can reach every LAN.
First, the root must be selected.
By ID, it is elected.
Least-cost paths from root are traced.
In the tree, these paths are placed.
A mesh is made by folks like me,
then bridges find a spanning tree.

Radia Perlman

- No loops, but paths may be not optimal
  - All frames go through the spanning tree

Conclusions

- Bridges use STP to remove loops from the active topology
- An example of bio-like software
  - All bridges have the same code, only one becomes root
  - No central intervention, plug and play
- The Bellman Ford algorithm of Bridges repairs any failures except loss of root
  - Handled by a separate keep-alife mechanism; loss of root causes a global reset
- RSTP is an optimization to speed up impact of topology changes
  - The active topology is the same as with STP
- To know more:
  - Radia Perlman, « Interconnections, Bridges and Routers »
  - CISCO RSTP White Paper

45