CSE/EE 461: Introduction to Computer Communications Networks
Winter 2010

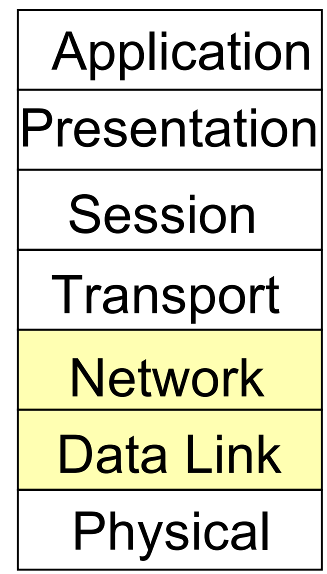
Module 4
Bridging LANs

John Zahorjan
zahorjan@cs.washington.edu
534 Allen Center

This Module: Bridging / Switching

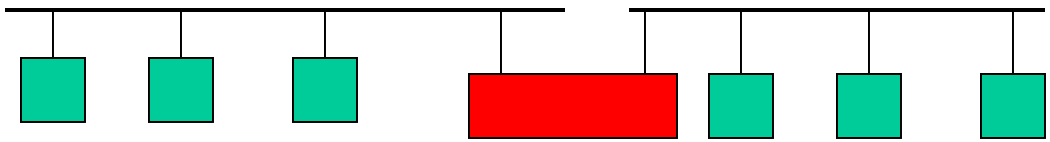
- Focus:
  - What to do when one shared LAN isn't big enough?

- Interconnecting LANs
  - Bridges and LAN switches
  - A preview of the Network layer

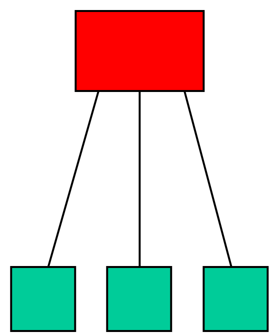


Terminology / Pictures are a little confusing

Original Ethernet(repeater)

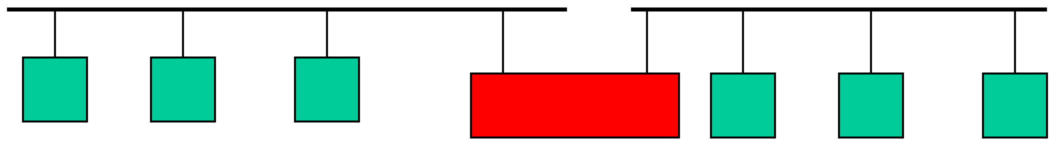


Modern Ethernet
(Hub)

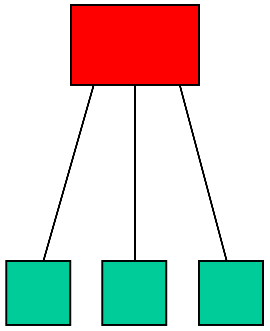


Not talking more about these today

Instead, we'll be talking about these



Bridges



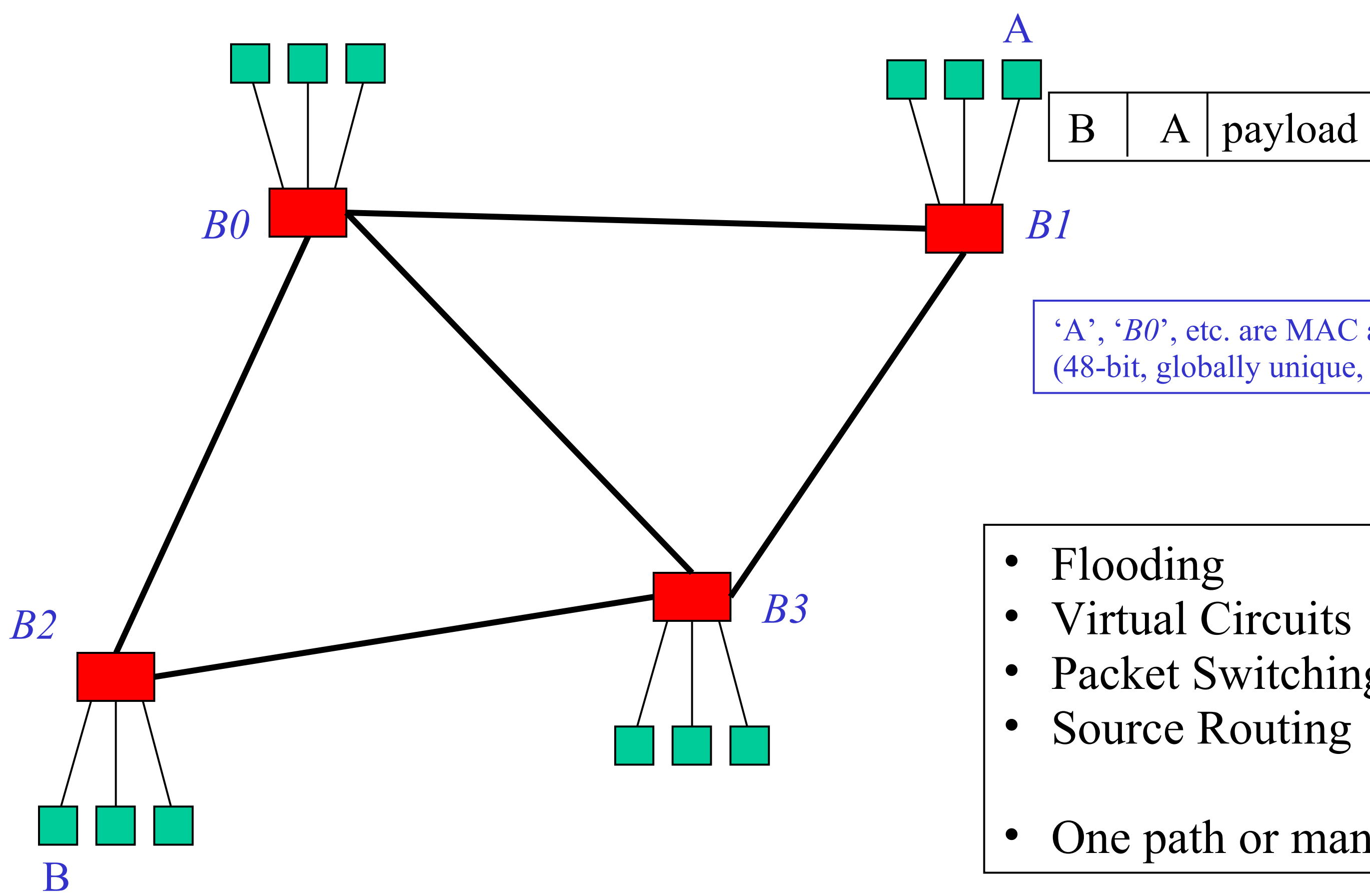
The Common Theme: Limits of a LAN

- One shared LAN can limit us in terms of:
  - Distance
  - Number of nodes
  - Performance

- How do we scale to a larger, faster network?
  - We must be able to interconnect LANs
  - Don't want to pass all packets by every host
    - Bridges/switches must make sensible choices about which outgoing links to place packets on

- For the system architectures we're most interested in, some packet buffering will take place
  - Store and forward

Some Choices



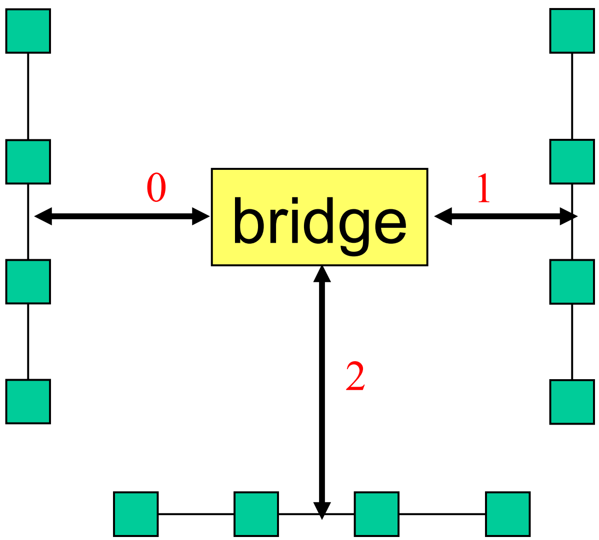
'A', '*B0*', etc. are MAC addresses
(48-bit, globally unique, 802.2, for instance)

- Flooding
- Virtual Circuits
- Packet Switching
- Source Routing

- One path or many?

First Realization: Bridges and Extended LANs

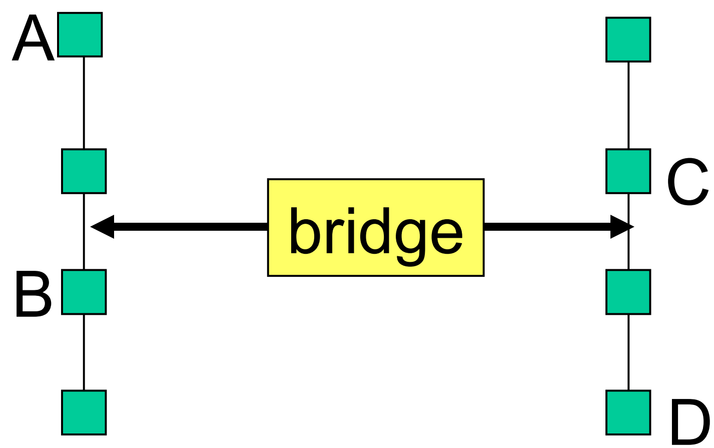
- "Transparently" interconnect LANs with bridge
  - Receive frames from each LAN and forward to the other
  - Each LAN is its own collision domain; bridge isn't a repeater



Note: We're operating below the level of IP here. (This isn't *routing*.)

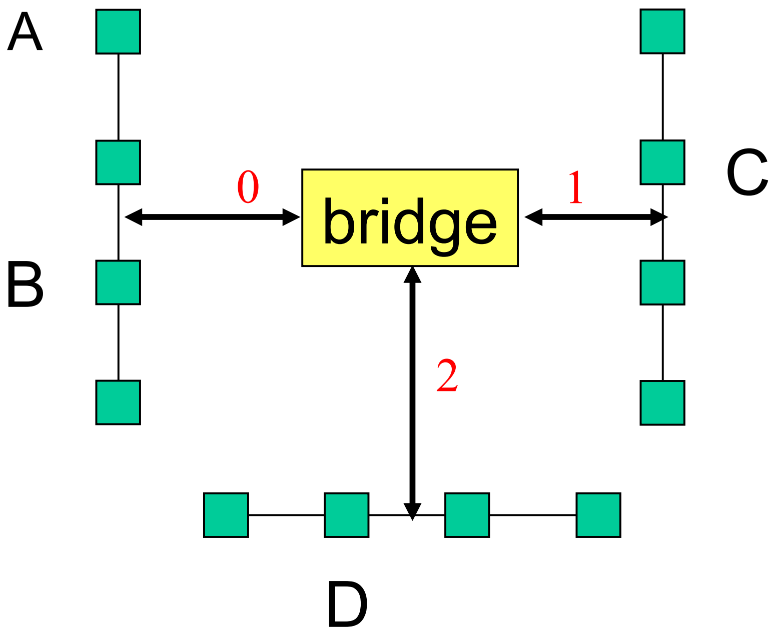
Learning Bridges

- To optimize overall performance:
  - Shouldn't forward A→B or C→D, should forward A→C and D→B



- How does the bridge know?
  - Learn who is where by observing <u>source</u> addresses and prune
  - Send
  - Forward using destination address; age for robustness

An Example

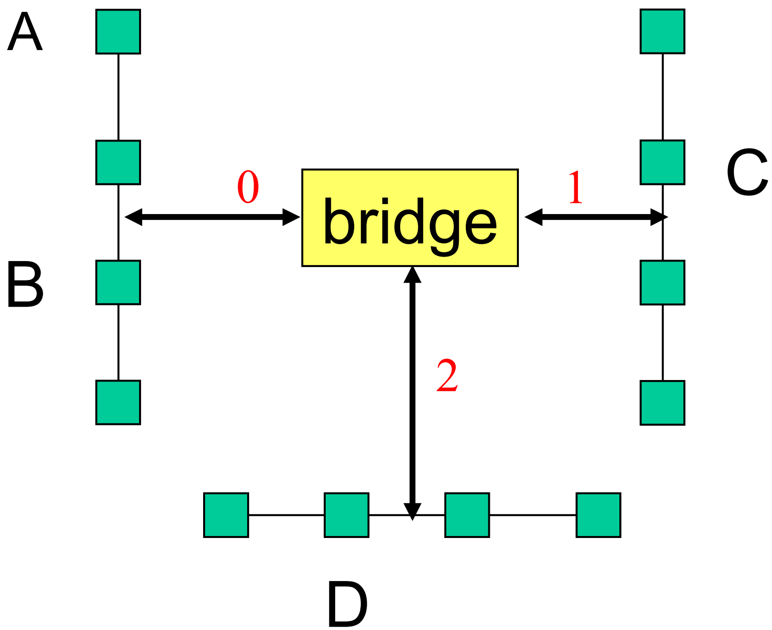


A → B
B → A
C → D
A → D
D → C

Forwarding Table

Dest. Address	Interface

After the Four Packets Have Been Sent



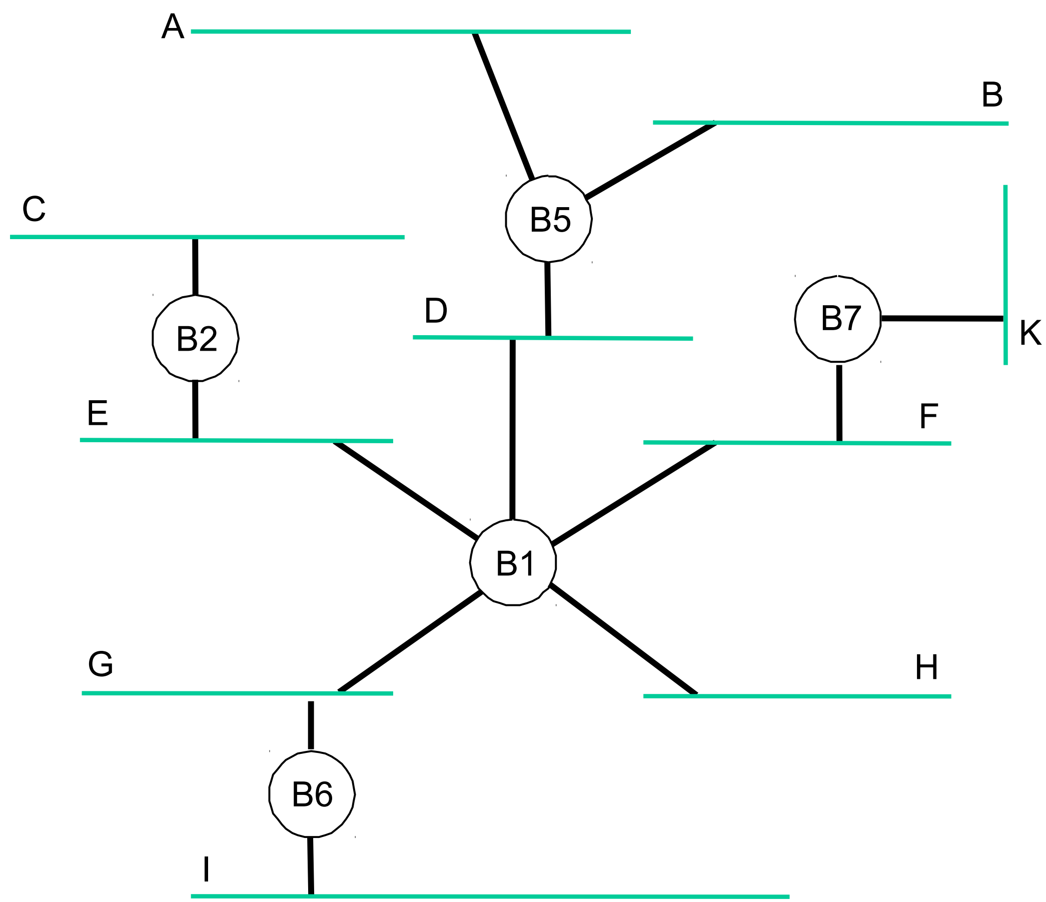
A → B : 1,2
B → A:
C → D: 0,2
A → D: 1,2
D → C: 1

Forwarding Table

Dest. Address	Interface
A	0
B	0
C	1
D	2

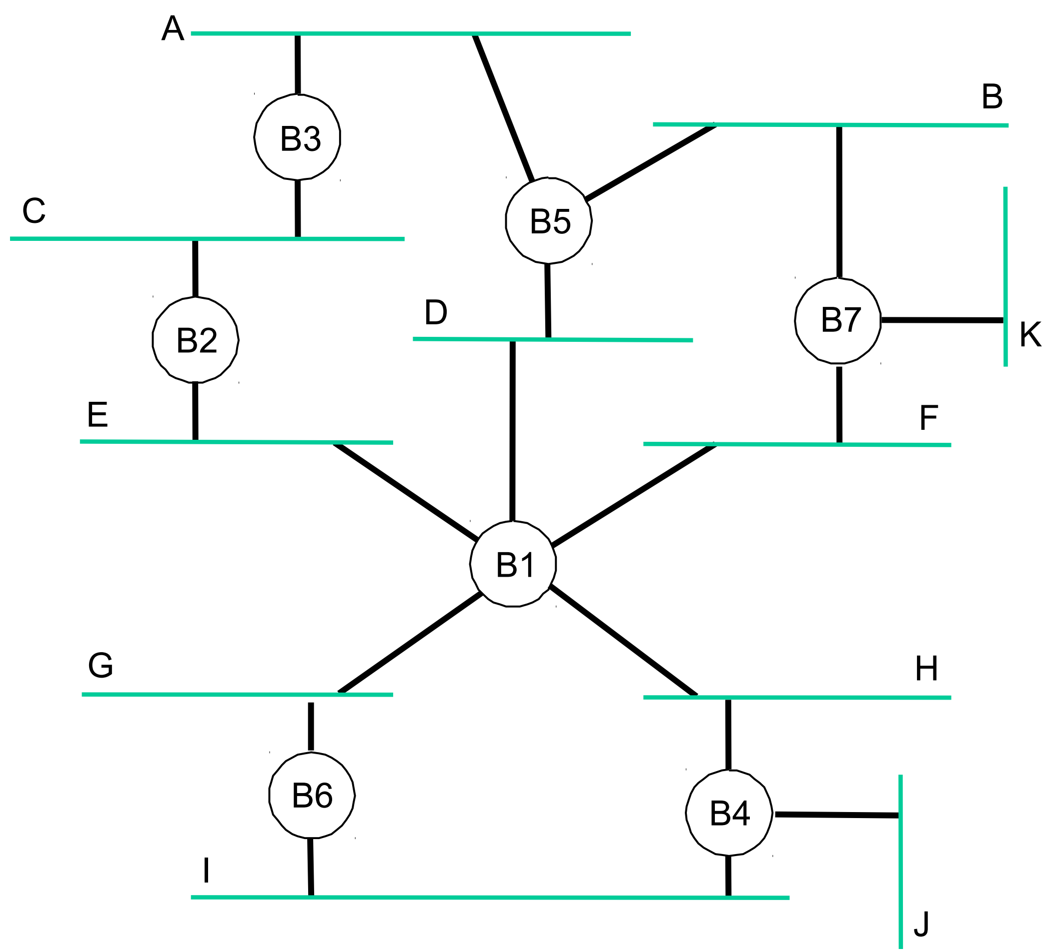
Why stop at one bridge?

Why not just keep doing this forever?



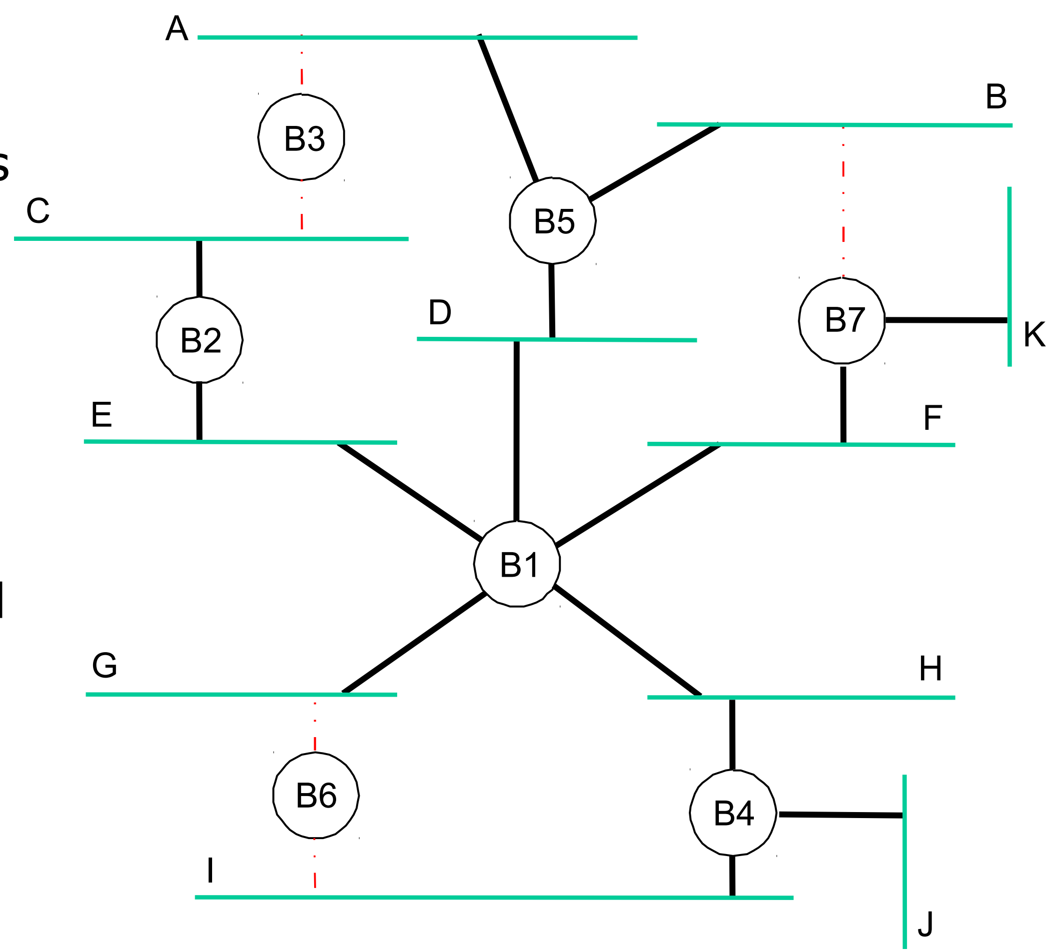
What's wrong with this picture?

- Redundancy added for fault tolerance
- Redundancy added by mistake
- Either way, what goes wrong?



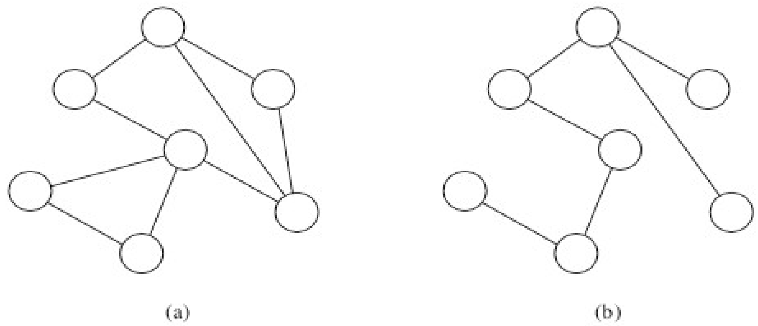
Spanning Tree Example

- Spanning tree selects bridge ports so there are no cycles
  - Prune some ports
  - Only one tree
- Q: How do we find a spanning tree?
  - Automatically, with a distributed algorithm



Spanning Tree

- Compute ST with *a* bridge as *root* such that
  - Root forwards onto all of its outgoing ports
  - Other bridges forward TO the root if a frame is received on a port “further from the root”, else they forward away from the root
    - Packet traversal: forwards (UP*) then (DOWN*)



Spanning tree vs. learning

- Once the spanning tree is in place…
  - the bridge uses the regular learning algorithm to figure out which port(s) to forward / flood packet on

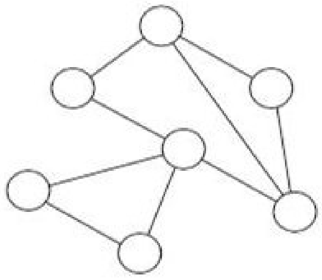
- Job of spanning tree algorithm is to disable some ports to eliminate cycles

Spanning Tree Algorithm

- Radia Perlman; IEEE 802.1 spec; http://www1.cs.columbia.edu/~ji/F02/ir02/p44-perlman.pdf

- Dynamic, distributed algorithm to compute spanning tree
  - Dynamic: robust against failures
  - Distributed:
    - needs no organization/management, but…
    - the usual complexities of "who knows what, when?" in distributed computations
      - All nodes must come to the same conclusions
      - Easy part: use some deterministic calculation (e.g., sorting)
      - Hard part: make sure everyone is working on the same data (or at least data that ends up giving the same result)

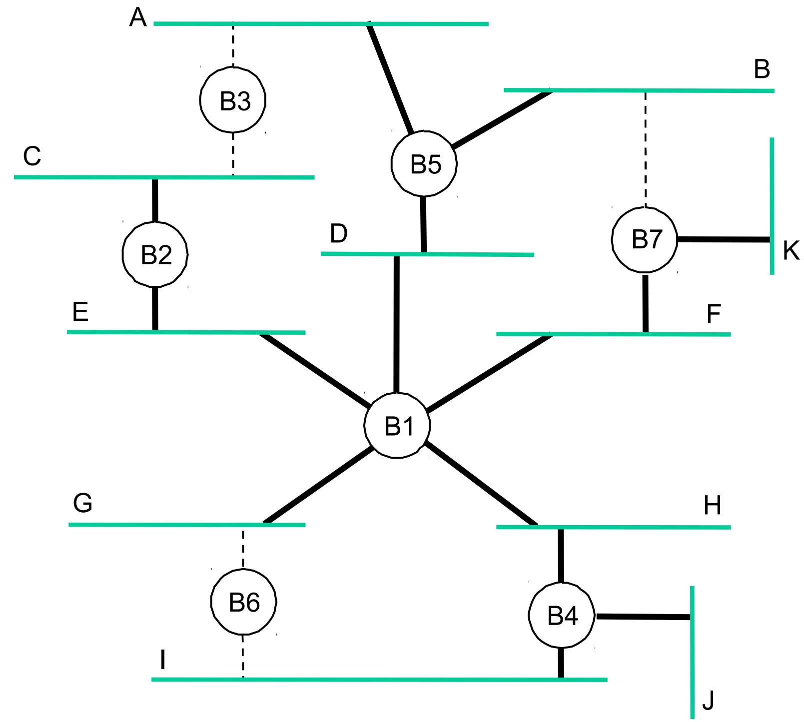
- Outline: Goal is to turn some bridge ports off
  1. Elect a root node for the tree
  2. Grow tree as shortest distances from the root
  3. Turn off ports that aren't on "best" paths



- Note: "best path" is constrained by (UP*)(DOWN*), it's not "best" for each source-dest pair

Algorithm Overview

- Elect a root
  - Each bridge has a unique id
    - e.g., B1, B2, B3
  - Inform each node of the id's of all other nodes *(sort of)*
  - Each picks the smallest node id as its idea of the root
  - Et voila
- Agree on a tree
  - Select as designated bridge on each LAN the one:
    - that is closest to the root as that LAN's designated bridge
    - Has smallest id in case of ties
- When done
  - Each bridge forwards frames over each LAN on which it is the designated bridge



How?

- Initially:
  - Each bridge knows what ports it has
  - Each bridge currently believes that it is the root
    - *It therefore believes it is responsible for forwarding packets to all of its connected LANs*
  - That's everything

- Bridges send configuration messages, containing:
  - id of bridge sending the message
  - id for what that bridge currently believes to be the root bridge
  - distance (hops) from sending bridge to root bridge

- Bridges receive configuration messages from immediate neighbors
  - Each receiver keeps track of the current best configuration message for each port
  - New information may change its idea of:
    - who the root is
    - its distance from the root
    - who is responsible for the LAN directly connected to one of its ports

"Best Configuration"

- Maintained per-port
- Goal:
  - have all bridges connected to a single LAN agree on which of them is responsible for it
  - Key: sorting (plus making sure they have the same information)

- Rules for "best":
  - Identifies a root with a smaller ID (than current best)
  - Identifies the same root, but has a smaller hop count to it
  - Root id and hop count same, but sending bridge has a smaller id

Algorithm More…

- When learn not designated bridge on LAN, stop forwarding configuration messages on it
  - in steady state, only designated bridges forward configuration messages

- Root bridge continues to send configuration messages periodically

- If a bridge does not receive any configuration messages during a period of time:
  - assumes topology has changed
  - starts generating configuration messages claiming to be root

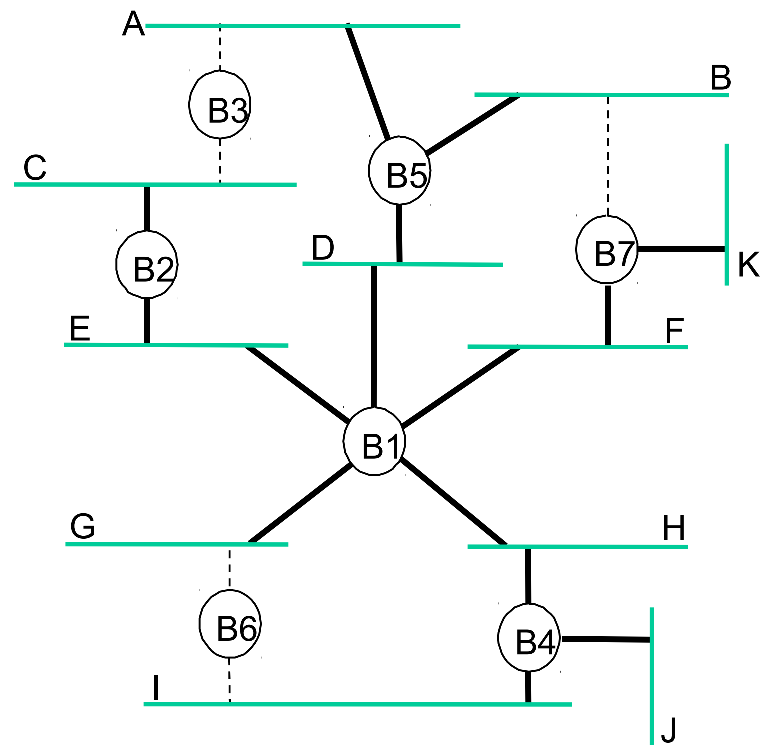
Algorithm Example

- Message format:
  – (root, dist-to-root, sending bridge)

- Sample message sequence to and from B3:
  1. B3 sends (B3, 0, B3) "to B2 and B5"
  2. B3 receives (B2, 0, B2) and (B5, 0, B5) and accepts B2 as root
  3. B3 sends (B2, 1, B3) to B5
  4. B3 receives (B1, 1, B2) and (B1, 1, B5) and accepts B1 as root

  1. B3 could send (B1, 2, B3) but doesn't as its nowhere "best"
     - B2 and B5 are better choices.
       - so B3 is NOT a designated bridge

  – B3 receives (B1, 1, B2) and (B1, 1, B5) again … stable
     - B3 turns off data forwarding to LANs A and C

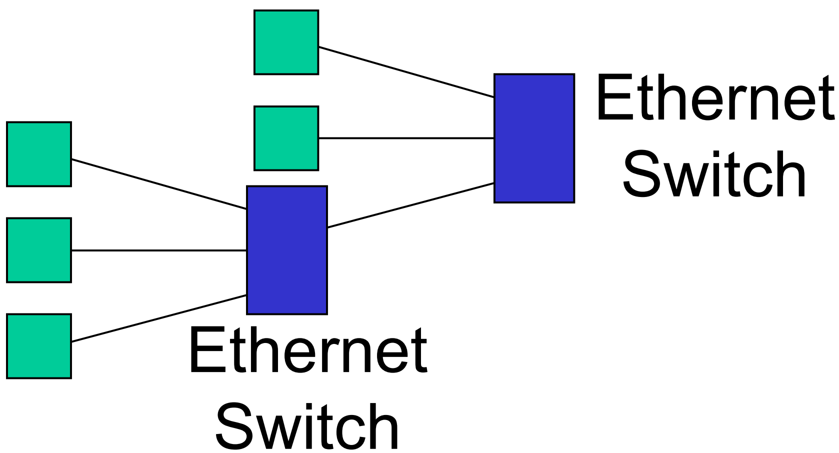


Some other tricky details

- Configuration information is aged
  - If the root fails a new one will be elected
- Reconfiguration is damped
  - Adopt new spanning trees slowly to avoid temporary loops
- What can happen during reconfiguration?
  - Loops?
  - Frames lost?
  - Frames duplicated?

LAN Switches

- LAN switches are multi-port bridges
  - Modern, high performance form of bridged LANs
  - Looks like a hub, but frames are switched, not shared
  - Every host on a separate port, or can combine switches



Limitations of Bridges/Switches

- LAN switches form an effective small-scale network
  - Plug and play for real!

- Why can't we build a large network using bridges?
  - Little control over forwarding paths
  - Size of bridge forwarding tables grows with number of hosts
  - Broadcast traffic flows freely over whole extended LAN
  - Spanning tree algorithm limits reconfiguration speed
  - Poor solution for connecting LANs of different kinds

Key Concepts

- We can overcome LAN limits by interconnection
  - Bridges and LAN switches
  - But there are limits to this strategy …
- Next Topic: Routing and the Network layer
  - How to grow large and really large networks