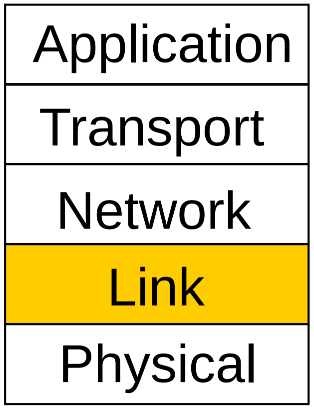
CSEP 561 – LAN Switches

David Wetherall
djw@cs.washington.edu

How to combine links into a simple network

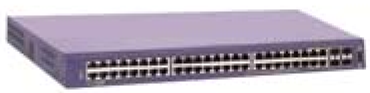
- Topics:
  - Switch internals
  - “Plug and play” LANs (switched Ethernet)
- Later:
  - Building more sophisticated networks with routers

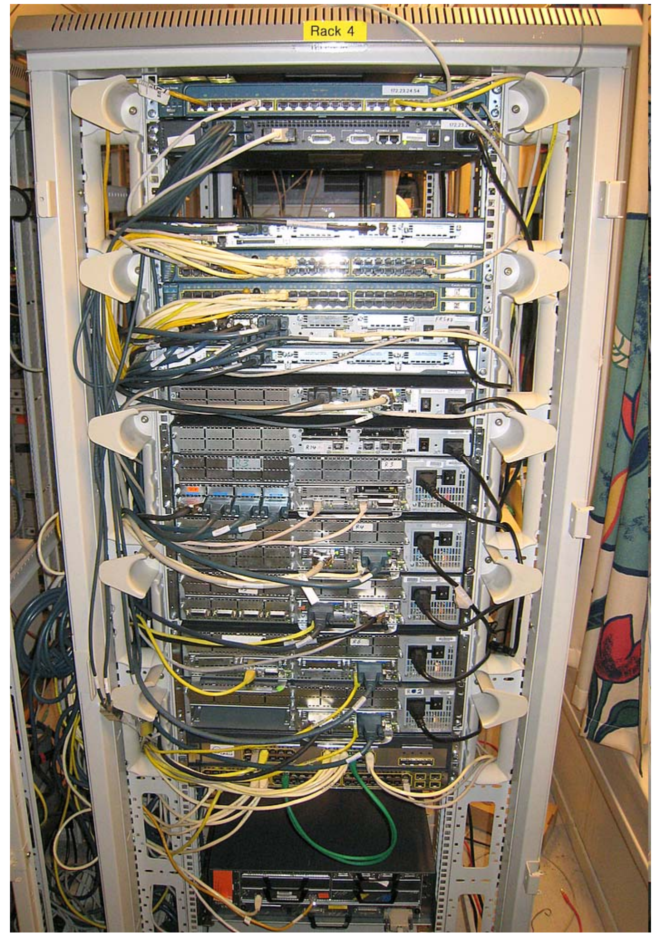


Terminology

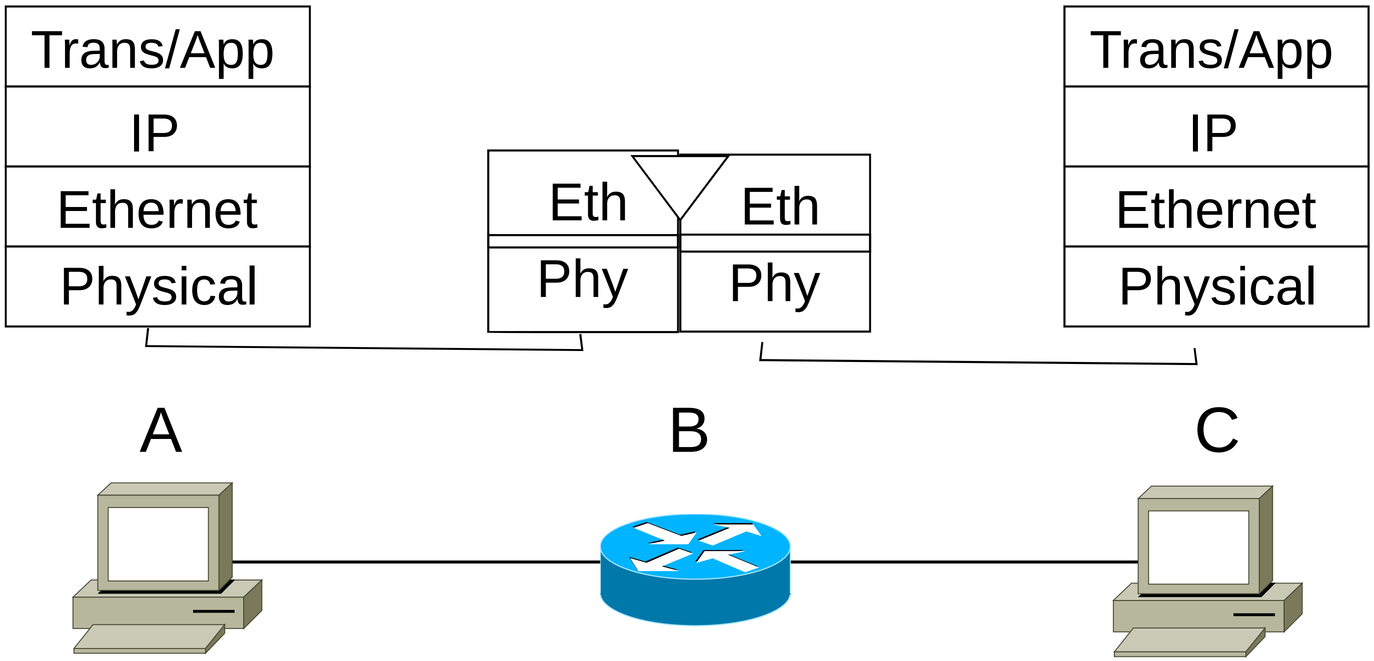
- Bridge
  - Old fashioned name for a LAN switch, e.g., Ethernet switch
  - Works at the link (Ethernet) layer
- Router
  - Switch that works at the network (IP) layer
- Switch
  - Generic term for a low-level interconnection device
- Gateway
  - Generic term for a high-level interconnection device

They can all look the same …



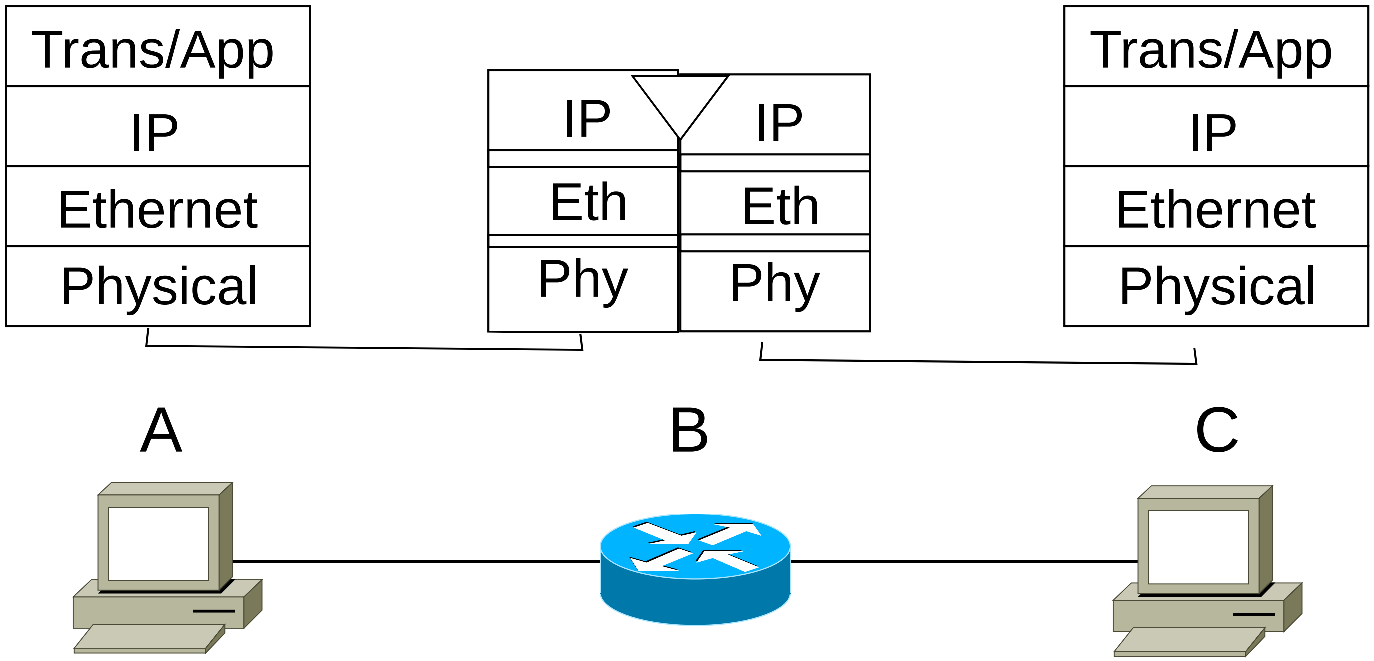


Sanity check, bridge



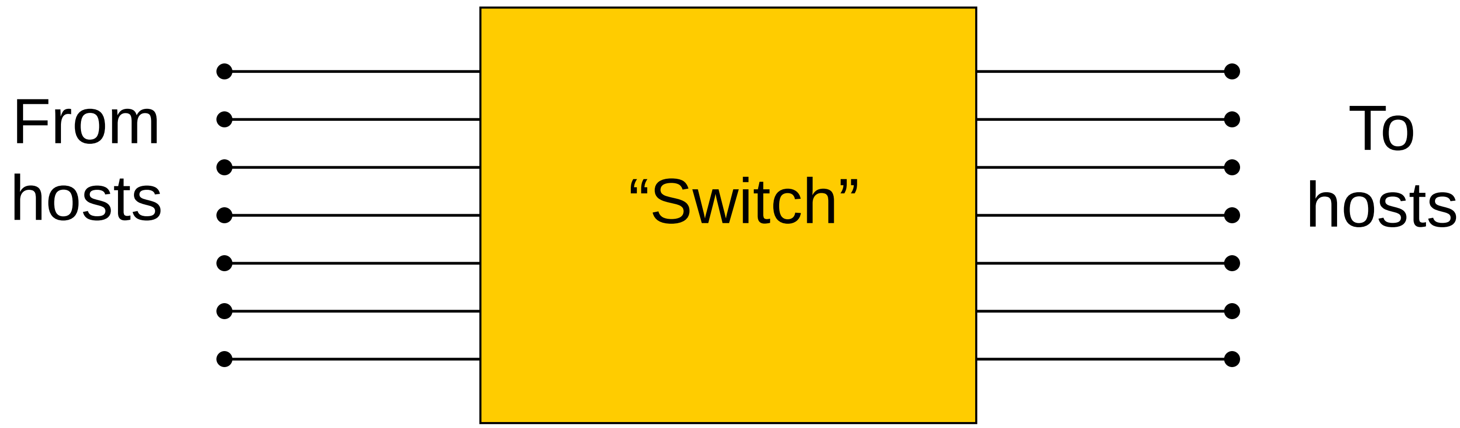
- What source and destination Ethernet / IP addresses are seen on each wire?

Sanity check, router



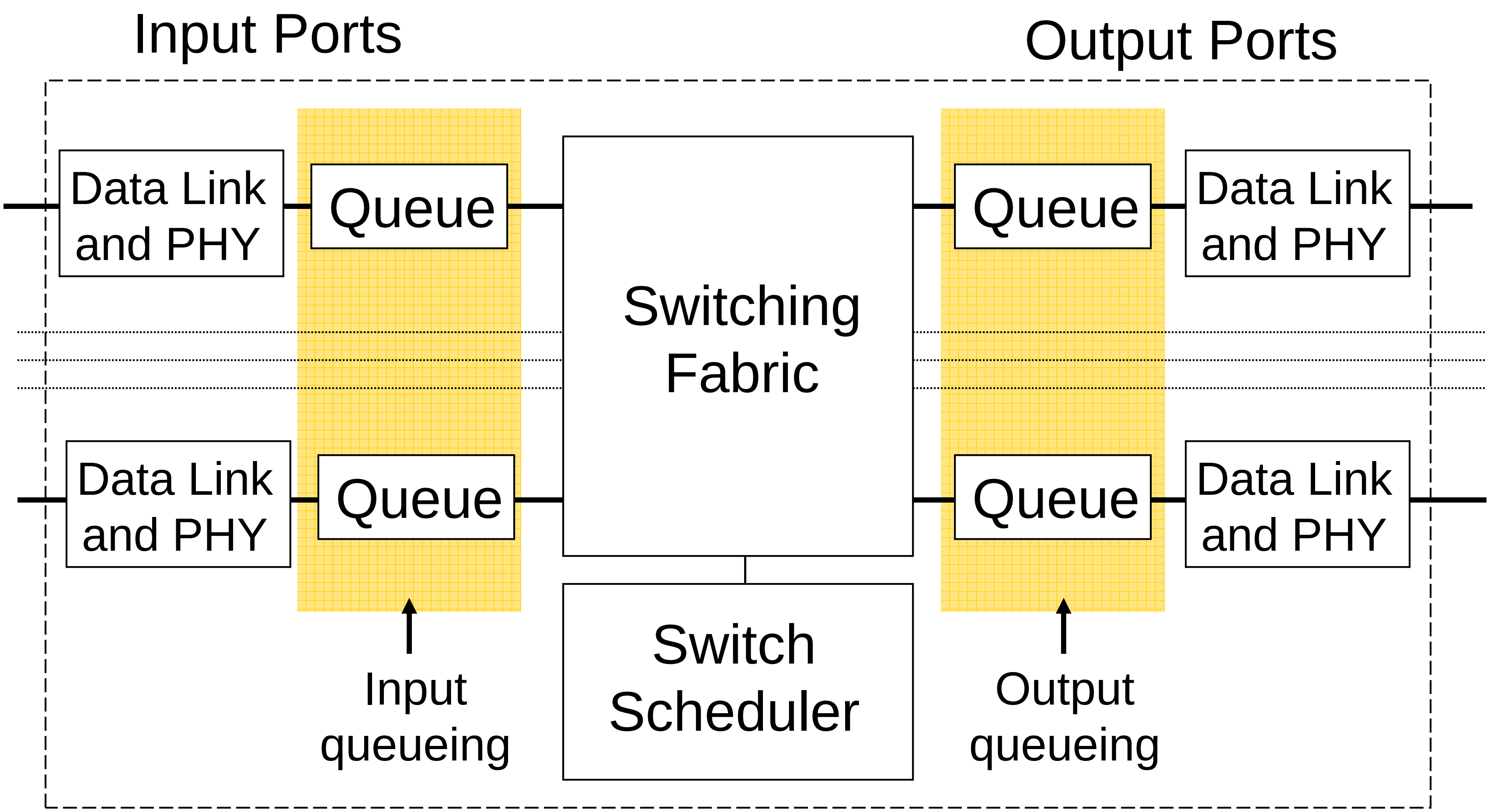
- What source and destination Ethernet / IP addresses are seen on each wire?

What’s in a Switch?



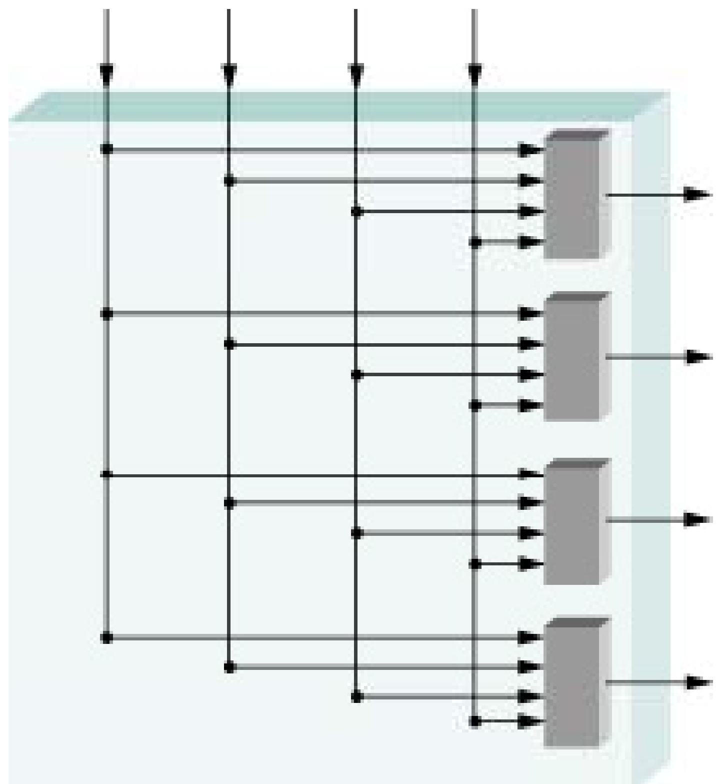
- By convention, draw input ports on left, output on right.
- In reality a single physical port handles both directions.
- Switch sends input “to the right output”; hub sends to all

Model of a Switch



Crossbar switch

- On/off setting of intersection points control connections from inputs to outputs

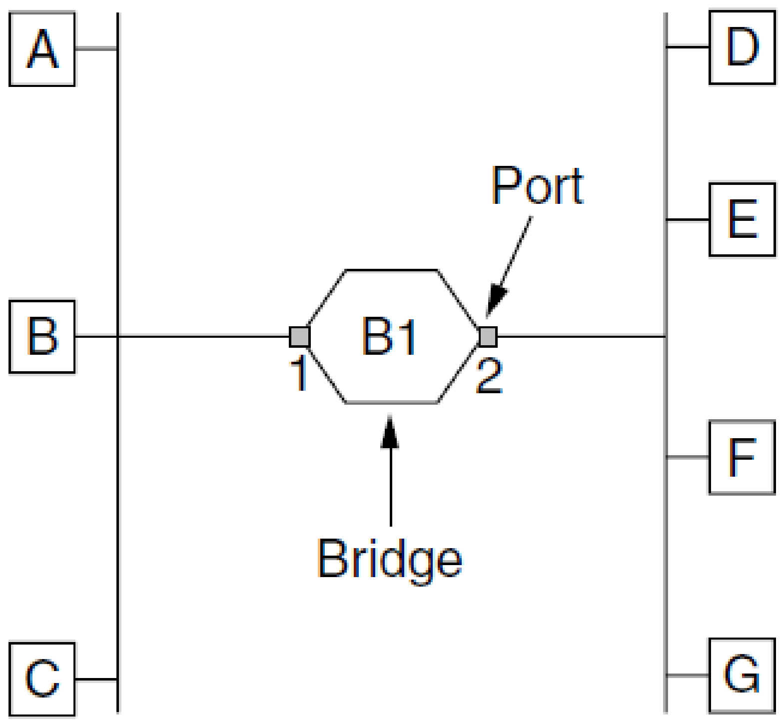


LAN Switches / Bridges

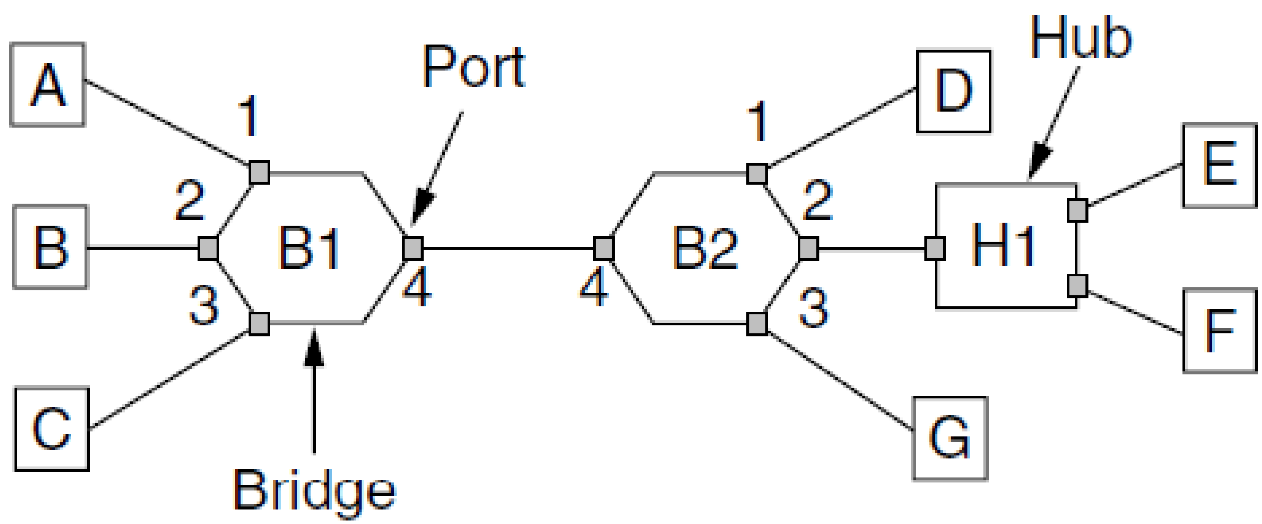
- When one LAN isn't enough, we can combine them
- This is "plug and play" using two algorithms:
  - 1. Backward learning
  - 2. Spanning tree computation

- Link layer operation implies that frames are forwarded using destination MAC address

Classic Ethernet – shared LANs

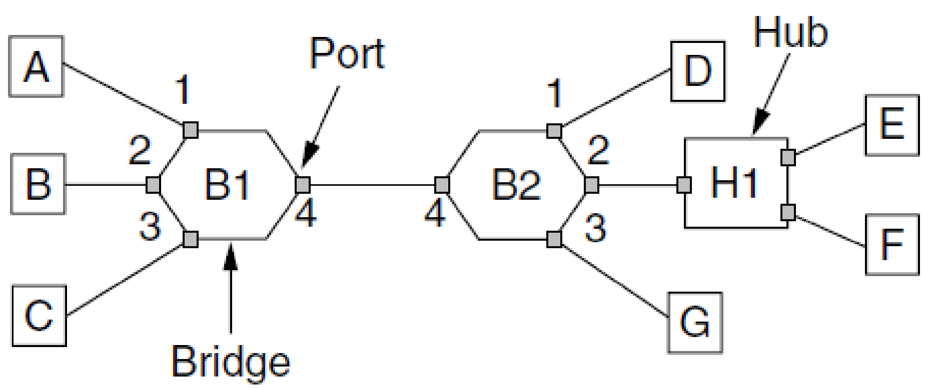


Modern Ethernet -- switched



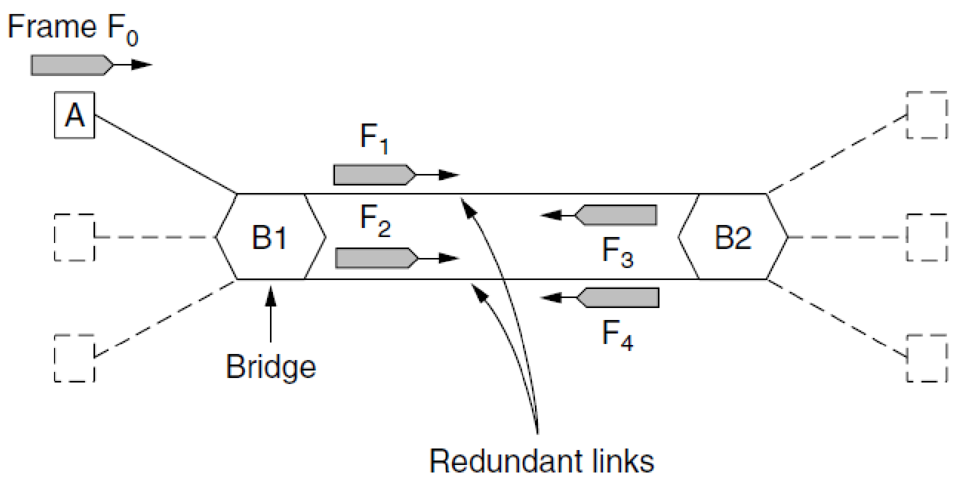
Backward Learning Algorithm

- To optimize overall performance:
  - Don't forward A→B or D→G between bridges, do for A→D and D→C



- But how does the bridge know?
  - Learn who is where by observing source addresses and prune
  - Forward using destination address; age for robustness
  - Broadcast if you don't know

Is redundancy good or bad?



- Seems useful (backup, more capacity)
- But causes a potential problem – forwarding loops
- Solution is the spanning tree algorithm

Radia Perlman says …

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 which must be sure to span
So packets 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.

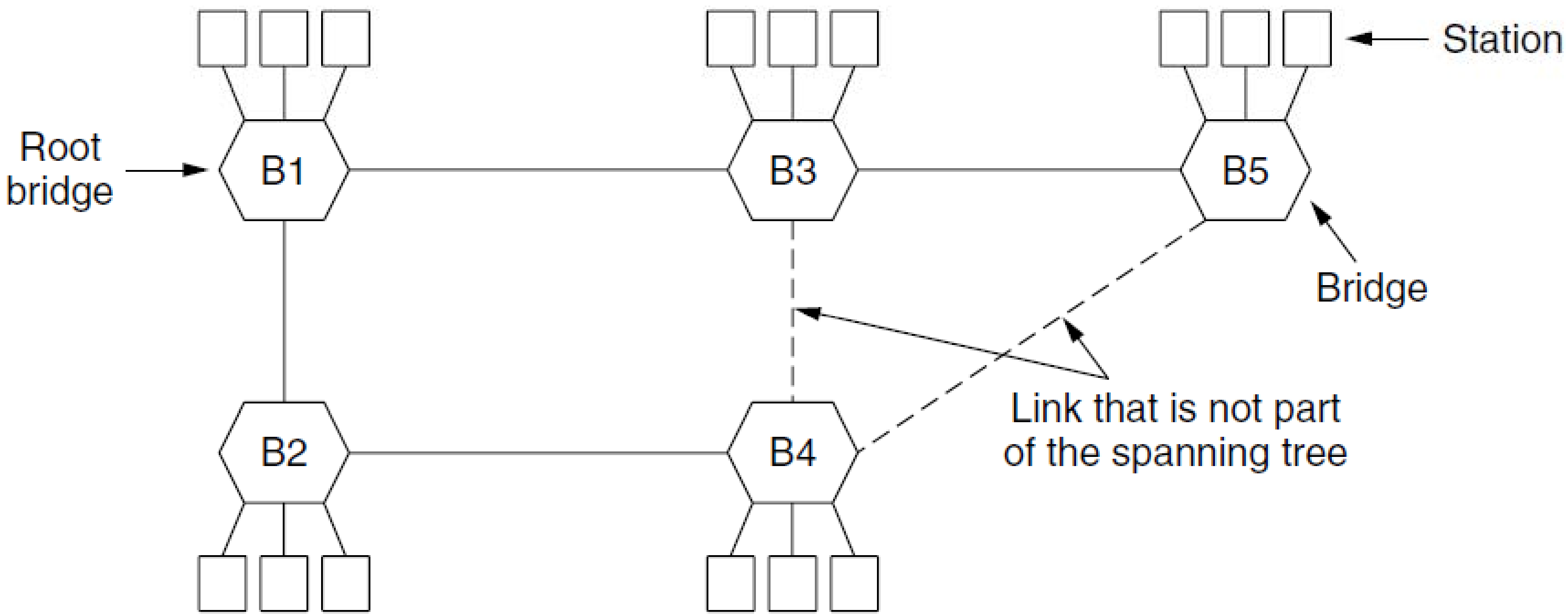
From:
“An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN”,
R. Perlman, SIGCOMM 1985.

Spanning Tree Algorithm

- Distributed algorithm to compute spanning tree
  - Robust against failures, needs no organization

- Outline:
  - Goal is to turn some bridge ports off to break loops
  1. Elect a root node of the tree (lowest address)
  2. Grow tree as shortest distances from the root (using lowest address to break distance ties)
  - All done by bridges sending periodic configuration messages over ports for which they are the “best” path
  - Then turn off ports that aren’t on “best” paths

Spanning tree example



Algorithm details

- Each bridge sends periodic messages to others containing:
  - Its address, address of the root bridge, and distance (in hops) to root
- Each bridge receives messages, updates "best" config.
  - Smaller root address is better, then shorter distance
  - To break ties, bridge with smaller address is better
- Initially, each bridge thinks it is the root
  - Sends configuration messages on all ports
- Later, bridges send only "best" configs
  - Add 1 to distance, send configs where still "best" (designated bridge)
  - Turn off forwarding on ports except those that send/receive "best"

Some Design Aspects

- All bridges to run the same algorithm
- Bridges start with no information and operate in parallel
- Bridges send periodic messages about their own state
- State that isn't refreshed is soon deleted (soft-state)
- If we all have the same inputs and are running the same algorithm, we converge to a globally consistent state.

This is a <u>common</u> design pattern for network protocols that adapts to failures. Learn it. Live it. Love it.

Perlman paper -- faults

- Algorithm tolerates a large variety of fail-stop faults
  - Switches, bridges failing (and reappearing), including the root
  - Links failing (and healing) including partitions
  - Potentially one-way connectivity
  - Hosts moving or corrupt switch tables (not part of sp. tree)

- Little is ruled out
  - E.g., unique MAC addresses assumed
  - But “one way” a problem in practice (e.g., duplex mismatch)
  - And what happens when the network is too large?
  - Errors can be very hard to debug (Boston hospital example)

Perlman paper -- improvements

- Algorithm was very fitting for the needs of the day.
- Various areas of improvement identified over time:
  - Traffic paths chosen are limited (e.g., one tree for entire network, not parallel paths, preferred paths, etc.)
  - Reconfiguration due to faults can be slow
  - Management improved with VLANs
  - Security not much of an issue in practice

- Perspective:
  - Excellent for what they are (small-scale enterprise network)
  - But use routers for larger, more diverse networks

Perlman paper -- deployment

- Top marks!
  - Can add new bridges/switches to old (hub) networks gradually
  - No change in configuration of the old equipment
  - No configuration needed for the new equipment
  - No constraints on what can be plugged in where
- Relies on old networks passing new messages that they do not understand (bridge management PDUs)
  - Be conservative in what you send, liberal in what you accept