

# Lecture 9:

## Bridging & Switching

---

CSE 123: Computer Networks  
Stefan Savage





# Lecture 9 Overview

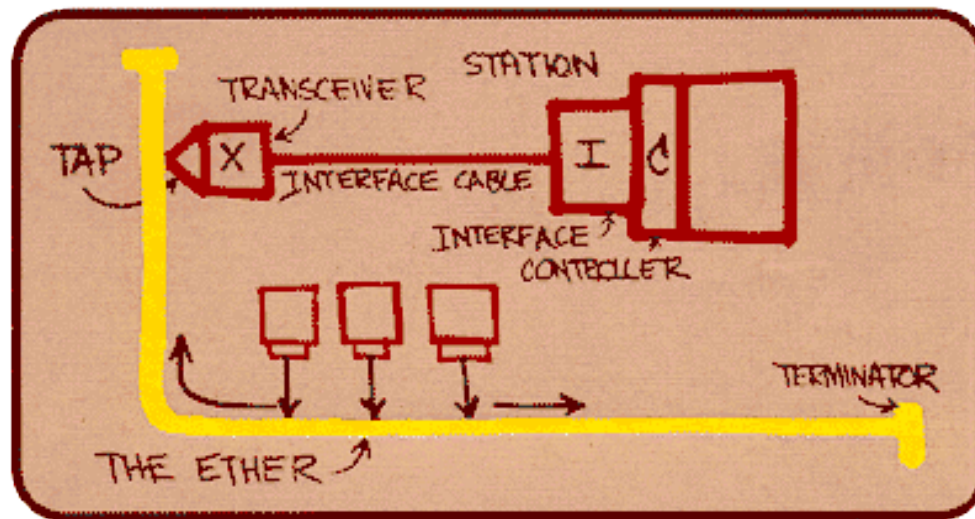
---

- Finishing up media access
  - ◆ Ethernet
  - ◆ Contention-free methods (rings)
- Moving beyond one wire
  - ◆ Link technologies have limits on physical distance
  - ◆ Also frequently on number of hosts connected
- Methods to interconnect LANs
  - ◆ Repeaters and bridges
  - ◆ Switching

# Ethernet



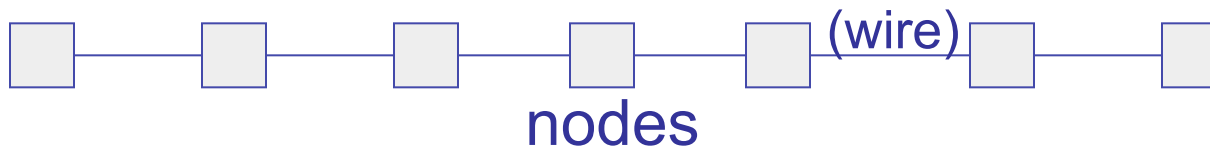
- First **local area network** (LAN)
  - ♦ Developed in early '70s by Metcalfe and Boggs at PARC
  - ♦ Originally 1Mbps, now supports 10Mbps, 100Mbps, 1Gbps and 10Gbps flavors (40/100G in development)
- Currently *the* dominant LAN technology
  - ♦ Becoming the dominant WAN technology





# Classic Ethernet

- IEEE 802.3 standard wired LAN  
(modified 1-persistent CSMA/CD)
- Classic Ethernet: 10 Mbps over coaxial cable
  - ◆ All nodes share same wire
  - ◆ Max length 2.5km, max between stations 500m



- Framing
  - ◆ Preamble, 32-bit CRC, variable length data
  - ◆ Unique 48-bit address per host (bcast & multicast addrs too)

|              |            |          |         |               |           |         |
|--------------|------------|----------|---------|---------------|-----------|---------|
| Preamble (8) | Source (6) | Dest (6) | Len (2) | Payload (var) | Pad (var) | CRC (4) |
|--------------|------------|----------|---------|---------------|-----------|---------|



# Ethernet improvements

---

- Problems with random delay with fixed mean
  - ◆ Few senders = unnecessary delay
  - ◆ Many senders = unnecessary collisions
- Binary exponential back-off balances delay w/load
  - ◆ First collision: wait 0 or 1 min frame times at random, retry
  - ◆ Second time: wait 0, 1, 2, or 3 times
  - ◆  $N$ th time ( $n \leq 10$ ): wait 0, 1, ...,  $2^n - 1$  times
  - ◆ Max wait 1023 frames; give up after 16 attempts



# Capture Effect

---

- Randomized access scheme is not fair
- Suppose stations  $A$  and  $B$  always have data to send
  - ◆ They *will* collide at some time
  - ◆ Both pick random number of “slots” (0, 1) to wait
  - ◆ Suppose  $A$  wins and sends
  - ◆ Next time they collide,  $B$ ’s chance of winning is halved
    - »  $B$  will select from 0,1,2,3 due to exponential back-off
- $A$  keeps winning: said to have **captured** the channel



# Ethernet Performance

---

- Much better than Aloha or CSMA in practice
- Source of protocol inefficiency: collisions
  - ◆ More efficient to send larger frames
    - » Acquire the medium and send lots of data
  - ◆ Less efficient if
    - » More hosts – more collisions needed to identify single sender
    - » Smaller packet sizes – more frequent arbitration
    - » Longer links – collisions take longer to observe, more wasted bandwidth



# Contention-free Protocols

---

- Problem with fixed partitioning:
  - ◆ Inefficient at low load (idle channels)
- Problem with contention-based protocols:
  - ◆ Inefficient at high load (collisions)
- Ideal(?): Contention-free protocols
  - ◆ Try to do both by explicitly taking turns
  - ◆ Can potentially also offer guaranteed bandwidth, latency, etc.

# Contention-free Approaches

---



## Polling

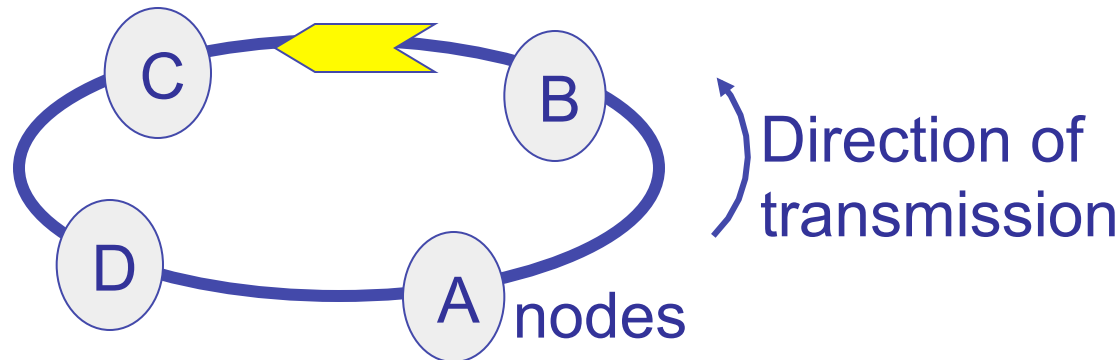
- Master node “invites” slave nodes to transmit in turn
  - ♦ Request to Send (RTS), Clear to Send (CTS) messages
- Problems:
  - ♦ Polling overhead
  - ♦ Latency
  - ♦ Single point of failure (master)

## Token Passing

- Control **token** passed from one node to next sequentially.
- Point-to-point links can be fast
- Problems:
  - ♦ Token overhead
  - ♦ Latency
  - ♦ Single point of failure (token)



# Token Ring (802.5)



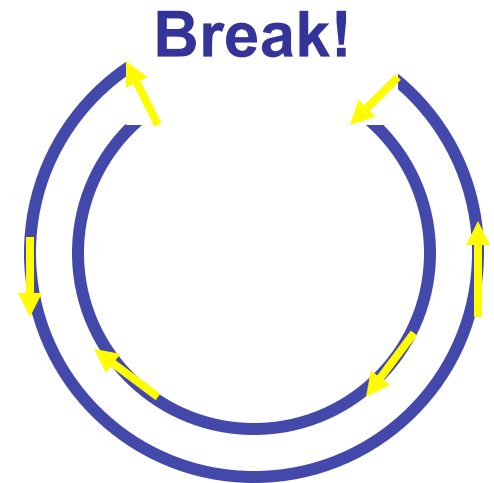
- Token rotates “permission to send” around nodes
- Sender injects packet into ring and removes later
  - ◆ Maximum **token holding time** (THT) bounds access time
  - ◆ Early or delayed token release
  - ◆ Round robin service, acknowledgments and priorities
- Monitor nodes ensure health of ring (alerts on failures)

# FDDI

## (Fiber Distributed Data Interface)



- Roughly a large, fast token ring
  - ◆ 100 Mbps and 200km vs 4/16 Mbps and local
  - ◆ Dual counter-rotating rings for redundancy
  - ◆ Complex token holding policies for voice etc. traffic
- Token ring advantages
  - ◆ No contention, bounded access delay
  - ◆ Support fair, reserved, priority access
- Disadvantages
  - ◆ Complexity, reliability, scalability





# Why Did Ethernet Win?

---

- Failure modes
  - ◆ Token rings – network unusable
  - ◆ Ethernet – node detached
- Good performance in common case
- Completely distributed, easy to maintain/administer
- Easy incremental deployment
- Volume → lower cost → higher volume ....



# Summary of Media Access

---

- How to divide shared channel among different users
  - ◆ Fixed partitioning (FDMA, TDMA, CDMA)
    - » Guaranteed bandwidth for each user, but wasteful when not used and can't allocate different bandwidth to different users
  - ◆ Contention-based protocols (CSMA, CSMA/CD)
    - » Try and backoff if fail; dynamic allocation of bandwidth on demand, works well at load but collisions a problem at high load
  - ◆ Contention-free protocols (Token Ring, FDDI, RTS/CTS)
    - » Explicit turn-taking; strong guarantees on access time and can make bandwidth guarantees, but complex and fragile to failure
- But... aren't there limits to what we can do with one shared channel?



# Problems with Busses

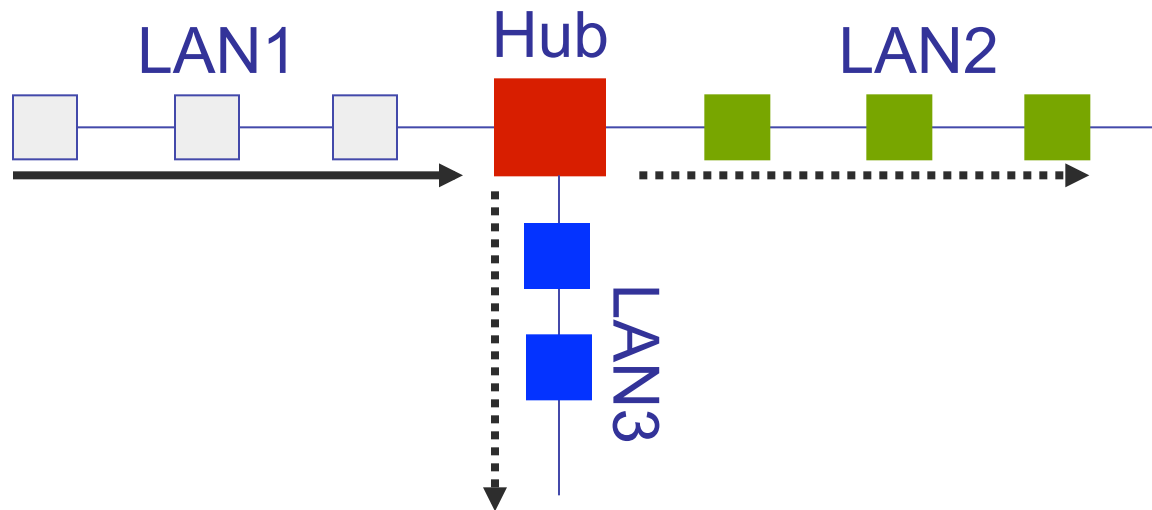
---

- One shared link (a bus) limits scale in terms of:
  - ◆ Distance (e.g., 2500 m for Ethernet)
  - ◆ Number of nodes (1024 nodes)
  - ◆ Performance (Capacity shared across all nodes)
- A better alternative is to have multiple busses
  - ◆ Each bus is of a limited size, scale, number of hosts, etc.
- Need the ability to connect multiple busses together
  - ◆ In other words move frames from one wire to another



# Hubs/Repeaters

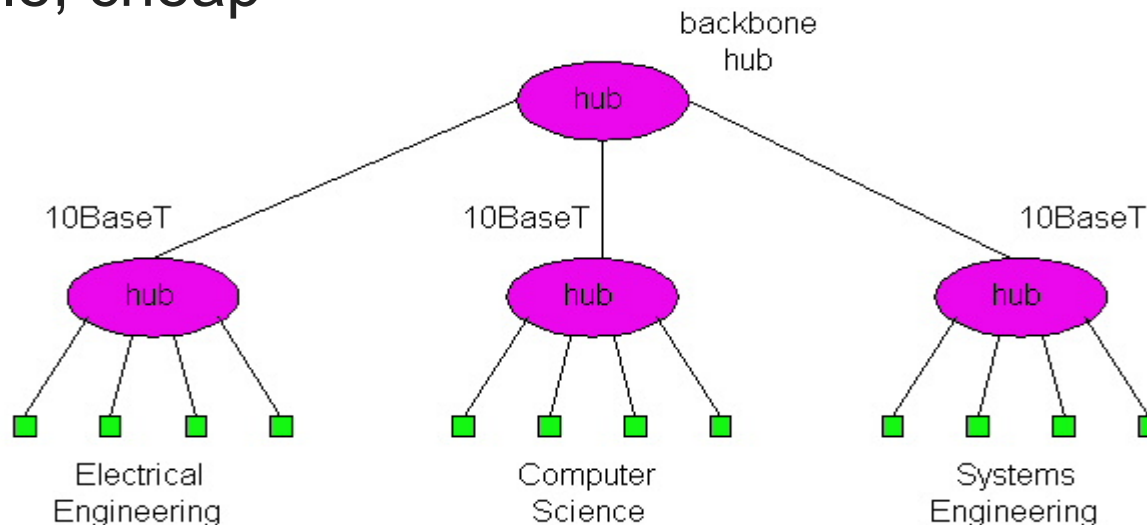
- Physical layer device
  - ◆ One “port” for each LAN
  - ◆ Repeat received *bits* on one port out *all* other ports





# Hub Advantages

- Hubs can be arranged into hierarchies
  - ◆ Ethernet: up to four hubs between any pair of nodes
- Most of LAN continues to operate if “leaf” hub dies
- Simple, cheap





# Still One Big Bus

---

- Single **collision domain**
  - ◆ No improvement in max throughput
  - ◆ Average throughput  $<$  as # of nodes increases
  - ◆ Why?
- Still limited in distance and number of hosts
  - ◆ Collision detection requirements
  - ◆ Synchronization requirements
- Requires performance homogeneity
  - ◆ Can't connect 10 Mbps and 100 Mbps networks

# Bridges

---

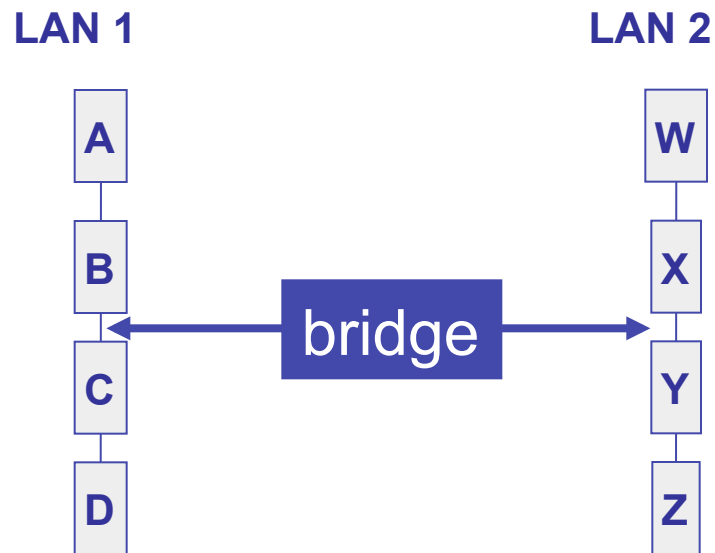


- Store and forward device
  - ◆ Data-link layer device
  - ◆ Buffers entire packet and *then* rebroadcasts it on other ports
- Creates *separate* collision domains
  - ◆ Uses CSMA/CD for access to each LAN (acts like a host)
  - ◆ Can accommodate different speed interfaces (issues?)
  - ◆ Separate CDs improves throughput (why?)
- Can significantly improve performance
  - ◆ Not all frames go everywhere. (Why did they with a hub?)



# Selective Forwarding

- Only rebroadcast a frame to the LAN where its destination resides
  - ◆ If A sends packet to X, then bridge must forward frame
  - ◆ If A sends packet to B, then bridge shouldn't





# Forwarding Tables

---

- Need to know “destination” of frame
  - ◆ Destination address in frame header (48bit in Ethernet)
- Need know which destinations are on which LANs
  - ◆ One approach: statically configured by hand
    - » Table, mapping address to output port (i.e. LAN)
  - ◆ But we’d prefer something automatic and dynamic...
- Simple algorithm:  
Receive frame *f* on port *q*  
Lookup *f.dest* for output port */\* know where to send it? \*/*  
**If** *f.dest* found  
**then if** output port is *q* **then** drop */\* already delivered \*/*  
          **else** forward *f* on output port;  
**else** flood *f*;  
*/\* forward on all ports but the one where frame arrived\*/*



# Learning Bridges

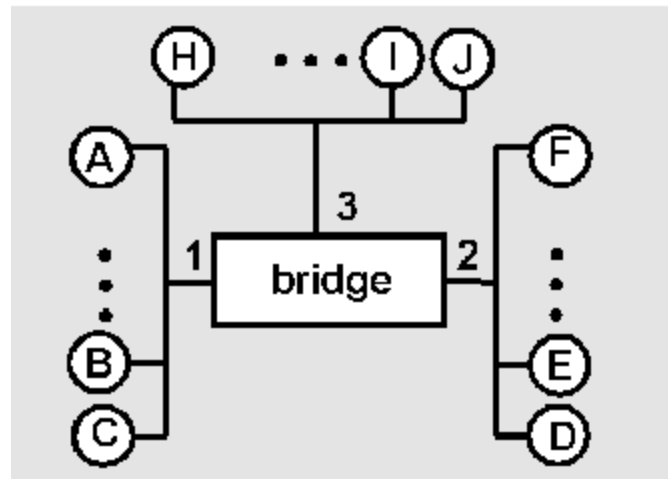
- Eliminate manual configuration by learning which addresses are on which LANs
- Basic approach
  - ◆ If a frame arrives on a port, then associate its **source** address with that port
  - ◆ As each host transmits, the table becomes accurate
- What if a node moves? Table aging
  - ◆ Associate a timestamp with each table entry
  - ◆ Refresh timestamp for each new packet with same source
  - ◆ If entry gets too stale, remove it

| Host | Port |
|------|------|
| A    | 1    |
| B    | 1    |
| C    | 1    |
| D    | 1    |
| W    | 2    |
| X    | 2    |
| Y    | 3    |
| Z    | 2    |



# Learning Example

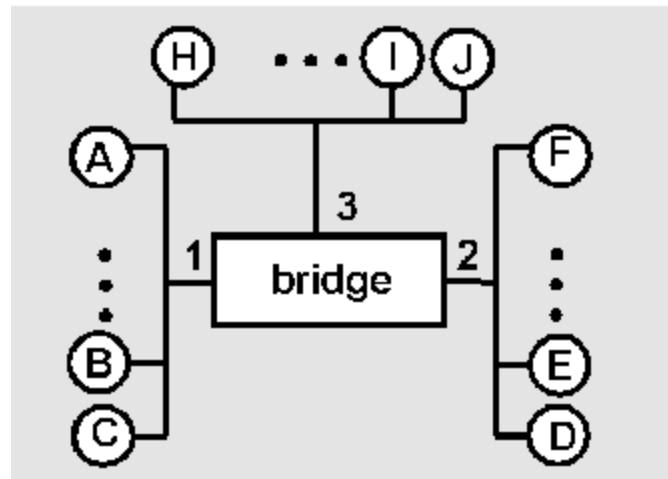
Suppose *C* sends frame to *D* and *D* replies back with frame to *C*



- C sends frame, bridge has no info about D, so floods to both LANs
  - ♦ bridge notes that C is on port 1
  - ♦ frame ignored on upper LAN
  - ♦ frame received by D



# Learning Example



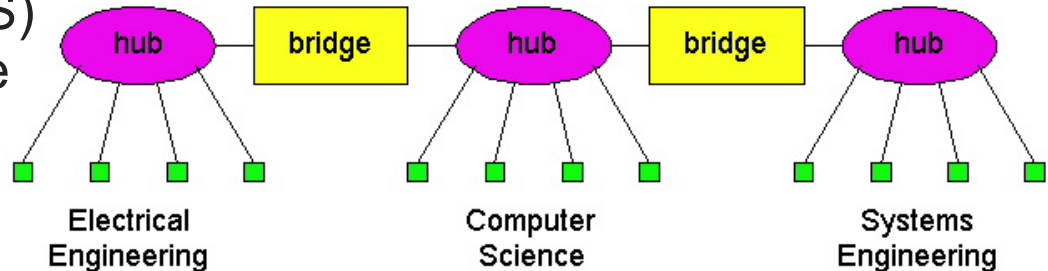
- *D* generates reply to *C*, sends
  - ♦ bridge sees frame from *D*
  - ♦ bridge notes that *D* is on port 2
  - ♦ bridge knows *C* on port 1, so **selectively** forwards frame via port 1



# Network Topology

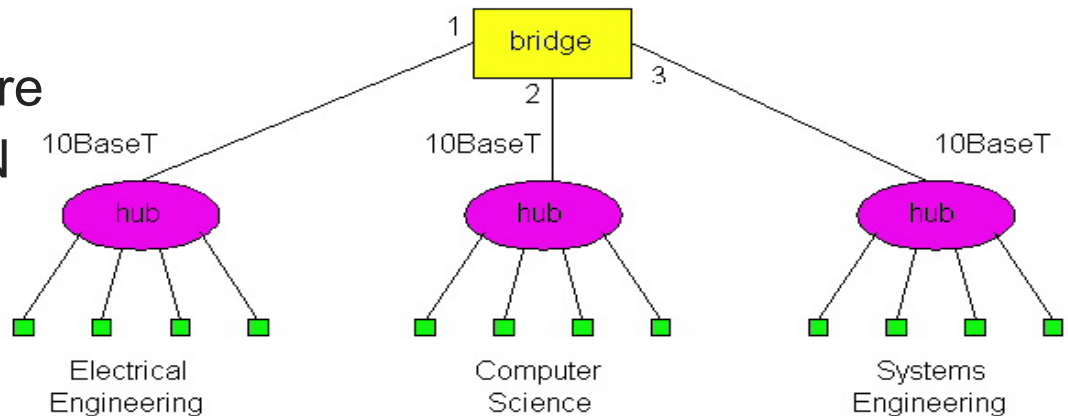
- Linear organization

- ◆ Inter-bridge hubs (e.g. CS) are single points of failure
- ◆ Unnecessary transit (e.g. EE $\leftrightarrow$ SE must traverse CS)



- Backbone/tree

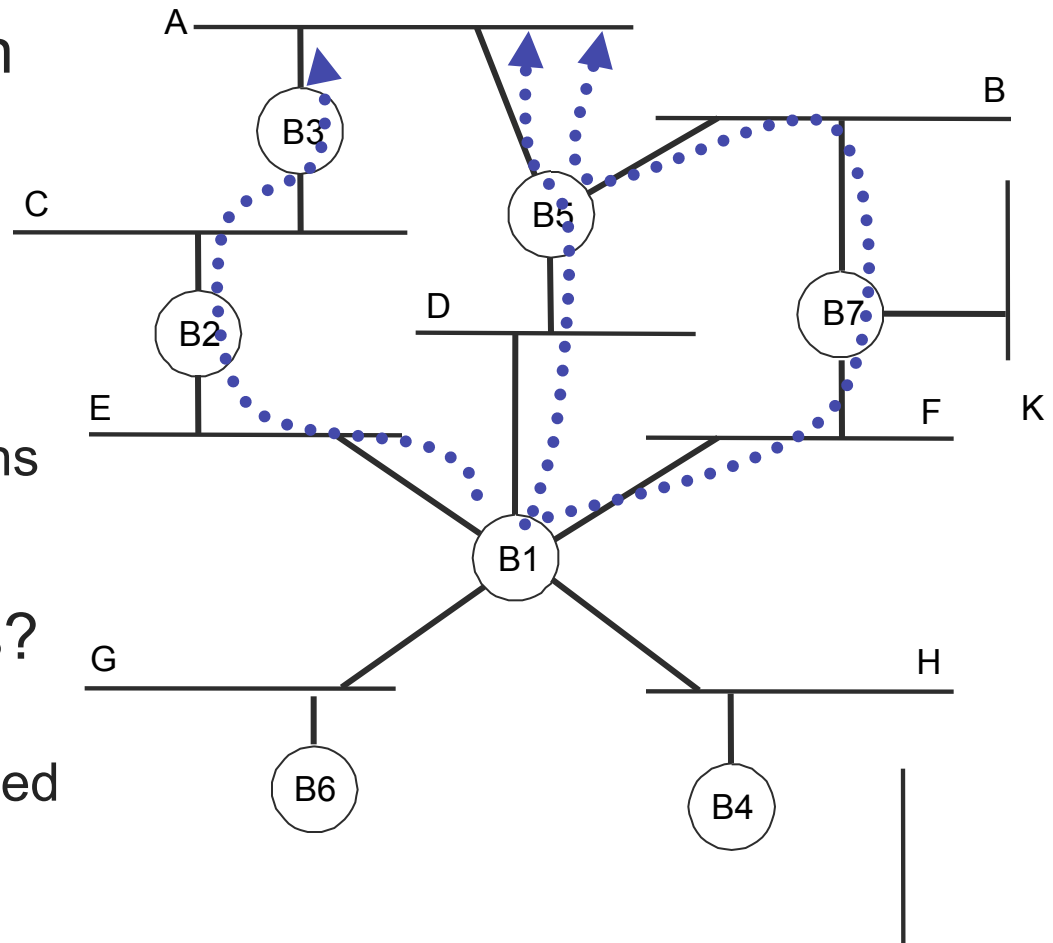
- ◆ Can survive LAN failure
- ◆ Manages all inter-LAN communication
- ◆ Requires more ports





# An Issue: Cycles

- Learning works well in tree topologies
- But trees are fragile
  - ◆ Net admins like redundant/backup paths
- How to handle Cycles?
  - ◆ Where should *B1* forward packets destined for LAN *A*?





# For Next Time

---

- Read 3.2-3.2.4 in P&D
- HW2 is out