

## EECS 122, Lecture 8

Kevin Fall  
kfall@cs.berkeley.edu

## Bridges

- Bridges interconnect network segments at the *link* layer (layer 2)
- Handle any layer 3 protocol (incl. non-routable ones); some can interconnect different media
- Mostly for LANs, also used in WANs (2 “half bridges” on ends of pt-to-pt links)

## Extended LANs

- Extending (interconnecting) multiple LANs. Appears as single LAN to layer 3.
- Essentially accepts and forwards all frames
- Benefits:
  - extend number of stations
  - extend size
  - limit interfering traffic

## The “no-frills” Bridge

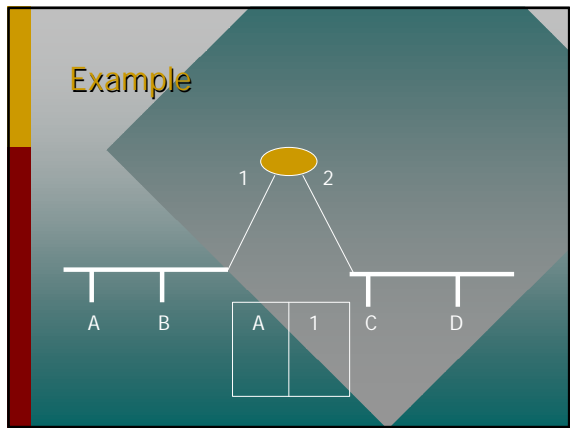
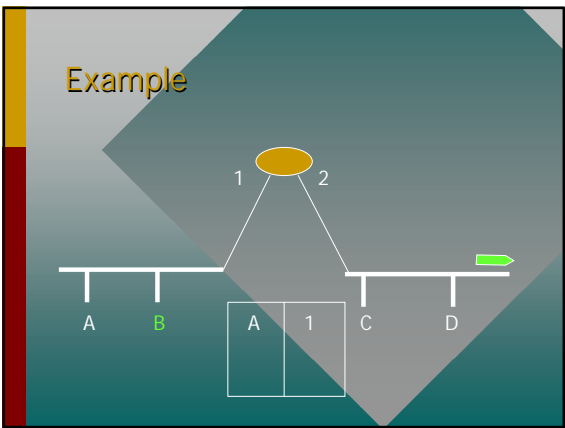
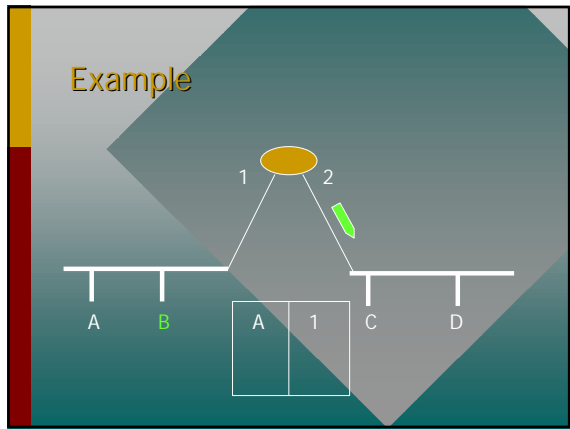
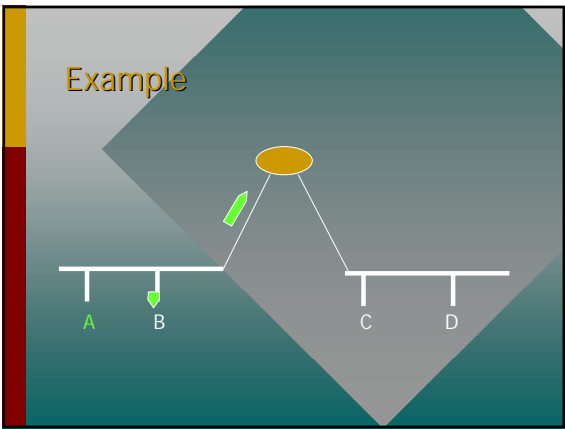
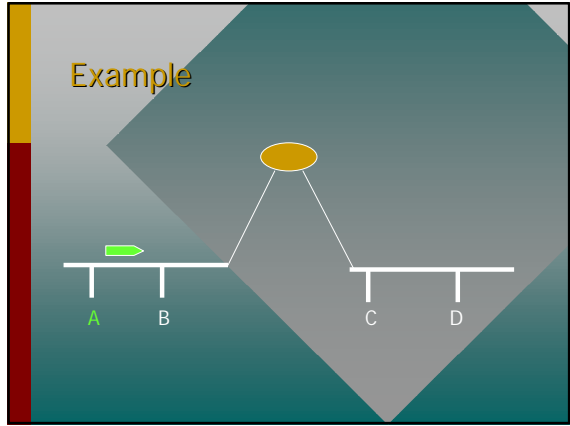
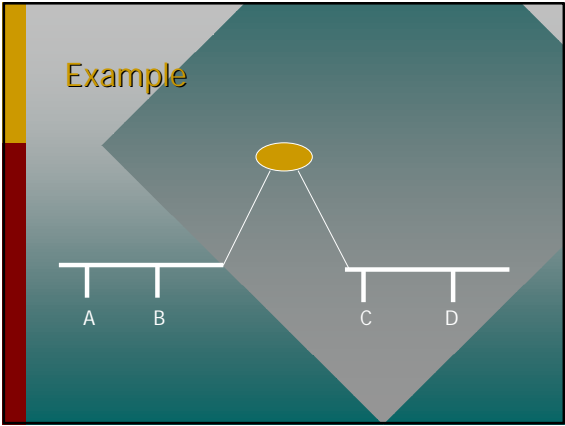
- Interconnect 2 or more LAN segments
- Listens in promiscuous mode, buffers packets and transmits them on other interfaces when able
- On average, still cannot exceed link bandwidth
  - bridge copies all traffic
  - small bursts accommodated in buffers

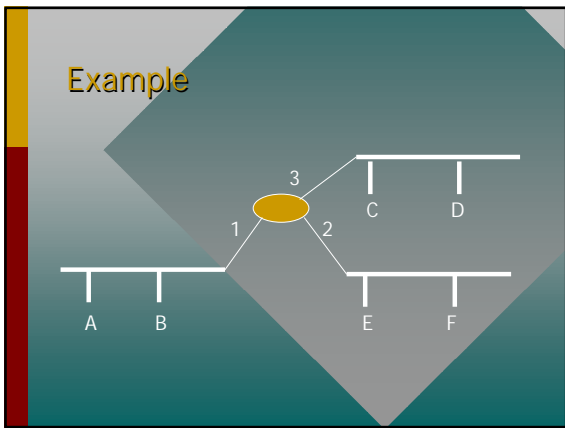
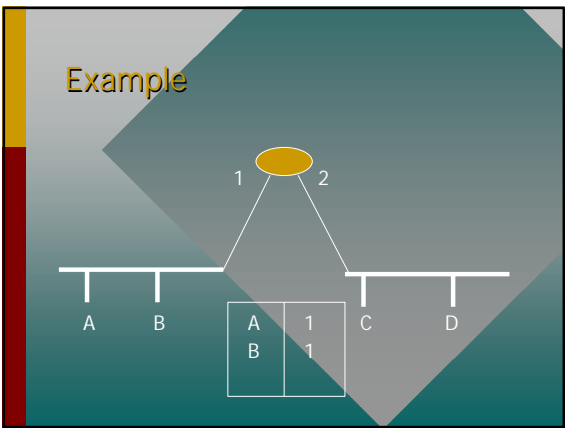
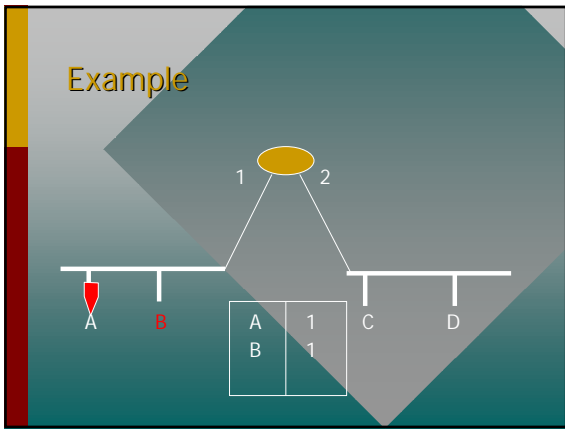
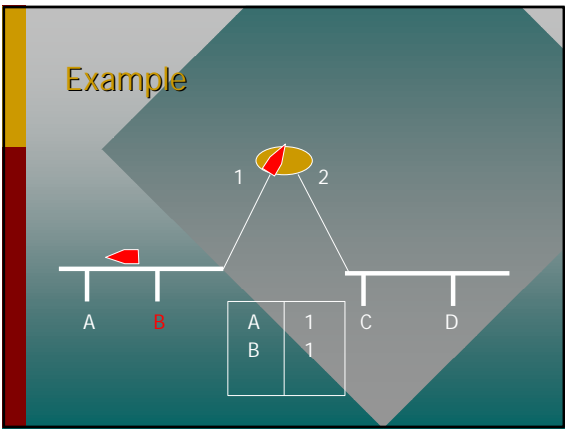
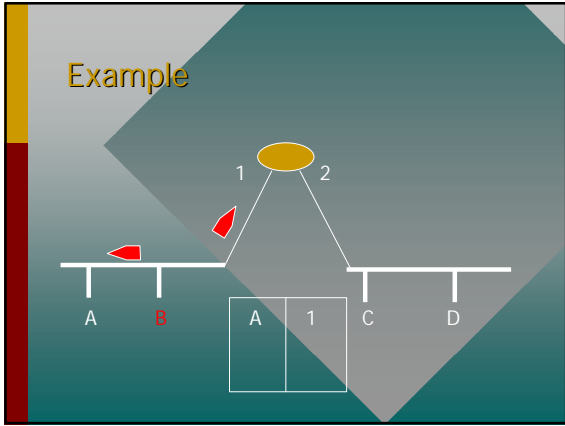
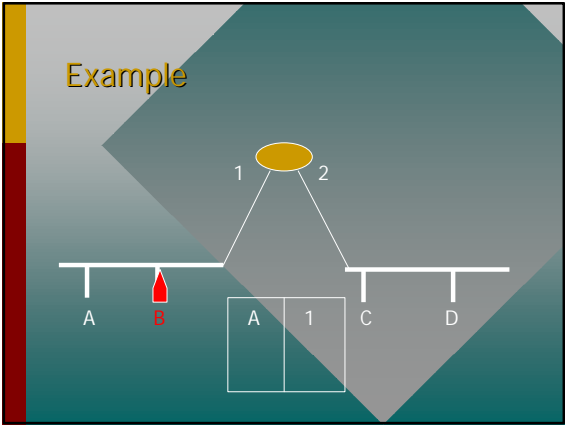
## The “learning” Bridge

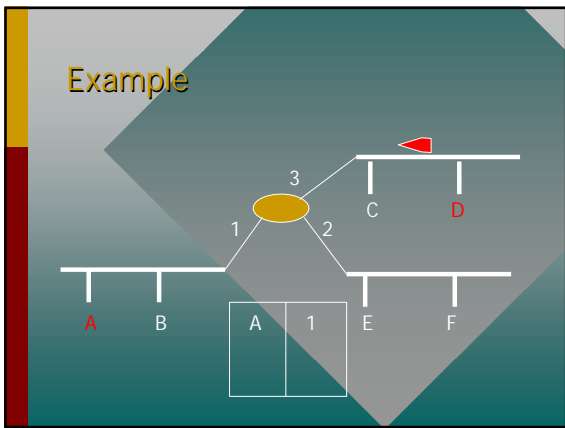
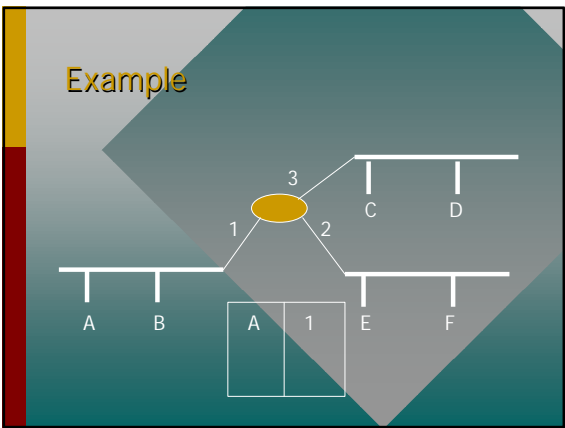
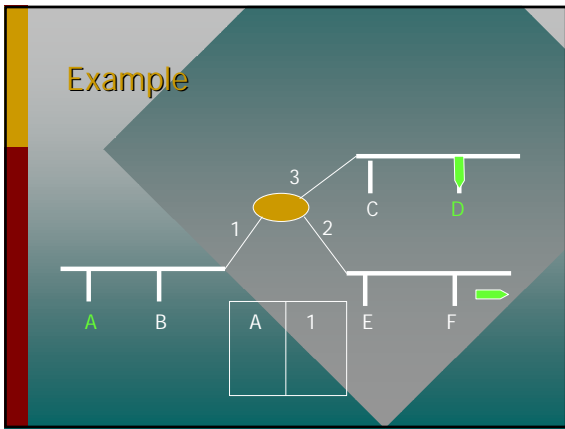
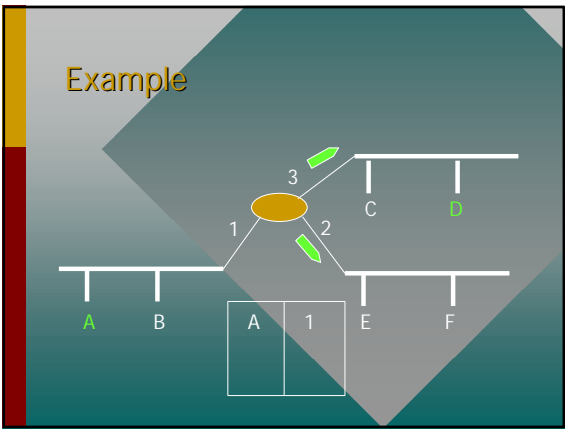
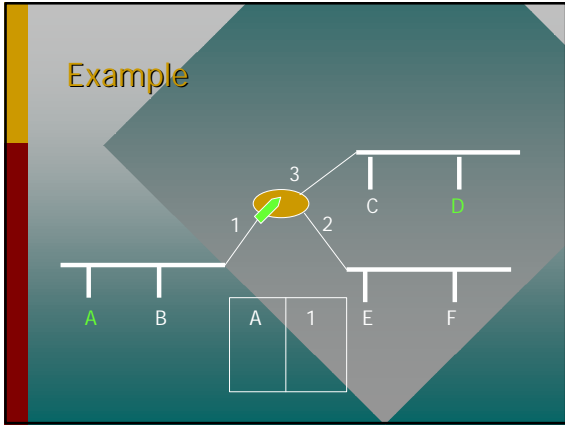
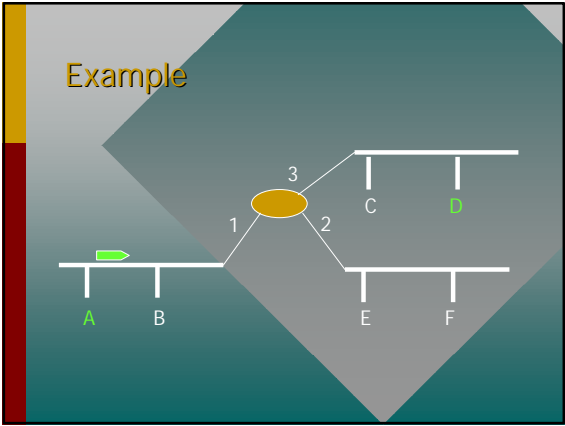
- Bridges “learn” which interfaces reach which end stations
  - could do this “by hand”, but a hassle
  - best if this happens *transparently*
- Learn by watching source addresses in frames
  - senders usually use their own addresses
  - (note that bridges don’t!)

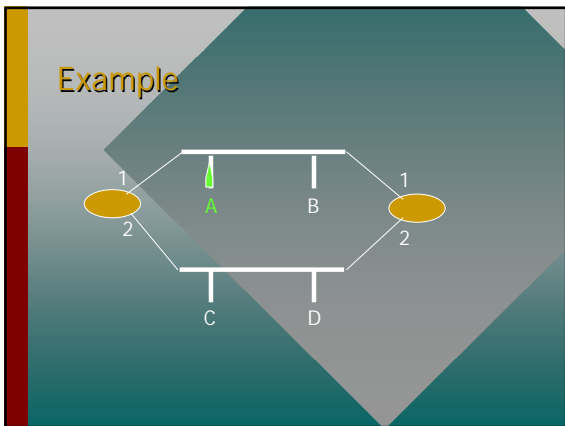
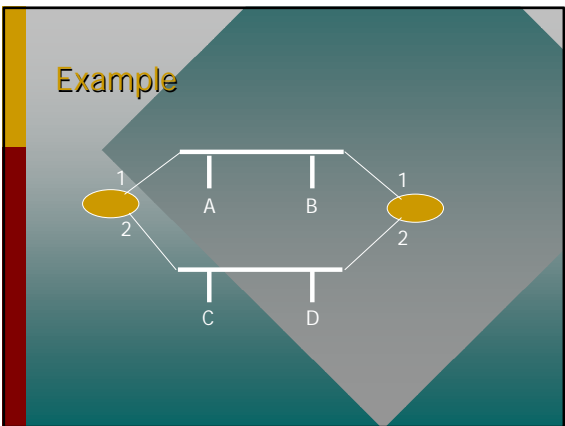
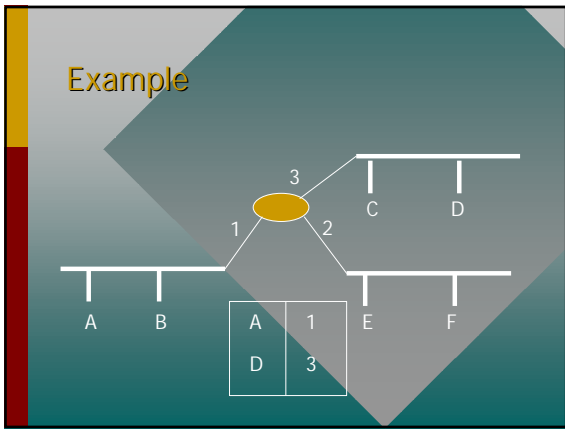
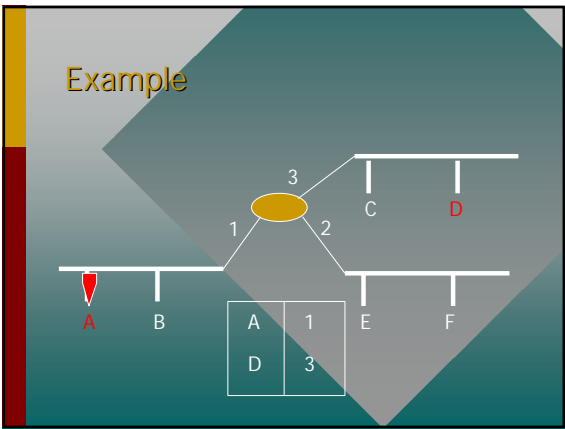
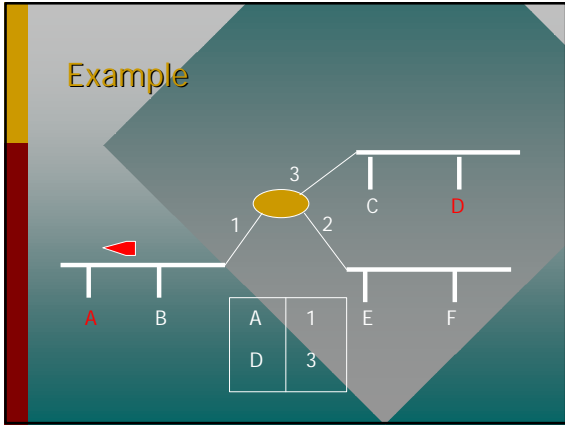
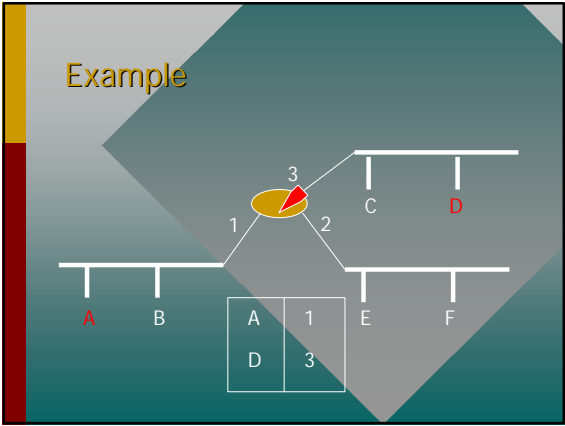
## Learning Strategy

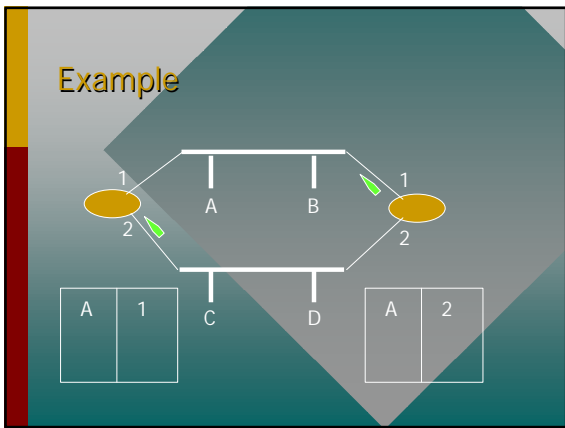
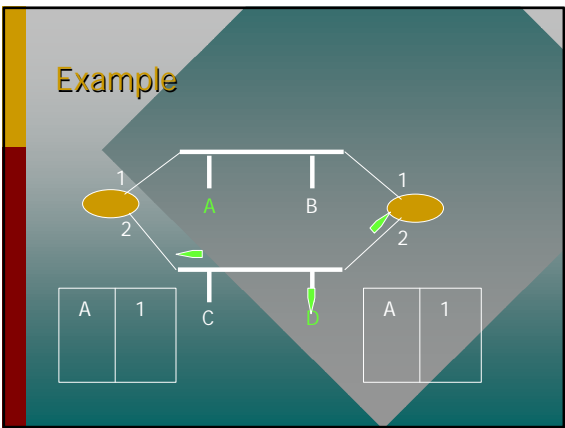
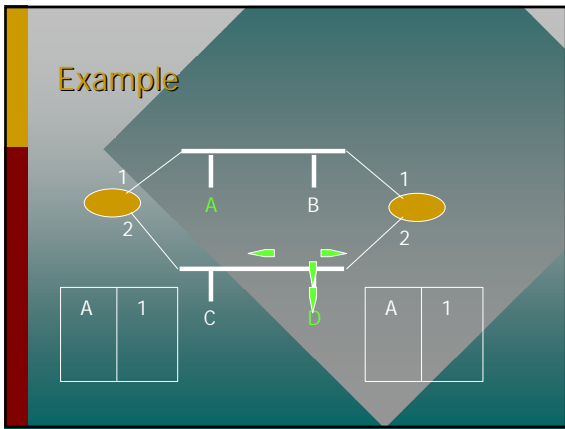
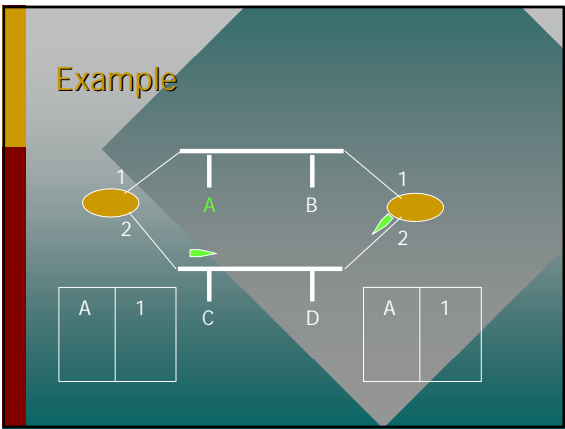
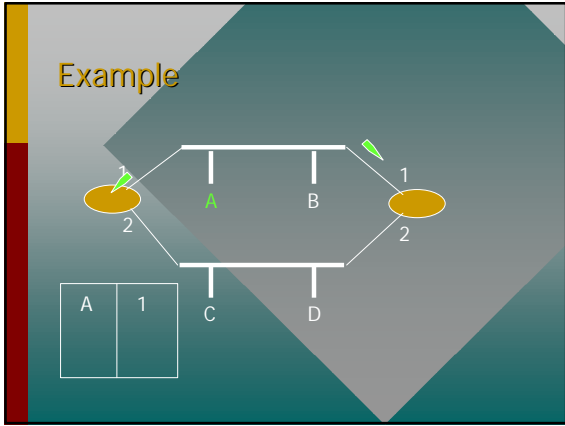
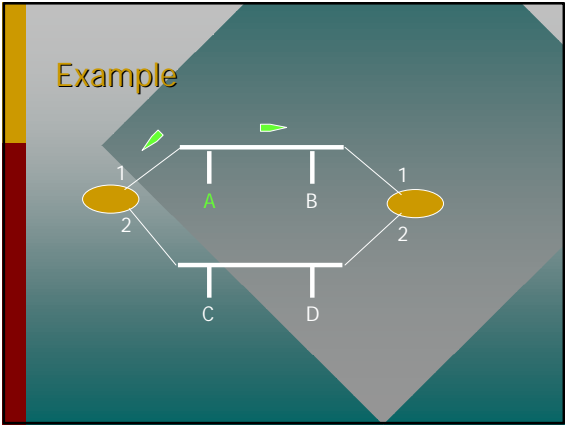
- Listen promiscuously for all traffic
- Store (src addr, port) tuple in “station cache” for each new sender observed
- For each received frame:
  - try to match frame dest to cache src entry
  - not there->send on all interfaces except rcv
  - is there->send on indicated, or filter if same as rcv interface
- Age cache entries

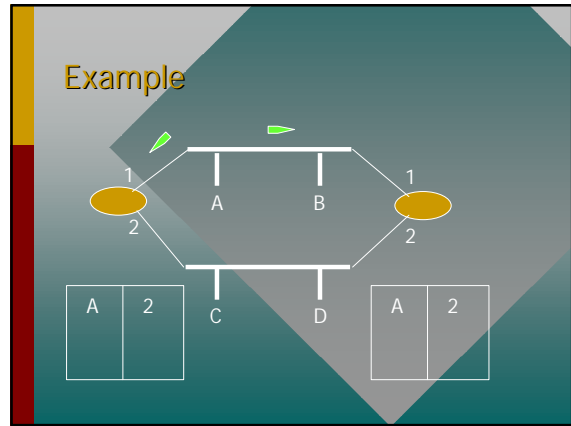
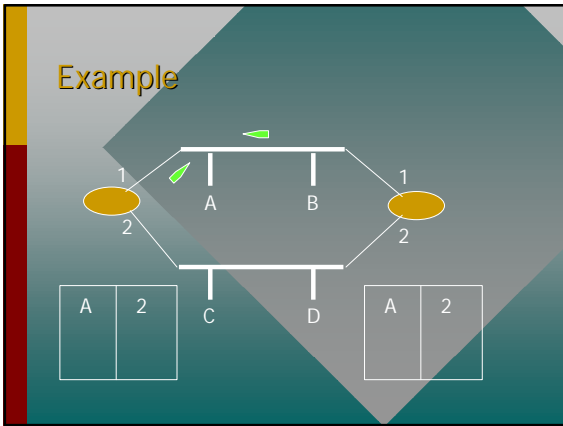












- ### Ouch... Loops Hurt
- With redundant paths, bridges can loop traffic
    - can happen forever (example)
    - with more than 2, can *cascade*
  - Cascade
    - each bridge with N interfaces may produce up to N-2 new copies!

- ### Loop Avoidance
- Consider LAN a graph  $G = (E, V)$ , with LANs as vertices, and bridges as edges [well, sort of... see footnote p.212]
  - Spanning Trees:
    - A spanning tree of an undirected, connected graph G is a subgraph which is both a tree and contains all vertices in G
    - Thus, the ST will throw out some edges and be cycle-free

- ### Spanning Tree
- Purpose will be to provide a single path to reach each network
  - Generally, graphs have many STs (even several MST's...CS 170)
  - Must be a distributed algorithm
  - Can result in some bridges not forwarding at all!

- ### Spanning Tree Computation
- Each bridge will decide over which ports it will forward frames
    - bridges have unique addresses *per port*
    - ports are also numbered by each bridge
    - bridges have a single unique identifier (e.g. the lowest address)

## Computation Outline

- Elect single bridge as root
- Calculate distance from each bridge to root bridge
- For each network, elect the bridge nearest the root to forward frames from that LAN to the root
- Choose a port on which to forward toward root (the *root port*)
- Select which ports are on the ST

## Configuration Messages

- Root election and ST formation are accomplished by *configuration messages*
  - messages sent to “all bridges” multicast address, using bridge’s src MAC address
  - Contents: Root ID, Bridge ID, Cost, [age]
    - Root ID: current assumed root ID
    - Bridge ID: sending bridge’s ID
    - Cost: cost of best path to root from sender
  - messages are *not* forwarded between LANs

## Election 1

- Bridges initially assume they are the root
  - uses its own ID as root, with zero cost
- Bridges save “best” configs they hear on each port (or its own):
  - $C1 > C2$  if  $root(C1) < root(C2)$ , otherwise
  - $C1 > C2$  if  $cost(C1) < cost(C2)$ , otherwise
  - $C1 > C2$  if  $bridgeID(C1) < bridgeID(C2)$
- Cost is # hops to root

## Election 2

- Upon receiving “better” config message, bridge stops sending its own config messages (but continues to forward others’ with a cost incremented by 1)
- Once stability is reached, only one bridge on each LAN (the *designated bridge*) is sending config messages on that LAN

## Calculating Root, Cost, and Port

- global root is MIN of local bridge ID and MIN of all received root IDs
- Distance to root will be smallest cost to global root plus one
- Root port is port on which message containing minimum cost to global root was received

## Calculating Designated Bridge

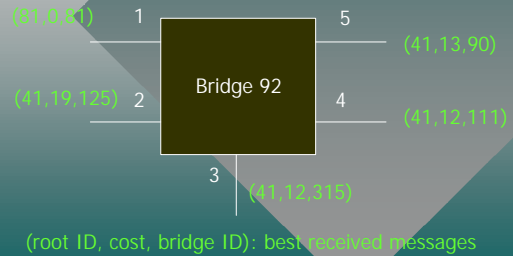
- Once root, cost, and port are known, a bridge knows what its own config messages would contain
- It will transmit its own config messages on ports where it is “best”



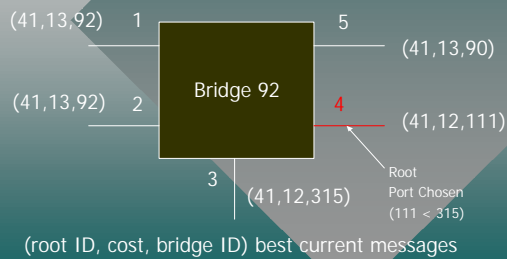
## Choosing Ports on the ST

- Put these ports in ST:
  - root port
  - all ports for which bridge is the designated bridge for the LAN
- Selected ports put into “forwarding” state (bridge will forward frames to/from)
- Other ports are “blocked” (no data, but configuration messages are processed)

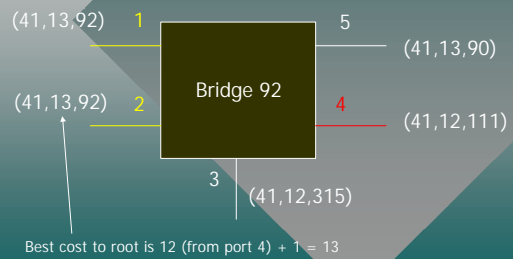
## Example [Perlman, p 58]



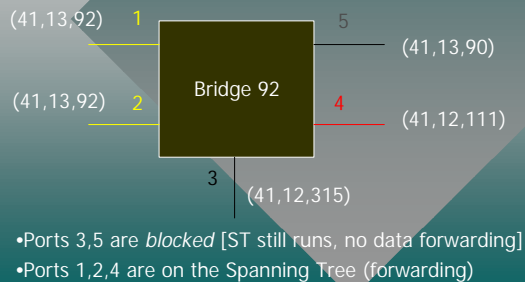
## Example (chooses 41 as root)



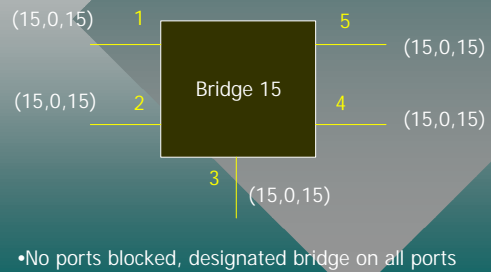
## Example (becomes designated bridge for 1,2)



## Example (becomes designated bridge for 1,2)



## Example (root bridge 15)



## Station Cache

- bridges learn and cache locations of stations
- stations may be moved, so bridges should “forget” about them
- --> use a time-out on station cache info
- not so easy to choose a suitable value

## Station Cache Timeout

- Too large:
  - traffic destined to moved node will be lost
- Too short:
  - un-necessary flooding (lots of traffic)
- So, if stations moving were the only concern, could use a timer on order of minutes

## Spanning Tree Recalculation

- ST recalculation can change active ports and associated station caches
- ST recalculation takes < minutes
- So, want small timeout (say, 15 secs)
- Standards committee could not make a definitive value

## Spanning Tree Recalculation

- Two admin-set values used:
  - long value, used in normal case
  - short value, used after ST re-compute
- Which to use? (how to detect ST recomp)
  - can bridges just detect this?
  - Some can, some can't

## Topology Change

- Want to inform all bridges, but without having traffic scale as # of bridges
- Operation
  - bridges noticing change send message on root port toward root
  - root config messages subsequently contain “topology changed” flag
  - a simple ACK scheme is used (see Perlman92 for details)

## Failures

- Algorithm so far doesn't detect or adapt to failures
- Approach
  - each per-port stored config message gets a *message age* field
  - if max age reached, bridge re-calculates
  - root bridge periodically transmits config message with age zero; these trigger designated bridges to send their config msgs

## A Small Snag...

- designated bridges receiving 0-age message from root send their own messages with age zero
- if that were the only time, no reason to include age info in config message
- new bridges' messages generate responses, but with aged value for root info; allows for discovery of failed root

## Spanning Tree Recalculation

- Recalculation on two events:
  - receipt of config message on port X
    - if better than current stored message for X, recalculate root, root path cost, and root port
  - timer tick
    - if the age in any stored config message expires, discard message and recalculate root, root path cost, and root port

## Temporary Loops

- during a topology change (new link/bridge starting or failing), time for info to propagate (esp. with congestion)
- Inconsistent data can cause:
  - loss of connectivity
  - temporary loops (worse!)

## Limiting Temporary Loops

- Probability is minimized by requiring bridge to wait before changing ports from blocking to forwarding state
- Wait time should be long enough for topology information to spread through the network
- ---> should be at least 2x max transit time across network

## Why is this?

- Assume bridges B1 and B2 are maximally distant from each other. B1 is root.
- B1 sends config message, not delayed. Sends another, very delayed (X secs), then B1 crashes.
- Bridges near B1 recompute, those near B2 wait  $\geq \text{max age} + X$  sec to recompute

## Why is this? [2]

- Suppose new root is B2
- Suppose 1st config message from B2 is delayed by X before reaching B1's area
- Then bridges near B1 will "hear" about new topology X time later
- Upshot: bridges near B1 could be up to 2X time out of date

## Bridge Limitations

- Scale: not very realistic to interconnect more than 10's of LANs
- Heterogeneity: really works best for homogeneous systems
- All broadcasts and multicasts are flooded