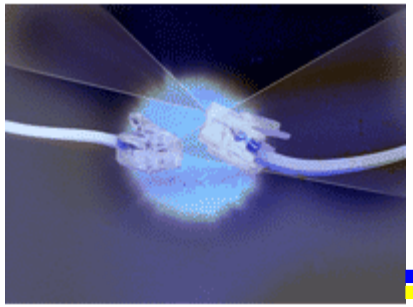
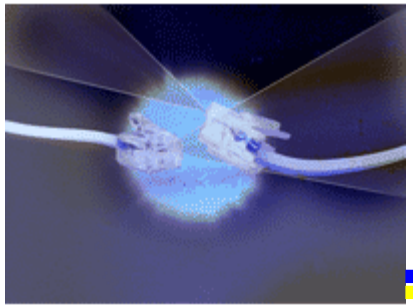


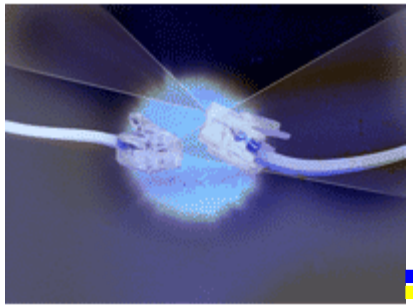
Bridging / LAN Switching

- A *bridge* or *LAN switch* is specifically designed to forward packets between shared-media LANs like Ethernets
- Simple Ethernet bridge – attach a node with two network adapters to each Ethernet
  - Run in promiscuous mode
  - Forwards every frame received from one LAN to the other



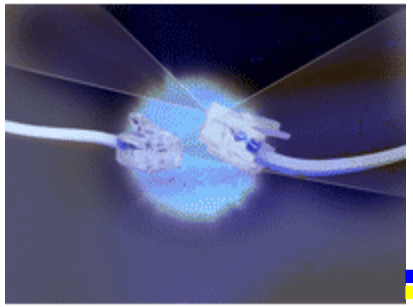
Ethernet Bridge (cont.)

- Advantages
  - Works around length limitations on single Ethernet (better than just using repeater)
  - Extremely simple
- Disadvantages
  - Forwards traffic that doesn't need to be forwarded; loads both LANs unnecessarily



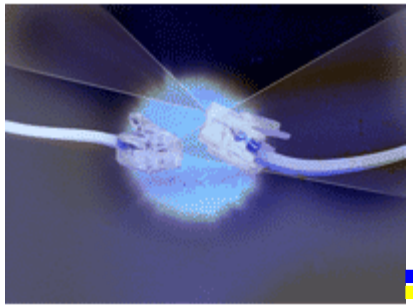
Learning Bridge

- Starts out doing simple bridging (forward all packets)
- Creates forwarding table dynamically
  - Extracts source address from every packet
  - Adds entry with address, interface port
- Uses forwarding table to decide when it should forward packets
  - Don’t forward if destination on same port on which packet was received



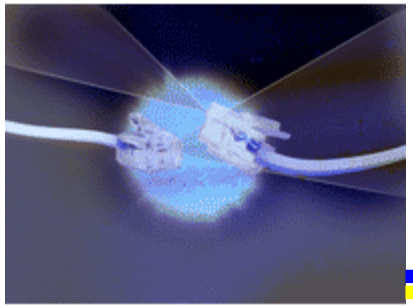
Learning Bridge (cont.)

- Fallback – if address not in table, always forward
- Forwarding table is not needed for correctness, only for optimization
  - If table fills up, bridge will still function even if it can't add an entry for a new node



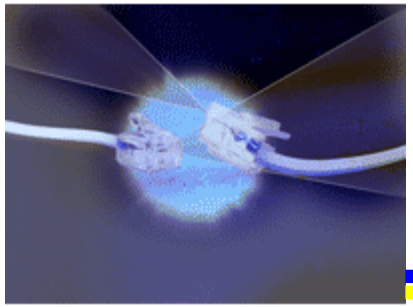
Learning Bridge (cont.)

- Include *TTL* (time to live) for each table entry
  - Reset entry for source address each time a packet is received
  - If TTL expires, remove entry from table
  - Allows bridge to adapt if node is moved to another LAN
  - Could also implement an LRU algorithm to clear space in cache using TTL



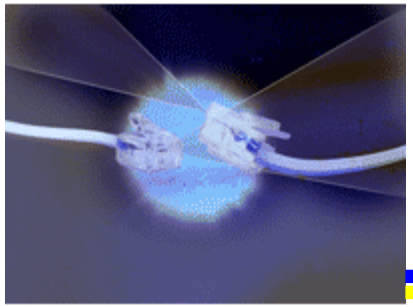
Cycles in LANs

- Bridges added to extend LAN – can also be added to provide redundancy
  - Allow for two different routes to a LAN, in case one of them fails
- If bridges are used to create an extended LAN that has a loop or cycle, the wheels fall off the whole network
  - Packets circulate through network forever



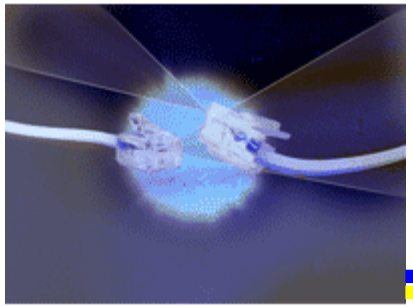
Removing Cycles

- Visualize extended LAN as a graph
  - Bridges and LANs are vertices
  - Links between LAN and bridge are edges
- Create a *spanning tree* that connects all vertices in the graph but contains no cycles
  - Actually only need to connect all LAN vertices; it's OK if a bridge becomes unconnected



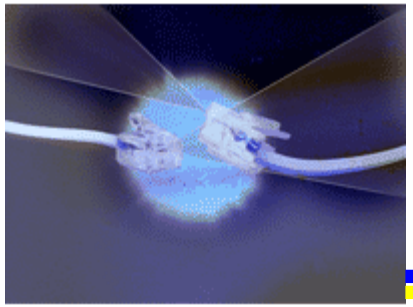
Spanning Tree Algorithm

- Distributed algorithm used by bridges to create the spanning tree
  - Each bridge must decide which interface ports it will use to forward packets
- Algorithm is dynamic, so bridges can reconfigure themselves in the event of a node or link failure
  - Restores the robustness that was the goal of creating loops in network



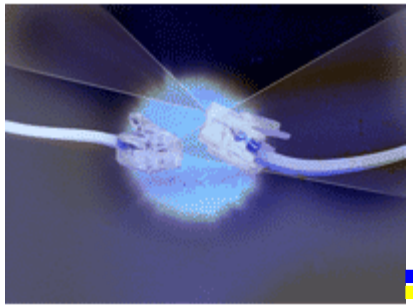
High-Level Spanning Tree Algorithm

- Assign an ID to each bridge
- Select bridge with smallest ID as root of tree (root bridge always forwards to all ports)
- Every other bridge computes shortest path to root bridge, notes which port on path
- All bridges on a LAN designate one bridge to forward frames toward the root
  - One closest to root wins – ties broken by choosing smallest bridge ID



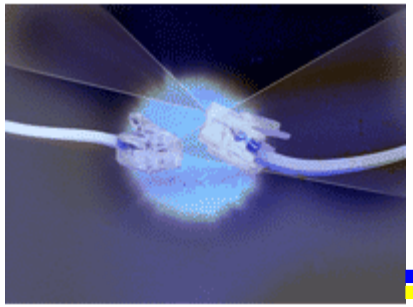
Now for some details …

- Bridges exchange configuration messages containing the following:
  - Bridge ID
  - ID that bridge believes to be the root bridge
  - Distance in hops from bridge to root bridge
- Each bridge records “best” config msg for each port
  - Both received and transmitted configs



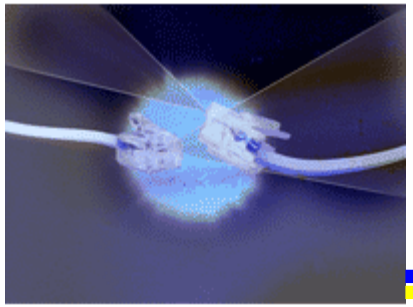
Spanning Tree Algorithm (cont.)

- Initially, every bridge assumes it is root, sends config msg with its ID and hop count of 0 on every port
- Receiving bridge compares config msg with the one it sent for the port. New one better if
  - It contains smaller bridge ID for root
  - It identifies root with same ID but smaller hop count
  - Root ID and distance equal, but sender has smaller ID



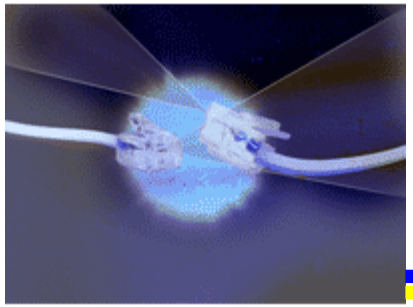
Spanning Tree Algorithm (cont.)

- If new config better, bridge increments distance to root count when storing
- When bridge receives config that indicates that it is not the root, it stops generating new config msgs. It only forwards config msgs from other bridges, after incrementing distance
- Likewise, if it receives config indicating it is not designated node for port (smaller distance or same distance and smaller ID), stops sending config msgs on that port



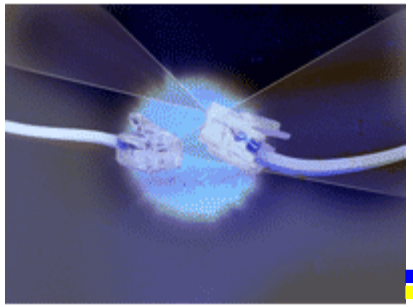
Spanning Tree Algorithm (cont.)

- After network stabilizes, only root bridge still generating config msgs
- Other bridges only forwarding configs over ports for which they are designated bridge
- Root generates config msgs periodically, and if a bridge doesn't receive, it starts the process over again
  - Will recover from failure, but won't route around congestion



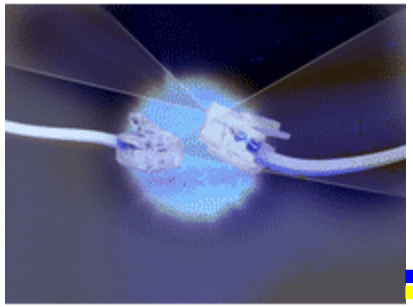
Broadcast / Multicast

- All bridges forward broadcast on all active ports
- Same thing done with multicast
  - Could be more efficient; might not be any nodes in multicast group on a particular LAN
  - Complicated to do, because you need to make sure another LAN downstream doesn't include nodes in multicast group
  - Need to send multicast packets periodically (put multicast address in source, bridge multicast address in destination)



Bridging Limitations

- Doesn’t scale well
  - Broadcasts in particular don’t scale
- Introduce *virtual LANs* (VLANs) to solve
  - Partition extended LAN into separate chunks (i.e. *color* extended LAN segments)
  - Packets can only go between LANs of same color
- Need to add VLAN header to packets



Bridging Limitations (cont.)

- Can only connect LANs with identical (or similar) frame headers
  - Bridge needs to find source address in frame
- Bridging LANs can introduce application complexity
  - Latency increases and can be variable
  - Bridges can congest and drop frames
  - Can now have multiple routes from source to destination