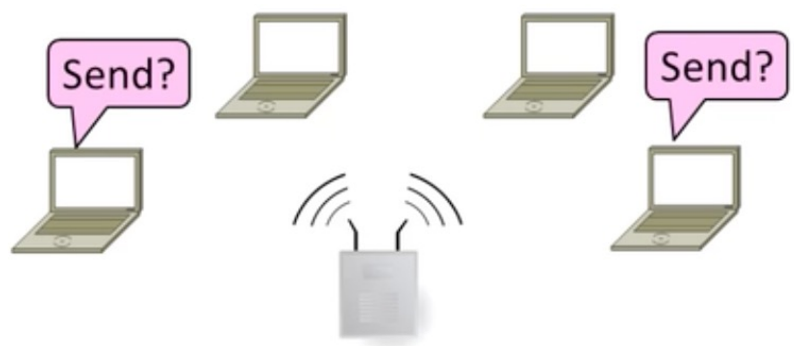
# Computer Network Design

Yalda Edalat – Spring 23

# Wireless LAN

- How do wireless nodes share a single link?

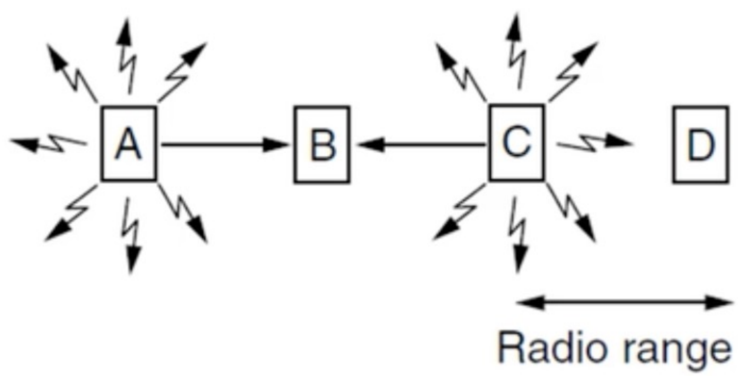


# Wireless Complications

- Wireless is more complicated than the wired case
    1. Nodes may have different areas of coverage – does not fit Carrier Sense
    2. Nodes can not hear while sending – can not Collision Detect

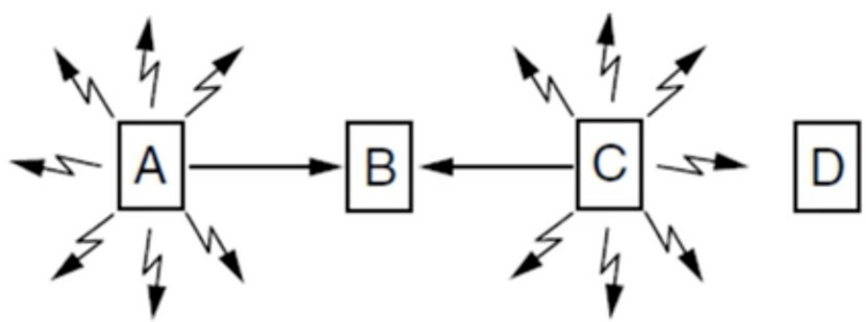
# Different Coverage Areas

- Wireless signal is broadcast and received nearby, where there is sufficient SNR



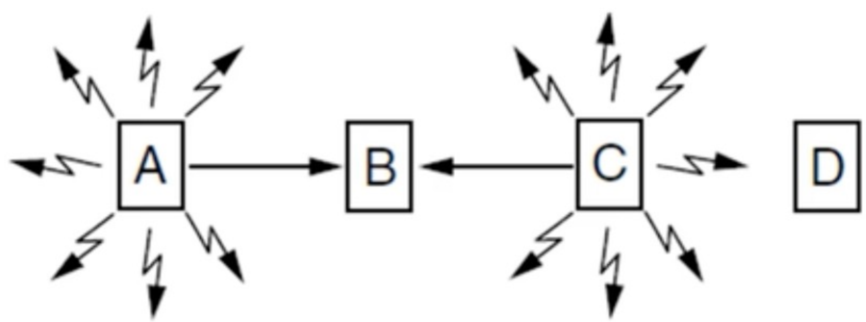
# Hidden Terminals

- Nodes A and C are hidden terminals when sending to B
  - Can't hear each other (to coordinate) yet collide at B
  - We want to avoid the inefficiency of collisions



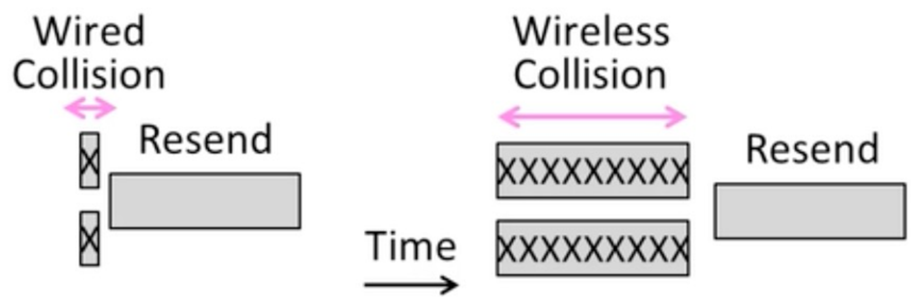
# Exposed Terminal

- B and C are exposed terminals when sending to A and D
  - Can hear each other yet don’t collide at receiver A and D
  - We want to send concurrently to increase performance



# Nodes Can Not Hear While Sending

- With wires, detecting collisions lowers their cost
- Nodes can't hear while sending -> More wasted time with wireless



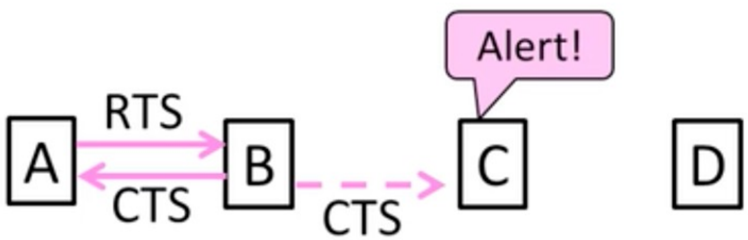
# Possible Solution: MACA

- MACA uses a short handshake instead of CSMA
  - 802.11 uses a refinement of MACA (later)

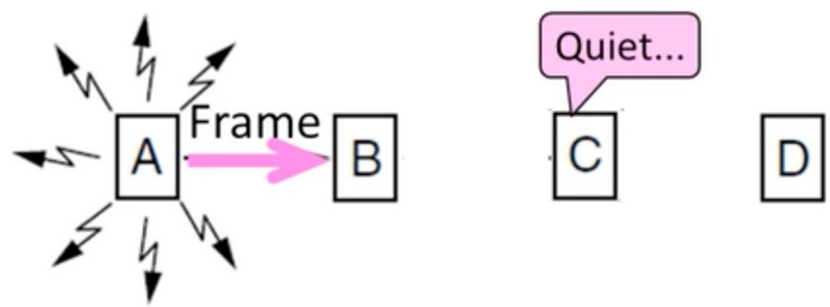
- Protocol rules: A sender node transmits a RTS (Request-To-Send, with frame length)
- The receiver replies with a CTS (Clear-To-Send, with frame length)
- Sender transmits the frame while nodes hearing the CTS stay silent
  - Collisions on the RTS/CTS are still possible, but less likely

# MACA - Hidden Terminals

- A->B with hidden terminal C
  1. A sends RTS, to B
  2. B sends CTS, to A and C too

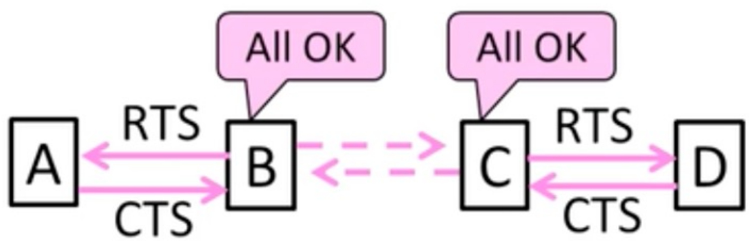




  3. A send frame while C defers



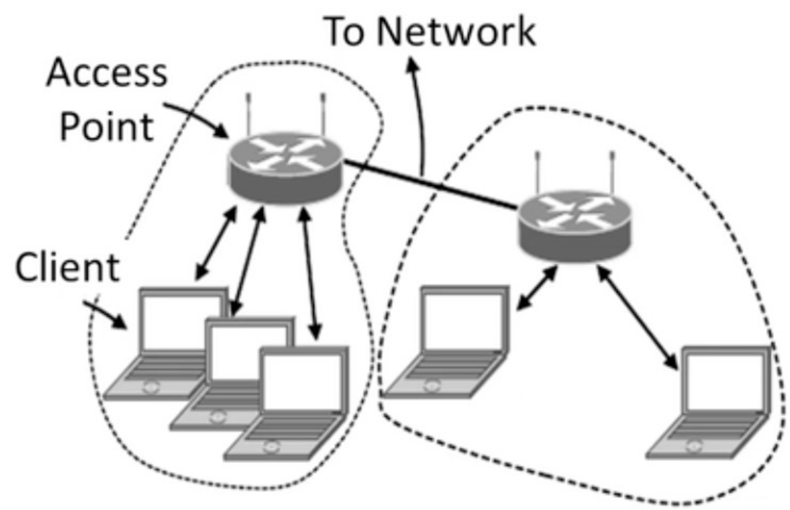
# MACA - Exposed Terminal

- B->A, C->D as exposed terminals
  - B and C send RTS to A and D
  - A and D send CTS to B and C

# 802.11, or WiFi

- Very popular wireless LAN started in the 1990s
- Clients get connectivity from a (wired) AP (Access Point)
- Various flavors have been developed over time
  - Faster, more features
  - E.g., 802.11a,b,g,n,ac,ax,…



# 802.11 Link Layer

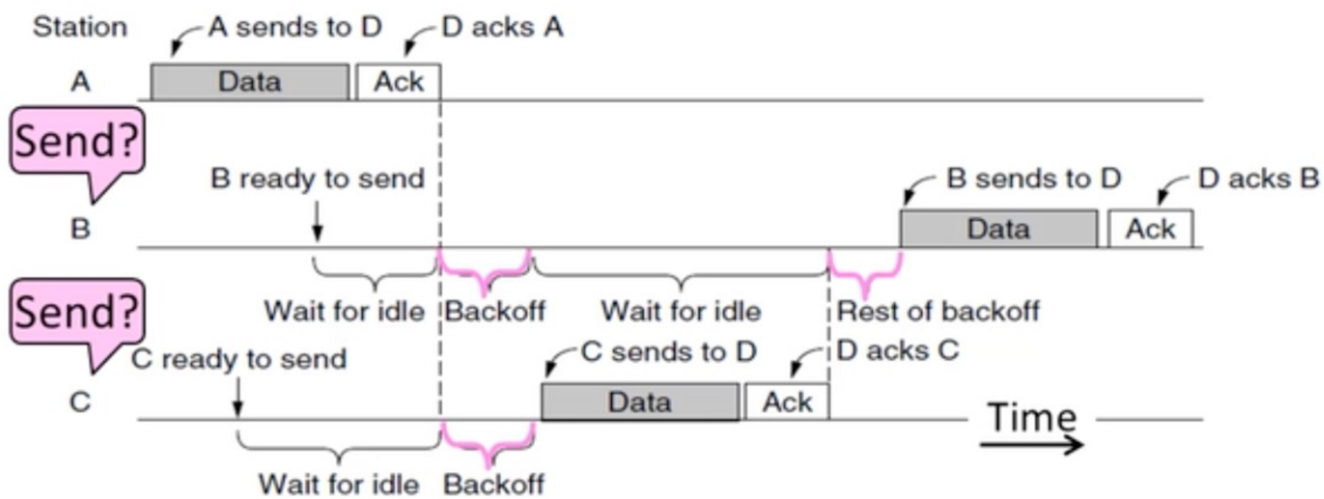
- Multiple access uses CSMA/CA, RTS/CTS optional
- Frames are ACKed and retransmitted
- Funky addressing (three addresses!) due to AP
- Errors are detected with a 32-bit CRC
- Many features (e.g., encryption, power saving,..)

Packet from Network layer (IP)

| | Frame control | Duration | Address 1 (recipient) | Address 2 (transmitter) | Address 3 | Sequence | Data | Check sequence |
|---|---|---|---|---|---|---|---|---|
| Bytes | 2 | 2 | 6 | 6 | 6 | 2 | 0–2312 | 4 |

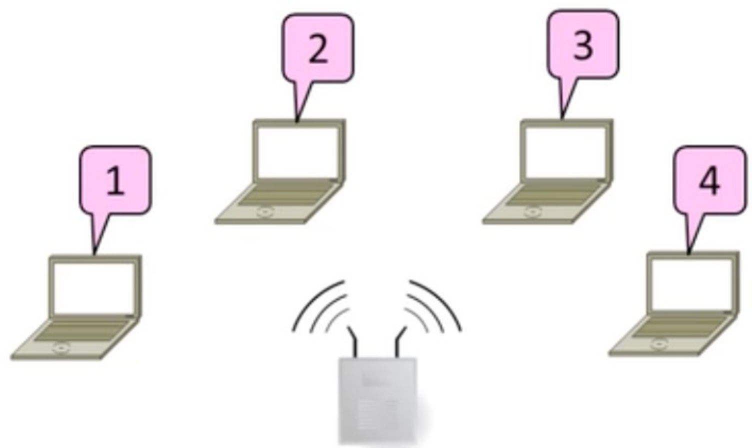
# 802.11 CSMA/CA for Multiple Access

- Sender avoids collisions by inserting small random gaps
  - E.g, when both B and C send, C picks a smaller gap, goes first



# Contention-Free Multiple Access Protocols

- A new approach to multiple access
  - Based on turns, not randomization



# Issues with Random Multiple Access

- CSMA is good under low load:
  - Grants immediate access
  - Little overhead (few collisions)

- But not so good under high load:
  - High overhead (expect collisions)
  - Access time varies (lucky/unlucky)

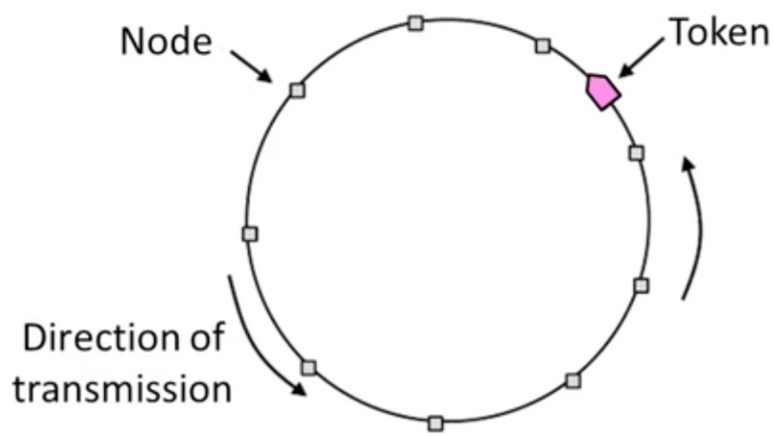
- We want to do better under load!

# Turn-Taking Multiple Access Protocols

- They define an order in which nodes get a chance to send
  - Or pass, if no traffic at present
- We just need some ordering …
  - E.g., Token ring
  - E.g., Node addresses

# Token Ring

- Arrange nodes in a ring; token rotates “permission to send” to each node in turn



# Turn-Taking Advantages

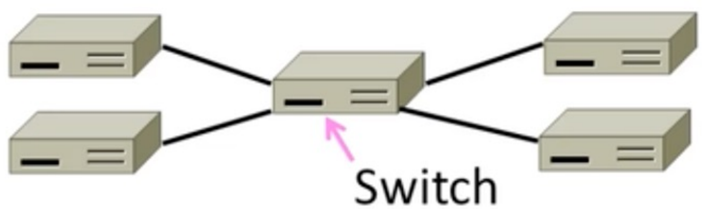
- Fixed overhead with no collisions
  - More efficient under load
- Regular chance to send with no unlucky nodes
  - Predictable service, easily extended to guaranteed quality of service

# Turn-Taking Disadvantages

- Complexity
  - More things that can go wrong than random access protocols!
    - E.g., what if the token get lost?
  - Higher overhead at low load

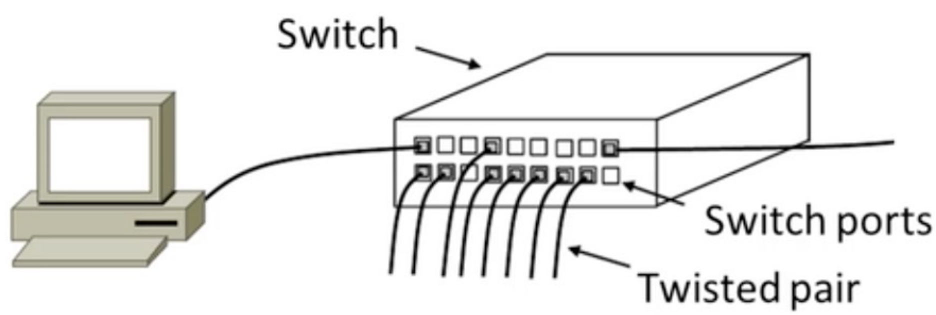
# LAN Switches

- How do we connect nodes with a switch instead of multiple access?
  - Uses multiple links/wires
  - Basis of modern (switched) Ethernet



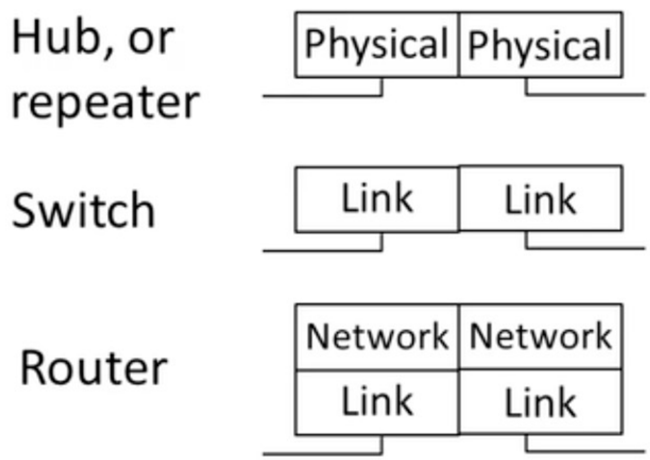
# Switched Ethernet

- Hosts are wired to Ethernet switches with twisted pair
  - Switch serves to connect the hosts
  - Wires usually run to a closet



# What Is In The Box?

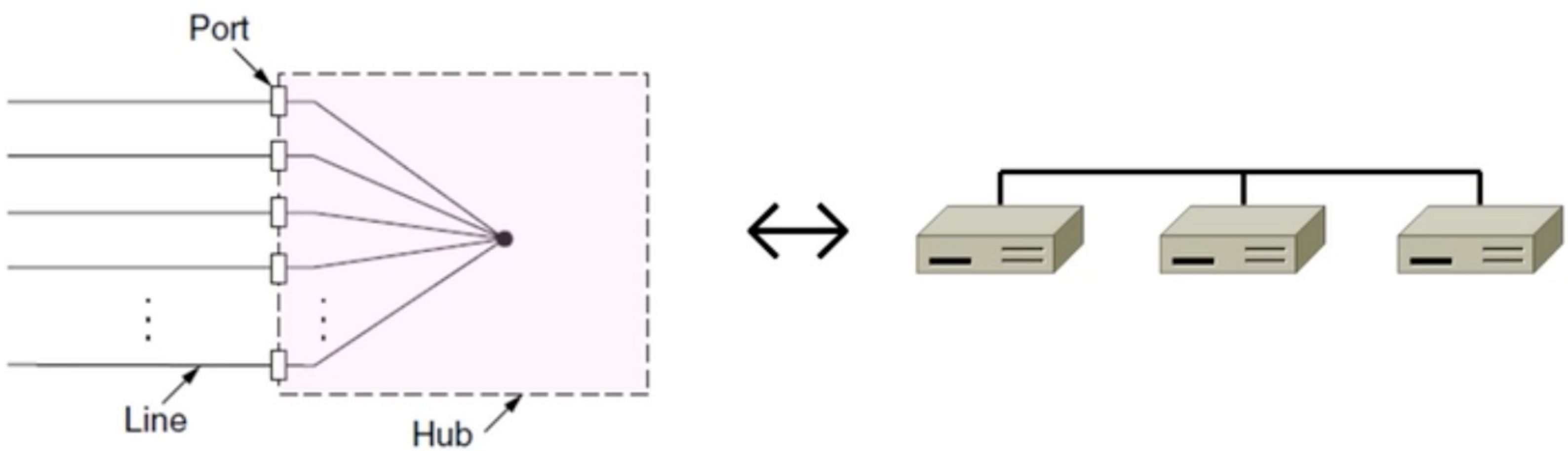
- Remember from protocol layers:




All look like this:

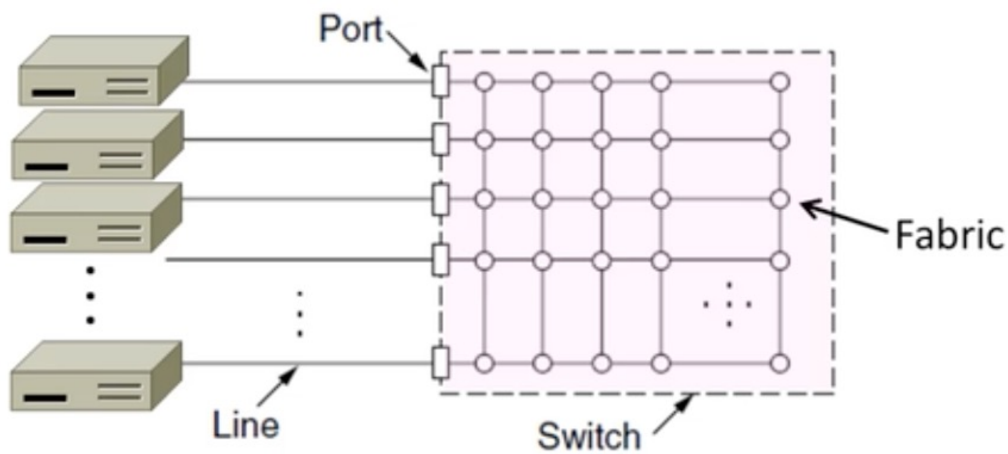
# Inside a Hub

- All ports are twisted together; more convenient and reliable than a single shared wire



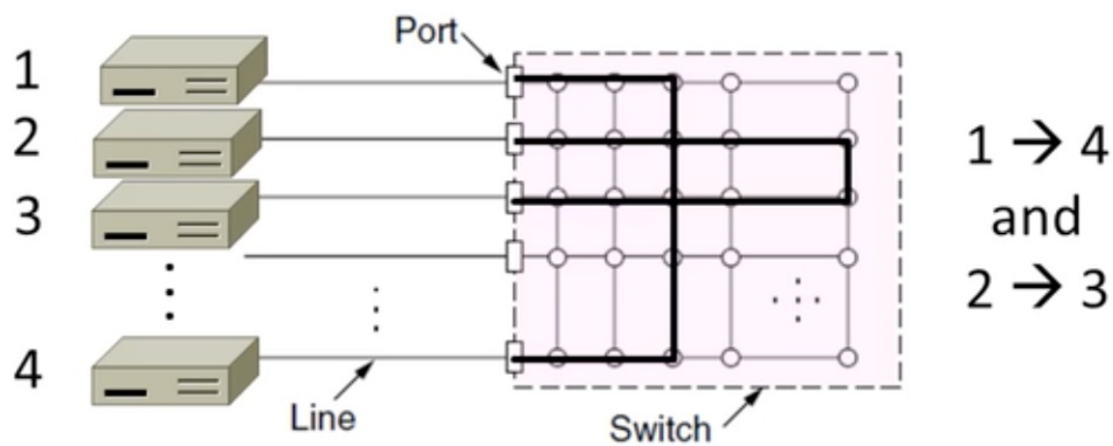
# Inside a Switch

- Uses frame addresses to connect input port to the right output port; multiple frames may be switched in parallel



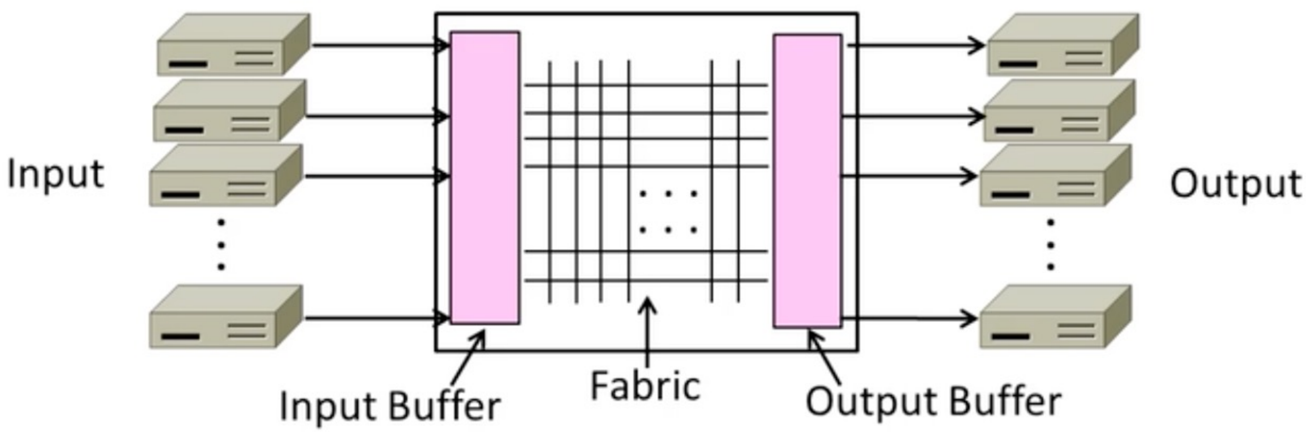
# Inside a Switch (2)

- Port may be used for both input and output (full-duplex)
  - Just send, no multiple access protocol



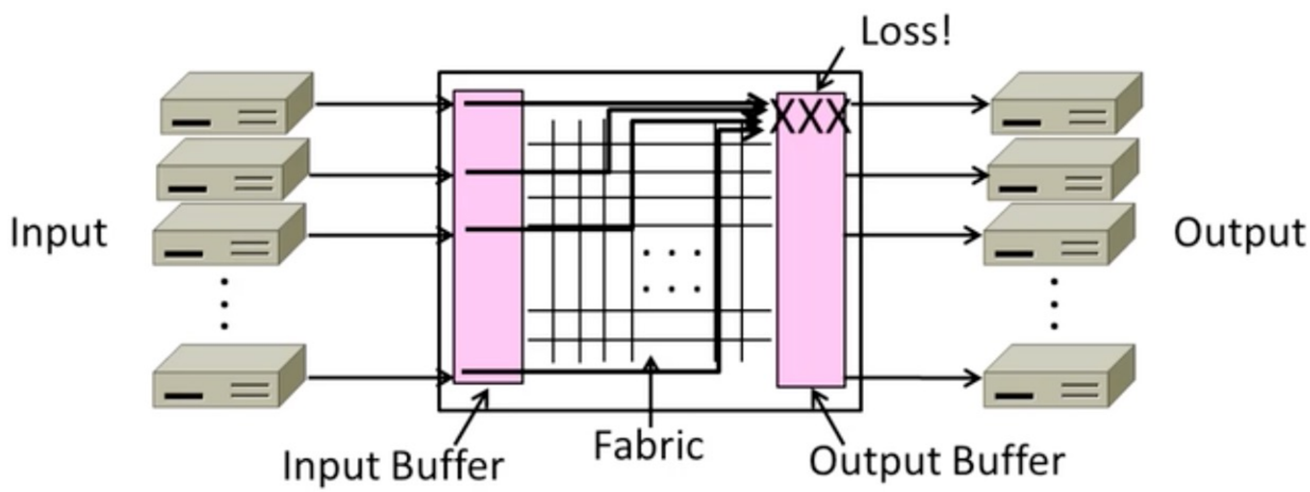
# Inside a Switch (3)

- Need buffers for multiple inputs to send to one output



# Inside a Switch (4)

- Sustained overload will fill buffer and lead to frame loss

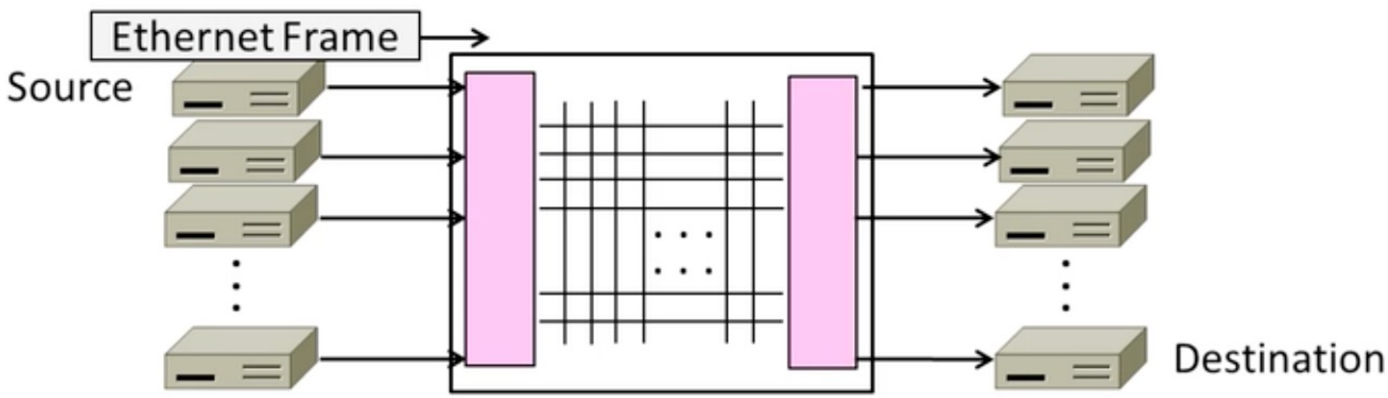


# Advantages of Switches

- Switches and hubs have replaced the shared cable of classic Ethernet
  - Convenient to run wires to one location
  - More reliable; wire cut is not a single point of failure that is hard to find
- Switches offer scalable performance
  - E.g., 100 Mbps per port instead of 100 Mbps for all nodes of shared cable/hub

# Switch Forwarding

- Switch needs to find the right output port for the destination address in the Ethernet frame. How?
  - Want to let hosts be moved around readily; don't look at Ip



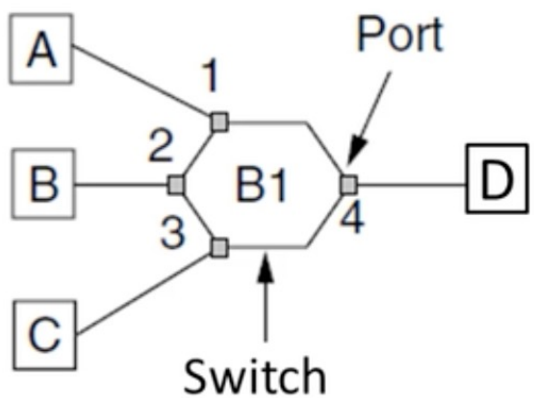
# Backward Learning

- Switch forwards frames with a port/address table as follows:

1. To fill the table, it looks at the source address of input frames
2. To forward, it sends to the port, or else broadcasts to all ports

# Backward Learning (2)

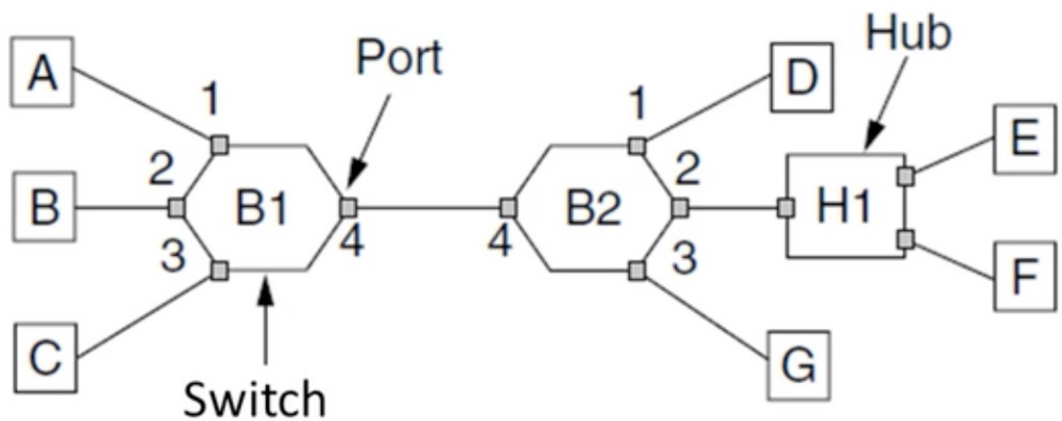
1. A sends to D




| Address | Port |
|---|---|
| A | |
| B | |
| C | |
| D | |

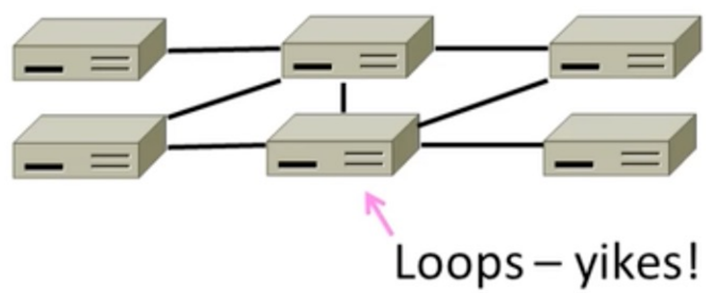
# Learning with Multiple Switches

- Just works with multiple switches and a mix of hubs **assuming no loops**, e.g., A sends to D then D sends to A



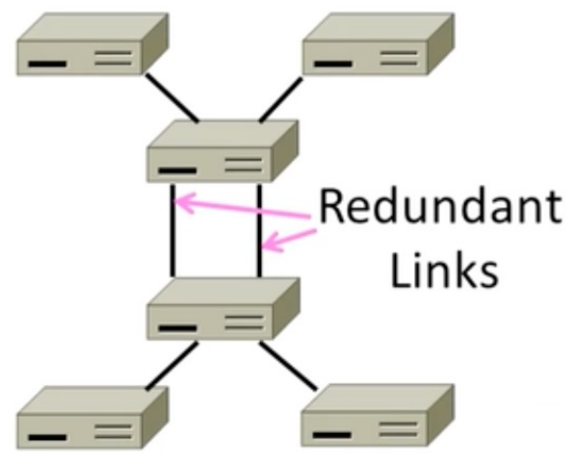
# Topic

- How can we connect switches in any topology so they just work?



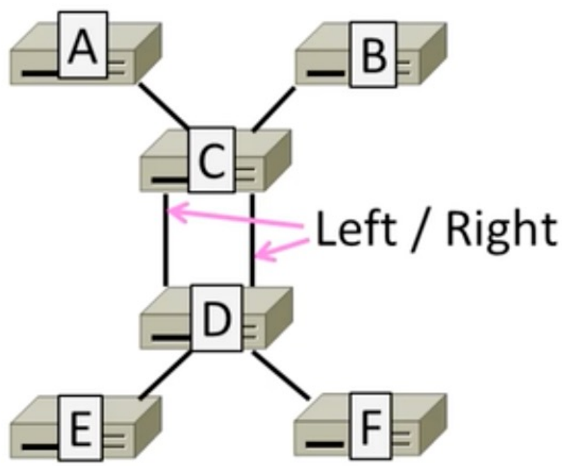
# Problem – Forwarding Loops

- May have a loop in the topology
  - Redundancy in case of failures
  - Or a simple mistake
- Want LAN switches to "just work"
  - Plug and play, no changes to hosts
  - But loops cause the problem



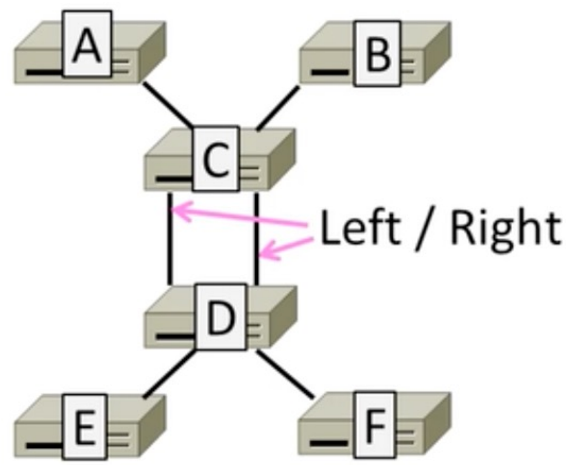
# Forwarding Loops (2)

- Suppose the network is started and A sends to F. What happens?



# Forwarding Loops (3)

- Suppose the network is started and A sends to F. What happens?
  - A->C->B, D-left, D-right
  - D-left->C-right, E, F
  - D-right->C-left, E, F
  - C-right->D-left, A, B
  - C-left->D-right, A, B
  - D-left->….
  - D-right->….

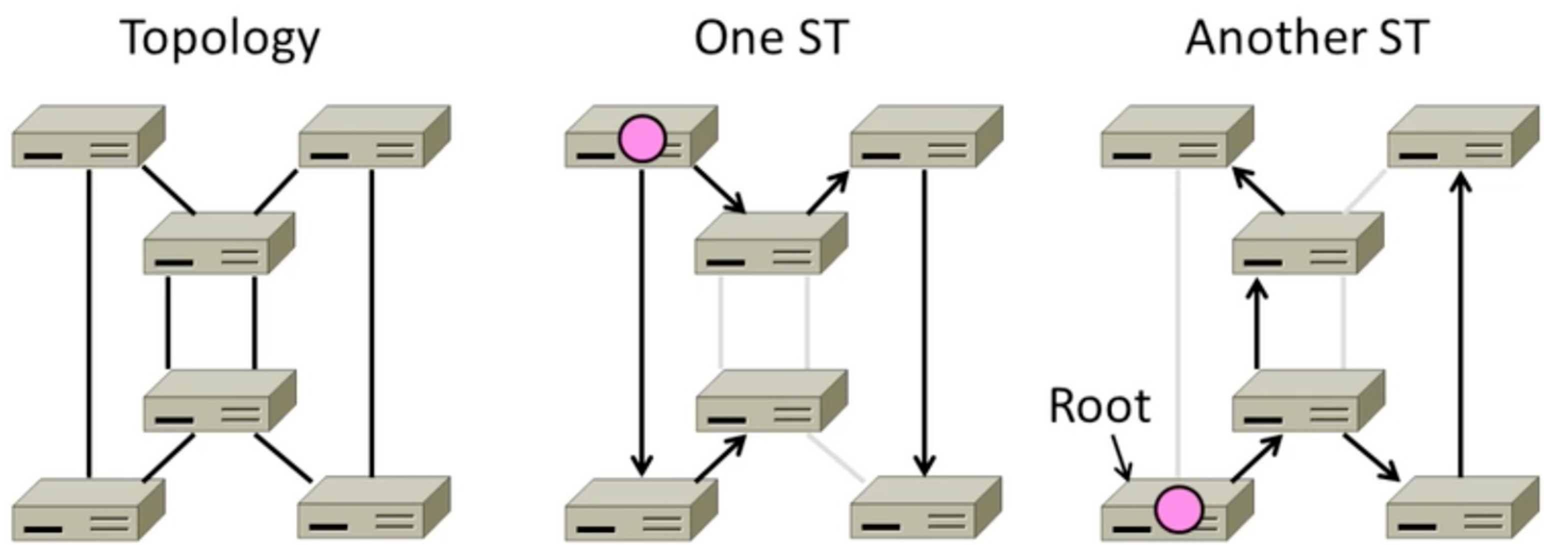


# Spanning Tree Solution

- Switches collectively find a spanning tree for the topology
  - A subset of links that is a tree (no loops) and reaches all switches
  - They switches forward as normal on the spanning tree
  - Broadcasts will go up to the root of the tree and down all the branches

- Done with:
  - Spanning Tree Protocol (STP)
  - Rapid-Spanning Tree Protocol (RSTP)
  - PVST+
  - Rapid-PVST+

# Spanning Tree (3)



# Spanning Tree Algorithm

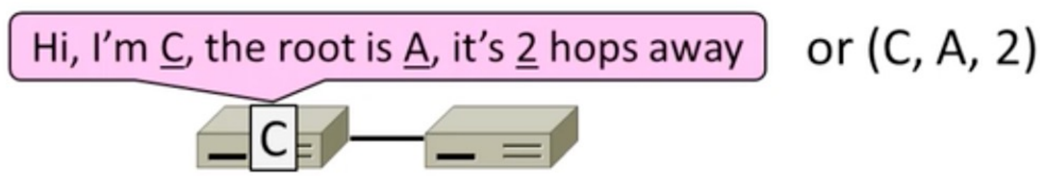
- Rules of the distributed game:
  - All switches run the same algorithm
  - They start with no information
  - Operate in parallel and send messages
  - Always search for the best solution
- Ensures a highly robust solution
  - Any topology , with no configuration
  - Adapts to link/switch failures,…

# Spanning Tree Algorithm (2)

- Outline:
  1. Elect a root node of the tree (switch with the lowest address)
  2. Grow tree as shortest distances from the root (using lowest address to break distance ties)
  3. Turn off ports for forwarding if they are not on the spanning tree

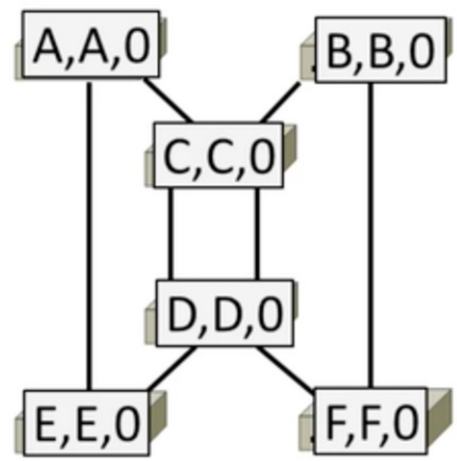
# Spanning Tree Algorithm (3)

- Details:
  - Each switch initially believes it is the root of the tree
  - Each switch sends periodic updates to neighbors with:
    - Its address, address of the root and distance (in hops) to root
  - Switches favors ports with shorter distances to lowest root
    - Uses lowest address as a tie for distances



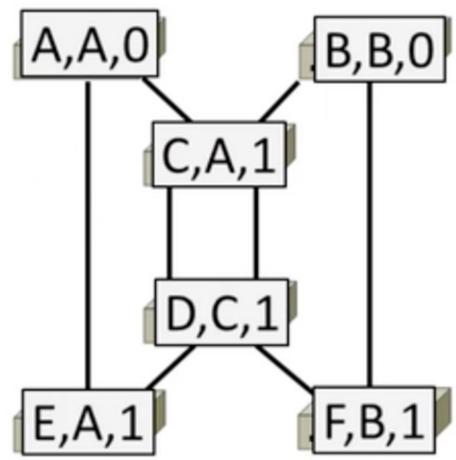
# Spanning Tree Example

- $1^{st}$ round, sending:
  - A sends (A, A, 0) to say it is root
  - B, C, D, E and F do likewise
- $1^{st}$ round, receiving:
  - A still thinks it is (A, A, 0)
  - B still thinks (B, B, 0)
  - C updates to (C, A, 1)
  - D updates to (D, C, 1)
  - E updates to (E, A, 1)
  - F updates to (F, B, 1)



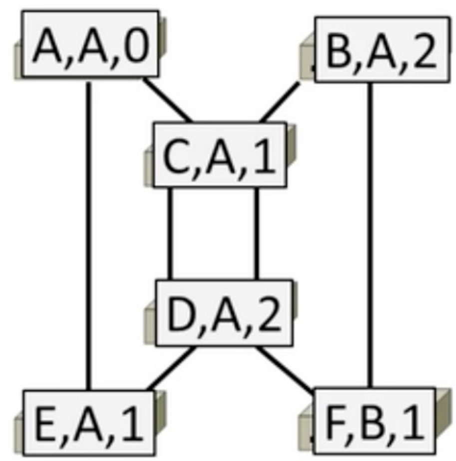
# Spanning Tree Example (2)

- $2^{nd}$ round, sending:
  - Nodes send their updated state
- $2^{nd}$ round receiving:
  - A remains (A, A, 0)
  - B updates to (B, A, 2) via C
  - C remains (C, A, 1)
  - D updates to (D, A, 2) via C
  - E remains (E, A, 1)
  - F remains (F, B, 1)



# Spanning Tree Example (3)

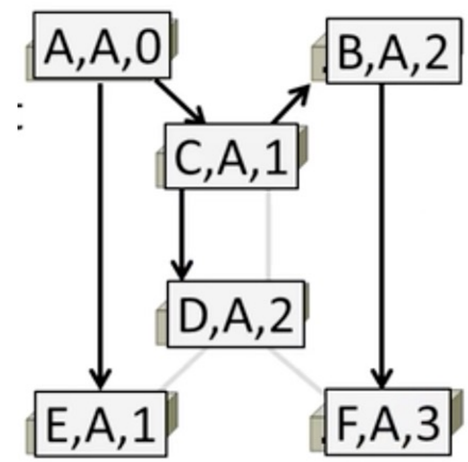
- 3rd round, sending:
  - Nodes send their updated state
- 3rd round receiving:
  - A remains (A, A, 0)
  - B remains (B, A, 2) via C
  - C remains (C, A, 1)
  - D remains (D, A, 2) via C-left
  - E remains (E, A, 1)
  - F updates to (F, A, 3) via B



# Spanning Tree Example (4)

- $4^{th}$ round
  - Steady-state has been reached
  - Nodes turn off forwarding that is not on the spanning tree

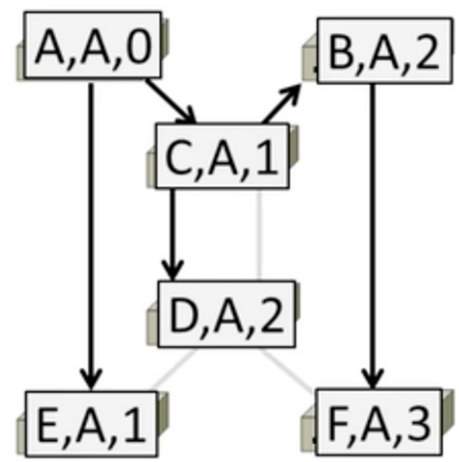
- Algorithm continues to run
  - Adapts by timing out information
  - E.g., if A fails, other nodes forget it, and B will become the new root



# Spanning Tree Example (5)

- Forwarding proceeds as usual on the ST
- Initially D sends to F:

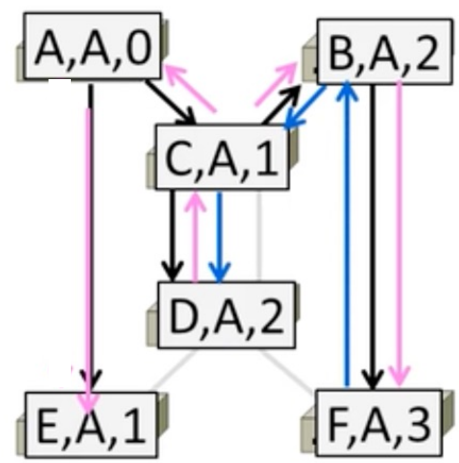
- And F sends back to D:



# Spanning Tree Example (6)

- Forwarding proceeds as usual on the ST
- Initially D sends to F:
  - D->C-left
  - C->A, B
  - A->E
  - B->F
- And F sends back to D:
  - F->B
  - B->C
  - C->D

  (Not such a great route!)



# To-do

- Quiz next week
- Lab 1 will be posted this week
- Research references due March $1^{st}$