Today's Agenda

- Previous class review (a quick refresher)
  - Learning Bridges
  - Distributed Spanning Tree Algorithm
    - Exercise to reinforce concept [Ex. 13, Page 288]
  - Internet Protocol
  - IP Datagram Format
  - IP Fragmentation and Reassembly
  - Exercise to reinforce concept [Ex. 36, Page 292]
- Today's Class
  - Routing
    - Introduction
    - Distance Vector
    - Link State

Due Dates

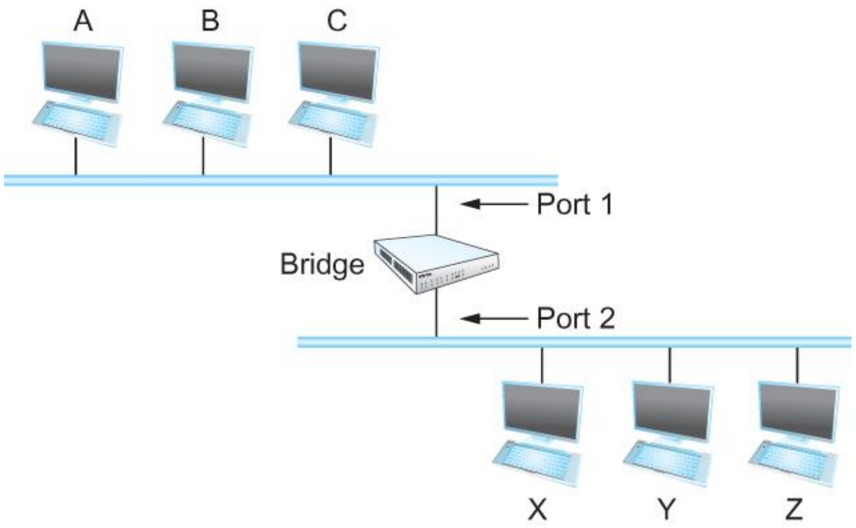
- Programming Assignment I (due March 29,11:59PM)
- ~~Term Paper Email (due February 5, 11:59PM)~~
- ~~Term Paper Proposal (due February 19, 11:59PM)~~
- **Midterm Examination (March 12)**
- Full term Paper (due May 6, 11:59PM)

Previous Class (Quick Refresher)

- Learning Bridges
- Distributed Spanning Tree Algorithm
  - Exercise to reinforce concept [Ex. 13, Page 288]
- Internet Protocol
- IP Datagram Format
- IP Fragmentation and Reassembly
- Exercise to reinforce concept [Ex. 36, Page 292]

Bridges and LAN Switches

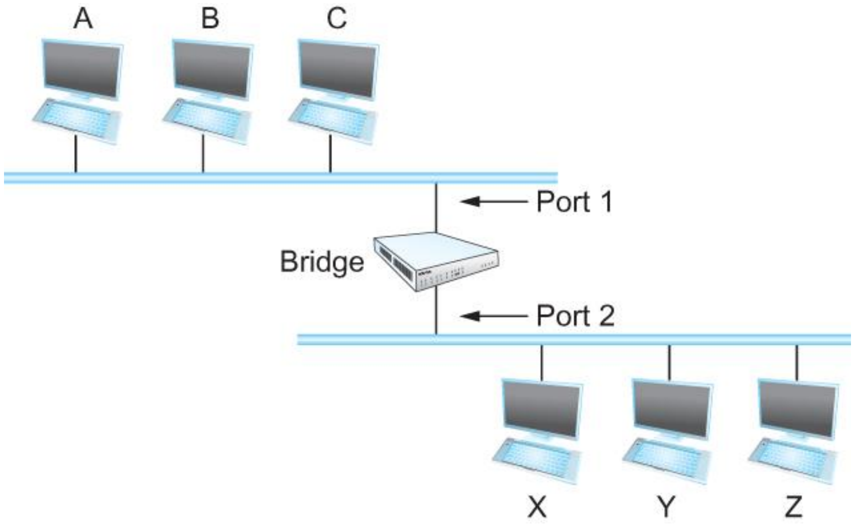
- Consider the following figure
  - When a frame from host A that is addressed to host B arrives on port 1, there is no need for the bridge to forward the frame out over port 2.



  - How does a bridge come to learn on which port the various hosts reside?

Bridges and LAN Switches

- Solution
  - Download a table into the bridge



Host	Port
A	1
B	1
C	1
X	2
Y	2
Z	2

  - Who does the download?
    - Human
      - Too much work for maintenance

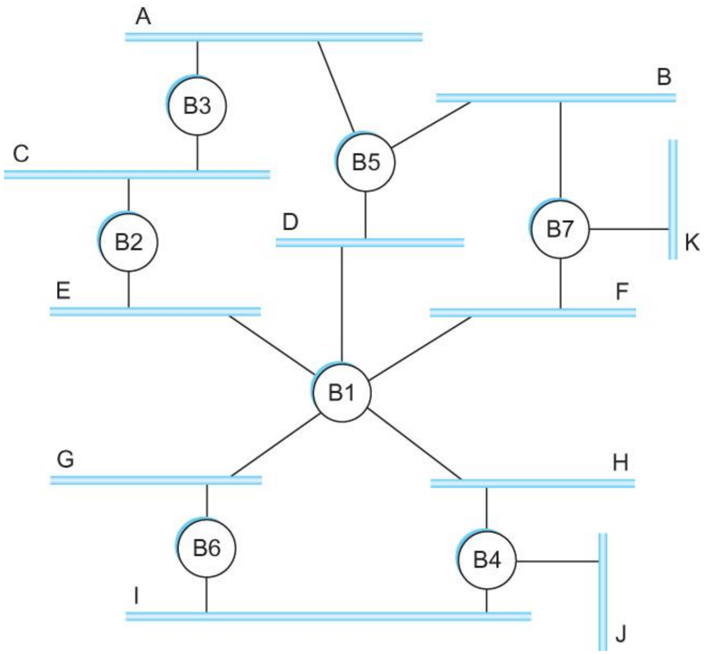
Bridges and LAN Switches

- Can the bridge learn this information by itself?
  - Yes

- How
  - Each bridge inspects the source address in all the frames it receives
  - Record the information at the bridge and build the table
  - When a bridge first boots, this table is empty
  - Entries are added over time
  - A timeout is associated with each entry
  - The bridge discards the entry after a specified period of time
    - To protect against the situation in which a host is moved from one network to another
- If the bridge receives a frame that is addressed to host not currently in the table
  - Forward the frame out on all other ports

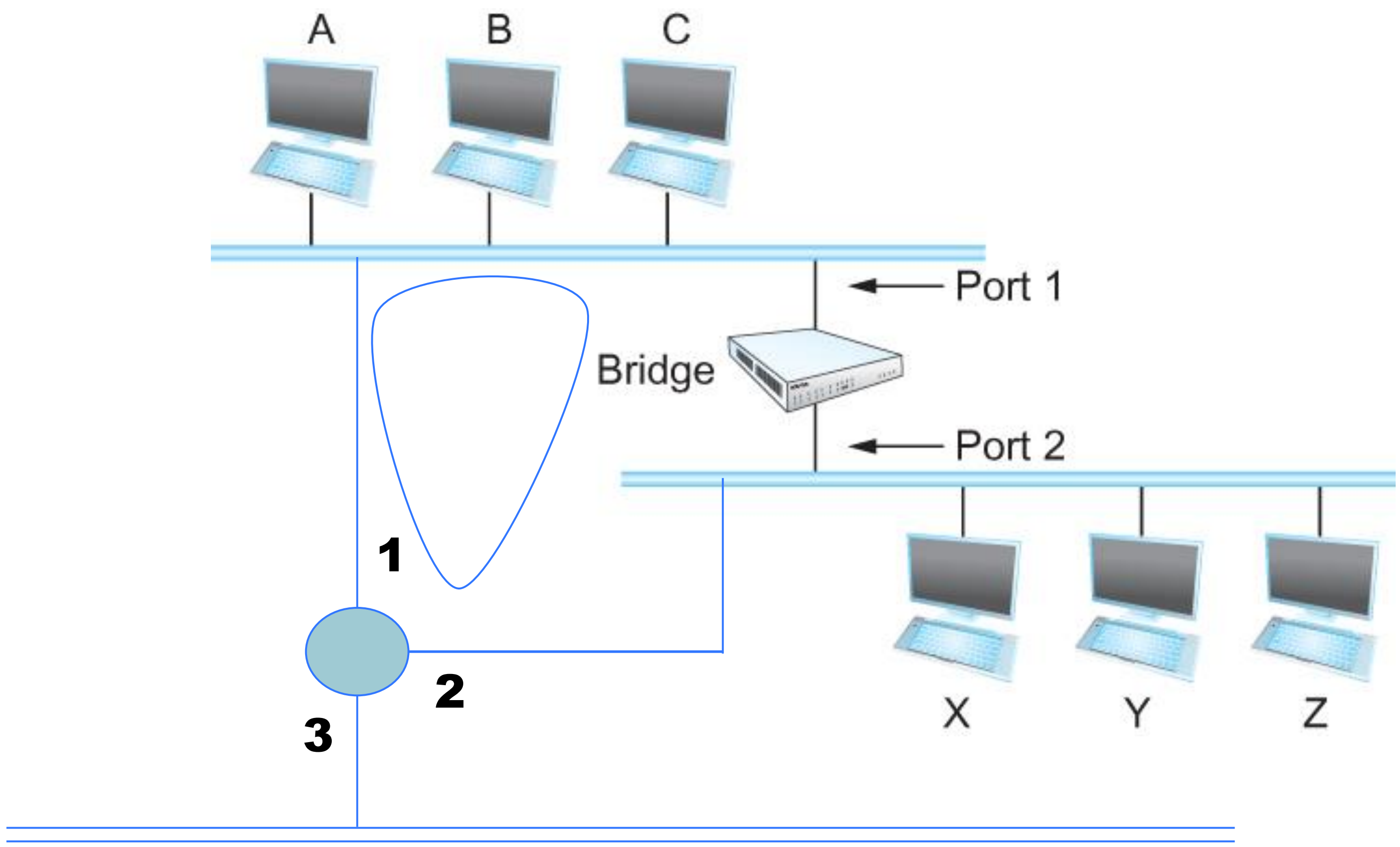
Bridges and LAN Switches

- Strategy works fine if the extended LAN does not have a loop in it
- Why?
  - Frames potentially loop through the extended LAN forever



  - Bridges B1, B4, and B6 form a loop

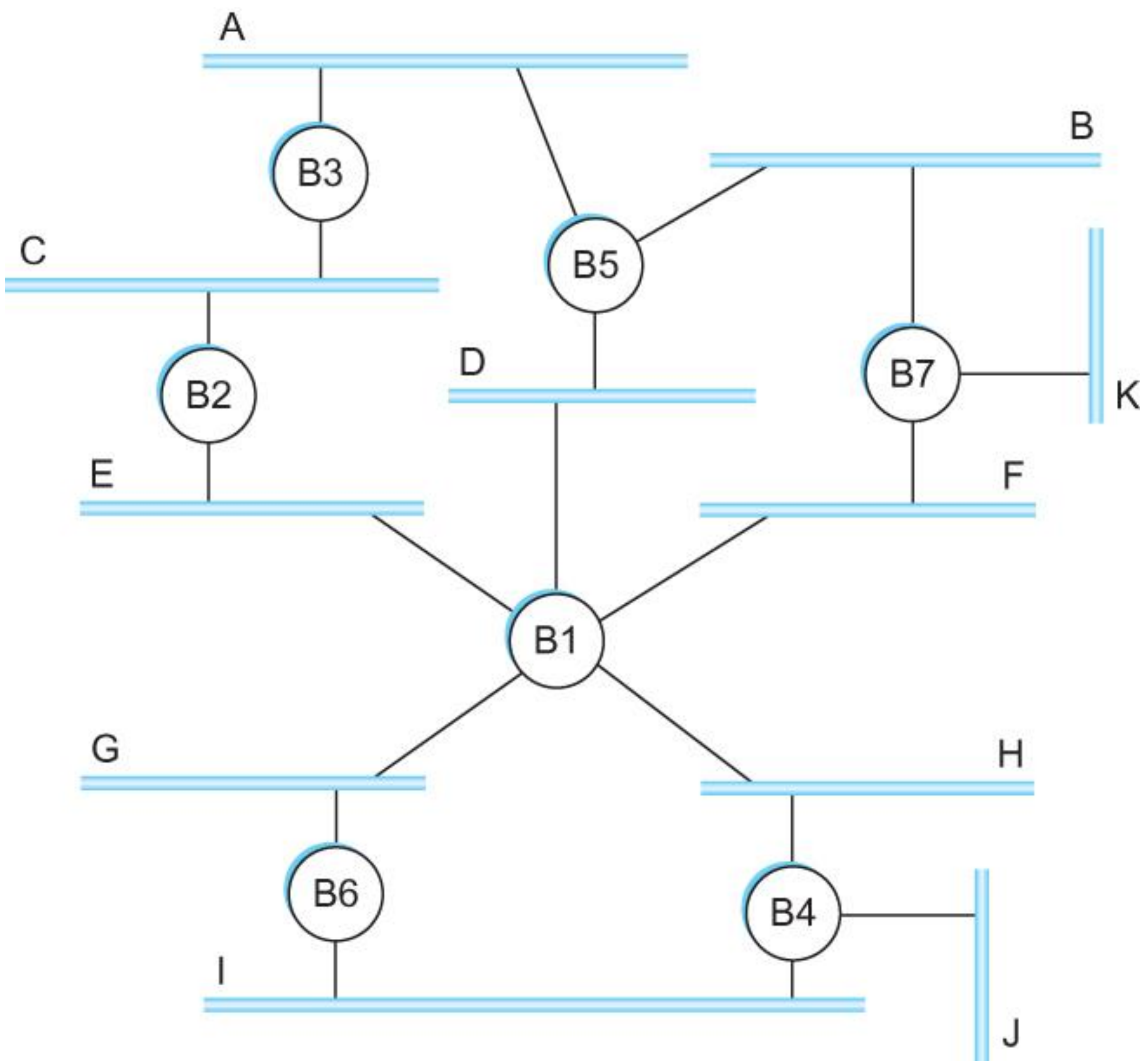
Bridges and LAN Switches



Bridges and LAN Switches

- How does an extended LAN come to have a loop in it?
  - Network is managed by more than one administrator
    - For example, it spans multiple departments in an organization
    - It is possible that no single person knows the entire configuration of the network
      - A bridge that closes a loop might be added without anyone knowing
  - Loops are built into the network to provide redundancy in case of failures
- Solution
  - Distributed Spanning Tree Algorithm

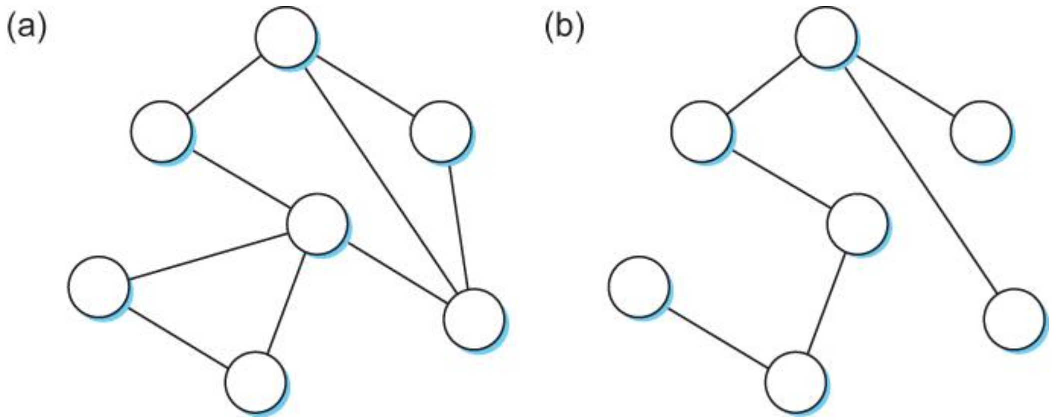
Spanning Tree Algorithm



Spanning Tree Algorithm

- Think of the extended LAN as being represented by a graph that possibly has loops (cycles)

- A spanning tree is a sub-graph of this graph that covers all the vertices but contains no cycles
  - Spanning tree keeps all the vertices of the original graph but throws out some of the edges



Example of (a) a cyclic graph; (b) a corresponding spanning tree.

Spanning Tree Algorithm

- Developed by Radia Perlman at Digital Equipment Corporation
  - A protocol used by a set of bridges to agree upon a spanning tree for a particular extended LAN
  - IEEE 802.1 specification for LAN bridges is based on this algorithm

  - Each bridge decides the ports over which it is and is not willing to forward frames
    - In a sense, it is by removing ports from the topology that the extended LAN is reduced to an acyclic tree
    - It is even possible that an entire bridge will not participate in forwarding frames

Spanning Tree Algorithm

- Algorithm is dynamic
  - The bridges are always prepared to reconfigure themselves into a new spanning tree if some bridges fail

- Main idea
  - Each bridge selects the ports over which they will forward the frames

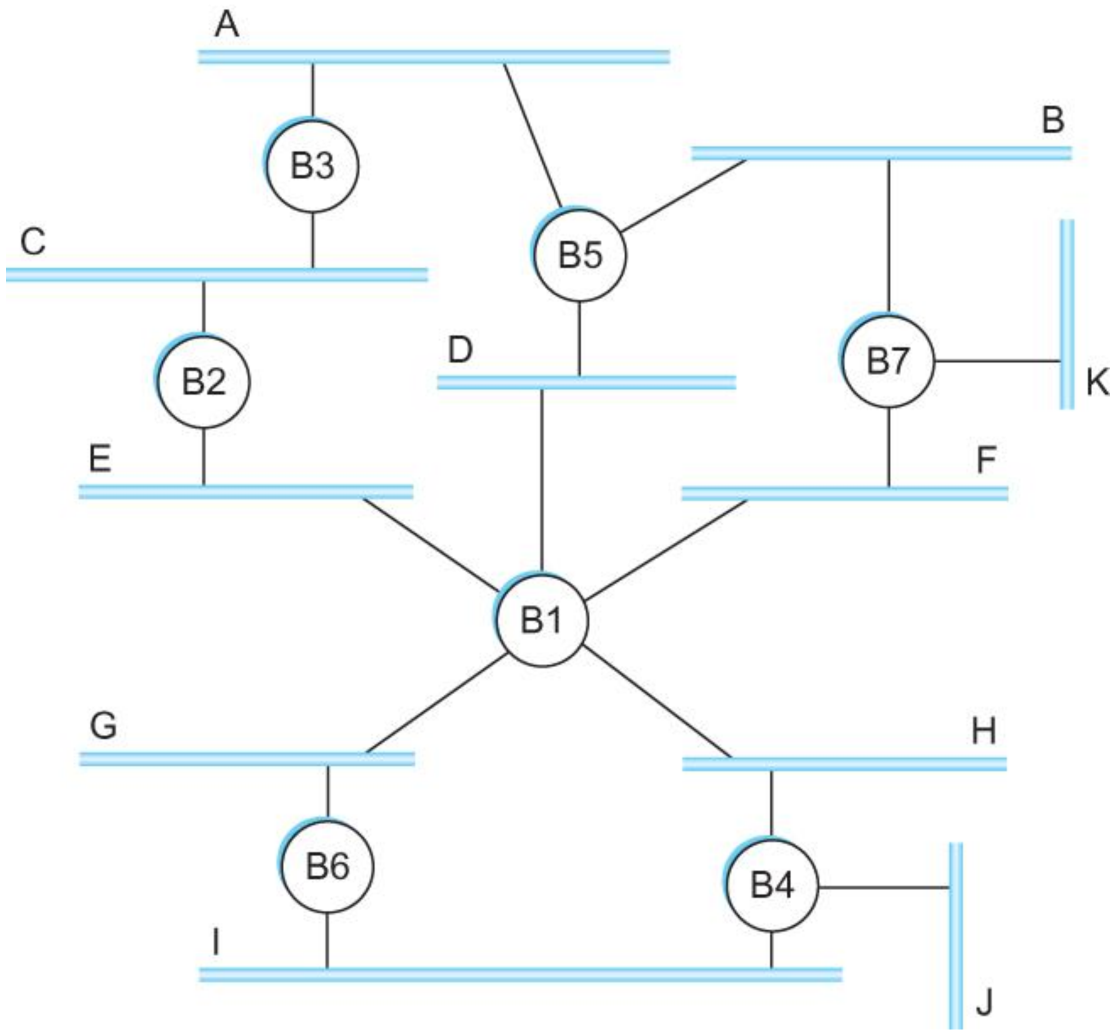
Spanning Tree Algorithm

- Algorithm selects ports as follows:
  - Each bridge has a unique identifier
    - B1, B2, B3,…and so on.
  - Elect the bridge with the smallest id as the root of the spanning tree
  - The root bridge always forwards frames out over all of its ports
  - Each bridge computes the shortest path to the root and notes which of its ports is on this path
    - This port is selected as the bridge's preferred path to the root
  - Finally, all the bridges connected to a given LAN elect a single *designated bridge* that will be responsible for forwarding frames toward the root bridge

Spanning Tree Algorithm

- Each LAN's designated bridge is the one that is closest to the root
- If two or more bridges are equally close to the root,
  - Then select bridge with the smallest id
- Each bridge is connected to more than one LAN
  - So it participates in the election of a designated bridge for each LAN it is connected to.
  - Each bridge decides if it is the designated bridge relative to each of its ports
  - The bridge forwards frames over those ports for which it is the designated bridge

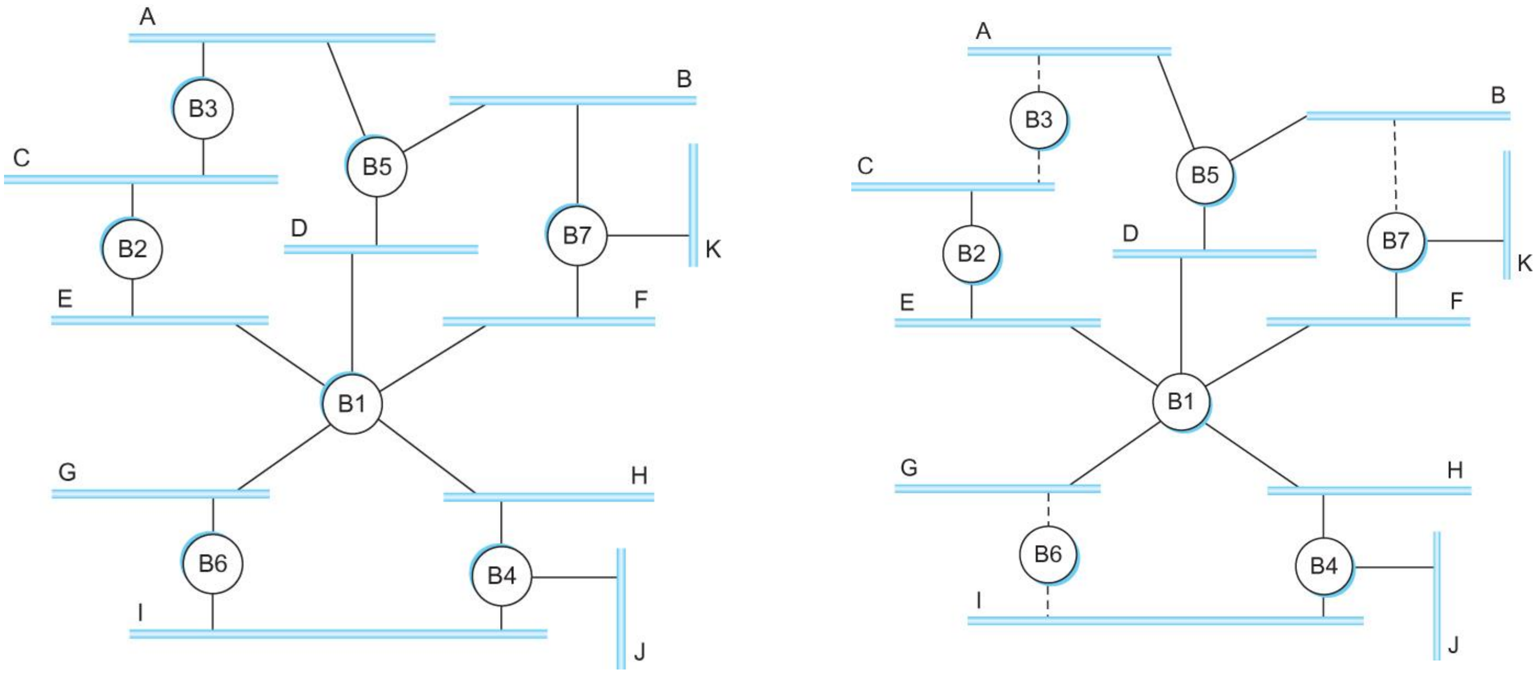
Spanning Tree Algorithm



- Elect the bridge with the smallest id as the root of the spanning tree
- Each LAN's designated bridge is the one that is closest to the root
- If two or more bridges are equally close to the root, then select bridge with the smallest id

Spanning Tree Algorithm

- B1 is the root bridge
- B3 and B5 are connected to LAN A, but B5 is the designated bridge
- B5 and B7 are connected to LAN B, but B5 is the designated bridge



Spanning Tree Algorithm

- Initially each bridge thinks it is the root, so it sends a configuration message on each of its ports identifying itself as the root and giving a distance to the root of 0
- Upon receiving a configuration message over a particular port, the bridge checks to see if the new message is *better* than the current best configuration message recorded for that port
- The new configuration is better than the currently recorded information if
  - It identifies a root with a smaller id or
  - It identifies a root with an equal id but with a shorter distance or
  - The root id and distance are equal, but the sending bridge has a smaller id

Spanning Tree Algorithm

- If the new message is better than the currently recorded one,
  - The bridge discards the old information and saves the new information
  - It first adds 1 to the distance-to-root field

- When a bridge receives a configuration message indicating that it is not the root bridge (that is, a message from a bridge with smaller id)
  - The bridge stops generating configuration messages on its own
  - Only forwards configuration messages from other bridges after 1 adding to the distance field

Spanning Tree Algorithm

- When a bridge receives a configuration message that indicates it is not the designated bridge for that port

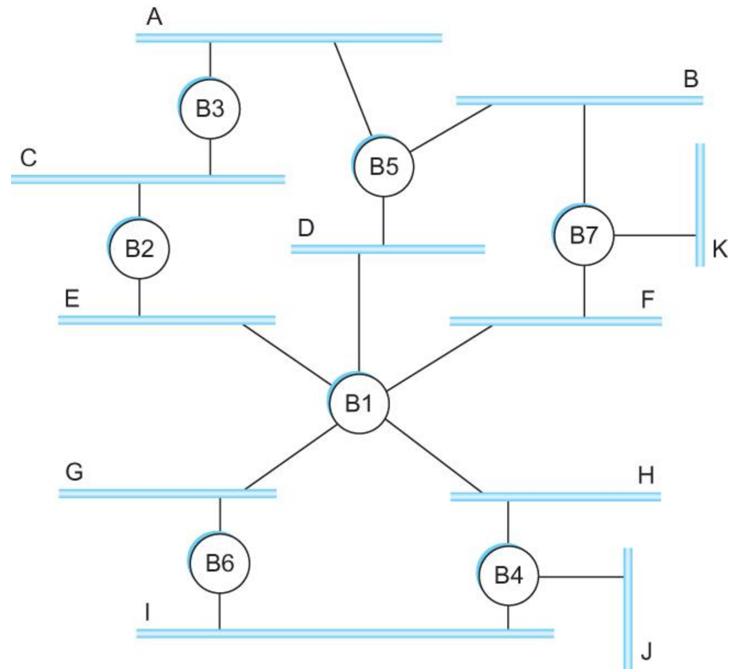
  => a message from a bridge that is closer to the root or equally far from the root but with a smaller id

    - The bridge stops sending configuration messages over that port

- When the system stabilizes,
  - Only the root bridge is still generating configuration messages.
  - Other bridges are forwarding these messages only over ports for which they are the designated bridge

Spanning Tree Algorithm

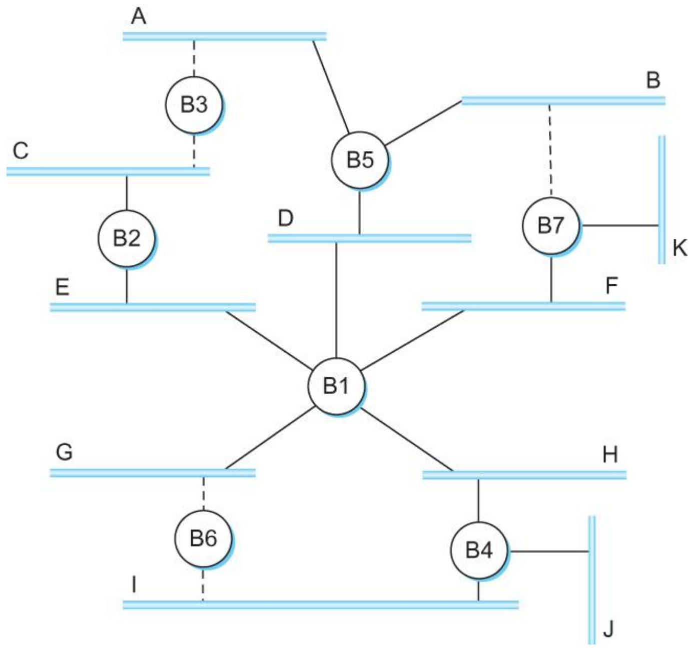
- Consider the situation when the power had just been restored to the building housing the following network



- All bridges would start off by claiming to be the root

Spanning Tree Algorithm

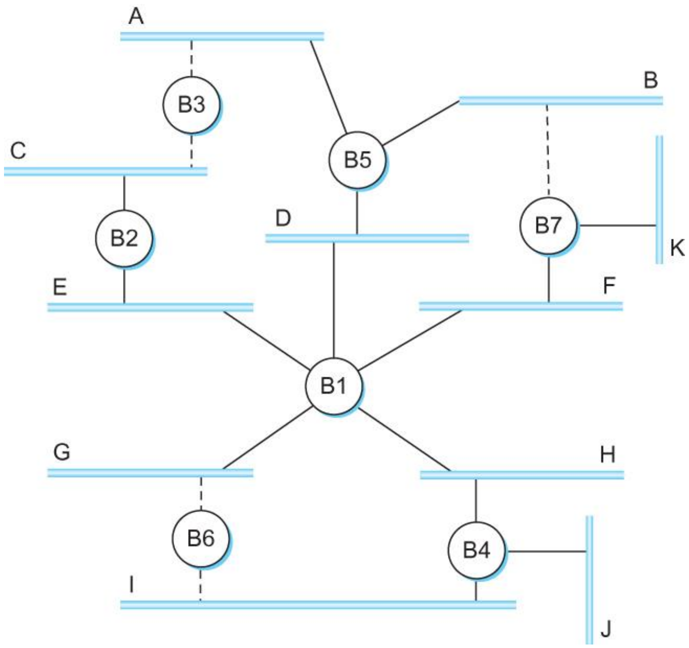
- Denote a configuration message from node X in which it claims to be distance d from the root node Y as (Y, d, X)



- Consider the activity at node B3

Spanning Tree Algorithm

- B3 receives (B2, 0, B2)
- Since 2 < 3, B3 accepts B2 as root
- B3 adds 1 to the distance advertised by B2 and sends (B2, 1, B3) to B5
- Meanwhile B2 accepts B1 as root because it has the lower id and it sends (B1, 1, B2) toward B3
- B5 accepts B1 as root and sends (B1, 1, B5) to B3
- B3 accepts B1 as root and it notes that both B2 and B5 are closer to the root than it is.
  - Thus B3 stops forwarding messages on both its interfaces
  - This leaves B3 with both ports not selected



Spanning Tree Algorithm

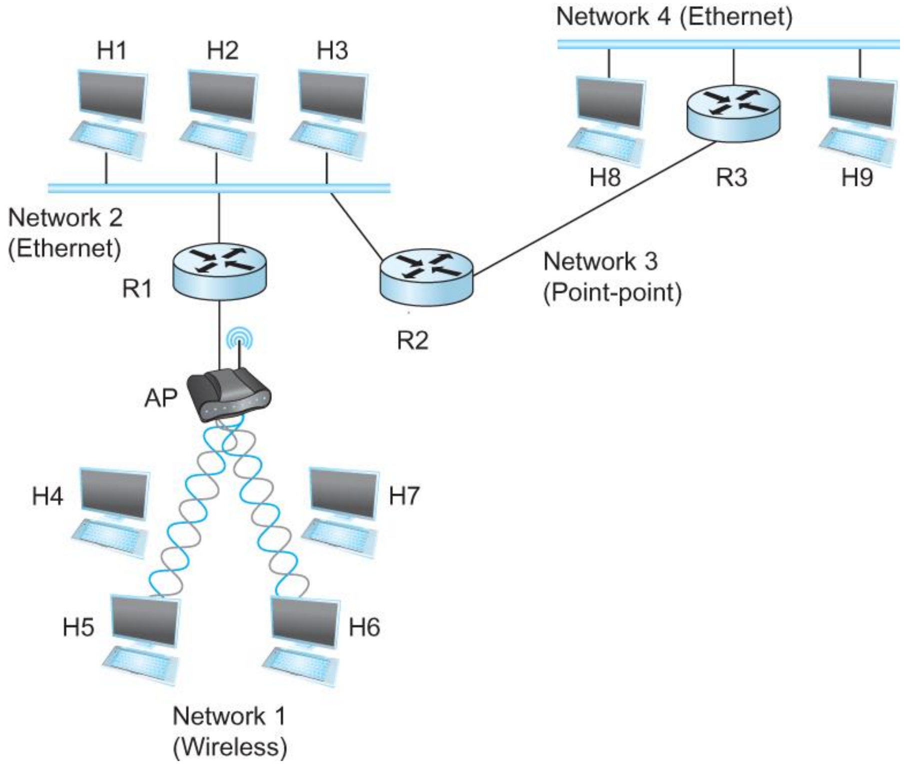
- Even after the system has stabilized, the root bridge continues to send configuration messages periodically
  - Other bridges continue to forward these messages
- When a bridge fails, the downstream bridges will not receive the configuration messages
- After waiting a specified period of time, they will once again claim to be the root and the algorithm starts again
- Note
  - Although the algorithm is able to reconfigure the spanning tree whenever a bridge fails, it is not able to forward frames over alternative paths for the sake of routing around a congested bridge

Spanning Tree Algorithm

- Limitation of Bridges
  - Do not scale
    - Spanning tree algorithm does not scale
    - Broadcast does not scale
  - Do not accommodate heterogeneity

Internetworking

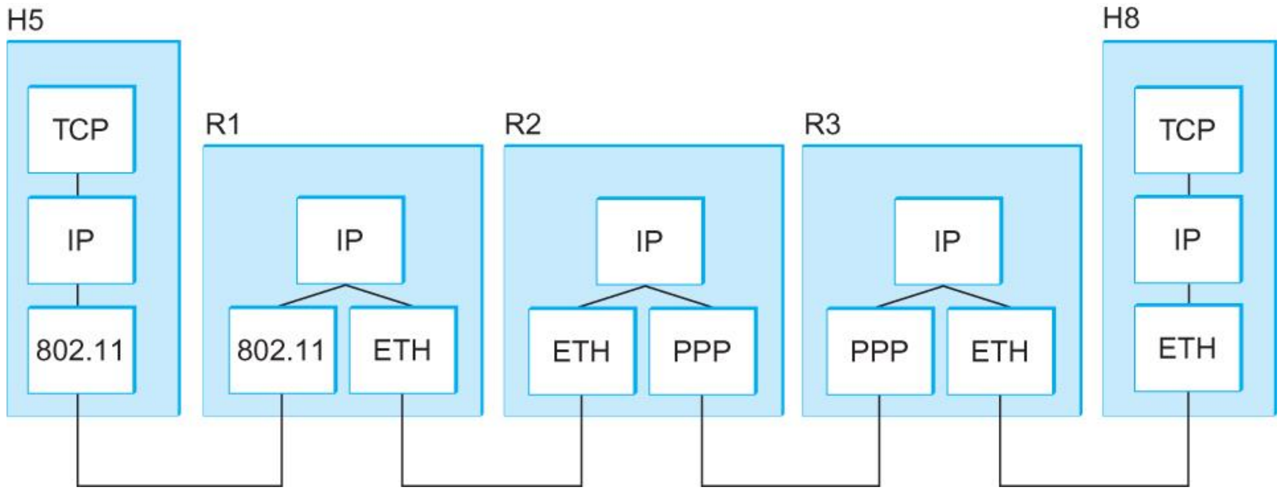
- What is internetwork
  - An arbitrary collection of networks interconnected to provide some sort of host-host to packet delivery service



A simple internetwork where H represents hosts and R represents routers

Internetworking

- What is IP
  - IP stands for Internet Protocol
  - Key tool used today to build scalable, heterogeneous internetworks
  - It runs on all the nodes in a collection of networks and defines the infrastructure that allows these nodes and networks to function as a single logical internetwork



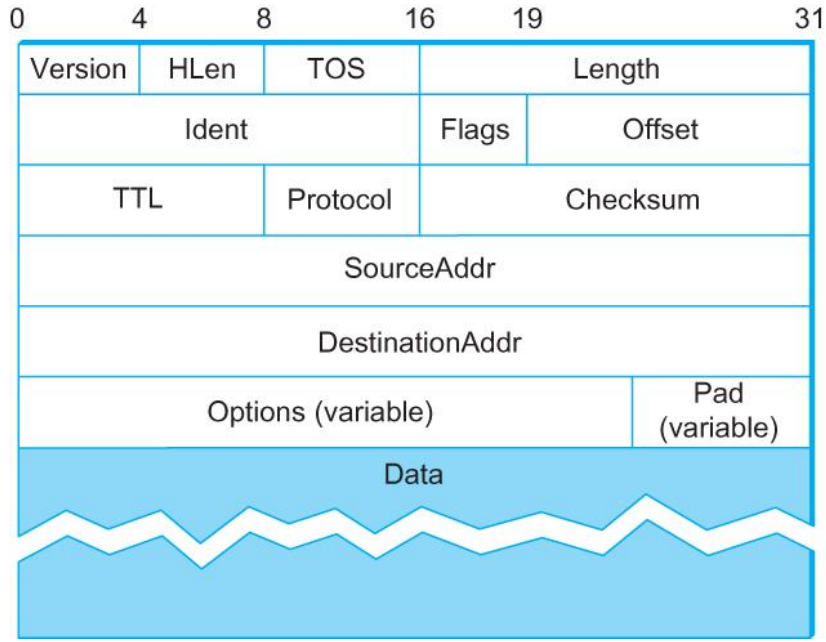
A simple internetwork showing the protocol layers

IP Service Model

- Packet Delivery Model
  - Connectionless model for data delivery
  - Best-effort delivery (unreliable service)
    - packets are lost
    - packets are delivered out of order
    - duplicate copies of a packet are delivered
    - packets can be delayed for a long time
- Global Addressing Scheme
  - Provides a way to identify all hosts in the network

Packet Format

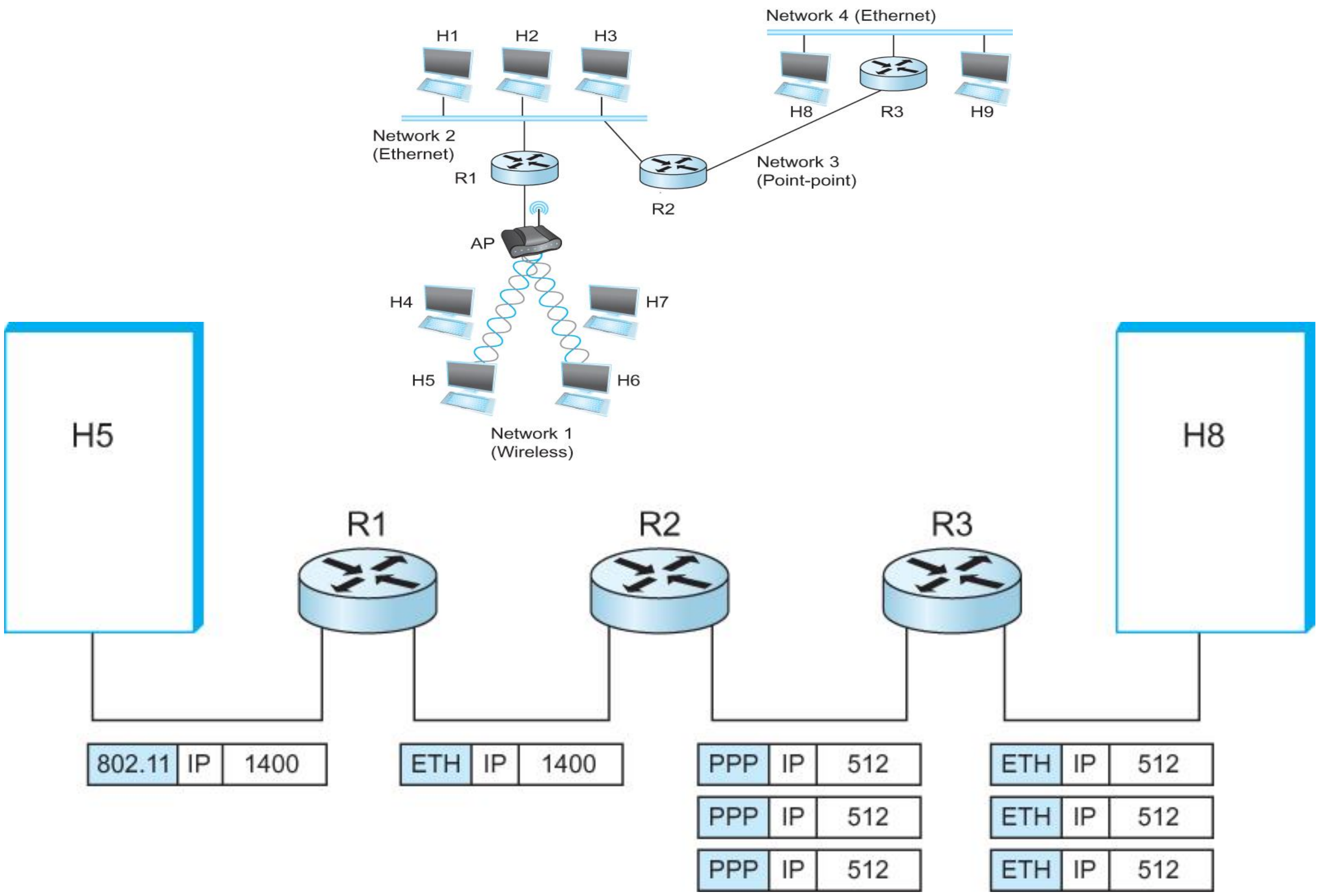
- Version (4): currently 6
- Hlen (4): number of 32-bit words in header
- TOS (8): type of service (not widely used)
- Length (16): number of bytes in this datagram
- Ident (16): used by fragmentation
- Flags/Offset (16): used by fragmentation
- TTL (8): number of hops this datagram has traveled
- Protocol (8): demux key (TCP=6, UDP=17)
- Checksum (16): of the header only
- DestAddr & SrcAddr (32)



IP Fragmentation and Reassembly

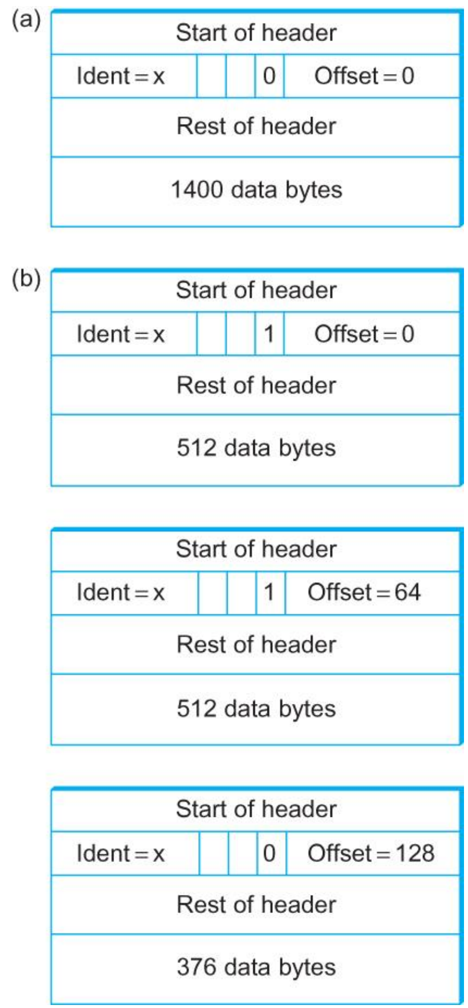
- Each network has some MTU (Maximum Transmission Unit)
  - Ethernet (1500 bytes), FDDI (4500 bytes)
- Strategy
  - Fragmentation occurs in a router when it receives a datagram that it wants to forward over a network which has (MTU < datagram)
  - Reassembly is done at the receiving host
  - All the fragments carry the same identifier in the *Ident* field
  - Fragments are self-contained datagrams
  - IP does not recover from missing fragments

IP Fragmentation and Reassembly



IP datagrams traversing the sequence of physical networks

IP Fragmentation and Reassembly



Header fields used in IP fragmentation. (a) Unfragmented packet; (b) fragmented packets.

Today’s Topic

Exercise to reinforce concept
[Ex. 39, Page 293]

Routing

Forwarding versus Routing

- Forwarding:
  - to select an output port based on destination address and routing table
- Routing:
  - process by which routing table is built

Routing

- Forwarding table VS Routing table
  - Forwarding table
    - Used when a packet is being forwarded and so must contain enough information to accomplish the forwarding function
    - A row in the forwarding table contains the mapping from a network number to an outgoing interface and some MAC information, such as Ethernet Address of the next hop
  - Routing table
    - Built by the routing algorithm as a precursor to build the forwarding table
    - Generally contains mapping from network numbers to next hops

Routing

(a)

Prefix/Length	Next Hop
18/8	171.69.245.10

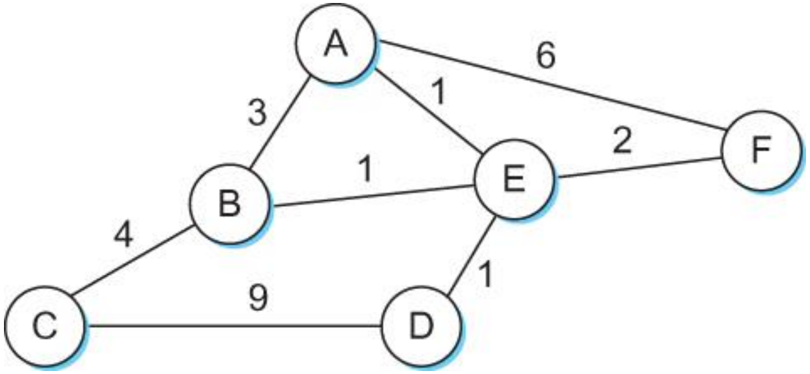
(b)

Prefix/Length	Interface	MAC Address
18/8	if0	8:0:2b:e4:b:1:2

Example rows from (a) routing and (b) forwarding tables

Routing

- Network as a Graph



- The basic problem of routing is to find the lowest-cost path between any two nodes
  - Where the cost of a path equals the sum of the costs of all the edges that make up the path

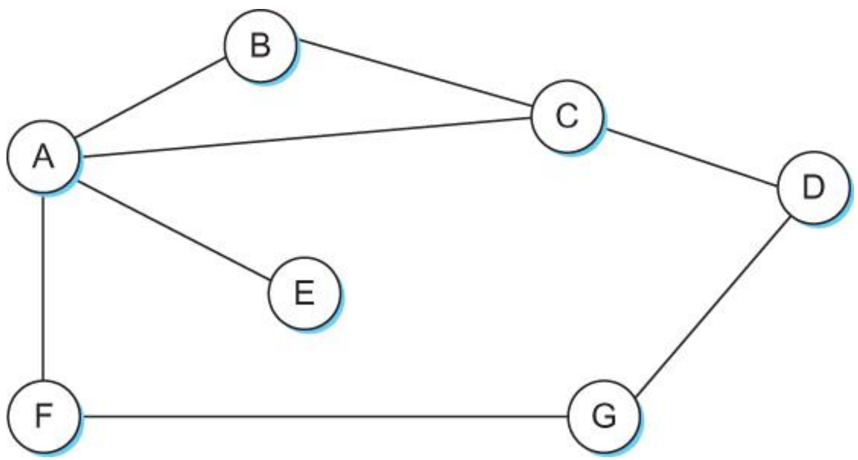
Routing

- For a simple network, we can calculate all shortest paths and load them into some nonvolatile storage on each node.
- Such a static approach has several shortcomings
  - It does not deal with node or link failures
  - It does not consider the addition of new nodes or links
  - It implies that edge costs cannot change

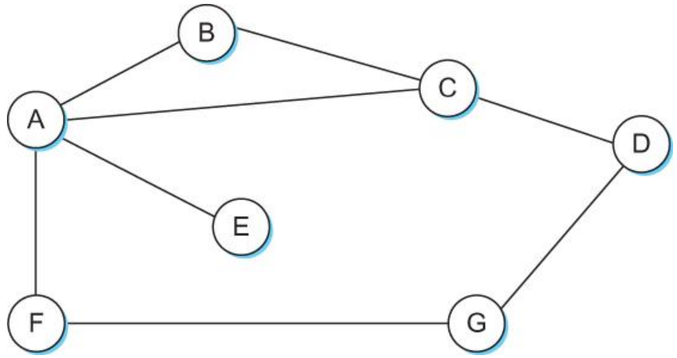
- What is the solution?
  - Need a distributed and dynamic protocol
  - Two main classes of protocols
    - Distance Vector
    - Link State

Distance Vector

- Each node constructs a one dimensional array (a vector) containing the “distances” (costs) to all other nodes and distributes that vector to its immediate neighbors
- Starting assumption is that each node knows the cost of the link to each of its directly connected neighbors



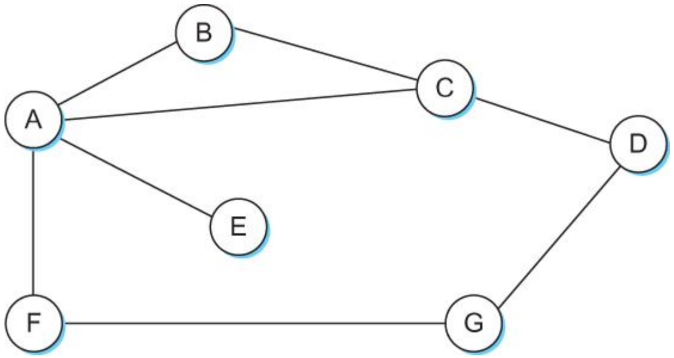
Distance Vector



Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Initial distances stored at each node (global view)

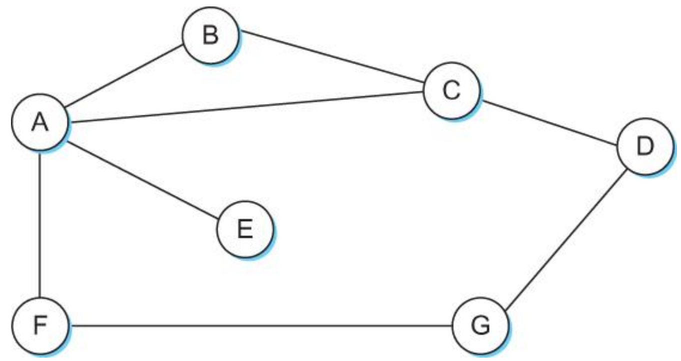
Distance Vector



Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

Initial routing table at node A

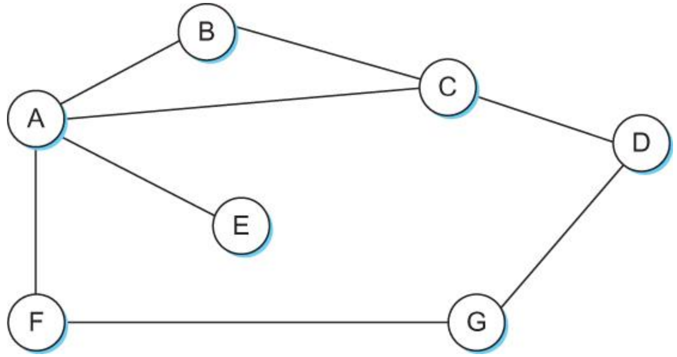
Distance Vector



Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Final routing table at node A

Distance Vector



Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

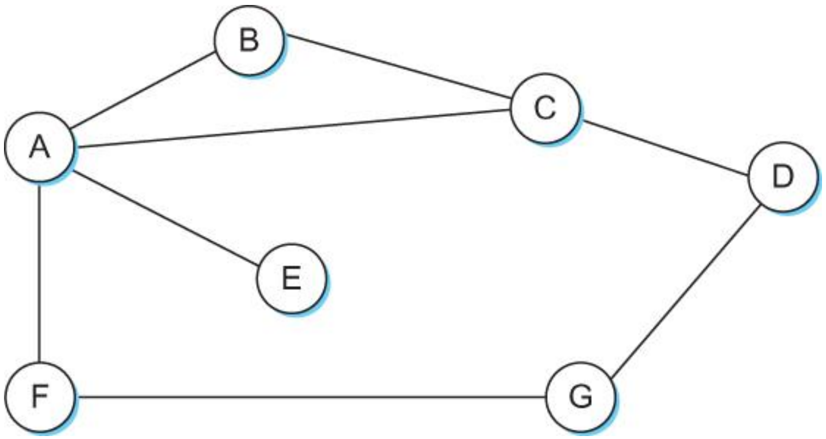
Final distances stored at each node (global view)

Distance Vector

- The distance vector routing algorithm is sometimes called as Bellman-Ford algorithm
- Every T seconds each router sends its table to its neighbor each router then updates its table based on the new information
- Problems include fast response to good news and slow response to bad news. Also too many messages to update

Distance Vector

- When a node detects a link failure
  - F detects that link to G has failed
  - F sets distance to G to infinity and sends update to A
  - A sets distance to G to infinity since it uses F to reach G
  - A receives periodic update from C with 2-hop path to G
  - A sets distance to G to 3 and sends update to F
  - F decides it can reach G in 4 hops via A



Distance Vector

- Slightly different circumstances can prevent the network from stabilizing
  - Suppose the link from A to E goes down
  - In the next round of updates, A advertises a distance of infinity to E, but B and C advertise a distance of 2 to E
  - Depending on the exact timing of events, the following might happen
    - Node B, upon hearing that E can be reached in 2 hops from C, concludes that it can reach E in 3 hops and advertises this to A
    - Node A concludes that it can reach E in 4 hops and advertises this to C
    - Node C concludes that it can reach E in 5 hops; and so on.
    - This cycle stops only when the distances reach some number that is large enough to be considered infinite
      - **Count-to-infinity problem**

Count-to-infinity Problem

- Use some relatively small number as an approximation of infinity
- For example, the maximum number of hops to get across a certain network is never going to be more than 16

- One technique to improve the time to stabilize routing is called *split horizon*
  - When a node sends a routing update to its neighbors, it does not send those routes it learned from each neighbor back to that neighbor
  - For example, if B has the route (E, 2, A) in its table, then it knows it must have learned this route from A, and so whenever B sends a routing update to A, it does not include the route (E, 2) in that update

Count-to-infinity Problem

- In a stronger version of split horizon, called *split horizon with poison reverse*
  - B actually sends that back route to A, but it puts negative information in the route to ensure that A will not eventually use B to get to E
  - For example, B sends the route (E, ∞) to A

EXERCISE 46, PAGE 294

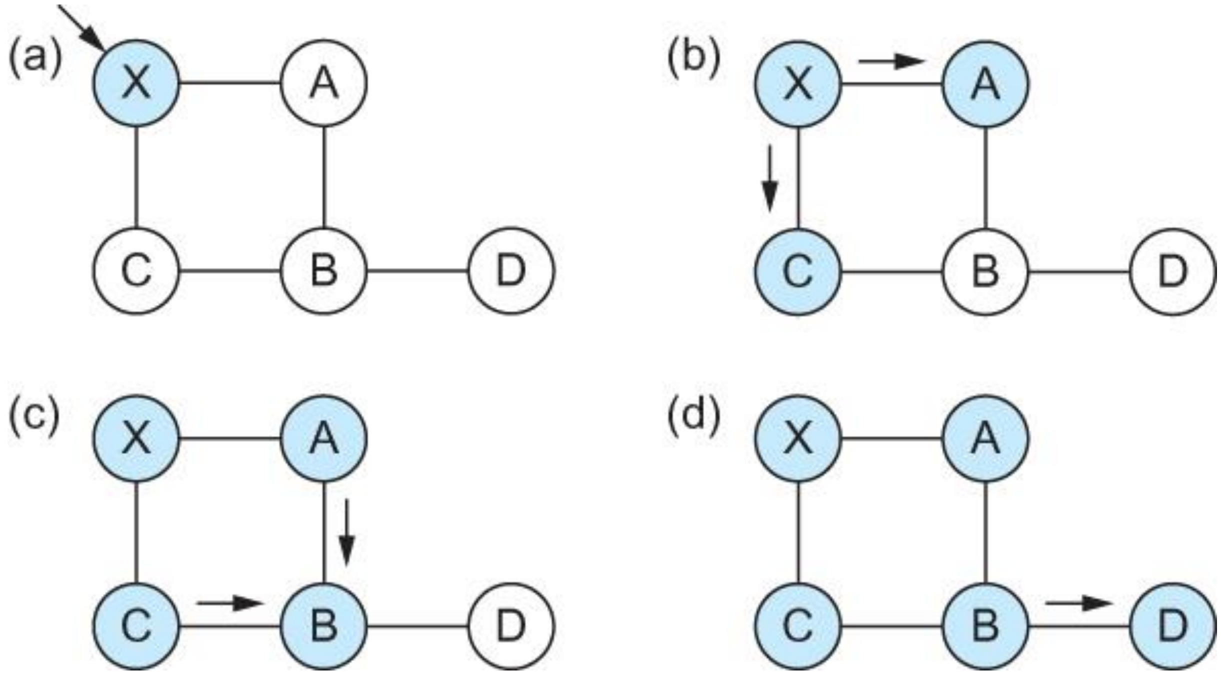
Link State Routing

Strategy: Send to all nodes (not just neighbors) information about directly connected links (not entire routing table).

- Link State Packet (LSP)
  - id of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet
- Reliable Flooding
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - generate new LSP periodically; increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP; discard when TTL=0

Link State

Reliable Flooding



Flooding of link-state packets. (a) LSP arrives at node X; (b) X floods LSP to A and C; (c) A and C flood LSP to B (but not X); (d) flooding is complete

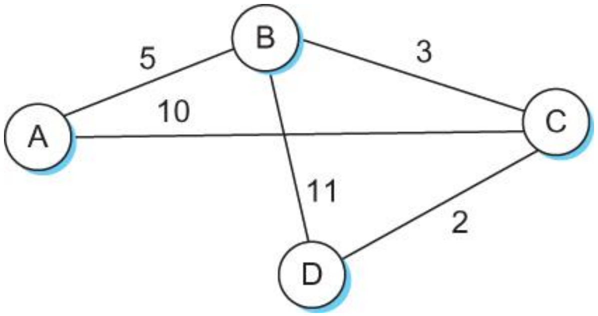
Shortest Path Routing

- In practice, each switch computes its routing table directly from the LSP's it has collected using a realization of Dijkstra's algorithm called the *forward search algorithm*
- Specifically each switch maintains two lists, known as **Tentative** and **Confirmed**
- Each of these lists contains a set of entries of the form (Destination, Cost, NextHop)

Shortest Path Routing

- The algorithm
  - Initialize the **Confirmed** list with an entry for myself; this entry has a cost of 0
  - For the node just added to the **Confirmed** list in the previous step, call it node **Next**, select its LSP
  - For each neighbor (Neighbor) of **Next**, calculate the cost (Cost) to reach this Neighbor as the sum of the cost from myself to Next and from Next to Neighbor
    - If Neighbor is currently on neither the **Confirmed** nor the **Tentative** list, then add (Neighbor, Cost, Nexthop) to the **Tentative** list, where Nexthop is the direction I go to reach Next
    - If Neighbor is currently on the **Tentative** list, and the Cost is less than the currently listed cost for the Neighbor, then replace the current entry with (Neighbor, Cost, Nexthop) where Nexthop is the direction I go to reach Next
  - If the **Tentative** list is empty, stop. Otherwise, pick the entry from the **Tentative** list with the lowest cost, move it to the **Confirmed** list, and return to Step 2.

Shortest Path Routing



Step	Confirmed	Tentative	Comments
1	(D,0,–)		Since D is the only new member of the confirmed list, look at its LSP.
2	(D,0,–)	(B,11,B) (C,2,C)	D's LSP says we can reach B through B at cost 11, which is better than anything else on either list, so put it on Tentative list; same for C.
3	(D,0,–) (C,2,C)	(B,11,B)	Put lowest-cost member of Tentative (C) onto Confirmed list. Next, examine LSP of newly confirmed member (C).
4	(D,0,–) (C,2,C)	(B,5,C) (A,12,C)	Cost to reach B through C is 5, so replace (B,11,B). C's LSP tells us that we can reach A at cost 12.
5	(D,0,–) (C,2,C) (B,5,C)	(A,12,C)	Move lowest-cost member of Tentative (B) to Confirmed, then look at its LSP.
6	(D,0,–) (C,2,C) (B,5,C)	(A,10,C)	Since we can reach A at cost 5 through B, replace the Tentative entry.
7	(D,0,–) (C,2,C) (B,5,C) (A,10,C)		Move lowest-cost member of Tentative (A) to Confirmed, and we are all done.

Shortest Path Routing

- Dijkstra's Algorithm - Assume non-negative link weights
  - N: set of nodes in the graph
  - l((i, j): the non-negative cost associated with the edge between nodes i, j $\in$N and l(i, j) = $\propto$ if no edge connects i and j
  - Let s $\in$N be the starting node which executes the algorithm to find shortest paths to all other nodes in N
  - Two variables used by the algorithm
    - M: set of nodes incorporated so far by the algorithm
    - C(n) : the cost of the path from s to each node n
    - The algorithm

```
M = {s}
For each n in N - {s}
    C(n) = l(s, n)
while ( N ≠ M)
  M = M ∪ {w} such that C(w) is the minimum
                                  for all w in (N-M)
  For each n in (N-M)
          C(n) = MIN (C(n), C(w) + l(w, n))
```

EXERCISE 63, PAGE 300