

Scuola Superiore Sant'Anna di Studi Universitari e di Perfezionamento



The Migration Toward the Optical Internet

Lesson 2

Luca Valcarenghi



TCP/IP Reference Model: Internet/Network Layer

- Internet/Network layer task
 - ROUTING packets from source to destination
- In TCP/IP reference model this task is accomplished by Internet Protocol (IP)
- All TCP, UDP, ICMP, and IGMP data gets transmitted as IP datagrams
- IP provides an UNRELIABLE, CONNECTIONLESS DATAGRAM delivery



Internet Protocol (IP) (1)

- UNRELIABLE
 - No guarantees that an IP datagram successfully gets to its destination
 - Best effort service
 - IP error (e.g., router down) handling algorithm:
 - Datagram thrown away
 - ICMP message sent back to the source
 - Any required reliability must be provided by upper layers (e.g., TCP)



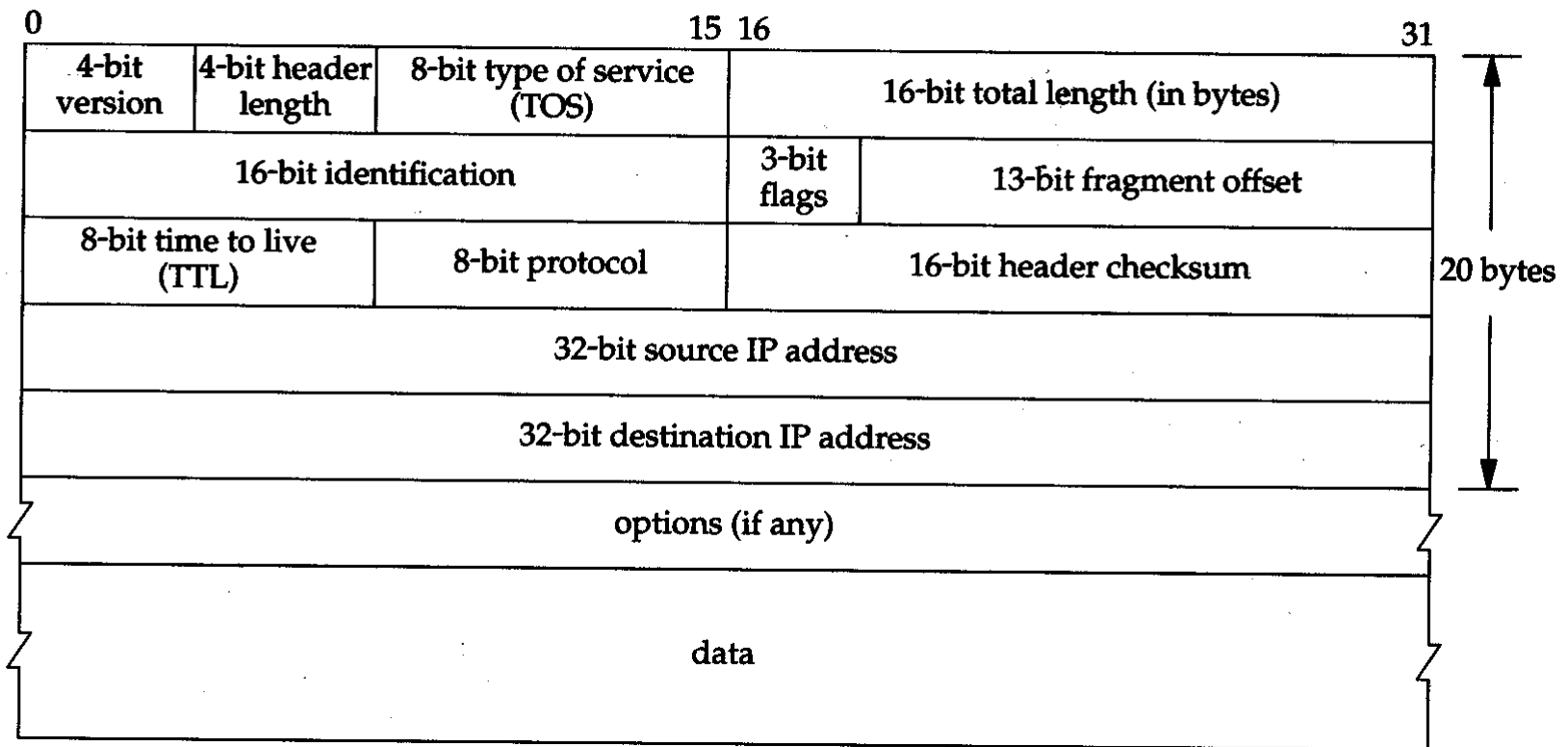
Internet Protocol (IP) (2)

- CONNECTIONLESS
 - IP does not maintain any state information about successive datagrams
 - Each datagram handled independently from all other datagrams
 - IP datagrams can get delivered out of order
- DATAGRAM
 - Unit of information of the Internet/Network layer
 - Usually represented in 32-bit words and identified from 0 to 31
 - The 4 bytes in the 32-bit value are transmitted in order: bit 0-7 first, then 8-15, etc. ⇒BIG ENDIAN byte ordering
 - Current version is no. 4 ⇒ IPv4
 - Recently proposed version no. 6 ⇒ IPv6
 - IPv6 includes QoS features



IP Datagram Format (1)

IP Header





IP Datagram Format (2)

- **VERSION:** IP protocol version
- **HEADER LENGTH:** number of 32-bit words in the header
 - header max length: 60 bytes ($15 \times 4 = 60$)
 - problem for explicit routing and record route
- **TYPE OF SERVICE (TOS)**
 - 3-bit precedence field (ignored)
 - 4 TOS bits (not supported by old TCP/IP implementations)
 - 1 unused bit (must be 0)



IP Datagram Format (3)

- **TOTAL LENGTH:** total length of the IP datagram in bytes
 - maximum size of an IP datagram 65535 bytes
- **IDENTIFICATION:** uniquely identifies each datagram sent by a host
 - normally increments by 1 each time a datagram is sent
- **FLAG:** one bit as the “more fragments” bit. This bit turned on for each fragment comprising a datagram except for the final fragment
- **FRAGMENT OFFSET:** offset in 8-byte units of this fragment from the beginning of the original datagram



IP Datagram Format (4)

- **TIME TO LIVE (TTL):** set an upper limit on the number of routers through which a datagram can pass
 - usually initialized by the sender to some value (32 or 64)
 - decremented by one by every router handling the datagram
- **PROTOCOL:** identifies which protocol gave the data for IP to send



IP Datagram Format (5)

- **HEADER CHECKSUM:** calculated over the IP header only
 - checksum calculated as the 16-bit ones' complement sum of the header (the entire header considered as a sequence of 16-bit words)
 - sender sets the value of the checksum
 - receiver calculates checksum as the sender and compares it with the 16-bit header checksum in the datagram
 - if checksum error \Rightarrow IP discard the received datagram
 - no error message is generated
- **SOURCE IP ADDRESS:** IP address of the sender
- **DESTINATION IP ADDRESS:** IP address of the receiver
 - never changed except when **EXPLICIT ROUTING** is used
- **OPTIONS:** variable-length lists of optional information for the datagram
 - Examples: security and handling restrictions, record route, strict source routing, etc.



Routing

- Routing algorithm functions
 - Selection of ROUTES for different source-destination (s,d) pairs
 - DELIVERY of message to correct destination once routes are selected
 - conceptually straightforward, based on variety of protocols and data structures (known as routing tables)



Routing Types

- **CENTRALIZED**
 - all route choices made at a central NODE
- **DISTRIBUTED**
 - computation of routes shared among network nodes with information exchanged between them as necessary
- **STATIC**
 - the route (path) used by the session of each (s,d) pair is fixed regardless of traffic conditions
 - it can only change in response to a link or node failure
- **DYNAMIC (ADAPTIVE)**
 - the routes (paths) used to route new traffic between sources and destinations change occasionally in response to congestion



IP Routing

- Host
 - Static Routing
 - IF destination directly connected to the host (e.g., point-to-point link) or on a shared network (e.g., Ethernet or Token Ring) IP datagram sent directly to the destination
 - OTHERWISE datagram sent to a default router
 - router delivers the datagram to its destination
- ROUTER
 - Static routing
 - Dynamic routing (RIP, OSPF)



Routing Table

- Example: `/>netstat -rn`
 - Flags
 - U: route is up
 - G: route is to a gateway
 - H: route is to a host (destination is a complete host address)
 - D: Route was created by redirect (IGMP)
 - M: route was modified by redirect (IGMP)

Destination Address	Next Hop Address	FLAGS	REFCNT	USE	Interface
140.252.13.65	140.252.13.25	UGH	0	0	emd0
127.0.0.1	127.0.0.1	UH	1	0	lo0
default	140.252.13.33	UG	0	0	emd0
140.252.13.32	140.252.13.34	U	4	25043	emd0



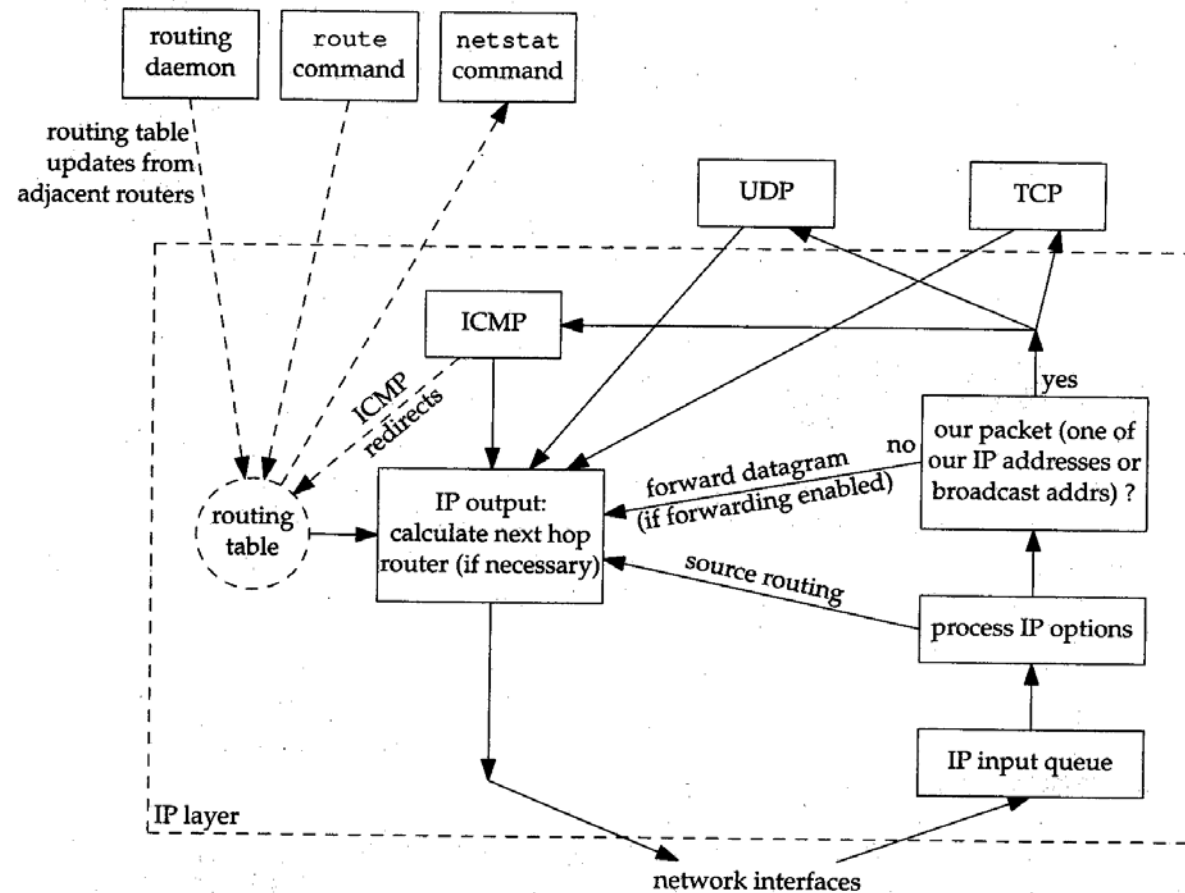
Dynamic Routing Protocols

- Used when the following conditions are NOT met
 - Network is small
 - Single point-to-point connection to other networks
 - No redundant routes (e.g., backup route available if primary route fails)
- Dynamic routing occurs when routers talk to adjacent routers (routers directly connected), informing each other of what networks each router is currently connected to
- Dynamic routing protocols used by routers to communicate with each other
 - Routing Information Protocol (RIP)
 - Open Shortest Path First (OSPF)



Dynamic Routing Protocols (2)

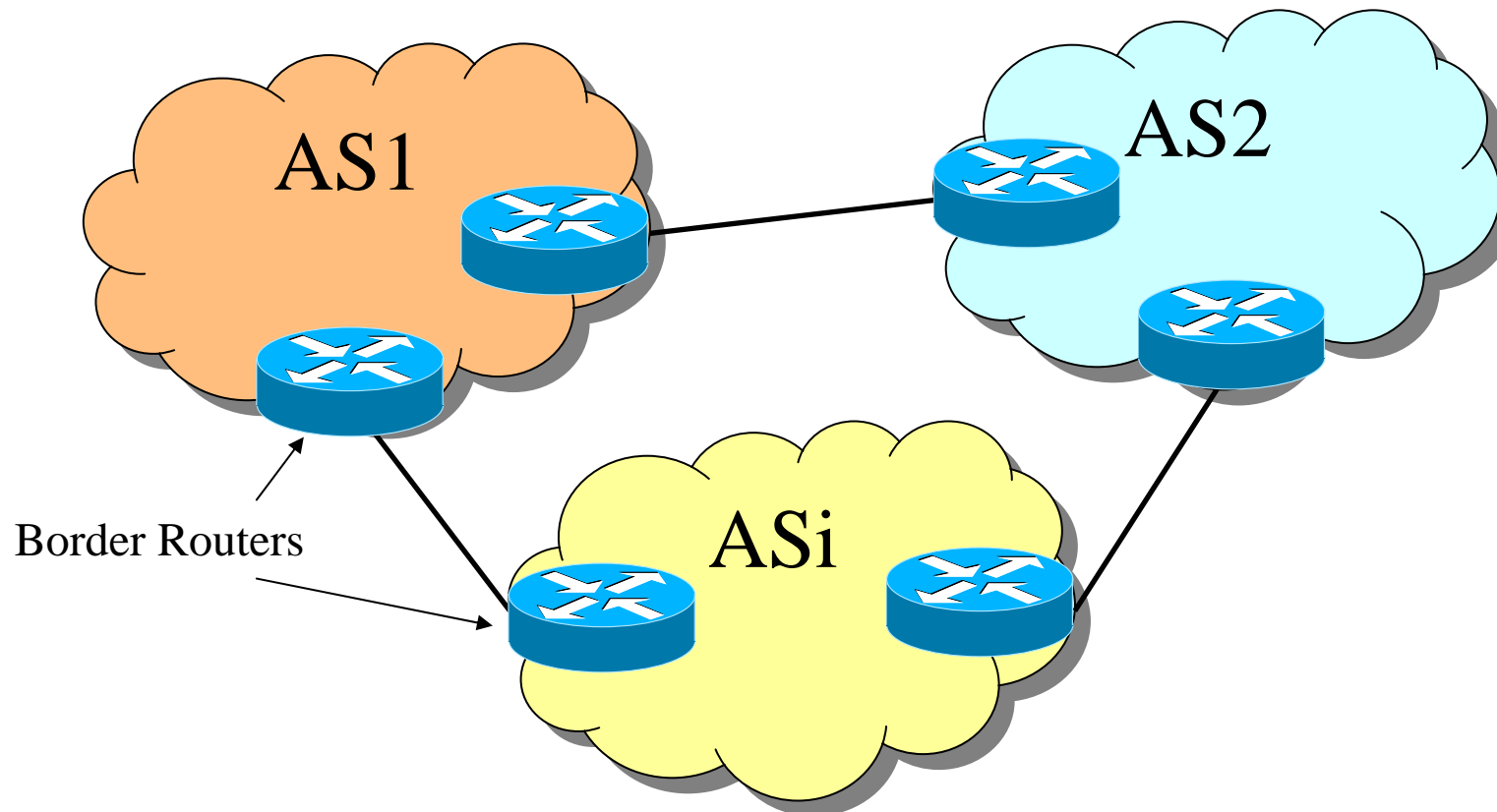
- Process running on a router usually called **ROUTING DAEMON**
- Routing daemon updates the kernel's routing table





Dynamic Routing Protocols (3)

- Internet organized in Autonomous Systems (AS)
 - Each AS is usually administered by a single entity (e.g., AS can be a corporation or campus network, NSFNET)





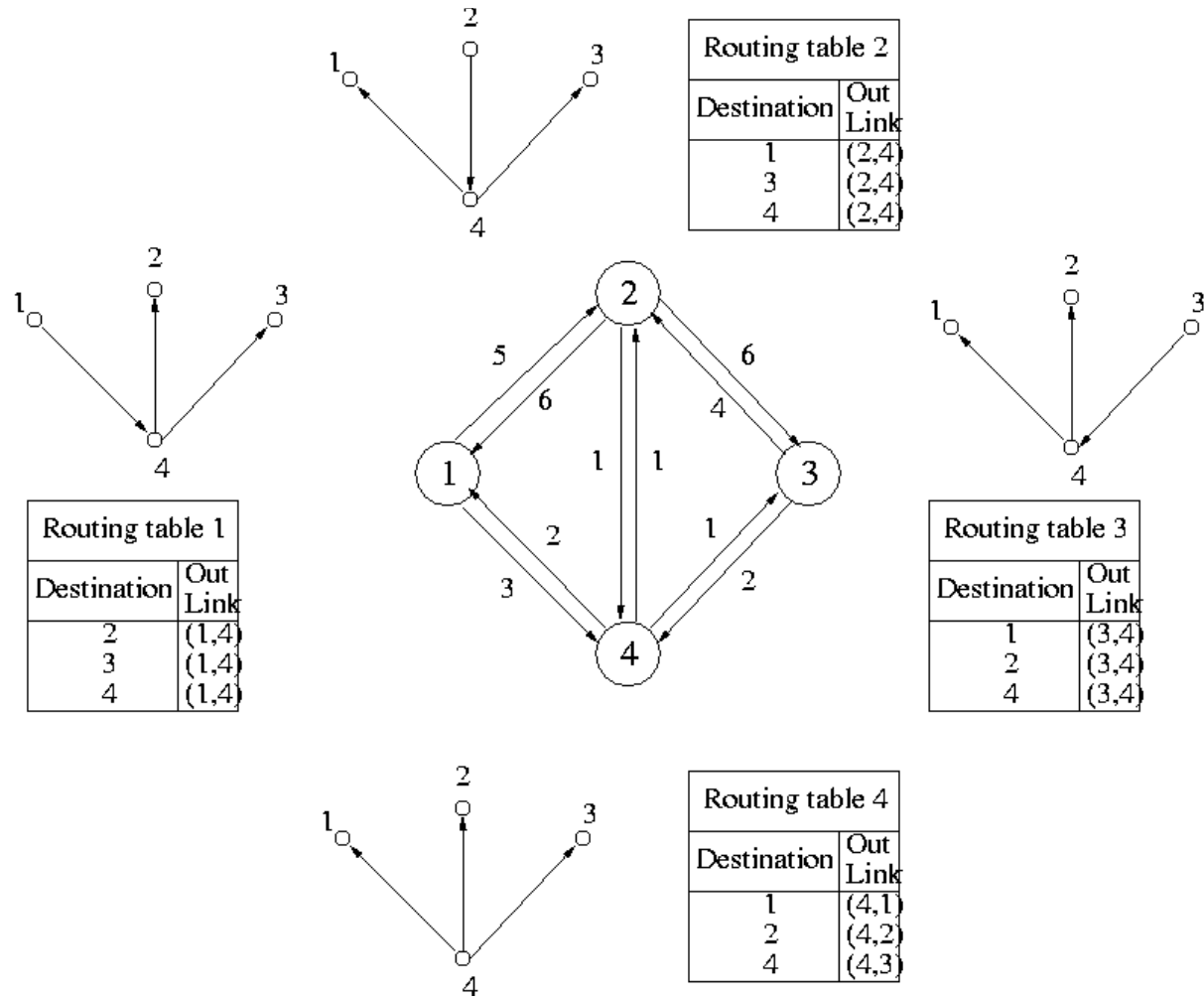
Routing Protocol Categories

- Interior Gateway Protocol (IGP) or Intradomain Routing Protocol
 - Within one AS
 - RIP
 - OSPF
- Exterior Gateway Protocol (EGP) or Interdomain Routing Protocol
 - Between different AS
 - Border Gateway Protocol (BGP)



ARPANET Routing Table Example

- Each node calculates a shortest path from itself to each destination
- The first link in the shortest path is entered in the node's routing table





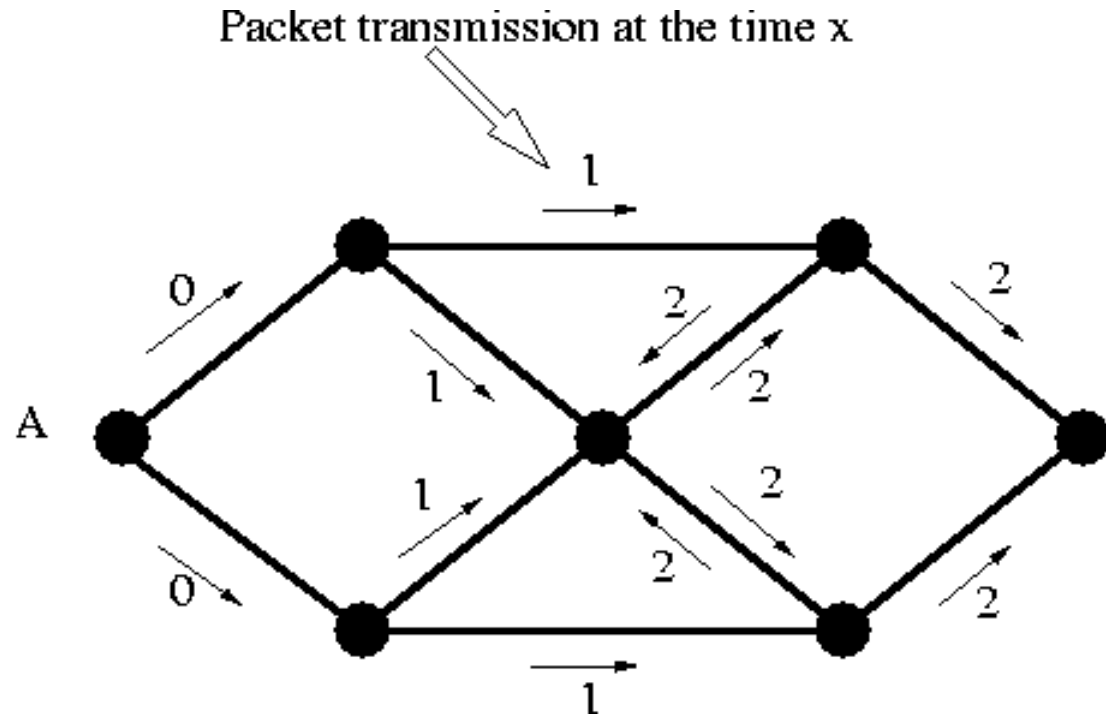
Broadcasting Techniques

- Used by routing protocols to transmit routing updates upon changes in the network topology due to link failures and repairs
 - Flooding
 - Spanning Tree



Flooding (1)

- The origin node A sends its information in the form of a packet to its neighbors
- The neighbors relay it to their neighbors and so on until the packet reaches all network nodes





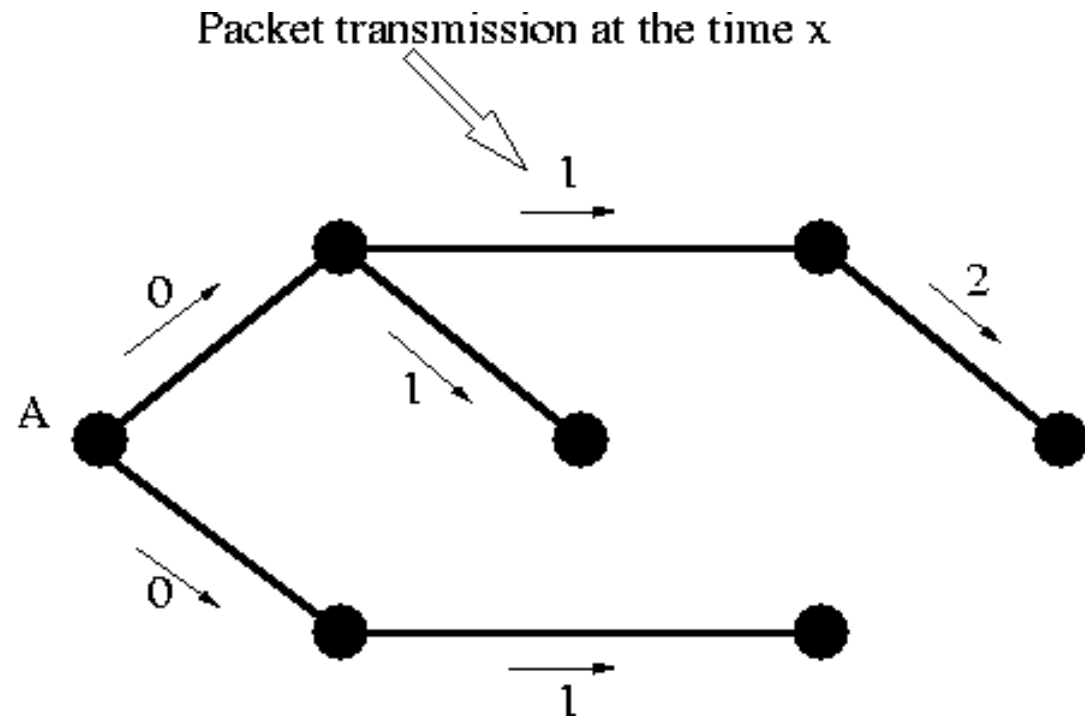
Flooding (2)

- Two rules to limit number of packet transmissions
 - A node does not relay the packet back to the node from which the packet was obtained
 - A node transmits the packet to its neighbors at most once
 - Ensured by including in the packet
 - ID number of the origin node
 - SEQUENCE NUMBER incremented with each new packet issued by the origin node
 - Node avoids transmitting the same packet more than once on each of its incident links by
 - Storing the highest sequence number received for each origin node
 - Not relaying packets with sequence number less or equal to the sequence number stored
- It requires a total number of packet transmissions per packet broadcast between L and $2L$
 - L = number of network links



Spanning Tree

- It is a connected subgraph of the network that includes all nodes and has no cycles
- It is more communication-efficient than flooding
- It requires a total of only $N-1$ packet transmissions per packet broadcast
 - N = number of network nodes





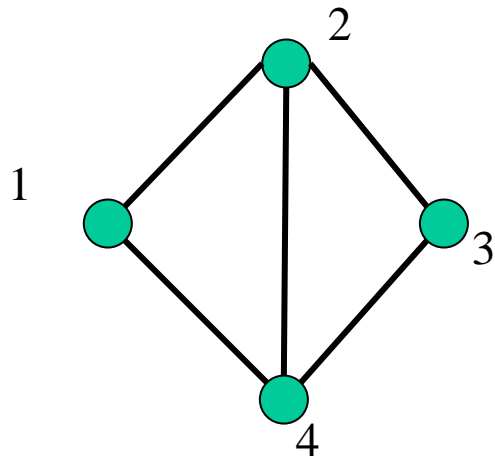
Network Algorithms and Shortest Path Routing

- Based on graph-theoretic problems
- Def. Undirected Graph
 - An Undirected Graph $G(V, E)$ is a finite nonempty set V of VERTICES and a collection E of pairs of distinct vertices from V . Each unordered pair of vertices E is called an EDGE
- If v_1 and v_2 are vertices in an undirected graph and (v_1, v_2) is an edge \Rightarrow this edge is said to be INCIDENT on v_1 and v_2



Network Algorithms and Shortest Path Routing (2)

- Def. WALK
 - A WALK in an undirected graph G is a sequence of vertices (v_1, v_2, \dots, v_l) such that each of the pairs $(v_1, v_2), (v_2, v_3), \dots, (v_{l-1}, v_l)$ is an edge of G
- Def. PATH
 - A walk with no repeated edges (nodes) is a PATH
- Def. CYCLE
 - A walk (v_1, v_2, \dots, v_l) with $v_1=v_l, l>3$, and no repeated nodes other than $v_1=v_l$ is called a cycle



WALKS	PATH	CYCLE
(1,4,2,3)	(1,4,2,3)	(1,4,2,1)
(1,4,2,1)	(2)	
(1,4,2,1,4,1)		
(2,3,2)		
(2)		



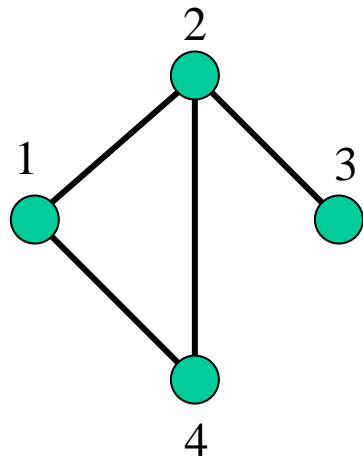
Network Algorithms and Shortest Path Routing (3)

- Def. CONNECTED GRAPH
 - A GRAPH is CONNECTED if for each vertex i there is a path $(i = v_1, v_2, \dots, v_l = j)$ to each other node j
- Lemma
 - Hypthesis: G connected graph, S any nonempty strict subset of vertices V
 - Thesis: at least one edge $(i,j) \exists$ with $i \in S$ and $j \notin S$
- Def. SUBGRAPH
 - $G'(V',E')$ is a SUBGRAPH of $G(V,E)$ if G' is a graph, $V' \subset V$, and $E' \subset E$

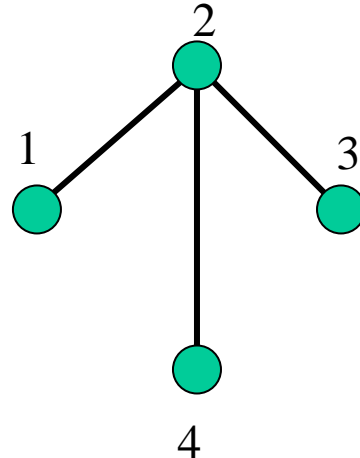


Network Algorithms and Shortest Path Routing (4)

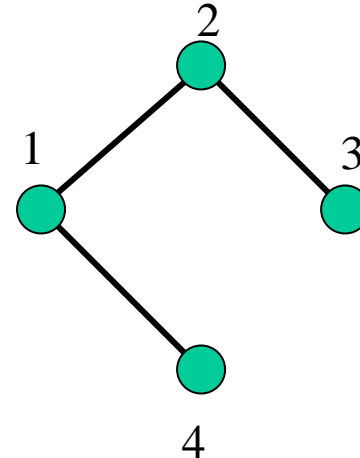
- Def. TREE
 - A TREE is a connected graph that contains no cycles
- Def. SPANNING TREE
 - A SPANNING TREE of a graph G is a subgraph of G that is a tree and includes all the nodes of G



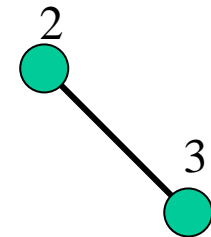
GRAPH



SUBGRAPH
SPANNING TREE



SUBGRAPH
SPANNING TREE

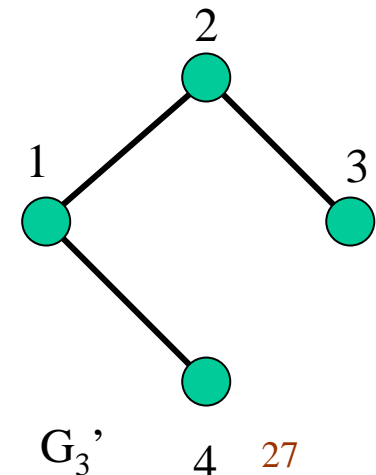
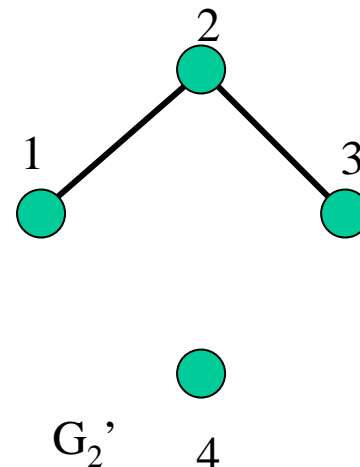
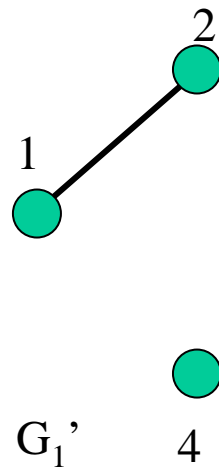
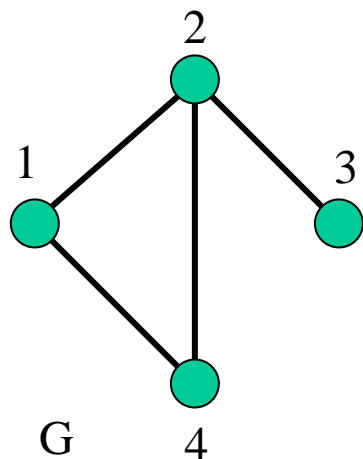


SUBGRAPH



Spanning Tree Algorithm

- Algorithm to build a spanning tree of an arbitrary connected graph
 1. v arbitrary vertex $\in V$, $V' = \{v\}$, and let $E' = \emptyset$
 2. If $V' = V \Rightarrow$ STOP $G'(V', E')$ is a spanning tree
Else GO TO step 3
 3. Let $(i, j) \in E$ be an edge with $i \in V'$, $j \in V - V'$, update V' and E' by $V' := V' \cup \{j\}$, $E' := E' \cup \{(i, j)\}$, GO TO step 2





Spanning Tree Algorithm (2)

- Proposition
 - Hyp.: G connected graph with V vertices and E edges
 - Th.:
 1. G contains a spanning tree
 2. $E \geq V-1$
 3. G is a tree if and only if $E=V-1$



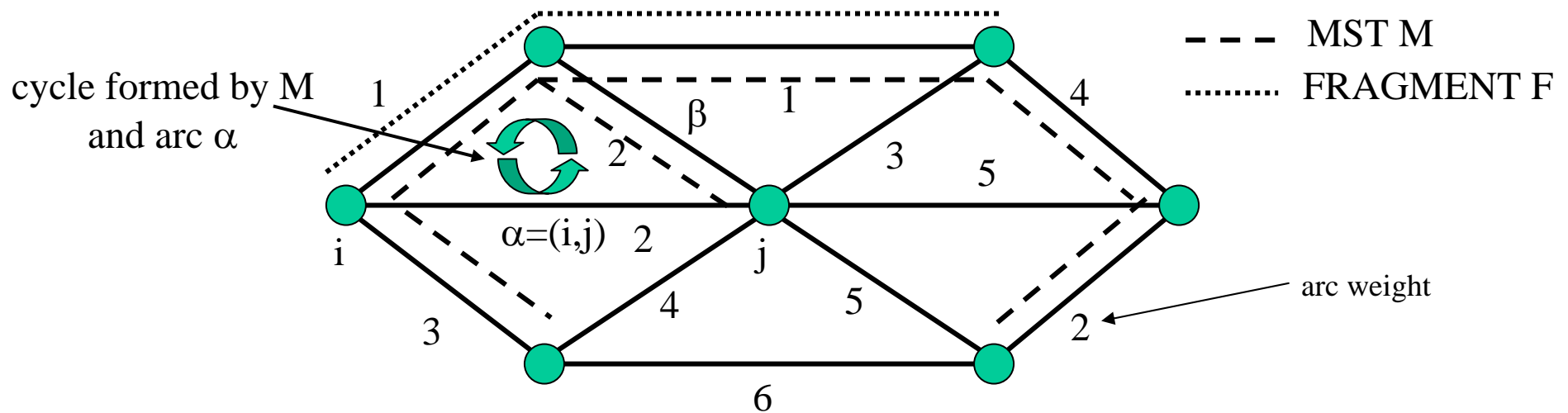
Minimum Weight Spanning Tree (MST)

- $G(V,E)$ connected undirected graph with V vertices and E edges, $w_{i,j}$ weight for each edge $(i,j) \in E$
- Def. MINIMUM WEIGHT SPANNING TREE
 - A MINIMUM WEIGHT SPANNING TREE (MST) is a spanning tree with minimum sum of arc weights
- An arc weight represents the communication cost of a message along the arc in either direction
- The total spanning tree weight represents the cost of broadcasting a message to all nodes along the spanning tree
- Def. FRAGMENT
 - Any subtree (i.e., a subgraph that is a tree) of an MST is called a FRAGMENT
- Def. OUTGOING EDGE
 - An EDGE having one vertex in a fragment and the other vertex not in this fragment is called an OUTGOING EDGE from the fragment



Central Proposition for MST Algorithms

- Proposition 1
 - Hyp.: fragment F , $\alpha=(i,j)$ minimum weight outgoing edge from F , vertex $j \notin F$
 - Th.: F , extended by edge α and vertex j , is a fragment





Central Proposition for MST Algorithms

- Proposition 1 proof
 - M is the MST of which F is a subtree
 - if edge $\alpha \in M \Rightarrow \text{STOP}$
 - otherwise let α be a minimum weight arc outgoing from F and $\alpha \notin M$. Let $\beta \neq \alpha$ an arc outgoing from F and $\beta \in M$ and to the cycle formed α and M . Deleting β from M and adding α results in a subgraph M' with $(N-1)$ edges and no cycles which, therefore, must be a spanning tree. Since α has less or equal weight than β , M' must be an MST, so F extended by α and j forms a fragment
- Proposition 1 can be used as the basis for MST construction algorithm



Disjoint Set Data Structure and Operations

- Disjoint-set data structure
 - maintains a collection $S = \{S_1, S_2, \dots, S_k\}$ of disjoint dynamic sets
- Each set is identified by a **REPRESENTATIVE** which is some member of the set
- Each element of a set is represented by an object x



Disjoint Set Data Structure and Operations (2)

- **MAKE-SET** (x)
 - creates a new set whose only member (and thus representative) is pointed to by x . Since the sets are disjoint, we require that x not already be in a set
- **UNION** (x, y)
 - unites the dynamic sets that contain x and y , say S_x and S_y , into a new set that is the union of these two sets. The two sets are assumed to be disjoint prior to the operation. Since we require the sets in the collection to be disjoint we “destroy” set S_x and S_y removing them from the collection S
- **FIND-SET** (x)
 - returns a pointer to the representative of the (unique) set containing x



MST Prim-Dijkstra Algorithm

- Prim-Dijkstra's (PD) algorithm operates much like Dijkstra's algorithm for finding shortest path
- PD algorithm starts with an arbitrarily selected single vertex as a fragment and enlarges the fragment by successively adding a minimum weight outgoing edge
- PD algorithm has the property that edges in the set A always form a single tree
- At each step, a light edge connecting a vertex in A (fragment) to a vertex in $V-A$ is added to the tree
- This strategy is “greedy” since the tree is augmented at each step with an edge that contributes the least to the tree's weight



MST Prim-Dijkstra Algorithm (2)

- $G(V,E)$ =graph, w =function calculating the weight of edge (u,v) , r =root, Q =priority queue, $key[v]$ =weight of vertex v in Q , $Adj[v]$ = vertices adjacent to v , $EXTRACT-MIN(Q)$ = extract min element in Q , $\pi [r]$ =parent of r
- $MST-PD (G, w, r)$

1. $Q \leftarrow V$
2. for each $u \in Q$
3. do $key [u] \leftarrow \infty$
4. $key [r] \leftarrow 0$
5. $\pi [r] \leftarrow NIL$
6. while $Q \neq \emptyset$
7. do $u \leftarrow EXTRACT-MIN(Q)$
8. for each $v \in Adj [u]$
9. do if $v \in Q$ and $w(u,v) < key [v]$
10. then $\pi [v] \leftarrow u$
11. $key [v] \leftarrow w(u,v)$

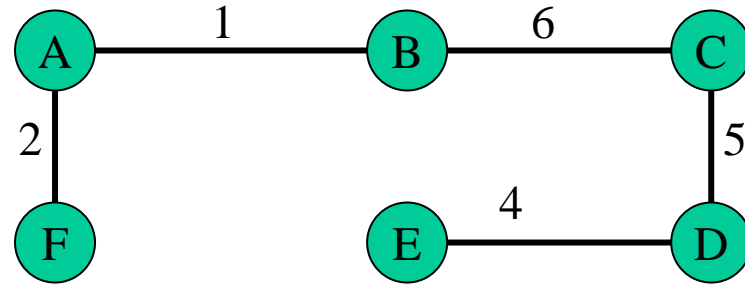
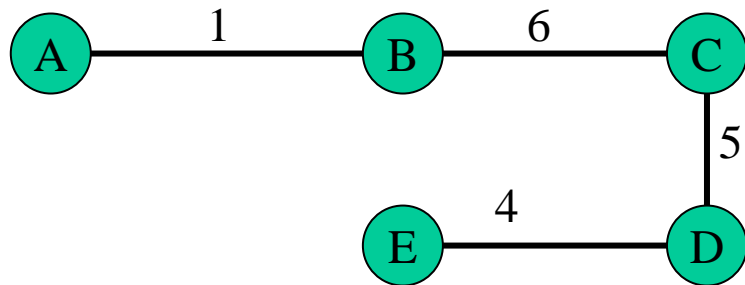
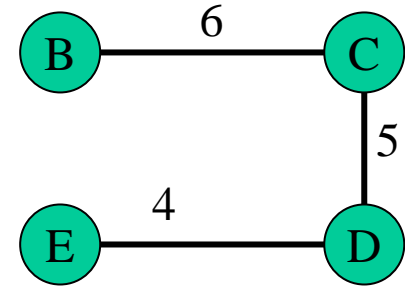
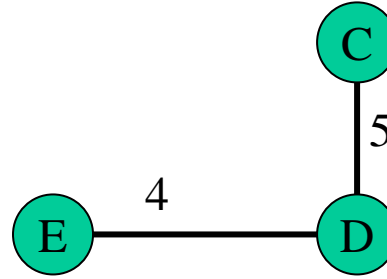
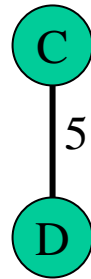
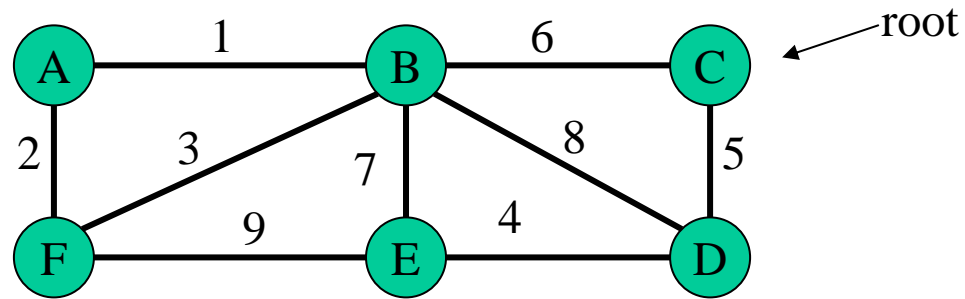


MST Prim-Dijkstra Algorithm (2)

- Lines 1-4 initialize the priority queue Q to contain all the vertices and set the key of each vertex to ∞ except for the the root r , whose key is set to 0
- Line 5 initialize $\pi [r]$ to NIL
- Line 7 identifies a vertex $u \in Q$ incident on a light edge crossing the cut $(V-Q, Q)$
- Removing u from the set Q adds it to the set $V-Q$ of vertices in the tree
- Line 8-11 update the key and π fields of every vertex v adjacent to u but not in the tree
- The updating maintains the invariants that $\text{key}[v] = w(v, \pi[v])$ and that $(v, \pi[v])$ is a light edge connecting v to some vertex in the tree
- The performance of PD algorithm depends on how the priority Q is implemented



MST Prim-Dijkstra Algorithm Example





MST Kruskal Algorithm

- Kruskal's algorithm proceeds by building up simultaneously several fragments that eventually join into a MST
- Only one edge at a time is added to the current set of fragments
- Kruskal's algorithm finds a safe edge to add to the growing forest (set of disjoint trees) by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of least weight
- Kruskal's algorithm is a greedy algorithm because at each step it adds to the forest an edge of least possible weight
- It is possible to add a minimum weight outgoing edge simultaneously to each fragment in a distributed algorithmic manner \Rightarrow distributed algorithm for constructing MST



MST Kruskal Algorithm (2)

- $G(V, E)$ =graph, w = function calculating the weight of the edge (u, v) , A =MST
- MST-Kruskal (G, w)
 1. $A \leftarrow \emptyset$
 2. for each vertex $v \in V$
 3. do MAKE-SET(v)
 4. sort the edges in E by nondecreasing weight w
 5. for each edge $(u, v) \in E$, in order by nondecreasing weight
 6. do if FIND-SET(u) \neq FIND-SET(v)
 7. then $A \leftarrow A \cup \{(u, v)\}$
 8. UNION(u, v)
 9. return A

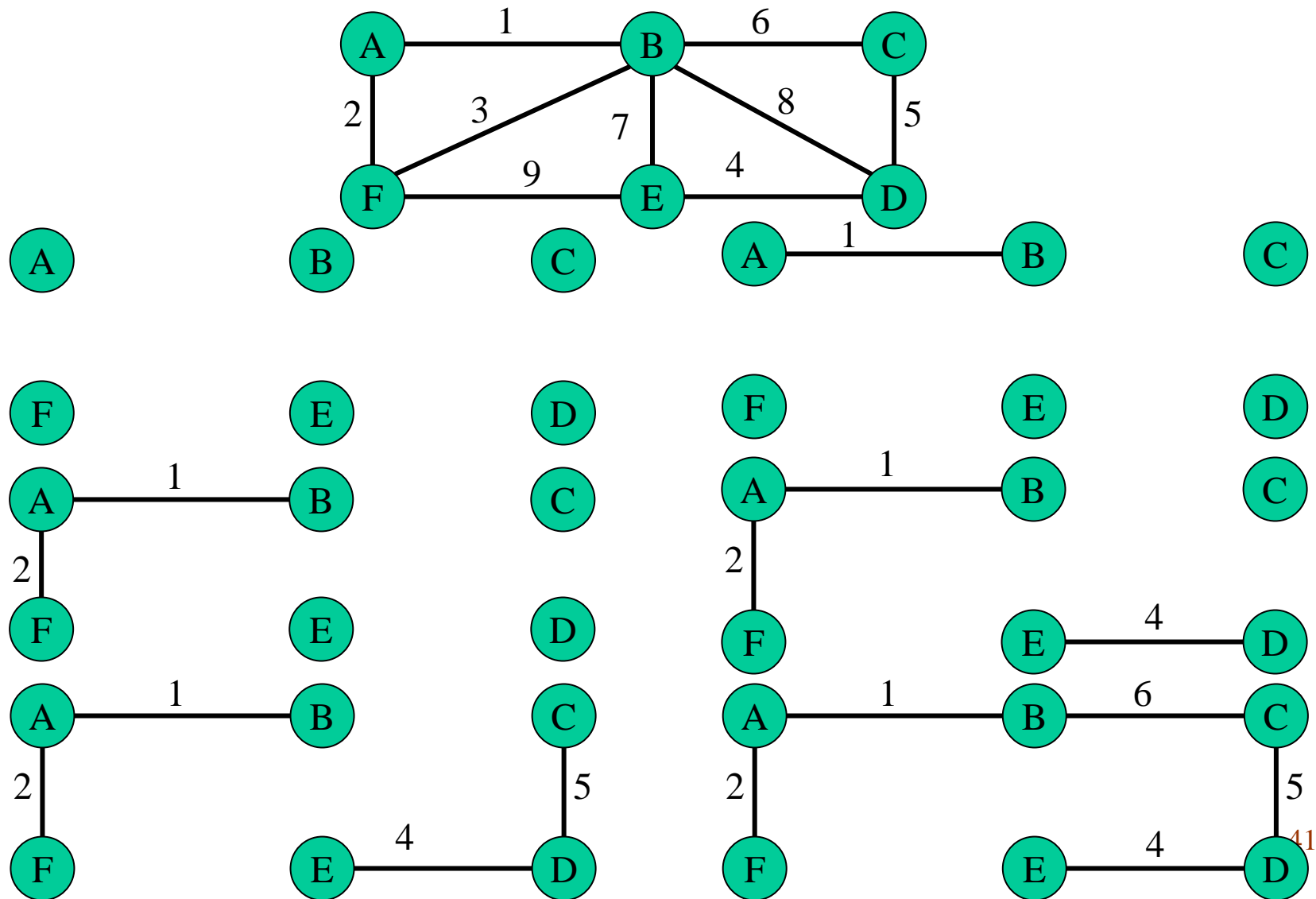


MST Kruskal Algorithm (3)

- Lines 1-3 initialize the set A to the empty set and create $|V|$ trees, one containing each vertex
- The edges in E are sorted into order by nondecreasing weight in line 4
- The for loop in lines 5-8 checks, for each edge (u,v) , whether the endpoints u and v belong to the same tree
 - IF so if edge (u,v) is added it creates a cycle \Rightarrow edge (u,v) discarded
 - ELSE the two vertices belong to different trees and edge (u,v) is added to A in line 7
- The vertices in the two trees are merged in line 8
- The running of Kruskal's algorithm for a graph $G(V,E)$ depends on the implementation of the disjoint set data structure



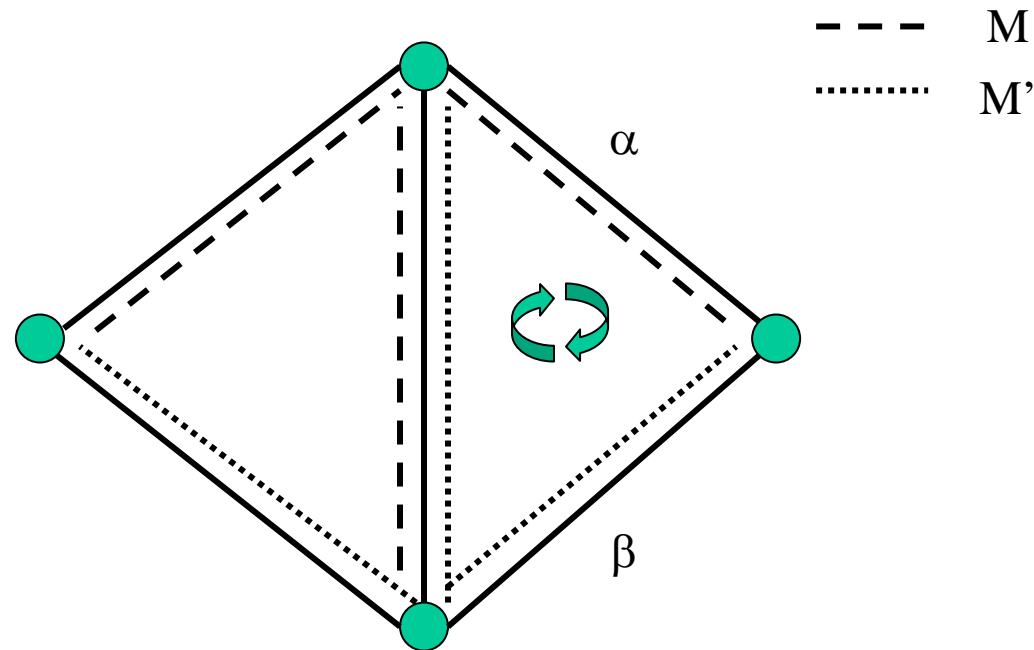
MST Kruskal Algorithm Example





MST Uniqueness

- Proposition 2
 - If all edge weights are distinct, there exists a unique MST





MST Uniqueness (2)

- Proof:
 - Suppose two distinct MSTs exist M and M'
 - α = minimum weight edge that belongs to either M or M' but not to both
 - Assume $\alpha \in M$
 - Consider the graph formed by the union $M'' = M' \cup \alpha$.
 - M'' must contain a cycle (since $\alpha \notin M'$) and at least one edge of this cycle, call it β , $\notin M$ (otherwise M would contain a cycle)
 - Since weight of $\alpha <$ weight of $\beta \Rightarrow$ deleting β from M' and adding α in its place results in a spanning tree of strictly smaller weight than M'
 - This is contradiction since M' is optimal



MST Uniqueness (3)

- When edge weights are not distinct the ties between edges with the same weight can be broken by using the identities of their nodes
- If $w(i,j)=w(k,m)$ with $i < j$ and $k < m$
 - Prefer (i,j) if $i < k$ or if $i = k$ and $j < m$
 - Prefer (k,m) otherwise