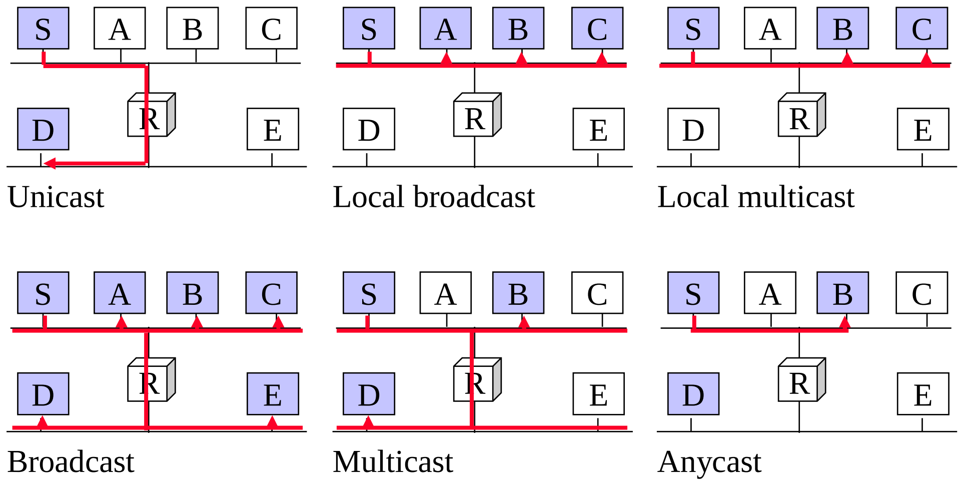
Multicast routing principles in Internet

Motivation

Recap on graphs

Principles and algorithms

Unicast, Broadcast, Multicast…



Unicast

Local broadcast

Local multicast

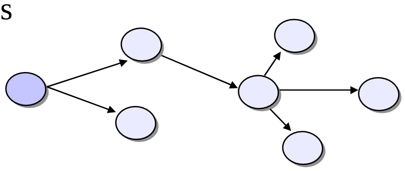
Broadcast

Multicast

Anycast

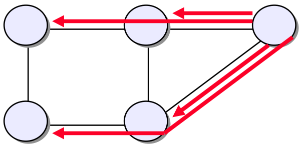
Multicast capability has been and is under intensive development since the 1990’s

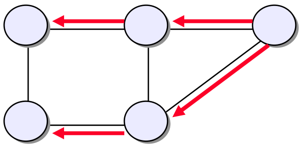
- MBone used to multicast IETF meetings from 1992
- Extends LAN broadcast capability to WAN in an efficient manner
- Valuable applications
  - resource discovery
  - multimedia conferencing, teaching, gaming
  - streaming audio and video
  - network load minimization by replacing many point-to-point transmissions



Multicast reduces network load and delay

- For example



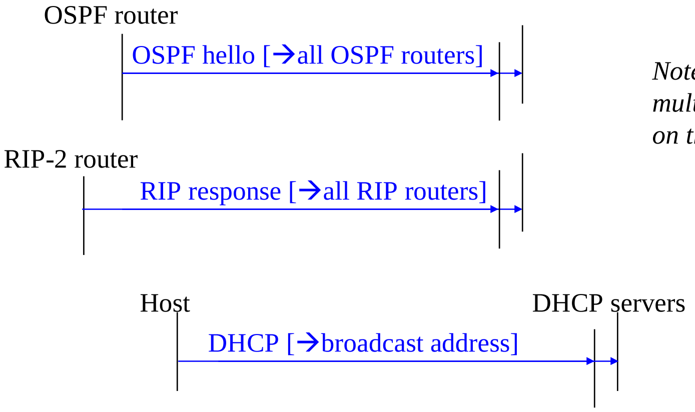


- 6 transmissions vs. 4 transmissions

- Generally unreliable transmission (UDP)
- In reliable multicast the source must retransmit missing packets with unicast

Resource discovery by multicast simplifies network management (1)

- No need for lists of neighbors, just use standard multicast address


OSPF router

OSPF hello [→all OSPF routers]

RIP-2 router

RIP response [→all RIP routers]

Host

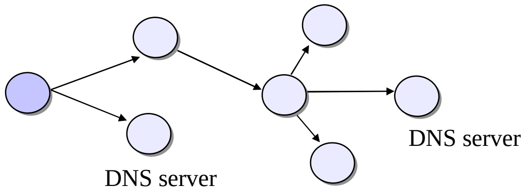
DHCP servers

DHCP [→broadcast address]

Note: in this case, multicast only on the local network

Resource discovery by multicast simplifies network management (2)

- How to find corporate DNS-server? Multicast to all nodes in corporate network ⇒ Routers need to forward multicast packets.

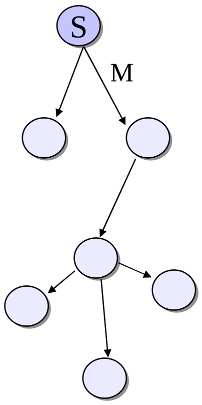


- Network is easily flooded with messages.
- TTL can be used to limit the scope of a broadcast – “expanding ring search”
  - ⇒ find nearest DNS (or other server)
  - when TTL=0 in multicast packet, no ICMP message is returned

Conferencing requirements include

- Multiple sources, multiple recipients, multiple media
- Variable membership
- Small conferences with intelligent media control (what is sent to where)
- Large conferences require media processing in special devices
- QoS is important
  - Low delay
  - Low delay variation
  - Low packet loss

Multipoint sessions differ from point-to-point communication



- Participants may join and leave the session.
- Receiver-makes good principle instead of session parameter negotiation.
- Window based flow control does not apply:
  → use UDP / connectionless protocols

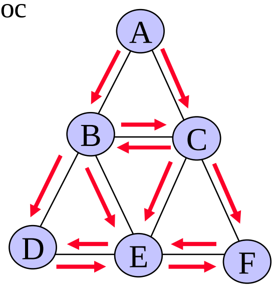
- Packets are sent to a group address instead of a host address

Multicast routing algorithms

Flooding is the simplest “multicast” algorithm

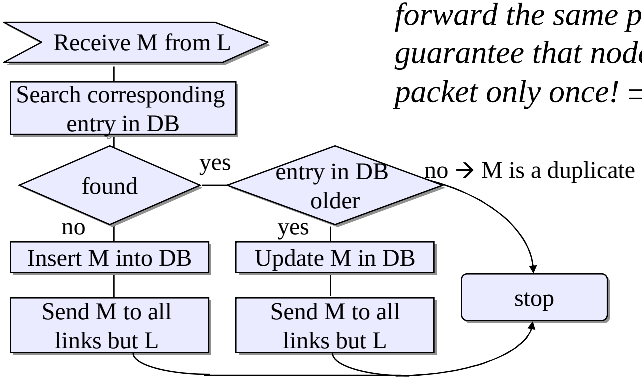
- Flooding distributes a packet/message to all nodes in the network
  - No group membership ⇒ broadcast
  - ”Multicast” implemented by filtering packets
- Used in OSPF, Usenet news, Peer-to-peer systems, Ad hoc routing protocols, etc.
- Avoiding duplicate receptions ⇒ avoiding loops
  - State information in the nodes
    - A permanent database as in OSPF
    - Cache of recently seen messages

    ⇒ Messages must have a unique identifier
  - State information in the message (trace information)

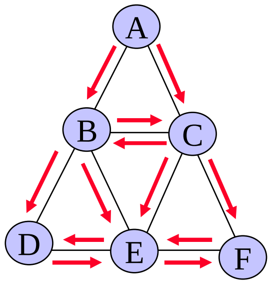


State information in the nodes avoids forwarding the same message twice

Flooding algorithm:

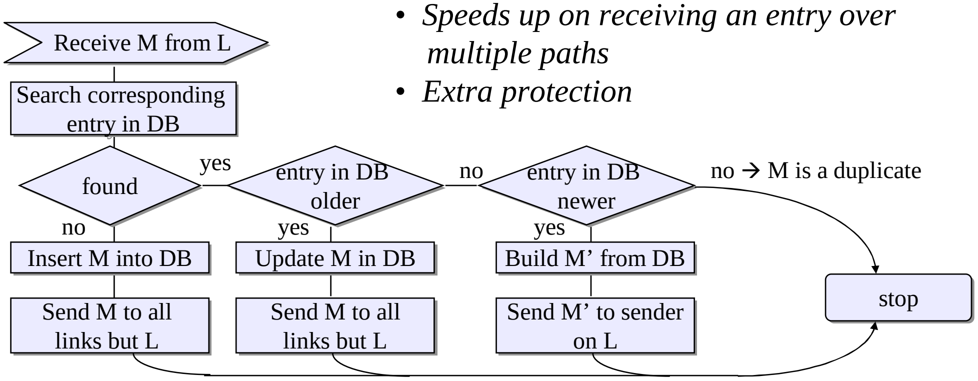

Receive M from L
Search corresponding entry in DB
found
yes
no
entry in DB older
no → M is a duplicate
yes
Insert M into DB
Update M in DB
Send M to all links but L
Send M to all links but L
stop

Flooding guarantees that node will not forward the same packet twice. It does not guarantee that node will receive the same packet only once! ⇒ greedy algorithm



OSPF updates the previous hop with the newest entry (if available)

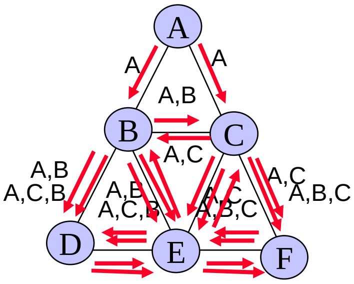
Flooding algorithm:


Receive M from L
Search corresponding entry in DB
found
yes
no
entry in DB older
no
entry in DB newer
no → M is a duplicate
yes
yes
Insert M into DB
Update M in DB
Build M' from DB
Send M to all links but L
Send M to all links but L
Send M' to sender on L
stop

Not necessary for correct flooding

- *Speeds up on receiving an entry over multiple paths*
- *Extra protection*

Trace information is an alternative to the database in flooding



- Trace info in message lists all passed nodes
- If the neighbor is in trace, do not send
- May forward the same message several times ⇒ not useful as such
- Traces can be combined with state information (DB) in node
  - First check trace, then DB
  - Avoids costly database reads
  - E.g. Usenet news

Observations about flooding

- Works well on the application layer, not efficient on network layer
  - Storing state information about all forwarded packets is not feasible
- Each node may receive the same message several times
  - Number of receptions depends on number of neighbors
- Flooding does not depend on routing tables ⇒ robust
- Limiting TTL
  - To avoid loops
  - To reduce the scope

Example: using flooding in peer-to-peer networks

- In e.g. Gnutella, Kazaa, etc. (unstructured peer-to-peer networks)
- Task: Find the users having the file X
- Implementation: Flood a search request to all users within a given distance (TTL). The users with file X send back a reply to the searching user.
- About 5 receptions of a message per node in a typical Gnutella topology
- Peer-to-peer systems use overlay networks (networks implemented on the application layer)

Networks are modeled as graphs

G = (V, E)

- V – set of ***vertices*** or ***nodes*** (non-empty, finite set)
- E – set of ***edges*** or ***links*.**

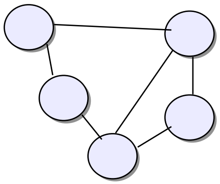
$E = \{e_j \mid j = 1, 2, \ldots, M\}$
$e_j = (v_i, v_k) = (i, k)$

- Nodes *i* and *k* are ***adjacent*** if link (*i*, *k*) exists.
- Nodes *i* and *k* are also called ***neighbors***.

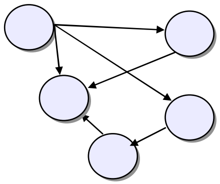
· Vertex, node – kärki, solmu
· Edge, link – syrjä, linkki, sivu, kaari, haara
· Adjacent – viereinen
· Neighbor – naapuri

Links are bi-directional, arcs are unidirectional

- Unidirectional links, $a_j = (v_i, v_k) = [i, k]$ are called ***arcs***.



Undirected graph (only links)



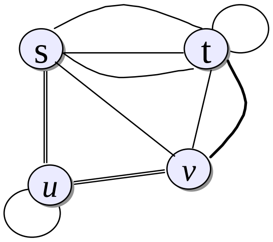
Directed graph (also arcs)

- The ***degree of a node*** is the number of links incident on the node (=number of neighbors in a simple graph)

- If links and nodes have properties, the graph is called a ***network***.

· Degree of a node – solmun aste
· Arc – kaari
· Directed graph – suunnattu graafi

Graphs with parallel links are called ***multigraphs***



- Links between a node and itself are ***self loops***.

- A graph with no parallel links and no self loops is a ***simple graph***.

- A ***path*** in a network is a sequence of links beginning at some node *s* and ending at some node *t* (= ***s,t-path***).

- If s = t, the path is called a ***cycle***. If an intermediate node appears no more than once, it is a ***simple cycle***.

· Cycle, loop – silmukka
· Path – polku

A graph is ***connected*** if there is at least one path between every pair of nodes.

- A subset of nodes with paths to one another is a ***connected component***.

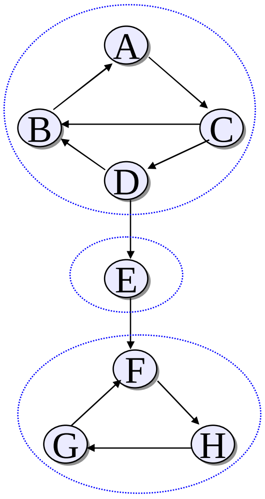
Reflective: By def. ∃ i,i-path
Symmetric: ∃ i,j-path ⇒ ∃ j,i-path
Transitive: ∃ i,j-path and ∃ j,k-path ⇒ ∃ i,k-path

⇨ Components are equivalence classes and the component structure is a partition of the graph.

Partition applies to links and nodes alike.

· Connected – yhteydellinen, yhdistetty

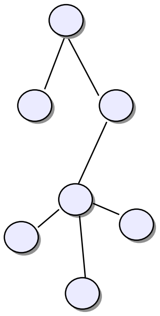
A directed graph is ***strongly connected*** if there is a directed path from every node to every other node.



- Directed connectivity is not symmetric.
- A subset of nodes with directed paths from any one node to any other is a ***strongly connected component***.
- A node belongs to exactly one strongly connected c. An arc is part of at most one strongly connected c.

· Strongly connected – vahvasti yhteydellinen
· Directed path – suunnattu polku

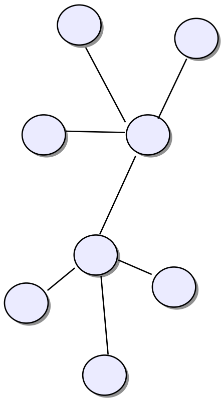
A ***tree*** is a connected graph without cycles



- A ***leaf*** has the degree 1
- Given a graph $G = (V, E)$, $H = (V', E')$ is a ***subgraph*** of G if $V' \subset V$ and $E' \subset E$
- A ***spanning tree*** is a connected graph without cycles. (Connects all nodes in the graph)
- A ***forest*** is a (not necessarily connected) graph without cycles

· Subgraph – aligraafi
· Tree – puu
· Spanning tree – virittäjäpuu
· Forest – metsä

Spanning trees model minimally connected networks



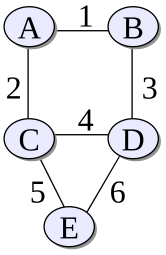
- A ***spanning tree*** connects all nodes without loops.
- Only a single path exists between any two nodes in a spanning tree ⇒ routing is trivial.
- If a graph has N nodes, any tree spanning the nodes has exactly N - 1 edges.
- Any forest with k components has exactly N - k edges.
  - proof by induction starting from graph with no edges.

A set of edges whose removal disconnects a graph is called a ***disconnecting set.***

- ***XY-cutset*** partitions a graph to subgraphs X and Y.
- In a tree any edge is a ***minimal cutset***.
- A minimal set of nodes whose removal partitions the remaining nodes into two connected subgraphs is called a ***cut***.

· Disconnecting set – erotusjoukko
· Cut – leikkaus
· XY-cutset – XY-leikkausjoukko

A graph can be presented with an ***adjacency matrix*** or an ***incidence matrix***



Node

$$\begin{array}{c|ccccc} \text{Node} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 0 & 0 \\ B & 1 & 0 & 0 & 1 & 0 \\ C & 1 & 0 & 0 & 1 & 1 \\ D & 0 & 1 & 1 & 0 & 1 \\ E & 0 & 0 & 1 & 1 & 0 \end{array}$$

Adjacency Matrix

Link

$$\begin{array}{c|cccccc} \text{Node} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline A & 1 & 1 & 0 & 0 & 0 & 0 \\ B & 1 & 0 & 1 & 0 & 0 & 0 \\ C & 0 & 1 & 0 & 1 & 1 & 0 \\ D & 0 & 0 & 1 & 1 & 0 & 1 \\ E & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

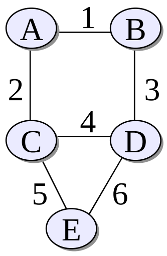
Incidence Matrix

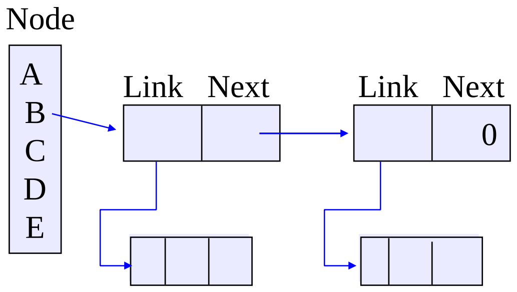
For an undirected graph, the adjacency matrix is symmetric.

For directed graphs, +1 is source and -1 is sink of an arc

· Adjacency matrix – Naapuruusmatriisi	*· Incidence matrix – Liitäntämatriisi*

For graph algorithms linked list presentation of adjacency is convenient

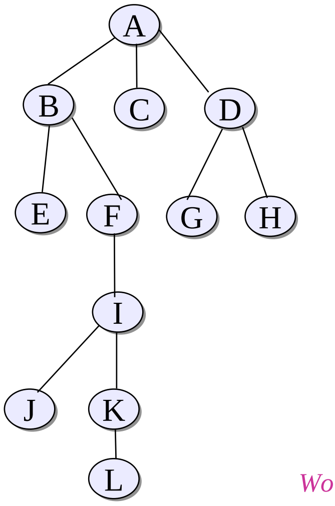



Node

Link Next

Link Next

A tree can be traversed by ***breadth-first-search***

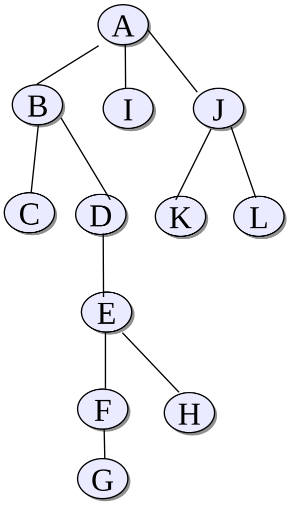


```
Void ← BfsTree (n, root, n_adj_list)
    dcl n_adj_list [n, list]  /* array of lists of neighbors
        scan_queue [queue]

    InitializeQueue (scan_queue)
    Enqueue (root, scan_queue)

    while NotEmpty (scan_queue)
        node ← Dequeue (scan_queue)
        Visit (node)
        for each (neighbor, n_adj_list[node])
            Enqueue (neighbor, scan_queue)
```

Works for directed links

· Breadth-first-search
– leveyshaku

A tree can also be traversed by ***depth-first-search***



```
Void ← DfsTree (n, root, n_adj_list)
    dcl n_adj_list [n, list]

    Visit (root)
    for each (neighbor, n_adj_list[node])
        DfsTree (n, neighbor, n_adj_list)
```

Works for directed links

· *Depth-first-search – syvyyshaku*

An undirected graph can be traversed by depth-first-search

```
Void ← Dfs (n, root, n_adj_list)
    dcl n_adj_list [n, list],
        visited [n]            /* keeps track of progress */

    void ← DfsLoop (node)
        if not visited [node]
            visited [node] ← TRUE
            Visit (node)
            for each (neighbor, n_adj_list[node])
                DfsLoop (neighbor)

    visited ← FALSE
    DfsLoop (root)
```

We can now find and label the connected components of an arbitrary graph

```
Void ← LabelComponents (n, n_adj_list)
    dcl n_component_nr[n], n_adj_list[n, list]

    void ← Visit(node)
    n_component_nr[node] ← ncomponents

    n_component_nr ← 0
    ncomponents ← 0
    for each (node, nodeset)
        if  (n_component_nr[node] = 0)
            ncomponents++
            Dfs (node, n_adj_list)
```

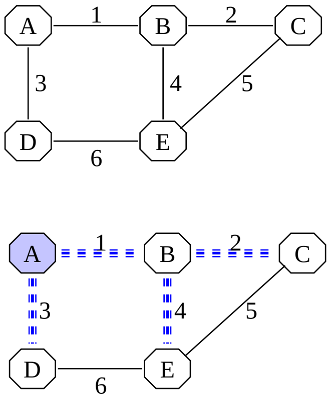
Minimum spanning tree (MST) is the spanning tree with minimum cost

- We assign a length to each edge of the graph.
  - "length" can be distance, cost, a measure of delay or reliability.
- We look for minimum total length, thus we talk about MST.
- If the graph is not connected, we may look for a minimum spanning forest.

$n = c + e$

where n is the number of nodes, c the number of components and e number of edges selected so far holds always.

Multicast to a spanning tree leads to reception only once in each node



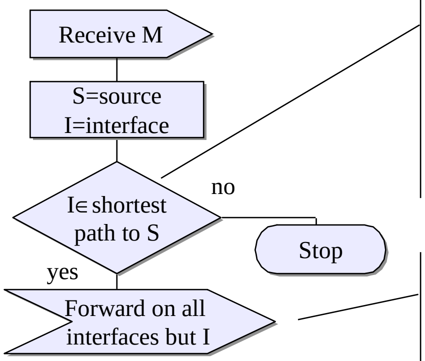
- Requires on/off bit (∈ ST) per link
- Disadvantages
  - No group membership
  - Concentrates traffic to the ST-links
- Ideal would be a tree that
  - spans the group members only
  - minimizes state information in nodes
  - optimizes routes based on metrics

A greedy minimum spanning tree algorithm

```
List ← Greedy (properties)
   dcl properties [list, list],
        candidate_set [list], solution [list]

   void ← GreedyLoop (*candidate_set, *solution)
      dcl test_set[list], candidate_set[list], solution[list]

      element ← BestElementOf (candidate_set)   /* for MST: shortest edge
      test_set ← element ∪ solution
      If test_set is feasible                   /* for MST: no cycles
          solution ← test_set
      candidate_set ← candidate_set \ element
      If candidate set is not Empty
          Greedy_Loop( *candidate_set, *solution)

   solution ← ∅
   If (candidate_set ← ElementsOf (properties)) is not Empty
      GreedyLoop (*candidate_set, *solution)
   return(solution)
```

Reverse-Path Forwarding is flooding on the shortest paths according to the routing table

- Reverse-path forwarding computes an implicit spanning tree per source



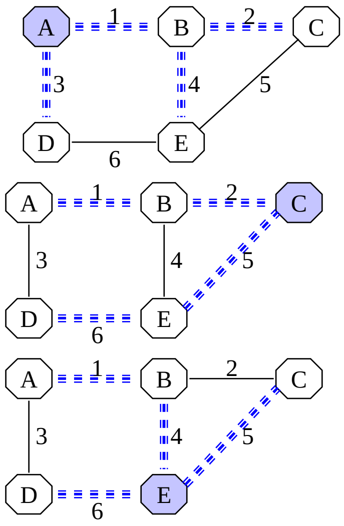
Note: In the unicast routing table, the path is computed from the current node to S. In symmetric networks = path from S to the current node.

DVMRP has a separate routing table with path from S to the current node.

Looking one step further: send only if the current node is on shortest path from S to next node.
Requires 1 bit per source and link in link state DB

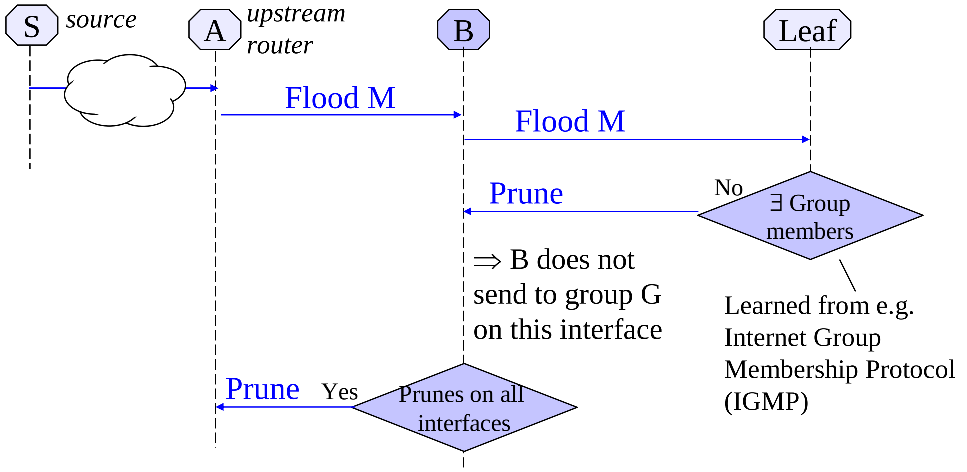
- First used in MBone

Reverse-Path Forwarding properties



- Different tree for each source ⇒ traffic is spread over multiple links leading to better network utilization
- Guarantees fastest possible delivery since it uses the shortest paths only
- No group membership ⇒ packets flooded to the whole network
  - can be scoped by TTL

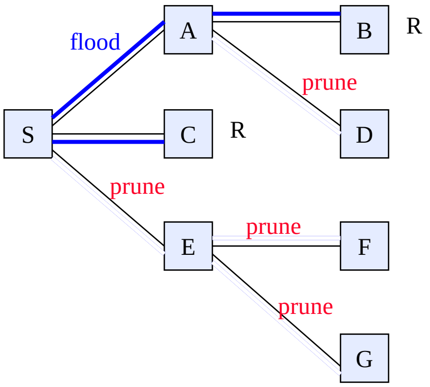
“Flood and prune” introduces dynamic group membership to Reverse-Path Forwarding



⇒ B does not send to group G on this interface

Learned from e.g. Internet Group Membership Protocol (IGMP)

”Flood and prune” – example



Drawbacks:

- first packet is flooded to the whole network
- all nodes must keep state per S and G.

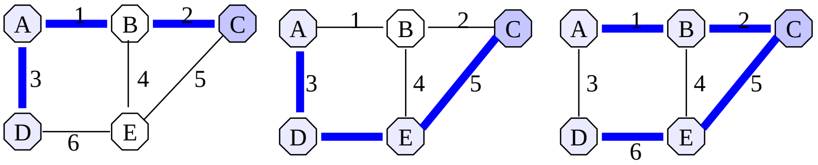
→ Suitable for dense trees

State is transient (timed out)

→ New members are detected

A *Steiner tree* spans the group members with a minimal total cost according to link metrics

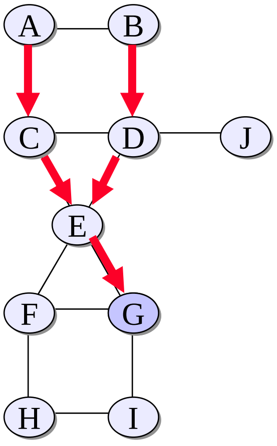
- Has never actually been used, only simulated:
  - Finding the minimum Steiner tree in a graph has exponential complexity
  - The tree is undirected: links must be symmetrical
  - Requires knowledge of the full topology, therefore it cannot be distributed (monolithic algorithm)
  - The tree is unstable when changes occur: traffic routes change dramatically when e.g a member leaves.



- Popular because of its mathematical complexity
- Leads to center-based approach (CBT, PIM)

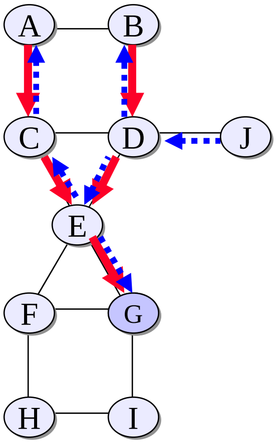
Center-based trees (1)

- Choose a center (rendezvous point, core)
- The recipients send join commands toward the center
  - Each router on the path toward the center processes the join message and adds the interface on which the join message is received to the forwarding table for the group. The join message continues to the next router toward the center.
  - If an intermediate router already is a member of the tree, it only adds the interface without forwarding the join message. Consequently, a branch is created in the multicast tree.



Center-based trees (2)

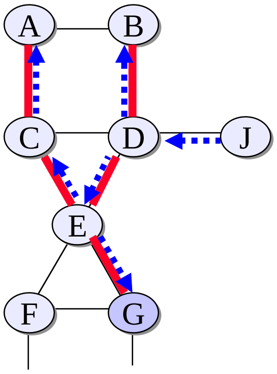
- Senders send packets to the center.
  - The first router that belongs to the group's tree intercepts the packet and forwards it to all interfaces of the multicast group. Each router receiving a packet forwards it on all interfaces belonging to the tree, except the one that the packet was received on.
  - Senders are not required to be members of the group

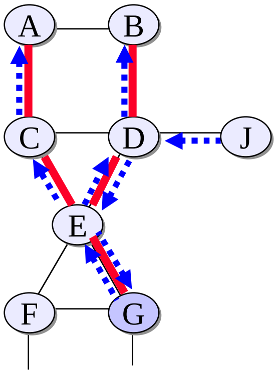


How to choose the center?

- Choosing a center that minimizes delay is a NP-hard problem
- Group membership varies

Unidirectional and bidirectional center-based trees





Bidirectional tree (e.g. CBT)

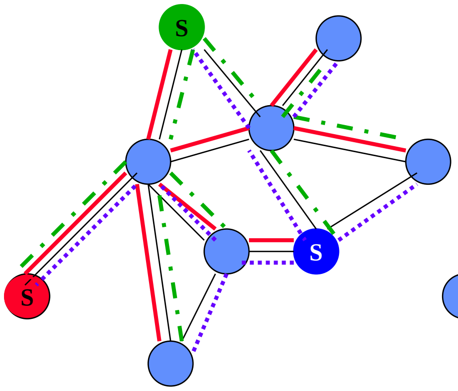
- First router intercepts the packet and distributes along tree

Unidirectional tree (e.g. PIM-SM)

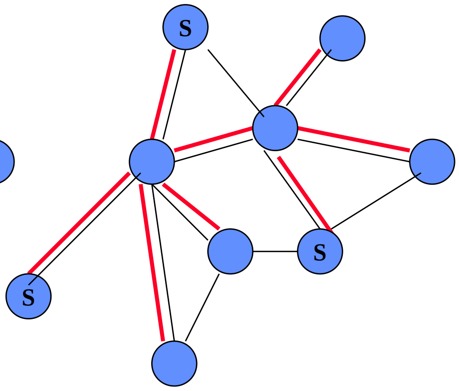
- The packet is first sent to the center, which distributes along the tree

Source based trees and shared trees

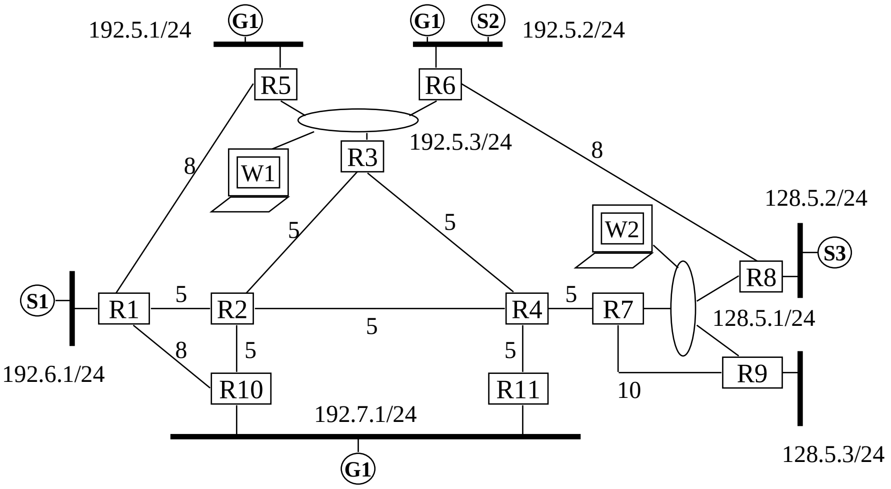
Source based trees (e.g. RPF)



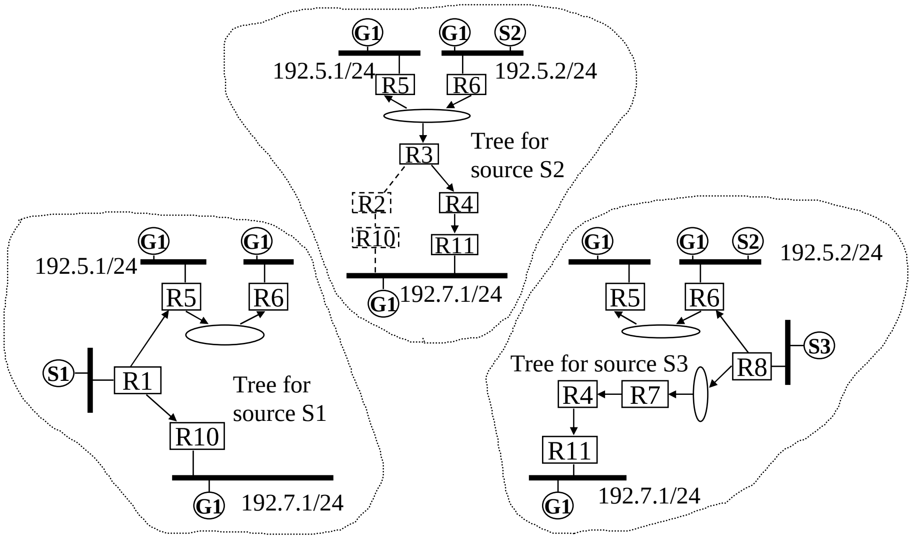
Shared tree (e.g. center-based tree)



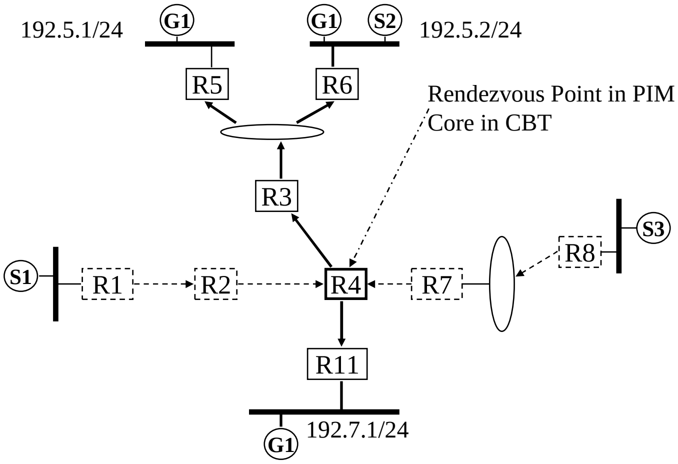
Multicast routing example



Source based trees for G1



Shared tree for G1


192.5.1/24
G1
G1
S2
192.5.2/24
R5
R6
Rendezvous Point in PIM
Core in CBT
R3
S1
R1
R2
R4
R7
R8
S3
R11
G1
192.7.1/24