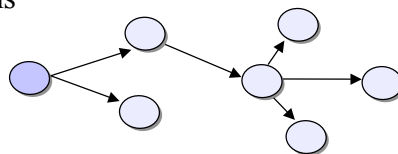


Multicast routing principles in Internet

Motivation
Recap on graphs
Principles and algorithms

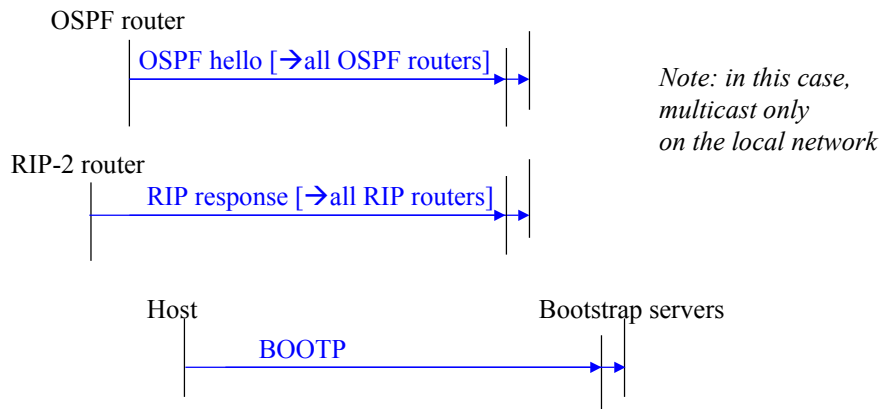
Multicast capability has been and is under intensive development since the 1990's

- 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



Resource discovery by multicast simplifies network management (1)

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

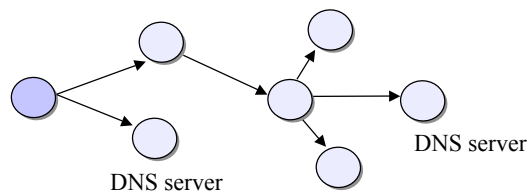


S-38.121 S-02 / RKa, NB

Multicast1-3

Resource discovery by multicast simplifies network management (2)

- How to find corporate DNS-server ⇒ multicast to all nodes in corporate network.



- 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, router does not return ICMP message!

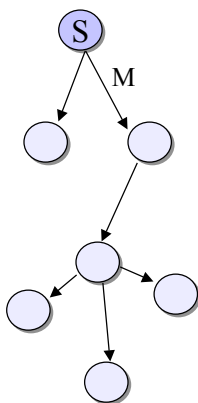
S-38.121 S-02 / RKa, NB

Multicast1-4

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

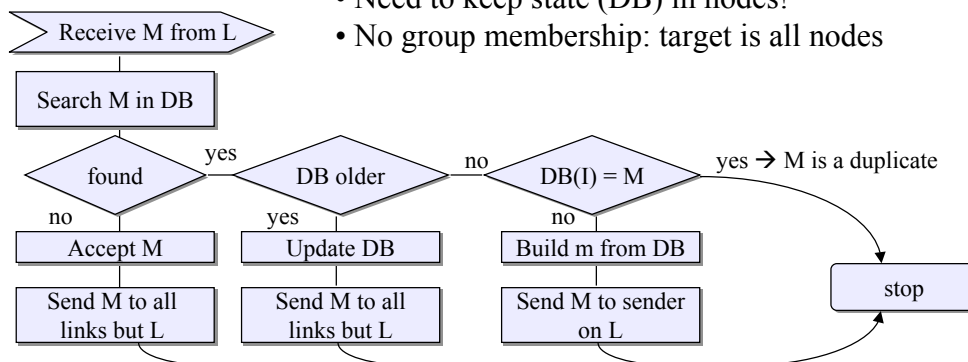
S-38.121 S-02 / RKa, NB

Multicast1-7

Flooding is the simplest multicast algorithm

Flooding algorithm:

- Need to keep state (DB) in nodes!
- No group membership: target is all nodes



- Examples: OSPF, Usenet news, etc.

S-38.121 S-02 / RKa, NB

Multicast1-8

Trace information is an alternative to the database in flooding

- Trace info in message lists all passed nodes
- Avoids costly database reads but may accept same message several times.
- If the neighbor is in trace, do not send

★ *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.*

⊕ *Flooding does not depend on routing tables ⇒ robust*

Networks are modeled as graphs

$$G = (V, E)$$

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

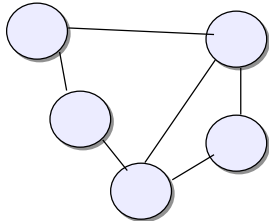
$$E = \{e_j \mid j = 1, 2, \dots, M\}$$

$$e_j = (v_i, v_k) = (i, k)$$

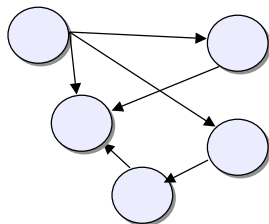
- Nodes i and k are **adjacent** if link (i, k) exists.
- Nodes i and k are also called **neighbors**.

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

Links are bi-directional, arcs are unidirectional



Undirected graph

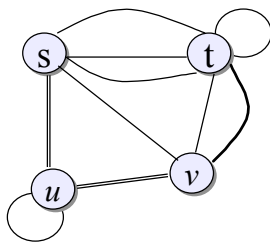


Directed graph

- The **degree of a node** is the number of its neighbors or the number of links incident on the node.
- Unidirectional links, $a_j = (v_i, v_k) = [i, k]$ are called **arcs**.
- 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**.
- 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. $\exists i, i$ -path

Symmetric: $\exists i, j$ -path $\Rightarrow \exists j, i$ -path

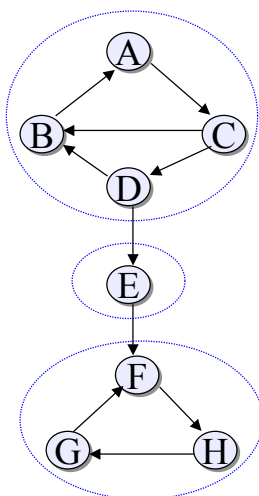
Transitive: $\exists i, j$ -path and $\exists j, k$ -path $\Rightarrow \exists i, k$ -path

⇒ 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 graph without cycles

- 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)
- If graph is not necessarily connected, we talk about a **forest**.

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

Spanning trees (ST) model minimally connected networks

- A **spanning tree** is a minimum cost network.
- Only a single path exists between any two nodes in a ST
⇒ 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*.

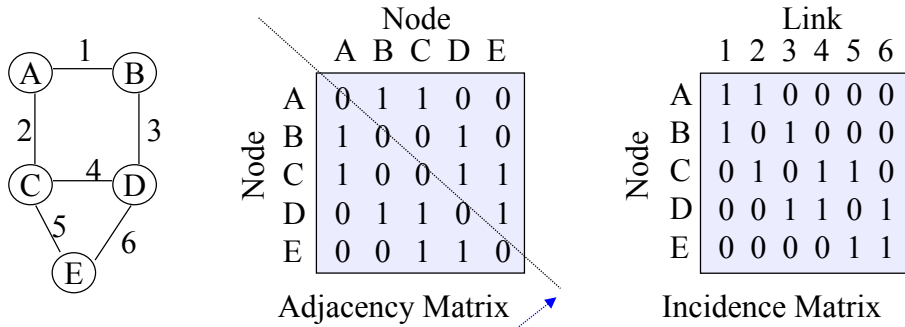
- *XY-cutset* partitions a graph to subgraphs X and Y.
- In a tree any edge is a *minimal cutset*.
- A minimal set of nodes whose removal partitions the remaining nodes into two connected subgraphs is called a *cut*.

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

Suomalaiset graafitermit

Vertex, node	- kärki, solmu	Connected	- yhteydellinen, yhdistetty
Edge, link	- syrjä, linkki, sivu, kaari, haara	Strongly connected	- vahvasti yhteydellinen
Adjacent	- viereinen	Subgraph	- aligraafi
Neighbor	- naapuri	Tree	- puu
Degree of a node	- solmun aste(?)	Spanning tree	- virittäjäpuu
Arc	- kaari	Forest	- metsä
Cycle, Loop	- silmukka	Disconnecting set	- erotusjoukko
Path	- polku	Cut	- leikkaus
Directed path	- suunnattu polku	XY-cutset	- XY-leikkausjoukko

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

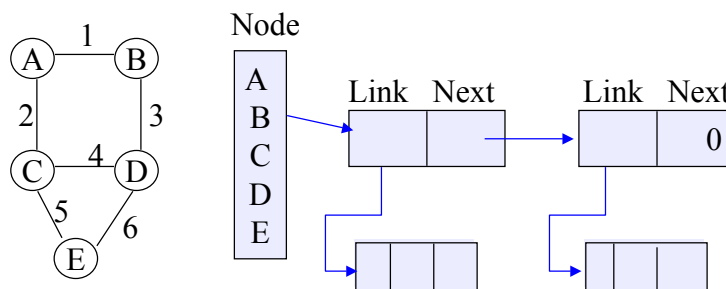


For an undirected graph, the adjacency matrix is symmetric.

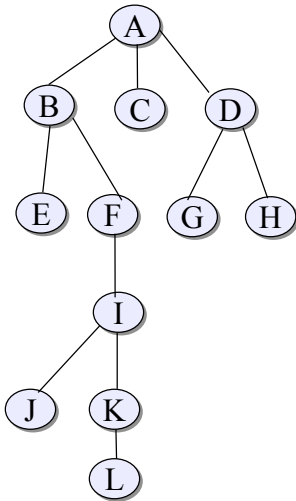
For directed graphs, +1 is source and -1 is sink of an arc

· Adjacency matrix – Naapuruusmatriisi · Incidence matrix – Liitänämatriisi

For graph algorithms linked list presentation of adjacency is convenient



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)
  
```

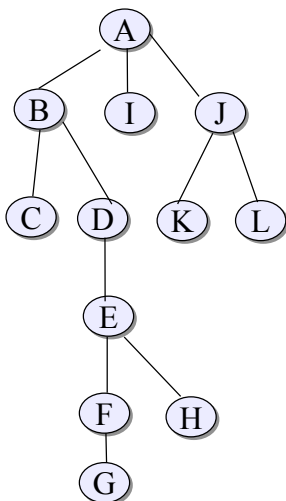
Works for directed links

· Breadth-first-search
- leveyshaku

S-38.121 S-02 / RKa, NB

Multicast1-21

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 -
syvvyshaku

S-38.121 S-02 / RKa, NB

Multicast1-22

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 +=1
      Dfs (node, n_adj_list)
```

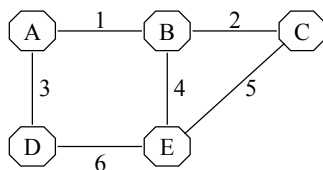
Minimum spanning tree (MST) is the spanning tree with minimum cost

- 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/cost, thus we talk about MST.
- If the graph is not connected, we may look for a minimum spanning forest.

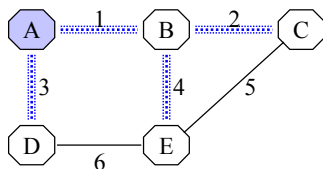
$$n = c + e$$

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 (\in ST) per link
- 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 MST 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)

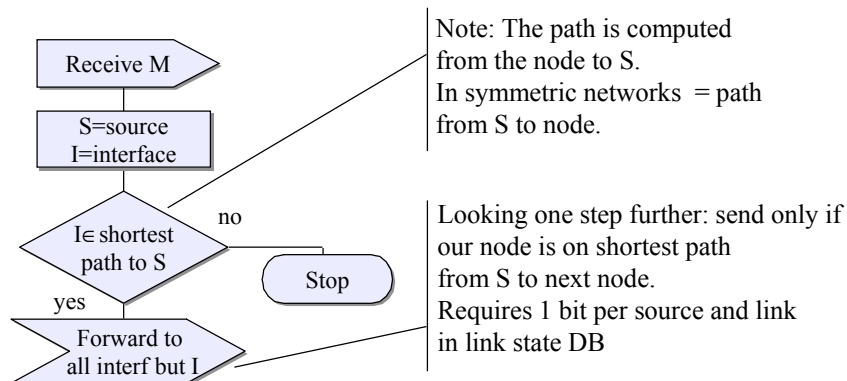
solution ← ∅
If (candidate_set ← ElementsOf (properties)) is not Empty
    GreedyLoop (*candidate_set, *solution)
return(solution)
    
```

S-38.121 S-02 / RKa, NB

Multicast1-27

Reverse-Path Forwarding computes an implicit spanning tree per source – OK for dense trees

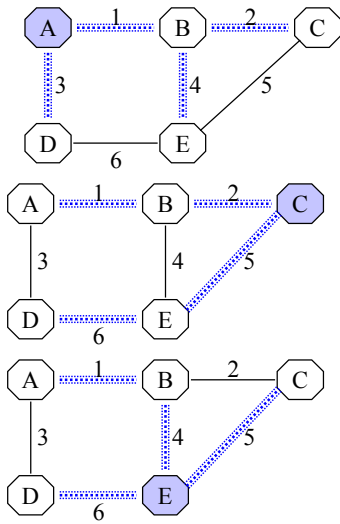
- RPF was first used in MBone



S-38.121 S-02 / RKa, NB

Multicast1-28

Reverse path forwarding properties

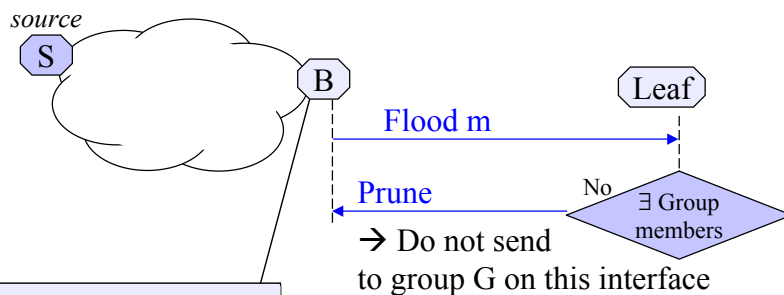


S-38.121 S-02 / RKa, NB

Multicast1-29

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

“Flood and prune” introduces dynamic group membership



If prunes on all interfaces, forward prune up the RPF tree

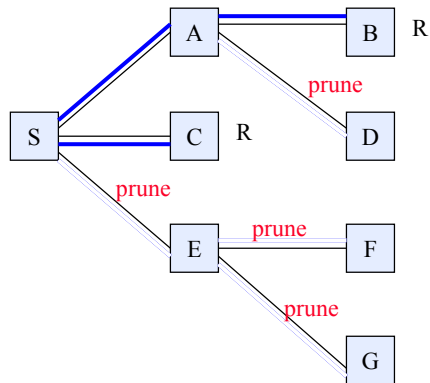
Drawbacks:

- first packet is flooded to the whole net
- nodes must keep state per S and G.
- state is transient (timed out)

S-38.121 S-02 / RKa, NB

Multicast1-30

”Flood and prune” – example

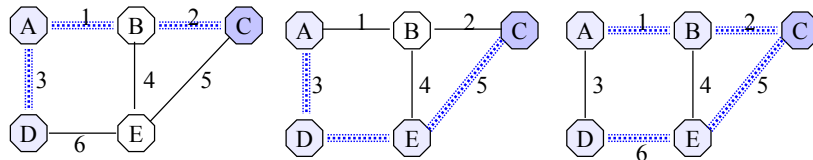


S-38.121 S-02 / RKa, NB

Multicast1-31

Steiner tree spans the group with the minimal 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
 - Algorithm is monolithic, can't be distributed
 - 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)

S-38.121 S-02 / RKa, NB

Multicast1-32

Center-based trees

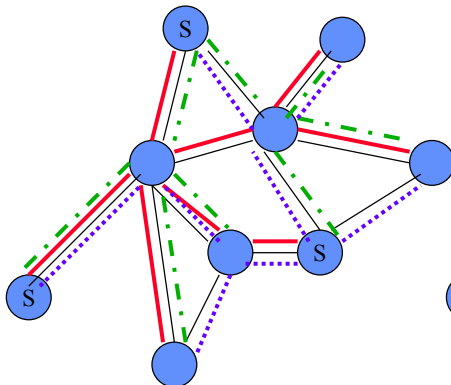
- Choose a center
- The recipients send join commands toward the center
 - Each router on the path toward the center processes the join message and adds the interface on which the join message is received to the forwarding table for the group. The join message continues to the next router toward the 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.
- 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

S-38.121 S-02 / RKa, NB

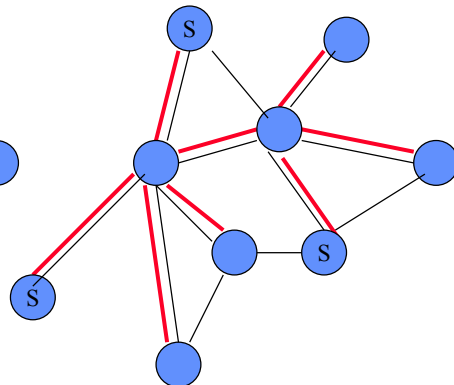
Multicast1-33

Shared trees and source based trees

Source based trees



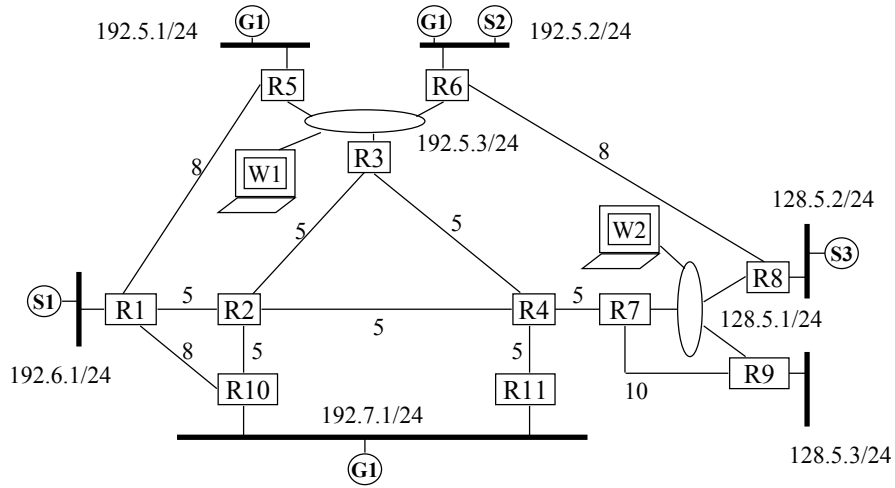
Shared tree



S-38.121 S-02 / RKa, NB

Multicast1-34

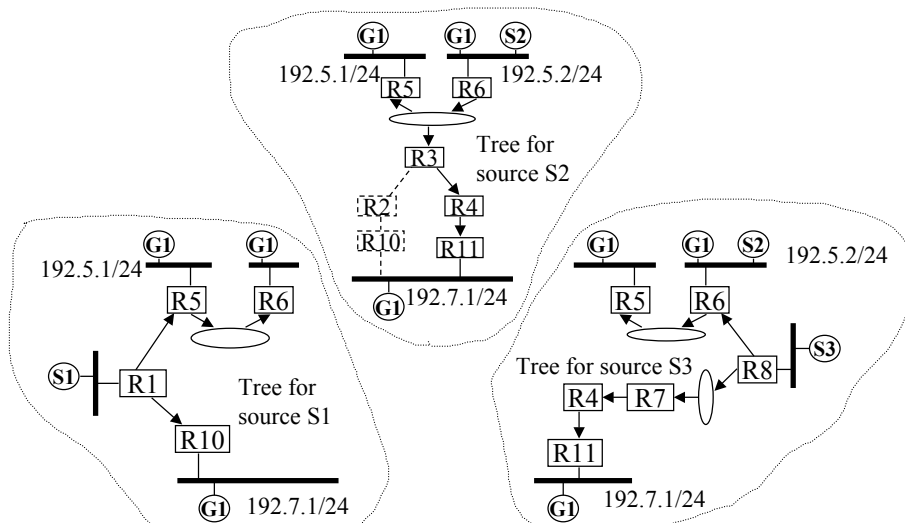
Multicast routing example



S-38.121 S-02 / RKa, NB

Multicast1-35

Source based trees for G1



S-38.121 S-02 / RKa, NB

Multicast1-36

Shared tree for G1

