Multicast routing principles in Internet

Motivation
Recap on graphs
Principles

Multicast capability has been and is under intensive development in 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
  – network load minimization by replacing many pt-to-pt transmissions
  – multimedia conferencing
Multicast addresses

32 bits

<table>
<thead>
<tr>
<th>Network</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110</td>
<td>28 bits - multicast group address</td>
</tr>
<tr>
<td>1111</td>
<td>experiments</td>
</tr>
</tbody>
</table>

Class

- **D**: Experiments
- **E**: All systems

224.0.0.0 - 239.255.255.255

- **224.0.0.1**: All systems
- **224.0.0.2**: All routers
- **224.0.0.4**: All DVMRP routers
- **224.0.0.1 - 224.0.0.255**: Local segment usage only
- **239.0.0.0 - 239.255.255.255**: Admin scoped multicast (local significance)
- **239.192.0.0 - 239.195.255.255**: Organisation local scope

Ethernet MAC address: MACprefix+G -> no lookup, no ARP

Note: + Sender does not need to belong to G.
+ Address space is flat!

Resource discovery by MC simplifies network management

- OSPF router
- RIP router

- OSPF hello [all ospf routers]
- RIP response

- No need for lists of neighbors, just use std MC address
- How to find corporate DNS - server --> MC to all nodes in corporate network.
- Network is easily flooded with messages.
- TTL can be used for Broadcast scope limitation
  -> find nearest DNS or whatever
  -- when TTL=0, router does not return ICMP msg!
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

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
Flooding is the simplest MC algorithm

Flooding algorithm is

1. Receive M from L
2. Search M in DB
3. If found:
   - Accept M
   - Send M to all links but L
4. If DB older:
   - Update DB
   - Send M to all links but L
5. If DB(I) = M:
   - Build m from DB
   - Send M to sender on L
6. Stop

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

• Examples: OSPF, usenet news...

Alternative to DB in flooding is trace info in the message

• Trace info in Message lists all passed nodes
• Avoids a costly DB reads but may accept same M several times.
• If neighbor is in trace, does not send

Flooding guarantees that node will not forward the same packet twice. It does not guarantee that node will receive same packet only once! --> Greedy algorithm.
+++Does not depend on routing tables -- > robust
Networks are modeled as Graphs.

\[ G = (V, E), \]
\[ V - \text{set of vertices or nodes (non-empty, finite set)} \]
\[ E = \{ e_j \mid j = 1, 2, \ldots, M \} - \text{set of edges or links} \]

\[ e_j = (v_i, v_k) = (i, k) \]

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

Links are bi-, arcs are unidirectional

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.
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, path is called a cycle. If an intermediate node appears no more than once, it is a simple cycle.

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.
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 component. An arc is part of *at most one* strongly connected component.

A **tree** is a graph without cycles

- Given a Graph $G = (V, E)$, $H$ is a subgraph of $G$ if $H = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$
- A **spanning tree** is a connected graph without cycles.
- If graph is not necessarily connected, we talk about a **forest**.
Spanning trees model minimally connected networks

- ST 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**.
Suomalaiset graafitermit

Vertex, node - kärki, solmu
Edge, link - syrjä, linkki, sivu, kaari, haara
Adjacent - viereinen
Neighbor - naapuri
Degree of a node - solmun aste(?)
Arc - kaari
Cycle, Loop - silmukka
Path - polku
Directed path - suunnattu polku
Connected - yhteydellinen, yhdistetty
Strongly connected

Subgraph - aligraafi
Tree - puu
Spanning tree - virittäjäpuu
Forest - metsä
Disconnecting set - erotusjoukko
Cut - leikkaus
XY-cutset - XY-leikkaus-joukko

Adjacency and Incidence Matrices are used to present Graphs

Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Node 1 2 3 4 5 6

Link

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Incidence Matrix

For an undirected graph
Adjacency matrix is symmetric.

For directed graphs +1 is source and -1 is sink of an arc
For graph algorithms linked list presentation of adjacency is convenient

A Tree can be traversed by Breadth-first-search

Works for directed links

```plaintext
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)
```
A Tree can also be traversed by Depth-first-search

An undirected graph can 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)
```

```
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 )
```

Works for directed links
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 is the ST 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.
MC 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 STlinks
- 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**

```c
List <- Greedy( properties )
dcl properties [list, list],
    candidate_set[3], solution[3]
void <- GreedyLoop( *candidate_set, *solution)
dcl test_set[3], candidate_set[3], solution[3]
    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 <- \emptyset
If (candidate_set <- ElementsOf(properties)) is not Empty
    GreedyLoop( *candidate_set, *solution)
return(solution)
```
Reverse-Path Forwarding computes an implicit spanning tree per source, is OK for dense trees

- RPF was first used in MBone

Receive M
S=Source
I=Interface

If shortest path to S
yes
Forward to all interf but I

no
Stop

Note: path is computed from S to node. In symmetric networks = path from node to S

Looking one step further: send only if our Node is on shortest path from S to next Node. Requires 1 bit/source and /link in link state DB

Reverse path forwarding properties

- No group membership but can be scoped by TTL
- Guarantees fastest possible delivery since uses shortest paths only
- Different tree for each source --> traffic is spread over multiple links leading to better network utilization
“Flood and prune” introduces dynamic group membership

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

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 and result is not necessarily optimal
  - 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)