Compact Routing

Algorithms

Christos Litsas

June 21, 2011

Outline

The Problem

Algorithms

The Problem

Given a network, represented as a graph we ask how to deliver one or more messages between its nodes.
A mechanism is required that is able to deliver packages of data from any node of the network to any other node.

What We Need...

- local routing tables
- routing strategy
  - A global preprocessing algorithm, which initializes the local data structures of all nodes and which-if this is allowed by the model of the network - assigns labels to the nodes and port numbers to the edges (otherwise, the node labels and port numbers, respectively, are part of the input graph and cannot be changed), and
  - a distributed algorithm, called the routing scheme, which implements an adequate routing function.

A Dummy Algorithm

Preprocessing phase:

- Calculate all shortest paths between all pairs of nodes.
- At each node v of the network a routing table is stored which contains for each node w of the network the port number of the edge leading from v to the next node on the shortest path from v to w.

Routing:

- If a data package whose header contains the label of a node w as destination address arrives at a node v, the routing algorithm at v searches in the local routing table of v for the entry belonging to w and sends the package through the edge determined by the port number found in the table.

Performance Measurements

- the memory space needed in each node (called local memory).
- the total memory space used by all nodes (called total memory),
- the stretch,
- the sizes of the addresses and package headers,
- the time needed to compute the routing function (called routing time or latency),
- the time needed for the preprocessing.

Universal Routing Strategies

- Labeled Routing
- Name-Independent Routing.

An Interval Routing Scheme for Trees

Preprocessing the tree is rooted at an arbitrary node, and the nodes are labeled via a depth first search (DFS).

Node Data

- its address $a(v)$
- the highest address occurring in the subtree rooted at v, denoted with f_v ,
- the port number of the edge leading to the parent of v, and
- a table with one entry (a_i, p_i) for each child v_i of v, (a_i: highest node address occurring in the subtree rooted at v_i and p_i is the port number $p(v, v_i)$ of the edge leading from v to v_i).

An Interval Routing Scheme for Trees

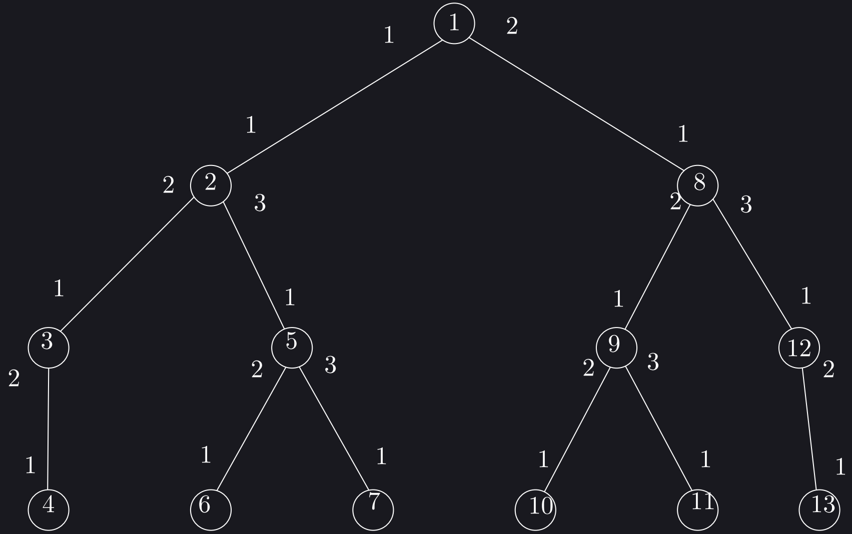

Figure: A network.

An Interval Routing Scheme for Trees

1. If $a(w) = a(v)$: The package has reached its destination. *Stop*.
2. If $a(w) < a(v)$ or $a(w) > f_v$: The destination is not a descendant of v. Send the package to the parent of v and *Stop*.
3. Otherwise, the destination node lies in a subtree rooted at a child of v. Search in the local memory the entry (a_i, p_i) with the smallest $a_i \geq a(w)$ and send the package through the port numbered with p_i.

An Improved Labeled Routing Scheme for Trees

The tree nodes are partitioned (by a DFS) into *heavy* and *light* nodes

Preprocessing the tree is rooted at an arbitrary node, and the nodes are *indexed* via a depth first search (DFS).

Node Data

- its index dfs_v,
- the highest address occurring in the subtree rooted at v, denoted with f_v,
- if v has a heavy child, the index of the heavy child, denoted with h_v; otherwise $h_v = f_v + 1$,
- the number of light nodes (including v itself if v is light) lying on the path from the root to v, called the light level of v and denoted with ℓv,
- the port number of the edge to the parent of v, denoted with $P_v[0]$, and
- the port number of the edge leading to the heavy child of v, denoted with $P_v[1]$ (if v has no heavy child, $P_v[1]$ contains an arbitrary entry).

An Improved Labeled Routing Scheme for Trees

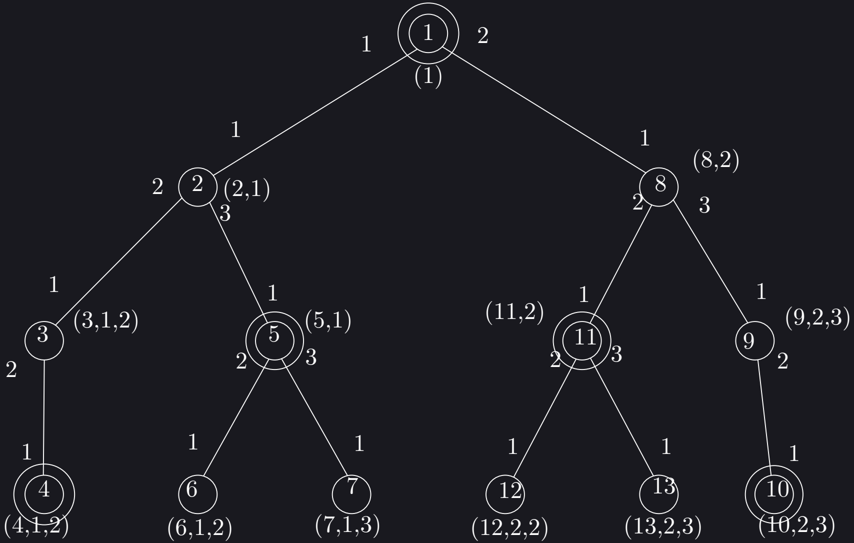

Figure: A network.

An Improved Labeled Routing Scheme for Trees

1. If $dfs_w = dfs_v$: The package has reached its destination. *Stop*.
2. If $dfs_w < dfs_v$ or $dfs_w > f_v$: The destination is not a descendant of v. Send the package through the edge labeled with $P_v[0]$ to the parent of v and stop.
3. If $dfs_w \geq h_v$: The destination is a node in the subtree rooted at the heavy child of v (because the heavy child is the last child visited by the DFS). Send the package through the edge labeled with $P_v[1]$ to the heavy child of v and *Stop*. 4. Otherwise, the destination must be a node in a subtree rooted at a light child of v. Send the package through the edge labeled with L_{w,ℓ_v+1} to the light child of v who lies on the path from v to the destination node (the port number L_{w,ℓ_v+1} can be extracted from the destination address).