Advanced routing topics

Tuomas Launiainen

Suboptimal routing

- Routing trees
- Measurement of routing trees

Adaptive routing

- Problems
- Fault-tolerant tables
- Point-of-failure rerouting
- Point-of-failure shortest path rerouting
- Correctness

Compact tables

- Routing with interwals

Restrictions used:

- *Bidirectional Links* (BL)
- *Connectivity* (CN)
- *Total Reliability* (TR)
- *Initial Distinct Values* (ID)

Suboptimal routing

- Optimal (shortest path guaranteed) routing is expensive
- Suboptimal routing does not guarantee shortest paths, but is often sufficient

Routing trees

Routing can be done using a single spanning tree, a *routing tree*. All messages are passed using only the edges in the routing tree.

- Relatively easy to construct
- Guaranteed delivery
- Guaranteed to use no more messages than the diameter of the tree

Centre-based routing tree

Since messages are delivered with no more than $diam(T)$ hops in a routing tree T, one logical choise for the routing tree is one rooted at the *centre* of the graph (a node that has the smallest distance to the farthest node from it).

Centre-based routing tree

Since messages are delivered with no more than $diam(T)$ hops in a routing tree T, one logical choise for the routing tree is one rooted at the *centre* of the graph (a node that has the smallest distance to the farthest node from it).

Construction:

1. Find the centre of the graph
2. Construct the shortest path spanning tree for that node

Centre-based routing tree

Since messages are delivered with no more than $diam(T)$ hops in a routing tree T, one logical choise for the routing tree is one rooted at the *centre* of the graph (a node that has the smallest distance to the farthest node from it).

Construction:

1. Find the centre of the graph
2. Construct the shortest path spanning tree for that node

The diameter of the spanning tree is bound from above: $diam(G) \leq diam(\mathrm{PT}(c)) \leq 2diam(G)$.

Median-based routing tree

Since a tree has no loops, each edge $e = (x, y)$ of the routing tree splits the tree in two: $T[x - y]$, and $T[y - x]$. This means that every message passing from one half to the other must go through e, the use of which costs $\theta(e)$.

Median-based routing tree

Since a tree has no loops, each edge $e = (x, y)$ of the routing tree splits the tree in two: $T[x - y]$, and $T[y - x]$. This means that every message passing from one half to the other must go through e, the use of which costs $\theta(e)$.

If all nodes send the same amount of messages on average, and the destinations are evenly distributed independent of the sender, the overall average cost of using T for routing is relative to:

$$\sum_{(x,y)\in T} |T[x - y]|\,|T[y - x]|\,\theta((x, y))$$

Median-based routing tree

Since a tree has no loops, each edge $e = (x, y)$ of the routing tree splits the tree in two: $T[x - y]$, and $T[y - x]$. This means that every message passing from one half to the other must go through e, the use of which costs $\theta(e)$.

If all nodes send the same amount of messages on average, and the destinations are evenly distributed independent of the sender, the overall average cost of using T for routing is relative to:

$$\sum_{(x,y)\in T} |T[x - y]|\,|T[y - x]|\,\theta((x, y))$$

This is also the sum of all distances between every pair of nodes.

Median-based routing tree (contd.)

If message passing is evenly distributed, the overall cost of a routing tree can be minimized by minimizing the sum of every distance in the tree. Unfortunately constructing such a tree is difficult.

Median-based routing tree (contd.)

If message passing is evenly distributed, the overall cost of a routing tree can be minimized by minimizing the sum of every distance in the tree. Unfortunately constructing such a tree is difficult.

A near-optimal solution, a median-based routing tree, is simple to construct, however. The *median* node of a graph is one that has the smallest sum of distances to every other node.

Median-based routing tree (contd.)

If message passing is evenly distributed, the overall cost of a routing tree can be minimized by minimizing the sum of every distance in the tree. Unfortunately constructing such a tree is difficult.

A near-optimal solution, a median-based routing tree, is simple to construct, however. The *median* node of a graph is one that has the smallest sum of distances to every other node.

The average cost of a median-based routing tree is (claimed to be) no worse than twice the average cost of the cost-minimizing routing tree.

Minimum-cost routing tree

Another natural choise is the spanning tree that minimizes the sum of costs of it's edges. It can be constructed with e.g. MegaMerger.

Measurement of routing trees

Examine how much a routing tree *stretches* the distance between two nodes.

- *Strecth factor*: the maximum streching between nodes: $\max_{x,y \in V} \frac{d_T(x,y)}{d_G(x,y)}$
- *Dilation*: the maximum length between neighbours in the original graph: $\max_{(x,y) \in E} d_T(x,y)$
- *Edge-stretch factor*: maximum stretch of an edge: $\max_{(x,y) \in E} \frac{d_T(x,y)}{\theta((x,y))}$ (also called the *dilation factor*)
- Also *average stretch factor* and *average dilation factor*

Adaptive routing

Adaptive routing tries to handle routing in a system where the costs of edges change.

- When the cost of a link (x, y) changes, both x and y are notified
- The restriction *Total Reliability* is replaced with *Total Component Reliability*
- New restriction *Cost Change Detection*
- A link failure can be described by setting it's cost to ∞

Map Update

A naïve approach:

- Every node contains a complete map of the network
- When a node detects one of it's links has changed, it updates it's map and sends an *update*-message to all it's neighbours
- When a node receives an *update*-message, it updates it's map and passes the message to it's other neighbours
- Periodic updates can be used even if no changes occur

This is very expensive, even more so with periodic messages. It can, however, handle any amount and type of changes. E.g. the second Internet routing protocol uses this scheme.

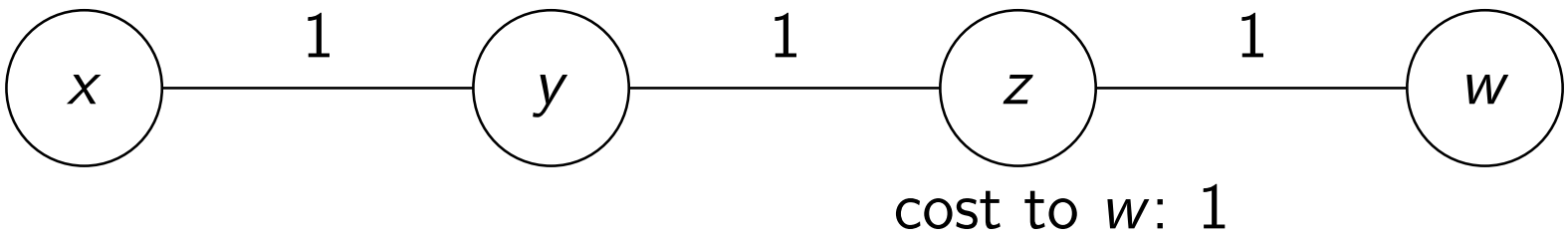
Vector Update

- Construct initial routing tables with *Iterative Construction*
- When a node detects a one of it's links has changed, it initiates a global update of routing tables, either by starting a new execution of Iterative Construction, or a new round of iterations, until the tables converge again

Starting a new round of iterations is preferable, since starting from scratch is expensive. This scheme was used in the first Internet routing protocol, but it has problems.

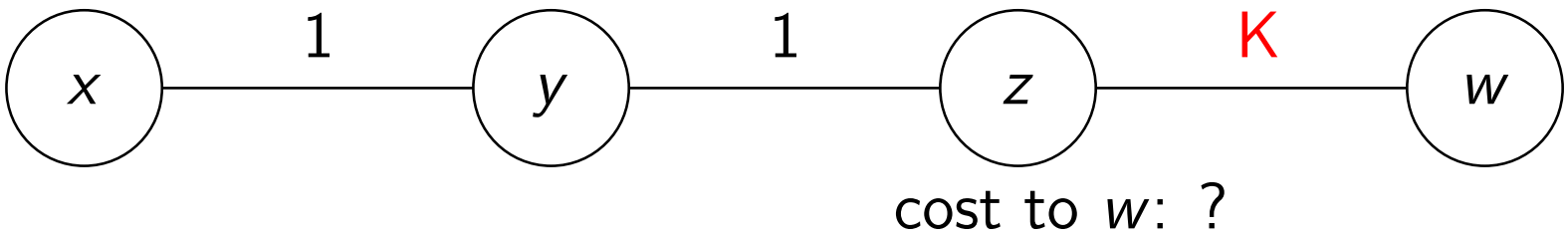
Vector Update (contd.)

Count-to-infinity problem



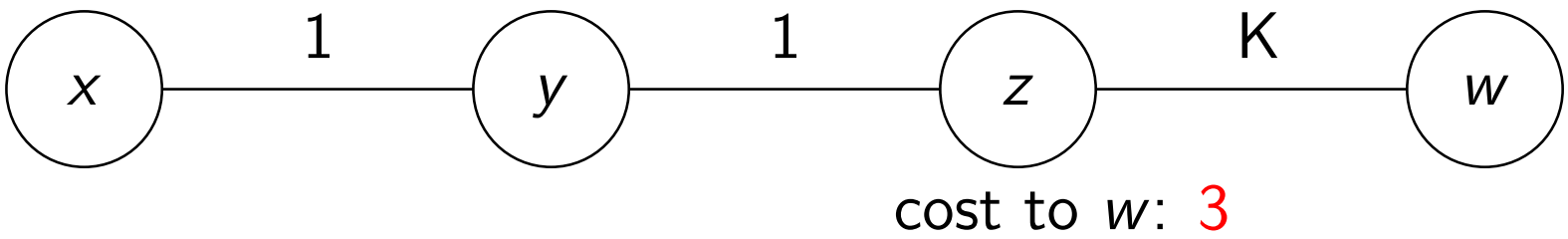
Vector Update (contd.)

Count-to-infinity problem



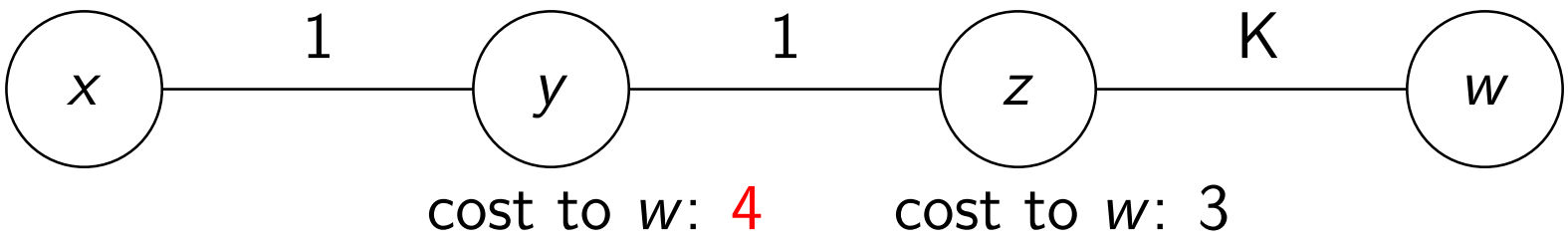
Vector Update (contd.)

Count-to-infinity problem



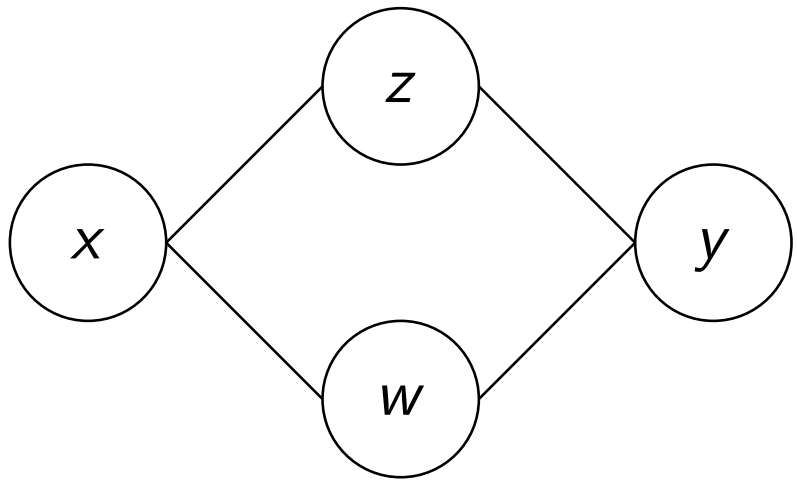
Vector Update (contd.)

Count-to-infinity problem



Oscillation

A problem concerning many approaches is *oscillation*. It occurs when the cost of using a link is proportional to the amount of traffic through it.



If x wants to send lots of messages to y, the best path will start to oscillate between z and w.

Fault-tolerant tables

Upholding optimal routing tables in a changing system is very expensive. If suboptimal routes are allowed and link failures are limited to a single link at a time (*single-link crash failure*), *fault-tolerant tables* can be used to relay messages with minimal communication.

Point-of-failure rerouting

- Each node stores two edge-disjoint paths to each destination.
- Messages are delivered through the shortest path from their source to their destination, assuming no link crashes have occurred.
- If the message arrives to a node whose link has crashed (point of failure), it is rerouted to it's destination via the alternate path.

Suboptimal service is provided only when a crash occurs, and information about crashes does not need to be relayed.

Point-of-failure shortest path rerouting

Swap edges

- In systems with bidirectional links, the shortest path from the source node s to the target node t is contained in the shortest path spanning tree rooted at t, PT(t). Each node $x, x \neq t$ in this tree has a parent $p_t(x)$.
- When a link $e_t[x] = (x, p_t(x))$ in PT(t) fails, it disconnects PT(t) into two subtrees.
- For each link $e_t[x]$ in PT(t), there exists a link e not in PT(t) that can reconnect the spanning tree if $e_t[x]$ fails. e is called the *swap edge* of $e_t[x]$.
- The best swap edge is called the *optimal swap*.

Point-of-failure shortest path rerouting (contd.)

Routing tables

To provide point-of-failure shortest path rerouting, each node needs to know the optimal swap in addition to the normal link for each destination. It also needs to know the shortest path to the optimal swap. The routing table of x thus has this row for each destination t, where (u, v) is the optimal swap for $(x, p_t(x))$:

Final Destination	Normal Link	Rerouting Link	Swap Destination	Swap Link
t	$(x, p_t(x))$	$(x, p_u(x))$	u	(u, v)

When a message reaches a point-of-failure node, the node then sends the message towards the optimal swap.

Point-of-failure shortest path rerouting (contd.)

Algorithm (sort of)

All messages contain the following fields: final destination, swap destination, swap link, and swap bit.

1. s sends a message to t: it sets the final destination to t, the swap destination and swap link to empty, and the swap bit to 0.
2. An intermediate node x receives a message to t:

Point-of-failure shortest path rerouting (contd.)

Algorithm (sort of)

All messages contain the following fields: final destination, swap destination, swap link, and swap bit.

1. s sends a message to t: it sets the final destination to t, the swap destination and swap link to empty, and the swap bit to 0.
2. An intermediate node x receives a message to t:
   (a) If the swap bit is 0 and the normal link $(x, p_t(x))$ is up, use that.

Point-of-failure shortest path rerouting (contd.)

Algorithm (sort of)

All messages contain the following fields: final destination, swap destination, swap link, and swap bit.

1. s sends a message to t: it sets the final destination to t, the swap destination and swap link to empty, and the swap bit to 0.
2. An intermediate node x receives a message to t:
   (a) If the swap bit is 0 and the normal link $(x, p_t(x))$ is up, use that.
   (b) If the swap bit is 0 and the normal link $(x, p_t(x))$ is down, set the swap link and swap destination according to the routing table, set the swap bit to 1, and send the message to the rerouting link.

Point-of-failure shortest path rerouting (contd.)

Algorithm (sort of)

All messages contain the following fields: final destination, swap destination, swap link, and swap bit.

1. s sends a message to t: it sets the final destination to t, the swap destination and swap link to empty, and the swap bit to 0.
2. An intermediate node x receives a message to t:
   (a) If the swap bit is 0 and the normal link $(x, p_t(x))$ is up, use that.
   (b) If the swap bit is 0 and the normal link $(x, p_t(x))$ is down, set the swap link and swap destination according to the routing table, set the swap bit to 1, and send the message to the rerouting link.
   (c) If the swap bit is 1 and $x = u$, set the swap bit to 0 and use the swap link in the message.

Point-of-failure shortest path rerouting (contd.)

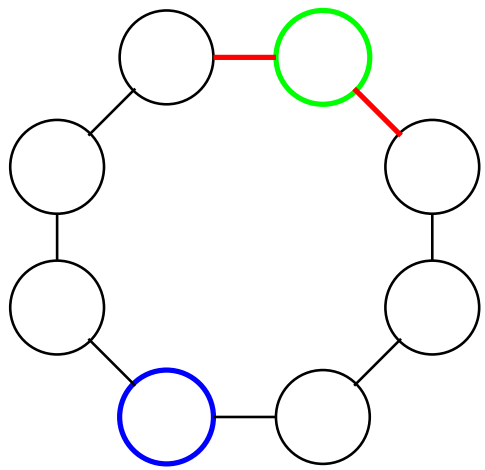
Algorithm (sort of)

All messages contain the following fields: final destination, swap destination, swap link, and swap bit.

1. s sends a message to t: it sets the final destination to t, the swap destination and swap link to empty, and the swap bit to 0.
2. An intermediate node x receives a message to t:
   (a) If the swap bit is 0 and the normal link $(x, p_t(x))$ is up, use that.
   (b) If the swap bit is 0 and the normal link $(x, p_t(x))$ is down, set the swap link and swap destination according to the routing table, set the swap bit to 1, and send the message to the rerouting link.
   (c) If the swap bit is 1 and $x = u$, set the swap bit to 0 and use the swap link in the message.
   (d) If the swap bit is 1 and $x \neq u$, send the message towards u using the routing table.

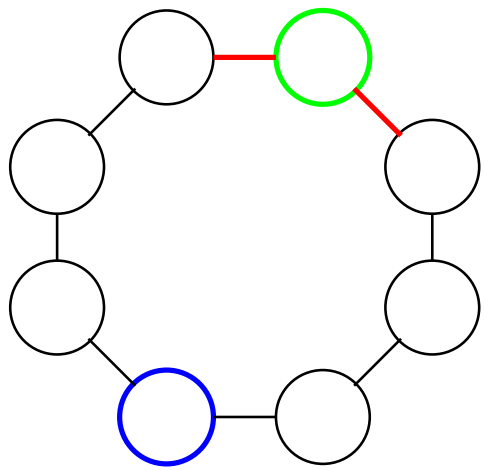
Correctness

Consider this example, where the blue node wants to send a message to the green node, but just before the message reaches it, the red link before it goes down.



Correctness

Consider this example, where the blue node wants to send a message to the green node, but just before the message reaches it, the red link before it goes down.



We cannot make any guarantees that a message will arrive to it's destination. We can, however, guarantee that *if* the changes in the system stop (for a long enough period), the message will be delivered through the point-of-failure shortest path.

Compact tables

Routing tables are quite large:

- $O(n^2 \log n)$ bits each for full routing tables
- $O(n \log n)$ bits each, if only destination-link pairs are stored (for each destination t, the routing table of x stores the link $(x, p_t(x))$)

Without special knowledge about node naming and network topology, this is about as good as we can get, since an entry needs to be made for each destination.

Routing with interwals

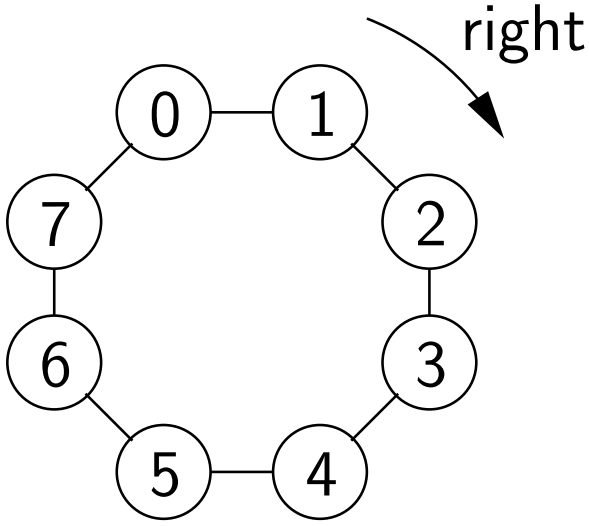
- We can assume that all node names are integers between 0 and $n-1$, inclusive.
- For two integers j and k, $0 \leq j, k \leq n-1$, an *inteval* is a sequence:

$$\begin{cases} \langle j, j+1, \ldots, k \rangle, & \text{if } j \leq k \\ \langle j, j+1, \ldots, n-1, 0, 1, \ldots, k \rangle & \text{if } j > k \end{cases}$$

- Instead of storing destination-link pairs, a routing table can store interwal-link pairs.
- Routing tables can be as small as $O(\log n)$.
- The ability to choose node names and knowledge about network topology is needed.

Routing with interwals (contd.)

Example



The routing table of 0	
Link	Interwal
left	$(1, 4) = \langle 1, 2, 3, 4 \rangle$
right	$(5, 7) = \langle 5, 6, 7 \rangle$

In sorted and directed rings $O(\log n)$ can be achieved.

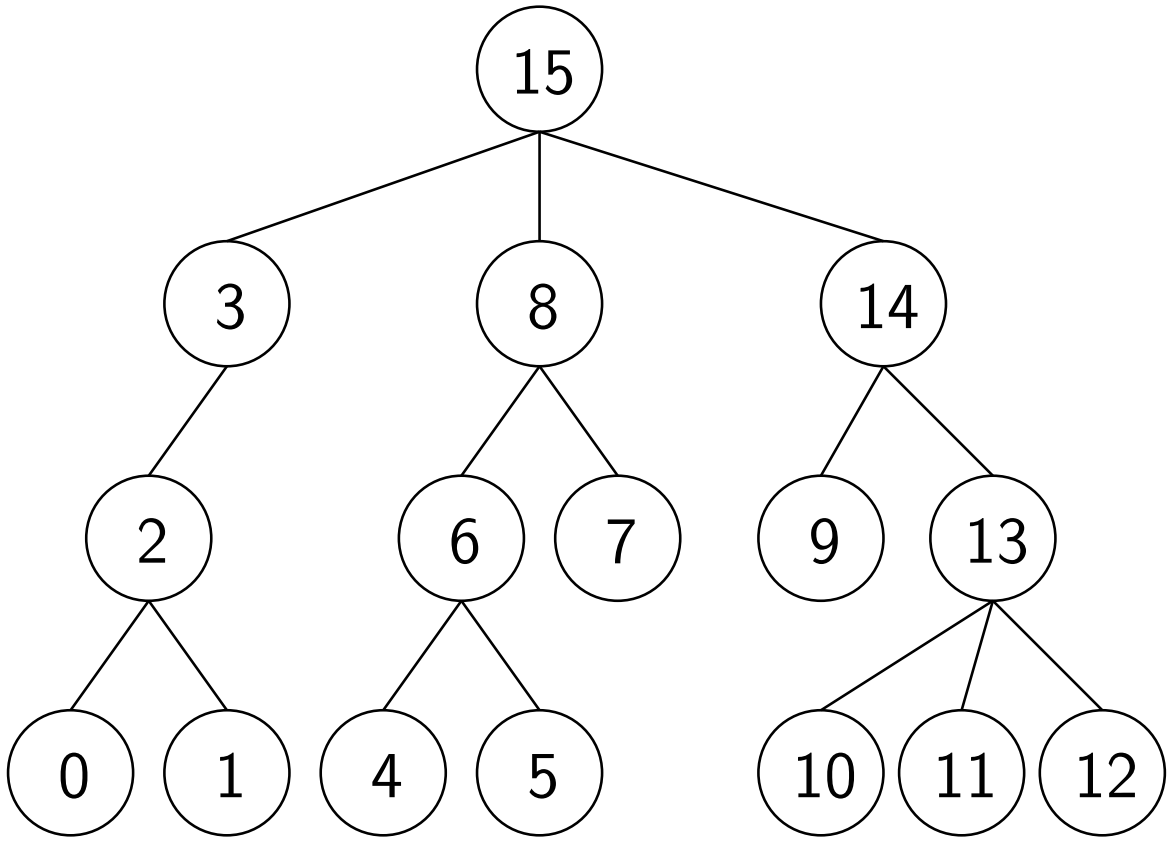
Routing with interwals (contd.)
Tree networks

Interwal routing can always be done in tree networks by renaming the nodes by *post-order traversal*.

$\Rightarrow$

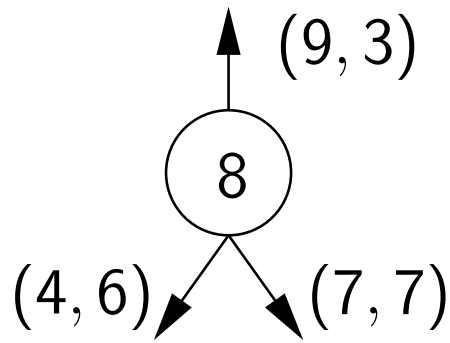
- Every node has a higher number than any of it's children
- Every subtree contains only consecutive integers

Routing with interwals (contd.)
Tree example



An example of a 16-node tree network with nodes named by post-order traversal.

Routing with interwals (contd.)
Tree example (contd.)

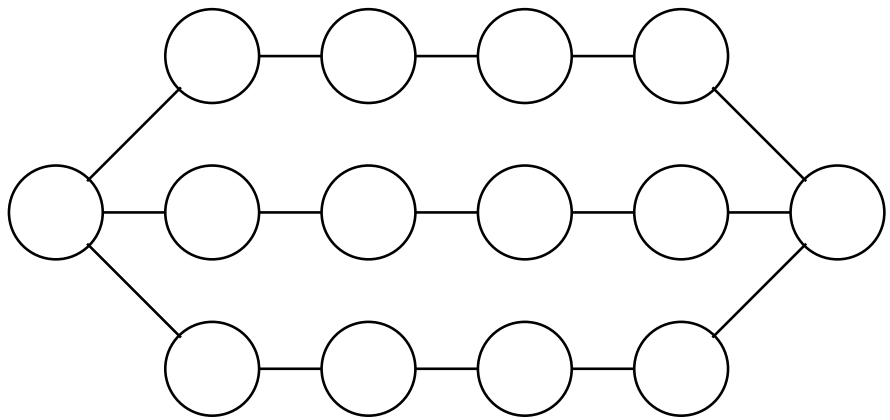


The routing table of 8

Link	Interwal
parent	$(9,3) = \langle 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3\rangle$
left	$(4,6) = \langle 4, 5, 6\rangle$
right	$(7,7) = \langle 7\rangle$

Routing with interwals (contd.)
Globe graph

Interwal routing is not always possible, however. Here is an example of such a case, called a globe graph:



Suboptimal routing with interwals

If providing shortest path routing is not necessary, for any network a routing tree can be constructed. As already seen, interwal routing always works in trees.

Recap

- Optimal routing is expensive, but suboptimal, efficient, and often good enough solutions exist. The primary method covered here uses some spanning tree to route traffic.

Recap

- Optimal routing is expensive, but suboptimal, efficient, and often good enough solutions exist. The primary method covered here uses some spanning tree to route traffic.
- Adaptive routing tries to cope with changing costs of links. Optimal solutions are very expensive, but e.g. point-of-failure rerouting uses no extra communication after the initial construction, and manages single-link crash failures. Message delivery cannot be guaranteed while crashes continue to occur, however.

Recap

- Optimal routing is expensive, but suboptimal, efficient, and often good enough solutions exist. The primary method covered here uses some spanning tree to route traffic.
- Adaptive routing tries to cope with changing costs of links. Optimal solutions are very expensive, but e.g. point-of-failure rerouting uses no extra communication after the initial construction, and manages single-link crash failures. Message delivery cannot be guaranteed while crashes continue to occur, however.
- Complete routing tables are very large. The only way to get below $O(n \log n)$ size per table is to use special knowledge about the network.