Routing algorithms

Jan Lönnberg, 51101M

October 2, 2002

Based on G. Tel: *Introduction to Distributed Algorithms*, chapter 4.

Contents

- Introduction
- Destination-based routing
- Floyd-Warshall (single processor)
- Toueg (distributed)
- Other algorithms
- Netchange algorithm
- Conclusion

Introduction

- Purpose of routing
- Desirable properties of routing
- Graph representation

Purpose of routing

- In many applications, there are separate *nodes* of some sort that wish to communicate with each other using communications *channels* of some sort.
- Example applications:
  - Telecommunications networks, e.g.:
    - POTS/PSTN
    - Mobile phone networks
    - Internet
    - Local area networks
  - Parallel (multiprocessor) computers
  - Processes in distributed applications

Purpose of routing

- However, not all nodes are usually connected to each other, as connecting all nodes directly to each other involves lots of wires/cables and/or high-powered transceivers.
- Nodes in network must *forward* other nodes' transmissions to correct destination.
- The process of determining where to forward packets and actually doing so is called *routing*.

Desirable properties of routing

- All packets should reach their destination (unless prevented by other factors, e.g. congestion).
- Data transfer should be as quick and efficient as possible. Using the shortest or fastest route helps achieve this.
- Routing computations should be as quick and easy as possible.
- The algorithm should adapt to:
  - Topology changes (new or removed channels).
  - Changing load.
- The algorithm should treat different users fairly.

Graph representation

- The network is represented as a graph.
- Each node is represented as a vertex.
- Each channel or connection between two nodes is represented as an edge.
- The weight of each edge is defined as the cost of using the edge in a transmission path. This cost typically approximates the delay imposed on a message sent through the channel.

Graph representation

- Typical weight functions:
  - Minimum hops (all edges have the same weight).
  - Shortest path (every edge has a constant non-negative weight).
  - Minimum delay (every edge has a non-negative weight that depends on the traffic in the channel).
- In realistic networks, all weights are positive; nodes connected with zero-weight channels can be considered a single node.
- The cost of sending a message along a route in the network is defined as the sum of the weights of the edges along the corresponding path.

Destination-based routing

- Simple routes
- Optimal sink trees

Simple routes

- For simplicity, we assume that all traffic between two nodes is sent along a single route. Splitting traffic over several routes (*bifurcated routing*) is quite complex.
- For each graph, there is at least one *optimal* or shortest path between every pair of nodes (assuming that the graph does not have negative-weight cycles) that is a *simple* path (has no cycles).
- Proof: For every non-simple path, we can generate a simple path that is shorter or equally long by removing all cycles. As there is only a finite amount of simple paths, at least one of the simple paths must be a shortest path.
- Similarly, if we assume that all cycles have positive length, all optimal paths are simple.

Optimal sink trees

- For each destination node, we can construct an *optimal sink tree*: a tree rooted at the destination and containing only the edges required for optimal paths from every source node to the destination.
- Construction:
  - Start with the destination node as root and nothing else.
  - Repeat until all nodes in the network are in the tree:
    - Take any node v not in the tree and add it to the tree.
    - Add edges in the optimal path (and the nodes they are connected to, if they are not yet in the tree) from v to the root to the tree until a vertex z in the tree is reached.

Optimal sink trees

- The initial sink tree is trivially optimal.
- In the path addition step the optimal path from v to the root is changed by replacing the partial path from z to the root with the existing optimal path in the tree. As this partial path is optimal (induction assumption) it can not be longer than the partial path it replaces. Thus, the resulting path in the tree from v to the root is optimal.
- Similarly, the paths from the other newly added vertices in the tree to the root must be optimal, otherwise the route from v to the root would not be optimal (contradicting the result of the previous step).
- Thus, the routes in the tree from all the newly added vertices to the root are optimal.

Floyd-Warshall

- Basic idea
- Description
- Pseudo-code
- Analysis

Floyd-Warshall — Basic idea

- Calculates shortest path between all pairs of nodes in a graph.
- Basic idea:
  - Start off with paths that consist only of a single edge.
  - Iteratively compute paths with an increasing set of possible intermediate nodes until all nodes can be intermediate nodes.
  - Add one node at a time (the *pivot node*) to the set of possible intermediate nodes.

Floyd-Warshall — Description

- Input: a weighted graph $G = (V, E)$ with edge weights ω_{uv} (for the edge from u to v).
- All cycles in the graph must have positive length.
- As proven earlier, the shortest simple path is optimal, so we only consider paths in which each node can only occur once.
- An S-path is a path in G in which the intermediate nodes belong to $S \subseteq V$.
- If $S' = S \cup \{w\}$, then a simple S'-path from u to v is either an S-path from u to w concatenated with an S-path from w to v or an S-path from u to v (remember, w may only occur once in the path).

Floyd-Warshall — Description

- Thus, a shortest S'-path from u to v is either a shortest S-path from u to v or a shortest path from u to w combined with a shortest path from w to u; whichever is shorter.
- Denoting the length of a shortest S-path from u to v $d^S(u,v)$, $d^{S'}(u,v) = \min(d^S(u,v), d^S(u,w) + d^S(w,v))$.
- As the set of V-paths in G is the same as the set of simple paths in G, the length of the shortest path from u to v is $d^V(u,v)$.

Floyd-Warshall — Description

- The Floyd-Warshall algorithm is a dynamic programming algorithm that calculates the shortest S-path for increasingly large S, adding one vertex at a time, using the relationships described here.
- The algorithm maintains a table $D[u, v]$ that contains the value of $d^S(u, v)$.
- The algorithm also maintains a table $P[u, v]$ containing the next hop on the route from u to v. As all shortest paths are simple, the route contains no cycles and the next hop description is sufficient.
- A sink tree T_v for destination v can be formed using the edges corresponding to the entries in $P[u, v]$ and vertex u for every $u \neq v$ for which $P[u, v] \neq$ **null**.

Floyd-Warshall — Pseudo-code

FLOYDWARSHALL(Graph $G = (V, E)$, Weights ω):

$S \leftarrow \emptyset$

for all $u, v \in V$

if $u = v$

$D[u, v] \leftarrow 0$

$P[u, v] \leftarrow u$

else if $uv \in E$

$D[u, v] \leftarrow \omega_{uv}$

$P[u, v] \leftarrow v$

else

$D[u, v] \leftarrow \infty$

$P[u, v] \leftarrow$ **null**

Floyd-Warshall — Pseudo-code

while $S \neq V$

Choose any $w \in V \setminus S$.

for all $u, v \in V$

if $D[u, v] > (D[u, w] + D[w, v])$

$D[u, v] \leftarrow D[u, w] + D[w, v]$

$P[u, v] \leftarrow P[u, w]$

$S \leftarrow S \cup \{w\}$

return(D,P)

Floyd-Warshall — Analysis

- As the main loop contains three nested loops, each with $|V|$ iterations and loop contents that take $\Theta(1)$ time, the total running time is $\Theta(|V|^3)$ (initialisation is $\Theta(|V|^2)$ using similar arguments).
- To run Floyd-Warshall, all information about the network must be available to a single processor.

Toueg

- Basic idea
- Simple algorithm
  - Description
  - Pseudo-code
- Improved algorithm
  - Description
  - Pseudo-code
  - Analysis

Toueg — Basic idea

- Distributed algorithm based on Floyd-Warshall.
- Each node need initially only be aware of the properties (weights and node at other end) of its direct channels.
- The algorithm ensures that each node gets distances and next hops for the packets it sends and/or forwards to any other node.

Toueg — Simple algorithm — Description

- Floyd-Warshall algorithm with variables split over nodes of the network.
- Every node has information on the shortest route (found so far) to every other node.
- In every step every node requires distance information from the pivot node, so the pivot node broadcasts the information over the entire net.
- The simple algorithm’s major problem is that it requires a broadcast to all nodes in the network before routing tables have been calculated! It also transmits data unnecessarily.

Toueg — Simple algorithm — Pseudo-code

TOUEG(Vertices V, N_u, Weights ω_u) on u:

$S_u \leftarrow \emptyset$

for all $v \in V$

if $u = v$

$D_u[v] \leftarrow 0$

$P_u[v] \leftarrow u$

else if $v \in N_u$

$D_u[v] \leftarrow \omega_{uv}$

$P_u[v] \leftarrow v$

else

$D_u[v] \leftarrow \infty$

$P_u[v] \leftarrow$ **null**

Toueg — Simple algorithm — Pseudo-code

while $S \neq V$

Choose $w \in V \setminus S$ according to predefined order.

if $u = w$

Broadcast D_u.

else

Receive D_w.

for all $v \in V$

if $D_u[v] > (D_u[w] + D_w[v])$

$D_u[v] \leftarrow D_u[w] + D_w[v]$

$P_u[v] \leftarrow P_u[w]$

$S \leftarrow S \cup \{w\}$

return (D_u, P_u)

Toueg — Improved algorithm — Description

- If $D_u[w] = \infty$ in the main loop, the test is always false. Thus, a node u for which $D_u[w] = \infty$ need not receive data from pivot w.
- In other words, w only needs to send routing data to nodes that have a next hop node and are therefore part of its sink tree T_w.
- However, although each node knows its parent in T_w, the parent does not know its children.
- Each node tells it neighbour every iteration whether it is a neighbour or not.

Toueg — Improved algorithm — Pseudo-code

TOUEG(Vertices V, N_u, Weights ω_u) on u:

$S_u \leftarrow \emptyset$

for all $v \in V$

if $u = v$

$D_u[v] \leftarrow 0$

$P_u[v] \leftarrow u$

else if $v \in N_u$

$D_u[v] \leftarrow \omega_{uv}$

$P_u[v] \leftarrow v$

else

$D_u[v] \leftarrow \infty$

$P_u[v] \leftarrow$ **null**

Toueg — Improved algorithm — Pseudo-code

while $S \neq V$

Choose $w \in V \setminus S$ according to predefined order.

for all $x \in N_u$:

if $P_u[w] = x$

Send *child*(w) message to x.

else

Send *notchild*(w) message to x.

Wait for $|N_u|$ *child*/*notchild* messages.

Toueg — Improved algorithm — Pseudo-code

if $D_u[w] < \infty$

  if $u \neq w$

    Receive D_w from $P_u[w]$.

  for all $x \in N_u$:

    if a *child* message was received from x

      Send D_w to x.

  for all $v \in V$

    if $D_u[v] > (D_u[w] + D_w[v])$

    $D_u[v] \leftarrow D_u[w] + D_w[v]$

    $P_u[v] \leftarrow P_u[w]$

  $S \leftarrow S \cup \{w\}$

return (D_u, P_u)

Toueg — Improved algorithm — Analysis

- The main loop is executed $|V|$ times. It contains a loop with $|V|$ iterations (and a few with $O(|V|)$ iterations) and loop contents that take $\Theta(1)$ time. The total running time is $\Theta(|V|^2)$.
- The improved algorithm removes the need for broadcasting.
- Assume that a pair consisting of an edge/path weight and a node name can be sent in W bits.
- Then, the *child* and *nochild* messages are W bits.
- The D_w messages are $|V|W$ bits.
- 2 *child*/*nochild* messages per edge and iteration and one table transfer per vertex at the most per iteration.
- $O(|V|^3W)$ bits total.

Other algorithms

- Alternative view
- Merlin-Segall
- Chandy-Misra
  - Basic idea
  - Pseudo-code
  - Analysis

Alternative view

- Shortest route from u to v trivial if $u = w$.
- Otherwise, the shortest route consists of a channel to a neighbour and a shortest route from that neighbour to v.
- Calculating this way only requires local data and information from neighbours.
- Also, each destination can be calculated separately.
- Formally, if $u \neq v$, $d(u, v) = \min_{w \in N_u}(\omega_{uw} + d(w, v))$.

Merlin-Segall

- Each node maintains an estimated distance and next hop for every destination.
- Every node sends each distance to every neighbour except the next hop node for that destination whenever its distance changes.
- If the sum of a received distance to a node and the weight of the channel to the node advertising this distance is less than the current distance, the next hop is set to the advertising node and the distance to the aforementioned sum.
- Takes $|V|$ update rounds.
- $O(|V|^2|E|W)$ bits total transmitted.
- Cycle-free even when updating.

Chandy-Misra — Basic idea

- Based on *diffusing computation* paradigm; one node starts the calculation and activates other nodes who in turn activate other nodes, and so on.
- Destination v_0 starts by telling its neighbours that it can reach itself at a distance of 0.
- Each node, on receiving information about a shorter route to a destination, updates its routing table (as in previous algorithms) and tells its neighbours.

Chandy-Misra — Pseudo-code

Global variables (for node u):

$D_u[v_0] \leftarrow \infty$

$P_u[v_0] \leftarrow$ **null**

StartChandyMisra(Vertices V, N_u, Weights ω_u) on v_0:

$D_{v_0}[v_0] \leftarrow 0$

for all $w \in N_{v_0}$

Send $(v_0, 0)$ to w.

Chandy-Misra — Pseudo-code

RECEIVEMESSAGE(Sender w, Message (v_0, d), Vertices N_u) on u:

if $d + \omega_{uw} < D_u[v_0]$

$D_u[v_0] \leftarrow d + \omega_{uw}$

$P_u[v_0] \leftarrow w$

for all $x \in N_u$

Send $(v_0, D_u[v_0])$ to x.

Chandy-Misra — Analysis

- Correctness of Chandy-Misra can be shown by considering the behaviour of the algorithm in an optimal sink tree inductively.
- Initially, the root has calculated its distance correctly.
- Induction hypothesis: when a node that is n hops away from the root or less becomes aware of an optimal route to the root, it transmits its distance information to its neighbours, which include its children in the sink tree. This happens after n iterations (transmissions) at the latest.
- Thus, nodes at a distance of $n + 1$ hops from the root will receive information about an optimal route after at the most $n + 1$ iterations.

Chandy-Misra — Analysis

- By induction, all nodes in the optimal sink tree will be aware of optimal routes within $|V|$ iterations, as this is the greatest possible distance.
- Problem: the amount of messages is exponentially bounded.
- If all weights are 1, the shortest path for one destination can be calculated in $O(|V||E|)$ messages of W bits each.

Netchange algorithm

- Assumptions
- Basic idea
- Pseudo-code
- Invariants
- Correctness
- Problems
- Modifications

Netchange algorithm — Assumptions

- The size of the network ($|V|$) is known to all nodes.
- The channels are FIFO.
- Nodes are aware of changes made to channels to which they are connected.
- Minimum hop paths are calculated.

Netchange algorithm — Basic idea

- Each node u keeps track of the estimated distance $D_u[w, v]$ from each neighbour w to every destination v.
- On receiving a message containing a distance from a neighbour to a destination, a node:
  - Updates its copy of the distance from the neighbour to the destination.
  - Recomputes the route to this destination.

Netchange algorithm — Basic idea

- When a channel fails, remove the neighbour from the list of neighbours and recompute all routes.
- When a new channel appears:
  - Add the new neighbour.
  - Set the estimated distance to the neighbour to $|V|$.
  - Send the entire distance table to the new neighbour.

Netchange algorithm — Basic idea

- Recomputation of route to destination v:
  - The distance from a node to itself is always 0.
  - For all other destinations, find the neighbour w with the least estimated distance d to the destination and add 1 to this distance.
  - If $d + 1 < |V|$, update the distance entry for the destination and change the next hop for the destination to w. Otherwise, set the next hop to **null** and the distance to $|V|$ (no route known).
  - If the distance to the destination has changed send a distance message to all neighbours.

Netchange algorithm — Pseudocode

Global variables and initialisation (for node u):

$N_u \leftarrow$ neighbours of u.

$G = (V, E) \leftarrow$ network.

for all $v \in V$:

$D_u[v] \leftarrow |V|$

$P_u[v] \leftarrow$ **null**

for all $w \in N_u$:

$D_u[w, v] \leftarrow |V|$

$D_u[u] \leftarrow 0$

$P_u[u] \leftarrow u$

for all $w \in N_u$:

Send distance message $(u, 0)$ to w.

Netchange algorithm — Pseudocode

RECOMPUTE(Vertex v) (for node u):

if $v = u$

$D_u[v] \leftarrow 0$

$P_u[v] \leftarrow u$

else

$d \leftarrow 1 + \min_{w \in N_u} D_u[w, v]$

if $d < |V|$

$D_u[v] \leftarrow d$

$P_u[v] \leftarrow w$ for which $1 + D_u[w, v] = d$ in last $\min$.

Netchange algorithm — Pseudocode

else

$D_u[v] \leftarrow |V|$

$P_u[v] \leftarrow$ **null**

if $D_u[v]$ has changed in this RECOMPUTE call

for all $x \in N_u$:

Send distance message $(v, D_u[v])$ to x.

Netchange algorithm — Pseudocode

RECEIVEDISTANCEMESSAGE(v,d) (for node u from w):

$D_u[w, v] \leftarrow d$

RECOMPUTE(v)

CHANNELFAIL(Neighbour w) (for node u):

$N_u \leftarrow N_u \setminus \{w\}$

for all $v \in V$ RECOMPUTE(v)

CHANNELREPAIR(w) (for node u from w):

$N_u \leftarrow N_u \cup \{w\}$

for all $v \in V$:

$D_u[w, v] \leftarrow |V|$

Send distance message $(v, D_u[v])$ to w.

Netchange algorithm — Invariants

- Neighbour table N_u reflects the channels that u is connected to.
- The estimated distance $D_w[v]$ from neighbour w of u to a destination v is in:
  - The neighbour distance table as $D_u[w, v]$, if no message about it is in transit.
  - The last distance message sent by w about its distance to v, if such a message is in transit.
- The node's estimated distance to the destination is 1 more than the minimum distance in the neighbour distance table to the destination, if this sum is less than $|V|$. If not, the node's estimated distance to the destination is $|V|$.

Netchange algorithm — Correctness

- When no more messages are being transferred, each node's estimated distance to a destination is 1 more than the minimum estimated distance from the neighbour to the destination (if such a neighbour with a distance less than $|V|$ to the destination exists) and 0 if the destination is the node itself. This is the same as the recursive definition of distance, so all routes that are found are optimal. Similarly, any routes found have a length of $|V| - 1$ at the most, and if no route exists, the distance is estimated at $|V|$.
- Proving that Netchange terminates involves defining a function from a state of the algorithm to a finite subset of $\mathbb{Z}$ and showing that state change in the algorithm decreases the value of the function, implying that the algorithm executes a finite amount of steps.

Netchange algorithm — Problems

- The algorithm only gives meaningful results when it has stabilised; i.e. when no messages are in transit. Thus, if topological changes are frequent, the algorithm is useless.
- Detecting a stable state is hard.

Netchange algorithm — Modifications

- By changing the recompute procedure, the algorithm can be adapted to shortest path routing. Proving that the resulting algorithm works is harder.
- The algorithm can also be adapted to changing channel weights, but this is not very useful if changes are frequent.

Conclusion

- Choosing a routing algorithm is a matter of horses for courses; things to consider include the frequency of changes in the network, whether the routing tables can be calculated in a centralised fashion and the the cost function one wishes to minimise.
- On static networks, simple algorithms can be used to calculate fixed routing tables.
- Dynamically changing networks require more complex distributed algorithms to be usable.