Routing Algorithms

CS60002: Distributed Systems

Pallab Dasgupta
Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur
Main Features

- **Table Computation**
  - The routing tables must be computed when the network is initialized and must be brought up-to-date if the topology of the network changes.

- **Packet Forwarding**
  - When a packet is to be sent through the network, it must be forwarded using the routing tables.
Performance Issues

**Correctness:** The algorithm must deliver every packet to its ultimate destination

**Complexity:** The algorithm for the computation of the tables must use as few messages, time, and storage as possible

**Efficiency:** The algorithm must send packets through good paths

**Robustness:** In the case of a topological change, the algorithm updates the routing tables appropriately

**Fairness:** The algorithm must provide service to every user in the same degree
Good paths …

**Minimum hop:** The cost of a path is the number of hops

**Shortest path:** Each channel has a non-negative cost – the path cost is the sum of the cost of the edges. Packets are routed along shortest paths.

**Minimum delay/congestion:** The bandwidth of a path is the minimum among the bandwidths of the channels on that path.

**Most robust path:** Given the probability of packet drops in each channel, packets are to be routed along the most reliable paths.
Destination-based Forwarding

// A packet with destination d was received or generated at node u
if \( d = u \)
    then deliver the packet locally
else send the packet to \( \text{table\_lookup}_u(d) \)
The algorithm computes the distance between each pair of nodes in $O(N^3)$ steps.
The simple distributed algorithm

// For node $u$ ...

var $S_u$ : set of nodes;
$D_u$ : array of weights;
$Nb_u$ : array of nodes;

begin

$S_u = \Phi$;
forall $v \in V$ do

if $v = u$ then

begin $D_u[v] = 0$; $Nb_u[v] = udef$ end
else if $v \in Neigh_u$ then

begin $D_u[v] = w_{u,v}$; $Nb_u[v] = v$ end
else begin $D_u[v] = \infty$ ; $Nb_u[v] = udef$ end;
The simple distributed algorithm contd...

while $S_u \neq V$ do
    begin pick $w$ from $V \setminus S_u$; \(\ll All nodes must pick the same w\)
        if $u = w$
            then broadcast the table $D_w$
            else receive the table $D_w$
        forall $v \in V$ do
            if $D_u[w] + D_w[v] < D_u[v]$ then
                begin
                    $D_u[v] = D_u[w] + D_w[v]$;
                    $Nb_u[v] = Nb_u[w]$
                end;
        $S_u = S_u \cup \{w\}$
    end
end
Important property of the simple algorithm

Let $S$ and $w$ be given and suppose that

1. For all $u$, $D_u[w] = d_S(u, w)$ and
2. If $d_S(u, w) < \infty$ and $u \neq w$, then $Nb_u[w]$ is the first channel of a shortest $S$-path to $w$

Then the directed graph $T_w = (V_w, E_w)$, where

$u \in V_w \iff D_u[w] < \infty$ and
$ux \in E_w \iff (u \neq w \land Nb_u[w] = x)$

is a tree rooted towards $w$. 
**Toueg’s improvement**

- **Toueg’s observation:**
  - A node \( u \) for which \( D_u[w] = \infty \) at the start of the \( w \)-pivot round does not change its tables during the \( w \)-pivot round.
  - If \( D_u[w] = \infty \) then \( D_u[w] + D_w[v] < D_u[v] \) is false for every \( v \).
  - Consequently, only the nodes that belong to \( T_w \) need to receive \( w \)'s table, and the broadcast operation can be done efficiently by sending the table \( D_w \) only via the channels that belong to the tree \( T_w \).
The Chandy-Misra Algorithm

\[ \text{var} \ D_u[v_0] \ : \ \text{weight} \ \ \ \ \ \text{init} \ \infty \ ; \]
\[ Nb_u[v_0] \ : \ \text{node} \ \ \ \ \ \text{init} \ udef \ ; \]

For node \( v_0 \) only:
\begin{align*}
\text{begin} & \ D_{v_0}[v_0] = 0 \ ; \\
& \ \ \ \ \ \text{forall} \ w \in Neigh_{v_0} \text{ do send } \langle \text{mydist}, v_0, 0 \rangle \text{ to } w \\
\text{end}
\end{align*}

Processing a \( \langle \text{mydist}, v_0, d \rangle \) message from neighbor \( w \) by \( u \):
\begin{align*}
\{ \langle \text{mydist}, v_0, d \rangle \in M_{wu} \} \\
\text{begin} & \ \text{receive } \langle \text{mydist}, v_0, d \rangle \text{ from } w \ ; \\
& \ \ \ \ \ \text{if } d + \omega_{uw} < D_u[v_0] \text{ then} \\
& \ \ \ \ \ \text{begin} \ D_u[v_0] = d + \omega_{uw} \ ; \ Nb_u[v_0] = w \ ; \\
& \ \ \ \ \ \ \ \ \ \ \ \text{forall} \ x \in Neigh_u \text{ do send } \langle \text{mydist}, v_0, D_u[v_0] \rangle \text{ to } x \\
& \ \ \ \ \ \text{end} \\
\text{end}
\end{align*}
The Netchange Algorithm

- **Computes routing tables according to** *minimum-hop* measure
- **Assumptions:**
  - **N1:** The nodes know the size of the network (*N*)
  - **N2:** The channels satisfy the FIFO assumption
  - **N3:** Nodes are notified of failures and repairs of their adjacent channels
  - **N4:** The cost of a path equals the number of channels in the path
- **Requirements:**
  - **R1.** If the topology of the network remains constant after a finite number of topological changes, then the algorithm terminates after a finite number of steps.
  - **R2.** When the algorithm terminates, the tables *Nb*[v] satisfy
    
    (a) if *v* = *u* then *Nb*[v] = *local*;
    
    (b) if a path from *u* to *v* ≠ *u* exists then *Nb*[v] = *w*, where *w* is the first neighbor of *u* on a shortest path from *u* to *v*;
    
    (c) if no path from *u* to *v* exists then *Nb*[v] = *udef*. 
The Netchange Algorithm

var Neigh\_u : set of nodes ; // The neighbors of u
\[ D_u \] : array of 0 .. N ; // \[ D_u[v] \] estimates \( d(u,v) \)
\[ Nb_u \] : array of nodes ; // \( Nb_u[v] \) is preferred neighbor for v
\[ ndis_u \] : array of 0 .. N ; // \( ndis_u[w, v] \) estimates \( d(w,v) \)

Initialization:
begin forall \( w \in Neigh_u, v \in V \) do \( ndis_u[w, v] = N \) ;
forall \( v \in V \) do
    begin \( D_u[v] = N \) ; \( Nb_u[v] = undefined \) end ;
    \( D_u[u] = 0 \) ; \( Nb_u[u] = local \) ;
forall \( w \in Neigh_u \) do send \( \langle mydist, u, 0 \rangle \) to \( w \)
The Netchange Algorithm contd.

Procedure Recompute( \( v \) ):

begin if \( v = u \)

then begin \( D_u[v] = 0 ; Nb_u[v] = \text{local} \) end

else begin\ /* estimate distance to \( v */

\( d = 1 + \min\{ ndis_u[w,v] : w \in \text{Neigh}_u \} ; \)

if \( d < N \) then

begin \( D_u[v] = d ; \)

\( Nb_u[v] = w \) with \( 1 + ndis_u[w,v] = d \)

end

else begin \( D_u[v] = N ; Nb_u[v] = \text{undefined} \) end

end ;

if \( D_u[v] \) has changed then

forall \( x \in \text{Neigh}_u \) do send \( \langle \text{mydist}, v, D_u[v] \rangle \) to \( x \)

end
The Netchange Algorithm contd.

Processing a \( \langle \text{mydist}, v, d \rangle \) message from neighbor \( w \):

\[
\{ \text{A } \langle \text{mydist}, v, d \rangle \text{ is at the head of } Q_{wv} \}
\]

\[
\text{begin receive } \langle \text{mydist}, v, d \rangle \text{ from } w ; \quad ndis_u[w,v] = d ; \text{Recompute}( v )
\]

\end}

Upon failure of channel \( uw \):

\[
\text{begin receive } \langle \text{fail, w} \rangle ; \quad \text{Neigh}_u = \text{Neigh}_u \setminus \{w\} ;
\]

\[
\text{forall } v \in V \text{ do } \text{Recompute}( v )
\]

\end}

Upon repair of channel \( uw \):

\[
\text{begin receive } \langle \text{repair, w} \rangle ; \quad \text{Neigh}_u = \text{Neigh}_u \cup \{w\} ;
\]

\[
\text{forall } v \in V \text{ do }
\]

\[
\text{begin } ndis_u[w,v] = N ;
\]

\[
\text{send } \langle \text{mydist, v, } D_u[v] \rangle \text{ to } w
\]

\end}