CSc72010

Shortest Paths and RIP

IP Layer

The layers in the 7-layer model:

physical – the cables
data link – communication between directly connected hosts (in Internet terminology, on a local area network – a LAN)
    Ethernet (802.3)
    Wireless (802.11)
network – communication between indirectly connected hosts (different LANs)
    IP
transport – communication between processes (there can be multiple processes on a host
    UDP (IP between processes)
    TCP (reliable, in-order, exactly once)
session -
presentation -
application – whatever

The above are the minimal services at the layers, other services can be offered as well. Also, a different layering may be used (e.g., ATM combines data link layer functionality with network functionality and some have investigated IP directly over fiber).

Shortest Paths

Assume now that we have directed weighted links. Each node knows the weights of all incident edges and the number of nodes in the graph.
Motivation is min-cost communication

Problem: Determine the shortest path from i_0 to each other node.

This algorithm is the basis of Bellman-Ford, aka RIP (Route Information Protocol). In the presence of failures this is an unstable routing protocol and isn't much used.

We use a tree again: every node should output:

- its parent in the tree
- its distance from i_0 in the (spanning) tree (sum of weights on edges)

We assume each node knows the weights of all incident edges and the number of nodes in the digraph.

Bellman-Ford

Each node keeps track of the shortest known distance from i_0 and updates it as new information arrives; also communicates changed distances.

```
automaton rip(r:Int, n:Set[Int])

signature
        send (v:Message, const r, j:Int)
        receive(v:Message, i:Int, const r)

states
        distance:Array[Int, Int] := constant(-1),
        neighbors:Set[Int] := n,
        sent:Set[Int] := {}

transitions
        send(v, i, j)
        pre
                v=distances /\ j \in nbrs /\ ~(j \in sent)
        eff
                sent := insert(j, sent)

        receive(v, i, j)
        eff
                for v[k] ~= -1 /\ k ~= r do
                        if v[k]+1 < distance[k] then
                                distance[k] := v[k]+1;
                                change := true
                        fi
                od
                if change then sent := {}
```

Terminates after n-1 rounds

Key properties:

1) No cycles (if there are no failures)
2) Every node eventually has the shortest distance to every other node.

Let's suppose all nodes run this concurrently. The result of the above algorithm is that destination nodes know the shortest path from each possible source. To use this for routing, in practice, all the source nodes need to know the shortest distances and the next node on the path to each destination. In fact, for routing, source nodes build a table like:

Destination	Next hop	Distance (to destination)	Go through

THis is non-trivial in a directed graph. The importance of this, these days, is that under certain circumstances wireless networks may have some unidirectional links. This table

should be initialized with all destinations, with go through=destination, and distance=infinity.

Just the idea:
Then, all nodes could broadcast their own information in a message containing quadruples <src, dst, parent, distance>. When a message arrives at the node X, for each quadruple <X, A, B, n> in the message, update the distance for destination A to be equal to n and if the entry in the "go through" column is A, then change it to B. If the entry is not A, and the distance is less than the distance, save the quadruple. The go through column works backwards along the path, until it gets back to an out-nbr of X. Each time you update the go through column, check the set of saved tuples for one with A equal to the value in the go through column; if found, change the go through column to B and discard the tuple.

RIP

This is the Internet algorithm that is based on the above shortest paths algorithm.
As usual, in practice, we assume a *bidirectional network* and therefore an undirected graph. here's how the algorithm works:

Assume:
Each node knows the "distance" to each directly-connected neighbor router – a link that is not there or down gets infinite cost.

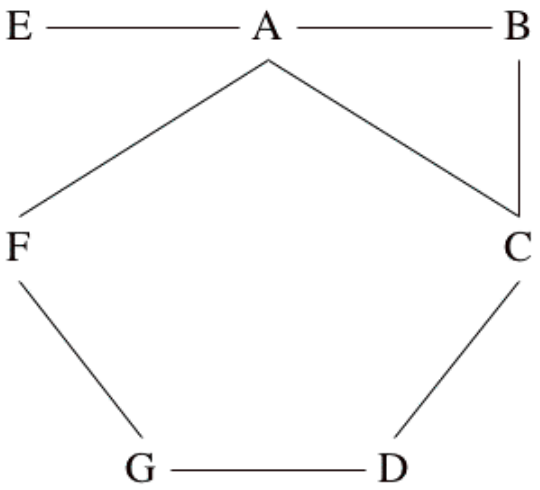
Routing table:
A table R_i of distances to routers, initially only to directly-linked neighbors.

Initially, periodically, and whenever there is a change to the routing table R_i, send routing table to all neighboring routers.

On receiving a routing vector R_j from neighbor j, update your own routing table R_i using the rule:
If $d_i(j) + R_j(k).distance < R_i(k).distance$ then
    change $R_i(k).to$ $d_i(j) + R_j(k).distance$
    change next-hop (parent) to j

Example from Peterson & Davies, Figure 4.15, Table 4.5

RIP Example



	A	B	C	D	E	F	G
A	0	<1,B>	<1,C>		<1,E>	<1,F>	
B	<1,A>	0	<1,C>				
C	<1,A>	<1,B >	0	<1,D>			
D			<1,C>	0			<1,G>
E	<1,A>				0		
F	<1,A>					0	<1,G>
G				<1,D>		<1,F>	0

Round 1. A sends routing table to B, C, E, F; B sends routing table to A, C; C sends

routing table to A, B; D sends routing table to C, G; E sends routing table to A; F sends routing table to A, G; G sends routing table to D, F.

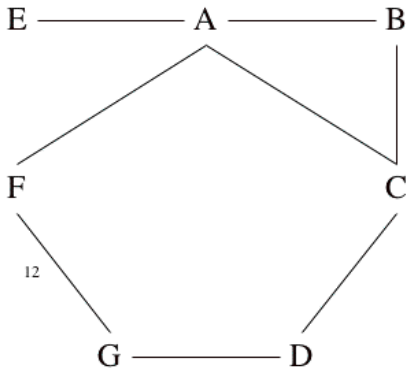
	A	B	C	D	E	F	G
A	0	<1,B>	<1,C>	<2,C.>	<1,E>	<1,F>	<2,F>
B	<1,A>	0	<1,C>	<2,D>	<2,A>	<2,C>	
C	<1,A>	<1,B >	0	<1,D>	<2,A>	<2,A>	<2,D>
D	<2,C>	<2,C>	<1,C>	0		<2,G>	<1,G>
E	<1,A>	<2,A>	<2,A>		0	<2,A>	
F	<1,A>	<2,A>	<2,A>	<2,G>	<2,A>	0	<1,G>
G	<2,F>		<2,D>	<1,D>		<1,F>	0

Round 2.

	A	B	C	D	E	F	G
A	0	<1,B>	<1,C>	<2,C.>	<1,E>	<1,F>	<2,F>
B	<1,A>	0	<1,C>	<2,D>	<2,A>	<2,C>	<3,A>
C	<1,A>	<1,B >	0	<1,D>	<2,A>	<2,A>	<2,D>
D	<2,C>	<2,C>	<1,C>	0	<3,C>	<2,G>	<1,G>
E	<1,A>	<2,A>	<2,A>	<3,A>	0	<2,A>	<3,A>
F	<1,A>	<2,A>	<2,A>	<2,G>	<2,A>	0	<1,G>
G	<2,F>	<3,F>	<2,D>	<1,D>	<3,F>	<1,F>	0

Change weights, so that we don't want to use the edge between F and G

RIP Example



	A	B	C	D	E	F	G
A	0	<1,B>	<1,C>		<1,E>	<1,F>	
B	<1,A>	0	<1,C>				
C	<1,A>	<1,B >	0	<1,D>			
D			<1,C>	0			<1,G>
E	<1,A>				0		
F	<1,A>					0	<12,G>
G				<1,D>		<12,F>	0

Round 1. A sends routing table to B, C, E, F; B sends routing table to A, C; C sends routing table to A, B; D sends routing table to C, G; E sends routing table to A; F sends routing table to A, G; G sends routing table to D, F.

	A	B	C	D	E	F	G
A	0	<1,B>	<1,C>	<2,C.>	<1,E>	<1,F>	<13,F>
B	<1,A>	0	<1,C>	<2,D>	<2,A>	<2,A>	
C	<1,A>	<1,B >	0	<1,D>	<2,A>	<2,A>	<2,D>
D	<2,C>	<2,C>	<1,C>	0		<13,G>	<1,G>
E	<1,A>	<2,A>	<2,A>		0	<2,A>	
F	<1,A>	<2,A>	<2,A>	<2,G>	<2,A>	0	<12,G>

G	<13,F>		<2,D>	<1,D>		<12,F>	0

Round 2.

	A	B	C	D	E	F	G
A	0	<1,B>	<1,C>	<2,C.>	<1,E>	<1,F>	<3,C>
B	<1,A>	0	<1,C>	<2,D>	<2,A>	< 2,A>	<3,C>
C	<1,A>	<1,B >	0	<1,D>	<2,A>	<2,A>	<2,D>
D	<2,C>	<2,C>	<1,C>	0	<3,C>	<3,C>	<1,G>
E	<1,A>	<2,A>	<2,A>	<3,A>	0	<2,A>	<14,A>
F	<1,A>	<2,A>	<2,A>	<2,G>	<2,A>	0	<12,G>
G	<2,F>	<3,F>	<2,D>	<1,D>	<14,F>	<12,F>	0

Round 3.

	A	B	C	D	E	F	G
A	0	<1,B>	<1,C>	<2,C.>	<1,E>	<1,F>	<3,C>
B	<1,A>	0	<1,C>	<2,D>	<2,A>	< 2,A>	<3,C>
C	<1,A>	<1,B >	0	<1,D>	<2,A>	<2,A>	<2,D>
D	<2,C>	<2,C>	<1,C>	0	<3,C>	<3,A>	<1,G>
E	<1,A>	<2,A>	<2,A>	<3,A>	0	<2,A>	<4,A>
F	<1,A>	<2,A>	<2,A>	<2,G>	<2,A>	0	<4,A>
G	<2,F>	<3,F>	<2,D>	<1,D>	<3,F>	<4,D>	0