# DVR

TDTS06 – Computer Networks – Assignment 4

## 1. How Distance Vector Routing Works

Distance vector routing (or DVR) is a distributed algorithm. This means that we don't need a global overview like in LS, each node receive information from his directly attached neighbours, process it, and then deliver the result to all his direct neighbours. This simple process, repeated various times gives us as a result the information about the entire network. In other words, is an iterative algorithm. Lastly, knowing this we can see that it also works in an asynchronous way.

Each node has a table with the travel cost from himself to his neighbours. This values don't need to be the direct ones. If, for example, a node receives information about a better route through one of his neighbours, the table should be update indicating that lowest cost.

For determining the lowest cost to a node we use the Bellman-Ford equation:

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

In our code, this is implemented in the function *processUpdate()* which is called every time that a node receives an update. Once it has the actualized table, the node informs his neighbours about it.

Poison reverse is implemented directly in the *alertNeighbours()* function, how it works is really simple. If a certain node A routes through B in order to get to C, when A will communicate his distance vector to C, A will say that his directly cost to C is infinite.

## 2. How to Test:

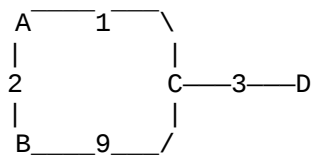
The simulator is already implemented in a way that provides you a window for each node. In this windows the program prints the state of the node at different times. Through the function *printDistanceTable()* we get the state of the costs table mentioned above. In our code this table is implemented as a HashMap using the node ID as key, and that node distance vector as value. Besides we have an array called *actCosts[]* which contains our node's actual distance vector, and another array called *route[]* that indicates through which node we route in order to get to the destination. The rest of the variables implemented by us aren't strictly necessary, but helpful to have a better idea of each node.

Once we have our simulator "working" the way of testing it is checking this windows for each node and corroborating that the results that are showing there correspond to our graph (at that specific time).

Alfonso Herrera ID: 950401T199
Juan Ramón del Caño ID: 960721T019

# 3. Poison Reverse Limitations:

The poison reverse works fine solving loops, but only in size 2. For example, in the graph




If the optimal path from B to D is B-A-C-D, there could be a case where, if instead of 3 the link cost between C and D changes to 20, C will decide to route over B in order to get to D (generating a count-to-infinity loop).

# 4. Beyond Poison Reverse:

There are other protocols, such as RIP that combine an implementation of Poison Reverse with Split Horizon. Split Horizon prevents the node B (in the previous example) to indicate that has a route to D. This way, when the link cost is increased from 3 to X, the count-to-infinity loop is avoided.

This is an improvement and prevents some loops, but not all of them.

### *References:*


- Wikipedia, DVR: https://en.wikipedia.org/wiki/Distance-vector_routing_protocol#Workarounds_and_solutions
- Wikipedia, RIP: https://en.wikipedia.org/wiki/Routing_Information_Protocol
- Wikipedia, Split Horizon: https://en.wikipedia.org/wiki/Split_horizon_route_advertisement