Linköping University

Department of Computer and Information Science

TDTS06 Computer Networks

Distance Vector Routing

Report

Group C/D 13

Chvátal Martin march011@student.liu.se
Peschke Lena lenpe782@student.liu.se

Teaching assistant :
Schmidt Johannes johannes.schmidt@liu.se

October 17, 2014

I How distance vector routing works

In general Distance vector routing uses a decentralised routing algorithm to find the least-cost path between the routers in a network. The distance vector routing algorithm considers a network as an undirected graph of routers, represented as nodes, and links, represented as edges. It is iterative, asynchronous and distributed [KR13, p. 371]. It makes use of the Bellman-Ford equation:

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\} \tag{1}$$

The equation states that the cost of the least-cost path from node x to y, $d_x(y)$, is the minimum of the sum of the cost to get from x to a neighbour v and the cost of the least-cost path from v to y.
The algorithm works by making the nodes exchange distance vectors. To make use of the equation, each node needs to know about all of the other nodes in the network and the cost to its direct neighbours. Let's walk through the algorithm step by step.

1. Initially, each node knows the cost to its direct neighbours. It sends its distance vector to its neighbours.

2. Upon receiving an update (link cost change or new distance vector from a neighbour), a node recomputes its distance vector using the Bellman-Ford equation (1). If there is a change, it sends its own new vector to all the neighbours.

3. "The process of receiving updated distance vectors from neighbours, recomputing routing table entries, and informing neighbours of changed costs of the least-cost path to a destination continues until no update messages are sent." [KR13, p. 375]

After step 3, the estimated costs have converged to the actual least-costs.

II How we tested the algorithm

We tested the algorithm with the three different simulated networks provided. They contain 3, 4 and 5 nodes and are represented in Figure 7 at [ZC13].
We timed the different executions (in ms) and compared them with each other. It is no surprise that the link change increases the convergence time, whereas the poisoned reverse decreases it again (which is its purpose). One value is surprising: the network with 5 nodes does not see any difference between an increase in link cost with and without poisoned reverse.

	Basic case	Link change	Poisoned reverse
3 nodes	33.449	322.306	64.250
4 nodes	93.796	20030.217	20023.406
5 nodes	189.272	20021.531	20021.531

III Poisoned reverse

When a link cost in the network increases, a count-to-infinity problem can occur. A packet can then be caught in a routing loop until the nodes have figured out the shortest path, which can take a while (as many iterations as the difference between the previous and the new least-cost). Poisoned reverse is an addition to the DVR algorithm which helps avoiding this situation. If a node x routes through y in order to get to z, x lies to y about its distance to z, so that an increase in the cost link between y and z results in an convergent update that takes only two iterations.

Where it fails Poisonous reverse fails if a count-to-infinity problem triggers a loop of three or more nodes [KR13, p. 378].
Let us consider a network of 4 routers, A, B, C and D. The network is represented below in Figure 1. For this example, we will analyse partial routing tables which show the different path costs from one node to all of the others in the network. The example is inspired by [She12].

Initial situation The routing tables for nodes A and B are represented in Tables 1 and 2. The least-cost routes to the different nodes in the network are emphasised in blue. In Table 1, any path through D has an ∞ cost because A and D are not connected, and to go via D, there is always a back-and-forth. In Table 2, the two ∞ result directly from poisonous reverse, since A and C route through B to get to D.

Step 1 The cost of the link between B and D changes from 1 to 50. B and D get the information and update their tables. B's new routing table is given in Table 3.

Step 2 B sends its updated vector across the network. A gets the information and updates its own table, given in Table 4. A believes there is a path via C, not knowing that C also routes via B.

Step 3 B gets the update from A and updates its own table, as can be seen in Table 5. As A advertises a route through C, B now also thinks there is a path with a lesser cost than 50 to D. From here on, we can see that a count-to-infinity situation occurs and will not stop until the actual cost is detected, requiring 47 more updates between A and B (if we leave C out).

	via		
	B	C	D
B	1	2	∞
C	2	1	∞
D	2	3	∞

Table 1: A's initial routing table

	via		
	A	C	D
A	1	2	∞
C	2	1	∞
D	∞	∞	1

Table 2: B's initial routing table

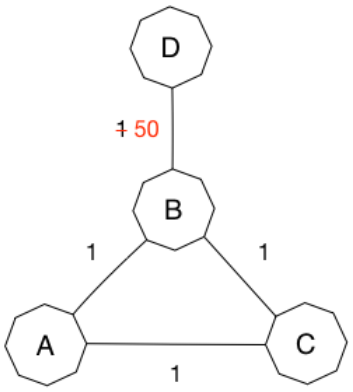

Figure 1: Example topology where poisoned reverse fails.

	via		
	A	C	D
A	1	2	∞
C	2	1	∞
D	∞	∞	50

Table 3: B's routing table after step 1

	via		
	B	C	D
B	1	2	∞
C	2	1	∞
D	51	3	∞

Table 4: A's routing table after step 2

	via		
	A	C	D
A	1	2	∞
C	2	1	∞
D	4	∞	50

Table 5: B's routing table after step 3

Solutions In practise, the Routing Information Protocol (RIP) uses DVR with poisoned reverse where infinity is 16 hops. This does not avoid the count-to-infinity problem, but limits the damage by stopping updates after it reaches "infinity". In addition, *hold-down timers* can be used: if a route becomes unusable, it is kept in the tables with an infinity value for a certain amount of time. If an update advertises a better or equally good route, it is replaced. On the other hand, if the update indicates a worse cost, it is ignored. [Koz05]
Another solution (which actually solves the problem) is using *path vectors*: in addition to the distance vectors, we can keep a record of the path, i. e. the nodes traversed to obtain the cost. When a link cost increases or a link disappears from the network, a node can check if the path is still valid by looking it up. This is used in BGP, but involves more overhead [Siu99].

References

[Koz05] Charles M. Kozierok. *The TCP/IP Guide*, chapter "RIP Special Features For Resolving RIP Algorithm Problems". No Starsh Press, 3rd edition, September 2005.

[KR13] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, 6th edition, 2013.

[She12] Scott Shenker. Poison Reverse. `http://www-inst.eecs.berkeley.edu/~ee122/fa12/discussion/Section108_Answers.pdf`, Fall 2012.

[Siu99] Prof. K.-Y. (Sunny) Siu. Distance-vector routing (distributed Bellman-Ford algorithm). `http://web.mit.edu/course/2/2.993/www/lectures/lecture14.txt`, April 1999.

[ZC13] Farrokh Ghani Zadegan and Niklas Carlsson. Distance vector routing. `https://www.ida.liu.se/~TDTS06/labs/2014/DV/default.html#Execution_of_the_Simulator`, August 2013.