Lab #3 - Distance Vector Routing Protocol

Karl-Johan Grinnemo
karl-johan.grinnemo@kau.se

Karlstad University — January 12, 2023

Introduction

The *routing algorithm* is that part of the network layer software responsible for deciding which output link an incoming packet should be transmitted on. There are two basic algorithms of *dynamic routing*: *distance-vector routing* and *link-state routing*. This lab allows students to practice the principles behind distance-vector routing.

Examination

The lab is graded as *pass* or *failed*. To pass, the student or students should hand in their answers to the theoretical questions provided in the separate document, `Lab_3_Exercises.pdf`. They should also demonstrate the lab for one of the lab assistants in the course and submit their source code according to instructions on Canvas.

Preparations

- Review Section 5.2, *Routing Algorithms* (especially Section 5.2.2, *Distance-Vector (DV) Routing Algorithm*) in the textbook.[1]
- Download the source code to the network emulator from Canvas (`Lab_3_Source_Code.zip`), and uncompress the zip file in the folder where you intend to develop your distance-vector routing protocol.

Description

In this lab, you are asked to implement a distance-vector routing protocol in the network emulator whose source code is found in the zip file: `Lab_3_Source_Code.zip`. The network emulator emulates the fixed four-node network depicted in Figure 1, and it is your task to implement a distance-vector routing protocol in the four nodes so that it works as described in the textbook[2].

The network emulator works similarly to the discrete-event network emulator in lab 2, *Transport Protocol*. The core of the network emulator – the event loop – resides in the source file, `sim_engine.c`. The source file `sim_engine.c` and the header file `sim_engine.h`, should not be modified by you. Instead, it is

[1] J. F. Kurose och K. W. Ross. *Computer Networking - A Top-Down Approach*, 7th edition.
[2] Section 5.2.2, *Distance-Vector (DV) Routing Algorithm*, in the 7th edition of the textbook.

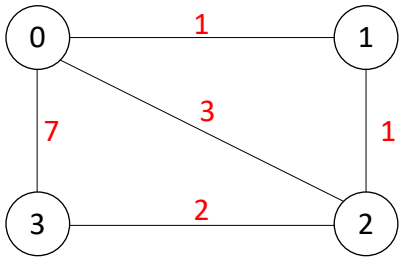

Figure 1: The four-node network emulates in the network emulator in laboration 3. The figures beside the links are the link distances (or costs). Note that the link distances are bidirectional. That is, the distance from node i to node j is the same as the distance from node j to node i.

your job to complement the network emulator with a distance-vector routing protocol whose code resides in the four source files: `node0.c`, `node1.c`, `node2.c`, and `node3.c`. In total, you are expected to implement eight functions with code as follows:

`rtinitX()` : The four functions, `rtinit0()`, `rtinit1()`, `rtinit2()`, and `rtinit3()` are called in the beginning of an emulation and are supposed to initialize the *distance tables* of the nodes 0, 1, 2, and 3, as well as any other required data structures. For example, in node 0 the function `rtinit0()` is supposed to initialize the distance table `dt0` of type `struct distance_table`, and maybe some other variables as well. The distance tables are 4×4 matrices where the ith row and jth column of a node's distance table holds the node's known distance to node i via node j. For example, the initial distance table for node 0 will hold the distances shown in the table below.

Column/ Row	0	1	2	3
0	0	999	999	999
1	1	999	999	999
2	3	999	999	999
3	7	999	999	999

As follows, the distance from node 0 to node 0 via node 0 is 0; the distance from node 0 to node 1 via node 0 is 1; the distance from node 0 to node 2 via node 0 is 3; and the distance from node 0 to node 3 via node 0 is 7. All other distances in the distance table are set to ∞, denoted by the integer 999.

After having done the initialization part, the code in the `rtinitX()` functions should send their known minimum distances to their neighboring nodes. For example, the code in `rtinit0()` should send its known minimum distances to nodes 1, 2, and 3, to its neighbors (that is, the same nodes: 1, 2, and 3). The transmission is done by invoking the function, `tolayer2()`, that takes an argument, `packet`, a *routing packet* of type `struct rtpkt`. The type `struct rtpkt` contains three elements:

1. `sourceid` is the identity of the node sending the packet, that is, the source node. For example, if node 0 is the source node, the source identity is 0.
2. `destid` is the identity of the node to which a packet is sent, that is, the destination node. For example, if node 1 is the destination node, then the destination identity is 1.
3. `mincost` is an array or vector which contains the minimum distances from the source node to nodes 0, 1, 2, and 3. That is, `mincost[0]` holds the minimum distance from the source node to node 0, `mincost[1]` holds the minimum distance from the source node to node 1, etc.

`rtupdateX` : The four functions, `rtupdate0`, `rtupdate1`, `rtupdate2`, and `rtupdate3` are at the heart of the distance vector routing protocol. These functions are supposed to receive routing packets from its neighboring nodes; update the distance table of the node in question according to the *Bellman–Ford algorithm*; and communicate any changes in the distance table to neighboring nodes. In the same way as in the `rtinitX()` functions, the routing packets are sent by invoking the function `tolayer2()`.

The global variable, `TRACE`[3], makes it possible for you to follow the progression of your distance-vector protocol. The default value is 1, which means that some but not all trace messages are displayed on the terminal; more trace messages are displayed when you increase `TRACE` to 2, 3, or 4. You turn off tracing by setting `TRACE` to 0. Other useful functions are the `printdtX()` functions which print out the content of the distance tables, for example, function `printdt0()` prints out the distance table of node 0.

When `TRACE` is set to 2 in your implementation, all routing packet exchanges and the updated distance tables should be displayed during an emulation run. When an emulation run ends, the content of the nodes' distance tables should be obvious from your trace printout.

End of Lab

[3]Defined in source file `sim_engine.c` and declared in `sim_engine.h`.