CSE 160 – Computer Networks

University of California, Merced

Project 2: Distance Vector Routing

Introduction

Your assignment is to extend your TOSSIM node to support efficient routing. In project 1, you were able to send packets from a source to a destination using flooding. In this project, you will be able to route packets hop-by-hop through the network. Packets will go through the network using a routing table you create using distance vector routing.

Objectives and Goals

Your goal is to be able to construct an up to date routing table to forward packets towards its destination.

1. **Neighbor Discovery.** Use neighbor discovery to determine your set of neighbors. You will need your code from project 1 working fully.
2. **Sharing Routing Table Information.** Each node should tell its immediate neighbors about its notion of distance to all other nodes.
3. **Shortest-Path Calculation using the new DV updates.** Build and keep an up-to-date routing table that allows you to determine the next-hop to forward a packet towards its destination.
4. **Forwarding.** Send packet using the next-hops as directed by the calculated routing table.

Where to Start

Read about distance vector routing in *Peterson 3.3.2*. This is extremely important for understanding DVR and will help you with your own implementation of DVR. In this project, you will be implementing a simpler form of RIP.

You should create a struct in order to store the DVR information. This struct can be placed as the payload of a normal packet. You can safely make the assumption that there will be no nodes which has an id that is larger than 255. Assume that the costs is based on hops and does not include link quality estimates. This struct created can be added to your payload through memcpy().

This DVR information should be sent to immediate neighbors, not flooded throughout the

network. Design decisions that should be considered include how often distance vectors should be sent out. Be sure to use as little packets as possible while still keeping nodes as up to date as possible. When sending your distance vectors to your neighbors be sure to use a separate AM Channel.

Also implement Split Horizon and Poison Reverse technique to reduce the chance of forming loops and use a maximum cost to counter the count-to-infinity problem. Don't forget to add the cost of the neighbor's link when you are calculating the cost of the route.

Requirements

Once the above has been completed you should be able to send a packet over multiple hops using distance vector routing instead of flooding. Check to see if pings work in multiple different topologies where links can be broken. Your routing protocol should be able to detect broken paths and repair itself by finding an alternative path.

To illustrate properly implemented distance vector, make sure to use the debug channel ROUTING_CHANNEL to showing how your implementations works. This channel should print out debug statements when a ping message is making each hop. The output should have the following format for each time it is routed to its destination.

DEBUG(1): Routing Packet - src: 3, dest: 10, seq: 0, next hop: 2, cost: 26

Also when the command "routingTableDump" is issued, an entire routing table should be printed to the screen. See command.h to see how to implement this command. The output should be as follow:

DEBUG (3): Routing Table:

DEBUG (3):	Dest	Hop	Count
DEBUG (3):	6	6	1
DEBUG (3):	5	5	1
DEBUG (3):	1	1	1
DEBUG (3):	2	2	1
DEBUG (3):	7	7	1
DEBUG (3):	4	2	2

Hints

You may find it beneficial to delay the first distance vector update to immediate neighbors while your neighbor discovery is still finding neighbors.

Deliverables

What is expected on the due date is the following:

- Source code of the TinyOS implementation with working distance vector and split horizon with poison reverse
- A single page document describing the design process and your design decisions.
- A document with short answers to the discussion questions.

A physical copy of the document and related questions is required on the due date. All documents and a copy of the source code should be submitted to class web page. Please create a compressed tarball such as Student-Name-proj2.tar.gz. You must do this before class on the day that it is due.

Additionally, you will need to demonstrate that the code you submitted works in the next lab, and be able to describe how it works. Also discuss the design process in the code.

Discussion Questions

1. What are the pros and cons of using distance vector routing compared to link state routing?
2. Does your routing algorithm produce symmetric routes (that follow the same path from X to Y in reverse when going from Y to X)? Why or why not?
3. What if a node advertised itself as having a route to some nodes, but never forwards packets to those nodes? Is there anything you can do in your implementation to deal with this case?
4. What happens if a distance vector packet is lost or corrupted?
5. What would happen if a node alternated between advertising and withdrawing a route to a node every few milliseconds? How might you modify your implementation to deal with this case?

Please be concise with your answers.

Grading Guidelines

Each part of the project is graded on a 5 point (0-4) scale, multiplied by the weight of the project. The weighted grades from all parts of the project are added together to produce the final grade.

The five point scale is based on how well you show your understanding of the problem, and in case of code how well your implementation works:

0 – nothing turned in

1 – show minimal understanding of the problem / most things don't work

2 – show some understanding of the problem / some things work

3 – show a pretty good understanding of the problem / most things work

4 – show excellent understand of the problem / everything works

The weights for project 2 are:

70% - Distance Vector implementation

20% - Split Horizon with Poison Reverse

5% - Write-up design decisions

5% - Discussion questions

Your submission will be graded on correctness, completeness, and your ability to explain its implementation.

Efficient Routing (optional for extra credit)

Note that grades for the course are established without considering extra credit. We then add any extra credit points. Doing extra credit is entirely optional, and not doing them will in no way harm your grade, even if everyone else in the class does the extra credit problems. We offer extra credit problems as a way of providing challenges for those students with both the time and interest to pursue certain issues in more depth.

Distance vector routing is a "mechanism" for controlling routes; it does not specify the "policy" used for selecting which routes to use and which ones to avoid. Clearly, the end user

performance can vary dramatically based on the route selection. For example, what if certain wireless hops were very lossy? Shortest path routing might choose those routes, even though the end user might be happier with another path through more reliable links.

Part 2 is to design and implement a routing algorithm that chooses better routes than the default selected by distance vector shortest path. Better is of course in the eyes of the beholder - lower loss rates, lower latency, higher bandwidth are all reasonable choices. Note that your design can and should make use of the facilities defined above but it need not interoperate with the code from other student projects (e.g., you can define extra protocol messages or carefully modify existing ones as needed). As a first step, for example, you'll probably want to implement something that measures the properties of a hop or a path, so that you can use that information in setting link weights for the shortest path computation or in choosing source routes for virtual circuits. You should test this with a lossy channel, where the links have a certain probability of dropping packets, so you can estimate the quality of each link.

Be sure to add to your turn in: a short description of your design and a simple illustrative test case.