Lab 2: Distance Vector with Failures

Due: October 7 at 12:01 AM

Early submission: September 30 at 12:01AM

In this lab you will be implementing distance vector routing and a simulator similar to the one you used for Lab 1. Similar to the first lab this project uses a Python virtual environment, so all of the instructions on getting started in Lab 1 are useful to review here.

Getting the Code

To acquire the simulator and lab infrastructure for this project go to https://classroom.github.com/a/wB19JKaZ, and accept the assignment. Similar to the last assignment this will create repository for you that you can clone. Once you have cloned this repository, similar to Lab 1 run:

If on OS X or Linux
> ./setup.sh

*If on Windows*
> setup

This should create the virtual environment, now look at the steps for Lab 1 for instructions on how to use the virtual environment.

Simulation Differences

Simulator documentation can be found at https://cs.nyu.edu/courses/fall19/CSCI-UA.0480-008/lab2-doc/, please make sure to read it carefully when working on the project. The simulator behaves similarly to what it did in lab 1, except for the differences noted here:

- The forwarding logic used for data packets is now different. Each data packet consists of a source and a destination. When it is received the switch executes the following protocol:
on recv(pkt, iface):
    if pkt.destination in table:
        send(pkt, table[pkt.destination])
    else:
        # We drop the packet.

Addresses, i.e., pkt.destination and pkt.src are integers in our simulation.

The protocol you provide needs to populate and update the table. To do this, given a SwitchRep object (sw in line 36, 64 and 69 of the template code) you can use:

- `sw.update_forwarding_table` to add or change what the switch does when receiving a packet.
- `sw.del_forwarding_entry` to delete a forwarding entry.
- `sw.get_forwarding_for_address` to get the current handling for an address.

- You have to handle link failures and recovery. In this project, the simulator implements protocols for detecting when links fail. It then calls into your logic (`process_link_down` on line 64 and `process_link_up` on line 65 respectively). You should respond to these events appropriately.
- When hosts connect to the network they send a message of type `HostIdentification`. Remember to handle this message appropriately.

We also provide a more complete specification at the end of this document.

**Problem**

You need to implement a distance vector routing algorithm (described in Lecture 4, 5 and 6) to provide shortest path routing. Your implementation needs to correctly handle failures.

Similar to Lab 1 you test your code by running

```
> python3 lab2.py topos/4-clique.yml
```

The simulator will throw an `assertion` if a problem is detected.

**Suggestions on Getting Started**

- Similar to last lab, we recommend using tools like `mypy` and `black`.
- You should start by getting basic functionality working without handling failures. You can test basic functionality by running > `python3 lab2.py topos/4-loop-simple.yml` This runs an instance of the simulator where no links are failed.
- When working on the project, it might be useful to see what the routing table, etc. looked like before a link is failed. On line 75 of the stencil code we have included a test function that you can use for this. Uncommenting
the print statement on line 77, 78 etc. will print forwarding state, you can
see documentation for these methods and add additional prints as necessary.

• As opposed to the last project, we do not anticipate providing the complete
protocol in lecture notes, so you need to work through the problems we
discussed in class by yourself. Please start early, since some of this is tricky.

Appendix: Simulator Mechanics

The simulator we have provided simulates networks consisting of a set of simple
switches and hosts. Both switches and hosts are assigned unique names (IDs)
and consist of a set of interfaces.

The switches in our simulator make a distinction between data and control
packets:

• A data packet is one that has been sent out by a host and is destined
to another host. When a switch receives a data packet it looks at the
destination address on the packet to decide what to do, see the protocol
above for information.

• A control packet is one that has been sent by a switch in order to
implement a protocol. When a switch receives a control packet it calls
the process_control_packet(self, sw: sim.SwitchRep, iface_id:
int, data: Any) function defined on line 37 of lab1.py. At this point
you can choose to handle this packet as you desire.

Each switch consists of several interfaces, for example in the figure above Switch
A, B and C have three interfaces each. An interface might not be connected to
anything (e.g., Interface 1 on Switch B), or connected to a link attached to an
interface on another switch or host (e.g., Interface 2 on Switch A is connected to
Interface 1 on Switch B).

Each switch in the simulator (and in the real world) consists of a distinct data
plane – responsible for quickly forwarding data packets – and a control plane –
responsible for configuring how data packets should be forwarded. The simulator
already include code for the data plane, and the ControlLogic class you provide
acts as the control plane in this case. When the simulation starts it creates an
instance of both for each switch in the network. The data plane communicates
with the control plane by calling into it for events, such as: * Calling initialize
(line 26) when it boots. * Calling process_control_packet (line 35) when a
control packet is received. * Calling process_link_down and process_link_up
when link failures or recovery is detected.

Appendix: Topology Format

The topology format is similar to Lab 1 except for a few differences:
hosts:
  h0: 1
  h1: 2
  h2: 3
  h3: 4
  h4: 5
switches:
  - s0
  - s1
  - s2
  - s3
nifaces: 5
edges:
  - [s0, s1, 10]
  - [s1, s2, 11]
  - [s2, s3, 12]
  - [s3, s0, 13]
  - [h0, s0, 14]
  - [h1, s1, 15]
  - [h2, s2, 16]
  - [h3, s3, 17]
  - [h4, s3, 18]
failure_events:
  - [down, l1]

The main differences are we now label each edge – this is the third parameter in the edges tuple. We also take a schedule of what things should fail and recover. down, 11 in this case says fail link 11 which is the link from s1–s2.