Computer Networks Programming Assignment 2

Fall 2019

Due: 13 December 2019

Introduction

In this program, we will make a simplified software-defined network. You will write three total programs:

1. A program that runs the forward-search algorithm (Dijkstra's algorithm) and creates a specifically formatted flow table which it communicates to the the . . .
2. Controller program. This program talks back and forth to the routing program by sending an adjacency matrix for the current state of the network. It receives back a flow table, which it sends on to the . . .
3. Simulated switch program. The switch program will be interactive, such that you can put in a destination IP address and ingress port and the program will print out which port that "packet" will be forwarded out. You will also be able to trigger routing updates by bringing ports up and down.

Please make each of the programs somewhat "chatty" so that there is good visibility of the work they are doing.

Requirements

You will be writing these three programs in C or Python 3. Here are the further specifications for each program.

Routing Program

The routing program should implement Dijkstra's algorithm as discussed in Lecture 16. It should listen on some TCP port for incoming connections from the controller. It will get one packet that is specified below. It should run the routing algorithm and generate the flow table (also specified below) for the packet switch that is listed in the packet. It should send this back to the controller and begin listening for a new adjacency matrix.

Adjacency Matrix Packet

First, let's talk about what an adjacency matrix is. Let there be a set M of vertices in the graph with a m members. The adjacency matrix is then an $m \times m$ array of 0's and 1's that describes the connectivity of the graph. If any two vertices $i \in M$ and $j \in M$ are connected by an edge, the value in cell i, j of the matrix is a 1. Otherwise it is a 0.

We're going to modify this very slightly so that we can concisely communicate port numbers to the Routing program. A value of 0 still means that there is no direct connection between the two vertices. A non-zero value indicates the port number that would be used to connect to that other vertex.

When the controller sends an adjacency matrix to the routing program it should have the following format, all in ASCII characters for simplicity:

```
SourceVertexID, NumVertices
0 = ADDRESS
1 = ADDRESS
...
NumVertices-1 = ADDRESS

x, x, x...
x, x, x...
.
.
.
x, x, x...
```

The file explicitly lists the IP addresses of all the devices (including the switches). This is necessary information to create the flow table. In this paradigm, the only meaningful difference between a switch and a host is that the switch will have multiple ports and the host will only have one. (Realistically, a host would never send a packet directly to a switch and would never need a route to one.)

The number of rows and columns is consistent with the value sent in `NumVertices`. When executing Dijkstra's Algorithm, the routing program should start at the node specified in `SourceVertexID`.

Your program should verify that `SourceVertexID` is a valid node.

The Flow Table Packet

The flow table returned to the controller will be a simple ASCII file of addresses and egress port numbers. (Yes, OpenFlow has more expressive rules, but we're only using destination-based forwarding anyways.) It should look something like:

```
141.219.10.20, 4
10.0.1.19, 2
...
```

The Controller Program

The Controller program has the simplest logic, but the most complex network handling. It needs to periodically open up a TCP connection to the Routing program and send an adjacency matrix. It needs to then listen for the incoming flow table, open a TCP connection to the Switch program and send the flow table on. The other thing it needs to be doing is listening for update events from the Switch program. It may receive these updates to indicate that a particular port has come online or has gone offline. Receiving these update packets are what trigger a new routing request. An initial adjacency matrix is provided with this assignment which can be used when the Controller starts up.

Update Packet Format

This packet will be very short and simple:

```
VertexID, ADD/DELETE, PortNum, IPAdrs
```

This indicates that the switch with ID number `VertexID`, is reporting that port number `PortNum` has either been `ADD`ed or `DELETE`d from the switch.

The Switch Program

The Switch program is interactive. A user will start it up after the Routing and Controller programs have been started. Whether it is a command line argument or part of the interaction with the user, the switch should determine its vertex ID. Then, the user will be prompted to `FORWARD`, `ADD`, or `DELETE` a port. The program will respond to each command appropriately:

- A `FORWARD` request should be completed with a properly formatted IP address. The program should indicate the port to forward the datagram out.

- A `ADD` request should be completed with a port number and the IP address of the connected device. The switch will then send an update packet to the controller with this new information. It should print out a confirmation that the new flow table has been received.
- The `DELETE` command works the same as `ADD` except that it removes a port from the switch. No IP address is needed here.

You are free to create 1-character aliases for each of these commands (`F`, `A`, and `D`).

When the program starts up, it will not have a flow table loaded, so it must request one. We will co-opt an `ADD` message and the fact that port 0 is never used to send an `ADD 0` update message to the Controller. This should not be interpreted as a "real" update, but rather an initial table request.

Example

Consider that we want to work with the switching network in Figure 1. We'll focus on switch B in the middle.

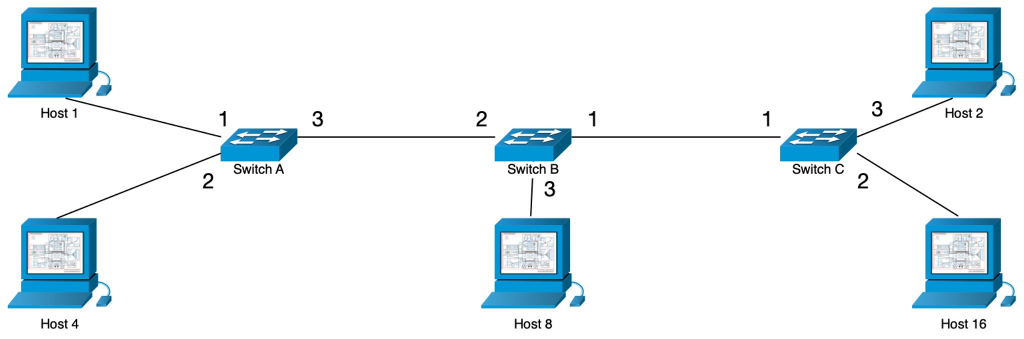

Figure 1: A Small Switching Network

Since I'm reusing an image that doesn't have all the same information we need for this assignment, let's use the following mappings.

Original Name	**Vertex Number**	**Address**
Host 1	0	10.0.0.1
Host 2	1	10.0.0.2
Host 4	2	10.0.0.3
Host 8	3	10.0.0.4
Host 16	4	10.0.0.5
Switch A	5	10.0.0.6
Switch B	6	10.0.0.7
Switch C	7	10.0.0.8

The adjacency matrix packet sent to the controller would look like this:

```
6, 8
0 = 10.0.0.1
1 = 10.0.0.2
2 = 10.0.0.3
3 = 10.0.0.4
4 = 10.0.0.5
5 = 10.0.0.6
6 = 10.0.0.7
7 = 10.0.0.8

0, 0, 0, 0, 0, 1, 0, 0
0, 0, 0, 0, 0, 0, 0, 1
0, 0, 0, 0, 0, 1, 0, 0
0, 0, 0, 0, 0, 0, 1, 0
0, 0, 0, 0, 0, 0, 0, 1
1, 0, 2, 0, 0, 0, 3, 0
0, 0, 0, 3, 0, 2, 0, 1
0, 3, 0, 0, 2, 0, 1, 0
```

The flow table returned from the Routing program would look like:

```
10.0.0.1, 2
10.0.0.2, 1
10.0.0.3, 2
10.0.0.4, 3
10.0.0.5, 1
10.0.0.6, 2
10.0.0.8, 1
```

Note that there is no route to itself, so there's no forwarding rule. This network isn't particularly interesting to route, but is a small, easily digestible example.

Now when we run the Switch program, it might look like this:

```
>> F 10.0.0.1
Forwarding packet out port 2.
>> F 10.0.0.7
No rule to match for packet.
>> exit
```

Deliverables

You should submit your three programs to Canvas. If you wrote the code in C, a makefile would be appreciated.

Each program is worth 75 points.