Homework 3: Networking
CSE 534, Spring 2018
Instructor: Aruna Balasubramanian
Due date: 4/23/2017, 9.00pm

In this homework you will acquire hands on experience in network design and implementation.You will need to do your homework using Mininet and Quagga. If you are not familiar with Mininet, you are strongly advised to start early.

First, set up Mininet and Mininext

Setup (0 points):

Your set up will be on Ubuntu. You will need to download a virtualization software such as a Virtual box or VMWare (it is recommended that you download this virtualization software even if you are using Ubuntu). Mininet works directly on the virtual box.

Once you have this, the mininet installation is as follows:

1. Download mininet version 2.1.0 from https://github.com/mininet/mininet/wiki/Mininet-VM-Images. [Only use 2.1.0 version]

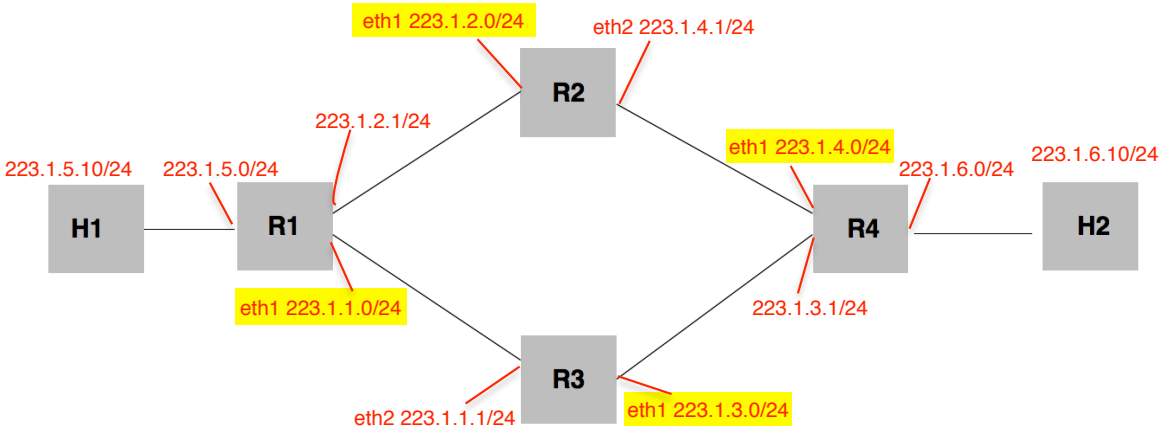
2. Next you need to download and install the MiniNExT extension. This extension lets you virtualize file system and allows you to build complex networks over Mininet.

3. Get familiar with Mininet by going over the tutorial: We recommend you go through http://mininet.org/walkthrough/

4. Install Quaga using sudo apt-get install quagga. Read up on Quagga from www.nongnu.org/quagga/

PART A (30 points):

A1. Create your own topology as below.



To do this, copy the MiniNExT *examples* folder (/home/mininet/USC-NSL-miniNExT-75c2781/examples/quagga-ixp) into your own folder. Lets call your new path “myfolder/” for convenience.

Change the myfolder/topo.py to reflect the above topology. You do not need to add any switch fabric, since hosts(H1, H2) and routers (R1, R2, R3, R4) are connected directly. You also need to assign IP addresses. Keep in mind that you will have to create separate subnets.

Start the configuration by running sudo python ./start.py

Check that the nodes are created using the “nodes” command.

Check the routing table at each node.

Submit: (a) the topo.py file you created, (b) the network topology figure, but this time including the IP addresses (with subnet information).

A2. Create static routes

Configure each node so that H1 can ping H2.

To do this, you need to change the IP forwarding variable to 1. You will then need to create a static routing table at each node.

Submit: (a) the routing tables at all nodes (as a screen capture) and explain what you did to configure them correctly. (b) provide the trace route output that gives the path between nodes H1 & H2.

PART B (30 points):

B1:

First remove the static routes.

In this part you configure R1, R2, R3 and R4 as a RIP router. Figure out how to do this by changing the RIP daemon. You will need to change the config files.

Submit: (a) the set of commands you used for the configuration in the correct order. (b) Explanation of each command.

B2:

Run the RIP daemon on each router and host. Estimate the convergence time: this is the time it takes for H1 to be able to ping H2.

Submit: (a) the routing tables at each node (both the kernel and the Quagga routing table), (b) the traceroute output that gives the path between nodes H1 & H2, and (c) the time taken for the ping, and (d) the convergence time.

B3:
Bring down **R1-R2 or R1-R3 (**whichever is on your current path from H1 to H2).

Estimate the time from when the link went down to when the connectivity was re-established.

Submit: (a) how you got the link to go down, (b) the time it takes for connectivity to be established, (c) provide the traceroute output that gives the new path between nodes H1 & H2.

Part C (40 points)

RIP-lite: Write your own "application" layer routing protocol that implements distance vector routing uses Bellman-Ford algorithm. To do this, you create a TCP connection with your next hop neighbor. Implement Bellman-Ford algorithm on top of this TCP connection. You will periodically exchange routing information with your neighbors. You will need to use socket programming for this.

This is an application level routing protocol because you are not expected to change the routing table. You maintain all information in the application-layer.

Assume the following weights between the links (we are assuming that each node including the hosts run the routing protocol, this is not usually the case):
R1-R2 = 10
R1-R3 = 6
R2-R4 = 4
R3-R4 = 5
H1-R1 = 2
R4-H2 = 2

Your routing protocol should run continuously in the background; if the weights change, the routing protocol needs to update the routes. Assume the weights of your own neighbors are stored in a file that you periodically check.

You can write your program in Java, Python, or C. Make sure that these languages are supported by mininet.

After you write your protocol, answer the following question:

C1. Run your protocol and get the time taken for the protocol to find the shortest path between H1 and H2.

Submit: (a) the routing protocol code. (b) The time taken for your protocol to find the shortest path with explanation of how you estimated this (c) the application layer routing table at each node.

C2. Assume that the weight of the link R1-R3 changes from 6 to 1. Estimate the time taken for the protocol to converge.

Submit: (a) The time taken for the protocol to converge. (b)) the application layer routing table at each node.

C3. If one of the links has a negative weight, explain how you handle this situation.

Useful tools

sudo mn –c: for cleaning the previous mininet configurations

sudo apt-get install: traceroute to install traceroute

sudo apt-get update

sudo apt-get install openjdk-7-jdk: to install Java