CSCD58 Assignment 2

Due: April 1, 2018. 11:59pm

Now that we're comfortable with socket programming, we're going to move a little further down the protocol stack. This assignment will have you building a simple router and testing various routing algorithms. You will work in groups of 2-3 for this assignment. Be sure to declare your groups on MarkUs prior to submitting your files. Members of the group must all be able to attend the same tutorial for the code presentation. Please note that you must work in groups for this assignment, you cannot work individually. You can use any programming language you wish, but you must be able to demonstrate your work to the TAs on MiniNet during the demo sessions.

Simple Router

In order to simplify things, we won't be working with actual IP packets (this is do-able, but involves a lot of bookkeeping to manage ethernet packets, headers, etc). Instead, we'll 'cheat' and just allow direct socket connections. This will make it easier to work with your MiniNet to simulate more complicated networks and actually observe the information flow. We will also skip over DNS and assume that all systems have simple static IP addresses which are manually configured a priori.

Simple End System

Your fist task is to build a simple end system with a single IP address that connects to a single node in the network. These end systems have three basic functions (you may add more if you wish):

- `initialize`: Given the end system's IP address, the end system will become active, open a socket to its next hop connection and send a simple message to the IP address 255.255.255.255 with TTL=0 in order to broadcast its existence.
- `send`: Given a destination IP address, a text message and a maximum number of hops (TTL), the end system will attempt to send the message through the network.
- `receive`: If an end system receives a message, it should display that message (and any other relevant information) to its output

Simple Router

Simple routers are nodes in the network that can connect to multiple end systems, and allow those end systems to send messages to one another by appropriately forwarding data. We're going to leave the details of the forwarding table implementation up to you, but the goal is to emulate a real router, so you should have a pretty good start on how to do that.

Forwarding Tables

To start with, the simple router won't have any information on its forwarding table, because it doesn't know anything other than the IP addresses of its own interfaces. Upon receiving messages from end systems, the router will start to build up a forwarding table, eventually learning the IP addresses of all of its immediate neighbours. Once this is done, it should be able to forward messages from one end system to another. You've successfully created a simple network. At this point, you should be able to type messages into one end system and see them appear on another.

A Multi Network Router

Now that we can communicate within a simple network, we need to figure out how to connect our networks together (after all, the Internet is all about networks of networks). When a router connects to another router, it must first initialize in a manner similar to an end system. But once the initial connection is made, the router now needs to know which packets should be sent to the new router.

Each router should have at least the following functionality:

- `get_forwarding_table`: retrieve the forwarding table for this router in a format that can be used for routing algorithms
- `set_forwarding_table`: set the forwarding table for this router given the data in some appropriate format
- `print_forwarding_table`: this will be useful in debugging, and also will allow us to easily see what your code is doing as the system updates
- `advertise`: broadcast a message to neighbouring routers with the destinations that you can reach and the distance within which you can reach them
- `update`: take in an advertisement from a neighbouring router and update your routing table accordingly

OSPF

In order to deploy a routing algorithm, you must create a `monitor` node, which is connected to all routers on the network. This node will periodically (you can decide when this should be triggered) get the topology of the network by asking all routers for their routing tables (hint: you may want to include some information in the table to show immediate neighbours vs distant links) and uses Dijkstra's algorithm to set new routing tables for every router based on minimizing hops.

RIP

Another method to update forwarding tables is RIP, which uses distance vector advertisements sent between routers to update state tables. No extra topology is required for this, but advertisements and updates must be able to propagate through the network appropriately.

Report

After implementing your code and testing your network, you will need to write a report that covers the following topics:

- Implementation: How your code works and how the various pieces fit together
- Design Decisions: What data structures and algorithms you used, and how they effect the efficiency and usability of your code
- Testing: How you tested your code, the various topologies used and how you decided whether your code was working or not
- Performance: An evaluation of the two routing algorithms in various networks, which works better and why and under what conditions each is optimized. This section should have a conclusion with details what routing algorithm makes the most sense for this network, and under what circumstances your decision would be altered to the other algorithm.

Extra Credit

You can complete any of the following to potentially get more than 100% on the assignment. If you complete an extra credit component, you must include it in your report as well as your presentation.

- Implement sub-nets, including variable length masks and subnet-level broadcasting [5%]
- Allow nodes to disconnect from the network and re-connect elsewhere with the same IP [5%]
- Have non-uniform link costs based on traffic intensity [5%]
- Implement poisoned reverse (must have completed non-uniform link costs) [5%]
- Simulate changes in bandwith of links and broken/dropped connections [5%]

Submission

By the deadline, you must submit the following to MarkUs:

- `A2.tar.gz` - A compressed file containing the three servers and any additional files necessary to run them
- `A2.pdf` - Your report file

In your tutorial following the submission deadline, you will present your report and your code to one of the TAs. You will have approximately 5 minutes to formally present your code and your findings, followed by a brief period where you TA may ask you questions for clarification.

Evaluation

Your assignment grade will be calculated as follows:

- 25% Working file submission
- 25% Code quality and documentation
- 25% Report
- 25% Presentation + Demonstration of code