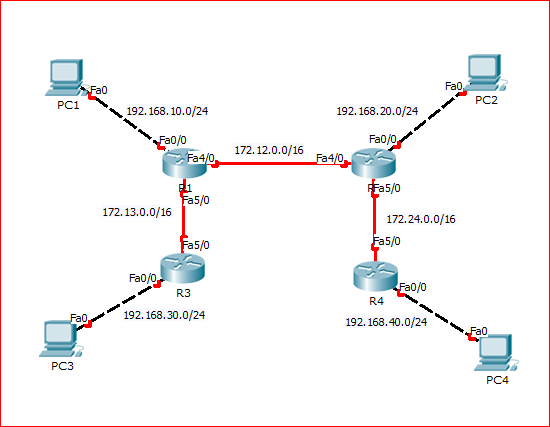
CSE322

Assignment 3: Implementation of a distance vector routing protocol

**Class Lecture and assignment specification**

**1. Understanding routing protocol and routing table**

-Open the file DemoRoutingProtocol.pkt.



-Delete all configurations from all the routers.

-Select one router, e.g., R1 and see its routing table is empty.

-See how configuring interfaces of R1 populates the routing table (initialize) with directly connected network.

-See how enabling RIP on directly connected routers update each routing table.

-Use simulation mode to inspect RIP packet format and content.

-Use simulation mode to inspect how data packet forwarding works.

-Use simulation mode to inspect how ARP works.

**2. How does a router work?**

2.1 Events and Actions

e.1: Interface up and configured

a.1: Update routing table, insert the new directly connected network

e.2: Interface down/shutdown

a.1: Delete corresponding row from routing table

a.2: Delete all other rows that use this interface

e.3: Routing protocol configured

a.1: Send routing update (routing table) to configured interfaces

a.2: Start update timer

a.3: Start invalidate timer for each (RIP Learned) row in routing table

e.4: Received route update packet from neighbors

a.1: Update routing table with newly learned networks

e.5: Received data packet (on any interface/port)

a.1: Lookup routing table for destination network to find exit interface

a.2: Forward packet through the exit interface

(Note: In our implementation 'data packets' are distinguished from 'route update packets' by using a special multicast address (224.0.0.0) in the destination IP field of the 'route update' packet.

e.6: Update timer expires

a.1: Send route update to configured interfaces

a.2: Restart timer

e.7: Invalidate timer expires

a.1: Delete corresponding row from routing table

2.2 How does routing tables get populated?

(i) During interface configuration

(ii) Through static route configuration

(iii) Through running dynamic routing protocols.

2.3 What are the data structures used?

D.1: Routing Table

D.2: ARP Table

**3. A Simple Network Emulator**

-Written in java

-Emulates basic devices (with very functions): PC, Switch, Router

-Implements protocol stack (layered architecture)

-Does not support GUI

3.1. Files

**3.1.1 Source Files**

C.1: H/W Related

PC/Host ---> SimHost

Switch ----> SimSwitch

Router ---> SimRouter

c.2: Protocol Stack Related

Physical Layer -- > SimPhy

DataLinkLayer ---> DataLinkLayer

Routing Protocol ---> RoutingProtocol

c.3: ConnectionDaemon -> Maintains the topology

**3.1.2 Configuration Files**

F.1: Config.txt-> Contains topology info and configuration values

F.2: ArpTable.txt (ARP resolves IP Address to MAC address)

3.2 Folder Structure

-Supplied folders has the following structure:

F1: Assignment3

F.1.1: Src -> Contains all the source .java files; particularly the skeletal code for 'RoutingProtocol'

F.1.2: Config -> Contains two files: Config.txt and ArpTable.txt

F.1.3: Lecture+Instruction -> Contains the files- (i) 'DemoRoutingProtocol.pkt' that was used in the class for demonstrating routing protocol and other related issues and (ii) this documentation. (iii) a flow chart for the RIP process.

F.1.4: Executables -> Contains the executables corresponding to the class files. If you want see how our version of the program works, run the supplied .exe files. Note that you need to have JRE version 7 installed to run these files.(Disclaimer: Check the .exe files for viruses before use. We would bear no responsibility in case they cause any harm to your computer.)

F.1.5: Javadocs -> Contains documentations of the source files, generated by javadoc.

F.1.6: Batch Files -> Windows batch files/scripts for convenience of running the emulator.

3.3 How to run

S.1. Run ConnectionDaemon. Open a command prompt, go to the folder Executables and run the file ConnectionDaemon.exe

S.2. Run devices: PC, Router, Switch. Open a command prompt, go to folder Executables and run the corresponding device's .exe file. The device name has to be supplied as the parameter. For example, if you want to run a host with name H1, type the following in the prompt: SimHost H1.

You must make sure that the device name must match with the one mentioned in the Config.txt file.

S.3. To verify whether routing protocol is working, i.e, the network has converged, go to any of the Host's console and issue the ping message to all other hosts and router interfaces. The syntax for ping like service in this emulator is: ipaddress:Message i.e., just write the destination IP, then a colon (:) and the message that will be displayed in the destination machine to make sure the message has reached successfully. If no route found to destination, the issuing host's console will display 'Destination Unreachable' message upon receiving feedback from the router that dropped that packet.

S.4. You may also emulate channel noise in the given emulator. To do so, run ConnectionDaemon in step 1 with either or both of the following two arguments: Arg 1. a number specifying the frame drop probability and Arg 2: a number specifying the frame error probability. For example, if your run: ConnectionDaemon .5 .5 where both the arguments are set to 1, for each transmitted frame in the network, 50% times it will be dropped and 50% times some of its bits will be distorted.

3.4 Inside the code

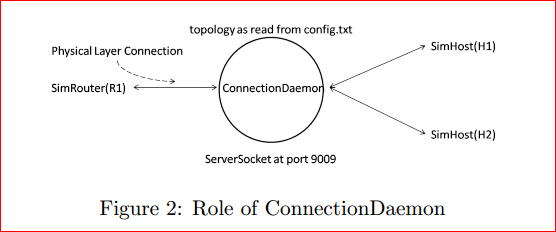
3.4.1 How the topology is maintained?

-The 'ConnectionDaemon' runs a server socket on port 9009 of local host. It reads the complete topology from the file Config.txt to determine the connection between devices.

-Each device (SimHost, SimSwitch and SimRouter) has a device id which supplied as a command line argument while running the device's class file.

-Each device has a SimPhy instance representing the bottom of protocol stack. SimPhy opens a client socket with 'ConnectionDaemon' and sends the corresponding device's id.

-ConnectionDaemon upon receiving the device ids from all devices, breezes the each device pair by linking the input stream from one side to the output stream to other side and vice versa.



3.4.2 How each device is configured?

-Each device has a device id which is supplied as command line argument (as mentioned above).

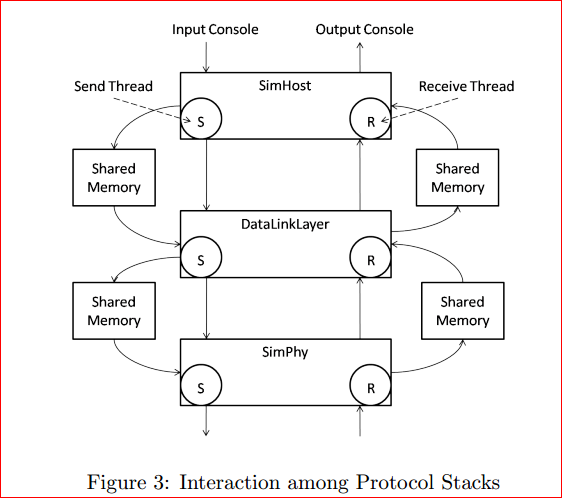
-Upon start up, each device reads the Config.txt file and load the configuration parameters prefixed by the device's id.

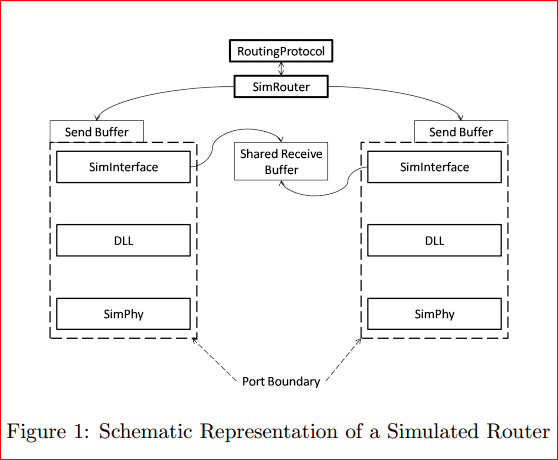
-Though in practical Ethernet networks, IP to MAC mapping is resolved through running ARP protocol, to keep things simple (in this simple emulator), IP-MAC mapping is stored statically in the ArpTable.txt file. A device requiring IP to MAC mapping (such as hosts and routers) loads this ArpTable.txt file to populate its ARP table instead of using dynamic ARP.

3.4.3 How does the protocol stack inside each device work?

-Each device maintains the layered protocol architecture. At the lowest level, there is SimPhy, representing physical layer. On top of each 'SimPhy' there is 'DataLinkLayer'. Since we will be using layer 3 communications only, all the layers above layer 2 will be represented by the main thread of the respective device file, e.g., SimHost, SimSwitch and SimInterface (representing interfaces of SimRouter).

-Between any two consecutive layers, there are two shared buffers: 'send buffer' and 'receive buffer'. Correspondingly, in each layer there are two threads: 'read thread' and 'write thread'. In any layer, the 'read thread' continuously reads from the 'receive buffer' shared with its immediately lower layer, process the data and write, if required to the 'receive buffer' shared with the upper layer. In the same way, the 'write thread' in any layer continuously reads from the 'send buffer' shared with its immediately upper layer, process the data and write, if required to the 'send buffer' shared with the lower layer.





3.4.4 Protocol Data Unit (PDU) formats

P.1: RIP Update format -> *to be implemented*

P.2: IP Packet format -> see 'utility.java' file

P.3: Frame Format -> see 'utility.java' file

**4. Requirement specification for the assignment**

4.1 Tasks

T.1 Tasks in Layer 3

-Implementation of a routing protocol such as Distance vector routing.

-Implementation of a router's forwarding Function.

-Fill up codes in the given stub file: RoutingProtocol.java

T.2 Tasks in Layer 2

-Implementation of frame checksum calculation and checksum verification.

-Implement the the following functions in the class Frame of the file Utilities.java: (i) calculateChecksum for checksum calculation, and (ii) hasChecksumError for checksum verification.

-To verify whether checksum is working, run ConnectionDaemon with non zero error probability is the way as described in section 3.3, step S.4.

T.3 Tasks in Layer 1

-Implement bit-stuffing in physical layer. Note that each frame is delimited by the special bit patter '01111110' (i.e., decimal 126). The goal of bit-stuffing is to ensure that this special bit sequence does not occur inside the frame. -Implement the following functions of the class SimPhy in the file Simphy.java: (i) bitStuff for bit stuffing and (ii) bitDestuff for bit de-stuffing.

--To verify whether bit stuffing and de-stuffing is working, create a topology with two hosts and then send a message with the character '~', e.g., “Hello~~world” from one host to another in the way as described in section 3.3, step S.3.

T.4 Implementation of suitable Graphical User Interface (GUI).

T.5 Report on the assignment

T.6 Optional and bonus

- Implement REQUEST message on start up of a router. For reference on how the request message works, see <http://en.wikipedia.org/wiki/Routing_Information_Protocol>

-Communication with our implementation (will not work if different protocol implemented).

-Improvement of the given stub code.

-Implementation of Address Resolution Protocol (ARP).

-Anything relevant you want!

4.2 Checkpoints

C1: To be completed within 11 th week

-Parts of T1: (i) Implementation of routing table, and (ii) Routing to directly connected network

-Task T2, T3 and T4

C2: To be completed within 13 th week

-Complete implemenation of T1

-Tasks T.5: Report and T.6: Optional Components

4.3. Submission

-For this assignment, you can form groups of maximum size two; this is an option not mandatory. You may choose to do it individually and independently.

-To keep it fair, all the groups will submit within the same window; however evaluation will be done in consecutive weeks.

-We assume all of you have a GMAIL account. To submit your file, follow the following steps.

S1. Create a folder and name it with your student number. In case of grouping, write the student numbers of all the two members separated by the ‘+’ sign, e.g., 1005001+1005002. We will refer to this folder as ‘parent folder’ onwards.

S2. Copy all your files and folders in the ‘parent folder’. Note that unlike the previous submission during assignment 1 where you were only asked to upload the source files, here you are required to upload all your project files.

S3. Upload the parent folder with all its contents in ‘Google Docs’ and then share it with the following e-mail address: <kmshahriar@gmail.com>

4.4. Evaluation

- We will download all your uploaded submissions after the deadline is over for each checkpoint. Checkpoint wise submission deadline is tabulated below:

|  |  |
| --- | --- |
| Checkpoint | Deadline for all sections |
| Checkpoint 1 | Monday 2:00 pm of 11th week |
| Checkpoint 1 | Monday 2:00 pm of 13th week |

- During evaluation, you will be given a copy of your submitted code that you will use for demonstration to evaluators/teachers.

-Like the previous two assignments, an evaluation sheet, showing the mark breakdown will also be provided for your self-assessment and convenience.