University of Dhaka
Department of Computer Science and Engineering

3rd Year 1st Semester
Session: 2017-18

CSE - 3101: Computer Networking

Programming Assignment 2: Distance Vector Routing Algorithm Implementation

Submitted To :
Upama Kabir, Professor
Department of Computer Science and Engineering
University of Dhaka

Submitted By :
Amit Roy (JH-40)

Date of Submission :
April 30, 2018

Assignment Name: Distance Vector Routing Protocol Implementation

Assignment Details: In this assignment, we implemented the distance vector routing protocol where we designed a program that will run at each router in a specified network. At each router the input to our porgram is a set of directly attached links and their costs. The program at each router does not know the entire network topology. The program will report the cost and next hop for the shortest paths to all other routers in the respective network. We extended the basic protocol to handle node failures and implemented poisoned reverse to solve the count to infinity problem.

i) Solution Design Description:

- **Language:** C/C++
- **Files:**
  - **dv_routing.cpp:** C++ file to run the program at each router. In this file we parse data from command line arguments, create own distance vectors, initialize own socket, bind socket with localhost with current router port no. We also created a child process from this program to run it parallely with the parent process using the fork() function.

    In a C or C++ program, fork() can be used to create a new process, known as a child process. This child is initially a copy of the the parent, but can be used to run a different branch of the program or even execute a completely different program. After forking, child and parent processes run in parallel.

    Then, we periodically wake up parent process and update own routing table and also periodically listen for advertisements and perform periodic tasks based on the received packet header.

    Packet Headers: A packet typically contains a header which determines for which purpose it will be sent to others. Header contains the type of the packet, the ID of the packet source router, the ID of the packet destination router.

BROADCAST: Packet contains a router's distance vector, used to update neighboring routers' routing tables. This packet is sent to the router's neighbors periodically.

WAKE_UP: Each router has a child process that periodically sends this packet to its parent process, waking up the parent from listening for packets. Through this method, the parent process is able to perform periodic tasks like sending its distance vector advertisement to neighboring routers and checking whether or not neighbors' timers have expired (to determine if they are invalid).

NODE_CLOSED: Alerts the router that one of the routers in the network has become invalid, causing this router to reset its routing table to its initial configuration (loaded from file), and also to tell its neighbors to reset their routing tables as well.

- **distance_vector.h:** Declares a class DV which takes neighbour information file name and self router name as input. The constructor parse neighbour router name, neighbour router port no and the link cost for each directly connected neighbour and store them in the neighbour list. It also maps router name with port no and port no with router name.

Functions:

- **getIndexOf(char router) const :** Returns index of router
- **char getSelfNameOf(int index) const :** Returns Name of Router
- **int getPortNoOf(char router) :** Returns Port No. of Router
- **dv_entry *getEntries() :** Returns Distance Vector Entry List
- **int getSize() const :** Returns Number of Active Routers
- **char getSelfName() const :** Return Self Router Name
- **int getSelfPort() :** Return Self Router Port No
- **sockaddr_in myaddr() const :** Return Self Router Address
- **dv_entry getDestinationDistanceVector(const char dest) const :** Return Distance Vector of Destination

- **vector<node> neighbors() const :** Returns Neighbours List

- **bool timerExpired(node &n) const :** Checks if timer is expired or not
- **void initializeMyAddress(int portno) :** Initialize Self Router Address and Port No
- **void startTimer(node &n) :** Start Timer of a Router
- **void reset(char dead) :** Reset routing table and routing list when a router is failed/killed.
- **void update(const void *recvBuffer, char source) :** Update Current Router's Distance Vector based on Received Advertisements,Print Distance Vector if Current Router's Distance Vector was Changed

- **int min(int originalCost, int selfTomiddleNodeCost, int middleNodeToDestCost, char originalName, char newName, bool &updated) const :** Minimum cost finding and updated flag set
- **void print(dv_entry dv[], char name, string msg, bool timestamp) const :** Print Distance Vector, Format: source, destination, cost to destination

- **node.h:** Contains basic information about a single router. Each router maintains a vector of node structures, representing the information for the router's neighboring routers. Such information includes the ID of the neighbor, the port number to reach the neighbor, the timer (to determine if the router is still valid), and the neighbor's socket address.

- **dv_entry.h:** Each router maintains its own routing table implemented as an array of dv_entry structures. Each dv_entry contains the cost of the shortest path to a particular destination and the next hop along the shortest path.

- **config.h:** Contains the information like MAX_ROUTERS, BUFFER_SIZE,selfName, myPortNo, Header Type of Packet

- **packet_header.h:** Contains some functions to create packet, extract data from packet, extract header from packet, wake up periodically to multicast advertisements and broadcast advertisements to all neighbours.

- **Functions:**
  - **void *makePacket(int type, char source, char dest, int dataLength, void *data):** Create Packet with Headers and Data
  - **void *finddata(void *packet, int length) :** Extract Data from Packet.
  - **packet_header findpacket_header(void *packet) :** Extract Header from Packet
  - **void sendToSelf(DV &dv, int socketfd, int type, char source = 0, char dest = 0, int dataLength = 0, void *data = 0) :** Wake up periodically to send advertisement.
  - **void sendToAll(DV &dv, int socketfd) :** Broadcast advertisement to all neighbours.

- **Makefile:**

```
CC =g++
BIN_F     ILE = dv_routing
SRC_FILE = dv_routing.cpp


all:
     $(CC) -o $(BIN_FILE) $(SRC_FILE)

clean:
     rm  $(BIN_FILE)
```

- ## Start-router.sh:

```
#!/bin/bash

#Assumed first router name as ‘A’ and port no. 2000
#Executed Other router name in alphabetical order
#Assigned port no. sequentially
#To assign different router name and port no. change in start-router1
#Run From Terminal ./start-router or ./start-router1

first1=config
second1=.txt
char=A
inputfilename=configA.txt
port=2000

for ((i=0;i<26;i++)) ;
do
      if [ -f "$inputfilename" ] ;
      then   xterm   -title   "$char"   -hold   -e   ./dv_routing   "$char"   "$port"
      "$inputfilename" & sleep 1
      char=$(echo "$char" | tr "0-9A-Z" "1-9A-Z_")
      inputfilename=${first1}${char}${second1}
      port=$((port+1))
      Fi;
done;
```

- ## Start-router1.sh:

```
#!/bin/bash

xterm -title A -hold -e ./dv_routing A 2000 configA.txt  & sleep 1
xterm -title B -hold -e ./dv_routing B 2001 configB.txt  & sleep 1
xterm -title C -hold -e ./dv_routing C 2002 configC.txt  & sleep 1
xterm -title D -hold -e ./dv_routing D 2003 configD.txt  & sleep 1
xterm -title E -hold -e ./dv_routing E 2004 configE.txt  & sleep 1
xterm -title F -hold -e ./dv_routing F 2005 configF.txt  & sleep 1
```

- **Usage:**

  First compile the dv_routing.cpp. Open a terminal and type

  $make clean
  $make

  I've tested the program according to the given topology with the given config files. If user wants to run the protocol in a different network topology then replace the network topology files of the project directory.

  According to the given format of the assignment, we assumed port no. of router A is 2000 and other routers name and port number increased lexicographical order. So give permission and then run the script start-router.sh

  $chmod 777 start-router.sh
  $./start-router.sh

  Or, if you want to change the router name, port number and config file name then simply change these in script start-router1.sh, give permission and then run it.

  $chmod 777 start-router1.sh
  $./start-router1.sh

If you want to kill a router then simply select the corresponding router window and press the close icon or **CTRL-C** from keyboard and wait untill the routers update their distance vectors and converges.

- **Difficulties**
  - **Simultaneous Packet Sending and Receiving:**

Most of the packets are sent in response to the received packets. But, routers need to send periodic advertisements to his neighbours regardless of whether or not it has received a packet. A simple receive-and-send loop will not solve this problem, because if a neighbour is unable to send packets to his neighbours then it's neighbour will never receive any message and also never send any packets.

We solved this problem by using the fork() function of C/C++. We maintained a parent process and a child process for each router. The parent process uses a receive-and-send loop and the child process sends a packet to parent process periodically. In this way, parent process wake up periodically and send packets to its neighbours.

  - **Updating Routing Table:**

Each packet contains the distance vector of the sender. When a router receives a packet it updates its routing table by comparing the cost of its current shortest cost to each destination with the cost of the path to the destination going through the source router that sent the packet. The cost calculated summing the cost to reach the sender with the cost from the sender to the destination. Through this update scheme of routing table, each routing table in the network converge to an optimum set of routes.

  - **Invalid Neighbour Router Detection:**

A router maintains a timer for each of its neighbouring routers. After receiving a packet from a neighbouring router, the receiving router resets the timer of the corresponding sending router. Since each valid router sends a packet periodically, the receiving router detects that a neighbour become invalid if it hasn't sent any packets in a while. This happens when a router is killed. If a packet is

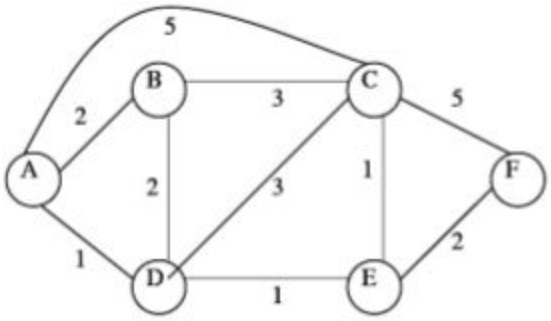
received from an invalid neighbour, the neighbour is treated valid again.

- **Alerting the rest of the network about a router failure:**

When a router detects that one of its neighbouring router is failed it broadcast this information to rest of the network. Every other router in the network reset its routing table to its initial configuration (config.txt). We set a limit on the number of hops a “reset” packet can travel to prevent broadcast storms. A “reset” packet contains a counter in its header which denotes the hop limit. Every time a “reset” packet leaves a router the counter in the packet header is decremented by one. When a “reset” packets counter reaches 0, it is no longer forwarded. A “reset” packet counter is initialized to n-2 (excluding sending router and failed router), where n is the number of routers in the network so that every other counter in the network receives the packet at least once. This is because of the worst case scenario. Consider the worst case, a network where routers R1, R2, ... Rn are connected in a single file line. When R1 becomes invalid, R2 detects that R1 is invalid, and sends a packet to R3 with counter n-2. We can see that when the packet has propagated to Rn, the counter will equal 1. So all routers in the network receive the packet at least once, and the packet will not propagate forever.

$R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow$ -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- $\rightarrow R_n$

- **Routing Tables (after convergence):**
    - Given Network Topology:




```
Router A:
shortest path to node B: the next hop is B and the cost is 2.0
shortest path to node C: the next hop is D and the cost is 3.0
shortest path to node D: the next hop is D and the cost is 1.0
shortest path to node E: the next hop is D and the cost is 2.0
shortest path to node F: the next hop is D and the cost is 4.0

Router B:
shortest path to node A: the next hop is A and the cost is 2.0
shortest path to node C: the next hop is C and the cost is 3.0
shortest path to node D: the next hop is D and the cost is 2.0
shortest path to node E: the next hop is D and the cost is 3.0
shortest path to node F: the next hop is D and the cost is 5.0

Router C:
shortest path to node A: the next hop is D and the cost is 3.0
shortest path to node B: the next hop is B and the cost is 3.0
shortest path to node D: the next hop is E and the cost is 2.0
shortest path to node E: the next hop is E and the cost is 1.0
shortest path to node F: the next hop is E and the cost is 3.0
```

```
Router D:
shortest path to node A: the next hop is A and the cost is 1.0
shortest path to node B: the next hop is B and the cost is 2.0
shortest path to node C: the next hop is E and the cost is 2.0
shortest path to node E: the next hop is E and the cost is 1.0
shortest path to node F: the next hop is E and the cost is 3.0

Router E:
shortest path to node A: the next hop is D and the cost is 2.0
shortest path to node B: the next hop is D and the cost is 3.0
shortest path to node C: the next hop is C and the cost is 1.0
shortest path to node D: the next hop is D and the cost is 1.0
shortest path to node F: the next hop is F and the cost is 2.0

Router F:
shortest path to node A: the next hop is E and the cost is 4.0
shortest path to node B: the next hop is E and the cost is 5.0
shortest path to node C: the next hop is E and the cost is 3.0
shortest path to node D: the next hop is E and the cost is 3.0
shortest path to node E: the next hop is E and the cost is 2.0
```

ii) Message Format/Packet Format:

We didn't followed the message format of DV/RIP. Message is created in the makePacket function of the packet_header.h header file.

File Name: packe_header.h
Line No: 24~45