Routing Protocols

Project assignment 3

Modern Networking concepts

CSE 589
Spring'14

Vijay Manoharan *manohara@buffalo.edu*

Table of contents

1. Introduction…….. 2
2. Data Structures ……..…….. 3
3. Functions…….…… 5
4. Program working and flow……..…… 8

Vijay Manoharan -----manohara@buffalo.edu

Introduction

The Project assignment is developing and implementing how the routing protocol works. The project implementation is programmed on C language. The routing protocol used in calculating the distance vector across different node is done using BELLMAN ford Routing algorithm. The developed codes are ran on different servers, namely

- Timberlake
- Nickelback
- Metallica
- Dragonforce
- Beatles

are the CSE student server of Buffalo.edu, the server exchange the message of their individual routing values using UDP socket messaging. Once the routing values are received the routing table is calculated for each update and are then updated to their own routing table. The entire project assignment is to demonstrate and implementation of how the routing table is calculated and behavior is checked to the given scenario and commands.

Vijay Manoharan -----manohara@buffalo.edu

DATA STRUCTURE

This section describes what data structures are used, why it is used and how it helps in the implementation.

- ***struct SERVERINFO***

```
struct SERVERINFO {
   unsigned short int serverid;
   unsigned char ip[50];
   unsigned short int port;
};
```

The above structure is used to store the details of the server

- serverid is used for storing the server id of all the server id given in the topology file.
- Ip and port variables are used to store the ip-address (ipv4) and port number of the server as given in the topology file.

The main use of the structure is store the details and access the details of server information whenever required. In our program we are using an array of the structure to store the details of all the server given.

- *struct distance_vector*

```
struct distance_vector {

   int connected;
   unsigned short int cost;
   int next_hopid;
   int timeout_count;
};
```

The above structure is used to store the routing information of the server. An array of structure is used created and index is referenced from the previous structure. This forms a table structure which we can refer to has the routing table. The routing information consist of following:

- cost – the least cost of reaching a given server from the server where the program is running.

Vijay Manoharan -----manohara@buffalo.edu

- Hopid- the next hop through which the given server can be reached from the server, where the program is running.
- Timeout_count- the value of time that is needed to be waited for receiving the routing information, after which the connection to that server will be assumed to broken from the server the program is running. At this point we change that particular routing information of the server to a cost of 9999.
- Connected- the value indicates whether the server has link to the given server or not.
  The main use of structure to maintain the least cost path to server and how to reach the server via the next hop.

- *struct MSG*

```
struct field {
   unsigned short int oxoo;
   unsigned short int id;
   unsigned short int cost;
   unsigned short int port;
   unsigned char ipadd[5];
};
struct MSG {
   short int No_update_fields;
   unsigned short int server_port;
   unsigned char server_ip[5];
   struct field server[5];
};
```

The above data structure MSG used in to maintain the routing message format and routing information as per the given size and format in the program specification documents. The structure also combines the 'fields' structure to store the information of other nodes. Name convention used here is similar to that of the project spec.

- *struct DVec n Cost_Matrix[n][n]*

The DVec structure is used to compute the routing table for given cost matrix, the cost matrix "costmatix[i][j]" gives the cost of reaching j from i. Using this two structure below and the costmatrix we calculate the routing table using bellman ford algorithm. The DV_Function is the function where the bellman ford algorithm is implemented.

```
struct DVec {
   unsigned short int nexthop[5];
   unsigned short int cost[5];
};
```


Vijay Manoharan -----manohara@buffalo.edu

FUNCTIONS

- *myIP()*
  - input: -nil.
  - Return –nil
  - The function is used to get the ip address of the machine where the program is running.
  - Used for- the function main aim is to identify the server id on which the program is running from the given topology file. The ip address obtain here will be used in combination of port number to identify the server id of the give topography.

- *Fileparser()*
  - input: -nil.
  - Return –integer
  - The Function is used to parse the given topology file at the command argument initially. And obtain the information of servers and details of the server.
  - The fileparser() also initializes the initial cost matrix for the given topology file.
  - It also checks for invalid entries in the topological file and returns appropriate error messages.
  - If the return value is 1 then the file is parsed successfully.
  - It also forms the initial routing table.

- *UDP_SOCKET()*
  - input: -nil.
  - Return –nil
  - It is the backbone of the entire program the function send messages timely using select socket function used inside the program.
  - The function also handles appropriate input from the STDIN and also receives the messages from all the server it is connected to.
  - The entire follow of the program is govern by this function.
  - Handles all the events required in maintaining the routing information and distance vector.

Vijay Manoharan -----manohara@buffalo.edu

- *Send_dv()*
  - input: - id ,message.
  - Return –nil
  - The function takes care sending the routing information to the neighboring nodes.
  - The function takes server-id and message as the input. The server id is the identifier to which the message must be sent and the message is the routing information to be sent.
  - The message is sent using UDP socket programming methods
- *Construct_costmatric()*
  - input: -nil
  - Return –nil
  - The function is used to calculate the initial cost matrix of the give topology file.

- *CreateMessage()*
  - input: -nil
  - Return –nil
  - The function is used to create the message to be sent to the neighboring nodes. This also takes care of the message format.
  - The function then calls the send_dv to send the corresponding message to the corresponding neighbors.
- *is_number()*
  - input: -string
  - Return –0 or 1
  - The function is used to check the given string is a number or not.
  - Returns 0 if the string is not a number. Otherwise 1 if the string is number.
  - Used in file parsing for validation of the content is a number or not.
- *word_count()*
  - input: -string
  - Return –count
  - The function is used to count the number of words in the given string.
  - The input is the string and the returned value is the count of the words present in the string.
  - Used for validation of the user commands from STDIN.

Vijay Manoharan -----manohara@buffalo.edu

- *checkipaddress()*
  - Input: -nil
  - Return –nil
  - The function is used to check the validation of any ipv4 address.
  - Used by fileparser function to check the ip validation.
  - The function returns10 if the all the ip address in the topology is valid.
  - On Error returns the line number where the ip address is wrong.
- *space_count()*
  - input: -string
  - Return –count
  - The function is used to count the number of space between words in the given string.
  - The input is the string and the returned value is the count of the space between words present in the string.
  - Used for validation of the user commands from STDIN.
- *Getindex()*
  - input: -serverid
  - Return –index
  - The function is used compute the index of the server detail for a given server-id.
  - The function takes the serverid as the input and returns the index of it from the table for the structure SERVERS

Vijay Manoharan -----manohara@buffalo.edu

Working and Flow

The entire program can be divided in the following parts:

File parsing:

The first process in the project to parse the given topology file for a given scenario. Once the initial topology is given the process is done:

- File format is given on the PA3.pdf
- The file parsing is done using fileparser().
- Check for any error in the number of nodes and number of neighboring nodes for the given setup. On mismatch, appropriate error message is thrown.
- Check for any error in the number of entries in the file, which is based on the above found values.
- Check for the id, ip address and node's values for its correctness and validation. If invalid details are found then appropriate error message is throw.
- Check for valid link details and cost details. If invalid details are found then appropriate error message is throw.
- The file parsing is taken care from line 1000 to 1267.

Once the validation is complete the data structures mentioned earlier are used to store the values of topology file.

Events

In our program there are 3 types of event which must be taken care.

- Timer event
  - On the timer event we must check for which server is timeout has covered.
  - If the time out is for the parent node on the server where the program is running. Then there must be message sent to other server which are the neighbors to the parent node.
  - This sending of message must be periodically done. Which is taken by select() statement on the line 307.
  - If the time out is from other node that the server did not receive message from its server for 3 consecutive timeouts and hence the link is assumed to be not existing the cost and other details must be updated. At line

Vijay Manoharan -----manohara@buffalo.edu

#164 we are assigning the values to the other nodes as 3 times the interval.
- The entire timer event is managed with lines #297 to #376.
- I have implement an array to hold the timer values. Every time the timer interrupt occurs, I will compute the least value for which the next interrupt will occur. Then the rest of timeout values will reduced be reduced from the time it was assigned.
- So by doing this I will have an event driven function that will trigger the event from a queue.

- Receive event
  - This event is handled whenever there is a message from the other server are received.
  - Then the message is parsed to get an appropriate information, that is used to calculate the routing information using bellman ford algorithm using the function DV_function() from lines #1292 to #1300.
  - After receiving a message from a server, the code will reset the timeout value for that server entry. And also the next timer event will be computed, timer for select will be set to that value.
  - The receive event is implement from lines #376 to #510.

- STDIN event
  - The STDIN is event is triggered on the user command from the standard input.
  - The following command are used can be issued.
    - Crash
    - Display
    - Disable <server-id>
    - Help
    - Update <server-id1> <server-id2> <cost>
    - Step
    - Packets
  - The commands are validated every time the user enters the command.
  - Once the validation of the command are done, then system will do its corresponding action.
  - The entire STDIN event is implemented from lines #511-828.

Vijay Manoharan -----manohara@buffalo.edu

- On successful execution of the command, the system return <command> Success.

Select implementation:

This is how I have implemented the select function.

```
For(;;)
{
selc_return = select(fdmax + 1, &readfds, NULL, NULL, &tv);

      if (selc_return == -1) {
         perror("select");
         continue;
               } else if (selc_return == 0) {
          ..
          ..
          ..
          //code for handling the timer events
          }
Else{
          if (FD_ISSET(sockfd, &readfds)) {
          ..
          ..
          ..
          //code for handling the Receiving message events.
          }
          If (FD_ISSET(fileno(stdin), &readfds)) {
          ..
          ..
          ..
          //code for handling the stdin events.
          }
}
```

Please note : MY code gets compiled with the warning only on the server BEATLES. Warning message is “manohara_server.c:1516:2: warning: no newline at end of file”, which makes no issues while running the code. Also when I tried to run the code for the first time, system throws an error but on running the same code again for 3-5 times the program executes without any changes made and without any issues.

END----------