Project 2
Routing Protocols

Name: Muhammed Zaki Bakshi
UBIT name: mbakshi
Person No.: 50204193

This project demonstrates distance vector routing algorithm discussed in class. Here 5 servers are used as routers who exchange routing updates with their neighbours in an effort to determine shortest path to each of the router in the network. The distance vector algorithm is based on Bellman–Ford equation which is:

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

which says the least cost from x to y is the minimum of $c(x,v) + d_v(y)$ taken over all neighbours v.

Implementation Details:

To implement this project I designed a data structure for node/router which contains all the necessary attributes to implement different aspects of the algorithm. This data structures is defined at line number 13 in `server_node.h`

```
typedef struct server_node{
	int id;
	char *ip_address;
	char *port;
	int cost;
	int next_hop_id;
	struct server_node *next;
	int alive_counter;
}server_node;
```

Here,

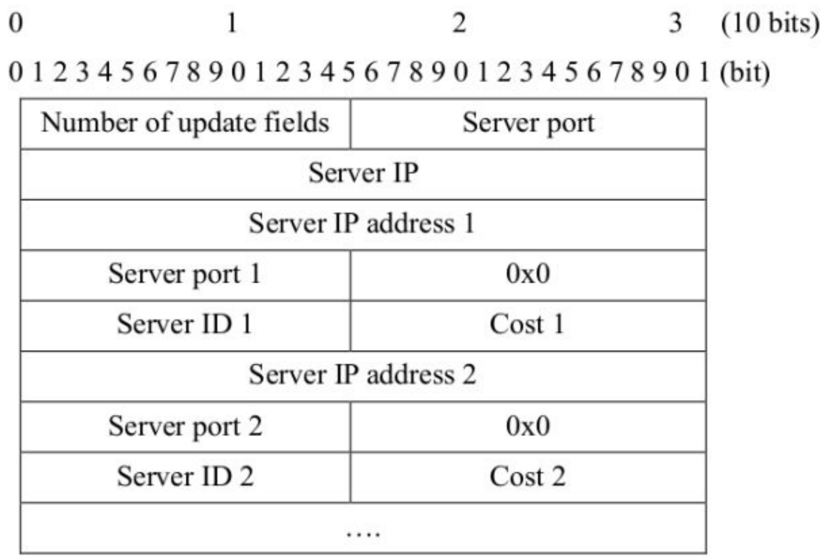
- **id** - this is id of the server node
- **ip_address** - IP address of the server node
- **port** - Listening port of server
- **cost** - C has two meanings here, when we refer to a neighbour in neighbour list it just means the cost of link to that neighbour. But when we refer to a remote host in a distance vector this means the length of shortest path to that server.
- **next_hop_id** - This field stores the id of the server where a packet should go to reach the server specified by id
- **alive_counter** - This is a counter which we increment after every timeout and set to zero when we receive data from that neighbour. Hence if this counter reaches to a value of 3, we can say that we have not received data from this neighbour in past 3 intervals.

Data Structure of routing table:

To implement a routing table I am creating a linked list of the server_node data structure which maintains the shortest path and next hop for every server in the network. I have declared the header of this linked list as `distance_vector` at line number 60 of mbakshi_proj2.c. The relevant functions pertaining to this linked list are defined in `server_node.c`

Data Structure of the update message:

According to the project description the update message format should be of the following format:



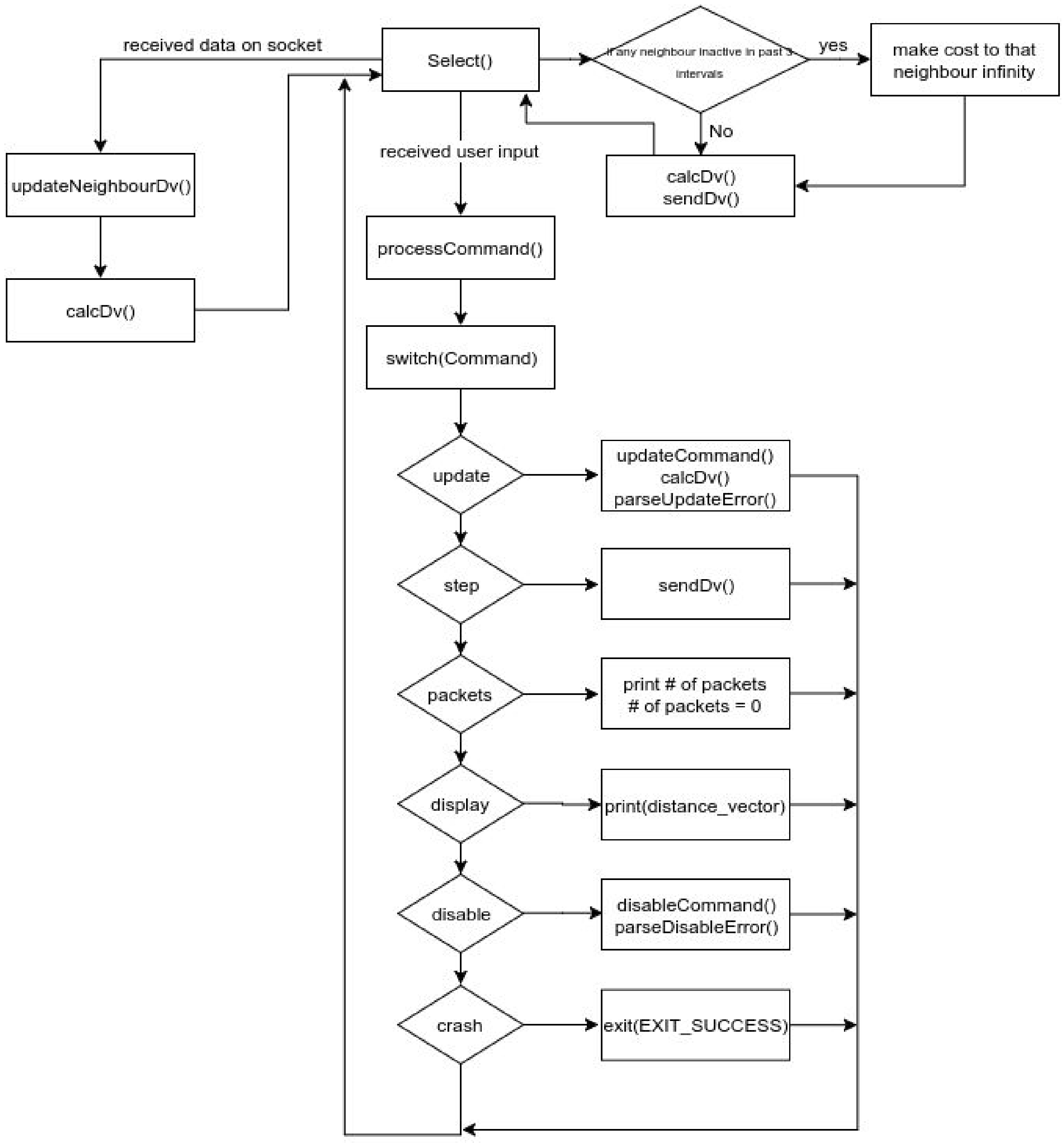
```
0                   1                   2                   3   (10 bits)
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 (bit)
```

Number of update fields	Server port
Server IP	
Server IP address 1	
Server port 1	0x0
Server ID 1	Cost 1
Server IP address 2	
Server port 2	0x0
Server ID 2	Cost 2
….	

which says each field should be of 32 bit length or 16 bit length. To implement this I am using an array of 32 bit integer, and packing it with the required data from the `distance_vector` linked list. I have defined the array at line number 446 in mbakshi_proj2.c as following:
`uint32_t send_buf[INT_BUF_SIZE];`
I am then using makeDVPacket() function to pack this array with the distance vector to be sent.

Flow diagram of the application:

Here is the high level flow diagram of the application which delineates the flow of the program and important function calls:



When the program begins at main() function it first validates command line arguments for topology file and timeout interval. Then it reads the topology file and creates the data structure for its own distance vector and neighbours' distance vector. After creating a listening socket it goes into an infinite loop which is running select() function. When select select function receives data from the listening socket then it updates its neighbour's distance vector and calculates distance vector. When it receives some user input it runs processCommand() and executes desired operations for respective commands. And when select() function gets timeout after the interval provided by user it checks if there are any neighbours who has not sent data in past 3 intervals, if yes then it changes the cost of link to that neighbour to infinity. After that the program again calculates distance vector using calcDv() function and sends distance vector to its neighbours using sendDv() function.

Code Documentation

The code for the project is divided into two directories:

1) **src:** Contains all the .c source files which defines all the functions.
2) **include:** Contains all the .h header files which declares all the functions data structures, and macros.

Source files:

1. **mbakshi_proj2.c:** This is the main source file of the project since it contains main() function which is the entry point of the application.

   1.1. **main():**
   Signature: `int main(int argc, char *argv[])`
   Description: Main Function. Entry point of application.
   Parameters:
   argc - number of command line arguments
   argv - array of command line arguments
   Returns:
   returns exit code for the process

   1.2. **inputValidation():**
   Signature: `void inputValidation(int argc, char **argv)`
   Description: Input validation for command line arguments
   Parameters:
   argc - number of command line arguments
   argv - reference for array of command line arguments
   Returns:
   void

   1.3. **get_in_addr():**
   Signature: `void *get_in_addr(struct sockaddr *addr)`
   Description: Takes sockaddr structure and returns reference to sockaddr_in or sockaddr_in6 based on IP version
   Parameters:
   addr - pointer to input sockaddr structure.
   Returns:
   returns pointer to sockaddr_in or sockaddr_in6 based on IP version

   1.4. **readTopologyFile():**
   Signature: `void readTopologyFile()`
   Description: Reads the topology file provided as command line argument
   Parameters:
   none
   Returns:
   void

1.5. sendUDP():

Signature: `void sendUDP(int socket, void *buf, size_t len, char *ip, char *port)`
Description: Sends input buffer to specified ip and port via UDP
Parameters:

socket -	file descriptor of sending socket
buf -	reference pointer for buffer to send
len-	length of buffer
ip-	ip address of the destination host
port-	port of the destination host

Returns:
void

1.6. updateNeighbourDv():

Signature: `void updateNeighbourDv(uint32_t *rec_array, int rec_int)`
Description: Updates neighbour's distance vector after receiving a message from a neighbour.
Parameters:

rec_array -	received array from listening socket
rec_int -	number of received 32 bit integers

Returns:
void

1.7. calcDV():

Signature: `void calcDV()`
Description: Calculates distance vector of host running the application
Parameters:
none
Returns:
none

1.8. makeDVPacket():

Signature: `int makeDVPacket(uint32_t send_buf[])`
Description: Creates distance vector packet ready to send
Parameters:
send_buf - reference to the buffer which will be packed by the function
Returns:
return total number of 4 byte integers to be sent.

1.9. sendDV():

Signature: `void sendDV()`
Description: Sends distance vector packet
Parameters:
none
Returns:

void

1.10. disableCommand():

Signature: `int disableCommand(char *arg1)`
Description: Executes user input disable command
Parameters:
arg1 - Command argument 1
Returns:
returns DISABLE_SUCCESS in case of success or other error code in case of failures. (refer error_codes.h)

1.11. disableNeighbour():

Signature: `int disableNeighbour(int neighbour_id)`
Description: Disables connection to specified neighbour
Parameters:
neighbour_id - ID of neighbour whose connection will be disabled
Returns:
returns 1 if success, 0 if failed

1.12. updateCommand():

Signature: `int updateCommand(char *arg1, char *arg2, char *arg3)`
Description: Executes user input update command
Parameters:
arg1 - First argument entered by user
arg2 - Second argument entered by user
arg3 - Third argument entered by user
Returns:
returns UPDATE_SUCCESS if success, other error codes if failed

1.13. sendUpdateMessage():

Signature: `void sendUpdateMessage(int neighbour_id, int cost)`
Description: Sends update message to a neighbour if its cost is changed
Parameters:
neighbour_id - ID of the neighbour whose cost is changed
cost - New cost for the neighbour
Returns:
void

2. command_handling.c:

Contains functions used to process and handle user entered commands.

2.1. convertToLower():

Signature: `void convertToLower(char *p)`
Description: Converts input string to lowercase to enable comparison
Parameters:

p - pointer to input command string

Returns:

void

2.2. processCommand():

Signature: `int processCommand(char *cmd_msg, int *command, char **arg1, char **arg2, char **arg3)`

Description: Processes user entered command cmd_msg and provides matched command and entered arguments

Parameters:

cmd_msg - User entered command message

command - reference to matched command macro (if matched to acceptable command)

arg1 - reference to first argument entered by user

arg2 - reference to second argument entered by user

arg3 - reference to third argument entered by user

Returns:

returns 1 if successful. 0 in case of any errors

3. common_methods.c:

Common methods used by the application.

3.1. stringTokenizer():

Signature: `int stringTokenizer(char *input_string, char *delimiter, char **tokens, int token_array_size)`

Description: Tokenized input_string using the delimiter provided

Parameters:

input_string -	input string to the function
delimiter -	delimiter used to split input string
tokens -	reference to output tokens array after splitting

token_array_size - size of the token array provided

Returns:

returns number of tokens

3.2. parseInt():

Signature: `int parseInt(const char *str, int *val)`

Description: A utility function to convert string to integer

Parameters:

str - reference to input string

val - reference to output int value

Returns:

returns 1 if success, 0 if string does not contain any valid integer

3.3. parseIPStringToInt():

Signature: `uint32_t parseIPStringToInt(char *ip)`

Description: Converts ip address represented in string in X.X.X.X format to 32bit integer

Parameters:

ip - reference to input string which contains ip address

Returns:

returns unsigned 32 bit integer

3.4. parseIPIntToString():

Signature: `void parseIPIntToString(int ip, char *ret_string)`

Description: Converts IP address represented in 32 bit integer to String X.X.X.X format

Parameters:

ip -	input ip address represented in 32 integer
ret-string -	reference to converted string to be returned

Returns:

void

4. error_codes.c

Contains functions to parse different error codes returned by command executing functions and display appropriate output.

4.1. parseUpdateError():

Signature: `void parseUpdateError(char *command, int error_code)`

Description: Parses the input error code for the input command and displays appropriate output

Parameters:

command -	reference to the command string
error_code -	input error code

Returns:

void

4.2. parseDisableError():

Signature: `void parseDisableError(char *command, int error_code)`

Description: Parses the input error code for the input command and displays appropriate output

Parameters:

command -	reference to the command string
error_code -	input error code

Returns:

void

5. server_node.c

Contains the implementation of all the necessary and relevant functions to maintain a linked list of server_node data structure.

5.1. addToServerNodeList():

Signature: `void addToServerNodeList(server_node **list, int id, char *ip_address, char *port, int cost)`

Description: create a new node using the values in input parameters and add to the end of the referenced list

Parameters:

list - reference to the head of the list

id - new node id

ip_address - new node ip address

port - new node port

cost - This can be a cost of the link to the node (in case of neighbour list) or minimum distance to a node (in case distance vector)

Returns:

void

5.2. removeFromServerNodeList():

Signature: `int removeFromServerNodeList(server_node **list, int id)`

Description: remove a node specified by the id from the referenced list

Parameters:

list - reference to the head of the list

id - id of the node to be removed

Returns:

returns 1 if successful, 0 in case there is no node with specified id

5.3. findById():

Signature: `server_node * findById(server_node **list, int id)`

Description: find a node in the reference list having specified id

Parameters:

list - reference to the head of the list

id - id of the node to be searched

Returns:

returns pointer to the node if found, returns NULL otherewise

5.4. findByIPPort():

Signature: `server_node * findByIPPort(server_node **list, char *ip_address, char *port)`

Description: find a node in the referenced list having specified ip address and port

Parameters:

list - reference to the head of the list

ip_address - IP address of the desired node

port - Port of the desired node

Returns:

returns pointer to the node if found, returns NULL otherewise

5.5. printServerNodeList():

Signature: `void printServerNodeList(server_node **list)`

Description: Prints id, cost and next hop id of all the nodes the referenced list

Parameters:

list - reference to the head of the list

Returns: void

Header Files:

This section lists the header files used in the project, which contains macros and data structures used in the project

1. **command_handling.h**

    1.1. MAXTOKENS - Maximum number of tokens(words) allowed in a command
    1.2. NUMBEROFCOMMANDS - Total number of commands
    1.3. CMD_UPDATE - Macro for update command
    1.4. CMD_STEP - Macro for step command
    1.5. CMD_PACKETS - Macro for packets command
    1.6. CMD_DISPLAY - Macro for display command
    1.7. CMD_DISABLE - Macro for disable command
    1.8. CMD_CRASH - Macro for crash command
    1.9. CMD_TEST1 - Macro for test1 command (used for debugging)
    1.10. CMD_TEST2 - Macro for test2 command (used for debugging)

2. **common_methods.h**
    2.1. DEBUG - Macro to enable or disable Debug output

3. **error_codes.h**
    3.1. UPDATE_SUCCESS - Error Code for Success on update command
    3.2. UPDATE_ID_NOT_INT - Error code when entered id is not integer
    3.3. UPDATE_FIRST_ID_NOT_SELF - Error code when enter id is not self id
    3.4. UPDATE_NEIGHBOUR_OUT_OF_BOUND - Error code when neighbour is out of bound
    3.5. UPDATE_COST_INCORRECT - Error code when incorrect cost is entered
    3.6. UPDATE_NEIGHBOUR_NOT_FOUND - Error code when neighbour is not found
    3.7. UPDATE_NOT_ENOUGH_PARAMETERS - Error code if not enough parameters are provided
    3.8. DISABLE_SUCCESS - Error code when disable command is successful
    3.9. DISABLE_ID_NOT_INT - Error code when id provided is not integer
    3.10. DISABLE_NEIGHBOUR_NOT_FOUND - Error code when there is no neighbour for id
    3.11. DISABLE_NOT_ENOUGH_PARAMETERS - Error code if not enough parameters are provided

4. **server_node.h**
    4.1. INF - Value for infinite cost