CNT 5106C

Project 3

**Documentation of Protocol**

**Haitham Gabr**

**I. Abstract**

Here’s a general description of usage scenario:

1. User issues a command on the client’s command line
2. Client validates command, printing a readable error message if something is wrong
   1. Makes sure the command is a known one
   2. Validates number of arguments for the given command
   3. Validates the given parameters whenever appropriate (name, IP address, port)
3. If the command is a server-related one (Find, Insert, Delete), the client sends the command to the server and printing whatever response the server responds with. If the server doesn’t respond after 6 seconds, client times out with a readable message.
4. If the command involves server-to-server communication, the requesting server uses the server-to-server protocol documented below.
5. If the command is a client-related one (Server, Quit), client processes it and prints a readable message immediately.
6. Special cases:
   1. Test command: client sends a message to the server, prints a timeout error if request times out, or a ready message if server responds.
   2. Kill command: client deletes server information before sending the command to the server.

**II. Client-to-Server protocol**

* **Test**
  + Message format:

Test

* + Client validations: none
  + Server response: ERROR: Unknown command
  + If server responds with an error, client prints a “ready” message.
  + If it times out, client prints a “timeout” message.
* **Insert**
  + Message format:

Insert <ip-address> <port-number>

* + Client validations:
    - Number of arguments must be 2.
    - <ip-address> must be a valid IP address.
    - <port-number> must be a number between 1024 and 65535.
  + Server response:
    - Success: Record added successfully.
  + Client prints the server response.
* **Delete**
  + Message format:

Delete [<ip-address>] [<port-number>]

* + Client validations:
    - Number of arguments must be between 0 and 2.
    - <ip-address> must be a valid IP address, if provided.
    - <port-number> must be a number between 1024 and 65535, if provided.
  + Server response:
    - Success: Record deleted successfully.
    - Non-existing record: ERROR: record not found.
  + Client prints the server response.
* **Find**
  + Message format:

Find <name> <ip-address>

* + Client validations:
    - Number of arguments must be 2.
    - <name> must be either:
      1. Alphanumeric with maximum length of 80.
      2. A wildcard ‘\*’.
    - <ip-address> must be:
      1. A valid IP address.
      2. Any of the four numbers can be replaced by a wildcard ‘\*’.
  + Server response:
    - A line containing number of records found, followed by a line for each record containing the details of this record.
  + Client prints the server response.
* **Kill**
  + Message format:

Kill

* + Client validations: none
  + Client deletes the current server information.
  + Server saves the records to a data file.
  + Server response: Database saved successfully. Server dying.
  + Client prints the server response.
* **Link**
  + Message format:

Link {<ip-address> <port>} | <server name>

* + Client validations:
    - Number of arguments must be 1 or 2.
    - <server name> must be alphanumeric with maximum length of 80.
    - <ip-address> must be a valid IP address, if provided.
    - <port-number> must be a number between 1024 and 65535, if provided.
  + Server response:
    - Success: <server name> linked successfully.
    - Unknown server: ERROR: Unknown server.
    - Duplicate name: ERROR: Name duplication.
    - Already linked: ERROR: server already linked.
  + Client prints the server response.
* **Unlink**
  + Message format:

Unlink {<ip-address> <port>} | <server name>

* + Client validations:
    - Number of arguments must be 1 or 2.
    - <server name> must be alphanumeric with maximum length of 80.
    - <ip-address> must be a valid IP address, if provided.
    - <port-number> must be a number between 1024 and 65535, if provided.
  + Server response:
    - Success: <server name> unlinked successfully.
    - Unknown server: ERROR: Unknown server.
    - Not linked: ERROR: server is not linked.
  + Client prints the server response.
* **Register**
  + Message format:

Register <client name> <port>

* + Client validations:
    - Number of arguments must be 2.
    - User must be not registered.
    - <client name> must be alphanumeric with maximum length of 80.
    - Port must be a number between 1024 and 65535.
  + Server response:
    - Success: client registered successfully.
    - Repeated name: ERROR: name is already used.
  + Client prints the server response.
* **Unregister**
  + Message format:

Unregister <client name>

* + Client validations:
    - Number of arguments must be 1.
    - User must be registered, and it has to have been under <client name>
    - <client name> must be alphanumeric with maximum length of 80.
  + Server response:
    - Success: client unregistered successfully.
  + Client prints the server response.
* **List**
  + Message format:

List <client list> <server list>

* + Client validations:
    - Number of arguments must be 2.
    - <client name> and <server list> must be either:
      1. alphanumeric with maximum length of 80
      2. A quoted comma-separated list of (a)
      3. A wildcard ‘\*’
  + Server response:
    - Success: results are printed along with their number.
    - Unknown server: ERROR: Unknown server.
  + Client prints the server response.
* **Send**
  + Message format:

Send <client list> <server list> <message>

* + Client validations:
    - Number of arguments must be 3.
    - <client name> and <server list> must be either:
      1. alphanumeric with maximum length of 80
      2. A quoted comma-separated list of (a)
      3. A wildcard ‘\*’
  + Server response:
    - Success: recipients are printed along with their number.
    - Unknown server: ERROR: Unknown server.
  + Client prints the server response.
* **Neighbors**
  + Message format:

Neighbors [<server list>]

* + Client validations:
    - Number of arguments must be 0 or 1.
    - <server list> must be either:
      1. alphanumeric with maximum length of 80
      2. A quoted comma-separated list of (a)
      3. A wildcard ‘\*’
  + Server response:
    - Success: results are printed along with their number.
    - Unknown server: ERROR: Unknown server.
  + Client prints the server response.
* **Forwarding**
  + Message format:

Forwarding [<server list>]

* + Client validations:
    - Number of arguments must be 0 or 1.
    - <server list> must be either:
      1. alphanumeric with maximum length of 80
      2. A quoted comma-separated list of (a)
      3. A wildcard ‘\*’
  + Server response:
    - Success: results are printed along with their number.
    - Unknown server: ERROR: Unknown server.
  + Client prints the server response.

**III. Server-to-Server protocol**

* **ServerLink**
  + Used to ask a server to establish two-way link.
  + Message format:

ServerLink <my-name> <my-port> <my-routing-info>

* + Response format:

<your-name> <your-routing-info>

* **ServerUnlink**
  + Used to ask a server to cancel the two-way link.
  + Message format:

ServerUnlink <my-name>

* + Response format:

OK

* **ForwardTo**
  + Used to ask a server to route a message to another server
  + Message format:

ForwardTo <destination> <message>

* + Response format:

OK

**IV. Other commands:**

* Server <ip-address> <port-number>
  + Client validations:
    - Number of arguments must be 2.
    - <ip-address> must be a valid IP address.
    - <port-number> must be a number between 1024 and 65535.
  + Client resets server information to the new IP address and port.
* Quit
  + Client validations: none
  + Client quits

**V. Routing Algorithm:**

The routing algorithm used is a distance vector algorithm. Each server keeps a distance vector for each of its neighbors. Routing information is updated based on Bellman-Ford formula using the neighbors distance vectors. Routing tables are changed in the following events:

1) A Link command is issued: A new server is linked and its distance vector is obtained. Routing table is recomputed. A new distance vector is produced and advertised to all neighbors.

2) An Unlink command is issued: A neighbor is lost and its distance vector is removed. Routing table is recomputed. A new distance vector is produced and advertised to all neighbors.

3) A new distance vector advertisement is received from a neighbor. Routing table is recomputed. A new distance vector is produced and advertised to all neighbors.

4) Every 15 seconds, All neighbors are pinged to makes sure they’re alive. If one is dead, its distance vector is removed. Routing table is recomputed. A new distance vector is produced and advertised to all neighbors.