# DOCUMENTATION

# SYC

# PROJECT

# “KNOCK KNOCK”

## Specification

The application consists of two processes: a server and a client.

First, the server process must be started. This process opens a number of UDP ports, given as parameters, and starts waiting on them for client packets. When a proper sequence of packets sent from a single origin (the same IP address and port) is notified, the server opens a randomly selected TCP port and sends this number to the address, from which a proper sequence of UDP packets was received. Next, the server waits for a TCP connection from this client and after a simple communication (request-response type) it disconnects and starts to listen to UDP ports again

The client process accepts as its parameters the server address and a sequence of port numbers, to which it should “knock”. After starting, it opens a UDP port, from which it sends a sequence of UDP packets to consecutive ports given as its parameters. After the last of the is sent, it waits for a UDP response from the server. Next, it connects with the server using TCP protocol and after a simple communication (request-response type) it disconnects and terminates.

If a port sequence is incorrect, there will be no response from the server. In such case the client should terminate with an error after a timeout.

1. The server is listening for UPD packages on the specified ports.

2. Server ports for UDP packets are opening in a specific sequence.

3. A sequence of UDP sessions is processing in a HashMap(`Map<Integer, Session> session`).

4. When the client starts the server address and port numbers (in the particular sequence) are passed to it via command line arguments.

5. Client sends the UDP packages of 256 bytes to the specified port, in the specific order.

6. The server reads IP address from packets and creates a Hash for each client.

7. A blocking que is used to indicate the correct sequence of event processing on the server. Various events are recorded in it.

8. There are two types of events. Event for retrieving the package and for session end.

9. The Session object is checked for sequence, `updateWhenRecievePackage()` method gets the correct sequence from the Server object. For each client a buffer(sequence field) is created which is corresponded to the current connection sequence from the client.

10. If the sequence of UDP ports was specified correctly then the server opens a TCP port with the random port number and sends UDP datagram packet with this TCP port number to the client.

11. The client receives a UDP packet, reads the TCP port number from it. Creates a TCP connection and sends a TCP message “Hello from client”.

12. After receiving the TCP message “Hello from client” from client the server sends a TCP response “Hello, it is me, server” and closes the TCP port.

13. If the sequence of UDP ports in the buffer was specified incorrectly then the TCP port number from the server is not send to the client. If there is timeout handler on the client side it waits till the end of timeout, otherwise client waits for the response indefinitely. This mechanism was done for secure purpose in case of there is an attempt to hack ports, so it takes more time to hack.

14. At the same time, the ability to connect another client is preserved.