POLISH-JAPANESE ACADEMY
OF INFORMATION TECHNOLOGY

UDP Protocol Design Implementation Port Knocking

Revision 1.0

Table of Contents

Revision History

Version	Name	Reason For Changes	Date
1.0	*Vikram Mandal*	*Initial Revision*	*10/01/2021*

Approved By

Approvals should be obtained for project manager, and all developers working on the project.

Name	Signature	Department	Date
Kazimierz Zielinski		PJAIT Information Technology	

1. Introduction

1.1 Purpose

This design will detail the implementation of the requirements as defined in the Software Requirements Specification – UDP Port Knocking (SKJ) 2020

1.2 System Overview

This project is intended to create a custom protocol which utilizes UDP protocol for sending client packets to the server and after wards the server responds back with a randomly selected port through UDP protocol. Later on, further communication is established through this randomly generated protocol using TCP protocol for some request- response.

1.3 Design Map

Summarize the information contained within this document or the family of design artifacts. Define all major design artifacts and/or major sections of this document and if appropriate, provide a brief summary of each. Discuss any significant relationships between design artifacts and other project artifacts.

2. Design Considerations

All design considerations were handled in current project considering that there are not any changes in original design specification.

2.1 Assumptions

The ports which are used are available in the host machine.

2.2 System Environments

- Client
  IP Address, Port1, Port2, Port3 …
- Server
  Port1, Port2, Port3 …

2.3 Design Methodology

- **2.3.1 Server**
  To design the server, we need to open a Datagram socket to accept UDP packets at a given port, to do this we need to create separate threads to handle parallel incoming connections at each UDP port inside **service()** method.

After we are done with all incoming packets we search for **"bye"** message (**case insensitive**)in the incoming packet to stop receiving any further packets from the client. When we receive a packet with the **"bye"** message we end the while loop and call the **process()** method to process the accumulated packets, generate a random TCP port and send the random port number as a response datagram to the original client from where the request had come. For this we take the help of **Map<ClientAddress,AddressStore>** ***packetCountsSent*** which stores the details of each incoming datagram(its origin and the packets).
After successfully sending the random port number, we initialize the TCP server with this random port number and wait for requests from the client. After successful requests-response TCP communication we close the TCP server.

- **2.3.2 Client**
  To design the client, we get the IP address and ports to knock through the program arguments. For each input IP and port combination, we create a datagram using the input data received form the standard input stream. (**System.in ()**). These datagrams are then sent to the intended UDP server.
  After successfully sending the datagram, the client receives the response from the UDP server with the random TCP port number generated. Using this random TCP port, we further communicate with the TCP server for some request response.

2.4 Risks and Volatile areas

As this application is based on Threads, so we need to be careful while creating multiple sever and client threads, we might face ***OutOfMemoryError*** in case we run out of the memory allocated to our program.

3. Development and Build

- The project is built on Java 11 technology, using its standard Socket protocols for TCP and UDP.
- For flawless build and code deployment, we have used Maven build tool and its plugins.
- We use maven JAR plugin to create jar file of the entire project which can be deployed in any virtual machine supported by JVM.

3.1 Application Build

To build the application and package it in to JAR, run below command:
mvn clean install

To deploy the code nexus repository, run below command (Make sure you have setup correct repo directories in **settings.xml**)
mvn clean deploy

3.2 Running the Client and Server

To run the Server application run below command

java -cp ./KnockKnock-1.0-SNAPSHOT.jar com.syc.portknock.Server 1025

To run the multiple client application run below command multiple times in different machines:

java -cp ./KnockKnock-1.0-SNAPSHOT.jar com.syc.portknock.Client 192.168.0.100 1025

3.3 Source Code

Entire project is available in public repository https://github.com/vkmguy/SKJ.git