Project Name: Proxy Network for Data Aggregation

1. Introduction

In this project, I developed a Proxy application that acts as a bridge between clients and keyvalue servers. The main goal was to create a network where clients can access data without knowing exactly which server holds the key or which protocol (TCP or UDP) it uses. The system supports complex connections, including cycles (loops) between proxies.

2. Implementation Details

I wrote the project using **Java 1.8**. All the logic is contained in a single file named Proxy.java. I did not modify the original client and server files provided in the task.

- **Handling Connections:** Since the proxy needs to listen to both TCP and UDP ports at the same time, I used Java Threads. One thread accepts TCP connections, while the main thread handles incoming UDP packets.
- **Data Storage:** I used a synchronized list to store information about connected nodes (servers and other proxies) to avoid issues when multiple threads access the data simultaneously.

3. How the Protocol Works (and Cycle Detection)

I used the standard text protocol (GET, SET, OK, NA) as described in the task. However, to get the full 500 points, I needed to support "arbitrary structures," meaning proxies could be connected in a loop (e.g., A connects to B, and B connects to A).

To prevent the program from crashing into an infinite loop, I added a simple extension to the protocol:

- When a proxy forwards a command, it adds a tag like visited:port1,port2 to the end of the message.
- Before processing a request, my proxy checks this tag.
- If the proxy sees its own port number in the visited list, it knows it has seen this request before. It immediately stops the process and returns "NA" (or an empty list) to break the loop.

4. Features

My project implements all the requirements for the final grade:

- **Dual Protocol:** It connects to both TCP and UDP servers seamlessly.
- **Aggregation:** The GET NAMES command searches the entire network and returns all available keys from all servers.
- **Routing:** It finds where a key is stored and routes GET VALUE or SET commands to the correct server.
- **Translation:** It allows a TCP client to communicate with a UDP server (and vice versa).
- **Loop Protection:** It works correctly even if the network has cycles.

5.How to Run

First, compile the code in the source directory:

Bash javac
*.java

Then, run the proxy by specifying its local port and the servers it should connect to:

Bash java Proxy -port <local_port> -server <address> <port>
...

Example Test Scenario: Here is a command to test the full system with a cycle:

Bash
Start Server 1 (TCP)
java TCPServer -port 9001 -key apple -value 100

Start Server 2 (UDP)
java UDPServer -port 9002 -key pear -value 200

Start Proxy B (Connects to Proxy A to make a loop)
java Proxy -port 8889 -server 127.0.0.1 9002 -server 127.0.0.1 8888

Start Proxy A (Main proxy)
java Proxy -port 8888 -server 127.0.0.1 9001 -server 127.0.0.1 8889