2117230020142

# MINI PROJECT

## COMPUTER NETWORKS

## TOPIC:

## CUSTOM DNS SERVER SIMULATION WITH UDP

# Computer Networks Mini Project

## Custom DNS Server Simulation with UDP

## Abstract

The Domain Name System (DNS) is a vital component of the internet that enables the translation of human-readable hostnames into numerical IP addresses required for network communication. This project simulates a simplified DNS server using the User Datagram Protocol (UDP) to demonstrate the fundamental principles of DNS resolution and client-server interaction. The server maintains a predefined DNS table containing hostname-to-IP mappings and listens for incoming client queries. Upon receiving a request, it performs a lookup and responds with the corresponding IP address or an error message if the hostname is not found. The client, implemented using UDP sockets, sends queries and displays responses in real-time. This project serves as an educational tool to understand DNS operations, socket programming, and lightweight communication protocols.

## Introduction

In modern networking, DNS plays an indispensable role in ensuring user-friendly communication between devices and resources. Without DNS, users would need to memorize IP addresses to access websites, which is impractical and inefficient. DNS provides a distributed, hierarchical system to resolve hostnames into IP addresses, thus acting as the backbone of internet usability.

This mini-project focuses on simulating a custom DNS server using Java's socket programming features with UDP. The server maintains a static DNS table, accepts queries from multiple clients, and responds with corresponding IP addresses. The UDP protocol is used due to its lightweight, connectionless nature, which closely aligns with real-world DNS functionality. By implementing this project, learners gain hands-on experience in client-server communication, network programming, and the internal working of DNS resolution.

.

## Implementation

The implementation of the custom DNS server simulation is based on a client-server architecture using the UDP protocol. The system is developed in Java, with UDP socket programming as the core technique for hostname-to-IP resolution. The key components of the implementation include:

### 1. DNS Server Module

- Initializes a UDP socket on a specific port (5050) to listen for client queries.
- Maintains a DNS table using a `HashMap` with hostname–IP address mappings.
- Continuously receives incoming datagrams containing hostname queries from clients.
- Looks up the requested hostname in the DNS table and prepares the response:
  - If found → returns the corresponding IP address.
  - If not found → returns the message “Host not found”.
- Sends the response back to the client via a UDP packet.

### 2. DNS Client Module

- Provides an interface for the user to input a hostname.
- Creates a UDP socket to send the query packet to the DNS server.
- Waits for the server’s response and receives the IP address or error message.
- Displays the server’s response on the client console.
- Supports multiple queries in a single session until the user types “exit”.

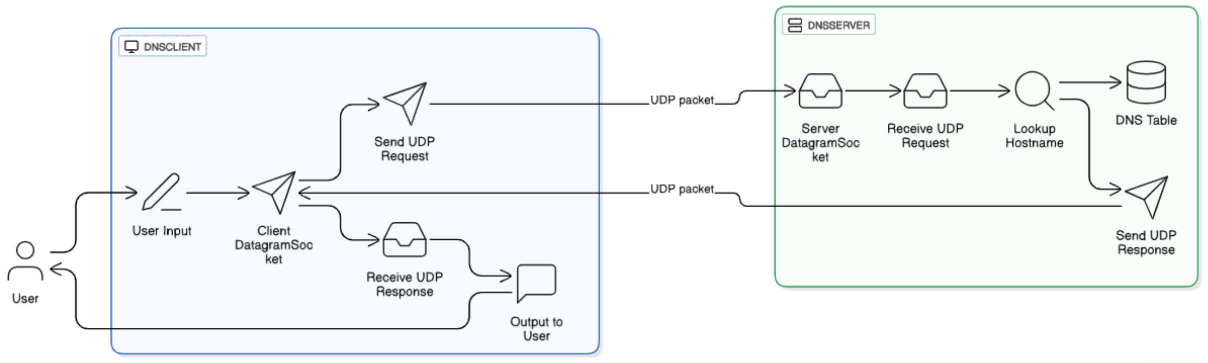
### 3. Query Processing and Communication

- Uses lightweight UDP communication, avoiding connection establishment overhead.
- Each query is encapsulated into a UDP datagram and transmitted to the server.
- Server parses the incoming datagram to extract the hostname.

- Response packets are encoded into byte streams and transmitted back to the client.
- Ensures efficient, real-time hostname resolution in a controlled environment.

The architecture minimizes complexity while clearly demonstrating DNS operations. The system supports multiple queries, ensures efficient resource usage, and provides a simple simulation of real-world DNS functionality. This makes it a valuable educational tool for learning socket programming and client-server interaction.

## Architecture Diagram:



## Research Paper:

| TITLE | AUTHOR | KEY RESULT | LIMITATIONS |
|---|---|---|---|
| Measuring DNS over TCP in the Era of Increasing DNS Response Sizes: A View from the Edge | Mike Kosek, Trinh Viet Doan, Simon Huber, Vaibhav Bajpai | Found that DNS-over-TCP (DoTCP) is generally slower than DNS-over-UDP (DoUDP), though the response time increase is less than 37% for most resolvers. Public resolvers showed comparable reliability. | Focuses on large-scale measurement data; does not delve into simulation frameworks or custom server setups. |

| | | | |
|---|---|---|---|
| Evaluating DNS Resiliency and Responsiveness with Truncation, Fragmentation & DoTCP Fallback | Pratyush Dikshit, Mike Kosek, Nils Faulhaber, Jayasree Sengupta, Vaibhav Bajpai | Analyzed over 14 million measurements and found that some resolvers fail to fallback to TCP when needed, increasing risk of fragmentation. Also, large EDNS(0) buffer sizes may cause vulnerabilities. | Primarily observational; not focused on UDP-based DNS simulation or custom table-based DNS servers. |
| Comparing the Effects of DNS, DoT, and DoH on Web Performance | Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, Nick Feamster | Compared conventional DNS, DNS-over-TLS (DoT), and DNS-over-HTTPS (DoH). Found that DoT/DoH often have higher response times, but sometimes load pages faster than traditional DNS depending on network conditions. | Emphasis on encrypted DNS protocols; not directly relevant to basic UDP DNS simulation. |
| T-DNS | (SIGCOMM 2014 authors) | Proposed an improved DNS design using TCP + TLS for privacy, larger payload support, and mitigation of spoofing and amplification attacks. | Uses TCP and TLS, not UDP; focuses more on performance/security enhancements than simulation. |
| Adaptive Transport Layer Security Model (ad-TLSM) | (AIIETA paper) | Improved cache hit rate by 25–40%, and reduced DNS traffic by 20–35%, via frequency/relevancy-based caching | Designed for encrypted DNS and cache optimization, not simple UDP-based DNS simulation. |

## Applications

- **Network Protocol Simulation:** Understand DNS operations in a simplified, controlled environment.
- **Educational Use:** Demonstrates core UDP socket programming concepts.
- **Testing Environment:** Simulate DNS resolution without relying on actual internet-based DNS.
- **Security Research:** Extendable to support spoofing detection and DNS over UDP simulations.

## CODE:-

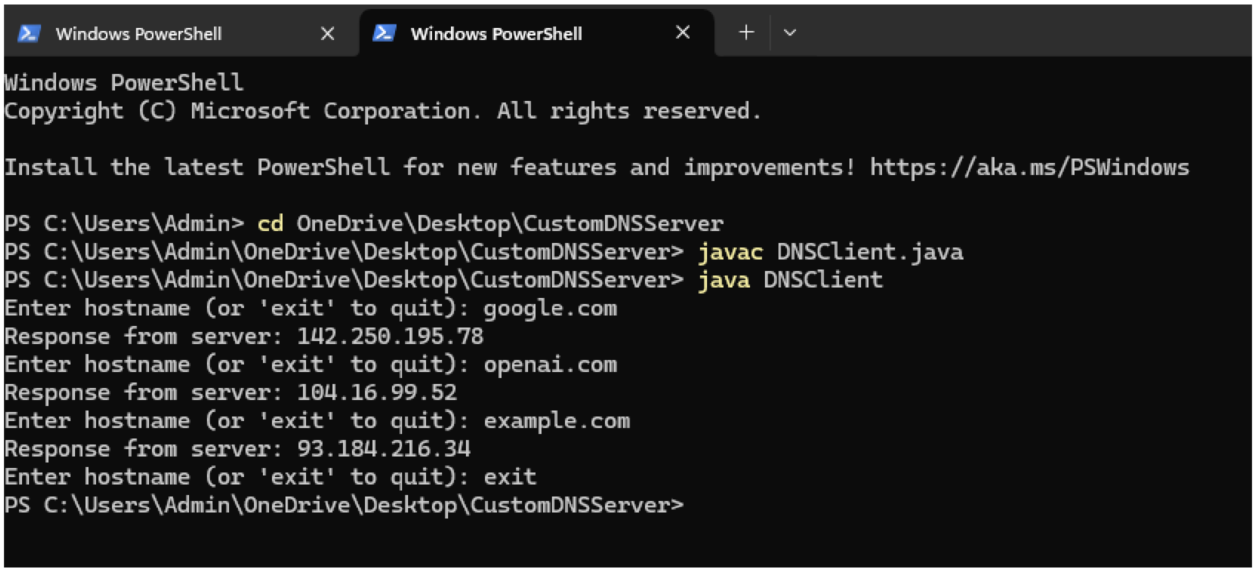
## DNSServer.java:

```
import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.HashMap;


public class DNSServer {

    public static void main(String[] args) {

        try {

            HashMap<String, String> dnsTable = new HashMap<>();

            dnsTable.put("google.com", "142.250.195.78");

            dnsTable.put("openai.com", "104.16.99.52");

            dnsTable.put("example.com", "93.184.216.34");
            DatagramSocket serverSocket = new DatagramSocket(5050);

            byte[] receiveBuffer = new byte[1024];

            byte[] sendBuffer;


            System.out.println("DNS Server started on port 5050...");


            while (true) {

                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);

                serverSocket.receive(receivePacket);

                String hostname = new String(receivePacket.getData(), 0,
receivePacket.getLength()).trim();


                System.out.println("Received query: " + hostname);


                String response = dnsTable.getOrDefault(hostname, "Host not found");


                sendBuffer = response.getBytes();

                InetAddress clientAddress = receivePacket.getAddress();

                int clientPort = receivePacket.getPort();

                DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length,
clientAddress, clientPort);

                serverSocket.send(sendPacket);

            }

        } catch (Exception e) {

            e.printStackTrace();
        }
    }
}
```

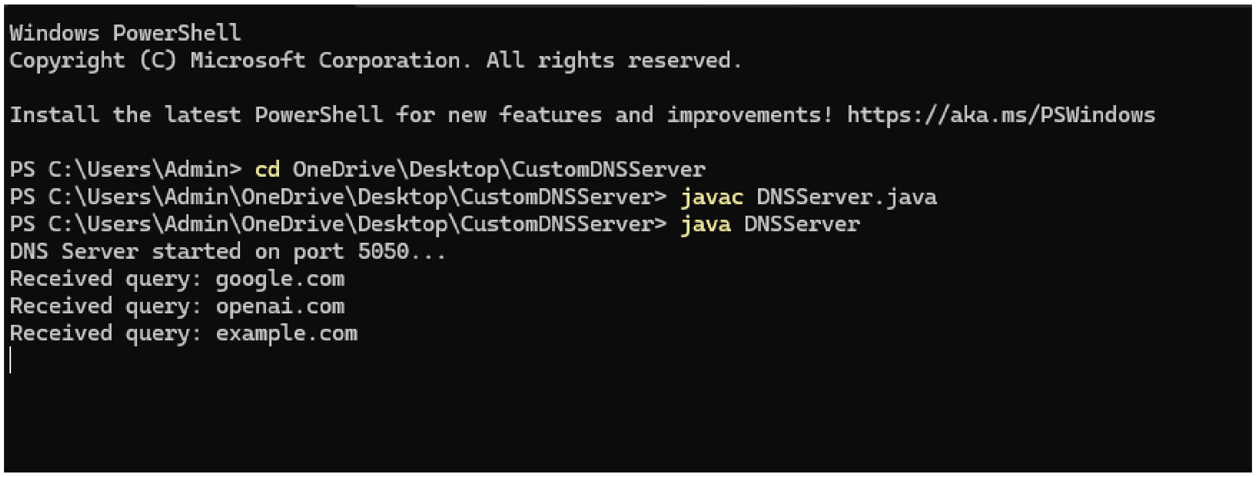
## DNSClient.java:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class DNSClient {
    public static void main(String[] args) {
        try {
            DatagramSocket clientSocket = new DatagramSocket();
            InetAddress serverAddress = InetAddress.getByName("localhost");
            Scanner sc = new Scanner(System.in);

            byte[] sendBuffer;
            byte[] receiveBuffer = new byte[1024];

            while (true) {
                System.out.print("Enter hostname (or 'exit' to quit): ");
                String hostname = sc.nextLine();
                if (hostname.equalsIgnoreCase("exit")) break;
                sendBuffer = hostname.getBytes();

                DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length,
serverAddress, 5050);

                clientSocket.send(sendPacket);



                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);

                clientSocket.receive(receivePacket);

                String response = new String(receivePacket.getData(), 0, receivePacket.getLength());



                System.out.println("Response from server: " + response);

            }



            clientSocket.close();

            sc.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}
```

## OUTPUT:-



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Admin> cd OneDrive\Desktop\CustomDNSServer
PS C:\Users\Admin\OneDrive\Desktop\CustomDNSServer> javac DNSClient.java
PS C:\Users\Admin\OneDrive\Desktop\CustomDNSServer> java DNSClient
Enter hostname (or 'exit' to quit): google.com
Response from server: 142.250.195.78
Enter hostname (or 'exit' to quit): openai.com
Response from server: 104.16.99.52
Enter hostname (or 'exit' to quit): example.com
Response from server: 93.184.216.34
Enter hostname (or 'exit' to quit): exit
PS C:\Users\Admin\OneDrive\Desktop\CustomDNSServer>
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Admin> cd OneDrive\Desktop\CustomDNSServer
PS C:\Users\Admin\OneDrive\Desktop\CustomDNSServer> javac DNSServer.java
PS C:\Users\Admin\OneDrive\Desktop\CustomDNSServer> java DNSServer
DNS Server started on port 5050...
Received query: google.com
Received query: openai.com
Received query: example.com
```

## Conclusion

This project successfully simulates a custom DNS server using UDP and demonstrates the essential working of DNS resolution. By implementing hostname-to-IP mapping, the project illustrates client-server communication and the role of UDP in connectionless data exchange. The approach provides a simplified yet effective model for understanding DNS operations, making it an excellent educational resource. Furthermore, the simulation can be extended with advanced features such as dynamic DNS entries, caching, or security mechanisms, thereby opening avenues for further research and development.