Chapter 18: Network Programming

18.0 Introduction

Network programming enables communication between software applications across different devices over a network. Whether you're developing a chat application, implementing web services, or building client-server systems, understanding network programming is essential in today's connected world.

This chapter explores the fundamental concepts, protocols, models, and programming techniques used to build networked applications, with practical implementation using Java (and references to Python/C for parallel understanding). It also covers low-level socket programming as well as higher-level abstractions, providing a comprehensive foundation for real-world application development.

18.1 Basics of Computer Networks

Before diving into code, it's crucial to understand how data flows between computers.

18.1.1 What is a Network?

A network is a collection of interconnected devices (computers, servers, routers) that communicate to share data and resources.

18.1.2 Types of Networks:

- **LAN (Local Area Network)**
- **WAN (Wide Area Network)**
- **MAN (Metropolitan Area Network)**
- **PAN (Personal Area Network)**

18.1.3 Network Models

- **OSI Model (7 layers)**
    - Physical, Data Link, Network, Transport, Session, Presentation, Application
- **TCP/IP Model (4 layers)**
    - Network Access, Internet, Transport, Application

18.1.4 Protocols

- **TCP (Transmission Control Protocol)** – reliable, connection-oriented
- **UDP (User Datagram Protocol)** – faster, connectionless
- **HTTP/HTTPS**, **FTP**, **SMTP**, **DNS**, etc.

18.2 Socket Programming

Socket programming provides an API for communication between machines.

18.2.1 What is a Socket?

A socket is one endpoint of a two-way communication link between two programs running on the network.

- **Socket = IP address + Port number**
- Types: Stream (TCP), Datagram (UDP)

18.2.2 Java Socket Classes

- `java.net.Socket` – client side (TCP)
- `java.net.ServerSocket` – server side (TCP)
- `java.net.DatagramSocket`, `java.net.DatagramPacket` – for UDP

18.3 TCP Programming in Java

18.3.1 TCP Server Example

```
import java.net.*;
import java.io.*;

public class TCPServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Server started...");
        Socket clientSocket = serverSocket.accept();
        System.out.println("Client connected.");

        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);

        String message = in.readLine();
        System.out.println("Received: " + message);
        out.println("Echo: " + message);

        in.close();
        out.close();
        clientSocket.close();
        serverSocket.close();
```

```
    }
}
```

18.3.2 TCP Client Example

```
import java.net.*;
import java.io.*;

public class TCPClient {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 5000);
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

        out.println("Hello Server!");
        System.out.println("Server response: " + in.readLine());

        in.close();
        out.close();
        socket.close();
    }
}
```

18.4 UDP Programming in Java

18.4.1 UDP Server Example

```
import java.net.*;

public class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];

        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
        socket.receive(receivePacket);
        String message = new String(receivePacket.getData()).trim();
        System.out.println("Received: " + message);

        socket.close();
    }
}
```

18.4.2 UDP Client Example

```
import java.net.*;
```

```
public class UDPClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket();
        byte[] sendData = "Hello UDP Server".getBytes();

        InetAddress IPAddress = InetAddress.getByName("localhost");
        DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, IPAddress, 9876);
        socket.send(sendPacket);

        socket.close();
    }
}
```

18.5 Important Concepts in Network Programming

18.5.1 Ports

- Range: 0–65535
- Well-known ports: 0–1023 (e.g., HTTP – 80, FTP – 21)

18.5.2 IP Addressing

- IPv4: e.g., 192.168.1.1
- IPv6: longer format to support more devices

18.5.3 DNS and Domain Names

Domain Name System maps domain names to IP addresses.

18.6 Multi-threaded Server Programming

To handle multiple clients simultaneously:

```
class ClientHandler extends Thread {
    private Socket clientSocket;

    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
    }

    public void run() {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
```

```
            String inputLine;

            while ((inputLine = in.readLine()) != null) {
                out.println("Echo from server: " + inputLine);
            }

            in.close();
            out.close();
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

And in the main server:

```
ServerSocket serverSocket = new ServerSocket(5000);
while (true) {
    Socket socket = serverSocket.accept();
    new ClientHandler(socket).start();
}
```

18.7 Protocol Design and Custom Application Layer

You can define your own application-level protocols using text or binary formats. Example:

Client → Server: `LOGIN username password` **Server → Client:** `200 OK` or `401 Unauthorized`

18.8 Higher-Level Networking with HTTP APIs

For RESTful API calls (in Java or Python):

Java Example using `HttpURLConnection`:

```
URL url = new URL("https://api.example.com/data");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
```

Python Example using `requests`:

```
import requests
response = requests.get("https://api.example.com/data")
print(response.json())
```

18.9 Common Network Programming Errors

- Port conflicts
- Firewall restrictions
- Packet loss (UDP)
- Blocking I/O (use threads or `NIO`)
- Encoding/decoding mismatch

18.10 Tools for Testing and Debugging

- **Wireshark** – packet analysis
- **Postman** – API testing
- **cURL** – command-line HTTP requests
- **Telnet** – manual TCP testing

18.11 Modern Trends in Network Programming

- **WebSocket programming** for real-time communication
- **gRPC** (Google Remote Procedure Calls)
- **Cloud-based messaging** (MQTT, AMQP)
- **ZeroMQ, Kafka** for distributed messaging

18.12 Summary

Network programming is a fundamental skill that allows applications to interact across machines. This chapter introduced key networking concepts, socket programming using TCP and UDP, multi-threading for concurrent connections, and high-level HTTP programming. Understanding these principles lays the groundwork for advanced distributed systems, cloud-native apps, and scalable web services.
