**Unit 1**

**Network Programming**

Network programming is the act of using computer code to write programs or processes that can communicate with other programs or processes across a network. Programmers use various programming languages, code libraries, and protocols to do the work. Network programming can be used to develop a wide range of applications, including web servers, email clients, file transfer utilities, remote administration tools, and more. It requires a deep understanding of network protocols, such as TCP/IP, HTTP, SMTP, and FTP, as well as programming languages like Python, Java, and C++. In network programming, developers typically write code that allows their applications to send and receive data over the network, handle network errors and interruptions, and implement security measures to protect against unauthorized access or data loss. Successful network programming requires careful planning and attention to detail to ensure that applications can communicate reliably and efficiently over the network.

**Features and Scope of Network Programming**

1. Client-Server applications: Network programming is commonly used to develop client-server applications, where one program (the server) provides a service that another program (the client) can access over a network. Examples include web servers, email servers, and database servers.
2. Distributed systems: Network programming enables developers to create distributed systems, where multiple computers or devices work together to achieve a common goal. Distributed systems can be used for tasks such as data processing, scientific computing, and online gaming.
3. IoT applications: Network programming is essential for developing Internet of Things (IoT) applications, which involve a network of interconnected devices communicating with each other. Examples include smart home systems, industrial automation, and wearable technology.
4. Real-time communication: Network programming can be used to create applications that provide real-time communication, such as instant messaging and video conferencing software. These applications require low latency and high bandwidth to ensure a smooth user experience.
5. Network security: Network programming is also critical for developing secure network applications. Developers need to implement encryption, authentication, and access control to protect against attacks and ensure the confidentiality, integrity, and availability of data.
6. Cloud computing: Network programming plays a vital role in cloud computing, which involves delivering computing resources over a network. Developers can use network programming to create cloud-based applications and services that can be accessed from anywhere in the world.

**Java Networking Primitives**

Java networking primitives refer to the low-level building blocks that are used to create networked applications in Java. These primitives include classes and interfaces that provide functionality for working with network protocols, sockets, addresses, and data transfer. Java networking primitives enables the programmers to develop network applications. These primitives are found within the java.net package.

**Connection Oriented Networking**

Connection-oriented networking refers to a type of network communication where a dedicated connection is established between two endpoints before any data transmission occurs. This type of networking provides a reliable and ordered data delivery mechanism, where packets are sent and received in a predetermined sequence, and lost packets are automatically retransmitted. In the context of Java networking primitives, connection-oriented networking is supported by the java.net.Socket and java.net.ServerSocket classes. These classes use the Transmission Control Protocol (TCP) to establish a reliable, bi-directional communication channel between a client and a server. The java.net.Socket class represents a client-side socket, and it is used to establish a connection to a server socket created by the java.net.ServerSocket class. Once a connection is established, data can be transmitted between the client and server using the input and output streams of the socket. The java.net.ServerSocket class represents a server-side socket, and it is used to listen for incoming connections from clients. When a client connects, the server socket accepts the connection and creates a new socket to handle the communication with the client.

**Connectionless Networking**

connectionless networking refers to the communication between two endpoints without establishing a dedicated connection between them. This type of networking is also known as datagram-oriented or packet-oriented networking, as data is transmitted in discrete units called datagrams or packets. Java provides several classes for implementing connectionless networking, including the DatagramSocket and DatagramPacket classes. DatagramSocket is used to create a socket for sending and receiving datagrams, while DatagramPacket is used to represent a packet of data. To send data over a connectionless network, you first create a DatagramSocket object and then create a DatagramPacket object containing the data you wish to send. You then use the send() method of the DatagramSocket object to send the packet to the destination address and port. To receive data, you create a DatagramSocket object and then use the receive() method of the DatagramSocket object to wait for incoming packets. When a packet is received, a new DatagramPacket object is created to store the data.

**Multicast Connectionless Networking**

Multicast is a type of connectionless networking where data is transmitted from one sender to multiple receivers at the same time. In multicast communication, a single packet is sent to a multicast group address, and any device that has joined that group can receive the packet. In Java networking, the Java MulticastSocket class provides support for multicast communication. The MulticastSocket class extends the DatagramSocket class, which is used for connectionless communication, and adds support for multicasting. To use multicast communication in Java, you first create a MulticastSocket object and join a multicast group using the joinGroup() method. You can then send data to the group using the send() method, which takes a DatagramPacket object as a parameter. To receive data, you create a DatagramPacket object and use the receive() method of the MulticastSocket object. When a packet is received, the data is placed in the DatagramPacket object.

**High Level Java Networking Abstraction**

High-level Java networking abstractions are a set of APIs that simplify the task of writing networked applications. These abstractions allow developers to work with network protocols and data at a higher level of abstraction than would be possible using low-level socket programming. For example, Java provides a Socket API that abstracts away much of the low-level details of network programming and provides a simple, stream-oriented interface for sending and receiving data over TCP or UDP. Other abstractions include URL handling, which provides built-in support for working with URLs and connecting to remote hosts, RMI, which allows Java objects running in different JVMs to invoke methods on each other, and HTTP handling, which provides support for making HTTP requests and retrieving responses. These abstractions make it easier for developers to write networked applications by providing simpler, more intuitive interfaces for working with network protocols and data.

**URL Based Programming**

URL-based programming is a common approach to working with remote resources over the internet, and is particularly relevant in the context of Java networking. Java provides built-in support for working with URLs through the URL class, which provides methods for parsing and manipulating URLs. Java developers can use the URL class to connect to remote servers, download resources, and perform other operations on remote resources using standard HTTP(S) methods. This makes it easy to build networked Java applications that interact with web-based services and APIs, and to perform tasks such as web scraping, data analysis, and more. Overall, URL-based programming is an important part of Java networking and a key tool for building robust and scalable networked applications.

**Remote Method Invocation(RMI)**

Remote Method Invocation (RMI) is an important aspect of Java network programming that enables distributed computing. RMI allows a Java object running in one JVM to invoke methods on a remote object running in another JVM, as if the remote object were a local object. RMI provides a high-level, object-oriented approach to remote method invocation that is easy to use and understand. When an object in one JVM makes a remote method call to an object in another JVM, RMI handles the low-level details of network communication, such as marshalling and unmarshalling parameters, sending the method invocation request to the remote JVM, and receiving and returning results. RMI is widely used in a variety of networked Java applications, including client-server architectures, distributed computing, and web services, and is an important tool for building robust and scalable networked applications.

**Network Programming Language, Tools and Platforms**

1. Perl

The 30-year old programming language continues to be popular among techies. With the new common gateway interface (CGI) scripting, Perl's popularity has increased over time. The programming language's ability to execute program on servers and dynamic websites is the reason behind its popularity. There is a lot more to Perl than just how language tends to network and system administration.

2. Bash

The native shell of Unix-based systems including Linux, macOS is the command-line interface called Bash. The network admins need to developer bash skills to easily executive commands using highly intricate syntax. For networking equipment, a number of products run on some form of Linux-based OS, allowing a great deal fo flexibility to manage devices in a secure manner. Roles best suited for bash programmers include Linux and macOS based system administration, automation and application development.

3. Tcl

Tool Command Language (Tcl) is among the most mature programming languages. It was born out of frustration due to developers' efforts to embed their own languages into the applications. Speed and power are the reasons why Tcl skills have gained popularity.

4. Go

Google created programming language, Go has its unique strengths. The programming language is used at the foundation of Docker, Kubernetes, Cloudflare, Netflix, and Uber. A number of mission-critical services are powered by Go.

5. Python

The open-source programming language is the top choice of open source community. Network admins use Python for automating system administration tasks. It can be used through the use of plugins and scripts to integrate into various workflows.

6. Java:

Java is a popular programming language for developing network applications. It has several libraries like Netty, JGroups, and Akka that allow developers to build distributed and scalable network applications.

7. Wireshark:

Wireshark is a popular network protocol analyzer that allows developers to capture and analyze network traffic. It provides a powerful toolset for debugging and troubleshooting network applications.

**Client and Server Application**

A client-server application in Java networking is a program that facilitates communication between a client and a server over a network. The two programs communicate through sockets, which are endpoints for sending and receiving data. A server program creates a ServerSocket object that listens on a specified port, waiting for incoming client connections. Once a client connects, the server creates a new socket to communicate with the client and spawns a new thread to handle the client's requests. The client program creates a socket object to connect to the server's IP address and port number, and uses that socket to send requests and receive responses from the server.

Both the client and server programs use various input/output streams and readers/writers to exchange data over the network. For example, a client can use a PrintWriter to send data to the server, and the server can use a BufferedReader to read that data. Similarly, the server can use a PrintWriter to send data to the client, and the client can use a BufferedReader to read that data. Java networking provides a robust framework for building such applications, with support for protocols like TCP and UDP, error handling, and security features. Developers can also use libraries such as Apache MINA or Netty to simplify the development of network applications in Java. Overall, a client-server application in Java networking is a powerful tool for building distributed systems and networked applications.

**Server Code**

import java.io.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(5000); // *create a server socket object listening on port 5000*

System.out.println("Server started...");

Socket clientSocket = serverSocket.accept(); // *wait for a client to connect and accept the connection*

System.out.println("Client connected...");

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true); // *create a PrintWriter object to send data to the client*

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())); // *create a BufferedReader object to read data from the client*

String inputLine;

while ((inputLine = in.readLine()) != null) { // *read lines of data from the client*

System.out.println("Received message: " + inputLine);

out.println("Server received message: " + inputLine); // *send a response back to the client*

}

out.close(); // *close the PrintWriter*

in.close(); // *close the BufferedReader*

clientSocket.close(); // *close the client socket*

serverSocket.close(); // *close the server socket*

}

}

**Client Code**

import java.io.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) throws IOException {

String serverHostname = "localhost"; // server hostname

int serverPort = 5000; // server port

Socket socket = new Socket(serverHostname, serverPort); // *create a socket object to connect to the server*

System.out.println("Connected to server...");

PrintWriter out = new PrintWriter(socket.getOutputStream(), true); // *create a PrintWriter object to send data to the server*

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream())); // *create a BufferedReader object to read data from the server*

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in)); // *create a BufferedReader object to read user input from the console*

String userInput;

while ((userInput = stdIn.readLine()) != null) { // *read lines of user input*

out.println(userInput); // *send the user input to the server*

String serverResponse = in.readLine(); // *read the response from the server*

System.out.println("Server response: " + serverResponse);

}

out.close(); // *close the PrintWriter*

in.close(); // *close the BufferedReader*

stdIn.close(); // *close the standard input*

socket.close(); // *close the socket*

}

}