UNIT-10: UDP

- In network programming with Java, UDP (User Datagram Protocol) is one of the **transport protocols** used for sending and receiving **datagrams over a network**.
- UDP is a **connectionless protocol**, which means that it **doesn't establish a persistent connection** between the sender and receiver. Instead, it sends individual packets called datagrams from one host to another.
- UDP (User Datagram Protocol) is an important transport protocol in computer networking due to the **following reasons**:

1. **Simplicity and Efficiency**: UDP is lightweight and doesn't have the overhead of establishing and maintaining connections, making it faster and more efficient for certain applications.
2. **Low Latency**: UDP offers low latency by avoiding acknowledgments and retransmissions, making it suitable for real-time applications where immediate data delivery is crucial.
3. **Broadcast and Multicast**: UDP supports broadcasting and multicasting, allowing a single packet to be sent to multiple recipients simultaneously, which is useful for streaming media and online gaming.
4. **Stateless Communication**: UDP is stateless, enabling scalability for server applications that handle a large number of clients.
5. **VoIP and Streaming**: UDP is commonly used in Voice over IP (VoIP) and multimedia streaming applications, where real-time transmission takes priority over reliability.
6. **DNS**: UDP is utilized by DNS for name resolution queries, which are typically small and time-sensitive.
7. **Internet of Things (IoT)**: UDP is frequently used in IoT devices and applications, where quick and efficient transmission of small data packets is essential.

STRUCTURE OF A UDP DATAGRAM

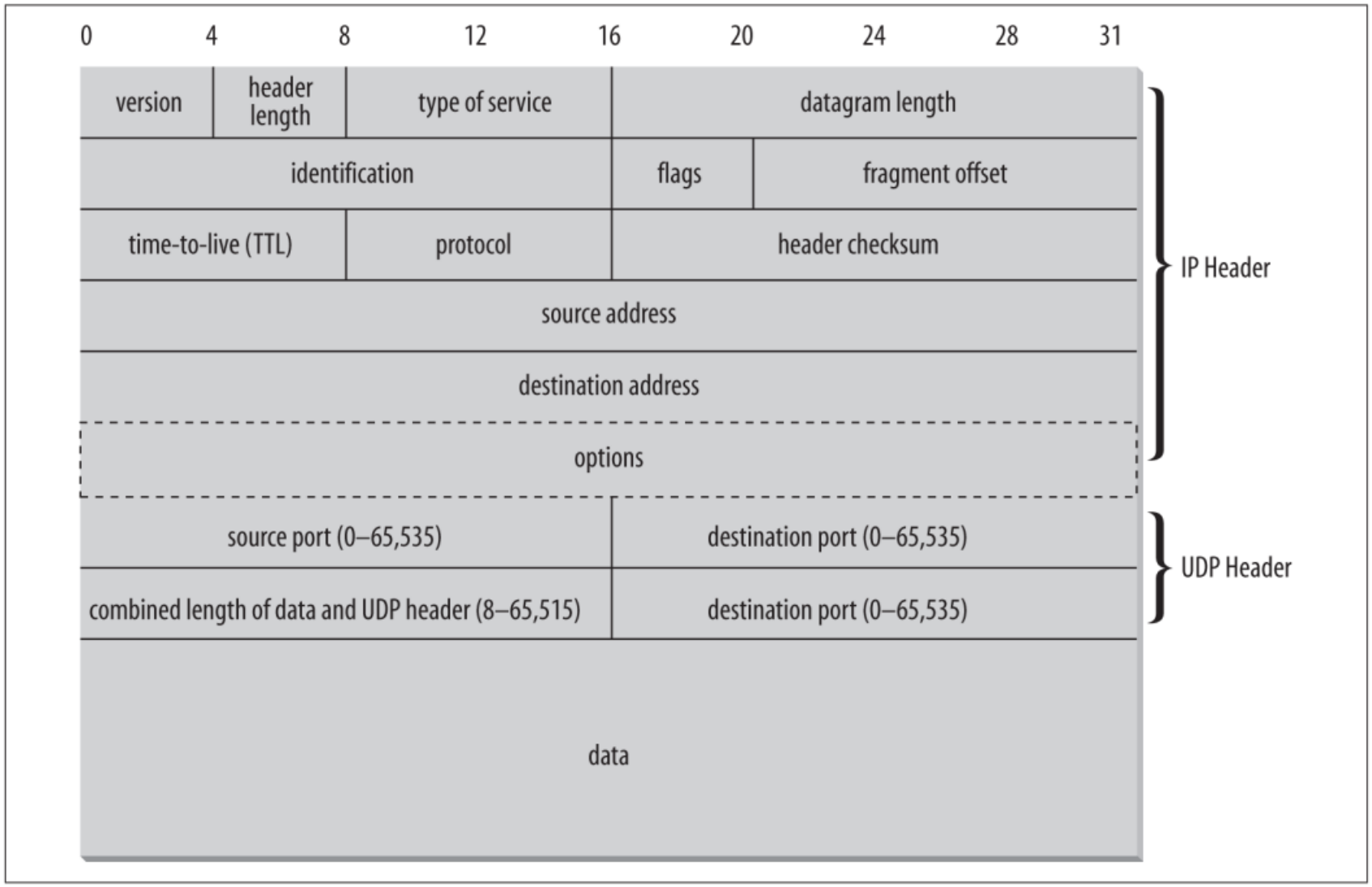

Figure 12-1. The structure of a UDP datagram

UNIT-10: UDP

- To use UDP in network programming with Java, you can make use of the **java.net package**, which provides classes for networking operations. The following **steps** outline the process of creating a UDP client and server using Java:

Client UDP:

1. Create a **DatagramSocket** object to send and receive UDP packets.
2. Create a **DatagramPacket** object to hold the data to be sent.
3. Convert your data into **bytes** and store it in the **DatagramPacket** object.
4. Specify the **destination address and port number** of the server by creating an instance of **InetAddress** and specifying the server's IP address and port number.
5. Use the **send**() method of the **DatagramSocket object to send the DatagramPacket to the server**.
6. **Close the DatagramSocket** object when done.

UNIT-10: UDP

UDP Server:

1. Create a **DatagramSocket** object and bind it to a **specific port number** on the server machine.
2. Create a **byte array to hold the received data**.
3. Create a **DatagramPacket** object to receive the incoming **UDP packet and specify the size of the buffer**.
4. Use **the receive() method** of the DatagramSocket object to receive the DatagramPacket.
5. **Extract the data** from the **DatagramPacket** object and process it as needed.
6. Optionally, send a response back to the client by creating a new DatagramPacket and using the **send() method** of the DatagramSocket.
7. Close the **DatagramSocket object** when done.

UDP CLIENT PROGRAM

```java
import java.net.*;
public class UdpClient1 {
    Run | Debug
    public static void main(String[] args) throws Exception {
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress serverAddress = InetAddress.getByName(host:"localhost");
        int serverPort = 12345;
        String message = "Hello, server!";
        byte[] sendData = message.getBytes();
        DatagramPacket sendPacket = new DatagramPacket(
                sendData, sendData.length, serverAddress, serverPort);
        clientSocket.send(sendPacket);
        clientSocket.close();
    }
}
```

UDP SERVER PROGRAM

```java
import java.net.*;
public class UdpServer2 {
    Run | Debug
    public static void main(String[] args) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(port:12345);
        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(
                receiveData, receiveData.length);
        serverSocket.receive(receivePacket);
        String message = new String(receivePacket.getData());
        System.out.println("Received from client: " + message);
        serverSocket.close();
    }
}
```

UDP ECHO CLIENT PROGRAM

```java
import java.net.*;
public class UdpClient {
    Run | Debug
    public static void main(String[] args) throws Exception {
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress serverAddress = InetAddress.getByName(host:"localhost");
        int serverPort = 12345;
        String message = "Hello, server!";
        byte[] sendData = message.getBytes();
        DatagramPacket sendPacket = new DatagramPacket(sendData,
                sendData.length, serverAddress, serverPort);
        clientSocket.send(sendPacket);
        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String receivedMessage = new String(receivePacket.getData());
        System.out.println("Received from server: " + receivedMessage);
        clientSocket.close();
    }
}
```

UDP ECHO SERVER PROGRAM

```java
public class UdpServer {
    Run | Debug
    public static void main(String[] args) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(port:12345);
        byte[] receiveData = new byte[1024];
        byte[] sendData;
        while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
                    receiveData.length);
            serverSocket.receive(receivePacket);
            String message = new String(receivePacket.getData());
            InetAddress clientAddress = receivePacket.getAddress();
            int clientPort = receivePacket.getPort();
            String responseMessage = "Hello, client!";
            sendData = responseMessage.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData,
                    sendData.length, clientAddress, clientPort);
            serverSocket.send(sendPacket);
            receiveData = new byte[1024]; // Clear the buffer for the next iteration
        }
    }
}
```

THE DATAGRAMPACKET CLASS

- The **DatagramPacket class** in Java is part of the **java.net** package and represents **a UDP packet or datagram**. It encapsulates the data being sent or received, along with the information about the source and destination addresses and ports. The DatagramPacket class provides methods to **access and manipulate the data and header information** of the packet.

➢ The **DatagramPacket** class has two constructors:

➢ **DatagramPacket(byte[] data, int length):** Constructs a DatagramPacket with the specified byte **array data** and **its length**. This constructor is commonly used when **receiving data**.

➢ **DatagramPacket(byte[] data, int length, InetAddress address, int port):** Constructs a DatagramPacket with the specified **byte array data**, **its length**, **destination address**, and **destination port**. This constructor is commonly used when sending data.

➢ The DatagramPacket class provides several methods for accessing and modifying the packet's properties:

- **getData**(): Returns the byte array that holds the data of the packet.
- **getLength**(): Returns the length of the data in the packet.
- **setData**(byte[] data): Sets the data of the packet using the provided byte array.
- **setLength**(int length): Sets the length of the data in the packet.
- **getAddress**(): Returns the source or destination IP address of the packet.
- **getPort**(): Returns the source or destination port number of the packet.
- **setAddress**(InetAddress address): Sets the source or destination IP address of the packet.
- **setPort**(int port): Sets the source or destination port number of the packet.

THE DATAGRAMPACKET CLASS

```java
import java.net.*;
public class UDPReceiver {
    Run | Debug
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(port:12345);
        byte[] buffer = new byte[1024];
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
        /*
         * DatagramPacket packet = new DatagramPacket(sendData,
         * sendData.length, receiverAddress, receiverPort);
         */
        socket.receive(packet);
        String message = new String(packet.getData(), offset:0, packet.getLength());
        System.out.println("Received message: " + message);
        socket.close();
    }
}
```

THE DATAGRAMPACKET CLASS

- The **DatagramSocket** class in Java is a **fundamental class for network communication using UDP** (User Datagram Protocol). It provides the functionality to **send and receive datagrams** (UDP packets) **over the network**.

1.Creating a DatagramSocket:

DatagramSocket(): Constructs a new DatagramSocket object **that binds to any available local port**.

DatagramSocket(int port): Constructs a new DatagramSocket object and binds it to the **specified local port**.

2.Sending Data:

send(DatagramPacket packet): **Sends a datagram packet to the destination specified** in the packet parameter.

3.Receiving Data:

receive(DatagramPacket packet): **Receives a datagram packet** and stores it in the provided packet object.

setSoTimeout(int timeout): Sets the **maximum time to wait for a packet to arrive when receiving data**. If no packet is received within the specified timeout, a SocketTimeoutException is thrown.

4.Closing the DatagramSocket:

close(): Closes the DatagramSocket and releases any resources associated with it.

THE DATAGRAMPACKET CLASS

5.Binding and Unbinding:

bind(SocketAddress localAddr): Binds the DatagramSocket to a **specific local address and port**.

bind(SocketAddress localAddr, boolean reuseAddr): Binds the DatagramSocket to a **specific local address and port**, with an option to **enable/disable the reuse of the address**.

6.Getting Socket Options:

getLocalAddress(): Returns the **local address** to which the DatagramSocket is bound.

getLocalPort(): Returns the local port to which the DatagramSocket is bound.

UDP PORT SCANNER

```java
import java.net.*;
public class UdpScanner {
    Run | Debug
    public static void main(String[] args) {
        try {
            for (int port = 1024; port <= 65535; port++) {
                try {
                    DatagramSocket socket = new DatagramSocket(port);
                    socket.close();
                    System.out.println("Port " + port +
                    " is available for UDP communication.");
                } catch (SocketException e) {
                    // Port is already in use
                    System.out.println("Port " + port + " is in use.");
                }
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

DATAGRAMCHANNEL

DatagramChannel is a class in Java that represents a channel for sending and receiving UDP datagrams. It is part of the Java NIO (New I/O) package, introduced in Java 1.4, which provides a non-blocking I/O API for efficient I/O operations.

The DatagramChannel class offers several advantages over the traditional DatagramSocket class when working with UDP:

Non-blocking I/O: DatagramChannel supports non-blocking I/O operations, allowing you to perform I/O operations without blocking the thread. This can be useful in scenarios where you need to handle multiple connections simultaneously or perform other tasks while waiting for I/O operations to complete.

Asynchronous I/O: DatagramChannel provides methods for asynchronous I/O operations using Java NIO's CompletionHandler interface. This allows you to initiate I/O operations and specify a callback handler that will be notified when the operation completes.

Channel-based architecture: DatagramChannel follows the channel-based I/O architecture introduced in Java NIO. This architecture provides a unified API for various types of channels (e.g., TCP channels, UDP channels, file channels) and promotes code reusability and modularity.

Improved performance: The Java NIO package, including DatagramChannel, is designed to provide better performance compared to the traditional I/O operations in Java. It achieves this through the use of operating system features like kernel-level event notification and memory-mapped I/O.

To use DatagramChannel, you can perform operations such as opening a channel, binding it to a specific local address and port, sending and receiving datagrams, and closing the channel. It offers methods like open(), bind(), send(), receive(), and close() for these operations.

DATAGRAMCHANNEL ECHO SERVER

```
public class DatagrmChnlServer {
    Run | Debug
    public static void main(String[] args) throws Exception {
        DatagramChannel channel = DatagramChannel.open();
        channel.bind(new InetSocketAddress(hostname:"localhost", port:12345));
        ByteBuffer buffer = ByteBuffer.allocate(capacity:1024);
        while (true) {
            buffer.clear();
            SocketAddress clientAddress = channel.receive(buffer);
            buffer.flip();
            String receivedMessage = new String(buffer.array(), offset:0, buffer.limit());
            System.out.println("Received from client: " + receivedMessage);
            String responseMessage = "Hello, client!";
            buffer.clear();
            buffer.put(responseMessage.getBytes());
            buffer.flip();
            channel.send(buffer, clientAddress);
        }
    }
}
```

DATAGRAMCHANNEL ECHO CLIENT

```java
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
public class DatagrmChnlClient {
    Run | Debug
    public static void main(String[] args) throws Exception {
        DatagramChannel channel = DatagramChannel.open();
        String message = "Hello, server!";
        ByteBuffer buffer = ByteBuffer.wrap(message.getBytes());
        channel.send(buffer, new InetSocketAddress(hostname:"localhost", port:12345));
        buffer.clear();
        channel.receive(buffer);
        buffer.flip();
        String receivedMessage = new String(buffer.array(), offset:0, buffer.limit());
        System.out.println("Received from server: " + receivedMessage);
        channel.close();
    }
}
```

DATAGRAMCHANNEL ECHO CLIENT

- In Java, you can set socket options for UDP sockets using the SocketOption class and the setOption() method provided by the DatagramSocket class. Here's an overview of the six socket options commonly used for UDP:

SO_TIMEOUT:
- This **option sets the maximum time the receive() method** of a DatagramSocket will block waiting for incoming data.
- The value is specified **in milliseconds**.
- Example usage: `socket.setSoTimeout(5000);`

SO_RCVBUF:
- This option sets the **receive buffer size** for the underlying socket.
- The value is specified in bytes.
- Example usage: `socket.setReceiveBufferSize(8192);`

SO_SNDBUF:
- This option **sets the send buffer size** for the underlying socket.
- The value is specified in bytes.
- Example usage: `socket.setSendBufferSize(8192);`

SO_REUSEADDR:
- This option allows **multiple DatagramSocket instances to bind to the same local address and port**, even if there are active connections on the socket.
- It is useful when you need to quickly reuse a socket that was recently closed.
- Example usage: `socket.setReuseAddress(true);`

SO_BROADCAST:
- This option **enables or disables the ability to send broadcast messages**.
- By default, UDP sockets are not allowed to send broadcast messages.
- Example usage: `socket.setBroadcast(true);`

IP_TOS:
- This option **sets the Type of Service (ToS) field in the IP header**.
- It allows you to specify the quality of service or priority for the outgoing packets.
- Example usage: `socket.setTrafficClass(0x10);`

EXAMPLE OF UDP SOCKET EXAMPLE

```java
import java.net.DatagramSocket;
import java.net.SocketException;
public class UdpSocOption {
    Run | Debug
    public static void main(String[] args) {
        try {
            // Create a UDP socket
            DatagramSocket socket = new DatagramSocket();
            // Set socket options
            socket.setSoTimeout(timeout:5000); // Set timeout to 5 seconds
            socket.setReceiveBufferSize(size:8192); // Set receive buffer size to 8192 bytes
            socket.setSendBufferSize(size:8192); // Set send buffer size to 8192 bytes
            socket.setReuseAddress(on:true); // Enable address reuse
            socket.setBroadcast(on:true); // Enable broadcast capability
            socket.setTrafficClass(tc:0x10); // Set IP TOS to high priority
            // Print the socket options
            System.out.println(x:"Socket options:");
            System.out.println("SO_TIMEOUT: " + socket.getSoTimeout());
            System.out.println("SO_RCVBUF: " + socket.getReceiveBufferSize());
            System.out.println("SO_SNDBUF: " + socket.getSendBufferSize());
            System.out.println("SO_REUSEADDR: " + socket.getReuseAddress());
            System.out.println("SO_BROADCAST: " + socket.getBroadcast());
            System.out.println("IP_TOS: " + socket.getTrafficClass());
            // Close the socket
            socket.close();
        } catch (SocketException e) {
            System.out.println(e.getMessage());
        }
    }
}
```