Unit-11 (IP Multicast)

As we know **Unicast** provide point-to-point communication. Unicast sockets create a connection with two well-defined endpoints. Internet multicasting is built on top of **UDP**. **Multicasting** in Java uses the **DatagramPacket** class,

one sender and one receiver IP Multicast in network programming refers to a communication method where a **single sender** can transmit data to **multiple receivers** simultaneously. It is particularly useful for applications such as **video streaming, online gaming, and distributed systems**. In Java, you can utilize the **java.net** package to implement IP Multicast functionality.

Multicast:

Multicasting is broader than **unicast**, point-to-point communication but narrower and more targeted than broadcast communication. Multicasting sends data from one host to many different hosts, but not to **everyone**; the data only goes to clients that have expressed an interest by joining a particular multicast group.

Multicast Addresses and Groups:

- A multicast address is the **shared address of a group of hosts** called a **multicast group**.
- IP addresses in the CIDR group 224.0.0.0/4 (i.e., they range from 224.0.0.0 to 239.255.255.255). CIDR, which stands for **Classless Inter-Domain Routing**, is a method for allocating IP addresses and routing Internet Protocol packets.
- All addresses in this range have the binary digits 1110 as their first four bits.
- IPv6 multicast addresses are in the CIDR group ff00::/8 (i.e., they all start with the byte 0xFF, or 11111111 in binary).

The multicast addresses in this range are divided into various categories:

Well-Known Multicast Addresses:

- 224.0.0.0: Reserved for all **systems** on the same network segment.
- 224.0.0.1: Reserved for all **hosts** on the same network segment.
- 224.0.0.2: Reserved for all **routers** on the same network segment.
- 224.0.0.5: Reserved for all **OSPF**(Open Shortest Path First)

IP networks and is based on the Shortest Path First (SPF) algorithm routers.

- 224.0.0.6: Reserved for all **OSPF**(Open Shortest path First) designated routers.(OSPF router It is widely used in larger networks due to its scalability and efficiency)
- 224.0.0.9: Reserved for all **RIPv2** routers.*(is used for routing of IPv4 addresses in small networks)*

Administratively Scoped Multicast Addresses:

- 239.0.0.0 to 239.255.255.255: Intended for local or organization-specific use within private networks.

Source-Specific Multicast (SSM) Addresses:

- 232.0.0.0 to 232.255.255.255: Reserved for SSM, where multicast traffic is only received from specific source addresses.

Client and Server:

In IP Multicast, both the client and server participate in the multicast communication. The **server** acts as the **sender**, and the **client** acts as the **receiver**, receiving multicast data sent by the server.

To establish a client-server IP Multicast communication, both the client and server need to perform specific tasks.

Server (Sender):

- Create a multicast socket and bind it to a specific port.
- Join a multicast group by specifying the multicast group address.
- Prepare the data to be sent.
- Send the data to the multicast group using the multicast socket.

Client (Receiver):

- Create a multicast socket and bind it to the same port as the server.
- Join the same multicast group as the server by specifying the multicast group address.
- Set up a loop to continuously receive multicast packets.
- Process the received data as required.

Routers and Routing:

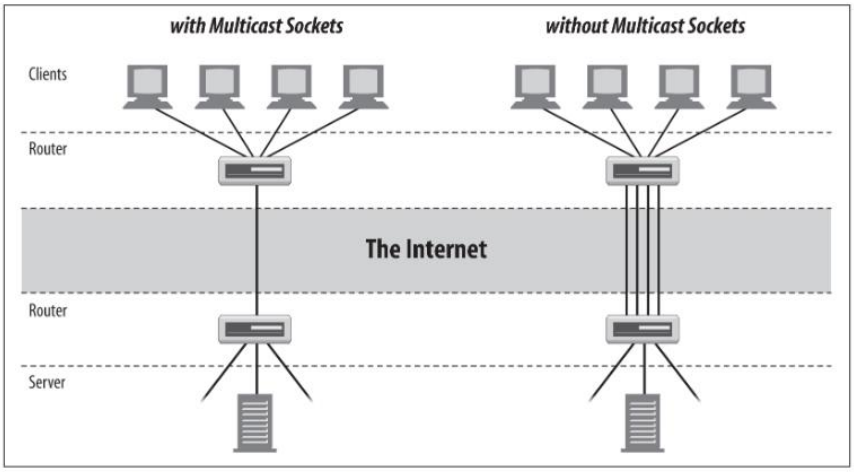

Figure 13-3. With and without multicast sockets

Routers play a critical role in IP Multicast by forwarding multicast packets across the network to ensure that they reach all the intended receivers. To facilitate this process, multicast routing protocols are used to establish and maintain the multicast distribution tree, which determines the path that multicast packets take from the sender to the receivers.

Working with Multicast Sockets:

In Java, you multicast data using the **java.net.MulticastSocket** class, a subclass of **java.net.DatagramSocket**:

```
public class MulticastSocket extends DatagramSocket implements Closeable, AutoCloseable
```

To receive data that is being multicast from a remote site, first create a **MulticastSocket** with the **MulticastSocket()** constructor. As with other kinds of sockets, you need to know the port to listen on.

```
MulticastSocket ms = new MulticastSocket(2300);
```

Next, join a multicast group using the MulticastSocket's **joinGroup()** method:

```
InetAddress group = InetAddress.getByName("224.2.2.2");
```

```
ms.joinGroup(group);
```

For data and enter a loop in which you receive the data by calling the **receive()** method inherited from the DatagramSocket class:

```
byte[] buffer = new byte[8192];

DatagramPacket dp = new DatagramPacket(buffer, buffer.length);

ms.receive(dp);
```

When you no longer want to receive data, leave the multicast group by invoking the socket's **leaveGroup**() method. You can then close the socket with the **close**() method inherited from **DatagramSocket**:

```
ms.leaveGroup(group);

ms.close();
```

Sending data to a multicast address is similar to sending UDP data to a unicast address.You do not need to join a multicast group to send data to it.

```
InetAddress ia = InetAddress.getByName("experiment.mcast.net");

byte[] data = "Here's some multicast data\r\n".getBytes("UTF-8");

int port = 4000;

DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);

MulticastSocket ms = new MulticastSocket();

ms.send(dp);
```

The Constructor:

The constructors are simple. You can either pick a port to listen on or let Java assign an anonymous port for you:

```
public MulticastSocket() throws SocketException

public MulticastSocket(int port) throws SocketException
```

public MulticastSocket(SocketAddress bindAddress) throws IOException

For example:

MulticastSocket ms1 = new MulticastSocket();

MulticastSocket ms2 = new MulticastSocket(4000);

SocketAddress address = new InetSocketAddress("192.168.254.32", 4000);

MulticastSocket ms3 = new MulticastSocket(address);

Communicating with a Multicast Group:

once a **MulticastSocket** has been created, it can perform four key operations:

1. Join a multicast group.

2. Send data to the members of the group.

3. Receive data from the group.

4. Leave the multicast group.

To join a group, pass an **InetAddress** or a **SocketAddress** for the multicast group to the **joinGroup()** method:

public void joinGroup(InetAddress address) throws IOException

public void joinGroup(SocketAddress address, NetworkInterface interface) throws IOException

Example:

```
try {
MulticastSocket ms = new MulticastSocket(4000);
InetAddress ia = InetAddress.getByName("224.2.2.2");
ms.joinGroup(ia);
byte[] buffer = new byte[8192];
while (true) {
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
ms.receive(dp);
String s = new String(dp.getData(), "8859_1");
```

```
System.out.println(s);
}
} catch (IOException ex) {
System.err.println(ex);
}
```

Leaving groups and closing the connection

Call the **leaveGroup()** method when you no longer want to receive **datagrams** from the specified **multicast group,**

public void leaveGroup(InetAddress address) throws IOException
public void leaveGroup(SocketAddress multicastAddress,NetworkInterface interface) throws IOException

Example:

```
MulticastSocket ms = new MulticastSocket(40);

InetAddress group = InetAddress.getByName("224.2.2.2");

ms.leaveGroup(group);

System.out.println("Leaving");
```

Multicast client and Server Example:

ServerMulticast.java

```
 // Specify the multicast group address and port
        InetAddress group = InetAddress.getByName("224.0.0.1");
        int port = 8888;
        // Create a multicast socket
        MulticastSocket socket = new MulticastSocket();
        // Join the multicast group
        socket.joinGroup(group);
        // Prepare the data to be sent
        String message = "Hello, multicast!";
        byte[] buffer = message.getBytes();
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group,
port);
        // Send the data to the multicast group
        socket.send(packet);
        // Leave the multicast group and close the socket
        socket.leaveGroup(group);
        socket.close();
```

ClientMulticast.java

```
// Specify the multicast group address and port
        InetAddress group = InetAddress.getByName("224.0.0.1");
        int port = 8888;
        // Create a multicast socket and bind it to the port
        MulticastSocket socket = new MulticastSocket(port);
```

```
        // Join the multicast group
        socket.joinGroup(group);
        // Create a buffer to receive incoming data
        byte[] buffer = new byte[1024];
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
        // Continuously receive multicast packets
        while (true) {
            socket.receive(packet);
            // Convert the packet data to a string
            String message = new String(packet.getData(), 0, packet.getLength());
            System.out.println("Received: " + message);
            // Clear the buffer for the next receive operation
            packet.setLength(buffer.length);
        }
        // Leave the multicast group and close the socket (not reached in this
example)
        socket.leaveGroup(group);
        socket.close();
```