

UNIT 11

IP Multicast

(2 Hours) — Network Programming (CAC355)

Please note

These notes may contain errors. They were generated automatically, so please verify important facts, API details and code against your textbook, the official Java documentation and your teacher's material before relying on them (especially for exams).

11.1 Multicasting

Normally network communication is **unicast**: one sender, one receiver. **Broadcast** sends to every host on a network, wasting bandwidth for uninterested hosts. **Multicast** sends a packet **once** to a **group** of interested receivers; routers replicate it only where needed. Typical uses: video conferencing, live streaming, stock-ticker feeds, online games, service discovery and software updates.

Type	Sender → Receivers	Example
Unicast	One → one	Web page request
Broadcast	One → all hosts on the LAN	ARP, DHCP discover
Multicast	One → members of a group	IPTV, mDNS, OSPF

Multicast: one send, many receivers (joinGroup)

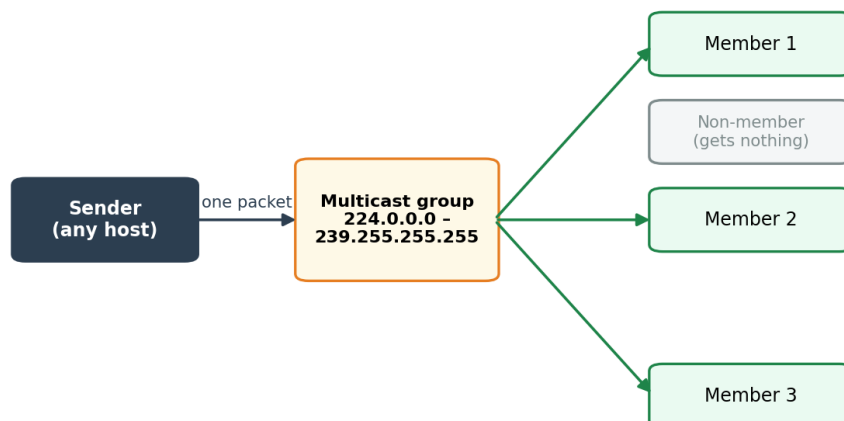


Figure 11.1: One packet is delivered to every member of the multicast group.

11.1.1 Multicast Addresses and Groups

A **multicast group** is identified by an IP address in the **Class D** range **224.0.0.0 – 239.255.255.255** (IPv4, binary prefix 1110) or by addresses beginning with **FF** in IPv6 (ff00::/8). A group is not tied to any particular host: hosts can join or leave at any time and senders need not be members.

Range	Meaning
224.0.0.0 – 224.0.0.255	Link-local, reserved (e.g. 224.0.0.1 all hosts, 224.0.0.2 all routers); never forwarded by routers.
224.0.1.0 – 238.255.255.255	Globally scoped, assigned to specific protocols or dynamically.
239.0.0.0 – 239.255.255.255	Administratively scoped (private, like RFC 1918) – good for experiments.

Each multicast packet carries a **TTL (time to live)** that limits how far it travels: 0 = same host, 1 = same subnet (default for Java), 32 = same site, 64 = same region, 128 = same continent, 255 = unrestricted.

Example 11.1 – Valid multicast address?

Which of these are valid IPv4 multicast addresses? (a) 192.168.1.5 (b) 224.0.0.9 (c) 239.1.2.3 (d) 240.0.0.1

Solution

(b) and (c). The valid range is 224.0.0.0 to 239.255.255.255. (a) is a private unicast address and (d) is in the reserved Class E range.

11.1.2 Clients and Servers

In multicasting the usual client/server roles are blurred. A **sender** (server) simply sends datagrams to the group address and port; any number of **receivers** (clients) **join the group** and read from the same port. There is no connection and delivery is **unreliable**, because multicast is built on UDP. The sender does not know who or how many are listening, which also means that the same program can act as both sender and receiver (as in a chat application).

11.1.3 Routers and Routing

Hosts tell their local router which groups they want using **IGMP** (Internet Group Management Protocol; MLD for IPv6). Multicast routers build **distribution trees** using protocols such as **PIM** (Protocol Independent Multicast) or DVMRP, so a packet is copied only on links that lead to group members. Without multicast-capable routers, multicast stays within a single LAN (TTL = 1), and many ISPs do not forward it across the public Internet.

Note: Multicast needs support in the OS, network interface, switches (IGMP snooping) and routers. If a program works on one machine but not across the network, check firewalls, TTL and the network interface used.

11.2 Working with Multicast Sockets

Java provides **java.net.MulticastSocket**, a subclass of **DatagramSocket** with extra methods for joining groups and setting the TTL. Packets are still sent and received with **DatagramPacket**.

11.2.1 The Constructors

Constructor	Description
<code>MulticastSocket()</code>	Binds to any free port (enough for a sender only).
<code>MulticastSocket(int port)</code>	Binds to the given port – receivers must use the group's port.
<code>MulticastSocket(SocketAddress addr)</code>	Binds to that address; pass null for an unbound socket (then <code>setReuseAddress(true)</code> and <code>bind()</code>).

Multicast sockets enable **SO_REUSEADDR** by default so several programs on one host can listen to the same group and port.

11.2.2 Communicating with a Group

The four steps for a receiver are: (1) create a **MulticastSocket** on the group's port, (2) **joinGroup()**, (3) **receive()** datagrams, (4) **leaveGroup()** and close. A sender needs only to build a packet addressed to the group and call **send()** (joining is not required).

Method	Description
<code>joinGroup(InetAddress mcastaddr)</code>	Start receiving datagrams sent to the group (deprecated form; ok for simple use).
<code>joinGroup(SocketAddress, NetworkInterface)</code>	Preferred form (Java 7+): join on a specific interface.
<code>leaveGroup(...)</code>	Stop receiving for that group.
<code>setTimeToLive(int) / getTimeToLive()</code>	Hop limit of outgoing packets (0–255).

Method	Description
setInterface(InetAddress) / setNetworkInterface(NetworkInterface)	Choose the outgoing interface on multihomed hosts.
setLoopbackMode(boolean)	Disable (true) or enable (false) delivery of own packets to the local host (a confusingly named option).

Receiver example

```
import java.net.*;

public class MulticastReceiver {
    public static void main(String[] args) throws Exception {
        InetAddress group = InetAddress.getByName("239.1.2.3");
        int port = 4446;
        try (MulticastSocket socket = new MulticastSocket(port)) {
            NetworkInterface nif = NetworkInterface.getByNamed("eth0"); // adjust name
            InetSocketAddress groupAddr = new InetSocketAddress(group, port);
            socket.joinGroup(groupAddr, nif);

            byte[] buf = new byte[1024];
            while (true) {
                DatagramPacket p = new DatagramPacket(buf, buf.length);
                socket.receive(p);
                String msg = new String(p.getData(), 0, p.getLength());
                System.out.println("Received: " + msg);
                if (msg.equals("exit")) break;
            }
            socket.leaveGroup(groupAddr, nif);
        }
    }
}
```

Listing 11.1: Multicast receiver (joins 239.1.2.3:4446)

Sender example

```
import java.net.*;

public class MulticastSender {
    public static void main(String[] args) throws Exception {
        InetAddress group = InetAddress.getByName("239.1.2.3");
        try (MulticastSocket socket = new MulticastSocket()) {
            socket.setTimeToLive(4); // stay within a few hops
            for (String msg : new String[] {"Hello group", "Second message", "exit"}) {
                byte[] data = msg.getBytes();
                socket.send(new DatagramPacket(data, data.length, group, 4446));
                Thread.sleep(500);
            }
        }
    }
}
```

Listing 11.2: Multicast sender

Note: The interface name ("eth0") differs between systems (e.g. "wlan0", "en0"). If NetworkInterface.getByNamed returns null, list interfaces with NetworkInterface.getNetworkInterfaces() (Unit 2) or use the older joinGroup(InetAddress) method. Start the receiver(s) first, then the sender.

Example 11.2 – Why can several receivers use the same port?

Two receivers on the same machine both bind port 4446. Why does this not throw “Address already in use”?

Solution

MulticastSocket sets `SO_REUSEADDR`, which allows multiple sockets to bind the same port. Each socket that has joined the group receives its own copy of every datagram.

Key Takeaways

- ✓ Multicast sends one packet to many receivers; groups use IPv4 addresses 224.0.0.0–239.255.255.255 (IPv6: ff00::/8).
- ✓ Hosts join groups via IGMP; multicast routers (PIM) forward packets only toward group members.
- ✓ TTL controls how far packets travel (Java default 1 = local subnet).
- ✓ Java uses MulticastSocket + DatagramPacket; receivers need `joinGroup()` and the group's port; senders can just `send()`.
- ✓ Multicast is UDP-based: no connection, no reliability, no ordering.
- ✓ Always `leaveGroup()` and `close()` the socket when finished.

— *End of Unit 11* —