

We now integrate with Microsoft Teams, helping you to connect your internal knowledge base with your chat. [Learn more.](#)

MulticastSocket constructors and binding to port or SocketAddress

Asked 6 years ago Active 5 years, 11 months ago Viewed 6k times



I may have a fundamental misunderstanding of the term binding here but I am confused about the usage of the [MulticastSocket](#) and it's constructors. They no not do what I understand they should do *should* do so any who can help me clear my misunderstanding would be appreciated.

7

Firstly what I am trying to achieve. I have tried to write a short program that creates a MulticastSocket bind it (i.e. listen) on a **specific network adapter** and then join a specific Multicast group. I have tried the following (client) code which works ok, I can multicast a packet to it without the Multicast socket timing out.

★

2

```
public class Main {
    public static final int DEFAULT_MULTICAST_PORT = 5555;
    public static final String multicastGroup = "225.4.5.6";
    public static final String adapterName = "eth0";
    public static final int MAX_PACKET_SIZE = 65507;

    CharBuffer charBuffer = null;
    Charset charset = Charset.defaultCharset();
    CharsetParameter decoder = charset.newDecoder();
    static ByteBuffer message = ByteBuffer.allocateDirect(MAX_PACKET_SIZE);
    static boolean loop = true;

    static byte[] buffer = new byte[MAX_PACKET_SIZE];

    public static void main(String[] args) {
        try {
            //MulticastSocket mSocket = new MulticastSocket(new
            InetAddress("192.168.2.23", DEFAULT_MULTICAST_PORT));
            MulticastSocket mSocket = new MulticastSocket(DEFAULT_MULTICAST_PORT);
            mSocket.setReuseAddress(true);
            mSocket.setSoTimeout(5000);
            NetworkInterface nic = NetworkInterface.getByNma(adapterName);
            mSocket.joinGroup(new InetAddress(multicastGroup,
            DEFAULT_MULTICAST_PORT), NetworkInterface.getByNma(adapterName));
            DatagramPacket p = new DatagramPacket(buffer, MAX_PACKET_SIZE);
            while (loop){
                try{
                    mSocket.receive(p);
                    System.out.println("Packet Received.");
                } catch (SocketTimeoutException ex){
                    System.out.println("Socket Timed out");
                }
            }
        } catch (IOException ex){
            System.err.println(ex);
        }
    }
}
```

Unfortunately as soon as I alter the MulticastSocket constructor to `MulticastSocket(SocketAddress bindaddr)` it stops working. It seems I can only use the bind-to-port constructor for it to work so what does it exactly bind to when this constructor is called as I have not specified a network adapter at this stage. (I know I join the group with a specific NetworkInterface later on, but how can I be sure that during the constructor call it doesn't bind to ANY adapter?)

I could also join a group without specifying the adapter then I don't know which adapter it's bound to.

Can anyone explain what binding to a port *only* actually does and is it possible to listen only on a specific NetworkInterface?

Updated #1 **

Having read the replies so far and discussed this with a work colleague, the following is my understanding of Java MulticastSocket:

1. MulticastSocket() creates a multicast socket bound a port selected at random (by the host's underlying OS bound to the wildcard address 0.0.0.0 i.e. ALL network cards. However calling this constructor with *null* creates an unbound MulticastSocket. In this scenario calling the `bind(SocketAddress)` method binds to a IP and port.
2. MulticastSocket(int port) creates a multicast socket bound to a specific port but on EVERY IP address.
3. MulticastSocket(SocketAddress sa) creates a multicast socket bound to the specified IP address (which could be ANY IP address, *even* an invalid Multicast address) and port.

Using option 2, this means *potentially* any packet sent to the specified port, regardless of its actual destination will be passed to the MulticastSocket. I say potentially because Multicast packets will only arrive if the group has been joined (but other packets destined to non-Multicast addresses will arrive *provided the port number matches?*)

Using option 3, I can bind to an IP address and *only* packets whose destination matches will arrive at the socket. It would be perfectly feasible with this option to bind to the IP of a particular network interface however no Multicast packets would be received because they would not be destined for the specific IP address of the network card (which is why I never saw them arrive on the code example). It is possible to also bind to a valid Multicast address but in this scenario *only* specific packets whose destination matches the bound multicast address would arrive at the socket, *regardless of calls to joinGroup()*.

Now the calls to `joinGroup()` do *nothing* to the socket itself but issue an IGMP request to the underlying network system to ensure that routers, the OS itself etc actually start *routing* the specified multicast packets up the hardware and through the network stack and finally to the Java MulticastSocket itself.

**** Update 2 **** Quoting from "UNIX Network Programming", Stevens, Fenner, Rudoff:

To receive a multicast datagram, a process must join the multicast group and it must also *bind* a UDP socket to the port number that will be used as destination port number for datagrams sent to the group. The two operations are distinct and both are required. Joining the group tells the host's IP layer and datalink layer to receive multicast datagrams sent to that group. Binding the port is how the application specifies to UDP that it wants to receive datagrams sent to that port. Some applications also bind the multicast address to the socket, in addition to the port. This prevents any other datagrams that might be received for that port to other unicast, broadcast or multicast addresses from being delivered to the socket.

I think that explains it all.

**** Update 3 **** Just wanted to post the code I tested and the comments explains what happens with each.

```
/**
 * This first creates an UNBOUND Multicast Socket and then binds to
 * a port (but accepting the wildcard IP 0.0.0.0.
 * The Following WORKS:
 */
/*MulticastSocket mSocket = new MulticastSocket(null);
mSocket.bind(new InetAddress(DEFAULT_MULTICAST_PORT));
mSocket.setReuseAddress(true);
mSocket.setSoTimeout(5000);
NetworkInterface nic = NetworkInterface.getByNma(adapterName);
mSocket.joinGroup(InetAddress.getByNma(multicastGroup));
*/

/**
 * The following creates a a network socket and binds in the constructor
 * to a local address and port. Consequently, it DOES NOT work because
 */
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
*/
/*MulticastSocket mSocket = new MulticastSocket(new
InetSocketAddress("192.168.2.23", DEFAULT_MULTICAST_PORT));
mSocket.setReuseAddress(true);
mSocket.setSoTimeout(5000);
NetworkInterface nic = NetworkInterface.getByNam(adapterName);
mSocket.joinGroup(InetAddress.getByNam(multicastGroup));*/

/**
 * The following binds to the same multicast group this is 'joined' later
 * and this works correctly. However if the join() is NOT called, no packets
 * arrive at the socket, as expected.
 */

/*MulticastSocket mSocket = new MulticastSocket(new
InetSocketAddress(multicastGroup, DEFAULT_MULTICAST_PORT));


mSocket.setSoTimeout(5000);
NetworkInterface nic = NetworkInterface.getByNam(adapterName);
// Comment out the following line and it no longer works correctly.
mSocket.joinGroup(InetAddress.getByNam(multicastGroup));*/

/**
 * The following binds to a specified port on 0.0.0.0 and joins
 * a specific Multicast group on a specific adapter. This must mean that the
 * IGMP must occur
 * on the specified adapter.
 *
 * ** This will ALSO receive packets addressed DIRECTLY to the ip 192.168.2.23
with the same
 * port as DEFAULT_MULTICAST_POR ***ONLY!***
 */
MulticastSocket mSocket = new MulticastSocket(DEFAULT_MULTICAST_PORT);
mSocket.setReuseAddress(true);
mSocket.setSoTimeout(5000);
NetworkInterface nic =
NetworkInterface.getByInetAddress(InetAddress.getByNam("192.168.2.23"));
mSocket.joinGroup(new InetSocketAddress(multicastGroup,
DEFAULT_MULTICAST_PORT), NetworkInterface.getByNam(adapterName));

/**
 * The following binds to a specific address and port (i.e. adapter address)
 * and then ONLY accepts UDP packets with destination equal to that IP.
 */
/*MulticastSocket mSocket = new MulticastSocket(new
InetSocketAddress("192.168.2.23", DEFAULT_MULTICAST_PORT));
mSocket.setReuseAddress(true);
mSocket.setSoTimeout(5000);
NetworkInterface nic =
NetworkInterface.getByInetAddress(InetAddress.getByNam("192.168.2.23"));*/
```

java multicast multicastsocket

edited Nov 9 '13 at 23:28

asked Oct 15 '13 at 22:28
 Kerry
4,377 6 44 82

Creating a Multicast Socket with SocketAddress means we are binding the socket to the SocketAddress right? So if we join to a Multicast group, why the socket unable to receive multicast datagrams???? – Sivanandham Dec 13 '18 at 13:04

2 Answers


▲ If you don't specify a local IP address when you create or bind it, it binds to 0.0.0.0, which means 'accept input via any NIC'. This is normally what you want.

1

▼ It is possible to bind to a specific IP address, which implicitly means the corresponding NIC, but some systems such as Linux seem to expect multicast sockets, if bound, to be bound to the multicast group itself. This doesn't make any sense to me: what if you want to join another group?

I think the best and most portable idea is to listen at 0.0.0.0 and join via either a specific NIC, or via *all* NICs, one at a time. The latter is necessary in multi-homed hosts unless you are confident that the default route to the multicast group is the one you want the join request sent out on, because that's what happens if you don't specify a join interface.

edited Oct 15 '13 at 23:01

answered Oct 15 '13 at 22:42
 user207421
271k 28 232 391

After further contemplation, I've added further info to the question and would appreciate your thoughts. – Kerry Oct 16 '13 at 11:33

If you use your option 3 you should also join via that interface, as I said above. It's still liable to the problem of receiving non-multicasts though. You could try binding to the multicast address itself, but it may not work on your platform, as I also said above. – user207421 Oct 16 '13 at 21:19

EJP I think we basically agree on what happens, but I should point out that in the tests I've done it's possible to bind a Multicast socket to a NON Multicast address and this essentially prevents any Multicast packet being received. I realise now that `join()` and `bind()` are two different things. It is also possible to use this MulticastSocket bound to a NON-Multicast address to receive datagrams destined to that NON-Multicast address. That makes sense as MulticastSocket is just a specialisation of DatagramSocket. I use Linux which allowed binding. I care not for Windows. – Kerry Oct 16 '13 at 21:58

I will post a summary of my tests soon for future reference if anyone else ponders this deeply on it... – Kerry Oct 16 '13 at 21:59

"It binds to 0.0.0.0, which means 'accept input via any NIC'" - this is not true – haelix May 28 '14 at 13:02

▲ I think that you are missing the point with your address binding:

0

▼ <http://download.java.net/jdk7/archive/b123/docs/api/java/net/MulticastSocket.html>
A multicast group is specified by a class D IP address and by a standard UDP port number. Class D IP addresses are in the range 224.0.0.0 to 239.255.255.255, inclusive. The address 224.0.0.0 is reserved and should not be used.

answered Oct 15 '13 at 22:55
 Germann Arlington
3,273 2 12 17

Missing the point now exactly? His multicast address looks OK to me at first glance. – user207421 Oct 15 '13 at 23:01

@EJP I am talking about the commented out code that does NOT work as I understood it `//MulticastSocket mSocket = new MulticastSocket(new InetSocketAddress("192.168.2.23", DEFAULT_MULTICAST_PORT));` – Germann Arlington Oct 15 '13 at 23:11

Germann, thanks for your reply. would you be able to comment on the additiona info I added? – Kerry Oct 16 '13 at 11:34

I did not check if traffic actually goes to ALL addresses - could be interesting, unfortunately I don't have enough computers at the moment to check myself. If you can set it up and report on your findings. – Germann Arlington Oct 16 '13 at 12:39

1 Germann, the thing I didn't realise is that `binding` and `joining` are two different things entirely. You can bind to ANY IP address, but you can only join a Multicast IP address. – Kerry Oct 16 '13 at 22:02