Algunas reflexiones sobre la comunicación Multicast

Si uno acude a la descripción de la clase MulticastSocket del API de sockets en Java

https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html

se encuentra con que todos los constructores de dicha clase habilitan la opción SO_REUSEADDR que permite la reutilización de direcciones. Dicha opción se utiliza muy a menudo en TCP/IP cuando nos interesa poder reactivar un servidor lo antes posible debido a una detención inapropiada. Si no se hiciese uso de dicha opción nos daría un fallo diciendo que el puerto está en uso y el sistema no lo liberaría hasta que pasasen varios minutos.

También se utiliza en servidores críticos cuando hay un elevado número de conexiones y donde una aproximación basada en aceptar una conexión desde un ServerSocket para luego derivar en un hilo el procesamiento de la comunicación con el cliente no es suficiente debido al cuello de botella que supone la aceptación de todas las conexiones en el mismo punto. Para eso se suele utilizar la opción SO_REUSEADDR de forma que se activan simultáneamente un número elevado de hilos donde todos ellos están aceptando conexiones y procesando la conexión aceptada. La diferencia fundamental entre SO_REUSEADDR y SO_REUSEPORT es que ésta última opción suele ser preferible al garantizar un equilibrio entre la carga computacional de los distintos hilos, cosa que no nos garantiza SO_REUSEADDR. Cada conexión usa un hash en el que intervienen la dirección IP local y remota así como el puerto local y remoto (asociación), de forma que se identifica unívocamente al proceso con el que se quiere comunicar en el servidor de forma que una vez abierto el canal los datos no se mezclen en recepción.

Ya centrándonos en la comunicación UDP y Multicast, el uso de SO_REUSEADDR es discutible, ya que en este caso lo que ocurre es que tenemos varios procesos usando el mismo puerto y que podrían estar robando datos entre sí. Pero hay una diferencia entre UDP y Multicast. Si consultamos el documento

http://www.kohala.com/start/mcast.api.txt

en una zona del mismo aparece el siguiente párrafo

```
More than one process may bind to the same SOCK_DGRAM UDP port if the bind()
is preceded by:

        int one = 1;
        setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one))

In this case, every incoming multicast or broadcast UDP datagram destined to
the shared port is delivered to all sockets bound to the port.  For backwards
compatibility reasons, THIS DOES NOT APPLY TO INCOMING UNICAST DATAGRAMS --
unicast datagrams are never delivered to more than one socket, regardless of
how many sockets are bound to the datagram's destination port.
```

Básicamente nos está diciendo que cuando la comunicación es Multicast o Broadcast, si enviamos un paquete de datos, éste es recibido por todos los sockets que hacen uso del mismo puerto. Pero por cuestiones de compatibilidad eso no ocurre con la comunicación Unicast por datagramas, donde el paquete se suministra únicamente a un socket y, por lo tanto, los procesos estarían robándose datos unos a otros.

Todo lo aquí descrito justifica la razón de por qué la práctica 1 funciona correctamente aunque ejecutemos todos los procesos en la misma máquina.