Objective: The goal of this assignment is to give you experience with socket programming. You will also become familiar with implementing a distributed algorithm and maintaining routing tables. This assignment is to be done independently.

Assignment: Using C or Java, your job is to design and implement the PIM-SM network-level multicast solution. To implement your solution, you need to specify the address of the rendezvous point (RP). To start your protocol, use the following command:

```
machine_name% pimsmd <multicast_group_address> <RP_IP_address> <operation>
```

where <operation> is one of RP, source, or receiver. Be sure your multicast_group_address is in the class D address range.

You should run your daemon on multiple machines, where you have at least one source, one RP, and at least one receiver (of course, you should also ensure your solution works with multiple sources or receivers).

The protocol should follow the basic operation of the PIM-SM protocol. The receivers should send join messages to the RP, who should in turn respond with a join-ACK. If the join-ACK is not received by the receiver, it should resend the join request. After some period of time, the source should start sending data packets to the multicast group through the multicast distribution tree that has been created.

Your solution should also enable receivers to leave the multicast tree through the transmission of a leave message. Once the leave message is received by the RP, it should stop transmitting data packets to that node.

You should keep track of some basic statistics, such as number of data packets transmitted by the source and number of data packets received by each receiver. As a finalize procedure, your daemon should print to the screen the appropriate statistics of each node. Also, while the daemon is running, each node should print a message to the screen when it sends or receives a control message. The message should be of the form

```
<action> <message_type> <senders_IP_address> <receivers_IP_address>
```

where <action> is either SEND or RECEIVE.

A couple of final notes: Your implementation should be able to handle dynamic joins and leaves. Also, to connect the receivers to the RP, you can simply open a socket between them. Thus your “multicast” solution may not be as optimized as a true multicast deployment, since the connections between the RP and the receivers will simply be through unicast.

Grading: The emphasis is on your design and implementation. Your output must demonstrate your implementation's ability to start an initial stream, handle additions and deletions of multiple receivers, and gracefully stop streaming once the initial sender has nothing more to transmit.

Turn-in: Submit a [your_user_name]-hw4.tar.gz file containing the following:
- Your source code
- A Makefile to compile your code
- OUTPUT containing the expected output generated by your program demonstrating dynamic additions and removals
- README with instructions on how to compile and run your program
- EVALUATION with a one-page evaluation of your design including any known bugs or limitations