Ambient Intelligence: Communication Protocol

Monica Arias Rivera

December 9, 2016

My implementation of a communication protocol on TinyOS has three distinct phases that are changed by the periodic timer *TimerToggle*: adding new motes to the list (broadcasting), transmitting information (communication) and removing unresponsive motes from the list (verification).

1 Broadcasting

The broadcasting phase happens in channel 6, and starts when we have just turned on the mote. It follows the next steps as a 'handshake' procedure to add all the motes that respond to its list:

1. Wait for a few seconds to listen to other motes broadcasts, with *TimerWait*, time determined by the mote's ID times 1 second
2. When *TimerWait* fires we send a *broadcast* message, set the *TimerOut* timer to wait 1second and we wait for an *answerToBroadcast* message
    - When *TimerOut* fires we start the broadcasting sequence from point 1
3. When a mote receives a *broadcast* message it sends an *answerToBroadcast* message
4. When a mote receives a *answerToBroadcast* message it sends a *channelProposal* message
5. When a mote receives a *channelProposal* message it sends a *channelConfirmation* message, adds the mote we have been communicating to its list and continues listening for more *broadcast* messages
6. When a mote receives a *channelConfirmation* message it adds the mote we have been communicating to its list, and continues listening for more *broadcast* messages

We can see that two motes have added each other to their lists because both sets of LEDs light up with a binary 5 at the same time. The next time one of them receives a broadcast from a known mote, it lights up with that mote's id. For instance, the first time motes 4 and 7 link, they both light up with 5. When mote 4 broadcasts again, mote 7 lights up with 4. And when mote 7 broadcasts again mote 4 lights up with 7.

2 Communication

To test the handshaking procedure previously explained, I implemented a communication phase that occurs on channel 8. When the mote enters this phase it sends an *information* message to the first mote on its list, and when an *information* message is received the mote responds with the same message. This creates a loop, and what we can see is that the LEDs blink with the other mote's ID. For instance, if mote 4 and 7 are communicating, mote 4 will blink 7, and mote 7 will blink 4.

3 Verification

Finally we verify that the motes on the list are still available. This verification occurs on channel 8, since we are confirming that there is still communication on the agreed channel. When the mote enters this phase it follows the next steps:

1. Send a *checkMote* message to the first mote on the list, and set *TimerCheck* to one second.

2. When a mote receives a *checkMote* message it responds with its own *checkMote* message. If it received a respond we set a flag variable *answeredReceived* to true.

3. When *TimerCheck* fires we check the *answeredReceived* flag, if it is false we remove that mote from the list; and we sent a *checkMote* message to the next mote on the list.

The feedback we have for this phase is that the mote's LEDs light up with the number of the mote they are checking for one second. If mote 6 has motes 3, 4 and 7 on its list, and they were added in that order, we will see the LEDs mark 3 for one second, then 4 for one second, then 7 for one second, and then they would turn off.

We know if the mote was removed from the list if the next time there is a broadcast they both signal 5, which means they re-added each other. Otherwise the next time there is a broadcast each other signals the other's ID, which means it is already in the list and it was not deleted.