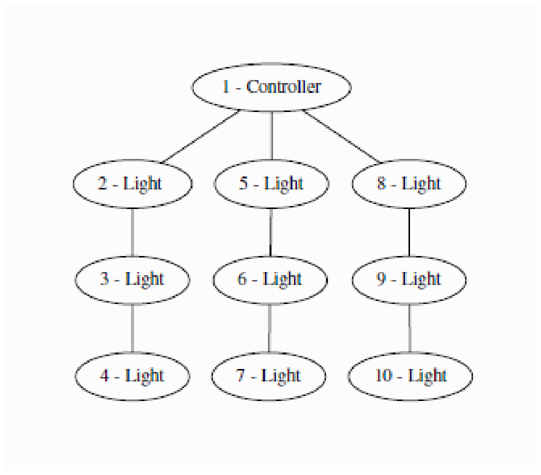
IoT Smart Lights Project

Alessia Guzzo - 10556670
Siddhant Samarth - 10758192

Introduction

The task of this project is to create a series of light patterns on a system consisting of 1 controller node and 9 "smart light" nodes, set up in the following structure. The node 1 is the controller node.



We have decided to use the following three patterns:

- Triangle: by lighting nodes 4, 6, 7, 10
- Cross: by lighting nodes 2, 4, 6, 8, 10
- Diamond: by lighting nodes 3, 5, 7, 9

These patterns must be repeated in the same sequence periodically and, after one pattern has been sent, all the nodes have to be switched off before switching them on according to the next pattern.
It has been assumed that the controller node (1) could only communicate with nodes 2, 5 and 8, which obviously have to be in the range of 1.
Finally, it is required to simulate the project with Cooja showing the nodes with their respective LEDs.

Implementation

We have used a multi-hop strategy to ensure all lights received the required instructions. The controller node communicates with the previously mentioned nodes by unicast messages, sending instructions for all the nodes.
To begin with, we have created two different directories with their relative source code, one for the controller mote and the other for the light motes, each containing the .h, the AppC.nC and the C.nC files.
Regarding the structure of the messages to be exchanged, in the file SmartController.h and SmartLight.h files, we have declared the structure of each communication packet which contains two variables: the **pattern vector** and the **sender ID**.

The pattern vector variable is an array with 9 components which are integer numbers nx_uint16_t, either with a value of 1 (light on) or 0 (light off): it allows us to send the patterns to every motes. The senderID is used because we need to keep track of which motes the message is coming from: with this info we can check the validity of the message received and also set every time the new sender of a message. If the senderID is N and the mote that is receiving the message is N+1 the message will be delivered correctly, the only case in which this rule is not valid is when controller (1) send messages to 5 and 8 that aren't N+1.

Controller

First of all, for the code writing part of the controller we focused on the use and setting of timers. In particular, we have declared two **TimerMilliC** components, the one used for the ON phase that is the one during which sending a pattern, and the other for the OFF phase, during that the off pattern (all LEDs off) is sent to all nodes. The **TimerOff (Timer0)** is the first to be initialized after the success of the StartDone event, with startOneShot of 3 seconds. When it is fired, the correspondent event is called and it consists of different steps: the creation of a new payload followed by the assignment of senderID and pattern_vector values (in this case pattern_vector_off), then we proceed to the unicast send of the messages using the function AMSend.send.
Regarding this part we have chosen to use a support variable called 'mex_num' to manage the delivery of the 3 messages one by one from controller to motes 2, 5 and 8 since we thought it would be safer to send a package after having the guarantee that the previous one was sent correctly.
After the successful sending of both message 2 and 5, the Timer0.startOneShot with period 50 ms is called recursively to continue the process and unicast the packet to the next receiver. Instead, after the sending of message to mote 8, **Timer1**.startOneShot (**TimerOn**) is called to turn on all lights after a delay of 5 seconds (thus maintaining the pattern for a while).
Thereafter, the event of Timer1 fired is called. In this part we do the same procedure as before with the difference that we use the variable 'pattern' to manage the circular succession of the 3 patterns. To sequentially circulate through the patterns we have created a pattern vector, which holds a value of 1, 2, or 3. After one pattern delivery we set pattern++, or pattern=1 if it is the last pattern, and we have to restart the cycle.
In this way, after all the lights have been switched off, it will start the period in which the light pattern is sent and this procedure will continue roundly.
In addition, we have used the locked variable in order to ensure no loss of packets.

Lights

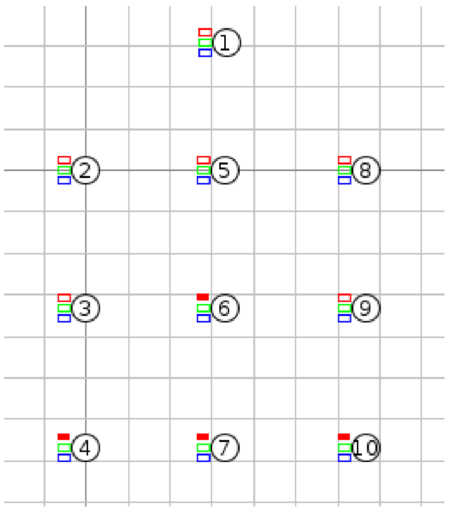
For the code part of the light mote we mainly have handled the receive event, which is called when a packet is delivered to a light mote.
For doing this we first have to extract the payload of the received packet in order to check the pattern sent from the controller and to check the senderID.
Then, we have created a payload of the size we want to set the new packet to be sent by the current mote.
Afterward, we have checked if senderID corresponds to the controller (1) or to the previous numerical node, to avoid both confusion caused by a mote receiving a packet from another mote in range and a situation where a mote broadcasts the same packet multiple times. If this condition is fulfilled we assign the pattern vector received in the correspondent field of the light mote and then check if the element of the pattern vector with index corresponding to the mote is 0, in this case we have called Leds.led0Off() or is 1, so we have used Leds.led0On().

In both cases we inspect if the senderID is equal to 3 or 6 or 9, in these cases it means we are in the last node of the branch so the current mote just does not send anything. Alternatively, we set the senderID to the TOS_NODE_ID of the current mote and call the AMSend.send function in order to deliver the new packet, consisting of the sender ID and the pattern vector, to the TOS_NODE_ID+1.

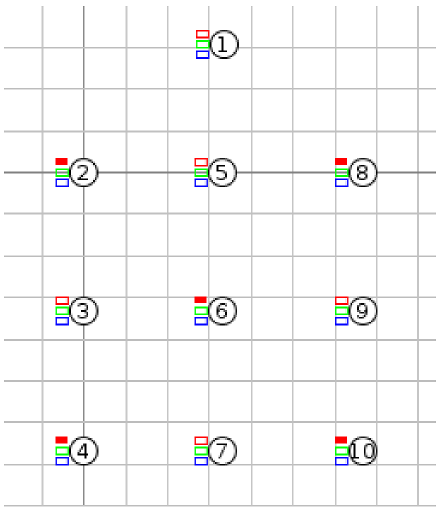
Simulation with Cooja

For debugging purposes we have added the components PrintfC and SerialStartC in the file SmartControllerAppC.nC for using printf() and printfflush() functions, this has allowed us to obtain a log file detailed in every crucial part of the simulation.
We have created a simulation with node 1 containing the executable (main.exe) of SmartController and the other 9 containing the executable of SmartLights, all the motes are of **Sky type**. In order to allow the communication between mote 1 and 2, 5, 8 we have set them one in the range of the other, the same way for every other motes with respect to his successor.
Regarding the LEDs, we have used only the first of every mote which is the red one.
Finally, we have executed the simulation with speed limit 100% in order to see all the transitions in a clear way and we have saved the log file with all our debugging statements. We have also modified the log file a little bit for the purpose of making it more readable since printf also prints random characters on Cooja.

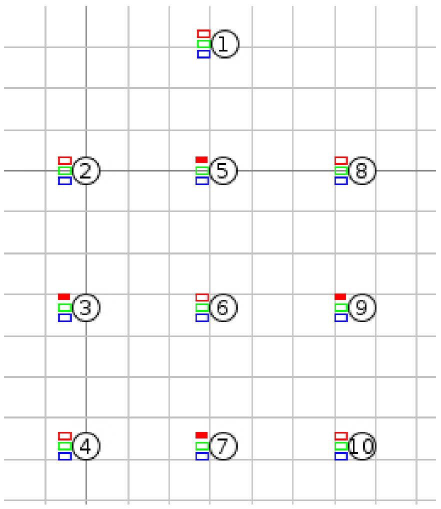
The three patterns look like this, respectively:



TRIANGLE



CROSS



DIAMOND