Project, Part 2 – Due Monday 10/28 11:59 PM

In the first part of the project you implemented a command interface for the **Telosb** nodes and the underlying service responsible for carrying out the commands. In this part we expand on that by adding wireless communication, so commands send to Node A over UART are transmitted to Node B over radio and the response is send all the way back and displayed in the terminal connected to Node A. In order to make this power efficient you are to implement a simple version of a receiver initiated MAC protocol as well.

Purpose

- To understand low power communication in packet based radios.

Prerequisites

- Read the TinyOS TEP, **Packet Protocols** http://www.tinyos.net/tinyos-2.1.0/doc/html/tep116.html
- Read the seminar paper, **RI-MAC: a receiver-initiated asynchronous duty cycle MAC protocol**.

Deliverables

- Code:

  A tar/zip containing the application folder, compilable when extracted to the cs450 directory.

- Report (at most 3 pages using 10pt type):

  A PDF containing: Design Choices, Bug Report, Evaluation, Future Work, Summary.

  Design Choices: You often have multiple choices when writing code. Let us know the reasoning behind the choices you made. Discuss the pros and cons.
  Bug Report: If something is not working, that you know of, let us know.
  Evaluation: Show us your application is working as intended, e.g., screen shots, logs, etc.
  Future Work: How would you improve the application? You are encouraged to include changes to the constraints set by the assignment.

  Remember, the only way we can know about any particularly clever solutions you've made, and give you credit for them, is if you tell us about them in the report. Also, if your code is not working properly, please document what you have done and achieved.

- Groups:

  You are allowed to form groups (max. 3 students) and share the source code. However, you have to write and hand-in an individual report each.

Sample Application

In the GIT cs450 folder you will find a new sample application, **tosProject2**. This application shows a possible solution for Project 1 including a UART communication protocol and command set. You are free to build upon this application or use your own solution from Project 1. The frame formats are described in the **ProbeC.nc** file. CRC check is not enabled although the frame format supports it. The file also includes some sample commands to test with. The application also has two message sockets included, arbitrarily named **Data** and **Control**, which you can use as a starting point for the radio communication.

Use the **tosProject2** application to ensure you have a working radio set and familiarize yourself with the TinyOS AM messages. However, first modify the Makefile with your assigned radio channel:

	Channel
Dunyuan	11
Zihan	12
Renyuan	13
Gregory Chandler	14
Noah Lampel	15
Kunal Anil	16
Siwei	17
Luke Emmett	18
Nir Michael	19
Michael Anthony	20
Nicholas Jerome	21
Yanan	22

Assignment

1. Implement a receiver initiated MAC.

   The three files: CC2420RadioC, ReceiverInitiatedC and ReceiverInitiatedP are where the radio duty-cycling is going to be implemented. The CC2420RadioC file replaces the standard component wiring with ReceiverInitiatedC/P when **CFLAGS+=-DCS450_RECEIVER_INITIATED** is enabled in the Makefile. Basically, we are intercepting the radio stack on the same level where the standard low-power listening MAC is implemented in TinyOS:

SENDING PART	RECEIVING PART	
CC2420TInyOSNetworkC	CC2420TInyOSNetworkC	
UniqueSenderC LinkC	UniqueSenderC	
LplC	LqlC	<<< This is the layer we are working on
CsmaC	CsmaC	

The ONLY file you should be working on to implement the receiver initiated MAC is ReceiverInitiatedP. **Unlike the previous assignment, you only need to code the body of the functions with missing code sections. The MAC itself can be implemented by only adding these missing lines of code.** You can read more about the CC2420 radio stack in Section 2 in the TEP, http://www.tinyos.net/tinyos-2.1.0/doc/html/tep126.html

Note that the **Receive.receive** event has the return value **message_t***, which means that you are supposed to swap buffers when you are passing message pointers between layers.

The Receiver Initiated MAC consists of four radio states:

1. STATE_RADIO_OFF
2. STATE_RADIO_STARTING
3. STATE_RADIO_ON
4. STATE_RADIO_STOPPING

where the only possible state transitions are 1 -> 2 -> 3 -> 4 -> 1 -> ... using the functions SubControl.start and SubControl.stop in state 1 and 3, respectively. Calls to these two functions in state 2 and 4 will be ignored and no startDone/stopDone will be signaled. Care must be taken to ensure that the system remains in a well-defined state.

Two timers:

1. TimerTurnRadioOn / BEACON_OFF_TIME_MS
2. TimerTurnRadioOff / BEACON_ON_TIME_MS

are used to cycle between turning the radio on and off. The two variables

BEACON_OFF_TIME_MS and BEACON_ON_TIME_MS specify how long the radio should be kept off and on, respectively, and can be used to control the radio duty-cycle. Here the trick is to set the on time low, yet still be able to send and receive packets, and the off time high without losing packets either.

When no message is waiting to be transmitted, a beacon is send once the radio has been switched on and kept on long enough to both receive a reply and automatically transmit an acknowledgement using radio hardware.

When a message is to be send, the radio is kept on regardless of the timers above, which are then only used to transmit a beacon following the regular schedule. Once a beacon is received from the intended recipient, the message is transmitted immediately. This could potentially lead to packet collisions if multiple nodes try to transmit to the same destination. A missing acknowledgement would then indicate that the packet did not reach its destination. Ideally, the packet should then be retransmitted using a random backoff timer the next time the beacon is heard. However, in our implementation we simply retransmit the message at the next beacon.

Besides the radio states we use three control flags to ensure the proper program flow:

SPLITCONTROL_STOPPED - True, when the upper layer has switched off the radio
SUBSEND_NOT_BUSY - True, when neither a beacon nor a message is being send
SEND_MESSAGE_PENDING - True, when a message is waiting for a beacon to be send

These flags are used to ensure that we keep the radio state consistent and do not transmit multiple messages simultaneously or accept more than one message from the upper layer at a time.

2. Implement command and data forwarding.

Extend the command set to include remote commands equivalent to the local set. I.e., remote commands should be transmitted over the radio and executed on the remote node with the result transmitted back again:

g. command: single read of *<sensor>.*
output: timestamp and raw sensor value.

h. command: continuous read of *<sensor>* every *<interval>.*
output: continuous timestamp and raw sensor value output.

i. command: stop all activity.
output: acknowledgement of command.

j. command: log all sensors to flash every *<interval>.*
effect: read all sensors and store them to flash as one record every interval.
output: acknowledgement of command.

k. command: report current write-cookie.
output: 32 bit write-cookie.

l. command: read *<number of records>* from *<address cookie>.*
effect: seek to the given address and read the number of records specified.
output: timestamp and raw sensor values for each record.

You are allowed to define how commands, parameters, and outputs are represented and coded over the radio. The *<sensor>* is one of the four sensors on the Telosb. The *<interval>* should support the range from 1 second to 1 day. The *<number of records>* should at least support the range 1 to 3.

Grading

There are many different solutions to the assignment above and it is up to you to choose which one to implement and discuss your choice in the report. We will grade the assignment based on (1) how well it works, (2) how robust the implementation and communication protocol are, (3) the written report. We will ***not*** be grading the assignment on how efficient your implementation is (that will be in Part 3 of the project).

To make things easier for yourself start out with a generous duty-cycling and low-frequency sampling rates. Once you have a working prototype you can start tweaking the duty-cycle and improving the sampling rate until you reach the assignment goal.