# 18748 - Wireless Sensor Networks – Spring 2016

# Carnegie Mellon University

### Lab Assignment 1

### Due Date: 6th February 2016, 11:59 PM

**General Lab Setup:**

1. Each team will be provided with a **sensor networking kit**, which includes
   a. *3 FireFly Nodes with attached sensor boards*
   b. *1 USB cables to connect the PC with FireFly Nodes*

2. **Software toolchain** required:
   a. binutils-2.20
   b. gcc-core-4.4.2
   c. avr-libc-1.6.7
   d. avrdude-5.11

   You have two options to setup the toolchain.

   (1) **[Recommended]** Manual installation.
   For installing the toolchain directly on your host machine, refer to
   http://www.nanork.org/projects/nanork/wiki/Linux-install (Linux)
   http://www.nanork.org/projects/nanork/wiki/Osx-install (OSX)
   http://www.nanork.org/projects/nanork/wiki/Windows-install (Windows + Cygwin)

   (2) Using a pre-installed virtual machine image.
   The VirtualBox VM image installed with Ubuntu and necessary tools can be downloaded at
   http://timex.ece.cmu.edu/wsn/nanorkVM/linux-dev.tgz

   Download and untar the tgz file and follow the instructions on
   http://www.nanork.org/projects/nanork/wiki/vm

3. **Repository access:**
   a. SVN access will be provided for each group to check out the nano-RK sources for your projects.
   b. Skeleton code for each lab will be automatically added to the svn repository when each lab is released.

4. **Lab submission guide:**
   a. Solutions (source code and a write-up) for each lab should be committed to each team's repository before the deadline.
   b. Don't forget to **"svn add"** new files you wrote. Any files not committed to the repository by the deadline will not be scored.

**Lab 1 - Objectives:**

1. Get familiarized with the **FireFly** platform and **nano-RK** real-time operating system
2. Characterize the variation of **Received Signal Strength Indication** with **Distance**

**Lab 1 - Assignment 1 - Getting started with the FireFly platform:**

1. Connect the PC to the FireFly node using the USB Cable
2. Use "**dmesg**" to make sure that **FTDI USB Serial Device Converter** got detected and attached to ttyUSBx (ex. ttyUSB0)
3. Check-out the nano-RK source code from each team's SVN repository (not from nanork.org).
4. Build the Self-Test code:
   1. **cd <your_local_repository>**
   2. **cd projects/tests/self-test**
   3. Set platform type in the **makefile**
      **PLATFORM = firefly3**
      **PROGRAMMING_PORT=/dev/ttyUSB0** (the programming port in dmesg)
   4. **make clean**
   5. **make**
5. **make program**
   Downloads the code to the FireFly Node using avrdude.
   Make sure that the device signature gets detected as 0x1e9704.
6. **minicom -s**
   1. Type A and set the path to */dev/ttyUSB0* (substitute the debugging port as shown by dmesg)
   2. Set the baudrate to *115200 8N1 with no Hardware / Software Flow Control*
   3. "*save setup as dfl*" to save as default and exit
7. You should be able to something like this:
   *Self Test Cycle : 1*
   *TX status : OK*
   *RX status : NO PKT*
   *Max wakeup time : 0*
   *OS tick time : 114 114 114*
   (Use Ctrl-a x to quit, Ctrl-a o for options, Ctrl-a z for help)
8. Download the code to another FireFly node and make sure you can both *Receive (RX)* and *Transmit(TX)* packets when the nodes are separated by a few feet
9. Refer to http://www.nanork.org/projects/nanork/wiki/Quick-Start for more details
10. No write-up and code submission required for Assignment 1.

**Lab 1 - Assignment 2 - Characterizing the variation of Received Signal Strength Indication with Distance**

1. Skeleton code for this assignment is available in
   **<your_local_repository>/projects/lab1/assignment2/**
   Contents of this folder include:
   *main.c*
   *makefile*
   *nrk_config.h*
2. main.c provides working code that transmits a packet every 1.5 seconds and receives packets from other nodes

3. NOTE: make sure that the **bmac_init(channel_number)** function call, uses the same channel as assigned to your team, otherwise you will interfere with other teams
4. Blue LED indicates the packets being transmitted
5. Orange LED indicates the packets being received
6. Understand the working of the code at a high level
7. Build the code and download it to two different nodes

   **make clean**
   **make**
   **make program**
8. Use **minicom** to observe the change in **RSSI** values when the **distance** between the node changes
9. Whenever a packet is received, the corresponding **Received Signal Strength Indication (RSSI)** value is stored in the local signed 8-bit integer called "*rssi*"
10. Dynamic range of the rf231 chip on the FireFly node is around **100 dB** and the raw Energy Detection(ED) value usually varies from **0 to 84**
11. Modify the code in the following manner
    a. Remove the segments of the code that manipulate the Blue and Orange LEDs
    b. Create a new function *display_rssi(rssi)* that takes in the *rssi* value and displays it using the LEDs
       - Observe that you have 4 LEDs and therefore you can display 16 different RSSI values
       - Use your available output LEDs wisely, to obtain uniform data points for the *RSSI vs Distance graph*
       - Call the function *display_rssi(rssi)* from *rx_task* at the appropriate location
12. Build this modified code and download it to two different nodes

    **make clean**
    **make**
    **make program**
13. Turn on the nodes and observe that when the nodes are closer the RSSI value is higher
14. In order to obtain an exact characterization of the variation of RSSI with Distance
    - Take the pair of nodes outdoors to a location where there is less RF interference and distances can be measured
      - Preferably the Gesling stadium (Phew!!! Finally a reason to go into the snow :D...)
    - Place the nodes at varying distances from each other and note down the RSSI readings
15. Convert the distances to meters and plot a graph that summarizes your measurements
16. Compute the path loss exponent from your measurements
    - Path loss exponent is given by L = 10 n log10(d) + C
      - L is the Path Loss
      - n is the Path Loss Exponent
      - d is distance in meters
      - C is a system constant
    - For two different data points, we can see that
      - (L2 - L1) = 10 n log10(d2/d1)
        - RSSI is linearly related to path loss, (L2-L1) = (RSSI2 - RSSI1)
      - As d2 and d1 are known from the data points, one can compute the value of 'n' as follows:
        - n = (L2 - L1) / (10 * log10(d2/d1))
      - Fixing the value of (L1, d1), compute the values of 'n' from other measurements and estimate the approximate value of 'n'
      - Also estimate the variance and standard deviation of your measurements
    - Write a **1-page report** showing your graph and detailed calculations.

- Commit this write-up and your modified source code to your repository.

**Lab 1 - Assignment 3 – Understanding task priorities and response times**

1. Skeleton code for this assignment is available in
   **<your_local_repository>/projects/lab1/assignment3/**
   Contents of this folder include:
   *main.c*
   *makefile*
   *nrk_config.h*
2. main.c provides working code that creates two tasks that they take a specific amount of time to execute.
   - Task 1 – Period: 1s, Priority: 1
   - Task 2 – Period: 2s, Priority: 2
3. Build the code and download it to a FireFly node
   **make clean**
   **make**
   **make program**
   Each task is released periodically, and uses **nrk_time_get(nrk_time_t *var)** to print out the completion time of each period.
   Using the values printed over minicom, estimate each task's **worst-case response time**.

   Note: Response time is "completion time – release time", and release time is the time instant the task becomes ready to execute (e.g., release times of Task 1 = 0s, 1s, 2s, ...). So, the worst-case response time is max (completion time – release time).

4. Change the task priorities as follows:
   - Task 1 – Period: 1s, Priority: 2
   - Task 2 – Period: 2s, Priority: 1
5. Build the code, download and estimate the worst-case response times of tasks 1 and 2. Explain any differences that you see.

6. **Bonus 1**: Find a tight CPU reserve budget for each task.

   The skeleton code sets each task's CPU reserve budget equal to the task period, but the actual amount of CPU budget required by each task is much smaller than that. Find a tighter (smaller) CPU reserve value for each task which does not crash the sensor node. And briefly explain how you find it.

7. **Bonus 2**: Plot graphs for both tasks to show their response times.

   The x-axis should show time and the y-axis should show the task execution state (e.g., executing = 1, not-executing = 1). The resulting plot should resemble a square wave. Do that for the two priority-assignment cases.

   Note1: Graph can be plotted using your favorite plotting tool (gnuplot, matlab etc.)
   Note2: Data can be collected over minicom for about 30 data points.