# Networking Level Laboratory

## Sensing and Data Visualization on PC

Developed by UNIVERSITY of HOUSTON ♦ UNT UNIVERSITY OF NORTH TEXAS Discover the power of ideas ♦ SAN JACINTO COLLEGE 2009-2012

## Purpose/Objective:

The goal of this experiment is to learn how to acquire data from sensors in TinyOS. It demonstrates a simple sensor application that periodically takes sensor readings and displays the values on the LEDs. Then students learn how to integrate the sensor network with a PC, where the sensor readings are visualized with a dedicated graphical user interface. This will allow you to collect data from a sensor network, send commands to mote nodes, and monitor the network traffic. Students will also learn the Java-based infrastructure for communicating with motes, and display the collected data as waveform on a graphical user interface (GUI).

## Introduction

Sensing is an integral part of sensor network applications. Usually sensing involves two tasks: configuring a sensor (and/or the hardware module it is attached to, for example the ADC or SPI bus) and reading the sensor data. It is necessary for any sensing application to know the configuration details (such as ADC input channel, the required reference voltage, etc.) of an attached sensor, before it can read data from it. In TinyOS 2.x platform, the sensing applications (such as Sense or RadioSenseToLeds or Oscilloscope) are independent of the sensing platforms. That is, it makes no difference to compile and download the program to telosb (which has integrated sensors on board) or micaz family (which uses external integration sensor board) and collect sensor readings. Going through this experiment will allow you to answer the following questions: (1) Since the sensing applications (such as Sense or RadioSenseToLeds or Oscilloscope) component only uses standard data acquisition interfaces (Read, ReadStream, or ReadNow), who is in charge of defining which sensor it samples? (2) Who is responsible for configuring which sensor to read from? (3) How can these sensing applications know the answer to the questions such as "what is the value range of the sensor data" or "should a temperature reading be interpreted in degrees Celsius or Fahrenheit"? (4) With multiple sensors on an integrated sensor board, how the sensing applications know which data were read from which sensor, and visualize it accordingly?

### The DemoSensorC Component

Every DemoSensorC component has the following signature code in its configuration file:

```
generic configuration DemoSensorC()
{
    provides interface Read<uint16_t>;
}
```

However, its implementation differs from platform to platform. For example, the DemoSensorC initiatives VoltageC for the telosb platform, which reads data from the MCU-internal voltage sensor. On the other hand, since the micaz family does not have any built-in sensors, its DemoSensorC uses system library components such as ConstantSensorC or SineSensorC, which return "fake" sensor data. In summary, the platform dependent DemoSensorC component provides a generic mechanism to redirect sensor data acquisition to platform independent sensing applications like Sense or RadioSenseToLeds or Oscilloscope. Usually the configuration of a sensor is done in the component that DemoSensorC instantiates. Hence, every platform that wants to run sensing applications such as Oscilloscope, Sense or RadioSenseToLeds has to provide its own version of DemoSensorC.

New sensor boards are typically coming with their own version of DemoSensorC (e.g., the *basicsb* sensorboard for the mica-family of motes define DemoSensorC.nc to be the light sensor on that board).

Let’s use the Sense application to demonstrate how DemoSensorC is used.

## The Sense Application

Sense is a simple sensing application that periodically samples the default sensor and displays the lowest three bits of the readings on the LEDs. You can find it in the directory: **/opt/tinyos-2.x/apps/Sense**. Let’s examine its configuration file and signature code in its implementation.

```
configuration SenseAppC { }
implementation {
components SenseC, MainC, LedsC, new TimerMilliC();
components new DemoSensorC() as Sensor;

SenseC.Boot -> MainC;
SenseC.Leds -> LedsC;
SenseC.Timer -> TimerMilliC;
SenseC.Read -> Sensor;
}
```

Implementation file:

```
module SenseC {
uses {
     interface Boot; I
      nterface Leds;
      interface Timer<TMilli>;
      interface Read<uint16_t>;
       }
}
```

The sequence of actions in the SenseC.nc implementation is as follows: SenseC.nc uses the Bootinterface to start a periodic timer after the system has been initialized. Every time the timer expires SenseC.nc signals a timer event and reads data via the Read<uint16_t> interface. Reading data operation is divided into a command Read.read() and an event Read.readDone(), referred to as a split-phase operation. Thus every time the timer expires, SenseC.nc calls Read.read() and waits for the Read.readDone() event. When data is signaled in the Read.readDone() event, SenseC.nc displays it on the LEDs: the least significant bit is displayed on LED 0 (0 = off, 1 = on), the second least significant bit is displayed on LED 1 and so on.

Note that in the configuration part, the following two lines of configuration code indicate that the generic DemoSensorC component provides the Read<uint16_t> interface to SenseC.nc.

```
components new DemoSensorC() as Sensor;
 SenseC.Read -> Sensor;
```

## The Oscilloscope Application

`Oscilloscope` is a sensing application not only periodically samples the default sensor via `DemoSensorC`, but also broadcasts a message with 10 accumulated readings over the radio. This can be picked up by the BaseStation node and relayed to PC via serial port and visualized in the designated GUI.

To run the `Oscilloscope` application, you will need at least two nodes: one node attached to your PC running the `BaseStation` application; and one or more nodes running the Oscilloscope application.

Let's take a look at the `OscilloscopeAppC.nc` configuration file and the signature code in its implementation file as shown in the figure below:

```
configuration OscilloscopeAppC{}
implementation{
components OscilloscopeC, MainC, ActiveMessageC, LedsC,
new TimerMilliC(),
new DemoSensorC() as Sensor,
new AMSenderC(AM_OSCILLOSCOPE),
new AMReceiverC(AM_OSCILLOSCOPE);

OscilloscopeC.Boot -> MainC;
OscilloscopeC.RadioControl -> ActiveMessageC;
OscilloscopeC.AMSend -> AMSenderC;
OscilloscopeC.Receive -> AMReceiverC;
OscilloscopeC.Timer -> TimerMilliC;
OscilloscopeC.Read -> Sensor;
OscilloscopeC.Leds -> LedsC;
}
```

Implementation Module

```
module OscilloscopeC{
uses {
   interface Boot;
   interface SplitControl as RadioControl;
   interface AMSend;
   interface Receive;
   interface Timer;
   interface Read;
   interface Leds;
  }
}
```

`Oscilloscope` is an integration of different building blocks introduced in previous parts in order to collect sensor readings and visualize them on PC. In short, like `Sense`, `Oscilloscope` uses `DemoSensorC` and a timer to periodically sample the default sensor. After it has gathered 10 sensor readings `OscilloscopeC` puts them into a message and broadcasts it via the AMSend interface. It uses the `Receive` interface to synchronize and the `SplitControl` interface to switch the radio on.

## Experiment Procedures:

**Step 1: Program Mica node with `Sense` application:**

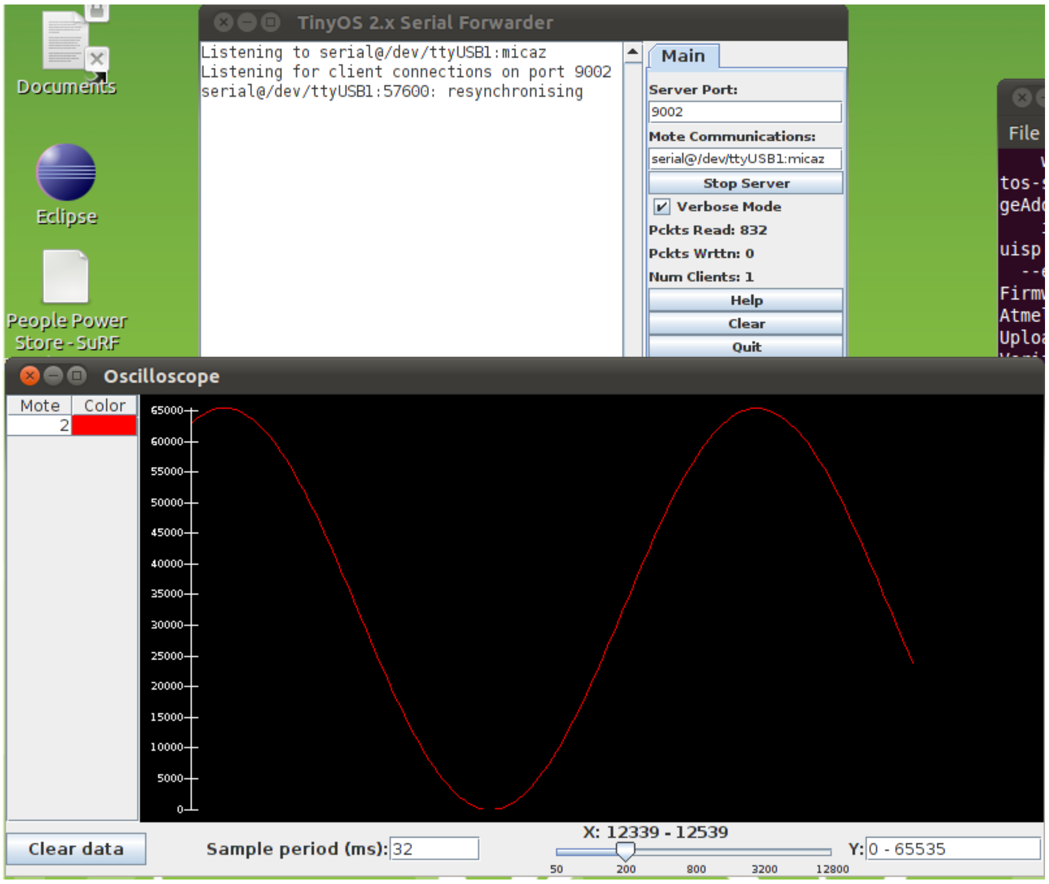
1. Attach a Mica node securely onto the programming board (MIB520).
2. Change the terminal's active directory to '**/opt/tinyos-2.x/apps/Sense**'.
3. Use command '**make micaz install.1 mib510,/dev/ttyUSB0**' to download the **`Sense`**.

Once you have installed the application, you will observe the three least significant bits of the sensor readings are displayed on the node's LEDs (0=off, 1=on).

**Challenge:** modify the Sense application to send the sensed data to PC via serialForwarder (Lab4)

**Step 2: Program Mica node with** Oscilloscope **application:**

1. Attach a Micaz node securely onto the programming board (MIB520).
2. Change the terminal's active directory to '**/opt/tinyos-2.x/apps/BaseStation**'.
3. Use command '**make micaz install.1 mib510,/dev/ttyUSB0**' to download the **`BaseStation`**.
4. Attach another Micaz node securely onto the programming board (MIB520).
5. Change the terminal's active directory to '**/opt/tinyos-2.x/apps/ Oscilloscope'**.
6. Use command '**make micaz install.2 mib510,/dev/ttyUSB0**' to download the `Oscilloscope`.
7. Detach node 2, and reattach node 1 (with installed BaseStation) onto MIB520 so it connects to PC.
8. Turn on node 2.
9. Running the Java GUI to visualize the sensor readings on your PC:
   1) Go to '**/opt/tinyos-2.x/apps/ Oscilloscope/java**' and use command '**make**'.
   2) Use command '**java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB2:micaz**' to connect to node 1 with BaseStation running using SerialForwarder.
   3) Use command '**./run**' to see a similar display as shown below.



```
Listening to serial@/dev/ttyUSB1:micaz
Listening for client connections on port 9002
serial@/dev/ttyUSB1:57600: resynchronising
```

Server Port: 9002
Mote Communications: serial@/dev/ttyUSB1:micaz
✓ Verbose Mode
Pckts Read: 832
Pckts Wrttn: 0
Num Clients: 1

| Mote | Color |
|---|---|
| 2 | 🟥 |

Sample period (ms): 32
X: 12339 - 12539
Y: 0 - 65535

The x-axis is the packet counter number and the y-axis is the sensor reading. To change the sample rate, you can edit the number in the "sample period (ms)" input box. When you press enter, a message containing the new rate is created and broadcast via the BaseStation node to all nodes in the network. You can clear all received data on the graphical display by clicking on the "clear data" button.

The `Oscilloscope` (or `Sense`) application displays the raw data as signaled by the `Read.readDone()` event. However, the GUI only let you adapt the visible portion of the y-axis to a plausible range (at the bottom right), but cannot adjust the scope based on how the values should be correctly interpreted.

**Challenge:** modify the `Oscilloscope` application to read in the light sensor data, using MTS300.

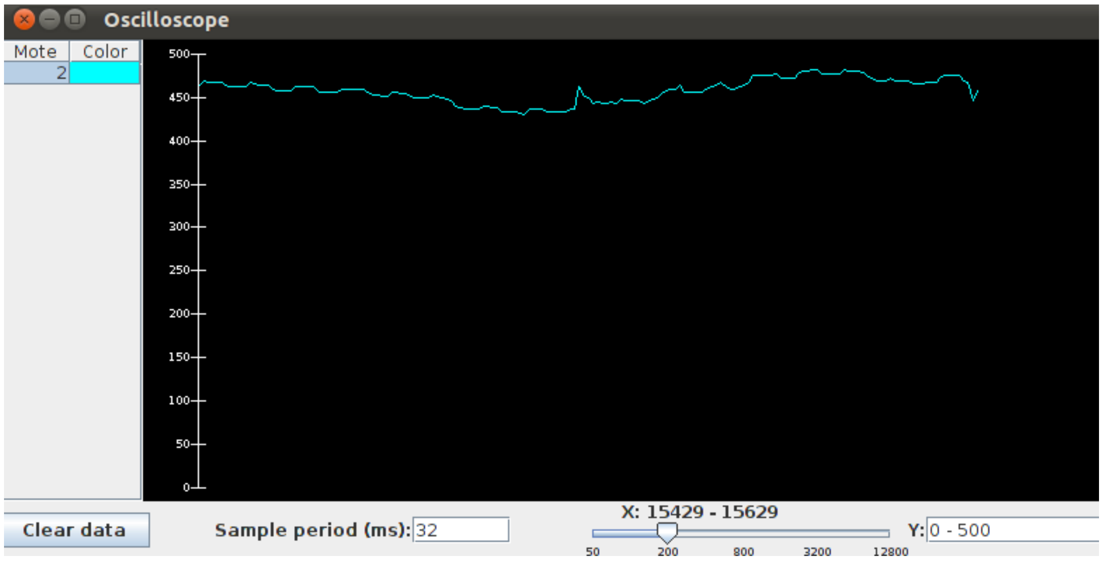
**Step 3: Modify the Default Sensor (DemoSensorC.nc):**

1. Change the terminal's active directory to '**/opt/tinyos-2.x/tos/platforms/micaz**'
2. Open the file '**DemoSensorC.nc**' in a text editor, such as 'emacs' using command:
   - emacs DemoSensorC.nc&
3. Change the line to '**components new PhotoC() as DemoSensor;**' or '**components new PhotoC() as DemoChannel;**' if it does not use PhotoC().

   This ensures that all calls to the DemoSensor to be directed towards the photo sensor '**PhotoC ()**' instead of the onboard voltage meter or other fake sensor such as SineSensor we used in previous step.

Now that the DemoSensor file has been changed the Oscilloscope application can be loaded onto a mote and it will make use of our photo sensor rather than the onboard voltage meter.  The next few steps will install the Base Station and Oscilloscope applications onto two different motes.

4. Attach a Micaz node securely onto the programming board (MIB520).
5. Change the terminal's active directory to '**/opt/tinyos-2.x/apps/Oscilloscope**'.
6. Use command '**SENSORBOARD=mts300 make micaz install.2 mib510,/dev/ttyUSB0**' to download the `Oscilloscope` application.
7. Attached the node 1 with BaseStation to MIB520. Turn on the power of node 2.
8. Change the active directory to '**/opt/tinyos-2.x/apps/Oscilloscope/java**'
9. Use serialForwarder we learned before to stream the sensor data to PC via MIB520 serial port: use command '**java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB1:micaz**'
10. If you wish to view the raw data received by the base station node, use command '**java net.tinyos.tools.Listen**'.
11. Use command '**./run**' to see a similar display as shown in the figure below.



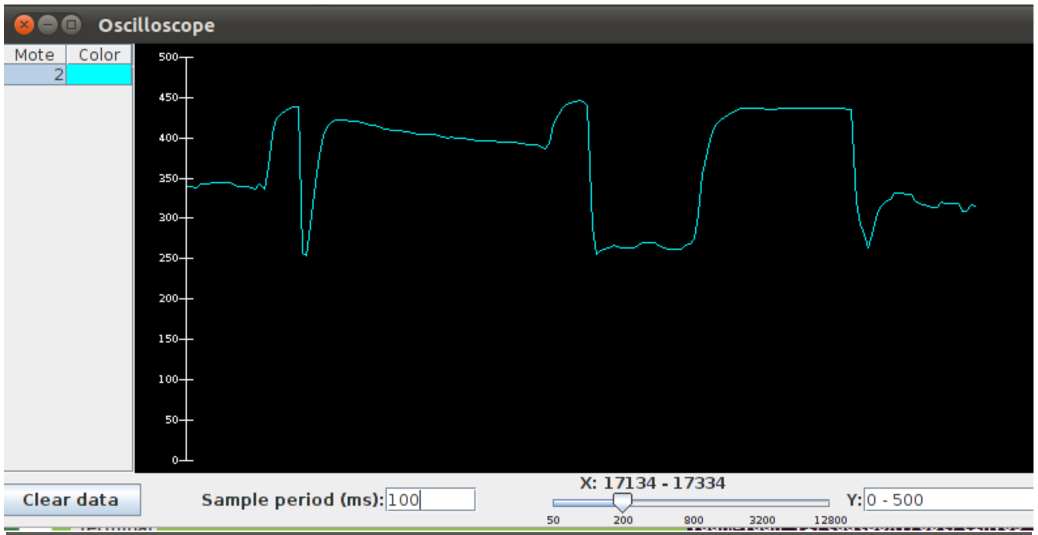
Sample period (ms): 32

Y: 0 - 500

Play with the GUI, for example, send a command over the air to increase the duration of data sampling from the sensor by increase the "Sample period (ms)" from 32 to 100, and see the changes as shown in the figures below.  Similarly, by adjusting the range of the x-axis, you can see all the data collected.

Which sensor data you are reading? Which DemoSensorC.nc file is actually used?

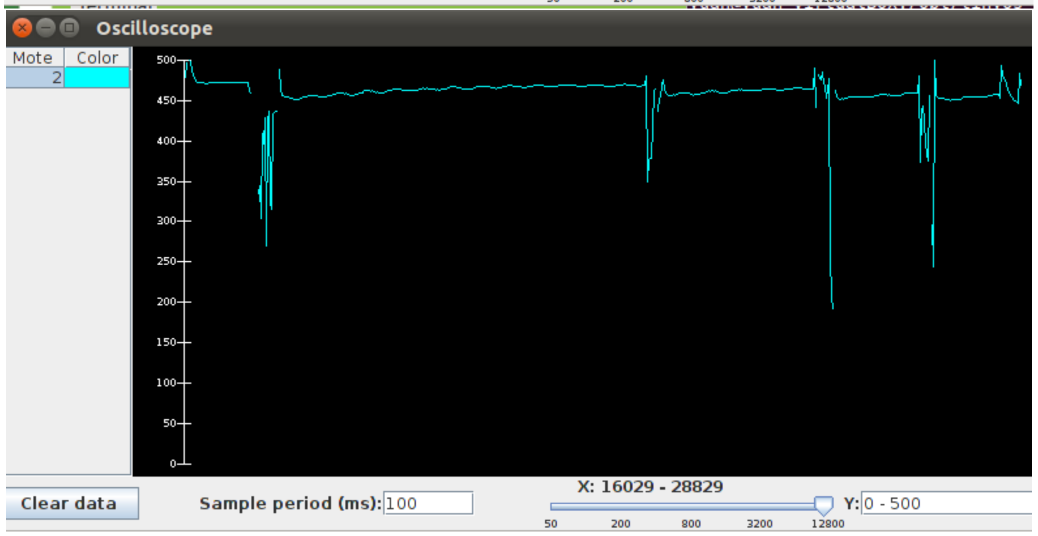
**Challenge:** Modify the application to read in both temperature and light sensor data using mts300 and display them in the GUI.



Sample period (ms): 100

X: 17134 - 17334

Y: 0 - 500



Sample period (ms): 100

X: 16029 - 28829

Y: 0 - 500

## Questions to answer in the lab report:

(1) Since the sensing applications (such as Sense or RadioSenseToLeds or Oscilloscope) component only uses standard data acquisition interfaces (Read, ReadStream, or ReadNow), who is in charge of defining which sensor it samples?

(2) Who is responsible for configuring which sensor to read from?

(3) How can these sensing applications know the answer to the questions such as “what is the value range of the sensor data” or “should a temperature reading be interpreted in degrees Celsius or Fahrenheit”?

(4) With multiple sensors on an integrated sensor board, how the sensing applications know which data were read from which sensor, and visualize it accordingly?

(5) Identify the LED 0, 1, and 2 using color on your micaz node.

## References:

[1] TinyOS Tutorial Lesson 5 Sensing: http://docs.tinyos.net/tinywiki/index.php/Sensing