Wireless Embedded Systems and Networking - Labs Based on the AIIT Lecture

Jaein Jeong
David E. Culler

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-14
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-14.html

February 13, 2008

Wireless Embedded Systems and Networking - Labs Based on the AIIT Lecture

Jaein Jeong and David Culler

EECS Computer Science Division
University of California, Berkeley, CA 94720
{jaein,culler}@eecs.berkeley.edu

August 10, 2007

Contents

1 Experience with IP-based Wireless Sensor Networks **9**

1.1 First Table-top IP/WSN . . . 9

1.1.1 Forming a group . . . 9

1.1.2 Checking your inventory . . . 10

1.1.3 Setting up the IP address of the gateway server . . . 10

1.1.4 Building a table-top network . . . 11

1.1.5 Do some table-top sensing . . . 12

1.1.6 Try out the web services . . . 13

1.1.7 Discussion . . . 14

1.2 Build a self-organized mesh over a physical extent . . . 14

1.2.1 Deploying sensor nodes . . . 14

1.2.2 Sketch . . . 14

1.2.3 Check all the nodes . . . 15

1.2.4 Connectivity . . . 16

1.2.5 Site Survey . . . 16

1.2.6 Connectivity and mesh routing . . . 17

1.2.7 Reliability . . . 17

1.2.8 Mobility, obstruction and connectivity . . . 18

1.2.9 Energy Status . . . 18

1.2.10 Lab wrap up . . . 18

1.3 Add your own external sensors . . . 18

1.3.1 Magnetic reed sensor . . . 18

1.3.2 Connecting sensors to expansion port . . . 19

1.3.3 Configuring switch port . . . 19

1.3.4 Interrupt vs. Sampling . . . 20

1.3.5 Using RTD . . . 20

1.3.6 Concept of resistive sensors . . . 20

1.3.7 Attach RTD to ADC port . . . 21

1.3.8 Configure ADC port, resistor, reference . . . 21

1.3.9 Measurement using a sensor nodes . . . 22

1.3.10 Conversion . . . 22

1.3.11 A sample sensor board . . . 22

2 Making USNs ubiquitous - build WSN applications as Web Services **24**

2.1 Custom IP/USN application via Web Services . . . 24

2.1.1 Compare REST and SOAP . . . 24

2.1.2 Some Examples of REST . . . 24

2.1.3 List of REST API . . . 25

2.1.4 Using SOAP . . . 25
2.1.5 List of SOAP API . . . 26
2.1.6 SOAP Example in Java: Get the last temperature readings from sensors . . . 26
2.1.7 SOAP Example in PHP: Get the last temperature readings from sensors . . . 30
2.2 A Few Exercises for Writing SOAP Applications . . . 30
2.2.1 Exercise 1: Get the names of nodes . . . 30
2.2.2 Exercise 2: Get the locations of the nodes . . . 32
2.2.3 Exercise 3: Get all data in a recent time window (1 day) . . . 34
2.2.4 Exercise 4: Request a sample from all enabled sensors . . . 36
2.2.5 Exercise 5: Get the battery voltage of the nodes . . . 38
2.2.6 Exercise 6: Get statistics on node traffic and network reliability . . . 40
2.3 Writing PHP SOAP applications for Web Service . . . 42
2.3.1 Installation Steps for PHP Web Service . . . 42
2.3.2 PHP Web Service Example . . . 43
2.3.3 Commandline PHP versus Web-based PHP . . . 46
2.3.4 Exercise 2-7: PHP Web Service Exercise . . . 47
2.4 Custom analysis of the networking . . . 47
2.4.1 Before the site survey . . . 47
2.4.2 How to do site survey using SOAP API . . . 47
2.4.3 Requesting site survey . . . 47
2.4.4 Getting the site survey results . . . 48
2.4.5 Run site survey for your network . . . 48

3 Using IP and 6LoWPAN Networking 49
3.1 Configuration for IP access . . . 49
3.1.1 Setting up the LoWPAN IP address . . . 49
3.1.2 Setting up the routing to nodes . . . 50
Exercise 3-1 . . . 50
3.1.3 Ping nodes . . . 50
Exercise 3-2 . . . 50
3.2 IP access using standard client programs . . . 50
3.2.1 IP services provided by a sensor node . . . 50
Exercise 3-3 . . . 52
Exercise 3-4 . . . 52

4 TinyOS 2.0 based embedded Applications 53
4.1 TinyOS Programming on Open Source Distribution . . . 53
4.1.1 How to get TinyOS open source distribution . . . 53
4.1.2 Directions for the rest of chapter . . . 53
4.2 TinyOS 2.0 Programming Projects . . . 54
4.2.1 Project 1: Push and Toggle . . . 54
4.2.2 Project 2: Blink and Count . . . 56
4.2.3 Project 3: PrintSerial . . . 58
4.2.4 Project 4: Single and Dual . . . 61
4.2.5 Project 5: Raw and Smooth . . . 66
4.2.6 Project 6: Counts and Readings . . . 70
4.2.7 Project 7: Request and RequestSample . . . 73
4.2.8 Project 8: CountToRadio and RadioToCount . . . 80

A SOAP Application Solution **82**
A.1 SOAP Java Application 82
A.2 SOAP PHP Application 88
A.3 PHP Web Service 91

B Custom IP Application Examples **92**

C Solution for TinyOS 2.0 Programming Projects Based on Open Source Distribution **96**
C.1 Project 1: Push and Toggle 96
C.2 Project 2: Blink and Count 97
C.3 Project 3: PrintSerial 99
C.4 Project 4: Single and Dual 100
C.5 Project 5: Raw and Smooth 103
C.6 Project 6: Counts and Readings 105
C.7 Project 7: Request and RequestSample 107
C.8 Project 8: CountToRadio and RadioToCount 109

List of Tables

1.1 Study the resistance of the RTD in ice water versus room temperature versus tea 20
1.2 Converting raw reading to temperature 22

2.1 Commandline PHP versus Web-based PHP 46

4.1 Tinyos Projects to be covered in this chapter 54

List of Figures

1.1 Sensor network kit to be distributed to each group . . . 10
1.2 Setting up the 802.15.4 radio channel for each group . . . 11
1.3 Reprogramming the bridge node . . . 11
1.4 Program motes . . . 11
1.5 Identify a node . . . 12
1.6 Enable Sensors . . . 12
1.7 Temperature and humidity sensor . . . 12
1.8 Sensor Graph . . . 13
1.9 Setting the heart beat rate . . . 13
1.10 Try out REST examples . . . 14
1.11 Rendering of the floor plan . . . 15
1.12 Placing Nodes . . . 15
1.13 Check Nodes . . . 16
1.14 Connectivity . . . 16
1.15 Site Survey . . . 17
1.16 Reliability . . . 17
1.17 Energy Status . . . 18
1.18 Magnetic Reed Sensor . . . 19
1.19 Connectint sensors to expansion port . . . 19
1.20 Configuring switch port: (a) enable device, (b) name the states . . . 19
1.21 Set the alarm on humidity . . . 20
1.22 Resistive Sensor . . . 21
1.23 Attach RTD to ADC port . . . 21
1.24 Configure ADC port, resistor, reference . . . 22
1.25 A sample sensor board . . . 23

2.1 REST API . . . 25
2.2 SOAP API . . . 27
2.3 REST reply for getting last temperature readings from sensors . . . 27
2.4 Java SOAP example: Get.java . . . 28
2.5 Finding the name of SOAP API method . . . 29
2.6 Finding the signature of SOAP API method . . . 29
2.7 Java get() function that gets the last temperature readings from sensors . . . 29
2.8 The execution result of java application GetLastTemperature . . . 29
2.9 PHP get() function that gets the last temperature readings from sensors . . . 30
2.10 The execution result of the PHP application get_last_temperature.php . . . 30
2.11 The execution result of the exercise 2-1 . . . 31
2.12 Skeleton code for Ex2_1.java . . . 31
2.13 Skeleton code for ex2_1.php . . . 32

2.14 The execution result of the exercise 2-2 . . . 32
2.15 Skeleton code for Ex2_2.java . . . 33
2.16 Skeleton code for ex2_2.php . . . 34
2.17 The execution result of the exercise 2-3 . . . 34
2.18 Skeleton code for Ex2_3.java . . . 35
2.19 Skeleton code for ex2_3.php . . . 36
2.20 The execution result of the exercise 2-4 . . . 36
2.21 Skeleton code for Ex2_4.java . . . 37
2.22 Skeleton code for ex2_4.php . . . 38
2.23 The execution result of the exercise 2-5 . . . 38
2.24 Skeleton code for Ex2_5.java . . . 39
2.25 Skeleton code for ex2_5.php . . . 40
2.26 The execution result of the exercise 2-6 . . . 40
2.27 Skeleton code for Ex2_6.java . . . 41
2.28 Skeleton code for ex2_6.php . . . 42
2.29 Modification to httpd.conf to activate PHP in the Apache web server . . . 42
2.30 PHP Web Service Example . . . 43
2.31 Header file to be included for a PHP source file . . . 44
2.32 PHP source code for lab2.php . . . 45
2.33 PHP source code for get_last_temperature_web.php . . . 46
2.34 PHP Exercise . . . 47
2.35 Requesting Site Survey . . . 48
2.36 Site Survey Results . . . 48

3.1 Finding the IP address of a sensor node . . . 49
3.2 To ping a node . . . 50
3.3 Port 7 (echo) . . . 51
3.4 Port 10 (leds) . . . 51
3.5 Port 11 (sysstat) . . . 51
3.6 Port 15 (netstat) . . . 51
3.7 Port 30 (raw sensor reading) . . . 51
3.8 telnet to port 30 . . . 52
3.9 An example of custom IP application written in Java . . . 52

4.1 Makefile: Makefile for Push . . . 55
4.2 PushAppC.nc: Configuration module for Push . . . 55
4.3 PushC.nc: Implementation module for Push . . . 56
4.4 Makefile: Makefile for Blink . . . 56
4.5 BlinkApp.nc: Configuration module for Blink . . . 57
4.6 BlinkC.nc: Implementation module for Blink . . . 57
4.7 Makefile: Makefile for SerialPrint . . . 58
4.8 run.sh: Shell script to run the Java client program . . . 58
4.9 PrintSerial.h: Header file to define the message structure . . . 58
4.10 PrintSerial.java: Java client program for PrintSerial . . . 59
4.11 PrintSerialAppC.nc: Configuration module for PrintSerial . . . 60
4.12 PrintSerialC.nc: Implementation module for PrintSerial . . . 61
4.13 Makefile: Makefile for Single . . . 62
4.14 run.sh: Shell script to run Java client . . . 62
4.15 PrintReading.java: java client program . . . 63

4.16 PrintReading.h: Header file for Single . . . 64
4.17 SingleAppC.nc: Configuration module for Single . . . 64
4.18 SingleC.nc: Implementation module for Single . . . 65
4.19 TemperatureC.nc: Wrapper for Telosb internal temperature sensor . . . 66
4.20 DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor . . . 66
4.21 Makefile: Makefile for Raw . . . 67
4.22 run.sh: Shell script to run Java client . . . 67
4.23 PrintReadingArr.java: Java client for Raw . . . 67
4.24 PrintReadingArr.h: Header file for Raw . . . 68
4.25 RawAppC.nc: Configuration module for Raw . . . 68
4.26 RawC.nc: Implementation module for Raw . . . 69
4.27 Makefile: Makefile for Counts . . . 70
4.28 run.sh: Shell script to run Java client . . . 70
4.29 PrintCounterReading.java: java Client for Counts . . . 71
4.30 PrintCounterReading.h: Header file for Counts . . . 71
4.31 CountsAppC.nc: Configuration module for Counts . . . 72
4.32 CountsC.nc: Implementation module for Counts . . . 73
4.33 Makefile: Makefile for Request . . . 74
4.34 run.sh: Shell script to run the Java client . . . 74
4.35 RequestReading.java: Java client for Request . . . 75
4.36 ClientWindow.java: Java GUI for Request . . . 76
4.37 RequestReading.h: Header file for Request . . . 77
4.38 TemperatureC.nc: Wrapper for Telosb internal temperature sensor . . . 77
4.39 DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor . . . 77
4.40 RequestAppC.nc: Configuration module for Request . . . 78
4.41 RequestC.nc: Implementation module for Request . . . 79
4.42 Makefile: Makefile for CountToRadio . . . 80
4.43 Counter.h: Header file for CountToRadio . . . 80
4.44 CountToRadioAppC.nc: Configuration module for CountToRadio . . . 80
4.45 CountToRadioC.nc: Implementation module for CountToRadio . . . 81

A.1 Solution code for Ex2_1.java . . . 82
A.2 Solution code for Ex2_2.java . . . 83
A.3 Solution code for Ex2_3.java . . . 84
A.4 Solution code for Ex2_4.java . . . 85
A.5 Solution code for Ex2_5.java . . . 86
A.6 Solution code for Ex2_6.java . . . 87
A.7 Solution code for ex2_1.php . . . 88
A.8 Solution code for ex2_2.php . . . 88
A.9 Solution code for ex2_3.php . . . 89
A.10 Solution code for ex2_4.php . . . 89
A.11 Solution code for ex2_5.php . . . 90
A.12 Solution code for ex2_6.php . . . 90
A.13 Solution for get_temperature_celsius.php . . . 91

B.1 Custom IP application: EchoClient . . . 92
B.2 Custom IP application: LedsClient . . . 93
B.3 Custom IP application: SysstatClient . . . 94
B.4 Custom IP application: NetstatClient . . . 94

B.5 Custom IP application: SensorReadClient 95

C.1 Makefile: Makefile for Toggle 96
C.2 ToggleAppC.nc: Configuration module for Toggle 96
C.3 ToggleC.nc: Implementation module for Toggle 97
C.4 Makefile: Makefile for Count 97
C.5 CountApp.nc: Configuration module for Count 98
C.6 CountC.nc: Implementation module for Count 99
C.7 Makefile: Makefile for Dual 100
C.8 TemperatureC.nc: Wrapper for Telosb internal temperature sensor 100
C.9 DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor 100
C.10 DualAppC.nc: Configuration module for Dual 101
C.11 DualC.nc: Implementation module for Dual 102
C.12 Makefile: Makefile for Smooth 103
C.13 SmoothAppC.nc: Configuration module for Smooth 103
C.14 SmoothC.nc: Implementation module for Smooth 104
C.15 Makefile: Makefile for Readings 105
C.16 ReadingsAppC.nc: Configuration module for Readings 105
C.17 ReadingsC.nc: Implementation module for Readings 106
C.18 Makefile: Makefile for RequestSample 107
C.19 TemperatureC.nc: Wrapper for Telosb internal temperature sensor 107
C.20 DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor 107
C.21 RequestSampleAppC.nc: Configuration module for RequestSample 107
C.22 RequestSampleC.nc: Implementation module for RequestSample 108
C.23 Makefile: Makefile for RadioToCount 109
C.24 RadioToCountAppC.nc: Configuration module for RadioToCount 109
C.25 RadioToCountC.nc: Implementation module for RadioToCount 110

Chapter 1

Experience with IP-based Wireless Sensor Networks

In this chapter, we demonstrate the capability of wireless sensor networks (WSNs) using Primer Pack from Arch Rock Corporation. Unlike traditional bottom-up approaches, using a cutting-edge WSN kit hides low-level details allows first-time users of WSNs to build their own low-power multi-hop WSNs very easily.

We start this chapter by building a table-top network, and this allows the audience to understand the basic operations of WSNs. Next, we extend this table-top network to a multi-hop mesh network by deploying the sensor nodes in a larger space, and this shows how a multi-hop mesh network behaves over time.

1.1 First Table-top IP/WSN

1.1.1 Forming a group

The first thing to do in this lab is to form a group and interview other members in the group. For each group, submit the name of group and the list of the names of the members. Let each person interview another member of the group focusing on the followings:

- Name
- Affiliation
- Reasons to attend this course. What do they want to get out of it?
- Background (system, networking, analog circuits, etc.)
- What programming languages do you like? Use? Know? e.g. C, Java, C++, PHP,
- What aspects of networking are you familiar with?
- Operating systems?
- Hardware design?
- Mathematical analysis?

1.1.2 Checking your inventory

Each group will have a dedicated wireless sensor network with a gateway server and a collection of Arch Rock Primer Pack nodes, K-motes as show in Figure 1.1.

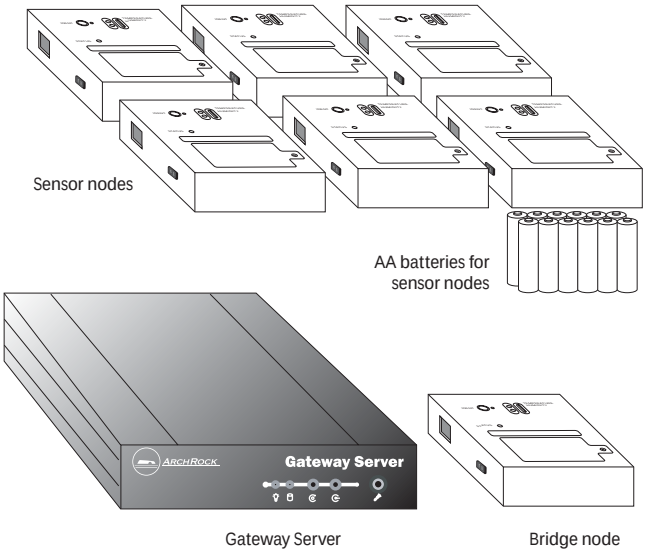

Figure 1.1: Sensor network kit to be distributed to each group

1.1.3 Setting up the IP address of the gateway server

Before we build a table-top network, a gateway server should be assigned an IP address either a DHCP address or a static address. When the gateway server is assigned a DHCP address, you need to find its IP address to access the gateway server. There are two ways to find a DHCP address.

- When you have an administrator access to the DHCP server: A DHCP server usually keeps a list of devices attached to the network. By comparing the Ethernet address of the device with the list of devices, you can find the IP address of the gateway server.

- Otherwise: A scanning program needs to be used to find a list of devices attached to the network. For Windows platforms, you can use "Bonjour" (`developer.apple.com/networking/bonjour/download/`). Once installed, Bonjour shows a list of devices in the same LAN in the Explorer Bar of the Microsoft Internet Explorer web browser.
  For Linux platforms, you can use "nmap". Suppose your loal network is 192.168.7.0/24, then you can run the following command:
  `nmap -st -p 80 '192.168.7.*'`
  The gateway server should appear with the identifier "King Young Technology Co."

When there is no DHCP server, the gateway server IP address is set to 192.168.69.3 by default. If you want to set a specific static IP address, you need to login to the gateway on the console mode. Before you go to the console mode, plug a monitor, a keyboard and a mouse to the gateway server before you turn on the gateway server. Once the gateway boots with the console, you can login as root and its password.

- **Step 1:** Type "setup" in the command line to set up the IP address.
  - In the screen, select "Configure the IP address."

- Type the static IP address, default gateway and DNS address that you are given from your local network administrator.
- Type 255.255.255.0 as subnet mask.

- **Step 2:** After finishing the setup command, type "/etc/init.d/networking restart" in the command line to restart the Ethernet link of the gateway server with the new IP address.

1.1.4 Building a table-top network

Once an IP address is given to a gateway server, a user can configure the gateway server through a web brower. Suppose the IP address of the gateway server is "192.168.0.2", type "http://192.168.0.2" in the address file of a web brower. The default ID and password for an administrator is "admin" and "XpressK".

Using the deployment page, pick a name for the deployment and select the 802.15.4 channel assigned to your group (from 11 to 26). Also select a security passphrase (Figure 1.2).

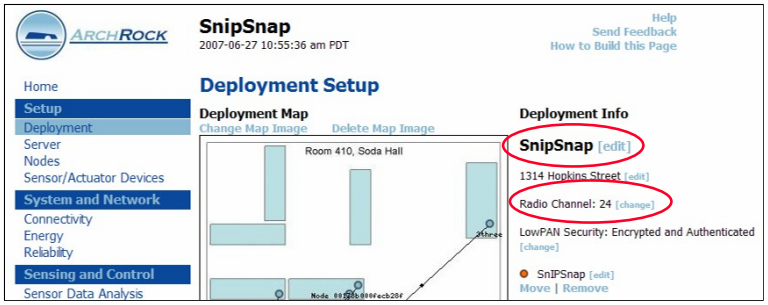

Figure 1.2: Setting up the 802.15.4 radio channel for each group

Place the server on the (empty) map. Program the bridge node by plugging the bridge node to a USB port of the gateway server and clicking the "Reprogram Bridge" link in the "Server" page (Figure 1.3).

Figure 1.3: Reprogramming the bridge node

Program each of the motes by plugging a mote in the USB port and clicking "Reprogram Node" link in the "Nodes" page (Figure 1.4). Once all the motes are programmed, plug the bridge node again. After that, discover the nodes by clicking "Discover Nodes" link in the "Nodes" page. Once all the nodes are discovered, register the nodes by clicking "Register All Unregistered Nodes" link.

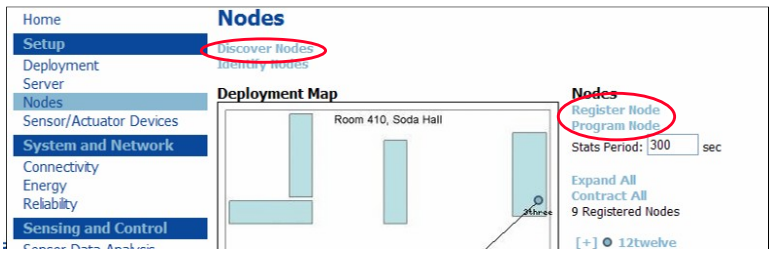

Figure 1.4: Program motes

Once all the nodes are registered, place each node on the map. Ping each node and see it flash the blue LED. Push ident button on the node and see it flash on the screen. Identify which node is which by EUID64. See that they have a short address, an IPv6 address, and an IPv4 address (Figure 1.5).

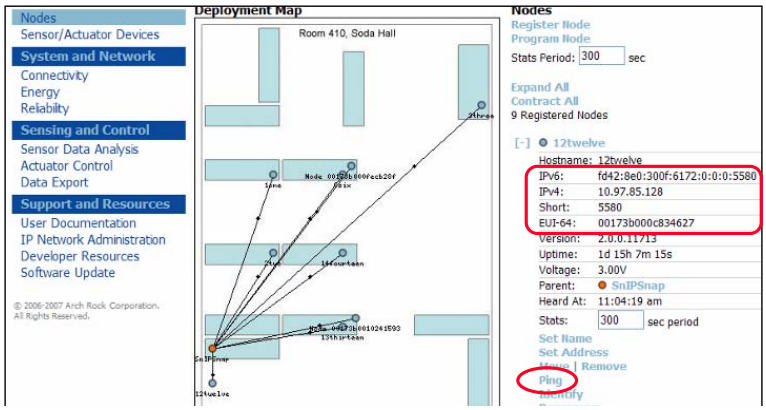

Figure 1.5: Identify a node

1.1.5 Do some table-top sensing

Go to the "Sensor and Actuators" page. Enable all internal sensors. Set the sample rate to 4 secs (Figure 1.6). Go to the "Sensor Data" page. Set the refresh. Cover the nodes. Blow on them. Put them in warm places (Figure 1.7).

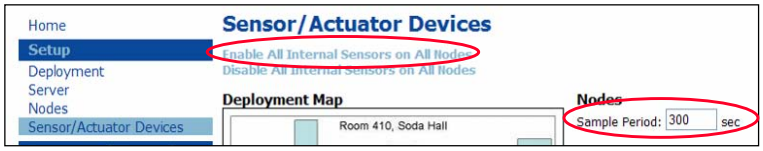

Figure 1.6: Enable Sensors

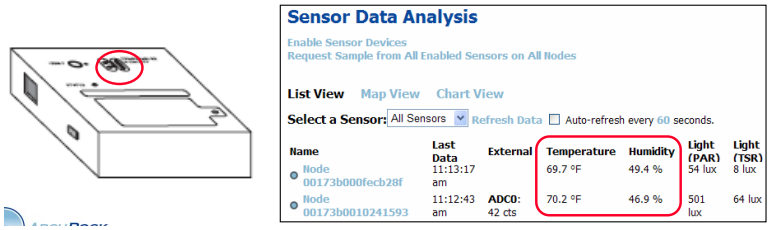

Figure 1.7: Temperature and humidity sensor

Click on the name of a node and open up the node web page view. See the graphs of the data over time (Figure 1.8).

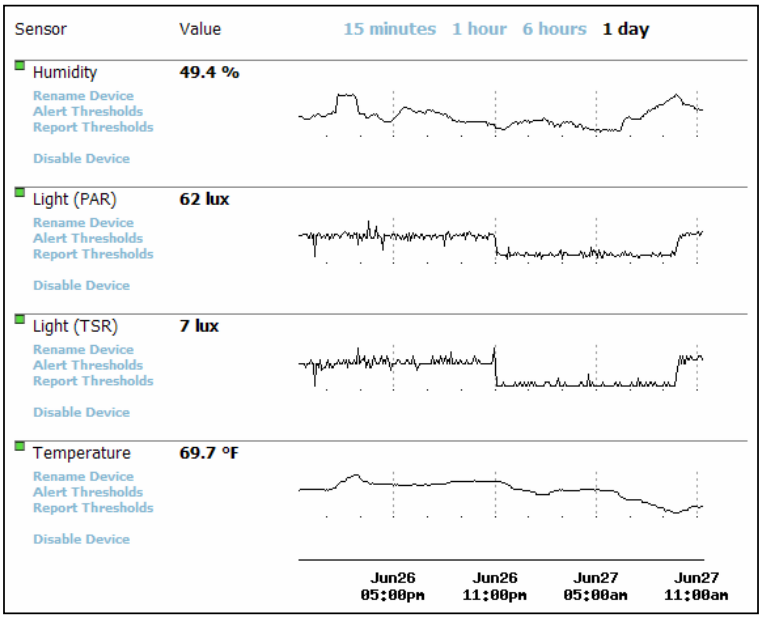

Figure 1.8: Sensor Graph

Pick a node. Set thresholds. Adjust the sample rate. Configure it to send email on alarm.

Adjust the heartbeat to 30 secs (Figure 1.9). Turn off a node. See it go red. Try to ping it. Look at the data.

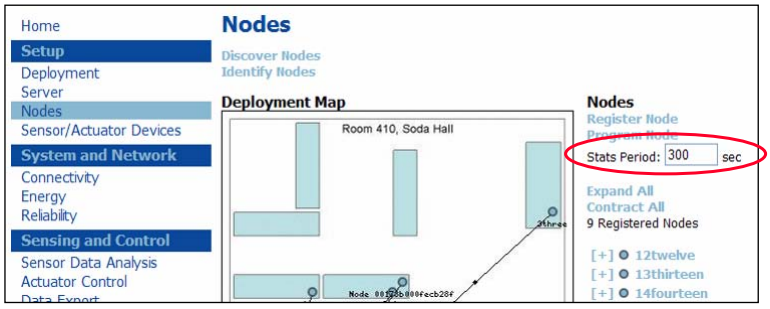

Figure 1.9: Setting the heart beat rate

1.1.6 Try out the web services

Notice that each page has "How to build this page" in the upper right corner. Click on the one for the network home page. Try out the REST examples.

Try out the documentation links in the lower left corner: Find the Developer Resources -¿ Arch Rock Server API References -¿ Application Documentation. Get the EUID and use the REST URL to retrieve various data on demand. Try out other attributes (Figure 1.10).

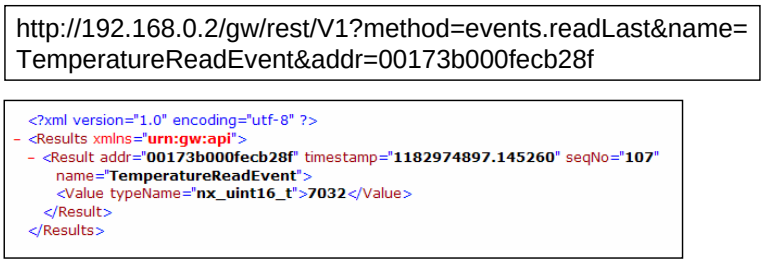

Figure 1.10: Try out REST examples

1.1.7 Discussion

You have become acquainted with your first IP-based USN. Discuss with the group what you have seen, how it compares with your expectations, how might this all be working. How might you deploy the network around the lab?

You are running your networks at a sample rate that is 10-100x of what it typical for environmental monitoring. However, all instrumentation has its limits. Can you cause an environmental change too short in duration for your sensor network to observe? Too small in magnitude? What are its fundamental limits?

1.2 Build a self-organized mesh over a physical extent

1.2.1 Deploying sensor nodes

Discuss as a group where to place your nodes in the lab and in its vicinity. Identify interesting places to sense. In addition to the environmental sensors, you can imagine adding open/close sensors, tilt, or other inputs.

Draw a rough sketch of the floor plan of the space that you will cover and indicate where you will place the nodes. Use all but one of the nodes, so we can set aside for later.

Discuss what you will be able to observe through the sensors. Discuss what you think the wirelss connectivity will be among the nodes. Discuss how you think information will get routed to the server.

1.2.2 Sketch

Upload your sketch, either by scanning it in or by rendering it in PowerPoint or your favorite drawing program to create a jpeg (Figure 1.11).

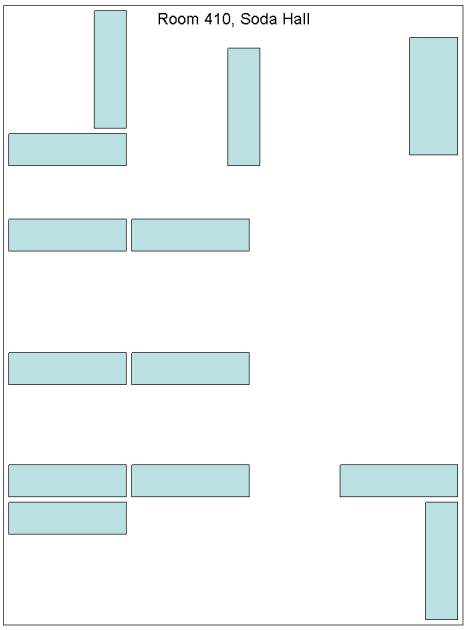

Figure 1.11: Rendering of the floor plan

Select up to 5 nodes to place at the intended spots in your lab space. For each one, name it in your network. Place it at the point of interest. Place its representative at the corresponding point on your map (Figure 1.12). Continue to use an accelerated rate. Maybe 5 sec is enough.

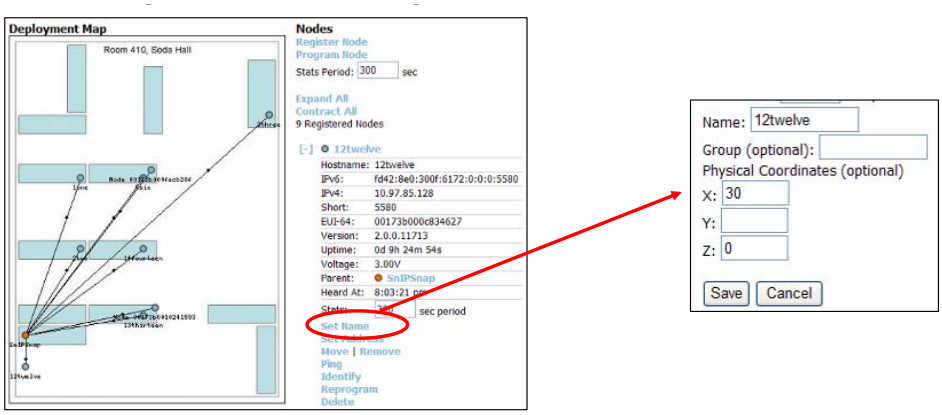

Figure 1.12: Placing Nodes

1.2.3 Check all the nodes

Observe whether all the nodes are part of the mesh (Figure 1.13). Are any missing?

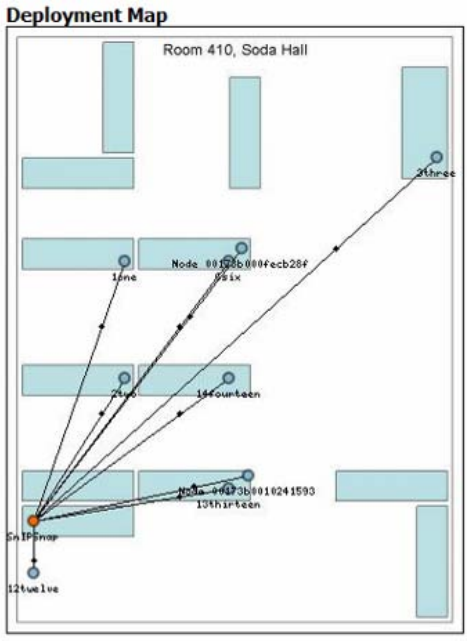

Figure 1.13: Check Nodes

1.2.4 Connectivity

Study the connectivity by going to the connectivity page. Study the map view (Figure 1.14(a)). What is routing through which? Go to the list view (Figure 1.14(b)). See the RSSI and link quality.

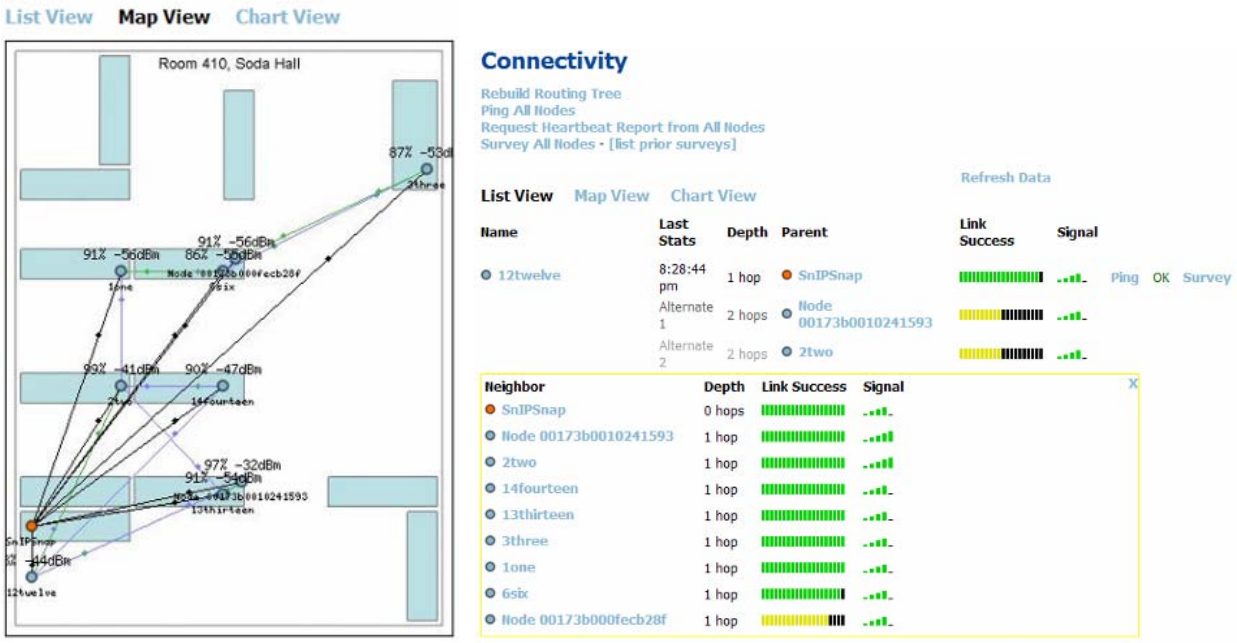

Figure 1.14: Connectivity

1.2.5 Site Survey

Run a site survey from each of the nodes.

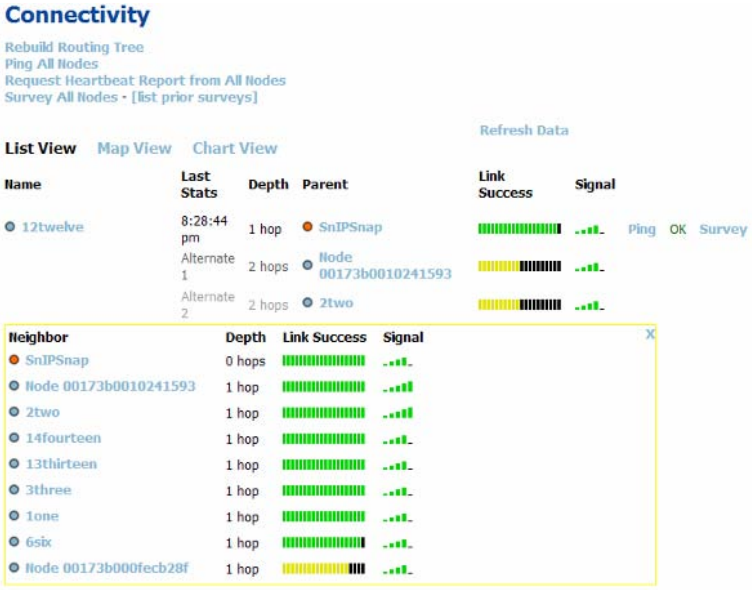

Figure 1.15: Site Survey

1.2.6 Connectivity and mesh routing

Discuss how the connectivity compares to your expectations. Discuss how the mesh routing to the server compares to your expectations. If the network is too space, add a node to fill it in. If the network is too shallow, extend with a node farther out.

1.2.7 Reliability

Use the reliability page. Reset the reliability statistics. Let it go for a bit.

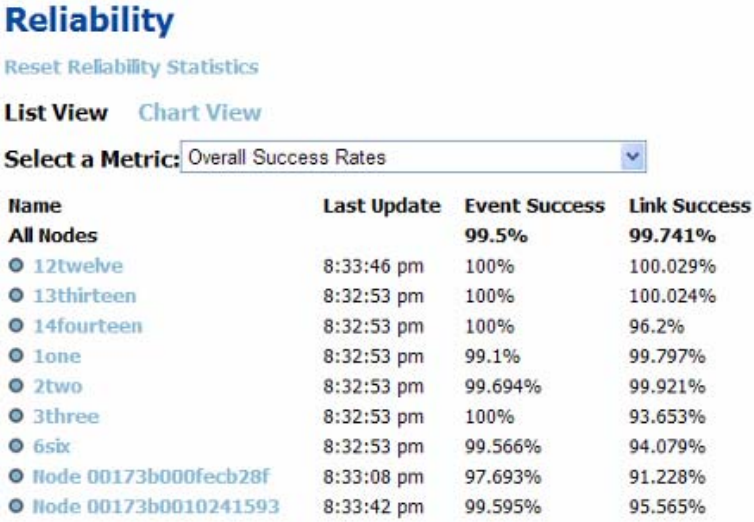

Figure 1.16: Reliability

1.2.8 Mobility, obstruction and connectivity

Move nodes around, move them on the map. See how the connectivity remains. Try to obstruct it and see how the routing changes.

1.2.9 Energy Status

Go to the energy page and see the duty cycle (Figure 1.17). Adjust the sample rate to something more typical, say 300s.

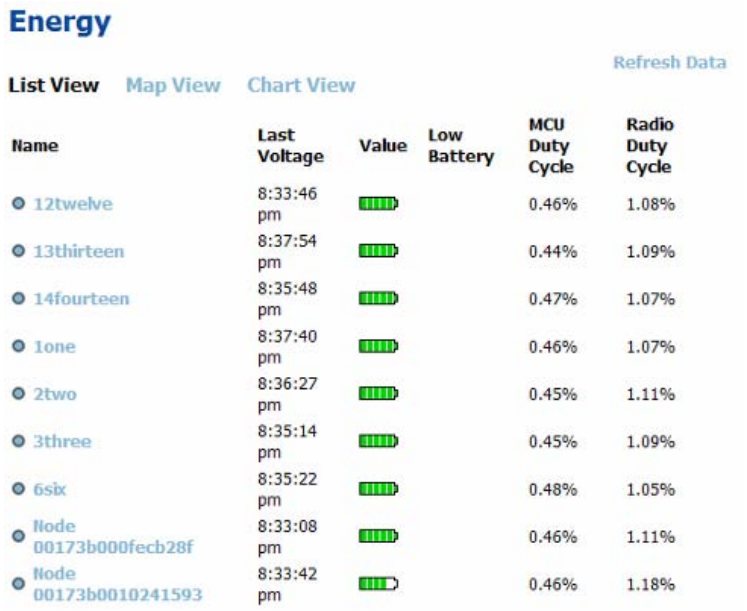

Figure 1.17: Energy Status

1.2.10 Lab wrap up

Place nodes where they can run over night. Verify that the network is robust. Turn on data warehouse. Reset stats. Let them run till tomorrow.

1.3 Add your own external sensors

1.3.1 Magnetic reed sensor

Study the resistance of the magnetic reed sensor open and closed (Figure 1.18). Discuss pull up, expected voltage and current.

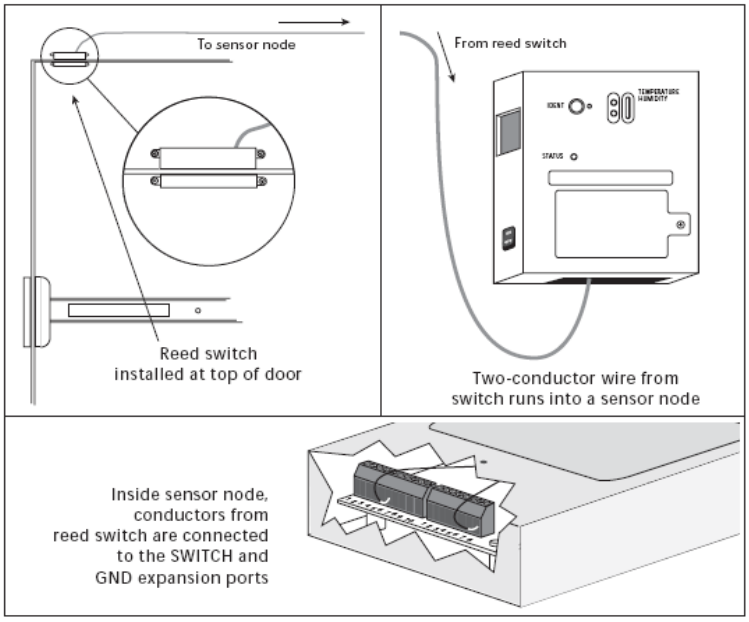

Figure 1.18: Magnetic Reed Sensor

1.3.2 Connecting sensors to expansion port

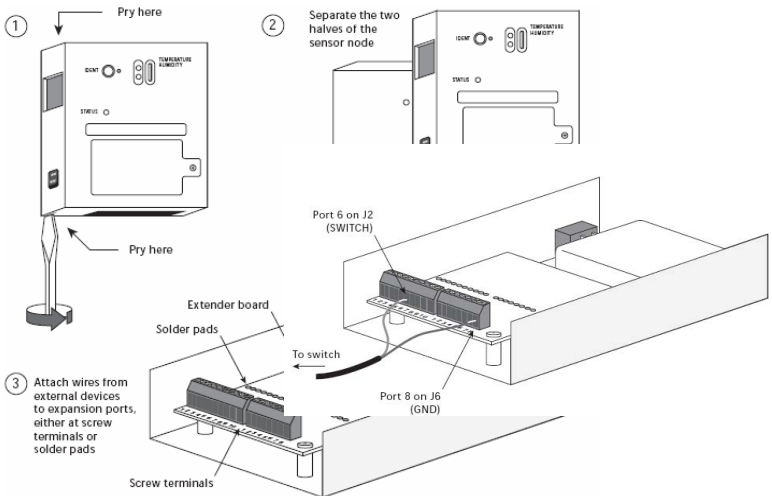

Figure 1.19: Connectint sensors to expansion port

1.3.3 Configuring switch port

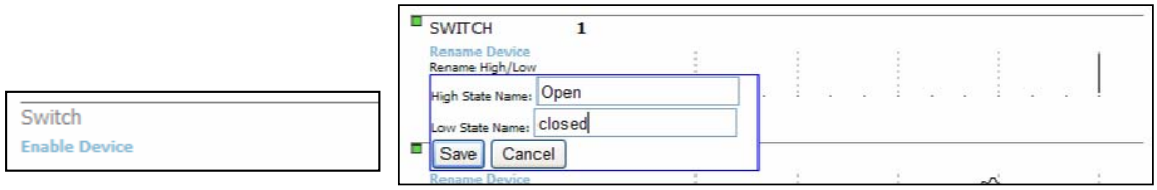

Figure 1.20: Configuring switch port: (a) enable device, (b) name the states

1.3.4 Interrupt vs. Sampling

Discuss interrupt versus sampling. Set the alarm on humidity (Figure 1.21).

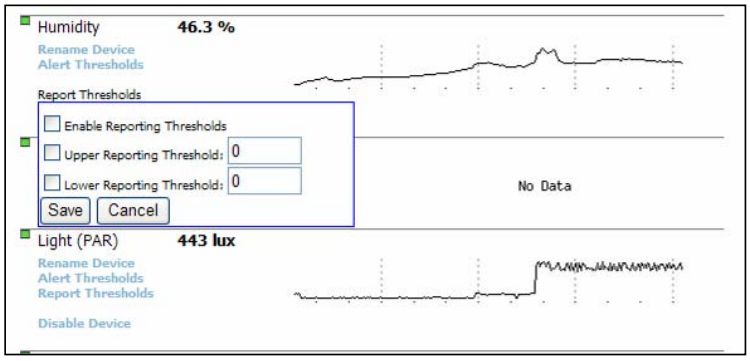

Figure 1.21: Set the alarm on humidity

1.3.5 Using RTD

Study the resistance of the RTD in ice water versus room temperature versus tea (Table 1.1).

Table 1.1: Study the resistance of the RTD in ice water versus room temperature versus tea

	Ice Water	**Room Temperature**
Resistance of RTD		
Temperature		

1.3.6 Concept of resistive sensors

Discuss voltage divider, reference voltage. Mapping readings to engineering units.

$$V_{ADC0} = \frac{R_{comp}}{R_{comp} + R_{th}} VCC \qquad V_{VPU0} = VCC$$
$$V_{ADC0} = 0 \qquad V_{VPU0} = 0$$

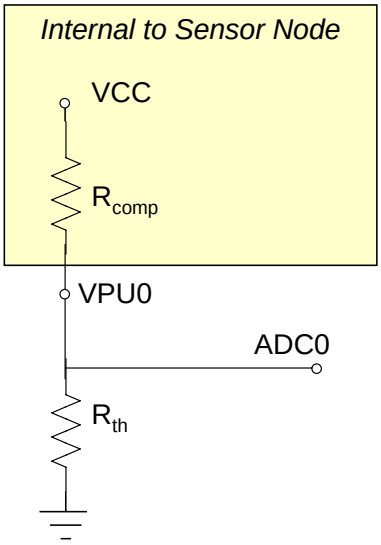

Figure 1.22: Resistive Sensor

In Figure 1.22, what value R_{comp} should be set to?
For a Primer Pack node, V_{ADC0} is between 0 and 4095 (12-bits).

1.3.7 Attach RTD to ADC port

Connect one end to ADCx, the other to GND (Figure 1.23). Try with your own node.

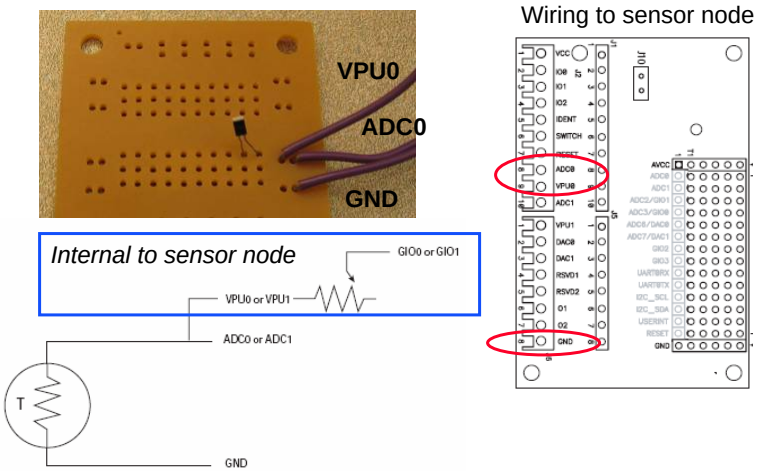

Figure 1.23: Attach RTD to ADC port

1.3.8 Configure ADC port, resistor, reference

For each ADC port, first, enable the port to use. And then configure each port (Figure 1.24).

Table 1.2: Converting raw reading to temperature

	Raw Reading	Temperature
Point 1		
Point 2		
Point 3		
Point 4		
Point 5		

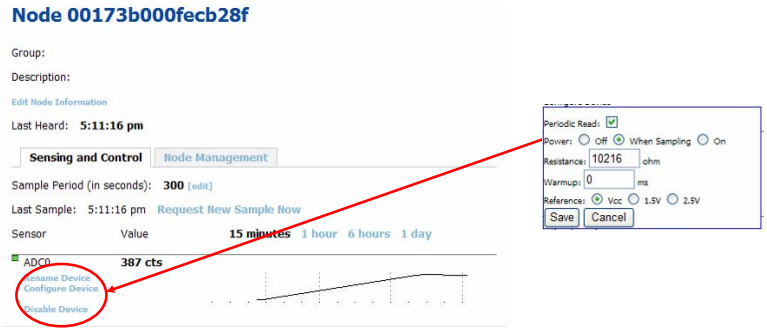

Figure 1.24: Configure ADC port, resistor, reference

1.3.9 Measurement using a sensor nodes

Conduct the same measurements using a sensor node. Measure temperature using the RTD connected to a node. Try both in ice water versus in room temperature. Compare results to expectations.

1.3.10 Conversion

Pull data into excel using the data export facility and do the conversions as in Table 1.2.

1.3.11 A sample sensor board

A sample sensor board will be used to demonstrate the concept of different types of sensors (Figure 1.25).

- Resistive sensor: photo resistor
- Voltage sensor: voltage divider with variable resistor
- Actuator: LED

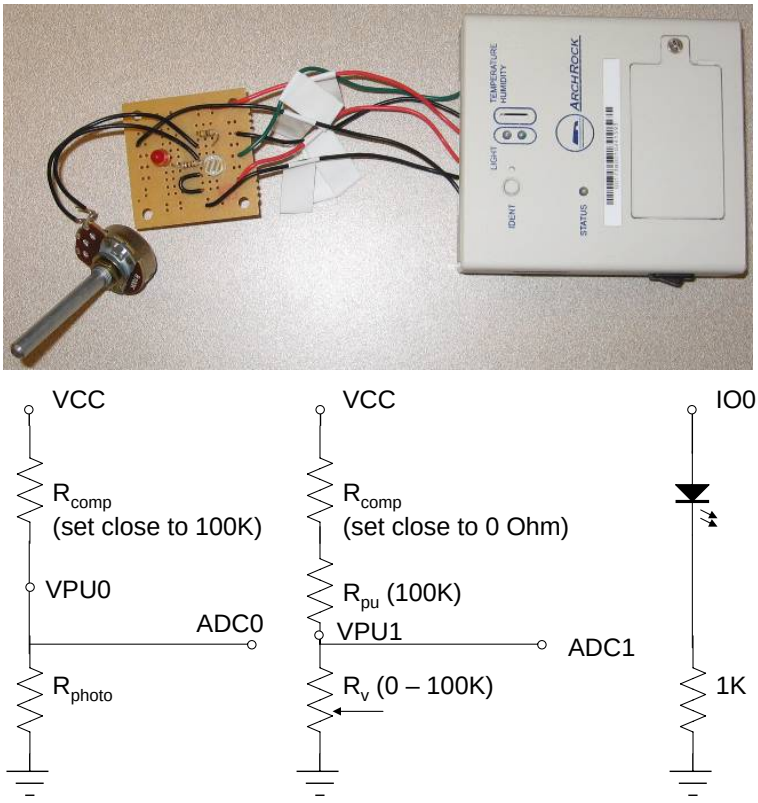

Figure 1.25: A sample sensor board

Chapter 2

Making USNs ubiquitous - build WSN applications as Web Services

2.1 Custom IP/USN application via Web Services

In the previous chapter, we introduced wireless sensor networks (WSNs) in the perspective of users by demonstrating how to build a network and how to get sensor readings. In this chapter, we explore the capabilities of WSNs as a programming platform. We will demonstrate how to write a web service for WSNs based on REST (Representational State Transfer) and SOAP (Simple Object Access Protocol) application programming interfaces that are provided for Arch Rock Primer Pack.

2.1.1 Compare REST and SOAP

Arch Rock Primer Pack provides two kinds of application programming interface (API): REST (Representational State Transfer) and SOAP (Simple Object Access Protocol).

REST API allows a client to access the gateway server through HTTP GET or POST requests. REST returns the response from the gateway server to the client in XML, and this XML response can be directly viewed or parsed further by the client. One advantage is REST is that it is easy to start with because it just needs a web browser. A user types a REST call in the web browser address field and the response is displayed in the web browser screen.

While REST API interacts with the gateway server using an HTTP request and reply, SOAP API allows a client to access the gateway using various kinds of host programming languages such as Java, PHP and C/C++ through SOAP library. The SOAP library reads the XML response from the gateway and parses it into a list of objects that can be understood by the host programming language. SOAP requires more time to start with, but it is easier to integrate with programming language.

2.1.2 Some Examples of REST

Assuming the IP address of the gateway server is 192.168.0.2

- Get the last temperature reading from the sensors:
  `http://192.168.0.2/gw/rest/V1/?method=events.readLast?name=TemperatureReadEvent`
- Read the sensor sample period:
  `http://192.168.0.2/gw/rest/V1/?method=attributes.get&name=SamplePeriod`
- Change the sensor sample period:
  `http://192.168.0.2/gw/rest/V1/?method=attributes.set&name=SamplePeriod&value=300000`

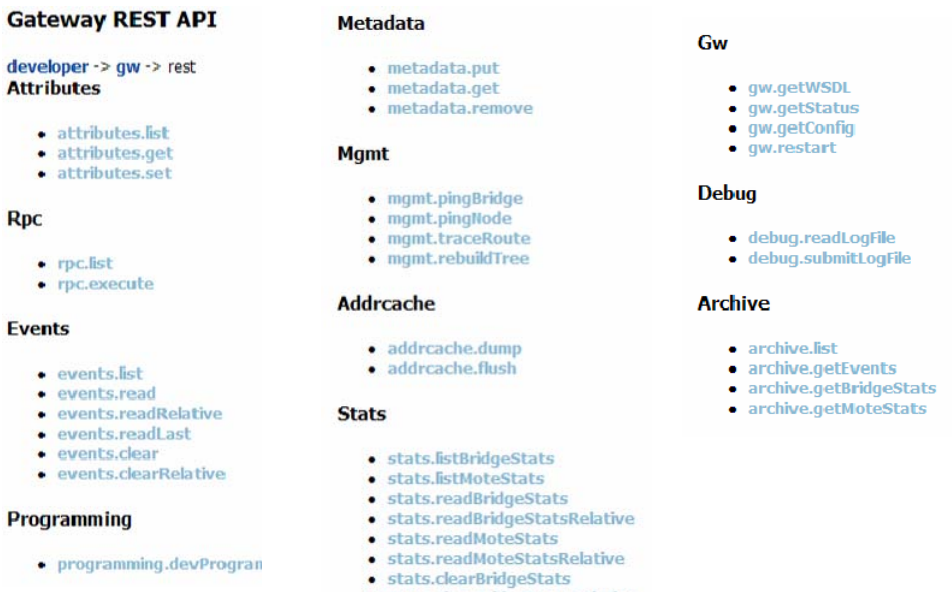

Figure 2.1: REST API

- Ping a node:
  `http://192.168.0.2/gw/rest/V1/?method=mgmt.pingNode&addr=ffffffffffffffff`

2.1.3 List of REST API

The list of REST API can be seen by seeing the following link: `http://192.168.0.2/developer/gw/rest/`

2.1.4 Using SOAP

SOAP can be used various programming languages such as Java, PHP, C/C++, C#, perl and python. While the details of installation procedure and usage can be different depending on programming languages, the general installation procedure and usage are as follows:

- **Step 1:** Install Development Kit such as JDK, PHP development kit and C/C++ compiler.
- **Step 2:** Install SOAP library and client stubs.
- **Step 3:** Access SOAP in host programming language.

The SOAP installation procedure and usage for Java are as follows:

- **Step 1:** Install JDK
- **Step 2a:** Install SOAP library.
  SOAP library consists of Apache Axis, commons-httpclient-3.0-rc3.jar and commons-codec-1.3.jar modules. After downloading these modules in local hard drive, add the path to these modules in CLASSPATH environment variable.
- **Step 2b:** Creat SOAP client stubs.
  Create SOAP client stubs by typing the following command:
  `java org.apache.axis.wsdl.WSDL2Java "http://192.168.0.2/gw/rest/V1/?method=gw.getWSDL"`
  This SOAP client stub marshall and unmarshall SOAP requests and replies into the syntax a Java program can understand.

- **Step 3a:** Acquiring gateway port.
  A Java application can access a gateway after acquiring the port. The following two lines of code do this job:
  ```
  GWServiceLocator locator = new GWServiceLocator();
  GWPort proxy = locator.getGWPort(portURL);
  ```

- **Step 3b:** Sending a request to gateway.
  The following lines of Java code show how to send a request once you have the access to the gateway port.
  ```
  GW__PingNode_Result[] results;
  results = proxy.mgmtPingNode(addr, 1, TIMEOUT, "*");
  System.out.println(results[i].getLongAddr());
  ```

The SOAP installation procedure and usage for PHP are as follows:

- **Step 1, 2:** Install PHP development kit.
  We used PHP 5.2.3 on a Windows machine. During the PHP installation, choose SOAP as an external component and do not select other external components, which can be due to a bug in the distribution of PHP.

- **Step 3a:** Acquire gateway port.
  While we created the client stub before we compile the code for Java, PHP doesn't require precreating client stub because it creates the client stub at run time. The following lines of PHP code acquire the gateway port:
  ```
  $url = http:// .  $gateway .  "/gw/rest/V1/?method=gw.getWSDL";
  $proxy = new SoapClient($url);
  ```

- **Step 3b:** Send a request to gateway.
  The following line of PHP code shows how to send a request once you have the access to the gateway port.
  ```
  $ret = $proxy->mgmtPingNode("ffffffffffffffff", 2, "*");
  ```

2.1.5 List of SOAP API

The list of SOAP API can be seen by seeing the following link: `http://192.168.0.2/developer/gw/soap/`

2.1.6 SOAP Example in Java: Get the last temperature readings from sensors

This example will demonstrate how to build a simple SOAP application. Suppose we want to get the last temperature readings from sensors. The corresponding REST call is as follows:
`http://192.168.0.2/gw/rest/V1/?method=events.readLast&name=TemperatureReadEvent`
The reply for this REST call is shown in Figure 2.3.

To make it easy to start, we provide a skeleton Java SOAP application called "Get.java" as in Figure 2.4. In order to implement a different function, you can change the line for method invocation:
`results = proxy.attributesGet(name, addr, TIMEOUT, "*");`

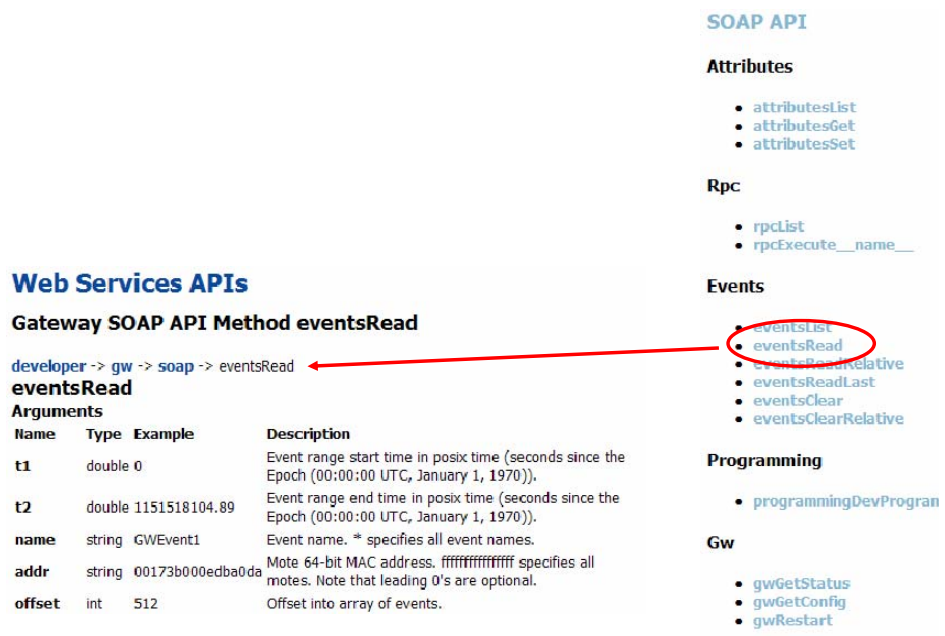

Figure 2.2: SOAP API

```
<?xml version="1.0" encoding="utf-8" ?>
- <Results xmlns="urn:gw:api">
  - <Result addr="00173b000fecb28f" timestamp="1182399635.628388"
      seqNo="1" name="TemperatureReadEvent">
      <Value typeName="nx_uint16_t">7203</Value>
    </Result>
  - <Result addr="00173b0010241593" timestamp="1182399635.636630"
      seqNo="116" name="TemperatureReadEvent">
      <Value typeName="nx_uint16_t">7236</Value>
    </Result>
</Results>
```

Figure 2.3: REST reply for getting last temperature readings from sensors

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class Get
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    Get(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        }
        catch (Exception e) {
            e.printStackTrace(); System.exit(2);
        }
    }
    public void get(String name, String addr)
    {
        GW__Attribute_Result[] results;
        GW__Attribute value;
        try {
            results = proxy.attributesGet(name, addr, TIMEOUT, "*");
            for (int i = 0; i < results.length; i++) {
                value = results[i].getValue();
                Method m = value.getClass().getMethod("get" + name, (Class[])null);
                Object o = m.invoke(value, (Object[])null);
                System.out.println("Mote " + results[i].getAddr() + ": " + name + " value is " + o);
            }
        }
        catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 2 && args.length != 3) {
            System.out.println("Usage: java Get gateway attrName [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName = args[1];
        String mote;
        if (args.length == 2) mote = "ffffffffffffffff";
        else mote = args[2];
        Get get = new Get(gateway);
        get.get(attrName, mote);
    }
}
```

Figure 2.4: Java SOAP example: Get.java

To find the name of the method you are going to implement, you can look up the SOAP API reference in `http://192.168.0.2/developer/gw/` . For example, Figure 2.5 shows that the corresponding SOAP API for the REST method `events.readLast` is `eventsReadLast`.

Once you find the name of SOAP API method to use, you need to find the signature for the method in order to match the type of parameters and the return value. Figure 2.6 shows that `eventsReadLast` method has a parameter list of `(java.lang.String, java.lang.String)` and a return value of `gw.GW__eventsReadLast_Result`. Then, you need to declare data structure needed for parameters and the return value.

Figure 2.5: Finding the name of SOAP API method

```
$ grep -n "eventsReadLast(" *java
GWPort.java:27:    public gw.GW__eventsReadLast_Result eventsReadLast(java.lang.String name,
java.lang.String addr) throws java.rmi.RemoteException;
```

Figure 2.6: Finding the signature of SOAP API method

Figure 2.7 shows the contents of Java `get()` function that gets the last temperature readings from sensors. Figure 2.8 shows the result when the java application `GetLastTemperature` is executed.

```
public void get(String name, String addr) {
    // 1. Intermediate Data Type Declaration
    GW__eventsReadLast_Result result;
    GW__Event_Result[] results;
    GW__Event value;
    try {
        // 2. Method Invocation
        result = proxy.eventsReadLast(name, addr);
        // 3. Reading Results
        results = result.getResults();
        for (int i = 0; i < results.length; i++) {
            value = results[i].getValue();
            System.out.println( "Mote " + results[i].getAddr() + " Time " + results[i].getTimestamp() +
                        " " + name + " " + value.getTemperatureReadEvent() );
        }
    }
    catch (Exception e) {
        e.printStackTrace(); System.exit(2);
    }
}
```

Figure 2.7: Java get() function that gets the last temperature readings from sensors

```
$ java GetLastTemperature 192.168.0.2
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
Mote 00173b000fecb28f Time 1.182555699518779E9 TemperatureReadEvent 7246
Mote 00173b0010241593 Time 1.182555699178323E9 TemperatureReadEvent 7297
```

Figure 2.8: The execution result of java application GetLastTemperature

2.1.7 SOAP Example in PHP: Get the last temperature readings from sensors

This example will demonstrate how to build a simple SOAP application in PHP. Suppose we want to get the last temperature readings from sensors. The corresponding REST call is as follows:
`http://192.168.0.2/gw/rest/V1/?method=events.readLast&name=TemperatureReadEvent` As we did for Java, you can look up the SOAP API reference to find the name of the method. However, the usage is a bit different due to the nature of PHP language. First, it does not require any explicit type declaration. Second, member variables are accessed directly, not through the methods as in Java.

Figure 2.9 shows the contents of Java `get()` function that gets the last temperature readings from sensors. Figure 2.10 shows the result of the PHP program `get_last_temperature.php`.

```
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->eventsReadLast($attrName, $mote);
    $ret_arr = $ret->results;
    foreach ($ret_arr as $result) {
        echo "Mote " . $result->addr .  " Time " . $result->timestamp .
             " " . $attrName .  " " . $result->value->TemperatureReadEvent ."\n";
    }
}
```

Figure 2.9: PHP get() function that gets the last temperature readings from sensors

```
$ php get_last_temperature.php 192.168.0.2
Mote 00173b000fecb28f Time 1182556899.4362 TemperatureReadEvent 7252
Mote 00173b0010241593 Time 1182556899.1605 TemperatureReadEvent 7304
```

Figure 2.10: The execution result of the PHP application get_last_temperature.php

2.2 A Few Exercises for Writing SOAP Applications

In the rest of this section, we provide a few examples of SOAP applications both in Java and PHP code:

- Java code: Ex2_1.java, Ex2_2.java, $\cdots$, Ex2_6.java
- PHP: ex2_1.php, ex2_2.php, $\cdots$, ex2_6.php

Based on the description and execution result, complete the code of each program by filling in the TODO in the skeleton code.

2.2.1 Exercise 1: Get the names of nodes

The first exercise is to get the names of the all the nodes in the network. The corresponding REST call is as follows:
`http://192.168.0.2/gw/rest/V1/?method=metadata.get&name=name`

```
$ java Ex2_1 192.168.0.2 (or php ex2_1.php 192.168.0.2)
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
Mote 00173b000c827501 Name 14fourteen
Mote 00173b000c834627 Name 12twelve
Mote 00173b000c847d22 Name 13thirteen
Mote 00173b000fecb28f Name Node 00173b000fecb28f
Mote 00173b0010203e21 Name 2two
Mote 00173b00102412ef Name 6six
Mote 00173b0010241593 Name Node 00173b0010241593
Mote 00173b00102426d2 Name 1one
Mote 00173b00102430f7 Name 3three
Mote fffffffffffffffe Name SnIPSnap
```

Figure 2.11: The execution result of the exercise 2-1

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class Ex2_1
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    Ex2_1(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        }
        catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(String name, String addr)
    {
        GW__MetaData_Result[] results;
        try {
            // TODO begin: complete the following segment of code:
            // results = <fill-in-your-code>;
            // for (int i = 0; i < results.length; i++) {
            //     System.out.println(
            //        "Mote " + <fill-in-your-code> +
            //        " Name " + <fill-in-your-code>
            //     );
            // }
            // TODO end
        }
        catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java Ex2_1 gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName = "name";
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        Ex2_1 get = new Ex2_1(gateway);
        get.get(attrName, mote);
    }
}
```

Figure 2.12: Skeleton code for Ex2_1.java

```
<?php
define("TIMEOUT", 2.0);
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    // TODO: begin
    //$ret = <fill-in-your-code>;
    //foreach ($ret as $result) {
    //    echo "Mote " . <fill-in-your-code> .
    //         " Name " . <fill-in-your-code> ."\n";
    //}
    // TODO: end
}
if ($argc != 1 && $argc != 2) { echo "Usage: php ex2_1.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "name";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
get($gateway, $attrName, $mote);
?>
```

Figure 2.13: Skeleton code for ex2_1.php

2.2.2 Exercise 2: Get the locations of the nodes

The next exercise is to get the locations of the all the nodes in the network. The corresponding REST call is as follows:
`http://192.168.0.2/gw/rest/V1/?method=metadata.get&name=map-x`
`http://192.168.0.2/gw/rest/V1/?method=metadata.get&name=map-y`

```
$ java Ex2_2 192.168.0.2 (or php ex2_2.php 192.168.0.2)
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
Mote 00173b000c827501 map-x 170
Mote 00173b000c834627 map-x 20
Mote 00173b000c847d22 map-x 170
Mote 00173b000fecb28f map-x 180
Mote 00173b0010203e21 map-x 90
Mote 00173b00102412ef map-x 170
Mote 00173b0010241593 map-x 185
Mote 00173b0010203e21 map-x 90
Mote 00173b00102412ef map-x 170
Mote 00173b0010241593 map-x 185
Mote 00173b00102426d2 map-x 90
Mote 00173b00102430f7 map-x 330
Mote fffffffffffffffe map-x 20
Mote 00173b000c827501 map-y 270
Mote 00173b000c834627 map-y 420
Mote 00173b000c847d22 map-y 355
Mote 00173b000fecb28f map-y 170
Mote 00173b0010203e21 map-y 270
Mote 00173b00102412ef map-y 180
Mote 00173b0010241593 map-y 345
Mote 00173b00102426d2 map-y 180
Mote 00173b00102430f7 map-y 100
Mote fffffffffffffffe map-y 380
```

Figure 2.14: The execution result of the exercise 2-2

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class Ex2_2
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    Ex2_2(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(String name, String addr)
    {
        GW__MetaData_Result[] results;
        try {
            // TODO: begin
            //results = <fill-in-your-code>;
            //for (int i = 0; i < results.length; i++) {
            //    System.out.println(
            //      "Mote " + <fill-in-your-code> +
            //      " " + name + " " +
            //      <fill-in-your-code>
            //    );
            //}
            // TODO: end
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java Ex2_2 gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        Ex2_2 get = new Ex2_2(gateway);
        attrName = "map-x";
        get.get(attrName, mote);
        attrName = "map-y";
        get.get(attrName, mote);
    }
}
```

Figure 2.15: Skeleton code for Ex2_2.java

```
<?php
define("TIMEOUT", 2.0);
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    // TODO: begin
    //$ret = <fill-in-your-code>;
    //foreach ($ret as $result) {
    //    echo "Mote " . <fill-in-your-code> .
    //         " " . $attrName . " " . <fill-in-your-code> . "\n";
    //}
    // TODO: end
}
if ($argc != 1 && $argc != 2) { echo "Usage: php ex2_2.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
$attrName = "map-x";
get($gateway, $attrName, $mote);
$attrName = "map-y";
get($gateway, $attrName, $mote);
?>
```

Figure 2.16: Skeleton code for ex2_2.php

2.2.3 Exercise 3: Get all data in a recent time window (1 day)

The next exercise is to get all the temperature readings in a recent time window, e.g. 1 day. The corresponding REST call is as follows:
`http://192.168.0.2/gw/rest/V1/?method=events.readRelative&t1Offset=-86400&t2Offset=0`

```
$ java Ex2_3 192.168.0.2 (or php ex2_3.php 192.168.0.2)
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
Mote 00173b0010241593 timestamp 1.18247200152407E9 TemperatureReadEvent 7281
Mote 00173b000fecb28f timestamp 1.18247200164372E9 TemperatureReadEvent 7227
Mote 00173b0010241593 timestamp 1.18247230151439E9 TemperatureReadEvent 7281
Mote 00173b000fecb28f timestamp 1.18247230165012E9 TemperatureReadEvent 7225
...
```

Figure 2.17: The execution result of the exercise 2-3

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class Ex2_3 {
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    Ex2_3(String gateway) {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(double t1Offset, double t2Offset, String name, String addr) {
        GW__eventsReadRelative_Result result;
        GW__Event_Result[] results;
        GW__Event value;
        org.apache.axis.types.UnsignedInt offset = new org.apache.axis.types.UnsignedInt(0);
        try {
            // TODO: begin
            //result = <fill-in-your-code>;
            //results = result.getResults();
            //for (int i = 0; i < results.length; i++) {
            //    value = <fill-in-your-code>;
            //    System.out.println(
            //      "Mote " + <fill-in-your-code> +
            //      " timestamp " +  <fill-in-your-code> +
            //      " " + name +
            //      " " + <fill-in-your-code>
            //    );
            //}
            // TODO: end
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args) {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java Ex2_3 gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        Ex2_3 get = new Ex2_3(gateway);
        double t1Offset = -86400.0;   // one day before
        double t2Offset = 0.0;        // now
        attrName = "TemperatureReadEvent";
        get.get(t1Offset, t2Offset, attrName, mote);
    }
}
```

Figure 2.18: Skeleton code for Ex2_3.java

```
<?php
define("TIMEOUT", 2.0);
function get($t1Offset, $t2Offset, $gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    // TODO: begin
    //$ret = <fill-in-your-code>;
    //$ret_arr = $ret->results;
    //foreach ($ret_arr as $result) {
    //    echo "Mote " . <fill-in-your-code> .
    //         " Time " . <fill-in-your-code> .
    //         " " . $attrName .
    //         " " . <fill-in-your-code> ."\n";
    //}
    // TODO: end
}
if ($argc != 1 && $argc != 2) { echo "Usage: php ex2_3.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "TemperatureReadEvent";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
$t1Offset = -86400;
$t2Offset = 0;
get($t1Offset, $t2Offset, $gateway, $attrName, $mote);
?>
```

Figure 2.19: Skeleton code for ex2_3.php

2.2.4 Exercise 4: Request a sample from all enabled sensors

The next exercise is to request a sample from all enabled sensors. The corresponding REST call is as follows:
`http://192.168.0.2/gw/rest/V1/?method=rpc.execute&name=sampleRequest`

```
$ php ex2_4.php 192.168.0.2 (or java Ex2_4 192.168.0.2)
Mote 00173b000fecb28f Time 1182558375.7276
Mote 00173b000c834627 Time 1182558375.7276
Mote 00173b0010203e21 Time 1182558375.7276
Mote 00173b00102426d2 Time 1182558375.7276
Mote 00173b000c827501 Time 1182558375.7276
Mote 00173b000c847d22 Time 1182558375.7276
Mote 00173b00102430f7 Time 1182558375.7276
Mote 00173b0010241593 Time 1182558375.7276
Mote 00173b00102412ef Time 1182558375.7276
```

Figure 2.20: The execution result of the exercise 2-4

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class Ex2_4
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    Ex2_4(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        }
        catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void rpcExecute(String addr)
    {
        Void_Result[] results;
        try {
            // TODO: begin
            //results = <fill-in-your-code>;
            //for (int i = 0; i < results.length; i++) {
            //    System.out.println(
            //      "Mote " + <fill-in-your-code> +
            //      " Time " + <fill-in-your-code>
            //    );
            //}
            // TODO: end
        }
        catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java Ex2_4 gateway [mote]");
            System.exit(3);
        }
        String gateway = args[0];
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        Ex2_4 stub = new Ex2_4(gateway);
        stub.rpcExecute(mote);
    }
}
```

Figure 2.21: Skeleton code for Ex2_4.java

```
<?php
define("TIMEOUT", 2.0);
function rpc_execute($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    // TODO: begin
    //$ret = <fill-in-your-code>;
    //foreach ($ret as $result) {
    //    echo "Mote " . <fill-in-your-code> .
    //         " Time " . <fill-in-your-code> . "\n";
    //}
    // TODO: end
}
if ($argc != 1 && $argc != 2) { echo "Usage: php ex2_4.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "name";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
rpc_execute($gateway, $attrName, $mote);
?>
```

Figure 2.22: Skeleton code for ex2_4.php

2.2.5 Exercise 5: Get the battery voltage of the nodes

The next exercise is to get the battery voltage of the nodes. The corresponding REST call is as follows: `http://192.168.0.2/gw/rest/V1/?method=events.readLast&name=NodeStatsEvent`

```
$ php ex2_5.php 192.168.0.2 (or java Ex2_5 192.168.0.2)
Mote 00173b00102426d2 Time 1182558331.1052 Voltage 2999
Mote 00173b000c847d22 Time 1182558222.7587 Voltage 2965
Mote 00173b0010203e21 Time 1182558420.274 Voltage 2999
Mote 00173b00102430f7 Time 1182558324.6725 Voltage 2999
Mote 00173b000c827501 Time 1182558419.6195 Voltage 2999
Mote 00173b000fecb28f Time 1182558256.1617 Voltage 2999
Mote 00173b000c834627 Time 1182558267.2399 Voltage 2999
Mote 00173b0010241593 Time 1182558359.115 Voltage 2852
Mote 00173b00102412ef Time 1182558402.1318 Voltage 2999
```

Figure 2.23: The execution result of the exercise 2-5

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class Ex2_5 {
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    Ex2_5(String gateway) {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(String name, String addr) {
        GW__eventsReadLast_Result result;
        GW__Event_Result[] results;
        GW__Event value;
        Nx_node_stats_t event;
        try {
            // TODO: begin
            //result = <fill-in-your-code>;
            //results = result.getResults();
            //for (int i = 0; i < results.length; i++) {
            //    value = <fill-in-your-code>;
            //    event = <fill-in-your-code>;
            //    System.out.println(
            //      "Mote " + <fill-in-your-code> +
            //      " Time " + <fill-in-your-code> +
            //      " Voltage " + <fill-in-your-code>
            //    );
            //}
            // TODO: end
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args) {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java Ex2_5 gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        Ex2_5 get = new Ex2_5(gateway);
        attrName = "NodeStatsEvent";
        get.get(attrName, mote);
    }
}
```

Figure 2.24: Skeleton code for Ex2_5.java

```
<?php
define("TIMEOUT", 2.0);
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    // TODO: begin
    //$ret = <fill-in-your-code>;
    //$ret_arr = $ret->results;
    //foreach ($ret_arr as $result) {
    //    echo "Mote " . <fill-in-your-code> .
    //         " Time " . <fill-in-your-code> .
    //         " Voltage " . <fill-in-your-code> . "\n";
    //}
    // TODO: end
}
if ($argc != 1 && $argc != 2) { echo "Usage: php ex2_5.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "NodeStatsEvent";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
get($gateway, $attrName, $mote);
?>
```

Figure 2.25: Skeleton code for ex2_5.php

2.2.6 Exercise 6: Get statistics on node traffic and network reliability

The next exercise is to get statistics on node traffic and network reliability. The corresponding REST call is as follows:
`http://192.168.0.2/gw/rest/V1/?method=events.readRelative&name=NodeStatsEvent&t1Offset=-86400&t2Offset=0`

```
$ php ex2_6.php 192.168.0.2 (or java Ex2_6 192.168.0.2)
Mote 00173b000c827501 timestamp 1182472228.9383 sent 1512  delivered 1428
Mote 00173b00102426d2 timestamp 1182472229.3708 sent 1477  delivered 1425
Mote 00173b0010241593 timestamp 1182472231.9069 sent 1783  delivered 1745
Mote 00173b000fecb28f timestamp 1182472263.1175 sent 2222  delivered 2098
Mote 00173b000c834627 timestamp 1182472278.3247 sent 1471  delivered 1422
Mote 00173b00102412ef timestamp 1182472349.9155 sent 1503  delivered 1433
Mote 00173b00102430f7 timestamp 1182472352.6 sent 1476  delivered 1415
```

Figure 2.26: The execution result of the exercise 2-6

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class Ex2_6 {
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    Ex2_6(String gateway) {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(double t1Offset, double t2Offset, String name, String addr) {
        GW__eventsReadRelative_Result result;
        GW__Event_Result[] results;
        GW__Event value;
        Nx_node_stats_t event;
        org.apache.axis.types.UnsignedInt offset = new org.apache.axis.types.UnsignedInt(0);
        try {
            // TODO: begin
            //result = <fill-in-your-code>;
            //results = result.getResults();
            //for (int i = 0; i < results.length; i++) {
            //    value = <fill-in-your-code>;
            //    event = <fill-in-your-code>;
            //    System.out.println(
            //      "Mote " + <fill-in-your-code> +
            //      " timestamp " + <fill-in-your-code> +
            //      " sent " + <fill-in-your-code> +
            //      " delivered " + <fill-in-your-code>
            //    );
            //}
            // TODO: end
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args) {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java Ex2_6 gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        Ex2_6 get = new Ex2_6(gateway);
        double t1Offset = -86400.0;   // one day before
        double t2Offset = 0.0;        // now
        attrName = "NodeStatsEvent";
        get.get(t1Offset, t2Offset, attrName, mote);
    }
}
```

Figure 2.27: Skeleton code for Ex2_6.java

```
<?php
define("TIMEOUT", 2.0);
function get($t1Offset, $t2Offset, $gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    // TODO: begin
    //$ret = <fill-in-your-code>;
    //$ret_arr = $ret->results;
    //foreach ($ret_arr as $result) {
    //    echo "Mote " . <fill-in-your-code> .
    //       " timestamp " . <fill-in-your-code> .
    //       " sent " . <fill-in-your-code> . " " .
    //       " delivered " . <fill-in-your-code> . "\n";
    //}
    // TODO: end
}
if ($argc != 1 && $argc != 2) { echo "Usage: php ex2_6.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "NodeStatsEvent";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
$t1Offset = -86400;
$t2Offset = 0;
get($t1Offset, $t2Offset, $gateway, $attrName, $mote);
?>
```

Figure 2.28: Skeleton code for ex2_6.php

2.3 Writing PHP SOAP applications for Web Service

In this section, we demonstrate how to write a PHP SOAP application for web service. Compared to other programming languages, PHP integrates well with web service requiring only minimum amount of code modification.

2.3.1 Installation Steps for PHP Web Service

The first thing you need to do is set up PHP on your machine. We tested with php-5.2.3-win32-installer on a Windows machine. Make sure that SOAP is selected as an external component when you install PHP.

Next, you need to install an Apache web server. We tested with apache_2.0.59-win32-x86-no_ssl on a Windows machine. After installing the Apache web server, you have to manually set up "httpd.conf" file to activate PHP in the web server (Figure 2.29).

```
# For PHP 5 do something like this:
LoadModule php5_module "c:/Program Files/PHP/php5apache2.dll"
AddType application/x-httpd-php .php
# configure the path to php.ini
PHPIniDir "C:/Program Files/PHP"
```

Figure 2.29: Modification to httpd.conf to activate PHP in the Apache web server

Then, you need to move PHP source code to the web server root directory. When we tested with the Apache web server, we used the following directory as the web server root:
`C:\Program Files\Apache Group\Apache2\htdocs`

2.3.2 PHP Web Service Example

The following example, get_last_temperature_web.php, is very similar to its command line counterpart get_last_temperature. A few lines of HTML syntax is embedded to display the output as an HTML table (Figure 2.30). Figure 2.31, Figure 2.32 and Figure 2.33 are the source code for this example.

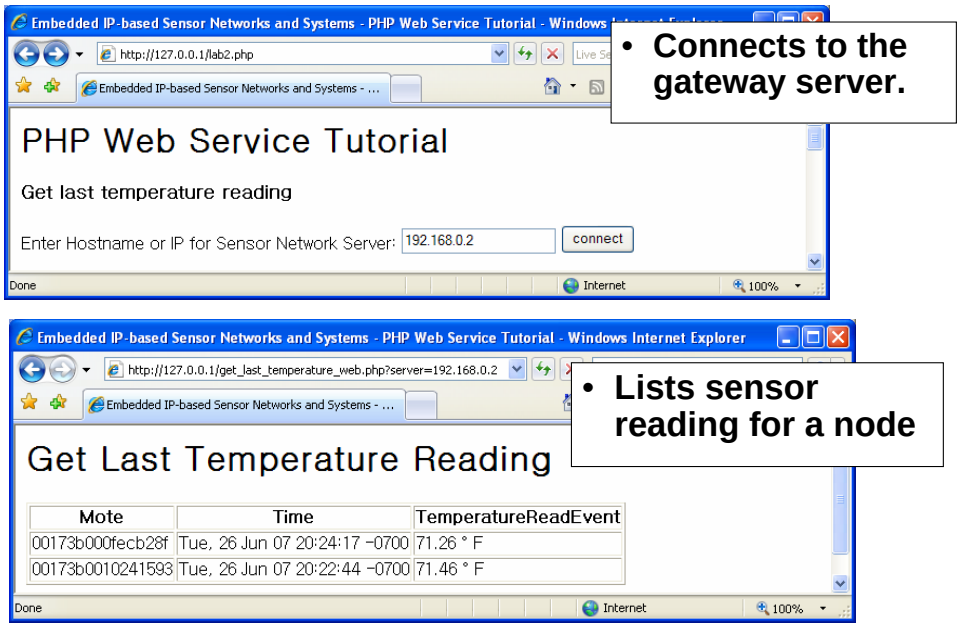

Figure 2.30: PHP Web Service Example

```
<?php
/* Simple pretty print of PHP objects within html stream */
function pprint_r($arg){
  echo "<pre>"; print_r($arg); echo "</pre>";
}
define (DEFAULTADDR, 'FFFFFFFFFFFFFFFF');
define (GWADDR, 'FFFFFFFFFFFFFFFE');
/* Address Conversions */
function EUIDtoIPv4($inets, $euid){
  foreach($inets as $inet) { if ($inet->addr == $euid) return $inet->ipv4Addr; }
  return NULL;
}
function IPv4toEUID($inets, $ip){
  foreach($inets as $inet) { if ($inet->ipv4Addr == $ip) return $inet->addr; }
  return NULL;
}
function findNode($nodes, $euid) {
  foreach($nodes as $node) { if ($node->addr === $euid) return $node; }
  return NULL;
}
function EUIDtoAddr($addrMaps, $euid){
  foreach($addrMaps as $map) { if ($map->longAddr == $euid) return $map->shortAddr; }
  return NULL;
}
/* Extract ReadEvents from list of all events. */
function GetReadEvents($elist) {
  $readEvents=array();
  foreach ($elist as $event) {
    $name = $event->name;
    if (strpos($name,"ReadEvent")=== (strlen($name)-9)){ $readEvents []= $name; }
  }
  return $readEvents;
}
?>
```

Figure 2.31: Header file to be included for a PHP source file

```
<?php require 'ARlib.inc'; ?>
<html>
  <head> <title>Embedded IP-based Sensor Networks and Systems - PHP Web Service Tutorial </title> </head>
  <body>
    <H1> PHP Web Service Tutorial </H1>
  <form action='get_last_temperature_web.php' method="GET">
  <H3> Get last temperature reading </H3>
    Enter Hostname or IP for Sensor Network Server: <input type="text" name="server" />
    <input type="submit" value="connect" />
  </form>
  <p>
  <form action='get_name_web.php' method="GET">
  <H3> List of Nodes </H3>
    Enter Hostname or IP for Sensor Network Server: <input type="text" name="server" />
    <input type="submit" value="connect" />
  </form>
  <p>
  <form action='get_location_web.php' method="GET">
  <H3> List of Node Locations </H3>
    Enter Hostname or IP for Sensor Network Server: <input type="text" name="server" />
    <input type="submit" value="connect" />
  </form>
  <p>
  <form action='get_last_one_day_temperature_web.php' method="GET">
    <H3> Temperature for the last period </H3>
    Enter Hostname or IP for Sensor Network Server: <input type="text" name="server" />
    <p>
    Enter the period to watch (in hours): <input type="text" name="hours" />
    <input type="submit" value="connect" />
  </form>
  <p>
  <form action='rpc_sample_request_web.php' method="GET">
  <H3> Sample Request </H3>
    Enter Hostname or IP for Sensor Network Server: <input type="text" name="server" />
    <input type="submit" value="connect" />
  </form>
  <p>
  <form action='get_last_battery_voltage_web.php' method="GET">
  <H3> Node battery voltage </H3>
    Enter Hostname or IP for Sensor Network Server: <input type="text" name="server" />
    <input type="submit" value="connect" />
  </form>
  <p>
  <form action='get_last_one_day_reliability_web.php' method="GET">
  <H3> Reliability for the last period </H3>
    Enter Hostname or IP for Sensor Network Server: <input type="text" name="server" /> <p>
    Enter the period to watch (in hours): <input type="text" name="hours" />
    <input type="submit" value="connect" />
  </form>
  <p>
  </body>
</html>
```

Figure 2.32: PHP source code for lab2.php

```
  <head>
    <title>Embedded IP-based Sensor Networks and Systems - PHP Web Service Tutorial </title>
  </head>
  <body>
    <H1> Temperature Readings of the Last Period </H1>
<?php
define("TIMEOUT", 2.0);
function get($t1Offset, $t2Offset, $gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->eventsReadRelative($t1Offset, $t2Offset, $attrName, $mote, 0, "");
    echo "<table border=1>";
    echo "<tr><th>Mote</th><th>Time</th><th>Temperature</th></tr>";
    $ret_arr = $ret->results;
    foreach ($ret_arr as $result) {
      $value = $result->value;
      $reading = $value->TemperatureReadEvent * 0.01;
      $converted = "$reading &#176 F";
      $date_out = date(DATE_RFC822,$result->timestamp);
      echo "<tr><td> $result->addr </td>";
      echo "<td> $date_out </td>";
      echo "<td> $converted </td></tr>";
    }
    echo "</table>";
    echo "<p>";
}
$gateway = $_GET['server']; // obtain argument from http GET
$hours = $_GET['hours']; // obtain argument from http GET
$attrName = "TemperatureReadEvent";
$mote = "ffffffffffffffff";
$t1Offset = -3600.0 * $hours;
$t2Offset = 0.0;
get($t1Offset, $t2Offset, $gateway, $attrName, $mote);
?>
  </body>
</html>
```

Figure 2.33: PHP source code for get_last_temperature_web.php

2.3.3 Commandline PHP versus Web-based PHP

Table 2.1 compares the commandline PHP with web-based PHP.

Table 2.1: Commandline PHP versus Web-based PHP

	Command Line PHP	**Web-based PHP**
Command Line Passing	Command line parameters. e.g. `$gateway = $argv[1];`	Named parameters. e.g. Caller: `<a href = \"node.php?server=$server&ip=$ip\">$ip</a>` e.g. Callee: `$server = $_GET['server'];`
Debugging	Both program result and error on console.	Progrm result on web browser. Error on web server error log.

Etc	Nested variables allowed.
	e.g: `echo $result->value-> NodeStatsEvent->voltage;`
		Nested variables should be declared in cascade for access.
		e.g. `$value = $result->value;`
		`$event = $value-> NodeStatsEvent;`
		`echo $event->voltage;`

2.3.4 Exercise 2-7: PHP Web Service Exercise

Modify get_last_temperature_web.php so that it displays temperature in Celsius rather than Fahrenheit (Figure 2.34).

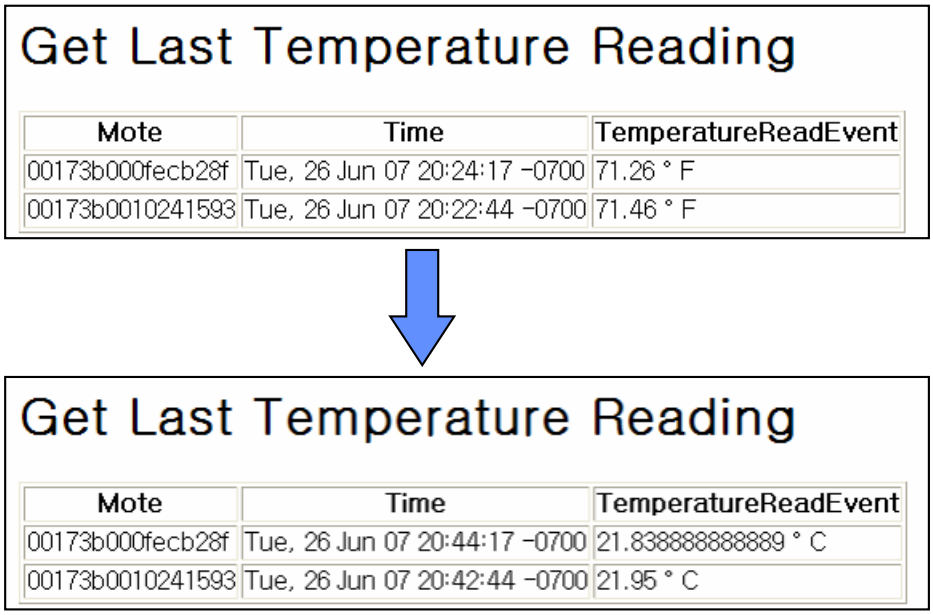

Figure 2.34: PHP Exercise

2.4 Custom analysis of the networking

In this section, we will demonstrate how to use the PHP web service to customize the analysis of the networking.

2.4.1 Before the site survey

Discuss the connectivity that you have seen.

At what RSSI, does a node forego a link and try an alternative?

How does the tree change over time?

2.4.2 How to do site survey using SOAP API

The two sample SOAP appliations, `rpc_connectivity__test_start.php` and `get_connectivity_survey_web.php` show how the site survey can be done using SOAP API.

The requestor program `rpc_connectivity__test_start.php` uses `rpcExecuteconnectivityTestStart()` function to request a site survey from node A to B. It gets the address of all the nodes in the network and iterates RPC for all permutations of source and destination.

The viewer program `get_connectivity_survey_web.php` uses the fact that a `ConnectivityTestReceiverReport` event is triggered for each site survey. It retrieves recent occurences of `ConnectivityTestReceiverReport` using `eventsReadRelative()` function.

2.4.3 Requesting site survey

Figure 2.35 shows how the requestor program is executed.

```
$ php rpc_connectivity_test_start.php 192.168.0.2
Surveying link from 00173b000c827501 to 21888.
Mote 00173b000c827501 Time 1183003949.4916
Surveying link from 00173b000c827501 to 1708.
Mote 00173b000c827501 Time 1183003949.6586
Surveying link from 00173b000c827501 to 10611.
Mote 00173b000c827501 Time 1183003949.8156
Surveying link from 00173b000c827501 to 12423.
Mote 00173b000c827501 Time 1183003949.9836
Surveying link from 00173b000c827501 to 21828.
Mote 00173b000c827501 Time 1183003950.2036
Surveying link from 00173b000c827501 to 22897.
Mote 00173b000c827501 Time 1183003950.4436
Surveying link from 00173b000c827501 to 10954.
Mote 00173b000c827501 Time 1183003950.6406
```

Figure 2.35: Requesting Site Survey

2.4.4 Getting the site survey results

Figure 2.36 shows how the viewer program is executed.

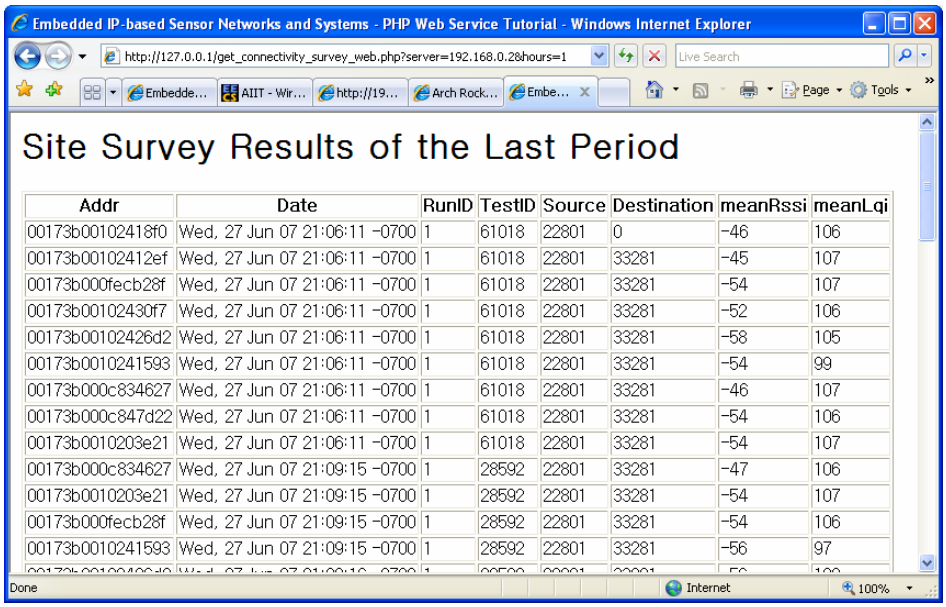

Figure 2.36: Site Survey Results

2.4.5 Run site survey for your network

Discuss the correlation between environmental conditions and network changes.

Chapter 3

Using IP and 6LoWPAN Networking

3.1 Configuration for IP access

3.1.1 Setting up the LoWPAN IP address

In order to access sensor nodes at IP level, you need to configure the gateway server and client machines to correctly route messages.

First, you need to find what IP address each sensor nodes has. In the "Nodes" page, expand the link for each node to find the IP address of each node. For example, the node "12twelve" has an IPv4 address of 10.97.85.128 and IPv6 address of fd42:8e0:300f:6172:0:0:0:5580 (Figure 3.1).

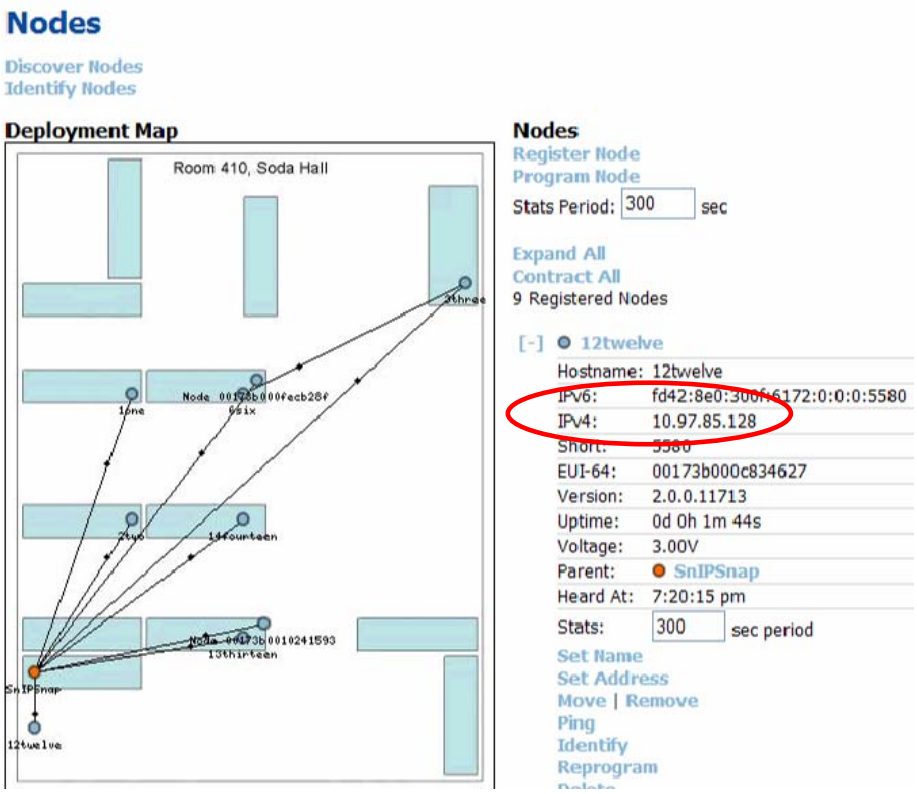

Figure 3.1: Finding the IP address of a sensor node

By default, a sensor node is given an IPv4 address of 10.97.0.0/16. If you are to give a different IP address prefix, you can set it at "Server" page - LoWPAN link. In a classroom environment where you want to maintain a different network for each group, it is a good idea to assign a different IP address prefix for each group. This makes sure that all the sensor nodes are assigned different IP address even though they are not programmed with the same gateway server.

If you change the LoWPAN IP address prefix, you need to discover the network again. This will make each sensor node assigned its LoWPAN IP address based on the new IP address prefix.

3.1.2 Setting up the routing to nodes

Sensor nodes are assigned their LoWPAN IP addresses from the gateway server, but you cannot access sensor nodes from a client machine at IP level yet because the route to the LoWPAN is not set up yet.

You can access the LoWPAN at IP level by adding the route to it. Suppose the IPv4 prefix is 10.97 and the IP address of the gateway on the Ethernet link is 192.168.0.2. Then, you can set up the route with the following command:

- Windows: route ADD 10.97.0.0 MASK 255.255.0.0 192.168.0.2
- Linux: route add -net 10.97.0.0 netmask 255.255.0.0 gw 192.168.0.2

Exercise 3-1

Type the routing command for your environment.

3.1.3 Ping nodes

- **Step 1:** Find IP address of a node. The procedure is described earlier in this section.
- **Step 2:** Type 'ping IP-address'. Figure 3.2 shows an execution result.

```
$ ping 10.97.85.128

Pinging 10.97.85.128 with 32 bytes of data:
Reply from 10.97.85.128: bytes=32 time=124ms TTL=32
Reply from 10.97.85.128: bytes=32 time=134ms TTL=32
Reply from 10.97.85.128: bytes=32 time=143ms TTL=32
Reply from 10.97.85.128: bytes=32 time=136ms TTL=32
Ping statistics for 10.97.85.128:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 124ms, Maximum = 143ms, Average = 134ms
```

Figure 3.2: To ping a node

Exercise 3-2

Ping one of the nodes in your network.

3.2 IP access using standard client programs

3.2.1 IP services provided by a sensor node

An Arch Rock Primer Pack node provides a couple of IP services: 7 (echo), 10 (leds), 11 (sysstat), 15 (netstat) and 30 (raw sensor reading).

```
$ java EchoClient 10.97.85.128
Connected to echo server
Hello, world.
Hello, world.
```

Figure 3.3: Port 7 (echo)

```
$ java LedsClient 10.97.85.128
Connected to LEDs server
3
```

Figure 3.4: Port 10 (leds)

```
$ java SysstatClient 10.97.85.128
Connected to sysstat server
pltfm   : telosb
proc    : MSP430F1611
eui64   : 00173B000C834627
version  : 2.0.0.11713
builton : 1178639115
uptime  : 6504
cputime : 30
radtime : 73
mV      : 2999
wdt     : 0
```

Figure 3.5: Port 11 (sysstat)

```
$ java NetstatClient 10.97.85.128
Connected to netstat server
Lnk     TX      PTX     RX      OVH
        109     52      527     115
Net     TX      RTX     DRP     OVR
        72      0       0       0
Rte             Addr    Hops    RSSI
        0:      0       0       -40
        1:      22897   1       -58
        2:      10954   1       -52
```

Figure 3.6: Port 15 (netstat)

```
$ java SensorReadClient 10.97.6.172
Connected to sensor reading server

Hum  : 0
Temp : 0
PAR  : 0
TSR  : 2
Vcc  : 4020
ADC0 : 1179
ADC1 : 1421
IO0  : 0
IO1  : 0
IO2  : 0
Swtch: 0
```

Figure 3.7: Port 30 (raw sensor reading)

There are different ways to access IP services on sensor nodes: telnet, netcat and custom IP application.

- **Telnet:**
  Syntax: `telnet <node IP address> <port>`
  An execution example is shown at Figure 3.8.

- **Netcat:** netcat can pull information from a host at certain port.
  Syntax: `netcat <node IP address> <port>`

- **Custom IP application:** You can also access IP services with your favorite programming language with TCP or UDP socket API. To interact with an IP service, a custom IP application needs to send and receive lines of characters as a message and response. Figure 3.9 shows an example of custom IP application written in Java.

```
$ telnet 10.97.6.172 30
Hum  : 0
Temp : 0
PAR  : 0
TSR  : 2
Vcc  : 4019
ADC0 : 1180
ADC1 : 1419
IO0  : 0
IO1  : 0
IO2  : 0
Swtch: 0
```

Figure 3.8: telnet to port 30

```
try {
  Socket theSocket = new Socket(hostname, 30);
  networkIn = new BufferedReader(
   new InputStreamReader(theSocket.getInputStream()));
  BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));
  out = new PrintWriter(theSocket.getOutputStream());
  System.out.println("Connected to sensor reading server");
  String reply reply = networkIn.readLine();
  while (reply != null) {
    System.out.println(reply);
    reply = networkIn.readLine();
  }
}  // end try
```

Figure 3.9: An example of custom IP application written in Java

Exercise 3-3

What are the raw sensor readings when you telneted to one of the nodes?

Exercise 3-4

What are the RSSI reading to the parent node and the first alternate parent node when you netcated to a node?

Chapter 4

TinyOS 2.0 based embedded Applications

4.1 TinyOS Programming on Open Source Distribution

In previous chapters, we explored various fuctions of wireless sensor networks (WSNs) with a top-down approach using a cutting-edge WSN kit from Arch Rock. In this chapter, we take a bottom-up approach and demonstrate how to build basic functions of WSN using an open source distribution of TinyOS 2.0.

4.1.1 How to get TinyOS open source distribution

The lastest version of TinyOS is TinyOS 2.x and available in several formats.

- **XubunTOS Live CD:**
  Researchers from Colorado School of Mines made TinyOS 2.x as a Linux live CD:
  `http://toilers.mines.edu/Public/XubunTOS`
  You can have a working TinyOS 2.x environment simply by booting with the live CD. It is a recommended distribution for classroom environment. The live CD automatically sets up the network connection in a DHCP network. If you need a static IP address for each client machine, you may need to set up the IP address after booting with the live CD to access the Internet.

- **Debian or Ubuntu Linux:**
  TinyOS 2.x is distributed as a Debian package and this makes it easy to install for Debian or Ubuntu Linux:
  `http://www.tinyos.net/scoop/story/2007/3/1/142950/4912`
  `http://www.5secondfuse.com/tinyos/install.html`
  This is a recommended distribution for development environment.

- **Other Linux or Windows Platform:**
  Manual installation is needed:
  `http://www.tinyos.net/tinyos-2.x/doc/html/install-tinyos.html`

4.1.2 Directions for the rest of chapter

This chapter assumes that you have a working TinyOS 2.x environment and an Internet access. Please follow the directions below:

Table 4.1: Tinyos Projects to be covered in this chapter

	TinyOS Concept	**Start**	**Produce**
1	module, configuration, command, event	1_Push	1_Toggle
2	generics, virtualized services	2_Blink	2_Count
3	network debugging	3_PrintSerial	
4	sensing, split-phase, parameterized	4_Single	4_Dual
5	post, task	5_Raw	5_Smooth
6	sending radio message	6_Counts	6_Readings
7	receiving radio message	7_Request	7_RequestSample
8	sending and receiving radio message	8_CountToRadio	8_RadioToCount

- **Step 1:** Run TinyOS live CD on each machine by inserting a live CD in the CD drive and boot the client machine with it.
- **Step 2:** Copy each of skeleton projects in directory $TOSROOT/apps.
- **Step 3:** Build the skeleton prjects and try them.
  - Go to the directory of each skeleton project.
  - Type `make telosb` for buillding an image.
  - Type `make telosb reinstall.<node id>` for program loading.
  - Optionally, run a Java client for user interaction.
- **Step 4:** Produce solutions based on skeleton projects and idea for extension as shown in Table 4.1.

4.2 TinyOS 2.0 Programming Projects

4.2.1 Project 1: Push and Toggle

- **Push:** a minimal TinyOS 2.0 application.
  Behavior - While the user button is pressed the red LED (led0) lights.
- **Toggle:**
  Behavior - When the user button is pressed, the red LED (led0) is toggled.
- **Key point:**
  Notify.notify() event is triggered whenever the user button is either pressed or released. Change the logic of the event handler so that red LED is toggled each time the user button is pressed.
- **Concepts Illustrated:**
  - **module** - the implementation of a functional element that can be composed into a larger elements through configurations. This file, with a name ending in 'M' is an example module. Modules contain state (variables), internal functions, functions that implement one side of an interface, and tasks that provide logical concurrency (not shown here). The implementation of a component is in terms of the namespace provided by its interfaces.
  - **configuration** - a component that composes a set of components by wiring together their interfaces.

- **commands** - externally accessible functions that can be called across an interface between components, typically to initiate actions.
- **events** - externally accessible handlers that can be signaled across and interface between components, typically to notify of an occurance.
- **interfaces** - bidirectional collections of typed function signatures for commands and envents. This module uses three interfaces provided by lower level subsystems.
  See `$TOSROOT/tinyos/tos/interfaces/`

```
COMPONENT=PushAppC
include $(MAKERULES)
```

Figure 4.1: Makefile: Makefile for Push

```
/**
 * Push - a minimal TinyOS 2.0 application
 *
 *            +---------+
 *            |  MainC  |
 *            +---------+
 *                 | init
 *                 |
 *    +--------------------------+
 *    |           PushC          |
 *    +--------------------------+
 *      | Boot     |Leds       |Notify
 *      |          |           |
 *      | Boot     |Leds       |Notify
 *   +-------+  +-------+  +-------------+
 *   | MainC |  | LedsC |  | UserButtonC |
 *   +-------+  +-------+  +-------------+
 *
 */
configuration PushAppC {}
implementation {
  components PushC;
  components MainC;
  components UserButtonC;
  components LedsC;
  PushC.Boot -> MainC;
  PushC.Notify -> UserButtonC;
  PushC.Leds -> LedsC;
}
```

Figure 4.2: PushAppC.nc: Configuration module for Push

```
#include <UserButton.h>
module PushC {
  uses {
    interface Boot;
    interface Notify<button_state_t>;
    interface Leds;
  }
}
implementation {
  event void Boot.booted() {
    call Notify.enable();
  }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) {
    call Leds.led0Toggle();
  }
}
```

Figure 4.3: PushC.nc: Implementation module for Push

4.2.2 Project 2: Blink and Count

- **Blink:**
  Behavior - Pressing the user button toggles the red LED (led0).
  Green LED (led1) blinks every second.

- **Count:**
  Behavior - Pressing the user button rotates a counter among 0, 1, 2 and 3.
  At 0, no timer is fired.
  At 1, Timer 0 with period 1s is fired.
  At 2, Timer 0 and Timer 1 (period 2s) are fired.
  At 3, Timer 0, Timer 1, Timer 2 (period 4s) are fired.
  Timer 0 toggles led0, Timer 1 toggles led1, Timer 2 toggles led2.

- **Concepts Illustrated:**
  Timer - a critical subsystem - TEP102
  Virtualized resource provided as a parameterized interface.

- **Key Point**

  - Instantiating a virtualized resource.
    `interface Timer<TMilli> as Timer0;`
    instantiates a milli-second timer among different timer types.
  - Instantiating multiple interfaces.
    `interface Timer<TMilli> as Timer0;`
    `interface Timer<TMilli> as Timer1;`
    `interface Timer<TMilli> as Timer2;`
    Access each timer as Timer0, Timer1, or Timer2.

```
COMPONENT=BlinkAppC
include $(MAKERULES)
```

Figure 4.4: Makefile: Makefile for Blink

```
/**
 * Blink - a TinyOS 2.0 application with timers
 *
 *            +---------+
 *            |  MainC  |
 *            +---------+
 *                 | init
 *                 |
 *    +-----------------------------------------+
 *    |           BlinkC                        |
 *    +-----------------------------------------+
 *      |Boot     |Leds       |Notify         |Timer0
 *      |         |           |               |
 *      |Boot     |Leds       |Notify         |Timer0
 *   +-------+  +-------+  +-------------+  +-------------+
 *   | MainC |  | LedsC |  | UserButtonC |  | TimerMilliC |
 *   +-------+  +-------+  +-------------+  +-------------+
 *
 */
configuration BlinkAppC {}
implementation {
  // Declare the components
  components BlinkC;
  components MainC;
  components UserButtonC;
  components LedsC;
  // Wire together the interfaces
  BlinkC.Boot -> MainC;
  BlinkC.Notify -> UserButtonC;
  BlinkC.Leds -> LedsC;
  components new TimerMilliC() as Timer0;
  BlinkC.Timer0 -> Timer0;
}
```

Figure 4.5: BlinkApp.nc: Configuration module for Blink

```
#include <UserButton.h>
module BlinkC {      // Definition in external interfaces
  uses {
    interface Boot;
    interface Notify<button_state_t>;
    interface Leds;
    interface Timer<TMilli> as Timer0;
  }
}
implementation {
  event void Boot.booted() {
    call Notify.enable();   // enable input subsystem upon boot
    call Timer0.startPeriodic(1024);   // Start 1s timer
  }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) {
    if (state == BUTTON_PRESSED) {
      call Leds.led0Toggle();
    }
  }
  event void Timer0.fired() {
    call Leds.led1Toggle();
  }
}
```

Figure 4.6: BlinkC.nc: Implementation module for Blink

4.2.3 Project 3: PrintSerial

- **PrintSerial:** A TinyOS application that shows the concept of network debugging.
- **Behavior:** When the user button is pressed, a counter variable is set to (counter + 1) mod 10. After that, a debugging message "Hello <counter>\n" is sent over the serial port (UART). In order to see the debugging message, plug the mote to a serial port and run the listen client by typing "source ./run.sh". The listen client reads a message from the serial port and prints it on console.
- **Concepts Illustrated:**
  Mote-PC serial communication - TEP113
  Tinyos 2.0 tutorial lesson 4

```
COMPONENT=PrintSerialAppC
BUILD_EXTRA_DEPS += PrintSerial.class
CLEAN_EXTRA = *.class PrintSerialMsg.java
PrintSerial.class: $(wildcard *.java) PrintSerialMsg.java
  javac *.java
PrintSerialMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=PrintSerialMsg PrintSerial.h print_serial_msg
-o $@
include $(MAKERULES)
```

Figure 4.7: Makefile: Makefile for SerialPrint

```
java PrintSerial -comm serial@/dev/ttyUSB0:telos
```

Figure 4.8: run.sh: Shell script to run the Java client program

```
#include "message.h"
typedef nx_struct print_serial_msg {
  nx_uint8_t buffer[TOSH_DATA_LENGTH];
} print_serial_msg_t;
enum {
  AM_PRINT_SERIAL_MSG = 0x0A,
};
```

Figure 4.9: PrintSerial.h: Header file to define the message structure

```
import java.io.IOException;
import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;
public class PrintSerial implements MessageListener {
  private MoteIF moteIF;
  public PrintSerial(MoteIF moteIF) {
    this.moteIF = moteIF;
    this.moteIF.registerListener(new PrintSerialMsg(), this);
  }
  public void messageReceived(int to, Message message) {
    PrintSerialMsg msg = (PrintSerialMsg)message;
    System.out.print("Received packet: ");
    for (int i = 0; i < msg.DEFAULT_MESSAGE_SIZE; i++) {
      char character = (char) msg.getElement_buffer(i);
      if (character == 0) {
        break;
      }
      System.out.print(character);
    }
    System.out.print("\n");
  }
  private static void usage() { System.err.println("usage: PrintSerial [-comm <source>]"); }
  public static void main(String[] args) throws Exception {
    String source = null;
    if (args.length == 2) {
      if (!args[0].equals("-comm")) {
        usage();
        System.exit(1);
      }
      source = args[1];
    }
    else if (args.length != 0) {
      usage();
      System.exit(1);
    }
    PhoenixSource phoenix;
    if (source == null) {
      phoenix = BuildSource.makePhoenix(PrintStreamMessenger.err);
    }
    else {
      phoenix = BuildSource.makePhoenix(source, PrintStreamMessenger.err);
    }
    MoteIF mif = new MoteIF(phoenix);
    PrintSerial serial = new PrintSerial(mif);
  }
}
```

Figure 4.10: PrintSerial.java: Java client program for PrintSerial

```
/**
 * PrintSerial - A TinyOS application that shows the concept of
 *               network debugging.
 *
 *           +---------+
 *           |  MainC  |
 *           +---------+
 *                | init
 *                |
 *   +-----------------------------------------------+
 *   |           PrintSerialC                        |
 *   +-----------------------------------------------+
 *    |Boot   |Leds    |Notify      |MilliTimer   | Control
 *    |       |        |            |             | AMSend, Packet
 *    |       |        |            |             |
 *    |Boot   |Leds    |Notify      |Timer        | SplitControl
 *    |       |        |            |             | AMSend, Packet
 *  +-----+ +-----+ +-----------+ +-----------+ +--------------------+
 *  |MainC| |LedsC| |UserButtonC| |TimerMilliC| |SerialActiveMessageC|
 *  +-----+ +-----+ +-----------+ +-----------+ +--------------------+
 *
 */
#include "PrintSerial.h"
configuration PrintSerialAppC {}
implementation {
  components PrintSerialC as App, LedsC, MainC;
  components SerialActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  App.Boot -> MainC.Boot;
  App.Control -> AM;
  App.AMSend -> AM.AMSend[AM_PRINT_SERIAL_MSG];
  App.Leds -> LedsC;
  App.MilliTimer -> TimerMilliC;
  App.Packet -> AM;
  App.Notify -> UserButtonC;
}
```

Figure 4.11: PrintSerialAppC.nc: Configuration module for PrintSerial

```
#include <UserButton.h>
#include "Timer.h"
#include "PrintSerial.h"
module PrintSerialC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE;
  char m_data[TOSH_DATA_LENGTH] = "Hello 0\n";
  uint8_t c = 0;
  event void Boot.booted() {
    call Control.start();  call Notify.enable();  call MilliTimer.startPeriodic(1024);
  }
  event void MilliTimer.fired() { call Leds.led1Toggle(); }
  void serial_print(char *str, uint8_t len) {
    if (locked) { return; }
    else {
      uint8_t i;
      print_serial_msg_t* rcm =
        (print_serial_msg_t*) call Packet.getPayload(&packet, NULL);
      if (call Packet.maxPayloadLength() < len) {
      return;
      }
      for (i = 0; i < len; i++) { rcm->buffer[i] = m_data[i]; }
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
                           sizeof(print_serial_msg_t)) == SUCCESS) {
      locked = TRUE;
      }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) { locked = FALSE; }
  }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) {
    if (state == BUTTON_PRESSED) {
      call Leds.led0Toggle();
      c = (c==9) ? 0 : c+1;
      m_data[6] = '0' + c;
      serial_print(m_data, sizeof(m_data));
    }
  }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```

Figure 4.12: PrintSerialC.nc: Implementation module for PrintSerial

4.2.4 Project 4: Single and Dual

- **Single:** A TinyOS application that shows the concept of sending.
- **Behavior:** On boot, a timer of 1s period is started. Each time the timer is fired, sampling of the node voltage is requested. When the node voltage reading is available, this program sends the voltage reading over serial port. This program also allows sending an error message and the temperature reading (to be used at 'Dual' application).

This program samples only node voltage. If you want to extend this program to sample node internal temperature as well, wire DemoTemperatureSensorC.nc module to configuration module (SingleAppC.nc).
In order to see the debugging message, plug the mote to a serial port and run the listen client by typing "source ./run.sh". The listen client reads a message from the serial port and prints it on console.

- **Concepts Illustrated:**
  - ADC and split-phase - TEP101, Tinyos 2.0 tutorial lesson 5
  - Mote-PC serial communication - TEP113, Tinyos 2.0 tutorial lesson 4
- **Key Points:**
  - Sampling sensor value is handled in a split transaction in TinyOS. At first step, the application requests the sampling by calling read(). When sampling is done, the system notifies the sensor reading as readDone() event.
  - When sampling multiple sensor readings, cascade the sensing request and event processing.

```
COMPONENT=SingleAppC
BUILD_EXTRA_DEPS += PrintReading.class
CLEAN_EXTRA = *.class PrintReadingMsg.java
PrintReading.class: $(wildcard *.java) PrintReadingMsg.java
  javac *.java
PrintReadingMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=PrintReadingMsg PrintReading.h
print_reading_msg -o $@
include $(MAKERULES)
```

Figure 4.13: Makefile: Makefile for Single

```
java PrintReading -comm serial@/dev/ttyUSB0:telos
```

Figure 4.14: run.sh: Shell script to run Java client

```
import java.io.IOException;
import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;
public class PrintReading implements MessageListener {
  public static short FLAG_BUFFER = 0x01;
  public static short FLAG_VOLTAGE_READING = 0x02;
  public static short FLAG_TEMPERATURE_READING = 0x04;
  private MoteIF moteIF;
  public PrintReading(MoteIF moteIF) {
    this.moteIF = moteIF;
    this.moteIF.registerListener(new PrintReadingMsg(), this);
  }
  public void messageReceived(int to, Message message) {
    PrintReadingMsg msg = (PrintReadingMsg)message;
    if ((msg.get_flag() & FLAG_VOLTAGE_READING) != 0) {
      System.out.print("Voltage: " + msg.get_voltage_reading() + " ");
    }
    if ((msg.get_flag() & FLAG_TEMPERATURE_READING) != 0) {
      System.out.print("Temperature: " + msg.get_temperature_reading() + " ");
    }
    if ((msg.get_flag() & FLAG_BUFFER) != 0) {
      for (int i = 0; i < msg.DEFAULT_MESSAGE_SIZE; i++) {
        char character = (char) msg.getElement_buffer(i);
        if (character == 0) {
          break;
        }
        System.out.print(character);
      }
    }
    System.out.print("\n");
  }
  private static void usage() {
    System.err.println("usage: PrintReading [-comm <source>]");
  }
  public static void main(String[] args) throws Exception {
    String source = null;
    if (args.length == 2) {
      if (!args[0].equals("-comm")) { usage(); System.exit(1); }
      source = args[1];
    }
    else if (args.length != 0) { usage(); System.exit(1); }
    PhoenixSource phoenix;
    if (source == null) { phoenix = BuildSource.makePhoenix(PrintStreamMessenger.err); }
    else { phoenix = BuildSource.makePhoenix(source, PrintStreamMessenger.err); }
    MoteIF mif = new MoteIF(phoenix);
    PrintReading serial = new PrintReading(mif);
  }
}
```

Figure 4.15: PrintReading.java: java client program

```
#include "message.h"
typedef nx_struct print_reading_msg {
  nx_uint8_t flag;
  nx_uint8_t buffer[TOSH_DATA_LENGTH - 5];
  nx_uint16_t voltage_reading;
  nx_uint16_t temperature_reading;
} print_reading_msg_t;
enum {
  AM_PRINT_READING_MSG = 0x0B,
  FLAG_BUFFER           = 0x01,
  FLAG_VOLTAGE_READING  = 0x02,
  FLAG_TEMPERATURE_READING = 0X04,
};
```

Figure 4.16: PrintReading.h: Header file for Single

```
/**
 * Single - A TinyOS application that shows the concept of sending.
 *
 *           +---------+
 *           |  MainC  |
 *           +---------+
 *                | init
 *                |
 *   +-----------------------------------------------+
 *   |           SingleC                             |
 *   +-----------------------------------------------+
 *    |Boot |   |Leds    |   | Control            |
 *    |     |   |        |   | AMSend, Packet     |
 *    |     |   |        |   |                    |
 *    |Boot |   |Leds    |   | SplitControl       |
 *    |     |   |        |   | AMSend, Packet     |
 *  +-----+ | +-----+    | +--------------------+ |
 *  |MainC| | |LedsC|    | |SerialActiveMessageC| |
 *  +-----+ | +-----+    | +--------------------+ |
 *          |            |                        |
 *          |Notify      |MilliTimer              | ReadVoltage
 *          |            |                        |
 *          |            |                        |
 *          |Notify      |Timer                   | Read
 *          |            |                        |
 *       +-----------+ +-----------+           +-----------+
 *       |UserButtonC| |TimerMilliC|           |DemoSensorC|
 *       +-----------+ +-----------+           +-----------+
 */
#include "PrintReading.h"
configuration SingleAppC {}
implementation {
  components SingleC, LedsC, MainC;
  components SerialActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  SingleC.Boot -> MainC.Boot;
  SingleC.Control -> AM;
  SingleC.AMSend -> AM.AMSend[AM_PRINT_READING_MSG];
  SingleC.Leds -> LedsC;
  SingleC.MilliTimer -> TimerMilliC;
  SingleC.Packet -> AM;
  SingleC.Notify -> UserButtonC;
  SingleC.ReadVoltage -> VoltageSensor.Read;
}
```

Figure 4.17: SingleAppC.nc: Configuration module for Single

```
#include <UserButton.h>
#include "Timer.h"
#include "PrintReading.h"
module SingleC {
  uses { interface SplitControl as Control;  interface Leds;  interface Boot;
         interface AMSend;  interface Timer<TMilli> as MilliTimer;
         interface Packet;  interface Notify<button_state_t>;
         interface Read<uint16_t> as ReadVoltage;
  }
}
implementation {
  message_t packet;  bool locked = FALSE;   uint8_t flag = 0;
  char m_error[13] = "Sample Error\n";  char m_data[TOSH_DATA_LENGTH - 5];
  uint16_t voltage_reading = 0, temperature_reading = 0;
  event void Boot.booted() {
    call Control.start();  call Notify.enable();  call MilliTimer.startPeriodic(1024);
  }
  void serial_print() {
    if (locked) { return; }
    else {
      uint8_t i;
      print_reading_msg_t* rcm = (print_reading_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->flag = flag;
      if (flag & FLAG_BUFFER) {
        for (i = 0; i < TOSH_DATA_LENGTH - 5; i++) { rcm->buffer[i] = m_data[i]; }
      }
      if (flag & FLAG_VOLTAGE_READING) { rcm->voltage_reading = voltage_reading; }
      if (flag & FLAG_TEMPERATURE_READING) { rcm->temperature_reading = temperature_reading; }
      flag = 0;
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
          sizeof(print_reading_msg_t)) == SUCCESS) {
        locked = TRUE;
      }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) { locked = FALSE; }
  }
  void report_problem() {
    uint8_t i;  flag = FLAG_BUFFER;
    for (i = 0; i < sizeof(m_error); i++) { m_data[i] = m_error[i]; }
    m_data[i] = 0;  serial_print();
  }
  void report_voltage_reading(uint16_t data) {
    flag = FLAG_VOLTAGE_READING;  voltage_reading = data;  serial_print();
  }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result != SUCCESS) { report_problem(); }
    report_voltage_reading(data);
  }
  event void MilliTimer.fired() {
    call Leds.led1Toggle();  if (call ReadVoltage.read() != SUCCESS) report_problem();
  }
  event void Notify.notify( button_state_t state ) { }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```

Figure 4.18: SingleC.nc: Implementation module for Single

```
generic configuration TemperatureC() {
  provides interface Read<uint16_t>;
}
implementation {
  components new Msp430InternalTemperatureC();
  Read = Msp430InternalTemperatureC.Read;
}
```

Figure 4.19: TemperatureC.nc: Wrapper for Telosb internal temperature sensor

```
generic configuration DemoTemperatureSensorC()
{
  provides interface Read<uint16_t>;
}
implementation
{
  components new TemperatureC() as DemoTemperatureSensor;
  Read = DemoTemperatureSensor;
}
```

Figure 4.20: DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor

4.2.5 Project 5: Raw and Smooth

- **Raw:** A TinyOS application that shows the concept of task.
- **Behavior:** This program samples node voltage whenever timer is fired. When `MAX_READINGS` samples are collected, this program report the samples over the serial port. The sampled data can be further processed in a task. 'Smooth' application extends this application as follows:

  1. calculates the statistics: 'Smooth' calculates the minumum, maximum and mean for sampled data in `raw_reading[]`.
  2. smoothens the sampled data: 'Smooth' calculates the exponential moving average of `raw_reading[]` into `smooth_reading[]`.

  In order to see the debugging message, plug the mote to a serial port and run the listen client by typing 'source ./run.sh'. The listen client reads a message from the serial port and prints it on console.

- **Concepts Illustrated:**
  - Schedulers and Tasks - TEP106, Tinyos 2.0 tutorial lesson 2
  - ADC and split-phase - TEP101, Tinyos 2.0 tutorial lesson 5
  - Mote-PC serial communication - TEP113, Tinyos 2.0 tutorial lesson 4
- **Key Points:** Calculating statistics and exponential moving average can take time. It is recommended to process a time consuming job in a task rather than process directly in the event handler.

```
COMPONENT=RawAppC
BUILD_EXTRA_DEPS += PrintReadingArr.class
CLEAN_EXTRA = *.class PrintReadingArrMsg.java
PrintReadingArr.class: $(wildcard *.java) PrintReadingArrMsg.java
  javac *.java
PrintReadingArrMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=PrintReadingArrMsg PrintReadingArr.h
print_reading_arr_msg -o $@
include $(MAKERULES)
```

Figure 4.21: Makefile: Makefile for Raw

```
java PrintReadingArr -comm serial@/dev/ttyUSB0:telos
```

Figure 4.22: run.sh: Shell script to run Java client

```
import java.io.IOException;
import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;
public class PrintReadingArr implements MessageListener {
  public static short FLAG_BUFFER = 0x01;
  public static short FLAG_VOLTAGE_READING = 0x02;
  public static short FLAG_TEMPERATURE_READING = 0x04;
  private MoteIF moteIF;
  public PrintReadingArr(MoteIF moteIF) {
    this.moteIF = moteIF;
    this.moteIF.registerListener(new PrintReadingArrMsg(), this);
  }
  public void messageReceived(int to, Message message) {
    PrintReadingArrMsg msg = (PrintReadingArrMsg)message;
    System.out.println("Node " + msg.get_nodeid() + " readings: ");
    System.out.print("Raw: ");
    for (int i = 0; i < msg.numElements_raw_reading(); i++) {
      System.out.print(msg.getElement_raw_reading(i) + " ");
    }
    System.out.print("\n");  System.out.print("Smooth: ");
    for (int i = 0; i < msg.numElements_smooth_reading(); i++) {
      System.out.print(msg.getElement_smooth_reading(i) + " ");
    }
    System.out.print("\n");
    System.out.println("Min: "  + msg.get_min() + " Max: " + msg.get_max() +
                       " Mean: " + msg.get_mean());
    System.out.print("\n");
  }
  private static void usage() { System.err.println("usage: PrintReadingArr [-comm <source>]"); }
  public static void main(String[] args) throws Exception {
    String source = null;
    if (args.length == 2) {
      if (!args[0].equals("-comm")) { usage(); System.exit(1); }
      source = args[1];
    }
    else if (args.length != 0) { usage(); System.exit(1); }
    PhoenixSource phoenix;
    if (source == null) { phoenix = BuildSource.makePhoenix(PrintStreamMessenger.err); }
    else { phoenix = BuildSource.makePhoenix(source, PrintStreamMessenger.err); }
    MoteIF mif = new MoteIF(phoenix);
    PrintReadingArr serial = new PrintReadingArr(mif);
  }
}
```

Figure 4.23: PrintReadingArr.java: Java client for Raw

```
#include "message.h"
enum {
  AM_PRINT_READING_ARR_MSG = 0x0C,
  MAX_READINGS = 5,
  SMOOTH_FACTOR = 3,
  DENOMINATOR = 10,
};
typedef nx_struct print_reading_arr_msg {
  nx_uint16_t nodeid;
  nx_uint16_t min;
  nx_uint16_t max;
  nx_uint16_t mean;
  nx_uint16_t raw_reading[MAX_READINGS];
  nx_uint16_t smooth_reading[MAX_READINGS];
} print_reading_arr_msg_t;
```

Figure 4.24: PrintReadingArr.h: Header file for Raw

```
/**
 * Raw - A TinyOS application that shows the concept of task.
 *
 *           +---------+
 *           |  MainC  |
 *           +---------+
 *                | init
 *                |
 *   +-----------------------------------------------+
 *   |           RawC                                |
 *   +-----------------------------------------------+
 *    |Boot |   |Leds    |   | Control            |
 *    |     |   |        |   | AMSend, Packet     |
 *    |     |   |        |   |                    |
 *    |Boot |   |Leds    |   | SplitControl       |
 *    |     |   |        |   | AMSend, Packet     |
 *  +-----+ | +-----+    | +--------------------+ |
 *  |MainC| | |LedsC|    | |SerialActiveMessageC| |
 *  +-----+ | +-----+    | +--------------------+ |
 *          |            |                        |
 *          |Notify      |MilliTimer              | ReadVoltage
 *          |            |                        |
 *          |            |                        |
 *          |Notify      |Timer                   | Read
 *          |            |                        |
 *       +-----------+ +-----------+           +-----------+
 *       |UserButtonC| |TimerMilliC|           |DemoSensorC|
 *       +-----------+ +-----------+           +-----------+
 */
#include "PrintReadingArr.h"
configuration RawAppC {}
implementation {
  components RawC, LedsC, MainC;
  components SerialActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  RawC.Boot -> MainC.Boot;
  RawC.Control -> AM;
  RawC.AMSend -> AM.AMSend[AM_PRINT_READING_ARR_MSG];
  RawC.Leds -> LedsC;
  RawC.MilliTimer -> TimerMilliC;
  RawC.Packet -> AM;
  RawC.Notify -> UserButtonC;
  RawC.ReadVoltage -> VoltageSensor.Read;
}
```

Figure 4.25: RawAppC.nc: Configuration module for Raw

```
#include <UserButton.h>
#include "Timer.h"
#include "PrintReadingArr.h"
module RawC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
    interface Read<uint16_t> as ReadVoltage;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE;  uint8_t counter = 0;
  uint16_t min;  uint16_t max;  uint16_t mean;
  uint16_t raw_reading[MAX_READINGS];
  uint16_t smooth_reading[MAX_READINGS];
  event void Boot.booted() {
    call Control.start();  call Notify.enable();  call MilliTimer.startPeriodic(250);
  }
  void serial_print() {
    if (locked) { return; }
    else {
      uint8_t i;
      print_reading_arr_msg_t* rcm = (print_reading_arr_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->nodeid = TOS_NODE_ID;
      rcm->min = min;
      rcm->max = max;
      rcm->mean = mean;
      for (i = 0; i < MAX_READINGS; i++) { rcm->raw_reading[i] = raw_reading[i]; }
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(print_reading_arr_msg_t)) == SUCCESS) {
        locked = TRUE;
      }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) { locked = FALSE; }
  }
  uint16_t smoothen(uint16_t prev_reading, uint16_t new_reading) {
    return (prev_reading * (DENOMINATOR - SMOOTH_FACTOR) + new_reading * SMOOTH_FACTOR) / DENOMINATOR;
  }
  task void process_voltage_reading() {
    int i;
    uint16_t temp_min = raw_reading[0];
    uint16_t temp_max = raw_reading[0];
    uint16_t temp_sum = raw_reading[0];
    for (i = 1; i < MAX_READINGS; i++) {
      temp_sum += raw_reading[i];
      if (raw_reading[i] < temp_min) { temp_min = raw_reading[i]; }
      if (raw_reading[i] > temp_max) { temp_max = raw_reading[i]; }
    }
    min = temp_min;
    max = temp_max;
    mean = temp_sum / MAX_READINGS;
    serial_print();
  }
  void store_voltage_reading(uint16_t data) {
    raw_reading[counter++] = data;
    if (counter == MAX_READINGS) { post process_voltage_reading(); counter = 0; }
  }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { store_voltage_reading(data); }
  }
  event void MilliTimer.fired() { call Leds.led1Toggle();  call ReadVoltage.read(); }
  event void Notify.notify( button_state_t state ) { }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```


Figure 4.26: RawC.nc: Implementation module for Raw

4.2.6 Project 6: Counts and Readings

- **Counts:** A TinyOS application that shows the concept of sending a radio message.
- **Behavior:** 'Counts' program starts a timer at an interval of 1s. Each time the timer is triggered, this program sends a radio message that contains its node ID and counter variable. The counter variable is incremented each time the timer is triggered.
  'Readings' extends this program by sampling the node voltage as well. In order to see the debugging message, a base station node needs be prepared. A base station node is a node that is programmed with 'BaseStationCC2420' program, which forwards all its received radio messages to and from the serial port.
  Try program multiple nodes with 'Counts' program and see the behavior with a base station node and java client application. You can see the debugging messages from multiple sensor nodes.
- **Concepts Illustrated:**
  Mote-mote communication - TEP111, TEP116, Tinyos 2.0 tutorial lesson 3
- **Key Points:** Wire DemoSensorC to the application and get the sensor reading using the previous lessons.

```
COMPONENT=CountsAppC
BUILD_EXTRA_DEPS += PrintCounterReading.class
CLEAN_EXTRA = *.class PrintCounterReadingMsg.java
PrintCounterReading.class: $(wildcard *.java) PrintCounterReadingMsg.java
  javac *.java
PrintCounterReadingMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=PrintCounterReadingMsg PrintCounterReading.h
print_counter_reading_msg -o $@
include $(MAKERULES)
```

Figure 4.27: Makefile: Makefile for Counts

```
java PrintCounterReading -comm serial@/dev/ttyUSB0:telos
```

Figure 4.28: run.sh: Shell script to run Java client

```
import java.io.IOException;
import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;
public class PrintCounterReading implements MessageListener {
  public static short FLAG_COUNTER = 0x01;
  public static short FLAG_VOLTAGE_READING = 0x02;
  private MoteIF moteIF;
  public PrintCounterReading(MoteIF moteIF) {
    this.moteIF = moteIF;
    this.moteIF.registerListener(new PrintCounterReadingMsg(), this);
  }
  public void messageReceived(int to, Message message) {
    PrintCounterReadingMsg msg = (PrintCounterReadingMsg)message;
    System.out.print("Node ID: " + msg.get_nodeid() + " ");
    if ((msg.get_flag() & FLAG_COUNTER) != 0) {
      System.out.print("Counter: " + msg.get_counter() + " ");
    }
    if ((msg.get_flag() & FLAG_VOLTAGE_READING) != 0) {
      System.out.print("Voltage: " + msg.get_voltage_reading() + " ");
    }
    System.out.print("\n");
  }
  private static void usage() {
    System.err.println("usage: PrintCounterReading [-comm <source>]");
  }
  public static void main(String[] args) throws Exception {
    String source = null;
    if (args.length == 2) {
      if (!args[0].equals("-comm")) { usage(); System.exit(1); }
      source = args[1];
    }
    else if (args.length != 0) { usage(); System.exit(1); }
    PhoenixSource phoenix;
    if (source == null) { phoenix = BuildSource.makePhoenix(PrintStreamMessenger.err); }
    else { phoenix = BuildSource.makePhoenix(source, PrintStreamMessenger.err); }
    MoteIF mif = new MoteIF(phoenix);
    PrintCounterReading serial = new PrintCounterReading(mif);
  }
}
```

Figure 4.29: PrintCounterReading.java: java Client for Counts

```
#include "message.h"
typedef nx_struct print_counter_reading_msg {
  nx_uint8_t flag;
  nx_uint16_t nodeid;
  nx_uint16_t counter;
  nx_uint16_t voltage_reading;
} print_counter_reading_msg_t;
enum {
  AM_PRINT_COUNTER_READING_MSG = 0x0D,
  FLAG_COUNTER = 0x01,
  FLAG_VOLTAGE_READING  = 0x02,
};
```

Figure 4.30: PrintCounterReading.h: Header file for Counts

```
/**
 * Counts - A TinyOS application that shows the concept of sending
 *          a radio message.
 *
 *           +---------+
 *           |  MainC  |
 *           +---------+
 *                | init
 *                |
 *   +-----------------------------------------------+
 *   |           CountsC                             |
 *   +-----------------------------------------------+
 *    |Boot |   |Leds    |   | Control            |
 *    |     |   |        |   | AMSend, Packet     |
 *    |     |   |        |   |                    |
 *    |Boot |   |Leds    |   | SplitControl       |
 *    |     |   |        |   | AMSend, Packet     |
 *  +-----+ | +-----+    | +--------------+       |
 *  |MainC| | |LedsC|    | |ActiveMessageC|       |
 *  +-----+ | +-----+    | +--------------+       |
 *          |            |                        |
 *          |Notify      |MilliTimer              | ReadVoltage
 *          |            |                        |
 *          |            |                        |
 *          |Notify      |Timer                   | Read
 *          |            |                        |
 *       +-----------+ +-----------+           +-----------+
 *       |UserButtonC| |TimerMilliC|           |DemoSensorC|
 *       +-----------+ +-----------+           +-----------+
 */
#include "PrintCounterReading.h"
configuration CountsAppC {}
implementation {
  components CountsC, LedsC, MainC;
  components ActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  CountsC.Boot -> MainC.Boot;
  CountsC.Control -> AM;
  CountsC.AMSend -> AM.AMSend[AM_PRINT_COUNTER_READING_MSG];
  CountsC.Leds -> LedsC;
  CountsC.MilliTimer -> TimerMilliC;
  CountsC.Packet -> AM;
  CountsC.Notify -> UserButtonC;
  CountsC.ReadVoltage -> VoltageSensor.Read;
}
```

Figure 4.31: CountsAppC.nc: Configuration module for Counts

```
#include <UserButton.h>
#include "Timer.h"
#include "PrintCounterReading.h"
module CountsC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
    interface Read<uint16_t> as ReadVoltage;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE;  uint8_t flag = 0;
  uint16_t counter = 0;  uint16_t voltage_reading = 0;
  event void Boot.booted() {
    call Control.start();  call Notify.enable();  call MilliTimer.startPeriodic(1024);
  }
  void serial_print() {
    if (locked) { return; }
    else {
      print_counter_reading_msg_t* rcm =
        (print_counter_reading_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->flag = flag;
      rcm->nodeid = TOS_NODE_ID;
      if (flag & FLAG_COUNTER) { rcm->counter = counter; }
      if (flag & FLAG_VOLTAGE_READING) { rcm->voltage_reading = voltage_reading; }
      flag = 0;
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
        sizeof(print_counter_reading_msg_t)) == SUCCESS) {
        locked = TRUE;
      }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) { locked = FALSE; }
  }
  void report_voltage_reading(uint16_t data) {
    flag |= FLAG_VOLTAGE_READING; voltage_reading = data;
  }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { report_voltage_reading(data); }
  }
  event void MilliTimer.fired() {
    call Leds.led1Toggle();
    counter++;
    flag |= FLAG_COUNTER;
    serial_print();
  }
  event void Notify.notify( button_state_t state ) { }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```

Figure 4.32: CountsC.nc: Implementation module for Counts

4.2.7 Project 7: Request and RequestSample

- **Request:** A TinyOS application that shows the concept of receiving a radio message.
- **Behavior:** When 'Request' program receives a radio message Receive.receive() event is triggered. Depending on the contents of the received message, the program can take an appropriate action. Our

program simply fills in the reply message with the node id and send it.
The exercise in this lesson is to extend this simple version of program so that it fills the contents of the reply message in response to the user request. The request message can choose any combinations of counter, voltage reading and temperature reading. For this, you need to parse the received message in Receive.receive().
In order to send a message and receive the reply message, a base station needs be prepared. A base station node is a node that is programmed with 'BaseStationCC2420' program, which forwards all its received radio messages to and from the serial port.
Try program multiple nodes with 'Request' program and see the behavior with a base station node and java client application. You can see the debugging messages from multiple sensor nodes.

- **Key Points:** In order to take appropriate actions to the request message, you need to parse the incoming message in Receive,receive() event handler.

```
COMPONENT=RequestAppC
BUILD_EXTRA_DEPS += RequestReading.class
CLEAN_EXTRA = *.class RequestReadingMsg.java
RequestReading.class: $(wildcard *.java) RequestReadingMsg.java
  javac *.java
RequestReadingMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=RequestReadingMsg RequestReading.h
request_reading_msg -o $@
include $(MAKERULES)
```

Figure 4.33: Makefile: Makefile for Request

```
java RequestReading -comm serial@/dev/ttyUSB0:telos
```

Figure 4.34: run.sh: Shell script to run the Java client

```
import java.io.IOException;  import net.tinyos.message.*;  import net.tinyos.packet.*;
import net.tinyos.util.*;  import java.util.*;  import java.io.*;
import java.net.*;  import javax.swing.*;
public class RequestReading implements MessageListener {
  public static final short FLAG_COUNTER = 0x01;
  public static final short FLAG_VOLTAGE_READING = 0x02;
  public static final short FLAG_TEMPERATURE_READING = 0x04;
  public static final short FLAG_TEXT = 0x08;
  public static boolean bCounter = true;
  public static boolean bVoltage = false;
  public static boolean bTemperature = false;
  private static MoteIF moteIF;
  private static ClientWindow clntWindow = null;
  public RequestReading(MoteIF moteIF, String nodeid, short flag) {
    this.moteIF = moteIF;  this.moteIF.registerListener(new RequestReadingMsg(), this);
  }
  public static void sendRequest(short nodeid) {
    RequestReadingMsg payload = new RequestReadingMsg();
    try {
      System.out.println("Sending packet to " + nodeid);
      payload.set_nodeid( nodeid );
      short flag = 0;
      if (bCounter) flag |= FLAG_COUNTER;
      if (bVoltage) flag |= FLAG_VOLTAGE_READING;
      if (bTemperature) flag |= FLAG_TEMPERATURE_READING;
      payload.set_flag( flag );
      moteIF.send(nodeid, payload);
    } catch (IOException exception) {
      System.err.println("Exception thrown when sending packets. Exiting.");  System.err.println(exception);
    }
  }
  public void messageReceived(int to, Message message) {
    RequestReadingMsg msg = (RequestReadingMsg)message;
    String strOut = "Node ID: " + msg.get_nodeid() + " ";
    if ((msg.get_flag() & FLAG_COUNTER) != 0) { strOut += "Counter: " + msg.get_counter() + " "; }
    if ((msg.get_flag() & FLAG_VOLTAGE_READING) != 0) { strOut += "Voltage: " + msg.get_voltage_reading() + " "; }
    if ((msg.get_flag() & FLAG_TEMPERATURE_READING) != 0)
      { strOut += "Temperature: " + msg.get_voltage_reading() + " "; }
    strOut += "\n";  clntWindow.printMessage(strOut);
  }
  private static void usage() {
    System.err.println(
      "usage: RequestReading [-comm <source>] [-v] [-t] [-c <counter_value>] [-m <text_message>] [-n <node_id>]");
    System.err.println("-v: get voltage reading");
    System.err.println("-t: get temperature reading");
    System.err.println("-n: set nodeid. Set to broadcast address if not specified.");
  }
  public static void main(String[] args) throws Exception {
    String source = null;  String nodeid = null;
    short flag = 0;  int i = 0;
    while (i < args.length) {
      if (args[i].equals("-comm")) { source = args[i+1]; i += 2; }
      else if (args[i].equals("-v")) { flag |= FLAG_VOLTAGE_READING;    i++; }
      else if (args[i].equals("-t")) { flag |= FLAG_TEMPERATURE_READING; i++; }
      else if (args[i].equals("-n")) { nodeid = args[i+1]; i += 2; }
      else { usage(); System.exit(1); }
    }
    PhoenixSource phoenix;
    if (source == null) { phoenix = BuildSource.makePhoenix(PrintStreamMessenger.err); }
    else { phoenix = BuildSource.makePhoenix(source, PrintStreamMessenger.err); }
    MoteIF mif = new MoteIF(phoenix);
    RequestReading serial = new RequestReading(mif, nodeid, flag );
    CreateGui();
  }
  private static void CreateGui ( )
  {
    JFrame clientFrame = new JFrame("RequestReading");
    clntWindow = new ClientWindow ( clientFrame );
    clientFrame.addWindowListener( clntWindow );  clientFrame.setSize( clntWindow.getPreferredSize() );
    clientFrame.getContentPane().add("Center", clntWindow);  clientFrame.show();
  }
}
```


Figure 4.35: RequestReading.java: Java client for Request

```
import java.awt.*;  import javax.swing.*;  import java.awt.event.*;
import java.sql.Time;  import java.sql.Date;  import java.sql.Timestamp;
public class ClientWindow extends JPanel implements WindowListener
{
  JFrame        myFrame       = null;
  final short DEFAULT_NODE_ID = 0;
  RequestReading listener  = null;
  JScrollPane   msgPanel      = new JScrollPane();
  JTextArea     msgWndw       = new JTextArea();
  JPanel        panelButton   = new JPanel();
  JCheckBox     cbCounter     = new JCheckBox();
  JCheckBox     cbVoltage     = new JCheckBox();
  JCheckBox     cbTemperature = new JCheckBox();
  JLabel        labelNodeid   = new JLabel();
  JTextField    fieldNodeid   = new JTextField();
  JButton       bSendRequest  = new JButton();
  ActionListener cbal = new ActionListener() { public void actionPerformed(ActionEvent e) { updateGlobals(); } };
  void updateGlobals() {
    RequestReading.bCounter = cbCounter.isSelected();  RequestReading.bVoltage = cbVoltage.isSelected();
    RequestReading.bTemperature = cbTemperature.isSelected();
  }
  public ClientWindow(JFrame frame) {
    try { myFrame = frame; jbInit(); }  catch(Exception e) { e.printStackTrace(); }
  }
  private void jbInit() throws Exception {
    this.setLayout(new BorderLayout());
    this.setMinimumSize(new Dimension(200, 400));  this.setPreferredSize(new Dimension(600, 400));
    msgPanel.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    msgPanel.setAutoscrolls(true);
    msgWndw.setFont(new java.awt.Font("Courier", 0, 12));
    msgPanel.getViewport().add(msgWndw, null);
    panelButton.setLayout(new GridLayout(7, 1) );
    panelButton.setMinimumSize(new Dimension(100, 170));
    panelButton.setPreferredSize(new Dimension(160, 170));
    cbCounter.setSelected(RequestReading.bCounter);
    cbCounter.setFont(new java.awt.Font("Dialog", 1, 10)); cbCounter.setText("Read Counter");
    cbCounter.addActionListener(cbal);
    cbVoltage.setSelected(RequestReading.bVoltage);
    cbVoltage.setFont(new java.awt.Font("Dialog", 1, 10)); cbVoltage.setText("Read Voltage");
    cbVoltage.addActionListener(cbal);
    cbTemperature.setSelected(RequestReading.bTemperature);
    cbTemperature.setFont(new java.awt.Font("Dialog", 1, 10)); cbTemperature.setText("Read Temperature");
    cbTemperature.addActionListener(cbal);
    labelNodeid.setFont(new java.awt.Font("Dialog", 1, 10)); labelNodeid.setText("node id:");
    fieldNodeid.setFont(new java.awt.Font("Dialog", 1, 10)); fieldNodeid.setText(Short.toString(DEFAULT_NODE_ID));
    panelButton.add(cbCounter, null); panelButton.add(cbVoltage, null);
    panelButton.add(cbTemperature, null); panelButton.add(labelNodeid, null);
    panelButton.add(fieldNodeid, null); panelButton.add(bSendRequest, null);
    bSendRequest.addActionListener(new java.awt.event.ActionListener() {
       public void actionPerformed(ActionEvent e) { bSendRequest_actionPerformed(e); }
    });
    bSendRequest.setFont(new java.awt.Font("Dialog", 1, 10)); bSendRequest.setText("Send Request");
    this.add(msgPanel, BorderLayout.CENTER);
    this.add(panelButton, BorderLayout.EAST);
  }
  public synchronized void windowClosing ( WindowEvent e ) { System.exit(1); }
  public void windowClosed      ( WindowEvent e ) { }  public void windowActivated   ( WindowEvent e ) { }
  public void windowIconified   ( WindowEvent e ) { }  public void windowDeactivated ( WindowEvent e ) { }
  public void windowDeiconified ( WindowEvent e ) { }  public void windowOpened      ( WindowEvent e ) { }
  public short getNodeid() {
    short n;
    try { n = Short.parseShort( fieldNodeid.getText() ); }
    catch (NumberFormatException exception) { fieldNodeid.setText("0"); n = 0; }
    return n;
  }
  void bSendRequest_actionPerformed(ActionEvent e) { RequestReading.sendRequest(getNodeid()); }
  public void printMessage(String str) { msgWndw.append(str); }
}
```

Figure 4.36: ClientWindow.java: Java GUI for Request

```
#include "message.h"
enum {
  AM_REQUEST_READING_MSG = 0x0F,
  FLAG_COUNTER = 0x01,
  FLAG_VOLTAGE_READING  = 0x02,
  FLAG_TEMPERATURE_READING  = 0x04,
  FLAG_TEXT = 0x08,
  MAX_TEXT_LENGTH = 18,
};
typedef nx_struct request_reading_msg {
  nx_uint8_t flag;
  nx_uint16_t nodeid;
  nx_uint16_t counter;
  nx_uint16_t voltage_reading;
  nx_uint16_t temperature_reading;
  nx_uint8_t buffer[MAX_TEXT_LENGTH];
} request_reading_msg_t;
```

Figure 4.37: RequestReading.h: Header file for Request

```
generic configuration TemperatureC() {
  provides interface Read<uint16_t>;
}
implementation {
  components new Msp430InternalTemperatureC();
  Read = Msp430InternalTemperatureC.Read;
}
```

Figure 4.38: TemperatureC.nc: Wrapper for Telosb internal temperature sensor

```
generic configuration DemoTemperatureSensorC()
{
  provides interface Read<uint16_t>;
}
implementation
{
  components new TemperatureC() as DemoTemperatureSensor;
  Read = DemoTemperatureSensor;
}
```

Figure 4.39: DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor

```
/**
 * Request - A TinyOS application that shows the concept of receiving
 *           a radio message.
 */
#include "RequestReading.h"
configuration RequestAppC {}
implementation {
  components RequestC, LedsC, MainC;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  components new DemoTemperatureSensorC() as TemperatureSensor;
  components ActiveMessageC as AM;
  components new AMSenderC(AM_REQUEST_READING_MSG);
  components new AMReceiverC(AM_REQUEST_READING_MSG);
  RequestC.Boot -> MainC.Boot;
  RequestC.Control -> AM;
  RequestC.AMSend -> AMSenderC;
  RequestC.Receive -> AMReceiverC;
  RequestC.Leds -> LedsC;
  RequestC.MilliTimer -> TimerMilliC;
  RequestC.Packet -> AM;
  RequestC.Notify -> UserButtonC;
  RequestC.ReadVoltage -> VoltageSensor.Read;
  RequestC.ReadTemperature -> TemperatureSensor.Read;
}
```

Figure 4.40: RequestAppC.nc: Configuration module for Request

```
#include <UserButton.h>
#include "Timer.h"
#include "RequestReading.h"
module RequestC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Receive;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
    interface Read<uint16_t> as ReadVoltage;
    interface Read<uint16_t> as ReadTemperature;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE;
  uint8_t flag = 0;
  uint16_t counter = 0;
  uint16_t voltage_reading = 0;
  uint16_t temperature_reading = 0;
  event void Boot.booted() {
    call Control.start();  call Notify.enable();  call MilliTimer.startPeriodic(1024);
  }
  task void sample_voltage() { call ReadVoltage.read(); }
  task void sample_temperature() { call ReadTemperature.read(); }
  task void reply_request() {
    if (locked) { return; }
    else {
      request_reading_msg_t* rcm = (request_reading_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->flag = flag;
      rcm->nodeid = TOS_NODE_ID;
      if (flag & FLAG_COUNTER) { rcm->counter = counter; }
      if (flag & FLAG_VOLTAGE_READING) { rcm->voltage_reading = voltage_reading; }
      if (flag & FLAG_TEMPERATURE_READING) { rcm->temperature_reading = temperature_reading; }
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
          sizeof(request_reading_msg_t)) == SUCCESS) { locked = TRUE; }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) { locked = FALSE; }
  }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { voltage_reading = data; post sample_temperature(); }
  }
  event void ReadTemperature.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { temperature_reading = data; post reply_request(); }
  }
  event void MilliTimer.fired() {
    call Leds.led1Toggle();
    counter++;
  }
  event void Notify.notify( button_state_t state ) {
    if (state == BUTTON_PRESSED) { call Leds.led0Toggle(); post sample_voltage(); }
  }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
  event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t len) {
    request_reading_msg_t* rcm = (request_reading_msg_t*)payload;
    call Leds.led2Toggle();
    if (rcm->nodeid == TOS_NODE_ID) { flag = 0; post reply_request(); }
    return bufPtr;
  }
}
```

Figure 4.41: RequestC.nc: Implementation module for Request

4.2.8 Project 8: CountToRadio and RadioToCount

- **CountToRadio:** A TinyOS application that shows the concept of sending a radio message.
- **Behavior:** 'CountToRadio' program starts a timer at an interval of 1s. Each time the timer is triggered, this program sends a radio message that contains its node ID and counter variable. The counter variable is incremented each time the timer is triggered. LED is set to last 3-bits of the counter value. The goal of this lesson is to write a receiver program 'RadioToCount' program that receives the counter message and sets its LED as last 3-bits of the receiverd counter value.
- **Concepts Illustrated:**
  Mote-mote communication - TEP111, TEP116, Tinyos 2.0 tutorial lesson 3

```
COMPONENT=CountToRadioAppC
include $(MAKERULES)
```

Figure 4.42: Makefile: Makefile for CountToRadio

```
#include "message.h"
typedef nx_struct counter_msg {
  nx_uint16_t nodeid;
  nx_uint16_t counter;
} counter_msg_t;
enum {
  AM_COUNTER_MSG = 0x10,
};
```

Figure 4.43: Counter.h: Header file for CountToRadio

```
/**
 * CountToRadio - A TinyOS application that shows the concept of sending
 *          a radio message.
 */
#include "Counter.h"
configuration CountToRadioAppC {}
implementation {
  components CountToRadioC, LedsC, MainC;
  components ActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  CountToRadioC.Boot -> MainC.Boot;
  CountToRadioC.Control -> AM;
  CountToRadioC.AMSend -> AM.AMSend[AM_COUNTER_MSG];
  CountToRadioC.Leds -> LedsC;
  CountToRadioC.MilliTimer -> TimerMilliC;
  CountToRadioC.Packet -> AM;
  CountToRadioC.Notify -> UserButtonC;
}
```

Figure 4.44: CountToRadioAppC.nc: Configuration module for CountToRadio

```
#include <UserButton.h>
#include "Timer.h"
#include "Counter.h"
module CountToRadioC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE;
  uint16_t counter = 0;
  event void Boot.booted() {
    call Control.start();
    call Notify.enable();
    call MilliTimer.startPeriodic(1024);
  }
  void send_counter() {
    if (locked) { return; }
    else {
      counter_msg_t* rcm = (counter_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->nodeid = TOS_NODE_ID;
      rcm->counter = counter;
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(counter_msg_t)) == SUCCESS) {
        locked = TRUE;
      }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) { locked = FALSE; }
  }
  event void MilliTimer.fired() {
    counter++;
    call Leds.set(counter & 0x07);
    send_counter();
  }
  event void Notify.notify( button_state_t state ) { }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```

Figure 4.45: CountToRadioC.nc: Implementation module for CountToRadio

Appendix A

SOAP Application Solution

A.1 SOAP Java Application

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class GetName
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    GetName(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(String name, String addr)
    {
        GW__MetaData_Result[] results;
        try {
            results = proxy.metadataGet(addr, name);
            for (int i = 0; i < results.length; i++) {
                System.out.println( "Mote " + results[i].getAddr() + " Name " + results[i].getValue() );
            }
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java GetName gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName = "name";
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        GetName get = new GetName(gateway);
        get.get(attrName, mote);
    }
}
```

Figure A.1: Solution code for Ex2_1.java

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class GetLocation
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    GetLocation(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(String name, String addr)
    {
        GW__MetaData_Result[] results;
        try {
            results = proxy.metadataGet(addr, name);
            for (int i = 0; i < results.length; i++) {
                System.out.println( "Mote " + results[i].getAddr() + " " + name + " " + results[i].getValue() );
            }
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java GetLocation gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        GetLocation get = new GetLocation(gateway);
        attrName = "map-x";
        get.get(attrName, mote);
        attrName = "map-y";
        get.get(attrName, mote);
    }
}
```

Figure A.2: Solution code for Ex2_2.java

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class GetLastOneDayTemperature
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    GetLastOneDayTemperature(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(double t1Offset, double t2Offset, String name, String addr)
    {
        GW__eventsReadRelative_Result result;
        GW__Event_Result[] results;
        GW__Event value;
        org.apache.axis.types.UnsignedInt offset = new org.apache.axis.types.UnsignedInt(0);
        try {
            result = proxy.eventsReadRelative(t1Offset, t2Offset, name, addr, offset, "");
            results = result.getResults();
            for (int i = 0; i < results.length; i++) {
                value = results[i].getValue();
                System.out.println( "Mote " + results[i].getAddr() +
                         " timestamp " + results[i].getTimestamp() +
                         " " + name + " " + value.getTemperatureReadEvent() );
            }
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java GetLastOneDayTemperature gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        GetLastOneDayTemperature get = new GetLastOneDayTemperature(gateway);
        double t1Offset = -86400.0;   // one day before
        double t2Offset = 0.0;        // now
        attrName = "TemperatureReadEvent";
        get.get(t1Offset, t2Offset, attrName, mote);
    }
}
```

Figure A.3: Solution code for Ex2_3.java

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class RpcSampleRequest
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    RpcSampleRequest(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void rpcExecute(String addr)
    {
        Void_Result[] results;
        try {
            results = proxy.rpcExecutesampleRequest(addr, TIMEOUT, "*");
            for (int i = 0; i < results.length; i++) {
                System.out.println("Mote " + results[i].getAddr() + " Time " + results[i].getTimestamp());
            }
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java RpcSampleRequest gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        RpcSampleRequest stub = new RpcSampleRequest(gateway);
        stub.rpcExecute(mote);
    }
}
```

Figure A.4: Solution code for Ex2_4.java

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class GetLastBatteryVoltage
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    GetLastBatteryVoltage(String gateway)
    {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(String name, String addr)
    {
        GW__eventsReadLast_Result result;
        GW__Event_Result[] results;
        GW__Event value;
        Nx_node_stats_t event;
        try {
            result = proxy.eventsReadLast(name, addr);
            results = result.getResults();
            for (int i = 0; i < results.length; i++) {
                value = results[i].getValue();
                event = value.getNodeStatsEvent();
                System.out.println( "Mote " + results[i].getAddr() +
                  " Time " + results[i].getTimestamp() + " Voltage " + event.getVoltage() );
            }
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args)
    {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java GetLastBatteryVoltage gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        GetLastBatteryVoltage get = new GetLastBatteryVoltage(gateway);
        attrName = "NodeStatsEvent";
        get.get(attrName, mote);
    }
}
```

Figure A.5: Solution code for Ex2_5.java

```
import java.lang.reflect.*;
import java.net.URL;
import gw.*;
public class GetLastOneDayReliability
{
    public static double TIMEOUT = 2.0;
    private GWPort proxy;
    GetLastOneDayReliability(String gateway) {
        try {
            URL portURL = new URL("http://" + gateway + "/gw/soap");
            GWServiceLocator locator = new GWServiceLocator();
            proxy = locator.getGWPort(portURL);
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public void get(double t1Offset, double t2Offset, String name, String addr) {
        GW__eventsReadRelative_Result result;
        GW__Event_Result[] results;
        GW__Event value;
        Nx_node_stats_t event;
        org.apache.axis.types.UnsignedInt offset = new org.apache.axis.types.UnsignedInt(0);
        try {
            result = proxy.eventsReadRelative(t1Offset, t2Offset, name, addr, offset, "");
            results = result.getResults();
            for (int i = 0; i < results.length; i++) {
                value = results[i].getValue();
                event = value.getNodeStatsEvent();
                System.out.println( "Mote " + results[i].getAddr() +
                                    " timestamp " + results[i].getTimestamp() +
                                    " sent " + event.getNet_sent() +
                                    " delivered " + event.getNet_delivered());
            }
        } catch (Exception e) { e.printStackTrace(); System.exit(2); }
    }
    public static void main(String [] args) {
        if (args.length != 1 && args.length != 2) {
            System.out.println("Usage: java GetLastOneDayReliability gateway [mote]"); System.exit(3);
        }
        String gateway = args[0];
        String attrName;
        String mote;
        if (args.length == 1) mote = "ffffffffffffffff";
        else mote = args[1];
        GetLastOneDayReliability get = new GetLastOneDayReliability(gateway);
        double t1Offset = -86400.0;   // one day before
        double t2Offset = 0.0;        // now
        attrName = "NodeStatsEvent";
        get.get(t1Offset, t2Offset, attrName, mote);
    }
}
```

Figure A.6: Solution code for Ex2_6.java

A.2 SOAP PHP Application

```
<?php
define("TIMEOUT", 2.0);
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->metadataGet($mote, $attrName);
    foreach ($ret as $result) { echo "Mote " . $result->addr .  " Name " . $result->value ."\n"; }
}
if ($argc != 1 && $argc != 2) { echo "Usage: php get_name.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "name";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
get($gateway, $attrName, $mote);
?>
```

Figure A.7: Solution code for ex2_1.php

```
<?php
define("TIMEOUT", 2.0);
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->metadataGet($mote, $attrName);
    foreach ($ret as $result) {
        echo "Mote " . $result->addr .  " " . $attrName . " " . $result->value ."\n";
    }
}
if ($argc != 1 && $argc != 2) { echo "Usage: php get_location.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
$attrName = "map-x";
get($gateway, $attrName, $mote);
$attrName = "map-y";
get($gateway, $attrName, $mote);
?>
```

Figure A.8: Solution code for ex2_2.php

```
<?php
define("TIMEOUT", 2.0);
function get($t1Offset, $t2Offset, $gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    //$ret = $proxy->eventsReadLast($attrName, $mote);
    $ret = $proxy->eventsReadRelative($t1Offset, $t2Offset, $attrName, $mote, 0, "");
    $ret_arr = $ret->results;
    foreach ($ret_arr as $result) {
        echo "Mote " . $result->addr .  " Time " . $result->timestamp .
             " " . $attrName .  " " . $result->value->TemperatureReadEvent ."\n";
    }
}
if ($argc != 1 && $argc != 2) { echo "Usage: php get_last_temperature.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "TemperatureReadEvent";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
$t1Offset = -86400;
$t2Offset = 0;
get($t1Offset, $t2Offset, $gateway, $attrName, $mote);
?>
```

Figure A.9: Solution code for ex2_3.php

```
<?php
define("TIMEOUT", 2.0);
function rpc_execute($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->rpcExecutesampleRequest($mote, 2, "*");
    foreach ($ret as $result) { echo "Mote " . $result->addr .  " Time " . $result->timestamp ."\n"; }
}
if ($argc != 1 && $argc != 2) { echo "Usage: php rpc_sample_request.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "name";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
rpc_execute($gateway, $attrName, $mote);
?>
```

Figure A.10: Solution code for ex2_4.php

```
<?php
define("TIMEOUT", 2.0);
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->eventsReadLast($attrName, $mote);
    $ret_arr = $ret->results;
    foreach ($ret_arr as $result) {
        echo "Mote " . $result->addr .  " Time " . $result->timestamp .
             " Voltage " . $result->value->NodeStatsEvent->voltage ."\n";
    }
}
if ($argc != 1 && $argc != 2) { echo "Usage: php get_last_battery_voltage.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "NodeStatsEvent";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
get($gateway, $attrName, $mote);
?>
```

Figure A.11: Solution code for ex2_5.php

```
<?php
define("TIMEOUT", 2.0);
function get($t1Offset, $t2Offset, $gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->eventsReadRelative($t1Offset, $t2Offset, $attrName, $mote, 0, "");
    $ret_arr = $ret->results;
    foreach ($ret_arr as $result) {
        echo "Mote " . $result->addr .
           " timestamp " . $result->timestamp .
           " sent " . $result->value->NodeStatsEvent->net_sent . " " .
           " delivered " . $result->value->NodeStatsEvent->net_delivered ."\n";
    }
}
if ($argc != 1 && $argc != 2) { echo "Usage: php get_last_one_day_reliability.php gateway [mote]\n"; exit(3); }
$gateway = $argv[1];
$attrName = "NodeStatsEvent";
if ($argc == 2) $mote = "ffffffffffffffff";
else $mote = $argv[2];
$t1Offset = -86400;
$t2Offset = 0;
get($t1Offset, $t2Offset, $gateway, $attrName, $mote);
?>
```

Figure A.12: Solution code for ex2_6.php

A.3 PHP Web Service

```
<html>
  <head>
    <title>Embedded IP-based Sensor Networks and Systems - PHP Web Service Tutorial </title>
  </head>
  <body>
    <H1> Get Last Temperature Reading </H1>
<?php
define("TIMEOUT", 2.0);
function get($gateway, $attrName, $mote)
{
    $url = "http://" . $gateway . "/gw/rest/V1/?method=gw.getWSDL";
    $proxy = new SoapClient($url);
    $ret = $proxy->eventsReadLast($attrName, $mote);
    echo "<table border=1>";
    echo "<tr><th>Mote</th><th>Time</th><th>$attrName</th></tr>";
    $ret_arr = $ret->results;
    foreach ($ret_arr as $result) {
        $value = $result->value;
        $reading = ($value->TemperatureReadEvent - 3200) * 0.01 * 5.0 / 9.0;
        $converted = "$reading &#176 C";
        $date_out = date(DATE_RFC822,$result->timestamp);
        echo "<tr><td> $result->addr </td>";
        echo "<td> $date_out </td>";
        echo "<td> $converted </td></tr>";
    }
    echo "</table>";
    echo "<p>";
}
$gateway=$_GET['server']; // obtain argument from http GET
$attrName = "TemperatureReadEvent";
$mote = "ffffffffffffffff";
get($gateway, $attrName, $mote);
?>
  </body>
</html>
```

Figure A.13: Solution for get_temperature_celsius.php

Appendix B

Custom IP Application Examples

```
import java.net.*;
import java.io.*;
public class EchoClient {
  public static void main(String[] args) {
    String hostname = "";
    if (args.length == 1) { hostname = args[0]; }
    else { System.err.println("Usage: EchoClient <gateway IP>"); System.exit(0); }
    System.out.println("Type '.' to end.");
    PrintWriter out = null;
    BufferedReader networkIn = null;
    try {
      Socket theSocket = new Socket(hostname, 7);
      networkIn = new BufferedReader(new InputStreamReader(theSocket.getInputStream()));
      BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));
      out = new PrintWriter(theSocket.getOutputStream());
      System.out.println("Connected to echo server");
      while (true) {
        String theLine = userIn.readLine();
        if (theLine.equals(".")) break;
        out.println(theLine);
        out.flush();
        System.out.println(networkIn.readLine());
      }
    }
    catch (IOException e) { System.err.println(e); }
    finally {
      try {
        if (networkIn != null) networkIn.close(); if (out != null) out.close();
      } catch (IOException e) {}
    }
  }  // end main
}  // end EchoClient
```

Figure B.1: Custom IP application: EchoClient

```
import java.net.*;
import java.io.*;
public class LedsClient {
  public static void main(String[] args) {
    String hostname = "";
    if (args.length == 1) { hostname = args[0]; }
    else { System.err.println("Usage: LedsClient <gateway IP>"); System.exit(0); }
    System.out.println("Type 0 to 7 to turn on/off LEDs.");
    System.out.println("Type '.' to end.");
    PrintWriter out = null;
    try {
      Socket theSocket = new Socket(hostname, 10);
      BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));
      out = new PrintWriter(theSocket.getOutputStream());
      System.out.println("Connected to LEDs server");
      while (true) {
        String theLine = userIn.readLine();
        if (theLine.equals(".")) break;
        char typedChar[] = new char[1];
        char convertedChar[] = new char[1];
        typedChar[0]     = theLine.charAt(0);
        convertedChar[0] = (char) (typedChar[0] - '0');
        String convertedLine = new String(convertedChar, 0, 1);
        out.println(convertedLine);
        out.flush();
      }
    }  // end try
    catch (IOException e) { System.err.println(e); }
    finally { if (out != null) out.close(); }
  }  // end main
}  // end LedsClient
```

Figure B.2: Custom IP application: LedsClient

```
import java.net.*;
import java.io.*;
public class SysstatClient {
  public static void main(String[] args) {
    String hostname = "";
    if (args.length == 1) { hostname = args[0]; }
    else { System.err.println("Usage: SysstatClient <gateway IP>"); System.exit(0); }
    PrintWriter out = null;
    BufferedReader networkIn = null;
    try {
      Socket theSocket = new Socket(hostname, 11);
      networkIn = new BufferedReader(new InputStreamReader(theSocket.getInputStream()));
      BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));
      out = new PrintWriter(theSocket.getOutputStream());
      System.out.println("Connected to sysstat server");
      String reply;
      reply = networkIn.readLine();
      while (reply != null) {
        System.out.println(reply);
        reply = networkIn.readLine();
      }
    }  // end try
    catch (IOException e) {
      System.err.println(e);
    }
    finally {
      try { if (networkIn != null) networkIn.close(); if (out != null) out.close(); }
      catch (IOException e) {}
    }
  }  // end main
}  // end SysstatClient
```

Figure B.3: Custom IP application: SysstatClient

```
import java.net.*;
import java.io.*;
public class NetstatClient {
  public static void main(String[] args) {
    String hostname = "";
    if (args.length == 1) { hostname = args[0]; }
    else { System.err.println("Usage: NetstatClient <gateway IP>"); System.exit(0); }
    PrintWriter out = null;
    BufferedReader networkIn = null;
    try {
      Socket theSocket = new Socket(hostname, 15);
      networkIn = new BufferedReader(new InputStreamReader(theSocket.getInputStream()));
      BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));
      out = new PrintWriter(theSocket.getOutputStream());
      System.out.println("Connected to netstat server");
      String reply;
      reply = networkIn.readLine();
      while (reply != null) {
        System.out.println(reply);
        reply = networkIn.readLine();
      }
    }  // end try
    catch (IOException e) { System.err.println(e); }
    finally {
      try { if (networkIn != null) networkIn.close(); if (out != null) out.close(); }
      catch (IOException e) {}
    }
  }  // end main
}  // end NetstatClient
```

Figure B.4: Custom IP application: NetstatClient

```
import java.net.*;
import java.io.*;
public class SensorReadClient {
  public static void main(String[] args) {
    String hostname = "";
    if (args.length == 1) { hostname = args[0]; }
    else { System.err.println("Usage: SensorReadClient <gateway IP>"); System.exit(0); }
    PrintWriter out = null;
    BufferedReader networkIn = null;
    try {
      Socket theSocket = new Socket(hostname, 30);
      networkIn = new BufferedReader(new InputStreamReader(theSocket.getInputStream()));
      BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));
      out = new PrintWriter(theSocket.getOutputStream());
      System.out.println("Connected to sensor reading server");
      String reply;
      reply = networkIn.readLine();
      while (reply != null) {
        System.out.println(reply);
        reply = networkIn.readLine();
      }
    }  // end try
    catch (IOException e) { System.err.println(e); }
    finally {
      try { if (networkIn != null) networkIn.close(); if (out != null) out.close(); }
      catch (IOException e) {}
    }
  }  // end main
}  // end SensorReadClient
```

Figure B.5: Custom IP application: SensorReadClient

Appendix C

Solution for TinyOS 2.0 Programming Projects Based on Open Source Distribution

C.1 Project 1: Push and Toggle

```
COMPONENT=ToggleAppC
include $(MAKERULES)
```

Figure C.1: Makefile: Makefile for Toggle

```
/**
 * Toggle - a minimal TinyOS 2.0 application
 *
 *            +---------+
 *            |  MainC  |
 *            +---------+
 *                 | init
 *                 |
 *    +--------------------------+
 *    |           ToggleC        |
 *    +--------------------------+
 *      | Boot     |Leds       |Notify
 *      |          |           |
 *      | Boot     |Leds       |Notify
 *   +-------+  +-------+  +-------------+
 *   | MainC |  | LedsC |  | UserButtonC |
 *   +-------+  +-------+  +-------------+
 *
 */
configuration ToggleAppC {}
implementation {
  components ToggleC;
  components MainC;
  components UserButtonC;
  components LedsC;
  ToggleC.Boot -> MainC;
  ToggleC.Notify -> UserButtonC;
  ToggleC.Leds -> LedsC;
}
```

Figure C.2: ToggleAppC.nc: Configuration module for Toggle

```
#include <UserButton.h>
module ToggleC {
  uses {
    interface Boot;
    interface Notify<button_state_t>;
    interface Leds;
  }
}
implementation {
  event void Boot.booted() {
    call Notify.enable();
  }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) {
    if (state == BUTTON_PRESSED) {
      call Leds.led0Toggle();
    }
  }
}
```

Figure C.3: ToggleC.nc: Implementation module for Toggle

C.2 Project 2: Blink and Count

```
COMPONENT=CountAppC
include $(MAKERULES)
```

Figure C.4: Makefile: Makefile for Count

```
/**
 * Count - a Tinyos 2.0 application with timers
 *
 *            +---------+
 *            |  MainC  |
 *            +---------+
 *                 | init
 *                 |
 *    +-----------------------------------------+
 *    |           ToggleC                       |
 *    +-----------------------------------------+
 *      |Boot     |Leds       |Notify         |Timer0,Timer1,Timer2
 *      |         |           |               |
 *      |Boot     |Leds       |Notify         |Timer0,Timer1,Timer2
 *   +-------+  +-------+  +-------------+  +-------------+
 *   | MainC |  | LedsC |  | UserButtonC |  | TimerMilliC |
 *   +-------+  +-------+  +-------------+  +-------------+
 *
 */
configuration CountAppC {}
implementation {
  // Declare the components
  components CountC;
  components MainC;
  components UserButtonC;
  components LedsC;
  // Wire together the interfaces
  CountC.Boot -> MainC;
  CountC.Notify -> UserButtonC;
  CountC.Leds -> LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;
  CountC.Timer0 -> Timer0;
  CountC.Timer1 -> Timer1;
  CountC.Timer2 -> Timer2;
}
```

Figure C.5: CountApp.nc: Configuration module for Count

```
#include <UserButton.h>
module CountC {      // Definition in external interfaces
  uses {
    interface Boot;
    interface Notify<button_state_t>;
    interface Leds;
    interface Timer<TMilli> as Timer0;
    interface Timer<TMilli> as Timer1;
    interface Timer<TMilli> as Timer2;
  }
}
implementation {
  bool isTimerRunning = FALSE;
  uint8_t counter = 0;
  void rotate_timer() {
    counter = (counter + 1) & 0x03;
    call Leds.set(0);
    if (counter == 0) {
      call Timer0.stop();
      call Timer1.stop();
      call Timer2.stop();
    }
    else if (counter == 1) {
      call Timer0.startPeriodic(1000);
      call Timer1.stop();
      call Timer2.stop();
    }
    else if (counter == 2) {
      call Timer0.startPeriodic(1000);
      call Timer1.startPeriodic(2000);
      call Timer2.stop();
    }
    else if (counter == 3) {
      call Timer0.startPeriodic(1000);
      call Timer1.startPeriodic(2000);
      call Timer2.startPeriodic(4000);
    }
  }
  event void Boot.booted() {
    call Notify.enable();   // enable input subsystem upon boot
  }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) {
    if (state == BUTTON_PRESSED) { rotate_timer(); }
  }
  event void Timer0.fired() { call Leds.led0Toggle(); }
  event void Timer1.fired() { call Leds.led1Toggle(); }
  event void Timer2.fired() { call Leds.led2Toggle(); }
}
```

Figure C.6: CountC.nc: Implementation module for Count

C.3 Project 3: PrintSerial

No exercise is given for this project.

C.4 Project 4: Single and Dual

```
COMPONENT=DualAppC
BUILD_EXTRA_DEPS += PrintReading.class
CLEAN_EXTRA = *.class PrintReadingMsg.java
PrintReading.class: $(wildcard *.java) PrintReadingMsg.java
  javac *.java
PrintReadingMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=PrintReadingMsg PrintReading.h
print_reading_msg -o $@
include $(MAKERULES)
```

Figure C.7: Makefile: Makefile for Dual

```
generic configuration TemperatureC() {
  provides interface Read<uint16_t>;
}
implementation {
  components new Msp430InternalTemperatureC();
  Read = Msp430InternalTemperatureC.Read;
}
```

Figure C.8: TemperatureC.nc: Wrapper for Telosb internal temperature sensor

```
generic configuration DemoTemperatureSensorC()
{
  provides interface Read<uint16_t>;
}
implementation
{
  components new TemperatureC() as DemoTemperatureSensor;
  Read = DemoTemperatureSensor;
}
```

Figure C.9: DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor

```
/**
 * Dual - A TinyOS application that shows the concept of sending.
 *
 *           +---------+
 *           |  MainC  |
 *           +---------+
 *                | init
 *                |
 *   +-----------------------------------------------+
 *   |           DualC                               |
 *   +-----------------------------------------------+
 *    |Boot |   |Leds    |   | Control            |    | ReadVoltage
 *    |     |   |        |   | AMSend, Packet     |    |
 *    |     |   |        |   |                    |    |
 *    |Boot |   |Leds    |   | SplitControl       |    | Read
 *    |     |   |        |   | AMSend, Packet     |    |
 *  +-----+ | +-----+    | +--------------------+ | +-----------+
 *  |MainC| | |LedsC|    | |SerialActiveMessageC| | |DemoSensorC|
 *  +-----+ | +-----+    | +--------------------+ | +-----------+
 *          |            |                        |
 *          |Notify      |MilliTimer              | ReadTemperature
 *          |            |                        |
 *          |            |                        |
 *          |Notify      |Timer                   | Read
 *          |            |                        |
 *       +-----------+ +-----------+    +----------------------+
 *       |UserButtonC| |TimerMilliC|    |DemoTemperatureSensorC|
 *       +-----------+ +-----------+    +----------------------+
 *
 */
#include "PrintReading.h"
configuration DualAppC {}
implementation {
  components DualC, LedsC, MainC;
  components SerialActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  components new DemoTemperatureSensorC() as TemperatureSensor;
  DualC.Boot -> MainC.Boot;
  DualC.Control -> AM;
  DualC.AMSend -> AM.AMSend[AM_PRINT_READING_MSG];
  DualC.Leds -> LedsC;
  DualC.MilliTimer -> TimerMilliC;
  DualC.Packet -> AM;
  DualC.Notify -> UserButtonC;
  DualC.ReadVoltage -> VoltageSensor.Read;
  DualC.ReadTemperature -> TemperatureSensor.Read;
}
```

Figure C.10: DualAppC.nc: Configuration module for Dual

```
#include <UserButton.h>
#include "Timer.h"
#include "PrintReading.h"
module DualC {
  uses {
    interface SplitControl as Control;  interface Leds;  interface Boot;
    interface AMSend;  interface Timer<TMilli> as MilliTimer;
    interface Packet; interface Notify<button_state_t>;
    interface Read<uint16_t> as ReadVoltage;  interface Read<uint16_t> as ReadTemperature;
  }
}
implementation {
  message_t packet;  bool locked = FALSE;  uint8_t flag = 0;
  char m_error[13] = "Sample Error\n";  char m_data[TOSH_DATA_LENGTH - 5];
  uint16_t voltage_reading = 0;  uint16_t temperature_reading = 0;
  event void Boot.booted() {
    call Control.start();  call Notify.enable();  call MilliTimer.startPeriodic(1024);
  }
  void serial_print() {
    if (locked) { return; }
    else {
      uint8_t i;
      print_reading_msg_t* rcm = (print_reading_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->flag = flag;
      if (flag & FLAG_BUFFER) {
        for (i = 0; i < TOSH_DATA_LENGTH - 5; i++) { rcm->buffer[i] = m_data[i]; }
      }
      if (flag & FLAG_VOLTAGE_READING) { rcm->voltage_reading = voltage_reading; }
      if (flag & FLAG_TEMPERATURE_READING) { rcm->temperature_reading = temperature_reading; }
      flag = 0;
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
          sizeof(print_reading_msg_t)) == SUCCESS) { locked = TRUE; }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) { locked = FALSE; }
  }
  void report_problem() {
    uint8_t i;  flag = FLAG_BUFFER;
    for (i = 0; i < sizeof(m_error); i++) { m_data[i] = m_error[i]; }
    m_data[i] = 0;
    serial_print();
  }
  void report_voltage_reading(uint16_t data) {
    flag |= FLAG_VOLTAGE_READING;
    voltage_reading = data;
    if (call ReadTemperature.read() != SUCCESS)  report_problem();
  }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result != SUCCESS) { report_problem(); }
    report_voltage_reading(data);
  }
  void report_temperature_reading(uint16_t data) {
    flag |= FLAG_TEMPERATURE_READING;
    temperature_reading = data;
    serial_print();
  }
  event void ReadTemperature.readDone(error_t result, uint16_t data) {
    if (result != SUCCESS) { report_problem(); }
    report_temperature_reading(data);
  }
  event void MilliTimer.fired() {
    call Leds.led1Toggle();
    if (call ReadVoltage.read() != SUCCESS) report_problem();
  }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) {
    if (state == BUTTON_PRESSED) { call Leds.led0Toggle(); }
  }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```


Figure C.11: DualC.nc: Implementation module for Dual

C.5 Project 5: Raw and Smooth

```
COMPONENT=SmoothAppC
BUILD_EXTRA_DEPS += PrintReadingArr.class
CLEAN_EXTRA = *.class PrintReadingArrMsg.java
PrintReadingArr.class: $(wildcard *.java) PrintReadingArrMsg.java
  javac *.java
PrintReadingArrMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=PrintReadingArrMsg PrintReadingArr.h
print_reading_arr_msg -o $@
include $(MAKERULES)
```

Figure C.12: Makefile: Makefile for Smooth

```
/**
 * Smooth - A TinyOS application that shows the concept of task.
 *
 *           +---------+
 *           |  MainC  |
 *           +---------+
 *                | init
 *                |
 *   +-----------------------------------------------+
 *   |           SmoothC                             |
 *   +-----------------------------------------------+
 *    |Boot |   |Leds    |   | Control            |
 *    |     |   |        |   | AMSend, Packet     |
 *    |     |   |        |   |                    |
 *    |Boot |   |Leds    |   | SplitControl       |
 *    |     |   |        |   | AMSend, Packet     |
 *  +-----+ | +-----+    | +--------------------+ |
 *  |MainC| | |LedsC|    | |SerialActiveMessageC| |
 *  +-----+ | +-----+    | +--------------------+ |
 *          |            |                        |
 *          |Notify      |MilliTimer              | ReadVoltage
 *          |            |                        |
 *          |            |                        |
 *          |Notify      |Timer                   | Read
 *          |            |                        |
 *       +-----------+ +-----------+           +-----------+
 *       |UserButtonC| |TimerMilliC|           |DemoSensorC|
 *       +-----------+ +-----------+           +-----------+
 *
 */
#include "PrintReadingArr.h"
configuration SmoothAppC {}
implementation {
  components SmoothC, LedsC, MainC;
  components SerialActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  SmoothC.Boot -> MainC.Boot;
  SmoothC.Control -> AM;
  SmoothC.AMSend -> AM.AMSend[AM_PRINT_READING_ARR_MSG];
  SmoothC.Leds -> LedsC;
  SmoothC.MilliTimer -> TimerMilliC;
  SmoothC.Packet -> AM;
  SmoothC.Notify -> UserButtonC;
  SmoothC.ReadVoltage -> VoltageSensor.Read;
}
```

Figure C.13: SmoothAppC.nc: Configuration module for Smooth

```
#include <UserButton.h>
#include "Timer.h"
#include "PrintReadingArr.h"
module SmoothC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
    interface Read<uint16_t> as ReadVoltage;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE; uint8_t counter = 0;
  uint16_t min; uint16_t max; uint16_t mean;
  uint16_t raw_reading[MAX_READINGS];
  uint16_t smooth_reading[MAX_READINGS];
  event void Boot.booted() {
    call Control.start(); call Notify.enable(); call MilliTimer.startPeriodic(250);
  }
  void serial_print() {
    if (locked) { return; }
    else {
      uint8_t i;
      print_reading_arr_msg_t* rcm = (print_reading_arr_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->nodeid = TOS_NODE_ID;
      rcm->min = min;
      rcm->max = max;
      rcm->mean = mean;
      for (i = 0; i < MAX_READINGS; i++) {
        rcm->raw_reading[i] = raw_reading[i]; rcm->smooth_reading[i] = smooth_reading[i];
      }
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(print_reading_arr_msg_t)) == SUCCESS) { locked = TRUE; }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) { if (&packet == bufPtr)  locked = FALSE;  }
  uint16_t smoothen(uint16_t prev_reading, uint16_t new_reading) {
    return (prev_reading * (DENOMINATOR - SMOOTH_FACTOR) + new_reading * SMOOTH_FACTOR) / DENOMINATOR;
  }
  task void process_voltage_reading() {
    int i;
    uint16_t temp_min = raw_reading[0];
    uint16_t temp_max = raw_reading[0];
    uint16_t temp_sum = raw_reading[0];
    smooth_reading[0] = raw_reading[0];
    for (i = 1; i < MAX_READINGS; i++) {
      temp_sum += raw_reading[i];
      if (raw_reading[i] < temp_min) { temp_min = raw_reading[i]; }
      if (raw_reading[i] > temp_max) { temp_max = raw_reading[i]; }
      smooth_reading[i] = smoothen(smooth_reading[i-1], raw_reading[i]);
    }
    min = temp_min;  max = temp_max;  mean = temp_sum / MAX_READINGS;
    serial_print();
  }
  void store_voltage_reading(uint16_t data) {
    raw_reading[counter++] = data;
    if (counter == MAX_READINGS) { post process_voltage_reading(); counter = 0; }
  }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { store_voltage_reading(data); }
  }
  event void MilliTimer.fired() { call Leds.led1Toggle(); call ReadVoltage.read(); }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) { if (state == BUTTON_PRESSED) call Leds.led0Toggle(); }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```


Figure C.14: SmoothC.nc: Implementation module for Smooth

C.6 Project 6: Counts and Readings

```
COMPONENT=ReadingsAppC
BUILD_EXTRA_DEPS += PrintCounterReading.class
CLEAN_EXTRA = *.class PrintCounterReadingMsg.java
PrintCounterReading.class: $(wildcard *.java) PrintCounterReadingMsg.java
  javac *.java
PrintCounterReadingMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=PrintCounterReadingMsg PrintCounterReading.h
print_counter_reading_msg -o $@
include $(MAKERULES)
```

Figure C.15: Makefile: Makefile for Readings

```
/**
 * Counts - A TinyOS application that shows the concept of sending
 *          a radio message.
 *
 *           +---------+
 *           |  MainC  |
 *           +---------+
 *                | init
 *                |
 *   +-----------------------------------------------+
 *   |           ReadingsC                           |
 *   +-----------------------------------------------+
 *    |Boot |   |Leds    |   | Control            |
 *    |     |   |        |   | AMSend, Packet     |
 *    |     |   |        |   |                    |
 *    |Boot |   |Leds    |   | SplitControl       |
 *    |     |   |        |   | AMSend, Packet     |
 *  +-----+ | +-----+    | +--------------+       |
 *  |MainC| | |LedsC|    | |ActiveMessageC|       |
 *  +-----+ | +-----+    | +--------------+       |
 *          |            |                        |
 *          |Notify      |MilliTimer              | ReadVoltage
 *          |            |                        |
 *          |            |                        |
 *          |Notify      |Timer                   | Read
 *          |            |                        |
 *       +-----------+ +-----------+           +-----------+
 *       |UserButtonC| |TimerMilliC|           |DemoSensorC|
 *       +-----------+ +-----------+           +-----------+
 *
 */
#include "PrintCounterReading.h"
configuration ReadingsAppC {}
implementation {
  components ReadingsC, LedsC, MainC;
  components ActiveMessageC as AM;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  ReadingsC.Boot -> MainC.Boot;
  ReadingsC.Control -> AM;
  ReadingsC.AMSend -> AM.AMSend[AM_PRINT_COUNTER_READING_MSG];
  ReadingsC.Leds -> LedsC;
  ReadingsC.MilliTimer -> TimerMilliC;
  ReadingsC.Packet -> AM;
  ReadingsC.Notify -> UserButtonC;
  ReadingsC.ReadVoltage -> VoltageSensor.Read;
}
```

Figure C.16: ReadingsAppC.nc: Configuration module for Readings

```
#include <UserButton.h>
#include "Timer.h"
#include "PrintCounterReading.h"
module ReadingsC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
    interface Read<uint16_t> as ReadVoltage;
  }
}
implementation {
  message_t packet;
  bool locked = FALSE;
  uint8_t flag = 0;
  uint16_t counter = 0;
  uint16_t voltage_reading = 0;
  event void Boot.booted() {
    call Control.start();
    call Notify.enable();
    call MilliTimer.startPeriodic(1024);
  }
  void serial_print() {
    if (locked) { return; }
    else {
      print_counter_reading_msg_t* rcm = (print_counter_reading_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->flag = flag;
      rcm->nodeid = TOS_NODE_ID;
      if (flag & FLAG_COUNTER) { rcm->counter = counter; }
      if (flag & FLAG_VOLTAGE_READING) { rcm->voltage_reading = voltage_reading; }
      flag = 0;
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
          sizeof(print_counter_reading_msg_t)) == SUCCESS) { locked = TRUE; }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) {
      locked = FALSE;
    }
  }
  void report_voltage_reading(uint16_t data) {
    flag |= FLAG_VOLTAGE_READING;
    voltage_reading = data;
    serial_print();
  }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { report_voltage_reading(data); }
  }
  event void MilliTimer.fired() {
    call Leds.led1Toggle();
    counter++;
    flag |= FLAG_COUNTER;
    call ReadVoltage.read();
  }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) { if (state == BUTTON_PRESSED) call Leds.led0Toggle(); }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
}
```

Figure C.17: ReadingsC.nc: Implementation module for Readings

C.7 Project 7: Request and RequestSample

```
COMPONENT=RequestSampleAppC
BUILD_EXTRA_DEPS += RequestReading.class
CLEAN_EXTRA = *.class RequestReadingMsg.java
RequestReading.class: $(wildcard *.java) RequestReadingMsg.java
  javac *.java
RequestReadingMsg.java:
  mig java -target=null $(CFLAGS) -java-classname=RequestReadingMsg RequestReading.h
request_reading_msg -o $@
include $(MAKERULES)
```

Figure C.18: Makefile: Makefile for RequestSample

```
generic configuration TemperatureC() {
  provides interface Read<uint16_t>;
}
implementation {
  components new Msp430InternalTemperatureC();
  Read = Msp430InternalTemperatureC.Read;
}
```

Figure C.19: TemperatureC.nc: Wrapper for Telosb internal temperature sensor

```
generic configuration DemoTemperatureSensorC()
{
  provides interface Read<uint16_t>;
}
implementation
{
  components new TemperatureC() as DemoTemperatureSensor;
  Read = DemoTemperatureSensor;
}
```

Figure C.20: DemoTemperatureSensorC.nc: Component for Telosb internal temperature sensor

```
#include "RequestReading.h"
configuration RequestSampleAppC {}
implementation {
  components RequestSampleC, LedsC, MainC;
  components new TimerMilliC();
  components UserButtonC;
  components new DemoSensorC() as VoltageSensor;
  components new DemoTemperatureSensorC() as TemperatureSensor;
  components ActiveMessageC as AM;
  components new AMSenderC(AM_REQUEST_READING_MSG);
  components new AMReceiverC(AM_REQUEST_READING_MSG);
  RequestSampleC.Boot -> MainC.Boot;
  RequestSampleC.Control -> AM;
  RequestSampleC.AMSend -> AMSenderC;
  RequestSampleC.Receive -> AMReceiverC;
  RequestSampleC.Leds -> LedsC;
  RequestSampleC.MilliTimer -> TimerMilliC;
  RequestSampleC.Packet -> AM;
  RequestSampleC.Notify -> UserButtonC;
  RequestSampleC.ReadVoltage -> VoltageSensor.Read;
  RequestSampleC.ReadTemperature -> TemperatureSensor.Read;
}
```

Figure C.21: RequestSampleAppC.nc: Configuration module for RequestSample

```
#include <UserButton.h>
#include "Timer.h"
#include "RequestReading.h"
module RequestSampleC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Receive;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
    interface Read<uint16_t> as ReadVoltage;
    interface Read<uint16_t> as ReadTemperature;
  }
}
implementation {
  message_t packet;  bool locked = FALSE;
  uint8_t flag = 0;  uint16_t counter = 0;
  uint16_t voltage_reading = 0;  uint16_t temperature_reading = 0;
  event void Boot.booted() {
    call Control.start();  call Notify.enable();  call MilliTimer.startPeriodic(1024);
  }
  task void sample_voltage() { call ReadVoltage.read(); }
  task void sample_temperature() { call ReadTemperature.read(); }
  task void reply_request() {
    if (locked) { return; }
    else {
      request_reading_msg_t* rcm = (request_reading_msg_t*) call Packet.getPayload(&packet, NULL);
      rcm->flag = flag;
      rcm->nodeid = TOS_NODE_ID;
      if (flag & FLAG_COUNTER) { rcm->counter = counter; }
      if (flag & FLAG_VOLTAGE_READING) { rcm->voltage_reading = voltage_reading; }
      if (flag & FLAG_TEMPERATURE_READING) { rcm->temperature_reading = temperature_reading; }
      if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
          sizeof(request_reading_msg_t)) == SUCCESS) { locked = TRUE; }
    }
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) { if (&packet == bufPtr) locked = FALSE; }
  event void ReadVoltage.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { voltage_reading = data; post sample_temperature(); }
  }
  event void ReadTemperature.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) { temperature_reading = data; post reply_request(); }
  }
  event void MilliTimer.fired() { call Leds.led1Toggle(); counter++; }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) {
    if (state == BUTTON_PRESSED) { call Leds.led0Toggle(); post sample_voltage();
    }
  }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
  event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t len) {
    request_reading_msg_t* rcm = (request_reading_msg_t*)payload;
    call Leds.led2Toggle();
    if (rcm->nodeid == TOS_NODE_ID) { flag = rcm->flag; post sample_voltage(); }
    return bufPtr;
  }
}
```

Figure C.22: RequestSampleC.nc: Implementation module for RequestSample

C.8 Project 8: CountToRadio and RadioToCount

```
COMPONENT=RadioToCountAppC
include $(MAKERULES)
```

Figure C.23: Makefile: Makefile for RadioToCount

```
#include "Counter.h"
configuration RadioToCountAppC {}
implementation {
  components RadioToCountC, LedsC, MainC;
  components new TimerMilliC();
  components UserButtonC;
  components ActiveMessageC as AM;
  components new AMSenderC(AM_COUNTER_MSG);
  components new AMReceiverC(AM_COUNTER_MSG);
  RadioToCountC.Boot -> MainC.Boot;
  RadioToCountC.Control -> AM;
  RadioToCountC.AMSend -> AMSenderC;
  RadioToCountC.Receive -> AMReceiverC;
  RadioToCountC.Leds -> LedsC;
  RadioToCountC.MilliTimer -> TimerMilliC;
  RadioToCountC.Packet -> AM;
  RadioToCountC.Notify -> UserButtonC;
}
```

Figure C.24: RadioToCountAppC.nc: Configuration module for RadioToCount

```
#include <UserButton.h>
#include "Timer.h"
#include "Counter.h"
module RadioToCountC {
  uses {
    interface SplitControl as Control;
    interface Leds;
    interface Boot;
    interface AMSend;
    interface Receive;
    interface Timer<TMilli> as MilliTimer;
    interface Packet;
    interface Notify<button_state_t>;
  }
}
implementation {
  message_t packet;
  uint16_t counter = 0;
  event void Boot.booted() {
    call Control.start();
    call Notify.enable();
    call MilliTimer.startPeriodic(1024);
  }
  event void AMSend.sendDone(message_t* bufPtr, error_t error) { }
  event void MilliTimer.fired() { }
  // state can be either BUTTON_PRESSED or BUTTON_RELEASED
  event void Notify.notify( button_state_t state ) { }
  event void Control.startDone(error_t err) { }
  event void Control.stopDone(error_t err)  { }
  event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t len) {
    counter_msg_t* rcm = (counter_msg_t*)payload;
    counter = rcm->counter;
    call Leds.set(counter & 0x07);
    return bufPtr;
  }
}
```

Figure C.25: RadioToCountC.nc: Implementation module for RadioToCount