KTH Electrical Engineering

Wireless Sensor Networks in TinyOS

Other tutorials

AITOR HERNÁNDEZ

Stockholm July 25, 2011

TRITA-EE 2XXX:YYY
Version: 1.0

Contents

Chapter 1

Introduction

In previous documents of these series we have learned how to install TinyOS and compile applications [4], the different tools available to debug the programs [2], the communication mechanism between the PC and the motes [3] and some practical experiments [1, 5, 6].

In the experiments we use some extra tools that are not in any document. For this reason, we would like to provide the information and guidance to some useful applications.

In this document, we explain how to use a serial communication between motes or other devices by using the Universal asynchronous receiver/transmitter (UART). By using UART, we face problem with the resource arbitration if we use the radio chip. Chapter 3 shows some examples and present the problems that could be found by using the serial communication.

On the other hand, in Chapter 2 we explain how to analyse easily the performance of the communications. The analysis will give the reliability, errors, delay and inter-arrival time.

Chapter 2

Performance Evaluation

In wireless communication, it is important to know the probability of success or failure that the packets have. It is clear that for protocol developers is mandatory, but for control experiments it is interesting to detect if the problems are due to a dropping packets, or the performance of the controller is not good enough.

This analysis does not include any extra tool or platform other than the motes. It consists on one mote which sniffs all the packets on the network given a certain channel. We show an example in a network using the IEEE 802.15.4 but it could be used within all the protocols that have a Acknowledge packet (ACK) mechanism.

2.1 Requirements

The requirements for having a performance analyser in our network are:

- One mote with the `SnifferPerformance` application
- Serial-Forwarder to read the values from the mote. [3]
- Application to analyse the packets from the serial forwarder { Matlab, Matlab/Simulink, LabView }

For the tutorial we have a demo network using the IEEE 802.15.4 standard protocol in the beacon-enabled mode. There is one mote transmitting to the coordinator of the network, and the coordinator forward the packet to another mote. Figure 2.1 shows the scenario.

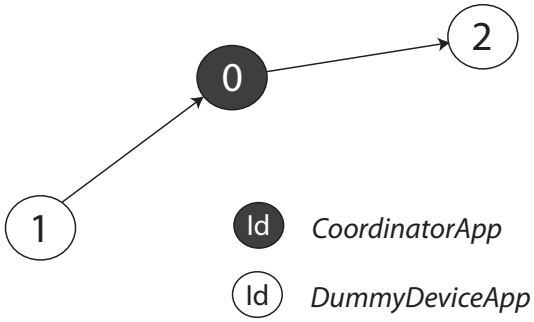


Figure 2.1: Topology of the network with one coordinator and two devices

2.2 Installation

To configure this tutorial compile the different application as it is explained in Figure 2.2. Figure 2.2 shows the scenario with all the motes involved on this tutorial with the application and the identification number that they need.

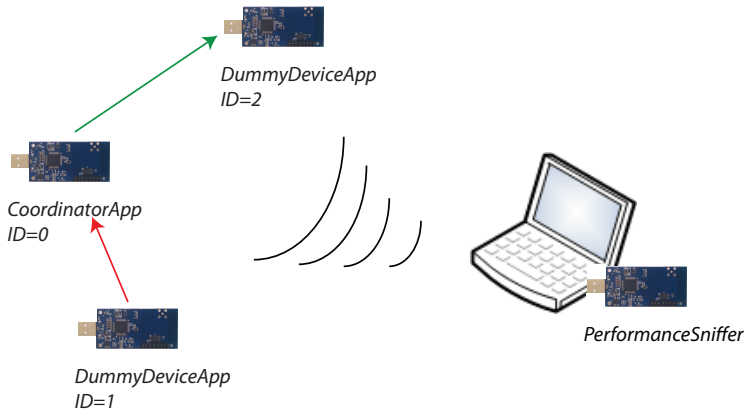


Figure 2.2: Scenario with all the motes involved on this tutorial

2.3 Source Code

In this Section, we show which parts of the code we need to modify in order to add this functionality to our scenario. Remark that this mechanism only works in protocol with ACK mechanism, where the transmitter knows if a packet has successfully received or not.

Below, we divide the modifications regarding the files we need to modify.

2.3.1 Header File

Although, it is not explained anywhere, for transmission it is better to define an `struc` message with the required fields. We consider that we are using a dummy message called `DummyInformationMsg` which contains:

```
typedef nx_struct DummyInformationMsg{
        nx_uint8_t dummyData[10];
}DummyInformationMsg;
```

For the performance evaluation we want to add another `struc` at the end of our previous message to analyse the performance. We define a `PerformanceParams struc` as:

```
typedef nx_struct PerformanceParams {
        nx_uint16_t pckTotal;
        nx_uint16_t pckSuccess;
        nx_uint16_t delay;
} PerformanceParams;
```

Then, the final message that we send is:

```
typedef nx_struct DummyInformationMsg{
        nx_uint8_t dummyData[10];
        PerformanceParams performValues;
}DummyInformationMsg;
```

It is really important that the `PerformanceParams` is placed at the end of the message, because in the `PerformanceSniffer` application, we only get the last bytes of the message which contain the `PerformanceParams`.

We need to add the `SLIDING_WINDOW_PERFORMANCE` definition with the desired number, for example 200 packets.

2.3.2 Transmitters Application

All the code that we need to modify is on the applications where **we transmit packets**. In our example both `CoordinatorApp` and `DummyDeviceApp` send packets.

In the previous Section we add a `PerformanceParams` field to the `DummyInformationMsg` message. In the code for the transmitter application is where we need to add the functionality.

1. Instantiate a new `PerformanceParams` variable and a temporal variable to compute the delay. In the global variable definitions we add:

```
// PERFORMANCE EVALUATION
PerformanceParams sendPerfParams;
uint32_t timestamp2Delay;
```

2. To compute the delay we need a component that provides the timing. It could be a {`LocalTime`, `Alarm` or `Timer` }. We select a `LocalTime` components with `TSymbolIEEE802154` units ($16\mu s$).

   a) Add the interface to the *uses* in the application file, { `TestCoordReceiverC.nc`, `TestDeviceSenderC.nc`}

```
uses interface LocalTime<TSymbolIEEE802154>;
```

   b) Add the wiring in the *wiring* file,`TestDataAppC.nc` for both applications.

```
components LocalTime62500hzC;
App.LocalTime -> LocalTime62500hzC;
```

3. Initialize the `PerformanceParams`. We initialize the variable in the `Boot.booted()` function:

```
sendPerfParams.pckTotal = 0;
sendPerfParams.pckSuccess = 0;
timestamp2Delay = 0;
```

4. Every time we send a packet we need to:

   - Copy the current `PerformanceParams` variable in the message
   - Set the new timestamps value in the delay temporal variable
   - Increment the number of total packets `pckTotal`

```
memcpy(&(dummyMsg->performValues), &sendPerfParams,
        sizeof(PerformanceParams));
timestamp2Delay = call LocalTime.get();
sendPerfParams.pckTotal ++;
```

   With our implementation of the IEEE 802.15.4, it is before the `call MCPS_DATA.request(...)` function.

5. When we receive the event for the end of transmission we need to:

   - Reset the counter if we reach the maximum
   - Increment the number of success packets `pckSuccess` if the transmission has been successes.
   - Set the delay by subtracting to the current timestamps the start time stored in `timestamp2Delay`

```
if (sendPerfParams.pckTotal >=
        SLIDING_WINDOW_PERFORMANCE) {
        sendPerfParams.pckTotal = 0;
        sendPerfParams.pckSuccess = 0;
        return;
}
if (status == IEEE154_SUCCESS) {
        sendPerfParams.pckSuccess ++;
        call Leds.led1Toggle();
}
sendPerfParams.delay = (uint16_t) timestamp -
        timestamp2Delay;
```

With our implementation of the IEEE 802.15.4, the event is `call MCPS_DATA.confirm(...)` function.

2.4 Usage and Results

If we have all the motes with their application and properly running, it is time to analyse the output of the `PerformanceSniffer` mote. As we have seen in the analysis of the inverted pendulum [1], we could analyse the data in LabView, but also in Matlab.

It is important to show the structure of the packets that we receive, in case we want to modify or add more fields to the packets. Figure 2.3 shows the structure.

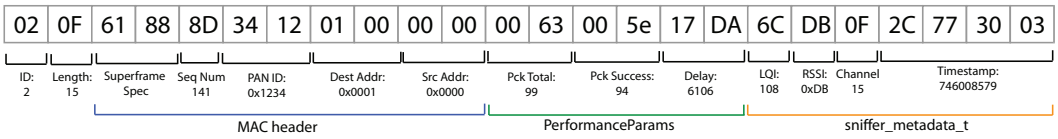


Figure 2.3: Structure of a sniffer packet transmitted over the USB port from the mote to the PC.

2.4.1 LabView

LabView is used to show the performance evaluation online, so when the network or experiment is running we can see how the communication is performing. In the folder `PerformanceEvaluation/LabView` there are the main LabView file and their dependencies.

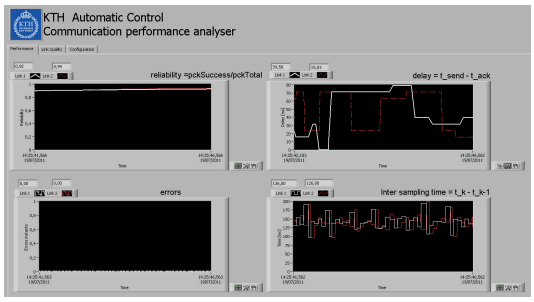
To use it, first we need to run the Serial-Forwarder server [3]. Once the `sf` is running on the server, we run the LabView file `PerformanceAnalizer.vi`.

Figure 2.5a shows the front panel with the *Communication performance* tab. In this tab, we can see four panel with the reliability, delay, errors and inter-arrival time.

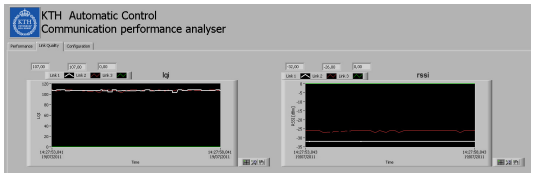
Figure 2.4b shows the Link Quality Indication (LQI) and Received Signal Strength Indication (RSSI) of the messages in the sniffer mote.

2.4.2 Matlab

For the Matlab case, we do not have the performance evaluation online, we just do a post processing after the experiments has finished. There are different ways to store the data by using the `seriallisten`. Below we have a list:



(a) Communication Performance tab



(b) Link quality tab

Figure 2.4: Screenshots of the `PerformanceSniffer.vi` file

- The default `seriallisten` (or `sf` + `sflisten`) requires a post-processing to get the values in Matlab, because we can not import hex number automatically
- If we modify the `seriallisten` and instead of printing decimal hex numbers, we print decimal number, we can easily regenerate the information in Matlab.
- The easy solution is to print automatically the information parsed. This is the method we explain with more detail. we modify the `seriallisten` program [3].

`seriallisten` modification

In Figure 2.3 we have the structure of the messages received from the Universal Serial Bus (USB) port. So, we could easily parse the information.

The program is placed in the folder `PerformanceEvaluation/Matlab/seriallisten`. To run the `seriallisten`, we type the following command:

```
./seriallisten /dev/ttyUSB3 115200 > test
```

where the `PerformanceSniffer` program is installed in the mote `/dev/ttyUSB3` and we redirect the standard output to the file called `test`

If we open the `test` file, we obtain a file with the following information

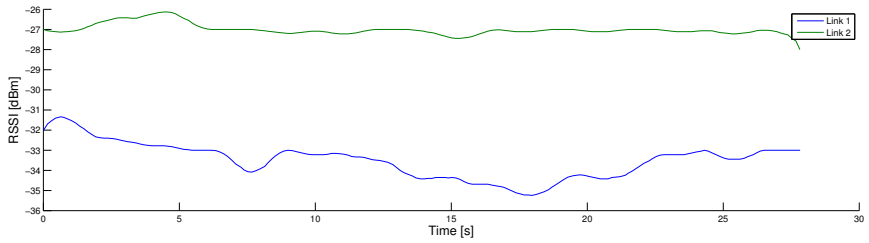
```
seqNum panId dstId srcId pckTotal pckSuccess delay lqi rssi channel timestamp
124 4660 0 1 139 139 374 107 -26 16 3865802241
10 4660 2 0 162 162 374 108 -32 16 3765204481
```

...

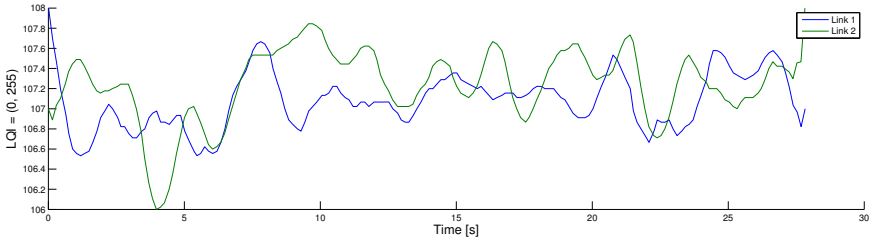
With the file `PerformanceEvaluation/Matlab/commPerformance.m`, we can show the results from the output. In the previous example by typing:

```
» commPerformance('test')
```

we obtain different results as we show in Figures 2.5.



(a) RSSI in the sniffer mote in dBm



(b) LQI in the sniffer mote

Figure 2.5: Some results in Matlab

Chapter 3

Serial Communication

This application provides the template to communicate between two motes using the UART. This scenario is not practical because usually we do not need to communicate between two motes through UART. The Chapter includes two application examples. Section 3.1 shows the example where two motes are transmitting one to each other. Section 3.2 shows an application example. In that case, we simulate a temperature, CO2 and humidity sensor that transmit data every two seconds with a certain structure.

Moreover, Section 3.3 shows some problems and solutions that you could find using the serial communication and other chips in the mote.

3.1 Example

This application shows the basic components that are required to stablish a UART connection between the two motes.

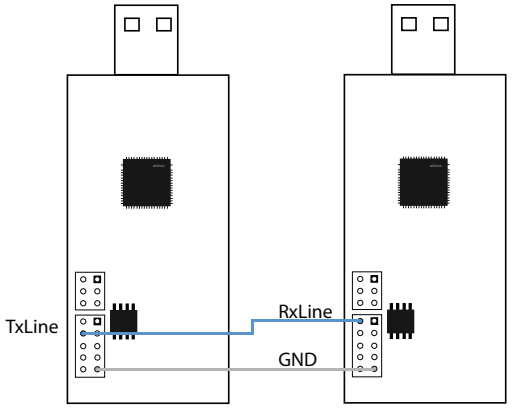


Figure 3.1: UART connection between two motes

To transmit data using the UART lines, we need to connect the motes as Figure 3.1 shows. The transmission line in the expansion header uses the UART 0 from the mote. Then, we need special mechanism to use serial communication and the radio at the same time.

To configure UART protocol, we need to connect the transmission/reception line and configure the protocol in the same way.

In TinyOS we have a specific `union` which configures the UART. The `union` includes information about, baud rate, modulation, length, parity,... and other configuration. More details could be found in the `msp430usart.h`.

In this example the sender is implemented to send one byte every 2 seconds. The blue Light-Emitting Diode (LED) blinks when the packet is sent. The receiver blinks the green LED when a byte is received. When the byte has been received the mote show the value with the `printf`.

3.2 TestTempSensor

In this example, we show a real application which is used to get the information from a sensor [7]. The sensor provides a serial communication to get the sensor values. Figure 3.3 shows the connection between the mote and the sensor board. Mention that they are not with the same scale.

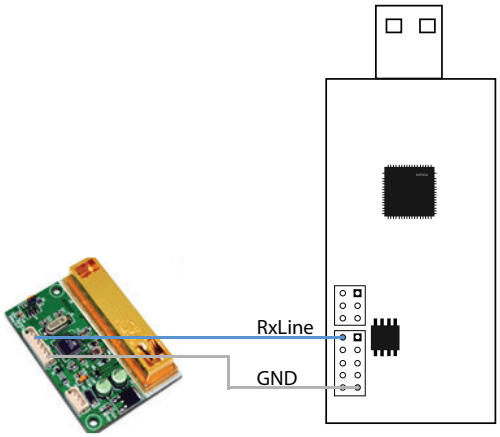


Figure 3.2: UART connection between two motes

The sensor board transmits periodically a packet with three values: temperature, CO2 and humidity. The structure of the packet could be found on the datasheet [7]. Figure **??** shows the packet format. It is important to remark that the board is sending 17 bytes, or characters that represent the different numbers.

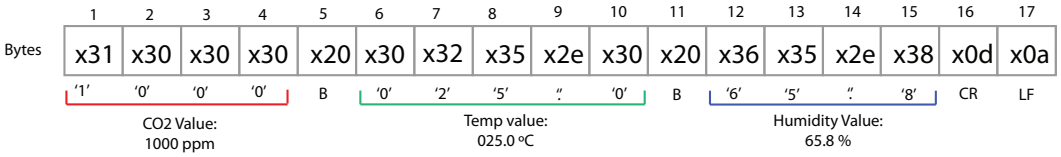


Figure 3.3: Packet format

3.3 Troubleshooting

Working with the serial communication (serial, I2C or Serial Peripheral Interface (SPI)), we have faced with different problems and we thought that is better to write them down.

I do not receive any message in the mote First of all, I recommend to check if the packet is sent and there is something that we should receive. For example, connect an oscilloscope to the transmission line of the transmitter. Secondly, a common mistake is that we forget to request the device and enable the interruptions for reception.

`call UartResource.request();` The `Resource` interface should be in the application. It allows us to share the resource with other components.

`call UartStream.enableReceiveInterrupt();` This command will enable the interruptions for reception. It only works if you own the resource.

I do no send any data through the transmission line If you do not see any data in the transmission line is probably because you do not own the resource. Be sure that you have the resource before transmission data by using:

```
call UartResource.isOwner();
```

I send data but it is not correct It could be because two things. One option could be that there is a mismatch between the receiver and the sender UART configuration: baud rate, parity, stop bit, ... Or it could be because of the data that you are sending. The send function needs an address as an argument.

Using the TKN15.4 implementation, the resource is never granted Depending on the configuration of the TKN15.4, the resource is always own by the implementation components. The main configuration is in the function `call MLME_SET.macRxOnWhenIdle(TRUE);`. If it is *TRUE* the implementation has the resource permanently, if it is *FALSE* the resource could be used in other components.

References

[1] *Inverted Pendulum Control over an IEEE 802.15.4 Wireless Sensor and Actuator Network*, Germany, feb. 2011. European Wireless Sensor Networks (EWSN).

[2] David Andreu and Aitor Hernandez. Debugging in wireless sensor networks for tinyos. Technical report, Royal Institute of Technology (KTH), July 2011.

[3] Aitor Hernandez. Communication between pc and motes in tinyos. Technical report, Royal Institute of Technology (KTH), July 2011.

[4] Aitor Hernandez. Getting started with tinyos at the automatic control lab. Technical report, Royal Institute of Technology (KTH), July 2011.

[5] Aitor Hernandez, Faisal Altaf, and Jose Araujo. Wireless crane. event triggered control. Technical report, Royal Institute of Technology (KTH), July 2011.

[6] Aitor Hernandez, Joao Faria, and Jose Araujo. Wireless water tanks. diferent scenarios and controllers. Technical report, Royal Institute of Technology (KTH), July 2011.

[7] Soha-Tech. Sh-300-dth - co2, temperature and humidity measurement. Technical report, Korea, 2010. URL http://www.soha-tech.com/.