# Networking Level Laboratory

## Mote-Mote Radio Communication


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

## Purpose/Objective:

The goal of this experiment is to understand various networking protocols a WSN can use; understand the difference between a base station/gateway, relay node, and the end nodes. Students will also get familiar with TinyOS interfaces and components that support radio communication. Students will learn how to use `message_t`, the TinyOS message buffer, how to send a message buffer to the radio, and what to do after receiving a message buffer from the radio.

## Introduction



The nodes in a wireless sensor networks can be configured taking a **mesh** network protocol, where every node is capable of sending and receiving data from other nodes; or taking **star** network protocol, where all nodes communicate with each other through a special node – cluster head. In general, the cluster head has more remaining power. Another special node is the 'base station' or 'gateway', through which the wireless communication is bridged to wired backbone network through a computer.

In order for nodes to communication with each other through radio/wireless communication, TinyOS supports a common message buffer abstraction, `message_t`, an *abstract data type* defined in `tos/types/message.h` file. TinyOS also supports a number of **interfaces** to abstract the underlying communications services and a number of **components** to provide/implement these interfaces. Within the structure `message_t`, developers are recommended to access the data field only through `Packet`, `AMPacket` and other interfaces, unless you **REALLY** know what you are doing:

```
typedef nx_struct message_t {
  nx_uint8_t header[sizeof(message_header_t)];
  nx_uint8_t data[TOSH_DATA_LENGTH];
  nx_uint8_t footer[sizeof(message_footer_t)];
  nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t
```

### Basic Communication Interfaces

Let's look at some of the interfaces in the `tos/interfaces` directory to familiarize ourselves with the communications system supported by TinyOS:

1. `Packet` - This interface provides commands for clearing a message's contents, getting its payload length, and getting a pointer to its payload area. It provides the basic mechanism to access members of the `message_t` structure.
2. `Send` - Provides the basic *address-free* **message sending** interface. This interface provides **commands** for sending a message, or canceling a pending message send; an **event** to indicate whether a message was sent successfully or not. It also provides functions for getting the message's maximum payload and a pointer to a message's payload area.
3. `Receive` - Provides the basic **message reception** interface. This interface provides an **event** for receiving messages; and **commands** for getting a message's payload length and a pointer to a message's payload area.

4. `PacketAcknowledgements` - Provides a mechanism for requesting acknowledgements (ACK) on a per-packet basis.
5. `RadioTimeStamping` - Provides time stamping information for radio transmission and reception.

## Active Message Interfaces

TinyOS provides the **Active Message** (AM) layer to multiplex access to the radio so that the same radio can be used by multiple services. The term "**AM type**" refers to the field used for multiplexing. It functions similarly to that of the Ethernet frame type field, IP protocol field, and the UDP port in that it is used also to multiplex access to a communication service. An AM packet includes a destination field, "AM address", to address packets to particular motes. Interfaces to support the AM services, also located in the `tos/interfaces` directory, include:

- `AMPacket` - Similar to `Packet`, it provides commands to access members of `message_t` structure. This interface provides **commands** for *getting* a node's AM address, and type and destination of an AM packet. Commands are also provided for *setting* an AM packet's destination and type, and checking whether the destination is the local node.
- `AMSend` - Similar to `Send`, provides the basic Active Message sending interface. The key difference between `AMSend` and `Send` is that `AMSend` takes a destination AM address in its `Send` command.

## Components

A number of components (located in `/tos/system` directory) implement the basic communications and active message (AM) interfaces. You need to get familiar with those interfaces and the components implementing them:

1. `AMReceiverC` - Provides the following interfaces: `Receive`, `Packet`, and `AMPacket`.
2. `AMSenderC` - Provides `AMSend`, `Packet`, `AMPacket`, and `PacketAcknowledgements` as `ACKs`.
3. `AMSnooperC` - Provides `Receive`, `Packet`, and `AMPacket`.
4. `AMSnoopingReciverC` - Provides `Receive`, `Packet`, and `AMPacket`.

## Naming Wrappes

Since TinyOS supports multiple platforms, each of which might have their own implementation of the radio drivers, an additional, platform-specific, naming wrapper called `ActiveMessageC` is used to bridge these interfaces to their underlying, platform-specific implementations. `ActiveMessageC` provides most of the communication interfaces presented above. `ActiveMessageC` for the micaz, telosa, telosb, and intelmote2 are all implemented by `CC2420ActiveMessageC`.

## Experiment Procedure:

In this experiment, we use BlinkToRadio applications (located in $TOSROOT/apps/). This simple application increments a counter, displays the counter's three least significant bits on the three LEDs, and sends a message with the counter value over the radio when a timer fires. Similar to the Blink

application, only one single timer and one counter will be used here. We will program two node with this application. As long as they are both within range of each other, the LEDs on both will keep blinking. If the LEDs on one (or both) of the nodes stops changing and holds steady, then that node is no longer receiving any messages from the other node.

**Step 1: Study the program files in the BlinkToRadio application folder:**

1. Identify how message structure is defined.
2. Study the code for **sending** a message over the Radio:
   1) Identify the interfaces (and components) that provide access to the radio and allow us to manipulate the `message_t` type – `AMSend`;
   2) Identify in the `module` block of the `BlinkToRadioC.nc` file the `uses` statements for the interfaces needed to send a message;
   3) Identify the declaration of any new variables and initialization code for sending a message;
   4) Identify the program and calls to interfaces needed for sending a message;
   5) Identify the implementation of any (non-intialization) events specified in the interfaces needed for sending a message;
   6) Identify in the `implementation` block of the application configuration file all the components statement needed to provide the interfaces chosen in step 5);
   7) Identify the 'wire' used by the application to the components providing the interfaces in the `implementation` block of the application configuration file.
3. Study the code for **Receiving** a message over the Radio:
   1) Identify the interfaces (and components) that provide access to the radio and allow us to manipulate the `message_t` type – `Receive`;
   2) Identify in the `module` block of the `BlinkToRadioC.nc` file the `uses` statements for the interfaces needed to receive a message;
   3) Identify any declaration of new variables and initialization code for receiving a message;
   4) Identify any program and calls to interfaces needed for receiving a message;
   5) Identify the implementation of any (non-intialization) events specified in the interfaces needed for receiving a message;
   6) Identify in the `implementation` block of the application configuration file all the components statement needed to provide the interfaces chosen in step 5);
   7) Identify the 'wire' used by the application to the components providing the interfaces in the `implementation` block of the application configuration file.

**Step 2: Program Mica nodes with BlinkToRadio application:**

1. Attach the first Mica node securely onto the programming board (MIB520).
2. Change the terminal's active directory to '**/opt/tinyos-2.x/apps/tutorials/BlinkToRadio**'. Use command '**make micaz install.1 mib510,/dev/ttyUSB0**' to load the `BlinkToRadio` onto the first Mica node.
3. Detach the first Mica node, and attach the second node onto the programming board securely.
4. Use command '**make micaz reinstall.2 mib510,/dev/ttyUSB0**' to load the `BlinkToRadio` onto the second Mica node.
5. Detach the second Mica node from the programming board.

**Step 3: Test the BlinkToRadio application!**

1. Put in the batteries for two Micaz nodes.
2. Turn the switch from 'OFF' to 'ON' on both nodes.
3. Wait and see the LEDs blinking!

**Congratulations!** You just figured out how to send and receive data on a Mica node via Radio Communication successfully!

## Exercise:

1. With every firing of the timer, three things will happen. What are they?

2. Go through the BlinkToRadioC.nc file, and answer following questions: (1) Why instead of using `ActiveMessageC` component to access the `message_t`, the program uses `AMSenderC`? (2) Why the radio is started using `ActiveMessageC.SplitControl` interface?

3. GO through the `BlinkToRadio.h` file, what does AM_BLINKTORADIO parameter indicate?

### References:

TinyOS Tutorial Lesson 3: http://docs.tinyos.net/tinywiki/index.php/Mote-mote_radio_communication