实验一 节点-节点无线通信实验

实验目的

本实验介绍了如何在 TinyOS 上进行节点与节点之间的无线通信。通过这个实验，熟悉通信相关的组件及接口以及如何以单播的方式发送和接收消息。

实验要求

根据提供的例子程序，详细了解程序结构，并尝试进行程序的修改运行。具体实验要求如下：

1、熟悉 TinyOS 无线通信的接口和通信流程；

2、修改例子程序，具体要求见后。

实验内容

1、基本概念介绍

TinyOS 提供了许多接口来抽象底层的通讯服务，并且包含了许多提供这些接口的组件。这些接口和组件都使用了一个共同的消息抽象－message_t。这是一个 nesC 的结构体，如下：

```
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;
```

一些典型的使用了 message_t 的接口如下：

- Packet：该接口提供了基本的操作 message_t 的功能，例如清楚消息内容，获取 payload 长度以及获取 payload 的地址指针等。
- Send：该接口提供了基本的不基于地址（address-free）的消息发送功能，例如发送一条消息以及取消一条待发消息的发送等。并且还提供了事件来提示发送是否成功。当然也提供了获取消息最大 payload 以及 payload 地址指针的功能。
- Receive：该接口提供了基本消息接收功能和获取 payload 信息的功能。
- PacketAcknowledgements：该接口提供了获取发送消息回执的机制。
- AMPacket：这个接口和 Packet 类似，提供了获取与设置一个节点的 AM 地址，AM 包的目的地址以及 AM 包的类型等功能。
- AMSend：这个接口和 Packet 类似，提供了获取与设置一个节点的 AM 地址，AM 包的目的地址以及 AM 包的类型等功能。

典型的提供了以上接口的组件有：

- AMReceiverC：提供了 Receive，Packet，AMPacket 接口。

- AMSenderC：提供了 AMSend，Packet，AMPacket 以及 PacketAcknowledge 接口。
- AMSnooperC：提供了 Receive，Packet 和 AMPacket 接口。
- AMSnoopingReceiverC：提供了 Receive，Packet 和 AMPacket 接口。
- ActiveMessageAddressC：提供了动态修改消息地址的命令。这个命令慎用，可能会导致网络奔溃。

2、消息发送

打开例子程序 BlinkToRadio，这个程序通过消息发送自身的计数器至对方，同时收到对方的消息后，解析出对方的计数器，按照这个计数器亮灯，使用单个 Timer 实现发送的频率间隔。

首先，我们定义数据传送的消息格式。消息包括两个部分：节点 ID 和计数值。

```
typedef nx_struct BlinkToRadioMsg
    {
       nx_uint16_t nodeid;
       nx_uint16_t counter;
    }BlinkToRadioMsg;
```

然后，现在要通过 Radio 把这条消息发送出去。

1)首先，确认使用的接口和组件。AMSenderC 组件来提供 AMSend 和 Packet 接口，使用 AMSend 接口来发送包，使用 Packet 接口来操作 message_t。使用 ActiveMessageC 提供的 SplitControl 接口来启动 Radio。因此在 BlinkToRadioC.nc 中可以看到如下声明：

```
module BlinkToRadioC
    {
       ...
       uses interface Packet;
       uses interface AMSend;
       uses interface SplitControl as AMControl;
    }
```

2)需要声明一些新的变量。message_t 用来数据传输，busy 用来标志是否在传输中。在 BlinkToRadioC.nc 中如下：

```
implementation
    {
       bool busy = FALSE;
       message_t pkt;
       ...
```

```
        }
```

3)处理 Radio 的初始化：

```
    event void Boot.booted()
        {
          call AMControl.start();
        }
```

4)实现 AMControl.startDone 和 AMControl.stopDone 这两个事件处理器如下：

```
        event void AMControl.startDone(error_t err)
        {
            if (err == SUCCESS)
              {
                  call
                  Timer0.startPeriodic(TIMER_PERIOD_MILLI);
              }else{
                  all AMControl.start();
              }
        }
```

5)发送消息逻辑。在 Timer0.fired 中添加代码：

```
        event void Timer0.fired()
        {
            ...
            if (!busy) {
            BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)(call
Packet.getPayload(&pkt, sizeof                (BlinkToRadioMsg)));
            btrpkt->nodeid = TOS_NODE_ID;
            btrpkt->counter = counter;
            if (call AMSend.send(AM_BROADCAST_ADDR, &pkt,
sizeof(BlinkToRadioMsg)) == SUCCESS) {
                    busy = TRUE;
                }
            }
        }
```

6)消息发送完毕后，清除忙标志位。

```
event void AMSend.sendDone(message_t* msg, error_t error) {
    if (&pkt == msg) {
        busy = FALSE;
    }
}
```

7)为每个提供接口的组件添加组件声明，其中 AM_BLINKTORADIO 参数表明 AMSenderC 的 AM 类型。在头文件中有定义。

```
implementation{
   ...
   components ActiveMessageC;
   components new AMSenderC(AM_BLINKTORADIO);
   ...
}
```

8)将接口的提供方和使用方连接起来。

```
implementation{
   ...
   App.Packet -> AMSenderC;
   App.AMPacket -> AMSenderC;
   App.AMSend -> AMSenderC;
   App.AMControl -> ActiveMessageC;
}
```

3、消息接受

接收到消息后，首先解析出消息中的计数器，然后计数器按照这个计数值的低三位亮灯，具体过程如下：

1) 使用 Receive 接口来接收包。在 BlinkToRadioC.nc 文件中，添加以下声明：

```
module BlinkToRadioC {
       ...
       uses interface Receive
    }
```

2)实现接口 Receive.receive 事件处理：

```
event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
```

```
len) {
            if (len == sizeof(BlinkToRadioMsg)) {
               BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
               call Leds.set(btrpkt->counter);
              }
               return msg;
            }
```

3)添加 Receive 接口对应的组件声明。

```
        implementation
            {
              ...
              components new AMReceiverC(AM_BLINKTORADIO);
              ...

            }
```

4)把接口的提供方与使用方连接起来。

```
        implementation
            {
              ...
              App.Receive -> AMReceiverC;

            }
```


5)测试程序

分别使用 make telosb install,1 以及 make telosb install,2 烧录两个节点。通电后查看效果。当按住某一个节点的 RESET 键时，另一个节点读数应当停止。

实验要求说明

实现类似跑马灯的效果，两个节点都维护计数值 counter，初始值为 1。节点 1 每隔 1 秒发送计数值到节点 2，节点 2 每隔 1 秒发送自身计数值＋1 到节点 1。节点 1 和节点 2 均接收对方发送的计数值，收到后更新 counter，用 LED 灯显示 counter 的末三位。

效果：节点 1，节点 2 都开着的时候，节点 1 和节点 2 的 LED 灯显示的值每隔 1 秒递增 1。此时按住其中一个节点的 RESET，另外一个节点的显示将不会停住不再变化，放开之后，又重新从 1 开始计数。提示：需要分辨节点的编号以发送不同的计数值。

实验过程说明

首先需要修改节点通信的相关信道，避免相互之间影响，如根据组号选择 26 信道，在 Makefile 文件中进行修改，PFLAGS+=-DCC2420_DEF_CHANNEL=26。在实验中分配的两个节点编号如 1， 2，分别对应上述实验要求中的节点 1、2，修改配置文件的相关变量。

```
    enum  {
  AM_POINTTOPOINT  =  6,
  TIMER_PERIOD_MILLI = 1000,
  NODE_ID_1 = 1,
  NODE_ID_2 = 2
};
```