

华中科技大学

课程实验报告

课程名称: 物联网通信技术实验

专业班级: IOT1601

学 号: U201614897

姓 名: 潘越

指导教师: 徐海银

报告日期: 2018年12月24日

计算机科学与技术学院

目 录

实验 1 节点-节点单波无限通信实验	1
1.1 实验目的	1
1.2 实验内容	1
1.3 实验过程	1
1.3.1 开发环境	1
1.3.2 实验步骤	1
1.4 实验结果与分析	3
1.4.1 程序测试	3
1.4.2 结果分析	5
1.5 心得体会与总结	5
实验 2 节点-PC 串口通信实验	6
2.1 实验目的	6
2.2 实验内容	6
2.3 实验过程	6
2.3.1 开发环境	6
2.3.2 实验步骤	6
2.4 实验结果与分析	11
2.4.1 程序测试	11
2.4.2 结果分析	13
2.5 心得体会与总结	15
实验 3 静态路由转发实验	16
3.1 实验目的	16
3.2 实验内容	16
3.3 实验过程	16
3.3.1 开发环境	16
3.3.2 实验步骤	16
3.4 实验结果与分析	19
3.4.1 程序测试	19
3.4.2 结果分析	19
3.5 心得体会与总结	25
附录 实验源代码	26
附录.1 实验目录结构	26
附录.2 实验一程序源代码	26
附录.2 实验二程序源代码	33
附录.3 实验三程序源码	44
参考文献	51

实验 1 节点-节点单波无限通信实验

1.1 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。

1.2 实验内容

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 无限通信的接口和通信流程。

(2) 修改例子程序，具体要求如下：

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。

1.3 实验过程

1.3.1 开发环境

本次实验中使用的环境配置如下：

- (1) 操作系统版本：Arch Linux x86_64
- (2) TinyOS 版本：TinyOS 3.0.0
- (3) 编译器及其版本：GCC version 8.2.1
- (4) 编程环境：Visual Studio Code

1.3.2 实验步骤

本次实验尝试从源码 (<https://github.com/tinyos/tinyos-main>) 安装 TinyOS 工具链，使用最新版本的 TinyOS，因此过程会与使用 TinyOS 2.1.2 版本有所区别。

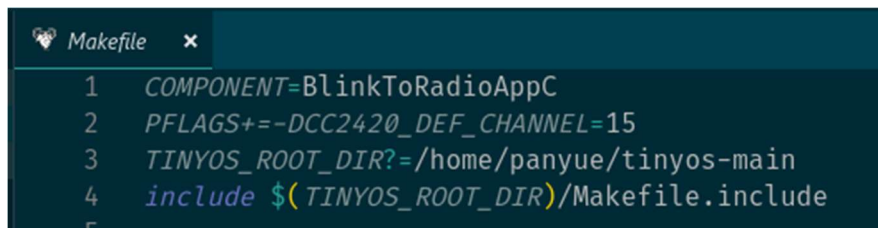
① 首先进入 TinyOS 的 BlinkToRadio 例程目录如下：

`tinycos-main/apps/tutorials/BlinkToRadio`

BlinkToRadio 可以实现以无线方式收发包。一个节点，每隔一秒会将自身的

counter 值加 1，然后通过无线传输的方式发给别的节点。其他节点收到该 counter 值，会通过节点的 led 灯显示该值。

②接着修改 Makefile 如下图



```

1 COMPONENT=BlinkToRadioAppC
2 PFLAGS+=-DCC2420_DEF_CHANNEL=15
3 TINYOS_ROOT_DIR?=/home/panyue/tinyos-main
4 include $(TINYOS_ROOT_DIR)/Makefile.include
5

```

图 1.1 Makefile 修改

其中，加上 PFLAGS+=-DCC2420_DEF_CHANNEL=15 这一句是为了设置信道，防止不同同学实验之间的相互干扰。

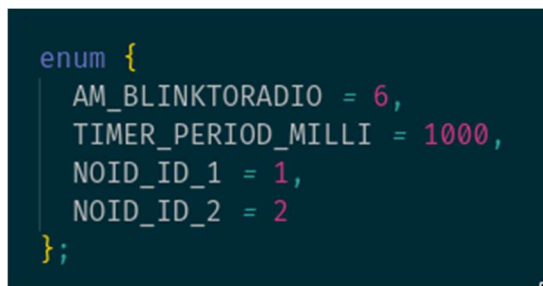
而如下的两句则是 TinyOS 3.0.0 版本以后的环境变量指定，从该版本起不再进行全局环境变量的配置，而改由 tinyos-main 目录下的 Makefile.include 文件指定，在编译时只需包含这个文件即可。

```

TINYOS_ROOT_DIR?=/home/panyue/tinyos-main
include $(TINYOS_ROOT_DIR)/Makefile.include

```

③接着修改定时器的时延，在 BlinkToRadio.h 中，定义如下图的常量：



```

enum {
    AM_BLINKTORADIO = 6,
    TIMER_PERIOD_MILLI = 1000,
    NOID_ID_1 = 1,
    NOID_ID_2 = 2
};

```

图 1.2 Makefile 修改

将 TIMER_PERIOD_MILLI 改为 1000，即 1s，同时使用两个常量标记两个节点的 ID。

④在每个节点发送的数据中，加上自己的 ID，而在接受数据时，只接受对方节点发送的数据，如下图：

```

event void Timer0.fired() {
    counter++;
    if (!busy) {
        BlinkToRadioMsg* btrpkt =
        (BlinkToRadioMsg*)(call Packet.getPayload(&pkt, sizeof(BlinkToRadioMsg)));
        if (btrpkt == NULL) {
            return;
        }
        btrpkt->nodeid = NOID_ID_1;
        btrpkt->counter = counter;
        if (call AMSend.send(AM_BROADCAST_ADDR,
            &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
            busy = TRUE;
        }
    }
}

```

图 1.3 发送数据函数

```

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
        if (btrpkt->nodeid == NOID_ID_2)
            setLeds(btrpkt->counter);
    }
    return msg;
}

```

图 1.4 接收数据函数

⑤对于节点 2，注意修改发送的数据为 `counter + 1` 即可。

至此程序编写完成，可以进行代码的烧录。

1.4 实验结果与分析

1.4.1 程序测试

完成代码编写后，在目录下执行 `make telosb install /dev/ttyUSB0`，编译程序并烧录至节点上，如下图：

```

[INFO] script
      11308 bytes in ROM
      368 bytes in RAM
[INFO] size (toolchain):
      text  data  bss  dec  hex filename
      11402   20   350  11772  2dfc build/telosb/main.exe
[INFO] generating symbol table
[INFO] generating listing
[INFO] creating ihex file
[INFO] writing TOS image
[INFO] writing TOS buildinfo
[INFO] running the wiring check
[INFO] found mote on /dev/ttyUSB0 ("using bsl,auto")
[INFO] installing telosb binary using bsl
tos-bsl --telosb -c /dev/ttyUSB0 -r -e -I -p build/telosb/main.ihex.out
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
11422 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out build/telosb/main.ihex.out

```

图 1.5 编译并烧录程序

同样的再将另一个程序烧录到节点 2 上，将两个电源打开，即可看到如下的结果：

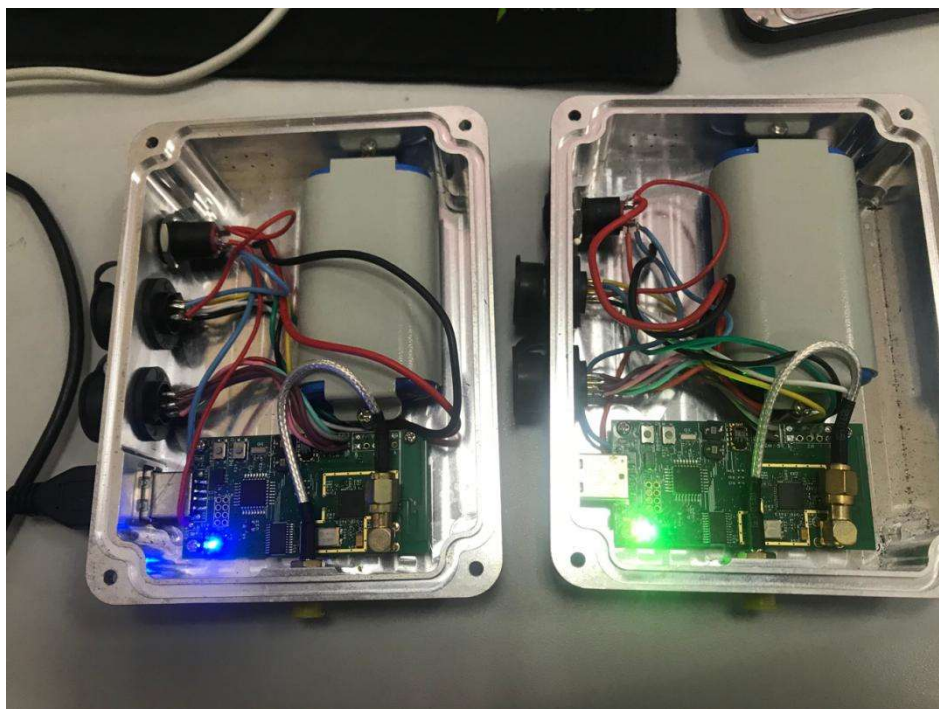


图 1.6 实验一跑马灯程序演示

关闭其中一个节点的电源，可以看到如下的结果：

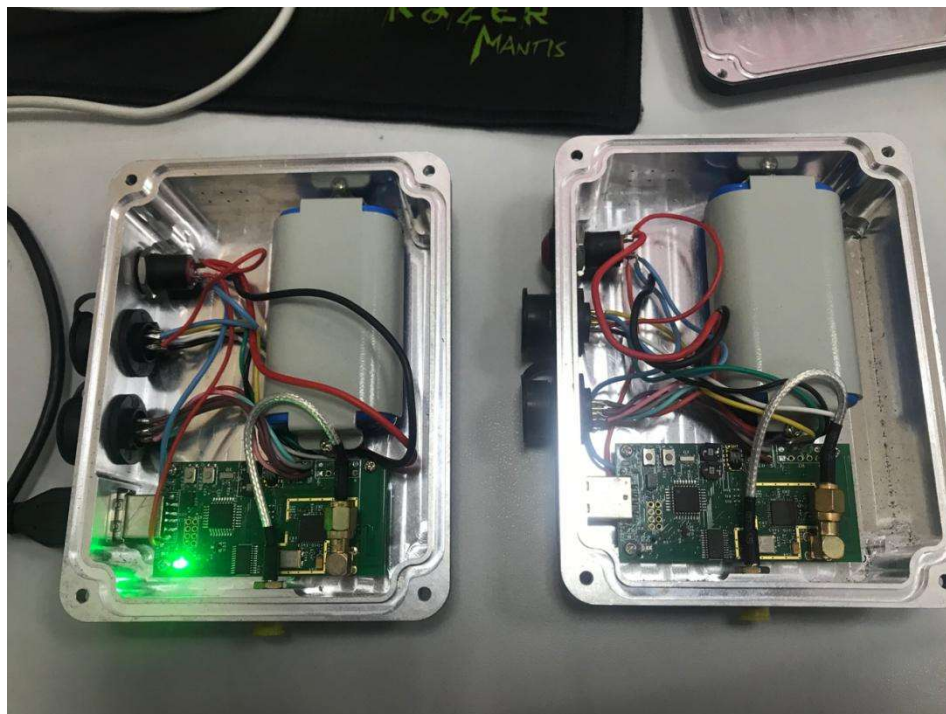


图 1.7 关闭其中一个节点的电源

1.4.2 结果分析

观察实验结果可以发现：

（1）当两个节点的电源都打开时，两个节点轮流按照蓝绿红的顺序亮灯，这是程序中节点 2 发送 `counter+1` 的效果体现。

（2）两个节点也出现了少数时间显示同样的颜色，这是网络发送数据的延迟的体现，属于正常现象。

（3）关闭其中一个节点之后，另一个节点不会再收到新的数据包，因此指示灯不再改变。

综合以上几点，验证我们的程序是符合实验要求，完全达到预期的实验目的的。

1.5 心得体会与总结

通过本次实验，时隔半年我又熟悉了一遍 TinyOS 的编程方法，加深了知识的掌握。同时在本次实验中使用全新的 TinyOS 3.0.0 版本，了解了新版本 TinyOS 工具链的变化和编译系统的变化，也为后面的实验打好了基础。

实验 2 节点-PC 串口通信实验

2.1 实验目的

本实验的目的是实现节点和 PC 间的串口双向通讯，通过串口连接，PC 可以从网络收集其他节点的数据，也可以发送数据或者命令到节点，因此，串口通信编程是无线传感器网络中的重要内容。

2.2 实验内容

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) 学会使用串口通信相关的接口函数，实现串口通信。
- (2) 修改 `BlinkToRadio` 程序，添加串口收发，实现一个简单的基站功能。
- (3) 了解串口双向通信的方法，学会使用 `mig` 工具以及 `SerialForwarder`。

具体的说，实验要求完成以下内容：

(1) 修改 `BlinkToRadio` 程序实现简单的基站程序 `RadioAndSerial`。使得当 `RadioAndSerial` 接收到无线数据包时 `Led2` 闪烁，并将数据包转发到串口；当串口接收到数据包时，`Led0` 闪烁，并将数据包转发到无线模块；

(2) 使用 `mig` 创建 `BlinkToRadioMsg` 的 java 对象，`BlinkToRadio` 发送的消息由 `RadioAndSerial` 接收并转发到串口，然后使用 `MsgReader` 读取 `BlinkToRadioMsg` 对象，展示接收到的数据。

扩展要求：实现 `BlinkToRadio.java`，进行 PC 和串口之间的双向通信，从而实现利用 PC 发送消息控制 `BlinkToRadio` 节点 `Led` 灯的闪烁。

2.3 实验过程

2.3.1 开发环境

本次实验中使用的环境配置如下：

- (1) 操作系统版本：Arch Linux x86_64
- (2) TinyOS 版本：TinyOS 3.0.0
- (3) 编译器及其版本：GCC version 8.2.1
- (4) 编程环境：Visual Studio Code

2.3.2 实验步骤

①首先根据实验指导验证例程的正确性。

进入 TinyOS 的 `TestSerial` 例程目录如下：

```
tinynos-main/apps/tests/TestSerial
```


执行命令，编译并烧录程序：

```
make telosb install /dev/ttyUSB0
```

接着运行 java 程序：

```
java TestSerial -comm serial@/dev/ttyUSB0:telosb
```

可以看到如下的输出：

```
The operating system is 'Linux' (amd64)
Trying to locate the file 'linux_amd64_toscomm.lib' in the classpath
Temporary file created: '/tmp/toscomm15401188766287341284.lib'
Library copied successfully. Let's load it.
Library loaded successfully
serial@/dev/ttyUSB0:115200: resynchronising
Sending packet 0
Received packet sequence number 5
Sending packet 1
Received packet sequence number 6
Sending packet 2
Received packet sequence number 7
Sending packet 3
Received packet sequence number 8
Sending packet 4
Received packet sequence number 9
Sending packet 5
Received packet sequence number 10
Sending packet 6
Received packet sequence number 11
Sending packet 7
```

图 2.1 TestSerial 程序输出

②进入 TinyOS 的 BaseStation 例程目录如下：

```
tinyos-main/apps/BaseStation
```

将其中一个节点烧录上 BaseStation 例程，另一个节点烧录上 BlinkToRadio 例程，通电，可以看到如下的效果：

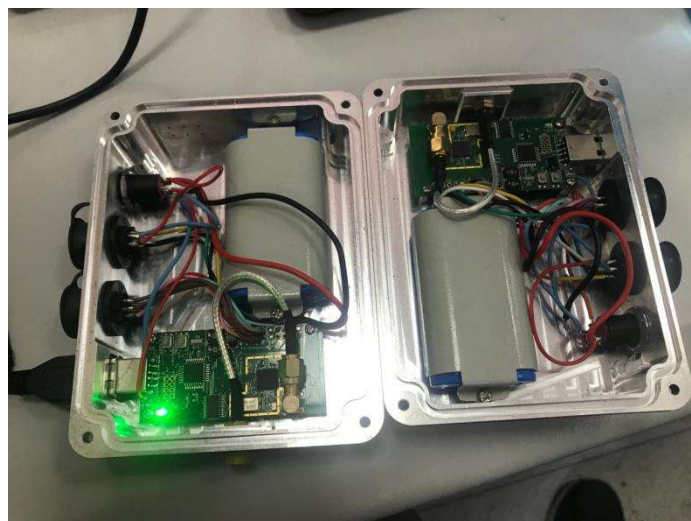


图 2.2 例程演示

观察到烧录了 BaseStation 的节点一直闪烁（因为收到了消息），当按下烧录了 BlinkToRadio 的节点的 reset 键之后，BaseStation 就停止闪烁（因为没有收到消息）。

在 PC 上执行如下命令，接收串口消息：

```
java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB0:telosb
```

可以看到如下的输出：

```
The operating system is 'Linux' (amd64)

Trying to locate the file 'linux_amd64_toscomm.lib' in the classpath
Temporary file created: '/tmp/toscomm14114008735814317340.lib'
Library copied successfully. Let's load it.
Library loaded successfully
serial@/dev/ttyUSB0:115200: resynchronising
00 FF FF 00 01 04 22 06 00 01 00 3D
00 FF FF 00 01 04 22 06 00 01 00 3E
00 FF FF 00 01 04 22 06 00 01 00 3F
00 FF FF 00 01 04 22 06 00 01 00 40
00 FF FF 00 01 04 22 06 00 01 00 41
00 FF FF 00 01 04 22 06 00 01 00 42
00 FF FF 00 01 04 22 06 00 01 00 43
00 FF FF 00 01 04 22 06 00 01 00 44
00 FF FF 00 01 04 22 06 00 01 00 45
00 FF FF 00 01 04 22 06 00 01 00 46
00 FF FF 00 01 04 22 06 00 01 00 47
00 FF FF 00 01 04 22 06 00 01 00 48
00 FF FF 00 01 04 22 06 00 01 00 49
00 FF FF 00 01 04 22 06 00 01 00 4A
00 FF FF 00 01 04 22 06 00 01 00 4B
00 FF FF 00 01 04 22 06 00 01 00 4C
00 FF FF 00 01 04 22 06 00 01 00 4D
```

图 2.3 串口消息输出

③接着就可以开始设计实验，实验的程序框架设计如下图：

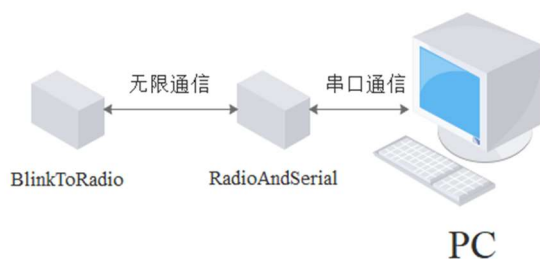


图 2.4 实验二程序框架

烧录有 BlinkToRadio 的节点发送消息到烧录有 RadioAndSerial 的基站节点，

基站节点收到消息后将其转发到串口，由 BlinkToRadio.java 接收，同时 PC 可以通过 BlinkToRadio.java 发送消息到串口，基站节点接收后将数据转发给 BlinkToRadio 节点。

④编写基站程序 RadioAndSerial

根据实验要求，本程序实现的比较简单，没有考虑消息队列和时延等问题，在收到无线消息后就立即将其转发到串口，在收到串口的消息后立即将其转发给相应的节点。

基站程序使用到的接口如下所示：

```
module RadioAndSerial {
    uses interface Boot;
    uses interface Leds;
    // Radio
    uses interface Packet as RadioPacket;
    uses interface AMSend as RadioAMSend;
    uses interface Receive as RadioReceive;
    uses interface SplitControl as RadioAMControl;
    // Serial
    uses interface Packet as SerialPacket;
    uses interface AMSend as SerialAMSend;
    uses interface Receive as SerialReceive;
    uses interface SplitControl as SerialControl;
}
```

图 2.5 基站程序使用到的接口

这里参考了 BlinkToRadio 例程和 TestSerial 例程，分开处理无线消息和串口消息。

程序的核心逻辑在于两个转发函数，其实现如下：

```
event message_t* RadioReceive.receive(message_t* msg, void* payload, uint8_t len) {
    if (!busy_r && len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* ppkt = (BlinkToRadioMsg*)payload;
        BlinkToRadioMsg* new_pkt = (BlinkToRadioMsg*)(call SerialPacket.getPayload(&packet, sizeof(BlinkToRadioMsg)));
        if (new_pkt == NULL) {
            return msg;
        }
        getRadioPkt();
        new_pkt->nodeid = ppkt->nodeid;
        new_pkt->counter = ppkt->counter;
        if (call SerialAMSend.send(new_pkt->nodeid, &packet, sizeof(BlinkToRadioMsg)) == SUCCESS) {
            busy_s = TRUE;
        }
    }
    return msg;
}
```

图 2.6 收到无线消息的处理

```
event message_t* SerialReceive.receive(message_t* msg, void* payload, uint8_t len) {
    if (!busy_s && len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* ppkt = (BlinkToRadioMsg*)payload;
        BlinkToRadioMsg* new_pkt = (BlinkToRadioMsg*)(call RadioPacket.getPayload(&packet, sizeof(BlinkToRadioMsg)));
        if (new_pkt == NULL) {
            return msg;
        }
        getSerialPkt();
        new_pkt->nodeid = ppkt->nodeid;
        new_pkt->counter = ppkt->counter;
        if (call RadioAMSend.send(new_pkt->nodeid, &packet, sizeof(BlinkToRadioMsg)) == SUCCESS) {
            busy_r = TRUE;
        }
    }
    return msg;
}
```

图 2.7 收到串口消息的处理

最后记得按照实验要求在 Makefile 中加上使用 26 号信道。

```
COMPONENT=RadioAndSerialAppC
PFLAGS+=-DCC2420_DEF_CHANNEL=26
TINYOS_ROOT_DIR?=/home/panyue/tinyos-main
include $(TINYOS_ROOT_DIR)/Makefile.include
```

图 2.8 基站程序 Makefile

⑤修改 BlinkToRadio 程序

首先仿照 TestSerial 例程中的 TestSerial.java 程序，完成 BlinkToRadio.java，用作和串口之间进行双向通信的程序。

程序的核心逻辑是消息收发函数，其实现如下：

```
public void sendPackets(int nodeid, int counter) {
    BlinkToRadioMsg payload = new BlinkToRadioMsg();
    try {
        payload.set_nodeid(nodeid);
        payload.set_counter(counter);
        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) {
    BlinkToRadioMsg msg = (BlinkToRadioMsg)message;
    System.out.println("\nReceived packet to: " + to + " nodeid: " + msg.get_nodeid() + " counter: " + msg.get_counter());
    System.out.println("Input the nodeid and counter(like 1 2): ");
    Scanner sc = new Scanner(System.in);
    int nodeid = sc.nextInt();
    int counter = sc.nextInt();
    System.out.println("Sending to " + nodeid + " number is: " + counter);
    sendPackets(nodeid, counter);
}
```

图 2.9 BlinkToRadio.java 消息收发函数

这里实现为，每收到一个消息，打印出其内部的信息（节点编号和计数器的值），同时输入要发送的数据节点和计数器的值。

接着修改 Makefile，使用 mig 工具创建 BlinkToRadioMsg 的 java 对象，如下图所示：


```
COMPONENT=BlinkToRadioAppC
PFLAGS+=-DCC2420_DEF_CHANNEL=26
TOSMAKE_PRE_EXE_DEPS += BlinkToRadio.class
TOSMAKE_CLEAN_EXTRA = *.class BlinkToRadioMsg.java

BlinkToRadio.class: $(wildcard *.java) BlinkToRadioMsg.java
    javac -target 1.6 -source 1.6 *.java

BlinkToRadioMsg.java:
    nesc-mig java $(CFLAGS) -java-classname=BlinkToRadioMsg BlinkToRadio.h BlinkToRadioMsg -o $@

TINYOS_ROOT_DIR?=/home/panyue/tinyos-main
include $(TINYOS_ROOT_DIR)/Makefile.include
```

图 2.10 BlinkToRadio 程序 Makefile

最后要注意，为了体现出程序的效果，在 BlinkToRadio 程序中修改一个细节，就是在收到消息后，不仅根据收到的消息中的计数器值改变灯的闪烁情况，而且将自己的计数器值置为收到的消息中的计数器的值。

至此，程序编写完成，可以进行代码的烧录。

2.4 实验结果与分析

2.4.1 程序测试

完成代码编写后，在目录下执行 `make telosb install /dev/ttyUSB0`，编译程序并烧录至两个节点上。

其中，RadioAndSerial 节点使用 USB 串口连接到 PC，另一个节点烧录 BlinkToRadio，并打开电源，可以看到如下的结果：

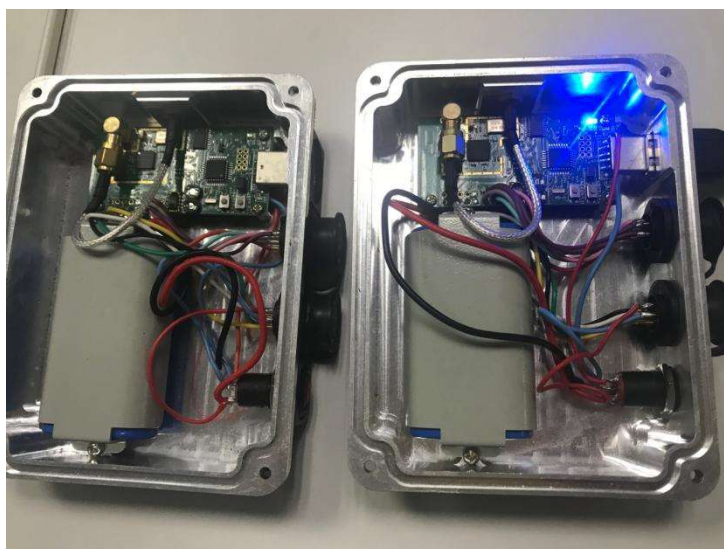


图 2.11 节点运行结果

我们在 BlinkToRadio 中设置的发送消息的间隔是 1s，观察发现每当基站程序收到一个消息时，就会改变蓝灯的状态。

在 PC 的命令行中运行如下程序：

```
java net.tinyos.tools.MsgReader -comm serial@/dev/ttyUSB0:telosb BlinkToRadioMsg
```

可以看到如下的输出，可以看到 MsgReader 直接输出的消息中的值。

```
Trying to locate the file 'linux_amd64_toscomm.lib' in the classpath
Temporary file created: '/tmp/toscomm8098922367544020376.lib'
Library copied successfully. Let's load it.
Library loaded successfully
serial@/dev/ttyUSB0:115200: resynchronising
1545201572620: Message <BlinkToRadioMsg>
  [nodeid=0x1]
  [counter=0x6]

1545201573587: Message <BlinkToRadioMsg>
  [nodeid=0x1]
  [counter=0x7]

1545201574560: Message <BlinkToRadioMsg>
  [nodeid=0x1]
  [counter=0x8]

1545201575533: Message <BlinkToRadioMsg>
  [nodeid=0x1]
  [counter=0x9]
```

图 2.12 MasReader 输出结果

接着使用我们编写的 BlinkToRadio.java 程序，执行如下的命令：

```
java BlinkToRadio -comm serial@/dev/ttyUSB0:telosb
```

可以看到如下的输出：

```
Trying to locate the file 'linux_amd64_toscomm.lib' in the classpath
Temporary file created: '/tmp/toscomm15546378771516894471.lib'
Library copied successfully. Let's load it.
Library loaded successfully
serial@/dev/ttyUSB0:115200: resynchronising
Received packet to: 1 nodeid: 1 counter: 2
Input the nodeid and counter(like 1 2):
1 9
Sending to 1 number is: 9

Received packet to: 1 nodeid: 1 counter: 3
Input the nodeid and counter(like 1 2):
1 9
Sending to 1 number is: 9

Received packet to: 1 nodeid: 1 counter: 4
Input the nodeid and counter(like 1 2):
1 1
Sending to 1 number is: 1

Received packet to: 1 nodeid: 1 counter: 10
Input the nodeid and counter(like 1 2):
1 1
Sending to 1 number is: 1

Received packet to: 1 nodeid: 1 counter: 11
Input the nodeid and counter(like 1 2):
1 4
Sending to 1 number is: 4

Received packet to: 1 nodeid: 1 counter: 10
Input the nodeid and counter(like 1 2):
1 5
Sending to 1 number is: 5

Received packet to: 1 nodeid: 1 counter: 2
```

图 2.13 BlinkToRadio.java 输出结果

同时可以看到两个节点的指示灯如下图：



图 2.14 节点运行结果

基站节点在收到串口消息时会改变红灯的状态，BlinkToRadio 节点在收到消息时，会根据消息中的计数器的值改变自己的指示灯状态。

2.4.2 结果分析

观察实验结果可以发现：

- (1) BlinkToRadio 节点确实每隔一秒发送一个消息，同时计数器的值+1，这一点可以在收到的数据输出中看出。
- (2) 串口消息发送是正常的，每发送一次串口消息，基站程序的红灯改变一次状态。
- (3) 双向通信是正常的，因为从 BlinkToRadio.java 的输出中可以看出，在我们向 BlinkToRadio 节点发送(1,9)之后收到的第三个数据包中，计数器的值已经变成了 10，而在发送(1,1)之后的收到的第四个数据包中，计数器的值已经变成了 2，综合验证双向通信完全正常。
- (4) 至于收到的消息打印出来后，发生的计数器变化延迟的情况，猜测是由于 PC 端进行了串口数据包的缓存，由于每隔一秒就会收到一个数据包，而我们输入数据比较慢，因此打印出来的就是之前缓存的已收到的数据包，所以才会出现计数器变化延迟的现象。
- (5) 根据计数器的值，改变指示灯的功能正常，而且可以看到这个指示灯的改变是在发送消息后马上实现的，没有明显延迟，下面列出几个结果。

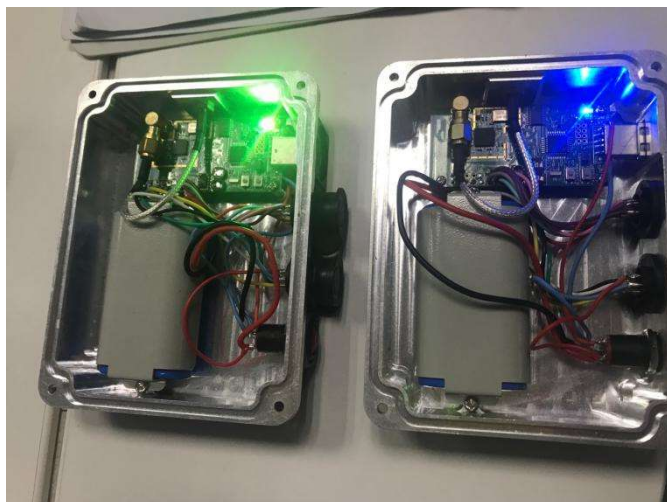


图 2.15 发送数据 3

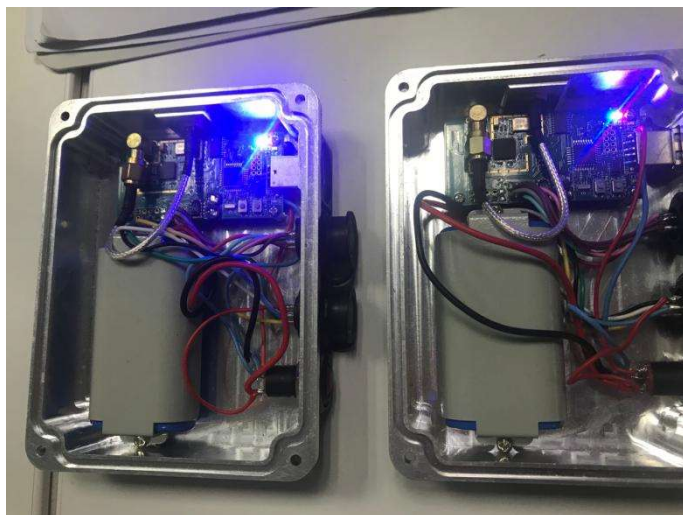


图 2.16 发送数据 5

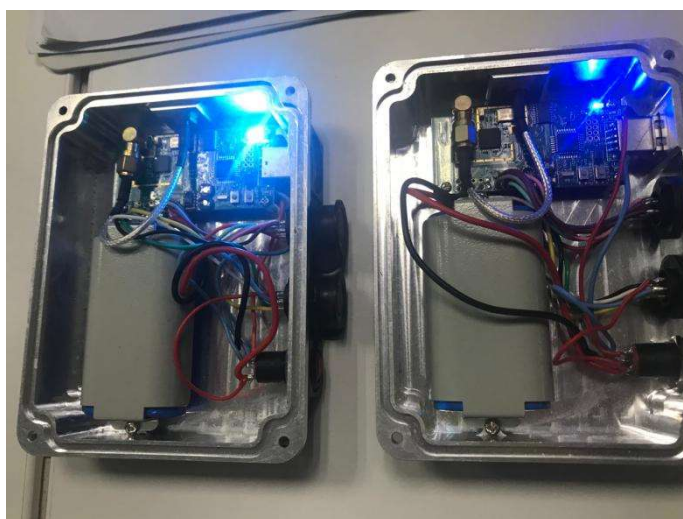


图 2.17 发送数据 7

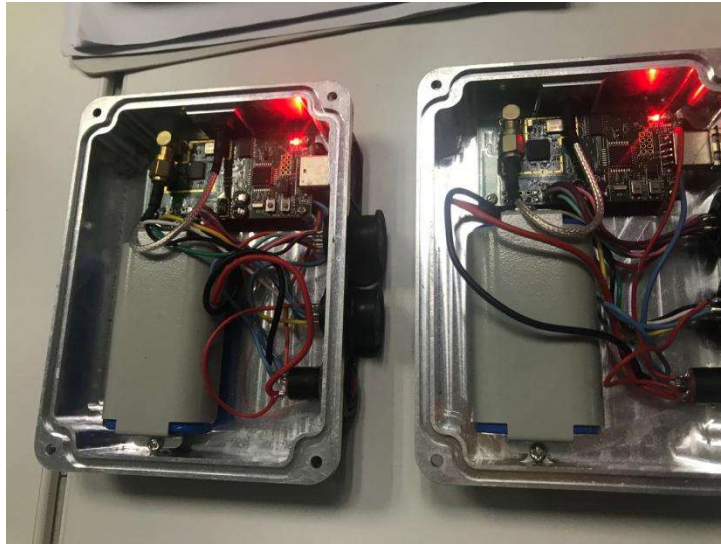


图 2.18 发送数据 1

综合以上几点，验证我们的程序是符合实验要求，完全达到预期的实验目的的。

2.5 心得体会与总结

这次实验还是比较复杂的，在写基站程序的时候遇到了很多坑，要完成转发的逻辑，使用好各个组件，还是要对 TinyOS 的编程模式有深刻的理解才行，这次实验也让我进一步掌握了基于组件编程的模式，熟练了 TinyOS 的无线通信组件和串口通信组件的使用，提升了自己的能力。

实验3 静态路由转发实验

3.1 实验目的

本实验的目的是实现节点和节点间的无线通讯和路由转发,可以使用串口发送消息到基站(指定编号为1的节点为基站),然后转发到指定节点,或者从某个节点定时发送数据,经过固定路径可以转发到基站节点,通过简单的静态路由实验了解无线传感网络的数据路由过程和传感器数据采集过程。

3.2 实验内容

根据要求修改实验二的程序 RadioAndSerial, 添加静态路由转发功能, 具体要求:

(1) 节点之间通过单播的方式收发数据。

(2) 节点路由方式为: 目标节点编号大于自身节点编号的, 将数据转发到编号为自身节点编号+1 的节点, 目标节点编号小于自身节点编号的, 转发到编号自身节点编号-1 的节点, 节点只能直接与相邻编号的节点通讯, 即数据只能在相邻节点之间转发。

(3) 中继节点的 LED 显示目标节点的编号, 停留时间为 1s, 目标节点 LED 显示发送过来, 停留时间为 3s, 数据包只转发一次, 不进行重复发送, 数据到达目标节点后计数值改为原来的计数值加上该节点的节点编号, 沿原路径返回到基站。

(4) 指定编号为 1 的节点为基站, 基站首先显示目标节点编号, 最后显示节点 3 的 counter 值加上发送过去的值基站进行串口读写为广播方式, 如果基站节点收到数据, 目标节点为自身时将数据发送到串口, 否则发送到无线模块。

3.3 实验过程

3.3.1 开发环境

本次实验中使用的环境配置如下:

- (1) 操作系统版本: Arch Linux x86_64
- (2) TinyOS 版本: TinyOS 3.0.0
- (3) 编译器及其版本: GCC version 8.2.1
- (4) 编程环境: Visual Studio Code

3.3.2 实验步骤

①实验三网络结构设计

本次路由转发实验的框架如下图:

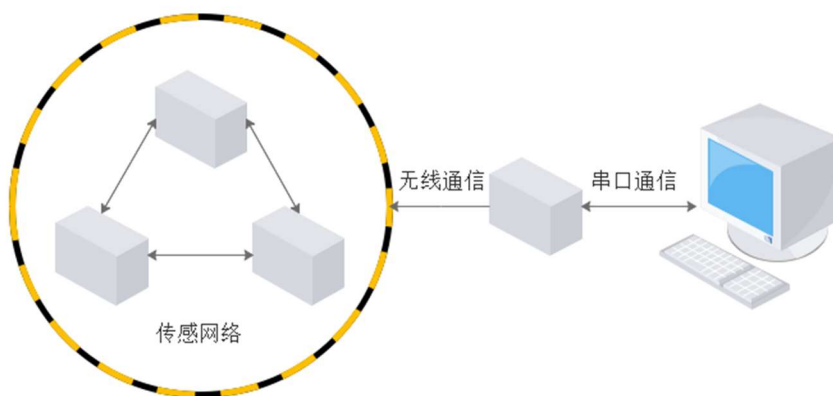


图 3.1 路由转发实验框架设计

将四个节点分别设置编号 1~4，其中节点 1 为基站节点，与 PC 相连，节点 2~4 为普通节点，只进行网络转发。

②BlinkToRadio 程序设计

本实验基于实验二的工作修改，对于 BlinkToRadio 程序，其核心是处理好转发的逻辑实现。

分析知，我们需要实现如下的程序逻辑：

1. 如果收到的消息中，目的节点就是自己

此时把 LED 灯设置为计数器 **counter** 的值，并且发送回基站节点 1。

此过程指示灯持续 3 秒。

2. 如果收到的消息中，目的节点大于自己

此时把 LED 灯设置为目的节点的值，并且转发到自己节点号+1 的节点。

此过程指示灯持续 1 秒。

3. 如果收到的消息中，目的节点小于自己

此时把 LED 灯设置为目的节点的值，并且转发到自己节点号-1 的节点。

此过程指示灯持续 1 秒。

我们只需要在 **receive** 函数里实现即可，如下图：

```

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
        BlinkToRadioMsg* new_pkt = (BlinkToRadioMsg*)(call Packet.getPayload(&pkt, sizeof(BlinkToRadioMsg)));
        if (btrpkt->nodeid == TOS_NODE_ID) {
            setLeds(btrpkt->counter);
            new_pkt->nodeid = 1;
            new_pkt->counter = btrpkt->counter;
            aim_node = TOS_NODE_ID - 1;
            call Timer0.startOneShot(3000);
        }
        else if (btrpkt->nodeid > TOS_NODE_ID) {
            setLeds(btrpkt->nodeid);
            new_pkt->nodeid = btrpkt->nodeid;
            new_pkt->counter = btrpkt->counter;
            aim_node = TOS_NODE_ID + 1;
            call Timer0.startOneShot(1000);
        }
        else {
            setLeds(btrpkt->nodeid);
            new_pkt->nodeid = btrpkt->nodeid;
            new_pkt->counter = btrpkt->counter;
            aim_node = TOS_NODE_ID - 1;
            call Timer0.startOneShot(1000);
        }
    }
    return msg;
}
    
```

图 3.2 BlinkToRadio receive 函数设计

其中，延时使用了定时器，在定时器触发时，调用 send 函数发送消息。

②RadioAndSerial 程序设计

该基站程序在实验二实现的基础上修改，也是增加路由转发逻辑，类似于 BlinkToRadio 的逻辑，在收到 Radio 的消息后，进行同样的处理，这里就不再说明了。

③BlinkToRadio.java 设计

本程序是运行在 PC 上的客户端，可以向串口读写数据。

和实验二中的相比，区别在于这里要先进行数据的发送而不是接受，发送的逻辑如下图所示：

```

int nodeid = 0;
int counter = 0;
BlinkToRadioMsg payload = new BlinkToRadioMsg();
try {
    while (true) {
        System.out.println("Input the nodeid and counter(like 1 2): ");
        Scanner sc = new Scanner(System.in);
        if (sc.hasNextInt()) {
            nodeid = sc.nextInt();
            if (nodeid == 0)
                break;
            counter = sc.nextInt();
        }
        System.out.println("Sending to " + nodeid + " number is: " + counter);
        payload.set_nodeid(nodeid);
        payload.set_counter(counter);
        moteIF.send(1, payload);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException exception) {}
    }
}
    
```

图 3.3 BlinkToRadio.java send 函数设计

完成以上步骤之后，可以进行编译运行程序。

3.4 实验结果与分析

3.4.1 程序测试

完成代码编写后，分别使用如下四条命令，将 RadioAndSerial 烧录到节点 1，将 BlinkToRadio 烧录到节点 2~4。

```
make telosb install,1 /dev/ttyUSB0
```

```
make telosb install,2 /dev/ttyUSB0
```

```
make telosb install,3 /dev/ttyUSB0
```

```
make telosb install,4 /dev/ttyUSB0
```

其中 install 后面的数字为指定节点编号。

然后将烧录有 RadioAndSerial 的节点 1 使用 USB 连接到 PC，然后打开节点 2~4 的电源，在 PC 上执行以下命令，运行 BlinkToRadio.java 程序。

```
java BlinkToRadio -comm serial@/dev/ttyUSB0:telosb
```

然后依次执行想结点 2, 3, 4 发送数据 5, 6, 7，观察实验结果，其中程序输入如下所示，确认收到了返回的消息。

```
Trying to locate the file 'linux amd64 toscmm.lib' in the classpath
Temporary file created: '/tmp/toscomm191414997078608642.lib'
Library copied successfully. Let's load it.
Library loaded successfully
serial@/dev/ttyUSB0:115200: resynchronising
Input the nodeid and counter(like 1 2):
2 5
Sending to 2 number is: 5
Input the nodeid and counter(like 1 2):

Received a packet, nodeid: 1 counter: 5
3 6
Sending to 3 number is: 6
Input the nodeid and counter(like 1 2):

Received a packet, nodeid: 1 counter: 6
4 7
Sending to 4 number is: 7
Input the nodeid and counter(like 1 2):

Received a packet, nodeid: 1 counter: 7
```

图 3.4 BlinkToRadio.java 输出

3.4.2 结果分析

下面针对节点运行中一步步的变化，对三次消息的发送进行分析。

实验开始时，如图 3.5，四个节点的 LED 灯均不亮。

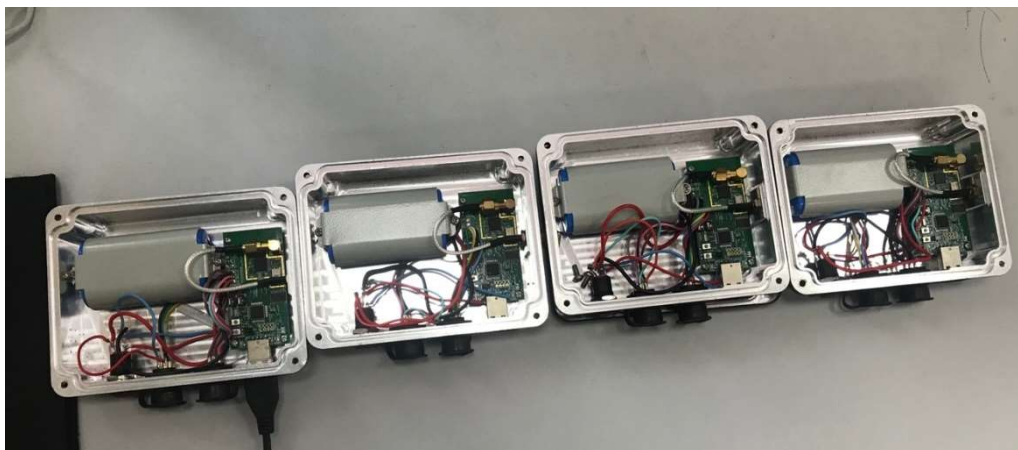


图 3.5 执行前：四个节点的 LED 灯均不亮

① 向节点 2 发送数据 5

首先，节点 1 收到来自 PC 的串口消息，设置 LED 灯为目的节点编号 2，等待 1 秒。

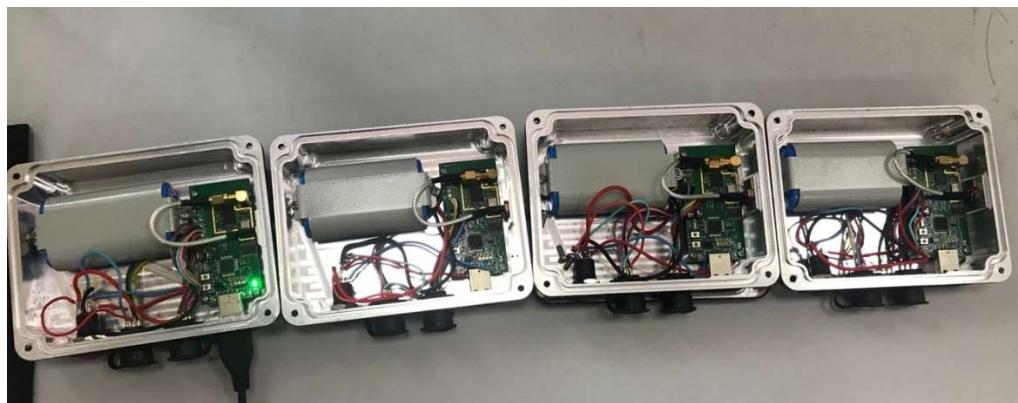


图 3.6 发给节点 2 数据 5：第一步

接着数据被转发给节点 2，设置 LED 灯为计数器的值 5，等待 3 秒。

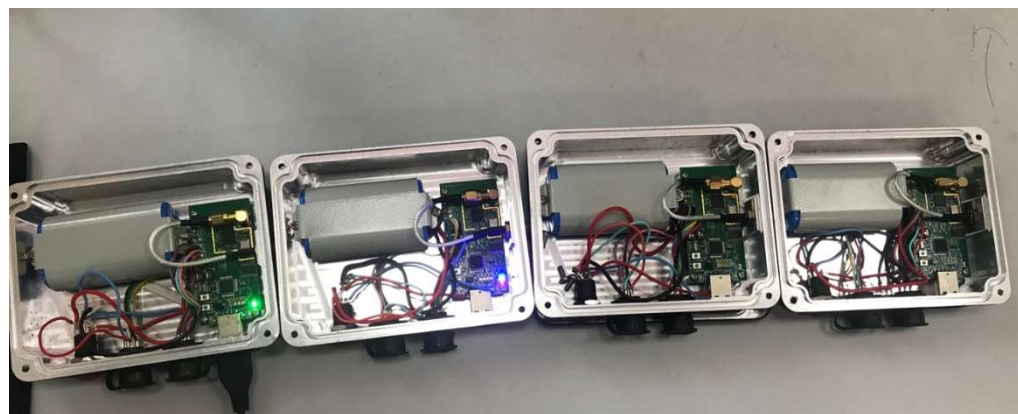


图 3.7 发给节点 2 数据 5：第二步

接着节点 2 发送相同的数据，返回给节点 1，节点 1 收到后，设置 LED 灯为目的节点编号 1，等待 1 秒。

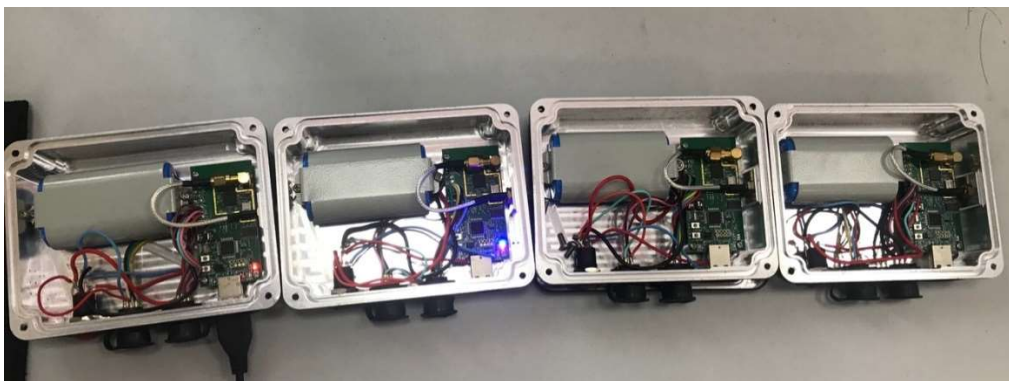


图 3.8 发给节点 2 数据 5：第三步

节点 1 再将数据发回 PC，终端显示出的消息如图 3.4。

② 向节点 3 发送数据 6

接着上一步继续测试向节点 3 发送消息。

首先，节点 1 收到来自 PC 的串口消息，设置 LED 灯为目的节点编号 3，等待 1 秒。

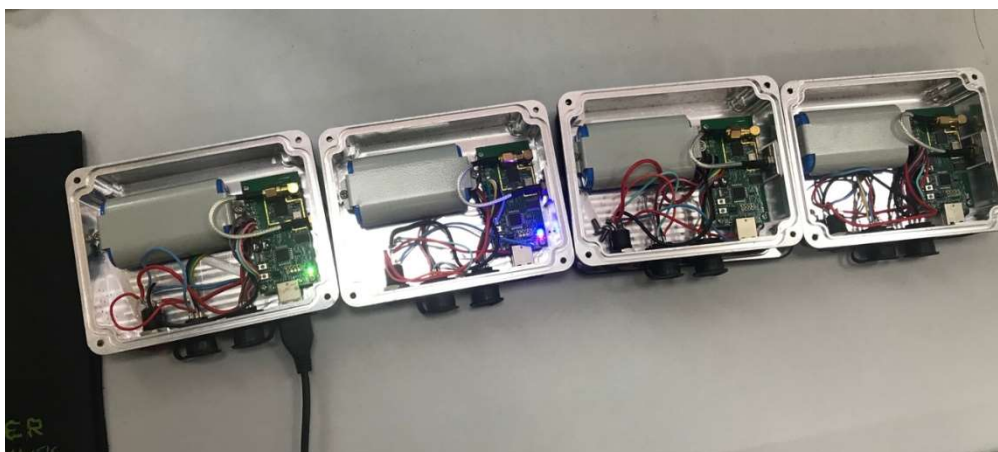


图 3.9 发给节点 3 数据 6：第一步

接着数据被转发到节点 2，设置 LED 灯为目的节点编号 3，等待 1 秒。



图 3.10 发给节点 3 数据 6：第二步

接着数据被转发给节点 3，设置 LED 灯为计数器的值 6，等待 3 秒。



图 3.11 发给节点 3 数据 6：第三步

接着节点 3 发送相同的数据，返回给节点 1，首先向节点 2 转发，节点 2 收到后，设置 LED 灯为目的节点编号 1，等待 1 秒。



图 3.12 发给节点 3 数据 6：第四步

接着节点 2 转发数据给节点 1，节点 1 收到后，设置 LED 灯为目的节点编号 1，等待 1 秒。

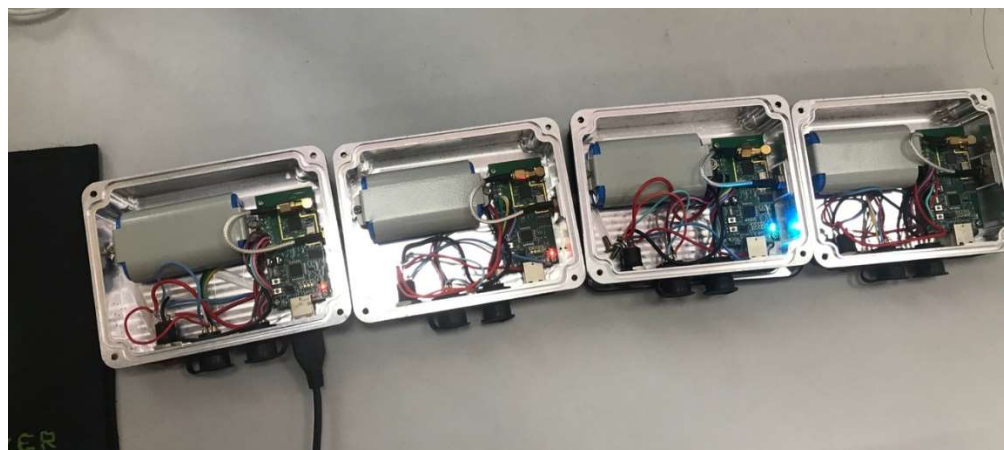


图 3.13 发给节点 3 数据 6：第五步

节点 1 再将数据发回 PC，终端显示出的消息如图 3.4。

② 向节点 4 发送数据 7

接着上一步继续测试向节点 4 发送消息。

首先，节点 1 收到来自 PC 的串口消息，设置 LED 灯为目的节点编号 4，等待 1 秒。

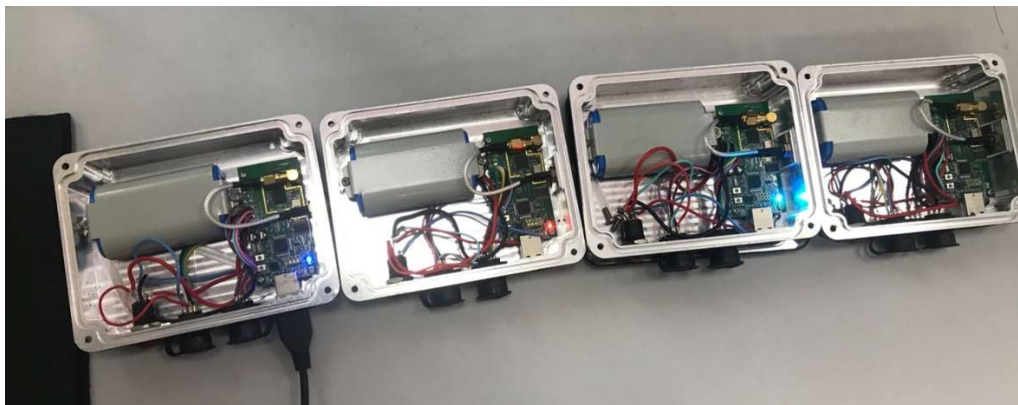


图 3.14 发给节点 4 数据 7：第一步

接着数据被转发到节点 2，设置 LED 灯为目的节点编号 4，等待 1 秒。

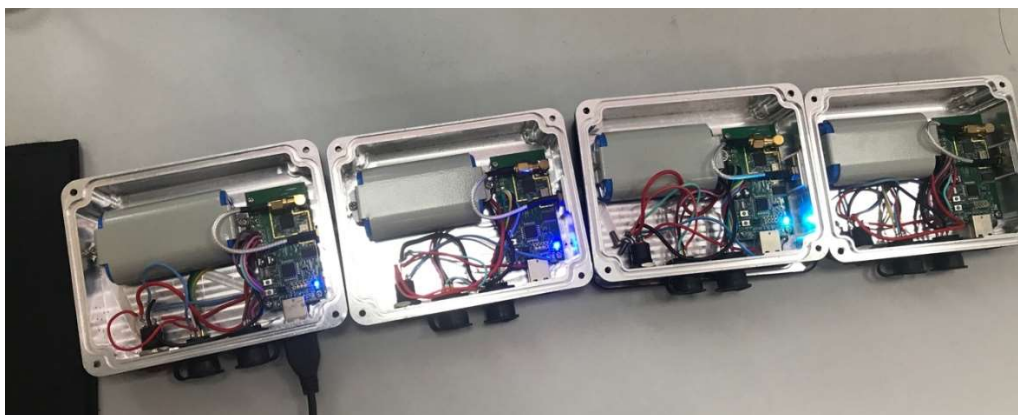


图 3.15 发给节点 4 数据 7：第二步

接着数据被转发到节点 3，设置 LED 灯为目的节点编号 4，等待 1 秒。



图 3.16 发给节点 4 数据 7：第三步

接着数据被转发给节点 4，设置 LED 灯为计数器的值 7，等待 3 秒。

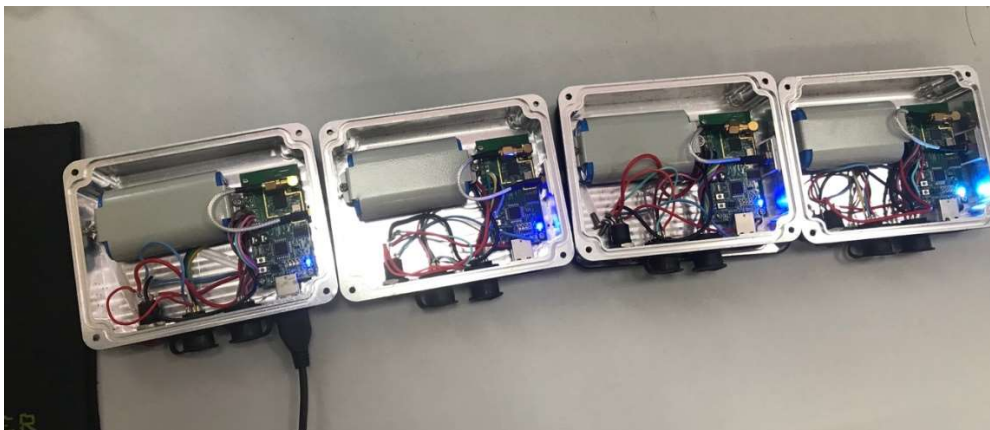


图 3.17 发给节点 4 数据 7：第四步

接着节点 4 发送相同的数据，返回给节点 1，首先向节点 3 转发，节点 3 收到后，设置 LED 灯为目的节点编号 1，等待 1 秒。



图 3.18 发给节点 4 数据 7：第五步

接着节点 3 转发数据给节点 2，节点 2 收到后，设置 LED 灯为目的节点编号 1，等待 1 秒。



图 3.19 发给节点 4 数据 7：第六步

接着节点 2 转发数据给节点 1，节点 1 收到后，设置 LED 灯为目的节点编号 1，等待 1 秒。

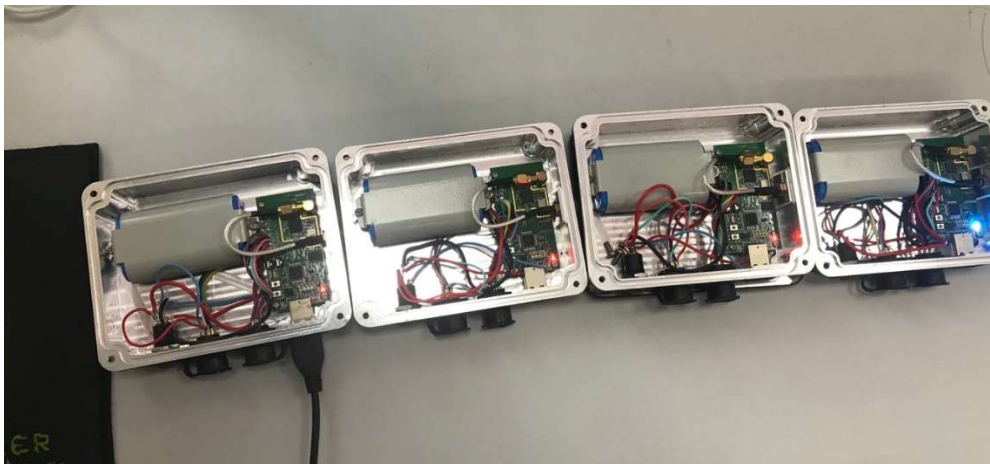


图 3.20 发给节点 4 数据 7：第七步

节点 1 再将数据发回 PC，终端显示出收到的消息如图 3.4。

综上，整个实验的流程以及返回的结果都没有问题，验证我们的程序是符合实验要求，完全达到预期的实验目的的。

3.5 心得体会与总结

通过这次实验，我更加熟悉了 nesC 语言的编程，同时也理解了无限通信中路由转发的原理，对课程中讲到的相关原理有了更深的理解，也提高了自己的能力。ZigBee 无线通信在物联网领域有着十分广泛的应用，这几次实验的经验也是一次很好的经历，为我以后的学习实践打下了基础。

附录 实验源代码

附录.1 实验目录结构

本实验提交的文件目录结构如下图

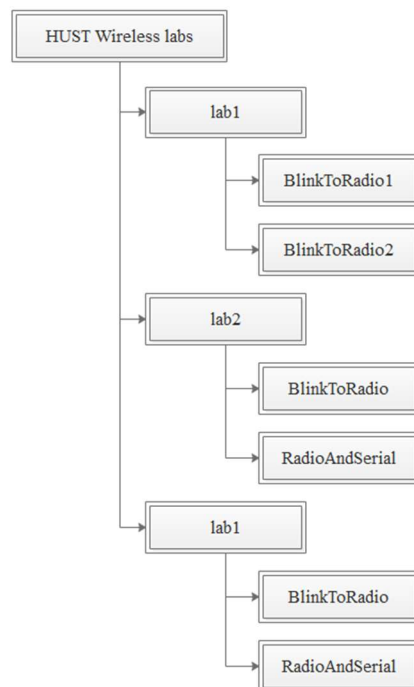


图 附录.1 文件目录结构图

附录.2 实验一程序源代码

代码段 附录.1 BlinkToRadio1/BlinkToRadio.h

```

/*
 * HUST-IOT-Network-Labs Wireless::lab1::BlinkToRadio1
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#ifndef BLINKTORADIO_H
#define BLINKTORADIO_H

enum {
    AM_BLINKTORADIO = 6,
    TIMER_PERIOD_MILLI = 1000,
    NOID_ID_1 = 1,
    NOID_ID_2 = 2
}
    
```

```
};

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

#endif
```

代码段 附录.2 BlinkToRadio1/BlinkToRadioAppC.nc

```
/*
 * HUST-IOT-Network-Labs Wireless::lab1::BlinkToRadio1
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#include <Timer.h>
#include "BlinkToRadio.h"

configuration BlinkToRadioAppC {
}

implementation {
    components MainC;
    components LedsC;
    components BlinkToRadioC as App;
    components new TimerMilliC() as Timer0;
    components ActiveMessageC;
    components new AMSenderC(AM_BLINKTORADIO);
    components new AMReceiverC(AM_BLINKTORADIO);

    App.Boot -> MainC;
    App.Leds -> LedsC;
    App.Timer0 -> Timer0;
    App.Packet -> AMSenderC;
    App.AMPacket -> AMSenderC;
    App.AMControl -> ActiveMessageC;
    App.AMSend -> AMSenderC;
    App.Receive -> AMReceiverC;
}
```

代码段 附录.3 BlinkToRadio1/BlinkToRadioC.nc

```

/*
 * HUST-IOT-Network-Labs Wireless::lab1::BlinkToRadio1
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#include <Timer.h>
#include "BlinkToRadio.h"

module BlinkToRadioC {
    uses interface Boot;
    uses interface Leds;
    uses interface Timer<TMilli> as Timer0;
    uses interface Packet;
    uses interface AMPacket;
    uses interface AMSend;
    uses interface Receive;
    uses interface SplitControl as AMControl;
}

implementation {

    uint16_t counter;
    message_t pkt;
    bool busy = FALSE;

    void setLeds(uint16_t val) {
        if (val & 0x01)
            call Leds.led0On();
        else
            call Leds.led0Off();
        if (val & 0x02)
            call Leds.led1On();
        else
            call Leds.led1Off();
        if (val & 0x04)
            call Leds.led2On();
        else
            call Leds.led2Off();
    }
}

```

```

}

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

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

event void AMControl.stopDone(error_t err) {
}

event void Timer0.fired() {
    counter++;
    if (!busy) {
        BlinkToRadioMsg* btrpkt =
            (BlinkToRadioMsg*)(call Packet.getPayload(&pkt,
sizeof(BlinkToRadioMsg)));
        if (btrpkt == NULL) {
            return;
        }
        btrpkt->nodeid = NOID_ID_1;
        btrpkt->counter = counter;
        if (call AMSend.send(AM_BROADCAST_ADDR,
            &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
            busy = TRUE;
        }
    }
}

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

```

```

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
len){
    if (len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
        if (btrpkt->nodeid == NOID_ID_2)
            setLeds(btrpkt->counter);
    }
    return msg;
}

```

代码段 附录.4 BlinkToRadio1/Makefile

```

COMPONENT=BlinkToRadioAppC
PFLAGS+=-DCC2420_DEF_CHANNEL=15
TINYOS_ROOT_DIR?=/home/panyue/tinyos-main
include $(TINYOS_ROOT_DIR)/Makefile.include

```

代码段 附录.4 BlinkToRadio2/BlinkToRadioC.nc

```

/*
 * HUST-IOT-Network-Labs Wireless::lab1::BlinkToRadio2
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#include <Timer.h>
#include "BlinkToRadio.h"

module BlinkToRadioC {
    uses interface Boot;
    uses interface Leds;
    uses interface Timer<TMilli> as Timer0;
    uses interface Packet;
    uses interface AMPacket;
    uses interface AMSend;
    uses interface Receive;
    uses interface SplitControl as AMControl;
}

implementation {

```



```

uint16_t counter;
message_t pkt;
bool busy = FALSE;

void setLeds(uint16_t val) {
    if (val & 0x01)
        call Leds.led0On();
    else
        call Leds.led0Off();
    if (val & 0x02)
        call Leds.led1On();
    else
        call Leds.led1Off();
    if (val & 0x04)
        call Leds.led2On();
    else
        call Leds.led2Off();
}

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

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

event void AMControl.stopDone(error_t err) {
}

event void Timer0.fired() {
    counter++;
    if (!busy) {
        BlinkToRadioMsg* btrpkt =
            (BlinkToRadioMsg*)(call Packet.getPayload(&pkt,
sizeof(BlinkToRadioMsg)));
        if (btrpkt == NULL) {

```

```

        return;
    }
    btrpkt->nodeid = NOID_ID_2;
    btrpkt->counter = counter + 1;
    if (call AMSend.send(AM_BROADCAST_ADDR,
        &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
        busy = TRUE;
    }
}

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

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
len){
    if (len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
        if (btrpkt->nodeid == NOID_ID_1)
            setLeds(btrpkt->counter);
    }
    return msg;
}
}

```

文件 BlinkToRadio2/BlinkToRadio.h, BlinkToRadio2/BlinkToRadioAppC.nc 和 BlinkToRadio/Makefile 与文件夹 BlinkToRadio1 内的同名文件完全一致, 即如代码段附录.1, 代码段附录.2, 代码段附录.4 所示, 因此不再重复收录。

附录.2 实验二程序源代码

代码段 附录.5 BlinkToRadio/BlinkToRadio.h

```
/*
 * HUST-IOT-Network-Labs Wireless::lab2::BlinkToRadio
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#ifndef BLINKTORADIO_H
#define BLINKTORADIO_H

enum {
    AM_BLINKTORADIOMSG = 6,
    TIMER_PERIOD_MILLI = 1000
};

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

#endif
```

代码段 附录.6 BlinkToRadio/BlinkToRadioAppC.nc

```
/*
 * HUST-IOT-Network-Labs Wireless::lab2::BlinkToRadio
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#include <Timer.h>
#include "BlinkToRadio.h"

configuration BlinkToRadioAppC {
}

implementation {
    components MainC;
```

```

components LedsC;
components BlinkToRadioC as App;
components new TimerMilliC() as Timer0;
components ActiveMessageC;
components new AMSenderC(AM_BLINKTORADIOMSG);
components new AMReceiverC(AM_BLINKTORADIOMSG);

App.Boot -> MainC;
App.Leds -> LedsC;
App.Timer0 -> Timer0;
App.Packet -> AMSenderC;
App.AMPacket -> AMSenderC;
App.AMControl -> ActiveMessageC;
App.AMSend -> AMSenderC;
App.Receive -> AMReceiverC;
}

```

代码段 附录.7 BlinkToRadio/BlinkToRadioC.nc

```

/*
 * HUST-IOT-Network-Labs Wireless::lab2::BlinkToRadio
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#include <Timer.h>
#include "BlinkToRadio.h"

module BlinkToRadioC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}

implementation {

  uint16_t counter;

```

```

message_t pkt;
bool busy = FALSE;

void setLeds(uint16_t val) {
    if (val & 0x01)
        call Leds.led0On();
    else
        call Leds.led0Off();
    if (val & 0x02)
        call Leds.led1On();
    else
        call Leds.led1Off();
    if (val & 0x04)
        call Leds.led2On();
    else
        call Leds.led2Off();
}

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

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

event void AMControl.stopDone(error_t err) {
}

event void Timer0.fired() {
    counter++;
    if (!busy) {
        BlinkToRadioMsg* btrpkt =
            (BlinkToRadioMsg*)(call Packet.getPayload(&pkt,
sizeof(BlinkToRadioMsg)));
        if (btrpkt == NULL) {
            return;

```

```

    }
    btrpkt->nodeid = TOS_NODE_ID;
    btrpkt->counter = counter;
    if (call AMSend.send(AM_BROADCAST_ADDR,
        &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
        busy = TRUE;
    }
}

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

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
len){
    if (len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
        setLeds(btrpkt->counter);
        counter = btrpkt->counter;
    }
    return msg;
}
}

```

代码段 附录.8 BlinkToRadio/BlinkToRadio.java

```

/*
 * HUST-IOT-Network-Labs Wireless::lab2::BlinkToRadio
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

import java.io.IOException;
import java.util.Scanner;

import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;

```

```

public class BlinkToRadio implements MessageListener {

    private MoteIF moteIF;

    public BlinkToRadio(MoteIF moteIF) {
        this.moteIF = moteIF;
        this.moteIF.registerListener(new BlinkToRadioMsg(), this);
    }

    public void sendPackets(int nodeid, int counter) {
        BlinkToRadioMsg payload = new BlinkToRadioMsg();
        try {
            payload.set_nodeid(nodeid);
            payload.set_counter(counter);
            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) {
        BlinkToRadioMsg msg = (BlinkToRadioMsg)message;
        System.out.println("\nReceived packet to: " + to + " nodeid: " +
msg.get_nodeid() + " counter: " + msg.get_counter());
        System.out.println("Input the nodeid and counter(like 1 2): ");
        Scanner sc = new Scanner(System.in);
        int nodeid = sc.nextInt();
        int counter = sc.nextInt();
        System.out.println("Sending to " + nodeid + " number is: " + counter);
        sendPackets(nodeid, counter);
    }

    private static void usage() {
        System.err.println("usage: BlinkToRadio [-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);
BlinkToRadio radio = new BlinkToRadio(mif);
}
}

```

代码段 附录.9 BlinkToRadio/Makefile

```

COMPONENT=BlinkToRadioAppC
PFLAGS+=-DCC2420_DEF_CHANNEL=26
TOSMAKE_PRE_EXE_DEPS += BlinkToRadio.class
TOSMAKE_CLEAN_EXTRA = *.class BlinkToRadioMsg.java

BlinkToRadio.class: $(wildcard *.java) BlinkToRadioMsg.java
    javac -target 1.6 -source 1.6 *.java

BlinkToRadioMsg.java:
    nescc-mig java $(CFLAGS) -java-classname=BlinkToRadioMsg
BlinkToRadio.h BlinkToRadioMsg -o $@

TINYOS_ROOT_DIR?=/home/panyue/tinyos-main
include $(TINYOS_ROOT_DIR)/Makefile.include

```

代码段 附录.10 RadioAndSerial/RadioAndSerial.h

```

/*

```



```

* HUST-IOT-Network-Labs Wireless::lab2::ReadAndSerial
*
* Created by zxcryp
*
* Github: zxc479773533
*/

#ifndef RADIOANDSERIAL_H
#define RADIOANDSERIAL_H

enum {
    AM_BLINKTORADIOMSG = 6,
};

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

#endif

```

代码段 附录.11 RadioAndSerial/RadioAndSerialAppC.nc

```

/*
* HUST-IOT-Network-Labs Wireless::lab2::ReadAndSerial
*
* Created by zxcryp
*
* Github: zxc479773533
*/

#include "RadioAndSerial.h"

configuration RadioAndSerialAppC {
}

implementation {
    components MainC;
    components LedsC;
    components RadioAndSerial as App;
    components ActiveMessageC as Radio;
    components SerialActiveMessageC as Serial;
    components new AMSenderC(AM_BLINKTORADIOMSG);
    components new AMReceiverC(AM_BLINKTORADIOMSG);
}

```

```

components new SerialAMSenderC(AM_BLINKTORADIOMSG);
components new SerialAMReceiverC(AM_BLINKTORADIOMSG);

App.Boot -> MainC;
App.Leds -> LedsC;

App.RadioAMControl -> Radio;
App.SerialControl -> Serial;

App.RadioPacket -> Radio;
App.RadioAMSend -> AMSenderC;
App.RadioReceive -> AMReceiverC;

App.SerialPacket -> Serial;
App.SerialAMSend -> SerialAMSenderC;
App.SerialReceive -> SerialAMReceiverC;
}

```

代码段 附录.12 RadioAndSerial/RadioAndSerialC.nc

```

/*
 * HUST-IOT-Network-Labs Wireless::lab2::ReadAndSerial
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */
#include "RadioAndSerial.h"

module RadioAndSerial {
  uses interface Boot;
  uses interface Leds;
  // Radio
  uses interface Packet as RadioPacket;
  uses interface AMSend as RadioAMSend;
  uses interface Receive as RadioReceive;
  uses interface SplitControl as RadioAMControl;
  // Serial
  uses interface Packet as SerialPacket;
  uses interface AMSend as SerialAMSend;
  uses interface Receive as SerialReceive;
  uses interface SplitControl as SerialControl;
}

```

```

implementation {

    // Radio packet and Serial packet
    message_t packet;
    // Busy mark for Radio and Serial
    bool busy_r;
    bool busy_s;

    void getRadioPkt() {
        call Leds.led2Toggle();
    }

    void getSerialPkt() {
        call Leds.led0Toggle();
    }
    // Start control
    event void Boot.booted() {
        call RadioAMControl.start();
        call SerialControl.start();
    }

    event void RadioAMControl.startDone(error_t err) {
        if (err == SUCCESS) {
            busy_r = FALSE;
        }
        else {
            call RadioAMControl.start();
        }
    }

    event void SerialControl.startDone(error_t err) {
        if (err == SUCCESS) {
            busy_s = FALSE;
        }
        else {
            call SerialControl.start();
        }
    }

    event void RadioAMControl.stopDone(error_t err) {
    }
}

```

```

event void SerialControl.stopDone(error_t err) {
}

event message_t* RadioReceive.receive(message_t* msg, void* payload,
uint8_t len) {
    if (!busy_r && len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* ppkt = (BlinkToRadioMsg*)payload;
        BlinkToRadioMsg* new_pkt = (BlinkToRadioMsg*)(call
SerialPacket.getPayload(&packet, sizeof(BlinkToRadioMsg)));
        if (new_pkt == NULL) {
            return msg;
        }
        getRadioPkt();
        new_pkt->nodeid = ppkt->nodeid;
        new_pkt->counter = ppkt->counter;
        if (call SerialAMSend.send(new_pkt->nodeid, &packet,
sizeof(BlinkToRadioMsg)) == SUCCESS) {
            busy_s = TRUE;
        }
    }
    return msg;
}

event message_t* SerialReceive.receive(message_t* msg, void* payload,
uint8_t len) {
    if (!busy_s && len == sizeof(BlinkToRadioMsg)) {
        BlinkToRadioMsg* ppkt = (BlinkToRadioMsg*)payload;
        BlinkToRadioMsg* new_pkt = (BlinkToRadioMsg*)(call
RadioPacket.getPayload(&packet, sizeof(BlinkToRadioMsg)));
        if (new_pkt == NULL) {
            return msg;
        }
        getSerialPkt();
        new_pkt->nodeid = ppkt->nodeid;
        new_pkt->counter = ppkt->counter;
        if (call RadioAMSend.send(new_pkt->nodeid, &packet,
sizeof(BlinkToRadioMsg)) == SUCCESS) {
            busy_r = TRUE;
        }
    }
    return msg;
}

```

```
event void RadioAMSend.sendDone(message_t* msg, error_t err) {  
    if (&packet == msg) {  
        busy_r = FALSE;  
    }  
}  
  
event void SerialAMSend.sendDone(message_t* msg, error_t err) {  
    if (&packet == msg) {  
        busy_s = FALSE;  
    }  
}  
}
```

代码段 附录.13 RadioAndSerial/Makefile

```
COMPONENT=RadioAndSerialAppC  
PFLAGS+=-DCC2420_DEF_CHANNEL=26  
TINYOS_ROOT_DIR?=/home/panyue/tinyos-main  
include $(TINYOS_ROOT_DIR)/Makefile.include
```

附录.3 实验三程序源码

代码段 附录.14 BlinkToRadio/BlinkToRadio.h

```
/*
 * HUST-IOT-Network-Labs Wireless::lab3::BlinkToRadio
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#ifndef BLINKTORADIO_H
#define BLINKTORADIO_H

enum {
    AM_BLINKTORADIOMSG = 6,
    TIMER_PERIOD_MILLI = 1000
};

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

#endif
```

代码段 附录.15 BlinkToRadio/BlinkToRadioAppC.nc

```
/*
 * HUST-IOT-Network-Labs Wireless::lab3::BlinkToRadio
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#include <Timer.h>
#include "BlinkToRadio.h"

configuration BlinkToRadioAppC {
}

implementation {
    components MainC;
```

```

components LedsC;
components BlinkToRadioC as App;
components new TimerMilliC() as Timer0;
components ActiveMessageC;
components new AMSenderC(AM_BLINKTORADIOMSG);
components new AMReceiverC(AM_BLINKTORADIOMSG);

App.Boot -> MainC;
App.Leds -> LedsC;
App.Timer0 -> Timer0;
App.Packet -> AMSenderC;
App.AMPacket -> AMSenderC;
App.AMControl -> ActiveMessageC;
App.AMSend -> AMSenderC;
App.Receive -> AMReceiverC;
}

```

代码段 附录.16 BlinkToRadio/BlinkToRadioC.nc

```

/*
 * HUST-IOT-Network-Labs Wireless::lab3::BlinkToRadio
 *
 * Created by zxcryp
 *
 * Github: zxc479773533
 */

#include <Timer.h>
#include "BlinkToRadio.h"

module BlinkToRadioC {
    uses interface Boot;
    uses interface Leds;
    uses interface Timer<TMilli> as Timer0;
    uses interface Packet;
    uses interface AMPacket;
    uses interface AMSend;
    uses interface Receive;
    uses interface SplitControl as AMControl;
}

implementation {

    uint16_t aim_node;

```

```

uint16_t counter;
message_t pkt;
bool busy;

void setLeds(uint16_t val) {
    if (val & 0x01)
        call Leds.led0On();
    else
        call Leds.led0Off();
    if (val & 0x02)
        call Leds.led1On();
    else
        call Leds.led1Off();
    if (val & 0x04)
        call Leds.led2On();
    else
        call Leds.led2Off();
}

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

event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
        busy = FALSE;
    }
    else {
        call AMControl.start();
    }
}

event void AMControl.stopDone(error_t err) {
}

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

event void Timer0.fired() {

```



```

        if (call AMSend.send(aim_node, &pkt, sizeof(BlinkToRadioMsg)) ==
SUCCESS) {
            busy = TRUE;
        }
    }

    event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
len){
        if (len == sizeof(BlinkToRadioMsg)) {
            BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
            BlinkToRadioMsg* new_pkt = (BlinkToRadioMsg*)(call
Packet.getPayload(&pkt, sizeof(BlinkToRadioMsg)));
            if (btrpkt->nodeid == TOS_NODE_ID) {
                setLeds(btrpkt->counter);
                new_pkt->nodeid = 1;
                new_pkt->counter = btrpkt->counter;
                aim_node = TOS_NODE_ID - 1;
                call Timer0.startOneShot(3000);
            }
            else if (btrpkt->nodeid > TOS_NODE_ID) {
                setLeds(btrpkt->nodeid);
                new_pkt->nodeid = btrpkt->nodeid;
                new_pkt->counter = btrpkt->counter;
                aim_node = TOS_NODE_ID + 1;
                call Timer0.startOneShot(1000);
            }
            else {
                setLeds(btrpkt->nodeid);
                new_pkt->nodeid = btrpkt->nodeid;
                new_pkt->counter = btrpkt->counter;
                aim_node = TOS_NODE_ID - 1;
                call Timer0.startOneShot(1000);
            }
        }
        return msg;
    }
}

```

代码段 附录.17 BlinkToRadio/BlinkToRadio.java

```

/*
 * HUST-IOT-Network-Labs Wireless::lab3::BlinkToRadio
 *

```

```

* Created by zxcryp
*
* Github: zxc479773533
*/

import java.io.IOException;
import java.util.Scanner;

import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;

public class BlinkToRadio implements MessageListener {

    private MoteIF moteIF;

    public BlinkToRadio(MoteIF moteIF) {
        this.moteIF = moteIF;
        this.moteIF.registerListener(new BlinkToRadioMsg(), this);
        int nodeid = 0;
        int counter = 0;
        BlinkToRadioMsg payload = new BlinkToRadioMsg();
        try {
            while (true) {
                System.out.println("Input the nodeid and counter(like 1 2): ");
                Scanner sc = new Scanner(System.in);
                if (sc.hasNextInt()) {
                    nodeid = sc.nextInt();
                    if (nodeid == 0)
                        break;
                    counter = sc.nextInt();
                }
                System.out.println("Sending to " + nodeid + " number is: " + counter);
                payload.set_nodeid(nodeid);
                payload.set_counter(counter);
                moteIF.send(1, payload);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException exception) {}
            }
        } catch (IOException exception) {
            System.err.println("Exception thrown when sending packets. Exiting.");
        }
    }
}

```

```

        System.err.println(exception);
    }
}

public void messageReceived(int to, Message message) {
    BlinkToRadioMsg msg = (BlinkToRadioMsg)message;
    System.out.println("\nReceived a packet, nodeid: " + msg.get_nodeid() + "
counter: " + msg.get_counter());
}

private static void usage() {
    System.err.println("usage: BlinkToRadio [-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);
    }

    MotelIF mif = new MotelIF(phoenix);
    BlinkToRadio radio = new BlinkToRadio(mif);
}
}

```

代码段 附录.18 BlinkToRadio/Makefile

COMPONENT=BlinkToRadioAppC

```
PFLAGS+=-DCC2420_DEF_CHANNEL=26
TOSMAKE_PRE_EXE_DEPS += BlinkToRadio.class
TOSMAKE_CLEAN_EXTRA = *.class BlinkToRadioMsg.java

BlinkToRadio.class: $(wildcard *.java) BlinkToRadioMsg.java
    javac -target 1.6 -source 1.6 *.java

BlinkToRadioMsg.java:
    nescc-mig java $(CFLAGS) -java-classname=BlinkToRadioMsg
BlinkToRadio.h BlinkToRadioMsg -o $@

TINYOS_ROOT_DIR?=/home/panyue/tinyos-main
include $(TINYOS_ROOT_DIR)/Makefile.include
```

参考文献

- 1 [美] William Stallings 著, 何军 等译. 无线通信与网络 (第 2 版). 北京: 清华大学出版社, 2005.
- 2 [美] Shahin hani 著, 沈建华, 王维华, 阔鑫译. ZigBee 无线网络与收发器. 北京: 北京航空航天大学出版社, 2013.
- 3 [美] Philip Levis, David Gay 著, TinyOS Programming. CAMBRIDGE UNIVERSITY PRESS, 2009.
- 4 陈鸣著. 计算机网络: 原理与实践. 北京: 高等教育出版社, 2013.
- 5 TinyOS 官方 wiki, URL: <http://tinyos.stanford.edu/tinyos-wiki/index.php>
- 6 TinyOS 官方开源仓库, URL: <https://github.com/tinyos/tinyos-main>