

北京邮电大学

无线传感器网络



传感器网络数据收发仿真

姓 名 黄链泽 李梓硕 郑毓恒

学 院 计算机学院

专 业 计算机科学与技术

班 级 2020211302

学 号 2020211246

2020211250

2020211262

任课教师 刘亮

2023 年 5 月

实验目的

1. 编写仿真脚本，实现传感器网络（至少三个）数据的发送和接收
2. 仿真过程记录到 log 日志中，仿真结束后对 log 日志进行分析，计算数据的丢包率和时延。

实验环境

- Ubuntu-14.04.6
- VMware Workstation 16 Pro
- TinyOS 2.1.2
- Python 2.7.6
- GCC 4.8.4

实验内容和步骤

实验在安装完成的 TinyOS 程序文件夹中自带的 RadioCountToLedsC 中进行，首先需要修改其中的 RadioCountToLedsC.h、RadioCountToLedsC.nc 和 RadioCountToLedsAppC.nc 三个文件的内容。

在 RadioCountToLedsC.h 文件，除了默认的 counter 成员，给自定义的 radio_count_msg_t 类型添加 src、hour、min 和 sec 四个成员，分别表示发送源节点和发送时间。

```
#ifndef RADIO_COUNT_TO_LEDS_H
#define RADIO_COUNT_TO_LEDS_H

typedef nx_struct radio_count_msg {
    nx_uint16_t counter;
    nx_uint16_t src;
    nx_uint16_t hour;
    nx_uint16_t min;
    nx_uint16_t sec;
} radio_count_msg_t;

enum {
    AM_RADIO_COUNT_MSG = 6,
};

#endif
```

由于在代码中需要得知发送的源节点，需要调用 `AMPacket.source()` 函数。在 `RadioCountToLedsAppC.nc` 中，将 `RadioCountToLeds.nc` 中的 `AMPacket` 接口连接到 `AMSenderC` 部件。

```
#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}
implementation {
  components MainC, RadioCountToLedsC as App, LedsC;
  components new AMSenderC(AM_RADIO_COUNT_MSG);
  components new AMReceiverC(AM_RADIO_COUNT_MSG);
  components new TimerMilliC();
  components ActiveMessageC;

  App.Boot -> MainC.Boot;

  App.Receive -> AMReceiverC;
  App.AMSend -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.Leds -> LedsC;
  App.MilliTimer -> TimerMilliC;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
}
```

在 `RadioCountToLedsC.nc` 文件中，为 module 新增接口 `AMPacket`。

```
module RadioCountToLedsC @safe() {
  uses {
    interface Leds;
    interface Boot;
    interface Receive;
    interface AMSend;
    interface AMPacket;
    interface Timer<TMilli> as MilliTimer;
    interface SplitControl as AMControl;
    interface Packet;
  }
}
```

新增两个函数，`writelog()`负责将数据的收发信息写入日志文件，`resetLog()`在重启脚本时重置日志文件。日志中的一行表示为一次发送或者接收，每行开头会以‘Send’或者‘Recv’以作区分。然后，后续的数字按顺序分别表示源节点号、发送时间（时:分:秒）。假如是Recv，接收信息类型，后续还有目的节点号和接收时间（时:分:秒）。

```
char logbuf[1024];

void writeLog(int isrecved, nx_uint16_t send_node, nx_uint16_t shour, nx_uint16_t
smin, nx_uint16_t ssec, nx_uint16_t recv_node, nx_uint16_t rhour, nx_uint16_t
rmin, nx_uint16_t rsec)
{
    FILE *fd = fopen("log.txt", "a+");
    if(fd==0)
    {
        printf("Opening Log File Error!\n");
        exit(0);
    }
    memset(logbuf, 0, sizeof(logbuf));

    if(!isrecved)
    {
        sprintf(logbuf, "Send %d %d:%02d:%02d\n", send_node, shour, smin,
ssec);
    }
    else
    {
        sprintf(logbuf, "Recv %d %d:%02d:%02d %d %d:%02d:%02d\n",
send_node, shour, smin, ssec, recv_node, rhour, rmin, rsec);
    }
    fprintf(fd, logbuf);

    fclose(fd);
}

void resetLog()
{
    FILE *fd = fopen("log.txt", "w+");
    if(fd==0)
    {
        printf("Opening Log File Error!\n");
        exit(0);
    }
    fclose(fd);
}
```

在事件函数 Boot.booted()中，也就是节点启动后调用的函数中，新增一条 debug 信息输出到屏幕上，提示节点已启动。并且调用 resetLog()函数，清空日志文件。

```
event void Boot.booted() {
    call Leds.led0On();
    dbg("Boot", "Application booted.\n");
    resetLog();
    call AMControl.start();
}
```

修改函数 MilliTimer.fired()函数，给发送报文写入发送源节点号和发送时间，然后调用 writeLog()写入发送信息，再修改 debug 信息的输出格式，新增发送时间和源节点号。

```
time_t timep;
struct tm *p;

event void MilliTimer.fired() {
    int hour, min, sec;
    counter++;

    dbg("RadioCountToLedsC", "RadioCountToLedsC: timer fired, counter
is %hu.\n", counter);
    if (locked) {
        return;
    }
    else {
        radio_count_msg_t* rcm = (radio_count_msg_t*)call
Packet.getPayload(&packet, sizeof(radio_count_msg_t));
        if (rcm == NULL) {
            return;
        }

        rcm->counter = counter;
        rcm->src = call AMPacket.source(&packet);
        time(&timep);
        p = gmtime(&timep);
        hour = p->tm_hour;
        min = p->tm_min;
        sec = p->tm_sec;
        rcm->hour = htons(hour);
        rcm->min = htons(min);
        rcm->sec = htons(sec);
        if (call AMSend.send(AM_BROADCAST_ADDR, &packet,
sizeof(radio_count_msg_t)) == SUCCESS) {
```

```

        writeLog(0, rcm->src, hour, min, sec, 0, 0, 0, 0);
        dbg("RadioCountToLedsC", "RadioCountToLedsC:    %d:%02d:%02d
packet %d is sent from Node %d.\n", hour, min, sec, counter, rcm->src);
        locked = TRUE;
    }
}
}

```

修改 Receive.receive()函数，在接收到数据后，获取信息。在报文信息中可以看到发送的时间和源节点号，然后获取接收时的时间和节点号。调用 writeLog()写入接收信息，再修改 debug 信息的输出格式，新增接收时间、源节点号和目的节点号。

```

event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t len) {
    int cur_hour, cur_min, cur_sec, shour, smin, ssec;

    if (len != sizeof(radio_count_msg_t)) {return bufPtr;}
    else {
        radio_count_msg_t* rcm = (radio_count_msg_t*)payload;

        shour = ntohs(rcm->hour);
        smin = ntohs(rcm->min);
        ssec = ntohs(rcm->sec);
        time(&timep);
        p = gmtime(&timep);
        cur_hour = p->tm_hour;
        cur_min = p->tm_min;
        cur_sec = p->tm_sec;

        writeLog(1, rcm->src, shour, smin, ssec, TOS_NODE_ID, cur_hour,
cur_min, cur_sec);
        dbg("RadioCountToLedsC", "%d:%02d:%02d Node %d Received
packet %d of length %hhu from Node %d.\n", cur_hour, cur_min, cur_sec,
TOS_NODE_ID, rcm->counter, len, rcm->src);

        if (rcm->counter & 0x1) {
            call Leds.led0On();
        }
        else {
            call Leds.led0Off();
        }
        if (rcm->counter & 0x2) {
            call Leds.led1On();
        }
    }
}

```

```

        else {
            call Leds.led1Off();
        }
        if (rcm->counter & 0x4) {
            call Leds.led2On();
        }
        else {
            call Leds.led2Off();
        }
        return bufPtr;
    }
}

```

接下来编写仿真脚本代码。本次实验用了 6 个节点，读取拓扑文件 topo6.txt，用 meyer-heavy.txt 文件的噪声道数据建立噪声模型。

```

#!/usr/bin/python
from TOSSIM import *
import sys

t = Tossim([])
r = t.radio()
f = open("topo6.txt", "r")

for line in f:
    s = line.split()
    if s:
        print " ", s[0], " ", s[1], " ", s[2];
        r.add(int(s[0]), int(s[1]), float(s[2]))

t.addChannel("RadioCountToLedsC", sys.stdout)
t.addChannel("Boot", sys.stdout)

noise = open("meyer-heavy.txt", "r")
for line in noise:
    str1 = line.strip()
    if str1:
        val = int(str1)
        for i in range(1, 7):
            t.getNode(i).addNoiseTraceReading(val)

for i in range(1, 7):
    print "Creating noise model for ",i;
    t.getNode(i).createNoiseModel()

```

```
t.getNode(1).bootAtTime(100001);  
t.getNode(2).bootAtTime(200001);  
t.getNode(3).bootAtTime(300001);  
t.getNode(4).bootAtTime(400001);  
t.getNode(5).bootAtTime(500001);  
t.getNode(6).bootAtTime(600001);  
  
for i in range(10000):  
    t.runNextEvent()
```

拓扑文件 topo6.txt 中，3 个数值指明一条链路：源节点、目标节点和增益。
文件内容如下：

```
1 2 -31.0  
1 3 -32.0  
1 4 -33.0  
1 5 -34.0  
1 6 -35.0  
2 1 -40.0  
2 3 -41.0  
2 4 -42.0  
2 5 -43.0  
2 6 -44.0  
3 1 -60.0  
3 2 -61.0  
3 4 -62.0  
3 5 -63.0  
3 6 -64.0  
4 1 -80.0  
4 2 -81.0  
4 3 -82.0  
4 5 -83.0  
4 6 -84.0  
5 1 -30.0  
5 2 -31.0  
5 3 -32.0  
5 4 -33.0  
5 6 -34.0  
6 1 -70.0  
6 2 -71.0  
6 3 -72.0  
6 4 -73.0  
6 5 -74.0
```


TOSSIM 不但可以仿真无限电的传播模型，还可以利用就近匹配算法 (CPM) 来模拟射频噪声、节点间的相互干扰以及外部信号源节点的干扰。CPM 算法利用噪声道文件产生一个统计模型。通过调用节点对象的 `addNoiseTraceReading` 命令创建噪声模型。在 `tos/lib/tossim/noise` 目录下有几个简单的噪声文件。例如，`meyer-heavy.txt` 文件保存的是斯坦福大学 Meyer 图书馆的噪声道数据，它的前十行如下：

```
-39
-98
-98
-98
-99
-98
-94
-98
-98
-98
```

在脚本代码后面在添加计算数据的丢包率和时延的部分。首先逐行读取日志文件，然后分别存储发送数据总数、成功接收总次数和时延总和。由于日志只有发送和接收时间，需要由脚本计算时延。读取后日志文件后，可以得到每个节点发送和接收到的数据总数，以及接收到的所有数据的传输时延的总和。最后，利用这些数据，计算六个节点的收发数据总数、总丢包率和平均时延。

```
Send = [0] * 6
Send_Succ = [0] * 6
Delay_Sum = [0] * 6

def cut_line(line):
    line_list = line.split(' ')
    if (len(line_list) < 3):
        return
    send_time_list = line_list[2].split(':')
    send_hour = int(send_time_list[0])
    send_min = int(send_time_list[1])
    send_sec = int(send_time_list[2])

    type = line_list[0]
    send_node = int(line_list[1])
    if (send_node == 0):
        return
    if (type == 'Send'):
        Send[send_node - 1] += 5
    else:
        recv_node = int(line_list[3])
        Send_Succ[send_node - 1] += 1
```

```

    recv_time_list = line_list[4].split(':')
    recv_hour = int(recv_time_list[0])
    recv_min = int(recv_time_list[1])
    recv_sec = int(recv_time_list[2])

    delay = (recv_hour - send_hour) * 3600 + (recv_min - send_min) * 60 + recv_sec -
send_sec
    delay = delay * 1000
    Delay_Sum[send_node - 1] += delay

log_file = open('log.txt','r')
while 1:
    line = log_file.readline()
    cut_line(line)
    if not line:
        break

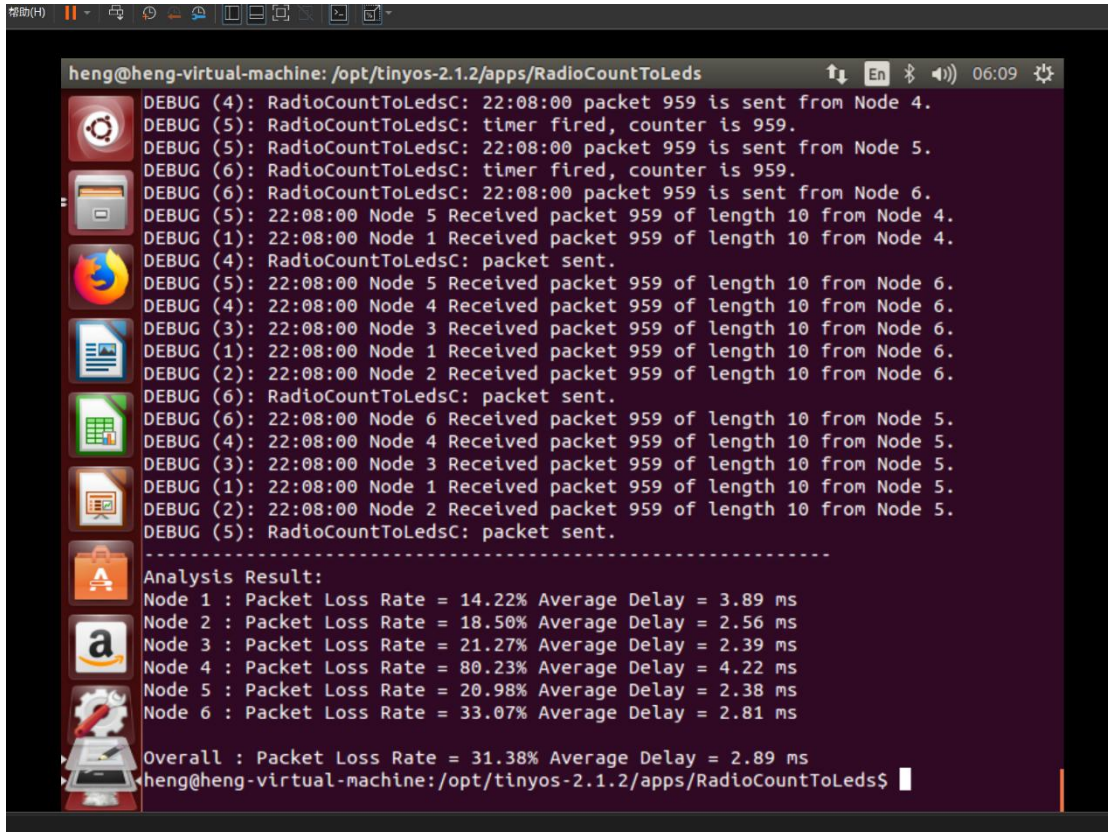
tot_send = 0
tot_succ = 0
tot_delay = 0

print "-----"

print "Analysis Result:"
for i in range(6):
    print ('Node %d : Packet Loss Rate = %.2f%% Average Delay = %.2f ms' %
        (i+1, (1 - 1.0 * Send_Succ[i] / Send[i]) * 100, 1.0 * Delay_Sum[i] / Send_Succ[i]))
    tot_send += Send[i]
    tot_succ += Send_Succ[i]
    tot_delay += Delay_Sum[i]
print "
print 'Overall : Packet Loss Rate = %.2f%% Average Delay = %.2f ms' % ((1-1.0*tot_succ /
tot_send) * 100, 1.0 * tot_delay / tot_succ)

```

在终端输入 `make micaz sim` 编译并生成 Tossim 应用程序，然后运行脚本程序。得到以下结果：



```
heng@heng-virtual-machine: /opt/tinyos-2.1.2/apps/RadioCountToLeds
DEBUG (4): RadioCountToLedsC: 22:08:00 packet 959 is sent from Node 4.
DEBUG (5): RadioCountToLedsC: timer fired, counter is 959.
DEBUG (5): RadioCountToLedsC: 22:08:00 packet 959 is sent from Node 5.
DEBUG (6): RadioCountToLedsC: timer fired, counter is 959.
DEBUG (6): RadioCountToLedsC: 22:08:00 packet 959 is sent from Node 6.
DEBUG (5): 22:08:00 Node 5 Received packet 959 of length 10 from Node 4.
DEBUG (1): 22:08:00 Node 1 Received packet 959 of length 10 from Node 4.
DEBUG (4): RadioCountToLedsC: packet sent.
DEBUG (5): 22:08:00 Node 5 Received packet 959 of length 10 from Node 6.
DEBUG (4): 22:08:00 Node 4 Received packet 959 of length 10 from Node 6.
DEBUG (3): 22:08:00 Node 3 Received packet 959 of length 10 from Node 6.
DEBUG (1): 22:08:00 Node 1 Received packet 959 of length 10 from Node 6.
DEBUG (2): 22:08:00 Node 2 Received packet 959 of length 10 from Node 6.
DEBUG (6): RadioCountToLedsC: packet sent.
DEBUG (6): 22:08:00 Node 6 Received packet 959 of length 10 from Node 5.
DEBUG (4): 22:08:00 Node 4 Received packet 959 of length 10 from Node 5.
DEBUG (3): 22:08:00 Node 3 Received packet 959 of length 10 from Node 5.
DEBUG (1): 22:08:00 Node 1 Received packet 959 of length 10 from Node 5.
DEBUG (2): 22:08:00 Node 2 Received packet 959 of length 10 from Node 5.
DEBUG (5): RadioCountToLedsC: packet sent.
-----
Analysis Result:
Node 1 : Packet Loss Rate = 14.22% Average Delay = 3.89 ms
Node 2 : Packet Loss Rate = 18.50% Average Delay = 2.56 ms
Node 3 : Packet Loss Rate = 21.27% Average Delay = 2.39 ms
Node 4 : Packet Loss Rate = 80.23% Average Delay = 4.22 ms
Node 5 : Packet Loss Rate = 20.98% Average Delay = 2.38 ms
Node 6 : Packet Loss Rate = 33.07% Average Delay = 2.81 ms
Overall : Packet Loss Rate = 31.38% Average Delay = 2.89 ms
heng@heng-virtual-machine:/opt/tinyos-2.1.2/apps/RadioCountToLeds$
```

从结果中，可以看到每次发送和接收都会在终端输出对应的信息。在仿真结束后，输出了每个节点的丢包率和时延，最后输出全部节点的丢包率和时延。总丢包率为 31.38%，时延是 2.89ms。

实验总结

通过本次实验，使用了 TOSSIM 应用程序进行仿真，学会了修改 nc 文件和头文件，编辑所需接口和部件，实现所需的额外功能。同时，经过编写脚本程序和修改 nc 文件，学到了仿真传感器网络数据的发送和接收的流程。例如，在本次实验，脚本程序提供了拓扑网络文件，建立了噪声模型。RadioCountToLeds 应用程序定时控制节点向其他所有节点广播发送数据，也处理了接收到的数据。