

**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 鄭毓恒**

**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 |

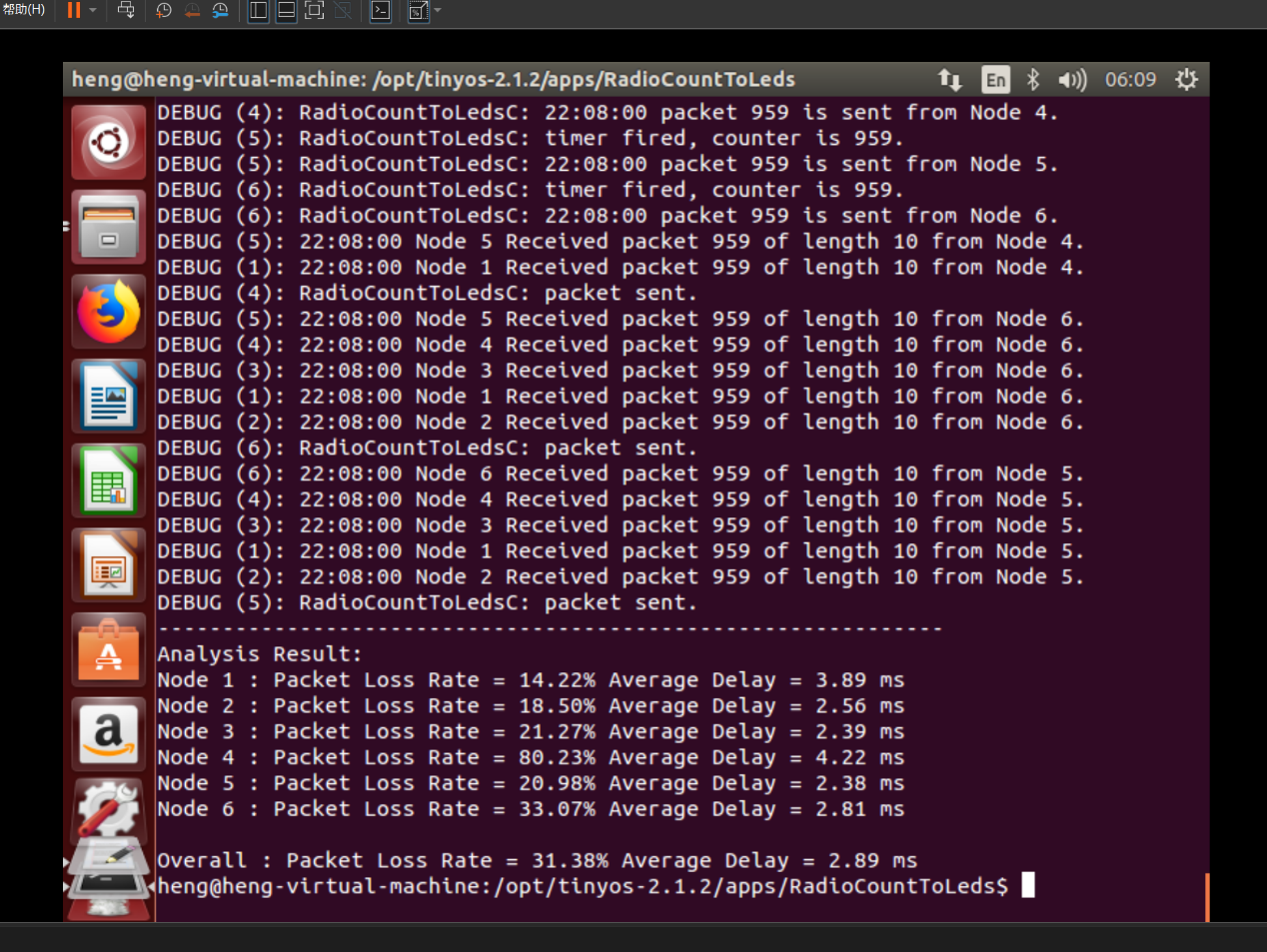
TOSSIM不但可以仿真无限电的传播模型，还可以利用就近匹配算法（CPM）来模拟射频噪声、节点间的相互干扰以及外部信号源节点的干扰。CPM算法利用噪声道文件产生一个统计模型。通过调用节点对象的addNoiseTraceReading命令创建噪声模型。在tos/lib/tossim/noise目录下有几个简单的噪声文件。例如，meyer­heavy.txt文件保存的是斯坦福大学Meyer图书馆的噪声道数据，它的前十行如下：

|  |
| --- |
| -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：



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。