**清华大学**

2012

**随机链状无线传感网的轻量级传输协议设计**

**专业课程实践论文**

软件92 陈星丞 2009013227，软件92 段春晖 2009013225

**随机链状无线传感网的轻量级传输协议设计**

目录

[一、 问题描述 2](#_Toc334727936)

[1.1项目背景 2](#_Toc334727937)

[1.2解决方案 2](#_Toc334727938)

[1.3研究问题 2](#_Toc334727939)

[二、 前期准备 3](#_Toc334727940)

[三、 协议设计 3](#_Toc334727941)

[3.1第一阶段 3](#_Toc334727942)

[3.1.1协议描述 3](#_Toc334727943)

[3.1.2自动机图示 4](#_Toc334727944)

[3.2第二阶段 6](#_Toc334727945)

[3.2.1协议描述 6](#_Toc334727946)

[3.2.2自动机图示 7](#_Toc334727947)

[四、 代码实现 9](#_Toc334727948)

[五、 结果分析 9](#_Toc334727949)

[5.1 输出说明 9](#_Toc334727950)

[5.2 参数调整 10](#_Toc334727951)

[六、 心得体会 13](#_Toc334727952)

# 问题描述

## 1.1项目背景

* 高可靠地收集空间上线性等距摆放的若干监测点信息；
* 除信息汇集点外，所有监测点无电源；
* 信息搜集周期为半小时左右，网络运行时间不超过一个月；
* 网络建成前，监测点之间的顺序是随机的；网络生成后，监测点之间的顺序固定不变；
* 监测点之间的最小距离10~20m；
* 节点的程序空间不超过8K字节，最好4K字节以内。

## 1.2解决方案

* **方案一**

网络由传感节点和汇集节点构成，传输采用单一的链状中继策略。



* **方案二**

网络由传感节点、区域汇集节点和汇集节点构成，传输采用分级链状中继策略。 

## 1.3研究问题

1. 低能耗的MAC协议设计
2. 高可靠性的自组织路由协议

# 前期准备

* 学习了NS3的使用，包括环境的配置、教程和手册的学习。
* 学习了802.15.4协议的相关内容，阅读了Mac和Routing相关的综述文献：
* A\_survey\_on\_routing\_protocols\_for\_wireless\_sensor\_networks(Kemal Akkaya, Mohamed Younis)；
* MAC\_Protocols\_for\_Wireless\_Sensor\_Networks\_A\_Survey（Ilker Demirkol, Cem Ersoy, and Fatih Alagöz, Bogazici University）。

# 协议设计

我们的协议设计经历了两个主要阶段。

## 3.1第一阶段

### 3.1.1协议描述

1. 采用的方法原型为PEGASIS或PEGASIS-hierachical。

不同的节点有不同的能力，每隔几个节点，存在一个超节点，其有远距离传输的能力。

1. 安装网络的过程如下：
2. 工作人员从车头开始，依次给每个车厢安置一个传感器，将它开启，并设定它的编号（从车头为0开始，依次递增1），并且标定它是不是在车尾。
3. 自检过程。此时还未发车，所有节点处于同一条直线上。利用过程Work进行自检，若不成功，则提醒工作人员修正。即自检过程也是工作过程，只不过此时车并没有开动，发现问题可以立刻处理。
4. 工作流程。每隔一段时间（例如一个小时），所有传感节点进入工作状态，执行过程Work。当每个节点完成自己的工作以后，就自行进入低耗状态待机。每次Work完成以后，车头会把结果向总部汇报。

注：**关于不同列车之间信号可能干扰的问题说明**：

如果一列车的结点在自组织的时候，不管你采取什么路由方法，如果旁边有一辆车也在进行传感器的自组织，最终的结果一定会把两列车的结点缠绕起来。这是因为传感器没办法识别信号的方向，它不能分辨出信号来自于平行于列车方向，还是不平行与列车方向。  
所以我们得到一个假设，就是一列车在自组织的时候是没有另一列车同时在自组织的。  
于是只需要让结点多一个状态，即 “我现在不是已经编好组号了？”  
如果不是，那么就不进入工作状态。我们就假设这样一个工作场景，在车头结点上，工作人员把所有所需的传感器打开（重启），它们都在没有编好组号的状态。而此时让车头结点发送一个广播，广播的内容包括“这是一个组号广播，当前列车的组号”。当前列车的组号可以用一个随机算法生成，也可以直接用车头结点出厂时的GUID。然后所有其他结点收到信息以后，告诉工作人员，我已经知道我的组号了。工作人员这时候就把这些普通结点依次按我之前的设计，输入其在当前组中的序列号，然后放到相应车厢上去。然后所有信息的通信都需要验证这条信息是不是与我同一组的结点所发？如果是，才处理，否则drop.

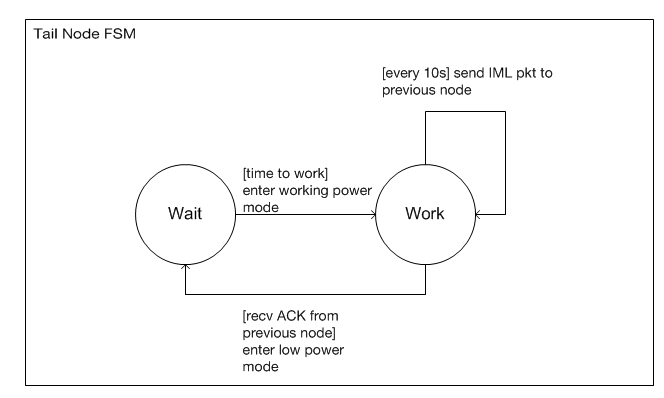
1. 关于节点规格的说明：

只生产三种规格的节点：其中一种是车头节点，其有电力供应，性能强大；一种是普通中间节点，其性能最低，只能把信号传到相邻的1~2个节点上；最后一种是中间的超节点，其内存大，可以存储大量信息，同时可以将信息发至更远，如10个节点。

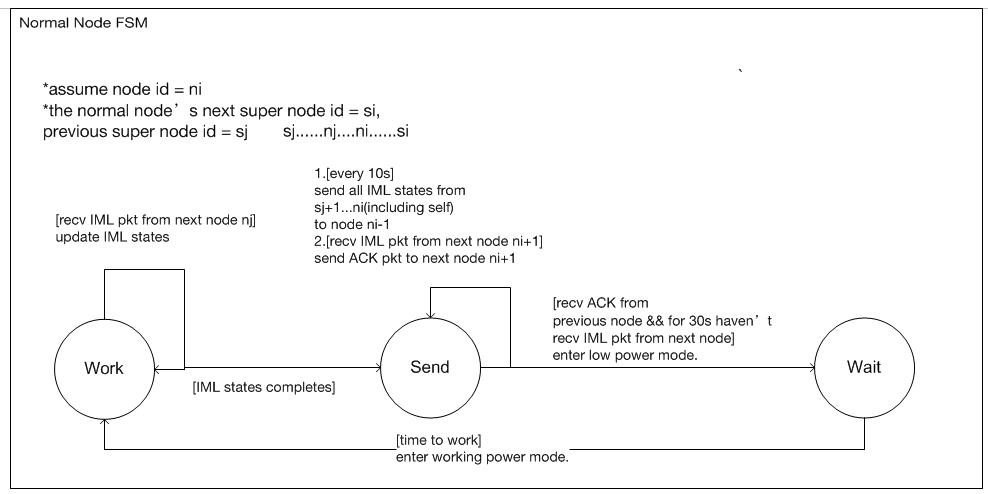
### 3.1.2自动机图示

与上述协议对应的节点的状态机如下：

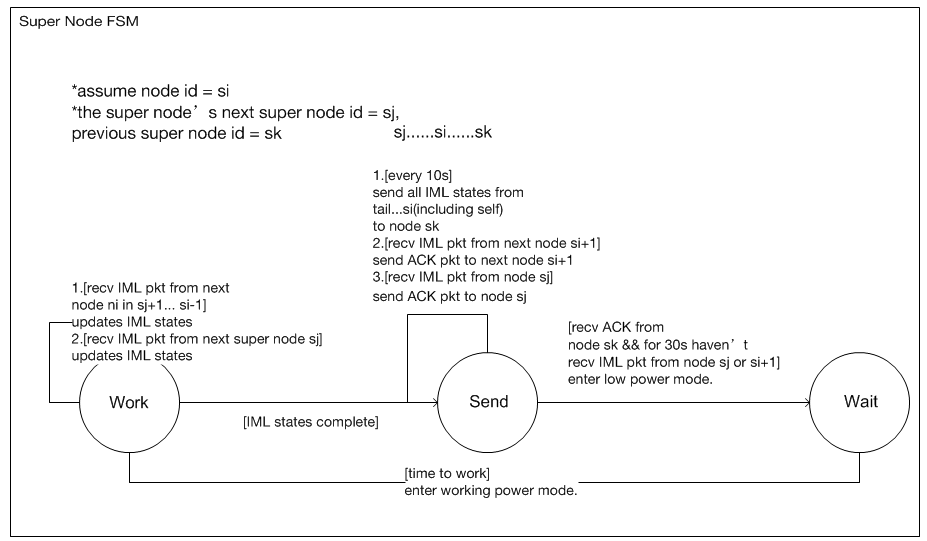
车尾节点：



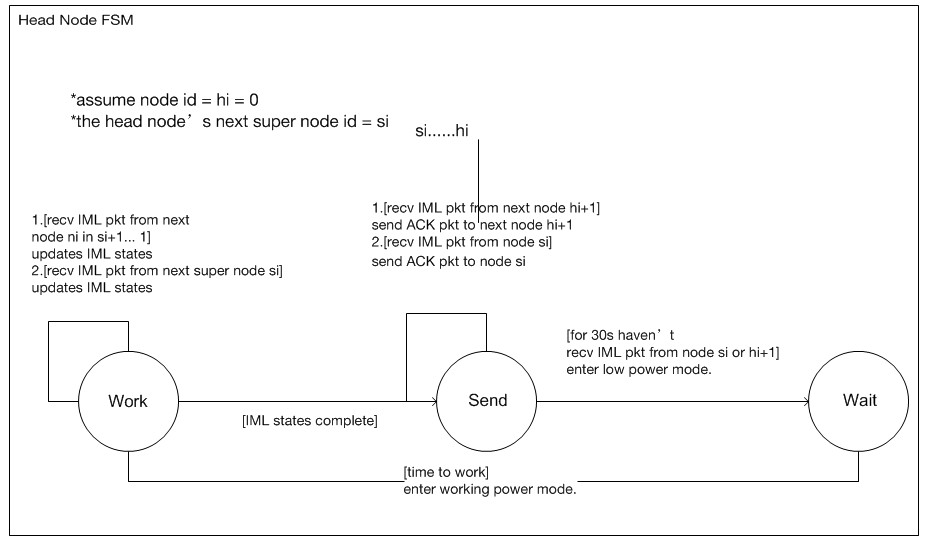
普通中间节点：



超级中间节点：



车头节点：



## 3.2第二阶段

在后来具体实现协议的过程中，我们发现第一阶段设计的协议存在一些不必要的成分，于是我们结合实际需要重新设计了一个较为完整且易于实现的协议。

### 3.2.1协议描述

首先做如下说明：

1. 节点发送的信号会按照一定的规律衰减，当节点的接收信号强度小于某个阈值时，该信号视为失效。
2. 节点发包时存在丢包率（为0到1的实数）。
3. 每个节点都有自己的信号发射功率，该功率与信号衰减规律相结合可以大致得到该节点的信号传输范围。而且通过发射功率可以估算出节点的能量消耗。
4. 节点收到一个信号后，可以得知当前收到信号的强弱。
5. 对于可能多条信号同时到达同一个节点的情况，可以：只接收信号最强的或只接收到达时间最早的信号。我们最终采用的方法是接收到达时间最早的信号。
6. 每个节点有以下基本属性：组号；在组中的序号；信号发射功率等。

其次，考虑具体的信号传输策略，我们设计了如下策略：

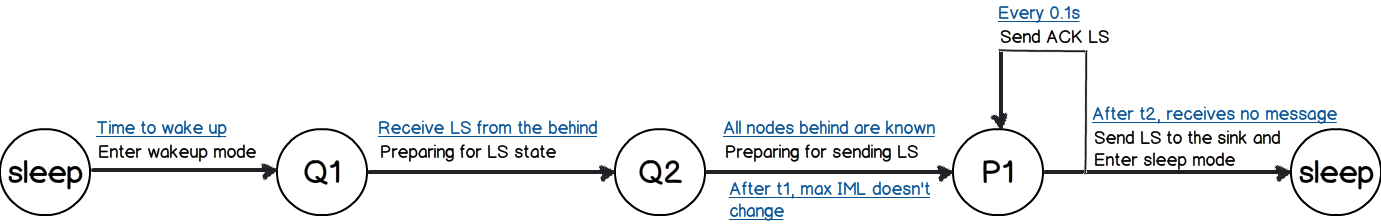
前提：所有节点都已经知道自己的组别和编号（编号规律依然是车头为0，然后向车尾依次递增）。

* 除了车头节点，其他所有节点均为普通节点。
* 对于一个节点来说，当它wakeup后，每隔一定时间（例如10s）发送IML信息，当它收到完整的所有后面节点的信息后，停止发送IML信息，再以一定频率（例如0.1s）发送完整信息（简称LS信息）给前面节点，同时往后面节点发送ACK，直到某个阈值（例如20s）再也没有收到后面节点的广播信息。
* 当某个节点接收到来自它前面节点的ACK后，则停止发送LS和IML信息。若此时已经经过发送ACK的阶段，则sleep。否则，转而向其后面节点发送ACK直到某个时间（预定阈值，例如10s）后仍然没有收到来自其后节点的广播，此时进入sleep状态。节点sleep的条件为：收到ACK且自己没有在发ACK。

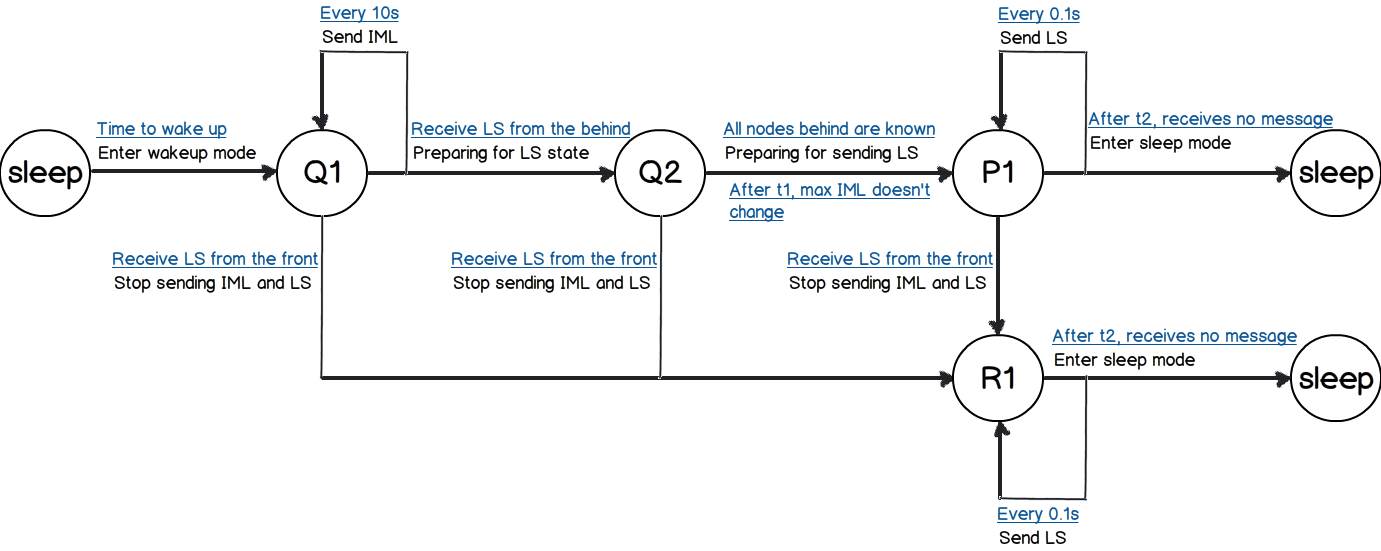
经过后来对协议不断的反复思考和验证，我们发现ACK信息可以和LS信息合并，即二者本质上是同一种信息。于是，现在只存在两种类型的信息：IML和LS。

### 3.2.2自动机图示

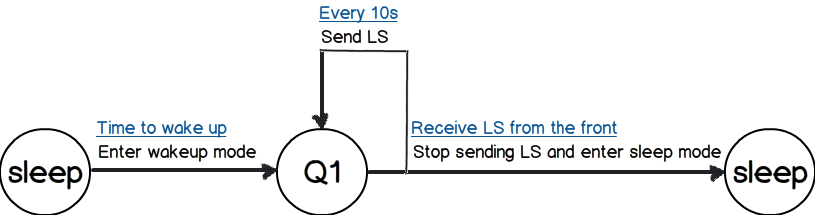
HeadNode（车头节点）状态图：



NormalNode（普通中间节点）状态图：

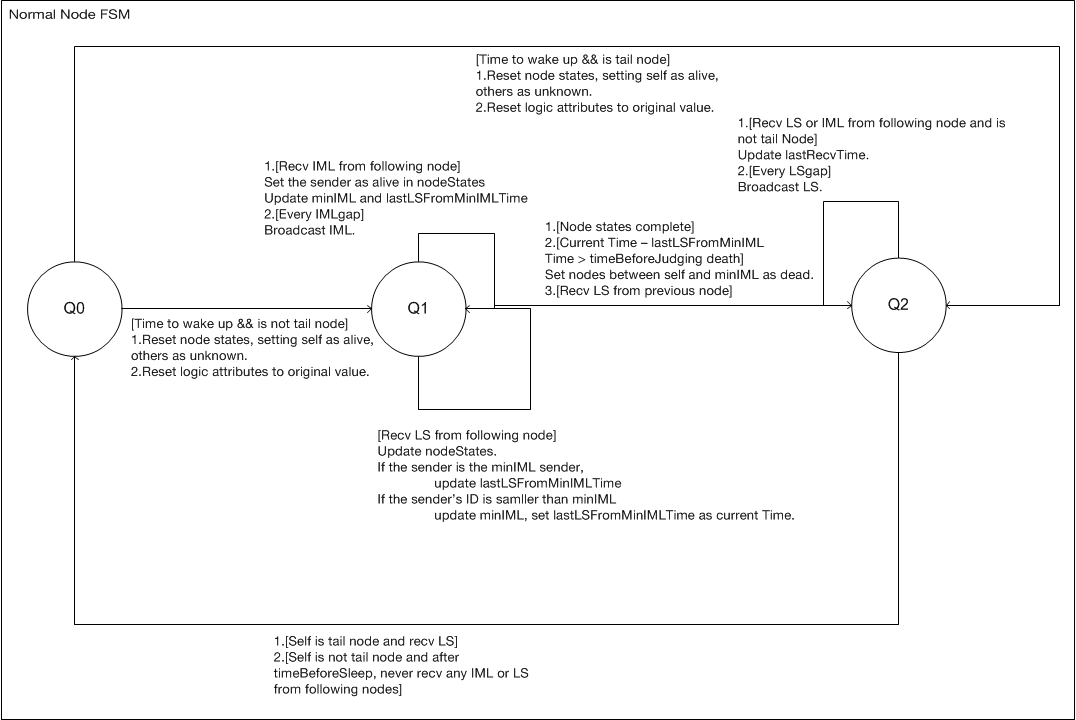


TailNode（车尾节点）状态图：

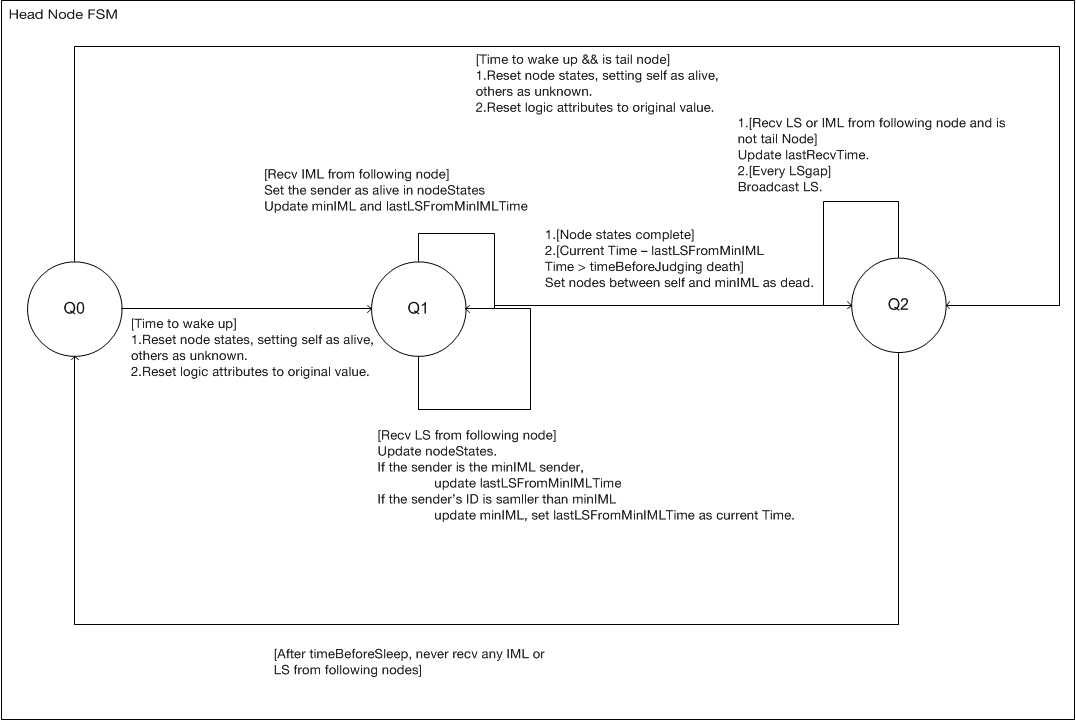


经过更深入的分析，发现上述状态图还可以进行简化，于是得到简化的状态图如下：

普通节点：



车头节点：



注：新的状态图基于以下原则：

1.每一个结点只负责自身与其后结点的状态记录与判定，对其之前的结点不加处理。

2.当一个结点发送Ls即意味着，它或者它之前的某结点已经完成的对自身与其后结点的判定。这时显然其后方结点的工作已没有必要，因为信息已经收集了。LS信息包括着信息发送者已知的结点的状态。LS信息接收者可以通过这些信息对自己的结点状态数组进行更新。

3.不做不必要的发送。即当一个结点B收到前面的结点A发出的Ls时，就可以认为前面的LS已经把其后的所有结点的状态判定完成了。所以A之后的所有结点理论上都可以进入休眠状态，因为结点信息只需要通过A往前传递就可以了。于是当前结点B的工作目的就转换为让其后的结点进入睡眠状态。

4.Q1状态表示当前结点的状态记录工作还没有完成。每一个结点定期发送IML信息，即I am alive，告知其前方结点自己的存在。

5.Q2状态表示当前结点的状态记录工作已经完成，当前工作的目标是让其后结点进入休眠状态。

6.Q0状态表示当前结点正在进行休眠。

# 代码实现

原先的计划，我们预计采用NS3来实现网络模拟。但是NS3中并不存在802.15.4模块，同时，现在有两个开源小组正在并行开发这个模块，但都没有可利用的代码。而NS3的上手难度又非常大，而且它考虑了很多我们这次仿真并不关心的信息，比如说硬件层次结构等等。

于是我们决定，利用C++这个非常高效的语言，手写一个网络模拟器，用以实现对我们设计的协议的模拟。利用OOP的强大工程优势，我们迅速完成了一个离散事件模拟器，来仿真信号的传输以及节点状态的统计。最终成功实现了上述协议描述的各项功能并在有节点损坏的情况下，仍能很好地工作。

* 操作系统：Windows7 Ultimate 64bit
* 开发环境：Microsoft Visual Studio 2010
* 编程语言：C++

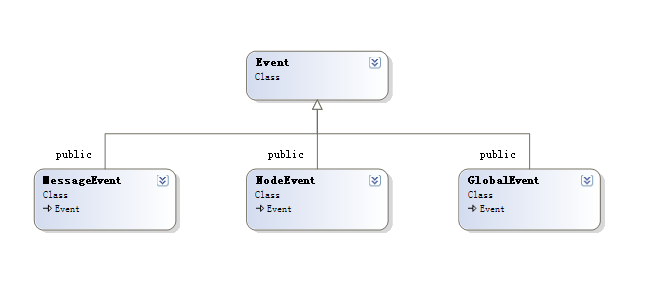
源代码参见src文件夹。

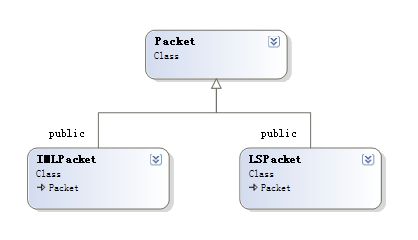
这里做如下说明：

由于节点发送一次信号的时长是非常短而且很难估计的，因此代码实现的过程中假定了收发信号是一瞬间的事，即节点发送信号在某时刻瞬间完成，而其他节点也只有在该时刻才有可能接收到该节点发送的信号。而且，对于某一特定节点来说，它会在某一时刻处理达到它的全部信号。

这个假设在实际应用中也是可行的，并不会使仿真结果和实际情况有很大偏差。

另附上相应的类图：







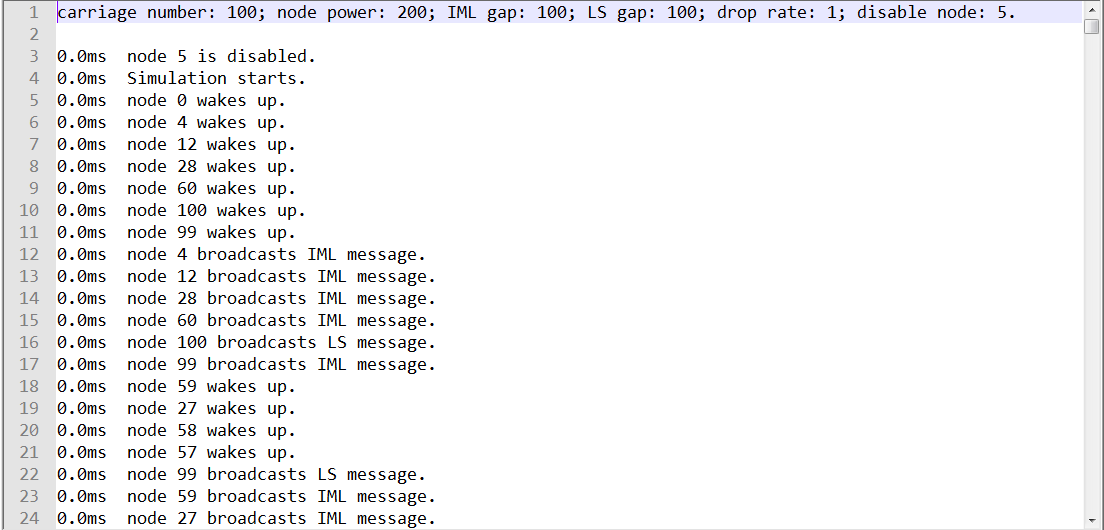
# 结果分析

## 5.1 输出说明

程序ctrl+F5运行后，会生成一个XCNS.log文件，该文件保存了仿真生成的实验结果。

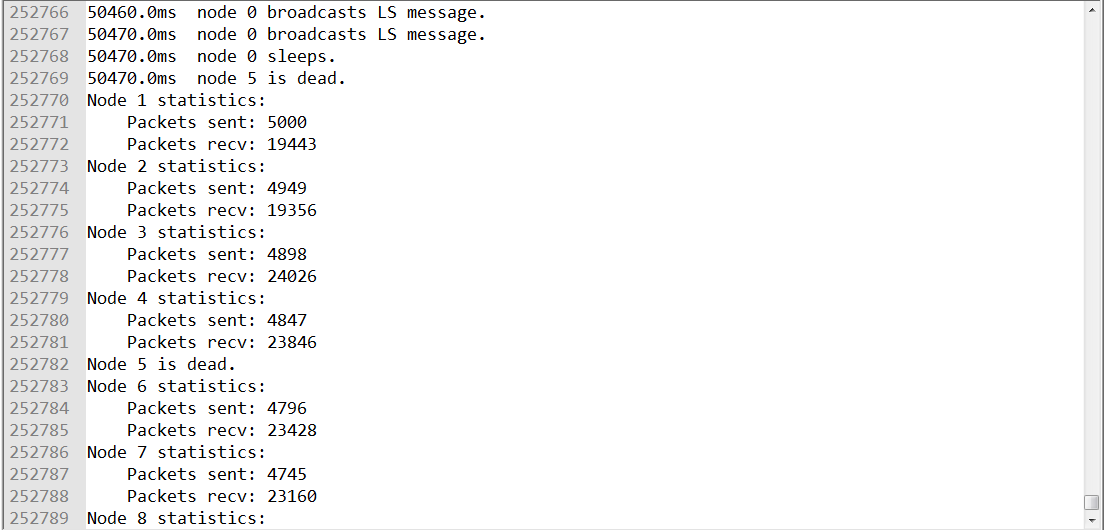
实验中可调的参数有六种，分别为：carriage number（车厢节数）, node power（发送功率，直接影响到信号能传递的车厢节数）, drop rate（丢包率）, IML gap（节点发送IML信号的时间间隔）, LS gap（节点发送LS信号的时间间隔）和disable node（坏掉的节点编号）。

result文件夹中的所有.log文件都是不同参数下仿真生成的实验结果。其中每个.log文件的第一行标明了该结果所对应的上述六种参数值。例如：XCNS\_1.log文件部分内容如下：



通过第一行，我们可以知道本次仿真共有100节车厢，节点的发送功率是200(mW)，发送IML和LS信号的时间间隔均为100个时间单位（其中每个时间单位为0.1ms），信号的丢包率为1%，而且编号为5的节点坏掉了。

.log文件记录了预定时间内信号发送和接收的情况以及每个节点的状态变化。在.log文件最后，对每个节点的收发包数进行了统计，例如：



这样有助于对结果有个比较直观的了解。

## 5.2 参数调整

1. **对carriage number进行调整：**

设置的值分别为2、5、10、15、20、50、100。通过分析仿真结果发现，随着车厢节数的增加，仿真时间明显增长，这是显而易见的，因为实验需要确认每个节点的存在状态，节点增加时仿真的时间自然会变长。

1. **对node power进行调整：**

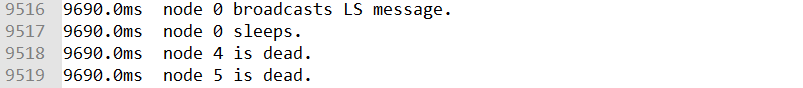
设置的值分别为100、200、800、1800和3200（单位均为毫瓦），对应的信号能传递的车厢节数分别为2节、3节、6节、9节和12节。通过分析仿真结果发现，node power对实验结果影响不大，在上述五种情况下，仿真的时间是相同的，说明node power对仿真时间基本没有影响。但其对节点接收包数有较大影响，因为node power增大时，节点的发送信号范围变大，能接收到信号的节点数也增多，因此各个节点的接收包数普遍增加。

1. **对drop rate进行调整：**

设置的值分别为1、3、5、10、15、20、30、40、50、60、70、80、90（单位均为百分制）。通过分析仿真结果发现，drop rate会影响每个节点的收发包数。drop rate对节点发送包数影响较小，随着drop rate的增加，节点发送包数有轻微的减少；但对节点接收包数有较大影响，随着drop rate的增加，节点接收包数有较大幅度的减少。当drop rate增加到90%时，每个周期的仿真时间明显增长，说明由于丢包率太大而影响到了仿真的时间。

1. **对IML gap进行调整：**

设置的值分别为10、50、100、500、1000、3000、5000、6000（单位均为1个时间单位，即0.1ms）。通过分析仿真结果发现，IML gap会影响发收包数，这是显而易见的，IML gap增大时，节点的发收包数显然会随之减少。另外，IML gap的值是有一定范围的，不能太大，因为这里有个限制条件：IML gap <= timeBeforeJudgingDeath（判断一个节点死亡时的时间阈值）。在此我们设置的timeBeforeJudgingDeath为5000，因此当IML gap大于5000时，结果就会出错。例如仿真时令IML gap = 6000，时，发现结果文件中输出了如下结果：



仿真出node 4和node 5都是死亡状态，这跟我们参数的设定情况不一致，因为我们只假定了node 5是坏掉的。因此，仿真结果出现了错误，这正是由于IML gap的设定不正确而造成的。因而需要注意IML gap和timeBeforeJudgingDeath的约束关系。

1. **对LS gap进行调整：**

类似于IML gap的情况，设置的值分别为10、50、100、500、1000、3000、5000、6000、10000（单位均为1个时间单位，即0.1ms）。通过分析仿真结果发现，LS gap同样会影响发收包数，这也是显而易见的，LS gap减小时，节点的发收包数显然会随之增加。而且，通过观察发现，LS gap对发收包数的影响程度较IML gap大，这是因为节点发送LS信息的频率高于IML信息。另外，LS gap对仿真时间有一定影响，LS信息发送频率较高时，仿真时间也会较短。在此，我们注意到，理论上LS gap的值也需要有一定范围：LS gap <= timeBeforeSleep（判断一个节点是否可以进入睡眠状态的时间阈值）。在此我们设置的timeBeforeSleep为5000，因此理论上LS gap的值不应该大于5000，但实际上即便LS gap为10000时，仿真结果看起来好像仍然没有错误。但是经过逐个分析各个节点是否都进入睡眠状态后，我们发现当LS gap > timeBeforeSleep时，有一些节点在仿真结束后未正常进入sleep状态，例如当LS gap = 10000时，编号为14、9、5、1的节点都未sleep。值得注意的是，此时的丢包率仅为1%，当丢包率较大时，经过实验发现将有更多的节点无法进入sleep状态。例如，我们将drop rate调整为30，当LS gap = 10000时，编号为17、15、14、13、12、10、8、6、5、4、3、2、1的13个节点都无法正常sleep。因而， LS gap和timeBeforeSleep的约束关系值得注意。

1. **对disable node进行调整：**

Disable node的个数可以为一个或多个，因此对disable node的设置随机选取了以下值（车厢节数为20）：

* + 1；
  + 5；
  + 10；
  + 15；
  + 3、8；
  + 7、8；
  + 9、19；
  + 1、19；
  + 1、2；
  + 3、4、5、6；
  + 8、12、16、18；
  + 1、3、5、7、9、11、13、15、17、19；
  + 1、2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17、18、19；

通过实验发现，上述红色标识出来的两组数据无法完成仿真，因为此时设置的节点发射功率最大仅能够发送信号到邻近的三节车厢的距离。因此，当有连续三节或三节以上车厢的节点同时坏掉时，信号就无法继续传递下去，也就无法完成整个列车的仿真。

另外，当节点坏掉的个数较多时，仿真需要的时间相对较少。这是因为坏掉节点很多时就省去了一些节点传递中间信号所产生的时间消耗。

* **最佳参数的选取：**

通过五十多组不同参数的实验，当车厢节数（一般为20节）和坏掉节点数（这里假定为1）固定的情况下，得到了一组相对而言最佳的参数取值，如下：

* Carriage number = 20
* Node power = 200(mW)
* Drop rate = 1(%)
* IML gap = 100(0.1ms)
* LS gap = 10(0.1ms)
* Disable node = {5}

在该最佳参数下，仿真完成的时间为9519ms。

# 六、空间需求分析

由于成本需求的控制，这个项目要求能够尽量减少节点的空间存储。目前这个协议很好的完成了这个需求。此处仅分析普通结点，头结点造价限制并不大。

## 内存空间：（以100节车厢为例）

nodeStates:一个byte数组，长度为100/4 = 25byte

id:一个int，4byte

groupID:一个int，4byte

isSleeping:一个bool,1byte

minIML：一个int，4byte

lastRecvTime:一个int，4byte

lastLSFromMinIMLTime:一个int，4byte

总计：46byte，远远小于1KB。可以说对内存空间的需求很小。

## 程序空间：（以100节车厢为例）

由状态机可知，普通结点只有三个状态，而且C++代码实现其逻辑只需要233行（以NormalNode.cpp估算，其实上是一个上界）。故而程序空间也不会很大。应该只需要极低的成本就可以完成。

# 八、心得体会

这次小学期是一个比较实际的选题，感觉也比较有意思。可能是之前和王老师的沟通没有很具体很及时，因此对任务的理解有所偏差，但是最后老师尊重了我们的意见，让我们用自己熟悉的方法完成了仿真，让我们在理论和实践方面都有所收获。再次感谢老师~