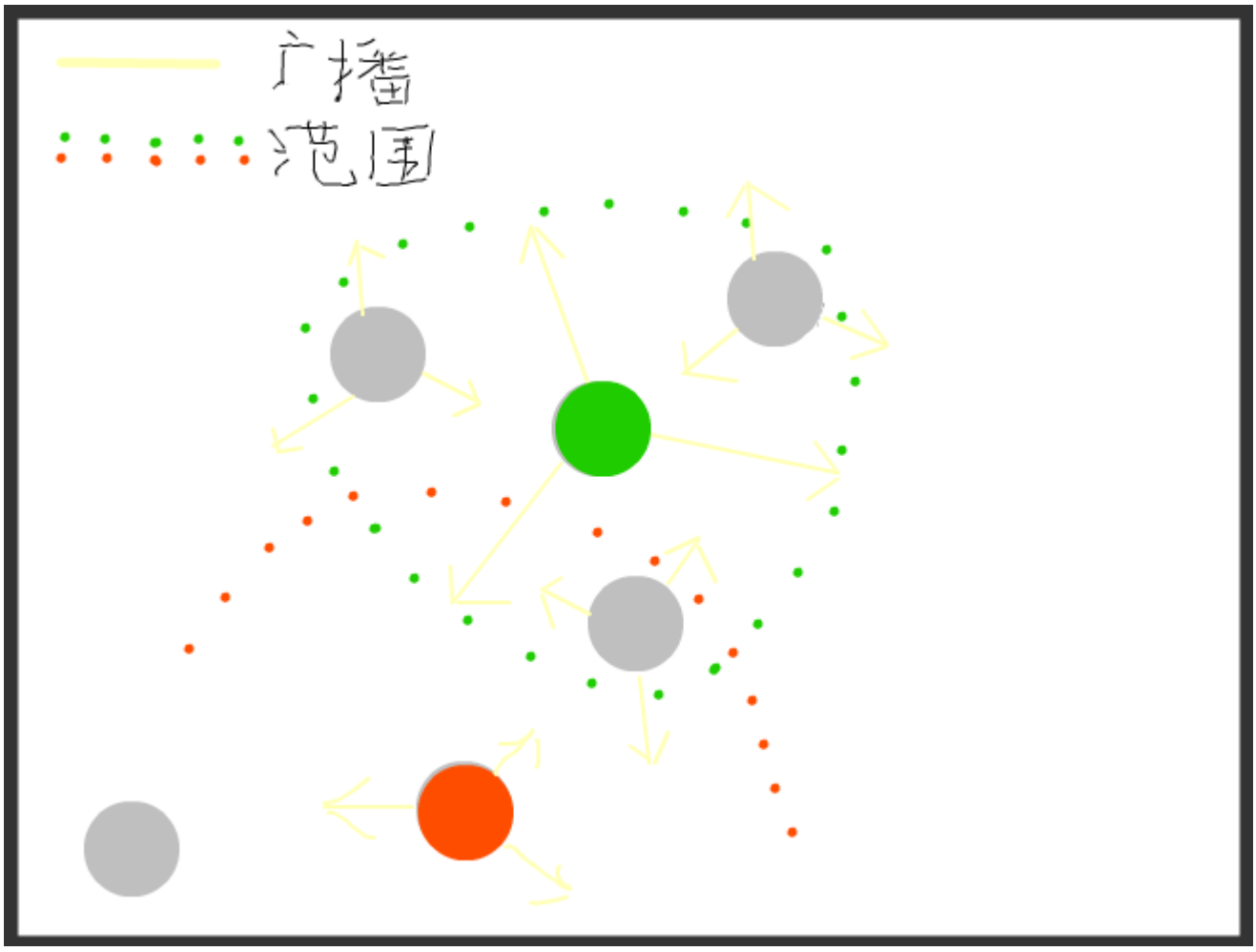
题目： 基于 TinyOS 的无线邻居表建立和维护



项目的功能分解：

1. 节点不断广播数据包（数据包内容为节点自己的 **ID**）

功能概述： 节点广播数据包的行为由周期为 SENDER_PERIOD_MILLI（等于 250ms，在项目头文件中定义）的定时器 Timer0 来触发。

要解决的问题： 定义广播数据包的格式，在触发定时器的事件中写入广播数据包的逻辑（这里我们用到了 ActiveMessage 的相关组件进行数据包的收发）

备注： ActiveMessage 相关组件实现了基于无线的单跳路由协议，也就是说仅确保在节点通信范围内的节点之间可以相互通信，并不确保节点与自己邻居节点的邻居进行通行（即间接通信）

2. 节点不断收到来自周围节点广播的数据包并对数据包进行处理

功能概述： 因为节点的广播和接收数据包都用到了 ActiveMessage 的相关组件，

ActiveMessage 其实是单跳路由协议的实现，所以只要是在无线信号的接收范围内的节点就会触发 receive 事件，来对接受到的数据包进行处理，数据包的处理逻辑大概可以描述为：如果发送方不在当前节点的邻居表中就将其加入邻居表，如果已经在邻居表中，我们会对邻居表信息进行更新。

要解决的问题： 其实这里并不是简单的将发送方加入邻居表中，而是会对发送方做一个评估，比如判断发送方是否第一次发来数据包，如果这样的话显然将发送方判定为邻居节点则为时尚早，当然我们的评估标准并不像例子中所说这么简单的，后续会有专门的内容对评估邻居节点做相应的介绍

3. 电脑发出邻居表查询请求

功能概述： 电脑通过与某个特殊节点的连接来发送想要查询邻居表的节点 id 并且对应 id 的节点可以感知到电脑的请求，为下一步返回邻居表信息做准备

要解决的问题： 电脑与特殊节点的通信显然涉及到串口通信；当特殊节点收到电脑发来的 id 后，如何将 id 扩散到整个网络是个棘手的问题，该问题可以通过 TinyOS 预先实现的 Dissemination 无线网络协议解决

备注： Dissemination 相关组件实现了无线网络协议 Dissmination，该协议的主要内容为，生产/消费者模型，具体来说只要网络中某个节点（即生产者）对变量做了修改（更新），那么位于当前无线网络中的所有节点（消费者）都可以感知得到（具体来讲消费者节点的相应事件会被触发）

4. 被请求节点将邻居表信息传送给电脑

功能概述： 当被请求节点感知到电脑发来的请求后，会在相应的回调事件中对请求作出回应，也就是会将邻居表信息回传给电脑

要解决的问题： 上面提到的 Dissemination 会将被请求的 id 广播到整个网络中，所以当节点事件触发后会先进行判断，看请求的 id 是否是自己的，如果不是则不予理会，否则会进行后续的回应；因为邻居表不是简单的几个数字那么简单，而是对每个邻居节点都有相应记录信息的链表实现，所以我们显然很难一次就将整个邻居表回传给电脑，我们在这里采用的策略是设定定时器 Timer1 来进行邻居表的发送，定时器每次触发后都会发送一项邻居信息给电脑，直到邻居表信息发送完毕；为了准确的将邻居信息发送到电脑连接的特殊节点，我们要用到另一个由 TinyOS 实现的无线网络协议 Collection

备注： Collection 相关组件实现了无线网络协议 Collection，该协议的主要内容是，网络中所有节点发送的数据都会到达某个预设的特殊节点（又叫作根节点）

5. 特殊节点

功能概述： 上面屡次提到了特殊节点。在电脑发出邻居表查询请求时，与电脑进

行串口通信，接收处理电脑请求的节点被描述为特殊节点，该节点之所以特殊，在于当该节点接收到串口发来的数据包后，它会将该请求通过 Dissemination 广播给整个网络。在被请求节点打算回传邻居表给电脑时，利用到了 Collection 协议，该协议规定数据包会被根节点收集，也就是我们所说的特殊节点会接受来自被请求节点的数据包，显然该节点又必须是上面提到与电脑进行串口通信的节点，因为只有这样的话，在特殊节点收到发来的邻居表信息后才能顺利的回馈给电脑。至此应该有所明白所谓逻辑上划分的特殊节点，在物理上只需用到一个节点即可，就是与电脑连接的那个节点，只不过在程序会根据节点 id 来对特殊节点做额外的逻辑处理。

6. 电脑上的数据包监听程序

功能概述： 监听程序将会收到特殊节点发送来的邻居信息，并对邻居信息整理收集后显示在客户端

要解决的问题： 我们前面提到过邻居表信息被分成许多个数据包进行发送的，所以我们需要某种机制来判断当前发送来的数据包是否为最后一个邻居表信息项，为此我们将邻居信息数据包中添加了一个域用来标识当前数据包是否为最后一个数据包；默认情况监听程序只会输出数据包的二进制格式，为了直观的观察收集到的邻居表信息，需要使用 mig 将 NesC 中定义的数据包格式映射为相应的 java 类；此外还需在监听程序中使用链表来等待收集所有的邻居表信息，然后再予以显示。

邻居节点的评估：

样例： 为了便于理解邻居节点评估时将会涉及到哪些问题，我们这里用场景为例进行说明

1. 某个节点向周围节点广播一次数据包之后失去联系，显然在这种情况下该该节点不应该被周围节点判 断为邻居节点，我们将这种情况定义为评定第一准则： 节点必须在收到周围节点发送数据包次数超过限定次数的最低次后才能将周围节点评定为邻居节点。

2. 某个节点在一个小时内向周围共广播了 10 次数据包，对于网络通信而言，一个小时广播 10 次说明该网络的可靠性极低，我们将这种情况定义为评定第二准则：只有周围节点发送的数据包丢包率低于最高限定比例时才可将其评定为邻居节点。

3. 某个周围节点虽然已经被评定为邻居节点了，但是在接下来的一小时内却没能够再发送来数据包，我们将这种情况定义为评定的第三准则： 在超过限定时间未收到邻居节点发送来的数据包后，将该邻居节点从邻居表中移除。

实现：

周围节点和邻居节点

为了方便评估节点是否可以成为邻居节点，我们将在通信范围的节点称之为周围节点，

只有当周围节点在后续的评估中通过上面叙述的准则后，才会考虑将其升级为邻居节点。

为此我们在节点中维护了两个数据结构，分别用以记录周围节点集和邻居节点集，一个我们将其称之为 neighbor table（在下文中将会简写为 ntb），即用于记录当前节点的邻居节点集，另一个数据结构将其称之为 detected table（在下文中将会简写为 dtb），即用于记录当前节点的周围节点集，以上两个数据结构都使用链表实现，至于具体的链表节点定义如下：

```
typedef nx_struct NeighborNode {
    nx_uint16_t nodeid;
    nx_float reliability;
    nx_uint8_t deleted;
    nx_struct NeighborNode* next;
} NeighborNode;

typedef nx_struct DetectedNode {
    nx_uint16_t nodeid;
    nx_uint32_t ftime;
    nx_uint32_t ltime;
    nx_uint32_t times;
    nx_uint8_t isneighbor;
    ux_uint8_t deleted;
    nx_struct DetectedNode* next;
} DetectedNode;
```

值得注意的是， NeighborNode 中定义的 reliability（可靠性）， 可靠性为后续路由实现中选取最佳路径奠定了基础，是通过在一定时间段内的丢包率来评估的。DetectedNode 中定义了，第一次收到数据包的时间，最后一次收到数据包的时间，以及总共收到数据包的次数，这些都是为了后续评估邻居节点提供的信息。此外可以注意到上面的两个数据结构都设有 deleted 域，此举是为了实现懒惰删除，即通过在节点信息中添加一个额外值用于记录该节点是否已被移除，这样做是为了避免频繁添加移除节点带来的性能损耗，但同样也是基于在真实情况下节点可能由于通信的原因只是暂时失联，再次回来成为邻居的可能性较大。

评估函数：

关于函数 *canbeneighbor*

邻居评估公式： *(node->times > PACKET_LEAST_TIMES) && (node->times > (node->ltime – node->ftime) / (PACKET_LOST_RATE*SENDER_PERIOD_MILLI))*

其中 SENDER_PERIOD_MILLI=250ms 是节点发送数据包的周期，PACKET_LOST_RATE=2 表示允许的最大的丢包率（最多允许两个数据包丢失一个），PACKET_LEAST_TIMES=3 表示节点要被评估为邻居节点至少收到三个以上的数据包，

这三个宏都在头文件中进行了定义

总之，上面公式表示的意思是：如果已经收到某节点发来的三个以上的数据包并且该节点的数据包丢包率在 2 以下（也就是说满足了我们上面所提到的第一和第二准则），就认为该节点是邻居节点，至于对节点的第三准则进行评估涉及到邻居节点的移除问题，这牵涉到了对广播数据包的处理流程，下面我们将会进行针对性的解释

广播数据包的处理流程:

对广播数据包的处理是在 ActiveMessage 的 receive 事件中发生的，主要的数据包处理逻辑放在了私有函数 dispatch 中，该函数内部的主要处理逻辑是这样的，遍历 dtb，添加/更新节点信息，或者从 dtb 以及 ntb 中移除节点信息，也就是说该函数用于完成对 dtb 和 ntb 的创建和维护

添加节点到 **dtb**： dtb 表中没有收到数据包中相应 id 的节点信息时，在 dtb 中添加新的节点

更新节点： 更新 dtb 表中对应于收到数据包 id 的项，并且评估该节点是否可以成为邻居节点，如果可以则将其添加到邻居表中

移除节点（第三准则的评估）： 如果 dtb 表中记录的 last time 与当前时间的差值大于 ACTIVE_PERIOD_MILLI（等于 2000ms）的话，我们可以认为该节点已经不在是当前节点的邻居，所以从 dtb 和 ntb（如果是邻居的话）中移除节点，如上面提到的，对于移除节点我们这里做了一个优化，即惰性删除。

遇到的问题:

1. 内存动态分配

因为 TinyOS 是静态内存分配机制，也就是说不能使用 C 语言中的 malloc 类函数进行动态的内存分配，经过查找 TinyOS 文档发现 PoolC(typedef pool_t, uint8_t POOL_SIZE)符合预先分配内存的要求，该 component 可以预先分配一定数量的对象供程序使用，其实就是内存池的概念

2. 单跳通信

ActiveMessage 是 TinyOS 中的单跳无线通信协议，组件 ActiveMessageC 主要用于控制无线通信的 starting, stoping；组件 components new AMSenderC(AM_NEIGHBORTABLE);提供三个 interfaces：Packet, AMPacket, AMSend 用于数据包的发送；组件 components new AMReceiverC(AM_NEIGHBORTABLE);用于 ActiveMessage 数据包的接收；注意发送和接收参数是一致的，该参数定义在 NeighborTable.h 文件中

3. 如何从电脑上获取邻居表信息?
该问题又分为两个子问题：**1.** 节点如何知道电脑想要查询哪个 nodeid 的邻居表信息
2. 在节点知道电脑想要查询自己后，如何将邻居表信息发送给电脑

1. 该功能的实现依赖了 TinyOS 提供的网络协议 Dissemination，该协议可以保证某个节点发送的数据包能被该节点所在网络的所有节点收到，
下面简单介绍一下 DisseminationC 和 DisseminatorC 组件
DisseminationC 组件用于控制 dissemination 协议的开启和停止
而 DisseminatorC 组件则提供了两个接口： DisseminationValue 和 DisseminationUpdate，前者被消费者所使用，后者被生产者所使用，具体来说当生产者调用 DisseminationUpdate.change()后，对应的消费者的 DisseminationValue.changed()事件就会被调用，我们的打算是将 id 为 1 的节点设为与电脑通信的节点，当该节点收到电脑发来的请求 id 后，通过调用 DiseminationUpdate.change()来将被请求的 id 扩散出去，网络中的所有节点将会触发事件 DisseminationValue.changed()，在 changed()中的内部逻辑中将会判断被请求的 id 是否为自己，如果不是自己的会忽略该请求，但如果是自己的话，节点就要请求给予响应了，也就是要将自己的邻居表信息发送给电脑，也就是我们要解决的第二个问题了

2. 要解决这个问题就要使用 TinyOS 提供的另一个网络协议 Collection，该协议的特点是某个被设置为根节点的节点将会收到该网络中所有节点发来的数据包，也就是被请求节点可以通过 collection 来发送数据，自然我们会将 id 为 1 的节点设置为根节点，具体的操作是： 被请求邻居信息的节点将会在 DisseminationValue.changed()逻辑中开启一个定时器用于发送邻居信息（因为邻居信息较多不可能一次完成传送，所以通过一个定时器来发送邻居表），发送邻居信息主要用到了组件 CollectionSenderC，在该节点发送数据包后根节点将会触发 CollectionC.Receive 事件，我们前面说过邻居表信息将会由若干的数据包组成，也就是说在电脑需要等待所有的数据包传送完成后才可以显示邻居信息，为此我们将数据包的某个域做了特殊的标记，邻居表的最后一个数据包的标记跟之前的是不同的

4. 电脑与节点的通信以及电脑的 **java** 监听程序实现
参照 TestSerial 项目来完成的
1. 电脑将 request id 传送给节点后（Serial.receive 事件中），节点才调用的 DisseminationUpdate，我们又将该接口称为应用程序的 **Input**

2. 根节点在收到邻居信息数据包时会及时的将其传送给电脑，也就是在根节点的 Collect.receive 事件中调用 Serial.send 来发送数据包给电脑，我们又将该接口成为应用程序的 **Output**