计算机网络 WSN 实验

王思伦 2011013245
杨磊 2011013256
欧阳方昕 2011013250

多跳 WSN 数据采集

共有三个节点，其中 ID 为 0 的节点作为基站节点，烧制 NodeBase 程序；而 ID 为 1、2 的节点以一定的速率(50ms, 100ms, 500ms)采集温度、湿度和光照数据，烧制 NodeSense 程序，2 号节点需要通过 1 号节点转播给基站节点。

我们设计了一种丢包重传机制：如果接收节点发现自己数据包序列号不连续，即丢包了，则将丢包的序列号写入丢包队列中，然后不断发送重发请求(1 time /s)知道接收到该数据包，或者发送节点明确告知该数据包已不存在无法重传。这样可以将丢包率降低至近乎于零。

Telosb 使用的温湿度传感器组件为 SensirionSht11C，光照传感器组件为 HamamatsuS1087ParC。计算公式如下：

- 温度 T（摄氏度）=-39.6 + 0.01 SOT，其中 SOT 为原始数据的低 14 位
- 相对湿度 RH（%）=-2.0468 + 0.0367 SORH - 1.5955*10^-6 * SORH^2，其中 SORH 为原始数据的低 12 位。
- 照度 L（kLux）=0.085 SOL，其中 SOL 为原始数据。

在 NodeBase 程序中，基站节点与串口建立通讯，NodeSense.java 负责接收串口消息并将温湿度、光照输出到 result.txt 中；SenseInterval.java 可以给串口发送消息以控制采样频率。

多点协作 Data Aggregation

在多点协作实验中，需要收集齐 2000 个数据并计算最大值、最小值、平均值、总和、中位数，最简单粗暴的方法便是将 2000 个数据一并收集在一个节点中，直接计算并返回结果。但这种算法存在诸多问题：1. 一个节点存储 2000 个数据过于奢侈且仅有一个节点参与了计算，这不符合分布式计算的初衷。 2. 当数据量增大到 2500 以上时以上算法完全失效。 3. 单一节点计算效率太低。于是我们考虑过两种比较合理的算法：

在第一种比较简单的算法中，我们假设已知数据大致的分布范围[0-10000]，使三个节点分别接收处于[0-3500]（ID 为 92），[3500-6500]（ID 为 91），[6500-10000]（ID 为 93）范围中的数据，每个节点只需要存储自己负责的那一部分范围的数据。然后指定中间的节点为司令节点，始终等待接收其他两个节点发送的

数据包：92 号节点返回其所在区间内的最小值、数据量、数据总和；93 号节点返回其所在区间的最大值、数据量、数据总和。当司令节点收到 92、93 号节点返回的数据包并确认 2000 个数据收集全了以后，只需对自己区间[3500-6500]中的数据排序，并计算中位数、平均数等整合成最后结果返回。

但我们可以观察到，当数据的数值分布范围未知、或者分布不均匀时，这种算法又会退化为最开始那种简单暴力的原始算法，即，有可能所有的 2000 个数据都分布在[3500-6500]，这样时间效率和空间效率没有丝毫提升。

为此可以选择第二种更完善的算法：真正的分布式计算。在这里，我们制定 91 号节点仅仅接收 Sequence Number 被 3 除余 0 的数据包, 92 号节点仅仅接收 Sequence Number 被 3 除余 1 的数据包，93 号节点接收 Sequence Number 被 3 除余 2 的数据包。这样每个节点接收 1/3 的数据量，节省了空间复杂的。最大值最小值的计算方法也一目了然，但难点在于计算中位数：可以指定 91 号节点为司令节点，每一轮产生一个随机数，把三个数组分别切割成两段，然后我们知道中位数一定在小于 random X 的三段或者大于等于 random X 的三段中，保留三段，砍掉三段。第二轮继续产生随机数分割三个数组，以此类推不断逼近中位数的下界，直到刚好砍掉了 999 个数，则剩下的三个数组中最小的那个就是中位数！

经过一天半的尝试和设计，我们发现第二种算法仍然过于复杂，并且节点之间通讯过多反而降低了效率，增高了差错的可能性，于是最终选择了第一种算法。

但第一种算法需要解决一个很严重的问题：各节点如何判断自己接收到了属于自己范围的所有数据包而没有丢失？为了解决这个问题，只有一个办法：让每个节点都保证自己收到过 2000 个数据包并进行了区间筛选，只不过不在自己区间的包不必保留，而丢失的包虽然不确定是否属于自己的范围，仍然需要加入丢包队列请求其他两个节点重传或者下一轮广播重传。

我们为每一个节点构建两个丢包队列，第一个类似于一种 Hash 表，设其长度为 DROP_QUEUE_LEN，若 Sequence Number 为 N 的数据包丢失，则置丢包队列的第 N%DROP_QUEUE_LEN 个元素为 N，这类似于一种 Hash 映射，如果该位置被其他余数相同的元素占据，则加入第二个丢包队列：第二个丢包队列类似于链表，直接添加到链表末尾即可。在实际试验中，若丢包队列溢出则程序会崩溃，所以需要根据丢包率和数据量设置合理的丢包队列长度，我们设为 600.

经验与教训

现场演示的时候出了很多 bug, 导致我们仅仅是倒数第二个完成任务的小组，明明花费了很长时间设计精密的算法并投入了大量的时间却落得如此失望的结果，反思一下有如下几点教训：

1. 简单暴力的方法对于期末紧张的同学来说才是上上策，如果你真的对自己很有把握或者在计算机网络方面态度端正并情有独钟，我们才建议你查阅网上资料、使用完美的分布式计算。

2. 在测试之前一定要确保自己试验仪器完好，我们组到了现场发现司令节点总是收不齐，原来是节点坏掉了。
3. 与其他组多交流、与 TA 多多沟通，我们组自作聪明的在 Makefile 里设置了 Channel 想屏蔽其他组的干扰，结果比赛开始后根本收不到广播节点的消息。
4. 编程细心、耐心，注意边界处理、下标计算，逻辑复杂则往往漏洞百出，我们小组一直调到比赛开始前半个小时。

这一次大作业，我们懂得：不是所有的努力都有回报，不是所有精美的算法设计都适合你，但就如果你选择了一条正统、精彩却荆棘丛生的道路，就算知道会比其他组投入时间多、收获成果少，我们也要坚持自己的选择，努力到最后一刻。感谢 TA@党主席一个学期的耐心帮助。