**网络实验报告 exp8**

**2020K8009929032 刘耕印**

注：本次实验的规模比较大，内容比较繁杂，因此在报告中我准备只贴上关键部位的少部分代码。又因为我自己创建了一些数据结构和算法，从实验报告中可能无法看到其全貌，具体内容请参考我的提交。其中，添加的数据结构和算法主要是用来根据数据库生成路由表的，主要分布在文件rtable.h和rtable.c中。

1. **实验内容**

本实验实现路由表的动态学习。内部网关协议采用简化的OSPF协议，包含hello和lsu两种数据包和一致性链路数据库，并使用dijkstra算法生成最小路由。

1. **实验准备**

实验开始前，先学习协议概要，理清实现思路。学习得知，mOSPF协议运行在每个路由器节点上，通过hello包发现自己的邻居节点，通过lsu包获知其余节点所涉及的各个网段和邻居。根据讲义和代码框架的提示，将额外创建四个线程来进行hello和lsu的交流。四个线程分别是：周期性发送hello，周期性发送lsu，邻居列表老化，以及数据库的老化。 一旦本节点所知的拓扑发生改变，就有必要对本节点的数据库进行更新；一旦本节点的数据库有更新，就应当重新构建路由表。拓扑发生改变有两种情况，即邻居列表更新和数据库更新（我没有在数据库中放入对应本节点的项目，在生成路由表时本节点邻居的参考是通过遍历所有端口完成的）。同时，在本地拓扑有变化时，又应当发送lsu包进行通知。根据这些思考，把数据库更新、lsu生成和广播以及重建路由表这三项工作作为关注重点，分别进行实现，封装后在各个相关线程中按需调用。

上面提到的三项工作中，数据库更新和lsu广播是各节点一致性数据库维护的基础，而重建路由表则是依赖于数据库，对数据库没有影响。因此，先实现前两项，确认数据库建立无误后，再单独实现第三项。

最后，实现路由表的重建。这将是一项稍显复杂的工作，但是无论如何是以dijkstra算法为核心的，因此思路比较清晰。细节在本报告的第三部分进行介绍。

在实现的过程中，时刻要注意数据是否有互斥访问的可能性。除了框架代码建议的用锁保护数据库和邻居列表之外，由于主线程在发包时会查询路由表，而路由表又随时可能重建，我还需要设立一个锁用于保护路由表的互斥访问。然而在重建路由表的过程中，又需要用到数据库和邻居列表的信息，为避免死锁，这一过程需要特殊处理。细节也放在后面介绍。

1. **代码实现**（内容较多，所以按实现顺序依次说明）
2. 实现上文提到的四个交流用线程。四个线程的代码均在mospf\_daemon.c中。另外，四个线程均需要用锁保护，以实现对邻居列表和数据库的互斥访问。
   1. send\_hello

周期性广播hello包，向邻居宣告自己的存在。OSPF是建立在ip协议之上的，所以hello包和lsu包都需要ether头和ip头。hello包的目的硬件地址和目的ip地址是特殊的，且不需要被转发。根据协议规定的格式，此线程的每个端口生成hello包并发送出去。具体来说只要把几乎固定的信息填入包中即可，代码就不贴了。

* 1. send\_lsu

周期性广播lsu包，描述自己的全部邻居。按格式依次产生ospf头、lsu头和lsu体。lsu体是一个某种数据结构的数组，用lsa表示，每个lsa描述本节点的一个端口的一个邻居。其中有该邻居的id，该邻居所在网段和这个网段的mask。另外，如果本节点某个端口没有邻居，也要为它创建一个lsa，id设为0，但网段和mask设为这个端口的属性。这样一来，其他节点知道有这个网段的存在，才能正确地产生路由表。代码如下：

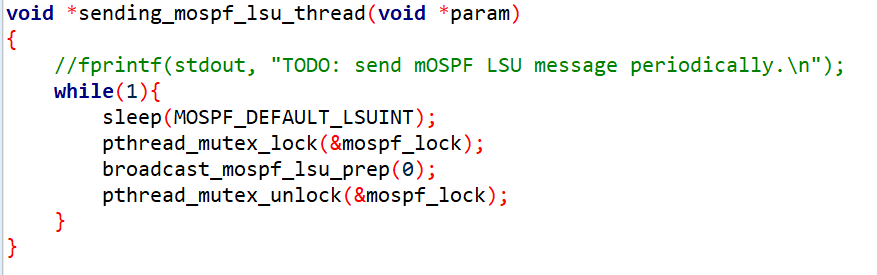


Figure 1 周期性广播lsu

我把broadcast\_prep函数实现在mospf\_proto.c内，其职责是产生本节点的lsu消息，若有需要的话更新数据库，并调用同样在该文件内实现的broadcast函数将其散播出去：

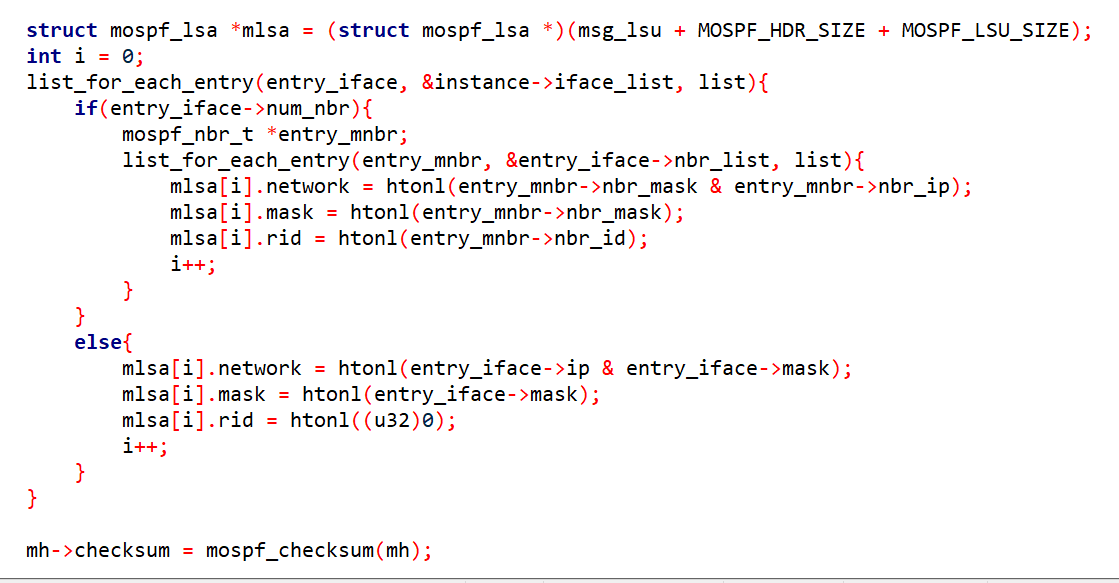


Figure 2 节选lsu信息的lsa部分生成

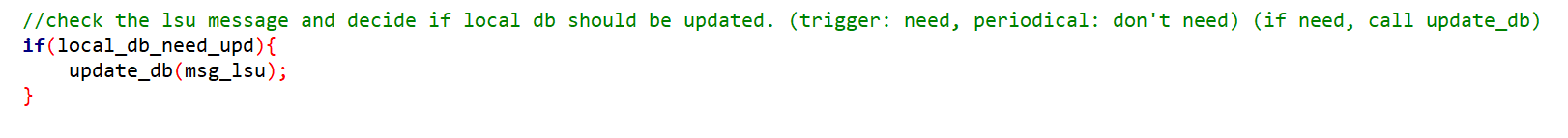


Figure 3 更新数据库（在周期性散播lsu时，由于这并不是由拓扑改变引起的广播，不更新数据库）

再后边就是广播lsu。这个广播是向所有已知的邻居广播。但在实现上，考虑到转发lsu时也可能用到这个函数，转发则不需要向收包的邻居发送，也不用把包释放掉，广播函数加了一些用于控制的参数。篇幅限制，代码略去。

* 1. check\_nbr

老化邻居列表。若发现有邻居已经失效，则将该邻居删除。删除后，由于拓扑已经改变，需要更新数据库，并根据更新后的拓扑生成lsu并广播。同时，更新数据库后还要重新构建路由表。其中，更新数据库和广播lsu通过调用之前提到的broadcast\_prep函数完成，重建路由表通过调用regenerate\_rtable完成。后边会提到这个函数。老化只需叠加nbr的alive属性并一直检查即可。这段代码也不贴了。

* 1. check\_db

老化数据库。老化方式依然是不断叠加db\_entry的alive并检查。如果发现有条目失效，视为一次数据库更新，需要重新生成路由表。但是，数据库表项和节点实际上是一一对应的（除了本节点），数据库失效不视为本节点拓扑的改变，不需要广播lsu。重建路由表还是调用regenerate\_rtable。这段也不贴了。

另外，在拓扑不变时，邻居失效和数据库失效理论上是不会发生的。

1. 处理送来的mospf包

为了构建完整的一致性链路数据库，需要对收到的mospf hello包和lsu包进行正确处理。

* 1. 处理hello包

hello包用于告知邻居本节点的存在。因此，收到邻居发来的hello包后，就应当把该邻居加入（收包端口的）邻居列表。这个邻居可能是已知的，那么只要把对应的alive清零即可：

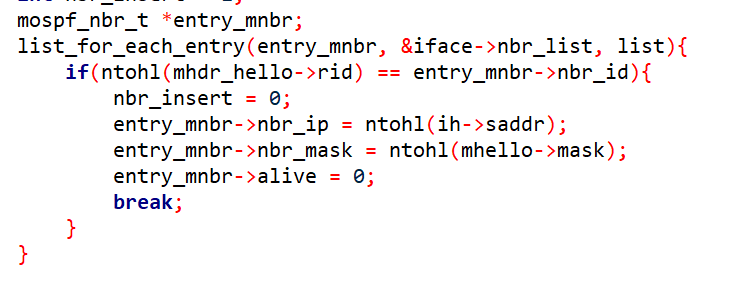


Figure 4 更新邻居生命期

若邻居是未知的，需要把该邻居加入到此端口的邻居列表中：

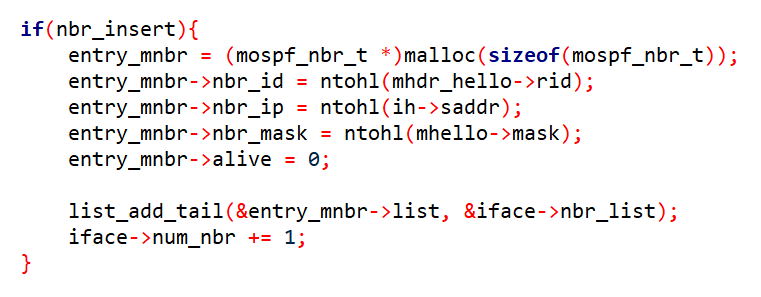


Figure 5 新增一个邻居

在有新增邻居的情况下，视为拓扑改变，因此还需要【更新数据库，广播lsu，重建路由表】。依然调用上边提到的函数：

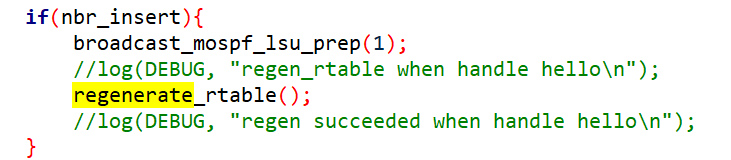


Figure 6 三板斧

* 1. 处理lsu包

收到的lsu是源自其他节点的，描述的是该节点的所有邻居。收到lsu后，我们应该转换它的形式，存进数据库里：



Figure 7 依据lsu消息更新数据库

数据库中可能已经存储了对应的条目，此时比较seq，若新来的lsu较大则依然设置lsu，否则直接结束。这里提一下，update\_db被我实现在mospf\_database.c中，功能如其名。

然后还是更新路由表。最后，收到的lsu要进行转发，以收到信息的端口和收到信息的iphdr为参数调用broadcast函数即可。

**最后要提一点：为了避免次数过多的无意义循环（count\_to\_infinity问题），我的处理方法是不转发id和本节点相同的lsu。但我也不是很确定这样行不行，理论课介绍了其他的方法。**

****

Figure 8 避免循环问题

以上的设计有一个点不太符合要求：即使因为拓扑改变导致了节点主动广播lsu，也不会重置周期性广播lsu的冷却时间。一种可能的解决办法是在send\_lsu线程中使用pthread\_cond\_timedwait，由触发拓扑改变的线程来主动唤醒此线程。

1. 路由表重建

在以上两部分完成后，各个路由器开始交换信息，收敛时已经建立起一致性链路数据库。接下来，考虑在数据库有所变化时重建路由表，以实时维护最新的路由表。OSPF协议使用dijkstra算法，以本节点为源点，计算出到所有其他网段的最短路径。由于我建立数据库时没有固定加入本节点的条目，因此涉及本节点的信息时，需要遍历本节点的端口作为代替。

首先，将数据库抽象成图拓扑。我采取最简单的图存储方法，即邻接矩阵。由于所有线路都是双工的，因此矩阵会是一个对称阵。所有链路开销均视为1。抽象过程中使用了一些辅助的数据结构，其建立不赘述了。

先给每个节点一个编号，下标为i的节点其rid为rids[i]，0号固定为本节点；

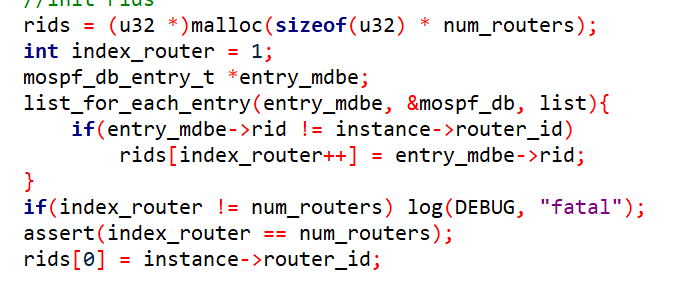


Figure 9 给每个节点一个编号

然后初始化邻接矩阵，各链路均视为不可达：

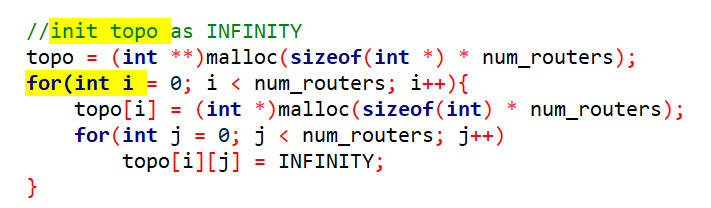


Figure 10 初始化图

接下来依据数据库填充邻接矩阵。先处理除本节点外的节点，它们有对应的数据库条目：

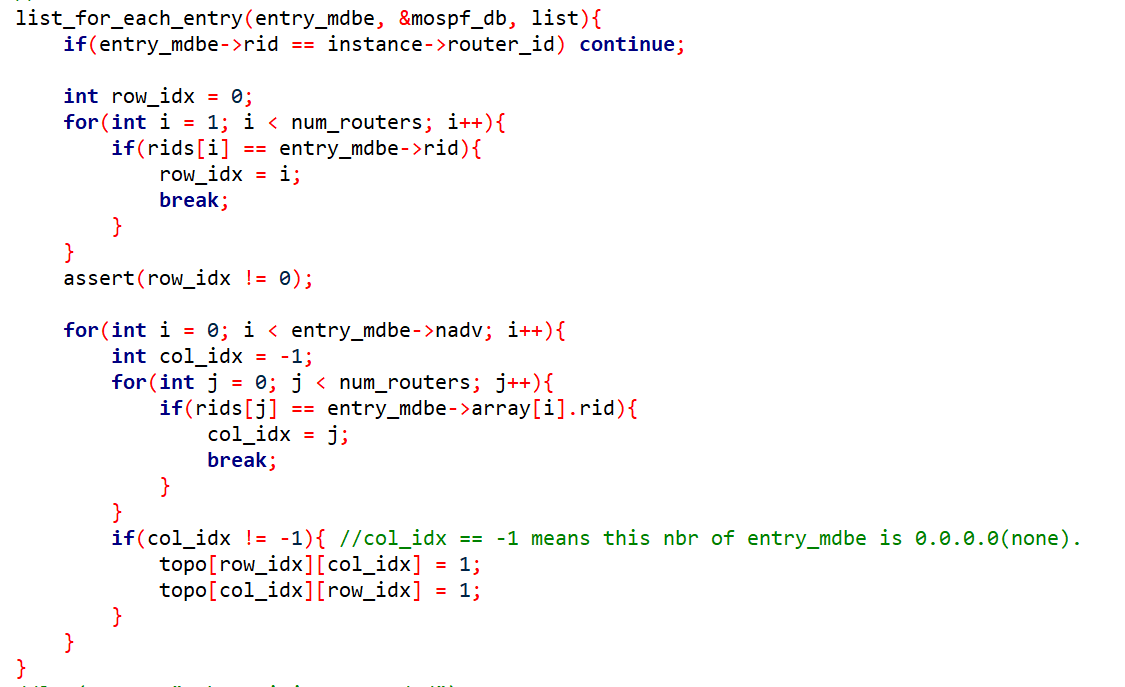


Figure 11 初始化其余节点的行

这里其实不应该把对称的元素也同时赋值，但我不这么做就会出问题，因为写第一版代码的时候没有考虑到数据库收敛前的情况。后来修改时发现似乎只能这样做。

最后依据本节点的邻居情况填充本节点那一行：

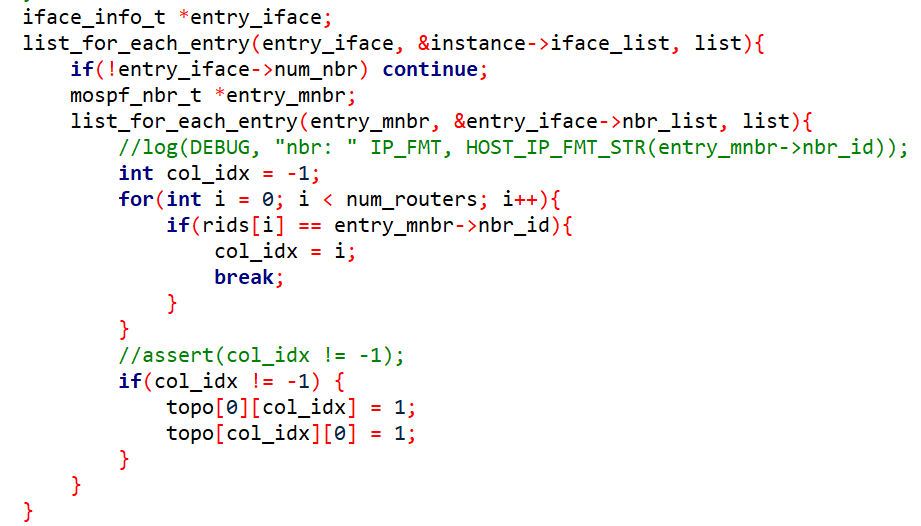


Figure 12 填充本节点的行

至此图拓扑建立成功。

然后，在建好的图拓扑上运行dijkstra算法，计算出到其余每个节点的最短路径和对应的前一跳节点。按照课件的思路实现即可，但有两点问题应该指出：

1. 课件的min\_dist函数应当返回的是当前dist最小值的下标，即当前离源点最近的节点，而不是离S集中节点最近的节点；
2. 课件将prev数组初始化为-1，这是不对的，因为一些和源点直连的节点可能因为从未满足更新条件而一直不更新。因此，应当把prev数组初始化为源点下标，即0。

实现如下：

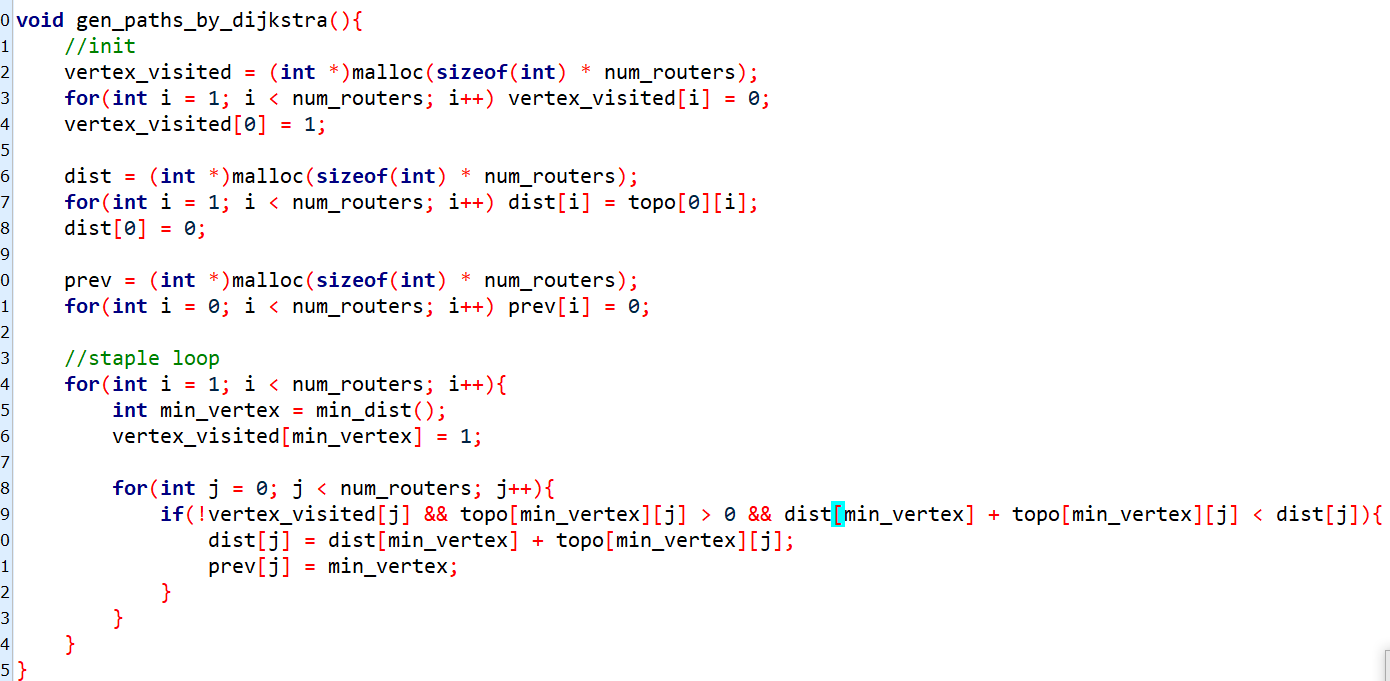


Figure 13 运行dijkstra算法获取最短路径

至此，已经获得了所有最短路径和所有前一跳信息。最后一步，用这些信息建立路由表。这又是一段很长的代码，但思路基本和课件是一致的，按开销由小到大依次处理每个节点的每个网段，递归地找到到该节点的第一跳，将对应网络、掩码、网关和端口合成一个路由表项。值得一提的是，以上的过程中要用到数据库和邻居信息，因此在查询时要用mospf锁保护。而操作路由表时又要以rtable锁保护。为了避免死锁，我不会在遍历的过程中同步生成路由表项，而是先把产生那些表项需要的信息保存起来，释放掉mospf锁后再申请rtable锁，利用保存的信息建立路由表。如下：

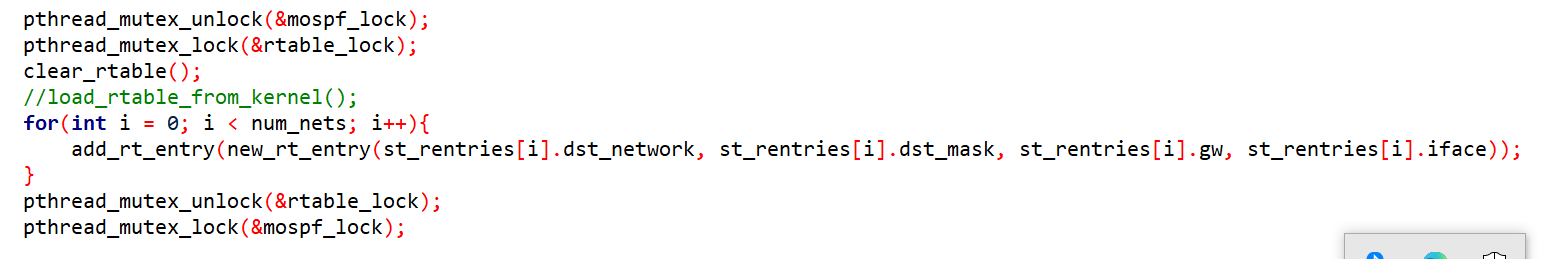


Figure 14 避免死锁

st\_entries存储着所有待加入的路由表项，同时也包含网关为0的那些表项。

regenerate\_rtable函数是将以上过程组合起来。

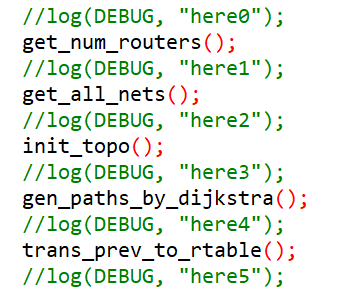


Figure 15 regenerate\_rtable函数

至此代码编写完成。

1. **实验结果**

首先展示一下数据库的生成，以r1为例，稳定时数据库如下：

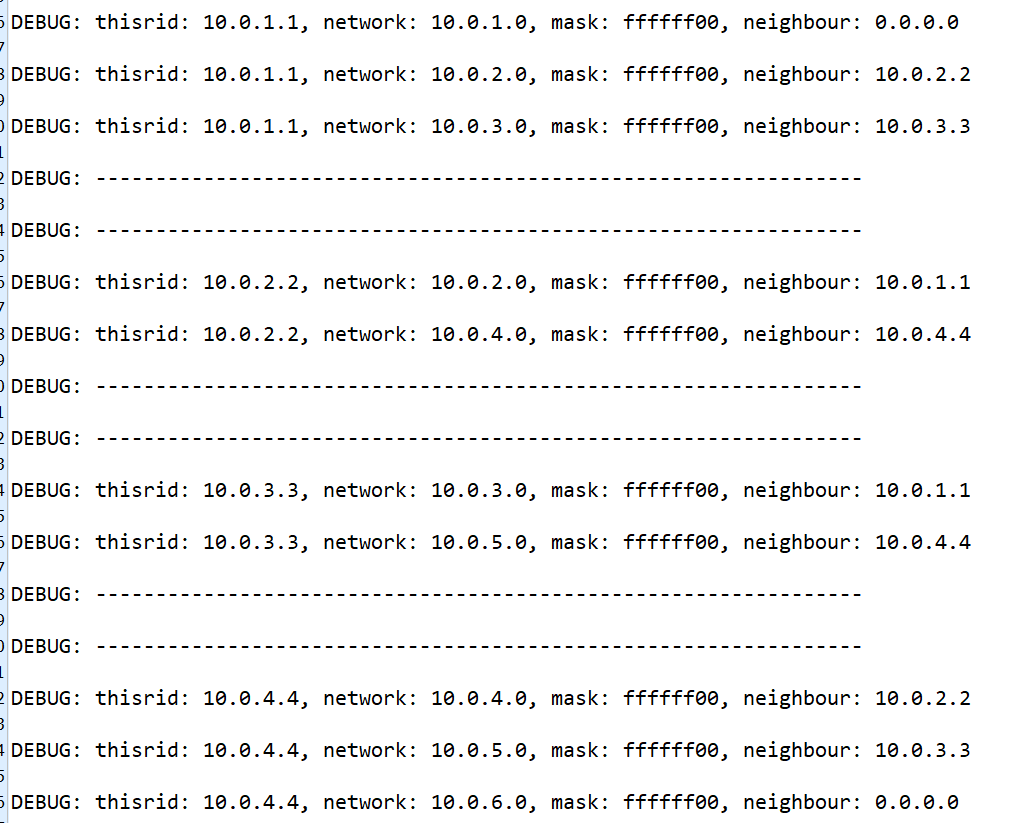


Figure 16 database of r1

其中有本节点的条目，过一段时间会被清扫掉。不用在意，重建路由表时用不到它。

与拓扑对照：

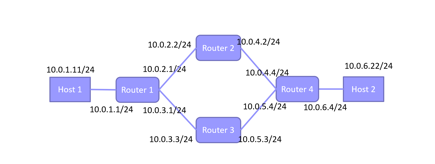


Figure 17 拓扑

结果是正确的。

然后，再展示一下稳定时的路由表：

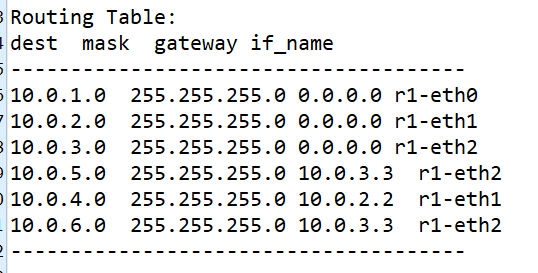


Figure 18 r1的路由表

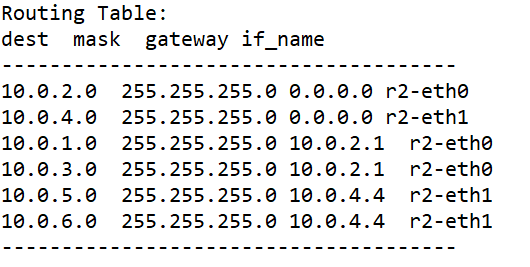


Figure 19 r2的路由表

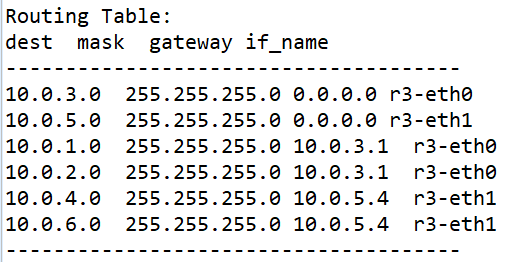


Figure 20 r3的路由表

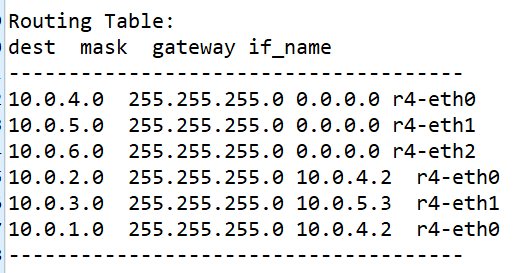


Figure 21 r4的路由表

符合预期。

然后，用traceroute进行连通测试以及变化测试：

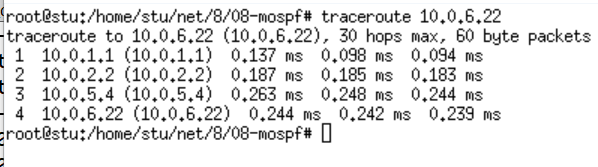


Figure 22 直接traceroute



Figure 23 down掉之前使用的链路

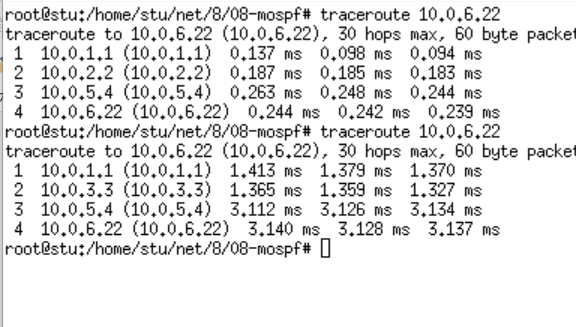


Figure 24 再traceroute

down掉之前使用的链路后，重新建立路由表，选取了另一条链路。至此功能看起来是正常的。

但在提交oj时出了点问题，我的判题一直卡住，和助教交流后也没有什么头绪。目前还在等待结果。



Figure 25 昨天交的代码今天还在卡着

1. **调研题目**

**1. 在构建一致性链路状态数据库中，为什么邻居发现使用组播(Multicast)机制，链路状态扩散用单播(Unicast)机制？**

邻居发现使用组播的好处在于当拓扑比较复杂时，组播可以向一个组只发送一条信息，开销比较小，可以高效地发现邻居节点。但是，组播方式难以支持确认报文（虽然mOSPF没有确认步骤），在可靠性上不如单播，于是对精度要求较高的lsu扩散使用单播。

**2． 该实验的路由收敛时间大约为20-30秒，网络规模增大时收敛时间会进一步增加，如何改进路由算法的可扩展性？**

下面是理论课的课件：

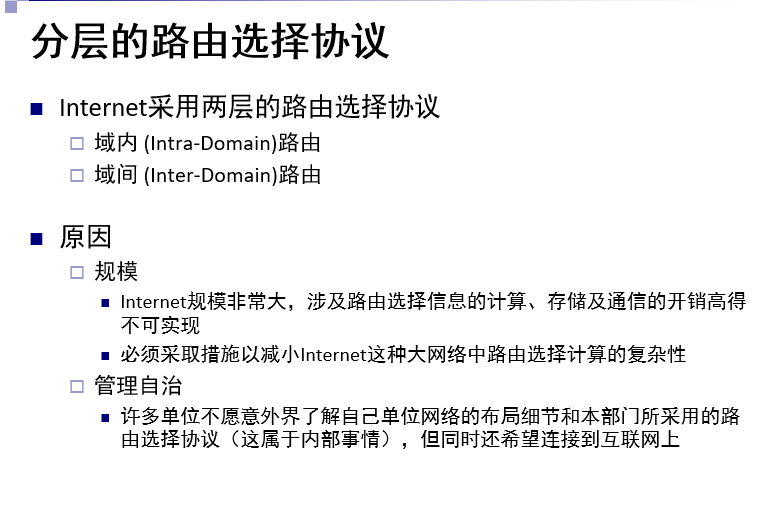


Figure 26 理论课课件

OSPF是一种域内协议，只在某个Autonomous System内起作用。于是，可以限制其规模。

**3. 路由查找的时间尺度为~ns，路由更新的时间尺度为~10s，如何设计路由查找更新数据结构，使得更新对查找的影响尽可能小？**

用前缀树组织路由表，可以对每位前缀都设置锁，于是更新某前缀下的路由表时，可以只锁住路由表的该部分。此时如果路由查找不涉及该部分，就可以查找和更新同时执行。另外，下面这篇文献还给出了优化更新范围的方法：[1]苏恭超, 李乐民. 高速IP路由查找中改善更新性能的方法[J]. 通信技术, 2003(4):3.

文献随报告一起提交。