Programming Assignment: RIP Router Algorithm

南京大学 电子科学与工程学院 吴康正 171180571

实验环境

Microsoft Windows 10 Version 2004
Microsoft Hyper-V Manager Version 10.0.19041.1
Ubuntu 16.04 xenial w/ x86_64 Linux 4.4.0-142-generic

文件结构

```
├── Makefile
├── obj
├── report
│   ├── out.txt
│   ├── report.md
│   └── report.pdf
├── prog3
└── src
    ├── config.c
    ├── node0.c
    ├── node1.c
    ├── node2.c
    ├── node3.c
    ├── node.c
    ├── node.h
    └── prog3.c
```

实验包含了数个目录，其中src目录存放源码，obj目录存放编译时产生的目标文件，report目录存放实验报告和终端输出结果out.txt，提供Makefile可以用于快速编译和清理。
node.c与node.h包含了所有节点共有的函数和定义，config.c文件包含了每个节点的初始化参数，如果需要修改节点关系和初始代价，可以在config.c文件中完成。

实现方式

> 对提供的prog3.c文件进行了两处修改：
> 增加了对标准库头文件的包含，同时修改了exit()函数调用增加了一个参数，因为我的编译器提示使用的exit()函数没有声明，而实验也没有要求我来实现这个函数，因此做出了此处修改，**这不会影响程序的执行。**
> 在程序的printf语句内容末尾增加了一个换行符，在开启了bonus部分后程序对链路代价事件的输出不会换行，导致输出界面较乱，为了美观考虑做出了此处修改，**这会导**

> **致部分程序输出排版由于增加的换行符而与原先不一样，但是对程序的运行没有任何影响。**

代码使用typedef的方式定义了每个节点的模版，节点的功能使用node.c文件中的函数具体实现，每一个节点的函数会直接调用node.c中的函数完成功能。

节点结构如下：

```
typedef struct
{
    int id;
    struct distance_table dt;
    int mincost[4];
    int neighbour[4];
} node_class;
```

每个节点的对象包含了节点id，节点的距离表、当前到每个节点的最小代价和当前节点到其他节点的相邻关系。节点id和相邻关系在节点初始化时写入节点，其中相邻关系为一个数组，如果当前节点与节点i相邻，则数组中对应的i为1，否则为0，特别的，为了提升泛用性考虑，节点与自身身的相邻关系也为0。

初始化函数node_constructor()与rtinit()

每个节点的文件定义了一个节点对象Nodex，在每个节点的初始化过程中，先后调用node_constructor()函数rtinit()函数对当前节点对象进行初始化，函数将所有代价初始化为999，并且将初始代价填入对应的位置。
完成对象的构建后，函数调用updatemincost()函数计算当前到每个节点的最小代价填入对象，并通过sendcost()函数发送给每个相邻的节点。

核心函数

最小代价计算函数updatemincost()

对于每一个节点，函数提取代价表中途径各个节点到此节点的代价进行比较，并将最小值记入newmincost数组对应的项中，然后将此数组与节点此时记录的mincost进行对比，如果代价有所变化，则将updated置为1并返回，否则返回updated为0。

代价发送函数sendcost()

按照当前节点记录的mincost和节点id构建packet进行发送，发送时，函数检查当前节点的邻居表，从而实现只对相邻节点发送cost包。

代价接收函数rtupdate()

收到从底层的消息后，节点首先判断信息是否是发给自己的（虽然在这没有必要），如果确定节点正确，就根据packet中提供的新数据更新代价表中对应节点的数据，并根据数据是否改变打印新的代价表。

代价表更新完成后，调用updatemincost()函数更新节点对象存储的到每个节点的最小代价，如果函数返回值为1，即最小代价有变化，则调用sendcost()函数发送新的mincost给其他节点。

代价表输出函数printdt()

输出函数写的比较花哨，加入了大量的判断，主要是为了提升函数的泛用性，可以直接由每个节点的邻居信息生成正确的表格格式，同时在节点数量不是4的情况下也可以正常工作。

接口函数

接口函数即为测试函数会调用的函数，如rtupdate0()等。
由于采用了对象的方式，因此接口函数的篇幅不大，直接调用对应对象函数进行初始化、处理等操作。

```
void rtupdate0(struct rtpkt* rcvdpkt)
{
    rtupdate(&Node0, rcvdpkt);
}
```

Bonus实现

使用linkhandler()函数实现每个节点遇到链路代价变化时的处理，函数由接口函数直接调用实现功能。此函数具有良好的泛用性，理论上，除了节点0和节点1，其他的节点（包括但不限于节点2与节点3）也可以直接调用此函数进行对应的链路代价处理。
当探测到链路代价变化后，函数首先根据邻居表neighbour判断涉及的链路是否真实存在，如果传入的linkid来自一个不应该与此节点直接相连的节点，则认为链路信息无效直接退出，否则信息有效，继续处理。程序从代价表取出链路原先的代价，然后对代价表中途径该节点的所有代价进行重新计算，即减去原先代价并加上新的代价，然后打印新的代价表。
代价表更新完成后，调用updatemincost()函数更新节点对象存储的到每个节点的最小代价，如果函数返回值为1，即最小代价有变化，则调用sendcost()函数发送新的mincost给其他节点。

实现效果

实验结果篇幅过长，因此不在此处全文给出，您可以在report文件夹下的out.txt中找到完整输出，也可以直接运行测试程序。
out.txt文件使用了符合ANSI标准的文本高亮控制码，使用cat命令重定向到终端模拟器就可以实现高亮显示。

```
minaduki@mininet-vm:~/computer_networking/assignments/RIPRouterAlgorithm
Enter TRACE:2
Time = 0.000. rtinit0() has been called.
Time = 0.000. Node 0 sent packet to Node 1 : 0 1 3 7
Time = 0.000. Node 0 sent packet to Node 2 : 0 1 3 7
Time = 0.000. Node 0 sent packet to Node 3 : 0 1 3 7
Time = 0.000. rtinit1() has been called.
```

```
Time = 0.000. rtinit1() has been called.
Time = 0.000. Node 1 sent packet to Node 0 : 1 0 1 999
...
...
MAIN: rcv event, t=0.947, at 3
 src: 0, dest: 3, contents:   0   1   3   7
Time = 0.947. rtupdate3() has been called.
Time = 0.947. Node 3 received packet from Node 0.
Time = 0.947. Distance Table for Node 3 has been modified!
  dest|     via
   D3 |    0    2
  ----|----------
     0|    7  999
     1|    8  999
     2|   10    2
Time = 0.947. Node 3 sent packet to Node 0 : 7 8 2 0
Time = 0.947. Node 3 sent packet to Node 2 : 7 8 2 0
...
...
MAIN: rcv event, t=7.579, at 3
 src: 0, dest: 3, contents:   0   1   2   4
Time = 7.579. rtupdate3() has been called.
Time = 7.579. Node 3 received packet from Node 0.
Time = 7.579. Distance Table for Node 3 has been modified!
  dest|     via
   D3 |    0    2
  ----|----------
     0|    7    4
     1|    8    3
     2|    9    2
MAIN: rcv event, t=7.941, at 1
 src: 0, dest: 1, contents:   0   1   2   4
Time = 7.941. rtupdate1() has been called.
Time = 7.941. Node 1 received packet from Node 0.
Time = 7.941. Distance Table for Node 1 has been modified!
  dest|     via
   D1 |    0    2
  ----|----------
     0|    1    3
     2|    3    1
     3|    5    3
MAIN: rcv event, t=8.086, at 0
 src: 3, dest: 0, contents:   4   3   2   0
Time = 8.086. rtupdate0() has been called.
Time = 8.086. Node 0 received packet from Node 3.
Time = 8.086. Distance Table for Node 0 has been modified!
  dest|     via
   D0 |    1    2    3
  ----|---------------
     1|    1    4   10
     2|    2    3    9
     3|    4    5    7
...
...
MAIN: rcv event, t=9.960, at 2
 src: 0, dest: 2, contents:   0   1   2   4
```

```
Time = 9.960. rtupdate2() has been called.
Time = 9.960. Node 2 received packet from Node 0.
Time = 9.960. Distance Table for Node 2 has been modified!
  dest|     via
   D2 |    0    1    3
  ----|---------------
     0|    3    2    6
     1|    4    1    5
     3|    7    4    2
MAIN: rcv event, t=10000.000, at -1
Time = 10000.000. linkhandler0() has been called.
Time = 10000.000. Node 0 detected the new cost of link to Node 1 is now
Time = 10000.000. Distance Table for Node 0 has been modified!
  dest|     via
   D0 |    1    2    3
  ----|---------------
     1|   20    4   10
     2|   21    3    9
     3|   23    5    7
Time = 10000.000. Node 0 sent packet to Node 1 : 0 4 3 5
Time = 10000.000. Node 0 sent packet to Node 2 : 0 4 3 5
Time = 10000.000. Node 0 sent packet to Node 3 : 0 4 3 5
Time = 10000.000. linkhandler1() has been called.
Time = 10000.000. Node 1 detected the new cost of link to Node 0 is now
Time = 10000.000. Distance Table for Node 1 has been modified!
  dest|     via
   D1 |    0    2
  ----|----------
     0|   20    3
     2|   22    1
     3|   24    3
Time = 10000.000. Node 1 sent packet to Node 0 : 3 0 1 3
Time = 10000.000. Node 1 sent packet to Node 2 : 3 0 1 3
...
...
MAIN: rcv event, t=10004.307, at 1
 src: 2, dest: 1, contents:   3   1   0   2
Time = 10004.307. rtupdate1() has been called.
Time = 10004.307. Node 1 received packet from Node 2.
Time = 10004.307. Distance Table for Node 1 has been modified!
  dest|     via
   D1 |    0    2
  ----|----------
     0|   20    4
     2|   23    1
     3|   25    3
Time = 10004.307. Node 1 sent packet to Node 0 : 4 0 1 3
Time = 10004.307. Node 1 sent packet to Node 2 : 4 0 1 3
MAIN: rcv event, t=10004.417, at 3
 src: 2, dest: 3, contents:   3   1   0   2
Time = 10004.417. rtupdate3() has been called.
Time = 10004.417. Node 3 received packet from Node 2.
Time = 10004.417. Distance Table for Node 3 has been modified!
  dest|     via
   D3 |    0    2
```

```
  ----|----------
     0|    7    5
     1|   11    3
     2|   10    2
Time = 10004.417. Node 3 sent packet to Node 0 : 5 3 2 0
Time = 10004.417. Node 3 sent packet to Node 2 : 5 3 2 0
MAIN: rcv event, t=10004.669, at 0
 src: 1, dest: 0, contents:   4   0   1   3
Time = 10004.669. rtupdate0() has been called.
Time = 10004.669. Node 0 received packet from Node 1.
Time = 10004.669. Distance Table for Node 0 has been modified!
  dest|     via
   D0 |    1    2    3
  ----|---------------
     1|   20    4   10
     2|   21    3    9
     3|   23    5    7
MAIN: rcv event, t=10005.498, at 2
 src: 1, dest: 2, contents:   4   0   1   3
Time = 10005.498. rtupdate2() has been called.
Time = 10005.498. Node 2 received packet from Node 1.
Time = 10005.498. Distance Table for Node 2 has been modified!
  dest|     via
   D2 |    0    1    3
  ----|---------------
     0|    3    5    6
     1|    7    1    5
     3|    8    4    2
MAIN: rcv event, t=10005.692, at 2
 src: 3, dest: 2, contents:   5   3   2   0
Time = 10005.692. rtupdate2() has been called.
Time = 10005.692. Node 2 received packet from Node 3.
Time = 10005.692. Distance Table for Node 2 has been modified!
  dest|     via
   D2 |    0    1    3
  ----|---------------
     0|    3    5    7
     1|    7    1    5
     3|    8    4    2
MAIN: rcv event, t=10006.615, at 0
 src: 3, dest: 0, contents:   5   3   2   0
Time = 10006.615. rtupdate0() has been called.
Time = 10006.615. Node 0 received packet from Node 3.
Time = 10006.615. Distance Table for Node 0 has been modified!
  dest|     via
   D0 |    1    2    3
  ----|---------------
     1|   20    4   10
     2|   21    3    9
     3|   23    5    7
MAIN: rcv event, t=20000.000, at 0
Time = 20000.000. linkhandler0() has been called.
Time = 20000.000. Node 0 detected the new cost of link to Node 1 is now
Time = 20000.000. Distance Table for Node 0 has been modified!
  dest|     via
```

```
   D0 |    1    2    3
  ----|---------------
     1|    1    4   10
     2|    2    3    9
     3|    4    5    7
Time = 20000.000. Node 0 sent packet to Node 1 : 0 1 2 4
Time = 20000.000. Node 0 sent packet to Node 2 : 0 1 2 4
Time = 20000.000. Node 0 sent packet to Node 3 : 0 1 2 4
Time = 20000.000. linkhandler1() has been called.
Time = 20000.000. Node 1 detected the new cost of link to Node 0 is now
Time = 20000.000. Distance Table for Node 1 has been modified!
  dest|     via
   D1 |    0    2
  ----|----------
     0|    1    4
     2|    4    1
     3|    6    3
Time = 20000.000. Node 1 sent packet to Node 0 : 1 0 1 3
Time = 20000.000. Node 1 sent packet to Node 2 : 1 0 1 3
...
...
MAIN: rcv event, t=20001.125, at 3
 src: 2, dest: 3, contents:   2   1   0   2
Time = 20001.125. rtupdate3() has been called.
Time = 20001.125. Node 3 received packet from Node 2.
Time = 20001.125. Distance Table for Node 3 has been modified!
  dest|     via
   D3 |    0    2
  ----|----------
     0|    7    4
     1|    8    3
     2|    9    2
Time = 20001.125. Node 3 sent packet to Node 0 : 4 3 2 0
Time = 20001.125. Node 3 sent packet to Node 2 : 4 3 2 0
...
...
MAIN: rcv event, t=20002.221, at 0
 src: 3, dest: 0, contents:   4   3   2   0
Time = 20002.221. rtupdate0() has been called.
Time = 20002.221. Node 0 received packet from Node 3.
Time = 20002.221. Distance Table for Node 0 has been modified!
  dest|     via
   D0 |    1    2    3
  ----|---------------
     1|    1    4   10
     2|    2    3    9
     3|    4    5    7
MAIN: rcv event, t=20002.854, at 2
 src: 3, dest: 2, contents:   4   3   2   0
Time = 20002.854. rtupdate2() has been called.
Time = 20002.854. Node 2 received packet from Node 3.
Time = 20002.854. Distance Table for Node 2 has been modified!
  dest|     via
   D2 |    0    1    3
  ----|---------------
```

```
      0|    3    2    6
      1|    4    1    5
      3|    7    4    2
MAIN: rcv event, t=20002.979, at 1
 src: 2, dest: 1, contents:   2   1   0   2
Time = 20002.979. rtupdate1() has been called.
Time = 20002.979. Node 1 received packet from Node 2.
Time = 20002.979. Distance Table for Node 1 has been modified!
   dest|     via
    D1 |    0    2
   ----|----------
      0|    1    3
      2|    3    1
      3|    5    3

Simulator terminated at t=20002.978516, no packets in medium
```

小结

本实验实现了一个简单的路由协议，不过实验整体更倾向于验证，可拓展性和部署可行性不是很强，主要是受限于测试代码中写死了节点的个数，导致对节点数量的拓展性较弱。

如果涉及到节点数量的可变性，我想到了两种方案，一种是将节点数使用宏定义的方法定义，然后在部署前修改宏定义的值和对应的config.c文件的节点配置，也就是实验中使用的方法，这种方法简单易行，但是其节点数量的"可变性"是静态的，也就是说一旦部署，就无法改变节点的数量，并不算是真正的可变。而且由于投鼠忌器，我没有修改prog3.c文件中的节点配置为宏定义模式，也就没有办法验证这种实现在其他节点数量时的可行性。

另一种方案则是使用动态分配的方式对各个节点的属性进行调整，但是这需要额外的代码来处理新的节点的问题，比如新节点需要发送一个"包"来通知其他节点，而如果要实现这个包势必要对原有的测试代码进行修改，从而实现对包的区分，同时节点的properties比如邻居信息等可能需要大量更改数据结构以实现动态性，需要考虑的各种情况会带来相当大的工作量，这对于一个普通的验证性作业有些overkill了（~~其实就是懒~~），因此对于这个方案并没有去实现任何除了空想之外的工作。

最后，通过此次实验，对路由算法有了更深入的理解，也对程序结构的规划有了新的认识。