

**计算机网络课程设计**

**报告**

实习任务名称： Project 3 实习报告

学 号： 20171001561

姓 名： 周麟

**中国地质大学地理信息工程学院**

**软件工程系**

**2020年 5 月**

**1.生成路由表**

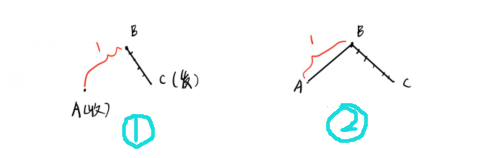
在一个生成树中，起始状态只有一个树形链路，并没有路由表，所以每个结点都要进行flood。在初次flood过程中当到达每个结点时，这时的包的类型是DiscoveryPacket，说明都来自邻居结点，都要记下邻居节点的信息。信息包括来自的端口，还有距离，还有邻居节点的名字。这里用两个字典ports和router\_table分别保存邻居结点名字对应的端口，还有自身到邻居结点对应的距离。因为包类型是DiscoveryPacket说明来自邻居结点，所以距离都是1，但是更新完路由表之后，就要将已经更新的路由表再次flood，分享给整个生成树，然后各个结点继续更新路由表，这时的包的类型就改成RoutingUpdate。

**2.更新路由表**

当需要对路由表进行更新操作时，必须判断此时收到包与当前节点之间的关系。有三种关系：

1. 收到包的来源在当前结点路由表中不存在。
2. 收到包的来源在当前节点存在，但是距离当前结点最近的结点在当前结点路由表中不存在
3. 收到包的来源在当前结点的路由表中不存在，但是在包来源结点路由表中距离当前节点最近的结点在当前节点中的路由表中存在。

为什么要强调最近的结点，因为最近结点的跳数是一。



如图①：这说明发包来源结点离当前节点最近结点原来不在当前结点的路由表中。

这个时候，C到A距离就可以看成BC+1。更新相应的ports和router\_table中的数据。

如图②：可以看成发包来源点离当前节点最近的结点原本就存在当前节点的路由表中，且包来源结点也存在当前节点的路由表中。也可以看成发包来源点离当前节点最近的结点原本就存在当前节点的路由表中，但是包来源结点不在当前节点的路由表中。

但是这两种情况的处理方式不同：

对于第一种处理方式：

当出现下图这样的情况，就比较AB和AC的距离。将之前路由表中的距离和现在距离相比，如果现在距离更短，就更新路由表的数据。

即：

if router\_table[self][packet.src] > packet.paths[temp] + 1:

router\_table[self][packet.src] = packet.paths[temp] + 1

self.ports = port对于第二种处理方式：

和图一的解决方式相同，由图可以发现，AC的距离都是AB+BC，所以他们的操作是相同的。

即：

try:

router\_table[self][packet.src]

except KeyError:

try:

router\_table[self][temp]

except KeyError:

router\_table[self][packet.src] = packet.paths[temp] + 1

self.ports = port

new\_m = RoutingUpdate()

new\_m.add\_destination(temp, router\_table[self][packet.src])

self.send(new\_m, port, flood=True)

return

router\_table[self][packet.src] = packet.paths[temp] + 1

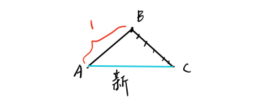
self.ports = port

new\_m = RoutingUpdate()

new\_m.add\_destination(temp, router\_table[self][packet.src])

self.send(new\_m, port, flood=True)

return



在处理完上述情况之后，将该路由表的更新再flood到每个结点，让每个结点据此更新，这时候的包的类型改成RoutingUpdate。

即：

new\_m = RoutingUpdate()

new\_m.add\_destination(temp, router\_table[self][temp])

self.send(new\_m, port, flood=True)

**3.不同类型包的操作**

（1）当收到类型为DiscoveryPacket的包，首先要判断这个包是否在整个生成树中连接。如果满足记录它对应端口和距离还有结点名。即：

if packet.is\_link\_up:

self.ports[cur\_src] = port

router\_table[self] = {}

router\_table[self][cur\_src] = 1

否则就删除这个结点，即：

else:

del self.ports[cur\_src]

（2）当收到类型为RoutingUpdate的包，先判断这个包的路径和端点和路径中的每个结点的距离是否大于最大跳数，在这个实习中规定最大跳数为100。即：

for path in packet.paths:

if packet.paths[path] >= self.max\_path\_numbers:

del packet.paths[path]

满足这个条件，才可以执行上述更新路由表的操作。

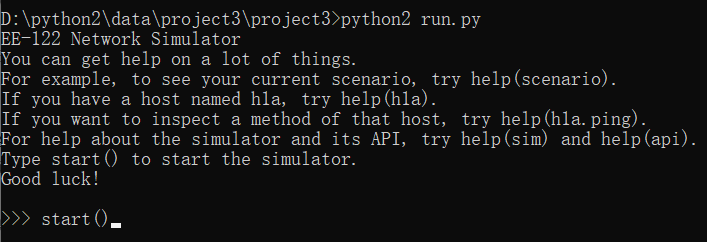
（3）如果收到其他的类型的包，就查找路由表得出相应的端口，然后发送出去，这个时候不需要flood。即：

port = self.ports.get(packet.dst)

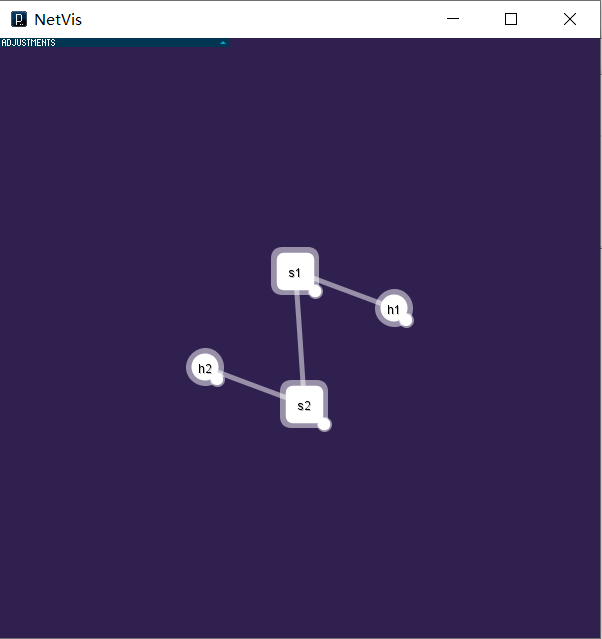
self.send(packet, port, flood=False)

**4.运行截图**

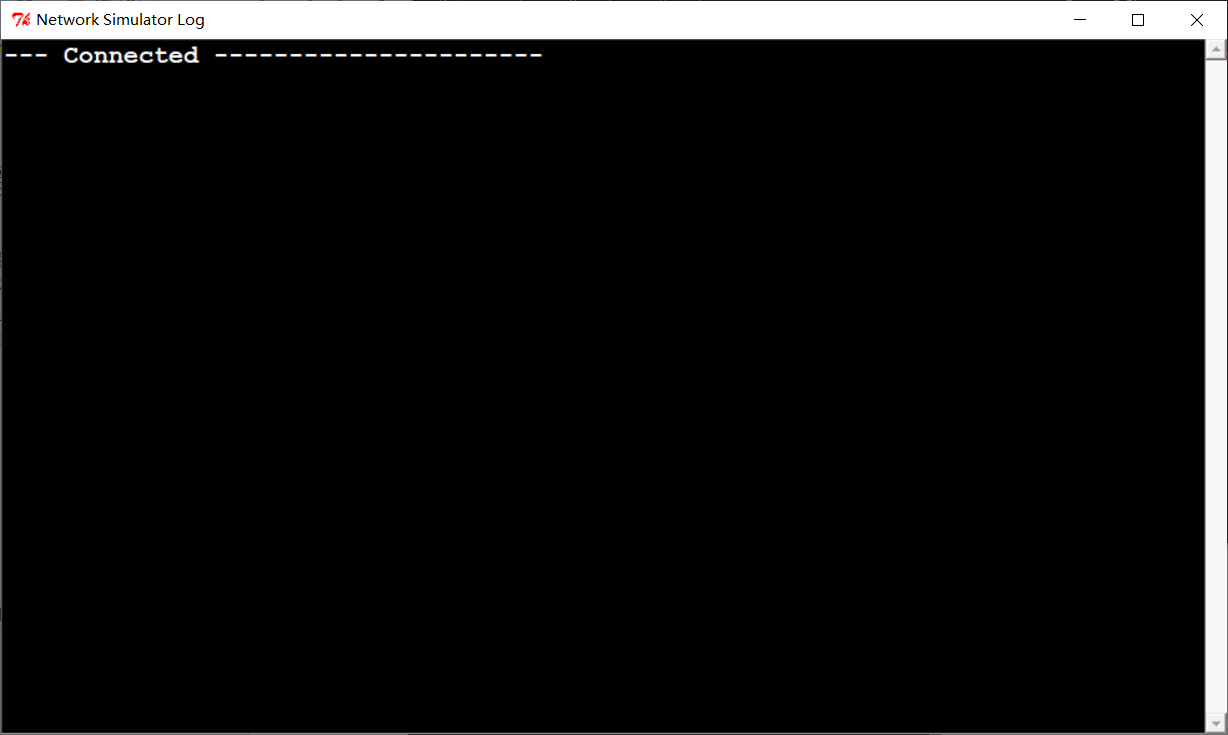
命令行窗口：



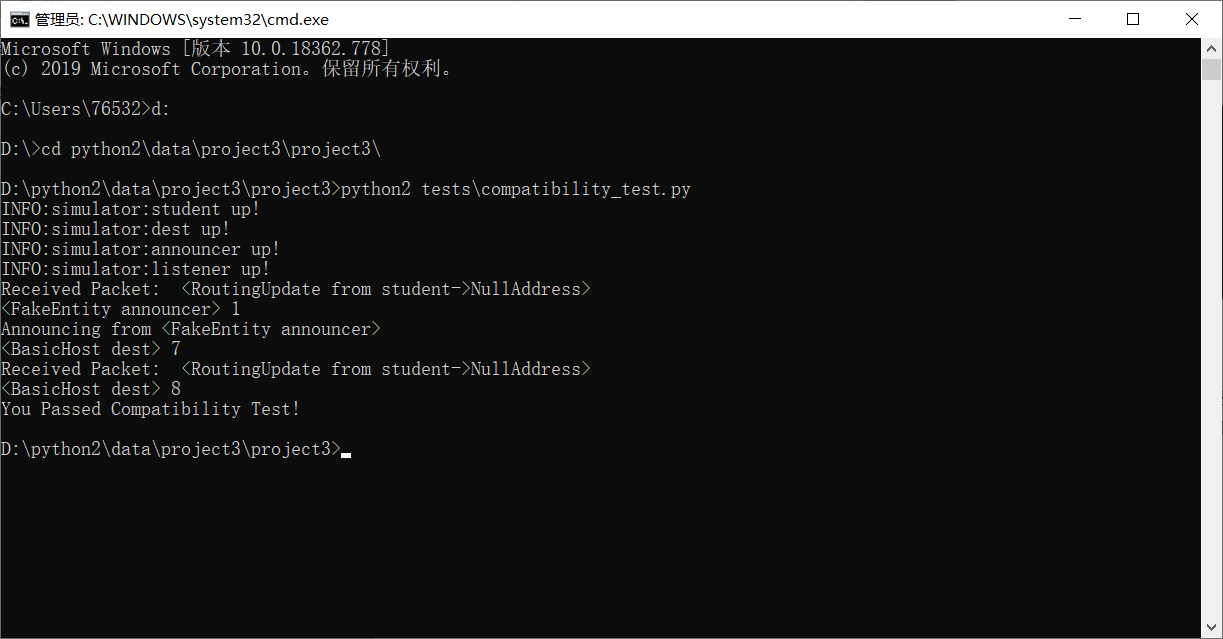
可视化：



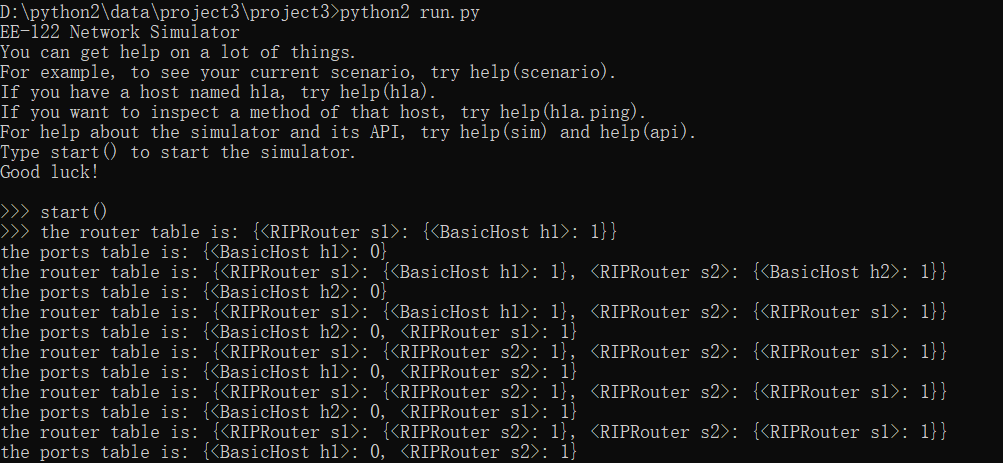
**Log窗口：**



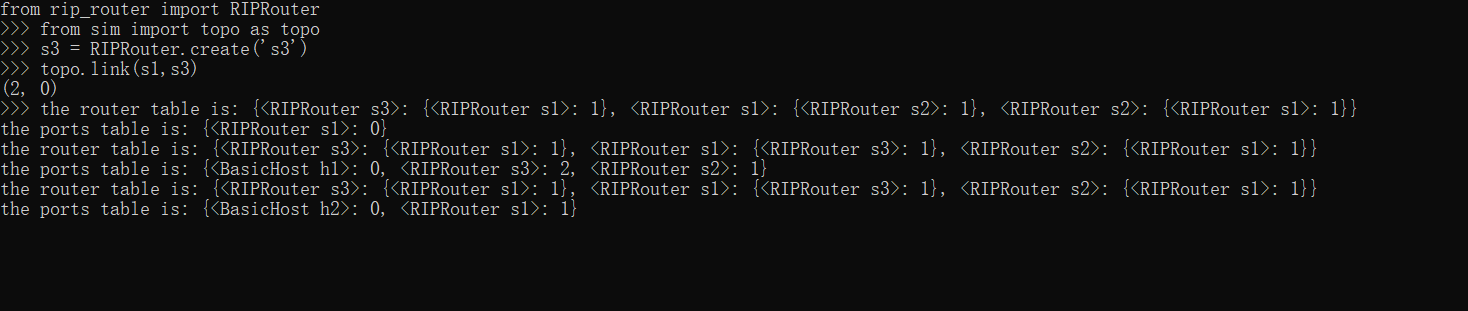
**测试窗口：**



**初始的路由表：**



**添加S3点之后：**



**总结**

本次实验实现了Rip的路由算法，模拟了在收到不同包的不同情况下对于路由表的更新的操作。通过这次实习，通过老师给的这个工程文件和可视化工具，让我能很直观的观察到在进行包flood时，包的具体位置和路由表的生成和更新的过程。