READ ME

姓名：周宁
学号：20171004140
班级：111171

实习思路:

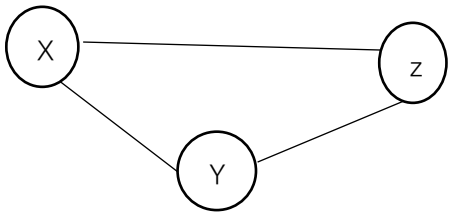
LearningSwitch：

1.对于学习型交换机设置一个字典类型的路由表 routingTable，用来储存目的地和端口号，key 是目的地（包括邻居已经邻居能到达的地方）,value 就是端口号。

2.当收到一个包时，首先判断这个包是否在自己的 routingTable 中，如果在就向相应的端口转发它，如果不在就记录下包的来源和相应的端口，并且将这个包广播出去。

RIPRouter:

1.对于 RIPRouter,设置的表格有三个,第一个是 routingTable,用来储存目的地和端口号，key 是目的地（包括邻居已经邻居能到达的地方）,value 就是端口号；第二个是 DistanceTable，用来储存目的地，和到达目的地的距离，key 是目的地（包括邻居已经邻居能到达的地方），value 是距离（在这里面是跳数）；第三个表 Table，是一个二维的字典储存着通过邻居到达目的地的距离（见下图）。所以对于 RIPRouter 需要维护三张表格。



Table：节点 x 的表格

	目的地 y	目的地 z	
通过邻居 y	1	2	
通过邻居 z	2	1	

2.当收到一个 DiscoveryPacket 时，进行判断 ：

if self.routingTable 中是否包含 packet.src and packet.is_link_up==true:

说明是发现了新邻居，将其加到 routingTable 中，并且对 DistanceTable、table 做相应的更新，然后从自己这里发一个 RtUpdatepacket 包，更新的 path 为 self 到的 packet.src 距离

elif self.routingTable 中包含(packet.src) and packet.is_link_up==False

说明这个邻居跟自己断开了连接，那么首先是从 routingTable、DistanceTable、table 删除与 packet.src 相关的信息，接下来继续从 routingTable、DistanceTable 中删除

通过 packet.src 到达的目的地的信息，并且将信息加入到自己做的 RtUpdatepacket 包中，申明到达这些地点的距离为 inf，最后在 table 中搜索，看存不存在 self 可以到达的目的地但是 routingTable 中没有记录（之前删除的时候可能会将这些目的地删除掉），如果存在就把这些目的地加入到 routingTable 中，并且选择最优的（跳数最少的）。这个信息也需要加入到 RtUpdatepacket 中（可能会覆盖掉点之前设置距离为 inf，但是这种覆盖往往是正确的），最后将这个 RtUpdatepacket 包发出去。

3. 当收到一个 RtUpdatepacket 时，要明确两点(1)RtUpdatepacket 包一定是邻居发过来的，(2)邻居可以到达的点我也可以到达。明白这两点后就对中 RtUpdatepacket 包中 path 进行遍历：

For dest in packet.all_dests():

If dest 不在 self.routingTable 当中：

说明这个地点以前没有记录过，就将此包记录在 self.routingTable 中，并且更新 self.routingTable 和 table，并且需要将更新的信息记录在 RtUpdatepacket 中。

Else:

说明这个目的地一定在 self.routingTable 中，这时需要看关于 dest 的信息是否需要更新，所以进行判断：

if DistanceTable[dest]>packet.getdistance(dest)+DistanceTable[packet.src]:

说明如果 self 通过 packet.src（这个邻居）到达 dest 比 self 之前到达 dest 的距离要短，需要对到达 dest 的端口和距离进行更新，并且将更新信息加到 RtUpdatepacket 中

Elif routingTable[dest]==routingTable[packet.src] and packet.get_distance(dest)+DistanceTable[packet.src]>DistanceTable[dest]:

说明 self 开始就是通过 packet.src（这个邻居）到达 dest，但是 packet.src 到达 dest 的距离变大了，这时候就是课本上出现的情况了，为了避免重复计数，我们需要采用毒性逆转的方式解决问题。首先我会从 routingTable、DistanceTable 中删除这个目的地，并且在包 RtUpdatepacket 中加入更新信息，说明到这个点的距离为 inf。

在上面两个判断完成后就可以进行发送 RtUpdatepacket 包了。

接着除了上述判断外，我们还需要将所有的 dest 信息（通过这个邻居到达 dest 的距离）加入到 table 当中，这个加入十分重要，所以接着：

self.table[packet.src][dest]=packet.get_distance(dest)+self.DistanceTable[packet.src]

上述过程均在 for 循环当中完成，在循环完成后，我们还需要遍历 table，寻找是否有存不存在 self 可以到达的目的地但是 routingTable 中没有记录（之前进行毒性逆转时删除掉的），如果存在，就把这些目的地加入到 routingTable 中，并且选择最优的（跳数最少的）。然后重新制造一个 RtUpdatepacket 包，并且将更新的信息放在当中发送出去。

4. 收到普通的 ping 或者 pong 包，直接查表转发。

实习遇到的问题：

1. 开始时我设计的 **RtUpdatepacket** 包中 path 是将 self 的 DistanceTable 发送出去，发现这样发送出去的包很大，并且会影响一些包的更新。

解决方法：采取更新了什么信息才发送什么，这样 **RtUpdatepacket** 包小了不少，并且更新也准确了很多

2. 在收到 **RtUpdatepacket** 包时，没有判断 dest 是否是 self 自己，全部加入到了 routingTable 当中，结果导致删除的时候，找不到这个键值，如下图：

```
File "E:\Filesave\PythonCode\project3\rip_router.py", line 49, in handle_rx
 for neighbor in  self.table[dest]:
eyError: <BasicHost h1b>
```

解决方法：

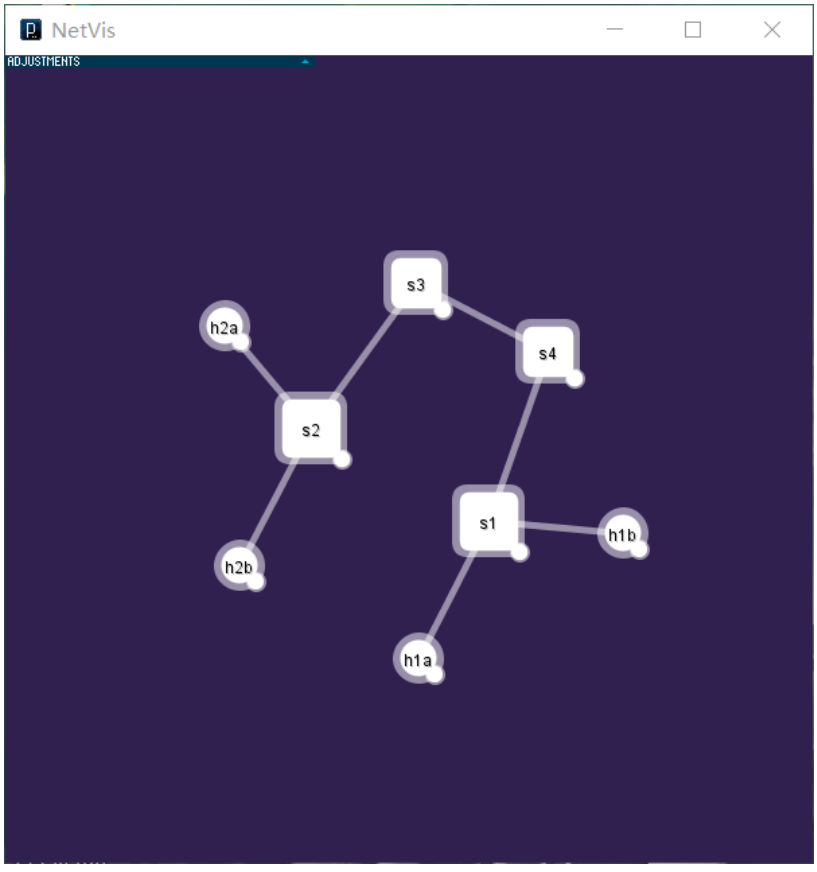
在遍历时加一个判断条件

```
    if dest==self:
        continue
```

实习结果展示：

LearningSwitch：

场景：



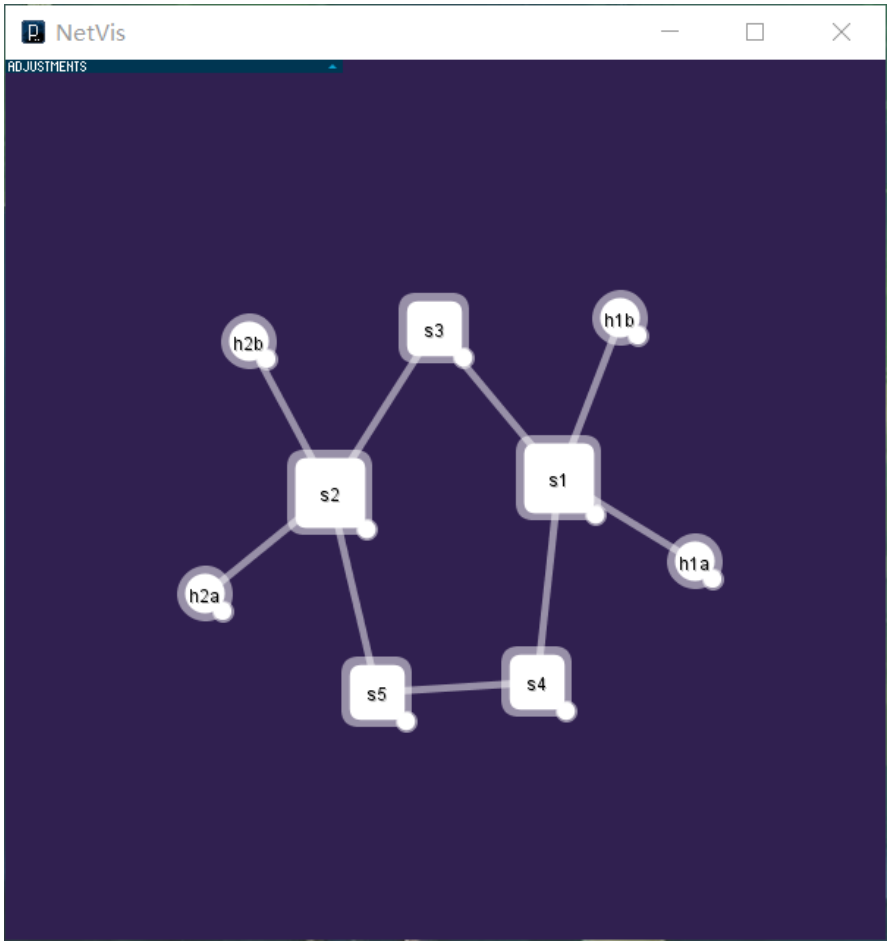
测试用例

```
If you want to inspect a method of that host, try help(h1a.ping).
For help about the simulator and its API, try help(sim) and help(api).
Type start() to start the simulator.
Good luck!

>>> start()
>>> h1a.ping(h1b)
>>> h1a.ping(h2b)
>>> h2a.ping(h1b)
>>>
```

结果：成功

RIPRouter:

场景：



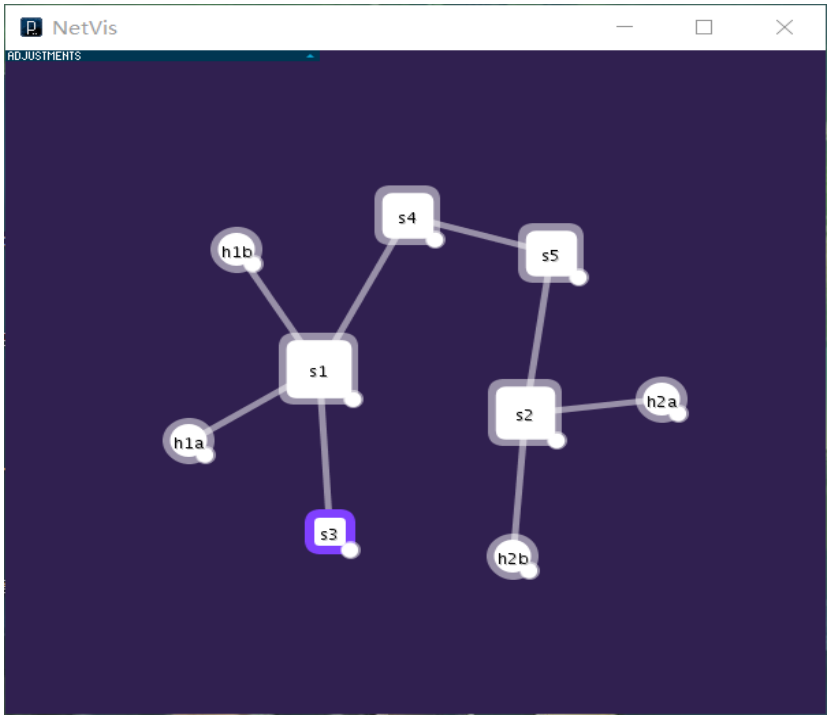
测试用例：

```
If you have a host named h1a, try help(h1a).
If you want to inspect a method of that host, try help(h1a.ping).
For help about the simulator and its API, try help(sim) and help(api).
Type start() to start the simulator.
Good luck!

>>> start()
>>> h1a.ping(h1b)
>>> h1a.ping(h2a)
>>>
```

结果：成功

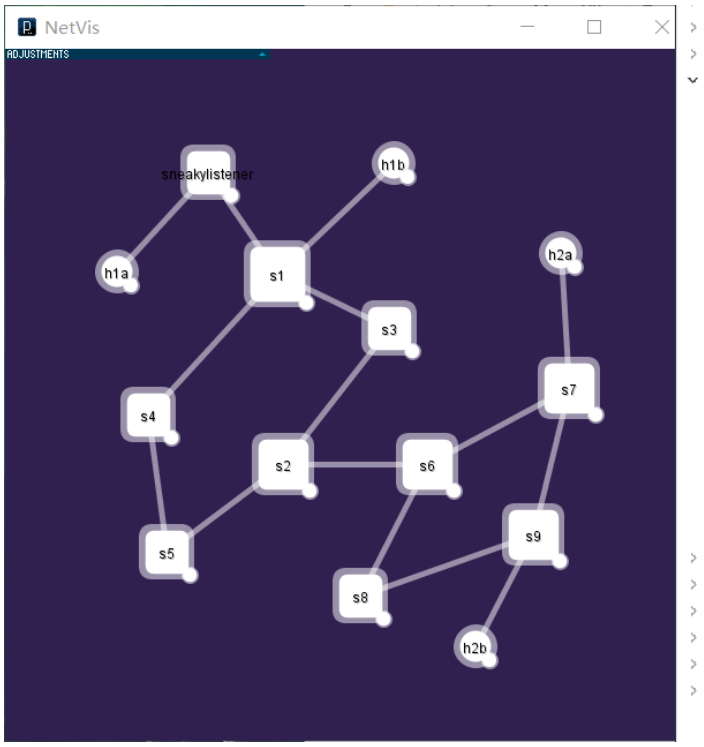
测试用例：

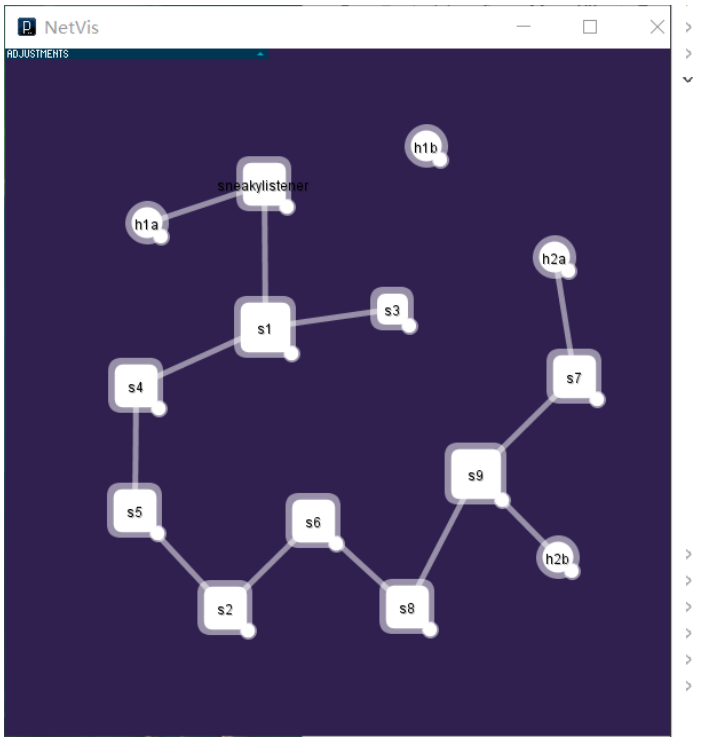
```
EE-122 Network Simulator
You can get help on a lot of things.
For example, to see your current scenario, try help(scenario).
If you have a host named h1a, try help(h1a).
If you want to inspect a method of that host, try help(h1a.ping).
For help about the simulator and its API, try help(sim) and help(api).
Type start() to start the simulator.
Good luck!

>>> start()
>>> topo.unlink(s3, s2)
>>> h1a.ping(h2a)
>>> h2a.ping(h1b)
>>>
```



结果：成功

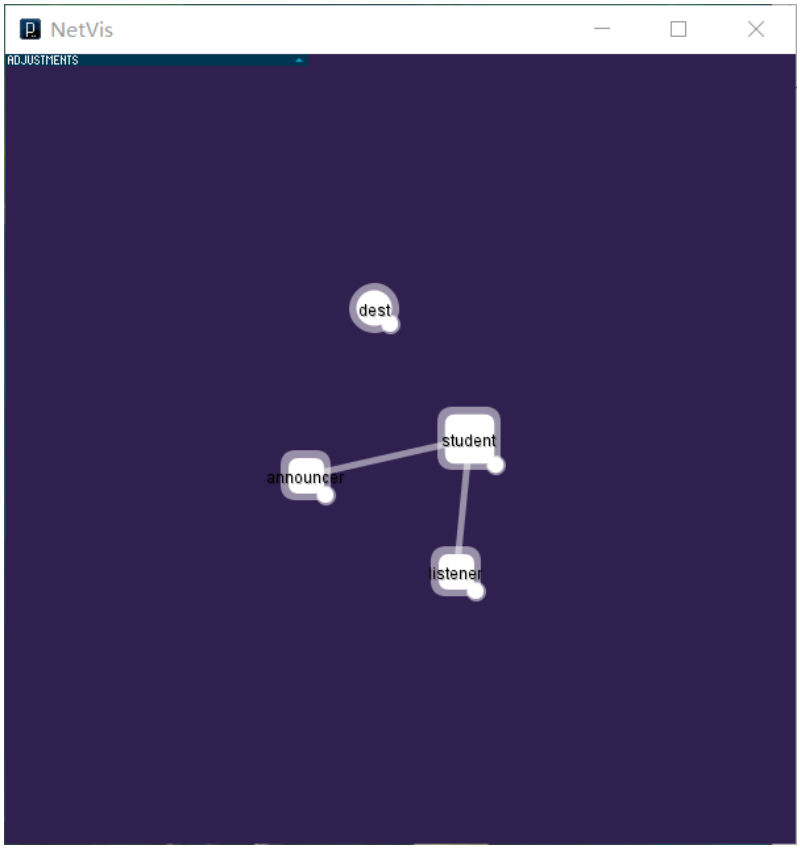
场景：


NetVis
ADJUSTMENTS
sneakylistener
h1b
h1a
s1
h2a
s3
s7
s4
s2
s6
s9
s5
s8
h2b


NetVis
ADJUSTMENTS
h1b
sneakylistener
h1a
h2a
s3
s1
s7
s4
s9
s5
s6
h2b
s2
s8

结果：成功

场景：




```
INFO:simulator:student up!
INFO:simulator:dest up!
INFO:simulator:announcer up!
INFO:simulator:listener up!
Received Packet: <RoutingUpdate from student->NullAddress>
<FakeEntity announcer> 1
Announcing from <FakeEntity announcer>
<BasicHost dest> 7
Received Packet: <RoutingUpdate from student->NullAddress>
<BasicHost dest> 8
You Passed Compatibility Test!
```

结果：成功

本次实习的感悟：

本次实习我们尝试着写了一下 DV 算法，这个算法看似简单，其实在细节方面还是有一些深度的，如果没有思考清楚的话，就会像我开始一样陷入困境，不过在你思考完成以后便会觉豁然开朗了，尤其是毒性逆转，开始我一直没有弄得为什么需要这样，现在我认为我已经明白了，防止一种重复的环路一样。本次实习总体来说收获还是蛮大的，第一次写分布式算法，并且明白了写程序之前一定要想清楚明白，不然真的会再后面修修改改十分痛苦。