Project3-Routing Protocols

——路由仿真

作者（Author）

姓名：
学号：
班级：

AnDJ

解决过程

一、 框架理解和使用

这个路由仿真的框架代码比之前的多很多，不过因为文档里明确说了只需要重写一两个部分的代码，更深入的细节我们也不可能了解地到。

1. 在 run.py 里面有很多的外部配置信息需要提前了解，主要有：

   Switch: 这个是我需要实现的东西，默认提供了不做任何处理的 flood 发送的 switch，我需要实现 learningswitch 和 riprouter，指定这两个为 switch 来嵌入框架。

   Scenatio: 默认给出了 linear 和 candy 两种场景，场景就是需要搭建的网络具体情形。在写两个 switch 时也要搭建合适的场景，在这里指定。

2. Switch 结构

   我需要重写的就是 switch 的 hand_rx 函数，这个函数在该 switch 收到包后调用，处理逻辑都需要在里面。Switch 的 get_port_count 等函数作为辅助使用。Hand_rx 函数的输入为 packet, port。

3. 三种包的结构

   DiscoveryPacket。这种包是泛滥包，包含的信息有：包源 src，去向 dst，延迟 latency 和是否是连接包 is_link_up。该包在两个 entity 连接后互相发送 discovery 包，is_link_up 为 True，延迟为 1，在两个 entity 断开也会发包，is_link_up 为 false。

   RoutingUpdatePacket。这种包是更新包，可以通过内置函数添加表中哪些项被更新和更新的内容，用于 rip 向邻居发送更新。

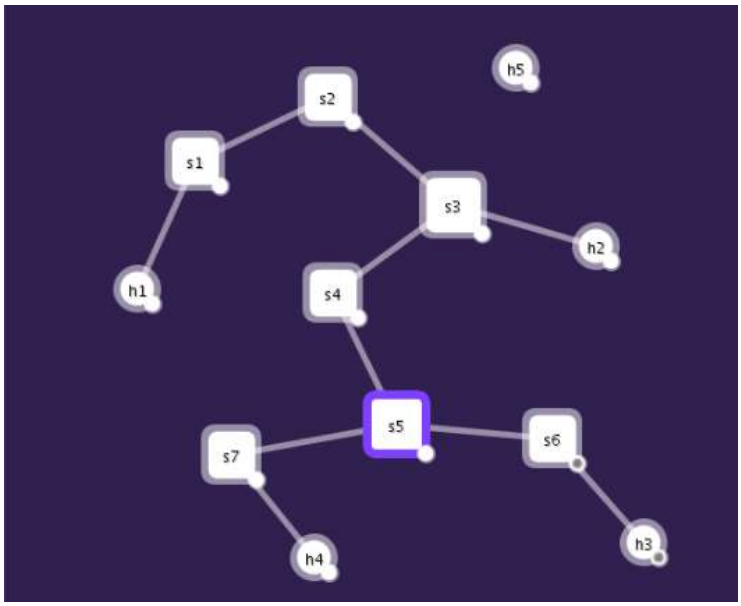
   Ping 和 pong 包。这两个包实现了 ping 和 pong 的功能，用来测试两点之间的发包过程。

4. 框架载入运行过程

   这一部分是我调试出来的，我认为是这样的：加载场景，两个 entity 连接后会互相发 discoverty 包。如果两个实体断开，那么互相会发 is_link_up 为 false 的 discovery 包。Ping 到包需要走到的目的地会产生相应的 pong 包返回。

二、 LearningSwitch 实现

这个 switch 是基于树形拓扑的，因此需要自行构造一个树形拓扑的场景。



这个 switch 要实现的功能比较简单，不能处理环路的情况，它需要做的是，拿到 discovery 包后将来源记录下来，在后期有往这个地方走的包可以查询表的方式直接 send。当在表中记录下来后，再洪水发送到其他端口。这样的话，全图的所有 switch 都有包含全图的点的一张表。我使用了一个字典结构 switchTable 进行存储，key:value 为 dst:port 这样的形式。当接到其他包如 ping 包时直接查表发送。这个实现思路较易。

三、 RIP Router 的实现

上面的 learning switch 只能实现树形拓扑的情况，这里要实现 RIP Router，这个需要能够解决环路的情形，实现课上讲的 Bellman-Ford 算法。

1. 建表。RIP switch 要存储一张关于距离的二维表 switchTable:

	neighbor1	neighbor2	neighbor3	neighbor4	…
point1	1	1	5	4	
point2	1	5	2	5	
point3	4	6	4	7	
point4	7	5	2	4	
point5	7	7	6	9	
…					

还有一张存储邻居的端口的表 switchPort:

	neighbor1	neighbor2	neighbor3	neighbor4	…
port	0	1	2	3	

这两张表我都是用字典来实现的。

2. 处理 discovery 包

1） 连接包（is_linked_up=true）

前面说过，在程序开始时两个 entity 要建立连接会互相发 discoverty 包，距离向量算法建立这个表的过程是要先初始化，那么这个初始化过程就是收到邻居发的 discovery 包，体现出来的就是 switchTable 的右向扩展，同时为了保

证这个表的完整，也就是说纵向保存的全图的点也要拷贝到右向扩展的纵向中去。

原表：

	neighbor1
point1	1
point2	1

先右扩展：

	neighbor1	neighbor2
point1	1	inf
point2	1	inf

再下扩展：

	neighbor1	neighbor2
point1	1	inf
point2	1	inf
neighbor2	inf	

补上角上的值：

	neighbor1	neighbor2
point1	1	inf
point2	1	inf
neighbor2	inf	1

扩展好后，发送更新包，更新包中添加更新的距离，这里就是添加 dst=neighbor，distance=1 的更新包。这里需要记住，就是说 discovery 只是发给邻居的，拿到 discovery 包后扩展表，不再洪水发送该包。注意排除在表中对自身节点建表的情况。

2) 断接包

这样的 discovery 包是在两个实体断开后相互发送的，和连接包不同，当一个实体收到断接包后，先是将 switchTable[packet.src]的所有元素置为 inf，因为两个实体无连接，那么当前 switch 不能够以 packet.src 为下一跳跳到任何地方。这样地话，可能会造成最优路径的变化。比如原先当前 switch 可以经 packet.src 花费最小地跳到指定点，但是断连了后，最优路径变为了其他的方式，同时也必须让该 switch 的邻居也对最优的情况变化进行更新，因此，如果出现了最优跳转改变，必须发送 routing update 给邻居进行调整。

3. 处理 routingUpdate 包

拿到 routingUpdate 包后肯定是该 switch 的邻居已经建好了，因此 routingUpdate 需要处理的情况只有两种：

Case 1:

routingUpdate 包包含了一些地图上刚加入的点但目前的表中没有的点。比如说 routingUdate 有一项是 packet.src 到 newPoint 点距离更新为 d，但是这个点是新添加到网络中来的，导致本表没有记录，因此需要针对此对本表进行纵向扩展，即添加一个新的地图上的点。过程如下：

原表：

	neighbor1
point1	1
point2	1

纵向扩展：

	neighbor1
point1	1
point2	1
newpoint	inf

再将(packet.src，newpoint)值更新为包中的 distance。产生更新包。

Case 2:

routingUpdate 包进行的更新的数据项在表中存在。这样的话，利用距离向量算法对表进行更新。我外层针对两个 case 设置了一个变量进行监控，如果该表更新，那就将更新的位置包装到新的 routingUpdate 再次向其他所有端口发送，延续更新下去，直到出现接收到的 routingUpdate 不产生表的更新，那么就不用发送新的更新包。

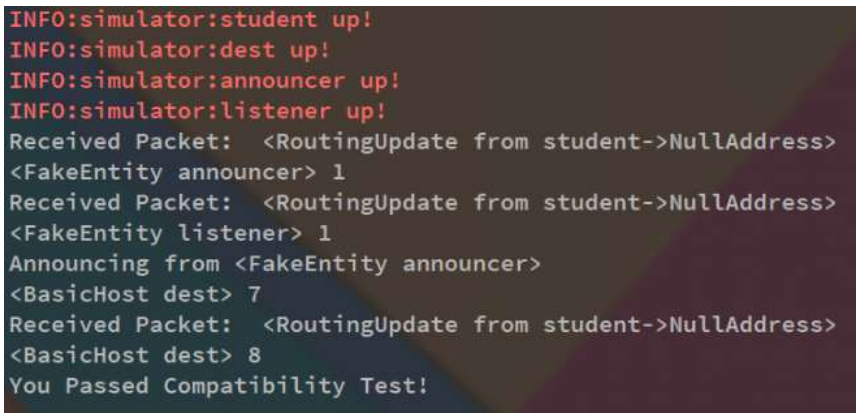
4. 处理其他包

   这里的其他包就是 ping 和 pong 包。拿到 ping 包和 pong 包后，在距离表中找到以哪一种下一跳能代价最小地发送到该包，并且这一跳是可以跳的（也就是 switch port 存在能跳到该下一跳的端口），那么就从这个端口发送出去。也就是两个条件，下一跳发包代价最小，并且能跳到下一跳。

四、 测试

代码框架给出了两个测试，一个是测试是不是保持兼容性，一个是给出了一个情景并测试连通。

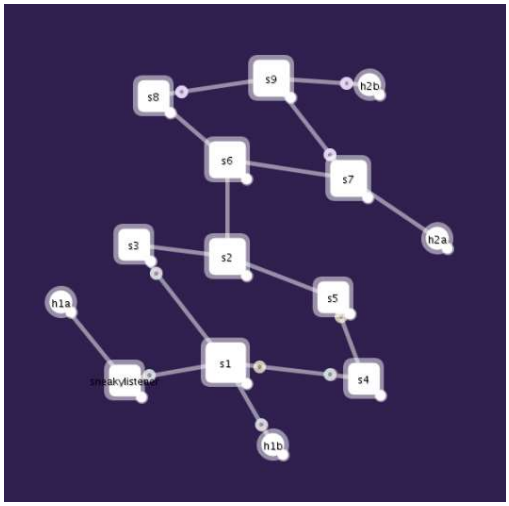
测试一：



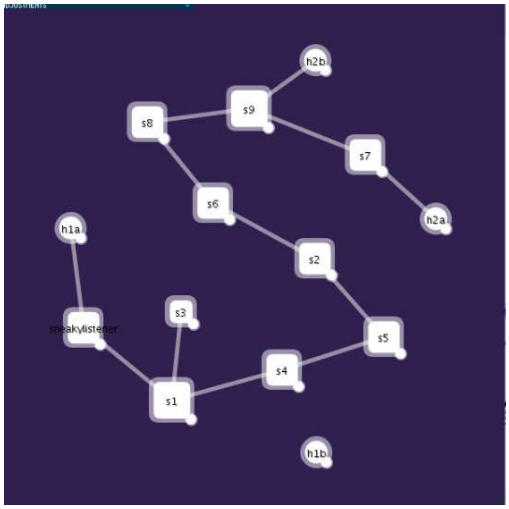
测试二：

测试二是这样的，建立一个复杂情景，建立好后会出现故障状态，就是有两条已经建好的路径断开不能使用了，需要考虑这种情况。以上已经将该问题解决。

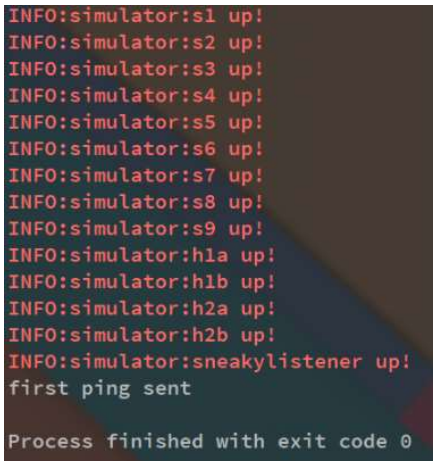
测试结果：



建表部分断开前



建表部分断开后



可以看到该 ping 包到达了指定位置并正常退出，测试成功。