《通信与网络》动态路由实验报告

漆耘含 无 63 2016011058

一、 实验目的

1．利用提供的网络仿真器，复习课堂所学计算机网络的相关路由算法知识，理解两种动态路由算法的基本思想，掌握二者的相同点和不痛点；

2．通过阅读相关代码实现，结合 GUI 界面，掌握距离矢量法（DV）的具体实现过程；

3．完成 Dijkstra 算法的部分代码实现，结合 GUI 界面，验证算法实现正确性，掌握链路状态法（LS）的实现过程；

4．通过阅读并完成相关代码实现，结合 GUI 界面，分析链路故障时，两种动态路由算法的解决故障过程；

5．初步掌握在 Linux 系统环境中，利用 python 实现网络方针和算法实现的能力。

二、 实验内容

1．（原第 C 题）理解代码实现，阅读 DVrouter.py，结合具体的注释，理解 Dv 算法的实现过程和解决链路故障、新增链路等问题时的方法。针对 0_net.json 文件，结合 GUI 界面，重点分析__init__()、handlePacket()、updateNode()、debugString()函数的功能和实现，明白__iniy__()函数中定义的 dict 含义。

(1) __init__()函数

__init__()函数的功能是对 DVrouter 类进行初始化，定义字典变量，并且初始化 Router 类，其中字典变量具体的含义如下：

routersNext	路由表
routersCost	从源点到节点的（最小）代价
routersPort	Router 的发送端口

routerAddr	端口号-路由名对应关系

（2） handlePacket()函数

如果是发送数据包（traceroute packet），并且发送的目的节点在路由表中，则发送该数据包；如果发送路由包，先调用更新节点（updateNode 函数），如果路由表更新了，则需要传播。对于 routersAddr 中的每一个端口，如果不等于更新节点的路由表中的端口，则将更新的路由表向其邻居传播（发送路由包）

（3） updateNode()函数

功能是更新节点。如果该包的目标节点（dst）不在 routersCost 中并且不是初始源节点，该包的源节点（src）在 routersCost 中，则将 dst 加入到 routersCost 中，并加入 routersNext；如果 dst 和 src 都在 routersCost 中，则选择最小代价，并更新 routersNext 和 routersCost。

（4） debugString()函数

在运行窗口中得到每一个路由的 routersNext 和 routersCost（点击任一路由）

2.（原第 D 题）阅读 LSrouter.py 和 LSP.py，理解 LS 算法的具体实现流程，回答思考题。

(1) 思考题 1：请给出 LSrouter.py 初始化函数中的 dict 含义（key-value）

routersLSP	当前的网络拓扑结构，对每一个节点，存放了与之相连节点的 addr 和 cost（代价）
routersAddr	端口号-路由名对应关系
routersPort	路由器的发送端口号
routersNext	路由表
routersCost	从源节点到节点的（最小）代价

(2) 思考题 2：请给出你对 LSP.py 中更新函数的执行过程的理解

LSP 中有三个变量：addr、seqnum、nbcost，其中，seqnum 应该类似于一种版本号，nbcost 是两节点之间的代价，LSP 更新函数流程如下：当新来的包对应的版本号大于当前版本号，并且当前 nbcost 和包里的 nbcost 不相等的时候，即代表有新的信息，则需要进行更新。

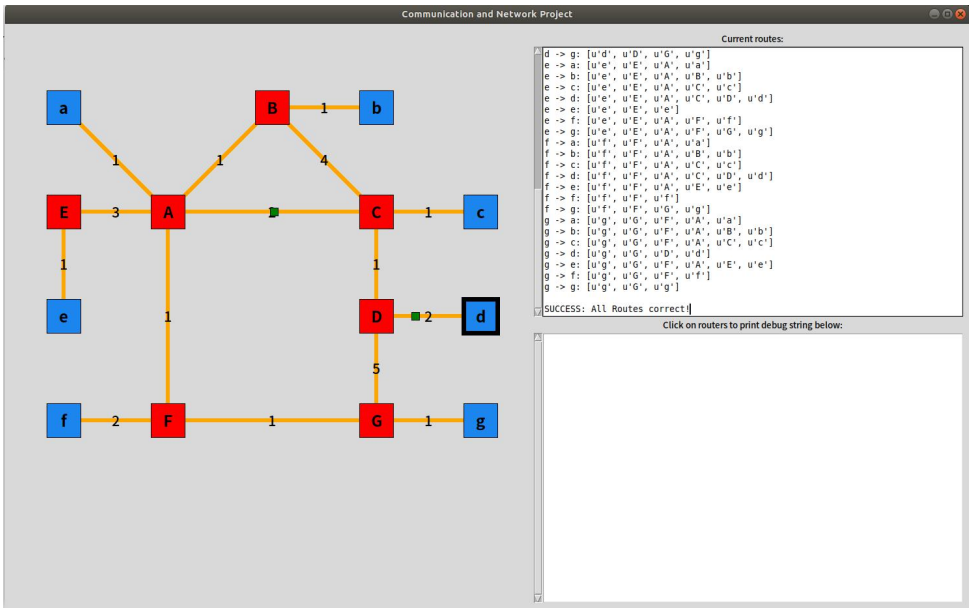
(3) 思考题 3：请给出你对 LSrouter.py 的 handlePacket()函数中 Routing Packet 处理过程的理解

当路由包的出发节点和目标节点相等，即自己传给自己，不需要传播更新；当路由包里面的 addr 信息不在网络拓扑图（routersLSP）中，则将其加入拓扑图中，或者如果包含了新的信息，则需要更新传播转发。

3.（原第 E 题）Dijkstra 算法实现：结合 Dijkstra 算法，补全 LSrouter.py 中的 calPath()函数。针对 0_net.json 文件，结合 GUI 界面，验证算法的正确定。

结合 Dijkstra 算法，利用优先级队列，首先将所有与源节点相连的节点和代价放入优先级队列中，其中，优先级队列是对以距离来进行排序，因此 pop 出来的是距离最小的（优先级最高的）。然后每一次出队的元素，如果 Next 已经在路径中，并且当前节点（addr）不在路径中，则将该节点加入其中，并将与该节点相连的节点入队；如果 Next 和当前节点（addr）都在路径中，则选取最小的代价，并将与其相连的节点入队；如果 Next 与 addr 相等，则直接更新，并将与其相连的节点入队。

验证结果如下：



由此可见，验证成功。

4.（原第 F 题）链路故障分析 1：针对 0_net_events.json 文件，首先观察 DVrouter.py 解决链路故障的过程，如每个 router 中存储的距离矢量和路由表的变化，可以修改 DVrouter.py 中的 debugString()函数进行分析。

通过运行程序可以发现，在运行一段时间之后，C 到 D 的链路断开，通过实际方针，输出节点的 routersCost，如下图（C 节点）：

```
{u'A': 2, u'c': 1, u'B': 4, u'D': 1}
{u'A': 2, u'c': 1, u'B': 4, u'D': 1, u'G': 6}
{u'A': 2, u'c': 1, u'B': 4, u'D': 1, u'G': 6, u'd': 3}
{u'A': 2, u'c': 1, u'B': 4, u'D': 1, u'G': 6, u'F': 3, u'd': 3}
{u'A': 2, u'a': 3, u'c': 1, u'B': 4, u'D': 1, u'G': 6, u'F': 3, u'd': 3}
{u'A': 2, u'a': 3, u'c': 1, u'B': 4, u'E': 5, u'D': 1, u'G': 6, u'F': 3, u'd': 3}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 1, u'G': 4, u'F': 3, u'b': 4, u'd': 3}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 1, u'G': 4, u'F': 3, u'f': 5, u'b': 4, u'd': 3}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 1, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'd': 3}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 16, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'd': 16}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 16, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 16}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 7, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 16}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 7, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 9}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 7, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 7}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 7, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 9}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 16, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 9}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 9, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 9}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 11, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 9}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 11, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 11}
{u'A': 2, u'a': 3, u'c': 1, u'B': 3, u'E': 5, u'D': 9, u'G': 4, u'F': 3, u'f': 5, u'g': 5, u'b': 4, u'e': 6, u'd': 11}
```

从上图可以看出，在链路还没断开的时候，是已经收敛的，而当 C 与 D 的链路断开之后，到 d 的距离立马变为 16，即 cost_max，相当于断开，之后再不断传递路由包，从而到最后的收敛状态。

上面是从现象进行分析的，下面从代码中寻找答案，在链路断开的时候，有

handleRemoveLink()函数，里面将相应的 routersCost 设置为 cost_max，然后在处理路由包的函数里面用到了 updateNode()函数，如下：

```
# choose the less cost or new info
if dst in self.routersCost:
    if src in self.routersCost:
        if (self.routersCost[dst] > self.routersCost[src] + cost) or (self.routersNext[dst] == src and src != dst):
            self.routersCost[dst] = self.routersCost[src] + cost
            self.routersNext[dst] = src
            # set COST_MAX as infinity
            if self.routersCost[dst] >  COST_MAX:
                self.routersCost[dst] =  COST_MAX
            return True, dst, self.routersCost[dst]
```

这里面有处理当链路断开的情况，然后利用路由包处理模块进行广播，从而达到不断更新每个节点的路由表直到最后收敛。

5.（原度 G 题）链路故障分析 2：针对 0_net_events.json 文件，通过补全 debugString()函数，观察 LSrouter.py 解决链路故障的过程。

首先，通过在 debugDtring()函数中编写函数体，类似于 DV 中，代码如下：

```
    def debugString(self):
        out = str(self.routersNext) + "\n" + str(self.routersCost)
        print self.routersCost
        return out
```

即在界面中输出 routersNext 和 routersCost，在终端中输出 routersCost，方便对整个变化过程进行分析，下面也是针对 C 路由进行分析：

```
{u'C': 0, u'B': 4}
{u'A': 2, u'a': 3, u'C': 0, u'B': 3, u'E': 5, u'g': 5, u'G': 4, u'F': 3, u'f': 5, u'c': 1, u'd': 11, u'b': 4, u'D': 9}
{u'A': 2, u'a': 3, u'C': 0, u'B': 3, u'E': 16, u'g': 16, u'G': 4, u'F': 3, u'f': 16, u'c': 1, u'd': 3, u'b': 4, u'e': 6, u'D': 1}
{u'A': 2, u'a': 3, u'C': 0, u'B': 3, u'E': 5, u'g': 5, u'G': 4, u'F': 3, u'f': 5, u'c': 1, u'd': 3, u'b': 4, u'e': 6, u'D': 1}
{u'A': 2, u'a': 3, u'C': 0, u'B': 3, u'E': 5, u'g': 5, u'G': 4, u'F': 3, u'f': 5, u'c': 1, u'd': 11, u'b': 4, u'e': 6, u'D': 9}
```

由上图可以看到，从刚开始稳定的状态，到 CD 链路断开后的稳定状态，LS 的变化过程状态比 DV 的要少，下面从理论进行分析：

在 handleRemoveLink()函数中，首先将 Cost 设置为 cost_max，然后计算依次路由表，再生成一个新包，seqnum（版本号）+1，即代表新的信息，然后将其发送出去，从而不断地广播到每一个节点，从而很快达到收敛。

三、 实验总结

本次大作业是对动态路由进行实际仿真，结合了课堂上学到的关于动态路由的算法，比如距离矢量法和链路法，通过实际编写部分 Dijstra 算法，让我对该算法有了更深刻的认识，同时，对客户-路由的动态过程有更直观的认识。同时，本次实验还用到了 python，让我的 python 编程能力得到一定的锻炼，同时还学习了一下仿真界面的代码，收货颇多！

最后，感谢老师和助教对《通信与网络》这门课的辛勤付出！