实验三　网络层路由实验

刘星雨　　无 08　2020010850

目录

1. 实验目的

a) 理解和掌握链路状态发和距离向量法的实现原理和区别

b) 理解和掌握链路状态法和距离向量法处理链路故障和新增链路的方式

c) 分析和对比链路状态法和距离向量法的计算复杂度和实际收敛速度

d) 初步掌握编程实现路由选择算法的能力

2. 链路状态法

a) 阅读并理解链路状态法中实现的函数功能

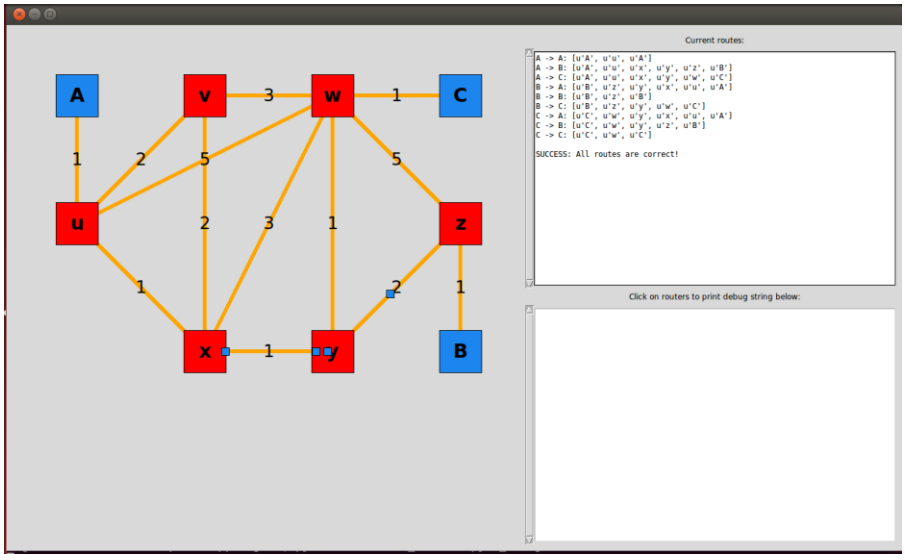
i. 初始化函数中定义的变量的物理含义

```
def __init__(self, addr, heartbeatTime):
    """class fields and initialization code here"""
    Router.__init__(self, addr)  # initialize superclass - don't remove
    self.routersLSP = {}  ### 链路状态包的节点ID
    self.routersAddr = {} ### 路由地址
    self.routersPort = {} ### 路由端口号
    self.routersNext = {} ### 邻居列表
    self.routersCost = {} ### 邻居的链路开销
    self.seqnum = 0 ### 序列号
    self.routersLSP[self.addr] = LSP(self.addr, 0, {})
```

ii. 结合链路状态法原理以及前向搜索算法流程，在下文中对该部分代码注释并补全关键代码

```python
def calPath(self):
    # Dijkstra Algorithm for LS routing
    self.setCostMax()
    #the present router uses itself to initialize a record if the verified table ,the cost of the record is 0
    # put LSP info into a queue for operations
    Q = PriorityQueue()
    #set a priority queue to store infomation
    for addr, nbcost in self.routersLSP[self.addr].nbcost.items():
        #for every neighbor of next
        Q.put((nbcost, addr, addr))
        #put it into the shitan table
    while not Q.empty():
        #if shitan table is not empty
        Cost, Addr, Next = Q.get(False)
        #pick the smallest cost point in the shitan table
        if Addr not in self.routersCost or Cost < self.routersCost[Addr]:
            #if neighbor point is not in the verified table ,or the cost is smaller than it already had
            ### TODO: Add two lines code to update Cost and Next for Addr
            self.routersCost[Addr]=Cost
            self.routersNext[Addr]=Next
            #replace present record with <neighbor,cost,nexthop>
            if Addr in self.routersLSP:
                #if neighbor point is not in the shitan table
                for addr_, cost_ in list(self.routersLSP[Addr].nbcost.items()):
                    Q.put((cost_ + Cost, addr_, Next))
                    #put it into the shitan table
```

iii. 基于正常 1_net.json 验证正确性



1. 客户端之间最小开销路径

源客户端→目的客户端	最小开销路径
A→A	A-u-A 2
A→B	A-u-x-y-z-B 6
A→C	A-u-x-y-w-C 5

B→A	B-z-y-x-u-A 6
B→B	B-z-B 2
B→C	B-z-y-w-C 5
C→A	C-w-y-x-u-A 5
C→B	C-w-y-z-B 5
C→C	C-w-C 2

2. 路由器 x 的转发表

目的节点	下一跳转发节点	开销
A	u	2
B	y	4
C	y	3
u	u	1
v	v	2
w	y	2
x	*	0
y	y	1
z	y	3

3. 路由器 x 的 LSP 信息记录

y—1;u—1;w—3;v—2

b) 链路状态改变实验

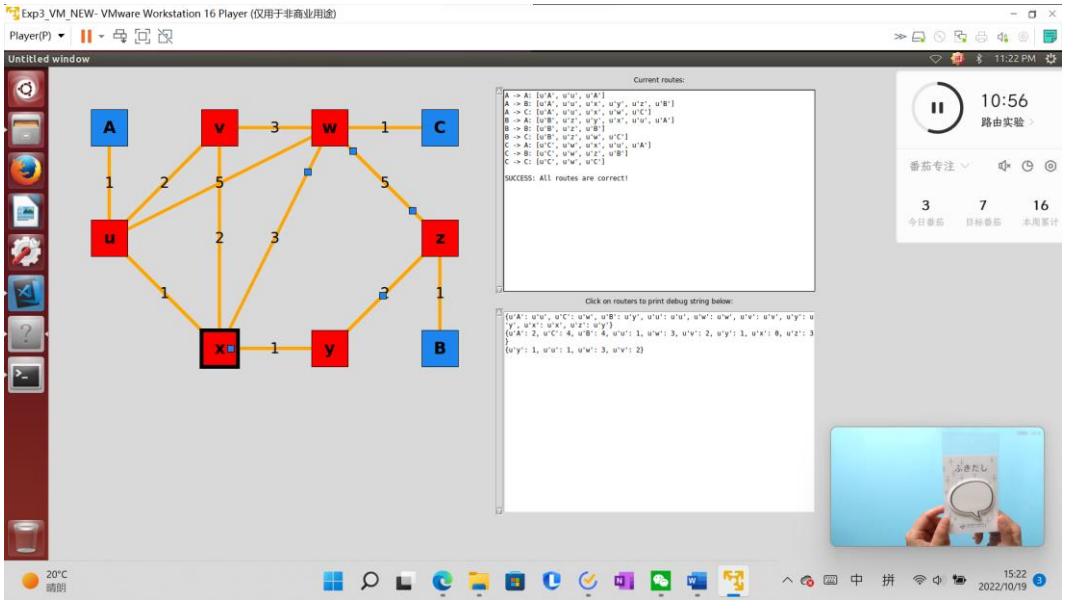
i. 阅读 handleRemoveLink 并注释

```python
def handleRemoveLink(self, port):
    """handle removed link"""
    addr = self.routersAddr[port]
    #find current point the address of the port which is removed
    self.routersLSP[self.addr].nbcost[addr] = COST_MAX
    #set the distance between two points is cost-max,because it's break
    self.calPath()#reset the dij for new route

    content = {}
    #it's the content of lsp
    content["addr"] = self.addr  # set address
    content["seqnum"] = self.seqnum + 1 #set sequence number
    content["nbcost"] = self.routersLSP[self.addr].nbcost
    #set neighbor cost
    self.seqnum += 1
    #sequence number of the current point +1
    for port1 in self.routersAddr:
        #for every port in the address of router
        if port1 != port:
            #if port is not the current one which is removed
            packet = Packet(Packet.ROUTING, self.addr, self.routersAddr[port1], dumps(content))
            #set its packet
            self.send(port1, packet)
            #the port send the packet
    pass
```

ii. 基于链路故障网络用 LS 进行路由选择验证其正确性

1. 最小开销路径



图片上可以看到这里是将之前 w-y 之间的路径 remove 了。表格中标红的部分是由于 remove　w-y 造成的 cost 变化的路径

源客户端→目的客户端	最小开销路径

A→A	A-u-A 2
A→B	A-u-x-y-z-B 6
A→C	A-u-x-w-C 6
B→A	B-z-y-x-u-A 6
B→B	B-z-B 2
B→C	B-z-w-C 7
C→A	C-w-x-u-A 6
C→B	C-w-z-B 7
C→C	C-w-C 2

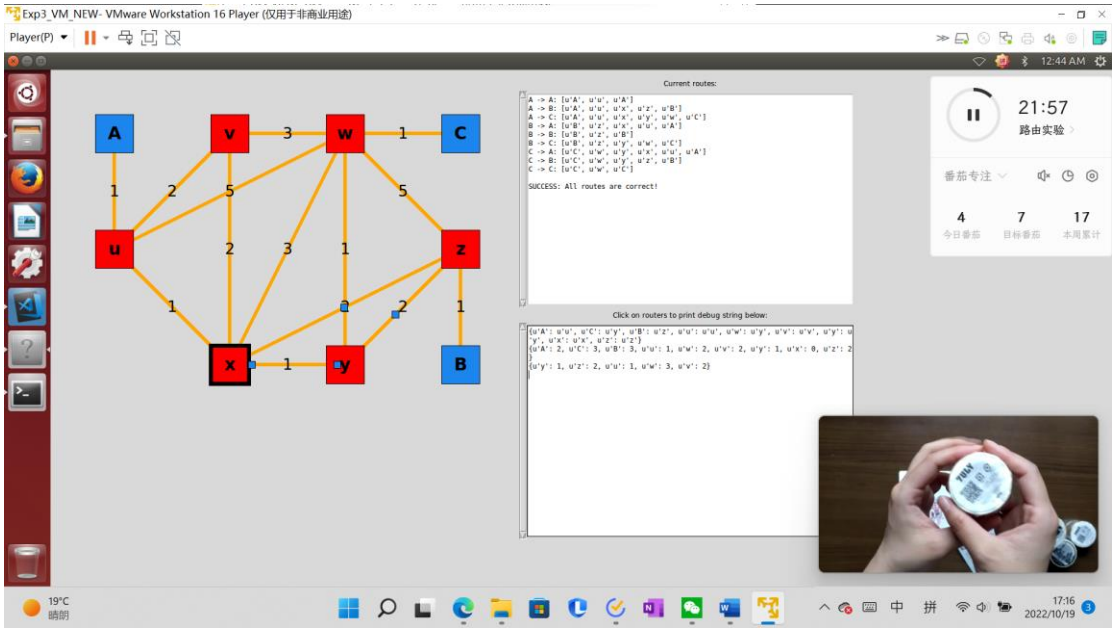
2. 路由器 x 的转发表

目的节点	下一跳转发节点	开销
A	u	2
B	y	4
C	w	4
u	u	1
v	v	2
w	w	3
x	*	0
y	y	1
z	y	3

3. 路由器 x 的 LSP 信息记录

y—1;u—1;w—3;v—2

c) 基于链路新增网络用 LS 进行路由选择



对比可以发现相较于原先的路由，新增了 x-z 这条路径

1. 客户端之间最小开销路径

源客户端→目的客户端	最小开销路径
A→A	A-u-A
A→B	A-u-x-z-B
A→C	A-u-x-y-w-C
B→A	B-z-x-u-A
B→B	B-z-B
B→C	B-z-y-w-C
C→A	C-w-y-x-u-A

C→B	C-w-y-z-B
C→C	C-w-C

2. 路由器 x 的转发表

目的节点	下一跳转发节点	开销
A	u	2
B	z	3
C	y	3
u	u	1
v	v	2
w	y	2
x	*	0
y	y	1
z	z	2

3. 路由器 x 的 LSP 信息记录

y—1;z—2;u—1;w—3;v—2

d) 比较链路状态改变和正常状态，理解 LS 处理状态改变的过程

链路状态改变和正常状态开销路径、转发表、lsp 等信息的区别，只是因为增加或删除了部分路径造成的。因此处理这个问题是路径连接信息在原来的基础上将部分路径的 cost 设置为 max 或者再增加一些连接为它们赋值 cost，随后还用

dij 算法生成一遍，给受到影响的点的 lsp 信息更改一下，就完成了链路状态改变了。

3. 路由选择算法效率比较

收敛时间	链路正常	链路故障	链路新增
距离向量法	24.72	64.89	45.27
链路状态法	34.24	55.64	55.20

4. 思考题

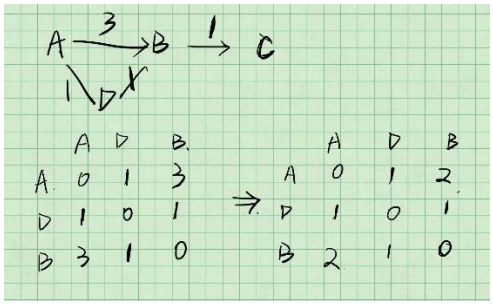
a) 理解 LSP 中 updateLSP 函数

```
def updateLSP(self, packetIn):
    if self.seqnum >= packetIn["seqnum"]:
        #if sequence number is bigger than seqnumber in the packet
        return False
        #we don't need to update
    self.seqnum = packetIn["seqnum"]
        #if sequence number is not bigger than the one in the packet,then update it
    if self.nbcost == packetIn["nbcost"]:
        #if sequence neighbor equals to that in the packet
        return False
        #we don't need to update it
    if self.nbcost != packetIn["nbcost"]:
        #if not equals
        self.nbcost = packetIn["nbcost"]
        #update it
        return True
```

b) DV 方法理解

DV 方法中，基于 bellman-fordman 方程进行距离向量更新时，其中 self.linksCost[src]是否可以换成 self.routersCost[src]?

不行。首先更换之后我们运行正常、删减、增加的代码，发现结果不太对。例如下图，



根据 handlenewlink 函数中的 linkscost 赋值方法，a-b 的 linkscost 会更新成 2。这样获得的 A-C 路径才是最短的，但是如果是 routerscost 的话，A-B 的距离就会取为 3，这样得到的 a-c 的距离就不是真正的最短路径了。