实验三 网络层路由实验

1. 实验目的

（1）理解和掌握链路状态法和距离向量法的实现原理和区别；
（2）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；
（3）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；
（4）初步掌握编程实现路由选择算法的能力。

2. 实验内容

（1） 理解实验原理，熟悉网络仿真器和图形化交互式界面；
（2） 阅读实验提供代码，掌握网络仿真器运行原理，理解网络仿真器中数据包、链路、客户端和路由器的具体实现；
（3） 阅读并补全链路状态法的关键代码，确定算法原理和代码实现的对应关系，掌握 Dijkstra 算法在其中的应用；
（4） 在网络仿真器中运行链路状态法，对比链路状态正常、链路故障和链路新增三种情况时路由路径和路由表变化；
（5） 在网络仿真器中运行距离向量法，对比链路状态正常、链路故障和链路新增三种情况时路由路径和路由表变化；
（6） 记录并对比链路状态法和距离向量法收敛速度，并分析实验现象。

3. 实验原理

路由是网络层至关重要的功能，需要确定从发送方到接收方所采用的一个“好”的路径，具体体现在成本最低、速度最快、开销最小等标准，为方便表述，下文统一用开销最小作为“好”的路径标准。具体而言，网络通过路由协议事先约定路由信息的发送过程的规定和标准，进行路由信息交换。常用的路由协议包括路由信息协议（RIP）、开放式最短路径优先协议（OSPF）和边界网关协议（BGP）等。路由选择算法则在路由协议中起着至关重要的作用，采用的路由选择算法决定了寻找最终路径的结果。常用的路由选择算法包括链路状态法（Link State，LS，对应 OSPF 协议）和距离向量法（Distance Vector，DV，对应 RIP 协议），本次实验将针对上述两种路由选择算法进行实现，并根据实验现象对比分析。

链路状态法的基本原理是节点之间传递链路状态和开销等信息，基于收到的链路状态信息，每个节点能够找到通往任意节点的最小开销路径。特别地，网络通过可靠洪泛（Reliable Flooding）保证所有节点都能得到来自其他节点的链路状态信息：一个节点沿着所有与其直接相连的链路把其链路状态信息发送出去，接收到这个信息的每个节点再沿着所有与它相连的链路转发出去，这个过程一直持续直到这个信息到达网络中所有节点。下面介绍一种路由器中常见的链路状态法实现方式：在具体实现过程中，每个节点通过创建链路状态包（Link-State Packet，

LSP）进行信息传递，LSP 包含如下信息：

- 创建 LSP 的节点 ID；
- 该节点的邻居列表，以及与每个邻居的链路开销；
- 序列号
- 数据包生命周期（Time To Live，TTL）

前两项用于路由计算，后两项则用于保证可靠的数据包洪泛过程。在路由器节点拥有来自其他节点的 LSP 信息后，会利用 Dijkstra 算法计算到达每个目的节点的最佳路由。特别地，路由器在实际过程中通过一种前向搜索算法（Forward Search），从它收集的 LSP 中直接计算路由表：在该算法实现中，每个路由器维护两张表，试探表（Tentative）和证实表（Confirmed）。每张表中有多条记录，每条记录的形式为<Destination, Cost, NextHop>（目的地，开销，下一跳）。算法如下：

（1） 当前路由器用自身节点初始化证实表中一条记录，这条记录开销为 0；

（2） 将前一步中加入证实表的那个节点称为 Next 节点，选择它的 LSP；

（3） 对于 Next 节点的每个邻居节点（Neighbor），计算达到这些邻居节点的开销（Cost），即从当前路由器节点到 Next 节点和再从 Next 节点到 Neighbor 节点的开销总和；

a) 如果 Neighbor 节点当前既不在证实表中，也不在试探表中，就把<Neighbor, Cost, NextHop>记录加入到试探表中，其中 NextHop 是当前路由器节点到 Next 节点所经的节点；

b) 如果 Neighbor 节点当前在试探表中，且开销小于当前登记在表中的开销，那么用记录<Neighbor, Cost, NextHop>替换当前记录，其中 NextHop 是当前路由器节点到 Next 节点所经的节点；

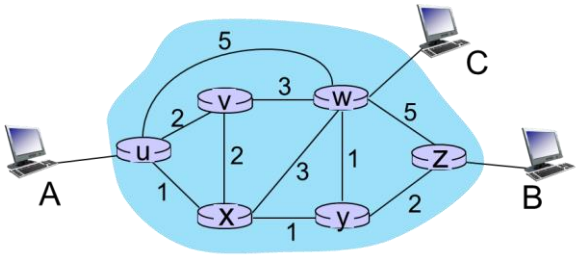
c) 如果试探表为空，则停止。否则，从试探表中挑选开销最小的记录，移入证实表，转（2）继续执行。

距离向量法的基本原理是在每个路由器节点构建一个距离向量，其中包含与所有其他路由器节点的开销，并将其距离向量传递给邻居节点。网络在运行距离向量法时，节点之间互相交换其所拥有的距离向量，并更新自身路由表，当所有节点获取一致的路由信息时，则算法收敛。每个节点仅知道自己路由表的内容，即由自身发送数据包到其他节点的距离和下一跳节点。

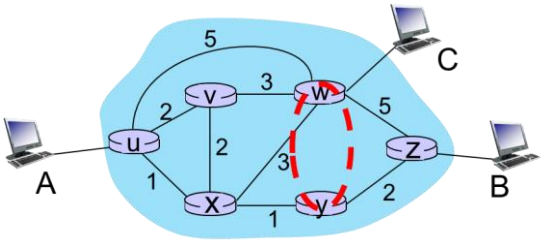
对比两种路由选择算法，在链路状态法中，每个节点和其余节点都进行通信（通过 LSP），但节点只告诉其他节点与自己直接相连链路的链路状态；在距离向量法中，每个节点只和与自己直接相连的节点进行通信，但节点把自己到所有结点的距离都告诉它们。此外，链路状态法是一种基于全局信息的路由选择算法，而距离向量法是一种基于局部信息的路由选择算法。此外，本次实验还涉及到链路状态改变（链路新增、链路故障）等情况，相关原理如毒性逆转（Poison Reverse）和水平分割（Split Horizon）等技术可参考课堂讲授知识和推荐教材。

本次实验已经通过 Python 搭建了一个网络仿真器，抽象了真实网络的许多细节，使同学们可以专注于具体路由选择算法的理解和掌握。具体地，我们在网络仿真器搭建了如下图（a）所示的网络结构，包含 3 个客户端 A、B、C，6 个路由器 u、v、w、x、y、z，链路上的数字表示链路开销。我们将实现和调用路由选择算法，找出客户端之间的最小开销路径，并记录路由信息。进一步地，我们设置该网络在指定时间内发生链路故障（如下图（b）所示）或链路新增（如下图（c）所示），并通过运行相应的路由选择算法，结合网络仿真器的可视化界面，

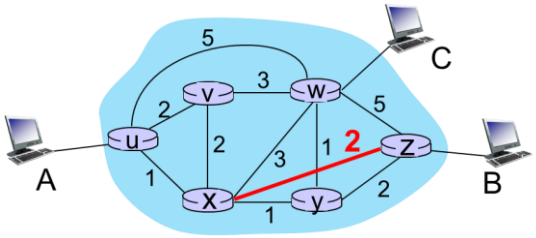
加深对上述情况处理原理和实现的理解。



（a）链路状态正常



（b）链路移除



（c）链路新增

4. 实验环境和操作流程

4.1 实验环境

（1） 为方便实验操作，本次实验需要预先安装虚拟机软件（Windows：VMware Workstation；Mac：VMware Fusion）（**实验 2 已经安装，则可跳过**），载入课程提供的环境配置完毕的虚拟机配置文件（.vmx 文件），在虚拟机中完成实验（**实验 3 提供的虚拟机环境与实验 2 不同，需要重新下载并载入**）。

- VMware Workstation 下载链接：
  https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html
- VMware Fusion 下载链接：
  https://cloud.tsinghua.edu.cn/f/b0fb45ffd8724c1ba6a0/?dl=1
  https://www.vmware.com/products/fusion/fusion-evaluation.html
- **课程实验提供的虚拟机配置文件下载链接：**
  https://cloud.tsinghua.edu.cn/f/a000bd5d1bb54bd9aa72/

（2） 下载相关配置文件 Exp3_NEW.zip 解压后，在 VMware 主界面中点击左上角“文件(F)”，在弹出下拉框中点击“打开(O)”，选择并打开下载的虚拟机配置文件（.vmx 文件），之后根据提示选择.vmdk 文件，点击“开启虚拟机”，选择“我已复制该虚拟机”，完成仿真环境搭建，若弹出“该虚拟机似乎正在使用中”提示，选择获取所有权。注意：使用 VMware Fusion 不要根据界面提示选择虚拟机，而需要直接双击.vmx 文件打开。

（3）启动虚拟机后，将进入 Ubuntu 系统。用户名：cn，密码：12345678，桌面

存放本次实验相关文件。可选择系统自带的 Visual Studio Code（VS Code）编写和调试代码。开启 VS Code 卡慢，可在 Terminal 输入下面命令启动 VS Code：

code --disable -gpu

（5）本次实验的关键文件和代码及作用参见下表。**注意：本次实验只需要修改 LSrouter.py 中 calPath 函数的代码，其它代码都已经写好（但需要阅读和理解）。**

表1 实验主要文件及功能

文件名	功能及说明
1_net.json	构建图（a）中网络的数据文件
1_net_remove.json	构建图（b）中网络的数据文件
1_net_add.json	构建图（c）中网络的数据文件
visualize_network.py	根据.json 文件构建网络，为用户 client 和路由器 router 开启多个线程，跟踪链路状态，执行相应的路由选择算法
packet.py	定义网络中 client 和 router 发送的数据包的 Packet 类
link.py	定义 router 和 client 之间的链路 Link 类
router.py	定义 router 的基类（Base Class）Router 类
client.py	定义周期性发送 traceroute 数据包[1]的用户 Client 类
LSrouter.py	从 Router 继承的基于链路状态法实现的 LSrouter 类
DVrouter.py	从 Router 继承的基于距离向量法实现的 DVrouter 类
LSP.py	定义链路状态法中的 LSP 类

4.2 网络仿真器介绍

如前所述，本次实验已经通过 Python 搭建了一个网络仿真器，抽象了真实网络的许多细节，使同学们可以专注于具体路由选择算法的理解和掌握。在进入虚拟机中后，打开 Terminal，进入 Project 文件夹通过以下命令可以打开采用指定算法和指定网络的仿真界面：

python visualize_network.py <Filename> <Alg>

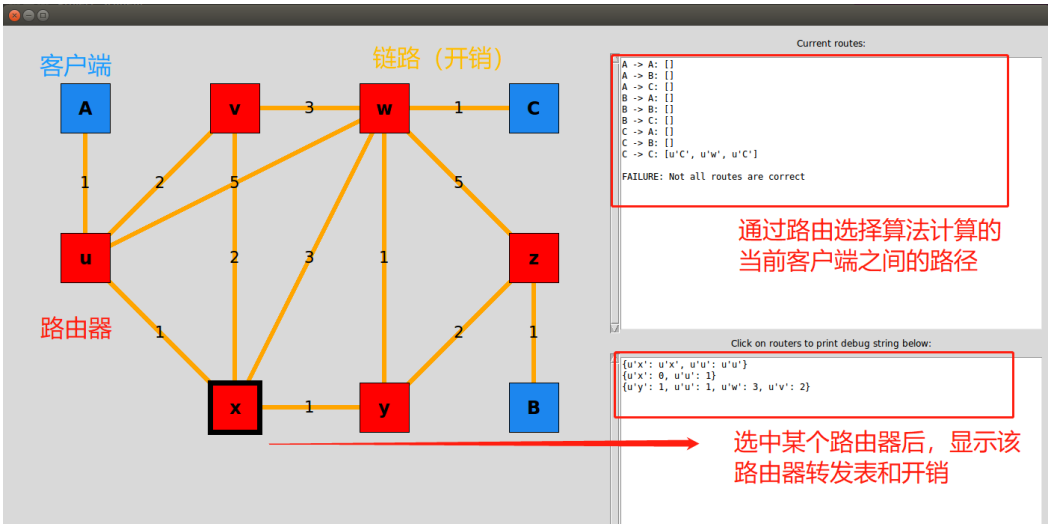
其中参数和可选值可参考下表：

参数	可选值
Filename	1_net.json，1_net_add.json，1_net_remove.json
Alg	LS，DV（分别对应链路状态法和距离向量法）

仿真界面如下图所示，其中，红色方块代表 router，蓝色方块代表 client，链

[1] traceroute 数据包是用于追踪数据包在网络上传输时的全部路径包括经过的路由器集合，在本实验中为网络仿真器可视化和校验服务，并不是路由协议中的数据包，不需要关注，更多内容可参考 https://zhuanlan.zhihu.com/p/404043710。

路上的数字代表相应的 cost。在运行过程中，链路上转移的蓝色小方块和红色小方块分别代表 client 发出的数据包（traceroute packet）和 router 发出的路由包（routing packet）。在实验过程中可通过选中路由器，结合右侧界面，查看当前客户端之间的路径和路由器相关信息。**注意：距离向量法已经完整实现，实验开始时可指定 DV 参数值，用于了解整个网络仿真器。**



5. 实验流程

5.1 网络仿真实现和理解

（1） 网络仿真器实现的整体过程如下：

a) 运行 visualize_network.py，启动网络仿真器；

b) 网络仿真器通过提供的数据文件，解析构建网络所需的拓扑结构、链路信息、客户端信息和路由器信息，并调用图形化界面实现函数搭建相应网络进行可视化；

c) 网络仿真器调用指定路由选择算法中的功能函数，在每个路由器中依据算法实现计算路由表和转发表等信息，并完成接收发数据的过程，其中路由计算过程中涉及到路由数据包（routing packet）的传输；

d) 网络仿真器周期性地让客户端发送 traceroute packet，进而实现对网络中当前路径的追踪，进一步调用图形化界面实现函数，在窗格中显示信息。当路径计算不完全正确时，右上角界面框会提示“Not all routes are correct!”，当路径计算完全正确时，右上角界面框会提示“All routes correct!”。

（2） 阅读关键代码，理解网络仿真器中数据包、链路、客户端和路由器的具体实现。这里给出相关代码文件及其主要函数的功能：

- packet.py：定义了网络传输中的数据包类，包含了定义数据包信息和类型的功能。下表介绍了 Packet 类的主要函数及功能：

函数	功能
__init__(self, kind, srcAddr, dstAddr, content)	初始化函数，定义了数据包的类型、源和目的地地址以及包内容

isTraceroute(self)	判断数据包是否是 traceroute 数据包
isRouting(self)	判断数据包是否是路由数据包
getContent(self)	获取数据包内容

- link.py：定义了链路类，对应路由器和客户端之间的链路，包含了收发数据包的功能。下表介绍了 Link 类的主要函数及功能：

函数	功能
__init__(self, e1, e2, l12, l21, latency)	初始化函数，定义了链路的两端节点和链路延迟
send(self, packet, src)	发送数据包的函数
recv(self, dst, timeout=None)	接收数据包的函数
changeLatency(self, src, c)	更改链路延迟的函数

- router.py：定义了路由器基类，包含了收发数据包、解决链路故障等功能函数。下表介绍了 Router 类的主要函数及功能：

函数	功能
__init__(self, addr, heartbeatTime=None[2])	初始化函数，定义了路由器的地址、端口对应的链路等
changeLink(self, change)	存储链路状态改变的信息
addLink(self, port, endpointAddr, link, cost)	将新加入的链路与路由器端口关联，并调用 handleNewLink 函数处理链路新增情况
removeLink(self, port)	将断开链路与路由器端口解绑，并调用 handleRemoveLink 函数处理故障链路情况
runRouter(self)	运行路由器的主要函数，监听链路状态改变信息并调用 addLink 函数或 removeLink 函数进行处理，调用 handlePacket 函数处理路由器收到的数据包
send(self, port, packet)	通过指定端口发送数据包
handlePacket(self, port, packet)	处理数据包的函数，由继承类实现
handleNewLink(self, port, endpoint, cost)	处理链路新增的函数，由继承类实现
handleRemoveLink(self, port)	处理链路断开的函数，由继承类实现
debugString(self)	用于网络仿真调试的函数，输出指定字符串信息

- client.py：定义了客户端类，包含了收发数据包等功能函数。特别地，客户端周期性地发送 traceroute 数据包实现网络路由追踪，用于网络仿真器可视化和校验。下表介绍了 Client 类的主要函数及功能：

函数	功能
__init__(self, addr, allClients, sendRate, updateFunction)	初始化函数，定义了客户端地址和发送速率等基本信息
changeLink(self, change)	添加客户端和路由器之间的链路
handlePacket(self, packet)	处理数据包的函数，主要对 traceroute 包操作
sendTraceroutes(self)	发送 traceroute 包的函数，给网络中每个客户端发送 traceroute 包，追踪客户端之间的数据转发路径
handleTime(self, timeMillisecs)	根据系统时间周期性发送 traceroute 包
runClient(self)	运行客户端的主要函数，调用 handlePacket 函数处理收

[2] 路由信息应该至少每隔 heartbeatTime 毫秒由这个路由器发送一次。

	到的数据包，调用 handlTime 函数发送 traceroute 包

- LSP.py：定义了链路状态数据包类。在初始化函数__init__(self, addr, seqnum, nbcost)定义了发送该数据包的路由器地址，序列号和相邻节点链路开销；在更新函数 updateLSP(self, packetIn)中根据接收数据包更新 LSP。

5.2 链路状态法理解与实现

（1） 阅读并理解链路状态法实现中的函数功能：

- LSrouter.py：定义了实现和运行链路状态法的 LSrouter 路由器类，从 Router 类继承。下表介绍了 LSrouter 类的主要函数及功能：

函数	功能
__init__(self, addr, heartbeatTime)	初始化函数，请结合链路状态法原理，在下文中给出该函数中定义变量的含义
handlePacket(self, port, packet)	处理数据包的函数，用于处理和更新收到的 LSP
handleNewLink(self, port, endpoint, cost)	处理链路新增的函数，将新增链路信息加入到自身的 LSP，并封装到数据包中洪泛广播
calPath(self)	基于 Dijkstra 算法实现的路径计算函数，请结合链路状态法原理，在下文中对该部分代码注释，并补全关键代码
handleRemoveLink(self, port)	处理链路断开的函数，需要更新 LSP 和重新计算路由，请结合 LSP 可靠洪泛内容，在下文中对该部分代码注释
handleTime(self, timeMillisecs)	根据系统时间周期性更新路由

（2） 结合链路状态法原理，给出初始化函数中变量的作用（按行填写）：

代码	变量作用

`def __init__(self, addr, heartbeatTime):`
`    """class fields and initialization code here"""`
`    Router.__init__(self, addr)  # initialize superclass`
`    self.routersLSP = {}  ###`
`    self.routersAddr = {} ###`
`    self.routersPort = {} ###`
`    self.routersNext = {} ###`
`    self.routersCost = {} ###`
`    self.seqnum = 0 ###`
`    self.routersLSP[self.addr] = LSP(self.addr, 0, {})`	

（3） 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（按行填写），并补全关键代码：

代码	注释

```
def calPath(self):
    # Dijkstra Algorithm for LS routing
    self.setCostMax()
    # put LSP info into a queue for operations
    Q = PriorityQueue()
    for addr, nbcost in self.routersLSP[self.addr].nbcost.items():
        Q.put((nbcost, addr, addr))
    while not Q.empty():
        Cost, Addr, Next = Q.get(False)
        if Addr not in self.routersCost or Cost < self.routersCost[Addr]:
            ### TODO: Add two lines code to update Cost and Next for Addr


            if Addr in self.routersLSP:
                for addr_, cost_ in list(self.routersLSP[Addr].nbcost.items()):
                    Q.put((cost_ + Cost, addr_, Next))
        pass # remember to delete this command after completing code
```

（4） 基于正常网络即 1_net.json，运行链路状态法进行路由选择，验证实现正确性。记录链路状态法计算的客户端之间的最小开销路径，以及路由器 x 的转发表和 LSP 信息：

a) 客户端之间的最小开销路径：

源客户端-目的客户端	最小开销路径
A → A	
A → B	
A → C	
B → A	
B → B	
B → C	
C → A	
C → B	
C → C	

b) 路由器 x 的转发表[3]：

目的节点	下一跳转发节点	开销
A		
B		
C		
u		
v		
w		
x		
y		
z		

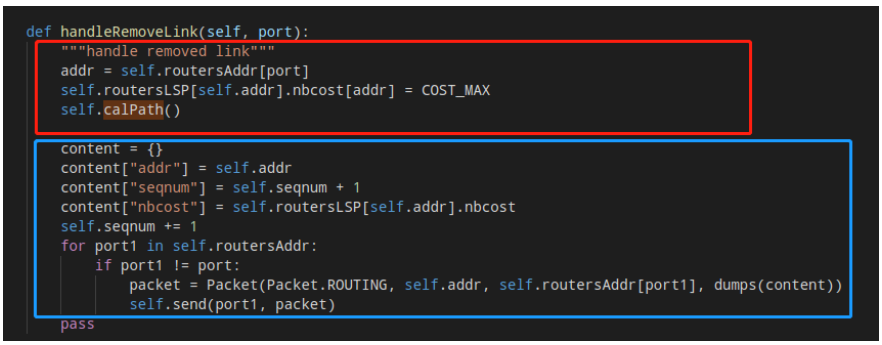
c) 路由器 x 的 LSP 信息记录。

5.3 链路状态改变实验

（1） 阅读和理解链路状态法中处理链路断开的函数 handleRemoveLink，在下

[3] 就路由原理而言，转发表中仅涉及路由器节点的转发路径，不涉及客户端。这里为方便理解，实验中同时给出了路由器到客户端的转发路径。后面记录表格同此处一致。

文中对该部分代码注释（按行注释）：

代码	注释
	

```
def handleRemoveLink(self, port):
    """handle removed link"""
    addr = self.routersAddr[port]
    self.routersLSP[self.addr].nbcost[addr] = COST_MAX
    self.calPath()

    content = {}
    content["addr"] = self.addr
    content["seqnum"] = self.seqnum + 1
    content["nbcost"] = self.routersLSP[self.addr].nbcost
    self.seqnum += 1
    for port1 in self.routersAddr:
        if port1 != port:
            packet = Packet(Packet.ROUTING, self.addr, self.routersAddr[port1], dumps(content))
            self.send(port1, packet)
    pass
```

（2） 基于链路故障（移除）网络即 1_net_remove.json，运行链路状态法进行路由选择，验证实现正确性。记录链路状态法计算的客户端之间的最小开销路径，以及路由器 x 的转发表和 LSP 信息：

a) 客户端之间的最小开销路径：

源客户端-目的客户端	最小开销路径
A → A[4]	
A → B	
A → C	
B → A	
B → B	
B → C	
C → A	
C → B	
C → C	

b) 路由器 x 的转发表：

目的节点	下一跳转发节点	开销
A		
B		
C		
u		
v		
w		
x		
y		
z		

c) 路由器 x 的 LSP 信息记录。

（3） 基于链路新增网络即 1_net_add.json，运行链路状态法进行路由选择，验证实现正确性。记录链路状态法计算的客户端之间的最小开销路径，以及路由器 x 的转发表和 LSP 信息：

a) 客户端之间的最小开销路径：

[4] 注意，网络仿真器的实现逻辑是客户端发送 traceroute 包给路由器，路由器告知路径。因此即使是从客户端自身作为源和目的地，也会返回经过路由器的路径，这种情况实际网络中是不需要经过路由器。

源客户端-目的客户端	最小开销路径
A → A	
A → B	
A → C	
B → A	
B → B	
B → C	
C → A	
C → B	
C → C	

b) 路由器 x 的转发表：

目的节点	下一跳转发节点	开销
A		
B		
C		
u		
v		
w		
x		
y		
z		

c) 路由器 x 的 LSP 信息记录。

（4） 通过比较链路状态改变和正常状态时最小开销路径和路由器转发表、LSP 的信息，理解链路状态法处理链路状态改变的过程。

5.4 距离向量法理解与实验

（1） 阅读并理解距离向量法实现中的函数功能

- DVrouter.py：定义了实现和运行距离向量法的 DVrouter 路由器类，从 Router 类继承。下表介绍了 DVrouter 类的主要函数及功能：

函数	功能
__init__(self, addr, heartbeatTime)	初始化函数
handlePacket(self, port, packet)	处理数据包的函数，用于更新和发送自己的距离表
updateNode(self, content)	更新路由器距离表的函数
handleNewLink(self, port, endpoint, cost)	处理链路新增的函数，根据新增链路信息更新距离向量
handleRemoveLink(self, port)	处理链路断开的函数，根据断开链路信息更新距离向量
handleTime(self, timeMillisecs)	根据系统时间周期性更新路由

（2） 【选做】理解 DV 实现过程，并写出从代码中理解出来的实现流程图。

5.5 路由选择算法效率比较

（1）记在不同网络拓扑结构上执行不同路由选择算法，通过手机秒表等工具，记录从开始运行到第一次出现路由正确的收敛时间

收敛时间（s）	链路正常	链路故障	链路新增
距离向量法			
链路状态法			

6. 实验考核

（1） 理解网络仿真器实现原理，正确运行已实现好的距离向量法；
（2） 分析链路状态法实现逻辑，补全关键代码并正确运行；
（3） 在三种链路状态的网络上运行链路状态法和距离向量法，理解算法实现原理，比较算法效率；
（4） 网络学堂中提交实验报告，报告内容合乎逻辑，表达清晰，有实验过程记录、截图及思考题回答。提交截止时间为 2022 年 10 月 21 日。

7. 实验思考题

(1) 请给出你对 LSP.py 中 LSP 更新函数的执行过程的理解，参见下面代码。

```
8       def updateLSP(self, packetIn):
9           if self.seqnum >= packetIn["seqnum"]:
10              return False
11          self.seqnum = packetIn["seqnum"]
12          if self.nbcost == packetIn["nbcost"]:
13              return False
14          if self.nbcost != packetIn["nbcost"]:
15              self.nbcost = packetIn["nbcost"]
16              return True
17
```

(2) 在距离向量法实现中，下面代码基于 Bellman-Ford 方程进行距离向量更新时，其中的 self.linksCost[src]能否换成 self.routersCost[src]，请说明原因。

```
# # choose the less cost or new info
if dst in self.routersCost:
    if src in self.routersCost:
        if (self.routersCost[dst] > self.linksCost[src] + cost) or (self.routersNext[dst] == src and src != dst):
            self.routersCost[dst] = self.linksCost[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]
```

8. 实验参考资料

（1）Python 教程：https://www.liaoxuefeng.com/wiki/1016959663602400