《通信与网络》实验三

网络层路由实验

2022年10月

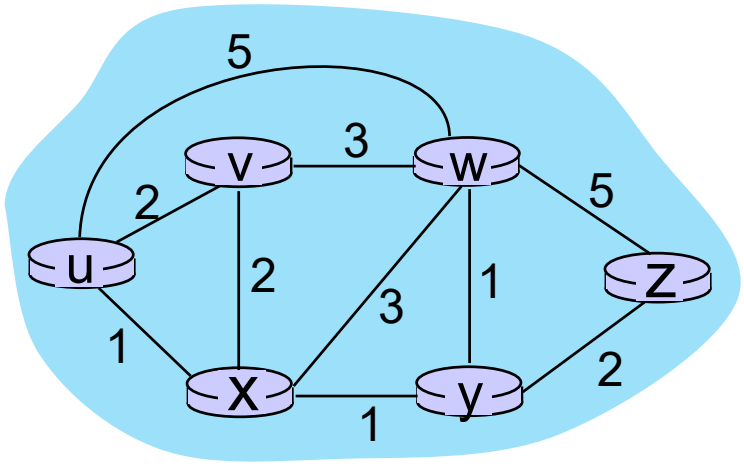
目录

- 路由选择算法回顾
- 网络仿真器介绍
- 实验内容介绍

一、路由选择算法回顾

路由选择

- **目标：寻找从发端到收端的一个“好”的路径**
  - 路径：数据报从发端到收端经过的路由器序列
  - “好”：最低成本/最快/最不拥挤/最小开销



例：$c(u,w)=5, c(u,z)=\infty$

- 图$G=(N,E)$
  - N为点（路由器）集合
  - E为边（链路）集合
  - 对边(x,y)定义其开销$c(x,y)$
  - 路径：节点序列$(x_1,x_2,\cdots,x_p)$

路由选择算法

- **集中式：所有路由器有完整的网络拓扑及链路开销信息**
  - 链路状态法：计算从源节点到其他所有节点的最小开销路径，根据结果确定源节点的转发表
  - 典型算法：Dijkstra算法
  - 典型协议：OSPF
- **分布式：与邻居交换信息，迭代计算路由器初始只有与邻居间的链路信息**
  - 距离向量法：每个节点只与邻居交换信息，维护自身的距离向量，确定转发表
  - 典型算法：Bellman-Ford算法
  - 典型协议：RIP

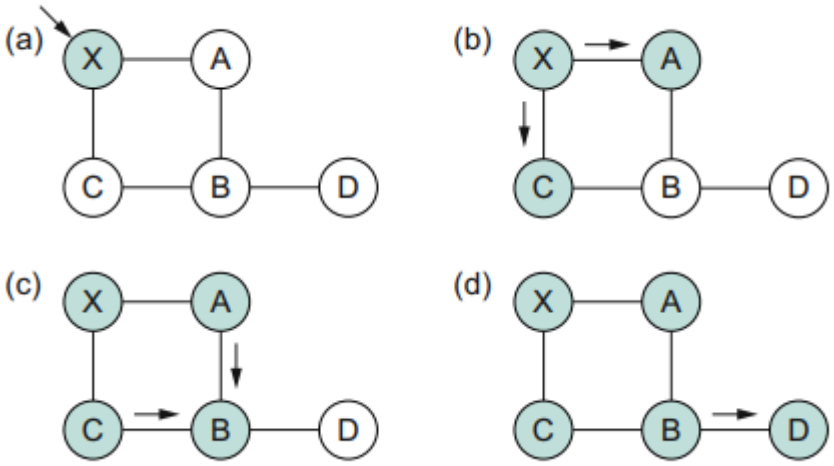
链路状态法

- **实现原理**
  - 通过可靠洪泛（Reliable Flooding）在网络中通过链路状态包传递链路信息，节点根据收到的链路状态信息计算路由
- **链路状态包（Link-State Packet，LSP）**
  - 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）

链路状态法

```
def updateLSP(self, packetIn):
    if self.seqnum >= packetIn["seqnum"]:
        return False
    self.seqnum = packetIn["seqnum"]
    if self.nbcost == packetIn["nbcost"]:
        return False
    if self.nbcost != packetIn["nbcost"]:
        self.nbcost = packetIn["nbcost"]
        return True
```

- **可靠洪泛：保证LSP正确到达所有节点**
- **实现示例：**
  - LSP在下图网络中传播，每个节点执行该算法



- **节点X收到来自节点Y的LSP：**
  - **IF 节点X尚未存储节点Y的LSP：**
    - **存储该LSP；**
  - **ELSE**
    - **IF 收到LSP序列号 > 存储LSP序列号：**
      - **保存并替换旧LSP，将其发送给Y以外的其他相邻节点**
    - **ELSE：**
      - **丢弃Y发来的LSP**

链路状态法

- **路由计算（课堂讲授）**
  - 节点根据收到的所有LSP信息，构造网络对应的图
  - 利用Dijkstra算法计算到其他所有节点路径

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5          then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9      find w not in N' such that D(w) is a minimum
10     add w to N'
11     update D(v) for all v adjacent to w and not in N' :
12         D(v) = min ( D(v),  D(w) + c(w,v) )
13     /* new least-path-cost to v is either old least-cost-path to v or known
14     least-cost-path to w plus direct-cost from w to v */
15 until all nodes in N'
```

符号含义

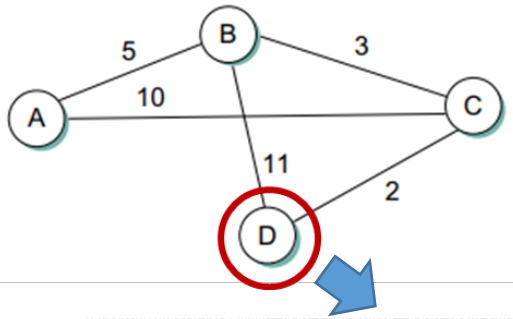
- N'：从源到v的最低开销路径已知的v集合
- $D(v)$：当前对源节点到节点v的最低开销路径的开销估计
- $p(v)$：从源到v最低开销路径上v的先导节点

链路状态法

- **路由计算（实际实现）**
  - 前向搜索（Forward Search）算法，Dijkstra思想
- **每个路由器维护试探表和证实表**
  - <Destination, Cost, NextHop>（目的地，开销，下一跳）

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。

链路状态法：前向搜索算法示例



步骤	证实表	试探表	注 释
1	(D,0,-)		因为D是证实表中惟一的新成员，所以观察它的LSP
2	(D,0,-)	(B,11,B) (C,2,C)	因D的LSP表明，我们可以以开销11通过B到达B，比表任何其他的路径都好，因此把它加入试探表中，同理C也加入
3	(D,0,-) (C,2,C)	(B,11,B)	把试探表中开销最小的记录C加入证实表中。接着，检查证实表中新的成员C的LSP
4	(D,0,-) (C,2,C)	(B,5,C) (A,12,C)	因为通过C到达B的开销是5，所以替换记录（B,11,B），C的LSP告诉我们可以以开销12到达A
5	(D,0,-) (C,2,C) (B,5,C)	(A,12,C)	因把试探表中开销最小的记录B加入证实表中，观察它的LSP
6	(D,0,-) (C,2,C) (B,5,C)	(A,10,C)	因为可以经过B以开销5到达A，所以替换试探表中的记录
7	(D,0,-) (C,2,C) (B,5,C) (A,10,C)		把试探表中开销最小的成员A移入证实表中，结束

链路状态法：前向搜索算法示例

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。

```
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))
```

试探表

Q

证实表

self.routersNext

&

self.routersCost

距离向量法

- Bellman-Ford方程
  - 若$D_x(y)$表示节点x到节点y的最低开销路径的开销，则有Bellman-Ford方程：

$$D_x(y) = \min_v [c(x,v) + D_v(y)]$$

对x的所有邻居v取最小值

x到v的直接链路开销

v到y的最低开销路径开销

  - $\boldsymbol{D}_x = [D_x(y): y \in N]$为节点$x$的距离矢量

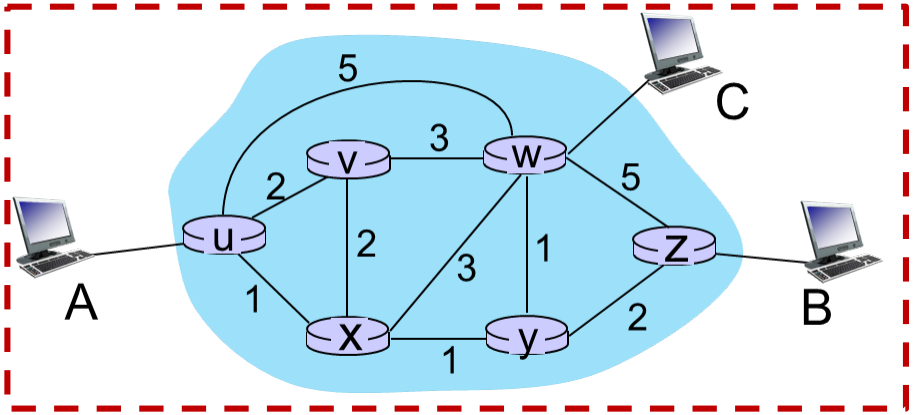
```
# # choose the less cost or new info
if dst in self.routersCost:
    if src in self.routersCost:
        if (self.routersCost[dst] > self.linksCost[src] + cost) or (self.routersNext[dst] == src and src != dst):
            self.routersCost[dst] = self.linksCost[src] + cost
            self.routersNext[dst]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

            # set COST_MAX as infinity
            if self.routersCost[dst] >  COST_MAX:
                self.routersCost[dst] =  COST_MAX
            return True, dst, self.routersCost[dst]
return None
```

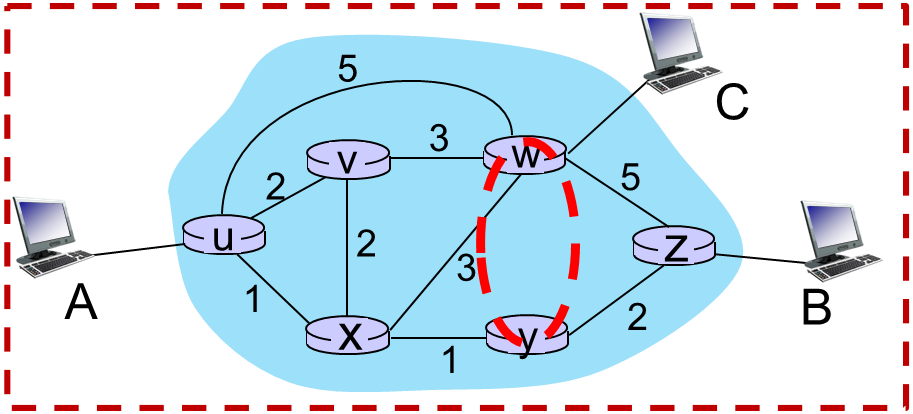
二、网络仿真器介绍

实验环境

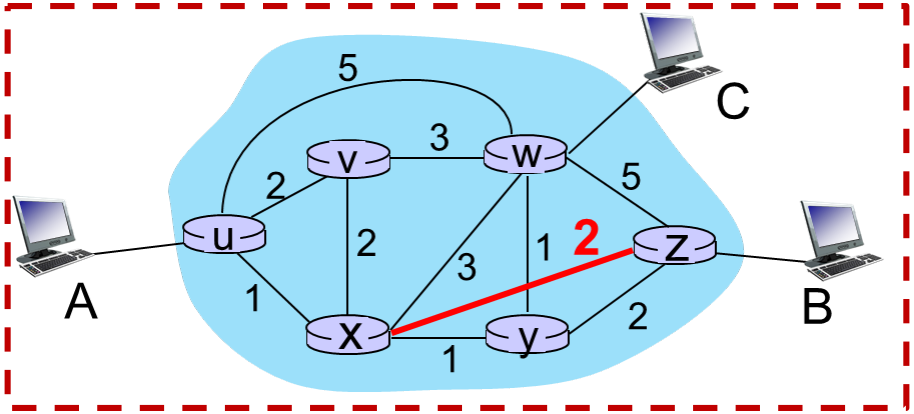
- 网络拓扑

链路正常





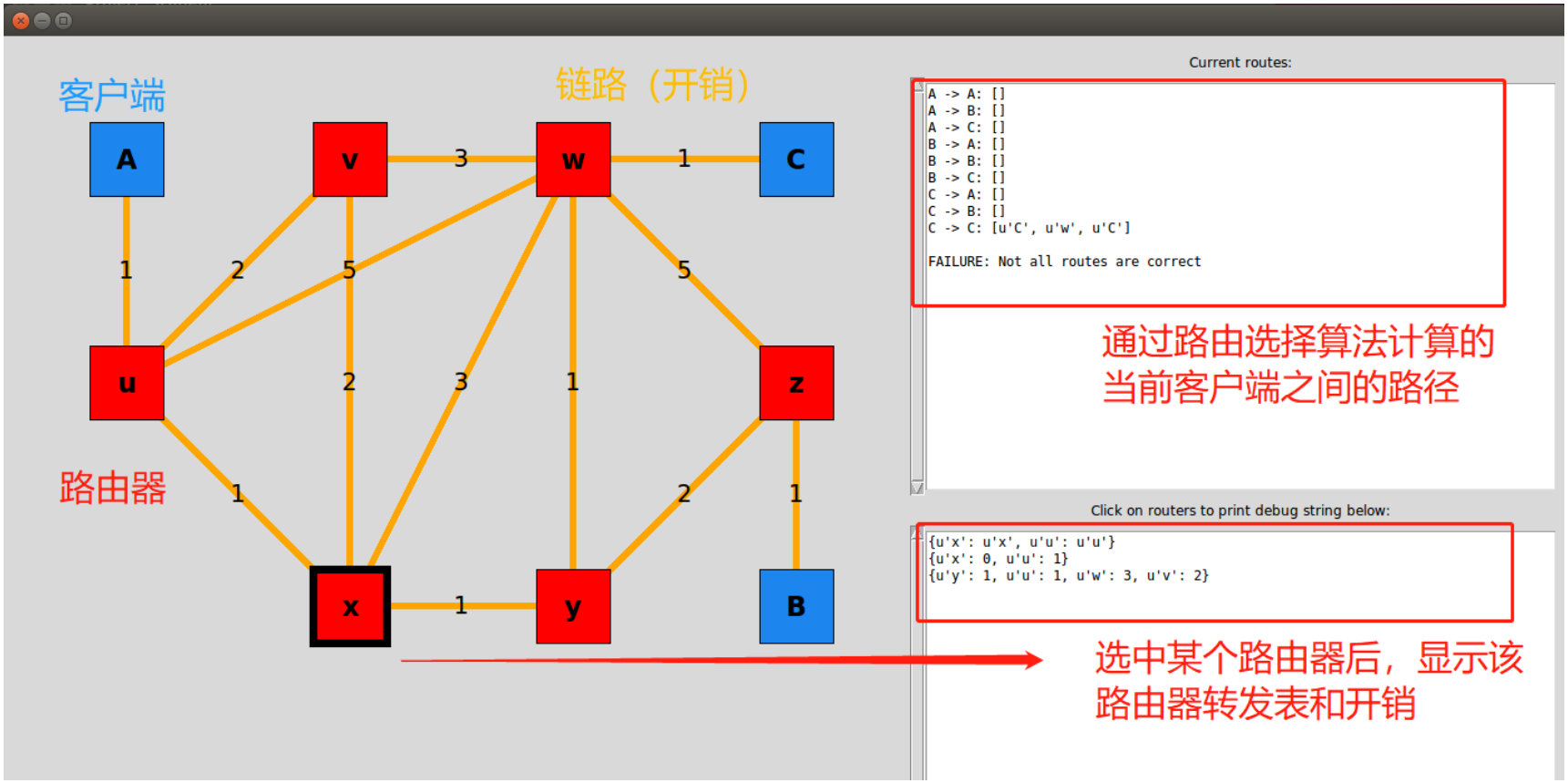
链路故障



链路新增

实验工具

- 图形化交互界面



三、实验内容介绍

1. 网络仿真实现和理解

- **基于课程提供的虚拟机实验**
  - 理解网络仿真器的整体实现过程

Packet

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

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

Client

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函数处理收到的数据包，调用handlTime函数发送traceroute包

Router

函数	功能
__init__(self, addr, heartbeatTime=None)	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

2. 链路状态法理解与实现

- 阅读并理解链路状态法实现中的函数功能

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物理含义
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	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

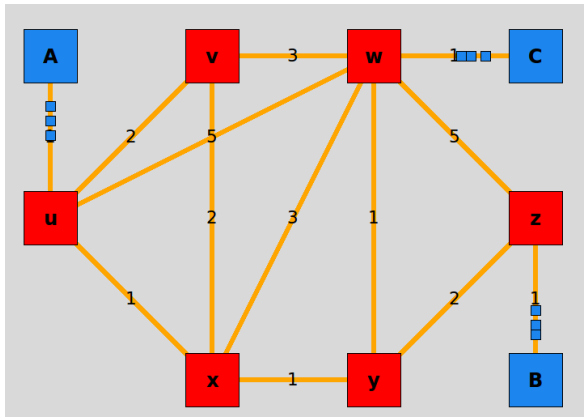
```python
def calPath(self):
    # Dijkstra Algorithm for LS routing
    self.setCostMax()
    # put LSP info into a queue for operations
    Q = PriorityQueue()
    for addr, nbcost in self.routersLSP[self.addr].nbcost.items():
        Q.put((nbcost, addr, addr))
    while not Q.empty():
        Cost, 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
```

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	

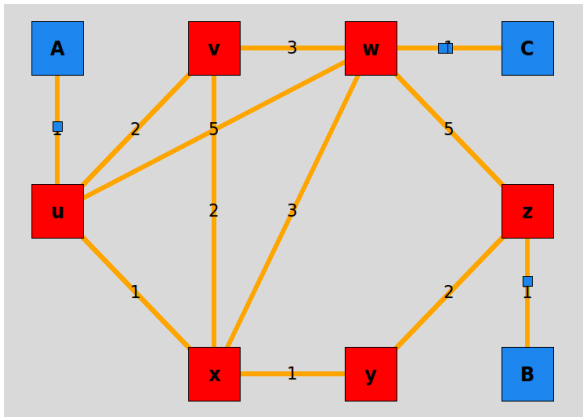
目的节点	下一跳转发节点	开销
A		
B		
C		
u		
v		
w		
x		
y		
z		

3. 链路状态改变实验

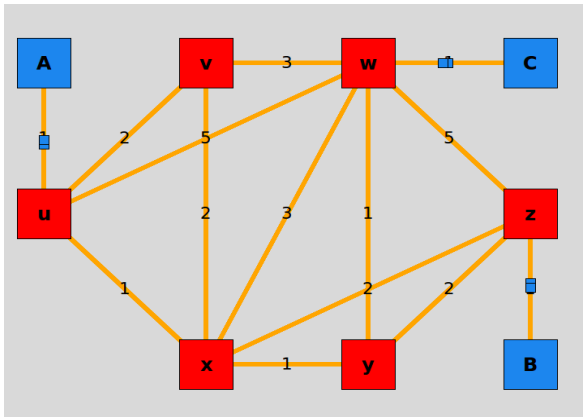
- 阅读并理解链路状态法解决链路状态改变的过程
- 记录客户端之间最小开销路径和路由器转发表变化



链路正常



链路故障



链路新增

4. 距离向量法理解与实验

- 【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。

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

```
def updateNode(self, content):
    """update node with routing packet"""
    data = loads(content)
    src = data["src"]
    dst = data["dst"]
    cost = data["cost"]

    if dst not in self.routersCost and dst != self.addr:
        if src in self.routersCost:
            self.routersCost[dst] = self.routersCost[src] + cost
            self.routersNext[dst] = src
            return True, dst, self.routersCost[dst]

    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

                if self.routersCost[dst] >  COST_MAX:
                    self.routersCost[dst] =  COST_MAX
                return True, dst, self.routersCost[dst]
    return None
```

5. 路由选择算法效率比较

- 记在不同网络拓扑结构上执行不同路由选择算法的收敛时间

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			

LS和DV算法的比较

- 报文复杂度（n个节点）
  - LS算法：共$O(n^2)$个报文
  - DV算法：邻居间交换报文；与收敛时间有关
- 收敛速度
  - LS算法：$O(n^2)$ 算法；可能振荡
  - DV算法：收敛时间不确定；可能遭遇无穷计数问题
- 鲁棒性：某路由器故障会发生什么？
  - LS算法：每个路由器仅计算自己的转发表
  - DV算法：可能传播错误的路径开销信息；
    转发表被其他节点使用：错误传播

6. 注意事项

- **编程语言和环境**
  - 提供Python代码和虚拟机环境

- **实验考核**
  - 提交实验报告至网络学堂
  - 实验报告需包括实验中的重要现象、思考题回答