虚拟路由Virtual Routing实验

实验要求

- 设计实验拓扑图，设定好链路的权值
- 使用软件方法模拟实现路由器的转发和选路功能，路由器之间周期性的交换路由消息
- 使用不同的转发算法以及不同的拓扑图，得到实验结果
- 在模拟的路由器拓扑中互相传递消息，测试连通性。

贡献

陈泓仰	邹元昊	周睿
整体框架搭建，路由器功能实现，适用于DV算法的发送报文程序实现，整合实验报告	报文类处理，路由表类处理，路由器转发表读取，DV算法实现，DV算法实验记录，debug	LS算法实现，LS连通性测试，LS算法实验记录

实验环境及运行

- 使用pycharm运行程序
- 先打开五个节点的NODE，等待路由表信息趋于收敛时，打开其中一个节点的test.py，先开receive端，再开另一个节点打开client端，发送报文。即可测试连通性

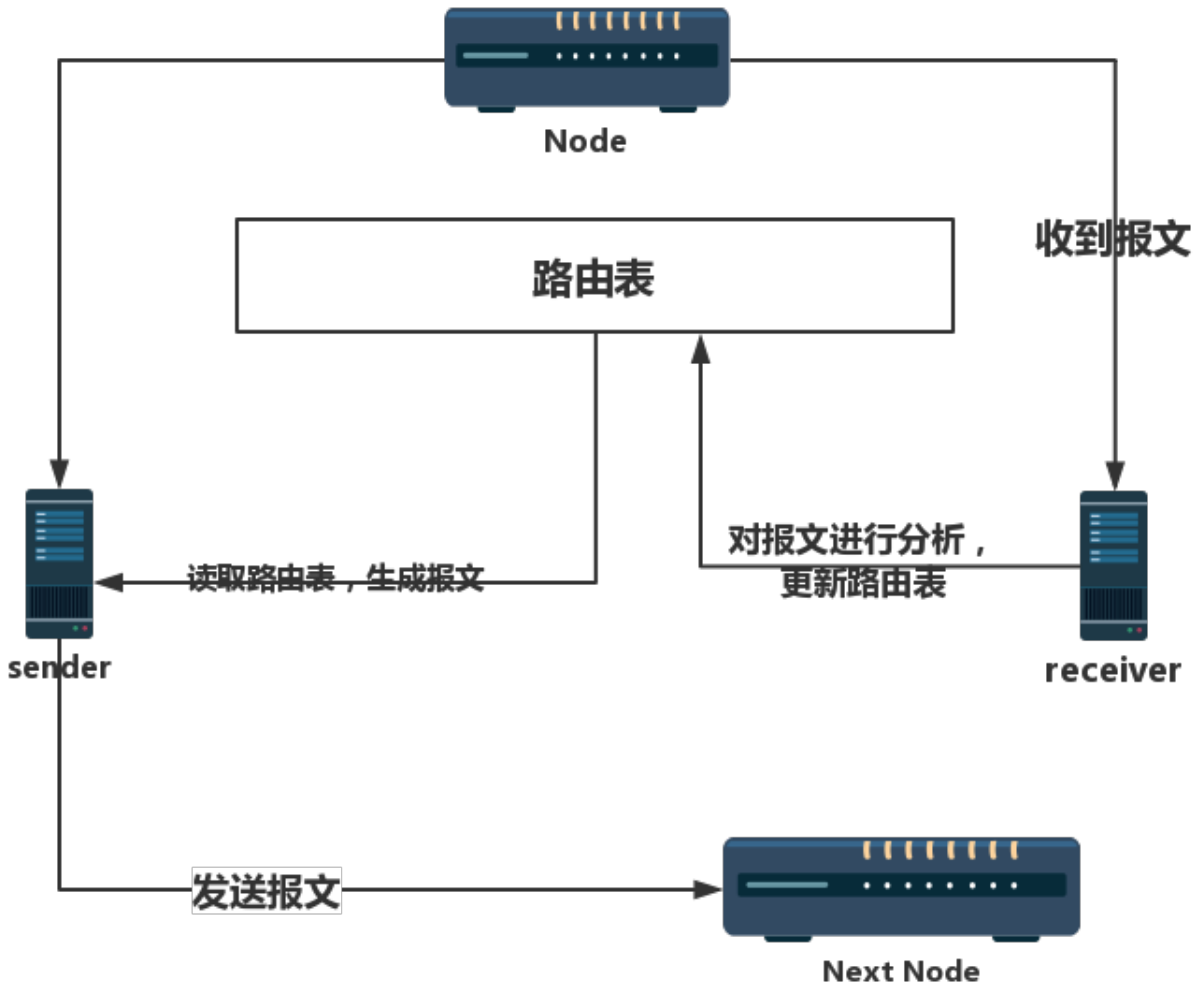
实现思路：

- 实验使用TCP连接来模拟网络层实现
- 自顶向下开发，先将整体的交换路由表信息的框架搭好，然后留出底层函数来实现路由选路算法
- 实验与上次实验实现方法大致相同，都是通过套接字来相互连接俩台主机，然后交换信息。而俩次实验的不同之处在于，由于我们需要选择下一台连接的主机，而且是动态的去选择最短路径，因此，我们需要进行大量的本地操作，不断修改本地路由表，得出一条最短路径。

具体实现：

整体框架：

- 由于我们需要频繁的进行路由器之间的数据交互（路由表的交换），因此设计一个方便传输以及方便进行本地操作的架构是非常必要的。
- 整体架构图：



路由器实现说明

路由表结构(class Route_info)：

1. 将路由表以及处理路由表的函数封装为路由表类，方便调用。
2. 为了能够简明清晰的记录下路由表的信息，我们采取如下的路由信息记录方式：

（源路由，目的地路由）	（权值，下一跳路由）

3. 采用这种方式记录路由信息，可以很方便的获取当前路由自己的表项，同时还能很方便的更新路由信息。

4. 路由表类函数说明：

```
class Route_info:#路由表类
    def __init__(self):新建路由表，同时将在本地存下路由表文件
    def init_add_neighbor(self, tup, tup_data):根据路由表后台邻居的信息，初始化拓扑结构信息
    def __get_map(self):获取路由表
    def shut_down(self,dest):某一个节点路由器宕机后处理路由表函数
    def __save_map(self):保存路由表
    def create_index(self,dest):根据路由表获取某一路由表项
    def update_route(self,cmd,index):更新路由表的算法实现
```

报文结构(class datagram)：

1. 为了方便路由器之间的路由表项的交换，我们参考IP协议的定义，舍去一些在这次实验中不需要的报文项，自己定义了一个用来传输信息的报文结构。结构如下：

COMMAND	IP	PORT	INDEX
选路算法类型	本机IP	本机端口	本机路由表

2. 同样的，为了方便对报文进行操作，我们将处理报文的函数封装为报文类。

3. 需要注意的是由于DS与LS算法实现方法不同，所以在报文中的index内容不同。对于DV算法来讲，只需要存下源路由是本机路由的相关表项即可，而对于LS算法，每台机子都需要知道全局链路状态，所以报文中的index是整个路由表。

4. 在生成了这个报文类，并进行初始化之后，将报文写入json文件中，然后转发到下一跳路由。下一跳路由得到这个json文件之后，按照自定报文结构，即可得到所需的信息（路由表）。使用这个方法，可以解决无法传输路由表类这个问题。

5. 报文类函数说明：

```
class datagram:
    def __init__(self, cmd = 0, IP = 0, PORT = 0, index = 0, json_name = ''):通过定义
    def create_datagram(self,nickname):将报文生成json文件
```

路由表后台数据说明（basic.py）：

1. 路由器初始化后保持不变的有：

    - a. 路由器自己的IP
    - b. 路由器用于交换路由表信息的PORT
    - c. 路由器与路由器之间用于转发连通程序报文的转发端口

- d. 路由器收连通程序之间收发数据的俩个端口

2. 由于我们本次实验是在一台机子上运行所有的路由器程序，路由器与路由器之间通过设定的不同端口相互连接。所以初始时我们只需要设置不同的端口即可。（同样的设置不同的IP也可以实现不同机子的连接，已经通过测试）

3. 路由器在运行的时候要根据拓扑结构更新路由表

4. 同时当某一节点路由器宕机时，其他路由器也要相应的维护自己的路由表。

DV算法实现重点函数说明：

- dv选路算法：

  update_route(self,cmd,index)

- 说明：

  - 函数需要的参数：cmd（判断什么类型的算法，只有DV算法，ls另外实现） index（从其他服务器得到的表项）

  - 这段代码实现的功能是，在路由器得到其他路由器传来的一条表项之后，对路由器本身的路由表进行修改

  - 首先路由器会先计算出，当前发送路由器A和这条表项的发送路由器B之间的长度是多少（这样方便计算）

  - 然后循环判断路由器中其他的dest，如果有A - dest的长度大于 A - B 加上B - dest的长度，则改变A - dest的长度

  - 最后在进行判断，是否自己的到的目的地的下一跳路由都是有效的（无效的为10000），因为有可能出现下一跳路由到目的地址的长度已经为10000的情况（就是目的地已经断开的情况），这样，我们将认为A - dest的长度也为10000

1. 代码：

```
def update_route(self,cmd,index):
        self.__get_map()
        if cmd == 'LS':
            pass
        elif cmd == 'DV':
            dest = None
            for value in index.keys():
                self.__map[value] = index[value]
                dest = value[0]
            addr = PC_IP
            length = self.__map[addr + dest][0]
            for value in index.keys():
                new_length = length + self.__map[value][0]
                if new_length < self.__map[addr + value[1]][0]:
                    self.__map[addr + value[1]] = (new_length,dest)
            for value in self.__map.keys():
                temp = self.__map[value][1]
                if value[0] == PC_IP and temp != value[1] and temp != PC_IP:
                    if self.__map[temp + value[1]][0] == 10000:
                        self.__map[value] = (10000,PC_IP)
        else:
            print('undifiend command name!!!')
        self.__save_map()
```

- 节点路由器宕机情况：

  shut_down(self,dest)

1. 说明：
   - 该函数实现的是从路由表中删除某条表项的功能，因为这个路由器已经和服务器断开了连接
   - 该函数需要的参数：dest是要删去的目标路由器的代号
   - 首先从本地读取该路由器的邻居，将他们放入neighbour_list中，再删去dest
   - 将起始地址为dest的表项全部都删除
   - 重置本机到neighbour_list中的参数的权值（和一开始的情况一样）不在列表中的置为10000
   - 然后运行更新路由表的算法，重置路由表。
   - 如果某条路由的下一跳路由，或者是目标地址是dest的话，将这些路由器的到dest的权值都置为10000
2. 代码：

```
def shut_down(self,dest):
        self.__get_map()
        neighbour_list = list(neighbour.keys())
        neighbour_list.remove(dest)
        delete_list = []
        for value in self.__map.keys():      #将dest从map中删除
            if value[0] == dest:
                delete_list.append(value)
        for value in delete_list:
            del self.__map[value]

        for value in self.__map.keys():                  #重置PC_IP所有权值
            if value[0] == PC_IP and value[1] in neighbour_list:
                tup = neighbour[value[1]]
                self.__map[value] = tup
            if value[0] == PC_IP and value[1] not in neighbour_list:
                if value[0] == PC_IP and value[1] == PC_IP:
                    tup = (0,value[0])
                    self.__map[value] = tup
                else :
                    tup = (10000,PC_IP)
                    self.__map[value] = tup
        for value in self.__map.keys():
            if self.__map[value][1] == dest:
                self.__map[value] = (10000,PC_IP)
        for another_temp in neighbour_list:              #更新路由表
            for value in self.__map.keys():
                if value[0] == PC_IP:
                    if self.__map[value][0] > self.__map[PC_IP + another_temp][0] +
                        tup = (self.__map[PC_IP + another_temp][0] + self.__map[anot
                        self.__map[value] = tup
        for value in self.__map.keys():
            if value[1] == dest:
                self.__map[value] = (10000,PC_IP)
        self.__save_map()
```

ping程序实现

- 由于这个程序时应用层上的程序，所以将它模拟为与路由器相连的某一个客户端。该程序与路由器进行信息传递时有自己报文协议。

msg	src ip	dest ip

- 实现思路，首先程序将报文通过路由器数据端口发送给第一条路由器，然后第一条路由器查询下一跳路由，将报文发给下一跳路由。然后每一跳路由都会检查报文里面设置的下一跳路由编号与当前路由编

号，若不相等则继续转发，若相等，则将报文发送给服务端，在服务端中进行显示。同时在每次转发报文的时候，该节点还会发送另外一个ping报文给路径上的前一个节点。最后在源节点上显示，路径上的所有路由器。

- 代码

init_recv(srcname,dataIP,dataPORT)

用来收取应用层数据报文，并将报文发送到下一跳路由

```
def init_recv(src_name,data_IP,data_PORT):
    addr = (data_IP,data_PORT)
    receive_socket = socket.socket()
    receive_socket.bind(addr)
    receive_socket.listen(10)
    print("trans....")
    newSocket, destAddr = receive_socket.accept()
    print("transed", destAddr)
    while True:
        datalength = calcsize('128s5s12s5s')
        data = newSocket.recv(datalength)
   decode_msg,decode_cmd,decode_src_ip,decode_dest_name = unpack('128s5s12s5s',data)
        msg = (decode_msg.decode('utf-8')).strip('\0')
        cmd = (decode_cmd.decode('utf-8')).strip('\0')
        src_ip = (decode_src_ip.decode('utf-8')).strip('\0')
        dest_name = (decode_dest_name.decode('utf-8')).strip('\0')
        list_dict = read_list(src_name)
        next_matric_name = get_next_matric(list_dict, dest_name)
        data2 = pack('128s5s5s5s5s', msg.encode('utf-8'), cmd.encode('utf-8'),src_na
                     next_matric_name.encode('utf-8'))
        #print("received"+str(route_PORT))
        next_IP = ip_dict[next_matric_name]
        next_PORT =RoutePort_list[next_matric_name]
        #newSocket.close()
        transport_socket = socket.socket()
        transport_socket.connect((next_IP,next_PORT))
        transport_socket.send(data2)
        transport_socket.close()
```

trans_show(routeIP,routePORT)

在路由器设置的转发端口中进行报文转发

```
def trans_show(route_IP,route_PORT):
    addr = (route_IP,route_PORT)
```

```
    receive_socket = socket.socket()
    receive_socket.bind(addr)
    receive_socket.listen(10)
    IP = '127.0.0.1'  # client.app receive
    PORT = RECV_DATA_PORT
    while True:
        print("connecting....")
        newSocket,destAddr = receive_socket.accept()
        print("connected",destAddr)
        datalength = calcsize('128s5s5s5s5s')
        data = newSocket.recv(datalength)
        decode_msg,decode_cmd,decode_src_name,decode_dest_name,decode_next_matric_na
        msg = (decode_msg.decode('utf-8')).strip('\0')
        cmd = (decode_cmd.decode('utf-8')).strip('\0')
        src_name = (decode_src_name.decode('utf-8')).strip('\0')
        src_IP = ip_dict[src_name]
        src_PORT = RoutePort_list[src_name]
        dest_name = (decode_dest_name.decode('utf-8')).strip('\0')
        next_matric_name = (decode_next_matric_name.decode('utf-8')).strip('\0')
        current_name = next_matric_name
        if cmd == 'S':
            if dest_name == next_matric_name:
                print(msg)
                to_client_socket = socket.socket()
                to_client_socket.connect((IP, PORT))
                data = pack("128s5s5s", msg.encode('utf-8'),cmd.encode('utf-8'),dest
                to_client_socket.send(data)
                to_client_socket.close()
            else:
                print(next_matric_name)
                list_dict = read_list(next_matric_name)
                next_matric_name = get_next_matric(list_dict,dest_name)
                data = pack('128s5s5s5s5s', msg.encode('utf-8'),cmd.encode('utf-8'),
                        next_matric_name.encode('utf-8'))
                newSocket.close()
                transport_socket = socket.socket()
                next_IP = ip_dict[next_matric_name]
                next_PORT = RoutePort_list[next_matric_name]
                transport_socket.connect((next_IP,next_PORT))
                transport_socket.send(data)
                transport_socket.close()
#            ping_socket.connect((src_IP, src_PORT))
            respose = 'R'
            res_msg = "ping back..."
            list_dict = read_list(current_name)
            ping_socket = socket.socket()
            ping_next_matric = get_next_matric(list_dict,src_name)
            ping_next_ip = ip_dict[ping_next_matric]
```

```
            ping_next_port = RoutePort_list[ping_next_matric]
            ping_socket.connect((ping_next_ip,ping_next_port))
            ping_data = pack("128s5s5s5s5s", res_msg.encode('utf-8'),respose.encode(
            ping_socket.send(ping_data)
            ping_socket.close()
        if cmd == 'R':
            if dest_name == next_matric_name:
                to_client_socket = socket.socket()
                to_client_socket.connect((IP, PORT))
                data = pack("128s5s5s", msg.encode('utf-8'),cmd.encode('utf-8'),src_
                to_client_socket.send(data)
            else:
                list_dict = read_list(next_matric_name)
                next_matric_name = get_next_matric(list_dict, dest_name)
                data = pack('128s5s5s5s5s', msg.encode('utf-8'), cmd.encode('utf-8')
                            dest_name.encode('utf-8'),
                            next_matric_name.encode('utf-8'))
                newSocket.close()
                transport_socket = socket.socket()
                next_IP = ip_dict[next_matric_name]
                next_PORT = RoutePort_list[next_matric_name]
                transport_socket.connect((next_IP, next_PORT))
                transport_socket.send(data)
```