# 计算机网络网络层实验

1.实验目的

学习和掌握距离向量算法

## 2.实验内容

### 2.1任务一

### 2.1.1各客户机运行及收敛截图如下

客户机1

客户机2

```
[liu@localhost 2]$ python3 server.py -p 5200 -n b -e 6 -c 8 -a 2
b"a{'a': 0, 'b': 2, 'e': 2}"
 HostName | Cost | NextStep |
         | 2
             8
                       С
     С
    e | 4 | a
b"c{'c': 0, 'b': 8, 'd': 3, 'a': 10, 'e': 12}"
b"e{'e': 0, 'a': 2, 'b': 4, 'd': 6, 'c': 9}"
b"a{'a': 0, 'b': 2, 'e': 2, 'c': 10, 'd': 8}"
| HostName | Cost | NextStep |
         | 2
| 8
     а
     С
                       С
     е
              4
           | 10
```

#### 客户机3

```
[liu@localhost 2]$ python3 server.py -n c -p 5200 -b 8 -d 3
b"b{'b': 0, 'a': 2, 'c': 8, 'e': 4}"
  -------
| HostName | Cost | NextStep |
   b | 8 | b
    d
                    d
         | 10 | b
    а
    e
b"d{'d': 0, 'c': 3, 'e': 6, 'b': 11, 'a': 13}"
b"b{'b': 0, 'a': 2, 'c': 8, 'e': 4, 'd': 10}"
| HostName | Cost | NextStep |
    b | 8 | b
    d
                    d
           10
                    b
         9 |
    е
                    d
```

客户机4

```
[liu@localhost 2]$ python3 server.py -n d -p 5200 -c 3 -e 6
b"c{'c': 0, 'b': 8, 'd': 3, 'a': 10, 'e': 12}"
| HostName | Cost | NextStep |
       | 3 | c
    С
           6
    е
                   е
    b
           11
                   С
       13 c
    а
b"e{'e': 0, 'a': 2, 'b': 4, 'd': 6, 'c': 9}"
b"c{'c': 0, 'b': 8, 'd': 3, 'a': 10, 'e': 9}"
| HostName | Cost | NextStep |
    С
           6
    e
                   е
    b
           10
                   е
          I 8 I
    а
                   е
```

#### 客户机5

### 2.1.2实验代码(任务一任务二任务三代码相同)

```
import socket
import os
import sys
import argparse
import threading
import time
import prettytable as pt
#此处用来设置各个客户机的ip地址
#似乎更恰当的方式是定义一个类,这种写法比较丑陋。。
```

```
10 | ipaddrs=
    {'a':'192.168.1.201','b':'192.168.1.202','c':'192.168.1.203','d':'192.168.1.204','e':
    '192.168.1.205'}
   #记录到达每一个客户机的花费
11
12
    cost={}
    #记录临近的客户机
13
14
   user=[]
15
   #记录每一个客户机的下一条地址
16
    next={}
17
    #记录可以到达的位置
18
    host=[]
19
    #调用argparse来处理命令行参数
20
    def parserinit(parser):
21
        #指定客户机的名字
22
        parser.add_argument("-n","--name",help="input the name")
23
        #指定客户机的端口
        parser.add_argument("-p","--port",type=int,help="bind the port")
24
25
        #到达不同客户机的花费
        parser.add_argument("-a","--costa",type=int,help="cost of a")
26
        parser.add_argument("-b","--costb",type=int,help="cost of a")
27
        parser.add_argument("-c","--costc",type=int,help="cost of c")
28
        parser.add_argument("-d","--costd",type=int,help="cost of d")
29
30
        parser.add_argument("-e","--coste",type=int,help="cost of e")
31
32
    def main():
33
        #处理命令行参数
34
        parser = argparse.ArgumentParser()
35
        parserinit(parser)
36
        args = parser.parse_args()
37
        init(args)
38
        #socket绑定与设置
39
        server = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
40
        server.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
        address = (("",args.port))
41
        server.bind(address)
42
        #一个线程用来发送消息
43
        thread1 = sendThread(server, args.name, address)
44
        #一个线程用来接受消息
45
46
        thread2 = recvThread(server,args.name)
47
        thread1.start()
48
        thread2.start()
49
50
    def init(args):
51
        cost[args.name] = 0
52
        if(args.costa):
53
            cost['a'] = args.costa
            user.append('a')
54
55
            next['a'] = 'a'
56
            host.append('a')
57
        if(args.costb):
58
            cost['b'] = args.costb
59
            user.append('b')
60
            next['b'] = 'b'
```

```
61
             host.append('b')
 62
         if(args.costc):
 63
             cost['c'] = args.costc
             user.append('c')
 64
 65
             next['c'] = 'c'
 66
             host.append('c')
 67
         if(args.costd):
 68
             cost['d'] = args.costd
 69
             user.append('d')
 70
             next['d'] = 'd'
             host.append('d')
 71
 72
         if(args.coste):
 73
             cost['e'] = args.coste
 74
             user.append('e')
 75
             next['e'] = 'e'
 76
             host.append('e')
 77
 78
     class recvThread(threading.Thread):
 79
         #初始化
 80
         def __init__(self, server, name):
 81
             threading.Thread.__init__(self)
 82
             self.server = server
 83
             self.name = name
 84
         def run(self):
             while True:
 85
 86
                 #接受消息
 87
                 message,addr = self.server.recvfrom(1024)
 88
                 #调试用。。。。
 89
                 print(message)
 90
                 #获得客户机的名字,用于下面的处理
 91
                 name = message[0]
 92
                 name = chr(name)
 93
                 #将字符串转化为字典
 94
                 message = eval(message[1:])
 95
                 #遍历字典
                 for key in message:
 96
 97
                    #之前存储过信息
 98
                    if key in cost:
                         #到发送者那里去,得到的都是第一手消息,所以只要不是不可达,就更新距离
 99
100
                         if key == self.name:
101
                             if message[key] != 9999:
102
                                 cost[name] = message[key]
103
                                next[name] = name
104
                         else:
105
                             #如果距离更短,则更新短的距离
106
                             if cost[name] + message[key] < cost[key]:</pre>
107
                                 cost[key] = cost[name] + message[key]
108
                                next[key] = name
109
                             #如果距离不短,但是是来自下一跳的最新的消息,就进行更新
110
                             elif next[key] == name:
111
                                cost[key] = cost[name] + message[key]
112
                                 next[key] = name
113
                    else:
```

```
114
                         cost[key] = cost[name] + message[key]
115
                         host.append(key)
116
                         next[key] = name
117
118
     class sendThread(threading.Thread):
         def __init__(self,server,name,address):
119
120
             threading.Thread.__init__(self)
121
             #设置时间
             self.interval = 20
122
123
             self.stopped = False
124
             self.server = server
125
             self.name = name
126
             self.address = address
         def run(self):
127
128
             while not self.stopped:
129
                 time.sleep(self.interval)
130
                 #开始发送信息
131
                 sendmsg(self.server,self.name,self.address)
132
     #发送信息
133
     def sendmsg(server,name,address):
134
         for eachone in user:
135
             ncost = cost.copy()
136
             if eachone != name:
137
                 for key in ncost:
                     #如果到达临近节点需要中转,那么就发送不可达
138
139
                     #因为临近节点可以从中转处获得正确的长度
140
                     if key == eachone:
141
                         if next[key] != eachone:
142
                             ncost[kev] = 9999
143
         message = name + str(ncost)
144
         server.sendto(message.encode(),(ipaddrs[eachone],5200))
145
         tb = pt.PrettyTable()
         tb.field_names = ["HostName","Cost","NextStep"]
146
         for eachone in host:
147
             if eachone != name:
148
                 if cost[eachone] >= 9999:
149
                     tb.add_row([eachone,"none",next[eachone]])
150
151
                 else:
                     tb.add_row([eachone,cost[eachone],next[eachone]])
152
153
         print(tb)
154
155
156
     if __name__ == '__main__':
157
         main()
158
```

## 2.2任务二

#### 2.2.1运行截图

客户机2路径变短

#### 客户机1重新收敛

+   HostName +	   Cost	+   NextStep +	<del> </del>  -
b   e	2   2	b   b	
d	8	e	
c	10	b	!
+	+		+

#### 客户机2重新收敛

+	+   Cost	   NextStep	-    -
-   a   c	2   8	a	
e	2	e	
d	8	e	
+	+	+	۲

#### 客户机3重新收敛

+   HostName +	Cost	++   NextStep   +	
l b	8	l b i	
e	9	ď	
a   10   b   +			

### 客户机4收敛

1	HostName	+   Cost +	++   NextStep   ++	
5	c	3   6	c	
H	e a	8	e     e	
1	b 	8 +	e   ++	

+	+   Cost +	     NextStep
a	2	a
b	2	a
d	6	d
c	9	d
+	+	++

### 2.3任务三

#### 2.3.1运行截图

客户机二完成收敛

终端a与b的连接后

可以看到b到a的距离不断的增大,形成了回环

### 2.4反向毒化

#### 2.4.1运行截图

建立连接

客户机一

```
[liu@localhost 2]$ python3 server.py -p 5200 -n a -b 2
+----+
| HostName | Cost | NextStep |
+----+
| b | 2 | b |
+----+
| b"b{'b': 0, 'a': 2, 'c': 3}"
+----+
| HostName | Cost | NextStep |
+----+
| b | 2 | b |
+----+
| c | 5 | b |
```

客户机二

#### 客户机三

断开连接后

客户机二

#### 客户机三

```
b"b{'b': 0, 'c': 3, 'a': 9999}"

+-----+

| HostName | Cost | NextStep |

+----+

| b | 3 | b |

| a | None | b |

b"b{'b': 0 'c': 3 'a': 9999}"
```

可以看到此时可以正常显示到达a的信息

#### 2.4.2代码

```
1 import socket
2 import os
3
   import sys
   import argparse
5
   import threading
   import time
6
7
   import prettytable as pt
8
   #此处用来设置各个客户机的ip地址
9
   #似乎更恰当的方式是定义一个类,这种写法比较丑陋。。
   ipaddrs=
10
   {'a':'192.168.1.201','b':'192.168.1.202','c':'192.168.1.203','d':'192.168.1.204','e':
   '192.168.1.205'}
11 #记录到达每一个客户机的花费
12
   cost={}
   #记录临近的客户机
13
14 | user=[]
15 #记录每一个客户机的下一条地址
16
   next={}
17
   #记录可以到达的位置
   host=[]
18
19 #调用argparse来处理命令行参数
20 def parserinit(parser):
21
     #指定客户机的名字
```

```
22
        parser.add_argument("-n","--name",help="input the name")
23
        #指定客户机的端口
        parser.add_argument("-p","--port",type=int,help="bind the port")
24
        #到达不同客户机的花费
25
        parser.add_argument("-a","--costa",type=int,help="cost of a")
26
        parser.add_argument("-b","--costb",type=int,help="cost of a")
27
28
        parser.add_argument("-c","--costc",type=int,help="cost of c")
        parser.add_argument("-d","--costd",type=int,help="cost of d")
29
        parser.add_argument("-e","--coste",type=int,help="cost of e")
30
31
32
    def main():
33
        #处理命令行参数
34
        parser = argparse.ArgumentParser()
35
        parserinit(parser)
36
        args = parser.parse_args()
37
        init(args)
38
        #socket绑定与设置
39
        server = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
40
        server.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
        address = (("",args.port))
41
        server.bind(address)
42
43
        #一个线程用来发送消息
44
        thread1 = sendThread(server, args.name, address)
45
        #一个线程用来接受消息
46
        thread2 = recvThread(server,args.name)
47
        thread1.start()
        thread2.start()
48
49
50
    def init(args):
51
        cost[args.name] = 0
52
        if(args.costa):
53
            cost['a'] = args.costa
54
            user.append('a')
            next['a'] = 'a'
55
56
            host.append('a')
57
        if(args.costb):
58
            cost['b'] = args.costb
59
            user.append('b')
60
            next['b'] = 'b'
            host.append('b')
61
62
        if(args.costc):
63
            cost['c'] = args.costc
64
            user.append('c')
65
            next['c'] = 'c'
66
            host.append('c')
67
        if(args.costd):
68
            cost['d'] = args.costd
69
            user.append('d')
70
            next['d'] = 'd'
71
            host.append('d')
72
        if(args.coste):
            cost['e'] = args.coste
73
74
            user.append('e')
```

```
75
             next['e'] = 'e'
 76
             host.append('e')
 77
 78
     class recvThread(threading.Thread):
 79
         #初始化
         def __init__(self, server, name):
 80
 81
             threading.Thread.__init__(self)
 82
             self.server = server
             self.name = name
 83
 84
         def run(self):
 85
             while True:
                 #接受消息
 86
 87
                 message.addr = self.server.recvfrom(1024)
 88
                 #调试用。。。。
 89
                 print(message)
                 #获得客户机的名字,用于下面的处理
 90
 91
                 name = message[0]
 92
                 name = chr(name)
 93
                 #将字符串转化为字典
 94
                 message = eval(message[1:])
 95
                 #遍历字典
 96
                 for key in message:
 97
                    #之前存储过信息
 98
                    if key in cost:
                         #到发送者那里去,得到的都是第一手消息,所以只要不是不可达,就更新距离
 99
100
                         if key == self.name:
                            if message[key] != 9999:
101
102
                                cost[name] = message[key]
103
                                next[name] = name
104
                         else:
                             #如果距离更短,则更新短的距离
105
106
                             if cost[name] + message[key] < cost[key]:</pre>
107
                                cost[key] = cost[name] + message[key]
108
                                next[key] = name
109
                             #如果距离不短,但是是来自下一跳的最新的消息,就进行更新
110
                             elif next[key] == name:
                                cost[key] = cost[name] + message[key]
111
112
                                next[key] = name
113
                     else:
114
                         cost[key] = cost[name] + message[key]
115
                         host.append(key)
116
                         next[key] = name
117
118
     class sendThread(threading.Thread):
119
         def __init__(self, server, name, address):
120
             threading.Thread.__init__(self)
121
             #设置时间
             self.interval = 20
122
123
             self.stopped = False
             self.server = server
124
125
             self.name = name
             self.address = address
126
127
         def run(self):
```

```
128
            while not self.stopped:
129
                time.sleep(self.interval)
130
                #开始发送信息
131
                sendmsg(self.server,self.name,self.address)
132
     #发送信息
133
     def sendmsg(server,name,address):
134
         #对相邻节点发送消息
135
         for eachone in user:
136
            ncost = cost.copy()
137
             #去除自身
            if eachone != name:
138
                #遍历cost列表
139
140
                for key in ncost:
                    #去除自身
141
142
                    if key != name:
143
                        #不是相邻节点
144
                        if key != eachone:
145
                            #由相邻路由器获得的信息即下一跳地址为相邻路由器,就设为不可达
146
                            if next[key] == eachone:
147
                                ncost[key] = 9999
148
                        else:
149
                    #如果到达临近节点需要中转,那么就发送不可达
150
                    #因为临近节点可以从中转处获得正确的长度
151
                            if next[key] != eachone:
                                ncost[key] = 9999
152
         #信息,并进行输出
153
154
        message = name + str(ncost)
155
         server.sendto(message.encode(),(ipaddrs[eachone],5200))
         tb = pt.PrettyTable()
156
         tb.field_names = ["HostName","Cost","NextStep"]
157
         for eachone in host:
158
            if eachone != name:
159
                if cost[eachone] >= 9999:
160
                    tb.add_row([eachone,"none",next[eachone]])
161
162
                else:
                    tb.add_row([eachone,cost[eachone],next[eachone]])
163
164
         print(tb)
165
166
167
     if __name__ == '__main__':
168
         main()
169
```

## 2.5思考题

逆向毒化可以避免两个网关之间的路由回路,但在三个网关的情况,还会出现互相欺骗。比如A,B,C两两之间相连,D仅与C相连。当D与C断开连接的时候,A从B知道B可以到D,但B其实是从C就到D,C与D断开连接后,C从A知道A可以到达B,B之后从C知道从C可以到达B。形成了回路。

## 3.实验心得

在本次实验过程中,通过虚拟机来模拟了路由收敛的过程,以及拓扑变化的过程,制造了路由回路,以及使用逆向毒化算法抑制了路由回路。熟悉了python的多线程,熟悉了服务端建立连接的方式,以及服务端交换信息的方式。熟悉了DV算法以及逆向毒化算法