计算机网络实验报告

本文是计算机网络——路由算法实验的实验报告。

1 实验目的

学习和掌握距离向量算法

2 实验环境

 $\begin{array}{l} {\rm CentOS~6.2+GCC~4.4.6} \\ {\rm Win10+Python3.7} \end{array}$

3 实验内容

3.1 内容

编程实现并分析以下过程:

模拟路由收敛

模拟拓扑变化

制造路由回路

抑制路由回路

3.2 实验原理

3.2.1 路由器的功能

1、路由选择(Routing)

选择一条正确的路径——寻找下一跳(next hop),同时使目的可达且路径最优(距离最短、延迟最小、费用最低······)。最后建立路由表(包括距离向量、链路状态、路径向量等)2、转发(Forwarding)

根据路由选择的结果,将数据包从输入接口转发至输出接口

3.2.2 DV 算法基本思想

- 1、使用"距离"度量路由 路由表保存到达各目标的最短距离及下一跳
- 2、使用"距离向量"交换路由信息相邻路由器之间交换路由表,各自计算最佳路由——到达目标的最短距离及下一跳3、所有路由器两两定时交换,将路由信息扩散至全网,最后达到收敛状态

3.2.3 DV 算法

- 1、定义:
- adj(i) 为节点 i 的所有相邻节点的集合
- c(i, n) 为一对相邻节点 i 和 n 之间的距离
- d(i, j) 为从节点 i 到节点 j 之间的最短距离 2、公式:
- d(i, i) = 0
- $d(i, j) = \min[c(i, n) + d(n, j)]$ 。 其中 $n \in adj(i)$
- c(i, n) 为初始条件,已知
- d(n, j) 为节点 i 从邻居节点 n 获知的路由信息,已知

4 实验内容

4.1 任务 1: 模拟路由收敛

已知网络拓扑如图所示,请使用 DV 算法模拟该网络的迭代收敛过程。

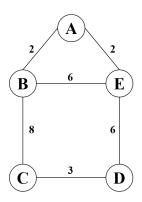


图 1 一个简单的网络拓扑结构

4.1.1 程序要求

建议使用 python3 编程;使用 socket 编程实现分布式;每次迭代后(每隔 Interval,如 30s),各节点输出路由表,输出格式可参考本课件"算法示例:一次迭代后的路由表";输出收敛后的路由表,即输出每对节点间的最短距离和下一跳。

4.1.2 源程序

```
1 import socket
 2 import sys
   import time
 4
   import threading
   import copy
 5
 6
 7
   """ thread that will call a function every interval seconds """
    class RepeatTimer(threading.Thread):
8
9
        def ___init___(self , interval , target):
10
            threading. Thread. ___init___(self)
11
            self.target = target
12
            self.interval = interval
13
            self.daemon = True
14
            self.__flag = threading.Event()
15
            self.__flag.set() # True
16
            self.__running = threading.Event()
17
            self.___running.set() # running--True
18
19
        def run(self):
20
            while self.__running.isSet():
21
                self.__flag.wait() # True--return, False--blocking until True
22
23
                self.target()
```

```
24
                time.sleep(self.interval)
25
        def pause(self):
26
            self.__flag.clear() # False--set blocking
27
28
        def resume(self):
29
30
            self.__flag.set() # True--stop blocking
31
32
        def stop(self):
            self.__flag.set()
33
            self. running.clear() # False
34
35
36
        getting the program parameters ',',
37
   def parse_argv():
38
       # Get the parameter list
39
        s = sys.argv[1:]
40
        length = len(s)
41
       # Incomplete information
42
        if length \leq 0 or (length - 1) % 3 != 0:
43
            print("error: parameters must be :"
44
                  "python dvroute.py <listening -port> <ip-address1 port1 distance1>
45
                      <ip-address2 port2 distance2> ......")
46
            return False
47
        parsed1 = \{\}
48
        port = s.pop(0)
49
       # Get the listening port number
50
51
        try:
            parsed1['port'] = int(port)
52
        except ValueError:
53
            print("error: port values must be integers. {0} is not an int.".format(
54
               port))
            return False
55
56
       # {'port': xxx, 'neighbors':[addr1,addr2,addr3], 'costs':[cost1,cost2,cost3
57
           ]}
        parsed1 ['neighbors'] = []
58
59
        parsed1['costs'] = []
        while len(s):
60
            ip = s.pop(0)
61
62
            port = s.pop(0)
```

```
63
             try:
                 port = int(port)
64
65
                 parsed1['neighbors'].append((ip, port))
             except ValueError:
66
                 print("error: port values must be integers. {0} is not an int.".
67
                    format(port))
68
                 return False
             distance = s.pop(0)
69
70
             try:
                 distance = float (distance)
71
                 parsed1['costs'].append(distance)
72
73
             except ValueError:
74
                 print ("error: link distance values must be numbers. {0} is not a
                    number.".format(distance))
                 return False
75
76
        return parsed1
77
78
    """ recalculate inter-node path costs using bellman ford algorithm """
79
    def update_costs(data, addr):
80
        # Gets the distance from the adjacent router
81
82
        dis = neighbors [addr][0]
        # Traverse the route table received
83
        for address in data.keys():
84
             if address == host addr:
85
                # we don't need to update the distance to ourselves
86
                 continue
87
             else:
88
                # iterate through neighbors and find cheapest route
89
                 if address not in routing.keys():
90
                     # If a node listed in costs is not in our list of nodes
91
92
                     # join the routing table
                     routing [address] = [dis + data[address][0], addr]
93
                 else:
94
                     if routing [address][1] == addr:
95
                         # The next hop is 'addr'
96
97
                         # update route table
                         routing [address][0] = dis + data [address][0]
98
99
                     else:
                         # To the destination network 'address', but the next hop
100
                             address is not 'addr'
101
                         if data[address][0] + dis < routing[address][0]:
```

```
# 'addr' is a closer link
102
103
                              # update route table
                              routing [address] = [data[address][0] + dis, addr]
104
105
106
107
     ''', Receive routing information''',
108
    def recv_costs():
         while True:
109
110
             try:
                 data, addr = skt.recvfrom (4096)
111
                 data = eval(data.decode('utf-8'))
112
                 if isinstance (data, dict):
113
                     # DICTIONARY is receved
114
                     # it is routing information
115
                     # update route table
116
                     if addr not in neighbors.keys():
117
118
                         # Do not process messages from routers not a neighbor
                         # (consider link disconnection)
119
120
                          continue
                     # update route table
121
122
                     update_costs(data, addr)
123
                     # update the updated-time of the neighbers
124
                     neighbors [addr][1] = time.time()
125
                 else:
                     # LIST is receved
126
                     # it is a command
127
                     # call to the modify link function
128
                     r \text{ cmd} = \text{data}[0]
129
130
                     r_parsed = data[1]
                     if r_cmd == 'linkdown':
131
132
                         linkdown (r_parsed)
                      elif r_cmd == 'linkup':
133
134
                          linkup(r_parsed)
                      else:
135
                          linkchange (r\_parsed)
136
             except ConnectionError:
137
138
                 # print(skt.gettimeout())
                 # print ("远程主机强迫关闭了一个现有的连接。")
139
140
                 pass
141
142
    ''' Send routing information '''
143
```

```
def send_costs():
144
145
        for address in neighbors.keys():
           # Dictionary -> byte stream
146
           skt.sendto(str(routing).encode('utf-8'), address)
147
148
149
150
    ''' Check the update time of your neighbor's router every THREE seconds '''
    def check_neighbors():
151
        while True:
152
153
           # The present time
154
           now time = time.time()
           # Traverse the adjacent router table
155
           for address in list(neighbors.keys()):
156
               if now_time - neighbors[address][1] > 6 * Interval:
157
                   # The last update of a router was too long
158
                   # removes it from the adjacent router table
159
160
                   neighbors.pop(address)
                   # Traverse
161
                   for add in list (routing.keys()):
162
                      # 'Address is the next hop address of a destination network
163
                          in the routing table
164
                      # the routing item needs to be removed
165
                       if routing [add][1] == address:
166
                          routing.pop(add)
           # sleep
167
           time.sleep(3)
168
169
    """ display routing info: cost to destination; route to take """
170
    def showrt():
171
        print(formatted_now())
172
        print("Distance vector list is:")
173
        print ("+-----+")
174
        print("
                                   | Cost |
                                                    _{
m Link}
175
                  Destination
        176
177
        for address in routing.keys():
178
179
           print ("|{ destination:^22}|{ cost:^7}|{ nexthop:^22}|".format(
180
               destination=str (address),
181
               cost=routing [address][0],
               next=str (routing [address ][1])))
182
        print("+------") # extra
183
           line
```

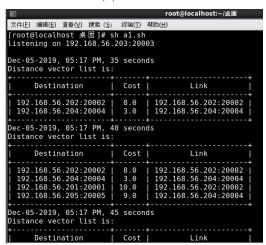
```
184
    <u>if</u> __name__ == '__main___':
185
186
        # Verify that parameters are correct
187
        parsed = parse_argv()
188
        if parsed == False:
189
            sys.exit(1)
190
        # print(parsed)
191
192
        # Different programs have different localhost, which needs to be manually
            modified within the program
        # 不同的程序对应的localhost不同,需要在程序内手动修改
193
        localhost = '127.0.0.1'
194
195
        # time between two transmissions, interval
        Interval = 30
196
        skt = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
197
        host_addr = (localhost, parsed['port'])
198
        skt.bind(host addr)
199
        # skt.setblocking(True)
200
        print('UDP Server on %s:%s...' % (host_addr[0], host_addr[1]))
201
202
        # Adjacent router: {adjacent router address: [distance, last update time]}
203
204
        neighbors = \{\}
        for i in range(len(parsed['neighbors'])):
205
            neighbors[parsed['neighbors'][i]] = [parsed['costs'][i], time.time()] #
206
                + timeout * 3
207
        # Routing table dictionary, {destination address: [distance, next hop
208
            address]}
209
        routing = {host_addr: [0, host_addr]}
210
        for i in range(len(parsed['neighbors'])):
211
212
            # Routing table dictionary, {destination address: [distance, next hop
                address]}
213
            routing [parsed ['neighbors'] [i]] = [parsed ['costs'] [i], parsed ['neighbors
                ' ] [ i ] ]
        # print(routing)
214
215
216
        # Send routing table information regularly
217
        ts = RepeatTimer(interval=Interval, target=send_costs)
218
        # Receive routing table information regularly
219
        tr = threading.Thread(target=recv_costs, daemon=True) # 守护线程, 当主线程
            结束时, 停止接收子线程
```

```
220
        # Print routing table information regularly
221
        t_showrt = RepeatTimer(interval=Interval, target=showrt)
222
        # Periodically check routing table information
223
        t_check = threading.Thread(target=check_neighbors, daemon=True) # 守护线
            程, 当主线程结束时, 停止检查
224
        ts.start()
225
        tr.start()
226
        t_showrt.start()
        t_check.start()
227
228
        cmds = ('linkdown', 'linkup', 'linkchange')
229
        while True:
230
            cmd = input()
231
232
            # print(cmd)
            if cmd in cmds:
233
234
                # Pause printing routing information
235
                t showrt.pause()
                 if cmd == 'linkdown':
236
                     parsed = input("link down : ")
237
                     ad_other = linkdown(parsed)
238
                 elif cmd == 'linkup':
239
240
                     parsed = input("link up : ")
241
                     ad_other = linkup(parsed)
242
                 else:
243
                     parsed = input("link change : ")
                     ad other = linkchange(parsed)
244
                 if ad_other != False:
245
                    # Send it to another router and change the routing table of the
246
                        other party
                     skt.sendto(str([cmd, ad_other[1]]).encode('utf-8'), ad_other[2])
247
                # Restore printing routing information
248
                t showrt.stopped = False
249
                t_showrt.resume()
250
251
            if cmd == 'close':
                break
252
253
        skt.close()
```

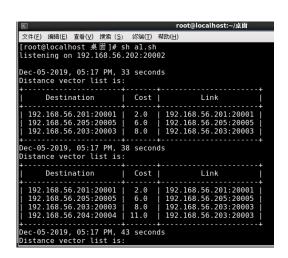
4.1.3 运行结果

回 文件(F) 编辑(E) 查看(V) 搜索 (S) 終端(T)	root@localhost:~/泉面 新助(日)	
[root@localhost 桌面]# listening on 192.168.56	sh al.sh		
Dec-05-2019, 04:53 PM, Distance vector list is		ds	-+
Destination	Cost	Link	
192.168.56.202:20002 192.168.56.205:20005 192.168.56.203:20003		192.168.56.202:20002 192.168.56.202:20002 192.168.56.202:20002	ij
Dec-05-2019, 04:53 PM, Distance vector list is		ds	-+
Destination	Cost	Link	1
192.168.56.202:20002 192.168.56.205:20005 192.168.56.203:20003 192.168.56.204:20004	2.0 2.0 2.0 10.0	192.168.56.202:20002 192.168.56.205:20005 192.168.56.202:20002 192.168.56.205:20005	

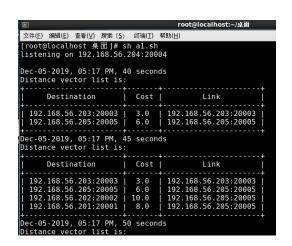
(a) 主机 A



(c) 主机 C



(b) 主机 B



(d) 主机 D

Dec-05-2019, 05:17 Distance vector li		second	is
Destination	!	Cost	Link
192.168.56.201:2 192.168.56.204:2 192.168.56.202:2	0004	6.0	192.168.56.201:20001 192.168.56.204:20004 192.168.56.202:20002
Dec-05-2019, 05:17 Distance vector li		second	is
Destination	!	Cost	Link
192.168.56.201:2 192.168.56.204:2 192.168.56.202:2	0004	2.0 6.0 4.0 9.0	192.168.56.201:20001 192.168.56.204:20004 192.168.56.201:20001 192.168.56.204:20004

(e) 主机 E

图 2 模拟路由收敛运行结果

4.2 任务 2: 模拟拓扑变化

在任务 1 的网络收敛后,将 B 和 E 之间的距离由 6 改为 2 (好消息!),模拟该变化导致的重新收敛过程。

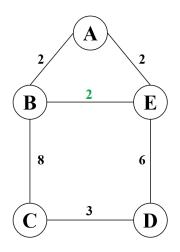


图 3 网络拓扑结构变化

4.2.1 源程序

添加函数如下:

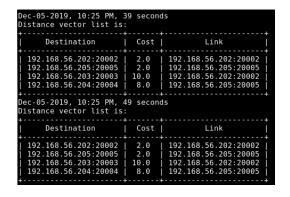
```
''', change the link ''',
   def linkchange(parsed):
2
        parsed = parsed.split()
3
        length = len(parsed)
 4
       # incompleted information
 5
        if length != 3:
 6
 7
            print("error: parameters must be :<neighbor-ip> <port> link-cost> ")
            return False
 8
 9
        ip = parsed[0]
10
11
        port = parsed[1]
        distance = parsed[2]
12
13
        try:
            port = int(port)
14
        except ValueError:
15
16
            print ("error: port values must be integers. {0} is not an int.".format(
                port))
            return False
17
18
        try:
            distance = float (distance)
19
20
        except ValueError:
```

```
21
           print ("error: link distance values must be numbers. {0} is not a number.
              ".format(distance))
22
           return False
23
       ad_temp = (ip, port)
24
       if ad_temp in neighbors.keys():
25
26
           neighbors[ad_temp] = [distance, time.time()]
27
           # comparison
28
           # change the routing table if the next hop address of the original
              routing path is ad_temp
29
           if routing [ad_temp][1] == ad_temp:
               # Modify the routing table dictionary, {destination address: [
30
                  distance, next hop address]}
               routing [ad_temp] = [distance, ad_temp]
31
           elif routing[ad_temp][0] >= distance:
32
               # Compare the distance if the next hop address of the original
33
                  routing path is not ad_temp (that is, arrive indirectly)
               # update if the distance is smaller
34
               # {destination address: [distance, next hop address]}
35
36
               routing [ad_temp] = [distance, ad_temp]
           print("the cost of {0} has been changed.".format(ad_temp))
37
           return ["linkchange", str(host_addr[0]) + ' ' + str(host_addr[1]) + ' '
38
              + str(distance), ad_temp]
39
       else:
           40
           return False
41
```

4.2.2 运行结果

[root@localhost 桌面]# sh al.sh listening on 192.168.56.201:20001						
Dec-05-2019, 10:24 PM, 59 seconds Distance vector list is:						
Destination	Cost	Link	!			
192.168.56.202:20002 192.168.56.205:20005	2.0 2.0	192.168.56.202:200 192.168.56.205:200				
+						
Destination	Cost	Link	į			
192.168.56.202:20002 192.168.56.205:20005 192.168.56.203:20003 192.168.56.204:20004	2.0 2.0 2.0 10.0	192.168.56.202:200 192.168.56.205:200 192.168.56.202:200 192.168.56.205:200	05 02			
+	+		+			

(a) 初始收敛



(b) 改变路径长度后

图 4 主机 A

[root@localhost 桌面]# listening on 192.168.56			92
Dec-05-2019, 10:25 PM, Distance vector list is		secon	ds
Destination	Ï	Cost	Link
192.168.56.201:20001 192.168.56.205:20005 192.168.56.203:20003		2.0 6.0 8.0	192.168.56.201:20001 192.168.56.205:20005 192.168.56.203:20003
Dec-05-2019, 10:25 PM, Distance vector list is		secon	ds
Destination	ļ	Cost	Link
192.168.56.201:20001 192.168.56.205:20005 192.168.56.203:20003 192.168.56.204:20004	+	2.0 4.0 8.0 10.0	192.168.56.201:20001 192.168.56.201:20001 192.168.56.203:20003 192.168.56.201:20001

(a) 初始收敛

linkchange 192.168.56.Dec-05-2019, 10:25 PM, 21 seconds
Distance vector List is:

Destination	Cost	Link
192.168.56.201:20001	2.0	192.168.56.201:20001
192.168.56.205:20005	4.0	192.168.56.201:20001
192.168.56.203:20003	8.0	192.168.56.201:20001
192.168.56.203:20004	10.0	192.168.56.201:20001
20005 2		
Dec-05-2019, 10:25 PM, 31 seconds
Distance vector List is:

Destination	Cost	Link
192.168.56.201:20001	2.0	192.168.56.201:20001
192.168.56.201:20001	2.0	192.168.56.201:20001
192.168.56.203:20003	8.0	192.168.56.203:20003
192.168.56.203:20003	8.0	192.168.56.203:20005

(b) 改变路径长度后

图 5 主机 B

[root@localhost 桌面]# sh a1.sh listening on 192.168.56.203:20003						
Dec-05-2019, 10:25 PM, 02 seconds Distance vector list is:						
Destination	Ï	Cost	Link			
192.168.56.202:20002 192.168.56.204:20004		8.0	192.168.56.202:20002 192.168.56.204:20004			
Dec-05-2019, 10:25 PM, 12 seconds Distance vector list is:						
Destination	Ï	Cost	Link			
192.168.56.202:20002 192.168.56.204:20004 192.168.56.205:20005 192.168.56.201:20001		8.0 3.0 9.0	192.168.56.202:20002 192.168.56.204:20004 192.168.56.204:20004 192.168.56.202:20002			
+	÷-		++			

(a) 初始收敛

Dec-05-2019, 10:25 PM, Distance vector list is		s .
Destination	Cost	Link
192.168.56.202:20002 192.168.56.204:20004 192.168.56.205:20005 192.168.56.201:20001	8.0 3.0 9.0 10.0	192.168.56.202:20002 192.168.56.204:20004 192.168.56.204:20004 192.168.56.202:20002
Dec-05-2019, 10:25 PM, Distance vector list is		ls .
Destination	Cost	Link
192.168.56.202:20002 192.168.56.204:20004 192.168.56.205:20005 192.168.56.201:20001	8.0 3.0 9.0 10.0	192.168.56.202:20002 192.168.56.204:20004 192.168.56.204:20004 192.168.56.202:20002

(b) 改变路径长度后

图 6 主机 C

[root@localhost 桌面]# sh a1.sh listening on 192.168.56.204:20004					
Dec-05-2019, 10:25 PM, 04 seconds Distance vector list is:					
Destination	Cost	Link	İ		
192.168.56.203:20003 192.168.56.205:20005	3.0 6.0	192.168.56.203:2000 192.168.56.205:2000			
Dec-05-2019, 10:25 PM, Distance vector list is		ds			
Destination	Cost	Link	Ĭ		
192.168.56.203:20003 192.168.56.205:20005 192.168.56.201:20001 192.168.56.202:20002	3.0 6.0 8.0 11.0	192.168.56.203:200 192.168.56.205:200 192.168.56.205:200 192.168.56.203:2000	05 05		

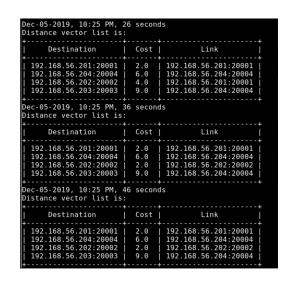
(a) 初始收敛

(b) 改变路径长度后

图 7 主机 D

[root@localhost 桌面]# sh al.sh listening on 192.168.56.205:20005					
Dec-05-2019, 10:25 PM, 06 seconds Distance vector list is:					
Destination	Cost	Link			
192.168.56.201:20001 192.168.56.204:20004 192.168.56.202:20002	6.0	192.168.56.201:20001 192.168.56.204:20004 192.168.56.202:20002			
Dec-05-2019, 10:25 PM, Distance vector list is		ds			
Destination	Cost	Link	!		
192.168.56.201:20001 192.168.56.204:20004 192.168.56.202:20002 192.168.56.203:20003	2.0 6.0 4.0 9.0	192.168.56.201:2000 192.168.56.204:20004 192.168.56.201:20001 192.168.56.204:20004			

(a) 初始收敛



(b) 改变路径长度后

图 8 主机 E

4.3 任务 3: 制造路由回路

将左图拓扑的 A 和 B 连接断开 (坏消息!),模拟该变化导致的重新收敛过程。



图 9 路由回路

4.3.1 源程序

添加函数如下:

```
',', disconnect the link',',
 2
   def linkdown(parsed):
       # split by the blank
 3
 4
        parsed = parsed.split()
        length = len(parsed)
 5
 6
       # incompleted information
 7
        if length != 2:
 8
            print("error: parameters must be :<neighbor-ip> <port> ")
            return False
 9
10
        ip = parsed[0]
11
12
        port = parsed[1]
13
        try:
            port = int(port)
14
        except ValueError:
15
```

```
print("error: port values must be integers. {0} is not an int.".format(
16
             port))
          return False
17
18
19
      ad_temp = (ip, port)
       if ad_temp in neighbors.keys():
20
21
          neighbors.pop(ad_temp)
          for address in list(routing.keys()):
22
23
              # If the next hop address of a destination network in the routing
                 table is ad_temp
24
              # remove
              if routing [address][1] == ad_temp:
25
                  routing.pop(address)
26
          print("{0} has been removed from neighbors.".format(ad_temp))
27
          28
29
       else:
          # no operation addresses in the adjacent router table
30
          print("there is no {0} in neighbors.".format(ad_temp))
31
32
          return False
```

4.3.2 运行结果

S						
○ 文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)					
Distance vector list is:						
Destination	Cost	Link				
192.168.56.201:20001 192.168.56.202:20002 192.168.56.203:20003	0 2.0 5.0	192.168.56.201:20001 192.168.56.202:20002 192.168.56.202:20002				
Dec-16-2019, 08:47 PM, 15 second Distance vector list is:	onds					
Destination	Cost	Link				
192.168.56.201:20001 192.168.56.202:20002 192.168.56.203:20003	0 2.0 5.0	192.168.56.201:20001 192.168.56.202:20002 192.168.56.202:20002				
linkdown link down : 192.168.56.202 20002 ('192.168.56.202', 20002) has been removed from neighbors. Dec-16-2019, 08:47 PM, 50 seconds Distance vector list is:						
Destination	Cost	Link				
192.168.56.201:20001	0	192.168.56.201:20001				
Dec-16-2019, 08:48 PM, 20 second Distance vector list is:	onds					
Destination	Cost	Link				
192.168.56.201:20001	0	192.168.56.201:20001				
Dec-16-2019, 08:48 PM, 50 seco Distance vector list is:	onds					
Destination	Cost	Link				
192.168.56.201:20001	0 	192.168.56.201:20001				

图 10 主机 A

<u> </u>			
文件(<u>F</u>) 编辑(<u>E</u>) 查看(<u>V</u>) 搜索(<u>S</u>)	终端(<u>T</u>)	帮助(<u>H</u>)	
192.168.56.202:20002 192.168.56.201:20001 192.168.56.203:20003		2.0 3.0	192.168.56.202:20002 192.168.56.201:20001 192.168.56.203:20003
('192.168.56.201', 20001) Dec-16-2019, 08:48 PM, 18 Distance vector list is:			noved from neighbors.
Destination	į	Cost	Link
192.168.56.202:20002 192.168.56.203:20003 192.168.56.201:20001	i	0 3.0 8.0	192.168.56.202:20002 192.168.56.203:20003 192.168.56.203:20003
Dec-16-2019, 08:48 PM, 48 Distance vector list is:	seco	nds	
Destination		Cost	Link
192.168.56.202:20002 192.168.56.203:20003 192.168.56.201:20001	i	3.0	192.168.56.202:20002 192.168.56.203:20003 192.168.56.203:20003
Dec-16-2019, 08:49 PM, 18 Distance vector list is:	seco	nds	
Destination	·····	Cost	Link
192.168.56.202:20002 192.168.56.203:20003 192.168.56.201:20001	i	0 3.0 20.0	192.168.56.202:20002 192.168.56.203:20003 192.168.56.203:20003
Dec-16-2019, 08:49 PM, 48 Distance vector list is:	seco	nds	
Destination	<u> </u>	Cost	Link
192.168.56.202:20002 192.168.56.203:20003 192.168.56.201:20001	i	0 3.0 26.0	192.168.56.202:20002 192.168.56.203:20003 192.168.56.203:20003

图 11 主机 B

医		
文件(E) 编辑(E) 查看(V) 搜索(S) 终端(E) Destination	L) 希助(<u>H</u>) I Cost	l Link I
+	-	- -
192.168.56.203:20003 192.168.56.202:20002	0 3.0	192.168.56.203:20003 192.168.56.202:20002
192.168.56.201:20001	5.0	192.168.56.202:20002
Dec-16-2019, 08:48 PM, 22 sec	+ onds	+
Distance vector list is:		
Destination	Cost	Link
192.168.56.203:20003	0	192.168.56.203:20003
192.168.56.202:20002 192.168.56.201:20001	3.0 11.0	192.168.56.202:20002 192.168.56.202:20002
÷	-	+
Dec-16-2019, 08:48 PM, 52 sec Distance vector list is:	onds	
Destination	Cost	Link
192.168.56.203:20003	0	192.168.56.203:20003
192.168.56.202:20002 192.168.56.201:20001	3.0 17.0	192.168.56.202:20002 192.168.56.202:20002
÷	+	+
Dec-16-2019, 08:49 PM, 22 sec Distance vector list is:	onds	
Destination	+ Cost	++ Link
+	.	
192.168.56.203:20003 192.168.56.202:20002	0 3.0	192.168.56.203:20003 192.168.56.202:20002
192.168.56.201:20001	23.0	192.168.56.202:20002
Dec-16-2019, 08:49 PM, 52 sec	+ onds	+
Distance vector list is:		
Destination	Cost	Link
192.168.56.203:20003	+ 0	192.168.56.203:20003
192.168.56.202:20002 192.168.56.201:20001	3.0	192.168.56.202:20002 192.168.56.202:20002
192.100.30.201.20001	29.0	192.100.30.202.20002

图 12 主机 C

4.4 任务 4: 解决路由回路

如果使用逆向毒化技术,重新模拟 A 和 B 链接断开所导致的重新收敛过程。

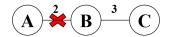


图 13 路由回路

4.4.1 源程序

修改部分函数:

```
""" recalculate inter-node path costs using bellman ford algorithm """
   def update_costs(data, addr):
2
3
       # Gets the distance from the adjacent router
        dis = neighbors [addr][0]
 4
       # Traverse the route table received
 5
        for address in data.keys():
 6
 7
            if address == host_addr:
                # we don't need to update the distance to ourselves
 8
9
                continue
10
            else:
11
                # iterate through neighbors and find cheapest route
                if address not in routing.keys():
12
13
                    # If a node listed in costs is not in our list of nodes
                    # join the routing table
14
15
                    routing [address] = [dis + data [address][0], addr]
                else:
16
                    if routing [address][1] == addr:
17
                        # The next hop is 'addr'
18
                        # update route table
19
                         routing [address][0] = dis + data [address][0]
20
                    else:
21
22
                        # To the destination network 'address', but the next hop
                            address is not 'addr'
23
                         if data[address][0] + dis < routing[address][0]:
                             # 'addr' is a closer link
24
25
                             # update route table
                             routing [address] = [data[address][0] + dis, addr]
26
            if routing [address][0] == float('inf'):
27
                routing[address][1] = None
28
29
    ''', Receive routing information''',
30
   def recv_costs():
31
        while True:
32
33
            try:
```

```
34
                data, addr = skt.recvfrom (4096)
                data = eval(data.decode('utf-8'))
35
                if isinstance (data, dict):
36
                    # DICTIONARY is receved
37
                    # it is routing information
38
                    # update route table
39
40
                    if addr not in neighbors.keys():
                        # Do not process messages from routers not a neighbor
41
42
                        # (consider link disconnection)
                        continue
43
                    for ad in data.keys():
44
                        # set float 'inf'
45
                        if data[ad][0] == 999999:
46
                            data[ad][0] = float('inf')
47
48
                    # update route table
49
                    update costs (data, addr)
50
                    # update the updated-time of the neighbers
51
                    neighbors [addr][1] = time.time()
52
53
                else:
                    # LIST is receved
54
                    # it is a command
55
                    # call to the modify link function
56
                    r_{cmd} = data[0]
57
                    r_parsed = data[1]
58
                    if r cmd == 'linkdown':
59
                        linkdown(r_parsed)
60
                    elif r cmd == 'linkup':
61
                        linkup(r_parsed)
62
                    else:
63
64
                        linkchange (r_parsed)
            except ConnectionError:
65
               # print(skt.gettimeout())
66
               # print ("远程主机强迫关闭了一个现有的连接。")
67
68
                pass
69
70
   def send_costs():
       for address in neighbors.keys():
71
72
           # Copy
73
            routing_poison_reverse = copy.deepcopy(routing)
74
               #Set the routing table to unreachable for routing items obtained
                   from adjacent routers for ad in routing_poison_reverse.
```

```
keys():
75
                 if routing_poison_reverse[ad][1] == address or
                    routing_poison_reverse[ad][0] == float('inf'):
76
                     # This routing item is obtained from the adjacent router
                     # Or this route is not accessible
77
                     # Float 'inf' converted to STR 'inf' cannot be converted back to
78
                          float 'inf'
                     routing_poison_reverse [ad][0] = 99999
79
80
                     # so it is replaced by a large number.
                         routing_poison_reverse [ad][0] = 99999
81
            # Dictionary -> byte stream
            skt.sendto(str(routing_poison_reverse).encode('utf-8'), address)
82
83
    ''', disconnect the link''',
84
    def linkdown(parsed):
85
        # split by the blank
86
        parsed = parsed.split()
87
        length = len(parsed)
88
        # incompleted information
89
        if length != 2:
90
             print("error: parameters must be :<neighbor-ip> <port> ")
91
92
             return False
93
94
        ip = parsed[0]
        port = parsed[1]
95
96
        try:
97
            port = int(port)
        except ValueError:
98
             print ("error: port values must be integers. {0} is not an int.".format(
99
                port))
            return False
100
101
102
        ad\_temp = (ip, port)
        if ad_temp in neighbors.keys():
103
             neighbors.pop(ad_temp)
104
             for address in list(routing.keys()):
105
106
                # If the next hop address of a destination network in the routing
                    table is ad temp
107
                # remove
108
                 if routing [address][1] == ad_temp:
                     routing [address] = [float('inf'), None]
109
110
```

```
print("{0} has been removed from neighbors.".format(ad_temp))
return ["linkdown", str(host_addr[0]) + ' ' + str(host_addr[1]), ad_temp
]

return ["linkdown" the adjacent router table
print("there is no {0} in neighbors.".format(ad_temp))
return False
```

4.4.2 运行结果

Dec-05-2019, 10:43 PM, Distance vector list is		secon	ds .
Destination	ļ	Cost	Link
192.168.56.201:20001 192.168.56.203:20003		2.0	192.168.56.201:20001 192.168.56.203:20003
Dec-05-2019, 10:43 PM, Distance vector list is		secon	ds
Destination	ļ	Cost	Link
192.168.56.201:20001 192.168.56.203:20003		inf 3.0	

(a) 主机 A

```
Dec-05-2019, 10:43 PM, 16 seconds
Distance vector list is:
                                    | Cost |
         Destination
                                                                   Link
   192.168.56.202:20002 |
192.168.56.203:20003 |
                                                      192.168.56.202:20002
192.168.56.202:20002
                                          2.0
5.0
linkdown 192.168.Dec-05-2019, 10:43 PM, 26 seconds
Distance vector list is:
         Destination
                                          Cost |
                                                                   Link
                                                      192.168.56.202:20002
192.168.56.202:20002
   192.168.56.202:20002 |
192.168.56.203:20003 |
                                          2.0
5.0
.
56.202 20002
Dec-05-2019, 10:43 PM, 36 seconds
Distance vector list is:
                                                                    Link
         Destination
                                           Cost
   192.168.56.202:20002
192.168.56.203:20003
```

(b) 主机 B

(c) 主机 C

图 14 逆向毒化后不会产生 loop

5 思考

5.1 演示说明

使用 UDP socket 实现 DV 信息的交换:

 $python\ client.py < listening-port > < ip-address1\ port1\ distance1 > < ip-address2\ port2\ distance2 > \dots .$

本节点通过 localhost: listening-port 进行监听(采用大端口如 20000) 邻居关系由三元组定义: IP 地址、监听端口号、与本节点的距离 拓扑改变所需功能:

linkchange <neighbor-ip> <port> <port> 改变边的大小(任务 2)
linkdown <neighbor-ip> <port> 取消和这个节点的链接(任务 3、4)
linkup <neighbor-ip> <port> 恢复以前使用 linkdown 取消的链接(便于调试)
本实验中在每个虚拟机中分别编写 shell 文件,并运行。

```
1 #命令行中键入: sh a1.sh
 2 #A
  python test2.py 20001 192.168.56.202 20002 2 192.168.56.205 20005 2
3
4
 5 #B
  python test2.py 20002 192.168.56.201 20001 2 192.168.56.205 20005 6
      192.168.56.203 20003 8
 7
8 #C
   python test2.py 20003 192.168.56.202 20002 8 192.168.56.204 20004 3
9
10
   #D
11
   python test2.py 20004 192.168.56.203 20003 3 192.168.56.205 20005 6
12
13
14
   \#E
   python test2.py 20005 10 192.168.56.201 20001 2 192.168.56.204 20004 6
15
       192.168.56.202\ 20002\ 6
16
17 #linkchange in B
18 linkchange 192.168.56.205 20005 2
```

任务 3、4 中使用的 shell 文件:

```
1 #命令行中键入: sh a2.sh
2 #A
3 python test2.py 20001 192.168.56.202 20002 2
```

```
5 #B
6 python test2.py 20002 192.168.56.201 20001 2 192.168.56.203 20003 3
7
8 #C
9 python test2.py 20003 192.168.56.202 20002 3
10
11 #linkdown in A
12 linkdown 192.168.56.202 20002
```

5.2 思考题解答

假设现有如图 15所示的网络。^[1] 现在结点 C 至结点 D 之间的链路断了,并且假设原来 从 B 到 D 的最短路径为 B—A—C—D,于是 B 将会从它的角度告知 C 这条最短路径。那么,在这种情形下,即使使用了毒性逆转技术,C 还是会选取 B 作为到达 D 的下一跳。这样,一个回环形成了。

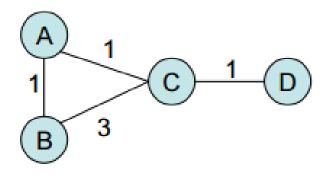


图 15 一个简单的网络拓扑结构

6 小结

通过本次实验,我熟悉了 DV 算法的执行过程以及针对 DV 算法存在的问题的解决方法。在实验过程中,主要需要注意更新前后的路由表的保存不能混乱,以尽可能模拟同步性。在编写毒性逆转时,需要对这个技术有详细的了解,在查阅相关资料后,解决了这个问题。总之,这次实验对于我理解 RIP 协议和 DV 算法很有帮助。

 $^{^{1}}$ https://people.mpi-sws.org/~gummadi/teaching/sp07/datanets/homework/homework2solution.pdf