网 络 高 级 编 程

实验报告

实验名称：网络拓扑管理

实验日期：2019.6.5

学生姓名：陈诚

学生学号：09016429

一、实验内容

利用 ping 和 traceroute 命令绘制网络拓扑结构图，找出网关，区分校园网内部 IP 和外部 IP。实验中发现了基于 UDP 的 traceroute 命令在实际使用时的缺陷，并基于 ICMP 实现了新的 traceroute 程序用于追踪并保存路由路径。基于 python 的 networkx 和 matplotlib 库，将这些路由路径最终绘制成了网络拓扑结构图。

二、实验过程

1.traceroute 命令的原理

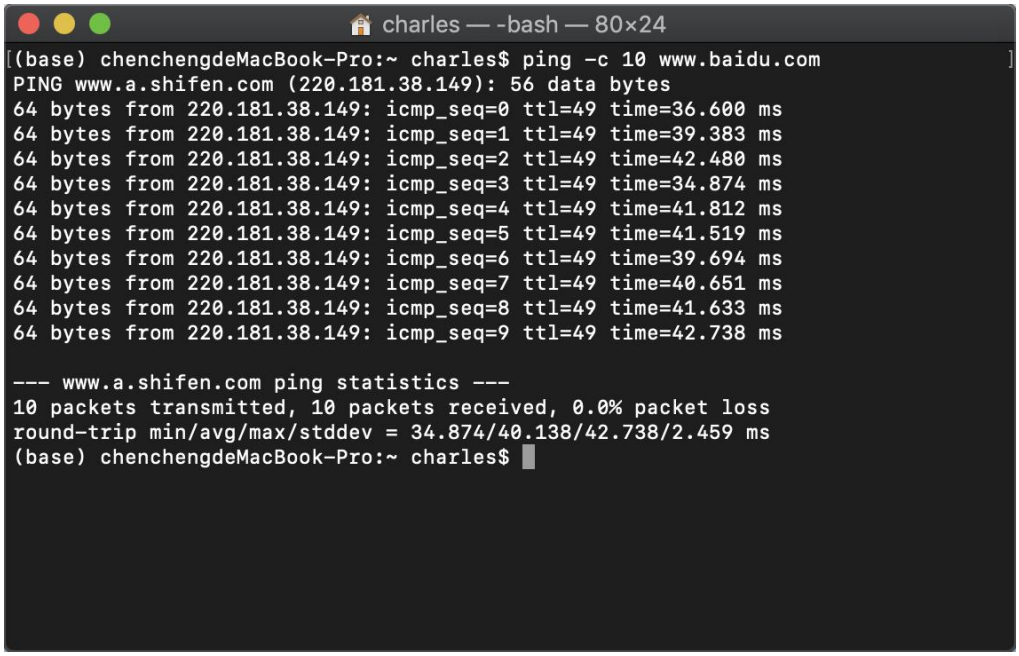
traceroute 命令利用 IP 生存时间（TTL）字段和 ICMP 错误消息来确定从一个主机到网络上其他主机的路由。

首先，traceroute 送出一个 TTL 是 1 的 IP 数据包到目的地，当路径上的第一个路由器收到这个数据包时，它将 TTL 减 1。此时，TTL 变为 0，该路由器会将此数据包丢掉，并送回一个[ICMP time exceeded]消息（包括发 IP 包的源地址，IP 包的所有内容及路由器的 IP 地址），traceroute 收到这个消息后，便知道这个路由器存在于这个路径上。

然后，traceroute 每次将送出的数据包的 TTL 加 1 来发现另一个路由器，这个重复的动作一直持续到某个数据包抵达目的地。当数据包到达目的地后，该主机则不会送回 ICMP time exceeded 消息，一旦到达目的地，由于 traceroute 通过 UDP 数据包向不可达端口(大于 30000)发送数据包，因此会收到[ICMP port unreachable]消息，故可判断到达目的地。

2.traceroute 命令实际使用结果

实际运行时用 traceroute 命令获取从当前校园网节点到 www.baidu.com 的路由路径，且已知可以 ping 通 www.baidu.com，如下图所示：



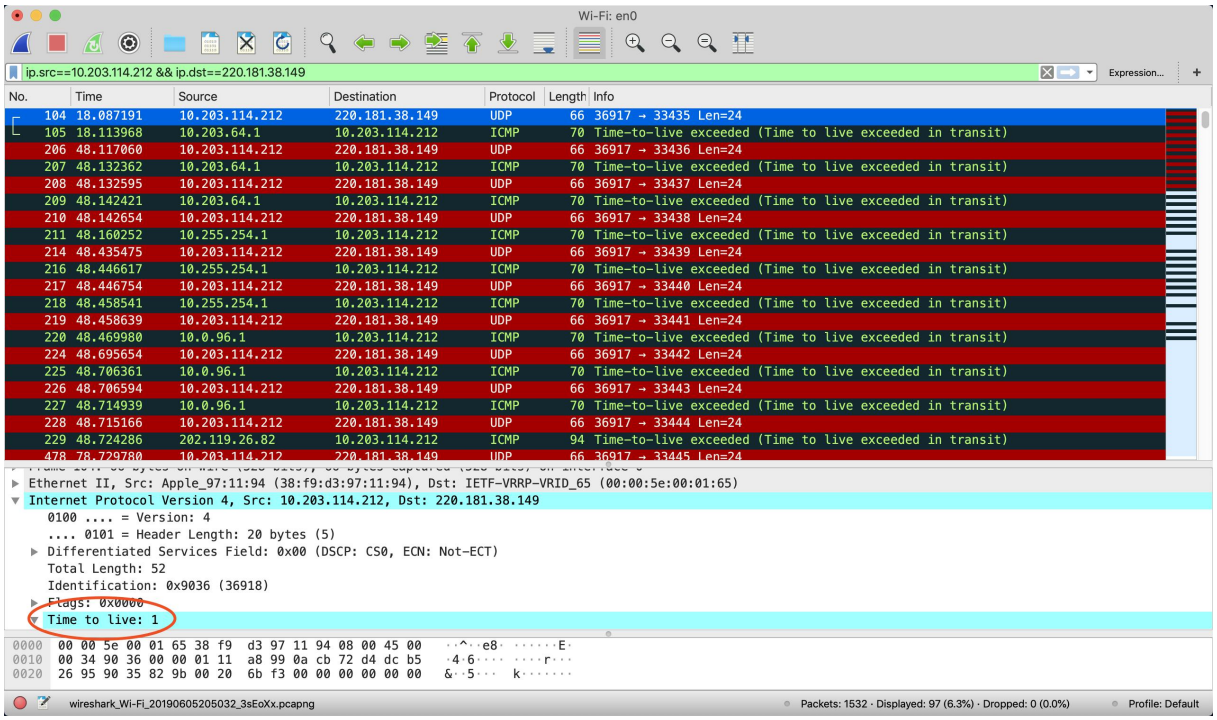
“traceroute www.baidu.com”的结果如下：

```
charles — traceroute www.baidu.com — 99×35
(base) chenchengdeMacBook-Pro:~ charles$ traceroute www.baidu.com
traceroute: Warning: www.baidu.com has multiple addresses; using 180.97.33.107
traceroute to www.a.shifen.com (180.97.33.107), 64 hops max, 52 byte packets
 1  10.203.64.1 (10.203.64.1)  25.339 ms  10.681 ms  34.413 ms
 2  10.255.254.1 (10.255.254.1)  17.761 ms  8.900 ms  12.269 ms
 3  10.0.96.1 (10.0.96.1)  15.687 ms  7.064 ms  10.315 ms
 4  202.119.26.82 (202.119.26.82)  10.429 ms  9.984 ms  9.473 ms
 5  58.213.113.65 (58.213.113.65)  66.368 ms  77.472 ms  44.685 ms
 6  221.231.175.197 (221.231.175.197)  12.920 ms  12.846 ms  12.641 ms
 7  * * *
 8  202.102.73.90 (202.102.73.90)  19.712 ms  14.369 ms  12.434 ms
 9  * * *
10  180.97.32.66 (180.97.32.66)  16.302 ms
    180.97.32.22 (180.97.32.22)  11.769 ms
    180.97.32.26 (180.97.32.26)  11.833 ms
11  10.203.194.1 (10.203.194.1)  135.281 ms  57.561 ms  59.555 ms
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

图中*表示 timed out，主机无法得到应该返回的 ICMP 错误信息包。下面结合 Wireshark 来进行抓包分析：

No.	Time	Source	Destination	Protocol	Length	Info
104	18.087191	10.203.114.212	220.181.38.149	UDP	66	36917 → 33435 Len=24
105	18.113968	10.203.64.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
206	48.117060	10.203.114.212	220.181.38.149	UDP	66	36917 → 33436 Len=24
207	48.132362	10.203.64.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
208	48.132595	10.203.114.212	220.181.38.149	UDP	66	36917 → 33437 Len=24
209	48.142421	10.203.64.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
210	48.142654	10.203.114.212	220.181.38.149	UDP	66	36917 → 33438 Len=24
211	48.160252	10.255.254.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
214	48.435475	10.203.114.212	220.181.38.149	UDP	66	36917 → 33439 Len=24
216	48.446617	10.255.254.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
217	48.446754	10.203.114.212	220.181.38.149	UDP	66	36917 → 33440 Len=24
218	48.458541	10.255.254.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
219	48.458639	10.203.114.212	220.181.38.149	UDP	66	36917 → 33441 Len=24
220	48.469980	10.0.96.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
224	48.695654	10.203.114.212	220.181.38.149	UDP	66	36917 → 33442 Len=24
225	48.706361	10.0.96.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
226	48.706594	10.203.114.212	220.181.38.149	UDP	66	36917 → 33443 Len=24
227	48.714939	10.0.96.1	10.203.114.212	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
228	48.715166	10.203.114.212	220.181.38.149	UDP	66	36917 → 33444 Len=24
229	48.724286	202.119.26.82	10.203.114.212	ICMP	94	Time-to-live exceeded (Time to live exceeded in transit)
478	78.729780	10.203.114.212	220.181.38.149	UDP	66	36917 → 33445 Len=24

可以看出，traceroute 命令默认每一跳发送 3 个 UDP 数据包。在某一个路由器中 TTL 超时后，返回一个错误类型为 Time-to-live exceeded 的 ICMP 数据包。



从图中红圈可知主机发出的第一个 UDP 包的 TTL 为 1，即 traceroute 程序首先通过 UDP 协议向目标地址 220.181.38.149 发送了一个 TTL 为 1 的数据包，然后在第一个路由器中 TTL 超时，返回一个错误类型为 Time-to-live exceeded 的 ICMP 数据包，此时通过该数据包的源地址可知第一站路由器的地址为 10.203.64.1。之后只需要不停增加 TTL 的值就能得到每一跳的地址。

3.traceroute 命令缺陷分析

然而实验中发现，大部分情况下 traceroute 并不能到达目的地，当 TTL 增加到一定大小之后就一直拿不到返回的数据包了：

```
11  10.203.194.1 (10.203.194.1)  135.281 ms  57.561 ms  59.555 ms
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

704	115.389394	10.203.114.212	220.181.38.149	UDP	66 36917 → 33468 Len=24
707	115.438858	220.181.17.94	10.203.114.212	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
712	115.709749	10.203.114.212	220.181.38.149	UDP	66 36917 → 33469 Len=24
713	115.744984	220.181.17.22	10.203.114.212	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
716	116.017891	10.203.114.212	220.181.38.149	UDP	66 36917 → 33470 Len=24
717	116.048764	220.181.17.150	10.203.114.212	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
726	117.349111	10.203.114.212	220.181.38.149	UDP	66 36917 → 33471 Len=24
741	122.351614	10.203.114.212	220.181.38.149	UDP	66 36917 → 33472 Len=24
769	127.351756	10.203.114.212	220.181.38.149	UDP	66 36917 → 33473 Len=24
789	132.354930	10.203.114.212	220.181.38.149	UDP	66 36917 → 33474 Len=24
794	137.358240	10.203.114.212	220.181.38.149	UDP	66 36917 → 33475 Len=24
821	142.361463	10.203.114.212	220.181.38.149	UDP	66 36917 → 33476 Len=24
827	147.364428	10.203.114.212	220.181.38.149	UDP	66 36917 → 33477 Len=24
832	152.366234	10.203.114.212	220.181.38.149	UDP	66 36917 → 33478 Len=24
845	157.369575	10.203.114.212	220.181.38.149	UDP	66 36917 → 33479 Len=24
905	162.374504	10.203.114.212	220.181.38.149	UDP	66 36917 → 33480 Len=24
915	167.377637	10.203.114.212	220.181.38.149	UDP	66 36917 → 33481 Len=24
919	172.382111	10.203.114.212	220.181.38.149	UDP	66 36917 → 33482 Len=24

实际上并不是所有网关都会如实返回 ICMP 超时报文。处于安全性考虑，大多数防火墙以及启用了防火墙功能的路由器缺省配置为不返回各种 ICMP 报文，其余路由器或交换机也可能被管理员主动修改配置变为不返回 ICMP 报文。因此，基于 UDP 实现的 Traceroute 程序不一定能拿到所有的沿途网关地址。

4.改进思路：基于 ICMP 实现的 traceroute

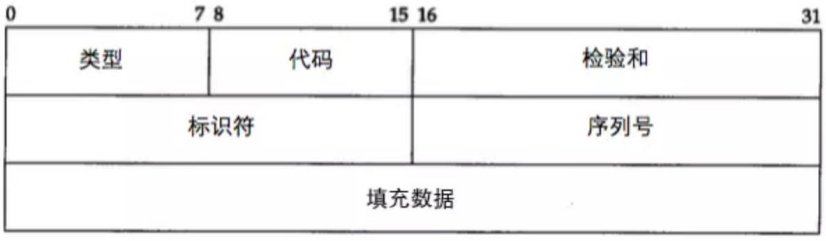
上述方案失败的原因是由于服务器对于 UDP 数据包的处理，所以在改进方案中不使用 UDP 协议，而是直接发送一个 ICMP 回显请求（echo request）数据包，服务器在收到回显请求的时候会向客户端发送一个 ICMP 回显应答（echo reply）数据包，在这之后的流程还是跟基于 UDP 实现的方案一样。这样就避免了 traceroute 数据包被服务器的防火墙策略墙掉。

完整程序则利用 traceroute 多个网站后获得的路由节点和路径来作为拓扑结构图的节点和边，利用 python 的 networkx 库来进行绘图，将 seu-wlan 的内部节点与外部网站的节点分成两类，最后根据拓扑图确定位于交界处的网关 IP。

三、ICMP 数据包说明

1.ICMP 数据包头部

ICMP 报头从 IP 报头的第 160 位开始（IP 首部 20 字节），头部的格式如下：



其中的类型字段用来表示消息的类型。报文中的标识符和序列号由发送端指定，如果这个 ICMP 报文是一个请求回显的报文（类型为 8，代码为 0），这两个字段会被原封不动的返回。

根据上图的格式可以定义如下头部结构：

header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, myID, 1)

2.返回的 ICMP 数据包

traceroute 中返回的 ICMP 数据包头部的类型共有 3 种：

1）Type==11: ICMP 超时，即 TTL 超时

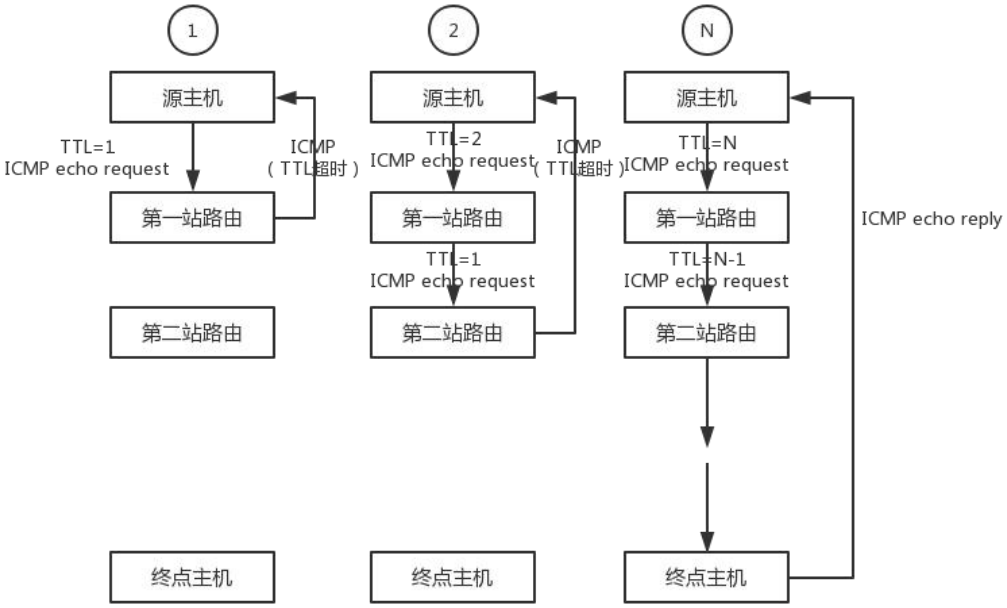
2）Type==3: 目的不可达

3）Type==0: Echo Reply，即到达终点

四、设计思路和流程

1.traceroute 方案的设计思路

如下图所示：



1.客户端发送一个 TTL 为 1 的 ICMP 请求回显数据包，在第一跳的时候超时并返回一个 ICMP 超时数据包，得到第一跳的地址。

2.客户端发送一个 TTL 为 2 的 ICMP 请求回显数据包，得到第二跳的地址。

……

N.客户端发送一个 TTL 为 N 的 ICMP 请求回显数据包，到达目标主机，目标主机返回一个 ICMP 回显应答，traceroute 结束。

2.拓扑结构图中的节点设计

实验中需要将校园网内部节点与外部节点分别标记出，而校园网内部节点 IP 开头为”10.”，因此可以根据 IP 格式将两种节点分类。如下所示：

```
if route[i].startswith('10.'):
    nodefileReader.write(route[i]+','+'0'+'\n')
else:
    nodefileReader.write(route[i]+','+'1'+'\n')
```

3.GUI 设计

“网络拓扑管理工具”的界面需要：

1）1 个输入文本框，用于输入地址以跟踪路径；

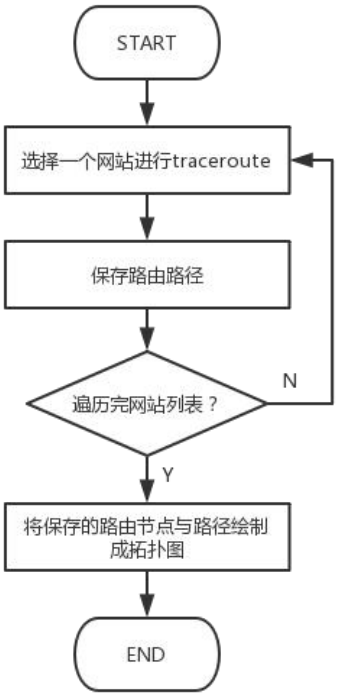
2）1 个输出文本框，用于实时显示 traceroute 过程中的结果；

3）3 个按钮，分别执行 traceroute、保存路由路径、绘制网络拓扑图的功能。

整体框架如下图：

4.完整程序的流程图

完整程序的流程图如下：



五、程序运行结果

1.traceroute 结果

由图可知，相比基于 UDP 实现的 traceroute 命令，基于 ICMP 实现的 traceroute 避免了数据包被服务器的防火墙策略墙掉的问题，同时 rtt 也很小。

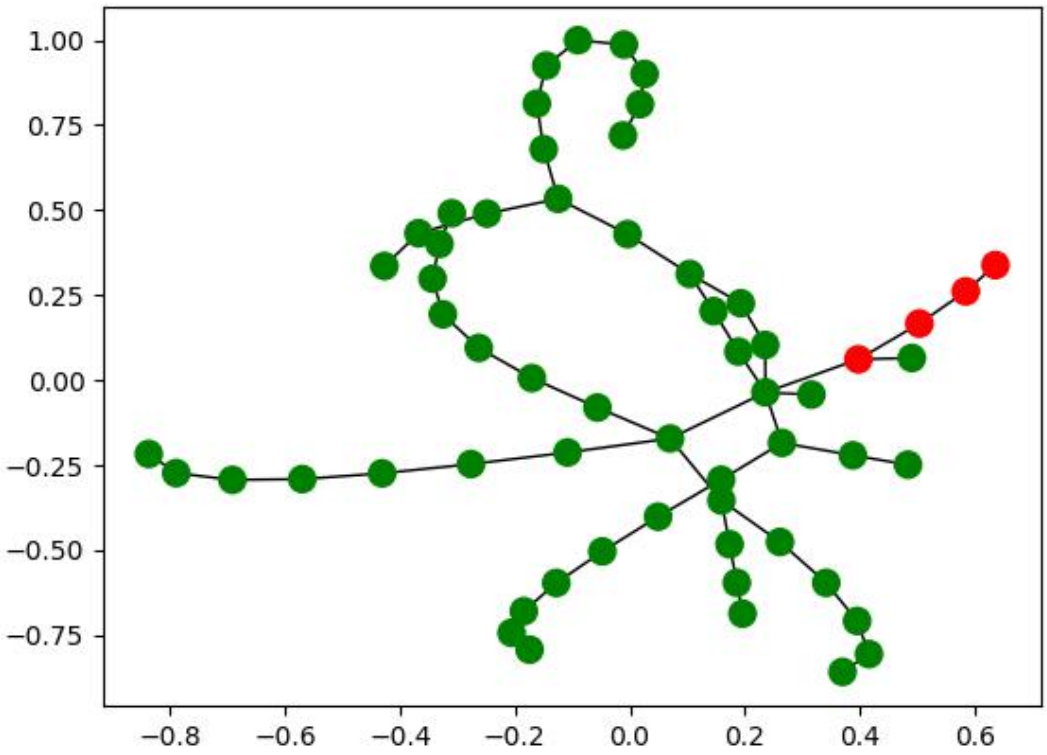
2.拓扑结构的节点与边

在对多个网站进行 traceroute 后我们得到了以下节点和路径：

10.203.114	0	10.203.114.212	10.203.64.1
10.203.64.1	0	10.203.64.1	10.255.254.1
10.255.254.	0	10.255.254.1	10.0.96.1
10.0.96.1	0	10.0.96.1	202.119.26.82
202.119.26.	1	202.119.26.82	202.112.53.133
202.112.53.	1	202.112.53.133	121.194.14.130
121.194.14.	1	121.194.14.130	121.194.14.142
121.194.14.	1	10.203.114.212	10.203.64.1
211.65.207.	1	10.203.64.1	10.255.254.1
101.4.116.2	1	10.255.254.1	10.0.96.1
101.4.117.2	1	10.0.96.1	202.119.26.82
101.4.112.6	1	202.119.26.82	211.65.207.77

由图可知，程序根据 IP 格式将 IP 分为‘0’、‘1’两类，分别代表内部节点和外部节点。

3.网络拓扑图



图中红色节点为校园网内部节点，绿色节点为外网节点。从图中可以直观的看到从 seu-wlan 到外部网站的 IP 节点，查看 label 可知该节点 IP 为 10.0.96.1。

六、实验体会

通过本次实验，我掌握了利用 traceroute 来获取路由路径并绘制网络拓扑结构图的方法，同时，我对 traceroute 命令的工作原理、实现方法以及缺陷都有了深刻的理解。实验中发现了 traceroute 在很多情况下无法完整地获取 IP 路径的问题，在经过查阅资料和学习后实现了基于 ICMP 的 traceroute 方案，并在此基础上获得了网络拓扑结构图，区分出了校园网内外部 IP，找出了交界处网关的 IP。

这次实验也让我对网络编程、对 python 在 socket 编程中的 API 有了更深的认识。

七、源程序代码及注释

1.Tracerout_ICMP.py

```
#from socket import *
import socket
import os
import sys
import struct
import time
import select
import binascii

ICMP_ECHO_REQUEST = 8
MAX_HOPS = 64
TIMEOUT = 2.0
TRIES = 2

def checksum(str_):
    #checksum of the packet
    str_=bytearray(str_)
    csum = 0
    countTo = (len(str_)//2)*2

    for count in range(0,countTo,2):
        thisVal = str_[count+1] * 256 + str_[count]
        csum = csum + thisVal
        csum = csum & 0xffffffff

    if countTo < len(str_):
        csum = csum + str_[-1]
        csum = csum & 0xffffffff

    csum = (csum >> 16) + (csum & 0xffff)
    csum = csum + (csum >> 16)
    answer = ~csum
```

```
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer

def build_packet():
    # Make the header
    myChecksum = 0
    myID = os.getpid() & 0xFFFF

    # Make a dummy header with a 0 checksum.
    # struct:Interpret strings as packed binary data
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, myID, 1)
    data = struct.pack("d", time.time())

    # Calculate the checksum on the data and the dummy header.
    # Append checksum to the header.
    myChecksum = checksum(header + data)
    if sys.platform == 'darwin':
        myChecksum = socket.htons(myChecksum) & 0xffff
        #Convert 16-bit integers from host to network byte order.
    else:
        myChecksum = socket.htons(myChecksum)

    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, myID, 1)
    packet = header + data
    return packet

def get_route(hostname):
    myAddr = socket.gethostbyname(socket.getfqdn(socket.gethostname()))
    destAddr = socket.gethostbyname(hostname)
    IProute = []
    IProute.append(myAddr)
    print("\nTraceroute to %s (IP:%s)"%(hostname,destAddr))
    print("Protocol: ICMP, %d hops max"%(MAX_HOPS))
    print("sourceAddr: %s"%myAddr)
    timeLeft = TIMEOUT
    for ttl in range(1,MAX_HOPS):
        for _ in range(TRIES):

            icmp = socket.getprotobyname("icmp")
            mySocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, icmp)
            mySocket.setsockopt(socket.IPPROTO_IP, socket.IP_TTL, struct.pack('I', ttl))
            mySocket.settimeout(TIMEOUT)
            try:
                d = build_packet()
```

```
                mySocket.sendto(d, (hostname, 0))
                t = time.time()
                startedSelect = time.time()
                whatReady = select.select([mySocket], [], [], timeLeft)
                howLongInSelect = (time.time() - startedSelect)

                if whatReady[0] == []: # Timeout
                    print ("*        *        * Request timed out.")

                recvPacket, addr = mySocket.recvfrom(1024)
                #print (addr)
                timeReceived = time.time()
                timeLeft = timeLeft - howLongInSelect

                if timeLeft <= 0:
                    timeLeft = 0
                    print ("*        *        * Request timed out.")

            except socket.timeout:
                continue

            else:
                icmpHeader = recvPacket[20:28]
                request_type, _, _, _, _ = struct.unpack("bbHHh", icmpHeader)

                if request_type == 11:
                    bytes = struct.calcsize("d")
                    timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
                    print (" %d       rtt=%.0fms, ip: %s" % (ttl,(timeReceived -t)*1000,
addr[0]))
                    IProute.append(addr[0])
                    break
                elif request_type == 3:
                    bytes = struct.calcsize("d")
                    timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
                    print (" %d       rtt=%.0fms, ip: %s" % (ttl,(timeReceived -t)*1000,
addr[0]))
                    IProute.append(addr[0])
                    break
                elif request_type == 0:
                    bytes = struct.calcsize("d")
                    timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
                    print (" %d          rtt=%.0fms, ip: %s" % (ttl,(timeReceived
-timeSent)*1000, addr[0]))
                    IProute.append(addr[0])
```

```
                return IProute
            else:
                print ("error")
                break
        finally:
            mySocket.close()
```

2.UI.py

```
#coding:utf-8
import tkinter
import socket
import struct
import time
import select
import Traceroute_ICMP as traceroute
import csv
import networkx as nx
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import matplotlib.pyplot as plt

#初始化 Tk
root = tkinter.Tk()
root.title("网络拓扑管理工具")
root.geometry('600x400')
#框架布局
frame1 = tkinter.Frame(root)
frame2 = tkinter.Frame(root)
frame3 = tkinter.Frame(root)
frame4 = tkinter.Frame(root)
#框架的位置布局
frame1.pack(side=tkinter.TOP)
frame2.pack(side=tkinter.TOP)
frame3.pack(side=tkinter.TOP)
frame4.pack(side=tkinter.TOP)
#frame1
tkinter.Label(frame1, text="请输入一个互联网地址以跟踪路径。", font=("Arial",
14)).pack(side=tkinter.TOP)
var = tkinter.Variable()
entry = tkinter.Entry(frame1, textvariable=var)
var.set("")
```

```
entry.pack(side=tkinter.LEFT)
tkinter.Label(frame1, text="（例如， 10.0.2.1 或 www.example.com）", font=("Arial", 12)).pack(side=tkinter.LEFT)
#frame3
scroll = tkinter.Scrollbar()
text = tkinter.Text(frame3, bg='grey')
scroll.config(command=text.yview) #将文本框关联到滚动条上
text.config(yscrollcommand=scroll.set) #将滚动条关联到文本框
scroll.pack(side=tkinter.RIGHT,fill=tkinter.Y)
text.pack(side=tkinter.LEFT,fill=tkinter.Y)
#frame4
f = plt.Figure(figsize=(5,4), dpi=100)
canvas = FigureCanvasTkAgg(f, master=frame4)
canvas.get_tk_widget().pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=1)
canvas.draw()
#frame2
global IProute
IProute = []

def tracert():
    text.delete(0.0,tkinter.END)
    text.insert(tkinter.END,'Start tracing route, please wait...\n')
    text.update()
    hostname = entry.get()
    myAddr = socket.gethostbyname(socket.getfqdn(socket.gethostname()))
    destAddr = socket.gethostbyname(str(hostname))
    global IProute
    IProute = []
    IProute.append(myAddr)
    text.insert(tkinter.END,"Traceroute to %s (IP:%s)\n"%(str(hostname),destAddr))
    text.insert(tkinter.END,"Protocol: ICMP, %d hops max\n"%(traceroute.MAX_HOPS))
    text.insert(tkinter.END,"sourceAddr: %s\n"%myAddr)
    text.update()
    timeLeft = traceroute.TIMEOUT
    for ttl in range(1,traceroute.MAX_HOPS):
        for _ in range(traceroute.TRIES):

            icmp = socket.getprotobyname("icmp")
            mySocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, icmp)
            mySocket.setsockopt(socket.IPPROTO_IP, socket.IP_TTL, struct.pack('I', ttl))
            mySocket.settimeout(traceroute.TIMEOUT)
            try:
                d = traceroute.build_packet()
                mySocket.sendto(d, (str(hostname), 0))
                t = time.time()
```

```
                startedSelect = time.time()
                whatReady = select.select([mySocket], [], [], timeLeft)
                howLongInSelect = (time.time() - startedSelect)

                if whatReady[0] == []: # Timeout
                    text.insert (tkinter.END,"*       *       * Request timed out.\n")
                    text.update()

                recvPacket, addr = mySocket.recvfrom(1024)
                #text.insert (addr)
                timeReceived = time.time()
                timeLeft = timeLeft - howLongInSelect

                if timeLeft <= 0:
                    timeLeft = 0
                    text.insert (tkinter.END,"*       *       * Request timed out.\n")
                    text.update()

            except socket.timeout:
                continue

            else:
                icmpHeader = recvPacket[20:28]
                request_type, _, _, _, _ = struct.unpack("bbHHh", icmpHeader)

                if request_type == 11:
                    bytes = struct.calcsize("d")
                    timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
                    text.insert (tkinter.END," %d rtt=%.0fms, ip: %s\n" %
(ttl,(timeReceived -t)*1000, addr[0]))
                    text.update()
                    IProute.append(addr[0])
                    break
                elif request_type == 3:
                    bytes = struct.calcsize("d")
                    timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
                    text.insert (tkinter.END," %d rtt=%.0fms, ip: %s\n" %
(ttl,(timeReceived -t)*1000, addr[0]))
                    text.update()
                    IProute.append(addr[0])
                    break
                elif request_type == 0:
                    bytes = struct.calcsize("d")
                    timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
                    text.insert (tkinter.END," %d rtt=%.0fms, ip: %s\n" %
```

```
(ttl,(timeReceived -timeSent)*1000, addr[0]))
                        text.insert(tkinter.END,'Traceroute over.\n')
                        text.update()
                        IProute.append(addr[0])

                        return
                    else:
                        text.insert (tkinter.END,"ERROR!\n")
                        text.update()
                        break
                finally:
                    mySocket.close()

def save():
    global IProute
    text.insert(tkinter.END,'Start saving...\n')
    text.update()
    IPs = []
    nodefile='node.csv'
    edgefile='edge.csv'
    if IProute is None:
        text.insert(tkinter.END,'ERROR in saving!\n')
        text.update()
        return

    nodefileReader = open(nodefile, 'a', encoding='utf-8')
    if nodefileReader is not None:
        text.insert(tkinter.END,'node file opened.\n')
        text.update()
    for i in range(len(IProute)):
        if IProute[i] in IPs:
            continue
        else:
            IPs.append(IProute[i])
        if IProute[i].startswith('10.'):
            nodefileReader.write(IProute[i]+','+'0'+'\n')
        else:
            nodefileReader.write(IProute[i]+','+'1'+'\n')

    edgefileReader = open(edgefile, 'a', encoding='utf-8')
    if edgefileReader is not None:
        text.insert(tkinter.END,'edge file opened.\n')
        text.update()
    for i in range(len(IProute)-1):
        edgefileReader.write(IProute[i]+','+IProute[i+1]+'\n')
```

```
        nodefileReader.close()
        edgefileReader.close()

        text.insert(tkinter.END,'...saved\n')
        text.update()

        return

def draw():
        text.insert(tkinter.END,'showing network graph!\nGreen node represents extranet ip\nRed node represents intranet ip')
        nodeList = []
        nodecolorTag = []
        edgeList = []
        nodecolor = []

        csvfile = open('node.csv','r')
        csv_reader_rows = csv.reader(csvfile)
        for one_line in csv_reader_rows:
                nodeList.append(one_line[0])
                nodecolorTag.append(one_line[1])
        #print(nodeList)
        #print(nodecolorTag)
        csvfile.close()

        csv_file = open('edge.csv','r')
        csv_reader_lines = csv.reader(csv_file)
        for one_line in csv_reader_lines:
                edgeList.append(one_line)
        csv_file.close()

        G = nx.Graph()
        G.add_nodes_from(nodeList)
        G.add_edges_from(edgeList)
        for node in G.nodes():
                if node.startswith('10.'):
                        nodecolor.append('r')
                else:
                        nodecolor.append('g')

        nx.draw_networkx(G, pos=nx.fruchterman_reingold_layout(G), node_color=nodecolor, with_labels=False, node_size=100, font_size=6)
        #f.draw(nx.draw(G, pos=nx.spring_layout(G), node_color=nodecolor, with_labels=True, font_size=6))
```

```
        plt.show()
        canvas.draw()

tkinter.Button(frame2, text="Traceroute", command=tracert, bd = 6).pack(side=tkinter.LEFT)
tkinter.Button(frame2, text="保存路由路径", command=save, bd = 6).pack(side=tkinter.LEFT)
tkinter.Button(frame2, text="显示网络拓扑图", command=draw, bd = 6).pack(side=tkinter.LEFT)

root.mainloop()
```