**基于Traceroute主动测量技术的**

**路由追踪实验报告**

学院：

班级：

学号：

姓名：

# 实验背景

因特网控制报文协议ICMP（Internet Control Message Protocol）是一个差错报告机制，是TCP/IP协议簇中的一个重要子协议，通常被IP层或更高层协议（TCP或UDP）使用，属于网络层协议，主要用于在IP主机和路由器之间传递控制消息，用于报告主机是否可达、路由是否可用等。这些控制消息虽然并不传输用户数据，但是对于收集各种网络信息、诊断和排除各种网络故障以及用户数据的传递具有至关重要的作用。

Ping（Packet Internet Groper）是一种因特网包探索器，用于测试网络连接量的程序。Ping是工作在 TCP/IP网络体系结构中应用层的一个服务命令， 主要是向特定的目的主机发送 ICMP（Internet Control Message Protocol 因特网报文控制协议）Echo 请求报文，测试目的站是否可达及了解其有关状态。

Traceroute（Windows 系统下是tracert）命令利用ICMP 协议定位您的计算机和目标计算机之间的所有路由器。TTL 值可以反映数据包经过的路由器或网关的数量，通过操纵独立ICMP 呼叫报文的TTL 值和观察该报文被抛弃的返回信息，traceroute命令能够遍历到数据包传输路径上的所有路由器。

Scapy是一个用来解析底层网络数据包的Python模块和交互式程序，可以用于数据包处理工具、数据包生成器、网络扫描器、网络发现、攻击工具和包嗅探工具。

# 实验内容

## 2.1 使用文档及运行样例

### 2.1.1 项目结构



### 2.1.2 使用方法

**1. 任务一**

开启Wireshark，在过滤器参数填入“icmp”只显示ICMP报文。

开启终端输入“ping seu.edu.cn”：

文本

描述已自动生成

共发送了四个ICMP Echo Request数据包，并均收到了相应的Reply。

在Wireshark中查看这一过程：

图片包含 图形用户界面

描述已自动生成

结果与分析一致，时序图详见2.4.1部分；

在终端输入“tracert seu.edu.cn”：

文本

描述已自动生成

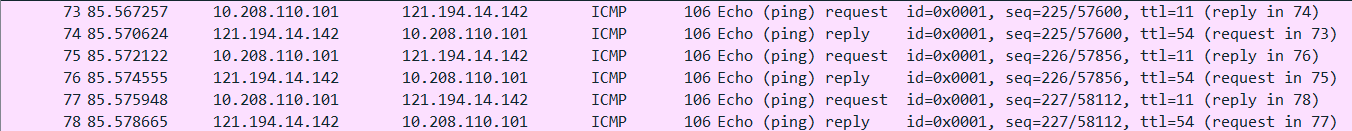
可以看到本机每次发送3个报文，共经过了11轮发送，ttl从1不断增加，最后到达目标主机121.194.14.142，也就是说在本次探测中最小ttl大于等于11才能够到达目标主机，需要11跳。

在Wireshark中查看这一过程：

表格

描述已自动生成

可以看到每轮发送三个ICMP Echo Request，但由于ttl较小无法到达目标主机，由所到达的最后一个路由返回超时信息，从而得到路径上需要经过的每一个路由。



最终ttl=11时成功到达目标主机，接收到了Echo Reply，追踪结束。时序图详见2.4.1部分。

不同时间多次的tracert的结果是可以不同的，因为网络拓扑中两个节点间的路径通常不止一条，中间路由的转发还与网络情况有关，在下面的两次tracert [www.baidu.com](http://www.baidu.com)中就出现了不同的路径：

文本

描述已自动生成

文本

描述已自动生成

**2. 任务二**

**ICMP\_Ping.py**

# mainly used in Windows system  
import ipaddress  
from optparse import OptionParser  
from random import randint  
  
from scapy.all import \*  
  
  
def ICMP\_Ping(*addr*, *ttl*):  
 # achieve the Ping using ICMP  
 # 构造ICMP Ping数据包，ttl从参数传入  
 pkt = IP(dst=*addr*, ttl=*ttl*, id=randint(0, 65535)) / ICMP(id=randint(0, 65535), seq=randint(0, 65535))  
 reply = sr1(pkt, timeout=1, verbose=False)  
 # 打印结果  
 if reply is not None:  
 reply.show()  
  
  
# python .\ICMP\_Ping.py --mode ping -a 10.208.64.0/24 -i 30  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = OptionParser()  
 parser.add\_option("--mode", dest="mode", help="选择模式<ping>")  
 parser.add\_option("-a", "--addr", dest="addr", help="指定一个IP地址或范围")  
 parser.add\_option("-i", "--TTL", dest="TTL", help="指定生存时间", default=30)  
 (options, args) = parser.parse\_args() # options: {'mode': 'ping', 'addr': '\*\*.\*\*.\*\*.\*\*'}  
 print("By:ICMP\_Ping in Active Measurement Experiment.")  
  
 if options.mode == "ping":  
 net = ipaddress.ip\_network(str(options.addr))  
 TTL\_number = int(options.TTL)  
 for item in net:  
 # 调用ICMP\_Ping函数实现功能  
 t = threading.Thread(target=ICMP\_Ping, args=(str(item), TTL\_number))  
 t.start()  
 else:  
 print("Please understand the input of ICMP\_Ping.py")  
 parser.print\_help()

在控制台中输入“python .\ICMP\_Ping.py --mode ping -a <ip地址或目标网段> -i <ttl>”来运行，-a的参数可以为IP地址（如58.192.118.142）也可以为网段（如10.208.64.0/24），会先经过ipaddress.ip\_network()处理，返回值为一个列表，输入一个地址时列表只有一项，输入一个网段时列表为网段内所有地址。

运行后多线程调用ICMP\_Ping()函数，根据传入的参数发送ICMP Echo Request，并接收Reply并打印。

**ICMP\_Treceroute.py**

# mainly used in Windows system  
from optparse import OptionParser  
from random import randint  
  
import networkx as nx  
from scapy.all import \*  
  
  
class Traceroute:  
 def \_\_init\_\_(*self*, *dst*, *max\_ttl*):  
 *self*.dst = *dst  
 self*.max\_ttl = *max\_ttl* @staticmethod  
 def visualization(*ip\_list*):  
 # achieve the IP visualization  
 graph = nx.Graph()  
 for i in range(len(*ip\_list*) - 1):  
 graph.add\_edge('[{}] {}'.format(i, *ip\_list*[i]), '[{}] {}'.format(i + 1, *ip\_list*[i + 1]))  
 nx.draw(graph, with\_labels=True, font\_weight='bold')  
 plt.show()  
  
 def tracert\_icmp(*self*, *ttl*):  
 # achieve the ICMP Tracert  
 try:  
 # 构造ICMP Ping数据包，ttl从参数传入  
 pkt = IP(dst=*self*.dst, ttl=*ttl*, id=randint(0, 65535)) / ICMP(id=randint(0, 65535), seq=randint(0, 65535))  
 # 记录发送、接收时间  
 send\_t = time.time()  
 reply = sr1(pkt, timeout=5, verbose=False)  
 recv\_t = time.time()  
 # 根据Tracert，未到达目标主机返回(1, 地址, 消耗时间);到达目标主机返回(2, 地址, 消耗时间)  
 ret\_val = 0  
 if reply.type == 11: # 11表示TTL超时  
 ret\_val = 1  
 elif reply.type == 0: # 0表示Ping成功  
 ret\_val = 2  
 return ret\_val, str(reply.src), recv\_t - send\_t  
 # 测试失败返回None  
 except Exception as e:  
 if re.match('.\*NoneType.\*', str(e)):  
 return None  
  
 def tracert(*self*):  
 ip\_list = []  
 for ttl in range(1, *self*.max\_ttl):  
 result = *self*.tracert\_icmp(ttl)  
 if result is None: # 测试失败，打印‘\*’  
 ip\_list.append('\*') # 存储每一跳的地址用于绘图  
 print('{} | {}\t| {}'.format(ttl, '\*', 'Time Out'))  
 else: # 未抵达目标主机，打印这一跳和消耗的时间；抵达目标主机，打印这一跳和消耗的时间，并跳出循环  
 ip\_list.append(result[1]) # 存储每一跳的地址用于绘图  
 time\_result = '%4.2f' % result[2]  
 print('{} | {} ms\t| {}'.format(ttl, time\_result, result[1]))  
 if result[0] == 2:  
 break  
 time.sleep(1)  
 # traceroute结束后，调用可视化函数  
 *self*.visualization(ip\_list)  
 # 使用scapy的traceroute函数及内置的可视化函数对比  
 ans, unans = traceroute(*self*.dst, maxttl=*self*.max\_ttl)  
 ans.graph(target='> graph.svg', type='svg')  
  
  
# python .\ICMP\_Traceroute.py --mode trace -a seu.edu.cn -M 30  
if \_\_name\_\_ == '\_\_main\_\_':  
 parser = OptionParser()  
 parser.add\_option('--mode', dest='mode', help='选择模式<trace>')  
 parser.add\_option('-a', '--addr', dest='addr', help='指定一个IP地址或范围')  
 parser.add\_option('-M', '--Max\_TTL', dest='Max\_TTL', help='指定trace时的最大生存时间', default=30)  
 (options, args) = parser.parse\_args() # options: {'mode': 'trace', 'addr': '\*\*.\*\*.\*\*.\*\*'}  
 print('By:ICMP\_Traceroute in Active Measurement Experiment.')  
  
 if options.mode == 'trace':  
 max\_ttl = int(options.Max\_TTL)  
 t\_obj = Traceroute(str(options.addr), max\_ttl)  
 t\_obj.tracert()  
 else:  
 print('Please understand the input of ICMP\_Traceroute.py')  
 parser.print\_help()

在控制台中输入“python .\ICMP\_Traceroute.py --mode ping -a <ip地址或域名> -M <max ttl>”来运行，-a的参数可以为IP地址（如58.192.118.142）也可以为域名（如seu.edu.cn）。

运行后调用Tracert()函数，在的循环中逐步增加ttl值，从1到max\_ttl，调用Tracert\_ICMP()函数，传入当前ttl值，在Tracert\_ICMP()中发送Ping报文，接收Reply并解析，当ttl较小不足以到达目标主机时，所到达的最后一个主机会返回time-to-live exceed信息，对应type=11，函数返回(1, 地址, 消耗时间)；当到达目标主机后则会返回正确的Echo Reply，对应type=0，函数返回(2, 地址, 消耗时间)；而超时则不会收到响应报文，返回None。

回到Tracert()中，根据返回值打印路由信息，并记录这些路由的IP地址，用于后面的绘图。直到Tracert\_ICMP()返回2，表示到达目标主机，追踪结束，调用Visualization()绘图。

使用matplotlib和networkx绘图，创建节点和边，节点表示路由、主机，标注先后顺序，得到的图详见2.4.2部分。

另外使用scapy的traceroute（根据文档，该函数并不使用ICMP，而是TCP或UDP实现）中内置的绘图函数graph（依赖ImageMagick和Graphviz），绘制路径图进行对比，详见2.4.2部分。

## 2.2 UML图与主要数据结构

**ICMP\_Treceroute.py**

文本

描述已自动生成

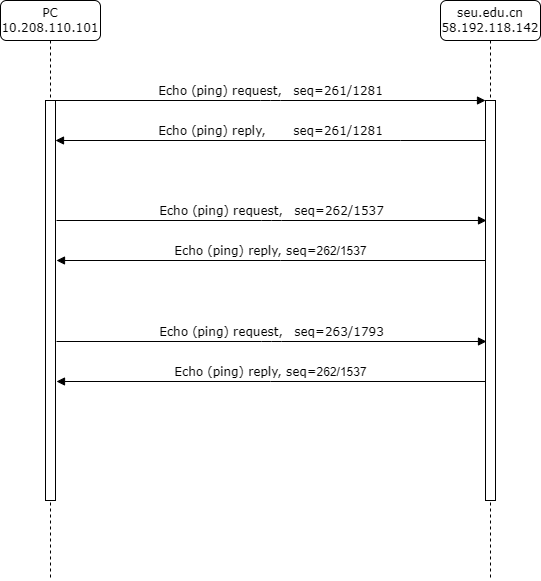
dst为str类型，是目标主机的IP地址或域名；max\_ttl为int类型，是traceroute中设定的最大ttl值。

## 2.3 主要算法说明

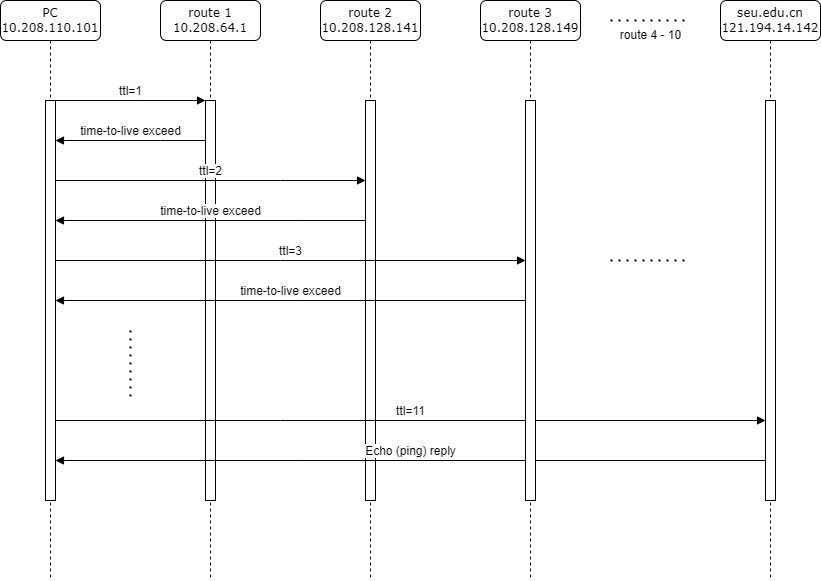
## 2.4 结果展示

### 2.4.1 任务一时序图

1. ping seu.edu.cn



2. tracert seu.edu.cn



### 2.4.2 任务二路径可视化

自己编写代码使用matplotlib生成的图像（节点位置自动生成，不太方便调整）方括号中的序号标明了到达顺序，\*表示该路由在设置上可能屏蔽了ping所以没有返回报文：

图表, 折线图

描述已自动生成

再使用scapy的traceroute中内置的绘图函数graph（依赖ImageMagick和Graphviz），绘制路径图对比：

图片包含 形状

描述已自动生成

与控制台打印出的结果是相同的：

文本

描述已自动生成

### 2.4.3 任务三问题回答

可以依靠报文中的seq字段来排序。

在Wireshark中打开ICMP\_Traceroute.pcap文件，可以看到接受到的数据包由于存在时延，是明显乱序的，主机也没有等待而是不断发送：

表格

描述已自动生成

但可以看到主机发送的ICMP Echo Request报文的seq是有序递增的，从1到24，逐次加1。

而查看一个由中间路由器返回的ttl exceed报文详细信息，可以看到其中是含有对应的seq序列号的，返回的seq与该路由器接收到的ICMP Echo Request中的seq是一致的：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

上图中，返回的seq=15，可以在pcap中看到seq=15的包是ttl=5时发送的，所以就可以对应起来：58.192.118.129是该路径上的第5个路由。

以此类推，主机可以根据返回报文的seq对其他所有路由进行排序。

# 实验总结

通过本次实验，更加了解了ping和traceroute的实现原理，之前只是会简单使用并不知道在底层是如何实现的，现在较详细的了解了使用ICMP实现的traceroute的原理，并了解除了ICMP还可以通过UDP、TCP实现。还能够通过Wireshark等抓包工具进行自主分析这一过程。