基于 ICMP 的 traceroute 程序

第五章实验 2 报告

赖泽强组

1 实验目的

- 熟悉 traceroute 程序的实现原理与效果。
- 了解数据包的传输路径以及如何遍历路由器。
- 理论结合实现，提高实践能力。

2 实验内容

实验参考系统自带的程序 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。

3 实验原理

traceroute 是用来侦测主机到目的主机之间所经路由情况的重要工具，也是最便利的工具。尽管 ping 工具也可以进行侦测，但是，因为 ip 头的限制，ping 不能完全的记录下所经过的路由器，所以 traceroute 正好就填补了这个缺憾。

traceroute 的原理在它收到目的主机的 IP 后，首先给目的主机发送一个 TTL=1 的 UDP 数据包，而经过的第一个路由器收到这个数据包以后，就自动把 TTL 减 1，而 TTL 变为 0 以后，路由器就把这个包给抛弃了，并同时产生一个主机不可达的 ICMP 数据报给主机。主机收到这个数据报以后再发一个 TTL=2 的 UDP 数据报给目的主机,然后刺激第二个路由器给主机发 ICMP 数据报。如此往复直到到达目的主机。这样，traceroute 就拿到了所有的路由器 ip。从而避开了 ip 头只能记录有限路由 IP 的问题。详细过程如下：

1）将传递到目的 IP 地址的 ICMP Echo 消息的 TTL 值被设置为 1，该消息报经过第一个路由器时，其 TTL 值减去 1，此时新产生的 TTL 值为 0。

2）由于 TTL 值被设置为 0，路由器判断此时不应该尝试继续转发数据报，而是直接抛弃该数据报。由于数据报的生存周期（TTL 值）已经到期，这个路由器会发送一个一个 ICMP 时间超时，即 TTL 值过期信息返回到客户端计算机。

3）此时，发出 traceroute 命令的客户端计算机将显示该路由器的名称，之后可以再发送一个 ICMP Echo 消息并把 TTL 值设置为 2。

4）第 1 个路由器仍然对这个 TTL 值减 1，然后，如果可能的话，将这个数据报转发到传输路径上的下一跳。当数据报抵达第 2 个路由器，TTL 值会再被减去 1，成为 0 值。

5）第 2 个路由器会像第 1 个路由器一样，抛弃这个数据包，并像第 1 个路由器那样返回一个 ICMP 消息。

6）该过程会一直持续，traceroute 命令不停递增 TTL 值，而传输路径上的路由器不断递减该值，直到数据报最终抵达预期的目的地。

7）当目的计算机接收到 ICMP Echo 消息时，会回传一个 ICMP Echo Reply 消息

4 实验环境

4.1 开发环境

语言	环境
C++	macOS 10.14.4 Apple LLVM version 10.0.1 (clang-1001.0.46.3)
Java	Windows 10 1903 IntelliJ IDEA 2018.3.3
Python	Windows 10 1903 PyCharm 2018.3.3

4.2 部署环境

可在 Windows，Linux，macOS 三平台正确编译运行。

5 实验步骤

traceroute 的用法为：traceroute <IP-address or domain-name>。

5.1 命令行参数的设置

首先我们将命令行的参数转换为 ip 地址。

```
1  u_long ulDestIP = inet_addr(argv[1]);
2      if (ulDestIP == INADDR_NONE)
3      {
4          //转换不成功时按域名解析
5          hostent* pHostent = gethostbyname(argv[1]);
6          if (pHostent)
7          {
8              ulDestIP = (*(in_addr*)pHostent->h_addr).s_addr;
9              //输出屏幕信息
10             cout << "\n 通过最多 " << DEF_MAX_HOP << " 个跃点跟踪\n 到 " << argv[1]
11                 << " [" << inet_ntoa(*(in_addr*)(&ulDestIP)) << "] 的路由:" <<
12 endl;
13         }
14         else //解析主机名失败
15         {
16             cerr << "\n 解析失败！" << argv[1] << '\n'
17                 << "error code: " << WSAGetLastError() << endl;
18             WSACleanup();
19             return -1;
20         }
21     }
```

5.2 使用 ICMP 协议创建 Raw Socket

```
SOCKET sockRaw = WSASocket(AF_INET, SOCK_RAW, IPPROTO_ICMP, NULL, 0,
WSA_FLAG_OVERLAPPED);
    if (sockRaw == INVALID_SOCKET)
    {
        cerr << "\nFailed to create a raw socket\n"
            << "error code: " << WSAGetLastError() << endl;
        WSACleanup();
        return -1;
    }
```

- Socket 地址由命令行中的参数填充。
- 创建 Socket 之后需要设置端口属性，在这里不详细说明。

5.3 创建 ICMP 包及开始探测路由

这部分的详细过程为：创建 ICMP 包发送缓冲区和接收缓冲区，填充待发送的 ICMP 包，开始探测路由，需要设置 IP 数据报头的 ttl 字段，输出当前跳站数作为路由信息序号，填充 ICMP 数据报剩余字段，记录序列号和当前时间，发送 ICMP 的 EchoRequest 数据报。这是发送部分。

接收 ICMP 的 EchoReply 数据报，因为收到的可能并非程序所期待的数据报，所以需要循环接收直到收到所要数据或超时，然后解码得到的数据包，如果解码正确则跳出接收循环发送下一个 EchoRequest 包。每次需要把 TTL 值加 1，最后输出屏幕信息。

核心代码如下：

```
    //创建 ICMP 包发送缓冲区和接收缓冲区
    //填充待发送的 ICMP 包
    //开始探测路由
    while (!bReachDestHost && iMaxHop--)
    {
        //设置 IP 数据报头的 ttl 字段
        setsockopt(sockRaw, IPPROTO_IP, IP_TTL, (char*)&iTTL, sizeof(iTTL));
        //输出当前跳站数作为路由信息序号
        //填充 ICMP 数据报剩余字段
        //记录序列号和当前时间
        //发送 ICMP 的 EchoRequest 数据报
        if (sendto(sockRaw, IcmpSendBuf, sizeof(IcmpSendBuf), 0,
            (sockaddr*)&destSockAddr, sizeof(destSockAddr)) == SOCKET_ERROR)
        {
            //如果目的主机不可达则直接退出
            return 0;
        }
        //接收 ICMP 的 EchoReply 数据报
        sockaddr_in from;
        int iFromLen = sizeof(from);
```

```
21          int iReadDataLen;
22          while (1)
23          {
24              //等待数据到达
25              iReadDataLen = recvfrom(sockRaw, IcmpRecvBuf, MAX_ICMP_PACKET_SIZE,
26                  0, (sockaddr*)&from, &iFromLen);
27              if (iReadDataLen != SOCKET_ERROR) //有数据包到达
28              {
29                  //解码得到的数据包
30              }
31              else if (WSAGetLastError() == WSAETIMEDOUT) //接收超时，打印星号
32              {
33                  cout << setw(9) << '*' << '\t' << "请求超时。" << endl;
34                  break;
35              }
36              else
37              {
38                  cerr << "\nFailed to call recvfrom\n"
39                      << "error code: " << WSAGetLastError() << endl;
40                  closesocket(sockRaw);
41                  WSACleanup();
42                  return -1;
43              }
44          }
45          //TTL 值加 1
46          iTTL++;
47      }
```

- 部分注释部分在这里没有详细给出，请见源代码。

6 实验总结

这个实验主要就是通过 traceroute 我们可以知道信息从你的计算机到互联网另一端的主机是走的什么路径。当然每次数据包由某一同样的出发点（source）到达某一同样的目的地（destination）走的路径可能会不一样，但基本上来说大部分时候所走的路由是相同的，通过这个实验使我们更加了解熟悉 traceroute 程序的实现原理与效果。

附录

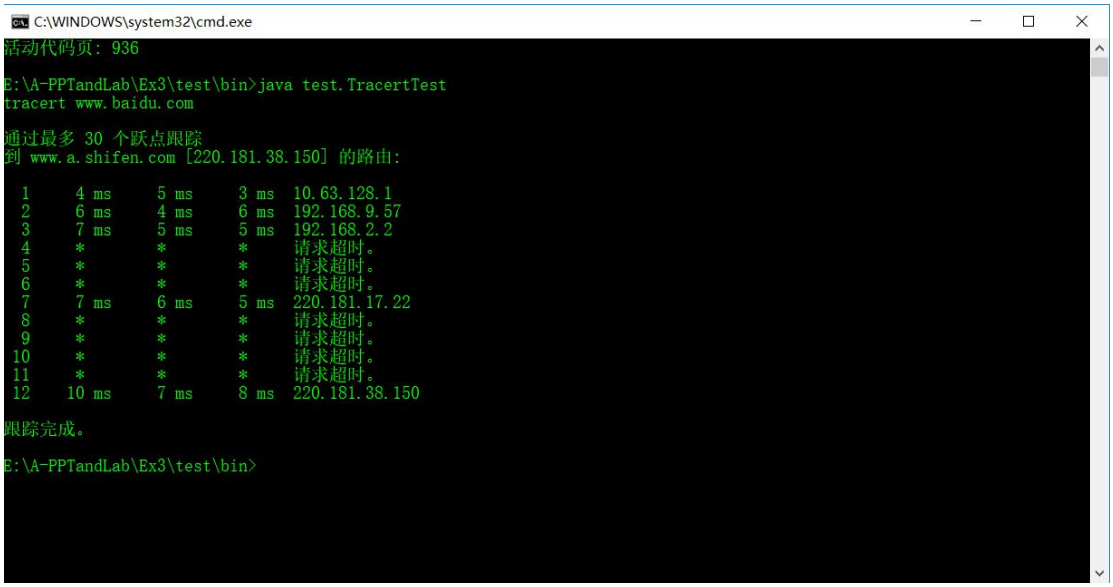

图 1 Java 运行截图

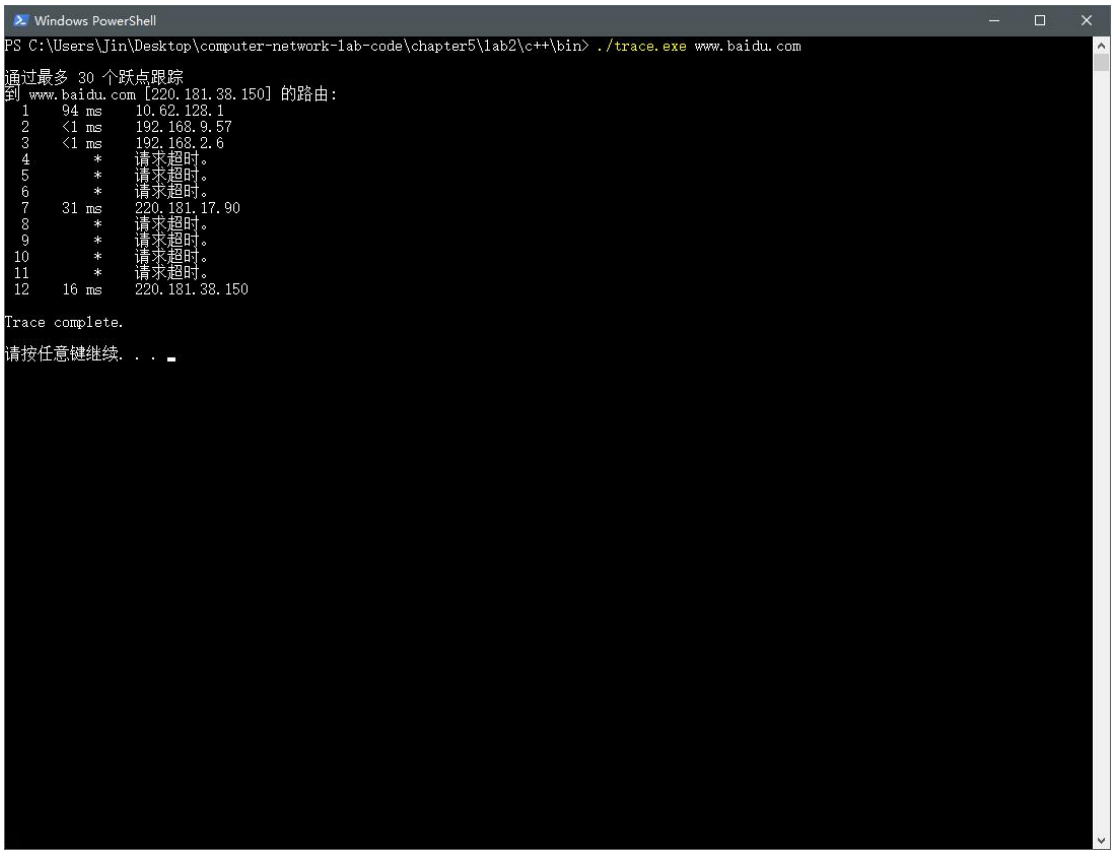

图 2 C++运行截图

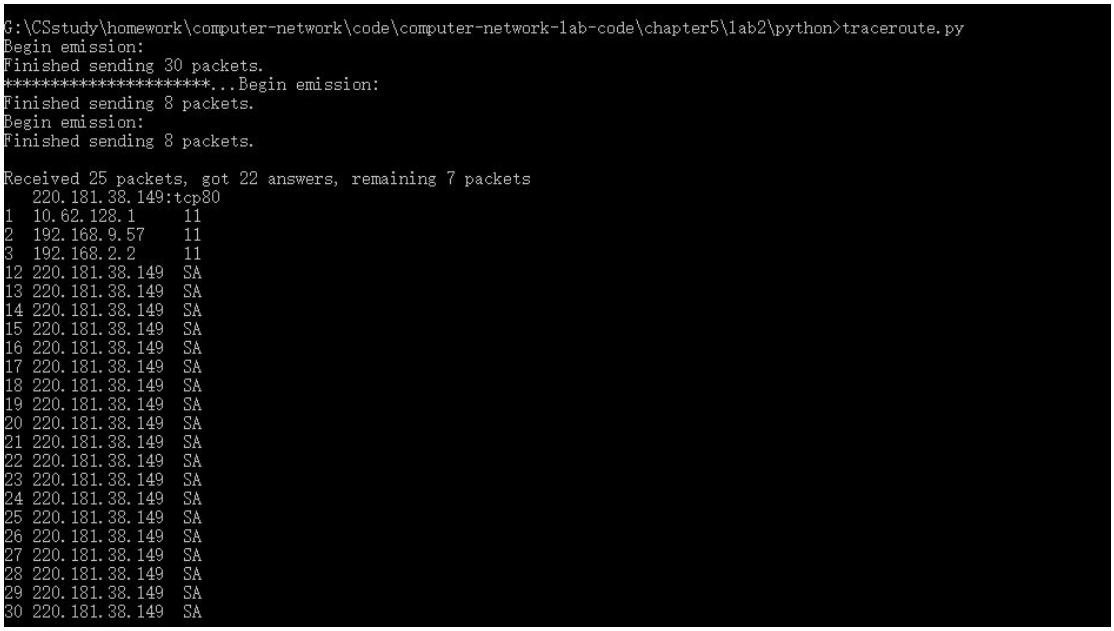

图 3 Python 运行截图