Lab2: Using ICMP messages to implement traceroute

常烁晨-521021910369

1、介绍

ICMP基于的traceroute是一种网络诊断工具，用于追踪数据包从源主机到目的主机的路径。它利用互联网控制消息协议（ICMP）发送一系列带有递增生存时间（TTL）值的数据包。每当数据包经过一个路由器，其TTL值减少1，当TTL值降至0时，路由器将丢弃该数据包并向源地址发送一个ICMP超时响应。通过记录这些响应，traceroute能够确定数据包穿越的每个路由器的地址和延迟时间。

在C++中实现基于ICMP的traceroute程序，核心是使用socket API创建一个套接字，以便发送ICMP回显请求消息并接收回显回复消息。通过设置IP头中的TTL字段，我们可以控制数据包的生存时间，从1开始递增，直至达到目标或达到预设的跳数上限。每发送一个具有特定TTL值的ICMP请求后，程序需监听ICMP回复，根据回复类型（ICMP_ECHOREPLY或ICMP_TIME_EXCEEDED）确定数据包是达到目的地还是途中被丢弃。对于每次传输，都需要计算并记录往返时间（RTT），以评估到达每个中间路由器的延迟。收到回复后，按照Linux traceroute的格式打印出源IP地址、RTT和其他相关统计数据。

2、代码

首先，参考 traceroute 的输出格式，程序最多进行 64 次返回，每次都是独立进行三次时间的测试。因此程序的主体部分是一个 64*3 的二重循环。

```
for (int ttl = 1; ttl <= max_ttl; ++ttl)
{
    bool is_final = false;
    for(int _ = 0; _ < 3; _++) {
```

```
struct sockaddr_in response_addr;
socklen_t addr_len = sizeof(response_addr);
char buffer[512];
struct timeval tv;
tv.tv_sec = timeout;
tv.tv_usec = 0;
setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);
```

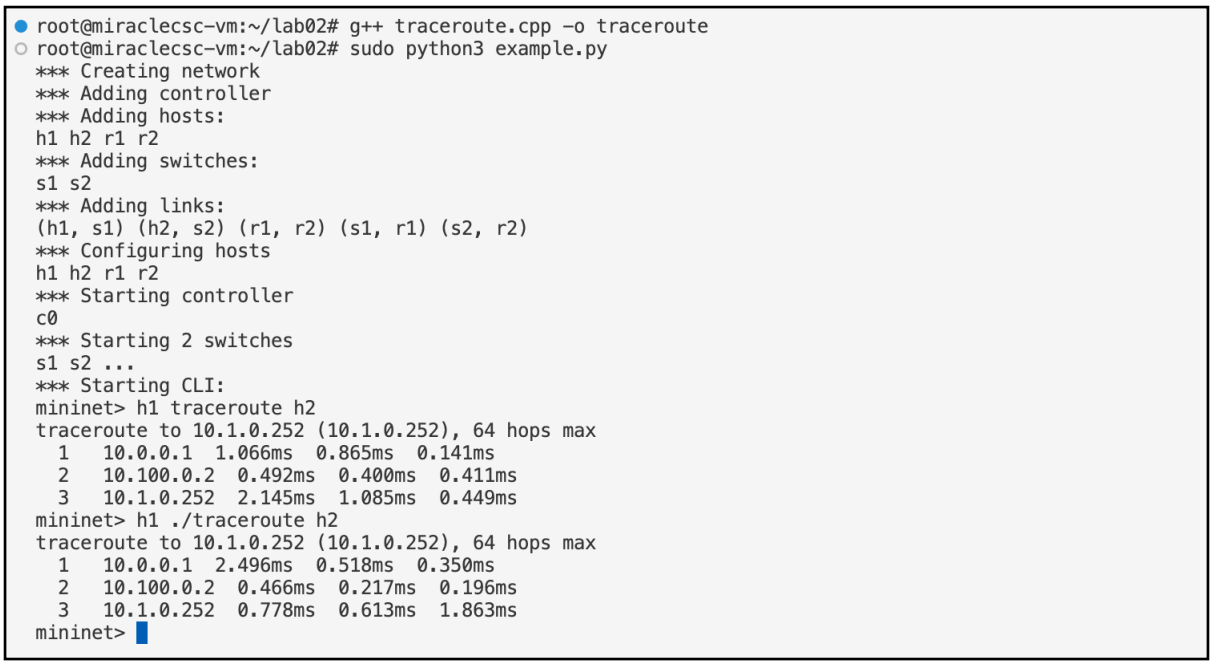
接下来是接收信息的逻辑。程序存储接收到的 ICMP 响应的来源地址，分配了足够的缓冲区来接收网络数据，此外，设置了套接字的接收超时选项，确保了程序在指定时间内没有接收到响应时能够正确地进行超时处理。

接下来程序接收 ICMP 响应，并计算接收时间与发送时间的差异，得到每一跳的往返时间。此外，通过解析接收到的数据包的 IP 和 ICMP 头部，程序能够提取出关键的路由信息，例如源 IP 地址和 ICMP 消息类型。程序根据 ICMP 消息类型判断响应的性质，如是否是目的主机的回显应答或中间路由器的超时响应。

```
int recv_len = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&response_addr, &addr_len);
if (recv_len > 0) {
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<std::chrono::duration<double, std::milli>>(end - start).count();

    struct iphdr *ip_hdr = (struct iphdr *)buffer;
    int ip_header_len = ip_hdr->ihl * 4;
    struct icmphdr *icmp_hdr = (struct icmphdr *)(buffer + ip_header_len);

    char ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &response_addr.sin_addr, ip, INET_ADDRSTRLEN);

    if (icmp_hdr->type == ICMP_ECHOREPLY) {
    else if (icmp_hdr->type == ICMP_TIME_EXCEEDED) {
```

3、输出结果

最后展示我们的程序与标准版的 traceroute 结果，可以看到两者能实现相同的输出内容。


root@miraclecsc-vm:~/lab02# g++ traceroute.cpp -o traceroute
root@miraclecsc-vm:~/lab02# sudo python3 example.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 r1 r2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (r1, r2) (s1, r1) (s2, r2)
*** Configuring hosts
h1 h2 r1 r2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> h1 traceroute h2
traceroute to 10.1.0.252 (10.1.0.252), 64 hops max
1 10.0.0.1 1.066ms 0.865ms 0.141ms
2 10.100.0.2 0.492ms 0.400ms 0.411ms
3 10.1.0.252 2.145ms 1.085ms 0.449ms
mininet> h1 ./traceroute h2
traceroute to 10.1.0.252 (10.1.0.252), 64 hops max
1 10.0.0.1 2.496ms 0.518ms 0.350ms
2 10.100.0.2 0.466ms 0.217ms 0.196ms
3 10.1.0.252 0.778ms 0.613ms 1.863ms
mininet>