

PING 程序的实现

162020321 朱家震

2023 年 6 月 2 日

1 实验内容

1.1 实验目的

理解 `tracert` 程序的概念，熟练使用原始套接字

1.2 实验环境

Linux, C

1.3 实验内容

设计一个简单的 `tracert` 程序。

2 实验设计

2.1 程序执行

首先我们在 Windows 上执行 `tracert` 程序，了解我们要做的输出是什么样的，执行出来的效果如下：

```

C:\Users\Administrator>tracert www.baidu.com

通过最多 30 个跃点跟踪
到 www.a.shifen.com [112.80.248.75] 的路由:

  1      3 ms      3 ms      1 ms  192.168.3.1
  2      5 ms     10 ms      5 ms  122.192.44.1
  3      6 ms     12 ms      8 ms  221.6.2.137
  4     16 ms     19 ms     11 ms  122.96.66.97
  5     10 ms      8 ms      8 ms  153.3.228.138
  6      9 ms     11 ms     10 ms  153.37.96.242
  7      *        *        *      请求超时。
  8     27 ms     16 ms     11 ms  182.61.216.0
  9      *        *        *      请求超时。
 10      *        *        *      请求超时。
 11     13 ms     12 ms     12 ms  112.80.248.75

跟踪完成。

```

图 1: windows 终端执行结果

```

(base) zhujiazhen@zhujiazhen-virtual-machine:~$ traceroute www.baidu.com
traceroute to www.baidu.com (112.80.248.75), 30 hops max, 60 byte packets
 1 bogon (192.168.60.2) 0.615 ms 13.647 ms 13.566 ms
 2 * * *
 3 * * *
 4 * * *
 5 * * *
 6 * * *
 7 * * *
 8 * * *
 9 * * *
10 * * *
11 * * *
12 * * *
13 * * *
14 * * *
15 * * *
16 * * *
17 * * *
18 * * *
19 * * *
20 * * *
21 * * *
22 * * *
23 * * *
24 * * *
25 * * *
26 * * *
27 * * *
28 * * *
29 * * *
30 * * *

```

图 2: Linux 终端执行结果

我们发现在 Linux 上执行的 traceroute 输出都是 “*”，也就是超时，只能收到第一个跃点的地址。通过查询发现 linux 虚拟机在 traceroute 时，默认使用 UDP 报文，而不是使用 ICMP 报文；而防火墙为了方便网络调试是放行了 ICMP 报文，但没有放行 UDP 报文，这就导致了 linux 虚拟机的 traceroute 报文（UDP）被防火墙拦截了，windows 虚拟机的 traceroute 报文（ICMP）正常通行。

```
(base) zhujiazhen@zhujiazhen-virtual-machine:~/net-experiment/mytracert$ sudo traceroute -I www.baidu.com
traceroute to www.baidu.com (112.80.248.75), 30 hops max, 60 byte packets
 1  bogon (192.168.60.2)  0.656 ms  0.610 ms  0.598 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  112.80.248.75 (112.80.248.75)  10.556 ms  10.546 ms  10.632 ms
```

图 3: Linux 终端执行结果

当我们使用-I 发送 ICMP 包时可以发现能够正常收回。接下来我们就根据 Linux 输出进行一个复现。

3 关键程序

其程序大体与 myping 程序类似，只需要在发送包之前设置包的 ttl 即可。在报告中只插入了与 myping 代码不同的函数，详情可以见附代码文件。

3.1 创建不同 ttl 的套接字

```
int create_socket(int ttl)
{
    int sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (sockfd < 0) {
        perror("socket error");
        return -1;
    }
    if (setsockopt(sockfd, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl)) < 0) {
        perror("setsockopt error");
        close(sockfd);
        return -1;
    }
    struct timeval timeout = {TIMEOUT, 0};
    if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(
        timeout)) < 0) {
        perror("setsockopt error");
        close(sockfd);
        return -1;
    }
    return sockfd;
}
```

在这里使用 *setsockopt* 函数设置 ttl 值，函数详情为：

```
int setsockopt(int sockfd, int level, int optname, const void *optval,
    socklen_t optlen);
```

sockfd: 标识一个套接口的描述字。

level: 选项定义的层次; 支持SOL_SOCKET、IPPROTO_TCP、IPPROTO_IP和IPPROTO_IPV6。

optname: 需设置的选项。

optval: 指针, 指向存放选项待设置的新值的缓冲区。

optlen: optval缓冲区长度。

3.2 接收 ICMP 包

在这里接收时,我们需要处理不同的ICMP_TYPE,根据情况是ICMP_TIME_EXCEEDED还是ICMP_ECHOREPLY 亦或是其他返回不同的值进行处理。

```
int recv_icmp_reply(int sockfd, struct sockaddr_in* from, struct timeval *
start_time)
{
    char packet[PACKET_SIZE];
    memset(packet, 0, sizeof(packet));
    socklen_t fromlen = sizeof(*from);
    struct timeval end_time;

    int n = recvfrom(sockfd, packet, sizeof(packet), 0, (struct sockaddr *)
from, &fromlen);
    gettimeofday(&end_time, NULL);
    if (n < 0) {
        if (errno == EAGAIN || errno == EWOULDBLOCK) {
            printf(" *          ");
            return 2;
        } else {
            perror("recvfrom error");
            return -1;
        }
    }

    struct iphdr *iph = (struct iphdr *)packet;
    struct icmphdr *icmph = (struct icmphdr *) (packet + (iph->ihl << 2));
    if (icmph->type == ICMP_ECHOREPLY) {
        double elapsed_time = (end_time.tv_sec - start_time->tv_sec) *
1000.0 + (end_time.tv_usec - start_time->tv_usec) / 1000.0;
        printf(" %fms ", elapsed_time);
        return 1;
    } else if (icmph->type == ICMP_TIME_EXCEEDED) {
        double elapsed_time = (end_time.tv_sec - start_time->tv_sec) *
1000.0 + (end_time.tv_usec - start_time->tv_usec) / 1000.0;
        printf(" %fms ", elapsed_time);
        return 0;
    } else {
        printf(" *          ");
    }
}
```

```

        return -1;
    }
}

```

3.3 tracer 函数

这个函数用于实现 tracer 的主要功能。根据实际的运行结果可以发现每一个跃点都是发送三个数据包的，这里我也每个跃点发送三个数据包。

```

void tracer(const char* host) {
    struct addrinfo* res = resolve_host(host);
    if (res == NULL) {
        return;
    }
    char ipstr[INET6_ADDRSTRLEN];
    inet_ntop(res->ai_family, &((struct sockaddr_in *)res->ai_addr)->sin_addr, ipstr, sizeof(ipstr));
    printf("Tracing route to %s [%s] over a maximum of %d hops:\n", host, ipstr, MAX_HOPS);
    int ttl, seq, sockfd, done = 0;
    struct sockaddr_in addr;

    for (ttl = 1; ttl <= MAX_HOPS && !done; ttl++) {
        printf("%2d  ", ttl);
        fflush(stdout);
        for (seq = 0; seq < 3; seq++) {
            sockfd = create_socket(ttl);
            if (sockfd < 0) {
                return;
            }
            memset(&addr, 0, sizeof(addr));
            addr.sin_family = AF_INET;
            addr.sin_addr = ((struct sockaddr_in *)res->ai_addr)->sin_addr;
            addr.sin_port = htons(0);

            if (send_icmp_request(sockfd, &addr, seq) < 0) {
                close(sockfd);
                continue;
            }

            struct timeval tv;
            gettimeofday(&tv, NULL);
            int ret = recv_icmp_reply(sockfd, &addr, &tv);

            if (ret < 0) {
                close(sockfd);
            }
        }
    }
}

```

```

        continue;
    }

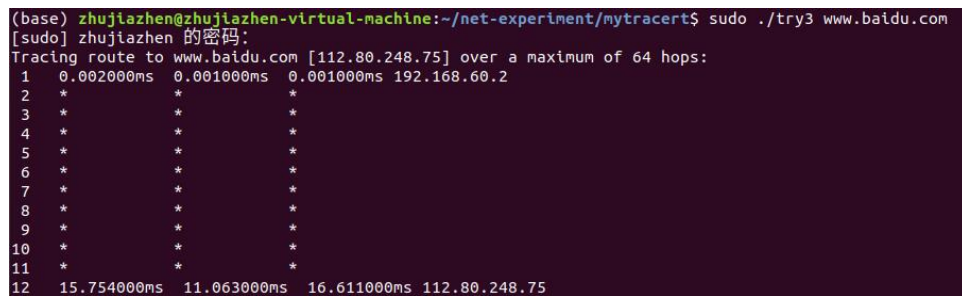
    char ipstr1[INET6_ADDRSTRLEN];
    format_addr(&addr, ipstr1, sizeof(ipstr1));

    if (ret == 2) continue;
    else if (ret == 0 && seq == 2) {
        printf("%-15s", ipstr1);
        continue;
    }
    else if (ret == 0) continue;
    else if (seq == 2) {
        if (done) {
            printf("%-15s", ipstr1);
            break;
        }
        printf("\n");
        continue;
    }
    if (seq == 2)
        printf("%-15s", ipstr1);
    if (strcmp(ipstr, ipstr1) == 0) {
        done = 1;
    }
    close(sockfd);
}
printf("\n");
}
freeaddrinfo(res);
}

```

4 实验结果与分析

最终，try3 的运行结果如下图所示。



```

(base) zhujiazhen@zhujiazhen-virtual-machine:~/net-experiment/mytracert$ sudo ./try3 www.baidu.com
[sudo] zhujiazhen 的密码:
Tracing route to www.baidu.com [112.80.248.75] over a maximum of 64 hops:
 0 0.002000ms 0.001000ms 0.001000ms 192.168.60.2
 1 * * *
 2 * * *
 3 * * *
 4 * * *
 5 * * *
 6 * * *
 7 * * *
 8 * * *
 9 * * *
10 * * *
11 * * *
12 15.754000ms 11.063000ms 16.611000ms 112.80.248.75

```

图 4: myping 终端执行结果

可以看到 Linux 下访问百度的时候只要两跳就能到，但是 Windows 的 cmd 的 tracert 却要好多跳，具体原理我还不是很清楚。