華中科技大學

“计算机网络安全”实验报告

——VPN 实验

院　　系：网络空间安全学院

专业班级：网安 1902 班

姓　　名：

学　　号：

日　　期：2022.06.11

评分项	实验报告评分（50%）	检查单分数（50%）	综合得分	教师签名
得分				

实验报告评分表

评分项目	分值	评分标准	得分
实验原理	10	10-8：原理理解准确，说明清晰完整 7-5：原理理解基本准确，说明较为清楚 4-0：说明过于简单	
VPN 系统设计	25	25-19：系统架构和模块划分合理，详细设计说明详实准确 18-11：系统架构和模块划分基本合理，详细设计说明较为详实准确 10-0：系统架构和模块划分不恰当，详细设计说明过于简单	
VPN 实现细节	25	25-19：文字表达清晰流畅，实现方法技术优良，与设计实现及代码一致 18-11：文字表达较清晰流畅，实现方法一般，与设计实现及代码有偏差 10-0：文字表达混乱，实现方法过于简单	
测试结果与分析	20	20-15：功能测试覆盖完备，测试结果理想，分析说明合理可信 14-9：功能测试覆盖基本完备，测试结果基本达标，分析说明较少 8-0：功能测试覆盖不够，测试未达到任务要求，缺乏分析说明	
体会与建议	10	10-8：心得体会真情实感，意见中肯，建议合理可行，体现了个人的思考 7-5：心得体会较为真实，意见建议较为具体 4-0：过于简单敷衍	
格式规范	10	图、表的说明，行间距、缩进、目录等不规范相应扣分	
总 分			

目　　录

1 实验原理

1.1 网络拓扑

该实验中构建的网络拓扑图如图 1-1 所示，docker 网络 10.0.2.0/24 模拟外部网络，docker 网络 192.168.60.0/24 模拟内部网络，二者网关均为主机，对应的网关 IP 地址分别为 10.0.2.8 和 192.168.60.1。docker 容器 HostU 模拟外网主机 10.0.2.7，docker 容器 HostV 模拟内网主机 192.168.60.101，外网主机需要与内网主机进行安全通信，因此需要在外网主机开启 VPN 客户端，通过 VPN 网关与内网主机进行通信。

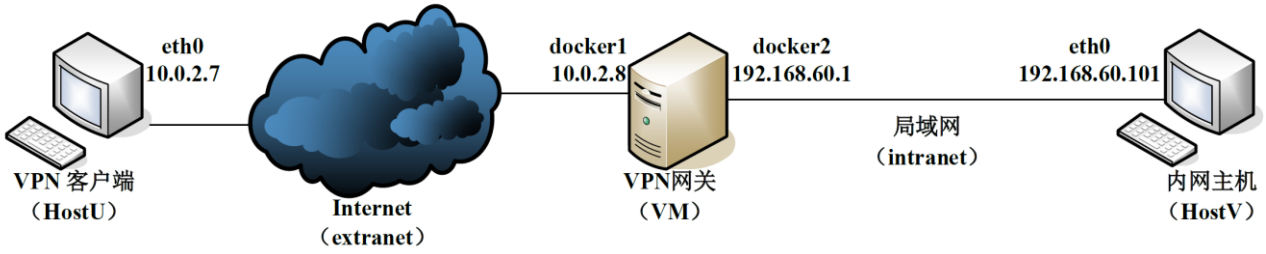

图 1-1 网络拓扑

1.2 通信机制

外网主机 10.0.2.7 向内网主机 192.168.60.101 的发送数据的通信流程如图 1-2 所示：

- 在外网主机与内网主机进行通信前，外网主机首先启动 VPN 客户端程序，与 VPN 网关上的 VPN 服务端程序建立 TLS 连接并协商外网主机的虚拟 IP，此处假设协商已经完成并且外网主机的虚拟 IP 为 192.168.53.55。
- 外网主机程序产生的数据通过 socket 进行 TCP/UDP 封装，希望发送到内网主机，此时数据包的目的地址为 192.168.60.101。
- 数据包进入内核后，内核在路由表中为这个数据包选择网络出口。由于内网主机与外部网络不直接，无法不经过 VPN 服务进行通信，因此 VPN 客户端在外网主机为发送到 192.168.60.0/24 的数据包添加一条路由表，指示这些数据包通过 VPN 客户端创建的 TUN0 虚拟网卡进入 VPN 客户端程序。同时，由于使用 TUN0 来路由数据包，内核相应地会使用 TUN0 的 IP 地址作为源 IP 地址，即 192.168.53.55。
- VPN 客户端程序监听 TUN0 虚拟网卡，当数据包进入 TUN0 时，VPN 客户端程序将其通过已经建立好的 TLS 连接，将需要发送的数据包通过 TLS 隧道进行传输（在被 TLS 进

行封装的同时被加密，作为新数据包的数据载荷），此时新数据包的源 IP 地址变为 10.0.2.7，目标 IP 地址变为 10.0.2.8。

- 新数据包经过外部网络传输至 VPN 网关，由于该数据包的内容是经过 TLS 的加密的，原数据包的数据不会被公共网络中的恶意用户得到。
- VPN 网关获得数据包后，数据载荷部分被 TLS 解密并交给 VPN 服务端程序。
- VPN 服务端程序将数据包通过 TUN0 交给内核，内核根据转发表将其通过与内网连接的 eth2 接口发出，发向内部网络并被内网主机 192.168.60.101 接收，数据包的发送完成。

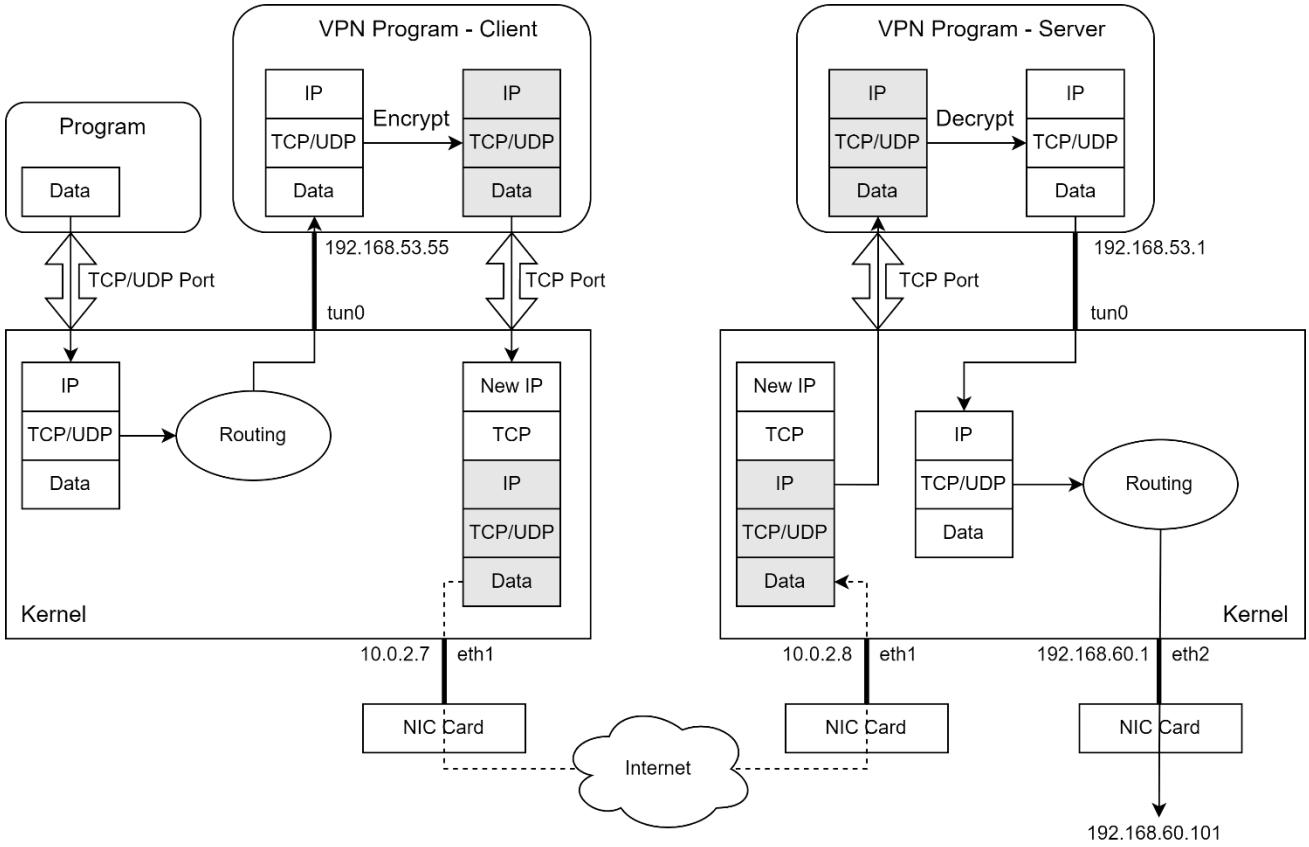

图 1-2 外网主机向内网主机发送数据的通信过程

从内网主机 192.168.60.101 向外网主机 10.0.2.7 的发送数据的通信流程与以上流程相反，且在 VPN 服务端进行加密，在 VPN 客户端进行解密。

1.3 加密原理

在该 VPN 客户端与服务端的实现中，使用了 TLS 提供的加密服务。基于 RSA 密钥协商算法的 TLS 首次握手流程如图 1-3 所示[1]：

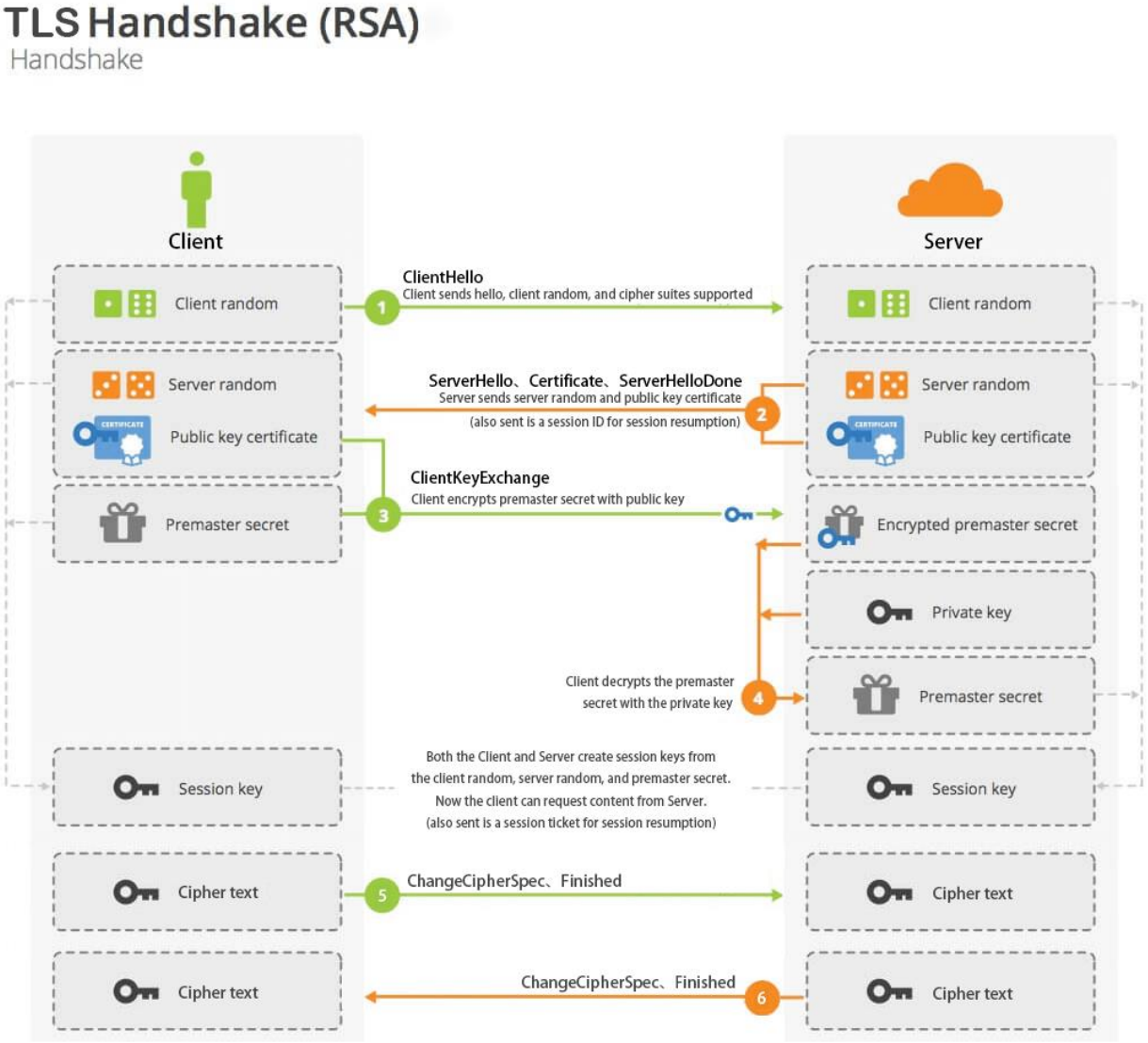

图 1-3 基于 RSA 密钥协商算法的 TLS 首次握手流程

- 握手开始，Client 先发送 ClientHello，在这条消息中，Client 会上报它支持的所有“能力”：client_version 中标识了 Client 能支持的最高 TLS 版本号；random 中标识了 Client 生成的随机数，用于预备主密钥和主密钥以及会话密钥的生成，总长度是 32 字节，其中前 4 个字节为时间戳，后 28 个字节为随机数；cipher_suites 标识了 Client 能够支持的密码套件；extensions 中标识了 Client 能够支持的所有扩展。
- Server 在收到 ClientHello 之后，如果能够继续协商，就会发送 ServerHello，否则发送 Hello Request 重新协商。在 ServerHello 中，Server 会结合 Client 的能力，选择出双方都支持的协议版本以及密码套件进行下一步的握手流程：server_version 中标识了经过协商以后，Server 选出了双方都支持的协议版本；random 中标识了 Server 生成的随机数，用于预备主密钥和主密钥以及会话密钥的生成，内容格式与

Client 的随机数相同；cipher_suites 标识了经过协商以后，Server 选出了双方都支持的密码套件;extensions 中标识了 Server 处理 Client 的 extensions 之后的结果。

- 当协商出了双方都能满足的密钥套件，根据需要 Server 会发送 Certificate 消息。Certificate 消息会带上 Server 的证书链。Certificate 消息的目的一是为了验证 Server 身份，二是为了让 Client 根据协商出来的密码套件从证书中获取 Server 的公钥。Client 拿到 Server 的公钥和 Server 的 random 会生成预备主密钥。
- 由于密钥协商算法是 RSA，需要 Server 在发送完 Certificate 消息以后直接发送 ServerHelloDone 消息。
- Client 收到 ServerHelloDone 消息以后，会开始计算预备主密钥，计算出来的预备主密钥会经过 RSA/ECDSA 算法加密，并通过 ClientKeyExchange 消息发送给 Server。RSA 密码套件的预备主密钥是 48 字节。前 2 个字节是 client_version，后 46 字节是随机数。Server 收到 ClientKeyExchange 消息以后就会开始计算主密钥和会话密钥。同时 Client 也会在自己本地算好主密钥和会话密钥。
- Client 发送完 ClientKeyExchange 消息紧接着还会继续发送 ChangeCipherSpec 消息和 Finished 消息。Server 也会回应 ChangeCipherSpec 消息和 Finished 消息。如果 Finished 消息校验完成，代表握手最终成功。
- 握手完成后，客户端与服务器使用协商的会话密钥对后续数据进行加密传输。

1.4 认证机制

在该 VPN 客户端与服务端的实现中，采取了两种认证机制：使用 TLS 的证书认证对服务器进行身份认证以及使用 shadow 文件中的账户信息对客户端进行身份认证。

使用 TLS 的证书认证对服务器进行身份认证：过程见“1.3 加密原理”，可以使用 OpenSSL 提供的证书验证函数 SSL_CTX_set_verify()对服务器提供的证书添加验证规则，并且可以根据证书的验证结果在更细的粒度上控制验证结果。

使用 shadow 文件中的账户信息对客户端进行身份认证：在客户端与服务器建立 TLS 连接后，客户端为了证明自己的身份，向服务器发送验证身份所用的用户名和密码信息；服务器接收到这些信息后，使用 shadow.h 提供的 getspnam()函数根据用户名在系统的 shadow 文件中寻找该用户名所对应的密码的散列值，若用户名不存在则验证失败；若用户名存在，再使用 crypt.h 提供的 crypt()函数计算用户提供的密码在相同算法下的散列值，将二者进行比较，若结果相同则说明用户认证成功，即可完成对客户端的身份认证。

2 VPN 系统设计

2.1 概要设计

VPN 客户端的工作流程如图 2-1 所示，采取单进程单线程的工作方式。客户端在 VPN 连接正式建立后（即经过 TCP 连接、TLS 连接、TLS 服务器认证、客户端身份认证、协商虚 IP、配置本地网络后），本机应用程序产生的发往内部网络的数据包将被发送到 TUN，由 VPN 客户端程序从 TUN 中读取并通过 SSL 连接将该数据包封装并发往 VPN 服务器；从内部网络发往本机的数据包将由 VPN 服务器通过 SSL 连接发到该 VPN 客户端，由 VPN 客户端程序从 SSL 连接中读取并发送到 TUN 中，最终通过内核发往目标应用。因此，VPN 客户端只需监听 TUN 中传输来的数据和 SSL 接受到的数据即可，因此采用了使用 select 监听多输入接口的方案，直到 SSL 连接断开或发生其他错误时程序终止。

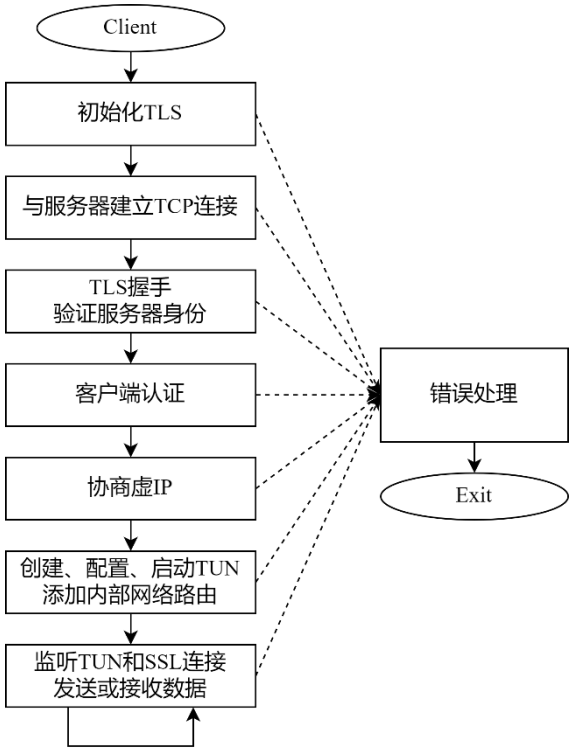

图 2-1 VPN 客户端的工作流程

VPN 服务器的主进程的工作流程如图 2-2 所示，采取多进程多线程的工作方式：主进程负责初始化工作与读取 TUN 中数据并分配到对应的命名管道，并不断等待客户端的连接，每当有新的客户端与服务器建立连接时，便开启新的进程与客户端为客户端提供服务。

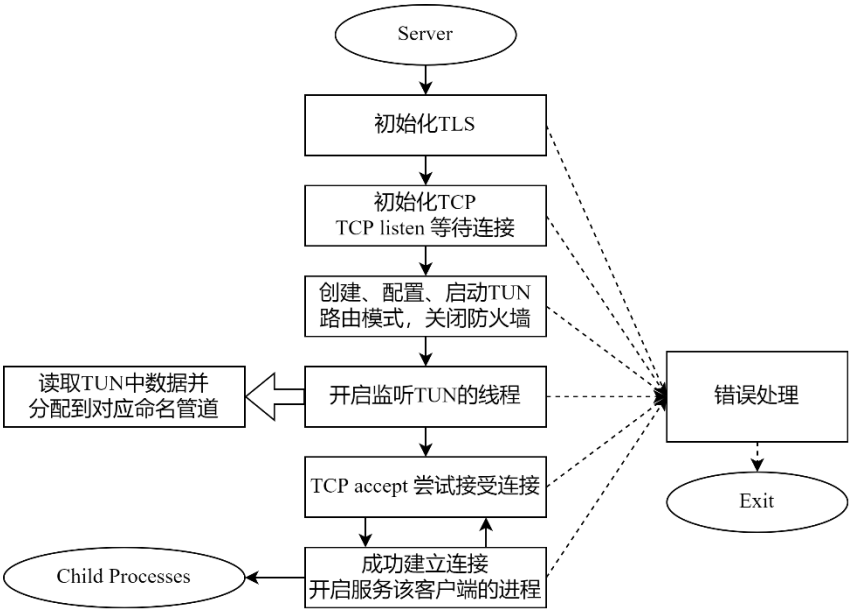

图 2-2 服务器主进程的工作流程

每当有新的客户端与服务器建立连接后，服务器将开启新的子进程向客户端提供服务，一个子进程的工作流程如图 2-3 所示，它负责完成连接建立的后半部分工作（包括建立 TLS 连接、客户端身份认证、协商虚 IP 并创建对应管道等），并完成与 VPN 客户端相似的工作：该客户端发往内部主机的数据包将通过 SSL 连接发送给 VPN 服务端的对应子进程，因此子进程需要将数据包提取并发往 TUN，进而经过内核的路由表发向内部主机；内部主机向 VPN 客户端发送的数据包将经过主进程的处理写入到对应的命名管道中，因此子进程需要将数据包从命名管道中读取并通过 SSL 发送给 VPN 客户端。

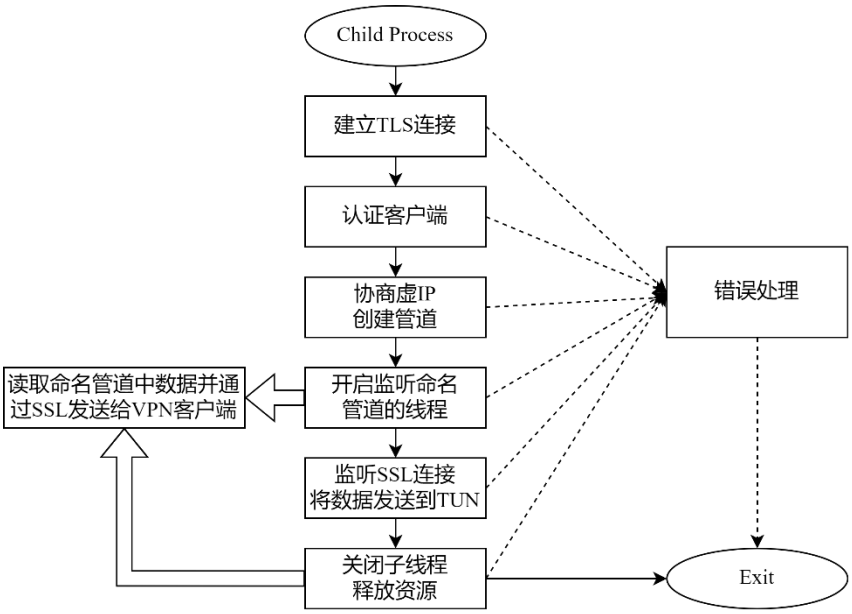

图 2-3 服务器子进程的工作流程

为了完成多客户端服务，使用了命名管道功能，为每个客户端根据其虚 IP 创建一个命名

管道。对于 VPN 客户端发送到内部网络主机的数据包，只需经 SSL 连接由 VPN 服务器将原始数据包发送到 TUN 中即可通过内核路由到响应的目标主机；对于内部网络主机发往外部网络 VPN 客户端的数据包，VPN 服务器主进程在从 TUN 中得到这个数据包后，将根据其目的地址的不同，将其写入相应的命名管道，由负责与目标外部主机进行通信的子进程读取并通过 SSL 发送给目标外部主机。多客户端的 VPN 服务器架构中内部网络主机向外部不同主机发送数据包的过程如图 2-4 所示。

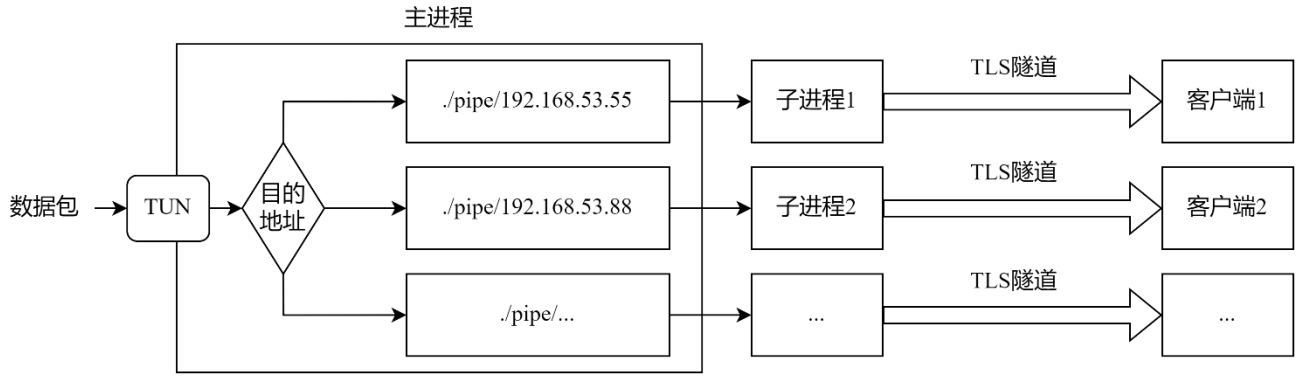

图 2-4 多客户端的 VPN 服务器架构

2.2 详细设计

2.2.1 认证模块

在客户端对服务器通过证书进行认证的回调函数见代码段 2-1，该函数输出证书的信息并对证书进行验证，忽略不严重的错误（如主机名不匹配）[ii]。

代码段 2-1 对服务器证书进行认证

```
int verify_callback(int preverify_ok, X509_STORE_CTX* x509_ctx) {
   char buf[300];
   X509* cert = X509_STORE_CTX_get_current_cert(x509_ctx);
   X509_NAME_oneline(X509_get_subject_name(cert), buf, 300);
   printf("[~] subject: %s.\n", buf);

   if (preverify_ok == 1) {
      printf("[+] Verification passed.\n");
   } else {
      int err = X509_STORE_CTX_get_error(x509_ctx);
      printf("[~] Verification error: %s.\n", X509_verify_cert_error_string(err));
      switch (err) {
```

```
        case X509_V_ERR_HOSTNAME_MISMATCH:
          printf("[+] Verification error ignored!\n");
          preverify_ok = 1;
          break;
        default:
          printf("[-] Verification failed: %s.\n", X509_verify_cert_error_string(err));
      }
    }
    return preverify_ok;
}
```

在服务器使用 shadow 文件中的账户信息对客户端进行身份认证见代码段 2-2，若认证失败则返回 AUTHENTICATION_ERROR，若认证成功则返回 NO_ERROR，分别输出相应的提示信息。

代码段 2-2 使用 shadow 文件中的账户信息对客户端进行身份认证

```
int login(char* username, char* password) {
   struct spwd* pw = getspnam(username);
   CHK_EXP_STR_RET_VAL(pw == NULL, "[-] Login failed: no such user.\n",
AUTHENTICATION_ERROR);

   printf("[~] Username : %s\n", pw->sp_namp);
   printf("[~] Password [Enc] : %s\n", pw->sp_pwdp);
   char* epasswd = crypt(password, pw->sp_pwdp);
   CHK_EXP_STR_RET_STR(strcmp(epasswd, pw->sp_pwdp), "[-] Login failed: wrong
password.\n",
AUTHENTICATION_ERROR, "[+] Login done!\n");

   return NO_ERROR;
}
```

2.2.2 命名管道实现的多客户端服务

多客户端服务中，对客户端向内网主机发送的数据包，其处理流程较为简单，只需将其通过 TUN 交给内核的路由表处理即可，此处不再赘述；而对内网主机向客户端发送的数据包，需要通过数据包的目的地址将其写入相应的命名管道，交由相应的进程处理。将内网主机向客户端发送的数据包写入相应命名管道的函数见代码段 2-3。

代码段 2-3 将数据包写入相应命名管道

```
void* TUN2PIPE(void* tunfd_a) {
  int tunfd = *((int*)tunfd_a);
  char buf[BUF_SIZE];
  while (1) {
    fd_set readFDSet;
    FD_ZERO(&readFDSet);
    FD_SET(tunfd, &readFDSet);
    select(tunfd + 1, &readFDSet, NULL, NULL, NULL);
    if (FD_ISSET(tunfd, &readFDSet)) {
      int len = read(tunfd, buf, BUF_SIZE);
      if (len != -1) {
        // Check if this packet has an unbroken IPv4 header.
        if (len > 0 && buf[0] == 0x45 && len >= 20) {
          unsigned int dstIP = (unsigned char)buf[19];
          char pipeFileName[64];
          // Write data to pipe.
          snprintf(pipeFileName, 64, "./pipe/" VPN_VIRTUAL_IP_POOL "%u", dstIP);
          int pipefd = open(pipeFileName, O_WRONLY);
          if (pipefd != -1) {
            write(pipefd, buf, len);
            printf("[o] TUN => ./pipe/" VPN_VIRTUAL_IP_POOL "%u.\n", dstIP);
          } else {
            printf("[-] Write packet to pipe file ./pipe/" VPN_VIRTUAL_IP_POOL
"%u failed! (%d: %s)\n", dstIP, errno, strerror(errno));
          }
        } else {
          // printf("[-] Receive from TUN done, but a wrong packet!\n");
        }
      } else {
        printf("[-] Receive from TUN failed! (%d: %s)\n", errno, strerror(errno));
      }
    }
  }
}
```

2.2.3 VPN 客户端使用 select 监听多输入接口

VPN 客户端只需监听 TUN 输入的数据和 SSL 连接接收到的数据，可以使用 select 同时监听这两个输入接口，在监听到输入后通过 FD_ISSET 判断输入接口是否接收到了数据，然后再分别对输入数据进行处理。VPN 客户端使用 select 监听多输入接口的代码见代码段 2-4。

代码段 2-4 VPN 客户端使用 select 监听多输入接口

```
TUN_SSL tun_ssl = {tunfd, ssl};
struct tcp_info info;
int len = sizeof(info);

while (1) {
   getsockopt(sockfd, IPPROTO_TCP, TCP_INFO, &info, (socklen_t*)&len);
   if (info.tcpi_state == TCP_ESTABLISHED) {
      fd_set fdset;
      FD_ZERO(&fdset);
      FD_SET(sockfd, &fdset);
      FD_SET(tunfd, &fdset);
      select((sockfd > tunfd ? sockfd : tunfd) + 1, &fdset, NULL, NULL, NULL);
      if (FD_ISSET(sockfd, &fdset)) { SSL2TUN(tun_ssl); }
      if (FD_ISSET(tunfd, &fdset)) { TUN2SSL(tun_ssl); }
   } else {
      printf("Connection closed by server.\n");
      SSL_shutdown(ssl);
      SSL_free(ssl);
      close(sockfd);
      return 0;
   }
}
```

3 VPN 实现细节

（1） 你的 VPN 客户端虚 IP 分别是如何获得的？是手动还是自动配置到 tun 虚网卡上的？自动配置如何实现？

① 在外网主机启动 VPN 客户端时，与 VPN 服务器的连接建立成功并互相验证对方的身份通过后，VPN 客户端将询问用户所期待使用的虚 IP 地址，并将用户输入发送至 VPN 服务器。VPN 服务器收到用户所期待使用的虚 IP 地址后，尝试为该用户分配该 IP：若该虚 IP 地址未被占用，则分配给该用户，并将该地址返回给 VPN 客户端；若该虚 IP 被占用，则在虚 IP 池中寻找其他虚 IP，若找到了其他可用虚 IP 则将其分配给该 VPN 客户端并将该地址返回给 VPN 客户端，否则向 VPN 客户端返回-1，表示目前没有可用的虚 IP，无法继续进行后续通信。

② 是自动配置到 tun 虚网卡上的。

③ 在经过以上虚 IP 的协商过程后,VPN 服务器会向 VPN 客户端返回分配给客户端的虚 IP，若此 IP 可用（即不为-1），则 VPN 客户端使用该虚 IP 作为 tun0 的 IP，并使用 system()函数执行 sudo ifconfig tun0 {虚 IP}/24 up 指令，配置并启动 tun0 虚网卡。

（2） 你的 VPN 客户端到内部子网的路由，是手动还是自动配置的？自动配置如何实现？

① 是自动配置的。

② 在完成建立 TLS 连接、验证服务器和客户端身份、协商并配置客户端虚 IP 后，VPN 客户端使用 system()函数执行 sudo route add -net {内部子网} tun0 指令，指明将发往内部子网的数据包发送到 tun0 虚网卡。

（3） 你的 VPN 客户端用户名口令认证是在隧道协商的什么阶段实现的？认证失败时双方是如何处理的？

① 是在 TLS 连接建立后，协商虚 IP 之前实现的。

② 认证失败时，VPN 服务器向 VPN 客户端返回“[-] Login failed!”信息并关闭连接，VPN 客户端在收到这一信息后，向用户输出认证失败的提示信息并关闭连接。若需要对 VPN 客户端认证做出优化，可以考虑在认证失败后开启新一轮的认证，提示用户输入新的认证信息，并再次交给服务器进行认证，在认证失败达到一定次数后再断开连接。

（4） 你的 VPN 服务器支持多客户端采用的多进程还是多线程架构？进程间通信采用的何种技术？VPN 服务器收到内网主机的应答报文时，如何判断应发送给哪个 VPN 客户端的隧

道？

① 多进程架构，每个客户端对应一个进程，对于每个客户端采取多线程提供服务。

② 进程间通信采用了命名管道技术，为每个客户端创建一个唯一的命名管道。

③ VPN 服务器收到内网主机的应答报文时，首先判断该报文是否具有完整的 IP 头，判断标准是报文的第一个字节为 0x45(表示 IPv4 且头部具有 20 字节)且报文长度大于 20，若不满足该条件则将其丢弃，若满足该条件则读取其 IP 头部的最后一个字节（表示目的地址的最后 8 位，由于在该设计中虚 IP 组成的虚拟子网的子网掩码为/24，因此只靠 IP 的后 8 位即可区分目标主机），并根据最后一个字节将该数据包写入该主机对应的命名管道，进而发送给对应的目标主机。

（5） 你的 VPN 客户端退出时，VPN 服务器如何发现？如何处理？

① VPN 客户端退出时，VPN 服务器中该 VPN 客户端对应的进程中监听该 SSL 连接的线程将会检测到 SSL 连接断开并返回，从而发现 VPN 客户端已退出。

② 由于监听 SSL 连接的线程是主线程，他将停止其他线程，输出相关提示信息并清理与该 VPN 客户端相关联的命名管道，最后结束该 VPN 客户端对应的进程。

（6） 你的 VPN 服务器退出时，VPN 客户端如何发现？如何处理？

① VPN 客户端在启动配置完毕后便一直等待并处理 tun0 中需要发送的数据和 SSL 中接收到的数据，在每次接收数据前，VPN 客户端都会通过 getsockopt()函数获取当前与服务器的 TCP 连接的状态。正常情况下，TCP 连接建立后状态应为 ESTABLISHED，因此只要当前状态不是 ESTABLISHED，即可发现 VPN 服务器退出，连接断开。

② VPN 客户端将清理相关资源、输出相关提示信息并退出。

（7） 请介绍你的设计实现中个人最满意的特色部分。

① 丰富的提示信息，包括运行状态、错误提示等；

② 美观的数据包内容输出，可以直观地查看数据包中的内容；

③ 虚 IP 支持指定和自动分配，提高了系统的可用性和灵活性。

4 测试结果与分析

4.1 测试环境搭建

（1）搭建图 1-1 所示网络拓扑所执行的指令及执行结果如下：

```
# 创建 docker 网络 extranet
root@VM:/$ sudo docker network create --subnet=10.0.2.0/24 --gateway=10.0.2.8 --opt
"com.docker.network.bridge.name"="docker1" extranet
# 创建 docker 网络 intranet
root@VM:/$ sudo docker network create --subnet=192.168.60.0/24 --gateway=192.168.60.1 --
opt "com.docker.network.bridge.name"="docker2" intranet

# 创建并运行容器 HostU 并删除默认路由
[05/06/22]seed@VM:~$ sudo docker run -it --name=HostU --hostname=HostU --net=extranet --
ip=10.0.2.7 --privileged "seedubuntu" /bin/bash
root@HostU:/# route del default
root@HostU:/# cd home
root@HostU:/home# mkdir xba
root@HostU:/home# cd xba
# 创建并运行容器 HostU2 并删除默认路由
[05/06/22]seed@VM:~$ sudo docker run -it --name=HostU2 --hostname=HostU2 --net=extranet --
ip=10.0.2.27 --privileged "seedubuntu" /bin/bash
root@HostU2:/# route del default
root@HostU2:/# cd home
root@HostU2:/home# mkdir xba
root@HostU2:/home# cd xba
# 创建并运行容器 HostV 并删除默认路由
[05/06/22]seed@VM:~$ sudo docker run -it --name=HostV --hostname=HostV --net=intranet --
ip=192.168.60.101 --privileged "seedubuntu" /bin/bash
root@HostV:/# route del default
root@HostV:/# route add -net 192.168.53.0/24 gw 192.168.60.1
root@HostV:/# sudo /etc/init.d/openbsd-inetd restart
```

（2）证书的生成与配置见附录 1。

（3）程序的编译及复制：

```
# 将源程序 vpn_server.c vpn_client.c vpn.h 及 Makefile 放置在 miniVPN 文件夹下
# 运行脚本 cp.sh 即可编译并将文件复制到 docker 容器中的 /home/xba 中
[06/12/22]seed@VM:~/.../miniVPN$ ./cp.sh
gcc -o vpn_client vpn_client.c -lssl -lcrypto
gcc -o vpn_server vpn_server.c -lssl -lcrypto -lcrypt -lpthread
```

4.2 认证服务器

首先启动各台主机并进行准备工作：

```
root@HostV:/# route del default
root@HostV:/# route add -net 192.168.53.0/24 gw 192.168.60.1
root@HostV:/# sudo /etc/init.d/openbsd-inetd restart
 * Restarting internet superserver inetd                                  [ OK ]
root@HostV:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.53.0    192.168.60.1    255.255.255.0   UG    0      0        0 eth0
192.168.60.0    *               255.255.255.0   U     0      0        0 eth0
```

图 4-1 开启 docker 内网主机 HostV 并配置路由与 telnet

```
[06/12/22]seed@VM:~/...miniVPN$ sudo ./vpn_server
[sudo] seed 的密码:
Enter PEM pass phrase:
[+] Setup TLS server done!
[+] Setup TCP server done!
[~] listen_sock = 3.
[+] Setup TUN interface done!
[~] Creating and upping TUN ...
[+] Creating and upping TUN done!
[~] Set ip_forward and clear firewall ...
[+] Set ip_forward and clear firewall done!
```

图 4-2 在虚拟机主机开启 VPN 服务器

```
127.0.0.1       localhost

10.0.2.8        XuBiang_Server

::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.0.2.7        HostU
```

图 4-3 开启 docker 外网主机 HostU 并配置 HOSTS

启动完成后，在 HostU 分别正常连接 VPN 服务器和修改时间到证书过期后连接 VPN 服务器，如图 4-4 和 4-5 所示：正常连接时可以与 VPN 服务器建立连接，提示进行身份认证；修改时间到证书过期后在 TLS 握手阶段证书认证失败，因此 TLS 连接建立失败。

```
root@HostU:/home/xba/miniVPN# ./vpn_client
[~] Server IP: XuBiang_Server
[~] Server PORT: 4433
Enter PEM pass phrase:
[+] Setup TLS client done!
[+] Setup TCP client done!
[~] subject: /C=CN/ST=HuBei/L=WuHan/O=HUST_CSE/OU=CSE_1902/CN=XuBiang.
[+] Verification passed.
[~] subject: /C=CN/ST=HuBei/O=HUST_CSE/OU=CSE_1902/CN=XuBiang_Server.
[+] Verification passed.
[+] SSL connection done, using AES256-GCM-SHA384.
[~] Username:
```

图 4-4 在 HostU 连接 VPN 服务器，正常连接

```
root@HostU:/home/xba/miniVPN# date -s "20230606"
Tue Jun  6 00:00:00 CST 2023
root@HostU:/home/xba/miniVPN# ./vpn_client
[~] Server IP: XuBiang_Server
[~] Server PORT: 4433
Enter PEM pass phrase:
[+] Setup TLS client done!
[+] Setup TCP client done!
[~] subject: /C=CN/ST=HuBei/L=WuHan/O=HUST_CSE/OU=CSE_1902/CN=XuBiang.
[~] Verification error: certificate has expired.
[-] Verification failed: certificate has expired.
[-] SSL connect failed.
3079694016:error:14090086:SSL routines:ssl3_get_server_certificate:certificate verify fa
iled:s3_clnt.c:1264:
root@HostU:/home/xba/miniVPN#
```

图 4-5 修改时间至服务器证书过期后再次连接 VPN 服务器，认证失败

4.3 认证 VPN 客户端

在认证服务器完成后，若服务器认证通过，则提示客户端输入用户名和密码进行身份认证。输入错误和正确的的用户名密码对见图 4-6 及 4-7。

```
root@HostU:/home/xba/miniVPN# ./vpn_client
[~] Server IP: XuBiang_Server
[~] Server PORT: 4433
Enter PEM pass phrase:
[+] Setup TLS client done!
[+] Setup TCP client done!
[~] subject: /C=CN/ST=HuBei/L=WuHan/O=HUST_CSE/OU=CSE_1902/CN=XuBiang.
[+] Verification passed.
[~] subject: /C=CN/ST=HuBei/O=HUST_CSE/OU=CSE_1902/CN=XuBiang_Server.
[+] Verification passed.
[+] SSL connection done, using AES256-GCM-SHA384.
[~] Username: seed
[~] Password(No Echoing):
[+] SSL write authentication done!
[+] Authentication result: [-] Login failed!
[-] Authentication failed, connection shutdown!
```

图 4-6 输入错误的用户名密码对，认证失败，连接关闭

```
root@HostU:/home/xba/miniVPN# ./vpn_client
[~] Server IP: XuBiang_Server
[~] Server PORT: 4433
Enter PEM pass phrase:
[+] Setup TLS client done!
[+] Setup TCP client done!
[~] subject: /C=CN/ST=HuBei/L=WuHan/O=HUST_CSE/OU=CSE_1902/CN=XuBiang.
[+] Verification passed.
[~] subject: /C=CN/ST=HuBei/O=HUST_CSE/OU=CSE_1902/CN=XuBiang_Server.
[+] Verification passed.
[+] SSL connection done, using AES256-GCM-SHA384.
[~] Username: seed
[~] Password(No Echoing):
[+] SSL write authentication done!
[+] Authentication result: [+] Login successfully!
[~] Your expected IP: 192.168.53.11
[+] SSL send expected IP 192.168.53.11 done!
[+] SSL negotiate virtual IP done: 192.168.53.11!
[+] Setup TUN interface done!
[~] Creating and upping TUN ...
[+] Creating and upping TUN done!
```

图 4-7 输入正确的用户名密码对，认证成功

4.4 加密隧道通信

在 HostU 通过 VPN 客户端连接到内网后，在 HostU 执行 ping 192.168.60.101 -c 2，并用 Wireshark 在 VPN 服务器的外端 10.0.2.8 进行监听，程序的执行结果如图 4-8 所示，VPN 客户端的输出如图 4-9 所示（仅为一次 ping 的往返报文，VPN 服务器同），VPN 服务器的输出如图 4-10 所示，Wireshark 截取的数据包如图 4-11 所示。观察图 4-11 中的数据包，数据部分与 VPN 客户端和服务器输出的数据并不一致，且 Wireshark 显示为 TLSv1.2 封装。

```
root@HostU:/# ping 192.168.60.101 -c 2
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=1.78 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=2.19 ms

--- 192.168.60.101 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1050ms
rtt min/avg/max/mdev = 1.783/1.986/2.190/0.208 ms
root@HostU:/#
```

图 4-8 建立 VPN 隧道后连接内网主机

```
[o] TUN ⇒ SSL:
(192.168.53.11 ⇒ 192.168.60.101)
45 00 00 54 be 4e 40 00 40 01 89 99 c0 a8 35 0b E··T·N@·@·····5·
c0 a8 3c 65 08 00 f0 87 00 b3 00 01 0a f0 a4 62 ··<e···········b
61 6e 0b 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 an··············
14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 ···········  !"#
24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 $%&'()*+,-./0123
34 35 36 37 00 00 00 00 00 00 00 00 00 00 00 00 4567············
[i] SSL ⇒ TUN:
(192.168.60.101 ⇒ 192.168.53.11)
45 00 00 54 41 2d 00 00 3f 01 47 bb c0 a8 3c 65 E··TA-··?·G···<e
c0 a8 35 0b 00 00 f8 87 00 b3 00 01 0a f0 a4 62 ··5···········b
61 6e 0b 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 an··············
14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 ···········  !"#
24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 $%&'()*+,-./0123
34 35 36 37 00 00 00 00 00 00 00 00 00 00 00 00 4567············
```

图 4-9 VPN 客户端展示的数据包内容

```
[i] SSL ⇒ TUN:
(192.168.53.11 ⇒ 192.168.60.101)
45 00 00 54 be 4e 40 00 40 01 89 99 c0 a8 35 0b E··T·N@·@·····5·
c0 a8 3c 65 08 00 f0 87 00 b3 00 01 0a f0 a4 62 ··<e···········b
61 6e 0b 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 an··············
14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 ···········  !"#
24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 $%&'()*+,-./0123
34 35 36 37 00 00 00 00 00 00 00 00 00 00 00 00 4567············
[o] TUN ⇒ ./pipe/192.168.53.11.
[o] PIPE ./pipe/192.168.53.11 ⇒ SSL:
(192.168.60.101 ⇒ 192.168.53.11)
45 00 00 54 41 2d 00 00 3f 01 47 bb c0 a8 3c 65 E··TA-··?·G···<e
c0 a8 35 0b 00 00 f8 87 00 b3 00 01 0a f0 a4 62 ··5···········b
61 6e 0b 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 an··············
14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 ···········  !"#
24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 $%&'()*+,-./0123
34 35 36 37 00 00 00 00 00 00 00 00 00 00 00 00 4567············
```

图 4-10 VPN 服务器展示的数据包内容

```
   21 2022-06-12 03:42:02.749333106  10.0.2.7    10.0.2.8    TLSv1.2   179 Application Data
   22 2022-06-12 03:42:02.750775506  10.0.2.8    10.0.2.7    TLSv1.2   179 Application Data
   23 2022-06-12 03:42:02.750870306  10.0.2.7    10.0.2.8    TCP        66 53366 → 4433 [ACK] Seq=3187808971
   24 2022-06-12 03:42:03.800188456  10.0.2.7    10.0.2.8    TLSv1.2   179 Application Data
▸ Frame 21: 179 bytes on wire (1432 bits), 179 bytes captured (1432 bits) on interface 0
▸ Ethernet II, Src: 02:42:0a:00:02:07 (02:42:0a:00:02:07), Dst: 02:42:9d:f4:d8:4c (02:42:9d:f4:d8:4c)
▸ Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
▸ Transmission Control Protocol, Src Port: 53366, Dst Port: 4433, Seq: 3187808858, Ack: 2608472185, Len: 113
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 108
      Encrypted Application Data: 57f48c221ea71fc6360413182f9f8a96d0d4145cdea84d3e...
0000  02 42 9d f4 d8 4c 02 42  0a 00 02 07 08 00 45 00   ·B···L·B ····· E·
0010  00 a5 d8 c4 40 00 40 06  49 80 0a 00 02 07 0a 00   ····@·@· I·····
0020  02 08 d0 76 11 51 be 02  1a 5a 9b 7a 20 79 80 18   ···v·Q·· ·Z·z y··
0030  01 19 18 a6 00 00 01 01  08 0a 03 1c db 68 03 1c   ···· ···· ·····h··
0040  ce 10 17 03 03 00 6c 57  f4 8c 22 1e a7 1f c6 36   ·· ····lW ··"····6
0050  04 13 18 2f 9f 8a 96 d0  d4 14 5c de a8 4d 3e 3b   ···/···· ··\··M>;
0060  6a 30 e2 19 98 d8 24 95  2f cb 90 d0 17 2d 7c bb   j0····$· /····-|·
0070  38 52 f2 1d 21 86 2c 92  92 1e be ab 87 01 3f d8   8R··!·,· ·····?·
0080  50 75 5a 53 52 8f 99 3d  fd ce cb b6 77 34 3f 1c   PuZSR··= ····w4?·
0090  5e 13 b3 51 7f d7 85 64  11 be 17 34 0b f2 9b 0e   ^··Q···d ···4····
00a0  84 c9 cf 03 49 71 a8 5c  a7 ac 6c 66 52 4b 80 86   ····Iq·\ ··lfRK··
00b0  67 e4 85                                          g··
```

图 4-11 经过 TLS 隧道封装的数据

4.5 支持多客户端

首先开启两台客户端主机 HostU 和 HostU2，分别开启 VPN 客户端并与 VPN 服务器建立连接以连入内网，然后再两台外网主机上分别运行 telnet 192.168.60.101 远程连接到内网主机，同时连接到内网主机后 HostU 和 HostU2 在内网主机查看网络连接状态，均可以看到两台远程主机分别以 192.168.53.11 和 192.168.53.12 连接到内网主机，如图 4-12 及 4-13 所示。

```
root@HostU:/# telnet 192.168.60.101
Trying 192.168.60.101 ...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
HostV login: seed
Password:
Last login: Sun Jun 12 00:08:25 CST 2022 from 192.168.53.12 on pts/1
sh: 1: cannot create /run/motd.dynamic.new: Directory nonexistent
[06/12/22]seed@HostV:~$ sudo netstat -tanlp | grep telnetd
[sudo] password for seed:
tcp        0      0 192.168.60.101:23       192.168.53.12:60690     ESTABLISHED 99/in.te
lnetd: 192.
tcp        0      0 192.168.60.101:23       192.168.53.11:56704     ESTABLISHED 119/in.t
elnetd: 192
[06/12/22]seed@HostV:~$
```

图 4-12 HostU 和 HostU2 均连接至内网主机后在 HostU 查看内网主机的网络连接

```
root@HostU2:/# telnet 192.168.60.101
Trying 192.168.60.101 ...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
HostV login: seed
Password:
Last login: Sun Jun 12 00:05:19 CST 2022 from 192.168.53.11 on pts/0
sh: 1: cannot create /run/motd.dynamic.new: Directory nonexistent
[06/12/22]seed@HostV:~$ sudo netstat -tanlp | grep telnetd
[sudo] password for seed:
tcp        0      0 192.168.60.101:23       192.168.53.12:60690     ESTABLISHED 99/in.te
lnetd: 192.
tcp        0      0 192.168.60.101:23       192.168.53.11:56704     ESTABLISHED 119/in.t
elnetd: 192
[06/12/22]seed@HostV:~$
```

图 4-13 HostU 和 HostU2 均连接至内网主机后在 HostU2 查看内网主机的网络连接

之后退出 HostU2 与 HostV 的远程连接，如图 4-14 所示，再在 HostU 检查 telnet 连接未断开且正常通信，查看网络状态仅剩自己 192.168.53.11，如图 4-15 所示。

```
[06/12/22]seed@HostV:~$ logout
Connection closed by foreign host.
root@HostU2:/#
```

图 4-14 HostU2 退出与 HostV 的远程连接

```
root@HostU:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
HostV login: seed
Password:
Last login: Sun Jun 12 00:08:25 CST 2022 from 192.168.53.12 on pts/1
sh: 1: cannot create /run/motd.dynamic.new: Directory nonexistent
[06/12/22]seed@HostV:~$ sudo netstat -tanlp | grep telnetd
[sudo] password for seed:
tcp        0      0 192.168.60.101:23       192.168.53.12:60690     ESTABLISHED 99/in.te
lnetd: 192.
tcp        0      0 192.168.60.101:23       192.168.53.11:56704     ESTABLISHED 119/in.t
elnetd: 192
[06/12/22]seed@HostV:~$ sudo netstat -tanlp | grep telnetd
tcp        0      0 192.168.60.101:23       192.168.53.11:56704     ESTABLISHED 119/in.t
elnetd: 192
[06/12/22]seed@HostV:~$
```

图 4-15 HostU 退出与 HostV 的远程连接依然正常

4.6 易用性和稳定性

在 HostU 连接服务器，HostU 的客户端程序输出如图 4-16 所示，请求分配的 IP 为 192.168.53.11 且成功分配到了该 IP。启动成功后查看 HostU 的网卡配置和路由表可知 TUN0 已被正确配置启动、路由已正确添加，如图 4-17 所示。

```
root@HostU:/home/xba/miniVPN# ./vpn_client
[~] Server IP: XuBiang_Server
[~] Server PORT: 4433
Enter PEM pass phrase:
[+] Setup TLS client done!
[+] Setup TCP client done!
[~] subject: /C=CN/ST=HuBei/L=WuHan/O=HUST_CSE/OU=CSE_1902/CN=XuBiang.
[+] Verification passed.
[~] subject: /C=CN/ST=HuBei/O=HUST_CSE/OU=CSE_1902/CN=XuBiang_Server.
[+] Verification passed.
[+] SSL connection done, using AES256-GCM-SHA384.
[~] Username: seed
[~] Password(No Echoing):
[+] SSL write authentication done!
[+] Authentication result: [+] Login successfully!
[~] Your expected IP: 192.168.53.11
[+] SSL send expected IP 192.168.53.11 done!
[+] SSL negotiate virtual IP done: 192.168.53.11!
[+] Setup TUN interface done!
[~] Creating and upping TUN ...
[+] Creating and upping TUN done!
[~] Adding route ...
[+] Adding route done!
```

4-16 VPN 客户端成功连接至服务器

```
root@HostU:/# ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.53.11  P-t-P:192.168.53.11  Mask:255.255.255.0
          inet6 addr: fe80::1726:e774:285f:c516/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:144 (144.0 B)

root@HostU:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        *               255.255.255.0   U     0      0        0 eth0
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
192.168.60.0    *               255.255.255.0   U     0      0        0 tun0
root@HostU:/#
```

图 4-17 HostU 的 TUN0 被成功配置启动且路由被正确添加

在 HostU2 连接 VPN 服务器并仍然请求分配 192.168.53.11，可见由于地址被占用，最终协商成功的地址是 192.168.53.2，如图 4-18 所示。启动成功后查看 HostU2 的网卡配置和路由表可知 TUN0 已被正确配置启动、路由已正确添加，如图 4-19 所示。

```
root@HostU2:/home/xba/miniVPN# ./vpn_client
[~] Server IP: XuBiang_Server
[~] Server PORT: 4433
Enter PEM pass phrase:
[+] Setup TLS client done!
[+] Setup TCP client done!
[~] subject: /C=CN/ST=HuBei/L=WuHan/O=HUST_CSE/OU=CSE_1902/CN=XuBiang.
[+] Verification passed.
[~] subject: /C=CN/ST=HuBei/O=HUST_CSE/OU=CSE_1902/CN=XuBiang_Server.
[+] Verification passed.
[+] SSL connection done, using AES256-GCM-SHA384.
[~] Username: seed
[~] Password(No Echoing):
[+] SSL write authentication done!
[+] Authentication result: [+] Login successfully!
[~] Your expected IP: 192.168.53.11
[+] SSL send expected IP 192.168.53.11 done!
[+] SSL negotiate virtual IP done: 192.168.53.2!
[+] Setup TUN interface done!
[~] Creating and upping TUN ...
[+] Creating and upping TUN done!
[~] Adding route ...
[+] Adding route done!
```

4-18 VPN 客户端成功连接至服务器且重新协商了虚 IP

```
root@HostU2:/# ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.53.2  P-t-P:192.168.53.2  Mask:255.255.255.0
          inet6 addr: fe80::5de4:fade:2d2:54a5/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:144 (144.0 B)

root@HostU2:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        *               255.255.255.0   U     0      0        0 eth0
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
192.168.60.0    *               255.255.255.0   U     0      0        0 tun0
root@HostU2:/#
```

图 4-19 HostU2 的 TUN0 被成功配置启动且路由被正确添加

在以上过程中，VPN 服务器的输出如图 4-20 及 4-21 所示，可以看到虚 IP 的协商过程：HostU 申请的 IP 没有被占用，因此 VPN 服务器直接将该 IP 分配给该客户端使用；而 HostU2 申请的 IP 已被 HostU 占用，因此 VPN 服务器在 IP 池中分配了另一个可用的 IP 给 HostU2 使用。

```
[~] Accept sock = 5.
[+] Client real IP: 10.0.2.7.
[+] SSL set fd done!
[+] SSL accept done!
[+] SSL connection established!
[+] SSL read username done!
[+] SSL read password done!
[~] Username : seed
[~] Password [Enc] : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvD
eCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/
[+] Login done!
[+] SSL send authentication information done!
[+] Send read expected IP done!
[+] The IP client expected is 192.168.53.11!
[+] Make pipe ./pipe/192.168.53.11 done!
[+] Send negotiate virtual IP done!
[+] 10.0.2.7's virtual IP: 192.168.53.11
```

图 4-20 HostU 申请的 IP 未被占用，直接分配

```
[~] Accept sock = 5.
[+] Client real IP: 10.0.2.27.
[+] SSL set fd done!
[+] SSL accept done!
[+] SSL connection established!
[+] SSL read username done!
[+] SSL read password done!
[~] Username : seed
[~] Password [Enc] : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvD
eCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/
[+] Login done!
[+] SSL send authentication information done!
[+] Send read expected IP done!
[+] The IP client expected is 192.168.53.11!
[-] Make pipe failed! (17: File exists)
[~] Try to alloc free IP ...
[+] Make pipe ./pipe/192.168.53.2 done!
[+] Send negotiate virtual IP done!
[+] 10.0.2.27's virtual IP: 192.168.53.2
```

图 4-21 HostU2 申请的 IP 被占用，在内存池中分配 IP

5　体会与建议

5.1 心得体会

略

5.2 意见建议

略

6 附录 1：证书生成与配置

①配置文件及目录生成：

```
[05/13/22]seed@VM:~/.../miniVPN$ cp /usr/lib/ssl/openssl.cnf openssl.cnf
[05/13/22]seed@VM:~/.../miniVPN$ vi openssl.cnf
[05/13/22]seed@VM:~/.../miniVPN$ cat openssl.cnf
…
[ CA_default ]

dir        = ./CA_XBA      # Where everything is kept
certs      = $dir/certs        # Where the issued certs are kept
crl_dir        = $dir/crl      # Where the issued crl are kept
database   = $dir/index.txt    # database index file.
…
[05/13/22]seed@VM:~/.../miniVPN$ mkdir CA_XBA
[05/13/22]seed@VM:~/.../miniVPN$ mkdir CA_XBA/certs
[05/13/22]seed@VM:~/.../miniVPN$ mkdir CA_XBA/crl
[05/13/22]seed@VM:~/.../miniVPN$ mkdir CA_XBA/newcerts
[05/13/22]seed@VM:~/.../miniVPN$ touch CA_XBA/index.txt
[05/13/22]seed@VM:~/.../miniVPN$ touch CA_XBA/serial
[05/13/22]seed@VM:~/.../miniVPN$ echo 1000 >> CA_XBA/serial
```

②创建证书颁发机构（CA）：

```
# 为 CA 生成自签名证书，其证书将作为根证书
# 口令为 xba0327
[05/13/22]seed@VM:~/.../miniVPN$ sudo openssl req -new -x509 -keyout ca.key -out ca.crt -
config openssl.cnf
Generating a 2048 bit RSA private key
.........................+++
.............................................................+++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
…
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:HuBei
Locality Name (eg, city) []:WuHan
```

```
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST_CSE
Organizational Unit Name (eg, section) []:CSE_1902
Common Name (e.g. server FQDN or YOUR name) []:XuBiang
Email Address []:
```

③为服务器生成密钥对和证书签名请求：

```
# 口令为 xbaserver
[05/13/22]seed@VM:~/.../miniVPN$ openssl genrsa -des3 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.....................++++++
.............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:

# 生成证书签名请求（CSR）
[05/13/22]seed@VM:~/.../miniVPN$ openssl req -new -key server.key -out server.csr -config
openssl.cnf
Enter pass phrase for server.key:
…
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:HuBei
Locality Name (eg, city) []:WuHan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST_CSE
Organizational Unit Name (eg, section) []:CSE_1902
Common Name (e.g. server FQDN or YOUR name) []:XuBiang_Server
Email Address []:
…
```

④为客户端生成密钥对和证书签名请求：

```
# 密钥为 xbaclient
[05/13/22]seed@VM:~/.../miniVPN$ openssl genrsa -des3 -out client.key 1024
Generating RSA private key, 1024 bit long modulus
..................++++++
...........++++++
e is 65537 (0x10001)
Enter pass phrase for client.key:
```

```
Verifying - Enter pass phrase for client.key:


# 生成证书签名请求（CSR）
[05/13/22]seed@VM:~/.../miniVPN$ openssl req -new -key client.key -out client.csr -config
openssl.cnf
Enter pass phrase for client.key:
...
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:HuBei
Locality Name (eg, city) []:WuHan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST_CSE
Organizational Unit Name (eg, section) []:CSE_1902
Common Name (e.g. server FQDN or YOUR name) []:XuBiang_Client
Email Address []:
...
```

⑤CA 签发证书：

```
# 为服务器签发证书
[05/13/22]seed@VM:~/.../miniVPN$ openssl ca -in server.csr -out server.crt -cert ca.crt -
keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
       Serial Number: 4096 (0x1000)
       Validity
           Not Before: May 13 13:06:12 2022 GMT
           Not After : May 13 13:06:12 2023 GMT
       Subject:
           countryName             = CN
           stateOrProvinceName      = HuBei
           organizationName        = HUST_CSE
           organizationalUnitName   = CSE_1902
           commonName              = XuBiang_Server
       X509v3 extensions:
           X509v3 Basic Constraints:
```

```
            CA:FALSE
        Netscape Comment:
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            C1:C3:39:3E:22:E4:D2:8F:58:6D:50:DB:9F:2A:9D:76:FE:86:60:2D
        X509v3 Authority Key Identifier:
            keyid:E5:98:66:1C:1D:40:24:35:91:A5:56:31:0E:C2:D4:64:C2:41:67:C6

Certificate is to be certified until May 13 13:06:12 2023 GMT (365 days)
Sign the certificate? [y/n]:y



1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

# 为客户端签发证书
[05/13/22]seed@VM:~/.../miniVPN$ openssl ca -in client.csr -out client.crt -cert ca.crt -
keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
       Serial Number: 4097 (0x1001)
       Validity
           Not Before: May 13 13:07:18 2022 GMT
           Not After : May 13 13:07:18 2023 GMT
       Subject:
           countryName             = CN
           stateOrProvinceName      = HuBei
           organizationName         = HUST_CSE
           organizationalUnitName    = CSE_1902
           commonName              = XuBiang_Client
       X509v3 extensions:
           X509v3 Basic Constraints:
```

```
            CA:FALSE
        Netscape Comment:
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            5A:A1:64:66:13:C3:02:DA:49:C9:8D:A5:7F:EB:E7:DF:0A:EF:AA:CA
        X509v3 Authority Key Identifier:
            keyid:E5:98:66:1C:1D:40:24:35:91:A5:56:31:0E:C2:D4:64:C2:41:67:C6

Certificate is to be certified until May 13 13:07:18 2023 GMT (365 days)
Sign the certificate? [y/n]:y



1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

⑥证书配置：

```
[05/13/22]seed@VM:~/.../miniVPN$ mkdir cert_client
[05/13/22]seed@VM:~/.../miniVPN$ cp ca.crt cert_client/ca.crt
[05/13/22]seed@VM:~/.../miniVPN$ cd cert_client
[05/13/22]seed@VM:~/.../cert_client$ openssl x509 -in ca.crt -noout -subject_hash
918ac40f
[05/13/22]seed@VM:~/.../cert_client$ ln -s ca.crt 918ac40f.0
[05/13/22]seed@VM:~/.../cert_client$ ls -l
总用量 4
lrwxrwxrwx 1 seed seed   6 5月  13 21:27 918ac40f.0 -> ca.crt
-rw-r--r-- 1 seed seed 1314 5月  13 20:50 ca.crt
[05/13/22]seed@VM:~/.../cert_client$ cd ..
[05/13/22]seed@VM:~/.../miniVPN$ mv client.* cert_client/
[05/13/22]seed@VM:~/.../miniVPN$ mkdir cert_server
[05/13/22]seed@VM:~/.../miniVPN$ mv server.* cert_server/
[05/13/22]seed@VM:~/.../miniVPN$ cp ca.crt cert_server/ca.crt
```

7 参考文献

[i] https://halfrost.com/https_tls1-2_handshake/

[ii] https://www.openssl.org/docs/man1.0.2/man3/