

“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2002班 |
| **姓 名：** | 汪闻韵 |
| **学 号：** | U202012056 |
| **日 期：** | 2023年5月23日 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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](#_Toc26708)

[1.1 网络拓扑 1](#_Toc6205)

[1.2 通信机制 1](#_Toc10293)

[1.3 加密原理 3](#_Toc11553)

[1.4 认证机制 4](#_Toc1042)

[2 VPN系统设计 4](#_Toc397)

[2.1 系统概要 4](#_Toc10212)

[2.2 miniVPNserver模块设计 5](#_Toc10106)

[2.3 miniVPNclient详细设计 16](#_Toc9238)

[3 VPN实现细节 18](#_Toc661)

[4 测试结果与分析 20](#_Toc22223)

[4.1 连通性测试 20](#_Toc1264)

[4.2 安全性测试 21](#_Toc19537)

[4.3 稳定性测试 23](#_Toc19028)

[5 体会与建议 24](#_Toc14218)

[5.1 心得体会 24](#_Toc20247)

[5.2 意见建议 25](#_Toc27669)

# 实验原理

## 网络拓扑

在本次实验中，实验环境的网络拓扑如图1-1所示，图中展示了实验环境中的网络布局。

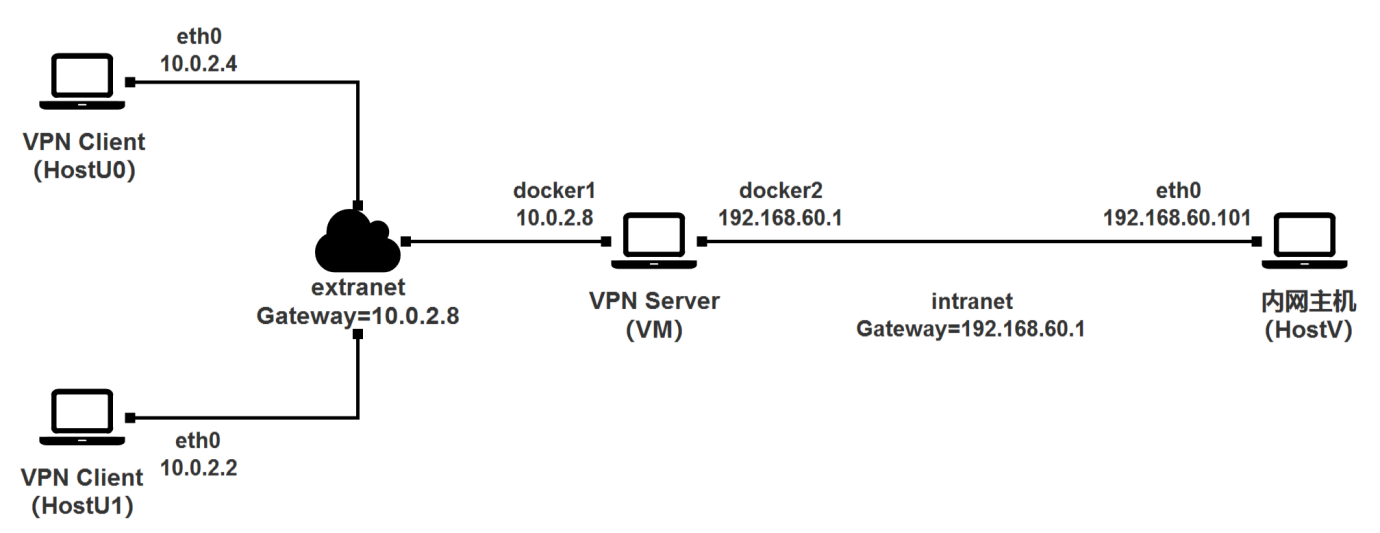


图1-1 实验环境网络拓扑图

实验环境中包括了HostU0和HostU1两台主机，它们分别运行着miniVPNclient软件。另外，我们还有一台名为VM的虚拟机，它上面安装了miniVPNserver软件。值得注意的是，为了模拟外网和内网的环境，我们使用了docker创建的虚拟网桥来实现extranet和intranet。初始情况下，HostU和HostV可以相互通信，无法真实模拟出HostU位于外网、HostV位于内网且彼此无法直接通信的情况。因此，我们需要在每台机器上执行命令"route del default"，以删除默认的路由表，使它们在形式上无法互通。同样地，HostV也需要进行相同的操作步骤来构建实验环境。

通信机制

本次实验中，我们使用了TUN技术来实现miniVPN，并设计了如图1-2所示的通信机制。

在图1-2中，描述了extranet中的主机向intranet中的主机发送数据的流程。 其中，图中的三个黄色方块表示三台独立的主机。

在HostU上，我们运行miniVPNclient软件。它会自动在/dev/net目录下创建一个名为tun的虚拟网卡，并使用IP地址192.168.53.5启动该虚拟网卡（tun0）。此外，由于需要与远程子网（192.168.60.0/24）进行通信，miniVPNclient还会自动在系统中添加一个路由表项，将所有目的IP地址在192.168.60.0/24范围内的数据包转发到虚拟网卡tun0。

因此，当用户发起对HostV（IP地址为192.168.60.101）的通信请求时，系统会将这些网络通信数据包写入tun0中。miniVPNclient会检测到tun0中有数据包写入，然后将这些数据包读取出来，并封装到一个TCP报文的数据部分中。接着，miniVPNclient通过与运行在VM中的miniVPNserver建立的TCP连接将该TCP报文发送出去。

运行在VM上的miniVPNserver会从TCP连接中接收到这个报文，并解封装出数据部分的数据。然后，它将这些数据写入到在VM中创建的虚拟网卡tun0中。系统会检测到tun0中有数据包写入，并通过网络层将其读取出来并发送出去。由于该数据包的目的IP地址是192.168.60.101，所以该数据包会被发送到docker2，然后docker2将其转发给HostV。

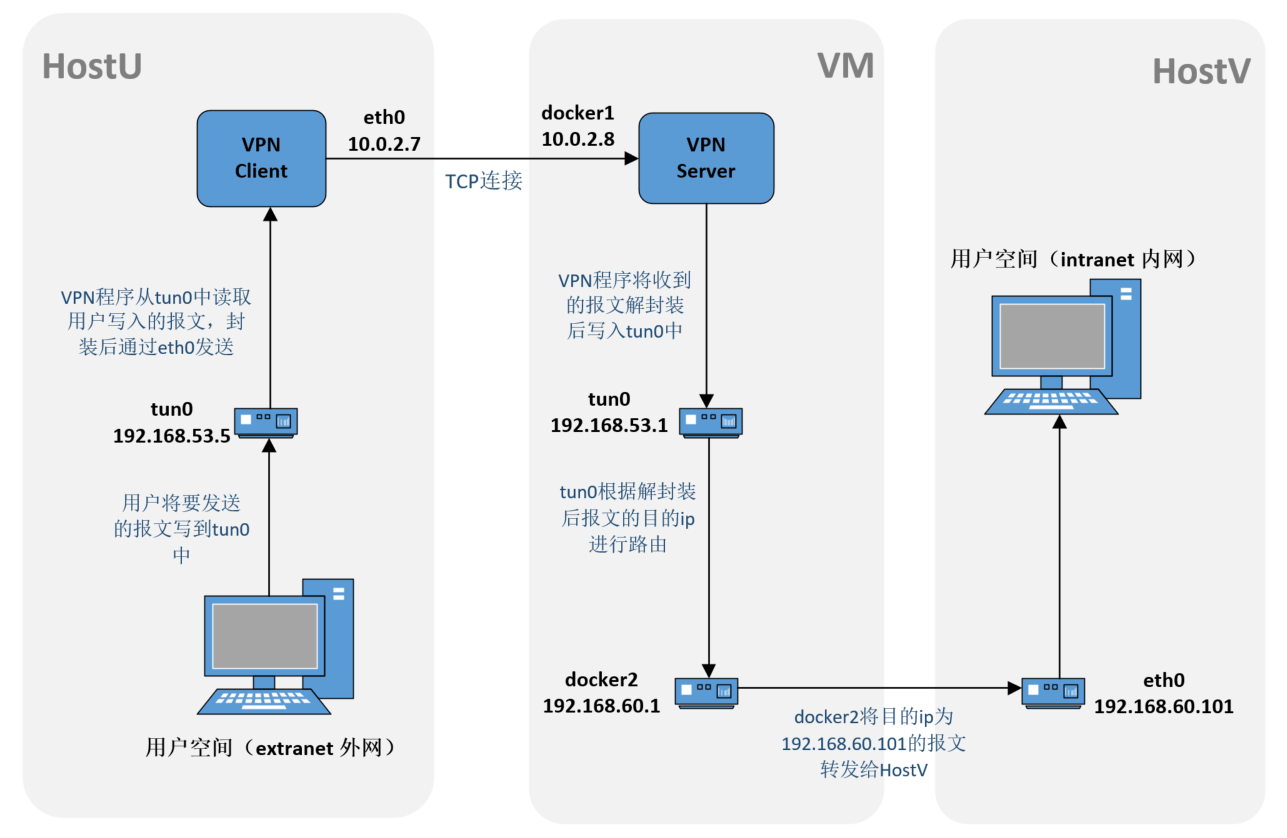


图1-2 extranet向intranet发送数据示意图

对于HostU和HostV来说，整个过程是透明的，他们并不察觉到他们发送的网络报文经历了封装、发送、解封装和接收的过程。因此，当HostV向HostU发送报文时，只需简单地将目的IP地址设置为192.168.53.5即可。在HostV中，我们可以将192.168.60.1设为所有报文的网关，这样报文就会被转发给docker2，而docker2会将报文发送给VM上的tun0。

当miniVPNserver程序检测到tun0接收到报文后，它会将该报文封装到一个TCP报文的数据段中，并发送给运行在HostU上的miniVPNclient。miniVPNclient接收到报文后，会从数据段中提取出原始报文，并将其写入到HostU上的tun0中。系统会读取该报文，并将其交付给之前发起与HostV通信的用户空间程序。

如图1-3所示，展示了报文的处理流程，整个过程中，HostU用户空间程序发出的网络报文好像经过了一个建立在公网上的隧道一样，直接到达了处于内网中的HostV。这也解释了为什么VPN有时被称为隧道。然而，在我看来，更形象地描述这个过程应该是"摆渡"。HostU用户空间发出的报文就像一个人开车过河，需要将车开到摆渡船上。摆渡船将车运送过河后，车再次下船并继续向目的地行驶。而miniVPN实现的正是这个"摆渡船"的功能。HostU和需要访问的内网中的HostV就像是河对岸的关系。通过miniVPN的"摆渡"功能以及Linux提供的tun虚拟网卡功能，我们成功实现了所需的通信机制。

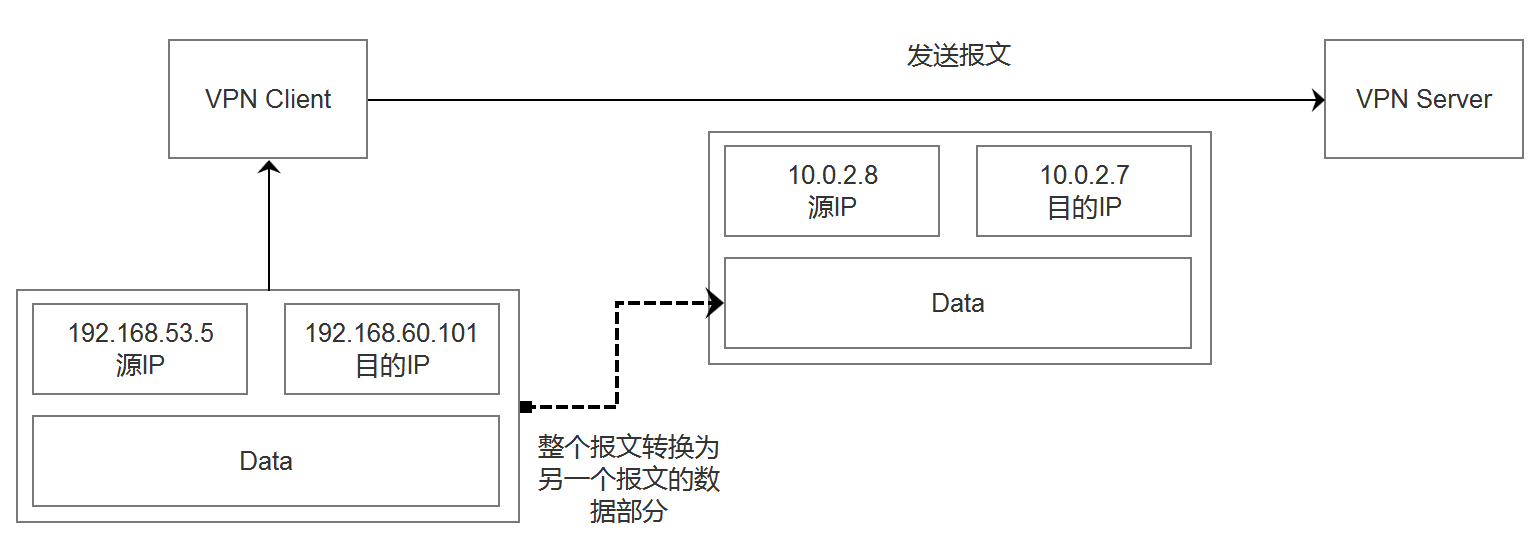


图1-3 报文的处理流程

加密原理

在之前介绍的miniVPN的"摆渡"过程中，由于仅简单地将原始报文作为数据段进行传输，无法确保通信过程的数据机密性和完整性。因此，我们需要使用能够提供这种保证的TLS协议进行传输。具体而言，miniVPNclient和miniVPNserver在建立TCP连接后，完成TLS v1.2所定义的服务器身份验证和密钥协商过程。随后，在后续的通信过程中，使用协商得到的密钥对经过"摆渡"的报文进行整体加密，到达目标主机后再由miniVPN进行解密。

TLS v1.2的握手流程如下：

1. miniVPNserver向miniVPNclient发送经过CA签名的证书，其中包含CA私钥对miniVPNserver公钥摘要值的签名以及公钥本身。miniVPNclient收到证书后，使用CA的公钥对公钥摘要值进行解密，并计算证书中附带的公钥的摘要值。将这两个摘要值进行比较，当它们相等时，表示服务器身份验证成功。
2. miniVPNclient基于AES256-GCM-SHA384算法生成一个加密密钥，并使用前一步中获得的miniVPNserver的公钥对该密钥进行加密。然后将加密结果发送给miniVPNserver。
3. miniVPNserver收到加密值后，使用自己的私钥进行解密，得到对称密钥。至此，密钥协商完成。
4. 在随后的隧道通信中，使用前述对称密钥对用户空间的报文进行整体加密，并将其封装到TCP报文的数据段中进行传输。

在这个过程中，首先需要为miniVPNserver生成一对公私钥，并为公钥生成证书。由于我们无法向真正的CA申请证书，因此我们需要自行生成CA的公私钥，并使用CA的私钥对miniVPNserver公钥的签名请求进行签名。此外，还需要将CA的公钥添加到HostU中的信任CA列表中。否则，由于对miniVPNserver公钥进行签名的CA并非信任的CA，身份验证仍无法完成。CA和miniVPNserver的公私钥均基于2048位的RSA算法。

认证机制

认证机制涵盖了客户端对服务器的身份验证和服务器对客户端的身份验证两个方面。

客户端对服务器的身份验证是通过使用CA进行验证的。在TLS握手过程中，服务器会发送自己的证书和公钥给客户端。客户端使用CA的公钥来验证服务器的证书是否经过CA的真实签名。如果验证成功且证书处于有效期内，那么客户端可以确认对服务器的身份验证是成功的。

服务器对客户端的身份验证发生在TLS握手完成后，也就是在客户端认证服务器身份之后。此时，基于TLS的隧道已经建立完成，因此可以认为客户端和服务器之间的数据通信是安全的。在这种情况下，服务器可以要求客户端使用账户和密码进行登录。账户和密码的验证是基于服务器上的"/etc/shadow"文件来确定的，客户端需要使用服务器上已存在的账户及其相应的密码进行登录。当服务器验证发现客户端发送的账户和密码与"/etc/shadow"文件中的记录匹配时，即可认为服务器对客户端的身份验证是成功的。

# VPN系统设计

## 系统概要

miniVPNserver包含SSL通信模块、管道通信模块、TUN通信模块以及主体模块,四个模块协作完成miniVPNserver的工作，如图2-1所示。它们主要负责的工作如下。

SSL通信模块：与miniVPNclient建立TLS连接、完成服务器身份的证明工作、完成密钥协商并使加解密过程对其他模块透明。

管道通信模块：负责创建命名管道并监听该管道，当从管道中读取到数据之后，将其交付给SSL通信模块。同时，还会根据管道中数据的情况，判断连接是否正常，当连接错误或者结束时，停止对管道的监听，并将管道文件删除。

TUN通信模块：负责监听虚拟网络设备tun0，从tun0中读取到报文数据后，根据报文的目的ip来将其写入对应的管道文件中。

主体模块：负责实现各个模块的初始化、启动过程。在接收到miniVPNclient的连接请求时，通过调用SSL通信模块来完成ip分配的工作。同时，主体模块还为实现ip占用信息共享、管道文件描述符等数据的进程间共享提供了共享内存区域初始化以及各自的同步锁（用于对共享内存区域访问进行锁操作）初始化的功能。

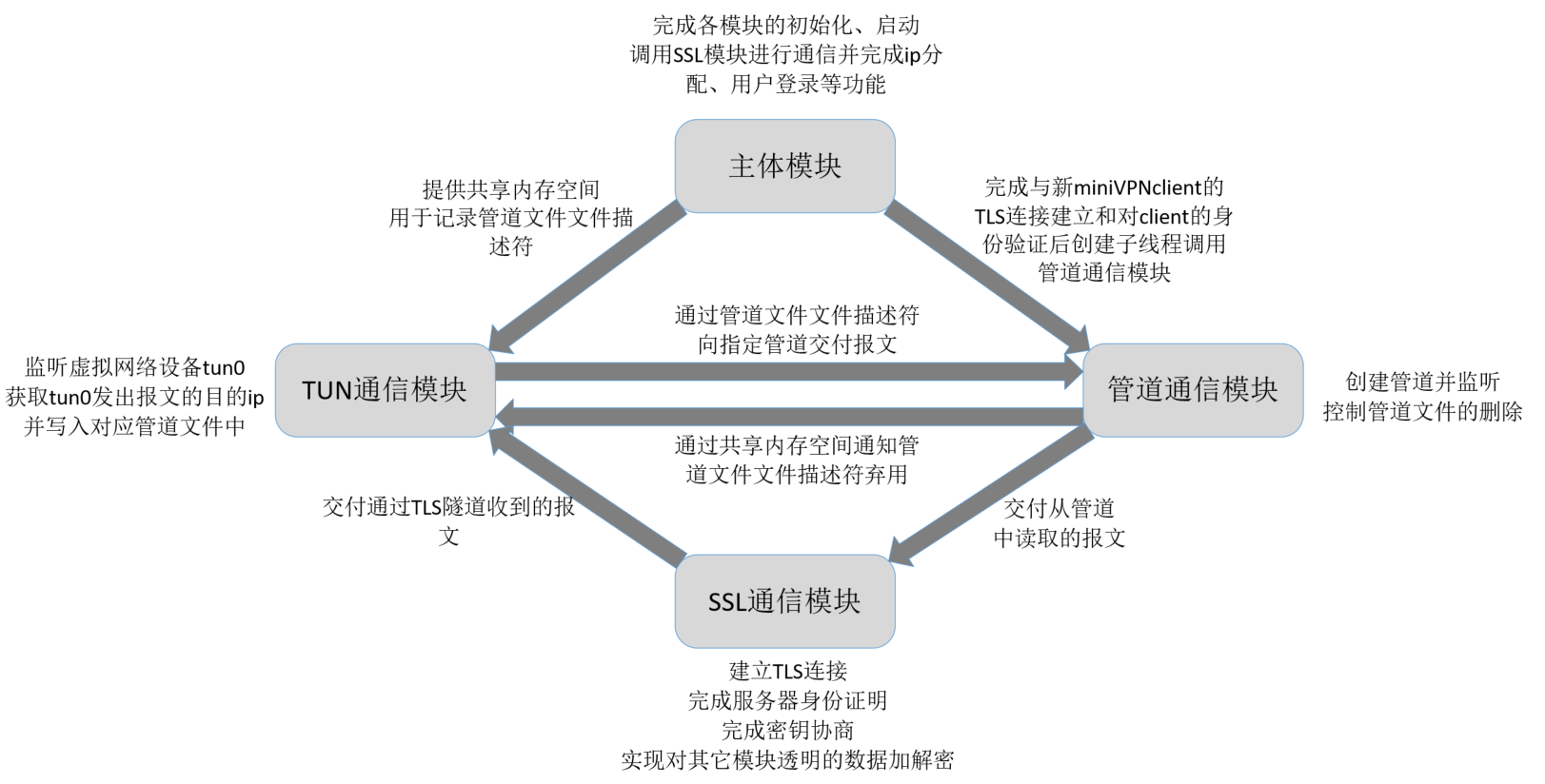


图2-1 miniVPNserver整体模块架构

miniVPNclient仅有一个主体模块，实现与miniVPNserver建立TLS连接，验证服务器身份，获取虚IP的功能，接收来自TLS隧道的报文，将数据部分解密后交付给虚拟网卡tun0，同时还要负责监听虚拟网卡tun0，从tun0中读取到报文后将报文加密封装并通过TLS隧道发送给miniVPNserver。

## miniVPNserver模块设计

SSL通信模块详细设计

SSL通信模块中主要的数据结构如下：

1. SSL结构体指针SSL\* ssl。
2. 通过共享内存访问的ip占用符数组ipFlag\* ipflags。
3. 通过共享内存访问的管道文件描述符表pipefdTable\* pipefdTables。
4. 两片共享内存区域各自的锁变量mutexWorker ipFlagMutexWorker和mutexWorker pipefdTableMutexWorker。

其中SSL结构体由ssl库提供，其他自定义结构体内容如下图2-2所示。



图2-2 SSL信模块自定义结构体

SSL通信模块并无什么特别设计的算法，毕竟数据的加解密过程都由openssl提供的函数为我们完成，下面图2-3为SSL通信模块的主要工作流程图。

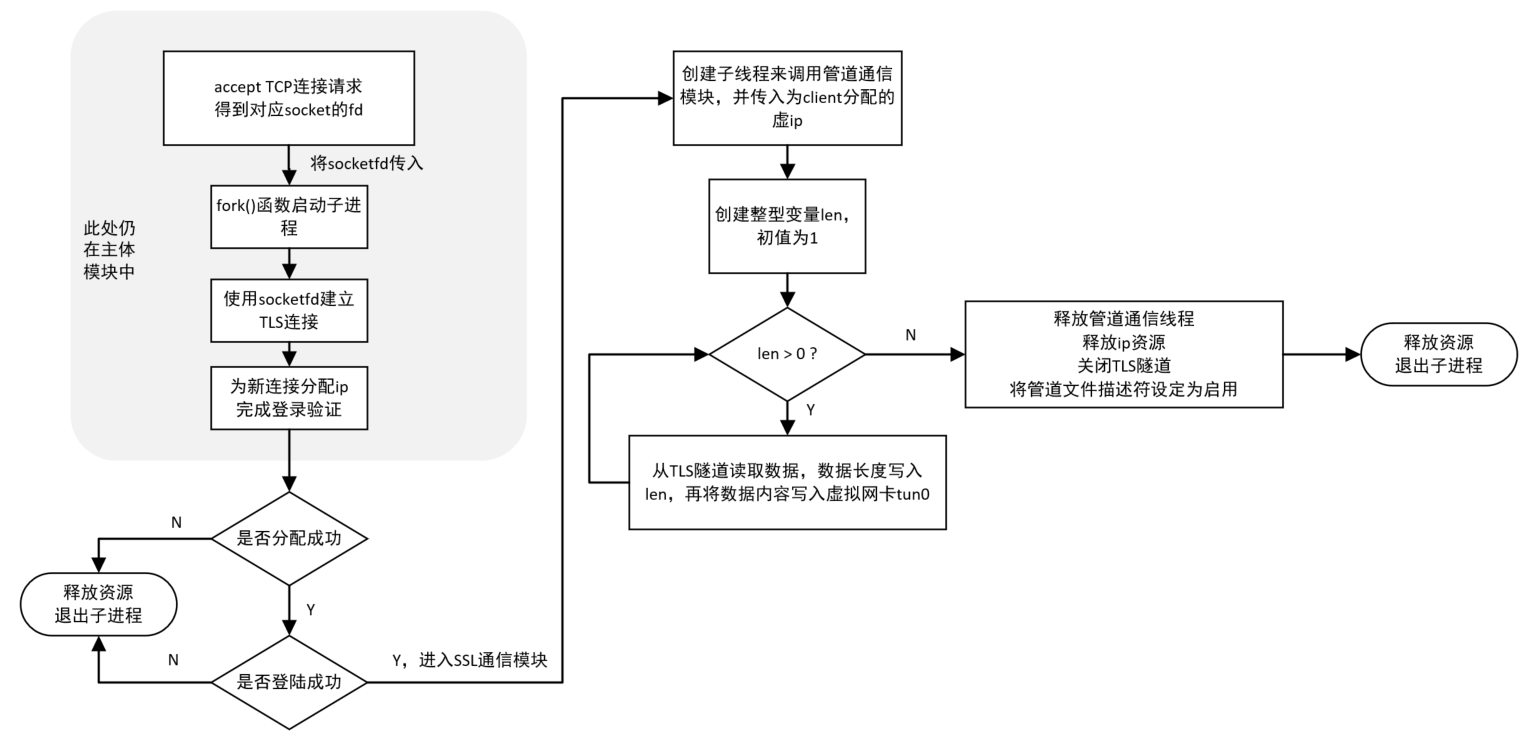


图2-3 SSL通信模块主要工作流程图

在图中，我们可以看到一些关键的模块和代码片段，其中主体模块扮演着核心角色。在建立TLS连接后，主体模块负责进行IP分配和用户验证登录的过程。一旦登录完成，SSL通信模块启动子线程调用管道通信模块的功能。随后，SSL通信模块会不断地从TLS隧道中读取数据。

在读取数据时，TLS隧道中的数据是经过加密的，但是SSL通信模块会执行解密操作，将解密后的明文数据写入缓冲区（buf），然后将数据从缓冲区写入tun0虚拟网卡中。值得注意的是，当TLS隧道中没有数据可读时，读取函数会进入阻塞状态，而不是直接返回长度为0的数据。这与后面提到的客户端退出导致的读取长度为0的情况是不同的。

在该模块中，有两个关键的代码段。首先，在图2-4中展示了从TLS隧道读取数据并将其写入tun0虚拟网卡的实现代码。其次，在图2-5中展示了将数据写入共享内存的代码，以便将IP解除占用和管道文件描述符启用的信息传递给其他进程。这些关键代码段确保了数据的传递和共享。

需要特别说明的是，在图2-5中，ipflags是指向共享内存空间的指针，用于保存IP占用情况的信息。而pipefdTables是指向共享内存空间的指针，用于保存管道文件和其文件描述符之间的映射关系。此外，ipFlagMutexWorker和pipefdMutexWorker是包含相应锁变量的结构体。

在代码中，我们可以观察到以下操作顺序：首先，使用pthread\_mutex\_lock函数对锁进行上锁，然后再访问相应的共享内存空间。例如，(ipflags->area)[malloc\_ip] = 0表示将当前分配的IP的占用情况设置为0，即表示未被占用。另外，pipefdTable\_delete(pipefdTables, worker.pipe, strlen(worker.pipe))表示从pipefdTables中删除与worker.pipe中保存的管道文件名对应的文件描述符，即执行弃用操作。通过这些操作，程序能够更新IP的占用情况并删除不再使用的管道文件对应的文件描述符，以确保相关数据的一致性和正确性。



图2-4 从TLS隧道读取数据并交付给tun0的代码展示图



图2-5 进程间信息传递代码展示图

TUN通信模块详细设计

TUN通信模块的主要数据结构如下：

1. 管道文件描述符表 pipefdTable\* pipefdTables，通过共享内存进行访问。
2. 管道文件描述符表对应的锁变量 mutexWorker pipefdTableMutexWorker，与SSL通信模块中的定义相同。

上述数据结构与SSL通信模块中的定义相同，并通过共享内存实现对管道文件描述符的访问。

在TUN通信模块中，主要算法的任务是根据从虚拟网卡设备tun0中读取到的IP数据报的目的IP，在管道文件描述符表（即pipefdTables）中查询相应IP对应的管道文件的文件描述符。如果查询得到的文件描述符值为0，则表示在文件描述符表中不存在有效的对应管道文件的文件描述符。

读取目的IP的算法相对简单。首先，从tun0中将数据读取到缓冲区buff中，然后检查首字节是否为0x45，其中4表示报文协议为IPv4，5表示IP报文头部长度为5 \* 4，即20字节。这样我们可以基本确定接收到的数据是一个完整的IPv4报文。接下来，直接读取buff的第17到20个字节，因为这部分是IP报文头的目的IP地址字段，如图2-6中所示。

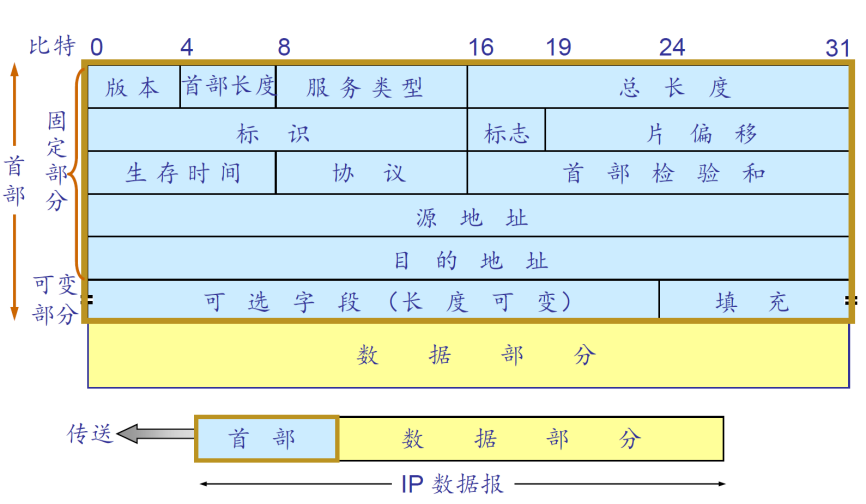


图2-6 IP数据报格式展示图

TUN通信模块的主体函数的执行流程如图2-7所示。在图中，程序首先声明一个大小为4的整型数组ipv4，命名为ipv4，并从指定位置的buff中读取目的IP的数值。在读取时，必须将数据强制转换为unsigned char类型后再读取，这是因为IP地址的数值是无符号的，必须使用无符号数据类型进行读取。由于数据是按字节读取的，所以使用unsigned char来进行强制转换。读取目的IP值完成后，程序使用sprintf函数构造包含管道文件完整路径的字符串，并将该字符串存储在变量ipdst中，同时还将目的IP地址的字符串形式存储在变量ipdstStr中。

TUN通信模块的重点代码在图2-7中都有详细给出。其中，特别重要的部分包括从IP数据报中读取目的IP、在文件描述符表中查找记录项、对文件描述符表进行更新以及在访问文件描述符表之前使用跨进程的锁变量进行上锁。这四部分内容对于TUN通信模块的正常运行至关重要。

在TUN通信模块的代码实现中，图2-7中已经提供了以下重要部分：

1. 从IP数据报中读取目的IP地址。

2. 在文件描述符表中查找相应记录项。

3. 对文件描述符表进行更新。

4. 在访问文件描述符表之前使用跨进程的锁变量进行上锁。

这四部分内容是TUN通信模块中的关键代码，它们负责实现从接收到的IP数据报中提取目的IP地址、查询和更新文件描述符表的相关操作，并使用跨进程的锁变量确保对文件描述符表的访问的互斥性。

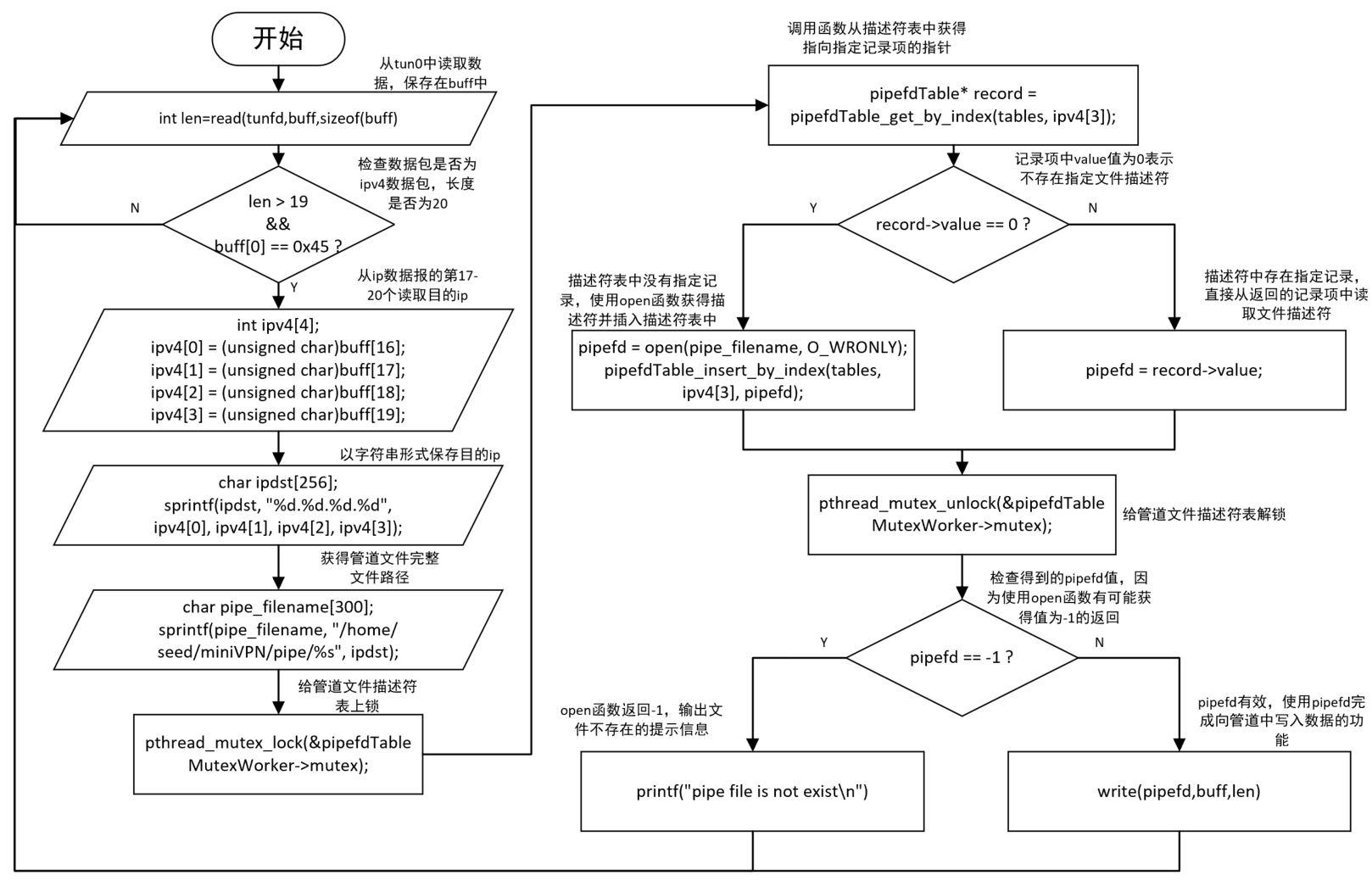


图2-7 TUN通信模块主体函数执行流程图

管道通信模块详细设计

管道通信模块主要的数据结构如下：

1. 线程变量pthread\_t pipeThread。
2. 用于承载数据信息进入子线程的结构体pipeWorker worker。
3. 指向子线程任务函数的函数指针void\* (\*pipeListen)(void\* \_worker)。

pthread\_t结构体由pthread库提供，其他自定义结构体如下图2-8所示。



图2-8 管道通信模块自定义结构体

管道通信模块负责持续监听指定的管道文件，并使用pipeWorker结构体中指定的SSL结构体将数据发送出去。主要的函数执行流程如图2-9所示。

管道通信模块由主体模块启动，主体模块首先创建一个pipeWorker结构体变量，并将其中的SSL结构体指针和管道文件的完整绝对路径写入。然后，主体模块调用管道通信模块的进入函数createSubThread，该函数接收一个pthread\_t线程变量指针和一个pipeWorker结构体变量指针作为参数。

一旦进入createSubThread函数，就进入了管道通信模块。首先，它尝试使用Linux内核提供的mkfifo函数创建管道文件，传入参数为指向管道文件完整绝对路径字符串的指针和创建后的权限属性0666。如果管道文件创建失败（mkfifo函数返回值为-1），createSubThread函数会直接退出，并结束管道通信模块的运行。如果创建成功，它将使用pthread库提供的pthread\_create函数启动子线程，并传入pipeListen函数的指针作为子线程要执行的函数。

createSubThread函数执行完毕后，流程图中的执行流程指向pipeListen函数。pipeListen函数的执行流程比较简单。它首先通过管道文件的绝对路径获得文件描述符，然后创建用于暂存从管道中读取数据的缓冲区buff。接下来，它使用len变量作为循环变量，进入读取-发送的循环。

在循环体中，len的值由读取情况决定。当正常读取数据时，len的值为读取的数据长度。当管道关闭时，read函数返回值为0。当读取失败或出错时，read函数的返回值为-1。如果len的值小于等于0，则循环结束，模块认为通信结束，并使用remove函数将管道文件删除。

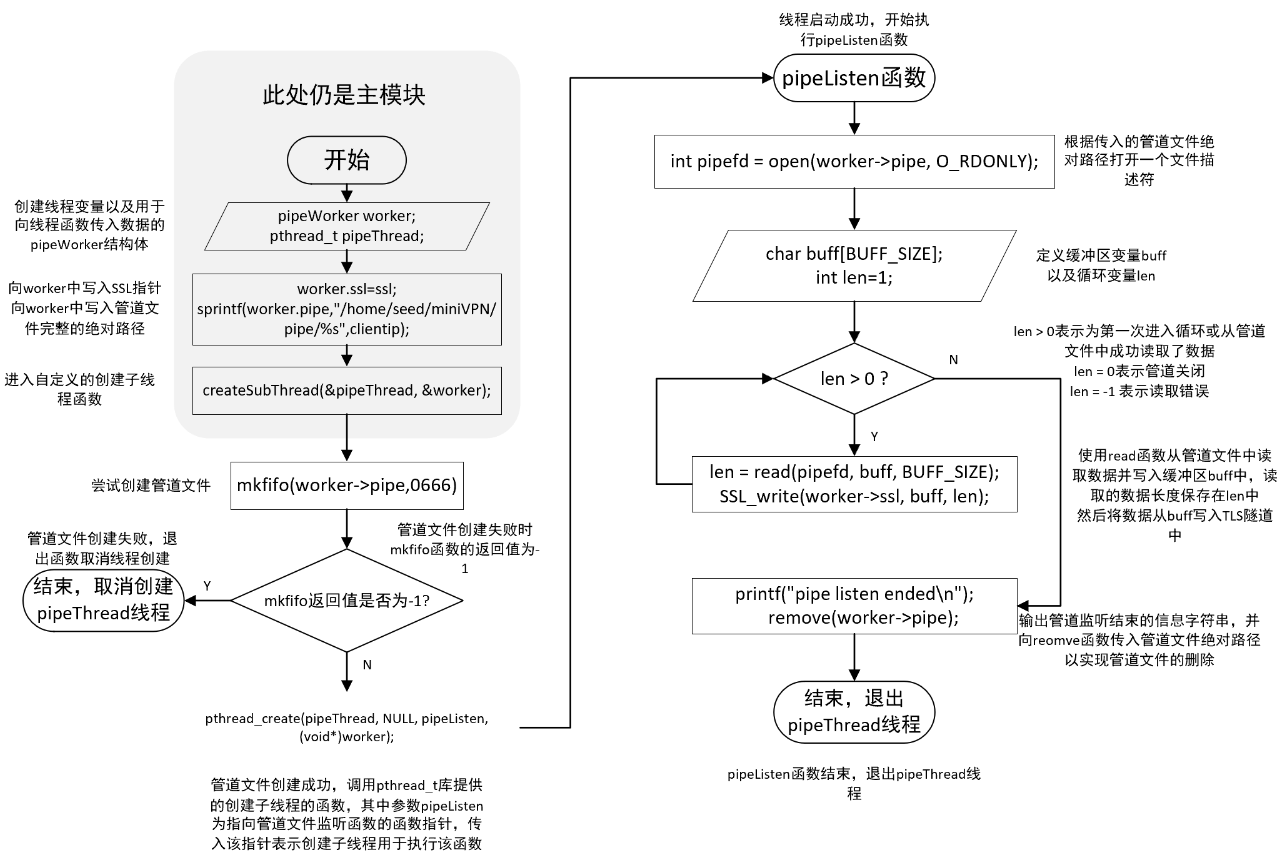


图2-9 管道通信模块主要函数执行流程图

管道通信模块的关键代码就是createSubThread函数中，尝试创建管道文件并根据创建结果决定是否启动子线程的代码，具体代码如图2-10所示。



图2-10 createSubThread函数代码展示图

主体模块详细设计

主体模块中主要的数据结构如下：

* + - 1. 两个共享内存区域的shmget函数返回的id，分别是int ipFlagShareMemoryId和int pipefdTableShareMemoryId。
      2. 指向共享内存区域的IP占用符数组指针 ipFlag\* ipflags。
      3. 指向共享内存区域的管道文件描述符表数组指针 pipefdTable\* pipefdTables。
      4. 两片共享内存区域各自的锁变量 mutexWorker ipFlagMutexWorker 和 mutexWorker pipefdTableMutexWorker。
      5. SSL结构体指针 SSL\* ssl。
      6. 一组用于启动TUN通信模块的变量，包括：用于启动TUN通信模块工作线程的线程变量 pthread\_t tunListenThread；用于传递数据给子线程的结构体 tunListenWorker tunWorker，其中包含tunfd（tun0虚拟网卡的文件描述符）和 pipefdTableShareMemoryId（管道文件描述符表共享内存区域的id）；指向子线程任务函数的函数指针 void\* (\*tunListen)(void\* \_worker)；tun0虚拟网卡的文件描述符 int tunfd。
      7. TCP Socket监听使用的socket号 int tcpListenSock。
      8. TCP建立连接后返回的socket号 int tcpClientConnectionSock。

在主体模块中，主要的算法有两个，工作流程如图2-11所示。一个是实现为客户端分配IP的算法，另一个是实现客户端基于服务器上/etc/shadow文件登录的算法。

分配IP的算法中只需在ipflags数组中顺序遍历，直到找到一个满足条件的IP（即ipflags[i]等于0），表示IP地址192.168.53.i可用。登录算法中，miniVPN客户端在获得分配的IP信息后，会要求用户输入用于登录的账户密码。主体模块发送完可用的IP信息后，开始调用SSL\_read函数读取miniVPN客户端发送的账户密码，并使用Linux提供的shadow库进行验证。验证结果有三种：登录成功、密码错误、用户用于登录的账户不存在。主体模块使用1、-1、-2三个值来代表这三种状态，并在相应时刻将其返回给客户端，以使客户端了解登录结果。

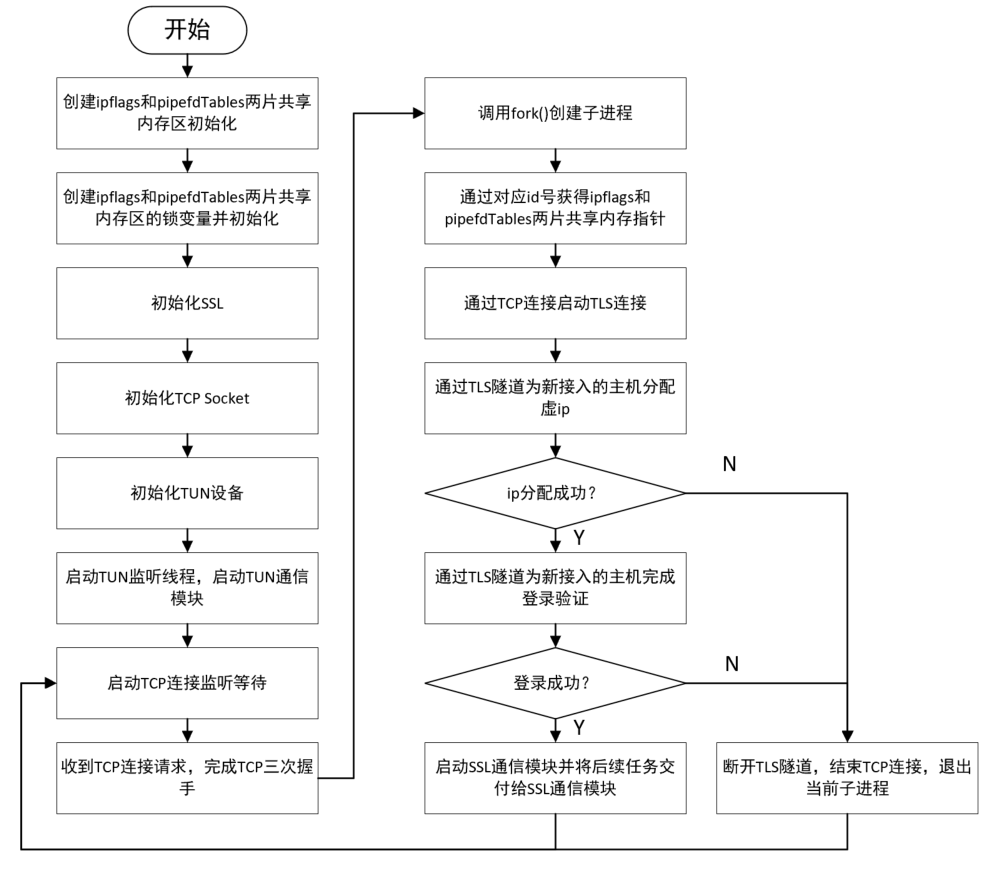


图2-11 主体模块工作流程图

主体模块负责执行多个任务，并且流程图中通过文字描述了这些任务的工作流程。以下是主体模块中几个重要部分代码的概述。

首先，通过使用shmget和shmat函数，主体模块声明和访问了共享内存区域。这些函数允许主体模块与其他模块之间共享数据。此外，为了确保数据的同步和访问安全，主体模块还创建了锁变量。这些代码的详细实现可以参考图2-12。

其次，主体模块包含了一个函数，用于为新接入的客户端分配IP地址。这个函数的作用是按顺序遍历IP占用符数组，并找到一个可用的IP地址分配给客户端。通过这个函数，主体模块能够动态地管理IP地址的分配。具体的代码实现可以参考图2-13。

最后，主体模块还包含了一个用于新接入客户端登录验证的函数。在这个函数中，主体模块接收客户端发送的账户密码信息，并利用Linux提供的shadow库进行验证。验证结果将分为三种情况：登录成功、密码错误和用户账户不存在。主体模块将使用特定的返回值来表示这些状态，并及时将结果返回给客户端，使其能够了解登录的结果。图2-14中给出了相应的代码实现。

通过上述的代码实现，主体模块能够有效地处理共享内存的声明和访问、为新客户端分配IP地址以及进行登录验证等任务。这些功能的实现为整个系统的正常运行提供了基础支持。



图2-12 共享内存声明及创建锁变量代码展示图



图2-13 为新接入客户端分配IP函数代码展示图



图2-14 登陆验证服务函数代码展示图

## miniVPNclient详细设计

主体模块中主要的数据结构图下：

1. 指向服务器域名字符串的指针char\* hostname.
2. 保存miniVPNserver在服务端监听端口号的整型值int port.
3. SSL结构体指针SSL\* ssl.

主体模块的主要任务是与miniVPNserver建立TCP连接，进行身份验证和密钥协商，并建立TLS隧道，获取服务端分配的虚拟IP，并实现登录验证功能。完成这些任务后，主体模块开始进行隧道通信。整个工作流程如图2-15所示。

模块中，重要的代码分别有两处，一处是实现登录验证部分的代码，其内容如图2-16所示，一处是实现用户登录时输入密码不回显的代码，这一处代码在后面有展示。

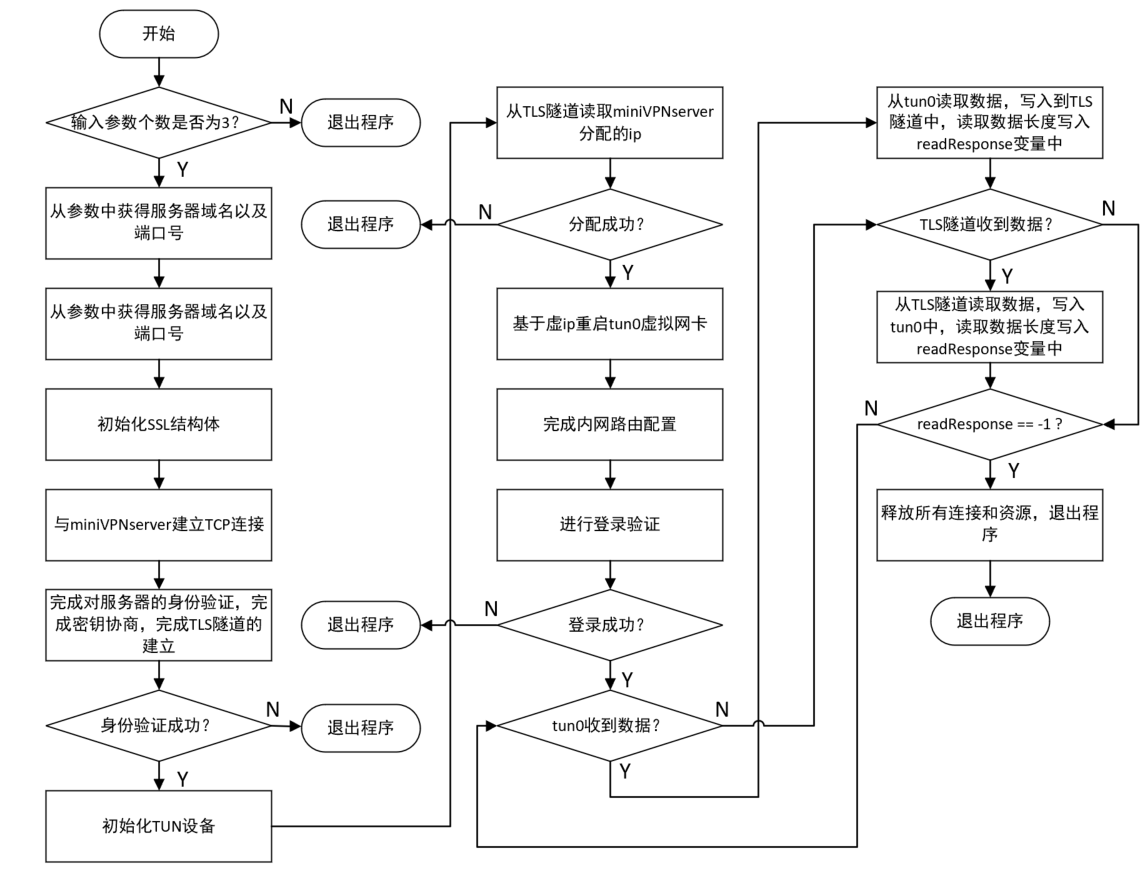


图2-15 miniVPNclient主体模块工作流程图

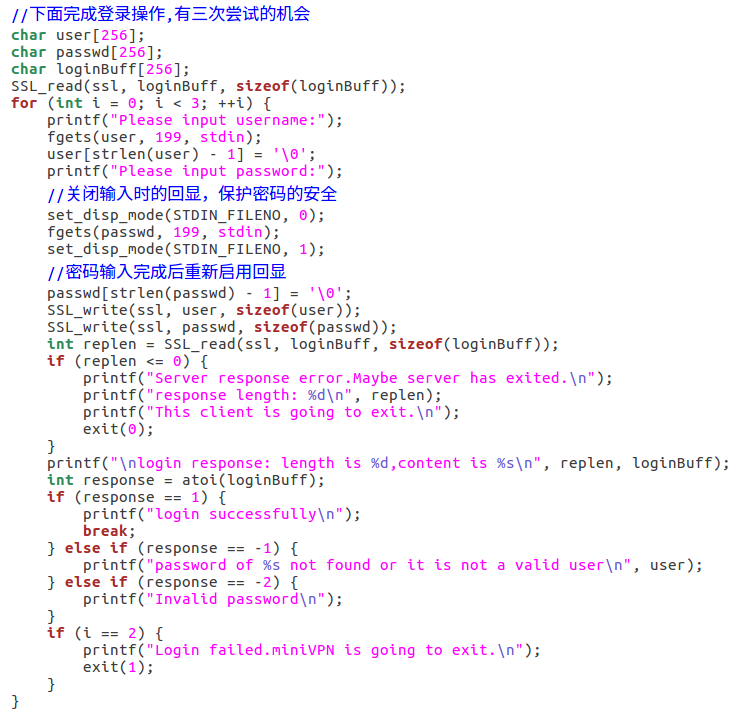


图2-16 miniVPNclient实现登录验证代码片段展示图

# VPN实现细节

1. 回答实验指导手册4.2第6步提出的问题：在HostU上，telnet到HostV。在保持telnet连接存活的同时，断开VPN隧道。然后我们在telnet窗口中输入内容，并报告观察到的内容。然后我们重新连接VPN隧道。需要注意的是，重新连接VPN隧道，需要将vpnserver和vpnclient都退出后再重复操作，请思考原因是什么。正确重连后，telnet连接会发生什么？会被断开还是继续？请描述并解释你的观察结果。

观察到，在telnet连接窗口中执行命令时，并没有任何反应或变化。然而，当我们重新启动服务器程序和客户端程序，并重新连接VPN隧道后，我们可以观察到之前输入的命令会在原telnet连接窗口中执行一次，而telnet连接保持不变。

可能的原因是，当隧道断开时，TCP连接并未中断。只有重新启动客户端程序或服务器程序是无法重新连接之前的连接的，因此我们需要双方都退出并重新连接，才能建立新的隧道。在隧道断开期间，TCP会将无法执行的命令缓存，并不断尝试执行。一旦隧道重新连接，并且指令信息能够发送出去，那么指令就可以成功执行。

1. 你的VPN客户端，是如何知道并配置自己的虚IP和服务器保护子网路由的？

虚拟IP：每个miniVPNclient在启动时，使用保留的未使用IP地址192.168.53.2来配置tun0网卡。TLS隧道建立后，miniVPNserver和miniVPNclient之间的通信实际上是在它们所在机器的真实物理网卡上进行的，这些网卡都有唯一确定的IP地址，不依赖于tun0虚拟网卡的IP。因此，即使同时有两个miniVPNclient使用192.168.53.2连接到miniVPNserver，报文的源机器也能够正确识别它们。这是因为tun0虚拟网卡的IP地址在客户端与内网机器进行通信时才会生效，而IP地址的分配是在miniVPNclient和miniVPNserver之间进行的。因此，所有的miniVPNclient都可以使用192.168.53.2作为初始启动tun0虚拟网卡的IP地址，但是miniVPNserver在分配IP地址时必须将这个IP保留不使用，否则当有一个miniVPNclient实际被分配到这个IP时，会导致报文转发错误的情况发生。

之后，miniVPNclient使用192.168.53.2启动tun0虚拟网卡后，会向miniVPNserver发起TLS连接。连接成功后，在隧道中写入字符串"192.168.53.2"。miniVPNserver读取到这个IP后，会知道需要为其分配一个IP地址。然后，miniVPNserver的SSL通信模块会查询可用的IP地址，并将其标记为已占用。然后，它会以完整的IP字符串形式将这个IP写入TLS隧道，例如"192.168.53.3"。miniVPNclient收到这个IP后，会使用它重新启动tun0虚拟网卡。

子网路由：miniVPNclient会自动为设备配置到内部子网的路由，这是通过在程序中添加代码system(“route add -net 192.168.60.0/24 tun0”)实现的，这一行代码将在完成ip分配之后才执行，否则ip分配过程中的tun0虚拟网卡重启的操作将导致导致路由失效。

1. 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？

miniVPNclient的用户名口令认证是在TLS隧道已经建立完成，ip分配成功之后，程序启动对tun0虚拟网卡的监听之前实现的。也就是当客户端用户名口令认证成功后，才启动对tun0虚拟网卡监听，在这之前，虽然从客户端设备到内部子网的路由已经添加，但是由于写入到tun0虚拟网卡中的报文不会被读取，导致从客户端设备到内部子网的网络通信实际上仍然是阻塞的。认证将有3次机会，失败时，miniVPNclient会关闭TLS隧道，并且自动结束程序，而miniVPNserver由于此时已经完成了ip分配，因此还需要将ipflags中这次连接使用的ip改为空闲，然后再结束为这一次隧道通信创建的子进程。

1. 你的VPN服务器支持多客户端采用的什么技术？VPN服务器收到保护子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？

支持多客户端采用的是多进程架构，miniVPNserver会使用fork()函数为每一个连接启动一个单独的进程。进程间通信采用的是共享内存技术，同时为了保证同一时间只有一个进程访问这片内存空间，程序还使用了可在进程之间使用的锁机制。

miniVPNserver收到来自内网主机的报文之后，会读取报文的第17到第20个字节，这一部分正是ipv4报文的目标ip地址部分，得到目的ip之后即可将这个报文写入与ip同名的管道文件中。即miniVPNserver创建管道文件时，根据miniVPNclient连接后分配的虚ip来创建管道文件，如当miniVPNclient获得的虚ip为192.168.53.5时，就会创建管道文件/home/seed/miniVPN/pipe/192.168.53.5。故可根据内网主机所发送报文的目的ip来确定应当将报文写入哪一个管道文件中。

1. 你的VPN客户端退出时，VPN服务器如何发现？如何处理？

miniVPNclient退出时，会在TLS隧道中写入一个长度为0的数据，这在正常通信过程中是不会出现的，因此当miniVPNserver从TLS隧道中读到一个长度为0的数据时，会判断对端miniVPNclient已经退出，于是结束通信，将为该miniVPNclient分配的虚IP改为空闲后退出子进程。

# 测试结果与分析

连通性测试

1. 单客户端

首先在VM上启动miniVPNserver，然后在HostU上启动miniVPNclient，再另起一个HostU的命令行，然后尝试ping HostV（192.168.60.101），结果如图4-1所示。在图中可以看到客户端在验证服务端证书时输出了证书中的详细信息。

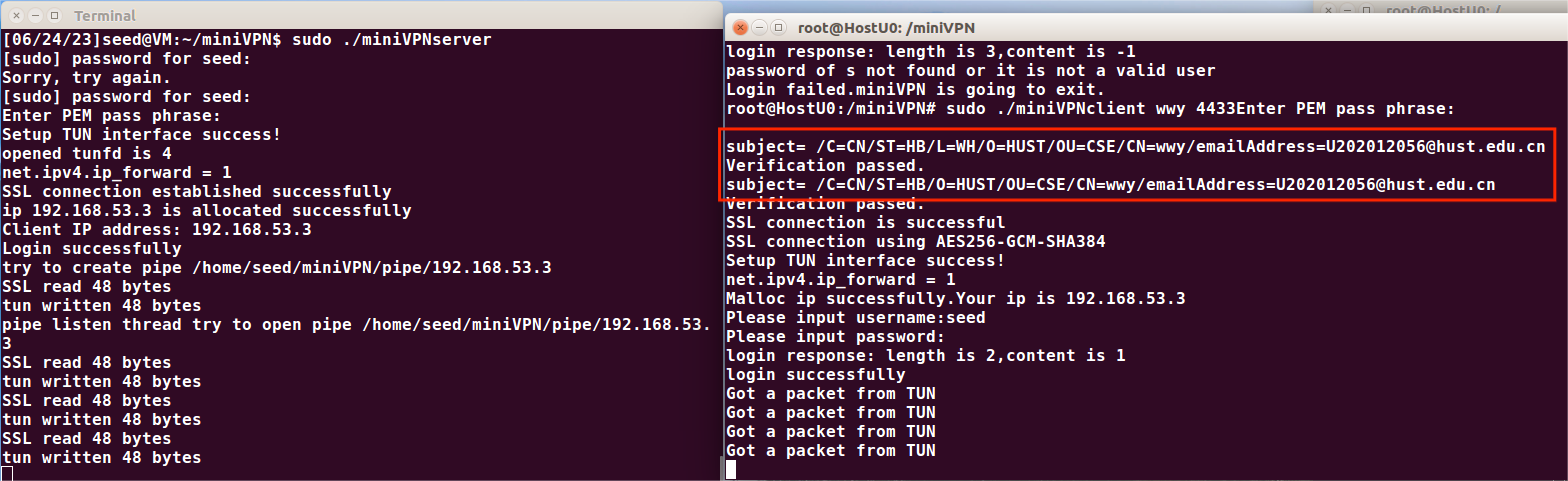


图4-1 基础连通性测试结果图

1. 多客户端

新增一个客户端运行在HostU1上，两个客户端连通之后同时进行ping操作。测试结果如图4-2所示。可以在图中看到HostU1和HostU0都能够正常的ping通，HostX的ip也自动分配为192.168.53.4，服务器端交替显示来自两个客户端的活动信息。

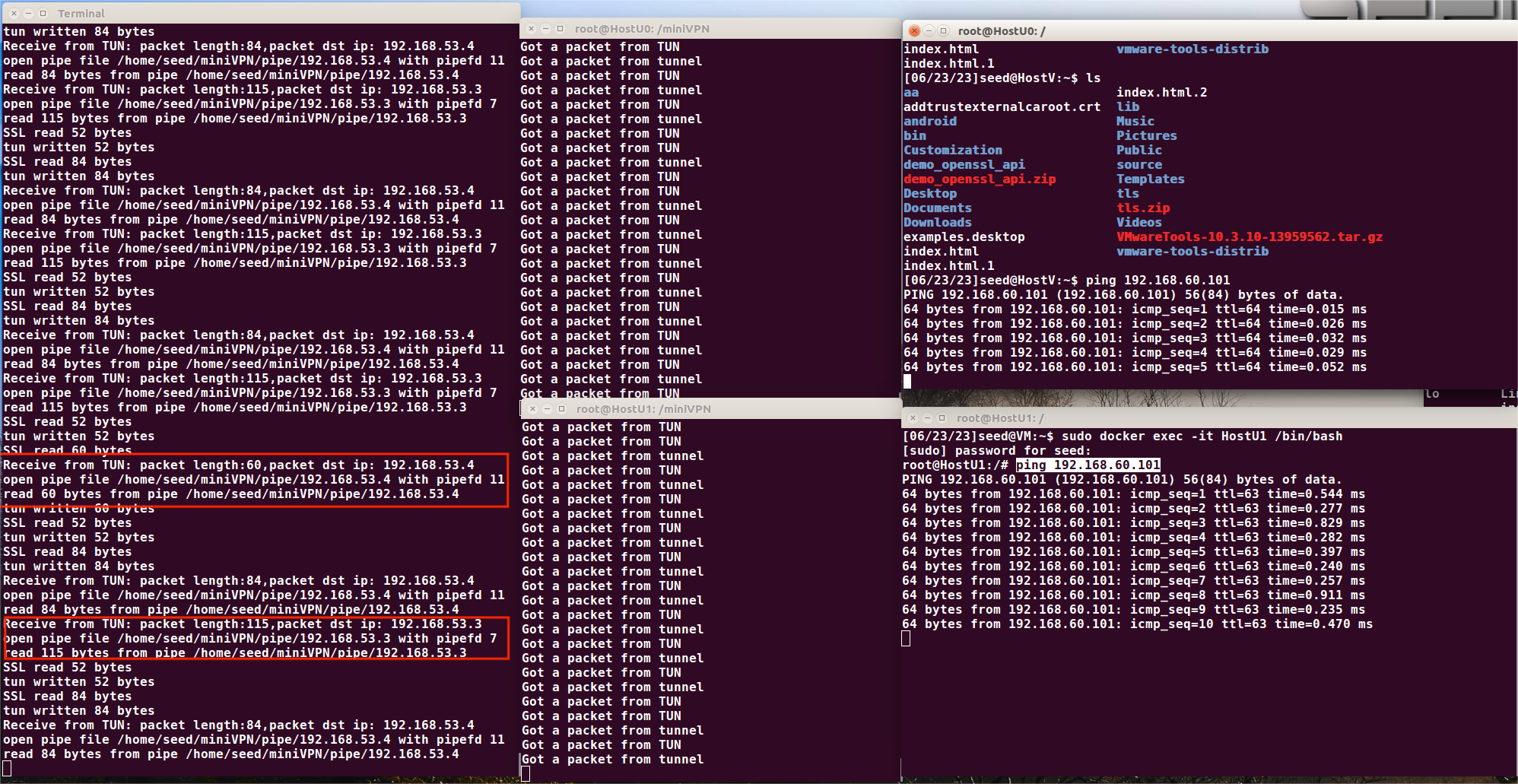
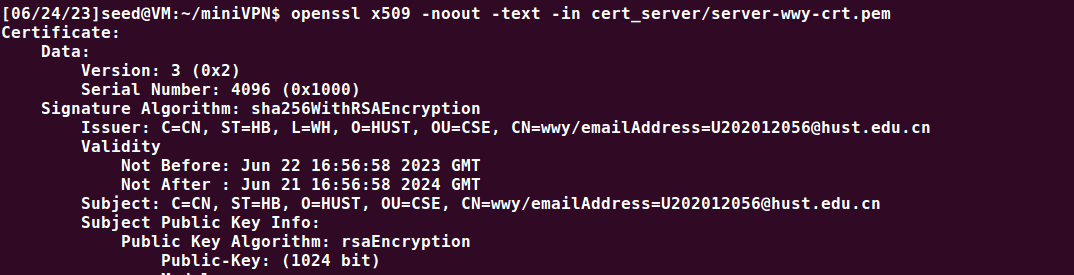


图4-2 双客户端连通测试

## 安全性测试

1. 用opensssl检查VPN服务器证书信息

在seed@VM中运行指令：openssl x509 -noout -text -in cert\_server/server-wwy-crt.pem。如下图4-3所示，证书主题包含个人信息，符合要求。

 图4-3 用opensssl检查VPN服务器证书信息

1. 使用错误的账户密码进行登录

在图4-4中可以看到，登录的第一次尝试中使用了错误的密码，程序返回了密码无效的结果，拒绝了我们的登录。第二次尝试中使用错误账户s登录，程序返回了账户无效的结果，再次拒绝了我们的登录。尝试三次过后，服务器依次显示用户登陆状态，客户端则自动退出。

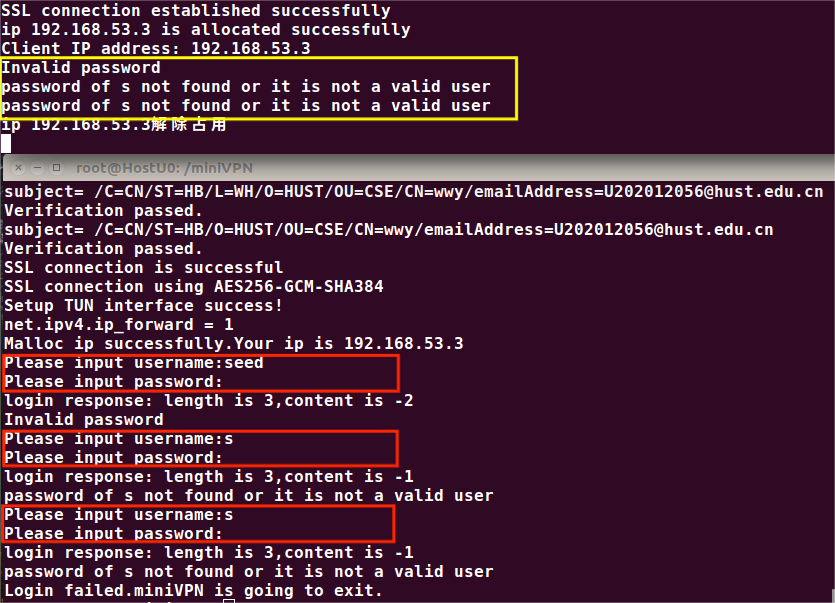


图4-4 使用错误账户和错误密码登录尝试示意图

1. 证书过期

由于证书的有效期为一年，这里将容器的时间改为2024年使得服务器的证书过期，测试结果如图4-5所示。可以在图中清楚的看见，客户端在验证证书时发现证书过期，于是直接退出了程序，结束了和服务端的连接。

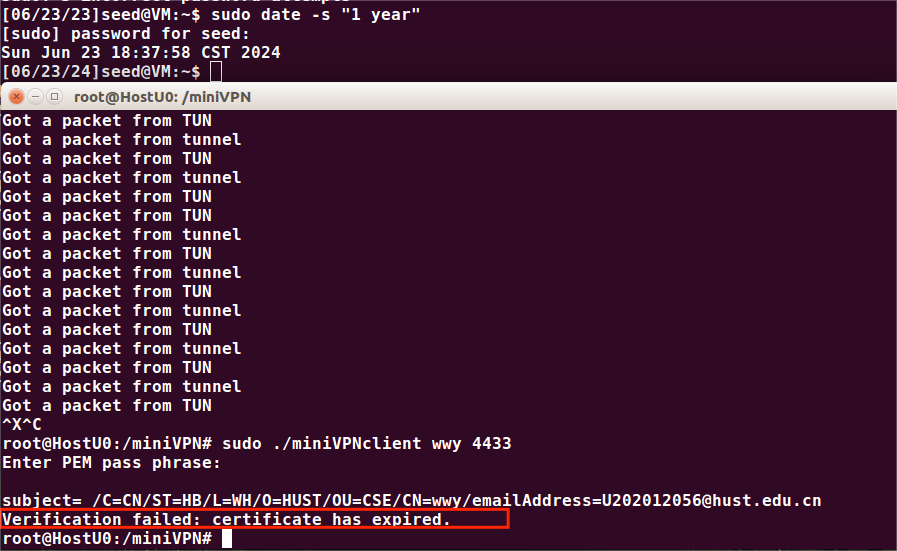


图4-5 证书过期测试结果图

1. 加密通道测试

VPN客户端ping内网主机，wireshark在VPN服务器外口截包检查。如下图4-6所示，VPN客户端ping内网主机，并在wireshark选择网卡docker1进行截包检查，发现能通信、经隧道封装、隧道为TLS，符合要求。

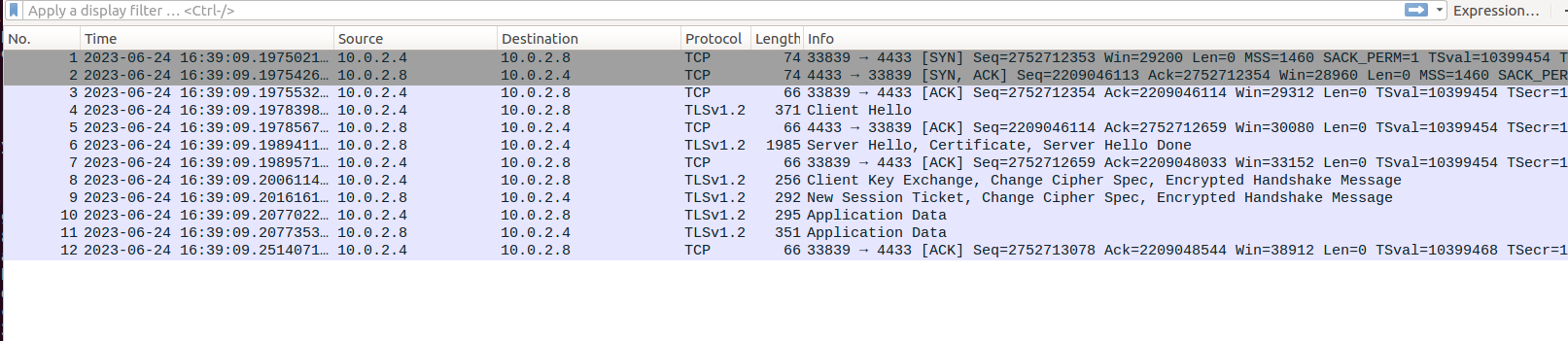


图4-6 SSL隧道加密通信测试结果图

## 稳定性测试

这一部分的设计受益于文件描述符表的功能，使得文件描述符能够得到复用，从而使程序能够在大量的数据包冲击下正常工作。这样的设计避免了重复获取文件描述符和打开管道文件的过程，防止程序耗尽文件描述符资源。在Linux系统中，每个进程最多持有1024个文件描述符，文件描述符的作用类似于Windows系统中的句柄。

如果程序没有实现文件描述符的复用功能，它将无法通过这一部分的测试。大多VPN程序的设计在获取文件描述符时会重新打开管道文件来获取新的文件描述符，并且不会释放之前使用的文件描述符，而这样的程序无法通过这一测试。

这项测试的方法是同时在HostU0和HostU1上执行命令"ping 192.168.60.101 -f"，其中参数"-f"使得程序每1秒发送100个ICMP echo数据包，而不是正常情况下的1秒1个。

测试结果如图4-7所示，HostU发送了近7000个数据包，HostX发送了近9000个数据包，而VPN程序仍然能够正常运行，并且平均延迟保持在3ms左右，该程序成功通过了这一测试。

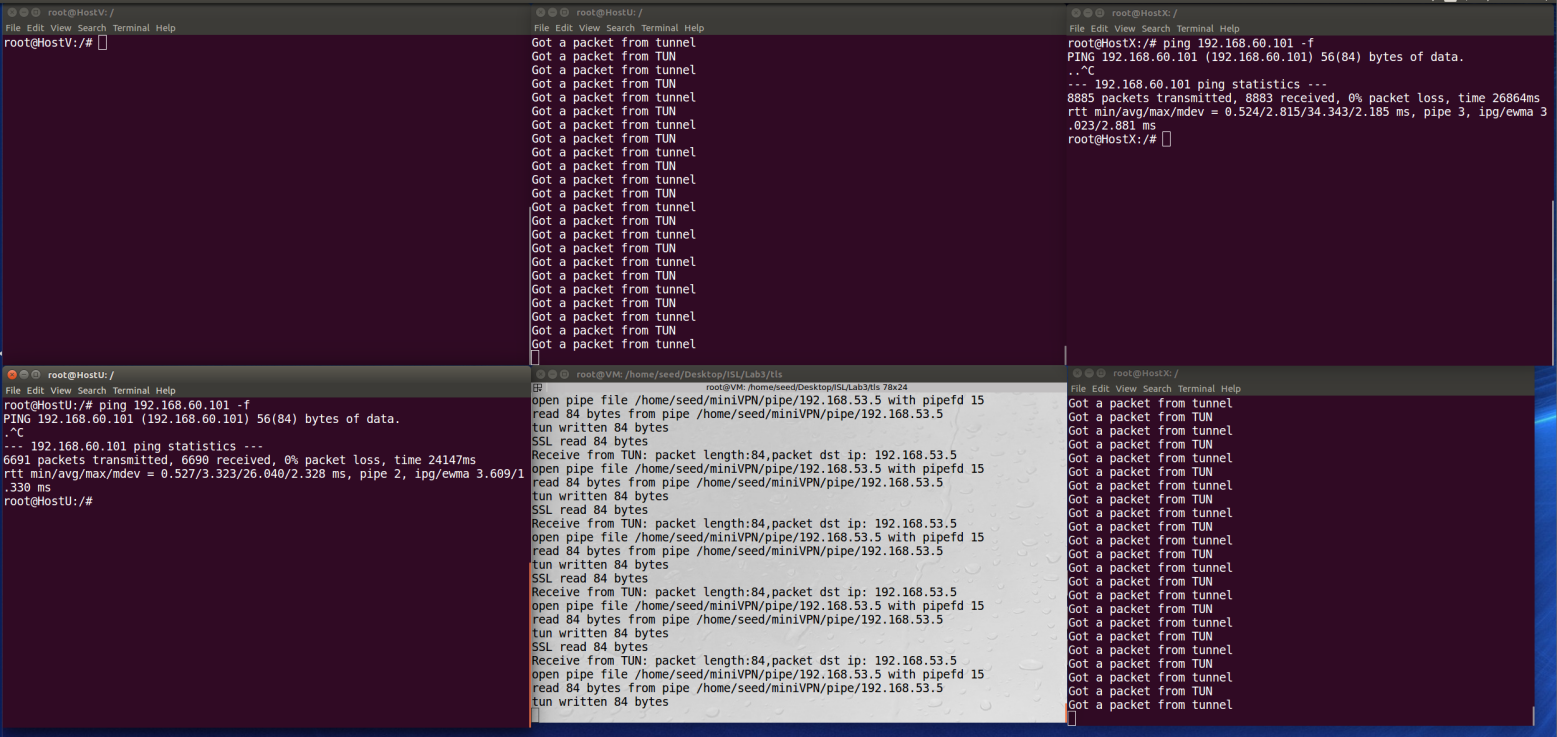


图4-7 ping -f命令测试

# 体会与建议

## 心得体会

本次实验使我深入了解了TLS VPN的工作流程，同时为了实现证书的验证和证书的生成，我比过去更加深入的了解了公钥基础设施的工作流程，了解了数字签名的工作流程。同时，为了实现进程间的通信，我第一次接触了Linux环境下的共享内存和跨进程的锁机制，这些都使我受益匪浅，坦白说，Linux环境下的共享内存机制相比Windows环境下的共享内存要更加方便易用。

在这次实验过程中，我因为实验做到一半跑去上课，才意外发现了原先文件描述符获取机制的漏洞，发现了这个使得程序不能长时间工作的巨大缺陷，经过长时间的思考以及在脑海中模拟程序的运行逻辑，才终于怀疑可能是open函数重复打开同一个文件时获取到的文件描述符可能不是同一个，然后又在脑海中类比文件描述符与Windows环境下句柄的关系，才得出可能是文件描述符资源耗尽导致的这个问题，经过查找资料了解到Linux确实会限制每个进程获取文件描述符个数之后，我查看了程序无法正常运行时和正常运行时所持有的文件描述符的个数，终于定位了这个问题。这才使得我决定设计一个文件描述符复用的机制，来保证程序不会重复的获取文件描述符导致程序资源耗尽而无法正常运行。

在本次实验里，除了网络通信的问题以外，绝大多数问题都是由程序的并发导致的，因此这次程序的编写也使我学习了很多并发相关的知识。

## 意见建议

希望老师能更多的讲解公钥基础设施相关的知识，这一部分在程序编写初期使我困惑良久。