Socket Programming Final Project

Rong Nie

December 26, 2020

Contents

我的期末选题是多播聊天系统，具有文件传输功能。使用C++语言，实际上是主要是C，但是为了使用了STL标准模板库，所以使用C++。环境为Ubuntu 16.04 LTS 32bits。

1 Overview

该多播聊天系统中的所有节点使用同一份程序文件并且地位均等。也就是说，该系统是一个分布式系统，没有中心节点。

每两个节点之间通过Diffie-Hellman 算法生成一对对称密钥用于数据加密。

每个程序可以管理自己手中持有的所有密钥，选择是否启用某个密钥。对于需要加密的数据，一个程序会遍历所有启用的密钥，将消息加密并发送，即对于一条想发送的信息，最后会发送好几个消息，用不同的密钥加密。这样可以实现信息管控，如果不想让某些节点看到消息，那么不发送使用其密钥加密过的消息即可。

程序使用了unp库，openssl库的libcrypto库（加密需要），pthread库。

2 The Programming Details

2.1 Framework

程序由一个头文件，一个只有主函数的cpp和一个其他函数所在的cpp组成。此外还有一个辅助c程序，该c文件编译后为一个可执行文件，会显示机上所有网卡的index和名字，用于主程序的编译，后面会提到如何使用。

```
/
├── headfiles
│   └── mcchat.h
└── cppfiles
    ├── mcchat.cpp
    ├── main.cpp
    └── ifindex.c
```

代码目录结构如上

程序使用了pthread多线程库。主要是想让消息的收和发在两个线程中。发送线程负责从终端读取用户输入，做相关操作和发送消息。接收线程负责从网络中接收消息，也会发送消息，不过仅限于DH密钥交互时。

```
1    ret = pthread_create( &servtid, NULL, servthr, &prams
         );
2     if( ret != 0 ){
3         err_quit( "Create thread error!");
4     }
```

```
5       ret = pthread_create( &clitid , NULL, clithr , &
            cliprams );
6       if( ret != 0 ){
7           err_quit( "Create thread error!");
8       }
```

程序中创建两个socket，一个加入多播组，用于消息接收，一个是普通的udp socket，用于消息发送。

所以收到消息后该消息的源地址是节点在局域网内的单播地址，这个地址是与主机一一对应的，足以标记身份。

2.2 Message Format

我们的每个消息都有1024 Bytes长度。也就是UDP头后有1024 Bytes。

消息类型共有7种，分别是节点退出消息，DH密钥生成发起消息，DH密钥生成回复消息，文件名消息，文件内容消息，文件尾消息，普通字符消息。 在这1024 Bytes中，前4字节是一个我们自定义的头。

```
1  struct mcheader
2  {
3          unsigned char mtype;
4          unsigned short mlen;
5  };
```

这个头理论上只有3字节，但是内存分配时会分配4字节。即sizeof的结果是4. 第一个成员unsigned char用于标记7种消息类型，第二个成员记录消息长度。我们的所有消息都是1024Bytes，其实大部分消息都是字符串型，0自然结尾，文件传输也是写满定长，但是文件的最后一个包无法写满，而且结尾字节不定，所以需要一个长度标记。头部不参与加密。

对于需要加密的信息，之前已经说过，一条信息会用好几个密钥分别加密分别发送，因为是多播情况，所以一个节点也会收到所有的被加密消息。那么他如何知道哪一条被加密消息是用自己与消息源之间的共享密钥加密的呢？

我采用的方法是在头部之后（头部之后的字节全参与加密）的4个字节填充为固定的“abcd”。这样，一个节点对所有的消息都用对应密钥解密，解密后如果头部后的4个字节是“abcd”，那么我们认为解密成功。

前三种消息不加密。

节点退出消息头部后字节无意义。

DH密钥生成发起消息需要将DH算法需要的质数P和一个公钥传输出去（所有的DH算法使用生成元约定为2，无需传输）。在openssl 库中，P和公钥都是一

数据包结构示意图

个BIGNUM结构体，有相关函数可以将这个类型与字符串类型进行转化。但是，这个字符串并不是一个所谓的以0结尾的可读字符串，所以需要将其长度也要传送出来。DH密钥生成回复消息中除了要回复给发起方的公钥外（长度信息同样需要放在消息中），还要加入其对应的请求消息的源地址，我们以点分十进制的方式放在消息中。

可结合数据包结构示意图理解。（图片没有按照真实长度比例！）
对发送buffer的写入大多通过memmove函数。

2.3 Socket Programming

socket加入多播组使用的是unp库的mcast_join等函数，收发使用的就是普通的sendto，recvfrom。

```
1 //in mcchat.cpp
2 bind(sockfd, servaddr,sizeof(sockaddr_in));
3 mcast_join(sockfd, servaddr, sizeof(sockaddr_in),
      IF_NAME,IF_INDEX);
4 mcast_set_loop(sockfd, 0);
```

这里的IF_NAME和IF_INDEX两个宏定义编译时需要在头文件中修改为合适的值，后面会提到具体如何做。

2.4 Key Management

我们使用一个STL的Map数据结构来存储所有的加密密钥。map的键是in_addr_t类型，也就是32位无符号整数，用来标记与谁共享该密钥。map的值是一个结构体。

```
1 std::map <in_addr_t,struct mckey> keytable;//global
    variable
2 struct mckey
3 {
4         DES_cblock key;
5         bool keyflg;
6 };
```

DES_cblock 是openssl库的一个数据结构，实际上等同于char[8]。而结构体中的bool类型就决定了我们发送数据时是否发送用该密钥加密的副本。当密钥建立完成时，是将其置为false的，也就是说需要使用命令将其置为true。

这个map是在main函数之外定义的全局变量，那么也就是说是两个线程都可以操作这个变量。我使用了pthread的pthread_mutex_t互斥锁来防止同时读写的问题。凡是对map的操作无论读写都要放在critical section中。

收到一个节点的退出消息时要将其对应的map项删掉。

2.5 Encryption and decryption

使用openssl的DES_系列函数。单次DES加密，加密模式为CBC。原因是DES系列函数提供了一个string to key的函数，而DH_系列函数最后生成的共享密钥是以char[]类型存储的，所以刚好可以关联。其实这个安全性不够，但是本身的实验性质，所以就使用这个。这个DES 加密，输入的原文字节数需要是8 的倍数。如果不足会补齐。我们的buffer 长度是1024，去掉4 字节头还剩1020，比1020小的8 的倍数是1016，再去掉4 字节“abcd”，只剩1012。 所以，无论是消息还是文件传输，单次最多1012 字节。

2.6 File Transmission

文件操作采用的是文件描述符+open+read、write函数的形式。每次读文件都读1012字节（即read函数中的参数设为1012），根据read返回值，当返回值小于1012时说明最后一部分被读出来。

因为udp无法保证数据包接收顺序，对于需要多次读的文件，直接循环多次读发送的话，发送的几个数据包间隔太短，可能接收顺序有问题。所以文件传输每两个分段之间要用usleep休眠一小会，这样接收顺序与发送顺序不一致错误发生的可能性就比较小。这样的话较大的文件可能要等待较长时间。

并且，不允许多播组内同时传输多个文件。因为我们没有在消息中加入任何标识。这个不允许同时传输由一个比较弱的机制实现。就是每个节点的程序中有一个bool变量，当收到文件名和文件内容类消息时，要将其置为false，收到文件末尾型消息时，将其置回true。当用户输入发送文件的指令时，发送进程会检测这个变量，如果true，那么可以发文件，如果false，就不行。这个机制是比较弱的，因为分布式系统之间存在通信延迟性和丢包的可能，可能会有一个

节点已经开始发送文件，但是另一个节点没有收到消息，也开始发文件的情况。这时就会发生错误。

此外，还有一个问题就是，程序对于文件名的是不做处理的。发送方将用户输入的文件名作为open函数的参数，如果打开成功，就会将整个文件名传给对方。对方拿到文件名后也直接将该文件名作为参数传入open。所以，如果，发送方的发送的文件不是当前目录下的文件，那么用户输入的文件名中就会有路径，这个路径也会传给接收方，但是接收方很有可能没有前面的路径文件夹，那么就会打开文件失败。所以文件传输必须将被传输文件与可执行文件放在同一目录下再行传输。

3 Some Defects

因为程序本身是再udp基础上的通信，而且我没有在udp之上再做一层reliable connection的机制，所以整个系统的鲁棒性是不太好的。上面提到了文件传输时的问题。其实密钥交互也没有一个双方检验机制来确认双方是否都生成了相同的密钥，只能通过多次密钥交互请求来尽可能保证，而且也无法保证是否我们想要通信的人是否收到了消息。

其次就是加密使用的openssl库的相关函数。我所使用的这些函数已经很老了，在官方文档上的描述是“deprecated”。不过这些函数参数简单，也不需要复杂的初始化等过程，并且可以很方便地将DES和DH密钥建立联系起来，这一点前面提到过。

4 About Compilation

在报告开头提到了环境，Ubuntu 16.04 LTS 32bits版本，需要unp库和openssl库。编译前，首先修改几个地方。

首先是makefile中的变量unppath，要修改成libunp.a的相对路径。如果与系统其他库的.a文件在一起就可以使用改为-lunp。
其次是在headfiles/mcchat.h中，最开头include语句的unp.h的位置也需要修改为相应位置。然后就可以使用make编译。此时编译后会生成两个可执行文件，一个main，一个ifindex。运行ifindex，会显示机上所有网卡的索引和名称。这时要再到headfiles/mcchat.h中，修改宏定义IF_NAME和IF_INDEX，修改为在局域网中的那张网卡的，ifindex运行时的对应值。然后再使用make编译即可，得到的main就是聊天程序。