基于 Wireshark 的 TCP 原理解析

试验过程

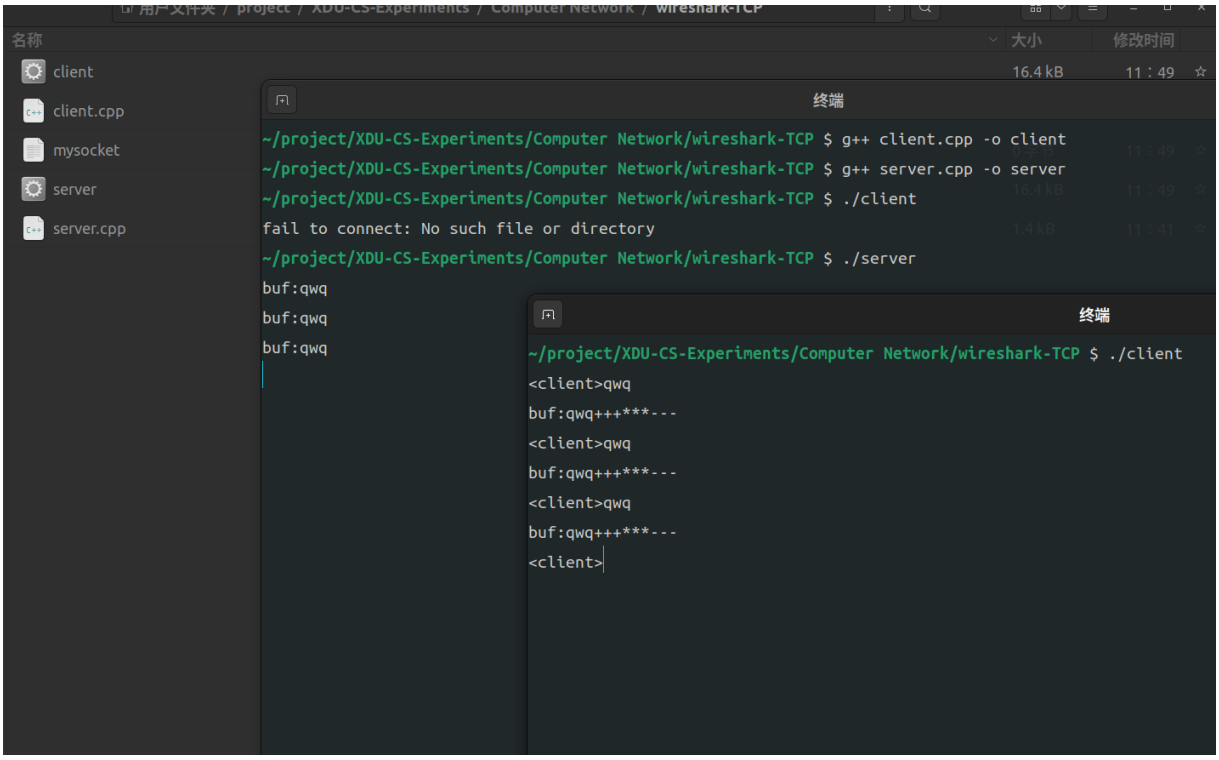
1. 试验环境为 Ubuntu22.04 LTS。
2. 编写用于本地 socket 通信的 server 和 client 程序。

```
 /*server.cpp*/
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<string.h>
#include<errno.h>
#include<sys/un.h>
#include<stdio.h>
#define N 64

int main(int argc, const char *argv[]) {
    int sockfd, connectfd;
    char buf[N];
    struct sockaddr_un serveraddr, clientaddr;
    socklen_t len = sizeof(clientaddr);
    sockfd = socket(AF_UNIX, SOCK_STREAM, 0);

    if(sockfd < 0)
      { perror("fail to socket"); return -1; }

    serveraddr.sun_family = AF_UNIX;
    strcpy(serveraddr.sun_path, "mysocket");

    if(bind(sockfd, (struct sockaddr*)&serveraddr, sizeof(serveraddr)) < 0)
      { perror("fail to bind"); return -1; }

    if(listen(sockfd, 5) < 0)
      { perror("fail to listen"); return -1; }

    if((connectfd = accept(sockfd, (struct sockaddr*)&clientaddr, &len)) < 0)
      { perror("fail to accept"); return -1; }

    while(1) {
        if(recv(connectfd, buf, N, 0) < 0)
          { perror("fail to recv"); return -1; }

        if(strncmp(buf, "quit", 4) == 0) break;

        buf[strlen(buf) - 1] = '\0';
        printf("buf:%s\n", buf);
        strcat(buf, "+++***---");

        if(send(connectfd, buf, N, 0) < 0)
          { perror("fail to send"); return -1; }
    }
    close(sockfd);
    return 0;
}
```

```
/*client.cpp*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<sys/types.h>
 #include<unistd.h>
 #include<sys/socket.h>
```

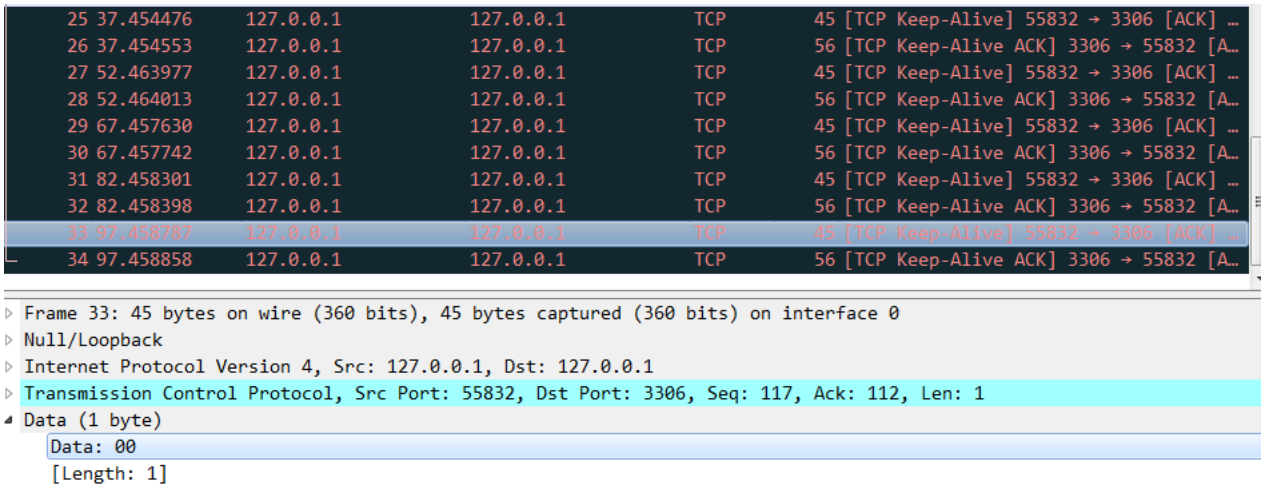
```c
#include<arpa/inet.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/un.h>

#define N 64

int main(int argc, const char *argv[]) {
    int sockfd;
    struct sockaddr_un serveraddr;
    char buf[N];

    sockfd = socket(AF_UNIX, SOCK_STREAM, 0);

    if(sockfd < 0)
      { perror("fail to sockfd"); return -1; }

    serveraddr.sun_family = AF_UNIX;
    strcpy(serveraddr.sun_path, "mysocket");

    if(connect(sockfd, (struct sockaddr*)&serveraddr, sizeof(serveraddr)) < 0)
      { perror("fail to connect"); return -1; }

    while(1) {
        printf("<client>");
        fgets(buf, N, stdin);

        if(send(sockfd, buf, N, 0) < 0)
          { perror("fail to send"); return -1; }

        if(strncmp(buf, "quit", 4) == 0) break;

        if(recv(sockfd, buf, N, 0) < 0)
          { perror("fail to recv"); return -1; }

        printf("buf:%s\n", buf);
    }
    close(sockfd);
    return 0;
}
```

3. 分别在两个不同的控制台上通过运行 server 和 client 程序，运行截图如下：



4.打开 Wireshark 监听 loopback。

实验结果

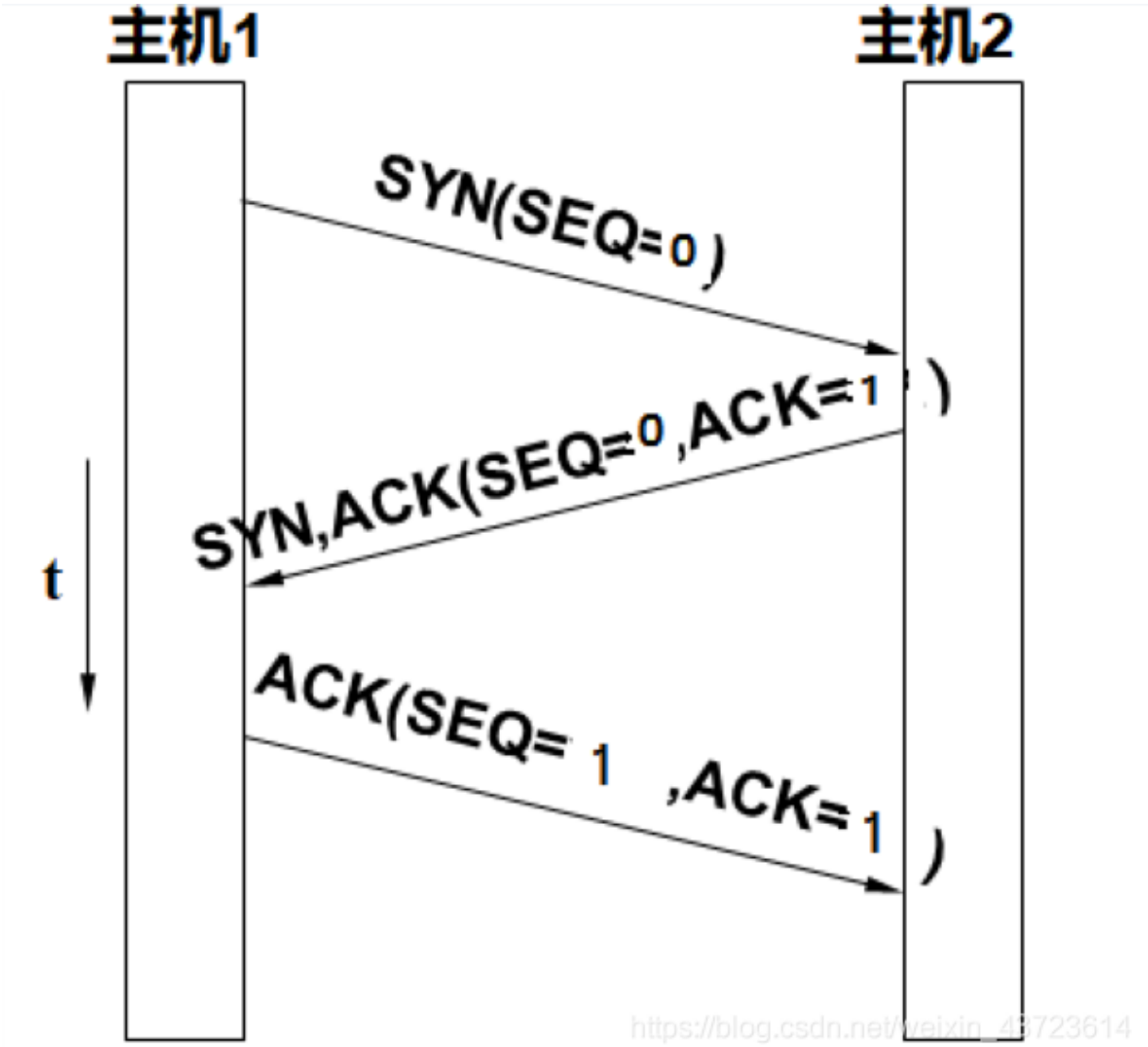


1、2、3前三条数据包表示三次握手的过程，12、16、17、18这四个数据包表示四次挥手的过程，可转化成下面的分析图解；从中可以看出序列号变化的规律，比如4号数据包Seq=1,数据长度Len=256；1+256=257，正好对应了5号数据包的Seq=257；4号数据包到11号数据包的序列号变化都符合这个规律，等于上一个序列号和数据长度之和。至于13到16号确认帧数据包是对前面客户端发送数据包(PSH=1)的应答出现在最后，是因为tcp通信是缓存流机制，客户端收到确认帧会存在延时，而不是客户端send()函数发送数据到缓存区马上会收到服务端的应答；考虑效率因素，等缓存区到达一定阈值后才可以传输，此时服务端才会发送应答，这就是确认帧ACK存在延时的原因。

实验结果分析

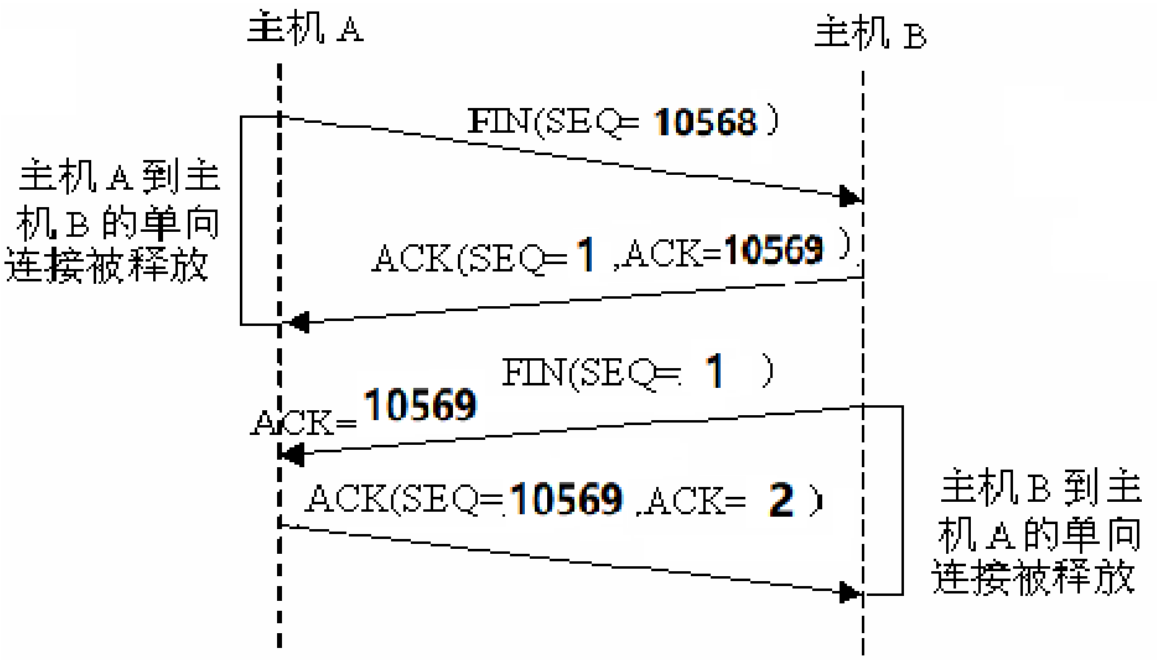
1. tcp建立连接：三次握手

   可以看到客户端(左侧)通过TCP首部发送一个SYN包(seq=0，一个初始化的随机值)作为建立连接的请求等待确认应答。接着服务器(右侧)发送ACK包确认应答，发送SYN包请求连接，其中seq=0，ack=1(seq=0是服务端随机初始化的一个值，ack=1表示已经收到序列号为0的数据包，期待下一帧收到的序列号为1)。最后客户端(左侧)针对SYN包发送ACK包(seq=1, ack =1)确认应答，发送序列号为1的所需确认帧。



2. tcp释放连接：四次挥手

   通常情况下释放TCP连接需要四个tcp端，每个方向上一个FIN和一个ACK。但是，第一个ACK和第二个FIN有可能被组合在同一个段中，从而减低到3个，比如12、17、18号数据包也是四次挥手的体现。结合抓包图可以看出，客户端捎带发出FIN,此时SEQ=10477+91=10568（存在最后一段数据长度Len=91）;服务端回应ACK=10569,表示已经收到10568seq的请求，释放单方向连接。同理，服务端捎带发出FIN,此时SEQ=1，最后客户端相应ACK=2，表示已经收到SEQ=1的请求，从而释放另一方单向连接。至此，完成TCP连接释放。



TCP 连接的释放过程