****

**计算机网络实验报告**

Socket编程基础



**学 院 国际工程学院**

**年 级 2018**

**专 业 计算机技术**

**学 号 2018229038**

**姓 名 李亚蓉**

**2018年 11月22 日**

**Socket编程基础**

1. **实验目的**

1、使用C语言基于Socket编写 简单的 TCP 和 UDP程序，实现服务器和客户端的简单的信息交互。

2、通过Wireshark软件分别对自己编写的TCP和UDP程序进行抓包

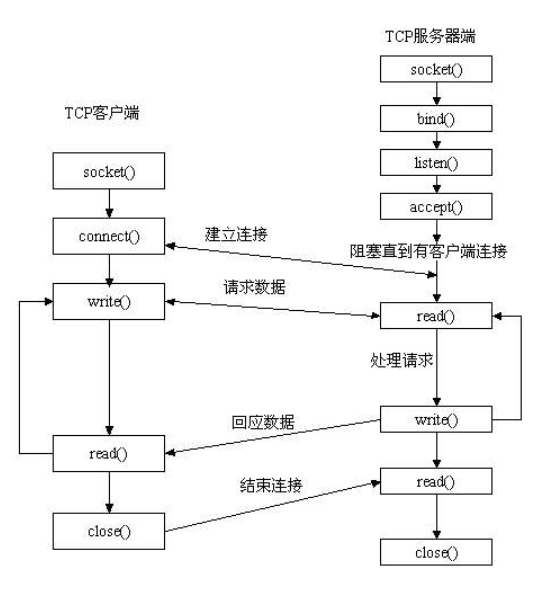
（1）观察TCP和UDP协议上的差异，并截图对比

（2）找出TCP三次握手、TCP关闭链接时的四次挥手等数据报文，深入理解tcp/udp的原理。

1. **实验原理**

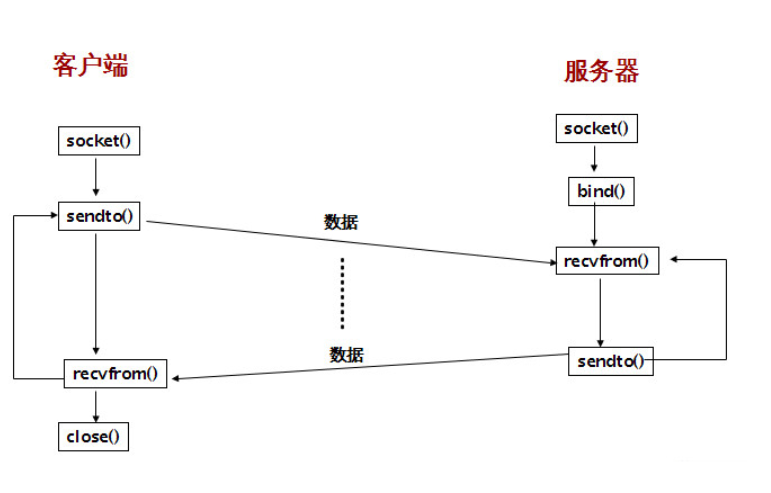
1、基于TCP（面向连接）

服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。

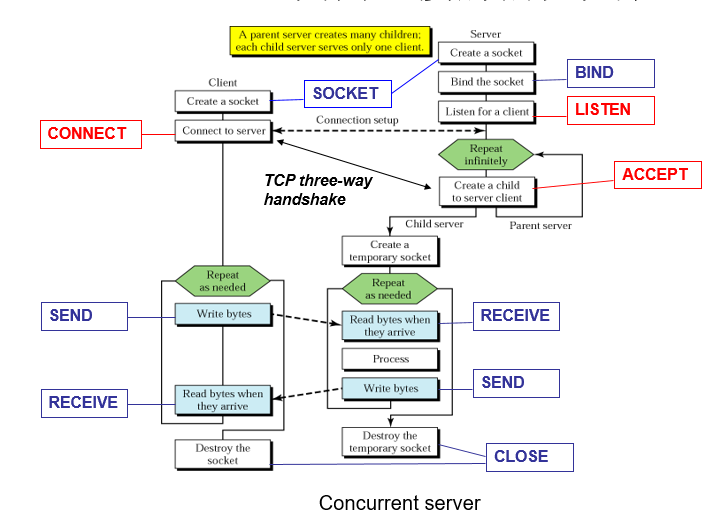


1. 基于UDP（面向无连接）

服务器先创建Socket，将Socket绑定（bind）到一个本地地址和端口上，等待数据传输（recvfrom）。这时候如果有个客户端创建Socket，并且向服务器发送数据（sendto），服务器就建立了连接，实现了数据的通信，连接结束后关闭连接。



1. **实验过程**



1. 基于TCP（面向连接）

（1）服务器端初始化Socket；

（2）与端口绑定(bind)，对端口进行监听(listen)；

（3）调用accept阻塞，等待客户端连接；

（4）客户端初始化一个Socket, 然后连接服务器；

（5） 客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据；

（6）最后关闭连接。

2、基于UDP（面向连接）

（1）服务器先创建Socket；

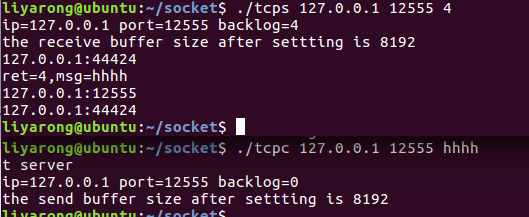
（2）将Socket绑定（bind）到一个本地地址和端口上，等待数据传输（recvfrom）；

（3）客户端创建Socket，并且向服务器发送数据（sendto），服务器就建立了连接，实现了数据的通信；

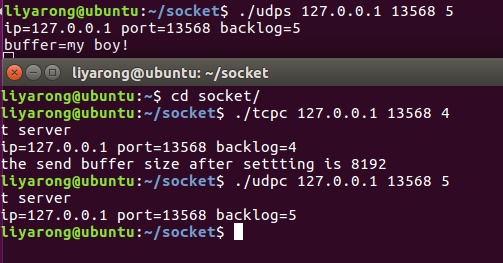
（4）连接结束后关闭连接。

1. **实验结果**
2. 使用C语言基于Socket编写 简单的 TCP 和 UDP程序，实现服务器和客户端的简单的信息交互。

（1）基于TCP的Socket编程

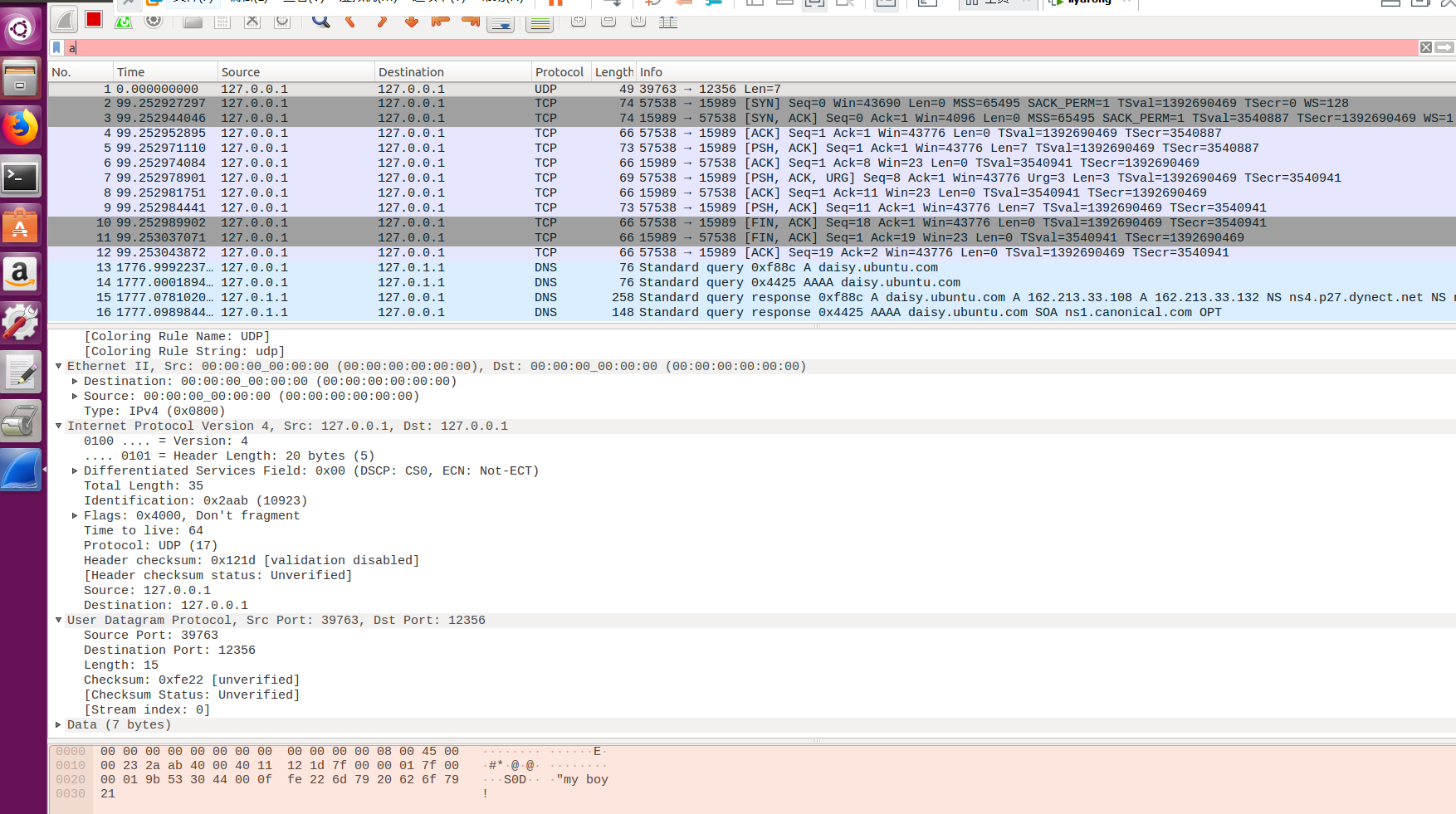


（2）基于UDP的Socket编程

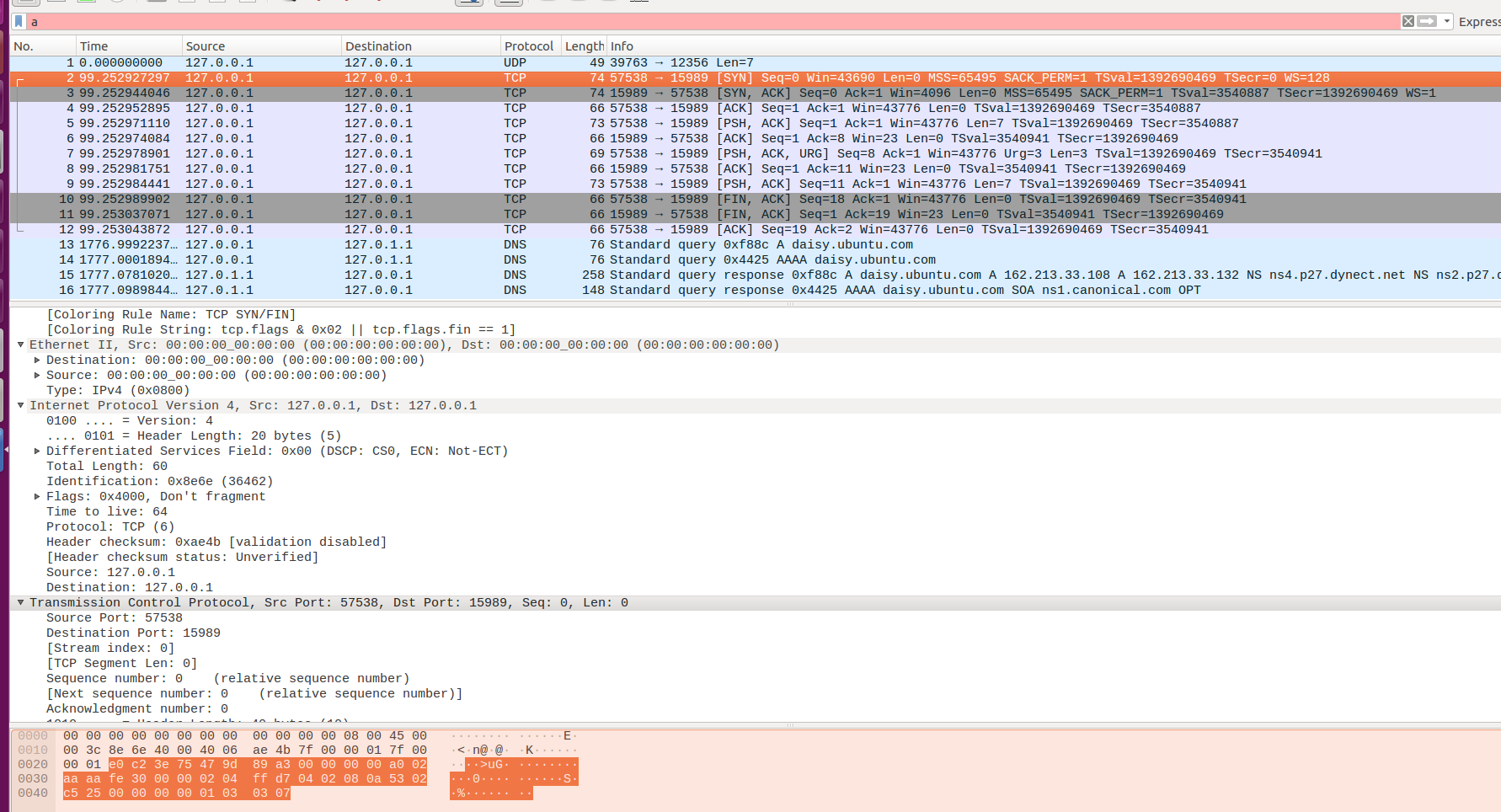


1. 通过Wireshark软件分别对自己编写的TCP和UDP程序进行抓包

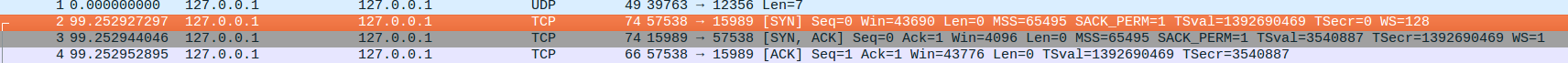
（1）UDP



（2）TCP



（3）TCP和UDP差异

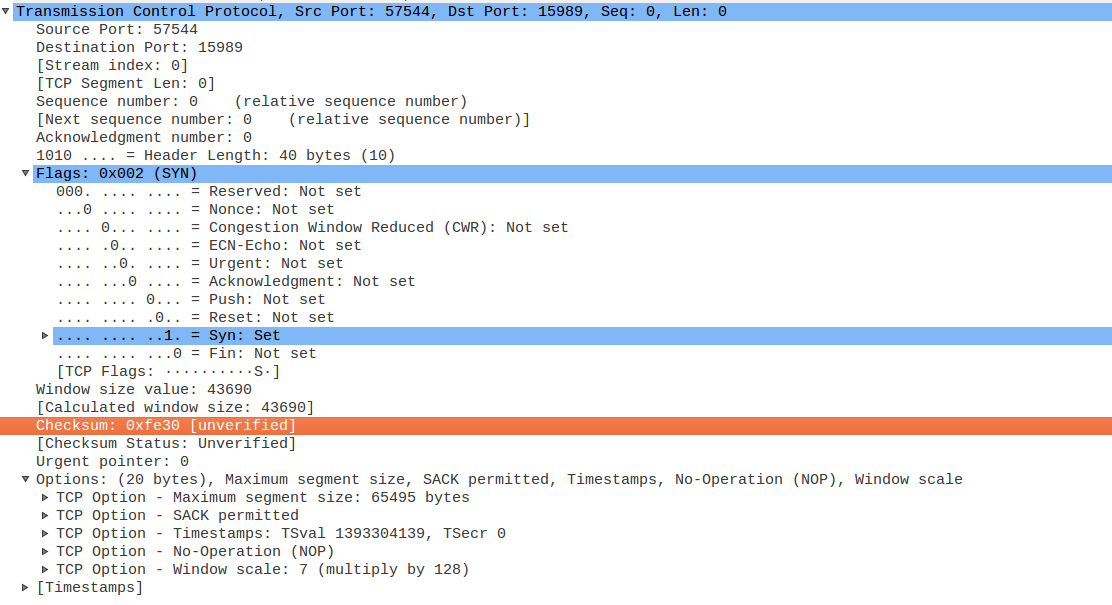


由上图截取的数据包可知，UDP是不面向连接的服务，可直接发送数据包。而TCP提供的是面向连接的服务，在发送数据包前需要进行三次握手建立连接。

**TCP三次握手：**



**第一次握手：**建立连接时，客户端发送SYN到服务器，并进入SYN\_SENT状态

数据包一些关键的属性如下：

**SYN ：**标志位，表示请求建立连接

**Seq = 0 ：**初始建立连接值为0，数据包的相对序列号从0开始，表示当前还没有发送数据

**Ack =0：**初始建立连接值为0，已经收到包的数量，表示当前没有接收到数据

**Window size:** 43690

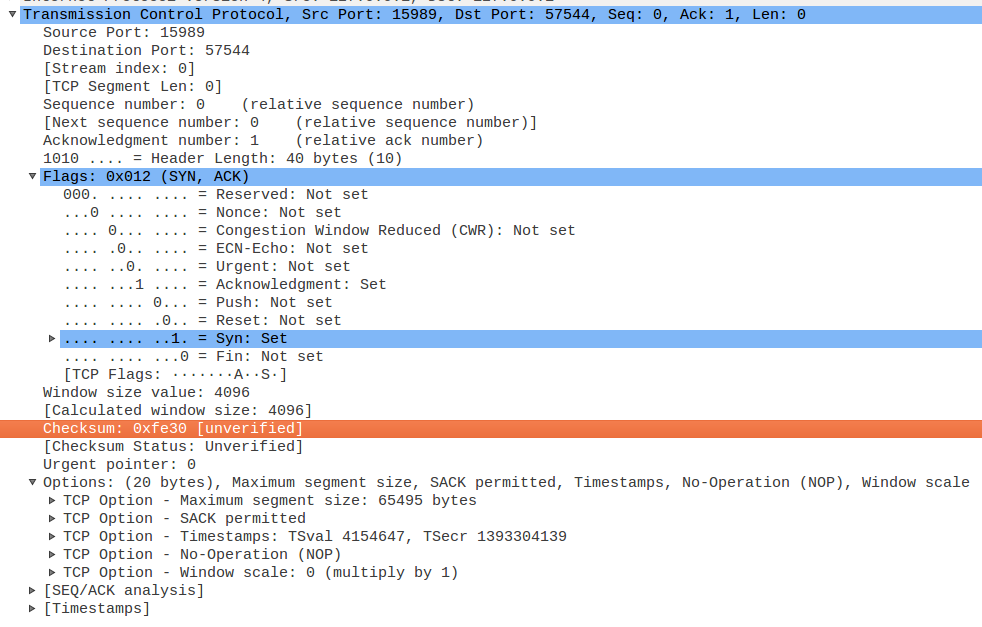
**Len = 0**

**Maximum segment size:** 65495 byte ，最长报文段，TCP包所能携带的最大数据量，不包含TCP头和Option。一般为MTU值减去IPv4头部(至少20字节)和TCP头部(至少20字节)

**SACK\_PERM = 1** 来自SACK permitted

**windows scale :** 7 (multiply by 128)： 窗口扩张，放在TCP头之外的Option

**第二次握手：**服务器收到请求后，回送SYN+ACK信令到客户端，此时服务器进入SYN\_RECV状态；数据包一些关键的属性如下：



**[SYN + ACK]:** 标志位，同意建立连接，并回送SYN+ACK

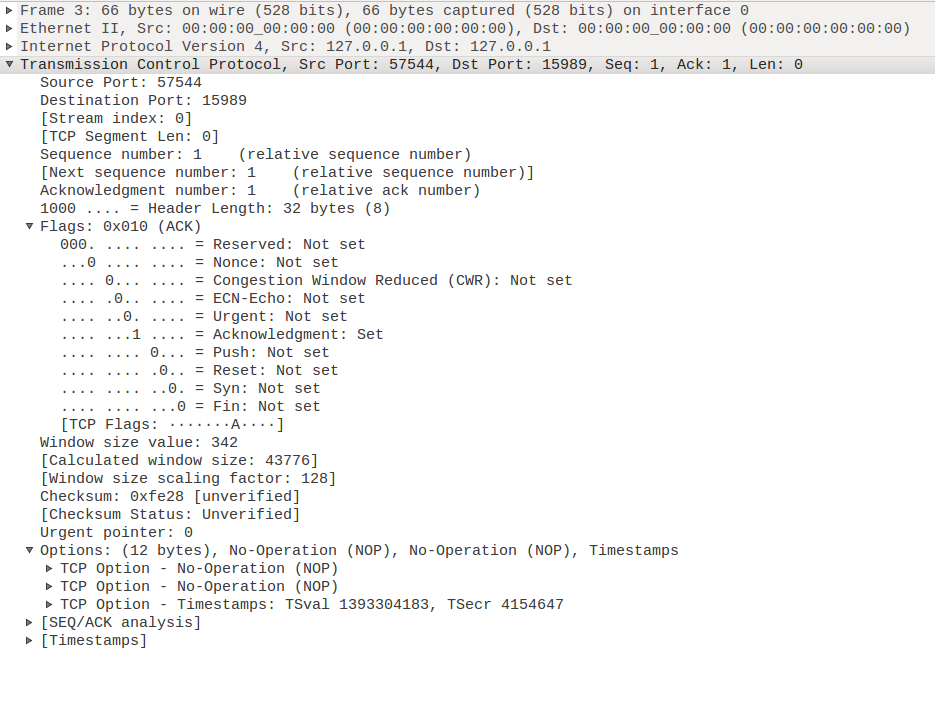
**Seq = 0 ：**初始建立值为0，表示当前还没有发送数据

**Ack = 1 :** 表示当前端成功接收的数据位数，虽然客户端没有发送任何有效数据，确认号还是被加1，因为包含SYN或FIN标志位。尽管客户端没有发送任何有效数据，确认号还是被加1，这是因为接收的包中包含SYN或FIN标志位（并不会对有效数据的计数产生影响，因为含有SYN或FIN标志位的包并不携带有效数据

其它标志和第一包相同。

**第三次握手：**客户端收到SYN+ACK包，向服务器发送确认ACK包，客户端进入ESTABLISHED状态，服务器收到请求后也进入ESTABLISHED状态，完成三次握手，此时TCP连接成功，客户端与服务器开始传送数据。

数据包一些关键的属性如下：



**ACK ：**标志位，表示已经收到记录

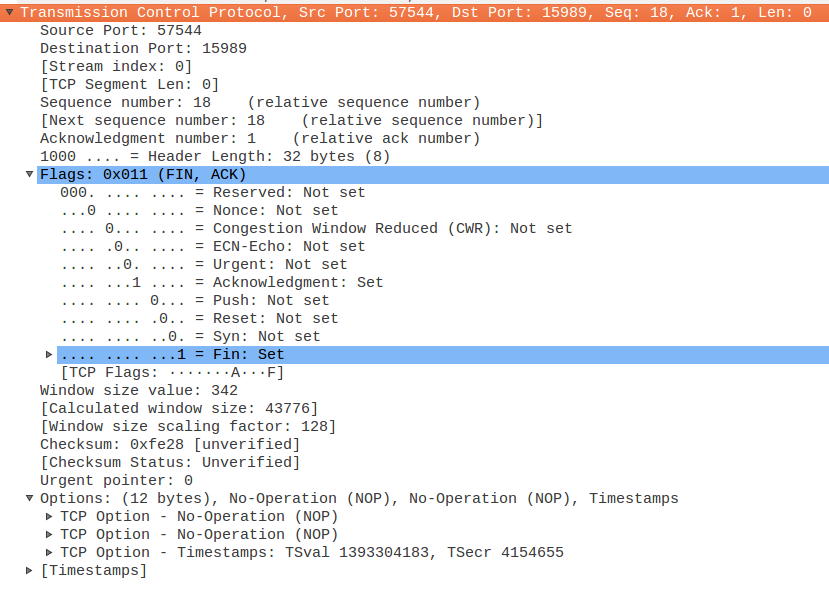
**Seq = 1 ：**表示当前已经发送1个数据

**Ack = 1 :** 表示当前端成功接收的数据位数，虽然客户端没有发送任何有效数据，确认号还是被加1，因为包含SYN或FIN标志位。尽管客户端没有发送任何有效数据，确认号还是被加1，这是因为接收的包中包含SYN或FIN标志位（并不会对有效数据的计数产生影响，因为含有SYN或FIN标志位的包并不携带有效数据)。

**TCP四次挥手**

**第一次挥手：** 客户端（127.0.0.1) 发送ACK ，通过FIN通知对方关闭连接

包关键属性如下：



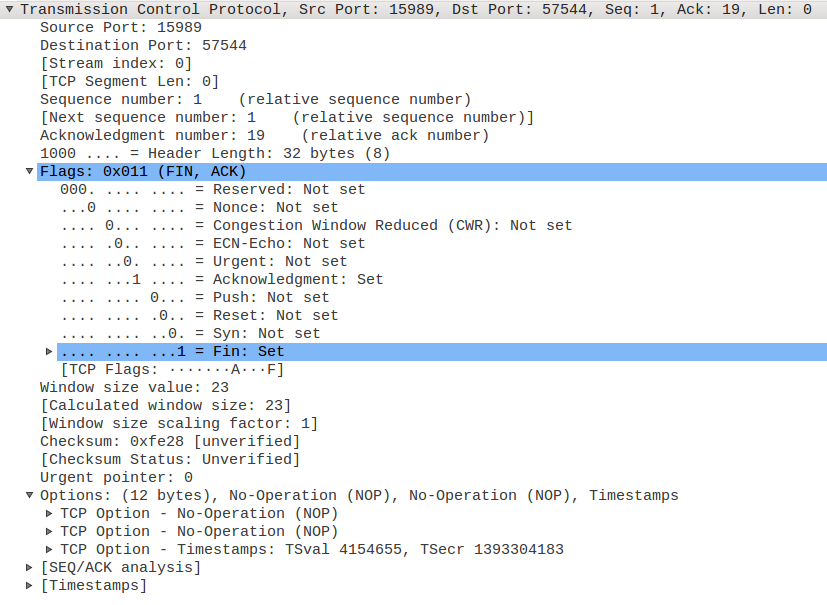
**ACK + FIN ：** 标志位

**seq = 18** ：已经发送的数据字节

**Ack = 1 ：**已经收到的数据字节+SYN包

**第二、三次挥手：**服务端（47.95.47.253)关闭的连接，通过FIN通知对方关闭连接

包关键属性如下：



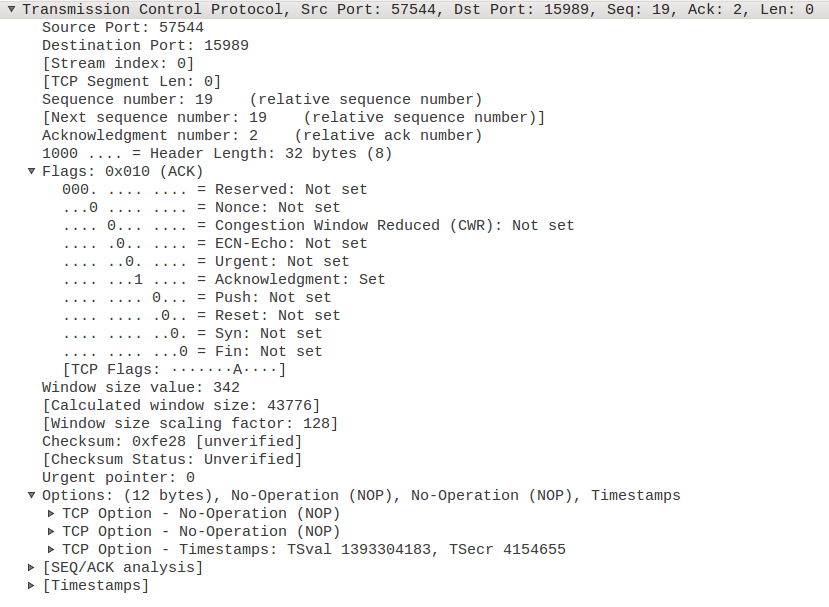
**ACK + FIN ：** 标志位

**Seq = 1：**已经发送的数据字节

**Ack = 19 ：**已经收到的数据字节+SYN包+ FIN包,Ack = 原Ack + 1, SYN标志位和FIN标志位也要占1位。

第四次挥手：客户端（47.95.47.253 )关闭连接，并发送ACK值和Ack确认号( Ack的值+1)

包关键属性如下：



**ACK ：** 标志位

**Seq = 19**：已经发送的数据字节

**Ack = 2** ：已经收到的数据字节+SYN包+ FIN包,Ack = 原Ack + 1, SYN标志位和FIN标志位也要占1位

1. **体会**

经过本次实验使用C语言基于Socket编写了简单的 TCP 和 UDP程序，实现服务器和客户端的简单的信息交互。通过Wireshark软件分别对自己编写的TCP和UDP程序进行抓包，更加清楚地了解到了TCP协议和UDP 协议的差异；通过找出并分析了TCP三次握手、TCP关闭链接时的四次挥手等数据报文，更加深入地理解了tcp/udp的原理。另外，在实验过程中，感谢老师和同学们的帮助，下次我会再接再厉！

源代码：

TCP SERVER

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <signal.h>

#include <unistd.h>

#include <stdlib.h>

#include <assert.h>

#include <stdio.h>

#include <string.h>

#define BUFFER\_SIZE 1024

void tsocket(int argc, const char \* argv[]);

int main(int argc, const char \* argv[]) {

tsocket(argc,argv);

return 0;

}

void tsocket(int argc, const char \* argv[]){

if(argc < 3){

exit(-1);

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

int backlog = atoi(argv[3]);

std::cout << "ip=" << ip << " port="<<port << " backlog=" << backlog << std::endl;

int fd;

int check\_ret;

fd = socket(PF\_INET,SOCK\_STREAM , 0);

assert(fd >= 0);

struct sockaddr\_in address;

bzero(&address,sizeof(address));

//转换成网络地址

address.sin\_port = htons(port);

address.sin\_family = AF\_INET;

//地址转换

inet\_pton(AF\_INET, ip, &address.sin\_addr);

//设置socket buffer大小

int recvbuf = 4096;

int len = sizeof( recvbuf );

setsockopt( fd, SOL\_SOCKET, SO\_RCVBUF, &recvbuf, sizeof( recvbuf ) );

getsockopt( fd, SOL\_SOCKET, SO\_RCVBUF, &recvbuf, ( socklen\_t\* )&len );

printf( "the receive buffer size after settting is %d\n", recvbuf );

//绑定ip和端口

check\_ret = bind(fd,(struct sockaddr\*)&address,sizeof(address));

assert(check\_ret >= 0);

//创建监听队列，用来存放待处理的客户连接

check\_ret = listen(fd, backlog);

assert(check\_ret >= 0);

struct sockaddr\_in addressClient;

socklen\_t clientLen = sizeof(addressClient);

//响应连接请求,接受连接，阻塞函数

int connfd = accept(fd, (struct sockaddr\*)&addressClient, &clientLen);//Addr将在函数调用后被填入对方的地址信息,ip,port

if(connfd < 0){

std::cout << "accept error";

}else{

//打印客户端信息

char showData[INET\_ADDRSTRLEN];

std::cout <<inet\_ntop(AF\_INET,&addressClient.sin\_addr,showData,INET\_ADDRSTRLEN)<<":" <<ntohs(addressClient.sin\_port)<<std::endl;

//接受数据

const int BUF\_LEN = 1024;

char sockBuf[BUF\_LEN];

size\_t ret;

memset(sockBuf, '\0', BUF\_LEN);

ret = recv(connfd, sockBuf, BUF\_LEN-1, 0);

printf("ret=%ld,msg=%s\n",ret,sockBuf);

memset(sockBuf, '\0', BUF\_LEN);

ret = recv(connfd, sockBuf, BUF\_LEN-1, MSG\_OOB);

printf("ret=%ld,msg=%s\n",ret,sockBuf);

memset(sockBuf, '\0', BUF\_LEN);

ret = recv(connfd, sockBuf, BUF\_LEN-1, 0);

printf("ret=%ld,msg=%s\n",ret,sockBuf);

//获取本地socket信息

struct sockaddr\_in tmpAddress;

clientLen = sizeof(tmpAddress);

getsockname(fd, (struct sockaddr\*)&tmpAddress, &clientLen);

std::cout <<inet\_ntop(AF\_INET,&tmpAddress.sin\_addr,showData,INET\_ADDRSTRLEN)<<":" <<ntohs(tmpAddress.sin\_port)<<std::endl;

//获取远端socket信息

getpeername(connfd,(struct sockaddr\*)&tmpAddress, &clientLen );

std::cout <<inet\_ntop(AF\_INET,&tmpAddress.sin\_addr,showData,INET\_ADDRSTRLEN)<<":" <<ntohs(tmpAddress.sin\_port)<<std::endl;

close(connfd);

}

close(fd);

}

TCP CLIENT

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <signal.h>

#include <unistd.h>

#include <stdlib.h>

#include <assert.h>

#include <stdio.h>

#include <string.h>

void tserver(int argc, const char \* argv[]);

int main(int argc, const char \* argv[]) {

tserver(argc,argv);

return 0;

}

void tserver(int argc, const char \* argv[]){

std::cout << "t server" << std::endl;

if(argc < 3){

exit(-1);

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

int backlog = atoi(argv[3]);

std::cout << "ip=" << ip << " port="<<port << " backlog=" << backlog << std::endl;

int fd;

int check\_ret;

fd = socket(PF\_INET,SOCK\_STREAM , 0);

assert(fd >= 0);

int sendbuf = 4096;

int len = sizeof( sendbuf );

setsockopt( fd, SOL\_SOCKET, SO\_SNDBUF, &sendbuf, sizeof( sendbuf ) );

getsockopt( fd, SOL\_SOCKET, SO\_SNDBUF, &sendbuf, ( socklen\_t\* )&len );

printf( "the send buffer size after settting is %d\n", sendbuf );

struct sockaddr\_in address;

bzero(&address,sizeof(address));

//转换成网络地址

address.sin\_port = htons(port);

address.sin\_family = AF\_INET;

//地址转换

inet\_pton(AF\_INET, ip, &address.sin\_addr);

check\_ret = connect(fd, (struct sockaddr\*) &address, sizeof(address));

assert(check\_ret >= 0);

//发送数据

const char\* oob\_data = "abc";

const char\* normal\_data = argv[3];

send(fd, normal\_data, strlen(normal\_data), 0);

send(fd, oob\_data, strlen(oob\_data), MSG\_OOB);

send(fd, normal\_data, strlen(normal\_data), 0);

close(fd);

}

UDPSERVER

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <signal.h>

#include <unistd.h>

#include <stdlib.h>

#include <assert.h>

#include <stdio.h>

#include <string.h>

//using namespace std;

#define BUFFER\_SIZE 1024

void tsocket(int argc, const char \* argv[]);

int main(int argc, const char \* argv[]) {

tsocket(argc,argv);

return 0;

}

void tsocket(int argc, const char \* argv[]){

if(argc < 3){

exit(-1);

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

int backlog = atoi(argv[3]);

std::cout << "ip=" << ip << " port="<<port << " backlog=" << backlog << std::endl;

int fd;

int check\_ret;

fd = socket(PF\_INET,SOCK\_DGRAM , 0);

assert(fd >= 0);

struct sockaddr\_in address;

bzero(&address,sizeof(address));

//转换成网络地址

address.sin\_port = htons(port);

address.sin\_family = AF\_INET;

//地址转换

inet\_pton(AF\_INET, ip, &address.sin\_addr);

//绑定ip和端口

check\_ret = bind(fd,(struct sockaddr\*)&address,sizeof(address));

assert(check\_ret >= 0);

while(1){

char buffer[BUFFER\_SIZE];

struct sockaddr\_in addressClient;

socklen\_t clientLen = sizeof(addressClient);

memset(buffer, '\0', BUFFER\_SIZE);

//获取信息

if(recvfrom(fd, buffer, BUFFER\_SIZE-1,0,(struct sockaddr\*)&addressClient, &clientLen) == -1)

{

perror("Receive Data Failed:");

exit(1);

}

printf("buffer=%s\n", buffer);

}

close(fd);

}

UDP CLIENT

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <signal.h>

#include <unistd.h>

#include <stdlib.h>

#include <assert.h>

#include <stdio.h>

#include <string.h>

void tserver(int argc, const char \* argv[]);

int main(int argc, const char \* argv[]) {

tserver(argc,argv);

return 0;

}

void tserver(int argc, const char \* argv[]){

std::cout << "t server" << std::endl;

if(argc < 3){

exit(-1);

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

int backlog = atoi(argv[3]);

std::cout << "ip=" << ip << " port="<<port << " backlog=" << backlog << std::endl;

int fd;

int check\_ret;

fd = socket(PF\_INET,SOCK\_DGRAM , 0);

assert(fd >= 0);

struct sockaddr\_in address;

bzero(&address,sizeof(address));

//转换成网络地址

address.sin\_port = htons(port);

address.sin\_family = AF\_INET;

//地址转换

inet\_pton(AF\_INET, ip, &address.sin\_addr);

//发送数据

const char\* normal\_data = argv[3];

if(sendto(fd,normal\_data,strlen(normal\_data),0,(struct sockaddr\*)&address,

sizeof(address)) < 0)

{

perror("Send File Name Failed:");

exit(1);

}

close(fd);

}