目录

[一 Notation Convention 2](#_Toc456102523)

[二 TCP/IP网络4层模型 3](#_Toc456102524)

[三 IP package format 6](#_Toc456102525)

[四 TCP package format 10](#_Toc456102526)

[五 socket编程 13](#_Toc456102527)

[六 TCP Connection establishment 16](#_Toc456102528)

[七 TCP Connection termination 19](#_Toc456102529)

[八 TCP Status 21](#_Toc456102530)

[九 tcpdump和wireshark抓包分析 23](#_Toc456102531)

### Notation Convention

HTTP: Hypertext Transfer Protocol，web传输协议，属于应用层。

TCP: Transmission Control Protocol，传输控制协议，属于传输层。

IP: Internet Protocol，IP网络协议，属于网络层。

Ethernet frame：以太网帧，属于最底层的链路层协议。

PDU: protocol data unit，协议数据单元。如TCP协议PDU，IP协议PDU等。

见wiki介绍：https://en.wikipedia.org/wiki/Protocol\_data\_unit

OCTET: any 8-bit sequence of data，可以理解为一个字节的数据。

MTU: Maximum transmission unit，最大传输size，如一个以太网包的最大size。

peer: 网络中的一个host，如一个PC host。

encapsulation: 封装，把fragment组装起来。

fragment: 分片，把一段数据分成多个fragment传输。

Congestion： 拥堵，网络堵塞。

duplex：双向的，二重的。

SYN: synchronous，同步，可以理解为client请求server应答。

ACK: acknowledge, 应答，确认。可以理解为server对client请求的回应。

FIN: finish，请求结束connect。

active close endpoint: 主动关闭连接的一端。

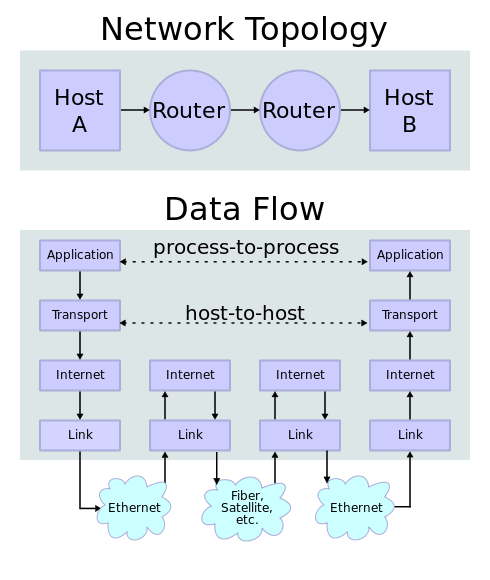
passive close endpoint: 被动关闭连接的一端，如server被动请求client发过来连接请求。

MSL：maximum segment lifetime，最大分节生命期，这是一个IP数据包能在互联网上生存的最长时间，超过这个时间将在网络中消失。MSL在RFC 1122上建议是2分钟，而源自berkeley的TCP实现传统上使用30秒，

### TCP/IP网络4层模型

见wiki <https://en.wikipedia.org/wiki/Internet_protocol_suite>

|  |  |
| --- | --- |
| 分层 | 协议 |
| Application Layer(应用层) | HTTP/FTP等 |
| Transport Layer(传输层) | TCP/UDP |
| Internet Layer(网络层) | IP |
| Link Layer(链路层) | Ethernet II |



Ethernet frame：以太网帧，属于最底层的链路层协议。

参加wiki介绍：https://en.wikipedia.org/wiki/Ethernet\_frame

以太网协议分为下面几种：

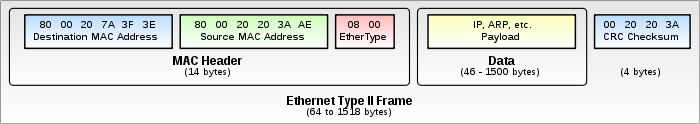
* Ethernet II
* Novell raw IEEE 802.3
* [IEEE 802.2 LLC](https://en.wikipedia.org/wiki/Ethernet_frame#IEEE_802.2_LLC)
* [IEEE 802.2 SNAP](https://en.wikipedia.org/wiki/Ethernet_frame#IEEE_802.2_SNAP)

一般的有线网络连接用的Ethernet II协议。我们用wireshark等抓包工具抓出来的都是以太网层的package。

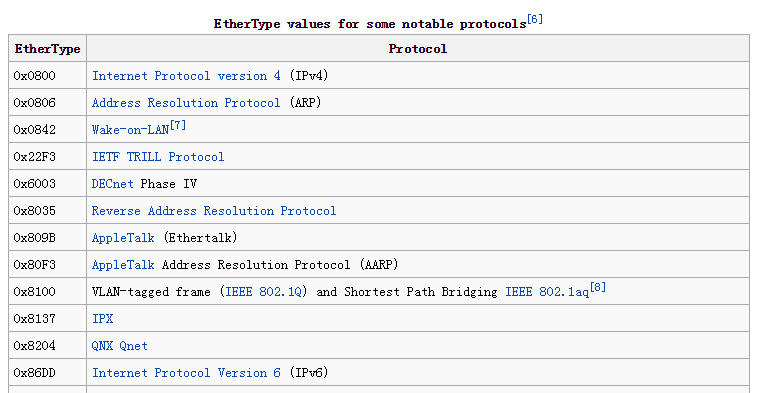
网络传输过程中的封包过程：



Ethernet II协议的package format:



* byte 0~6 , Dest Mac Addr, 6个字节，上图中的Dst Mac地址为80:00:20:7A:3F:3E。
* byte 7~12, Src Mac Addr, 6个字节，上图中的Src Mac地址为80:00:20:20:3A:A3。
* byte 13~14 , 协议type, <https://en.wikipedia.org/wiki/EtherType>里有列type对应的含义：



0x0800说明Frame Data里面是Ipv4协议package。

* byte 15 ~ byte 1514: Frame Data，上层的IP协议数据。图中的45~1500是指Frame Data最少有45个bytes，最长有1500个bytes，所以一个package里面有效数据最多有1500个bytes。MTU大小为1500 bytes，这是由以太网物理条件限制的。
* 最后面4个byte: crc checksum，用来校验数据完整性。用wireshark或tcpdump抓到的包里面，没有看到这4个byte数据，应该是给底层用的。

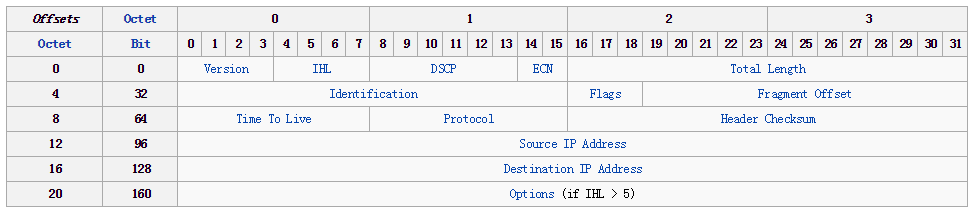
### IP package format

IP协议介绍：<https://en.wikipedia.org/wiki/Internet_Protocol>

IPv4: <https://en.wikipedia.org/wiki/IPv4>

Ipv6: <https://en.wikipedia.org/wiki/IPv6>

现在主要还是在用Ipv4，package format：

**Version**

IP协议的版本，Ipv4时，该值为4；IPv6时，该值为6.

**Internet Header Length (IHL)**

表示IP协议的header长度。单位是4个bytes。如IHL=5，表示IP package header为20(4\*5)个bytes。因为只有4个bit，所以IHL最大为15，即60(15\*4)个bytes。

**Differentiated Services Code Point (DSCP)**

New technologies are emerging that require real-time data streaming and therefore make use of the DSCP field. An example is Voice over IP (VoIP), which is used for interactive data voice exchange。可以理解为用来实现实时的数据传输。

**Explicit Congestion Notification (ECN)**

This field is defined in RFC 3168 and allows end-to-end notification of network congestion without dropping packets. ECN is an optional feature that is only used when both endpoints support it and are willing to use it. It is only effective when supported by the underlying network。可以理解为用来提示网络拥堵。

**Total Length**

This 16-bit field defines the entire packet size, including header and data, in bytes. The minimum-length packet is 20 bytes (20-byte header + 0 bytes data) and the maximum is 65,535 bytes — the maximum value of a 16-bit word.

整个ip package的size，包括header和data，单位为bytes。最小为20个bytes(只有header，没有data)。因为只有16位，所以最大只能为65535(0xFFFF)个bytes。

**Identification**

2 bytes。This field is an identification field and is primarily used for uniquely identifying the group of fragments of a single IP datagram.

用来标识该package属于那一组fragment，方便receipter去组装fragment。

**Flags**

A three-bit field follows and is used to control or identify fragments.

* bit 0: Reserved; must be zero.[note 1]
* bit 1: Don't Fragment (DF)
* bit 2: More Fragments (MF)

bit1用来标识该package没有做分片。

bit2用来表示该package有做分片，目前的package只是其中一个fragmant。如果是最后一个fragment，则设定为0。

**Fragment Offset**

The fragment offset field is measured in units of eight-byte blocks. It is 13 bits long and specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram. The first fragment has an offset of zero. This allows a maximum offset of (213 – 1) × 8 = 65,528 bytes, which would exceed the maximum IP packet length of 65,535 bytes with the header length included (65,528 + 20 = 65,548 bytes)。

用来记录是第几个fragment，receipter可以根据这个值去把fragment组装起来。第一个fragment的fragment offset为0。

**Time To Live (TTL)**

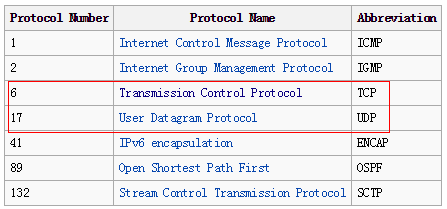
An eight-bit time to live field helps prevent datagrams from persisting on an internet. This field limits a datagram's lifetime. It is specified in seconds, but time intervals less than 1 second are rounded up to 1. When the datagram arrives at a router, the router decrements the TTL field by one. When the TTL field hits zero, the router discards the packet and typically sends an ICMP Time Exceeded message to the sender.

指定该IP package的生存周期，每经过一个router，TTL会被减1。当TTL为0时，router把这个包丢弃掉，并发送ICMP package给sender。

**Protocol**

This field defines the protocol used in the data portion of the IP datagram。

指定data里面数据的用的协议。如TCP/UDP/ICMP等。



**Header Checksum**

The 16-bit checksum field is used for error-checking of the header。

用来校验IP package header。

**Source address**

This field is the IPv4 address of the sender of the packet. Note that this address may be changed in transit by a network address translation device.

Sender的IPv4 地址，可能会给网络地址转换设备修改。

**Destination address**

This field is the IPv4 address of the receiver of the packet. As with the source address, this may be changed in transit by a network address translation device.

目的地receipter的IPv4地址。

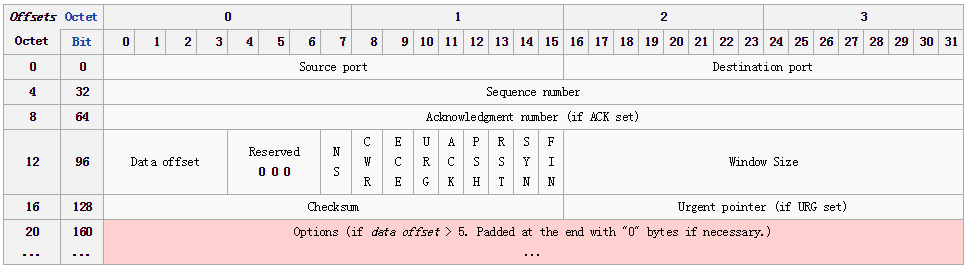
**Options**

可选区域。当IHL大于5时，会有这段，不常用。

### TCP package format

见wiki介绍：<https://en.wikipedia.org/wiki/Transmission_Control_Protocol>

TCP header：



**Source port (16 bits)**

发送端的端口号。在host上，端口号和application绑定，所以通过端口号，可以找到对应的application。有一些约定的端口号：

FTP (20 and 21)/SSH (22)/HTTP (80)

**Destination port (16 bits)**

target端的端口号

**Sequence number (32 bits)**

has a dual role:

1. If the SYN flag is set (1), then this is the initial sequence number. The sequence number of the actual first data byte and the acknowledged number in the corresponding ACK are then this sequence number plus 1.
2. If the SYN flag is clear (0), then this is the accumulated sequence number of the first data byte of this segment for the current session.

如果SYN标识位为1，此值就是初始化的package编号，client下一次发送的package编号，与server收到这个package后发送的ACK包的acknowledged num，都是在此基础上+1。

如果SYN标志位为0，那么这个是累计的目前session里发送的数据大小。可以理解为，如果上个发送的package sequence number为A，tcp data里有N个bytes的数据，即发送了N个bytes的数据，那么这次发送的package sequence number 为N+A。如果endpoint每次只是发送ACK包(即只有ACK flag)，而不发送data，则sequence num都是同一个值。

**Acknowledgment number (32 bits)**

if the ACK flag is set then the value of this field is the next sequence number that the receiver is expecting. This acknowledges receipt of all prior bytes (if any). The first ACK sent by each end acknowledges the other end's initial sequence number itself, but no data.

只有ACK标志为1时，这个字段的数据才有效。一般可以理解为，如果收到的包的sequence num为A，包里面的tcp data size为N个bytes，则要发送出去的包的acknowledge num为A+N，表明接收方收到了N个bytes的数据。因为发送方知道上次发送的sequence num，所以再收到这个ACK包后，根据acknowledge num的值，可以确认接收方是否有完全收到数据。

如client发送的package(有设PSH和ACK):

sequence num = 0xB048C654, acknowledge num = 0xCD7131BF，tcp包里面的data区域长度为0xA个bytes。

则server端收到这个ACK后，发送的package(有设ACK)应该是：

acknowledge num = 0xB048C654 + 0xA = 0xB048C65E。

**Data offset (4 bits)**

也就是头部长度，指出TCP负载（数据）的开始位置。以4字节为单位，如"0101"表示20（5\*4）字节位置的数据为负载开始，也就是头部长度为20字节。

**Reserved (3 bits)**

预留字段

**Flags (9 bits) (aka Control bits)**

NS: ECN-nonce隐藏保护

CWR: Congestion Window Reduced (CWR) flag，拥堵窗口减少标志位。

ECE: Explicit Congestion Notification，拥堵相关的设定flag。

URG: urgent，表示这是一个携有紧急资料的封包

ACK: ackownledge，当该标识位置1时，Acknowledgment 字段起作用。

PSH: Push function，请求把buffer里面的数据push给application。

RST: Reset the connection，重启链接。

SYN: Synchronize sequence numbers，SYN请求标识位。

FIN: No more data from sender，表面这是最后一个package。释放一个连接。

如果FIN为1，表示传送结束，然後双方发出结束回应进而正式终止一个TCP传送过程。

**Window size (16 bits)**

16位，接收窗口大小。此字段用来进行流量控制，单位为字节数，这个值是本机期望一次接收的字节数。这里一般称为“滑动视窗(Sliding Window)”

**Checksum (16 bits)**

16位，对整个TCP报文段，即TCP头部和TCP数据进行校验和计算，并由目标端进行验证。

**Urgent pointer (16 bits)**

16位，它是一个偏移量。指向后面是优先数据的字节，紧急指针指出在本报文段中的紧急数据的最后一个字节的序号，和序号字段中的值相加表示紧急数据最后一个字节的序号。

**Options (Variable 0–320 bits, divisible by 32)**

如果data offset设定大于5（即20个字节），header里面需要加入option字段。长度不定，但长度必须以字节记

**Padding**

不定长，填充的内容必须为0，它是为了保证header size能够被32整除。

### socket编程

Client endpoint:

int fd = 0;

char buffer[MAXLINE];

// create socket fd

socketfd = socket(AF\_INET, SOCK\_STREAM, 0);

/// set dest server info: ip and port

struct sockaddr\_in servaddr;

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr. sin\_port = htons(6666);

inet\_pton(AF\_INET, “172.21.34.233.”, &servaddr.sin\_addr);

// connect

connect(socketfd, (struct sockaddr\*)&servaddr, sizeof(servaddr));

// send data

while(1) {

fgets(buffer, sizeof(buffer), stdin);

if(strcmp(buffer, "exit\n") == 0) {

break;

}

if(send(socketfd, buffer, strlen(buffer), 0) == -1) {

printf("send msg to server error: %s(errno: %d)", strerror(errno), errno);

return -1;

}

}

// close connect

close(socketfd);

Server endpoint:

int listenfd,connectfd;

struct sockaddr\_in serverAddr;

char buffer[MAXLINE];

int n;

// create listen socket

listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

// bind port

memset(&serverAddr, 0 , sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serverAddr.sin\_port = htons(6666);

bind(listenfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr);

// listen socket

listen(listenfd, 10)

// accept connection of client, return connected fd

connectfd = accept(listenfd, (struct sockaddr\*)NULL, NULL);

// receive data from client

while(1) {

n = recv(connectfd, buffer, MAXLINE, 0)

if (n == 0) { //client close socket

printf("client closed connection\n");

break;

}

buffer[n] = '\0';

printf("receive msg from client: %s\n", buffer);

}

// close connect

close(connectfd);;

close(listenfd);

* client: socket 🡺 init dest sockaddr\_in 🡺 connect 🡺 send 🡺 close
* server: socket 🡺 init bind sockaddr\_in 🡺 bind 🡺 listen 🡺 accept 🡺 receive 🡺 close
* server端有2个socket fd： listen socket fd和connected socket fd。
* listen socket fd：

用来监听端口，如果有client connect请求过来，accept会返回一个connected socket fd，表示和client端已建立了连接。

1. connect socket fd：

表示和client建立了连接的socket fd。可以直接用来从client收发数据。

1. accept：

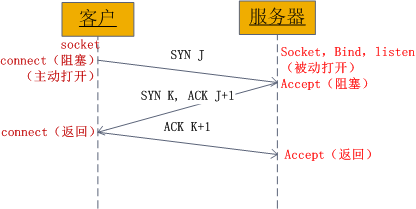
该函数是阻塞的，只有client有connect请求过来，才会return。

flow如下：



### TCP Connection establishment

client端和server端建立连接的过程称为3次握手，如下：



按照wiki上的介绍：

To establish a connection, TCP uses a **three-way handshake**. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

1. SYN: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A.
2. SYN-ACK: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number i.e. A+1, and the sequence number that the server chooses for the packet is another random number, B.
3. ACK: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A+1, and the acknowledgement number is set to one more than the received sequence number i.e. B+1.

At this point, both the client and server have received an acknowledgment of the connection. The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.

简单理解为下面flow:

1. client给server发送请求连接的package

SYN flag有置起来。sequence num为随机数A。

1. server收到client发过来的包后，发送给client一个响应包

SYN和ACK flag都有置起来。sequence num为随机数B，acknowledge num为A+1。

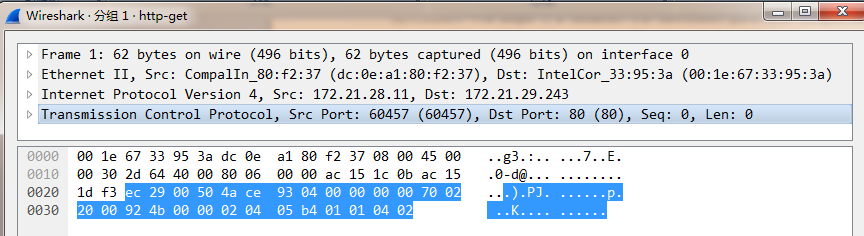
表示已收到client发过来的请求连接的包，并请求和client建立连接。

1. client收到server发过来的包后，发送给client一个响应包。

ACK flag有置起来。acknowledge num为B+1，sequence num为接受到的包的acknowledge num，即A+1。

下面是在PC机下用浏览器访问apach server时，用wireshark抓到的client和server建立连接的package：

1. package 1: client发送给server的package



深色区域是tcp package。

source port (byte 0~ 1) : 0xEC29 ，即60457。

dst port( byte 2~ 3) : 0x0050 , 即80， HTTP默认的端口号。

sequence num(byte 4~ 7): 0x4ACE9304，即tcp包编号。

acknowledge num(byte 8~ 11): 0x00000000，因为发送的是请求包，所以这里为0。

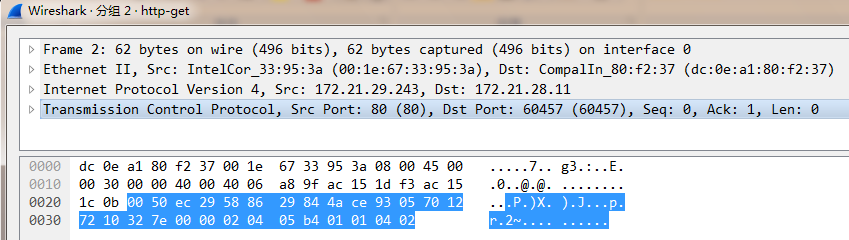
Data Offset (bit 96 ~ 100) : 0x7，表示offset为28（7\*4）个bytes。

ACK (bit 107): 0，表示不是ACK package。

SYN (bit 110): 1，表示是请求连接package。

FIN(bit 111): 0，表示不是请求连接断开package。

1. package 2: server发送给client的包



source port (byte 0~ 1) : 0x0050 ，即HTTP默认的port。

dst port( byte 2~ 3) : 0xEC29，和package 1 可以对应起来。

sequence num(byte 4~ 7): 0x58862984，即tcp包编号。

acknowledge num(byte 8~ 11): 0x4ACE9305，正好是上个package的sequence num +1。

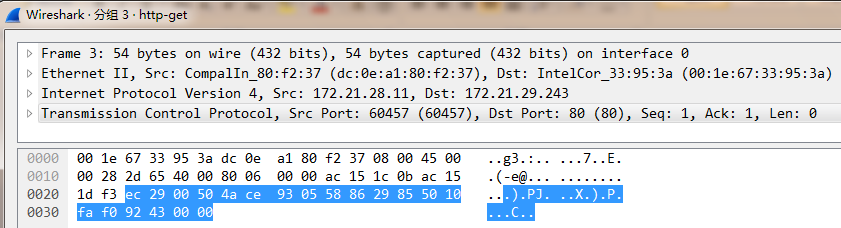
Data Offset (bit 96 ~ 100) : 0x7，表示offset为28（7\*4）个bytes。

ACK (bit 107): 1，表示是回应package。

SYN (bit 110): 1，表示是请求连接package。

FIN(bit 111): 0，表示不是请求连接断开package。

1. package 3: client发送给package的包



source port (byte 0~ 1) : 0xEC29 ，即60457。

dst port( byte 2~ 3) : 0x0050 , 即80， HTTP默认的端口号。

sequence num(byte 4~ 7): 0x4ACE9305，正好是package 2里面的acknowledge num。

acknowledge num(byte 8~ 11): 0x58862985，正好是package 2里面的sequence num +1。

Data Offset (bit 96 ~ 100) : 0x5，表示offset为20（5\*4）个bytes。

ACK (bit 107): 1，表示是ACK package。

SYN (bit 110): 0，表示不是请求连接的package。

FIN(bit 111): 0，表示不是请求连接断开package。

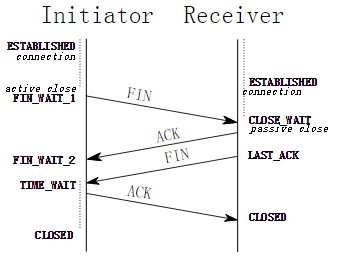
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| package num | direct | SYN | ACK | sequence  num | acknowledged num |
| 1 | C🡺S | 1 | 0 | 0x4ACE9304 | 0x00000000 |
| 2 | S🡺C | 1 | 1 | 0x58862984 | 0x4ACE9305 |
| 3 | C🡺S | 0 | 1 | 0x4ACE9305 | 0x58862985 |

### TCP Connection termination

The connection termination phase uses a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint. After the side that sent the first FIN has responded with the final ACK, it waits for a timeout before finally closing the connection, during which time the local port is unavailable for new connections; this prevents confusion due to delayed packets being delivered during subsequent connections.

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK.

当client或server请求连接断开的时候，需要发送FIN package，flow如下：



这个是TCP 四次握手的流程。左边是主动发起断开连接的一端(active close)，右边是被动断开连接的一端(passive close)。

1. 左边发送FIN请求断开连接包，进入FIN\_WAIT\_1的状态。
2. 右边收到FIN包后，发送ACK确认包。进入CLOSE\_WAIT状态。
3. 左边收到ACK确认包后，进入FIN\_WAIT\_2的状态，即等待右边发送FIN包。
4. 右边发送FIN请求断开连接包，进入LAST\_ACK状态。
5. 左边收到FIN包后，发送ACK确认包。进入状态TIME\_WAIT状态，即等待2\*MSL。

说明：

* 步骤2和步骤4有可能合并，当右边收到FIN包后，可以也选择关闭连接，即发送同时带有ACK和FIN flag的包。
* 步骤5 发送ACK后还要等待2\*MSL时间才进入closed状态的原因：

在进行关闭连接四路握手协议时，最后的ACK是由主动关闭端发出的，如果这个最终的ACK丢失，被动关闭端将重发最终的FIN，因此主动关闭端必须维护状态信息允许它重发最终的ACK。如果不维持这个状态信息，那么客户端将响应RST分节，服务器将此分节解释成一个错误（在java中会抛出connection reset的SocketException)。因而，要实现TCP全双工连接的正常终止，必须处理终止序列四个分节中任何一个分节的丢失情况，主动关闭的客户端必须等待2\*MSL时间才关闭真正关闭连接。如果左边发送的ACK，右边没有收到，则经过MSL后，右边会重新发送FIN，这样左边还可以收到FIN，进而继续重新发送ACK。主要是为了确保右边可以接收到左边发过来的ACK。

### TCP Status

tcp endpoint主要分为下面几个status：

**LISTEN**

(server) represents waiting for a connection request from any remote TCP and port.

**SYN-SENT**

(client) represents waiting for a matching connection request after having sent a connection request.

**SYN-RECEIVED**

(server) represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

**ESTABLISHED**

(both server and client) represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

**FIN-WAIT-1**

(both server and client) represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

**FIN-WAIT-2**

(both server and client) represents waiting for a connection termination request from the remote TCP.

**CLOSE-WAIT**

(both server and client) represents waiting for a connection termination request from the local user.

**CLOSING**

(both server and client) represents waiting for a connection termination request acknowledgment from the remote TCP.

**LAST-ACK**

(both server and client) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

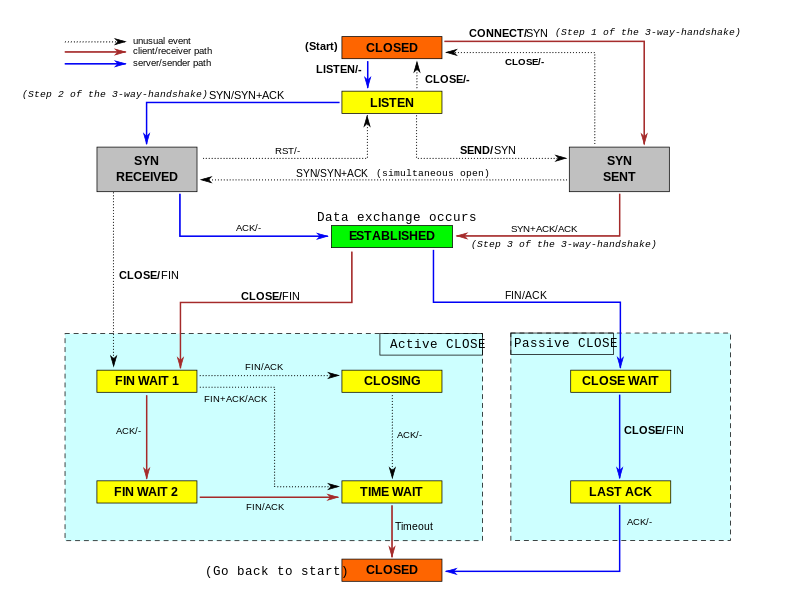
**TIME-WAIT**

(either server or client) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. [According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes known as two MSL (maximum segment lifetime).]

**CLOSED**

(both server and client) represents no connection state at all.

各个状态之间的转换关系：



FIN/ACK表示收到了一个FIN的包，之后发送一个ACK的包。

SYN+ACK/ACK表示收到了一个SYN+ACK的包，之后发送了一个ACK的包。

蓝线可以理解为server端的flow。

红线可以理解为client端的flow。

虚线是指不normal的flow。

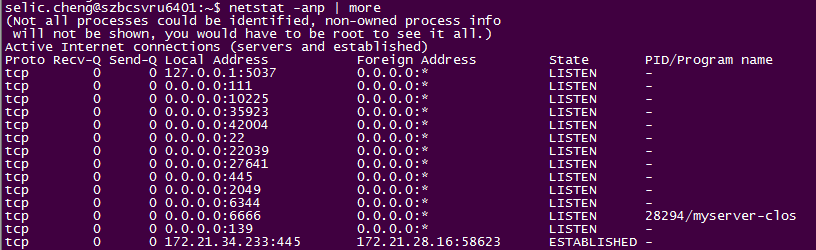
### tcpdump和wireshark抓包分析

我们一般用netstat、tcpdump和wireshark来分析网络中的包信息。

1. netstat查看网络状态

netstat -anp | more

==》 查看socket资源状态

可以看到myserve-close进程在监听6666端口。

1. tcpdump抓包

tcpdump -i eth0 dst 172.21.34.233 or src 172.21.34.233 -w test.pcap

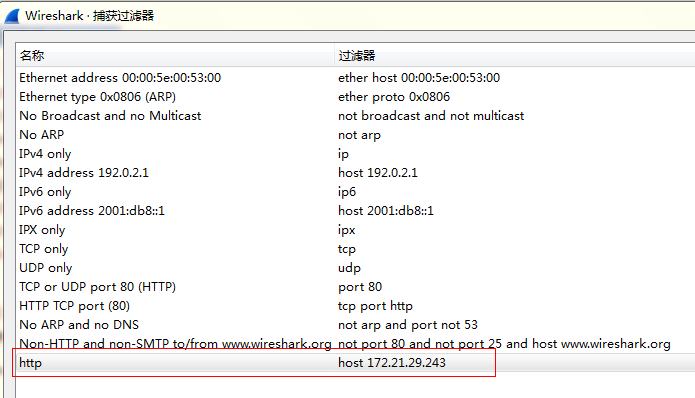
==》 监控echo网口，把发到172.21.34.233或从172.21.34.233的包抓出来，写到test.pcap文件里面。可以用wireshark工具图test.pcap文件。

1. wireshark是windows平台上的软件，用来抓取网络数据包。一般可以在linux平台上用tcpdump程序抓包，保存到pcap文件中，在windows平台再用wireshark来分析协议。

wireshark设定抓包的过滤规则

1. 启动软件后选择 捕获==》 捕获过滤器

新建一个过滤规则，如：

捕获规则的名字是http，内容是host 172.21.29.243。

表示只捕获src或dst ip地址为172.21.29.243的包。

capture syntax可以参考：

<https://www.wireshark.org/docs/wsug_html_chunked/ChCapCaptureFilterSection.html>

<http://www.tcpdump.org/manpages/pcap-filter.7.html>

如：

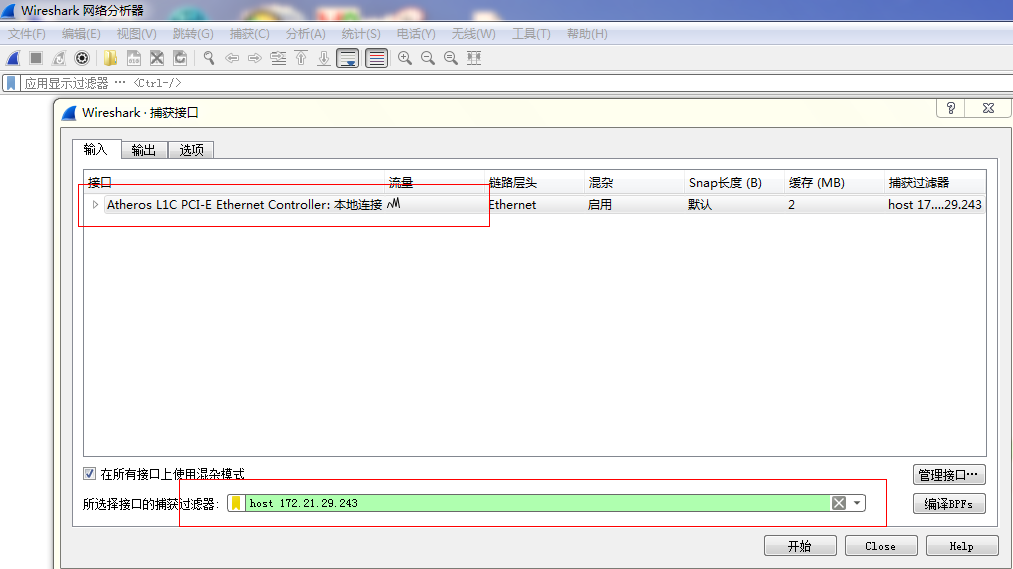
dst 172.21.34.233 or src ip 172.21.34.233

🡺 ip v4 src/dst address 172.21.34.233

dst port 80 or src port 80

🡺 端口号是80的包。

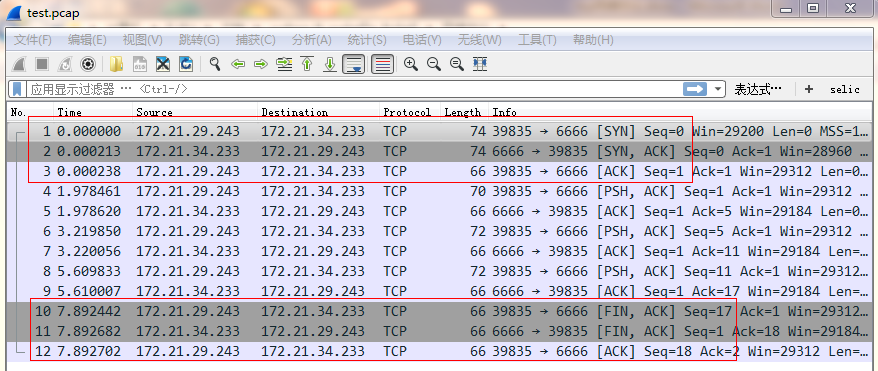
1. 捕获 ==》 选项，选择设定的过滤规则





test.pcap是在linux下用tcpdump命令抓到的包：

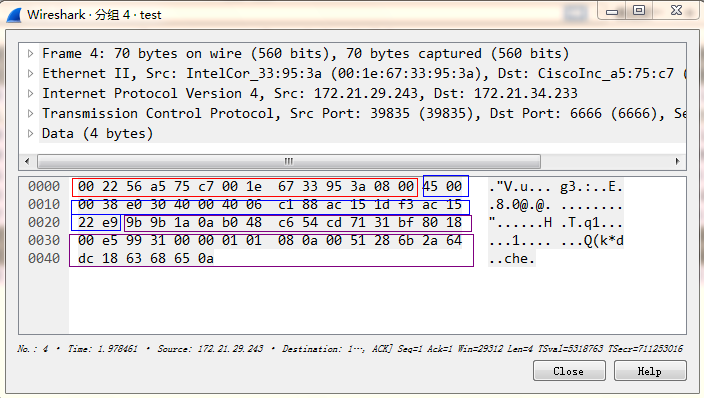
tcpdump -i eth0 dst 172.21.34.233 or src 172.21.34.233 -w test.pcap



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| package num | direct | ACK | PSH | SYNC | FIN | sequence num | acknowledged num | data |
| 1 | C🡺S | 0 | 0 | 1 | 0 | 0xB048C653 | 0x00000000 |  |
| 2 | S🡺C | 1 | 0 | 1 | 0 | 0xCD1731BE | 0xB048C654 |  |
| 3 | C🡺S | 1 | 0 | 0 | 0 | 0xB048C654 | 0xCD7131BF |  |
| 4 | C🡺S | 1 | 1 | 0 | 0 | 0xB048C654 | 0xCD7131BF | che\n |
| 5 | S🡺C | 1 | 0 | 0 | 0 | 0xCD7131BF | 0xB048C658 |  |
| 6 | C🡺S | 1 | 1 | 0 | 0 | 0xB048C658 | 0xCD7131BF | selic\n |
| 7 | S🡺C | 1 | 0 | 0 | 0 | 0xCD7131BF | 0xB048C65E |  |
| 8 | C🡺S | 1 | 1 | 0 | 0 | 0xB048C65E | 0xCD7131BF | debug\n |
| 9 | S🡺C | 1 | 0 | 0 | 0 | 0xCD7131BF | 0xB048C664 |  |
| 10 | C🡺S | 1 | 0 | 0 | 1 | 0xB048C664 | 0xCD7131BF |  |
| 11 | S🡺C | 1 | 0 | 0 | 1 | 0xCD7131BF | 0xB048C665 |  |
| 12 | C🡺S | 1 | 0 | 0 | 0 | 0xB048C665 | 0xCD7131C0 |  |

1. 前面的3个包是tcp建立连接的过程，即tcp三次握手。
2. 后面的3个包是tcp断开连接的过程，这里只有三次握手，而没有四次握手。是因为server收到FIN后，也同时断开连接，所以FIN和ACK在同一个包里了。
3. 除了第一个包，后面几个包都有ACK标识。
4. 带PSH标识的包，表明包里面有传输数据。

以第4个包为例，



* 红色为ethernet header

共14个bytes。

可以看到：

dst MAC addr: 00:22:56:A7:75:C6

src MAC Addr: 00:1E:67:33:95:3A

Ether Type: 0x0800，即Ipv4协议。

* 蓝色为ip header

共20个bytes。

version为4，表示为Ipv4协议。

IHL为5，表示header有20个bytes。

total length: 0x38,即56个bytes。因为header有20个字节，所以data有36个字节。

protocol：0x06，即TCP传输协议。

src ip addr: 0xAC.0x15.0x1D.0xF3,即172.21.29.243。

dst ip addr: 0xAC.0x15.0x22.0xE9,即172.21.34.233。

没有option数据。

* 最后面是tcp header + tcp data.

由前面ip header可知，ip header后的是tcp协议。

tcp header + tcp data总共有36个bytes。

src port: 0x9B9B,即39835

dst port: 0x1A0A,即6666

sequence num: 0xB048C654

acknowledged num : 0xCD7131BF

ACK flag: 1

PSH flag: 1

SYN flag: 0

FIN flag: 0

data offset: 0x8，即为8\*4 = 32个bytes。即tcp header为32个bytes。tcp data为36-32 = 4个bytes。

tcp data为0x6368650A，即字符串”che\n”。