延迟确认那些事

TCP 回包的磨叽姐

再聊 ACK

是不是每个数据包都对应一个 ACK 包?

是不是接收端收到数据以后必须立刻马上回复确认包?

回复一个空的 ACK 实在是太浪费了

减少网络流量的一个重要举措

接收端这个时候恰好有数据要回复客户端

ACK 搭上顺风车一块发送

期间又有客户端的数据传过来

把多次 ACK 合并成一个立刻发送出去

一段时间没有顺风车

不能让接收端等太久，一个空包也得发

回复一个空的 ACK 实在是太浪费了

减少网络流量的一个重要举措

接收端这个时候恰好有数据要回复客户端

ACK 搭上顺风车一块发送

期间又有客户端的数据传过来

把多次 ACK 合并成一个立刻发送出去

一段时间没有顺风车

不能让接收端等太久，一个空包也得发

延迟确认（delayed ack）

什么时候需要回复 ACK?

```
static void __tcp_ack_snd_check(struct sock *sk, int ofo_possible)
{
  struct tcp_sock *tp = tcp_sk(sk);

      /* More than one full frame received... */
  if (((tp->rcv_nxt - tp->rcv_wup) > tp->ack.rcv_mss
       /* ... and right edge of window advances far enough.
        * (tcp_recvmsg() will send ACK otherwise). Or...
        */
       && __tcp_select_window(sk) >= tp->rcv_wnd) ||
      /* We ACK each frame or... */
      tcp_in_quickack_mode(tp) ||
      /* We have out of order data. */
      (ofo_possible &&
       skb_peek(&tp->out_of_order_queue))) {
    /* Then ack it now */
    tcp_send_ack(sk);
  } else {
    /* Else, send delayed ack. */
    tcp_send_delayed_ack(sk);
  }
}
```

1、如果接收到了大于一个frame 的报文，且需要调整窗口大小

2、处于 quickack 模式（tcp_in_quickack_mode）

3、收到乱序包（We have out of order data.）

什么时候需要回复 ACK?

```
/* Send ACKs quickly, if "quick" count is not exhausted
 * and the session is not interactive.
 */

static __inline__ int tcp_in_quickack_mode(struct tcp_sock *tp)
{
  return (tp->ack.quick && !tp->ack.pingpong);
}

/* Delayed ACK control data */
struct {
  __u8  pending;  /* ACK is pending */
  __u8  quick;    /* Scheduled number of quick acks */
  __u8  pingpong; /* The session is interactive   */
  __u8  blocked;  /* Delayed ACK was blocked by socket lock*/
  __u32 ato;    /* Predicted tick of soft clock   */
  unsigned long timeout;  /* Currently scheduled timeout    */
  __u32 lrcvtime; /* timestamp of last received data packet*/
  __u16 last_seg_size;  /* Size of last incoming segment  */
  __u16 rcv_mss;  /* MSS used for delayed ACK decisions */
} ack;
```

其中pingpong 就是判断交互连接的，只有处于非交互 TCP 连接才有可能即进入 quickack 模式。

什么是交互式和 pingpong 呢?

顾名思义，其实有来有回的双向数据传输就叫 pingpong，对于通信的某一端来说，`R-W-R-W-R-W...`（R 表示读，W 表示写）

演示

```
public class DelayAckServer {
    private static final int PORT = 8888;

    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket();
        serverSocket.bind(new InetSocketAddress(PORT));
        System.out.println("Server startup at " + PORT);
        while (true) {
            Socket socket = serverSocket.accept();
            InputStream inputStream = socket.getInputStream();
            OutputStream outputStream = socket.getOutputStream();
            int i = 1;
            while (true) {
                BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
                String line = reader.readLine();
                if (line == null) break;
                System.out.println((i++) + " : " + line);
                outputStream.write((line + "\n").getBytes());
            }
        }
    }
}
```

演示

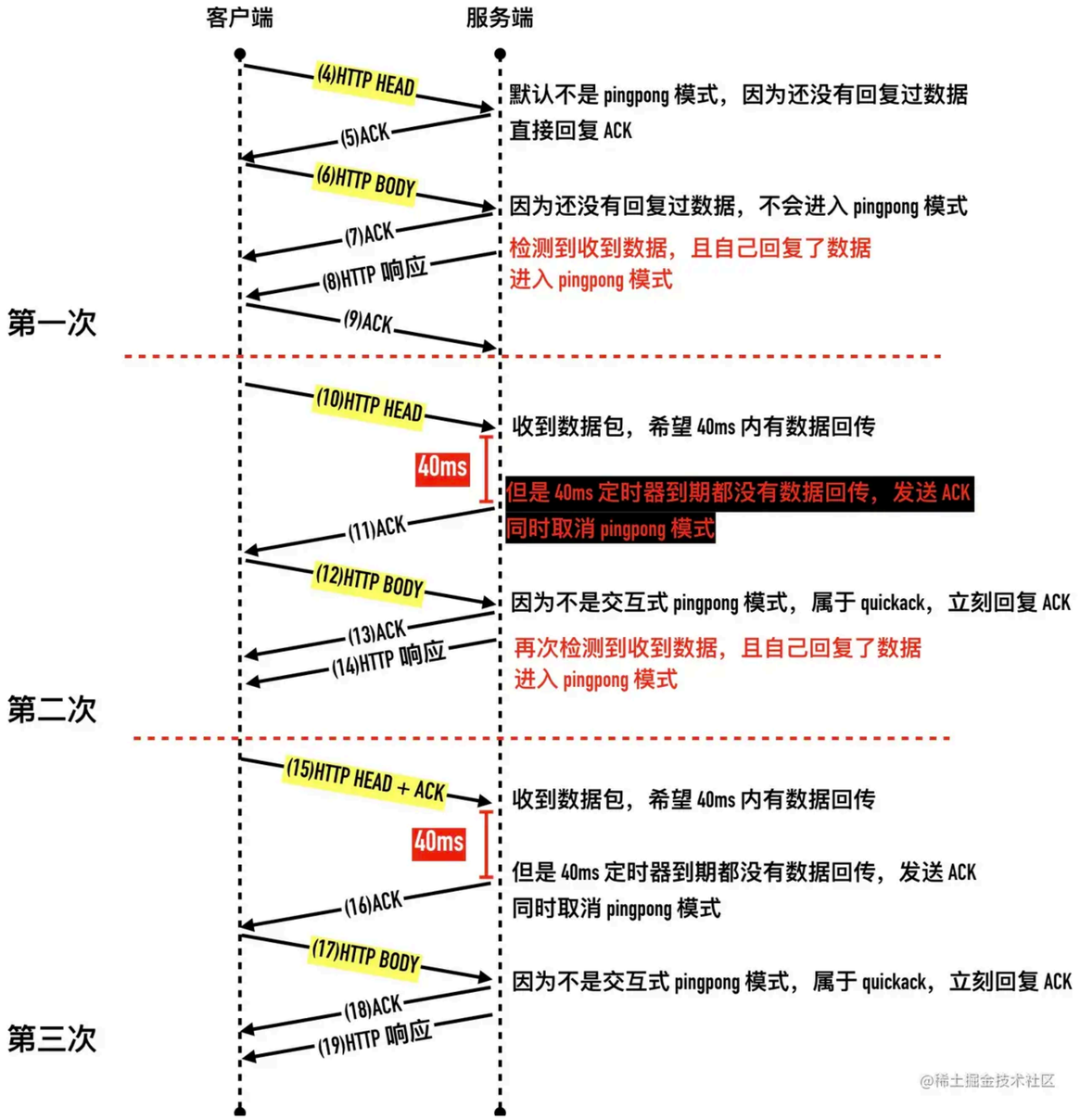
```
public class DelayAckClient {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket();
        socket.connect(new InetSocketAddress("server_ip", 8888));
        InputStream inputStream = socket.getInputStream();
        OutputStream outputStream = socket.getOutputStream();
        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
        String head = "hello, ";
        String body = "world\n";

        for (int i = 0; i < 10; i++) {
            long start = System.currentTimeMillis();
            outputStream.write(("#" + i + " " + head).getBytes()); // write
            outputStream.write((body).getBytes()); // write
            String line = reader.readLine(); // read
            System.out.println("RTT: " + (System.currentTimeMillis() - start) + ": " + line);
        }
        inputStream.close();
        outputStream.close();
        socket.close();
    }
}
```

包分析

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=268816418 TSecr=0
2	0.000167	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=3401616
3	0.000185	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=268816419 TSecr=340161632
4	0.001704	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=10 TSval=268816420 TSecr=340161632
5	0.001840	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [ACK] Seq=1 Ack=11 Win=29056 Len=0 TSval=340161634 TSecr=268816420
6	0.001851	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=11 Ack=1 Win=29312 Len=6 TSval=268816420 TSecr=340161634
7	0.001881	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [ACK] Seq=1 Ack=17 Win=29056 Len=0 TSval=340161634 TSecr=268816420
8	0.002282	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [PSH, ACK] Seq=1 Ack=17 Win=29056 Len=16 TSval=340161634 TSecr=268816420
9	0.002327	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [ACK] Seq=17 Ack=17 Win=29312 Len=0 TSval=268816421 TSecr=340161634
10	0.002798	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=17 Ack=17 Win=29312 Len=10 TSval=268816421 TSecr=340161634
11	0.046394	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [ACK] Seq=17 Ack=27 Win=29056 Len=0 TSval=340161678 TSecr=268816421
12	0.046421	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=27 Ack=17 Win=29312 Len=6 TSval=268816465 TSecr=340161678
13	0.046526	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [ACK] Seq=17 Ack=33 Win=29056 Len=0 TSval=340161679 TSecr=268816465
14	0.046690	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [PSH, ACK] Seq=17 Ack=33 Win=29056 Len=16 TSval=340161679 TSecr=268816465
15	0.046885	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=33 Ack=33 Win=29312 Len=10 TSval=268816465 TSecr=340161679
16	0.090509	10.211.	211	TCP	8888 → 58936 [ACK] Seq=33 Ack=43 Win=29056 Len=0 TSval=340161722 TSecr=268816465
17	0.090527	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=43 Ack=33 Win=29312 Len=6 TSval=268816509 TSecr=340161722
18	0.090641	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [ACK] Seq=33 Ack=49 Win=29056 Len=0 TSval=340161723 TSecr=268816509
19	0.090827	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [PSH, ACK] Seq=33 Ack=49 Win=29056 Len=16 TSval=340161723 TSecr=268816509
20	0.092213	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=49 Ack=49 Win=29312 Len=10 TSval=268816511 TSecr=340161723
21	0.135543	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [ACK] Seq=49 Ack=59 Win=29056 Len=0 TSval=340161768 TSecr=268816511
22	0.135569	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=59 Ack=49 Win=29312 Len=6 TSval=268816554 TSecr=340161768
23	0.135695	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [ACK] Seq=49 Ack=65 Win=29056 Len=0 TSval=340161768 TSecr=268816554
24	0.135877	10.211.55.5	10.211.55.10	TCP	8888 → 58936 [PSH, ACK] Seq=49 Ack=65 Win=29056 Len=16 TSval=340161768 TSecr=268816554
25	0.136093	10.211.55.10	10.211.55.5	TCP	58936 → 8888 [PSH, ACK] Seq=65 Ack=65 Win=29312 Len=10 TSval=268816554 TSecr=340161768

延迟确认，44ms

交互流程



packetdrill 模拟

```
--tolerance_usecs=100000
0.000 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
0.000 setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
0.000 bind(3, ..., ...) = 0
0.000 listen(3, 1) = 0

0.000 < S 0:0(0) win 32792 <mss 1000, sackOK, nop, nop, nop, wscale 7>
0.000 > S. 0:0(0) ack 1 <...>

0.000 < . 1:1(0) ack 1 win 257

0.000 accept(3, ..., ...) = 4

+ 0 setsockopt(4, SOL_TCP, TCP_NODELAY, [1], 4) = 0

// 模拟往服务端写入 HTTP 头部: POST / HTTP/1.1
+0 < P. 1:11(10) ack 1 win 257

// 模拟往服务端写入 HTTP 请求 body: {"id": 1314}
+0 < P. 11:26(15) ack 1 win 257

// 往 fd 为4 的 模拟服务器返回 HTTP response {}
+ 0 write(4, ..., 100) = 100


// 第二次模拟往服务端写入 HTTP 头部: POST / HTTP/1.1
+0 < P. 26:36(10) ack 101 win 257

// 抓包看服务器返回

+0 `sleep 1000000`
```

packetdrill 模拟

```
--tolerance_usecs=100000
0.000 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
0.000 setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
0.000 bind(3, ..., ...) = 0
0.000 listen(3, 1) = 0

0.000 < S 0:0(0) win 32792 <mss 1000, sackOK, nop, nop, nop, wscale 7>
0.000 > S. 0:0(0) ack 1 <...>

0.000 < . 1:1(0) ack 1 win 257

0.000 accept(3, ..., ...) = 4

+ 0 setsockopt(4, SOL_TCP, TCP_NODELAY, [1], 4)

// 模拟往服务端写入 HTTP 头部: POST / HTTP/1.1
+0 < P. 1:11(10) ack 1 win 257

// 模拟往服务端写入 HTTP 请求 body: {"id": 1314}
+0 < P. 11:26(15) ack 1 win 257

// 往 fd 为4 的 模拟服务器返回 HTTP response {}
+ 0 write(4, ..., 100) = 100


// 第二次模拟往服务端写入 HTTP 头部: POST / HTTP/1.1
+0 < P. 26:36(10) ack 101 win 257

// 抓包看服务器返回

+0 `sleep 1000000`
```

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.0.2.1	192.168.139.46	TCP	49604 → 8080 [SYN] Seq=0 Win=32792 Len=0 MSS=1000 SACK_PERM=
2	0.000065	192.168.139.46	192.0.2.1	TCP	8080 → 49604 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=146
3	0.000189	192.0.2.1	192.168.139.46	TCP	49604 → 8080 [ACK] Seq=1 Ack=1 Win=32896 Len=0
4	0.000300	192.0.2.1	192.168.139.46	TCP	49604 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=32896 Len=10
5	0.000316	192.168.139.46	192.0.2.1	TCP	8080 → 49604 [ACK] Seq=1 Ack=11 Win=29312 Len=0
6	0.000339	192.0.2.1	192.168.139.46	TCP	49604 → 8080 [PSH, ACK] Seq=11 Ack=1 Win=32896 Len=15
7	0.000352	192.168.139.46	192.0.2.1	TCP	8080 → 49604 [ACK] Seq=1 Ack=26 Win=29312 Len=0
8	0.000375	192.168.139.46	192.0.2.1	TCP	8080 → 49604 [PSH, ACK] Seq=1 Ack=26 Win=29312 Len=100
9	0.000389	192.0.2.1	192.168.139.46	TCP	49604 → 8080 [PSH, ACK] Seq=26 Ack=101 Win=32896 Len=10
10	0.043645	192.168.139.46	192.0.2.1	TCP	8080 → 49604 [ACK] Seq=101 Ack=36 Win=29312 Len=0

当 Nagle 算法遇到延迟确认

Nagle 算法和延迟确认本身并没有什么问题，但一起使用就会出现很严重的性能问题了。Nagle 攒着包一次发一个，延迟确认收到包不马上回。

当 Nagle 算法遇到延迟确认

socket.setTcpNoDelay(true); // 禁用 Nagle 算法

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=277480985 TSecr=0 WS=128
2	0.000540	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=348828048 TSecr=277480985
3	0.000574	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=277480986 TSecr=348828048
4	0.001728	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=10 TSval=277480987 TSecr=348828048
5	0.001778	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=11 Ack=1 Win=29312 Len=6 TSval=277480987 TSecr=348828048
6	0.001920	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [ACK] Seq=1 Ack=11 Win=29056 Len=0 TSval=348828050 TSecr=277480987
7	0.001932	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [ACK] Seq=1 Ack=17 Win=29056 Len=0 TSval=348828050 TSecr=277480987
8	0.002214	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [PSH, ACK] Seq=1 Ack=17 Win=29056 Len=16 TSval=348828050 TSecr=277480987
9	0.002256	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [ACK] Seq=17 Ack=17 Win=29312 Len=0 TSval=277480987 TSecr=348828050
10	0.002733	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=17 Ack=17 Win=29312 Len=10 TSval=277480988 TSecr=348828050
11	0.002769	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=27 Ack=17 Win=29312 Len=6 TSval=277480988 TSecr=348828050
12	0.002885	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [ACK] Seq=17 Ack=33 Win=29056 Len=0 TSval=348828051 TSecr=277480988
13	0.002972	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [PSH, ACK] Seq=17 Ack=33 Win=29056 Len=16 TSval=348828051 TSecr=277480988
14	0.003135	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=33 Ack=33 Win=29312 Len=10 TSval=277480988 TSecr=348828051
15	0.003155	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=43 Ack=33 Win=29312 Len=6 TSval=277480988 TSecr=348828051
16	0.003298	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [ACK] Seq=33 Ack=49 Win=29056 Len=0 TSval=348828051 TSecr=277480988
17	0.003478	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [PSH, ACK] Seq=33 Ack=49 Win=29056 Len=16 TSval=348828051 TSecr=277480988
18	0.003644	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=49 Ack=49 Win=29312 Len=10 TSval=277480989 TSecr=348828051
19	0.003698	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=59 Ack=49 Win=29312 Len=6 TSval=277480989 TSecr=348828051
20	0.004013	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [ACK] Seq=49 Ack=65 Win=29056 Len=0 TSval=348828052 TSecr=277480989
21	0.004151	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [PSH, ACK] Seq=49 Ack=65 Win=29056 Len=16 TSval=348828052 TSecr=277480989
22	0.004408	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=65 Ack=65 Win=29312 Len=10 TSval=277480990 TSecr=348828052
23	0.004431	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=75 Ack=65 Win=29312 Len=6 TSval=277480990 TSecr=348828052
24	0.004569	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [ACK] Seq=65 Ack=81 Win=29056 Len=0 TSval=348828052 TSecr=277480990
25	0.004766	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [PSH, ACK] Seq=65 Ack=81 Win=29056 Len=16 TSval=348828052 TSecr=277480990
26	0.004913	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=81 Ack=81 Win=29312 Len=10 TSval=277480990 TSecr=348828052
27	0.004944	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=91 Ack=81 Win=29312 Len=6 TSval=277480990 TSecr=348828052
28	0.005140	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [ACK] Seq=81 Ack=97 Win=29056 Len=0 TSval=348828053 TSecr=277480990
29	0.005256	10.211.55.5	10.211.55.10	TCP	8888 → 58942 [PSH, ACK] Seq=81 Ack=97 Win=29056 Len=16 TSval=348828053 TSecr=277480990
30	0.005462	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=97 Ack=97 Win=29312 Len=10 TSval=277480991 TSecr=348828053
31	0.005483	10.211.55.10	10.211.55.5	TCP	58942 → 8888 [PSH, ACK] Seq=107 Ack=97 Win=29312 Len=6 TSval=277480991 TSecr=348828053

黑色背景部分的是客户端发送给服务端的请求包