《计算机网络协议开发》实验报告

第2次实验

网络协议源代码阅读实验

姓名：陈越琦

学号：121160005

计算机系13级

邮箱：Yueqichen.0x0@gmail.com

时间: 03/06/2017

实验目的

通过阅读实现网络协议的源代码，了解网络协议开发中常用的协议结构定义方法，有限状态机实现方法，网络协议实现的一般流程和常用技巧。

注释备注

因为源代码的命名风格非常得好，所以有些常量如MAX_TCP_CONNECTIONS，全局变量如gLocalPort，全局数据结构及其成员变量如struct MyTcpSeg，完全没有注释的必要，所以在lab02.c文件中，只对部分变量如gTCB数组进行了注释。

基于类似的原因，部分函数的参数如srcAddr，以及绝大部分函数的返回值如int tcp_socket（返回为-1表示出现异常，返回为0一切顺利）也没有注释。但是对于extern int waitIpPacket()这类返回值有重要作用的函数，都逐一注释。此外，函数的功能注释与重要语句的注释不做明显的区分。

还有一点需要指出的是，我根据[1]中倡导的代码风格对源代码进行了部分改写，从而方便个人阅读方便（源代码原先的风格也是值得学习的）。

实验内容

1. 源代码使用了停等协议。在图1中，客户端每发送一次分组（line 582），都要等待10秒（line 586）接收来自服务器端的分组，而不是使用GBN或者选择性重传协议中，一次发送多个分组而不需要等待确认的滑动窗口协议。停等协议的效率固然比较低，但是实现起来要比GBN和选择性重传简单。

2. 源代码实现了校验和与为每个字节分配序列号以及提供确认的功能，但是没有实现重传机制。图2和图3分别是发送分组之前，计算校验和与接收分组之后验证校验和的函数。图4的TCP结构体中的成员变量seq (line 81)相当于一个计数器，为每个发送出去的字节都分配序列号。在发送分组之前，都会对TCP首部的序列号字段进行赋值（图5 line 199）。在tcp_establishe函数中（图6）为每个成功接收的报文都发送一个报文确认（line 302），同时发送确认号（line 301）。

3. 源代码中除了没有实现重传机制等功能以外，我认为还有以下的问题：

   - tcp四次关闭的过程没有完整走完：在tcp_finwait_2函数中，在接收到服务器发出的FIN段后，客户端并没有及时进入TIME_WAIT状态，而是直接进入了CLOSED状态（当然，在没有实现重传机制的情况下，CLOSED可以认为就是TIME_WAIT）。

   - 源代码缺少足够的错误应对机制。导致整个tcp链接过程会非常得脆弱。比如说在函数tcp_close中，第二次tcp_input函数调用如果受到replay 攻击，程序除了把分组

丢弃并没有其他的行为，再加上没有重传机制，导致客户端没有发送ACK分组就不正常终止TCP连接，而服务器一直在等待客户端的ACK分组。

```
573 // send to server after establishing tcp connection
574 int tcp_send(int sockfd, const unsigned char* pData,
575             unsigned short datalen, int flags) {
576   char buffer[2048];
577   int len;
578
579   if (gTCB[sockfd].current_state != ESTABLISHED)
580     return -1;
581
582   tcp_output((char *)pData, datalen, flags,
583              gTCB[sockfd].local_port, gTCB[sockfd].remote_port,
584              gTCB[sockfd].local_ip, gTCB[sockfd].remote_ip);
585
586   len = waitIpPacket(buffer, 10);
587
588   // header of tcp segment is at least 20 Bytes
589   if( len < 20 )
590     return -1;
591
592   tcp_input(buffer, len,
593             htonl(gTCB[sockfd].remote_ip),
594             htonl(gTCB[sockfd].local_ip));
595   return 0;
596 }
```

图 1: TCP send

```
125 // called by tcp_kick
126 // calculate checksum for tcp segment
127 unsigned short tcp_calc_checksum(struct MyTCB* pTcb,
128                                  struct MyTcpSeg* pTcpSeg) {
129   int i = 0;
130   int len = 0;
131   unsigned int sum = 0;
132   unsigned short* p = (unsigned short*)pTcpSeg;
133
134   if( pTcb == NULL || pTcpSeg == NULL )
135     return 0;
136
137   for( i=0; i<10; i++)
138     sum += p[i];
139
140   sum = sum - p[8] - p[6] + ntohs(p[6]);
141
142   // if segment has data
143   if ((len = pTcpSeg->len) > 20) {
144     if (len%2 == 1) {
145       pTcpSeg->data[len - 20] = 0;
146       len++;
147     }
148
149     for (i = 10; i < len/2; i++)
150       sum += ntohs(p[i]);
151   }
152
153   //should add ip for checksum
154   sum = sum + (unsigned short)(pTcb->local_ip>>16)
155             + (unsigned short)(pTcb->local_ip&0xffff)
156             + (unsigned short)(pTcb->remote_ip>>16)
157             + (unsigned short)(pTcb->remote_ip&0xffff);
158   sum = sum + 6 + pTcpSeg->len;
159   sum = ( sum & 0xFFFF ) + ( sum >> 16 );
160   sum = ( sum & 0xFFFF ) + ( sum >> 16 );
161   return (unsigned short)(~sum);
162 }
163
```

图 2: calculate checksum

```
375 // called by tcp_input
376 // check the checkcum in tcp segment
377 int tcp_check(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg) {
378   int i = 0;
379   int len = 0;
380   unsigned int sum = 0;
381   unsigned short* p = (unsigned short*)pTcpSeg;
382   unsigned short *pIp;
383   unsigned int myip1 = pTcb->local_ip;
384   unsigned int myip2 = pTcb->remote_ip;
385   if (pTcb == NULL || pTcpSeg == NULL)
386     return -1;
387   for( i=0; i<10; i++)
388     sum = sum + p[i];
389   // why translate again, see convert_tcp_hdr_ntoh()
390   sum = sum - p[6] + ntohs(p[6]);
391   // if this segment has data
392   if ((len = pTcpSeg->len) > 20) {
393     if (len % 2 == 1) {
394       pTcpSeg->data[len - 20] = 0;
395       len++;
396     }
397
398     for (i = 10; i < len/2; i++)
399       sum += ntohs(p[i]);
400   }
401   // should add ip for tcp checksum
402   sum = sum + (unsigned short)(myip1>>16)
403             + (unsigned short)(myip1&0xffff)
404             + (unsigned short)(myip2>>16)
405             + (unsigned short)(myip2&0xffff);
406   sum = sum + 6 + pTcpSeg->len;
407
408   sum = ( sum & 0xFFFF ) + ( sum >> 16 );
409   sum = ( sum & 0xFFFF ) + ( sum >> 16 );
410
411   if ((unsigned short)(~sum) != 0) {
412     // TODO:
413     printf("check sum error!\n");
```

图 3: check checksum

```
74 struct MyTCB
75 {
76     STATE current_state;
77     unsigned int local_ip;
78     unsigned short local_port;
79     unsigned int remote_ip;
80     unsigned short remote_port;
81     unsigned int seq;
82     unsigned int ack;
83     unsigned char flags;
84     int iotype;
85     int is_used;
86     int data_ack;
87     unsigned char data[2048];
88     unsigned short data_len;
89 };
```

图 4: TCP struct

```c
// constuct segment for tcp before sent out
int tcp_construct_segment(struct MyTcpSeg* pTcpSeg, struct MyTCB* pTcb,
                          unsigned short datalen, unsigned char* pData) {
  pTcpSeg->src_port = pTcb->local_port;
  pTcpSeg->dst_port = pTcb->remote_port;
  pTcpSeg->seq_num = pTcb->seq;
  pTcpSeg->ack_num = pTcb->ack;
  pTcpSeg->hdr_len = (unsigned char)(0x50);
  pTcpSeg->flags = pTcb->flags;
  pTcpSeg->window_size = 1024;
  pTcpSeg->urg_ptr = 0;

  if( datalen > 0 && pData != NULL )
    memcpy(pTcpSeg->data, pData, datalen);

  pTcpSeg->len = 20 + datalen;
  return 0;
}
```

图 5: construct tcp segment

```c
// handle segment during the established
int tcp_established(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg) {
  struct MyTcpSeg my_seg;

  if( pTcb == NULL || pTcpSeg == NULL )
    return -1;

  if (pTcb->iotype == INPUT) { // receive
    if (pTcpSeg->seq_num != pTcb->ack) { // sequence number dismatch ack number
      tcp_DiscardPkt((char*)pTcpSeg, TCP_TEST_SEQNO_ERROR);
      //to do: discard packet
      return -1;
    }

    // confirmation of a successfully received segment
    if ((pTcpSeg->flags & 0x3f) == 0x10) {
      memcpy(pTcb->data, pTcpSeg->data, pTcpSeg->len - 20);
      pTcb->data_len = pTcpSeg->len - 20;
      if (pTcb->data_len == 0)  { // no data

      } else { // prepare for response, subpath of tcp_recv()
        pTcb->ack += pTcb->data_len;
        pTcb->flags = 0x10; // confirm receiving a tcp segment
        tcp_construct_segment(&my_seg, pTcb, 0, NULL);
        tcp_kick(pTcb, &my_seg);
      }
    }
  } else {
    if ((pTcpSeg->flags & 0x0F) == 0x01)  // FIN = 1, subpath of tcp_close()
      pTcb->current_state = FIN_WAIT1;
    // normal send, subpath of tcp_send()
    // tcp segment has already been constructed through call stack
    tcp_kick(pTcb, pTcpSeg);
  }
  return 0;
}
```

图 6: tcp established

实验启示

在本次实验中，通过阅读tcp客户端源码，我加深了对TCP建立和拆除连接过程的理解。同时我也学习了如何优雅得实现有限状态机，并对网络协议开发的一般流程和协议结构定义方法有了更多的掌握，为接下来的实验奠定了基础。

实验时间统计

- 理解源码并添加注释：3.5 hour
- 回答思考问题及报告撰写：1.5 hour

参考文献

[1] http://google-styleguide.googlecode.com/