网络传输机制实验2报告

张翔雨 2018K8009929035

一、实验题目：网络传输机制实验2

二、实验内容

内容1：

- 运行给定网络拓扑(tcp_topo.py)
- 在节点h1上执行TCP程序
  - 执行脚本(disable_offloading.sh , disable_tcp_rst.sh)
  - 在h1上运行TCP协议栈的服务器模式 (./tcp_stack server 10001)
- 在节点h2上执行TCP程序
  - 执行脚本(disable_offloading.sh, disable_tcp_rst.sh)
  - 在h2上运行TCP协议栈的客户端模式，连接h1并正确收发数据 (./tcp_stack client 10.0.0.1 10001)
    - client向server发送数据，server将数据echo给client
- 使用tcp_stack.py替换其中任意一端，对端都能正确收发数据

内容2：

- 修改tcp_apps.c(以及tcp_stack.py)，使之能够收发文件
- 执行create_randfile.sh，生成待传输数据文件client-input.dat
- 运行给定网络拓扑(tcp_topo.py)
- 在节点h1上执行TCP程序
  - 执行脚本(disable_offloading.sh , disable_tcp_rst.sh)
  - 在h1上运行TCP协议栈的服务器模式 (./tcp_stack server 10001)
- 在节点h2上执行TCP程序

    - 执行脚本(disable_offloading.sh, disable_tcp_rst.sh)
    - 在h2上运行TCP协议栈的客户端模式 (./tcp_stack client 10.0.0.1 10001)
        - Client发送文件client-input.dat给server，server将收到的数据存储到文件server-output.dat
- 使用md5sum比较两个文件是否完全相同
- 使用tcp_stack.py替换其中任意一端，对端都能正确收发数据

三、实验过程

1.添加状态处理逻辑

在`established`状态的处理逻辑中添加处理非`FIN`包的处理逻辑，即PSH包和ACK包，若收到数据长度是0则为ACK包，反之为数据包。基本处理逻辑是当收到数据包时，先判断`ring_buffer`是否已满，若已经满了则进入睡眠，等待读缓冲区函数唤醒，然后获取锁并写入缓冲区，之后唤醒因缓冲区为空而睡眠的进程并发送ACK包。

在处理`FIN`包时，标志此次传输已经结束，需要退出所有等待的进程，因此调用`wait_exit`函数设置dead位为1，并向所有等待的进程广播。

```
        case TCP_ESTABLISHED:
            if(cb->flags & TCP_FIN)
            {
                tsk->rcv_nxt = cb->seq+1;
                wait_exit(tsk->wait_recv);
                tcp_set_state(tsk, TCP_CLOSE_WAIT);
                tcp_send_control_packet(tsk, TCP_ACK);
            }
            else if(cb->flags == (TCP_PSH | TCP_ACK))
            {
                if(cb->pl_len == 0)
                {
                    tsk->snd_una = cb->ack;
                    tsk->rcv_nxt = cb->seq + 1;
                    tcp_update_window_safe(tsk, cb);
                }
                else
                {
                    while (ring_buffer_full(tsk->rcv_buf))
                        sleep_on(tsk->wait_recv);
                    pthread_mutex_lock(&tsk->rcv_buf->lock);
                    write_ring_buffer(tsk->rcv_buf, cb->payload, cb->pl_len);
                    pthread_mutex_unlock(&tsk->rcv_buf->lock);
```

```
                tsk->rcv_nxt = cb->seq + cb->pl_len;
                tsk->snd_una = cb->ack;
                wake_up(tsk->wait_recv);
                tcp_send_control_packet(tsk, TCP_ACK);
            }
        }
        break;
```

2.读写函数设计

（1）实现int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len)

主要功能是读取缓冲区数据，并根据读取的状态返回不同的值。当缓冲区为空时进入等待状态，等待收到数据包时唤醒，读过程结束后唤醒因缓冲区满而等待的进程。若被`wait_exit`的广播唤醒，则再次进行等待判断时，会因为dead位为1而等待失败，此时read函数返回0代表连接结束。

```
int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len)
{
    while( ring_buffer_empty(tsk->rcv_buf) )
    {
        if(sleep_on(tsk->wait_recv)<0)
            return 0;
    }
    int plen = read_ring_buffer(tsk->rcv_buf, buf, len);
    wake_up(tsk->wait_recv);
    return plen;
}
```

（2）实现int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len)

主要功能是根据buf内容发送对应的数据包，首先要判断代发送数据的大小，若超过一个包的大小，则需要分多个包发送。返回发送的长度。

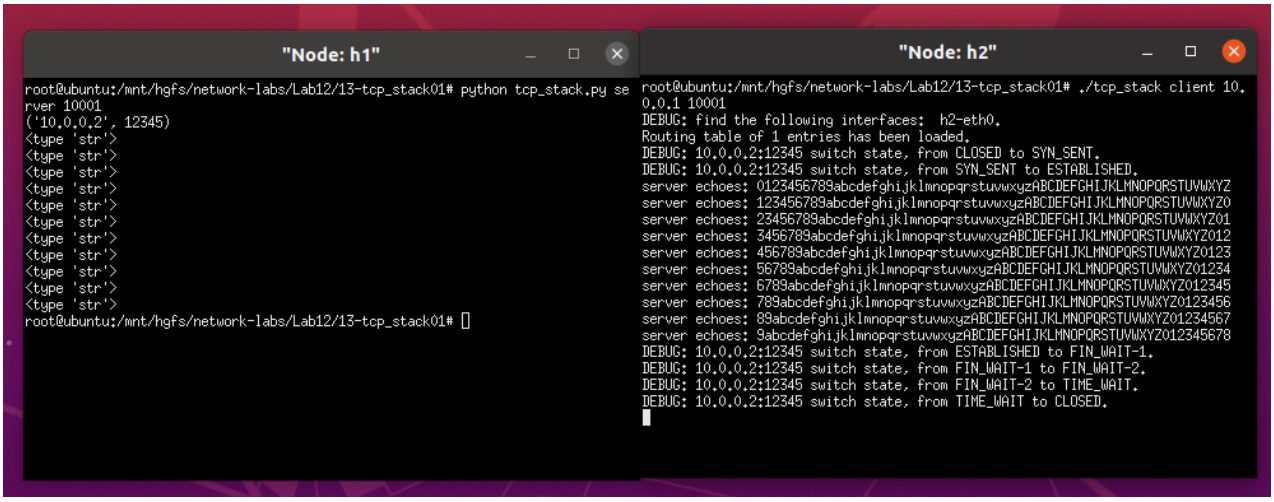
```
int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len)
{
    int seq = tsk->snd_nxt;
    int send_len = len;
    int data_len = 1514 - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE - TCP_BASE_HDR_SIZE;
    while (len > data_len)
    {
        char * packet = (char *)malloc(1514);
        memcpy(packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE,
               buf + (tsk->snd_nxt - seq), data_len);
```

```
        tcp_send_packet(tsk, packet, 1514);
        usleep(10000);
        len -= data_len;
    }
    int pkt_size = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE + len;
    char *packet = (char *)malloc(pkt_size);
    memset(packet, 0, ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE);
    memcpy(packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE,
                buf + (tsk->snd_nxt - seq), len);
    usleep(100000);
    tcp_send_packet(tsk, packet,pkt_size);
    return send_len;
}
```

四、实验结果

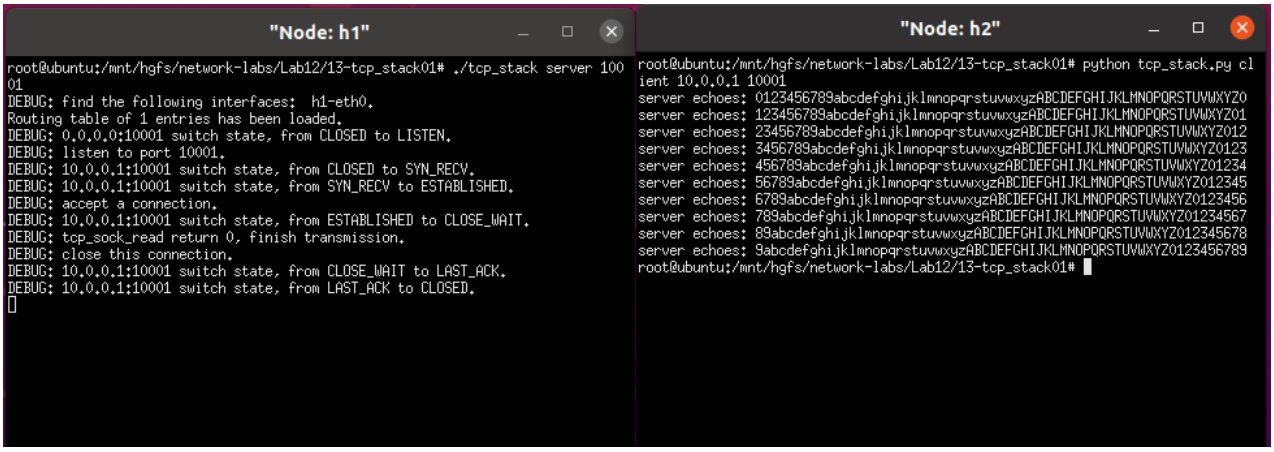
1.内容1：收发数据测试

（1）server脚本与自编写client交互



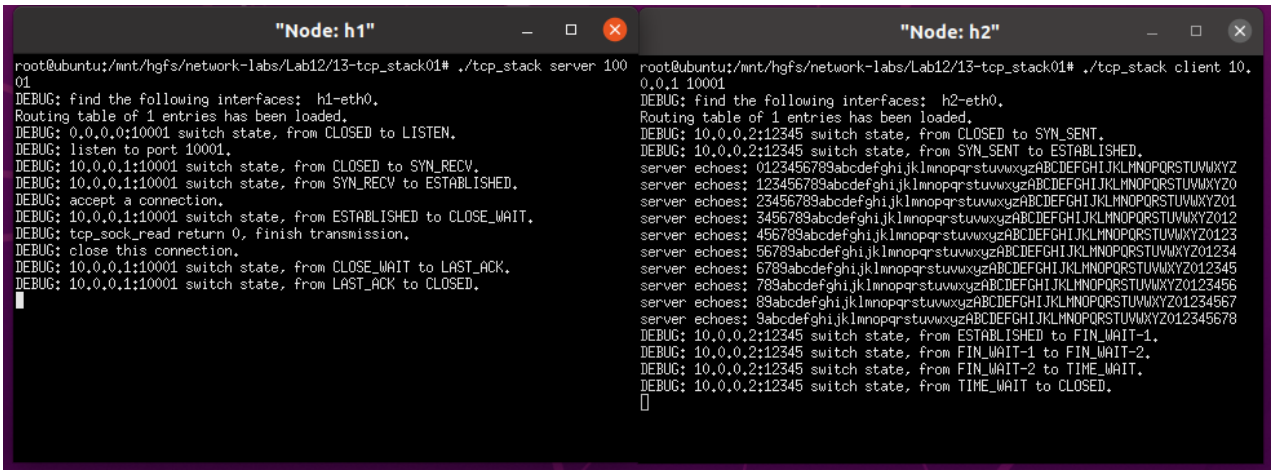
可以看到客户端收到了正确的服务器返还的数据，并按照正常的窗台转移退出，服务器收到包的次数也相同，测试正确。

（2）client脚本与自编写server交互



可以看到客户端脚本打印的数据证明服务器返回的数据正确，服务器按照正确的状态转移退出。

（3）自编写server与client交互‘

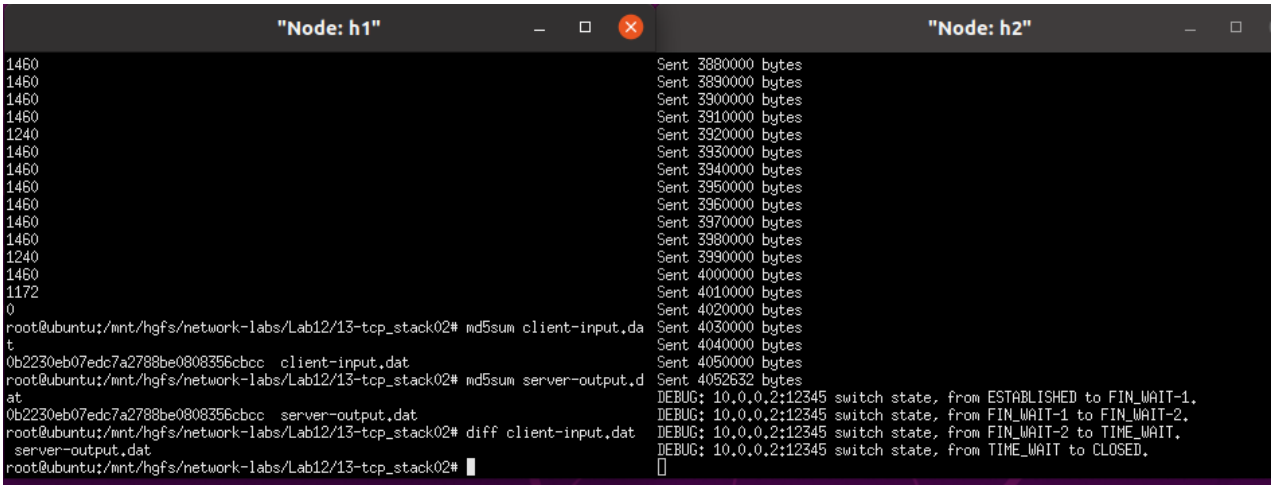


可以看到打印数据正确，两者都按照正确的状态转移退出。

2.内容2：收发文件测试

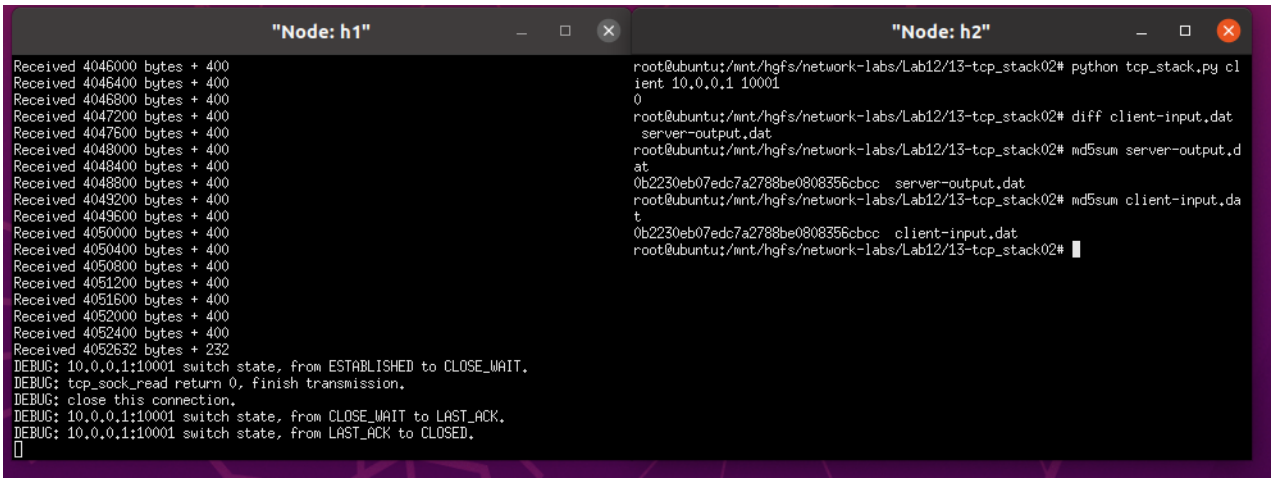
（1）server脚本与自编写client交互

一次发送10000字节，验证了大数据包发送的正确性。



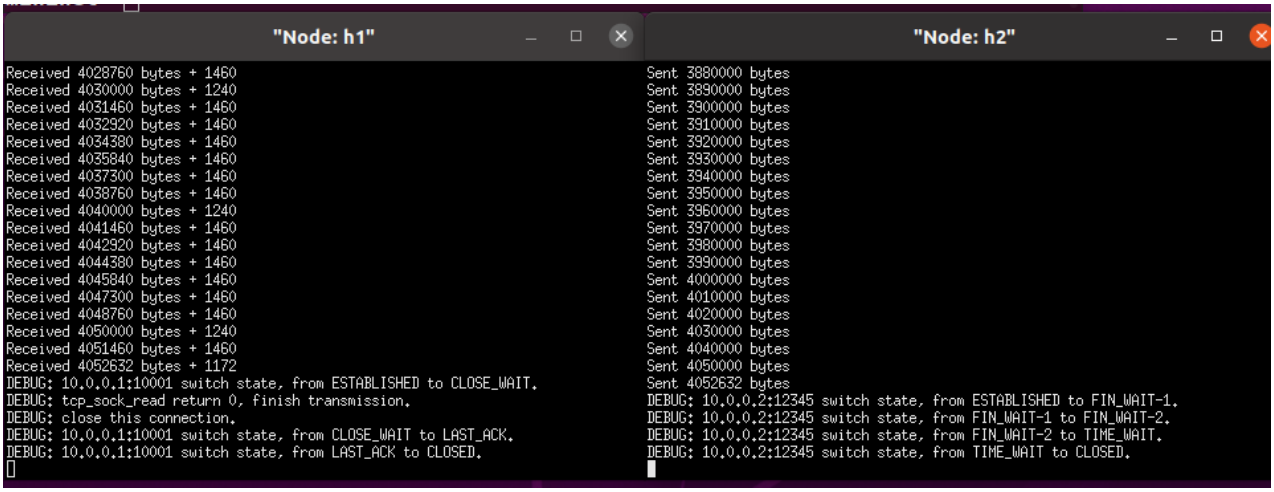
发送完成后，在服务器命令行（左）执行md5sum与diff命令，发现md5sum相同，diff未返回不同的地方，证明实验成功。

（2）client脚本与自编写server交互



发送完成后，在客户端命令后（右）执行md5sum和diff命令，发现md5sum相同，diff未返回不同的地方，证明实验成功。

（3）自编写server与client交互



先看到发送和接受字节数正确，且状态转移正确。退出后在当前目录下执行md5sum和diff命令，发现md5sum相同，diff未返回不同的地方，证明实验成功。

```
samuel@ubuntu:/mnt/hgfs/network-labs/Lab12/13-tcp_stack02$ md5sum server-output.
dat
0b2230eb07edc7a2788be0808356cbcc  server-output.dat
samuel@ubuntu:/mnt/hgfs/network-labs/Lab12/13-tcp_stack02$ md5sum client-input.d
at
0b2230eb07edc7a2788be0808356cbcc  client-input.dat
samuel@ubuntu:/mnt/hgfs/network-labs/Lab12/13-tcp_stack02$ diff client-input.dat
 server-output.dat
samuel@ubuntu:/mnt/hgfs/network-labs/Lab12/13-tcp_stack02$
```

六、实验总结

本次实验代码量较少，需要注意的地方却有一些。由于本次实现的读写函数不包含tcp重传与检测机制，导致在与脚本交互的过程中出现问题，通过wireshark抓包测试找到了重传发生的位置，发现对ACK包的序列号处理不当导致发生重传。