网络传输机制实验4报告

张翔雨 2018K8009929035

一、实验题目：网络传输机制实验4

二、实验内容

- 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_loss.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
- 记录h2中每次cwnd调整的时间和相应值，呈现到二维坐标图中

三、实验过程

实现TCP拥塞控制状态机

拥塞控制状态的转移主要由ACK包决定，在tsk结构体中添加 `ack_time` 域来记录收到重复ACK包的次数。

对于收到的包 `ack=snd_una` 的情况，即为重复ACK，若在OPEN状态下会累积计数，当到达3时会进入RECOVERY状态，此时启动快重传以及后续的快恢复机制：将ssthresh缩小为一半，同时记录恢复点，之后启动快恢复机制，将发送队列的第一个包进行重传，后续在接受到ACK包后进行重传。在RECOVERY状态下，没收到2个ACK就使cwnd-1，实现cwnd缩小为原来的一半的机制。

```
        if (cb->ack == tsk->snd_una)
        {
            tsk->ack_time++;
            switch (tsk->nrstate)
            {
                case OPEN:
                    if (tsk->ack_time > 2)
                    {
                        tsk->nrstate = RECOVER;
                        tsk->ssthresh = (tsk->cwnd + 1) / 2;
                        tsk->recovery_point = tsk->snd_nxt;
                        struct send_buffer *entry = list_entry((&tsk->send_buf)-
>next,typeof(*entry), list);
                        if (!list_empty(&tsk->send_buf))
                        {

                            char *temp = (char *) malloc(entry->len * sizeof(char));
                            memcpy(temp, entry->packet, entry->len);
                            ip_send_packet(temp, entry->len);
                        }
                    }
                    break;
                case RECOVER:
                    if (tsk->ack_time > 1)
                    {
                        tsk->ack_time -= 2;
                        tsk->cwnd -= 1;
                        if (tsk->cwnd < 1)
                            tsk->cwnd = 1;
                    }
                    break;
                default:
                    break;
            }
        }
```

对于有效的ACK包，新增加cwnd加1的操作，在OPEN状态下，每确认到一个数据包，就对cwnd进行一次增大。在增大操作时，引入新的计数器，如果cwnd小于ssthresh时，实现cwnd自增1，实现慢启动机制，反之则给新计数器加1如果新计数器超过cwnd，清除新计数器的值，给cwnd+1，实现拥塞避免

```
if(tsk->nrstate == OPEN)
{
    if (tsk->cwnd < tsk->ssthresh)
        tsk->cwnd ++;
else
{
    tsk->cnt ++;
    if (tsk->cnt >= tsk->cwnd)
```

```
    {
        tsk->cnt = 0;
        tsk->cwnd ++;
    }
}
}
```

在有效ACK包情况下，首先需要检查当前ack的值是否已回到了恢复点，若到达恢复点则可以回到OPEN状态。同时还需要进行快恢复机制的处理，需要将收到的ack包对应的数据包进行重传。

在重传计时器检查的过程中，如果发生了超时重传，则需要将状态置为LOSS，记录恢复点，并将ssthresh减半，将cwnd重置为1

```
if (tsk->nrstate != LOSS)
    tsk->recovery_point = tsk->snd_nxt;
tsk->nrstate = LOSS;
tsk->ssthresh = (tsk->cwnd + 1) / 2;
tsk->cwnd = 1;
tsk->snd_wnd = MSS;
pthread_mutex_lock(&tsk->file_lock);
cwnd_dump(tsk);
pthread_mutex_unlock(&tsk->file_lock);
```

2.信息记录

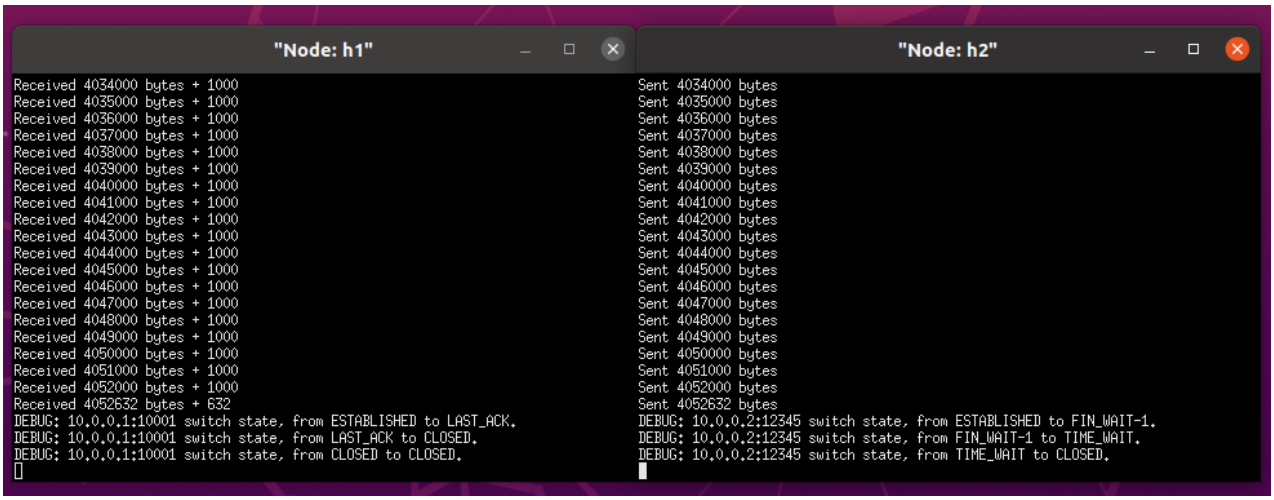
实现void cwnd_dump(struct tcp_sock *tsk)

在客户端进入自己的处理程序后，会打开对应名字的文件，在每次cwnd值变化时（超时重传和收到ACK包时），执行该文件，写入对应的时间和cwnd值到文件中。同时引入文件读写锁，每次读写文件时保证是互斥访问

```
void cwnd_dump(struct tcp_sock *tsk)
{
    struct timeval now;
    gettimeofday(&now, NULL);
    long long int duration = 1000000 * ( now.tv_sec - start.tv_sec ) + now.tv_usec -
start.tv_usec;
    fprintf(record, "%lld\t%d\n", duration, tsk->cwnd);
}
```

四、实验结果

收发文件测试

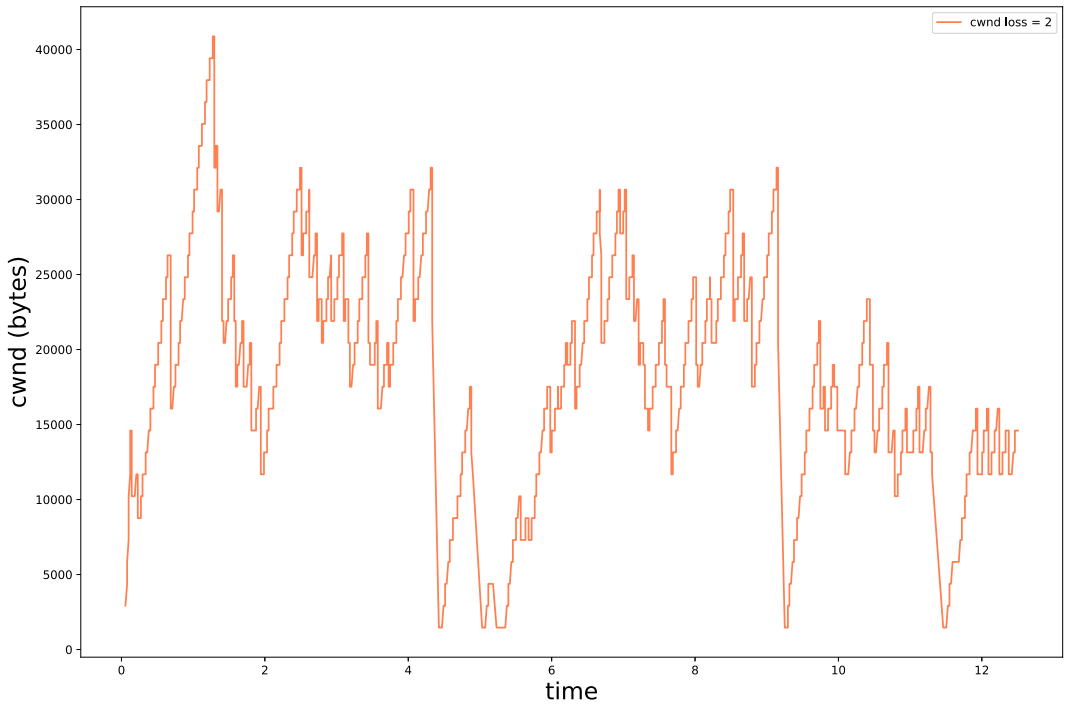


先看到发送和接受字节数正确，且状态转移正确。退出后在当前目录下执行md5sum和diff命令，发现md5sum相同，diff未返回不同的地方，证明实验成功。

```
samuel@ubuntu:/mnt/hgfs/network-labs/Lab14/16-tcp_stack$ md5sum *.dat
35ca4144c79796a3950ebe1053ba1743  client-input.dat
35ca4144c79796a3950ebe1053ba1743  server-output.dat
samuel@ubuntu:/mnt/hgfs/network-labs/Lab14/16-tcp_stack$ diff *.dat
samuel@ubuntu:/mnt/hgfs/network-labs/Lab14/16-tcp_stack$
```

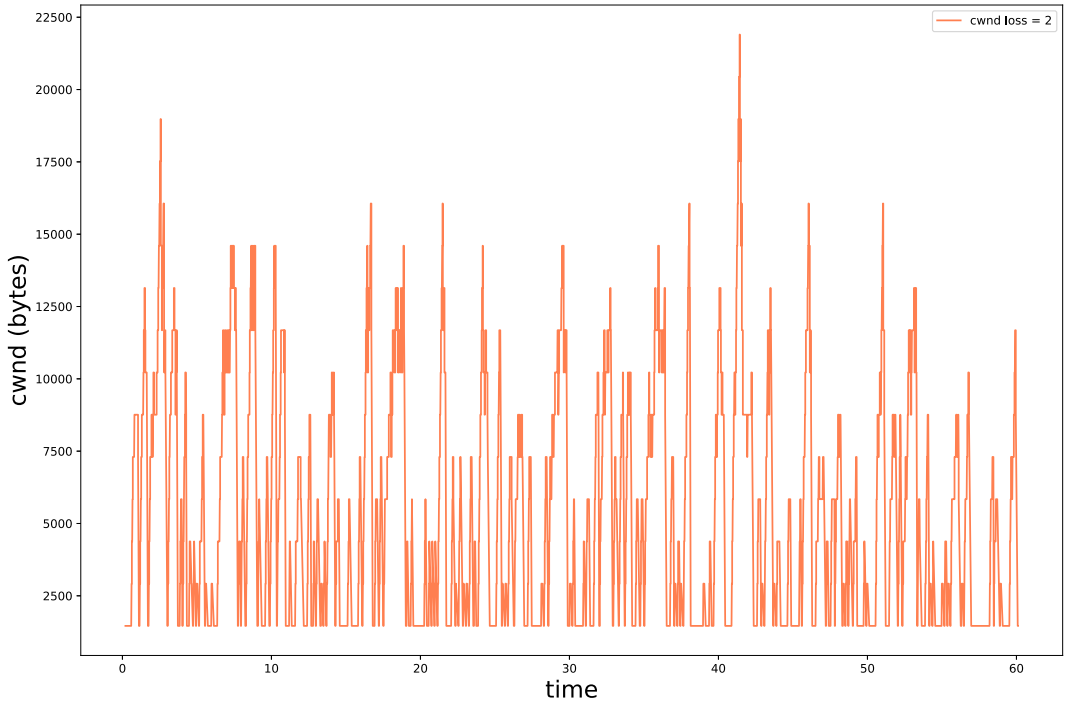
cwnd变化绘图

首先在loss=2的环境下进行实验，得到如下图所示的变化曲线。



与课件给出的示例图进行对比，发现趋势相似，开始慢启动状态cwnd数值从MSS开始指数级增长，当发生丢包时，进入快重传与快恢复，可以看到cwnd数值减半，发生超时重传后，cwnd迅速锐减至MSS，重新开始慢启动，当达到ssthress后，cwnd线性增加。

为了观察loss对实验现象的影响，在实验过程中增加了链路的loss为10，也绘制了曲线图进行比较，可以发现超时重传的现象明显增多。



六、实验总结

本次实验对TCP NewReno拥塞控制机制进行了较为完整的实现，难度比前一次实验有所下降，主要实现了拥塞控制状态的迁移以及窗口的控制，这部分实验关于cwnd的实现曾经在数据包队列实验中有所提及，当时在思考题中调研了一些拥塞控制机制，并且也对cwnd的变化趋势进行了分析。现在自己实现了一种拥塞控制机制后，再与前面的实验进行对比反思，感觉收获很多。