

Lab 4 Writeup

My name: 李易

My Student ID: 191830079

This lab took me about 15 hours to do. I did attend the lab session.

Program Structure and Design of the TCPSender:

TCPSender主要包括两大方面的实现：发送segment的逻辑实现（包括接受ack，维护发送segment的数据结构等）和重传逻辑实现（维护一个内部的timer）。

segment的发送最为复杂，涉及到很多相关的细节，因此很容易出bug。主要的任务包括（讲义上给出）：

1. 追踪TCPReceiver的接收窗口（处理收到的ackno和window size）。这里我们根据window size和ackno可以得到发送端的窗口左、右边缘序列号。右边缘序列号减去next_seqno就得到窗口的剩余空间大小。（左边缘序列号对应上一lab中的first unassembled bytes, 正好可以用于做checkpoint）
2. 通过从ByteStream读取，创建新的TCPsegment（如果需要的话，包括SYN和FIN标志）并发送，尽可能地填满接收窗口。只有当接收窗口已满或没有字节可读入（ByteStream为空）时，TCPSender才能停止发送segments。
3. 追踪已发送但是没有收到ackno的segments（称为outstanding segments），在这部分我们把outstanding segment push到相关队列即可。

具体发送segment的逻辑可以参考书上的图实现：

```

/* 假设发送方不受TCP流量和拥塞控制的限制，来自上层数据的长度小于MSS，且数据传送只在一个
方向进行。*/

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (永远) {
    switch (事件)
        事件：从上面应用程序接收到数据e
            生成具有序号NextSeqNum的TCP报文段
            if (定时器当前没有运行)
                启动定时器
            向IP传递报文段
            NextSeqNum=NextSeqNum+length(data)
            break;

        事件：定时器超时
            重传具有最小序号但仍未应答的报文段
            启动定时器
            break;

        事件：收到ACK，具有ACK字段值y
            if (y > SendBase) {
                SendBase=y
                if (当前仍无任何应答报文段)
                    启动定时器
            }
            break;

} /* 结束永远循环 */

```

```

void TCPSender::_segment_handler(TCPSegment &seg)
{
    seg.header().seqno = wrap(_next_seqno, _isn);
    _bytes_in_flight += seg.length_in_sequence_space();
    _next_seqno += seg.length_in_sequence_space();
    _outstanding_segments.push(seg);
    _segments_out.push(seg);
    if(!_timer_running)
    {
        _timer_running = true;
        _time_elapsed = 0;
    }
}

```

然后，收到ack时，将整段小于ackno的segment退出outstanding segment队列。

第二部分的实现逻辑上简单很多。除了发送segment时启动timer之外，只需要在收到完整segment ack时重设RTO和consecutive retransmission和在tick函数中判断是否超时即可。总体讲义上非常清楚，没有复杂难懂的逻辑。

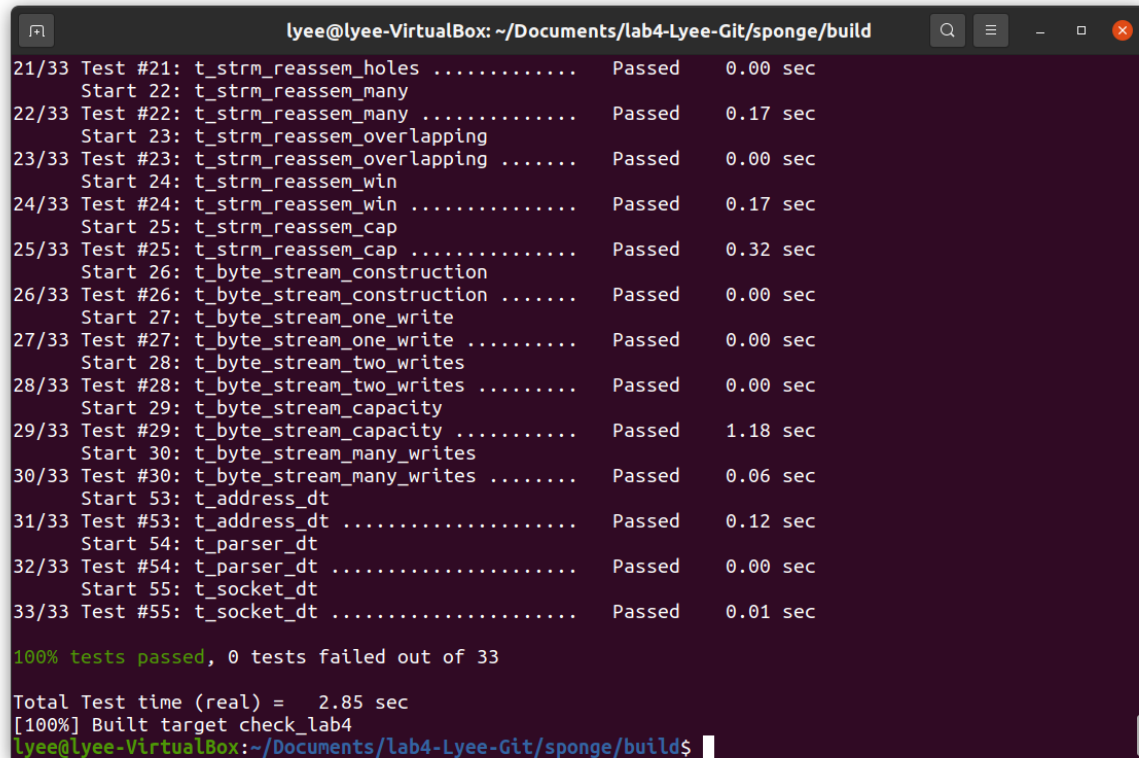
Implementation Challenges:

本次lab同样有一半时间在debug。发送的时候有很多状态变量需要维护。

发送时一定要主要FIN的发送，eof且窗口有剩余空间则segment应该带上FIN。

最耗时的bug在于fill window时，如果设置ByteStream为空时不进入发送循环，则可能出现发送流为空且FIN未发送的情况（FIN恰好没有之前的segment中被捎带）。这时如果不针对发送流为空的情况处理退出，则又有可能出现无限循环。因此只能对特殊情况特殊处理。

Remaining Bugs:



```
lyee@lyee-VirtualBox: ~/Documents/lab4-Lyee-Git/sponge/build
21/33 Test #21: t_strm_reassem_holes ..... Passed    0.00 sec
      Start 22: t_strm_reassem_many
22/33 Test #22: t_strm_reassem_many ..... Passed    0.17 sec
      Start 23: t_strm_reassem_overlapping
23/33 Test #23: t_strm_reassem_overlapping ..... Passed    0.00 sec
      Start 24: t_strm_reassem_win
24/33 Test #24: t_strm_reassem_win ..... Passed    0.17 sec
      Start 25: t_strm_reassem_cap
25/33 Test #25: t_strm_reassem_cap ..... Passed    0.32 sec
      Start 26: t_byte_stream_construction
26/33 Test #26: t_byte_stream_construction ..... Passed    0.00 sec
      Start 27: t_byte_stream_one_write
27/33 Test #27: t_byte_stream_one_write ..... Passed    0.00 sec
      Start 28: t_byte_stream_two_writes
28/33 Test #28: t_byte_stream_two_writes ..... Passed    0.00 sec
      Start 29: t_byte_stream_capacity
29/33 Test #29: t_byte_stream_capacity ..... Passed    1.18 sec
      Start 30: t_byte_stream_many_writes
30/33 Test #30: t_byte_stream_many_writes ..... Passed    0.06 sec
      Start 53: t_address_dt
31/33 Test #53: t_address_dt ..... Passed    0.12 sec
      Start 54: t_parser_dt
32/33 Test #54: t_parser_dt ..... Passed    0.00 sec
      Start 55: t_socket_dt
33/33 Test #55: t_socket_dt ..... Passed    0.01 sec

100% tests passed, 0 tests failed out of 33

Total Test time (real) = 2.85 sec
[100%] Built target check_lab4
lyee@lyee-VirtualBox:~/Documents/lab4-Lyee-Git/sponge/build$
```

感觉进一步还可以对timer进行封装。然后重传的逻辑可能还有一些需要打磨的地方，等下个lab再看。