Lab 3 Writeup
=============

My name: 郑凯琳 TAY KAI LIN

My Student number : 205220025

This lab took me about 10 hours to do. I did attend the lab session.

---

1. Program Structure and Design:

TCP Sender 功能：输出的字节流 --> 不可靠数据报的有效负载段

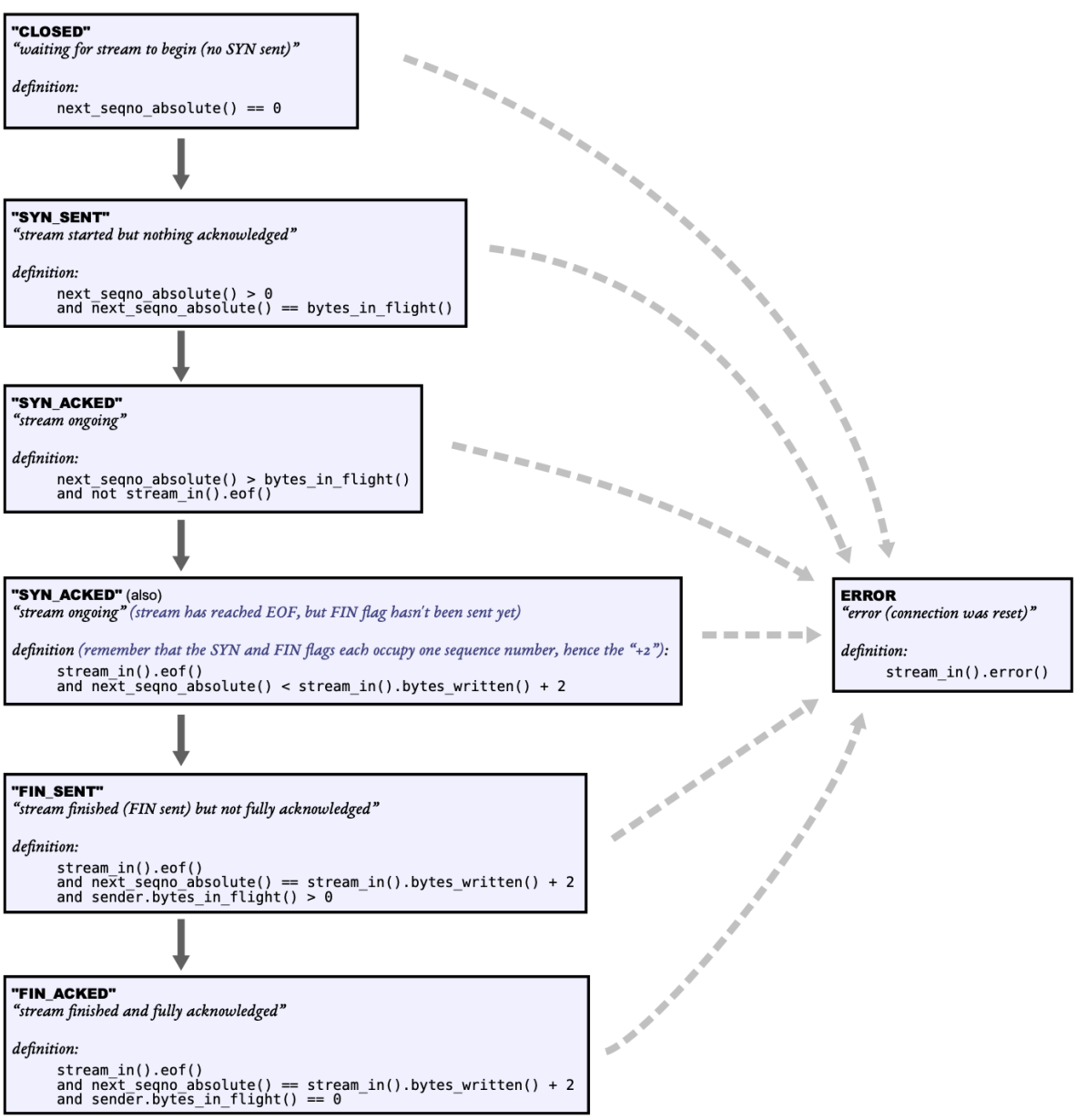
1. 填充，发包
* 根据 Sender 当前状态对可发送的窗口

2. 确认对方当前收到的字节流进度
* 根据对方通知的窗口大小和 ackno

3. 超时重传机制
* 根据时间变化（RTO），定时重传还没有 ack 的报文

<u>3.1 void fill_window() 窗口填充</u>

思路 & 核心代码：



* 配合重传机制的实现：当填充成功一个数据包时，也需要对其进行缓存备份。

（1）//CLOSED
- 条件：初始化
- 需要发送 SYN 包
- 需要填充 SYN 包

Code:

```
    //CLOSED
    if(next_seqno_absolute() == 0)
    {
```

```
        //Send the syn
        TCPSegment seg;
        seg.header().syn = true;
        seg.header().seqno = _isn;

        _segments_out.push(seg);

        _next_seqno = _next_seqno + seg.length_in_sequence_space();
        bytes_in_flight_ = bytes_in_flight_ + seg.length_in_sequence_space();

        //Cache the segments
        _cache_segment(next_seqno_absolute() , seg);
    }
```

（2）//SYN_ACKED
- 条件：`ByteStream` 中还有数据可写入 Sender ，且对方窗口大小足够
- 正常按照 payload 大小限制填充数据包

Code:

```
    //SYN_ACKED
    if(!stream_in().eof() && next_seqno_absolute() > bytes_in_flight())
    {
        size_t seg_maxsz_ = TCPConfig::MAX_PAYLOAD_SIZE;

        size_t wdws_remsz_ = peer_windows_size_ ? peer_windows_size_ : 1;
        size_t flight_size_ = bytes_in_flight();
        if(wdws_remsz_ < flight_size_)
            return;

        wdws_remsz_ = wdws_remsz_ - flight_size_;
        string read_stream_ = _stream.read(min(seg_maxsz_, wdws_remsz_));

        while(!read_stream_.empty())
        {
            TCPSegment seg;
            seg.header().seqno = next_seqno();
            seg.payload() = Buffer(std::move(read_stream_));

            wdws_remsz_ = wdws_remsz_ - seg.length_in_sequence_space();

            if(stream_in().eof() && next_seqno_absolute() <
stream_in().bytes_written() + 2 && wdws_remsz_ >= 1)
            {
                seg.header().fin = true;
```

```
                wdws_remsz_ = wdws_remsz_ - 1;
            }

            _next_seqno = _next_seqno + seg.length_in_sequence_space();
            bytes_in_flight_ = bytes_in_flight_ + seg.length_in_sequence_space();
            _segments_out.push(seg);

            //Cache the seg
            _cache_segment(next_seqno_absolute(), seg);
            read_stream_ = _stream.read(min(seg_maxsz_, wdws_remsz_));
        }
    }
```

（3）//SYN_ACKED
- 条件：`ByteStream` 已经 `eof`，但是 FIN 包还未发送
- 需要填充 FIN 包

Code:

```
    //SYN_ACKED
    //stream has reached EOF, but FIN flag hasn't been sent yet
    if(stream_in().eof() && next_seqno_absolute() < stream_in().bytes_written() +
2)
    {
        //Send the fin
        size_t wdws_remsz_ = peer_windows_size_ ? peer_windows_size_ : 1;
        size_t flight_size_ = bytes_in_flight();
        if(wdws_remsz_ <= flight_size_)
            return;

        TCPSegment seg;
        seg.header().seqno = next_seqno();
        seg.header().fin = true;

        _next_seqno = _next_seqno + seg.length_in_sequence_space();
        bytes_in_flight_ = bytes_in_flight_ + seg.length_in_sequence_space();
        windows_size_ = windows_size_ - seg.length_in_sequence_space();

        _segments_out.push(seg);

        //Cache the segments
        _cache_segment(next_seqno_absolute(), seg);
    }
```

<u>3.2 void ack_received(const WrappingInt32 ackno, const uint16_t window_size) ACK 确认</u>

思路 & 核心代码：

- 若为有效确认（缓存的未确认数据包被成功确认），则需要：
  - 更新重传机制相关变量（重传时间，重传次数）
  - 删除确认成功数据包的缓存
  - 填充窗口 -- 因为对端成功确认了数据，对端可用窗口变大了
- 更新对端窗口大小

Code:

```
    int n_bytes_ = next_seqno() - ackno;
    if(n_bytes_ < 0)
        return;

    //Pop all the cache segments
    vector<uint64_t> acknos_pop_;

    for(auto & seg : segments_cache_)
    {
        uint64_t abs_ackno_ = unwrap(ackno, _isn, seg.first);
        if(abs_ackno_ >= seg.first)
            acknos_pop_.push_back(seg.first);
    }

    for(auto & ackno_pop_ : acknos_pop_)
        segments_cache_.erase(ackno_pop_);

    //Extend window size
    if(window_size > peer_windows_size_)
    {
        windows_size_ += window_size - peer_windows_size_;
    }
    peer_windows_size_ = window_size;

    //Valid ackno
    if(!acknos_pop_.empty())
    {
        windows_size_ = max(window_size, uint16_t(1));
        bytes_in_flight_ = n_bytes_;

        time_ = 0;
        retx_times_ = 0;
```

```
        _initial_retransmission_timeout =
default_initial_retransmission_timeout_;

        fill_window();
    }
```

<u>3.3 void tick(const size_t ms_since_last_tick): 定时重传</u>

思路 & 核心代码：

- 每发送数据包会对其进行缓存
- 当超过 RTO 时间没有收到这个包的确认时，
    - 执行重传
    - 更新等待时间为当前 RTO * 2，
- 若有多个数据包超时，只重传 `seqno` 最小的数据包

Code:

```
    if(segments_cache_.empty())
        return;

    time_ = time_ + ms_since_last_tick;

    if(time_ >= _initial_retransmission_timeout && retx_times_ <=
TCPConfig::MAX_RETX_ATTEMPTS)
    {
        _segments_out.push(segments_cache_.begin()->second);
        time_ = 0;
        if(peer_windows_size_ != 0)
        {
            _initial_retransmission_timeout = _initial_retransmission_timeout *
2;
            retx_times_ += 1;
        }
    }
```

2. Implementation Challenges:

...

...

3. Remaining Bugs:

```
...

...
```