# 《计算机网络》实验作业3-2——基于UDP的可靠传输协议

## 姓名：陈豪斌 学号：1911397 专业：信息安全

### 实验内容

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、确认重传等。**流量控制采用滑动窗口机制**，完成给定测试文件的传输。

### 协议设计

本协议头设计和实验3-1类似，但是多了一条窗口长度的大小设置，以通告发送方接收方剩余的缓冲区容量还有多少。

C++结构定义如下：

```
namespace fake_tcp {
struct ProtocolHeader {
  uint8_t version;

  uint8_t flag;

  uint16_t checksum;

  uint32_t sequence_number;

  uint32_t acknowledgement_number;

  uint32_t message_len;

  uint32_t window_size;
};
} // Namespace fake_tcp.
```

其他协议内部的定义和实验3-1完全一致，在此不赘述。

## 滑动窗口机制设计

TCP并不是每一个报文段都会回复ACK的，可能会对两个报文段发送一个ACK，也可能会对多个报文段发送1个ACK【累计ACK】，比如说发送方有1/2/3 3个报文段，先发送了2,3 两个报文段，但是接收方期望收到1报文段，这个时候2,3报文段就只能放在缓存中等待报文1的空洞被填上，如果报文1，一直不来，报文2/3也将被丢弃，如果报文1来了，那么会发送一个ACK对这3个报文进行一次确认。

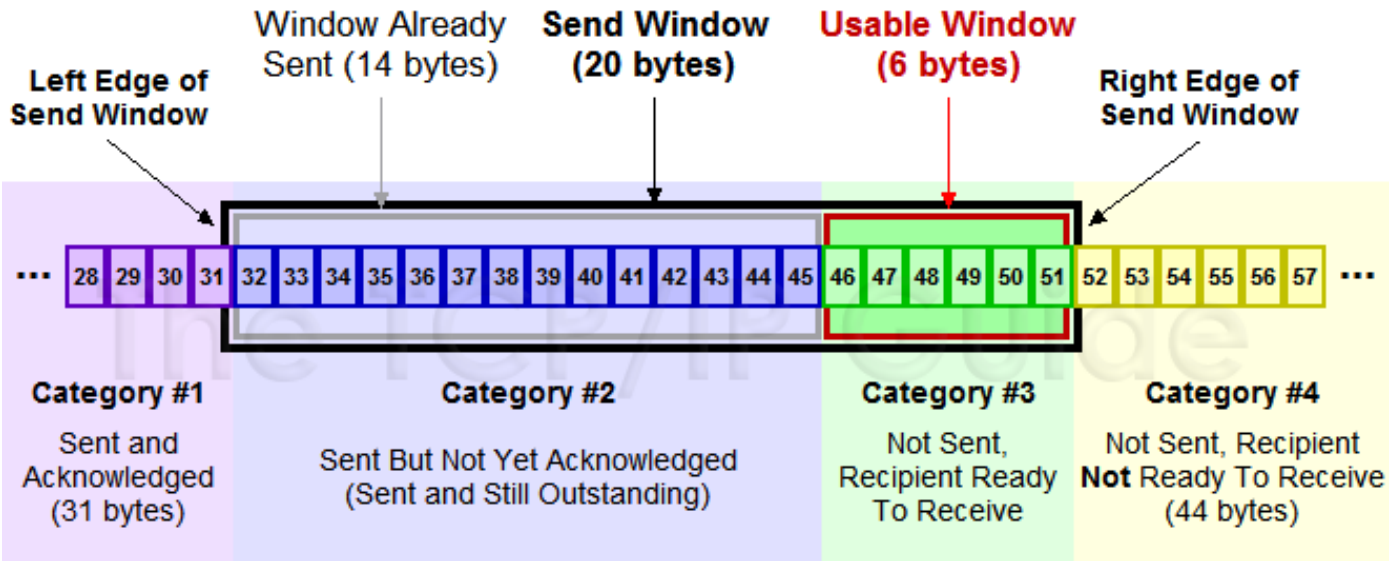
接收方在初始化的时候会用一个宏定义`MAXIMUM_WINDOW_SIZE`来初始化自己的缓冲区大小，在本协议设计中，缓冲区大小设置为`160000`字节，并以`10000`字节为分组大小进行数据包的传送，其定义为：

```
std::vector<std::pair<unsigned char*, size_t>> storage;
// 表示：数据包指针 + 数据包大小
```

同时接收方还会用三个变量来标记窗口的各端：

- `window_left`和`window_right`：以缓冲区指针的形式来标记当前的接收窗口的左端和右端，如果`window_left = window_right`，则说明没有剩余窗口大小可用了，需要等待缓冲区写入到文件后才能重新设置指针指向的位置；
- `next_sequence`：以sequence的形式标记下一个待接收的包的sequence应该是什么，用来一次性将16个数据表写入到文件之中，防止出现乱序情况，让文件的数据发生差错。

对于接收方，一旦缓冲区满了之后就会写入到文件，然后更新窗口大小。而对于乱序包的处理，目前的策略是将其**一律丢弃**。

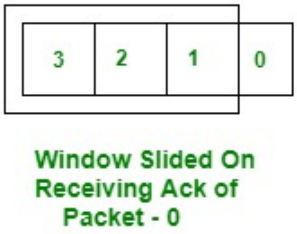



其基本逻辑可以由如下代码阐述：

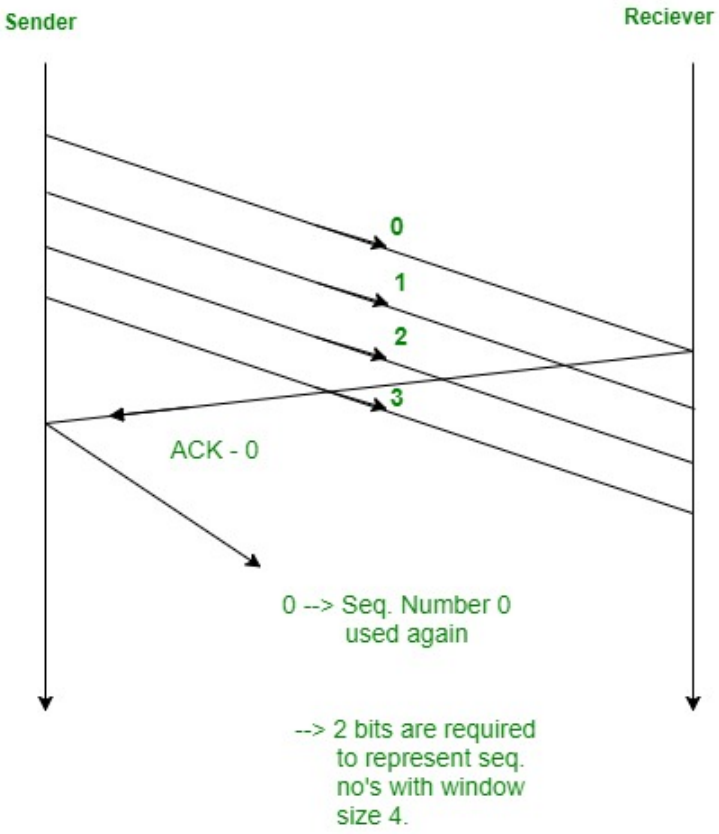
```
        // Increment the next sequence expected.
```

```
        next_sequence++;
        // Push to the queue.
        received_sequences.push_back(seq);
        const uint32_t message_len = *((uint32_t*)(message + 12));
        // Write to the storage.
        unsigned char* data = new unsigned char[message_len];
        memcpy(data, message + HEADER_SIZE, message_len);
        storage.emplace_back(data, message_len);
        check_storage_full();

        // Send back ack.
        unsigned char* send_buf = new unsigned char[HEADER_SIZE];
        bzero(send_buf, HEADER_SIZE);
        // Set message header.
        *((uint8_t*)(send_buf)) = PROTOCOL_VERSION;
        *((uint8_t*)(send_buf + 1)) = fake_tcp::ACK_FLAG;
        *((uint16_t*)(send_buf + 2)) = 0b0;
        *((uint32_t*)(send_buf + 8)) = (seq + 1) % 0xffffffff;
        *((uint32_t*)(send_buf + 16)) =
            MAXIMUM_WINDOW_SIZE / BUFFER_SIZE - storage.size();
```




Window Slided On Receiving Ack of Packet - 0



对于发送方，其只需要维护一个变量为`window_size`用以标记当前剩余可以发送的窗口大小是什么，它也是由`window_right - window_left + 1`的形式给出。在发送方一次性发送完毕所有的数据报文之后，会等待接收方的ACK报文，并重新设置窗口大小或者从重发缓冲区内发送丢失的数据报文给接收方。同样地，发送方也会维护一个变量叫做`last_acked`用来标记累计确认的最后一个sequence对应的数据包是哪个。如果接收方迟迟没有响应的话，就会根据这个变量重新发送数据包，再等待对方的响应。

```
uint32_t cur_sequence;

uint32_t next_sequence;

uint32_t window_left, window_right;

uint32_t window_size;

uint32_t last_acked;
```