《计算机网络》实验作业3-1——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。流量控制采用停等机制，完成给定测试文件的传输。

协议设计

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;
};
} // Namespace fake_tcp.
```

本协议参考了TCP协议头的设计，其协议头可以分为以下几个数据区域：

- 协议版本号`version`，每个实验都会迭代更新一次，本实验设置为无符号整形`0x01ul`。
- 协议类型`flag`，本质上是一个枚举类型，不同的flag的含义如下：

```
enum header_type {
  ERR,  // 0b1
  SYN,  // 0b10
  ACK,  // 0b100
  RST,  // 0b1000
  FIN,  // 0b10000
  BEGIN, // 0b100000
  FILE, // 0b1000000
};
```

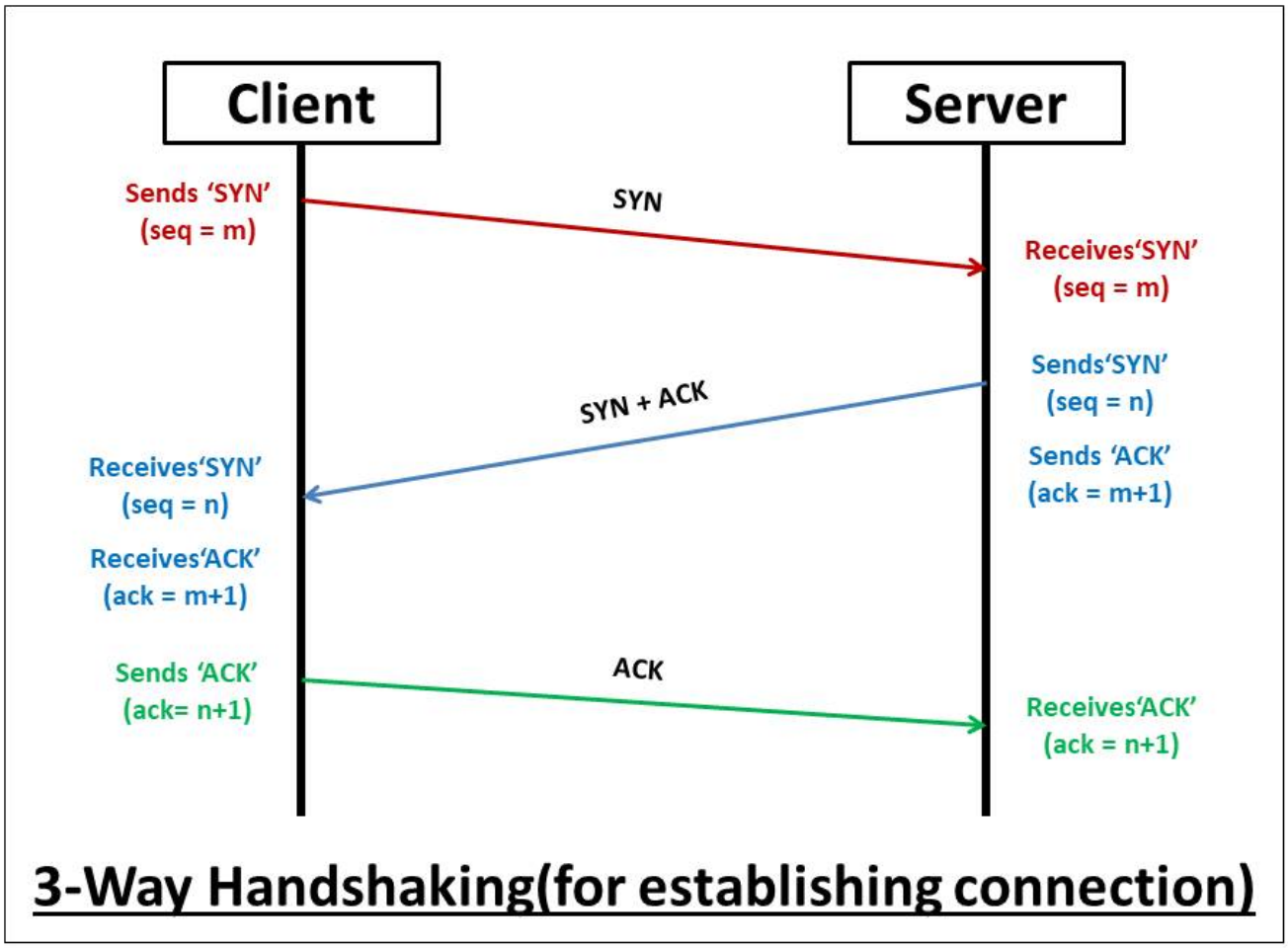
  - `ERR`代表这个协议传输过程中或者对方发生了致命错误，传输的数据尽管通过了校验，但是还是存在问题；
  - `SYN`代表客户端发起握手请求；
  - `ACK`代表双方接收到消息后发送的确认消息；
  - `RST`代表连接重置（一般发生在建立连接最失败、心跳包发送无响应的请求）；
  - `FIN`代表关闭连接；
  - `BEGIN`代表文件传输开始；
  - `FILE`代表文件正在传输。
- 校验和`checksum`，用来保存本次消息的校验和信息；
- 序列号`sequence_number`；
- 应答号`acknowledgement_number = sequence_number + 1`；
- 消息体长度`message_len`；
- 伪协议头`PseudoHeader`，一般仅包含发送方和接收方的IP地址信息以及一些额外信息，例如消息长度和协议信息等，也可以用来计算校验和，定义如下：

```
struct PseudoHeader {
  uint32_t saddr;

  uint32_t daddr;

  // MUST BE ZERO FOR ALIGNMENT.
  uint8_t zeros;

  uint8_t protocol;

  uint16_t packet_len;
};
```

其中全零项就是为了计算校验和的时候能够保证以16比特来处理消息数组。

可靠连接的连接建立过程

参考TCP的三次握手行为，本协议的建立连接过程和TCP三次握手的情况完全一致。



我们暂且假设数据都不出错，但是会发生丢包和延迟。注意客户端和服务器端的socket都设置了超时时间，一旦过了超时时间后就会重发数据包。

服务器端

1. 首先服务器端会绑定到UDP的IP地址和对应的端口，等待客户端的建立连接请求`SYN`数据包的发送；
2. 如果服务器端接收到了`SYN`包，那么它会立即发送一个对应的ACK包给客户端，然后调用函数接口`establish_connection_server`，来进行建立连接的过程：

```
int fake_tcp::Party::establish_connection_server(const uint32_t& sequence)
{
    unsigned char* recv_buffer = new unsigned char[BUFFER_SIZE];
```

```
    handle_recv(socket, recv_buffer, &addr, &dst_addr);

    const uint8_t flag = *((uint8_t*)(recv_buffer + 1));
    const uint32_t seq = *((uint32_t*)(recv_buffer + 4));
    const uint32_t ack = *((uint32_t*)(recv_buffer + 8));

    if (flag == fake_tcp::ACK_FLAG && ack == sequence + 1) {
      std::cout << "[Server] Connection established!" << std::endl;
      storage.emplace_back(new unsigned char[STORAGE_SIZE]);
      established = true;
      return 0;
    } else if (flag == fake_tcp::BEGIN_FLAG) {
      // Send back reset flag.
      std::cout << "[Server] Not yet established! Try to send a RESET flag..."
                << std::endl;
      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::RST_FLAG;
      *((uint16_t*)(send_buf + 2)) = 0b0;
      *((uint32_t*)(send_buf + 4)) = 0b0;
      *((uint32_t*)(send_buf + 8)) = 0b0;

      bool context = false;
      do_check(&context, fake_tcp::dst_ip, fake_tcp::src_ip, send_buf,
               HEADER_SIZE);

      sendto(socket, send_buf, HEADER_SIZE, 0, (struct sockaddr*)(&dst_addr),
             sizeof(sockaddr));

      return -1;
    }
}
```

其机理为，阻塞接受自身的socket上的信息，并根据协议指定的解析方式对接受的数据进行处理，并查看是否为客户端的ack回复以及数字大小是否正确。注意很有可能客户端在服务器尚未建立连接之前就开始发送数据包了，这种情况下服务器端会告诉它尚未建立连接，返回一个RST数据包，让客户端重新开始。

客户端

客户端相对服务器端的逻辑就更加简单了，它发送`SYN`包然后接受应答后发送`ACK`就认为自己已经建立了连接。

差错检测

差错检测是基于校验和的检测。正如TCP检测差错的过程，本协议也会从UDP层获取发送方和接收方的IP地址，然后拼接成伪协议头，最后和协议头、消息体共同组成差错检验的数据缓冲区，进行差错检验。核心思想是将数据以字对齐，剩下不足的位数补零。然后我们将所有的数据加在一起后取反即为checksum的值。在差错检验之前这个域暂且会被设置成为0。差错检验的代码如下：

```
uint16_t fake_tcp::checksum(unsigned char* header, size_t len) {
  // Check.
  uint16_t* ptr = reinterpret_cast<uint16_t*>(header);
  uint16_t sum = 0;
  while (len > 1) {
    sum += *(ptr++);
    len -= 2;
  }
  if (len) {
    sum += ((*ptr) & htons(0xff00));
  }
  // Fold sum to 16 bits: add carrier to the result
  while (sum >> 16) {
    sum = (sum & 0xffff) + (sum >> 16);
  }

  return (uint16_t)(~sum);
}
```

这样检验和的验证和计算也很简单，只是验证的时候看看校验和的值是否为0就可以了。

确认重传机制

在发送一个数据之后，就开启一个定时器，若是在这个时间内没有收到发送数据的ACK确认报文，则对该报文进行重传，在达到一定次数还没有成功时放弃并发送一个复位信号（我们认为对方应该是已经退出了连接）。这一部分我们可以采用循环+socket的延时机制来实现。

```
// Set default timeout for the socket. Just for convenience.
struct timeval timeout;
timeout.tv_sec = 0;
timeout.tv_usec = 1000000;
// Try to set the timeout.
setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
```