南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

计算机网络实验报告

编程作业 3 基于 UDP 服务设计可靠传输协议并编程实现$_2$

于文明

年级：2020 级

专业：信息安全

指导教师：徐敬东，张建忠

2022 年 12 月 3 日

摘要

关键字：socket UDP rdt3.0

目录

一、 实验内容

在实验 3-1 的基础上，将停等机制改成基于滑动窗口的流量控制机制，采用固定窗口大小，支持累积确认，完成给定测试文件的传输。

作业要求

- 多个序列号；
- 发送缓冲区、接受缓冲区；
- 滑动窗口：Go Back N；
- 有必要日志输出（须显示传输过程中发送端、接收端的窗口具体情况）。

二、 协议设计 [1]

本实验在 3_1 的基础上，采用 GBN(Go-Back-N) 协议

(一) 报文格式

报文格式仍然延续了实验 3-1 的报文格式，具体如下:

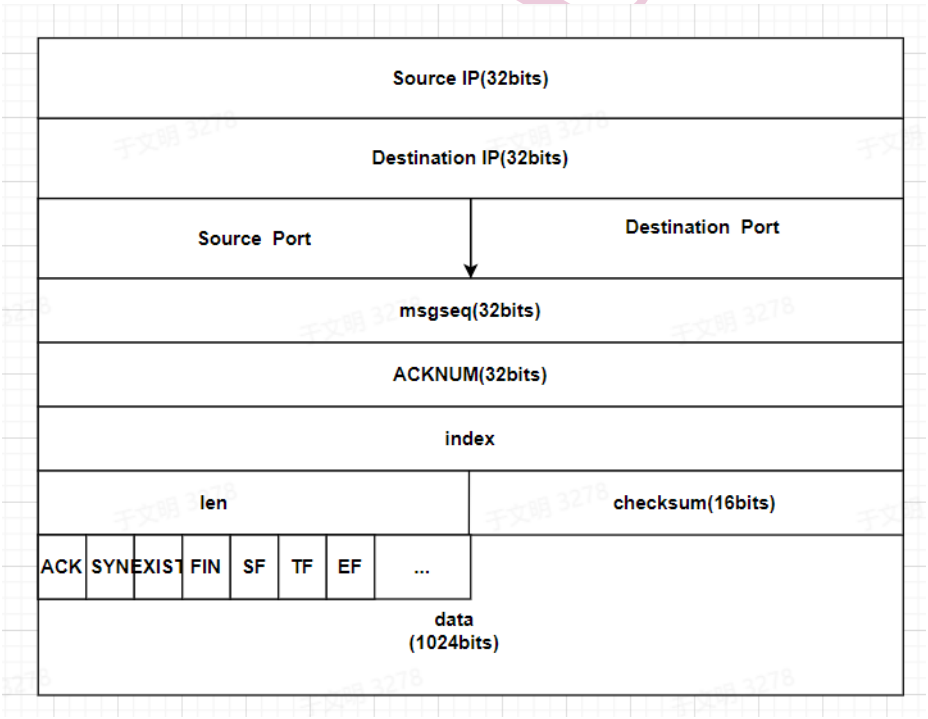

图 1: 报文格式

消息序列号字段 msgseq, 每次发送数据包，msgseq 增加 1

确认号字段 acknum, 它表示接收方期望收到发送方下一个报文段的第一个字节数据的编号。其值是接收计算机即将接收到的下一个序列号，也就是下一个接收到的字节的序列号加 1。在 GBN 的接收端，可以累计确认检查 acknum 确认消息发送成功

(二) 流程图

实验采用 GBN 协议和累计确认机制保证消息接收的正确性，防止丢包，具体交互图如下:

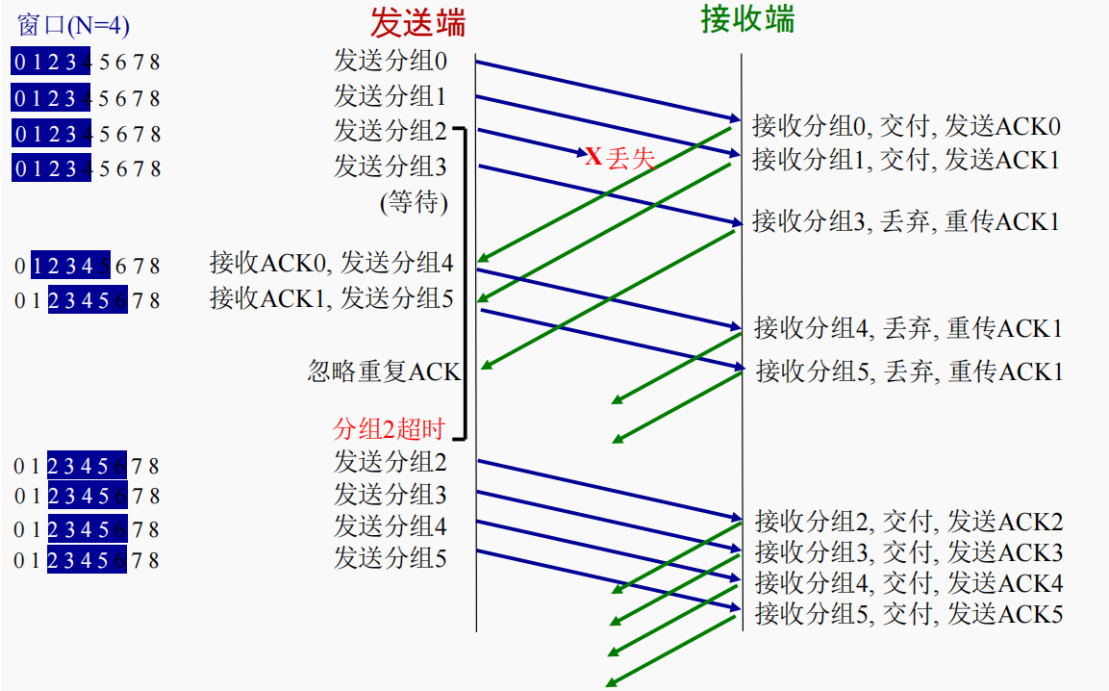

图 2: 交互

(三) 状态转换机 (FSM)

本实验程序部分根据 GBN 状态机实现, 发送端和接收端如下:

1. 客户端

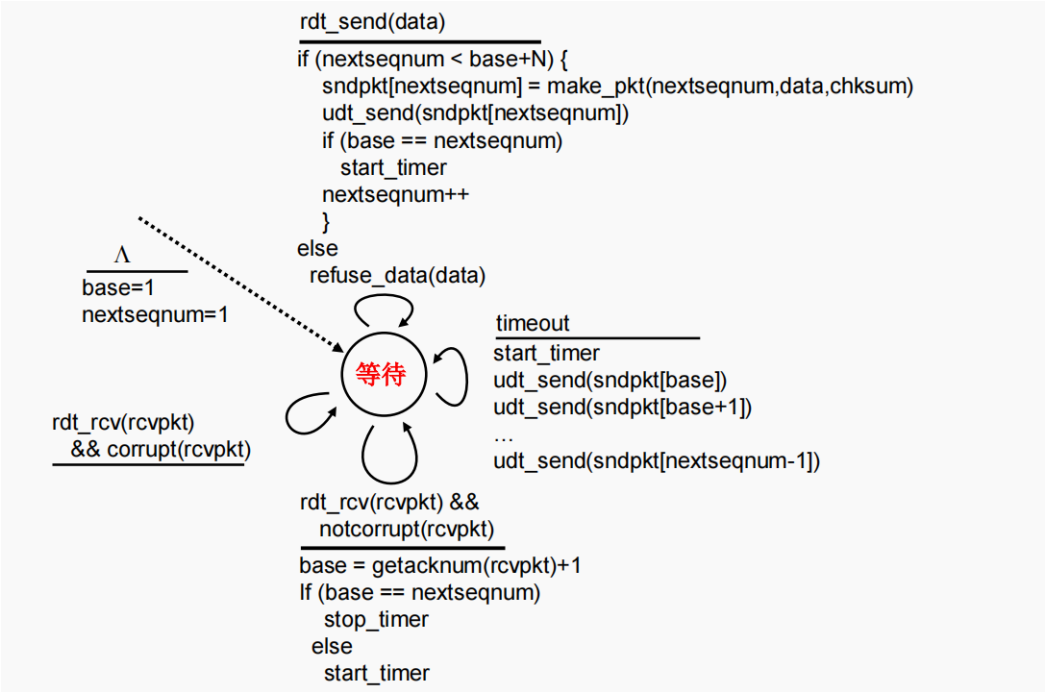

图 3: 客户端状态机

可以将客户端的状态机分为以下几个状态:

- 初始状态，base,nexseqnum 赋初始值
- 发送数据包，如果 nextseqnum<base+N, 即将要发送的数据包在窗口内，发送数据包, 如果 base==nextseqnum, 即开始发送窗口内的数据包, 设置时钟.nextseqnum++
- 接收 ack, 检查校验和无误后，更新 base, 如果 base==nextseqnum, 停止时钟否则，更新时钟
- 超时，重置时钟，发送窗口内未确认的数据包

2. 服务端

只使用 ACK，确认按序正确接收的最高序号分组会产生重复的 ACK，需要保存希望接收的分组序号 (expectedseqnum)

失序分组（未按序到达）处理

- 不缓存、丢弃
- 重发 ACK，确认按序正确接收的最高序号分组

服务端状态机如下:

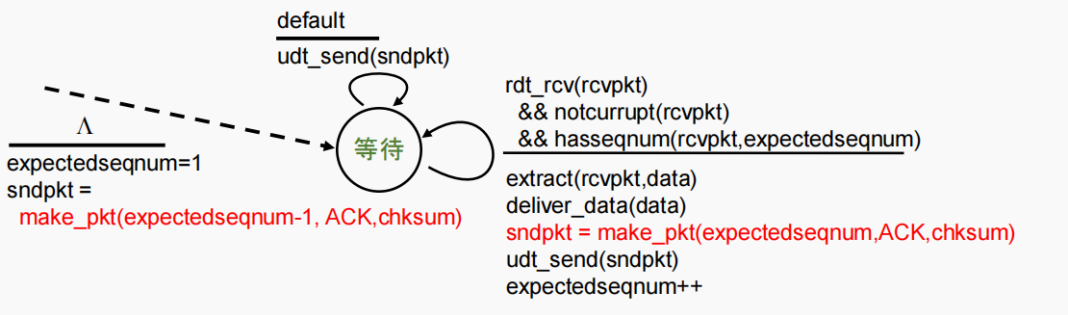

图 4: 服务端状态机

三、 核心代码

(一) 客户端状态机实现

sendFSM

```
while (1) {
    //printf("pktNum:%d\n", pktNum);
    if (base > pktNum) {
        break;
    }
    //滑动窗口有空闲,发送包
    if (base <= nextseqnum && nextseqnum < (base + N) && nextseqnum <=
        pktNum) {
        sendto(clientSocket, (char*)&sendpkts[nextseqnum], sizeof(
            sendpkts[nextseqnum]), 0, (struct sockaddr*)&servAddr, sizeof
            (servAddr));
        log(sendpkts[nextseqnum], 1);
        if (base == nextseqnum) {
            start = clock();    //设置时钟
            count = 0;
            stopTimer = true;
        }
        nextseqnum++;
        //printf("%d %d\n", base, nextseqnum);
    }
```

(二) 累计确认

recvFSM

```
            //超时重传 timeout
    if (stopTimer && clock() - start > TIME_OUT) {
        if (count >= MAX_TIMES) {
            printf("重传次数过多, 文件传输失败! \n");
            return 0;
```

```
            }
            printf("开始超时重传! \n");
            start = clock();
            stopTimer = true;
            for (int j = base; j < nextseqnum; j++) {
                sendto(clientSocket, (char*)&sendpkts[j], sizeof(sendpkts[j])
                    , 0, (struct sockaddr*)&servAddr, sizeof(servAddr));
                log(sendpkts[j], 1);
            }
            count++;
        }
```

(三) 服务端状态机实现

recvFSM

```
        if (pkt.checkChecksum()) {
            if (pkt.seq == expectednumseq) {
                memcpy(buffer[i], pkt.data, pkt.datalen);
                pkt2.setAck();
                pkt2.acknum = expectednumseq++;
                pkt2.setChecksum();
                memcpy(&lastACK, &pkt2, sizeof(packet));
                sendto(serverSocket, (char*)&pkt2, sizeof(pkt2), 0, &clntAddr
                    , nSize);
                log(pkt2, 1);
                i++;
            }
            else {
                printf("数据传输乱序! \n");
                sendto(serverSocket, (char*)&lastACK, sizeof(lastACK), 0, &
                    clntAddr, nSize);
                log(lastACK, 1);
            }
        }
        else {
            printf("校验和错误! \n");
        }
```

四、 附加部分

(一) ack 接收多线程

将 client 端的接收 ack 启动一个多线程处理

ackhandler

```
DWORD WINAPI ACKHandler(LPVOID param) {
```

```
    //接收ACK
    packet temp;
    while (recvfrom(clientSocket, (char*)&temp, sizeof(temp), 0, &fromAddr, &
        addrLen) > 0) {
        log(temp, 2);
        if (temp.checkChecksum()) {
            if (temp.ackseq >= base && temp.ackseq < base + N) {
                base = temp.ackseq + 1;
                if (base == nextseqnum) {
                    stopTimer = false;  //关闭时钟
                    return 0;
                }
                else {
                    start = clock();
                    stopTimer = true;       //重启时钟
                    //count = 0;
                }
            }
            else {
                printf("无效的ACK序列号! \n");
            }
        }
        else {
            printf("校验和错误! \n");
        }
    }
};
```

(二) 缓冲区大小的设置

初始我将发送和接收的缓冲区大小设置得很大, 这样就导致了一些问题: 比如对内存的占用很大, 效率低。之后根据助教提示, 将缓冲区设置为 packet 长度, 然后只存储一个数据包, 每次读取相应文件内容后立即发送, 之后清理缓冲区。

ackhandler

```
filebuffer[10240]
...

sendfile():
    //读取1024个字节, 然后发送
    memset(filebuffer,0,sizeof(filebuffer));
    readfile();
    memcpy(packet,filebuffer,sizeof(filebuffer));
    sendto(...)
```

五、 实验结果

(一) 发送端日志

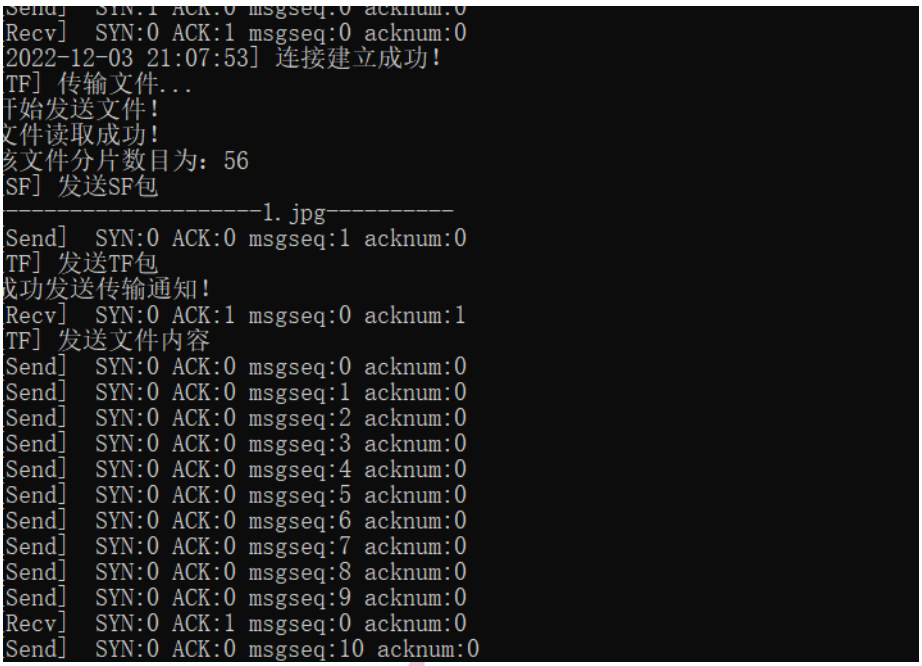

图 5: 发送端日志

(二) 路由器设置

图 6: 路由器设置

(三) 丢包记录

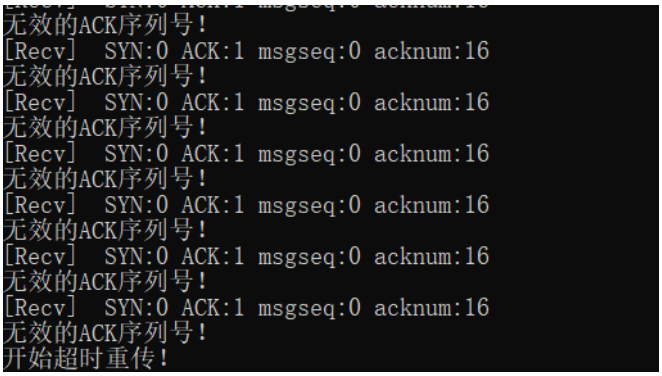

图 7: 路由器设置

(四) 接收端日志

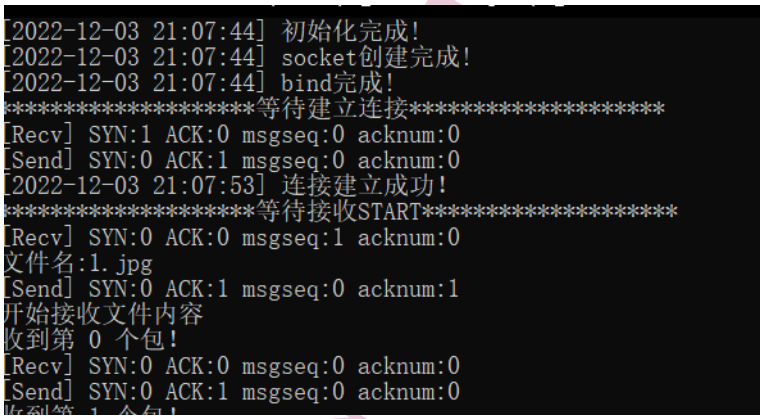

图 8: 接收端设置

六、 附录

本实验相关的资源 (源文件,测试文件,实验结果截图等) 都已上传至 github(https://github.com/Metetor/network)

参考文献

[1] 基思.W. 罗斯詹姆斯.E. 库罗斯. **计算机网络*:* 自顶向下方法**. 机械工业出版社, 2009.