

实验3-3基于UDP服务设计拥塞控制的可靠传输

姓名: 孟笑朵

学号: 2010349

- 实验要求
- 参考资料
- 协议设计
- 实验原理
- 程序运行界面

实验要求

在实验3-2的基础上，选择实现一种 **拥塞控制算法**，也可以是改进的算法，完成给定测试文件的传输。

拥塞控制算法: 本次实验中采用的拥塞控制算法是reno算法，当收到三个重复的ACK或是超过了RTO时间且尚未收到某个数据包的ACK，Reno就会认为丢包了，并认定网络中发生了拥塞。Reno会把当前的sssthresh的值设置为当前cwnd的一半，但是并不会回到slow start阶段，而是将cwnd设置为（更新后的）sssthresh+3MSS，之后cwnd呈线性增长。

建立连接为三次握手模式🤝，断开连接为两次挥手模式👋与实验3-1一致，累计确认和重传机制和实验3-2中一致，这里不做赘述。

参考资料

1. [4.17 如何基于 UDP 协议实现可靠传输? | 小林coding\(xiaolincoding.com\)](http://xiaolincoding.com/4.17)
2. [4.2 TCP 重传、滑动窗口、流量控制、拥塞控制 | 小林coding\(xiaolincoding.com\)](http://xiaolincoding.com/4.2)

协议设计

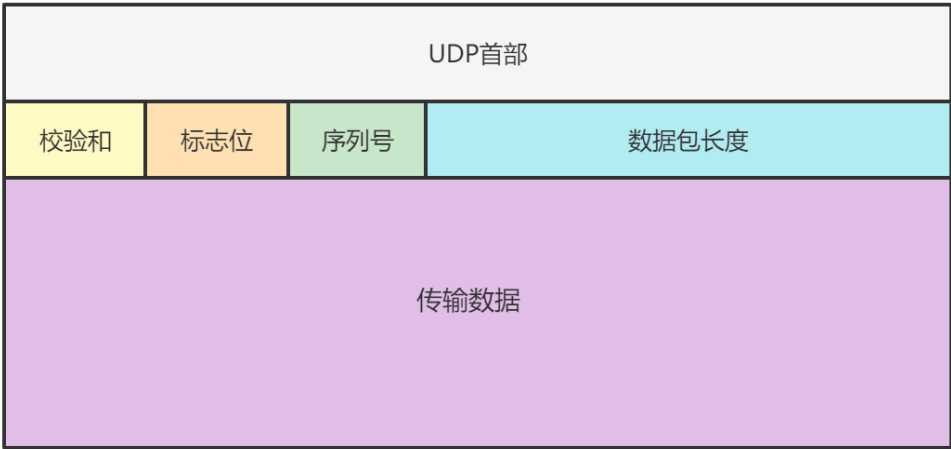
在协议设计部分, 与实验3-1协议设计部分一致.
我们知道UDP传输本身有自己的数据报格式, 如下图所示

■ UDP数据报格式

- 长度: 包含头部、以字节计数
- 校验和: 为可选项, 用于差错检测



另外, UDP在发送和接收数据报的时候会产生**伪首部**, 用于计算校验和, 伪首部包含了源IP地址和目的IP地址, 以及协议类型等字段, 在这里为了专门检验传输数据部分, 我们专门计算了针对当前传输数据的校验和, 在UDP本身数据报报头的基础上, 我们在数据部分增加专门用于可靠传输的协议头部分, 如下所示为当前数据报报头:



说明

1. 校验和: 一个字节, 专门用于计算标志位数据部分的校验和;
2. 标志位: 一个字节, 包含三次连接的SYN, ACK, 两次挥手的FIN, 数据传输的ACK, 数据传输是否为最后一个数据报的END标志位;
3. 序列号: 一个字节, 数据传输的SEQ
4. 数据包长度: 传输数据为可选长度, 标记传输数据的长度, 4个字节(也就是说Data部分不能超过4个字节表示的数据范围)

实验原理

Reno算法主要涉及以下几个重要阶段:

慢启动阶段：

初始拥塞窗口 $CWND=1$ ，经过每个RTT时间， $CWND$ 翻倍。在慢启动阶段，拥塞窗口大小呈指数形式增长。在每收到一个ACK后， $CWND$ 加一（直至 $CWND$ 达到阈值 $SSTH$ ）。

当连接初始建立或报文超时未得到确认时，TCP拥塞控制进入慢启动阶段。

拥塞避免：

拥塞窗口达到该阈值时，慢启动阶段结束，进入拥塞避免阶段。在拥塞避免阶段，每个RTT时间， $CWND$ 加一，此时拥塞窗口呈线性增长。

快速恢复：

当收到三个重复的ACK或是超过了RTO时间且尚未收到某个数据包的ACK，Reno就会认为丢包了，并认定网络中发生了拥塞。

Reno会把当前的 $ssthresh$ 的值设置为当前 $cwnd$ 的一半，但是并不会回到slow start阶段，而是将 $cwnd$ 设置为（更新后的） $ssthresh+3MSS$ ，之后 $cwnd$ 呈线性增长。

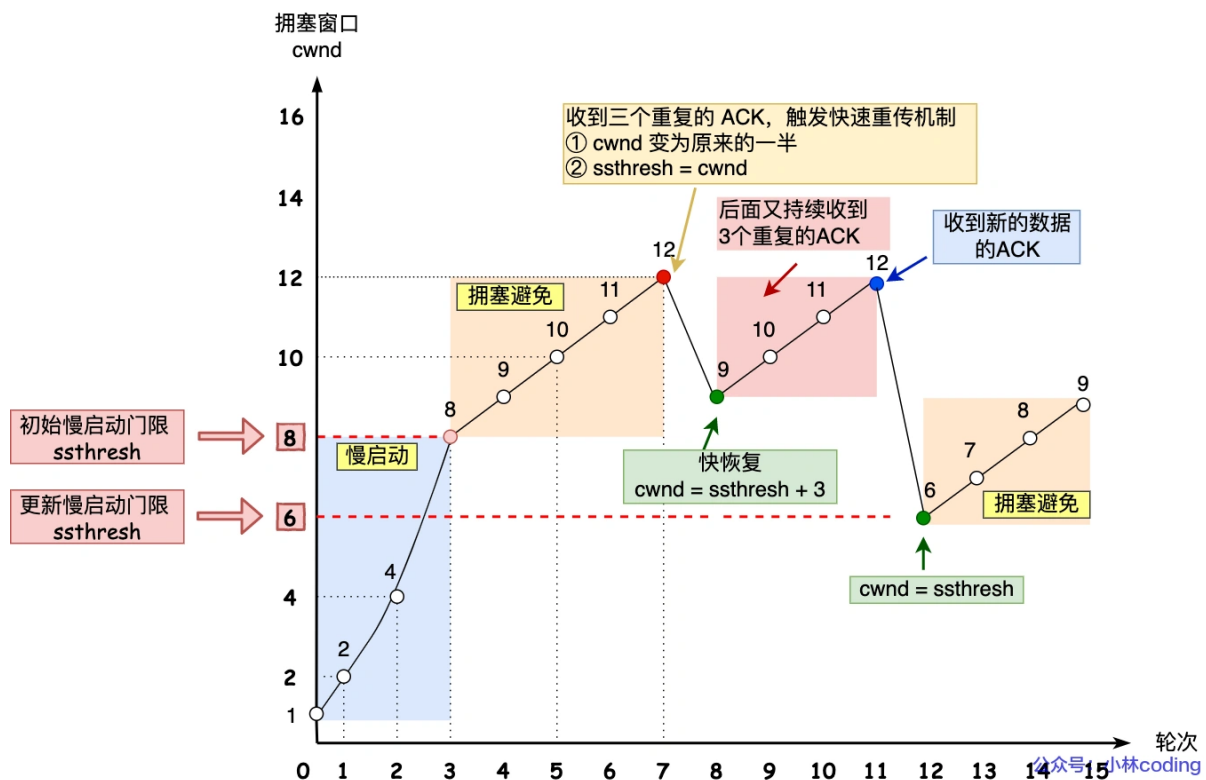
丢失检测：

（1）超时检测：阈值 $ssthresh = cwnd/2$ ； $cwnd=1$ ，进入慢启动阶段

（2）丢失检测：通过三次重复ACK检测丢失。阈值 $ssthresh = cwnd/2$ ； $cwnd=ssthresh+3$ ，进入线性增长（拥塞避免阶段）。

在恢复重传的过程中，采用的恢复方式为GBN，所有未经确认的数据包都会被重传。

上述过程可以用如下所示的图来解析：



根据上述实验原理，可以设计对应的拥塞控制算法代码，如下所示为客户端接受数据线程：

//接收数据线程

```

DWORD WINAPI recv_msg(LPVOID lparam) {
    char recv[3];
    int lentmp = sizeof(m_ServerAddress);
    while (1) {
        //确认完毕所有数据包
        if (send_ok == num)
            break;
        if (recvfrom(m_ClientSocket, recv, 3, 0,
(sockaddr*)&m_ServerAddress, &lentmp) != SOCKET_ERROR && check_sum(recv,
3) == 0 &&
            recv[1] == ACK) {
            if (startflag)dupACK++;
            if ((unsigned char)recv[2] == unsigned char((send_ok) %
((int)UCHAR_MAX + 1))) {
                if (CWND <= SSTH)
                {//慢启动阶段
                    CWND++;
                    mtx.lock();
                    printf("%s%d\n", "当前窗口大小:", CWND);
                    mtx.unlock();
                }
                else//拥塞避免
                {
                    //线性增长
                    cwnd_temp++;
                    if (cwnd_temp % CWND == 0 && CWND < (int)UCHAR_MAX) {
                        CWND++;
                        mtx.lock();
                        printf("%s%d\n", "当前窗口大小:", CWND);
                        mtx.unlock();
                    }
                }
            }
            send_ok++;
            mtx.lock();
            printf("%s%d\n\n", "已经确认", send_ok);
            list.pop();
            mtx.unlock();
            base++;
            dupACK = 0;
        }
    }
}

```

```

        startflag = 1;
        resend = 0;
        continue;
    }
    if (dupACK == 3)//快恢复阶段
    {
        startflag = 0;
        dupACK = 0;
        SSTH = CWND / 2;
        CWND = CWND / 2 + 3;//进入拥塞避免阶段
        cwnd_temp = SSTH + 1;
        next_send = base;
        resend++;
        mtx.lock();
        has_send -= list.size();
        printf("%s%d\n", "当前窗口大小:", CWND);
        while (!list.empty())
        {
            list.pop();
        }
        cout << "重传第" << has_send << "个包" << endl;
        mtx.unlock();
        continue;
    }
    //超时
    if (clock() - list.front().first > TIMEOUT) {
        startflag = 0;
        dupACK = 0;
        SSTH = CWND / 2;
        CWND = 1;
        cwnd_temp = SSTH + 1;//进入慢启动阶段
        next_send = base;
        resend++;
        mtx.lock();
        has_send -= list.size();
        printf("%s%d\n", "当前窗口大小:", CWND);
        while (!list.empty())
        {
            list.pop();
        }
    }

```

```


        cout << "重传第" << has_send << "个包" << endl;
        mtx.unlock();
    }
}
//出错
else {
    startflag = 0;
    dupACK = 0;
    SSTH = CWND / 2;
    CWND = 1; //慢启动
    cwnd_temp = SSTH + 1;
    next_send = base;
    resend++;
    mtx.lock();
    has_send -= list.size();
    printf("%s%d\n", "当前窗口大小:", CWND);
    while (!list.empty()) {
        list.pop();
    }
    cout << "重传第" << has_send << "个包" << endl;
    mtx.unlock();
}
}
return 1;
}

```

程序运行界面

程序运行方式:

1. 按照下图所示设置好路由文件的路由器IP地址和服务器IP地址

 Router

路由器IP:

服务器IP:

端口:

服务器端口:

丢包率:
 %

延时:
 ms

确定


修改


日志

count:3.
 Delay 1 ms.
 count:4.
 Delay 1 ms.
 count:5.
 Delay 1 ms.
 count:6.
 Delay 1 ms.
 count:7.

2. 先运行服务端,再运行客户端;

3. 将对应的素材放置于Client.exe所在文件夹中,在客户端运行输入对应文件名称即可看到程序传输运行状态,结束后在对应的Server.exe所在文件夹中,可以看到对应传输成功的文件.

 D:\MySpace\MaterialsSpace\大三上\计算机网络

 D:\MySpace\MaterialsSpace\大三上\计算机网络\2010349_孟笑朵_实验3-3\Client\Client.exe

```

已经确认接受第1159个包
已经确认接受第1160个包
已经确认接受第1161个包
已经确认接受第1162个包
已经确认接受第1163个包
已经确认接受第1164个包
已经确认接受第1165个包
已经确认接受第1166个包
已经确认接受第1167个包
已经确认接受第1168个包
已经确认接受第1169个包
已经确认接受第1170个包
已经确认接受第1171个包
已经确认接受第1172个包
已经确认接受第1173个包
已经确认接受第1174个包
已经确认接受第1175个包
已经确认接受第1176个包
已经确认接受第1177个包
已经确认接受第1178个包
已经确认接受第1179个包
已经确认接受第1180个包
我到这里啦信息接收成功
文件接收成功

开始断开连接
成功接收第一次挥手消息
成功发送第二次挥手
挥手成功, 断开连接
请按任意键继续. . .
        
```

```

已经发送1178
当前窗口大小:6
已经确认1178

sending seq:154
已经发送1179
当前窗口大小:1
重传第1178个包
sending seq:154
已经发送1179
当前窗口大小:2
已经确认1179

sending seq:155
已经发送1180
当前窗口大小:3
已经确认1180

文件发送成功
文件传输时间 118 s
吞吐量: 399.898644 kbps

开始断开连接
客户端发送挥手消息
客户端发送挥手消息
客户端发送挥手消息
客户端发送挥手消息
接收第二次挥手
挥手成功, 断开连接
请按任意键继续. . .
        
```