计算机网络编程
实验报告

班级：07111707

组长：1120171189 崔程远

成员：1120172149 吴沁璇

1120172153 张澈

1120172163 王晓媛

1120172736 张鉴昊

1120172765 曾煜瑾

1120173326 曾紫飞

北京理工大学

计算机学院

2020 年 5 月

第六章 实验 6　基于 TCP 拥塞控制算法实现数据发送

1. 实验目的

加深对 TCP 拥塞控制算法的掌握和理解。

2. 实验内容

按照 TCP 拥塞控制算法实现数据发送，不必考虑接收确认，仅仅按照拥塞窗口的变化进行数据发送。配置文件指定 MSS 和初始门限，出现三个重复 ACK 的轮次，出现超时的轮次，以及最后的结束轮次。接收程序简单接收信息即可。

配置文件关键要点：

MSS=1024

Threshold=32768

TriACKRound=16

TimeoutRound=22

EndRound=26

程序运行屏幕输出要点：

发送程序显示 MSS 和初始门限值

显示轮次数和当前拥塞窗口值

显示发送数据

重复一直到结束轮次

3. 实验原理

TCP 进行拥塞控制的算法有四种，即慢开始、拥塞避免、快重传和快恢复。

慢开始指的是：当主机开始发送数据时，由于并不清楚网络的负荷情况，所以如果立即把大量数据字节注入网络，那么就有可能引起网络发生阻塞。经验证明，较好的方法是先探测一下，即由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大阻塞窗口数值。在 TCP 刚刚连接好并开始发送 TCP 报文段时，先令拥塞窗口 cwnd = 1，即一个最大报文段长度 MSS。每收到一个对新报文段的确认后，将 cwnd 加 1，即增大一个 MSS。使用慢开始算法后，每经过一个传输轮次，拥塞窗口 cwnd 就会加倍，即 cwnd 的大小指数式增长。这样慢开始一直把拥塞窗口 cwnd 增大到一个规定的慢开始门限 ssthresh，然后改用拥塞避免算法。

拥塞避免指的是：发送端的拥塞窗口 cwnd 每经过一个往返时延 RTT 就增加一个 MSS 的大小，而不是加倍，使 cwnd 按线性规律缓慢增长（即加法增大），而当出现一次超时（网络拥塞）时，令慢开始门限 ssthresh 等于当前 cwnd 的一半（即乘法减小）。这样做的目的是迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完。

快重传指的是：接收方不要等待自己发送数据时才进行捎带确认，而是要立即发生确认，即使收到了失序的报文段也要立即发出对已收到的报文段的重复确认。快重传算法规定，发送方只要一连收到 3 个重复确认，就知道接收方确实没有收到报文段，因而应当立即进行重传。

快恢复指的是：发送端收到连续 3 个重复确认时，执行“乘法减小”算法，把慢开始门限 ssthresh 设置为出现拥塞时发送方 cwnd 的一半。与慢开始的不同之处是，它把 cwnd 的值设置为慢开始门限 ssthresh 改变后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

实际上，慢开始、拥塞避免、快重传和快恢复几种算法应当是同时应用在拥

塞控制机制之中的，当发送方检测到超时的时候，就采用慢开始和拥塞避免，当发送方接收到重复确认时，就采用快重传和快恢复。

4. 实验环境

语言	集成开发环境	编译器
C++	Visual Studio 2017	gcc version 4.8.1
Java	Eclipse 2019	java version "1.8.0_65"
Python	Pycharm 2017	Python 3.7.0

5. 实验步骤

三份代码的结构和输出完全一致，下面以C++代码为例进行分析

- 变量定义

```
int MSS = 1024;
int Threshold = 32768;
int TriAckRound = 16;
int TimeoutRound = 22;
int EndRound = 26;
```

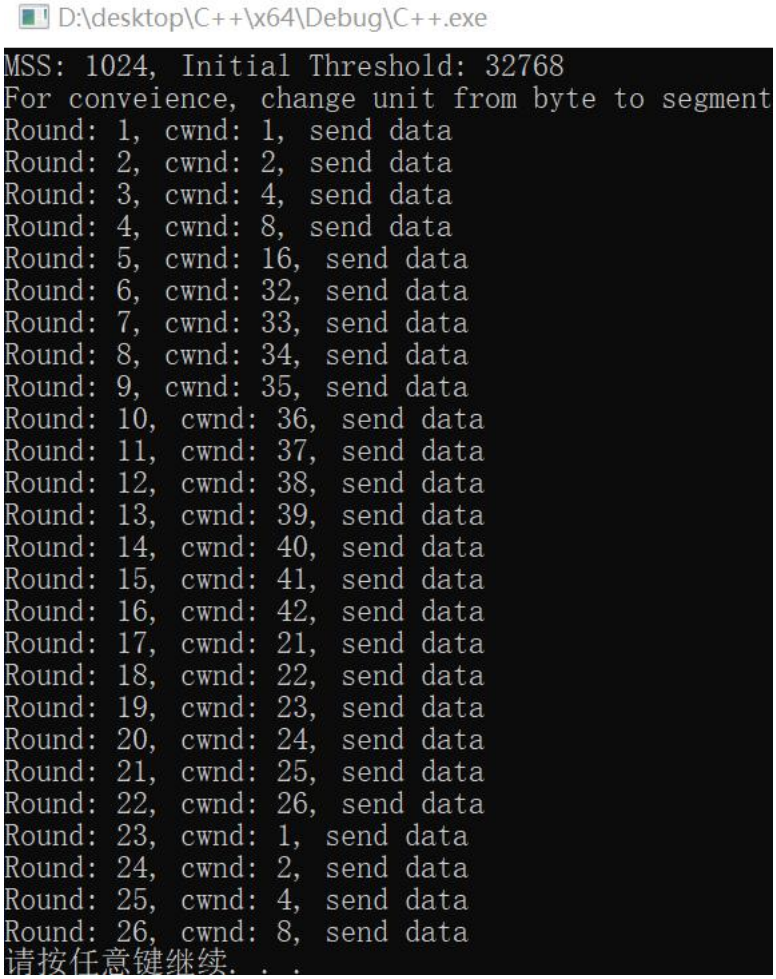
- 处理过程

```
int cwnd = 1;
    int ssthresh = Threshold / MSS;
    printf("MSS: %d, Initial Threshold: %d\n", MSS, Threshold);
    printf("For conveience, change unit from byte to segment, so initial threshold
is %d segments\n", ssthresh);
    for (int i = 1; i <= EndRound; i++)
    {
        printf("Round: %d, cwnd: %d, send data\n", i, cwnd);
        if (i == TriAckRound)
        {
            ssthresh = cwnd / 2;
            cwnd = ssthresh;
        }
        else if (i == TimeoutRound)
        {
            ssthresh = cwnd / 2;
            cwnd = 1;
        }
        else
        {
            if (cwnd < ssthresh)
            {
                cwnd *= 2;
            }
            else
            {
                cwnd++;
            }
```

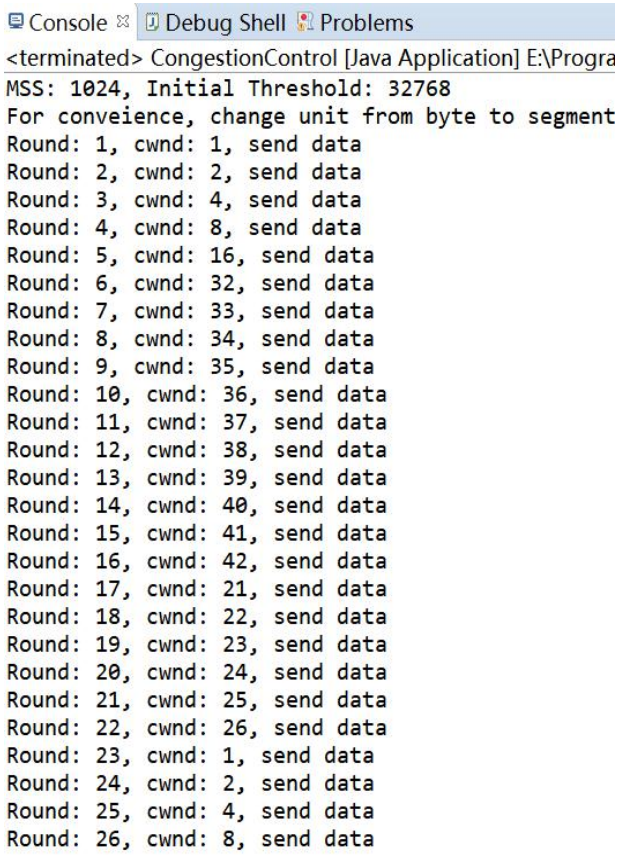
```
        }
    }
```

6. 实验结果

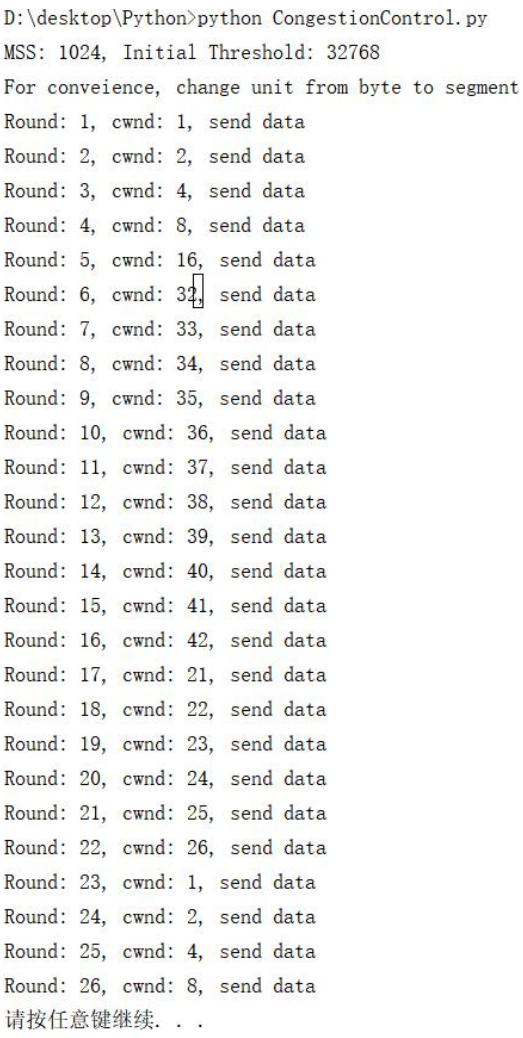
- C++


D:\desktop\C++\x64\Debug\C++.exe
MSS: 1024, Initial Threshold: 32768
For conveience, change unit from byte to segment
Round: 1, cwnd: 1, send data
Round: 2, cwnd: 2, send data
Round: 3, cwnd: 4, send data
Round: 4, cwnd: 8, send data
Round: 5, cwnd: 16, send data
Round: 6, cwnd: 32, send data
Round: 7, cwnd: 33, send data
Round: 8, cwnd: 34, send data
Round: 9, cwnd: 35, send data
Round: 10, cwnd: 36, send data
Round: 11, cwnd: 37, send data
Round: 12, cwnd: 38, send data
Round: 13, cwnd: 39, send data
Round: 14, cwnd: 40, send data
Round: 15, cwnd: 41, send data
Round: 16, cwnd: 42, send data
Round: 17, cwnd: 21, send data
Round: 18, cwnd: 22, send data
Round: 19, cwnd: 23, send data
Round: 20, cwnd: 24, send data
Round: 21, cwnd: 25, send data
Round: 22, cwnd: 26, send data
Round: 23, cwnd: 1, send data
Round: 24, cwnd: 2, send data
Round: 25, cwnd: 4, send data
Round: 26, cwnd: 8, send data
请按任意键继续. . .

• java



• Python



7. 实验总结

这个实验很简单，只是简单地模拟发送端，由于题目指出不必考虑接收确认，因此可以没有接收程序。发送端一直循环到结束轮次，一开始采用慢开始算法，cwnd 从 1 开始，当检测到超时的时候，就采用慢开始和拥塞避免。当接收到 3 个重复确认时，就采用快恢复。