南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络

实验 3：基于 UDP 服务设计可靠传输协议并编程实现

姓名：韩佳迅

学号：2012682

年级：2020 级

专业：计算机科学与技术

2022 年 12 月 20 日

目录

一、 实验要求

基于给定的实验测试环境，通过改变延迟时间和丢包率，完成下面 3 组性能对比实验：
（1）停等机制与滑动窗口机制性能对比；
（2）滑动窗口机制中不同窗口大小对性能的影响；
（3）有拥塞控制和无拥塞控制的性能比较。

二、 实验内容

(一) 实验所用的协议

在之前的实验中，已经完成了三种协议的实现。

1. 报文格式

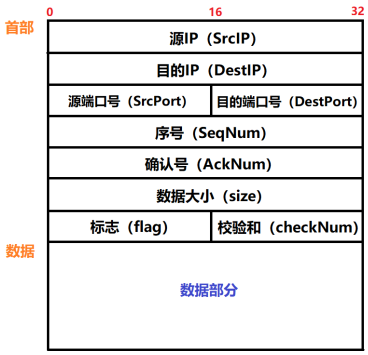

图 1: 报文格式

报文格式分为首部和数据部分：

首部： 1—4 字节：源 IP
5—8 字节：目的 IP
9—10 字节：源端口号
11—12 字节：目的端口号
13—16 字节：序号
17—20 字节：确认号
21—24 字节：数据大小
25—26 字节：标志
27—28 字节：校验和

数据： 剩余字节流是数据部分

并在报文段里设置校验和字段，发送时设置校验和，收到时检验校验和是否正确。若不正确，说明报文损坏。

2. 建立连接：三次握手

实现的协议在标准三次握手的基础上，进行了部分改进，最终实现的结构示意图如下：

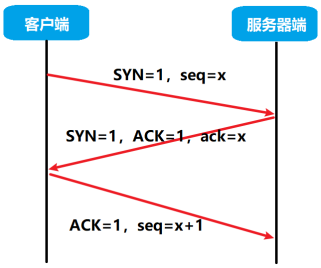

图 2: 三次握手

1. 客户端向服务器端发送数据包：

   SYN=1，seq=x

2. 服务器端向客户端回复数据包：

   SYN=1，ACK=1

   ack=x【这里对标准三次握手进行了更改：回复的 ack = 上一个包发来的 seq】

3. 客户端向服务器端发送数据包：

   ACK=1

   seq=x+1【序列号 +1】

3. 释放连接：四次挥手

协议在标准四次挥手的基础上，进行了部分改进，最终实现的结构示意图如下：

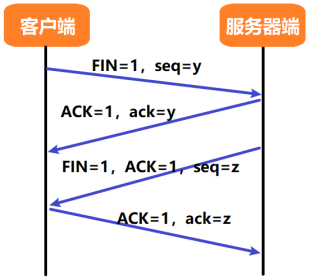

图 3: 四次挥手

1. 客户端向服务器端发送数据包：

   FIN=1，seq=y

2. 服务器端向客户端回复数据包：

   ACK=1，ack=y【回复的 ack = 上一个包发来的 seq】

3. 服务器端向客户端发送数据包：

   FIN=1，ACK=1

   seq=z

4. 客户端向服务器端发送数据包：

   ACK=1，ack=z【回复的 ack = 上一个包发来的 seq】

4. 停等机制

使用 ***rdt3.0*** 协议

- 发送端

  发送 ***pkt n*** 并计时，开始等待回复

  – 超时未收到 ***ack n***，重发并重新计时

  – 收到 ***ack m*** **(*m != n*)**，继续等待

  收到 ***ack n***

  继续发送下一个 ***pkt n+1***

- 接收端

  等待 ***pkt n***

  – 当收到重复的 pkt（***pkt m***，***m<n***），丢掉这个 pkt，并回复对应的 ***ack m***

  收到 ***pkt n***，回复 ***ack n***

  继续等待下一个 pkt

5. 滑动窗口机制：滑动窗口、累计确认、快速重传

协议基于 ***Go-Back-N(GBN)*** 协议，添加了三次快速重传，实现了滑动窗口、累计确认。

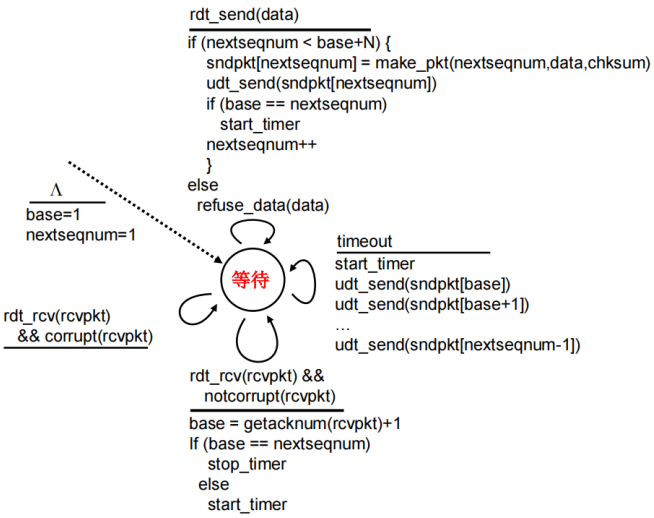

图 4: GBN 状态机

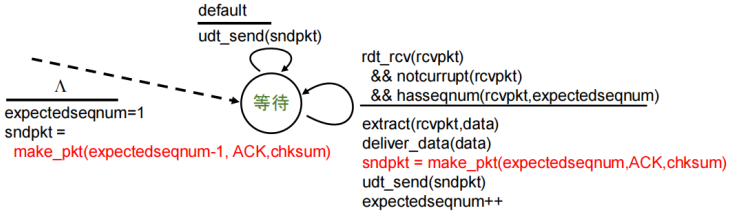

图 5: GBN 状态机

滑动窗口

- 设置窗口大小固定为 $CWND$。
- 设置两个指针，来控制窗口的滑动和下一个序号的发送：
  - $base$：基序号，是滑动窗口的开始位置，指向已发送为确认的第一个序号，或窗口的第一个序号
  - $nextseqnum$：指向下一个要发送的序号

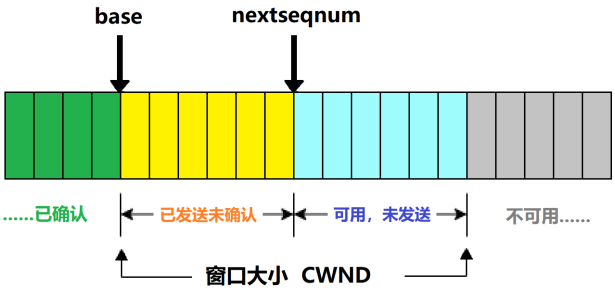

图 6: 滑动窗口

累计确认

- 发送端当发送端收到确认报文时：
  - 首先检查确认报文是否 $ack < base$，若小于，说明这是上一次发送的失序确认报文，不移动窗口
  - 若确认报文 $ack >= base$，说明这是对此窗口内的确认，$base$ 移动到 $ack+1$ 的位置，窗口右移
- 接收端
  - **理想情况**：接收端每接收到发来的报文，若该报文的序号**等于**接收端期待接收的报文序号，则回复一个确认报文，确认号 $ack = seq$，并接收该报文段，将其交付给上层应用。
  - **失序**：当接收端收到了**不等于**期待序号值的报文，则回复一个确认报文，确认号 ack = 接收端累计确认的最后一个报文号（也就是期待接收的报文序号的前一个值），并丢弃该报文段。

超时重传 滑动窗口每发送报文时，会设置定时器。当超时未收到报文时，会重传当前窗口内所有已经发送的报文，即 $base$ 到 $nextseqnum$ 的所有报文，而 $base$、$nextseqnum$ 位置不变。

快速重传 由于接收端每次收到失序的报文时，会回复 ack = 期待值的确认报文，因此当报文丢失或失序时，发送端会连续收到多个重复且冗余的 ACK 报文端。因此，实验中规定，当连续收到三次冗余的 ACK 报文时，可认为报文丢失，需要重传当前窗口内 $base$ 到 $nextseqnum$ 的所有报文。

6. 拥塞控制

协议使用 ***new reno*** 机制。

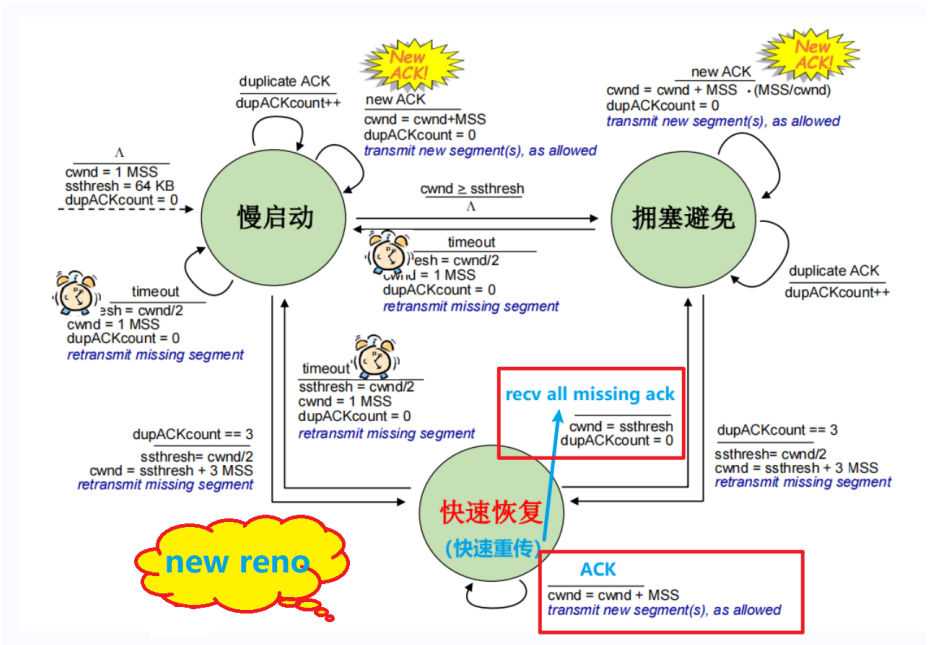

图 7: new reno

在 ***new reno*** 协议中，存在三种状态：慢启动、拥塞避免、快速恢复（快速重传）。

- **慢启动**

  在连接刚建立时，为了试探网络的拥塞情况，先只发送一个报文段。

  慢启动开始后，再逐渐增大拥塞窗口 cwnd，每经过一个传输轮次，cwnd 加倍（即 cwnd 的值随传输轮次线性增长）这样可以使分组注入网络的速率更加合理。

- **拥塞避免**

  拥塞避免状态的思路是让 cwnd 缓慢增大（相较于慢启动和快速恢复阶段）。

- **快速恢复（快速重传）**

  当发送端收到三个冗余 ACK：

  – 说明网络可能有报文丢失，此时发送端会重发尚未确认的报文，而不用等待计时器超时。

  – 也说明当前网络可能存在拥塞，为了预防拥塞，阈值会被缩减为 cwnd 的一半，同时 cwnd 也会缩小

  在此阶段，如果还能收到 ACK，说明网络还没有发生严重拥塞，因此还可以适当提高窗口 cwnd 大小

注： 具体的状态转换规则见图7。

New reno 的改进：***Reno*** 快速恢复算法中，发送方只要收到一个新的 ACK 就会退出快速恢复状态而进入拥塞避免阶段，***New reno*** 算法中，只有当所有丢失的包都重传并收到确认后才退出快速恢复状态。

三、 实验结果及分析

在下面的三组对比实验中，需要控制以下变量相同：

1. 报文格式：保证每个报文的格式和大小相同
2. 超时时间：计时器超过一段时间则判定为超时
3. IP 和端口号
4. 测试文件

注：

1. 实验数据使用四个测试文件多次实验的平均时延和平均吞吐率
2. 实验中存在两个原有的变量——延时、吞吐率，下面的实验针对这两个变量进行了测试：
   - 在延时为 0 时改变丢包率
   - 在丢包率为 0 时改变延时

(一) 实验一：停等机制与滑动窗口机制性能对比

在控制变量的条件下，测得的结果如表1、表2和图8、图9。分析实验结果：

- 滑动窗口机制总体比停等机制有更高的性能：时延更短、吞吐率更大
  - 滑动窗口机制可以同时发送多条消息，而停等机制一次只能发送一个报文。因此，滑动窗口对信道的利用率更高，传输时延更低，吞吐率更大。
- 丢包率为 0，改变时延，滑动窗口存在加速效果，但不明显。
  - 分析打印的日志可知，这主要是因为，当发送一段时间后且没有丢包，滑动窗口会达到动态平衡，每当窗口收到一个确认报文，就会有一个空闲位置，也会立即发送新报文段，所以性能提升不明显。
- 时延为 0，改变丢包率，滑动窗口的加速效果非常明显。这是因为：
  - 采用流水线发送机制。
  - 有丢包时，滑动窗口机制存在三次快速重传，可以避免超时等待的时间，当收到三次冗余的 ACK 即可重新发包。

延时 0ms		**丢包率 0%**	**丢包率 1%**	**丢包率 3%**	**丢包率 5%**	**丢包率 7%**
停等机制	时延 s	2.3	35	102	160.5	232
	吞吐率 B/s	2.69E+06	1.68E+05	5.78E+04	3.69E+04	2.54E+04
滑动窗口	时延 s	2	6.5	7	7	9
	吞吐率 B/s	2.97E+06	9.38E+05	8.43E+05	8.43E+05	6.55E+05

表 1

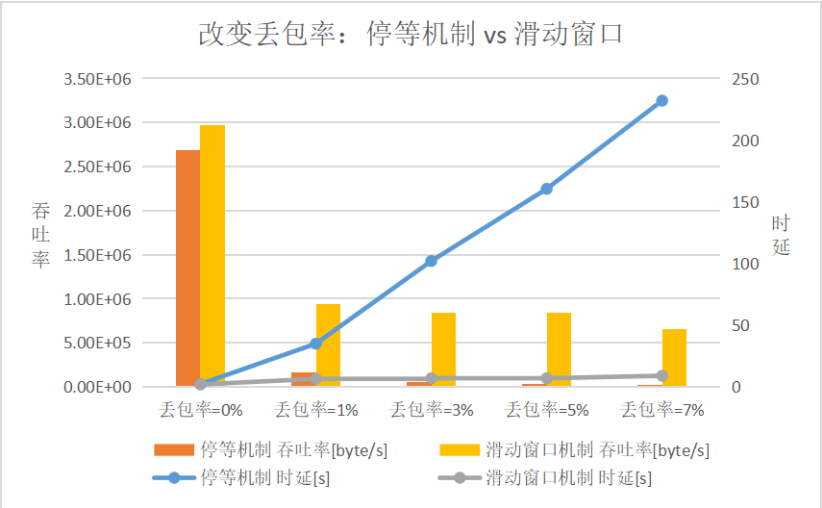

图 8: 改变丢包率：停等机制 vs 滑动窗口

丢包率 0%		**延时 0ms**	**延时 5ms**	**延时 10ms**	**延时 15ms**	**延时 20ms**
停等机制	时延 s	2.3	11	18.5	19	22.5
	吞吐率 B/s	2.69E+06	5.36E+05	3.28E+04	3.10E+05	2.68E+05
滑动窗口	时延 s	2	10	14	18	22
	吞吐率 B/s	2.97E+06	5.84E+05	4.21E+05	3.28E+04	2.68E+05

表 2

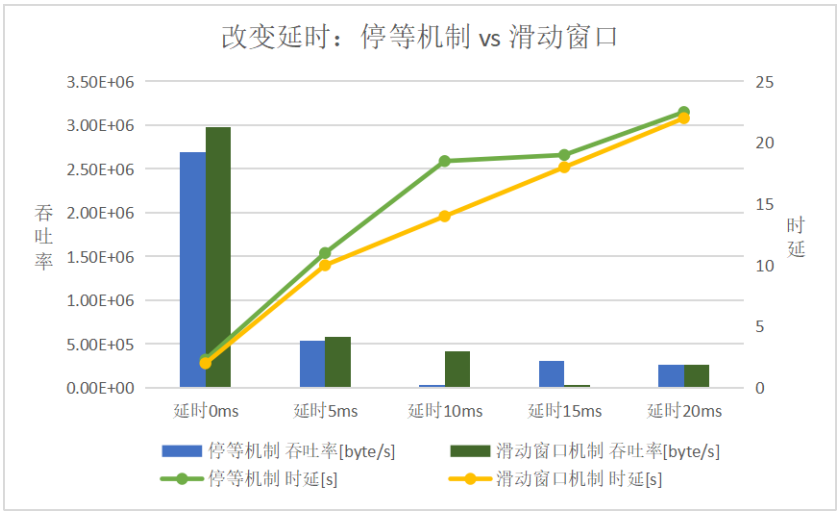

图 9: 改变延时：停等机制 vs 滑动窗口

(二) 实验二：滑动窗口机制中不同窗口大小对性能的影响

在控制变量的条件下，测得的结果如表3、表4和图10、图11。分析实验结果：

- 从总体上看，在不同的丢包率和延时下，窗口越大，时延越低、吞吐率越大

因为窗口越大，可以同时发送的报文就越多，有利于更加充分的利用信道，降低时延，提高吞吐率。

- 丢包率为 0 时，只改变延时，滑动窗口的加速比未达到窗口大小比。
  - 这是因为当滑动窗口达到平衡时，窗口内一收到一个确认报文，就会有一个空闲位置，也会立即发送新报文段，所以达到一个动态平衡的状态。（这主要是因为在本程序中，没有上层应用缓存报文的时延，导致一有空位就可以发送，而在实际中，存在上层应用产生和给派报文的过程，以及窗口内的缓存时间）。
- 窗口大小为 1 时的效果跟停等机制的效果相近
- 窗口小于等于 3 时，无法快速重传，导致效果与停等机制的效果相近。
- 三次快速重传使得滑动窗口加速效果更明显，且在丢包率高时更明显

延时 0ms		**丢包率 0%**	**丢包率 1%**	**丢包率 3%**	**丢包率 5%**	**丢包率 7%**
窗口 =1	时延 s	2.3	33	103	162	225
	吞吐率 B/s	2.69E+06	1.69E+05	5.78E+04	3.69E+04	2.55E+04
窗口 =3	时延 s	2.5	37	110	162	227
	吞吐率 B/s	2.69E+06	1.66E+05	5.83E+04	3.69E+04	2.55E+04
窗口 =5	时延 s	2	6.5	7	7	9
	吞吐率 B/s	2.97E+06	9.38E+05	8.43E+05	8.43E+05	6.55E+05
窗口 =7	时延 s	2	6	7	7.5	9
	吞吐率 B/s	2.97E+06	9.73E+05	8.43E+05	8.43E+05	6.55E+05
窗口 =9	时延 s	1.8	5.4	6.5	7.5	9
	吞吐率 B/s	2.98E+06	9.92E+05	9.38E+05	8.43E+05	6.55E+05

表 3

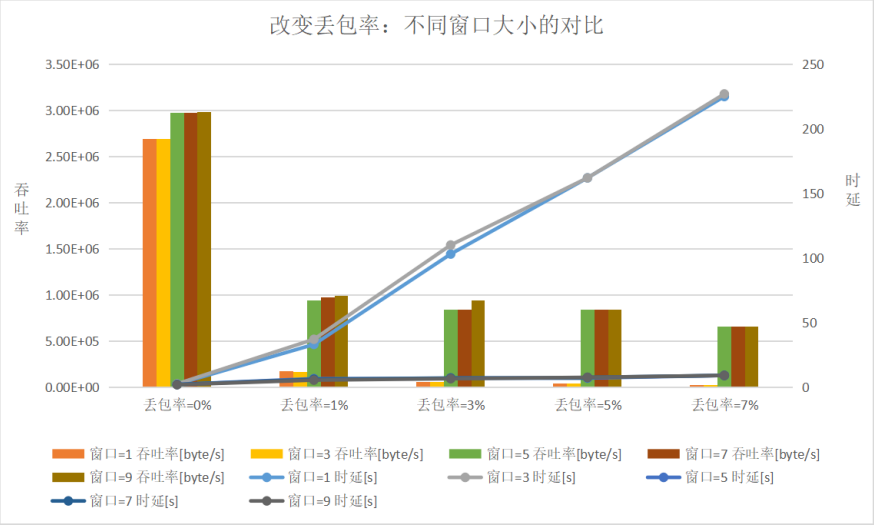

图 10: 改变丢包率：不同窗口大小的对比

丢包率 0%		延时 0ms	延时 5ms	延时 10ms	延时 15ms	延时 20ms
窗口 =1	时延 s	2.3	12	18	20.5	23
	吞吐率 B/s	2.69E+06	5.35E+05	3.28E+04	3.09E+05	2.68E+05
窗口 =3	时延 s	2.5	12	18.5	20	24
	吞吐率 B/s	2.69E+06	5.35E+05	3.28E+04	3.09E+05	2.67E+05
窗口 =5	时延 s	2	10	14	18	22
	吞吐率 B/s	2.97E+06	5.84E+05	4.21E+05	3.28E+04	2.68E+05
窗口 =7	时延 s	2	10.5	14	16	22
	吞吐率 B/s	2.97E+06	5.84E+05	4.21E+05	3.31E+04	2.69E+05
窗口 =9	时延 s	1.8	9	16	16	19.5
	吞吐率 B/s	2.98E+06	5.91E+05	4.18E+05	3.31E+04	2.82E+05

表 4

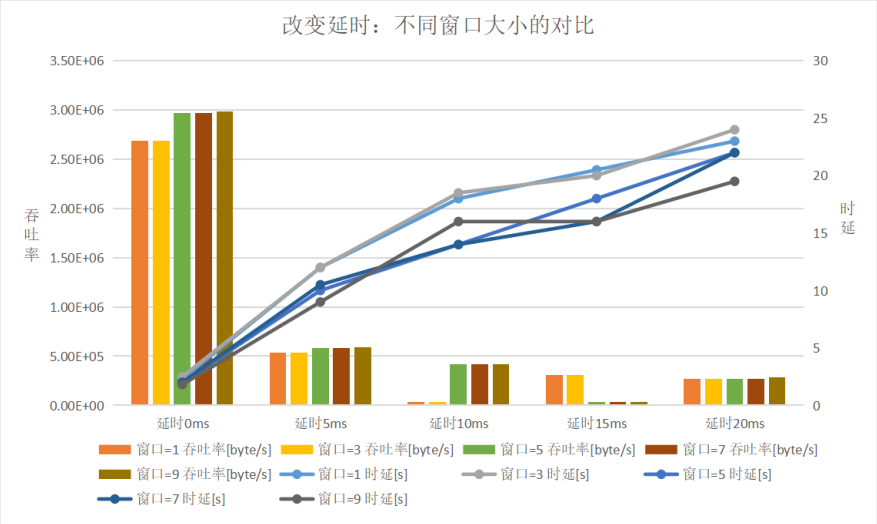

图 11: 改变延时：不同窗口大小的对比

(三) 实验三：有拥塞控制和无拥塞控制的性能比较

在控制变量的条件下，测得的结果如表5、表6和图12、图13。分析实验结果：

- 延时为 0，只改变丢包率时，当丢包率较低时，有拥塞控制的效果较好；而当丢包率升高时，有拥塞控制的效果较低于无拥塞控制的情况。
  - 丢包率较低时，程序停留在初始的慢启动的状态更久，阈值和窗口大小更大（大于固定窗口大小），因此效果较好
  - 当丢包率升高，程序离开慢启动时的窗口较小，阈值较低。而后经常在慢启动、拥塞控制、快速恢复之间切换，导致窗口大小一直较小，且丢包率也会影响重传和恢复，导致有拥塞控制的反而性能较差。
- 丢包率为 0 时，只改变延时。

– 从理论上来说，由于不存在丢包，有拥塞控制的平均窗口会更大些，性能应更好。而实际操作时，由于 socket 编程函数的缓冲区不够或电脑自身原因，当连续发送较多报文时，存在部分丢包现象，因此可能存在有拥塞控制的性能较弱的状况。

- 由于本实验的路由器程序在丢包时，非集中丢包，而是每隔一段时间丢一个，导致拥塞控制算法在不同网络拥塞程度下对窗口大小的改变优势没有完全体现。但是在真实的复杂网络环境下，相比于没有拥塞控制或者 reno 算法来说，使用 new reno 拥塞控制算法会有更好的性能。

延时 0ms		**丢包率 0%**	**丢包率 1%**	**丢包率 3%**	**丢包率 5%**	**丢包率 7%**
无拥塞控制	时延 s	2	6.5	7	7	9
	吞吐率 B/s	2.97E+06	9.38E+05	8.43E+05	8.43E+05	6.55E+05
有拥塞控制	时延 s	1.5	5.5	11	13	15.5
	吞吐率 B/s	3.38E+06	1.69E+06	5.23E+05	4.14E+05	3.39E+05

表 5

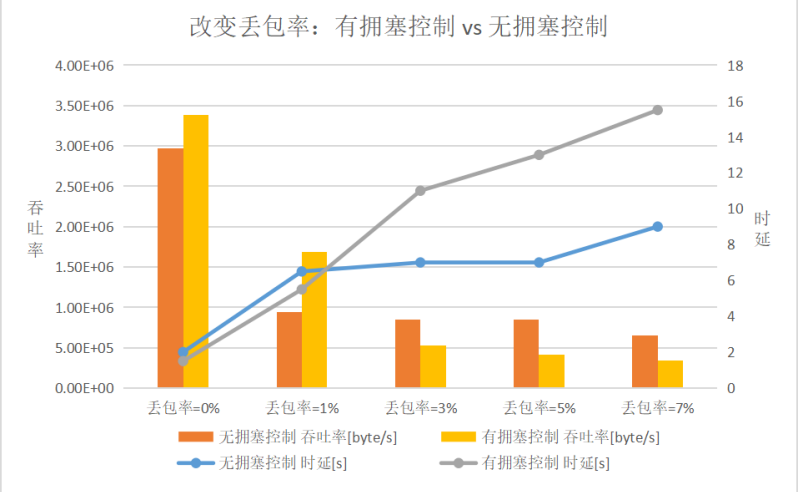

图 12: 改变丢包率：有无拥塞控制的对比

丢包率 0%		**延时 0ms**	**延时 5ms**	**延时 10ms**	**延时 15ms**	**延时 20ms**
无拥塞控制	时延 s	2	10	14	18	22
	吞吐率 B/s	2.97E+06	5.84E+05	4.21E+05	3.28E+04	2.68E+05
有拥塞控制	时延 s	3	8.5	12	23	27
	吞吐率 B/s	2.13E+06	5.98E+05	5.17E+05	2.33E+05	1.05E+05

表 6

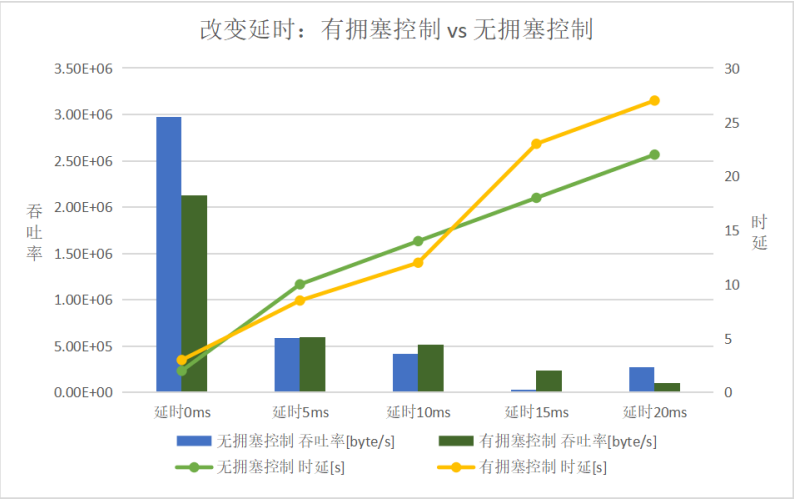

图 13: 改变延时：有无拥塞控制的对比