南開大學
Nankai University

计算机学院

计算机网络实验报告

性能测试

姓名：赵一名

学号：2013922

专业：计算机科学与技术

2022 年 12 月 28 日

目录

1 概述

在本次实验中 [1]，我进行了三组不同的测试，分别为（1）停等机制与滑动窗口机制性能对比。(2) 滑动窗口机制中窗口大小对性能的影响。（3）有拥塞控制和无拥塞控制算法的比较。实验中用到的测试程序为前三次实验中编写的程序。

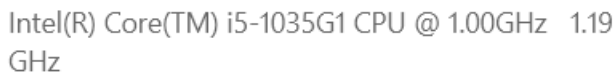
1.1 实验中时间的测量

实验中关于运行时间的测量，我都采用了高精度计时函数 $gettimeofday()$。

```
1
2  void time_measure()
3  {
4      timeval *start=new timeval();
5      timeval *stop=new timeval();
6      double durationTime=0.0;
7      init();
8      gettimeofday(start,NULL);
9      col_major();
10     gettimeofday(stop,NULL);
11     durationTime =stop->tv_sec*1000+double(stop->tv_usec)/1000
12                  -start->tv_sec*1000-double(start->tv_usec)/1000;
13     cout << "duration time: " << double(durationTime) << " ms" << endl;
14 }
```

1.2 运行环境

下面是我电脑的 cpu 配置



1.3 关于测试的一些说明

测试用的程序是实验 3.1，3.2，3.3 中编写的程序，测试文件我选用给定的文件中的 $1.jpg$ 这张图片，这张图片的大小为 $1857353btyes$

图 1.1: 图片 1

同时，考虑到文件是在本地进行传输，所以每次传送时数据包的中的数据段大小也和传送速率有着极大的关系，但这并不是本次实验的重点，所以我直接将数据段的大小设为 $4086bytes$。

除此之外，考虑到 $printf$ 语句是相当耗时的，所以我去掉了代码中打印相关信息的代码，以保证测试的数据更加准确。

2 停等机制和滑动窗口机制的对比

首先进行停等机制与滑动窗口机制的对比，我在测试中使用的是实验 3.1 与实验 3.2 的代码，其中 3.2 的滑动窗口机制的实现我采用了 GBN 协议，滑动窗口的大小我设计为 5 个最大报文段的大小。

首先比较传输速率随路由器时延的变化，在测量过程中丢包率设为 0%

路由器时延/ms	停等 (Mbit/s)	滑动窗口 (Mbit/s)
0	3.34	4.21
20	0.66	0.72
40	0.43	0.12
80	0.25	0.03
160	0.11	0
320	0	0

表 1: 吞吐量随路由器时延的变化

根据测量得到的数据做出折线图如下图所示：

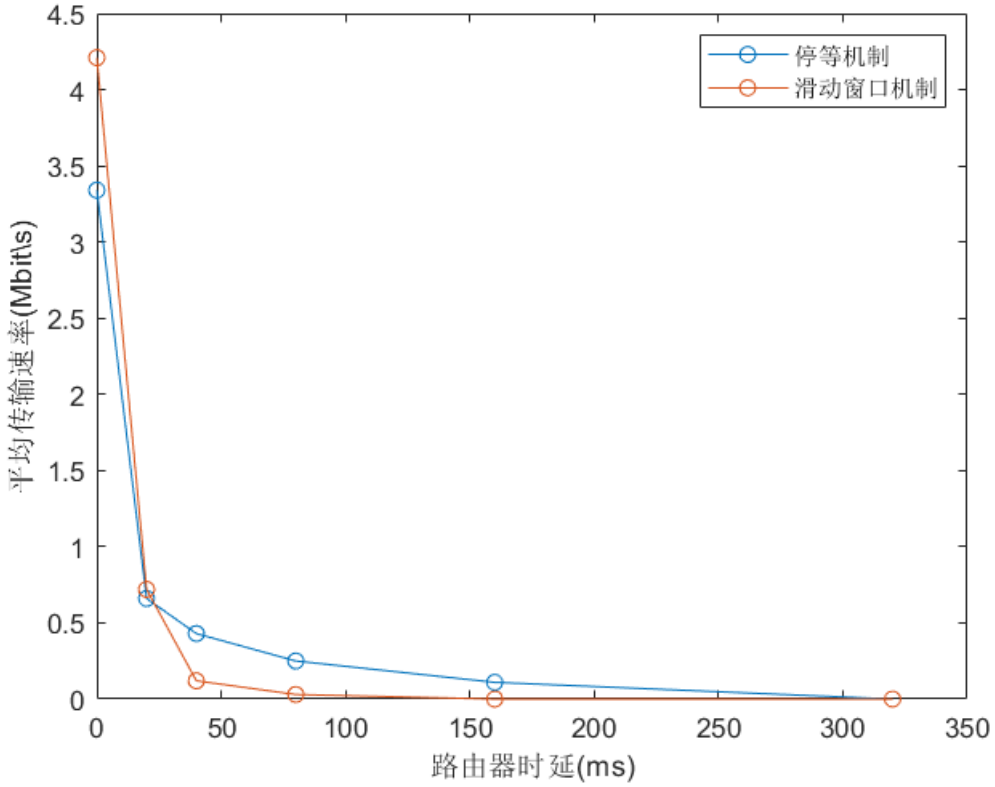


图 2.2: 吞吐量随路由器时延的变化

可以看到，在滑动窗口的大小为 12 个最大报文段的前提下，当路由器时延为 0 时，滑动窗口的吞吐量明显大于停等机制的吞吐量，其性能优化比为

$$P = 4.21/3.34 = 126.0\%$$

我认为这有两方面原因，一方面是滑动窗口机制中采用了流水线类型的发送机制，这要快于停等机制，另一方面是我在实现 3.2 的代码时进行了一定的优化。

同时也可以发现，当网络延时较大时，两者表现均急剧下降，这是由于我在程序中设定的重发计时器的时间间隔为固定的 200ms 所以实际中当时延大于 200ms 会导致传输速率非常之慢。

接下来比较传输速率随丢包率的变化，在测量过程中路由器时延设为 0ms，以下数据都是重复五次取平均值的结果。

丢包率/ms	停等 (Mbit/s)	滑动窗口 (Mbit/s)
0	3.34	4.21
5	1.14	0.94
10	0.79	0.31
15	0.64	0.11
20	0.47	0.05
25	0.36	0

表 2: 吞吐量随丢包率的变化

根据测量得到的数据做出折线图如下图所示：

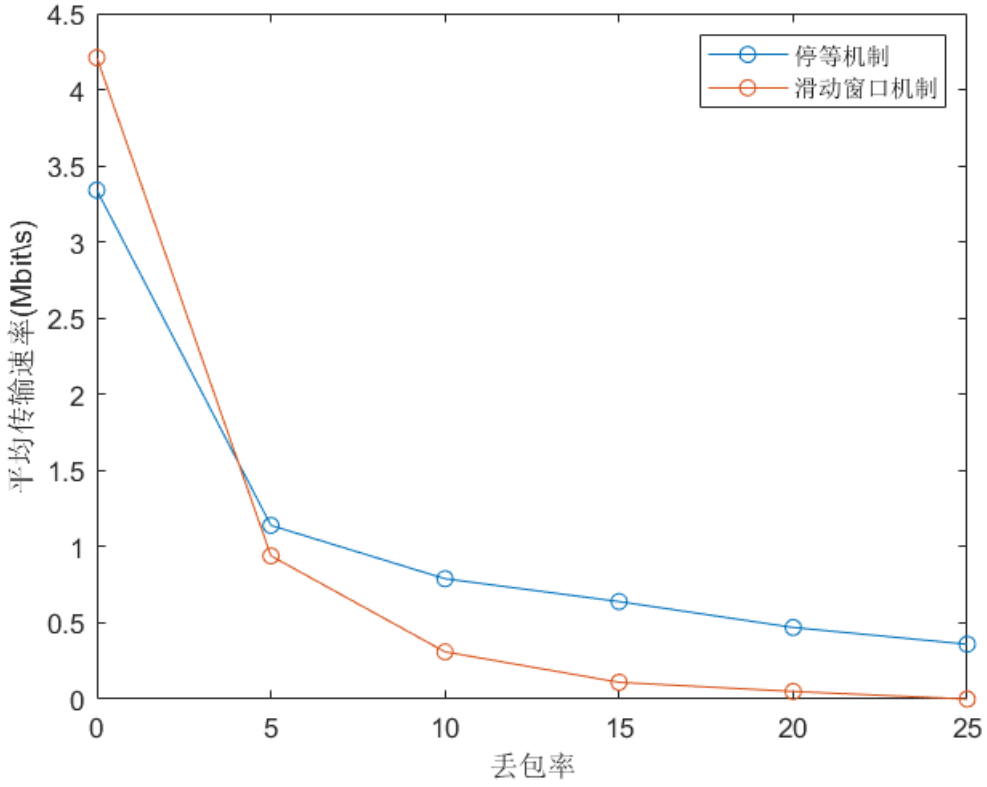


图 2.3: 吞吐量随丢包率的变化

从上图中可以看到，当丢包率小于 5 时，滑动窗口机制是优于停等机制的，但当丢包率增大时，可以发现丢包率的增大对于滑动窗口机制性能的破坏是摧毁性的，其性能急剧下降，我认为这是由于 3.2 中我采用了 *GBN* 协议，大量的重传导致上述结果。

3 滑动窗口机制中窗口大小对性能的影响

对于这一部分的测试，我仍采用了实验 3.2 中设计的程序。

参考上一节的实验，我将路由器时延选为 20ms，丢包率选为 5% 来进行这一节的测试，我分别进行了窗口大小为 5，10，15，20，50 的几组测试，测试结果如下表

滑动窗口大小	无延迟与丢包 (Mbit/s)	有延迟与丢包 (Mbit/s)
5	4.21	0.82
10	5.01	1.47
15	4.63	1.62
20	4.38	1.32
50	4.49	0.53

表 3: 吞吐量随滑动窗口大小的变化

根据测量得到的数据做出折线图如下图所示：

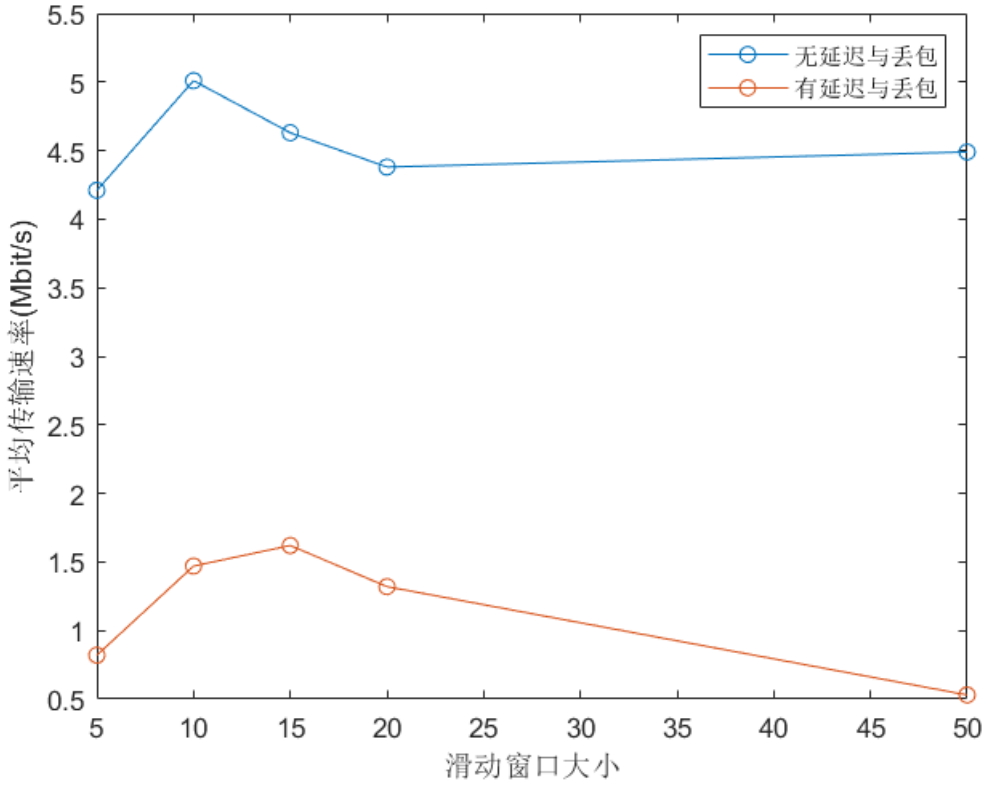


图 3.4: 吞吐量随滑动窗口大小的变化

从测试结果可以看到，无论是无延迟与丢包的情况还是有延迟与丢包的情况，随着滑动窗口的大小增大，两者的性能都是一个递增又递减的过程。而且可以发现，有延迟与丢包的情况下这个变化的趋势更加明显，尤其是当窗口大小为 50 时其性能呈断崖式下降。

我认为出现上述现象的原因是，在初期随着滑动窗口的数量增加，发送方传输数据的并发度增加，因此表现出良好的递增趋势。但是当滑动窗口增大到一定值时，因为窗口数过大，一次性发送太多的数据包可能会导致丢包现象的发生（导致超时重传），这是非常耗时，得不偿失的，因此窗口数增大到一定值时，性能急剧下降。

4 有拥塞控制和无拥塞控制算法的比较

对于这一节的测试，我采用了实验 3.2 与实验 3.3 中设计的程序。

滑动窗口的大小我选为了上一节中表现最好的一组滑动窗口（大小为 10），且有拥塞控制的一方采用了 *reno* 算法，接下来我分别测试了随着时延变化与丢包率变化时有拥塞控制与无拥塞控制的吞吐量。

首先比较传输速率随路由器时延的变化，在测量过程中丢包率设为 0%。考虑到第 2 节的测试结果，我仅对的路由器延时小于 50ms 的情况做了测试。

路由器时延/ms	无拥塞控制 (Mbit/s)	有拥塞控制 (Mbit/s)
0	4.21	4.32
10	1.92	2.28
20	0.72	1.54
30	0.37	0.52
40	0.12	0.29
50	0.09	0.17

表 4: 吞吐量随路由器时延的变化

根据测量得到的数据做出折线图如下图所示:

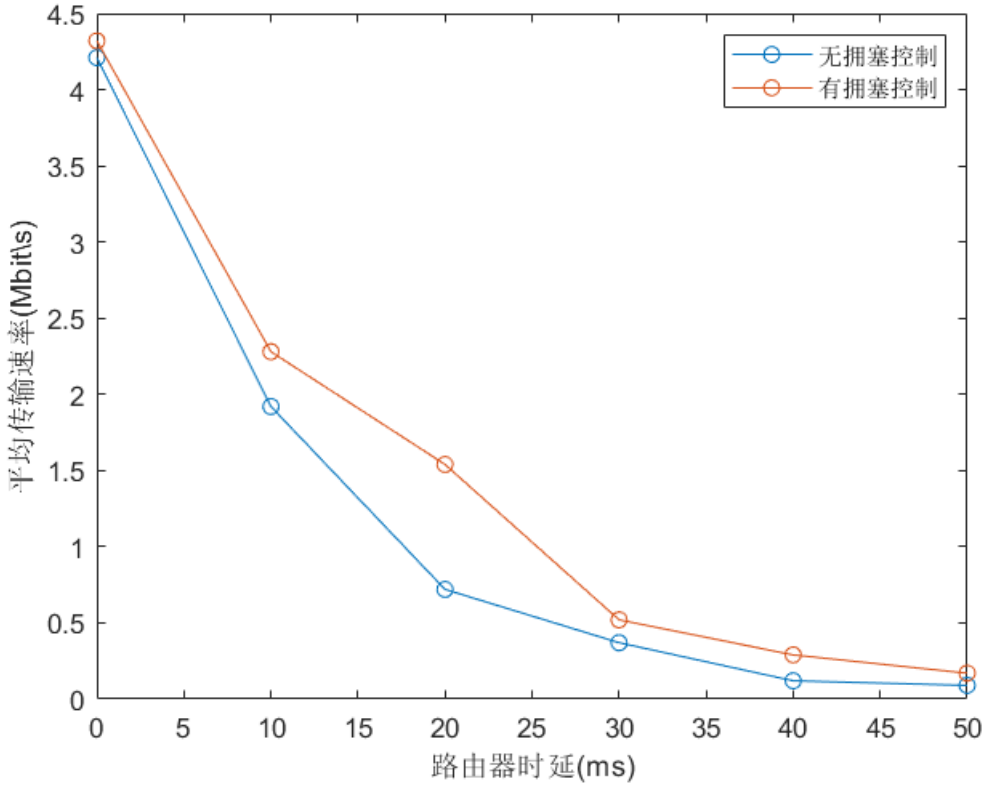


图 4.5: 吞吐量随路由器时延的变化

从测试结果中可以发现，加入了拥塞控制算法后的性能明显优于优于无拥塞控制算法的一方，我认为主要原因是应为我在编写 3.3 的代码时在 3.2 的基础上做了一定的改进，因为这次测试中的延时并未达到我设定的计时器的时间。

接下来比较传输速率随丢包率的变化，在测量过程中路由器时延设为 0ms，以下数据都是重复五次取平均值的结果。

丢包率/ms	无拥塞控制 (Mbit/s)	有拥塞控制 (Mbit/s)
0	4.21	4.32
5	0.94	2.33
10	0.31	1.29
15	0.11	0.90
20	0.05	0.38
25	0	0

表 5: 吞吐量随丢包率的变化

根据测量得到的数据做出折线图如下图所示:

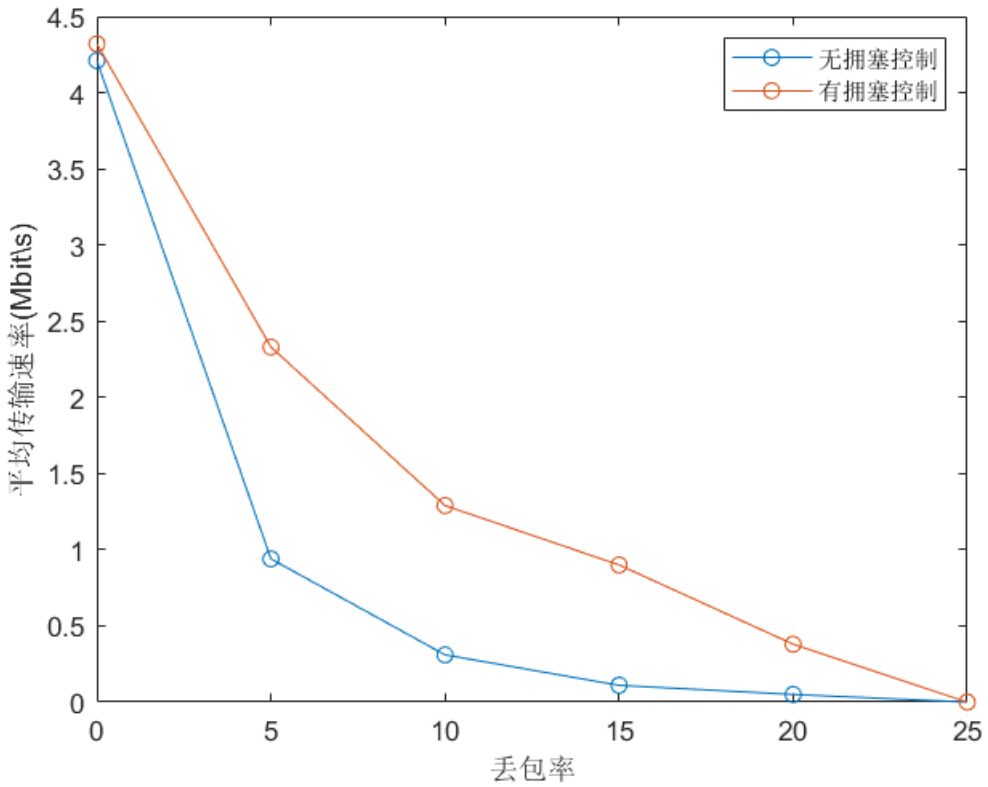


图 4.6: 吞吐量随丢包率的变化

从测试结果中可以发现，加入了拥塞控制算法后的性能明显优于优于无拥塞控制算法的一方，这是因为为拥塞控制的一方采用 *reno* 算法，其会探测接收方的网络拥堵情况从而将窗口设置为合适的大小，并将自身状态在慢启动，拥塞避免和快速恢复三个状态之间来回切换。

除此之外,产生丢包现象时,有拥塞控制的这一方中我还实现了快速恢复策略(收到三个 *duplicate ack* 就重发数据包)，而无拥塞控制的一方我并没有加入这个功能，所以在面对较大的丢包率时，有拥塞控制的一方还能保证一定的传输速率，而无拥塞控制的这一方性能快速下降。

参考文献

[1] 库罗斯. 计算机网络: 自顶向下方法. 机械工业出版社, 2009.