
南开大学

Nankai University

# 南　　开　　大　　学

## 计 算 机 学 院

**计算机网络实验报告**

---

# 基于 UDP 服务设计可靠传输协议 Part4

---

2011763 黄天昊

年级：2020 级

专业：计算机科学与技术

指导教师：徐敬东、张建忠

2022 年 12 月 28 日

# 摘要

在基于 UDP 服务实现可靠传输协议 Part4 中，对于传输过程中细节问题（停等机制与滑动窗口机制、滑动窗口机制中不同窗口大小、有拥塞控制和无拥塞控制）的对比分析为本部分的重点内容，分析中主要以丢包率以及延迟时间作为变量，以文件传输的吞吐率和时延作为性能测试指标，除了对不同机制的性能对比分析之外，本实验报告还会针对一些因为本机传输本机的特殊实验环境出现的现象进一步进行分析。



**关键字：UDP 可靠数据传输; 丢包率; 延迟时间; 吞吐率; 时延**


# 目录


**一、 停等机制与滑动窗口机制性能对比** **1**

**二、 滑动窗口机制中不同窗口大小对性能的影响** **3**

**三、 有拥塞控制和无拥塞控制的性能比较** **4**

**四、 总结** **5**

# 一、 停等机制与滑动窗口机制性能对比

对于停等机制，在每次发送一个数据分组之后都会一直等待接收端返回相应的 ACK 响应才会继续发送下一个数据分组，而滑动窗口则会将窗口内部的所有数据分组都进行发送，当窗口的 base 也就是按顺序下一个发送等待接收的数据分组，GBN 并不带有缓冲区所以窗口内的所有数据分组都默认丢失，而 SR 则带有缓冲区。GBN 只有一个计时器所以当超时时将窗口内的所有数据分组都进行重传，而 SR 则是对窗口内部每一个数据都设置一个计时器，如果对应计时器超时那么重传对应分组。

这里使用默认的窗口大小，也就是 10 进行实验，为了避免实验的偶然性，对每一个实验进行五次的时延和吞吐率的测试，取其平均值为最终结果，数据分组的大小为 4096 字节。

首先就是不设置任何延迟时间和丢包率进行实验测试（均使用测试文件中的 1.jpg 文件进行测试，1.jpg 的大小为 1857353 字节）。

对于每次实验进行五次重复实验，取其平均值作为最终结果，以避免偶然情况的发生，最后一行为平均值结果，以下为统计数据：

| | 吞吐率(Bytes/ms) | | 时延(ms) | |
|---|---|---|---|---|
| 次数 | 停等机制 | 滑动窗口 | 停等机制 | 滑动窗口 |
| 1 | 122.283 | 3309.86 | 15164 | 563 |
| 2 | 108.065 | 3570.91 | 17535 | 520 |
| 3 | 109.365 | 3501.62 | 17321 | 526 |
| 4 | 115.654 | 3268.94 | 16352 | 568 |
| 5 | 114.12 | 3251.45 | 16542 | 564 |
| 平均 | 113.8974 | 3380.556 | 16582.8 | 548.2 |

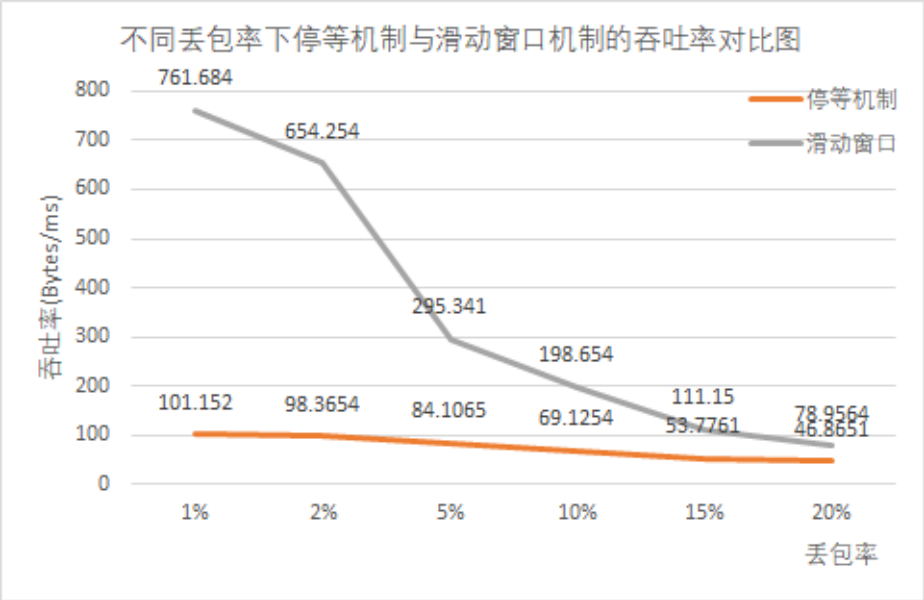
可以看到如果在没有任何延迟时间以及丢包率的情形下，滑动窗口机制的传输效率与停等机制相比大大提升了（窗口大小为 5，显然本机传输本机 5 的大小并不会因为数据分组过多而导致丢包）。从结果的数值停等机制为 16582.8ms，而滑动窗口仅为 548.2ms，如果没有丢包率和延迟时间的影响，滑动窗口的效率几乎为停等机制的 31 倍，我认为这里差距悬殊的原因主要是由于实验是本机向本机传输文件，没有延迟时间，又没有丢包率，那么最耗时间的就是 UDP 向下封装发送的过程了，停等机制发送时需要经历这样的阶段，接收时还要等待这样一个阶段，而滑动窗口则是将这一部分时间重叠了不少（不管是单线程亦或是多线程），所以能够有大幅度的效率提升。

接下来我们来看看在丢包率不断增大的情形下两种机制的表现。同样的对每次实验进行五次重复实验取平均值，如下：

| | 吞吐率(Bytes/ms) | | 时延(ms) | |
|---|---|---|---|---|
| 丢包率 | 停等机制 | 滑动窗口 | 停等机制 | 滑动窗口 |
| 1% | 101.152 | 761.684 | 18741 | 2441 |
| 2% | 98.3654 | 654.254 | 19841 | 2816 |
| 5% | 84.1065 | 295.341 | 20954 | 6258 |
| 10% | 69.1254 | 198.654 | 26814 | 9974 |
| 15% | 53.7761 | 111.15 | 34510 | 16842 |
| 20% | 46.8651 | 78.9564 | 39451 | 24405 |

以上实验设置的计时器的超时重传时间均为 200ms，以控制变量。因为在实际的传输过程中，最大的丢包率也就在 5% 左右，所以这里实验之中最大丢包率到 20% 就截止了。

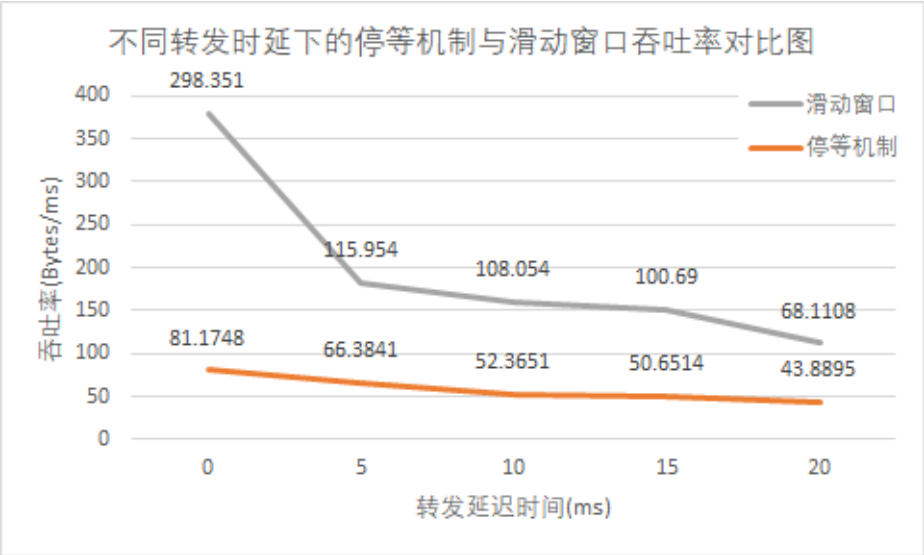
将停等机制与滑动窗口进行对比可以得到结果如下图：




可以看到滑动窗口收到了丢包率的极大影响，当丢包率达到 20% 时基本与停等机制的效率相同，因为滑动窗口还要重新发送窗口内的数据分组，可以设想如果丢包率进一步扩大很有可能出现滑动窗口的效率更低的情形，因为重传的过程之中也有可能继续丢包。

接下来我们继续进行延迟时间的实验，固定丢包率为接近实际的 5% 的丢包率，改变延迟时间仍然采用五次重复实验，最终使用平均值作为结果。

| 延迟时间 (ms) | 吞吐率(Bytes/ms) | | 时延(ms) | |
|---|---|---|---|---|
| | 停等机制 | 滑动窗口 | 停等机制 | 滑动窗口 |
| 0 | 81.1748 | 298.351 | 22874 | 6324 |
| 5 | 66.3841 | 115.954 | 29874 | 16845 |
| 10 | 52.3651 | 108.054 | 36541 | 19874 |
| 15 | 50.6514 | 100.69 | 39541 | 23054 |
| 20 | 43.8895 | 68.1108 | 43658 | 28412 |



可以看到除了 0ms 到 5ms 时传输效率有一个明显的变化，其实无论是在 5ms 还是 20ms 时的滑动窗口的吞吐率差的绝对值其实是类似的，是因为大家的延迟时间是相同的，所以发送每个数据分组到达的时间其实是差不多的，但是滑动窗口可能会重传更多的数据分组而导致有更长的时延，所以这时的吞吐率的斜率较为平缓。

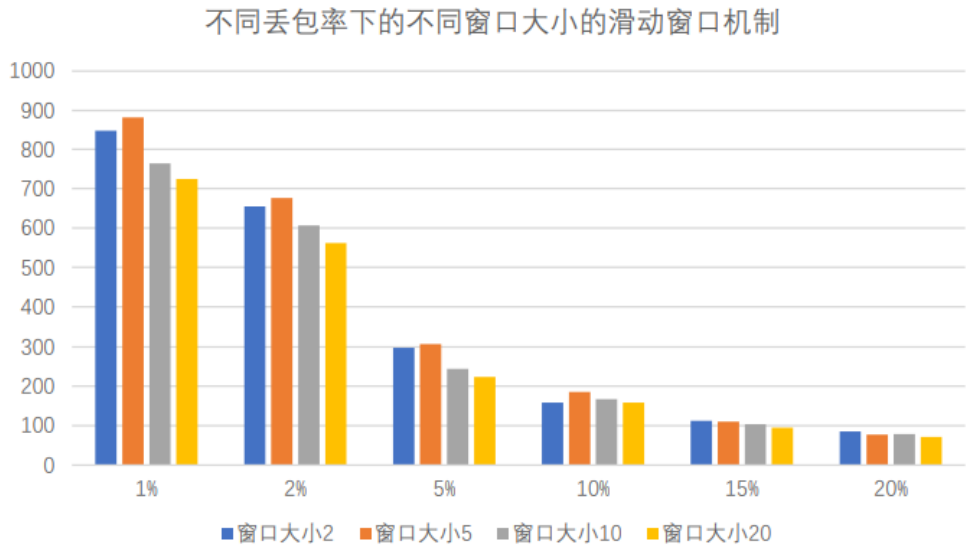
从 0ms 到 5ms 的斜率较大，是因为此时还是 UDP 向下封装传输占据了大部分的时间，而停等机制需要一直等 ACK 响应的原因，该原因与前一部分相同。至此，停等机制与滑动窗口机制的分析基本完成，我们能够成功发现，相较于停等机制如果采取大小为 5 的窗口，10% 的丢包率能够有 5ms 延迟时间，能够有接近 3 倍的传输效率提升。

## 二、 滑动窗口机制中不同窗口大小对性能的影响

接下来这一部分的实验分析主要是针对在滑动窗口中的不同的滑动窗口大小来进行对比分析的。滑动窗口太小，无法充分发挥滑动窗口的优势，快速地传输文件，而滑动窗口太大又会过多占用硬件资源 (收发两方的缓冲区过大)，并且由于 router 缓冲区固定大小，会因为缓冲区已满丢弃多余的分组而导致性能下降，因此选择一个合适的滑动窗口大小是一个值得我们讨论的话题。

接下来设置不同的滑动窗口的大小，来观察不同滑动窗口的大小在不同的丢包率和不同的延迟时间下的表现是怎样的。

| 丢包率 | 不同窗口大小对应的吞吐率(Bytes/ms) | | | |
|---|---|---|---|---|
| | 2 | 5 | 10 | 20 |
| 0% | 3148.06 | 3340.98 | 3468.69 | 3298.35 |
| 1% | 848.65 | 862.54 | 816.36 | 723.984 |
| 2% | 652.41 | 656.312 | 609.31 | 561.697 |
| 5% | 278.35 | 327.065 | 246.64 | 223.697 |
| 10% | 169.245 | 185.264 | 169.34 | 158.645 |
| 15% | 112.356 | 112.654 | 103.64 | 94.6591 |
| 20% | 85.3412 | 77.3589 | 79.3411 | 71.6412 |



在丢包率小于等于 10% 的时候，对比窗口大小为 2 和窗口大小为 4 的数据，可以发现此时增大数据窗口能够获得一定的传输效率提升，但是我们可以发现窗口大小超过 10 之后反而没有获得传输效率的提升，这是因为窗口大小变大之后，那么每次丢包需要重传的数据分组就会增加(对于 GBN 来说)，重传的数据分组量增加就带来更多的时间消耗，并且多重传的数据分组也会带来多的丢包情形，从而导致传输效率的下降，也就是说其实不同的丢包率下的最佳的滑动窗口大小是不相同的。

其实，我们也可以看到，当丢包率大于 10% 时，不同窗口大小的传输效率已经几乎相同了，此时因为丢包而产生的时间消耗占据了主导地位。

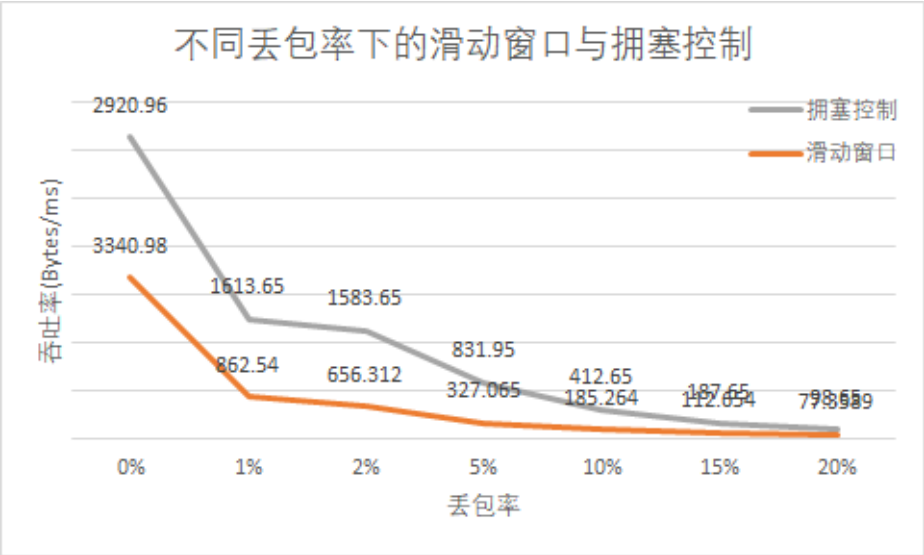
# 三、 有拥塞控制和无拥塞控制的性能比较

我们从滑动窗口的分析之中已经可以得知，其实滑动窗口的大小选取和实际的丢包率以及延迟时间是息息相关的，也就是说如果能在当时的丢包率下选择合适的窗口大小就能够达到较好的传输效率。这一思想与拥塞控制的思想不谋而合，对于拥塞控制而言，就是一个逐步试探、探测的过程，如果网络实际条件好就会逐步增大拥塞窗口的大小，滑动窗口由拥塞窗口和接收端的通告窗口所决定，一般来说通告窗口都不会太小，而如果出现了丢包或者拥塞的现象，拥塞控制会通过减小拥塞窗口的大小来控制数据分组的发送情况。所以其实拥塞控制有点类似于预测网络的情形，理论上来讲拥塞控制能够带来较大的性能提升。

固定滑动窗口大小为 5，而该部分的拥塞控制算法为为了适配 GBN 而实现的改进的 New Reno 算法，我们首先固定延迟时间，选择不同的丢包率来进行比较分析，对于滑动窗口的数据采取第二部分之中重复实验的数据，统计结果如下:

| 丢包率 | 吞吐率(Bytes/ms) | |
|---|---|---|
| | 滑动窗口 | 拥塞控制 |
| 0% | 3340.98 | 2920.96 |
| 1% | 862.54 | 1613.65 |
| 2% | 656.312 | 1583.65 |
| 5% | 327.065 | 831.95 |
| 10% | 185.264 | 412.65 |
| 15% | 112.654 | 187.65 |
| 20% | 77.3589 | 98.65 |

首先我们可以看到在不会丢包的情形下，因为滑动窗口始终保持着 5 的大小，并且拥塞控制在慢启动阶段每次接收一个新的 ACK 就会增加 1，但是在拥塞避免阶段是线性递增，增加的速度不快，且 1.jpg 的大小不是很大，这些因素共同作用导致滑动窗口在丢包率为 0 的情况下效率比拥塞控制高。




(该图有误差)

接下来我们就能够发现无论丢包率在 20% 以下是多少，传输的效率都有了不错的提升，在丢包率小于 10% 的情形下，传输效率有 2-3 倍的提升，而在 10%- 20% 的丢包率时则是约有 50% 左右的效率提升。在实验过程之中通过仔细的观察，发现其实每次传输过程之中在丢包率大于 5% 以后，基本上窗口大小波动的最大值在 10 左右，10 的中位数即为 5，所以能更加体现出拥塞控制的探测作用，也就是说在没有出现随机丢包时，总是认为网络情况是好的，所以会不断地增加数据分组的发送数量，而当出现丢包的情形之后就会认为网络情况变差，就会减小窗口大小。并且效率的提升还取决于三次重复的 ACK 响应之后就会认为出现了丢包的现象，在我的改进设计之中为了优化性能，也进行了独立的设计（也就是快速恢复阶段的设计，详细请见 UDP 可靠传输 3-3），提前判断丢包现象就可以避免计时器的超时重传了。所以，可以发现使用了改进的 New Reno 拥塞控制算法后，性能有了很不错的提升，对于 1.jpg 通常 3-4 秒就可以完成传输。

# 四、 总结

在本次实验之中，深入地对比分析了停等机制、滑动窗口机制和拥塞控制机制的差异，并且对实验过程之中特殊的实验环境导致的问题进行了分析。在实现过程之中，对于多线程与单线程的相关知识进行了深入了解。参考 TCP 的机制，我们仍需要对基于 UDP 服务的可靠数据传输设计进行一定的改编与优化。

**个人 GitHub 仓库链接**：https://github.com/Skyyyy0920/Computer-Network

# 参考文献

**1、03-计算机网络-第三章-2022.pdf**

**2、上机作业 3 讲解-2022.pdf**