
南開大學
Nankai University

计算机学院

计算机网络实验报告

# Lab3-4：基于 UDP 服务设计可靠传输协议并编程实现

姓名：何禹姗

学号：2211421

专业：计算机科学与技术

2024 年 12 月 17 日

# 目录

**1 实验要求** **2**

**2 停等机制与滑动窗口机制性能对比** **2**

**3 滑动窗口机制中不同窗口大小对性能的影响** **5**

**4 有拥塞控制和无拥塞控制的性能比较** **8**

**5 总结** **11**

# 1 实验要求

基于给定的实验测试环境，通过改变延时和丢包率，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。

# 2 停等机制与滑动窗口机制性能对比

这部分测试滑动窗口（累计确认）的窗口大小为：client 端为 20,server 端为 1。

（1）先进行控制丢包率为 0%，查看不同延时下的停等机制与滑动窗口（累计确认）的性能，结果在下表中：

表 1: 丢包率 0% 时的性能数据

| 丢包率 0% | 延时 0ms | 延时 10ms | 延时 20ms | 延时 30ms | 延时 40ms |
|---|---|---|---|---|---|
| **停等机制** | | | | | |
| 时延 ms | 7 | 2849 | 5697 | 5692 | 8539 |
| 吞吐率 bytes/ms | 265 336 | 651.932 | 326.023 | 326.309 | 217.514 |
| **滑动窗口（累计确认）** | | | | | |
| 时延 ms | 4 | 2651 | 5124 | 5364 | 8215 |
| 吞吐率 bytes/ms | 464 338 | 700.623 | 362.481 | 346.262 | 226.093 |

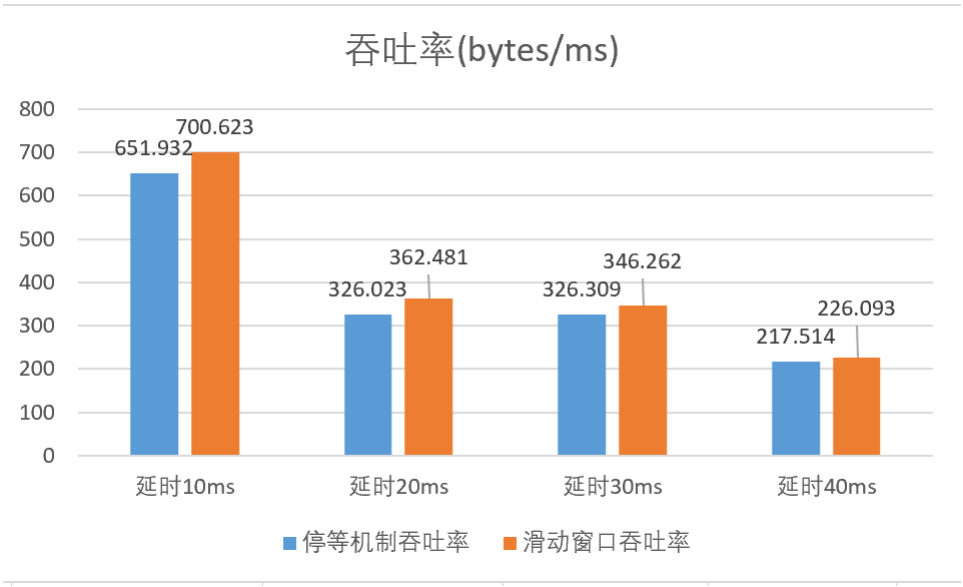


图 2.1: 吞吐率

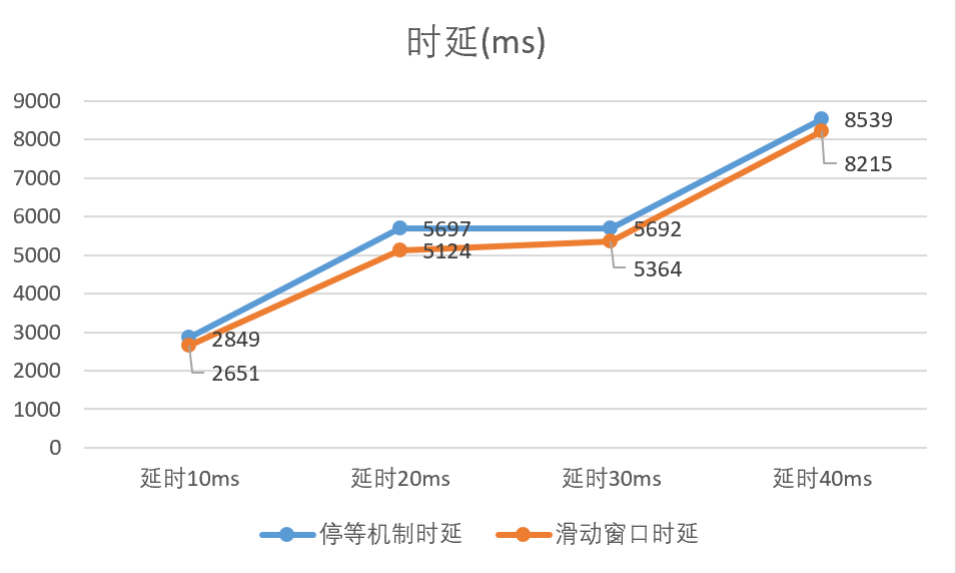


图 2.2: 时延

分析：丢包率为 0%，改变时延，滑动窗口（累计确认）存在加速效果，但是加速并不明显。分析打印日志可知，这是因为当发送一段时间后且没有丢包，滑动窗口回答道动态平衡，每当窗口收到一个确认报文，就会有一个空闲位置，也会立即发送一个新的报文段，所以如果丢包率为 0%，动态平衡后与停等机制无大差异，所以性能提升不明显。

滑动窗口机制可以在高延时情况下保持较高的发送速率，因为发送端可以连续发送多个数据包，即使确认延迟，发送端仍然可以利用窗口内的空间发送数据包，从而保持较高的吞吐率。

（2）进行控制延时为 0ms，查看不同丢包率下的停等机制与滑动窗口（累计确认）的性能，结果在下表中：

表 2: 延时 0ms 时的性能数据

| 延时 0ms | 丢包率 0% | 丢包率 1% | 丢包率 3% | 丢包率 6% | 丢包率 10% |
|---|---|---|---|---|---|
| **停等机制** | | | | | |
| 时延 ms | 7 | 1509 | 3010 | 5514 | 11 523 |
| 吞吐率 bytes/ms | 265 336 | 1230.85 | 617.061 | 336.843 | 161.187 |
| **滑动窗口（累计确认）** | | | | | |
| 时延 ms | 15 | 1021 | 5521 | 5013 | 9020 |
| 吞吐率 bytes/ms | 123 824 | 1819.15 | 336.416 | 370.507 | 205.915 |

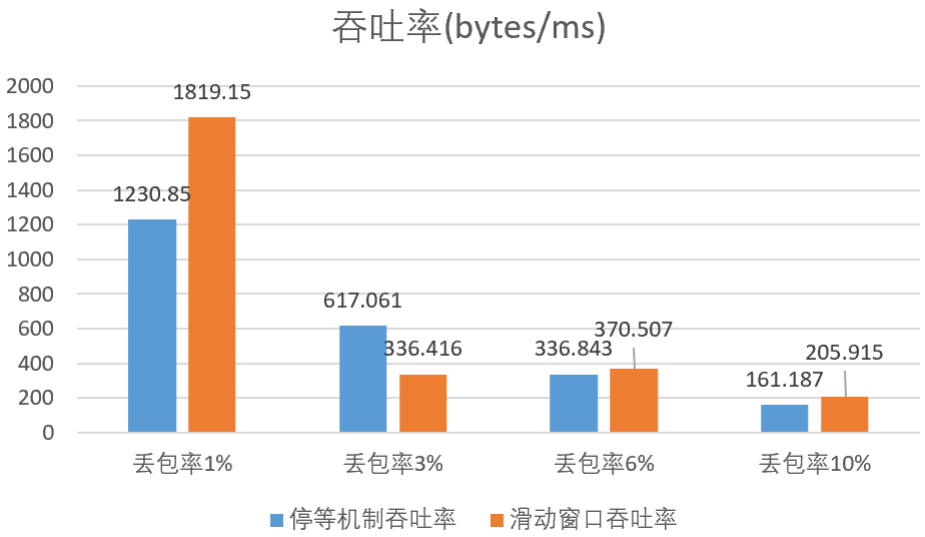


图 2.3: 吞吐率

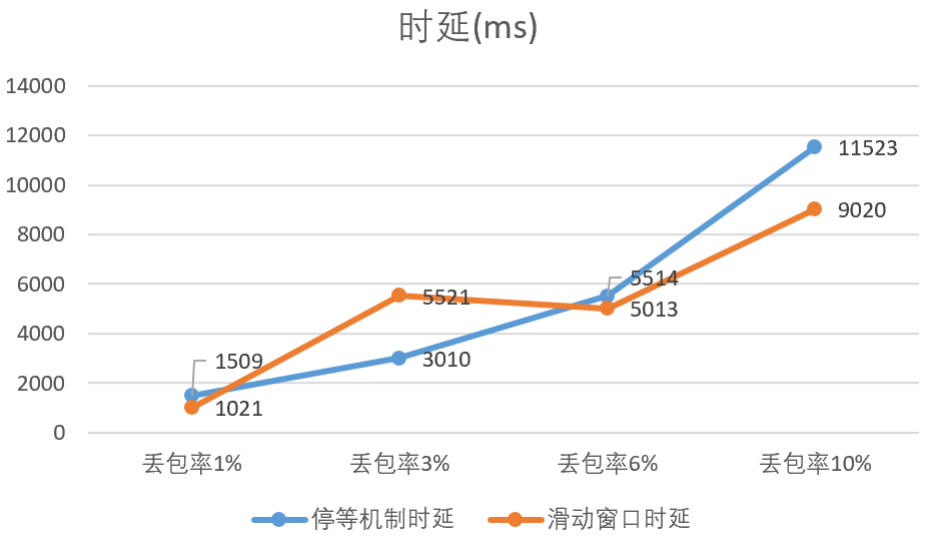


图 2.4: 时延

分析：时延为 0ms，改变丢包率，滑动窗口加速效果十分明显，原因如下：有丢包时，基于 GBN 协议的滑动窗口机制实现了三次快速重传机制，可以避免超时等待的时间，当收到三次重复的 ACK 即可重新发包，大大加速了因为丢包需要等待的时间。

同时我发现，在丢包率为 3% 时，停等机制比滑动窗口机制效率高，我又经过几轮测试，结果仍为此，分析原因可能为，停等机制在每次发送一个数据包后等待确认，因此在网络状况不佳（如高丢包率）时，它能够更精确地控制发送速率，避免过度发送导致的进一步拥塞，并且当数据包丢失时，停等机制会立即重传该数据包，从而减少累积的重传次数和等待时间。而滑动窗口机制使用累积确认，这意味着接收端只确认收到的数据包序列中的最后一个数据包。如果中间某个数据包丢失，接收端不会确认后续的数据包，导致发送端需要重传多个数据包。所以虽然滑动窗口机制可以动态调整窗口大小，但在高丢包率情况下，窗口调整可能不够及时或有效，导致更多的重传和等待时间。

# 3 滑动窗口机制中不同窗口大小对性能的影响

针对滑动窗口（累计确认）进行实验，这里由于 server 端的窗口大小被固定为 1，所以这里改变窗口大小只改变 client 端的窗口大小。

（1）先进行控制丢包率为 0%，查看不同延时下和不同窗口大小对滑动窗口（累计确认）的性能影响，结果在下表中：

表 3: 丢包率 0% 时的性能数据

| 丢包率 0% | 延时 0ms | 延时 10ms | 延时 20ms | 延时 30ms | 延时 40ms |
|---|---|---|---|---|---|
| | | **窗口大小 =1** | | | |
| 时延 ms | 7 | 2846 | 5691 | 6203 | 8511 |
| 吞吐率 bytes/ms | 265 336 | 652.619 | 326.367 | 299.428 | 218.23 |
| | | **窗口大小 =4** | | | |
| 时延 ms | 5 | 2842 | 5606 | 6425 | 8598 |
| 吞吐率 bytes/ms | 371 471 | 653.537 | 331.315 | 289.082 | 216.022 |
| | | **窗口大小 =8** | | | |
| 时延 ms | 4 | 2805 | 5560 | 6307 | 8606 |
| 吞吐率 bytes/ms | 464 338 | 662.158 | 334.056 | 294.491 | 215.821 |
| | | **窗口大小 =16** | | | |
| 时延 ms | 4 | 2784 | 5646 | 6422 | 8605 |
| 吞吐率 bytes/ms | 464 338 | 667.153 | 328.968 | 289.217 | 215.846 |
| | | **窗口大小 =32** | | | |
| 时延 ms | 4 | 2877 | 5517 | 6338 | 8582 |
| 吞吐率 bytes/ms | 464 338 | 645.587 | 336.66 | 293.05 | 216.424 |

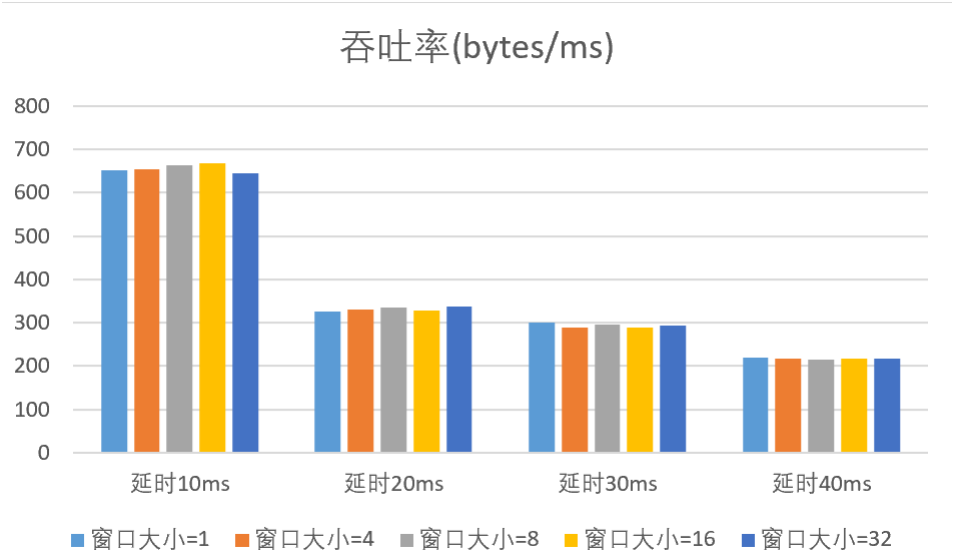


图 3.5: 吞吐率

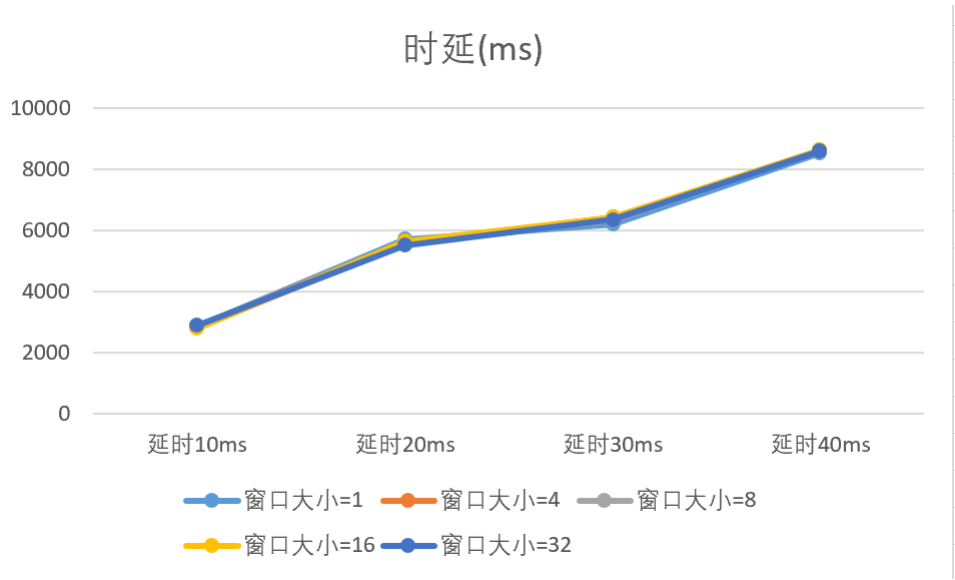


图 3.6: 时延

分析：对于基于 GBN 协议的滑动窗口机制，在丢包率为 0% 时，在不同的延时下，窗口越大，时延越低，吞吐率越大。分析可知窗口越大，可以同时发送端报文就越多，有利于更加充分的利用信道，降低时延，提高吞吐率。

但在丢包率为 0% 时，只改变延时，滑动窗口大小对吞吐率和时延的影响并不明显，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，窗口内收到一个确认报文，就会有一个空闲位置，也会立即发送一个新报文段，达到动态平衡状态。同时，窗口大小为 1 时的效果跟停等机制效果相近。

同时根据实验数据可以发现，在丢包率为 0% 时，只改变延时，滑动窗口大小对时延和吞吐率影响并不明显，分析原因可知，在丢包率为 0% 的情况下，数据包传输过程中没有丢失，这意味着所有发送的数据包都能被正确接收。这种情况下，滑动窗口机制的主要作用是控制发送速率，而不是处理重传。在没有丢包的情况下，吞吐率主要由网络带宽和窗口大小共同决定。窗口大小的增加可以提高吞吐率，但这种提高在高延时情况下会被延时的影响所抵消。

（2）进行控制延时为 0ms，查看不同丢包率下和不同窗口大小对滑动窗口（累计确认）的性能影响，结果在下表中：

表 4: 延时 0ms 时的性能数据

| 延时 0ms | 丢包率 0% | 丢包率 1% | 丢包率 3% | 丢包率 6% | 丢包率 10% |
|---|---|---|---|---|---|
| | | **窗口大小 =1** | | | |
| 时延 ms | 7 | 1008 | 1508 | 7519 | 10 522 |
| 吞吐率 bytes/ms | 265 336 | 1842.61 | 1231.67 | 247.021 | 176.521 |
| | | **窗口大小 =4** | | | |
| 时延 ms | 5 | 1006 | 1005 | 8519 | 7516 |
| 吞吐率 bytes/ms | 371 471 | 1846.28 | 1848.11 | 218.025 | 247.12 |
| | | **窗口大小 =8** | | | |
| 时延 ms | 4 | 1507 | 2507 | 6041 | 8518 |
| 吞吐率 bytes/ms | 464 338 | 1232.48 | 740.867 | 308.838 | 218.05 |
| | | **窗口大小 =16** | | | |
| 时延 ms | 4 | 1010 | 4011 | 9522 | 12 025 |
| 吞吐率 bytes/ms | 464 338 | 1838.96 | 463.065 | 195.059 | 154.458 |
| | | **窗口大小 =32** | | | |
| 时延 ms | 4 | 505 | 5014 | 8019 | 9022 |
| 吞吐率 bytes/ms | 464 338 | 3677.93 | 370.433 | 231.619 | 205.869 |

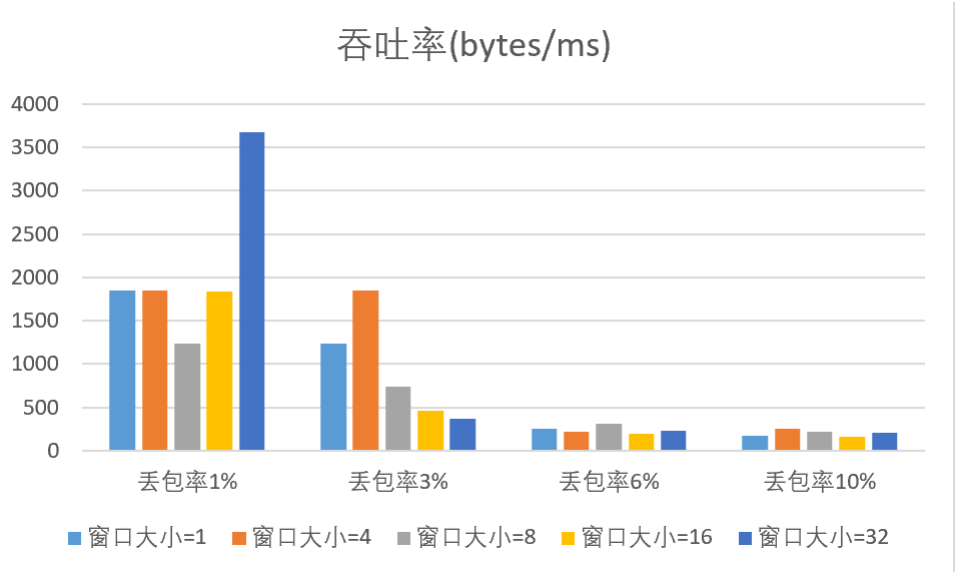


图 3.7: 吞吐率

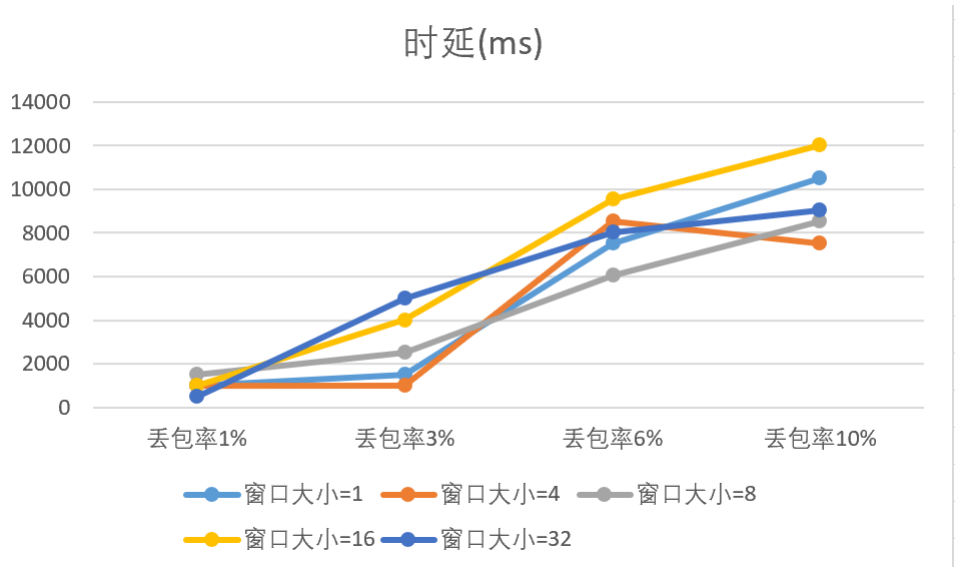


图 3.8: 时延

分析：对于基于 GBN 协议的滑动窗口机制，在延时为 0ms 时，随着丢包率的增加，时延显著增加。这是因为当数据包丢失时，需要重新传输这些数据包，这会增加总的传输时间；随着丢包率的增加，吞吐率显著下降。这是因为数据包的丢失会导致更多的重传，从而减少了有效传输的数据量。

在丢包率为 0% 的情况下，随着窗口大小的增加，时延略有下降，这是因为在没有丢包的情况下，较大的窗口可以更有效地利用带宽，减少等待时间；随着窗口大小的增加，吞吐率增加，这是因为较大的窗口允许发送更多的数据包，从而提高了吞吐率。

但在高丢包率下，即使窗口大小增加，时延仍然显著增加，吞吐率显著下降，这是因为每次发生超时重传的时候，会将窗口内的所有数据包重传，窗口越大，重传的数据包越多，重传机制会占用大量的时间和带宽。

综上可以看出，丢包率是影响时延和吞吐率的主要因素。随着丢包率的增加，时延显著增加，吞吐率显著下降。窗口大小在低丢包率下对性能有正面影响，但在高丢包率下效果有限，甚至可能因为重传机制而变得更差。

# 4 有拥塞控制和无拥塞控制的性能比较

无拥塞控制的滑动窗口大小设为 20。

（1）先进行控制丢包率为 0%，查看有拥塞控制和无拥塞控制对程序性能的影响，结果在下表中：

表 5: 丢包率 0% 时的性能数据

| 丢包率 0% | 延时 0ms | 延时 10ms | 延时 20ms | 延时 30ms | 延时 40ms |
|---|---|---|---|---|---|
| **无拥塞控制** | | | | | |
| 时延 ms | 7 | 2849 | 5697 | 5692 | 8539 |
| 吞吐率 bytes/ms | 265 336 | 651.932 | 326.023 | 326.309 | 217.514 |
| **有拥塞控制** | | | | | |
| 时延 ms | 4 | 2851 | 5698 | 5720 | 8561 |
| 吞吐率 bytes/ms | 464 338 | 651.474 | 325.966 | 324.712 | 216.955 |

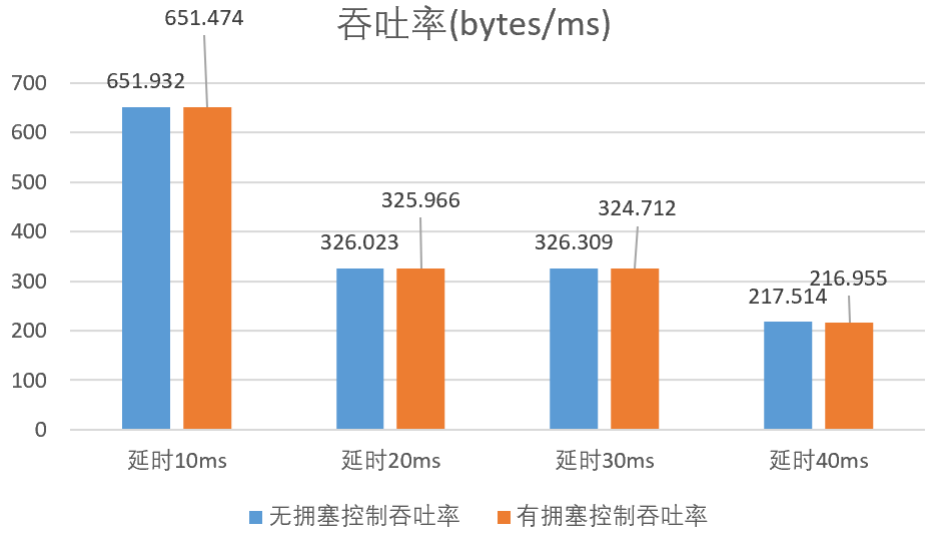


图 4.9: 吞吐率

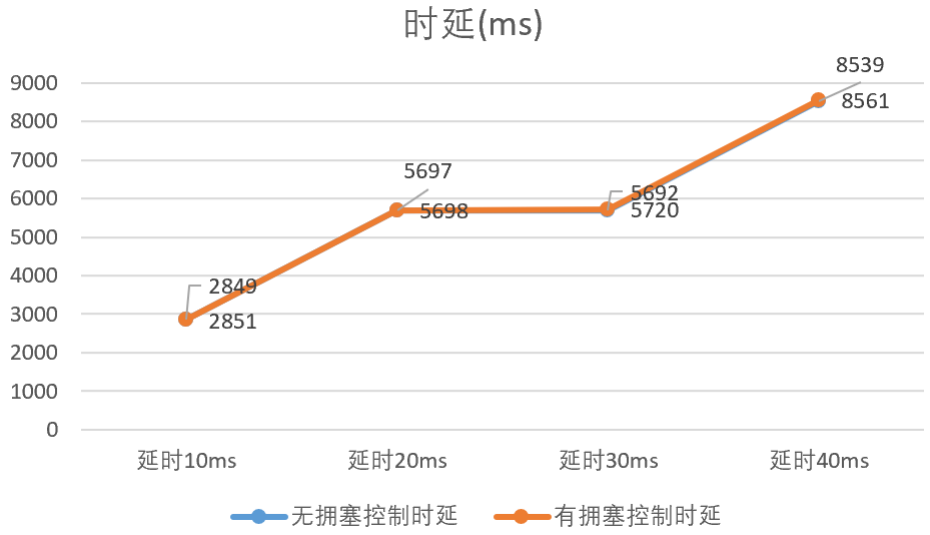


图 4.10: 时延

分析：在丢包率为 0% 时，改变延时，可以发现，在所有延时条件下，有拥塞控制的时延略低于无拥塞控制的时延，随着延时的增加，时延显著增加，但有拥塞控制和无拥塞控制之间的差异不大。在低延时（0ms）情况下，有拥塞控制的吞吐率显著高于无拥塞控制，随着延时的增加，吞吐率显著下降，且在高延时情况下，有拥塞控制和无拥塞控制的吞吐率差异较小。

在低延时（0ms）情况下，有拥塞控制的吞吐率显著高于无拥塞控制，分析原因为，拥塞控制采用滑动窗口机制，根据网络反馈动态调整窗口大小，在低延时情况下，这种机制能够快速响应网络状态，从而提高吞吐率；无拥塞控制意味着发送端以固定速率发送数据包，不考虑网络拥塞情况，这可能导致数据包堆积和丢包，尤其是在网络负载较高时。

而在高延时情况下，有拥塞控制和无拥塞控制的吞吐率差异较小，分析原因为，在高延时（如 40ms）情况下，数据包在网络中的传输时间显著增加。这意味着发送端接收到网络反馈的时间也会相应增加，由于反馈延迟，拥塞控制机制无法及时调整发送速率，即使拥塞控制试图动态调整窗口大小，但由于

反馈信息滞后，这种调整可能不那么有效。

（2）进行控制延时为 0ms，查看有拥塞控制和无拥塞控制对程序性能的影响，结果在下表中：

表 6: 延时 0ms 时的性能数据

| 延时 0ms | 丢包率 0% | 丢包率 1% | 丢包率 3% | 丢包率 6% | 丢包率 10% |
|---|---|---|---|---|---|
| **无拥塞控制** | | | | | |
| 时延 ms | 7 | 1509 | 3010 | 5514 | 11 523 |
| 吞吐率 bytes/ms | 265 336 | 1230.85 | 617.061 | 336.843 | 161.187 |
| **有拥塞控制** | | | | | |
| 时延 ms | 4 | 506 | 1006 | 2009 | 5515 |
| 吞吐率 bytes/ms | 123 824 | 3670.66 | 1846.28 | 924.516 | 336.782 |

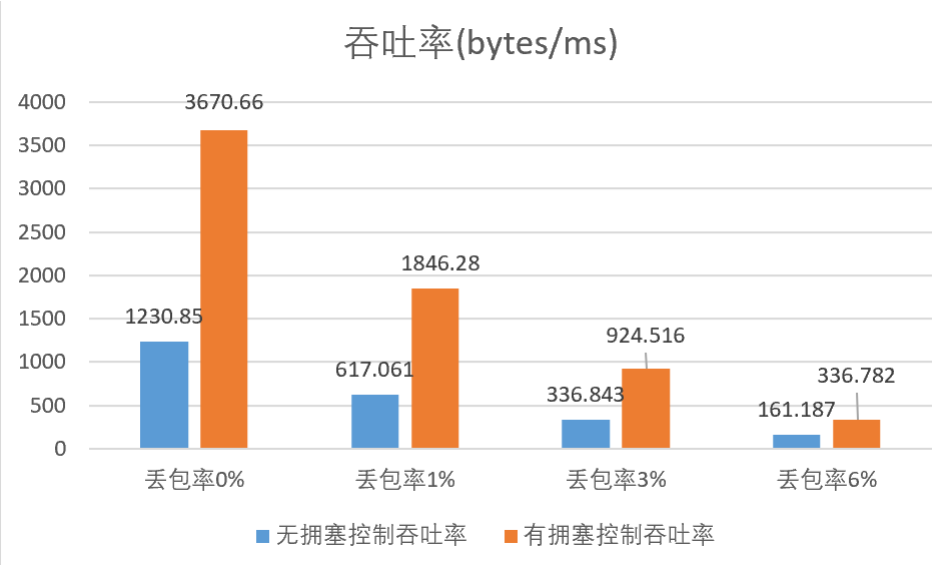


图 4.11: 吞吐率

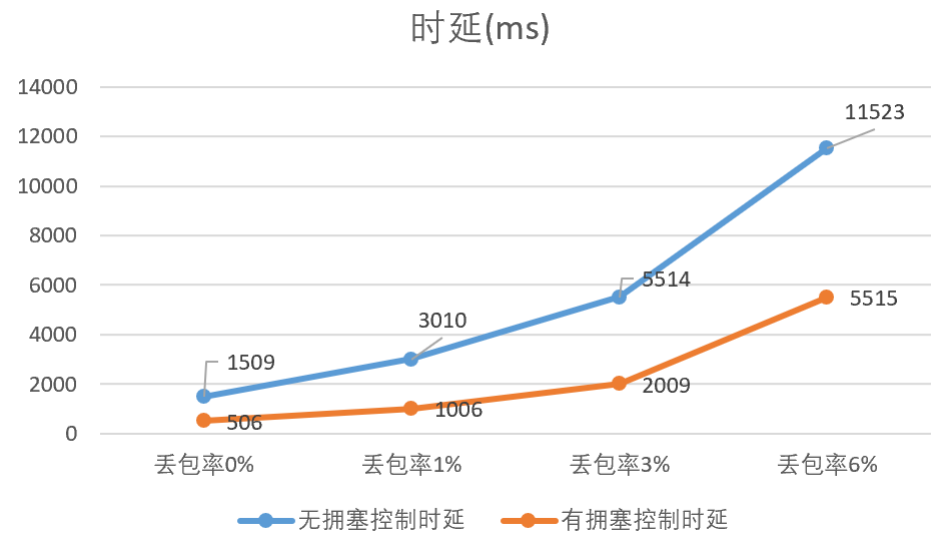


图 4.12: 时延

分析：在延时为 0ms 时，改变丢包率，无拥塞控制随着丢包率的增加，时延显著增加，吞吐率显著下降；有拥塞控制也随着丢包率的增加，时延显著增加，吞吐率显著下降，但是变化幅度均小于无拥塞控制。在低丢包率情况下，有拥塞控制的时延明显低于无拥塞控制。随着丢包率的增加，时延显著增加，但有拥塞控制的时延增幅较小。

分析有拥塞控制变化幅度小于无拥塞控制的原因，拥塞控制机制能够根据网络反馈动态调整发送速率和窗口大小。在丢包率增加时，拥塞控制会减小发送速率以避免进一步的网络拥塞，并且拥塞控制机制能够快速响应网络状态的变化，如丢包率的增加，从而减少数据包的重传次数和等待时间。同时，拥塞控制实现了快速重传，而不仅仅依赖于超时重传，也减少了超时等待时间。

存在丢包的情况下，采用拥塞控制机制可以更好地应对网络状况变化，保持相对稳定的吞吐率和较低的时延。所以在高丢包率情况下，有拥塞控制的系统表现更稳定，时延和吞吐率的变化幅度较小。

# 5 总结

1、停等协议：停等协议使数据传输十分可靠，每次发送一个数据包后等待确认，确保每个数据包的可靠传输；然而在高延时或高丢包率情况下，效率低下。每次发送后都需要等待确认，导致吞吐率显著下降，由于每次发送后都需要等待确认，即使在网络状况良好时，也会引入额外的延时。

2、滑动窗口机制（累积确认）：采用滑动窗口机制十分高效，允许连续发送多个数据包，无需等待每个数据包的确认，提高了吞吐率；但在高丢包率情况下，如果中间某个数据包丢失，接收端不会确认后续的数据包，导致发送端需要重传多个数据包，增加延时和降低吞吐率。

3、拥塞控制：拥塞控制会使传输更加稳定，且快速响应，能够根据网络反馈动态调整发送速率，减少网络拥塞，能够快速响应网络状态的变化，如丢包率的增加，从而保持较低的时延和较高的吞吐率；在某些情况下，初始性能可能不如其他机制，尤其是在低丢包率和低延时条件下。

4、在低延时（0ms）和低丢包率（0%）的情况下，滑动窗口机制表现出更好的时延和吞吐率，随着延时和丢包率的增加，停等机制在中等丢包率（如 3%）下的表现优于滑动窗口机制。

5、经过实验我认为，停等协议适用于低延时和中等丢包率的情况，能够有效应对丢包，保持较高的吞吐率。滑动窗口机制适用于低延时和低丢包率的情况，能够提供更高的吞吐率和较低的时延。而拥塞控制适用于高延时和高丢包率的情况，能够动态调整发送速率，减少网络拥塞。我们应该根据具体的网络环境和需求选择最合适的机制，以达到最佳的网络性能。