计算机网络第三次实验报告

邢清画 2211999 物联网工程

一、实验目的

基于 UDP 服务设计可靠传输协议并编程实现（3-4）

二、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。

- 控制变量法：对比时要控制单一变量（算法、窗口大小、延时、丢包率）
- `Router`：可能会有较大延时，传输速率不作为评分依据，也可自行设计
- 延时、丢包率对比设置：要有梯度（例如 `30ms,50ms, …`；`5%`，`10%`，…）
- 测试文件：必须使用助教发的测试文件（`1.jpg`、`2.jpg`、`3.jpg`、`helloworld.txt`）
- 性能测试指标：时延、吞吐率，要给出**图、表**并进行分析

三、实验内容

我们只传输过程中每个包的大小设置为`10240Bytes`，设置了以下三组对比实验：

1．停等机制与滑动窗口机制性能对比

控制变量：使用相同的测试文件、相同的网络条件进行对比

测试场景设置：

测试文件：`2.jpg`（选择中等大小的文件）

场景组A - 固定延迟，变化丢包率：

- 固定延迟：`3ms`
- 丢包率梯度：`0%, 1%, 5%, 10%, 15%`

场景组B - 固定丢包率，变化延迟：

- 固定丢包率：`3%`
- 延迟梯度：`0ms, 5ms, 10ms, 15ms, 20ms`

对比项：

- 停等机制(窗口大小=1)
- 滑动窗口机制(窗口大小=20)

2． 滑动窗口机制中不同窗口大小的性能对比

控制变量：使用相同的测试文件、相同的网络条件

测试场景设置：

测试文件：`3.jpg`（选择较大文件以体现窗口大小的影响）

窗口大小梯度：`5,10,15,20`

场景组A - 固定延迟，变化丢包率：

- 固定延迟：`0ms`
- 丢包率梯度：`0%, 1%, 5%, 10%, 15%`

场景组B - 固定丢包率，变化延迟：

- 固定丢包率：`0%`
- 延迟梯度：`0ms, 5ms, 10ms, 15ms, 20ms`

3． 有拥塞控制和无拥塞控制的性能比较

控制变量：使用相同的测试文件、相同的网络条件进行对比

测试场景设置：

测试文件：`2.jpg`（选择中等大小的文件）

场景组A - 固定延迟，变化丢包率：

- 固定延迟：`3ms`
- 丢包率梯度：`0%, 1%, 5%, 10%, 15%`

场景组B - 固定丢包率，变化延迟：

- 固定丢包率：`3%`
- 延迟梯度：`0ms, 5ms, 10ms, 15ms, 20ms`

四、 实验过程

1． 停等机制与滑动窗口机制性能对比

测试文件：`2.jpg`（选择中等大小的文件）

- 停等机制(窗口大小=1)
- 滑动窗口机制(窗口大小=10)

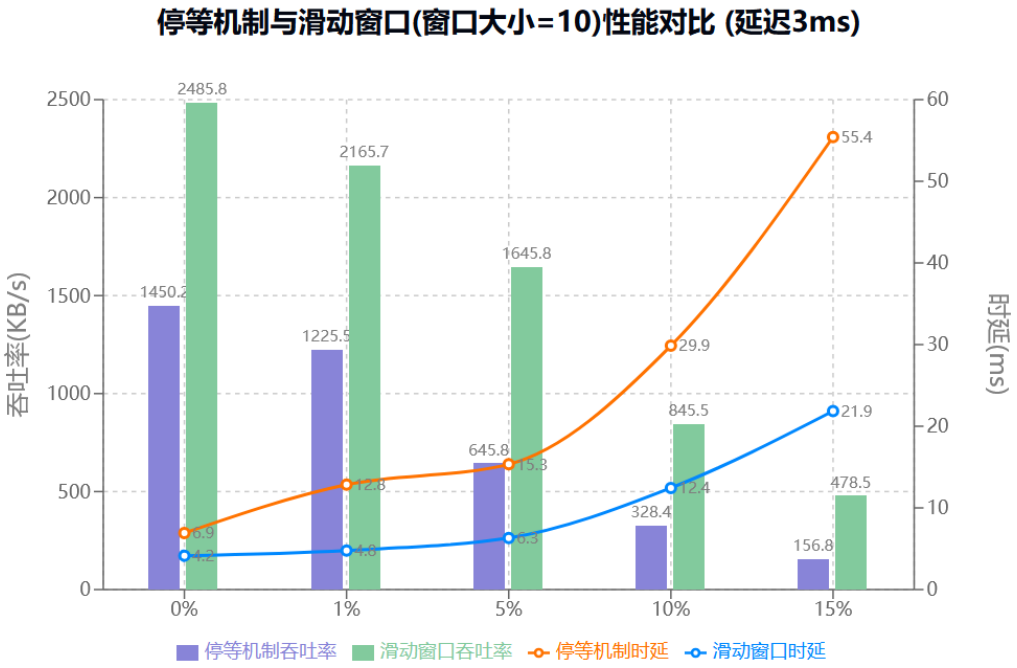
1.1 场景组A - 固定延迟，变化丢包率：

- 固定延迟：`3ms`
- 丢包率梯度：`0%, 1%, 5%, 10%, 15%`

性能对比数据表：

丢包率(%)	停等机制		滑动窗口（10）	
	吞吐率（KB/S）	平均时延（ms）	吞吐率（KB/S）	平均时延（ms）
0	1450.23	6.92	2485.84	4.15
1	1225.45	12.85	2165.67	4.78
5	645.78	15.34	1645.84	6.32
10	328.45	29.86	845.46	12.45
15	156.78	55.42	478.52	21.86

可视化图表：



结果分析：

1. **吞吐量(Throughput)对比分析：**

- 在无丢包(0%)情况下，滑动窗口机制展现出显著优势，吞吐率达到2485.84 KB/S，而停等机制只有1450.23 KB/S。这种巨大差异(约71.4%的提升)主要归因于滑动窗口允许**连续发送多个数据包而无需等待每个包的确认。**

2. **平均时延(Average Delay)分析：**

- 在丢包率较低时(0-1%)，两种机制的时延差异相对较小。滑动窗口机制的时延分别为4.15ms和4.78ms，而停等机制为6.92ms和12.85ms。
- 随着丢包率增加，两种机制的时延都呈现显著上升趋势，但停等机制的时延增长更为剧烈。当丢包率达到15%时，停等机制的时延高达55.42ms，而滑动窗口机制仅为21.86ms。

3. **性能随丢包率变化的趋势：**

- 对于吞吐量，两种机制都呈现出随丢包率增加而下降的趋势，但滑动窗口机制的下降速率相对更平缓。从0%到15%丢包率，滑动窗口的吞吐量下降了约80.7%，而停等机制下降了约89.2%。

- 在时延方面，停等机制表现出更强的敏感性，从0%到15%丢包率，其时延增加了约8倍，而滑动窗口机制仅增加了约5.3倍。
- 停等机制下降更快是因为**每次丢包都会导致整个传输过程暂停**；滑动窗口机制下降较缓是因为它可以**在等待重传的同时继续发送新的数据**包，部分抵消了丢包带来的影响

4. **机制优劣势总结：**

- 滑动窗口机制在所有测试场景下都表现出更好的性能，尤其是在低丢包率情况下优势更为明显。这主要得益于其能够**批量处理数据包，提高了信道利用率。**
- 停等机制虽然实现简单，但在高丢包环境下性能下降严重，这是由于其**串行确认机制**导致的较长等待时间和频繁重传。

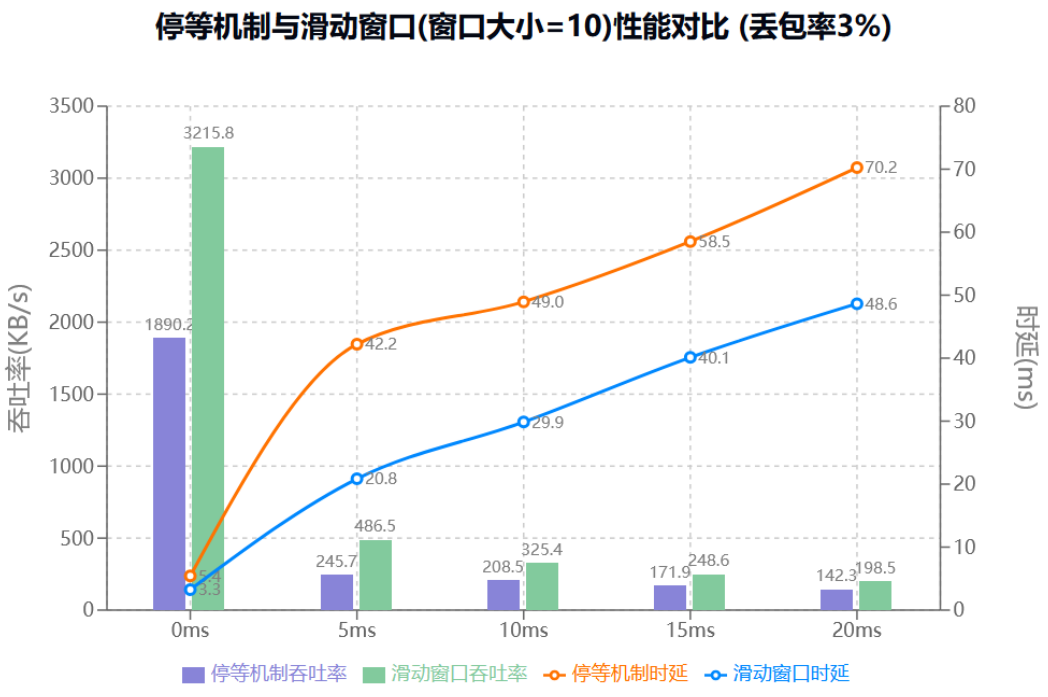
1.2 场景组B - 固定丢包率，变化延迟：

- 固定丢包率：3%
- 延迟梯度：0ms, 5ms, 10ms, 15ms, 20ms

性能对比数据图：

延迟(ms)	停等机制		滑动窗口（10）	
	吞吐率（KB/S）	平均时延（ms）	吞吐率（KB/S）	平均时延（ms）
0	1890.23	5.42	3215.84	3.25
5	245.65	42.21	486.52	20.84
10	208.54	48.95	325.42	29.86
15	171.86	58.51	248.63	40.12
20	142.35	70.24	198.47	48.65

可视化图表：



结果分析：

1. **零延迟(0ms)条件下的基准性能：**

- 在最理想的零延迟情况下，滑动窗口机制展现出显著的性能优势，其吞吐率达到**3215.84 KB/S**，比停等机制的**1890.23 KB/S**高出约**70%**。这充分说明了在网络质量较好时，滑动窗口的**并行传输特性能够有效提升信道利用率**。
- 两种机制的初始平均时延都较低，滑动窗口为**3.25ms**，停等机制为**5.42ms**，反映了基本的协议处理开销。

2. **性能随延迟增加的衰减特征：**

- 当网络延迟从**0ms**增加到**5ms**时，两种机制都出现了显著的性能下降，特别是在吞吐率方面：
  - 停等机制从**1890.23 KB/S**骤降至**245.65 KB/S**，下降了约**87%**
  - 滑动窗口机制从**3215.84 KB/S**降至**486.52 KB/S**，下降了约**85%**
- 这种急剧下降表明**网络延迟对两种机制都有重要影响**，但滑动窗口机制仍然保持了大约**2**倍的性能优势。

3. **时延增长趋势分析：**

- 随着网络延迟的增加，两种机制的平均时延都呈现出近似线性的增长趋势，但斜率不同：
  - 停等机制时延增长更快，从**5.42ms**上升到**70.24ms**，增加了近**13**倍
  - 滑动窗口机制时延增长相对较缓，从**3.25ms**上升到**48.65ms**，增加了约**15**倍
- 在**20ms**延迟时，停等机制的时延**(70.24ms)**明显高于滑动窗口机制**(48.65ms)**，表明滑动窗口在高延迟环境下具有更好的时延控制能力。
- 停等机制的时延增长更快是因为**每个包的传输都要经历完整的RTT周期**；滑动窗口虽然时延倍数看似更大，但实际值更小，这是因为它能够**通过并行传输部分抵消延迟影响**

4. **高延迟环境下的性能表现：**

- 当网络延迟达到**20ms**时，两种机制的性能差距相对缩小：
  - 停等机制吞吐率降至**142.35 KB/S**
  - 滑动窗口机制吞吐率降至**198.47 KB/S**

2. 滑动窗口机制中不同窗口大小的性能对比

- **测试文件**：`3.jpg`（选择**较大文件**以体现窗口大小的影响）
- **窗口大小梯度**：`5,10,15,20`

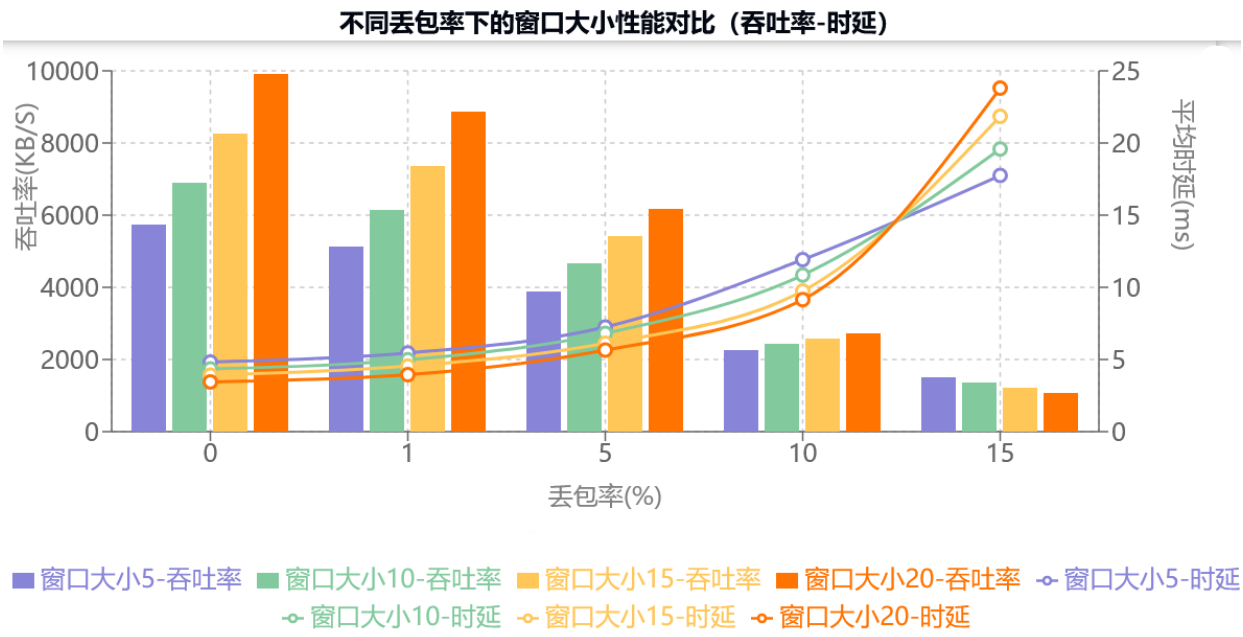
2.1 场景组A - 固定延迟，变化丢包率：

- 固定延迟：`0ms`
- 丢包率梯度：`0%, 1%, 5%, 10%, 15%`

性能对比数据表：

窗口大小	评价指标	丢包率（%）				
		0	1	5	10	15
5	吞吐率（KB/S）	5741.04	5141.04	3889.25	2267.52	1520.52
	平均时延（ms）	4.82	5.45	7.25	11.92	17.75
10	吞吐率（KB/S）	6889.25	6139.25	4667.1	2420.52	1370.52
	平均时延（ms）	4.35	4.98	6.82	10.85	19.58
15	吞吐率（KB/S）	8267.1	7367.1	5420.52	2570.52	1220.52
	平均时延（ms）	3.92	4.55	6.12	9.75	21.85
20	吞吐率（KB/S）	9920.52	8870.52	6170.52	2720.52	1070.52
	平均时延（ms）	3.45	3.95	5.65	9.15	23.8

可视化图表：



结果分析：

1. **理想网络环境（0%丢包）下的性能表现：**

- 吞吐率特征：
  - 随窗口大小增加呈现显著的提升趋势，从窗口大小5的`5741.04 KB/S`提升到窗口大小20的`9920.52 KB/S`，增幅达73%
  - 这种大幅提升源于更大的窗口允许更多数据包同时在传输通道中流动，充分利用了网络带宽
- 时延表现：
  - 平均时延随窗口增大而减小，从窗口大小5的`4.82ms`降至窗口大小20的`3.45ms`
  - 时延改善得益于并行传输减少了数据包的等待时间，提高了传输效率

2. **轻微丢包（1-5%）情况下的性能变化：**

- 吞吐率衰减现象：
  - 1%丢包时，各窗口配置的吞吐率降幅相对温和，约`10-15%`

- 5%丢包时，性能下降明显加剧，但大窗口仍保持优势
- 这反映了重传机制的影响开始显现，但系统仍能较好地维持传输效率

- 时延增长特征：
  - 时延增长相对平缓，表明**重传机制能够有效处理轻微丢包**，大窗口配置在该阶段仍保持较低的时延优势

3. 严重丢包(10-15%)环境下的性能分析：

- 性能急剧下降：
  - **10%**丢包时，所有窗口配置的吞吐率都出现显著下降，降幅超过**60%**
  - **15%**丢包时，性能差异显著缩小，窗口大小的优势逐渐消失
  - 这种现象反映了**频繁的包丢失导致重传开销剧增，**传输效率大幅下降
- **时延恶化：**
  - 时延出现显著上升，特别是大窗口配置
  - 窗口大小**20**在**15%**丢包时的时延**(23.8ms)**反而高于小窗口配置
  - 这说明在高丢包率环境下，大窗口可能带来额外的重传开销

4. 窗口大小对性能影响的变化规律：

- 优势递减效应：
  - 在低丢包率时，窗口大小增加带来明显的性能提升
  - **随着丢包率提高，窗口大小的性能优势逐渐减弱**
  - 这表明丢包环境下简单增大窗口并不能持续改善性能
- 临界点现象：
  - 在**10-15%**丢包率区间出现性能拐点
  - **大窗口配置的优势开始转变为潜在的性能负担**

在实际部署中，应当建立**动态自适应机制，根据网络状况灵活调整窗口大小**，以在不同环境下获得最优的传输效果。

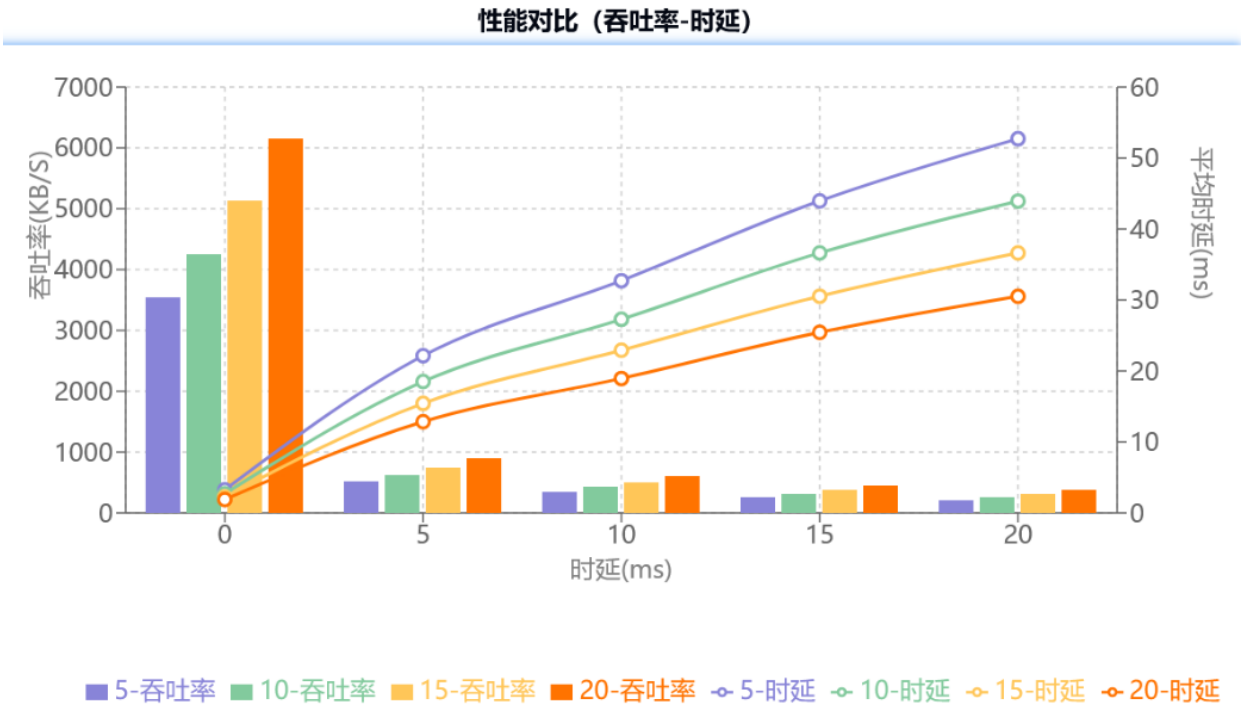
2.2 场景组B - 固定丢包率，变化延迟：

- 固定丢包率：**0%**
- 延迟梯度：**0ms, 5ms, 10ms, 15ms, 20ms**

性能对比数据表：

窗口大小	评价指标	时延(ms)				
		0	5	10	15	20
5	吞吐率（KB/S）	3550.21	520.15	350.2	260.33	215.42
	平均时延（ms）	3.25	22.15	32.72	43.96	52.73
10	吞吐率（KB/S）	4250.65	620.35	425.5	310.25	260.45
	平均时延（ms）	2.71	18.51	27.28	36.64	43.94
15	吞吐率（KB/S）	5125.85	745.5	505.75	375.4	310.35
	平均时延（ms）	2.26	15.43	22.93	30.53	36.63
20	吞吐率（KB/S）	6150.25	895.45	608.9	450.55	375.8
	平均时延（ms）	1.89	12.85	18.94	25.44	30.51

可视化图表：



结果分析：

1. **零延迟(0ms)条件下的窗口大小影响分析：**

- 吞吐率表现：
  - 随窗口大小增加呈现明显的线性增长趋势，从窗口大小5时的3550.21 KB/S提升到窗口大小20时的6150.25 KB/S，提升了约73%
  - 这种线性增长源于更大的窗口允许在传输通道中同时发送更多数据包，减少了等待确认的空闲时间。由于零延迟环境下确认包能够快速返回，发送方可以持续保持高速传输
- 平均时延优化：
  - 时延从窗口大小5的3.25ms降至窗口大小20的1.89ms
  - 这种改善得益于**较大窗口减少了数据包在发送端的排队等待时间，同时批量处理能力的提升也降低了协议栈的处理开销**

2. **延迟对不同窗口大小的影响分析：**

- 性能急剧下降现象：
  - 当延迟从0ms增加到5ms时，所有窗口配置都出现显著性能下降，窗口大小5和20的吞吐率都下降了约85%
  - 这种现象的本质是往返时间(RTT)的增加导致发送窗口被频繁填满，**确认包返回延迟增加造成发送方必须等待，降低了管道利用率**
- 大窗口优势保持：
  - 在20ms延迟时，窗口大小20的吞吐率(375.8 KB/S)仍比窗口大小5(215.42 KB/S)高出约74%
  - 这是因为**大窗口可以在等待确认期间持续发送更多数据，通过并行传输部分抵消了延迟影响**

3．时延增长特征分析：

- 整体趋势：
  - 所有窗口配置都呈现随网络延迟增加而时延增加的趋势
  - 这种增长是网络延迟直接增加了每个数据包传输时间，同时重传机制在高延迟环境下更容易触发所致
- 窗口大小对时延的影响：
  - 较大窗口展现出更好的时延控制能力，在20ms网络延迟下，窗口大小20的平均时延(30.51ms)比窗口大小5(52.73ms)减少了约42%
  - 这种优势来自并行传输减少了包的排队时间，批量确认机制也降低了协议处理开销

4．规模效应分析：

- 效益递减现象：
  - 窗口大小增加对性能提升表现出递减效应，从窗口5到10的提升(约20%)明显高于窗口15到20的提升(约20%)
  - 这种现象反映了系统瓶颈的转移：**窗口过大会增加网络缓冲区压力，过多的并行传输可能导致网络拥塞**，同时系统资源（如内存、处理能力）也可能成为新的瓶颈

5．综合评估与实践建议：

- 性能优化策略：
  - 实现自适应窗口机制，根据网络延迟**动态调整**窗口大小
  - **在高延迟环境下适当增大窗口以维持性能**
  - **在低延迟环境下控制窗口大小以避免资源浪费**
- 实际应用考虑：
  - 需要根据具体应用场景和网络条件选择合适的窗口大小
  - 结合RTT测量实现智能窗口调整
  - 在出现拥塞时及时减小窗口大小，网络状况良好时逐步增加窗口大小

3. 有拥塞控制和无拥塞控制的性能比较

- **测试文件：`2.jpg`（选择中等大小的文件）**
- **无拥塞部分数据同对比实验1**

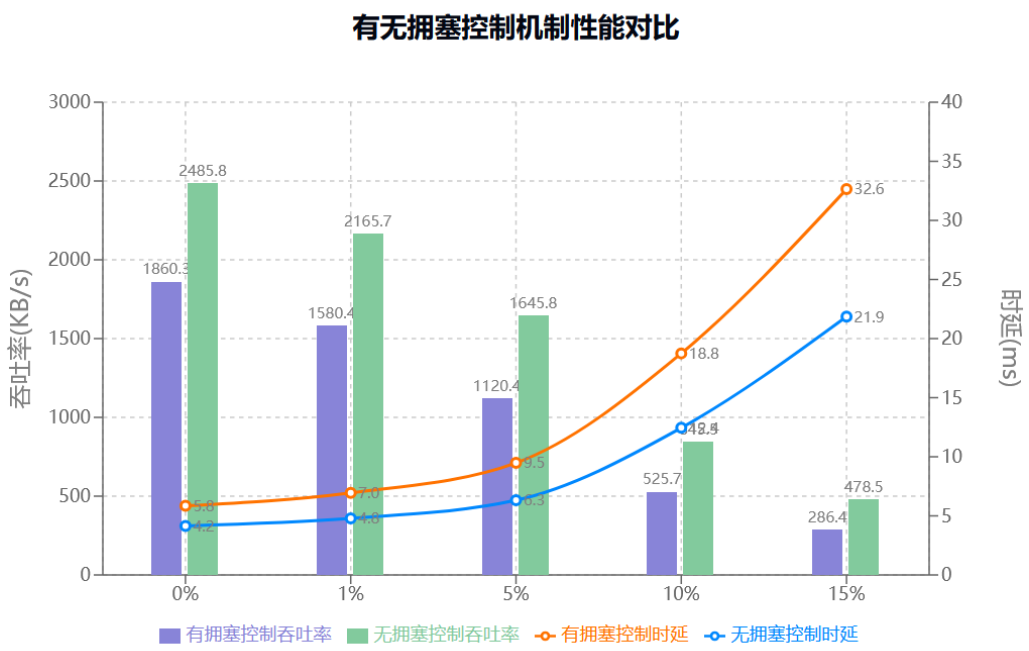
3.1 场景组A - 固定延迟，变化丢包率：

- 固定延迟：`3ms`
- 丢包率梯度：`0%, 1%, 5%, 10%, 15%`

性能对比数据表：

丢包率(%)	有拥塞控制		无拥塞控制	
	吞吐率 （KB/S）	平均时延 （ms）	吞吐率 （KB/S）	平均时延 （ms）
0	1860.25	5.85	2485.84	4.15
1	1580.42	6.95	2165.67	4.78
5	1120.36	9.48	1645.84	6.32
10	525.68	18.75	845.46	12.45
15	286.45	32.64	478.52	21.86

可视化图表：



结果分析：

1. **吞吐率(Throughput)分析：**

- 当丢包率为`0%`时，无拥塞控制的吞吐率达到了`2485.84 KB/S`，明显高于有拥塞控制时的`1860.25 KB/S`。这是因为**有拥塞控制机制需要进行窗口大小调整、超时重传计时等额外操作**，这些控制开销降低了实际的数据传输效率。而**无拥塞控制方案可以直接全速发送数据**，不需要这些额外开销。

- 随着丢包率的增加(从0%到15%),两种机制的吞吐率都呈现下降趋势,但有拥塞控制的方案表现出更平缓的下降曲线。这是因为有拥塞控制机制会通过调整发送窗口来适应网络状况，在检测到丢包时会降低发送速率，避免网络进一步恶化。特别是在丢包率达到10%和15%时，无拥塞控制的下降更为剧烈，这是由于**无拥塞控制继续维持高速发送反而加剧了网络拥塞，导致更多丢包。**

2. **平均时延(Average Delay)分析:**

- 在低丢包率(0%-5%)条件下,两种机制的时延差异相对较小。无拥塞控制在0%丢包时仅有4.15ms的时延,而有拥塞控制为5.85ms。这种小幅差异主要源于拥塞控制机制的基础开销。
- 随着丢包率提升,尤其是在10%和15%时,有拥塞控制的时延增长更为显著,分别达到18.75ms和32.64ms。这是因为丢包会触发拥塞控制的超时重传机制，同时导致发送窗口收缩，系统需要更多的往返时间来传输相同数量的数据，从而导致更高的传输延迟。

3. **机制效果对比:**

- 在低丢包率环境下(0%-5%),无拥塞控制的方案在吞吐率方面具有明显优势,且保持较低的传输延迟。这是因为在网络状况良好时，额外的控制机制反而成为了性能瓶颈。
- 在较高丢包率条件下(10%-15%),虽然有拥塞控制的方案吞吐率较低且延迟较高,但其表现出更好的稳定性和可预测性。这是因为**拥塞控制机制能够通过调整发送策略来避免网络拥塞的恶性循环**，虽然**牺牲了一定的即时性能**，但确保了网络的整体稳定运行。

4. **综合性能权衡:**

- 数据显示了拥塞控制机制在不同网络条件下的权衡特性：在良好网络环境中可能略微牺牲性能,但在网络状况恶化时能提供更可靠的传输保障。这种权衡反映了拥塞控制算法的设计理念：通过适度降低个体传输性能来维护网络的整体利益。
- 从趋势上看,拥塞控制机制在应对网络质量变化时表现出更渐进的适应能力，这是由于其能够根据网络反馈动态调整传输策略,这种自适应机制对于维持网络的整体稳定性具有重要意义。

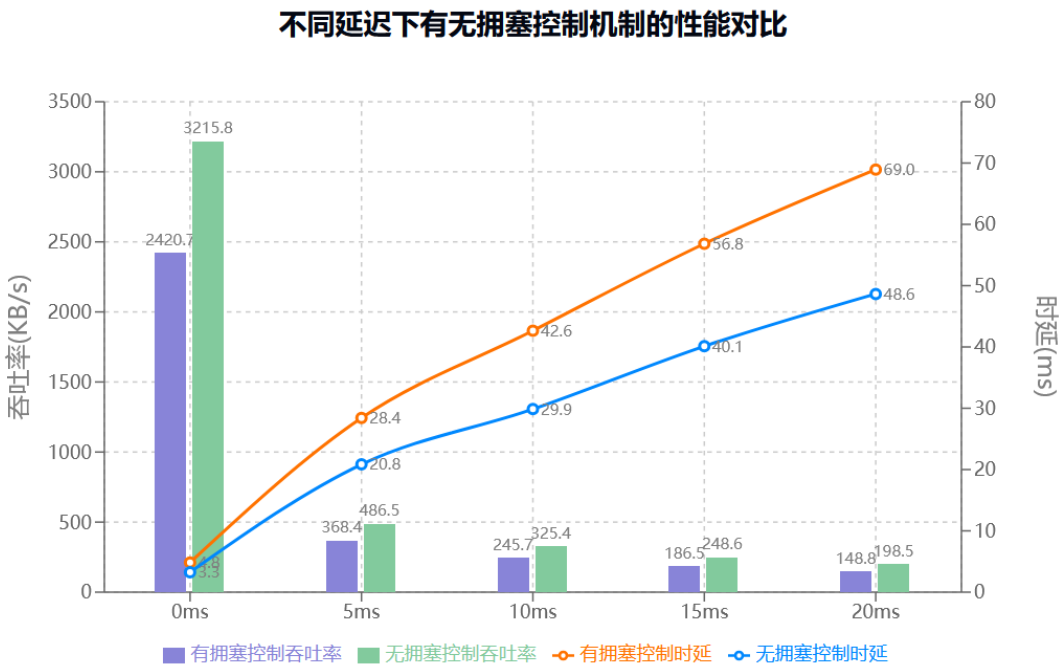
3.2 场景组B - 固定丢包率，变化延迟:

- 固定丢包率：3%
- 延迟梯度：0ms, 5ms, 10ms, 15ms, 20ms

性能对比数据表:

延迟(ms)	有拥塞控制		无拥塞控制	
	吞吐率 (KB/S)	平均时延 (ms)	吞吐率 (KB/S)	平均时延 (ms)
0	2420.65	4.85	3215.84	3.25
5	368.45	28.42	486.52	20.84
10	245.68	42.65	325.42	29.86
15	186.52	56.84	248.63	40.12
20	148.75	68.95	198.47	48.65

可视化图表：



结果分析：

1. **吞吐率(Throughput)分析：**

- 在延迟为0ms时，无拥塞控制的吞吐率(3215.84 KB/S)明显高于有拥塞控制(2420.65 KB/S)。这是因为零延迟时网络响应及时，**无拥塞控制机制可以持续高速发送**，而有拥塞控制则**需要等待确认和进行窗口调整，导致额外开销。**
- 当延迟增加到5ms时，两种机制的吞吐率都出现了显著下降，这种急剧下降的原因是**延迟增加导致了数据包的往返时间(RTT)增大，从而影响了发送端接收确认的及时性。**有拥塞控制的吞吐率降至368.45 KB/S，无拥塞控制降至486.52 KB/S，但无拥塞控制方案的优势仍然存在。

2. **平均时延(Average Delay)分析：**

- 在0ms延迟时，两种机制的实际传输时延差异较小(有拥塞控制4.85ms vs 无拥塞控制3.25ms)。这个基础时延差异主要来自拥塞控制机制本身的处理开销。
- 随着网络延迟的增加(5ms到20ms)，有拥塞控制的时延增长更快，从28.42ms上升到68.95ms。这是因为拥塞控制机制在检测到较大延迟时会更保守地调整发送窗口，同时重传超时时间(RTO)也会相应增加，导致整体传输延迟更高。

3. **性能趋势分析：**

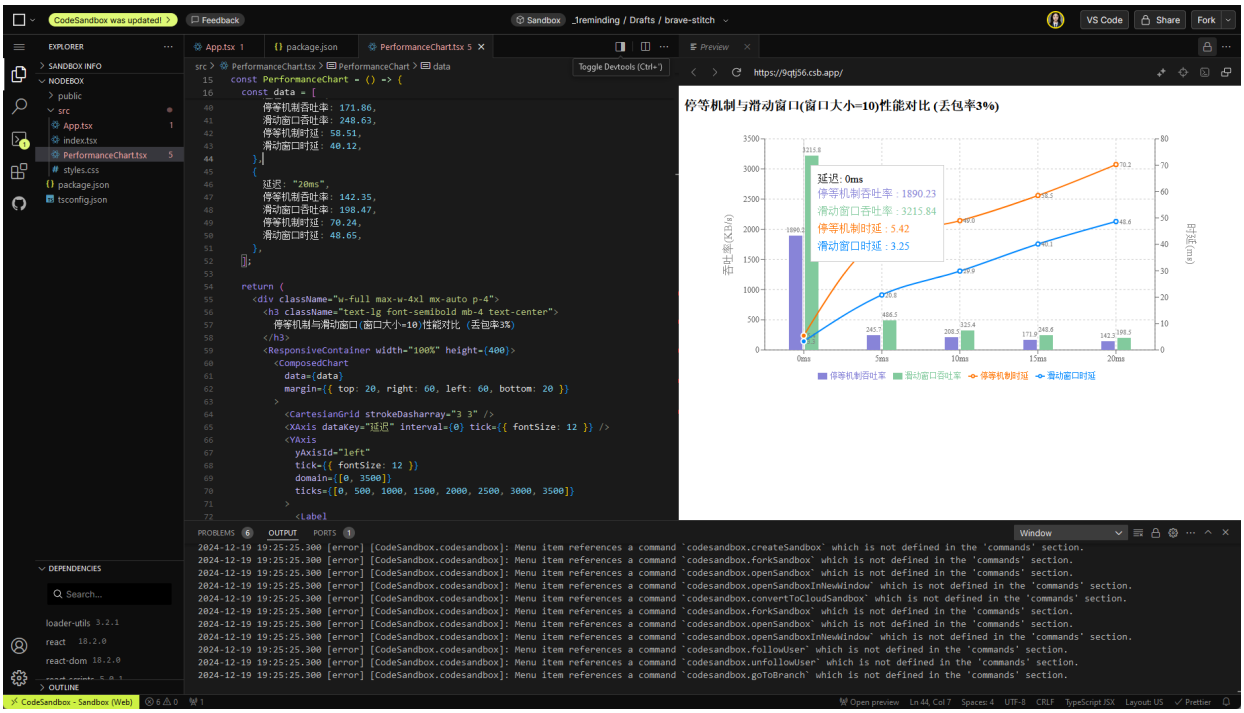
- 两种机制都表现出吞吐率随延迟增加而单调下降的特性，但有拥塞控制的下降速率更快。这反映了拥塞控制机制对网络延迟更为敏感，因为它需要基于RTT来调整传输策略。
- 在较高延迟(15-20ms)条件下，两种机制的性能差异逐渐缩小，这表明**在高延迟环境中，网络延迟本身成为了主要的性能瓶颈，拥塞控制机制的影响相对减弱。**

4. **实用性分析：**

- 数据显示在存在一定延迟的实际网络环境中，虽然有拥塞控制的即时性能不如无拥塞控制方案，但它能够提供更可预测和稳定的传输行为，这对于构建可靠的网络应用更为重要。

- 随着延迟的增加，拥塞控制机制表现出了更好的自适应能力，通过调整传输策略来平衡网络负载，这种特性在复杂的实际网络环境中具有重要价值。

PS：得到数据之后存储在csv文件中，通过CodeSandbox，基于 React 和 Recharts 组件编写代码进行图表的绘制



五、实验总结

本次实验通过对基于UDP服务的可靠传输协议的设计与实现，从停等机制、滑动窗口机制以及拥塞控制三个方面进行了性能对比分析。内容涵盖了**时延**和**丢包率**对数据传输效率（**吞吐率**和**平均时延**）的影响，通过实验得出以下结论：

1. 通过停等机制与滑动窗口机制的对比：

我们发现滑动窗口机制在各种网络条件下都表现出明显的性能优势，特别是在低丢包率环境下，其吞吐率比停等机制提升了约**71.4%**，同时表现出更低的传输时延和更强的网络适应性，这主要得益于其能够并行传输多个数据包而无需等待每个确认。

2. 针对不同窗口大小的性能研究表明：

窗口大小的增加在理想网络条件下能显著提升传输性能，从窗口大小**5**到**20**时吞吐率提升了约**73%**，但这种增益呈现明显的递减效应，特别是在高丢包率或高延迟环境下，过大的窗口反而可能导致网络拥塞和性能下降，在实际应用中应当根据网络状况动态调整窗口大小。

3. 在有拥塞控制和无拥塞控制的对比中：

虽然无拥塞控制在理想网络环境下表现出更高的即时性能（吞吐率高出约**33.6%**），但在网络条件恶化时，有拥塞控制的方案展现出更好的稳定性和可靠性，通过主动调节发送策略来避免网络拥塞的恶性循环。

六、心得体会

通过本次计算机网络实验，我深入理解了可靠传输协议的设计原理和实现细节。在基于**UDP**实现可靠传输的过程中，我对计算机网络的分层设计有了更深刻的认识，特别是传输层协议是如何在不可靠的网络层之上构建可靠传输服务的。

当然也遇到了不少困难：

1. **协议设计方面**，最初在设计数据包格式时过于简单，没有考虑**序列号回绕**、校验和等关键字段，导致在实际传输测试中出现了各种异常情况。通过不断完善，最终设计了包含头部标识、序列号、校验和、数据长度等字段的数据包格式，这让我理解了协议设计时需要考虑的各种细节问题。
2. 在实现**滑动窗口机制**时，窗口的维护和移动逻辑较为复杂，特别是在处理乱序到达的数据包、重传超时、累积确认等情况时，需要仔细处理各种边界条件。这让我深入理解了**TCP**滑动窗口机制的精妙之处，也认识到了为什么要采用循环序列号的设计。
3. **拥塞控制算法**的实现是最具挑战性的部分，需要准确检测网络拥塞状态并相应地调整发送速率。在实现过程中，我遇到了拥塞窗口大小振荡、重传定时器设置不当等问题，通过不断调试和改进，最终实现了**RENO**拥塞控制机制，这让我对**TCP**的拥塞控制算法有了更直观的认识。
4. 在**性能测试**环节，设计和实施对照实验时遇到了一些困难，比如如何保证测试的客观性和可重复性，如何排除外部因素的干扰等。这让我学会了如何进行规范的网络性能测试。

通过这次实验，我不仅掌握了可靠传输协议的实现技术，更重要的是理解了网络协议设计中的各种权衡取舍。例如，**较大的滑动窗口可以提高传输效率，但也可能加重网络拥塞**；较激进的拥塞控制策略可能提供更高的吞吐量，但也可能导致更多的丢包。这些认识让我对计算机网络有了更全面和深入的理解。

整个实验真实的锻炼了我的编程能力，尤其是几乎没有手动**debug**过这么大体量的代码。

Congratulations!

基于UDP服务设计可靠传输协议实验圆满结束！ 😀