计算机网络实验报告 Lab3-4

基于 UDP 服务设计可靠传输协议

作者：曹珉浩

组织：南开大学计算机学院

时间：December 10, 2023

学号：2113619

目录

第 1 章 实验要求与简单回顾

1.1 本次实验具体要求

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：

- 停等机制与滑动窗口机制性能对比
- 滑动窗口机制中不同窗口大小对性能的影响（累计确认和选择确认两种情形）
- 滑动窗口机制中相同窗口大小情况下，累计确认和选择确认的性能比较

1.2 协议设计

1.2.1 报文格式设计

我们的报文格式如下所示，分为数据报首部和具体数据部分。其中，在停等机制和累计确认机制中，我们使用了除**接收窗口通告**和**服务端窗口开始位置**两个字段外的所有字段；在选择确认机制中，使用了如下的所有字段，它们的长度类型在下图中已经给出，具体含义和作用在此不做赘述，可以参考前面的三个报告。

图 1.1: 协议设计：数据报具体格式

1.2.2 连接建立与释放的设计

由于我们整个 Lab3 都是在 UDP 之上实现可靠数据传输，因此连接的建立与释放都是仿照 TCP 实现的，即三次握手建立连接、四次挥手释放连接，这个过程的示意以及序列号变化情况如下图所示：

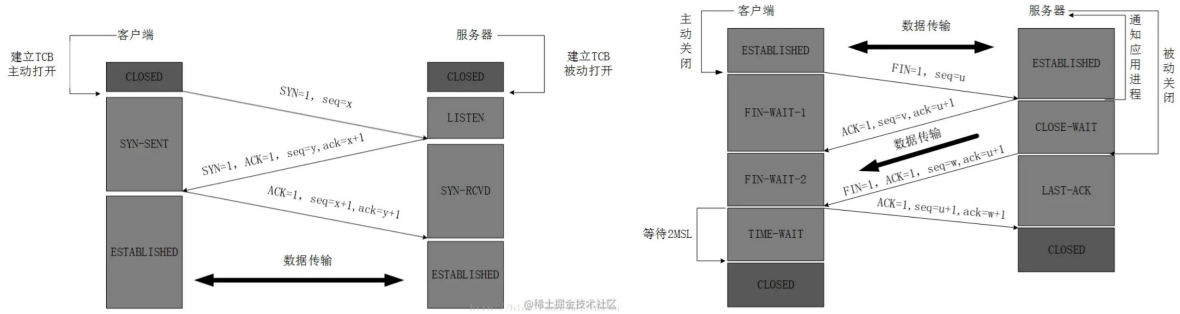

图 1.2: 协议设计：连接的建立与释放

1.2.3 停等机制实现的设计

停等机制是指，**发送端发送一个分组，然后等待接收端响应**，如果超过一定时间没有收到接收端的响应，那么重传这个分组，而若分组仅仅是被延迟或者 ACK 丢失，会造成接收端的重复接收，这时还需要接收端根据序列号判断分组是否重复，如果重复要进行丢弃，根据如上概念，我们客户端的发送逻辑设计如下：

- 客户端将数据封装到结构体中之后，就进行发送，并开始计时
- 如果在超时时间之内收到服务端传来的报文并且 ACK 标志位、序列号、校验和等都无误，则数据报传输成功，进行下一轮传输
- 如果在超时时间之内收到了 ACK 报文但上述三个条件发生差错，那么不做处理，等待超时后重传报文
- 如果超时仍未收到 ACK 报文，则重新传输

在停等机制中，服务端的接收逻辑非常简单，只需要判断收到的报文序列号是不是当前所期待的序列号即可，如果是就同意接受，并期待下一个序列号，否则就拒绝接受，这样做就可以同时解决乱序与重复的现象，因为它们分别在期待序列号之后与之前，都会被拒绝。

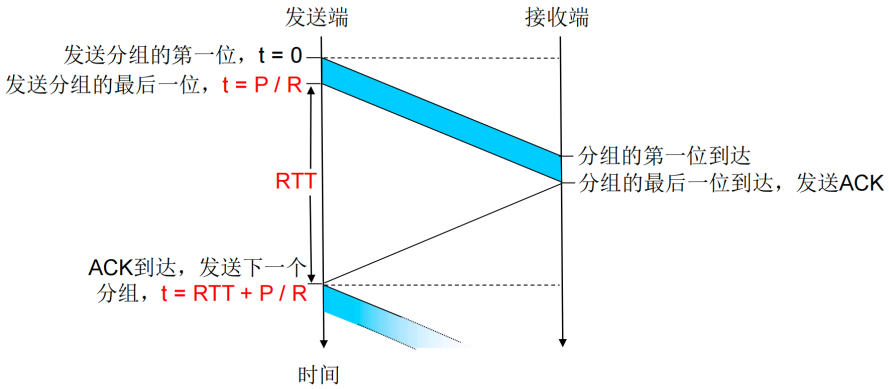

图 1.3: 协议设计：停等机制收发示意图

1.2.4 累积确认机制实现的设计

在停等机制中，当一个数据报未得到接收方确认时，发送方不能发送下一个数据报，因此存在很严重的效率问题。在 rdt3.0 以后，实现了流水线优化，两种典型的流水线协议就是我们 Lab3-2 和 3-3 中实现的**累积确认**和**选择确认**，在本节中我们先介绍累积确认的协议设计。

在 GBN 协议中，允许发送方发送多个分组而不需等待确认，发送这种分组的上限就是滑动窗口大小 N：

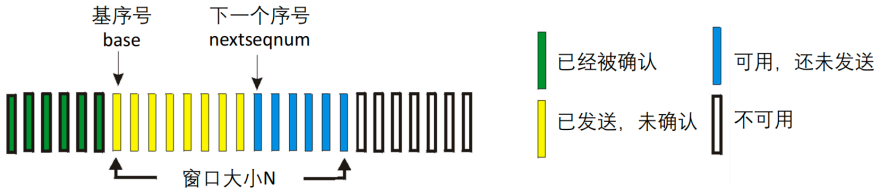

图 1.4: 协议设计：累计确认机制滑动窗口示意图

其中，基序号定义为最早的未被确认的序列号，nextseqnum 则指向下一个待发分组的序列号。可以看到，这两个序号和窗口结束的位置把整体的序号空间分为四份：

- 在基序号以前，对应于已经发送并得到确认的分组，它们无需再做任何操作
- 在基序号和 nextseqnum 之间，对应已发送但未收到确认的分组，这也是 GBN 算法能够提升效率的根本所在，相较于停等机制，GBN 算法在未收到确认之间可以继续发送一些报文段
- 在 nextseqnum 和窗口结束位置之间，即 base 和 base+N 之间，是可以发送但还未发送的报文分组
- 在窗口结束位置之外，是不可发送的分组

基于如上的 GBN 概念，我们累积确认的发送端发送逻辑如下：初始时，将基序号和 nextseqnum 都置为 0，接着不断传输数据报，每次传输后，将 nextseqnum 右移，直至达到窗口边界 (base+N) 或者全部数据报传输完毕，而当产生**超时重传时**，我们要重新发送 [base,nextseqnum] 这个范围内的全部报文。

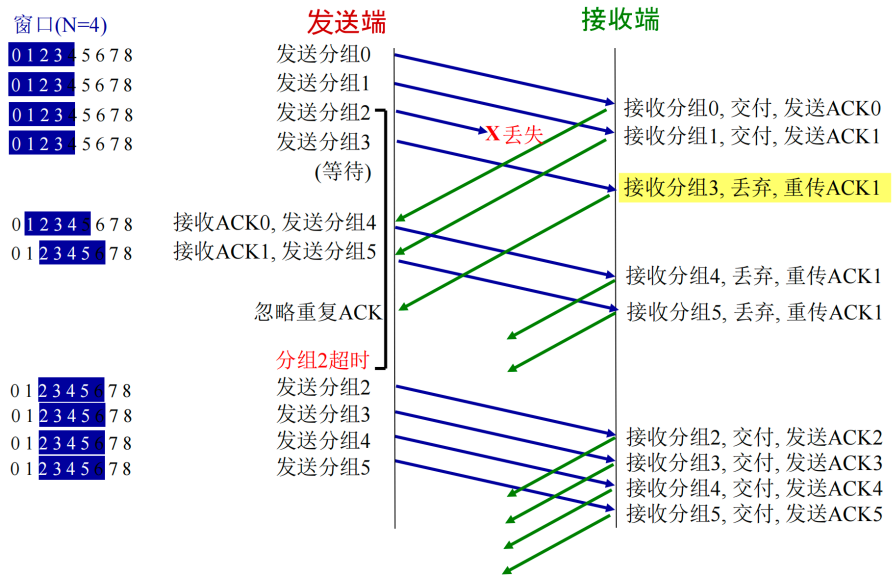

图 1.5: 协议设计：累计确认机制交互示例

在 GBN 机制中，接收端的逻辑其实并未进行多少更改，他的行为依然是判断发来的报文段是不是期待报文段，如果不是就拒绝接受并且在回复报文中告诉发送端自己期待的序列号，跟停等机制几乎一样，而在这个过程中我们也可以看到，如果产生了乱序或者丢失现象，那么发送端就会收到**很多重复且相同的 ACK**，由此引申出我们在 GBN 中实现的**快速重传算法**，即发送端收到三次重复 ACK 时，就可以假设 ACK 序列号指示的报文段丢失，就立即重传报文段而无需等待超时，进一步提升了 GBN 的效率。

1.2.5 选择确认机制实现的设计

我们看到在 GBN 机制中，接收端没有收到自己期待的序列号时就拒绝接受，这时发送端超时或者收到三次重复 ACK 时会重传 [base,nextseqnum] 之间所有的报文段，这也是一笔很大的开销，因此引入了另一种流水线算

法 SR。在选择确认中，接收端的逻辑进行了一定程度上的修改：**接收端会独立确认每个正确接收的分组，并缓存乱序分组，并且每个分组独立定时**，基于接收端的收发逻辑，**发送端只重传未收到 ACK 的分组**即可，而不用像 GBN 一样需要发送很多，可以看到，选择确认以牺牲一些接收端的存储空间为代价，换来了传输速率的提升。

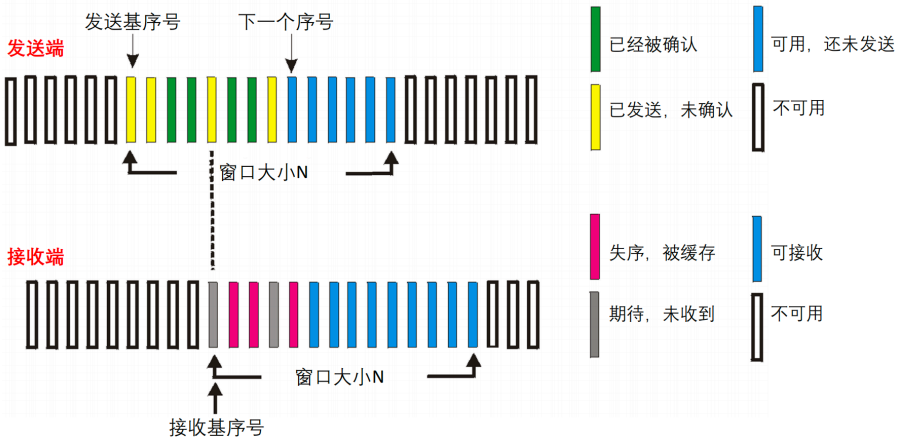

图 1.6: 协议设计：选择确认机制滑动窗口示意图

基于如上的 SR 概念，我们选择确认实现中，发送端的发送逻辑为：发送端对每一个数据报的发送建立一个线程，线程的工作是发送指定的顺序数据报，并等待接收服务端发来的 ACK，如果超出等待时间，那么重传数据报，线程当收到服务端发来的 ACK 时结束工作，并释放资源。同时，客户端应该总是维护活跃的线程数 < 窗口大小，并在每新建一个线程后隔一段时间 (sleep 若干毫秒) 再创建新的线程 (如果满足条件的话)，以确保客户端发出的报文段有序，并且始终不会超出窗口大小。

而接收端的接受逻辑也需要进行改动，因为在选择确认中，接收端不能因为发来的报文序号不是窗口开始位置的序号就拒绝接收，而是独立确认每个发送来的报文，无论是否有序。对于顺序的报文段，滑动窗口不断向前移动；而对于乱序的报文段，滑动窗口的开始位置不动，并缓存失序的报文段，当失序报文填满了接收端的滑动窗口时，接收端就拒绝接收，直到收到最大有序序号的下一个序号对应的报文段。

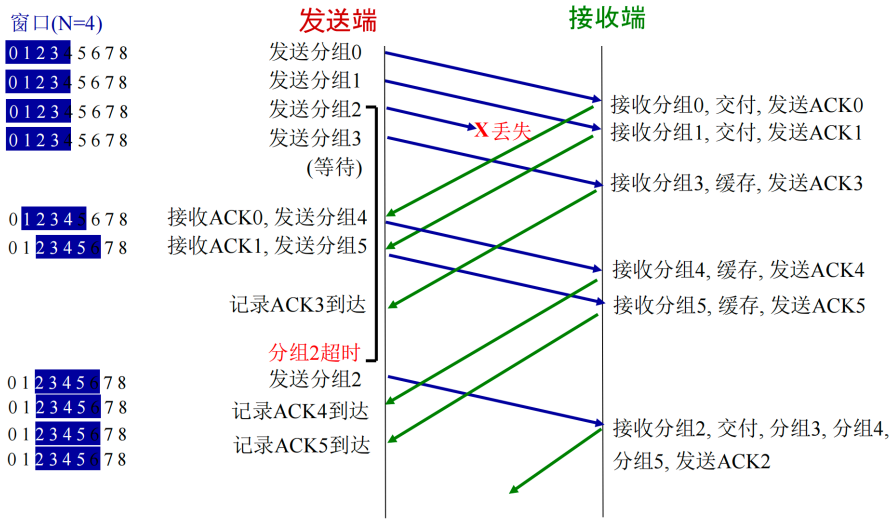

图 1.7: 协议设计：选择确认机制交互示例

至此，我们简要回顾了协议设计，主要是数据报设计，连接建立与释放的设计，以及三种不同流量控制机制的收发逻辑设计，它们具体的代码实现和进一步介绍，可以参考前面的三个报告。

第 2 章 对比实验

2.1 停等机制与滑动窗口机制性能对比

在本小节中，我们将主要进行两组实验，第一组实验固定延时为 0ms，接着不断增加丢包率；第二组固定丢包率为 2%，接着不断增加延时，横向对比停等机制与滑动窗口机制的性能。

性能测试指标依然是传输文件得到的传输时间与吞吐率，**我们对四个文件都进行了测试，为了避免冗余，本节以及后续的表格中列出的数据，都是四个文件的平均传输时间以及平均吞吐率**。

2.1.1 第一组实验及分析

设置滑动窗口的大小为 8，固定变量丢包率为 0%，接着不断增加延时，经过若干测试，停等机制和滑动窗口机制的表现结果如下表和下图所示：

流量控制机制	测量指标	延时 = 1ms	延时 = 2ms	延时 = 4ms	延时 = 8ms	延时 = 16ms
停等机制	传输时间 吞吐率	11 s 523.7 KB/s	12 s 480.0 KB/s	13 s 443.1 KB/s	18 s 320.0 KB/s	19s 303.2 KB/s
累积确认机制	传输时间 吞吐率	9 s 640.0 KB/s	10 s 576.0 KB/s	11 s 523.7 KB/s	12 s 480.0 KB/s	17 s 338.8 KB/s
选择确认机制	传输时间 吞吐率	11 s 523.7 KB/s	11 s 523.7 KB/s	12 s 480.0 KB/s	13 s 443.1 KB/s	18 s 320.0 KB/s

表 2.1: 实验一：丢包率为 0，改变延时，滑动窗口机制与停等机制性能对比

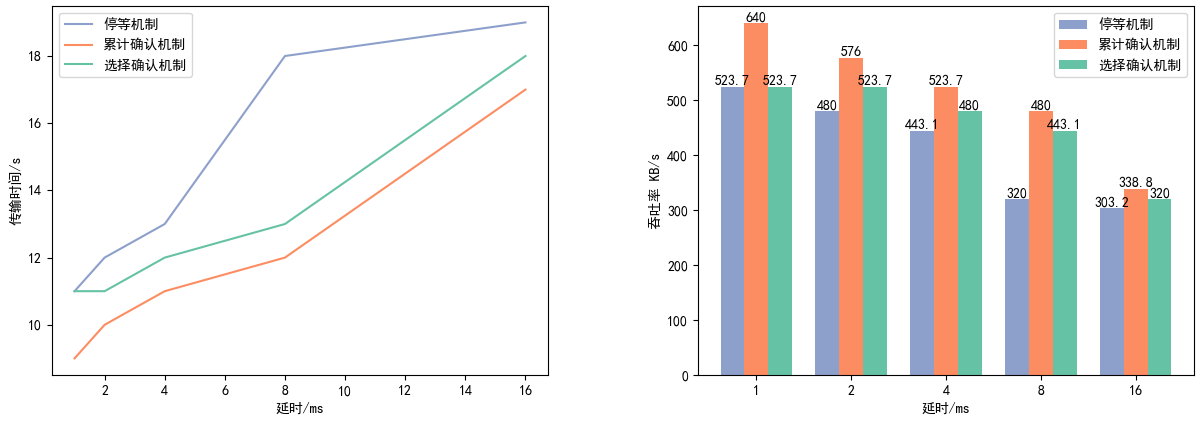

图 2.1: 实验一：丢包率为 0，改变延时，滑动窗口机制与停等机制性能对比结果图

可以看到，**两种滑动窗口机制的性能要比停等机制好**，表现为传输时间短，吞吐率大，这是因为两种滑动窗口机制是基于流水线的，并行效率高，对信道的利用率高，因此有更优秀的性能，在下面的实验中我们也可以看到，当丢包率进一步提升时，流水线算法的优越性还会进一步体现。

同时注意到选择确认的性能不如累积确认，这主要是丢包率太小导致的，在 Lab3-3 的设计中，我们为了让线程有序发送报文段，采取了让主线程创建线程后休眠一段时间的设计，因此会拖慢选择确认的效率，在后续实验中我们可以看到，当丢包率进一步增大时，选择确认的效率会比累积确认高。

2.1.2 第二组实验及分析

设置滑动窗口的大小为 8，固定变量延时为 0ms，接着不断增加丢包率，经过若干测试，停等机制和滑动窗口机制的表现结果如下表和下图所示：

流量控制机制	测量指标	丢包率 = 1%	丢包率 = 2%	丢包率 = 4%	丢包率 = 8%	丢包率 = 10%
停等机制	传输时间 吞吐率	14 s 411.4 KB/s	18 s 320.0 KB/s	30 s 192.0 KB/s	59 s 97.6 KB/s	70 s 82.3 KB/s
累积确认机制	传输时间 吞吐率	16 s 360.0 KB/s	19 s 303.2 KB/s	21 s 274.3 KB/s	24 s 240.0 KB/s	26 s 221.5 KB/s
选择确认机制	传输时间 吞吐率	18 s 320.0 KB/s	20 s 288.0 KB/s	21 s 274.3 KB/s	23 s 250.4 KB/s	24 s 240.0 KB/s

表 2.2: 实验二：延时为 0，改变丢包率，滑动窗口机制与停等机制性能对比

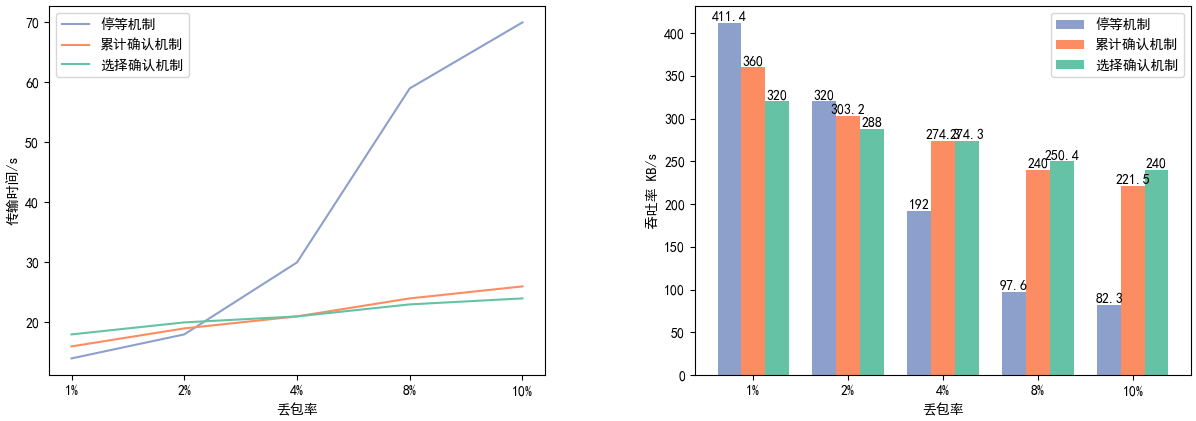

图 2.2: 实验二：延时为 0，改变丢包率，滑动窗口机制与停等机制性能对比结果图

可以看到，随着丢包率的增加，停等机制的表现越来越差，传输时间的上升趋势很明显，并且丢包率从 1% 增加至 10% 时，吞吐率低了近六倍，而两种滑动窗口机制的吞吐率虽然也有一定程度的下降，但整体表现的比停等机制平稳的多，整体表现与加速效果也更好。

同时我们看到，在丢包率较小时，累计确认机制的效率较高于选择确认机制，其中原因之一还是由于我们线程有序性管理造成的一些性能损失，此外，还要归功于我们在累计确认机制中实现的快速重传，它可以帮我们省去一些等待时间。但当传输文件更大或者丢包率更大的时候，选择确认的效率就要优于累积确认了，因为选择确认是单个数据报的收发，而累积确认还是要重发区间内的所有报文，我们会在后面的实验中验证这一点。

2.2 滑动窗口机制中不同窗口大小对性能的影响

在本小节中，我们将固定延时为 2ms，窗口大小为 8，丢包率分别为 0%，2%，4%，6%，8%，然后不断增加窗口大小，对比累积确认算法和选择确认算法随着窗口大小的变化对性能的影响。

2.2.1 累计确认

累积确认机制在固定丢包率和延时的情况下，改变窗口大小得到的表现性能如表2.3所示，可以看到，无论哪一组丢包率，**随着窗口大小的增加，累计确认机制均表现为传输时间越来越短，并且吞吐率越来越大**，这是因为窗口越大，并行化程度就越大，信道利用就会越充分，传输效率越好。

窗口大小	测量指标	丢包率 = 1%	丢包率 = 2%	丢包率 = 4%	丢包率 = 6%	丢包率 = 8%
N = 4	传输时间 吞吐率	16 s 360.0 KB/s	20 s 288.0 KB/s	28 s 205.7 KB/s	50 s 115.2 KB/s	61 s 94.4 KB/s
N = 6	传输时间 吞吐率	15 s 384.0 KB/s	19 s 303.2 KB/s	23 s 250.4 KB/s	32 s 180.0 KB/s	39 s 147.7 KB/s
N = 8	传输时间 吞吐率	14 s 411.4 KB/s	19 s 303.2 KB/s	21 s 274.3 KB/s	25 s 230.4 KB/s	29 s 198.6 KB/s
N = 10	传输时间 吞吐率	13 s 443.0 KB/s	16 s 360.0 KB/s	21 s 274.3 KB/s	24 s 240.0 KB/s	27 s 213.3 KB/s
N = 12	传输时间 吞吐率	12 s 480.0 KB/s	14 s 411.4 KB/s	18 s 320.0 KB/s	22 s 261.8 KB/s	24 s 240.0 KB/s

表 2.3: 累积确认：固定丢包率和延时，不同窗口大小的性能表现

同时注意到，当丢包率低时，相较于停等机制，累计确认机制的加速效果并不明显；N=4 时，即窗口大小较小时，随着丢包率的增加，传输时间的变化趋势也和停等机制差不多，这主要是因为 **N=4 时很难发生快速重传**(因为快速重传就占了三个窗口大小)，因此重传原因几乎都是因为超时，那么这个代价就比较昂贵，并且每个数据报都要等待一个 2ms 的延时；当窗口大小变大的时候，快速重传的作用很明显，比如到了 N=6 时，丢包率较大也不会引起传输时间的爆炸增长，更明显的现象如下图所示：

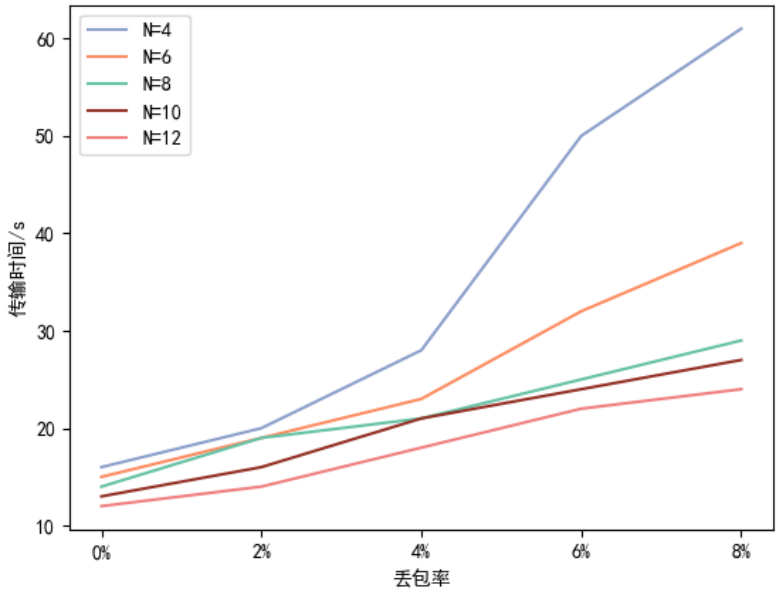

图 2.3: 固定丢包率和延时，不同窗口大小的性能表现结果图

2.2.2 选择确认

选择确认机制在固定丢包率和延时的情况下，改变窗口大小得到的表现性能如表2.4所示，可以看到，依然存在我们之前所说的，丢包率小时由于线程有序性而导致效率降低不如 GBN 的情况，但当丢包率较大时，选择确认的性能就会比累积确认更高，更直观的对比图将在下一节中给出，我们还是先来看选择确认机制中窗口大小如何影响性能。

各窗口大小在不同组的表现图如图2.4所示，沿着纵轴向下看，可以看到选择确认这种滑动窗口机制中，**传输效率也是随着窗口大小的增加而提升的**，这个提升归功于**发送端窗口：允许在未收到 ACK 时接着发送，提高**

窗口大小	测量指标	丢包率 = 1%	丢包率 = 2%	丢包率 = 4%	丢包率 = 6%	丢包率 = 8%
N = 4	传输时间 吞吐率	20 s 288.0 KB/s	23 s 250.4 KB/s	28 s 205.7 KB/s	41 s 140.5 KB/s	52 s 110.8 KB/s
N = 6	传输时间 吞吐率	18 s 320.0 KB/s	21 s 274.2 KB/s	24 s 240.0 KB/s	30 s 192.0 KB/s	37 s 155.7 KB/s
N = 8	传输时间 吞吐率	18 s 320.0 KB/s	20 s 288.0 KB/s	22 s 261.8 KB/s	23 s 250.4 KB/s	26 s 221.5 KB/s
N = 10	传输时间 吞吐率	15 s 384.0 KB/s	17 s 338.8 KB/s	20 s 288.0 KB/s	22 s 261.8 KB/s	24 s 240.0 KB/s
N = 12	传输时间 吞吐率	14 s 411.4 KB/s	15 s 384.0 KB/s	18 s 320.0 KB/s	20 s 288.0 KB/s	23 s 250.4 KB/s

表 2.4: 选择确认：固定丢包率和延时，不同窗口大小的性能表现

了并行性、以及**接收端窗口：对每个分组独立接收，并缓存乱序报文段，发送端只需发送未收到 ACK 的分组而不用重传窗口中的所有报文，降低了重传损耗**。并且我们在选择确认机制中没有快速重传而选择确认在丢包率较大时效率由于累计确认，这说明发送端缓存乱序报文段带来的提升要比快速重传大。

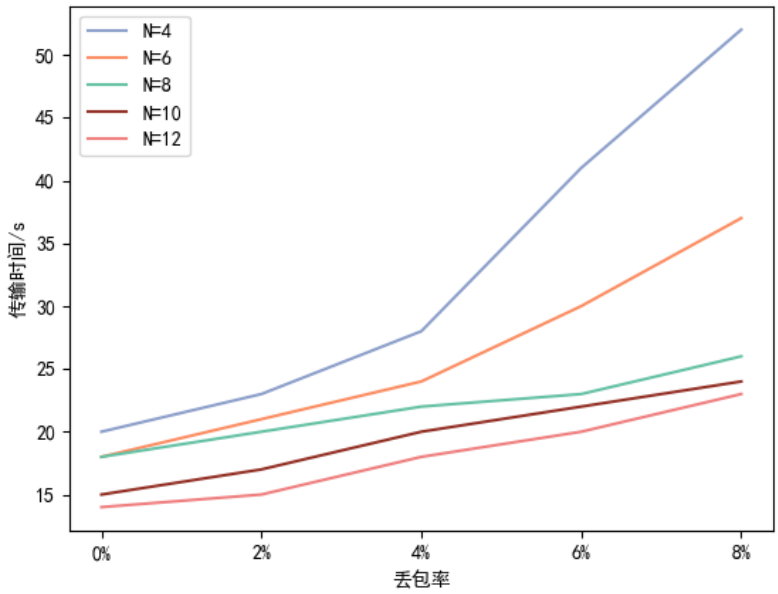

图 2.4: 固定丢包率和延时，不同窗口大小的性能表现结果图

2.3 累计确认和选择确认的性能比较

我们选取上一小节中丢包率为 1，2，4，8 的四列 (其中前两列为低丢包率，高两列为高丢包率) 为代表，绘制簇形柱状图来比较二者的性能差异，四张对比图2.5如图所示 (**选择确认为红色，累计确认为蓝色**)：

挑选了四列典型数据，这下可以清晰的看到累计确认和选择确认之间的性能对比，我们在前面也说过很多次，由于实现时要控制线程的有序性，选择确认牺牲了一些性能，因此在低丢包率时对比累计确认没有明显的优势，**但当丢包率增大时，选择确认的性能明显优于累计确认**，这是以牺牲接收端的存储空间为代价的。

此外，在 TCP 的实际实现中，丢包率较低或者文件大小较小、延时较低时，选择确认的性能并不一定会比累积确认低，我们实验的结论并不是哪种滑动窗口机制一定比谁更好，而是：**在这种低丢包率和相对稳定的网络环境下，累积确认可能已经足够有效，并且具有较低的开销**，这时我们可以根据实际情况选择不使用选择确

认机制，这样可以节省一些用户的空间。**但在高丢包率、拥塞或不稳定的网络环境中，选择确认可以更精确地处理丢失的数据包，提高网络的效率和可靠性**。

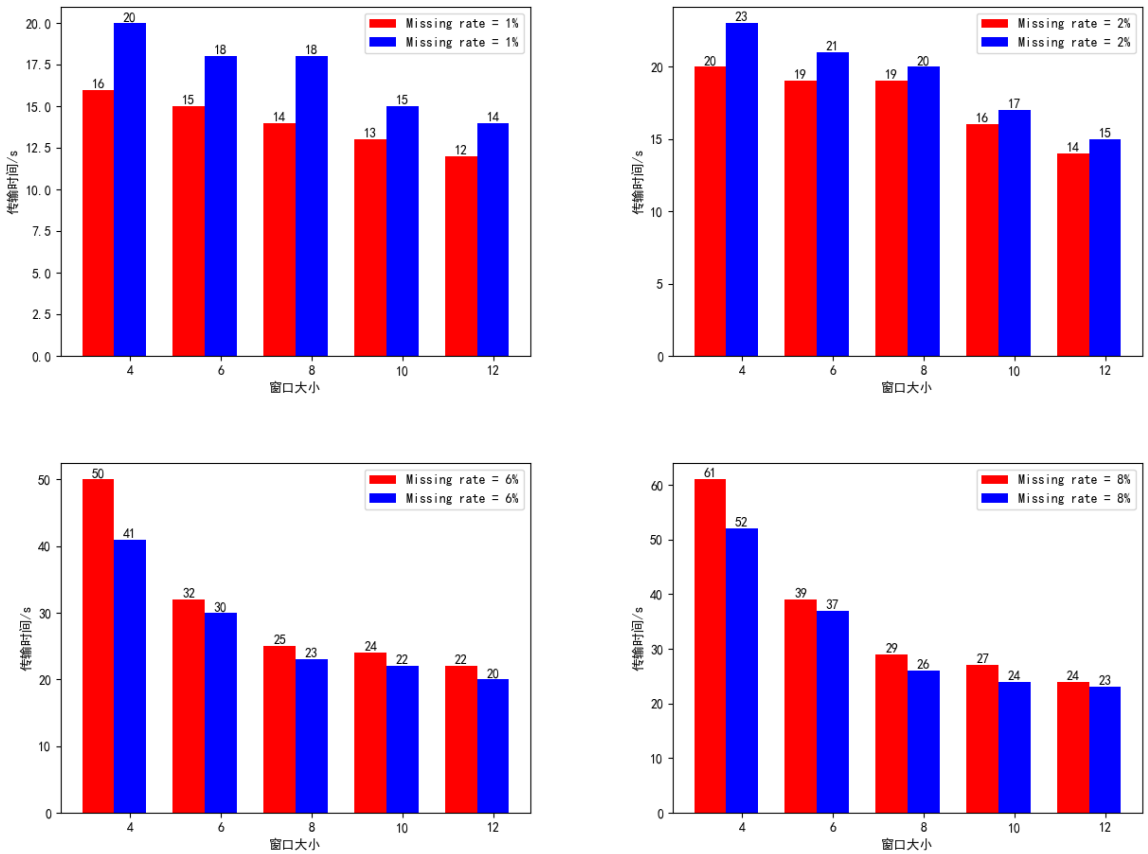

图 2.5: 累计确认和选择确认在相同窗口大小下的性能对比 (不同丢包率情况下) 示意图