计算机网络实验6-对比实验

2012039 冯朝芃

实验概述

使用控制变量法，基于给定的实验测试环境，通过改变延迟时间和丢包率，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。

这次实验中无特别说明均采用大小适中的2.jpg作为测试文件，被测程序不大量输出日志。

停等机制和滑动窗口对比

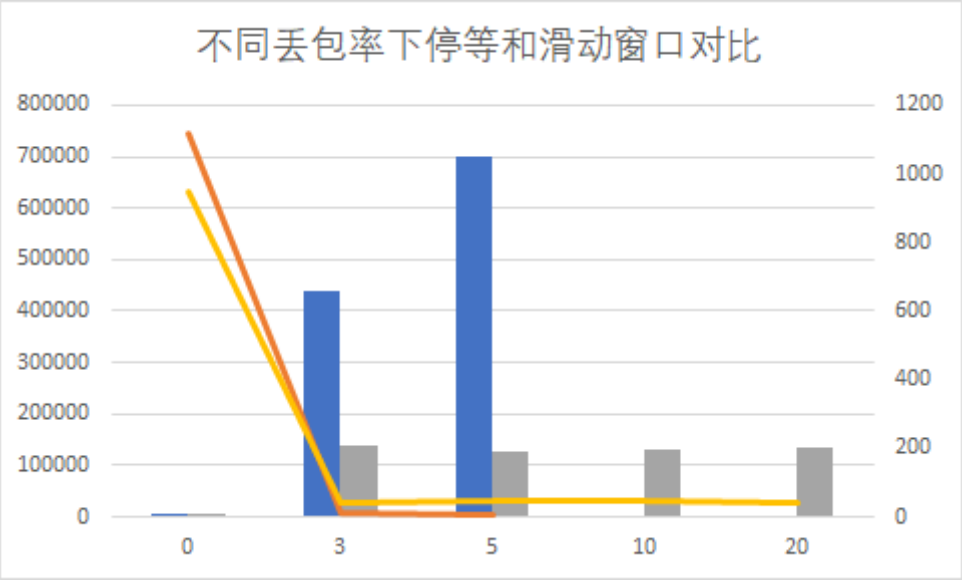
滑动窗口发送程序采用GBN算法，设置窗口大小为65536Bytes。

不设置时延控制丢包率：

实验结果展示表：

丢包率（%）	**0**	**3**	**5**	**10**	**20**
停等时间（sec）	7.929	658.126（超时180次）	1051.73（超时303次）	/	/
停等吞吐率（Bytes/sec）	743915	8962.58	5608.38	/	/
滑动窗口时间（sec）	9.327	209.239	190.28	196.778	202.514
滑动窗口吞吐率（Bytes/sec）	632412	28190.3	31018.8	29975.4	29126.4

使用图表展示上述结果：



上图中，折线反映的是吞吐率变化情况，柱状图反应传输时间。

性能分析：

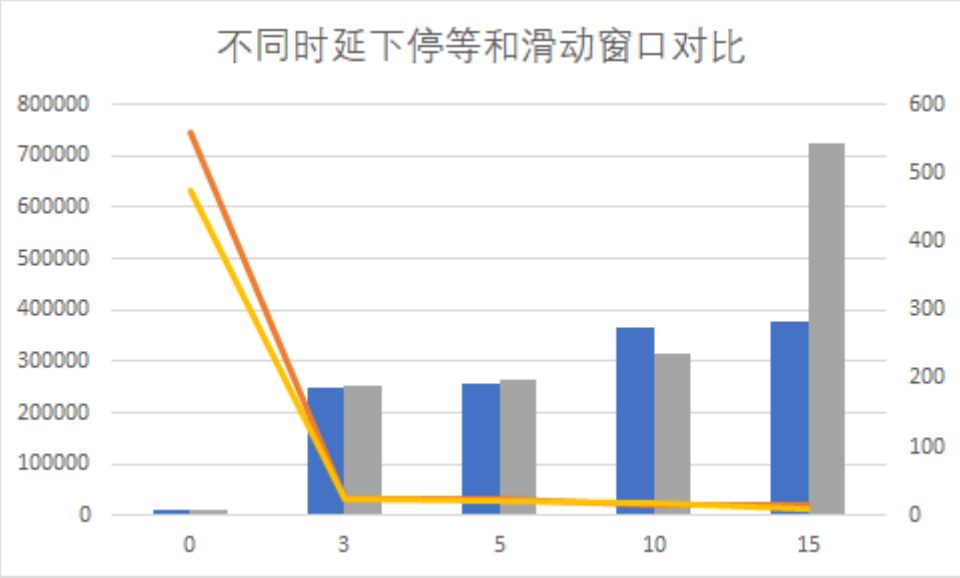
对上述结果进行分析，可以看到在没有时延、丢包的情况下，停等机制因为代码简单、运行较快，在传输吞吐率上略胜一筹。然而一旦引入丢包，停等机制因为频繁超时重传导致效率严重下降，在超时时间均为3s的情况下，两种方法的吞吐率差异显著，滑动窗口明显更好。

停等机制因为耗时过长，“/”部分表示没有进行测试。

不设置丢包率控制时延：

实验结果展示表：

时延（ms）	0	3	5	10	15
停等时间（sec）	7.929	185.61	192.978	274.452	281.556
停等吞吐率（Bytes/sec）	743915	31779	30565.7	21491.9	20949.7
滑动窗口时间（sec）	9.327	189.976	198.587	234.738	544.695（176次重传）
滑动窗口吞吐率（Bytes/sec）	632412	31048.7	29702.4	25128	10829



上图中，折线反映的是吞吐率变化情况，柱状图反应传输时间。

性能分析：

由上图可知，在不引入丢包而只有时延的情况下，停等和滑动窗口的传输效率不相上下。但随着时延的增大，由于滑动窗口对时延较敏感（本人的实现中精确记录了每个窗口中数据包的发送时间），频繁的出现超时重传，且滑动窗口采用GBN方法，重传数据量大，因而效率明显下降。

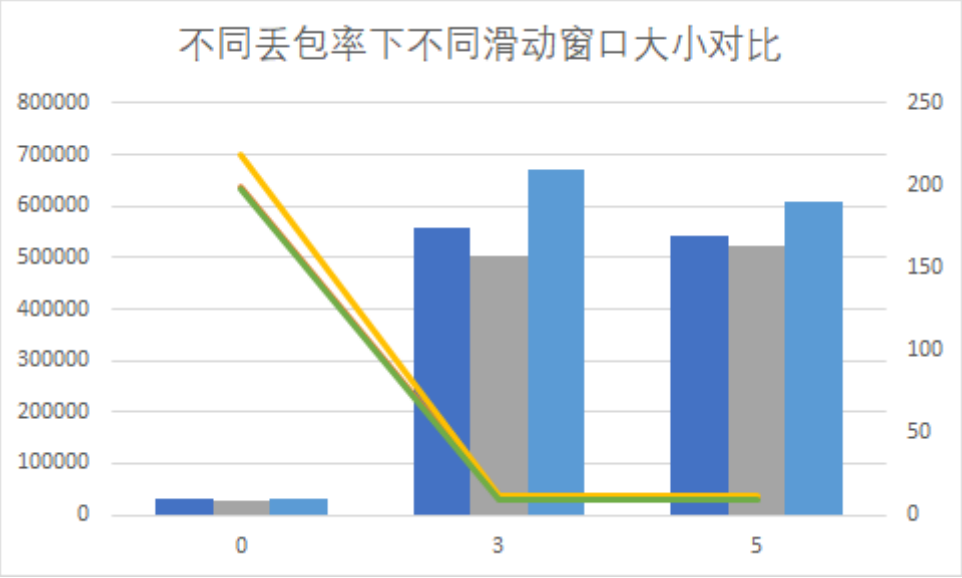
滑动窗口机制中不同窗口大小对性能的影响

滑动窗口发送程序采用GBN算法，窗口大小取16384、32768、65536Bytes进行比较。

不设置时延控制丢包率：

实验结果展示表：

丢包率（%）	0	3	5
滑动窗口16384时间（sec）	9.227	174.942	169.638
滑动窗口16384吞吐率（Bytes/sec）	639266	33716.9	34771.1
滑动窗口32768时间（sec）	8.402	157.69	163.292
滑动窗口32768吞吐率（Bytes/sec）	702036	37405.7	36122.4
滑动窗口65536时间（sec）	9.327	209.239	190.28
滑动窗口65536吞吐率（Bytes/sec）	632412	28190.3	31018.8



上图中，折线反映的是吞吐率变化情况，柱状图反应传输时间。

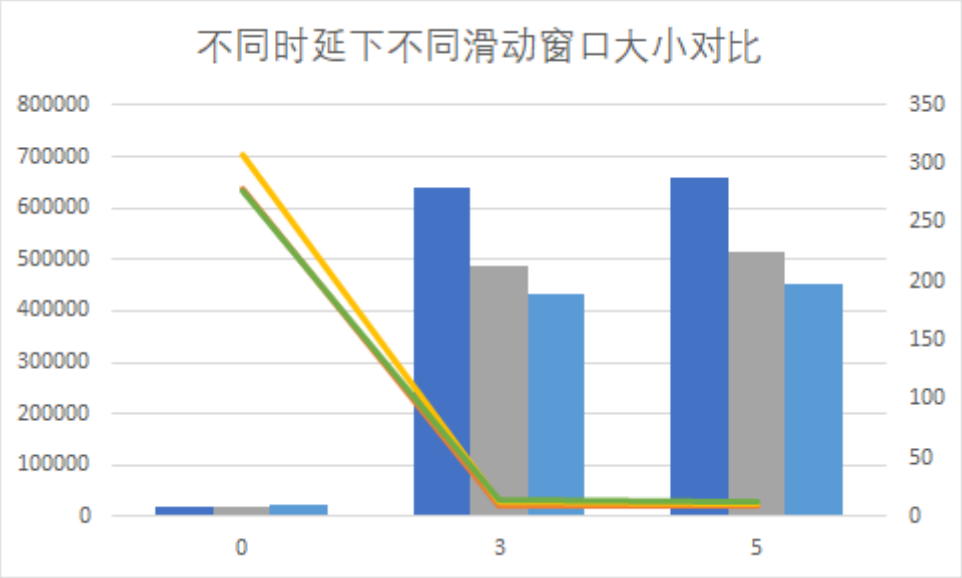
性能分析：

发现不论丢包率是否为0，传输吞吐率对于滑动窗口大小并非有某种线性对应的关系，从图中容易得知，无时延仅加入丢包时，当窗口取32768时传输效率最高。这也就启发我们应寻找最适合当前网络状况的窗口大小以获得最佳的传输速率。

不设置丢包率控制时延：

实验结果展示表：

时延（ms）	0	3	5
滑动窗口16384时间（sec）	9.227	279.912	288.944
滑动窗口16384吞吐率（Bytes/sec）	639266	21072.7	20414
滑动窗口32768时间（sec）	8.402	213.266	224.626
滑动窗口32768吞吐率（Bytes/sec）	702036	27658	26259.2
滑动窗口65536时间（sec）	9.327	189.976	198.587
滑动窗口65536吞吐率（Bytes/sec）	632412	31048.7	29702.4



上图中，折线反映的是吞吐率变化情况，柱状图反应传输时间。

性能分析：

GBN算法对时延是较为敏感的，从图中可知，对于不同的时延，窗口越大传输效率越高，更大的窗口有利于一次发送更多的数据、提升传输效率。

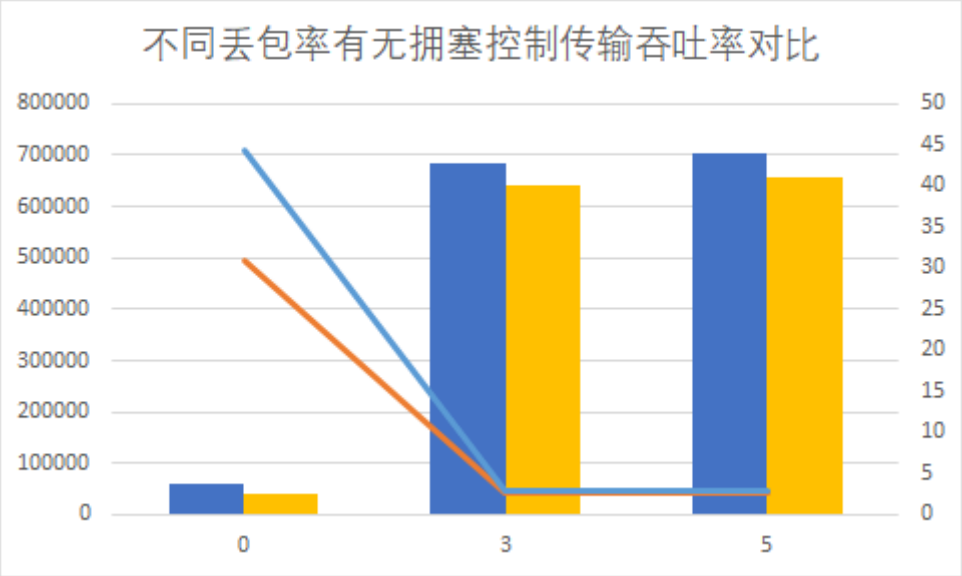
但窗口也不能无限增大，由于路由器转发数据本身有一定的时延，如果窗口过大，等窗口中所有数据都发送完，最开始发送的数据包已经超时，导致重传，如此往复，可能导致性能降低。

有拥塞控制和无拥塞控制的性能比较

这一部分使用给出的1.jpg文件进行测试。固定窗口版本设置窗口大小为65536Bytes。

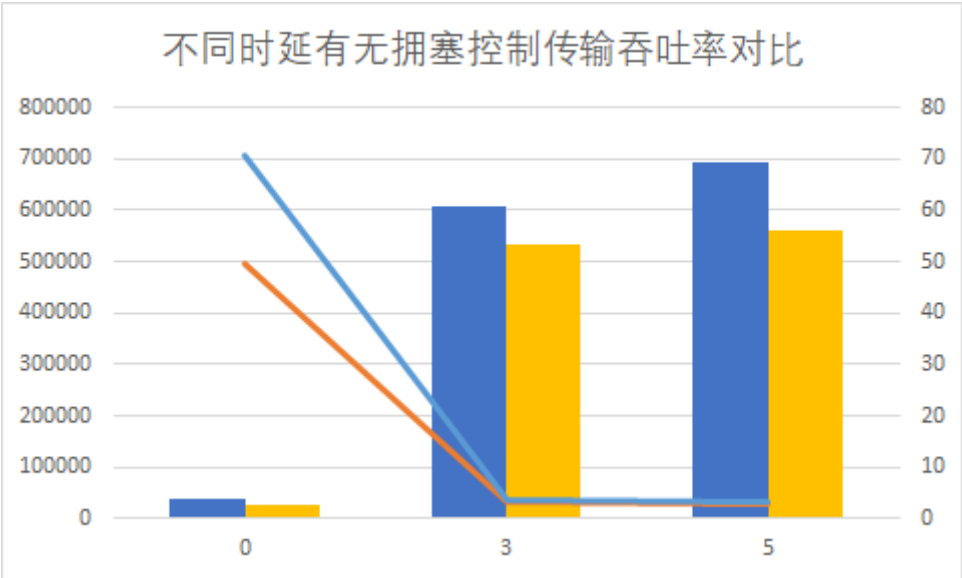
不设置时延控制丢包率：

丢包率（%）	0	3	5
有拥塞控制时间（sec）	3.755	42.797	44.045
有拥塞控制吞吐率（Bytes/sec）	494635	43399.1	42169.4
无拥塞控制时间（sec）	2.624	40.004	40.959
无拥塞控制吞吐率（Bytes/sec）	707833	46429.2	45346.6



不设置丢包率控制时延：

时延（ms）	0	3	5
有拥塞控制时间（sec）	3.755	60.913	69.357
有拥塞控制吞吐率（Bytes/sec）	494635	30491.9	26779.6
无拥塞控制时间（sec）	2.624	53.506	55.986
无拥塞控制吞吐率（Bytes/sec）	707833	34713	33175.3



性能分析：

不难发现，在任何一种时延或丢包率之下，有拥塞控制的效果总是比没有的略差一些，这是因为本次实验使用的固定窗口大小65536Bytes，基本上就是可变窗口能够跑到的最大的拥塞控制窗口大小了，之后就会因为丢包、超时等导致拥塞控制窗口变小。在恢复过程中，拥塞控制窗口急剧减小，由于使用RENO+GBN的实现机制，重传的数据量较大，这样容易引起重复ACK再次触发重传，如此往复一段时间后虽然也能恢复到拥塞避免阶段，但效率无疑更低。这样固定窗口算法窗口大小保持恒定，加入拥塞控制后窗口大小波动，效率自然下降。