深 圳 大 学 实 验 报 告

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | 计算机网络 | | |
| 实验名称 | 实验4：拥塞控制算法 | | |
| 学院 | 计算机与软件学院 | | |
| 专业 | 软件工程（腾班） | | |
| 指导教师 | 张磊 | | |
| 报告人 | 黄亮铭 | 学号 | 2022155028 |
| 实验时间 | 2024年5月15日 | | |
| 提交时间 | 2024年5月21日 | | |

教务处制

# 实验目的与要求

1. 理解数据传输过程中可能会发生拥塞
2. 了解数据传输过程中发生拥塞时的现象和表现
3. 理解拥塞控制算法的基本思想
4. 理解不同拥塞控制算法的基本工作原理和区别
5. 理解和掌握拥塞控制算法；
6. 基于提供的资料和代码，完成任务要求；
7. 按照任务要求进行实验，并按要求绘制实验结果图；
8. 对实现代码和实验结果进行截图展示；
9. 撰写实验报告。

# 实验过程

任务1：拥塞现象观察

任务目标及要求

1. 理解拥塞产生的原因，并观察拥塞所带来的现象；
2. 学会使用OMNet++模拟器，对比不同网络状态下数据传输的表现，并通过作图进行分析；
3. 利用OMNet++模拟器模拟数据传输，记录不使用拥塞算法时，数据传输在不同网络环境下的性能表现（丢包率（LossRate）以及有效吞吐量（GoodPut））；
4. 在多种网络配置下进行对比实验，并将实验数据作图展示，进行分析；

任务步骤

**Step1：安装OMNet++模拟器与INET框架，并新建项目**

1. 下载 OMNet++ 与 INET。



图1：压缩包下载成功

1. 具体安装步骤请参考 安装文档。

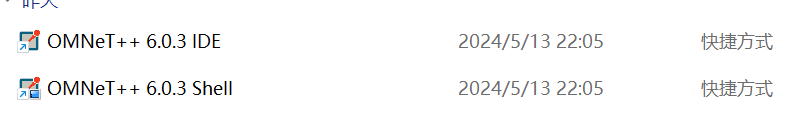
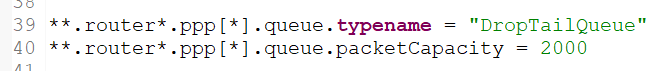


图2：软件安装成功

**Step2：运行项目并观察拥塞现象**

观察拥塞现象并分析原因：在不同网络配置和传输配置下，观察不适用拥塞控制算法时，数据传输过程中Lossrate与Goodput的变化，并回答下述问题：

* 保持上述的链路参数配置与传输参数配置不变（即默认参数），此时的Lossrate与Goodput分别是多少？
  1. 将实验参数配置修改成与实验文档一致。



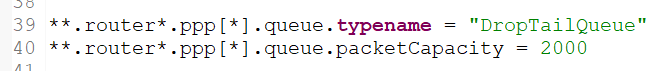


图3：omnet.ini文件

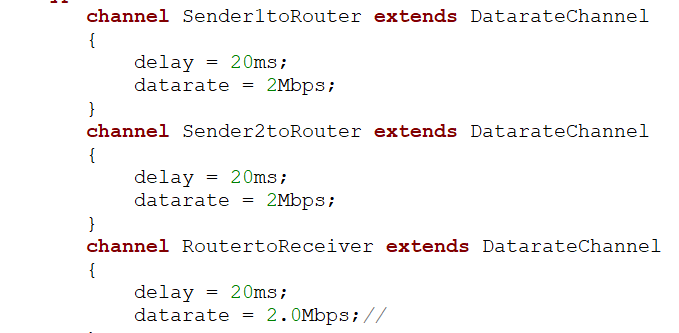
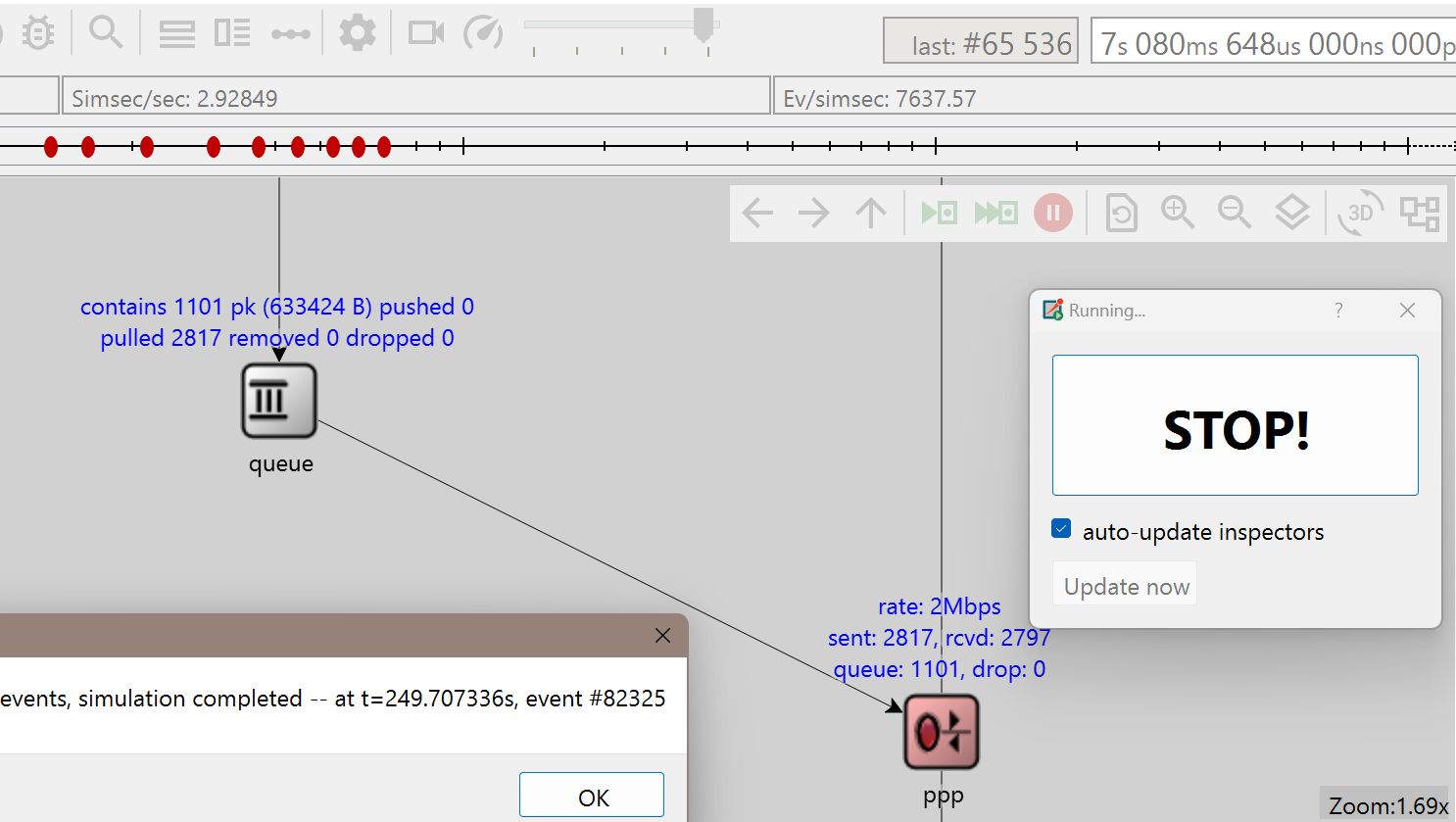


图4：package.ned文件

* 1. 运行模拟器。



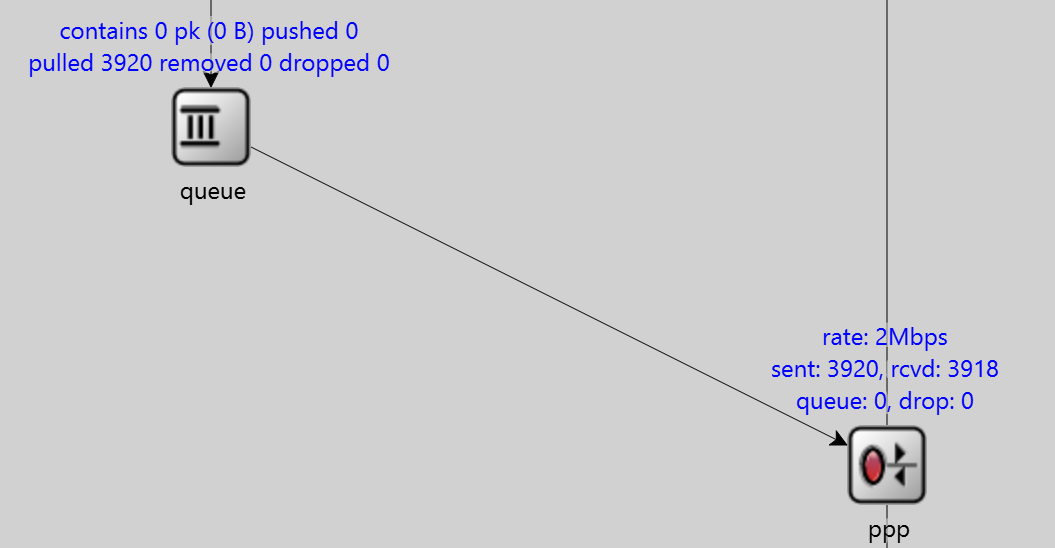


图5：模拟器运行结果

* 1. 计算得到结果。
* 其他配置不变，将路由器的队列长度依次更改为[1800，1600，1400，1200，1000]。此时Lossrate与Goodput随着队列长度的降低发生了什么变化？为什么会这样？
  1. 更改实验参数配置：将路由器的队列长度依次更改为[1800，1600，1400，1200，1000]。
  2. 运行模拟器。

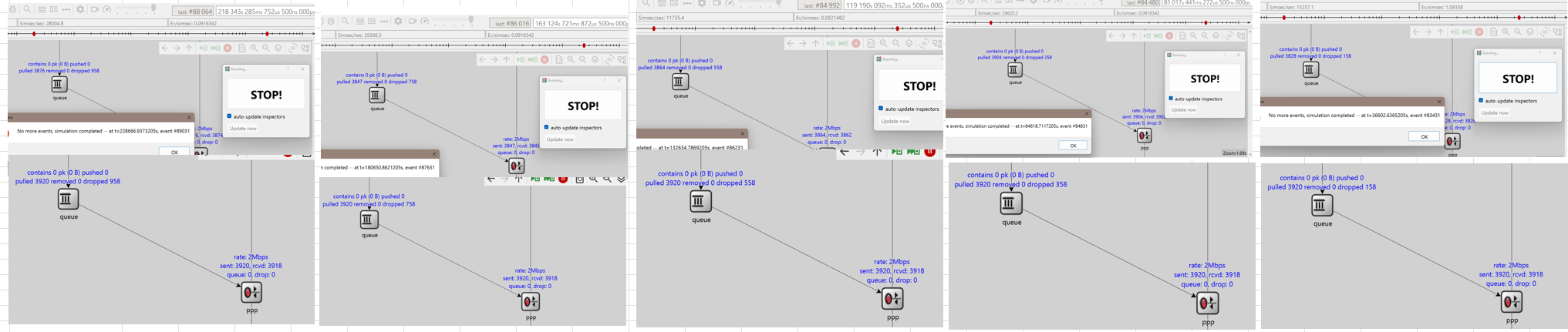


图6：模拟器运行结果

* 1. 实验结果列表。

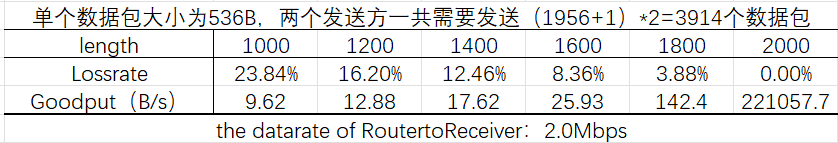


图7：实验结果

* 1. 实验结果可视化。

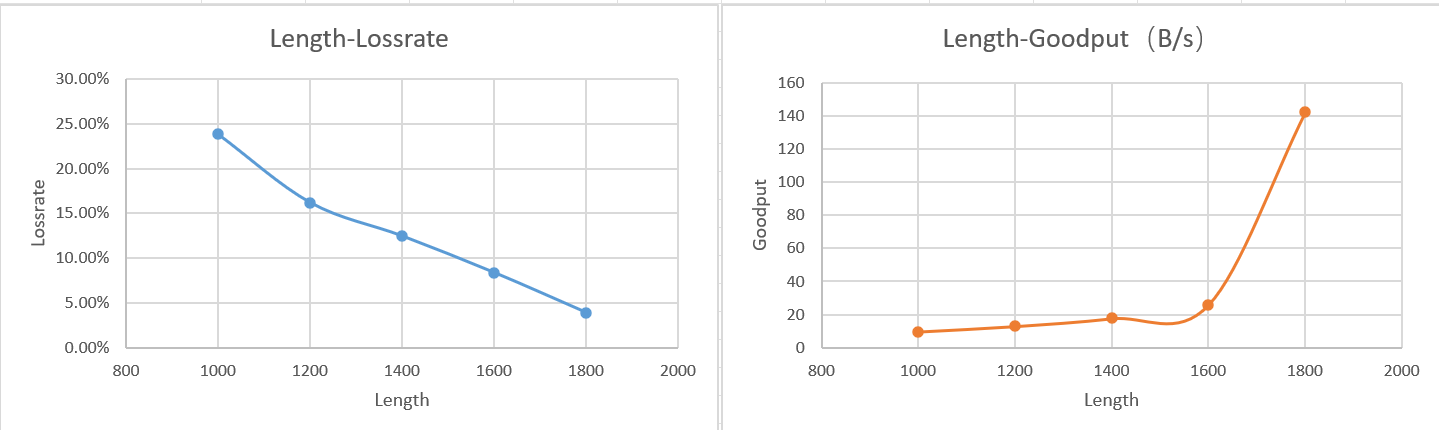


图8：可视化结果

* 1. 分析。

由④中的可视化结果可知：**Lossrate随着路由器队列长度的降低而增高，呈线性上升趋势；Goodput随着路由器队列长度的降低而降低，呈指数下降趋势。**

可能的原因：**对于Lossrate，**路由器队列用于存储数据包，当路由器处理流量时，队列中的数据包等待被转发。如果队列长度较大，能够存储更多的数据包，从而在高流量情况下减少丢包现象。当队列长度减少时，存储数据包的容量变小。在高流量情况下，队列更容易被填满，一旦队列满了，新来的数据包就会被丢弃。因为每减少一个队列单元，就减少了一个数据包存储位置，所以这包率随队列长度减少而增加的关系基本符合是线性的。**对于Goodput，**当队列长度减少，丢包率增加，网络需要更多的重传来补偿丢失的数据包。重传占用了额外的带宽，导致有效数据传输速率下降。此外，TCP协议具有拥塞控制机制，当丢包率增加时，TCP会降低发送速率来适应网络条件。这导致有效吞吐量急剧下降。

* 将路由器的队列长度更改回默认值，并依次更改RoutertoReceiver（router与receiver之间的链路）的datarate为[1.8Mbps，1.6Mbps，1.4Mbps，1.2Mbps，1.0Mbps]。Lossrate与Goodput是如何变化的？为什么？
  1. 更改实验参数配置：更改RoutertoReceiver（router与receiver之间的链路）的datarate为[1.8Mbps，1.6Mbps，1.4Mbps，1.2Mbps，1.0Mbps]。
  2. 运行模拟器。

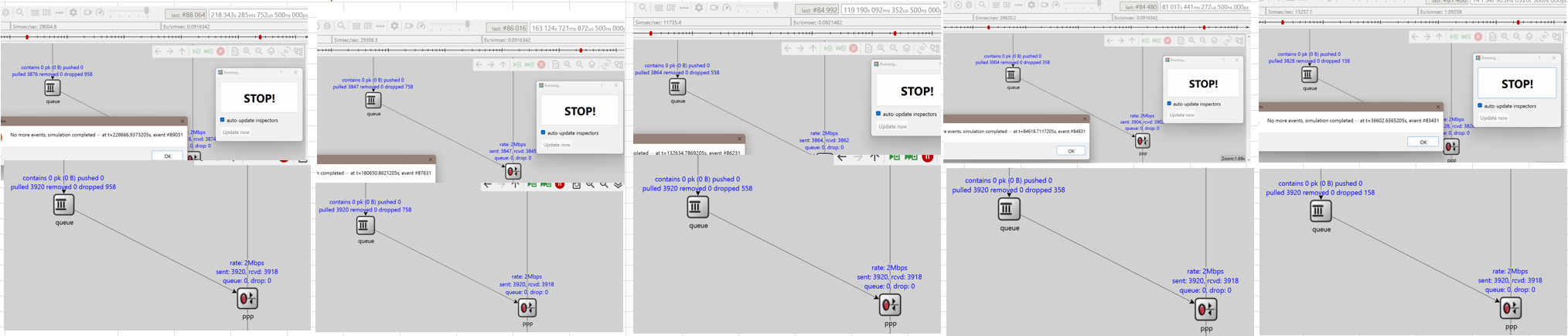


图9：模拟器运行结果

* 1. 实验结果列表。

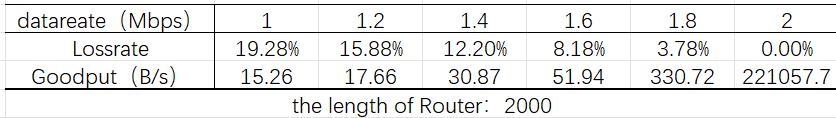


图10：实验结果

* 1. 实验结果可视化。

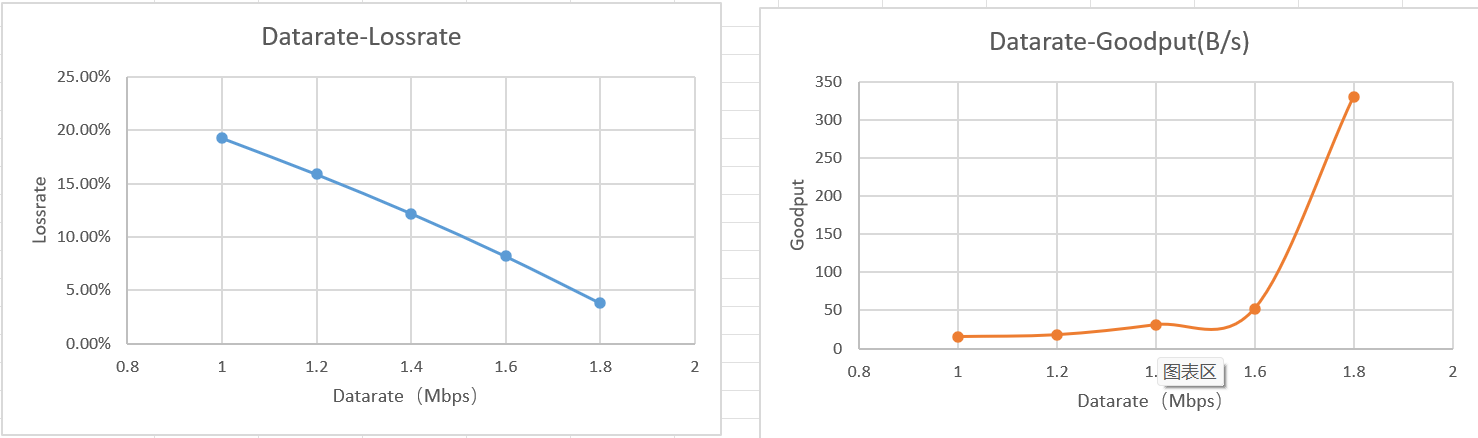


图11：可视化结果

* 1. 分析。

由④中的可视化结果可知：**Lossrate随着datarate的下降而上升，呈线性上升趋势；Goodput随着datarate的下降而下降，呈指数下降趋势。**

可能的原因：**对于Lossrate，**链路速度决定了数据包传输的速率。较低的链路速度在单位时间内传输的数据包更少，增加了队列中的数据包的积压，提高了丢包的可能性。当链路速度下降时，数据包传输变慢，队列中的数据包等待时间变长，容易导致队列溢出（队列满）。这种情况下，新的数据包将会被丢弃。由于每单位时间内传输的数据包数量减少，导致丢包率增加。因为链路速度每减少一定比例，会相应地增加丢包的概率，所以这种趋势通常是线性的。**对于Goodput，**Goodput是成功传输的有用数据的速率，不包括重传的数据包。它反映了网络的实际有效传输性能。当链路速度下降时，传输速率降低，Goodput也会相应下降。链路速度下降导致丢包率上升，网络需要更多的重传来补偿丢失的数据包。重传占用了额外的带宽，进一步降低有效数据传输速率。TCP协议的拥塞控制机制在丢包率增加时，会显著降低发送速率，以适应链路的传输能力。由于TCP的拥塞控制调整速率通常是指数级的，这导致Goodput下降呈现指数趋势。

总结

产生拥塞的原因：**① 链路速度不足。**链路速度决定了数据传输的最大速率。当流量数据超过链路的处理能力时，数据包会在路由器的队列中等待，造成拥塞。如果链路速度太低而数据太多会导致队列溢出，数据包被会丢弃，网络中重传请求增加，占用带宽，进一步加剧拥塞。**② 高流量负载。**如果网络中数据量突然增加或者数据量持续高于链路传输能力，容易导致路由器队列迅速积满，导致拥塞。**③ 不合理的队列长度。**如果队列设置过短，容易导致队列溢出，增加丢包率，进而引起重传，加剧网络拥塞。如果队列设置过长，可能会增加延迟。

任务二：实现TCP-reno拥塞控制算法

任务目标及要求

1. 实现经典的TCP-reno拥塞控制算法；
2. 利用OMNet++模拟器模拟数据传输，观察不同场景下传输数据时，TCP-reno的性能表现，并作图进行分析；
3. 理解并实现TCP-reno，解释reno中每个阶段的开始与退出条件；
4. 在多种网络配置下进行对比（与任务1的结果进行对比）实验，并将实验数据作图展示，进行分析；

任务步骤

**Step1：阅读并理解TCP-reno的工作流程。**

理解TCP-reno的设计思想，以及为什么在原有的慢启动(Slow Start)和拥塞避免(Congestion Avoidance)中加入了快速重传(Fast Retransmit)和快速恢复(Fast Recovery)算法？

在传统的TCP拥塞控制中，当一个数据包丢失时，发送方需要等待一个超时事件后才会重新发送丢失的数据包。这段等待时间可能会显著降低传输效率。**因此引入快速重传机制。**当发送方收到三个重复的ACK时，立即重传认为丢失的数据包，而不必等待超时。这大大缩短了丢包检测和重传的时间，提高了数据传输效率。

传统的TCP拥塞控制在检测到丢包后会将拥塞窗口重置为1个MSS，并重新进入慢启动阶段。**①** 这种剧烈的拥塞窗口减小和慢启动过程会导致传输速率大幅波动，从而影响网络性能。**②** 此外，发送方仍然能接收来自接收方的确认信息，说明网络拥塞不是十分严重，因此不必将拥塞窗口置为1个MSS，重新启动慢启动过程。**因此引入快速回复机制。**快速恢复机制在检测到丢包后，不是直接进入慢启动，而是将慢启动阈值设置为当前拥塞窗口的一半，并将拥塞窗口设为慢启动阈值的大小，然后进入拥塞避免阶段。这种方法避免了剧烈的速率波动，更快地恢复到合适的传输速率。

**Step2：在OMNet++中实现TCP-reno。**

请根据文件中的代码注释，补全关键部分的代码（TODO parts），实现OMNet++中实现TCP-reno。

1. 打开 $YOURWORKSHOP$/TcpMultipleSender/src/TcpCustomReno.cc文件。
2. 实现TcpCustomReno::recalculateSlowStartThreshold()：用于重新计算慢启动阈值，决定何时从慢启动阶段切换到拥塞避免阶段。

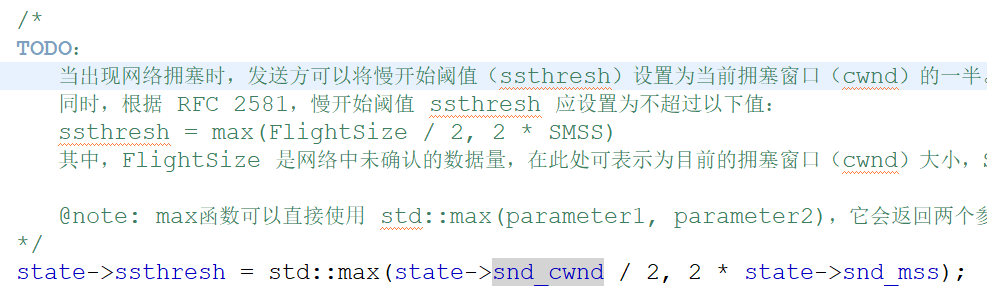


图12：补充代码

1. 实现TcpCustomReno::processRexmitTimer(TcpEventCode& event)：处理超时事件的重传定时器，TCP传输过程中发生数据包丢失会触发该函数，启动慢开始算法。

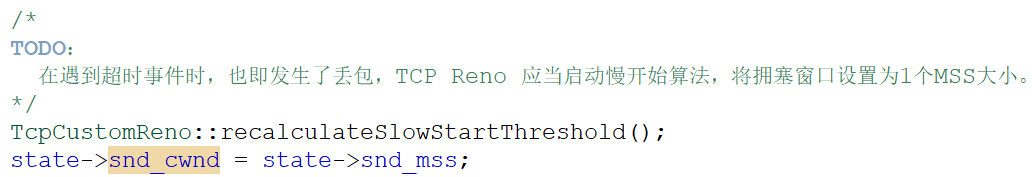


图13：补充代码

1. 实现TcpCustomReno::receivedDuplicateAck()：收到重复"重复ACK数量阈值"个ACK时，启动快重传算法和快恢复算法。

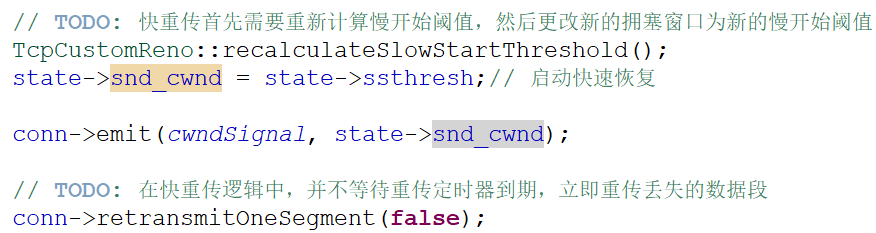


图14a：补充代码

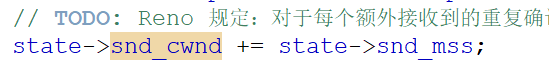


图14b：补充代码

1. 实现TcpCustomReno::receivedDataAck(uint32\_t firstSeqAcked)：在接收方收到位确认的数据包的ACK时会被调用。

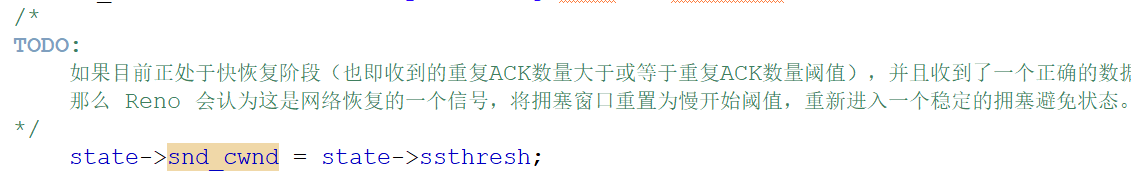


图15a：补充代码

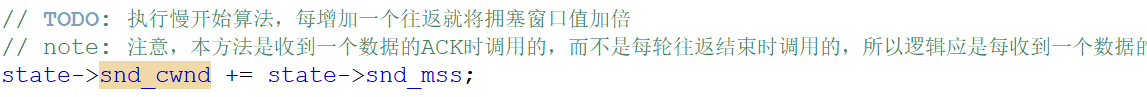


图15b：补充代码

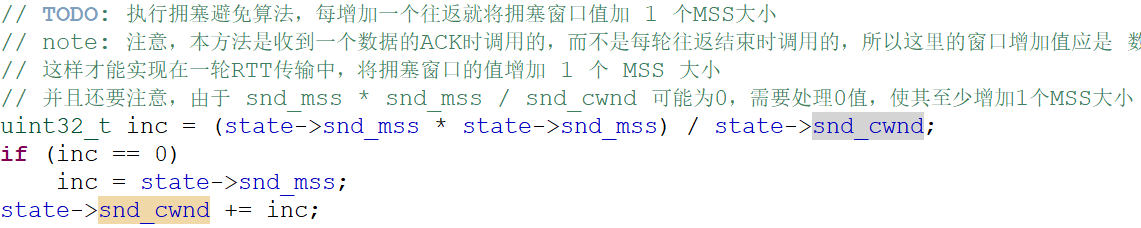


图15c：补充代码

**Step3：在不同网络环境中运行TCP-reno。**

* 使用任务1中默认的链路参数配置与传输参数配置，此时的Lossrate与Goodput分别是多少？
  1. 将链路参数配置与传输参数配置修改至与任务一默认参数一致。

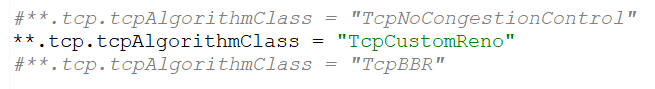
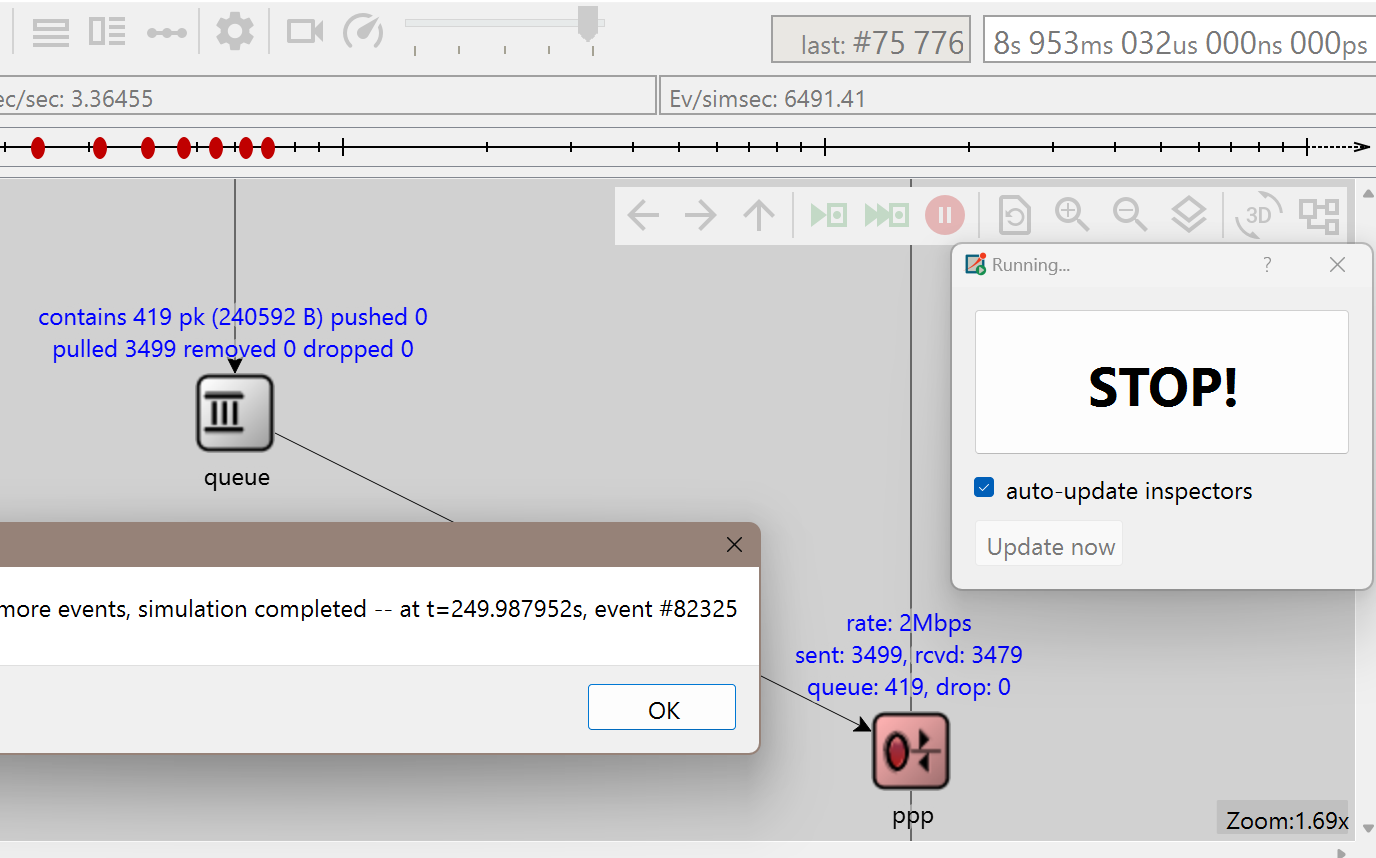




图16：实验参数

* 1. 运行模拟器。



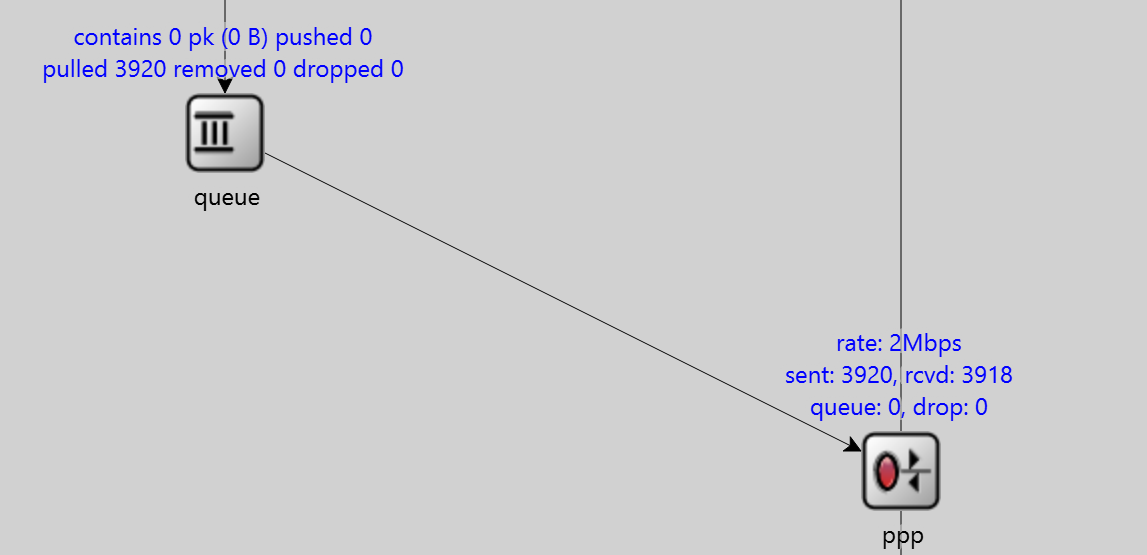


图17：模拟器运行结果

* 1. 计算得到结果。
* 其他配置不变，将路由器的队列长度依次更改为[1800，1600，1400，1200，1000]。记录Lossrate与Goodput的变化。
  1. 更改实验参数配置，将路由器的队列长度依次更改为[1800，1600，1400，1200，1000]。
  2. 运行模拟器。

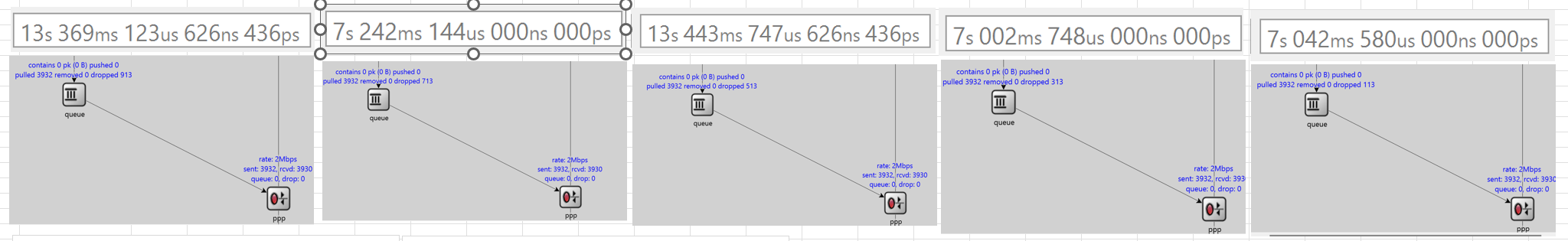


图18：运行结果

* 1. 运行结果列表。

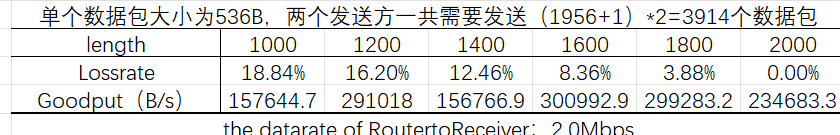


图19：结果列表

* 1. 运行结果可视化。

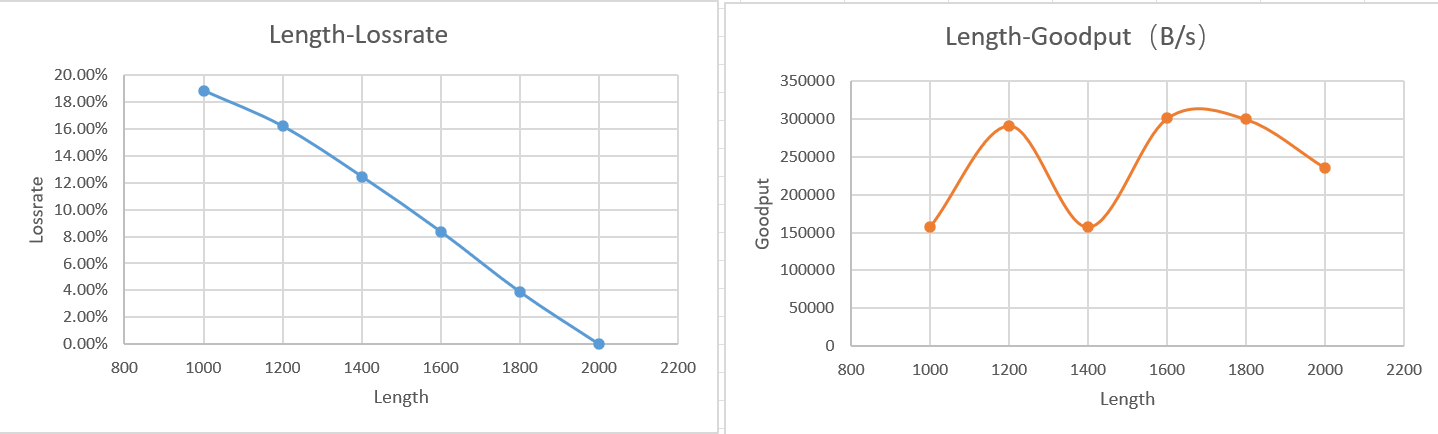


图20：可视化结果

* 将路由器的队列长度更改回默认值，并依次更改RoutertoReceiver（router与receiver之间的链路）的datarate为[1.8Mbps，1.6Mbps，1.4Mbps，1.2Mbps，1.0Mbps]。记录Lossrate与Goodput的变化。
  1. 更改实验参数配置，更改RoutertoReceiver（router与receiver之间的链路）的datarate为[1.8Mbps，1.6Mbps，1.4Mbps，1.2Mbps，1.0Mbps]。
  2. 运行模拟器。

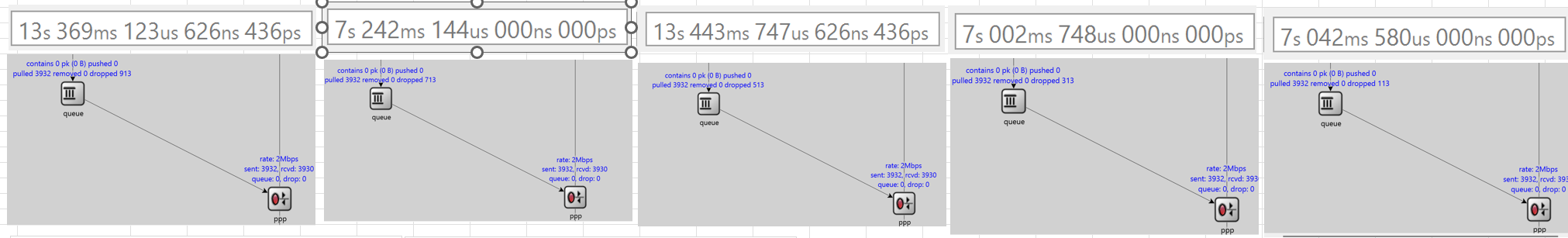


图20：运行结果

* 1. 运行结果列表。

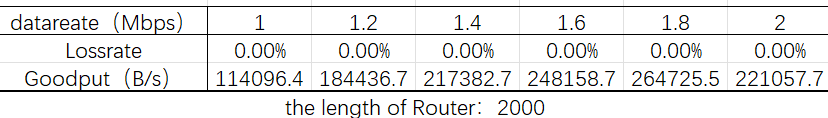


图21：结果列表

* 1. 运行结果可视化。

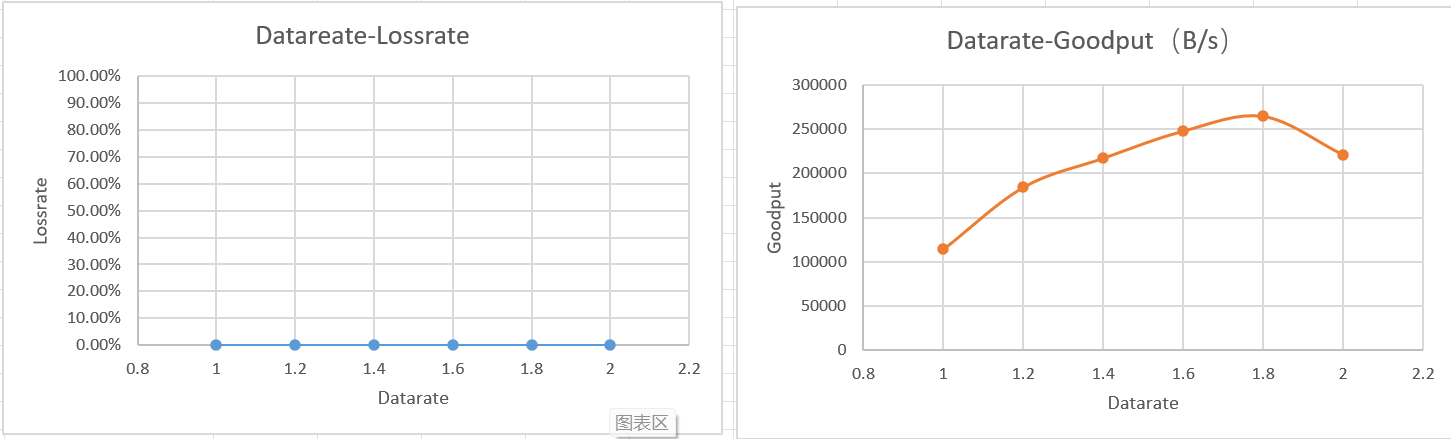


图22：可视化结果

* 将使用TCP-reno拥塞控制算法下的传输表现进行作图并对比。请解释在TCP-reno是如何控制数据包的发送以对进行拥塞控制的？

**解释在TCP-reno是如何控制数据包的发送以对进行拥塞控制的。**

TCP-reno主要通过慢启动、拥塞避免、快速重传和快速恢复这几个机制协同工作，实现拥塞控制。

* 1. 慢启动：初始时，拥塞窗口设为1个MSS。每次成功接收到一个ACK，cwnd增加1个MSS。这使得cwnd呈指数增长。慢启动一直持续到cwnd达到慢启动阈值，然后转入拥塞避免阶段。
  2. 拥塞避免：当cwnd达到ssthresh时，进入拥塞避免阶段。在拥塞避免阶段，每个RTT（Round-Trip Time）内cwnd增加1个MSS。这使得cwnd线性增长，而非指数增长。
  3. 快速重传：当发送方收到三个重复的ACK时，表明接收方下一个期望的数据包未收到，立即重传被认为丢失的数据包。
  4. 快速恢复：进入快速恢复阶段时，将ssthresh设为当前cwnd的一半。将cwnd设为ssthresh。在快速恢复期间，每收到一个重复的ACK，cwnd增加一个MSS，直到未确认的数据包被确认。一旦重传的数据包被确认，cwnd设为ssthresh，并进入拥塞避免阶段。

**工作步骤：1.连接开始：**使用慢启动快速增长cwnd，直到检测到网络的容量极限（ssthresh）。**2.达到ssthresh：**进入拥塞避免阶段，缓慢增加cwnd，控制传输速率。**3.丢包检测：**通过快速重传立即处理丢包，避免等待超时。**4.恢复阶段：**通过快速恢复机制，迅速恢复传输速度，减小因丢包引起的性能损失。

**将使用TCP-reno拥塞控制算法下的传输表现进行作图并对比。**

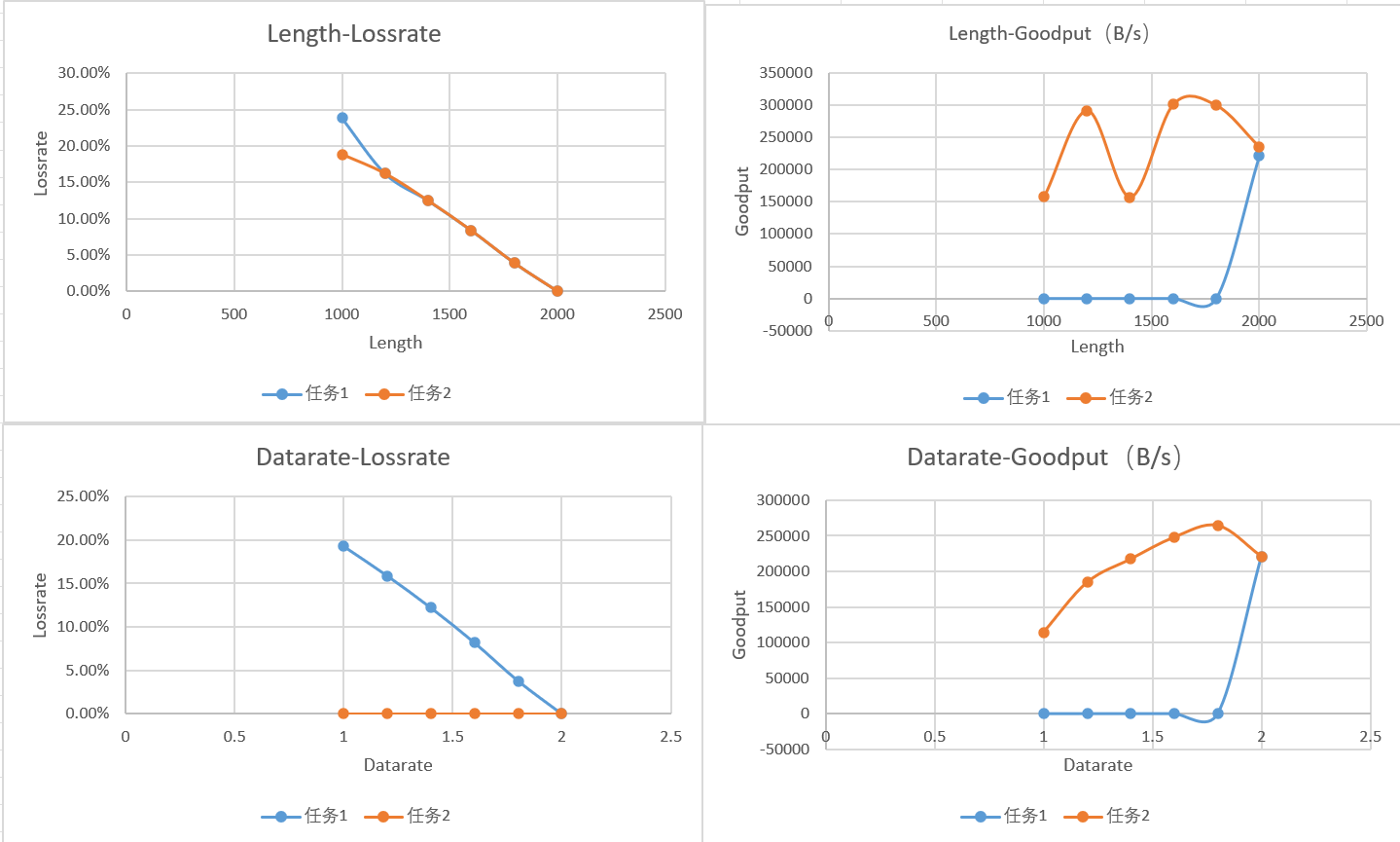


图23：TCP-reno和无拥塞控制对比

**总结**

1. **TCP-reno和无拥塞控制对于路由器队列长度与丢包率的关系影响相对较小，**只有在路由器长度为1000方有差距显现。TCP-reno控制下和无拥塞控制下路由器队列长度与丢包率的关系均为线性关系。
2. **TCP-reno和无拥塞控制对于路由器队列长度与有效吞吐量的关系影响较大。**由图23可以发现，TCP-reno控制的有效吞吐量在测量路由器队列长度范围内均比无拥塞控制的有效吞吐量高，证明TCP-reno的有效性。
3. **TCP-reno和无拥塞控制对于链路速度与丢包率的关系影响较大。**由图23可以发现，在TCP-reno控制下，链路丢包率为0。而无拥塞控制链路丢包率随链路速度降低而升高。再次证明了TCP-reno算法的有效性。
4. **TCP-reno和无拥塞控制对于链路速度与有效吞吐量的关系影响较大。**由图23可以发现，TCP-reno控制的有效吞吐量在测量路由器队列长度范围内均比无拥塞控制的有效吞吐量高，证明TCP-reno的有效性。

任务3：实现TCP-bbr拥塞控制算法

任务目标及要求

1. 实现TCP-bbr拥塞控制算法；
2. 利用OMNet++模拟器模拟数据传输，观察不同场景下传输数据时，TCP-bbr的性能表现，并作图进行分析；
3. 理解并实现TCP-bbr，解释bbr与reno这两个拥塞控制算法最核心的区别以及bbr的优势；
4. 在多种网络配置下进行对比（与任务1的结果、任务2的结果进行对比）实验，并将实验数据作图展示，进行分析；

任务步骤

**Step1：阅读并理解TCP-bbr的工作流程。**

请阅读TCP-bbr相关文章和文档：说明TCP-bbr与TCP-reno最本质的区别是什么？分析TCP-bbr相比于TCP-reno进行拥塞控制的优势是什么？

**最本质的区别：TCP-reno基于丢包和延迟（超时）进行网络拥塞控制；TCP-bbr基于带宽和RTT进行网络拥塞控制。**

优势：① TCP-reno在检测到丢包后会显著降低发送速率，导致带宽利用率降低。而TCP-bbr可以确保发送速率接近实际的瓶颈带宽，高效利用带宽。② TCP-reno在路由器队列满，而丢包的时候开始进行拥塞控制。一方面数据包排队导致高延迟，另一方面重发数据包导致网络拥塞进一步加剧。TCP-bbr通过控制发送速率避免队列积压，使数据包以较低延迟发送到接收方。

**Step2：在OMNet++中实现TCP-bbr。**

1. 请打开 $YOURWORKSHOP$/TcpMultipleSender/src/XXXXX.cc文件。
2. 基于给出的代码框架，补充代码内缺失内容并解释实现的内容（标记为TODO的部分，部分需要编程实现，部分只需要解释说明）。

**TODO之编程实现部分。**

* 计算RTT，实现app\_limited\_until的更新机制。

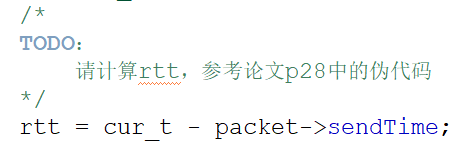


图24a：计算RTT

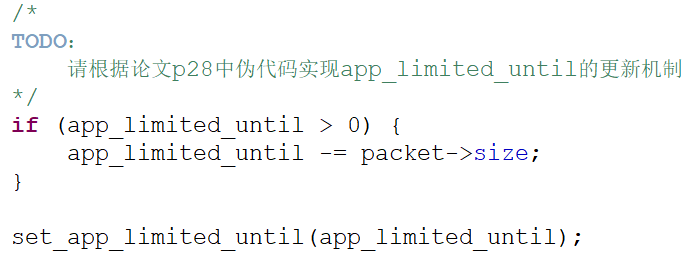


图24b：实现app\_limited\_until的更新机制

* 根据sendSize和spendT计算bw。

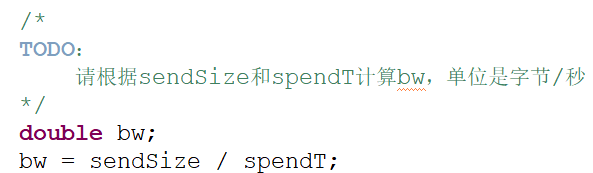


图25：计算bw

* BBR\_STARTUP状态下，对pacing\_gain和cwnd\_gain赋值；BBR\_DRAIN状态下，对pacing\_gain赋值。

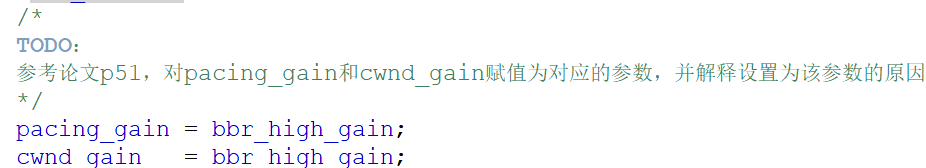


图26a：BBR\_STARTUP状态下赋值

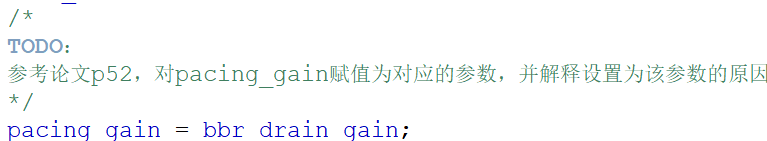


图26b：BBR\_DRAIN状态下赋值

* 实现BDP的赋值。

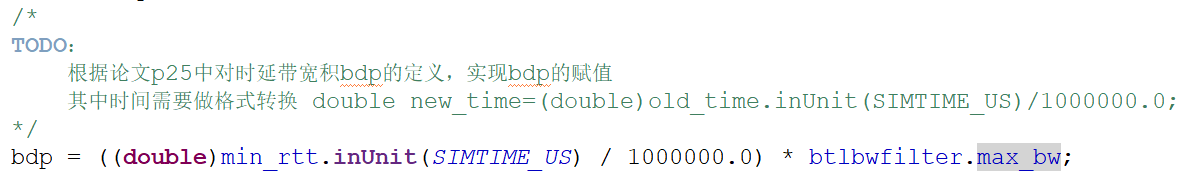


图27：BDP赋值

**TODO之解释说明部分。**

**请阅读bbr\_update\_model，理解bbr是如何更新rtt和bw估计以及状态机的。**

**更新bw：**TCP-bbr维护一个滑动窗口，记录一段时间内在链路中存在的最大数据量（正确发送ACK的包的序号到当前已发送但未确认的第一个包的序号），同时记录正确发送ACK的包的序号的时间到当前时间的时间间隔，最后将两者相除即可得到bw。

**更新RTT：**每当接收端ACK一个数据包时，发送端可以计算该数据包的RTT。TCP-bbr维护一个滑动的时间窗口（RTT值）。如果新的RTT值小于当前最小RTT值，则更新最小RTT值。如果超出一个时间窗口，则直接认为当前RTT值为最小RTT值。

**更新状态机：**①Startup：TCP-bbr在连接开始时进入Startup状态，通过指数增加发送速率，迅速探测并达到瓶颈带宽。当检测到带宽增长停止时，转换到Drain状态。② Drain：在Drain状态，BBR通过降低发送速率，排空网络中的队列，使得数据包的数量接近BDP（带宽时延积）。Drain状态结束后，进入ProbeBW状态。③ ProbeBW：在ProbeBW状态，BBR周期性地轻微调整发送速率，探测带宽的变化。大部分时间保持在稳定发送速率，以维持高带宽利用率和低延迟。④ ProbeRTT：在ProbeRTT状态，BBR通过短时间内显著降低发送速率，探测最小RTT。ProbeRTT状态周期性触发，以确保RTT估计的准确性。

**请问bbr算法是如何控制发送速率的？拥塞窗口的作用是什么？**

通过估算当前瓶颈带宽bw和最小RTT动态调整发送速率。TCP-bbr计算带宽时延积确定发送窗口大小，然后根据不同状态选择不同的增益系数。

**请问bbr算法是根据什么计算下一次发送时间间隔的？**

通过估算当前瓶颈带宽bw和最小RTT获得瓶颈带宽估计值，然后根据当前状态的增益系数计算发送速率，最后根据发送速率和数据包大小即可计算出发送间隔。

**请根据论文p26中的BtlBw公式解释btlbwfilter是如何工作的？rtt\_cnt的作用是什么？**

btlbwfilter是一个用于过滤和保持最近一段时间内最大带宽值的结构。该结构使用一个循环数组来存储在多个RTT期间测量的带宽样本，并在每次更新时计算出这些样本中的最大带宽值并存储在max\_bw中。

Rtt\_cnt是一个计数器，记录了经过的RTT（Round-Trip Time）次数。可以通过对rtt\_cnt的取模实现在循环数组中记录当前样本带宽并删除过时的样本带宽。

**请根据论文p51解释bbr\_check\_full\_bw\_reached()是如何判断是否达到最大带宽的？**

在StartUP状态下，TCP-bbr通过bw估计管道是否已满。如果它注意到有三轮尝试将交付率翻倍但是实际上几乎没有增加（增长幅度小于25%），那么它估计bw已经是最大带宽，然后退出 Startup 并进入 Drain。

**请从StartUP阶段的主要任务的角度解释btlbwfilter.max\_bw>=bw\_thresh时要将count\_full\_bw\_cnt的计数重置为0？**

如果当前测量的最大带宽btlbwfilter.max\_bw已经达到了阈值bw\_thresh，说明当前的发送速率已经足够高，进一步的增长可能并不会显著提高带宽。这时重置count\_full\_bw\_cnt的目的是：① 允许算法在后续的RTT中继续观察带宽变化，确保不会过早地认为已经达到了瓶颈带宽；② 通过重置计数器，可以避免因短期波动或误测导致的错误判断，从而确保只有在多次RTT测量中都确认带宽没有显著增长时，才认为达到了瓶颈带宽。

**参考论文p31和p49解释bbr算法中cycle gain的作用。**

在TCP-bbr中，cycle\_gain的作用是通过动态调整发送速率来优化带宽利用率和控制网络中的排队延迟。TCP-bbr的发送速率调控基于对链路带宽bw和最小RTT的估计，并且在其控制机制中引入了周期性增益变化，以不同的增益值（pacing gain）来调整发送速率。这种方式可以确保既能充分利用带宽，也能避免网络拥塞。

**针对bbr\_is\_next\_cycle\_phase()解释三个return的含义是什么？为什么要区分三种return？**

* 1. 当pacing\_gain为1时，TCP-bbr处于常规带宽探测阶段。此时，只需判断是否已经过了一个完整的RTT周期。如果是，则进入下一个周期阶段。
  2. 当 pacing\_gain大于1时，TCP-bbr处于增加发送速率以探测更高带宽的阶段。此时，除了需要经过一个完整的RTT周期外，还需要检查发送中的数据量是否达到了按当前增益计算的BDP。
  3. 当pacing\_gain小于1时，TCP-bbr处于减速阶段。此时，只需满足以下任一条件即可进入下一个周期阶段：经过了一个完整的RTT周期或发送中的数据量低于按增益为1计算的BDP。

**参考论文p51，对pacing\_gain和cwnd\_gain赋值为对应的参数，并解释设置为该参数的原因。**

StartUP状态下发送速率应呈指数级增长（每轮翻倍），为了以最平稳的方式实现翻倍，所以将pacing\_gain和cwnd\_gain赋值为2/ln2，这是允许每轮发送速率翻倍的最小值。

**参考论文p52，对pacing\_gain赋值为对应的参数，并解释设置为该参数的原因。**

Drain状态的目的是快速排空在StartUP状态下积压的队列，因此使用StartUP状态下使用的pacing\_gain的倒数来排空队列。综上，pacing\_gain赋值为ln2/2。

**Step3：在不同网络环境下运行TCP-bbr。**

将ini文件中的\*\*.tcp.tcpAlgorithmClass 赋值为 "TcpBBR"，使用TCP-bbr拥塞控制算法，观察不同网络状态下的数据传输表现。

* 使用任务1中默认的链路参数配置与传输参数配置，此时的Lossrate与Goodput分别是多少？
  1. 将实验参数设置为任务1中默认的链路参数配置与传输参数配置。
  2. 运行模拟器。



图28a：运行时间

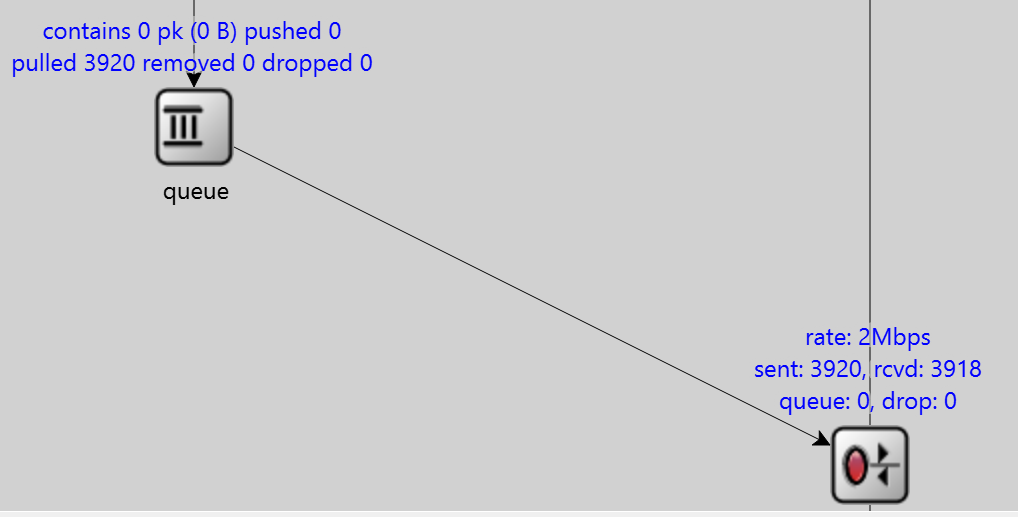


图28b：发送包数

* 1. 计算得到结果。
* 其他配置不变，将路由器的队列长度依次更改为[1800，1600，1400，1200，1000]。记录Lossrate与Goodput的变化。
  1. 更改实验参数，将路由器的队列长度依次更改为[18 00，1600，1400，1200，1000]。
  2. 运行模拟器。

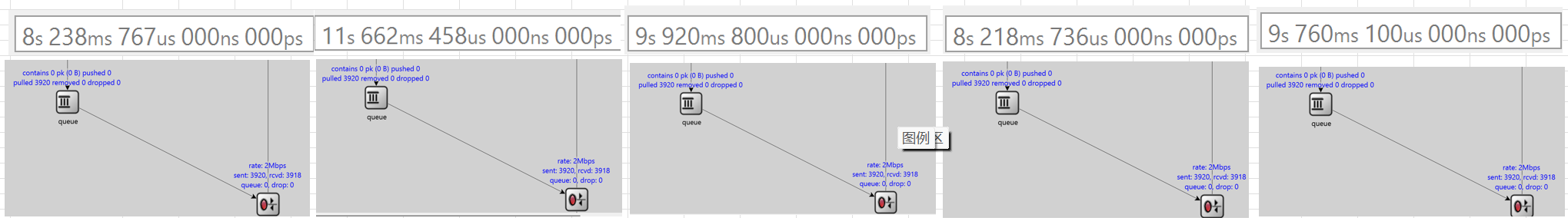


图29：运行结果

* 1. 实验结果列表。

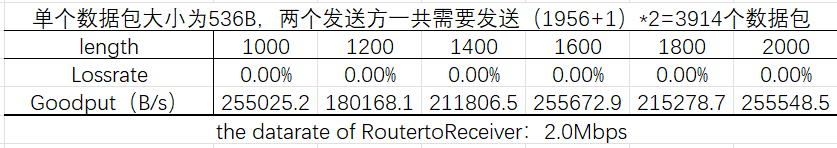


图30：结果列表

* 1. 实验结果可视化。

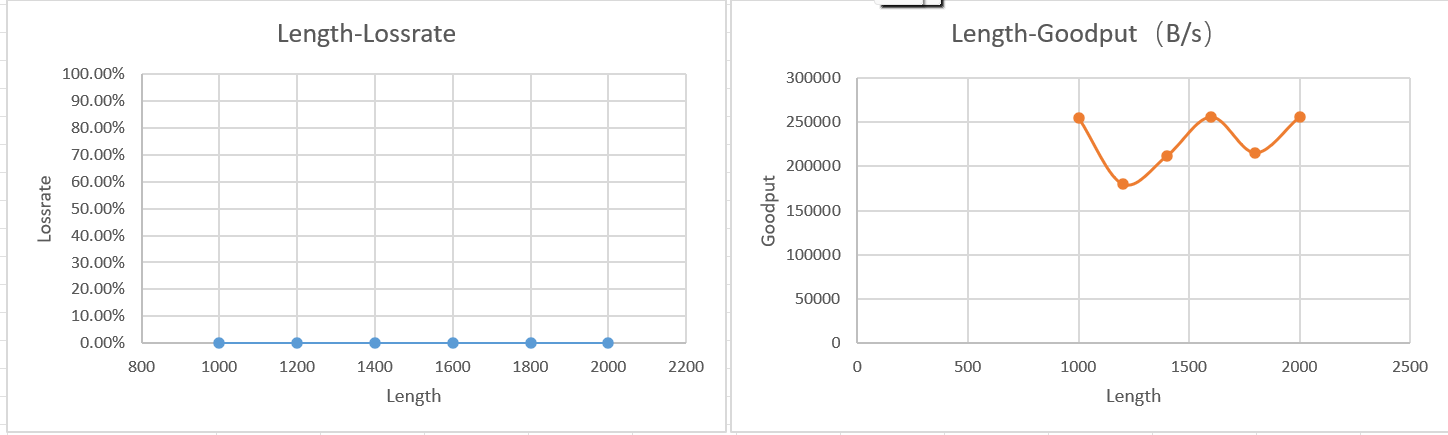


图31：可视化结果

* 将路由器的队列长度更改回默认值，并依次更改RoutertoReceiver（router与receiver之间的链路）的datarate为[1.8Mbps，1.6Mbps，1.4Mbps，1.2Mbps，1.0Mbps]。记录Lossrate与Goodput的变化。
  1. 更改实验参数，依次更改RoutertoReceiver（router与receiver之间的链路）的datarate为[1.8Mbps，1.6Mbps，1.4Mbps，1.2Mbps，1.0Mbps]。
  2. 运行模拟器。

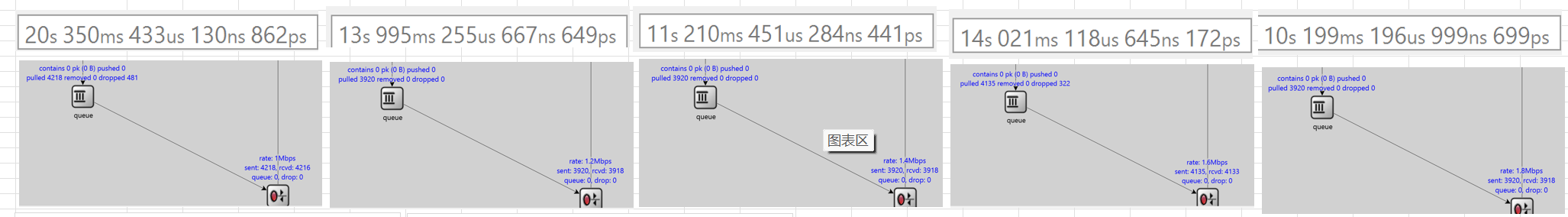


图32：运行结果

* 1. 实验结果列表。

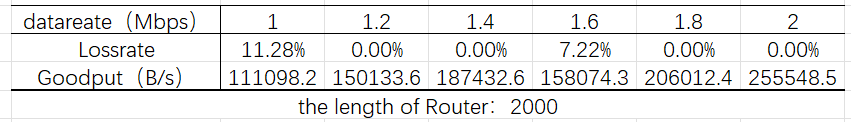


图33：结果列表

* 1. 实验结果可视化。

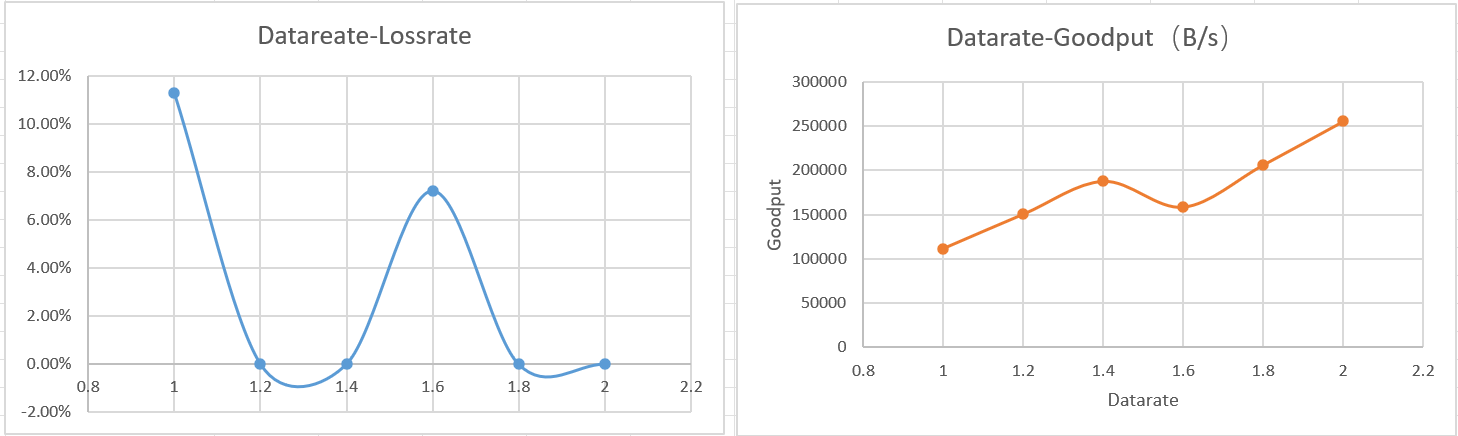


图34：可视化结果

将使用TCP-bbr拥塞控制算法下的传输表现进行作图并对比。请解释在TCP-bbr是如何控制数据包的发送以对进行拥塞控制的？

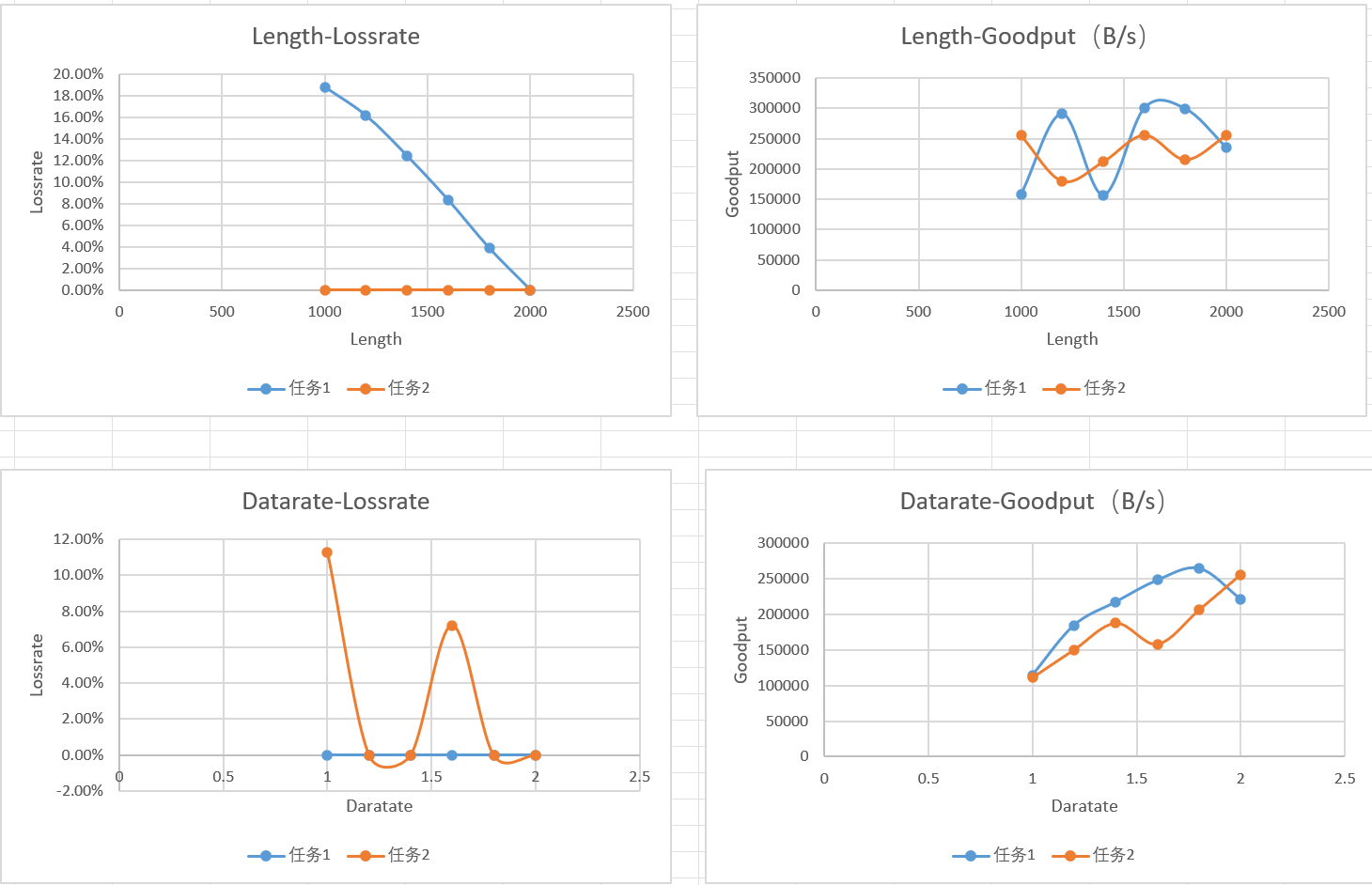


图35：TCP-bbr和TCP-reno对比

**解释在TCP-bbr是如何控制数据包的发送以对进行拥塞控制。**

①Startup：TCP-bbr在连接开始时进入Startup状态，通过指数增加发送速率，迅速探测并达到瓶颈带宽。当检测到带宽增长停止时，转换到Drain状态。② Drain：在Drain状态，BBR通过降低发送速率，排空网络中的队列，使得数据包的数量接近BDP（带宽时延积）。Drain状态结束后，进入ProbeBW状态。③ ProbeBW：在ProbeBW状态，BBR周期性地轻微调整发送速率，探测带宽的变化。大部分时间保持在稳定发送速率，以维持高带宽利用率和低延迟。④ ProbeRTT：在ProbeRTT状态，BBR通过短时间内显著降低发送速率，探测最小RTT。ProbeRTT状态周期性触发，以确保RTT估计的准确性。

# 实验结果

任务1：拥塞现象观察

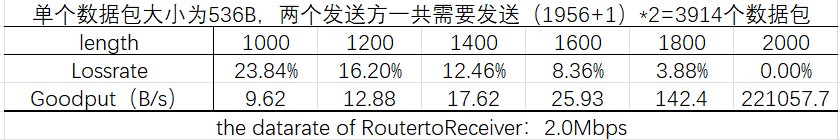


图36：结果列表

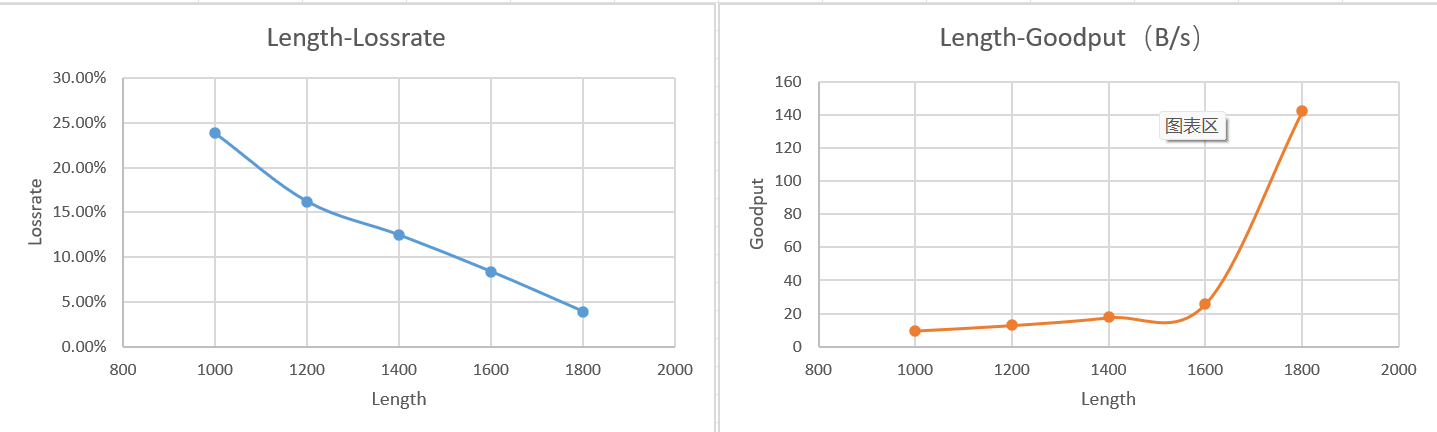


图37：可视化结果

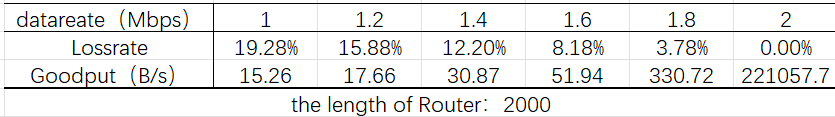


图38：结果列表

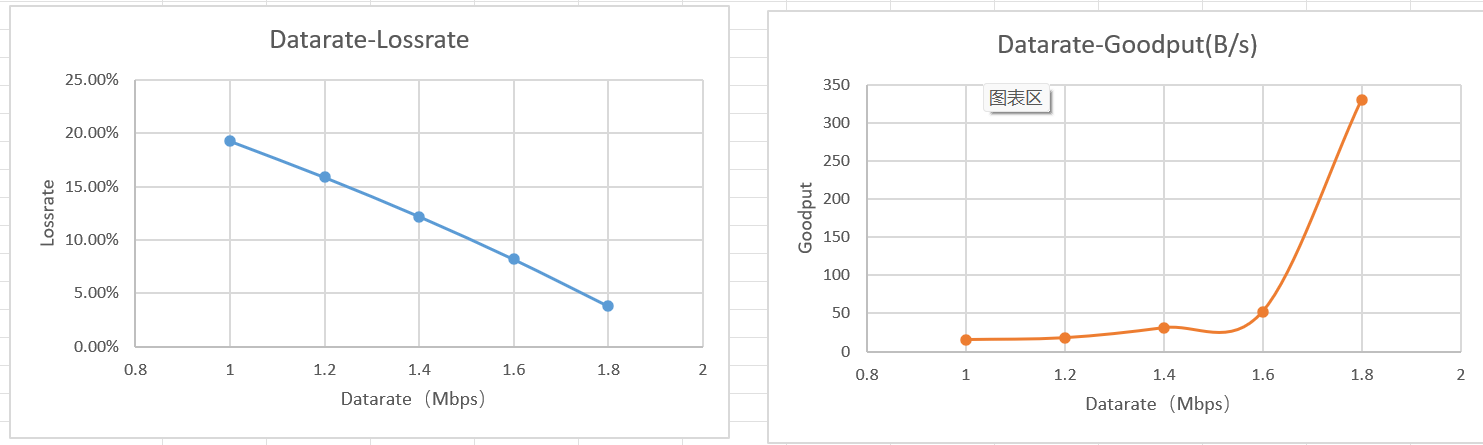


图39：可视化结果

任务2：实现TCP-reno拥塞控制算法

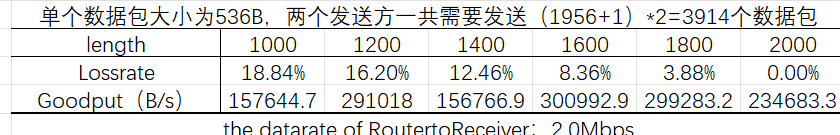


图40：结果列表

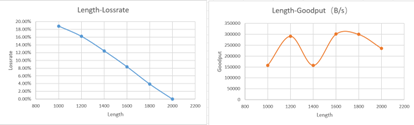


图41：可视化结果



图42：结果列表

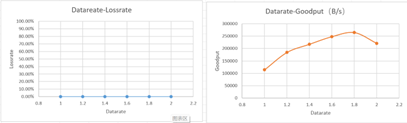


图43：可视化结果

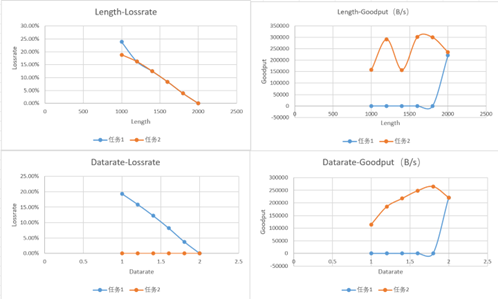


图44：TCP-reno和无拥塞控制对比

任务3：实现TCP-bbr拥塞控制算法

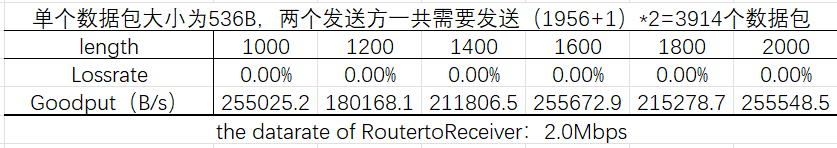


图45：结果列表

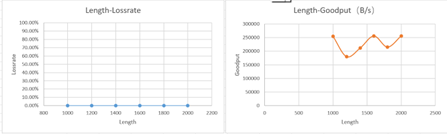


图46：可视化结果



图47：结果列表

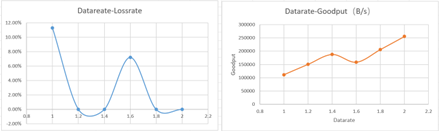


图48：可视化结果

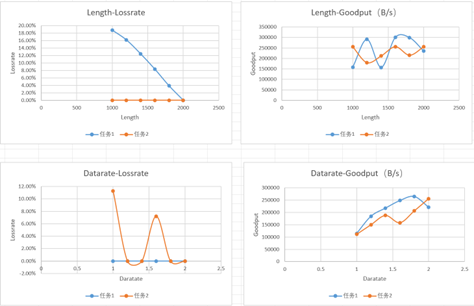


图49：TCP-reno和TCP-bbr对比

# 实验分析

任务1

**Lossrate随着datarate的下降而上升，呈线性上升趋势；Goodput随着datarate的下降而下降，呈指数下降趋势。**可能的原因：**对于Lossrate，**链路速度决定了数据包传输的速率。较低的链路速度在单位时间内传输的数据包更少，增加了队列中的数据包的积压，提高了丢包的可能性。当链路速度下降时，数据包传输变慢，队列中的数据包等待时间变长，容易导致队列溢出（队列满）。这种情况下，新的数据包将会被丢弃。由于每单位时间内传输的数据包数量减少，导致丢包率增加。因为链路速度每减少一定比例，会相应地增加丢包的概率，所以这种趋势通常是线性的。**对于Goodput，**Goodput是成功传输的有用数据的速率，不包括重传的数据包。它反映了网络的实际有效传输性能。当链路速度下降时，传输速率降低，Goodput也会相应下降。链路速度下降导致丢包率上升，网络需要更多的重传来补偿丢失的数据包。重传占用了额外的带宽，进一步降低有效数据传输速率。TCP协议的拥塞控制机制在丢包率增加时，会显著降低发送速率，以适应链路的传输能力。由于TCP的拥塞控制调整速率通常是指数级的，这导致Goodput下降呈现指数趋势。

**产生拥塞的原因：① 链路速度不足。**链路速度决定了数据传输的最大速率。当流量数据超过链路的处理能力时，数据包会在路由器的队列中等待，造成拥塞。如果链路速度太低而数据太多会导致队列溢出，数据包被会丢弃，网络中重传请求增加，占用带宽，进一步加剧拥塞。**② 高流量负载。**如果网络中数据量突然增加或者数据量持续高于链路传输能力，容易导致路由器队列迅速积满，导致拥塞。**③ 不合理的队列长度。**如果队列设置过短，容易导致队列溢出，增加丢包率，进而引起重传，加剧网络拥塞。如果队列设置过长，可能会增加延迟。

任务2

1. **TCP-reno和无拥塞控制对于路由器队列长度与丢包率的关系影响相对较小，**只有在路由器长度为1000方有差距显现。TCP-reno控制下和无拥塞控制下路由器队列长度与丢包率的关系均为线性关系。
2. **TCP-reno和无拥塞控制对于路由器队列长度与有效吞吐量的关系影响较大。**由图23可以发现，TCP-reno控制的有效吞吐量在测量路由器队列长度范围内均比无拥塞控制的有效吞吐量高，证明TCP-reno的有效性。
3. **TCP-reno和无拥塞控制对于链路速度与丢包率的关系影响较大。**由图23可以发现，在TCP-reno控制下，链路丢包率为0。而无拥塞控制链路丢包率随链路速度降低而升高。再次证明了TCP-reno算法的有效性。
4. **TCP-reno和无拥塞控制对于链路速度与有效吞吐量的关系影响较大。**由图23可以发现，TCP-reno控制的有效吞吐量在测量路由器队列长度范围内均比无拥塞控制的有效吞吐量高，证明TCP-reno的有效性。

任务3

1. TCP-bbr在丢包率方面的表现明显优于TCP-reno，原因在于它们**最本质的区别：TCP-reno基于丢包和延迟（超时）进行网络拥塞控制；TCP-bbr基于带宽和RTT进行网络拥塞控制。**TCP-bbr较于TCP-reno有如下优势：① TCP-reno在检测到丢包后会显著降低发送速率，导致带宽利用率降低。而TCP-bbr可以确保发送速率接近实际的瓶颈带宽，高效利用带宽。② TCP-reno在路由器队列满，而丢包的时候开始进行拥塞控制。一方面数据包排队导致高延迟，另一方面重发数据包导致网络拥塞进一步加剧。TCP-bbr通过控制发送速率避免队列积压，使数据包以较低延迟发送到接收方。
2. 在有效吞吐量方面，两种拥塞控制算法的表现相差无几。原因在于两种算法都能比较好地以接近链路瓶颈的发送速率发送数据包。

# 实验总结

* + 1. 通过本次实验，我了解了数据传输过程中出现拥塞的原因。
    2. 通过本次实验，我了解了拥塞现象的在网络中的表现。
    3. 通过本次实验，我了解了TCP-reno和TCP-bbr两种拥塞控制算法各自的工作原理。
    4. 通过本次实验，我了解了TCP-reno和TCP-bbr两种拥塞控制算法的区别以及它们各自的优势。

# 思考题

在什么情况下TCP-Reno比TCP-BBR表现更好，反之亦然？请设计实验场景来验证你的假设，并解释其原因。要求：设计两种网络拓扑结构，分别运行TCP-Reno和TCP-BBR，并记录丢包率（LossRate）和有效吞吐量（GoodPut）。分析并解释实验结果，讨论两种算法在不同网络条件下的优势和劣势。

TCP-reno表现更好的情况：**① 低延迟、高带宽的网络。**在这种环境下，TCP-Reno能够快速恢复从丢包中，并且其拥塞窗口的增长方式可以迅速利用高带宽。由于网络延迟低，TCP-Reno的基于丢包的拥塞控制机制不会带来明显的传输性能问题。**② 稳定的网络环境。**在稳定的网络环境下，TCP-Reno的丢包检测和窗口调整机制能够有效维持高吞吐量，并且由于网络的稳定性，TCP-Reno的拥塞控制算法能保持较好的性能。

TCP-bbr表现更好的情况：**① 高延迟、低带宽的网络。**TCP-bbr通过主动测量带宽和RTT来调节发送速率，不依赖丢包作为拥塞信号，能够更好地适应高延迟和低带宽环境。TCP-bbr可以避免由于高RTT导致的慢启动和拥塞窗口收缩。**② 动态变化的网络环境。**TCP-bbr在动态网络环境下能够通过实时测量带宽和RTT来调整发送速率，提供稳定的传输性能。它的带宽探测机制使其在网络条件变化时依然能维持较高的吞吐量。

为了验证假设，我们设计两种网络拓补结构，两种均由两个个客户端、一个路由器（队列长度为1800）以及一个服务端构成。这两种网络分别为低延迟、高带宽网络（20ms、1.6Mbps）以及高延迟、低带宽网络（200ms、1.0Mbps）。

1. 运行模拟器，实验结果见下图。

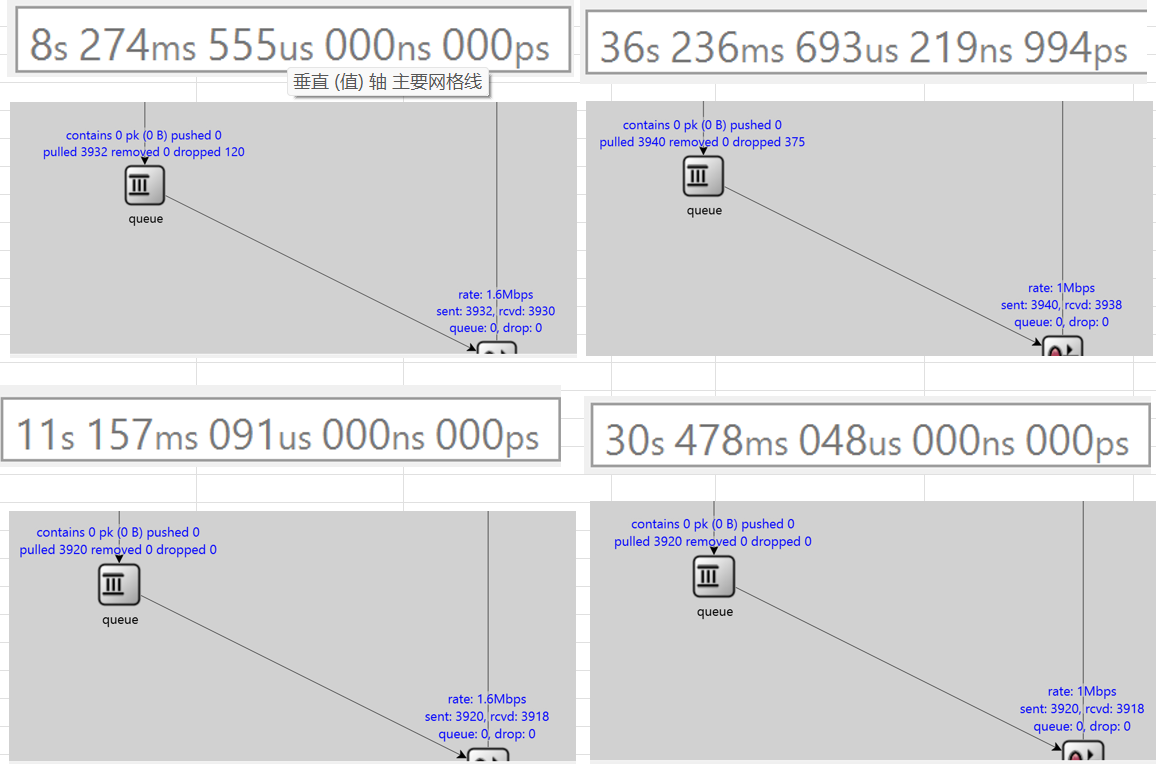


图50：实验结果（左低延迟、高带宽网络，右高延迟、低带宽）

1. 实验结果列表。

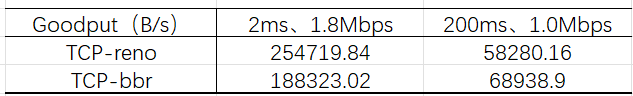


图51a：Gooput

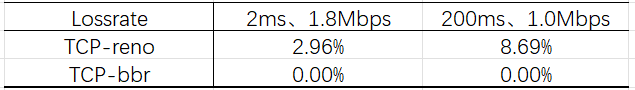


图52b：Lossrate

1. 实验结果可视化。

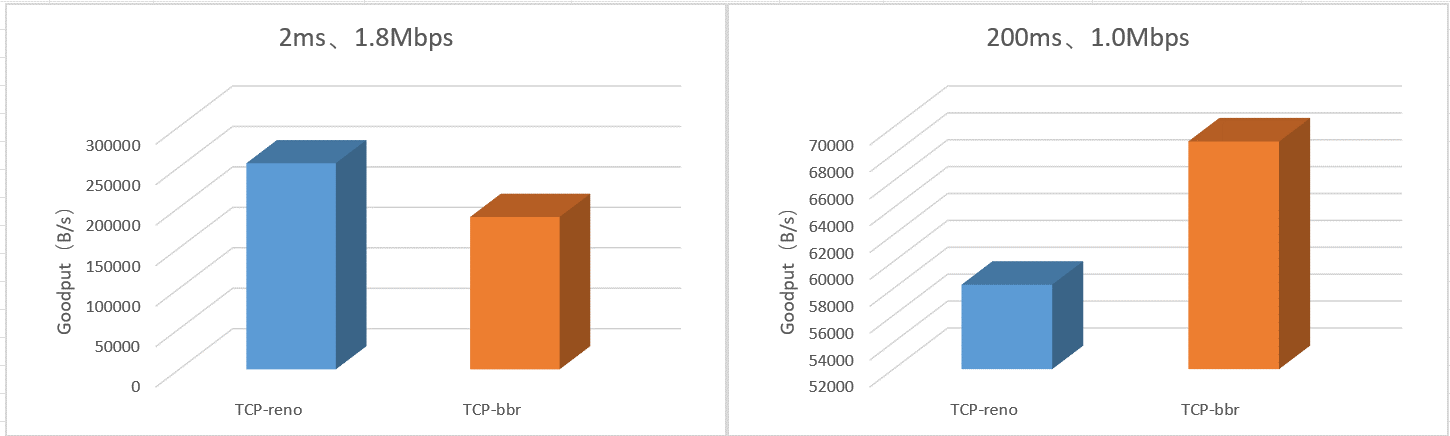


图51a：Gooput

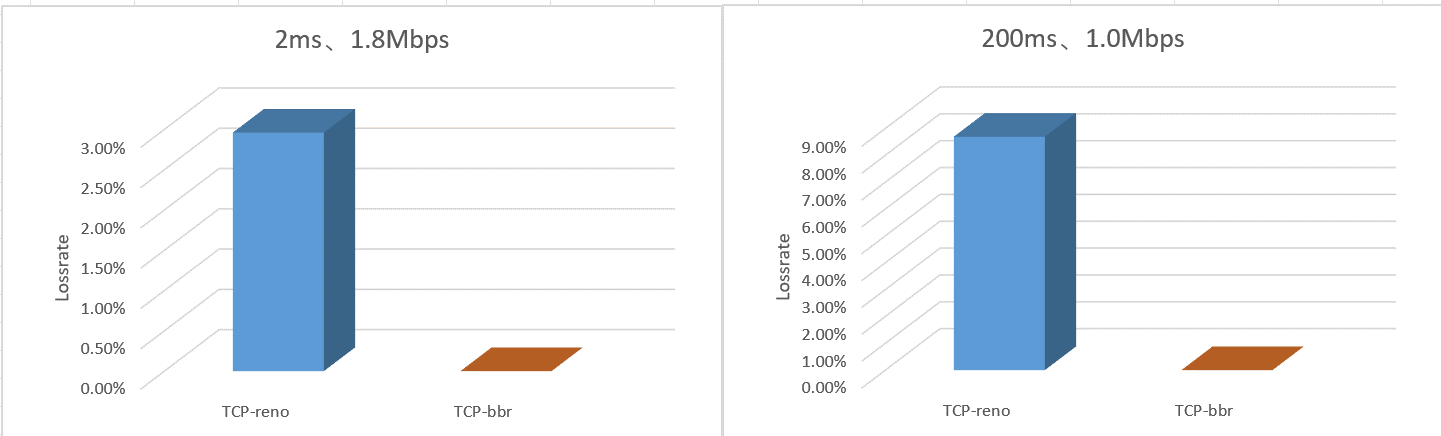


图52b：Lossrate

1. 实验结果分析。

由上展示的实验结果可知，实验假设正确。TCP-reno在低延迟、高带宽的网络下表现更好，TCP-bbr在高延迟、低带宽的网络下表现更好。

TCP-reno是基于丢包和时延的算法，在低延迟、高带宽的网络中始终能以接近带宽的发送速率进行发送活动；TCP-bbr在低延迟、高带宽的网络中，StartUP和Drain两种状态可能需要消耗较多的时间，在这一段时间里无法有效利用带宽。

TCP-reno在高延迟、低带宽的网络中会发生多次重传现象，导致网络拥塞进一步加重，丢包数量增加；TCP-bbr在高延迟、低带宽的网络中则能比较好地以网络最大带宽的发送速率发送数据包。

指导教师批阅意见

成绩评定

指导教师签字：

年 月 日

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。