**华中科技大学计算机学院**

**《计算机通信与网络》实验报告**

姓名 班级 学号

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目 | Socket编程  （40%） | NS2实验  （20%） | CPT实验  （20%） | 平时成绩(20%) | 总分 |
| 得分 |  |  |  |  |  |

教师评语：

教师签名：

给分日期：

**目 录**

[实验一 Socket编程实验 1](#_Toc497467181)

[**1.1 环境** 1](#_Toc497467182)

[**1.2 系统功能需求** 1](#_Toc497467183)

[**1.3 系统设计** 1](#_Toc497467184)

[**1.4 系统实现** 1](#_Toc497467185)

[**1.5 系统测试及结果说明** 1](#_Toc497467186)

[**1.6 其它需要说明的问题** 1](#_Toc497467187)

[实验二 基于NS2的协议分析实验 2](#_Toc497467188)

[**2.1 环境** 2](#_Toc497467189)

[**2.2 实验要求** 2](#_Toc497467190)

[**2.3 实验步骤说明及结果分析** 2](#_Toc497467191)

[**2.4 其它需要说明的问题** 2](#_Toc497467192)

[实验三 基于CPT的组网实验 3](#_Toc497467193)

[**3.1 环境** 3](#_Toc497467194)

[**3.2 实验要求** 3](#_Toc497467195)

[**3.3 基本部分实验步骤说明及结果分析** 3](#_Toc497467196)

[**3.4 综合部分实验设计、实验步骤及结果分析** 3](#_Toc497467197)

[**3.5 其它需要说明的问题** 3](#_Toc497467198)

[心得体会与建议 4](#_Toc497467199)

[**4.1 心得体会** 4](#_Toc497467200)

[**4.2 建议** 4](#_Toc497467201)

# 实验一 Socket编程实验

## **1.1 环境**

### **1.1.1 开发平台**

（开发机器的硬件配置、系统软件组件、开发平台、使用的第三方组件）

### **1.1.2 运行平台**

（软件运行机器的硬件配置、系统软件组件、第三方组件）

## **1.2 系统功能需求**

（说明系统需要实现的功能，主要指实验指导手册上的要求）

## **1.3 系统设计**

（说明你是如何设计系统的，包括但不限于系统架构、功能模块划分、应用层协议设计，可以超过系统功能需求界定的功能范围）

## **1.4 系统实现**

（说明你是如何实现系统的，包括但不限于技术关键及解决方案、关键功能的流程图）

## **1.5 系统测试及结果说明**

（说明系统测试的硬件环境，包括机器配置、网络配置等，说明系统测试的结果，对其进行分析，是否符合功能需求及你的设计要求，关于性能部分，必须给出速度曲线图，并结合协议设计进行分析）

## **1.6 其它需要说明的问题**

（除了上述问题外，你觉得还需要说明的内容可在本处撰写）

# 实验二 基于NS2的协议分析实验

## **2.1 环境**

（实验机器的硬件配置、系统软件组件、第三方软件）

## **2.2 实验要求**

（说明实验需要完成的工作，主要指实验指导手册上的要求）

## **2.3 实验步骤说明及结果分析**

### **2.3.1 第一项实验的步骤及结果分析**

实验步骤：

1. 新建6个节点用于tcp传入，5个节点用于udp传入。共计28个节点
2. 把各个节点连接起来。
3. 设置6个节点的传输层代理为tcp，对应6个tcpsink接收节点。设置5个节点的传输层代理为udp，对应5个null接收节点。
4. 生成脚本并执行，观察nam演示现象。
5. 分析实验结果。
6. 包重传率

TCP：6个节点中最小包重传率如图2.1，最大包重传率如图2.2

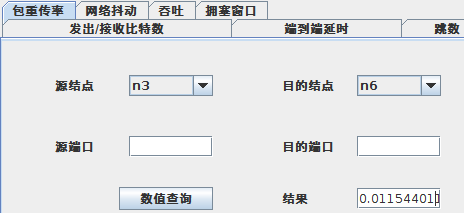
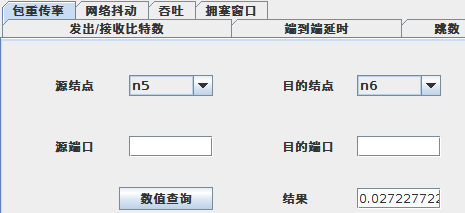
 

图2.1 TCP最小包重传率 图2.2 TCP最大包重传率

UDP:由于udp为不可靠传输，在丢包以后不会进行重传，因此所有udp节点的包重传率为0，如图2.3所示

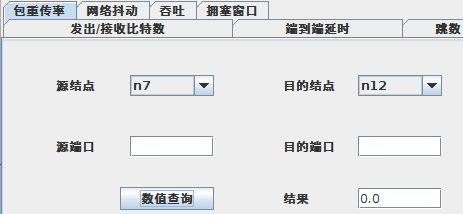


图2.3 UDP包重传率

1. 网络抖动

TCP网络抖动分析如图2.4所示

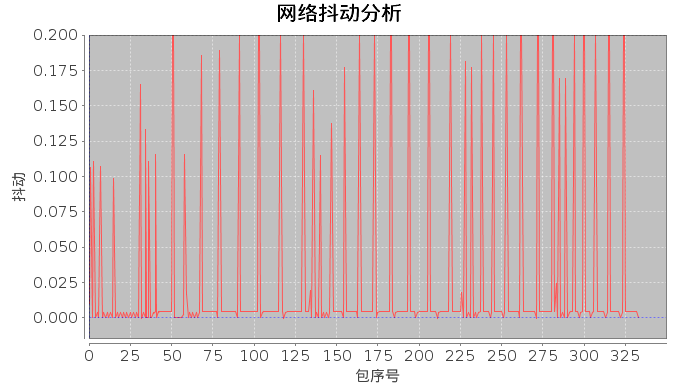


图2.4 TCP网络抖动分析

UDP网络抖动分析如图2.5所示

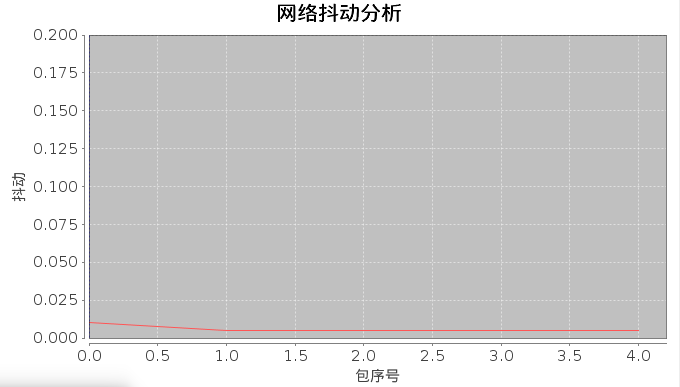


图2.5 UDP网络抖动分析

可以看出tcp的抖动很明显，而udp几乎没有网络抖动

1. 网络吞吐量

TCP网络吞吐量如图2.6所示

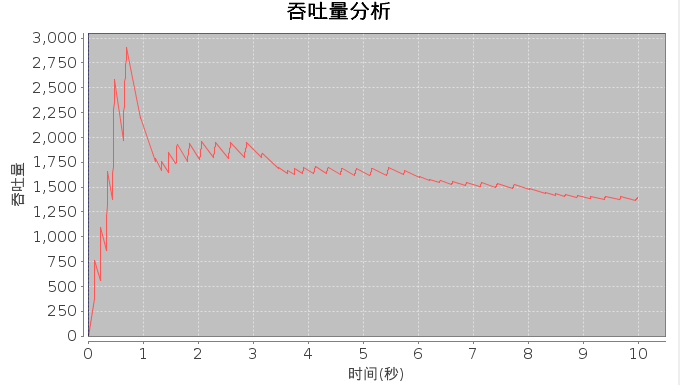


图2.6 TCP网络吞吐量

UDP网络吞吐量如图2.7所示

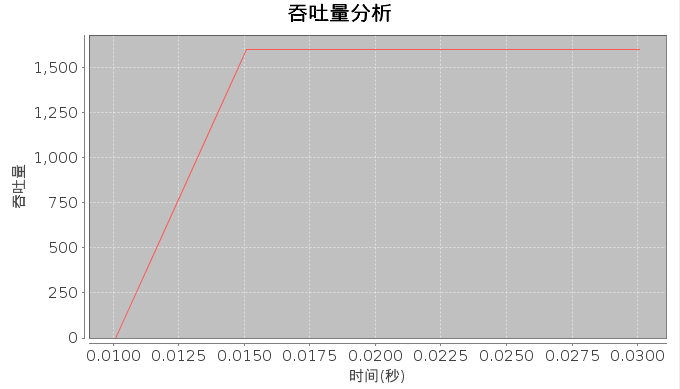


图2.7 UDP网络吞吐量

这里可以看到，udp仅用0.015秒就完成了数据的发送，传输速率远远高于tcp协议。

公平性是在发生拥塞时各源端(或同一源端建立的不同TCP连接或UDP数据报)能公平地共享同一网络资源(如带宽、缓存等)

Udp恒定速率发送数据，因此他是不公平的，在网络拥塞时他的发送速率也不会降低到“公平”级别。Tcp通过拥塞窗口的控制实现网络公平性。

面向连接的TCP和无连接的UDP在拥塞发生时对拥塞指示的不同反应和处理，导致对网络资源的不公平使用问题。在拥塞发生时，有拥塞控制反应机制的TCP数据流会按拥塞控制步骤进入拥塞避免阶段，从而主动减小发送入网络的数据量。但对无连接的数据报UDP，由于没有端到端的拥塞控制机制，即使网络发出了拥塞指示（如数据包丢失、收到重复ACK等），UDP也不会像TCP那样减少向网络发送的数据量。结果遵守拥塞控制的TCP数据流得到的网络资源越来越少，没有拥塞控制的UDP则会得到越来越多的网络资源，这就导致了网络资源在各源端分配的严重不公。

1. 端到端延时

TCP端到端最大如图2.8所示，最小如图2.9所示

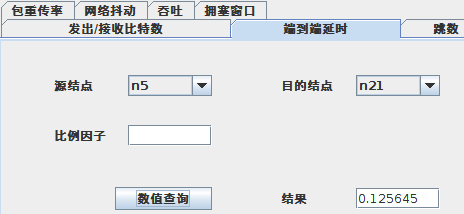
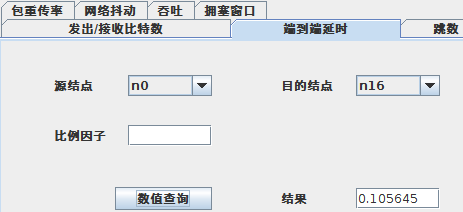
 

图2.8 TCP端到端最大延时 图2.9 TCP端到端最小延时

UDP由于没有回包，无法计算端到端延时，如图2.10所示

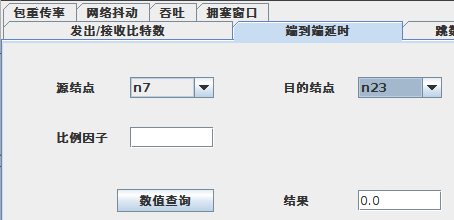


图2.10 UDP端到端延时

### **2.3.2 第二项实验的步骤及结果分析**

（对第二项实验的实验过程进行描述，并对结果结合理论课只是进行分析）

1. 拥塞窗口

TCP/Tahoe拥塞窗口分析如图2.11所示

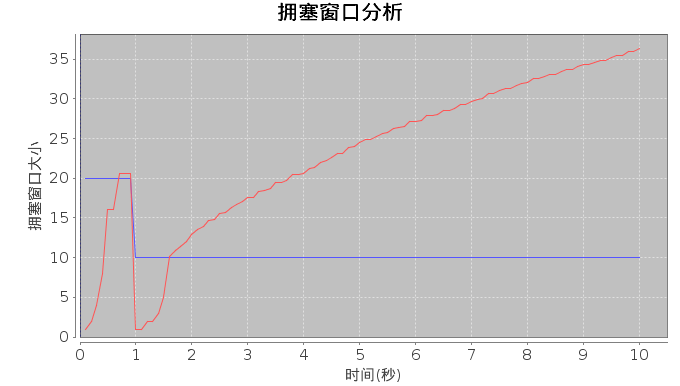


图2.11 TCP/Tahoe拥塞窗口分析

TCP/Reno拥塞窗口分析如图2.12所示

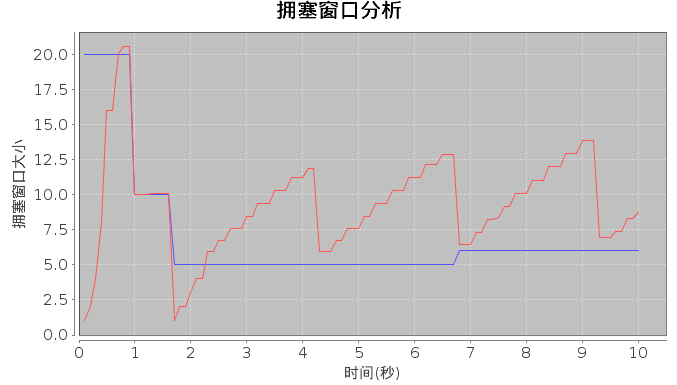


图2.12 TCP/Reno拥塞窗口分析

TCP/NewReno拥塞窗口分析如图2.13所示

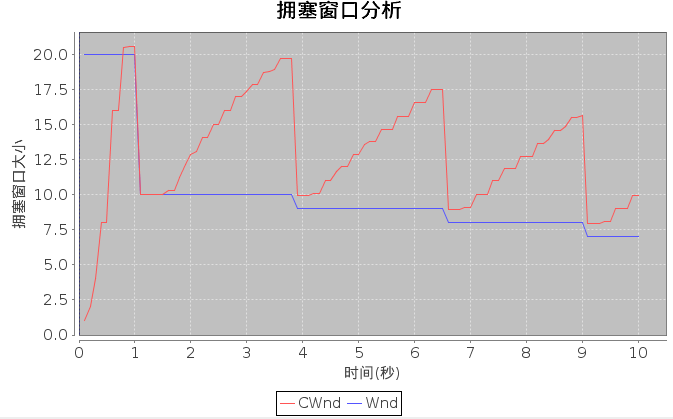


图2.13 TCP/NewReno拥塞窗口分析

TCP/SACK拥塞窗口分析如图2.14所示

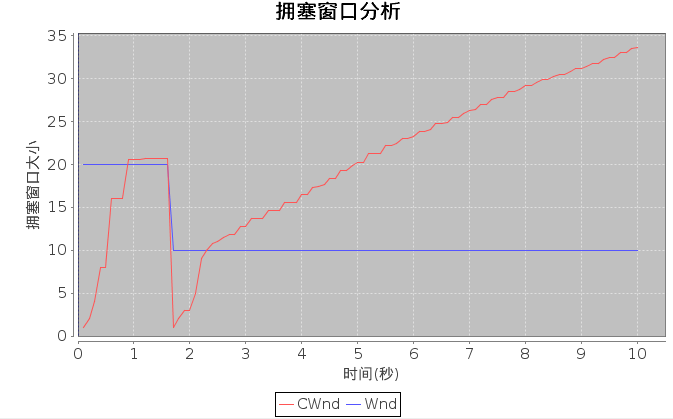


图2.14 TCP/SACK拥塞窗口分析

TCP/FACK拥塞窗口分析如图2.15所示

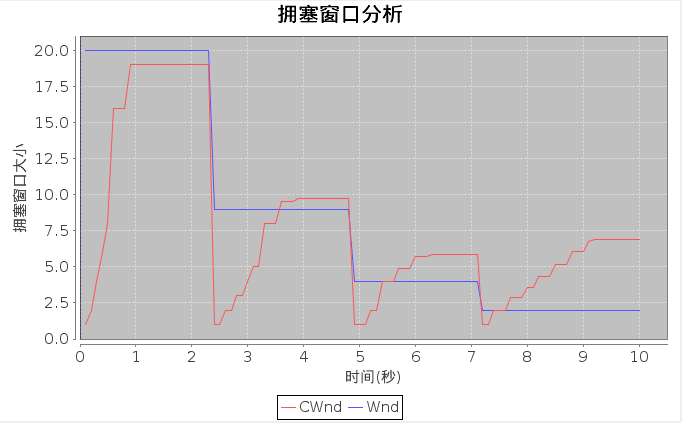


图2.15 TCP/FACK拥塞窗口分析

TCP/Vegas拥塞窗口分析如图2.16所示

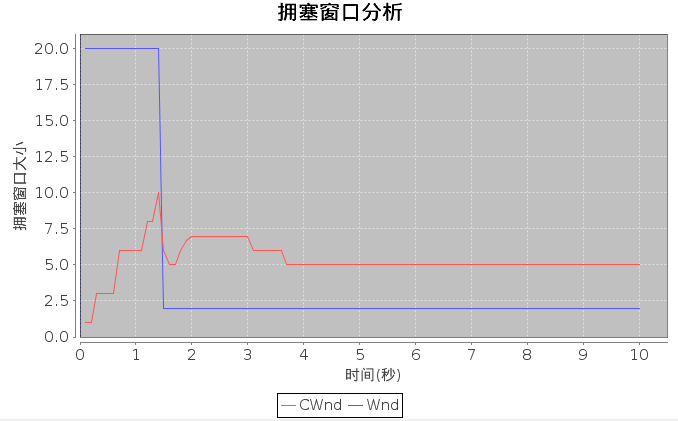


图2.16 TCP/Vegas拥塞窗口分析

1. 阈值变化

TCP/Tahoe每次丢包后拥塞窗口减半

1. 吞吐量

TCP/Tahoe吞吐量分析如图2.17所示

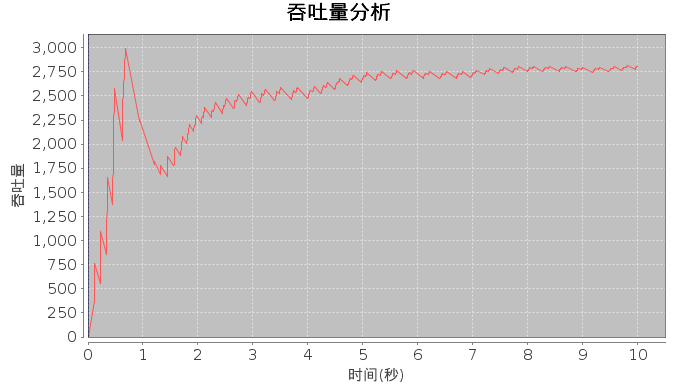


图2.17 TCP/Tahoe吞吐量分析

TCP/Reno吞吐量分析如图2.18所示

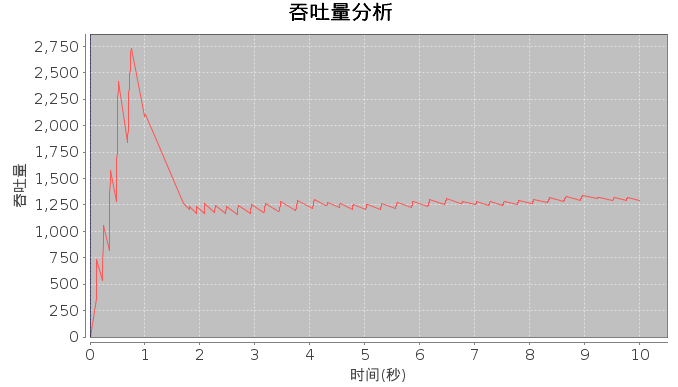


图2.18 TCP/Reno吞吐量分析

TCP/NewReno吞吐量分析如图2.19所示

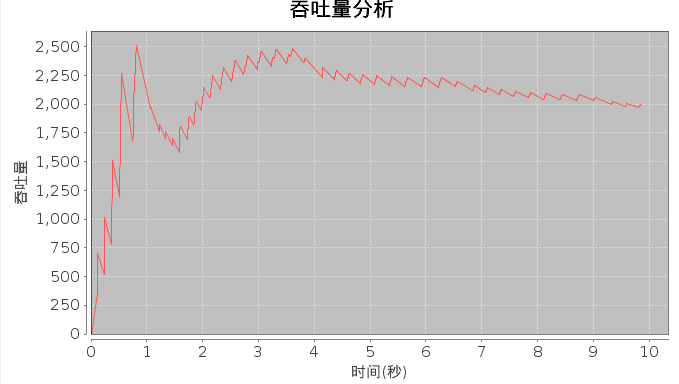


图2.19 TCP/NewReno吞吐量分析

TCP/SACK吞吐量分析如图2.20所示

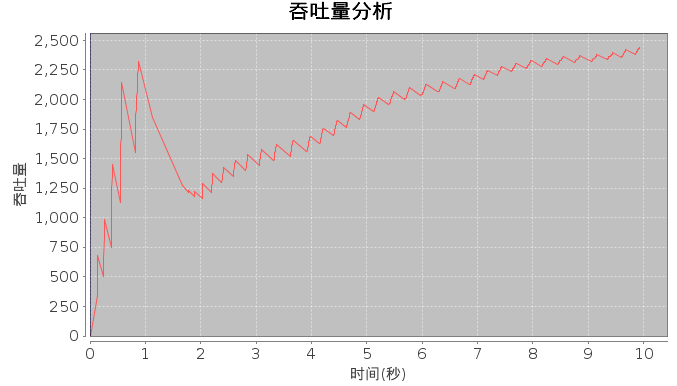


图2.20 TCP/SACK吞吐量分析

TCP/FACK吞吐量分析如图2.21所示

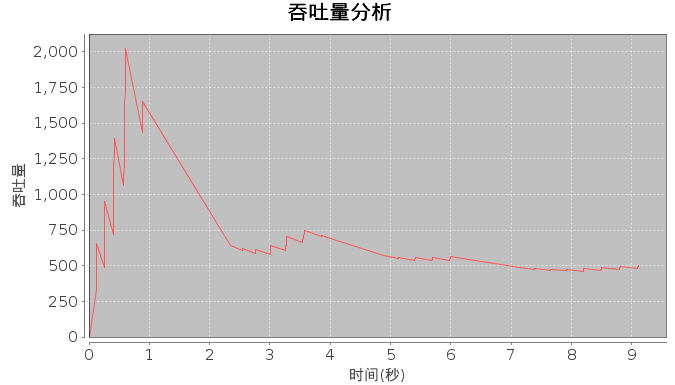


图2.21 TCP/FACK吞吐量分析

TCP/Vegas吞吐量分析如图2.22所示

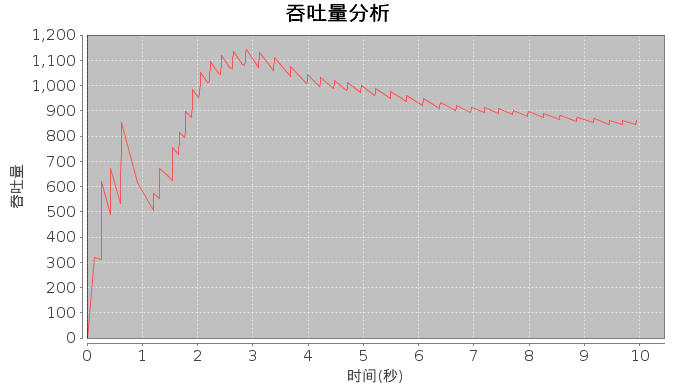


图2.20 TCP/Vegas吞吐量分析

1. 网络效率

TCP/Tahoe发送端节点发送的总比特数如图2.21所示，接收端节点接收到的总比特数如图2.22所示。

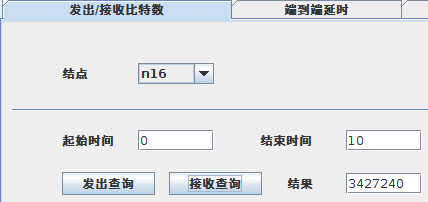
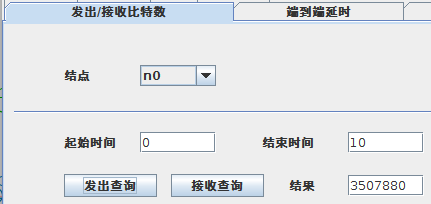


图2.21 TCP/Tahoe发送端 图2.22 TCP/Tahoe接收端

因此TCP/Tahoe的网络效率为3427240/3507880 = 0.997

TCP/Reno发送端节点发送的总比特数如图2.23所示，接收端节点接收到的总比特数如图2.24所示。

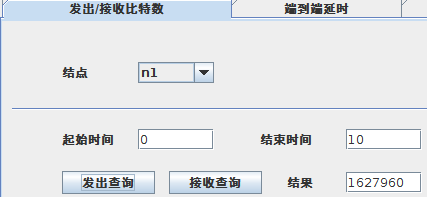
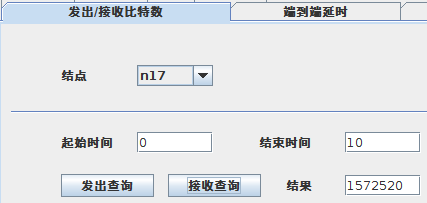
 

图2.23 TCP/Reno发送端 图2.24 TCP/Reno接收端

因此TCP/Reno的网络效率为1572520/1627960 = 0.966

TCP/NewReno发送端节点发送的总比特数如图2.25所示，接收端节点接收到的总比特数如图2.26所示。

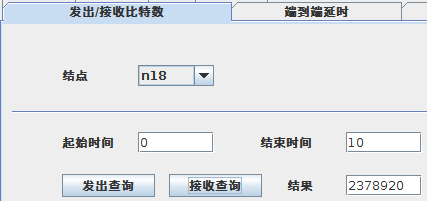
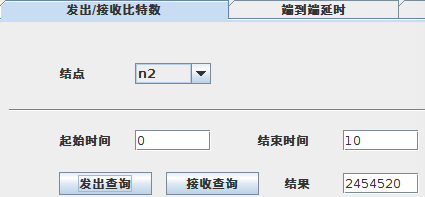


图2.25 TCP/NewReno发送端 图2.26 TCP/NewReno接收端

因此TCP/NewReno的网络效率为2378920/2454520 = 0.969

TCP/SACK发送端节点发送的总比特数如图2.27所示，接收端节点接收到的总比特数如图2.27所示。

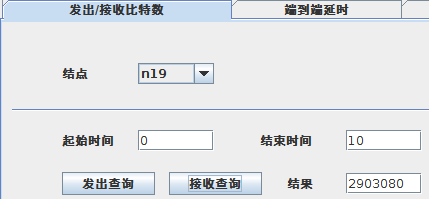
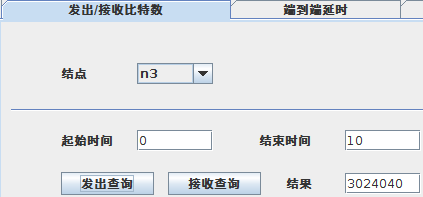


图2.27 TCP/SACK发送端 图2.28 TCP/SACK接收端

因此TCP/SACK的网络效率为2903080/3024040 = 0.960

TCP/FACK发送端节点发送的总比特数如图2.29所示，接收端节点接收到的总比特数如图2.30所示。

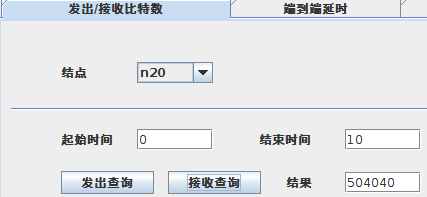
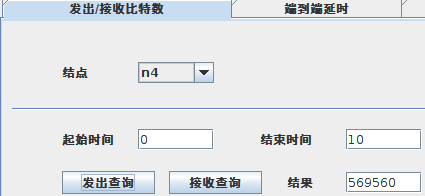


图2.29 TCP/FACK发送端 图2.30 TCP/FACK接收端

因此TCP/FACK的网络效率为504040/569560 = 0.885

TCP/Vegas发送端节点发送的总比特数如图2.31所示，接收端节点接收到的总比特数如图2.32所示。

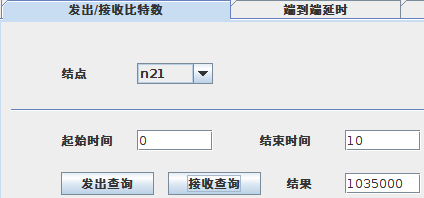
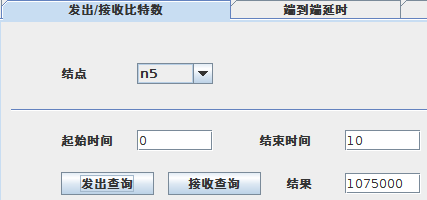


图2.31 TCP/Vegas发送端 图2.32 TCP/Vegas接收端

因此TCP/Vegas的网络效率为1035000/1075000 = 0.963

1. 带宽利用率

带宽利用率 = 吞吐量/带宽（带宽为100M）

由于无法精确得到结果，大概的比较如下

TCP/NewReno>TCP/Tahoe>TCP/SACK>TCP/Vegas>TCP/Reno>TCP/FACK

1. 拥塞控制能力

Tahoe 包括慢启动、拥塞避免和快速重传。在慢启动阶段：起初，发送端发送一个TCP报文段数据(通常是51 2 字节),即CW ND =1TCP报文段，以后当发送端每收到一个来 自接收端的确认 (ACK ),就对CWND加1 ,故CWND在慢启动阶段每一个周转时间(RTT )内加倍一次。当CWND增加到慢启动闭值SSTHRESH (通常设为64K字节),就进入拥塞避免阶段。 在拥塞避免阶段：发送端每收到一个确认，就对CWND加1/CWND。所以,慢启动阶段和拥塞避免阶段CWND相应每个RTT,分别是一个指数函数和一个线性函数。

TCP Reno。Reno修改Tahoe的快速重传为快速恢复，可以更有效的恢复丢包 从检测到丢包到收到重传包的确认帧这段时间称为快速重传/快速恢复阶段。从Reno运行机制中很容易看出，为了维持一个动态平衡，必须周期性地产生一定量的丢失，再加上AIMD机制--减少快，增长慢，尤其是在大窗口环境下，由于一个数据报的丢失所带来的窗口缩小要花费很长的时间来恢复，这样，带宽利用率不可能很高且随着网络的链路带宽不断提升，这种弊端将越来越明显。公平性方面，根据统计数据，Reno的公平性还是得到了相当的肯定，它能够在较大的网络范围内理想地维持公平性原则。由图2.12可以看出，在一段时间没有丢包后窗口的恢复过程中，相比其他算法要缓慢许多。

TCP NewReno在Reno快速恢复的基础上稍加了修改，可以恢复一个窗口内多个包丢失的情况。具体来讲就是：Reno在收到一个新的数据的ACK时就退出了快速恢复状态了，而NewReno需要收到该窗口内所有数据包的确认后才会退出快速恢复状态，从而更一步提高吞吐量。

TCP SACK就是改变TCP的确认机制，最初的TCP只确认当前已连续收到的数据，SACK则把乱序等信息会全部告诉对方，从而减少数据发送方重传的盲目性。比如说序号1，2，3，5，7的数据收到了，那么普通的ACK只会确认序列号4，而SACK会把当前的5，7已经收到的信息在SACK选项里面告知对端，从而提高性能，当使用SACK的时候，NewReno算法可以不使用，因为SACK本身携带的信息就可以使得发送方有足够的信息来知道需要重传哪些包，而不需要重传哪些包。

TCP FACK如果重传的包数据比较多的话，又会导致本来就很忙的网络就更忙了。所以，FACK用来做重传过程中的拥塞流控。

TCP的拥塞控制是基于丢包的，一旦出现丢包，于是调整拥塞窗口，然而由于丢包不一定是由于网络进入了拥塞，但是由于RTT值与网络运行情况有比较密切的关系，于是TCP Vegas利用RTT值的改变来判断网络是否拥塞，从而调整拥塞控制窗口。如果发现RTT在增大，Vegas就认为网络正在发生拥塞，于是开始减小拥塞窗口，如果RTT变小，Vegas认为网络拥塞正在逐步解除，于是再次增加拥塞窗口。由于Vegas不是利用丢包来判断网络可用带宽，而是利用RTT变化来判断，因而可以更精确的探测网络的可用带宽，从而效率更好。然而Vegas的有一个缺陷，并且可以说致命的，最终影响TCP Vegas并没有在互联网上大规模使用。这个问题就是采用TCP Vegas的流的带宽竞争力不及未使用TCP Vegas的流，这是因为网络中路由器只要缓冲了数据，就会造成RTT的变大，如果缓冲区没有溢出的话，并不会发生拥塞，但是由于缓存数据就会导致处理时延，从而RTT变大，特别是在带宽比较小的网络上，只要一开始传输数据，RTT就会急剧增大，这个在无线网络上特别明显。在这种情况下，TCP Vegas降低自己的拥塞窗口，但是只要没有丢包的话，从上面看到标准的TCP是不会降低自己的窗口的，于是两者开始不公平，再这样循环下去，TCP Vegas的效率就非常低了。其实如果所有的TCP都采用Vegas拥塞控制方式的话，流之间的公平性会更好，竞争能力并不是Vegas算法本身的问题。

### **2.3.3 第三项实验的步骤及结果分析**

1. 阈值变化

先进先出FIFO算法拥塞窗口分析如图2.33

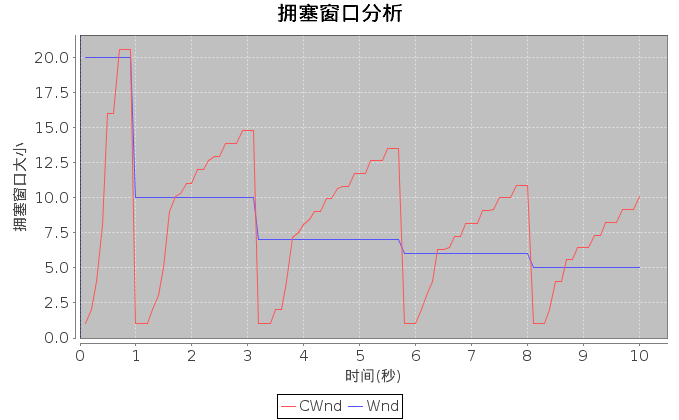


图2.33 FIFO算法拥塞窗口分析

随机早期检测算法RED拥塞窗口分析如图2.34

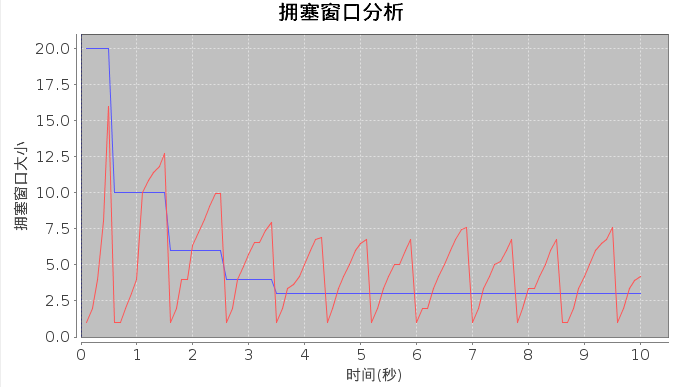


图2.34 RED拥塞窗口分析

公平排队算法FQ拥塞窗口分析如图2.35

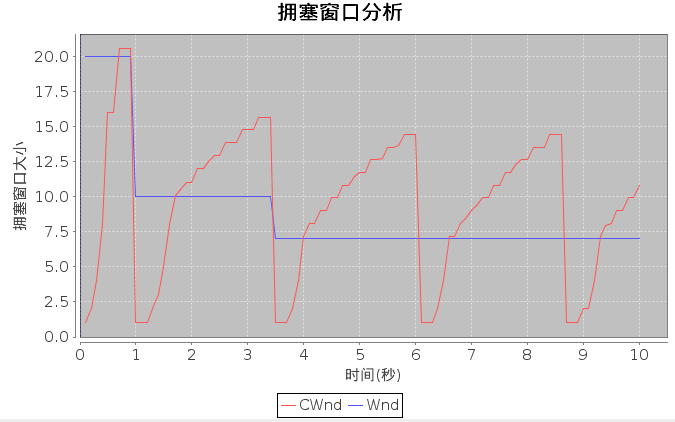


图2.35 FQ拥塞窗口分析

随机公平排队算法SFQ拥塞窗口分析如图2.36

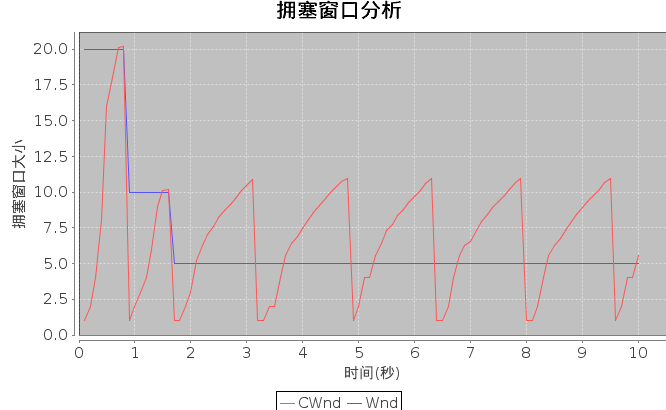


图2.36 SFQ拥塞窗口分析

差额轮循算法DRR拥塞窗口分析如图2.37

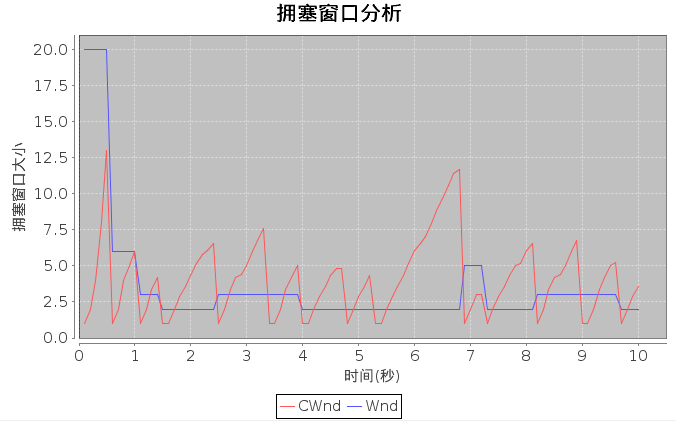


图2.37 DRR拥塞窗口分析

1. 吞吐量

先进先出FIFO吞吐量分析如图2.39所示

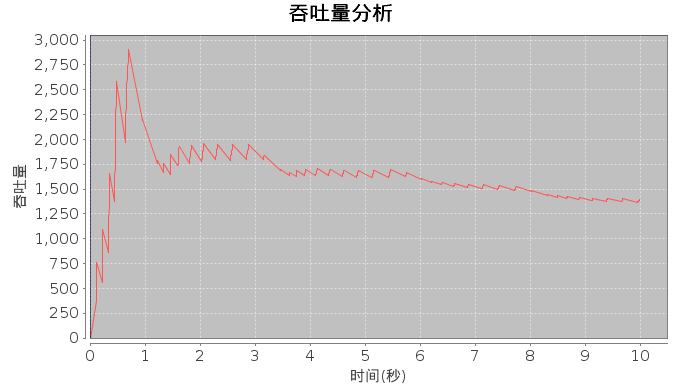


图2.39 FIFO吞吐量分析

随机早期检测算法RED吞吐量分析如图2.40所示

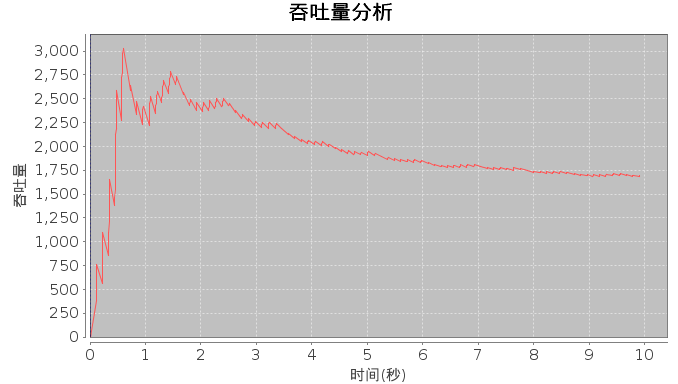


图2.40 RED吞吐量分析

公平排队算法FQ吞吐量分析如图2.41所示

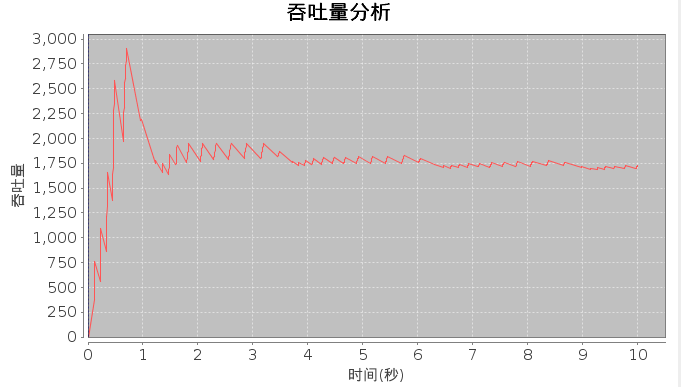
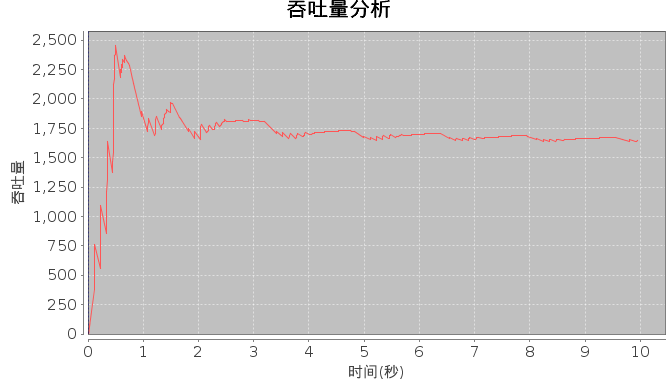


图2.41 FQ吞吐量分析

随机公平排队算法SFQ吞吐量分析如图2.42所示



2.42 SFQ吞吐量分析

差额轮循算法DRR吞吐量分析如图2.43所示

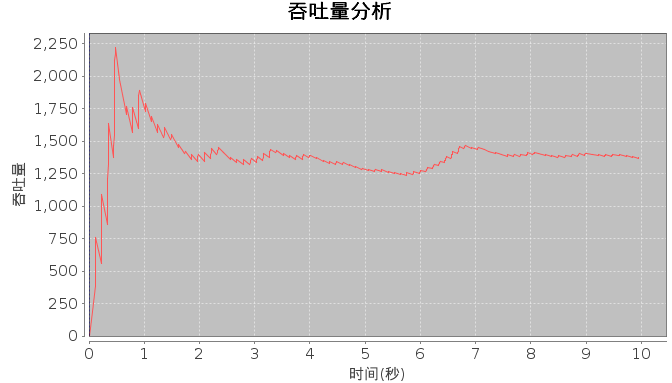


图2.43 DRR吞吐量分析

1. 网络效率

先进先出FIFO算法发送端节点发送的总比特数如图2.45所示，接收端节点接收到的总比特数如图2.46所示。

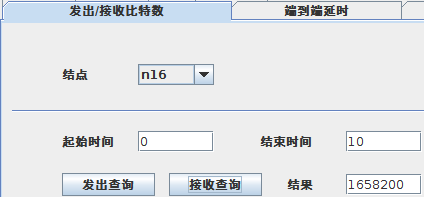
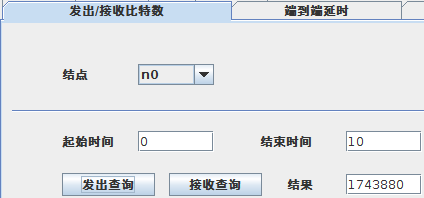


图2.45 FIFO发送端 图2.46 FIFO接收端

因此FIFO的网络效率为1658200/1743880 = 0.951

随机早期检测算法RED发送端节点发送的总比特数如图2.47所示，接收端节点接收到的总比特数如图2.48所示。

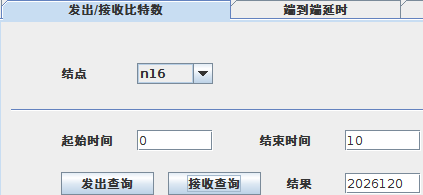
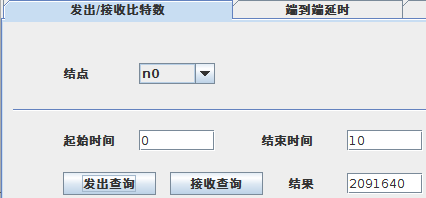


图2.47 RED发送端 图2.48 RED接收端

因此RED的网络效率为2026120/2091640 = 0.967

公平排队算法FQ发送端节点发送的总比特数如图2.49所示，接收端节点接收到的总比特数如图2.50所示。

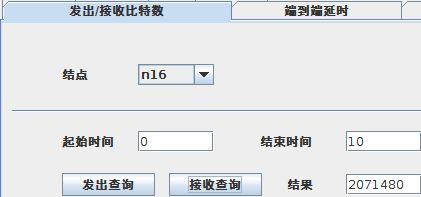
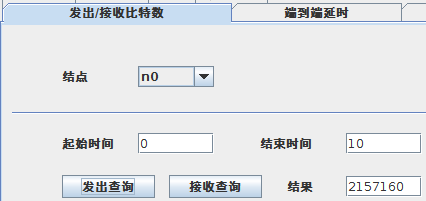


图2.49 FQ发送端 图2.50 FQ接收端

因此FQ的网络效率为2071480/2157160 = 0.960

随机公平排队算法SFQ发送端节点发送的总比特数如图2.51所示，接收端节点接收到的总比特数如图2.52所示。

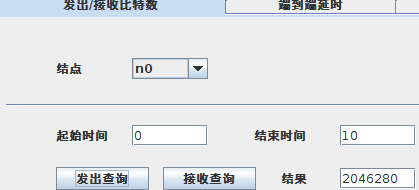
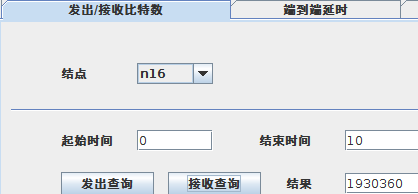
 

图2.51 SFQ发送端 图2.52 SFQ接收端

因此SFQ的网络效率为1930360/2046280 = 0.943

差额轮循算法DRR发送端节点发送的总比特数如图2.53所示，接收端节点接收到的总比特数如图2.54所示。

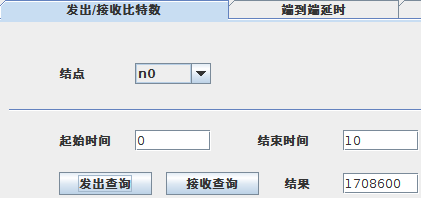
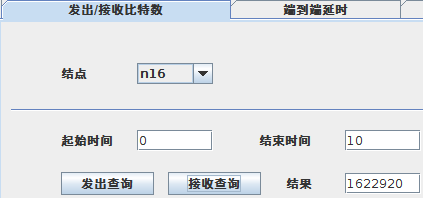
 

图2.53 DDR发送端 图2.54 DDR接收端

因此DDR的网络效率为1622920/1708600 = 0.950

1. 带宽利用率

FQ>RED>SFQ>FIFO>DDR

1. FIFO（先入先出队列，First In First Out Queuing）

FIFO按照时间到达的先后决定分组的转发次序。用户的业务流在某个设备能够获得的资源取决于分组的到达时机及当时的负载情况。Best-Effort报文转发方式采用的就是FIFO的排队策略。 如果设备的每个端口只有一个基于FIFO的输入或输出队列，那么恶性的应用可能会占用所有的网络资源，严重影响关键业务数据的传送。 每个队列内部报文的发送（次序）关系缺省是FIFO。

RED（随机早期检测randomearly detection） 若发生路由器的尾部丢弃，可能影响到很多条TCP连接，结果就是这许多的TCP连接在同一时间进入慢开始状态。这在术语中称为全局同步。全局同步会使得网络的通信量突然下降很多，而在网络恢复正常之后，其通信量又突然增大很多。

为避免发生网络中的全局同步现象，在路由器采用RED排队策略。基本思想：通过监控路由器输出端口队列的平均长度来探测拥塞，一旦发现拥塞逼近，就随机地选择连接来通知拥塞，使他们在队列溢出导致丢包之前减小拥塞窗口，降低发送数据速度，从而缓解网络拥塞。由于RED是基于FIFO队列调度策略的，并且只是丢弃正进入路由器的数据包，因此其实施起来也较为简单。

FQ（公平队列Fair Queuing）。FQ是为了公平地分享网络资源，尽可能使所有流的延迟和抖动达到最优而推出的。它照顾了各方面的利益，主要表现在：不同的队列获得公平的调度机会，从总体上均衡各个流的延迟。短报文和长报文获得公平的调度：如果不同队列间同时存在多个长报文和短报文等待发送，应当顾及短报文的利益，让短报文优先获得调度，从而在总体上减少各个流的报文间的抖动。

SFQ(Stochastic Fairness Queueing 随机公平队列)是公平队列算法家族中的一个简单实现.它的精确性不如其它的方法,但实现了高度的公平,需要的计算量亦很少。

SFQ算法主要针对一个TCP会话或者UDP流.流量被分成相当多数量的FIFO队列中,每个队列对应一个会话.数据按照简单轮转的方式发送, 每个会话都按顺序得到发送机会. 这种方式非常公平,保证了每一个会话都不会没其它会话所淹没.

SFQ之所以被称为"随机",是因为它并不是真的为每一个会话创建一个队列,而是使用一个散列算法,把所有的会话映射到有限的几个队列中去. 因为使用了散列,所以可能多个会话分配在同一个队列里,从而需要共享发包的机会,也就是共享带宽.为了不让这种效应太明显,SFQ会频繁地改变散列算法, 以便把这种效应控制在几秒钟之内(时间由参数设定).

## **2.4 其它需要说明的问题**

（除了上述问题外，你觉得还需要说明的内容可在本处撰写）

# 实验三 基于CPT的组网实验

## **3.1 环境**

（实验机器的硬件配置、系统软件组件、第三方软件）

## **3.2 实验要求**

（说明实验需要完成的工作，主要指实验指导手册上的要求）

## **3.3 基本部分实验步骤说明及结果分析**

### **3.3.1 组网实验的步骤及结果分析**

### **3.3.2 路由配置实验的步骤及结果分析**

### **3.3.3 VLAN划分实验的步骤及结果分析**

### **3.3.4访问控制配置实验的步骤及结果分析**

## **3.4 综合部分实验设计、实验步骤及结果分析**

### **3.4.1 实验设计**

### **3.4.2 实验步骤**

### **3.4.3 结果分析**

## **3.5 其它需要说明的问题**

（除了上述问题外，你觉得还需要说明的内容可在本处撰写）

# 心得体会与建议

## **4.1 心得体会**

## **4.2 建议**