**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 周铭昊 班级 CS1503班 学号 U201514559

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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：处理器Intel(R) Xeon(R) CPU E3-1231 v3 @3.40GHz，网络适配器Realtek PCIe GBE Family Controller，显示适配器NVIDIA GeForce GTX 1060 6GB。

系统软件组件：Windows 10专业版 64位操作系统，基于x64的处理器

开发平台：Qt Creator3.4.2 (based on Qt5.5.0)

使用的第三方组件：Sqlite数据库

### **1.1.2 运行平台**

Windows 10/8/7/XP系统，LINUX系统。

## **1.2 系统功能需求**

基于TCP和UDP协议实现一个即时通讯工具，具体功能要求包括：

1. 工具包括服务器端和客户端；
2. 具备用户注册、登录、找回密码功能（基于TCP协议）；
3. 两个用户如果同时在线，采用点到点通信方式进行聊天，服务器也不保存聊天内容（基于TCP协议）；
4. 支持离线消息（基于TCP协议）；
5. 支持点到点可靠文件传输（基于UDP协议）；
6. 存储在服务器端的数据需要进行强加密；
7. 支持不少于两组用户同时在线交流和传输文件；
8. 文件传输具有良好的性能，能够充分利用网路带宽；
9. 人机交互友好，软件易用性强。

## **1.3 系统设计**

### **1.3.1 概念设计**

结合需求分析的内容，我概括抽象出了系统功能框架图如图1.1。

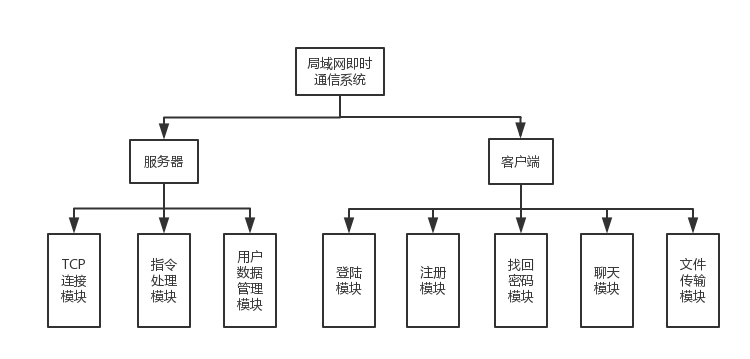


图1.1 系统功能框架图

1．TCP连接模块

本软件支持多用户同时在线，因此服务器需要同时与多个客户端建立连接，所以在服务器端需要建立一个TCP连接数组来保存各个已经建立好的连接。不单单是连接，服务器设置一个监听端口，一旦监听到客户端请求连入时，同意建立连接后取出用户的套接字存入数组中。

2．指令处理模块

本软件通过给服务器发送指令来控制实现各种功能。所有指令如表1-1所示。

表1-1

|  |  |
| --- | --- |
| 指令名称 | 功能和参数 |
| -login | 登录指令，参数格式为 用户名#密码 |
| -signup | 注册指令，参数格式为 用户名#密码#密保问题#密保答案 |
| -findback | 找回密码指令，参数格式为 用户名#密保答案#新密码 |
| -getback | 获取密保问题指令，参数格式为 用户名 |
| -logout | 离线指令，参数格式为 用户名 |
| -checkchat | 确认聊天对象指令，参数格式为 用户名#好友名 |
| -file | 发送文件指令，参数格式为 发送方#接收方#文件名#文件大小 |
| -tofile | 确认接收文件指令，参数格式为 是否接受#发送方#接收方#接收方ip#接收方端口 |
| -getoffline | 获取离线信息指令，参数格式为 用户名 |
| -send | 发送信息指令，参数格式为 发送方#接收方#信息 |

服务器处理完数据后，给客户端返回相应信息。

3． 用户数据管理模块

服务器端采用sqlite数据库来保存用户的数据，包括用户名，密码，密保问题，密保答案，在线状态。静态数据库db文件保存在本地，无法直接从中获取用户信息，因此也达到了本地数据加密的目的。

4． 登陆模块

登录模块中，客户端与服务器建立tcp连接并给服务器发送一条指令，当接收到服务器返回的登陆成功指令后，弹出成功提示，并进入聊天界面（登陆界面自动关闭）。否则提示错误信息。在登陆界面中可以设置连接的服务器的ip和端口，默认为本机回环127.0.0.1

5． 注册模块

注册模块中，客户端在填写完所有信息后，给服务器发送一条注册指令，当接收到服务器返回的登陆成功指令后，弹出成功提示。此时在服务器的数据库中添加一位用户信息。

6． 找回密码模块

本模块中，客户端在填写完所有信息后，给服务器发送一条找回密码指令，当接收到服务器返回的登陆成功指令后，弹出成功提示。此时在服务器的数据库中用户的密码进行更新。

7． 聊天模块

本模块中，用户首先选择聊天对象并确认，然后在客户端显示对方的在线情况，从而确定是在线发消息还是发送离线消息。在聊天框内输入想发送的信息并点击发送后，输入框内的信息清除，并显示到聊天框内。

8． 文件传输模块

本模块可分为发送文件和接受，发送方发送文件时，先设置一个端口（建议设置在6000-9000之间的四位数），然后在弹出的窗口中选择文件，选择完成后给服务器发送一条信息代表请求发送，等待接收方的反馈，接收方确认接受后，两个客户端之间通过udp传输文件数据；接收方拒绝后，服务器给发送方返回一条信息代表接收方拒绝。

### **1.3.2 数据库设计**

数据库（DB）是存储在计算机内、有组织、可共享的相关数据的集合。数据库中的数据按一定的数据模型组织、描述和存储，具有较小的冗余度，较高的数据独立性和扩展性，并可为各种用户共享。合理的数据库结构设计可以提高数据存储的效率，保证数据的完整和一致。同时，合理的数据库结构也有利于程序的实现。系统开发的总体任务是实现各种信息的系统化、规格化和自动化。

设计数据库系统时应该首先了解系统各个方面的需求，包括现有的以及将来可能增加的需求。本系统中对数据库的需求主要是保存用户名密码等信息，同时用一个标志位status来表示用户的在线情况，1代表在线，2代表离线。

数据流程如图1.2。

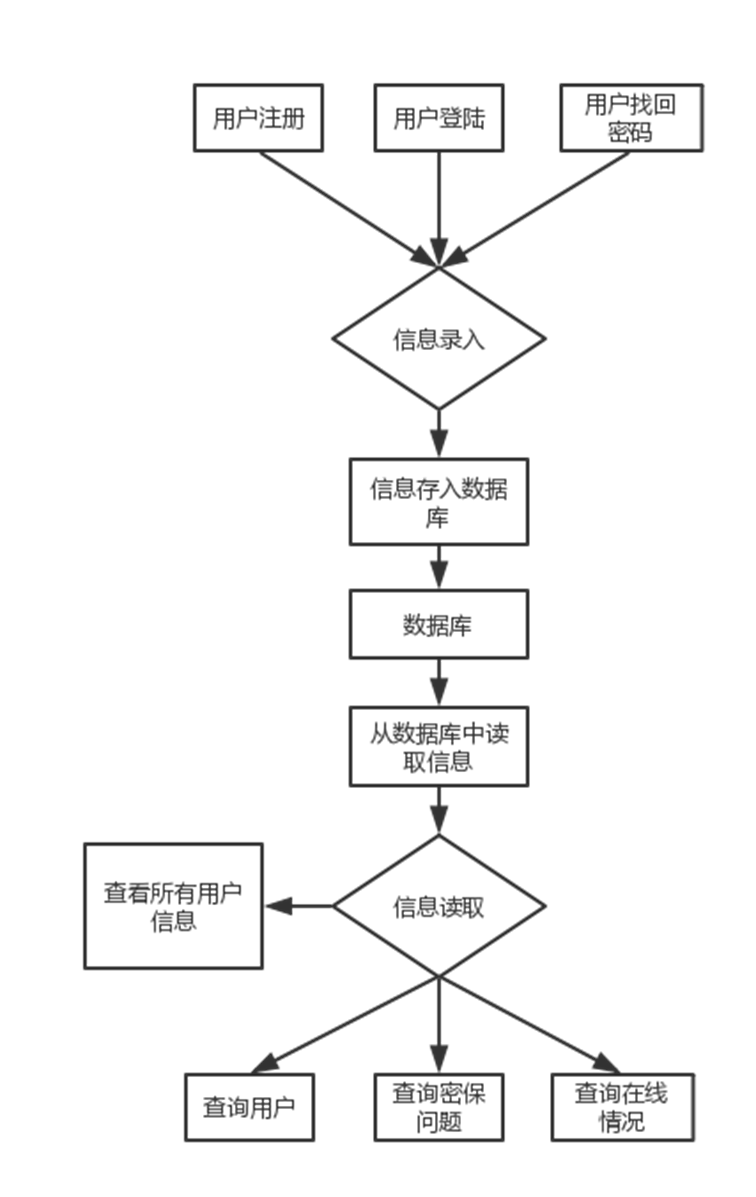


图1.2 局域网即时通信系统数据流程

### **1.3.3 数据库表格设计**

根据概要设计给出的数据库模型，我对数据库逻辑结构做了这样的设计：

在本系统中共设计了两个表，表名分别为：users， offline。

用户基本信息（表users）如表1-2。

表1-2 用户基本信息

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 可否为空 | 说明 |
| username | 文本 | Not null | 用户名 |
| password | 文本 | Not null | 密码 |
| Question | 文本 | Not null | 密保问题 |
| Answer | 文本 | Not null | 密保答案 |
| status | 数字 | Not null | 在线状态 |

离线信息（表offline）如表1-3。

表1-3 离线信息

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 可否为空 | 说明 |
| sender | 文本 | Not null | 发送方 |
| receiver | 文本 | Not null | 接收方 |
| message | 文本 | Null | 离线信息 |

### **1.3.4 启动界面设计**

使用Qt编写程序的一般步骤是，首先利用Qt的可视化编程工具来设计本程序的ui，生成框架程序，再往里边填写代码。这是一种“填空式”的编程方法，首先生成框架，然后根据目标程序的要求，看哪些地方需要修改，再往里面填写代码。

启动Qt创建一个名为ILLW\_client的工程，在弹出New Project对话框时选择Qt Widgets Application（桌面qt应用），在Qt Widgets Application的Detail中设置工程保存位置，将类名设置为clientwidget，基类设置为QWidget，按Finish按钮，建立ILLW\_client工程。

工程建立后，首先打开界面文件（后缀为.ui），在左侧的Filter中选择对应的控件拖动到主页面上，ui文件具体设计如图1.3所示，在安排好ui的所有控件并进行布局后，编译运行将可以看到如图1.4所示的启动界面。



图1.3 ui具体设计图

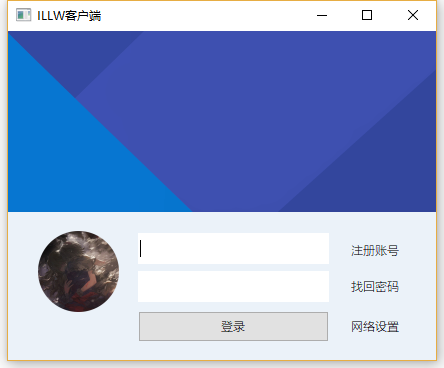


图1.4 系统界面主窗口

主窗口上有两个输入框（以及两个隐藏输入框），一个用于输入用户名，一个用于输入密码，有三个按钮（以及一个隐藏按钮）分别为：注册账号、找回密码、网络设置，单机网络设置按钮后，弹出隐藏部分，效果如图1.5。该部分用于设置客户端所连接的服务器IP和端口，设置完成后单击保存设置按钮则确认修改。

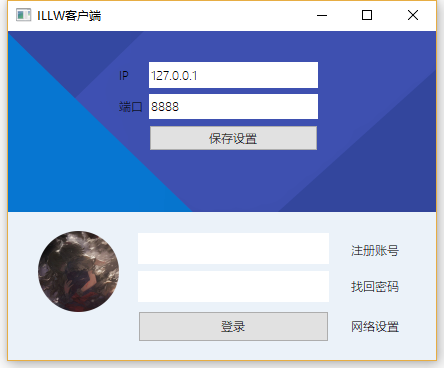


图1.5 系统界面主窗口2

主窗口在初始化时从路径 ../ILLW\_SOURCE/back.png 获取背景图片并进行绘制。

### 1.3.5 聊天系统设计

聊天系统主要实现的功能是：

1．显示本用户的用户名

2．选择聊天对象

3．获得聊天对象的在线状态

4．与选中的聊天对象进行即时通信

5．发送文件以及接收文件

6．获取离线消息

7．下线操作

聊天系统的ui主界面设计如图1.6所示，控件排布与其他聊天软件大致相似，其中的添加好友、删除好友两个按钮还未设置功能。当用户关闭本窗口时，即代表用户下线，此时给服务器发送一条下线指令，服务器收到后把本用户的在线状态status改为0。

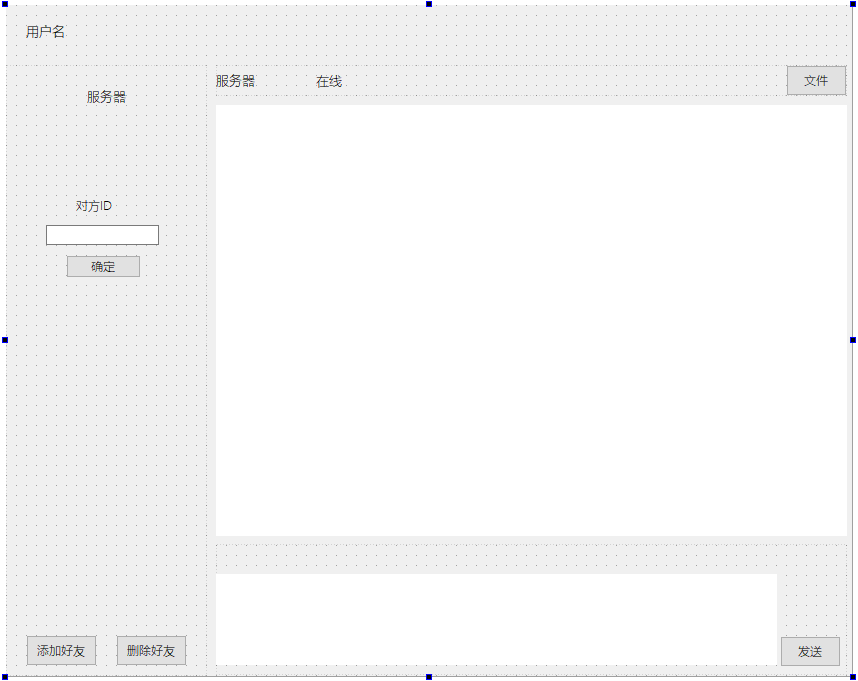


图1.6 聊天系统主窗口ui

### 1.3.6 服务器系统设计

服务器系统的主要功能就是对客户端发送过来的指令进行分析执行，然后返回结果给客户端。在本地建立一个db文件保存用户数据文件，服务器可以查看并且修改用户的信息。

服务器系统窗口如图1.7。

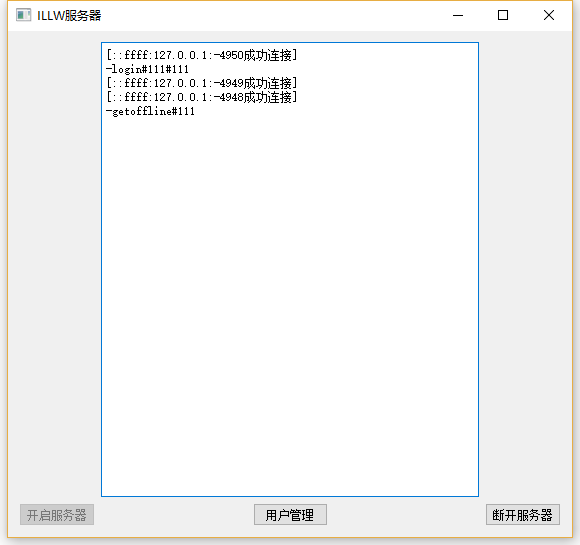


图1.7 服务器系统

## **1.4 系统实现**

**1.4.1 Windows10风格界面的实现**

实现Windows10风格界面是通过改变颜色、外形以及对实体边框进行修饰得出来的效果，主要是在ui文件中，对类的属性进行修改。主要涉及到QWidget、QAbstractButton、QPushButton、QLineEdit、QLabel几个Qt中的类。

要得到无边框的对话框，需要修改QLineEdit中的属性，在ui设计界面选中想要修改的对话框，切换到QLineEdit下，勾选frame即可。

按钮属性可以通过QWidget下的font来进行选择，本系统选择的字体为[微软雅黑 Light,9]。在QLabel下的pixmap中选择一张图片文件可以使得便签显示为一张图，用来表示头像。

经过以上的过程就能把系统界面做成Windows10的风格了，它的效果可以让界面显得不是那么粗糙。

**1.4.2 用户登录的实现**

用户登录模块的主要工作是：先与服务器建立TCP连接，随后获取用户名和密码对应的两个行编辑区的信息，并且把这些信息与登录命令组包成-login#用户名#密码的格式发送给服务器。 随后收到服务器返回的信息后，对信息进行分包，然后获取登陆提示是成功还是失败，如果成功，弹出成功提示并且进入到聊天界面；如果失败，仅弹出失败提示。

遵循以上的步骤，我在ILLW\_client工程下创建了clientwidget类，先在类中添加一些与登录操作有关的成员变量：

Ui::ClientWidget \*ui;//ui界面文件

QTcpSocket \*tcpSocket;//通信套接字

QString Server\_ip;//服务器ip

qint16 Server\_port;//服务器端口

按下登录按钮后，在ClientWidget::on\_buttonLogin\_clicked()里进行登录操作，代码如下：

void ClientWidget::on\_buttonLogin\_clicked()

{

//主动和服务器建立连接

tcpSocket->connectToHost(QHostAddress(Server\_ip),Server\_port);

//获取编辑区内容

username = ui->lineEditID->text();

QString passward=ui->lineEditPassword->text();

if(username=="" || passward=="")

{

QMessageBox::information(this,"警告","输入不能为空",QMessageBox::Ok);

return;

}

//组包并发送

QString loginflag="-login";//登录标志

QString data=loginflag+"#"+username+"#"+passward;

//用户名和密码

QByteArray message;

QDataStream out(&message,QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_4);

out<<data;

tcpSocket->write(message);//发送流

//tcpSocket->write(data.toLatin1());

connect(tcpSocket,&QTcpSocket::readyRead,

[=]()

{

QDataStream in(tcpSocket);

in.setVersion(QDataStream::Qt\_5\_4);

QString data;

in >> data;

QStringList list = data.split("#");//分割收到的字符串

if(list[0]=="-login"&&list[2]==username)//第一个命令为登录

{

if(list[1]=="true")

{

QMessageBox::information(this,"提示","登录成功",QMessageBox::Ok);//登录成功,关闭其他窗口

cw.init(Server\_ip,Server\_port,username);//初始化

cw.show();

sw.s\_Socket->disconnectFromHost();

fw.f\_Socket->disconnectFromHost();

tcpSocket->disconnectFromHost();

sw.close();

fw.close();

this->close();//关闭其他窗口

}

else

QMessageBox::information(this,"提示","登录失败",QMessageBox::Ok);//登录失败

}

}

);

}

用户成功登录窗口如图1.8，单击确认后进入聊天窗口。

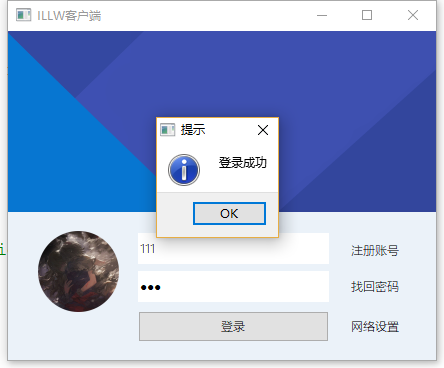


图1.8 用户成功登录窗口

用户登录失败窗口如图1.9

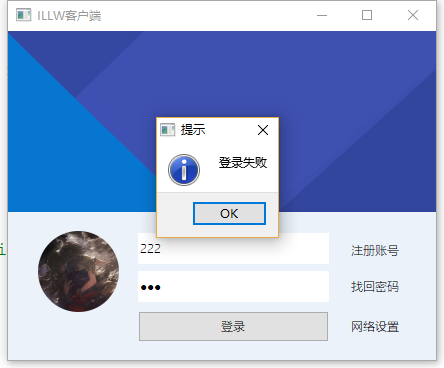


图1.9 用户登录失败窗口

**1.4.3 用户注册的实现**

用户注册功能的实现基本和上面说到的用户登录的实现过程一样，都分为：与服务器建立连接，获取编辑区内容，向服务器发送组包后的数据，接收服务器的数据并分包，弹出相应提示。

遵循以上的步骤，我在ILLW\_client工程下创建了signupwidget类，类的定义如下：

class SignupWidget : public QWidget

{ Q\_OBJECT

public:

explicit SignupWidget(QWidget \*parent = 0);

~SignupWidget();

QTcpSocket \*s\_Socket;//通信套接字

QString s\_ip;//服务器ip

qint16 s\_port;//服务器端口

private slots:

void on\_buttonSignup\_clicked();

private:

Ui::SignupWidget \*ui;

QString username;

};

用户注册窗口如图1.10。

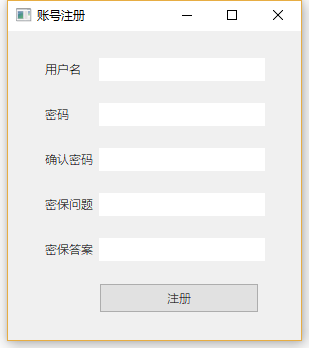


图1.10用户注册窗口

注册窗口中用到的所有控件如图1.11。

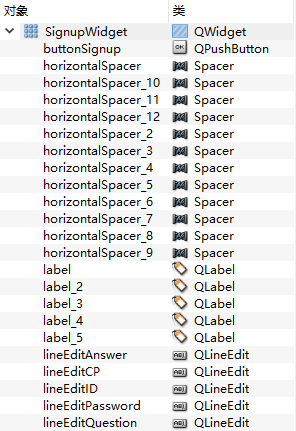


图1.11 注册窗口的控件

**1.4.4 用户找回密码的实现**

找回密码功能的实现基本和上面说到的用户登录的实现过程一样，都分为：与服务器建立连接，获取编辑区内容，向服务器发送组包后的数据，接收服务器的数据并分包，弹出相应提示。只是其中多了一个步骤：获取用户的密保问题。用户在找回密码时，先输入用户名，然后单击获取密保问题按钮，然后服务器返回密保问题显示在编辑框内，随后用户再输入其他信息。

遵循以上的步骤，我在ILLW\_client工程下创建了findback类，类的定义如下：

class Findback : public QWidget

{

Q\_OBJECT

public:

explicit Findback(QWidget \*parent = 0);

~Findback();

QTcpSocket \*f\_Socket;//通信套接字

QString f\_ip;//服务器ip

qint16 f\_port;//服务器端口

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::Findback \*ui;

QString username;

};

找回密码窗口如图1.12。



图1.12 找回密码窗口

**1.4.5 聊天系统的实现**

聊天模块的主要工作是：先获取好友，随后获取用户名和密码对应的两个行编辑区的信息，并且把这些信息与登录命令组包成-login#用户名#密码的格式发送给服务器。 随后收到服务器返回的信息后，对信息进行分包，然后获取登陆提示是成功还是失败，如果成功，弹出成功提示并且进入到聊天界面；如果失败，仅弹出失败提示。在用户登录时给服务器发送一条信息，让服务器把发送给本用户的离线信息转发过来，以实现自动获取离线消息的功能。

遵循以上的步骤，我在ILLW\_client工程下创建了ChatWidget类，类的定义如下：

class ChatWidget : public QWidget

{

Q\_OBJECT

public:

explicit ChatWidget(QWidget \*parent = 0);

~ChatWidget();

void init(QString Server\_ip,qint16 Server\_port, QString username);

QTcpSocket \*c\_Socket;//通信套接字

QString c\_ip;//服务器ip

qint16 c\_port;//服务器端口

QString username;

QString friendname;

signals:

protected:

void closeEvent(QCloseEvent \*event);//关闭窗口事件

private slots:

void on\_pushButton\_clicked();

void on\_buttonSend\_clicked();

void readMessage();

void on\_buttonFile\_clicked();

private:

Ui::ChatWidget \*ui;

TransWidget tw;

};

用户登录后，在ChatWidget::init ()里进行获取离线信息操作，代码如下：

void ChatWidget::init(QString Server\_ip,qint16 Server\_port, QString username1)

{

c\_ip = Server\_ip;

c\_port = Server\_port;

username = username1;

ui->username->setText(username);//设置名字

c\_Socket->connectToHost(QHostAddress(c\_ip),c\_port);//和服务器建立连接

//组包并发送一条-getoffline指令

QString getofflineflag="-getoffline";//获取离线信息标志

QString data=getofflineflag+"#"+username;

//用户名

QByteArray message;

QDataStream out(&message,QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_4);

out<<data;

c\_Socket->write(message);//发送流

connect(c\_Socket,&QTcpSocket::readyRead,

this,&ChatWidget::readMessage);

tw.init(Server\_ip, Server\_port, username1);

}

聊天窗口如图1.13，离线消息获取样例如图1.14

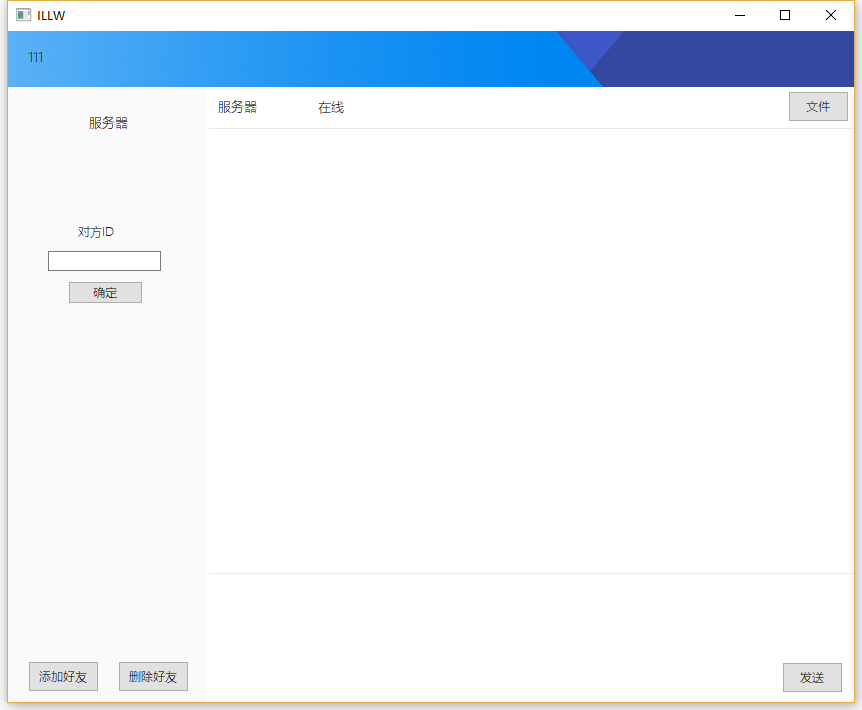


图1.13 聊天窗口

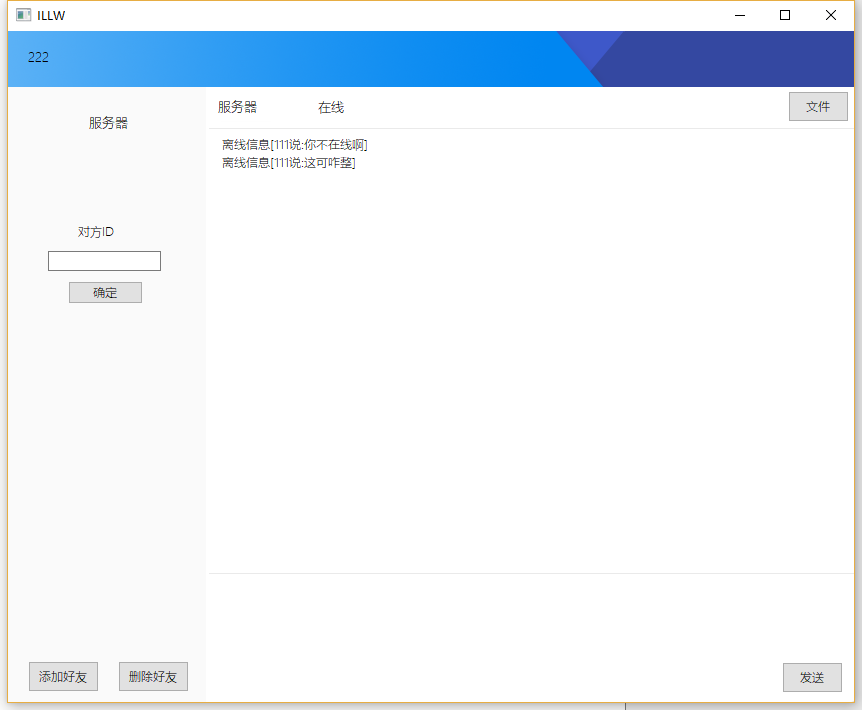


图1.14 离线消息获取样例

**1.4.6 文件传输系统的实现**

文件传输模块的主要工作流程是：先获取好友，单击文件按钮，绑定一个端口（建议值6000~9000之间），选择想要发送的文件 。随后接收方弹出一个窗口，可以选择是否接收。确认接收后双方开始传输数据。

在实际操作时，发现发送文件时会时画面卡住，解决方案是新建一个子线程，把发送文件的操作放在线程中进行，这样不会影响到主线程的运行。文件传输协议采用自定义的可靠UDP传输，在对文件进行分包后（一个包8000字节），对每一个包进行编号，传输过程中，接收方接收到对应的编号的文件以后，返回一个对应编号的ack包，只有当接收到对应的ack时，发送方才开始发送下一个包。如果这个ack包丢失，在发送方中定义一个计时器，超过一定时间后进行重传。

遵循以上的步骤，我在ILLW\_client工程下创建了TransWidget类和MyThread类，其中MyThread类的定义如下：

class MyThread : public QObject

{

Q\_OBJECT

public:

explicit MyThread(QObject \*parent = 0);

//线程处理函数

void initSocket();

void setFlag(bool flag= true);

bool isStop;

bool resendflag;

void sendData();

void readPendingDatagrams();

void timerUpDate();

QUdpSocket \*UdpSender;

QFile file;

QString fileName; //保存文件路径

QString ipRec;//接收方ip

int portRec;//接收方端口

QString filee;

qint64 fileSize; //文件大小信息

QString localip;

int port;//设定的端口

QByteArray lastline;

QByteArray line;

QByteArray lastdatagram;

QTimer \*timer;

int m\_nTimerID;

signals:

void mySignal(int sendcount);

void finSignal();

public slots:

};

两个用户确认进行文件传输后，绑定端口，进行udp连接，发送方udp连接的操作在函数MyThread::initSocket()中进行，主要代码如下：

void MyThread::initSocket()

{

if(isStop == false)

{

k=0;

resendflag = false;

file.setFileName(fileName);

if (!file.open(QIODevice::ReadOnly))

{

return;

}

UdpSender = new QUdpSocket(this);

UdpSender->bind(QHostAddress::Any, port);

connect(UdpSender,&QUdpSocket::readyRead,this,&MyThread::readPendingDatagrams);//收到数据时调用

timer=new QTimer(this);

connect(timer,&QTimer::timeout,this,&MyThread::timerUpDate);

sendData();//send第一个包

}

}

发送方发送文件窗口如图1.15所示

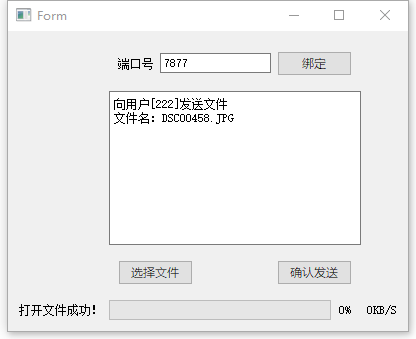


图1.15 发送文件窗口

接收方接收文件窗口如图1.16所示

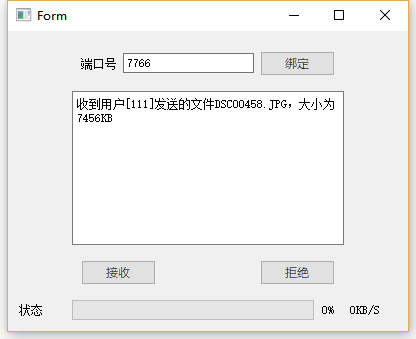


图1.16 接受文件窗口

文件传输完毕后，显示传输速度，两窗口如图1.17所示

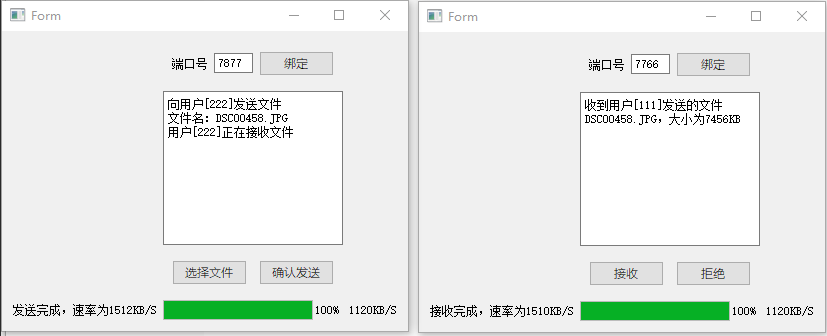


图1.17 文件传输完毕

**1.4.7 服务器指令系统的实现**

服务器指令系统的功能是：获取相应指令后，对数据进行分包，同时进行相应操作，如表1-1所示，共有10种指令，

服务器获取到的指令将会在主面板上显示，如图1.18所示。

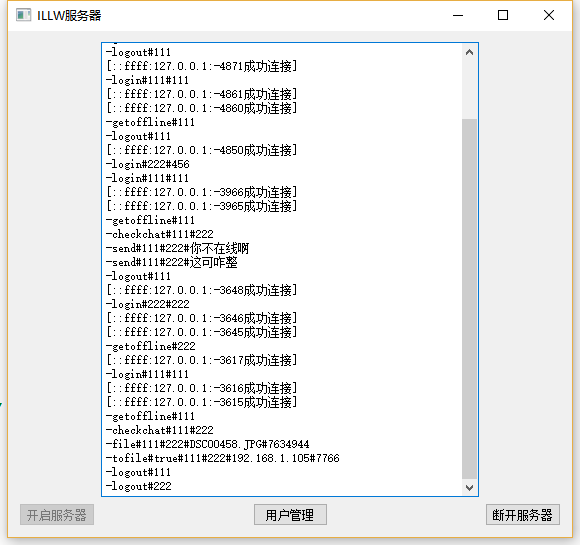


图1.18 服务器主面板

**1.4.8 用户数据管理的实现**

用户数据管理主要用到数据库中的两张表，一个是用户信息users表，另一个是离线信息offline表，两张表的具体信息可以参见表4-1和表4-2。用户的每一个操作都有一个对应的函数来控制数据库的变化，以下简要举一个例子

登录操作：首先查询对应的用户名，然后比对用户名和密码是否匹配，如果没有查询到用户或者密码不匹配，则返回一个bool类型的FALSE代表登陆失败，如果用户名和密码匹配成功，则返回一个bool类型的TRUE代表登陆成功，随后把数据库中该用户的status更新为1。的具体代码如下：

bool Sqlite::login(QString username,QString password)//登录

{

//开启一个事务

QSqlDatabase::database().transaction();

QSqlQuery query;

int row = 0;

//qDebug()<<model->rowCount();

while (row < model->rowCount())

{

QSqlRecord record = model->record(row);

if ((record.value("username") == username)&&(record.value("password") == password))

{

//成功登录后status变为1

query.prepare("UPDATE users SET status=:status where username=:username");

query.bindValue(":status",1);

query.bindValue(":username",username);

query.exec();//执行SQL语句的

model->select();//把操作后的表显示出来，这句可能容易出错

QSqlDatabase::database().commit();//提交操作

return true;

}

else

row++;

}

return false; //如果未查询到则返回false

}

用户的操作可以影响数据库中的内容，而服务器端除了可以查询用户信息之外，还可以删除选定的用户信息，让这位用户无法再用自己的账号登陆，除非重新注册账号。

用户管理界面如图1.19所示。

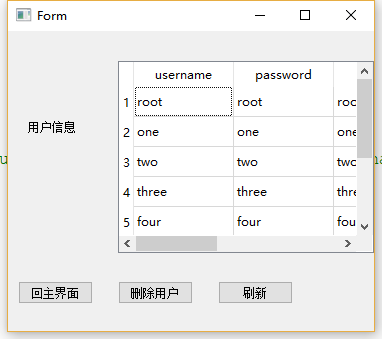


图1.19 用户管理界面

删除用户主要步骤为取出选中的模型索引，随后删除所有行，代码如下：

void Sqlite::on\_buttonDel\_clicked()

{

QItemSelectionModel \*sMode1 = ui->tableView->selectionModel();//获取选中的模型

//取出模型中的索引

QModelIndexList list = sMode1->selectedRows();

//删除所有行

for(int i = 0;i<list.size();i++)

{

model->removeRow(list.at(i).row());

//modeloff->removeRow(list.at(i).row());

}

}

删除用户three后，用户管理界面如图1.20所示。

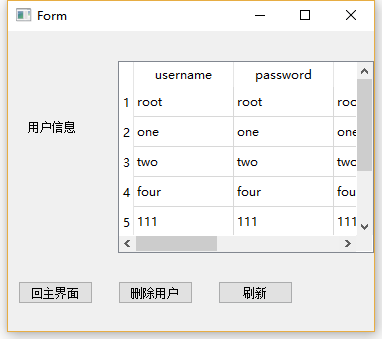


图1.20 删除用户three之后的用户信息

## **1.5 系统测试及结果说明**

删除用户three后，本程序的系统测试我想通过一位用户的历程来进行，从注册，登录，找回密码，获取离线信息，发送文件共六项测试。

1. 注册测试

注册一位名为TEST的用户，把他的密码、密保问题、密保答案都设置为TEST。

注册成功后界面如图1.21所示。随后在服务器端查询用户数据，如图1.22所示，新注册用户TEST成功的进入了用户列表，注册功能完成。



图1.21 用户注册成功界面

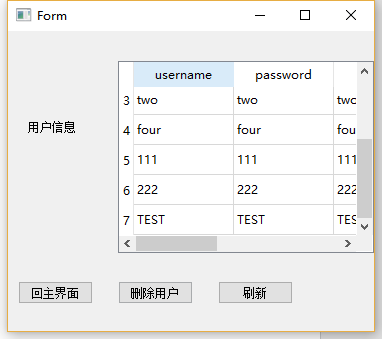


图1.22 删除用户three之后的用户信息

1. 登录与离线测试

TEST用户回到客户端主界面，输入用户名与密码，弹出登录成功提示。此时在用户信息界面可以看到，TEST的status一项变成了1，如图1.23(a)所示，标识用户登录成功。当用户离线后，TEST的status一项变成0，如图6.3(b)所示。登陆与离线测试完成。

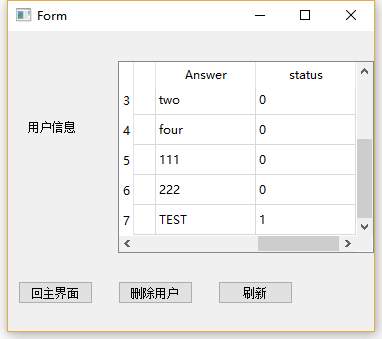
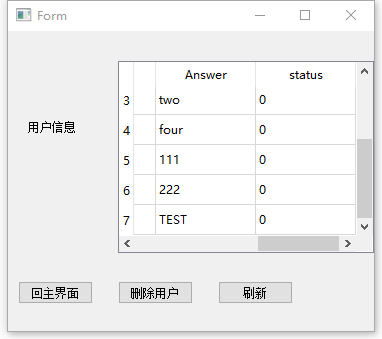
 

图6.3(a) 图6.3(b)

1. 找回密码测试

当用户忘记密码或者想要修改密码时，单击找回密码按钮，通过回答密保问题来核实身份，身份核实成功后可以进行密码修改。这项测试中，我们把TEST用户的密码改为111。找回密码界面如图6.4所示。

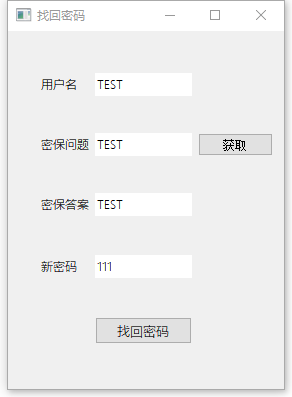


图6.4 TEST用户找回密码

单击找回密码按钮后，回到服务器端查看TEST用户的密码，发现已经被修改成111，如图6.5所示。找回密码测试完成。

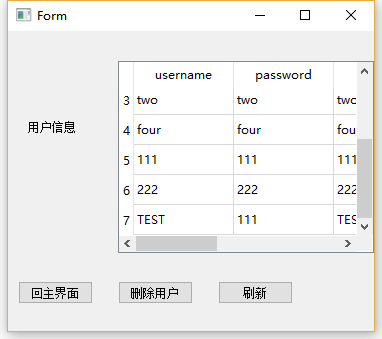


图6.5 TEST用户更新密码

1. 离线信息功能

在root用户还未登录时，TEST用户给root发送一条消息，那么这条消息即为离线信息，在root用户之后自动获取，获取的结果如图6.6所示。离线信息功能测试完成。

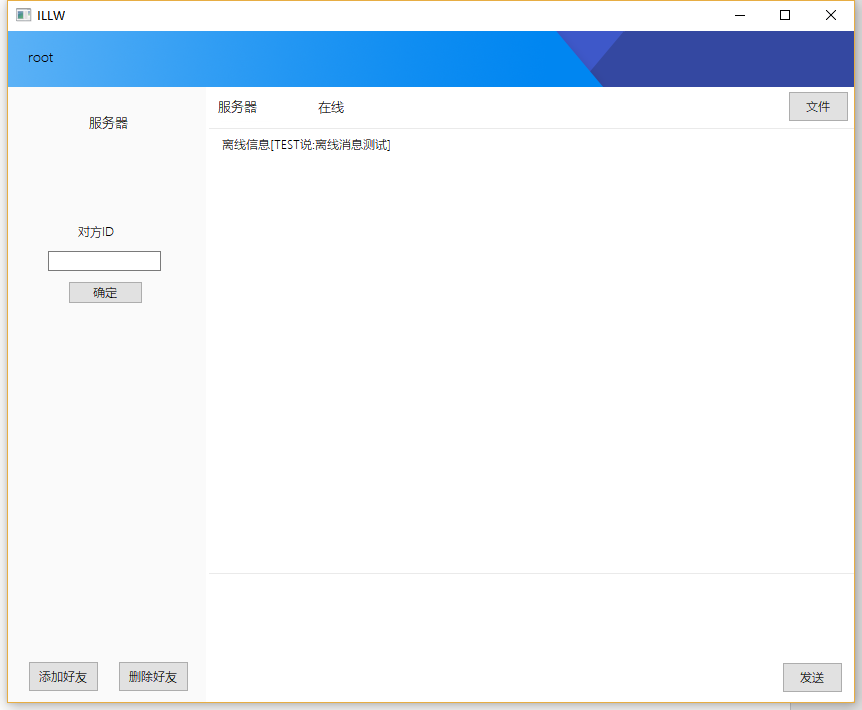


图6.6 离线消息测试

1. 文件传输测试

只有两个在线的用户之间才可以互传文件，那么将root和TEST两个用户都上线，传输一张图片文件。传输结果如图6.7所示，文件传输测试完成

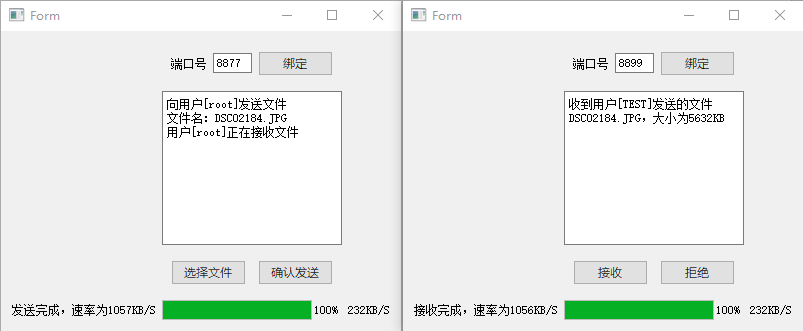


图6.7 传输文件结果

## **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

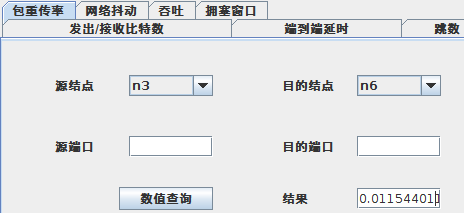
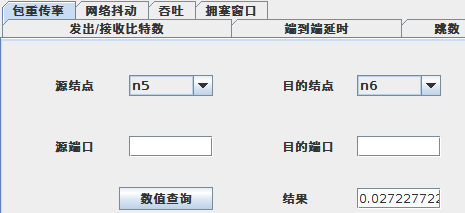
 

图2.1 TCP最小包重传率 图2.2 TCP最大包重传率

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

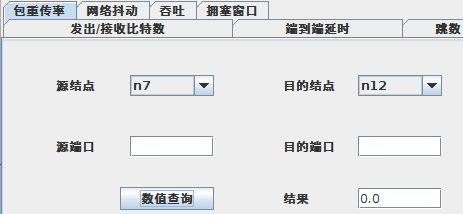


图2.3 UDP包重传率

1. 网络抖动

TCP网络抖动分析如图2.4所示

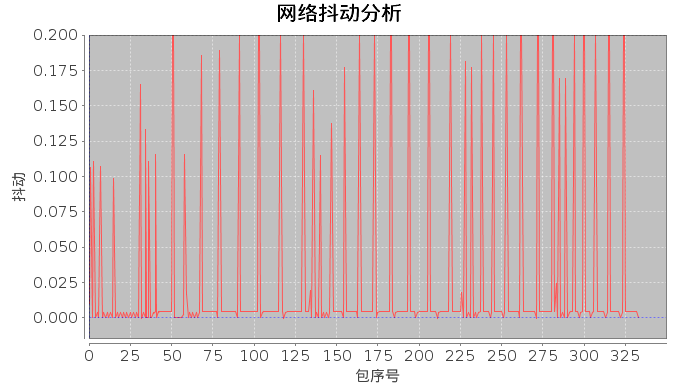


图2.4 TCP网络抖动分析

UDP网络抖动分析如图2.5所示

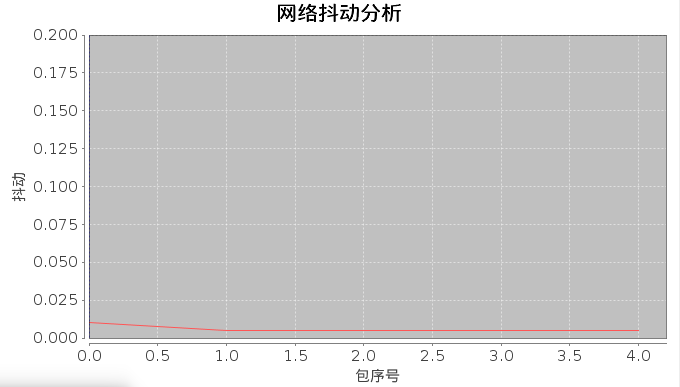


图2.5 UDP网络抖动分析

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

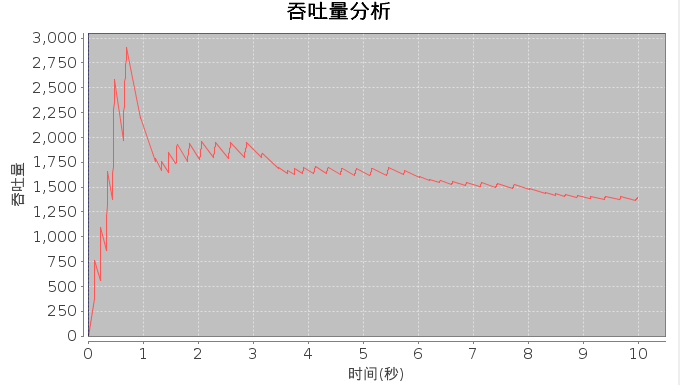


图2.6 TCP网络吞吐量

UDP网络吞吐量如图2.7所示

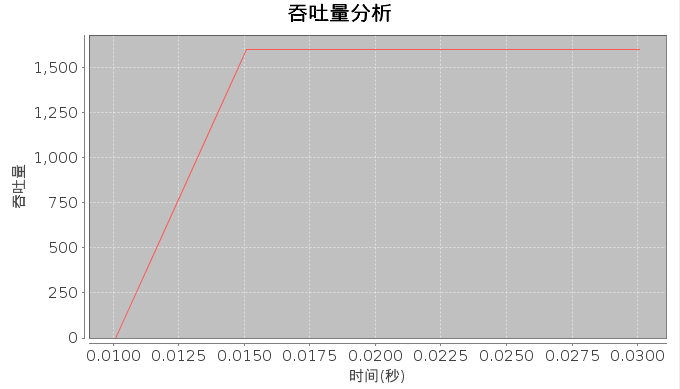


图2.7 UDP网络吞吐量

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

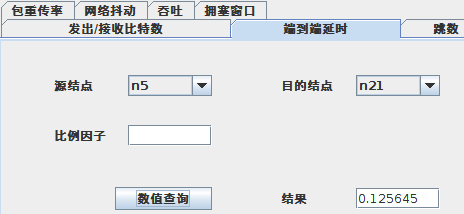
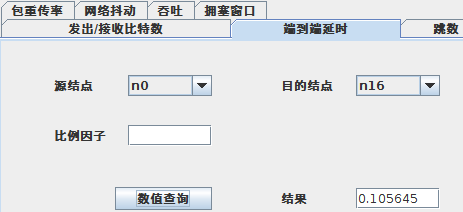
 

图2.8 TCP端到端最大延时 图2.9 TCP端到端最小延时

UDP由于没有回包，无法计算端到端延时，如图2.10所示

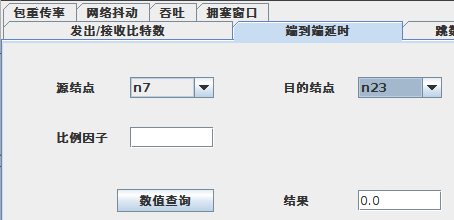


图2.10 UDP端到端延时

### **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

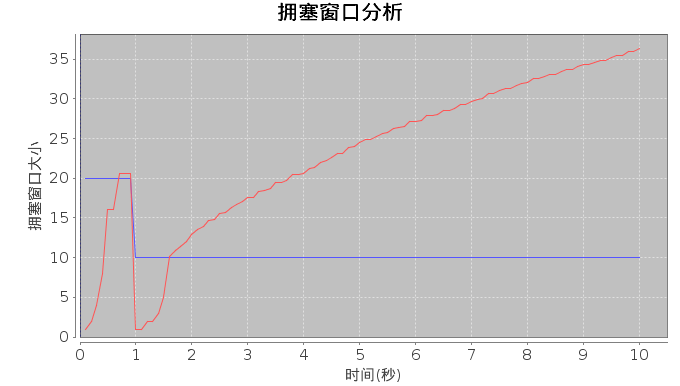


图2.11 TCP/Tahoe拥塞窗口分析

TCP/Reno拥塞窗口分析如图2.12所示

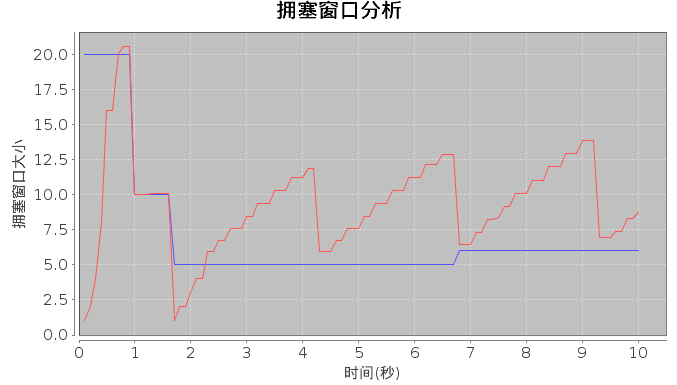


图2.12 TCP/Reno拥塞窗口分析

TCP/NewReno拥塞窗口分析如图2.13所示

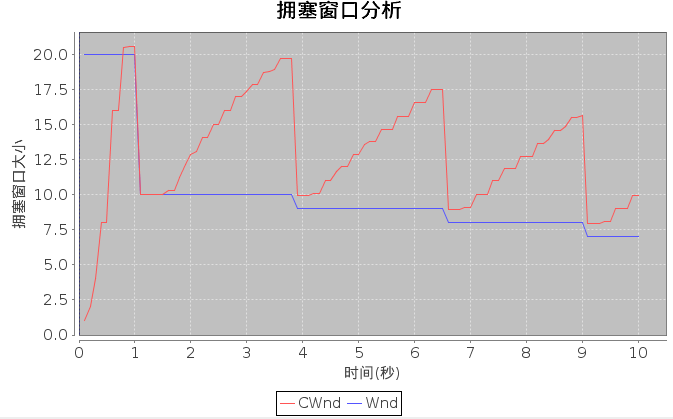


图2.13 TCP/NewReno拥塞窗口分析

TCP/SACK拥塞窗口分析如图2.14所示

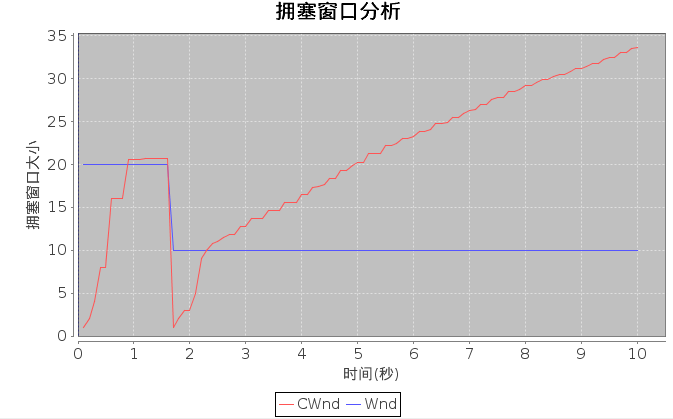


图2.14 TCP/SACK拥塞窗口分析

TCP/FACK拥塞窗口分析如图2.15所示

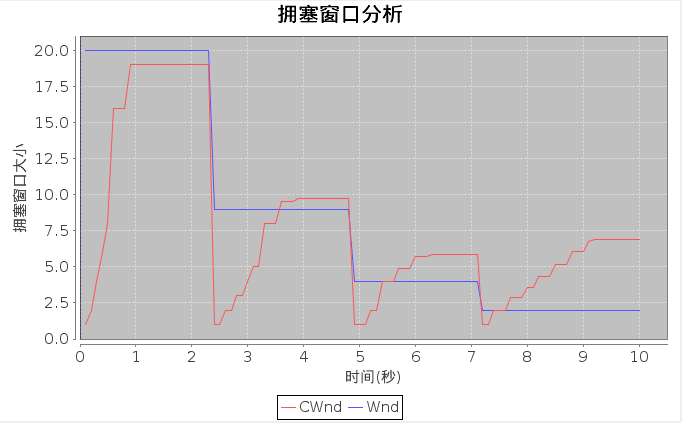


图2.15 TCP/FACK拥塞窗口分析

TCP/Vegas拥塞窗口分析如图2.16所示

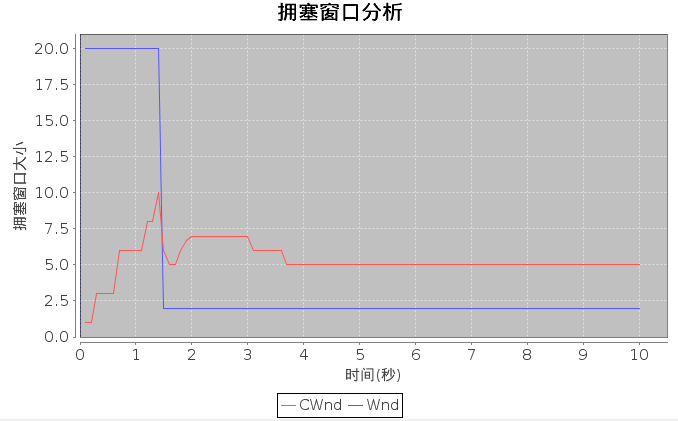


图2.16 TCP/Vegas拥塞窗口分析

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

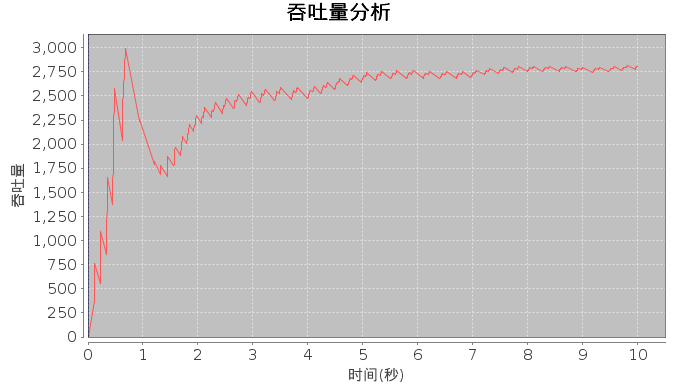


图2.17 TCP/Tahoe吞吐量分析

TCP/Reno吞吐量分析如图2.18所示

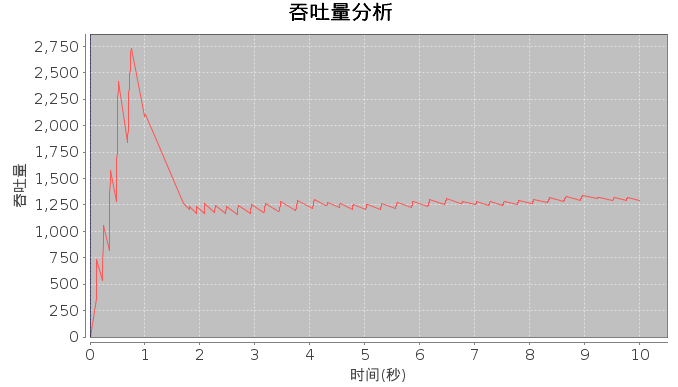


图2.18 TCP/Reno吞吐量分析

TCP/NewReno吞吐量分析如图2.19所示

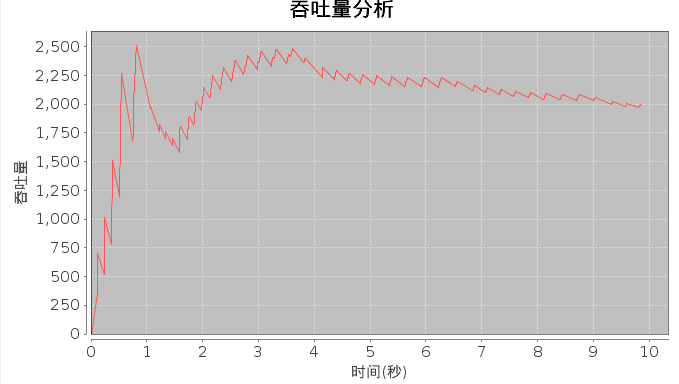


图2.19 TCP/NewReno吞吐量分析

TCP/SACK吞吐量分析如图2.20所示

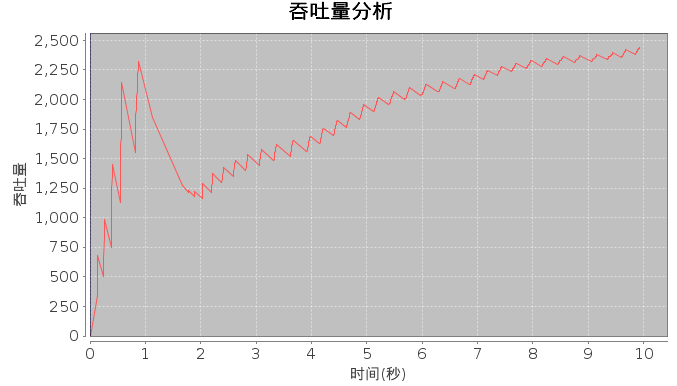


图2.20 TCP/SACK吞吐量分析

TCP/FACK吞吐量分析如图2.21所示

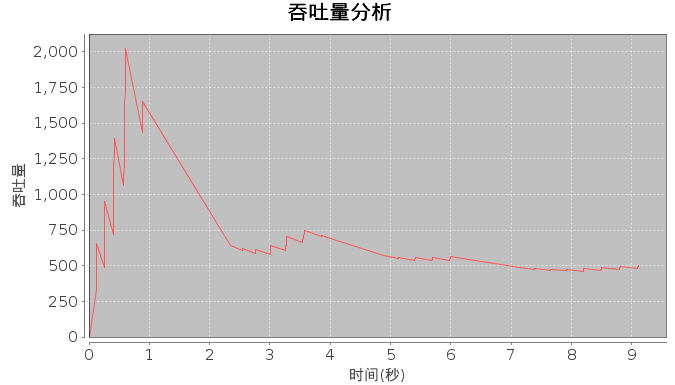


图2.21 TCP/FACK吞吐量分析

TCP/Vegas吞吐量分析如图2.22所示

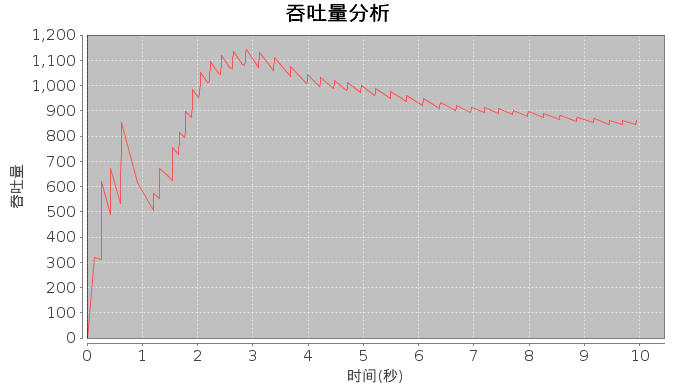


图2.20 TCP/Vegas吞吐量分析

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。

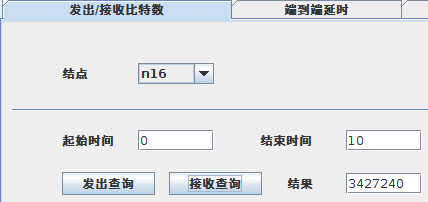
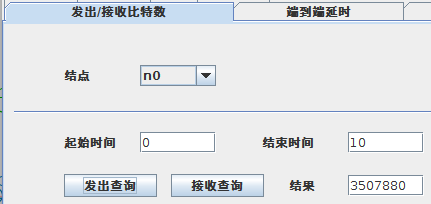


图2.21 TCP/Tahoe发送端 图2.22 TCP/Tahoe接收端

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。

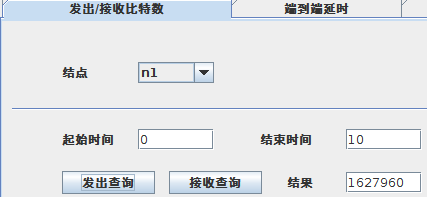
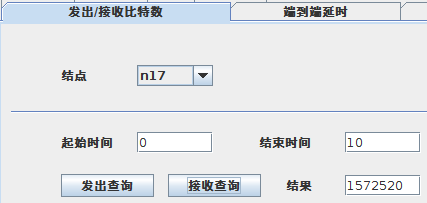
 

图2.23 TCP/Reno发送端 图2.24 TCP/Reno接收端

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。

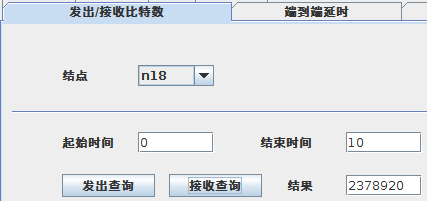
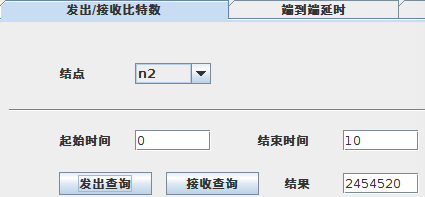


图2.25 TCP/NewReno发送端 图2.26 TCP/NewReno接收端

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。

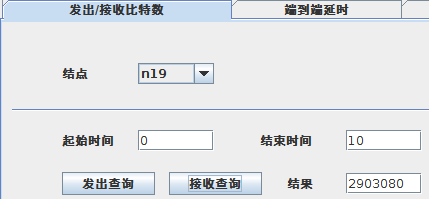
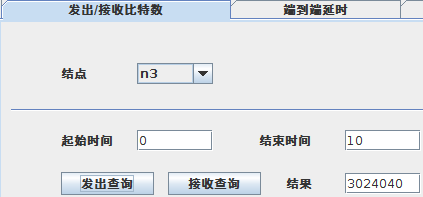


图2.27 TCP/SACK发送端 图2.28 TCP/SACK接收端

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。

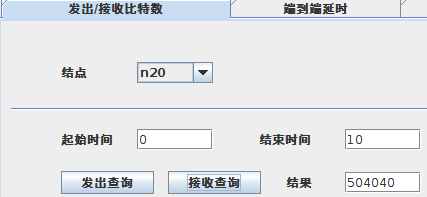
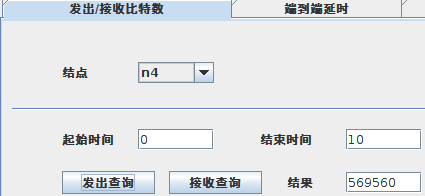


图2.29 TCP/FACK发送端 图2.30 TCP/FACK接收端

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。

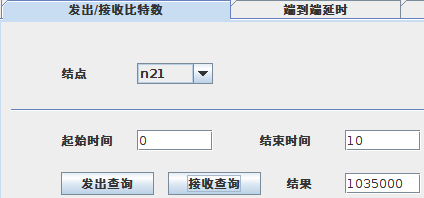
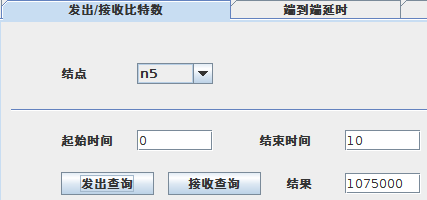


图2.31 TCP/Vegas发送端 图2.32 TCP/Vegas接收端

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

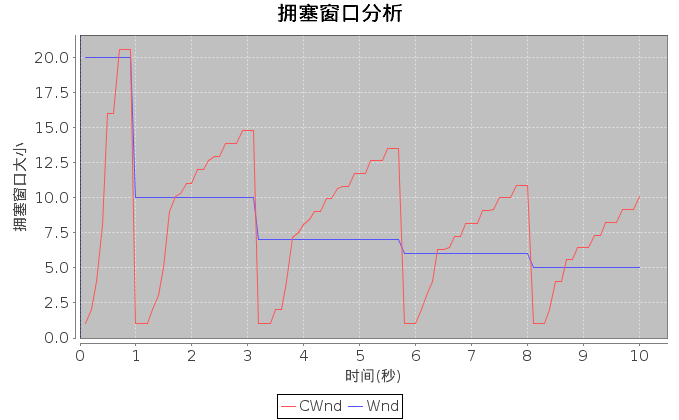


图2.33 FIFO算法拥塞窗口分析

随机早期检测算法RED拥塞窗口分析如图2.34

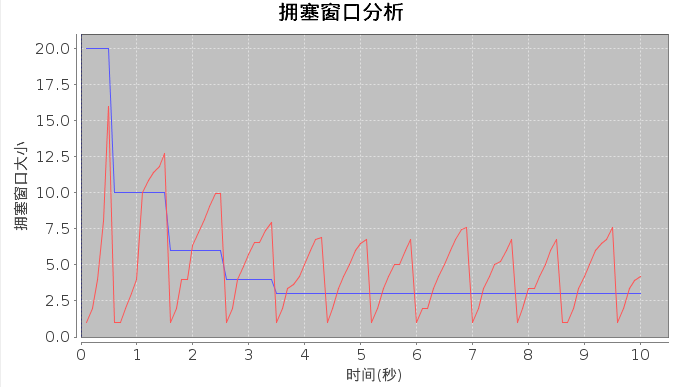


图2.34 RED拥塞窗口分析

公平排队算法FQ拥塞窗口分析如图2.35

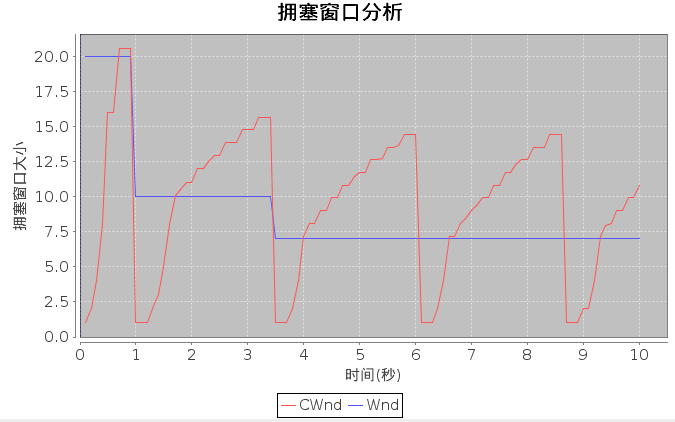


图2.35 FQ拥塞窗口分析

随机公平排队算法SFQ拥塞窗口分析如图2.36

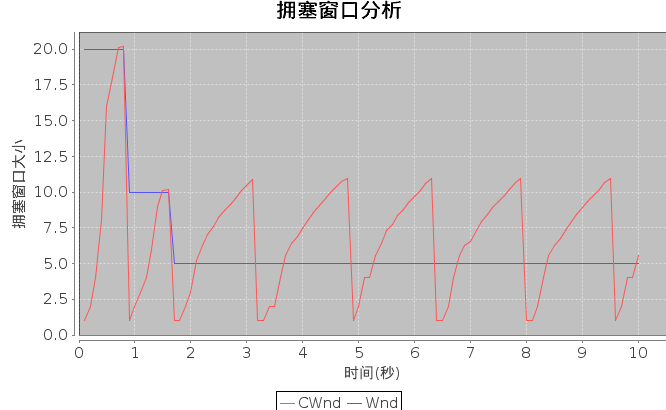


图2.36 SFQ拥塞窗口分析

差额轮循算法DRR拥塞窗口分析如图2.37

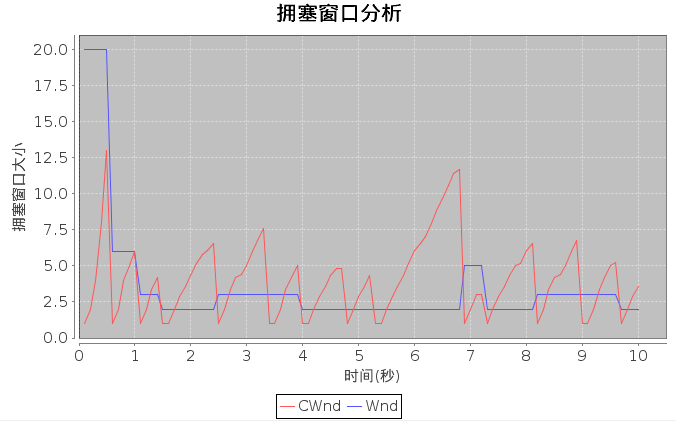


图2.37 DRR拥塞窗口分析

1. 吞吐量

先进先出FIFO吞吐量分析如图2.39所示

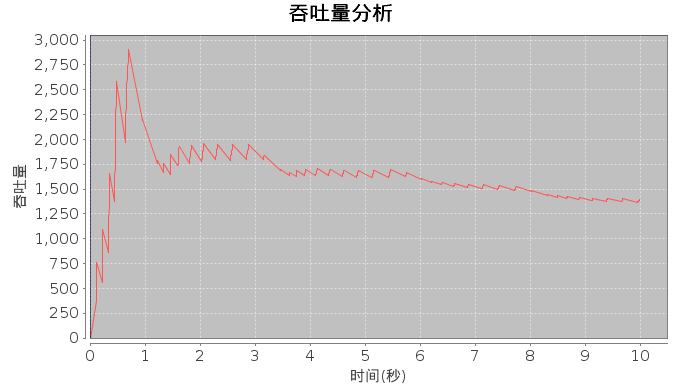


图2.39 FIFO吞吐量分析

随机早期检测算法RED吞吐量分析如图2.40所示

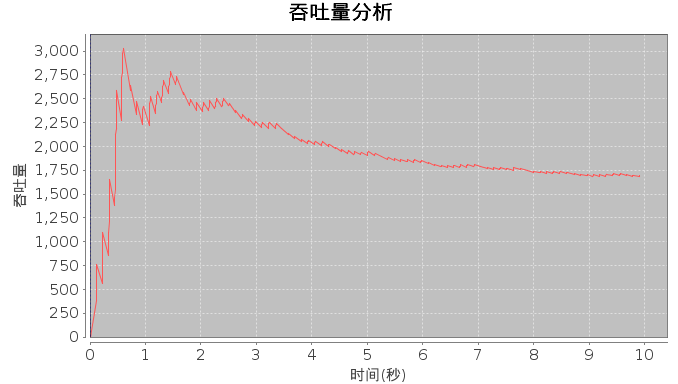


图2.40 RED吞吐量分析

公平排队算法FQ吞吐量分析如图2.41所示

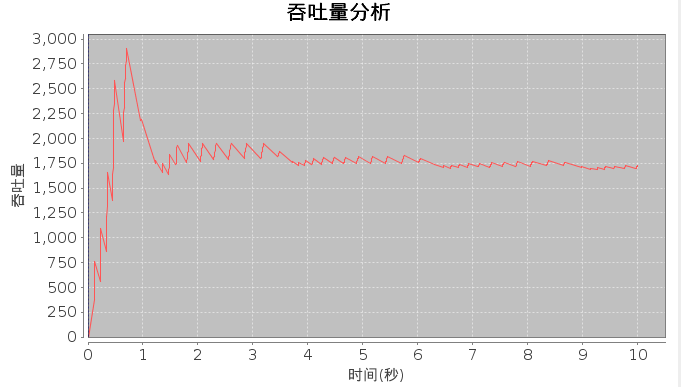
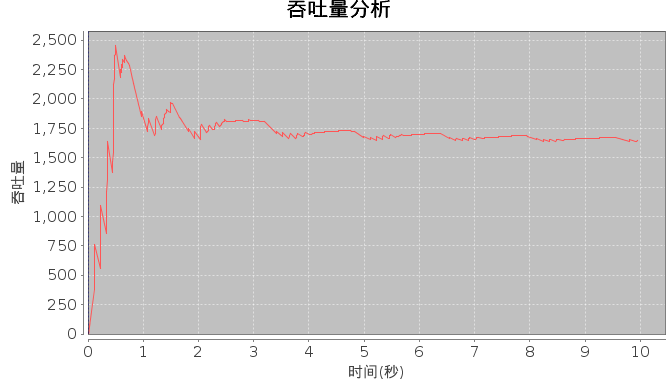


图2.41 FQ吞吐量分析

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

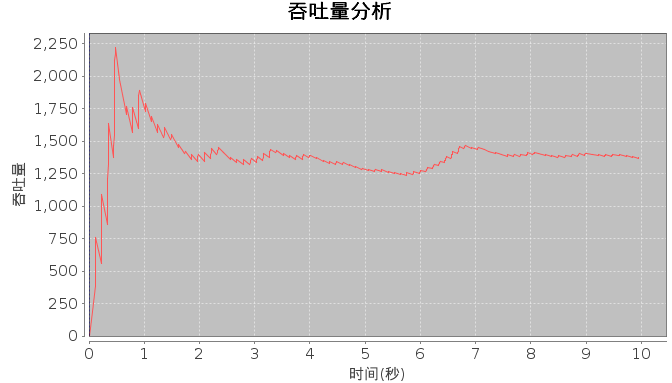


图2.43 DRR吞吐量分析

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。

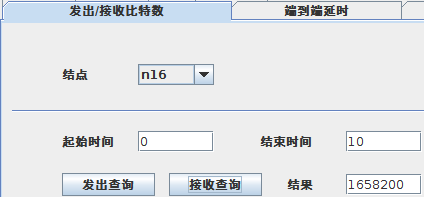
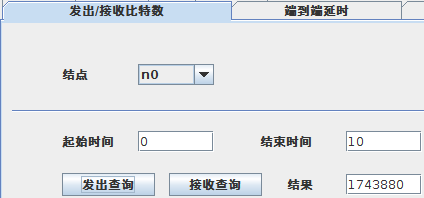


图2.45 FIFO发送端 图2.46 FIFO接收端

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。

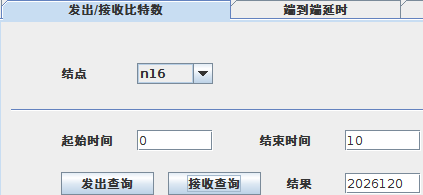
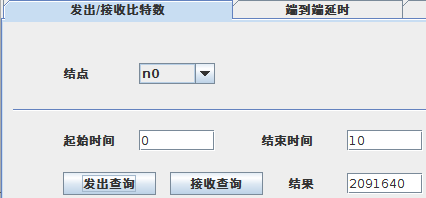


图2.47 RED发送端 图2.48 RED接收端

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。

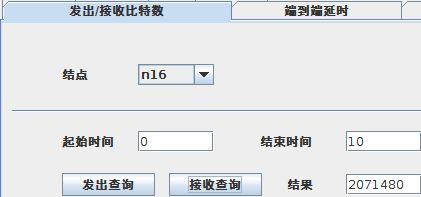
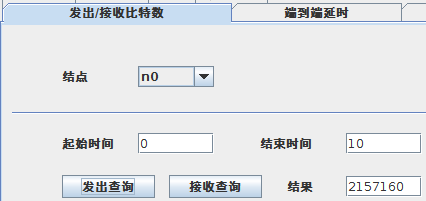


图2.49 FQ发送端 图2.50 FQ接收端

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。

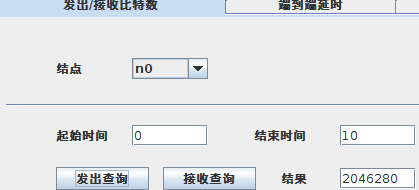
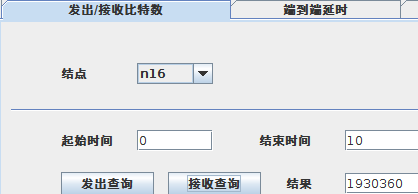
 

图2.51 SFQ发送端 图2.52 SFQ接收端

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。

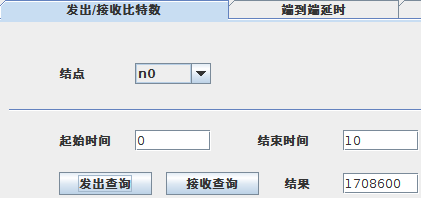
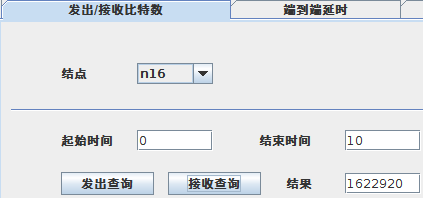
 

图2.53 DDR发送端 图2.54 DDR接收端

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**