



哈尔滨工业大学
Harbin Institute of Technology

计算机网络 课程实验报告

实验名称	可靠数据传输协议（停等与 GBN）					
姓名	陈泊舟		院系	计算机科学与技术学院		
班级	1703110		学号	1173710224		
任课教师	李全龙		指导教师	李全龙		
实验地点	格物 214		实验时间	2019.11.02 上午 12 节		
实验课表现	出勤、表现得分(10)		实验报告 得分(40)		实验总分	
	操作结果得分(50)					
教师评语						

实验目的：本次实验的主要目的

1. 停等协议的设计与实现

理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。

2. GBN 协议的设计与实现

理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

实验内容：概述本次实验的主要内容，包含的实验项

1. 停等协议的设计与实现

1.1 简单实现

基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。

1.2 验证

模拟引入数据包的丢失，验证所设计协议的有效性。

1.3 改进

改进所设计的停等协议，支持双向数据传输。

1.4 应用实现

基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。

2. GBN 协议的设计与实现

2.1 简单实现

基于 UDP 设计一个简单的 GBN 协议，实现单线可靠数据传输（服务器到客户的数据传输）。

2.2 验证

模拟引入数据包的丢失，验证所设计协议的有效性。

2.3 改进

改进所设计的 GBN 协议，支持双向数据传输。

2.4 改进为 SR

将所设计的 GBN 协议改进为 SR 协议。

实验过程：以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交

1. 停等协议的设计与实现

1.1 简单实现

停等协议是最简单但也是最基础的数据链路层协议。很多有关协议的基本概念都可以从这个协议中学习到。停止等待就是每发送完一个分组就停止发送，等待对方的确认。在收到确认后再次发送下一个分组。

实际上其本质就是将 GBN 协议的窗口大小设置为 1 即可。所以实验过程就很简单了，先实现 GBN 协议，然后保留原始参数 N，作为窗口大小的控制变量，最后将 N 设置为 1 即可。

1.2 验证

模拟引入数据包的丢失，验证所设计协议的有效性。验证请见 GBN 协议的实现，方法完全相同。

1.3 改进

改进所设计的停等协议，支持双向数据传输。改进请见 GBN 协议的实现，方法完全相同。

1.4 应用实现

基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。

在实现了 GBN 停等协议的基础之上进行简单的包装即可，在这里只实现了简单的文件传输应用。整个应用分为客户端和服务端，在服务端有三个文件可以用于提供文件传输服务，分

别是 file1.txt, file2.txt 和 file3.txt, 客户端可以请求获取这三个文件的内容, 首先客户向服务器发送请求, 服务器获知该客户请求的是哪一个文件, 然后从发送过来的 packet 中获知传递信息的源 IP 地址, 并向该服务器开始发送文件内容, 此时客户端应不断调用 receive 函数接受数据, 直至接受结束, 传输停止, 退出应用。

2. GBN 协议的设计与实现

2.1 简单实现

基于 UDP 设计一个简单的 GBN 协议, 实现单向可靠数据传输 (服务器到客户的数据传输)。首先来回顾一下 GBN 协议。

分组头部包含 **k-bit** 序列号

窗口尺寸为 **N**, 最多允许 **N** 个分组未确认



ACK(n): 确认到序列号 **n** (包含 **n**) 的分组均已被正确接收

- 可能收到重复 **ACK**

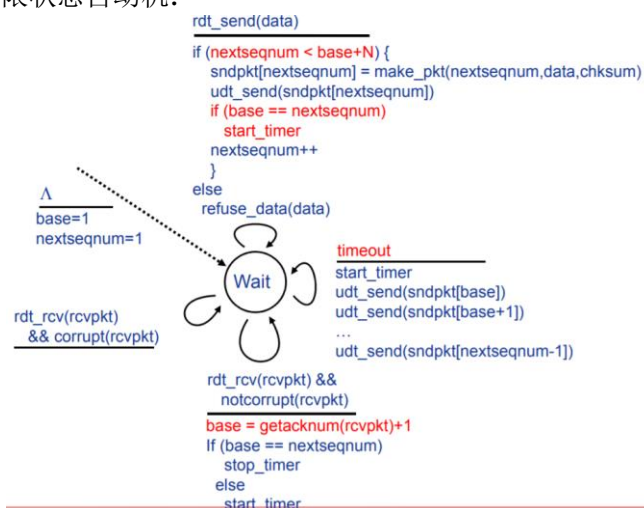
为空中的分组设置 **计时器(timer)**

超时 Timeout(n) 事件: 重传序列号大于等于 **n**, 还未收到 **ACK** 的所有分组

如上图所示, 是 MOOC 中讲的关于 GBN 协议的内容, 首先应该将任务划分为发送方和接收方。GBN 协议的核心是发送方, 一方面要发送数据, 另一方面要接收数据。发送数据指的是, 将数据从本端口发送到目的 IP 上的目的端口号; 接受数据指的是接收方接收到数据之后要发送 ack 确认信息, 发送方要检测这个信息, 并确定窗口是否要向前滑动。对于接收方来说, 需要的工作就简单很多了, 只需要不断的接收数据, 并在成功接收数据之后, 给发送方回复一个 ACK 即可, 但是注意这个 ACK 是要带序号的, 也就是标志了你接受了正确接受了哪个数据。

接下来是程序设计中的细节, 包括定时器的开启和关闭, 窗口的滑动控制等等。

首先是发送方的有限状态自动机:



在这个自动机中, 其实只有一个状态, 那就是等待, 当出现不同的事件, 分别进行不同的处理。首先是初始状态, **base** 和 **nextseqnum** 都设置为 1。第二种情况, 如果收到错误 ACK, 则忽视回复信息, 如果收到正确的 ACK, 则检测相应的序列号 **x**, **x** 代表序列号为 **x** 以及 **x** 之前的所有分组都已经成功接受了, 则窗口要向前滑动, 滑动的方式就是讲 **base** 的值设置为 **x + 1**, 在这次更新之后, 如果 **base** 和当前的 **nextseqnum** 相等的话, 就要讲之前开启的定时器关闭, 以免计时器超时, 重发之前已经确认的分组, 从而造成网络资源的浪费, 如果不相等就要重新开启一个定时器, 以便重传接下来要进行传输的分组。第三种情况, 如果计时器超时, 就重发之前还没有确认的所有分组, 也就是编号从 **base** 开始, 直到 **nextseqnum - 1** 的所有分组。第四种情况, 也

是发送数据的情况，如果窗口内部还有能够发送的分组，那就继续发送，发送结束之后，讲 `nextdeqnum` 的值增加 1，并且在第一次发送的时候，需要注意，如果 `base` 和 `nextseqnum` 相等的话，需要开启一个计时器。

关于数据的接受方，接受到一个数据报判断一下是不是当前自己想要的的数据报，如果是就接受，并且向数据的发送方发送 ACK 确认消息，如果不是，就直接丢弃。

2.2 验证

模拟引入数据包的丢失，验证所设计协议的有效性。关于验证算法是不是真实有效，因为现在网络环境的误码率非常的低，所以在短时间内发送数据一般不会出现丢包的现象。为了进一步展示实验效果，我将序列号为 50 的数据分组进行了标记，并且在发送的过程中直接将其跳过，从而检验有没有真正实现了重传机制。至于实验结果将会在下一个部分进行展示。

2.3 改进

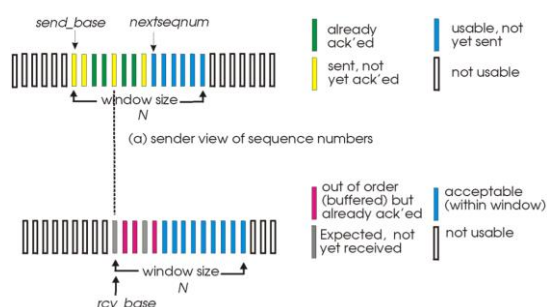
改进所设计的 GBN 协议，支持双向数据传输。简单地讲是新双向数据传输其实很简单，就是每一个端口上都要开两个线程，一个用于接收数据，另一个用于发送数据。但是在实现的时候有一些点还是需要注意的。比如每一个端口只能对应一个 `socket`，如果你在使用的时候，在两个线程中本别开了一个 `socket` 进行数据交换，就会给你抛出异常：

```
Exception in thread "main" java.net.BindException: Address already in use: Cannot bind
    at java.net.DualStackPlainDatagramSocketImpl.socketBind(Native Method)
    at java.net.DualStackPlainDatagramSocketImpl.bind0(DualStackPlainDatagramSocketImpl.java:84)
    at java.net.AbstractPlainDatagramSocketImpl.bind(AbstractPlainDatagramSocketImpl.java:93)
    at java.net.DatagramSocket.bind(DatagramSocket.java:392)
    at java.net.DatagramSocket.<init>(DatagramSocket.java:242)
    at java.net.DatagramSocket.<init>(DatagramSocket.java:299)
    at java.net.DatagramSocket.<init>(DatagramSocket.java:271)
    at gbn.test.main(test.java:42)
```

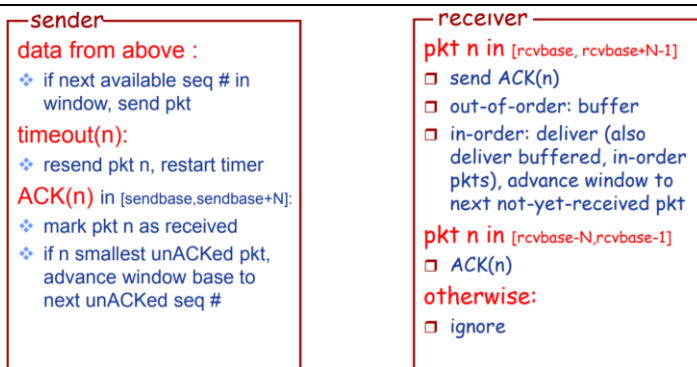
解决的办法就是将 `socket` 设置为静态变量，这样在不同的类中都能调用。关于代码的实现其实在实现 GBN 协议的基本功能时都已经实现了，在这里只是简单地将两部分函数进行拼接即可。只需要将读取和发送数据分别放在两个线程，并且在接受数据的线程中，还要判断读取的数据是确认消息还是作为接收方要接受的数据。

2.4 改进为 SR

将所设计的 GBN 协议改进为 SR 协议。设计 SR 的目的是为了解决重传浪费的问题。在 SR 协议下，每次重传只要将超时的一个数据报进行重传即可。对于提前到达的数据，在接受端设置缓存将其保留。



更加详细的解释结合 MOOC 讲解的内容进行：



发送方与 BGN 协议大致相似，但是并不是累计 ACK，而是只要接收到的 ACK 是在窗口范围之内的，就标记为已经接收，在每一次接收到 ACK 之后，更新发送端窗口。

接收方差别还是比较大的，在 SR 协议中，接受方也有一个窗口，针对所有在窗口内的数据报都是期望的数据报，这些数据报可以不按序到达，只要到达，就将其缓存起来，如果是当前 base 变量对应的数据报，就可以将其从缓存中去除，并写入到文件中。而针对收到的在窗口之外的数据报，与之前的处理一样，直接丢弃。

实验结果：采用演示截图、文字说明等方式，给出本次实验的实验结果

1. 停等协议的设计与实现

1.1 简单实现

基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。

核心代码：

```
private static final int N = 1;
```

将后退 N 步协议的窗口大小设置为 1 即可。

结果展示：

数据发送方在发送数据时处在一个循环中，如果某次循环是发送数据，则输出相应提示信息，并将最新发送的数据报的序列号打印输出；如果在某次循环不是发送信息，就输出提示信息。这些是在实现 GBN 协议时使用的检测方法，关于其他的核心代码展示细节将会在后面的环节进行展示。

现在将窗口大小设置为 1，输出结果进行展示。

```
send message 1//发送第一个数据报
base//此时还没有收到 ACK，窗口不向前滑动，base 值仍然为 1
1
Seqnum//下一个要发送的数据包的序列号是 1
2
Zaiwaimian//在新的循环中没有进行数据传递，输出提示信息
Zaiwaimian//由于中间提示信息过多，在此不再进行展示
... ..
zaiwaimian
receive ack acklack//收到 ACK1，窗口向前滑动，也就是可以发送下一个数据
zaiwaimian
... ..
zaiwaimian
send message 2
base
2
```

```
seqnum
3
Zaiwaimian
... ..
```

1.2 验证

模拟引入数据包的丢失，验证所设计协议的有效性。在 GBN 协议中设计了实验进行验证，实验结果将在 GBN 协议的环节进行展示。

1.3 改进

改进所设计的停等协议，支持双向数据传输。

核心代码：

设置为 P1 与 P2 两个文件进行实现。但是这两个文件中的代码除了端口号以及获取数据输出方向，其他的都是一样的。在这里仅以 P1 为例进行展示代码。

`new recvData1().start();` //处理接受数据的线程



//处理发送数据的核心代码

```
while(true) {
    if(seqnum < base + N) {
        int next = reader.read(buffer);
        if(next == -1) { //如果next是-1，传递最后的数据后跳出循环，略
            return;
        }
        //如果不是文件结尾，就进行文件传输，代码略
        unacked.put(new Integer(seqnum), message) //缓存文件，以防重传
        if(seqnum == base) { //判断计时器开启的条件
            timer = new Timer();
            timer.schedule(new myTimer(timer), timeout);
        }
        seqnum ++; //期待发送的序列号增加1
    }
    System.out.println("out"); //如果不在循环体执行，输出提示信息
}
}
```

结果展示：

两个端口各自从本地读取文件，并将各自接受的文件写入新的文件。实验使用的两个端口号分别是 6666 和 9999。关于双向传输的实质还是数据发送和接受，在这里仅仅展示接受的结果，就是能将各自的数据分别写入到文件中。在程序运行之前这两个文件都是空的，在运行之后分别为 13、14KB，这足以说明实现了双向数据传输。

本地新建的两个文件如下：

 6666recvdata.txt	2019/11/5 0:03	文本文档	14 KB
 9999recvdata.txt	2019/11/5 0:03	文本文档	13 KB

1.4 应用实现

基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。

核心代码：

关于文件传输应用的实现其实就是将数据的单向传输进行包装，一方面将数据从文件中读取出来，另一方面给用户一定的提示信息，并且给出一定的选择余地，仅此而已。


```

System.out.println("Please choose a file that you want to get from
server.");
System.out.println("You can choose one of the three files as
below:file1.txt,file2.txt,file3.txt.");
System.out.println("Please input a number represent the file.\nIf you
input an other number,you will be requested to input another number,");
System.out.println("or you can input -1 to quit the application.");

```

结果展示:

服务器备用数据:

 file1.txt	2019/11/5 13:33	文本文档	237 KB
 file2.txt	2019/11/5 13:34	文本文档	255 KB
 file3.txt	2019/11/5 13:34	文本文档	301 KB

传输结果: 由于网络环境问题, 只将数据传输了一部分进行展示。数据最终存储在如下文件:

 serverRecvdata.txt	2019/11/6 8:28	文本文档	3 KB
--	----------------	------	------

2. GBN 协议的设计与实现

2.1 简单实现

基于 UDP 设计一个简单的 GBN 协议, 实现单向可靠数据传输 (服务器到客户的数据传输)。

核心代码:

具体的实现原理已经在上面进行了简单的介绍, 基于 UDP 的单项可靠数据传输主要依赖的是错误恢复机制, 因此给出如何判断超时, 以及重传数据即可。

首先是关于超时的判断:

//发送数据的主线程中的判断

```

        if(seqnum == base) { //每次发送数据之后, 判断这两个值是否相等, 如果是
则重新开启计时器

```

```

        //begin timer
        timer = new Timer();
        timer.schedule(new myTimer(timer), timeout);
    }

```

//接收数据线程中的重传判断

```

        if(Sender.base == Sender.seqnum) Sender.timer.cancel();
        else {
            Sender.timer = new Timer();
            Sender.timer.schedule(new myTimer(Sender.timer),
Sender.timeout);
        }

```

重传的方法:

```

public static void func() throws IOException {
    DatagramSocket socket = Sender.socket;
    for (int i = base; i < seqnum; i++) {
        //从内存中读取要进行重传的数据, 进行重传, 发送范围如循环所示
        socket.send(packet);
        System.out.println("send message " + new Integer(i).toString());
    }
}

```


结果展示:

与上面的停等协议不同的是窗口大小为 5，在运行代码，查看 mess.txt 文件，结果如下:

```
send message 1
base
1
seqnum
2
zaiwaimian
send message 2
base
1
seqnum
3
zaiwaimian
send message 3
base
1
seqnum
4
zaiwaimian
send message 4
base
1
seqnum
5
zaiwaimian
send message 5
base
1
seqnum
6
Zaiwaimian//在此之前，数据发送端将窗口内所有数据发送出去，但是 base 的值始终没有改变
... ..//接下来的这段事件在等待接收从数据接收端传来的 ACK
Zaiwaimian//很长一段时间之后，开始陆陆续续地收到 ACK，这个时候，窗口开始不断地向前
滑动，数据的发送也变得更加流畅
receive ack ack1ack
receive ack ack2ack
receive ack ack3ack
send message 6
base
4
Seqnum
7
... ..
```


2.2 验证

模拟引入数据包的丢失，验证所设计协议的有效性。

核心代码：

因为现在的网络传输错误率比较低，所以很少会出现错误，关于制造错误的方法在前面已经介绍过了。没有所谓的核心代码，就是将要传输的数据中的某一个在传输的时候直接跳过即可，是一个 if 判断。

结果展示：

还是展示错误条件下的程序有没有正确重传相应的分组，重传分组编号为 50。

其中 49 号分组和 51 号分组收到两次相应的 ACK，这足以说明在 50 号分组丢失的情况下进行了重传，并且根据提示信息可以知道，50 号分组也正确到达了接收端。

```
send message 49
receive ack ack49ack
receive ack ack49ack
base
50
seqnum
50
zaiwaimian
receive ack ack50ack
send message 50
base
51
seqnum
51
zaiwaimian
send message 51
receive ack ack51ack
receive ack ack51ack
```

2.3 改进

改进所设计的 GBN 协议，支持双向数据传输。关于双向传输，停等协议与 GBN 协议的演示结果完全相同，顶多就是文件传输速度上有一些差别，在次就不再进行过多的展示了。

2.4 改进为 SR

将所设计的 GBN 协议改进为 SR 协议。

核心代码：

关于核心代码，SR 协议比 GBN 协议多出来的就是数据接收端的滑动窗口，并且数据分组重传的方式有所改变，以及相应进行改变的数据缓存方式。

接收端的滑动窗口：

```
public static int seqnum = 0; //初始为0
public static File file = new File("sr_recvdata.txt");
public static final int N = 5; //窗口大小为5
public static int base = 1; //初始为1
```

数据缓存：

```
public static HashMap<Integer, Integer> acked = new HashMap<Integer, Integer>(); //记录在窗口内已经ACK的数据报
```

```
public static HashMap<Integer, myTimer> tasks = new HashMap<Integer, myTimer>(); //记录可能需要重传的任务
重传:
//重传直接对应到序列号, 只重传相应的分组
public static void func(int n) throws IOException {
    DatagramSocket socket = Sender.socket;
    DatagramPacket packet = new DatagramPacket(unacked.get(new Integer(n)).getBytes(), unacked.get(new Integer(n)).getBytes().length, InetAddress.getLocalHost(), 9999);
    socket.send(packet);
    System.out.println("send message " + new Integer(n).toString());
}
```

结果展示: 关于结果展示, mess.txt 运行的结果中, 假设 50 号数据报丢失, 只重传 50 号数据报, 很简单, 就不再进行演示。

问题讨论:

主要还是java工具的选择, Java给出的DatagramPacket和DatagramSocket这两个类, 将UDP协议进行了很好的封装和调用, 在使用的时候就提供了很大的便利。

逻辑的组织, 此次实验要比实验一稍微难一些, 而且新增了一些工具的使用, 比如Timer, FileWriter, 并且这些工具在使用的时候, 可能会有一些不熟练, 但是到了哪一步该使用什么样的技术还是很清晰的。针对于文件读取, 直接使用FileReader会出现错误, 传到接收端时候就是乱码了。所以需要使用BufferedReader进行包装后按字符串读取文件, 可以解决问题。

心得体会: 结合实验过程和结果给出实验的体会和收获

实验设置了很多任务, 但其实很多都是基于的扩展, 在实现的时候只要将代码复制过来稍加改动, 就能实现, 还是很简单的, 并且也很好的回顾了课程内容, 此次实验很有意义。