



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-GBN 协议的设计与实现					
姓名	郭子阳		院系	计算机		
班级	1703101		学号	1170300520		
任课教师	刘亚维		指导教师	刘亚维		
实验地点	格物 207		实验时间	2019.11.2		
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) 理解滑动窗口协议的基本原理；
- 2) 掌握 GBN 的工作原理；
- 3) 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：

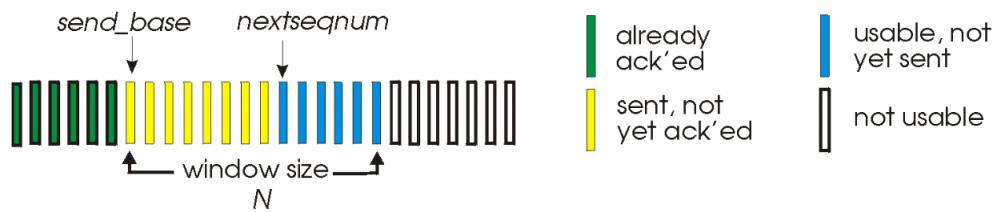
概述本次实验的主要内容，包含的实验项等。

- 1) 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）。
- 2) 模拟引入数据包的丢失，验证所设计协议的有效性。
- 3) 改进所设计的GBN协议，支持双向数据传输；
- 4) 将所设计的GBN协议改进为SR协议。

实验过程：

一、GBN协议

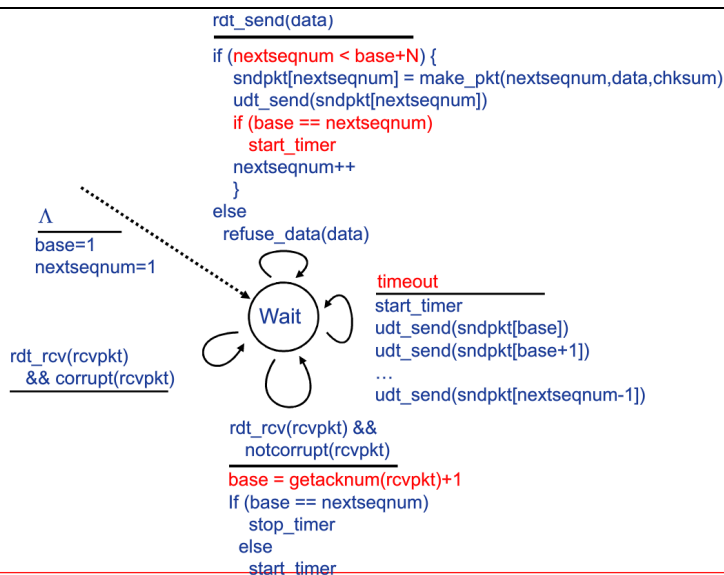
GBN属于传输层的协议，负责接收应用层传来的数据，将应用层的数据报发送到目标IP和端口。滑动窗口的分组的序号以及可以被立即发送的分组的序号，随着发送方对ACK的接收，窗口不断的向前移动，窗口的允许N个分组未确认。



GBN采用累计确认机制，未空中的分组设置计时器，当timeout时间触发后，重传序列号大于等于n，还未发送的分组。组的发送格式设计为：Base(1Byte) + seq(1Byte) + data(max 1024Byte)

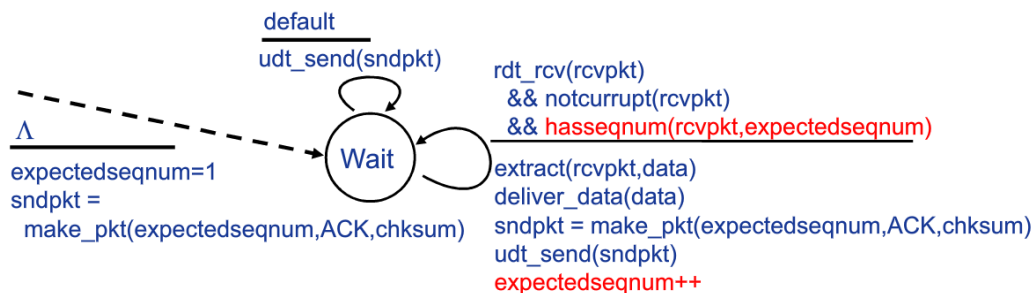
GBN协议数据传输过程如下：从上层应用层获得一个完整的数据报，并将这个数据报进行拆分。如果被发送但未收到确认的分组数目未达到窗口长度,就将窗口剩余的分组全部用来发送新构造好的数据，剩余未发送的数据分组后,开始等待接收从接收方发来的确定信息(ACK)，GBN协议采取了累积确认，当发送方收到接收方对于分组n以及分组n之前的分组全部都收到了。对于已经确认的分组，就将窗口滑动到未确认的分组位置。如果发生丢包，就需要重新发送，直到收到接收方的ACK。timeout事件触发后，GBN协议会将当前所有已发送但未被确认的分组，一旦定时器发现窗口内的第一个分组超时，则窗口内所有分组都要被重传。每个定时器的重置。

发送方扩展FSM如下图所示：



接收方按序接收分组，直接丢弃乱序到达的分组。

接收方扩展FSM如下图所示：



实现过程中，发送方首先定义窗口大小,起始 base 的值，窗口采用链表的数据结构存储，如果窗口内有3满,计时器开始计时,之后进入接收ACK的状态,收到ACK之后,更新滑动窗口的位置,之后如果计时器超时,就将窗口环该过程，直到所有需要传送的数据都已经发送完成,并且窗口中的分组都已经全部确认。

具体实现代码如下：

```

do {
    // 循环将窗口发满
    while(timers.size() < windowSize && sendIndex < content.length && sendSeq < 256) {
        timers.add(0);
        datagramBuffer.add(new ByteArrayOutputStream());
        length = Math.min(content.length - sendIndex, maxLength);

        // 拼接数据帧, 按照 seq + data 的顺序拼接
        ByteArrayOutputStream one = new ByteArrayOutputStream();
        byte[] temp = new byte[1];
        temp[0] = new Long(sendSeq).byteValue();
        one.write(temp, 0, 1);
        one.write(content, sendIndex, length);

        // 向目的主机发送
        DatagramPacket packet = new DatagramPacket(one.toByteArray(), one.size(), host, targetPort);
        datagramSocket.send(packet);

        // 将发送的内容暂存在缓存中
        datagramBuffer.get((int)(sendSeq - base)).write(content, sendIndex, length);
        sendIndex += length;
    }
}
    
```

```
        System.out.println("发送数据包: base " + base + " seq " + sendSeq);
        sendSeq ++;
    }

    // 设置超时时间1000ms
    datagramSocket.setSoTimeout(1000);
    DatagramPacket receivePacket;

    // 循环从目的主机接收ack
    try {
        while(!checkWindow(timers)) {
            byte[] recv = new byte[1500];
            receivePacket = new DatagramPacket(recv, recv.length);
            datagramSocket.receive(receivePacket);
            // 取出ack 的序列号
            int ack = (int)((recv[0] & 0xFF) - base);
            timers.set(ack, -1);
        }
    } catch (SocketTimeoutException e) {
        // 单个socket 超时, 重传所有未确认分组
        for(int i = 0; i < timers.size(); i++) {
            int tempTime = timers.get(i);
            if(tempTime != -1) {
                ByteArrayOutputStream resender = new ByteArrayOutputStream();
                byte[] temp = new byte[1];
                temp[0] = new Long(i + base).byteValue();
                resender.write(temp, 0, 1);
                resender.write(datagramBuffer.get(i).toByteArray(), 0, datagramBuffer.get(i).size());
                DatagramPacket datagramPacket = new DatagramPacket(resender.toByteArray(), resender.size());
                datagramSocket.send(datagramPacket);
                System.err.println("重新发送数据包: base " + base + " seq " + (i + base));
                timers.set(i, 0);
            }
        }
    }

    int i = 0;
    int s = timers.size();
    // 确认并删除所有已经确认过的缓存 (窗口滑动)
    while(i < s) {
        if(timers.get(i) == -1) {
            timers.remove(i);
            datagramBuffer.remove(i);
            base ++;
            s --;
        } else {
            break;
        }
    }

    // 更新发送序号
    if(base >= 256) {
        base -= 256;
        sendSeq -= 256;
    }

} while (sendIndex < content.length || timers.size() != 0);
```

接收方没有缓存，只记录一个seq值，每成功接收一个数据帧，seq++，将seq不是目标值得数据帧直接丢弃。发送方发送一个ACK=seq的确认数据帧，直到发送方没有数据传来。

具体实现代码如下：

```
while (true) {
    count ++;
    try {
        byte[] recv = new byte[1500];
        receivePacket = new DatagramPacket(recv, recv.length, host, targetPort);
        datagramSocket.receive(receivePacket);

        long seq = recv[0] & 0xFF;
        // 若不是期望接收的分组，则丢弃
        if(receiveBase != seq) {
            continue;
        }

        // 模拟丢包
        if(count % loss == 0) {
            continue;
        }

        result.write(recv, 1, receivePacket.getLength() - 1);
        receiveBase ++;

        recv = new byte[1];
        recv[0] = new Long(seq).byteValue();
        receivePacket = new DatagramPacket(recv, recv.length, host, targetPort);
        datagramSocket.send(receivePacket);
        System.out.println("接收到数据包: seq " + seq);

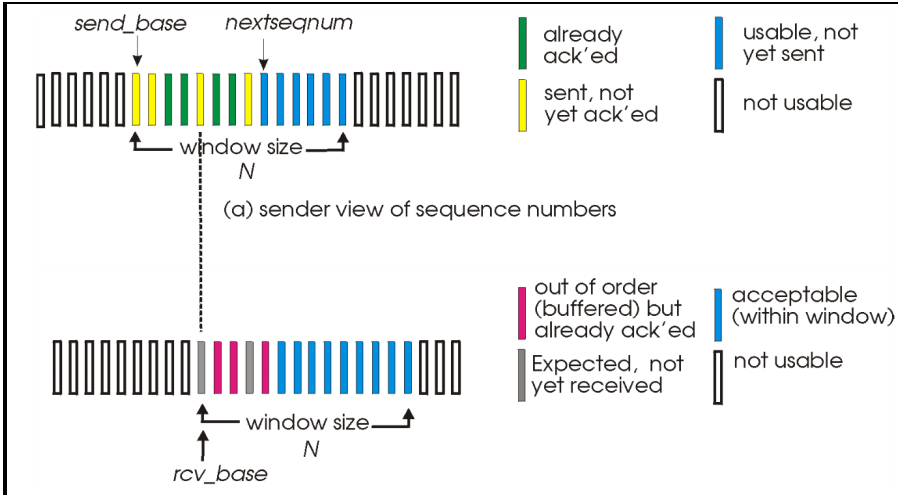
        time = 0;
    } catch (SocketTimeoutException e) {
        time ++;
    }
    // 超出最大接收时间，则接收结束，写出数据
    if(time > receiveMaxTime) {
        break;
    }
}
```

二、SR协议

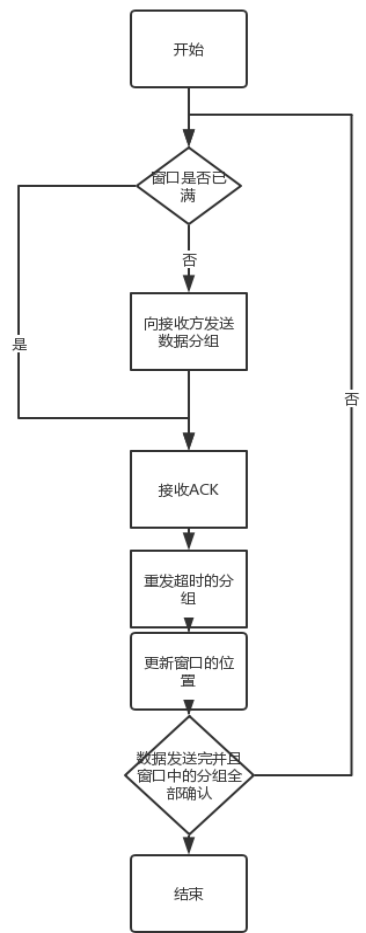
SR协议为每一个已发送但未被确认的分组都需要设置一个定时器，当定时器超时的时候只发送它对应的分组。是窗口内的第一个分组，则窗口需要一直移动到已发送但未确认的分组序号。

接收方,需要设置一个窗口大小的缓存，对乱序到达的数据帧进行缓存，并发送相应序号的ACK，并及时更新发送方。

窗口如下图所示：



发送方在GBN发送方的基础上，对每一个未被确认的分组进行计时，并只对超时的那一个分组进行重传。
实现流程图如下图所示：



接收方增加一个同发送方的对分组的缓存,用于缓存乱序到达的分组,同样使用链表数据结构。接收分组后，对应的位置，然后发送数据分组对应seq的ACK，通知发送方已经成功接收。更新滑动窗口的位置，之后进行下数据传来,超过接收方设定的最大时间,就结束循环,将接收到的数据拼接成一个完整的Byte数组,传给应用层。

接收方的实现流程图如下:



具体实现代码如下:

```
while (true) {
    try {
        byte[] recv = new byte[1500];
        receivePacket = new DatagramPacket(recv, recv.length, host, targetPort);
        datagramSocket.receive(receivePacket);
        // 模拟丢包, 即接收之后不处理, 当成没接收到
        if(count % loss != 0) {
            // 提取出接收到的base 和序列号
            long base = recv[0] & 0xFF;
            long seq = recv[1] & 0xFF;
            if(receiveBase == -1) {
                receiveBase = base;
            }
            // 若发送端base 更新 (即已经确认了几个数据帧)
            if(base != receiveBase) {
```

```

// 从缓存中取出已经确认完成的数据帧拼接
ByteArrayOutputStream temp = getBytes(datagramBuffer, (base - receiveBase) > 0 ?
// 空出缓存
for(int i = 0; i < base - receiveBase; i++) {
    datagramBuffer.remove(0);
    datagramBuffer.add(new ByteArrayOutputStream());
}
result.write(temp.toByteArray(), 0, temp.size());
max -= (base - receiveBase);
receiveBase = base;
}
if(seq - base > max) {
    max = seq - base;
}
// 将接收到的数据帧写入缓存
ByteArrayOutputStream rcvBytes = new ByteArrayOutputStream();
rcvBytes.write(rcv, 2, receivePacket.getLength() - 2);
datagramBuffer.set((int) (seq - base), rcvBytes);
// 返回ACK
rcv = new byte[1];
rcv[0] = new Long(seq).byteValue();
receivePacket = new DatagramPacket(rcv, rcv.length, host, targetPort);
datagramSocket.send(receivePacket);
System.out.println("接收到数据包: base " + base + " seq " + seq);
}
count++;
time = 0;
} catch (SocketTimeoutException e) {
    time++;
}
// 超出最大接收时间, 则接收结束, 写出数据
if(time > receiveMaxTime) {
    ByteArrayOutputStream temp = getBytes(datagramBuffer, max + 1);
    result.write(temp.toByteArray(), 0, temp.size());
    break;
}
}
}

```

三、双向传输

发送方和接收方使用固定的IP和端口之间进行数据传输,直到双方的传输结束。发送方在使用send()函数进行接收。但如果要同时收发,需要同时开一个发送线程和一个接收线程,两个线程独立运行,就可以实现双向传输。

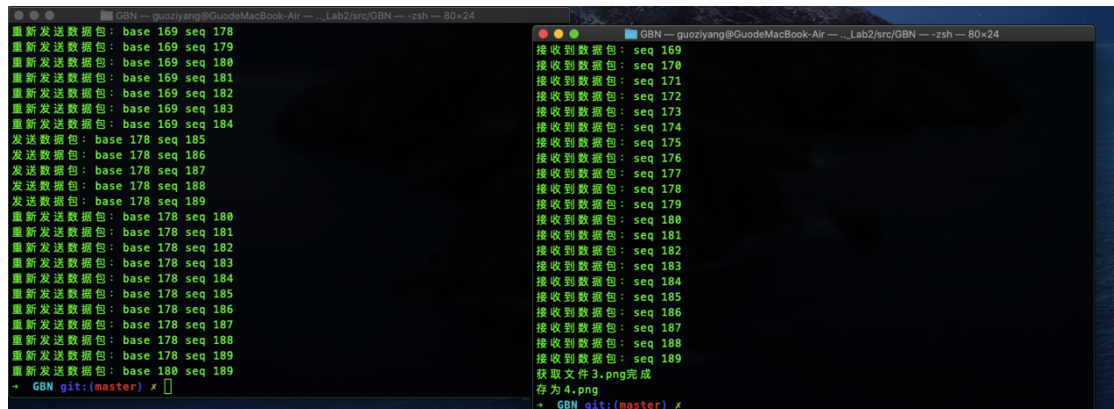
四、模拟丢包

在接收端,固定一个变量loss,每隔loss个分组,接收端将该分组接收但不处理,模拟分组丢失情况。具体实现时,每次收到数据帧就加一,如果count 对一个数取余=0就不发送ACK,模拟这一分组丢失的情况,然后测试发送方的接收情况。

实验结果:

GBN和SR都使用双向数据传输

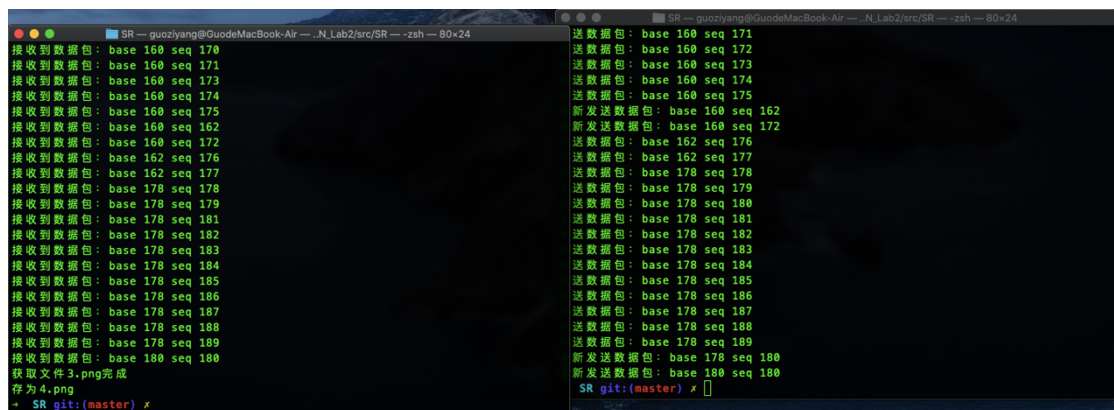
首先客户端先向服务器发送一张图片，服务器接收到后保存，之后再向客户端发一张图片
GBN结果如下：只要有一个分组超时，就会重发所有未确认分组



```
GBN — guoziyang@GuodeMacBook-Air — _Lab2/src/GBN — zsh — 80x24
重新发送数据包: base 169 seq 178
重新发送数据包: base 169 seq 179
重新发送数据包: base 169 seq 180
重新发送数据包: base 169 seq 181
重新发送数据包: base 169 seq 182
重新发送数据包: base 169 seq 183
重新发送数据包: base 169 seq 184
发送数据包: base 178 seq 185
发送数据包: base 178 seq 186
发送数据包: base 178 seq 187
发送数据包: base 178 seq 188
发送数据包: base 178 seq 189
重新发送数据包: base 178 seq 180
重新发送数据包: base 178 seq 181
重新发送数据包: base 178 seq 182
重新发送数据包: base 178 seq 183
重新发送数据包: base 178 seq 184
重新发送数据包: base 178 seq 185
重新发送数据包: base 178 seq 186
重新发送数据包: base 178 seq 187
重新发送数据包: base 178 seq 188
重新发送数据包: base 178 seq 189
重新发送数据包: base 180 seq 189
+ GBN git:(master) x

GBN — guoziyang@GuodeMacBook-Air — _Lab2/src/GBN — zsh — 80x24
接收到数据包: seq 169
接收到数据包: seq 170
接收到数据包: seq 171
接收到数据包: seq 172
接收到数据包: seq 173
接收到数据包: seq 174
接收到数据包: seq 175
接收到数据包: seq 176
接收到数据包: seq 177
接收到数据包: seq 178
接收到数据包: seq 179
接收到数据包: seq 180
接收到数据包: seq 181
接收到数据包: seq 182
接收到数据包: seq 183
接收到数据包: seq 184
接收到数据包: seq 185
接收到数据包: seq 186
接收到数据包: seq 187
接收到数据包: seq 188
接收到数据包: seq 189
获取文件 3.png 完成
保存 4.png
+ GBN git:(master) x
```

SR结果如下：只重发超时的分组



```
SR — guoziyang@GuodeMacBook-Air — _N_Lab2/src/SR — zsh — 80x24
接收到数据包: base 160 seq 170
接收到数据包: base 160 seq 171
接收到数据包: base 160 seq 173
接收到数据包: base 160 seq 174
接收到数据包: base 160 seq 175
接收到数据包: base 160 seq 176
接收到数据包: base 160 seq 177
接收到数据包: base 160 seq 178
接收到数据包: base 160 seq 179
接收到数据包: base 160 seq 180
接收到数据包: base 160 seq 181
接收到数据包: base 160 seq 182
接收到数据包: base 160 seq 183
接收到数据包: base 160 seq 184
接收到数据包: base 160 seq 185
接收到数据包: base 160 seq 186
接收到数据包: base 160 seq 187
接收到数据包: base 160 seq 188
接收到数据包: base 160 seq 189
接收到数据包: base 180 seq 180
获取文件 3.png 完成
保存 4.png
+ SR git:(master) x

SR — guoziyang@GuodeMacBook-Air — _N_Lab2/src/SR — zsh — 80x24
发送数据包: base 160 seq 171
发送数据包: base 160 seq 172
发送数据包: base 160 seq 173
发送数据包: base 160 seq 174
发送数据包: base 160 seq 175
新发送数据包: base 160 seq 162
新发送数据包: base 160 seq 172
发送数据包: base 160 seq 176
发送数据包: base 160 seq 177
发送数据包: base 170 seq 178
发送数据包: base 170 seq 179
发送数据包: base 170 seq 180
发送数据包: base 170 seq 181
发送数据包: base 170 seq 182
发送数据包: base 170 seq 183
发送数据包: base 170 seq 184
发送数据包: base 170 seq 185
发送数据包: base 170 seq 186
发送数据包: base 170 seq 187
发送数据包: base 170 seq 188
发送数据包: base 170 seq 189
新发送数据包: base 178 seq 180
新发送数据包: base 180 seq 180
+ SR git:(master) x
```

心得体会：

深入理解了GBN协议和SR协议的原理和实现流程