**RDT 实验报告**

**学 号 3020244294**

**姓 名 石子跃**

**学 院 智能与计算**

**专 业 网络安全**

**年 级 2020**

**任课教师 仇超**

**2022年 4月 27日**

### 1. 协议需求分析

#### Stop-and-Wait

**需求**: 如何检测丢包，需要增加一种新的协议机制，比特交错协议。

**所适用的场景**: 这是一个功能正确的协议。但是性能方面不佳，特别是在今天的高速网络中更是如此。定义发送方的利用率为：发送方实际发送比特进信道的那部分时间与发送时间之比，分析表明效率极低。而且我们还忽略了底层协议处理时间，路由器的处理和排队时延等问题，这将进一步增加时延，使其性能更加糟糕。发送方这时发送数据包每次只能发送一个，并且必须等待这个数据包的ACK，才能发送下一个。虽然在效率上比较低，带宽利用率明显较低，不过在网络环境较差，或是带宽本身很低的情况下，还是适用的。

#### Go-Back-N

**需求**:停等协议由于其特性导致网络利用率不高，于是需要回退N协议，在不需要确认数据包ACK的情况下仍可进行传输，增加传输效率。

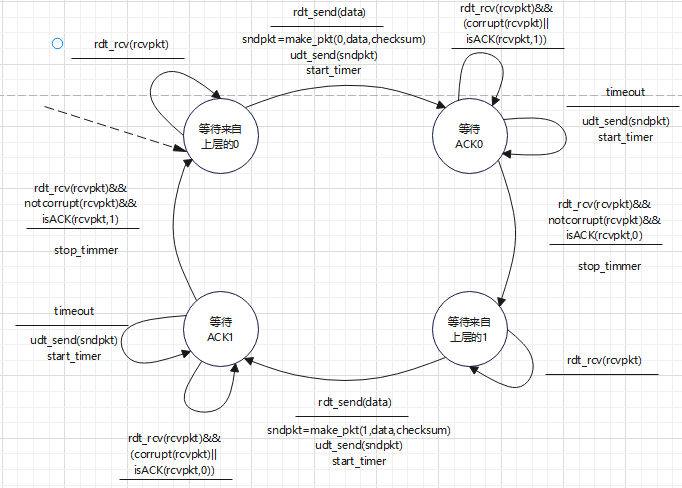
**所适用的场景**:网络质量良好丢包率偏低的网络中，因为单个分组端差错能够引起大量分组端重传，这将导致非常多的资源浪费。

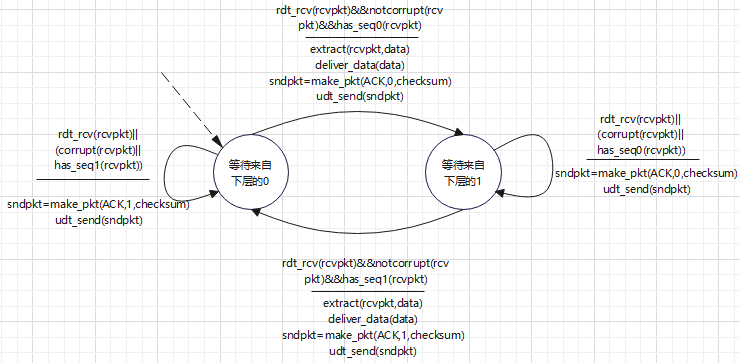
### 2. 协议的设计

#### Stop-and-Wait

**设计原理**：停等协议(stop－and－wait)，这时接受方的窗口和发送方的窗口大小都是1，1个比特就够表示了，所以也叫比特滑动窗口协议。发送方这时自然发送每次只能发送一个，并且必须等待这个数据包的ACK，才能发送下一个。

**FSM 图**：

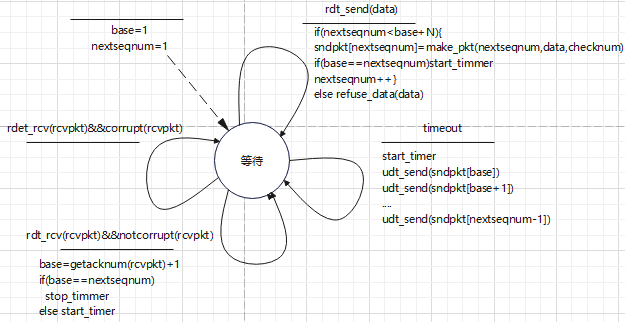


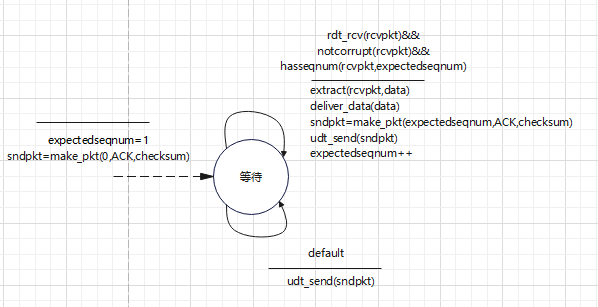


#### Go-Back-N

**设计原理**：返回N协议(GO-Back -N)为了提高效率填满管道,在发送方等待确认时,应当有多个分组正在传送中.也就是说我们需要让多喝分组处于等待确认的状态,以便在发送方等待确认的同时,信道也能保持忙碌状态.返回N协议(GO-Back -N GBN)的关键是发送方能够在收到确认之前发送多个分组,但接收方只能缓存一个分组.发送方为发送出去的分组保留副本,直到来自接收方确认达到.

**FSM 图**：





### 3. 协议的实现

#### Stop-and-Wait

**主要数据结构**：

struct Sender

{

  int state;//当前01比特顺序

  int acknum;//

  float time\_out;//rtt最长等待时间

  struct pkt last\_pkt;//上一个传输的数据包

} A;

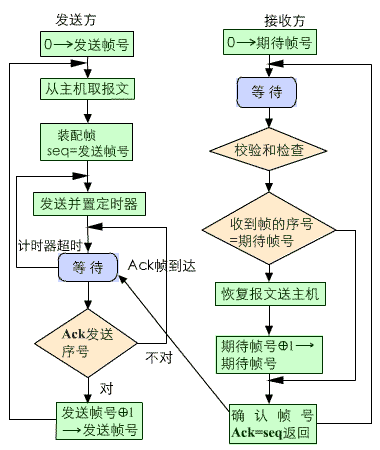
struct Reciver

{

  int seqnum;//01交错

} B;

**流程图**：



#### Go-Back-N

**主要数据结构**：

struct Sender

{

    int base;

    int nextseqnum;

    int N;

    float time\_out;

    struct pkt pkt\_buf[MAXBUFF];

} A;

struct Receiver

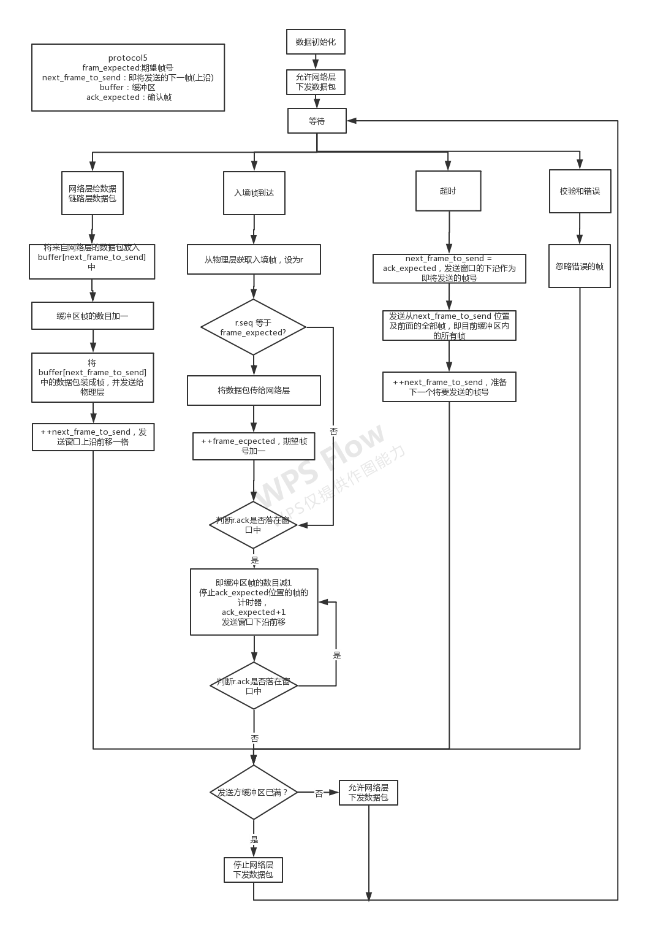
{

    int expectedseqnum;

    struct pkt last\_pkt;

} B;

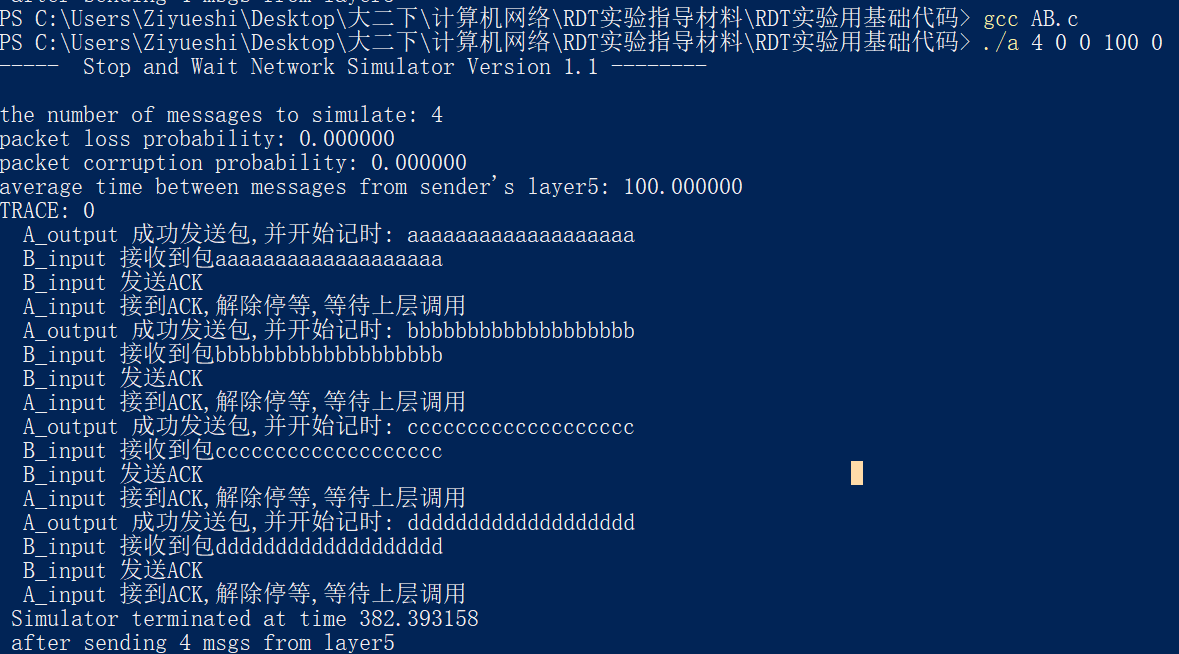
**流程图**：



### 4. 协议功能测试

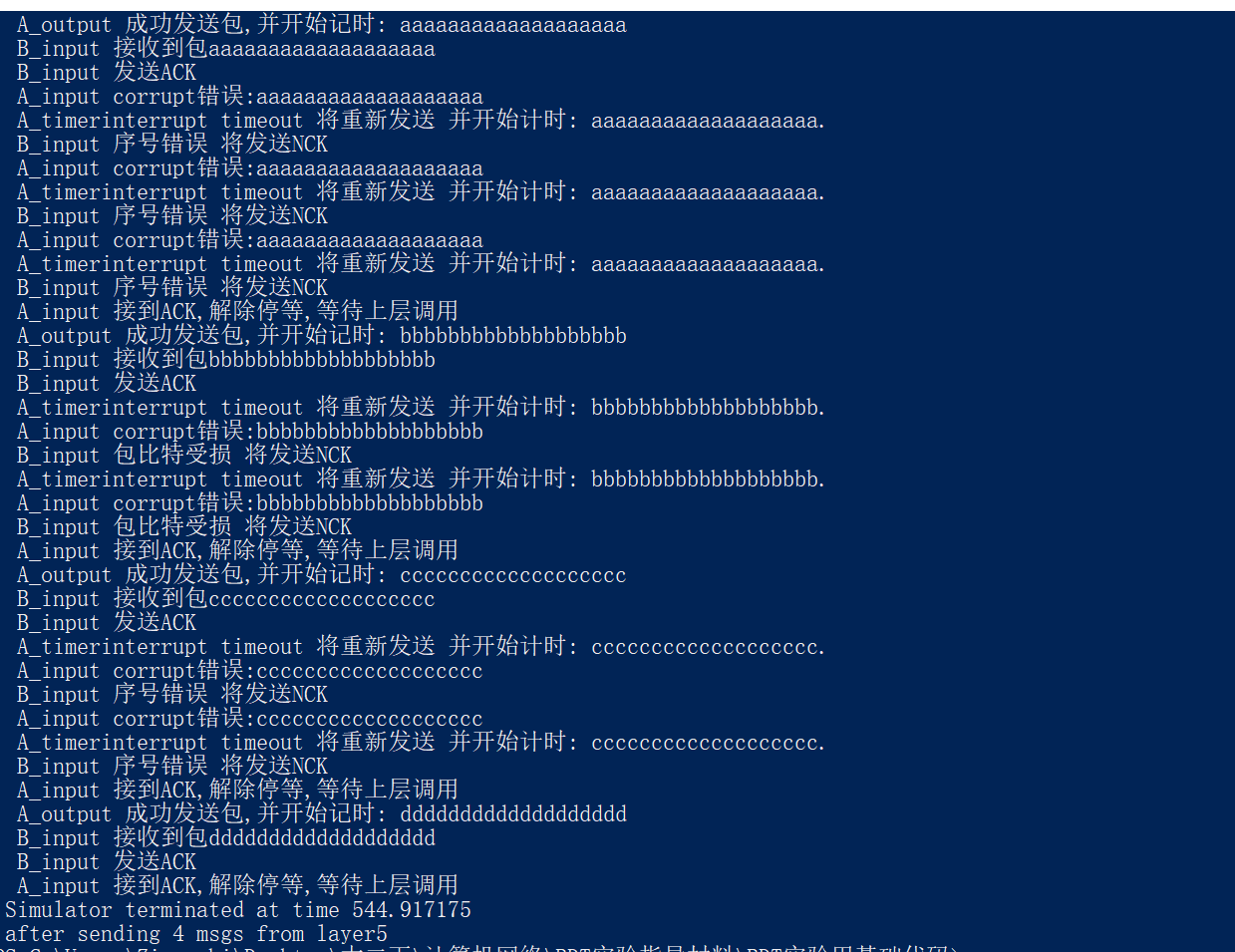
#### Stop-and-Wait

1) 无 error 无 loss；



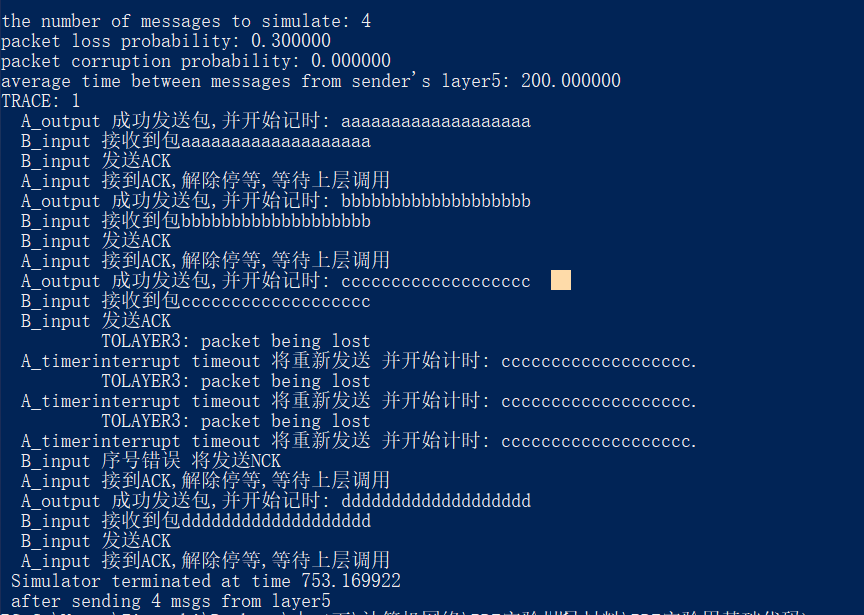
符合FSM，当A发送一个数据包后要等待Binput发送的ACK，由于此测试五error，无loss，所有顺序正确。

2) 数据包、ACK 包出现 error；



发送a时，B返回的ACK包出现错误，A会重新发送a,结果B检查到比特交错错误,会发送nck，这时又有A的时间超时又发送了一次a，最后接到了ack.于是A端口可以继续接受顶层的调用，发送b，可能是由于设置的超时时间太短总容易触发超时，后面的情况如上所述。

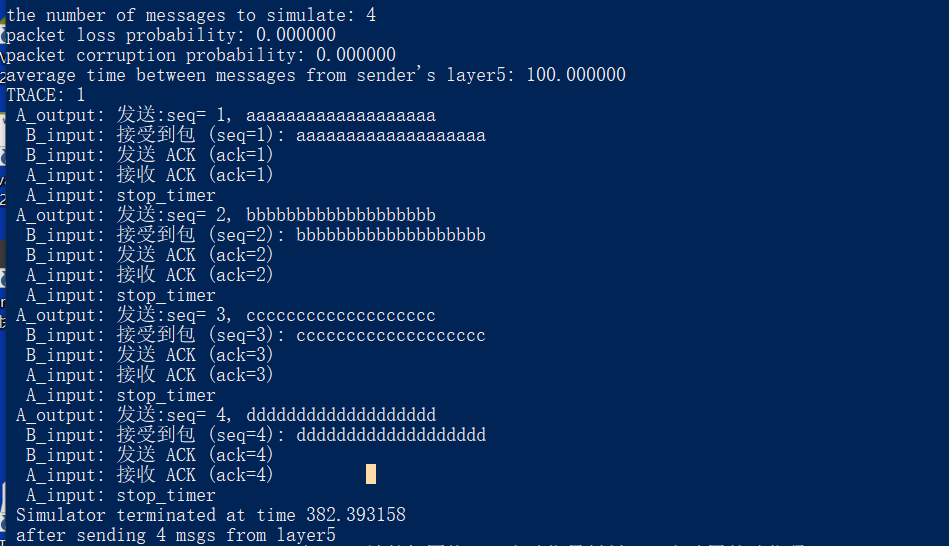
3) 数据包、ACK 包有 loss；



前两个和第四个包很正常，但是c居然lost了三次，每一次都在超时后重新发送了c,但是此时B已接受c，所以B返回NCK，这时A收到后会解除限制接受上层调用。

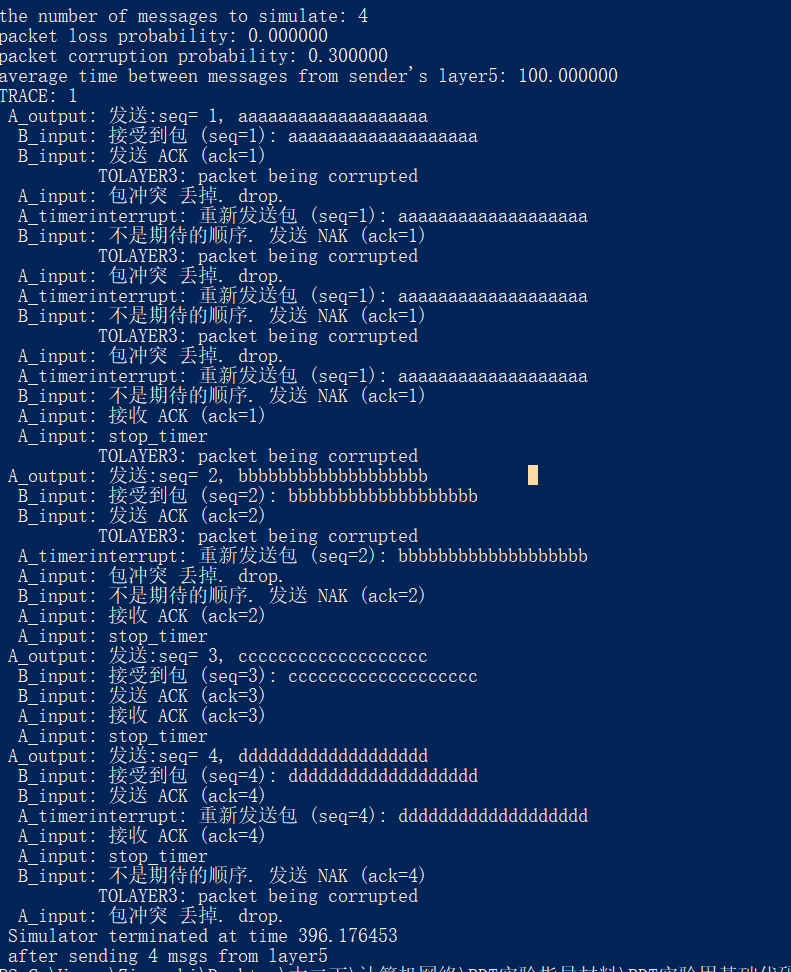
#### Go-Back-N

1) 无 error 无 loss；

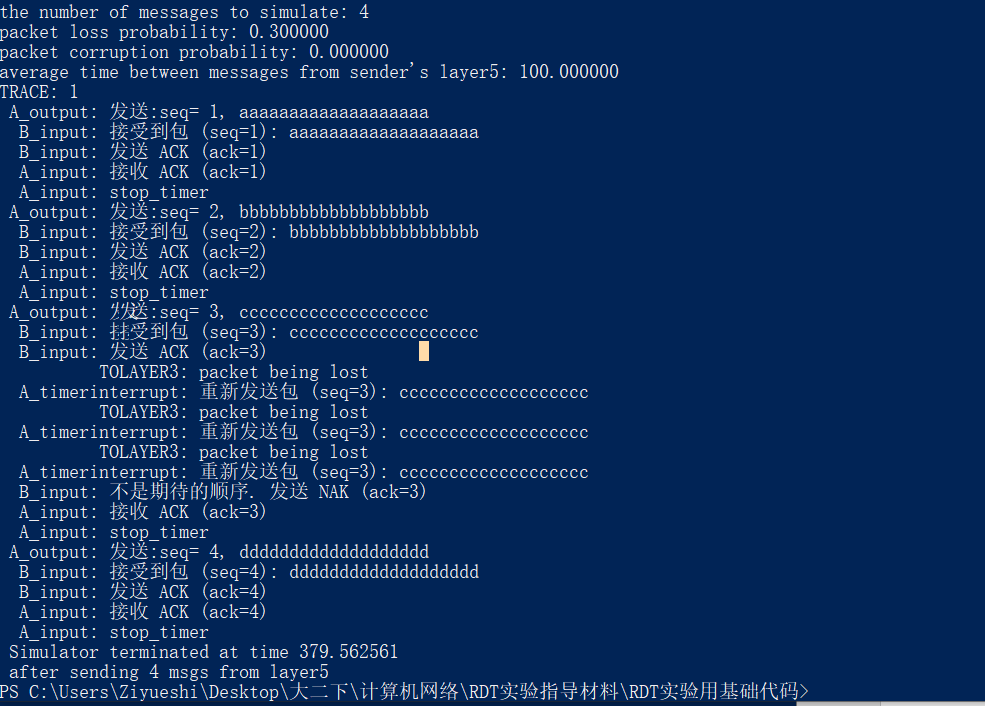


发现顺序很正确，证明程序和预期一致。

2) 数据包、ACK 包出现 error；

  
首先B的ACK包发生错误，导致A丢掉此包。等待，结果超时，A将重新发送所有未ACK的包，这时B发送了他最近接受的一个包，结果又包错误，重复上述步骤，直到A接受ACK。开始发送b，结果又是返回ACK时错误，重复上述。结果发送d的时候都正确，却引发了超时，只得重新发送d。

3) 数据包、ACK 包有 loss；



1. b的传输正确无loss，当B返回c的ACK时发生loss，连续三次，触发三次A的超时。最后B发送的NCK到A结束。继续D。

### 5. 协议性能测试

RTT=15, msg\_num=20, avg\_time\_btw\_msg=30

#### 场景二、 参数设置： RTT=5, msg\_num=20, avg\_time\_btw\_msg=20

当出错率和丢包率无论哪一个上升时都会对吞吐率产生影响，但是在情景1中，出错率和丢包率的变化对于无论是GBN还是SAW影响挺小的。最后一个两者相差巨大，个人分析觉得是因为RTT设定时间太短于是GBN总是会超时，导致了重复传输的信息暴增，导致了这个现象。当出错率为0，丢包率为百分之五，出错率为0.1丢包率为百分之七，GBN效率突然暴增。在设定的网络延迟小，且每个包到达时间间隔短时可能SAW效率更高。

### 6. 总结

经过本次实验，我了解了运输层的SAW和GBN协议。让我熟练掌握了其实现协议原理，c代码编写等相关知识。深入理解了可靠数据传输的基本原理。掌握 Stop-and-Wait 和 Go-Back-N 两种协议的设计方法，能够使 用 FSM 图进行设计。掌握 Stop-and-Wait 和 Go-Back-N 两种协议的具体实现方法。也暴露了许多问题，在调试过程中不可避免的多走了许多弯路。