**《计算机网络》****课程设计报告**

**及第一周进度报告**



**可靠数据传输机制的仿真实现**

**学 号 刘原驰 潘辰誉**

**姓 名 3021244213 3021244228**

**学 院 智能与计算学部**

**专 业 计算机科学与技术**

**年 级 2021级**

**任课教师 石高涛**

**2023年 3 月 29 日**

# 一、报告摘要

本次实验为在仿真环境条件下，实现RDT可靠传输协议中的Stop-and-Wait和Go-Back-N协议。具体即是在实验所给的实验框架的条件下，结合FSM图进行设计，然后使用C语言补充编写传输层可靠数据传输代码，实现Stop-and-Wait和Go-Back-N两种协议。对于本周工作具体集中于对Stop-and-Wait协议的理解、设计、实现与测试。最终我们设计实现了针对于传输过程中的等待、有序、丢包、错包以及具体到数据包、ACK包的丢包、错包的各种模拟情况的对应处理的实现。根据最终对于此的trace跟踪分析的结果，我们发现对于Stop-and-Wait协议中的各种模拟情况与处理的结果均符合设计要求与预期情况，这表明我们本次对于Stop-and-Wait协议的实现已经成功完成。

# 任务要求

**stop-and-wait**

对于stop-and-wait协议而言，其基本思路是发送方每次发送一个分组后就停止发送新的分组，而是等待收到接受方收到分组的正确确认后，才开始发送下一个新的分组，而在这个过程中可能会出现乱序、丢包、错包的情况，对于此协议的模拟实现即是通过对于这几种情况的单独与综合问题的解决来实现的。对此的实现可以划分为两个部分来进行阐述，一是对于出错的检验，及如何检验发生了丢包和错包，二是对于已出现的问题，应当如何进行相应的解决。

在此过程中具体需要实现的模拟处理如下：

1. 没有丢包和错包的情况下，接收方返回ACK，发送方收到正确ACK，然后再未发完既定分组数的情况下继续发送分组
2. 在数据包丢包的情况下，发送方重新发送个分组
3. 在接收方返回的ACK包丢包的情况下，发送方重新发送这个分组
4. 在数据包坏包的情况下，发送方收到接受方反馈、然后重新发送这个分组。
5. 在ACK包坏包的情况下，发送方接收到错误信息，即非正确反馈，重新发宋这个分组。
6. 只有在数据包不丢包、不坏包、且序列号无误的情况下，接收方才能返回相应的ACK。

**Go-Back-N**

对于stop-and -wait协议而言，其基本思路是发送方同时发送多个分组，在发送方等待接收方反馈的确认时，仍有多个分组（各种处理状态总数量不超过窗体大小）正在传送，即发送方在收到当前分组的正确确认前能继续发送接下来的多个分组，总体上会形成流水线式的传输。但是对于接收方而言，其只能按序一个个接收，并返回确认。对于接收方后，只有按序收到正确确认后，才可向后移动。

对于具体实现其除需要满足stop-and-end协议中的中间四条外，还需要模拟实现的内容如下：

1. 对于本过程中的发送方而言，需要设定相应的缓冲区来存储已发送但尚未得到确认消息的分组和等待被发送的分组。
2. 发送方和接收方都需要设立窗口，其中发送窗口大小代表最多允许的尚未被确认的分组的数量。接收窗口大小为1，功能上只能按次序接收分组，并返回相应的ack。
3. 在发送收到当前位置的ACK后，窗口右移，可以开始发送新的分组。
4. 对于定时器而言，只用到一个用于保存发送方对最古老的尚未得到确认的分组的定时器，超时时，重传从其开始的的所有尚未确认的分组。

# 三、协议设计

## **3.1 总体设计**

对于本实验中初始提供的实验框架而言，我们可以总体上对其进行分析。

首先对本实验中的函数进行分析。本实验中提供的初始函数抓哟可以划分为计时开始、计时终止函数和传递函数（第四层传至第三次层、第三层传至第五层）两大类。

starttimer(calling\_entity,increment)函数是计时启动函数，其作用是根据传入的参数来启动计时器，其中参数calling\_entity的取值为 0（启动A端定时器）或1（启动B端定时器），increment为在定时器中断之前将经过的时间量，对于其而言在实验中用于开启计时以便于判断等待是否超时、是否需要重发分组，其中保存的参数increasement用于保存与记录最终的时间。

stoptimer(calling\_entity)函数是计时终止函数，其与starttimer()相对应，其作用是用于终止计时器，结束计时，其中参数calling\_entity取值为 0（停止A端定时器）或1（停止B端定时器）。

对于传递函数而言，是函数tolayer5(calling\_entity,message)和 函数tolayer3(calling\_entity,packet)，其中参数calling\_entity取值为 0时均表示A端发送，取值为1时均表示B端发送，这两个函数作用在于传输经过我们填写的传输层代码处理后的数据分组或消息。

然后我们对于实验整体框架进行分析，在主函数mian()中，会先调用init（）函数，对初始仿真环境进行初始化，读入键入的相关仿真参数，同时检验实验环境是否符合实验条件，若符合条件则会根据所输入的参数调用相关的一系列函数生成相应的仿真数据，然后会调用A\_init()和B\_init()函数，完成对于A端和B端数据的初始化，然后通过循环开始对于仿真模拟已生成的数据的传输。对于具体传输过程而言，主体部分的检验是在未达到仿真初始定义的发包数时，根据对于初始定义的参数evtype、eventity两个参数，即来自第三层、第五层还是超时和来自A端还是B端两个参数来对应的调用相关的input、output函数和超时处理函数。具体情况是当消息来自第五层时，此时将会向第四层发送消息msg，此时会生成相应的数据，然后判断所处理的位置，若此时是A端数据，则会调用A\_output(),若是B端数据，则会调用B\_output()；当消息来自第三层时，此时将会传输发送分组pkt，然后判断所处理的位置，若此时是A端数据，则会调用A\_input()向B端发送分组,若是B端数据，则会调用B\_input()向A端发送分组；若此时消息类型为超时信号，则会进行判断出现超时的位置，若为A端，则会调用A\_timerinterrupt()函数，处理超时问题；若为B端，则会调用B\_timerinterrupt()函数，处理B端超时问题。

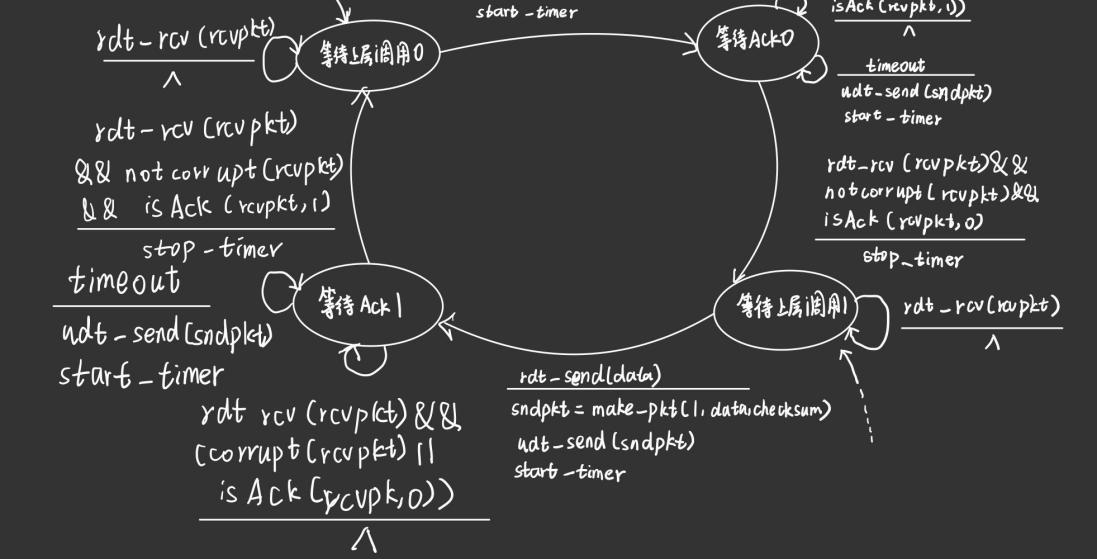
## **3.2 Stop-and-Wait协议的设计**

**3.2.1 Stop-and-wait协议的设计原理**

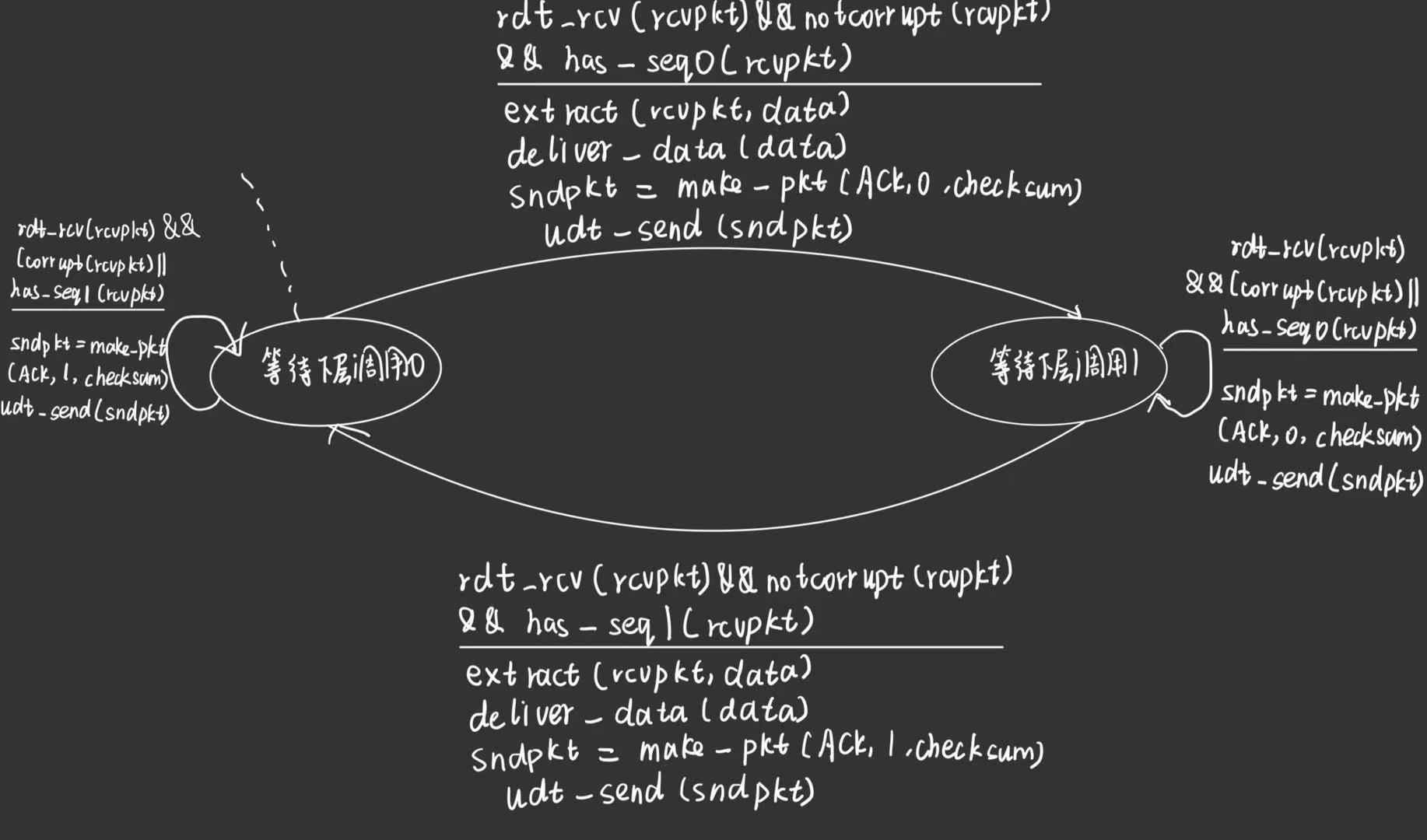
对于stop-and-wait协议在本实验模拟的要求下而言，其设计原理是发送方将数据分成固定大小的数据块，并将每个数据块封装成一个分组。发送方每次只会发送一个分组，然后等待接收方的确认的ACK。接收方在收到分组后，会检查分组是否有错误。如果接收方发现所收到的分组有错误，则接收方会丢弃该分组，并发送一个不正确的ACK给发送方，然后是发送方重新发送该分组。如果所接收到的分组没有错误，则接收方会发送一个正确的ACK给发送方，表示已成功且正确接收该分组。发送方在收到正确的ACK后，就会开始发送下一个分组。如果发送方在一定时间内没有收到ACK，则会认为在上述某一过程中出现了分组丢失的情况，然后重新发送该分组。

## **3.2.2 Stop-and-wait协议的FSM图**

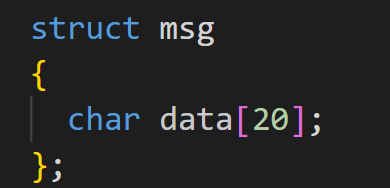
发送端：



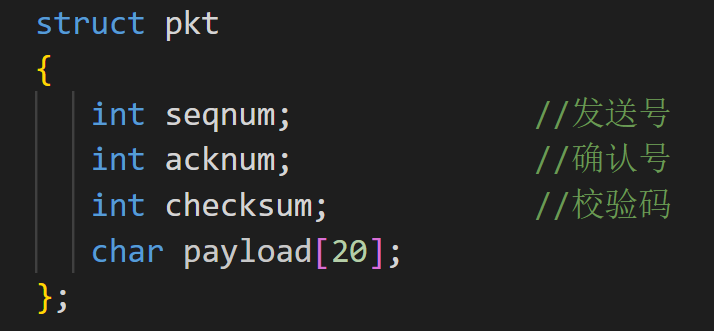
接收端：



**3.2.3 Stop-and-wait协议的主要数据结构**



结构体msg是传输的消息的本体，是从第五层向第四层传输的消息。



结构体pkt是分组传输的信息，其包含发送序号seqnum、确认号acknum、校验和chenum和信息数组payload的几个属性。其实从第四层向第三层传输的信息。

**3.2.4 Stop-and-wait协议的协议规则**

1. 发送方按照输入脚本数据的需求，运行初始化函数，生成对应的分组，并根据相关函数生成传输层可以发送到第三层（数据链路层)的分组，分组有调用相关函数得到的序号、校验和、ACK值以及信息属性。
2. 发送方通过第三层（数据链路层）发送分组到接收方。
3. 接收方接收到分组后，会对所收分组的校验和和序号属性进行检验，并发送一个ACK回到发送方（无论是否成功均会发送，即只要收到分组就会发送，但检验成功发送的是正确的ACK，失败发送的是不正确的）
4. 发送方接收到来自接收方发送的正确的ACK值后，发送下一分组并重复步骤2-4。
5. 如果发送方在一定时间内未收到ACK，即计时器超时，其将重新发送当前的分组

## **3.3 Go-Back-N协议的设计**

说明Go-Back-N协议的主要数据结构和协议规则。用FSM图表示发送端和接收端的主要工作流程。

说明发送端缓冲区的管理，滑动窗口的设计方法。

# 协议实现

# **4.1 Stop-and-Wait协议的实现**wps

检

验

符

合

不超时

超时

正确

不正确

尚

未

到

达

指

定

数

量

具体下列实现阐述，根据上述的流程图展开。

当发送端开始发送信息时，其会相应的调用A\_output函数，然后A\_output函数会调用函数generatePkg(),根据从第五层传向第四层的消息msg生成一个分组，同时计算其对应的校验和并将ACK值定义未-1（undefine状态），然后将A发送的数据拷贝到A的缓冲区进行备份，在备份完成后调用 tolayer3()函数，向B发送分组同时启动计时器，开始计时，进而实现对于传输层分组传输丢包情况的模拟与处理。这一过程即是传输层对于其从应用层收到的数据从接受处理到发送的过程的模拟。

在本过程中，所使用的generatePkg()所生成的pkg的属性中，seqnum为从0开始，根据发包顺序依次递增，acknum初始默认值为-1（undefine状态），若接收方B正确接收，则会发送ACK包，将acknum的值改为1（OK状态）；若接受失败，则会改为0（error状态）。对于校验和checksum而言，其有checksum=acknum +seqnum+（int）payload中的每个字符。

当对应的数据包传输到接收端时，接收端B会调用B\_input函数，然后其会对收到的分组的校验和和序号进行检验以确定所受分组是否正确，如果检验均无误，其会先调用tolayer5将数据先从传输层发给其所对应的应用层，然后调用函数generatePkg()，生成一个ACK值为OK的分组传给A；如果检验存在失败，则表明该过程中出现了坏包的情况，其会生成错误预警，但在本实验中由于不产生NAK，因此不做处理，对于此情况，处理为等待A端计时器超时重发分组来解决。

接着对于此过程而言，然后是A端接受ACK包，对应此情况是发送方A调用A\_input()函数，对收到的从A端返回的分组的ACK进行检验，如果收到的ACK包为OK（1），则会输出正确并停止计时，然后对seqnum进行加一操作，以结束此次发送分组的过程，然后使A停止等待，可以开始发送新的分组；而对于其他情况则不做其它处理，只根据计时器来进行，如果计时器超时，则会停止计时并重发存储在缓冲区中的数据包之后开启新的计时器。

由于本实验中要求不使用NAK，因此在此过程中，只有两种情况会检验ACK，一种是完全不出现错误时，发送方A端会收到（正确的）ACK，这种情况下处理是停止计时，然后若未达预定数目，则继续进行发包，另一种是ACK包是坏包，在此情况下，其检验失败，会跳过计时函数并结束计时，然后调用A\_input函数重发数据包。而对于其他情况而言，均通过是否超时来进行判断，若超时则通过重新发送当前分组来进行尝试性解决。

## **4.2 Go-Back-N协议的实现**

只实现单向传输即可。

# 五、实验结果及分析

## **5.1 Stop-and-Wait协议的功能测试与结果分析**

**5.1.1 测试环境**

OS:ubuntu 22.04 LTS

编译器：gcc 11.3.0

**5.1.2 功能测试**

(1)**无error无loss的情况下，数据分组均能够正确发送和接收**

在初始输入条件为：100 0 0 1000 2

我们可以看到（部分结果，包含一个具体结果和最终计时输出结果）：



成功实现对于100个包的发送和接收过程，此过程中所出现的所有情况均为上述结果中的类似结果。

此过程中的流程均如上图所示为

A\_output:发送分组

B\_input:接受分组

B\_input：校验和检验成功，发送ACK

A\_input：A收到B端ACK，可以继续发送新分组或结束

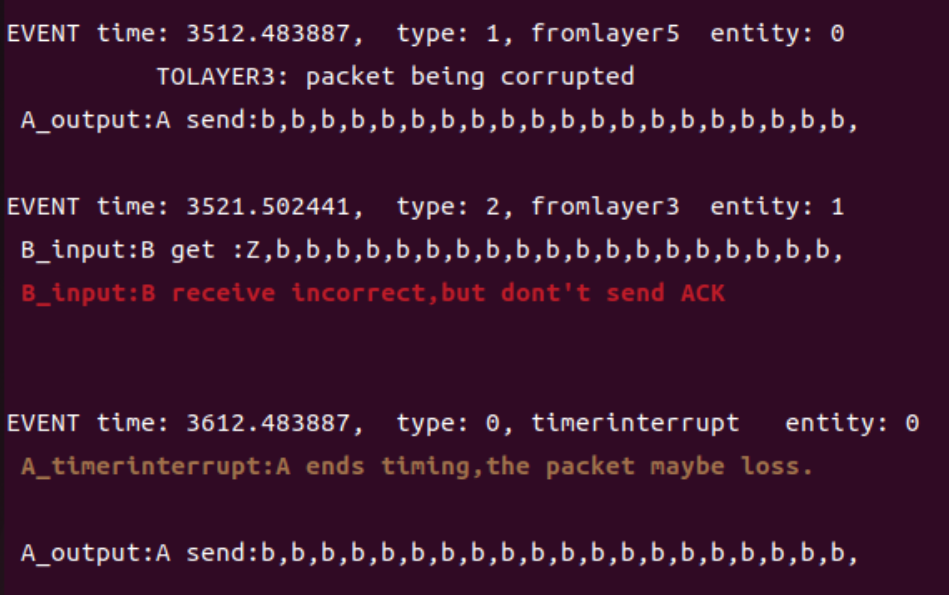
上述处理是符合初始设计和预期的，此外，最终输出结果表示成功发送100个包，模拟无误。

1. **没有 loss，只有 error 的情况，该情况只设置坏包概率，此条件下会出现仿真模拟会模拟出两种如下情况**

在初始输入条件为：100 0 0.2 1000 2

1. **数据包出现 error**

我们可以看到：



该情况模拟的是A端向B端发送的数据分组是坏包，可以看到我们这里对这个数据包进行的处理是：在B端检测所收分组的checknum不符合时，其不做任何处理，使得A段计时函数超时，进而A端重发数据包，尝试解决问题，这是符合初始设计和预期的。

该过程的流程是

A\_output:发送分组

数据分组坏包

B\_input：校验和检验失败，不做处理

ACK包坏包

A\_timeinterrupt:超时，停止计时，调用A\_output函数来重发数据分组

如果下一次发送没有出现任何问题，则

A\_output:重新发送分组

B\_input:接受分组

B\_input：校验和检验成功，发送ACK

A\_input：A收到B端ACK，可以继续发送新分组或结束

(b)**ACK 包出现 error**

我们可以看到：



该情况模拟的是B端向A端反馈发送的ACK包是坏包，可以看到我们这里对这个数据包进行的处理是：在A端检测ACK包不符合时，A端自动重发了数据包，这是符合初始设计和预期的。

该过程的流程是

A\_output:发送分组

B\_input:接受分组

B\_input：校验和检验成功，发送ACK

ACK包坏包

A\_input：A收到B端ACK，校验失败，重发分组

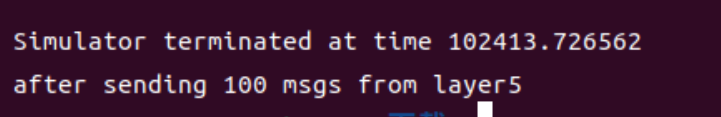
如果下一次发送没有出现任何问题，则

A\_output:重新发送分组

B\_input:接受分组

B\_input：校验和检验成功，发送ACK

A\_input：A收到B端ACK，可以继续发送新分组或结束



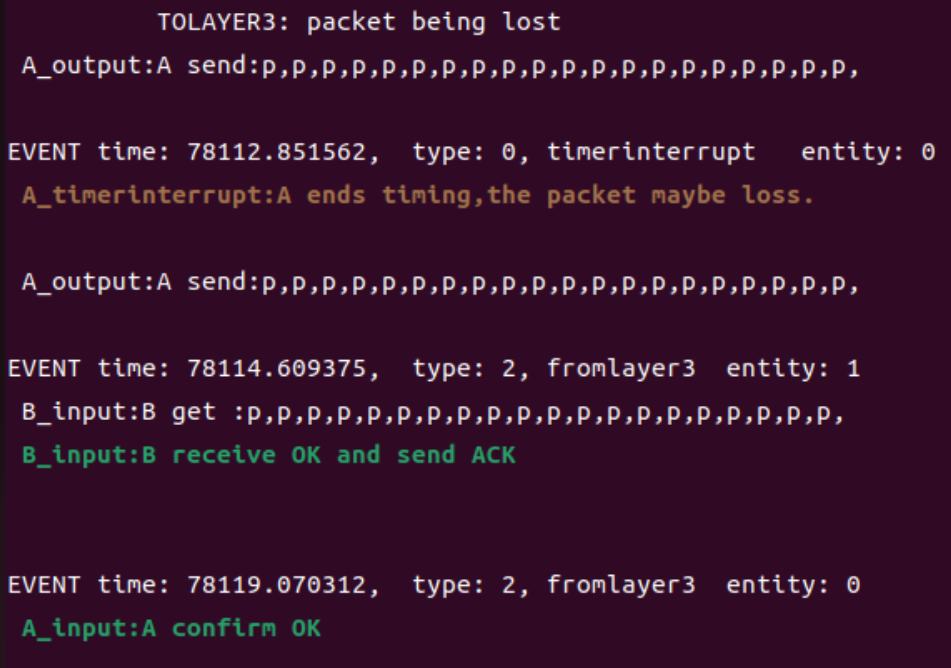
最终输出结果表示成功发送100个包，模拟无误。

1. **没有 error，只有 loss 的情况，该情况只设置坏包概率，此条件下会出现仿真模拟会模拟出两种如下情况**

在初始输入条件为：100 0 0.2 1000 2

**(a)数据包出现 loss**

我们可以看到：



该情况模拟的是A端发送的数据分组在向B发送过程中丢失，可以看到我们这里对这个数据包进行的处理是：在A端等待ACK包超时时，A端自动重发了数据包，这是符合初始设计和预期的。在此之后A重发了数据，碰巧各步均成功，问题得到了解决。

该过程的流程是

A\_output:发送分组

分组丢失

A\_timeinterrupt:超时，停止计时，调用A\_output函数来重发数据分组

如果下一次发送没有出现任何问题，则

A\_output:重新发送分组

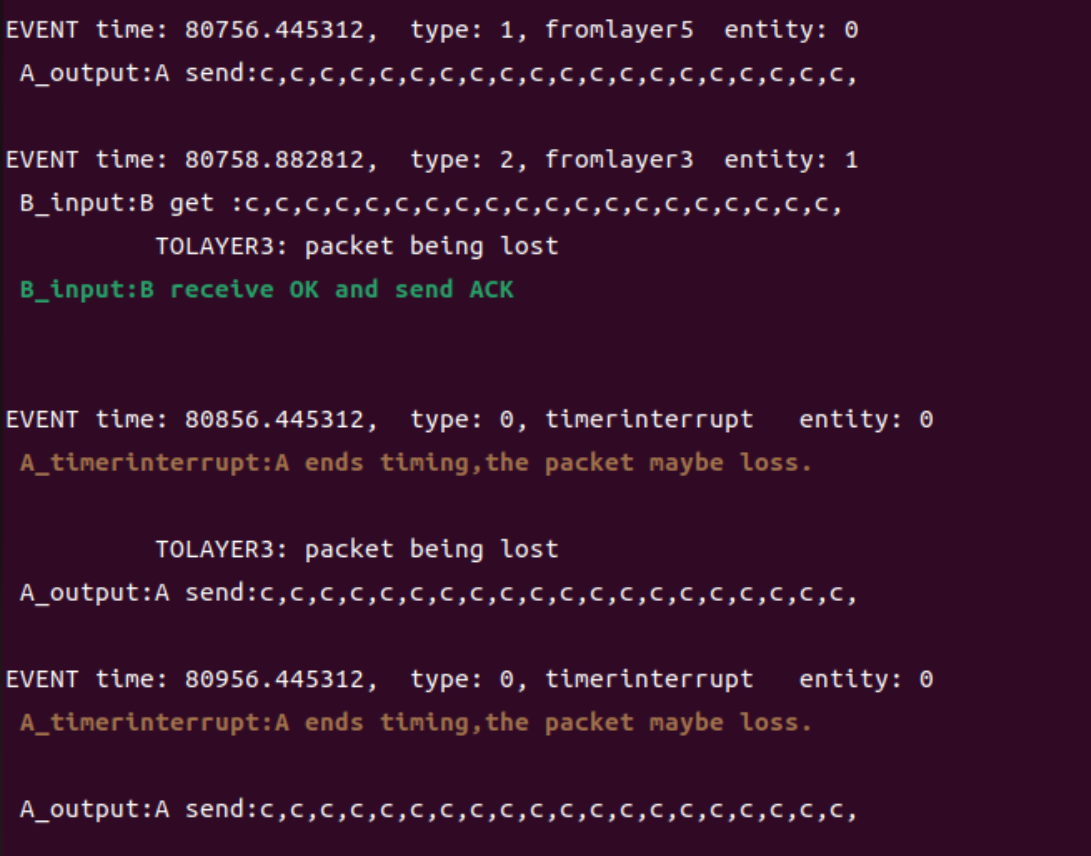
B\_input:接受分组

B\_input：校验和检验成功，发送ACK

A\_input：A收到B端ACK，可以继续发送新分组或结束

**(b)ACK 包出现 loss：**

我们可以看到：



该情况模拟的是B端向A端反馈发送的ACK在发送过程中丢失，可以看到我们这里对这个数据包进行的处理是：在A端等待ACK包超时时，A端自动重发了数据包，但在上图中重发中又出现丢包，故又一次重发了数据包，最终结果是得到了解决，这是符合初始设计和预期的。

该过程的流程是

A\_output:发送分组

B\_input:接受分组

B\_input：校验和检验成功，发送ACK

ACK包丢失

A\_timeinterrupt:超时，停止计时，调用A\_output函数来重发数据分组

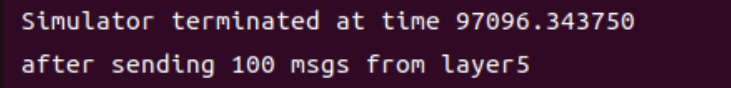
如果下一次发送没有出现任何问题，则

A\_output:重新发送分组

B\_input:接受分组

B\_input：校验和检验成功，发送ACK

A\_input：A收到B端ACK，可以继续发送新分组或结束



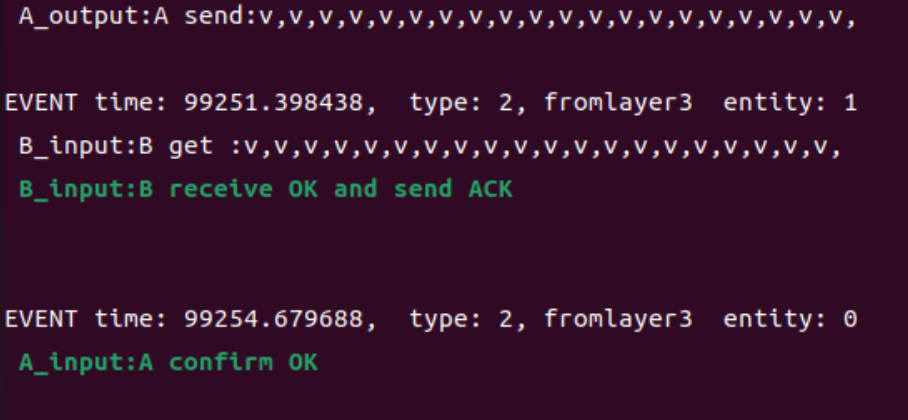
最终输出结果表示成功发送100个包，模拟无误。

(4)**有error，也有loss的情况：**

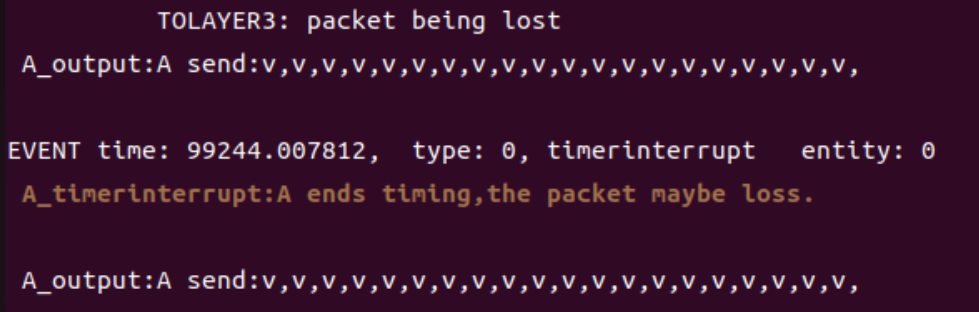
在初始输入条件为：100 0.2 0.2 1000 2

我们可以看到上述所有情况均出现：

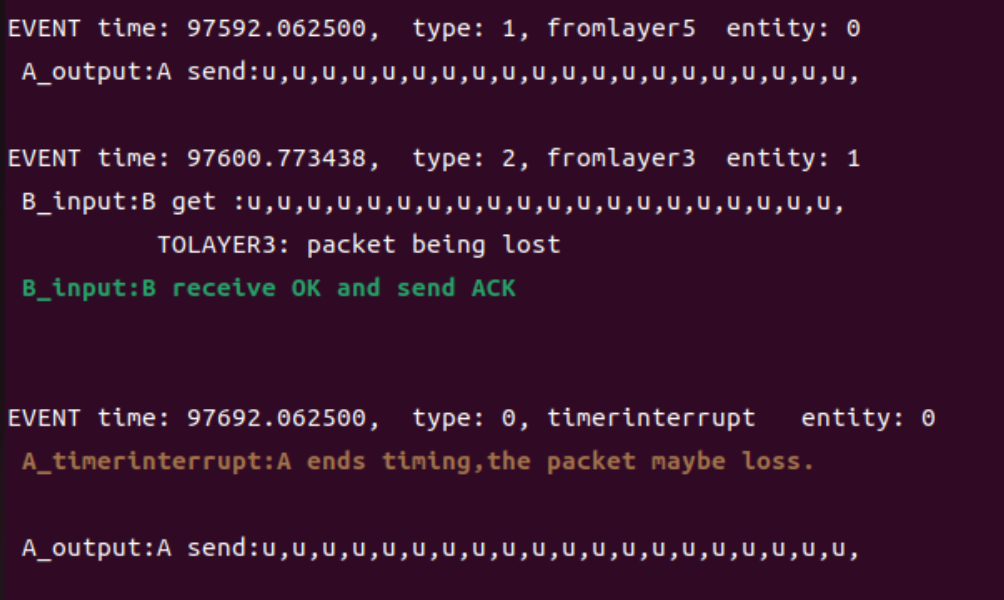
1. 无误情况



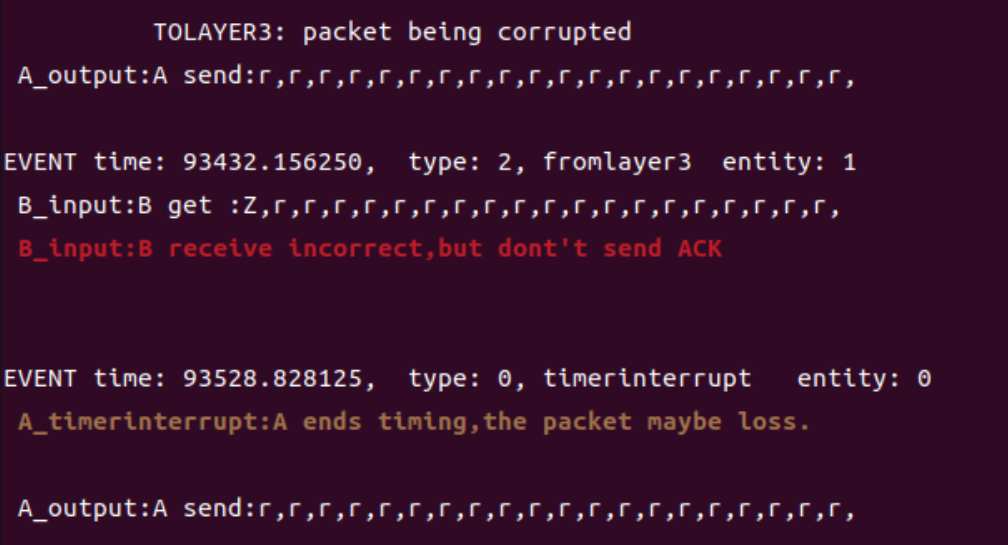
1. 数据分组loss



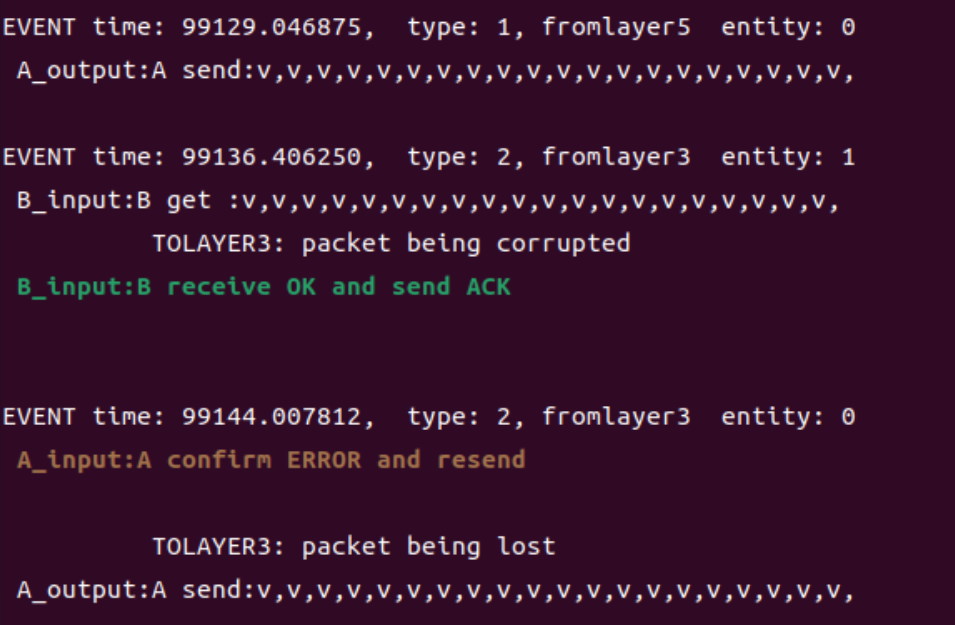
1. ACK包loss



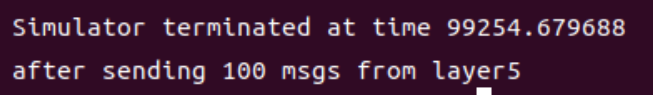
1. 数据分组error



1. ACK包error



由于所对应的处理方式与实现均与前三种情况重合，故在这里不再单独做重复性解释。



最终输出结果表示成功发送100个包，模拟无误。

**5.1.3结果分析**

对于所测结果进行分析，可以看出在实验指导书所要求的测试条件下（合理选择仿真参数:发送数据包个数（>=100）、数据包发送间隔、出错率、 丢包率等），仿真结果中出现了各种可能出现错误的情况，同时也对于各种出现的丢包和错包进行了合理的、符合预期设计的正确处理，这表明我们对stop-and-wait协议的模拟仿真实现已成功。

## **5.2 Go-Back-N协议的功能测试与结果分析**

# 六、总结

对于本周对于stop-and-wait协议的实现中，首先，我们发现在实现过程中，需要注意的细节设置非常多，例如数据包的格式、发送和接收的顺序、超时时间等等。而在机制的设置上，我们遇到了以下几类关键问题：

* 构造超时重传机制，以应对数据分组或确认分组的丢失或延迟。
* 如何给数据分组和确认分组编号，以防止重复或乱序的传输。
* 如何计算信道利用率，以评估协议的效率和性能。

其次，在调试过程中，我们发现了一些自己之前没有注意到的问题，通过查找资料解决了这些问题。

最后，对于本次实验我们有如下建议：

* 希望在实验指导书中提供更多的参考资料和示例代码，以便于我们在遇到困难时能够自主查阅和学习。同时，也希望能够有更多的互动和反馈，让我们能够及时得到老师或助教的指导和评价。
* 希望在实验中增加一些扩展内容或挑战性任务，让我们能够尝试更多的可能性和创新性。

总体来说，这次实践让我们更深入地了解了计算机网络协议的实现过程，并且提高了我们在网络编程能力和调试能力。

附：进度报告：本周所有任务均已成功完成