天津大学

《计算机网络》课程设计报告

第一周：Stop-and-Wait 协议

学　　号 3020205015

姓　　名 石云天

学　　院 未来技术学院

专　　业 计算机科学与技术

年　　级 2020

任课教师 周晓波

2023 年 4 月 4 日

一、任务要求

本次实验要求在仿真环境中编写传输层可靠数据传输代码以实现单向传输的 Stop-and-Wait 与 Go-Back-N 协议。网络仿真过程的模拟、数据包发送与接收以及除传输层外各层功能已经封装为基础代码，这些基础代码已经构建出一套完善的网络仿真环境，如下图 1 所示：

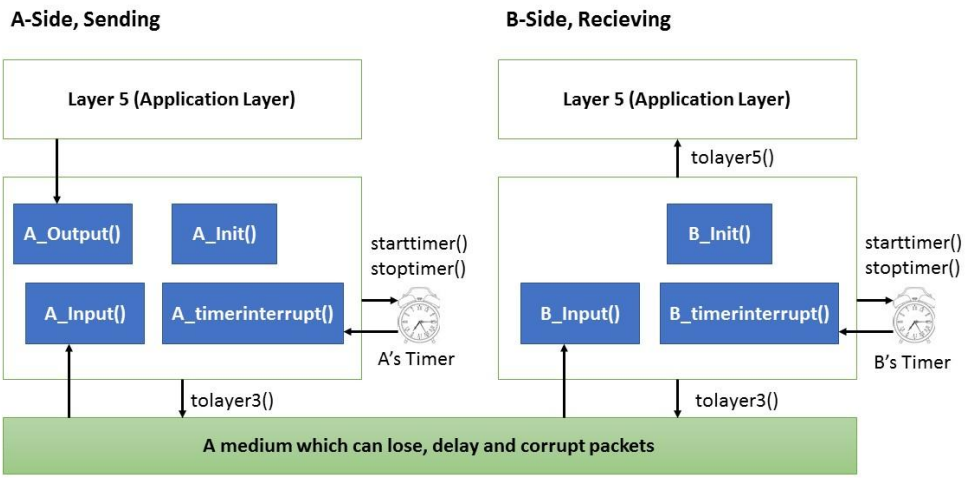

图 1 RDT 仿真框架图

上图中每个网络节点的传输层代码需要自行填补，主要包括节点传输层的初始化操作、节点接收到应用层消息的处理过程、节点接收到网络层数据包的处理过程以及节点计时器到时的响应过程等内容，所有代码均集成在 A_Output(), A_Input(),A_Init(),A_timerinterrupt,B_Input(),B_Init(),B_timerinterrupt 这七个接口函数中。在此过程中要解决的主要问题有：

1.处理比特差错。发送端对数据进行校验和运算后发送给接收端，接收端需进行差错检验并进行接收方反馈，发送端若未接收到接收端的肯定确认，则进行重传。

2.加入分组序号。发送端对数据进行分组编号，接收端反馈需要带有分组编号：接收端肯定确认则对当前分组发送一个 ACK，否定确认则对上次正确接收的分组发送一个 ACK。

3.解决丢包问题。发送端使用倒计数计时器，如果超时仍未接收到来自接收端的肯定确认，则进行重传。

在完成所有节点的算法后，运行网络仿真，记录下仿真过程与结果，并对结果进行分析和总结。

二、协议设计

2.1 总体设计

2.1.1 可调用函数模块

在 Stop-and-Wait 与 Go-Back-N 协议的设计与实现中。框架代码现有的函数已经提供了一套完整的网络仿真环境，可以完成发送端数据的产生等数据传输过程中除传输层外网络各层的功能。这一模块的功能是 Stop-and-Wait 与 Go-Back-N 协议共同需要用到的。

从图 1 可以看出，这其中用到的函数主要有 tolayer3()、tolayer5()、starttimer()与 stoptimer()四个。其中 starttimer 函数用于打开一个发送端或接收端的倒计数定时器，若当前同一端已有一个计时器，则无法打开新的计时器。stoptimer 函数用于终止当前已经打开的发送端或接收端的计时器。tolayer3 函数用于将封装好的数据包发送至信道，进行数据包传输，主要用于发送端发送数据包以及接收端发送 ACK 包。tolayer5 函数主要用于接收端将数据包解封装所得的 message 发送至上层。

2.1.2 缓冲区模块

缓冲区模块是发送端和接收端对信道传输中可能出现的比特差错和丢包等错误进行正确处理的前提。缓冲区中存储的主要是此前成功发送的数据包与 ACK 包，具体内容在下文中有更详细的解释。

2.1.3 发送端模块

发送端模块具体实现的是发送端在数据传输中的操作，主要包括 A_Output(), A_Input(),A_Init(),A_timerinterrupt 四个函数。其中 A_Init()函数主要用于发送端相关变量的初始化；A_Output()函数用于将接收到的上层发来的 message 封装为 pkt 结构的数据包，并发送到信道使其传输到接收端；A_Input()函数用于接收 ACK 包并进行相关检验；A_timerinterrupt 函数用于对计时器超时的情况进行相关处理。

2.1.4 接收端模块

接收端模块主要功能实现的是接收端在可靠数据传输中的操作，主要包括 B_Input(),B_Init()两个函数。其中 B_Init()函数用于接收端相关变量与接收端缓冲区的初始化；B_Input()函数用于接收发送端发送的数据包，并对其是否按序与完好进行检验。对于完好的数据包，B_Input()函数将其解封装得到的 message 发送至上层，并向发送端发送 ACK 包表示接收成功。

2.2 Stop-and-Wait 协议的设计

2.2.1 数据结构设计

2.2.1.1 msg 与 pkt 结构体

msg 与 pkt 结构体是框架代码中给出的内容，在两个协议的实现中都需要进行调用，其结构如下：

```
struct msg{
    char data[20];
};

struct pkt{
    int seqnum;
    int acknum;
    int checksum;
    char payload[20];
};
```

msg 结构体是上层调用发送端想要发送到接收端的消息内容。pkt 结构体是在发送端进行封装然后发送到信道，与 msg 结构体相比多三个变量 sequnm、acknum 与 checksum。其中 seqnum 是发送端发送到信道的数据包序号；acknum 是接收端发送到信道的 ACK 包序号；checksum 是校验和，用于检验数据包在信道传输的过程中是否出现比特差错。

2.2.1.2 全局变量

Stop-and-Wait 协议实现中主要设置了以下几个全局变量：(int)A_status,(int) B_status 与(float)timeout。其中 A_status 表示当前发送端按序应当发送的数据包的序号，在 A_Init()中初始化为 0。在 A_Input()中收到 A_status 序号相应的 ACK 包后，发送端确认该序号对应的数据包已经被接收端成功接收，(A_status+1)%2；B_status 表示当前 B 端按序应当接收的数据包的序号，在 B_Init 函数中，将其初始化为 0。在 B_Input()中接收到序号为 B_status 的数据包后，(B_status+1)%2；timeout 变量是倒计数定时器的倒计数数值，由于数据包在 layer3 中传输一次用时约为 5sec，在数据包发送与 ACK 包接收的时间和上需要留出一定冗余，设置为 15sec。

2.2.1.3 缓冲区设置

对 A、B 端分别建立缓冲区 A_buf 与 B_buf，两者均为 pkt 类型的结构体。由于 Stop-and-Wait 协议中发送的数据包序号为 0、1 交替，故缓冲区只需存储一个数据包便满足要求。A_buf 用于存储当前发送端发送的前一个数据包，在 A_Output()每发送一个数据包后进行备份，便于在计时器超时时进行数据包重传；

B_buf 存储当前接收端发送的前一个 ACK 包，通过 B_Init()函数进行初始化，并在 B_Input()每发送一个 ACK 包后进行备份。当接收端接收到比特差错或失序的数据包时，会向发送端发送 B_buf 中的 ACK 包以通知发送端数据包出现问题，需要进行重传。

2.2.2 协议规则设计

Stop-and-Wait 协议通过在每个数据包的 header 中设置校验来处理信道传输中可能出现的比特差错问题，通过设置倒计数计时器来处理可能存在的丢包问题。

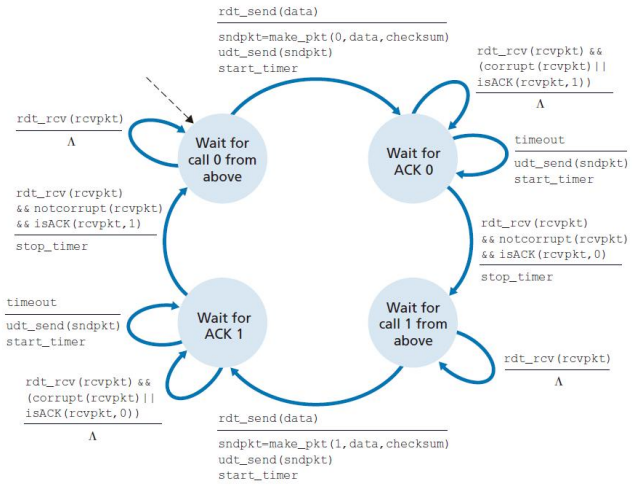

图 2 发送端 FSM

如上图 2 所示，发送端首先等待上层发送来的消息 0，在封装 header 之后，将其发送到信道，并启动计时器，开始等待接收端发送的 ACK 包。如果接收到的 ACK 消息与当前发送到信道的数据包序号不符，或者数据包内容出现比特差错，则不予处理，等待至 timeout。若 timeout 时仍未接收到消息，发送端会判定信道发送过程中出现丢包，会重新发送数据包 0。

发送端会重复上述过程直至收到接收端的 ACK 消息为止，然后开始等待来自上层的调用 1。由于 Stop-and-Wait 协议要求消息确定发送成功之后才发送下一个消息，序号通过 0、1 交替即可使得消息按序发送。

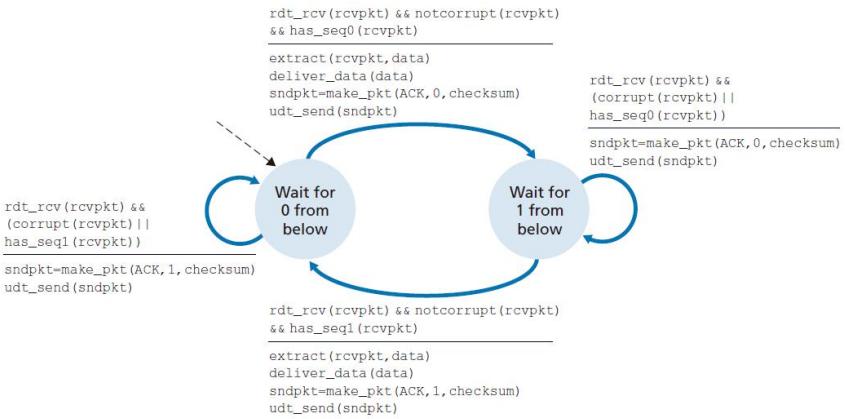

图 3 接收端 FSM

如上图 3 所示，接收端等待来自下层的数据包 0，在接收到信道发送的数据包时会进行数据包序号和内容的检查，判断是否出现乱序或比特差错等问题。若在等待调用 0 时收到了序号为 1 的数据包或出现比特差错，接收端会向发送端发送一个序号为 1 的 ACK 消息。如果一切正常，则会向发送端发送一个序号为 0 的 ACK 消息，并开始等待来自下层的数据包 1。

2.3 Go-Back-N 协议的设计

2.3.1 数据结构设计

2.3.1.1 全局变量

Go-Back-N 协议实现中主要设置以下几个全局变量：(int)Max_windowsize, (int)A_base,(int)A_nextseq,(int)B_status 与(float)timeout。其中 Max_windowsize 表示滑动窗口的大小，此处设置为 4；A_base 为当前滑动窗口的基序号值，表示当前未接收到 ACK 包的最早数据包的序号值；A_seqnum 指当前还未发送的最早数据包的序号值，以上两个值都 A_Init 函数中进行初始化；B_status 表示当前接收端按序应当接收的数据包的序号，在 B_Init 函数中，将其初始化为 0；timeout 变量是倒计数定时器的倒计数数值，由于数据包在 layer3 中传输一次用时约为 5sec，在数据包发送与 ACK 包接收的时间和上需留出一定冗余，设置为 15sec。

2.3.1.2 缓冲区设置

Go-Back-N 协议实现中设置了以下两个缓冲区：A_buf 与 B_buf。其中 A_buf 是 pkt 结构体的数组，其大小设置为 64，用来存储发送端发送的数据包，以便接收端判定失序时进行对应数据包的重传。Go-Back-N 协议中设置的滑动窗口，即为缓冲区 A_buf 中从 A_base 开始到 A_base+Max_windowsize 结束部分 pkt 结构体数组；B_buf 为一个 pkt 结构体，存储接收端此前接收到的最后一个按序数据包对应的 ACK 包，用于在接收到失序数据包时进行 ACK 包重传使发送端得知当前存在失序问题。

2.3.2 协议规则设计

Go-Back-N 协议通过设置一个滑动窗口以限定可以发送的多个分组的序号范围，允许发送端发送在窗口范围内的数据包，实现 pipelining 的发送。在滑动窗口的实现过程中，Go-Back-N 协议设置基序号 base 来表示目前最早没有接收到相应 ACK 包的数据包序号，设置“下一个序号”nextseqnum 以表示当前还未发送的最早数据包序号，并设置 N 为滑动窗口大小。Go-Back-N 协议在接收端实行累计应答机制（cumulative ACK），即发送端接收到的接收端发送的 ACK n 时，序号 n 及 n 以前的数据包都已成功发送并接收。

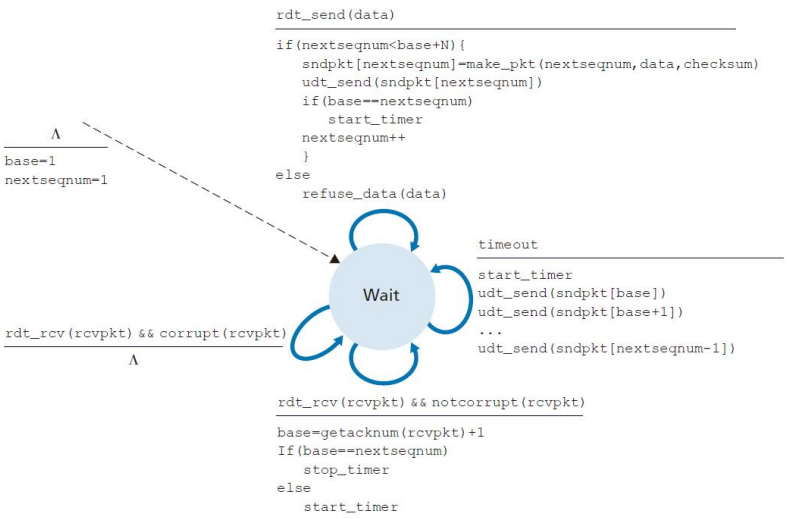

图 4 发送端 FSM

如上图 4 所示，发送端将滑动窗口的 base 和 nextseqnum 进行初始化，并等待来自上层 layer5 的调用。在接收到上层发送的信息后，判断 nextseqnum 是否小于 base+N，若小于则对信息进行 header 封装，发送到信道，并将 nextseqnum++，同时若当前发送信息的序号与 base 值相等，则表示当前发送的为滑动窗口中的第一个数据包，打开倒数计时器并进行计时；若不小于则代表当前要发送的数据包超出滑动窗口的范围，不予发送。

在等待接收接收端发来的 ACK 包时，发送端可以持续发送在滑动窗口中的数据包，这便是流水线的实现方式。若接收到按序且完好的 ACK 包，则表示 base 对应的数据包发送成功，base++。若当前 base==nextseqnum，则表示可发的数据包已成功发送完毕，终止计时器；反之，终止当前计时器并开启新计时器。若接收到的 ACK 包失序或出现比特差错等情况，则不做处理等待至超时。当计时器出现超时，则判定最近一个成功发送的数据包（序号为 base-1）之后发送的所有数据包均失序，此时需要开启新的倒计时计时器并全部进行重传。

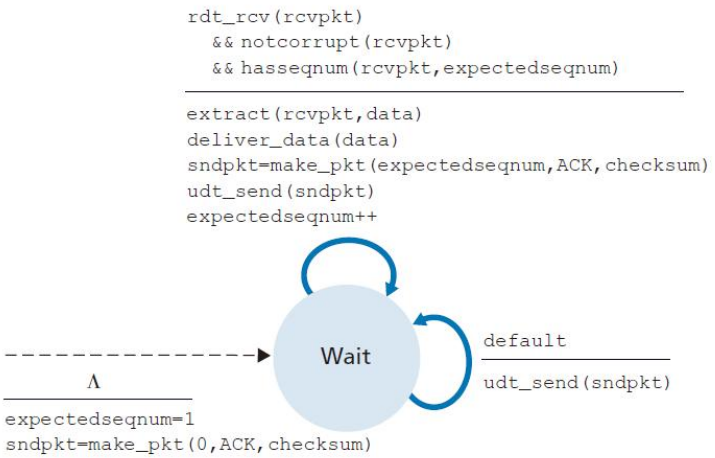

图 5 接收端 FSM

如上图 5 所示，接收端等待来自下层 layer3 的调用。在接收到下层发送的数据包后，判断其是否按序且完好。若出现失序或比特差错等情况，则发送最后一个按序数据包对应的 ACK 包；若数据包按序且完好，则将数据包解封装得到的信息发送到 layer3，并发送相应的 ACK 包。

三、协议实现

3.1 Stop-and-Wait 协议的实现

3.1.1 A_Init 函数与 B_Init 函数

A_Init()将 A_Status 初始化为 0，表示初始时发送端要按序发送的第一个数据包序号为 0。B_Init()将 B_Status 与 B_buf 初始化，其中将 B_Status 置为 0，表示初始时接收端要按序接收的第一个数据包序号为 0。同时对 B_buf 初始化，建立一个序号为-1 的 ACK 包以保证如果初始接收到第一个数据包存在失序或比特差错时，存在 ACK 包可以发送到发送端。

3.1.2 A_Output 函数

伪代码如下：

```
A_Output(message)
struct msg message;
{
    将 message 封装为 pkt 结构的数据包;
    将待发送的数据包存入 A_buf 中;
    tolayer3(0,packet);
    打开倒计数计时器;
    return;
}
```

首先建立一个 pkt 结构体，对 layer3 发送来的 message 进行封装。其中序号值 seqnum 设置为当前的 A_Status%2；acknum 由于在 A 端发送的信息包中无实际作用，设置为与 seqnum 相同的值；由于在信道传输过程中，seqnum、acknum 与数据内容都有可能出现比特差错，校验和 checksum 设置为上述内容之和。在封装 message 的过程中，将内容备份至 pkt 类型的结构体 A_buf 中，在完成对 message 的封装后，A 端调用 tolayer5()将封装好的数据包发送到信道，并打开 A 端的计时器。

3.1.3 A_Input 函数与 B_Input 函数

A_Input()函数伪代码如下：

```
A_Input(packet)
struct pkt packet;
{
    对 ACK 包的校验和进行判断；
    if 检验和出错 or 失序
        return；
    终止当前计时器；
    A_Status = (A_Status + 1) % 2;
    return;
}
```

A_Input()中，参数为从信道中收到的 B 端发送到信道的 ACK 包。首先对 ACK 包的 checksum 进行校验，检验其是否存在比特差错，完后检查 acknum 值是否与当前 A_Status%2 的值相同，不同则为乱序数据包。若存在比特差错或乱序，则直接 return，退出 A_Input()函数，等待 A 端由于 timeout 进行信息重传。若一切正常，则终止 A 端计时器，并更新 A_Status 的值。

B_Input()函数伪代码如下：

```
B_Input(packet)
struct pkt packet;
{
    对数据包的校验和与序号进行判断；
    if 检验和出错 or 失序
        发送 B_buf 中的 ACK 包；
        return；
    将完好的数据包解封装，将得到的 message 发送至 layer5；
    建立 pkt 类型的 ACK 包；
    将待发送的 ACK 包备份在 B_buf 中；
    发送 ACK 包；
    B_Status = (B_Status + 1) % 2;
    return;
}
```

B_Input()函数中，参数为从信道中收到的 A 端发送到信道的数据包，流程大致与 A_Input()大致相同，首先对数据包的 seqnum 进行检查，若为乱序数据包，则向 A 端发送数据包 B_buf。然后对数据包的检验和进行检查，若出现比特损失，则不做任何处理，直接 return，退出当前函数，等待 A 端由于 timeout 进行数据包重传。

若一切正常，则建立 pkt 类型的结构体 pac，作为回传到 A 端的 ACK 包。pac 的 seqnum 值与 acknum 值均为 B_Status%2，校验和 checksum 的计算方法与 A_Output()中相同。同样的，需要将 pac 备份至 B_buf 中，以备后来接收到乱序的数据包时发往 A 端。

3.1.4 A_timerinterrupt 函数

A_timerinterrupt()是 A 端计时器 timeout 时调用的函数，用以进行数据包重传。由于先前数据包已经备份到 A_buf 中，此处只需调用 tolayer3()将 A_buf 发送到信道，并建立一个新的 A 端计时器即可。

3.2 Go-Back-N 协议的实现

3.2.1 A_Init 函数与 B_Init 函数

A_Init()将 A_base 和 A_nextseq 这两个全局变量初始化为 0，表示初始时发送端要按序发送的第一个数据包序号为 0。B_Init()将 B_Status 与 B_buf 初始化，其中将 B_Status 置为 0，表示初始时接收端要按序接收的第一个数据包序号为 0。同时对 B_buf 初始化，建立一个序号为-1 的 ACK 包以保证如果初始接收到第一个数据包存在失序或比特差错时，存在 ACK 包可以发送到发送端。

3.2.2 A_Output 函数

伪代码如下：

```
A_Output(message)
struct msg message;
{
    if 待发送数据包序号 A_nextseq 在滑动窗口外
        return;
     将 message 封装成一个 pkt 结构数据包;
    将待发送的数据包存入 A_buf 中;
    tolayer3(0,packet);
    if 滑动窗口基序号 A_base 与当前发送的数据包序号 A_nextseq 相同
        打开倒计数计时器;
    A_nextseq++;
    return;
}
```

该函数主要功能是将上层 layer5 发送来的 message 封装为 pkt 结构的数据包并发往 layer3。首先判断准备发送的数据包序号 A_nextseq 是否小于等于 A_base+Max_windowsize，若小于等于，则表示在滑动窗口内，将 message 进行封装，并设置 seqnum,acknum 与 checksum 的值。其中，seqnum 值为数据包序号，赋值为 A_nextseq；acknum 在发送端发送的数据包中并无实际作用，同样赋值为

A_nextseq；checksum 为校验和，赋值为 seqnum、acknum 与信息的和。若大于，则表示不在滑动窗口内，直接返回终止函数即可。

在完成对 message 的封装后，将其存入缓冲区 A_buf 中，以便出现失序或比特差错时进行重传，备份后即可将其发送到 layer3。此时，还需要判断当前 A_base 与 A_nextseq 的值是否相等，若相等则开启倒计数计时器，在判断后，A_nextseq++。

3.2.3 A_Input 函数与 B_Input 函数

A_Input()函数伪代码如下：

```
A_Input(packet)
struct pkt packet;
{
    对 ACK 包的校验和进行判断；
    if 检验和出错
        return；
    A_base = packet 序号 + 1；
    if 滑动窗口中所有数据包都成功发送
        终止当前计时器；
    else
        终止当前计时器；
        打开新计时器；
    return;
}
```

A_Input()的主要功能是对接收到的接收端发送的 ACK 包进行检验和处理。当前发送端应当按序接收到的 ACK 包的序号是滑动窗口的基序号 A_base 值，A_Input()首先对 ACK 包的校验和 checksum 与序号 acknum 进行校验和判断，若出现比特差错，则直接返回终止函数。若出现失序，例如当前应接收 ACK0，而接受到 ACK1，由于 Go-Back-N 协议实行累计应答机制(cumulative ACK)，因此判断 packet1 及以前数据包都成功发送；若 ACK 包完好，则将 A_base 赋值为数据包序号+1，表示原先序号为 A_base 的数据包已成功发送。随后判断 A_base 与 A_nextseq 是否相等，若相等，则表示当前上层发送至发送端的信息都已成功发送，此时终止计时器；若不相等，则表示滑动窗口中仍有已发送但未确认成功发送的数据包，此时终止之前的计时器，并开启新的计时器。

B_Input()函数伪代码如下：

```
B_Input(packet)
struct pkt packet;
{
```

```
        对数据包的校验和与序号进行判断；
        if 检验和出错 or 失序
            发送 B_buf 中的 ACK 包；
            return；
        将完好的数据包解封装，将得到的 message 发送至 layer5；
        建立 pkt 类型的 ACK 包；
        将待发送的 ACK 包备份在 B_buf 中；
        发送 ACK 包；
        B_Status++;
        return;
    }
```

B_Input()函数的主要功能是对接收到的数据包的校验和与序号进行判断，若存在比特差错或失序，则向发送端发送当前接收到的最后一个按序数据包对应的 ACK 包并直接返回终止函数；若数据包按序且完好，则解封装并将响应消息发送至 layer5，建立相应的 ACK 包并将其备份后发送到 layer3。接收端按序应当接收到的数据包的序号为 B_status，在确定当前接收的数据包按序且完好后，B_status++。

3.2.4 A_timerinterrupt 函数

若发送端计时器超时，则判定当前滑动窗口中从基序号 A_base 到序号 A_nextseq-1 的所有数据包均失序，都需要进行重传。A_timerinterrupt()开启新的倒计数计时器，并从 A_buf 中取出相应数据包发送到信道 layer3。

四、实验结果及分析

4.1 Stop-and-Wait 协议的功能测试与结果分析

4.1.1 无 error 无 loss 情况

设置数据包个数为 100，丢包率和错误率均为 0，数据包间隔为 1000。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

```
EVENT time: 77338.109375,  type: 1, fromlayer5  entity: 0
A sending:
  seq:0, ack:0 check:FFFFE1E1
  message: iiiiiiiiiiiiiiiiiiii
```

```
EVENT time: 77346.062500,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:0, ack:0 check:FFFFE1E1
  message:iiiiiiiiiiiiiiiiiiii

B receiving:0,looking for:0
B:sending:
  seq:0, ack:0 check:FFFFBC73
  message:ACK

EVENT time: 77355.804688,  type: 2, fromlayer3  entity: 0
A receiving acknum:0,looking for:0
ACK!


EVENT time: 78696.242188,  type: 1, fromlayer5  entity: 0
A sending:
  seq:1, ack:0 check:FFFFD7D6
  message: jjjjjjjjjjjjjjjjjjjj

EVENT time: 78705.101563,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:1, ack:0 check:FFFFD7D6
  message:jjjjjjjjjjjjjjjjjjjj

B receiving:1,looking for:1
B:sending:
  seq:0, ack:1 check:FFFFBC72
  message:ACK

EVENT time: 78706.710938,  type: 2, fromlayer3  entity: 0
A receiving acknum:1,looking for:1
ACK!
```

由此可以看出，在 A、B 端发送、接收数据包后，会有数据输出到终端。通过这些输出到终端的内容可以看出，A 端在发送下一个数据包前，都会调整数据

包的编号，B 端接收到数据包后，都会对当前数据包发送一个 ACK 包用于确认，A 端在收到正确的 ACK 包后会等待下一次来自上层的调用。

4.1.2 无 loss 只有 error 情况

设置数据包个数为 100，错误率为 0.2，丢包率为 0，数据包间隔为 1000。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

4.1.2.1 数据包出现 error

```
EVENT time: 14503.739258,  type: 1, fromlayer5  entity: 0
A sending:
   seq:0, ack:0 check:FFFFA5A5
   message: oooooooooooooooooooo

          TOLAYER3: packet being corrupted

EVENT time: 14507.018555,  type: 2, fromlayer3  entity: 1
B receiving:
   seq:0, ack:0 check:FFFFA5A5
   message:Zooooooooooooooooooo

B receiving:0,looking for:0
Corrupt!
B:sending:
   seq:0, ack:1 check:FFFFBC72
   message:ACK

EVENT time: 14513.459961,  type: 2, fromlayer3  entity: 0
A receiving acknum:1,looking for:0
Wrong ACK number!


EVENT time: 14523.739258,  type: 0, timerinterrupt   entity: 0
A resending:
   seq:0, ack:0 check:FFFFA5A5
   message:oooooooooooooooooooo
```

```
EVENT time: 14529.326172,  type: 2, fromlayer3  entity: 1
B receiving:
    seq:0, ack:0 check:FFFFA5A5
    message:oooooooooooooooooooo

B receiving:0,looking for:0
B:sending:
    seq:0, ack:0 check:FFFFBC73
    message:ACK

EVENT time: 14536.808594,  type: 2, fromlayer3  entity: 0
A receiving acknum:0,looking for:0
ACK!
```

由此可以看出，B 端接收到错误的数据包后，会向 A 端返回一个编号为上一次正确接收的数据包编号的 ACK 数据包。A 端在接收到 ACK 包后发现此时收到的 ACK 编号与目标编号不符，于是忽略该 ACK 包，继续等待从 B 端传来的 ACK 包，而非进入下一发包流程或重发当前数据包，直到 A 端等待 ACK 包超时，A 端重发数据包，得到 B 端正确回应后，才进入下一发包流程。

4.1.2.2 ACK 包出现 error

```
EVENT time: 10992.177734,  type: 0, timerinterrupt   entity: 0
A resending:
  seq:0, ack:0 check:FFFFCDCD
  message:kkkkkkkkkkkkkkkkkkkk

EVENT time: 10993.942383,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:0, ack:0 check:FFFFCDCD
  message:kkkkkkkkkkkkkkkkkkkk

B receiving:0,looking for:0
B:sending:
  seq:0, ack:0 check:FFFFBC73
  message:ACK

          TOLAYER3: packet being corrupted
```

```
EVENT time: 11000.044922,  type: 2, fromlayer3  entity: 0
A receiving acknum:0,looking for:0
Corrupt!


EVENT time: 11012.177734,  type: 0, timerinterrupt   entity: 0
A resending:
  seq:0, ack:0 check:FFFFCDCD
  message:kkkkkkkkkkkkkkkkkkkk

EVENT time: 11020.889648,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:0, ack:0 check:FFFFCDCD
  message:kkkkkkkkkkkkkkkkkkkk

B receiving:0,looking for:1
Received!
B:sending:
  seq:0, ack:0 check:FFFFBC73
  message:ACK

EVENT time: 11023.408203,  type: 2, fromlayer3  entity: 0
A receiving acknum:0,looking for:0
ACK!
```

当 ACK 包出现错误时，A 端忽略该 ACK 包，继续等待从 B 端传来的 ACK 包，而非进入下一个发包流程或重发当前数据包。直到 A 端等待 ACK 包超时，A 端重发数据包，得到 B 端的正确回应，才进入下一发包流程。

4.1.3 无 error 只有 loss 情况

设置数据包个数为 100，错误率为 0，丢包率为 0.2，数据包间隔为 1000。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

4.1.3.1 数据包出现 loss

```
EVENT time: 11194.677734,  type: 0, timerinterrupt   entity: 0
A resending:
```

```
  seq:0, ack:0 check:FFFFC3C3
  message:lllllllllllllllllll

         TOLAYER3: packet being lost

EVENT time: 11214.677734,  type: 0, timerinterrupt   entity: 0
A resending:
  seq:0, ack:0 check:FFFFC3C3
  message:lllllllllllllllllll


EVENT time: 11215.838867,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:0, ack:0 check:FFFFC3C3
  message:lllllllllllllllllll

B receiving:0,looking for:0
B:sending:
  seq:0, ack:0 check:FFFFBC73
  message:ACK


EVENT time: 11220.035156,  type: 2, fromlayer3  entity: 0
A receiving acknum:0,looking for:0
ACK!
```

可以看出当数据包丢失时，B 未接收到数据包，因此不会向 A 发送 ACK 包。直到 A 等待 ACK 包超时，A 重发数据包，得到 B 的正确回应，才进入下一发包流程。

4.1.3.2 ACK 包出现 loss

```
EVENT time: 13048.839844,  type: 0, timerinterrupt   entity: 0
A resending:
  seq:1, ack:0 check:FFFFB9B8
  message:mmmmmmmmmmmmmmmmmmmm
```

```
EVENT time: 13050.383789,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:1, ack:0 check:FFFFB9B8
  message:mmmmmmmmmmmmmmmmmmmm

B receiving:1,looking for:1
B:sending:
  seq:0, ack:1 check:FFFFBC72
  message:ACK

         TOLAYER3: packet being lost

EVENT time: 13068.839844,  type: 0, timerinterrupt   entity: 0
A resending:
  seq:1, ack:0 check:FFFFB9B8
  message:mmmmmmmmmmmmmmmmmmmm


EVENT time: 13074.757813,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:1, ack:0 check:FFFFB9B8
  message:mmmmmmmmmmmmmmmmmmmm

B receiving:1,looking for:0
Received!
B:sending:
  seq:0, ack:1 check:FFFFBC72
  message:ACK


EVENT time: 13082.152344,  type: 2, fromlayer3  entity: 0
A receiving acknum:1,looking for:1
ACK!
```

可以看出，当 B 端正确接收到数据包后，会将数据包传到上层。当 ACK 包丢失时，A 未接收到 ACK 包，从而继续等待 ACK 包。直到 A 等待 ACK 包超时，A 重发数据包，B 得

到已经正确接收的数据包编号，便不再上传数据到上层，并向 A 端发送带有上次正确收到的数据包编号的 ACK 包，此时 A 端得到 B 端的正确回应，于是进入下一发包流程。

4.1.4 有 error 有 loss 情况

设置数据包个数为 100，错误率为 0.2，丢包率为 0.2，数据包间隔为 1000。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

1.数据包出现 loss，则不会出现 ACK 包，故同上述数据包出现 loss 的情况。

2.数据包出现 error，ACK 包出现 loss。

```
EVENT time: 27006.437500,  type: 0, timerinterrupt   entity: 0
A resending:
  seq:0, ack:0 check:FFFF0000
  message:ffffffffffffffffffff

         TOLAYER3: packet being corrupted

EVENT time: 27014.144531,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:0, ack:0 check:FFFF0000
  message:Zffffffffffffffffffff

B receiving:0,looking for:1
Corrupt!
B:sending:
  seq:0, ack:0 check:FFFFBC73
  message:ACK

         TOLAYER3: packet being lost

EVENT time: 27026.437500,  type: 0, timerinterrupt   entity: 0
A resending:
  seq:0, ack:0 check:FFFF0000
  message:ffffffffffffffffffff


EVENT time: 27029.335938,  type: 2, fromlayer3  entity: 1
```

```
B receiving:
  seq:0, ack:0 check:FFFF0000
  message:ffffffffffffffffffff

B receiving:0,looking for:1
Received!
B:sending:
  seq:0, ack:0 check:FFFFBC73
  message:ACK

EVENT time: 27031.738281,  type: 2, fromlayer3  entity: 0
A receiving acknum:0,looking for:0
ACK!
```

可以看到，接收端在接收到错误的数据包后，会向发送端返回一个编号为上一次正确接收的数据包编号的 ACK 数据包，但该 ACK 包丢失，A 未收到任何 ACK 包，继续等待从 B 传来的 ACK 包。直到 A 等待 ACK 包超时，A 重发数据包。得到 B 的正确回应，才会进入下一发包流程。

4.2 Go-Back-N 协议的功能测试与结果分析

4.2.1 无 error 无 loss 情况

设置数据包个数为 500，丢包率和错误率均为 0，数据包间隔为 5。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

1. 开始阶段（buffer 中数据包较少，不足以填满发送窗口）

```
EVENT time: 0.467849,  type: 1, fromlayer5  entity: 0
A sending:
  seq:0, ack:0 check:FFFF3232
  message: aaaaaaaaaaaaaaaaaaaa

EVENT time: 5.960295,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:0, ack:0 check:FFFF3232
  message:aaaaaaaaaaaaaaaaaaaa
B receiving:0,looking for:0
B:sending:
  seq:0, ack:0 check:FFFFBC73
```

```
  message:ACK

EVENT time: 8.038575,  type: 1, fromlayer5  entity: 0
A sending:
  seq:1, ack:0 check:FFFF2827
  message: bbbbbbbbbbbbbbbbbbbb

EVENT time: 8.459425,  type: 2, fromlayer3  entity: 0
A receiving acknum:0,looking for:0
ACK!

EVENT time: 9.439863,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:1, ack:0 check:FFFF2827
  message:bbbbbbbbbbbbbbbbbbbb
B receiving:1,looking for:1
B:sending:
  seq:0, ack:1 check:FFFFBC72
  message:ACK

EVENT time: 11.610157,  type: 1, fromlayer5  entity: 0
A sending:
  seq:2, ack:0 check:FFFF1E1C
  message: cccccccccccccccccccc

EVENT time: 14.876553,  type: 2, fromlayer3  entity: 0
A receiving acknum:1,looking for:1
ACK!
```

前期开始阶段由于 buffer 中仅有一个数据包，因此输出结果基本与 Stop-And-Wait 协议相似，两者不同点主要在于：只要窗口未满，发送端 A 在接收到 ACK 包前也可以发送新的数据包，在收到上层调用时不再判断当前是否接受上层调用，而直接将上层发送来的数据包存在 buffer 中以备后续发送。

2. 正式运行阶段（buffer 中数据包足以填满发送窗口）

```
EVENT time: 2382.109131,  type: 1, fromlayer5  entity: 0
```

```
EVENT time: 2382.551514,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:12, ack:0 check:FFFFB9AD
  message:mmmmmmmmmmmmmmmmmmmm
B receiving:12,looking for:12
B:sending:
  seq:0, ack:12 check:FFFFBC67
  message:ACK

EVENT time: 2384.762939,  type: 1, fromlayer5  entity: 0

EVENT time: 2386.840820,  type: 2, fromlayer3  entity: 0
A receiving acknum:11,looking for:11
ACK!
A sending:
  seq:3, ack:0 check:FFFF7370
  message: ttttttttttttttttttt

EVENT time: 2388.591064,  type: 1, fromlayer5  entity: 0

EVENT time: 2390.226318,  type: 2, fromlayer3  entity: 0
A receiving acknum:12,looking for:12
ACK!
A sending:
  seq:4, ack:0 check:FFFF6965
  message: uuuuuuuuuuuuuuuuuuuu

EVENT time: 2390.331299,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:13, ack:0 check:FFFFAFA2
  message:nnnnnnnnnnnnnnnnnnnn
B receiving:13,looking for:13
B:sending:
  seq:0, ack:13 check:FFFFBC66
  message:ACK
```

```
EVENT time: 2393.038818,  type: 2, fromlayer3  entity: 0
A receiving acknum:13,looking for:13
ACK!
A sending:
  seq:5, ack:0 check:FFFF5F5A
  message: vvvvvvvvvvvvvvvvvvvv
```

运行一段时间后，由于 buffer 中的数据足以填满窗口，因此每次发送端 A 在接受上层调用时，A 不会立刻将数据发送至下层，而是将数据包存入 buffer 中。在 ACK 到来时，窗口可以向后移动，会向下层发送新的数据包。

4.2.2 无 loss 只有 error 情况

设置数据包个数为 200，错误率为 0.2，丢包率为 0，数据包间隔为 25。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

4.2.2.1 数据包出现 error

```
EVENT time: 328.894867,  type: 0, timerinterrupt   entity: 0
A sending:
  seq:10, ack:0 check:FFFFCDC3
  message: kkkkkkkkkkkkkkkkkkkk
A sending:
  seq:11, ack:0 check:FFFFC3B8
  message: llllllllllllllllllll
         TOLAYER3: packet being corrupted
A sending:
  seq:12, ack:0 check:FFFFB9AD
  message: mmmmmmmmmmmmmmmmmmmm
A sending:
  seq:13, ack:0 check:FFFFAFA2
  message: nnnnnnnnnnnnnnnnnnnn
A sending:
  seq:14, ack:0 check:FFFFA597
  message: oooooooooooooooooooo
A sending:
  seq:15, ack:0 check:FFFF9B8C
  message: pppppppppppppppppppp
```

```
          TOLAYER3: packet being corrupted
A sending:
  seq:0, ack:0 check:FFFF9191
  message: qqqqqqqqqqqqqqqqqqq

EVENT time: 331.947998,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:10, ack:0 check:FFFFCDC3
  message:kkkkkkkkkkkkkkkkkkk
B receiving:10,looking for:10
B:sending:
  seq:0, ack:10 check:FFFFBC69
  message:ACK

EVENT time: 336.136047,  type: 2, fromlayer3  entity: 0
A receiving acknum:10,looking for:10
ACK!

EVENT time: 340.454895,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:11, ack:0 check:FFFFC3B8
  message:Zlllllllllllllllllll
B receiving:11,looking for:11
Corrupt!
B:sending:
  seq:0, ack:10 check:FFFFBC69
  message:ACK

EVENT time: 341.082520,  type: 1, fromlayer5  entity: 0
A sending:
  seq:1, ack:0 check:FFFF8786
  message: rrrrrrrrrrrrrrrrrrr

EVENT time: 341.797943,  type: 2, fromlayer3  entity: 1
B receiving:
```

```
  seq:12, ack:0 check:FFFFB9AD
  message:mmmmmmmmmmmmmmmmmmmm
B receiving:12,looking for:11
Wrong packet!
B:sending:
  seq:0, ack:10 check:FFFFBC69
  message:ACK

EVENT time: 342.847656,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:13, ack:0 check:FFFFAFA2
  message:nnnnnnnnnnnnnnnnnnn
B receiving:13,looking for:11
Wrong packet!
B:sending:
  seq:0, ack:10 check:FFFFBC69
  message:ACK

EVENT time: 348.737396,  type: 2, fromlayer3  entity: 0
A receiving acknum:10,looking for:11
Wrong ACK number!

EVENT time: 350.178772,  type: 2, fromlayer3  entity: 0
A receiving acknum:10,looking for:11
Wrong ACK number!

EVENT time: 436.136047,  type: 0, timerinterrupt   entity: 0
A sending:
  seq:11, ack:0 check:FFFFC3B8
  message: lllllllllllllllllll
A sending:
  seq:12, ack:0 check:FFFFB9AD
  message: mmmmmmmmmmmmmmmmmmmm
         TOLAYER3: packet being corrupted
A sending:
```

```
  seq:13, ack:0 check:FFFFAFA2
  message: nnnnnnnnnnnnnnnnnnn
A sending:
  seq:14, ack:0 check:FFFFA597
  message: oooooooooooooooooooo
A sending:
  seq:15, ack:0 check:FFFF9B8C
  message: pppppppppppppppppppp
A sending:
  seq:0, ack:0 check:FFFF9191
  message: qqqqqqqqqqqqqqqqqqqq
          TOLAYER3: packet being corrupted
A sending:
  seq:1, ack:0 check:FFFF8786
  message: rrrrrrrrrrrrrrrrrrrr
A sending:
  seq:2, ack:0 check:FFFF7D7B
  message: ssssssssssssssssssss
          TOLAYER3: packet being corrupted
```

当数据包出现 error 时，接收端 B 会将该数据包丢弃，并发送上一次正确接收数据包对应的 ACK 包。若之后接收到 A 端发送的后续数据包，由于这些数据包与当前接收端 B 期待接收的数据包不符，因此将它们全部丢弃，并发送上一次正确接收数据包对应的 ACK 包。此时由于 A 端接收到许多重复的 ACK 包，因此 A 端也直接忽视这些 ACK 包，窗口不后移，并在超时后重传窗口内所有数据包。

4.2.2.2 ACK 包出现 error

```
EVENT time: 559.064148,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:12, ack:0 check:FFFFB9AD
  message:mmmmmmmmmmmmmmmmmmmm
B receiving:12,looking for:12
B:sending:
  seq:0, ack:12 check:FFFFBC67
  message:ACK
          TOLAYER3: packet being corrupted
```

```
EVENT time: 559.324890,  type: 1, fromlayer5  entity: 0

EVENT time: 562.838013,  type: 2, fromlayer3  entity: 0
A receiving acknum:12,looking for:12
Corrupt!

EVENT time: 563.309082,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:13, ack:0 check:FFFFAFA2
  message:nnnnnnnnnnnnnnnnnnn
B receiving:13,looking for:13
B:sending:
  seq:0, ack:13 check:FFFFBC66
  message:ACK
         TOLAYER3: packet being corrupted

EVENT time: 565.563782,  type: 2, fromlayer3  entity: 0
A receiving acknum:13,looking for:12
Corrupt!

EVENT time: 566.202209,  type: 1, fromlayer5  entity: 0

EVENT time: 571.633362,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:14, ack:0 check:FFFFA597
  message:oooooooooooooooooooo
B receiving:14,looking for:14
B:sending:
  seq:0, ack:14 check:FFFFBC65
  message:ACK

EVENT time: 573.470825,  type: 2, fromlayer3  entity: 0
A receiving acknum:14,looking for:12
ACK!
```

```
A sending:
  seq:4, ack:0 check:FFFF6965
  message: uuuuuuuuuuuuuuuuuuuu
A sending:
  seq:5, ack:0 check:FFFF5F5A
  message: vvvvvvvvvvvvvvvvvvvv
A sending:
  seq:6, ack:0 check:FFFF554F
  message: wwwwwwwwwwwwwwwwwwww
```

当来自接收端 B 的 ACK 包出现错误时，A 端忽略该 ACK 包，继续等待后续 B 端传来的 ACK 包。当 A 端接收到目标序号之后的 ACK 包时，A 端通过累计应答机制，认为当前 ACK 包对应的数据包以及之前所有数据包均已成功发送。如上所示结果，编号 12 与 13 的 ACK 包被破坏，而编号 14 的 ACK 包被 A 端成功接收，因此 A 端会认为编号 12-14 的数据包均已成功发送，窗口后移 3 个单位，发送三个新数据包。

4.2.3 无 error 只有 loss 情况

设置数据包个数为 200，错误率为 0，丢包率为 0.2，数据包间隔为 25。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

4.2.3.1 数据包出现 loss

```
EVENT time: 1180.937378,  type: 0, timerinterrupt   entity: 0
A sending:
  seq:9, ack:0 check:FFFF372E
  message: zzzzzzzzzzzzzzzzzzzz
         TOLAYER3: packet being lost
A sending:
  seq:10, ack:0 check:FFFF3228
  message: aaaaaaaaaaaaaaaaaaaa
         TOLAYER3: packet being lost
A sending:
  seq:11, ack:0 check:FFFF281D
  message: bbbbbbbbbbbbbbbbbbbb
         TOLAYER3: packet being lost
A sending:
  seq:12, ack:0 check:FFFF1E12
```

```
  message: ccccccccccccccccccc
A sending:
  seq:13, ack:0 check:FFFF1407
  message: ddddddddddddddddddddd
A sending:
  seq:14, ack:0 check:FFFF09FC
  message: eeeeeeeeeeeeeeeeeeee
A sending:
  seq:15, ack:0 check:FFFFFFF0
  message: ffffffffffffffffffff
A sending:
  seq:0, ack:0 check:FFFFF5F5
  message: gggggggggggggggggggg

EVENT time: 1183.836182,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:12, ack:0 check:FFFF1E12
  message:ccccccccccccccccccc
B receiving:12,looking for:9
Wrong packet!
B:sending:
  seq:0, ack:8 check:FFFFBC6B
  message:ACK

EVENT time: 1186.239502,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:13, ack:0 check:FFFF1407
  message:ddddddddddddddddddddd
B receiving:13,looking for:9
Wrong packet!
B:sending:
  seq:0, ack:8 check:FFFFBC6B
  message:ACK
        TOLAYER3: packet being lost
```

```
EVENT time: 1186.906128,  type: 1, fromlayer5  entity: 0

EVENT time: 1188.761963,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:14, ack:0 check:FFFF09FC
  message:eeeeeeeeeeeeeeeeeeee
B receiving:14,looking for:9
Wrong packet!
B:sending:
  seq:0, ack:8 check:FFFFBC6B
  message:ACK

EVENT time: 1191.445435,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:15, ack:0 check:FFFFFFF0
  message:ffffffffffffffffffff
B receiving:15,looking for:9
Wrong packet!
B:sending:
  seq:0, ack:8 check:FFFFBC6B
  message:ACK

EVENT time: 1192.404663,  type: 2, fromlayer3  entity: 0
A receiving acknum:8,looking for:9
Wrong ACK number!

EVENT time: 1195.042358,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:0, ack:0 check:FFFFF5F5
  message:gggggggggggggggggggg
B receiving:0,looking for:9
Wrong packet!
B:sending:
  seq:0, ack:8 check:FFFFBC6B
  message:ACK
```

```
EVENT time: 1199.724243,  type: 2, fromlayer3  entity: 0
A receiving acknum:8,looking for:9
Wrong ACK number!

EVENT time: 1203.932373,  type: 1, fromlayer5  entity: 0

EVENT time: 1206.309326,  type: 2, fromlayer3  entity: 0
A receiving acknum:8,looking for:9
Wrong ACK number!

EVENT time: 1213.857422,  type: 2, fromlayer3  entity: 0
A receiving acknum:8,looking for:9
Wrong ACK number!

EVENT time: 1245.376465,  type: 1, fromlayer5  entity: 0

EVENT time: 1248.652588,  type: 1, fromlayer5  entity: 0

EVENT time: 1280.937378,  type: 0, timerinterrupt   entity: 0
A sending:
  seq:9, ack:0 check:FFFF372E
  message: zzzzzzzzzzzzzzzzzzz
A sending:
  seq:10, ack:0 check:FFFF3228
  message: aaaaaaaaaaaaaaaaaaa
A sending:
  seq:11, ack:0 check:FFFF281D
  message: bbbbbbbbbbbbbbbbbbb
         TOLAYER3: packet being lost
A sending:
  seq:12, ack:0 check:FFFF1E12
  message: ccccccccccccccccccc
A sending:
  seq:13, ack:0 check:FFFF1407
```

```
  message: ddddddddddddddddddd
A sending:
  seq:14, ack:0 check:FFFF09FC
  message: eeeeeeeeeeeeeeeeeee
         TOLAYER3: packet being lost
A sending:
  seq:15, ack:0 check:FFFFFFF0
  message: fffffffffffffffffff
         TOLAYER3: packet being lost
A sending:
  seq:0, ack:0 check:FFFFF5F5
  message: ggggggggggggggggggg
```

可以看出当数据包丢失时，B 未接收到丢失的数据包，当接收到后续数据包时便会认为接收到错误数据包，忽视这些数据包并向 A 发送上次正确接收的数据包对应的 ACK 包。A 端会收到很多重复的 ACK 包，便会忽视这些 ACK 包，直到超时后才会从丢失的数据包开始进行数据包重传。

4.2.3.2 ACK 包出现 loss

```
EVENT time: 794.269287,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:14, ack:0 check:FFFFA597
  message:ooooooooooooooooooo
B receiving:14,looking for:14
B:sending:
  seq:0, ack:14 check:FFFFBC65
  message:ACK
         TOLAYER3: packet being lost

EVENT time: 799.986938,  type: 2, fromlayer3  entity: 1
B receiving:
  seq:15, ack:0 check:FFFF9B8C
  message:ppppppppppppppppppp
B receiving:15,looking for:15
B:sending:
  seq:0, ack:15 check:FFFFBC64
  message:ACK
```

```
EVENT time: 809.518066,  type: 2, fromlayer3  entity: 0
A receiving acknum:15,looking for:14
ACK!
A sending:
  seq:6, ack:0 check:FFFF554F
  message: wwwwwwwwwwwwwwwwwwww
         TOLAYER3: packet being lost
A sending:
  seq:7, ack:0 check:FFFF4B44
  message: xxxxxxxxxxxxxxxxxxxx
```

当 B 端发送的 ACK 包丢失时，A 端无法接收到目标 ACK 包，随后 B 端接收到下一个数据包，B 端发送该数据包对应的 ACK 包，A 端接收到该 ACK 包，由于 Go-Back-N 协议使用累计应答机制，虽然与 A 端预期接收到的 ACK 包不同，但 A 端仍认为之前数据包均已成功接收，A 端随后会发送新数据包。

4.2.4 有 error 有 loss 情况

设置数据包个数为 200，错误率为 0.1，丢包率为 0.1，数据包间隔为 25。此处展示终端中的输出（其中棕色为 A 端输入信息，绿色为 B 端输出信息，此处仅展示两个数据包，更多结果见附件）。

1.数据包出现 loss，则不会出现 ACK 包，故同上述数据包出现 loss 的情况。

2.数据包出现 error，ACK 包出现 loss，故同上述数据包出现 error 的情况。

4.2.5 Go-Back-N 协议测试结果总结

测试结果可汇总为下表 1：

表 1 Go-Back-N 协议测试结果汇总

	ACK 包正常/乱序	ACK 包丢失	ACK 包错误
数据包正常	发送端接收到正常或乱序的 ACK 包，通过累计应答机制继续发送数据包；接收端接收到正确的数据包，将数据发送至上层	发送端接收到后续正确的 ACK 包（若未收到则超时重传），进行累计应答；接收端接收到正确的数据包，将数据发送至上层	发送端接收到后续正确的 ACK 包（若未收到则超时重传），进行累计应答；接收端接收到正确的数据包，将数据发送至上层
数据包乱序	发送端接收到窗口外数据包对应的 ACK 包，不进行任何操作，直至接收到正确的 ACK 包，否则超时重传；接收端接收到乱序数据包，发送上次正确接收的数据包对应的 ACK 包，若后续接收到正确的数据包，则发送该数据包对应的 ACK 包		
数据包丢失	发送端不断接收到窗口外数据包对应的 ACK 包，不进行任何操作，直至超时重传；接收端接收到乱序数据包，发送上次正确接收的数据包对应的 ACK 包		
数据包错误	发送端不断接收到窗口外数据包对应的 ACK 包，不进行任何操作，直至超时重传；接收端接收到错误数据包，发送上次正确接收的数据包对应的 ACK 包，接下来接收到乱序数据包，仍然发送上次正确接收的数据包对应的 ACK 包		

由测试结果可见，无论出现丢包或数据包损坏等情况，程序均可正确处理，并最终输出正确结果，因此可以证明本次对 Go-Back-N 协议的程序实现是可行无误的。

五、总结

通过本次实验对 Stop-and-Wait 与 Go-Back-N 协议的设计与实现，使我进一步加深了对可靠数据传输的理解，也进一步锻炼了通过 FSM 图进行协议设计的能力。在测试过程中，无论任何情况，程序均可以正确处理，并最终输出正确结果，也进一步看出本次实验中两个协议的程序实现是正确无误的。除代码实现外，报告书写也很大程度提升了我文档书写的能力。