**北京邮电大学软件学院**

**2019-2020 学年第 1学期实验报告**

**课程名称： 通信协议软件设计\_\_\_\_\_\_\_\_**

**实验名称： 基于SPIN的通信协议验证\_\_\_\_\_\_\_**

**实验完成人：**

**姓名： 宋振铭 学号： 2017211861**

**姓名： 陈凌云 学号： 2017211868**

**姓名： 吴志镛 学号： 2017211869**

**姓名： 赵景煜 学号： 2017211880**

**姓名： 陆琦伟 学号： 2017211881**

**姓名： 薛子豪 学号： 2017522041**

**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_ \_\_**\_\_雷友珣**\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2019 年 11 月 30 日**

**小组报告一：**

1. **实验目的**

通过本实验使学生理解协议验证的意义、理解通信协议性质、掌握基于PROMELA语言的协议建模和验证。

1. **实验任务**

基于Linux操作系统，安装基于PROMELA语言的协议验证工具SPIN，并对所选择的通信协议使用PROMELA语言建模，使用PROMELA语言描述需要验证的协议性质，使用协议验证工具SPIN对协议的PROMELA模型进行验证。

1. **实验内容**
2. 下载协议验证工具SPIN软件包，并在Linux操作系统环境下安装SPIN软件。
3. 对所选定的通信协议进行分析，理解协议机制和工作原理，用PROMELA语言对该协议进行建模。
4. 选择需要验证的协议性质：系统不变性（Invariant）、无死锁（Deadlock）、无活锁（Livelock）。用PROMELA语言的断言（assertion）、标签（end、progress、accept标签）、never声明或线性时态逻辑（LTL）表达式来描述需要验证的协议性质。
5. 使用SPIN工具对协议进行验证。
6. **实验环境**
7. Linux系统主机；
8. SPIN软件6.5.0版本，软件下载网址：

<https://github.com/nimble-code/Spin>

<http://spinroot.com/spin/Src/index.html>

**五、 小组分工**

陈凌云：设计协议实现过程，使用promela语言进行实现发送方和接收方

吴志镛：使用promela语言实现接收方和发送方，并添加验证语句

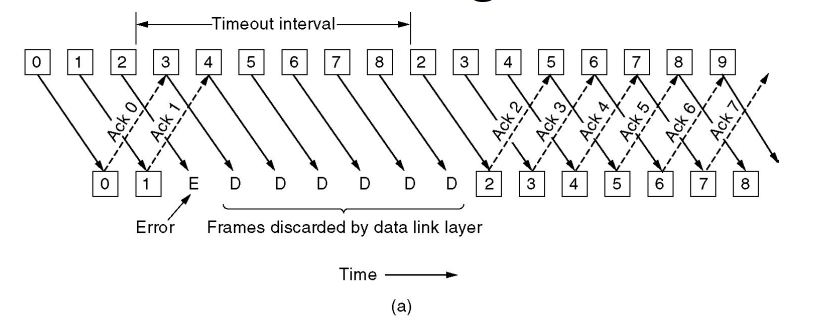
赵景煜：设计协议验证内容，使用ispin软件验证协议，对出现的程序错误进行调试

**六、 实验过程详述**

## 1.选择协议：

**本次实验我们选择的协议与上次实验相同，也是回退N协议，下面简单的讲述回退N协议的独特之处：**

其不同于 onebit 的sliding window 协议 ，发送消息之后并不会等待接收方的响应之后才发送下一帧，而是持续地向接收方发送消息。如果中途有一帧对方没有接收到，则之后发送的帧不会被接收方接受，直到没有接收到响应的帧的timer到时才将此帧重发。详细图如下：

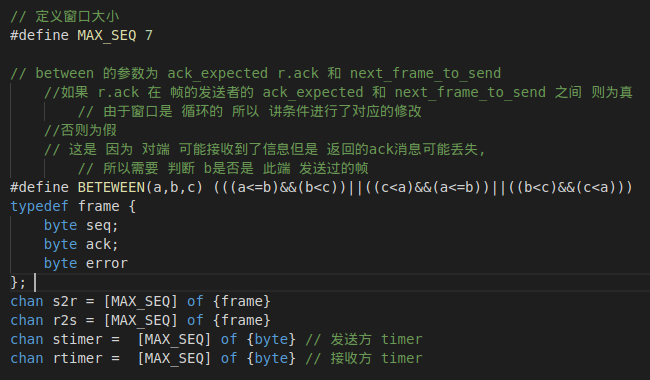


## 2.PROMELA模型建模：

**2.1. 预定义 和 类型定义：**

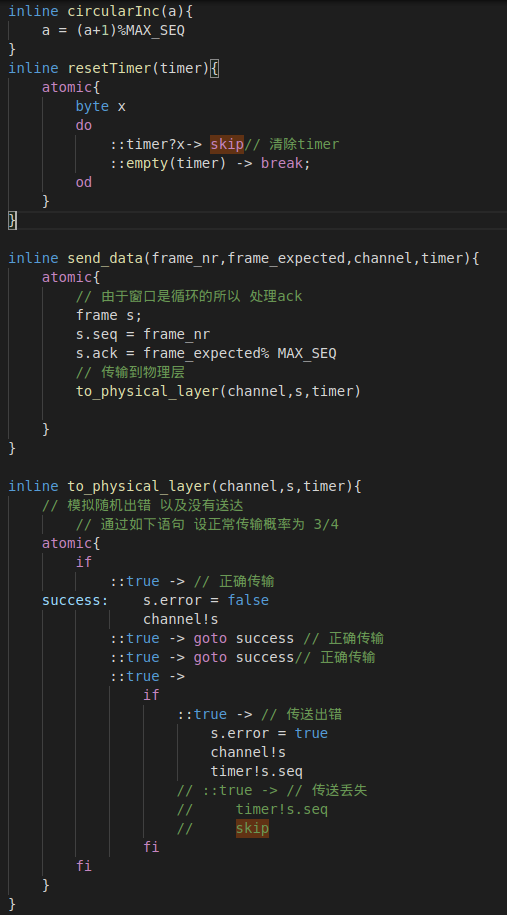
**首先我们定义我们的帧结构，窗口大小，两个定时器，发送方窗口**

**和接收方窗口其中：（详细见下图）**

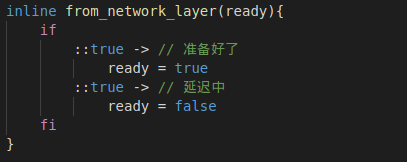
1. frame 为帧的结构，其中 error 表示帧丢失 或是 帧校验出错
2. s2r r2s 表示发送和接收端各自的窗口
3. stimer 和 rtimer 表示 发送和接收端各自的 计时器

## 2.2. 初始化函数和功能函数

定义好的我们所需的数据之后，我们需要一些初始话函数和功能函数来服务哦我们的两个发送方进程和接收方进程，为了便于我们在Spin中进行结果的感测，函数基本上使用 atomic 作为原子操作，我们的初始化都采用incline进行说明，进行分析，我们发现，需要一个circularInc函数来初始化初始化窗口的帧号，resetTimer重新设定计时器信号，需要send\_data函数发送数据，还需要模拟物理层传输的函数，这些函数的详细实现如下:

* 1. 1.circularInc函数：于窗口是循环的 所以 inc 也使用取模的方法循环方式递增。
  2. resetTimer函数通过判定timer的时间清楚原来计时并重新设定
  3. send\_data将帧结构中的数据打包好进行数据的发送，传送到物理层
  4. to\_physical\_layer 模拟 物理层的传输
     1. 为了方便，对于正确的传输，我们认为其不需要计时，便于对流程的观察
     2. 这里为了提高正确传输率，顾使用了多个 goto success
     3. 此外 为了能够在 生成msc图中观察到传送的丢失，我们讲 error 同时表示 丢失 和 出错两种情况
  5. from\_network\_layer

用于模拟 网络层传输数据的 延迟



## 2.3. 实现进程：

## 2.3.1 SENDER：

## 大致过程可以描述如下：

1. 变量的声明（定义计时器，缓冲区，下一个需要发送的帧，接收的ack等信息）
2. 运转逻辑

1)模拟网络层的延迟，在满足条件的情况下 从网络层获取信息，

1. 帧到达，即相当于 from\_physical\_layer 有来自物理层的信息
2. 在无错误的情况下则对帧进行处理，改变进程的状态
3. 计时器到达

1）注意这里我们没有直接使用 promela 自带的 timeout 因为其要求完全无进程可执行的时候才会触发，与我们现在模拟回退N协议的进程不太适合

2）这里使用自定义计时器，通过 if 的随机选择来模拟计时器的随机触发

## 发送者的进程实现代码和详解如下：

proctype slwGBNSender(){

byte next\_frame\_to\_send = 0;

byte ack\_expected = 0;

byte nbuffered = 0;byte timeoutAck;

byte frame\_expected = 0// 当前的缓冲区的帧数量

frame r;/\*接受数据的帧\*/bool networkMsg = false; /\*网络层是否准备好了\*/

chan topeer,peer;topeer = s2r;peer=r2s;

do

// 模拟网络层的 延迟

// 这里要求 只有 sender 端的 networklayer 才有信息

::empty(peer) ->

if

::(!networkMsg && nbuffered<MAX\_SEQ) ->

atomic{

from\_network\_layer(networkMsg)

}

// 来自网络层的信息

::networkMsg->

atomic{

nbuffered ++

//发送信息

send\_data(next\_frame\_to\_send,frame\_expected,topeer,stimer)

circularInc(next\_frame\_to\_send)

networkMsg = false

}

::else -> skip

fi

// 帧到达

::peer?r->

atomic {

if

::r.error // 消息发生错误

skip; // 直接忽略操作 当做是没有接受到

::else ->

if

::(r.seq == frame\_expected)-> //预期帧 直接获取

// to\_network\_layer

// 作为 接收方 则要 发送ack 信息

circularInc(frame\_expected);

::else -> skip

fi

printf("actexp = %d,nfts = %d",ack\_expected,next\_frame\_to\_send)

do // 处理 ack

// 之所以 只要 让 r.ack 在 ack\_expected 和 next\_frame\_to\_send之间

// 是因为 对端必须 获得到指定的 seq 帧才能返回 ack

::BETEWEEN(ack\_expected,r.ack,next\_frame\_to\_send) ->

nbuffered -- // 消息传送成功

if

::(stimer??[eval(ack\_expected)])->

stimer??eval(ack\_expected)

::else ->skip

fi

circularInc(ack\_expected);

:: else -> break

od

fi

}

// 存在计时器

::stimer?<timeoutAck> ->

atomic{

if

::stimer?timeoutAck-> //超时

resetTimer(stimer) // 重置计时器队列

next\_frame\_to\_send = ack\_expected

int i;

for ( i : 1 .. nbuffered ){

send\_data(next\_frame\_to\_send,frame\_expected,topeer,stimer)

circularInc(next\_frame\_to\_send)

}

::true -> skip // 未超时

fi

}

od

}

## 2.3.2 RECEIVER：

## 大致过程可以描述如下：

1. 变量的声明（定义计时器，缓冲区，需要接收的帧，返回的ack等信息）
2. 帧到达的处理

基本上同发送者的逻辑

不同的地方在于接受到 需求的帧之后 会立刻向发送端 发送ack 消息

1. 计时器

由于 发送者和 接受者同时使用计时器和 只有发送者启用计时器效果一致，所以这里进行注释让 msc图更直观

## 接收者的进程实现代码和详解如下：

proctype slwGBNReceiver(){

byte next\_frame\_to\_send = 0;byte ack\_expected = 0;byte frame\_expected = 0

byte nbuffered = 0/\*当前的缓冲区的帧数量\*/

byte timeoutAck;

frame r;

chan topeer,peer

//sender->topeer = s2r;peer=r2s;

topeer = r2s;peer=s2r;

do

// 帧到达

::peer?r->

atomic {

if

::r.error // 消息发生错误

skip; // 直接忽略操作 当做是没有接受到

::else ->

do // 处理 ackS

::BETEWEEN(ack\_expected,r.ack,next\_frame\_to\_send) ->

nbuffered -- // 消息传送成功

if

::(rtimer??[eval(ack\_expected)])->

rtimer??eval(ack\_expected)

::else ->skip

fi

circularInc(ack\_expected);

:: else -> break

od

if

::(r.seq == frame\_expected)-> //预期帧 直接获取

// to\_network\_layer

// 作为 接收方 则要 发送ack 信息

nbuffered ++

//发送信息

send\_data(next\_frame\_to\_send,frame\_expected,topeer,rtimer)

circularInc(next\_frame\_to\_send);

circularInc(frame\_expected);

::else -> skip

fi

fi

}

// 超时

// 存在计时器

// ::rtimer?<timeoutAck> ->

// atomic{

// if

// ::rtimer?timeoutAck-> //超时

// resetTimer(rtimer)

// next\_frame\_to\_send = ack\_expectedS

// int i;

// for ( i : 1 .. nbuffered ){

// send\_data(next\_frame\_to\_send,frame\_expected,topeer,rtimer)

// circularInc(next\_frame\_to\_send)

// }

// ::skip // 未超时

// fi

// }

od

}

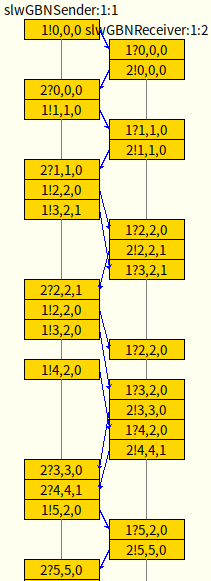
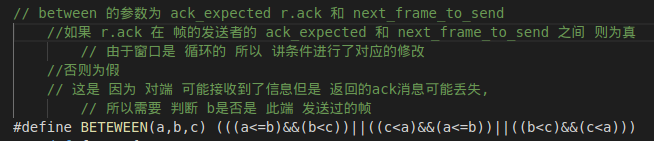
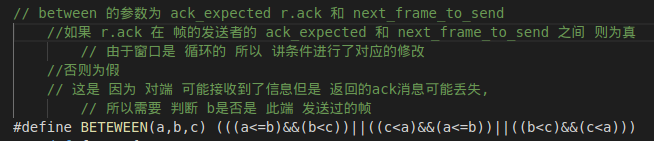
## 3. 模型模拟

使用的模拟为随机模拟，seed = 100

如下为一个包含计时器到时重发步骤的 msc图:  
其中 信道 1 为 发送方 的窗口

信道 2 为 接收方的窗口

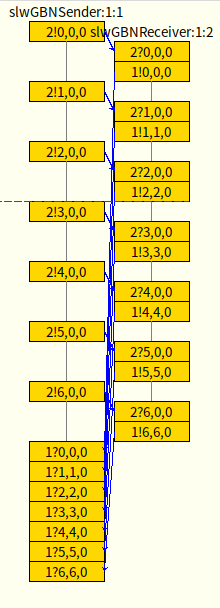
计时器的信道 经过过滤 没有显示在 mvc图中

1. 其中 信号的 消息 为 信道?seq,ack,error
2. sender 和 receiver 初始化完毕 开始信息的交互
3. receiver 接受到 sender 发送的信息 并且满足要求 所以接受并发送回 ack
4. 左部发送方 连续发送了 1!2,2,0 和 1!3,2,1,验证了发送方发送消息不必等待接收方的ack验证
5. 发送方1!3,2,1 的 error 位置1 声明传输错误， 此外接收方对成功接收到的 1?2,2,0 即第二帧 返回的 ack 信息 发生了错误(2!2,2,1)
6. 由于没有接收到 第二帧的ack ，发送方的计时器到时，从第二帧开始重发消息
7. 重复的消息没有发生错误，发送方正确接收
8. 都是接收方回应的 第四帧确认消息发生错误(2!4,4,1)
9. 而此时发送方继续发送了第五帧
10. 接收方接收到第五帧, 由于第四帧成功接收到了，所以可以返回第五帧的确认消息，并且返回没有错误
11. 发送正确接收 开启下一次消息的发送（这是因为 接收消息的时候 使用了 BETWEEN 宏定义条件）如下：

## 4. 协议验证

## 4.1. 安全性检查(死锁检测)



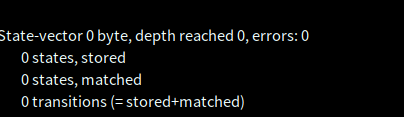
通过安全性检验 发现存在死锁:

1. 死锁的来由是 因为 在 发送 接收 1?0,0,0 的时候 无法 通过 BEWTEEN函数 的条件（ack\_expected == r.ack == next\_frame\_to\_send）:
   1. 而原因是因为 当 发送方 从网络层获取的数据达到窗口上限 ，并且还未收到 接收方的任何 响应 ack 时。接收方就认定下一个要发送的帧为 窗口的第一帧 。
   2. 解决方法：

允许 next\_frame\_to\_send 大小 等于 MAX\_SEQ ,但是在 真正使用 next\_frame\_to\_send的时候 要在使用前讲其 % MAX\_SEQ 即进行窗口位移操作

在信道中有消息的时候优先进行消息的处理而非 去 获取网络层的数

1. 修复后：



## 4.2. 活性检查:

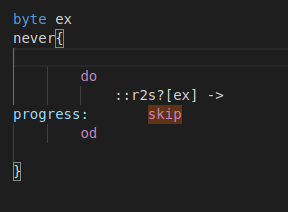
结果分析，程序的运行是否取得进展取决于 接收方是否接收到了指定的信息。

也可以转化为 接收方 是否向发送方 发出了ack 信息。

也就是说 如果程序中存在 接受法永远不会想发送方发送 ack信息的情况的话那么 就是不正常的情况：

### 4.2.1 设计：

使用 progress 标记来标记 接收方向发送方 发送的语句

—— 都是由于atomic 内部不能使用 progress 所以 考虑 使用 never 和 progress组合 来识别 r2s 的信道的状态 ：

如左图， 我们利用 never 自动和 其他进程异步 的特性 标注了 r2s?[ex] 即 接收方向发送方发送数据的信道不为空的情况为 progress。

这样就可以 使用 verify 来判定系统中存不存在 r2s 持续为空的情况

### 4.2.2 验证结果

可以看到 勾选了 none-progress cycles 和 user claim 之后 verifier 没有找到错误

**六、实验心得**

通过这次实验，对PROMELA语言验证协议有了更好的理解，对于上课一些没有听懂的PROMELA一些语句语义在实验中我们也有了更好的了解，同时对于SPIN的工具的使用也熟悉了一些，只是感觉这个实验不太适合小组完成，反倒是比较适合单独做，因为小组两个人完成都感觉有些多，有了很多重复的工作，最后和每人单独做一遍差不多。

**小组报告二：**

1. **实验目的**

通过本实验使学生理解协议验证的意义、理解通信协议性质、掌握基于PROMELA语言的协议建模和验证。

1. **实验内容**

1) 下载协议验证工具SPIN软件包，并在Linux操作系统环境下安装SPIN软件。

2) 对所选定的通信协议进行分析，理解协议机制和工作原理，用PROMELA语言对该协议进行建模。

3) 选择需要验证的协议性质：系统不变性（Invariant）、无死锁（Deadlock）、无活锁（Livelock）。用PROMELA语言的断言（assertion）、标签（end、progress、accept标签）、never声明或线性时态逻辑（LTL）表达式来描述需要验证的协议性质。

4) 使用SPIN工具对协议进行验证。

1. **实验环境**
2. Linux系统主机；
3. SPIN软件6.5.0版本，软件下载网址：

<https://github.com/nimble-code/Spin>

<http://spinroot.com/spin/Src/index.html>

1. **人员分工**

宋振铭：使用Premela语言实现滑动窗口协议

陆琦伟：使用Premela语言实现滑动窗口协议

薛子豪：设计 LTL 语句 并使用ispin 软件对协议进行验证

1. **协议设计**

# 发送方

我们通过在发送方进程中定义三个Progress实现发送方的不同功能

**Progress1**

在进程开始时初始化发送方，或在发送超时时重置发送方状态，从未收到验证的包处开始重发。

**Progress2**

Progress2对发送窗口未满和发送窗口已满两种情况分别进行处理。

当发送窗口未满时，向接收方发送数据包并模拟发送数据包时可能遇到的三种情况。

若正确发送，则发送的下一帧后移，窗口大小减一。若发送出错，则发送数据类型为错误的数据包并将发送的下一帧后移，窗口大小减一。若发送数据包丢失，则直接将发送的下一帧后移，窗口大小减一。每次发送数据包后，随机选择是否继续发送窗口内的数据包或停止发送。

**Progress3**

Progress3负责接收应答的数据包。

当收到错误的应答时，忽略收到的数据包。当收到正确的应答时，收下报文并确认报文，重置window, ack\_expected, 的值。当进入Progress3 后未能进入以上两个分支，则认为发送报文超时，回到Progress1进行重发。

# 接收方

接收方会对接收到的数据包进行验证，并根据情况返回验证消息。

若接收到的报文为期望报文，则向发送方返回当前收到包的验证消息并更新last\_expected和frame\_expected的值。若收到的报文非期望报文，则向发送发发送上一个收到的数据包所对应的验证消息，该消息可能是正确的，也可能是错误的。若收到的报文是错误的，则直接将其丢弃。

1. **实验结果**

本小组在此次实验中使用Premela语言实现了回退N帧的滑动窗口协议。

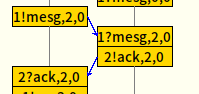
我们使用spin软件对协议进行模拟，模拟部分的结果如下。其中共有正确发送，发送方丢包，发送方出现错误，接收方丢包，接受方出现错误共5种情况。

# 协议模拟

下图中每个小方块表示发送消息的操作。例如标有 1!mesg,2,0 的小方块表示该消息由信道1送出，消息类型为mesg,消息的seq为2，ack为0 。图像左边的部分表示发送方的状态，图像右边的部分表示接收方的状态。

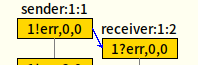
## 正确发送

如下所示，发送方向接收方发送数据包，该数据包seq为2，ack为0。接收方收到数据包后向发送方回送确认消息，该消息seq为2，ack为0。



## 发送方出现错误

如下所示，发送方发送数据包时出现错误，接收方不接受该数据包。发送方继续发送其它数据包。当之前发送出现错误的数据包因没有收到确认消息而超时时重发该数据包。



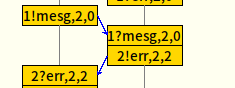
## 接收方丢包

如下所示，接收方收到发送方发送的数据包之后送回的验证消息丢失。发送方继续发送其它数据包，当之前已发送的数据包因没有收到确认消息而超时时重发该数据包。

C:\Users\LG\AppData\Local\Temp\BNZ.5dde48401960f32d\接收方丢包.PNG

## 接收方出现错误

如下所示，接收方收到发送方发送的数据包之后送回的验证消息错误。发送方收到该数据包后继续发送其它数据包，当之前已发送的数据包因没有收到正确的确认消息而超时时重发该数据包。

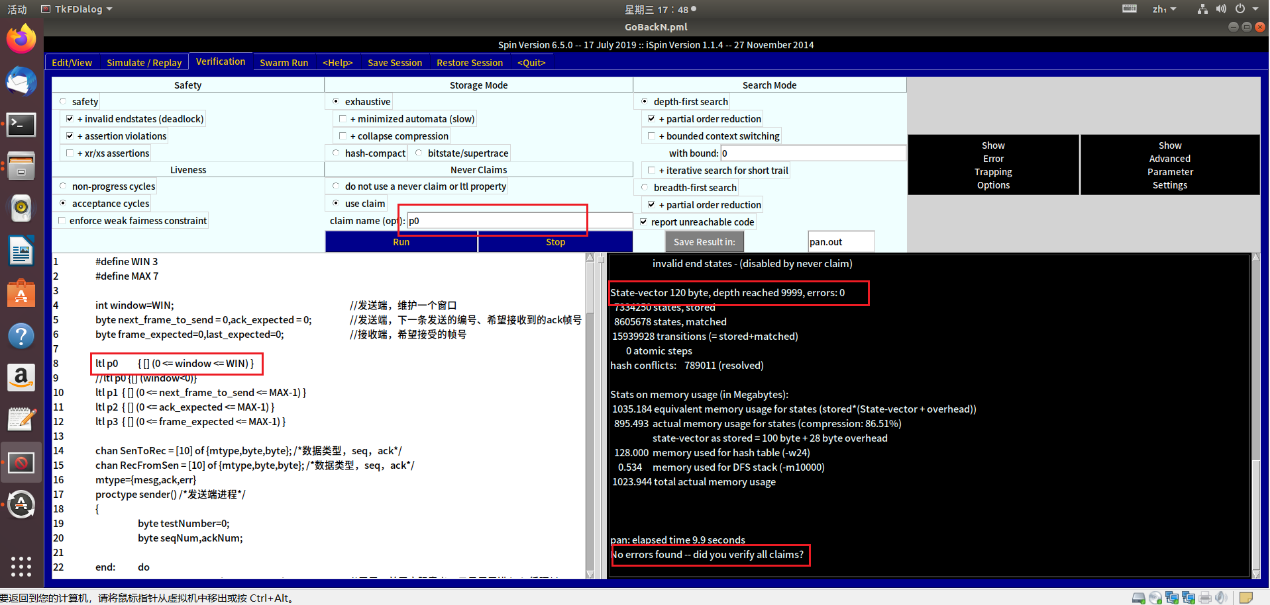


# 协议验证

我们分别使用的5条LTL语句对所实现协议的性质进行验证，得到如下结果

## 窗口大小在设定范围内

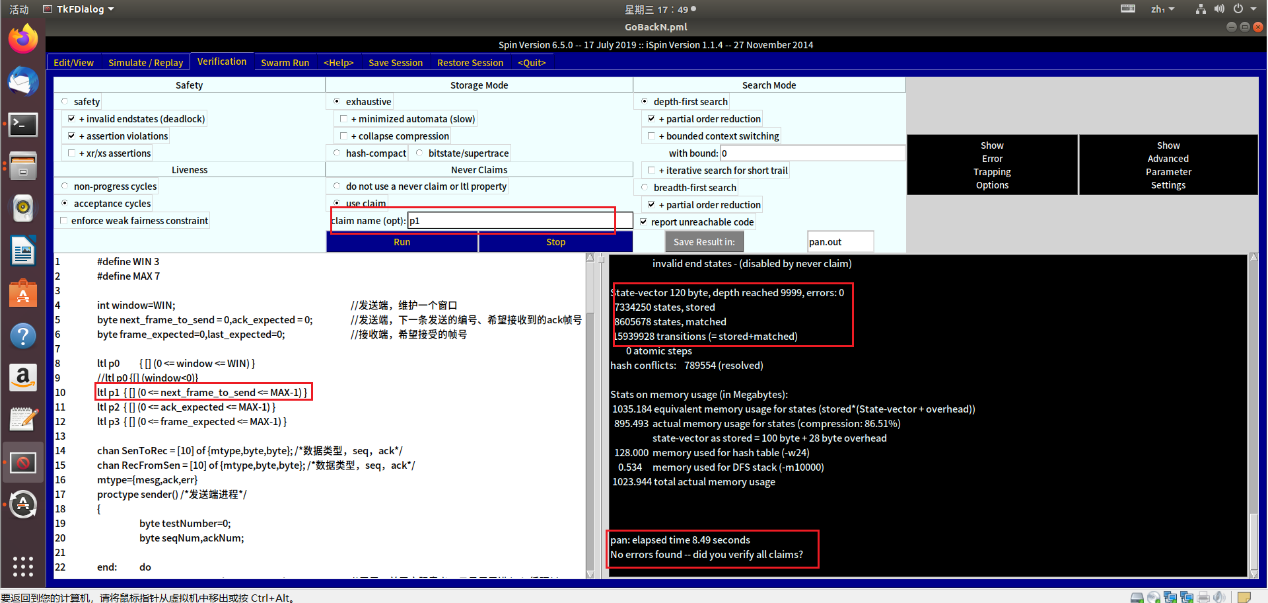
ltl p0 { [] (0 <= window <= WIN) }



No Errors found. 验证结果符合预期

## 发送方下一次发送帧的帧号在设定范围内

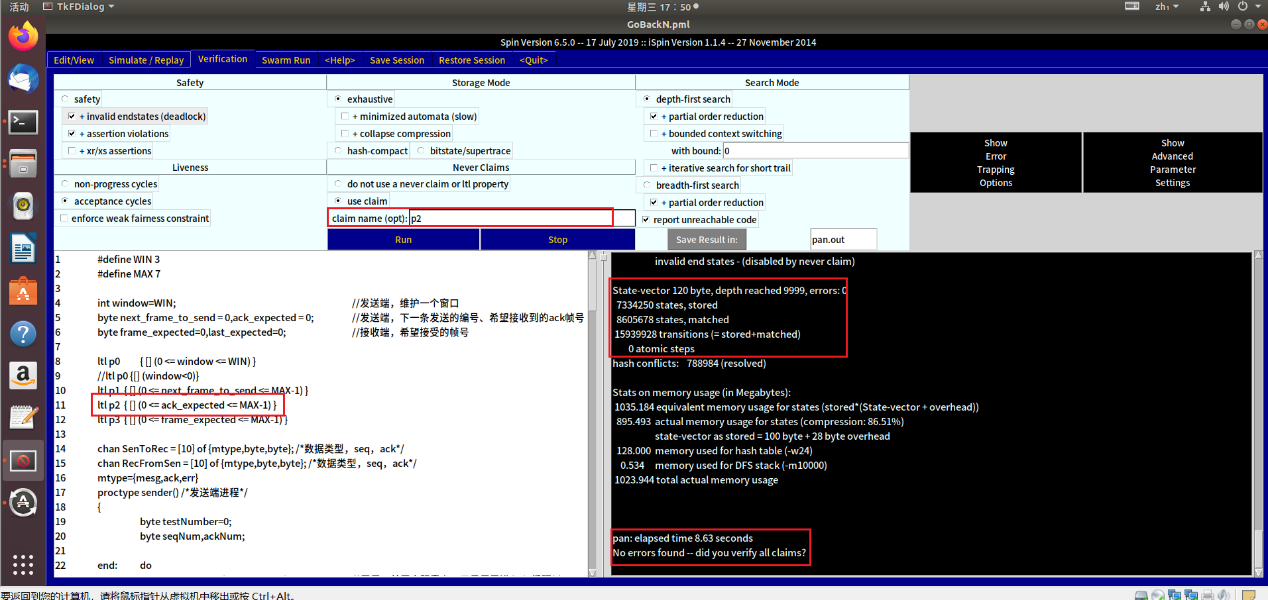
ltl p1 { [] (0 <= next\_frame\_to\_send <= MAX-1) }



No Errors found. 验证结果符合预期。

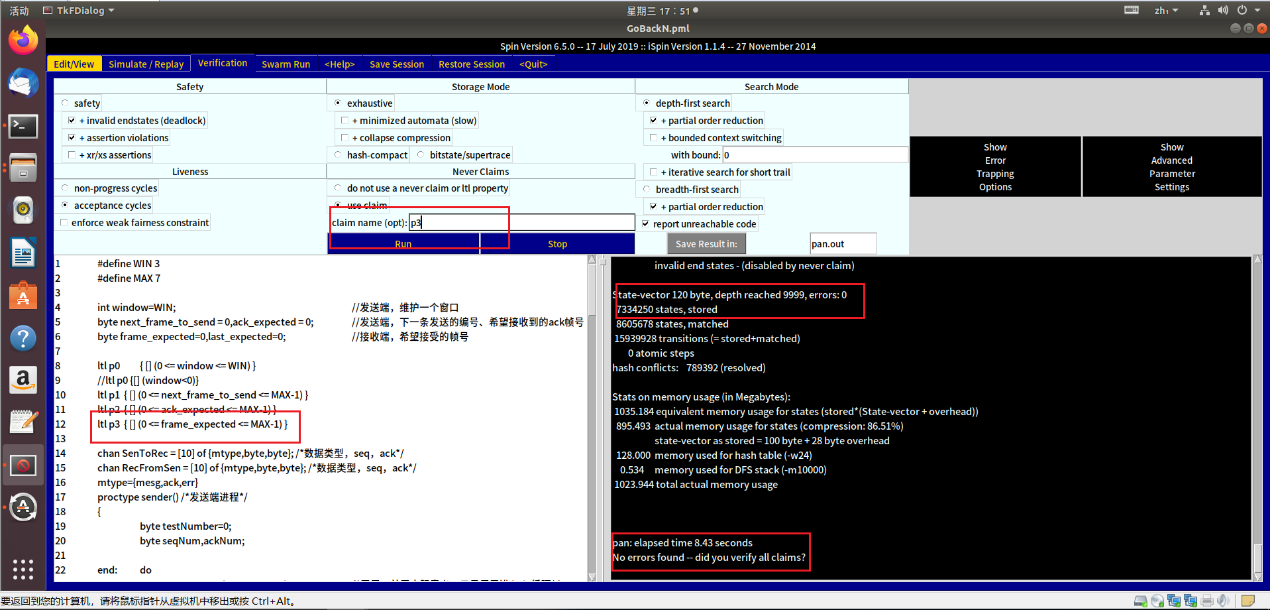
## 发送方下一次接收帧的ack值在设定范围内

ltl p2 { [] (0 <= ack\_expected <= MAX-1) }



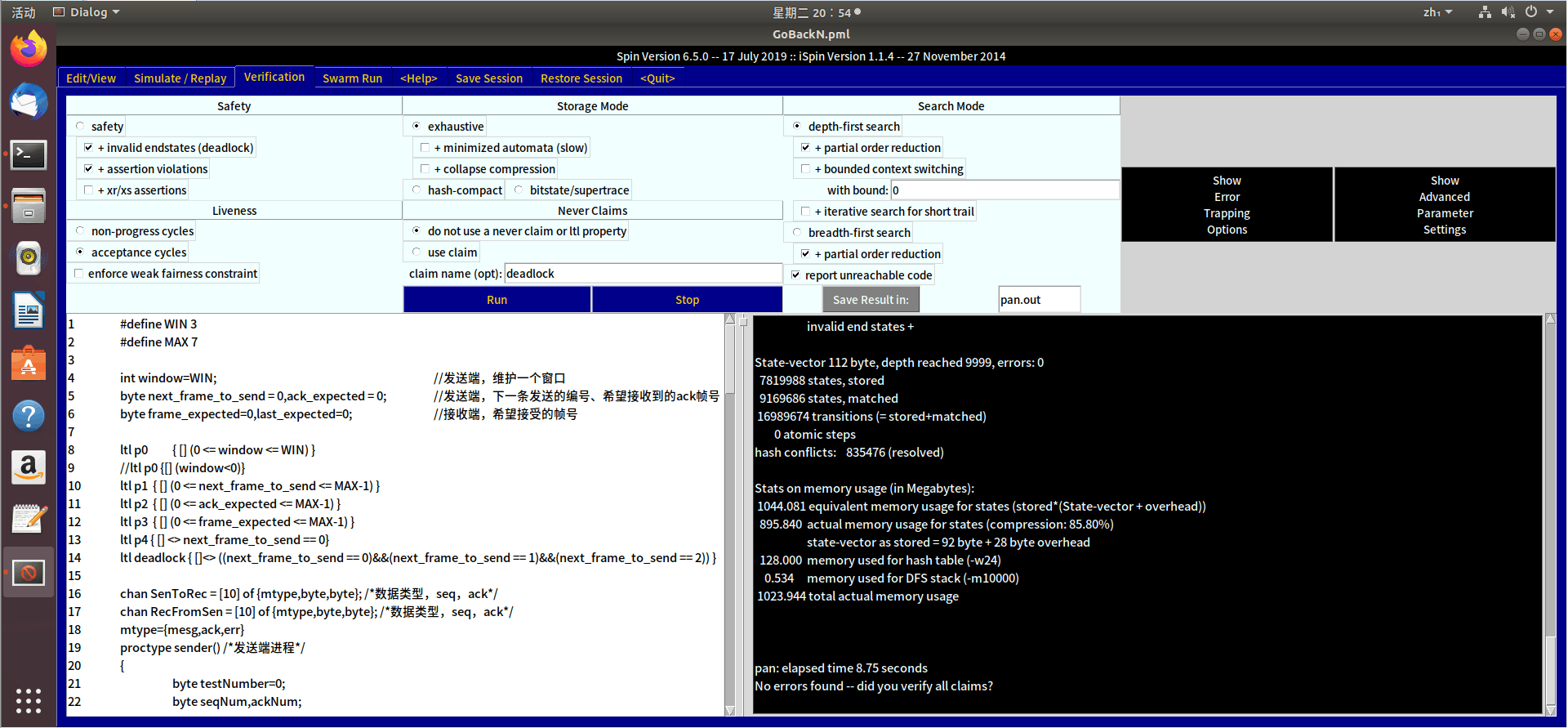
No Errors found. 验证结果符合预期。

## 接收方期待接收的帧号在设定范围内

ltl p3 { [] (0 <= frame\_expected <= MAX-1) } 

No Errors found. 验证结果符合预期。

## 协议不会产生死锁

ltl deadlock { ( ([]<>(next\_frame\_to\_send == 0))&&  ([]<>(next\_frame\_to\_send == 1))&& ([]<>(next\_frame\_to\_send == 2))) } 

No Errors found. 验证结果符合预期。

1. **总结和心得**

在本次实验中，我们学习到了如何使用Promela语言描述并设计通信协议，以及如何使用ispin软件对Promela 程序进行调试和模型验证。通过对回退N帧协议的模拟和验证，我们对通信协议的实现有了更深层次的了解。

1. **附录**

PREMELA程序源代码：

#define WIN 3

#define MAX 7

int window=WIN; //发送端，维护一个窗口

byte next\_frame\_to\_send = 0,ack\_expected = 0; //发送端，下一条发送的编号、希望接收到的ack帧号

byte frame\_expected=0,last\_expected=0; //接收端，希望接受的帧号

// 线性时态逻辑 验证协议的不同性质

ltl p0 { [] (0 <= window <= WIN) }

//ltl p0 {[] (window<0)}

ltl p1 { [] (0 <= next\_frame\_to\_send <= MAX-1) }

ltl p2 { [] (0 <= ack\_expected <= MAX-1) }

ltl p3 { [] (0 <= frame\_expected <= MAX-1) }

ltl deadlock { ( ([]<>(next\_frame\_to\_send == 0))&&  ([]<>(next\_frame\_to\_send == 1))&& ([]<>(next\_frame\_to\_send == 2))) }

chan SenToRec = [10] of {mtype,byte,byte}; /\*数据类型，seq，ack\*/

chan RecFromSen = [10] of {mtype,byte,byte}; /\*数据类型，seq，ack\*/

mtype={mesg,ack,err}

proctype sender() /\*发送端进程\*/

{

byte testNumber=0;

byte seqNum,ackNum;

end: do

:: testNumber=testNumber; /\*无用，并无实际意义，只是用于进入do循环\*/

progress1:

next\_frame\_to\_send = ack\_expected;

window=WIN

progress2:

if

::window<1-> //剩余窗口小于1时，跳过发送准备接受ack

goto progress3

::else->skip

fi

if

::SenToRec!mesg,next\_frame\_to\_send,0-> /\*正确发送\*/

next\_frame\_to\_send=(next\_frame\_to\_send+1)%MAX;/\*后移\*/

window=window-1; //每次发送过后窗口大小减一

if

::goto progress2;/\*窗口内连续发送\*/

::skip /\*不发送\*/

fi

::SenToRec!err,next\_frame\_to\_send,0-> /\*发送出错\*/

next\_frame\_to\_send=(next\_frame\_to\_send+1)%MAX;/\*后移\*/

window=window-1;

if

::goto progress2;/\*窗口内连续发送\*/

::skip /\*不发送\*/

fi

::skip-> /\*报文丢失\*/

next\_frame\_to\_send=(next\_frame\_to\_send+1)%MAX;/\*后移\*/

window=window-1;

if

::goto progress2;/\*窗口内连续发送\*/

::skip /\*不发送\*/

fi

fi

progress3:

if

:: RecFromSen?err,seqNum,ackNum -> /\*收到错误报文忽略\*/

skip

:: RecFromSen?ack,seqNum,ackNum -> /\*正确应答\*/

if //当收到的ack帧号在循环中间时，收下报文

::(ack\_expected<=ackNum & ackNum<next\_frame\_to\_send)->

ack\_expected=(ackNum+1)%MAX;

window=window+1;

window =( (window>WIN)-> WIN:window)

goto progress2;

::(ack\_expected<=ackNum & next\_frame\_to\_send <ack\_expected)->

ack\_expected=(ackNum+1)%MAX;

window=window+1;

window =( (window>WIN)-> WIN:window)

goto progress2;

::(next\_frame\_to\_send<ack\_expected & ackNum<next\_frame\_to\_send)->

window=window+1;

ack\_expected=(ackNum+1)%MAX;

window =( (window>WIN)-> WIN:window)

goto progress2;

::else->skip

fi

::timeout->goto progress1 ;/\*超时，报文重发\*/

fi

od

}

proctype receiver() /\*接收端进程\*/

{

byte seqNum,ackNum;

end: do

::SenToRec?mesg,seqNum,ackNum->

if

::(seqNum==frame\_expected)-> /\*收到报文为期望报文\*/

last\_expected=frame\_expected;

frame\_expected=(frame\_expected+1)%MAX;

if

::RecFromSen!ack,seqNum,seqNum;

::RecFromSen!err,seqNum,seqNum;

::skip;

fi

::(seqNum!=frame\_expected)-> /\*无效报文\*/

if

::RecFromSen!ack,seqNum,last\_expected;

::RecFromSen!err,seqNum,last\_expected;

::skip

fi

fi

::SenToRec?err,seqNum,ackNum-> /\*错误报文直接丢弃\*/

if

//::RecFromSen!ack,seqNum,last\_expected;

//::RecFromSen!err,seqNum,last\_expected;

::skip

fi

od

}

init

{

run sender();

run receiver();

}