**北京邮电大学软件学院**

**2020-2021学年第一学期实验报告**

**课程名称： 通信协议软件设计**

**实验名称： 基于SDL的通信协议描述**

**项目完成人：**

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

**日 期： 2020年 11 月 6 日**

# 实验目的

通过本实验使学生理解通信协议形式化描述的意义、掌握SDL描述语言及其开发工具。

# 实验内容

1. 下载、安装SDL语言开发环境PragmaDev Studio。
2. 选定一通信协议（如滑动窗口协议）进行分析。
3. 使用PragmaDev Studio对选定的协议进行SDL描述，包括该协议的SDL系统图、功能块图、进程图。若所选定的协议提供单工通信服务功能，需要分别对协议发送方协议实体和接收方协议实体均给出SDL系统描述。
4. 使用PragmaDev Studio对所描述的协议进行模拟分析。

# 实验环境

1. Windows系统主机或Linux系统主机；
2. PragmaDev Studio软件；

# 协议SDL描述

本小组共实现了协议4(1位滑动窗口协议)、协议5（回退N协议）、协议6（选择重传协议）共三个传输协议。

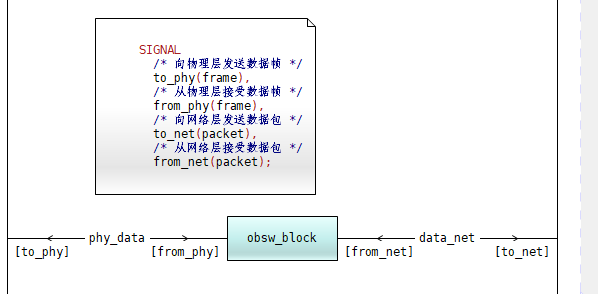
其中为实现模块重用和条理的清晰性使用了过程

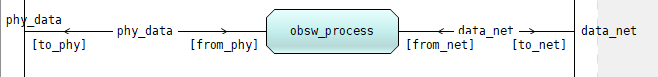
* 1. 1位滑动窗口协议：
     1. 协议描述
        1. SDL系统图

##### 系统中的数据定义（part1）



##### 系统功能块与外界交互（part0）

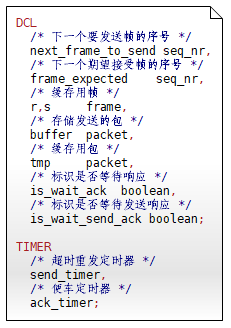
1.  Signal 为来自网络层/物理层，通向网络层/物理层的四个信号，代表传输数据的流向。
2. 整个系统包含一个功能块obsw\_block，其连接两个双向信道 data\_net(网络层数据连路层通道) 和 phy\_data(数据链路层物理层通道)，信道上传输对应的Singal。
   * + 1. SDL功能块



obsw\_block功能块中仅包含一个进程块obsw\_process，其直接与环境进行通信。

* + - 1. SDL进程块

##### 进程中的数据定义：



##### SDL语句描述：

###### 初始化进程：

启动后，首先将数据进行初始化。进入ready状态。

###### 等待发送和接受

本协议实体可作发送方也可以作接收方，ready0为目前没有传输的状态（即没有处于发送后等待确认状态，也没有处于接受数据后等待发送确认信息的状态）。可以接受来自网络层和物理的数据。

###### 发送数据类型数据帧：

ready状态是发送方相关的状态，当主动发送数据时（无论是否也担任接收方角色），即网络层有数据传过来时，都会进入这段ready到loop的流程中。如果在ready0状态接受了网络层的数据，则会通过to\_send跳转过来。

之后协议实体便构建数据类型的帧，发往物理层。并取消捎带确认计时器、设置超时计时器。同时设置两个标识。is\_wait\_ack标识置为true用来标识协议实体等待确认，is\_wait\_send\_ack标识置为false用来标识协议实体不再等待发送确认消息（如果发送之前协议实体也在等待发送ack确认消息，使发送的数据帧携带ack消息，也做到了确认（即捎带确认））。

###### 接收处理数据帧：

loop状态是接收方相关状态（同时也担任处理确认消息的作用）。进入这段loop状态后处理接收数据帧有两种方式：在ready状态发送完数据类型的帧后，以及直接从ready0接收物理层来帧后通过to\_receive调换过来。

首先进行判断来的数据帧是否为数据类型的。是则再判断是否为期望数据帧。如果是则进一步检查数据是否无误。无误则将帧中数据取出送往网络层并移动发送方的滑动窗口。如果数据帧有错，则直接舍弃等待发送方重传，此时应根据是否在等待确认消息进入不同状态（等待进loop，不等进ready0）；如果接收数据不为期望的数据，则跳过进一步检查和之后的送到网络层的步骤。

之后两种情况：协议实体在等待或不等待确认消息。如果不等待确认则不用再检查数据帧的ack值，直接跳转到to\_ack处发送确认消息并在之后进入ready0状态。如果在等待确认，则设置捎带确认定时器，以及等待发送确认标识为真。接着检查确认消息。当数据帧为ack类型时，如果现在还在等待确认消息则继续检查确认消息，否则回到ready0状态。

检查相应消息时，如果数据帧中ack字段等于next\_frame\_to\_send说明上一次发送的数据对方已经收到了，则取消重发定时器，移动接受窗口，将等待确认标识设为假。接着根据是否需要发送确认消息进入不同的状态（需要进ready，不需要进ready0）。如果ack字段不等于next\_frame\_to\_send，说明需要重传一次上一次发送的数据（因为可能是上次发送的数据帧接收方没有收到）。

###### 超时重传与流量控制：

在loop状态等待接受确认信息时，如果超时定时器被触发时，则进入to\_send重新发送数据帧。如果网络层又来新的数据，则回复网络层错误，请等待。

###### 单独发送确认：

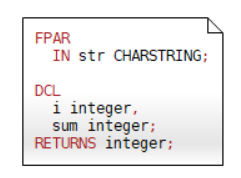
在ready状态，当捎带确认定时器到时间后（或是从to\_ack跳转过来需要直接确认时），不再等待搭发送数据类型数据帧的便车，而是单独构造确认类型的确认帧并发送。同时将需要发送确认的标识设置为false，进入ready0等待。

##### 过程块定义

###### get\_char\_sum过程块解析：

该函数通过check来检验，数据帧在传输的过程中是否发生错误

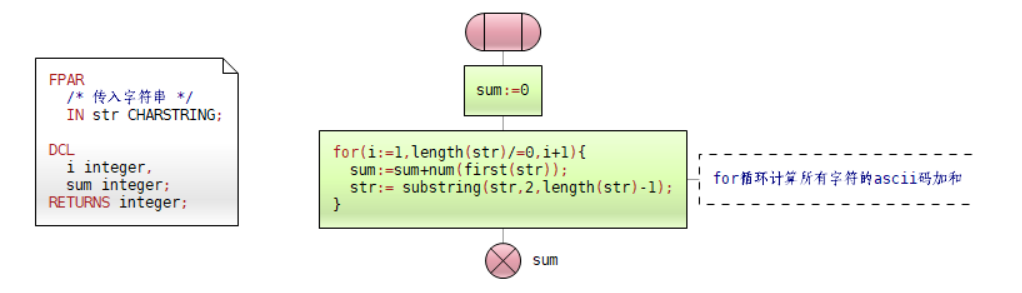
数据定义



过程块声明：



过程块定义：

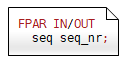


计算数据帧中数据包的字符串ASCII值之和，返回结果。

###### inc\_seq过程块解析：

该函数对传入的seq\_nr类型的值进行循环加一。

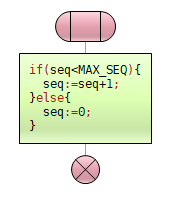
数据定义



过程块声明：

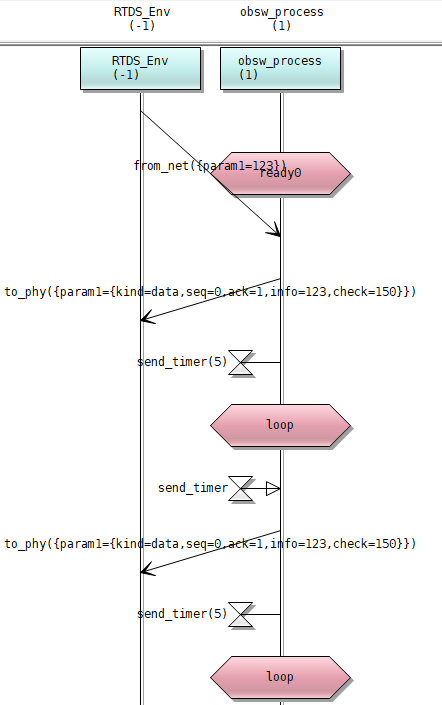
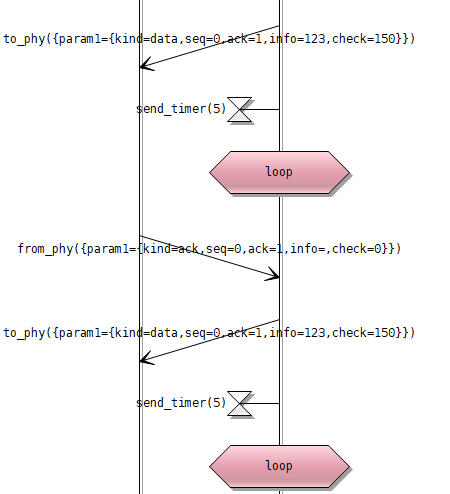


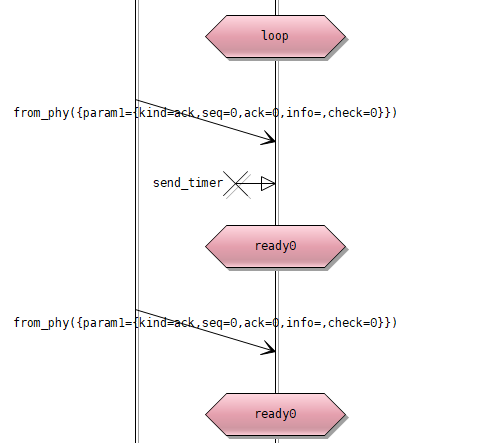
过程块定义：

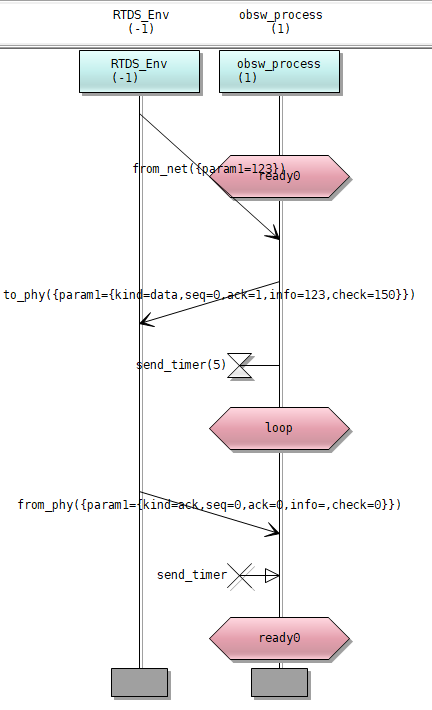
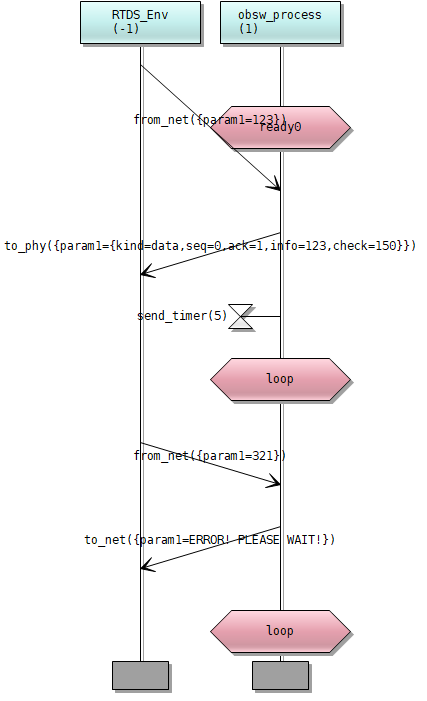
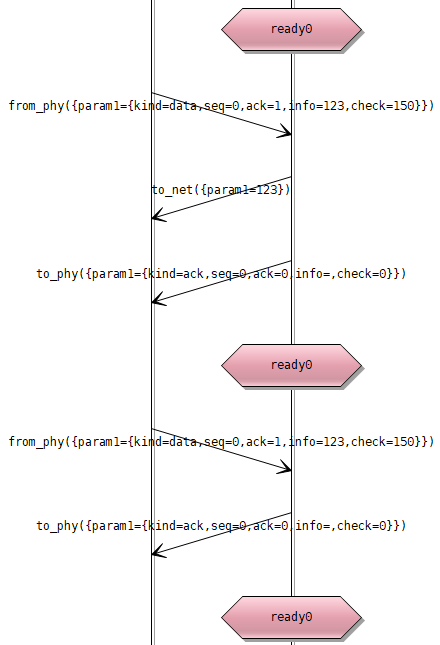
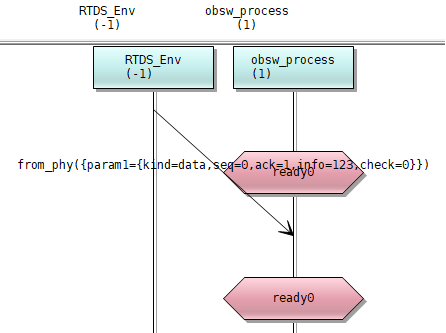
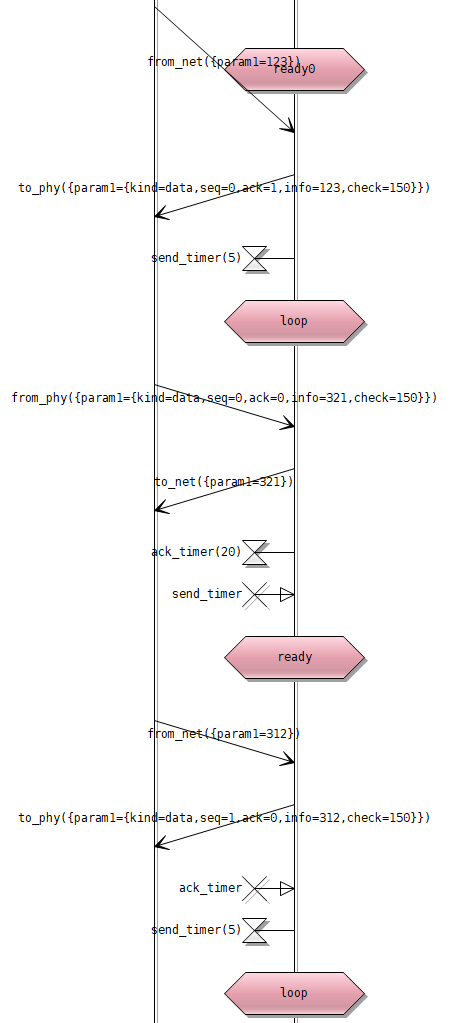


当seq值小于MAX\_SEQ时，seq加一；而当seq等于MAX\_SEQ时seq变为0。

* + 1. 模拟分析
       1. 超时重传

1. 等待超时重发Ready0状态下，网络层发来消息；
2. 协议实体将消息封装成帧，发往物理层，并启动计时器，进入loop状态等待接受；
3. 物理层没有按时传回确认帧，协议实体重新发送该帧，并重新启动计时器，回到loop状态继续等待；（将持续超时重发，直到接收到ack消息）
   * + 1. 收到错误确认帧
4. Loop状态下，物理层发来确认帧；
5. 但确认帧的序号与期待的序号不等。
6. （情况一）如果此时协议实体仍在等待正确的确认帧，则重发数据帧，并重新设定超时定时器。进入loop继续等待



1. （情况二）如果此时协议实体收到过了正确的确认帧，则不理会此错误确认帧，进入ready0状态。（此图前面发送的数据帧为seq=0，ack=1）
   * + 1. 收到正确确认帧
2. 在loop状态下，物理层发来确认帧。
3. 确认帧的序号与期待的序号相等。
4. 取消超时重传定时器。进入ready0状态
   * + 1. 流量控制
5. 在ready0状态下，网络层发来数据。
6. 协议实体将数据包成帧，发送到物理层，进入loop状态开始等待确认。
7. 此时，又从网络层发来数据，此时将会构建错误回应信息，并发回网络层，提示请等待。
   * + 1. 接收到正确/错误数据帧
8. （正确帧）当从物理层收到一帧正确的数据帧后，会将帧中数据取出来送往网络层后，构建一个确认帧并送往物理层(这里没有等待捎带确认是因为现在协议实体只作接收方，而没有发送过数据)，进入ready0完成本次数据接受。
9. （重复帧）如果此时物理层又发来重复的数据帧（可能是对方没有收到发送的确认帧，超时重发了），在接受后，并不再往网络层传输数据，但是构建确认帧已告诉发送方，该帧已经收到。并进入ready0状态。
10. （校验错误帧）如果接受到错误的数据帧是因为其校验值字段与实际校验值不同，则协议实体将会舍弃本次的接收帧，不做其他操作。等待对方超时重传。
    * + 1. 发送捎带确认的数据帧
11. 当处于ready0状态时，从网络层传来数据。此时协议实体将作为发送方封装发送数据帧。并设置超时重发定时器进入loop状态等待确认帧。
12. 从物理层传来捎带确认的正确的数据帧。将其中的数据取出发送到网络层后，将设置捎带确认的定时器，并取消超时重传定时器。进入ready状态等待从网络层传来数据。
13. 从网络层传来新的数据，此时捎带确认定时器还没有超时，则对上一帧数据的确认消息ack将搭上本次封成的数据帧发送出去。并取消捎带确认的定时器，同时设置超时重传定时器，并进入loop状态等待接受新的帧。

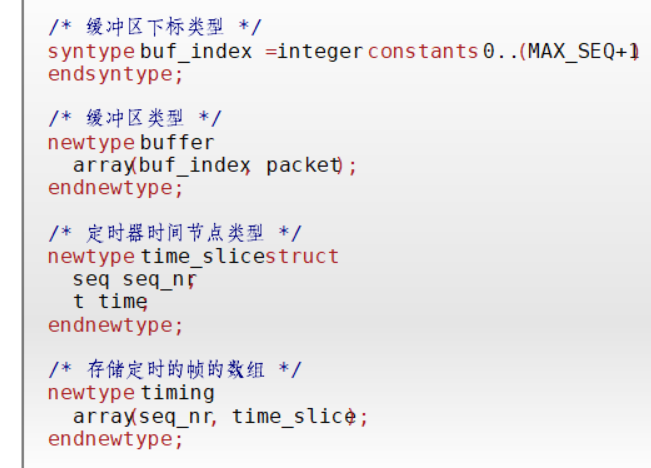
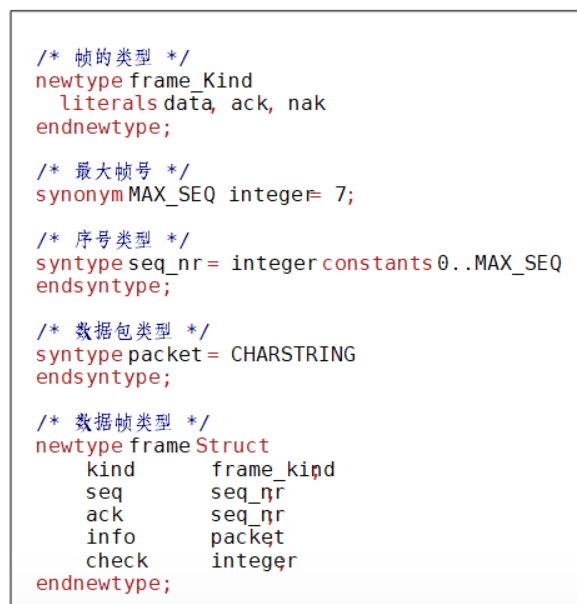
**需要说明的是**：本协议尝试捎带确认只会发生在两个协议对等体都互相发送数据时（即同时作为发送方和接收方）。

一方协议实体在得到对上一次发送数据的确认时，并接收到了新的发来的数据时，下一次发送数据将会尝试捎带确认。

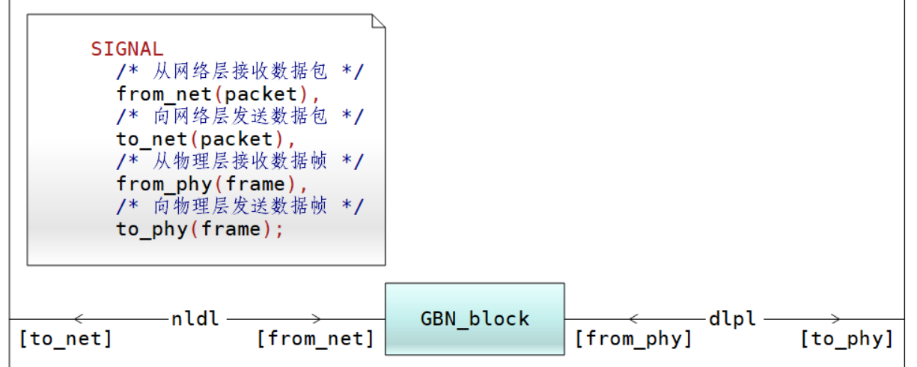
而当只有一方发送数据时，另一方必定只负责接收和确认。这种情况下接收方将直接发送ack类型的确认帧

* 1. 回退N协议：
     1. 协议描述
        1. SDL系统图

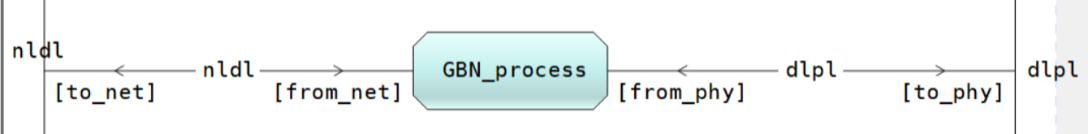
##### 系统中的数据定义（part0）



##### 系统功能块与外界交互（part1）



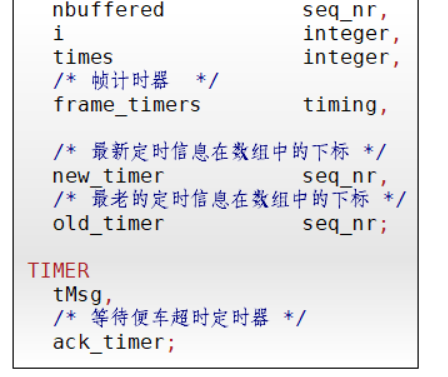
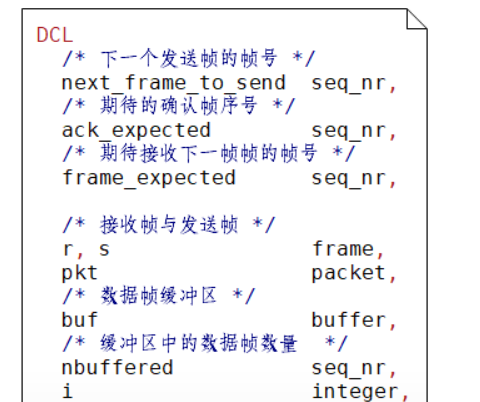
1. Signal 为来自网络层/物理层，通向网络层/物理层的四个信号，代表传输数据的流向。
2. 整个系统包含一个功能块GBN\_block，其连接两个双向信道 nldl(网络层数据连路层通道) 和 dlpl(数据链路层物理层通道)，信道上传输对应的Singal。
   * + 1. SDL功能块



GBN\_block功能块中仅包含一个进程块GEN\_process，其直接与环境进行通信。

* + - 1. SDL进程块

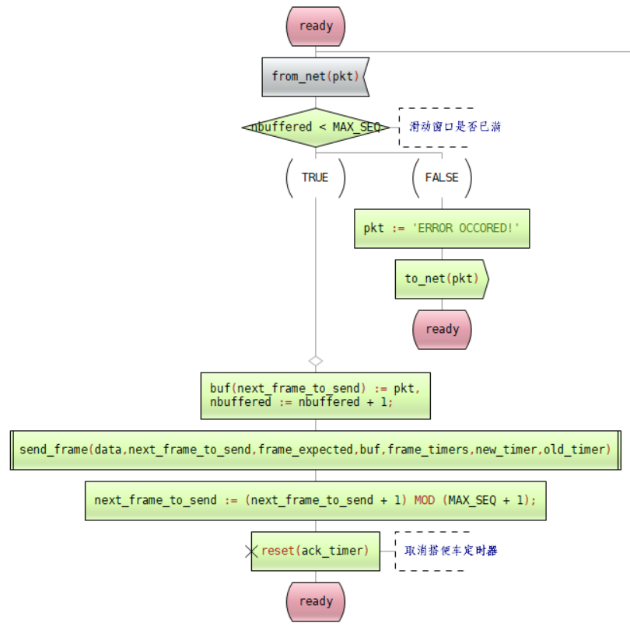
##### 进程中的数据定义：



##### SDL语句描述：

###### 进程入口：

数据初始化进入ready状态。

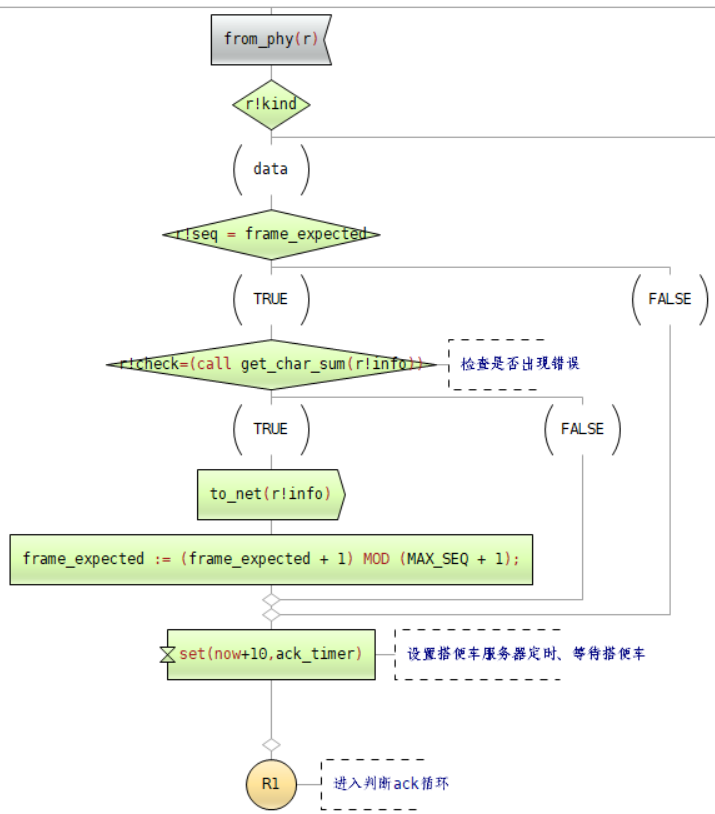


###### 从网络层接收到数据包

判断缓冲区中的帧数量是否小于MAX\_SEQ保证可发送的帧数量最多为MAX\_SEQ个。如果不满足条件，则向网络层发送缓冲区已满的消息，然后重新进入ready状态。如果满足条件，则将该数据包放入缓冲区中，之后调用send\_frame函数向物理层发送数据，改变下一发送帧的序号，发送数据帧完成，重新进入ready状态。

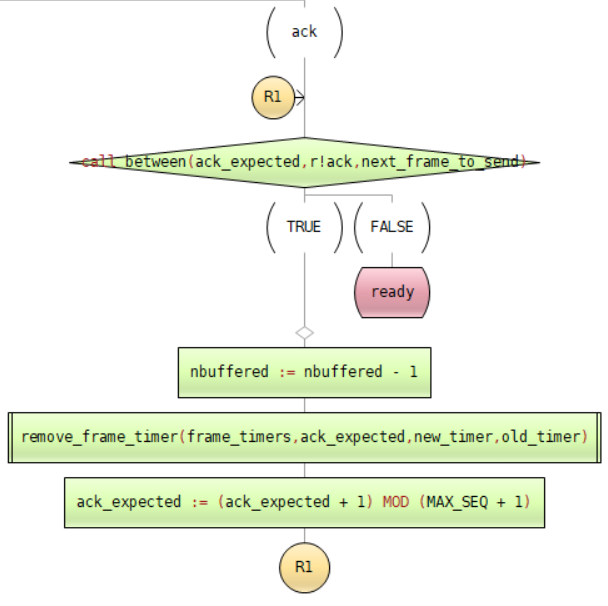
###### 从物理层接收数据帧：

帧类型为data：

判断数据帧序号是否等于期待的下一帧帧号，如果为期待的下一帧帧号且数据帧校验通过，数据没有发生错误，则将该数据帧的数据包传入到网络层，改变期待的下一帧帧号，设置捎带确认计时器转入到识别确认帧状态（data数据帧可能包含确认帧的消息），否则跳过将数据包发送到网络层这一过程。

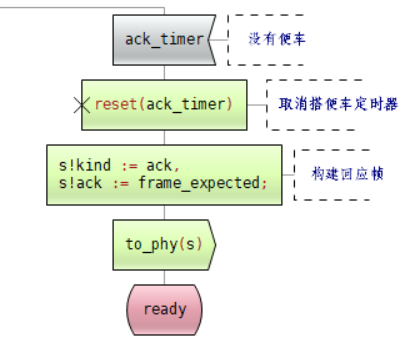
接收帧类型为ack：

如果判断接受到的ack值是否满足条件，如果不满足，则重新进入ready状态，如果满足则nbuffered减一，释放一个缓冲区，移除期待确认帧的序号的时钟计时器，改变期待确认帧的序号。循环这一过程，直到不满足条件为止。



当捎带确认计时器超时后：

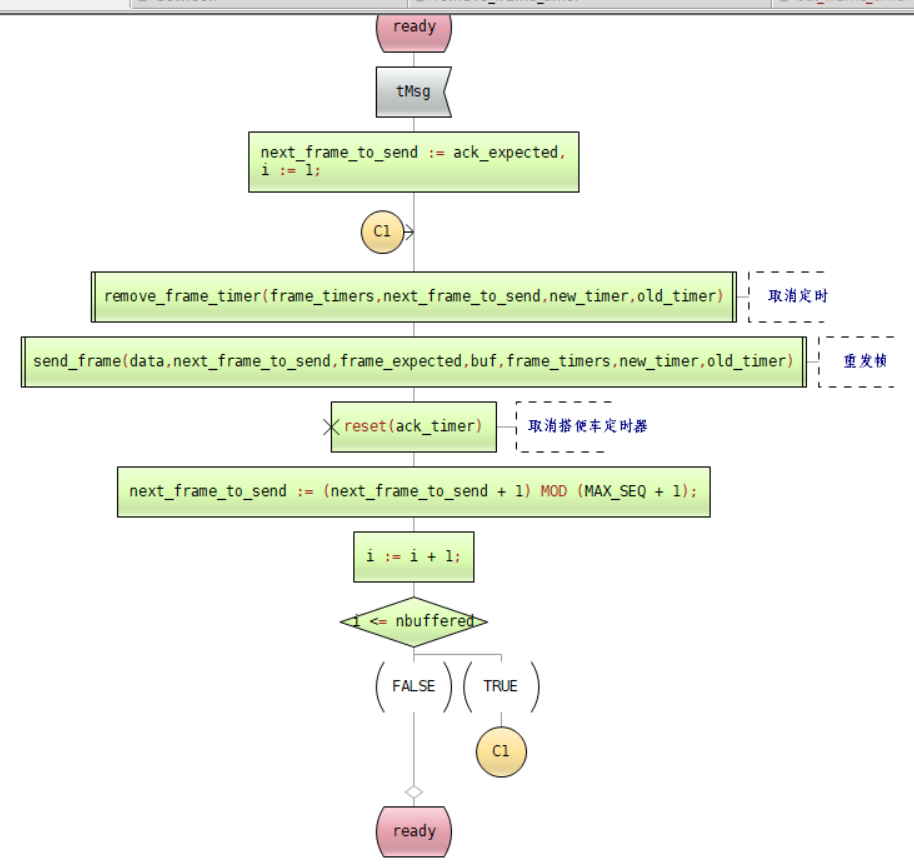
取消捎带确认计时器，向物理层发送确认帧，之后进入ready状态。



当数据帧计时器超时后：

将下一发送帧的序号改为期待接收确认帧的序号，设置计数器i=1，

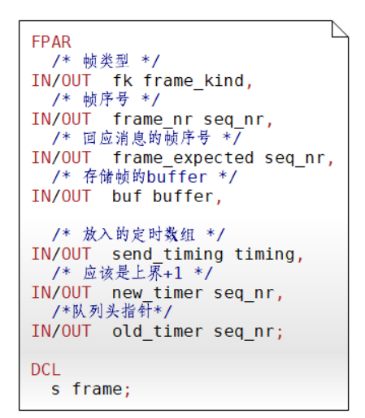
取消计时器定时，重新发送数据帧，取消捎带确认计时器，改变下一发送帧序号，计数器I += 1，如果i<=缓冲区内帧的数量，则继续循环重新发送下一个数据帧，否则进入ready状态。



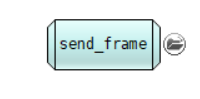
##### 过程块定义

###### send\_frame()解析：

数据定义：

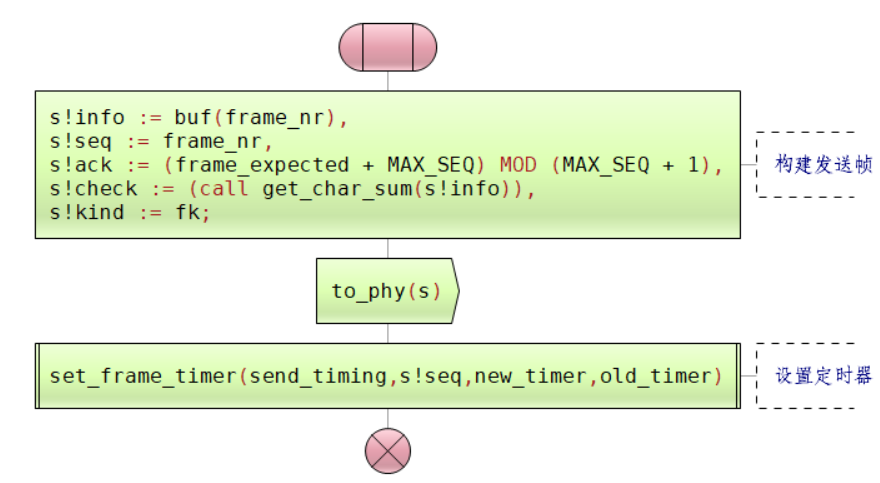


函数声明：



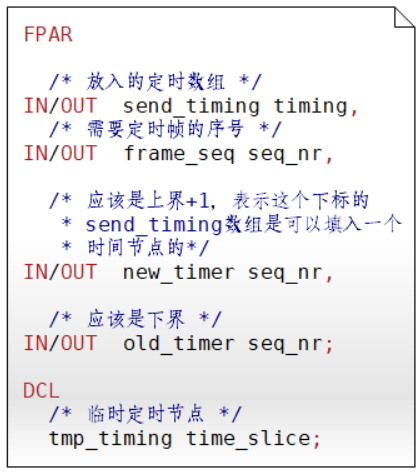
函数定义：

将缓冲区中的数据包放入到发送帧中，添加数据帧序号，添加确认帧信息（确认帧信息捎带确认），设置校验码，设置发送帧的类型为fk（data），之后为该号数据帧设置定时器。

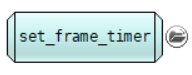


###### set\_frame\_timer()解析：

数据定义：

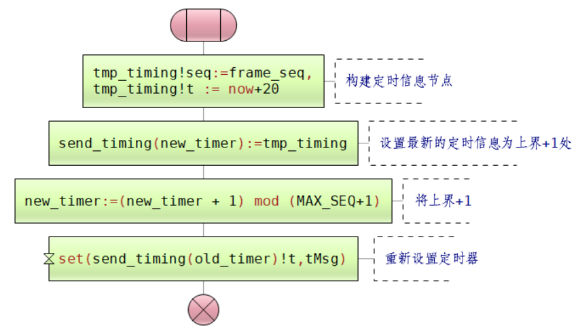


函数声明：



函数定义：

对该号数据帧设置定时器，定时重发时间为20之后。设置最新帧定时消息，之后+1取模，对旧的定时器进行重新设置。

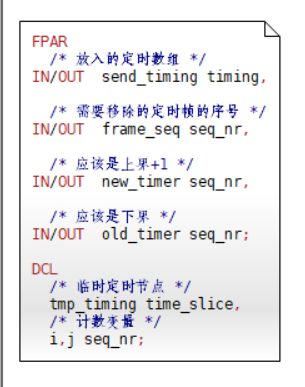


###### get\_char\_sum()解析：

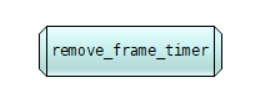
同协议4

###### remove\_frame\_timer()解析

数据定义：



函数声明



函数定义：

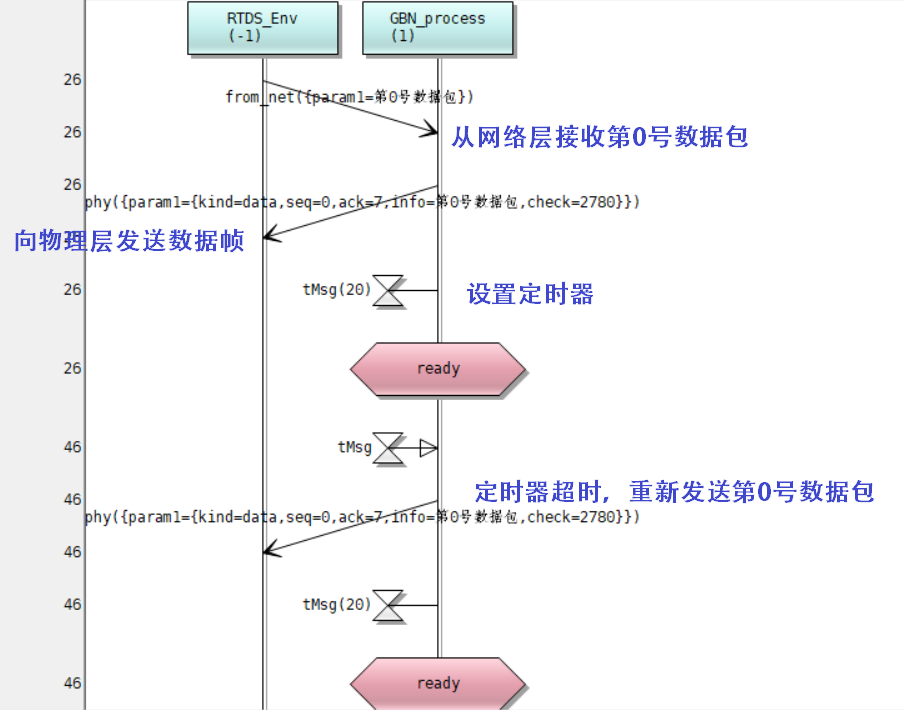
取消一个帧的定时，实际上就是将其定时信息在定时信息循环队列中删去。则分为两种情况：要取消的是最老的定时器以及非最老的定时器。

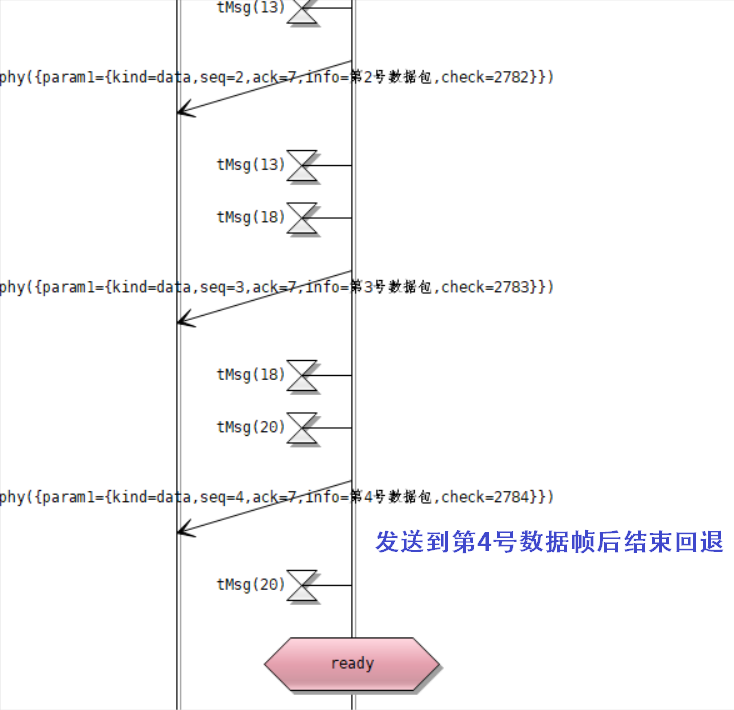
当要取消的是最老的定时器时只需将定时器循环队列下界前移，重新设定定时器即可（如果只有一个信息则不需要再设定定时器而是取消定时器）。

如果要取消的不是最老的，则定时器不用改变，只使用覆盖的形式将需要删去对应的定时信息删去，然后将循环队列的上界减小。

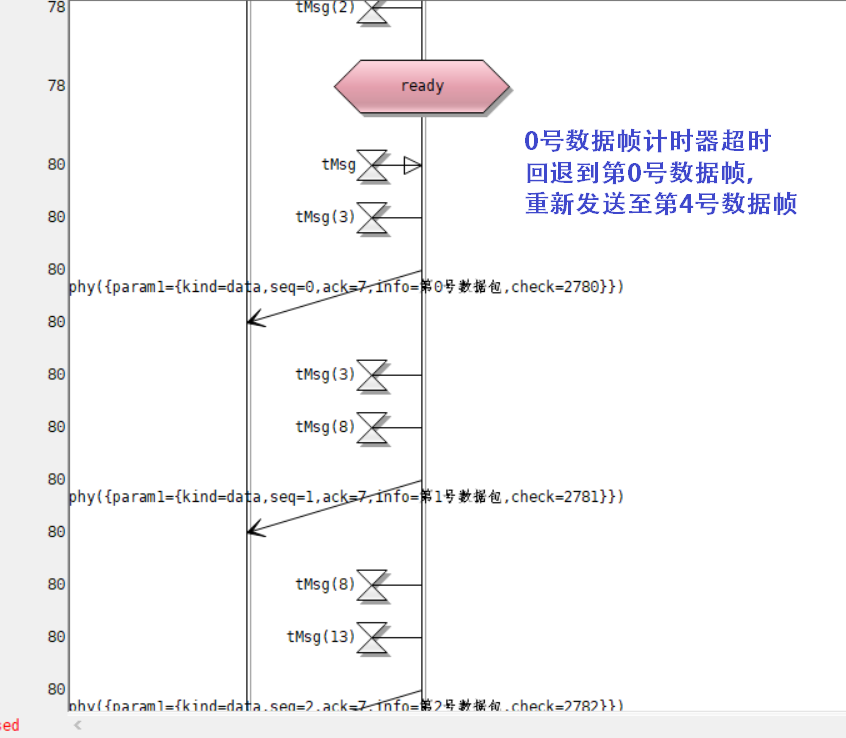


* + 1. 模拟分析
       1. 等待超时重发





* + - 1. 接收网络层多个数据包超时等待重发



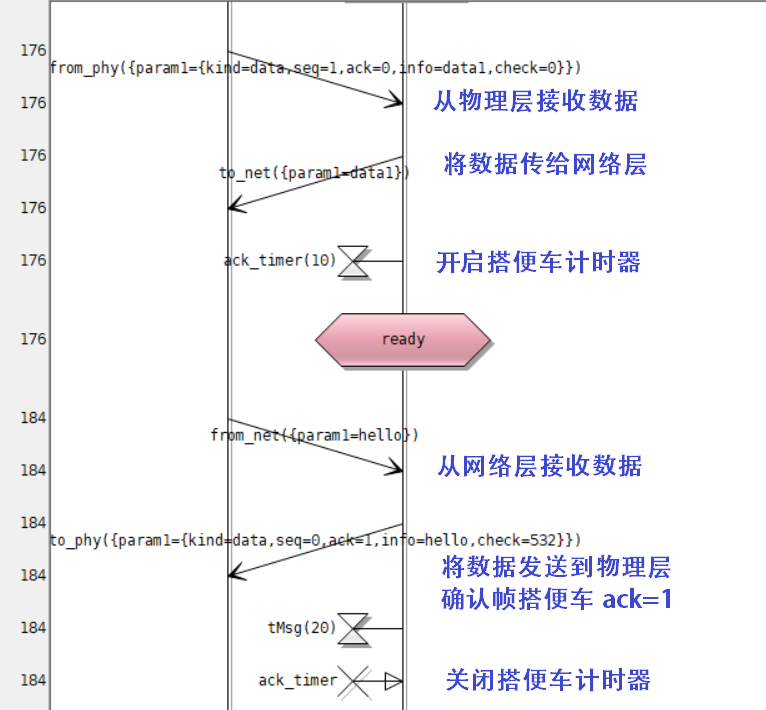
* + - 1. 接受物理层确认帧



* + - 1. 接收物理层数据帧（不捎带确认）

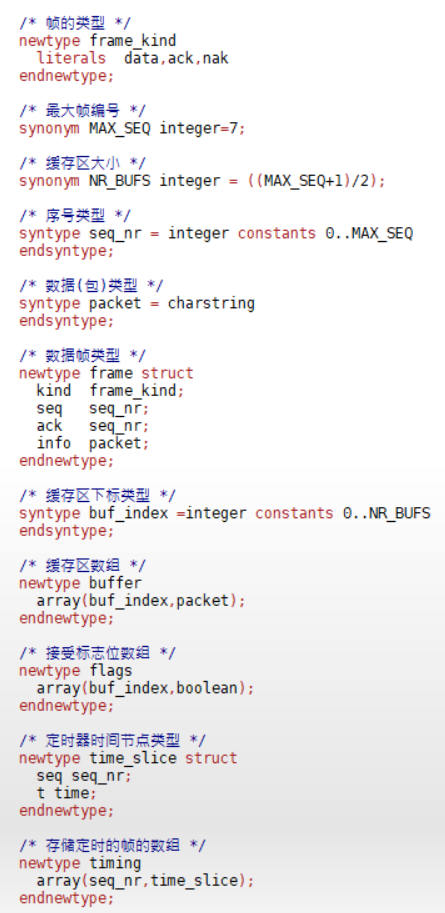


* + - 1. 接收物理层数据帧（捎带确认）

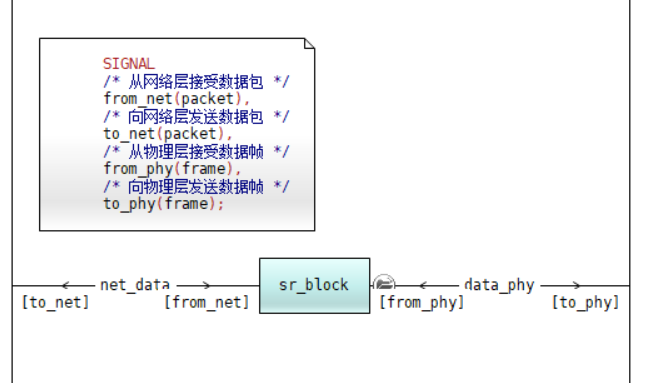


* 1. 选择重传协议：
     1. 协议描述
        1. SDL系统图

##### 系统中的数据定义（part1）



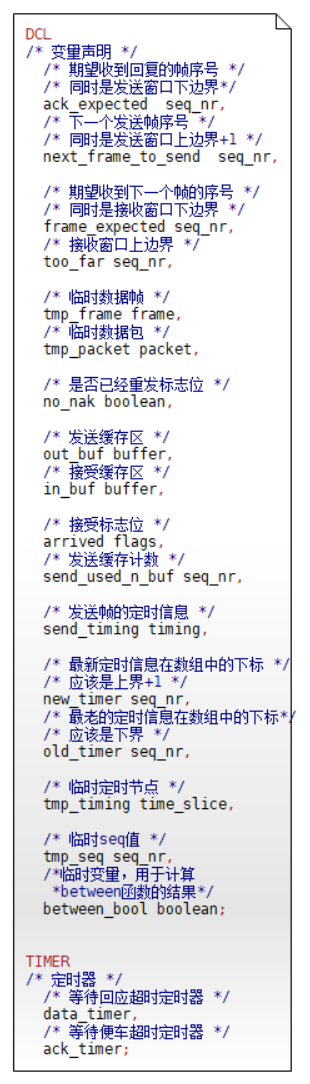
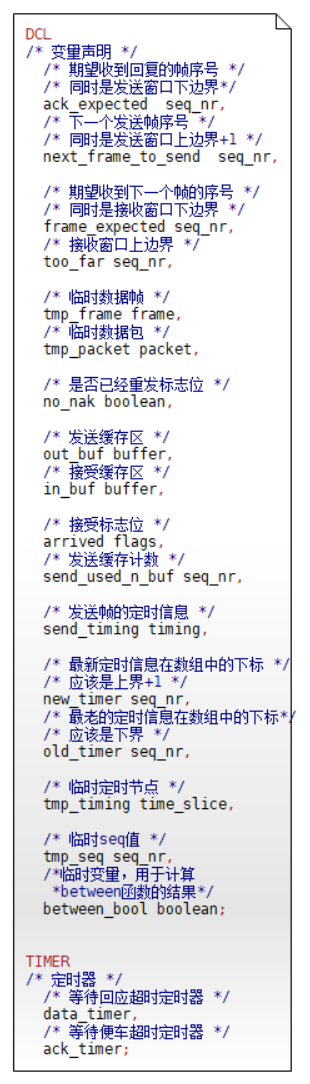
##### 系统功能块与外界定义（part0）

1.  Signal包含来自网络层/物理层，通向网络层/物理层的四个信号，代表数据传输的流向
2. 系统包含一个功能块sr\_block，连接两个双向信道：net\_data为网络层和数据链路层之间的信道，data\_phy为数据链路层和物理层之间的信道。信道上可以传输对应的Signal。
   * + 1. SDL功能块

Sr\_block中包含sr\_process进程，直接与环境进行通信。

* + - 1. SDL进程块

##### 进程中的数据定义

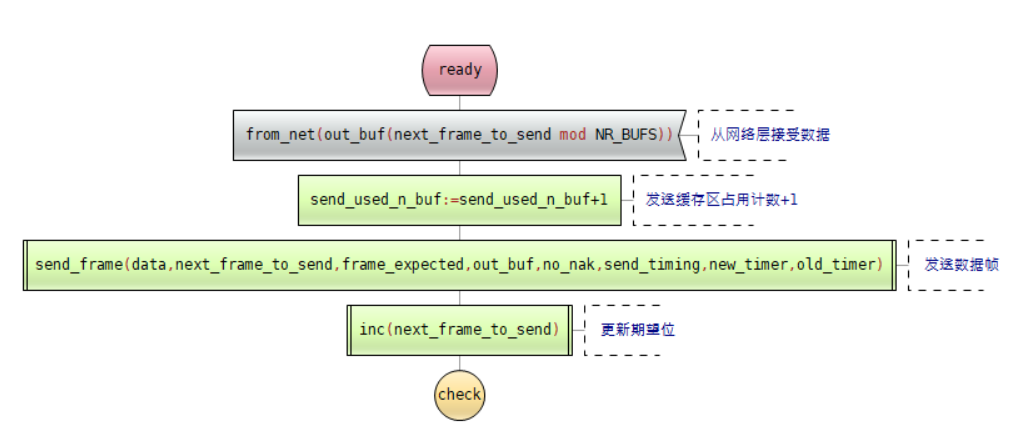


##### SDL语句描述

###### 初始化进程：

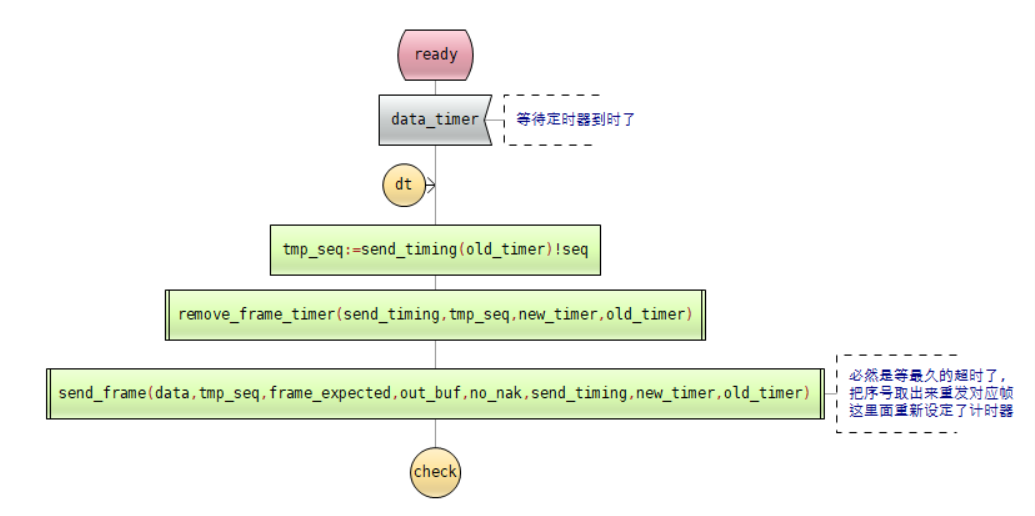
启动后，首先将数据进行初始化。进入ready状态。

###### 从网络层接收数据



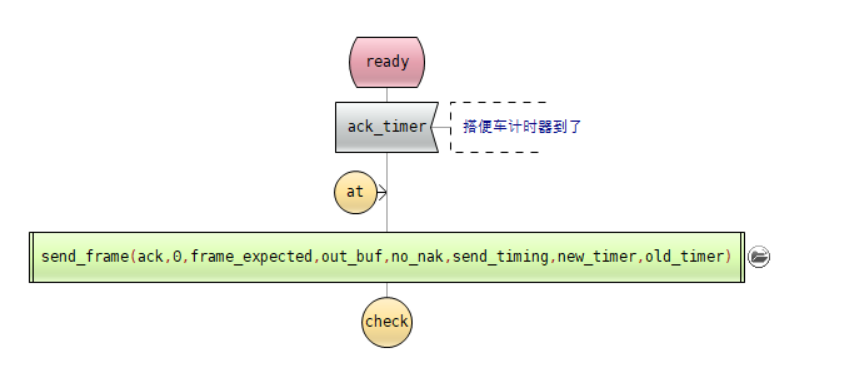
从网络层接收数据后，进程将数据写入缓冲区，将数据发送到物理层，并更新期望位。

###### 等待计时器触发



等待计时器触发，表明进程没有在期望的时间内收到ack消息。系统将队列中等待最久的消息进行重传。

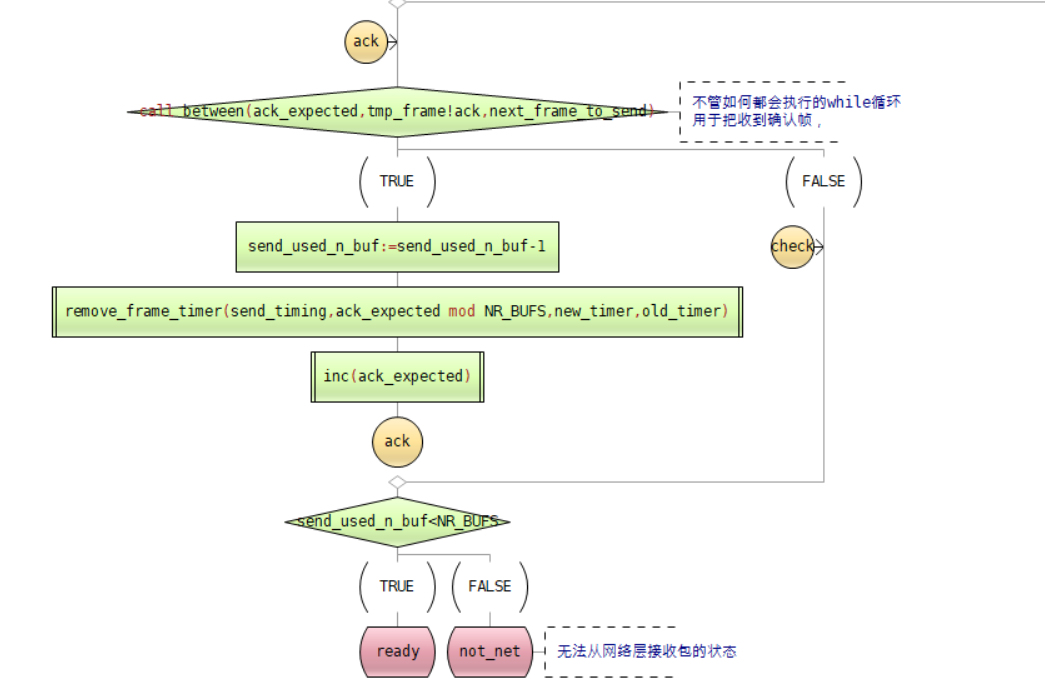
###### 捎带确认计时器触发



捎带确认计时器触发，表明系统在期望的时间内没有发送data包，不能以捎带确认方式发送ack消息。此时系统发送一条单独的ack消息。

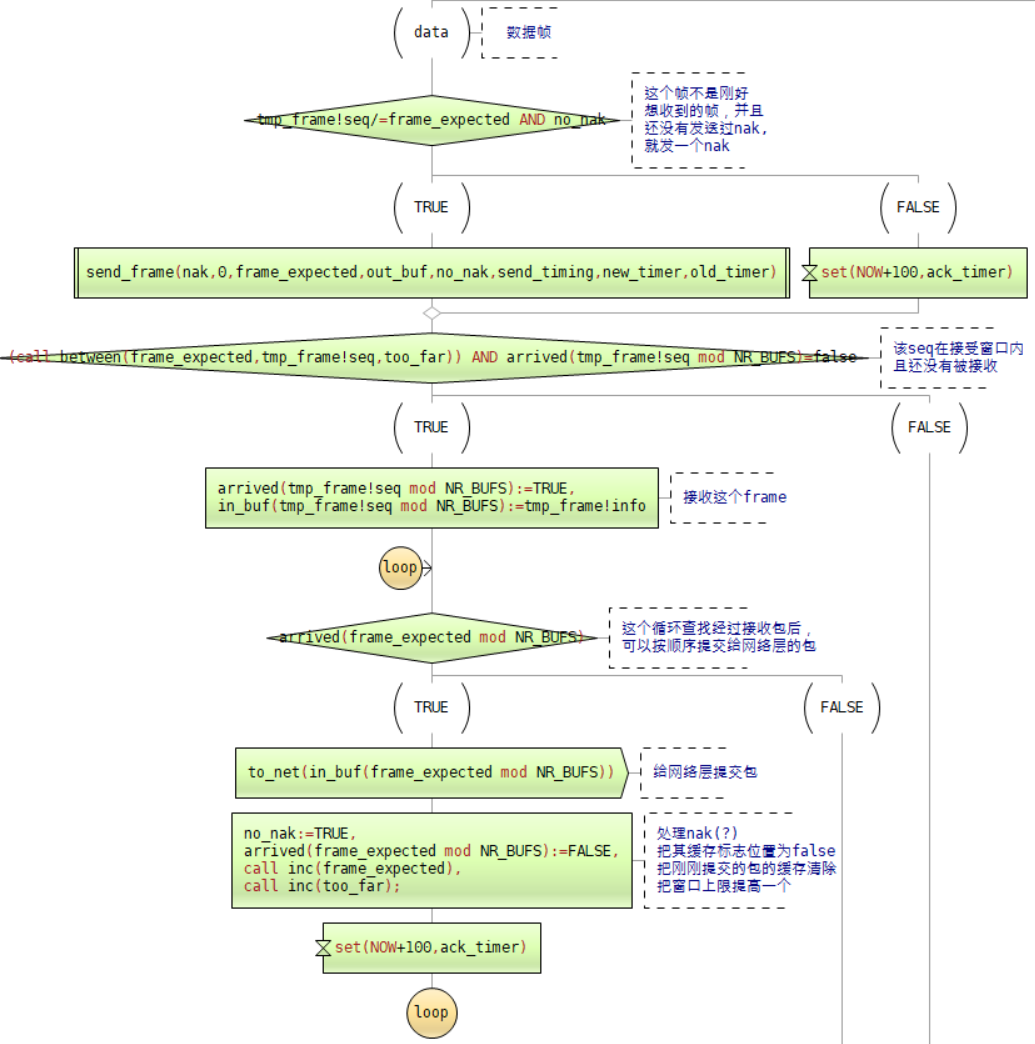
###### 从物理层接收到帧

接收到ack帧：



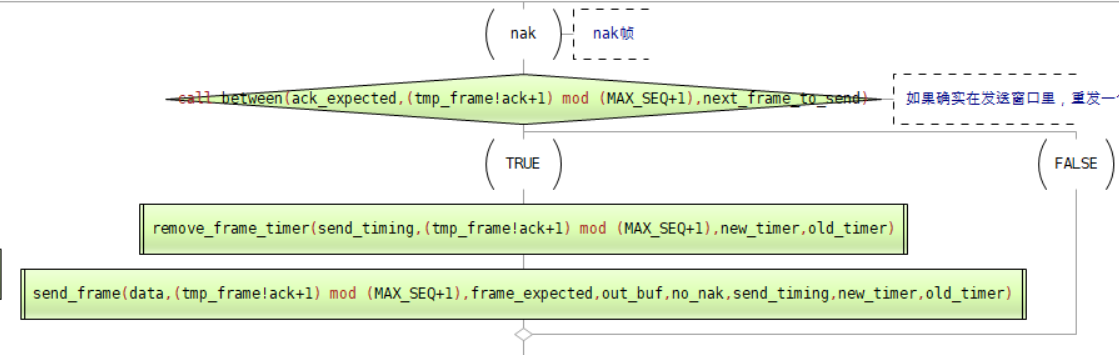
接收到ack帧后，进程会释放已ack的数据帧对应的缓冲区，取消对应的超时重传定时器，并更新期望的ack号。

接收到data帧



如果收到data帧的seq号不在接收窗口中，则发送nak；否则，设定捎带确认计时器，并检查当前已接收的序列，将可以按序提交的数据发送到网络层。

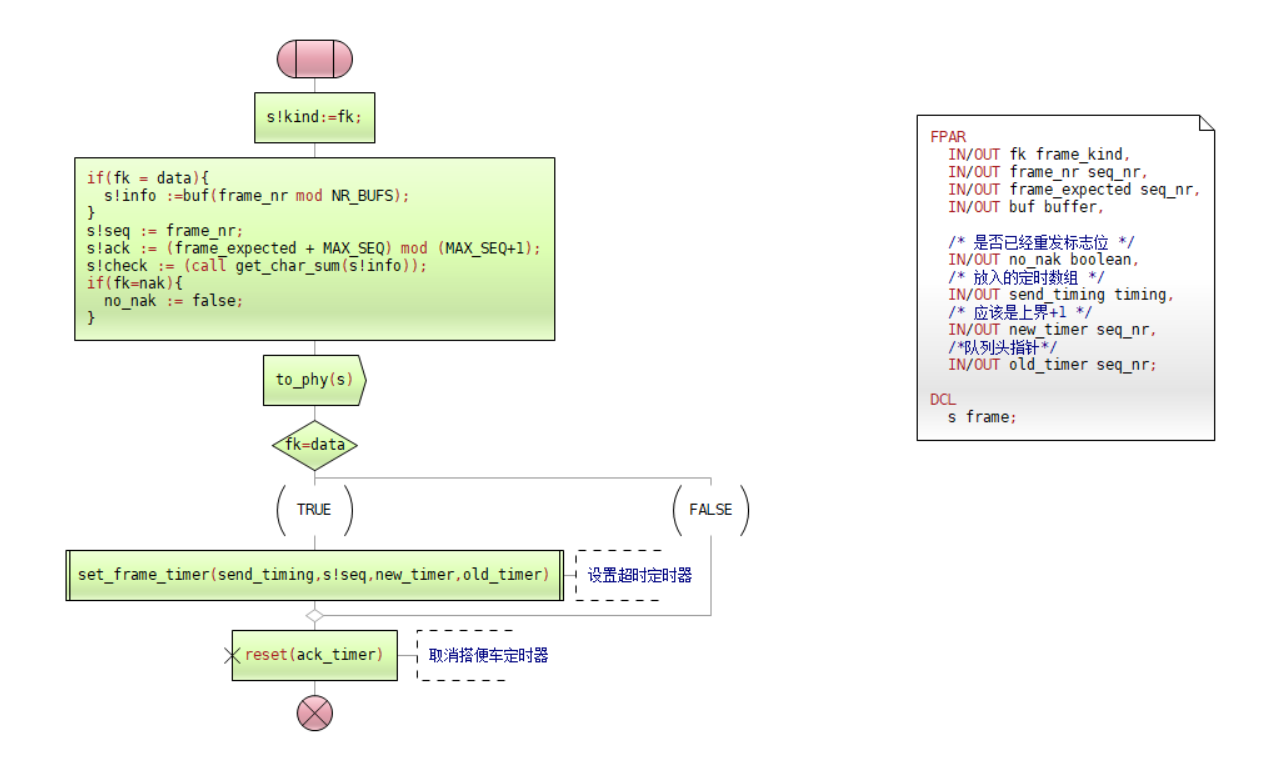
接收到nak帧



接收到nak帧后，重发没有被成功接收的data帧，并重新设定计时器。

##### 过程块定义

###### Send\_frame子过程



Send\_frame过程用于封装、发送数据帧。该过程根据系统当前状态，将seq号、ack号填入一个帧。如果帧为data类型，还需要从缓冲区里读取帧的内容，填入info字段，并计算校验和。

将帧发往物理层后，如果发送的是data帧，该过程将设定计时器，如果在限定时间内没有接收到ack消息，则重发这个data帧。

###### Set\_frame\_timer子过程

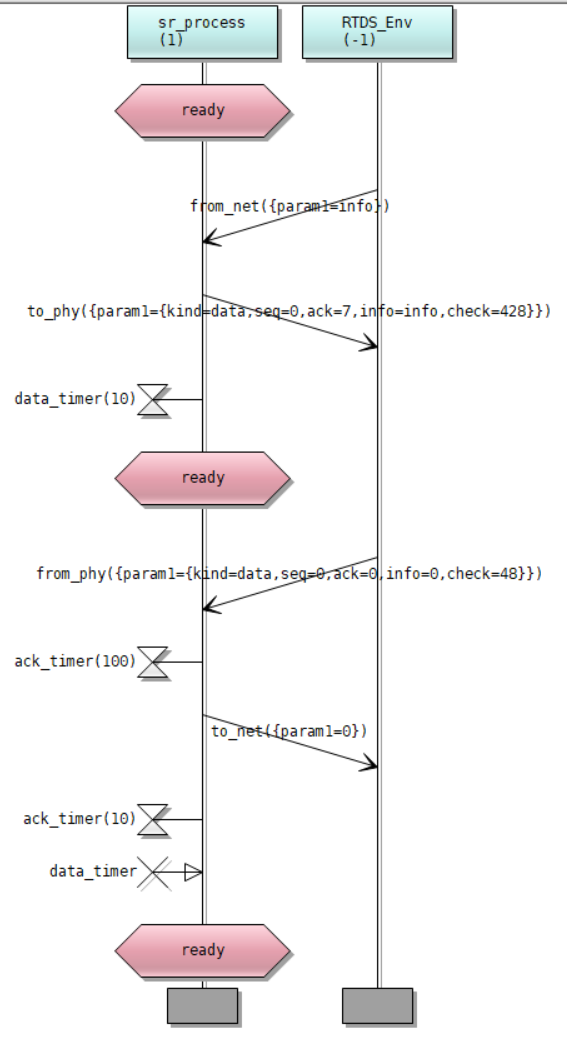
同回退N协议

###### Remove\_frame\_timer过程

同回退N协议

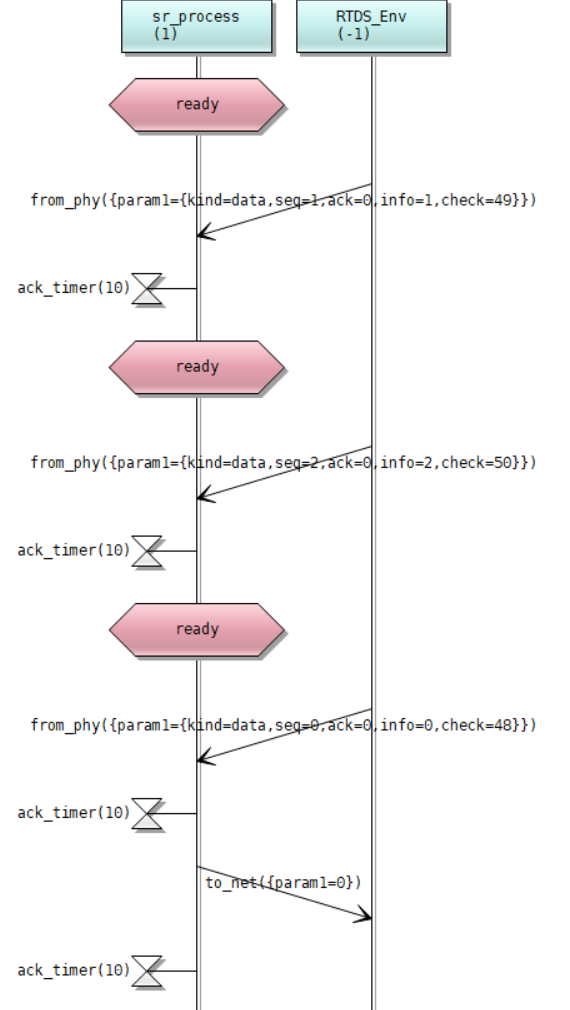
* + 1. 模拟分析
       1. 捎带确认测试

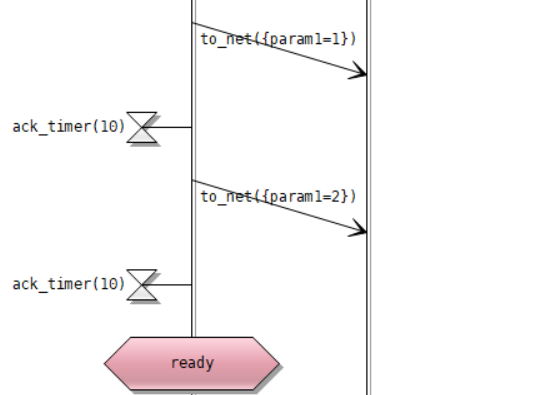
向物理层发送数据，物理层回传data包，并使用捎带确认方式发送ack号，系统正确识别ack消息，并等待下一个data包，尝试以捎带确认方式发送ack号。



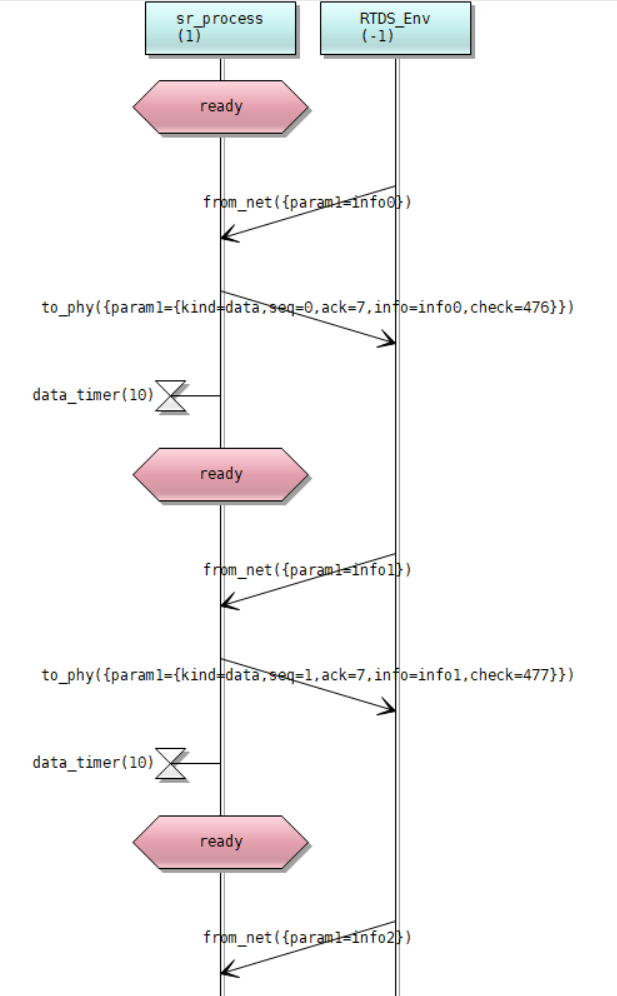
* + - 1. 随机接收测试

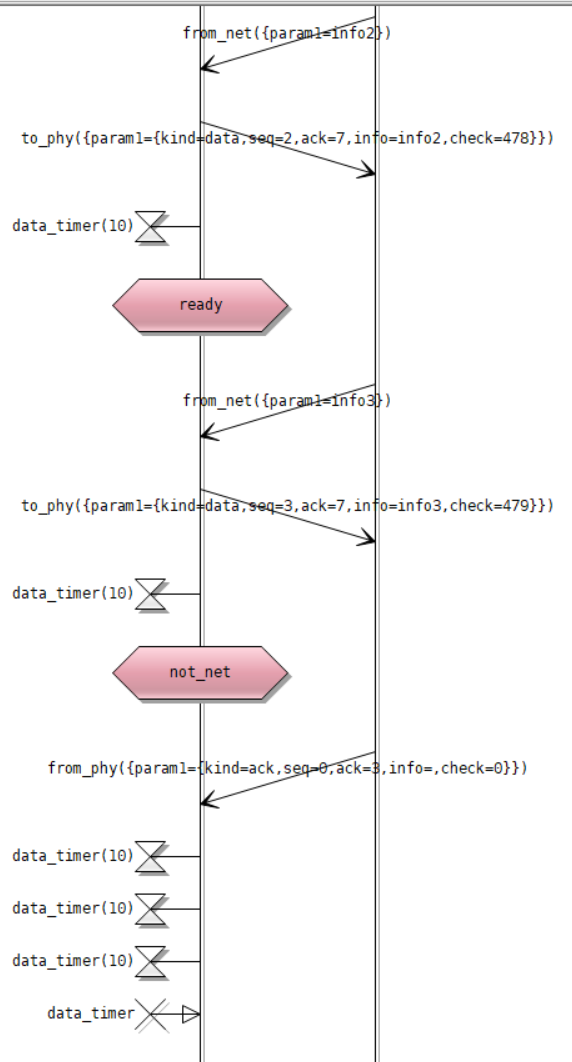
乱序发送三个包（seq为1、2、0）在接收到seq=0的数据包后向网络层依次发送三个数据包





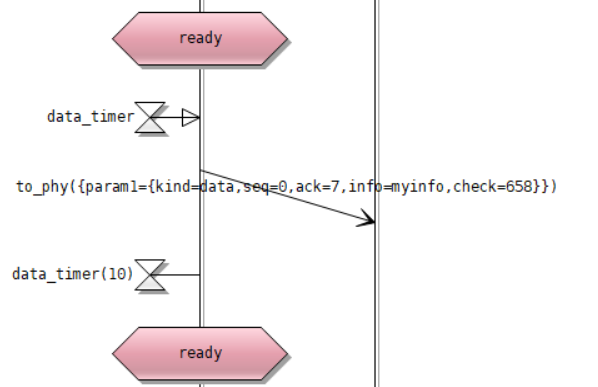
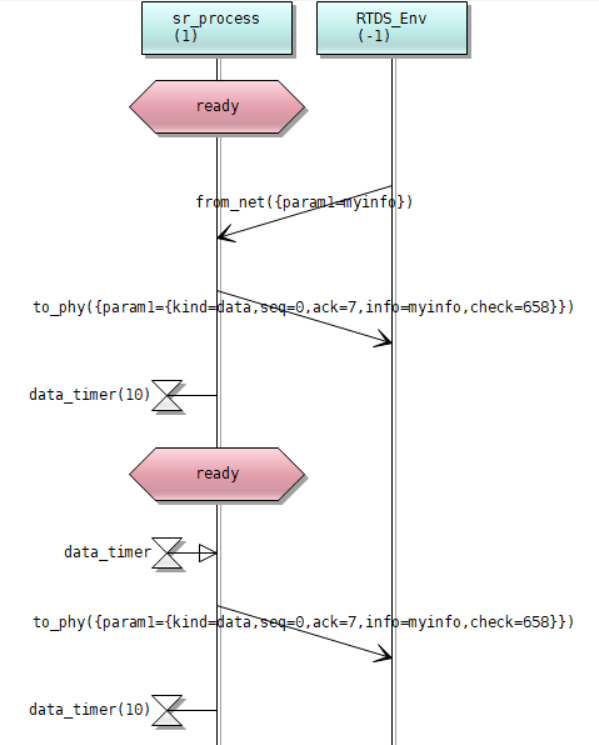
* + - 1. 拒绝网络层发送请求



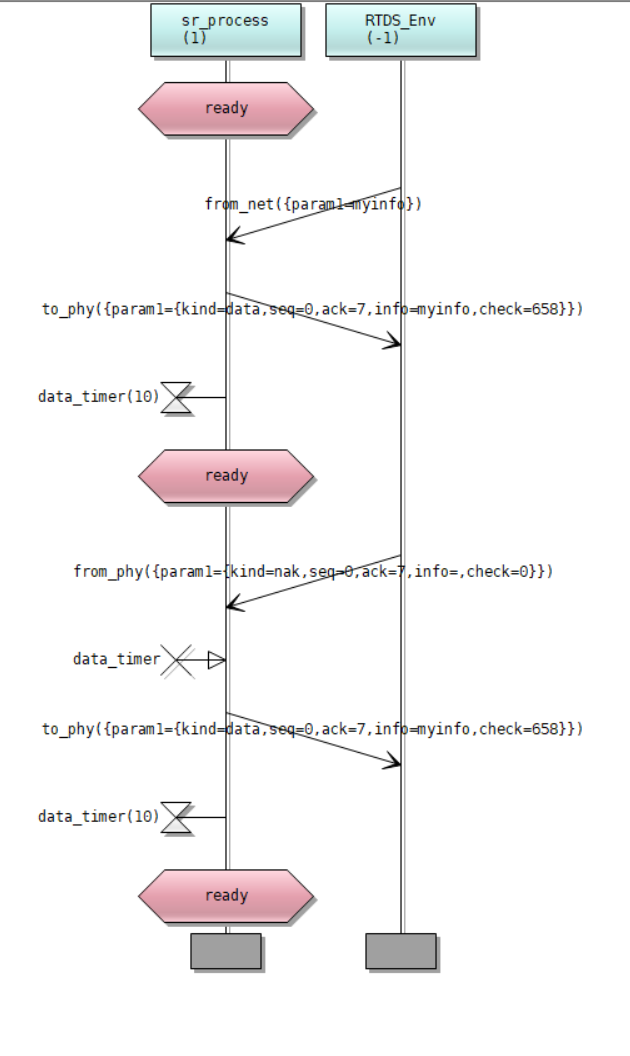




* + - 1. 超时重传



* + - 1. Nak重传



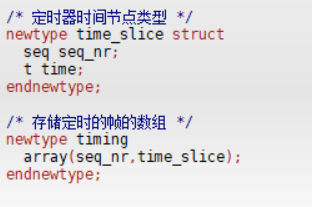
# 问题与解决

* 1. 单个时钟模拟多个时钟

当我们想使用SDL的定时器数组时，发现Pragma Dev并不支持。我们若要对选择重传协议中要实现的“对每一个已经发出的帧设置一个超时计时器”，则要么建立多个不同的计时器，要么用一个计时器模拟。前者的扩展性不好，并且维护困难，所以我们考虑使用后者的方案

经过思考和书中提示，我们选择了队列这种数据结构来模拟多个时钟 。具体设计方法如下

对每一个发送的帧设置一个时间片，时间片按照顺序存入定时器队列，这两种数据结构的定义如下所示：



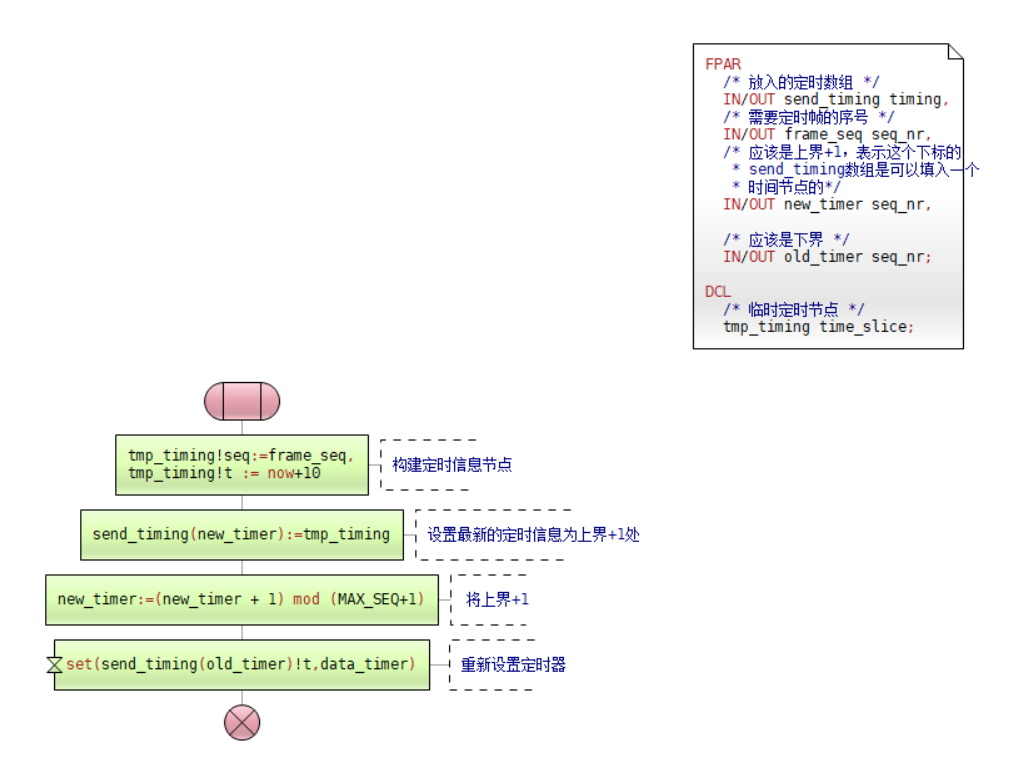
数据类型定义

时间片结构定义了时钟对应的帧序号seq、一个时间点t，对应着序号为seq的帧在时间t时超时。

因为没有直接定义的队列，我们又用一个数组和头尾指针模拟了队列，也即send\_timing和new\_timer/old\_timer，old\_timer总是指向数组中的队首，表示最老的时间片，new\_timer总是指向队尾+1的位置，表示下一个时间片插入的位置，整个队列是循环使用的，也即下标7的下一位是下标0。

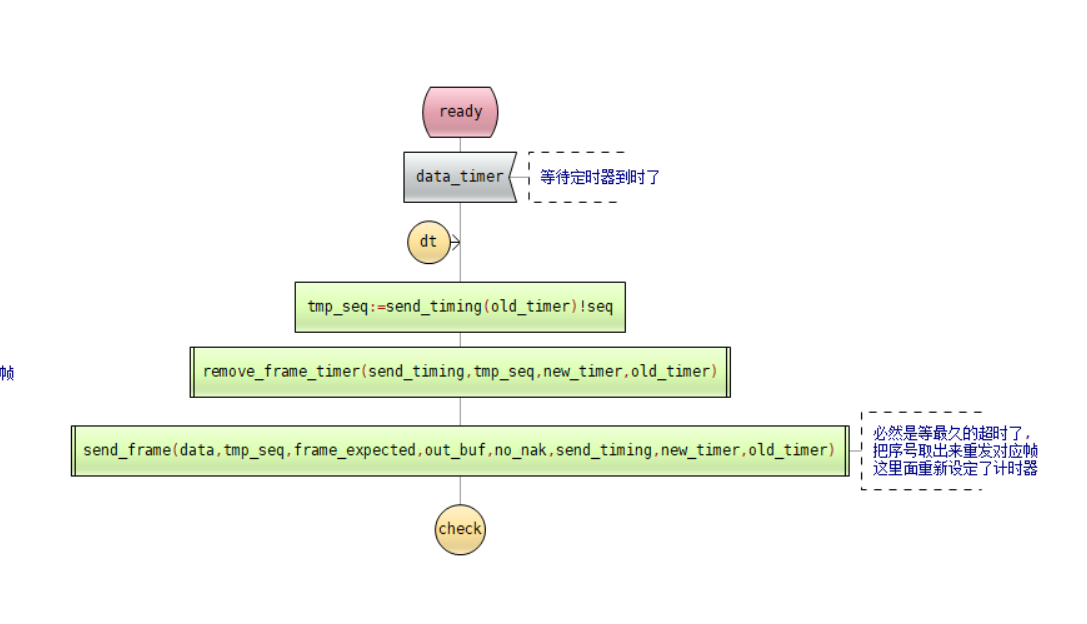
在发送帧时，为该帧构建一个时间片并将其加入队尾。由于有可能是超时触发了重传（也从队首取出了一个时间片），需要重新设置一下计时器，如果这个发送帧是由网络层主动发送，则这里重新设定计时器还是最早的时间片，不会产生影响。

下图表示了向队列中加入时间片的过程



Set\_frame\_timer子过程

若超时定时器触发了则进入如下过程，将队首时间片取出（remove\_frame\_timer），再重新发送该帧（send\_frame）。



主进程中触发超时计时器的情况

若从

* 1. 过程的抽象

在协议中，经常需要反复执行某些操作并读写某些数据，如发送帧、设定或撤销计时器、计算校验和等。我们采用编程中函数的思想，把常用操作封装为过程，通过传递参数的方式，执行访问计时器队列、发送帧等操作，从而降低代码的复杂度，提高可维护性。

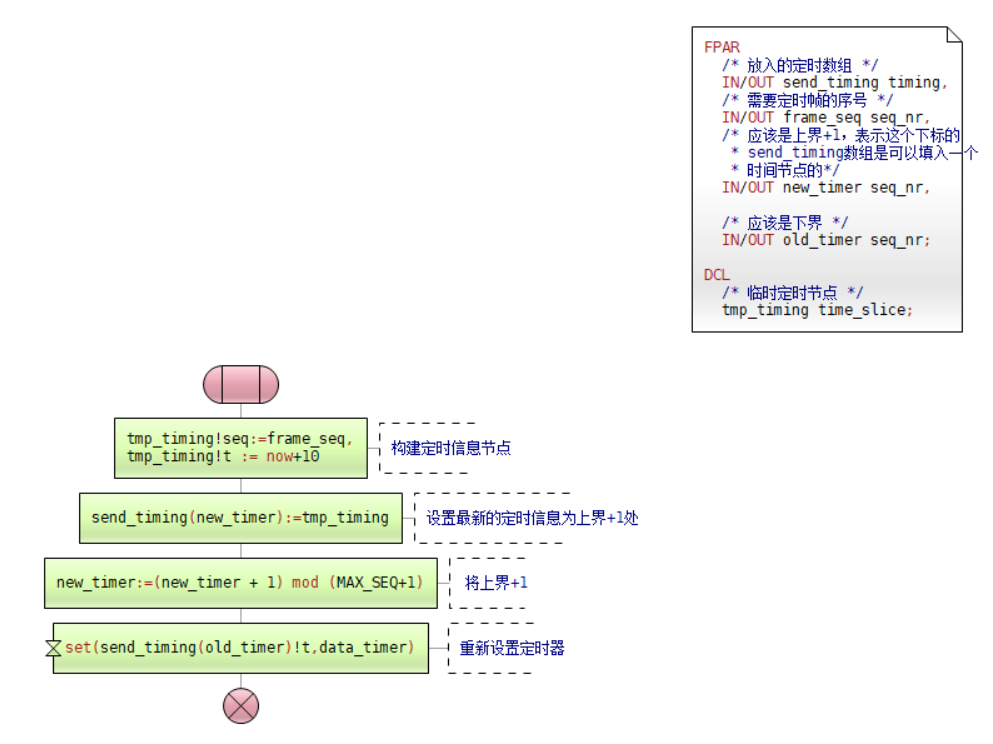
例如，对于常用的发送帧操作，我们将其进行封装为send\_frame过程，通过传入参数来指定要发送帧的类型和信息，在过程内部进行访问缓存、成帧、发送、设定计时器等操作。

以下为send\_frame过程的参数表和流程：

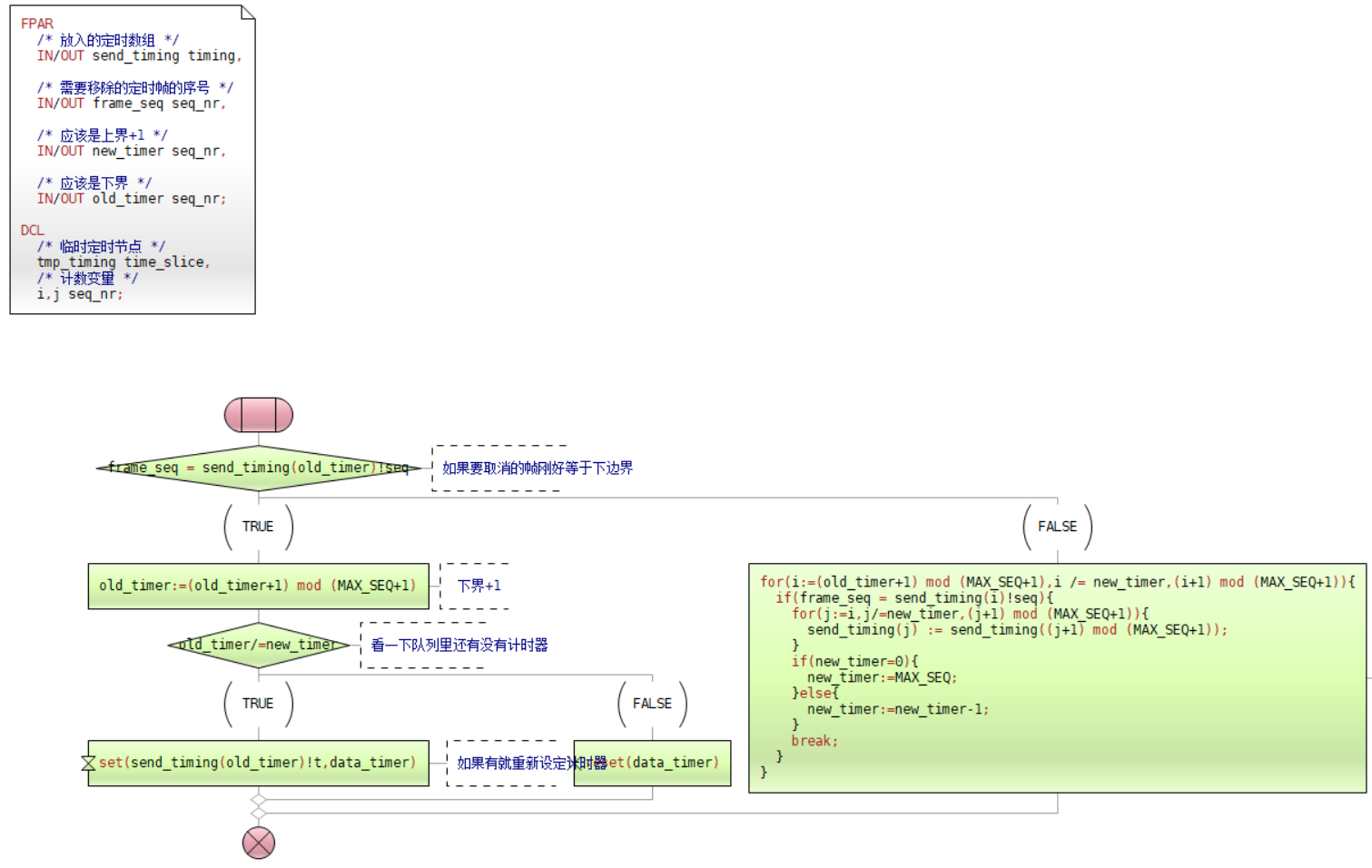


又例如：选择重传协议使用队列来维护计时器，考虑到如在主进程中直接维护队列，很容易导致错误访问等问题，我们通过封装设定计时器、取消计时器两个过程，实现对计时器队列的访问，以简化代码设计。

以下是set\_frame\_timer的参数表和流程：



以下是remove\_frame\_timer的参数表和流程：



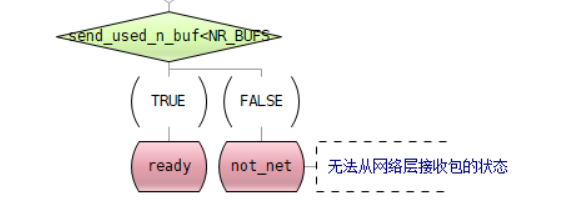
* 1. 实现拒绝网络层消息

在实验过程中我们发现，有两个函数我们必须要考虑，即enable/disable\_network\_layer。这两个函数如何实现书上并没有给出，我们经过思考和探讨后认为，实现这两个函数有两种方式

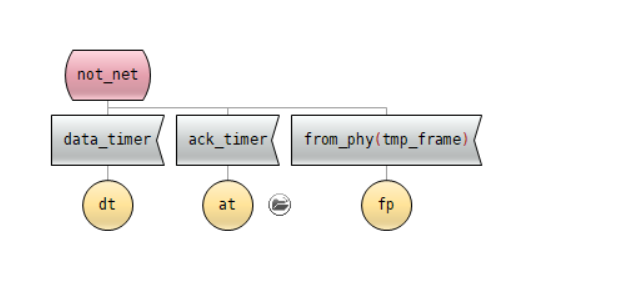
* + - 1. 在数据链路层程序中设计两种ready状态，其中一种不接受网络层发包，其余事件均可触发
      2. 向网络层程序发送特定的控制消息。

前者的好处是不用增加消息类型的定义，且不用考虑多种网络层协议，缺点是网络层对数据链路层状态并不知情，会导致网络层阻塞，或者网络层到数据链路层的管道/消息队列阻塞。后者的好处是网络层清楚了数据链路层的状态，但是设计更复杂，定义更多。

我们选择了第一种实现，其中检查缓冲区是否已满，进入不接受网络状态的部分进程如图。



not\_net状态，只接受其他三种事件



# 附录

（附上实验文档，如：问题分析、设计方案、算法、设计图、程序、仿真结果、运行结果、调试心得等，具体内容根据实验要求来定。源代码请附在这里。源代码排版请特别注意，用5号字体，行间距为单倍行距。注意节省空间，不要浪费纸张。）