西安电子科技大学网络信院

计算机网络大作业
课程实验

实 验 报 告

后退 N 帧算法模拟

组号：第 4 组

组员：丁友涛、邹珂琳

学号：22049200180、22049200424

日期：2024. 12. 21

一、 程序设计思路

1. ARQ算法

ARQ协议，即自动重传请求（Automatic Repeat-reQuest），是OSI模型中的错误纠正协议之一。

它通过使用确认和重传这两个机制，在不可靠服务的基础上实现可靠的信息传输。

如果发送方在发送后一段时间之内没有收到确认帧，它通常会重新发送。

重传的请求是自动进行的，接收方不需要请求发送方重传某个出错的分组

2. 返回N帧

2.1 定义

后退N帧ARQ就是从出错处重发已发出过的N个帧

2.2 发送方数据分类

发送方根据滑动窗口状态,将数据分为以下四类:

① 发送完确认的帧:数据发送完毕,已经收到了接收端确认信息;

② 发送完等待确认的帧:数据发送完毕,还没有收到确认信息;

③ 还能发送的帧:滑动窗口中,还没有发送的帧;

④ 还不能发送的帧:滑动窗口后面的帧;

2.3 发送方需处理事件

超时事件:如果出现帧丢失,帧延迟、未接收到确认帧等错误,就会回退到上一个确认的帧后面的第一帧位置,重传N帧;

2.4　接收方数据

接收发送帧，并返回ACK

2.5　接收方需处理事件

收到正确帧:收到的帧正确,并且顺序正确;为接收的N帧发送ACK确认信息,将该帧的数据交给上层;

没有收到正确帧:收到错误帧,或顺序错误;接收方为最近的正确的帧发送ACK,丢弃错误帧;

二、具体程序设计

1. 成帧

def send_frame(self, key_pressed)函数：

1.1 检查发送窗口是否已满can_send()

can_send() 方法判断 Sn - Sf（即已发送的帧数量）是否小于窗口大小：

如果窗口已满，无法发送新的帧，返回一个错误信息（Send window is full. Cannot send. Timeout all the frames need resend.），并将 Sn 重置为 Sf（模拟重传所有帧）。

否则，生成帧的序列号 frame_num 为当前的 Sn（发送方的下一个帧序列号）。通过取模操作，Sn 自增并确保它在 0 到 MAX_SEQ - 1 之间。

1.2　根据按键生成不同类型的帧

按键 1：发送有效帧

按键 2：发送损坏的帧。并设置一个超时定时器（5秒后触发重发）。损坏的帧不更新发送方的窗口状态。

按键 3：发送有效帧，但 ACK 丢失

1.3 帧的可视化

调用pygame库实现

2. 发送方

2.1 发送帧启动定时并缓存

2.1.1 定时器

定时器的作用是模拟数据帧发送后的超时等待。如果发送方在规定时间内没有收到接收方的 ACK（确认帧），就会触发重发操作。

2.1.2 代码实现

```
elif key_pressed == 2:
    if not self.can_send():
        self.Sn = self.Sf
    if (self.Timefalg == 0):
        self.Timefalg = 1
        timer = threading.Timer(5.0, self.set_sn_to_sf)
        timer.start()
    print("Sent corrupt frame. Sf and Rn remain unchanged.")
    return "Sent corrupt frame. Sf and Rn unchanged."
```

在发送错误帧时，通过self.Timefalg判断定时器是否开启，用于确保只启动了一个定时器，若未开启，使用Python 的 threading.Timer 启动了一个定时器。定时器的作用是设置一个 5 秒的延迟，当定时器到期时，会调用self.set_sn_to_sf 方法，触发重发操作。

2.2 接收损坏帧休眠

2.2.1 休眠

通过让接收方忽略损坏帧和不发送 ACK，我们实际上模拟了一个接收方“暂停”处理的状态，直到它收到一个有效的帧。

2.2.2 代码实现

```
def receive_ack(self, key_pressed, sn_prev):
    message = ""
    if key_pressed == 2:
        print("Receiver received corrupt frame. No ACK sent.")
        message = "Receiver received corrupt frame. No ACK sent."
    return message
```

在接收者中，定义当收到按键2时，返回信息 Receiver received corrupt frame. No ACK sent. 表示接收方收到损坏帧并没有发送 ACK。这可以看作是模拟了一个“休眠”状态，接收方对损坏帧没有做任何处理。

2.3 累计确认

2.3.1 定义

后退N帧协议中,采用累计确认方式,如果收到一个确认帧,默认已经收到了该帧,及之前的全部帧。

2.3.2 代码实现

```
class Sender:
    def __init__(self):
        self.Sf = 0
        self.Sn = 0
        self.Timefalg = 0
        self.cunt = 1
        self.window_size = WINDOW_SIZE_SENDER
```

使用cunt用于累计统计。

当接收到有效的 ACK 后，发送方会根据接收到的 ACK 来移动

窗口，完成累计确认（Sn值和Sf值）。

```
elif key_pressed == 3:
    if not self.can_send():
        self.Sn = self.Sf
        self.cunt += 1
```

未接收到有效ACK但是接收到帧时，通过增加cunt，标记下如果以后有ACK的话就会以cunt步长移动Sf，从而实现累计确认功能。

2.4 响应ACK：删除帧、停止计时、移动窗口

2.4.1 删除帧

当接收到 ACK 后，发送方会调整窗口起始位置 Sf，从而“删除”已经确认的帧。

2.4.2 停止计时

```
def set_sn_to_sf(self):
    self.Sn = self.Sf
    self.Timefalg = 0
    print("Timeout!!!!!!!!!!!!!!!!!!!!!!")
```

在发送损坏帧时，如果没有接收到 ACK且超过设置时间，定时器会触发重发，即set_sn_to_sf函数，将定时器停止。如果 ACK 被收到，则不触发计时。

2.4.3 移动窗口

```
                else:
    sender.Sf = (sender.Sf + sender.cunt) % MAX_SEQ
    sender.cunt = 1
    message = f"Sender received ACK{ack_frame_num}. Sf incremented."
    message_time = current_time
    animation_phase = 0
```

sender.Sf = (sender.Sf + sender.cunt) % MAX_SEQ：根据 cunt 的值来更新 Sf，从而“删除”已确认的帧，并移动窗口。

sender.cunt = 1：每次更新完 Sf 后，重置 cunt，确保累计确

认的步长从头开始。

2.5 超时重发

```
    timer = threading.Timer(5.0, self.set_sn_to_sf)

def set_sn_to_sf(self):
    self.Sn = self.Sf
    self.Timefalg = 0
    print("Timeout!!!!!!!!!!!!!!!!!!!!!")
```

在发送有效帧时，发送方会启动一个定时器，并设置一个超时时间（例如 5 秒）。如果在超时时间内没有收到 ACK，定时器会触发并执行一个回调函数，该回调函数会调用 set_sn_to_sf()函数，从而重新发送该帧。

3. 接收方

3.1 损坏帧和丢失帧不回复ACK

```
def receive_ack(self, key_pressed, sn_prev):
    message = ""
    if key_pressed == 2:
        print("Receiver received corrupt frame. No ACK sent.")
        message = "Receiver received corrupt frame. No ACK sent."
    return message
elif event.key == pygame.K_3:
    key_pressed = 3
    if animation_phase == 0:
        frame_num = sender.Sn
        result = sender.send_frame(key_pressed)
        message = result
        message_time = current_time
        current_frame_num = frame_num
        current_key_pressed = key_pressed
```

接收损坏的帧：与上面的代码相同，当接收方收到损坏的帧时，打印损坏帧的消息并不发送ACK。

接收丢失的帧：接收方将忽略该帧，不发送ACK。

3.2 窗口内帧标记

在 Receiver 类中，接收窗口的管理和帧的接收是通过 Rn（接收方期望的帧序列号）来控制的。接收方窗口的大小在这里是 1，表示接收方一次只能接收一个帧。每次接收到期望的帧时，Rn 会更新，指向下一个期望接收的帧。

3.3 交付有效帧

```
if animation_phase == 1:
    elapsed = current_time - animation_start_time
    if elapsed < animation_duration:
        progress = elapsed / animation_duration
        cur_x = forward_start_pos[0] + (forward_end_pos[0] -
forward_start_pos[0]) * progress
        cur_y = forward_start_pos[1] + (forward_end_pos[1] -
forward_start_pos[1]) * progress

        moving_rect = pygame.Rect(cur_x - 17, cur_y - 17, 35, 35)
        pygame.draw.rect(WINDOW, ORANGE, moving_rect)
        frame_text = FONT.render(str(current_frame_num), True, BLACK)
        frame_text_rect = frame_text.get_rect(center=(cur_x, cur_y))
        WINDOW.blit(frame_text, frame_text_rect)
    else:
        # 动画结束，判断是否是期待帧
        if receiver.Rn == current_frame_num:
            receiver.Rn = (receiver.Rn + 1) % MAX_SEQ
            message = f"Receiver received frame {current_frame_num}. Rn
incremented."
            # 帧到达后进入网络层动画
            animation_phase = 3
            animation_start_time = current_time
            network_start_pos = forward_end_pos
            network_end_pos = NETWORK_LAYER_CENTER
```

动画阶段（animation_phase == 1）控制帧的传输过程：当发送方将帧发送到接收方时，接收方会检查接收到的帧是否是期望的帧（即 Rn == current_frame_num）。如果是，接收方就会交付该帧，并更新 Rn 为下一个期望的帧（(receiver.Rn + 1) %

MAX_SEQ）。

animation_phase = 3：当帧成功交付给网络层时，进入网络层的动画阶段。

帧到达网络层：通过 network_start_pos 和 network_end_pos，帧会以动画的形式从接收方的队列移动到网络层。

3.4 发送ACK

```
def receive_ack(self, key_pressed, sn_prev):
    message = ""
    if key_pressed == 2:
        print("Receiver received corrupt frame. No ACK sent.")
        message = "Receiver received corrupt frame. No ACK sent."
    return message
```

接收方的 ACK 发送是通过 Receiver 类中的 receive_ack 方法实现的。该方法接收一个 key_pressed 参数来决定当前接收到的帧类型（是否是损坏的帧等）。

```
elif animation_phase == 2:
    elapsed = current_time - animation_start_time
    if elapsed < animation_duration:
        progress = elapsed / animation_duration
        cur_x = ack_start_pos[0] + (ack_end_pos[0] - ack_start_pos[0]) *
progress
        cur_y = ack_start_pos[1] + (ack_end_pos[1] - ack_start_pos[1]) *
progress

        moving_rect = pygame.Rect(cur_x - 17, cur_y - 17, 35, 35)
        pygame.draw.rect(WINDOW, GREEN, moving_rect)
        frame_text = FONT.render(f"ACK{ack_frame_num}", True, BLACK)
        frame_text_rect = frame_text.get_rect(center=(cur_x, cur_y))
        WINDOW.blit(frame_text, frame_text_rect)
    else:
        sender.Sf = (sender.Sf + sender.cunt) % MAX_SEQ
        sender.cunt = 1
        message = f"Sender received ACK{ack_frame_num}. Sf incremented."
```

```
        message_time = current_time
        animation_phase = 0
```

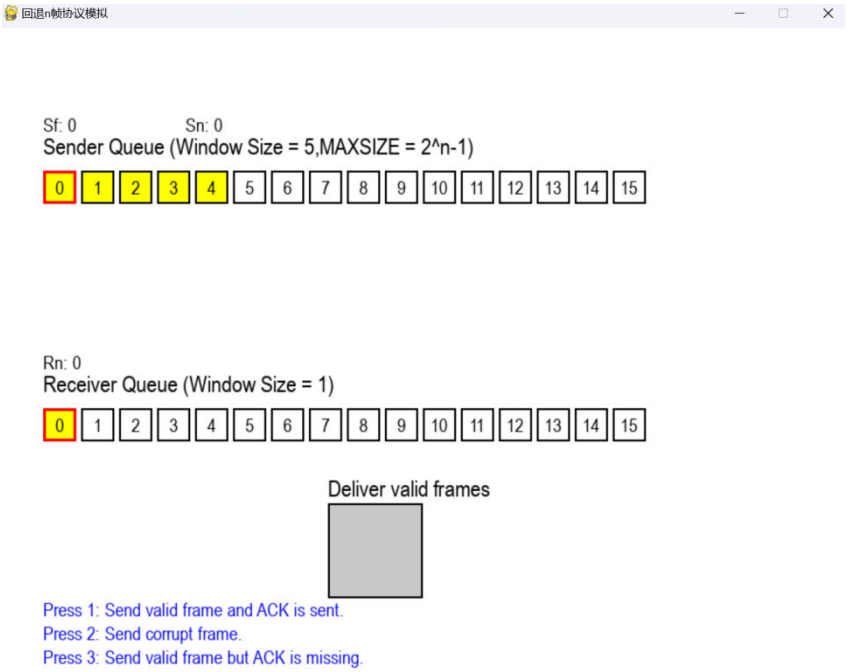
根据接收到的帧的情况动画展示决定是否发送 ACK。

进入 ACK 发送阶段：通过控制 animation_phase == 2，接收方开始发送 ACK 帧，动画显示 ACK 从接收方发送到发送方。

ACK 完成：ACK 动画完成后，发送方收到 ACK 并更新其窗口，继续发送下一个帧。

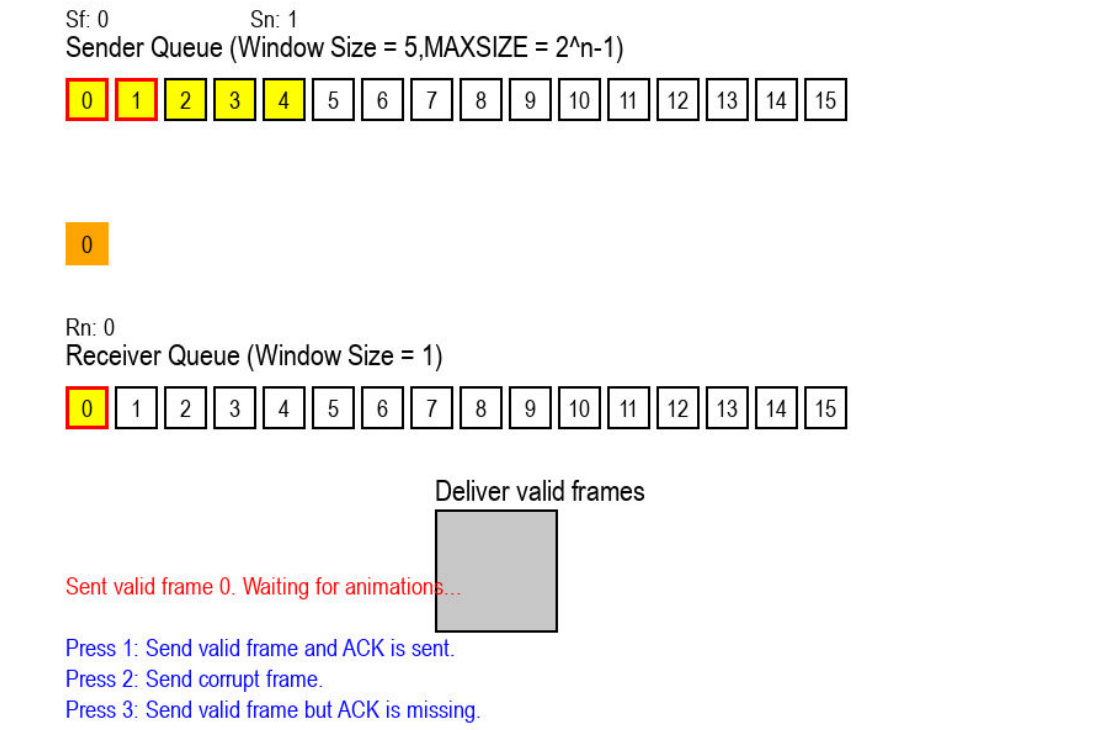
三、可视化界面运行演示

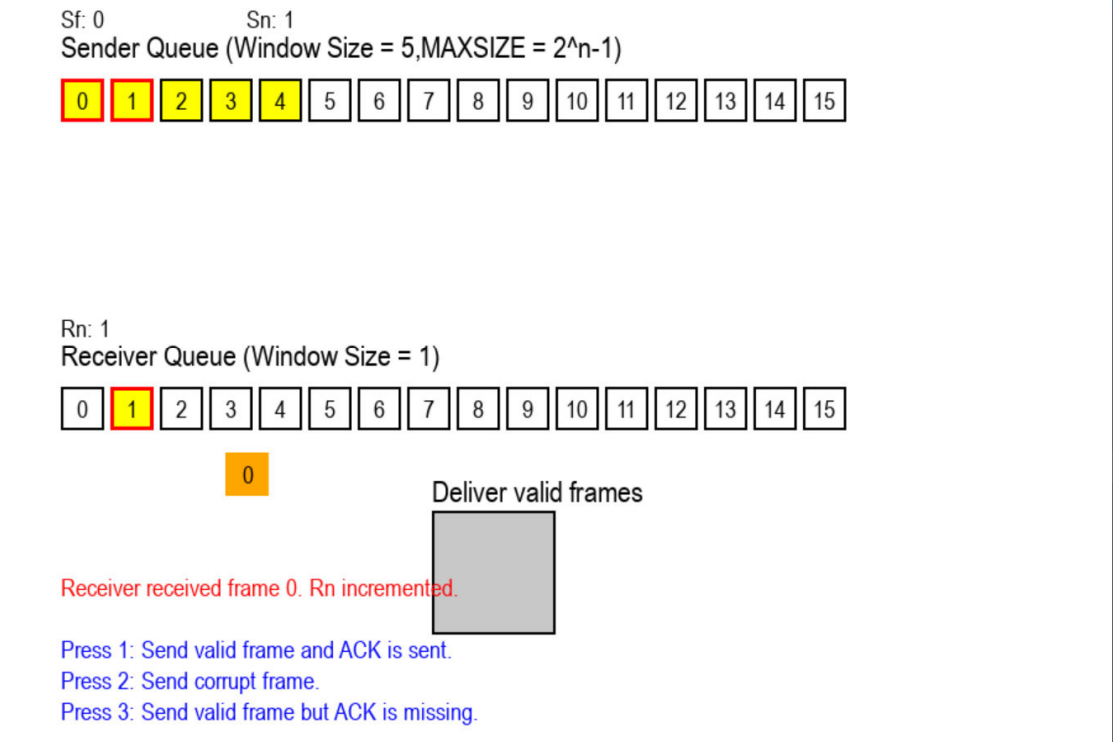
1. 界面展示



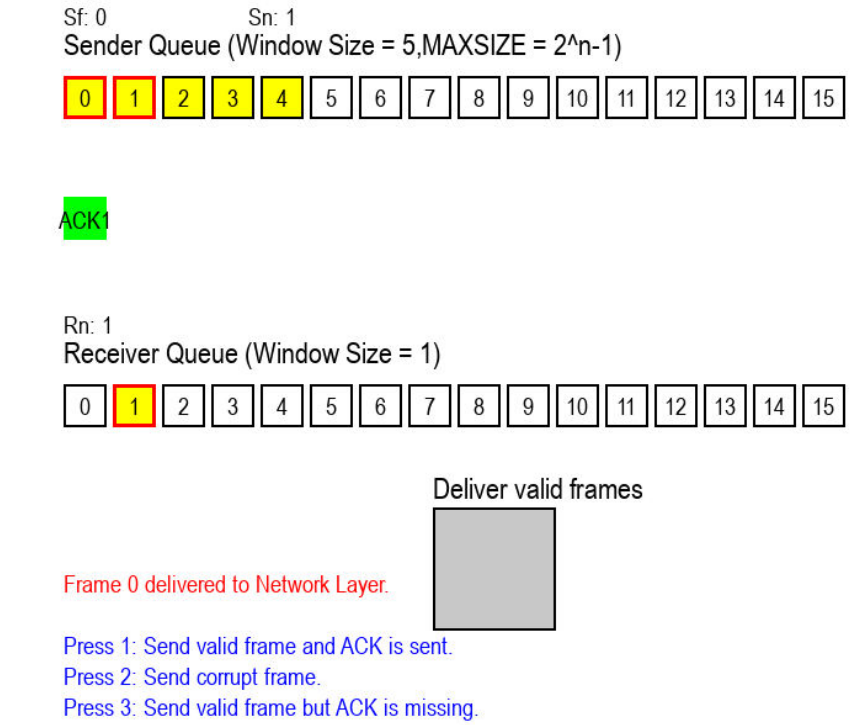
2. 按键为1，发送正确帧

2.1 上传数据帧到网络层

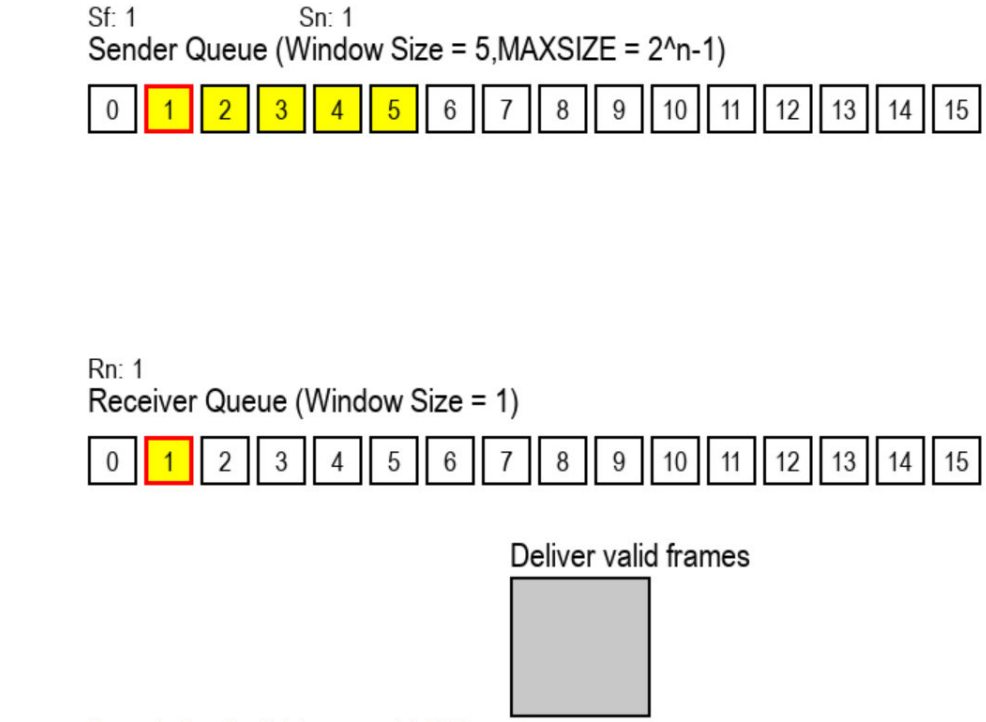

Sf: 0
Sn: 1
Sender Queue (Window Size = 5,MAXSIZE = 2^n-1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0
Rn: 0
Receiver Queue (Window Size = 1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Deliver valid frames
Sent valid frame 0. Waiting for animations...
Press 1: Send valid frame and ACK is sent.
Press 2: Send corrupt frame.
Press 3: Send valid frame but ACK is missing.


Sf: 0
Sn: 1
Sender Queue (Window Size = 5,MAXSIZE = 2^n-1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Rn: 1
Receiver Queue (Window Size = 1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0
Deliver valid frames
Receiver received frame 0. Rn incremented.
Press 1: Send valid frame and ACK is sent.
Press 2: Send corrupt frame.
Press 3: Send valid frame but ACK is missing.

2.2 返回ACK

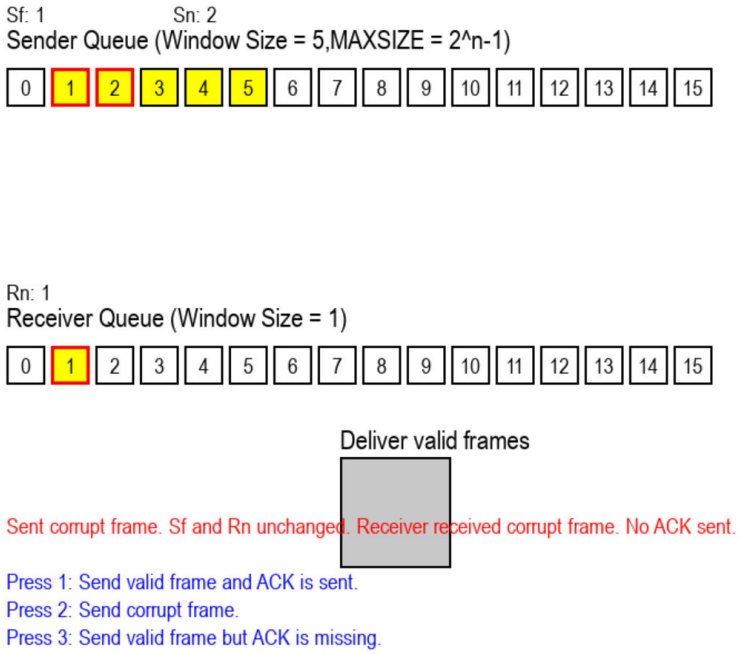

Sf: 0
Sn: 1
Sender Queue (Window Size = 5,MAXSIZE = 2^n-1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
ACK1
Rn: 1
Receiver Queue (Window Size = 1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Deliver valid frames
Frame 0 delivered to Network Layer.
Press 1: Send valid frame and ACK is sent.
Press 2: Send corrupt frame.
Press 3: Send valid frame but ACK is missing.

2.3 更新Sf，Sn


Sf: 1
Sn: 1
Sender Queue (Window Size = 5,MAXSIZE = 2^n-1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Rn: 1
Receiver Queue (Window Size = 1)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Deliver valid frames
Press 1: Send valid frame and ACK is sent.
Press 2: Send corrupt frame.
Press 3: Send valid frame but ACK is missing.

3. 按键为2

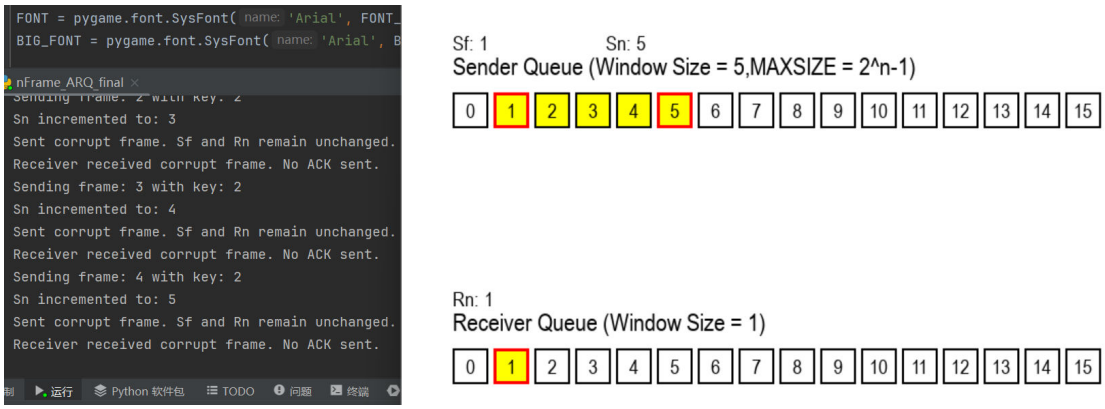
3.1 在传递过程中帧丢失或者损坏



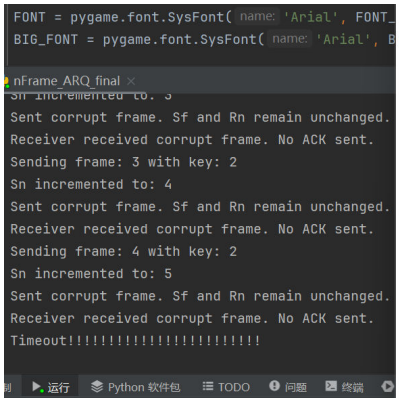
显示报错，并开启计时

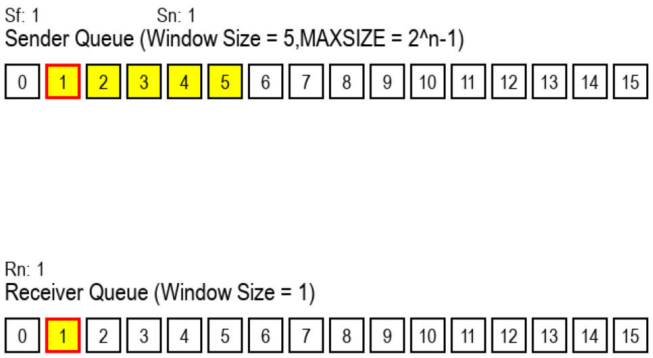
3.2 计时器报错并实现重发

重发前



重发后

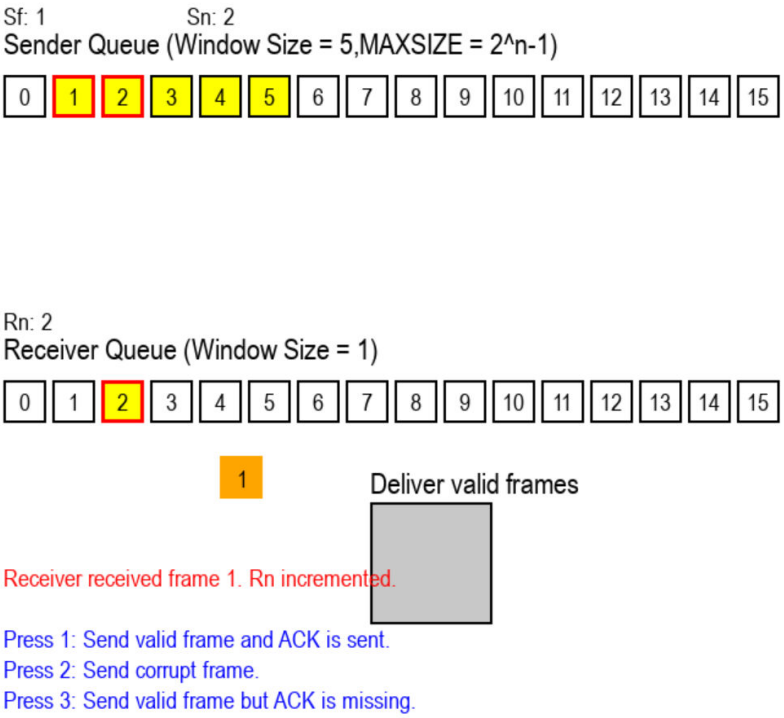




输出Time out！！！！！！，同时Sn重置

4. 按键为3

帧被传送到网络层，但是由于无ACK，Sf不改变



实现累计确认功能：

模拟发送帧1、帧2，其ACK均丢失，发送帧3后Receiver回复ACK3实现累计缺认

Sf: 1 Sn: 3

Sender Queue (Window Size = 5,MAXSIZE = 2^n-1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Rn: 3

Receiver Queue (Window Size = 1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Deliver valid frames

Press 1: Send valid frame and ACK is sent.

Press 2: Send corrupt frame.

Press 3: Send valid frame but ACK is missing.

Sf: 1 Sn: 4

Sender Queue (Window Size = 5,MAXSIZE = 2^n-1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

ACK4

Rn: 4

Receiver Queue (Window Size = 1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

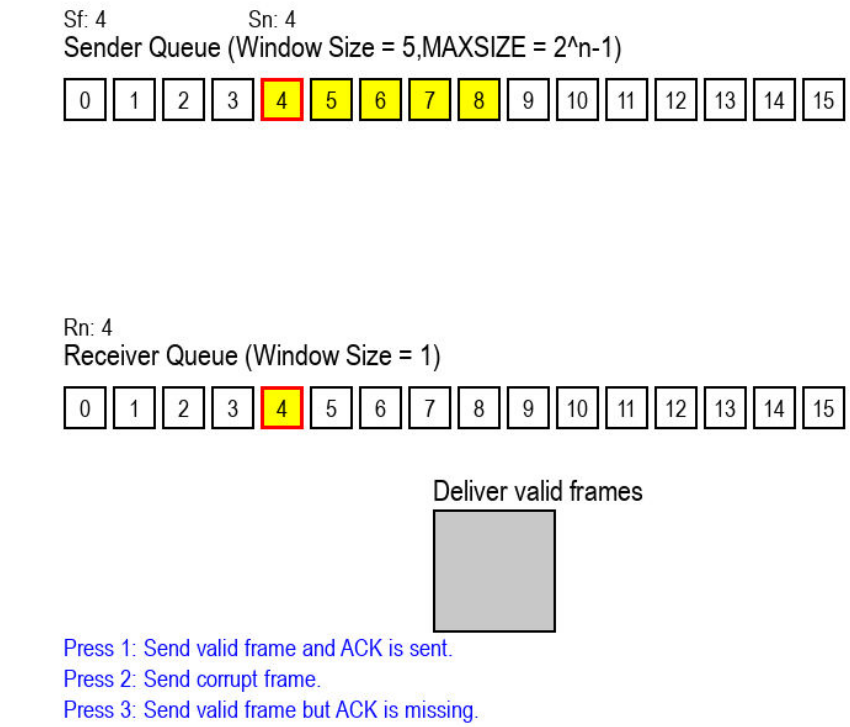
Deliver valid frames

Frame 3 delivered to Network Layer.

Press 1: Send valid frame and ACK is sent.

Press 2: Send corrupt frame.

Press 3: Send valid frame but ACK is missing.



当Receiver接收到不想要的帧时不移动窗口：