一个TCP滑动窗口演示程序

介绍

slide_window.py 是一个TCP滑动窗口演示程序，使用 *matplotlib* 库进行图像绘制，可以模拟TCP协议的滑动窗口协议。
运行时，需要确保 *matplotlib* 库已经安装:

```
pip install matplotlib
```

用法

在 plot_tcp_with_rectangle_movement 函数的开头有参数设置部分，可以根据需要修改参数。

```
# 参数
total_packets = 40 # 总报文数
window_size = 7 # 窗口大小
data_transfer_time = 0.01 # 数据传输时间
loss_rate = 20 # 丢包率
timeout = 25 # 超时时间
timer = [0] * (total_packets + window_size) # 超时计时器
frame1 = 0
sent_status = [False] * (total_packets + window_size) # 发送状态
ack_sent_status = [False] * (total_packets + window_size) # 确认发送状态
received_status = [False] * (total_packets + window_size) # 接收状态
ack_status = [False] * (total_packets + window_size) # 确认状态
```

编译执行后可以进行滑动窗口演示。

在 *samples* 文件夹下有两个示例，文件名标记了两个程序的丢包率。

原理

动画实现

使用 *matplotlib* 库的动画功能可以实现动态的窗口绘制过程。
用 *FuncAnimation* 函数实现动画，传入 *update* 函数作为动画的更新函数，并传入 *frames* 参数作为帧数。
使用 *matplotlib* 库绘制每一帧，不断刷新画面实现动画效果。

```
ani = FuncAnimation(fig, update, frames= 20 * total_packets + 10000,
interval=data_transfer_time * 1000, repeat=False)
```

框架绘制

在 plot_tcp_with_rectangle_movement 函数中，首先绘制了接收方和发送方的缓存区，以及相关的状态信息。

```
# 绘制背景
ax.clear()
ax.set_xlim(-4, total_packets + 1)
ax.set_ylim(-4, 5)
ax.axis('off')
ax.text(-3.7, 2.75, 'sender', ha='center', va='center', fontsize=16)
ax.text(-4.2, -1.5,'receiver', ha='center', va='center', fontsize=16)
ax.text(-1, 2.25, 'sent', ha='center', va='center', fontsize=12)
ax.text(-1, 2.75, 'ack', ha='center', va='center', fontsize=12)
ax.text(-1, -1.5, 'recv', ha='center', va='center', fontsize=12)
ax.text(-1, 3.25, 'timer', ha='center', va='center', fontsize=12)
ax.text((total_packets + 1 - 5) / 2, 4, 'time = ' + str(frame1) + '   timeout
= ' + str(timeout) + '   loss rate = ' + str(loss_rate), ha='center',
va='center', fontsize=12)

# 绘制缓存
for i in range(total_packets):
    color3 = 'pink' if timer[i] == -1 else 'white'
    ax.add_patch(patches.Rectangle((i, 3), 1, 0.5, edgecolor='black',
facecolor=color3))
    ax.text(i + 0.5, 3.25, timer[i], ha='center', va='center', fontsize=8)
    color1 = 'royalblue' if sent_status[i] else 'lightblue'
    ax.add_patch(patches.Rectangle((i, 2), 1, 0.5, edgecolor='black',
facecolor=color1))
    color2 = 'green' if ack_status[i] else 'lightgreen'
    ax.add_patch(patches.Rectangle((i, 2.5), 1, 0.5, edgecolor='black',
facecolor=color2))
    ax.text(i + 0.5, 2.25, str(i + 1), ha='center', va='center', fontsize=8)
for i in range(total_packets):
    color = 'green' if received_status[i] else 'white'
    ax.add_patch(patches.Rectangle((i, -2), 1, 1, edgecolor='black',
facecolor=color))
    ax.text(i + 0.5, -1.5, str(i + 1), ha='center', va='center', fontsize=8)
fg = True
for i in range(total_packets):
    if not ack_status[i]:
        fg = False
        break
if fg:
    ax.text((total_packets + 1 - 5) / 2, 0, 'transmission completed',
ha='center', va='center', fontsize=17)
    return
frame1 += 1
```

窗口绘制过程

用 window_size 参数控制窗口大小，窗口大小决定了滑动窗口协议的窗口大小。
window_start 和 recv_window_start 分别表示发送方窗口的起始位置和接收方窗口的起始位置，窗口的大小

由 window_size 参数决定。

```
# 绘制窗口
window_rect = patches.Rectangle((window_start, 2), window_size, 1,
edgecolor='red', facecolor='none', linewidth=2)
ax.add_patch(window_rect)
ack_window_rect = patches.Rectangle((recv_window_start, -2), window_size, 1,
edgecolor='red', facecolor='none', linewidth=2)
ax.add_patch(ack_window_rect)
```

超时定时器的维护

用帧数 $frame1$ 记录作为时间的度量标准，每一帧都会更新超时计时器。
用 $timer$ 数组记录每个报文的超时计时器，超时计时器的超时时间由 $timeout$ 参数决定。
如果某字节已经发送但没有收到确认，则超时计时器加一。
如果超时计时器超过 $timeout$，则认为该字节丢失，则将该报文的发送状态设置为 $False$,等待发送方重传。

```
# 维护超时计时器
for i in range(total_packets):
    if (not ack_status[i]) and (sent_status[i]):
        timer[i] += 1
for i in range(total_packets):
    if timer[i] > timeout:
        sent_status[i] = False
        ack_sent_status[i] = False
        timer[i] = 0
```

发送方的发送过程

发送方每次检测发送窗口中是否包含未发送的报文，如果有，则将窗口中第一个未发送的报文发送出去。
发送方发送报文时，将该报文的发送状态设置为 $True$，并将该报文的超时计时器设置为 0。

```
# 发送报文
for i in range(window_start, window_start + window_size):
    if not sent_status[i]:
        new_moving_packets.append((i, 2, 'royalblue'))
        timer[i] = 0
        sent_status[i] = True
        break
```

模拟报文的传输过程

用 moving_packets 数组记录当前正在传输的报文，每一帧都会更新该数组。 如果报文颜色为蓝色，则表示该报文正在发送，则向下移动；如果报文颜色为绿色，则表示该报文已经接收，则向上移动。

每次随机生成一个数字，如果数字小于丢包率，则认为该报文丢失，则跳过该帧。
如果没有发生丢包，则更新报文的位置，并存入 new_moving_packets 数组。
如果报文到达接收方窗口，则将该报文的接收状态设置为 $True$。
如果报文到达发送方窗口，则将该报文的确认状态设置为 $True$，并将该报文的超时计时器设置为 -1。
遍历一遍 moving_packets 数组，将其中的报文移动到新的位置，并更新 new_moving_packets 数组。
最后清空 moving_packets 数组，将 new_moving_packets 数组中的报文加入到 moving_packets 数组。

```
new_moving_packets = []
# 模拟报文移动过程
for x, y, color in moving_packets:
    if color == 'royalblue':# 蓝色报文向下移动
        new_y = y - 0.4 if y > -2 else -2
        if new_y > -2:
            if random.randint(1, 1000) <= loss_rate:
                continue
            new_moving_packets.append((x, new_y, color))
            ax.add_patch(patches.Rectangle((x, new_y), 1, 1,
edgecolor='black', facecolor=color))
        else:
            received_status[x] = True
            if not ack_sent_status[x]:
                new_moving_packets.append((x, -2, 'green'))
                ack_sent_status[x] = True
    elif color == 'green':# 绿色报文向上移动
        new_y = y + 0.4 if y < 2 else 2
        if new_y < 2:
            if random.randint(1, 1000) <= loss_rate:
                continue
            new_moving_packets.append((x, new_y, color))
            ax.add_patch(patches.Rectangle((x, new_y), 1, 1,
edgecolor='black', facecolor=color))
        else:
            ack_status[x] = True
            timer[x] = -1
moving_packets.clear()
moving_packets.extend(new_moving_packets)
```

窗口位置更新

窗口位置由 window_start 和 recv_window_start 两个变量控制，每一帧都会更新窗口位置。
每次检查窗口开始的第一个报文是否被确认，如果确认，则窗口开始位置向后移动。

```
# 移动窗口
if ack_status[window_start] and window_start + window_size < total_packets:#
发送窗口
    window_start += 1
if received_status[recv_window_start] and recv_window_start + window_size <
total_packets: # 接收窗口
    recv_window_start += 1
```