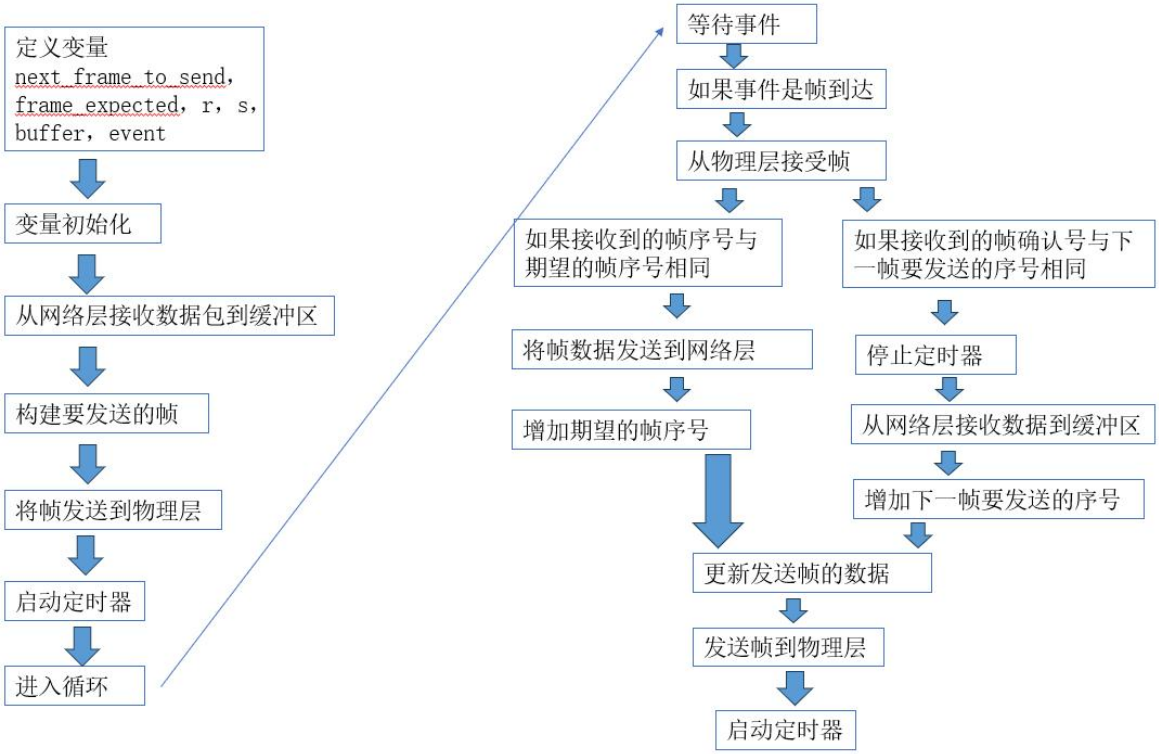
2152118 史君宝 计算机网络 第二次课程报告

协议 4：

```
typedef enum {frame_arrival, cksum_err, timeout} event_type;
void protocol4 (void)
{    seq_nr next_frame_to_send;
     seq_nr frame_expected;
     frame r, s;
     packet buffer;
     event_type event;
     next_frame_to_send = 0;
     frame_expected = 0;
     from_network_layer(&buffer);
     s.info = buffer;
     s.seq = next_frame_to_send;
     s.ack = 1 - frame_expected;
     to_physical_layer(&s);
     start_timer(s.seq);
```

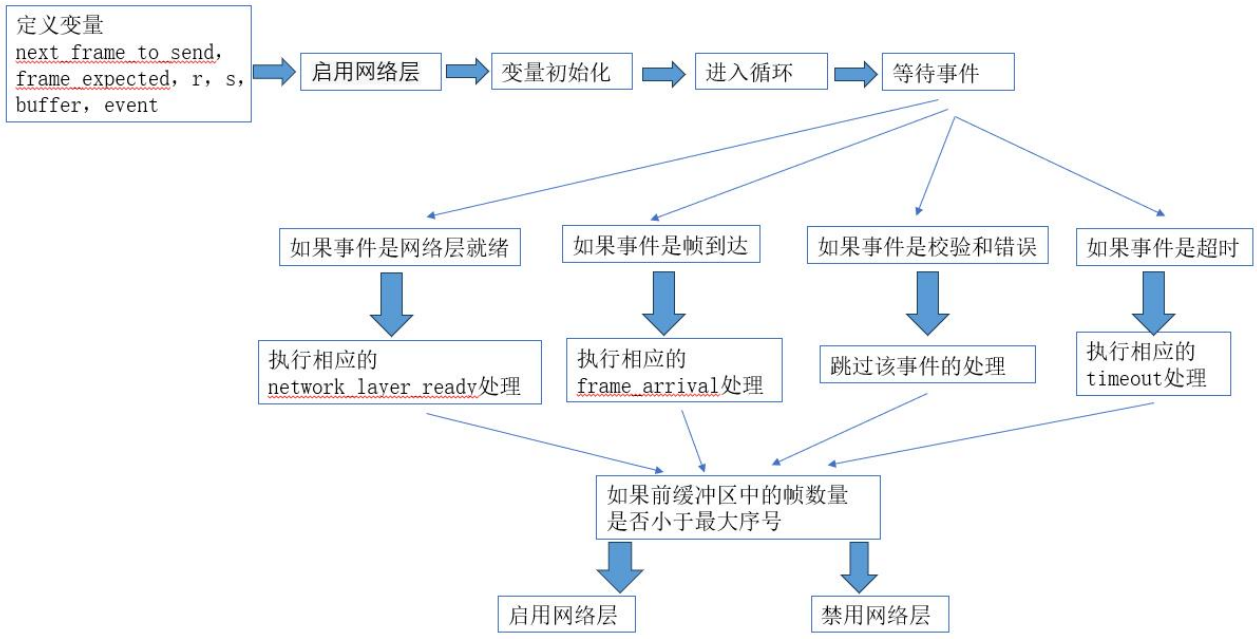
```
while (true)
{    wait_for_event(&event);
     if (event == frame_arrival)
     {    from_physical_layer(&r);
          if (r.seq == frame_expected)
          {   to_network_layer(&r.info);
              inc(frame_expected);        }
          if (r.ack == next_frame_to_send)
          {   stop_timer(r.ack);
              from_network_layer(&buffer);
              inc(next_frame_to_send);   }      }
     s.info = buffer;
     s.seq = next_frame_to_send;
     s.ack = 1 - frame_expected;
     to_physical_layer(&s);
     start_timer(s.seq);    }
```



协议 5：

```
void protocol5(void)
{ ……
   enable_network_layer();
   ack_expected = 0;
   next_frame_to_send = 0;
   frame_expected = 0;
   nbuffered = 0;
```

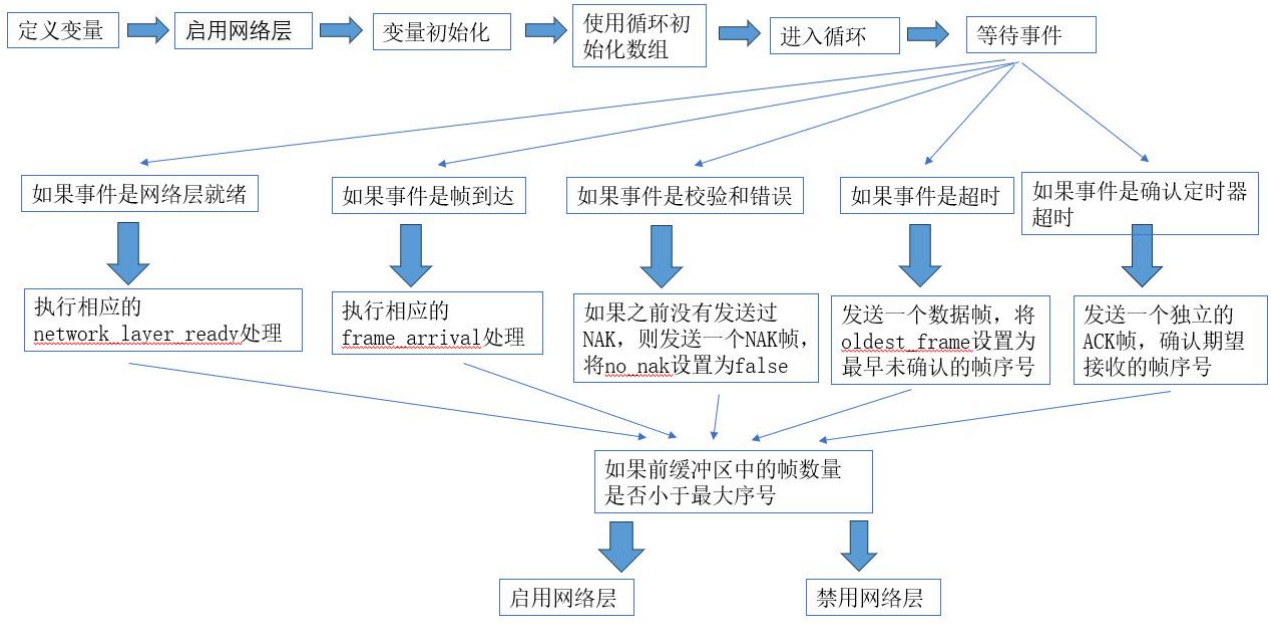
```
while (true)
{   wait_for_event(&event);
   switch(event)
   {    case network_layer_ready:
                network_layer_ready处理
        case frame_arrival: frame_arrival处理
        case cksum_err:break;
        case timeout: timeout处理
   }
   if (nbuffered < MAX_SEQ)
        enable_network_layer();
   else
        disable_network_layer(); } }
```



协议 6：

```
void protocol6(void)
{ ……
  enable_network_layer();
  ack_expected = 0;
  next_frame_to_send = 0;
  frame_expected = 0;
  too_far = NR_BUFS;
  nbuffered = 0;
  for (i=0;i<NR_BUFS;i++)
       arrived[i] = false;
```

```
while (true)
{   wait_for_event(&event);
    switch(event)
    {   case network_layer_ready: 发送数据处理
        case frame_arrival: 帧到达处理
        case cksum_err:收到一个坏帧且没有发过nak，则发nak
           if (no_nak) send_frame(nak, 0, frame_expected, out_buf);
        case timeout: 数据帧超时重发
           send_frame(data, oldest_frame, frame_expected, out_buf);
        case ack_timeout: 确认定时器超时，发一单独的ack
           send_frame(ack,0,frame_expected, out_buf);
    }
    if (nbuffered <NR_BUFS)
        enable_network_layer();
    else
        disable_network_layer(); } }
```



协议 4、5 和 6 三个协议都是用于可靠数据传输的协议，它们在实现数据可靠性方面有一些区别。下面对协议的效率进行分析：

（1）协议 4 效率：

协议 4 是最简单的可靠传输协议，发送方发送一个数据帧后必须等待对应的确认帧才能发送下一个数据帧。由于发送方必须等待确认帧，这导致了低效率，发送方的利用率较低。发送方和接收方之间的往返时间较长时，停等协议的效率更低，传输延迟较高。

（2）协议 5 效率：

协议 5 引入了滑动窗口机制，允许发送方连续发送多个帧而不需要等待每个帧的确认。发送方可以利用窗口内的多个帧发送，而不需要等待每个帧的确认，从而提高了发送方的利用率。窗口大小的选择会影响协议 5 的效率。如果窗口大小太小，发送方的利用率较低；如果窗口大小太大，可能会导致网络拥塞和带宽浪费。

（3）协议 6 效率：

协议 6 进一步改进了协议 5，引入了选择重传机制，允许发送方仅重传丢失或损坏的帧，而不是重传整个窗口。发送方和接收方之间的确认和重传操作更加灵活，可以提高协议 6 的效率。选择重传协议通过减少不必要的重传操作，减少了网络拥塞和带宽浪费的可能性，提高了传输效率。

总体而言，协议 4 是最简单但效率最低的协议，协议 5 在发送方的利用率上有所提升，而协议 6 引入了选择重传机制，进一步提高了效率。协议的效率还受到网络延迟、带宽和窗口大小等因素的影响，因此在实际应用中需要根据具体情况进行选择和调优。