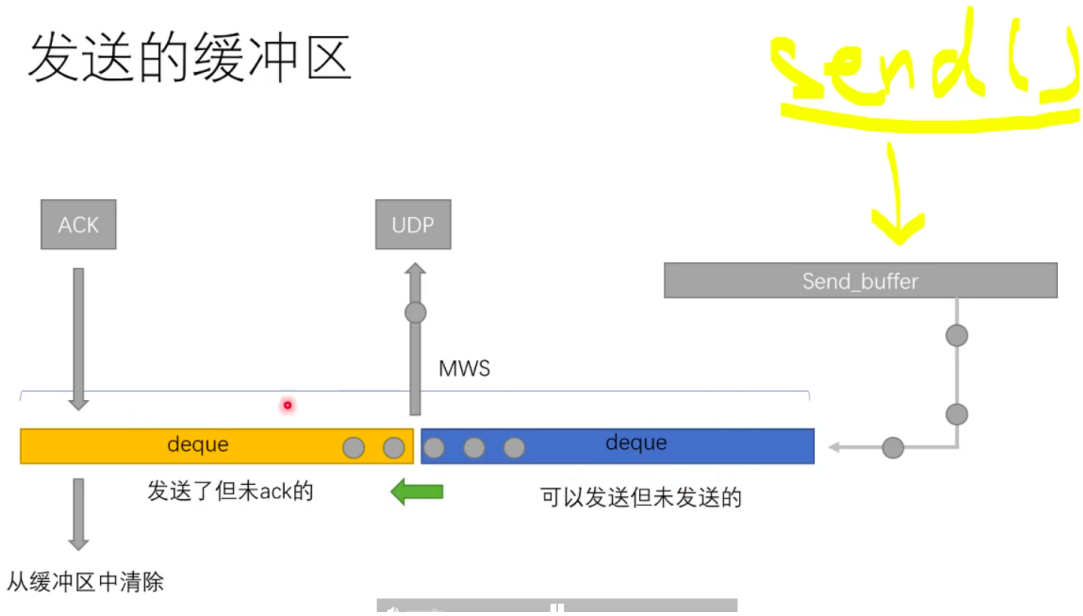
状态机—recv 线程

双端的队列，缓冲区，---发送窗口， （python 数据结构：双端队列保证线程安全，使用 list，线程不安全，）



为什么使用deque

- Deque 支持线程安全，内存高效添加(append)和弹出(pop)，从两端都可以，两个方向的大概开销都是 O(1) 复杂度。
- 虽然 list 对象也支持类似操作，不过这里优化了定长操作和 pop(0) 和 insert(0, v) 的开销。它们引起 O(n) 内存移动的操作，改变底层数据表达的大小和位置。

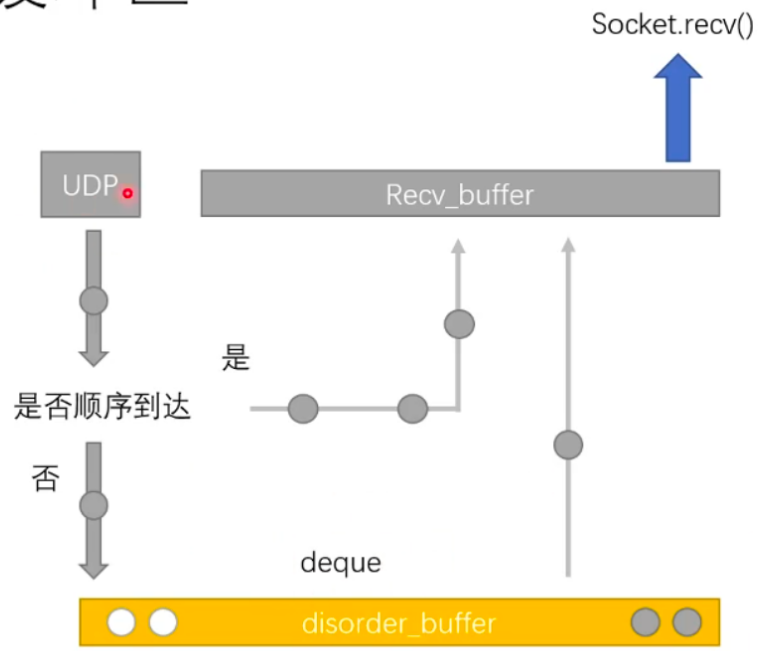
deque

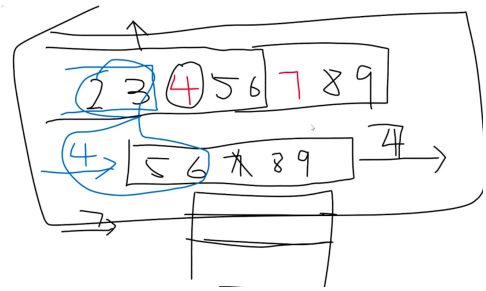
```
Q = deque()

q.append(element)
q.appendleft(element)
Element = q.pop()
Element = q.popleft()
```

仍使用双端队列，
处理无序的包。（丢失的包），---重复确认的 ack
重传的 segment 和无序的包，判断是否可以组成有序的

接受的缓冲区





文档解析

- Receiver 的要求

- 1. receiver需要接受两个参数：receiver_port（开放的udp端口号）和 FileReceiver.txt（用于保存传输过来的文本数据的文件的文件名）

- 2. receiver应在接收到数据段后立即生成ACK，不用实现延迟ACK，确认段的格式必须与PTP数据段完全相似，但它不包含任何有效载荷。

- 3.receiver缓冲无序的到达段。

- 4. receiver应该首先在receiver_port 上打开一个UDP 侦听套接字，然后等待来自发送方的段到达。 发送方发送的第一个段是一个 SYN 段，接收方应该回复一个 SYNACK 段。

定时器

Timer Objects

This class represents an action that should be run only after a certain amount of time has passed — a timer. `Timer` is a subclass of `Thread` and as such also functions as an example of creating custom threads.

Timers are started, as with threads, by calling their `start()` method. The timer can be stopped (before its action has begun) by calling the `cancel()` method. The interval the timer will wait before executing its action may not be exactly the same as the interval specified by the user.

For example:

```
def hello():
    print("hello, world")

t = Timer(30.0, hello)
t.start()  # after 30 seconds, "hello, world" will be printed
```

class threading.**Timer**(*interval*, *function*, *args=None*, *kwargs=None*)

Create a timer that will run *function* with arguments *args* and keyword arguments *kwargs*, after *interval* seconds have passed. If *args* is `None` (the default) then an empty list will be used. If *kwargs* is `None` (the default) then an empty dict will be used.

Changed in version 3.3: changed from a factory function to a class.

cancel()

Stop the timer, and cancel the execution of the timer's action. This will only work if the timer is still in its waiting stage.

文件的读写和字符串格式化

数字	格式	输出	描述
3.1415926	{:.2f}	3.14	保留小数点后两位
3.1415926	{:+.2f}	+3.14	带符号保留小数点后两位
-1	{:+.2f}	-1.00	带符号保留小数点后两位
2.71828	{:.0f}	3	不带小数
5	{:0>2d}	05	数字补零 (填充左边, 宽度为2)
5	{:x<4d}	5xxx	数字补x (填充右边, 宽度为4)
10	{:x<4d}	10xx	数字补x (填充右边, 宽度为4)
1000000	{:,}	1,000,000	以逗号分隔的数字格式
0.25	{:.2%}	25.00%	百分比格式
1000000000	{:.2e}	1.00e+09	指数记法
13	{:>10d}	13	右对齐 (默认, 宽度为10)
13	{:<10d}	13	左对齐 (宽度为10)
13	{:^10d}	13	中间对齐 (宽度为10)

11	`'{:b}'.format(11)`
	`'{:d}'.format(11)`
	`'{:o}'.format(11)`
	`'{:x}'.format(11)`
	`'{:#x}'.format(11)`
	`'{:#X}'.format(11)`
		1011
		11
		13
		b
		0xb
		0XB
			进制

^、<、> 分别是居中、左对齐、右对齐，后面带宽度，: 号后面带填充的字符，只能是一个字符，不指定则默认是用空格填充。

+ 表示在正数前显示 +，负数前显示 -； （空格）表示在正数前加空格

b、d、o、x 分别是二进制、十进制、八进制、十六进制。

此外我们可以使用大括号 {} 来转义大括号，如下实例：

段的转换

- 1. 使用Struct
- 2. 使用json
- 3. 使用pickle

pickle

可以被封存/解封的对象

下列类型可以被封存：

- None、True 和 False
- 整数、浮点数、复数
- str、byte、bytearray
- 只包含可封存对象的集合，包括 tuple、list、set 和 dict
- 定义在模块最外层的函数（使用 def 定义，lambda 函数则不可以）
- 定义在模块最外层的内置函数
- 定义在模块最外层的类
- 某些类实例，这些类的 __dict__ 属性值或 __getstate__() 函数的返回值可以被封存（详情参阅 封存类实例 这一段）。

尝试封存不能被封存的对象会抛出 PicklingError 异常，异常发生时，可能有部分字节已经被写入指定文件中。尝试封存递归层级很深的对象时，可能会超出最大递归层级限制，此时会抛出 RecursionError 异常，可以通过 sys.setrecursionlimit() 调整递归层级，不过请谨慎使用这个函数，因为可能会导致解释器崩溃。