

讲师介绍--专业来自专注和实力



Darren老师

曾供职于国内知名半导体公司（珠海扬智/深圳联发科），曾在某互联网公司担任音视频通话项目经理。主要从事音视频驱动、多媒体中间件、流媒体服务器的开发，开发过即时通讯+音视频通话的大型项目，在音视频、C/C++/GO Linux服务器领域有丰富的实战经验。

课程安排

1. 如果做到可靠性传输
2. UDP与TCP，我们如何选择
3. UDP如何可靠，KCP协议在哪些方面有优势
4. KCP协议精讲

■ 1 如果做到可靠性传输

- ACK机制
- 重传机制
- 序号机制
- 重排机制
- 窗口机制

2 UDP与TCP，我们如何选择

选项	UDP	TCP
是否连接	无连接	面向连接
是否可靠	不可靠传输，不使用流量控制和拥塞控制	可靠传输，使用流量控制和拥塞控制
连接对象个数	支持一对一，一对多，多对一和多对多交互通信	只能是一对一通信
传输方式	面向报文	面向字节流
首部开销	首部开销小，仅8字节	首部最小20字节，最大60字节
适用场景	适用于实时应用（IP电话、视频会议、直播等） 游戏行业、物联网行业	适用于要求可靠传输的应用， 例如文件传输

3 UDP如何可靠，KCP协议在哪些方面有优势

以10%-20%带宽浪费的代价换取了比 TCP快30%-40%的传输速度。

RTO翻倍vs不翻倍：

TCP超时计算是 $RTO \times 2$ ，这样连续丢三次包就变成 $RTO \times 8$ 了，十分恐怖，而KCP启动快速模式后不 $\times 2$ ，只是 $\times 1.5$ （实验证明1.5这个值相对比较好），提高了传输速度。

选择性重传 vs 全部重传：

TCP丢包时会全部重传从丢的那个包开始以后的数据，KCP是选择性重传，只重传真正丢失的数据包。

快速重传：

发送端发送了1,2,3,4,5几个包，然后收到远端的ACK: 1, 3, 4, 5，当收到ACK3时，KCP知道2被跳过1次，收到ACK4时，知道2被跳过了2次，此时可以认为2号丢失，不用等超时，直接重传2号包，大大改善了丢包时的传输速度。

3 UDP如何可靠，KCP协议在哪些方面有优势2

以10%-20%带宽浪费的代价换取了比 TCP快30%-40%的传输速度。

延迟ACK vs 非延迟ACK:

TCP为了充分利用带宽，延迟发送ACK（NODELAY都没用），这样超时计算会算出较大 RTT时间，延长了丢包时的判断过程。KCP的ACK是否延迟发送可以调节。

UNA vs ACK+UNA:

ARQ模型响应有两种，UNA（此编号前所有包已收到，如TCP）和ACK（该编号包已收到），光用UNA将导致全部重传，光用ACK则丢失成本太高，以往协议都是二选其一，而 KCP协议中，除去单独的 ACK包外，所有包都有UNA信息。

非退让流控:

KCP正常模式同TCP一样使用公平退让法则，即发送窗口大小由：发送缓存大小、接收端剩余接收缓存大小、丢包退让及慢启动这四要素决定。但传送及时性要求很高的小数据时，可选择通过配置跳过后两步，仅用前两项来控制发送频率。以牺牲部分公平性及带宽利用率之代价，换取了开着BT都能流畅传输的效果。

4 KCP精讲-名词说明

■ kcp官方: <https://github.com/skywind3000/kcp>

■ 名词说明

用户数据: 应用层发送的数据, 如一张图片2Kb的数据

MTU: 最大传输单元。即每次发送的最大数据

RTO: Retransmission TimeOut, 重传超时时间。

cwnd:congestion window, 拥塞窗口, 表示发送方可发送多少个KCP数据包。
与接收方窗口有关, 与网络状况(拥塞控制)有关, 与发送窗口大小有关。

rwnd:receiver window,接收方窗口大小, 表示接收方还可接收多少个KCP数据包

snd_queue:待发送KCP数据包队列

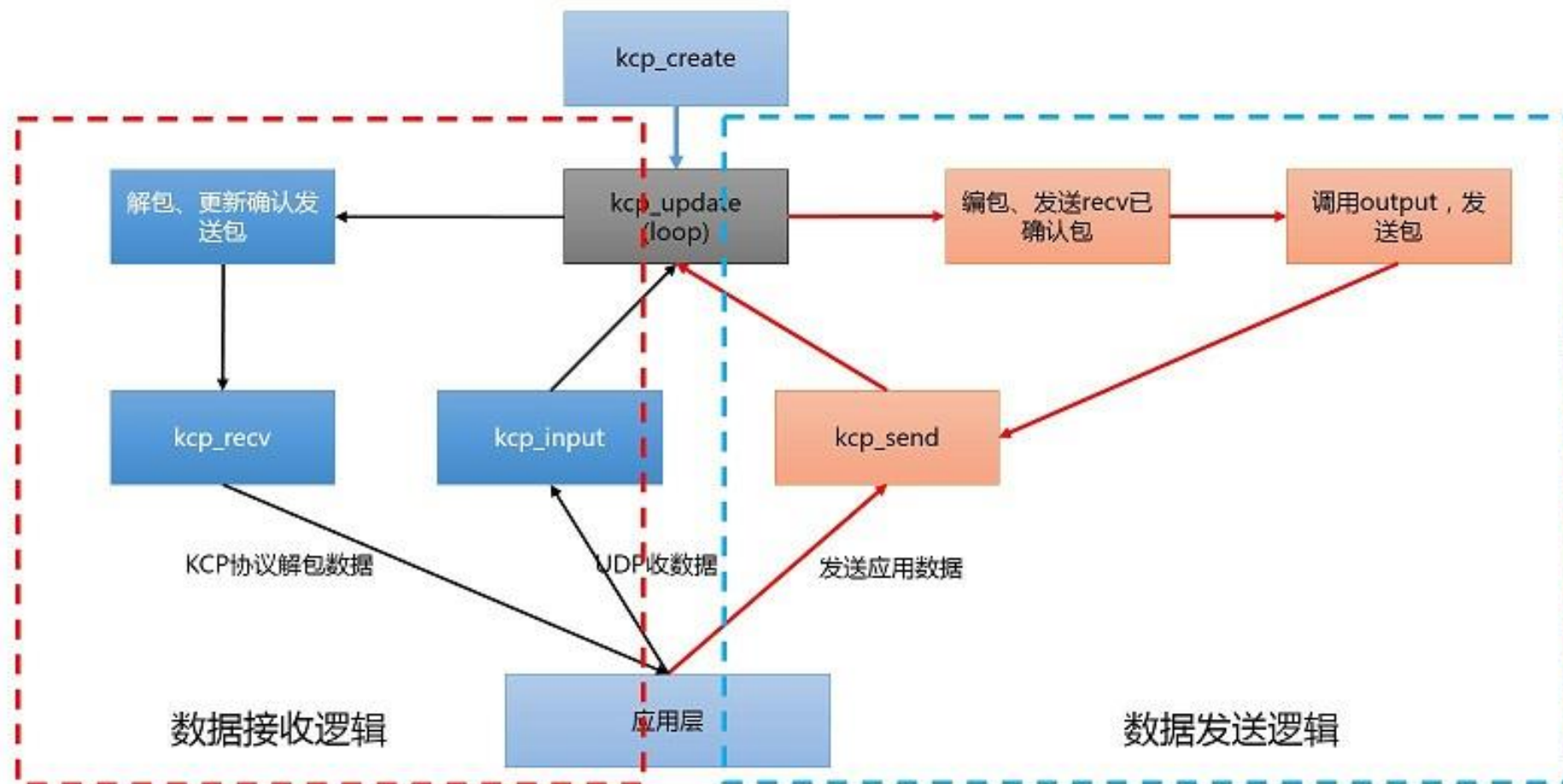
snd_nxt:下一个即将发送的kcp数据包序列号

snd_una:下一个待确认的序列号

4.2 kcp使用方式

1. 创建 KCP对象: `ikcpcb *kcp = ikcp_create(conv, user);`
2. 设置传输回调函数（如UDP的send函数）: `kcp->output = udp_output;`
3. 循环调用 update: `ikcp_update(kcp, millisec);`
4. 输入一个应用层数据包（如UDP收到的数据包）:
`ikcp_input(kcp, received_udp_packet, received_udp_size);`
5. 发送数据: `ikcp_send(kcp1, buffer, 8);`
6. 接收数据: `hr = ikcp_recv(kcp2, buffer, 10);`

4.3 kcp源码流程图



4.4 kcp配置模式

1. 工作模式: `int ikcp_nodelay(ikcpcb *kcp, int nodelay, int interval, int resend, int nc)`
 - ❑ `nodelay`: 是否启用 `nodelay` 模式, 0 不启用; 1 启用。
 - ❑ `interval`: 协议内部工作的 `interval`, 单位毫秒, 比如 10ms 或者 20ms
 - ❑ `resend`: 快速重传模式, 默认 0 关闭, 可以设置 2 (2 次 ACK 跨越将会直接重传)
 - ❑ `nc`: 是否关闭流控, 默认是 0 代表不关闭, 1 代表关闭。

普通模式: `ikcp_nodelay(kcp, 0, 40, 0, 0);`

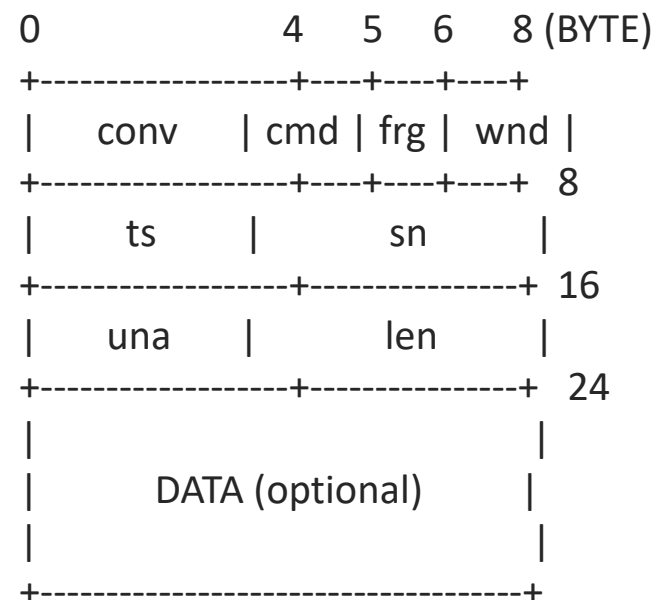
极速模式: `ikcp_nodelay(kcp, 1, 10, 2, 1)`

2. 最大窗口: `int ikcp_wndsize(ikcpcb *kcp, int sndwnd, int rcvwnd);`
该调用将会设置协议的最大发送窗口和最大接收窗口大小, 默认为 32, 单位为包。

3. 最大传输单元: `int ikcp_setmtu(ikcpcb *kcp, int mtu);`
`kcp` 协议并不负责探测 MTU, 默认 `mtu` 是 1400 字节

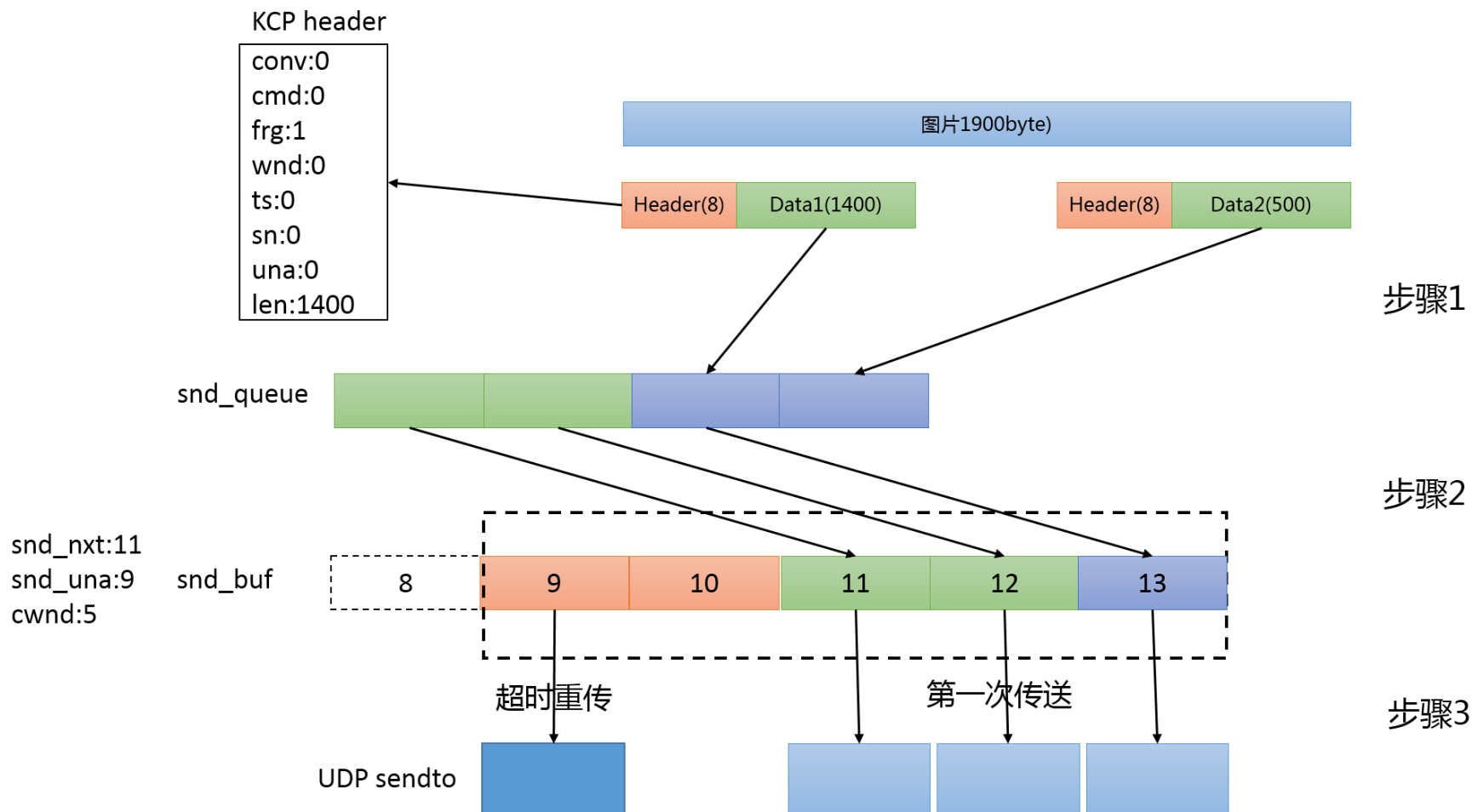
4. 最小 RTO: 不管是 TCP 还是 KCP 计算 RTO 时都有最小 RTO 的限制, 即便计算出来 RTO 为 40ms, 由于默认的 RTO 是 100ms, 协议只有在 100ms 后才能检测到丢包, 快速模式下为 30ms, 可以手动更改该值: `kcp->rx_minrto = 10;`

4.5 kcp协议头

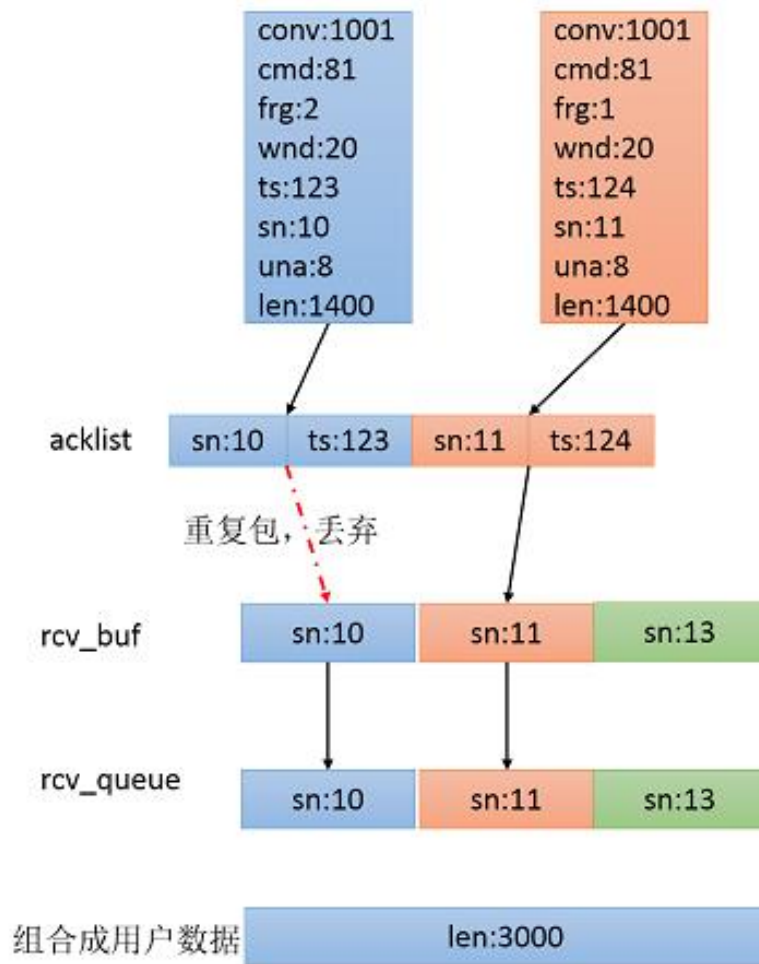
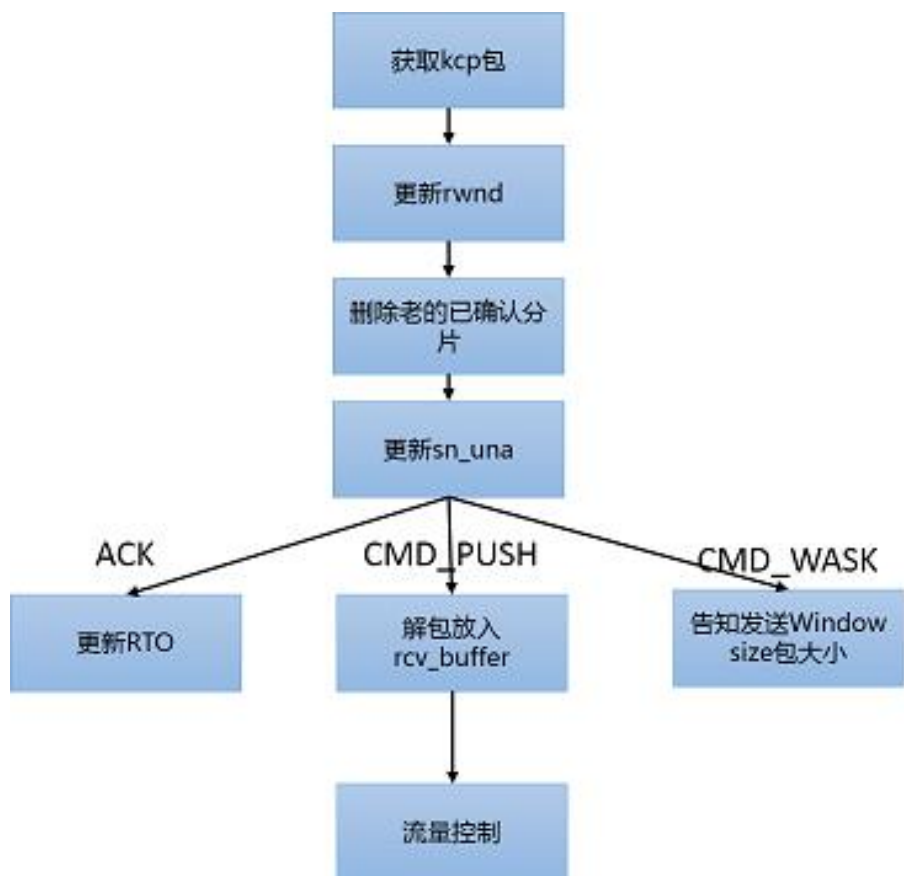


- ❑ conv:连接号。UDP是无连接的，conv用于表示来自于哪个客户端。对连接的一种替代
- ❑ cmd:命令字。如，IKCP_CMD_ACK确认命令，IKCP_CMD_WASK接收窗口大小询问命令，IKCP_CMD_WINS接收窗口大小告知命令，
- ❑ frg:分片，用户数据可能会被分成多个KCP包，发送出去
- ❑ wnd:接收窗口大小，发送方的发送窗口不能超过接收方给出的数值
- ❑ ts:时间序列
- ❑ sn:序列号
- ❑ una:下一个可接收的序列号。其实就是确认号，收到sn=10的包，una为11
- ❑ len: 数据长度
- ❑ data:用户数据

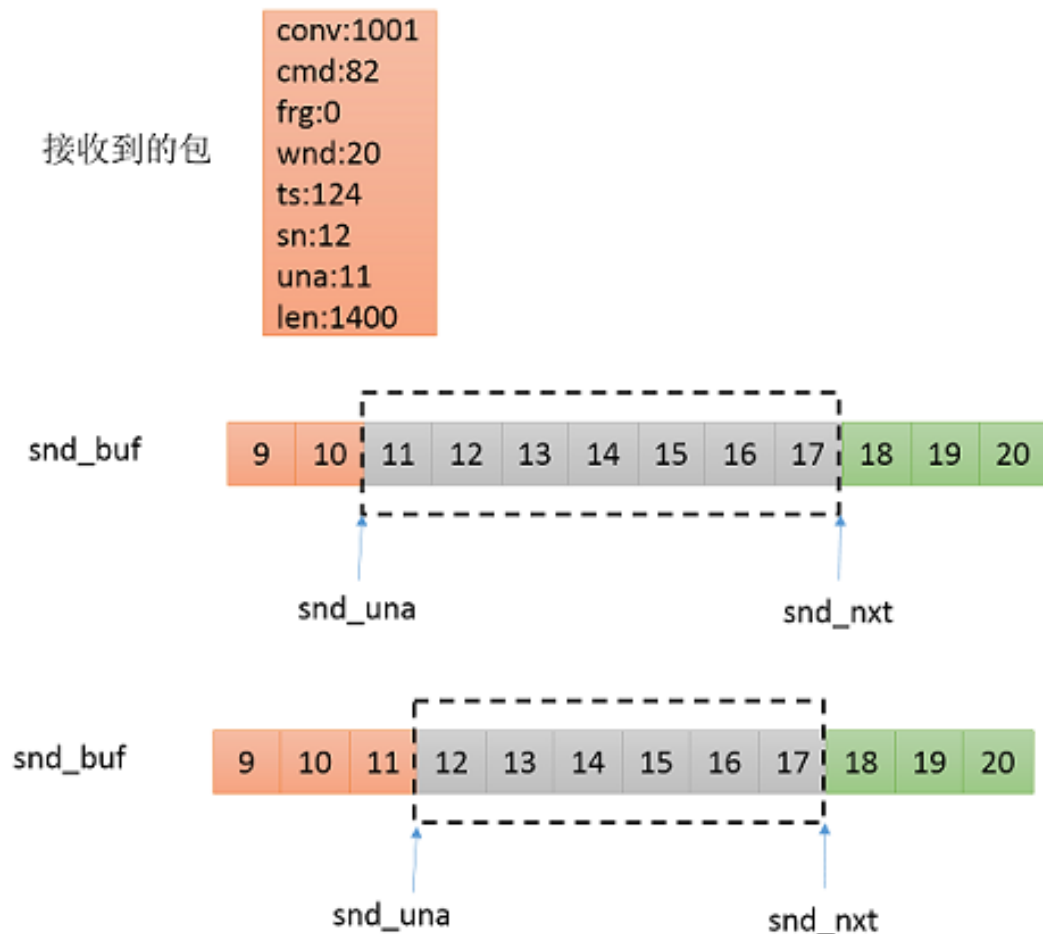
4.6 kcp发送数据过程



4.7 kcp接收数据过程



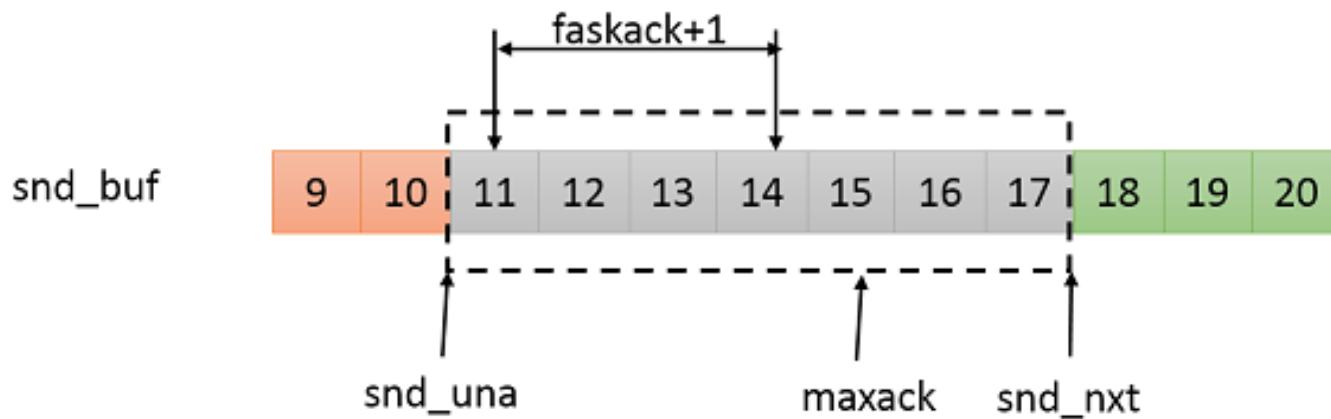
4.8 kcp确认包处理流程



4.9 kcp快速确认

接收到的包

```
conv:1001  
cmd:82  
frg:0  
wnd:20  
ts:124  
sn:15  
una:11  
len:1400
```



4.10 流量控制和拥塞控制

RTO计算（与TCP完全一样）

RTT：一个报文段发送出去，到收到对应确认包的时间差。

SRTT(kcp->rx_srtt)：RTT的一个加权RTT平均值，平滑值。

RTTVAR(kcp->rx_rttval)：RTT的平均偏差，用来衡量RTT的抖动。

■ 5.0 QUIC衍生版本XQUIC（仅作学习参考）

<https://www.yuque.com/docs/share/01d83f75-0fd5-4c9f-976a-0dfcf417e0cc?#>

《[阿里XQUIC：标准QUIC实现自研之路](#)》