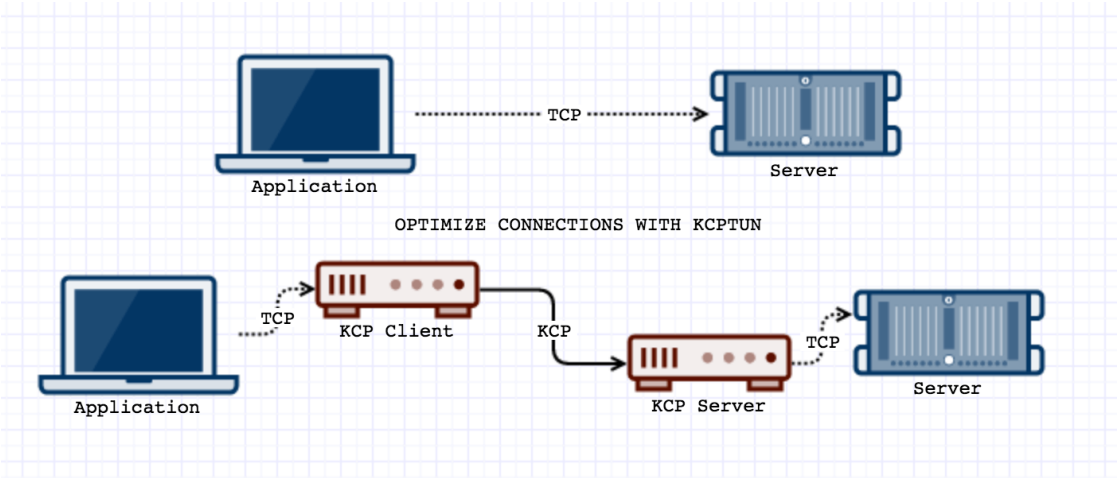
王者荣耀低延迟核心技术 UDP可靠传输实现

笔记本:	零声		
创建时间:	2019/6/21 16:20	**更新时间:**	2019/6/21 19:53
作者:	152fypmm665		
URL:	https://www.cnblogs.com/findumars/p/5794040.html		

KCP协议是什么?

而 KCP 是一个快速可靠协议，能以比 TCP 浪费10%-20%的带宽的代价，换取平均延迟降低 30%-40%，且最大延迟降低三倍的传输效果。
KCP协议的实现：https://github.com/skywind3000/kcp

Kcptun 是一个非常简单和快速的，基于 KCP 协议的 UDP 隧道，它可以将 TCP 流转换为 KCP+UDP 流，见下图：



TCP vs KCP

- **RTO翻倍vs不翻倍:**

TCP的RTO计算

```
初始化:  RTO = 1

第一次计算:
SRTT = R
RTTVAR = R/2
RTO = SRTT + max(G,4*RTTVAR)
(R是本次RTT值，K为4)

以后的计算:
RTTVAL = 0.75*RTTVAL + 0.25*|SRTT - R'|
SRTT = 0.875*SRTT + 0.125*R'
RTO = SRTT + max(G,4*RTTVAR)
```

TCP超时计算是RTOx2，这样连续丢三次包就变成RTOx8了，十分恐怖，而KCP启动快速模式后不x2，只是x1.5（实验证明1.5这个值相对比较好），提高了传输速度。
【PS】有些公司都会去调整这个RTO的计算。

- **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。

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。

KCP协议在UDP的基础上封装了一层协议：

```
0                4      5     6      8 (BYTE)
+-----------------+------+------+------+
|      conv       | cmd  | frg  |  wnd |
+-----------------+------+------+------+   8
|         ts      |          sn         |
+-----------------+---------------------+  16
|       una       |          len        |
+-----------------+---------------------+   24
|                                       |
|            DATA (optional)            |
|                                       |
+---------------------------------------+
```

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

KCP如何使用?

- **创建KCP对象**

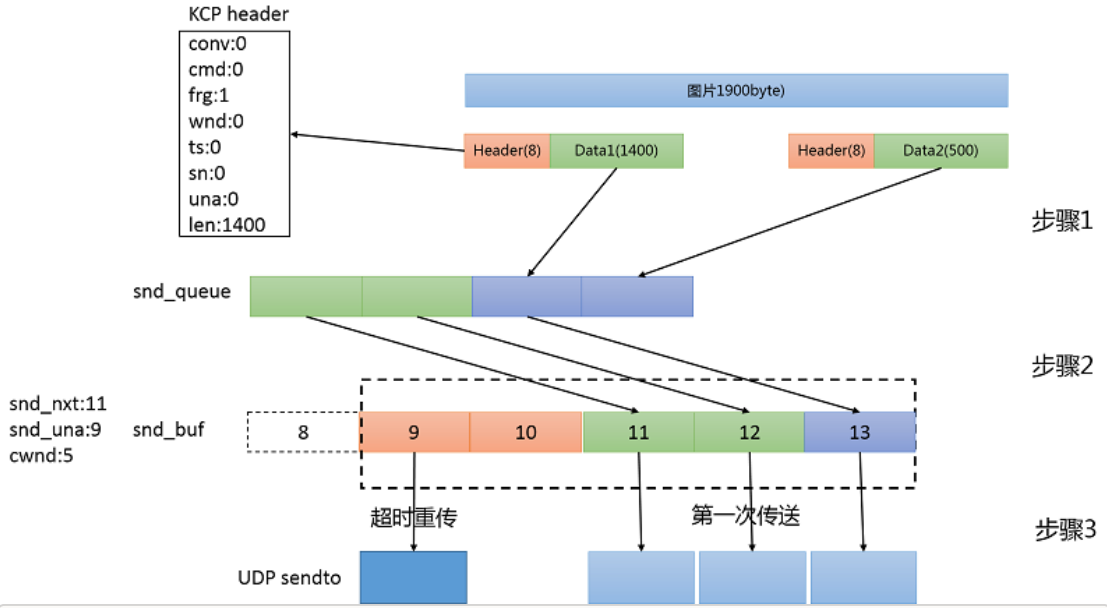
```
// 初始化 kcp对象，conv为一个表示会话编号的整数，和tcp的 conv一样，通信双
// 方需保证 conv相同，相互的数据包才能够被认可，user是一个给回调函数的指针
ikcpcb *kcp = ikcp_create(conv, user);
```

- **数据发送过程**
    - 应用层发送数据

ikcp_send(ikcpcb kcp, const char buffer, int len)
该函数的功能非常简单，把用户发送的数据根据MSS进行分片。比如，用户发送1900字节的数据，MTU为1400byte。因此，该函数会把1900byte的用户数据分成两个包，一个数据大小为1400，头frg设置为1，len设置为1400；第二个包，头frg设置为0，len设置为500。切好KCP包之后，放入到名为snd_queue的待发送队列中。

    - KCP内核层发送数据

KCP会不停的进行update更新最新情况，数据的实际发送在update时进行。发送过程如下图所示：




```
// KCP的下层协议输出函数，KCP需要发送数据时会调用它
// buf/len 表示缓存和长度
// user指针为 kcp对象创建时传入的值，用于区别多个 KCP对象int udp_output(const char
*buf, int len, ikcpcb *kcp, void *user)
{
   ....
}

// 设置回调函数
kcp->output = udp_output;
```

    - 循环调用update获取KCP发送状态
- **数据接收过程**

KCP的接收过程是将UDP收到的数据进行解包，重新组装顺序的、可靠的数据后交付给用户。

    - 应用层接收数据原始的UDP数据包

调用recvfrom或者recv接口

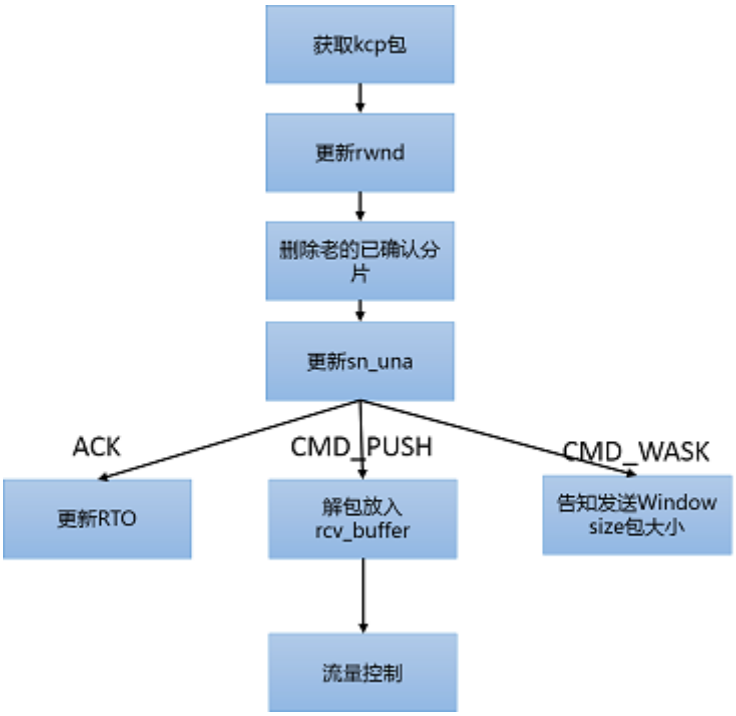
```
ssize_t recv(int sockfd, void *buf, size_t len, int flags);


ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

    - 调用ikcp_input，让KCP内核层去确认数据包

```
ikcp_input(kcp, received_udp_packet, received_udp_size);
```

kcp包对前面的24个字节进行解压，包括conv、 frg、 cmd、 wnd、 ts、 sn、 una、 len。根据una，会删除snd_buf中，所有una之前的kcp数据包，因为这些数据包接收者已经确认。根据wnd更新接收端接收窗口大小。根据不同的命令字进行分别处理。数据接收后，更新流程如下所示：



1、IKCP_CMD_PUSH数据发送命令

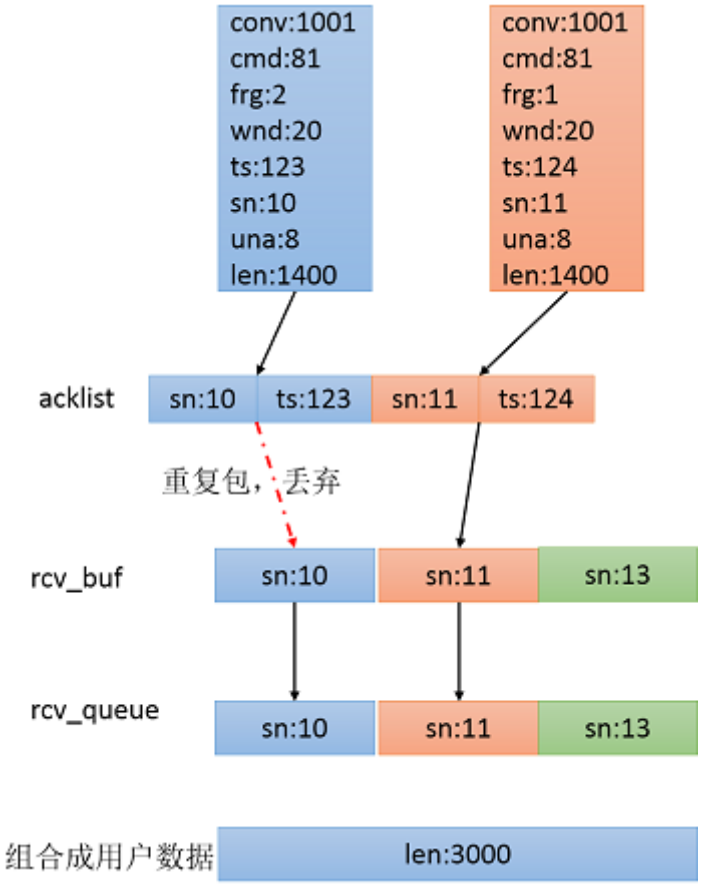
a、KCP会把收到的数据包的sn及ts放置在acklist中，两个相邻的节点为一组，分别存储sn和ts。**update时会读取acklist，并以IKCP_CMD_ACK的命令返回确认包。**如下图中，收到了两个kpc包，acklist中会分别存放10,123,11,124。

b、kcp数据包放置rcv_buf队列。丢弃接收窗口之外的和重复的包。然后将rcv_buf中的包，移至rcv_queue。原来的rcv_buf中已经有sn=10和sn=13的包了，sn=10的kcp包已经在rcv_buf中了，因此新收到的包会直接丢弃掉，sn=11的包放置至rcv_buf中。

c、把rcv_buf中前面连续的数据sn=11，12，13全部移动至rcv_queue，rcv_nxt也变成14。

rcv_queue的数据是连续的，rcv_buf可能是间隔的

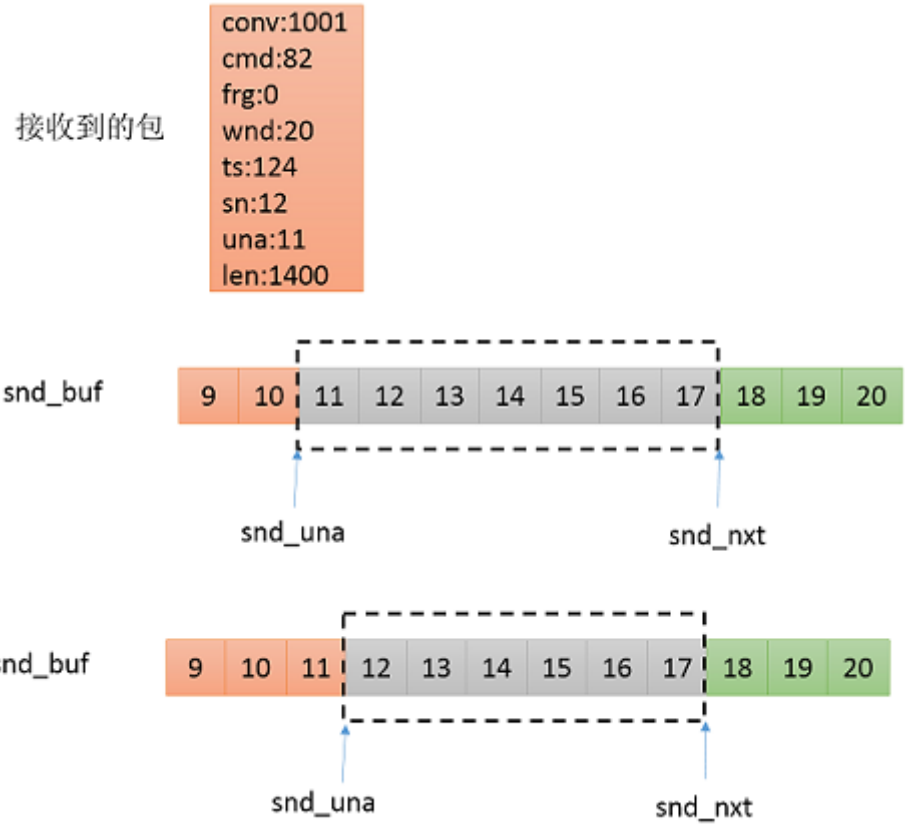
d、kcp_recv函数，用户获取接收到数据（去除kcp头的用户数据）。该函数根据frg，把kcp包数据进行组合返回给用户。



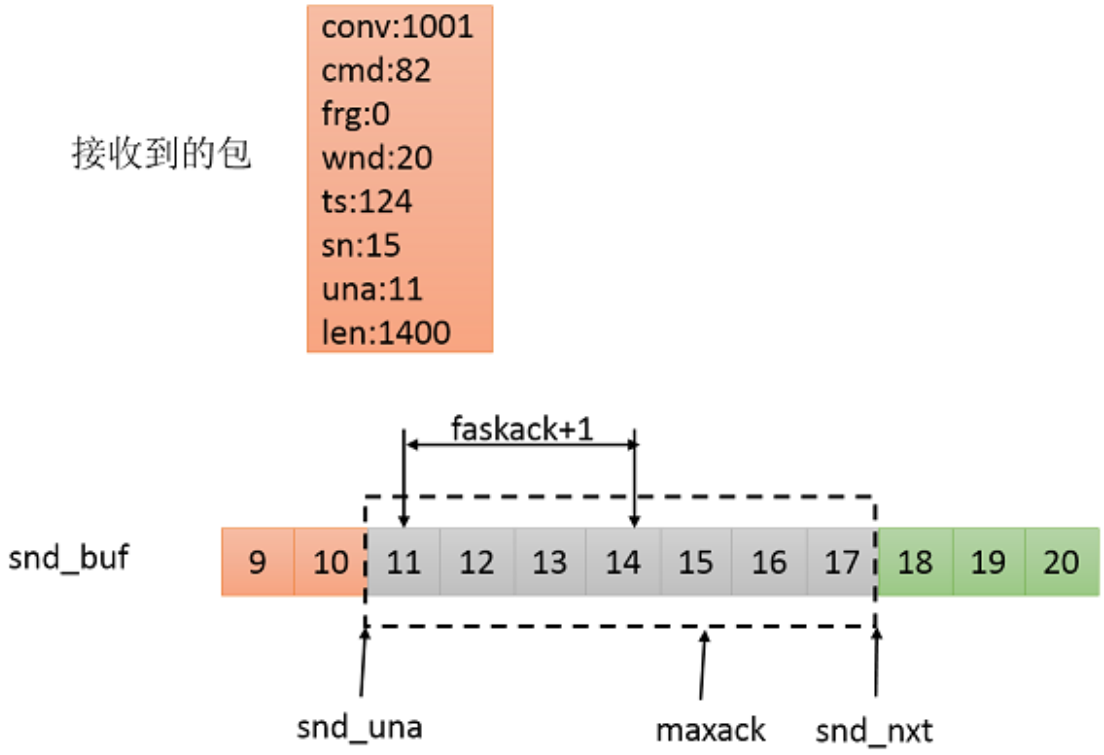
2、IKCP_CMD_ACK数据确认包

两个使命：1、RTO更新，2、确认发送包接收方已接收到。

正常情况：收到的sn为11,una为12。表示sn为11的已经确认，下一个等待接收的为12。发送队列中，待确认的一个包为11，这个时候snd_una向后移动一位，序列号为11的包从发送队列中删除。



异常情况：如下图所示，sn!=11的情况均为异常情况。sn<11表示，收到重复确认的包，如本来以为丢失的包重新又收到了，所以产生重复确认的包；sn>17，收到没发送过的序列号，概率极低，可能是conv没变重启程序导致的；112，则启动快速重传。



确认包发送，接收到的包会全部放在acklist中，以IKCP_CMD_ACK包发送出去

- 调用ikcp_recv去接收数据

```
int ikcp_recv(ikcpcb *kcp, char *buffer, int len)
```