RUDP的实现（KCP over UDP）

背景

在给马儿加上UDP通信的时候，想着借助KCP来保证数据可靠性。本想着都是很成熟的方案，应该能很迅速接入，就没好好去看KCP源码了(实际上现在也没好好看)。网上搜罗了一堆别人的各种Demo后，就直接上调试。结果就是，浪费了好几天时间去做调试，最后发现是抄的Demo不对。所以，记录一下要注意的点。

1、KCP介绍

不介绍了，网上一搜一大堆，自行查阅。KCP项目：https://github.com/skywind3000/kcp。有几篇还不错的文章放在参考链接里了。

2、Golang 服务端

Golang有已经封装好的库可以直接使用，而且，封装的很优雅：

```
package main

import (
    "log"

    "github.com/xtaci/kcp-go/v5"
)

func main() {
    log.Println("Start Server...")
    // 监听端口   不启用AES加密
    if listener, err := kcp.ListenWithOptions("0.0.0.0:12333", nil, 0, 0); err
== nil {
        for {
            s, err := listener.AcceptKCP()
            // 设置KCP重传相关参数   服务端客户端保持一致
            s.SetNoDelay(1, 10, 2, 1)
            // 设置窗口大小
            s.SetWindowSize(128, 128)
            if err != nil {
```

```
                log.Fatal(err)
            } else {
                log.Println("Got a client: ", s.RemoteAddr())
            }
            go handleEcho(s)
        }
    } else {
        log.Fatal(err)
    }
}

// handleEcho send back everything it received
func handleEcho(conn *kcp.UDPSession) {
    buf := make([]byte, 4096)
    randStr := "abcdefghijklmnopqrstuvwxyz"
    for {
        n, err := conn.Read(buf)
        if err != nil {
            log.Println(err)
            return
        } else {
            log.Println("recv: ", string(buf[:n]), n)
        }
        for i := 0; i < n; i++ {
            buf[i] = randStr[i%26]
        }
        n, err = conn.Write(buf[:n])
        if err != nil {
            log.Println(err)
            return
        } else {
            log.Println("send: ", string(buf[:n]), n)
        }
    }
}
```

上面这段代码的作用是，服务端在接收到客户端发来的数据后，会按照交互次数依次回传abcde...，回传长度和发送长度保持一致。

3、C 客户端

马儿是纯C写的，所以，要来个纯C的客户端，这就开始踩坑了，为了节省空间，这里只放一些关键流程：

a、初始化

1、最开始要调用 `ikcpcb* ikcp_create(IUINT32 conv, void *user);` 来创建kcp对象
conv: 理解成会话ID吧。
user: 后续会被用在回调函数中，这里你可以定义一个结构体放进去，方便你在回调函数中完成相应功能，比如发送数据。

2、建立UDP连接， `sockfd = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);`

3、设置回调函数：
`kcp->output = udpOutPut;`
或者调用 `void ikcp_setoutput(ikcpcb *kcp, int (*output)(const char *buf, int len, ikcpcb *kcp, void *user))`
回调函数，主要的实现是发送数据包。

b、发送数据

1、设置KCP参数： `ikcp_nodelay(kcp, 1, 10, 2, 1);` 与服务端保持一致（不一致会怎么样没测试过

2、设置滑动窗口大小： `ikcp_wndsize(kcp, 128, 128);` 与服务端保持一致（不一致会怎么样没测试过

3、发送数据： `int ikcp_send(ikcpcb *kcp, const char *buffer, int len);`
这个时候，发送的数据会被写到kcp结构体的内存中，并不是直接被发送出去。

4、更新kcp结构体， `ikcp_update(kcp, iclock());` 这里需要把当前时钟信息传入，用于更新kcp状态，并且看是否要调用 `ikcp_flush` ，真正的发送数据的地方在 `ikcp_flush` 。大多demo都是在循环调用，收发都在这个大循环里，于是，整的很懵，感觉流程总是不清晰。

5、如上所述，由于并不是每个 `ikcp_update(kcp, iclock());` 都会调用到 `ikcp_flush` 来把buffer中的数据发送出去。所以，建议在每个 `ikcp_send` 后加上 `ikcp_update` 和 `ikcp_flush` ，能有效减少延时感。当然，在这之前，你需要先建立UDP连接。

c、接收数据

1、先调用 `recvfrom` （全文讲的都是KCP over UDP），接收到包含KCP头的完整UDP数据包。

2、调用 `ikcp_input` ，这里会将上面接收到的UDP数据，解析并存到 `kcp` 结构体中。

3、调用 `ikcp_recv` ，如 `out_size = ikcp_recv(send->pkcp, buf, recv_size);` 如果接收到

的数据都能组装好，通过了校验，这里`out_size`就是服务端通过KCP发过来的数据的大小，`buf`中存放接受到的数据。

4、调用`ikcp_update`，接受到数据后，还要调用`update`更新一下kcp状态。

d、踩的坑

总结下来，就是没有细细看代码，跟流程吧，网上找了一些各种while循环的Demo，跑起来看似能跑，实际上数据包是乱的，导致写代码1小时，调试2天，并且总怀疑是自己方案问题，来来回回换了pipe、libev多种思路，结果发现是自己对kcp不熟悉。
即，没有去深入了解：
什么时候需要调用`ikcp_update`？
什么时候需要调用`ikcp_flush`？
什么时候数据包接收成功了？
`recv_from`后的数据，调用`ikcp_input`返回是什么意思？
调用`ikcp_recv`在什么情况下，是拿到了正确的数据？
`ikcp_input`传入的buf需不需要与`ikcp_recv`保持一致，需不需要做清空操作？
...

4、Demo源码

`Talk is cheap, Show me the code.`
https://github.com/purpleroc/KCP_over_UDP

其中：
https://github.com/purpleroc/KCP_over_UDP/blob/main/client.cpp 是while循环处理kcp的实现。
https://github.com/purpleroc/KCP_over_UDP/blob/main/client_libev.cpp 是使用libev来做的实现。

附录：参考链接

详解 KCP 协议的原理和实现：https://luyuhuang.tech/2020/12/09/kcp.html
UDP可靠传输协议KCP的一些理解：https://www.cnblogs.com/wenlong-4613615/p/16908147.html