分享的问题：

1. TCP本来就具有数据包校验功能，那么为什么基于TCP的上层协议还需要自定义数据校验？
2. TCP具有Keepalive机制为什么需要应用层使用心跳包来探测对方是否存活？

为什么要分享这几个问题：

不知道有没有伙伴和我一样有这些问题，比如：

- TCP不是可靠的网络传输协议吗，TCP的超时重传、拥塞控制、CheckSum校验难道还不能保证数据的可靠性？
- TCP具有开启keepalive的选项，为什么应用层去发送心跳包？
- 在TCP网络编程中，客户端服务器建立了连接，但是没有发送数据连接会不会断开？多久断开？为什么断开？

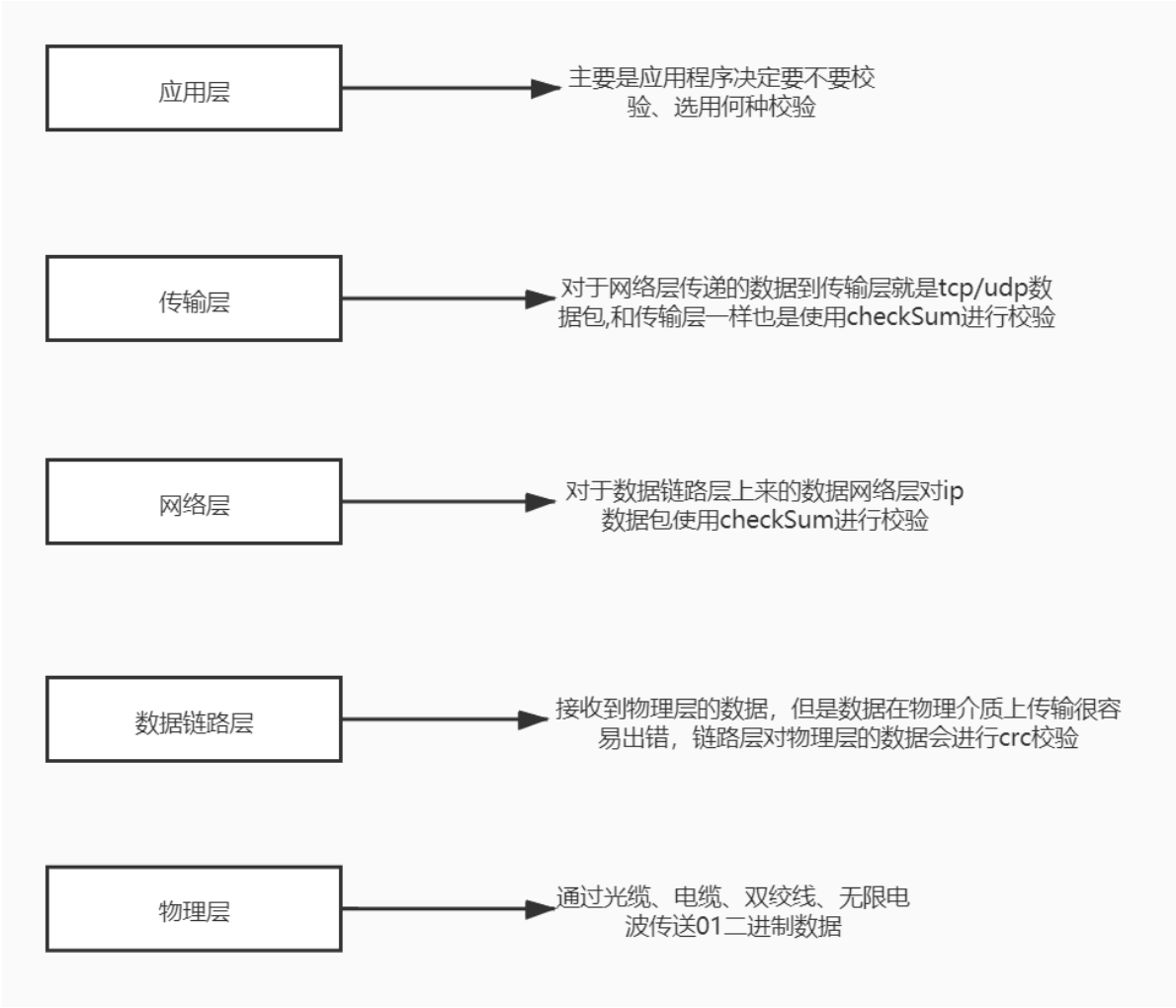
最近在看交控的一些设备驱动代码，终端设备的接入方式基本为网口接入和串口接入使用自定义的通信协议封装数据来传送数据帧，对于网口接入的设备往往会使用goroutine的方式去发送心跳数据包，并且自定义的通信协议也有数据校验的过程

百度百科定义：

> 传输控制协议（TCP，Transmission Control Protocol）是一种面向连接的、可靠的、基于字节流的传输层通信协议

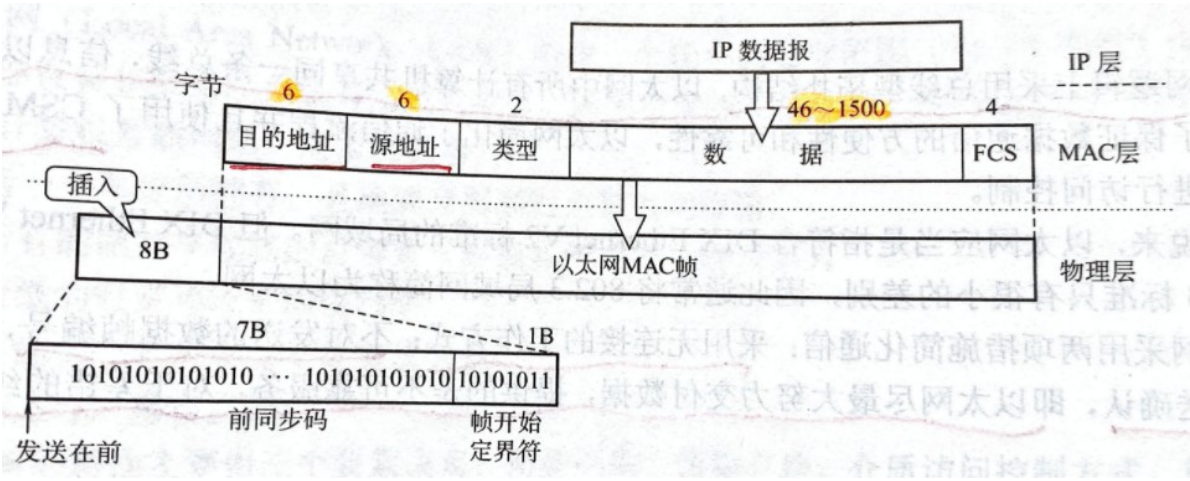
TCP本来就具有数据包校验功能，那么为什么基于TCP的上层协议还需要自定义数据校验

回顾一下网络模型：



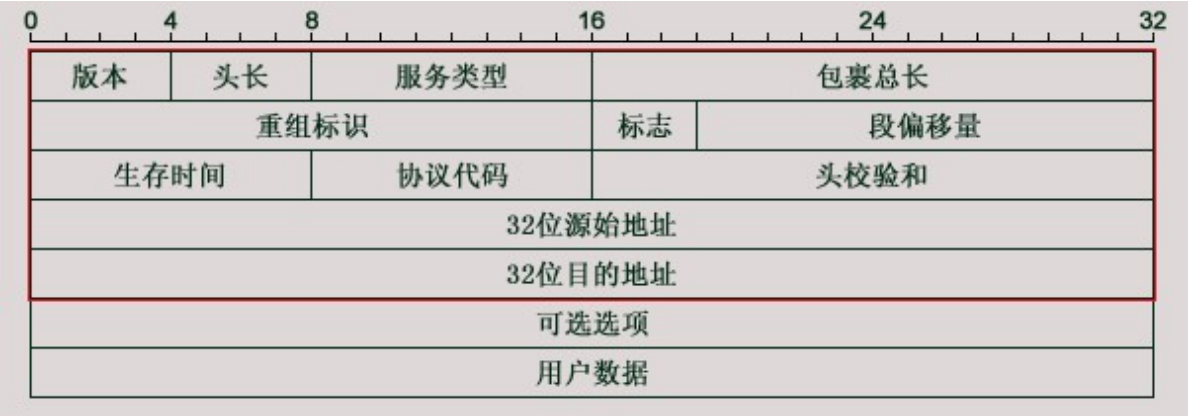
各层数据包的封装就像俄罗斯套娃一样

以太网帧：



以太网帧的校验码FCS采用32位循环冗余码(CRC);不但需要检验MAC帧的数据部分，还要检验目的地址、源地址和类型字段，但是不校验前导码

IP数据包：



IP数据包也包含校验码，但是它只对IP数据报的头部信息计算校验码并不包含数据区域，而且计算校验和的算法为将IP数据包头按16位划分为一个单元，对各个单元采取反码加法运算(对每个单元取反码相加，高位溢出位会加到低位，所有单元计算完后将得到的和填入校验和字段)

TCP数据包：

TCP数据包也包含了校验和，生成校验和的算法和IP数据包的一样，但是不同的是TCP的校验和是把数据区域也加入了计算的校验和的范围内而IP数据包不是

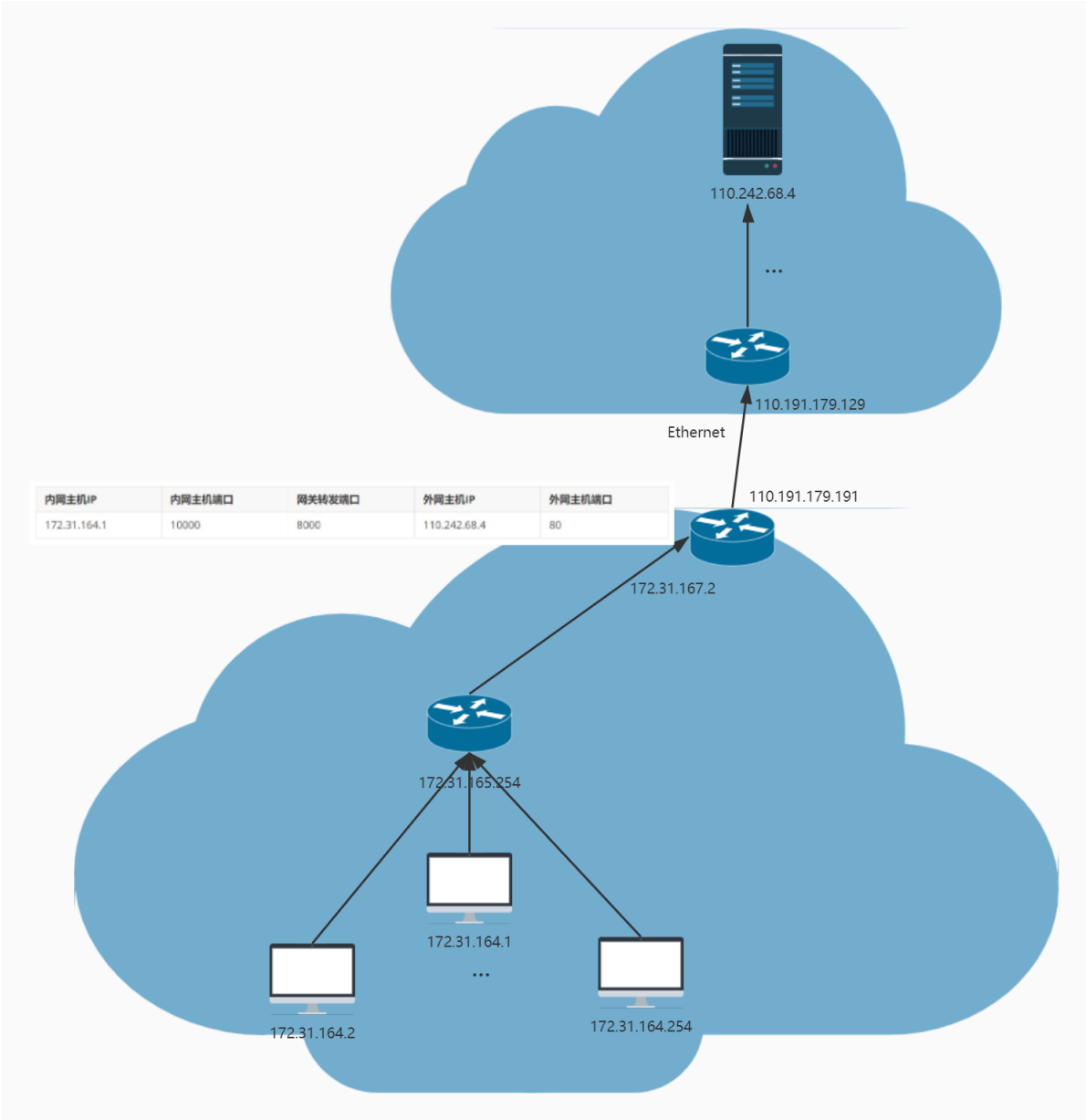
所以：

1. 虽然在Mac帧层使用了Crc32位进行校验但是还是可能会有出错的概率，导致错误的数据包传递给了上层，我们都知道数据在物理层上传输是使用电信号、光信号但是这些信号在传输过程中是很容易出错的，比如电信号，它在物理层上的传输是通过规定阈值来区分01的，比如1伏一下表示0，一伏以上表示1，但是数据在物理上传输是会受到磁场等干扰，所容易原来的0可能变成1,1也可能变成0，这里的核心思想是，距离数据的发源地越近的地方，将错误的数据丢弃，避免错误的数据奔袭万里贻害网络资源
2. 如果一个错误的数据包在数据链路层没有被检查到，那么在上传的IP层、TCP层有没有可能也没有检测到？当然有可能！所以我们为了保证万无一失还可以在应用层使用复杂的检测算法来检测数据的完整性

TCP具有KeepAlive机制为什么需要应用层使用心跳包来探测对方是否存活

不知道有没有同事试过或者想过建立好的连接如果遇到断网、断电等情况会怎么样?

在说明TCP KeepAlive机制之前，聊一聊内网主机一般如何上外网：我们的绝大多数主机，不管是公司电脑还是家庭电脑的ip都是运营商提供的内网IP，内网IP可以和本局域网的其他主机直接通信，但是要和外部网络的主机通信就需要借助网关，一般使用的NAT网络地址转换协议



比如一台主机它的内网ip为**172.31.164.1**端口10000的程序要去访问外网ip**110.242.68.4**端口80的程序，它是没办法直接访问的，于是它把数据包发送给它的网关**172.31.165.254**端口8000，网关会把ip数据报里的源地址修改为自己的ip**172.31.165.254**源端口修改为8000，并生成一张端口转换表：

内网主机IP	内网主机端口	网关转发端口	外网主机IP	外网主机端口
172.31.164.1	10000	8000	110.242.68.4	80

当网关8000端口接收到主机110.242.68.4回复的数据时，会查询端口映射表，把ip数据报里的目的IP修改为内网主机172.31.164.1并把目的端口修改为10000，通过这个过程内网IP就完成了和外网IP的通信

但是我们知道一台设备它的端口最多为65536个(0-65535),当内网中的主机特别多或主机上特别多进程要访问外部ip时，网关上的端口可能就不够用了，所以网关会监控端口的状况，如果网关上的端口很久没有转发数据了，那么就会把该端口释放掉，供其他需要和外部通信的内部主机使用，并且不会进行通知

TCP KeepAlive是用来干嘛的：使用TCP连接的C/S模式的应用中，当TCP连接建立后，如果一方发生宕机、断网、路由器故障等情况时，另一方是没有办法感知到连接失效的，导致它一直维护着这个连接，如果是服务端，那么维护过多的半开连接还会造成系统资源的浪费。所以不管是客户端还是服务端都需要快速感知到对方还是否存活，这就有了TCP层面的KeepAlive机制。TCP KeepAlive机制是指不管是客户端还是服务端都可以在建立连接的时候指定是否开启KeepAlive，开启之后(TCP协议层面默认不开启)如果连接没有发送数据达到一定时间(TCP_KeepAlive_Idle 参数Linux下默认7200s也就是2h)，会向对方发送TCP KeepAlive数据包，对方收到数据包之后必须ACK，如果对方应该特殊情况收不到这个

KeepAlive包那么就没办法回复ACK，这是会定时(TCP_KeepAlive_Interval参数，Linux下默认75s)发送KeepAlive包，当这些KeepAlive包未被应答累计到了一定的数量(TCP_KeepAlive_Probes参数，linux下默认为9)之后，TCP就会返回错误。

心跳包HeartBeat: HeartBeat和TCP的KeepAlive机制类似，都是通过发送数据来检验对方是否存活，不同的是应用层面的HeartBeat不用要求另一方应用层面回复数据，可以通过TCP层面的数据包是否被ACK来检验对方是否存活。

所以已经有KeepAlive机制了为什么还要应用层HeartBeat呢?

1. TCP的KeepAlive是传输层面的，TCP协议层面是规定默认不开启的，而且不同操作系统（windows、linux）去实现TCP协议时参数设置还有差异
2. TCP层面的KeepAlive和应用层的HeartBeat并不冲突，是不同层在进行控制，可以达到更好的效果
3. TCP的KeepAlive机制是传输层协议在发送接收数据包，发送接收的内容不可控制，对于一些要监听设备状态IM应用，HeartBeat就是必须的

参考资料:

https://www.zhihu.com/question/370717865?utm_source=wechat_session

https://my.oschina.net/zhongwcool/blog/4617109

https://www.cnblogs.com/fhefh/archive/2011/10/18/2216885.html

https://www.bilibili.com/read/cv10935452?from=search