**Protocols**

protocols define format, order of msgs sent and received

among network entities, and actions taken on msg transmission, receipt

*network edge:* hosts: clients and servers

*access networks, physical media:* wired, wireless

*network core:* §interconnected routers §network of networks

digital subscriber line (DSL)数字用户线

*frequency division multiplexing （FDM）频分复用*

二进制的运算算术运算二进制的加法：0+0=0，0+1=1 ，1+0=1， 1+1=10(向高位进位)；即7=111

10=1010 3=11

二进制的减法：0-0=0，0-1=1(向高位借位) 1-0=1，1-1=0 (模二加运算或异或运算) ；

二进制的乘法：0 \* 0 = 0　0 \* 1 = 0，1 \* 0 = 0，1 \* 1 = 1 二进制的除法：0÷0 = 0，0÷1 = 0，1÷0 = 0 (无意义)，1÷1 = 1 ；

逻辑运算二进制的或运算：遇1得1 二进制的与运算：遇0得0 二进制的非运算：各位取反。

**TCP UDP 校验算法**

2015年08月04日 16:16:40 阅读数：1042更多

个人分类： [计算机网络](https://blog.csdn.net/u012271722/article/category/5708057)

 经常看计算机网络相关的书时，每次看到关于IP或者是UDP报头校验和时，都是一笑而过，以为相当简单的东西，不就是16bit数据的相加吗！最近在学习Ping命令的源待时，看到里面有关于校验和的算法。一头雾水，后来查找资料，看到校验和是16bit字的二进制反码和。总是觉得很奇怪，为什么会用反码和，而不是直接求和呢？或者是补码和呢？因为在计算机里面数据是以补码的形式存在啊！经过看书查资料，下面总结一些这个校验和算法具体是怎么实现的。

    首先，IP、ICMP、UDP和TCP报文头都有检验和字段，大小都是16bit，算法基本上也是一样的。

    在发送数据时，为了计算数据包的检验和。应该按如下步骤：

    1、把校验和字段设置为0；

    2、把需要校验的数据看成以16位为单位的数子组成，依次进行二进制反码求和；

    3、把得到的结果存入校验和字段中

    在接收数据时，计算数据包的检验和相对简单，按如下步骤：

    1、把首部看成以16位为单位的数字组成，依次进行二进制反码求和，包括校验和字段；

    2、检查计算出的校验和的结果是否为0；

    3、如果等于0，说明被整除，校验和正确。否则，校验和就是错误的，协议栈要抛弃这个数据包。

    虽然说上面四种报文的校验和算法一样，但是在作用范围存在不同：IP校验和只校验20字节的IP报头；而ICMP校验和覆盖整个报文(ICMP报头+ICMP数据)；UDP和TCP校验和不仅覆盖整个报文，而且还有12个字节的IP伪首部，包括源IP地址(4字节)、目的IP地址(4字节)、协议(2字节)、TCP/UDP包长(2字节)。另外UDP、TCP数据报的长度可以为奇数字节，所以在计算校验和时需要在最后增加填充字节0(填充字节只是为了计算校验和，可以不被传送)。

    在UDO传输协议中，校验和是可选的，当校验和字段为0时，表明该UDP报文未使用校验和，接收方就不需要校验和检查了！那如果UDP校验和的计算结果是0时怎么办？书上有一句话：“如果校验和的计算结果为0，则存入的值为全1(65535)，这在二进制反码计算中是等效的”

那么校验和到底怎么计算了？

1、什么是二进制反码求和

    对一个无符号的数，先求其反码，然后从低位到高位，按位相加，有益处则向高位进1(和一般的二进制法则一样),**若最高位有进位，则向最低位进1.**

    首先这里的反反码好像和以前学的有符号反码不一样，这里不分正负数，直接每个为都取反。

    上面加粗的那句话和我们平时的加法法则不一样，最高位有进位，则向最低位进1。确实有些疑惑，为什么要这样呢？自习分析一下，上面的这种操作，使得在发送加法进位溢出时，溢出值并不是10000，而是1111.也即是当相加结果满1111时溢出，这样也可以说明为什么0000和1111都表示0了。

    下面是两种二进制反码求和的运算：

    原码加法运算：3(0011)+5(0101)=8(1000)

                  8(1000)+9(1001)=1(0001)

    反码加法运算：3(1100)+5(1010)=8(0111)

                  8(0111)+9(0110)=2(1101)

    从上面的例子中，当加法未发生溢出时，原码与反码加法运算结果一样；当有溢出时，结果就不一样了，原码是满10000溢出，而反码是满1111溢出，所以相差正好是1.

    另外，关于二进制反码求和运算需要说明的一点是，先取反后相加与先相加后取反，得到的结果是一样的。

**200 OK**

§request succeeded, requested object later in this msg **301 Moved Permanently**

§requested object moved, new location specified later in this msg (Location: )

**400 Bad Request**

§request msg not understood by server **404 Not Found**

§requested document not found on this server **505 HTTP Version Not Supported**

**451 Unavailable for Legal Reasons**

**429 Too Many Requests**

**418 I’m a Teapot**