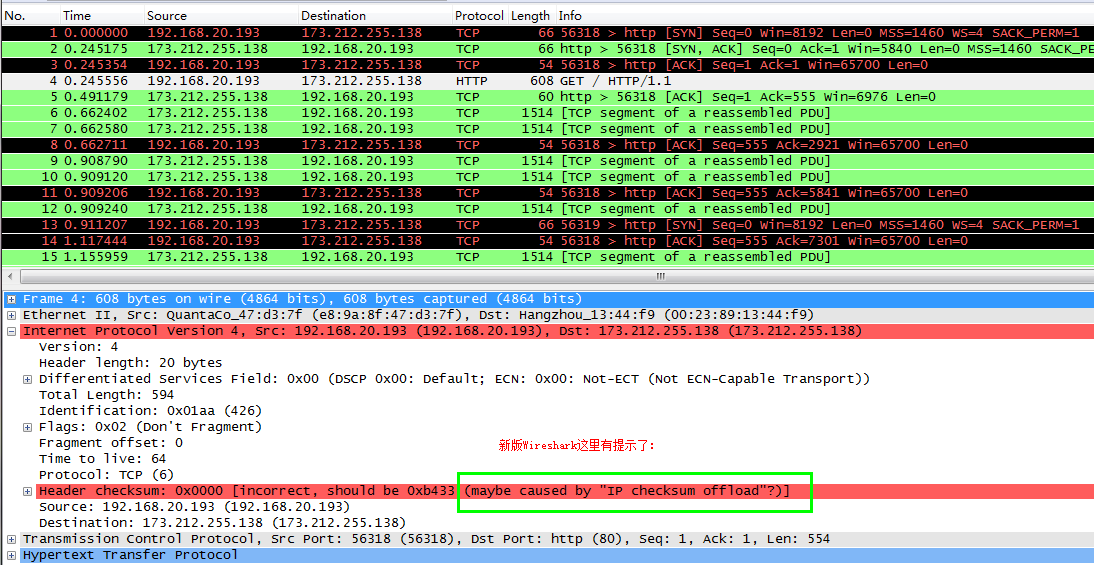
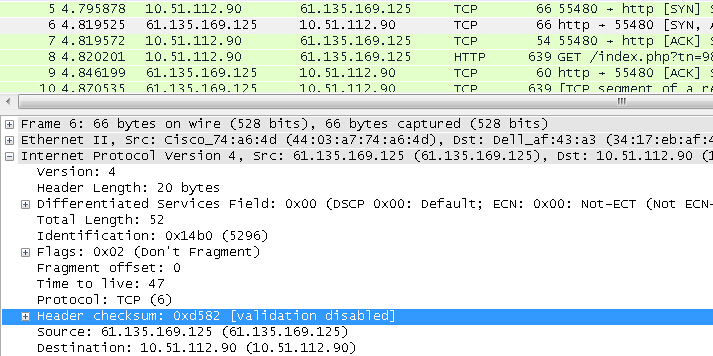
**Wireshark 抓包校验和错误**

# 1.发现问题

在抓包的过程发现很多从本地主机（客户机）发出的 TCP 包有很多都是IP checksum error ，但应用正常，从 server 返回的包也一切正常，很是奇怪。如图，很多黑色的条目，黑色一般就是有问题的包：

[](http://www.sealinger.com/uploads/2011/08/ip_checksum_error1.png)



Wireshark,Ethereal抓包时发现TCP校验和错误

Q:在Windows平台上用Ethereal抓包时发现TCP校验和错误，

但应用层的反应告诉我，这个报文的TCP校验和没问题。

A:网卡配置->高级->Rx Checksum Offload/Tx Checksum Offload，

很可能你的这两处设置是Enable，将之调整成Disable即可，代价是网络性能降低。

一般由操作系统的TCP/IP协议栈完成TCP/UDP/IP校验和的计算工作，

这两处设置成Enable之后，协议栈不再进行校验和的计算，而是由网卡自己完成。

如果在前述位置没有发现Rx Checksum Offload/Tx Checksum Offload项，

有两种可能，一种是网卡本身不支持这种功能，另一种是网卡驱动未提供配置项，后一种情形居多。

事实上该问题与使用何种Sniffer软件无关。

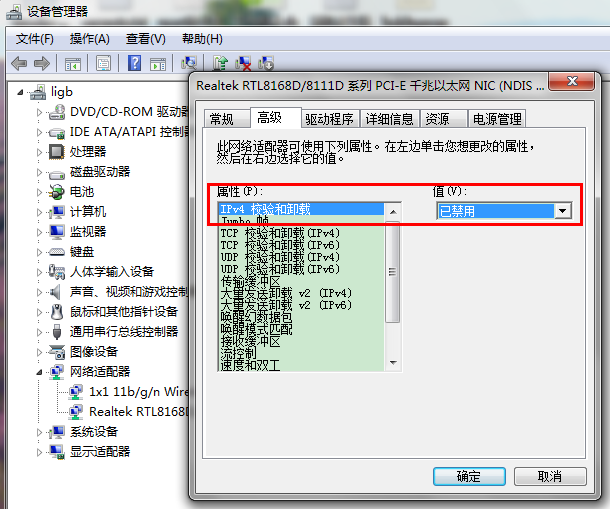
Q：以为是程序的问题，可是接收方却说收到的包一点问题都没有。

A：操作系统偷懒，把计算校验和的工作扔给了网卡，网卡在包发出去才加上校验和，wireshark在OS层抓包的时候，包的校验和还没被添加，是0x0000，于是认为header checksum incorrect。

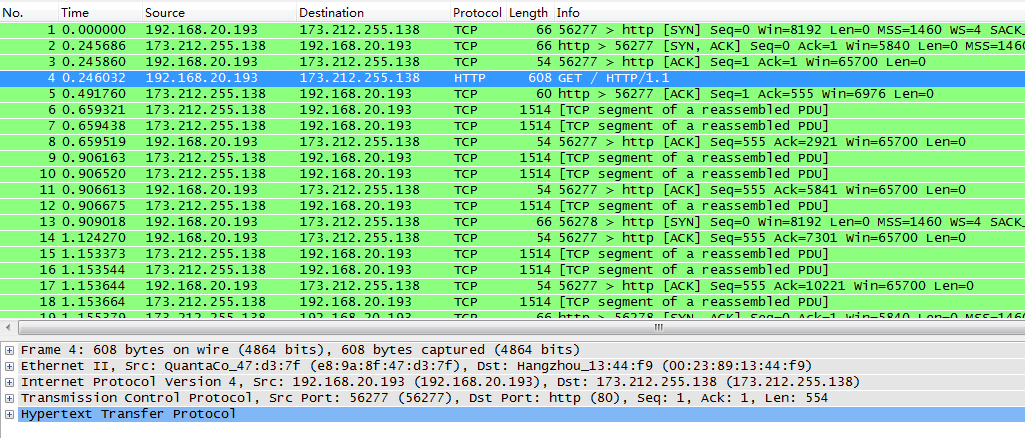
Q:如何让操作系统加入校验和。

**解决方法1：**

把网卡的属性修改一下就可以，禁用 Checksum Offload（原理是让网卡硬件自己计算校验和，而不是交给操作系统的 tcp/ip 协议栈来计算）：

[](http://www.sealinger.com/uploads/2011/08/ip_checksum_error2.png)

禁用 Checksum Offload 后，抓的包显示清爽多了：

[](http://www.sealinger.com/uploads/2011/08/ip_checksum_error3.png)

**解决方法2：**

1. 单击 **开始**、单击 **运行**，键入 **regedit**，然后单击 **确定**。
2. 找到并单击以下注册表子项： HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
3. 在右窗格中确保 DisableTaskOffload 注册表项存在。如果此项不存在，请按照下列步骤以将项添加操作：
   1. 在 **编辑** 菜单上指向 **新建**，然后单击 **DWORD 值**。
   2. 键入 **DisableTaskOffload**，然后按 ENTER 键。
4. 单击 **DisableTaskOffload**。
5. 在 **编辑** 菜单上单击 **修改**。
6. 在 **数值数据** 框中键入 **1**，然后按 ENTER 键。
7. 退出注册表编辑器。
8. 重启后正常了

# 2.IP校验和计算

IP首部校验和的计算方法:

  1.把校验和字段清零。

  2.然后对每16位(2字节)进行二进制反码求和,反码求和的意思是先对每16位求和，再将得到的和转为反码。

  接下来详细描述反码求和的步骤:看下面的代码

  算法：

SHORT checksum(USHORT\* buffer, int size)  
{  
    unsigned long cksum = 0;  
    while(size>1)  
    {  
        cksum += \*buffer++;  
        size -= sizeof(USHORT);  
    }  
    if(size)  
    {  
        cksum += \*(UCHAR\*)buffer;  
    }  
    cksum = (cksum>>16) + (cksum&0xffff);   
    cksum += (cksum>>16);   
    return (USHORT)(~cksum);  
}

  参数buffer是指向16位整数的指针，刚开始指向的是IP首部的起始地址，参数size是IP首部的大小。while循环是将IP首部的内容以16位为单元加在一起，如果没有整除(即size还有余下的不足16位的部分)，则加上余下的部分，此时的cksum就是相加后的结果，这个结果往往超出了16位，因为校验和是16位的，所以要将高16位和计算得到的cksum再加工。

  再加工第一步:cksum = (cksum>>16) + (cksum&0xffff); sum>>16是将高16位移位到低16位,sum&0xffff是取出低16位,相加得到新的cksum。

  再加工第二步:cksum += (cksum>>16); 第一步相加时很可能会产生进位，因此要再次把进位移到低16位进行相加。

  这样就加工好了，接下来就是取反，并强制转换为16位，这样就得到了最终的校验和。

  校验和计算出来了，接下来就是该如何校验:

  接收方进行校验时，也是对每16位进行二进制反码求和。接收方计算校验和时的首部与发送方计算校验和时的首部相比，多了一个发送方计算出来的校验和。因此，如果首部在传输过程中没有发生差错，那么接收方计算的结果应该为全一，因为接收方计算除校验和以外的部分得到值是校验和的反码，再加多出来的校验和当然是全一了。

  最后对上述过程举个例子:

  IP头：

  45 00    00 31

  89 F5    00 00

  6E 06    00 00（校验字段）

  DE B7   45 5D       ->    222.183.69.93

  C0 A8   00 DC     ->    192.168.0.220

  计算：

  4500 + 0031 +89F5 + 0000 + 6e06+ 0000 + DEB7 + 455D + C0A8 + 00DC =3 22C4

  0003 + 22C4 = 22C7

  ~22C7 = DD38      ->即为应填充的校验和

  当接受到IP数据包时，要检查IP头是否正确，则对IP头进行检验，方法同上：

  计算：

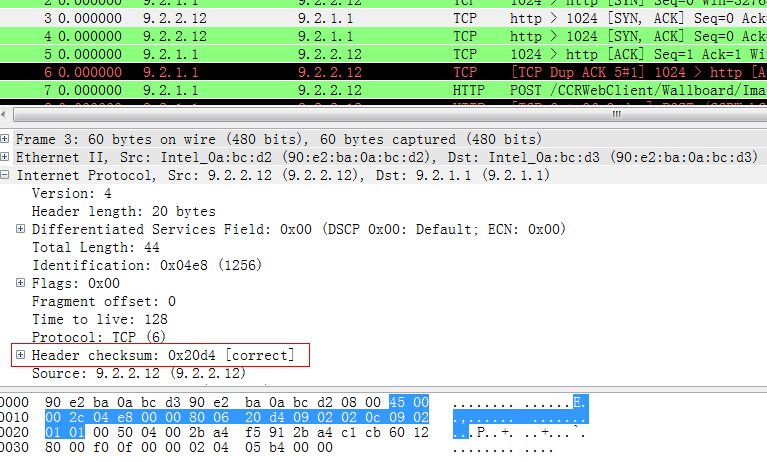
  4500 + 0031 +89F5 + 0000 + 6E06+ DD38 + DEB7 + 455D + C0A8 + 00DC =3 FFFC

  0003 + FFFC = FFFF

  得到的结果是全一，正确。

# 3.个人举例：

根据校验和计算方式，个人验证校验和方式：



图中校验和显示正确，我们就按照之前的计算方式来验证一下；下方蓝色部分是IP头部选择按照接收包的计算方式得到：

4500+002c+04e8+0000+8006+20d4+0902+020c+0902+0101 = 0000 FFFF

0000+FFFF=FFFF在加1后刚好是0000