传统上，报文进入主cpu的linux协议栈，走桥或路由模块转发。但在嵌入式系统上，由于主cpu的主频较低，只有600MHz，无法处理1Gbps这样的大流量。所以通常，都需要其他的硬件模块配合转发。

1. 软硬件结合的SoC

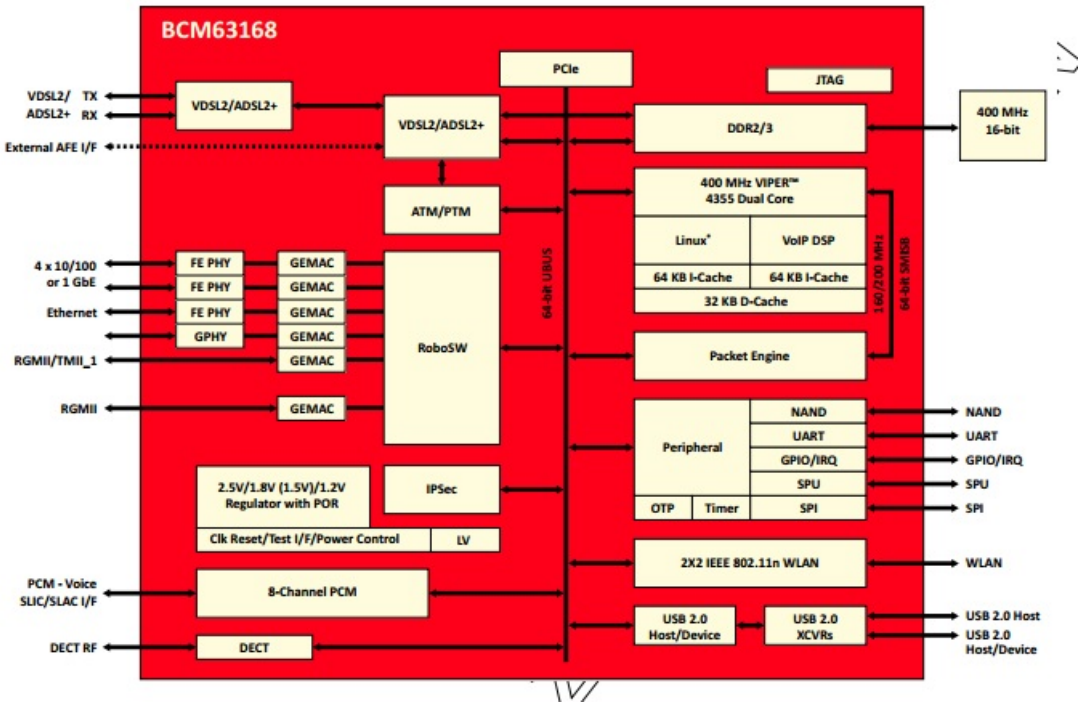
以前的63168芯片使用fcache加速器，数据包先走linux cpu正常转发，并分析报文的特征产生转发规则，确定报文的五元组作为match，确定报文的修改方法和出接口作为result，将规则配置到硬件加速模块，后续的报文在硬加速模块中如果匹配到规则，则直接由硬加速模块修改报文后转发，属于软硬件结合的模型。

成本考虑，硬加速的条目数通常有限制，比如只有256条，也即只支持对256条流加速。家庭网关中，经常性地流的数量达到4K或8K的级别。如果将所有的规则都配置到硬件加速器，会导致硬加速器中的条目频繁变化，我称之为抖动，无法发挥硬件加速的优势。

为解决这个问题，增加一个软件加速器模拟硬件，消除规则配置时的抖动。在linux协议栈分析报文特征产生规则后，先配置到软件加速器，由它往硬件加速器配置。可配置的规则数，取决于主芯片的内存大小，一般都可以配置很多。一个报文进入cpu后，先进软件加速器，查找报文对应的规则，根据规则要求的方式修改报文确定出接口，不进linux协议栈直接转发，提升了性能。当一个规则频繁被命中，当达到一定门限后，就将规则配置到硬件加速器中，由硬件加速器来转发。这种模式硬加速只对高负荷的流生效，优化性能。

条目的删除，无论是硬加速，还是软加速，都依赖于定时器。当一个条目长时间没有命中的时候，定时器会将条目删掉。或者当一条流结束后，主动删除条目。需要注意的是，删除加速器条目，需要检测，协议栈的条目是否也需要同步删除。

很容易想到，这里的软件加速器可以脱离硬件加速器存在。因此可以将这个加速器移植到其他的cpu上，加速软件转发。



2. 硬件转发的SoC ---- 软件定义网络SDN

PON MAC芯片68380，使用Runner芯片处理报文转发。主cpu给runner设置各种规则，报文进入runner后匹配规则，修改报文头部、做QoS调度或限速，选择出接口，可以将报文转发到cpu处理，也可以直接转发到出接口。如果没有规则被匹配到，报文会被丢弃。因此在68380中，规则是核心。由于PON MAC的主要应用在家庭网关中，它的规则条目数存在很多限制，比如上行只有256条L2 ACL规则，下行只有128条L2 ACL规则。同时，它有8192条L3的规则。这都是和家庭网关的定位相匹配的。这类芯片的报文转发流程可以不需要主cpu参与。相比上一种，要灵活很多。

runner中包含很多器件，比如桥、路由、ACL、filter、流量控制等等。具体使用哪些模块，模块之间怎么连接，可以由软件来定义。

3. 交换芯片处理转发

前面提到的63168和68380芯片，都属于SoC，也即将很多硬块集成在一个片子上。如果性能要求更高，比如MDU产品中，通常需要2K以上的L2 ACL规则。这种场景靠SoC很难完成，一般采用一颗功能强大的交换芯片，如56024，配合一颗性能一般的主cpu，如powerPC 8313。交换芯片和主cpu之间采用PCI-e相连，这里将PCI-e作为网线相连。

4. 转发面和控制面分离

在MDU产品中，主控对线卡的管理，有两条路径：1）如果是报文，那么报文转发都通过交换芯片，如果交换芯片的负载较大，有可能导致主控管理不了线卡。2）通过CPLD去设置线卡的寄存器。如果CPLD无法满足需求，可以升级为EPLD。对第一个问题的解决方法，一种方式是提高线卡管理报文的优先级，能够尽快处理。另外一种更加彻底的方法，是将管理报文和数据报文分离，也即分为数据面和控制面。数据面只负责报文转发，控制面负责线卡的管理。这也是9300的思路。

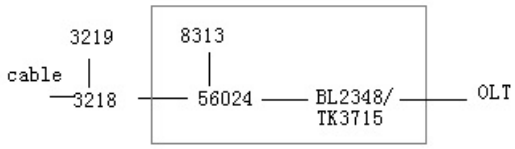
---

1. 项目背景

下图是9026的系统框图。3218是博通Docsis套片，通过一个GMII和交换芯片相连。56024是博通低端交换芯片，我们主要用到它的ACL流转发功能。8313是主控管理芯片，powerPC的。BL2348和TK3715分别是GPON和 EPON的子卡。为降低整机成本，13年9月，选择68380作为9026降成本方案的PON卡芯片，主要的考虑因素：

1）68380同时支持GPON、EPON和GE上行。可以做统一方案的子卡。

2）68380具备一定的ACL功能，上行256条，下行128条。使用68380，可以将56024上行的cos2vlan移到68380，下行的mac2vid，则由68380的桥模块自动转化，不占用规则。希望能够逐步替代56024的功能。

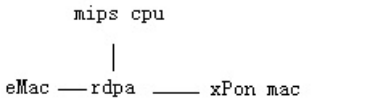


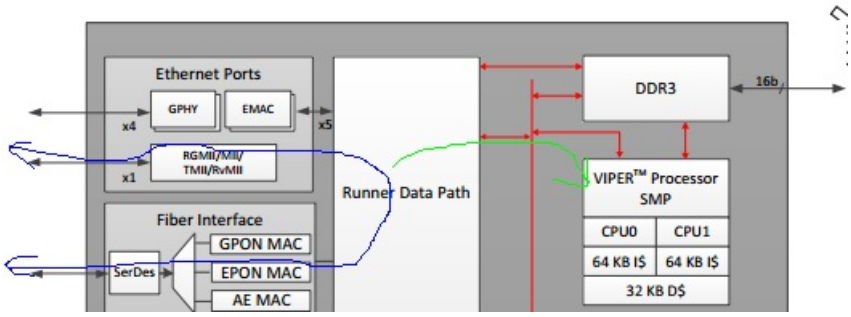
但经过近一年的使用，最初68380的两个优势，实际的结果不太理想。
1）68380在2014年7月才支持GE上行，且烽火验证GE仍然有问题。而且，如果GE上行使用68380，硬件布局上存在困难。最终该方案被否决。GE上行重新出单板。
2）现网需求逐步复杂，单纯地上行cos2vlan，下行mac2vid无法满足需求。实际上还有下行mac2cos的需求。这样下行的规则就捉襟见肘，不够用了。68380起到的作用有限。
3）此外，实际验证中，还发现68380的lan口最多只能够支持128个vlan，超过后无法配置。也即68380最多只能够支持128个CM。
4）56024规则不够用，反推博通，提出了通过一个ge口中转，将2048条规则合理分配，做两级ACL，很好的解决了现网的问题。
通过2）3）4），实际上当初希望用68380做一部分ACL的想法完全落空，且变得毫无必要。不仅如此，选用该芯片实际上还带来一些困难。
1）博通的RGMII功能不完善，直到4.14L04的SDK才支持。而南京规模使用的SDK是4.14L03，没有RGMII功能的。导致不能直接拿南京的版本用，还得另外自己开发版本。导致了软件开发周期拉长。
2）EPON版本，下行ingress_class对广播包失效，和博通一起分析了近三周，仍然无法解决。最终下行用镜像功能将所有的报文全部重定向到68380的RGMII口，规避了问题。
3）还是RGMII功能，直到14年8月，在产线上发现会有上下行同时打流的时候，报文会被反方向报文给修改。要解决这个问题，需要升级SDK到4.14L04a。
通过1）2）3），可见博通的SDK其实仍然不够成熟。对于咱们这种仅仅需要透传功能的ONU来说，是完全没有必要的。
不清楚当初选择这个芯片是否受到供货方面问题影响。

在这个项目中，解决了哪些问题。比如丢包问题，phy调试，板间通讯，phy提包，68380的桥连接以及vlan配置等等。

2. 68380介绍

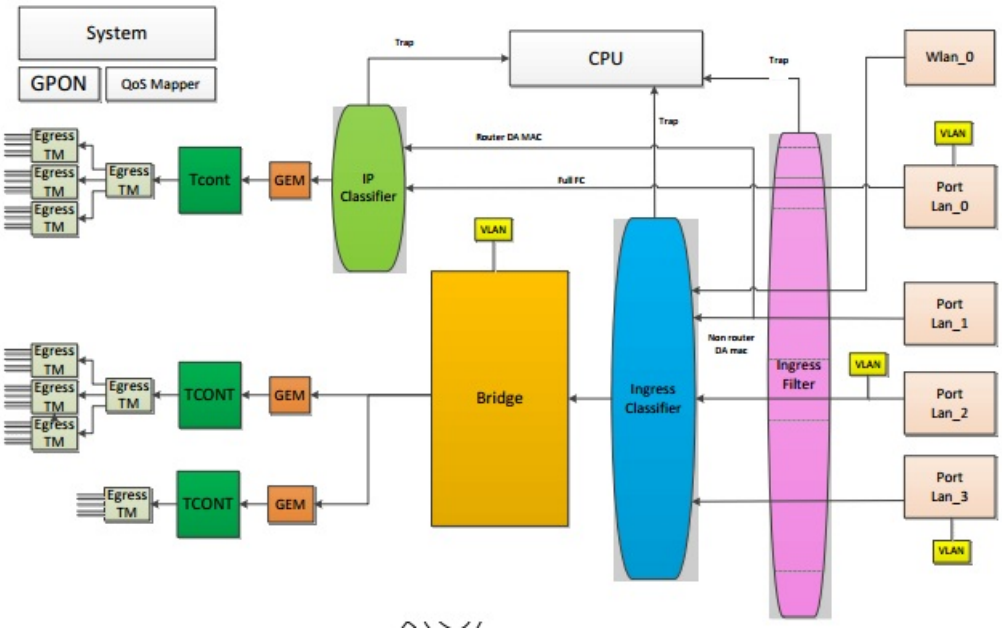
下图是68380框图的简化版，它包含主控cpu、rdpa、xPon和ethernet的MaC。可以发现这个框图和9026的框图非常类似，只是9026使用GMII或PCI-e之类的总线连接各个芯片，而68380是将所有这些器件做到一个芯片中。两者基本的逻辑是一致的。
在这个框图中，rdpa的地位和交换芯片是等同的，事实上很多的功能也和交换芯片是类似的，比如ingress_class功能比对ACL功能。可以将runner当作片上的交换芯片。
68380支持GPON和EPON上行。



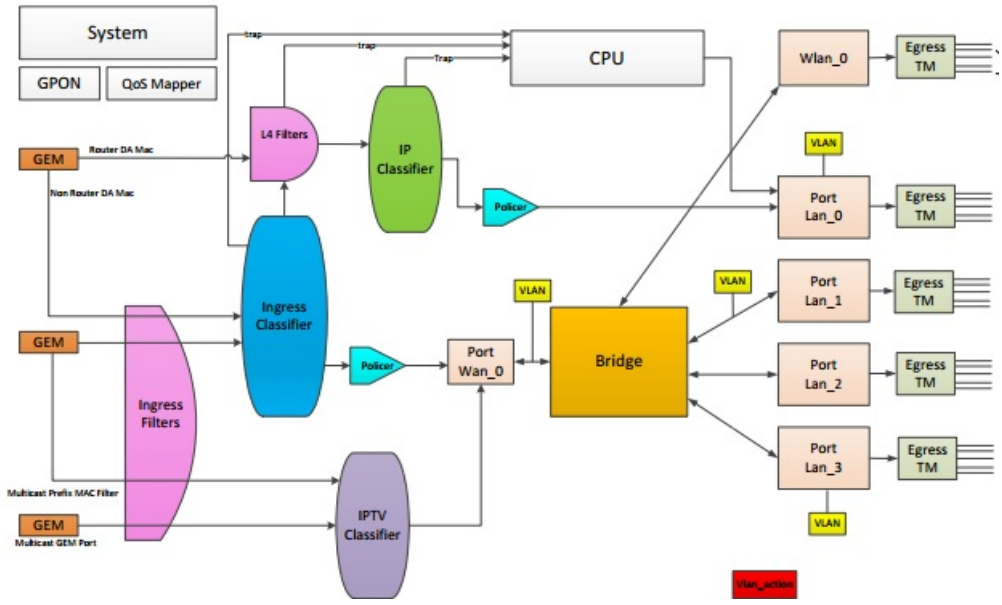


3. 68380 RDPA

下面 两张图是68380的硬件框图。filter是第一个模块，用于将报文提到cpu。也可以在ACL中配置规则提包到cpu。filter提包的匹配条件要简单一些，一般是根据tpid之类的来提包。如果要根据vlan来提包，那么就必须用ACL来做。
9026-2中，要求68380做上行和下行的报文重定向。
68380支持为接口配置默认流，在默认流中，可以指定报文的出接口。可以配置从lan4口默认流走gem0，反方向类似。
9026-2第一次系统测试就是这样用的，但是发现一个问题，wan侧有多个gem口，导致上行方向没有办法用默认流，只能走ingress_class。也即，必须是olt上配置vport的时候，将vlan信息传递到moci模块，omci模块根据这个vlan信息配置上行的ingress_class规则。由于omci模块配置vport的方式比较奇怪，它会首先将所有的vport删除后再重新配置，所以也要求将所有ingress_class规则删除后，再重新配置。因为博通本身的接口不是很友好，所以这里花费了一些功夫才调试通过。
上行报文转发流程

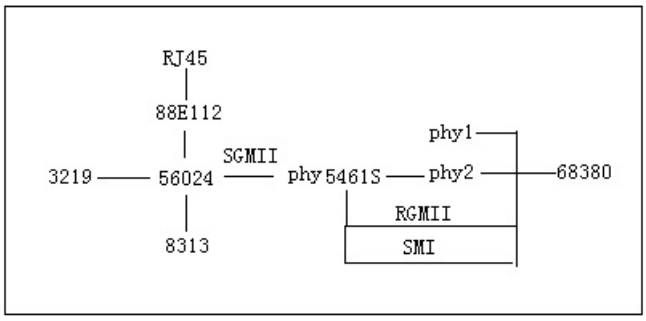


下行报文转发流程

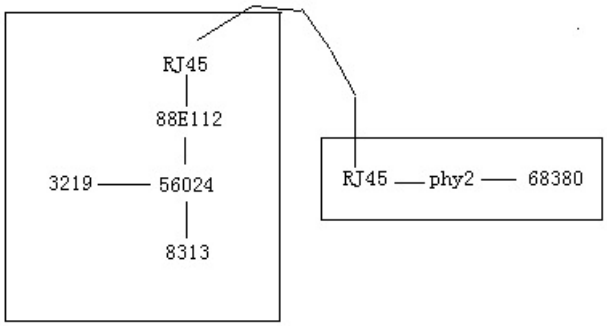


4. BSP

目前的框图。68380通过phy2、RGMII、SMI和phy5461S相连，这个phy负责SGMII和RGMII的转换。
我们在这里调试了SMI接口，通过这个接口设置phy的寄存器，调通业务。



但我想我们可以省掉这个phy。68380的phy2上出一个RJ45口，通过网线直连到56024现有的GE上联口。如下图所示。



初步考虑有有以下优势：
1. 省掉phy成本。省掉一个5461S phy（￥17），为phy2增加一个变压器（￥2）。节省15元。
2. 开发难度小。比如我们现在调试的phy对接、RGMII对接等问题，使用这个方案，都可以省掉，也没有任何风险。开发周期将会缩短很多。
3. 如果，还想在省一点，可以将主控和PON卡分离。子卡这边使用2层PCB板，还可以省点。
4. 如果PCB分离。那么GE上行可以直接去掉子卡，PON上行再加上，这样就非常灵活。
总体而言，和以前的PON子卡相比，这种方案采用网线直连，而以前采用插槽相连，新的方案无论是结构、硬件还是软件上，都要更灵活。

5. SSP

在bsp调试完成后，56024和68380双向发包，双向有计数，调试通过。
在68380中配置了报文重定向，对于wan侧，omci和oam报文都有特殊的通道，保证不会被当作业务流重定向。那么lan侧的报文，应该如何提取到cpu呢？lan侧没有像wan侧那样有多个通道。
前面提到，filter可以根据tpid来提包，且实际验证，发现filter位于ingress_class和port def_flow之前。因此，我们修改主控的代码，保证从主控到68380的板间通讯报文有一个特殊的etherType（IP报文为0x8100，ARP报文0x0806），设置为0x8811。配置filter的规则，提取所有0x8811的报文到cpu。修改内核代码，对所有tpid为0x8811的报文，分析它的三层头部，将tpid还原为0x0800，或0x0806。为什么不直接修改tpid？因为如果修改了tpid，交换芯片会把它当作一个untag报文，为它再加上vlan，而68380这边，对于tpid也有特殊处理。
反方向的，从68380发出的板间通讯报文，因为交换芯片可以根据vlan来提包，作为板间通讯报文，所以不需要修改etherType。

6. 板间通讯

1. 板件通讯的通道目前是通的，check 报文以及构包发过去的 omci 报文都能上主控，但消息格式尤其是消息头，需要详细确认才能完全适配好（要试验）。
2. 内部进程之间通信接口 以 李瑞 提供为准。
3. 如需要新创建应用程序与板间通讯模块通信，需要看下面的适配说明。（不需要另创应用程序，则忽略）。
一、 68380 程序之间通信的适配说明
1 由 smd 来启动应用程序需更改两个文件
修改 SGBC/userspace/public/include/cms_eid.h
添加一个表示应用程序的 EID，例：

```
  EID_ZTE_APP=300,
  EID_ZTE_INTCOMM=301,
```

表示添加了两个应用程序 APP 及 intercomm
修改 SGBC/userspace/public/libs/cms_util/scripts/eid_bcm_base.txt
添加应用程序的启动配置，例：

```
BEGIN
 eid = EID_ZTE_APP
 name = zte_app
 flags = EIF_MDM|EIF_LAUNCH_ON_BOOT|EIF_MESSAGING_CAPABLE
END
BEGIN
 eid = EID_ZTE_INTCOMM
 name = intercomm
 flags = EIF_MDM|EIF_LAUNCH_ON_BOOT|EIF_MESSAGING_CAPABLE
END
```

表示添加了两个应用程序 zte_app 及 intercomm 的配置 (smd在启动这两个应用程序的时候需要)
2 代码的初始化。以APP为例，细节参考 zte_app 下的代码：

```
int main(int argc, char** argv){
   CmsRet cmsReturn;

   cmsReturn = zteapp_cmc_init(); // 应用程序之间要通信必须做的初始化，注意替换 EID
   if (cmsReturn != CMSRET_SUCCESS)
   {
     // Signal MDM error.
     cmsLog_error("zteapp_cmc_init failed, cmsReturn: %d", cmsReturn);
     return cmsReturn;
```

```
    }

    zteapp_thread_init(); // 起了几个线程，用来接受，处理 和 发送消息。
    zteapp_sleep_forever();
    return 0;
}
```

3 编译，修改 Makefile 即可

4 以下入库记录中提供了 zte_app 和 intercomm 程序被 smd 加载，然后两者之间通信的示例：

其中 zte_app 只是纯示范代码，intercomm 是移植的csp平台的板间通讯代码

7. 业务

主要有两种典型业务：

1）mgnt-ip下发 OLT通过omci下发管理ip到68380的omcid，然后由它通过进程间通讯发给板间通讯进程，在通过板间通讯流程发到主控。

2）sn下发 主控OAM配置界面设置sn信息，然后通过omci信息发到68380。

涉及到同步消息和异步消息的区别。

8. 版本升级

版本升级的主要目标，复用现有的9026的升级流程。

制作版本，在博通版本基础上添加MAP平台的版本头 -> 9026主控实用download命令下载版本到主控flash -> reset sub-card 1重启子卡 -> 子卡boot中通过tftp取版本头，和本地版本头比较，检测是否有新的版本 -> 如果有新的版本，通过tftp下载整个版本，并更新当前版本头到flash中。 -> 使用boot的标准升级函数，将版本写到flash中。

解决的问题

1. flash空间不够。解决版本，download的时候，只将版本下载到内存中，将版本头保存在flash中，供show version使用。同时修改tftp服务器的路径，使得能够去内存路径中修改版本。

2. 版本头比较函数不对，导致版本无法下载。判断逻辑有问题。修改后，版本只能从老版本往新版本升级，没有办法回退。如果需要支持这个功能，陈强可安排修改。

9. EPON版本

EPON和GPON版本的不同之处：

1）BSP方面，调整了68380的PON相关的极性。

2）板间通讯不同。olt发送标准oam和扩展oam到68380，68380的cpu将报文通过socket发到主控的oam处理进程，由主控来处理。不像gpon那样有专门的板间通讯模块。

3）合一版本。通过IIC读取光模块中的最大速率参数，gpon和epon不一样，分别为1.25Gbps和1.24Gbps。根据上行方式，为主控和子卡加载不同的模块和启动脚本。

主要故障

李瑞 9026的子卡不支持配置多个vport 必须下行的时候，为每个vport，即gem配置一条ingress_class规则。

常用调试命令

PON子卡的一些调试手段

主控和PON子卡的基本连接:交换芯片56024的GE3<------>PON子卡的lan4口（也就是两边的内置Gephy对接。注意这个口是焊死在板子上的，另外，PON卡68380这边还出了一个调试网口，如上图的6）

a.子卡这边透传的包：
上行包：56024的ge3->lan4->wan0->gem->tcont->olt
下行包：olt->tcont->gem->wan0->lan4->56024的ge3

b.板间通讯的包是从的pon子卡的lan4口过来通过filter提到CPU口的！
c.olt上下来的组播流是通过iptv对象向lan4，以及交换芯片这边送的！

针对丢包查证的话：
上行透传的包，可以通过:
1.lan4，wan0，gem，tcont的计数查看是否丢包;
例如，可以同时输入以下四条命令：

```
bs /b/e port/index=lan4 stat format:line
bs /b/e port/index=wan0 stat format:line
bs /b/e gem/index=1 stat format:line
bs /b/e tcont/index=1 stat format:line
```

这样就可以看到一般通过lan4，wan0，gem，tcont的包的所有计数以及丢包情况，可以看看包丢在哪里了！
如果加上wan0口的镜像功能就更好了：
bs /b/c port/index=wan0 mirror_cfg={rx_dst_port={port/index=lan0},tx_dst_port={port/index=lan0}}
注意，目前只有wan0口有镜像功能，wan0口是软件层面虚拟的一个对象接口！
2.wan0 口镜像报文到lan0口抓包查看
bs /b/c port/index=wan0 mirror_cfg={rx_dst_port={port/index=lan0},tx_dst_port={port/index=lan0}}
3.olt上抓包，或者看看有没有学到相关报文的mac
下行透传的包，参照上行的报文查证丢包在什么地方的方法是一样的，只不过顺序是相反的！

4.上CPU口的报文，可以直接dump到CPU口的报文，查看报文内容！
bs /b/c cpu/index=host rxq_cfg[3]={dump_data=yes} 开
bs /b/c cpu/index=host rxq_cfg[3]={dump_data=no} 关
5.查看丢包原因
bs /d/r pvdc 1 上行丢包原因统计与计数
bs /d/r pvdc 0 下行丢包原因统计与计数
****

常用命令：
1.查看交换芯片到PON子卡LAN4的包过来的情况，查看LAN4包计数
bs /b/e port/index=lan4 stat format:line
Object: port/index=lan4. Object type: port. Owned by: system
==========
stat
rx_valid_pkt : 1265 这里的计数是上行的从交换芯片56024的ge3口过来的报文（DHCP,板间通讯报文等等）
rx_crc_error_pkt : 0
rx_discard_1 : 0
rx_discard_2 : 0
bbh_drop_1 : 0
bbh_drop_2 : 0
bbh_drop_3 : 0
rx_discard_max_length : 0
rx_discard_min_length : 0
tx_valid_pkt : 1025 这里的计数是上行的pon口或者cpu口过来的报文计数
tx_discard : 0
discard_pkt : 0 丢包计数
2.上行lan4的默认配置（除板间通讯的报文外，其他的都直接透传出去）
bs /b/e port/index=lan4
Object: port/index=lan4. Object type: port. Owned by: system
==========
index : lan4
emac_cfg : {emac_param=
{loopback=no,rate=1g,crc=no,fd=yes,pad=no,allow_too_long=no,check_length=no,preamble_length=0,back2back_gap=0,non_back2back_gap=0,
min_interframe_gap=5,rx_flow_control=yes,tx_flow_control=yes},mode=rgmii,enable=yes}
emac_stat : {rx=
{byte=0,packet=0,frame_64=0,frame_128_255=0,frame_256_511=0,frame_512_1023=0,frame_1024_1518=0,frame_1519_mtu=0,multicast_packet=0,broadcast_packet=0,unicast_packet=0,alignment_error=0,frame_length_error=0,code_error=0,carrier_sense_error=0,fcs_error=0,control_frame=0,pause_control_frame=0,unknown_opcode=0,undersize_packet=0,oversize_packet=0,fragments=0,jabber=0,overflow=0},tx=
{byte=5864,packet=38,frame_64=6,frame_128_255=0,frame_256_511=4,frame_512_1023=4,frame_1024_1518=0,frame_1519_mtu=0,fcs_error=0,multicast_packet=18,broadcast_packet=20,unicast_packet=0,excessive_collision=0,late_collision=0,single_collision=0,multiple_collision=0,total_collision=0,pause_control_frame=0,deferral_packet=0,excessive_deferral_packet=0,jabber_frame=0,control_frame=0,oversize_frame=0,undersize_frame=0,fragments_frame=0,error=0,underrun=0}}
cfg : {emac=emac4,sal=no,dal=no,sal_miss_action=forward,dal_miss_action=host,physical_port=port4}
tm_cfg : {policer=null,policer_unknown_da=null,egress_tm={egress_tm/dir=ds,index=4}}
sa_limit : {max_sa=0,num_sa=0}
def_flow :
{qos_method=flow,wan_flow=1,action=forward,policer=null,forw_mode=flow,egress_port=wan0,queue_id=0,vlan_action=null,opbit_remark=no,opbit_val=0,ipbit_remark=no,ipbit_val=0,dscp_remark=no,dscp_val=0,pbit_to_gem_table=null}
stat :
{rx_valid_pkt=0,rx_crc_error_pkt=0,rx_discard_1=0,rx_discard_2=0,bbh_drop_1=0,bbh_drop_2=0,bbh_drop_3=0,rx_discard_max_length=0,rx_discard_min_length=0,tx_valid_pkt=60,tx_discard=0,discard_pkt=0}
flow_control : {rate=0,burst_size=0,fc_thresh=0,src_address=00:00:00:00:00:00}
mirror_cfg : {rx_dst_port=null,tx_dst_port=null}
transparent : no
US links
bridge/index=0

3.下行默认规则配置（下行olt过来的报文直接根据该默认规则透传到lan4口出去到交换芯片56024）

```
#bs /b/e gem
Object: gem/index=2. Object type: gem. Owned by: gpon
==============================
            index : 2
         gem_port : 4095
        flow_type : ethernet
    ds_encryption : no
           enable : yes
           us_cfg : {tcont=null}
           ds_cfg : {discard_prty=low,destination=iptv}
             stat : {rx_packets=0,rx_bytes=0,tx_packets=0,tx_bytes=0,rx_packets_discard=0,tx_packets_discard=0}
Object: gem/index=1. Object type: gem. Owned by: gpon
==============================
            index : 1
         gem_port : 129
        flow_type : ethernet
    ds_encryption : no
           enable : yes
      ds_def_flow :
{qos_method=flow,wan_flow=128,action=forward,policer=null,forw_mode=flow,egress_port=lan4,queue_id=0,vlan_action=null,opbit_remark=no,opbit_val=0,ipbit_remark=no,ipbit_val=0,dscp_remark=no,dscp_val=0,pbit_to_gem_table=null}
           us_cfg : {tcont={tcont/index=1}}
           ds_cfg : {discard_prty=low,destination=eth}
             stat : {rx_packets=3,rx_bytes=448,tx_packets=0,tx_bytes=0,rx_packets_discard=0,tx_packets_discard=0}
Object: gem/index=0. Object type: gem. Owned by: gpon
==============================
            index : 0
         gem_port : 1
        flow_type : omci
    ds_encryption : no
           enable : yes
           us_cfg : {tcont={tcont/index=0}}
           ds_cfg : {discard_prty=high,destination=omci}
             stat : {rx_packets=0,rx_bytes=0,tx_packets=0,tx_bytes=0,rx_packets_discard=0,tx_packets_discard=0}
```

4.dump上cpu口的报文

```
# bs /b/c cpu/index=host rxq_cfg[3]={dump_data=yes} 开
# bs /b/c cpu/index=host rxq_cfg[3]={dump_data=no} 关
```

5.wan0口统计

```
# bs /b/e port/index=wan0 stat format:line
Object: port/index=wan0. Object type: port. Owned by: gpon
==============================
stat
   rx_valid_pkt : 26
   rx_crc_error_pkt : 0
   rx_discard_1 : 0
   rx_discard_2 : 0
   bbh_drop_1 : 0
   bbh_drop_2 : 0
   bbh_drop_3 : 0
   rx_discard_max_length : 0
   rx_discard_min_length : 0
   tx_valid_pkt : 60
   tx_discard : 0
   discard_pkt : 0
```

6.wan0镜像功能（目前68380的SDK只支持wan0的镜像功能）

```
# bs /b/c port/index=wan0 mirror_cfg={rx_dst_port={port/index=lan0},tx_dst_port={port/index=lan0}} 由于68380只出了一个lan0调试口，所以咱们只能镜像到这个lan0口
# bs /b/e port/index=wan0 可以查看配置情况
Object: port/index=wan0. Object type: port. Owned by: gpon
==============================
            index : wan0
              cfg : {emac=none,sal=no,dal=no,sal_miss_action=host,dal_miss_action=host,physical_port=none}
```

tm_cfg : {policer=null,policer_unknown_da=null,egress_tm=null}
sa_limit : {max_sa=0,num_sa=0}
stat :
{rx_valid_pkt=1,rx_crc_error_pkt=0,rx_discard_1=0,rx_discard_2=0,bbh_drop_1=0,bbh_drop_2=0,bbh_drop_3=0,rx_discard_max_length=0,rx_discard_min_length=0,tx_valid_pkt=0,tx_discard=0,discard_pkt=0}
flow_control : {rate=0,burst_size=0,fc_thresh=0,src_address=00:00:00:00:00:00}
mirror_cfg : {rx_dst_port={port/index=lan0},tx_dst_port={port/index=lan0}}
transparent : no