我们一般更愿意使用调试器从堆栈中获取变量的值。但面对复杂的数据结构活控制流程上，和仔细思考并在关键点添加自检代码相比，在堆栈中步进却不是那么有用。在语句上面点击比读取关键点的输出花费更多的时间。相对于单步跟踪到关键代码，决定在哪里添加打印语句花费的时间更少，特别是我们能够确定打印位置时。更重要的是，程序的调试状态不是那么容易保存下来。

最开始接触内核开发时会很不习惯，因为没有调试器可以使用，没有堆栈可以跟踪，所有能够使用的调试手段只有print这一原始工具。但在一段时间的使用后，会越来越发现这一简单工具的不简单之处。

本文总结在实践过程中的一些调试经验，并且不断更新中。

零、要面对，要接受，要解决

发生故障是令人不愉快的事情，但问题既然出现，就一定有它的原因，要面对，要接收，要解决，心平气和地按照调试经验集，去思考流程，去检查代码，去设计打印语句，去分析日志。故障总是需要解决的，也总是能够解决的。

不回避错误，如果问题不在自己的责任范围内，那么就尽快找人协调。

走查代码

一、缩小故障范围

在内核开发中，由于代码庞杂，可能出问题的地方多，所以有时问题难以定位，那么此时需要考虑的是，如何缩小故障出现的范围。

1. 理清代码流程，发现出错代码段。

加速器加速后无法访问网页

首先，该故障在加速器加速后产生，因此故障可能产生于加速器相关的代码。因此检查所有blog相关的代码是否都有移植过来。在这一步中发现，pppoe中有一个clone的步骤，某一种情况下blog_p字段不能被拷贝到新的skb，代码移植不完整。**在正确的代码基础上去解决故障**，移植是否完全？

然后，故障是在PPPoE拨号后访问网页出错，那么桥模式下拨号会不会出错？IpoE下会不会出错？桥模式下访问网页，可以判断故障是不是产生在驱动代码中。IpoE下拨号上网验证，可以判断故障是不是与ppp驱动有关。经验证桥模式下可以拨号，那么问题可能存在于L3的协议栈中，或者说存在与NetFilter中。此时考虑包在L3的转发时的流程，打印相关文件的调试开关。

CMF加速modem即重启

在解决这个故障的时候，由于无法定位故障出现的代码位置，所以修改了驱动中的包转发代码，将协议栈短路。比如说LAN口进入的包在协议栈中需要经过若干路径达到WAN口，我们将这一过程短路，将包在驱动级直接从LAN口送到WAN口，从而屏蔽协议栈的影响。

2. 定位出错代码发生时间

每日构造

3. 根据故障发生条件，寻找参照组

在H168N修改IPv6 WAN连接导致引用计数的故障中，分析对引用计数的dev_hold和dev_put操作，一共有49个之多。逐个排除仍不得其解。最后发现仅在一些特殊情况下才会出错。获取正常时对引用计数操作的日志，通过对比发现线索。

在QoS拥塞控制故障解决过程中，通过打开qos_mgr的调试开关，获得了大量的打印日志。如果只看错误的日志完全没有头绪，但后来找南京方面要来正确的打印日志，两者一对比就会发现问题。比如配置的命令中，设备名称不正确，配置的速率为0。如果没有参照组且不熟悉业务，那么很难发现TC命令中存在的一些问题。

在加速器访问网页的故障中，有两个对照组：1）关闭加速器和打开加速器获得的打印日志对比；2）BCM和CSP打开加速器后打印日志的对比。前者容易获得，但效果却比后者要差一些。

二、对日志的分析

2. 仔细分析日志

在解决QoS故障时，TC命令配置下去后，报有一条错误提示，但一直没有注意到，最后发现问题与TC配置有关。

在日志分析中，有一些很细微的错误或者不一致，不要放过，可能就会发现问题所在。

3. 分析出错位置的内存打印

如果涉及到跟硬件寄存器操作相关的一些故障，由于代码中大多是通过结构体指针的方式来操作，牵涉较多，比较难定位。有时可以使用： dumpmem 和 setmem 这样两个命令单独操作寄存器地址，可以缩小要debug的范围。同时，这个命令可以帮助了解寄存器各个bit的设置和作用。

注意：如果异常是由硬件问题产生的，异常堆栈参考价值不大，因为此时寄存器并不能反映硬件的问题。比如异常重启的故障，是由于CPU的硬件之间协调不当产生，这种情况下，看寄存器效果不大。

三、打印

打印是最基本的调试方法，但有以下经验需要注意。

1. 不滥用

除非是需要打印语句来判断代码走到哪一步出错，否则不要胡乱加上一堆打印，干扰思路。

2. 仔细设计打印语句

在哪一行加入打印有最好的效果？是否需要判断指针是否为NULL？需要打印哪些关键的字段？哪些字段最能够体现流程的状态？这些不仅是调试阶段需要做

的事情，在设计阶段就应该考虑到，并作为调试语句加入到代码中。

3. 打印语句的级别

在拥塞故障解决故障时，发现可以在运行期间控制输出调试的级别，很强大。

4. 时间戳

对于时间敏感的操作，可以加入时间戳计算过程运行时间，如加入jiffies。注意尽量地让输出语句对流程本身没有影响。

5. 打印信息之间要有区分度，要醒目

使用__FUNCTION__和__LINE__宏自动添加行位置和函数名称

打印语句使用前导符，如####，>>>>>>，总之要容易辨认

包与包之间要能够区分开。

在sk_buff**结构体的尾部**添加pkt_num字段，在包进入协议栈时中计数这个字段，并打印包的属性，如ip头部信息，然后就可以根据抓包大致对应到某一个数据包，之后在协议栈中打印时，都加上这个pkt_num的打印，就可以判断该包在协议栈中的路径。

注意：由于CPU中也会有一些skb包到协议栈中，导致skb->pkt_num计数器可能失效，所以在打印包的时候，需要判断pkt_num是否在合理范围内。

```
#ifdef CSP_KERNEL_LOG
#define CSPPRINTK(fmt, args...)  if(skb->pkt_num > 0 && skb->pkt_num < 10000){ \
printk("#### skb%d %s[%d] ", skb->pkt_num, __FUNCTION__, __LINE__); \
printk(fmt, args...);\
}
#else
#define CSPPRINTK(fmt, args...)
#endif
```

对包计数后，在sk_buff释放的时候打印此包释放时的dev,可以看到一个包释放时的位置，也可以有助于缩小范围。能够做到打印信息是可区分的，从而有利于分析故障。

6. 抓取日志

打开串口工具的capture功能。可以分析大量出现的日志，也可以用于beyond compare之类的工具比较。

四、环境的搭建与故障的复现

1. 搭建本地环境，**减少干扰**

在解决加速后无法访问网页的故障中，在本地搭建了调试环境，大大地提高了调试效率。

在加速后无法访问的故障中，最初为了分析故障，在modem上做了mirror。但CSP的mirror功能是在协议栈中做的，不像BCM在驱动中做，所以mirror并不能真实反映包在协议栈中的流程，反而会干扰思路，后将mirror关闭。

2. 复现故障

故障能否稳定复现？发生故障时能否有日志信息记录？有没有更简单的方法来复现故障？

比如在加速后重启的故障中，要复现该bug有两个必要条件，通业务，大流量。12月份时，由于拨号不通，所以采取在9806上端口直接转发的方式搭建L2环境。但后来拨号通了，思路却没有跟进。其实将两个modem配成桥，然后分别拨号，即可以通L2。谋定而后动，考虑清楚要达成什么目标，需要采取哪些措施，及备用措施。比如大流量，可以用冲包工具，也可以使用ftp。

而在香港BT连接数过多的故障中，由于实验室很难复现故障出现的环境，所以修改代码后也没法验证。

3. 抓包分析

五、具备背景知识

加速器重启的故障都是与相关结构体定义不一致有关。而我在做BCM的NET合并的时候却并没有充分考虑到这些，或者说没有意识到sk_buff及buffer内存划分不一致会带来这些麻烦。这也说明了充分理解待解决问题的背景知识的重要性。问题不是孤立的。

拥塞控制故障解决后，TC命令可以正常配置下去，但是仍然不能对包进行拥塞，所有的包都有一样的优先级。后来发现是由于在测试时，发送的包的长度为60字节，太小，拥塞控制基本上没什么效果，所以看不出来。后来修改发送包的长度为512字节，拥塞控制的效果就可以很明显看出来，问题解决。

背景知识很大一部分是对源代码和功能的理解深度，对源代码如果比较清楚的话，象加速后无法访问网页这样的问题，对源代码增加打印跟踪，相似的这样的问题是容易的解决的，所以结论是应该尽可能熟悉源代码。

另外，背景知识体现在对业务的熟练程度，所以平常在测试时，就应该多留一份心思，以求深刻理解业务。

典型故障及解决过程：

1. CMF加速modem即重启

两种情况：

加速即重启：即使拨号modem也会重启

大量冲包时重启：csp的modem在发包量比较少的时候，工作很好，加速器可以工作。但当充包的数量或大小达到一定程度后，modem就会重启。使用Ixchariot冲包，计划冲1000次包，但当充包409次时，modem就重启了。另外，如果减少充包次数，但增加每次充包的file_size(1000000时)，modem重启得会更快。

故障解决：sk_buff中部分宏开关和BCM打开得不一致，导致字段偏移不一致，出错。

在驱动中，有一块缓冲区用于保存所有的sk_buff，如果发包的速率较慢，在下一个包来临之前，上一个包已经发送完成了，那么新的包仍然会发在缓冲区的头部。但如果发包的速率快了，缓冲区会同时保存多个sk_buff，由于加速器对sk_buff及skb_shared_info的访问都是通过字段偏移量来的，所以可能出现"重叠"的情况，出现错误。

2.加速器条目错位

L3能够加速，但是加速条目被放到brlist中。

原因：Netfilter中blog移植不完整，没有对skb的nfct进行计数，导致加速器在判断包的路径时出错，将包判断为走L2加速。

3. QoS的拥塞控制不生效

故障定位：TC配置时设备名称不正确，速率为0不正确

4.加速器加速后网页无法访问

Modem工作在三层，也就是PPPoE/IPoE模式下，Modem内网的PC无法访问外部网页。但是在关闭加速器的情况下，网页访问正常。

broadcom修改了netfilter的一处代码，不显眼，没有用CONFIG_BLOG宏标明，所以不知道此处修改和加速器有关，也就没有将此处修改合入CSP kernel 。通过对包流程的打印等反复分析，定位到了此处。

5. BT下载过多导致机顶盒无法使用

香港版本出现的问题：1）开始BT后，出现马赛克。2）添加TC保证igmp包先处理后，在BT流很多的时候，组播出现马赛克

香港的测试方法：一个V猫，4个LAN口，一个挂机顶盒，其余挂PC。在每个PC上，开启上百个BT同时下载，看机顶盒是否会出现马赛克。

香港方面的期望值：即使开N多BT，也不要影响机顶盒。但是这个时候，正常的上网不通，却是可以接受的。

赵士新的解决办法：使用TC来保证一些包能够被优先接收。从上行下来的vlan包带有cos，根据cos确定包的优先级，高优先级的包使用1024的连线限制，低优先级的使用252的连线限制。普通IP包（包括BT流），可用的连接数为252。这样限制BT的连接数量。