目录

[模块的物理接口类型和软件协议 3](#_Toc19535790)

[Linux上常用的驱动和工具 4](#_Toc19535791)

[Ubuntu上安装USB/PCIE驱动. 5](#_Toc19535792)

[Linux 如何通过串口收发AT 6](#_Toc19535793)

[CMUX 7](#_Toc19535794)

[PPP 8](#_Toc19535795)

[QLog MDM和ASR的抓LOG和DUMP工具 9](#_Toc19535796)

[QLog 参数说明 10](#_Toc19535797)

[QFirehose和QFLash MDM模块的固件升级工具 11](#_Toc19535798)

[QFirehose 参数说明 12](#_Toc19535799)

[QFirehose 升级失败常见原因- 异常断电重启 13](#_Toc19535800)

[QFirehose 升级失败常见原因- MD5效验错误 14](#_Toc19535801)

[QFirehose 升级失败常见原因- 杂项 15](#_Toc19535802)

[ASR和MDM模块的网卡驱动 17](#_Toc19535803)

[ASR的USB网卡使用 18](#_Toc19535804)

[MDM的USB网卡 19](#_Toc19535805)

[MDM的qmi\_wwan\_q驱动说明 20](#_Toc19535806)

[Quectel-CM 参数说明 21](#_Toc19535807)

[QMAP功能说明 22](#_Toc19535808)

[QMI不通的常见原因 – 模块本身导致的 23](#_Toc19535809)

[QMI不通的常见原因 – 上位机导致的 24](#_Toc19535810)

[PCIE驱动 25](#_Toc19535811)

[What is USB device 26](#_Toc19535812)

[What is USB VendorID ProductID Product Name 27](#_Toc19535813)

[What is USB Interface 28](#_Toc19535814)

[What is USB Endpoint 30](#_Toc19535815)

[What is USB Driver 31](#_Toc19535816)

[What is USB Zero length Packet 32](#_Toc19535817)

[Relationship between USB Interface and USB Driver 33](#_Toc19535818)

[How to install Quectel USB Driver on Ubuntu-PC 34](#_Toc19535819)

[What is AT command an QMI 35](#_Toc19535820)

[How to catch USB usbmon log 37](#_Toc19535821)

[Example to Analysis USB usbmon log 38](#_Toc19535822)

[What is QMAP 40](#_Toc19535823)

[How to use QMAP 41](#_Toc19535824)

[What is IP route table 43](#_Toc19535825)

[What is DNS 44](#_Toc19535826)

[TCP/IP 4层模型以及常用协议图 46](#_Toc19535827)

[9x07模块支持的网卡方式 47](#_Toc19535828)

[什么是上网 48](#_Toc19535829)

[路由表的作用 49](#_Toc19535830)

[网关和子网掩码的作用 50](#_Toc19535831)

[ARP的作用 51](#_Toc19535832)

[DNS的作用 52](#_Toc19535833)

[如何获得IP、子网掩码、网关、DNS 53](#_Toc19535834)

[DHCP动态获取 IP、子网掩码、网关、DNS 54](#_Toc19535835)

[静态设置 IP、子网掩码、网关、DNS 55](#_Toc19535836)

[quectel-CM 的作用 56](#_Toc19535837)

[通过usbmon抓取USB总线数据 和 tcpdump抓网络包分析问题 58](#_Toc19535838)

[RAW IP模式 – ecm 和 qmi\_wwan的区别之一 59](#_Toc19535839)

[RAW IP模式 – 必选拉高DTR才能收发QMI和使能网卡模式 62](#_Toc19535840)

[Ubuntu上安装EC20 USB 驱动 63](#_Toc19535841)

[qmi\_wwan/Gobinet网卡驱动的作用 64](#_Toc19535842)

[Linux使用EC20 USB网卡的3种方式 66](#_Toc19535843)

[9x07内部的网桥 68](#_Toc19535844)

[GobiNet/qmi\_wwan支持网桥 69](#_Toc19535845)

[GobiNet/qmi\_wwan 一次URB传送多个IP包 (该方法废弃，使用QMAP) 72](#_Toc19535846)

## 模块的物理接口类型和软件协议

系统组的驱动和工具，不是根据模块的型号来的，而是根据模块的物理接口，以及跑在物理接口上的软件协议来的。

比如:

1. 不管啥模块，只要是用主串口，对系统组来说，就是同一个模块，不予区分，都是用CMUX程序。
2. QMI协议，不管物理接口是USB接口的还是PCIE接口，都是用quectel-CM拨号。

|  |  |  |
| --- | --- | --- |
| 物理串口 | MTK/RDA的2G模块、MDM模块、Intel模块、ASR模块 | * CMUX   PPP |
| USB接口 | MDM模块 | * Usb-serial-option 串口驱动   PPP   * Qmi/mbim/rndis/ecm 网卡驱动 |
| Intel模块 | * Usb-ACM 串口驱动   PPP |
| ASR模块 | * Usb-serial-option 串口驱动   PPP   * rndis/ecm 网卡驱动 |
| PCIE接口 | SDX24，SDX55 | Quectel\_Linux\_PCIE\_MHI\_Driver   * 串口驱动 * Qmi/mbim网卡驱动 |

## Linux上常用的驱动和工具

打包常用的，定期更新给FAE、SW、ST.

./Doc/kba-20190602\_guide\_to\_use\_qmap.pdf

./Doc/KBA\_EC200&UC200\_Linux&Android\_USB\_driver\_移植说明.pdf

./Doc/Quectel\_Android&Linux\_CMUX\_Driver\_User\_Guide\_V1.1.pdf

./Doc/Quectel\_Android\_RIL\_Driver\_User\_Guide\_V1.8.pdf

./Doc/Quectel\_LTE&5G\_QFirehose\_Linux&Android\_User\_Guide\_V1.1.pdf

./Doc/Quectel\_QLog\_Linux&Android\_User\_Guide\_V1.1.pdf

./Doc/Quectel\_QwinLog\_User\_Guide\_Tara\_20190117.pdf

./Doc/Quectel\_LTE&5G\_Linux\_USB\_Driver\_User\_Guide\_V2.0.pdf

./Doc/To-FAE-Android拨号上网问题的排查\_20190425.docx

./Doc/[toFAE]用usbmon抓取模块的usb总线数据并分析AT\_QMI\_IP包\_Carl\_V1.1\_20190110.docx

./Driver/qmi\_wwan\_q.zip

./Driver/ Quectel\_Android\_RIL\_Driver\_V2.8.8\_beta.tgz

./Driver/Quectel\_Linux\_CMUX\_Driver\_V2.0.1.tgz

./Driver/ Quectel\_Linux\_PCIE\_MHI\_Driver\_V1.2.4.zip

./Driver/Quectel\_WCDMA&LTE\_Linux&Android\_GobiNet\_Driver\_ V1.6.1.zip

./Driver/usb-serial-option.zip

Quectel\_Linux\_USB\_Serial\_Option\_Driver\_V1.0 for ubuntu

cdc\_mbim\_x55\_link\_state.zip (ONLY for SDX55)

./Examples/qfota.zip

./Examples/rild-only.zip

./Tool/ LTE\_QWinLog\_V1.5.zip

./Tool/Quectel\_Linux\_PPP\_Script\_V1.0.0.tgz

./Tool/Quectel\_LTE&5G\_QFirehose\_Linux&Android\_V1.2.3.zip

./Tool/Quectel\_QConnectManager\_Linux\_V1.5.5.zip

./Tool/Quectel\_QLog\_Linux&Android\_V1.3.3.zip

## Ubuntu上安装USB/PCIE驱动.

下面的驱动，我们直接提供修改好的源码，只需要make install安装。然后重启Ubuntu即可生效

* Usb-serail-option 驱动 用于MDM和ASR模块的串口
* Qmi\_wwan MDM的网卡驱动
* GobiNet MDM的网卡驱动
* PCIE MHI SDX24和SDX55的PCIE 串口和网卡驱动

如果想增加或者删除某个模块的驱动，直接在源码里增加或者删除对应模块的VID和PID即可，

然后重新make, make install安装,同时需要重启模块才能生效

drivers\usb\serial\option.c

{ USB\_DEVICE(0x2C7C, 0x0125) }, /\* Quectel EC25 \*/

qmi\_wwan\_q.c

{ QMI\_FIXED\_INTF(0x2C7C, 0x0125, 4) }, /\* Quectel EC25 \*/

GobiUSBNet.c

GOBI\_FIXED\_INTF( 0x2c7c, 0x0125 ), // Quectel EC25

下列驱动，都是Linux内核自带，不需要安装，直接使用

* USB ECM
* USB RNDIS
* USB MBIM
* USB ACM intel模块的USB转串口驱动

Ubuntu上测试模块，必须卸载modemmanager和qcserial

$ sudo apt remove modemmanager 影响quectel-CM

$ sudo rm /lib/modules/`uname -r`/kernel/drivers/usb/serial/qcserial.ko 影响QFLASH升级

## Linux 如何通过串口收发AT

* minicom
* busybox microcom
* 不推荐使用cat /dev/ttyUSB2 & echo –en “ati\r” > /dev/ttyUSB2

串口需要先按照串口编程要求，先配置下，才能读写，否则可能会出现读写异常

* Android 可以使用echo 方式发AT

但是只能使用查看radio log的方式来看AT的返回结果

logcat –b radio –v time | grep ATC &

直接cat /dev/ttyUSB2 会导致RIL认为AT没有返回，从而导致RIL运行异常。

* Liunx C 代码收发AT
* 基于AT的应用-fota参考程序

## CMUX

* 在一个物理串口的基础上，虚拟出多个虚拟串口，使得可以同时AT/PPP/GNSS等.
* 所有模块都共用这份CMUX驱动，CMUX是GSM标准协议，所以不需要为某个模块单独修改。
* Android RIL集成了CMUX，所以在Android上不需要再单独使用CMUX
* ./gsm0710muxd\_bp -s /dev/ttyUSB0 -c 1 -b 460800

-s 指定物理串口名字

-c 1 使能硬件流控, 默认关闭流控

-b 指定串口波特率.比如115200/230400/460800/921600/3000000，默认115200

* 帧长固定127，因为帧长短不影响PPP速度，且有的2G模块只支持短帧
* 默认创建3路虚拟串口 /dev/chn/1 /dev/chn/2 /dev/chn/3
* 不管有没有开启硬件流控，MDM模块的RTS必须拉低(MDM9x07 BUG)
* 物理串口流控使能时。

RTS为输出信号，用于指示本设备准备好，可接收数据，低电平有效，低电平说明本设备可以接收数据。

CTS为输入信号，用于判断是否可以向对方发送数据，低电平有效，低电平说明本设备可以向对方发送数据

但是MDM和ASR模块的RTS/CTS标反了， 所以是RTS接MCU的RTS，CTS接MCU的CTS。

* MDM模块最大波特率支持3M，测试3M，必须去掉EVB上1.8转3.3V的电平转换IC(支持不了这么高的速度)
* EVB上电平转换IC存在的情况下，最高可以测到460800
* 如果EVB和AP是1.8V串口直连测试，电平转换IC也必须去掉
* 虚拟串口是平等的，即1可用于AT，也可用于PPP，没有特别限制

一般1用于AT，2用于PPP，3用于GNSS

* 设置URC从虚拟串口/dev/chn/1上报

AT+QURCCFG="URCPORT","UART1"

AT+QCFG="CMUX/URCPORT",1

## PPP

* PPP包含pppd和chat 2个应用程序、PPP内核驱动、以及Quectel提供的脚本。

Quectel提供的PPP拨号脚本

/etc/ppp/peers/quectel-ppp

/etc/ppp/peers/quectel-chat-connect

/etc/ppp/peers/quectel-chat-disconnect

不能在Windows上编辑这些脚本，否则pppd和chat不识别这些脚本

Quectel\_Linux\_PPP\_Script\_V1.0.0.tgz中readme专门强调了这点。

* pppd、chat，PPP内核驱动都是开源的，客户自己移植，不需要Quectel提供。
* PPP拨号也是生成一个网卡ppp0，用户通过网卡ppp0上网，这点和其他的网卡驱动一样。
* 模块支持网卡驱动的情况下，不推荐使用PPP拨号，使用复杂且网速上不去
* Modem口，MDM是/dev/ttyUSB3, ASR是/dev/ttyUSB2, CMUX推荐/dev/chn/2，Intel是/dev/ttyACM0.
* /etc/ppp/peers/quectel-ppp 除了定义modem口，还会定义串口波特率，默认115200，这个串口波特率和实际的PPP上网速度没有关系，比如USB PPP上网，照样可以上10+Mbits。
* /etc/ppp/peers/quectel-chat-connect 里写了很多AT, 除了ATD\*99#，其他都不是必须的，只是为了方便查看模块的注网状态而已。

OK ATE0

OK ATI;+CSUB;+CSQ;+CPIN?;+COPS?;+CGREG?;&D2

OK AT+CGDCONT=1,"IP","3gnet",,0,0

## QLog MDM和ASR的抓LOG和DUMP工具

* 支持以下功能，且程序会自动识别 : ASR or MDM? LOG or DUMP?
  1. MDM抓LOG和抓DUMP
  2. ASR抓LOG
* 可配合QWINLOG、QXDM实现远程抓LOG，取代ql-tty2tcp
* 不支持远程dump.
* Android上提供编译好的可执行程序.
* 工具包里自带一些常用的MDM filter文件,MDM filter 由模块组提供
* 程序里内置default.cfg
* 有些MDM模块需要 at+qcfg=\"dbgctl\",0，这些模块默认是不吐LOG的(防止对手抓LOG，对比分析)
* 要想抓DUMP,必须先配置模块成DUMP模式。客户调试阶段建议设成DUMP模式

at+qcfg=\"aprstlevel\",0

at+qcfg=\"modemrstlevel\",0

设置成功后，当模块发生dump后，模块端口只有DM口。

当模块未dump时，可以使用测试指令查看效果：at+qtest="dump",1。

正式量产后，设置成重启模式，如下

at+qcfg=\"aprstlevel\"

at+qcfg=\"modemrstlevel\",0

* LOG最少都有几十MB，所以几百KB的log不用给研发分析了。
* DUMP的大小是模块内存的大小，128MB/256MB/512MB

## QLog 参数说明

* $ ./QLog –h 会显示参数说明
* -p 指定DM口，比如/dev/ttyUSB0. 只有一个模块的情况下，建议不使用改参数，因为QLog会自动查找DM口。
* -s 指定log存放目录，不指定的话，就存在当前目录下。 ‘-s 9000’的话，QLog就自动工作成tty2tcp模式， 可以使用'QPST/QWinLog/CATStudio'等工具，连接客户板子的IP和端口9000来实现远程抓LOG

QXDM LOG量很大，一般每秒几MB，所以对网络要求比较高，出现如下打印，表示QXDM LOG无法及时发送给PC，工具会自动停止抓LOG。比如公司的wifi网络就不行.

[036.375]Fail to poll fd = 5, errno : 0 (Success)

[036.375]qlog\_poll\_write fd=5, size=4095, timeout=200, wc=3866

[036.375]savelog fail 3866/4095, break

* -f 指定cfg文件，ASR模块不需要cfg文件。‘-f dump’的会发命令给模块，让模块进DUMP，类似于at+qtest="dump",1
* -n 指定抓取log文件数量上限，最大512，超过数量之后，会自动删除最老的log文件，默认是0，表示没有数量上限
* -m 指定单个log文件的大小，单位MB，范围是2~512, 默认128

## QFirehose和QFLash MDM模块的固件升级工具

* Sahara: 该协议用于在上位机和模块的内存之间传输数据。

抓DUMP就是利用此协议从模块把模块的内存数据全部读出来。

模块升级的第一步，也是用该协议，把mbn文件传送到模块的内存。

* Firehose: 高通新升级协议，用于把Image写到模块的nand上。该协议简单和速度快。

该模式下，模块的VID和PID是05c6/9008,要求不能加载任何USB驱动，而Linux里默认驱动是qcserial.c。

* Stream: 高通老升级协议, 特点是速度特别慢.
* Fastboot: Google定义的升级协议，用于Android手机，高通移植到MDM上。
* Fastboot和Stream会被逐步废弃，优先使用Firehose协议升级，优先使用QFirehose工具升级。
* QFlash 支持Firehose/stream/Fastboot 三种升级协议，通过
* 参数指定。 3为firehose
* 判断固件是否支持Firehose升级，看固件包里有没有update/firehose目录即可。
* Android上提供编译好的可执行程序.
* QFirehose支持远程升级，客户板子空间不够，放不下模块固件。
* QFirehose升级必须在Ubuntu真机，不支持虚拟机.能用Qfirehose升级不要用QFlash.
* QFirehose升级成功与否的判断

通过最后一行log，success or fail

[072.302] Upgrade module **successfully**.

或者百分比之100%

[071.295] Upgrade progress: **100**

## QFirehose 参数说明

* $ ./QFirehose –h 会显示参数说明
* -f FW\_DIR 解压之后的固件目录名

“-f FW\_DIR/update/image\_name” 则升级单个文件，替代fastboot

* -p 指定DM口，比如/dev/ttyUSB0, 不指定的话，QFirehose会自动查找DM口，建议不指定。
* -s 通过/sys path会指定需要升级的模块，类似/sys/bus/usb/devices/xx 形式。

车载V2X应用，一般会用AG15+E20/AG35 2个模块，或者视频直播行业会挂多个模块以提高上传速率。

通过-s参数，可以实现同时升级多个模块。

* -e 全擦NAND，类似与QFIL的全擦功能，5G模块目前用的比较多.
* -n 将会跳过MD5检查。
* -l [dir\_name] 同步log到文件,将会创建log文件qirehose\_timestamp.log.
* -z 0/1 是否发送USB零包，默认是1，发送USB零包。 XHCI控制器在Linux 内核4.2及其以下版本有个bug，不能发送零包。

如果客户用的USB控制器是xHCI(USB3.0)，且内核版本<=4.2, 则可能没有修复这个bug。

1. 优先建议客户修复这个bug，而不是用-z 0的方式来升级，因为这个bug的存在也会导致网卡上网有问题
2. 目前就X20及其之后模块，支持 -z 0 的方式升级
3. 源码qfirehose.c里有注释说明别人是如何修复这个bug，可以让客户提供给他的芯片厂商

* 如何远程升级

1. 前提: 客户MCU和Host通过网线相连，且可以互相ping通, 且网络好。比如公司的wifi网络就比较慢，

特别是隔了网段的，就没法升级。

1. 客户的MCU上，即模块的USB挂载的地方，运行 –p 9008
2. Host PC 上，即固件包放的地方，运行 –p 客户板子IP地址:9008

## QFirehose 升级失败常见原因- 异常断电重启

LOG如下，正在升级过程中出现,如下的write失败。

通过dmesg log也可以进一步确认，可以看到usb断开。

车载行业里常见，车载里面，模块的USB一般接到车机,跑Android或者Linux系统，模块的串口/pwrkey/vbat/reset等接到单片机。 单片机会通过串口判断模块的状态，根据状态做异常流处理。

升级过程中，模块的串口肯定是不回任何AT，导致单片机认为模块异常，从而重启模块。

[008.499] Upgrade progress: 15

**[008.705] qusb\_noblock\_write write=-1, errno: 110 (Connection timed out)**

[008.705] qusb\_noblock\_write cur=0, min\_size=8192

[008.705] fh\_send\_rawmode\_image send fail reads=8192, writes=0

[008.705] fh\_send\_rawmode\_image send fail filesend=5857280, filesize=40501248

[008.705] send finished

[008.705] firehose\_protocol.c fh\_send\_rawmode\_image 554 fail

[008.705] fh\_send\_rawmode\_image fail

[008.705] firehose\_protocol.c firehose\_main 684 fail

[008.705] Upgrade module fail.

root@cqh6:~# dmesg -c

[329903.290116] CPU Budget:update CPU 0 cpufreq max to 1488000 min to 480000

[329903.290151] CPU Budget hotplug: cluster0 min:0 max:4

[329906.290140] CPU Budget hotplug: cluster0 min:0 max:4

**[329912.564864] usb 2-1.2: USB disconnect, device number 9**

[329913.570357] usb 2-1.2: new high-speed USB device number 10 using sunxi-ehci

[329913.680978] usb 2-1.2: New USB device found, idVendor=2c7c, idProduct=0125

## QFirehose 升级失败常见原因- MD5效验错误

LOG如下，正在升级过程中出现,如下的MDM5检查失败.

模块固件包有个md5.txt文件，保存了每个image名字和对应的md5效验码。Md5.txt保存了发版本的时候，计算出来的md5. 固件包被拷贝到客户MCU上， QFirehose再计算一次，通过对比，就可以检查出固件包损坏的情况。

[002.456] md5 checking: EG25GGBR07A08M2G\_01.001V03.01.001V03/update/usrdata.ubi fail

**[002.456] find 0123456789ABCDEF012345679ABCDEF0, should be 0BBD68BBAE4CB3CF63A4A68C97257347**

[002.456] Totals checking 18 files md5 value, 1 file fail!

[002.456] qfirehose.c main 190 fail

2种情况下容易出现md5错误

1. 固件未损坏，发版本的人，替换了某个image，但未更新md5.txt，一般beta版本容易出现
2. 网络传输，U盘拷贝过程中出现了image损坏。

解决方法:

1. 最保险的方法，是找模块固件发布人或者PM，重新要固件包
2. 如果知道固件未损坏，只是md5.txt未更新导致，可以删除md5.txt里对应的行，跳过对应image的检查

## QFirehose 升级失败常见原因- 杂项

* XHCI USB零包导致的失败。 Log比较明显，升级%0就会失败

[008.499] Upgrade progress: 0

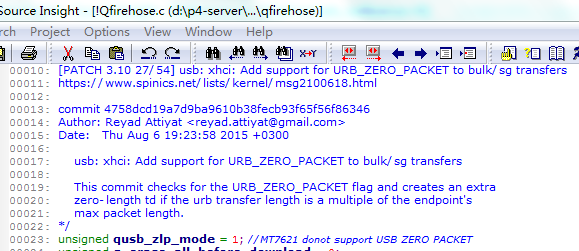
如果客户的MCU支持USB3.0且内核版本在V4.2及其以下，可能会有此bug。

Dmesg log可以看到客户的usb host controller chip是xhci/ehci/ohci等等。

[ 177.751833] usb 1-1.2: new high-speed USB device number 6 using **xhci**-hcd

用Ubuntu PC的话，最简单解决方法是安装Ubuntu16.04及其之后的版本

客户的话,源码QFirehose.c里有注释指导客户如何修复这个bug.



* USB2.0未接导致的失败。

升级的第一步，就是模块重启，进入紧急下载模式，跑的是高通芯片内部固化的PBL，只支持USB2.0.

所以高速模块(比如x20,x24，x55)，客户不仅要接USB3.0，还必须接USB2.0。

不接USB2.0导致现象就是，客户设备上刚开始USB能识别到模块，升级程序跑下，USB就识别不到模块了

* PWRKY常拉低导致的失败。

部分高通芯片(好像是X20和X55)，目前有bug，如果PWRKY拉低超过15秒，芯片就会重启。

而M.2接口定义的，PWRKEY就是常拉低的。

* 紧急下载模式下，加载了其他USB驱动，比如qcserial

[007.475] P: /dev/bus/usb/002/023 idVendor=05c6 idProduct=9008

[007.475] C: /dev/bus/usb/002/023 bNumInterfaces: 1

[007.475] I: If#= 0 Alt= 0 #EPs= 2 Cls=ff Sub=ff Prot=ff

[007.475] E: Ad=81 Atr=02 MxPS= 512 Ivl=0ms

[007.475] E: Ad=01 Atr=02 MxPS= 512 Ivl=0ms

**[007.475] Fail to claim interface 0, errno: 16 (Device or resource busy)**

**[007.475] Error: when module in 'Emergency download mode', should not register any usb driver**

[007.475] Error: it register to usb driver ' qcserial ' now, should delete 05c6&9008 from the source file of this driver

[007.475] Delete 05c6&9008 from 'drivers/usb/serial/qcserial.c' or disable qcserial from kernel config

**ASR和Intel模块的升级说明**

* ASR不提供升级协议文档和参考源码，故系统组不提供ASR固件升级工具。
* Intel升级，Intel不运行我们释放源码给客户，故客户提供交叉编译器给我们，我们编译好再给客户。

## ASR和MDM模块的网卡驱动

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 物理方式 | 模块网卡类型 | Linux 驱动 | 网卡名 | 物理层数据 |
| 物理串口  CMUX虚拟串口  USB串口 | PPP | CONFIG\_PPP | ppp0  运营商IP | PPP帧  多路复用帧   * 使用复杂，且网速上不去，模块支持USB网卡的情况下，优先使用USB网卡 |
| USB网卡 | rmnet/qmi/ndis  at+qcfg=”usbnet”,0 | qmi\_wwan&cdc\_wdm  CONFIG\_USB\_NET\_QMI\_WWAN  内核自带qmi\_wwan.c  但移远提供qmi\_wwan\_q.c, 方便客户使用。  内核版本 >= 3.4 | wwan0  /dev/cdc-wdm0  运营商IP | * 可用QMI协议实现控制命令(AT) * 默认直接传输IP帧 * 可通过QMAP功能，实现IP聚合和IP复用功能 * IP聚合，一次URB传输多个IP包，可减少CPU loading * IP复用，可实现多网卡功能 |
| GobiNet  移远提供源码 | usb0/eth0  /dev/qcqmi0  运营商IP |
| ecm  a+qcfg=”usbnet”,1  车载方案OpenLinux | CONFIG\_USB\_NET\_CDCETHER | Usb0/eth0  局域网IP | * 无法实现控制功能 * 直接传输以太网帧 * 驱动是系统自带，很老的内核都支持 * 不支持IP聚合和复用功能，不支持IP聚合功能，决定了不适用高速场景 |
| mbim  at+qcfg=”usbnet”,2  Win10平板EM/EP系列 | CONFIG\_USB\_NET\_CDC\_MBIM  内核版本 >= 3.18 | wwan0  /dev/cdc-wdm0  运营商IP | * 可用MBIM协议实现控制命令(AT) * 默认直接传输MBIM帧 * 默认就支持IP聚合 * 支持IP复用，可实现多网卡功能 |
| rndis  at+qcfg=”usbnet”,3 | CONFIG\_USB\_NET\_RNDIS\_HOST | usb0  局域网IP | * 无法实现控制功能 * 驱动是系统自带，很老的内核都支持 * 协议支持IP聚合，但Linux自带驱动没有使用，所以同ECM，不适用于高速场景. |
| PCIE网卡 |  | 移远提供源码 | pcie\_mhi0 |  |

## ASR的USB网卡使用

* ASR仅支持ECM和RNDIS
* ASR需要使用AT拨号，MDM是模块自动autoconnect的
* Android上支持RNDIS, (ECM的网卡名叫eth0，会与Android的优先以太网卡冲突)

## MDM的USB网卡

* 尽量不要让客户用ECM和RNDIS，不支持IP聚合，高速情况下，CPU 负载高
* 高速模块，不支持ECM和RNDIS
* 非EM系列模块，尽量不要推荐客户用MBIM
* Qmi\_wwan和GobiNet, 现在都是移远提供源码，所以随意
* 高速模块尽量推荐客户使能IP聚合功能 kba-20190602\_guide\_to\_use\_qmap.pdf
* 多PDN/网卡功能，使用IP复用功能 kba-20190602\_guide\_to\_use\_qmap.pdf
* 5G网络必须使用QMAP(高通目前设计)
* 尽量不要让客户使用AT$QCRMCALL，无法使用QMAP

为了支持此功能，模块内部也必须要做很多开发才支持，不同的模块版本，可能支持程度不一致。

导致各种不同的客户问题，带来技术支持上麻烦。

如要使用，GobiNet驱动下面变量需要修改为1, 修改之后不会生成设备节点/dev/qcqmiX.

./GobiUSBNet.c:94:static uint \_\_read\_mostly **qcrmcall\_mode** = 0;

./GobiUSBNet.c:95:module\_param( qcrmcall\_mode, uint, S\_IRUGO | S\_IWUSR );

Qmi\_wwan驱动没有特别修改.

* 尽量推荐客户用我们quectel-CM工具拨号，这样有问题好分析

开源的拨号工具有libqmi、libmbim、uqmi、umbim等等。



* Quectel-CM 支持QMI和MBIM拨号（自动判断网卡模式）
* Android上支持QMI和MBIM网卡。
* Android上支持AT$QCRMCALL拨号。下面2种情况下，可能不得不用

1. 部分RK芯片的OTG有bug，不能很好的支持QMI。
2. Android5.0及其之后的版本，有个bug，默认不会创建qmi\_wwan的设备节点/dev/cdc-wdm0

客户可选择修复这个bug，如果没修复，RIL自动选择AT拨号.

* QMI\RMNET\NDIS是同一个东西，不同人的叫法而已

## MDM的qmi\_wwan\_q驱动说明

之前我们的USB驱动文档，有一节专门是讲qmi\_wwan驱动移植，其实就是怎么在内核自带的源码qmi\_wwan.c上，加上我们模块的修改。

这种做法有很多不方便的地方。

1. 增加客户的工作量
2. 移植起来容易出错，比如DTR没加好，导致QMI不通， RAW-IP没加好，导致udhcp获取不到IP
3. 我们后面又增加网桥，QMAP功能，靠文档说明很麻烦。

所以我们直接提供qmi\_wwan\_q.c 给客户。

1. 客户可以像使用GobiNet，直接拷贝过去使用，只需要修改Makefile即可，非常方便。
2. 可以与内核自带的qmi\_wwan.c共存，不影响客户使用其他家的模块
3. 兼容V3.4及其之后所有的内核版本，如果在Ubuntu上使用，直接make install即可，然后重启ubuntu。
4. 网桥、QMAP功能，代码里直接用宏控制好，客户要用，直接修改对应的宏即可
5. USB驱动文档更新，只保留MDM9x07及其的高通芯片

## Quectel-CM 参数说明

* $ ./quectel-CM –h 显示参数说明
* -s [apn [user password auth]] 指定APN等参数，其中auth参数 0 表示none, 1 表示pap， 2表示chap

如果客户先用at+cgdcont和AT+QICSGP设置过，也可以不用这个参数

* -p pincode 如果sim卡有pin码，可以解pin
* -v 显示详细的QMI和MBIM数据
* -f logfile 保存log到文件中，如果客户有问题，用-v –f 两个参数来抓LOG
* -i 网卡名，如果客户系统里挂载多个模块，可以同时运行多个quectel-CM，同时拨号，通过-i 网卡名的方式选模块。
* -n pdn\_index, 指定拨第几路PDN，默认拨第一路。下面情况需要关注这个参数

1. Cdma/evdo网络，只支持一路拨号，所以这个参数是没有意义的
2. 美国Verizon网络，必须用第三路拨号，其他路有固定的作用，比如IMS之类的
3. 车载行业，存在同时公网和私网2路PDN，所以需要指定第几路拨号
4. 多网卡/多PDN拨号的应用场景. 同时运行多个quectel-CM，同时拨号，通过-n的方式指定拨第几路PDN。

* 多网卡/多PDN拨号，也叫IP复用(QMAP)，需要先运行quectel-qmi-proxy，

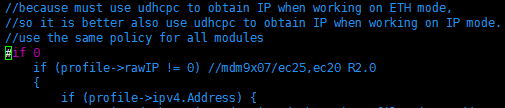
指令： ./quectel-qmi-proxy -d /dev/cdc-wdm0 &

详细见kba-20190602\_guide\_to\_use\_qmap.pdf

* -4 拨IPV4， 默认只拨IPV4
* -6 拨IPV6, “-4 -6”就是拨IPV4+IPV6
* **IP地址获取方式**

QMI拨号的时候，默认是调用busybox udhcpc获取IP地址，实际上可以通过QMI命令直接获取IP，DNS等信息。然后通过ifconfig/ip等命令直接设置到Linux系统。

可以**使能**源码udhcpc.c里如下代码段，来使能这个功能



MBIM拨号的时候，由于模块不支持DHCP(有些对比模块是支持的)，所以只支持通过MBIM命令获取IP、DNS等信息。

## QMAP功能说明

* 详细见kba-20190602\_guide\_to\_use\_qmap.pdf
* MDM9x07之前的模块，比如老EC20，UC20, 每个URB里包含一个以太网报

MDM9x07及其之后的芯片，高通改成了每个URB里包含一个或者多个IP包。

也就是文档里讲的**RAW IP**，同时引入了QMAP功能。

* QMAP 包含IP聚合和IP复用功能。

IP聚合支持一个URB里可以包含多个IP包

IP复用： 在IP聚合的基础上，包含的多个IP包，可以属于不同的PDN，这些就可以在一个物理网卡的基础上实现了多路PDN拨号。

* IP聚合的作用。假设一个URB里只能包含一个IP包，以cat4为例，最大下行150Mb/s

每秒数据需要 150Mb/1500B = 12500个URB

一个URB在客户的MCU里，就是一次中断以及后续的软件处理流程。

频繁的中断会导致客户的MCU负载很高，反过来会导致客户的网速上不去。

* IP复用： 客户需要多路PDN拨号功能，同时IP复用包含IP聚合功能。

IP复用时，再原来的网卡基础上，比如wwan0(物理网卡)。 Qmi\_wwan/gobinet驱动会在虚拟出几个网卡，名字叫wwan0.X， X从1开始，wwan0.1 对应第一路PDN，wwan0.2对应第二路PDN

## QMI不通的常见原因 – 模块本身导致的

* QMI不通的LOG如下

[09-16\_05:16:32:110] **QmiThreadSendQMITimeout** pthread\_cond\_timeout\_np timeout

* 模块包含Linux和modem2个部分，每次modem重启(包含第一次开机)，需要拉下USB的DTR，模块的QMI才能工作。
* 有些模块版本，modem初始化比较慢，qmi-wwan/GobiNet驱动拉DTR的时候，modem还没好，造成QMI不通
* 有些模块版本，换不同运营商商的SIM卡，modem会重启一次，但是模块没有重启，即USB没有断开再重新，导致QMI不通 模块发生crash，但是只是modem重启，而不是整个模块重启

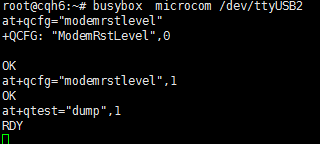
正式量产后，设置成重启模式，如下

at+qcfg=\"aprstlevel\"

at+qcfg=\"modemrstlevel\",0

* 如何判断发生了modem reset

1. AT口上突然上报了RDY



1. 抓模块的debug log，可以看到crash日志, 类似如下LOG.

**Fatal error on the modem**.modem subsystem failure reason: quectel\_common\_atc.c:2635:func: quectel\_mcfg\_sys\_restart\_handler, MBN restart .subsys-restart: subsystem\_restart\_dev(): Restart sequence requested for modem, restart\_level = RELATED.

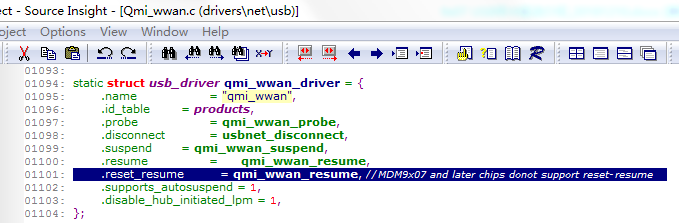
1. QMI刚开始通的，用着用着不通了。

[09-16\_05:16:32:110] **QmiThreadSendQMITimeout** pthread\_cond\_timeout\_np timeout

## QMI不通的常见原因 – 上位机导致的

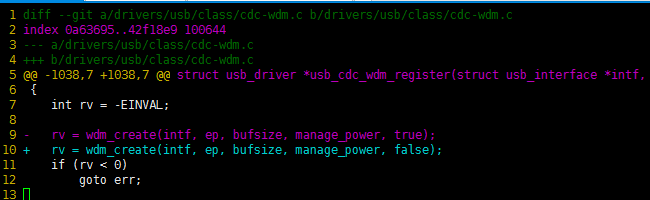
* 有些MCU（samsung、RK、MTK）在睡眠唤醒回来，会USB重新枚举模块，但只要驱动里加上了如下reset\_resume。

就不会看到网卡掉了再重新加载，但是会导致QMI不通



* 模块工作在MBIM模式，但错误的加载了QMI\_WWAN驱动，尤其是EM模块，出厂默认是MBIM模式。
* V4.9~V4.11 内核自带的cdc-wdm.c驱动bug，会导致qmi\_wwan的QMI不通

usb-serial-option/patch/cdc-wdm.patch 说明如何修复这个bug



## PCIE驱动

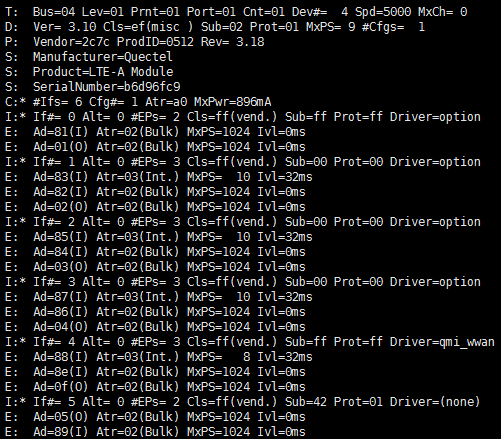
* Quectel\_Linux\_PCIE\_MHI\_Driver\_V1.2.4
* 包含PCIE串口和PCIE网卡（RMNET/MBIM）驱动
* 默认使能了IP聚合功能
* QMI/MBIM由源码Makefile里ccflags-y += -DCONFIG\_MHI\_NETDEV\_MBIM,取消注释为MBIM模式。

模块固件是同一个，不像USB接口，QMI和MBIM需要通过at+qcfg=”usbnet”切换

* 抓QXDM ./QLog –p /dev/mhi\_DIAG
* 抓DUMP ./QLog –p /dev/mhi\_SAHARA
* QMI拨号 ./Quectel-CM –d /dev/mhi\_QMI0
* MBIM拨号 ./quectel-CM –d /dev/mhi\_MBIM
* 升级 ./QFirehose –p /dev/mhi\_BHI (高通芯片和基线目前有bug, 不能用)
* 其他参考驱动包里的readme和log
* 当使用PCIE的时候，客户也可以继续使用第三方开源工具，比如libqmi、uqmi、umbim等工具，设备节点从/dev/cdc-wdm0改成对应的/dev/mhi节点即可

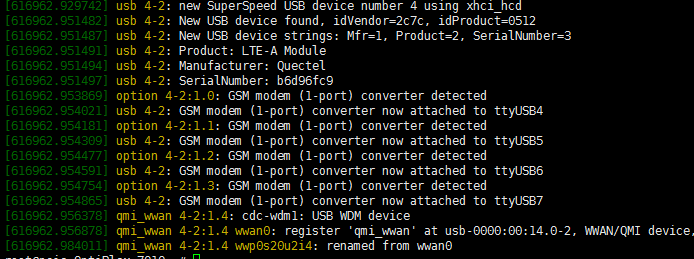
## What is USB device

# cat /sys/kernel/debug/usb/devices



Detail see [Kernel]\ Documentation\usb\ proc\_usb\_info.txt 

root@pcie-OptiPlex-7010:~# dmesg



## What is USB VendorID ProductID Product Name

C:\Users\Administrator\AppData\Roaming\Tencent\Users\7712269\QQ\WinTemp\RichOle\%SUQ_5X1}W80)5SO4~Y`AJ4.png

Vendor:

Quectel is 2c7c, Qualcomm is 056c

Product:

EM12 is 0512, EC25 is 0125, ECxx is …

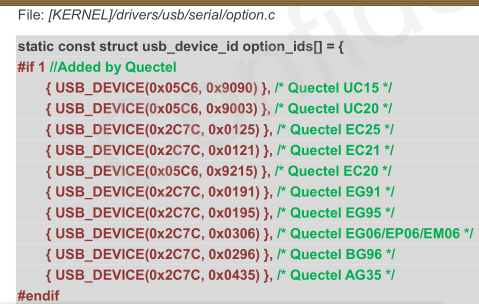
Product Name:

EM12 is ‘LTE-A Module’, if fact should be ‘EM12G’

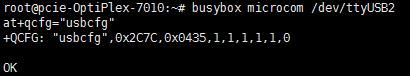
Can see more Quectel USB Product information in

<Quectel\_WCDMA&LTE\_Linux\_USB\_Driver\_User\_Guide\_V1.8.pdf>

Chapter 3.2.1. Add VID and PID



at+qcfg="usbcfg" can be used to modify VendorID&ProductID



## What is USB Interface

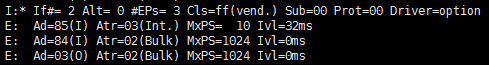
One USB device have server USB Interfaces, and have different Function per Interface.

For Quectel LTE Modules,

Generally have next interfaces and functions

0-DM 1-GPS 2-AT 3-PPP 4-usbnet(qmi/ecm/mbim/rndis) 5-adb

Next is the second USB interface of EM12



Diver=option

This usb interface attach usb-serial-option driver.

Cls=ff(vendor)

USB Interface Class and sub class define what function this interface have.



For USB interfaces have the Same USB Class and sub class can Share the Same USB driver.

usbnet cdc\_mbim, usbnet cdc\_ecm, usbnet cdc\_rndis all have own USB Class type.

cdc is the major Class

mbim/ecm/rndis is sub class

Other USB class: USB Mouse, USB Keyboard, USB Mass Storage

Usually Windows/Linux/Other OS will provide USB driver for these USB Class.

Class=ff means no Class or Vendor define Class.

USB Vendor (like Quectel) should provide USB driver for ‘ff’ Class.

Next is difference between qmi and mbim.

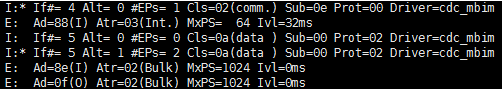
qmi class is ff

mbim class is 02, Sub Class is 0e

at+qcfg="usbnet",0 qmi, rmnet, ndis

C:\Users\Administrator\AppData\Roaming\Tencent\Users\7712269\QQ\WinTemp\RichOle\@TCJSJ0IA~41W(K12]Z7)FO.png

at+qcfg="usbnet",2 mbim



at+qcfg="usbnet",1 ecm

at+qcfg="usbnet",3 rndis

qmi: the default usbnet type for most modules.

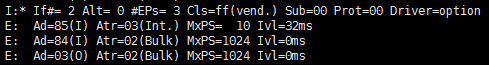
ecm/rndis: the default usbnet type for OpenLinux(Automotive) modules

mbim: the default usbnet type for EMxx modules, for Window10 notebook.

## What is USB Endpoint

Next is the second USB interface of EM12

The function is send and receive AT Commands between USB Host and EM12



One USB Interface usually have several USB Endpoint,

USB Endpoint used to transfer data between USB Host and EM12

E: Ad=84(I)

E: USB Endpoint

Ad: address, every EP have his address

(I): input, EP direction, means this EP used to receive data from EM12,

For EM12 AT, means receive AT Command from EM12

(O): output, EP direction, means this EP used to send data to EM12,

For EM12 AT, means send AT Command to EM12

Attr=02(Bulk): attribute

EP type, Bulk type used to transfer larger amount of data

Like AT command, PPP frame, IP packets, QXDM log

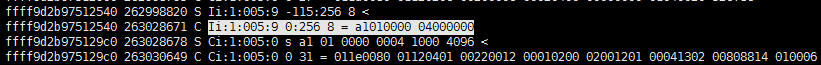
Attr=03(Int):

EP type, Interrupt type, fixed intervals time to receive Fixed Length data.

For EM12 AT, used to report RI/DCD status from EM12

For EM12 usbnet, used to report qmi/mbim message is availed inside EM12, and wait Host PC to read it form EM12

Next is example for EM12-USB-NET\_QMI\_WWAN interrupt EP message



MaxPS:

Max Packet Size

On USB Hardware level, transfer unit is Packet.

One Packet contain 1024 bytes (USB3.0), 512 bytes (USB2.0), 64 bytes (USB1.1)

## What is USB Driver

1. Export device file, let Linux Application can read and write EM12.

For next:

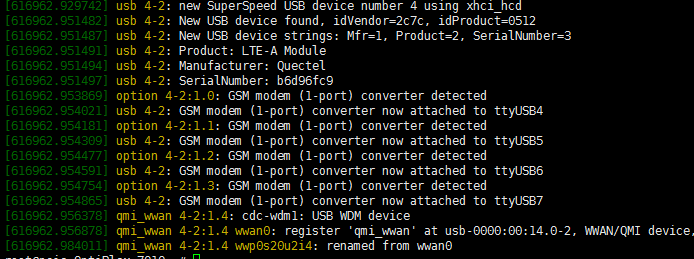
Export /dev/ttyUSB4 for DM

Export /dev/ttyUSB6 for AT

Export /dev/cdc-wdm0 for transfer QMI

Export usbnet card wwan0 for internet access

#dmesg



1. Call URB (USB Request Block) functions provided by Linux USB Host Driver to read and write USB Endpoint

An URB consists of all relevant information to execute any USB transaction and deliver the data and status back.

URB contain 3 important arguments:

2.1 EP address: which EP want to read/write

2.2 data address and length: the data want to read/write

2.3 callback function: when URB is finished, all data are transferred, Linux USB Host Driver will call this callback to tell USB driver.

## What is USB Zero length Packet

Linux USB Host Driver how to judge all data transferred?

For write data:

Logic is very simply, all data are transfer to EM12.

For read data,

For example: I want to read 2048 bytes from at command EP.

On USB Hardware level, transfer unit is Packet.

One Packet contain 1024 bytes (USB3.0), 512 bytes (USB2.0), 64 bytes (USB1.1)

Let us take 512 for example:

Linux USB Host Driver will read one packet, if the valid data length in this packet is 512 bytes,

He will continue to read next packet,

Until valid data length (length can be 0) is less than 512, he will think all data are transferred, and call the URB callback function.

So if I want to send 1025 bytes length to EM12, I want to send three Pakcet on Hardware level.

Packet1 (valid length is 512)

Packet2 (valid length is 512)

Packet3 (valid length is 1) to indicate all data are transferred.

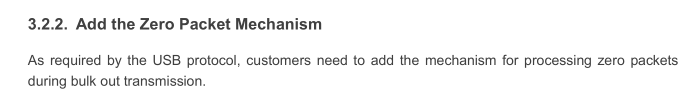
So if I want to send 1024 bytes length to EM12, I also want to send three Pakcet on Hardware level.

Packet1 (valid length is 512)

Packet2 (valid length is 512)

Packet3 (valid length is 0) to indicate all data are transferred.

Next chapter is let USB Host Driver send the last Zero Lenghth Pakcet.



## Relationship between USB Interface and USB Driver

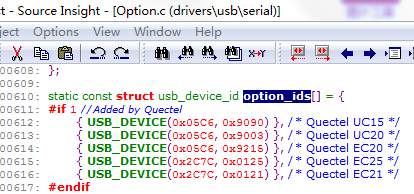
Every USB Driver have an USB interface table, means this Driver are suitable for these Interfaces.

3.2. USB Serial Driver for UCxx/EC2x/EGxx/EP06/EM06/BG96/AG35

3.2.1. Add VID and PID

Means usb-serial-option driver suitable for All Interfaces of Quectel modules.

In fact, this logic is wrong, not suitable for qmi/mbim/ecm/rndis interface.

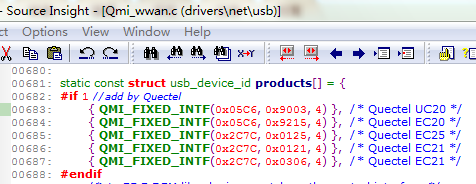


3.5. QMI WWAN Driver for UC20/EC2x/EGxx/EP06/EM06/BG96/AG35

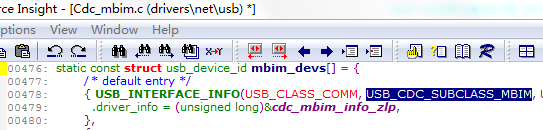
3.5.1. Add VID and PID

Means usb-net-qmi\_wwan driver suitable for interface 4 of Quectel modules.

In fact, this logic is wrong, not suitable for mbim/ecm/rndis interface (configure by at+qcfg="usbnet")



Next means usb-net-cdc\_mbim driver suitable for all interfaces which‘s USB class is MBIM class.



When a USB Driver match a USB devices.

Every USB driver have one function named XXX\_probe(), XXX\_probe() will be called.

Like option\_probe(),qmi\_wwan\_probe(),GobiUSBNetProbe(),usbnet\_probe()

## How to install Quectel USB Driver on Ubuntu-PC

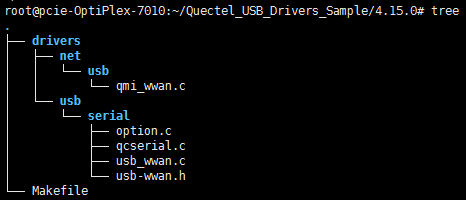
Step1: check Ubuntu’s Linux Kernel Version, for example 4.15.0

C:\Users\Administrator\AppData\Roaming\Tencent\Users\7712269\QQ\WinTemp\RichOle\E4NT1VNFSKQG[GQU@~@K$MK.png

Step2: download kernel source codes form kernel.org

<https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.15.tar.gz>

step3: extract kernel tar.gz and create next files, as next:



File: Makefile provide by Quectel

Other files copy from kernel tar.gz (download at step2)

Step4: according <Quectel\_WCDMA&LTE\_Linux\_USB\_Driver\_User\_Guide\_V1.8.pdf> to port Quectel Linux USB drivers.

In fact, we have provide lots of source files for different Kernel Version.

See Quectel\_Linux\_USB\_Drivers\_Sample\_V1.0.0.tgz

But if one NEW VID&PID not in this sample files, you have to add it by yourself.

C:\Users\Administrator\AppData\Roaming\Tencent\Users\7712269\QQ\WinTemp\RichOle\GM2122C5JWODN{NAW{UK~6U.png

Step5: just run command ‘make’ to install driver, and reboot Ubuntu-PC

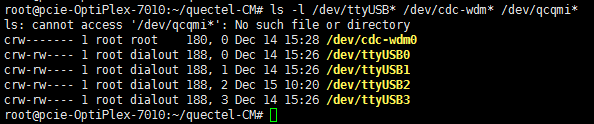
For detail information, please see Makefile or log of command make install

Or chapter 3.2.7. Build and Load Driver as a Kernel Module for PC in Linux

root@pcie-OptiPlex-7010:~/Quectel\_USB\_Drivers\_Sample/4.15.0# make install CC=gcc-5

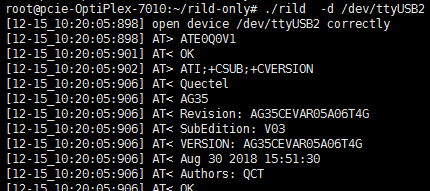
steps6: reboot PC

# ls -l /dev/ttyUSB\* /dev/cdc-wdm\* /dev/qcqmi\*



## What is AT command and QMI

AT Command:



at command format like next string line, end with ‘\r’

at+cmd=arg1,arg2,arg3,…

at command response may be have several string lines. Like next

+cmd: arg1,arg2,arg3,…

Each string line end with ‘\r’

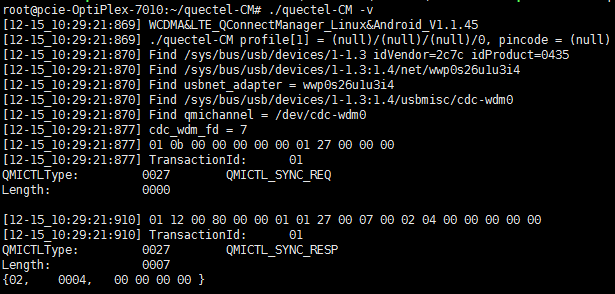
And the final response string line must be next:

OK/ERROR/CME ERROR/NO CARRIER

Argument type can be int string float

Easy for human read, but hard to transfer Complex argument, like \0 \r \n

QUALCOMM ® MSM™ Interface (QMI) is next:

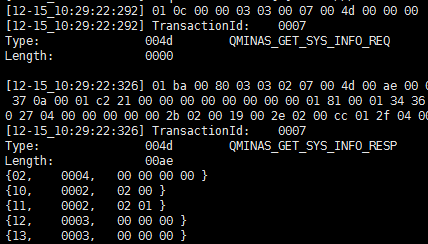


Argument format is TLV

T: argument tag or argument type

L: argument length

V: argument data



For QMI detail format, refer to next PDF



Compare to AT command:

Easy for transfer Complex argument, but hard for human read.

So there are some QMI tool help to send/receive QMI

1. Quectel-CM, for All Linux Platform, and used for Both qmi\_wwan and Gobinet
2. uqmi, only for OpenWrt, used only for qmi\_wwan

<https://oldwiki.archive.openwrt.org/doc/howtobuild/wireless-router-with-a-3g-dongle#ppp_chat_and_uqmi>

1. libqmi, for Linux Platform used glibc, like Ubuntu, used only for qmi\_wwan

<https://www.freedesktop.org/wiki/Software/libqmi>

## How to catch USB usbmon log

Detail see [Kernel]\ Documentation\usb\ usbmon.txt 

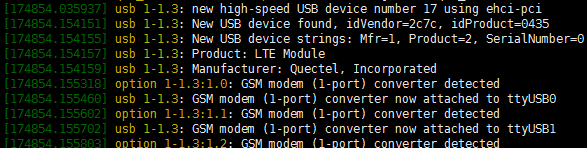
usbmon is tool can be used to catch data(at,ppp,net traffic) on the USB bus.

usbmon Is very Important and efficient tool to Analysis any USB problems

**At most time, if you report any USB issue to RD, you should send usbmon log at the same time.**

Step1: by dmesg log check to modules attach on which USB BUS., next is bus 1

# dmesg



Step2: mount USB MON driver.

For embedded Linux, enable CONFIG\_USB\_MON in kernel config

For Ubuntu

root@pcie-OptiPlex-7010:~# modprobe usbmon

root@pcie-OptiPlex-7010:~# mount -t debugfs none /sys/kernel/debug/

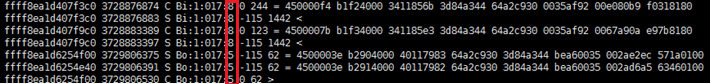
mount: none is already mounted or /sys/kernel/debug busy

root@pcie-OptiPlex-7010:~# ls /sys/kernel/debug/usb/usbmon/

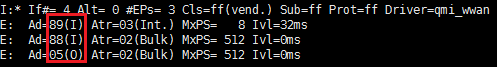
0s 0u 1s 1t 1u 2s 2t 2u 3s 3t 3u 4s 4t 4u

Step3: catch log

# cat /sys/kernel/debug/usb/usbmon/1u



# cat /sys/kernel/debug/usb/devices



By above two picture, it is easy to understand what USB Endpoint is.

## Example to Analysis USB usbmon log

Next is usbmon log of ‘ping 8.8.8.8’ on qmi\_wwan’s wwan0

ffff8ea0e7a94300 447940972 S Bo:1:017:5 -115 84 = 45000054 71784000 4001e2fe 64b17171 **08080808** 08000d7e 24c80005 737d145c

ffff8ea0e7a94300 447941113 C Bo:1:017:5 0 84 >

ffff8ea1d5eb50c0 448235255 C Bi:1:017:8 0 84 = 45000054 00000000 2801ac77 **08080808** 64b17171 0000157e 24c80005 737d145c

ffff8ea1d5eb50c0 448235264 S Bi:1:017:8 -115 1442 <

You find next KEY information:

1. RAW IP PACKET

As describe in < Quectel\_WCDMA&LTE\_Linux\_USB\_Driver\_User\_Guide\_V1.8.pdf >

Chapter 3.5.2. Add Support for Raw IP Mode for EC25/EC21/EGxx/EP06/ EM06/BG96/AG35

On the USB BUS, we directly send and receive IP Packet **(not Ethernet packet, no MAC header)**

So MAC address is not use for wwan0. The MAC address of wwan0 is Random generate when Host Linux boot up.

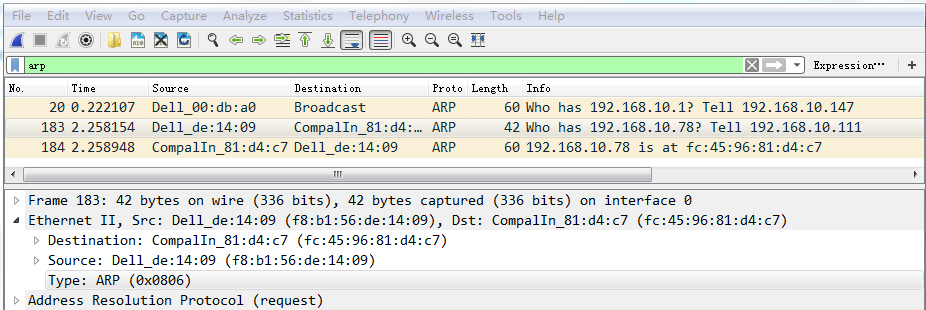
Like ARP(Address Resolution Protocol) protocol do not support by modules,

Because ARP is not IP Packet.

In fact, our modules is a point-to-point net card, not a Ethernet card

So wwan0 do not support Ethernet Bridge

Next is ARP in wireshark (<https://www.wireshark.org>)



1. ONE IP Packet in ONE URB

In IP Packet, red **0054** indicate IP Packet length

In URB, blue **84** indicate data length

ffff8ea1d5eb50c0 448235255 C Bi:1:017:8 0 **84** = 4500**0054**

Usually, ONE URB finished, means ONE USB interrupt, and URB callback function will be called.

Take LTE Cat4 modules for example.

The Max download speed is 150Mbit/s.

ONE URB contain ONE IP Packet, and the MAX MTU is 1500bytes.

So about (150Mbit /1500byte)/s URB interrupts per second,

is 12,500 USB interrupts.

Such high interrupt frequency will make Host CPU loading very high.

So we must find a way to pack multiple IP Packets in ONE URB, to reduce interrupt frequency.

The solution is:

Qualcomm Mux and Aggregation Protocol (QMAP)

## What is QMAP

Qualcomm Mux and Aggregation Protocol (QMAP)

Contain two protocol:

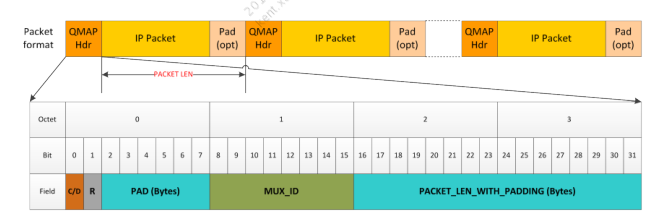
Aggregation: pack multiple IP Packet (belong to one same PDN) in ONE URB, to reduce to reduce interrupt frequency

Mux: pack multiple IP Packet (can belong to different PDN) in ONE URB, to reduce interrupt frequency.

These IP packets can belong to different PDN,

Means we can setup multiple Data Call on ONE wwan0(qmi\_wwan)/usb0(GobiNet)

Now the data in URB is next: QMAP包



Max size of MDM9x07 is 4KByte

Max size of MDM9x40 & SDX20 is 16KByte

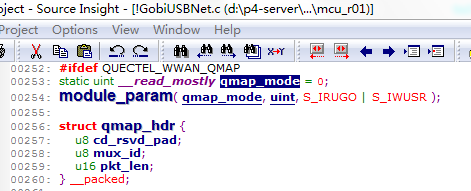
## How to use QMAP

For GobiNet:

Modify the value of qmap\_mode

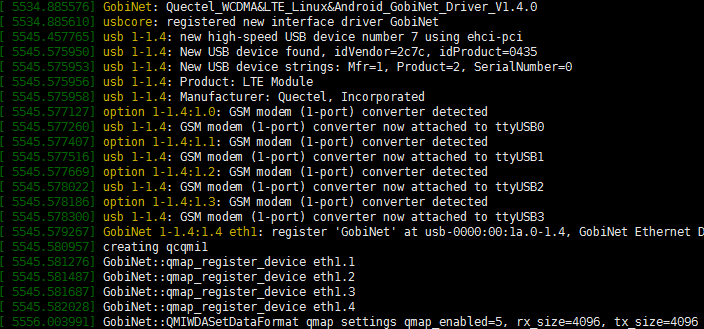
Set to 1, means enable IP Aggregation

Set to value large than 1,for example 4, means enable IP Mux



If qmap\_mode is 4.

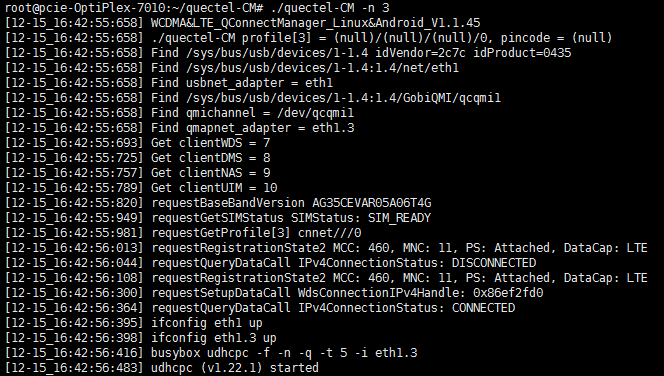
In dmesg log, you can find 4 eth1.X will be created.



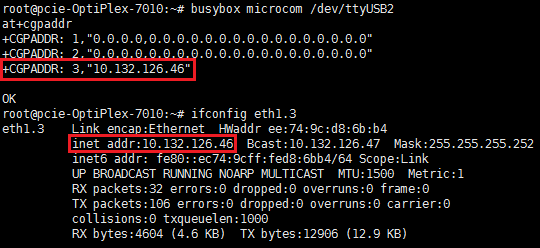
eth1.X used for PDN-X.

# ./quectel-CM -n X

Setup data call on PDN-X



Use at+cgpaddr and ifconfig to check PDN-3‘s IP address

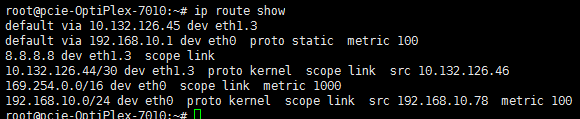


For qmi\_wwan:

Please Contact Quectel FAE&RD to get patch!

## What is IP route table

Next command show the IP route table.



When you want to access one IP address. Like ping 8.8.8.8

IP route table determine which net card will used when access this IP address

Every line in above table called **rule**

The format is:

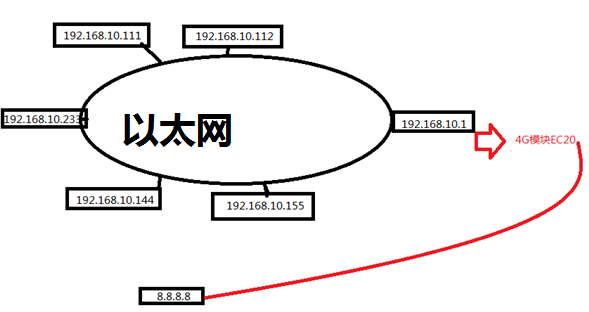
<IP range> via gateway dev netcard

If the target IP is in <IP range>, the packets send to this IP will use dev netcard.

If the target IP and your-self IP is not in the same sub net, the Ethernet Packet should first send to gateway.

And the gateway will go on send the packet to target IP.

This means: Target IP address is 8.8.8.8. But Target MAC address is gateway’s MAC address



IP range:

192.168.10.0/24 means range 192.168.10.0 ~ 192.168.10.255

8.8.8.8 or 8.8.8.8/32, means only 8.8.8.8

default, or 0.0.0.0/0, means ALL IP address, 0.0.0.0~255.255.255.255

So you want access 8.8.8.8 by wwan0,

You need add ruler as next

ip route add 8.8.8.8/32 via 0.0.0.0 dev wwan0

for our modules, gateway have no use (transfer IP packet on USB BUS), can be ignore.

Because no matter which IP address you want to access, the packets are all sent to our module.

ip route del <rule> can be used to delete

## What is DNS

The domain name system (DNS) is the way that internet website names are located and translated into internet protocol (IP) addresses.



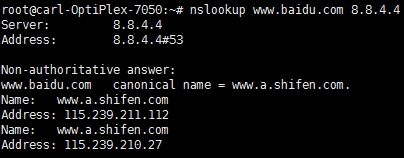
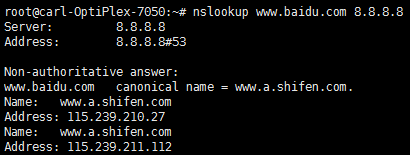
There are two DNS in resolv.conf

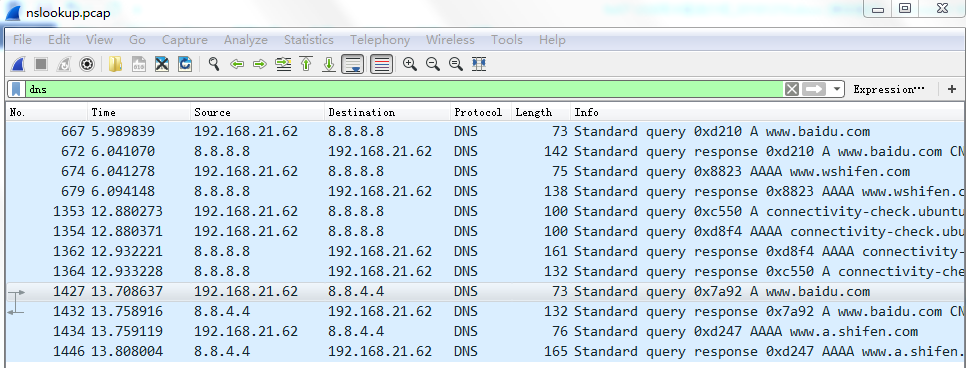
1. First use DNS1 to query IP
2. If DNS1 get fail , then use DNS2 to query IP
3. If all get fail, then cannot access website

If there are public PDN data call and private PDN data call at the same time.

Website name of public PDN should use public PDN‘s DNS

Website name of private should use private PDN‘s DNS

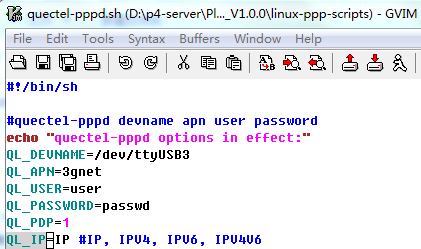




**How to Dial IPV6**

Quectel\_Linux\_PPP\_Script\_V1.0.0

quectel-pppd.sh

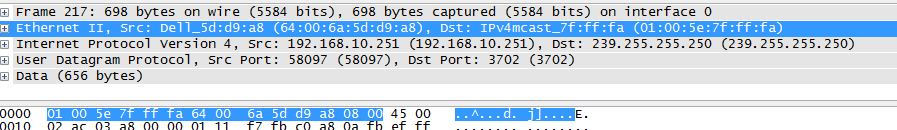


quectel-CM

|  |  |
| --- | --- |
| argument |  |
| -6 | dial IPV4V6 |
| -i wwanX/usbX | If have Multiple Quectel Modules, select one |
| -v | Print All QMI |
| -n X | Dial PDN-X |
| -f logfile | Save log to file |
| -p pincode | If SIM have PIN |

## TCP/IP 4层模型以及常用协议图

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 应用层 |  | DNS协议/DHCP协议 |  |  | 备注 |
| 传输层 | TCP协议 | UDP协议 | ICMP(ping) |  | 和网卡类型无关 |
| 网络互连层 | IP协议 | | | ARP（only for 以太网） |
| 主机到网络层 | 物理层:   1. 物理串口 2. 物理串口通过CMUX(gsm0710)虚拟出来的虚拟串口 3. USB串口 /dev/ttyUSB3 4. USB网卡   数据链路层: PPP帧、以太网帧、MBIM帧、IP帧  网线、wifi/电磁波、USB线、串口线、光钎等等，一切可以用于传递信息的物理载体都可以用于传输IP包。  网卡驱动需要根据其物理载体特性对IP包做软件封装以便方便在这种物理载体上传输。 | | | | 和网卡类型有关 |



## 9x07模块支持的网卡方式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 物理方式 | 模块网卡类型 | Linux 驱动 | 拨号方式 | 网卡名 | 物理层数据 |
| 物理串口  CMUX虚拟串口  USB串口 | PPP | CONFIG\_PPP | ppp拨号 | ppp0  公网IP | PPP帧  多路复用帧 |
| USB网卡 | rmnet/qmi/ndis  at+qcfg=”usbnet”,0 | qmi\_wwan&cdc\_wdm  CONFIG\_USB\_NET\_QMI\_WWAN  添加移远模块的vid和pid  内核版本 >= 3.4 | libqmi(Ubuntu)  uqmi(openWRT)  quectel-CM  AT$QCRMCALL=1,1  Autoconnect(不推荐) | wwan0  /dev/cdc-wdm0  公网IP | IP帧 |
| GobiNet  移远提供源码 | usb0/eth0  /dev/qcqmi0  公网IP |
| ecm  a+qcfg=”usbnet”,1  车载方案OpenLinux | CONFIG\_USB\_NET\_CDCETHER | Autoconnect  AT+QICSGP  at+qcfg="cdmaruim", 1  at+qctpwdcfg | usb0  局域网IP | 以太网帧 |
| mbim  at+qcfg=”usbnet”,2  Win10平板EM/EP系列 | CONFIG\_USB\_NET\_CDC\_MBIM  内核版本 >= 3.18 | mbim协议拨号  libmbim(Ubuntu) | wwan0  /dev/cdc-wdm0  公网IP | MBIM帧 |
| rndis  at+qcfg=”usbnet”,3 | CONFIG\_USB\_NET\_RNDIS\_HOST | Autoconnect | usb0  局域网IP | 以太网帧 |

免驱:

免驱不是说不需要驱动，而是说不要移远提供驱动。这些驱动都是操作系统自带。

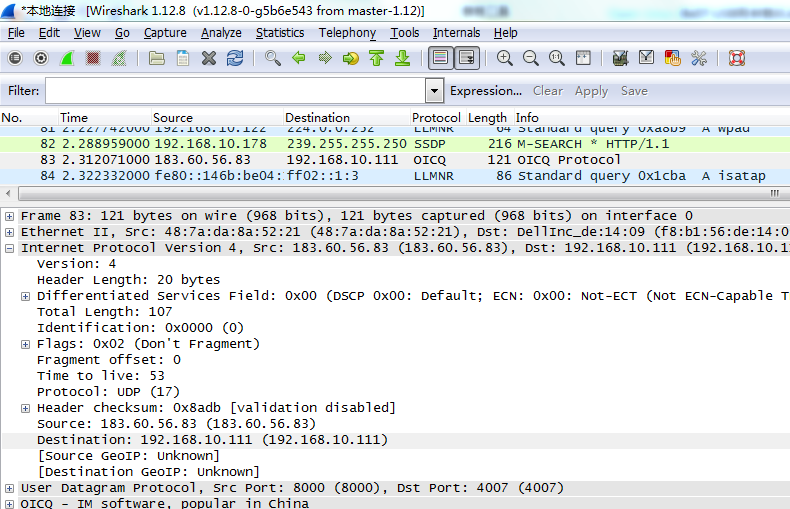
Linux必须使能对应的配置项CONFIG\_XXX。

qmi\_wwan驱动虽然是内核自带，但是移远必须在驱动代码里添加vid和pid，所以不是免驱。

更多描述见< Quectel\_WCDMA&LTE\_Linux\_USB\_Driver\_User\_Guide\_V1.8.pdf >

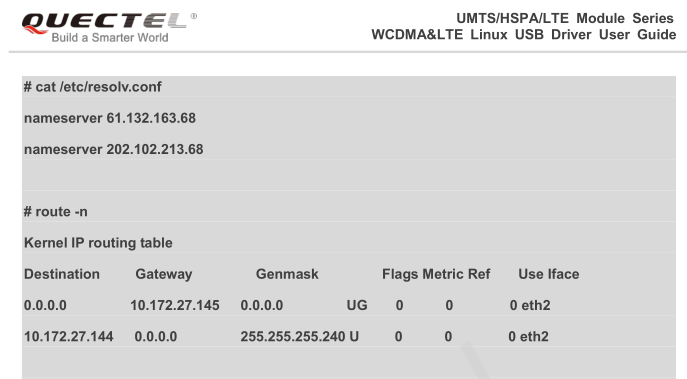
## 什么是上网

从网络互连层(IP协议层)看上网的本质就是2个网卡(或者说IP地址)之间互相收发IP包，如下图



所以为了能够上网，必须满足如下条件

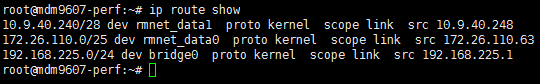
1. 本机有一个网卡且有有效的IP地址 --- 需要IP
2. 本机有访问目标IP的路由规则 --- 需要网关、子网掩码
3. 如果访问域名。本机有DNS --- 需要DNS



## 路由表的作用

Linux中含有路由表，这个表中含有路由规则: 定义了要访问的目标IP会通过哪个网卡上网。

如果你要访问的IP在这个表里找不到对应的规则，则你无法访问这个IP.



10.9.40.240/28 dev rmnet\_data1 表示目标IP 10.9.40.240/28 通过网卡 rmnet\_data1上网

192.168.225.0/24 dev bridge0表示目标IP 192.168.225.0/24 通过网卡 bridge0上网

从上面这个表中，可以看出，我们是无法ping 8.8.8.8，因为没有对应的路由规则。

|  |  |  |
| --- | --- | --- |
| 路由规则目标IP | IP范围 | 子网源码netmask |
| 8.8.8.8/32 | 8.8.8.8 | 32表示255.255.255.255 (FF.FF.FF.FF) |
| 192.168.10.0/24 | 192.168.10.0 ~ 192.168.10.255 | 24表示 255.255.255.0 (FF.FF.FF.00) |
| 192.168.10.0/28 | 192.168.10.0 ~ 192.168.10.15 | 28表示255.255.255.240 (FF.FF.FF.F0) |
| 0.0.0.0/0 | 0.0.0.0 ~ 255. 255.255.255 | 0表示0.0.0.0 (0.0.0.0) |

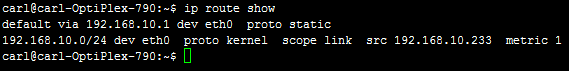
ip route show

ip route add 目标IP via 网关 dev 网卡名

ip route del 目标IP via 网关 dev 网卡名

![C:\Users\Administrator\AppData\Roaming\Tencent\Users\7712269\QQ\WinTemp\RichOle\Z002~K8@CQ]L28R](@]2U4R.png](data:image/png;base64,)

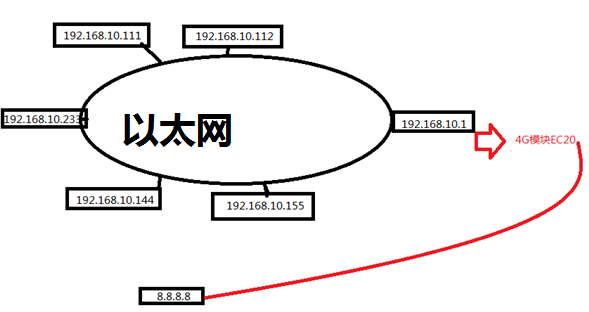
## 网关和子网掩码的作用

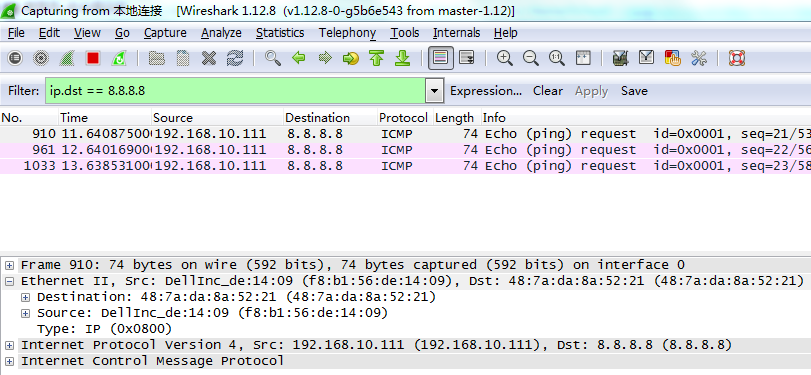


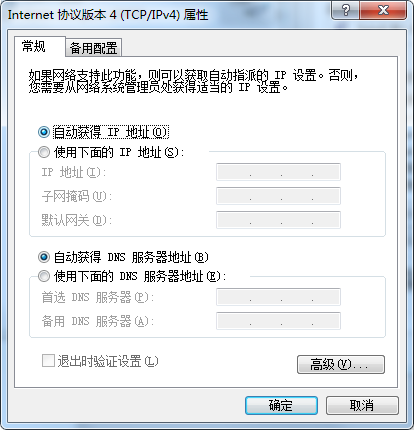
如果目标IP和本机IP在一个子网内，则不要通过网关就可以直接访问。

否则IP包需要先发送给网关，再由网关转发给目标IP.

子网掩码决定目标IP和本机IP在不在一个子网内。







## ARP的作用

以太网(局域网)帧的结构。

以太网(局域网) 为什么不直接传输IP包，而非要封装成以太网帧传输。

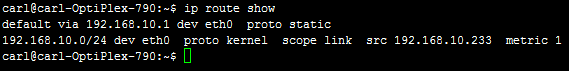
为什么网卡要设置个MAC地址。

为什么需要ARP协议

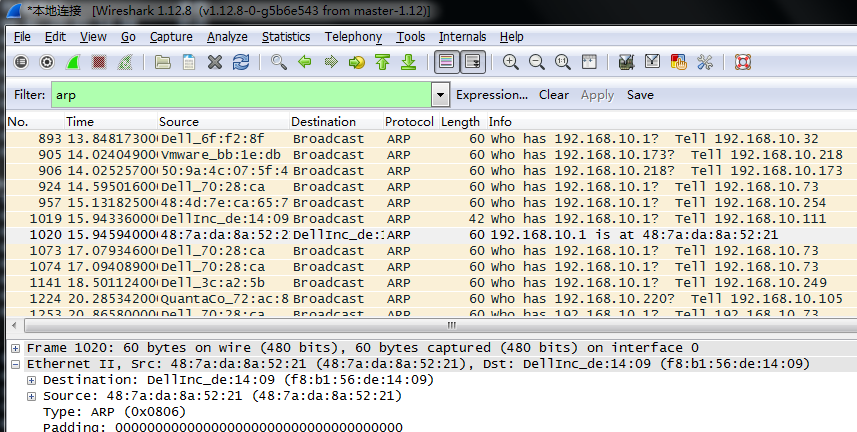
|  |  |  |  |
| --- | --- | --- | --- |
| 6 bytes 目的网卡MAC | 6 bytes 源头网卡MAC | 类型(IP or ARP or …) | IP包 or ARP包 or … |

比如我要访问8.8.8.8, 根据路由规则，我需要把IP包发给网关192.168.10.1

所以我必须先通过ARP协议查询到192.168.10.1这台机器的MAC地址，才能把IP包封装成以太网帧，然后发送给192.168.10.1



地址解析协议，即ARP（Address Resolution Protocol），是根据IP地址获取物理地址的一个协议



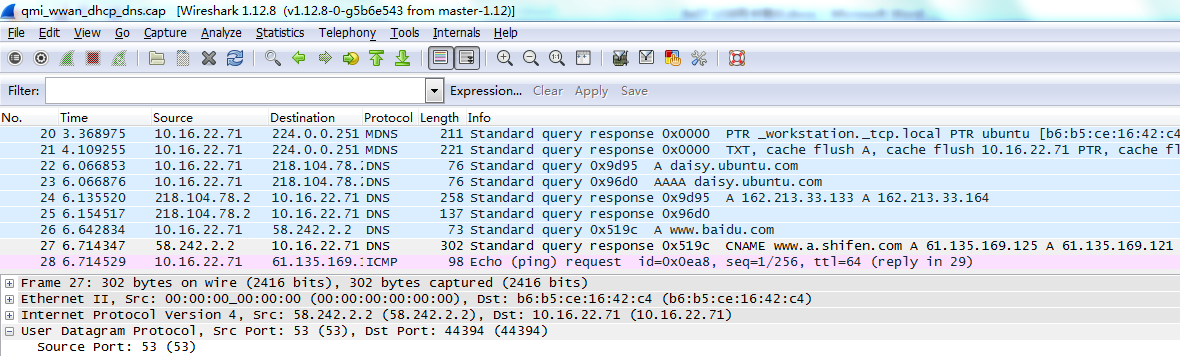
## DNS的作用

DNS 动态域名解析服务器，用于把域名(比如[www.baidu.com](http://www.baidu.com))解析为IP地址。

root@ubuntu:~# ping [www.baidu.com](http://www.baidu.com)

PING www.a.shifen.com (61.135.169.125) 56(84) bytes of data.

64 bytes from 61.135.169.125: icmp\_seq=1 ttl=53 time=96.8 ms





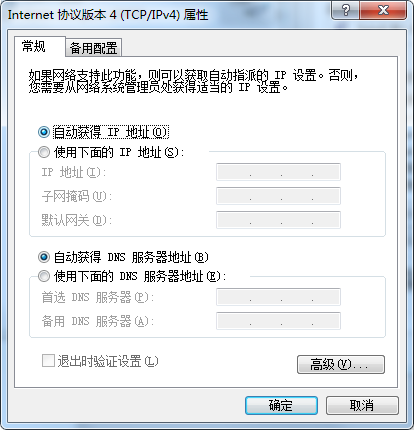
Linux系统一般把DNS保存在/etc/resolv.conf里.

但是Ubuntu、Android、Openwrt等各种系统，各自有自己的DNS保存方式



## 如何获得IP、子网掩码、网关、DNS

1. DHCP动态获取
2. 静态设置 (如何获得这些参数)



## DHCP动态获取 IP、子网掩码、网关、DNS

root@ubuntu:~# busybox udhcpc -fnq -i wwan0

udhcpc (v1.21.1) started

Sending discover...

Sending select for 10.16.22.71...

Lease of 10.16.22.71 obtained, lease time 7200

/etc/udhcpc/default.script: Resetting default routes

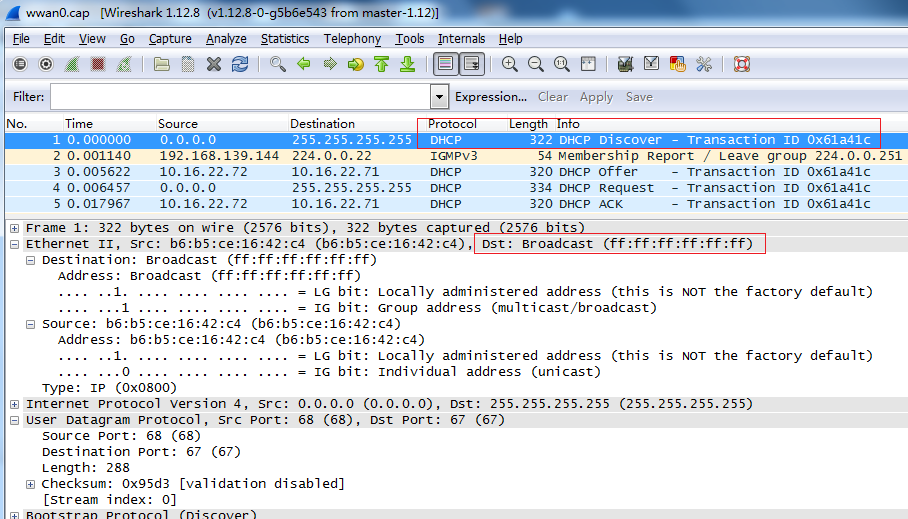
SIOCDELRT: No such process

/etc/udhcpc/default.script: Adding DNS 58.242.2.2

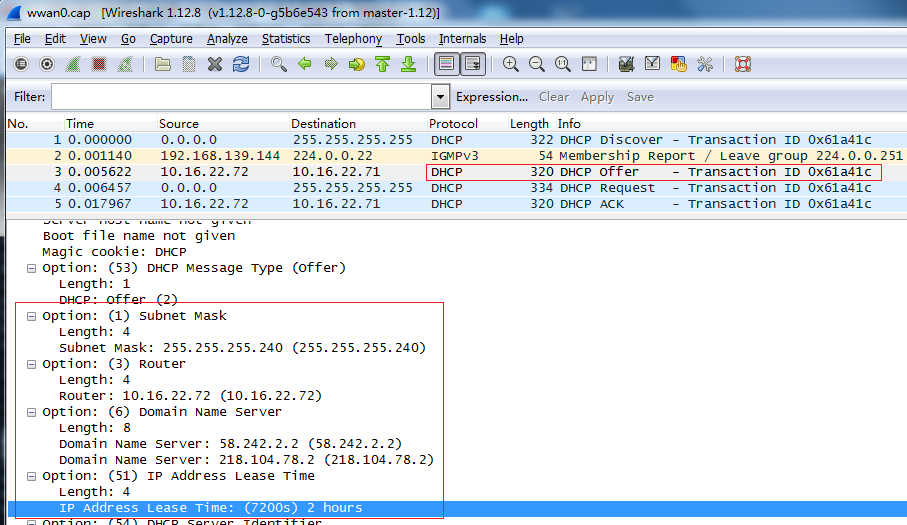
/etc/udhcpc/default.script: Adding DNS 218.104.78.2



1. DHCP协议底层是UDP协议
2. DHCP Server Discover. 由于不知道谁是网关，所以目的IP和目标MAC地址是全0xFF（广播地址）

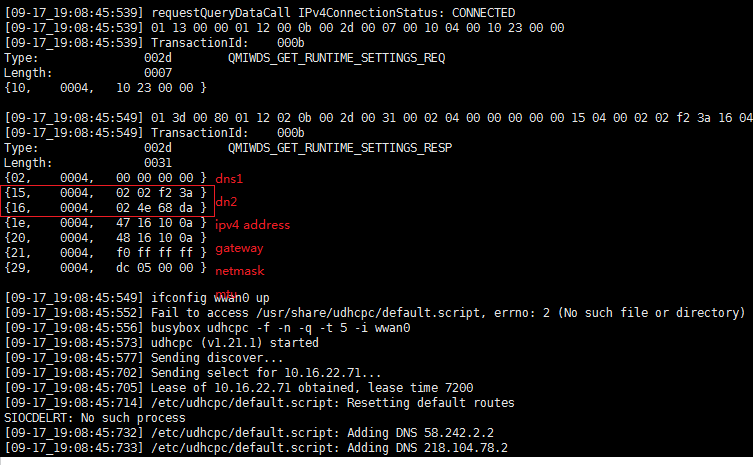


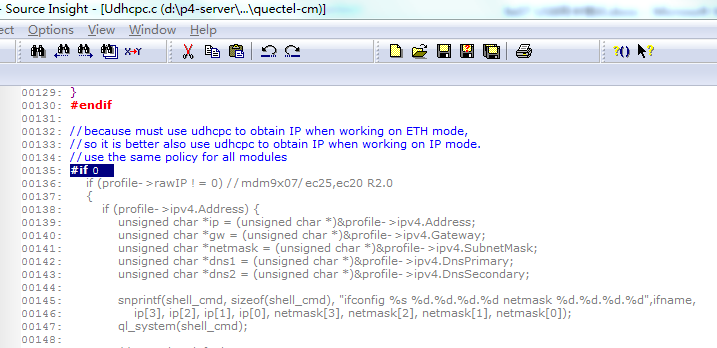
1. 网关内部运行DHCP Server(UDP Server) 网关返回 IP、子网掩码、网关、DNS。

如果通过EC20上网，那么EC20(相当于PC而言)就是网关

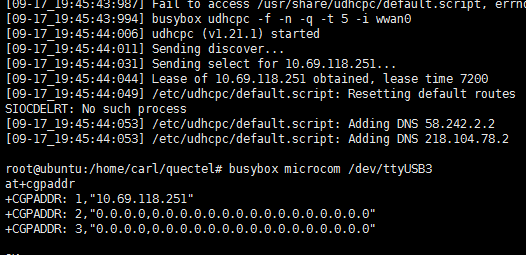
## 静态设置 IP、子网掩码、网关、DNS

./quectel-CM **-v**





还可以通过AT获得IP地址(但是不能通过AT获取子网掩码、网关、DNS)



## quectel-CM 的作用

# ./quectel-CM -s uninet

[01-01\_00:01:52:879] WCDMA&LTE\_QConnectManager\_Linux&Android\_V1.1.32

//设定拨号参数，用第几路拨号，以及APN、user、password、auth等。

//默认第一路拨号，可以使用参数 –n X， 使用第X路拨号

[01-01\_00:01:52:880] ./quectel-CM profile[1] = uninet///0, pincode = (null)

// 自动查找移远模块，以及对应网卡名字和qmi设备节点

//如果客户设备挂载多个模块，使用参数 –i 网卡名字 指定在哪个模块上拨号

[01-01\_00:01:52:881] Find /sys/bus/usb/devices/1-2 idVendor=2c7c idProduct=0125

[01-01\_00:01:52:882] Find /sys/bus/usb/devices/1-2:1.4/net/wwan0

[01-01\_00:01:52:882] Find usbnet\_adapter = wwan0

[01-01\_00:01:52:883] Find /sys/bus/usb/devices/1-2:1.4/usbmisc/cdc-wdm0

[01-01\_00:01:52:883] Find qmichannel = /dev/cdc-wdm0

//初始化QMI通道，确保可以和模块收发QMI

[01-01\_00:01:52:892] cdc\_wdm\_fd = 7

[01-01\_00:01:52:995] Get clientWDS = 19

[01-01\_00:01:53:027] Get clientDMS = 1

[01-01\_00:01:53:059] Get clientNAS = 3

[01-01\_00:01:53:091] Get clientUIM = 1

[01-01\_00:01:53:123] Get clientWDA = 1

//通过QMI获得模块版本号

[01-01\_00:01:53:155] requestBaseBandVersion EC20CFTBR02A01T4G\_OCPU\_BETA0901 1 [Nov 30 2016 04:00:00]

//通过QMI获得SIM卡状态

[01-01\_00:01:53:219] requestGetSIMStatus SIMStatus: SIM\_READY

//设置和查询拨号参数

[01-01\_00:01:53:220] requestSetProfile[1] uninet///0

[01-01\_00:01:53:283] requestGetProfile[1] uninet///0

//查询PS、数据业务注册状态

[01-01\_00:01:53:315] requestRegistrationState2 MCC: 460, MNC: 1, PS: Attached, DataCap: LTE

//查询拨号状态

[01-01\_00:01:53:347] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED

[01-01\_00:01:53:411] requestRegistrationState2 MCC: 460, MNC: 1, PS: Attached, DataCap: LTE

//如果PS注册成功，则QMI IPV4拨号，如果还要IPV6 拨号，加上参数 -6

[01-01\_00:01:53:571] requestSetupDataCall WdsConnectionIPv4Handle: 0x879ebb70

//查询拨号状态

[01-01\_00:01:53:635] requestQueryDataCall IPv4ConnectionStatus: CONNECTED

//拨号成功，则调用程序busybox udhcpc,通过DHCP协议从模块获取IP、网关、子网掩码、DNS信息

[01-01\_00:01:53:667] ifconfig wwan0 up

[01-01\_00:01:53:685] busybox udhcpc -f -n -q -t 5 -i wwan0

[01-01\_00:01:53:702] udhcpc (v1.20.2) started

[01-01\_00:01:53:718] Sending discover...

[01-01\_00:01:53:723] Sending select for 10.46.250.77...

[01-01\_00:01:53:726] Lease of 10.46.250.77 obtained, lease time 7200

//程序busybox udhcpc在成功获取IP等信息之后，通过调用脚本/usr/share/udhcpc/default.script为Linux系统设置默认路由，DNS等。

[01-01\_00:01:53:743] deleting routers

route: SIOCDELRT: No such process

[01-01\_00:01:53:763] adding dns 58.242.2.2

## 通过usbmon抓取USB总线数据 和 tcpdump抓网络包分析问题

tcpdump用于抓取网卡上数据包

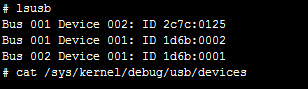
# tcpdump –i wwan0 –w 1.cap & // 1.cap 可以拿到Windows上用Wireshark 打开分析

usbmon用于抓取usb总线上的数据

1. 先确保Linux支持debugfs，并挂载debugfs



1. 通过lsusb命令确保模块挂载到哪条USB总线上

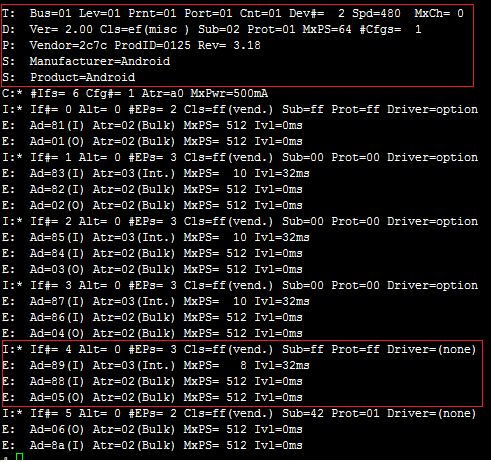


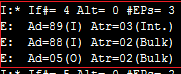
1. 或者通过# cat /sys/kernel/debug/usb/devices 确认模块的挂载在哪条总线

以及确保USB网卡的usb endpoint号。

挂载在哪条总线(Bus)是有客户板子的硬件决定的。

模块的USB网卡固定在模块的USB第4个interface，endpoint也是固定不变的。





interrupt中断端点9，用于模块上报模块内部有qmi可读取

input输入端点8，用于从模块读取IP包

output输出端点5，用于往模块发送IP包。

## RAW IP模式 – ecm 和 qmi\_wwan的区别之一

模块使用qmi\_wwan的抓包，直接传输IPV4包，也即是所说的RAW IP模式

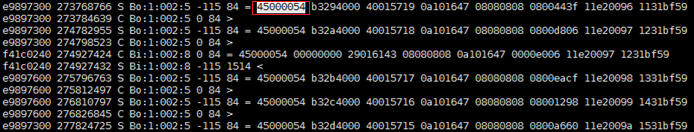
# cat /sys/kernel/debug/usb/usbmon/1u //用于抓取USB总线1上的数据包

# lsusb

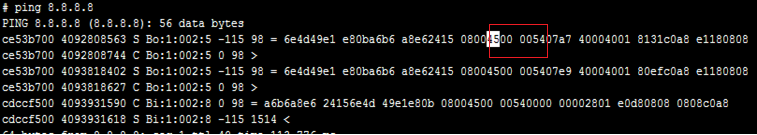
Bus 001 Device 002: ID 2c7c:0125

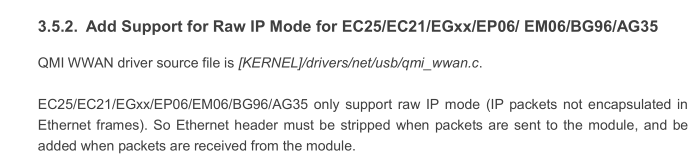
Bo:1:002:5 是发送给模块的包

Bi:1:002:8 是从模块收到的包



模块使用ecm的抓包，IPV4包被封装在以太网帧中





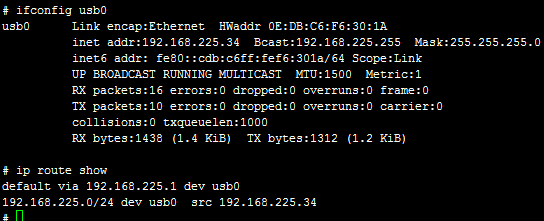
函数qmi\_wwan\_tx\_fixip()/qmi\_wwan\_rx\_fixup()用于转换以太网帧和IP帧

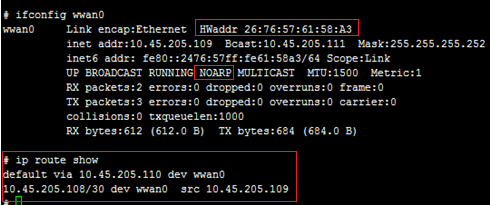


既然qmi\_wwan直接传输的是IP包:

1. MAC地址是没有意义的 --- 上位机Linux开机随机生存的一个MAC地址。即使一台PC上连接多个EC20，所有的EC20 USB网卡的MAC地址都是一样的。
2. 既然没有MAC地址，那么ARP也是没意义的
3. 网关也是没意义的，因为上位机Linux的IP包只能发给模块，没有其他选择

ECM模式

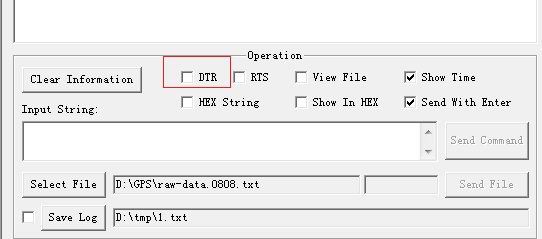




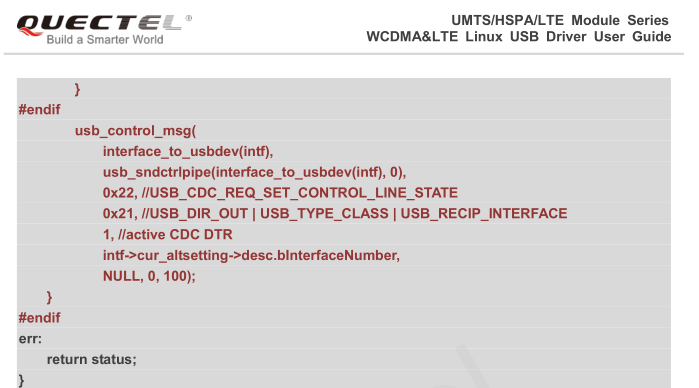
ECM模式的网关是192.168.225.1 子网掩码是255.255.255.0

QMI\_WWAN模式的网关是10.45.205.110 子网掩码是255.255.255.252

## RAW IP模式 – 必选拉高DTR才能收发QMI和使能网卡模式



* + 1. Add Support for Raw IP Mode for EC25/EC21/EGxx/EP06/ EM06/BG96/AG35



AT+QCFG="modemrstlevel",0 控制模块内部modem reset的时候，会不会重启整个模块。

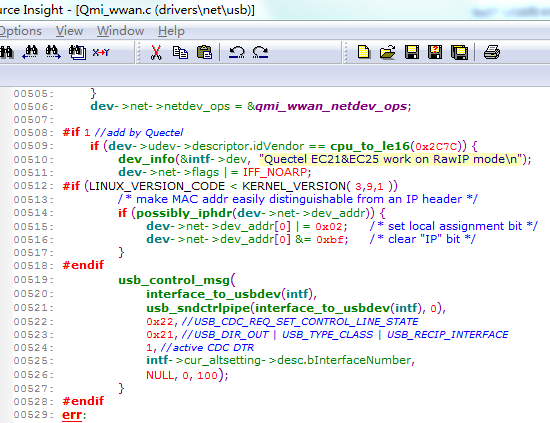
比如模块换运营商SIM卡，然后第一次开机的时候，会modem reset一次。

这种情况下需要重新拉高DTR，初始化QMI和网卡功能。

但是USB没断开，上位机PC压根不知道。所以网卡不能用

## qmi\_wwan/Gobinet网卡驱动的作用

1. 在初始化的时候拉高DTR，使得可以收发QMI和使能网卡模式
2. 设置网卡不支持ARP



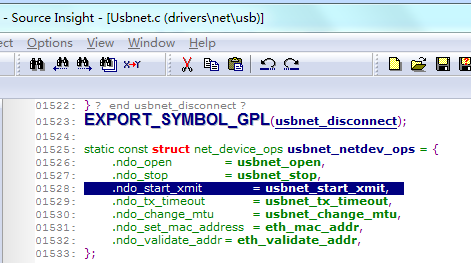
1. 建立字符设备文件，使得可以收发QMI，从而使得qmi工具可以通过QMI协议拨号

/dev/cdc-wan0 qmi\_wwan/cdc\_wwan.c

/dev/qcqmi0 GobiNet/QMIDevice.c

1. 注册一个网卡，并提供收发数据包的函数。

对于USB接口的网卡来说，基本都是一样的，直接用内核现成的。



1. 提供fixup()函数:

tx\_fixup把从内核网络协议栈收到的以太网帧转成IP帧，然后通过USB总线发给EC20。

rx\_fixip 把从模块收到IP帧转成以太网帧，再发给内核网络协议栈

rx\_fixup 里需要虚构一个以太网帧头。

|  |  |  |
| --- | --- | --- |
| 6 bytes 目的MAC | 6 bytes 源头MAC | 类型(IP or ARP or …) |

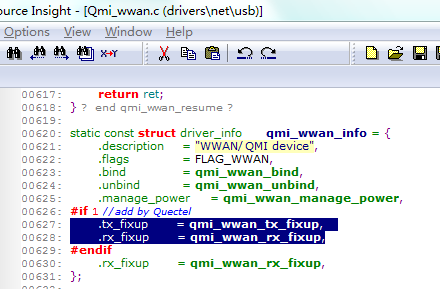
类型: 0x8000, 表示IP帧 ?

目的MAC地址就是wwan0网卡的mac

源头MAC地址

qmi\_wwan里使用00:00:00:00:00 (有的客户的内核不认这个MAC导致不能上网)

GobiNet里使用02:50:f3:00:00:00



## Ubuntu上安装EC20 USB 驱动

如果是ECM、MBIM、RNDIS都是内核自带的，免驱，没啥好说的，插上就用

如果是usb转串口、qmi\_wwan、GobiNet

参考文档 Quectel\_WCDMA&LTE\_Linux\_USB\_Driver\_User\_Guide\_V1.8.pdf

一般流程都是

1. 获取你的板子对应的内核源码
2. 按照文档修改源码
3. 编译内核
4. 烧写新的内核到你的板子

Ubuntu上我们这上面的步骤简化成一个命令:

use\_quectel-modules\_on\_ubuntu.docx

carl@ubuntu:~/quectel\_linux/Quectel\_USB\_Drivers\_Sample/3.19.0$ **sudo make install**

rm -rf \*~ .tmp\_versions modules.order Module.symvers

find . -type f -name \*~ -o -name \*.o -o -name \*.ko -o -name \*.cmd -o -name \*.mod.c | xargs rm -rf

make -C /lib/modules/3.19.0-25-generic/build M=/home/carl/quectel\_linux/Quectel\_USB\_Drivers\_Sample/3.19.0 modules

……

## Linux使用EC20 USB网卡的3种方式

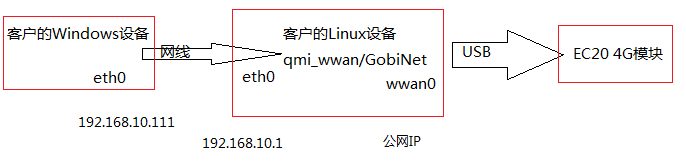
1. 常规方式

Linux通过USB接模块，Linux上qmi拨号，Linux上网

1. 路由器NAT方式

类似与第一种方式，客户的Linux设备可以通过EC20上网。

同时Windows PC也可以通过EC20模块上网，但是需要客户的Linux设备转发。起个网关的作用



下面2条命令可以把一台Linux设备网关网关

//使能IP转发，默认Linux会直接丢弃一个目的IP地址不是自己的IP包。

//如果使能IP转发。则Linux会在路由表中找一个满足目标IP条件的网卡，转发出去。

echo 1 > /proc/sys/net/ipv4/ip\_forward

//如果IP包从wwan0转发出去，则这个IP包的源IP必须伪装成wwan0的ip

iptables -t nat -A POSTROUTING -o wwan0 -j MASQUERADE --random

客户的Windows设备上网。假设ping 8.8.8.8.

Linux设备上抓wlan0的包，则这个IP包的源地址是192.168.10.111

Linux设备上抓wwan0的包，则这个IP包的源地址必须变成wwan0的IP地址.

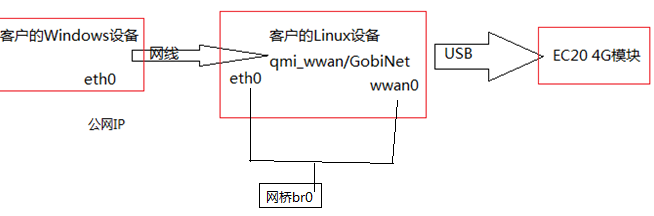
NAT（Network Address Translation，网络地址转换）

1. 网桥模式上网

Windows PC直接获取公网IP，这样在Window PC上建立的tcp socket server，可以被外部公网访问。

这种情况下客户的Linux设备不能通过网卡wwan0上网。

因为公网IP只有一个，被分配给Windows PC上的网卡，则wwan0就没有IP了。



客户的Linux设备上，eth0网卡收到的以太网包，直接原样转发给EC20网卡wwan0。

1. 客户的Windows PC上是以太网卡eth0，所以他发送以太网帧之前，会先发送ARP请求，得到网关IP对应的MAC地址。

而我们模块是工作在RAW IP模式，并不处理ARP请求。那就需要GobiNet/qmi\_wwan驱动来回复这个ARP请求

1. 从EC20 4G收到IP包时，需要添加以太网帧头，这个时候目的MAC地址，就必须是Windows PC的MAC地址了。

|  |  |  |
| --- | --- | --- |
| 6 bytes 目的MAC | 6 bytes 源头MAC | 类型(IP or ARP or …) |

## 9x07内部的网桥

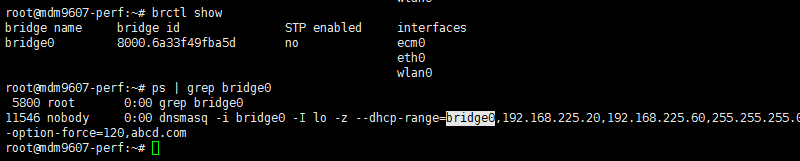
90x7可以通过usb/ecm上网，对应模块内部的网卡ecm0

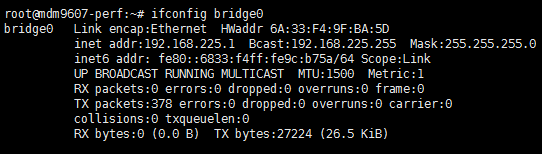
9x07可以通过wifi 热点上网，对应模块内部的网卡wlan0

9x07可以通过网线上网，对应模块内部的网卡 eth0

这3种完全不同的网卡上网方式，其对应的IP地址都是192.168.225.X.

说明他们都是在同一个子网内，是可以互通的。





实现原理就是把3个网卡都加到同一个网桥里面。

这3个网卡组合成一个新的网卡bridge0.

任意一个网卡收到的以太网包都会转发给网桥里的其他网卡。

各个终端从模块获取到的网关地址192.168.225.1, 子网掩码是255.255.255.0， DNS是192.168.225.1

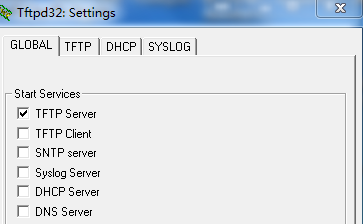
模块内部的dnsmasq

即是DHCP server，负责给终端分配IP地址、网关、子网掩码、DNS信息

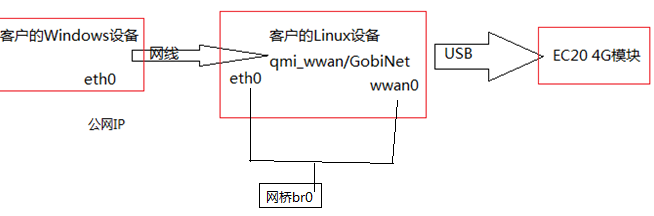
又是DNS代理服务器，收到中断的DNS解析请求，转发给真正的DNS服务器。

注意这是ECM模式下的DHCP server，qmi\_wwan/GobiNet模式下，模块的dhcp server运行在modem内部。

下面这个小软件，可以把你自己的Windows电脑变成DHCP server，给别的电脑分配IP



## GobiNet/qmi\_wwan支持网桥



客户的Linux设备上，eth0网卡收到的以太网包，直接原样转发给EC20网卡wwan0。

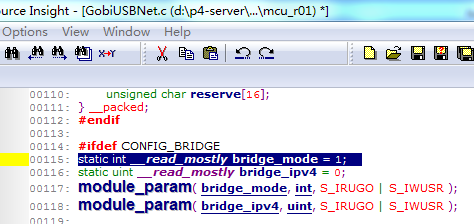
1. 客户的Windows PC上是以太网卡eth0，所以他发送以太网帧之前，会先发送ARP请求，得到网关IP对应的MAC地址。

而我们模块是工作在RAW IP模式，并不处理ARP请求。那就需要GobiNet/qmi\_wwan驱动来回复这个ARP请求

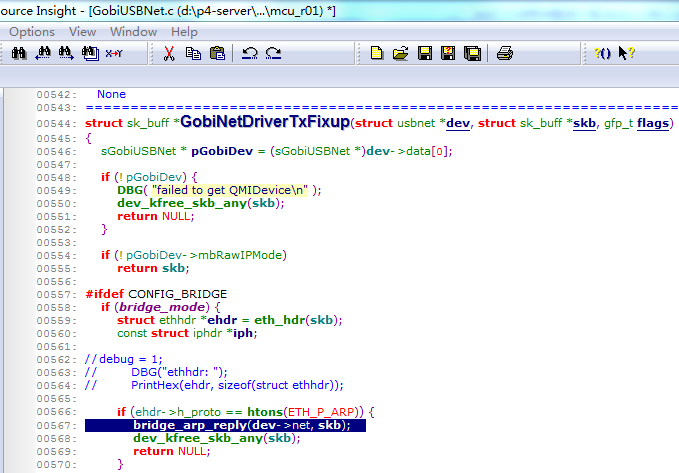
1. 从EC20 4G收到IP包时，需要添加以太网帧头，这个时候目的MAC地址，就必须是Windows PC的MAC地址了。

|  |  |  |
| --- | --- | --- |
| 6 bytes 目的MAC | 6 bytes 源头MAC | 类型(IP or ARP or …) |

1. 使能网桥支持

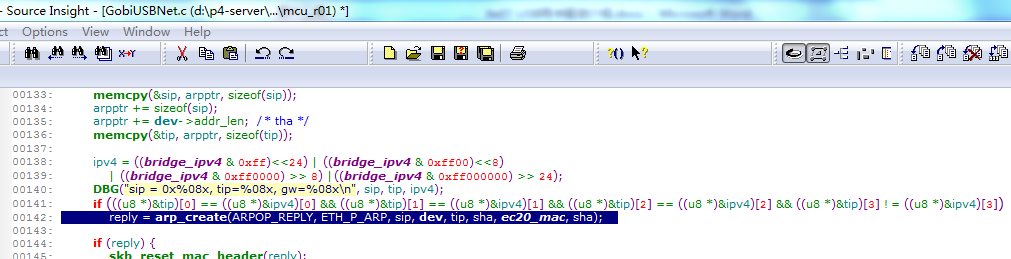


1. GobiNet驱动里收到Windows PC发送过来的ARP请求，自己构造ARP REPLY，发给Windows PC



1. 如何构造 ARP reply。

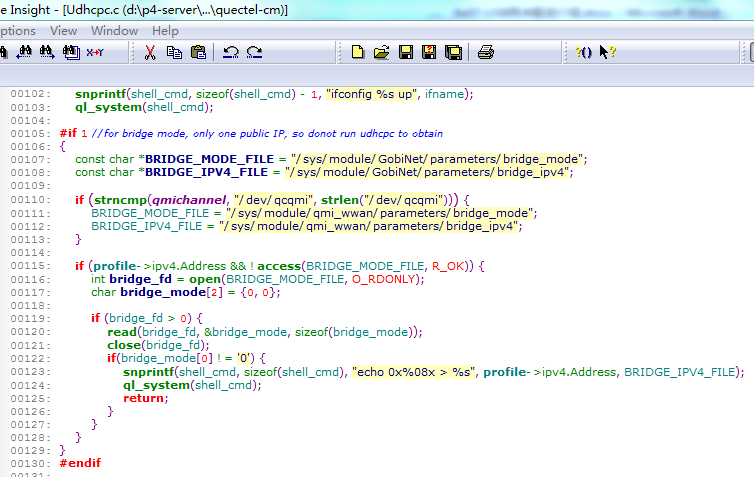
假设模块拨号获取IP是10.1.1.2, 则10.1.1.0/24网段里除10.1.1.2之外的IP对应的MAC地址都是我们EC20。



1. 驱动如何获得IP 10.1.1.2

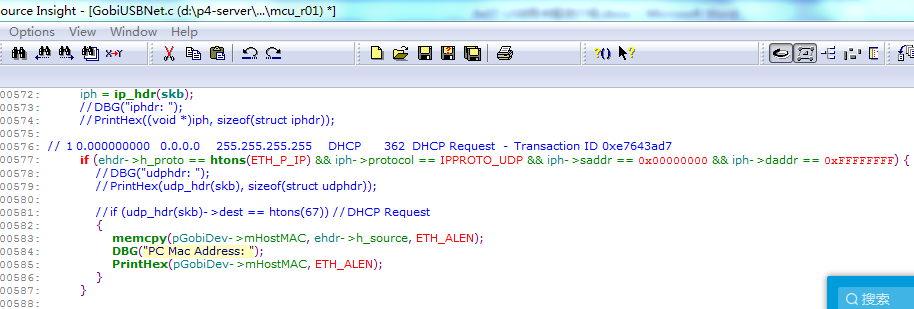
Quectel-CM通过QMI拨号成功之后，再通过QMI获取IP地址，然后告诉GobiNet/qmi\_wwan驱动。

注意不能调用dhcp程序去获取IP地址，因为公网IP只有一个，如果Linux通过dhcp获取到了，则后面Windows PC就无法通过dhcp获取到了。（模块内部的的dhcp server设定如此）



1. 如何获得Windows PC的mac，使得可以把从模块收到IP包，转成发给Windows PC的以太网包。

其实目的mac地址也可以用FF:FF:FF:FF:FF:FF 即广播包，Windows PC也可以接收到。



## GobiNet/qmi\_wwan 一次URB传送多个IP包 (该方法废弃，使用QMAP)

LTE CAT4下载速率是100Mbit, 12.5MBytes.

USB网卡通用做法是一次urb只传输一个IP包，最大1500bytes。

一次 urb 传输就是对应一次MCU中断，以及后续软件处理流程。

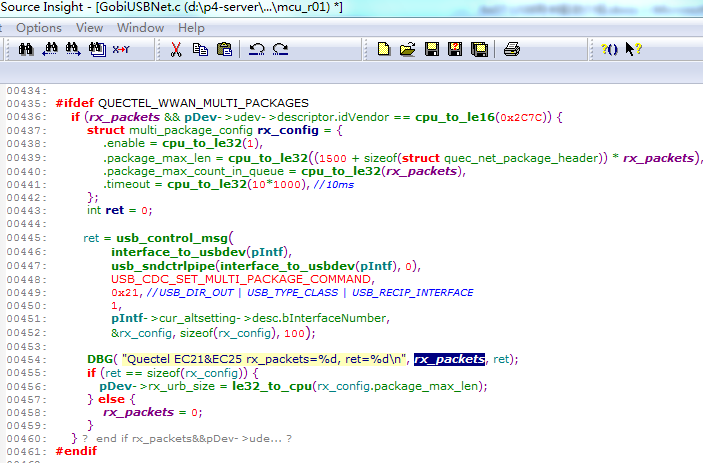
如果下载速率接近理论速度。则每秒会有 12.5MB/1500B = 8333次中断。

在一些低频率的MCU上, 如此高的频率会导致cpu loading非常高。

会导致cpu发烫，功耗高、以及网速上不去，等等各种问题。

所以需要驱动和模块配合，一次URB中传输多个IP包，来降低中断频率。

1. 初始化的时候，告诉模块一次最多封装 rx\_packets 个IP包。



1. 从模块收到的URB中，解析出多个IP包，封装成以太网包，提交给内核网络协议栈

