功耗相关分享

一、手机耗电相关

主要耗电因素：

屏：屏种类，亮度

CPU：平台，运行核数与频率

GPU：平台，运行时频率

网络:wifi 、数据流量工作状态

信号：信号强弱，

GPS

Camera

Audio

各种sensor

二、功耗相关数据获取

由于功耗涉及很多模块，测试过程麻烦，测试过程中可能各种环境不一致而导致结果不一致，所以最好测试到异常时就能够抓取数据，同时确认是哪个具体场景才出的问题。

抓取log过程：

1.将手机时间与电脑时间统一同步网络时间

2.开启MTKlog，关闭ModemLog开关

3.开启功耗dump信息：

```
adb shell dumpsys batterystats --reset
adb shell dumpsys batterystats --enable full-wake-history
```

4.发现耗电异常后，执行如下命令：

```
adb shell dumpsys batterystats > f:\battersystats.txt
adb shell  cat /sys/kernel/debug/wakeup_sources >   f:\wakeup_sources.txt
adb shell bugreport > f:\bugreport.txt
```

5.关闭mtklog，保存电流图

三、功耗分析

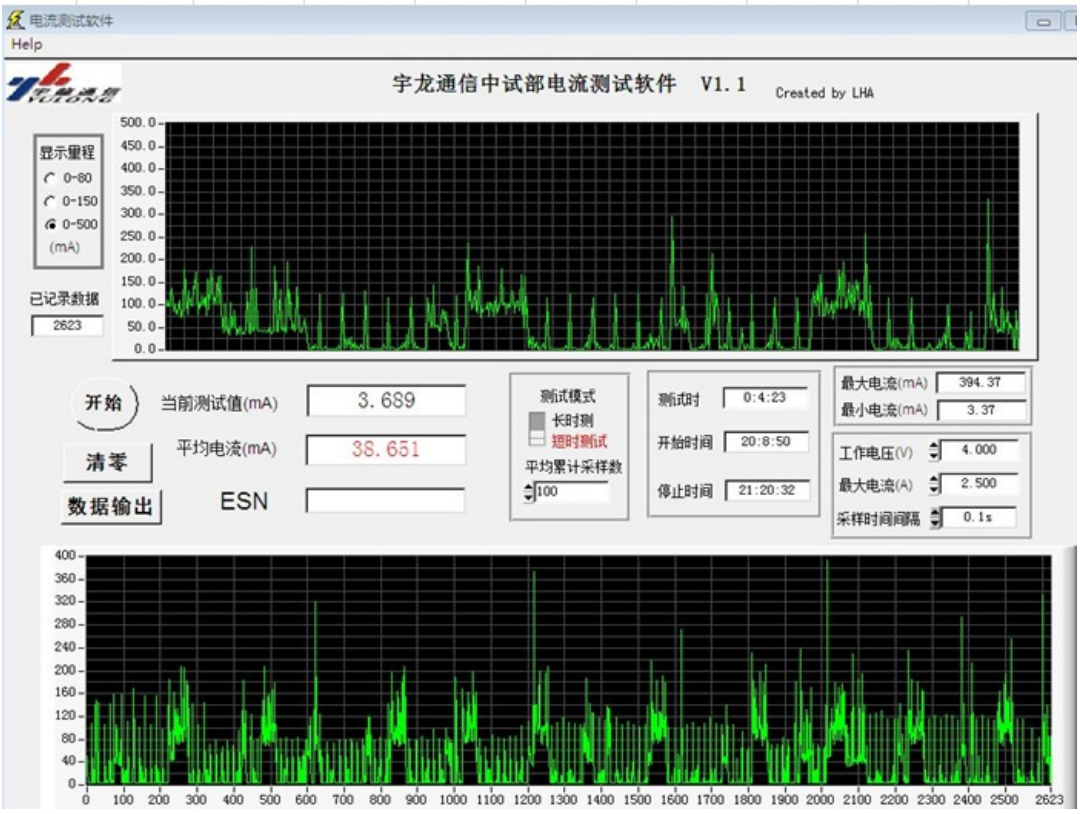
3.1 分析数据与工具

目前我们功耗测试主要是项目量产前会测试整机功耗。以下为测试用例与标准。

然后就是客户反馈功耗异常时，我们根据场景来来复现电流异常。

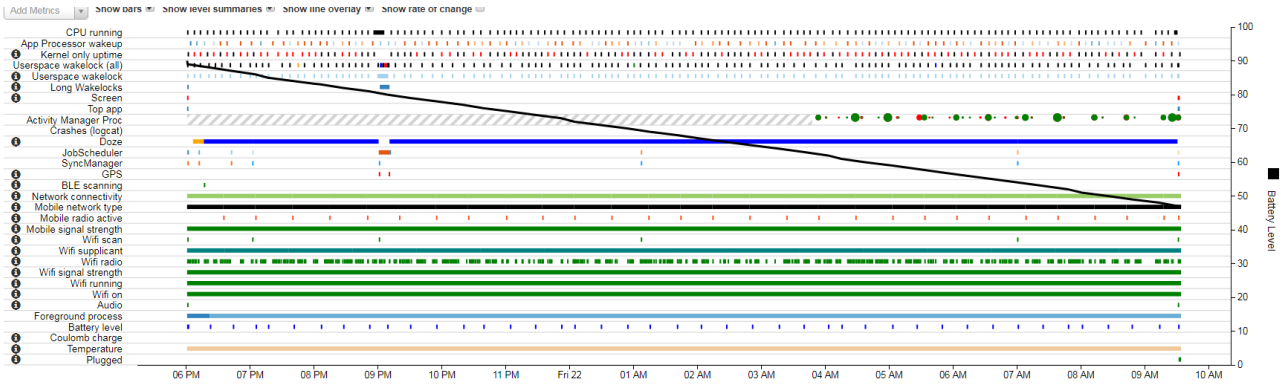
我们的电流测试目前主要是通过电源来获取当前电流。

测试给出电流图基本如下：



然后我们需要根据电流图来确定哪些时间点的电流过高，然后查找这段时间的log来分析原因。

对于不接电源而是使用电池来测试的，测试完成后，在关机前，我们必须抓取bugreport与mtklog，通过工具解析bugreport出来，然后确认哪段时间比较异常，此工具是google提供的，具体搭建方式可参考http://www.cnblogs.com/jytian/p/5647798.html



3.2 分析切入点

3.2.1 亮屏情况功耗异常：

影响因素多，而且也没有比较好的控制措施。

大致分析流程：

1.先验证亮屏情况下，IDLE待机是否异常，然后一个场景一个场场景叠加，确认是哪个场景功耗异常

2.找同平台客户对比机验证，判定是否共性问题

3.亮屏情况下更多的需要硬件与驱动确定各个模块的功耗是否正常，我们主要在cpu消耗过高的时候看是否有进程异常。

3.2.2 我们目前更多的是分析待机功耗这一块。待机情况下功耗主要有以下几个点：

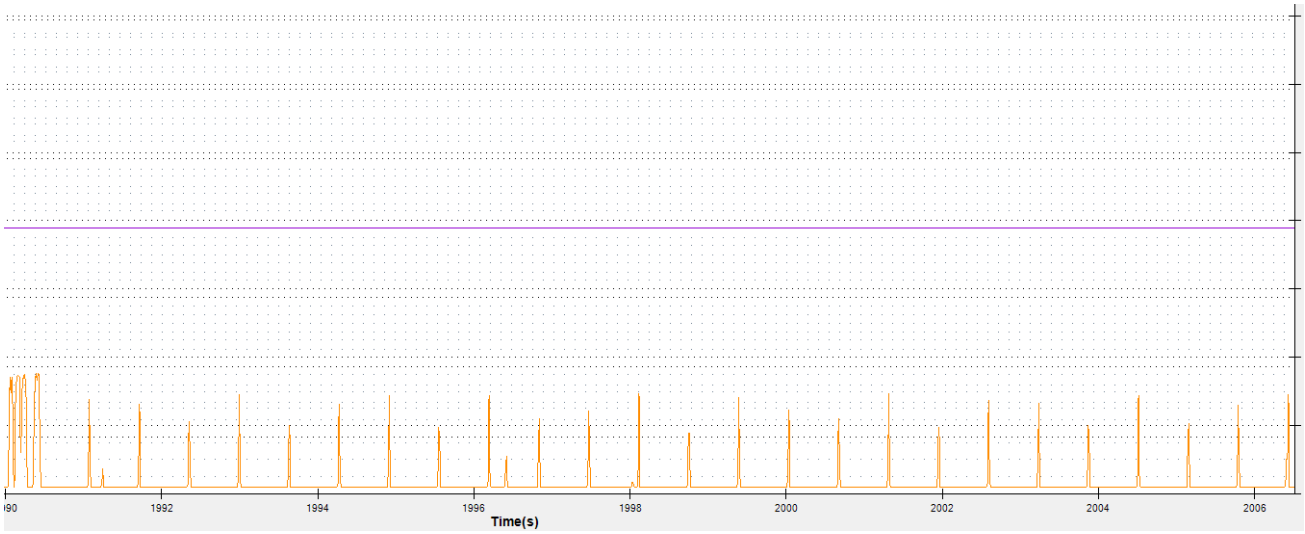
1.modem工作

正常情况下，手机需要不断paging来确保网络可以在任何时间找到UE,并发起业务，不同网络的时间间隔也不一样。

一般我们只需要保证所处环境信号良好，手机有正常校准，写入imei，然后正常待机的电流符合标准即可。

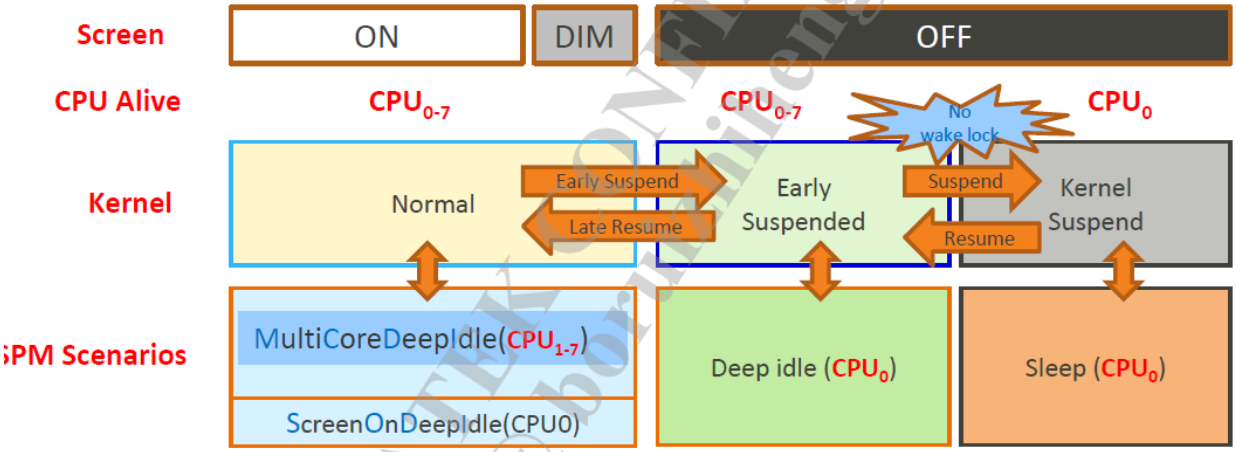
通过开关飞行模式来确认modem是否有异常，这块异常交由射频处理。

正常的电流图如下



2.唤醒源唤醒cpu工作

3.持有wakelock导致无法休眠



手机灭屏之后，当所有wake lock释放完之后会进入suspend，之后由spm控制，而从suspend状态中resume回来的前提是先把cpu唤醒。

wakelock 相关

如果有模块持有wakelock一直不释放，系统就无法进入suspend状态，电流会一直保持在一个比较高的状态，耗电就会变快。

如何确认wakelock导致

1.从电流图看，电流一段时间一直保持在一个比较高的位置，或者直接从抓取的bugreport.txt中搜索wake lock

```
All kernel wake locks:
Kernel Wake lock WLAN AHB ISR: 15m 27s 109ms (5629 times) realtime
Kernel Wake lock PowerManagerService.WakeLocks: 15m 3s 175ms (808 times) realtime
Kernel Wake lock microarray_process_wakelock: 6m 55s 774ms (83 times) realtime
```

```
Line 125970:   All partial wake locks:
Line 125971:   Wake lock u0a83 *job*/com.ss.android.article.news/com.ss.android.message.PushJobService: 9m 37s 196ms (4 times) max=0 realtime
Line 125972:   Wake lock 1000 deviceidle_maint: 58s 797ms (5 times) max=0 realtime
Line 125973:   Wake lock u0a98 WakerLock:232772769: 51s 838ms (246 times) max=0 realtime
```

若相对你测试时间来说，这段持锁时间占比较多，就需要留意了。

2.从抓到的syslog中也可以找到每个wakelock申请与释放的时间，我们确认是哪个wakelock没有释放，了解原因再来确认如何修改。

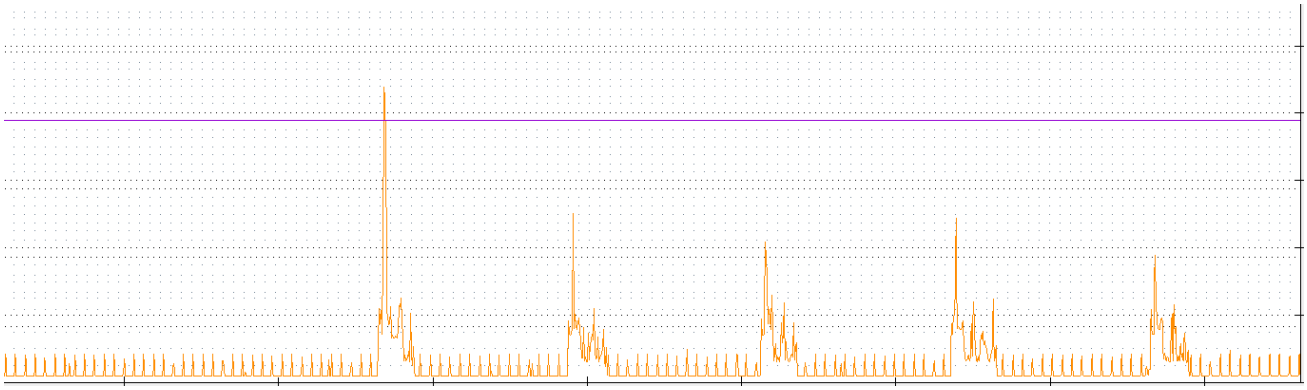
wakelock log的关键字如下：

PowerManagerService: acquireWakeLockInternal: lock=253958695, flags=0x1, tag="RILJ", ws=null, uid=1001, pid=1572

PowerManagerService: releaseWakeLockInternal: lock=253958695 [RILJ], flags=0x0, total_time=14ms

唤醒相关，这一块涉及较多，分析比较麻烦。

如果是系统唤醒之后，若没有wakelock存在会继续进入suspend状态，所以电流显示一般是波动一段时间后恢复正常。



分析时这个时我们一般要先找到唤醒源。

1.我们将kernel log文件用工具转换一下，使其内时间显示与其他log一致。

2.从log中我们需要关注三种log，suspend时间点，唤醒源，唤醒时间点。

```
<5>[  114.369568]<2>-(0)[91:kworker/u8:1][SLP] ##################### 进入suspend
<2>[  114.369568]<2>-(0)[91:kworker/u8:1][SLP] Chip_pm_enter
<5>[  114.369568]<2>-(0)[91:kworker/u8:1][SLP] #####################
<4>[  114.369568]<2>-(0)[91:kworker/u8:1][slp_check_pm_mtcmos_pll]
<5>[  114.369568]<2>-(0)[91:kworker/u8:1]vbat_val=3905, g_spm_timer=5400   G_spm_timer是battery meter预估此次sleep最长的时间
<6>[  114.369568]<2>-(0)[91:kworker/u8:1] mtk_wdt_mode_config  mode value=dc, tmp:220000dc,pid=91
<4>[  114.369568]<2>-(0)[91:kworker/u8:1][WDT] suspend
<2>[  114.369568]<2>-(0)[91:kworker/u8:1][SPM] md_settle = 99, settle = 99   md_settle是sleep的最后一句log
<2>[  114.369568]<2>-(0)[91:kworker/u8:1][SPM] sec = 5400, wakesrc = 0x380c564 (1)(1)
<2>[  114.369568]<2>-(0)[91:kworker/u8:1][SPM] wake up by EINT, timer_out = 5825074, r13 = 0x1c061020, debug_flag = 0x1f
<2>[  114.369568]<2>-(0)[91:kworker/u8:1][SPM] r12 = 0x20, raw_sta = 0x20, idle_sta = 0x93f, event_reg = 0x90100000, isr = 0x8
<2>[  114.369568]<2>-(0)[91:kworker/u8:1][SPM] suspend dormant state = 0
<5>[  114.369568]<2>-(0)[91:kworker/u8:1]EINT_STA:                          Wake up by XXX，代表唤醒源.
<5>[  114.369568]<2>-(0)[91:kworker/u8:1]EINT Module - index:0,EINT_STA =  timer_out = 5825074, 除以32768代表此时睡眠的时间, 就是177S.
<5>[  114.369568]<2>-(0)[91:kworker/u8:1]EINT 17 is pending
```

```
[begin]<2>[  113.476567]-0)[SPM] sec = 401, wakesrc = 0x1e84c5e6 (1)(1)
<2>[  113.476567]-0)[SPM] wake up by CLDMA_MD, timer_out = 1061, r13 = 0x340e133c, debug_flag = 0x110
<2>[  113.476567]-0)[SPM] r12 = 0x4000000, raw_sta = 0x4000100, idle_sta = 0x1ed, event_reg = 0x90100000, isr = 0x8
<2>[  113.476567]-0)[SPM] mempll mode = 2, suspend dormant state = 0, md32_flag = 0xbeef, md32_flag2 = 0
<5>[  113.491223] 1)[ccci1/mcd]CLDMA_MD wakeup source:(1/14)
<6>[  113.578597] (0)[55:cfinteractive][Power/cpufreq] _calc_new_opp_idx(): for early_suspend, idx = 4
<6>[  113.579095] (0)[95:kworker/u16:1]PM: suspend exit 2015-05-11 08:23:58.043500385 UTC
<6>[  114.608384] (1)[95:kworker/u16:1]PM: suspend entry 2015-05-11 08:23:59.072789462 UTC
<6>[  114.608394] (2)[95:kworker/u16:1]PM: Syncing filesystems ... done.
<6>[  114.651572] (2)[95:kworker/u16:1]PM: suspend exit 2015-05-11 08:23:59.115977000 UTC
<6>[  114.812796] (2)[95:kworker/u16:1]PM: suspend entry 2015-05-11 08:23:59.277200770 UTC
[end]<2>[  115.082620]-0)[SPM] md_settle = 99, settle = 99
```

从唤醒的第一句话，到sleep的最后一句话，【end】-【beging】= 唤醒持续的时间，115.08 – 113.47 = 1.61S.

一般我们常见的唤醒源如下：

EINT:

这种最常见的便是apk的alarm唤醒。

```
01-05 11:56:15.018955 <4>[  722.889389] -(0)[1044:system_server][SPM] md_settle = 99, settle = 99
01-05 11:56:15.018955 <4>[  722.889389] -(0)[1044:system_server][SPM] sec = 900, wakesrc = 0xe04c5e4 (1)(1)
01-05 11:56:17.742171 <4>[  722.889389] -(0)[1044:system_server][SPM] wake up byEINT, timer_out = 98245, r13 = 0x47238, debug_flag = 0x9f
01-05 11:56:17.742171 <4>[  722.889389] -(0)[1044:system_server][SPM] r12 = 0x20, raw_sta = 0x20, idle_sta = 0x9fe, event_reg = 0x90100000, isr = 0x0
01-05 11:56:17.742171 <4>[  722.889389] -(0)[1044:system_server][SPM] suspend dormant state = 0, md32_flag = 0x0, md32_flag2 = 0
01-05 11:56:17.742171 <4>[  722.889389] -(0)[1044:system_server][SPM] log_wakesta_index = 16
01-05 11:56:17.742171 <5>[  722.889389] -(0)[1044:system server]EINT STA:
```

可在log中查找关键字 wakeup alarm

```
01-05 11:56:18.044955  1015  1073 D AlarmManager: wakeup alarm = Alarm{3830d3b type 2 when 831161 com.android.providers.calendar); package = com.android.providers.calendarneedGroupi
01-05 11:56:18.045190  1015  1073 V ActivityManager: Broadcast: Intent { act=com.android.providers.calendar.intent.CalendarProvider2 flg=0x114 (has extras) } ordered=true userid=0 c
```

conn2AP:

这种唤醒，最常见的就是wifi的，一般路由有数据要给到手机时会出现，kernellog中一般这样：

wake up byCONN2AP, timer_out = 18968, r13 = 0x4f000, debug_flag = 0x9f

```
01-05 11:53:41.974709 <4>[  676.422586] -(0)[1044:system_server][SPM] sec = 900, wakesrc = 0xe04c5e4 (1)(1)
01-05 11:53:42.744759 <4>[  676.422586] -(0)[1044:system_server][SPM] wake up byCONN2AP, timer_out = 18968, r13 = 0x4f000, debug_flag = 0x9f
01-05 11:53:42.744759 <4>[  676.422586] -(0)[1044:system_server][SPM] r12 = 0x400, raw_sta = 0x400, idle_sta = 0x9fe, event_reg = 0x90100000, isr = 0x0
01-05 11:53:42.744759 <4>[  676.422586] -(0)[1044:system_server][SPM] suspend dormant state = 0, md32_flag = 0x0, md32_flag2 = 0
01-05 11:53:42.744759 <4>[  676.422586] -(0)[1044:system_server][SPM] log_wakesta_index = 4
01-05 11:53:42.744759 <7>[  676.422586] -(0)[1044:system server][name:mtk wdt&] mtk wdt mode config  mode value=5d, tmp:2200005d,pid=1044
```

然后以下会数据包的信息。

[1603:tx_thread][name:wlan_gen2&]nicRxCheckWakeupReason:(RX INFO)IP Packet from:123.180.248.253, IP ID 0x3c7b wakeup host

我们可以根据ip地址来找到对应app再确认是否正常。

modem:

modem唤醒源一般如wake up by CLDMA_MD这种，同时下方会有

[md1]CLDMA_MD wakeup source:(1/10)这种信息。

10代表channel，以下为常见channel。

Channel ID	Channel name	User	进一步 debug 的手段
6/8	CCCI_UART1	Modem log	关闭 modem log
42/43	CCCI_MD_LOG		
32/33	CCCI_RPC	Modem 使用 AP 资源（比如 SIM 卡中断）	检查 **DWS** 配置、**硬件**中断是否异常
14/15	CCCI_FS	Modem 读写 nvram	需要 protocol 分析 **modem log**
20/22/24/26/28/30	CCCI_CCMNI	有数据传输	**Netlog** 即可
34/36	CCCI_IPC	4G modem 跟 WCN 有频段是重叠的，因此必要的时候要两边同步	需要 protocol 分析 **modem log**
10/12	CCCI_UART2	MUXD，通常都是 AT cmd	**Radio log** 基本可以定位问题，有必要的话再配合 modem log

一般很常见的便是10，20，如果很频繁的话便需要分析

10代表modem有变化，通过at command唤醒系统，这种需要结合radiolog，看给AP传递什么指令，然后根据指令再去online上搜索。

20的的话，代表有数据进来，需要唤醒系统，需要结合mainlog或networklog来分析是哪个app引起的。

- 请在main log搜关键字Posix_connect Debug
- 05-25 10:27:24.077 30686 1105 D Posix : [Posix_connect Debug]Process com.tencent.mobileqq:MSF :80
- 05-25 10:27:43.706 30686 1105 D Posix : [Posix_connect Debug]Process com.tencent.mobileqq:MSF :80
- 05-25 10:32:08.410 26307 26342 D Posix : [Posix_connect Debug]Process com.netease.cloudmusic :80
- 05-25 10:37:25.997 24769 1199 D Posix : [Posix_connect Debug]Process com.wumii.android.mimi :1878
- 05-25 10:37:54.736 26447 26490 D Posix : [Posix_connect Debug]Process com.tencent.mm :80
- 05-25 10:37:54.736 26447 26489 D Posix : [Posix_connect Debug]Process com.tencent.mm :80
- 05-25 10:37:54.744 26447 26488 D Posix : [Posix_connect Debug]Process com.tencent.mm :80

CLDMA是modem跟AP沟通的桥梁

- 网络的唤醒，底层一般会有：
- <2>[2758.615588]-0)[SPM] wake up by CLDMA_MD, timer_out = 291562, r13 = 0x1406133c, debug_flag = 0x1803011f
- CLDMA_MD wakeup source:(3/20)

4.漏电或待机后某些器件没有关闭

这种的情况下电流图的信息一般是底电流保持在一个差不多固定的值。

基本都是硬件与驱动的同事跟进，基本不会流转到我们这来，但也有可能我们打开了某些器件而没有关闭，这个他们也会找我们排查。

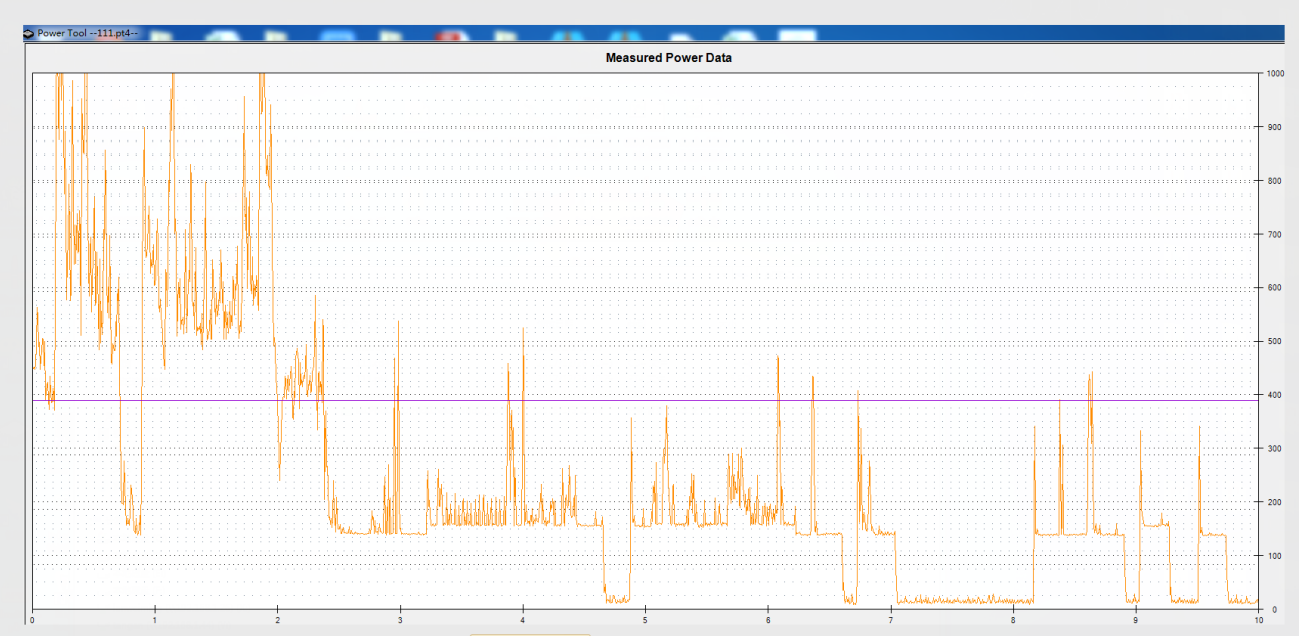
5.系统没有唤醒，但电流有突然起来。

这种情况意味这系统一直处于suspend状态，可能是modem或wifi自己起来做事，并没有唤醒系统，这种情况我们需要先确认是wifi还是modem问题，然后使用power monitor，对好时间抓取电流图与log提eservice给mtk协助处理。

四、案例分析

1.外单因未开alarm拦截导致回退电流问题

电流图如下：



从电流图中可以大概看出，系统在4min到5min之间有休眠下去，休眠下去之后又被频繁地唤醒。

从log中可以看出。

```
| -(0)[0:swapper/0][ccci1/cldma]wake up by CLDMA_MD L2(0/1)(f000/18)!
| -(0)[1173:system_server][SLP] @@@@@@@@@@@@@@@@@@@@@@  Chip_pm_enter   @@@@@@@@@@@@@@@@@@@@@@
| -(0)[1173:system_server][SPM] wake up by  R12_CCIF0_EVENT_B, timer_out = 401728, r13 = 0x4600112c, debug_flag =
| -(2)[0:swapper/2][ccci1/cldma]wake up by CLDMA_MD L2(0/1)(f000/18)!
| -(0)[1173:system_server][SLP] @@@@@@@@@@@@@@@@@@@@@@  Chip_pm_enter   @@@@@@@@@@@@@@@@@@@@@@
| -(0)[1173:system_server][SPM] wake up by  R12_CCIF0_EVENT_B, timer_out = 2315490, r13 = 0x4604112c, debug_flag =
| -(3)[0:swapper/3][ccci1/cldma]wake up by CLDMA_MD L2(0/1)(f000/18)!
| -(0)[1173:system_server][SLP] @@@@@@@@@@@@@@@@@@@@@@  Chip_pm_enter   @@@@@@@@@@@@@@@@@@@@@@
| -(0)[1173:system_server][SPM] wake up by  R12_CCIF0_EVENT_B, timer_out = 459125, r13 = 0x4604112c, debug_flag =
| -(0)[9852:Thread-23][ccci1/cldma]wake up by CLDMA_MD L2(11/0)(f000/18)!
| -(0)[1173:system_server][SLP] @@@@@@@@@@@@@@@@@@@@@@  Chip_pm_enter   @@@@@@@@@@@@@@@@@@@@@@
| -(0)[1173:system_server][SPM] wake up by  R12_CCIF0_EVENT_B, timer_out = 406326, r13 = 0x4400112c, debug_flag =
| -(0)[1173:system_server][SLP] @@@@@@@@@@@@@@@@@@@@@@  Chip_pm_enter   @@@@@@@@@@@@@@@@@@@@@@
| -(0)[1173:system_server][SPM] wake up by  R12_CCIF0_EVENT_B, timer_out = 220513, r13 = 0x4604112c, debug_flag =
| -(1)[0:swapper/1][ccci1/cldma]wake up by CLDMA_MD L2(0/1)(f000/18)!
| -(0)[1173:system_server][SLP] @@@@@@@@@@@@@@@@@@@@@@  Chip_pm_enter   @@@@@@@@@@@@@@@@@@@@@@
| -(0)[1173:system_server][SPM] wake up by  R12_CCIF0_EVENT_B, timer_out = 357196, r13 = 0x4600112c, debug_flag =
| -(0)[1173:system_server][SLP] @@@@@@@@@@@@@@@@@@@@@@  Chip_pm_enter   @@@@@@@@@@@@@@@@@@@@@@
| -(0)[1173:system_server][SPM] wake up by  R12_CCIF0_EVENT_B, timer_out = 2200389, r13 = 0x4400112c, debug_flag =
```

可以看到系统休眠下后，是被唤醒源R12_CCIFO_EVENT_B所唤醒，由于各个平台的唤醒源的描述不一样，这个对应的就是之前的EINT。

然后对应syslog查找wakeup alarm.

```
2010_0101_114541__1\sys_log_4__2017_1228_204648 (25 hits)
: wakeup alarm = Alarm{d492ba6 type 2 when 12191126 com.google.android.gms}; package = com.google.
r: wakeup alarm = Alarm{d71f0d0 type 0 when 1262318692898 com.android.settings}; package = com.and
r: wakeup alarm = Alarm{afc2aef type 0 when 1262320698646 com.google.android.googlequicksearchbox}
r: wakeup alarm = Alarm{64d49fc type 0 when 1262348587760 com.android.providers.media}; package =
r: wakeup alarm = Alarm{5b7885 type 0 when 1262368706041 com.android.vending}; package = com.andro
r: wakeup alarm = Alarm{e0a870b type 0 when 1262384678742 com.android.providers.calendar}; package
r: wakeup alarm = Alarm{d2309e8 type 0 when 1264897393419 com.google.android.gms}; package = com.g
r: wakeup alarm = Alarm{6601801 type 2 when 12221190 com.google.android.gms}; package = com.google
r: wakeup alarm = Alarm{ee9692d type 2 when 12228121 com.google.android.gms}; package = com.google
r: wakeup alarm = Alarm{d57564b type 0 when 1514464302920 com.android.settings}; package = com.and
r: wakeup alarm = Alarm{bc23ee6 type 0 when 1514464304529 com.android.vending}; package = com.andr
r: wakeup alarm = Alarm{1850504 type 2 when 12232429 com.android.phone}; package = com.android.pho
r: wakeup alarm = Alarm{1ecdfa3 type 0 when 1514464319449 com.android.vending}; package = com.andr
r: wakeup alarm = Alarm{c9bb3f5 type 2 when 12265795 android}; listener package = DeviceIdleContro
r: wakeup alarm = Alarm{39e758a type 2 when 12288915 com.google.android.gms}; package = com.google
r: wakeup alarm = Alarm{1c955c4 type 2 when 12327416 com.google.android.gms}; package = com.google
r: wakeup alarm = Alarm{87553a type 2 when 12596090 com.android.phone}; package = com.android.phon
```

可以看到大部分都是gms包里面的，但我们加的alarm 拦截唤醒是会把唤醒类型的alarm转化为非唤醒类型的，可以确认是功能没有打开。

五、已用优化措施

我们目前所用的优化都是基于待机方面的优化。

内单：

1.alarm唤醒拦截

2.待机网络管控

3.jobscheduler与sync优化处理

4.纯净后台

外单：

1.alarm 唤醒拦截

2.待机gms wakelock管控

3.doze

4.jobscheduler与sync

5.必要alarm的间隔，如datashaping、销量统计。