Tina Linux 蓝牙
常见问题与调试指南

版本号: 1.4
发布日期: 2024.8.27

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.01.20	AWA1427	初始版本。
1.1	2022.06.06	AWA1427	全文更新文档格式。
1.2	2023.11.02	AWA1887	修复文档里错误的内容描述与格式。
1.3	2023.11.27	AWA1887	增加文档适用 buildroot 构建方式。
1.4	2024.8.27	KPA0568	增加 NF3205q_RF 说明。

目　录

插　图

1 前言

1.1 文档简介

本文档介绍 Tina Linux 平台蓝牙常见问题排查手段，方便用户遇到蓝牙问题能通过文档获取排查思路。

1.2 目标读者

Tina Linux 开发用户、蓝牙开发使用人员。

1.3 适用范围

Tina Linux 平台。

1.4 文档约定

1.4.1 标志说明

⚠ 注意

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-1: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示 (寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

2 通用调试指南

2.1 HCI Log

HCI Log 用来分析蓝牙设备之间的交互行为是否符合预期，是否符合蓝牙规范。可以通过 BlueZ 协议栈附带的 hcidump 工具获取，命令如下：

```
hcidump -i hci0 -w /tmp/bt_snoop.cfa &
```

- -i：指定的 hci 设备节点，一般为 hci0，可以通过 hciconfig -a 命令查看设备信息；
- -w：保存到文件，文件路径与名字自定义，后缀名建议使用 cfa。

抓取结束，把 hcidump 进程 kill 掉即可，再通过 adb pull 将日志拉到本地，在 PC 端使用 Front-Line 软件打开查看。Frontline ComProbe Protocol Analysis System 是 Frontline 提供的一款蓝牙协议日志分析工具，用户可以通过互联网学习使用。如果分析问题需要完整的 HCI Log，需要从蓝牙启动就开始抓，需要修改蓝牙的启动脚本，小机端路径：/etc/bluetooth/bt_init.sh。

```
hci_start()
{
    h=`ps | grep "$bt_hciattach" | grep -v grep`
    if [ -n "$h" ] ;then
        echo "Bluetooth init has been completed!!"
    else
        start_hci_attach
    fi
    hciconfig hci0 up
    hcidump -i hci0 -w /tmp/bt_snoop.cfa &
    if [ $? == 0 ] ; then
        return 0
    else
        return 1
    fi
}
```

2.2 Firmware

介绍常见模组蓝牙 firmware 如何更新，以及 log 抓取。

2.2.1 XR 系列模组

XR 系列模组包括 XR829 和 XR819S，其 firmware 根据不同晶振有 24M 和 40M 两种区分，更新时务必确认当前设备使用类型，否则蓝牙功能异常。

2.2.1.1 获取资料

- 内部同事：在内部 Git 仓库下载获取；
- 外部客户：通过全志客户服务平台获取。

2.2.1.2 小机替换

使用 adb push firmware 文件到设备端，例如替换 XR829 40M 的蓝牙 firmware：

```
adb push fw_xr829_bt_40M.bin /lib/firmware/fw_xr829_bt.bin
```

2.2.1.3 SDK 更新

更新到具体模组目录下，文件名字有 40M 代表是 40M 的 FW，路径参考：

```
Tina4.0:
tina/package/firmware/linux-firmware/xxx
Tina5.0:
tina/platform/allwinner/wireless/firmware/xxx
```

2.2.1.4 firmware 日志抓取

调试蓝牙问题时，定位问题与 Host 端没有关系，并且怀疑与 Controller 端有关系，需要抓 firmware 日志进一步排查；

XR 系列模组支持两种方式获取蓝牙 firmware log。

（1）芯片专用引脚

a.XR829 芯片为 18 号引脚（标注一般是 CLKREQ)，XR819S 芯片是 36 号引脚（标注一般是 UART0_TX）；

b. 把特殊引脚用线引出来，同时也要引出板子上的任意 GND；

c. 特定引脚线接到 USB 转 TTL 串口工具的 RX 上，GND 线接到工具的 GND 实现共地；

d. 打开电脑端串口工具软件，设置波特率为 115200，打开蓝牙后就可以看到 firmware log 了。

（2）fwlog 工具

在 Tina 平台上可以通过 fwlog 程序抓取 XR829 的 firmware log，其原理是通过监听特殊的 HCI log 来实现；通过该工具只能抓取特定的 firmware log，这个需要 firmware 端配合使用，一般是方便调试。因此不合适所有的场景。

fwlog 工具的使用方法：

```
adb shell fwlog -R > E:/xxx/fwlog.txt
```

2.2.2 Realtek 系列模组

2.2.2.1 获取资料

联系模组厂获取最新资料包，一般包含 firmware bin 和 firmware config 文件。

2.2.2.2 小机替换

使用 adb push firmware 文件到设备端，例如替换 RTL8723ds 的 firmware：

```
adb push rtl8723d_fw /lib/firmware/rtlbt
adb push rtl8723d_config /lib/firmware/rtlbt
```

2.2.2.3 SDK 更新

更新到具体模组目录下，参考路径：

```
Tina4.0:
tina/package/firmware/linux-firmware/xxx
Tina5.0:
tina/platform/allwinner/wireless/firmware/xxx
```

2.2.2.4 firmware 日志抓取

请联系模组厂提供支持。

2.3 Hciattach

介绍 Hciattach 源码更新方式。Hciattach 是对蓝牙模组进行初始化的关键程序，UART 初始化、firmware 下载等步骤都在其中进行。

2.3.1 XR 模组

XR 模组的 Hciattach 使用 BlueZ 协议栈自带的，加入了符合 XR 模组初始化的修改。一般用户无需修改，了解下即可。关键的源码路径：

```
tina/out/.../target/bluez-5.54/tools/hciattach_xradio.c
```

对 Bluez package 修改的 patch 路径：

```
Tina4.0:
tina/package/utils/bluez/patches
Tina5.0 openwrt:
tina/openwrt/package/feeds/utils/bluez/patches
Tina5.0 buildroot:
tina/buildroot/buildroot-xxx/package/bluez5_utils/
```

2.3.2 Realtek 模组

Realtek 使用专有的 hciattach，在 Tina 平台上命名为 rtk_hciattach，源码路径：

```
Tina4.0:
tina/package/utils/rtk_hciattach
Tina5.0 openwrt:
tina/openwrt/package/feeds/utils/rtk_hciattach
```

此软件包由 Realtek 提供，如果需要更新直接替换即可。

2.3.3 其它模组

参考 Realtek 模组的 hciattach 的源码路径，如需更新直接替换模组厂提供的更新包。

2.4 BlueZ 协议栈

2.4.1 patch 更新

协议栈问题修改都是通过 patch 的形式，把对应的 patch 添加到 tina/…/utils/bluez/patches 路径即可。

在此目录下通过 mm 命令可以编译生成新的 bin 文件。

2.4.2 设备缓存

BlueZ 协议栈会对扫描到、已配对的设备信息进行保持。

2.4.2.1 缓存路径介绍

小机端默认路径：

```
/etc/lib/bluetooth
```

如下图所示：

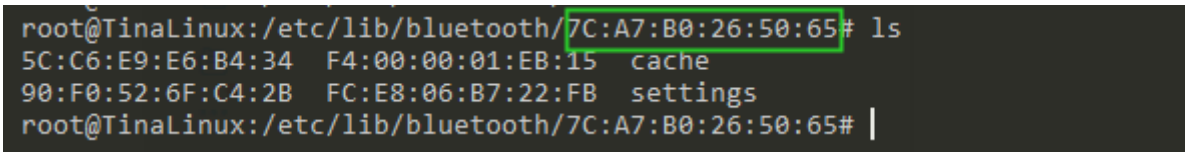


图 2-1: 缓存路径

圈出的文件夹 7C:A7:B0:26:50:65 是本设备 mac 地址，在该目录下看到其它以 mac 地址命名的文件，这些表示已配对的设备。cache 文件夹保存扫描缓存。在已配对的设备文件夹内部，都会有一个 info 文件，里面保存了设备信息，如 LinkKey 等，如下图：

图 2-2: 缓存信息

2.4.2.2 如何修改缓存路径

BlueZ 支持修改 Makefile 更改缓存路径，Makefile 路径…/utils/bluez/Makefile。但是注意修改的路径必须是可读可写的。例如修改到 data 分区：

把 Makefile 里面的 –localstatedir=/etc 修改为 –localstatedir=/data。

修改后小机端的缓存路径变为/data/lib/bluetooth。

2.4.2.3 修改打印等级

把协议栈的打印可以修改设备中的 bt_init.sh 文件完成。

bt_init.sh 在小机端的路径：/etc/bluetooth/bt_init.sh。

修改 bluetoothd 程序的启动参数，加上-d，如下：

```
bluetoothd -n -d &
# /etc/bluetooth/bluetoothd start
```

2.5 Bluealsa

升级 btmanager 时可能也需要对 bluealsa 进行更新，因此主要介绍 bluealsa 的更新。

2.5.1 小机替换

编译出的 bluealsa bin 文件路径：

```
tina/out/.../target/bluez-alsa-xxxxx/src
```

通过 adb push 到/usr/bin/bluealsa 路径，操作之后建议重启机器再验证。

2.5.2 SDK 更新

相关路径：

```
Tina4.0:
tina/package/multimedia/bluez-alsa
tina/package/multimedia/bluez-alsa-3.1
Tina5.0 openwrt:
tina/openwrt/package/thirdparty/multimedia/bluez-alsa
Tina5.0 buildroot:
tina/buildroot/buildroot-xxx/package/bluez-alsa/
```

大版本更新直接替换 bluez-alsa 目录，再根据 bluez-alsa 目录的 Makefile 判断当前的版本：

```
PKG_NAME:=bluez-alsa
PKG_VERSION:=20211122
PKG_RELEASE:=1
PKG_MAINTAINER:=
PKG_LICENSE:=MIT
PKG_LICENSE_FILES:=LICENSE.txt

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_BUILD_DIR := $(COMPILE_DIR)/$(PKG_NAME)-20211122
```

可以看出版本信息为 20211122，源码包的路径为

```
PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz=bluez-alsa-20211122.tar.gz
```

再把相应的压缩包放到 dl 目录即可。

如果不是大版本更新，修复问题都是通过 patch 的形式，把对应的 patch 添加到 multimedia/bluez-alsa/patches 路径即可，在此目录下通过 mm 命令可以编译生成新的 bin 文件。

2.5.3 修改打印等级

可以通过修改 makefile 调整：multimedia/bluez-alsa/Makefile。

增加如下配置, 注意需要重新编译。

```
CONFIGURE_ARGS += --enable-debugtina/package/multimedia/bluez-alsa
```

2.6 btmanager

2.6.1 小机替换

（1）bt_test

bt_test 是 demo 可执行程序，路径：

```
/usr/bin/bt_test
```

（2）libbtmg.so

libbtmg.so 是 btmanager 编译生成的动态库，路径：

```
/lib/libbtmg.so
```

2.6.2 SDK 更新

软件包替换路径:

```
Tina4.0:
tina/package/allwinner/wireless/btmanager4.0
Tina5.0:
tina/platform/allwinner/wireless/btmanager
```

目录下可以直接通过 mm -B 编译, 编译生成的 bt_test 程序和 so 路径如下：

（1）bt_test

```
tina/out/.../target/btmanager-v4.0/demo
```

（2）libbtmg.so

```
tina/out/.../target/btmanager-v4.0/src
```

2.6.3 打印日志动态调整

btmanager 支持在使用过程中动态调整打印配置，方便用户 Debug。

2.6.3.1 调整打印输出方向

btmanager 有多种打印输出设置，分别通过如下命令：

```
echo type=0 > /tmp/log_btmg_io  关闭打印，不打印
echo type=1 > /tmp/log_btmg_io  打印到屏幕
echo type=4 > /tmp/log_btmg_io  保持到文件，文件默认路径：/tmp/log_hub.txt
echo type=5 > /tmp/log_btmg_io  打印到屏幕，且保存在文件中。
```

2.6.3.2 调整打印等级

前面调整打印等级方向不包含打印登记，打印等级调整通过如下命令：

```
echo level = x > /tmp/log_btmg_io
x表示打印等级：
例如：echo level = 4 > /tmp/log_btmg_io
```

2.7 Driver

在开发调试过程中，可能需要对蓝牙驱动进行更新验证。

2.7.1 XR 模组

根据当前的内核版本获取到对应的驱动资料包，然后使用资料包中的驱动文件直接替换，一般是 xradio_btlpm.c、hci_ldisc.c 等文件，一般很少需要更新。驱动代码路径：

```
tina/.../linux-xx/drivers/bluetooth
```

2.7.2 Realtek 模组

Realtek 的驱动可能偶尔需要更新。模组资料包里面也提供移植指南，但很多用户不清楚在 Tina 平台上怎么移植更新，所以在此根据现有的驱动资料包进行说明，这里只介绍 UART 接口模组的驱动更新。更新之前要确认资料包的驱动可以适配当前的内核版本。模组厂提供的资料包中有 bluetooth_uart_driver 这个文件夹，内容如下：

```
├── hci_h4.c
├── hci_ldisc.c
├── hci_rtk_h5.c
├── hci_uart.h
├── Kconfig
├── Makefile
├── rtk_coex.c
└── rtk_coex.h
```

可以直接替换 hci_h4.c、hci_ldisc.c、hci_rtk_h5.c、hci_uart.h 文件，如果有编译错误，请具体问题具体分析。

rtk_coex.c 和 rtk_coex.h 是给 USB 接口模组使用的，可以不用更新。Kconfig 和 Makefile 不能直接替换，关键信息如下：

```
Makefile：
  hci_uart-$(CONFIG_BT_HCIUART_RTL3WIRE) += hci_rtk_h5.o


Kconfig：
  config BT_HCIUART_RTL3WIRE
  bool "Realtek Three-wire UART (H5) protocol support"
  depends on BT_HCIUART
  help
   Realtek Three-wire UART (H5) transport layer makes it possible
   to use Realtek Bluetooth controller with Three-wire UART.
   Say Y here to compile support for Realtek Three-wire UART.
```

2.8 Tools

2.8.1 RF 测试工具

RF 测试需要专门的软件测试工具，不同的模组厂的工具都不一样。因此简单介绍常用模组厂的蓝牙测试工具路径。

2.8.1.1 XR 模组

RF 测试工具是 btetf。

（1）小机端路径

```
/usr/bin/btetf
```

（2）SDK 源码路径

```
Tina4.0:
tina/package/utils/rftest/xxx
Tina5.0 openwrt:
tina/openwrt/package/feeds/utils/rftest/xxx
```

2.8.1.2 NF 模组

RF 测试工具是 myftm，此工具的运行依赖专门 wifi 驱动加载 BTfw。以 nf3205q 为例，相关的文件如下：

```
insmod wlan.ko con_mode=5
```

（1）小机端路径

```
/usr/bin/myftm
```

（2）SDK 源码路径

```
Tina5.0 buildroot:
longan/bsp/drivers/net/wireless/nforetek/nf3205q/fw-api/hw/
```

2.8.1.3 Realtek 模组

RF 测试工具是 rtlbtmp，此工具的运行还依赖专门的 mp fw 和 mp fw_config。以 RTL8723ds 为例，相关的文件如下：

```
rtlbtmp
mp_rtl8723d_fw
mp_rtl8723d_config
```

（1）小机端路径

```
/usr/bin/rtlbtmp
```

（2）SDK 源码路径

```
Tina4.0:
tina/package/utils/rftest/xxx
Tina5.0 openwrt:
tina/openwrt/package/feeds/utils/rftest/xxx
```

3 通用排查指南

蓝牙问题可能会有各种各样的现象，有的问题可能从 Log 中并不能容易定位到问题点，就需要对问题进行排查定位问题点。常见的问题排查建议可以通过以下的几种方法，先初步定位问题点。

3.1 硬件排查

遇到以下类似问题，可以先测试验证硬件射频指标以及天线是否正常，这个过程需要硬件同事协助：

（1）无法扫描到设备或很难扫描到设备；

（2）部分设备无法扫描到；

（3）无法被周围设备扫描到；

（4）无法连接或很难连接成功；

（5）连接频繁断开；

（6）A2DP 蓝牙音乐频繁卡顿；

（7）BLE 数据收发异常，丢包。

3.2 检查软件版本

以上《硬件排查》中提到的问题，同样可以对软件版本进行排查，但是建议先保证硬件是正常的再对软件进行排查。

检查软件版本主要是对新旧版本软件进行测试对比，验证是否是新版本软件引入的问题。

软件版本排查建议按照如下顺序来：

（1）蓝牙 firmware 版本；

（2）btmanager 版本；

（3）蓝牙驱动版本；

（4）bluealsa 版本（A2DP 蓝牙音乐相关）；

（5）上层应用的版本。

排查过程是灵活的，请根据实际的情况按需排查。

3.3 兼容性排查

出现设备连接失败，蓝牙音乐播放卡顿等现象，也有可能是以下的因素导致：

（1）本设备蓝牙协议栈的参数与对端设备不匹配；

（2）蓝牙模组与对端设备存在兼容性问题；

（3）对端设备某些行为不符合协议规范。

尝试使用其它同类型的设备进行验证确认，确认是否与设备兼容性有关。

3.4 Wi-Fi/BT 共存

Wi-Fi 和 BT 一般会共同使用射频模块，在检查硬件指标正常后，还出现如难连接、丢包严重、蓝牙音乐播放卡顿等问题，先把 Wi-Fi 的干扰影响排除，单独验证 BT Only 的情况。建议直接把 Wi-Fi 模组的驱动卸载后去复现问题，如果问题有改善，可以判断与 Wi-Fi/BT 共存有关，这种情况需要联系模组厂提供支持，因为模组厂才能调整模组的共存测量。

3.5 环境干扰

检查了硬件指标后，遇到难连接、丢包严重、蓝牙音乐播放卡顿等问题，也有可能是测试环境太恶劣，严重不符合日常应用的场景。如果有条件可以使用其它同类型产品验证在此环境下是否工作正常。使用手机扫描周围的 Wi-Fi 路由器、蓝牙设备的数量做下参考，找一个干净的环境复现排查问题。

3.6 对比 Demo

有的问题是在用户应用层集成阶段引入的，一般有以下的情况：

（1）API 使用错误；

（2）参数设置错误；

（3）应用处理、逻辑异常。

建议用户出现问题后，如果 btmanager 的示例 demo 有相关的功能，可以用 demo 进行验证对比。

4 经典蓝牙-BT

4.1 基础功能

基础功能介绍与具体 Profile 无关的案例，例如蓝牙初始化、功能增减等。

4.1.1 蓝牙初始化失败

蓝牙启动分为四个阶段：hciattach 初始化阶段、hci0 up 阶段、协议栈启动阶段、btmanger 启动阶段;。

4.1.1.1 hciattach 初始化阶段

hciattach 阶段主要是对模组上电、复位、下载 fw、配置 UART 等。

可以执行命令运行一次，确认是否有异常 log，固件是否能够下载成功。

A. 命令上电复位

```
echo 0 > /sys/class/rfkill/rfkill0/state;
echo 1 > /sys/class/rfkill/rfkill0/state;
```

B. 命令启动 hciattach

（1）xradio 模组

```
hciattach -n ttyS1 xradio &
```

（2）realtek 模组

```
rtk_hciattach -n -s 115200 /dev/ttyS1 rtk_h5 &
```

（3）其他模组

请参考相应的 bt_init.sh 文件；

进行以上步骤之后，再执行 hciconfig -a 能看到 hci0 节点，说明 hciattach 已经执行成功。

如果失败了 hciattach 通常有以下几个原因：

（1）硬件工作条件没有满足；

（2）hciattach 没有移植正确；

（3）firmware 固件不匹配；

（4）hci uart 驱动未使能。

4.1.1.2 hci0 up 阶段

hciattach 启动成功后，执行 hciconfig hci0 up 命令，hci0 如果无法 up 成功，通常情况下可能是 hci0 blocked 了，使用 rfkill list 命令检查 hci0 是否 blocked：

```
root@TinaLinux:/# rfkill list
2: hci0: bluetooth
    Soft blocked: no
    Hard blocked: no
```

以上显示为 no blocked，如果 blocked 的状态为 yes，表示 blocked 了，一般是内核的配置选择了错误，请检查：

```
root@TinaLinux:/# rfkill list
2: hci0: bluetooth
    Soft blocked: no
    Hard blocked: no
```

4.1.1.3 协议栈初始化阶段

BlueZ 协议栈的可执行程序是 bluetoothd，一般在启动完成 hciattach，hci0 up 之后，就可以启动协议栈了。bluetoothd 依赖于 dbus，所以通常情况下如果启动 bluetoothd 失败，大多数都是 dbus 没有启动成功。可以使用使用 ps 命令检查 dbus-daemon 进程：

```
root@TinaLinux:/# ps | grep dbus
 925 root     1316 S    /usr/sbin/dbus-daemon --system
1721 root     1064 S    grep dbus
```

如果后台没有该进程，则说明 dbus-daemon 没有启动成功，需要进一步检查；另外需注意，默认情况下蓝牙启动和协议栈启动都是在启动脚本/etc/bluetooth/bt_init.sh 中完成的，在启动该脚本默认就认为 dbus 已经启动，所以有时 bluetoothd 启动异常，可能是 dbus 正在启动还未完成，bluetoothd 就开始启动了，存在时序问题。

4.1.1.4 btmanager 启动阶段

前面的 3 个阶段都是在调用 btmanager 的 bt_manager_enable API 后开始初始化的，btmanager 阶段出现的问题可能是状态异常、出现程序异常退出等问题，同时确保小机端有 bluetooth.json 文件，文件路径：

```
/etc/bluetooth/bluetooth.json
```

运行崩溃的情况，需要通过堆栈信息来分析问题，开启 coredump 可以把关键的堆栈信息获取到，配置方法如下：

（1）勾选编译配置

make menuconfig

```
Global build settings —>
 [*] Enable process core dump support
 Binary stripping method (none) —>

Development —>
 <*> gdb··· GNU Debugger
```

选上之后需要重新编译固件。

（2）小机端配置

在小机端分别输入以下命令：

```
ulimit -c unlimited
echo /tmp/core > /proc/sys/kernel/core_pattern
```

使能 coredump 之后，程序运行崩溃，会将 coredump 信息存储在/tmp/core 目录下, 执行 gdb bt_test /tmp/core。

这里 bt_test 是应用的名称，请根据实际情况进行更换。输入之后会进入二级 shell，再输入 bt 命令就可以看到堆栈信息。

4.1.1.5 案例 1：HCI Device 初始化失败

问题描述：

HCI Device 初始化失败，hci0 设备节点无法生成。

问题背景：

主控：R328

模组：XR829

问题表现：

在执行 bt_test 或者应用程序之后，如果看到打开失败的打印，并且使用 hciconfig -a 命令依旧无法看到蓝牙接口信息，或者执行 hciconfig hci0 up 无法。常见的异常日志如下：

```
1970-01-16 03:13:12:000: BTMG[_bt_manager_enable:225]:  enable state: 1, now bt adapter state : 0
1970-01-16 03:13:12:001: BTMG[bt_test_adapter_status_cb:66]:  bt is turnning on.
bring up hci0 failed
1970-01-16 03:13:23:108: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:26:108: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:29:109: BTMG[bt_platform_init:31]:  detect hci0.....
```

```
1970-01-16 03:13:32:109: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:35:109: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:38:110: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:41:110: BTMG[bt_platform_init:36]:  init hci device failed
1970-01-16 03:13:41:110: BTMG[_bt_manager_enable:246]:  BT turn on fail
1970-01-16 03:13:41:110: BTMG[bt_test_adapter_status_cb:49]:  BT is off
1970-01-16 03:13:41:112: BTMG[pfd1_thread_process:946]:  enter
```

问题分析：

1. 出现 hci0 初始化失败的打印，怀疑是在 hciattach 阶段出现异常。
2. 通过命令再次确认问题。

将设备重启后：

a. 命令上电复位

echo 0 > /sys/class/rfkill/rfkill0/state;

echo 1 > /sys/class/rfkill/rfkill0/state;

b. 命令启动 hciattach

hciattach -n ttyS1 xradio &

执行上面命令之后，如果没有看到 Device setup complete 的打印，而是看到 uart sync 的报错，并且最终是 Initialization timed out。

3. 怀疑是硬件工作条件没有满足

序号	检查点	检查思路
1	检查两路电源供电,VCC/VCCIO	万用表测量
2	检查 BT-RESETN 是否拉高，符合时序要求	万用表测量
3	检查 AP-WAKE-BT 电平	万用表测量
4	检查 UART 功能是否正常	单独测试自发自收，保证 uart 节点通信正常
5	检查主时钟晶振是否正常	示波器测量
6	检查次时钟晶振是否正常（可选）	示波器测量
7	检查固件下载是否正常	检查文件路径，启动时序，uart 配置（波特率，流控等）

参考《Tina_Linux_ 蓝牙模组移植指南》，排查硬件的工作条件，关键排查点：

4. 在检查中发现 AP-WAKE-BT 电平为低，没有被拉高，所以蓝牙模组没处于正常工作状态。
5. AP-WAKE-BT 引脚的电平一般是在 hciattach 初始化的过程中，通过 LPM 驱动的设备节点”/proc/bluetooth/sleep/btwake” 进行设置的。
6. 检查小机端发现没有”/proc/bluetooth/sleep/btwake” 这个节点，确认 LPM 驱动没有正常加载。
7. 检查了内核配置，发现 LPM 驱动没有配置，配上之后蓝牙可以正常打开。

根本原因：

在模组移植过程中没有将蓝牙 LPM 驱动配置，没有”/proc/bluetooth/sleep/btwake” 这个节点，从而导致 hciattach 初始化过程中无法将 AP-WAKE-BT GPIO 拉高。

解决办法：

正确配置 LPM 驱动

思路总结：

- 确认问题点；
- 检查硬件是否正常；
- 检查软件配置是否正常。

4.1.1.6 案例 2：btmanager enable 失败

问题描述：

出现 Dbus 的错误打印信息。

问题背景：

平台: V833

模组：XR829

问题表现：

打开蓝牙出现低概率失败，关键的 Log 如下:

```
[14:37:07:930](process:915): GLib-CRITICAL **: g_variant_get_type: assertion 'value != NULL'[   32.265937] set mode: 4
[14:37:07:930](process:915): GLib-CRITICAL **: g_variant_type_is_subtype_of: assertion 'g_variant_type_check (type)'
        failed
[14:37:07:931](process:915): GLib-CRITICAL **: g_variant_get_boolean: assertion 'g_variant_is_of_type (value,
        G_VARIANT_TYPE_BOOLEAN)' failed
[14:37:07:945](process:915): GLib-CRITICAL **: g_variant_unref: assertion 'value != NULL' failed
[14:37:07:945]34.643509: _[   32.297571] change mode:4 finish
[14:37:07:946]get_managed_objects:36: e->message
[14:37:08:212]bluetoothd[1226]: Bluetooth daemon 5.54
```

```
[14:37:08:249]bluetoothd[1226]: Starting SDP server
[14:37:08:250]bluetoothd[1226]: kernel lacks bnep-protocol support
[14:37:08:264]bluetoothd[1226]: System does not support network plugin
[14:37:08:266]bluetoothd[1226]: Bluetooth management interface 1.14 initialized
[14:37:08:322]bluetoothd[1226]: Endpoint registered: sender=:1.2 path=/org/bluez/hci0/A2DP/SBC/Source/1
[14:37:08:334]bluetoothd[1226]: Endpoint registered: sender=:1.2 path=/org/bluez/hci0/A2DP/SBC/Source/2
```

问题分析：

1. 出现问题时，通过日志和复现确认是否有 hci0 设备，确认了 hci0 设备是存在的。所以启动已经进行到 hci0 up 阶段，是 btmanager 阶段失败了；
2. 从”GLib-CRITICAL **” 相关的打印看是 btmanager 内部通过 dbus 进行一些资源的初始化失败了；
3. “Bluetooth daemon 5.54” 打印表示 BlueZ 协议栈刚刚开始初始化；
4. 在 btmanager 初始化时，正常情况下是需要先完成 bluetoothd 的启动，再到 btmagner 后续的启动。

这里明显顺序不对；

5. 怎么才能保证这个时序？尝试在/etc/bluetooth/bt_init.sh 脚本中关于 bluethood 启动的地方添加适当延时，保证 bluetoothd 能在延时期间启动完成；
6. 修改后验证不再复现。

根本原因：

btmanager 和 bluetoothd 启动的时序有交叉，bluetoothd 还没完全初始化好，btmanager 内部就通过 Dbus 访问 bluetoothd，导致访问失败，进而导致 enable 失败。

解决办法：

在 bluetoothd 进程初始化时加上适当延时：

```
d=`ps | grep bluetoothd | grep -v grep`
[ -z "$d" ] && {
   # bluetoothd -n &
   /etc/bluetooth/bluetoothd start
   sleep 1
}
```

思路总结：

• 确认打开失败属于哪个阶段；

• 打开每个阶段的关键 Log；

• 分析每个阶段的关键 Log。

4.1.2 蓝牙 MAC 定制

不同模组厂的 MAC 地址修改方式有差异，但思路大同小异：

（1）怎么保存 MAC 地址；

（2）MAC 地址怎么烧；

（3）怎么获得烧写的 MAC 地址；

（4）蓝牙初始化的时候怎么让 MAC 地址生效。

4.1.2.1 XR 模组蓝牙 MAC 定制

问题描述：

用户期望可以定制 XR829 蓝牙的 mac 地址。

问题背景：

平台：R818

模组：XR829

问题表现：

无法使用期望的蓝牙 Mac 地址。

```
1970-01-16 03:13:12:000: BTMG[_bt_manager_enable:225]:  enable state: 1, now bt adapter state : 0
1970-01-16 03:13:12:001: BTMG[bt_test_adapter_status_cb:66]:  bt is turnning on.
bring up hci0 failed
1970-01-16 03:13:23:108: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:26:108: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:29:109: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:32:109: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:35:109: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:38:110: BTMG[bt_platform_init:31]:  detect hci0.....
1970-01-16 03:13:41:110: BTMG[bt_platform_init:36]:  init hci device failed
1970-01-16 03:13:41:110: BTMG[_bt_manager_enable:246]:  BT turn on fail
1970-01-16 03:13:41:110: BTMG[bt_test_adapter_status_cb:49]:  BT is off
1970-01-16 03:13:41:112: BTMG[pfd1_thread_process:946]:  enter
```

问题分析：

XR829 模组蓝牙 MAC 地址是在蓝牙初始化阶段，由 Host 端传到 Controller 的。

Tina 系统 Mac 地址一般保存在机器/etc/bluetooth/xr_bt.conf 文件中，如下图所示：

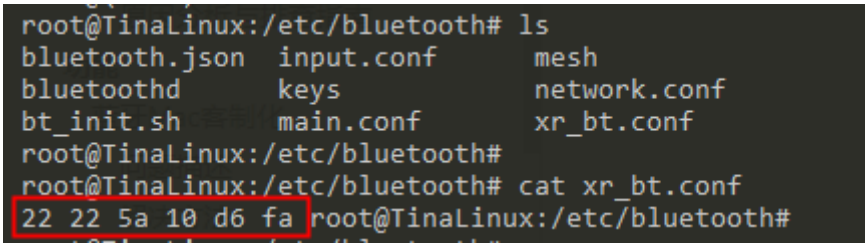


图 4-1: MAC 地址文件

xr_bt.conf 文件是在设备第一次启动的时候创建的，写入该文件中的 Mac 地址可以是随机生成的，也可以从设备的某个文件分区中读取到的，此逻辑相关的代码实现路径：

```
tina/out/xxxxxx/compile_dir/target/bluez-5.54/tools/hciattach_xradio.c
```

如果 xr_bt.conf 文件中已经有合法的 mac 地址，蓝牙初始化的时候就会直接使用。

使用随机的 Mac 地址无法满足用户的需求。因此需要把蓝牙 Mac 写到设备的机器的私有分区或者安全分区中，应用层 sunxi-addr-mgt 驱动获取到。

sunxi-addr-mgt 驱动路径：

```
tina/.../linux-xx/drivers/misc/sunxi-addr-mgt
```

如果 SDK 中没有此驱动，请咨询 AW 技术窗口。还需要确认 dts 配置。

```
    addr_mgt: addr_mgt@0 {
      compatible    = "allwinner,sunxi-addr_mgt";
      type_addr_wifi = <0x0>;
      type_addr_bt   = <0x0>;
      type_addr_eth  = <0x0>;
      status        = "okay";
    };
```

驱动提供节点给用户层使用：

```
/sys/class/addr_mgt/addr_bt
```

因此在 hciattach 中可以通过该节点获取 Mac 地址，增加如下修改：

```
diff --git a/tools/hciattach_xradio.c b/tools/hciattach_xradio.c
index 693aa45..50591da 100644
--- a/tools/hciattach_xradio.c
+++ b/tools/hciattach_xradio.c
@@ -52,6 +52,10 @@
 #define XR_BT_CONF_PATH_NAME "/etc/bluetooth/xr_bt.conf"
 #endif

+#ifndef ADDR_MGT_BT_PATH_NAME
+#define ADDR_MGT_BT_PATH_NAME "/sys/class/addr_mgt/addr_bt"
+#endif
+
 #define UNUSED(x)  (void)(x)

 #define SHOW_LOG     0
@@ -785,16 +789,31 @@ static int xradio_generate_bdaddr(unsigned char *buf)
```

```
   int fd;
   FILE* conf_fd = NULL;
   unsigned mac_hex;
-  int i = 12; /* from MSB to LSB*/
+  int i; /* from MSB to LSB*/
+  int addr_bt_fd;
+  bdaddr_t bdaddr;
+  char addr_bt[18] = {'\0'};
+
+  addr_bt_fd = open(ADDR_MGT_BT_PATH_NAME, O_RDWR);
+  if (addr_bt_fd) {
+     read(addr_bt_fd, addr_bt, sizeof(addr_bt) - 1);
+     str2ba(addr_bt, &bdaddr);
+     for (i = 12; i >= 7; i--) {
+        buf[i] = bdaddr.b[i-7];
+     }
+     close(addr_bt_fd);
+     if (check_bdaddr_valid(buf))
+        return 0;
+  }

   conf_fd = fopen(XR_BT_CONF_PATH_NAME, "r");
   if(conf_fd) {
+     i = 12;
      fscanf(conf_fd, "%x", &mac_hex);
```

根本原因：

随机 Mac 地址方式无法满足 Mac 定制的需求。

解决办法：

增加配置 sunxi-addr-mgt，修改 hciattach 获取蓝牙 Mac 地址的逻辑。

4.2 A2DP Source

支持 A2DP Source 的设备一般连接蓝牙音箱/耳机播放音乐，例如扫描笔产品。常见的问题一般是以下几类：

（1）扫描

（2）连接断开

（3）回连

（4）蓝牙音乐

4.2.1 扫描问题

扫描不到的问题可以从这几个方向思考：

（1）目标设备是否正常工作?

（2）底层协议栈是否正常获得扫描设备信息?

（3）btmanager 处理是否正常?

（4）应用层是否有特殊的过滤?

（5）设备的硬件指标是否达标?

因此可以做以下的排查：

4.2.1.1 验证对端设备

确认对端设备已经处于可发现模式，再使用手机（或者其他设备）扫描，确认设备是否能被正常扫描到。

4.2.1.2 hcitool 工具扫描

Hcitool 工具通过直接调用协议栈的扫描接口下发扫描命令，正常打开蓝牙后，在 adb 窗口中输入 hcitool scan 命令：

```
root@TinaLinux:/# hcitool scan
Scanning ...
	FC:53:9E:3E:0C:86	DDEE
	00:19:86:00:03:C9	PCJIJIAN
	64:A2:00:69:37:5B	zzj
	48:45:20:FE:F0:B9	NBZENGYUG
	7C:2A:DB:F9:67:B7	 RedS
```

4.2.1.3 bluetoothctl 工具扫描

bluetoothctl 工具是 BlueZ 协议栈的一个客户端，如果 hci device 已经初始化完成，并且 bluetoothd 进程也运行起来，可以直接使用，使用此工具可以排查是否为 btmanager 的问题，请按照如下步骤：

（1）打开蓝牙

不经过 btmanager api 打开，通过输入命令：

```
/etc/bluetooth/bt_init.sh start
hciconfig hci0 up
```

当然由于 bluetoothctl 和 btmanager 并不冲突，如果当前蓝牙已经通过 btmanager 打开了，该步骤可以忽略；

（2）运行 bluetoothctl

直接在 adb 窗口中输入 bluetoothctl 即可进入 bluetoothctl 的命令终端后，然后开始扫描操作：

```
scan on  -->打开扫描
scan off -->关闭扫描
devices  -->扫描列表
```

如果使用 bluetoothctl 工具能扫描到设备，可以初步判断是 btmanager 的问题，获取最新的 btmanager 版本验证试试。

4.2.1.4 btmanager demo 扫描

如果应用无法正常获取到扫描结果，使用 bt_test 对比确认下是否是应用处理的问题。

需要注意的是如果设备已经配对过，是不会再有扫描上报的，所以需要检查下已配对列表信息。

4.2.1.5 检查 hcilog

检查 HCI log 确认 HOST 端是否下发 inquiry 命令，是否下发成功。

4.2.1.6 案例 1：客户应用无法获取扫描结果

问题描述：

客户应用发起扫描，但是搜索不到部分设备。

问题背景：

平台: R818

模组：XR829

问题表现：

致在 BQB 测试的时候，无法搜索到 PTS dongle 蓝牙设备。

问题分析：

1. 本次问题从底层往上分析，按照 HCI log -> 协议栈-> btmanager-> 应用的顺序；
2. 抓取 HCI log，检查是否成功下发了 inquiry 命令：

82	Event	0x200a	Low Energy	HCI_LE_Set_Advertising_Enable	Command Complete	Success	4	7
83	Command	0x200b	Low Energy	HCI_LE_Set_Scan_Parameters			7	11
84	Event	0x200b	Low Energy	HCI_LE_Set_Scan_Parameters	Command Complete	Success	4	7
85	Command	0x200c	Low Energy	HCI_LE_Set_Scan_Enable			2	6
86	Event	0x200c	Low Energy	HCI_LE_Set_Scan_Enable	Command Complete	Success	4	7
87	Command	0x0401	Lnk Ctrl	Inquiry			5	9
88	Event	0x0401	Lnk Ctrl	Inquiry	Command Status	Success	4	7
89	Event				LE Advertising Report		26	29

图 4-2: 检查 inquiry 命令

从图中看出已经正常下发。

3. 检查 HCI log 中固件上报的 inquiry Result 看是否有目标设备：

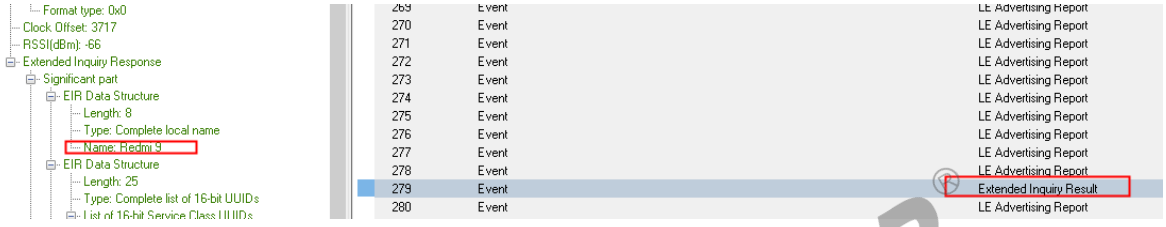


图 4-3: inquiry_result

类似这个，已扫描到目标设备。

4. 通过 hcitool 工具扫描检查，结果正常。
5. 通过 bluetoothctl 工具检查，结果正常。
6. 通过 bt_test 验证，结果正常。
7. 经过以上实验后，基本就怀疑与应用有关系，于是让客户检查涉及扫描相关的逻辑与 API 的使用。

最终发现是应用层对设备进行了过滤。

根本原因：

用户应用对 COD 为蓝牙/耳机类型的 Audio 设备过滤了，而 BQB 测试的 PTS dongle 不是 Audio 类型设备。

解决办法：

在 BQB 测试固件中把过滤去掉。

思路总结：

1. 如果对相关流程不熟悉的情况下，请按照从上到下或者从下到上的顺序进行验证排查。
2. 如果已经有了相关的怀疑点，可以单独对某个步骤进行排查，减少验证时间。

4.2.2 连接断开问题

连接蓝牙音箱/耳机也经常遇到连接失败、难连接的问题。请参考本文档**第 3 章通用排查指南**，或先从以下几个方向思考:

（1）目标设备是否正常工作?

（2）应用和 btmanager 是否正确使用，API 是否正确调用?

（3）连接请求命令是否正确下发给蓝牙模组?

（4）设备的硬件指标是否达标？环境是否很恶劣?

4.2.2.1 检查对端设备

检查对端设备是否已经正常开机，使用手机等设备连接验证是否有问题。

4.2.2.2 检查应用与中间件

（1）检查应用调用 API 是否正确调用;

（2）检查是否正确初始化 A2DP Source，因为蓝牙耳机/音箱设备支持 A2DP Sink，只有 A2DP Source 设备才能正常连上 A2DP Sink 设备;

如果是 bt_test 则是指定参数：

```
bt_test -i -p a2dp-source
```

如果是客户应用，则会通过 API 使能：

```
int bt_manager_enable_profile(int profile);
```

传参：BTMG_A2DP_SOUCE_ENABLE

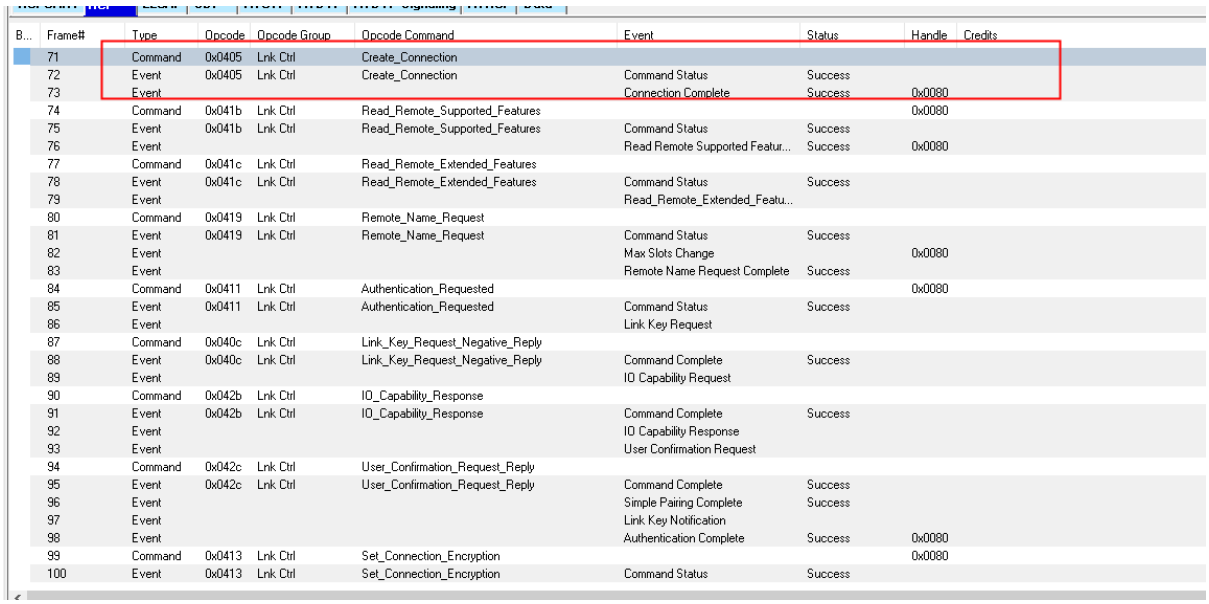
如果是注册多个 profile，则使用或的关系。如：BTMG_A2DP_SOUCE_ENABLE | BTMG_GATT_SERVER_ENABLE。

（3）如果设备未配对（从未连接过或者取消配对），在连接之前，需要先发起扫描，并且要扫描到，才能发起连接。

4.2.2.3 检查 HCI log

通过抓取分析 HCI 来判断，如果下发了还是有问题，可以先尝试更换蓝牙固件验证。

（1）检查 HOST 是否有发起连接, 连接是否正常;

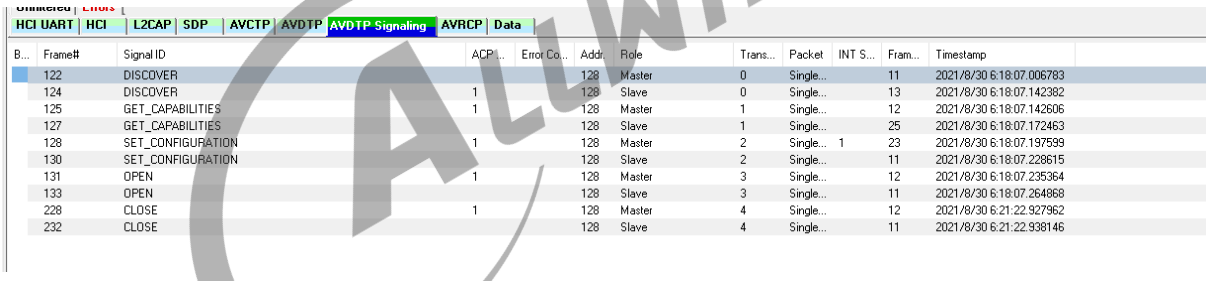


B...	Frame#	Type	Opcode	Opcode Group	Opcode Command	Event	Status	Handle	Credits
	71	Command	0x0405	Lnk Ctrl	Create_Connection				
	72	Event	0x0405	Lnk Ctrl	Create_Connection	Command Status	Success		
	73	Event				Connection Complete	Success	0x0080	
	74	Command	0x041b	Lnk Ctrl	Read_Remote_Supported_Features			0x0080	
	75	Event	0x041b	Lnk Ctrl	Read_Remote_Supported_Features	Command Status	Success		
	76	Event				Read Remote Supported Featur...	Success	0x0080	
	77	Command	0x041c	Lnk Ctrl	Read_Remote_Extended_Features				
	78	Event	0x041c	Lnk Ctrl	Read_Remote_Extended_Features	Command Status	Success		
	79	Event				Read_Remote_Extended_Featu...			
	80	Command	0x0419	Lnk Ctrl	Remote_Name_Request				
	81	Event	0x0419	Lnk Ctrl	Remote_Name_Request	Command Status	Success		
	82	Event				Max Slots Change		0x0080	
	83	Event				Remote Name Request Complete	Success		
	84	Command	0x0411	Lnk Ctrl	Authentication_Requested			0x0080	
	85	Event	0x0411	Lnk Ctrl	Authentication_Requested	Command Status	Success		
	86	Event				Link Key Request			
	87	Command	0x040c	Lnk Ctrl	Link_Key_Request_Negative_Reply				
	88	Event	0x040c	Lnk Ctrl	Link_Key_Request_Negative_Reply	Command Complete	Success		
	89	Event				IO Capability Request			
	90	Command	0x042b	Lnk Ctrl	IO_Capability_Response				
	91	Event	0x042b	Lnk Ctrl	IO_Capability_Response	Command Complete	Success		
	92	Event				IO Capability Response			
	93	Event				User Confirmation Request			
	94	Command	0x042c	Lnk Ctrl	User_Confirmation_Request_Reply				
	95	Event	0x042c	Lnk Ctrl	User_Confirmation_Request_Reply	Command Complete	Success		
	96	Event				Simple Pairing Complete	Success		
	97	Event				Link Key Notification			
	98	Event				Authentication Complete	Success	0x0080	
	99	Command	0x0413	Lnk Ctrl	Set_Connection_Encryption			0x0080	
	100	Event	0x0413	Lnk Ctrl	Set_Connection_Encryption	Command Status	Success		

图 4-4: 检查 HCILog 是否发起连接

如果没有看到 HOST 发起连接的命令，那就说明上层应用接口没有调用发起连接或者发起连接异常。

（2）检查 AVDTP 连接是否正常建立。



HCI UART | HCI | L2CAP | SDP | AVCTP | AVDTP | AVDTP Signaling | AVRCP | Data

B...	Frame#	Signal ID	ACP\...	Error Co...	Addr...	Role	Trans...	Packet	INT S...	Fram...	Timestamp
	122	DISCOVER			128	Master	0	Single...		11	2021/8/30 6:18:07.006783
	124	DISCOVER	1		128	Slave	0	Single...		13	2021/8/30 6:18:07.142382
	125	GET_CAPABILITIES	1		128	Master	1	Single...		12	2021/8/30 6:18:07.142606
	127	GET_CAPABILITIES			128	Slave	1	Single...		25	2021/8/30 6:18:07.172463
	128	SET_CONFIGURATION	1		128	Master	2	Single...	1	23	2021/8/30 6:18:07.197599
	130	SET_CONFIGURATION			128	Slave	2	Single...		11	2021/8/30 6:18:07.228615
	131	OPEN	1		128	Master	3	Single...		12	2021/8/30 6:18:07.235364
	133	OPEN			128	Slave	3	Single...		11	2021/8/30 6:18:07.264868
	228	CLOSE	1		128	Master	4	Single...		12	2021/8/30 6:21:22.927962
	232	CLOSE			128	Slave	4	Single...		11	2021/8/30 6:21:22.938146

图 4-5: 检查 AVDTP 连接

4.2.2.4 检查设备硬件 RF 指标

如果命令已经下发到蓝牙固件端，更换了固件还是连接失败，那就应该确认设备的 RF 是否正常。

4.2.2.5 检查测试环境

如果设备 RF 测试也达标了，还是有问题，有可能环境实在是太恶劣，超出了硬件的规格，可以找个干净的环境测试。

4.2.2.6 案例 1：蓝牙链路断开慢

问题描述：

样机已经连上一个蓝牙耳机，关闭蓝牙耳机后，样机没有及时收到断开事件。

问题背景：

平台：R818

模组：RTL8723DS

问题表现：

设备断开不及时，影响到应用其它逻辑处理。

问题分析：

ACL 链路断开的快慢与 Link supervision timeout 有关，蓝牙 Controller 使用 link supervision timeout 监测连接是否丢失了。一般以下的这种情况 link supervision timeout 会发挥作用：

（1）设备连接超出范围或者被干扰导致物理链接丢失；

（2）设备掉电引起的物理链接丢失

link supervision timeout 一般是双方设备协商的，一般常见的是 5s 或者 20s。

可以主动去调整 link supervision timeout。

根本原因：

样机跟对端蓝牙设备协商的 link supervision timeout 太长。

解决办法：

btmanager 有提供 API 去调整 link supervision timeout：

```
int bt_manager_set_link_supervision_timeout(const char *addr, int slots)
```

addr 是对端蓝牙设备的 mac 地址；

slots 为超时时间，单位为 0.625ms，实际超时时间 = slots * 0.625ms。

需要注意：

（1）必须在设备已连接的情况下去设置，否则无效；

（2）设备断开连接会失效，所以每次连接成功之后需要设置一次。

4.2.3 回连问题

回连有主动回连和被动回连两种。

（1）主动回连

主动回连是指蓝牙打开后，设备可以主动连接周围的蓝牙耳机的功能。目前 btmanager 内部没有主动回连的逻辑。如果用户需要此功能，需要在应用层自己实现该逻辑。

（2）被动回连

被动回连是指蓝牙耳机/音箱主动连接，一般是在蓝牙耳机/音箱刚刚开机或者断开后发起。

4.2.3.1 案例 1：拒绝被动回连

问题描述：

设备已经连接上蓝牙耳机 A，此时打开蓝牙耳机 B，耳机 B 会回连上样机，这样会导致有两个已连接设备，对一些状态管理有影响。用户希望在蓝牙耳机 A 已经连接的情况下，拒绝蓝牙耳机 B 无回连。

问题背景：

平台：R818

模组：RTL8723DS

问题表现：

设备会出现同时连接两个设备的问题，影响状态管理。

问题分析：

1. 蓝牙耳机 B 回连的动作，我们无法阻止，只能想办法拒绝他。应用层得知连上事件的时候，整个连接流程已经完成，只能是连上了再把它断开，但是做法可能会引入其它问题，不是最好的解决方案。
2. 对端蓝牙发起连接，是一个请求-确认过程，因此可以我们在收到对端请求连接实际时就拒绝了。Linux 平台此部分的逻辑在内核的 Bluetooth core 中处理。

根本原因：

本地设备无法停止对端设备的回连行为。

解决办法：

修改驱动：linux-4.9/net/bluetooth/hci_event.c

在连接请求处理函数 hci_conn_request_evt 中增加如下代码：

```
if (ev->link_type == ACL_LINK) {
  if ((hci_conn_num(hdev, ACL_LINK) != 0) && (hdev->dev_type == HCI_PRIMARY)) {
    BT_INFO("already exist acl link, reject new! %s, %d", __func__, __LINE__);
    hci_reject_conn(hdev, &ev->bdaddr);
    return;
  }
}
```

首先判断连接类型是否是 ACL_LINK, 然后获取当前连接数，如果当前已经有连接了，就拒绝本次的连接请求。

思路总结:

多了解蓝牙相关流程，从蓝牙交互上考虑问题。

4.2.4 蓝牙播音问题

A2DP Source 蓝牙音乐常见播放卡顿、丢音的问题，播放音乐的流程是固定的。一个已连接的蓝牙耳机/音箱设备，bluealsa 会将它抽象为一个虚拟声卡。

播放大致的流程如下：

（1）应用层通过 bt_manager_a2dp_src_stream_send() API 把 pcm 数据传入 btmanager；

（2）btmanager 中先将音频数据写到内部的缓存区；

（3）btmanager 中另一个线程会从缓存区取出数据写到 bluealsa 虚拟声卡。由于蓝牙设备连上后一般协商 48000 或者 44100 的采样率，因此虚拟声卡一般只支持 48000 或者 44100。所以此过程可能会包含 Alsa 库的对音频的重采样，例如虚拟声卡支持 44100，而 pcm 数据是 16000 的，alsa 处理音频时就会把 16000 转成 44100;

（4）bluealsa 中会将 pcm 数据编码为 SBC、AAC 等格式，再通过 socket 发给内核蓝牙 core 封装成 ACL 格式的包；

（5）内核蓝牙 core 将数据包发给 HCI 驱动 (UART)，然后发给蓝牙 Controller;

（6）蓝牙 Controller 会将音频数据通过射频通信发给对端 Sink 设备；

（7）Sink 设备的 Controller 收到之后又会通过 HCI 上报给 Sink 设备的 HOST，解码之后播放。

出现播放卡顿、丢音一般是速率不稳定、数据不完整引起的，根据上面的流程，我们可以这几个方向思考卡顿、丢音问题：

（1）应用层传入的音频数据是否完整?

（2）应用层传入数据是否及时?

（3）btmanager 内部数据处理后，数据是否完整?

（4）btmanager 内将数据写入虚拟声卡是否及时?

（5）Alsa 音频重采样后，数据是否完整?

（6）Bluealsa 在编码后，数据是否完整?

（7）HCI 层（UART）传输时，数据是否完整?

（8）蓝牙模组之间无线射频通信，数据是否完整?

（9）对端 Sink 设备是否正确处理音频数据了?

4.2.4.1 应用层排查

方法 1：对比 btmanager demo

bt_test 使用方法请参考《Tina_Linux_ 蓝牙软件开发指南》demo 章节。如果测试出现卡顿，可排除客户应用引起的，可以直接跳到 **4.2.4.2 btmanager 层排查**章节进行下一步的排查。

方法 2：检查应用层的数据是否有丢

（1）用户在应用层 Debug 确认；

（2）通过 btmanager 把数据 Dump 出来。

btmanager 提供有调试节点 Debug，输入命令之后立即生效。

打开 Dump A2DP Source 数据的开关, 请通过 adb 输入：

```
echo mask =128 > /tmp/log_btmg_io
```

关闭 Dump A2DP Source 数据的开关, 请通过 adb 输入：

```
echo mask =0 > /tmp/log_btmg_io
```

当然也可以在应用层调用 bt_manager_set_ex_debug_mask() 设置 128 或者 0 进行开关，但不推荐此方法。

注意：如果不需要 DUMP 数据了，请及时关闭，否则可能会影响到性能；

Dump 的数据保存在小机端，路径:

```
/tmp/a2dp_src_stream.raw
```

通过 adb pull 将文件取出，再通过 PC 软件打开查看，推荐软件：Audacity。软件使用方法请通过互联网查阅，打开文件需要选择文件的一些音频信息（单声道、采样率等），这个需要用户明确自己应用发的音频的参数，否则打开异常。

方法 3：检查应用层发送速率

用户在调用 bt_manager_a2dp_src_stream_send() API 发送数据的时候，由于某些原因导致调

用 API 不及时，导致数据发慢了，这种情况可以通过日志看出，卡顿关键日志信息：“fifo is empty:a2src(0,1)”、“An underrun has occurred”。

默认打印如下：

```
1970-01-17 08:50:20:181: [pcm_open:156]: pcm open:bluealsa:DEV=FC:E8:06:B7:22:FB
1970-01-17 08:50:20:216: [aw_pcm_set_sw_params:105]: set threadshold :17640
1970-01-17 08:50:20:360: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,1)
1970-01-17 08:50:20:414: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,2)
1970-01-17 08:50:20:449: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,3)
1970-01-17 08:50:20:500: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,4)
1970-01-17 08:50:20:552: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,5)
1970-01-17 08:50:20:627: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,6)
1970-01-17 08:50:20:654: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,7)
1970-01-17 08:50:20:705: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,8)
1970-01-17 08:50:20:756: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,9)
1970-01-17 08:50:20:807: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,10)
1970-01-17 08:50:20:983: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,11)
1970-01-17 08:50:21:019: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,12)
1970-01-17 08:50:21:094: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,13)
1970-01-17 08:50:21:121: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,14)
1970-01-17 08:50:21:172: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,15)
1970-01-17 08:50:21:223: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,16)
1970-01-17 08:50:21:300: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,17)
1970-01-17 08:50:21:328: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,18)
1970-01-17 08:50:21:381: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,19)
1970-01-17 08:50:21:381: BTMG[comsumer_thread:634]: time_ms[1020] empty_count:19
1970-01-17 08:50:21:409: BTMG[comsumer_thread:612]: trigger cache timeout:a2src,19
1970-01-17 08:50:21:709: [aw_pcm_write:305]: An underrun has occurred
1970-01-17 08:50:21:709: BTMG[_a2dp_src_pcm_write:62]: pcm write fail
1970-01-17 08:50:21:802: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,1)
1970-01-17 08:50:21:838: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,2)
1970-01-17 08:50:21:888: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,3)
```

如果 btmanager 的打印等级设置为 DEBUG，如果看到 “data cache is below threshold” 的打印，这个可以

基本说明数据发慢了，日志如下：

```
1970-01-17 08:51:38:930: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,4)
1970-01-17 08:51:38:957: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:38:981: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,5)
1970-01-17 08:51:39:008: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:39:055: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,6)
1970-01-17 08:51:39:058: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:39:081: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,7)
1970-01-17 08:51:39:108: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:39:132: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,8)
1970-01-17 08:51:39:159: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:39:182: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,9)
1970-01-17 08:51:39:209: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:39:232: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,10)
1970-01-17 08:51:39:260: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:39:311: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
1970-01-17 08:51:39:362: BTMG[bt_a2dp_src_stream_send:150]: data cache is below threshold
```

通过以上的 log 能够间接说明速率发慢了，但是还是不够直观。通过 btmanager 提供有打开速率统计的 Debug 节点。

打开命令：

```
echo mask =16 > /tmp/log_btmg_io
```

关闭命令：

```
echo mask =0 > /tmp/log_btmg_io
```

打开之后，速率统计会每隔 500ms 打印一次，如下所示：

```
1970-01-17 09:14:08:033: [aw_pcm_write:305]: An underrun has occurred
1970-01-17 09:14:08:034: BTMG[_a2dp_src_pcm_write:62]: pcm write fail
1970-01-17 09:14:08:081: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,1)
1970-01-17 09:14:08:127: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,2)
1970-01-17 09:14:08:177: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,3)
1970-01-17 09:14:08:227: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,4)
1970-01-17 09:14:08:302: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,5)
1970-01-17 09:14:08:305: BTMG[bt_a2dp_src_stream_send:132]: time_ms[521] tot[36864] len[4096] speed[70756]
1970-01-17 09:14:08:328: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,6)
1970-01-17 09:14:08:378: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,7)
1970-01-17 09:14:08:429: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,8)
1970-01-17 09:14:08:503: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,9)
1970-01-17 09:14:08:534: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,10)
1970-01-17 09:14:08:585: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,11)
1970-01-17 09:14:08:635: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,12)
1970-01-17 09:14:08:685: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,13)
1970-01-17 09:14:08:812: BTMG[bt_a2dp_src_stream_send:132]: time_ms[505] tot[40960] len[4096]speed[81108]
1970-01-17 09:14:08:861: BTMG[comsumer_thread:628]: fifo is empty:a2src(0,14)
1970-01-17 09:14:08:861: BTMG[comsumer_thread:634]: time_ms[1130] empty_count:14
1970-01-17 09:14:08:863: BTMG[comsumer_thread:612]: trigger cache timeout:a2src,14
1970-01-17 09:14:09:162: [aw_pcm_write:305]: An underrun has occurred
1970-01-17 09:14:09:163: BTMG[_a2dp_src_pcm_write:62]: pcm write fail
```

如上 Log 中 “BTMG[bt_a2dp_src_stream_send:132]: time_ms[505] tot[40960] len[4096]speed[81108]” 的信息可以得知此时的速率为 81108。在这里测试的是 44100HZ 双声道的音频，正常的速率为 176400，上面的 log 因为故意在发送前加了很大延时，所以速率偏差很大。

这里的速率表示 1s 的音频数据大小，速率的计算方法：

速率 = 音频采样率 * 音频通道数 * 采样位数

- 音频采样率：常见的有 16000、44100、48000 等；
- 音频通道数：一般为 1（单声道）或者 2（双声道）；
- 采样位数: 一般为 2 字节；每个采样数据记录的是振幅, 采样精度取决于采样位数的大小:

  1 字节 (8bit) 只能记录 256 个数, 也就是只能将振幅划分成 256 个等级；

  2 字节 (16bit) 可以细到 65536 个数, 这已是 CD 标准, 一般情况是这种；

  4 字节 (32bit) 能把振幅细分到 4294967296 个等级。

正常情况都是 2。

例如上面说到的 44100HZ 2 声道 16bit 的音频，速率计算如下：

```
速率 = 44100 * 2 * 2 = 176400
```

通过查看速率统计，来判断是否发慢了：

- 如果统计速率一直小于理论速率，属于不正常现象，需要进一步往前排查；
- 如果统计速率在理论速率上下波动，波动幅度不大，属于正常现象；
- 如果统计速率大于或者稍微大于理论速率，属于正常现象。

4.2.4.2 btmanager 层排查

如果排查应用层没发现异常，可以往下排查 btmanager。

方法 1：通过 aplay 播放对比

如果怀疑 btmanager 对数据处理有问题，则可以单独做个尝试让数据不经过 btmanager。可以通过 aplay 直接对 bluealsa 虚拟声卡播放，由于 bluealsa 版本不一样命令有差异，为了兼容老版本，在此分开介绍：

（1）通过在终端输入 “bluealsa -V” 命令获取当前版本号；

（2）bluealsa v1.3.1 版本命令：

```
aplay -D bluealsa:HCI=hci0,DEV=XX:XX:XX:XX:XX:XX,PROFILE=a2dp /tmp/bt_test.wav
```

（3）bluealsa v3.0.0 以上版本命令：

```
aplay -D bluealsa:DEV=XX:XX:XX:XX:XX:XX,PROFILE=a2dp /tmp/bt_test.wav
```

- DEV=XX:XX:XX:XX:XX:XX：表示当前已连接的设备 mac 地址；
- /tmp/bt_test.wav：表示要播放的音频，路径和文件请根据实际情况选择。

注意：在使用 aplay 播放之前，务必先连接上蓝牙设备，并且不能与 btmanager API 同时播放。

方法 2：检查数据是否有丢失

btmanager 提供有调试节点 Debug 把数据 Dump 出来：

打开命令：

```
echo mask =256 > /tmp/log_btmg_io
```

关闭命令：

```
echo mask =0> /tmp/log_btmg_io
```

Dump 出的数据保存在小机端，路径：

```
/tmp/a2dp_bluealsa.raw
```

注意：如果不需要 DUMP 数据了，请及时关闭，否则可能会影响到性能。

方法 3：检查写入 bluealsa 是虚拟声卡的速率

这种情况发生的可能性较小，有必要可以通过速率统计进行排查。

打开命令：

```
echo mask =8> /tmp/log_btmg_io
```

关闭命令：

```
echo ex_dbg 0 > /tmp/bt_io
```

输入命令立即生效，参考日志如下：

```
1970-01-03 00:16:24:577: BTMG[_a2dp_src_pcm_write:89]: time_ms[1551] tot[163840] len[4096]
ex_frames[1024] ac_frames[-32] speed[105635]
1970-01-03 00:16:25:932: [aw_pcm_write:305]: An underrun has occurred
1970-01-03 00:16:25:962: BTMG[_a2dp_src_pcm_write:63]: pcm write fail
1970-01-03 00:16:26:083: BTMG[_a2dp_src_pcm_write:89]: time_ms[1506] tot[159744] len[4096]
ex_frames[1024] ac_frames[1024] speed[106071]
1970-01-03 00:16:27:315: [aw_pcm_write:305]: An underrun has occurred
1970-01-03 00:16:27:345: BTMG[_a2dp_src_pcm_write:63]: pcm write fail
1970-01-03 00:16:27:613: BTMG[_a2dp_src_pcm_write:89]: time_ms[1529] tot[159744] len[4096]
ex_frames[1024] ac_frames[1024] speed[104476]
1970-01-03 00:16:28:707: [aw_pcm_write:305]: An underrun has occurred
1970-01-03 00:16:28:737: BTMG[_a2dp_src_pcm_write:63]: pcm write fail
1970-01-03 00:16:29:117: BTMG[_a2dp_src_pcm_write:89]: time_ms[1504] tot[155648] len[4096]
ex_frames[1024] ac_frames[1024] speed[103489]
```

速率的计算说明与判断请参考 “应用层排查” 章节，这里不再赘述。

4.2.4.3 bluealsa 层排查

在应用层和 btmanager 都无异常情况下，需要继续往下排查。

方法 1：抓 HCI Log 分析

1. 通过 HCI Log 可以间接反应 Bluealsa 中数据处理是否有异常，注意抓取的 HCI log 必须包含设备连接的过程，否则在 FrontLine 软件中 A2DP 数据包无法正常解码出来。

使用 FrontLine 软件打开 HCI log 后，主界面如下，点击下图所示的图标，即可进入音频播放界面：

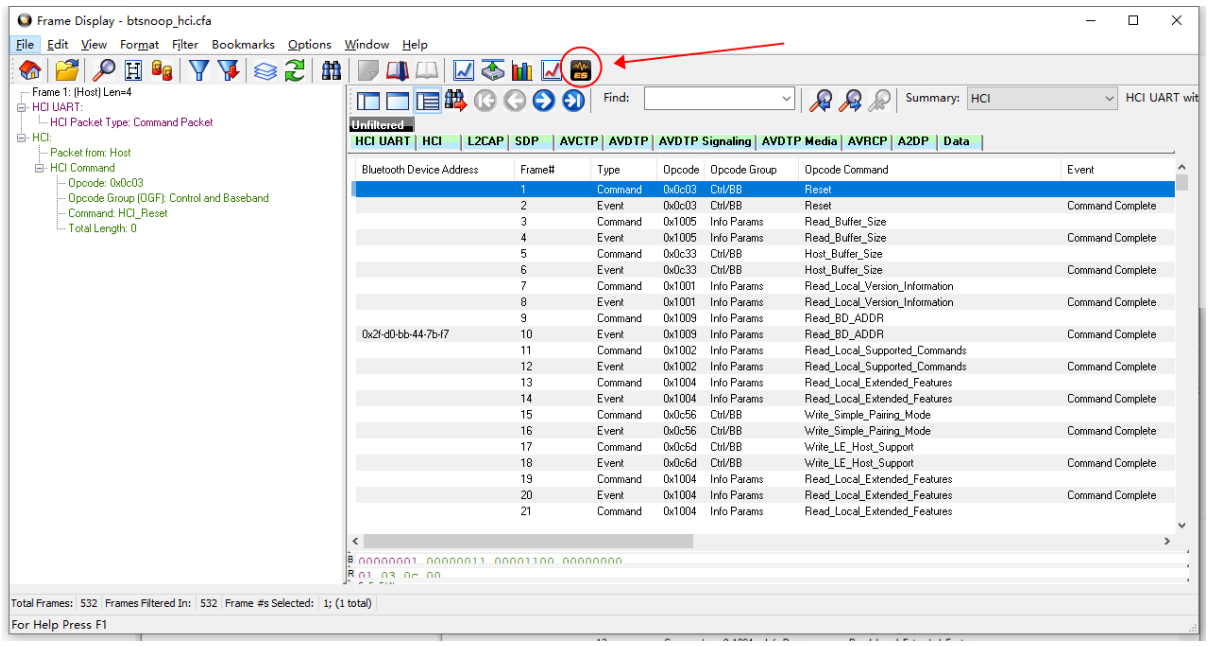


图 4-6: HCI_Log 分析音频

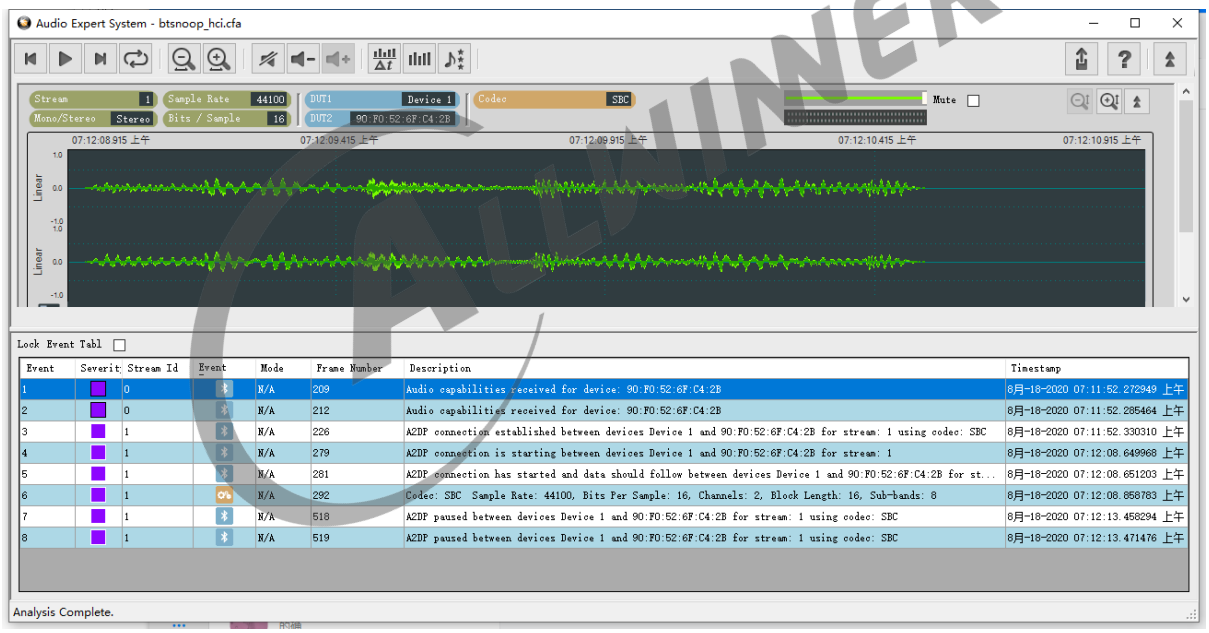


图 4-7: 播放界面

如果要导出音频，点击如下图标：

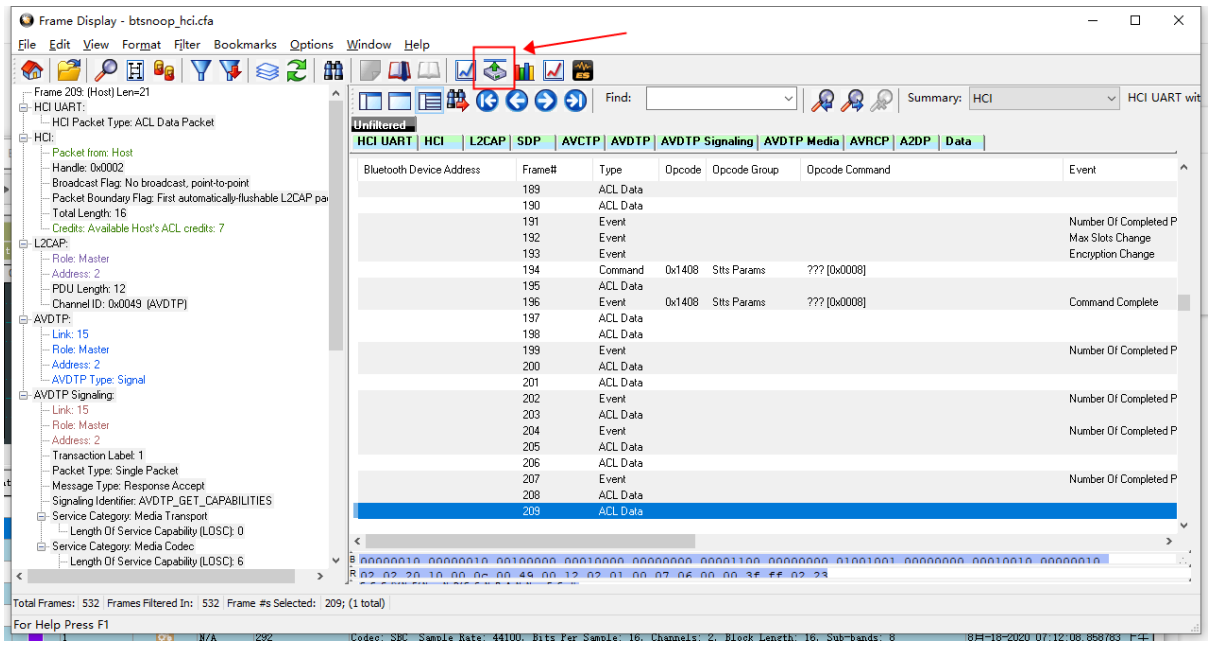


图 4-8: 导出音频

2. 分析 A2DP 数据帧

如果通过听感判断不出卡顿，也不能保证就是没问题了，因为 FrontLine 软件里有 250ms 的缓存。所以还是需要再次确认一下 A2DP 的数据帧有没有异常。

（1）检查 A2DP 数据包是否有明显异常，如下图所示直接有标红的报错：

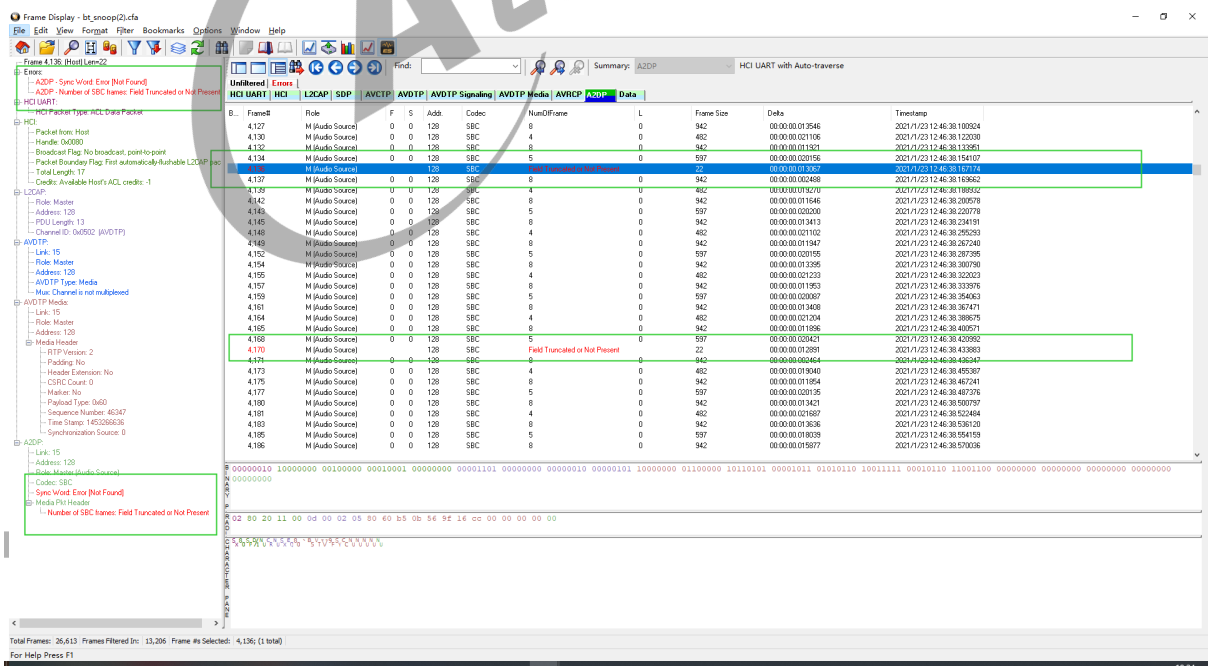


图 4-9: 分析 A2DP 数据帧

（2）检查每个数据包的时间间隔是否平稳，如果忽快忽慢就会有问题。

4.2.4.4 HCI 层排查

如果进行了前面的排查没发现有异常，需要继续往下排查 HCI 传输层。目前在 Tina 平台上使用的蓝牙 HCI 接口一般是 UART。UART 传输时丢包了也会导致蓝牙音乐播放卡顿等问题，需要找负责 UART 模组的同事确认 UART 自发自收是否有问题。

4.2.4.5 无线射频传输排查

蓝牙传输是一种无线传输，容易受到环境的干扰，因此蓝牙传输是否稳定与环境有很大的关系，蓝牙模块或者设备的射频性能也是至关重要。

方法 1：检查机器射频指标

在进行 WiFi/BT 测试之前，就应该要保证机器的射频性能指标达标。射频性能指标需要找硬件同事确认，如果不太确定是否达标，务必先重新测试确认。

方法 2：BT only 测试

关闭 Wi-Fi 或者将 Wi-Fi 驱动卸载后复现问题，排查是否与 Wi-Fi/BT 共存有关。

方法 3：更换干净环境测试

板子的射频性能再好，它的抗干扰能力也是有个上限的。如果使用射频性能达标的机器验证还是存在卡顿的问题，需要了解当前测试环境复杂度，了解下测试环境中的 WiFi、蓝牙设备的数量以及他们当前的行为。同时将机器拿到干扰小的环境测试对比看下是否有改善，至于机器应该在哪种程度干扰下播放不卡顿，这需要结合产品的应用场景和模组厂的意见，甚至市面上的竞品来进行考量。

方法 4：抓 air log 分析

抓空包一般使用 Ellisys Bluetooth Analyzer 工具，如果用户有抓 air log 的条件，建议到干净的环境下抓下 air log 提供给模组厂分析。或者让模组厂协助抓取分析。

4.2.4.6 兼容性排查

可能对端设备本身就有问题，或者与它存在兼容性问题。

方法 1: 使用同类设备对比

如果使用手机等设备连接播放也存在卡顿问题，可以判断是耳机/音箱端的问题。

方法 2：检查是否与音频编解码格式有关

通过 HCI log 确认卡顿时使用的编码格式，再强制修改本地设备编码格式为不一样的格式。比如卡顿时双方协商的是 AAC 格式，强制修改为 SBC 格式之后看下是否还卡顿。需要修改 bluealsa 源码

的 a2dp.c 里面的 *a2dp_codecs[]。

如下代码所示，SBC 是默认打开的，并且 SBC 的优先级是最低的：

```
const struct a2dp_codec *a2dp_codecs[] = {
#if ENABLE_LDAC
  &a2dp_codec_source_ldac,
# if HAVE_LDAC_DECODE
  &a2dp_codec_sink_ldac,
# endif
#endif
#if ENABLE_APTX_HD
  &a2dp_codec_source_aptx_hd,
# if HAVE_APTX_HD_DECODE
  &a2dp_codec_sink_aptx_hd,
# endif
#endif
#if ENABLE_APTX
  &a2dp_codec_source_aptx,
# if HAVE_APTX_DECODE
  &a2dp_codec_sink_aptx,
# endif
#endif
#if ENABLE_FASTSTREAM
  &a2dp_codec_source_faststream,
  &a2dp_codec_sink_faststream,
#endif
#if ENABLE_AAC
  &a2dp_codec_source_aac,
  &a2dp_codec_sink_aac,
#endif
#if ENABLE_MPEG
# if ENABLE_MP3LAME
  &a2dp_codec_source_mpeg,
# endif
# if ENABLE_MP3LAME || ENABLE_MPG123
  &a2dp_codec_sink_mpeg,
# endif
#endif
  &a2dp_codec_source_sbc,
  &a2dp_codec_sink_sbc,
  NULL,
};
```

方法 3：检查是否与音频采样率有关

通过强制采样率为 48000Hz 或者 44100Hz 进行对比。

修改 bluealsa 源码的 a2dp.c 文件，例如强制修改 SBC 的采样率为 48000Hz：

```
 int a2dp_codecs_init(void) {
   if (config.a2dp.force_44100)
     a2dp_codec_source_sbc.capabilities.sbc.frequency = SBC_SAMPLING_FREQ_44100;

+   a2dp_codec_source_sbc.capabilities.sbc.frequency = SBC_SAMPLING_FREQ_48000;
...
```

4.2.4.7 案例 1：A2DP Source 播放丢尾音

问题描述：

样机连接蓝牙耳机播放音乐，概率出现丢尾音，播放不完整。

问题背景：

平台：R818

模组：XR829

问题表现：

播放不完整，丢尾音。

问题分析：

按照排查步骤：

1. 应用层排查

由于丢音，于是先通过 Dump 数据的方式进行排查，Dump 出的数据正常，因此应用数据发送正常。

2. btmanager 排查

检查 btmanager 往 bluealsa 虚拟声卡发送的数据是否有丢，同样通过 DUMP 数据的方式。

发现数据有丢，所以怀疑与 btmanager 相关。

3. 使用 bt_test 测试验证

发现使用 bt_test 并不能复现问题。

4. 检查 API 调用的流程

a2dp source 的播放流程如下：

```
a.bt_manager_a2dp_src_init

b.bt_manager_a2dp_src_stream_start

c.bt_manager_a2dp_src_stream_send

d.bt_manager_a2dp_src_stream_stop
```

```
e.bt_manager_a2dp_src_deinit
```

让用户检查 API 调用流程与参数。后来发现在出现丢音的场景，调用的 API 是 bt_manager_a2dp_src_stream_stop(true)。

根本原因：

用户的应用调用 API 参数使用错误导致。

解决办法：

bt_manager_a2dp_src_stream_stop 的定义如下：

```
int bt_manager_a2dp_src_stream_stop(bool drop)
```

drop 参数说明：

- true：立即停止播放，但是会丢弃数据；
- false：保证缓存数据处理完再停止播放。

最终修改应用调用 bt_manager_a2dp_src_stream_stop() 的传参。

根据合适场景正确使用 bt_manager_a2dp_src_stream_stop。

思路总结：

如果不希望马上停止播放，请调用 bt_manager_a2dp_src_stream_stop(false)，否则会丢数据导致丢音。

4.2.4.8 案例 2：特定耳机播放无声

问题描述：

样机连接大部分耳机、音箱播放均没有出现播放无声都问题，但是连接 JBL JR310BT 耳机就很会出现播放无声的情况，并且概率非常高，但是该耳机使用手机去连接播放正常。

问题背景：

平台：V833

模组：XR829

问题表现：

播放无声

问题分析：

1. 此问题发现的时候已经确定与特定的耳机有关，并且该耳机使用手机播放正常，因此初步怀疑是某些行引起了兼容性问题。
2. 排查应用层

把数据 Dump 出来，数据正常，所以与应用层应该没关系。使用 bt_test 测试，似乎也没有问题。

3. 排查 btmanager

出现问题的时候，btmanager 的 log 一切都是正常的，声卡也打开了，数据也在处理发送，并且把数据 Dump 出来也是正常的，因此看起来这个阶段正常。

4. 分析 HCI log

抓取 HCI log 分析看到下发的数据也是完整的，没看到有什么数据异常。因此还需要往下查。

5. HCI 传输层分析

由于连接其它耳机是正常，所以 UART 应该没有问题，所以暂时不用做自发自收测试。

6. 怀疑是兼容性问题

兼容性问题通过 air log 分析才直观，因此抓了 air log，初步发现了问题点：

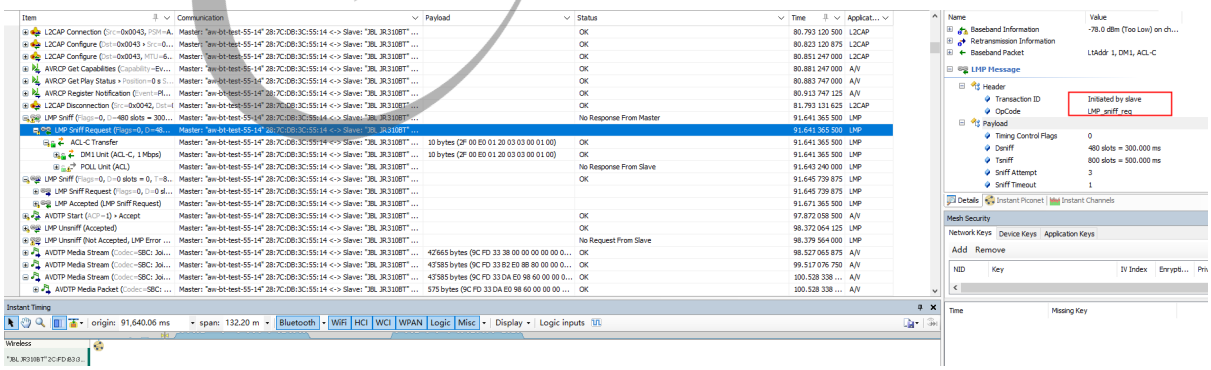


图 4-10: air_log

耳机端已经进入 Sniff 状态，不会接收样机发送的音频数据，所以出现播放无声的现象。

7. 进入 Sniff 是需要时间的，可以推断，之前播放无声之所以是概率出现，是因为等待时间不一样引起。于是我们做了一个实验：

连上蓝牙耳机之后，等 10 秒之后才开始播放;

果然验证跟预期一样，只要时间够长，就变成了必现问题。

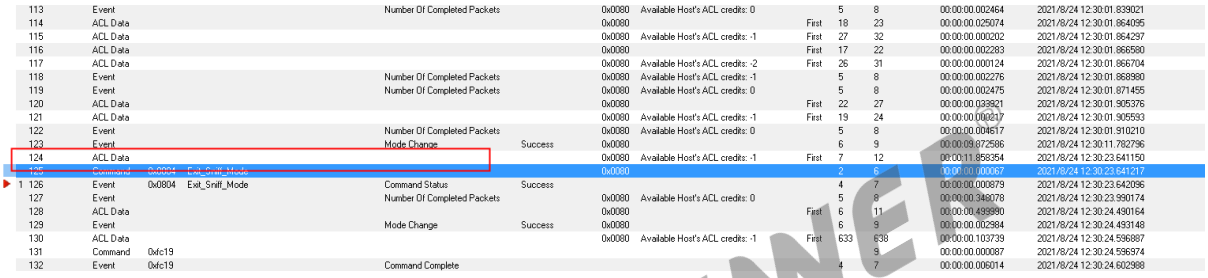
根本原因：

耳机进入了 sniff 模式没能及时退出。

耳机没能及时退出 sniff 模式呢？原因有两个：

（1）样机发起 unsniff req 流程后，JBL 耳机也发起 unsniff req 流程，作为 master，我方拒绝了该重复的 LMP 流程，从而导致 JBL 音箱异常，但是从协议层面来说，我方拒绝是合理的。

（2）JBL 发起 unsniff 流程的原因是我方 HOST 端先发送 A2DP START 数据包给 JBL，再启动 unsniff 流程：



113	Event		Number Of Completed Packets		0x0080	Available Host's ACL credits: 0		5	8	00:00:00.002464	2021/8/24 12:30:01.839021
114	ACL Data				0x0080		First	18	23	00:00:00.025074	2021/8/24 12:30:01.864095
115	ACL Data				0x0080	Available Host's ACL credits: -1	First	27	32	00:00:00.000202	2021/8/24 12:30:01.864297
116	ACL Data				0x0080		First	17	22	00:00:00.002283	2021/8/24 12:30:01.866580
117	ACL Data				0x0080	Available Host's ACL credits: -2	First	26	31	00:00:00.000124	2021/8/24 12:30:01.866704
118	Event		Number Of Completed Packets		0x0080	Available Host's ACL credits: -1		5	8	00:00:00.002276	2021/8/24 12:30:01.868980
119	Event		Number Of Completed Packets		0x0080	Available Host's ACL credits: 0		5	8	00:00:00.002475	2021/8/24 12:30:01.871455
120	ACL Data				0x0080		First	22	27	00:00:00.033921	2021/8/24 12:30:01.905376
121	ACL Data				0x0080	Available Host's ACL credits: -1	First	19	24	00:00:00.000217	2021/8/24 12:30:01.905593
122	Event		Number Of Completed Packets		0x0080	Available Host's ACL credits: 0		5	8	00:00:00.004617	2021/8/24 12:30:01.910210
123	Event		Mode Change	Success	0x0080			6	9	00:00:09.872586	2021/8/24 12:30:11.782796
124	ACL Data				0x0080	Available Host's ACL credits: -1	First	7	12	00:00:11.858354	2021/8/24 12:30:23.641150
125	Command	0x0804	Exit_Sniff_Mode		0x0080			2	6	00:00:00.000067	2021/8/24 12:30:23.641217
126	Event	0x0804	Exit_Sniff_Mode	Command Status	Success			4	7	00:00:00.000879	2021/8/24 12:30:23.642096
127	Event		Number Of Completed Packets		0x0080	Available Host's ACL credits: 0		5	8	00:00:00.349078	2021/8/24 12:30:23.990174
128	ACL Data				0x0080		First	6	11	00:00:00.499880	2021/8/24 12:30:24.490164
129	Event		Mode Change	Success	0x0080			6	9	00:00:00.002984	2021/8/24 12:30:24.493148
130	ACL Data				0x0080	Available Host's ACL credits: -1	First	633	638	00:00:00.103739	2021/8/24 12:30:24.596887
131	Command	0xfc19							9	00:00:00.000087	2021/8/24 12:30:24.596974
132	Event	0xfc19		Command Complete				4	7	00:00:00.006014	2021/8/24 12:30:24.602988

图 4-11: hci_log

解决办法：

避免让 JBL 耳机发起 unsniff 流程。

JBL 耳机发起 unsniff 流程的原因是我方先发送 A2DP START 数据包给 JBL，然后再启动 unsniff 流程。如果先启动 unsniff 流程再发送 A2DP START 数据包，则可以解决此问题，此逻辑可以通过修改 linux 内核的代码完成，补丁如下所示：

```
diff --git a/include/net/bluetooth/hci_core.h b/include/net/bluetooth/hci_core.h
index d8768f971fa5..34cf8feb808e 100644
--- a/include/net/bluetooth/hci_core.h
+++ b/include/net/bluetooth/hci_core.h
@@ -34,6 +34,8 @@
 /* HCI priority */
 #define HCI_PRIO_MAX  7

+extern atomic_t exit_sniff_cmd;
+
 /* HCI Core structures */
 struct inquiry_data {
    bdaddr_t    bdaddr;
diff --git a/net/bluetooth/hci_core.c b/net/bluetooth/hci_core.c
index 3fd542e075b8..58d20c0cb610 100644
--- a/net/bluetooth/hci_core.c
+++ b/net/bluetooth/hci_core.c
@@ -3507,6 +3507,10 @@ void hci_send_acl(struct hci_chan *chan, struct sk_buff *skb, __u16 flags)
```

```
   BT_DBG("%s chan %p flags 0x%4.4x", hdev->name, chan, flags);

+  if (chan->conn->type == ACL_LINK ) {
+     hci_conn_enter_active_mode(chan->conn, bt_cb(skb)->force_active);
+  }
+
   hci_queue_acl(chan, &chan->data_q, skb, flags);

   queue_work(hdev->workqueue, &hdev->tx_work);
@@ -3787,10 +3791,10 @@ static void hci_sched_acl_pkt(struct hci_dev *hdev)
            break;

         skb = skb_dequeue(&chan->data_q);
-
+#if 0
         hci_conn_enter_active_mode(chan->conn,
                    bt_cb(skb)->force_active);
-
+#endif
         hci_send_frame(hdev, skb);
         hdev->acl_last_tx = jiffies;

@@ -3840,9 +3844,10 @@ static void hci_sched_acl_blk(struct hci_dev *hdev)
         if (blocks > hdev->block_cnt)
            return;

+#if 0
         hci_conn_enter_active_mode(chan->conn,
                    bt_cb(skb)->force_active);
-
+#endif
         hci_send_frame(hdev, skb);
         hdev->acl_last_tx = jiffies;
```

思路总结：

怀疑是设备兼容性问题尽快想办法抓 airl log 分析。

4.3 A2DP Sink

A2DP Sink 常见问题一般有以下的类型：

（1）无法被扫描；

（2）连接失败；

（3）连接断开；

（4）蓝牙音乐。

4.3.1 无法被扫描问题排查

无法被扫描到可以从以下方向思考：

（1）设备是否处于可扫发现状态？

（2）设备的天线是否正常？是否能把信号传递给别人？

4.3.1.1 检查设备硬件射频

检查天线是否正常使用，射频指标是否达标。

4.3.1.2 检查蓝牙的状态

对经典蓝牙来说：

- 如果要被对端扫描到，则需要使能 inquiry scan；
- 如果要能够被连接，则需要使能 page scan。

因此需要想办法判断本设备是否已经使能 inquiry scan。

方法 1：通过 hciconfig -a 命令

设备端执行 hciconfig -a 命令，如下图所示：

```
root@TinaLinux:/# hciconfig -a
hci0:   Type: Primary  Bus: UART
        BD Address: 22:22:21:79:2D:90  ACL MTU: 1021:8  SCO MTU: 255:4
        UP RUNNING PSCAN ISCAN
        RX bytes:655 acl:0 sco:0 events:37 errors:0
        TX bytes:1151 acl:0 sco:0 commands:37 errors:0
        Features: 0xbf 0xfe 0xcd 0xfe 0xdb 0xfd 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH SNIFF
        Link mode: SLAVE ACCEPT
        Name: 'aw-bt-test-2D-90'
        Class: 0x040000
        Service Classes: Rendering
        Device Class: Miscellaneous,
        HCI Version: 4.1 (0x7)  Revision: 0xc21
        LMP Version: 4.1 (0x7)  Subversion: 0xc21
        Manufacturer: not assigned (1597)

root@TinaLinux:/# s
```

图 4-12: hciconfig-a 命令

方法 2：检查 HCI log

检查下发的 write_scan_enable 命令是否参数中是否使能了 inquiry scan:

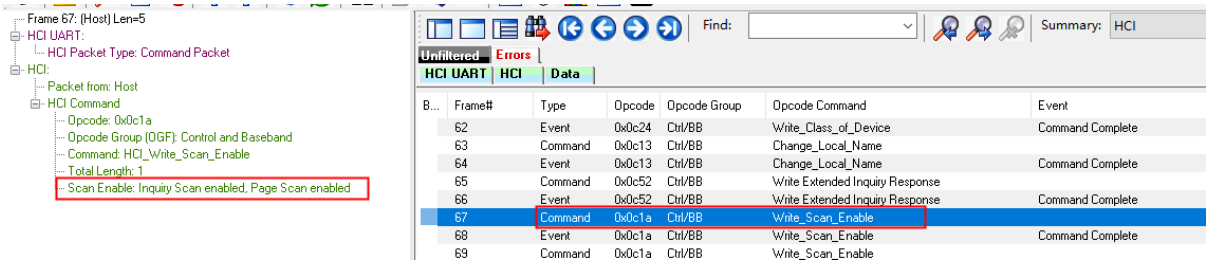


图 4-13: 检查 HCI_log

方法 3：通过 bluetoothctl 工具判断

进入 bluetoothctl 工具的二级终端，再输入 show 命令即可查看:

```
root@TinaLinux:/# bluetoothctl
Agent registered
[bluetooth]# show
Controller 22:22:21:79:2D:90 (public)
        Name: BlueZ 5.54
        Alias: aw-bt-test-2D-90
        Class: 0x00040000
        Powered: yes
        Discoverable: yes
        DiscoverableTimeout: 0x000000b4
        Pairable: yes
        UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        Modalias: usb:v1D6Bp0246d0536
        Discovering: no
Advertising Features:
        ActiveInstances: 0x00
        SupportedInstances: 0x05
        SupportedIncludes: tx-power
        SupportedIncludes: appearance
        SupportedIncludes: local-name
```

图 4-14: bluetoothctl 工具

（1）检查 “Discoverable” 字段是否是 yes，同时 Discoverable 是有超时，超时时间一般是 180s。超过这段时间协议栈会自动将 inquiry scan 进行 disable。

（2）某些手机设备，即使蓝牙设备打开了 inquiry scan，但是没有注册对的 profile，可能也会搜索不到。

4.3.1.3 正确设置蓝牙可发现

通过以上排查，如果设备没有使能 inquiry scan，可能有两种情况：

情况 1：API 使用错误

检查应用是否正确调用如下 API：

```
int bt_manager_set_scan_mode(btmg_scan_mode_t mode)
typedef enum {
  BTMG_SCAN_MODE_NONE,
  BTMG_SCAN_MODE_CONNECTABLE,
  BTMG_SCAN_MODE_CONNECTABLE_DISCOVERABLE, //可被发现，可被连接
} btmg_scan_mode_t;
```

情况 2：btmanaer 调用失败

如果 API 已经调用，但是还是没有可发现，可以再通过 hciconfig 工具设置：

```
hciconfig hci0 piscan
```

如果通过命令设置后正常了，需要检查 btmanager 的版本。

情况 3：蓝牙 firmware 问题

如果 API 已经正确调用，而且从 HCI log 看已经正常下发命令，则可能是蓝牙模组端异常，需要更新 firmware 验证。

4.3.2 连接失败问题排查

方法 1：检查是否有注册 A2DP Sink Profile 以及是否使能可被连接。

参考 <4.3.1.2 检查蓝牙的状态 > 章节。

方法 2：检查 HCI log

（1）检查对端设备是否有发起连接请求

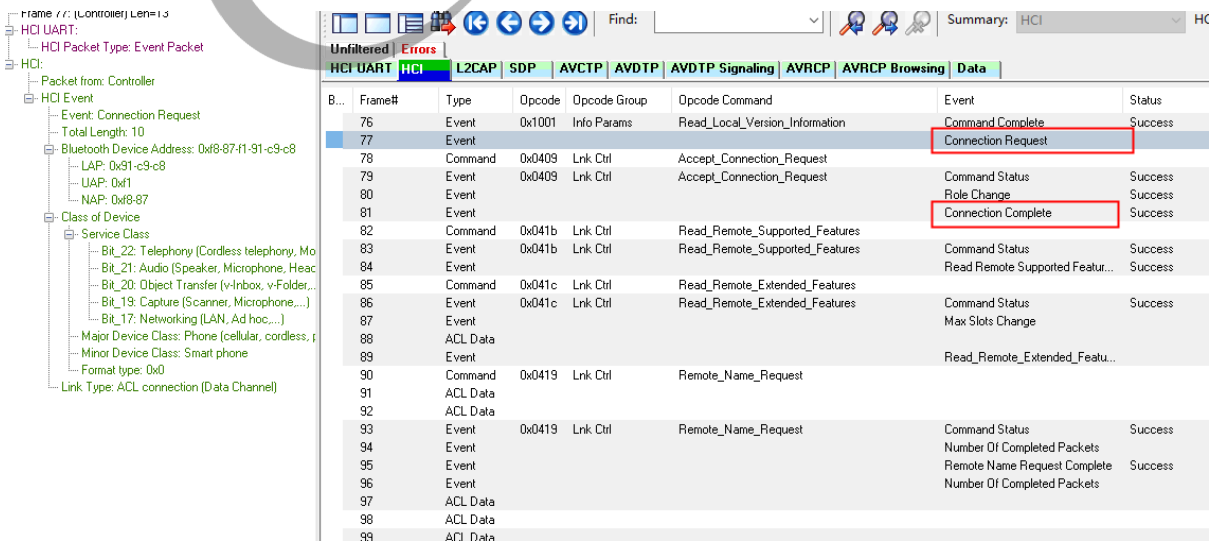


图 4-15: 检查对端设备是否有发起连接

（2）AVDTP Signaling 连接是否正常

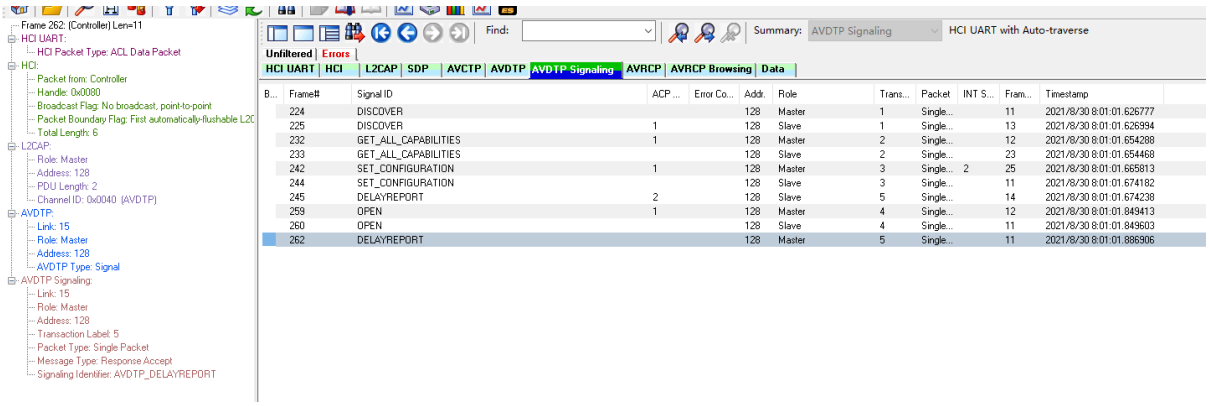


图 4-16: AVDTP_Signaling 连接是否正常

可以对比正常和异常的 hci log，确认有什么差异。

4.3.3 蓝牙音乐问题排查

A2DP Sink 蓝牙音乐可能有如下情况：

（1）播放无声；

（2）播放卡顿。

4.3.3.1 播放无声

方法 1：检查声卡是否选择正确

声卡配置在 bluetooth.json 中，小机端路径：/etc/bluetooth/bluetooth.json，如下图所示：

```
root@TinaLinux:/etc/bluetooth# cat bluetooth.json
{
        "profile":{
                "a2dp_sink":1,
                "a2dp_source":0,
                "avrcp":1,
                "hfp_hf":1,
                "hfp_ag":0,
                "gatt_client":0,
                "gatt_server":0
        },
        "a2dp_sink":{
                "device":"default",
                "buffer_time":400000,
                "period_time":100000
        },
        "a2dp_source":{
                "hci_index":0,
                "DEV":"00:00:00:00:00:00",
                "DELAY":20000
        },
        "hfp_pcm":{
                "rate":16000,
                "phone_to_dev_cap":"hw:snddaudio2",
                "phone_to_dev_play":"default",
                "dev_to_phone_cap":"CaptureMic",
                "dev_to_phone_play":"hw:snddaudio2"
        }
}
```

图 4-17: 检查声卡

检查平台的声卡类型和播放参数，默认是选择 default。可以先使用 aplay 命令单独播放播放 44.1Khz 和 48Khz 的音频源文件，可以初步排除是音频驱动异常的原因。

方法 2：检查 HCI log

如下图所示，查看 A2DP Sink 一栏，确定是否有音频数据，进而判断手机端有传过来音频，排除掉固件端以下的问题。

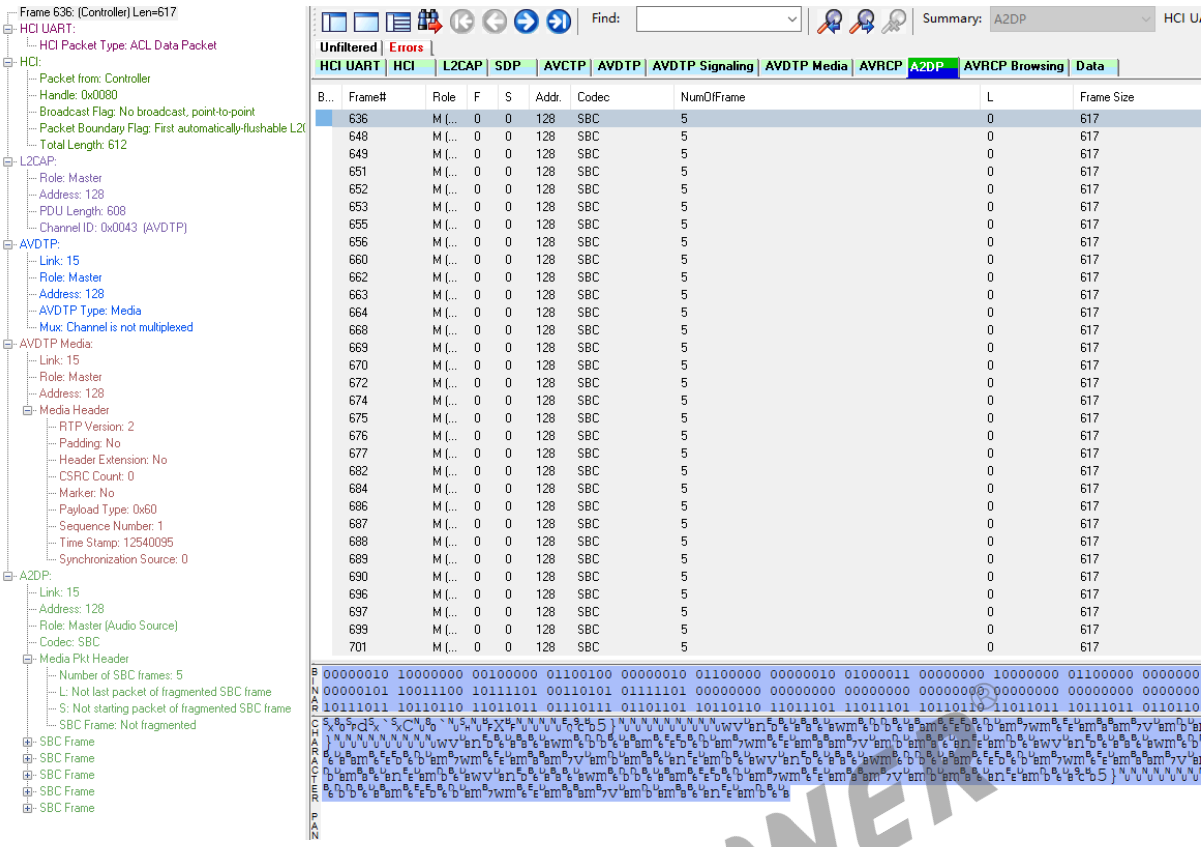


图 4-18: 确定是否有音频数据

4.3.3.2 播放卡顿

通用排查指南

按照通用排查指南检查。

音频驱动

蓝牙的播放需要通过音频驱动来实现声音输出，因此可以先初步判断音频驱动是否正常。通过 aplay 来

播放 44.1KHz 和 48KHz 的音频源文件，可以初步排除是音频驱动异常的原因。

btmanager 层

（1）使用 bluealsa 进行播放验证是否存在卡顿：

```
bt_test -p avrcp &
bluealsa -p a2dp-sink &
bluealsa-aplay 00:00:00:00:00:00 -vv &
```

然后就可以使用手机连接播放对比连。

（2）检查速率

检查写入声卡的速率，btmanager 提供有打开速率统计的 Debug 节点。

打开命令：

```
echo mask =4 > /tmp/log_btmg_io
```

关闭命令：

```
echo mask =0 > /tmp/log_btmg_io
```

具体使用方式参考 A2DP Source 蓝牙音乐-btmanager 层排查章节。

4.4 SPP

4.4.1 案例 1: 设备取消配对失败

问题描述：

样机作为 SPP Client，对已连接的设备进行取消配对，出现异常，取消配对失败。

问题背景：

平台：R818

模组：RTL8723DS

问题表现：

取消配对失败。

问题分析：

1. 检查在连接之前，是否已经对设备已经配对过。
2. 确认连设备连接之前尚未使用 bt_manager_pair() 进行配对。

根本原因：

目前版本 SPP 连接尚未包含配对。

解决办法：

第一次连接新设备，需要先通过 bt_manager_pair() 对设备进行配对。

只有配对完成后，才能进行取消配对操作。

5 蓝牙低功耗-BLE

5.1 GATT Server

5.1.1 案例 1: 概率出现无法广播问题

问题描述：

用户进行系统压测，出现使用手机无法扫描到 BLE 广播的问题。

问题背景：

平台：R328

模组：XR829

问题表现：

log 中有 “Failed to initialze ATT transport laye” 的异常打印。

问题分析：

1. 通过 log 找到相应的代码。

```
static void server_listen_cb(int fd, uint32_t events, void *user_data)
{
    gatt_server_t *server = user_data;
    int accept_fd;
    int mtu = 517;
....
    BTMG_DEBUG("accept_fd %d", accept_fd);
    server->fd = accept_fd;
    server->att = bt_att_new(accept_fd, false);
    if (!server->att) {
        BTMG_ERROR("Failed to initialze ATT transport layer");
        goto fail;
    }
....
```

出现 “Failed to initialze ATT transport layer” 的打印之后，GATT Server 必然初始化失败了，因此手机是不能扫到广播的。于是过滤 server_listen_cb 相关的打印：

```
08-05-084733:server_listen_cb(): accept_fd 44
08-05-084800:server_listen_cb(): accept_fd 17
08-05-084822:server_listen_cb(): accept_fd 17
```

```
08-05-084842:server_listen_cb(): accept_fd 17
08-05-084903:server_listen_cb(): accept_fd 17
08-05-084921:server_listen_cb(): accept_fd 54
08-05-084939:server_listen_cb(): accept_fd 55
08-05-084953:server_listen_cb(): accept_fd 57
08-05-085010:server_listen_cb(): accept_fd 58
08-05-085025:server_listen_cb(): accept_fd 60
08-05-085059:server_listen_cb(): accept_fd 60
08-05-085118:server_listen_cb(): accept_fd 61
....
....
08-05-091347:server_listen_cb(): accept_fd 117
08-05-091402:server_listen_cb(): accept_fd 126
08-05-091418:server_listen_cb(): accept_fd 125
08-05-091435:server_listen_cb(): accept_fd 128
08-05-091435:server_listen_cb() ERROR: Failed to initialze ATT transport layer
```

可以看到 accept_fd 在前面断开、连接的时候，accept_fd 一直都是 17，一段时间后数值就开始不断增加，直到增加至 128 后，就出现了 “Failed to initialze ATT transport layer” 的报错。这里看起来非常可疑，由于 BLE 正常打开关闭，连接断开均不会引起 accept_fd 数值增长，如果其它地方也使用到 socket，在使用后没有及时 close 掉，也会导致这个 fd 增加。

2. 发现 fd 的 128 临界条件与 mainloop_add_fd 有关系，可以查看 mainloop_add_fd 的实现，代码路径：

```
tina/out/.../compile_dir/target/bluez-5.54/src/shared/mainloop.c
```

3. 怀疑是用户的应用引起的，先让用户排查应用 socket。

客户排查应用并找到了 socket fd 的问题，并解决了增加的问题，再压测两天该 GATT Server 问题，问题没有出现了，结果跟我们猜想的预期一样。

根本原因：

用户应用的 socket fd 溢出，影响到协议栈。

解决办法：

修复 socket fd 溢出问题。

5.1.2 案例 2: iPhone 无法发现服务

问题描述：

苹果手机连上了测试样机，并发现了样机上所有 GATT Service。苹果手机与手机断开，用户修改了样机端 GATT Service 的内容，苹果手机再次连上测试样机后，无法正常发现服务。

问题背景：

平台：R328

模组：XR829

问题表现：

无法正常发现服务。

问题分析：

1. 使用 Android 手机对比测试，没发现有问题，因此怀疑是与 iPhone 兼容性的问题；
2. 使用 Android 手机安装 nrf connect 软件，并配置为 GATT Server，进行相同的验证，发现同样存在问题，所以怀疑 iPhone 端；
3. 通过网上查阅资料，更新服务器的服务后，iPhone 需要重启后才能正常发现服务。于是进行验证，确认如此。

根本原因：

与 iPhone 手机系统有关。

解决办法：

iPhone 需要重启后才能正常发现服务。

思路总结：

如果怀疑是兼容性问题，使用 Android 手机配置成 GATT Server 做对比。

5.1.3 案例 3: 通知指示失败

问题描述：

用户应用发送通知或者指示失败。

问题背景：

平台：R328

模组：XR829

问题表现：

通知或者指示发送失败。

问题分析：

1. 检查 GATT Service 的 Characteristic 是否已经添加 CCC，如下所示：

```
chr1.permissions = BT_GATT_PERM_READ | BT_GATT_PERM_WRITE;
chr1.properties = BT_GATT_CHAR_PROPERTY_READ | BT_GATT_CHAR_PROPERTY_WRITE |
        BT_GATT_CHAR_PROPERTY_NOTIFY;
chr1.svc_handle = service_handle;
chr1.uuid = TEST_CHAR_UUID1;
bt_manager_gatt_server_add_characteristic(&chr1);

desc1.permissions = BT_GATT_PERM_READ | BT_GATT_PERM_WRITE;
desc1.uuid = CCCD_UUID;
desc1.svc_handle = service_handle;
bt_manager_gatt_server_add_descriptor(&desc1);
```

2. 使用手机安装 nrf connect 软件，连接设备找到相应的 Characteristic 确认是否包含了 CCC（Client Characteristic Configuration）, 如下图所示：

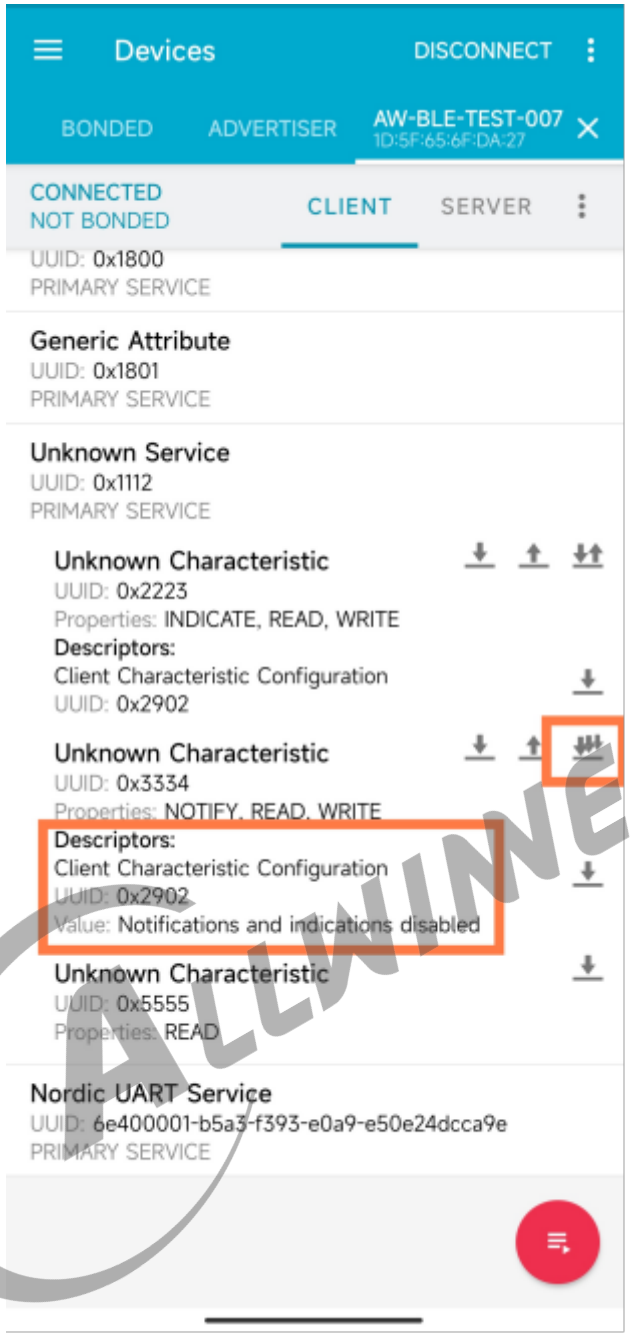


图 5-1: nRf 界面

看到是有 0x2902 这个 UUID，证明服务器端已经正常添加 CCC 了。但是看 Value 的值显示 NOTIFY 是 Disabled 的状态。

3. 尝试改变 CCC 的 Value，通过点击 “小箭头” 即可切换状态，如上面图片所示。
4. 客户修改应用使能后，可以正常通知。

根本原因：

GATT Client 端没有去使能 GATT Server 端的 CCC 配置。

解决办法：

GATT Client 端正确去使能 CCC。

思路总结：

1. Server 端的 Characteristic 需要添加 Client Characteristic Configuration descriptor（UUID=0X2902）；
2. Client 端需要主动去设置 Server 端 CCC 的值。

5.1.4 案例 4: BLE 名称设置失败

问题描述：

用户反馈调用 API 设置 BLE 广播名称无效。

问题背景：

平台：R328

模组：XR829

问题表现：

无法设置期望的 BLE 名称。

问题分析：

1. 一般 BLE 的名称是在广播数据里面设置的，如下图所示，最终调用 bt_manager_le_set_adv_data：

```
1: bt_gatt_server_demo.c
221
222 static int le_set_adv_data(const char *ble_name, uint16_t uuid)
223 {
224     int dd;
225     uint8_t manuf_len;
226     int index;
227     char advdata[31] = { 0 };
228     char uuid_buf[5] = { 0 };
229     int advdata_len = 0;
230
231     index = 0;
232
233     advdata[index] = 0x02; /* flag len */
234     advdata[index + 1] = 0x01; /* type for flag */
235     advdata[index + 2] = 0x1A; //0x05
236
237     index += advdata[index] + 1;
238
239     advdata[index] = strlen(ble_name) + 1; /* name len */
240     advdata[index + 1] = 0x09; /* type for local name */
241     int name_len;
242     name_len = strlen(ble_name);
243     strcpy(&(advdata[index + 2]), ble_name);
244     index += advdata[index] + 1;
245
246     advdata[index] = 0x03; /* uuid len */
247     advdata[index + 1] = 0x03; /* type for complete list of 16-bit uuid */
248     advdata[index + 2] = (char)(uuid & 0xFF);
249     advdata[index + 3] = (char)((uuid >> 8) & 0xFF);
250     index += advdata[index] + 1;
251
252     btmg_adv_data_t adv;
253     adv.data_len = index;
254     memcpy(adv.data, advdata, 31);
255
256     return bt_manager_le_set_adv_data(&adv);
257 }
```

图 5-2: 设置广播数据

2. 用户反馈此函数已经正常调用，没有正常下发，并且广播数据中有设备名称，使用手机 nrf_connect 软件扫描，扫出的名字会经常是 “BlueZ 5.54”。所以怀疑是内核驱动没有修改。
3. 因为之前有发现内核驱动会设置默认的 SCAN_RSP_DATA，所以让客户检查 SDK 使用的内核驱动代码，发现没有打上相关补丁。

根本原因：

内核驱动会设置默认的 SCAN_RSP_DATA，里面携带的扫描回复数据，名字是 “BlueZ 5.54”。

解决办法：

方法 1：修改内核代码。

```
diff --git a/net/bluetooth/hci_request.c b/net/bluetooth/hci_request.c
index 1015d9c8d97d..c7f2dac0907d 100644
--- a/net/bluetooth/hci_request.c
+++ b/net/bluetooth/hci_request.c
@@ -1075,7 +1075,7 @@ void __hci_req_update_scan_rsp_data(struct hci_request *req, u8 instance)

   cp.length = len;
-  hci_req_add(req, HCI_OP_LE_SET_SCAN_RSP_DATA, sizeof(cp), &cp);
+  // hci_req_add(req, HCI_OP_LE_SET_SCAN_RSP_DATA, sizeof(cp), &cp);
 }
 static u8 create_instance_adv_data(struct hci_dev *hdev, u8 instance, u8 *ptr)
@@ -1169,7 +1169,7 @@ void __hci_req_update_adv_data(struct hci_request *req, u8 instance)

   cp.length = len;

-  hci_req_add(req, HCI_OP_LE_SET_ADV_DATA, sizeof(cp), &cp);
+  // hci_req_add(req, HCI_OP_LE_SET_ADV_DATA, sizeof(cp), &cp);
 }
```

方法 2：修改扫描回复数据

调用 btmanager 的扫描回复数据接口，将广播数据也设置到扫描回复数据里面，覆盖内核默认设置的数据。

5.2 GATT Client

GATT Client 一般遇到的问题是连接、数据收发等问题。

5.2.1 连接问题

方法 1：使用手机连接对比验证对端设备是否有问题。

方法 2：检查 MAC 地址类型。

要确认对端 BLE Mac 地址类型是否与 API 设置的类型一致，只有设置的 MAC 地址类型一致，才能正常连接。

例如对端设备的 BLE MAC 地址类型设置为 PUBLIC 类型，而调用 bt_manager_gatt_client_connect() API 时设置的是 RANDOM 类型，这种情况不可能连接成功。

5.2.2 接收通知指示问题

检查应用层是否正常配置 GATT Server 端的 Client Characteristic Configuration Descriptor，需要使用 API：

```
int bt_manager_gatt_client_register_notify_indicate(int conn_id, int char_handle)
```

Notify 和 Indicate 都是使用该 API 注册。

著作权声明

商标声明

免责声明