|  |  |
| --- | --- |
| **密级状态：** 绝密( ) 秘密( ) 内部资料( ) 公开( √ )  Security Class: Top-Secret ( ) Secret ( ) Internal ( ) Public ( √ ) | |
| Rockchip\_Wifidisplay\_使用说明 Rockchip\_Wifidisplay\_Introduction | |
| （技术部，第二系统产品部）   |  |  |  | | --- | --- | --- | | **文件状态：**  **Status:**  [ ] 草稿  [ ] Draft  [ ] 正在修改  [ ] Modifying  [√ ] 正式发布  [ √ ] Released | **文件标识：**  **File No.:** | RK-SM-YF-250 | | **当前版本：**  **Current Version：** | V1.0 | | **作 者：**  **Author:** | 王剑辉  Wang Jianhui | | **完成日期：**  **Finish Date:** | 2020-04-03 | | **审 核：**  **Auditor:** | 邓训金、张崇松  Deng Xunjin/Zhang Chongsong | | **审核日期：**  **Finish Date:** | 2020-04-03 |   (Technical Department, R & D Dept. II） | |
|  |  |
| 福州瑞芯微电子股份有限公司  Fuzhou Rockchip Electronics Co., Ltd. |  |

**免责声明**

本文档按“现状”提供，福州瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

**Disclaimer**

This document is provided “as is” and Fuzhou Rockchip Electronics Co. Ltd (“the company”) makes no express or implied statement or warranty as to the accuracy, reliability, completeness, merchantability, specific purpose and non-infringement of any statement, information and contents of the document. This document is for reference only.

This document may be updated without any notification due to product version upgrades or other reasons.

**商标声明**

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

**Brand Statement**

Rockchip, RockchipTM icon, Rockchip and other Rockchip trademarks are trademarks of Fuzhou Rockchip Electronics Co., Ltd., and are owned by Fuzhou Rockchip Electronics Co., Ltd.

All other trademarks or registered trademarks mentioned in this document are owned by their respective owners.

**版权所有 © 2019 福州瑞芯微电子股份有限公司**

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

**Copyright © 2019 Fuzhou Rockchip Electronics Co., Ltd.**

Beyond reasonable use, without the written permission, any unit or individual shall not extract or copy part or all of the content of this document, and shall not spread in any form.

福州瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： [www.rock-chips.com](http://www.rock-chips.com)

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： [fae@rock-chips.com](mailto:fae@rock-chips.com)

Fuzhou Rockchip Electronics Co., Ltd.

Address: No. 18 Building, A District, No.89,software Boulevard Fuzhou,Fujian,PRC

Website: [www.rock-chips.com](http://www.rock-chips.com)

Customer service tel.: +86-4007-700-590

Customer service fax: +86-591-83951833

Customer service e-mail: [fae@rock-chips.com](mailto:fae@rock-chips.com)

**版 本 历 史 Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号  Version no. | 作者  Author | 修改日期  Revision Date | 修改说明  Revision description | 备注  Remark |
| V1.0 | 王剑辉  Wang Jianhui | 2020-04-03 | 发布初版  Initial version release |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**目 录Contents**

[Rockchip\_Wifidisplay\_使用说明 1](#_Toc20083)

[Rockchip\_Wifidisplay\_Introduction 1](#_Toc10774)

[前 言 1](#_Toc16914)

[1 Wifidisplay（Miracast）理论知识简介 1](#_Toc15232)

[1.1 Wifidisplay功能框架图 1](#_Toc23091)

[1.2 常见名称介绍 1](#_Toc14148)

[1.3 Wifidisplay协议流程 2](#_Toc27392)

[1.4 Wifidisplay source显示框架 5](#_Toc19375)

[1.5 Wifidisplay Android源码 5](#_Toc24558)

[1.6 Android WifiDisplay实现 6](#_Toc17024)

[1.6.1 Source端实现 6](#_Toc13897)

[1.6.2 Sink端实现 8](#_Toc23579)

[1.7 Wifidisplay使用说明 11](#_Toc11281)

[2 Wifidisplay Debug介绍 11](#_Toc26910)

[2.1 iperf测试P2P吞吐率和丢包率 11](#_Toc6038)

[2.2 视频出现卡顿，利用Wireshark分析pcap包 12](#_Toc22019)

[2.2.1 TCPDUMP抓包 12](#_Toc3379)

[2.2.2 利用Wireshark分析pcap 包 12](#_Toc24687)

[2.2.3 Wifidisplay log关键字 15](#_Toc19395)

[3 FAQ 15](#_Toc11359)

[3.1 Wifidisplay无法投屏到Win10上 15](#_Toc28089)

[3.2 投屏画面卡顿，经常出现花屏问题 16](#_Toc24400)

[3.3 Wifidisplay source端设备设定默认分辨率 16](#_Toc3080)

[3.4 Wifidisplay sink端设定支持的分辨率 20](#_Toc23062)

[3.5 Wifidisplay Sink端主动请求I帧 22](#_Toc10558)

[3.6 Wifidisplay Source端设置I帧间隔时间 23](#_Toc11775)

[3.7 Wifidisplay去除HDCP加密补丁 24](#_Toc11902)

[3.8 Android9.0系统加wifidisplay sink功能弹提示框错误 25](#_Toc4209)

[3.9 Wifidisplay sink延时优化 25](#_Toc20793)

[3.10 Android9.0和Android10.0 Wifidisplay source功能 36](#_Toc26669)

[3.11 Wifidisplay Sink不能显示Netflix播放内容 36](#_Toc12254)

[3.12 Wifidisplay HDCP2.x key烧写 41](#_Toc10703)

[3.13 Wifidisplay投射主副屏选择 41](#_Toc9433)

[3.14 Wifidisplay投屏显示问题 42](#_Toc9556)

[3.15 Wifidisplay连续投屏12.4h，画面出卡顿 42](#_Toc2700)

# 前 言

**概述**

本文档主要介绍Wifidisplay原理，以及Rockchip平台 Android版本无线投屏（Wifidisplay）问题排查、debug手段，旨在帮助软件开发工程师对Rockchip平台Android版本的wifidisplay模块debug、性能优化等。

**读者对象**

本文档主要适用于以下工程师：

技术支持工程师

软件开发工程师

# Wifidisplay（Miracast）理论知识简介

## Wifidisplay功能框架图

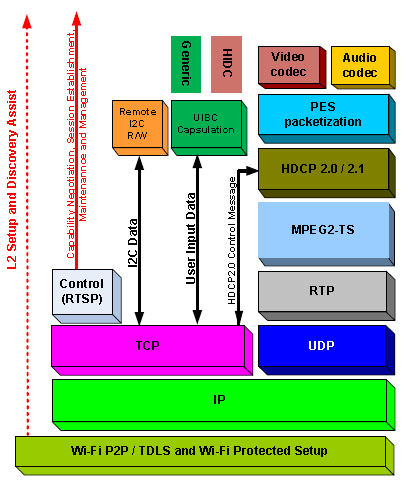


图1 WFD涉及的技术及协议框图

## 常见名称介绍

Miracast是Wi-Fi联盟（Wi-Fi Alliance）对支持Wi-Fi Display功能的设备的认证名称。通过Miracast认证的设备将在最大程度内保持对Wi-Fi Display功能的支持和兼容。

Wifidisplay（WFD）是一种基于Wi-Fi Direct技术的设备互联并实时共享屏幕内容的解决方案，它拓宽了Wi-Fi技术的覆盖范围，通过设备之间的直接连接，给用户带来新的连接和共享体验。接收端设备可通过发现、配对、连接并呈现源设备的多媒体内容。

UIBC是User Input Back Channel，即反向控制功能，sink端设备控制source。

HDCP是高带宽数字内容保护。

Source是能提供多媒体内容传输的设备。

Sink是能接受多媒体内容并将其呈现的设备。

Session是一个WFD在传输和接收设备之间连接。

TDLS是Tunneled Direct Link Setup一项802.11协议，在两个连接到同一个Ap的客户端设备之间建立的直接连接，Wifidisplay2.0协议才支持这种连接方式，目前RK平台的Wifidisplay是1.0版本的协议，所以不支持TDLS传输方式，都是用Wifi P2P传输方式。

Wi-Fi Direct是一种P2P的无线互联技术，它所建立的网络（piconet）是一种改进型的ad hoc网络，采用无线通讯模式。

## Wifidisplay协议流程

建立WifiDisplay主要步骤如下：

1. WFD Device Discovery（WFD设备发现）

2. WFD Service Discovery (Optional)（WFD服务发现（可选））

3. Device Selection（设备选择）

4. WFD Connection Setup（WFD连接）

5. WFD Capability Negotiation（WFD能力协商）

6. WFD Session Establishment（WFD会话建立）

7. User Input Back Channel Setup (Optional)（UIBC反向控制）

8. Link Content Protection Setup (Optional)（内容保护，即数据加密）

9. Payload Control（负载控制）

10. WFD Source and WFD Sink standby (Optional)

11. WFD Session Teardown（会话终止）

WFD设备通过wifiP2P连接后，Sink端与Source端建立TCP连接，Sink端为Client而Source端为Server。默认端口为7236，执行的协议为RTSP协议。建立连接后进行RTSP协商。步骤6，协商成功后建立会话；步骤7，UIBC通道建立，用于Sink端反向控制Source端，该步骤为可选实现；步骤8，对传输的内容做加密保护（HDCP），步骤9，开始音频及视频流的传输与控制，Payload Control：传输过程中，设备可根据无线信号的强弱，甚至设备的电量状况来动态调整传输数据和格式。可调整的内容包括压缩率，视音频格式，分辨率等内容。步骤11，会话终止。

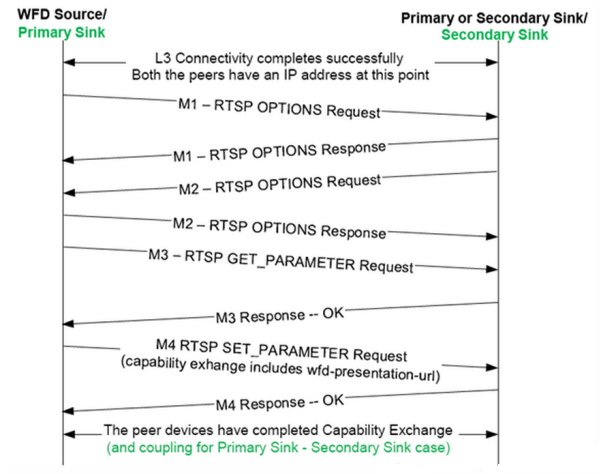


图 2 会话建立及协商过程图

RTSP M1和M2主要协商Source和Sink都支持的RTSP methods。RTSP M3和M4主要协商Source和Sink在会话中使用的参数。

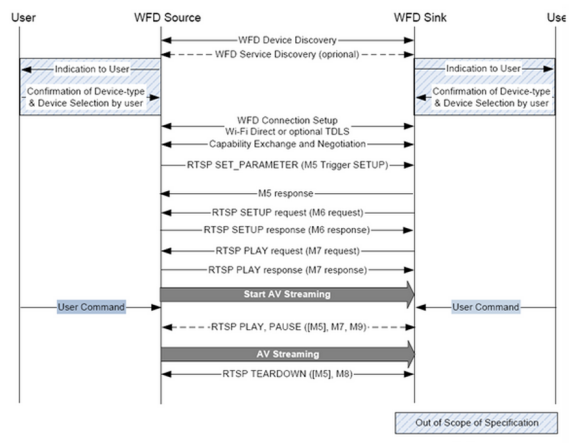


图3 RTSP协议控制图

RTSP协议控制中主要有以下几种状态SETUP、PLAY、PAUSE、TEARDOWN，当Wifidisplay source和Wifidisplay sink之间成功地完成了RTSP M7请求和响应消息的交换时，source和sink就建立了WFD会话。

对于WifiDisplay会话管理有以下模型可供参考，该结构大致分为四个层次，UI、Session Policy Management、协议实现层及基于Wifi的网络传输层。在协议实现层中主要分为几个模块WFD Discovery、WFD Link Establishment、UIBC、Capability Negotiation、Session/Stream Control等。

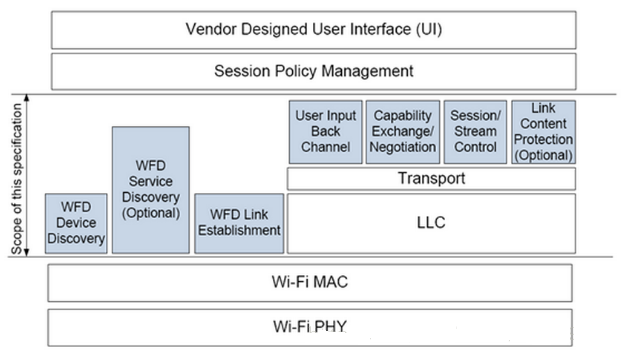


图4 WFD设备会话管理的模型

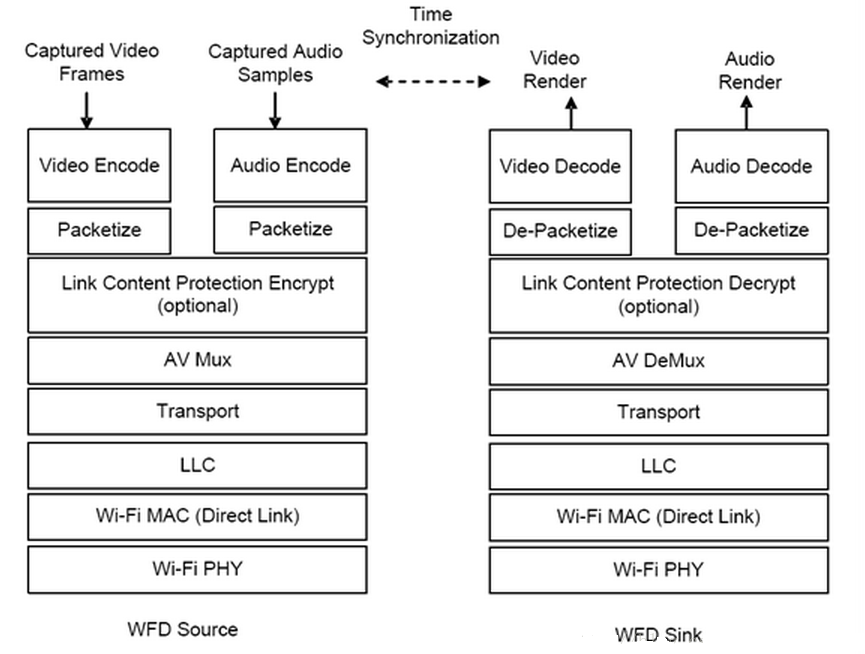


图5 音频及视频流控制模型

实时流协议RTSP是一个应用层协议，用于控制具有实时特性的数据（例如多媒体流）的传送。RTSP协议一般与RTP/RTCP和RSVP等底层协议一起协同工作，提供基于Internet的整套的流服务。它可以选择发送通道（例如：UDP、组播UDP和TCP）和基于RTP的发送机制。它可以应用于组播和点播。RTP, RTCP, RSVP定义如下：

1. 实时传输协议RTP(Real-time Transport protocol)

2. 实时传输控制协议RTCP(Real-time Transport Control protocol)

3. 实时流协议RTSP(Real Time Streaming protocol)

4. 资源预留协议RSVP(Resource Reserve Protocol)

客户端与服务器运行实时流控制协议RTSP，以对该流进行各种VCR控制信号的交换，如播放（PLAY）、停止（PAUSE）、快进、快退等。当服务完毕，客户端提出拆线（TEARDOWN）请求。服务器使用RTP/UDP协议将媒体数据传输给客户端，一旦数据抵达客户端，客户端应用程序即可播放输出。在流式传输中，使用RTP/RTCP/UDP和RTSP/TCP两种不同的通信协议在客户端和服务器间建立联系。

[LLC层](https://www.baidu.com/s?wd=LLC%E5%AD%90%E5%B1%82&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)是由传输驱动程序实现的，主要功能为传输可靠性保障和控制，数据包的分段与重组，数据包的顺序传输。

## Wifidisplay source显示框架

为了实现WifiDisplay，Google在Android现有显示系统的基础上加入虚拟设备的支持，下图给出了Android显示系统的架构图。

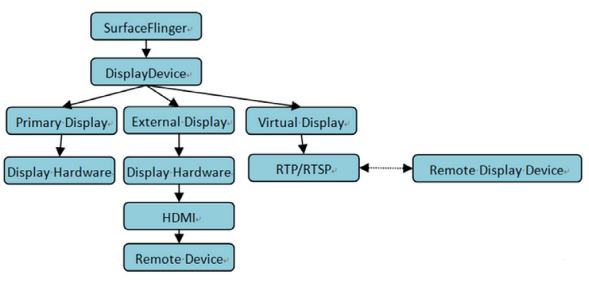


图6 DisplayDevice显示架构图

## Wifidisplay Android源码

涉及Wifidisplay模块的源码路径（以Android10.0为例）：

**Display Manager**

framework/base/services/core/java/com/android/server/display/

WifiDisplayController.java

WifiDisplayAdapter.java

PersistentDataStore.java

DisplayManagerService.java

**RTSP**

frameworks/av/media/libstagefright/rtsp

**MediaRouter**

frameworks/base/media/java/android/media/MediaRouter.java

**Wi-Fi P2P**

frameworks/base/wifi/java/android/net/wifi/p2p/

**Wi-Fi Display source端逻辑代码**

frameworks/av/media/libstagefright/wifi-display

WifiDisplaySource.cpp--Source端的协议交互

PlaybackSession.cpp--会话过程管理

MediaPuller.cpp--镜像媒体读取

Converter.cpp--编码

TSPacketizer.cpp--TS打包

Sender.cpp--RTP打包发送

**Wi-Fi Display sink端app代码（需要发邮件找fae窗口申请）**

WifiDisplaySink.cpp--负责SINK端的协议

RTPSink.cpp--RTP接收

TunnelRenderer.cpp--镜像数据的呈现

## Android WifiDisplay实现

### Source端实现

当用户点击了optionMenu中enable wifi display选项时，会触发相关的设备扫描及更新操作，在WifiDisplaySettings和WifiDisplayController都有注册ContentObserver来监控这个值的变化。触发设备扫描是在WifiDisplayController中通过updateWfdEnableState()进行的，最终通过WifiP2pManager.requestPeers来完成设备的扫描工作，获取扫描到的设备列表是在WifiDisplaySettings通过update(int changes)进行的。对于设备连接状态的管理主要通过updateConnection()来进行。由于设备的连接过程是一个异步过程，所以在设备操作相关的过程中会反复调用updateConnection()来判定设备状态及更新连接操作。

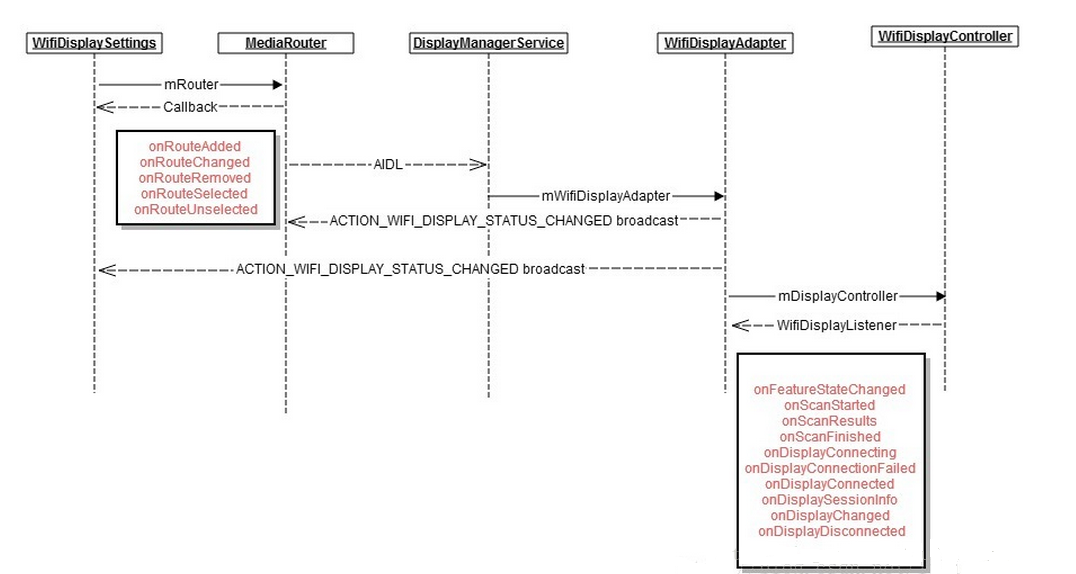


图7 设备发现流程图

设备列表更新及管理通过下面函数update实现：

private void update(int changes) {

boolean invalidateOptions = false;

// Update settings.

if ((changes & CHANGE\_SETTINGS) != 0) {

mWifiDisplayOnSetting = Settings.Global.getInt(getContentResolver(),

Settings.Global.WIFI\_DISPLAY\_ON, 0) != 0;

mWifiDisplayCertificationOn = Settings.Global.getInt(getContentResolver(),

Settings.Global.WIFI\_DISPLAY\_CERTIFICATION\_ON, 0) != 0;

mWpsConfig = Settings.Global.getInt(getContentResolver(),

Settings.Global.WIFI\_DISPLAY\_WPS\_CONFIG, WpsInfo.INVALID);

// The wifi display enabled setting may have changed.

invalidateOptions = true;

}

// Update wifi display state.

if ((changes & CHANGE\_WIFI\_DISPLAY\_STATUS) != 0) {

mWifiDisplayStatus = mDisplayManager.getWifiDisplayStatus();

// The wifi display feature state may have changed.

invalidateOptions = true;

}

// Rebuild the routes.

final PreferenceScreen preferenceScreen = getPreferenceScreen();

preferenceScreen.removeAll();

// Add all known remote display routes.

final int routeCount = mRouter.getRouteCount();

for (int i = 0; i < routeCount; i++) {

MediaRouter.RouteInfo route = mRouter.getRouteAt(i);

if (route.matchesTypes(MediaRouter.ROUTE\_TYPE\_REMOTE\_DISPLAY)) {

preferenceScreen.addPreference(createRoutePreference(route));

}

}

// Additional features for wifi display routes.

if (mWifiDisplayStatus != null

&& mWifiDisplayStatus.getFeatureState() == WifiDisplayStatus.FEATURE\_STATE\_ON) {

// Add all unpaired wifi displays.

for (WifiDisplay display : mWifiDisplayStatus.getDisplays()) {

if (!display.isRemembered() && display.isAvailable()

&& !display.equals(mWifiDisplayStatus.getActiveDisplay())) {

preferenceScreen.addPreference(

new UnpairedWifiDisplayPreference(getActivity(), display));

}

}

// Add the certification menu if enabled in developer options.

if (mWifiDisplayCertificationOn) {

buildCertificationMenu(preferenceScreen);

}

}

// Invalidate menu options if needed.

if (invalidateOptions) {

getActivity().invalidateOptionsMenu();

}

}

### Sink端实现

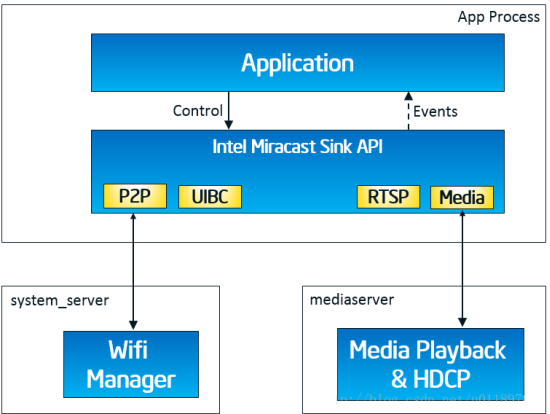


图8 设备发现流程图

图8给出了sink端的实现框架图，从框架图可以看出APP主要和Sink API交互，Sink API和框架服务中的Wifi server及mediaserver交互，APP通过Control interface进行WFD相关的控制操作，底层状态的接收则通过Events interface，也就是一些相关的回调方法来处理。

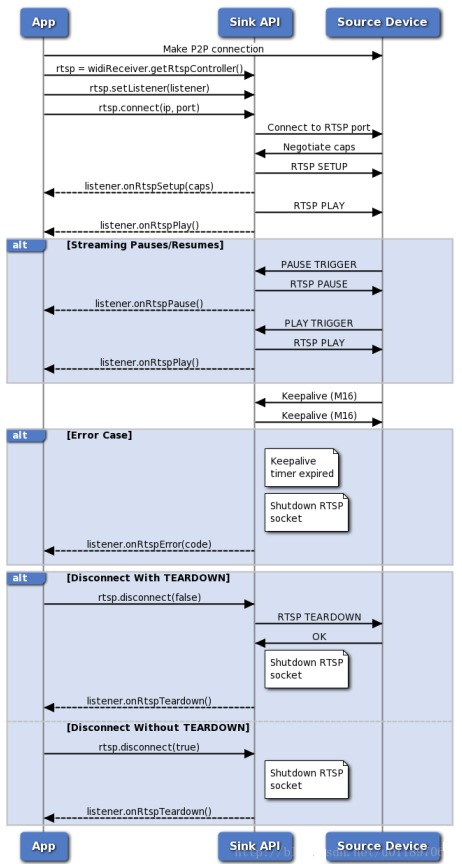


图9 RTSP会话流程图

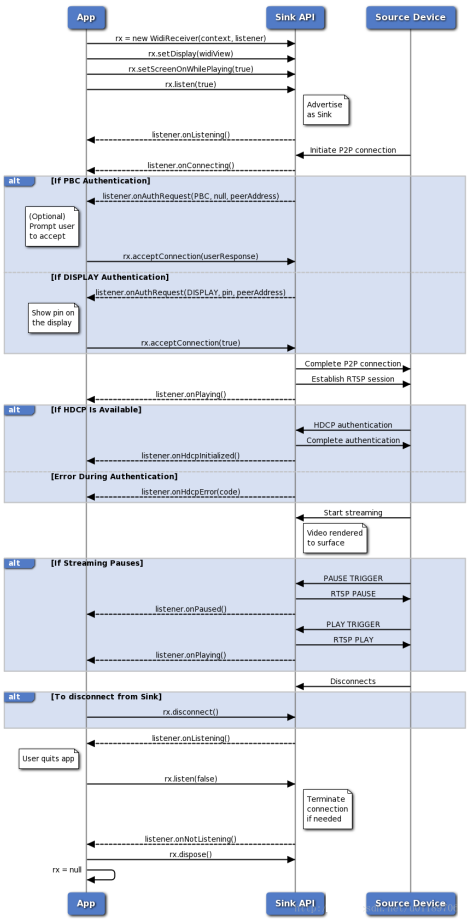


图10 WFD控制流程图

图9给出了Intel实现的sink端的RTSP会话管理流程图，RTSP的协议实现主要通过C++实现，对于协商后相关的状态反馈通过回调函数完成，如果想进一步了解相关的流程，请查看相关的代码。

图10给出了WFD会话管理的流程图，WFD中除了RTSP的实现，还包括连接认证（Connection Auth）、视频流加密及解密(HDCP)、UIBC实现等。

## Wifidisplay使用说明

使用说明分为source端和sink端。

wifidisplay source端源码集成到系统SDK里面，目前所有的RK平台Android SDK源码都带有wifidisplay source功能。使用方法：首先准备一台带有wifidisplay sink功能的电视或者带有sink功能的RK平板，然后打开原生Settings->设备->显示->投射（Android6.0版本），打开原生Settings -> Connected Devices -> Cast -> Enable wireless display（Android8.1版本）。

wifidisplay sink端是集成一个单独APK，默认SDK上没有源码，需要源码的话可以在ftp://www.rockchip.com下载，然后放SDK源码package/apps目录下面编译，烧写固件，打开WifiDisplay.apk。

# Wifidisplay Debug介绍

## iperf测试P2P吞吐率和丢包率

首先确认RK平台的设备是用于sink（接收）端设备还是用于source（发送）端设备。用于做sink端设备时，需要测试设备的下行吞吐率和丢包率；用于做source端设备时，需要测试设备的上行吞吐率和丢包率。下面分别说明设备吞吐率和丢包率测试方法和步骤：

RK平台设备做sink，其他平台手机做source（如nexus4）：

1. 两端设备都安装好iperf.apk
2. 两端设备wifi direct连接
3. 打开iperf，在RK平台设备端（sink端）输入 -u -s -i 1 -w 256K，然后选择“打开”按钮；在手机（source）端的iperf窗口中输入./iperf -u -c 10.255.255.251 -b 10M -i 1 -w 256K -t 23123

备注：10.255.255.251是RK平台（sink）端的p2p直连状态下的ip地址，实际测试过程中，替换使用。

1. 在手机（source）端点击“打开”按钮后，可在RK平台端看到一些report，最后一项是丢包率。假如两个设备靠得很近，正常是不会出现丢包的（低概率出现0.1%以下丢包属于正常），丢包严重的话，基本可以确认是Wifi模组硬件相关问题，需要排查下Wifi射频参数是否校准正确，Wifi天线是否正常匹配，环境是否有视频干扰等。

RK平台设备做source，其他平台电视做sink（如小米电视）：

1. 两端设备都安装好iperf.apk
2. 两端设备wifi direct连接
3. 打开iperf，在其他平台电视端（sink端）输入 -u -s -i 1 -w 256K，然后选择“打开”按钮；在RK设备（source）端的iperf窗口中输入./iperf -u -c 10.255.255.251 -b 10M -i 1 -w 256K -t 23123

备注：10.255.255.251是其他平台电视（sink）端的p2p直连状态下的ip地址，实际测试过程中，替换使用。

4）在RK设备（source）端点击“打开”按钮后，可在其他平台电视端看到一些report，最后一项是丢包率。假如两个设备靠得很近，正常是不会出现丢包的（低概率出现0.1%以下丢包属于正常），丢包严重的话，基本可以确认是Wifi模组硬件相关问题，需要排查下Wifi射频参数是否校准正确，Wifi天线是否正常匹配，环境是否有视频干扰等。

## 视频出现卡顿，利用Wireshark分析pcap包

### TCPDUMP抓包

RK平台设备，adb shell进入机器终端：

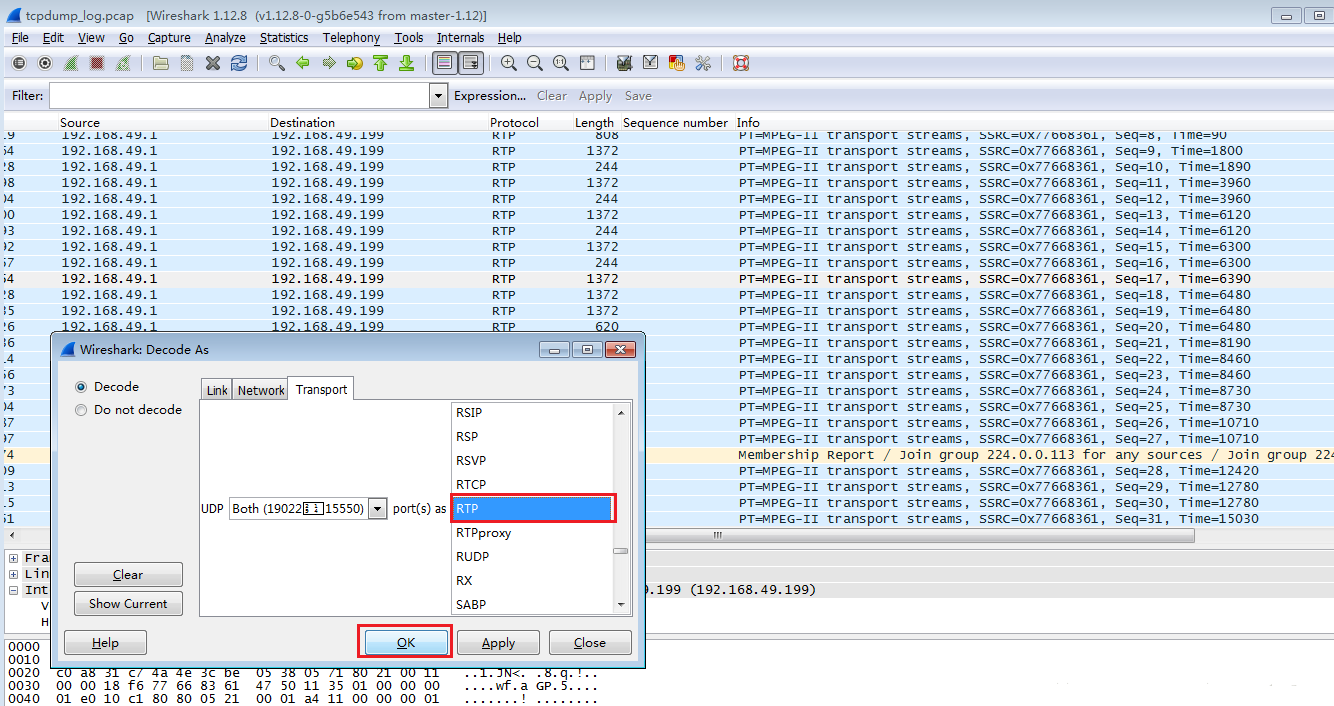
tcpdump -i any -s 0 -w /data/tcpdump\_log.pcap

注意–i any不能改成特定的interface, 因为p2p interface是动态生成的，有可能是p2p-p2p0-0、p2p-p2p0-1、p2p-p2p0-2等，抓取的pcap包尽量涉及从发送投屏请求到投屏图像显示一段时间，当分析投屏卡顿问题时，需要抓取到卡顿过程的pcap包。

### 利用Wireshark分析pcap 包

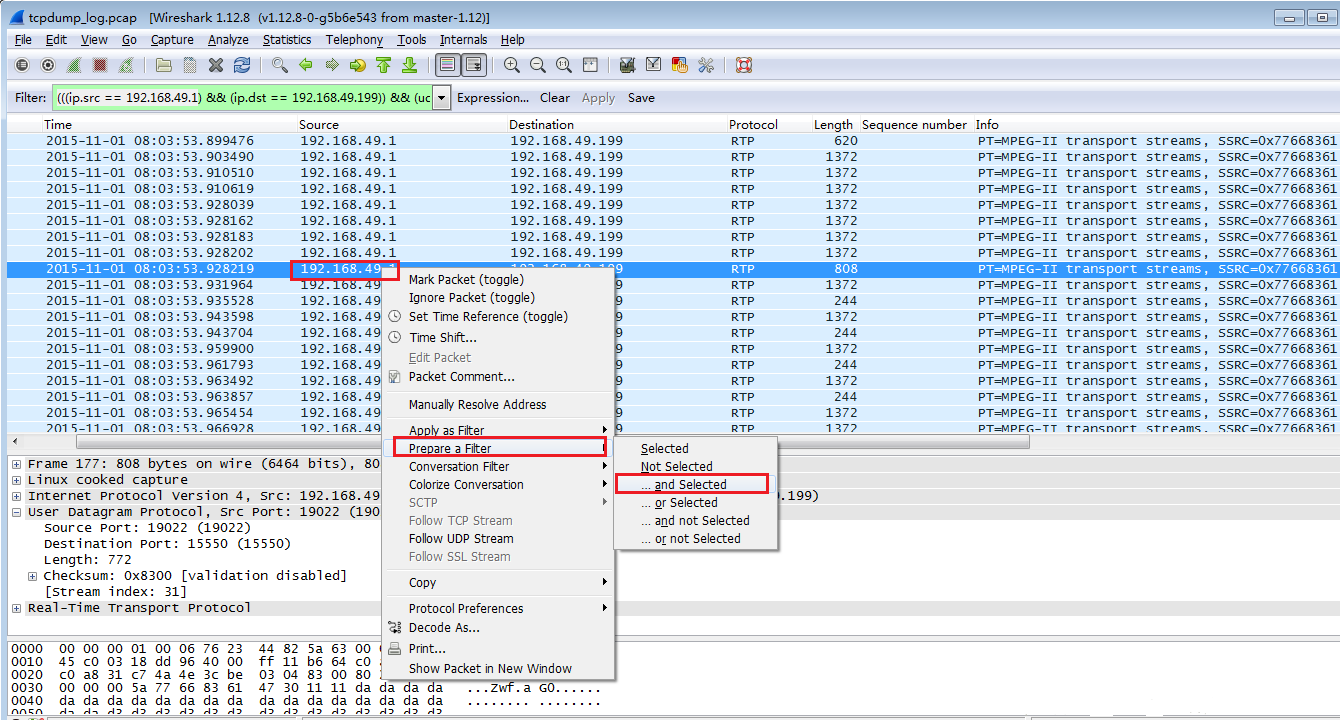
**下面分析一个正常的pcap包：**

1）点击目标UDP包，右键选择Decode As，将UDP decode成RTP包



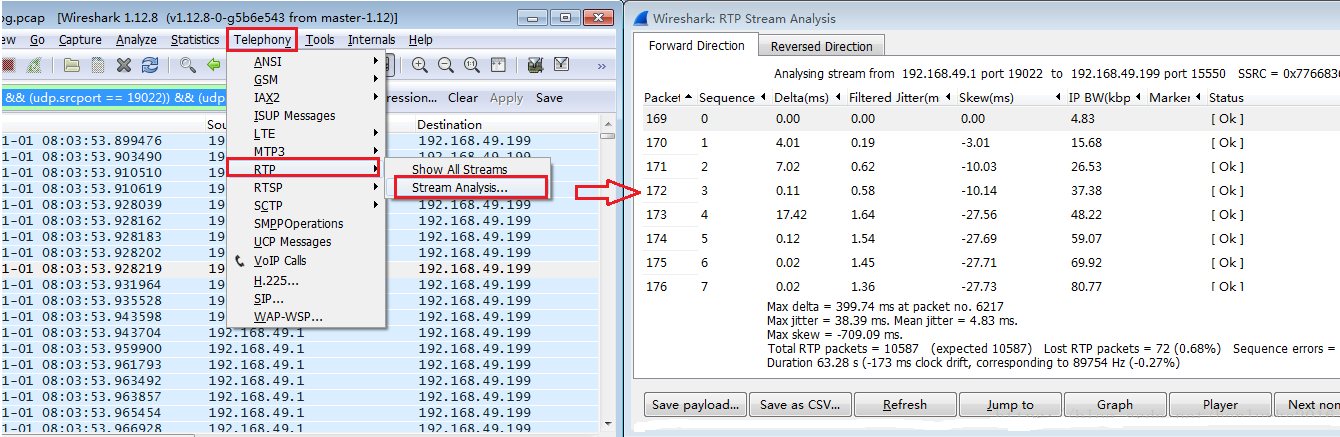
2）过滤RTP包，过滤条件为IP & PORT，通过Prepare a Filter选择

如这次的过滤条件为：(((ip.src == 192.168.49.1) && (ip.dst == 192.168.49.199)) && (udp.srcport == 19022)) && (udp.dstport == 15550)



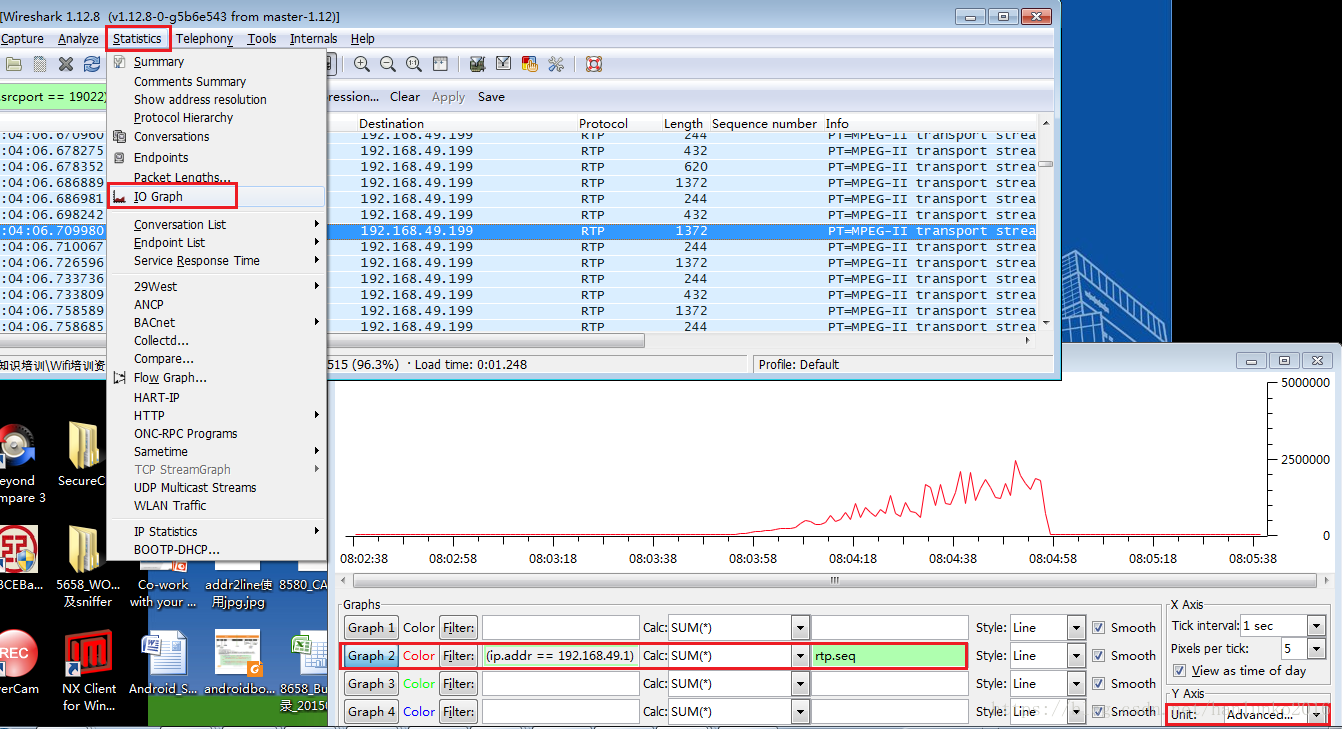
3）对过滤后的RTP包进行分析：Stream Analysis

可以从分析结果看到丢包率仅为0.68%,  乱序只有3次，播放应该是很流畅的



1. 通过IO Graph分析数据包丢失和延迟

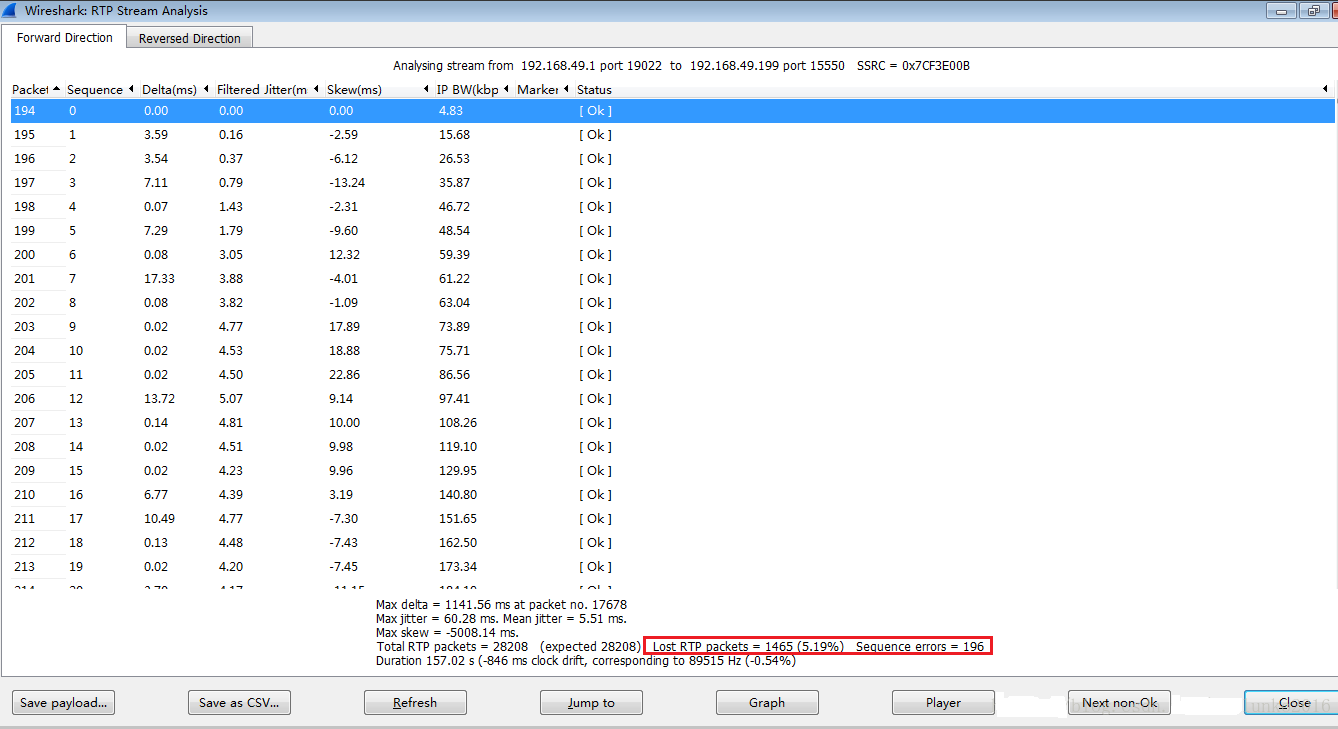
我们要使用函数功能时，须设置Y Axis Unit为Advanced，过滤条件：(ip.addr == 192.168.49.1) and (ip.addr == 192.168.49.199)，函数：SUM(\*)，统计字段：rtp.seq，从下图的波形看起来还是比较平滑，没有出现明显的峰值和下降。



**下面分析一个卡顿严重的pcap包：**

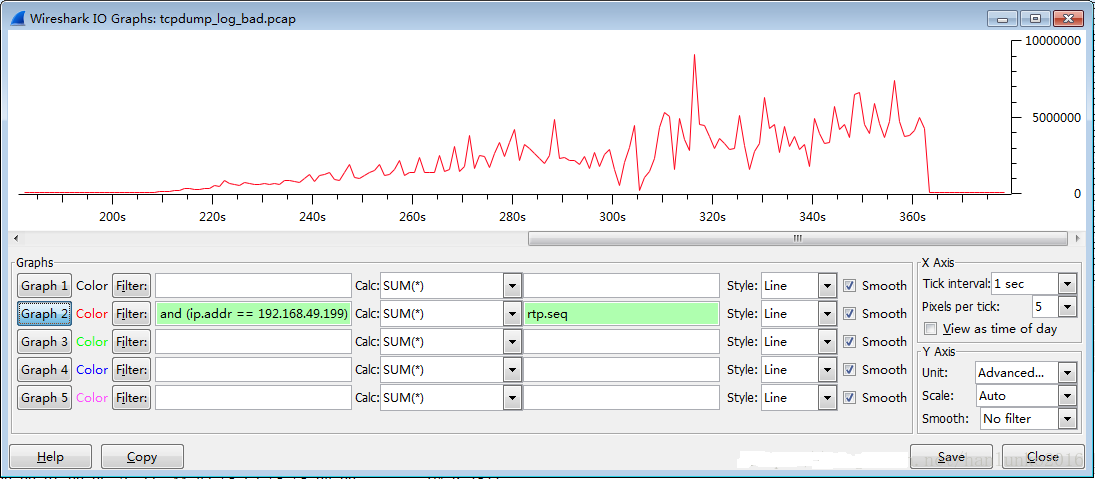
步骤1）和2）和上面分析的正常的视频步骤相同，请参考上面的分析步骤。

3）对过滤后的RTP包进行分析：Stream Analysis



4）通过IO Graph分析数据包丢失和延迟

我们要使用函数功能时，须设置Y Axis Unit为Advanced，过滤条件：(ip.addr == 192.168.49.1) and (ip.addr == 192.168.49.199)，函数：SUM(\*)，统计字段：rtp.seq，从下图看出现了若干明显的峰值和下降，说明丢包的情况比较严重。



### Wifidisplay log关键字

log太多时，可以搜索以下关键字，从这句log开始source端已经打开wifidsplay功能：

ActivityManager: START u0 {flg=0x8000 cmp=com.android.settings/.Settings$DisplaySettingsActivity (has extras)}

# FAQ

## Wifidisplay无法投屏到Win10上

有些客户在开发过程中，发现RK平板无法投屏到Windows10系统上，可能是这两个原因导致，需要一个个去排查。

1. 客户的Windows10系统不支持1280x720@p24显示参数。
2. 客户的Windows10系统不支持HDCP加密。

diff --git a/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp b/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp  
index 0d2b7df..ded490d 100755  
--- a/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp  
+++ b/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp

//修改投屏分辨率  
@ -78,11 +78,11 @ WifiDisplaySource::WifiDisplaySource(  
mSupportedSourceVideoFormats.disableAll();

mSupportedSourceVideoFormats.setNativeResolution(  
- VideoFormats::RESOLUTION\_CEA, 15); // 1280x720 p24  
+ VideoFormats::RESOLUTION\_CEA, 5); // 1280x720 p24

// Enable all resolutions up to 1280x720p24  
 mSupportedSourceVideoFormats.enableResolutionUpto(  
- VideoFormats::RESOLUTION\_CEA, 15,  
+ VideoFormats::RESOLUTION\_CEA, 5,  
 VideoFormats::PROFILE\_CHP, // Constrained High Profile  
 VideoFormats::LEVEL\_32); // Level 3.2  
 }

//关掉HDCP加密  
@ -591,7 +591,7 @ status\_t WifiDisplaySource::sendM1(int32\_t sessionID) {

status\_t WifiDisplaySource::sendM3(int32\_t sessionID) {  
 AString body =  
- "wfd\_content\_protection\r\n"   
+ /\*"wfd\_content\_protection\r\n"\*/  
 "wfd\_video\_formats\r\n"   
 "wfd\_audio\_codecs\r\n"

## 投屏画面卡顿，经常出现花屏问题

经常有客户报投屏过程中出现卡顿和花屏现象，可以用iperf工具先测试下Wifi模组的P2P吞吐率和丢包率，测试方法参考2.1节，排查下wifi模组射频参数是否校准，天线匹配是否正常。也可以用TCPDUMP抓包，用Wireshark分析pcap包，debug方法参考2.2节，确认wifi硬件是否正常，环境是否有干扰。假如没办法自己分析，可以把卡顿时的pcap包抓一份放redmine单号上，抓取命令：

tcpdump -i any -s 0 -w /data/tcpdump\_log.pcap

排除Wifi硬件射频参数校准和环境干扰还会卡顿，可以通过以下三种方法来优化投屏卡顿：

1. 降低投屏分辨率和帧率，具体实现方法如3.3和3.4节。
2. Sink端主动请求I帧，调整Source端I帧间隔时间，具体实现方法如3.5和3.6节。
3. 尽量使用5G频端wifi，5G带宽会比2.4G好。

## Wifidisplay source端设备设定默认分辨率

diff --git a/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp b/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp  
index 0d2b7df..ded490d 100755  
--- a/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp  
+++ b/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp  
@ -78,11 +78,11 @ WifiDisplaySource::WifiDisplaySource(  
mSupportedSourceVideoFormats.disableAll();

mSupportedSourceVideoFormats.setNativeResolution(  
- VideoFormats::RESOLUTION\_CEA, 15); // 1280x720 p24  
+ VideoFormats::RESOLUTION\_CEA, 5); // 1280x720 p24

// Enable all resolutions up to 1280x720p24  
 mSupportedSourceVideoFormats.enableResolutionUpto(  
- VideoFormats::RESOLUTION\_CEA, 15,  
+ VideoFormats::RESOLUTION\_CEA, 5,  
 VideoFormats::PROFILE\_CHP, // Constrained High Profile  
 VideoFormats::LEVEL\_32); // Level 3.2  
 }

Source端所有支持的分辨率数组如下：

frameworks/av/media/libstagefright/wifi-display/VideoFormats.cpp

const VideoFormats::config\_t VideoFormats::mResolutionTable[][32] = {

{

// CEA Resolutions

{ 640, 480, 60, false, 0, 0},

{ 720, 480, 60, false, 0, 0},

{ 720, 480, 60, true, 0, 0},

{ 720, 576, 50, false, 0, 0},

{ 720, 576, 50, true, 0, 0},

{ 1280, 720, 30, false, 0, 0},

{ 1280, 720, 60, false, 0, 0},

{ 1920, 1080, 30, false, 0, 0},

{ 1920, 1080, 60, false, 0, 0},

{ 1920, 1080, 60, true, 0, 0},

{ 1280, 720, 25, false, 0, 0},

{ 1280, 720, 50, false, 0, 0},

{ 1920, 1080, 25, false, 0, 0},

{ 1920, 1080, 50, false, 0, 0},

{ 1920, 1080, 50, true, 0, 0},

{ 1280, 720, 24, false, 0, 0},

{ 1920, 1080, 24, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

},

{

// VESA Resolutions

{ 800, 600, 30, false, 0, 0},

{ 800, 600, 60, false, 0, 0},

{ 1024, 768, 30, false, 0, 0},

{ 1024, 768, 60, false, 0, 0},

{ 1152, 864, 30, false, 0, 0},

{ 1152, 864, 60, false, 0, 0},

{ 1280, 768, 30, false, 0, 0},

{ 1280, 768, 60, false, 0, 0},

{ 1280, 800, 30, false, 0, 0},

{ 1280, 800, 60, false, 0, 0},

{ 1360, 768, 30, false, 0, 0},

{ 1360, 768, 60, false, 0, 0},

{ 1366, 768, 30, false, 0, 0},

{ 1366, 768, 60, false, 0, 0},

{ 1280, 1024, 30, false, 0, 0},

{ 1280, 1024, 60, false, 0, 0},

{ 1400, 1050, 30, false, 0, 0},

{ 1400, 1050, 60, false, 0, 0},

{ 1440, 900, 30, false, 0, 0},

{ 1440, 900, 60, false, 0, 0},

{ 1600, 900, 30, false, 0, 0},

{ 1600, 900, 60, false, 0, 0},

{ 1600, 1200, 30, false, 0, 0},

{ 1600, 1200, 60, false, 0, 0},

{ 1680, 1024, 30, false, 0, 0},

{ 1680, 1024, 60, false, 0, 0},

{ 1680, 1050, 30, false, 0, 0},

{ 1680, 1050, 60, false, 0, 0},

{ 1920, 1200, 30, false, 0, 0},

{ 1920, 1200, 60, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

},

{

// HH Resolutions

{ 800, 480, 30, false, 0, 0},

{ 800, 480, 60, false, 0, 0},

{ 854, 480, 30, false, 0, 0},

{ 854, 480, 60, false, 0, 0},

{ 864, 480, 30, false, 0, 0},

{ 864, 480, 60, false, 0, 0},

{ 640, 360, 30, false, 0, 0},

{ 640, 360, 60, false, 0, 0},

{ 960, 540, 30, false, 0, 0},

{ 960, 540, 60, false, 0, 0},

{ 848, 480, 30, false, 0, 0},

{ 848, 480, 60, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

{ 0, 0, 0, false, 0, 0},

}

};

## Wifidisplay sink端设定支持的分辨率

frameworks/av/media/libstagefright/wifi-display/sink/WifiDisplaySink.cpp

char parameterByte[3]; // for save the transfer byte modify by lance 2013.07.31

memset(parameterByte, 0, 3);

resolutionToParameterByte(parameterByte, displayWidth, displayHeight, info.fps);

AString wfdVideoFormatsString(parameterByte);

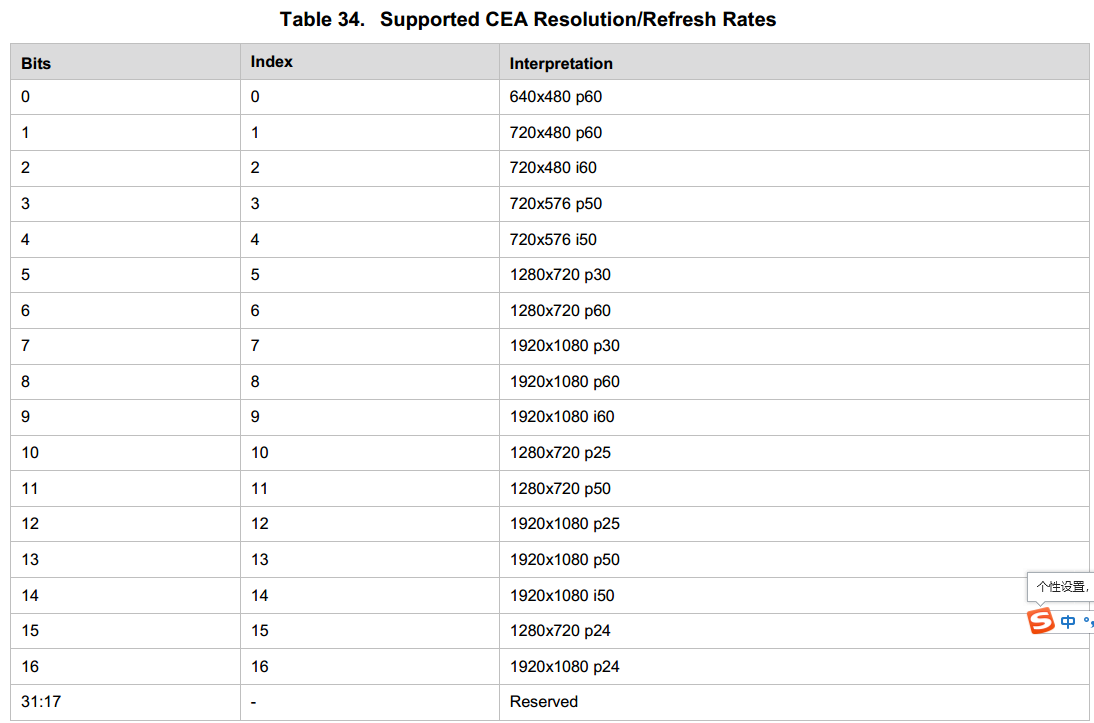
wfdVideoFormatsString.append(" 00 02 08 00008423 00000055 00000555 00 0000 0000 00 none none,01 08 00008423 00000055 00000555 00 0000 0000 00 none none");

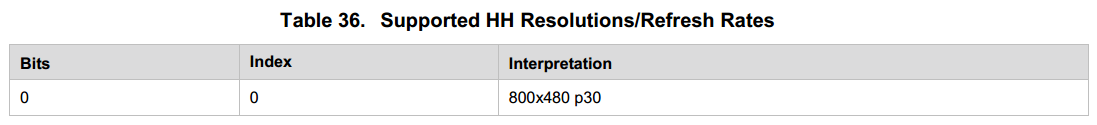
Android8.1以前wifi-display的sink源码是在framework/av/media/libstagefright目录下面，从Android8.1开始，我们把wifi-display的sink代码移到sink apk里面了。从上面代码看出Wifidisplay sink支持的分辨率设置主要是00008423 00000055 00000555。用2进制，从右到左，第0bit最左边。需要的就对应位置1，不需要就置0，然后转换成16进制。

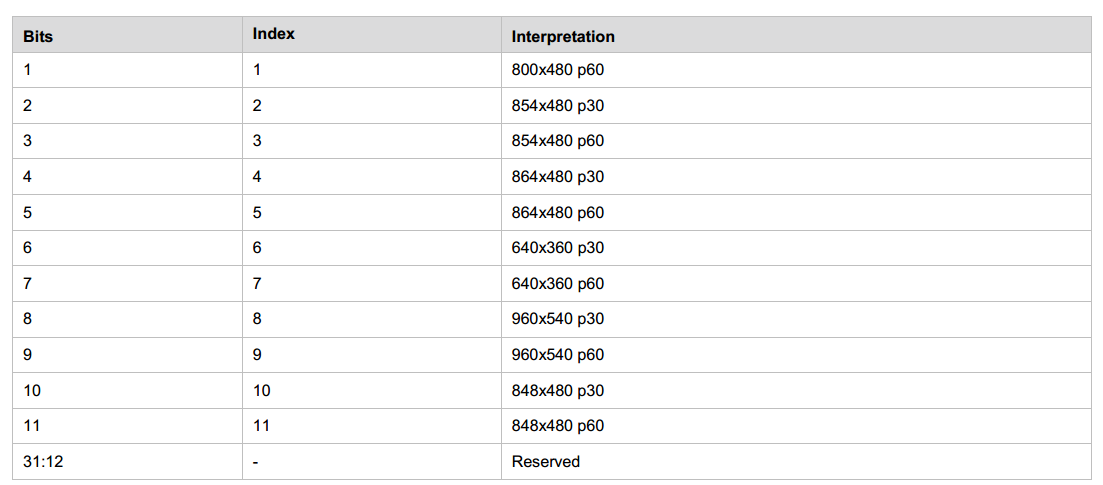
00008423是支持的CEA组分辨率。

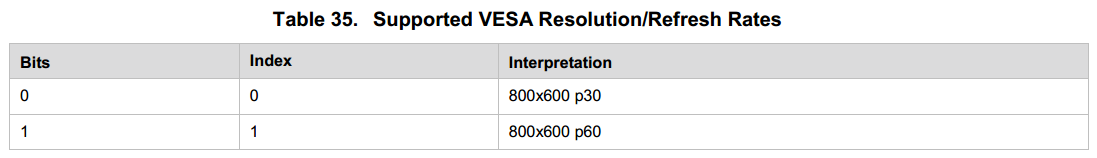
00000055 是支持的VESA组分辨率。

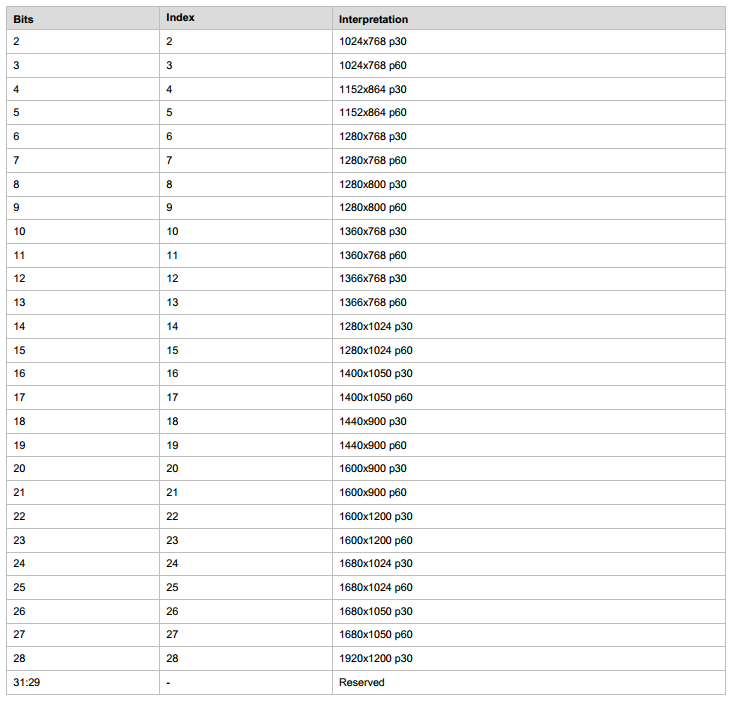
00000555是支持的HH组分辨率。











## Wifidisplay Sink端主动请求I帧

Android平台WifiDisplaySink.cpp文件里面已经封装了sendIDR接口，sendIDR是发送请求I帧的接口，当wifidisplay sink设备端出现画面卡顿时（码流数据丢包），可以发送请求I帧，快速恢复。sendIDR接口实现如下：

void WifiDisplaySink::sendIDR(int32\_t sessionID, const char \*uri)

{

    AString request = AStringPrintf("SET\_PARAMETER %s RTSP/1.0\r\n", uri);

    AppendCommonResponse(&request, mNextCSeq);

    request.append("Content-Type: text/parameters\r\n");

    request.append("Content-Length: 17\r\n");

    request.append("\r\n");

    request.append("wfd\_idr\_request\r\n");

    status\_t err =

    mNetSession->sendRequest(sessionID, request.c\_str(), request.size());

    ALOGI("%s\n",request.c\_str());

    registerResponseHandler(

            sessionID, mNextCSeq, &WifiDisplaySink::onReceiveIdrResponse);

    if (err != OK) {

        return;

    }

    ++mNextCSeq;

}

Wifidisplay sink端发送请求I帧补丁如下：

wangjianhui@ubuntu:~/3\_rk3399\_Android7.1\_all/packages/apps/WifiDisplay$ git diff .

diff --git a/jni/TunnelRenderer.cpp b/jni/TunnelRenderer.cpp

index c918985..321b483 100755

--- a/jni/TunnelRenderer.cpp

+++ b/jni/TunnelRenderer.cpp

@@ -908,6 +908,7 @@ sp<ABuffer> TunnelRenderer::dequeueBuffer() {

}

#endif

+ property\_set("persist.sys.wfd.droppingpacket", "1");

ALOGI("dropping packet. extSeqNo %d didn't arrive in time but newSeqNo %d",

mLastDequeuedExtSeqNo + 1, extSeqNo);

diff --git a/jni/WifiDisplaySink.cpp b/jni/WifiDisplaySink.cpp

index 109b72e..f06a9dd 100755

--- a/jni/WifiDisplaySink.cpp

+++ b/jni/WifiDisplaySink.cpp

@@ -158,6 +158,8 @@ void \*WifiDisplaySink::ThreadWrapper(void \*)

fds[0].fd = -1;

goto wfd\_sink\_thread\_end;

}

+

+ property\_set("persist.sys.wfd.droppingpacket", 0);

while(!end\_flag)

{

int ret;

@@ -220,6 +222,11 @@ void \*WifiDisplaySink::ThreadWrapper(void \*)

}

}

+

+ if (property\_get\_int32("persist.sys.wfd.droppingpacket", 0) == 1) {

+ property\_set("persist.sys.wfd.droppingpacket", 0);

+ sendIDR(mSessionID,mUrl->c\_str());

+ }

}

wfd\_sink\_thread\_end:

ALOGD("end of threadloop end\_flag %d errno %d",end\_flag,errno);

## Wifidisplay Source端设置I帧间隔时间

diff --git a/media/libstagefright/wifi-display/source/Converter.cpp b/media/libstagefright/wifi-display/source/Converter.cpp

index 379d769..ed7a42c 100755

--- a/media/libstagefright/wifi-display/source/Converter.cpp

+++ b/media/libstagefright/wifi-display/source/Converter.cpp

@@ -171,8 +171,8 @@ status\_t Converter::initEncoder() {

} else {

mOutputFormat->setInt32("bitrate", videoBitrate);

mOutputFormat->setInt32("bitrate-mode",OMX\_Video\_ControlRateConstant);

mOutputFormat->setInt32("frame-rate", 30);

- mOutputFormat->setInt32("i-frame-interval", 15); // Iframes every 15 secs

+ mOutputFormat->setInt32("i-frame-interval", 3); // Iframes every 3 secs

// Configure encoder to use intra macroblock refresh mode

mOutputFormat->setInt32("intra-refresh-mode", OMX\_VIDEO\_IntraRefreshCyclic);

默认是15s发送一个I帧，修改后默认3s发送一个I帧。发送I帧间隔时间不能太小，否则会出现视频码流包太大，消耗网络带宽。

## Wifidisplay去除HDCP加密补丁

在实际使用过程中，有些客户使用的第三方接收端设备或者发送端设备不带HDCP加密功能，需要把RK的设备去除HDCP加密功能，才能实现投屏功能。可以使用下面补丁：

RK设备做source功能

--- a/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp

+++ b/media/libstagefright/wifi-display/source/WifiDisplaySource.cpp

@@ -591,7 +591,7 @@ status\_t WifiDisplaySource::sendM1(int32\_t sessionID) {

 status\_t WifiDisplaySource::sendM3(int32\_t sessionID) {

     AString body =

- "wfd\_content\_protection\r\n"

+ //"wfd\_content\_protection\r\n"

         "wfd\_video\_formats\r\n"

         "wfd\_audio\_codecs\r\n"

         "wfd\_client\_rtp\_ports\r\n";

RK设备做sink功能

--- a/media/libstagefright/wifi-display/sink/WifiDisplaySink.cpp

+++ b/media/libstagefright/wifi-display/sink/WifiDisplaySink.cpp

@@ -1038,6 +1038,7 @@ void WifiDisplaySink::onGetParameterRequest(

                }

 #else

         if (strstr(request\_param, "wfd\_content\_protection")) {

+#if 0

 #ifdef WFD\_HDCP\_SUPPORT

              mUsingHDCP = true;

 #endif

@@ -1045,6 +1046,8 @@ void WifiDisplaySink::onGetParameterRequest(

                 body.append(AStringPrintf("wfd\_content\_protection: HDCP2.1 port=%d\r\n",kHDCPDefaultPort));

              else

                 body.append("wfd\_content\_protection: none\r\n");

+#endif

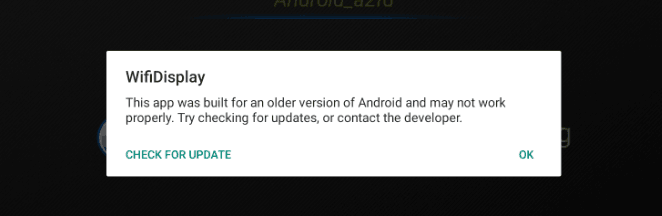
+ body.append("wfd\_content\_protection: none\r\n");

         }

 #endif

## Android9.0系统加wifidisplay sink功能弹提示框错误

Android9.0系统，带上wifidisplay sink功能后，有弹如下图错误。



解决方案：

framewk/base/services/core/java/com/android/server/am/AppWarnings.java

onStartActivity(ActivityRecord r){  
        showUnsupportedCompileSdkDialogIfNeeded(r);  
        showUnsupportedDisplaySizeDialogIfNeeded(r);  
        showDeprecatedTargetDialogIfNeeded(r);//这行代码加个过滤条件，即可解决  
}  
public void showDeprecatedTargetDialogIfNeeded(ActivityRecord r) {  
        if (r.appInfo.targetSdkVersion < Build.VERSION.MIN\_SUPPORTED\_TARGET\_SDK\_INT) {  
            mUiHandler.showDeprecatedTargetDialog(r);  
        }  
}

## Wifidisplay sink延时优化

客户使用过程中由于wifi干扰导致投屏延时严重问题，可以通过丢弃一些过时的数据包来优化延时效果。以Android8.1为例补丁如下（可以通过提redmine获取补丁）：

From 680dcc219fe6dc7a2efe63b43608b0f10ef27b99 Mon Sep 17 00:00:00 2001

From: WangJianhui <wjh@rock-chips.com>

Date: Thu, 25 Oct 2018 15:05:21 +0800

Subject: [PATCH] nuplayer: optimize wfd sink display rate.

Signed-off-by: WangJianhui <wjh@rock-chips.com>

---

media/libmediaplayerservice/nuplayer/NuPlayer.cpp | 4 +

.../nuplayer/NuPlayerDecoder.cpp | 6 +

.../nuplayer/NuPlayerRenderer.cpp | 130 +++++++++++++++++++++

.../nuplayer/NuPlayerRenderer.h | 13 +++

.../nuplayer/NuPlayerSource.h | 4 +

.../nuplayer/StreamingSource.cpp | 23 +++-

.../nuplayer/StreamingSource.h | 8 ++

media/libstagefright/mpeg2ts/ATSParser.h | 2 +

8 files changed, 188 insertions(+), 2 deletions(-)

diff --git a/media/libmediaplayerservice/nuplayer/NuPlayer.cpp b/media/libmediaplayerservice/nuplayer/NuPlayer.cpp

index df36046..2bf7cd3 100644

--- a/media/libmediaplayerservice/nuplayer/NuPlayer.cpp

+++ b/media/libmediaplayerservice/nuplayer/NuPlayer.cpp

@@ -1488,6 +1488,10 @@ void NuPlayer::onStart(int64\_t startPositionUs, MediaPlayerSeekMode mode) {

flags |= Renderer::FLAG\_REAL\_TIME;

}

+ if (mSource->isWFDStreaming()) {

+ flags |= Renderer::FLAG\_WFD\_STREAMING;

+ }

+

bool hasAudio = (mSource->getFormat(true /\* audio \*/) != NULL);

bool hasVideo = (mSource->getFormat(false /\* audio \*/) != NULL);

if (!hasAudio && !hasVideo) {

diff --git a/media/libmediaplayerservice/nuplayer/NuPlayerDecoder.cpp b/media/libmediaplayerservice/nuplayer/NuPlayerDecoder.cpp

index ac187cc..9bc1fb6 100644

--- a/media/libmediaplayerservice/nuplayer/NuPlayerDecoder.cpp

+++ b/media/libmediaplayerservice/nuplayer/NuPlayerDecoder.cpp

@@ -769,6 +769,12 @@ bool NuPlayer::Decoder::handleAnOutputBuffer(

if (mRenderer != NULL) {

// send the buffer to renderer.

+ if (mSource->isWFDStreaming()) {

+ int64\_t sys\_start\_time = mSource->getWFDStartSysTimeUs();

+ int64\_t audio\_start\_mediaTimeUs = mSource->getWFDStartMediaTimeUs();

+ if ((sys\_start\_time != -1) && (audio\_start\_mediaTimeUs != -1))

+ mRenderer->setWFDTimeUs(sys\_start\_time, audio\_start\_mediaTimeUs);

+ }

mRenderer->queueBuffer(mIsAudio, buffer, reply);

if (eos && !isDiscontinuityPending()) {

mRenderer->queueEOS(mIsAudio, ERROR\_END\_OF\_STREAM);

diff --git a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp

index bd866cb..d63f89b 100644

--- a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp

+++ b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp

@@ -34,6 +34,8 @@

#include <inttypes.h>

+#define DIRECT\_RENDER\_NO\_AVSYNC

+

namespace android {

/\*

@@ -86,6 +88,7 @@ const NuPlayer::Renderer::PcmInfo NuPlayer::Renderer::AUDIO\_PCMINFO\_INITIALIZER

// static

const int64\_t NuPlayer::Renderer::kMinPositionUpdateDelayUs = 100000ll;

+FILE \*omx\_rs\_txt;

NuPlayer::Renderer::Renderer(

const sp<MediaPlayerBase::AudioSink> &sink,

@@ -104,6 +107,11 @@ NuPlayer::Renderer::Renderer(

mVideoDrainGeneration(0),

mAudioEOSGeneration(0),

mPlaybackSettings(AUDIO\_PLAYBACK\_RATE\_DEFAULT),

+ sys\_start\_time(-1),

+ audio\_start\_timeUs(-1),

+ last\_adujst\_time(-1),

+ last\_timeUs(-1),

+ last\_cont\_timeUs(-1),

mAudioFirstAnchorTimeMediaUs(-1),

mAnchorTimeMediaUs(-1),

mAnchorNumFramesWritten(-1),

@@ -1044,6 +1052,121 @@ bool NuPlayer::Renderer::onDrainAudioQueue() {

CHECK(entry->mBuffer->meta()->findInt64("timeUs", &mediaTimeUs));

ALOGV("onDrainAudioQueue: rendering audio at media time %.2f secs",

mediaTimeUs / 1E6);

+ if (mFlags & FLAG\_WFD\_STREAMING) {

+ int64\_t sys\_time = systemTime(SYSTEM\_TIME\_MONOTONIC) / 1000;

+ CHECK\_EQ(mAudioSink->getPosition(&numFramesPlayed), (status\_t)OK);

+ uint32\_t numFramesPendingPlayout = mNumFramesWritten - numFramesPlayed;

+ if(sys\_start\_time == 0)

+ {

+ sys\_start\_time =sys\_time;

+ ALOGD("sys\_start\_time == 0 %lld",(long long)sys\_start\_time);

+ }

+ if(audio\_start\_timeUs == 0)

+ {

+ audio\_start\_timeUs = mediaTimeUs;

+ ALOGD("audio\_start\_timeUs == 0 %lld",(long long)audio\_start\_timeUs);

+ }

+ if(last\_adujst\_time == 0)

+ last\_adujst\_time = sys\_time;

+

+ if (last\_cont\_timeUs > mediaTimeUs) {

+ ALOGD("###mediatimeus has been reset, %lld %lld", (long long)last\_cont\_timeUs, (long long)mediaTimeUs);

+ last\_timeUs = 0;

+ }

+ last\_cont\_timeUs = mediaTimeUs;

+

+ int64\_t pending\_time = numFramesPendingPlayout \* mAudioSink->msecsPerFrame() \* 1000ll;

+

+ if(sys\_start\_time + (mediaTimeUs - audio\_start\_timeUs) - pending\_time < sys\_time - 100000ll )

+ {

+ if(last\_timeUs <= mediaTimeUs )//loop tntil the real mediaTimeUs catch up with the old setted one , if there is no data,the old setted is also faster than the real mediaTimeUs.so it's okay

+ {

+ if(sys\_start\_time + (mediaTimeUs - audio\_start\_timeUs) - pending\_time< sys\_time - 300000ll || (sys\_time - last\_adujst\_time > 20000000ll && sys\_start\_time + (mediaTimeUs - audio\_start\_timeUs) - pending\_time< sys\_time - 100000ll))//recalcu late the mediaTimeUs.

+ {

+ int retrtptxt;

+ if((retrtptxt = access("data/test/omx\_rs\_txt\_file2",0)) == 0)

+ {

+ if(omx\_rs\_txt == NULL)

+ omx\_rs\_txt = fopen("data/test/omx\_rs\_txt2.txt","a");

+ if(omx\_rs\_txt != NULL)

+

+ {

+ if(sys\_time - last\_adujst\_time > 20000000ll && sys\_start\_time + (mediaTimeUs- audio\_start\_timeUs) - pending\_time < sys\_time - 100000ll)

+ fprintf(omx\_rs\_txt,"NuPlayer::Renderer::onDrainAudioQueue adjust start %lld %lld sys %lld %lld mediaTimeUs %lld last %lld delta %lld %lld %lld\n",(long long)sys\_start\_time,(long long)audio\_start\_timeUs,(long long)last\_adujst\_time,(long long)sys\_time, (long long)mediaTimeUs,(long long)last\_timeUs,(long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs) + pending\_time), (long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs)),(long long)(sys\_time-last\_adujst\_time));

+ else

+ fprintf(omx\_rs\_txt,"NuPlayer::Renderer::onDrainAudioQueue before delay 300msstart %lld %lld sys %lld %lld mediaTimeUs %lld last %lld delta %lld %lld %lld\n",(long long)sys\_start\_time,(long long)audio\_start\_timeUs,(long long)last\_adujst\_time,(long long)sys\_time,(long long)mediaTimeUs,(long long)last\_timeUs,(long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs) + pending\_time),(long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs)),(long long)(sys\_time-last\_adujst\_time));

+ fflush(omx\_rs\_txt);

+

+ }

+ }

+ ALOGD("catchup:%lld:%lld:%lld:%lld:%lld:%lld:%zu",(long long)sys\_start\_time,(long long)audio\_start\_timeUs,(long long)mediaTimeUs,(long long)sys\_time,(long long)(((sys\_time - sys\_start\_time - (mediaTimeUs - audio\_start\_timeUs))/11)\*11),(long long)pending\_time, mAudioQueue.size());

+ mediaTimeUs +=((sys\_time - sys\_start\_time - (mediaTimeUs - audio\_start\_timeUs) ) / 11) \*11;

+ last\_adujst\_time = sys\_time;

+ }

+ else

+ {

+ int retrtptxt;

+ if((retrtptxt = access("data/test/omx\_rs\_txt\_file2",0)) == 0)

+ {

+ if(omx\_rs\_txt == NULL)

+ omx\_rs\_txt = fopen("data/test/omx\_rs\_txt2.txt","a");

+ if(omx\_rs\_txt != NULL)

+ {

+ fprintf(omx\_rs\_txt,"NuPlayer::Renderer::onDrainAudioQueue before dec delay 100-300 ms start %lld %lld sys %lld %lld mediaTimeUs %lld last %lld delta %lld %lld %lld\n",(long long)sys\_start\_time,(long long)audio\_start\_timeUs,(long long)last\_adujst\_time,(long long)sys\_time,(long long)mediaTimeUs,(long long)last\_timeUs,(long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs) + pending\_time), (long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs)),(long long)(sys\_time-last\_adujst\_time));

+ fflush(omx\_rs\_txt);

+

+ }

+ }

+ }

+ }

+ else

+ {

+ int retrtptxt;

+ if((retrtptxt = access("data/test/omx\_rs\_txt\_file2",0)) == 0)

+ {

+ if(omx\_rs\_txt == NULL)

+ omx\_rs\_txt = fopen("data/test/omx\_rs\_txt2.txt","a");

+ if(omx\_rs\_txt != NULL)

+

+ {

+ fprintf(omx\_rs\_txt,"NuPlayer::Renderer::onDrainAudioQueue before dec delay 100-300 ms start %lld %lld sys %lld %lld mediaTimeUs %lld last %lld delta %lld %lld %lld\n" ,(long long)sys\_start\_time,(long long)audio\_start\_timeUs,(long long)last\_adujst\_time,(long long)sys\_time,(long long)mediaTimeUs,(long long)last\_timeUs,(long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs) + pending\_time), (long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs)),(long long)(sys\_time-last\_adujst\_time));

+ fflush(omx\_rs\_txt);

+

+ }

+ }

+ {

+ ALOGV("discarding the late mediatimeUs %lld:%lld:%lld:%lld",(long long)mediaTimeUs,(long long)sys\_start\_time,(long long)audio\_start\_timeUs, (long long)last\_timeUs);

+ entry->mNotifyConsumed->post();

+ mAudioQueue.erase(mAudioQueue.begin());

+ entry = NULL;

+ }

+ continue;

+ }

+ }

+ else

+ {

+ int retrtptxt;

+ if((retrtptxt = access("data/test/omx\_rs\_txt\_file2",0)) == 0)

+ {

+ if(omx\_rs\_txt == NULL)

+ omx\_rs\_txt = fopen("data/test/omx\_rs\_txt2.txt","a");

+ if(omx\_rs\_txt != NULL)

+

+ {

+ fprintf(omx\_rs\_txt,"NuPlayer::Renderer::onDrainAudioQueue before less than 100ms start %lld %lld sys %lld %lld mediaTimeUs %lld last %lld delta %lld %lld %lld\n",(long long)sys\_start\_time,(long long)audio\_start\_timeUs,(long long)last\_adujst\_time,(long long)sys\_time,(long long)mediaTimeUs,(long long)last\_timeUs ,(long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs) + pending\_time),(long long)(sys\_time-sys\_start\_time-(mediaTimeUs - audio\_start\_timeUs)),(long long)(sys\_time-last\_adujst\_time));

+ fflush(omx\_rs\_txt);

+ }

+ }

+ }

+

+ if (last\_timeUs > mediaTimeUs) {

+ ALOGD("error,last\_timeUs=%lld, mediaTimeUs=%lld", (long long)last\_timeUs, (long long)mediaTimeUs);

+ }

+

+ last\_timeUs = mediaTimeUs;

+ if(sys\_time - last\_adujst\_time > 20000000ll)

+ last\_adujst\_time = sys\_time;

+ }

onNewAudioMediaTime(mediaTimeUs);

}

@@ -1285,6 +1408,13 @@ void NuPlayer::Renderer::postDrainVideoQueue() {

// discontinuity. If we have not drained an audio buffer that was

// received after this buffer, repost in 10 msec. Otherwise repost

// in 500 msec.

+ if(mFlags & FLAG\_WFD\_STREAMING) {

+#ifdef DIRECT\_RENDER\_NO\_AVSYNC

+ msg->post();

+ mDrainVideoQueuePending = true;

+ return;

+#endif

+ }

delayUs = realTimeUs - nowUs;

int64\_t postDelayUs = -1;

if (delayUs > 500000) {

diff --git a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h

index f58b79c..ffc3b56 100644

--- a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h

+++ b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h

@@ -34,6 +34,7 @@ struct NuPlayer::Renderer : public AHandler {

enum Flags {

FLAG\_REAL\_TIME = 1,

FLAG\_OFFLOAD\_AUDIO = 2,

+ FLAG\_WFD\_STREAMING = 8,

};

Renderer(const sp<MediaPlayerBase::AudioSink> &sink,

const sp<AMessage> &notify,

@@ -89,6 +90,11 @@ struct NuPlayer::Renderer : public AHandler {

bool isStreaming,

const sp<AMessage> &notify);

+ void setWFDTimeUs(int64\_t sysTimeUs, int64\_t mediaTimeUs) {

+ sys\_start\_time = sysTimeUs;

+ audio\_start\_timeUs = mediaTimeUs;

+ }

+

enum {

kWhatEOS = 'eos ',

kWhatFlushComplete = 'fluC',

@@ -172,6 +178,13 @@ private:

AVSyncSettings mSyncSettings;

float mVideoFpsHint;

+ //wifi display instance

+ int64\_t sys\_start\_time;

+ int64\_t audio\_start\_timeUs;

+ int64\_t last\_adujst\_time;

+ int64\_t last\_timeUs;

+ int64\_t last\_cont\_timeUs;

+

int64\_t mAudioFirstAnchorTimeMediaUs;

int64\_t mAnchorTimeMediaUs;

int64\_t mAnchorNumFramesWritten;

diff --git a/media/libmediaplayerservice/nuplayer/NuPlayerSource.h b/media/libmediaplayerservice/nuplayer/NuPlayerSource.h

index 8ba9c0d..8a992a2 100644

--- a/media/libmediaplayerservice/nuplayer/NuPlayerSource.h

+++ b/media/libmediaplayerservice/nuplayer/NuPlayerSource.h

@@ -146,6 +146,10 @@ struct NuPlayer::Source : public AHandler {

return INVALID\_OPERATION;

}

+ virtual bool isWFDStreaming() {return false;}

+ virtual int64\_t getWFDStartSysTimeUs() {return -1;}

+ virtual int64\_t getWFDStartMediaTimeUs() {return -1;}

+

protected:

virtual ~Source() {}

diff --git a/media/libmediaplayerservice/nuplayer/StreamingSource.cpp b/media/libmediaplayerservice/nuplayer/StreamingSource.cpp

index fc0803b..04acdd0 100644

--- a/media/libmediaplayerservice/nuplayer/StreamingSource.cpp

+++ b/media/libmediaplayerservice/nuplayer/StreamingSource.cpp

@@ -33,7 +33,7 @@

namespace android {

-const int32\_t kNumListenerQueuePackets = 80;

+const int32\_t kNumListenerQueuePackets = 200;

NuPlayer::StreamingSource::StreamingSource(

const sp<AMessage> &notify,

@@ -41,6 +41,9 @@ NuPlayer::StreamingSource::StreamingSource(

: Source(notify),

mSource(source),

mFinalResult(OK),

+ mWFDFlag(false),

+ mWFDStartSysTimeUs(-1),

+ mWFDStartMediaTimeUs(-1),

mBuffering(false) {

}

@@ -91,6 +94,12 @@ void NuPlayer::StreamingSource::start() {

parserFlags |= ATSParser::ALIGNED\_VIDEO\_DATA;

}

+ if ((sourceFlags >> 16 & 0xFFFF) == 0x1234) {

+ mWFDFlag = true;

+ parserFlags |= ATSParser::WIFI\_DISPLAY;

+ ALOGD("NuPlayer::StreamingSource::start sourceFlags %x",sourceFlags);

+ }

+

mTSParser = new ATSParser(parserFlags);

mStreamListener->start();

@@ -129,8 +138,18 @@ void NuPlayer::StreamingSource::onReadBuffer() {

type = mask;

}

- mTSParser->signalDiscontinuity(

+ if (mWFDFlag)

+ {

+ int64\_t sys\_timeUs;

+ int64\_t mediaTimeUs;

+ if (extra->findInt64("wifidisplay\_sys\_timeUs", &sys\_timeUs) && extra->findInt64("timeUs", &mediaTimeUs)) {

+ mWFDStartSysTimeUs = sys\_timeUs;

+ mWFDStartMediaTimeUs = mediaTimeUs;

+ }

+ } else {

+ mTSParser->signalDiscontinuity(

(ATSParser::DiscontinuityType)type, extra);

+ }

} else if (n < 0) {

break;

} else {

diff --git a/media/libmediaplayerservice/nuplayer/StreamingSource.h b/media/libmediaplayerservice/nuplayer/StreamingSource.h

index 2e1d2b3..f993580 100644

--- a/media/libmediaplayerservice/nuplayer/StreamingSource.h

+++ b/media/libmediaplayerservice/nuplayer/StreamingSource.h

@@ -45,6 +45,10 @@ struct NuPlayer::StreamingSource : public NuPlayer::Source {

virtual bool isRealTime() const;

+ virtual bool isWFDStreaming() {return mWFDFlag != 0;}

+ virtual int64\_t getWFDStartSysTimeUs() {return mWFDStartSysTimeUs;}

+ virtual int64\_t getWFDStartMediaTimeUs() {return mWFDStartMediaTimeUs;}

+

protected:

virtual ~StreamingSource();

@@ -61,6 +65,10 @@ private:

sp<NuPlayerStreamListener> mStreamListener;

sp<ATSParser> mTSParser;

+ bool mWFDFlag;

+ int64\_t mWFDStartSysTimeUs;

+ int64\_t mWFDStartMediaTimeUs;

+

bool mBuffering;

Mutex mBufferingLock;

sp<ALooper> mLooper;

diff --git a/media/libstagefright/mpeg2ts/ATSParser.h b/media/libstagefright/mpeg2ts/ATSParser.h

index 41c19cd..908b295 100644

--- a/media/libstagefright/mpeg2ts/ATSParser.h

+++ b/media/libstagefright/mpeg2ts/ATSParser.h

@@ -68,6 +68,8 @@ struct ATSParser : public RefBase {

TS\_TIMESTAMPS\_ARE\_ABSOLUTE = 1,

// Video PES packets contain exactly one (aligned) access unit.

ALIGNED\_VIDEO\_DATA = 2,

+ // WiFi Display Flag

+ WIFI\_DISPLAY = 4,

};

enum SourceType {

## Android9.0和Android10.0 Wifidisplay source功能

Google发布Android9.0及以上版本，已经把Wifidsplay source功能去掉了。2019年11月份RK已经把Wifidisplay source功能代码移植进去了，需要source功能的客户可以把SDK代码更新到最新，或者提redmine获取补丁包。

## Wifidisplay Sink不能显示Netflix播放内容

检查Wifdisplay Sink apk代码是否有打开HDCP加密，打开HDCP加密需要在apk里面的jni目录下面的Android.mk文件修改下：LOCAL\_CPPFLAGS += -DWFD\_HDCP\_SUPPORT，再加上下面补丁。还有确认板子是否有烧写过HDCP2.x key，同时确认Source端设备是否有支持HDCP2.x加密功能。

diff --git a/media/libmediaplayerservice/HDCP.cpp b/media/libmediaplayerservice/HDCP.cpp

index afe39367f..bd14ec379 100644

--- a/media/libmediaplayerservice/HDCP.cpp

+++ b/media/libmediaplayerservice/HDCP.cpp

@@ -164,7 +164,7 @@ void HDCP::ObserveWrapper(void \*me, int msg, int ext1, int ext2) {

}

void HDCP::observe(int msg, int ext1, int ext2) {

- Mutex::Autolock autoLock(mLock);

+ //Mutex::Autolock autoLock(mLock); //should be removed as it is a callback from hdcp library.

if (mObserver != NULL) {

mObserver->notify(msg, ext1, ext2, NULL /\* obj \*/);

diff --git a/media/libmediaplayerservice/MediaPlayerService.cpp b/media/libmediaplayerservice/MediaPlayerService.cpp

index 939307ba0..7bd493bbc 100755

--- a/media/libmediaplayerservice/MediaPlayerService.cpp

+++ b/media/libmediaplayerservice/MediaPlayerService.cpp

@@ -350,7 +350,14 @@ sp<IOMX> MediaPlayerService::getOMX() {

}

sp<IHDCP> MediaPlayerService::makeHDCP(bool createEncryptionModule) {

- return new HDCP(createEncryptionModule);

+ ALOGI("MediaPlayerService::getHDCP");

+ Mutex::Autolock autoLock(mLock);

+

+ if (mHDCP.get() == NULL) {

+ mHDCP = new HDCP(createEncryptionModule);

+ }

+

+ return mHDCP;

}

sp<IRemoteDisplay> MediaPlayerService::listenForRemoteDisplay(

diff --git a/media/libmediaplayerservice/MediaPlayerService.h b/media/libmediaplayerservice/MediaPlayerService.h

index 5e6e1c5c7..698f09e75 100755

--- a/media/libmediaplayerservice/MediaPlayerService.h

+++ b/media/libmediaplayerservice/MediaPlayerService.h

@@ -486,6 +486,7 @@ private:

SortedVector< wp<MediaRecorderClient> > mMediaRecorderClients;

int32\_t mNextConnId;

sp<IOMX> mOMX;

+ sp<IHDCP> mHDCP;

};

// ----------------------------------------------------------------------------

diff --git a/media/libstagefright/mpeg2ts/ATSParser.cpp b/media/libstagefright/mpeg2ts/ATSParser.cpp

index a256a4df1..cc57c5ce6 100644

--- a/media/libstagefright/mpeg2ts/ATSParser.cpp

+++ b/media/libstagefright/mpeg2ts/ATSParser.cpp

@@ -22,6 +22,9 @@

#include "CasManager.h"

#include "ESQueue.h"

#include "include/avc\_utils.h"

+#include <binder/IServiceManager.h>

+#include <media/IHDCP.h>

+#include <media/IMediaPlayerService.h>

#include <android/hardware/cas/native/1.0/IDescrambler.h>

#include <cutils/native\_handle.h>

@@ -201,6 +204,7 @@ private:

List<off64\_t> mPesStartOffsets;

ElementaryStreamQueue \*mQueue;

+ sp<IHDCP> mHDCP;

bool mScrambled;

bool mSampleEncrypted;

@@ -1226,6 +1230,54 @@ status\_t ATSParser::Stream::parsePES(ABitReader \*br, SyncEvent \*event) {

optional\_bytes\_remaining -= 3;

}

+ //HDCP Private Data

+ bool useHDCP = false;

+ uint32\_t streamCTR = 0;

+ uint64\_t outInputCTR = 0;

+ if(optional\_bytes\_remaining >= 17){

+ uint8\_t hdcp\_private\_flag = br->getBits(8);

+ optional\_bytes\_remaining -= 1;

+ if(hdcp\_private\_flag==0x8e){

+ /\* stream Counter \*/

+ br->skipBits(13);

+ streamCTR = br->getBits(2) << 30;

+ br->getBits(1);

+ streamCTR |= br->getBits(15) << 15;

+ br->getBits(1);

+ streamCTR |= br->getBits(15);

+ br->getBits(1);

+

+ /\* input Counter \*/

+ br->skipBits(11);

+ outInputCTR = ((uint64\_t)br->getBits(4)) << 60;

+ br->getBits(1);

+ outInputCTR |= ((uint64\_t)br->getBits(15)) << 45;

+ br->getBits(1);

+ outInputCTR |= br->getBits(15) << 30;

+ br->skipBits(1);

+ outInputCTR |= br->getBits(15) << 15;

+ br->getBits(1);

+ outInputCTR |= br->getBits(15);

+ br->getBits(1);

+

+ useHDCP = true;

+ //ALOGV("this packet has HDCP flag, CTR vlaue:%d:%lld", streamCTR, outInputCTR);

+ optional\_bytes\_remaining -= 16;

+ }

+ }

+

+ if(useHDCP){

+ if(mHDCP==NULL){

+ sp<IServiceManager> sm = defaultServiceManager();

+ sp<IBinder> binder = sm->getService(String16("media.player"));

+ sp<IMediaPlayerService> service = interface\_cast<IMediaPlayerService>(binder);

+ CHECK(service != NULL);

+ mHDCP = service->makeHDCP(false);

+ if (mHDCP != NULL)

+ ALOGD("successfully make HDCP decrypt object");

+ }

+ }

+

br->skipBits(optional\_bytes\_remaining \* 8);

// ES data follows.

@@ -1250,15 +1302,51 @@ status\_t ATSParser::Stream::parsePES(ABitReader \*br, SyncEvent \*event) {

ALOGV("There's %u bytes of payload, PES\_packet\_length=%u, offset=%d",

dataLength, PES\_packet\_length, pesOffset);

- onPayloadData(

- PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

- br->data(), dataLength, pesOffset, event);

+ if (mHDCP != NULL) {

+ const uint8\_t \*pesData = br->data();

+ size\_t dataSize = br->numBitsLeft()/8;

+ void \*outData = malloc(dataSize);

+ //ALOGD("packet length unkowned:%d",dataSize);

+ status\_t err = mHDCP->decrypt(pesData,dataSize,streamCTR,outInputCTR,outData);

+ if(err==OK){

+ onPayloadData(PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

+ (uint8\_t\*)outData, dataSize, pesOffset, event);

+ }else{

+ ALOGE("decrypt pes failed");

+ onPayloadData(PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

+ (uint8\_t\*)pesData, dataSize, pesOffset, event);

+ }

+ free(outData);

+ outData = NULL;

+ }else {

+ onPayloadData(

+ PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

+ br->data(), dataLength, pesOffset, event);

+ }

br->skipBits(dataLength \* 8);

} else {

- onPayloadData(

- PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

- br->data(), br->numBitsLeft() / 8, pesOffset, event);

+ if (mHDCP != NULL) {

+ const uint8\_t \*pesData = br->data();

+ size\_t dataSize = br->numBitsLeft()/8;

+ void \*outData = malloc(dataSize);

+ //ALOGD("packet length unkowned:%d",dataSize);

+ status\_t err = mHDCP->decrypt(pesData,dataSize,streamCTR,outInputCTR,outData);

+ if(err==OK){

+ onPayloadData(PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

+ (uint8\_t\*)outData, dataSize, pesOffset, event);

+ }else{

+ ALOGE("decrypt pes failed");

+ onPayloadData(PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

+ (uint8\_t\*)pesData, dataSize, pesOffset, event);

+ }

+ free(outData);

+ outData = NULL;

+ }else {

+ onPayloadData(

+ PTS\_DTS\_flags, PTS, DTS, PES\_scrambling\_control,

+ br->data(), br->numBitsLeft() / 8, pesOffset, event);

+ }

size\_t payloadSizeBits = br->numBitsLeft();

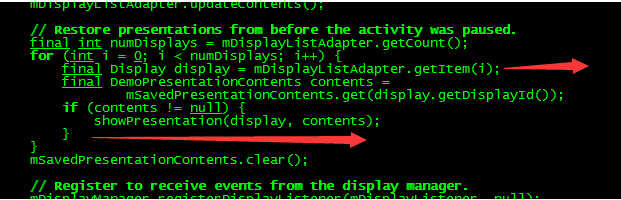
if (payloadSizeBits % 8 != 0u) {

## Wifidisplay HDCP2.x key烧写

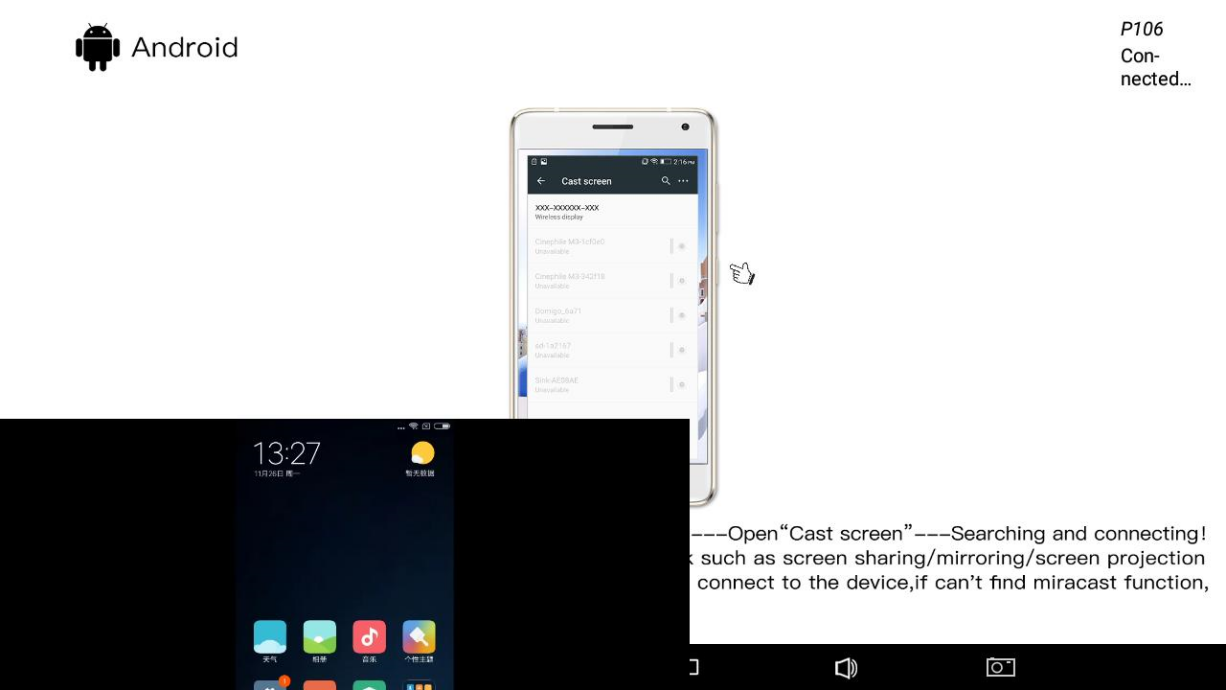
Wifidisplay的HDCP key是用的HDCP2.x，和HDMI HDCP Transmit key不是同一个，需要单独烧写，可以通过FAE窗口获取到烧写HDCP key的工具[RKDevInfoWriteTool\_v1.2.3.zip](https://redmine.rockchip.com.cn/attachments/download/596367/RKDevInfoWriteTool_v1.2.3.zip)。烧写过程在工具设置中把RPMB改成兼容模式，即可烧写。

## Wifidisplay投射主副屏选择

双屏异显的场景，默认投屏的是主屏，假如需要把副屏的内容通过Wifidisplay投射出去，可以双屏异显的app把presentation view 指定用wifidisplay 的id 绑定。在PresentationActivity.java文件。



## Wifidisplay投屏显示问题



没有全屏显示，只显示其中一部分。可以通过修改setprop sys.display.oritation 1来解决。

## Wifidisplay连续投屏12.4h，画面出卡顿

有些客户发现长时间投屏后，出现画面卡顿问题。在framework/av目录下面打上下面修改：

diff --git a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp

index 0b5e159..b0e8561 100644

--- a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp

+++ b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.cpp

@@ -868,7 +868,7 @@ size\_t NuPlayer::Renderer::fillAudioBuffer(void \*buffer, size\_t size) {

postEOSDelayUs = getPendingAudioPlayoutDurationUs(ALooper::GetNowUs());

}

ALOGV("fillAudioBuffer: notifyEOS "

- "mNumFramesWritten:%u finalResult:%d postEOSDelay:%lld",

+ "mNumFramesWritten:%lld finalResult:%d postEOSDelay:%lld",

mNumFramesWritten, entry->mFinalResult, (long long)postEOSDelayUs);

notifyEOS(true /\* audio \*/, entry->mFinalResult, postEOSDelayUs);

}

@@ -1179,7 +1179,7 @@ bool NuPlayer::Renderer::onDrainAudioQueue() {

return reschedule;

}

-int64\_t NuPlayer::Renderer::getDurationUsIfPlayedAtSampleRate(uint32\_t numFrames) {

+int64\_t NuPlayer::Renderer::getDurationUsIfPlayedAtSampleRate(uint64\_t numFrames) {

int32\_t sampleRate = offloadingAudio() ?

mCurrentOffloadInfo.sample\_rate : mCurrentPcmInfo.mSampleRate;

if (sampleRate == 0) {

@@ -1187,7 +1187,7 @@ int64\_t NuPlayer::Renderer::getDurationUsIfPlayedAtSampleRate(uint32\_t numFrames

return 0;

}

// TODO: remove the (int32\_t) casting below as it may overflow at 12.4 hours.

- return (int64\_t)((int32\_t)numFrames \* 1000000LL / sampleRate);

+ return (int64\_t)((int64\_t)numFrames \* 1000000LL / sampleRate);

}

// Calculate duration of pending samples if played at normal rate (i.e., 1.0).

diff --git a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h

index 2e5f9b2..f6472aa 100644

--- a/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h

+++ b/media/libmediaplayerservice/nuplayer/NuPlayerRenderer.h

@@ -147,7 +147,7 @@ private:

uint32\_t mFlags;

List<QueueEntry> mAudioQueue;

List<QueueEntry> mVideoQueue;

- uint32\_t mNumFramesWritten;

+ uint64\_t mNumFramesWritten;

sp<VideoFrameScheduler> mVideoScheduler;

bool mDrainAudioQueuePending;

@@ -289,7 +289,7 @@ private:

void startAudioOffloadPauseTimeout();

void cancelAudioOffloadPauseTimeout();

- int64\_t getDurationUsIfPlayedAtSampleRate(uint32\_t numFrames);

+ int64\_t getDurationUsIfPlayedAtSampleRate(uint64\_t numFrames);

DISALLOW\_EVIL\_CONSTRUCTORS(Renderer);

};