



Android12 Miracast 开发指南

版本号: 1.0
发布日期: 2021.11.20

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.11.20	AWA1646	建立初始版本

目 录

1 前言	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 WifiDisplay 简介	2
2.1 WifiDisplay 协议流程	3
2.2 Android WifiDisplay 实现	7
2.3 Android WifiDisplay 实现	8
2.3.1 Source 端实现	8
2.3.2 Sink 端的实现	9
3 Miracast 使用说明	14
4 附录	20
4.1 Miracast 移植及其代码调试	20
4.1.1 Miracast 代码移植	20
4.1.2 Miracast 代码编译报错修复	20

插图

2-1 WFD 涉及的技术及协议框图	3
2-2 会话建立及协商过程图	4
2-3 RTSP 协议控制图	5
2-4 WFD 设备会话管理的模型	6
2-5 音频及视频流控制模型	7
2-6 DisplayDevice 的隔离示意图	8
2-7 设备发现流程图	8
2-8 Source 端 RTSP 连接流程图	9
2-9 实现框架结构图	10
2-10 RTSP 会话流程图	11
2-11 WFD 控制流程图	12
3-1 Source 设备的设置图	15
3-2 Source 设备的设备连接	15
3-3 Source 设备的 cast 图	16
3-4 Source 设备启用无线显示	16
3-5 Source 搜索可连接的设备	17
3-6 Sink 设备邀请连接	18
3-7 Source 设备邀请连接	19
4-1 移植编译报错图	21

1 前言

1.1 编写目的

为了让多媒体开发人员熟悉 Miracast 流程，实现 Miracast 功能定制和简单调试。

1.2 适用范围

本 Miracast 说明适用于全志科技 Android 系统产品。

1.3 相关人员

多媒体开发人员。

2 WifiDisplay 简介

Wi-Fi Display 经常和 Miracast 联系在一起。实际上，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。

Miracast 的 Android 实现涉及到系统的多个模块，包括：

- MediaPlayerService 及相关模块：因为 Miracast 本身就牵扯到 RTP/RTSP 及相应的编解码技术。
- SurfaceFlinger 及相关模块：SurfaceFlinger 的作用是将各层 UI 数据混屏并投递到显示设备中去显示。现在，SurfaceFlinger 将支持多个显示设备。而支持 Miracast 的远端设备也做为一个独立的显示设备存在于系统中。
- WindowManagerService 及相关模块：WindowManagerService 用于管理系统中各个 UI 层的位置和属性。由于并非所有的 UI 层都会通过 Miracast 投递到远端设备上，所以 WindowManagerService 也需要修改以适应 Miracast 的需要。例如手机中的视频可投递到远端设备上去显示，但假如在播放过程中，突然弹出一个密码输入框（可能是某个后台应用程序发起的），则这个密码输入框就不能投递到远端设备上去显示。
- DisplayManagerService 及相关模块：DisplayManagerService 服务是 Android 4.2 新增的，用于管理系统中所有的 Display 设备。
- WifiService 及相关模块：WifiDisplay 协议的实现建立在 WifiP2P 的基础上，其中涉及的 Wifi 技术包括 WiFi-Direct (WiFi P2P)、Wi-Fi Protected Setup (Wifi 网络自动配置及添加网络)、11n/WMM/WPA2 (11n 就是 802.11n 协议，它将 11a 和 11g 提供的 Wi-Fi 传输速率从 54Mbps 提升到 300 甚至 600Mbps。WMM 是 Wi-Fi Multimedia 的缩写，是一种针对实时音视频数据的 QoS 服务。而 WPA2 意为 Wi-Fi Protected Access 第二版，主要用来给传输的数据进行加密保护)。

下图给出了 WFD 涉及的技术及协议框图，基于 WifiP2P 网络技术，利用 RTSP 作为音频及视频流控制协议，涉及了流媒体的传输、控制、加密、解密、编码及解码等技术流程。WFD 中涉及的技术层面比较多，相关的协议也比较多，包括了 WIFI P2P 技术、RTSP 及 RTP 技术、流媒体技术以及音视频编解码相关的技术，如果要对 WFD 有比较深入的了解，还需要花费较多的时间去研究相关的技术细节。

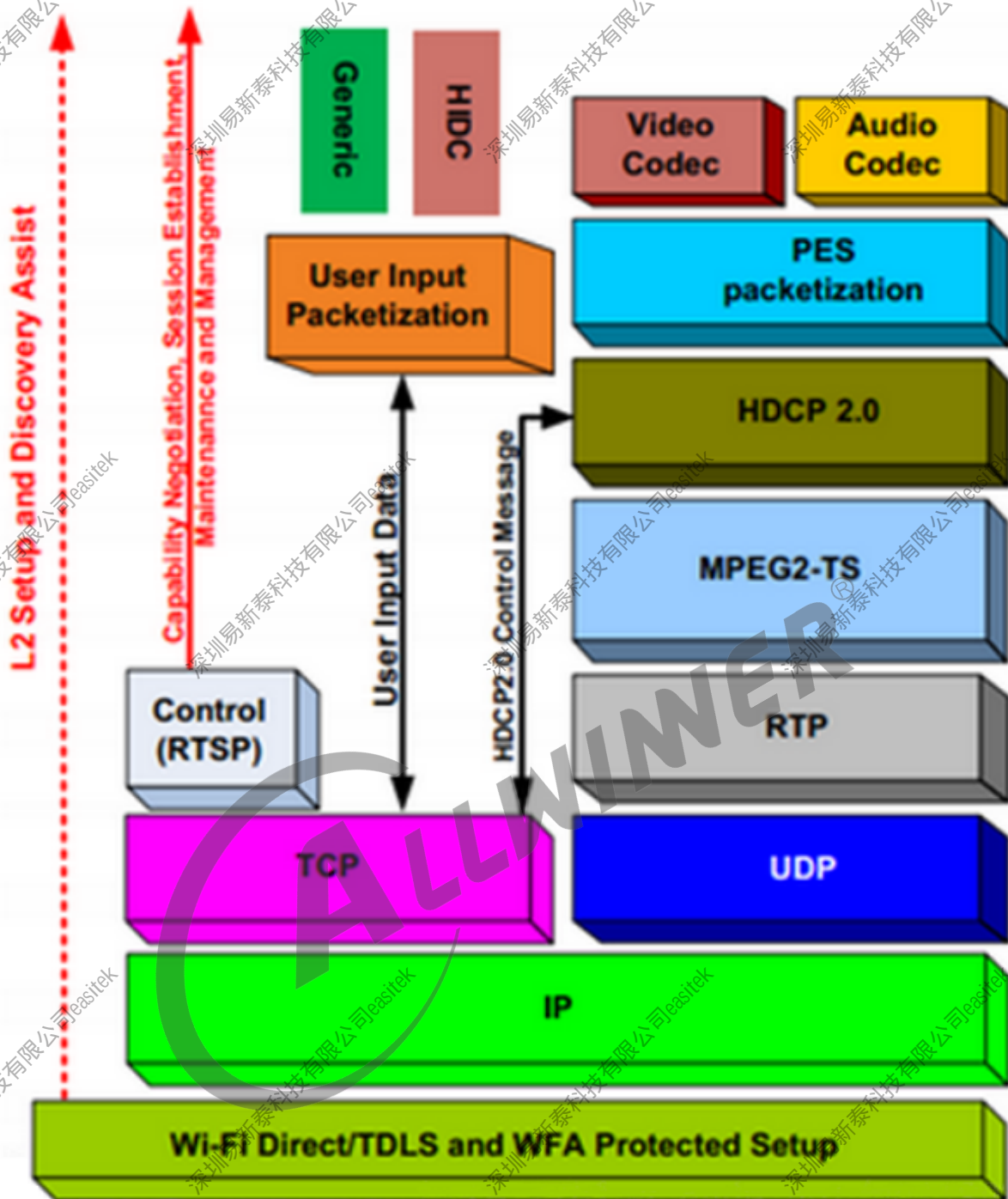


图 2-1: WFD 涉及的技术及协议框图

2.1 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, 协商成功后建立会话)
7. User Input Back Channel Setup (Optional) (UIBC 反向控制, UIBC 通道建立, 用于 Sink 端反向控制 Source 端, 该步骤为可选实现)
8. Link Content Protection Setup (Optional) (内容保护, 即数据加密, 对传输的内容做加密保护 (HDCP))
9. Payload Control (负载控制, 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)
10. WFD Source and WFD Sink standby (Optional)
11. WFD Session Teardown (会话终止)

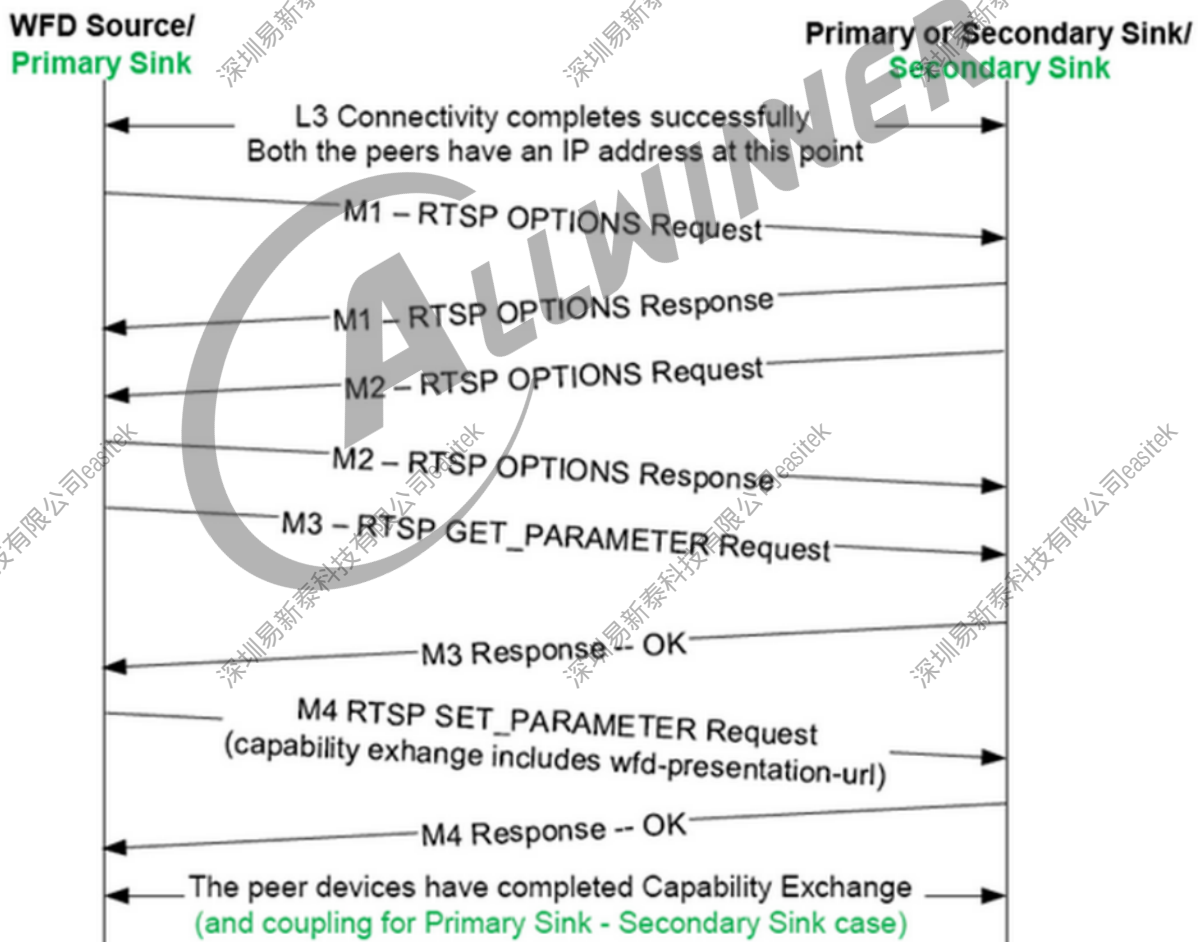


图 2-2: 会话建立及协商过程图

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。

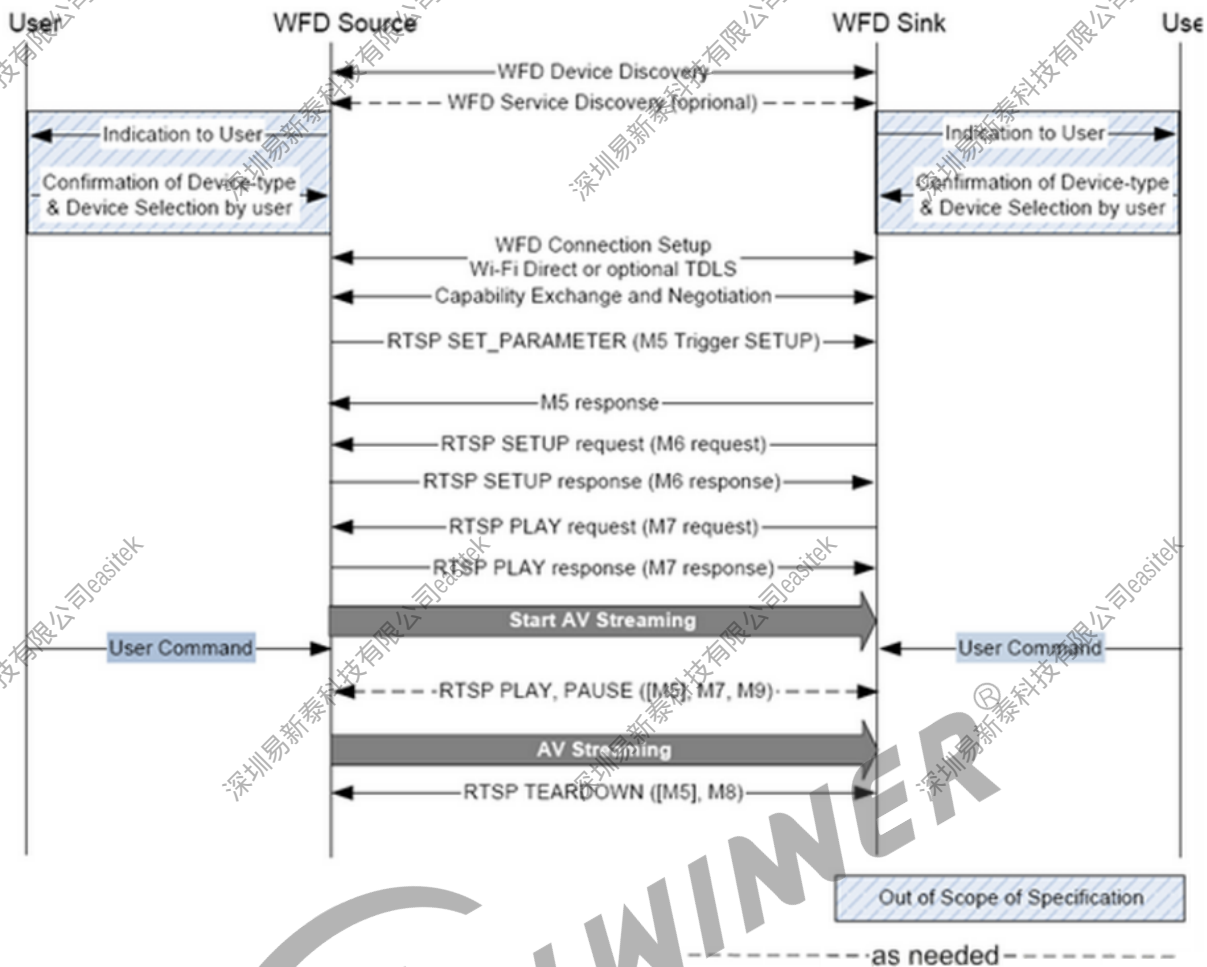


图 2-3: RTSP 协议控制图

当 WFD Source 和 WFD Sink 之间成功完成 RTSP M7 请求和响应消息的交换时，WFD 会话就建立起来了。RTSP 协议控制中主要有以下几种状态 SETUP、PLAY、PAUSE、TEARDOWN WFD Source and WFD Sink session management Reference Model 通过下面命令抓取了 WifiDisplay 相关的协议包，主要是 RTSP 控制流相关的协议包。tcpdump -i any -w /savePath 具体的协议包相关的内容如 WFD 设备会话管理的模型所示，协议中相关的流程及步骤和 WFD 涉及的技术及协议框图、会话建立及协商过程图中的交互流程是一致的，具体包括以下几个主要步骤 OPTIONS、GET_PARAMETER、SET_PARAMETER、SETUP、PLAY、TEARDOWN 等，这些都是 RTSP 中相关的协议内容。当 Source 与 Sink 设备完成 PLAY 的交互后，Source 端便开始传输音频及视频流给 Sink 端，Sink 端作为被动接收端，只需要在 P2P interface 的 19000（默认 RTP 数据传输端口）绑定监听接收来自 Source 端的数据流对相关的音视频流做处理即可。音频及视频流控制模型给出了音视频流的协议包，可以看到音视频的传输通过 MPEG TS、MPEG PES 等相关协议作为传输载体。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 Ddiscovery、WFD Link Establishment、UIBC、Capability Negotiation、Session/Stream Control 等。

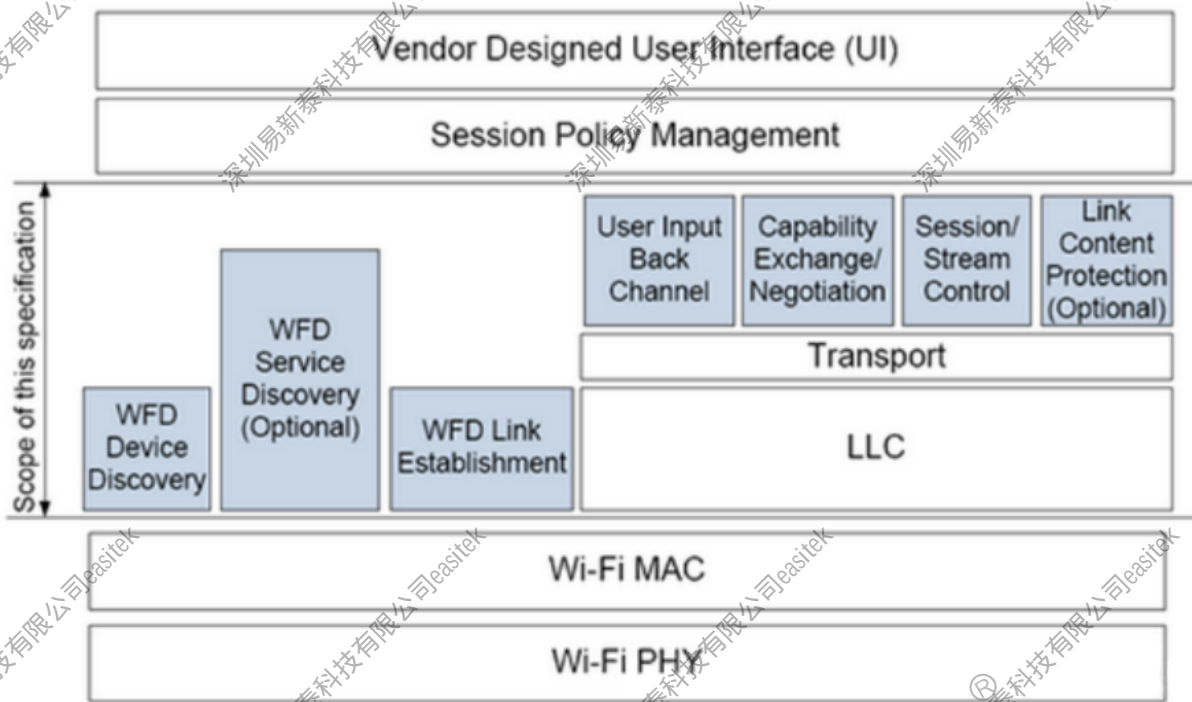


图 2-4: WFD 设备会话管理的模型

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。

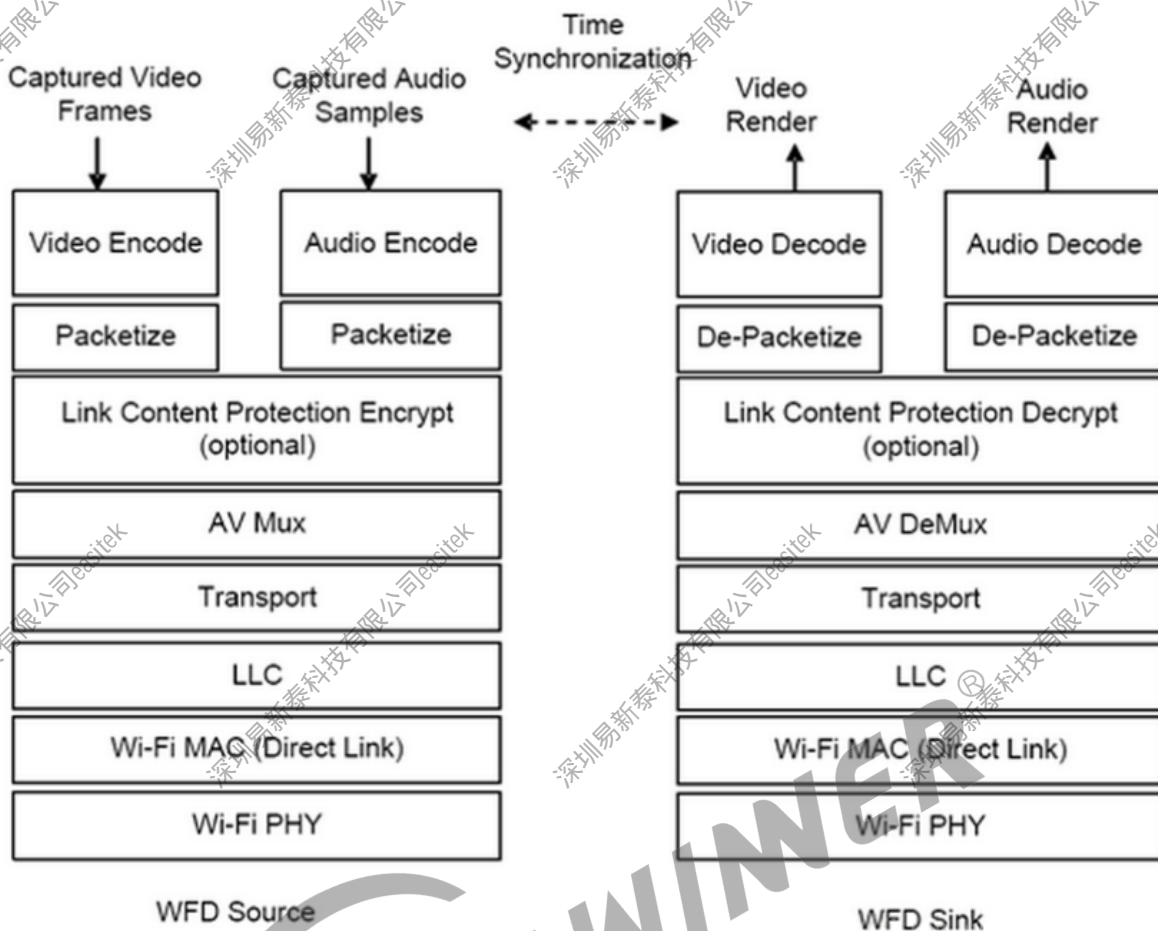


图 2-5: 音频及视频流控制模型

2.2 Android WifiDisplay 实现

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

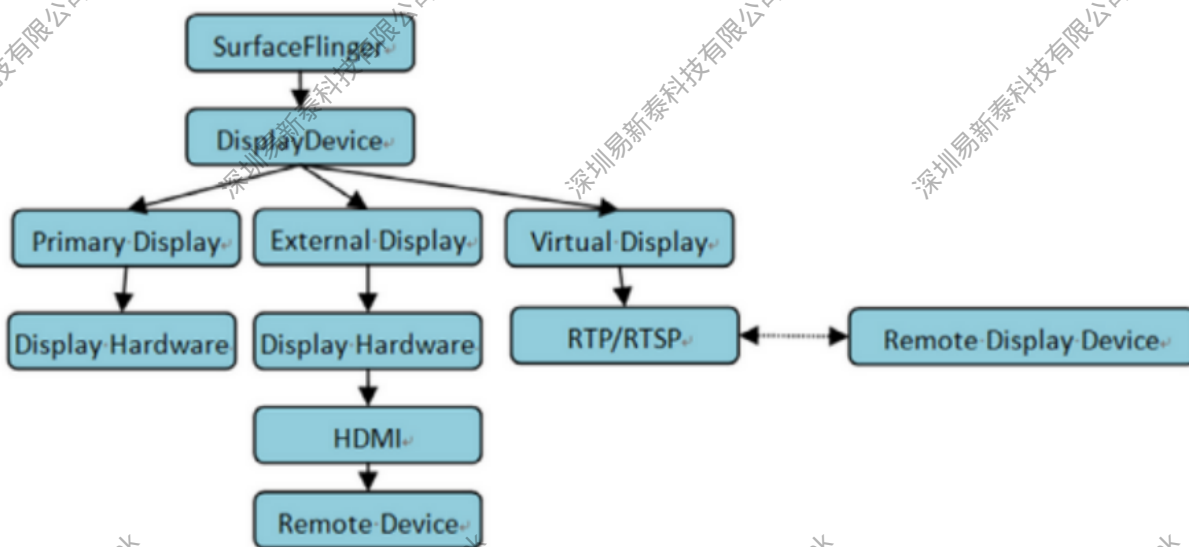


图 2-6: DisplayDevice 的隔离示意图

2.3 Android WifiDisplay 实现

2.3.1 Source 端实现

基于 Android 代码 Source 端入口在原生 Settings-> 设备-> 显示-> 投射, 这个功能如果正常使用时, 需要更改一个配置项:true 该配置项路径为 frameworks/base/core/res/res/values/config.xml, 该入口的主要作用是扫描并发现 sink 设备。

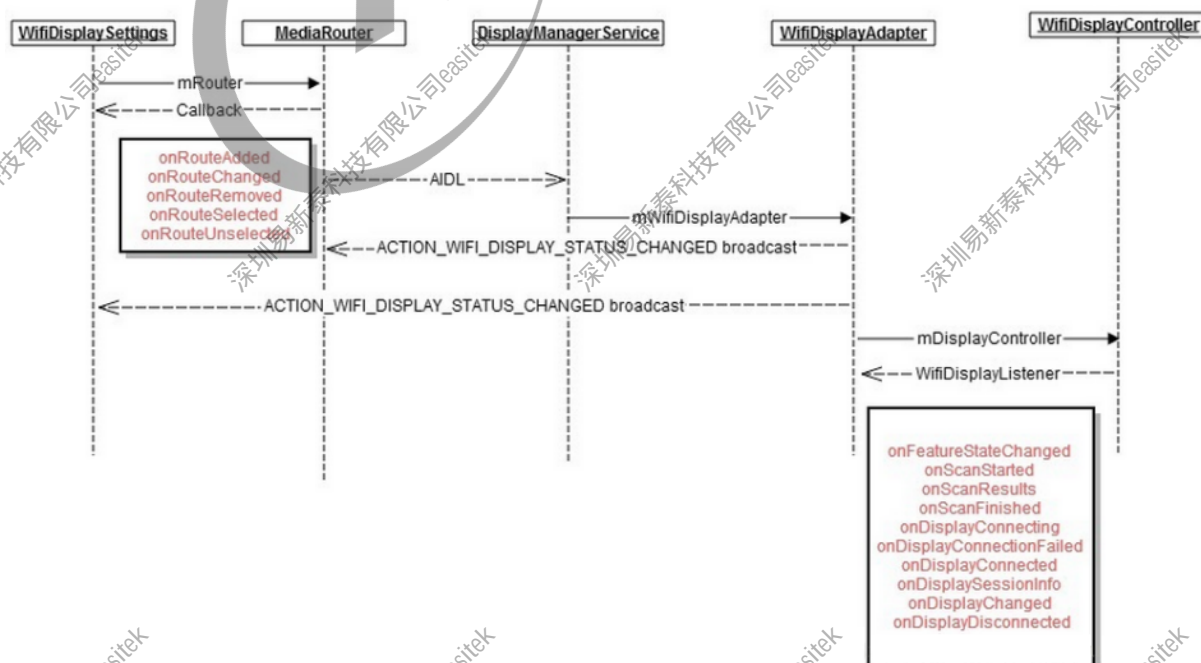


图 2-7: 设备发现流程图

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。触发设备扫描的是在 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。

下图是 Source 端设备建立连接的流程图，主要建立 RSTP 协议的 Socket 连接，通过接收 Sink 端的协议信息解析相关操作，代码流程如下图所示。socket 的建立主要在 WidiSourceRtsp.cpp 的 prepareListenSocket 函数中实现，并在这个 socket 上监听是否有客户端的连接请求，rtsp 消息处理在 cbHandleParserEvent 中处理。在 WidiSession.cpp 中 RSTP 及 media_event 处理主要通过 WorkHandler 处理，相关消息处理在函数 onMessageReceived 中处理。

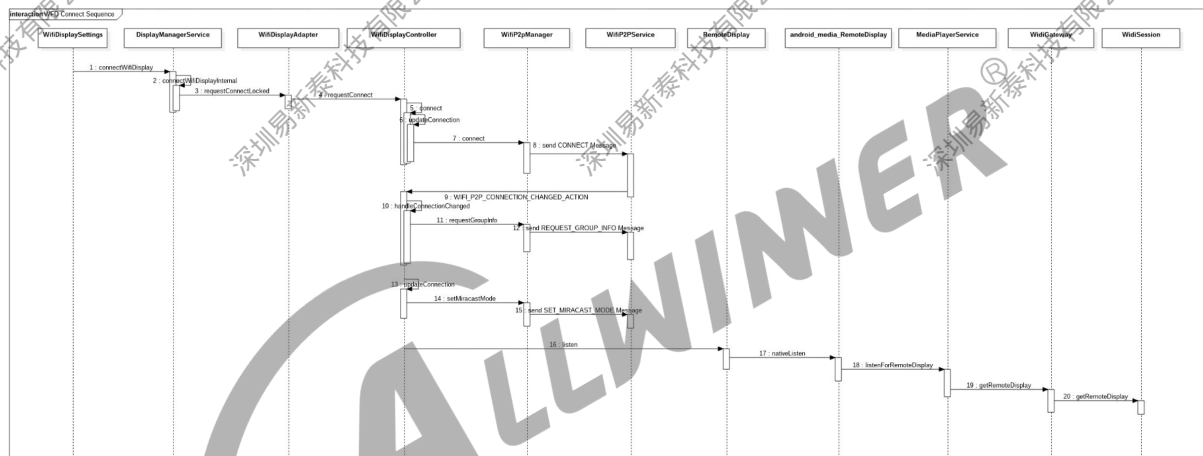


图 2-8: Source 端 RTSP 连接流程图

2.3.2 Sink 端的实现

下图给出了 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。

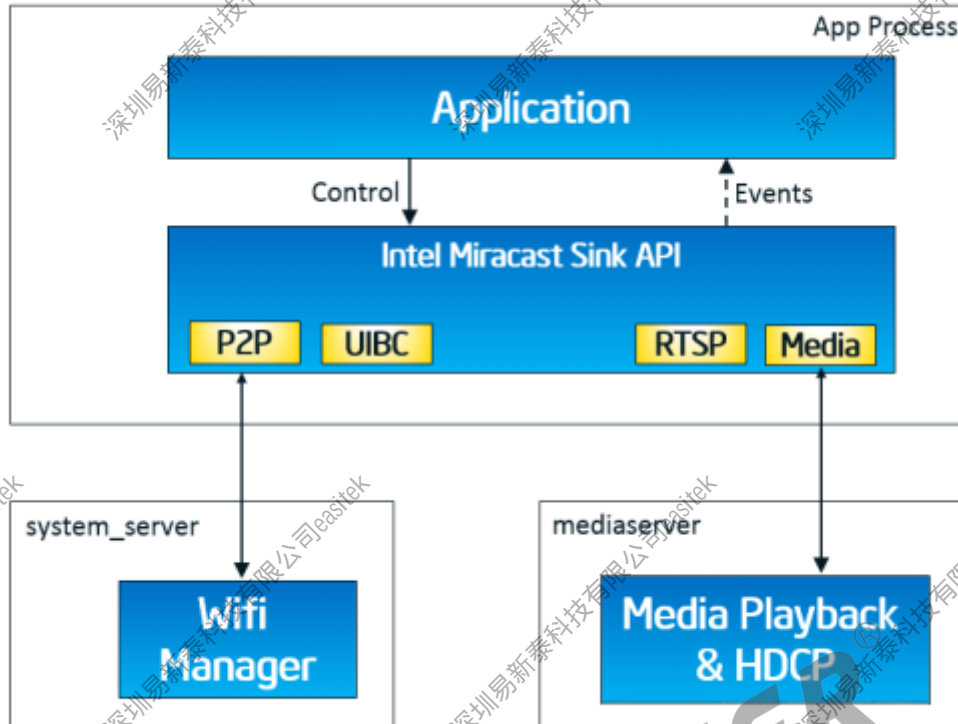


图 2-9: 实现框架结构图

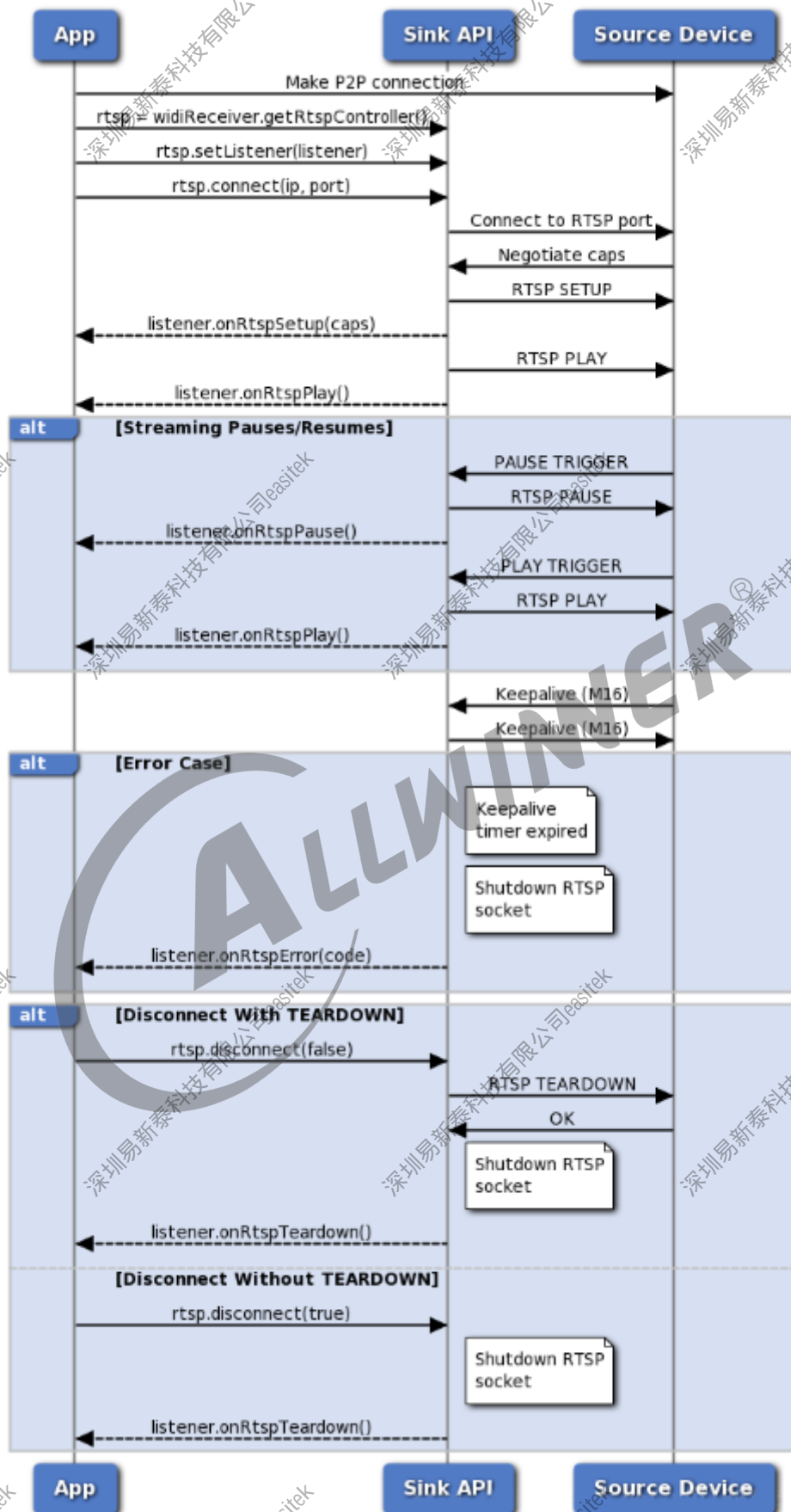


图 2-10: RTSP 会话流程图

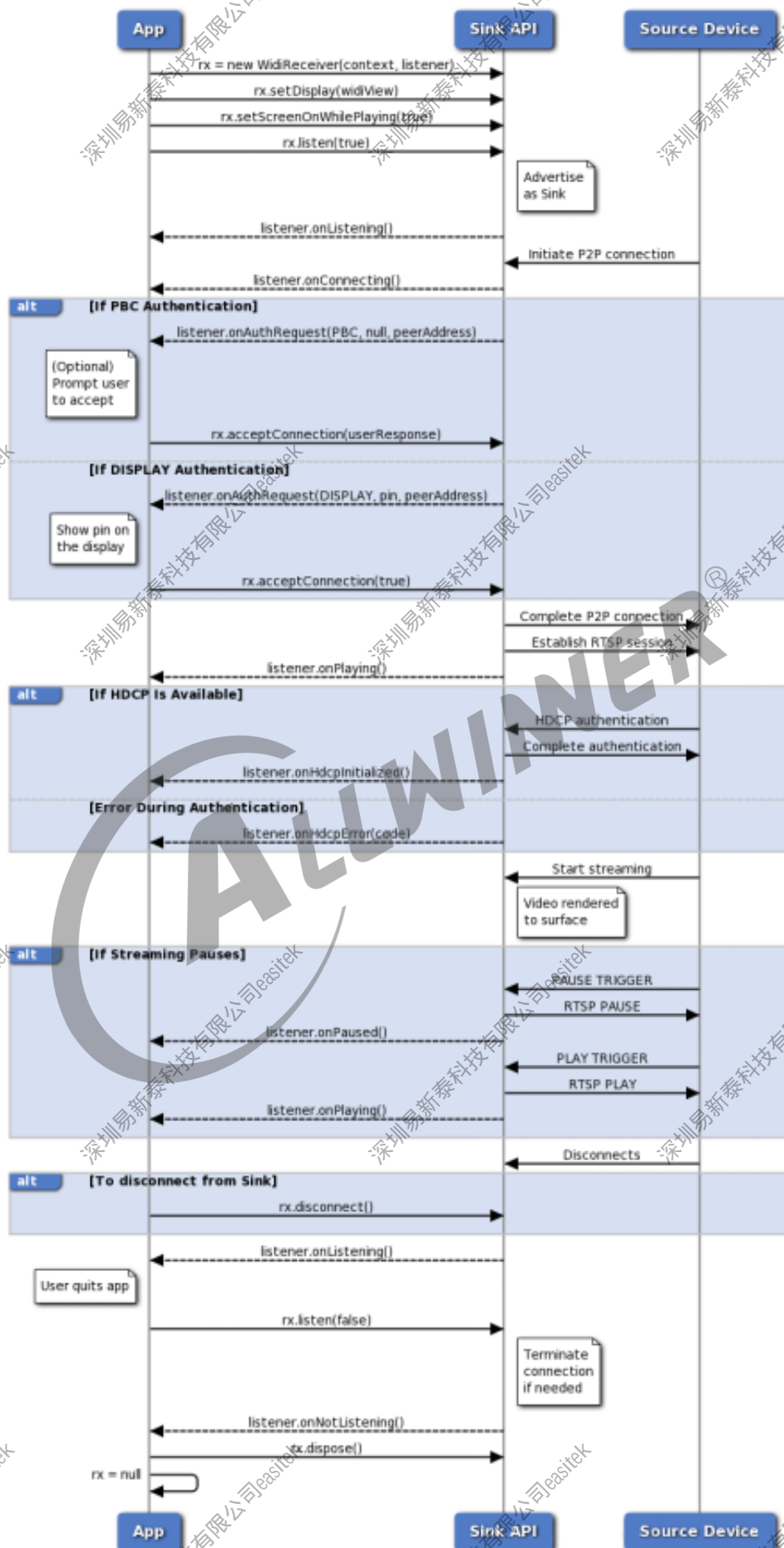


图 2-11: WFD 控制流程图

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。WFD 控制流程图给出了 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。

3 Miracast 使用说明

前提：需保证 Source 端和 Sink 端设备都在同一网段内。

Sink 端准备：

Sink 端多为一些投屏产品，这里以全志的投屏产品进行 Sink 端的连接说明。

1. 修改设备名称：

这一步的目的是设置一个个性化的设备名字，Source 端进行设备搜索时更加容易发现 Sink 设备，

依次进入 setting -> Device Preferences -> About -> Device name -> change，在 change 这一步中，可以选择一些默认的设备名称，也可以选择 Enter custom name，然后输入你想要的设备命名。

2. WiFi 连接：

依次进入 setting -> Network & Internet -> Wi-Fi，选择你要连接的 WiFi，输入密码，连接成功，返回主页面。

3. 打开 sink 端的 MiracastReceiver 应用：

等待 Source 端发送连接请求。

Source 端准备：

1. 网络连接：

依次进入机器的 setting -> Network & Internet -> Wi-Fi

选择与 Sink 端相同 WiFi，输入密码，连接成功，这是为了保证 Source 端与 Sink 端在同一网段内。

2. 搜索 Sink 端设备：

进入 setting，选择 Connected devices，

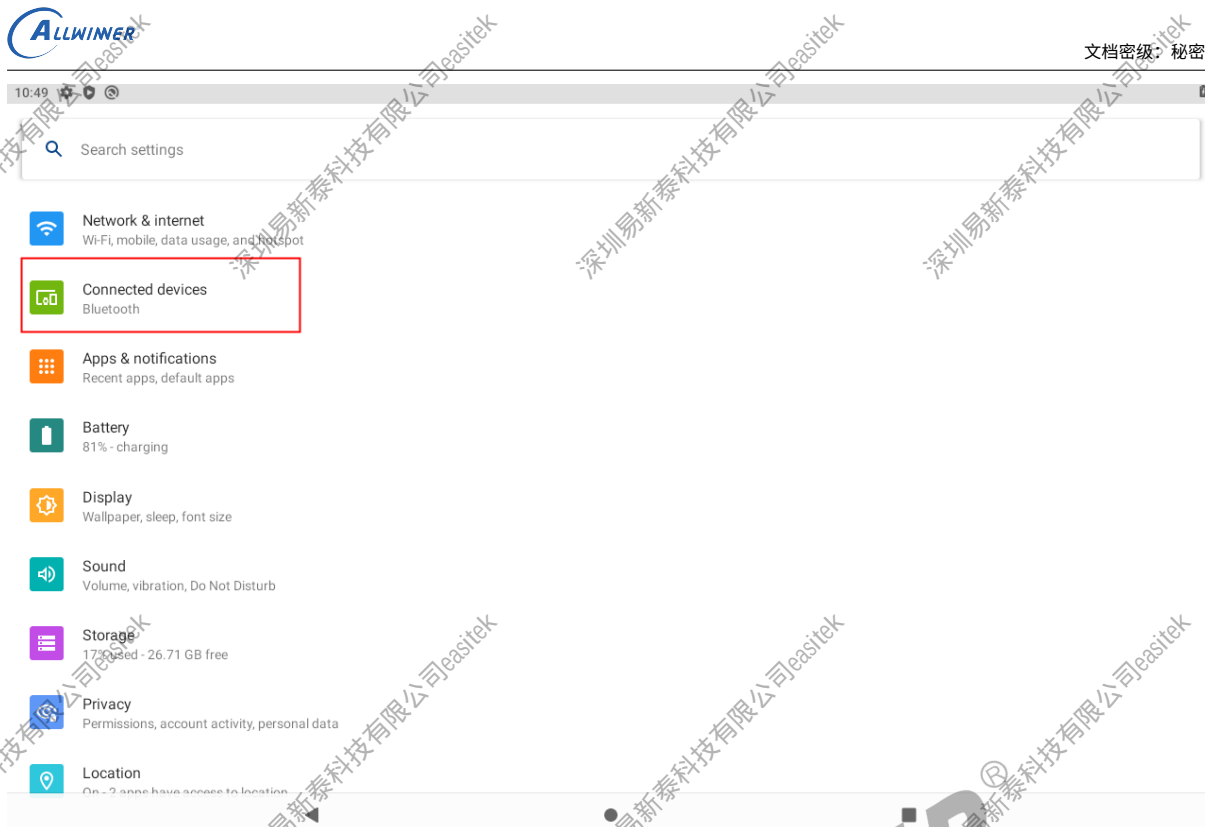


图 3-1: Source 设备的设置图

进入 Connected devices 后, 选择 Connection preferences,



图 3-2: Source 设备的设备连接

然后在 Connection preferences 中选择 Cast，



图 3-3: Source 设备的 cast 图

在 Cast 中，将 Enable wireless display 打开，



图 3-4: Source 设备启用无线显示

启用无线显示后，即可在列表中看到可连接的设备，选择需要连接的设备。



图 3-5: Source 搜索可连接的设备

3. Sink 连接邀请：

此时 Sink 端会弹出一个提示窗口：Invitation to connect，选择 ACCEPT。

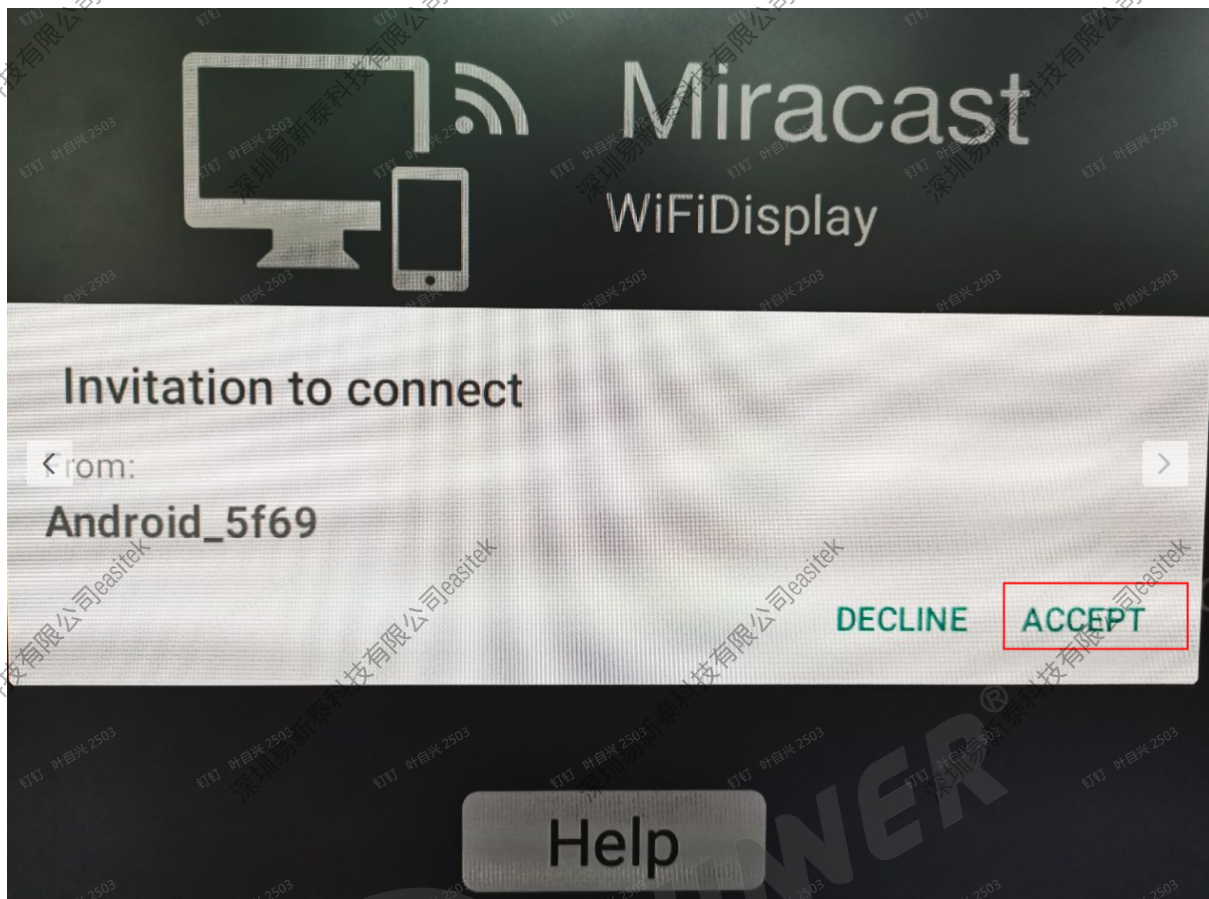


图 3-6: Sink 设备邀请连接

4. Source 端邀请连接：

选择完 Sink 端连接邀请后，此时 Source 端会弹出一个窗口：Invitation to connect，同样选择 ACCEPT 即可。

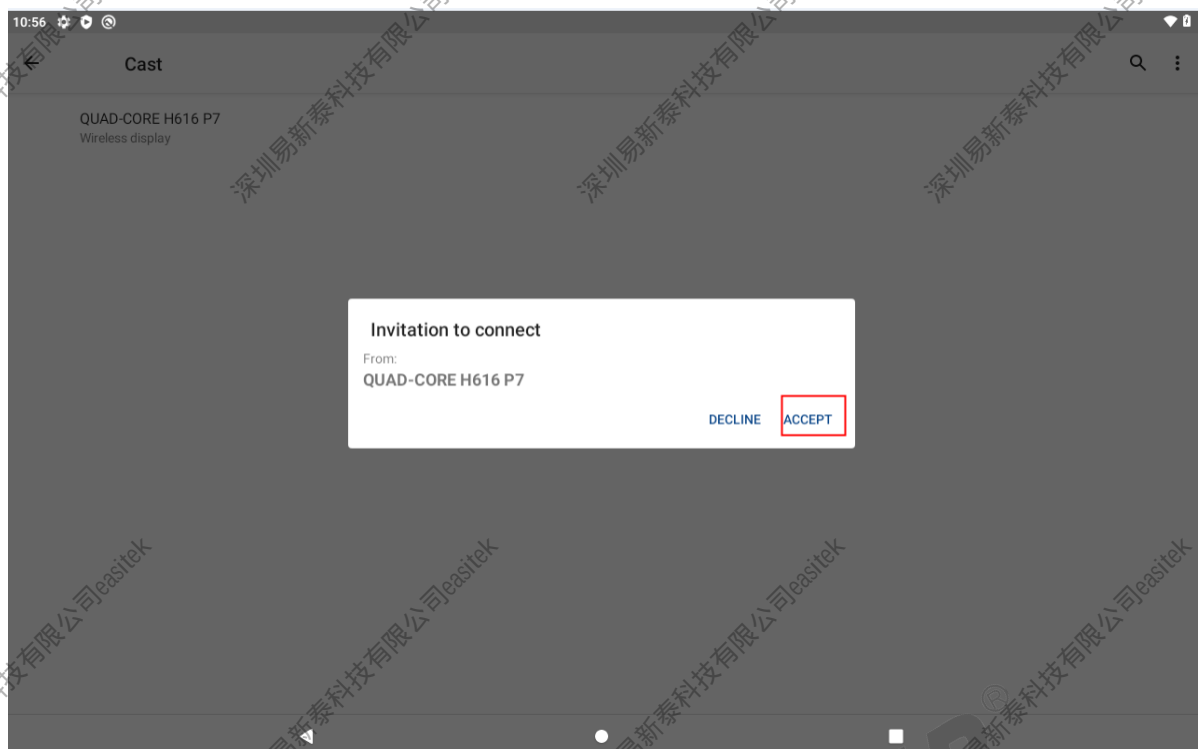


图 3-7: Source 设备邀请连接

最后，投屏连接成功！可以将 Source 端的内容投到大屏幕上啦！

4 附录

4.1 Miracast 移植及其代码调试

4.1.1 Miracast 代码移植

开发过程的第一步是代码移植过程，需要移植的代码可参考 AndroidP、AndroidQ、AndroidR、AndroidS 等 Android 版本的移植提交项目

AndroidP 移植补丁链接：

P: <http://gerrit.allwinnertech.com:8081/#/c/79381/>

AndroidQ 移植补丁链接：

Q: <http://gerrit.allwinnertech.com:8081/#/c/93863/>

AndroidR 移植补丁链接：

R: <http://gerrit.allwinnertech.com:8081/#/c/135911/>
R: <http://gerrit.allwinnertech.com:8081/#/c/136003/>
R: <http://gerrit.allwinnertech.com:8081/#/c/135909/>

AndroidS 移植补丁链接：

S: <http://gerrit.allwinnertech.com:8081/#/c/platform/pdk/platform/frameworks/av/+/163599/>
S: <http://gerrit.allwinnertech.com:8081/#/c/platform/pdk/platform/frameworks/av/+/165559/>
S: <http://gerrit.allwinnertech.com:8081/#/c/platform/pdk/platform/prebuilts/abi-dumps/vndk/+/165016/>

上述几版代码移植中，AndroidQ 版本的移植因为一些编译问题，开发者将 foundation 文件夹的代码 copy 了一份，这样会导致代码的冗余，不建议这么做，后续开发者可参考 AndroidR、AndroidS 的移植。

另外需要注意的是，代码移植需要增加 CAPTURE_AUDIO_OUTPUT 和 RECORD_AUDIO 权限，音频数据才能正常被录制，否则接收端将无声音输出。参考链接如下：

<http://gerrit.allwinnertech.com:8081/#/c/platform/tidy/platform/frameworks/base/+/108404/>
<http://gerrit.allwinnertech.com:8081/#/c/platform/tidy/platform/frameworks/av/+/168022/>

4.1.2 Miracast 代码编译报错修复

- 1. 代码移植完之后接下来是编译过程，这部分大多是头文件缺失或是头文件的路径发生了变化导致找不到头文件和某些变量导致的问题。这部分报错较多，需要一一解决。

- 2. 在 AndroidR 上遇到 android.hardware.IHDCPObserve 识别不到的错误，参考补丁：

<http://gerrit.allwinnertech.com:8081/#/c/135909/>

- 3. 在移植过程中因为增加了相关文件的接口，VNDK 库可能会发生变化，不能保证 ABI 兼容性，报错如下图所示。此时需将 new 中的文件 copy 一份至 old 路径中，但是提交中需要修改有关文件的路径，因为 SDK 中的路径为绝对路径，提交时为相对路径，可参考补丁：

<http://gerrit.allwinnertech.com:8081/#/c/136003/>

```
oe_a:readonly:sb_a --partition system_b:readonly:sb_b --partition vendor_a:readonly:sb_a --partition vendor_b:readonly:sb_b --partition product_a:readonly:sb_a --partition product_b:readonly:sb_b
out/target/product/keres-b3/super_empty.img
2020-10-28 11:29:26 - common.py - INFO :
2020-10-28 11:29:26 - build_super_image.py - INFO : Done writing image out/target/product/keres-b3/super_empty.img
[ 60% 9319/9492] target OX: AWCamera
Stripped invalid locals information from 2 methods.
In out/target/common/obj/APPS/AWCamera_intermediates/dex/d8_input.jar:android/support/v4/media/AudioFocusHandler$AudioFocusHandlerImpl$Base$NoisyIntentReceiver.class:
Methods with invalid locals information:
void android.support.v4.media.AudioFocusHandler$AudioFocusHandlerImpl$Base$NoisyIntentReceiver.onReceive(android.content.Context, android.content.Intent)
Type information in locals-table is inconsistent. Cannot constrain type: INT for value: v37(usage) by constraint OBJECT.
In out/target/common/obj/APPS/AWCamera_intermediates/dex/d8_input.jar:android/support/v7/widget/Toolbar.class:
Methods with invalid locals information:
void android.support.v7.widget.Toolbar.onLayout(boolean, int, int, int, int)
Type information in locals-table is inconsistent. Cannot constrain type: INT for value: v419 by constraint OBJECT.
Some warnings are typically a sign of using an outdated Java toolchain. To fix, recompile the source with an updated toolchain.
[ 77% 7314/9464] /frameworks/av/media/libstagefright/foundation/libstagefright_foundation_header_abi_diff libstagefright_foundation.so.abidiff
FAILED: out/soong/.intermediates/frameworks/av/media/libstagefright/foundation/libstagefright_foundation/android_vendor.30_arm_armv7-a-neon-cortex-a7_shared/libstagefright_foundation.so.abidiff
(prebuilt/clang-tools/linux-x86_64/bin/header-abi-diff -allow-unreferenced-changes -allow-unreferenced-elf-symbol-changes -lib libstagefright_foundation -arch arm -o out/soong/.intermediates/frameworks/av/media/libstagefright_foundation/android_vendor.30_arm_armv7-a-neon-cortex-a7_shared/libstagefright_foundation.so.abidiff -new out/soong/.intermediates/frameworks/av/media/libstagefright_foundation/android_vendor.30_arm_armv7-a-neon-cortex-a7_shared/libstagefright_foundation.so.lsdump -old prebuilt/abi-dumps/vndk/30/64/arm_armv7-a-neon-source-based/libstagefright_foundation.so.lsdump) || (echo '
References with: $ANDROID_BUILD_TOP/development/vndk/tools/header-checker/utls/create_reference_dumps.py -l libstagefright_foundation' && (mkdir -p $D1ST_DIR/abidiffs && cp out/soong/.intermediates/efright/foundation/libstagefright_foundation/android_vendor.30_arm_armv7-a-neon-cortex-a7_shared/libstagefright_foundation.so.abidiff $D1ST_DIR/abidiffs/) && exit 1)
')
error: VNDK Library: libstagefright_foundation's ABI has EXTENDING CHANGES Please check compatibility report at: out/soong/.intermediates/frameworks/av/media/libstagefright_foundation/libstagefright_
0_arm_armv7-a-neon-cortex-a7_shared/libstagefright_foundation.so.abidiff
error: Please update ABI references with: $ANDROID_BUILD_TOP/development/vndk/tools/header-checker/utls/create_reference_dumps.py -l libstagefright_foundation
[ 77% 7350/9464] /frameworks/base/packages/SystemUI/SystemUI-core.kapt
warning: some .JAR files in the classpath have the Kotlin Runtime library bundled into them. This may cause difficult to debug problems if there's a different version of the Kotlin Runtime library in
oving these libraries from the classpath
out/soong/.intermediates/frameworks/base/packages/SettingsLib/SettingsLib/android_common/turbine-combined/SettingsLib.jar: warning: library has kotlin runtime bundled into it
out/soong/.intermediates/prebuilt/tools/common/m2/kotlinx-coroutines-core/android_common/turbine-combined/kotlinx-coroutines-core.jar: warning: library has kotlin runtime bundled into it
11:32:05 ninja failed with: exit status 1
```

图 4-1：移植编译报错图

- 4. 在 AndroidS 中遇到同样的 VNDK 异常报错问题，排查发现在 WiFiDisplay 中部分文件与 android 的 foundation 是重复的，这才导致该问题，保留一份即可，可以参考如下补丁：

<http://gerrit.allwinnertech.com:8081/#/c/platform/pdk/platform/frameworks/av/+/165559/>

- 5. 待编译通过后编出固件烧录至平板中可进行测试。测试时可用盒子和平板作为接收端和发送端进行，二者连共同 wifi，盒子端打开 MiracastReceiver，平板端 Settings/Connected devoces/Connection preferences/Cast 右上角选择 Enable wireless display 选项，此界面会出现盒子的名称，点击即可建立连接。注：平板烧机开机之后先 adb root、adb remoun、setenforce 一番，如果没有这些操作的话会因为一些权限问题导致链接不上。与 Miracast 相关的上层代码在 /androidR/frameworks/base/services/core/java/com/android/server/display/WifiDisplayController.java，投屏的大步骤都在这里，可以在这里进行打印调试，此外就是 wifi-display 目录下的代码，Miracas 的主要功能大部分都在这里。

著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。