Android Miracast
开发指南

版本号: 1.2
发布日期: 2024.3.8

版本历史

版本号	日期	制/修订人	内容描述
1.0	2023.03.09	AWA1782	基础文档。
1.1	2023.11.21	AWA1782	1. 修改 miracst 使用说明图，使用最新 Android14 的操作图。 2. 修改 miracast 使用说明相关描述。 3. 添加“相关术语”描述介绍。 4. 增加 WifiDisplaySource 数据流图和内部类相关描述。 5. 删除 wifidisplay 网络连接流程图。 6. 增加调试方法章节。 7. 增加 FAQ 章节。 8. 修改 MiracastSink 端框图。
1.2	2024.3.8	AWA1782	1. 更新 wfd 相关说明框图。 2. 增加 miracast 连接拓扑描述。 3. 增加 miracsat sink 端代码路径和库介绍。 4. 增加相关配置项 5. 删除 wifidisplay 网络连接流程图。 6. 增加 miracast sink 端的丢包分析调试。 7. 增加 miracast source 源数据和编码数据获取调试方法描述。

目　录

插　图

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 开发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。

1.4 相关术语

- Miracast：Miracast 是由 Wi-Fi 联盟于 2012 年所制定的，以 Wi-Fi 直连（Wi-Fi Direct）为基础的无线显示标准。支持此标准的 3C 设备可透过无线方式分享视频画面。因此，Miracast 又叫 WIFI-DISPLAY，在谷歌源码中该模块的命名也是 wifi-display。
- Wi-Fi Direct：是一个 WIFI 协议标准，Wi-Fi Direct 标准允许无线网络中的设备无需通过无线路由器即可相互连接。与蓝牙技术类似，这种标准允许无线设备以点对点形式互连，与蓝牙相比，数据传输速度更快。
- RTP：全称 Real-time Transport Protocol，是一种用于实时传输音频和视频数据的协议。Miracast 通过 RTP 来传输 MPEG2-TS 流。
- RTSP：全称 Real-Time Streaming Protocol，是一种用于实时流媒体传输的网络协议，它提供了一种灵活和可扩展的方式来控制和传输实时流媒体数据，并通过配合 RTP 和 RTCP 协议实现了流媒体的传输和控制功能。
- RTCP：全称 Real-time Transport Control Protocol，是一种用于实时流媒体传输的控制协议，它是 RTP 的补充协议。RTCP 通过在 RTP 会话中传输控制和监控信息，提供了对实时流媒体传输的控制和质量反馈。
- MPEG2-TS：全称 MPEG-2 Transport Stream，是一种用于传输音频、视频和数据的标准协议。它具有容错、多路复用和多种媒体格式支持等特点，被广泛应用于数字电视广播和流媒体传输中。

- WFD Source：WFD Source 是投屏的发送端，即将作为 source 的设备会将自己的音视频图像投射到接收端设备上，常见于平板，手机等便携设备。
- WFD Sink：WFD Sink 是投屏的接收端，负责接收 source 传输的音视频数据并进行解码输出。常见于电视，盒子等大屏设备。
- UDP：全称 User Datagram Protocol，是一种无连接的、不可靠的传输层协议。它常用于实时应用，如实时音频和视频传输、在线游戏等。

2 Miracast 功能简介

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 56Mbps 提升到 300 甚至 600Mbps。WMM 是 Wi-Fi Multimedia 的缩写，是一种针对实时视音频数据的 QoS 服务。而 WPA2 意为 Wi-Fi Protected Acess 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。

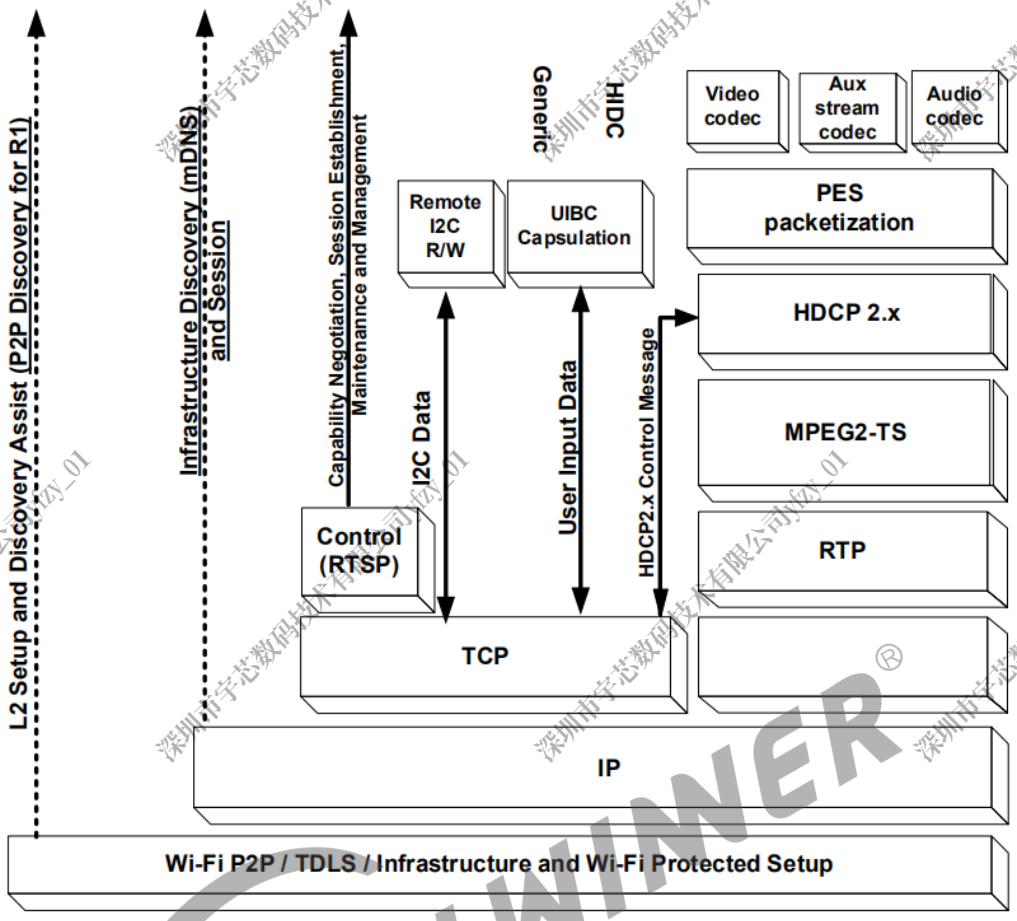

图 2-1: WFD 涉及的技术及协议框图

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 （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 （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（内容保护，即数据加密，对传输的内容做加密保护（HDCP），该步骤为可选实现）
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 (WFD Source 和 Sink 的休眠待机，该步骤为可选实现)
11. WFD Session Teardown （会话终止）

如下是 Wi-Fi Display 官方文档中关于 RTSP 会话建立，配对，协商，到最终 A/V 流传输的时序流程图。

- 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。

一般来说，我们要对投屏的分辨率，编码格式等数据进行调试，可以抓取 M3/M4 阶段的参数，来确认最终协商的配置是否符合预期。

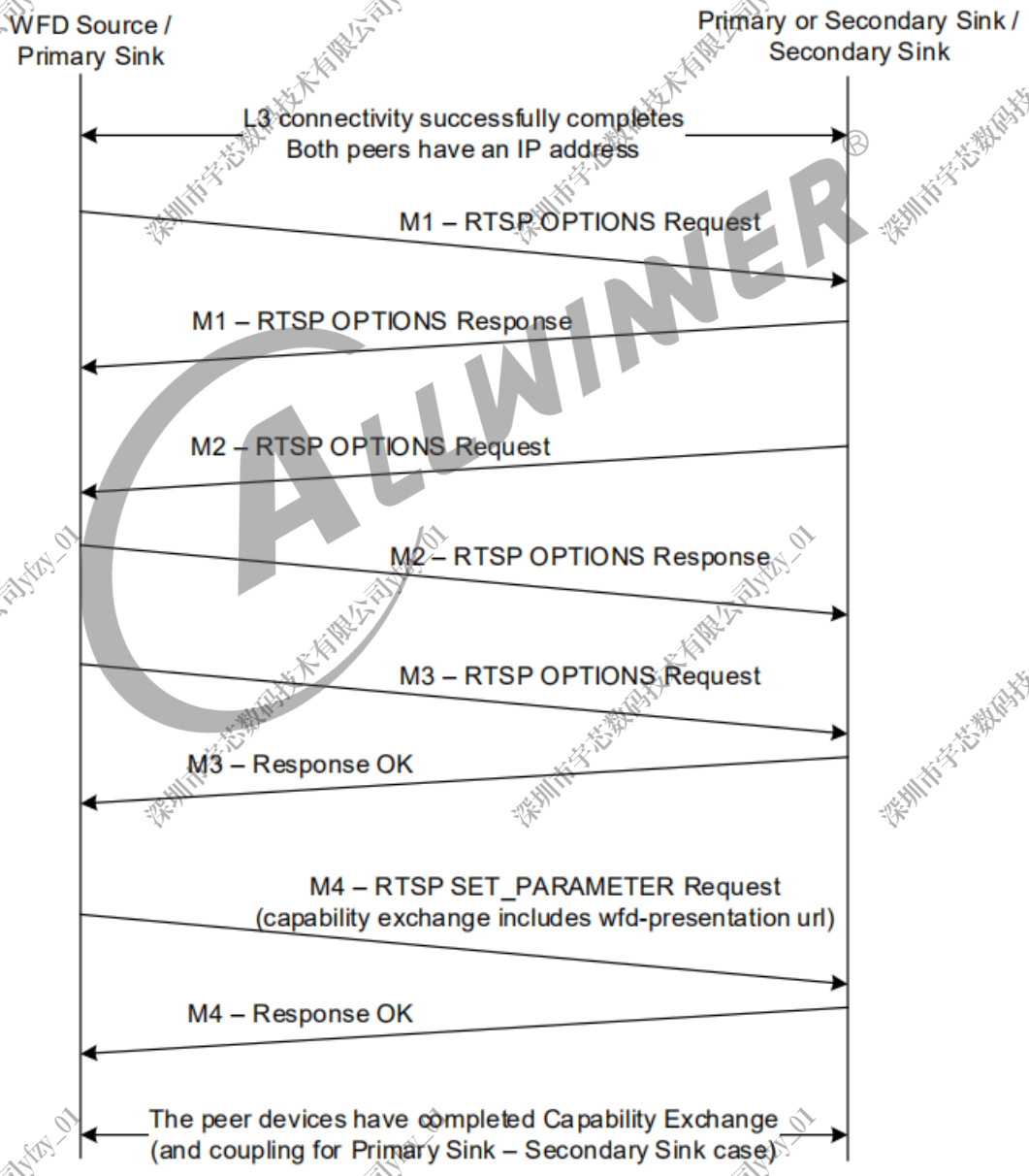

Figure 13. WFD capability negotiation (or coupling) flow using RTSP

图 2-2: 会话协商过程图

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。

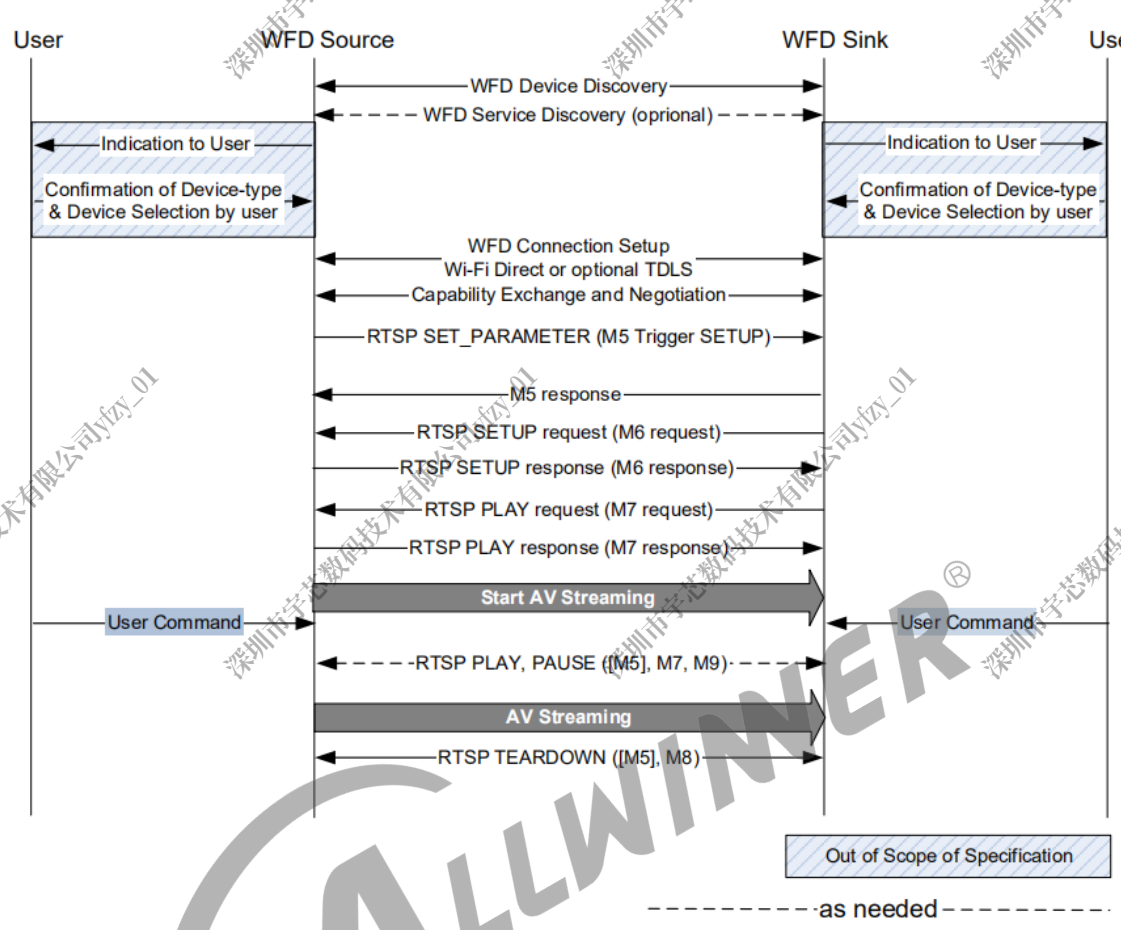

Figure 14. Time-line of a WFD session

图 2-3: RTSP 协议控制图

开发者可以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 /sdcard/test.dump
```

具体的协议包相关的内容如前面 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。

2.2 Miracast 简易连接拓扑

如下为一个 WFD 的 source 端和一个 WFD 的 sink 端进行连接，传输 A/V 流和控制信息的拓扑图。

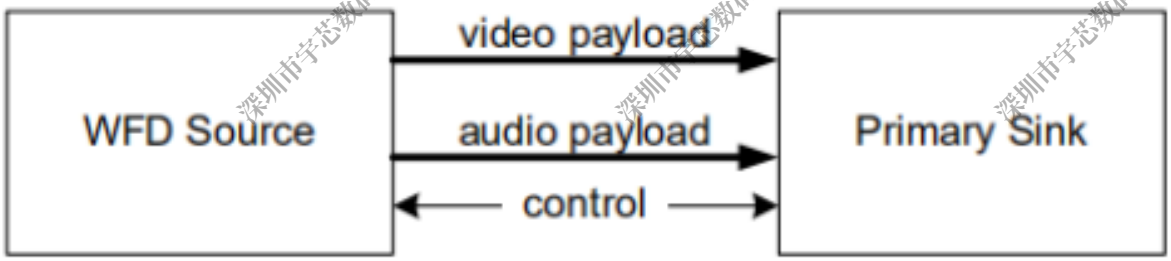

图 2-4: Miracast 一对一连接拓扑

Source 端从显示管理系统获取的镜像数据经过音视频编码（H264），然后进行 HDCP 加密和 PES packetization 及 TS 流化（转换为 TS 流）后，最后打包成 RTP 包经过 UDP 通道发送到 SINK 端；Source 常见于平板，手机等便携设备。

SINK 端要经过相反的处理过程，从 UDP 通道接收 RTP 包，然后进行 TS 解析和 PES 去 packetization 化和 HDCP 解密，最后送给解码器进行解码，然后将解码后的数据直接显示。Sink 常见于电视，盒子等大屏设备。

2.3 WifiDisplay 会话管理模型

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。

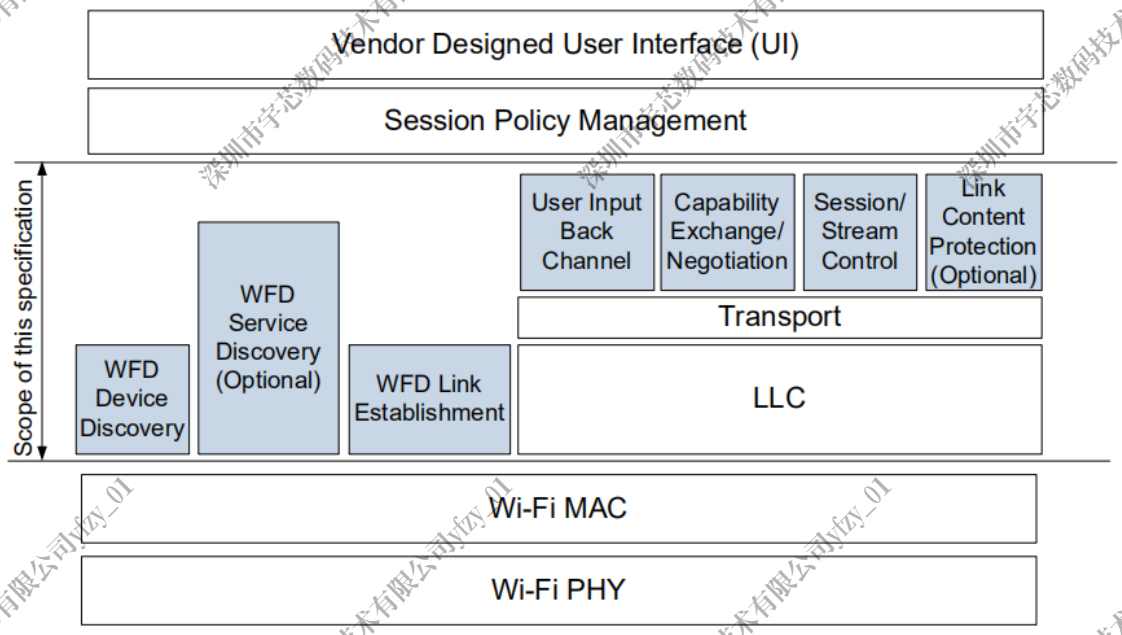

图 2-5: WFD 设备会话管理的模型

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 端作为接收端只需要绑定 RTP 数据传输端口并接收来自 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，音视频的传输通过 MPEG TS 协议作为传输载体。

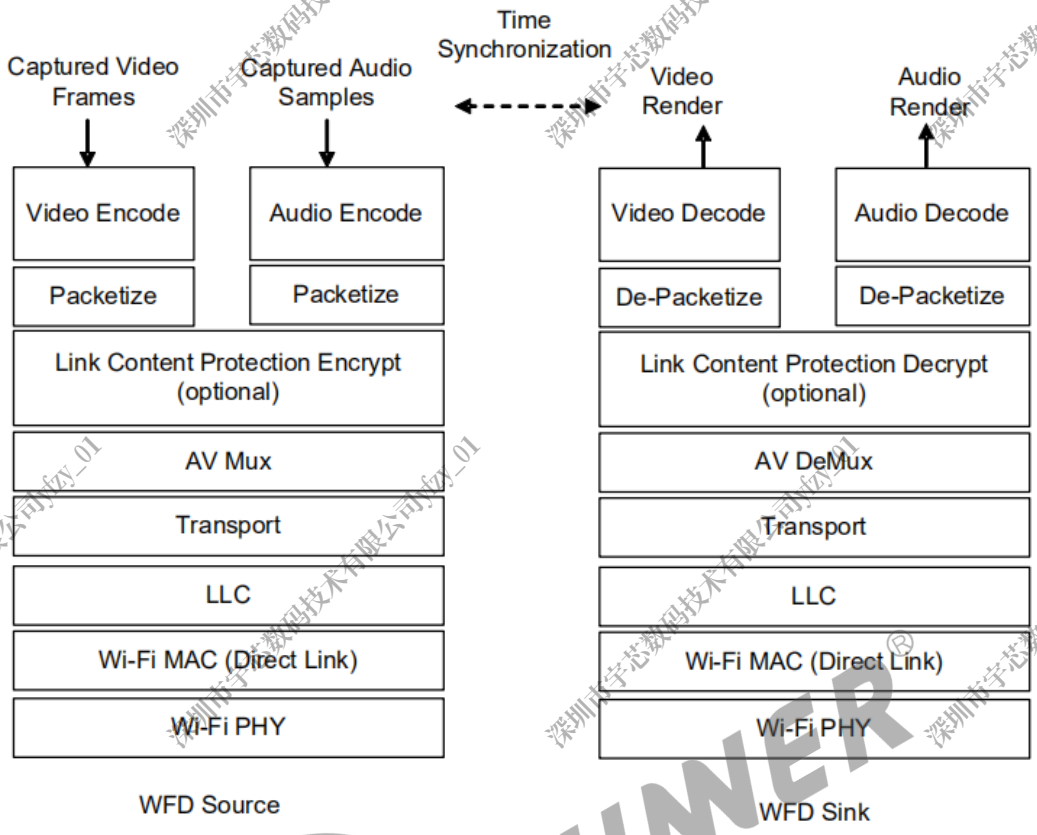

图 2-6: 音频及视频流控制模型

3 Miracast Android 实现

Miracast 在 android 上分为 source 发送端以及 sink 接收端，不同产品包下默认预置支持的 miracast 类型也不同，如下所示：

- source 端：平板默认支持
- sink 端：OTT 盒子、TV、平板上默认支持

3.1 WFD Source 端实现

为了实现 WifiDisplay 的发送端功能，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。

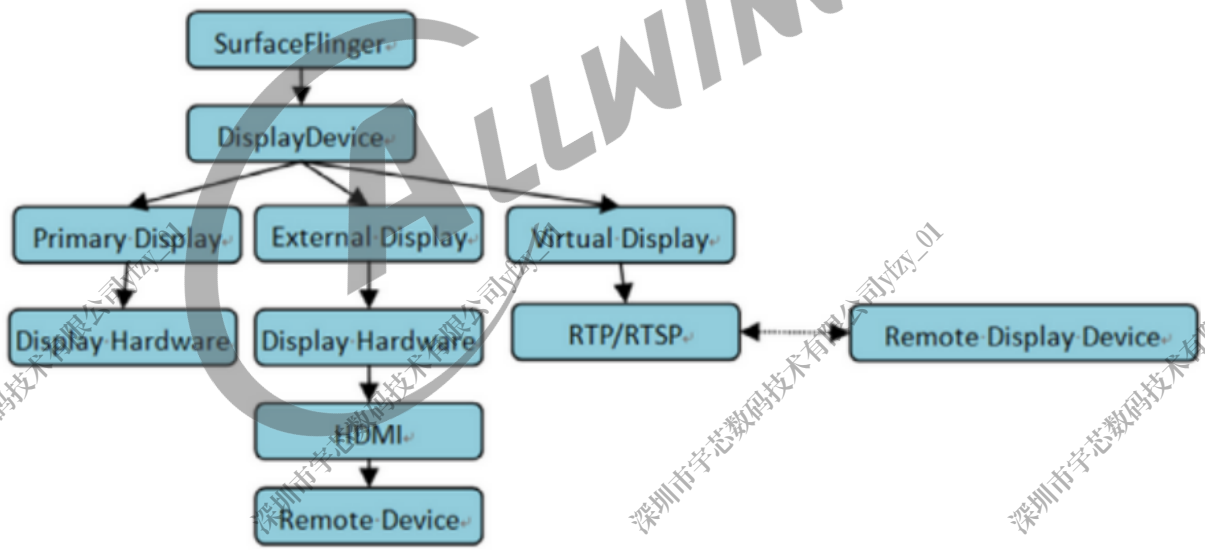

图 3-1: DisplayDevice 的隔离示意图

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，需要更改一个如下项。

```
<bool name="config_enableWifiDisplay">true</bool>
```

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。

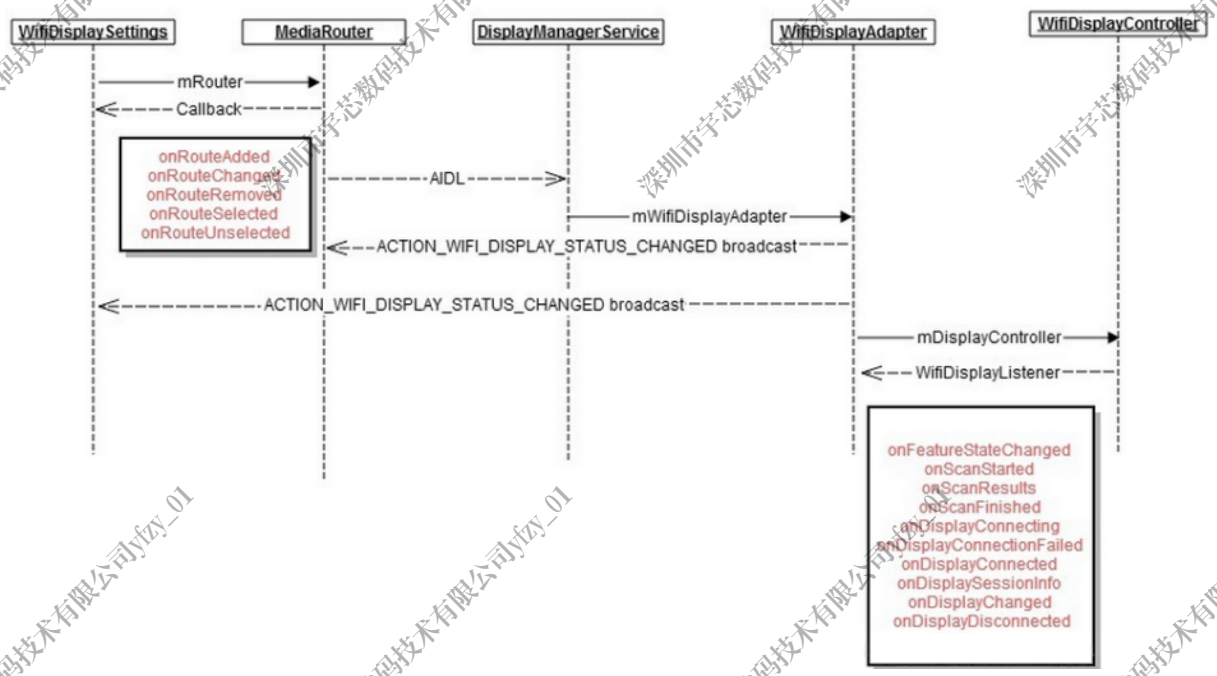

图 3-2: 设备发现流程图

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 WifiDisplay-Settings 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。

关于投屏中间件部分，在系统源码frameworks/av/media/libstagefright/wifi-display目录下，这里面包括 PlaybackSession.cpp、MediaPuller.cpp、Converter.cpp、TSPacketizer.cpp、RTPSender.cpp 等 C++ 类文件和相应的头文件，分别负责 Source 端的会话过程管理、镜像媒体读取、编码、TS 打包及 RTP 打包发送等过程。

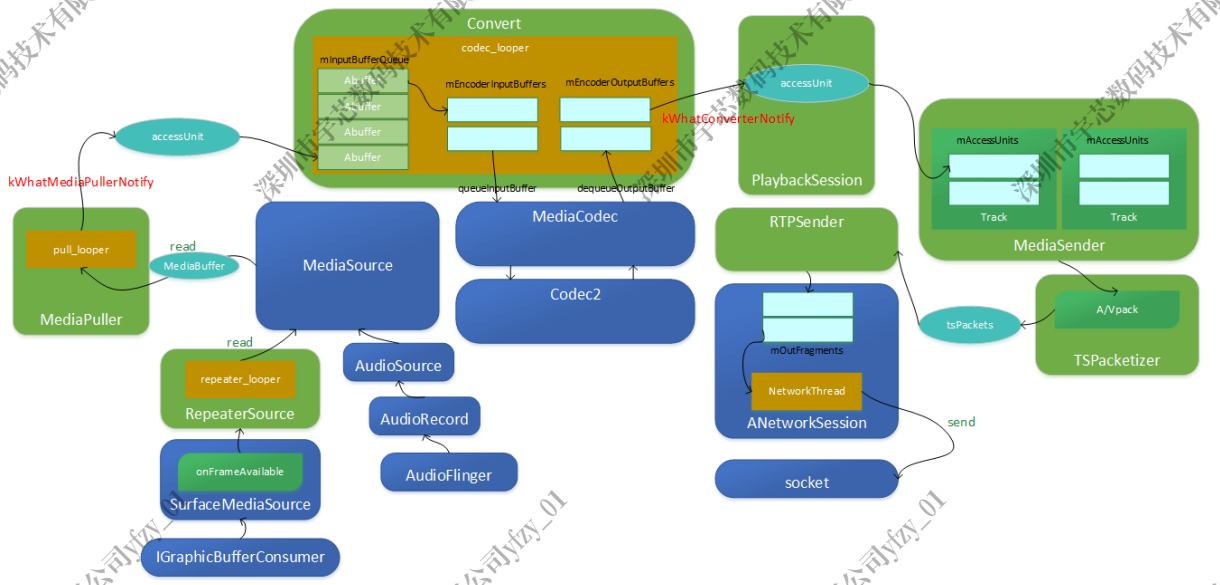

图 3-3: miracast 数据流

- WifiDisplaySource：负责 Source 端的协议交互
- PlaybackSession：负责会话过程管理
- MediaPuller：负责镜像媒体流读取
- Converter：负责媒体编码，对象中包括一个 MediaCodec 对象具体负责编码过程
- MediaSender：负责 RTP 打包发送
- TSPacketizer：负责 TS 流打包

上述是 WifiDisplay 相关的功能模块，而 WifiDisplay 的入口实际是 WifiDisplaySource.cpp 的类，当 P2P 连接建立后，WifiDisplayControleIr 会创建一个 WifiDisplaySource 的实例，而 WifiDisplaySource 的内部主要工作就是负责完成整个 RTSP 过程，同时创建多媒体流投屏播放的一个会话实例：PlaybackSession，该会话实例一旦创建，则 source 端开始进行音视频数据获取，编码和输出 ts 流的动作。

3.2 WFD 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 MiracastRecevier 应用通过 JNI 接口与底层 WFDPlayer 交互，WFDPlayer 内部实际是类似播放器的功能，从 WIFI 底层获取到 RTP 数据后，进行解封装和解码，随后送往渲染组件进行渲染，WFDSinkView 继承 SurfaceView 的实现，最终投屏输出渲染在上面。

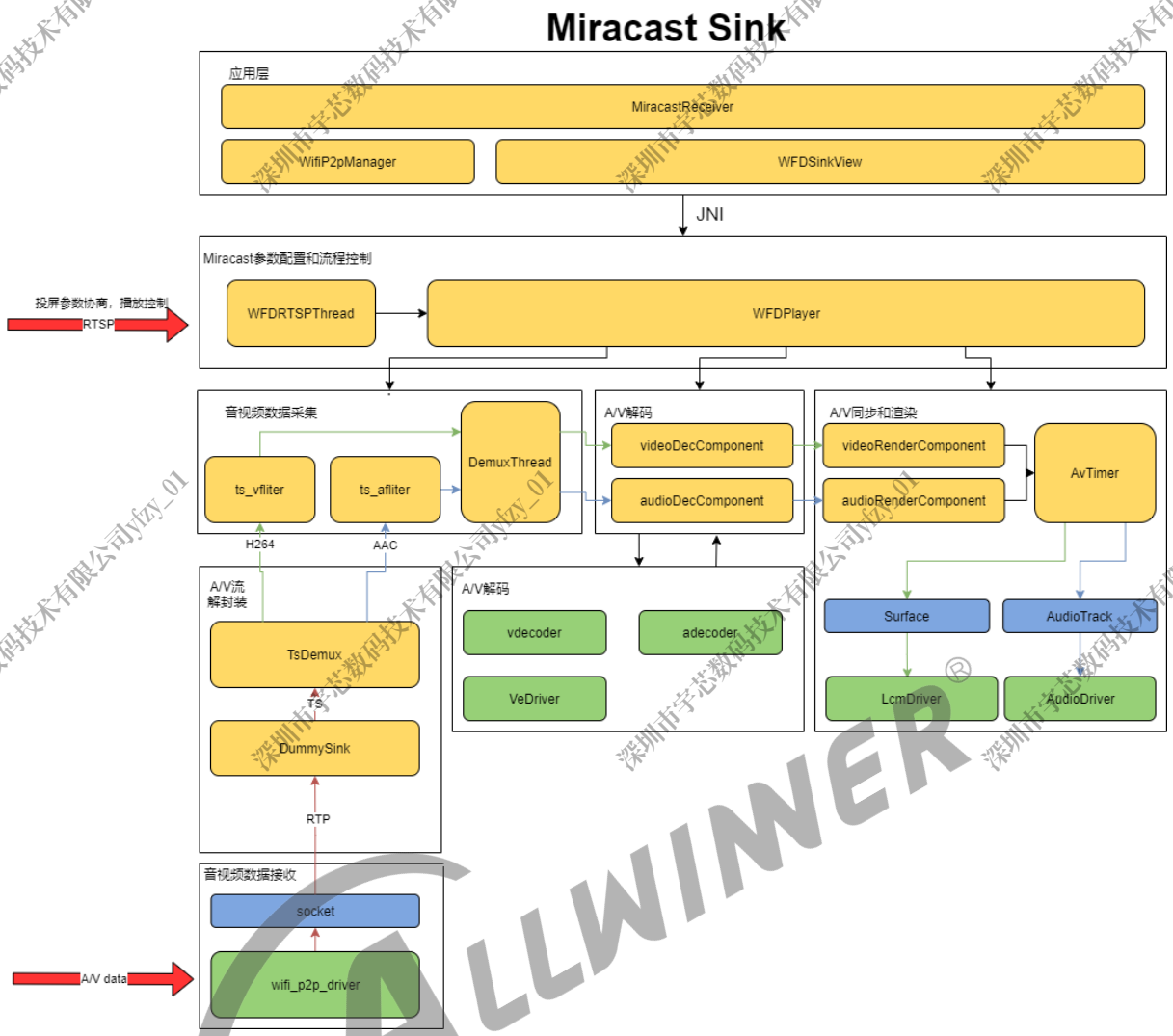

图 3-4: 实现框架结构图

Miracast Sink 端目前是以闭源库和 apk 的形式预置在系统 SDK 的vendor/aw/public/prebuild/apk/MiracastReceiver路径下，具体包含的软件包如下：

- MiracastReceiver.apk：投屏接收端 app 应用
- libcdv_output.so：负责渲染音视频渲染输出
- libcdv_playback.so：负责播放控制、音视频同步的逻辑等
- libwfdrtsp.so：网络协议 rtsp 的具体实现
- libwfdplayer.so：负责播放器创建以及音视频 TS 流解析
- libjni_WFDManager.so：app 到 wfdmanager 的 C++ 层的 JNI 接口层
- libwfdmanager.so：负责底层 wfdplayer 控制和事件回调

MiracastReceiver 应用，如客户需要自定义应用 UI 界面等，可以向全志 FAE 进行申请开放 apk 应用源码，但相关 so 库不对外开源。

4 Miracast 使用说明

前提：需保证 Source 端和 Sink 端设备都在同一局域网环境内，注意这里的系统投屏设置界面以 Android14 为例，其他 Android 版本会存在 UI 显示内容差异，但入口都是一样的。

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. 打开 WiFi 开关：
   依次进入 setting -> Network & Internet -> Wi-Fi，打开 WIFI 开关。
   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. 打开 WiFi 开关：
   依次进入机器的 setting -> Network & Internet -> Wi-Fi，打开 WIFI 开关。
2. 搜索 Sink 端设备：
   进入 setting，选择 Connected devices。

图 4-1: Source 设备的设置图

进入 Connected devices 后，选择 Connection preferences，然后在 Connection preferences 中选择 Cast。

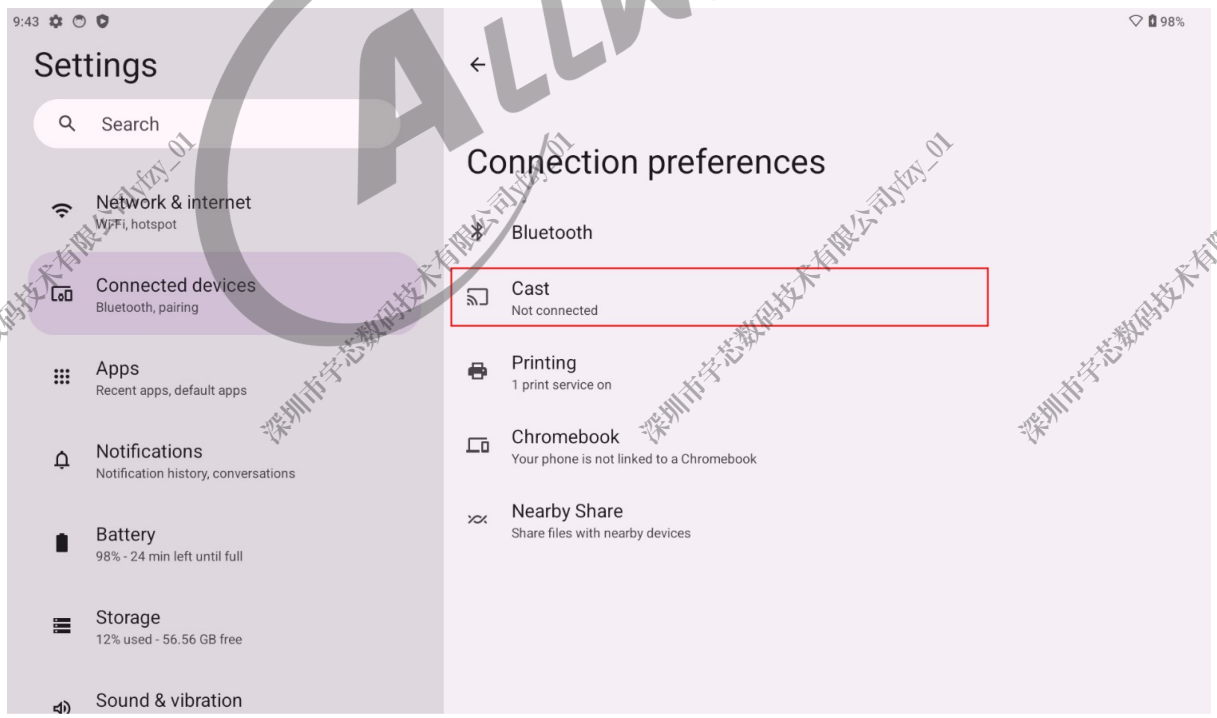

图 4-2: Source 设备的设备连接

在 Cast 中，将 Enable wireless display 打开，此时会自动搜索附近支持 miracast 投屏的设备。

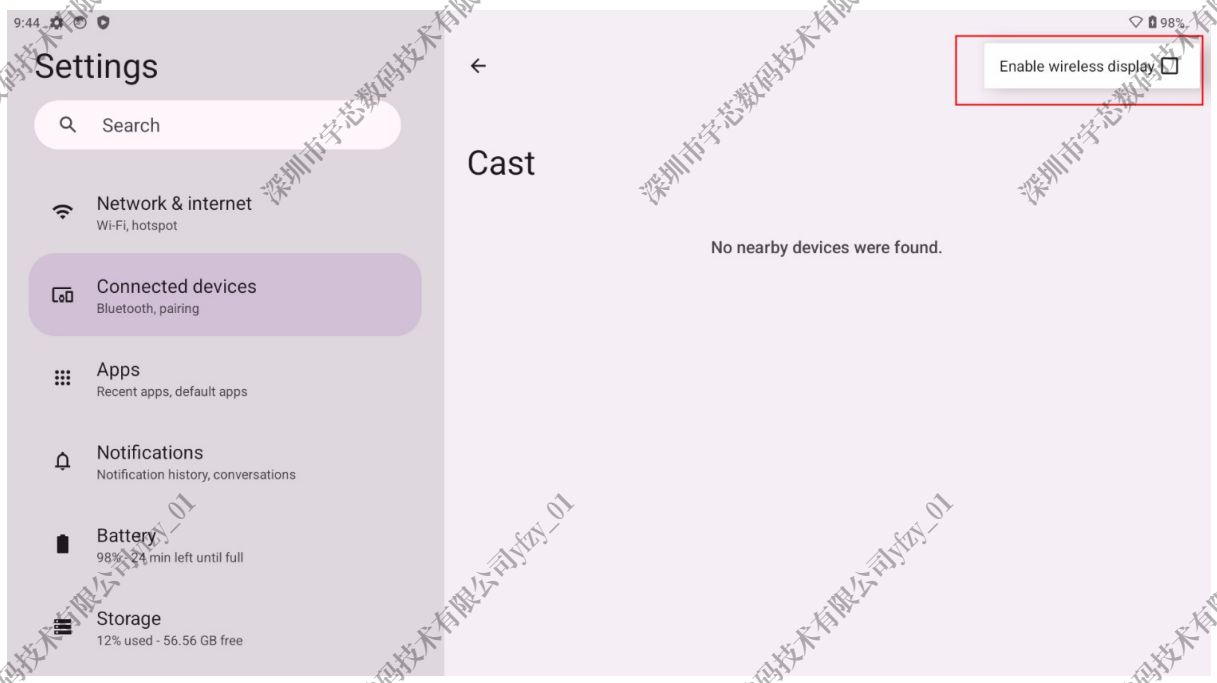

图 4-3: Source 设备启用无线显示

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。

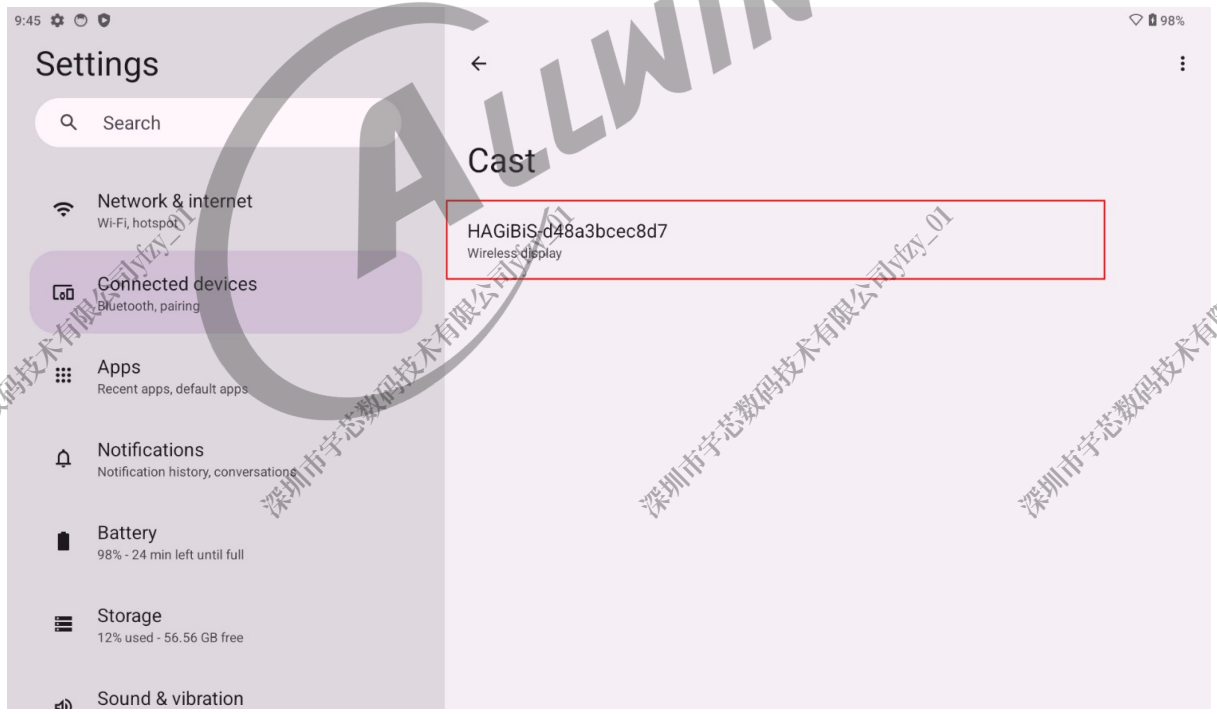

图 4-4: Source 搜索可连接的设备

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。

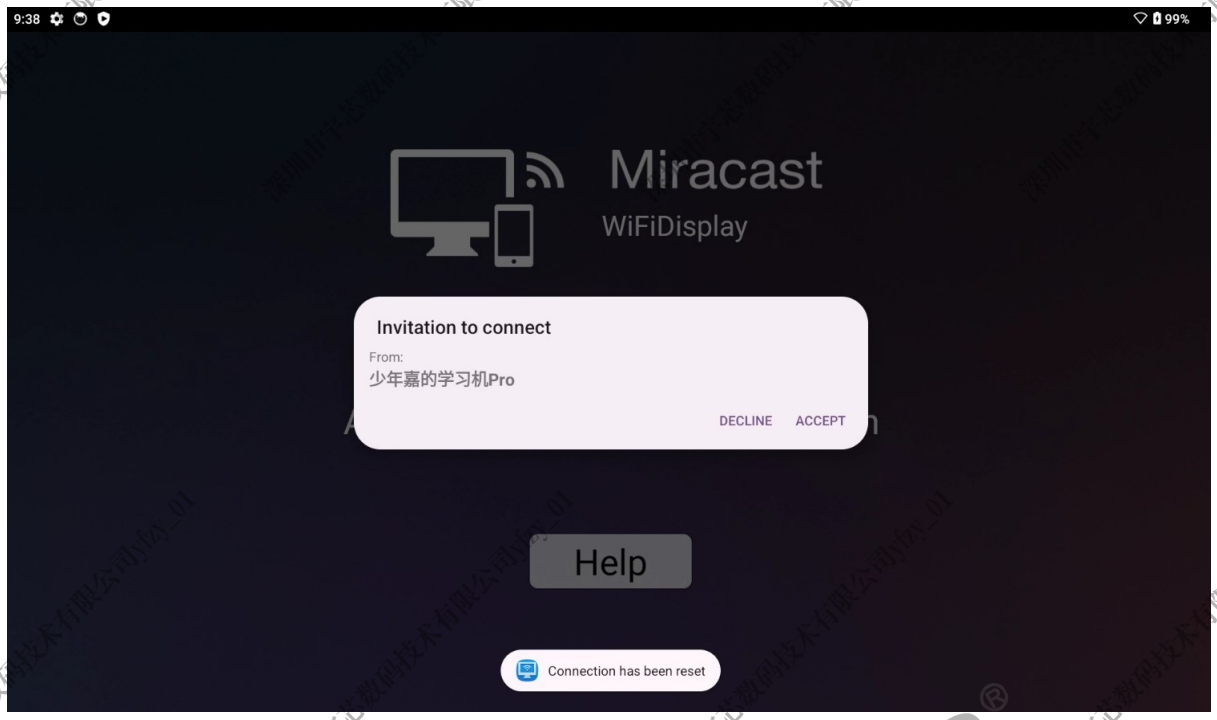

图 4-5: Sink 设备邀请连接

最后，投屏连接成功，可以将 Source 端的内容投到大屏幕上。

5 Miracast 调试方法

5.1 Miracast 发送端

在 Miracast 的整个链路分析过程中，主要涉及的环节是：**设备连接-> 源数据获取-> 数据编码->TS 打包-> 数据发送**，因此我们的调试分析也是集中在这几个环节上，逐步剖析哪个环节的数据出现问题，再结合代码流程深入分析。

5.1.1 分析设备连接

一般出现如下日志，无法连接上 sink 端。

```
06-27 17:30:26.275 I/WifiDisplayController(  575): Timed out waiting for Wifi display RTSP connection after 30 seconds: 客
        厅电视
06-27 17:30:26.275 I/WifiDisplayController(  575): Wifi display connection failed!
06-27 17:30:26.275 I/WifiDisplayController(  575): Retrying Wifi display connection.  Retries left: 2
06-27 17:30:26.275 I/WifiDisplayController(  575): Stopped listening for RTSP connection on 192.168.49.35:7236 from Wifi
        display: 客厅电视
```

这里表示等待 RTSP incoming 超时，这个时候需要排查 sink 端的 wifi p2p 连接状态，一般情况下是 source 端建立和 RTSP 服务的 socket 后，一段时间没有监听到 sink 客户端的连接导致超时关闭了 RTSP 服务，如下为 WifiDisplayController.java 相关代码。

```
// Step 6. Listen for incoming RTSP connection.
if (mConnectedDevice != null && mRemoteDisplay == null) {
    // Use extended timeout value for certification, as some tests require user inputs
    int rtspTimeout = mWifiDisplayCertMode ?
        RTSP_TIMEOUT_SECONDS_CERT_MODE : RTSP_TIMEOUT_SECONDS;
    mHandler.postDelayed(mRtspTimeout, rtspTimeout * 1000);
}

//超时的runnable
private final Runnable mRtspTimeout = new Runnable() {
    @Override
        public void run() {
        if (mConnectedDevice != null
            && mRemoteDisplay != null && !mRemoteDisplayConnected) {
            Slog.i(TAG, "Timed out waiting for Wifi display RTSP connection after "
                + RTSP_TIMEOUT_SECONDS + " seconds: "
                + mConnectedDevice.deviceName);
            handleConnectionFailure(true);
        }
    }
};
```

另外也可以通过 dumpsys display 过滤出 WifiDisplay 信息，通过这里可以看到当前支持 WifiDisplay 的 sink 端设备，以及当前可连接/保存状态，用于定位分析设备无法搜索或者无法连接问题。

```
mWifiDisplayScanRequestCount=0
WifiDisplayAdapter
mCurrentStatus=WifiDisplayStatus{featureState=3, scanState=0, activeDisplayState=0, activeDisplay=null, displays=[荣耀
    智慧屏X2 (12:38:1f:4f:c9:5a), isAvailable false, canConnect false, isRemembered true, QUAD-CORE H618 p2 (b4
    :29:46:9e:67:8f), isAvailable false, canConnect false, isRemembered true, 荣耀智慧屏X2 (12:2d:41:08:cb:1e), alias 荣
    耀智慧屏11, isAvailable false, canConnect false, isRemembered true, 客厅电视 (e2:76:d0:5b:8a:60), isAvailable true,
    canConnect true, isRemembered true, QUAD-CORE tv303 perf1 (18:9f:45:7d:47:b4), isAvailable true, canConnect
    true, isRemembered false], sessionInfo=WifiDisplaySessionInfo:
mWifiDisplayOnSetting=true
```

设备连接过程以及双方 RTSP 协商，还有一些配合分析的 TAG，用于确认双方连接情况和参数选择：

- WifiDisplayController：查看 wifi display 的状态，以及 RTSP 连接跟监听的端口 ip。
- WifiDisplaySource：查看 RTSP 协商的过程（通过 the test xx）以及参数协商后的 video，audio 格式，编码参数等。

```
06-28 15:53:55.451 I/WifiDisplayController(  575): Connecting to Wifi display: 客厅电视
06-28 15:53:55.486 I/WifiDisplayController(  575): Initiated connection to Wifi display: 客厅电视
06-28 15:54:00.287 I/WifiDisplayController(  575): Connected to Wifi display: 客厅电视
06-28 15:54:00.287 I/WifiDisplayController(  575): Stopping Wifi display scan.
06-28 15:54:00.292 I/WifiDisplayController(  575): Listening for RTSP connection on 192.168.49.35:7236 from Wifi display:
    客厅电视
06-28 15:54:00.297 E/WifiDisplaySource( 7246): the_test, 2
06-28 15:54:00.297 E/WifiDisplaySource( 7246): the_test, 4
06-28 15:54:01.056 I/NetworkSession( 7246): incoming connection from 192.168.49.1:41644 (socket 13)
06-28 15:54:01.057 I/NetworkSession( 7246): added clientSession 2
06-28 15:54:01.057 I/WifiDisplaySource( 7246): We now have a client (2) connected.
06-28 15:54:01.117 I/WifiDisplaySource( 7246): Picked video resolution 1280 x 800 p30
06-28 15:54:01.117 I/WifiDisplaySource( 7246): Picked AVC profile 1, level 1
06-28 15:54:01.117 I/WifiDisplaySource( 7246): Using AAC audio.
06-28 15:54:01.117 I/WifiDisplaySource( 7246): Sink does not support content protection.
06-28 15:54:01.405 W/RTPSender( 7246): Huh? Received data on RTP connection...
06-28 15:54:01.405 W/RTPSender( 7246): Huh? Received data on RTP connection...
06-28 15:54:01.405 I/WifiDisplaySource( 7246): Received PLAY request.
06-28 15:54:01.406 I/WifiDisplaySource( 7246): deferring PLAY request until session established.
06-28 15:54:01.408 I/WifiDisplayController(  575): Opened RTSP connection with Wifi display: 客厅电视
06-28 15:54:03.915 I/RTPSender( 7246): lost 99.22 % of packets during report interval.
```

打开 Converter.cpp 文件的调试开关，可以看到建立连接后，初始码率，帧率等信息。

```
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#define LOG_NDEBUG 0
#define LOG_TAG "Converter"
#include <utils/Log.h>

#include "Converter.h"

#include "MediaPuller.h"
#include "media/stagefright/foundation/avc_utils.h"

#include <cutils/properties.h>
#include <gui/Surface.h>
#include <media/ICrypto.h>
#include <media/MediaCodecBuffer.h>
#include <media/stagefright/foundation/ABuffer.h>
#include <media/stagefright/foundation/ADebug.h>
#include <media/stagefright/foundation/AMessage.h>
#include <media/stagefright/MediaBuffer.h>
#include <media/stagefright/MediaCodec.h>
```

图 5-1: Converter 调试宏

5.1.2 源数据和编码后的数据获取

Miracast 发送端通过 MediaCodec 进行音视频编码，因此在 Android12 以上版本，可以在 Codec2 服务中获取编码输入和输出数据，以此来确认投屏的源数据跟编码数据是否正常。具体调试方式，可以参考《Android 14 多媒体开发指南》调试方法章节，这里不做赘述。

5.1.3 MPEG-TS 数据获取

编码后的数据，会经过 ts 打包器进行打包后，再以 RTP 包形式发送出去。在 MediaSender 中，去掉 mLogFile 的注释，可以保存 ts 码流数据，如下所示。

```

MediaSender::MediaSender(
        const sp<ANetworkSession> &netSession,
        const sp<AMessage> &notify)
    : mNetSession(netSession),
      mNotify(notify),
      mMode(MODE_UNDEFINED),
      mGeneration(0),
      mPrevTimeUs(-1ll),
      mInitDoneCount(0),
      mLogFile(NULL) {
      //mLogFile = fopen("/data/record/log.ts", "wb");
}
```

图 5-2: TS 数据保存

通过 MediaSender 的 TS 码流数据，可以方便确认编码-> 打包过程是否存在数据异常。mLogFile 保存下来的是发送到 sink 端的 ts 数据，也就是经过编码压缩后的 ts 流。它跟 sink 端看到画面是一样的。保存下来的 ts 流，可先用电脑端播放器进行播放，如果声音画面出现卡顿，则在数据编码打包环节可能存在异常。如果播放正常，则需要排查网络发送以及接收端情况。

5.2 Miracast 接收端

5.2.1 丢包分析

Miracast 接收端，常见的是播放出现花屏问题，而花屏的原因经常来源于接收端出现了丢包。一般遇到花屏问题，首先我们需要在投屏复现过程中，实时抓取 tcpdump 数据来确认 wifi 驱动的收包情况。

```
tcpdump -i any -w /sdcard/test.dump
```

得到的报文数据，通过 wireshark 工具可以在报文中提取视频流，步骤如下：

1. wireshark 打开报文；
2. 过滤出 UDP 报文，选择 Decoder As, 选择 RTP；
3. 界面将出现 MPEG TS 包，这个时候选择电话-> RTP-> 流分析，此步完成后可以看到 RTP 数据包丢包率，延迟等信息报告
4. 将 RTP 流保存为 “不同步的正向音频”，存为 raw 数据文件，该文件即为 ts 的码流文件；

通过上面的步骤，我们可以直接获取导接收端收到的 RTP 数据，确定当前从底层驱动收到的码流是否有花屏/卡顿/等现象。如果 dump 出来的码流文件播放存在异常，那么需要分析 wifi 驱动以及空中包丢包情况。否则进一步往上分析网络协议栈的情况。

当 tcpdump 抓包的数据没有花屏，但 miracast 应用内出现花屏，可能是网络协议栈出现了丢包。对于 UDP 而言，丢包最大可能就是内核的 UDP 接收缓冲区满了，导致内核丢包，可以通过命令进行查看：

```
cat proc/net/snmp |grep Udp
```

输出的信息如下所示。

```
1|h713-tuna_p3:/ # cat proc/net/snmp |grep Udp
Udp: InDatagrams NoPorts InErrors OutDatagrams RcvbufErrors SndbufErrors InCsumErrors IgnoredMulti
Udp: 204909 9 28 390 28 0 0 74
UdpLite: InDatagrams NoPorts InErrors OutDatagrams RcvbufErrors SndbufErrors InCsumErrors IgnoredMulti
UdpLite: 0 0 0 0 0 0 0 0
h713-tuna_p3:/ #
h713-tuna_p3:/ #
h713-tuna_p3:/ #
```

图 5-3: 查看 udp 缓冲区数据

可以看到，由于 RevbufErrors 而导致的 UDP 丢包数是 28 个。一般来说这种情况，可以先确认 socket 的设置 recv 缓冲区大小，以及 rmem_max 的值，确认 buffer 缓冲区是否足够。

```
sysctl net.core.rmem_default ///查看接收套接字缓冲区的默认大小

sysctl net.core.rmem_max //查看接收套接字缓冲区的最大大小

sysctl -w net.core.rmem_max=xxxx //设置接收套接字缓冲区的最大大小
```

这种情况，一般是应用从内核缓冲区取包速度小于底层驱动接受包速度，数据量大的时候，容易出现缓冲区满导致花屏。这时主要考虑两个因素，cpu 占用高导致应用内部线程无法及时调度；另一个因素是应用本身收包不及时，比如两个 recvfrom 中间，应用可能存在一些耗时处理，这里需要结合 systrace 进一步分析。

如果 udp 看到 RevbufErrors 数量并没有增加，但 RTP 接收端出现丢包的情况，可以过滤RTSPClient 这个 TAG，确认是否有如下打印。

```
12-15 14:23:30.521 I/cedarc  ( 5862): <RequestVideoStreamBuffer:1341>: stream buffer 0 ring back.
12-15 14:23:30.926 D/RTSPClient( 5862): ReorderingPacketBuffer::getNextCompletedPacket: Detect 1 rtp packets(33006-33006)
lost
12-15 14:23:30.936 D/cedarc  ( 5862): <H264CheckNewFrame:2090>: here3:bDecSecondFieldError=1
12-15 14:23:30.936 E/cedarc  ( 5862): <H264CheckNewFrame:2093>: here1, the first slice of the frame is not 0
12-15 14:23:31.145 I/cedarc  ( 5862): <DebugDecoderSpeedInfo:1031>: hardware speed: avr = 149.00 fps, max = 153.06 fps, mi
n = 140.19 fps
```

图 5-4: RTP 包乱序丢包打印

如果出现上述打印，那么需返回排查 tcpdump 的数据包，查看是否底层驱动收包就存在乱序的情况。如下为 wireshark 中进行 RTP 包分析的数据，这里看到中间存在 wrong sequence number 即包序号错误的情况，如绿色标记的序列为 33007 的包，对应到我们日志中的 RTP packet。这里包序号出错，会导致上层检测 RTP 包序号时由于接收期望序号与实际序号不一致，造成丢包。这种情况需要进一步分析 wifi 驱动的接收情况，一般与驱动收包乱序有关。

Wireshark · RTP 流分析 · test1.dump

Forward

192.168.49.1:23760 → 192.168.49.248:58930

SSRC	0xdeadbeef
Max Delta	93.28 ms @ 19436
Max Jitter	12.19 ms
Mean Jitter	0.22 ms
Max Skew	-128.16 ms
RTP Packets	39471
Expected	39472
Lost	1 (0.00 %)
Seq Errs	14
Start at	19.947644 s @ 126
Duration	124.89 s
Clock Drift	5 ms
Freq Drift	90004 Hz (0.00 %)

Reverse

192.168.49.248:58930 → 192.168.49.1:23760

SSRC	0x00000000
Max Delta	0.00 ms @ 0
Max Jitter	0.00 ms
Mean Jitter	0.00 ms
Max Skew	0.00 ms
RTP Packets	0
Expected	1
Lost	1 (100.00 %)
Seq Errs	0
Start at	0.000000 s @ 0
Duration	0.00 s
Clock Drift	0 ms
Freq Drift	1 Hz (0.00 %)

正向 | 反向 | 图形

分组	序列	Delta (ms)	抖动 (ms)	扭曲	带宽	标记	状态
132	18	0.17	0.15	0.30	75.94		Wrong sequence number
134	6	0.17	0.16	-0.23	97.63		Wrong sequence number
135	7	0.30	0.17	-0.52	108.48		Wrong sequence number
136	8	0.08	0.17	-0.59	119.33		Wrong sequence number
137	9	0.08	0.16	-0.67	130.18		Wrong sequence number
138	10	0.20	0.16	-0.86	141.02		Wrong sequence number
139	11	0.09	0.16	-0.95	151.87		Wrong sequence number
140	12	0.08	0.15	-1.01	162.72		Wrong sequence number
141	13	0.07	0.15	-1.08	173.57		Wrong sequence number
142	14	0.08	0.14	-1.16	184.42		Wrong sequence number
143	15	0.08	0.14	-1.23	195.26		Wrong sequence number
144	16	0.07	0.13	-1.30	206.11		Wrong sequence number
145	17	0.07	0.13	-1.37	216.96		Wrong sequence number
33405	33007	84.72	3.38	-49.80	2603...	•	Wrong sequence number
126	0	0.00	0.00	0.00	10.85		✓
127	1	0.20	0.01	-0.14	21.70		✓
128	2	0.09	0.01	-0.23	32.54		✓
129	3	0.07	0.02	-0.29	43.39		✓
130	4	0.81	0.07	-1.10	54.24		✓
131	5	0.10	0.15	0.39	65.09		✓
133	19	0.29	0.16	0.01	86.78		✓
146	20	1.60	0.22	-2.95	227.81		✓
147	21	0.10	0.21	-3.03	234.14	•	✓
148	22	2.26	0.47	1.37	237.47	•	✓
149	23	25.78	1.08	-8.88	245.31	•	✓
151	24	21.67	1.63	0.88	248.64	•	✓
152	25	0.19	1.53	0.75	251.97	•	✓
153	26	0.20	1.45	0.57	262.82		✓
154	27	0.09	1.36	0.48	264.64	•	✓
159	28	139.43	7.80	-103.82	267.97	•	✓
160	29	0.22	7.32	-103.66	278.82		✓
161	30	0.09	6.87	-103.73	289.66		✓
162	31	0.08	6.44	-103.81	300.51		✓
163	32	0.08	6.04	-103.87	309.86	•	✓
164	33	0.08	7.67	-71.75	317.70	•	✓
165	34	0.08	7.20	-71.65	321.02	•	✓
166	35	0.12	6.76	-71.74	324.35	•	✓

图 5-5: wireshark 查看 UDP 包分析

6 Miracast 相关配置

在 A133 的平板上，为了减少 AAC 的编码负载，降低 cpu 使用率，配置了media.wfd.use-pcm-audio属性默认为 true，即当对端在 M3 阶段返回支持 PCM 裸传情况下，发送端会直接发送 PCM 数据而不经过 AAC 的编码。如果走 PCM 通路，会看到如下打印：

```
01-23 09:48:32.933   822   822 V MediaRouter: Selecting route: RouteInfo{ name=小湃快投-221_mirror, description=Wireless display, status=Connecting...,
01-23 09:48:33.038  4490  4490 W android_priv_cmd: Android private cmd "MIRACAST 1" on wlan0
01-23 09:48:33.046  4490  4490 W wrong cmd: (null) in handle_private_cmd
01-23 09:48:33.376   529 29937 I NetworkSession: incoming connection from 192.168.49.227:38382 (socket 11)
01-23 09:48:33.376   529 29937 I NetworkSession: added clientSession 2
01-23 09:48:33.376   529 29938 I WifiDisplaySource: We now have a client (2) connected.
01-23 09:48:33.431   529 29938 I WifiDisplaySource: Picked video resolution 1280 x 720 p30
01-23 09:48:33.432   529 29938 I WifiDisplaySource: Picked AVC profile 1, level 1
01-23 09:48:33.432   529 29938 I WifiDisplaySource: wfd_audio_codecs = (LPCM 00000003 00, AAC 0000000F 00)
01-23 09:48:33.432   529 29938 I WifiDisplaySource:  supportsAAC=1, supportsPCM=1
01-23 09:48:33.432  1187  1187 V MediaRouter: Selecting route: RouteInfo{ name=小湃快投-221_mirror, description=Wireless display, status=Connecting...,
01-23 09:48:33.432   529 29938 I WifiDisplaySource: Using PCM audio.
01-23 09:48:33.432   529 29938 I WifiDisplaySource: Sink does not support content protection.
01-23 09:48:33.493   529 29938 D WifiDisplaySource: config audio buf before miracast setup![10]
01-23 09:48:33.498   529 29938 D MediaCodecList: codecHandlesFormat: no format, so no extra checks
01-23 09:48:33.498   529 29938 D MediaCodecList: codecHandlesFormat: no format, so no extra checks
01-23 09:48:33.503   529 29941 D CCodec  : allocate(c2.allwinner.avc.encoder)
01-23 09:48:33.507   529 29941 I CCodec  : setting up 'default' as default (vendor) store
01-23 09:48:33.508   402   884 D C2Store : dlopen libcodec2_hw_avcenc.so.
01-23 09:48:33.514   402   884 I android.hardware.media.aw.c2@1.0-service: missing struct descriptor #Param::CoreIndex(--004) for field values of struc
```

图 6-1: PCM 裸传日志

这种情况下，如果出现对端没有声音，可以考虑设置media.wfd.use-pcm-audio属性为 false，再进行投屏。如果此时声音播放正常，则证明接收端没有支持 PCM 裸传通路但又返回了 PCM 的能力。

7 FAQ

常见的一些投屏问题，目前在全志一号通平台上的均有相关 FAQ 指南，以下列举的几类典型问题可以直接 FAQ 系统上搜索下述标题，并根据 FAQ 文档进行问题排查。

1. Miracast 投射屏幕卡顿问题分析：FAQ1492
2. Miracast 投屏 source 端投射分辨率异常：FAQ1700
3. Miracast 投屏出现黑屏问题分析：FAQ1801
4. 如何对 Miracast 码率进行设置：FAQ1987
5. Mircast Sink 端花屏绿屏分析：FAQ2168

著作权声明

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





（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。