基于Wi-Fi Direct的音频传输系统

程嘉淦，王鹤翔，江月虎，宋昱龙

摘要

我们实现了一个软件系统：它可以通过Wi-Fi Direct建立一个网络，自主发现网络中的某一设备并向该设备发送音频信号。该设备收到音频信号后，如果有多个来源，将对这些音频信号混频。最终这个设备讲接收的音频信号播放。

应用场景

本项目实现之初旨在共享局域网中的音频设备（如蓝牙耳机或大型扬声器）。除此之外，本项目实现的协议能够用于搭建基于 WiFi Direct 网络的高效的实时音频传输网络，能够用于实现音频会议等场景。

背景技术

1. Wi-Fi Direct技术

相较于蓝牙技术，Wi-Fi技术有带宽更大、传输距离更远、抗干扰能力更强等优点，而且还能够使用传统的TCP/IP网络协议栈。但是传统Wi-Fi网络的建立需要有一台设备充当接入点（AP），不能方便的构建网络。而Wi-Fi Direct技术则解决了这个问题，它不需要指定的设备充当AP，多台设备可以以Ad-hoc自组网的形式组织网络。为了方便快捷的传输音频信号，因此我们选用Wi-Fi Direct技术组成网络。

1. 服务发现协议、心跳协议和音频控制协议

在该部分，我们称音频信号的发送者为Client端，音频信号的接受者为Server端。

在我们设想的情景中，Server端可能是网络中的任意一台设备，因此Server端的地址是不确定的。如果让用户自行寻找Server端的地址，这显然是一件不合理的事情，因此我们需要设计一个服务发现协议，让Client端可以自动的发现网络中的Server端。同时，为了确保通信的正确进行，我们需要Server端和Client端每隔一段时间都去确认双方是否都还在当前网路中。为此我们设计了心跳协议。

为了方便Server端接收并播放音频流，Client端在开始发送音频前和停止发送音频后，都应该向Server端告知自身当前的播放状态。因此我们设计了音频控制协议。

服务发现协议和心跳协议如图1所示。其过程大致如下：

（1）Client端：向网络广播消息“client”，表示有新的Client加入网络中。

（2）Server端：在接受到广播消息后，向广播源（即对应的Client）发送消息“server”，表明自己的身份。

心跳协议由Client端定时执行。其过程大致如下：

（1）Client端：向Server发送心跳消息“server?”并等待回复。如果不能在较短时间内接收到Server的回复，则认为该Server已经离开网络。

（2）Server端：接收到Client的心跳消息后回复消息“serverYes”。如果在规定时间内没有接收到Client的心跳消息，则认为该Client已经离开网络。

图片包含 就坐

描述已自动生成

图1：服务发现协议和心跳协议

音频控制协议如图2所示。音频控制协议包含两种子协议：请求播放和请求暂停。其过程大致如下：

请求播放：

（1）Client端：向Server发送消息“play”。

（2）Server端：接收到消息后，准备好接收音频流，并向Client发送消息“playOk”。

（3）Client端：接收到消息后，通过RTP协议向Server发送音频信号。

请求暂停：

（1）Client端：向Server发送消息“stop”并停止发送音频信号。

图片包含 就坐

描述已自动生成

图2：音频控制协议

1. RTP协议和RTCP协议
2. RTP概览

我们项目中对音频信号的传输是通过RTP实现的。RTP协议详细说明了在互联网上传输音频和视频的标准数据包格式，常用于流媒体系统中，是一个多播协议但也可以用于单播应用中，非常符合我们项目的需求。

RTP协议是一种基于UDP的无连接的传输协议，和RTCP控制协议一起使用。RTP协议为Internet上端到端的实时传输提供时间信息和流同步，但它并不保证服务质量。服务质量控制功能由RTCP协议来提供。

图3展示了一个典型流媒体应用的体系结构。RTP协议被划分在传输层，建立在UDP上。因此，同UDP协议一样，为了实现实时传输功能，RTP也有固定的封装形式，接下来将具体介绍RTP数据包的封装结构。

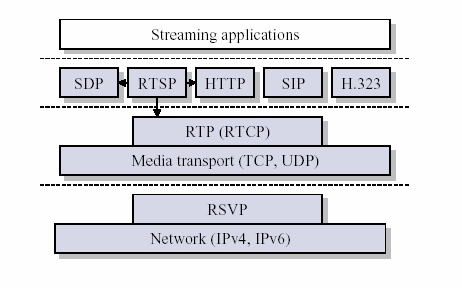


图3：典型流媒体应用的体系结构

1. RTP的封装格式

图4展示了RTP报文的头部格式。其各部分含义如下：

* 版本号（V）：2比特，用来标志使用的RTP版本。
* 填充位（P）：1比特，如果该位置位，则该RTP包的尾部就包含附加的填充字节。
* 扩展位（X）：1比特，如果该位置位的话，RTP固定头部后面就跟有一个扩展头部。
* CSRC计数器（CC）：4比特，含有固定头部后面跟着的CSRC的数目。
* 标记位（M）：1比特,该位的解释由配置文档（Profile）来承担.
* 载荷类型（PT）：7比特，标识了RTP载荷的类型。
* 序列号（SN）：16比特，发送方在每发送完一个RTP包后就将该域的值增加1，接收方可以由该域检测包的丢失及恢复包序列。序列号的初始值是随机的。
* 时间戳（timestamp）：32比特，记录了该包中数据的第一个字节的采样时刻。在一次会话开始时，时间戳初始化成一个初始值。即使在没有信号发送时，时间戳的数值也要随时间而不断地增加。时间戳是去除抖动和实现同步不可缺少的。
* 同步源标识符（SSRC）：32比特，同步源就是指RTP包流的来源。在同一个RTP会话中不能有两个相同的SSRC值。该标识符是随机选取的，用于唯一标识RTP会话中的参与者。
* 贡献源列表（CSRC List）：0～15项，每项32比特，用来标志对一个RTP混合器产生的新包有贡献的所有RTP包的源。由混合器将这些有贡献的SSRC标识符插入表中。SSRC标识符都被列出来，以便接收端能正确指出交谈双方的身份。（混合器用于需要将多个源的音频包进行合并以及接收者能接收的音频编码格式不一致的情况中）

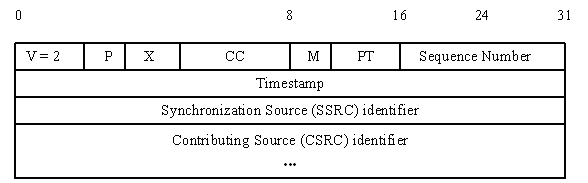


图4：RTP报文的头部格式

1. RTCP的封装结构

RTCP也是基于UDP进行传送的，封装的主要是一些控制信息，用于检测与反馈服务质量、同步媒体、标识多播组中的成员等等。在RTP会话期间，各参与者周期性地传送RTCP包。RTCP包中含有已发送的数据包的数量、丢失的数据包的数量等统计资料，各参与者可以利用这些信息动态地改变传输速率。RTP和RTCP配合使用，它们能以有效的反馈和最小的开销使传输效率最佳化，因而特别适合传送网上的实时数据。

由于RTCP仅包含一些控制信息，因此分组长度很短，可以将很多RTCP分组封装在一个UDP包中。RTCP有如下五种分组类型：

|  |  |  |
| --- | --- | --- |
| 类型 | 缩写表示 | 用途 |
| 200 | SR(Sender Report) | 发送端报告 |
| 201 | RR(Receiver Report) | 接收端报告 |
| 202 | SDES(Source Description Items) | 源点描述 |
| 203 | BYE | 结束传输 |
| 204 | APP | 特定应用 |

上述五种封装类型的内部结构基本相同，下面以SR为例进行介绍：

发送端报告分组SR用于使发送端以多播的方式向所有接收端报告发送情况，主要内容包括：相应RTP流的SSRC值、RTP流中最新产生的RTP分组的时间戳、RTP流包含的分组数、RTP流包含的字节数等。图5展示了RTCP SR头的结构，其中各部分的含义如下：

* 版本（V）：同RTP包头域。
* 填充（P）：同RTP包头域。
* 接收报告计数器（RC）：5比特，该SR包中的接收报告块的数目，可以为零。
* 包类型（PT）：8比特，对应上表的五种分组类型。
* 长度域（Length）：16比特，其中存放的是该SR包以32比特为单位的总长度减一。
* 同步源（SSRC）：SR包发送者的同步源标识符。与对应RTP包中的SSRC一样。
* NTP Timestamp（Network time protocol）SR包发送时的绝对时间值。NTP的作用是同步不同的RTP媒体流。
* RTP Timestamp：与NTP时间戳对应，与RTP数据包中的RTP时间戳具有相同的单位和随机初始值。
* Sender’s packet count：从开始发送包到产生这个SR包这段时间里，发送者发送的RTP数据包的总数. SSRC改变时，这个域清零。
* Sender’s octet count：从开始发送包到产生这个SR包这段时间里，发送者发送的净荷数据的总字节数（不包括头部和填充）。发送者改变其SSRC时，这个域要清零。
* 同步源n的SSRC标识符：该报告块中包含的是从该源接收到的包的统计信息。
* 丢失率（Fraction Lost）：表明从上一个SR或RR包发出以来从同步源n(SSRC\_n)来的RTP数据包的丢失率。
* 累计的包丢失数目：从开始接收到SSRC\_n的包到发送SR,从SSRC\_n传过来的RTP数据包的丢失总数。
* 收到的扩展最大序列号：从SSRC\_n收到的RTP数据包中最大的序列号，
* 接收抖动（Interarrival jitter）：RTP数据包接受时间的统计方差估计
* 上次SR时间戳（Last SR,LSR）：取最近从SSRC\_n收到的SR包中的NTP时间戳的中间32比特。如果目前还没收到SR包，则该域清零。
* 上次SR以来的延时（Delay since last SR,DLSR）：上次从SSRC\_n收到SR包到发送本报告的延时。

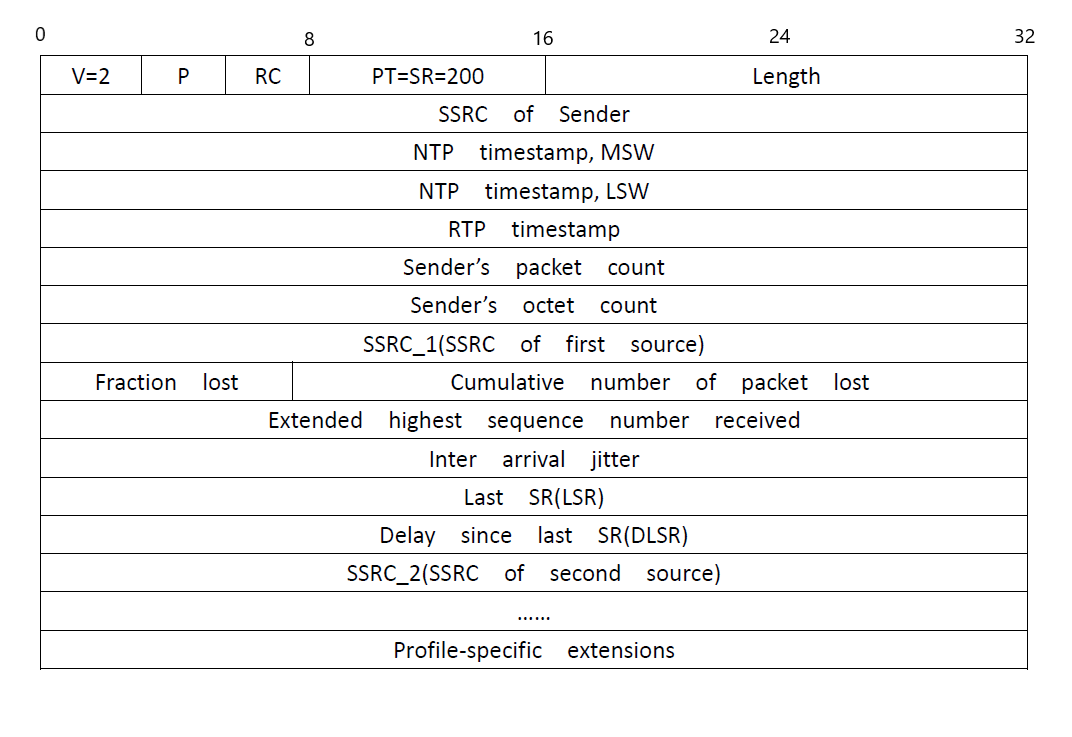


图5：RTCP SR头的结构

1. RTP的会话过程

当应用程序建立一个RTP会话时，需要首先确定一对目的传输地址，包含一个网络IP地址以及一对端口号。这对端口号一个用于RTP包，一个用于RTCP包，其中RTP数据发向偶数的UDP端口，RTCP包发向相邻的奇数UDP端口，这样就构成了一个UDP端口对。图6展示了RTP包发送的过程。接收过程与此相反。

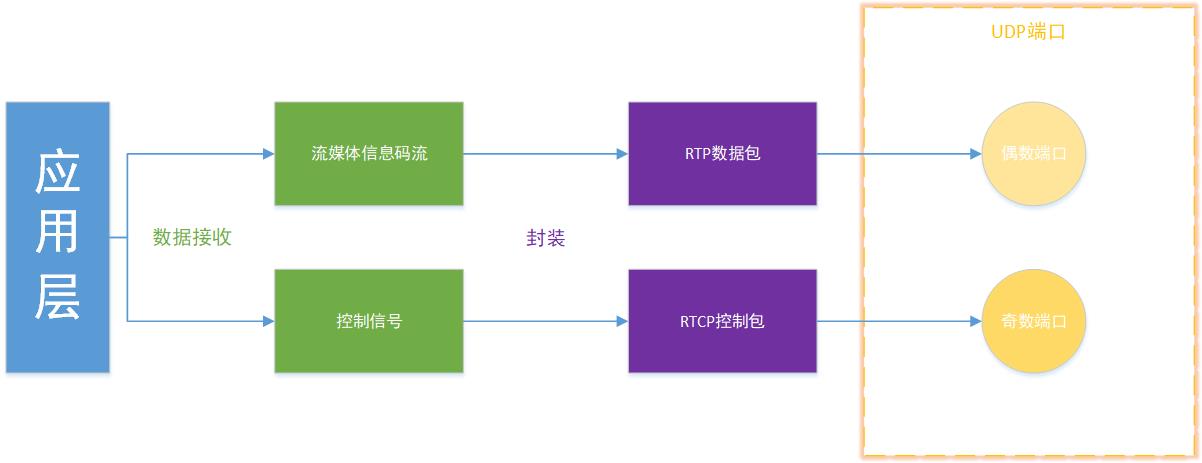


图6：RTP包的发送流程

1. 音频数据的处理

处理数据前，我们会把音频数据解码为PCM格式。发送端读取音频文件获取音频输入流，将音频流切割为指定的块大小，然后通过RTP发送。接收端收到字节流后对其进行处理，通过已打开的相应格式的源数据行将处理过的字节流写入数据行中，通过混频器传递到输出端口。图7展示了这个过程。



图7：音频数据的传输过程

对于每个发送者，动态添加新的与之对应的接收buffer和混合buffer，所有buffer大小一定。每次接收到数据时将数据存入到该数据发送者对应的接收buffer中（I），当接收buffer满，数据转移到混合buffer中。如果混合buffer为空，直接转移（II），若混合buffer满，考虑到两个及以上个发送者时，进一步检测是否所有的buffer均满（III）。若所有的混合buffer均满，将当前所有发送者的混合buffer进行混音，发送到输出端口进行播放，并清空当前所有发送者的混合buffer（IV）；若有混合buffer未满，则丢弃当前接收者混合buffer中的数据，直接将接收buffer数据转移（V）。图8和图9分别展示了单发送端和多发送端的这一处理过程。

同时，在每一轮接收数据过程中，检测每一个发送者多少轮没发送过数据。若一定轮次里均没有发送过数据，则动态删除该发送者的接收和混合buffer。

图片包含 物体

描述已自动生成

图8：单发送端处理过程

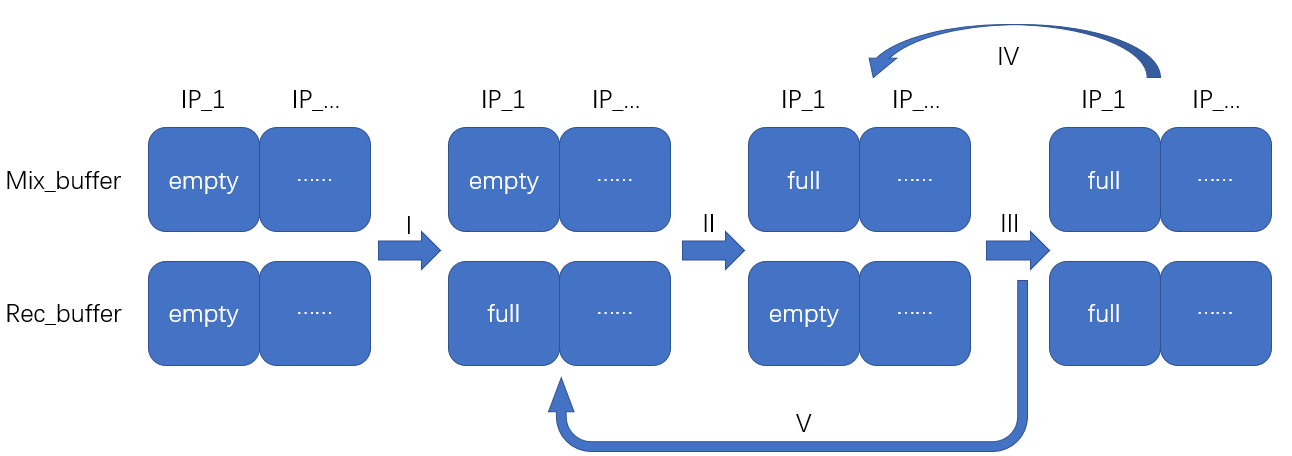


图8：多发送端处理过程

我们采用的是带符号位的PCM编码方式，每一字节数据在-128~127之间。混音时，取两个数据包的每一字节，进行以下运算。这样两个声音就可以混合起来。



技术细节

在我们的实现中，心跳协议的重复时间间隔为30秒，Client端等待时间为1秒。考虑到网络传输的延时，Server端等待时间为30.5秒。

为了实现RTP包的封装，我们使用jlibrtp库。图9展示了jlibrtp库的使用过程：

1. 首先，调用jlibrtp库的实时传输会话类RTPSession建立收发端的会话。该类可以创建一个RTP会话，并设置传输的RTP端口和RTCP端口，以及与RTP包的相关的时间戳数据等。
2. 然后，通过RTPSessionRegister方法用于添加RTP会话的参与者，同时开启接收包的AppCallerThread线程类。线程类的run方法在接受到RTP数据包时，会去掉RTP包头，获取传输数据内容，并调用回调函数receiveData。在该函数中，我们将RTP负载存入buffer，然后进行音频数据处理和混频。

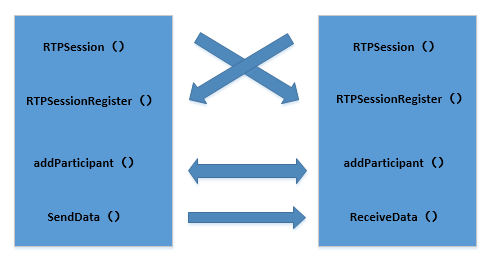


图9：jlibrtp库的使用过程

在实现音频数据处理和混频过程中，我们选择每个数据块的大小为1024字节，buffer的大小为30个数据块。发送者30轮未发送数据后将会被删除。

buffer的大小会影响丢包率。如果buffer过小，会导致丢包率升高。buffer过大，则会浪费大量的内存空间。

实现内容

1. 安卓端

安卓端 Server 和 Client 由同一软件完成，如图10所示。在初始界面运行服务发现协议，由一台终端发起连接 (Create New Network) 与其它终端互联 (Search)。在网络中终端互相成功发现后，每个终端可以通过按钮切换在音频传输中担任的角色 (Server/Client)。通常每一组网络中有且仅有一个终端为音频 Server，用来接收网络中 Client 的音频数据并混音播放。

图片包含 天空

描述已自动生成

图10：安卓终端界面

1. PC端

PC端在进入软件开始便选择自身音频传输中的身份 (Server/Client)。Server 端可以直接启动并持续监听网络中是否有 Client 传送数据。Client 端则需要搜索网络中存在的 Server，主动与其连接并选择音频文件进行发送。

图片包含 屏幕截图

描述已自动生成

图11：PC终端界面