AIRPLAY 协议

一、介绍

AIRPLAY 是由苹果公司实现的一套协议族，用来实现在 **Apple TV** 上浏览 iPhone、iPod touch、iPad（硬件设备）或者 iTunes（软件）中的各种媒体内容。 AirPlay 支持如下几种使用场景:

- 从 iOS 设备上传输并显示照片、幻灯片；
- 从 iOS 设备或者 Itunes 软件中传输并播放音频；
- 从 iOS 设备或者 Itunes 软件中传输并播放视频；
- 对 iOS 设备或者 OS X Mountain Lion 进行**屏幕镜像**。由于此功能需要硬件的硬解码支持，所以只能在 iPad 2、iPhone 4S、带 Sandy Bridge CPU 的 Mac 电脑（或更新的设备）上支持。

最初这套协议名字叫 **AirTunes**，只支持音频流播放。 后来苹果开发 Apple TV 时，对此协议进行了扩充和改进，加入了视频支持，并改名叫做 AIRPLAY。AIRPLAY 协议基于一些知名的网络标准协议，如 **Multicast DNS**、**HTTP**、**RTSP**、**RTP** 或 **NTP** 以及其他的一些自定义扩展。

由于我们只关注音频部分，所以下面研究的重点是 **AirTunes** 服务。

二、实现机制

实现 AIRPLAY 协议的软件不需要再做任何配置就能发现同一网络中的相关设备，这主要得益于 **Bonjour**（基于 **M-DNS** 协议实现）

Bonjour：苹果为基于组播域名服务(multicast DNS)的开放性 **Zeroconf 标准**所起的名字。

Zeroconf (零设置网络标准)：全称为 Zero configuration networking，中文名则为零配置网络服务标准，是一种用于自动生成可用 IP 地址的网络技术，不需要额外的手动配置和专属的配置服务器。“零配置网络”的目标，是让非专业用户也能便捷的连接各种网络设备，例如计算机，打印机等。整个搭建网络的过程都是通过程式自动化实现。如果没有 zeroconf，用户必须手动配置一些服务，例如 DHCP、DNS，计算机网络的其他设置等。这些对非技术用户和新用户们来说是很难的事情。

具体例子为：用户拥有一台 apple tv 和一台 iPhone4s，那之只要都连入到同一个无线局域网内，iphone4s 就会自动找出 apple tv，那么在播放音乐或者视频时候，用户只要点击推送，就可以讲音乐和视频推送到 apple tv 上播放。

除了 **Bonjour** 以外，实现 **Zeroconf 协议**的还有 Avahi 和 howl。

下面以 **AirTunes** 服务为例来具体看是如果实现服务发现的。首先发布 **RAOP**（Remote Audio Access Protocol ）服务，其格式如下：

RAOP服务

```
name: 5855CA1AE288@Apple TV
type: _raop._tcp
port: 49152
txt:
 txtvers=1
 ch=2
 cn=0,1,2,3
 da=true
 et=0,3,5
 md=0,1,2
 pw=false
 sv=false
 sr=44100
 ss=16
 tp=UDP
 vn=65537
 vs=130.14
 am=AppleTV2,1
 sf=0x4
```

name 字段由设备的 MAC 地址和远程设备的名称组成（通常就在客户端上显示此设备名称）

TXT 参数中包含以下字段：

键值名	值	描述
txtvers	1	TXT record version 1
ch	2	audio channels: stereo
cn	0,1,2,3	audio codecs
et	0,3,5	supported encryption types
md	0,1,2	supported metadata types
pw	false	does the speaker require a password?
sr	44100	audio sample rate: 44100 Hz
ss	16	audio sample size: 16-bit
tp	UDP	supported transport: TCP or UDP
vs	130.14	server version 130.14
am	AppleTV2,1	device model

Audio codecs（音频编码）

CN	描述
0	PCM
1	Apple Lossless (ALAC)
2	AAC
3	AAC ELD (Enhanced Low Delay)

Encryption Types

ET	描述
0	no encryption
1	RSA (AirPort Express)
3	FairPlay
4	MFiSAP (3rd-party devices)
5	FairPlay SAPv2.5

Metadata Types

MD	描述
0	text
1	artwork
2	progress

RAOP 从本质上来说是实时流协议（RTSP，其内容为实时流传输协议和控制协议），只不过增加了基于身份验证请求-应答的一步。实时流协议是应用层协议，用来实现和控制实时数据的传送。

RAOP 服务用两个信道实现流媒体音乐：一个是用实时流协议的控制信道；另一个是数据信道用来发送原始数据。以 iTunes 客户端（v6.0.4）和 Airport Express 路由器（简称 ApEx）之间的数据交换为例分析 **RAOP** 服务过程如下所示：

从 iTunes 到 ApEx 传输

```
OPTIONS * RTSP/1.0
CSeq: 1
User-Agent: iTunes/6.0.4 (Macintosh; N; PPC)
Client-Instance: 99BB1C4A4056F46D
DACP-ID: 99BB1C4A4056F46D
Active-Remote: 4294936225
Apple-Challenge: X/GmLMLuFvgWf8Y1bQuUug
```

从 ApEx 到 iTunes 传输

```
RTSP/1.0 200 OK
CSeq: 1
Public: ANNOUNCE, SETUP, RECORD, PAUSE, FLUSH, TEARDOWN, OPTIONS, GET_PARAMETER, SET_PARAMETER
Apple-Response:
fUG5XtwIbJDWcpYX7p81z7bYfWD7UKa9VkIQk40szRYTOkP8VJ+3Ol7YRdvwR2hMxUtjoDDIjqFdDiSu50
SfxfEtnquj7nFyR8gqJKnXNnpgegBaaFatoCLHTaH7Nc5H4yH/MQ2qrHtJ/5i+R7ElCd29xaC31r/wfDYg
xMy2YmoeaRnvudSUOURHsOO8mEbJYfNp1rC2+W7EGwYyN/QQ98/kREUPO1et2qz7THEUsOn22ql/2VA88E
gpyDsQMNIuUdOXdjrH1Moqz+yGOKmKJyP8WoehJPhfW1da4YJSW2qAahQZRgJ7x7M3KUGEhzut8pf6CP/U
1FRJqj7KFMTwTg
Audio-Jack-Status: connected; type=analog
```

从上面可以看出 iTunes 客户端提供自己的版本号和一个随机生成的 22byte 的加密的苹果请求参数给 ApEx。然后 ApEx 回复一个响应，这个响应是由储存在 ApEx 的私钥加密后的请求参数。然后 iTunes 用非对称密钥对的公钥对该值进行验证（这种私钥加密公钥验证的方法具体实现细节如果感兴趣可以参考

http://zh.wikipedia.org/wiki/RSA%E5%8A%A0%E5%AF%86%E6%BC%94%E7%AE%97%E6%B3%95)。这一步目的是 iTunes 用来验证是否正在与一个 ApEx 对话。在这一步交流过后上述连接断开。

接下来，iTunes 在同一个端口建立另外一个与 ApEx 相连的 RTSP 连接，同时提供一个随机产生的 AES 密钥给 ApEx。这个 AES 密钥是经过 RSA 加密过的，其密钥由 iTunes 提供（即非对称密钥对的公钥）。然后通过 ApEx 的私钥解密来验证是否正在跟一个 iTunes 对话。值得欣喜的是目前通过逆向工程已经破解了非对称密钥对。上述过程如下所示：

从 iTunes 到 ApEx 传输

```
ANNOUNCE rtsp://10.0.1.2/3233609434 RTSP/1.0
CSeq: 1
Content-Type: application/sdp
Content-Length: 563
User-Agent: iTunes/6.0.4 (Macintosh; N; PPC)
Client-Instance: 99BB1C4A4056F46D
DACP-ID: 99BB1C4A4056F46D
Active-Remote: 4294936225

v=0
o=iTunes 3233609434 0 IN IP4 10.0.1.2
s=iTunes
c=IN IP4 10.0.1.1
t=0 0
m=audio 0 RTP/AVP 96
a=rtpmap:96 AppleLossless
a=fmtp:96 4096 0 16 40 10 14 2 255 0 0 44100
a=rsaaeskey:HSyPEnWds0b2Qoc1733RyWmInqHXn61V8UarTBW+cwPrSV4DqP8kChGxGnJ9QJAyQQvTcuVhL
J2MCGP2ddANQWeguvxJfyIZuM9bwX4ZA3FgWWF6QOTyDVy7ppK587Mh1Y6+GYujTdMZ6ukbC3thXmC5PyipVI
EOR3By9AJGpVTWR8LpG5dcuwkXbzlrmqr4IT7bsffpAm/5wzqk0lcrNiI/QcYqC0jZ744mNAkQIQqijVR/IoO
F6o4KpvwUIXIlhPJm87m4ghTLuXEqDhtdcmKza/uRm0l0KwcHkS/ON4WgvgiuHzlMML8pVDBKeAY1R6x2sGxs
GWTWOE3FsMFM/w
a=aesiv:EBqQ4XNBST+PpC28SX1oXA
```

从 ApEx 到 iTunes 传输

```
RTSP/1.0 200 OK
CSeq: 1
Audio-Jack-Status: connected; type=analog
```

然后，ApEx 告诉 iTunes 哪一个端口用于数据连接(server_port,6000)。

从 iTunes 到 ApEx 传输

```
SETUP rtsp://10.0.1.2/3233609434 RTSP/1.0
CSeq: 2
Transport: RTP/AVP/TCP;unicast;interleaved=0-1;mode=record;control_port=0;timing_port=0
User-Agent: iTunes/6.0.4 (Macintosh; N; PPC)
Client-Instance: 99BB1C4A4056F46D
DACP-ID: 99BB1C4A4056F46D
Active-Remote: 4294936225
```

从 ApEx 到 iTunes 传输

```
RTSP/1.0 200 OK
CSeq: 2
Session: 8090DBF0
Transport: RTP/AVP/TCP;unicast;interleaved=0-1;mode=record;control_port=0;timing_port=0;server_port=6000
Audio-Jack-Status: connected; type=analog
```

在控制包里的 RTSP 序列和时间戳的交换如下所示：
从 iTunes 到 ApEx 传输

```
RECORD rtsp://10.0.1.2/3233609434 RTSP/1.0
CSeq: 3
Session: 8090DBF0
Range: npt=0- RTP-Info: seq=49770;rtptime=1068774379
User-Agent: iTunes/6.0.4 (Macintosh; N; PPC)
Client-Instance: 99BB1C4A4056F46D
DACP-ID: 99BB1C4A4056F46D
Active-Remote: 4294936225
```

从 ApEx 到 iTunes 传输

```
RTSP/1.0 200 OK
CSeq: 3
Audio-Jack-Status: connected; type=analog
```

下面展示如果调整音量参数：
从 iTunes 到 ApEx 传输

```
SET_PARAMETER rtsp://10.0.1.2/3233609434 RTSP/1.0
CSeq: 4
Session: 8090DBF0
Content-Type: text/parameters
Content-Length: 20
User-Agent: iTunes/6.0.4 (Macintosh; N; PPC)
Client-Instance: 99BB1C4A4056F46D
DACP-ID: 99BB1C4A4056F46D
Active-Remote: 4294936225
volume: -15.000711
```

从 ApEx 到 iTunes 传输

```
RTSP/1.0 200 OK
CSeq: 4
Audio-Jack-Status: connected; type=analog
```

最后展示是如何关闭会话的：
从 iTunes 到 ApEx 传输

```
TEARDOWN rtsp://10.0.1.2/3233609434 RTSP/1.0
CSeq: 6
Session: 8090DBF0
User-Agent: iTunes/6.0.4 (Macintosh; N; PPC)
Client-Instance: 99BB1C4A4056F46D
DACP-ID: 99BB1C4A4056F46D
Active-Remote: 4294936225
```

从 ApEx 到 iTunes 传输

```
RTSP/1.0 200 OK
CSeq: 6
Connection: close
Audio-Jack-Status: connected; type=analog
```

重新来看下图：

```
RTSP/1.0 200 OK
CSeq: 1
Public: ANNOUNCE, SETUP, RECORD, PAUSE, FLUSH, TEARDOWN, OPTIONS, GET_PARAMETER, SET_PARAMETER
```

知 ApEx 支持的方法有：**ANNOUNCE, SETUP, RECORD, PAUSE, FLUSH, TEARDOWN, OPTIONS, GET_PARAMETER** 和 **SET_PARAMETER**。

ANNOUNCE：ANNOUNCE 会告诉 RTSP 服务器音频流使用的是会话描述协议。并且也会告知相关的编码信息和加密密钥信息。协议交互如下：**ANNOUNCE** for **Apple Lossless** audio from iTunes

CLIENT → SERVER

```
ANNOUNCE rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 3
Content-Type: application/sdp
Content-Length: 348
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3) AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575

v=0
o=iTunes 3413821438 0 IN IP4 fe80::217:f2ff:fe0f:e0f6
s=iTunes
c=IN IP4 fe80::5a55:caff:fe1a:e187
t=0 0
m=audio 0 RTP/AVP 96
a=rtpmap:96 AppleLossless
a=fmtp:96 352 0 16 40 10 14 2 255 0 0 44100
a=fpaeskey:RlBMWQECAQAAAAA8AAAAAPFOnNe+zWb5/n4L5KZkE2AAAAAQlDx69reTdwHF9LaNmhiRURT
a=aesiv:5b+YZi9Ikb845BmNhaVo+Q
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
Server: AirTunes/130.14
CSeq: 3
```

SETUP：SETUP 会初始化一个记录会话，发送必要的传输信息建立三个 UDP 信道：

CHANNEL	DESCRIPTION
server	audio data
control	sync and retransmit requests
timing	master clock sync

协议交互如下：setup a record session

CLIENT → SERVER

```
SETUP rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 4
Transport: RTP/AVP/UDP;unicast;interleaved=0-1;mode=record;control_port=6001;timin
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3) AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
Transport: RTP/AVP/UDP;unicast;mode=record;server_port=53561;control_port=63379;ti
Session: 1
Audio-Jack-Status: connected
Server: AirTunes/130.14
CSeq: 4
```

RECORD：RECORD 启动音频流，RTP-Info 头文件里包含了下面的参数：

NAME	SIZE	DESCRIPTION
`seq`	16-bit	initial RTP sequence number
`rtptime`	32-bit	initial RTP timestamp

启动音频流的协议交互如下：

CLIENT → SERVER

```
RECORD rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 5
Session: 1
Range: npt=0-
RTP-Info: seq=20857;rtptime=1146549156
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3) AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
Audio-Latency: 2205
Server: AirTunes/130.14
CSeq: 5
```

FLUSH：FLUSH 关闭音频流。其协议交互如下：

CLIENT → SERVER

```
FLUSH rtsp://fe80::217:f2ff:fe0f:e0f6/3413821438 RTSP/1.0
CSeq: 31
Session: 1
RTP-Info: seq=25009;rtptime=1148010660
User-Agent: iTunes/10.6 (Macintosh; Intel Mac OS X 10.7.3) AppleWebKit/535.18.5
Client-Instance: 56B29BB6CB904862
DACP-ID: 56B29BB6CB904862
Active-Remote: 1986535575
```

SERVER → CLIENT

```
RTSP/1.0 200 OK
RTP-Info: rtptime=1147914212
Server: AirTunes/130.14
CSeq: 31
```

TEARDOWN：TEARDOWN 结束 RTSP 会话，其协议交互上面已经有图介绍过。

如果想有更深入的了解可以参考 http://nto.github.io/AirPlay.html#audio-rtsprequests 中关于音频的部分。

三、 AIRPLAY 协议的实现、移植与调试：

在 linux 下实现 AIRPLAY 协议的开源软件有 shairport、xmbc（其音频部分实现机制也是基于 shairport 的）。

shairport 是一个模拟 ApEx 路由器的软件，用以达到传输来自 iTunes 和其他兼容设备的音乐的目的。它是作为支持 **RAOP** 的一个服务端。其 0.X 版本是 perl 脚本写的，但目前 shairport 已经更新至 1.0-dev，该版本已经去除 perl 脚本部分，改为完全由 C 写成，这更方便了我们进行移植工作。但是目前其还在开发中，还没正式发布，潜在着 bug。1.0-dev 版本后必需的依赖库是 OpenSSL，可选的依赖库有 libao、 PulseAudio、 avahi（或者 howl）。通过分析知，如果 libao 和 PulseAudio 安装后 shairport 将有 5 个输出后端（即 alsa、ao、pusle、dummy、pipe）可以选择，而即便不安装 libao 和 PulseAudio，shairport 将仍有 3 个输出后端（即 alsa、dummy、pipe），不影响 shairport 的功能（播放时会产生延迟），如果追求高质量的音质最好还是要安装 PulseAudio。通过不断的尝试，可以选择交叉编译 OpenSSL 和 howl，以避免复杂的交叉编译。

总共需要进行交叉编译的有 alsa-lib、 OpenSSL、howl、zlib 和 shairport1.0-dev

交叉编译前依然是先新建安装目录 shairport，并约定目录（在 shairport 目录下进行）：

WORK_DIR=$PWD

TARGET_INC="$WORK_DIR"/include

TARGET_LIBS="$WORK_DIR"/lib

TARGET_BIN="$WORK_DIR"/bin

TARGET_SBIN="$WORK_DIR"/sbin

交叉编译 alsa-lib：

1../configure --prefix="$WORK_DIR" --bindir="$TARGET_BIN" --sbindir="$TARGET_BIN" --libexecdir="$TARGET_BIN" --libdir="$TARGET_LIBS" --includedir="$TARGET_INC" --enable-shared --disable-static --host=mips-linux --build=i686-linux --disable-alisp --disable-python --disable-old-symbols --disable-seq --disable-rawmidi

2.make 时出现以下错误：

parser.c: In function 'uc_mgr_scan_master_configs':

parser.c:1138: error: 'versionsort' undeclared (first use in this function)

parser.c:1138: error: (Each undeclared identifier is reported only once
parser.c:1138: error: for each function it appears in.)
make[2]: *** [parser.lo] Error 1
解决方法：patch -p1 < alsa-lib-1.0.24.1-uclibc-missing-versionsort.patch
3.下面又出现问题：
aserver.o: In function `pcm_shm_cmd':
aserver.c:(.text+0x25f0): warning: Warning: snd_pcm_hwsync() is deprecated, consider to use snd_pcm_avail()
../src/.libs/libasound.so: undefined reference to `atomic_sub'
../src/.libs/libasound.so: undefined reference to `atomic_add'
collect2: ld returned 1 exit status
make[1]: *** [aserver] Error 1
make[1]: Leaving directory `/home/wcy/shairport/alsa-lib-1.0.24.1/aserver'
make: *** [all-recursive] Error 1
解决方法：在 alsa-lib-1.0.24\include 中的 iatomic.h 中定义的有关 mips 架构的 atomic_add 和 atomic_sub 函数的实现时，将 extern 改为 static。

交叉编译 openssl
1../config shared no-asm --prefix=/home/wcy/shairport/
--openssldir=/home/wcy/shairport/openssl-1.0.1e

2. 修改 Makefile:
CC=mips-linux-gcc
RANLIB=/home/wcy/Lsdk/build/gcc-4.3.3/build_mips/staging_dir/usr/bin
NM=mips-linux-nm AR=mips-linux-ar

交叉编译 howl

1../configure --prefix="$WORK_DIR" --bindir="$TARGET_BIN" --sbindir="$TARGET_BIN" --libexecdir="$TARGET_BIN" --libdir="$TARGET_LIBS" --includedir="$TARGET_INC" --host=mips-linux --build=i686-linux
2. make 时出现错误

```
make[3]: Entering directory `/usr/src/gnome2A/howl-0.9.8/src/autoipd'
/bin/sh ../../libtool --mode=link gcc  -DHOWL_KERNEL -static  -o autoipd
autoip.o linux_autoip.lo posix_main.lo ../../src/lib/howl/libhowl.la -lpthread
gcc -DHOWL_KERNEL -o autoipd autoip.o linux_autoip.o posix_main.o
../../src/lib/howl/.libs/libhowl.so -lpthread  -Wl,--rpath
-Wl,/usr/src/gnome2A/howl-0.9.8/src/lib/howl/.libs -Wl,--rpath -Wl,/opt/gnome29/lib
gcc: linux_autoip.o: No such file or directory
gcc: posix_main.o: No such file or directory
make[3]: *** [autoipd] Erreur 1
make[3]: Leaving directory `/usr/src/gnome2A/howl-0.9.8/src/autoipd'
make[2]: *** [all-recursive] Erreur 1
make[2]: Leaving directory `/usr/src/gnome2A/howl-0.9.8/src/autoipd'
make[1]: *** [all-recursive] Erreur 1
make[1]: Leaving directory `/usr/src/gnome2A/howl-0.9.8/src'
make: *** [all-recursive] Erreur 1
```

解决方法如下：

```
I just updated gnome-2.10.modules to use 0.9.10, and it looks like we have a
different problem with --disable-static.

The explicit "AM_LDFLAGS = -static" bits are gone, but things still fail to
build in the autoipd directory.

On Linux, the configure script defines the following (rewrapped for clarrity):
  AUTOIPD_EXTRA_OBJECTS='$(top_builddir)/src/autoipd/linux_autoip.lo
                         $(top_builddir)/src/autoipd/posix_main.lo'

Now in the src/autoipd/ Makefile.am, we have the following:
  LDADD = $(AUTOIPD_EXTRA_OBJECTS) $(top_builddir)/src/lib/howl/libhowl.la \
          $(PLATFORM_LIBS)
  ...
  autoipd_DEPENDENCIES = linux_autoip.o posix_main.o

So linux_autoip.o and posix_main.o get built before trying to link autoipd.
However, when linking it looks for linux_autoip.lo and posix_main.lo.  However,
the .lo files aren't listed as dependencies, so don't get built.  If I change to
the directory and run "make linux_autoip.lo posix_main.lo", things build okay.

I believe the correct fix is to get AUTOIPD_EXTRA_OBJECTS to list the .o files
instead of .lo files.
```

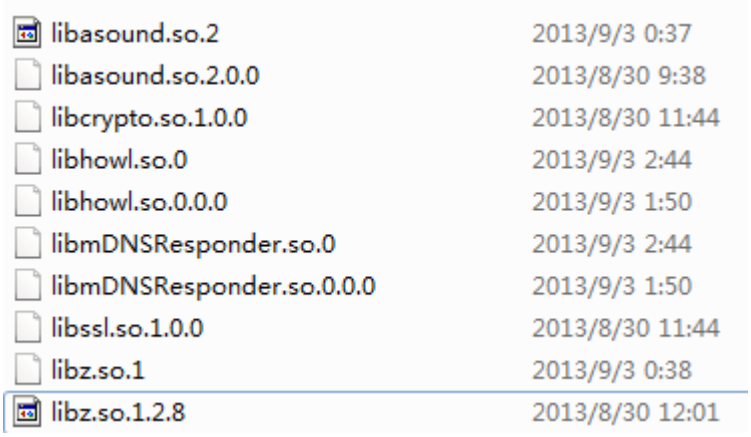
即：进入 src/autoipd　执行命令 make linux_autoip.lo posix_main.lo

交叉编译 zlib

./configure –prefix=../../ Lsdk/build/gcc-4.3.3/build_mips/staging_dir/usr
修改 Makefile:
CC=../../Lsdk/build/gcc-4.3.3/build_mips/staging_dir/usr/bin/mips-linux-gcc
CPP=mips-linux-gcc
AR=mips-linux-ar
RANLIB=mips-linux-ranlib
prefix =../../Lsdk/build/gcc-4.3.3/build_mips/staging_dir/usr

交叉编译 shairport
./configure 之后修改 Makefile:
添加 CC=../../Lsdk/build/gcc-4.3.3/build_mips/staging_dir/usr/bin/mips-linux-gcc
修改 config.mk:
删去 CONFIG_PULSE=yes
修改 CFLAGS 和 LDFLAGS：
CFLAGS+= -D_REENTRANT -I/../../Lsdk/build/gcc-4.3.3/build_mips/staging_dir/usr/include/alsa
LDFLAGS+= -L/../../Lsdk/build/gcc-4.3.3/build_mips/staging_dir/usr/lib -lm -lpthread -lssl -ldl -L/../lib -lz -lasound -lcrypto
修改 config.h:
删去#define CONFIG_PULSE

移植到板子上需要三个可执行文件，分别是 mDNSResponder、mDNSPublish 和 shairport（三者的大小加起来 175kb），且总共需要移植的动态库有 10 个（大小总共加起来有 6.59mb）如下图所示：



动态库放在/lib，可执行程序放在/bin 中即可。调试过程中如果出现“can not load ****.so.*，可通过创建 Symbolic Link 来解决。

调试的结果是程序正常运行，协议交互正常，音频播放功能暂时还没办法测试有待完善（需求带音频部分的板子和外接音箱）

测试环境为 win 7 iTunes 和芯片 9341tplink 的路由器。测试原理:由于路由器和 win7 在同一个网络中，所以 win7 下当 iTunes 软件开启时将能发现路由器中的扬声器设备（虚拟）即图中的蓝色矩形框中会显示相应的小喇叭，其名字对应于开启服务时的输入，这里为“gdpisen”

图中 78DC5128A107 为路由器的 MAC 地址，_raop._tcp 为协议类型，5002 为端口号。