H盒子系列有线网络模块

使用说明书

3.0

2019.09.16

文档履历

版本号	日期	制/修订人	内容描述
1.0	2017.04.05		初始版本
2.0	2019.01.28		for AndroidP
3.0	2019.09.16		for AndroidQ

目录

1. 前言

1.1 编写目的

本文档目的是让客户了解 H 系列 AndroidQ 有线网络框架，能够根据提供的 API 接口函数进行二次开发；同时也为网络开发以及维护人员提供参考。

1.2 适用范围

本模块说明适用于 H 系列芯片 AndroidQ 平台。

1.3 相关人员

网络系统开关人员以及关注该功能的相关人员。

1.4 相关术语

Ethernet：以太网，一种局域网采用的通信协议标准;

PPP：Point to Point Protocol，点对点协议；

PPPoE：PPP over Ethernet，一种在以太网中传输 PPP 帧的技术；

DHCP：Dynamic Host Configuration Protocol，动态主机配置协议;

2. 有线网络系统框架概述

2.1 PPPoE 系统框图

有线网络上网方式包括 DHCP、静态 IP 以及 PPPoE；其中 PPPoE 是为进行宽带拨号而添加的新功能；AndroidQ PPPoE 的整体框图如下图所示：

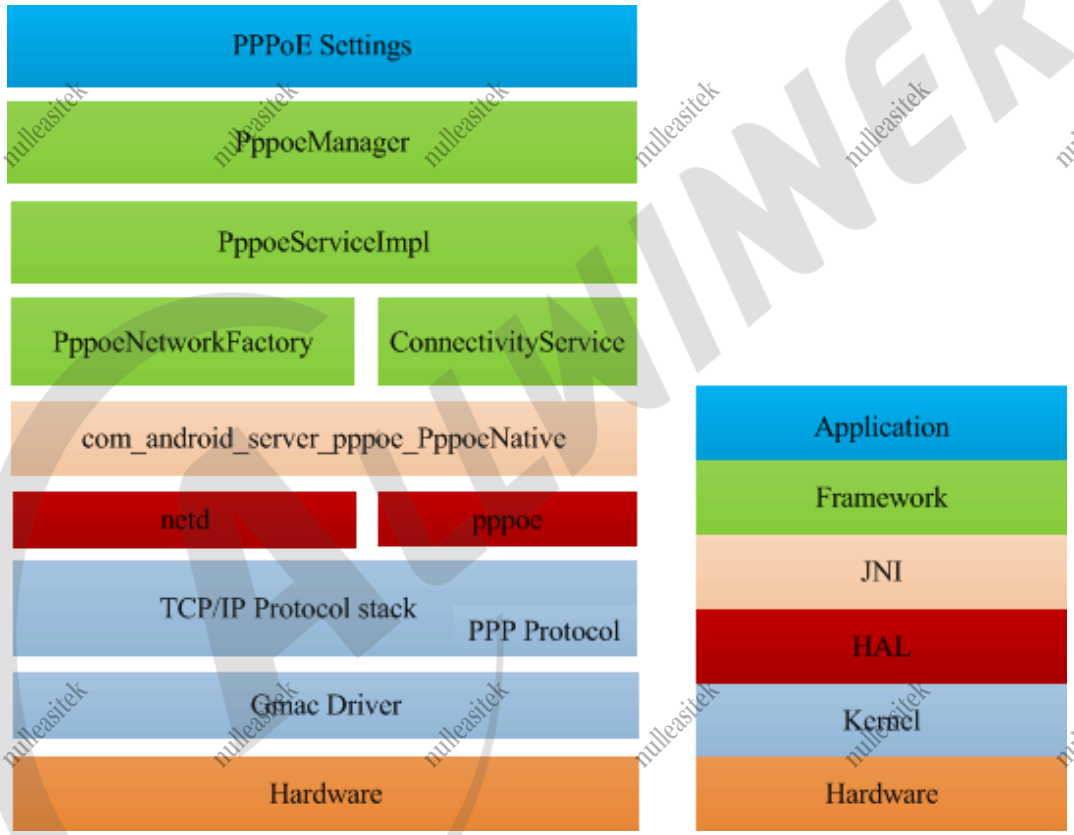

图 1: PPPoE 整体框架图

AndroidQ PPPoE 从大体结构上可以分为六个层次：Application、Framework、JNI、HAL、Kernel、Hardware。

Application 包括 PPPoE 设置 apk, 目前 AndroidQ 在 TvSettings 中开发此功能。

Framework 层中的 PppoeManager 类为应用层提供编程 API 接口，PppoeManager 通过 AIDL 与 PppoeService 关联，API 接口方法最终在 PppoeServiceImpl 类中实现；PPPoE 框架实现了一个 Pp-

poeNetworkFactoy 类，该类是由 PppoeServiceImpl 类创建，负责 PPPoE 的打开关闭、处理网口插拔事件、向应用层发送广播通知 PPPoE 状态变化等，该类会向 ConnectivityService 注册 mNetworkAgent 来更新 dns、路由等信息；ConnectivityService 和 PppoeNetworkFactory 都会透过 NetworkManagerService 来对网口进行操作。

JNI 层主要是 com_android_server_pppoe_PppoeNative 类，该类与 HAL 层的 pppoe 模块交互。

HAL 层包括 netd、pppoe 两个部分。pppoe 模块主要负责 PPPoE 拨号、PPPoE 链路维护；netd 则直接透过 netlink 机制与内核交互，主要功能有：接收网口 up/down 消息、网线 link in/out 消息、网口 add/remove 消息、配置网口 IP 地址、更新网口 DNS、添加删除路由表等。

Kernel 层中与网络相关的包括 TCP/IP 网络协议栈，Gmac 驱动；网络协议栈是网络报文接收与发送的必经之路，报文经过协议栈的传输后到达 Gmac 驱动，Gmac 驱动负责与具体的网口硬件交互。

Hardware 层包括以太网控制器和 PHY 两部分，负责在硬件上传输比特流。

2.2 DHCP/静态 IP 系统框图

AndroidQ dhcp 和静态 IP 整体框图如下所示：

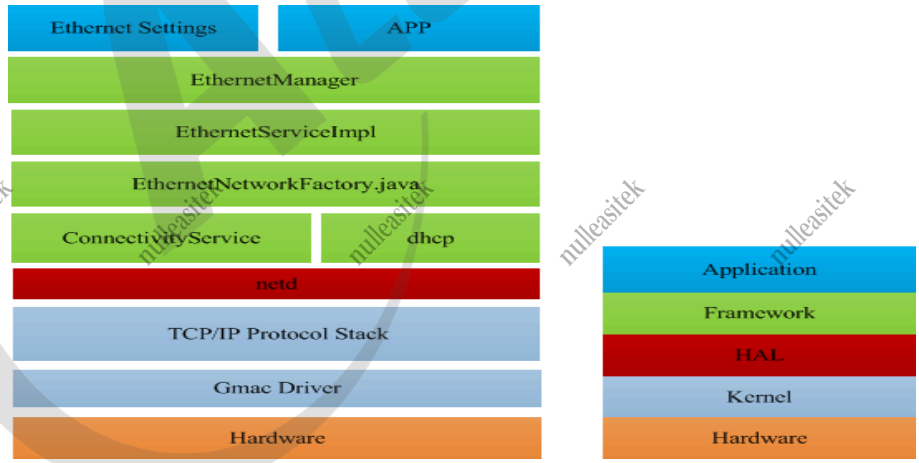

图 2: DHCP/静态 IP 整体框架图

可以看到 DHCP/静态 IP 与 PPPoE 的框架结构大致类似，在 Framework 层，EthernetManager 为应用层提供编程接口，原生代码里面 EthernetNetworkFactory 类的功能与 PPPoE 的 PppoeNetworkFactory 类相似；相比于 android 4.4，dhcp 的功能在 Framework 层直接实现。

3. 有线网络模块设计

3.1 有线网络源码分布图

3.1.1 PPPoE 功能源码分布

AndroidQ PPPoE 功能源码分布如下图所示：

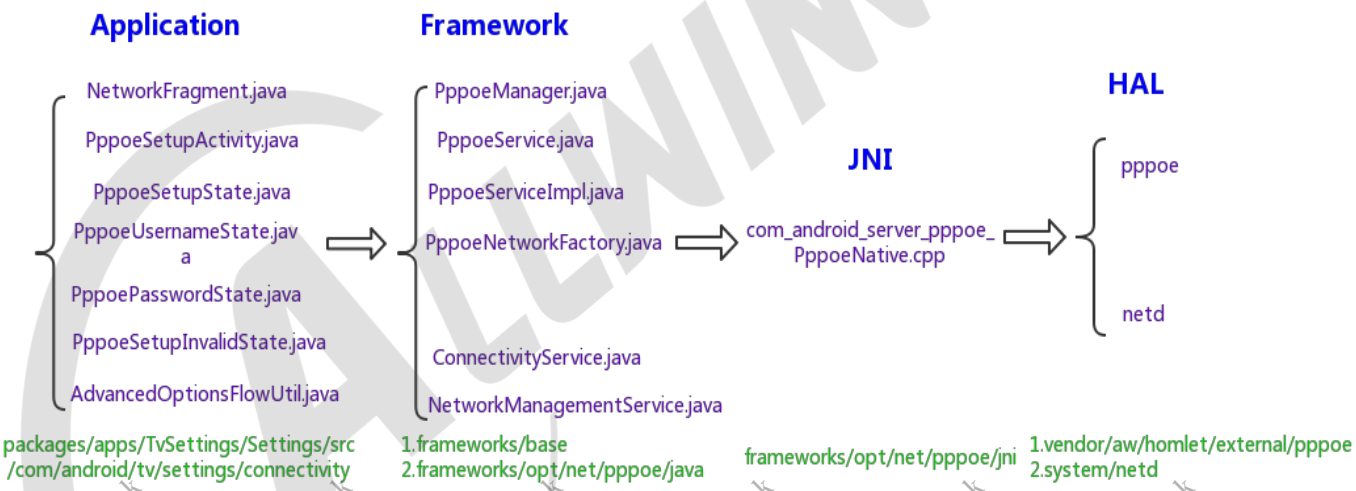

图 3: AndroidQ PPPoE 源码分布图

由于 AndroidQ 使用 TvSettings，因此 PPPoE 的相关功能也在 TvSetting 上进行了添加。PPPoE 设置的源码位于 packages/apps/TvSettings/Settings/src/com/android/tv/settings/connectivity 目录下。为了统一设计风格，代码是参考 TvSettings 中原生的以太网设置代码添加的。包括：NetworkFragment.java、PppoeSetupActivity.java、PppoeSetupState.java、PppoeUsernameState.java、PppoePasswordState.java、PppoeSetupInvalidState.java、AdvancedOptionsFlowUtil.java。AndroidQ 上 TvSettings 网络设置部分的代码使用状态机机制，每一个设置的界面一般都对应着一个状态，读者可以查阅 TvSettings 自带的以太网设置相关的代码了解这个风格，有助于理解 PPPoE 的代码。总的来说，相比之前 AndroidN 的 PPPoE Settings 的代码，只是界面逻辑上的不同，打开、关闭、配置的这些接口没有改变。

PPPoE 设置相关的 java 文件与作用：

1、NetworkFragment.java:此文件包含TvSettings中对wifi和以太网的控制逻辑。在此文件中添加了PPPoE打开、关闭、以及网络信息的显示的逻辑。
2、PppoeSetupActivity.java：状态流程的初始化。
3、PppoeSetupState.java：对应PPPoE Setup的第一个界面，主要是选择配置有线（ETH0）PPPoE还是无线(WLAN0)PPPoE。
4、PppoeUsernameState.java和PppoePasswordState.java：PPPoE账号密码的输入的界面，获取用户输入的信息。
5、PppoeSetupInvalidState.java：如果输入的用户名或者密码为空，则无效，会跳转至此界面。
6、AdvancedOptionsFlowUtil.java：里面添加了一个对输入的用户和密码判断处理的函数，processPppoeSetup。

Framework 层的源码主要位于 frameworks/base 与 frameworks/opt/net/pppoe 目录中, 其中 PppoeManager.java 向应用层提供 API，具体 API 的实现由 PppoeServiceImpl 类完成，该类主要与 PppoeNetworkFactory 进行交互，而 PppoeNetworkFactory 是实现 PPPoE 拨号功能的核心类，其主要作用如下：

1、负责应用层网络连接/断开的具体实现；
2、其内部类InterfaceObserver继承自BaseNetworkObserver类，用以捕获网络状态的改变，比如网线的插拔、网口的生成等；
3、通过networkAgent机制通知ConnectivityService网络状态的变化；

JNI 层主要是 com_android_server_pppoe_PppoeNative 类，主要负责 PPPoE 的打开关闭，以及获取 PPPoE 状态。

HAL 层包括 netd、pppoe 两个部分。netd 透过 netlink 机制直接与 Kernel 交互，主要功能有：接收网口 up/down 消息、网线 link in/out 消息、网口 add/remove 消息、配置网口 IP 地址、更新网口 DNS、添加删除路由表等；pppoe 模块主要负责 PPPoE 拨号、PPPoE 链路维护。

3.1.2 DHCP/静态 IP 源码分布

AndroidQ DHCP/静态 IP 功能源码分布如下图所示：

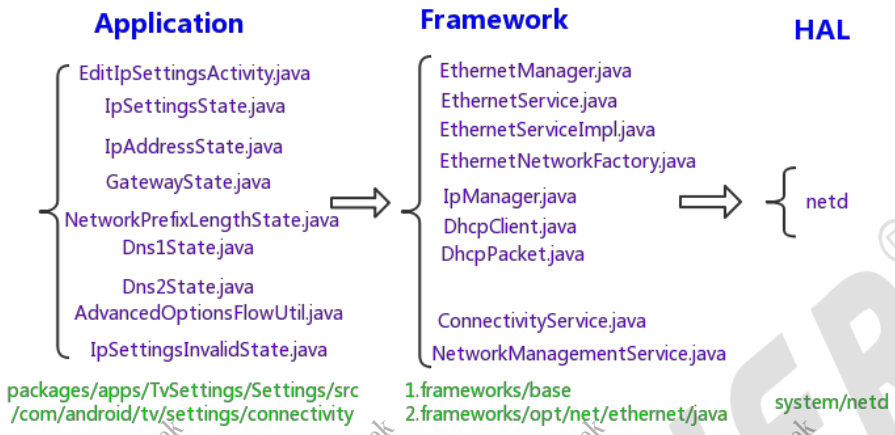

图 4: AndroidQ DHCP/静态 IP 源码分布图

DHCP/静态 IP 设置的源码位于 packages/apps/TvSettings/Settings/src/com/android/tv/settings/connectivity 目录下，此部分代码为 TvSettings 自带的代码。主要的代码在 EditIpSettingsActivity.java、IpSettingsState.java、IpAddressState.java、GatewayState.java、NetworkPrefixLengthState.java、Dns1State.java、Dns2State.java、IpSettingsInvalidState.java，以上代码主要是负责 DHCP 和静态 IP 信息的设置，上面 PPPoE 的代码就是参考这些代码添加的。

Framework 层的源码主要分布在 frameworks/base 目录以及 frameworks/opt/net/ethernet 目录，其中，EthernetManager.java 向应用层提供 API，具体 API 的实现由 EthernetServiceImpl 类完成，该类主要与 EthernetNetworkFactory 进行交互，而 EthernetNetworkFactory 是实现 DHCP/静态 IP 上网的核心类，其主要作用如下：

1、负责应用层网络连接/断开的具体实现；
2、其内部类InterfaceObserver继承自BaseNetworkObserver类，用以捕获网络状态的改变，比如网线的插拔、网口的生成等；
3、通过networkAgent机制通知ConnectivityService网络状态的变化；

IpManager.java、dhcpClient.java、dhcpPacket.java 负责 dhcp 获取 IP 地址等信息。

netd 的作用在 PPPoE 源码部分已做介绍，这里不在赘述。

3.2 PPPoE、DHCP、静态 IP 上网流程图

3.2.1 PPPoE 上网流程图

有线网络上网模式分为 DHCP、静态 IP 和 PPPoE 三种形式，其中，当选择 PPPoE 联网方式时(``设置"->" 网络和互联网" 中选择打开 PPPoE 上网开关，或者插入网线)，上网流程图如下图所示：

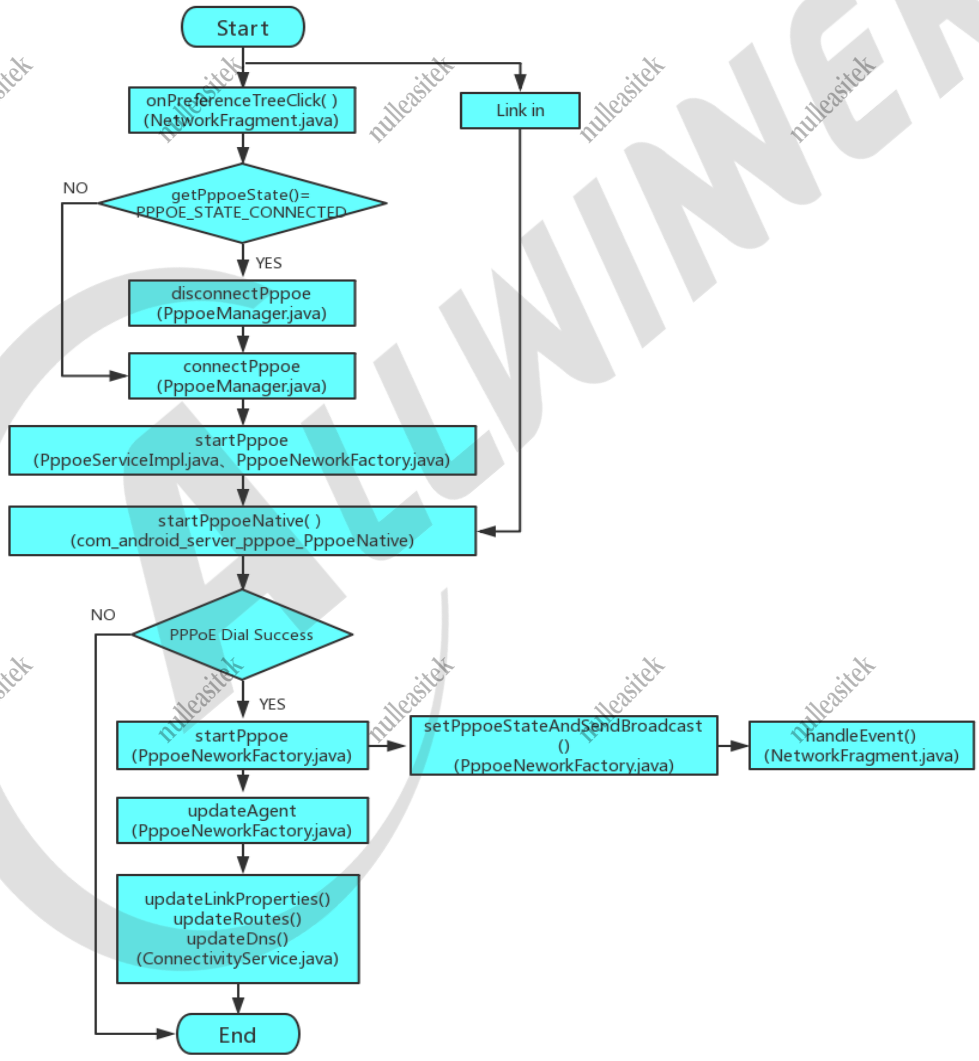

图 5: AndroidQ PPPoE 上网流程图

当在" 设置"->" 网络和互联网" 打开 PPPoE 开关后，首先判断如果 PPPoE 已经连接上则首先调用 disconnectPppoe 断开 PPPoE 连接，然后调用 connectPppoe 重新进行连接，最终调用到 JNI 层的 startPppoeNative 函数，该函数调用 HAL 层的 pppoe 模块进行拨号，如果拨号失败，则整个拨号过程就退出；如果拨号成功，则会首先调用 setPppoeStateAndSendBroadcast 向应用层发送连接成功的消息，应用层收到此消息用以更新 PPPoE 的网络信息，然后调用 updateAgent 通知 ConnectivityService，在 ConnectivityService 中主要的工作是更新当前网络接口的路由和 dns 信息。

3.2.2 DHCP/静态 IP 上网流程图

DHCP/静态 IP 上网流程图如下图所示：

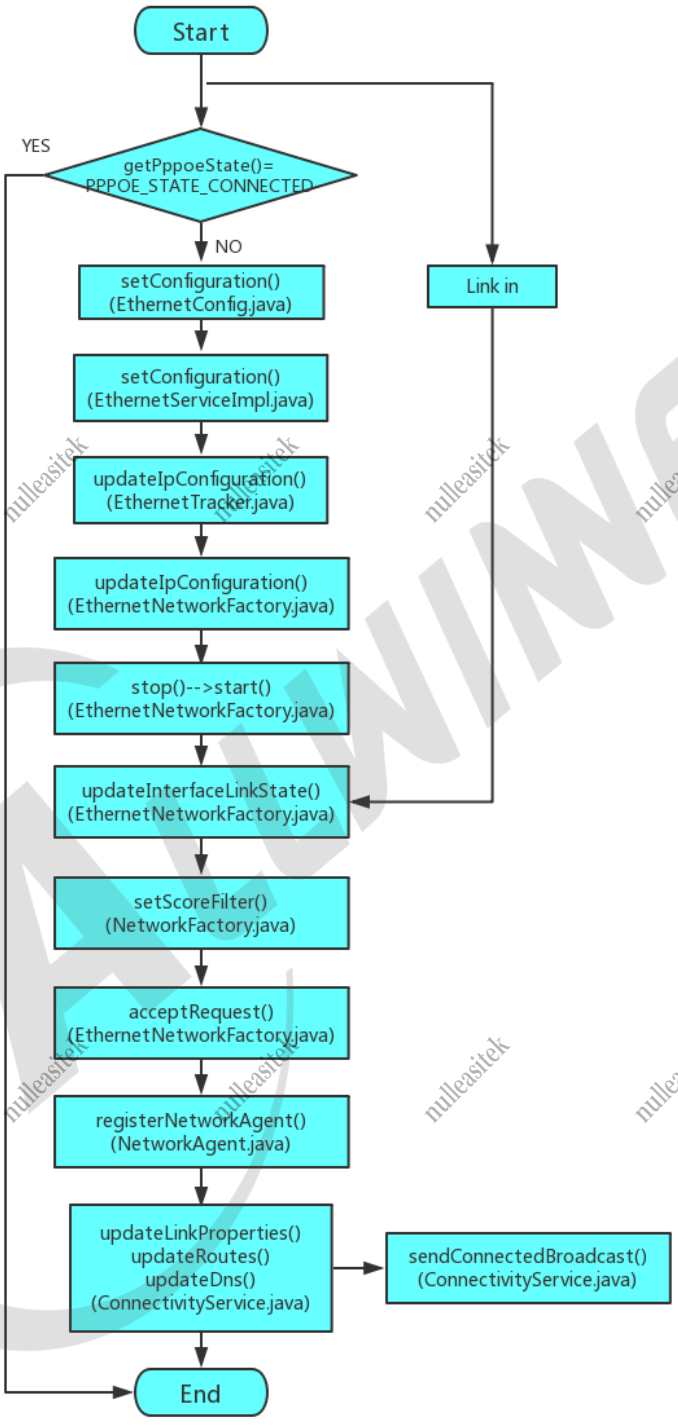

图 6: AndroidQ DHCP/静态 IP 上网流程图

默认 PPPoE 的优先级高于 DHCP/静态 IP，所以在设置中首先判断如果 PPPoE 已经连接上，则不会尝试进行 DHCP/静态 IP 的连接，如果 PPPoE 断开，则会尝试连接，首先调用 setConfiguration 传入需要设置的参数信息。后面调用了 setScoreFilter 方法进行网络评分，然后调用 onRequestNetwork 方法，对于静态 IP，透过 netd 设置其 IP 地址，对于 DHCP，则通过发送 dhcp 报文获得 ip 地址、网关、dns 等信息。在 onRequestNetwork 方法中会调用 registerNetworkAgent 方法向 ConnectivityService 进行注册，在 ConnectivityService 中会设置当前网口的路由以及 dns 信息，然后调用 sendConnectedBroadcast 方法通知应用层，应用层收到此广播后更新网络显示信息。

3.3 有线网络/无线网络切换示意图

在 AndroidQ 中有线网络的优先级要高于 wifi（优先级：PPPoE > DHCP/静态 IP >wifi），也就是说当有线网络和 wifi 同时连接时，默认使用有线网络，当有线网络断开时，使用 wifi 进行上网。在有线和无线网络同时连接后，其网络切换示意图如下所示：

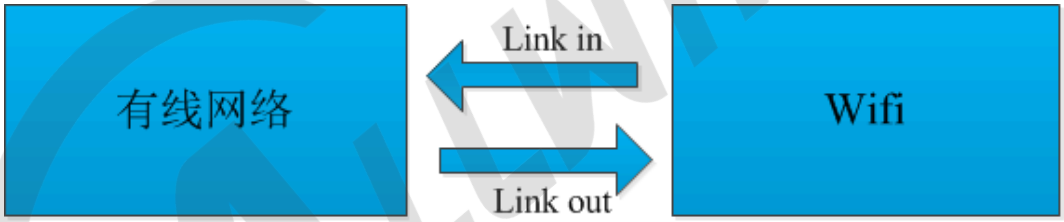

图 7: AndroidQ 有线/无线网络切换示意图

在两者同时连上的情况下，当拔出网线后，则有线网络断开，切向 wifi 上网；当插入网线后再切回有线网络上网。

4. 模块使用

4.1 PPPoE 部分

4.1.1 主要变量

PppoeManager.java 类中的主要变量定义如下：

```
public static final String EXTRA_PPPOE_ERRMSG = "pppoe_errmsg";
public static final String PPPOE_STATE_CHANGED_ACTION =
                                        "android.net.pppoe.PPPOE_STATE_CHANGED";
public static final String EXTRA_PPPOE_STATE = "pppoe_state";
public static final String EXTRA_PREVIOUS_PPPOE_STATE = "previous_pppoe_state";

public static final int PPPOE_STATE_DISCONNECTED = 0;
public static final int PPPOE_STATE_CONNECTING = 1;
public static final int PPPOE_STATE_CONNECTED = 2;
public static final int PPPOE_STATE_DISCONNECTING = 3;

public static final int PPPOE_EVENT_CONNECTING = 0;
public static final int PPPOE_EVENT_CONNECT_SUCCESSED = 1;
public static final int PPPOE_EVENT_CONNECT_FAILED = 2;
public static final int PPPOE_EVENT_DISCONNECTING = 3P;
public static final int PPPOE_EVENT_DISCONNECT_SUCCESSED = 4;
```

4.1.2 API 方法

```
类名：android.net.PppoeManager
PppoeManager mPppoeManager = (PppoeManager )getSystemService(
                                        Context.PPPOE_SERVICE)

作用： 返回一个PppoeManager对象；
参数： Context.PPPOE_SERVICE；
返回值：PppoeManager对象；
```

pubilc List<String> getPppoeUserInfo(String iface)
作用： 获取PPPoE的账户名和密码
参数： iface，物理网口名称，例如eth0、wlan0
返回值：List中第一个元素保存账户名，第二个元素保存账户密码

public boolean connectPppoe(String iface)
作用： 开启拨号，且将PPPoE已启用信息保存到SettingProvider中
参数： 物理网口名称
返回值：true：成功，false：失败

public boolean disconnectPppoe(String iface)
作用： 断开拨号，且将PPPoE已禁用信息保存到SettingProvider中
参数： 物理网口名称
返回值：true：成功，false：失败

public int getPppoeState(String iface)
作用： 获取当前PPPoE的状态，该状态在framework中维护
参数： 物理网口名称
返回值：
PPPOE_STATE_DISCONNECTED： PPPoE连接已断开
PPPOE_STATE_CONNECTING： PPPoE正在连接
PPPOE_STATE_CONNECTED： PPPoE已连接
PPPOE_STATE_DISCONNECTING：PPPoE正在断开连接

public boolean isPppoeEnabled()
作用： 判断PPPoE功能是否已启动
参数： 无
返回值：true：已启用，false：已禁用

public boolean getPppoeInterfaceName()
作用： 获取PPPoE使用的网络接口名称
参数： 无
返回值：eth0或者wlan0。

public LinkProperties getPppoelinkProperties()
作用： 获取PPPoE连接的网络信息；
参数： 无

返回值：LinkProperties类对象；

4.1.3 广播定义

事件通知
Action：
public static final String PPPOE_STATE_CHANGED_ACTION =
"android.net.pppoe.PPPOE_STATE_CHANGED"

Extra:
key : public static final String EXTRA_PPPOE_STATE =
"pppoe_state"
value:
PPPOE_EVENT_CONNECTING = 0
PPPOE_EVENT_CONNECT_SUCCESSED = 1
PPPOE_EVENT_CONNECT_FAILED = 2
PPPOE_EVENT_DISCONNECTING = 3
PPPOE_EVENT_DISCONNECT_SUCCESSED = 4

如果PPPoE是PPPOE_EVENT_CONNECT_FAILED状态，则需要携带错误信息：
key: public static final String EXTRA_PPPOE_ERRMSG =
"pppoe_errmsg"
value:
1、FATAL_ERROR
2、OPTION_ERROR
3、NOT_ROOT
4、NO_KERNEL_SUPPORT
5、USER_REQUEST
6、LOCK_FAILED
7、OPEN_FAILED
8、CONNECT_FAILED
9、PTYCMD_FAILED
10、NEGOTIATION_FALIED
11、PEER_AUTH_FAILED
12、IDLE_TIMEOUT
13、CONNET_TIME
14、CALLBACK
15、PEER_DEAD
16、HANGIP
17、LOOPBACK

18、INIT_FAILED
19、AUTH_TOPEER_FAILED
20、TRAFFIC_LIMIT
21、CHID_AUTH_FAILED
22、LINK OUT

4.2 DHCP/静态 IP 部分

4.2.1 API 方法

```
EthernetManager mEthManager = (EthernetManager)getSystemService(
                                        Context.ETHERNET_SERVICE)
作用： 返回一个EthernetManager对象；
参数： Context.ETHERNET_SERVICE;
返回值：EthernetManager对象；
```

```
public void setConfiguration(IpConfiguration config)
作用： 开启DHCP/静态IP上网；
参数： config，对于DHCP，给IpConfiguration类的ipAssignment变量赋值即可（值为DHCP）；对于静态IP,
       则还需要设置IP地址、网络前缀长度、网关、dns等信息。
返回值：无
```

```
public IpConfiguration getConfiguration()
作用： 获取DHCP/静态IP上网信息；
参数： 无;
返回值：IpConfiguration类对象；通过该对象访问ipAssignment变量即可判断当前的是DHCP上网还是静态IP上网。
```

```
public boolean isAvailable()
作用： 判断DHCP/静态IP是否可用；
参数： 无;
返回值：true表示DHCP/静态IP可用，false表示不可用。
```

```
ConnectivityManager mConnectivityManager = (ConnectivityManager)getSystemService(
                                                    Context.CONNECTIVITY_SERVICE)
作用： 返回一个ConnectivityManager对象；
参数： Context.CONNECTIVITY_SERVICE；
返回值：ConnectivityManager对象；

public LinkProperties getLinkProperties(Network network)
作用：返回当前network的连接信息，包括IP地址、路由、dns等；
参数：network
返回值：LinkProperties类对象。
```

4.2.2 广播定义

```
Action：
public static final String CONNECTIVITY_ACTION =
                        "android.net.conn.CONNECTIVITY_CHANGE";
```

```
Extra:
key: public static final String EXTRA_NETWORK_TYPE = "networkType";
value:

        public static final int TYPE_WIFI = 1;
        public static final int TYPE_ETHERNET = 9;
        public static final int TYPE_PPPOE = 18;
```

4.3 编程示例

4.3.1 PPPoE 编程示例

PPPoE 设置的代码在 AndroidN 和 AndroidQ 区别主要在界面逻辑上，调用的接口没有改变。由于 AndroidQ TvSettings 上添加修改的 Java 文件比较多，读者如果有我司 AndroidQ 的 PPPoE 设置的代码，可以先了解 AndroidQ 上的实现方式。因此在此列出 AndroidN 和 AndroidQ 的源码以及简单介绍。

4.3.1.1 AndroidN

PPPoE 相关的接口使用请参考 Setting 中的 PPPoE 部分，源码位置：package/app/Settings/src/com/android/settings/pppoe/ 其中相关类的作用描述如下：
PppoeSettings.java: PPPoE 的打开关闭、网络信息显示；
PppoeModeDialog.java: 设置 PPPoE 使用的网口名、账户、密码；

4.3.1.2 AndroidQ

PPPoE 相关的接口使用请参考 TvSettings 中的 PPPoE 部分，源码位置：packages/apps/TvSettings/Settings/src/com/android/tv/settings/connectivity/ 其中相关类的作用描述如下：
NetworkFragment.java: 此文件包含 TvSettings 中对 wifi 和以太网的一些控制逻辑。在此文件中添加了 PPPoE 的打开、关闭、以及网络信息的显示的逻辑。
PppoeSetupActivity.java：状态流程的初始化。
PppoeSetupState.java：对应 PPPoE Setup 的第一个界面，主要是选择配置有线（ETH0）PPPoE 还是无线 (WLAN0)PPPoE。
PppoeUsernameState.java 和 PppoePasswordState.java：PPPoE 账号密码的输入的界面，获取用户输入的信息。
PppoeSetupInvalidState.java：如果输入的用户名或者密码为空，则无效，会跳转至此界面。
AdvancedOptionsFlowUtil.java：里面添加了一个对输入的用户和密码判断处理的函数，processPppoeSetup。

4.3.2 DHCP/静态 IP 编程示例

DHCP/静态 IP 相关的代码在 AndroidQ TvSetting 上是自带的，读者同样有两种渠道了解。

4.3.2.1 AndroidN

DHCP/静态 IP 相关的接口使用请参考 Setting 中的以太网部分，源码位置：package/app/Settings/src/com/android/settings/ethernet/ 其中相关类的作用描述如下：
EthernetSettings.java、EthernetEnabler.java：DHCP/静态 IP 打开关闭、网络信息显示；

DhcpModeDialog.java：DHCP 设置；
StaticModeDialog.java：静态 IP 设置；

4.3.2.2 AndroidQ

DHCP/静态 IP 相关的接口请参考 TvSettings 中的以太网部分，源码位置：packages/apps/TvSettings/Settings/src/com/android/tv/settings/connectivity/ 其中相关类的作用描述如下：
EditIpSettingsActivity.java：状态流程的初始化。
IpSettingsState.java: 选择 DHCP 和静态 IP 的界面。
IpAddressState.java：IP 地址设置界面。
GatewayState.java：网关设置界面。
NetworkPrefixLengthState.java: 网络前缀长度设置界面。
Dns1State.java、Dns2State.java: DNS1 和 DNS2 设置界面。
AdvancedOptionsFlowUtil.java：对设置好的信息进行处理。
IpSettingsInvalidState.java: 如果输入的配置无效，会跳转至此界面。

5. FAQ

5.1 如何解决 PPPoE 拨号失败的问题

导致 PPPoE 拨号失败的原因有很多，从用户的角度，可以检查账号密码是否正确；联系运营商查看是否欠费或者账号是否异常。从开发者的角度，可以通过查看 PPPoE 拨号失败的错误信息或者利用 wireshark 工具进行抓包分析。

5.2 如何利用机顶盒开启热点

首先需要保证有线网络可以成功上网，然后在" 设置"->" 网络和互联网"->"WiFi 热点" 设置 wifi 热点名称和密码，然后打开热点开关，利用手机等可以连接 wifi 热点并能成功上网。

6. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.