----------人生不设限----------

Author: james
E-mail: jian-yang@tcl.com
Create time:16/12/01

简介：VoWIFI 内容主要包含网络基本知识 、MBN机制&&命令&&工具的使用、Android Code 部分理解，目前我所涉及到的技能，待后面学习后继续补充。（ 中间有很多我自己遇到的困难和解决过程将详细记录，避免大家重复入坑 ）
这篇文档将被我视为我的转正文档，希望大家积极提问和不吝赐教。

I 网络基本知识

IMS ：IP Multimedia Subsystem (IMS) core network
IMS 相关的技术："VoLTE", "ViLTE", "VoWiFi", "ViWiFi", "UtLTE", "UtWiFi"
代码 ImsPhoneCallTracker.java

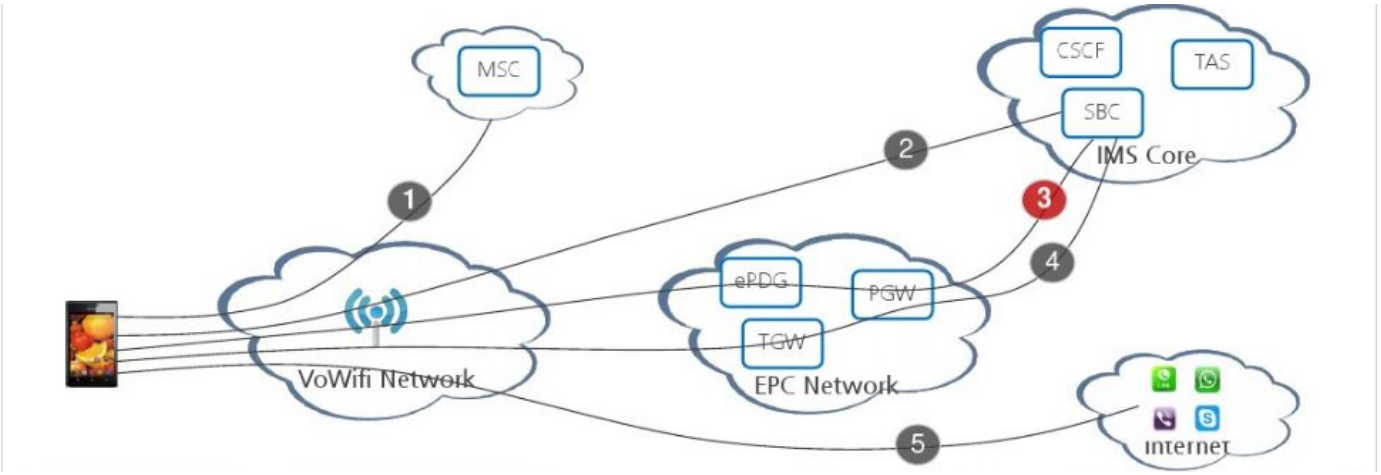
```
1.  public final class ImsPhoneCallTracker extends CallTracker {
2.  static final String LOG_TAG = "ImsPhoneCallTracker"
3.  //Indices map to ImsConfig.FeatureConstants
4.  /**
5.  * ImsFeatureEnabled 值代表 ImsFeatureStrings 中功能当前是否支持
6.  * 例如 当 mImsFeatureEnabled = {true, false, false, false, false, false}
7.  * 说明 此时支持VoLTE状态 其他功能都不支持
8.  */
9.  private boolean[] mImsFeatureEnabled = {false, false, false, false, false, false};
10. private final String[] mImsFeatureStrings = {"VoLTE", "ViLTE", "VoWiFi", "ViWiFi",
11. "UtLTE", "UtWiFi"};
```

a. "VoLTE": VoLTE即Voice over LTE，它是一种IP数据传输技术，无需2G/3G网，全部业务承载于4G网络上，可实现数据与语音业务在同一网络下的统一，是基于IMS 的语音业务
b. "ViLTE"：ViLTE 即Video over LTE， 它和VoLTE 一样通过 Sip （ Session Initiation Protocol ）消息去和网络交互
c. "VoWiFi"：VoWIFI即Voice over Wi-Fi 终端连入Wi-Fi ，在通过其他WiFi的服务器连入IMS网络 实现语音通话
d. "ViWiFi"：ViWIFI即Video over Wi-Fi 终端连入Wi-Fi ，在通过其他WiFi的服务器连入IMS网络 实现 视频通话
UtLTE UtWIFI 暂时没有资料，没有使用

VoWIFI

区别于 VoIP（举例 微信语音和视频），无需第三方应用，应用不需要在线，直接相当于打电话免费

原理图一（架构对比）



EPC ：Evolved Packet Core 演进的分组核心网 该流程中同时包含了MAPCON流程（即P-GW地址通知流程)

SBC ：Session Border Controller 会话边界控制器

CSCF ：Call Session Control Function 呼叫对话控制功能

TAS：为用户提供与IMS AS之间的同步功能，完成补充业务

AS： Application Server 应用服务器

线路 一 ：基于CS方案的VoWiFi，TMO之前提供的方案，（已不支持）

MSC ：Mobile Switching Center 移动交换中心

线路 二：Wi-Fi直接连入IMS网络 无法保证 WLAN-LTE切换前后UE地址一致 （技术上实现有困难）

SBC ：Session Border Controller 会话边界控制器

线路 三：Untrusted Acess 目前VoWiFi 方案

公共WLAN非可信接入3GPP网络（S2b接口）

ePDG Evoloed packet Data Gateway

PGW作为WLAN-LTE切换的锚点 PGW ：PDN Gateway 分组数据网网关

要求UE支持IPsec以及IKEV2鉴权

已完成标准化 可商用

线路 四：trusted Acess

运营商部署的WLAN可信接入3GPP网络（S2a接口）

TGW 用来本地分流网络

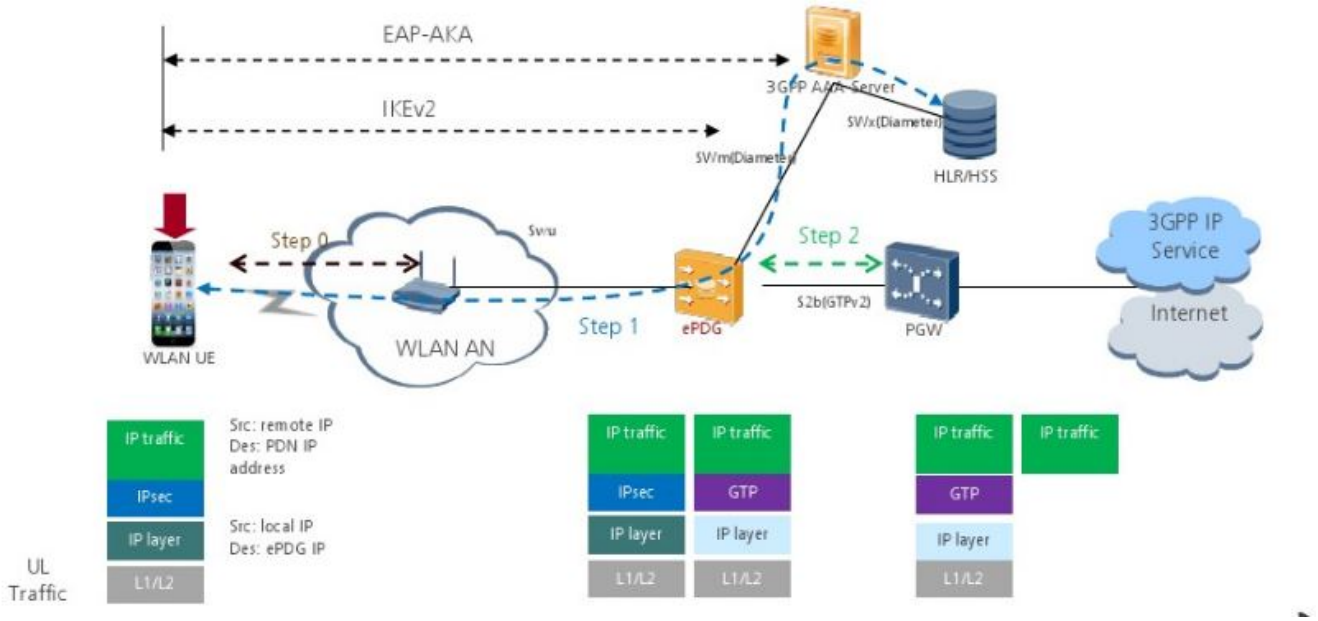
PGW作为WLAN-LTE切换的锚点

WiFi-LTE切换标准还在讨论

线路 五：OTT Solution

需要下载APP 不支持call waiting /call forwarding等功能 VoIP

线路 三 原理图



step 0 ：UE执行WLAN认证，WLAN网络分配给UE一个local IP

step 1：UE 通过本地配置或者DNS机制获取ePDG的地址，发起IKEv2 /EAP-AKA流程

EAP-AKA认证

成功认证后，ePDG从AAA获取UE的签约数据

step 2 ：ePDG 和PGW之间建立GTP隧道 PGW分配远端IP地址以及P-CSCF地址通过GTP消息带给ePDG

step 3 ：IKEv2过程结束，ePDG会将远端 UE的IP地址 发送给UE

ePDG的理解

演进的分组数据网关（ePDG）：ePDG的主要功能是确保数据传输的UE通过不可信的非3GPP接入网连接到EPC。为了这个目的，在ePDG与UE之间建立一个IPsec隧道。

要求终端支持 Ipsec 和 IKEv2 协议 数据业务连到 internet 语音业务 IPSec 连接到 ePDG

UE中如何设置 连接到ePDG：

1. UE本地配置ePDG的地址
2. UE根据PLMN构造ePDG的FQDN，到Internet DNS做域名查询 获取ePDG的地址

FQDN ：Fully Qualified Domain Name 全称域名

UE 采用1.静态配置ePDG地址形式 UE选择ePDG是固定的，因此ePDG间将无法做到负荷分担

举例：TMO 在文件iwlan_s2b_config.txt 配置如下

epdg_ipv4_address:208.54.73.77;

UE 若采用2. 根据vPLMN信息构造ePDG 的FQDN ，ePDG的FQDN的格式如下：

epdg.epc.mnc<MNC>.mcc<MCC>.pub.3gppnetwork.org;

举例：TMO 在文件iwlan_s2b_config.txt 配置如下

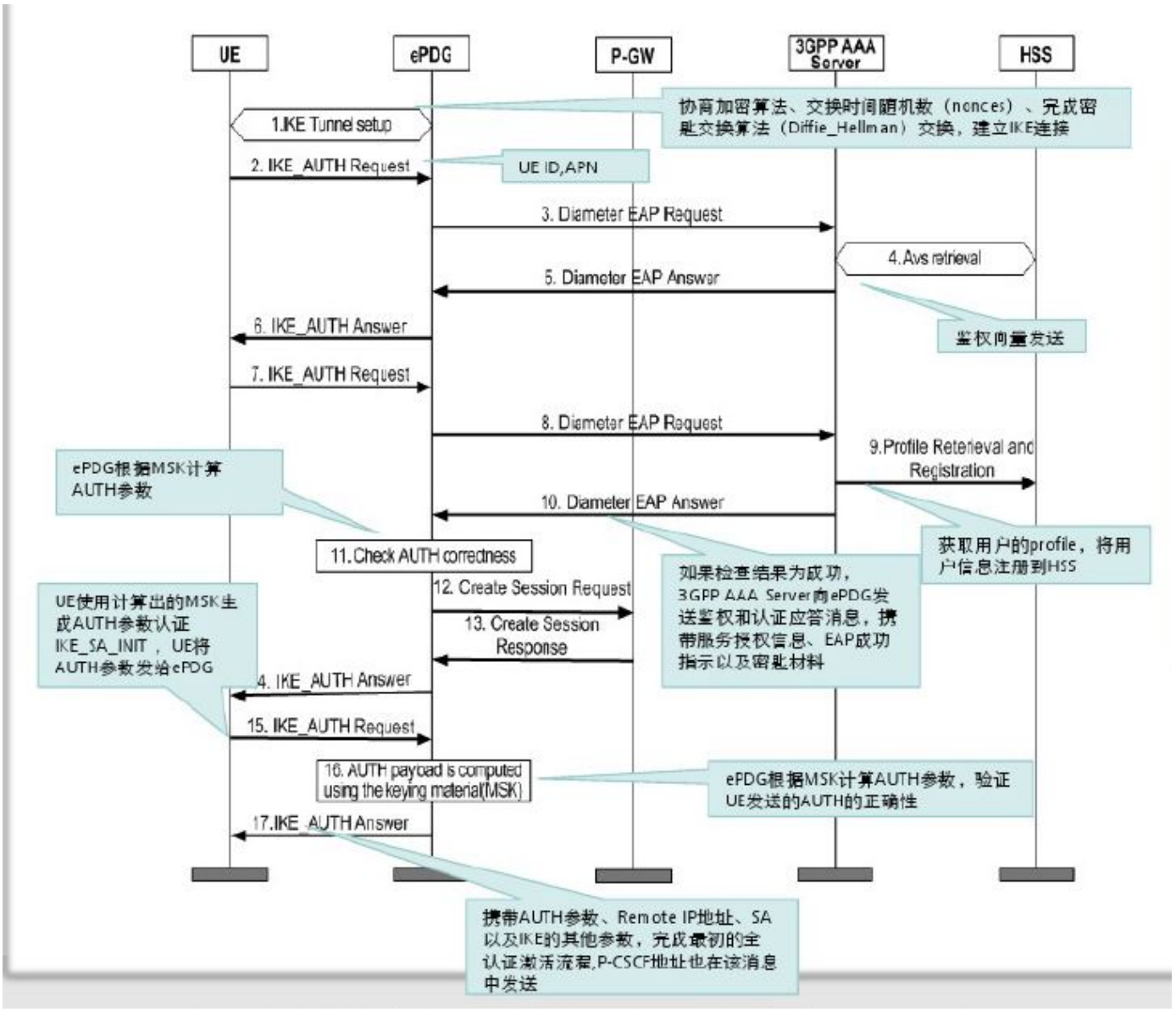
epdg_fqdn:ss.epdg.epc.mnc260.mcc310.pub.3gppnetwork.org;

UE 构造域名后，到internet DNS做域名解析，获取多个ePDG地址，ePDG地址间可以可以负荷分担

同一个FQDN支持对应多个ePDG设备 UE根据DNS返回的ePDG列表 选择一个ePDG接入 ，如果不行选择下一个

由于UE选择归属地网络的ePDG接入，没有漫游的场景

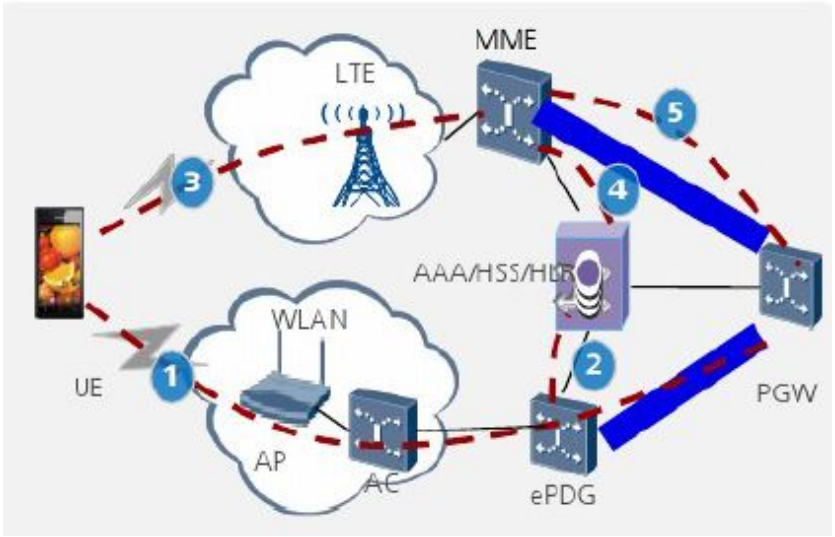
WLAN S2b架构的PDN连接的建立（对应线路 三 原理图）



Handover 概念

Vowifi 和 VOLTE 相互无缝切换的技术 主要涉及概念

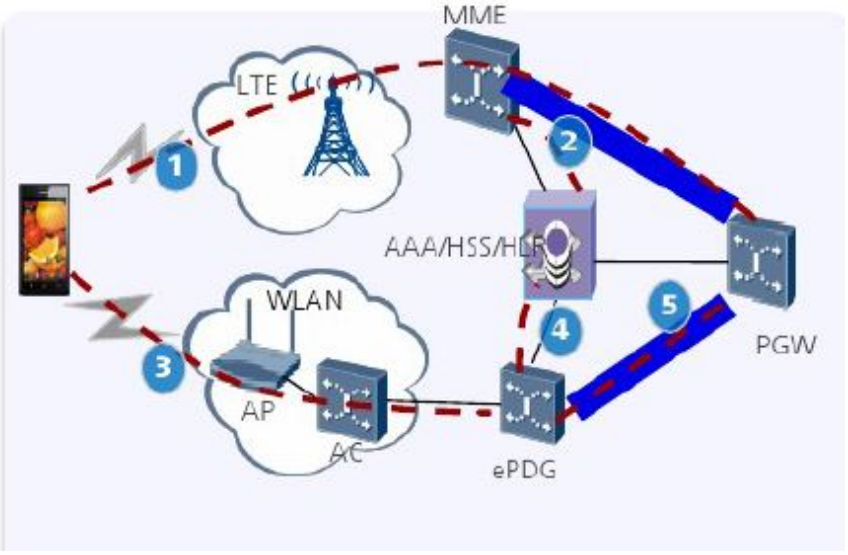
Wi-Fi Acess >> LTE 原理图



1. UE 通过S2b非可信接入到ePDG 和PGW
2. AAA会将APN和PGW信息注册到HSS
3. UE 切换到LTE 发起Handover 接入
4. MME从HSS获取APN和PGW信息
5. MME 选择和切换前相同的PGW接入 PGW删除老侧资源

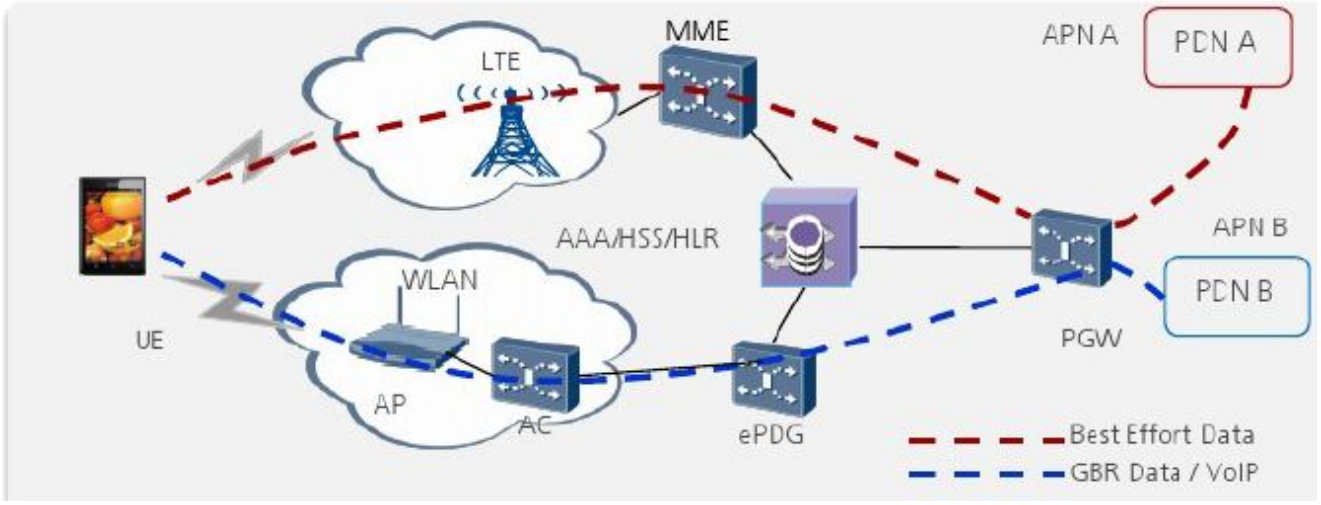
补充 HLR ：Home Location Register 归属位置寄存器 HSS ：Home Subscriber Server 归属用户服务器
补充 MME： Mobility Management Entity 移动管理实体

LTE >> Wi-Fi Acess 原理图



1. UE接入到LTE 网络
2. MME将APN和PGW信息注册到HSS
3. UE 切换到Wi-Fi网络
4. ePDG 从HSS中获取APN和PGW信息
5. ePDG选择和切换前相同的PGW接入 PGW删除老侧资源

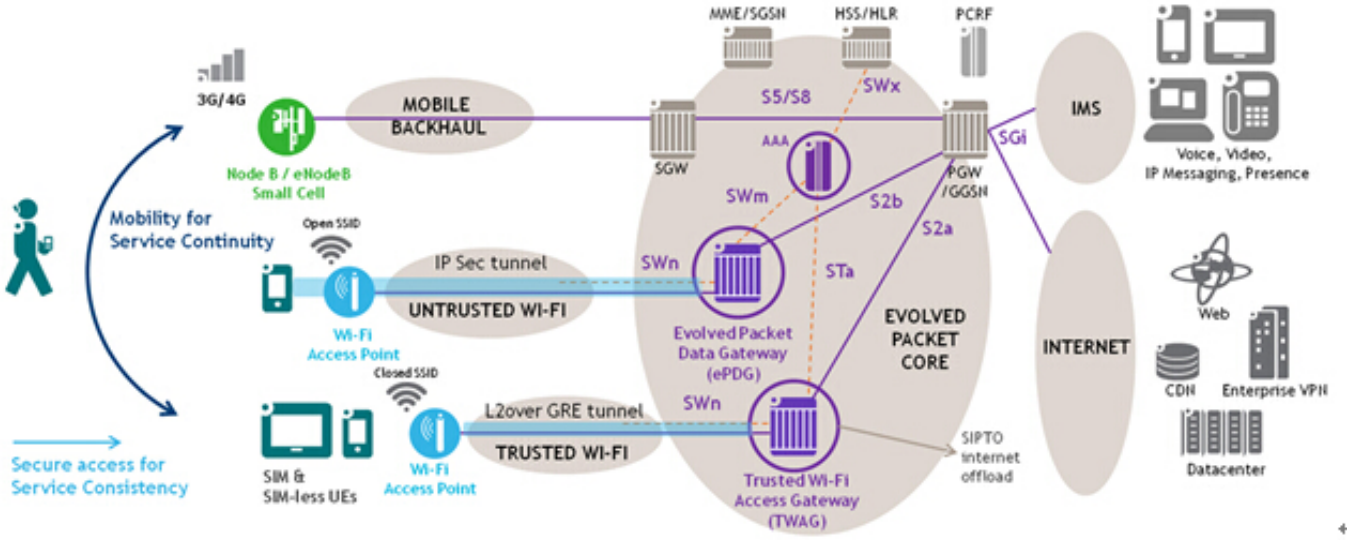
LTE 和 WLAN的共同接入



UE 同时注册上LTE和WLAN网络 LTE和WLAN网络分别针对不同的APN对应的PDN连接
PGW/AAA 支持将UE PDN连接的APN和PGW信息注册上HSS
MME 支持将UE在LTE的 PDN连接的APN和PGW信息注册上HSS，HSS将其更新至AAA,上述场景保证MAPCON切换的连续性

MAPCON：基于PDN的粒度进行切换的MAPCON（Mutiple Access PDN Connetion）技术
是指用户在不同的接入系统中建立的是不同的PDN连接，即不同的接入系统中的PDN连接必须使用不同的APN
MAPCON技术的实现比较简单，用户的附着流程基本没有改动，只需分别在两个系统进行附着。在切换流程中，如果用户已经在两个系统都附着了，只需要把一个系统上要切换的业务切换到另一个系统上，在目标消息中为其建立相应的PDN连接，并释放源系统中相应的承载即可；如果用户只在源系统进行了附着，那么在切换前，用户需要先在目标系统执行附着流程，并能够提供要切换的PDN连接的APN。此外，MAPCON对HSS和AAA有所增强，主要是要实现PDN GW信息的动态推送，PDN GW信息原来是在切换流程中通过接入认证过程获得的。

完整的VoLTE VoWi-Fi 原理图

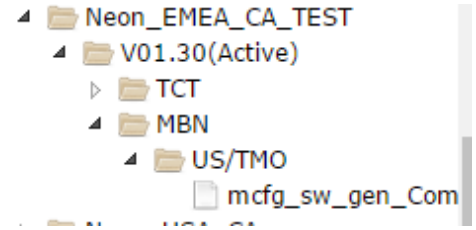


II VoWi-Fi Modem 配置

原理：将carrier的的MBN上传到QCN网站上，编译时从网站上取数据，编成Tarball 或者Default MBN
小白到入门我经历了 纯手配 、编tarball && default MBN 、Black version 三个过程（调试都是基于TMO的卡）

一 纯手动配置过程（MBN 部分 Code部分 APN部分）

ps:高通的已经帮我们配置好TMO的MBN ，基于已有的条件修改的
步骤：
1. gedit
amss_8976/modem_proc/mcfg/mcfg_gen/generic/NA/TMO/mcfg_sw_gen_Commercial.xml
2. 将文件上传到QCN网站 地址：wukong>Develop>QCN



注意事项 ：选中上传MBN文件 必须填写一致

文件的名称 Carrier Index Version 选择 SW的MBN

```
<?xml version="1.0" encoding="utf-8"?>
<NvData>
    <NvConfigurationData carrierIndex="1" version="0x07010519" type="1"/>
    <NvItemData id="71" mcfgAttributes="0x19" mcfgVariant="1">
        <Member sizeOf="1" type="uint8">7 </Member>
        <Member sizeOf="13" type="uint8">84 77 79 </Member>
    </NvItemData>
```

对应QCN填写参数

> TCT 目录下最基本的配置
> MBN 中时不同运营商的配置（优先级高于TCT下的NV参数）
3. gedit vendor/tct/source/qcn/auto_make_tar/product_parameter_vesrion.xml
在idol4 项目中添加参数

```
<parameter name="Neon_EMEA_CA_TEST" version="V01.30" output="neon_emea_ca_test_op.tar" perso="true" server="sh" use_mcfg_mbn="true">
    <mbn name="TMO" type="mcfg_sw" path="mbn/mcfg_sw_mbn/US/TMO/VoLTE/Commercial/mcfg_sw.mbn" />
</parameter>
```

> parameter name 是QCN网站上的 建立的文件的名称
> parameter version 是版本号
> parameter output 是版本号对应编出tar包的名字
> parameter use_mcfg_mbn 如果使用TMO MBN 需要增加这个参数为True
> mbn name TMO的就设为TMO
> mbn path 是指本地基线代码的路径不是QCN网站的路径　其中US/TMO 必须和QCN网站上一样 还需要在签名中查看是否是添加

注意事项：经常配置新的运营商编译失败
问题在 MBN path不对
需要gedit vendor/tct/source/qcn/auto_make_tar/signmbn_idol4/st.sh 确认是否有path

```
case $3 in

....
    TMO)
        qdsp6sw=mbn/mcfg_sw_mbn/US/TMO/VoLTE/Commercial/mcfg_sw.mbn
```

signmbn_idol4 是签名文件，详细问谢晨

4.开始编译 tarll ball （使用的动态MBN）

1. 进入项目路径 执行命令
2. source build/envsetup.sh
3. choosecombo
4. make tar j16

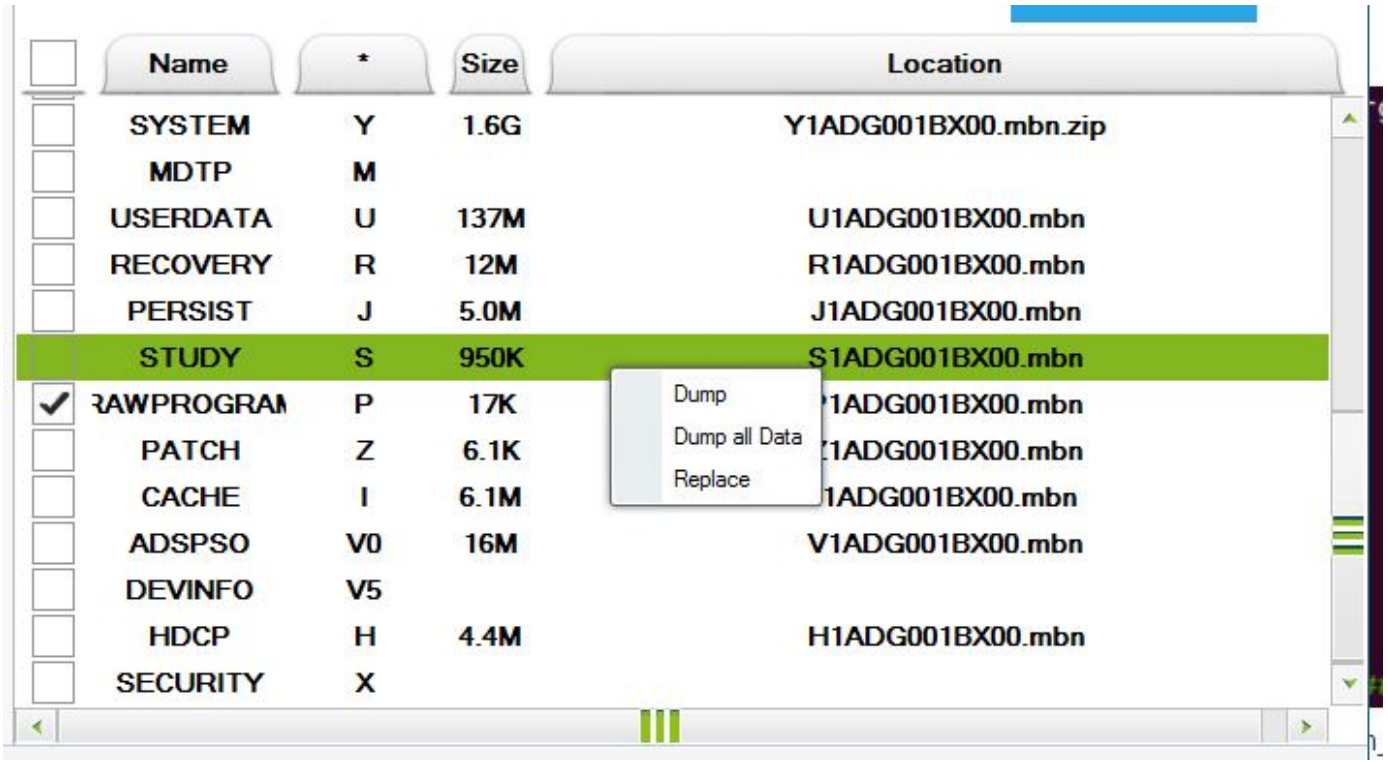
生成 tarball

```
Please find it at folder: /local/Code/idol4/out/target/product/idol4/tarball
Generate a splash.img from the picture
6
116166
932410
1248254
848898
2183896
961832
3219451
974853
spalsh image size:  4194304
Build splash.img done.

Auto Generate Tarball End!


#### make completed successfully (04:29 (mm:ss)) ####
```

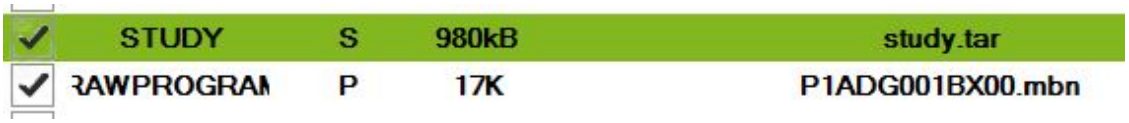
5. cd /local/Code/idol4/out/target/product/idol4/tarball 会看见tarball

```
neon_emea_ca_test_op.tar
```

6. 接下来需要将tarball 刷入手机 有两种方式
   1. fastboot 命令刷 （目前怀疑刷入手机有时候有问题）
   2. TeleWeb 刷入（建议使用TeleWeb 刷 ，目前没有出现不生效的情况）
7. 使用TeleWeb 刷 将tarball 改为 study.tar mv neon_emea_ca_test_op.tar study.tar
8. 在TeleWeb 上勾选P文件 替换 S文件



9.替换完 点击Download 按钮



10. reset 手机后 ，VoWi-Fi 没有注册上 其中一个原因是因为 从QXDM log中发现 只有第一步 第二步流程没有发起

先检查 手机终端CNEservice 有没有起来 com.quicinc.cne.CNEService 对网络动态控制策略， 可实现LTE-Wi-Fi切换

1. 确认WiFi是否可用

KeyWord: wifi_avail

```
03:57:25.936 ds_mppm_cmd_hdlr.c 04801 ds_mppm_cmd_process_mppm_dsd_event: Old DSD Preferred System Mask is 0x10011 Current DSD Preferred System Mask is 0x11 wifi_avail 1
```

2. 查看激活PDP的RAT （log中没有）

KeyWord: HandleWifiMeasurement

```
03:57:15.690 PDPRATHandlerVoLTE.cpp 00850 HandleWifiMeasurement triggering PDP activate on IWLAN
```

所以重新检查QCN网站的配置文件 发现所有的EFS文件都是为空的

举例：

TargetPath	BuildPath
/sd/rat_acq_order	modem_proc/r

双击后在 QCN网站显示为0KB 需要手动上传 双击 该栏

Brower 导入 对应EFS文件
所有EFS 文件路径在 （ 举例TMO运营商 ）
amss_8976/modem_proc/mcfg/mcfg_gen/scripts/data/efs_files/tmo/

所有EFS 替换完了，重新编tarball 刷入手机

11. 设置 properity 参数 和 Nv Value
如果不识卡 修改NV 00453 0x00
确保 00071 是不是 TMO 确认MBN 有没有激活
Set following properties 获取root权限 进行修改和添加

1. adb shell setprop persist.dbg.wfc_avail_ovr 1
2. adb shell setprop persist.data.iwlan.enable TRUE
3. adb shell setprop persist.radio.calls.on.ims TRUE
4. adb shell setprop persist.radio.vrte_logic 1
5. adb shell setprop persist.radio.domain.ps FALSE
6. adb shell setprop persist.radio.jbims 1
7. adb shell setprop persist.radio.ROTATION_ENABLE 1
8. adb shell setprop persist.rmnet.mux ENABLED
9. adb shell setprop persist.rcs.supported 1
10. adb shell setprop persist.ims.enableADBLogs 1
11. adb shell setprop persist.ims.enableDebugLogs 1
12. adb shell setprop persist.radio.VT_ENABLE 1
13. adb shell setprop persist.radio.VT_HYBRID_ENABLE 1
14. adb shell setprop persist.radio.VT_USE_MDM_TIME 0
15. adb shell setprop persist.dbg.volte_avail_ovr 1
16. adb shell setprop ro.config.videocall.show true
17. adb shell setprop persist.cne.rat.wlan.chip.oem nqc

设置NV值 查看NV值得 代表的参数 可以通过QXDM NVbrower 或者
在http://wukong.tclcom.com/wiki/index.php/Idol4_NV_items 查看

1. NV67218 set to 1 (IMS Enable /nv/item_files/ims/IMS_enable IMS)
2. NV70287: set to 1 （Hybrid Feature to be Enabled/Disabled /nv/item_files/ims/ims_hybrid_enable IMS） nv/item_files/ims/ims_hybrid_enable);=> 必须要将 ims_hybrid_enable设置为1,否则将不会发起IWLAN的IKEV2的鉴权等.
3. NV70315(wlan_offload_config) = 2 （Wlan Offload Config /nv/item_files/data/wlan_config/wlan_offload_config Data）
4. NV67332(Version =3, audio_offload = 0); （ IMS Media Config /nv/item_files/ims/qp_ims_media_config IMS）
5. NV69679: ds_apn_switching = 0x01; （Data APN Switching /nv/item_files/data/dsd/ds_apn_switching Data）
6. NV70316: data_wlan_throttle_timer = 120; （Data Wlan Throttle Timer

/nv/item_files/data/dsd/data_wlan_throttle_timer Data)

7. NV71567: epc_handoff_retry_count = 0;default value = 0); (EPC Handoff Retry Count
/nv/item_files/data/3gpp2/epc_handoff_retry_count Data)

8. NV72536：WIFI OOS Linger timer = 180; (WIFI OOS Linger Timer
/nv/item_files/data/wlan_config/wifi_oos_linger_timer Data)

9. NV72554 = 1000; (Data LTE MAPCON Hystersis Timer /nv/item_files/data/wlan_config/LTE_NULL_hysteresis_timer
Data)

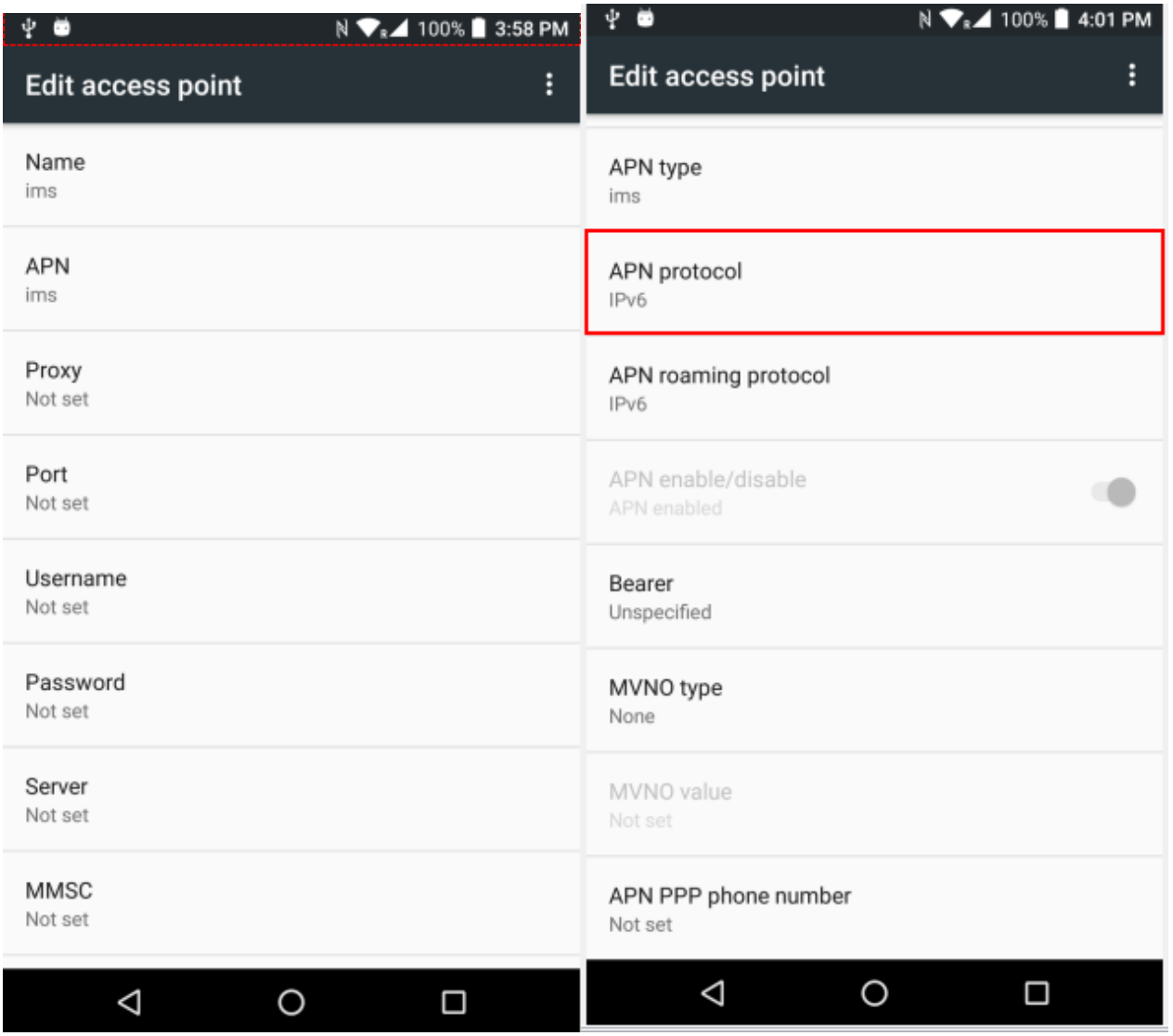
10. NV72555 =2000; (EHRPD NULL Hysteresis Timer/nv/item_files/data/wlan_config/EHRPD_NULL_hysteresis_timer
Data)

11.NV72556=3000; (Data LTE MAPCON Wait Timer/nv/item_files/data/wlan_config/wait_for_LTE_attach_timer
Data)

12. NV73545无法写入和读取，直接把文件放在了/nv/item_files /modem/mmode/wifi_config"下，需要
配置成0 1 0 5 0 0 0 0 0 0 0 (Wifi Config /nv/item_files/modem/mmode/wifi_config MMode)

13. NV73717 设置成1 (IMS Disable Emergency Call Over VoWIFI
/nv/item_files/ims/qp_ims_emerg_wifi_disable IMS)

12. 设置APN 为IMS



13.使用 EFS explore 查看EFS 文件，主要文件如下

1. /data/iwlan_s2b_config.txt

```
epdg_fqdn:ss.epdg.epc.mnc260.mcc310.pub.3gppnetwork.org;
epdg_ipv4_address:208.54.73.77;
```

2. /data/pdn_policy_db.txt
3. /efsprofiles/imshandoverconfig
4. /data/ps_sys_data_configurations.txt //AT Command 设置的
5. /data/andsf.xml //默认profile文件的顺序
6. /data/default_andsf.xml //默认profile文件的顺序

可以直接通过EFS Explore 拖进去 重启 验证 是否注册上VoWi-Fi

14.AT Command 需要超级终端连接手机 手动修改 Profile 配置

```
AT$QCPDPIMSCFGE?
$QCPDPIMSCFGE: 1 , 0 , 0 , 0
$QCPDPIMSCFGE: 2 , 1 , 0 , 0
$QCPDPIMSCFGE: 3 , 0 , 0 , 0
$QCPDPIMSCFGE: 4 , 0 , 0 , 0


OK
at+cgdcont?
+CGDCONT: 1,"IPV4V6","","0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0",0,0,0,0
+CGDCONT: 2,"IPV6","ims","0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0",0,0,0,0
+CGDCONT: 3,"IP","epc.tmobile.com","0.0.0.0",0,0,0,0
+CGDCONT: 4,"IPV4V6","tmus","0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0",0,0,0,0

OK
```

二、TarBall 或者Default MBN

TarBall

1.
gedit amss_8976/modem_proc/mcfg/mcfg_gen/generic/NA/TMO/mcfg_sw_gen_Commercial.xml
2. 将参数添加mcfg_sw_gen_Commercial.xml文件中 如果加的是EFS文件，在buildpath中本地Code中寻找文件是否存在 nv值按照原来的加
3. NV值 在上面已经提过，参照步骤11
4. 改完后最新的MCFG文件 通过 脚本生成 新的MCFG 防止上传到QCN网站没有引入 EFS文件 参照纯手动配置步骤3
5. 通过 脚本 mcfg_extraFile_to_xml.py 生成新的MCFG文件
路径：smb://172.24.116.58/share/swd- 2/telecom/Members/james/BlackBerry
关于高通提供的XML 文件，是需要转换后才能上传到参数系统网站的。
脚本介绍

此脚本用于 将现有 AMSS代码仓库中的MBN XML源文件，转换成可供参数系统使用的形式。（参数版本系统后更新可用）

脚本会搜集XML源文件 中所 有引用到的外部文件，并将外部文件的内容提取出来，写成新的XML文

件。 这个新的XML文件到 时候　可以直接导入到参数系统。

使用方法如下：

脚本存放路径

```
amss_8976/vendor/script/mcfg_extraFile_to_xml.py
```

脚本运行命令：

cd amss_8976

python vendor/script/mcfg_extraFile_to_xml.py mcfgxml 文件路径

编完后，参照我之前手动配置 上传至QCN ，并且配置本地的st.sh 和product_parameter_vesrion.xml文件
最后就是编译Tarball ，由于编tarball 会遍历一遍mk文件，为节约时间 在build/core/main.mk 添加一段脚本

```
# --mindepth=2 makes the prunes not work.
ifneq (,$(filter $(MAKECMDGOALS),tar))
    $(info skip scan all Android.mk files to speed up make tar)
    subdirs:=vendor/tct/source/qcn/
sendif
```

编译tarball 参照我纯手动配置 命令 刷入tarball 说明MBN部分已经配置好了
6.设置properties参数 参照手动配置列表 步骤11
7. 设置APN 为IMS 参照手动配置列表 步骤12

最后应该是可以注册上VoWi-Fi（需要用TMO卡激活）

Default MBN

1.改完
amss_8976/modem_proc/mcfg/mcfg_gen/generic/NA/TMO/mcfg_sw_gen_Commercial.xml 后
2. cd amss_8976/modem_proc/mcfg/build
　执行 perl ./build_mcfgs.pl --force-regenerate --force-rebuild --source-dir=generic/**NA/TMO** --configs=mcfg_sw:Commercial –xml
3.解决了其中的编译问题，如果编译成功可以使用下面的命令来编译modem
 修改编译脚本 将Default MBN 内置到Modem 中 生成 NON-HOLS.bin文件中
gedit amss_8976/linux_build.sh
在 build_modem( )中添加 如下

```
if [[ "$project" == "idol452" && "$operator" == "cricket" ]]; then
        ./build.sh $CPU_CHIP_TYPE.gen.prod bparams=-k MCFG_SW_TYPE=Non_VoLTE_Cricket MCFG_SW_PRODUCT=generic/NA/ATT MCFG_HW_PRODUCT=generic/common/Default OEM_BUILD_VER=$non_version $append
+    elif [[ "$project" == "idol4" && "$operator" == "global" ]]; then
+   ./build.sh $CPU_CHIP_TYPE.gen.prod bparams=-k MCFG_SW_TYPE=Commercial MCFG_SW_PRODUCT=generic/NA/TMO MCFG_HW_PROD+   UCT=generic/common/Default OEM_BUILD_VER=$non_version $append
    else
```

```
6.          ./build.sh $CPU_CHIP_TYPE.gen.prod bparams=-k OEM_BUILD_VER=$non_version $append
```

4. 执行 ./linux_build.sh -m idol4 global 生成 NON-HLOS.bin 替换入N文件
5. 设置properties参数 参照手动配置列表 步骤11
6. 最后设置APN IMS 参照手动配置列表 步骤12

Black Verion

编一个Default MBN的Black Version

1. 从下载Black Version 8976（idol4/ NEON）

2.gedit amss_8976/linux_build.sh

在 build_modem( )中添加 如下

```
1.  if [[ "$project" == "idol452" && "$operator" == "cricket" ]]; then
2.          ./build.sh $CPU_CHIP_TYPE.gen.prod bparams=-k MCFG_SW_TYPE=Non_VoLTE_Cricket MCFG_
    SW_PRODUCT=generic/NA/ATT MCFG_HW_PRODUCT=generic/common/Default OEM_BUILD_VER=$non_versio
    n $append
3.  +    elif [[ "$project" == "idol4" && "$operator" == "global" ]]; then
4.  +   ./build.sh $CPU_CHIP_TYPE.gen.prod bparams=-k MCFG_SW_TYPE=Commercial MCFG_SW_PRODUCT=
    generic/NA/TMO MCFG_HW_PROD+   UCT=generic/common/Default OEM_BUILD_VER=$non_version $appe
    nd
5.      else
6.          ./build.sh $CPU_CHIP_TYPE.gen.prod bparams=-k OEM_BUILD_VER=$non_version $append
```

3. cd amss_8976\modem_proc\mcfg\mcfg_gen\generic\NA\TMO

将修改完的MCFG 文件（配完NV Value）

4. 执行 perl ./build_mcfgs.pl --force-regenerate --force-rebuild --source-dir=generic/**NA/TMO** --configs=mcfg_sw:Commercial –xml

5. 设置 **Android Properties**

device\tct\idol4\system.prop

添加

```
1.  persist.dbg.wfc_avail_ovr=1
2.
3.  persist.data.iwlan.enable=TRUE
4.  persist.radio.calls.on.ims=TRUE
5.  persist.radio.vrte_logic=1
6.  persist.radio.domain.ps=FALSE
7.  persist.radio.jbims=1
8.  persist.radio.ROTATION_ENABLE=1
9.  persist.rmnet.mux=ENABLED
10. persist.rcs.supported=1
11. persist.ims.enableADBLogs=1
12. persist.ims.enableDebugLogs=1
13.
14. # for video call
15. persist.radio.VT_ENABLE=1
```

```
persist.radio.VT_HYBRID_ENABLE=1
persist.radio.VT_USE_MDM_TIME=0
persist.dbg.volte_avail_ovr=1

# show video call button in Dialer
ro.config.videocall.show=true
```

6. APN IMS

gedit custo_wimdata_ng\wcustores\apns-conf.xml

add ims apn

```
  <apn carrier="T-Mobile IMS"
      mcc="310"
      mnc="260"
      apn="ims"
      mmsc=""
      type="ims"
      protocol="IPV6"
      roaming_protocol="IPV6"
  />
```

7. SDM开关

device\tct\common\perso\plf\sys_properties\isdm_sys_properties.plf

set <SDMID>JRD_PRODUCT_PACKAGES_JrdWFCManager</SDMID> value 1

set<SDMID>ro_com_android_dataroaming</SDMID> value true

vendor\tctalone\TctAppPackage\Mms\isdm_MmsRes.plf

set <SDMID>feature_rcs_supported</SDMID> 0x01

8.TMO 运营商的 Overlay 的值

frameworks/base/core/res/res/values-mcc310-mnc260/config.xml

add code

```
    <!-- customization keys for IMS { -->
    <bool name="jrd_ims_feature_right_status_bar_icon">true</bool>
    <bool name="jrd_ims_feature_left_status_bar_icon">false</bool>
    <bool name="jrd_ims_feature_settings_item_ims">true</bool>
    <bool name="jrd_ims_feature_settings_item_volte">true</bool>
    <bool name="jrd_ims_feature_settings_item_vowifi">true</bool>
    <bool name="jrd_ims_feature_function_disable_vowifi_on_roaming">false</bool>
    <bool name="jrd_ims_feature_settings_fragment_vowifi_preference">true</bool>
    <!-- OFF = 0; ON = 1; -->
    <integer name="jrd_ims_config_vowifi_enabled_default">1</integer>
    <!-- WIFI_ONLY = 0; CELLULAR_PREFERRED = 1; WIFI_PREFERRED = 2; -->
    <integer name="jrd_ims_config_vowifi_preference_default">2</integer>
    <!-- default config of ims feature, enabled = 1, disabled = 0 -->
    <integer name="jrd_ims_config_ims_enabled_default">1</integer>
    <!-- } customization keys for IMS -->
```

编译完版本 Download 插上TMO 卡就可以直接注册上VoWi-Fi

LOG 分析

Modem 侧：

MAPCON PDN Policy Manager (MPPM)

1.选择并决定接入技术（RAT），用于建立指定APN的PDN连接。

2. 管理PDN策略数据库，数据库包含了各APN的接入技术（RAT）优先级

WLAN Proxy Interface – WLAN_IFACE (0x8020)

通过AP上的WLAN连接，发送IKEv2信令和建立手机和ePDG的IPSec tunnel连接。

ePDG Interface – IWLAN_S2B_IFACE (0x8021)

Modem上的应用程序使用该接口与ePDG进行数据交互。

EAP/IKEv2

modem侧 用来鉴权和 IPsec的隧道的建立

APN preferred system

```
DS_SYS_APN_PREF_SYS_WWAN – 0    //无线广域网
DS_SYS_APN_PREF_SYS_WLAN – 1   //无线局域网
DS_SYS_APN_PREF_SYS_IWLAN – 2 //工业无线局域网
```

IMS 选择不同的无线接入的方式 ，会直接进行切换选择网络

使用举例：

```
In PDN policy database
PDN_APN_String:IMS; //APN是ims
Override_Type:API;
Supported_RAT_Priority_List:WWAN, IWLAN;  //选择RAT的优先级
```

Register Log

分析材料 QXDM Log，SIP Message

logMask 地址：smb://172.24.116.58/share/swd-2/telecom/Members/james/BlackBerry/VoWiFi/IMS_logmask_latest_ConfigurationFile.dmc

注册流程

1. 确认 Wi-Fi 是否可用

KeyWord：wifi_avail

```
03:57:25.936        ds_mppm_cmd_hdlr.c  04801  ds_mppm_cmd_process_mppm_dsd_event: Old DSD Preferred System Mask is 0x10011 Current DSD Preferred System Mask is 0x11 wifi_avail 1
```

2.查看激活对应APN的PDN连接的RAT

KeyWord：mppm

```
MSG        [05006/01] DS Protocol Services/Medium09:56:27.075     ds_mppm_rat_calc.c  01082  ds_mppm_run_pref_rat_per_pdn_calc_algo () updated last pref_sys_mask for API based APN overrides is 0x1 and current pref RAT is RADIO_IWLAN
```

```
2.  MSG        [05006/01] DS Protocol Services/Medium09:56:27.076              ds_mppm.c  0354
    6  ds_mppm_get_pref_rat_for_apn_string_ex: MPPM calculated  pref_rat RADIO_IWLAN, is_lte_a
    ttach_allowed 0 for apn_string IMS
```

3.发起 S2b/ePDG Interface

KeyWord：Entering

```
1.  // WLAN proxy (0x8020::0) interface is up
2.  09:56:03.004     ds_wlan_proxy_mh_sm.c  00674  DS_WLAN_PROXY_SM: SM id:0 Entering SIO_CONF
    IG state
3.  09:56:03.004     ds_wlan_proxy_mh_sm.c  00678  Entering WLAN_PROXY_SM_STATE_SIO_CONFIG sta
    te from state WLAN_PROXY_SM_STATE_CLOSED
4.  09:56:03.006     ds_wlan_proxy_mh_sm.c  01143  DS_WLAN_PROXY_SM: SM id:0 Entering RM_CONFI
    G state
5.  09:56:03.006     ds_wlan_proxy_mh_sm.c  01147  Entering WLAN_PROXY_SM_STATE_RM_CONFIG stat
    e from state WLAN_PROXY_SM_STATE_SIO_CONFIG
6.  09:56:03.341     ds_wlan_proxy_mh_sm.c  01639  DS_WLAN_PROXY_SM: SM id:0 Entering NET_UP s
    tate
7.  09:56:03.341     ds_wlan_proxy_mh_sm.c  01643  Entering WLAN_PROXY_SM_STATE_NET_UP state f
    rom state WLAN_PROXY_SM_STATE_RM_CONFIG
8.
9.  // IKEv2 tunnel configuration
10. 09:56:16.831     ds_iwlan_s2b_pdn_sm.c  00564  Entering DS_IWLAN_S2B_PDN_SM_TUNNEL_PRE_CON
    FIG_STATE state from state DS_IWLAN_S2B_PDN_SM_CLOSED_STATE
11. 09:56:16.879     ds_iwlan_s2b_pdn_sm.c  00564  Entering DS_IWLAN_S2B_PDN_SM_TUNNEL_CONFIGU
    RING_STATE state from state DS_IWLAN_S2B_PDN_SM_TUNNEL_PRE_CONFIG_STATE
12.
13. // IKEv2 tunnel is up
14. 09:56:26.181     ds_iwlan_s2b_pdn_sm.c  00564  Entering DS_IWLAN_S2B_PDN_SM_SIO_CONFIGURIN
    G_STATE state from state DS_IWLAN_S2B_PDN_SM_TUNNEL_CONFIGURING_STATE
15. 09:56:26.182     ds_iwlan_s2b_pdn_sm.c  00564  Entering DS_IWLAN_S2B_PDN_SM_RM_CONFIGURING
    _STATE state from state DS_IWLAN_S2B_PDN_SM_SIO_CONFIGURING_STATE
16.
17. // PDN connection is up on s2b/epdg interface
18. 09:56:26.582     ds_iwlan_s2b_pdn_sm.c  00564  Entering DS_IWLAN_S2B_PDN_SM_NET_UP_STATE s
    tate from state DS_IWLAN_S2B_PDN_SM_RM_CONFIGURING_STATE
```

4.查看当前使用的IP协议

KeyWord：ds_iwlan_s2b_pdn_sm_enter_net_up_state

```
1.  09:56:26.582     ds_iwlan_s2b_pdn_sm.c  02449  ds_iwlan_s2b_pdn_sm_enter_net_up_state: v4
    iface not in use iface cb ptr 0x0
2.  09:56:26.582     ds_iwlan_s2b_pdn_sm.c  02457  ds_iwlan_s2b_pdn_sm_enter_net_up_state: v6
    iface in use. move iface and phys link to up
```

5. DNS 查询FQDN，获取ePDG的地址

KeyWord：Sending DNS

```
1.  03:57:15.773        ds_iwlan_s2b_epdg_addr_resolver.c  01044  ds_iwlan_s2b_epdg_addr_reslv
    r_send_dns_query: Sending DNS query
```

6. 建立IPsec隧道

KeyWord：ikev2

```
1.  03:57:53.290        ds_iwlan_s2b_ikev2_hdlr.c  03499  ds_iwlan_s2b_ikev2_callback. event:
```

```
1, cause: 0
03:57:53.290        ds_iwlan_s2b_ikev2_hdlr.c  01706  ds_iwlan_s2b_ikev2_copy_sa_info: Tot
al incoming cfg attr 4.
03:57:53.290        ds_iwlan_s2b_ikev2_hdlr.c  01797  ds_iwlan_s2b_ikev2_copy_sa_info: cfg
 index 0. cfg id 8.
03:57:53.290        ds_iwlan_s2b_ikev2_hdlr.c  01798  IPV6 Address 2607:fb90:140d:386:0:6:
ff6b:4e01
```

7.IPsec隧道建立成功

KeyWord：DS_IWLAN_S2B_PDN_SM

```
03:57:53.728        ds_iwlan_s2b_pdn_sm.c  00564  Entering DS_IWLAN_S2B_PDN_SM_NET_UP_STAT
E state from state DS_IWLAN_S2B_PDN_SM_RM_CONFIGURING_STATE
```

8 .获取P-CSCF地址

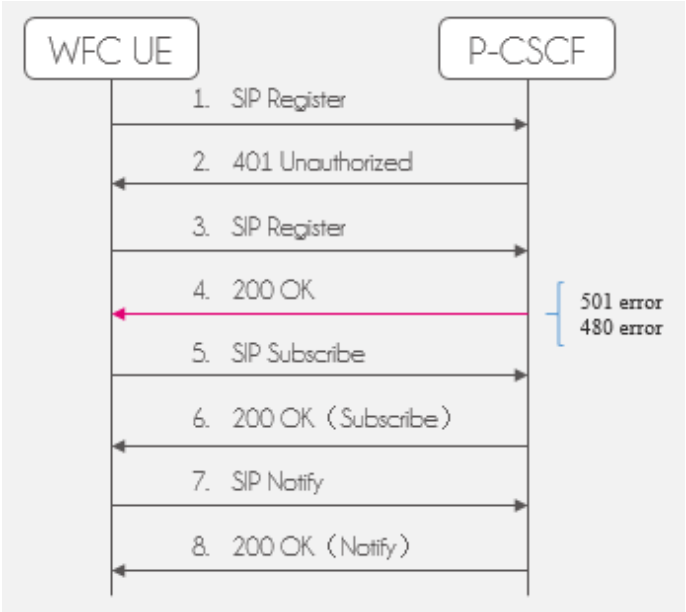
KeyWord：PCSCFListHandler

```
./export_ok.txt:03:57:55.922        PCSCFListHandler.c  00543  SetDplPcscfAddress - Using
IP address: fd00:976a:c206:3::4
./export_ok.txt:03:57:55.922        PCSCFListHandler.cpp  00544  SetDplPcscfAddress - Usin
g IP port: 5060: RegType[0]
```

9.通过IPsec隧道，发送Sip消息

KeyWord：sipConnection.cpp.*EVENT_SIP

```
03:57:56.005         sipConnection.cpp  03002  EVENT_SIP_REG_REQ_SENT: 43 Code = 0 RAT = 6
03:57:56.014         sipConnection.cpp  03763  EVENT_SIP_REQUEST_SEND: REGISTER sip:msg.pc
.t-mobile.com SIP/2.0
03:57:56.767         sipConnection.cpp  02838  EVENT_SIP_REG_RESP_RCVD : 44 Code = 401 RAT
 = 6
03:57:57.185         sipConnection.cpp  03002  EVENT_SIP_REG_REQ_SENT: 43 Code = 0 RAT = 6
03:57:57.186         sipConnection.cpp  03763  EVENT_SIP_REQUEST_SEND: REGISTER sip:msg.pc
.t-mobile.com SIP/2.0
```



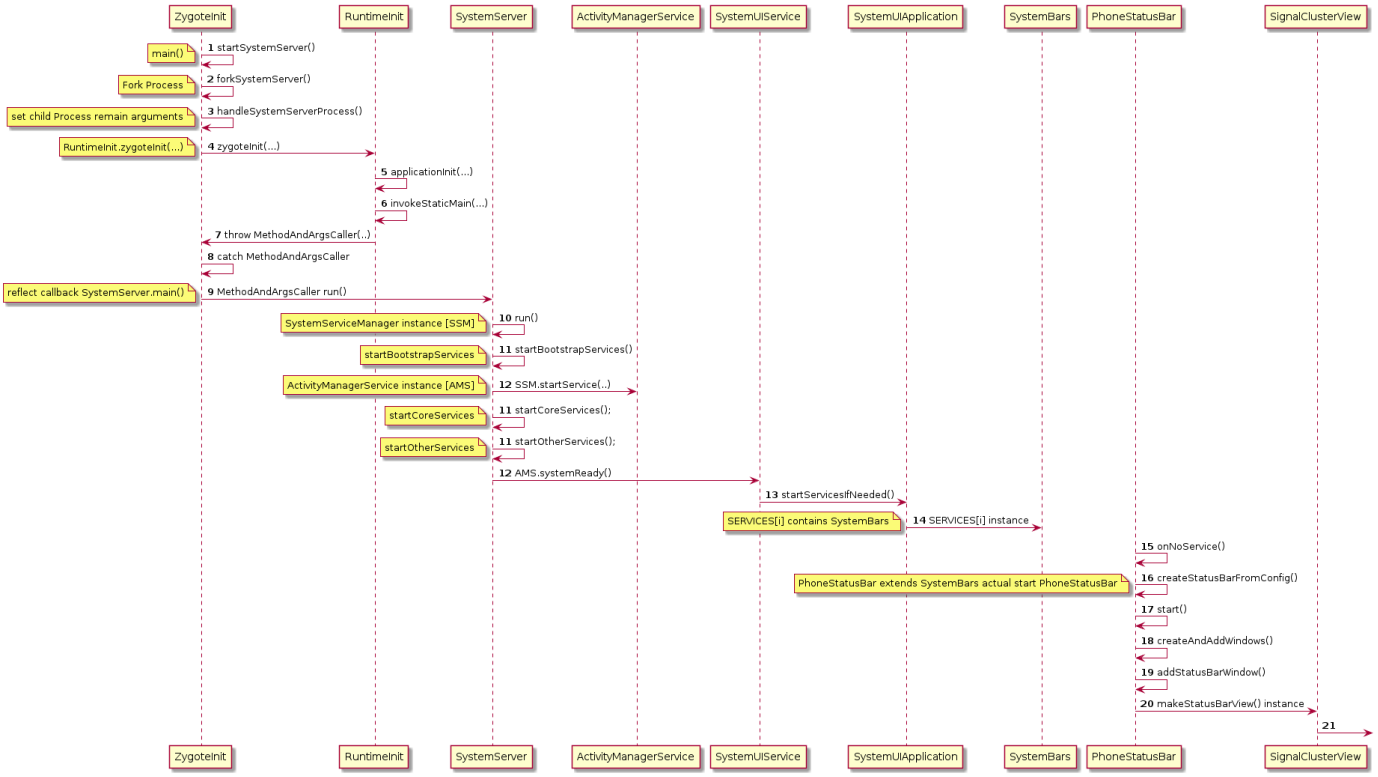
Android Code 架构

1. SystemUI Code
2. JrdWFCManager.apk
3. Frameworks Code

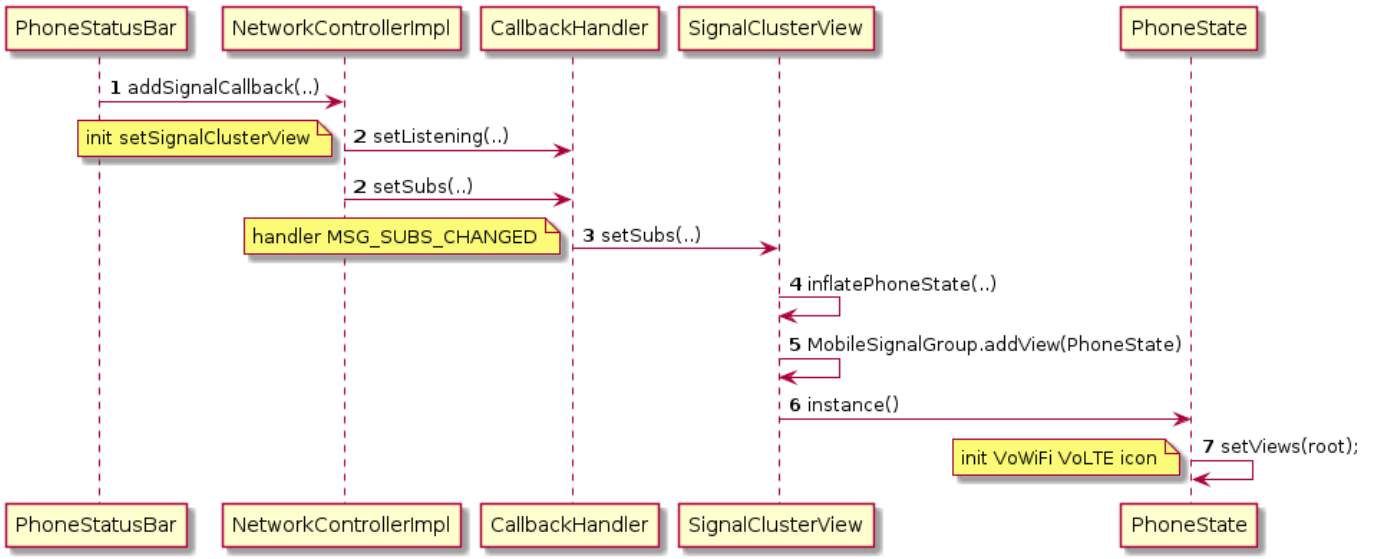
1. SystemUI 显示 VoWiFi 图标 原理

SystemUI 接收到 JrdWFCManager.apk 发送的广播 关于ims state 的状态更新 显示VoWiFi 图标

补充：Android Zygote 到 SystemUI 启动流程



PhoneStatusBar 初始化开始



step 1 PhoneStatusbar

```
1.      // ========================================================================
```

```
    // Constructing the view
    // ======================================================================================
    protected PhoneStatusBarView makeStatusBarView() {
          final SignalClusterView signalCluster =
            (SignalClusterView) mStatusBarView.findViewById(R.id.signal_cluster); //init SignalClusterView
    ...
          mNetworkController = new NetworkControllerImpl(mContext, mHandlerThread.getLooper());
          /* init NetworkControllerImpl 广播接收者,接收信号的改变,包括wifi,手机信号,飞行模式等 接收imsstate "JRD_IMS_REGISTER_STATE_CHANGED"*/
    ....
          mNetworkController.addSignalCallback(signalCluster);
```

step 2 NetworkControllerImpl

```

    public void addSignalCallback(SignalCallback cb) {
        mCallbackHandler.setListening(cb, true);// 2 setListeber SignalClusterView
        mCallbackHandler.setSubs(mCurrentSubscriptions); // 2 setSub
    ....
    }
```

Callbackhandler `setListening`

```
    public void setListening(SignalCallback listener, boolean listening) {
        obtainMessage(MSG_ADD_REMOVE_SIGNAL, listening ? 1 : 0, 0, listener).sendToTarget();
    }
```

handler `MSG_ADD_REMOVE_SIGNAL`

```
            case MSG_ADD_REMOVE_SIGNAL:
                if (msg.arg1 != 0) {
                    mSignalCallbacks.add((SignalCallback) msg.obj);
                } else {
                    mSignalCallbacks.remove((SignalCallback) msg.obj);
                }
```

step 3 Callbackhandler `setSubs`

```
    public void setSubs(List<SubscriptionInfo> subs) {
        obtainMessage(MSG_SUBS_CHANGED, subs).sendToTarget();
    }
```

handler `MSG_SUBS_CHANGED`

```
            case MSG_SUBS_CHANGED:
                for (SignalCallback signalCluster : mSignalCallbacks) {
                    signalCluster.setSubs((List<SubscriptionInfo>) msg.obj);
                }
                break;
```

此处`signalCluster 就是 setListening 传入的SignalClusterView`

step 4 SignalClusterView

```
```

```
    @Override
    public void setSubs(List<SubscriptionInfo> subs) {
        if (hasCorrectSubs(subs)) {
            return;
        }
        // Clear out all old subIds.
        mPhoneStates.clear();
        if (mMobileSignalGroup != null) {
            mMobileSignalGroup.removeAllViews();
        }
        final int n = subs.size();
        for (int i = 0; i < n; i++) {
            inflatePhoneState(subs.get(i).getSubscriptionId());
        }
        if (isAttachedToWindow()) {
            applyIconTint();
        }
    }
```

step 5 SignalClusterView

```
    private PhoneState inflatePhoneState(int subId) {
        PhoneState state = new PhoneState(subId, mContext);
        if (mMobileSignalGroup != null) {
            /* MODIFIED-BEGIN by jian-yang, 2016-11-15,BUG-3425227*/
            /// R2380308 '''[HK][FT][WFC] no VOWIFI icon display when airplane mode on'''
bob.shen 2016-07-04 */
            mMobileSignalGroup.addView(state.mImsType);
            /* MODIFIED-END by jian-yang,BUG-3425227*/
            mMobileSignalGroup.addView(state.mMobileGroup);
        }
        mPhoneStates.add(state);
        return state;
    }
```

step 5 PhoneState

```
        public PhoneState(int subId, Context context) {
            ViewGroup root = (ViewGroup) LayoutInflater.from(context)
                    .inflate(R.layout.mobile_signal_group, null);
            setViews(root);
            mSubId = subId;
        }
```

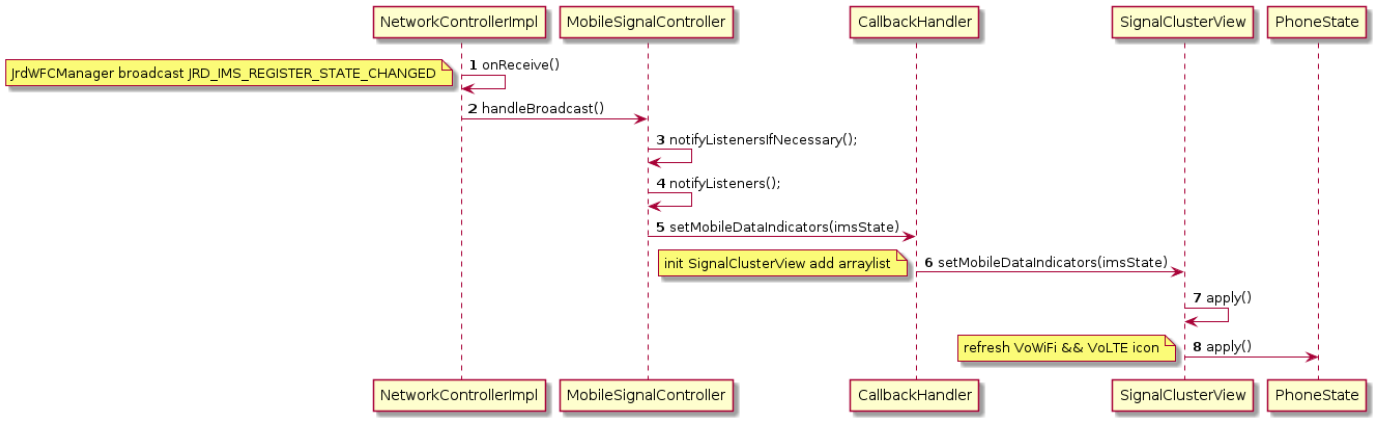
step 6 PhoneState

```
        public void setViews(ViewGroup root) {
...
            mImsType = new ImageView(root.getContext());
            LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(LayoutParams.WRAP
_CONTENT, LayoutParams.WRAP_CONTENT);
            lp.bottomMargin = -28;
            mImsType.setLayoutParams(lp);
```

```
7.              mImsType.setVisibility(View.GONE);
```

只是创建一个View 创建一个位置

VoWiFi Imsstate 状态更新



step 1 NetworkControllerImpl

```
1.      @Override
2.      public void onReceive(Context context, Intent intent) {
3.          ....
4.
5.          } else if (action.equals("android.intent.action.JRD_IMS_REGISTER_STATE_CHANGED"))
    {
6.              for (MobileSignalController controller : mMobileSignalControllers.values()) {
7.                  controller.handleBroadcast(intent);//MobileSignalController
8.              }
9.          }
```

step 2 MobileSignalController

```
1.      public void handleBroadcast(Intent intent) {
2.      ...
3.          else if (action.equals("android.intent.action.JRD_IMS_REGISTER_STATE_CHANGED")) {
4.              mCurrentState.imsRegState = intent.getIntExtra("reg_state", -1);
5.              notifyListenersIfNecessary();
6.          }
```

由于MobileSignalController extend SignalController 最后调用 重写的notifyListeners（ ）

step 3 SignalController

```
1.      public void notifyListenersIfNecessary() {
2.          if (isDirty()) {
3.              saveLastState();
4.              notifyListeners();
5.          }
6.      }
```

step 4 MobileSignalController

```
1.      @Override
2.      public void notifyListeners() {
```

```
3.  ....
4.          mCallbackHandler.setMobileDataIndicators(statusIcon, qsIcon, typeIcon, qsTypeIcon,
5.                  activityIn, activityOut, dataActivityId, mobileActivityId,
6.                  icons.mStackedDataIcon, icons.mStackedVoiceIcon,
7.                  /* MODIFIED-BEGIN by jian-yang, 2016-11-15,BUG-3425227*/
8.                  dataContentDescription, description, icons.mIsWide, mCurrentState.imsRegSt
    ate,
9.                  mSubscriptionInfo.getSubscriptionId());
```

step 5 CallbackHandler

```
1.      @Override
2.      public void setMobileDataIndicators(final IconState statusIcon, final IconState qsIcon
    ,
3.              final int statusType, final int qsType, final boolean activityIn,
4.              final boolean activityOut, final int dataActivityId, final int mobileActivityI
    d,
5.              final int stackedDataIcon, final int stackedVoiceIcon,
6.              final String typeContentDescription, final String description, final boolean i
    sWide,
7.              final int imsState, final int subId) { // MODIFIED by jian-yang, 2016-11-15,BU
    G-3425227
8.          post(new Runnable() {
9.              @Override
10.             public void run() {
11.                 for (SignalCallback signalCluster : mSignalCallbacks) {
12. //                    signalCluster.setMobileType(mIsMobileMMS);
13.                     signalCluster.setMobileDataIndicators(statusIcon, qsIcon, statusType,
    qsType,
14.                             activityIn, activityOut, dataActivityId, mobileActivityId,
15.                             stackedDataIcon, stackedVoiceIcon,
16.                             typeContentDescription, description, isWide, imsState, subId);
     // MODIFIED by jian-yang, 2016-11-15,BUG-3425227
17.                 }
18.             }
19.         });
20.     }
```

step 6 SignalClusterView

```
1.      @Override
2.      public void setMobileDataIndicators(IconState statusIcon, IconState qsIcon, int status
    Type,
3.              int qsType, boolean activityIn, boolean activityOut, int dataActivityId,
4.              int mobileActivityId, int stackedDataId, int stackedVoiceId,
5.              String typeContentDescription, String description, boolean isWide, int imsStat
    e, int subId) {
6.  .....
7.              state.mImsRegState = imsState;
8.  .....
9.              apply();
```

step 7 SignalClusterView

```
1.      private void apply() {
2.  ....
3.          for (PhoneState state : mPhoneStates) {
```

```
            if (state.apply(anyMobileVisible)) {
                if (!anyMobileVisible) {
                    firstMobileTypeId = state.mMobileTypeId;
                    anyMobileVisible = true;
                }
            }
        }
```

step 8 PhoneState

```
        public boolean apply(boolean isSecondaryIcon) {
....
            switch (mImsRegState) {
                case 0: // null
                    break;
                case 1: // VoLTE registered
                    resID = R.drawable.stat_sys_ic_volte;
                    break;
                case 2: // VoWiFi registered
                    resID = R.drawable.stat_sys_ic_vowifi;
                    break;
            }
```

2. JrdWFCManager.apk Code

Code Structure of JrdWFCManager

1. FeatureProcessHandler
2. WFCStateProxy
3. FeatureUiHandler

JrdWFCManager Application 分析

一般App 启动的时候就会出现一个启动界面Activity ，需要在AndroidManifest.xml 注册Activity 添加如下

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

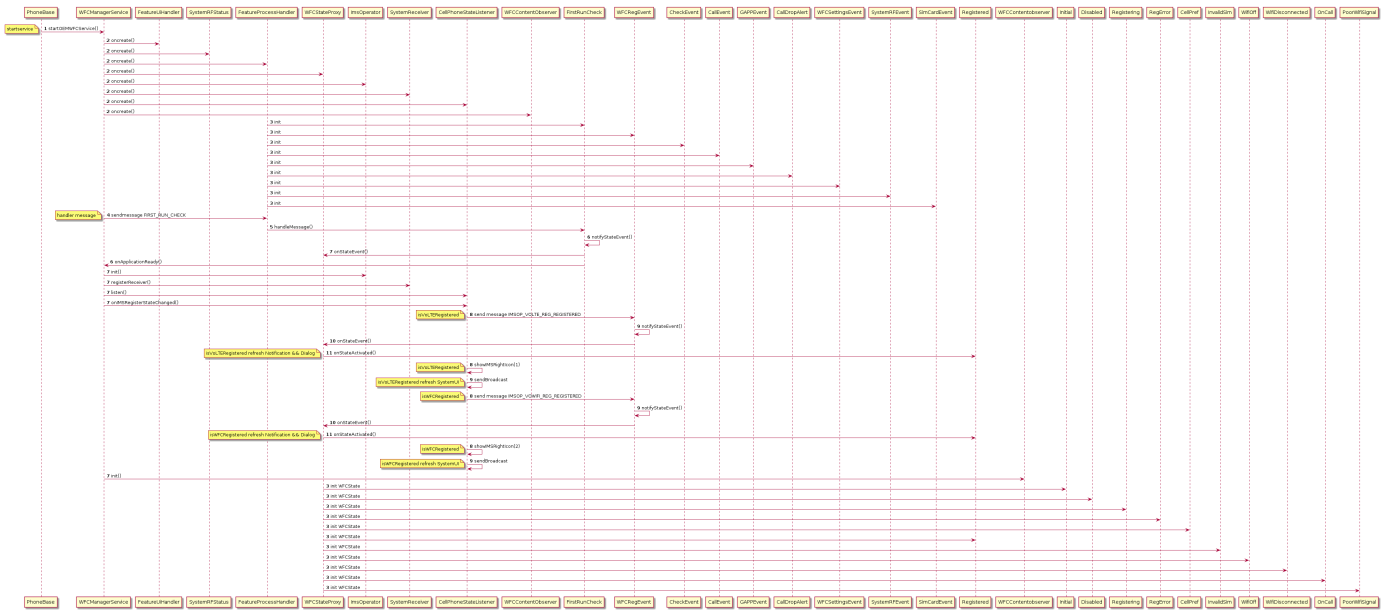
但是JrdWFCManager没有配置，属于后台进程 主要用来获取数据

```
    <application
        android:name=".JrdWFCManager"
        android:label="@string/wificalling_label"
        android:persistent="true" //开机通过AMS启动，优先级最高，如果进程被杀死会自己起来
```

```

ActivityManagerService
    public void systemReady(final Runnable goingCallback) {
....
              try {
                    List apps = AppGlobals.getPackageManager().
                        getPersistentApplications(STOCK_PM_FLAGS);
                    if (apps != null) {
                        int N = apps.size();
                        int i;
                        for (i=0; i<N; i++) {
                            ApplicationInfo info
                                = (ApplicationInfo)apps.get(i);
                            if (info != null
                                    && validNewProc(info.packageName, UserHandle.getUserId
(info.uid))
                                    && !info.packageName.equals("android")) {
                                addAppLocked(info, false, null /* ABI override */);
                            }
                        }
                    }
                } catch (RemoteException ex) {...}
```

需要启动service获取数据

JrdWFCManager Init



step 1 PhoneBase

```
    @IMS.JrdFeature({IMS.FR_TASK_949479})
    private void startOEMWFCService(Context context) {
```

```
        Intent intent = new Intent();
        intent.setAction("com.jrdcom.jrdwfcmanager.service.JrdWFCManagerService");
        intent.setPackage("com.jrdcom.jrdwfcmanager");
        context.startService(intent);
    }
```

由于 JrdWFCManager AndroidManifest 配置了action

```
        <service
            android:name=".service.WFCManagerService"
            android:exported="true">
            <intent-filter>
                <action android:name="com.jrdcom.jrdwfcmanager.service.JrdWFCManagerServic
e"/>
            </intent-filter>
        </service>
```

所以会启动WFCManagerService

Step 2 WFCManagerService

```
    @Override
    public void onCreate() {
        Logger.logd("onCreate");
        mFeatureUiHandler = new FeatureUiHandler(this);
        mSystemRFStatus = new SystemRFStatus();

        HandlerThread handlerThread = new HandlerThread(THREAD_NAME_FEATURE_HANDLER);
        handlerThread.start();
        mProcessHandler = new FeatureProcessHandler(handlerThread.getLooper(), this);
        mWFCStateProxy = new WFCStateProxy(this);

        mImsOperator = new ImsOperator(this);
        mSystemReceiver = new SystemReceiver(this);
        mCellPhoneStateListener = new CellPhoneStateListener(this);
        mWFCContentobserver = new WFCContentObserver(this);

        mProcessHandler.sendMessage(IMPL_MSG.FIRST_RUN_CHECK);
    }
```

Step 3 FeatureProcessHandler (跟主线流程)

```
    public FeatureProcessHandler(Looper looper, IComponentHolder holder) {
        super(looper);
        mHolder = holder;

        mHandlerImpl = new FirstRunCheck(holder);
        mHandlerImpl
                .setSuccessor(new WFCRegEvent(holder))
                .setSuccessor(new CheckEvent(holder))
                .setSuccessor(new CallEvent(holder))
                .setSuccessor(new GAPPEvent(holder))
                .setSuccessor(new CallDropAlert(holder))
                .setSuccessor(new WFCSettingsEvent(holder))
                .setSuccessor(new SystemRFEvent(holder))
                .setSuccessor(new SimCardEvent(holder));
    }
```

Step 4 WFCManagerService

```
mProcessHandler.sendMessage(IMPL_MSG.FIRST_RUN_CHECK);
```

Step 5 FirstRunCheck

```
    @Override
    protected boolean handleMessage(IMPL_MSG what, Message msg) {
....

        switch (what) {
            case FIRST_RUN_CHECK:
                if (!JrdIMSSettings.getBoolean(context, CONTENT_DEFAULT_PROFILE_SET, false
)) {
                    // set wfc default value to QMI
                    mHolder.getImsOperator().setWifiCallingPreference(IMPL_MSG.__FIRST_RUN
_CHECK, defEnabledVal, defPreference);

                    mCheckActionType = SET_DEFAULT_VALUE;
                } else {
                    mHolder.getProcessHandler().sendMessage(IMPL_MSG.__FIRST_RUN_CHECK);
                }
                break;
            case __FIRST_RUN_CHECK:
                switch (mCheckActionType) {
                    case SET_DEFAULT_VALUE:
                        JrdIMSSettings.put(context, CONTENT_IS_WFC_REGISTERED, false);
                        JrdIMSSettings.put(context, CONTENT_DEFAULT_PROFILE_SET, true);
                        break;
                }

....
                notifyStateEvent(STATE_EVENT.INITIAL);
                mHolder.getFeatureUiHandler().post(new Runnable() {
                    @Override
                    public void run() {
                        mHolder.onApplicationReady();
                    }
                });
                break;
        }
        return true;
    }
```

Step 6 FirstRunCheck _FeatureImplementor

```
    protected final void notifyStateEvent(STATE_EVENT event, ImsReasonInfo imsReasonInfo)
{
        _State.EventData data = new _State.EventData();
        data.event = event;
        data.imsReasonInfo = imsReasonInfo;
        mHolder.getWFCStateProxy().onStateEvent(data);
    }
```

Step 7 WFCStateProxy init state

```
    public void onStateEvent(EventData eData) {
```

```
...
        switch (eData.event) {
            case INITIAL:
                nextState = obtainState(Initial.class);
                mPreState = mCurrentState = nextState;
                break;
            case LOCALE_CHANGED:nextState = onSystemLocaleChanged(eData);break;
            case CALL_DROP_ALERT: nextState = onCallDropAlert(eData);break;
            case ON_CALL:nextState = onOnCall(eData); break;
            case OFF_CALL:nextState = onOffCall(eData);break;
            case WIFI_ON:nextState = onWifiOn(eData);break;
            case WIFI_OFF:nextState = onWifiOff(eData);break;
            case WIFI_CONNECTED:nextState = onWifiConnected(eData);break;
            case WIFI_DISCONNECTED:nextState = onWifiDisconnected(eData);break;
            case POOR_WIFI_SIGNAL:nextState = onPoorWifiSignal(eData);break;
            case AIRPLANE_ON:nextState = onAirplaneOn(eData);break;
            case AIRPLANE_OFF:nextState = onAirplaneOff(eData);break;
            case AUTO_CHECK:
            case REGISTERING:nextState = onRegistering(eData);break;
            case REGISTERED:nextState = onRegistered(eData);break;
            case REGERROR:nextState = onRegerror(eData);break;
            case INVALID_SIM:nextState = onInvalidSim(eData);break;
            case ENABLED:nextState = onEnabled(eData);break;
            case DISABLED:nextState = onDisabled(eData);break;
            case WIFI_PREF:nextState = onWifiPref(eData);break;
            case WIFI_ONLY:nextState = onWifiOnly(eData);break;
            case CELL_PREF:nextState = onCellPref(eData);break;
            case NO_CELL_DIAL_TIP:
            case NO_CELL_MSG_TIP:
            case NO_CELL_DIALER_ENTERED_NOTIFY:
            case NO_CELL_MMS_ENTERED_NOTIFY:onNoCellEvents(eData);break;
            default:nextState = mCurrentState;
        }
            mPreState = mCurrentState;
            mCurrentState = nextState;

            eData.eventHandleState = mPreState;
            mPreState.onStateDeactivated(eData);
            mCurrentState.onStateActivated(eData);
        } else {
            eData.eventHandleState = mCurrentState;
            mCurrentState.onStateActivated(eData);
        }


```

Step 6 WFCStateProxy

```
                mHolder.getFeatureUiHandler().post(new Runnable() {
                    @Override
                    public void run() {
                        mHolder.onApplicationReady();
                    }
                });
```

Step 6 WFCManagerService

```
    // IComponentHolder {
```

```
    @Override
    public void onApplicationReady() {
        Logger.logd("onApplicationReady");
        mImsOperator.init();
        mSystemReceiver.registerReceiver();
        mCellPhoneStateListener.listen();
        mCellPhoneStateListener.onIMSRegisterStateChanged();
        mWFCContentobserver.init();
    }
```

Step 7 WFCManagerService init ImsOperator

```
    @Override
    public void init() {
    }
```

Step 7 WFCManagerService init SystemReceiver

```
    public void registerReceiver() {
        if (!mIsRegistered) {
            IntentFilter intentfilter = new IntentFilter();
            intentfilter.addAction(WifiManager.WIFI_STATE_CHANGED_ACTION);
            intentfilter.addAction(WifiManager.NETWORK_STATE_CHANGED_ACTION);
            intentfilter.addAction(WifiManager.RSSI_CHANGED_ACTION);
            intentfilter.addAction(ConnectivityManager.CONNECTIVITY_ACTION);
            intentfilter.addAction(TelephonyIntents.ACTION_ANY_DATA_CONNECTION_STATE_CHANGED);
            intentfilter.addAction(TelephonyIntents.ACTION_SIM_STATE_CHANGED);
            intentfilter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
            intentfilter.addAction(Intent.ACTION_LOCALE_CHANGED);
            intentfilter.addAction(Intent.ACTION_NEW_OUTGOING_CALL);

            intentfilter.addAction(ImsManager.ACTION_IMS_REGISTRATION_ERROR);
            intentfilter.addAction(TEST_ACTION);
            mHolder.getContext().registerReceiver(this, intentfilter);
            mIsRegistered = true;
        }
    }
```

Step 7 WFCManagerService init CellPhoneStateListener

```
    public void listen() {
        Context context = mHolder.getContext();
        mProcessHandler = mHolder.getProcessHandler();
        TelephonyManager tm = (TelephonyManager) context.getSystemService(Context.TELEPHONY_SERVICE);
        tm.listen(this, PhoneStateListener.LISTEN_SERVICE_STATE |
                PhoneStateListener.LISTEN_PRECISE_CALL_STATE |
                PhoneStateListener.LISTEN_VOLTE_STATE |
                PhoneStateListener.LISTEN_SIGNAL_STRENGTHS |
                PhoneStateListener.LISTEN_IMS_REGISTER_STATE_CHANGE |
                PhoneStateListener.LISTEN_IMS_CALL_HANDOVER);
    }
```

Step 7 WFCManagerService init CellPhoneStateListener onIMSRegisterStateChanged()初始化刷新状态

```
    public void onIMSRegisterStateChanged() {
```

```
        FeatureProcessHandler.WFCStatusData wfcData = new FeatureProcessHandler.WFCStatusData();

        boolean isVoLTERegistered = SystemRFStatus.isVoLTERegistered();
        if (isVoLTERegistered) {
            mHolder.getProcessHandler().sendMessage(WFCUtils.IMPL_MSG.IMSOP_VOLTE_REG_REGISTERED, wfcData);
            showIMSRightIcon(1);
            return;
        }

        boolean isWFCRegistered = SystemRFStatus.isWFCRegistered();
        if (isWFCRegistered) {
            mHolder.getProcessHandler().sendMessage(WFCUtils.IMPL_MSG.IMSOP_VOWIFI_REG_REGISTERED, wfcData);
            showIMSRightIcon(2);
            return;
        }

        mHolder.getProcessHandler().sendMessage(WFCUtils.IMPL_MSG.IMSOP_REG_REGISTERING, wfcData);
        showIMSRightIcon(0);
    }
```

Step 7 WFCManagerService init WFCContentObserver

```
    public void init() {
        ContentResolver cr = mHolder.getContext().getContentResolver();
        cr.registerContentObserver(Uri.withAppendedPath(JrdIMSSettings.CONTENT_URI,
                WFCUtils.CONTENT_NOTIFY_LOW_SIGNAL_SEND_MSG), true, this);
        cr.registerContentObserver(Uri.withAppendedPath(JrdIMSSettings.CONTENT_URI,
                WFCUtils.CONTENT_NOTIFY_DIALER_ENTERED), true, this);
        cr.registerContentObserver(Uri.withAppendedPath(JrdIMSSettings.CONTENT_URI,
                WFCUtils.CONTENT_NOTIFY_MMS_ENTERED), true, this);

        cr.registerContentObserver(URI_WFC_IMS_ENABLED, true, this);
        cr.registerContentObserver(URI_WFC_IMS_MODE, true, this);
    }
```

Step 8 WFCRegEvent 举例 VoWiFi
IMSOP_VOWIFI_REG_REGISTERED

```
    @Override
    protected boolean handleMessage(IMPL_MSG what, Message msg) {
            case IMSOP_VOWIFI_REG_REGISTERED:
                if (rfStatus.isMissing911Address()) {
                    rfStatus.setIsMissing911Address(false);
                    rfStatus.set911PopupDisabled(false);
                }
                notifyStateEvent(STATE_EVENT.REGISTERED);
                break;
```

Step 9 WFCRegEvent _FeatureImplementor

```
    protected final void notifyStateEvent(STATE_EVENT event, ImsReasonInfo imsReasonInfo)
{
```

```
        _State.EventData data = new _State.EventData();
        data.event = event;
        data.imsReasonInfo = imsReasonInfo;
        mHolder.getWFCStateProxy().onStateEvent(data);
    }
```

Step 10 WFCStateProxy

```
    public void onStateEvent(EventData eData) {
            case REGISTERED:
                nextState = onRegistered(eData);
                break;
```

Step 11 Registered

```
    @Override
    public void onStateActivated(EventData eData) {
      //更新通知和Dialog
            uiData.notification = NotificationManager.getNotificationPack(NOTIFICATION_CAL
L_END);
            if (SystemRFStatus.isWFCRegistered()) {
                what |= UI_HANDLER_BIT_POPUP;
                uiData.dialog = DialogManagerActivity.getDialogPack(WFCUtils.DIALOG_FIRST_
WFC);
            }
    }
```

Step 8 CellPhoneStateListener

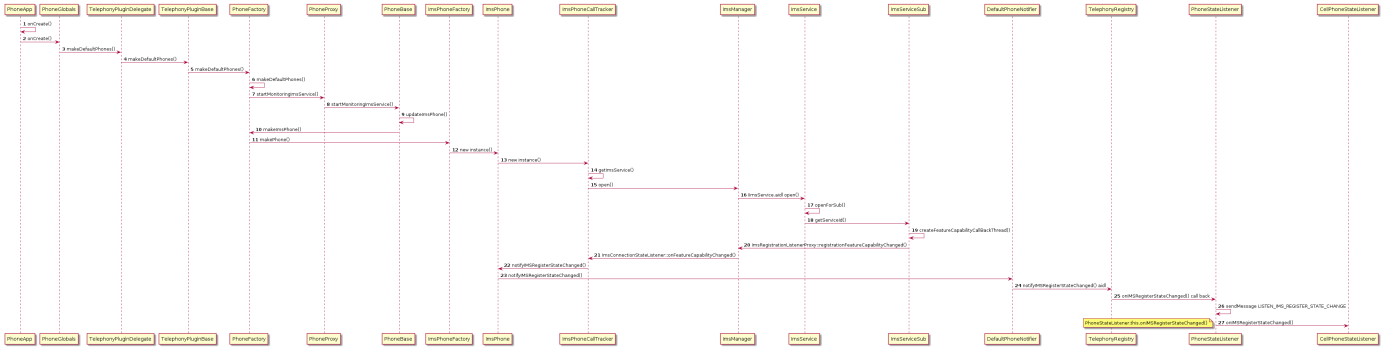
```
    private void showIMSRightIcon(int imsRegState) {
        if (!JrdIMSSettings.isFeatureOn(mHolder.getContext(), com.android.internal.R.bool.
jrd_ims_feature_right_status_bar_icon)) {
            Logger.logd("jrd_ims_feature_settings_fragment_vowifi_preference is false, ski
p show icon.");
            imsRegState = 0;
        }

        Logger.logd("imsRegState: " + imsRegState);
        Intent i = new Intent("android.intent.action.JRD_IMS_REGISTER_STATE_CHANGED");
        i.putExtra("reg_state", imsRegState);
        JrdWFCManager.getContext().sendBroadcast(i);
    }
```

Step 9 sendBroadcast Update SystemUI

Frameworks Code

JrdWFCManager refresh

PhoneApp init



Step 21 ImsPhoneCallTracker

```
1.          @Override
2.          @IMS.JrdFeature({IMS.FR_TASK_949479})
3.          public void onFeatureCapabilityChanged(int serviceClass,
4.                  int[] enabledFeatures, int[] disabledFeatures) {
5.  .....
6.                  // Check enabledFeatures to determine capabilities. We ignore disabledFeat
    ures.
7.                  for (int  i = ImsConfig.FeatureConstants.FEATURE_TYPE_VOICE_OVER_LTE;
8.                          i <= ImsConfig.FeatureConstants.FEATURE_TYPE_UT_OVER_WIFI &&
9.                          i < enabledFeatures.length; i++) {
10.                     if (enabledFeatures[i] == i) {
11.                         // If the feature is set to its own integer value it is enabled.
12.                         if (DBG) log("onFeatureCapabilityChanged(" + i + ", " +
13.                                 mImsFeatureStrings[i] + "): value=true");
14.                         mImsFeatureEnabled[i] = true;
15.                     } else if (enabledFeatures[i]
16.                             == ImsConfig.FeatureConstants.FEATURE_TYPE_UNKNOWN) {
17.                         // FEATURE_TYPE_UNKNOWN indicates that a feature is disabled.
18.                         if (DBG) log("onFeatureCapabilityChanged(" + i + ", " +
19.                                 mImsFeatureStrings[i] + "): value=false");
20.                         mImsFeatureEnabled[i] = false
21. ....
22.                     for (ImsPhoneConnection connection : mConnections) {
23.                         connection.updateWifiState();
24.                     }
25.                     mPhone.onFeatureCapabilityChanged();
26.                 }
27.
28.                 if (tmpIsVideoCallEnabled != isVideoCallEnabled()) {
29.                     mPhone.notifyForVideoCapabilityChanged(isVideoCallEnabled());
30.                 }
31.
32.                 /// T949479 '''WIFI Calling''' bob.shen 2015-11-19
33.                 mPhone.notifyIMSRegisterStateChanged();
34.             }
35.         }
```

Step 22 ImsPhone ImsPhoneBase

```
1.      /* { T949479 '''WIFI Calling''' bob.shen 2015-11-19 */
2.      @IMS.JrdFeature({IMS.FR_TASK_949479})
3.      public void notifyIMSRegisterStateChanged() {
4.          mNotifier.notifyIMSRegisterStateChanged();
5.      }
```

Step 23 DefaultPhoneNotifier

```
    @Override
    @IMS.JrdFeature({IMS.FR_TASK_949479})
    public void notifyIMSRegisterStateChanged() {
        try {
            if (mRegistry != null) {
                mRegistry.notifyIMSRegisterStateChanged();
            }
        } catch (RemoteException ex) {
            ex.printStackTrace();
        }
    }
```

Step 24 TelephonyRegistry aidl

```
    /**
     * added for wifi calling
     * @hide
     */
    @Override
    @IMS.JrdFeature({IMS.FR_TASK_949479}) // MODIFIED by jian-yang, 2016-11-15,BUG-3425227
    public void notifyIMSRegisterStateChanged() throws RemoteException {
        synchronized (mRecords) {
            for (Record r : mRecords) {
                if (VDBG) {
                    log("notifyIMSRegisterStateChanged:  r=" + r);
                }
                if ((r.matchPhoneStateListenerEvent(PhoneStateListener.LISTEN_IMS_REGISTER
_STATE_CHANGE))) {
                    try {
                        r.callback.onIMSRegisterStateChanged();
                    } catch (RemoteException ex) {
                        mRemoveList.add(r.binder);
                    }
                }
            }
            handleRemoveListLocked();
        }
    }
```

Step 25 PhoneStateListener callback

```
        @Override
        @IMS.JrdFeature({IMS.FR_TASK_949479})
        public void onIMSRegisterStateChanged() throws RemoteException {
            Message.obtain(mHandler, LISTEN_IMS_REGISTER_STATE_CHANGE, 0, 0, null).sendToT
arget();
        }
```

Step 26 PhoneStateListener sendMessage LISTEN_IMS_REGISTER_STATE_CHANGE

```
                    case LISTEN_IMS_REGISTER_STATE_CHANGE:
                        PhoneStateListener.this.onIMSRegisterStateChanged();
                        break;
```

Step 27 CellPhoneStateListener

```
1.
2.      public void onIMSRegisterStateChanged() {
3.          FeatureProcessHandler.WFCStatusData wfcData = new FeatureProcessHandler.WFCStatusD
    ata();
4.
5.          boolean isVoLTERegistered = SystemRFStatus.isVoLTERegistered();
6.          if (isVoLTERegistered) {
7.              mHolder.getProcessHandler().sendMessage(WFCUtils.IMPL_MSG.IMSOP_VOLTE_REG_REGI
    STERED, wfcData);
8.              showIMSRightIcon(1);
9.              return;
10.         }
11.
12.         boolean isWFCRegistered = SystemRFStatus.isWFCRegistered();
13.         if (isWFCRegistered) {
14.             mHolder.getProcessHandler().sendMessage(WFCUtils.IMPL_MSG.IMSOP_VOWIFI_REG_REG
    ISTERED, wfcData);
15.             showIMSRightIcon(2);
16.             return;
17.         }
18.
19.         mHolder.getProcessHandler().sendMessage(WFCUtils.IMPL_MSG.IMSOP_REG_REGISTERING, w
    fcData);
20.         showIMSRightIcon(0);
21.     }
```