# 第一章 wpa\_supplicant和hostapd的开发人员文档

源代码中此文档和注释的目的是为其他开发人员提供足够的信息，以了解如何实现wpa\_supplicant和hostapd，如何修改它们，如何支持新驱动程序以及如何移植源代码 到其他操作系统。 如果缺少任何信息，请随时联系Jouni Malinen <j@w1.fi>获取更多信息。 作为补丁文件的贡献在同一地址也非常受欢迎。 请注意，本软件根据用户的选择以双许可证，GPLv2或BSD许可。 对wpa\_supplicant和hostapd的所有贡献都应使用兼容的许可条款。

可以从项目主页http://w1.fi/wpa\_supplicant/获取对组合的wpa\_supplicant和hostapd Git存储库的源代码和只读访问权限。 此开发人员的文档也可以从http://w1.fi/wpa\_supplicant/wpa\_-supplicant-devel.pdf以PDF文件的形式获得。

## 1.1 wpa\_supplicant

wpa\_supplicant是适用于Linux，BSD和Windows的WPA请求者，支持WPA和WPA2（IEEE 802.11i / RSN）。 请求者是客户端站中使用的IEEE 802.1X / WPA组件。 它实现与WPA Authenticator的密钥协商，并且可以选择控制wlan驱动程序的漫游和IEEE 802.11身份验证/关联。

wpa\_supplicant的设计目标是为所有WPA功能使用硬件，驱动程序和独立于操作系统的可移植C代码。 源代码分为单独的C文件，如代码结构页面所示。 所有硬件/驱动程序特定功能都在单独的文件中，这些文件实现了定义良好的驱API， 有关移植到不同目标板和操作系统的信息，请参见移植页面。

EAPOL（IEEE 802.1X）状态机实现为与EAP对等实现交互的单独模块。 除了针对正常生产使用的程序外，wpa\_supplicant源代码树还包含许多测试和开发工具，可以更轻松地测试程序，而无需使用无线卡设置完整的测试设置。 这些工具还可用于实现自动测试套件。

wpa\_supplicant实现一个控制接口，外部程序可以使用该接口来控制wpa\_supplicant守护程序的操作并获取状态信息和事件通知。 有一个小型C库，提供辅助函数以方便控制界面的使用。 该库也可以与C ++一起使用。

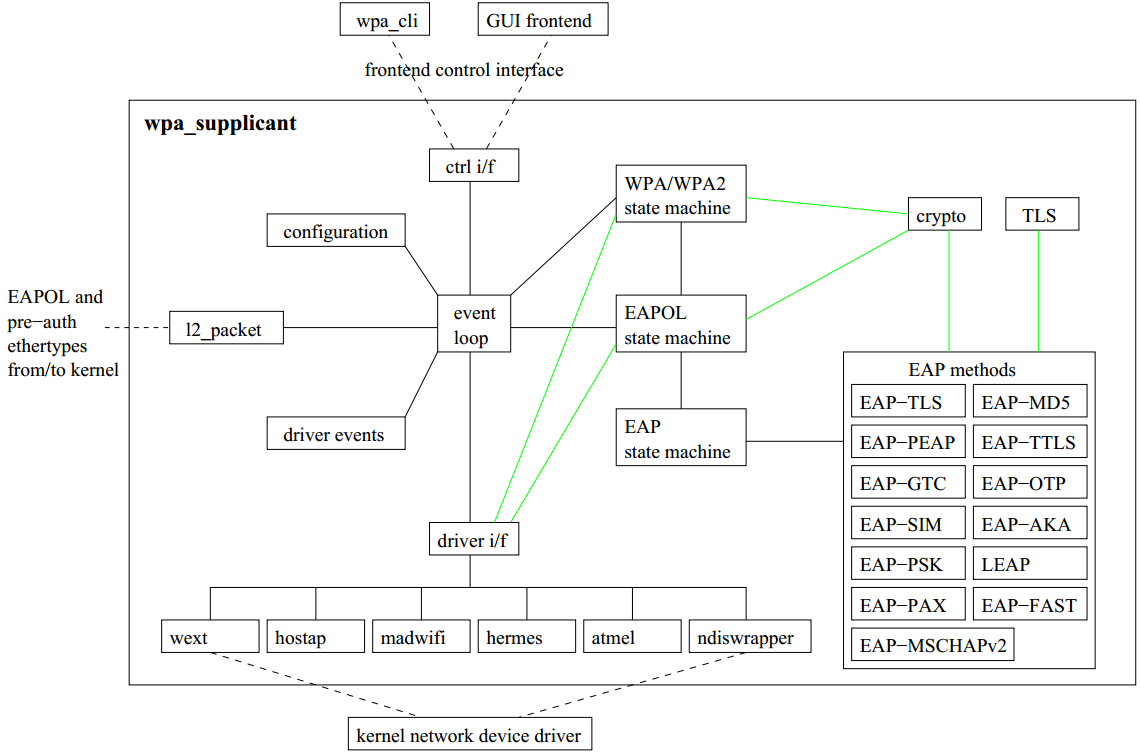


图1.1：wpa\_supplicant模块

## 1.2 hostapd

hostapd包括IEEE 802.11接入点管理（身份验证/关联），IEEE 802.1X / WPA / WPA2身份验证器，EAP服务器和RADIUS身份验证服务器功能。 它可以使用各种配置选项构建，例如，独立的AP管理解决方案或支持多种EAP方法的RADIUS认证服务器。

hostapd的设计目标是为所有WPA功能使用硬件，驱动程序和独立于操作系统的可移植C代码。 源代码分为单独的C文件，如代码结构页面所示。 所有特定于硬件/驱动程序的功能都在单独的文件中，这些文件实现了定义良好的驱 有关移植到不同目标板和操作系统的信息，请参见移植页面。

EAPOL（IEEE 802.1X）状态机实现为与EAP服务器实现交互的单独模块。 同样，RADIUS身份验证服务器位于其自己的独立模块中。 IEEE 802.1X和RADIUS身份验证服务器都可以使用EAP服务器功能。

hostapd实现了一个控制接口，外部程序可以使用该接口来控制hostapdt守护程序的操作并获取状态信息和事件通知。 有一个小型C库，提供辅助函数以方便控制界面的使用。 该库也可以与C ++一起使用。

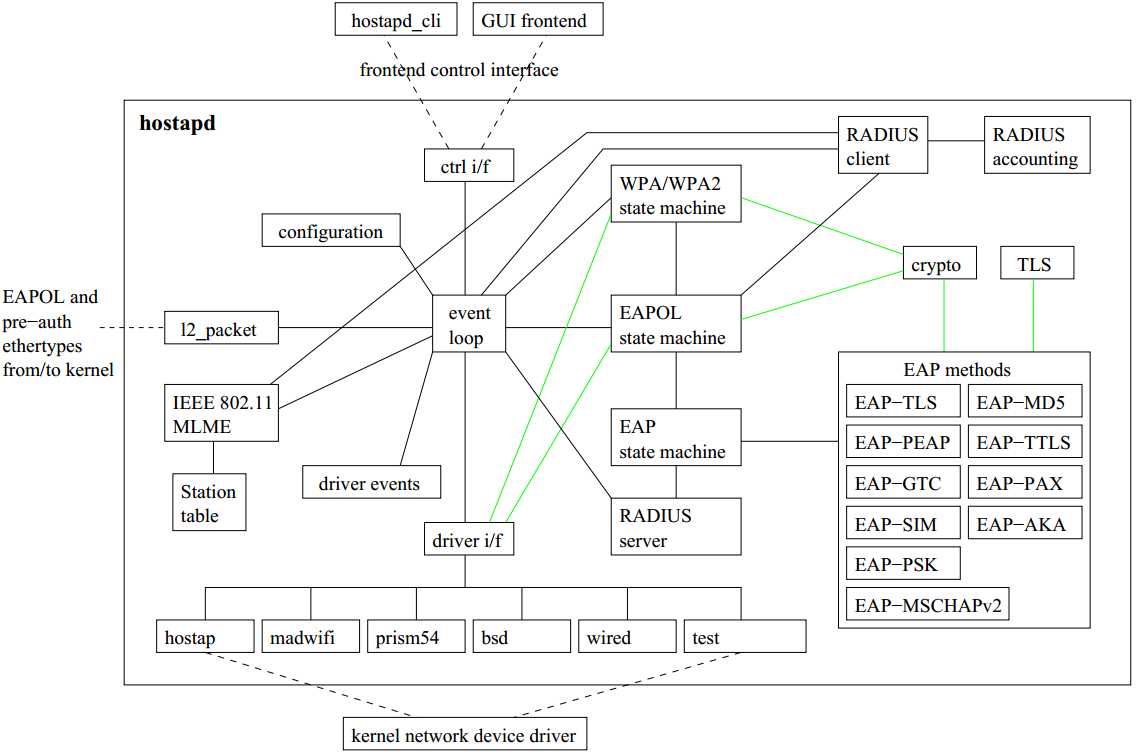


图1.2：hostapd模块

# 第二章 源代码的结构

wpa\_supplicant实现分为多个独立模块。 核心代码包括用于控制网络选择，关联和配置的功能。 独立模块包括WPA代码（密钥握手，PMKSA缓存，预身份验证），EAPOL状态机和EAP状态机和方法。 此外，通用辅助函数还有许多单独的文件。

WPA和EAPOL / EAP状态机都可以在除wpa\_supplicant之外的其他程序中单独使用。 例如，包含的测试程序eapol\_test和preauth\_test正在使用这些模块。

驱动程序接口API在driver.h中定义，所有与硬件/驱动程序相关的功能在driver \_ \* .c中实现。

## 2.1 wpa\_supplicant核心功能

wpa\_supplicant.c 程序初始化，主控制循环

main.c main（）用于类UNIX操作系统和MinGW（Windows）; 这使用命令行参数来配置wpa\_supplicant。

events.c 驱动程序事件处理 wpa\_supplicant\_event（）和相关函数。

wpa\_supplicant\_i.h wpa\_supplicant核心的内部定义，不应包含在独立模块中。

## 2.2通用辅助函数

wpa\_supplicant使用通用辅助函数，其中一些函数与hostapd共享。 目前使用以下C文件：

eloop.c和eloop.h事件循环（具有可注册超时的select（）循环，套接字读取回调和信号回调）

common.c和common.h常用的辅助函数。

defs.h由多个文件共享的定义。

l2\_packet.h，l2\_packet\_linux.c和l2\_packet\_pcap.c第2层（链接）访问包装器（包括本机Linux实现和libdnet / libpcap的包装器）。 移植到libdnet / libpcap不支持的新操作系统时，可能需要添加新的l2\_packet实现。 Makefile可用于选择包含哪个l2\_packet实现。 l2\_packet\_linux.c使用Linux数据包套接字，l2\_packet\_pcap.c使用libpcap和libdnet有一个更便携的版本。

pcsc\_funcs.c和pcsc\_funcs.h用于PC / SC lite SIM和智能卡读卡器的包装器

priv\_netlink.h来自Linux内核头文件的私有版netlink定义; 一旦合适的版本变得普遍可用，这可以用C库头文件替换

version.h版本号定义

wireless\_copy.h来自内核头文件的Linux无线扩展定义的私有版本; 一旦合适的版本变得普遍可用，这可以用C库头文件替换。

## 2.3 加密功能

md5.c和md5.h MD5（如果包含TLS支持，则替换为加密库）HMAC-MD5（用于消息真实性验证的密钥校验和）

rc4.c和rc4.h RC4（广播/默认密钥加密）

sha1.c和sha1.h SHA-1（如果包含TLS支持，则替换为加密库）HMAC-SHA-1（用于消息真实性验证的密钥校验和）PRF-SHA-1（伪随机（密钥/随机数生成）函数） PBKDF2-SHA-1（ASCII密码到共享密钥）T-PRF（用于EAP-FAST）TLS-PRF（RFC 2246）。

sha256.c和sha256.h SHA-256（如果包含TLS支持，则替换为加密库）。

aes\_wrap.c，aes\_wrap.h，aes.c AES（如果包含TLS支持，则替换为加密库），具有128位KEK的AES密钥包装算法，RFC3394（广播/默认密钥加密），一键CBC MAC（ OMAC1）散列AES-128，AES-128 CTR模式加密，AES-128 EAX模式加密/解密，AES-128 CBC。

crypto.h加密库包装器的定义。

crypto\_openssl.c libcrypto的包装函数（OpenSSL）。

crypto\_internal.c Wrapper用于内部加密实现。

crypto\_gnutls.c libgcrypt的包装函数（由GnuTLS使用）。

ms\_funcs.c和ms\_funcs.h MSCHAPV2和LEAP的辅助函数。

tls.h TLS库包装器的定义。

tls\_none.c TLS库包装器的虚拟实现，用于不包含TLS功能的情况。

tls\_openssl.c用于openssl的TLS库包装器。

tls\_internal.c用于内部TLS实现的TLS库

## 2.4 TLS library

asn1.c和asn1.h ASN.1 DER解析

bignum.c和bignum.h大数学数学

rsa.c和rsa.h RSA

x509v3.c和x509v3.h X.509v3证书解析和处理

tlsv1\_client.c，tlsv1\_client.h TLSv1客户端（RFC 2246）

tlsv1\_client\_i.h TLSv1客户端的内部结构

tlsv1\_client\_read.c TLSv1客户端：读取握手消息

tlsv1\_client\_write.c TLSv1客户端：写握手消息

tlsv1\_common.c和tlsv1\_common.h常见的TLSv1例程和定义

tlsv1\_cred.c和tlsv1\_cred.h TLSv1凭据

tlsv1\_record.c和tlsv1\_record.h TLSv1记录协议

## 2.5配置

config\_ssid.h每个网络配置项的定义

config.h wpa\_supplicant配置的定义

config.c配置解析器和常用功能

config\_file.c文本文件的配置后端（例如，wpa\_supplicant.conf）

config\_winreg.c Windows注册表的配置后端

## 2.6控制端口

wpa\_supplicant有一个控制接口，可用于从外部程序获取状态信息和管理操作。 wpa\_supplicant发行版中包含此接口的示例命令行界面（wpa\_cli）和GUI（wpa\_gui）。

ctrl\_iface.c和ctrl\_iface.h wpa\_supplicant端的控制端口

ctrl\_iface\_unix.c基于UNIX域套接字的控制接口后端

ctrl\_iface\_udp.c基于UDP套接字的控制接口后端

ctrl\_iface\_named\_pipe.c Windows命名管道的控制接口后端

wpa\_ctrl.c和wpa\_ctrl.h用于外部程序的库函数，用于提供对wpa\_supplicant控件接口的访问

wpa\_cli.c使用wpa\_supplicant控制接口的示例程序

## 2.7 WPA supplicant

wpa.c和wpa.h WPA状态机和4路/组密钥握手处理

preauth.c和preauth.h PMKSA缓存和预身份验证（RSN / WPA2）

wpa\_i.h WPA代码的内部定义; 不包括在其他模块中。

## 2.8 EAP peer

EAP对等实现是一个单独的模块，除了wpa\_supplicant之外，其他程序可以使用它。

eap.c和eap.h EAP状态机和方法接口。

eap\_defs.h常见的EAP定义。

eap\_i.h EAP状态机和EAP方法的内部定义; 不包括在其他模块中。

eap\_sim\_common.c和eap\_sim\_common.h EAP-SIM和EAP-AKA的通用代码。

eap\_tls\_common.c和eap\_tls\_common.h EAP-PEAP，EAP-TTLS和EAP-FAST的通用代码。

eap\_tlv.c和eap\_tlv.h EAP-PEAP和EAP-FAST的EAP-TLV代码。

eap\_ttls.c和eap\_ttls.h EAP-TTLS。

eap\_pax.c，eap\_pax\_common.h，eap\_pax\_common.c EAP-PAX。

eap\_psk.c，eap\_psk\_common.h，eap\_psk\_common.c EAP-PSK（注意：WPA-PSK不需要这个）。

eap\_sake.c，eap\_sake\_common.h，eap\_sake\_common.c EAP-SAKE。

eap\_gpsk.c，eap\_gpsk\_common.h，eap\_gpsk\_common.c EAP-GPSK。

eap\_aka.c，eap\_fast.c，eap\_gtc.c，eap\_leap.c，eap\_md5.c，eap\_mschapv2.c，eap\_otp.c，eap\_peap.c，eap\_-sim.c，eap\_tls.c其他EAP方法实现。

## 2.9 EAPOL supplicant

eapol\_supp\_sm.c和eapol\_supp\_sm.h EAPOL请求状态机和IEEE 802.1X处理

## 2.10 Windows port

ndis\_events.c用于接收NdisMIndicateStatus（）事件并将其传递给wpa\_supplicant的代码。

driver\_ndis.c更易于使用的形式。

win\_if\_list.c用于列出当前网络接口的外部程序。

## 2.11测试程序

radius\_client.c和radius\_client.h eapol\_test的RADIUS身份验证客户端实现

radius.c和radius.h eapol\_test的RADIUS消息处理

eapol\_test.c具有集成RADIUS身份验证客户端的独立EAP测试工具

preauth\_test.c独立的RSN预身份验证工具

wpa\_passphrase.c WPA ASCII密码短语到PSK转换

**第三章 wpa\_supplicant控制端口**

wpa\_supplicant实现一个控制接口，外部程序可以使用该接口来控制wpa\_supplicant守护程序的操作并获取状态信息和事件通知。 有一个小的C库，以单个C文件的形式，wpa\_ctrl.c，提供帮助函数以便于使用控制接口。 外部程序可以将此文件链接到它们中，然后使用wpa\_ctrl.h中记录的库函数与wpa\_supplicant进行交互。 该库也可以与C ++一起使用。 wpa\_cli.c和wpa\_gui是使用此库的示例程序。

进程间通信有多种机制。例如，Linux版本的wpa\_supplicant使用UNIX域套接字作为控制接口和Windows版本UDP套接字。使用wpa\_ctrl.h中定义的函数可以用于从外部程序隐藏使用的IPC的详细信息。

## 3.1使用控制接口

需要与wpa\_supplicant通信的外部程序（例如GUI或配置实用程序）应链接到wpa\_ctrl.c中。 这允许他们使用辅助函数通过wpa\_ctrl\_open（）打开与控制接口的连接，并使用wpa\_ctrl\_request（）发送命令。

wpa\_supplicant使用控制接口进行两种类型的通信：命令和未经请求的事件消息。命令是一对消息，来自外部程序的请求和来自wpa\_supplicant的响应。这些可以使用wpa\_ctrl\_request（）执行。未经请求的事件消息由wpa\_supplicant发送到控制接口连接，而没有来自外部程序的特定请求以接收每个消息。但是，外部程序需要使用wpa\_ctrl\_attach（）附加到控制接口以接收这些未经请求的消息。

如果控制接口连接同时用于命令和未经请求的事件消息，则可能在命令请求和响应之间接收未经请求的消息。wpa\_ctrl\_- request（）调用者需要提供一个回调函数msg\_cb来处理这些消息。通常通过调用wpa\_ctrl\_open（）两次打开两个控制接口连接更容易，然后使用其中一个连接用于命令，另一个连接用于未经请求的消息。这种方式命令请求/响应对不会被未经请求的消息破坏。wpa\_cli是如何仅为一个目的使用一个连接的示例，wpa\_gui演示了如何使用两个单独的连接。

一旦不再需要控制接口连接，就应该通过调用wpa\_ctrl\_- close（）来关闭它。如果连接用于未经请求的事件消息，则应首先通过调用wpa\_ctrl\_detach（）来分离它。

## 3.2控制接口命令

以下命令可与wpa\_ctrl\_request（）一起使用：

### 3.2.1 PING

此命令可用于测试wpa\_supplicant是否正在回复控制接口命令。 如果连接已打开且wpa\_supplicant正在处理命令，则预期答复为PONG。

### 3.2.2 MIB

请求MIB变量列表（dot1x，dot11）。 输出是一个文本块，每行以variable = value格式。 例如：

dot11RSNAOptionImplemented=TRUE

dot11RSNAPreauthenticationImplemented=TRUE

dot11RSNAEnabled=FALSE

dot11RSNAPreauthenticationEnabled=FALSE

dot11RSNAConfigVersion=1

dot11RSNAConfigPairwiseKeysSupported=5

dot11RSNAConfigGroupCipherSize=128

dot11RSNAConfigPMKLifetime=43200

dot11RSNAConfigPMKReauthThreshold=70

dot11RSNAConfigNumberOfPTKSAReplayCounters=1

dot11RSNAConfigSATimeout=60

dot11RSNAAuthenticationSuiteSelected=00-50-f2-2

dot11RSNAPairwiseCipherSelected=00-50-f2-4

dot11RSNAGroupCipherSelected=00-50-f2-4

dot11RSNAPMKIDUsed=

dot11RSNAAuthenticationSuiteRequested=00-50-f2-2

dot11RSNAPairwiseCipherRequested=00-50-f2-4

dot11RSNAGroupCipherRequested=00-50-f2-4

dot11RSNAConfigNumberOfGTKSAReplayCounters=0

dot11RSNA4WayHandshakeFailures=0

dot1xSuppPaeState=5

dot1xSuppHeldPeriod=60

dot1xSuppAuthPeriod=30

dot1xSuppStartPeriod=30

dot1xSuppMaxStart=3

dot1xSuppSuppControlledPortStatus=Authorized

dot1xSuppBackendPaeState=2

dot1xSuppEapolFramesRx=0

dot1xSuppEapolFramesTx=440

dot1xSuppEapolStartFramesTx=2

dot1xSuppEapolLogoffFramesTx=0

dot1xSuppEapolRespFramesTx=0

dot1xSuppEapolReqIdFramesRx=0

dot1xSuppEapolReqFramesRx=0

dot1xSuppInvalidEapolFramesRx=0

dot1xSuppEapLengthErrorFramesRx=0

dot1xSuppLastEapolFrameVersion=0

dot1xSuppLastEapolFrameSource=00:00:00:00:00:00

### 3.2.3 状态

请求当前的WPA / EAPOL / EAP状态信息。 输出是一个文本块，每行以variable = value格式。 例如：

bssid=02:00:01:02:03:04

ssid=test network

pairwise\_cipher=CCMP

group\_cipher=CCMP

key\_mgmt=WPA-PSK

wpa\_state=COMPLETED

ip\_address=192.168.1.21

Supplicant PAE state=AUTHENTICATED

suppPortStatus=Authorized

EAP state=SUCCESS

### 3.2.4状态 – VERBOSE

与STATUS相同，但具有更多详细程度（即，更多变量=值对）。

bssid=02:00:01:02:03:04

ssid=test network

id=0

pairwise\_cipher=CCMP

group\_cipher=CCMP

key\_mgmt=WPA-PSK

wpa\_state=COMPLETED

ip\_address=192.168.1.21

Supplicant PAE state=AUTHENTICATED

suppPortStatus=Authorized

heldPeriod=60

authPeriod=30

startPeriod=30

maxStart=3

portControl=Auto

Supplicant Backend state=IDLE

EAP state=SUCCESS

reqMethod=0

methodState=NONE

decision=COND\_SUCC

ClientTimeout=60

### 3.2.5 PMKSA

显示PMKSA缓存

Index / AA / PMKID / expiration (in seconds) / opportunistic

1 / 02:00:01:02:03:04 / 000102030405060708090a0b0c0d0e0f / 41362 / 0

1. / 02:00:01:33:55:77 / 928389281928383b34afb34ba4212345 / 362 / 1

### 3.2.6 SET <variable> <value>

设置变量：

* EAPOL::heldPeriod
* EAPOL::authPeriod
* EAPOL::startPeriod
* EAPOL::maxStart
* dot11RSNAConfigPMKLifetime
* dot11RSNAConfigPMKReauthThreshold
* dot11RSNAConfigSATimeout

示例命令：

SET EAPOL::heldPeriod 45

### 3.2.7 LOGON

IEEE 802.1X EAPOL状态机登录。

### 3.2.8 LOGOFF

IEEE 802.1X EAPOL状态机注销。

### 3.2.9 REASSOCIATE

强制重新关联。

### 3.2.10 RECONNECT

如果断开连接（即，如REASSOCIATE，但仅在断开连接状态下才连接）。

### 3.2.11 PREAUTH <BSSID>

使用给定的BSSID开始预身份验证。

### 3.2.12 ATTACH

将连接附加为未经请求的事件的监视器。 这可以使用wpa\_ctrl\_attach（）完成。

### 3.2.13 DETACH

将连接分离为未经请求的事件的监视器。 这可以使用wpa\_ctrl\_detach（）完成。

### 3.2.14 LEVEL <debug level>

更改调试级别。

### 3.2.15 RECONFIGURE

强制wpa\_supplicant重新读取其配置数据。

### 3.2.16 TERMINATE

终止wpa\_supplicant进程。

### 3.2.17 BSSID <network id> <BSSID>

为网络设置首选BSSID。 可以从LIST\_NETWORKS命令输出接收网络ID。

### 3.2.18 LIST\_NETWORKS

列出已配置的网络

network id / ssid / bssid / flags

0 example network any [CURRENT]

### 3.2.19 DISCONNECT

在连接之前断开连接并等待REASSOCIATE或RECONNECT命令。

### 3.2.20 SCAN

请求新的BSS扫描。

### 3.2.21 SCAN\_RESULTS

获取最新的扫描结果。

bssid / frequency / signal level / flags / ssid

00:09:5b:95:e0:4e 2412 208 [WPA-PSK-CCMP] jkm private

02:55:24:33:77:a3 2462 187 [WPA-PSK-TKIP] testing

00:09:5b:95:e0:4f 2412 209 jkm guest

### 3.2.22 BSS

获取详细的每BSS扫描结果。 BSS命令可用于一次迭代扫描结果一个BSS并从找到的BSS获取所有信息。 这提供了对通过SCAN\_RESULTS可用的相同数据的访问，但是以避免大量扫描结果不适合ctrl\_iface消息的问题的方式。

使用BSS命令选择BSS有两种选择：“BSS <idx>”请求扫描结果表中索引（0 .. size-1）标识的BSS信息和“BSS <BSSID>”请求信息 对于给定的BSS（基于00：01：02：03：04：05格式的BSSID）。

BSS信息以下列格式呈现。 请注意，可能会在此field = value数据中添加新字段，因此ctrl\_iface用户应准备忽略它不理解的值。

bssid=00:09:5b:95:e0:4e

freq=2412

beacon\_int=0

capabilities=0x0011

qual=51

noise=161

level=212

tsf=0000000000000000

ie=000b6a6b6d2070726976617465010180dd180050f20101000050f20401000050f20401000050f2020000

ssid=jkm private

### 3.2.23 SELECT\_NETWORK <network id>

选择网络（禁用其他网络）。 可以从LIST\_NETWORKS命令输出接收网络ID。

### 3.2.24 ENABLE\_NETWORK <network id>

启用网络。 可以从LIST\_NETWORKS命令输出接收网络ID。 特殊网络ID都可用于启用所有网络。

### 3.2.25 DISABLE\_NETWORK <network id>

禁用网络。 可以从LIST\_NETWORKS命令输出接收网络ID。 特殊网络ID都可用于禁用所有网络。

### 3.2.26 ADD\_NETWORK

添加新网络。 此命令创建一个空配置的新网络。 新网络已禁用，配置完成后，可以使用ENABLE\_NETWORK命令启用它。 ADD\_NETWORK返回新网络的网络ID或失败时失败。

### 3.2.27 REMOVE\_NETWORK <network id>

删除网络。 可以从LIST\_NETWORKS命令输出接收网络ID。 特殊网络ID都可以用来删除所有网络。

### 3.2.28 SET\_NETWORK <network id> <variable> <value>

设置网络变量。 可以从LIST\_NETWORKS命令输出接收网络ID。 此命令使用与配置文件相同的变量和数据格式。 有关更多详细信息，请参阅示例wpa\_supplicant.conf。

* ssid (network name, SSID)
* psk (WPA passphrase or pre-shared key)
* key\_mgmt (key management protocol)
* identity (EAP identity)
* password (EAP password)
* ...

### 3.2.29 GET\_NETWORK <network id> <variable>

获取网络变量。 可以从LIST\_NETWORKS命令输出接收网络ID。

### 3.2.30 SAVE\_CONFIG

保存当前配置。

## 3.3互动请求

如果wpa\_supplicant在身份验证期间需要其他信息（例如，密码），则它将在未经请求的事件消息中使用特定前缀CTRL-REQ-（WPA\_CTRL\_REQ宏）。 外部程序（例如，GUI）可以通过在具有匹配字段名称的命令中使用CTRL-RSP-（WPA\_CTRL\_RSP宏）前缀来提供这样的信息。

可以通过这种方式从用户请求以下字段：

* IDENTITY (EAP identity/user name)
* PASSWORD (EAP password)
* NEW\_PASSWORD (New password if the server is requesting password change)
* PIN (PIN code for accessing a SIM or smartcard)
* OTP (one-time password; like password, but the value is used only once)
* PASSPHRASE (passphrase for a private key file)

CTRL-REQ-<field name>-<network id>-<human readable text>

CTRL-RSP-<field name>-<network id>-<value>

例如，来自wpa\_supplicant的请求：

CTRL-REQ-PASSWORD-1-Password needed for SSID test-network

以及来自GUI的匹配回复：

CTRL-RSP-PASSWORD-1-secret

### 3.3.1 GET\_CAPABILITY <option> [strict]

获取支持的功能列表（eap，pairwise，group，proto）。 支持的功能显示为以与wpa\_supplicant配置相同的格式使用的空间单独的值列表。 如果添加了可选参数'strict'，则仅包含驱动程序声明明确支持的值。 如果没有这个，如果驱动程序没有提供查询功能的机制，则包括所有可用功能。

示例请求/回复对：

GET\_CAPABILITY eap

AKA FAST GTC LEAP MD5 MSCHAPV2 OTP PAX PEAP PSK SIM TLS TTLS

GET\_CAPABILITY pairwise

CCMP TKIP NONE

GET\_CAPABILITY pairwise strict

GET\_CAPABILITY group

CCMP TKIP WEP104 WEP40

GET\_CAPABILITY key\_mgmt

WPA-PSK WPA-EAP IEEE8021X NONE

GET\_CAPABILITY proto

RSN WPA

GET\_CAPABILITY auth\_alg

OPEN SHARED LEAP

### 3.3.2 AP\_SCAN <ap\_scan value>

更改ap\_scan值：0 =无扫描，1 = wpa\_supplicant请求扫描并使用扫描结果选择AP，2 = wpa\_supplicant不使用扫描，只是请求驱动程序关联并处理AP选择。

### 3.3.3 INTERFACES

列出已配置的接口

wlan0

eth0

# 第四章 驱动程序包装器实现(driver.h, drivers.c)

所有依赖于硬件和驱动程序的功能都在单独的C文件中，这些文件实现了定 wpa\_supplicant的其他部分设计为独立于硬件，驱动程序和操作系统。

驱动程序包装器需要实现目标操作系统/驱动程序中用于控制无线LAN设备的任何调用。例如，在Linux的情况下，这些主要是针对Linux无线扩展（WE）的一些粘合代码和ioctl（）调用以及netlink消息解析。由于WPA所需的功能最近才添加到Linux Wireless Extensions（版本18）中，因此在许多驱动程序接口实现中使用了一些特定于驱动程序的代码。一旦目标驱动程序包含对WE-18的完全支持，这些驱动程序相关部件可以用driver\_wext.c中的通用代码替换。之后，至少在理论上，所有Linux驱动程序都可以使用相同的驱动程序包装代码。

驱动程序包装器需要实现driver.h中定义的部分或全部功能。通过使用函数指针填充struct wpa\_driver\_ops来注册这些函数。wpa\_- supplicant的硬件独立部分将调用这些函数来控制驱动程序/wlan卡。此外，还需要支持驾驶员事件。事件回调函数wpa\_supplicant\_event（）及其参数记录在driver.h中。另外，需要在drivers.c文件中注册指向'struct wpa\_driver\_ops'的指针。

移植到其他操作系统时，应修改驱动程序包装程序以使用目标操作系统的本机接口。 在移植到新操作系统期间，可能会发现驱动程序包装器和通用wpa\_supplicant代码之间的接口的一些额外要求。 这些将通过修改接口和更新其他驱动程序包装器来逐个处理。 目标是避免在没有充分理由的情况下更改此接口，以限制wpa\_supplicant的其他包装器和硬件独立部分所需的更改数量。 当需要进行更改时，建议的方法是以向后兼容的方式使它们允许编译现有的驱动程序接口实现而不进行任何修改。

通用Linux无线扩展功能在driver\_wext.c中实现。当内核驱动程序支持通用的ioctl（）和无线事件时，所有Linux驱动程序包装器都可以使用它们。驱动程序特定功能在单独的C文件中实现，例如driver\_hostap.c。这些文件需要定义struct wpa\_ driver\_ops条目，该条目将在调用驱动程序函数时在wpa\_supplicant.c中使用。struct wpa\_driver\_ops条目在drivers.c中注册。

通常，在开始实现新的驱动程序接口示例之前，首先看看几个驱动程序接口示例可能很有用。driver\_hostap.c和driver\_wext.c包含一个完整的Linux驱动程序实现，它使用基于wpa\_supplicant的WPA IE控制和漫游。driver\_ndis.c（在driver\_ndis\_.c的帮助下）是Windows NDIS接口的完整界面示例，用于自行生成WPA IE并决定何时漫游的驱动程序。这些示例实现包括对所有安全模式的完全支持。

## 4.1 WPA的驱动程序要求

WPA引入了设备驱动程序的新要求。 至少需要实现其中一些，以便为wpa\_supplicant提供足够的支持。

### 4.1.1 TKIP/CCMP

WPA要求成对密码套件（单播数据包的加密算法）是TKIP或CCMP。 这些是新的加密协议，因此需要修改驱动程序以支持它们。 根据所使用的wlan硬件，这些部分可能由硬件/固件实现。

TKIP和CCMP的规范可从IEEE（IEEE 802.11i修订版）获得。 Host AP驱动程序（driver / modules / hostap\_ {tkip，ccmp} .c）中也提供了两种加密协议的全功能，硬件独立实现。 此外，Linux 2.6内核树具有可用于Linux驱动程序的WEP，TKIP和CCMP的通用实现。

驱动程序还需要提供配置机制以允许用户空间程序配置TKIP和CCMP。 Linux Wireless Extensions v18增加了对配置这些算法和单个/非默认密钥的支持。 如果目标内核不包含WE-18，则可以使用私有ioctl来提供类似的功能。

### 4.1.2漫游控制和扫描支持

wpa\_supplicant可以可选地基于从信标和/或探测响应帧接收的信息来控制AP选择（配置中的ap\_scan = 1模式）。这意味着驱动程序应支持扫描过程的外部控制。对于Linux，建议使用新的无线扩展扫描支持（即'iwlist wlan0 scan'）。当前的驱动程序包装器（driver\_wext.c）将此用于扫描结果。

扫描结果还必须包含WPA信息元素。 WE-18增加了对此的支持。 对于旧版本，可以使用自定义事件将完整的WPA IE（包括元素ID和长度）作为扫描结果中包含的十六进制字符串提供。

wpa\_supplicant还需要能够请求驱动程序与特定BSS关联。 当前主机AP驱动程序和匹配的driver\_hostap.c包装程序对此请求使用以下序列。 类似/相同的机制也应该可用于其他驱动程序。

* 使用私有ioctl为AssocReq设置WPA IE
* 使用SIOCSIWESSID设置SSID
* 使用SIOCSIWFREQ设置通道/频率
* 使用SIOCSIWAP设置BSSID（最后一个ioctl将触发驱动程序请求关联）

### 4.1.3 WPA IE生成

wpa\_supplicant选择使用哪些密码套件和密钥管理套件。基于此信息，它生成WPA IE。这是在关联调用中提供给驱动程序接口的。这与自己生成WPA IE的Windows NDIS驱动程序不匹配。

wpa\_supplicant通过在关联调用中提供选定的密码和密钥管理套件，允许类似Windows NDIS的行为。如果驱动程序生成其自己的WPA IE并且与wpa\_supplicant生成的不同，则驱动程序必须通知wpa\_supplicant所使用的WPA IE（即，它在（重新）关联请求中使用的那个）。此通知使用EVENT\_ASSOCINFO事件完成（请参阅driver.h）。 wpa\_supplicant通常配置为使用ap\_scan = 2模式和控制WPA IE生成和漫游的驱动程序。

# 第五章 EAP对等实现

可扩展身份验证协议（EAP）是RFC 3748中定义的身份验证框架。wpa\_supplicant使用单独的代码模块进行EAP对等实现。该模块设计为仅使用一组最小的直接函数调用（主要是调试/事件函数），以便在其他程序中使用。EAP实现的设计基于RFC 4137。状态机在此RFC中定义，因此是对等状态机和方法之间的接口。因此，此RFC提供了有用的信息，用于理解wpa\_supplicant中的EAP对等实现。

EAP状态机中使用的一些术语是指EAPOL（IEEE 802.1X），但如果EAP模块是为除wpa\_supplicant之外的其他程序构建的，则下层没有严格要求IEEE 802.1X。 这些术语应理解为指RFC 4137中定义的较低层。

## 5.1添加EAP方法

每个EAP方法都作为一个单独的模块实现，通常作为一个名为eap\_ <方法名称> .c的C文件，例如eap\_md5.c。所有EAP方法在对等状态机和方法特定功能之间使用相同的接口。这允许添加新的EAP方法而无需修改核心EAP状态机实现。

需要通过将新EAP方法添加到eap\_methods.c的eap\_peer\_register\_methods（）函数中的构建（Makefile）和EAP方法注册列表中来注册新的EAP方法。每个EAP方法应使用构建时配置选项，例如EAP\_TLS，以便可以选择构建中包含哪些方法。

EAP方法必须实现eap\_i.h中定义的接口。struct eap\_method定义了每个EAP方法必须提供的所需函数指针。 此外，使用此结构注册EAP类型和名称。 此接口基于RFC 4137的第4.4节。

建议EAP方法在处理消息时使用通用辅助函数eap\_msg\_alloc（）和eap\_hdr\_validate（）。这允许代码共享并且可以避免遗漏所接收的分组的一些所需的验证步骤。此外，如果需要，这些功能可以更容易地在扩展EAP标头和旧版EAP标头之间进行更改。

添加使用供应商特定EAP类型（扩展类型，如RFC 3748，第5.7章中定义）的EAP方法时，必须通过将供应商ID而不是EAP\_VENDOR\_IETF传递给eap\_peer\_method\_alloc（）来注册新方法。这些方法不能尝试通过为类型254注册遗留EAP方法来模拟扩展类型。有关作为扩展类型实现的EAP方法实现的示例，请参阅eap\_vendor\_test.c。

## 5.2将EAP实施用作库

Git存储库有一个eap\_example目录，其中包含一个示例，显示如何将来自wpa\_supplicant和hostapd的EAP对等和服务器代码用作库。 示例程序初始化EAP服务器和EAP对等实体，然后运行EAP-PEAP / MSCHAPv2身份验证。

eap\_example\_peer.c显示了控制EAP对等实现所需的初始化和粘合代码。 eap\_example\_server.c对EAP服务器执行相同操作。 eap\_example.c是一个绑定EAP服务器和客户端部分的示例，以允许显示EAP身份验证。

在此示例中，EAP消息在服务器之间传递，并且对等体在同一进程内通过直接函数调用传递。实际上，服务器和对等功能可能驻留在单独的设备中，并且EAP消息将基于外部协议在设备之间传输。例如，在IEEE 802.11中使用IEEE 802.1X EAPOL状态机来控制EAP消息的传输，并且WiMax支持可选的PMK EAP认证机制，其发送IEEE 802.16e中定义的EAP消息。

EAP库链接了大量src / utils和src / crypto目录中的辅助函数。其中大多数都是合适的，但是可能需要通过从库中删除此文件并以更好地适应的方式重新实现其中的函数来替换src / utils / wpa\_debug.c中的调试输出代码。

# 第六章 EAP服务器实现

可扩展身份验证协议（EAP）是RFC 3748中定义的身份验证框架。hostapd使用单独的代码模块进行EAP服务器实现。该模块设计为仅使用一组最小的直接函数调用（主要是调试/事件函数），以便在其他程序中使用。EAP实现的设计基于RFC 4137。状态机在此RFC中定义，因此服务器状态机和方法之间的接口也是如此。 因此，此RFC提供了有用的信息，用于理解hostapd中的EAP服务器实现。

EAP状态机中使用的一些术语是指EAPOL（IEEE 802.1X），但如果EAP模块是为除wpa\_supplicant之外的其他程序构建的，则下层没有严格要求IEEE 802.1X。 这些术语应理解为指RFC 4137中定义的较低层。

## 6.1添加EAP方法

每个EAP方法都作为一个单独的模块实现，通常作为一个名为eap\_ <方法名称> .c的C文件，例如eap\_md5.c。 所有EAP方法都使用服务器状态机和方法特定功能之间的相同接口。 这允许添加新的EAP方法而无需修改核心EAP状态机实现。

需要通过将新EAP方法添加到eap\_methods.c的eap\_server\_register\_methods（）函数中的构建（Makefile）和EAP方法注册列表中来注册新的EAP方法。每个EAP方法应使用构建时配置选项，例如EAP\_TLS，以便可以选择构建中包含哪些方法。

EAP方法必须实现eap\_i.h中定义的接口。struct eap\_method定义了每个EAP方法必须提供的所需函数指针。 此外，使用此结构注册EAP类型和名称。 此接口基于RFC 4137的第4.4节。

建议EAP方法在处理消息时使用通用辅助函数eap\_msg\_alloc（）和eap\_hdr\_validate（）。这允许代码共享并且可以避免遗漏所接收的分组的一些所需的验证步骤。此外，如果需要，这些功能可以更容易地在扩展EAP标头和旧版EAP标头之间进行更改。

添加使用供应商特定EAP类型的EAP方法（RFC 3748中定义的扩展类型，第5.7章）时，必须通过将供应商ID而不是EAP\_VENDOR\_IETF传递给eap\_server\_method\_alloc（）来注册新方法。这些方法不能尝试通过为类型254注册遗留EAP方法来模拟扩展类型。有关作为扩展类型实现的EAP方法实现的示例，请参阅eap\_vendor\_test.c。

# 第七章 hostapd控制接口

hostapd实现了一个控制接口，外部程序可以使用该接口来控制hostapd守护程序的操作并获取状态信息和事件通知。有一个小的C库，以单个C文件的形式，wpa\_ctrl.c，提供帮助函数以便于使用控制接口。外部程序可以将此文件链接到它们中，然后使用wpa\_ctrl.h中记录的库函数与wpa\_supplicant进行交互。 该库也可以与C ++一起使用。 hostapd\_cli.c是使用此库的示例程序。

进程间通信有多种机制。例如，Linux版本的hostapd使用UNIX域套接字作为控制接口。使用wpa\_ctrl.h中定义的函数可以用于从外部程序隐藏使用的IPC的详细信息。

## 7.1使用控制接口

需要与hostapd通信的外部程序（例如GUI或配置实用程序）应链接到wpa\_ctrl.c中。 这允许他们使用辅助函数通过wpa\_ctrl\_open（）打开与控制接口的连接，并使用wpa\_ctrl\_request（）发送命令。

hostapd使用控制接口进行两种类型的通信：命令和未经请求的事件消息。命令是一对消息，来自外部程序的请求和来自hostapd的响应。这些可以使用wpa\_ctrl\_request（）执行。hostapd将未经请求的事件消息发送到控制接口连接，而无需外部程序的特定请求来接收每条消息。但是，外部程序需要使用wpa\_ctrl\_attach（）附加到控制接口以接收这些未经请求的消息。

如果控制接口连接同时用于命令和未经请求的事件消息，则可能在命令请求和响应之间接收未经请求的消息。wpa\_ctrl\_- request（）调用者需要提供一个回调函数msg\_cb来处理这些消息。通常通过调用wpa\_ctrl\_open（）两次打开两个控制接口连接更容易，然后使用其中一个连接用于命令，另一个连接用于未经请求的消息。这种方式命令请求/响应对不会被未经请求的消息破坏。wpa\_cli是如何仅为一个目的使用一个连接的示例，wpa\_gui演示了如何使用两个分离连接。

一旦不再需要控制接口连接，就应该通过调用wpa\_ctrl\_close（）来关闭它。如果连接用于未经请求的事件消息，则应首先通过调用wpa\_ctrl\_detach（）来分离它。

## 7.2控制接口命令

以下命令可与wpa\_ctrl\_request（）一起使用：

### 7.2.1 PING

此命令可用于测试hostapd是否正在回复控制接口命令。如果连接已打开且hostapd正在处理命令，则预期答复为PONG。

# 第八章 移植到不同的目标板和操作系统

wpa\_supplicant旨在轻松移植到不同的硬件（板，CPU）和软件（操作系统，驱动程序）目标。它已经与许多操作系统和众多无线卡模型和驱动程序一起使用。主要的wpa\_supplicant存储库包括对Linux，FreeBSD和Windows的支持。此外，代码已移植到许多其他操作系统，如VxWorks，PalmOS，Windows CE和Windows Mobile。在硬件方面，wpa\_supplicant用于各种系统：台式机，笔记本电脑，PDA和带有CPU的嵌入式设备，包括x86，PowerPC，arm / xscale和MIPS。支持大端和小端配置。

## 8.1 ANSI C之上的额外功能

wpa\_supplicant主要使用大多数目标都可用的ANSI C函数。 但是，使用了几种在现代UNIX系统上常见的附加功能。 其中的数量与common.h中的原型一起列出（#ifdef CONFIG\_ANSI\_C\_EXTRA）。这些函数可能需要实现或至少定义为目标OS或C库中的本机函数的宏。

许多常见的ANSI C函数通过os.h中的包装器定义使用，以便在标准C库不可用的情况下，使用特定于平台的版本轻松替换这些函数。此外，os.h定义了几个常见的平台特定函数，这些函数在os\_unix.c中为UNIX（如目标）和os\_win32.c（对于Win32 API）实现。 如果目标平台不支持这些示例中的任何一个，则可能需要添加新的os \_ \* .c文件。

除非定义了OS\_NO\_C\_LIB\_DEFINES，否则通过定义os \_ \*（）包装器来使用标准ANSI C和POSIX函数来直接使用它们，以避免额外的大小和速度成本。如果目标平台需要不同版本的函数，则可以修改os.h以定义合适的宏，或者可以为构建定义OS\_NO\_C\_LIB\_DEFINES，然后可以在新的os \_ \* .c包装器文件中实现包装函数。

common.h定义了用于处理不同大小和字节顺序的整数的辅助宏的数量。可能需要为目标平台添加这些定义的合适版本。

## 8.2配置后端

wpa\_supplicant实现了一个配置接口，允许轻松替换后端，以便根据目标平台从合适的源读取配置数据。 config.c实现了可以与所有配置后端共享的通用代码。 每个后端都在自己的config \_ \* .c文件中实现。

包含的config\_file.c后端使用文本文件进行配置，config\_winreg.c使用Windows注册表。 如果目标平台使用不同的配置参数机制，则可以将这些文件用作新配置后端的示例。 此外，config\_none.c可以用作构建新配置后端的空白起点。

## 8.3 驱动接口

除非已经支持目标OS和驱动程序，否则大多数移植项目必须实现驱动程序包装器。 这可以通过添加新的驱动程序接口模块或修改现有模块（driver \_ \* .c）来完成，如果新目标与其中一个类似。驱动程序包装器实现描述了驱动程序接口的详细信息，并讨论了移植wpa\_supplicant这部分所涉及的任务。

## 8.4 l2\_packet（链路层访问）

wpa\_supplicant需要访问发送和接收具有两种以太网类型的第2层（链路层）数据包：EAP-over-LAN（EAPOL）0x888e和RSN预认证0x88c7。l2\_packet.h在核心wpa\_supplicant实现中定义了用于此的接口。

如果目标操作系统支持链接层访问的通用机制，那么这可能是为wpa\_supplicant提供所需功能的最佳机制。 Linux数据包套接字就是这种通用机制的一个例子。 如果这不可用，则可能需要为网络堆栈或驱动程序实现单独的接口。这通常是在设备驱动程序和OS网络堆栈之间运行的中间或协议驱动程序。如果这种机制不可行，则界面也可以直接在设备驱动程序中实现。

主要的wpa\_supplicant存储库包括使用数据包套接字的Linux的l2\_packet实现（l2\_packet\_linux.c），使用libpcap / libdnet库的更多可移植版本（l2\_packet\_pcap.c;这也支持WinPcap），以及FreeBSD特定版本的libpcap接口（l2\_packet\_freebsd.c）。

如果libpcap（接收）和libdnet（发送）支持目标操作系统，则l2\_packet\_pcap.c可能会在最小或没有更改的情况下使用。 如果不是这种情况或需要链路层的专有接口，则可能需要添加新的l2\_packet模块。 或者，如果链接层访问与驱动程序接口实现集成，则struct wpa\_driver\_ops :: send\_- eapol（）处理程序可用于覆盖l2\_packet库。

## 8.5事件循环

wpa\_supplicant使用单个进程/线程模型和事件循环来提供事件回调（已注册超时，已接收数据包，信号）。 eloop.h定义了事件循环接口。 eloop.c是使用select（）和套接字的这种事件循环的实现。这适用于大多数UNIX / POSIX系统。移植到其他操作系统时，可能需要使用提供类似功能的操作系统特定机制替换该实现。

## 8.6 控制接口

wpa\_supplicant使用控制接口允许外部处理获取状态信息并控制操作。目前，这是通过基于套接字的通信实现的; 支持UNIX域套接字和UDP套接字。如果目标操作系统不支持套接字，则可能需要修改此接口以使用其他机制（如消息队列）。 控制接口是可选组件，因此也可以在不移植此部件的情况下运行wpa\_supplicant。

控制接口的wpa\_supplicant端在ctrl\_iface.c中实现。匹配客户端实现为wpa\_ctrl.c中的控件接口库。

## 8.7程序入口点

wpa\_supplicant定义了一组可用于初始化主请求者处理的函数。每个操作系统都有一个启动新处理或线程的机制。这通常是具有一组特定参数和调用约定的函数。该函数负责初始化wpa\_supplicant。

main.c包含类UNIX操作系统的入口点，即main（）函数，它使用命令行参数来设置wpa\_supplicant的参数。 移植到其他操作系统时，需要类似的特定于操作系统的入口点实现。 它可以在一个新文件中实现，然后用wpa\_supplicant而不是main.o链接。 main.c也是如何完成初始化过程的一个很好的例子。

请求者初始化函数在wpa\_supplicant\_i.h中定义。在大多数情况下，入口点函数应该从获取配置参数开始。在此之后，通过调用wpa\_supplicant\_init（）初始化全局wpa\_supplicant上下文。在此之后，可以使用wpa\_supplicant\_add\_iface（）添加现有网络接口。然后使用wpa\_supplicant\_run（）来启动主事件循环。一旦在程序终止时返回，wpa\_supplicant\_deinit（）用于释放全局上下文数据。

可以动态使用wpa\_supplicant\_add\_iface（）和wpa\_supplicant\_remove\_iface（）来根据何时需要进行wpa\_supplicant处理来添加和删除接口。例如，当插入和弹出热插拔网络适配器时，可以这样做。如果希望同时完全启用/禁用接口的wpa\_supplicant处理，则在启用/禁用网络接口时也可以执行此操作。

## 8.8简单构建示例

启动移植项目的一种方法是从一个非常简单的wpa\_supplicant构建开始，支持WPA-PSK，一旦构建正确，就开始添加功能。

以下命令可用于构建非常简单的wpa\_supplicant版本：

cc -o wpa\_supplicant config.c eloop.c common.c md5.c rc4.c sha1.c \

config\_none.c l2\_packet\_none.c tls\_none.c wpa.c preauth.c \

aes\_wrap.c wpa\_supplicant.c events.c main\_none.c drivers.c

最终结果并不是非常有用，因为它使用空函数进行配置解析和第2层数据包访问，并且不包含驱动程序接口。 但是，这是一个很好的起点，因为构建完成后所有功能都存在，并且只需包含列出的C文件就可以轻松配置到构建系统。

一旦成功构建此版本，最终结果可以通过添加适当的程序入口点（main \* .c），驱动程序接口（driver \_ \* .c和匹配CONFIG\_DRIVER\_ \* define以在drivers.c中注册）来实现最终结果，配置 解析器/编写器（config \_ \* .c）和第2层数据包访问实现（l2\_packet \_ \* .c）。 添加完这些组件后，最终结果应该是一个正在运行的WPA / WPA2-PSK启用的请求者。

在验证基本功能工作之后，可以通过链接更多文件和定义C预处理器定义来添加更多功能。目前，可用选项和需要包含哪些文件的最佳信息来源是用于使用make构建请求者的Makefile。对于使用不同类型的make工具或基于GUI的项目配置的构建系统，将需要类似的配置。