NAME

net80211_vap - 802.11 network layer virtual radio support

SYNOPSIS

#include <net80211/ieee80211_var.h>

int
ieee80211_vap_setup(*struct ieee80211com *, struct ieee80211vap *, const char name[IFNAMSIZ], int unit, int opmode, int flags, const uint8_t bssid[IEEE80211_ADDR_LEN], const uint8_t macaddr[IEEE80211_ADDR_LEN]*);

int
ieee80211_vap_attach(*struct ieee80211vap *, ifm_change_cb_t media_change, ifm_stat_cb_t media_stat*);

void
ieee80211_vap_detach(*struct ieee80211vap **);

DESCRIPTION

The **net80211** software layer provides a support framework for drivers that includes a virtual radio API that is exported to users through network interfaces (aka vaps) that are cloned from the underlying device. These interfaces have an operating mode (station, adhoc, hostap, wds, monitor, etc.) that is fixed for the lifetime of the interface. Devices that can support multiple concurrent interfaces allow multiple vaps to be cloned.

The virtual radio interface defined by the **net80211** layer means that drivers must be structured to follow specific rules. Drivers that support only a single interface at any time must still follow these rules.

The virtual radio architecture splits state between a single per-device *ieee80211com* structure and one or more *ieee80211vap* structures. Vaps are created with the SIOCIFCREATE2 request. This results in a call into the driver's *ic_vap_create* method where the driver can decide if the request should be accepted.

The vap creation process is done in three steps. First the driver allocates the data structure with malloc(9). This data structure must have an *ieee80211vap* structure at the front but is usually extended with driver-private state. Next the vap is setup with a call to **ieee80211_vap_setup**(). This request initializes **net80211** state but does not activate the interface. The driver can then override methods setup by **net80211** and setup driver resources before finally calling **ieee80211_vap_attach**() to complete the process. Both these calls must be done without holding any driver locks as work may require the process block/sleep.

A vap is deleted when an SIOCIFDESTROY ioctl request is made or when the device detaches (causing all associated vaps to automatically be deleted). Delete requests cause the *ic_vap_delete* method to be called. Drivers must quiesce the device before calling **ieee80211_vap_detach**() to deactivate the vap and isolate it from activities such as requests from user applications. The driver can then reclaim resources held by the vap and re-enable device operation. The exact procedure for quiescing a device is unspecified but typically it involves blocking interrupts and stopping transmit and receive processing.

MULTI-VAP OPERATION

Drivers are responsible for deciding if multiple vaps can be created and how to manage them. Whether or not multiple concurrent vaps can be supported depends on a device's capabilities. For example, multiple hostap vaps can usually be supported but many devices do not support assigning each vap a unique BSSID. If a device supports hostap operation it can usually support concurrent station mode vaps but possibly with limitations such as losing support for hardware beacon miss support. Devices that are capable of hostap operation and can send and receive 4-address frames should be able to support WDS vaps together with an ap vap. But in contrast some devices cannot support WDS vaps without at least one ap vap (this however can be finessed by forcing the ap vap to not transmit beacon frames). All devices should support the creation of any number of monitor mode vaps concurrent with other vaps but it is the responsibility of the driver to allow this.

An important consequence of supporting multiple concurrent vaps is that a driver's *iv_newstate* method must be written to handle being called for each vap. Where necessary, drivers must track private state for all vaps and not just the one whose state is being changed (e.g. for handling beacon timers the driver may need to know if all vaps that beacon are stopped before stopping the hardware timers).

SEE ALSO

ieee80211(9), ifnet(9), malloc(9)