NAME

ieee80211_proto - 802.11 state machine support

SYNOPSIS

#include <net80211/ieee80211_var.h>

void
ieee80211_start_all(*struct ieee80211com* *);

void
ieee80211_stop_all(*struct ieee80211com* *);

void
ieee80211_suspend_all(*struct ieee80211com* *);

void
ieee80211_resume_all(*struct ieee80211com* *);

enum ieee80211_state;
int
ieee80211_new_state(*struct ieee80211vap* *, *enum ieee80211_state*, *int*);

void
ieee80211_wait_for_parent(*struct ieee80211com* *);

DESCRIPTION

The **net80211** layer that supports 802.11 device drivers uses a state machine to control operation of vaps. These state machines vary according to the vap operating mode. Station mode state machines follow the 802.11 MLME states in the protocol specification. Other state machines are simpler and reflect operational work such as scanning for a BSS or automatically selecting a channel to operate on. When multiple vaps are operational the state machines are used to coordinate operation such as choosing a channel. The state machine mechanism also serves to bind the **net80211** layer to a driver; this is described more below.

The following states are defined for state machines:

IEEE80211_S_INIT Default/initial state. A vap in this state should not hold any dynamic state (e.g. entries for associated stations in the node table). The driver must quiesce the hardware; e.g. there should be no interrupts firing.

IEEE80211_S_SCAN Scanning for a BSS or choosing a channel to operate on. Note that scanning can also take place in other states (e.g. when background scanning is active); this state is entered when initially bringing a vap to an operational state or after an event such as a beacon miss (in station mode).

IEEE80211_S_AUTH Authenticating to an access point (in station mode). This state is normally reached from IEEE80211_S_SCAN after selecting a BSS, but may also be reached from IEEE80211_S_ASSOC or IEEE80211_S_RUN if the authentication handshake fails.

IEEE80211_S_ASSOC Associating to an access point (in station mode). This state is reached from IEEE80211_S_AUTH after successfully authenticating or from IEEE80211_S_RUN if a DisAssoc frame is received.

IEEE80211_S_CAC Doing Channel Availability Check (CAC). This state is entered only when DFS is enabled and the channel selected for operation requires CAC.

IEEE80211_S_RUN Operational. In this state a vap can transmit data frames, accept requests for stations associating, etc. Beware that data traffic is also gated by whether the associated "port" is authorized. When WPA/802.11i/802.1x is operational authorization may happen separately; e.g. in station mode wpa_supplicant(8) must complete the handshakes and plumb the necessary keys before a port is authorized. In this state a BSS is operational and associated state is valid and may be used; e.g. *ic_bss* and *ic_bsschan* are guaranteed to be usable.

IEEE80211_S_CSA Channel Switch Announcement (CSA) is pending. This state is reached only from IEEE80211_S_RUN when either a CSA is received from an access point (in station mode) or the local station is preparing to change channel. In this state traffic may be muted depending on the Mute setting in the CSA.

IEEE80211_S_SLEEP Asleep to save power (in station mode). This state is reached only from IEEE80211_S_RUN when power save operation is enabled and the local station is deemed sufficiently idle to enter low power mode.

Note that states are ordered (as shown above); e.g. a vap must be in the IEEE80211_S_RUN or "greater" before it can transmit frames. Certain **net80211** data are valid only in certain states; e.g. the *iv_bsschan* that specifies the channel for the operating BSS should never be used except in IEEE80211_S_RUN or greater.

STATE CHANGES

State machine changes are typically handled internal to the **net80211** layer in response to ioctl(2) requests, received frames, or external events such as a beacon miss. The **ieee80211_new_state**() function is used to initiate a state machine change on a vap. The new state and an optional argument are supplied. The request is initially processed to handle coordination of multiple vaps. For example, only one vap at a time can be scanning, if multiple vaps request a change to IEEE80211_S_SCAN the first will be permitted to run and the others will be *deferred* until the scan operation completes at which time the selected channel will be adopted. Similarly **net80211** handles coordination of combinations of vaps such as an AP and station vap where the station may need to roam to follow the AP it is associated to (dragging along the AP vap to the new channel). Another important coordination is the handling of IEEE80211_S_CAC and IEEE80211_S_CSA. No more than one vap can ever be actively changing state at a time. In fact **net80211** single-threads the state machine logic in a dedicated taskqueue(9) thread that is also used to synchronize work such as scanning and beacon miss handling.

After multi-vap scheduling/coordination is done the per-vap *iv_newstate* method is called to carry out the state change work. Drivers use this entry to setup private state and then dispatch the call to the **net80211** layer using the previously defined method pointer (in OOP-parlance they call the "super method").

net80211 handles two state changes specially. On transition to IEEE80211_S_RUN the IFF_DRV_OACTIVE bit on the vap's transmit queue is cleared so traffic can flow. On transition to IEEE80211_S_INIT any state in the scan cache associated with the vap is flushed and any frames pending on the transmit queue are flushed.

DRIVER INTEGRATION

Drivers are expected to override the *iv_newstate* method to interpose their own code and handle setup work required by state changes. Otherwise drivers must call **ieee80211_start_all**() in response to being marked up through an SIOCSIFFLAGS ioctl request and they should use **ieee80211_suspend_all**() and **ieee80211_resume_all**() to implement suspend/resume support.

There is also an **ieee80211_stop_all**() call to force all vaps to an IEEE80211_S_INIT state but this should not be needed by a driver; control is usually handled by **net80211** or, in the case of card eject or vap destroy, work will be initiated outside the driver.

SEE ALSO

ioctl(2), wpa_supplicant(8), ieee80211(9), ifnet(9), taskqueue(9)

HISTORY

The state machine concept was part of the original **ieee80211** code base that first appeared in NetBSD 1.5.