NAME

ieee80211_amrr - 802.11 network driver transmit rate control support

SYNOPSIS

#include <net80211/ieee80211_amrr.h>

void
ieee80211_amrr_init(*struct ieee80211_amrr *, struct ieee80211vap *, int amin, int amax, int interval*);

void
ieee80211_amrr_cleanup(*struct ieee80211_amrr **);

void
ieee80211_amrr_setinterval(*struct ieee80211_amrr *, int interval*);

void
ieee80211_amrr_node_init(*struct ieee80211_amrr *, struct ieee80211_amrr_node *,*
*struct ieee80211_node **);

int
ieee80211_amrr_choose(*struct ieee80211_node *, struct ieee80211_amrr_node **);

void
ieee80211_amrr_tx_complete(*struct ieee80211_amrr_node *, int ok, int retries*);

void
ieee80211_amrr_tx_update(*struct ieee80211_amrr_node *, int txnct, int success, int retrycnt*);

DESCRIPTION

ieee80211_amrr is an implementation of the AMRR transmit rate control algorithm for drivers that use the **net80211** software layer. A rate control algorithm is responsible for choosing the transmit rate for each frame. To maximize throughput algorithms try to use the highest rate that is appropriate for the operating conditions. The rate will vary as conditions change; the distance between two stations may change, transient noise may be present that affects signal quality, etc. **ieee80211_amrr** uses very simple information from a driver to do it's job: whether a frame was successfully delivered and how many transmit attempts were made. While this enables its use with virtually any wireless device it limits it's effectiveness--do not expect it to function well in difficult environments and/or respond quickly to changing conditions.

ieee80211_amrr requires per-vap state and per-node state for each station it is to select rates for. The

API's are designed for drivers to pre-allocate state in the driver-private extension areas of each vap and node. For example the ral(4) driver defines a vap as:

```
struct rt2560_vap {
        struct ieee80211vap     ral_vap;
        struct ieee80211_beacon_offsets ral_bo;
        struct ieee80211_amrr   amrr;

        int     (*ral_newstate)(struct ieee80211vap *,
                    enum ieee80211_state, int);
};
```

The *amrr* structure member holds the per-vap state for **ieee80211_amrr** and ral(4) initializes it in the vap create method with:

```
ieee80211_amrr_init(&rvp->amrr, vap,
    IEEE80211_AMRR_MIN_SUCCESS_THRESHOLD,
    IEEE80211_AMRR_MAX_SUCCESS_THRESHOLD,
    500 /* ms */);
```

The node is defined as:

```
struct rt2560_node {
        struct ieee80211_node   ni;
        struct ieee80211_amrr_node amrr;
};
```

with initialization done in the driver's *iv_newassoc* method:

```
static void
rt2560_newassoc(struct ieee80211_node *ni, int isnew)
{
        struct ieee80211vap *vap = ni->ni_vap;

        ieee80211_amrr_node_init(&RT2560_VAP(vap)->amrr,
            &RT2560_NODE(ni)->amrr, ni);
}
```

Once **ieee80211_amrr** state is setup, transmit rates are requested by calling **ieee80211_amrr_choose**() in the transmit path; e.g.:

```
tp = &vap->iv_txparms[ieee80211_chan2mode(ni->ni_chan)];
if (IEEE80211_IS_MULTICAST(wh->i_addr1)) {
        rate = tp->mcastrate;
} else if (m0->m_flags & M_EAPOL) {
        rate = tp->mgmtrate;
} else if (tp->ucastrate != IEEE80211_FIXED_RATE_NONE) {
        rate = tp->ucastrate;
} else {
        (void) ieee80211_amrr_choose(ni, &RT2560_NODE(ni)->amrr);
        rate = ni->ni_txrate;
}
```

Note a rate is chosen only for unicast data frames when a fixed transmit rate is not configured; the other cases are handled with the **net80211** transmit parameters. Note also that **ieee80211_amrr_choose**() writes the chosen rate in *ni_txrate*; this eliminates copying the value as it is exported to user applications so they can display the current transmit rate in status.

The remaining work a driver must do is feed status back to **ieee80211_amrr** when a frame transmit completes using **ieee80211_amrr_tx_complete**(). Drivers that poll a device to retrieve statistics can use **ieee80211_amrr_tx_update**() (instead or in addition).

SEE ALSO

ieee80211(9), ieee80211_output(9)