

MadWifi Driver Summary

Wenhua Zhao (whzhao@gmail.com)

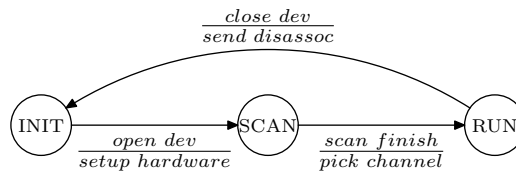
August 21, 2007

Using MadWifi driver, we can create multiple virtual interfaces on one physical network card. Each virtual interface can work in different modes, namely AP, STA, IBSS(adhoc), Monitor and WDS(bridge).

This section briefly describe the behavior of the AP and STA operations according to the MadWifi driver.

1. AP Operation

The state diagram of AP mode is shown in the following figure.



When the driver is loaded, it probes the physical network card and then sets up the device (`ath_attach()`). The driver also automatically creates a virtual network interface operating in the mode specified (`ieee80211_create_vap()`). The initial state of the virtual interface is INIT. While in INIT state, the hardware does not receive packets.

When the AP interface is opening (for example, by using the command `ifconfig ath0 up`), the driver sets the hardware properly and enters SCAN state. In SCAN state, the AP scans all supported channels. The scan includes active scan, in which the AP sends out probe request, and passive scan, in which the AP listens to beacons from neighboring APs. In SCAN state, the AP does not transmit packets.

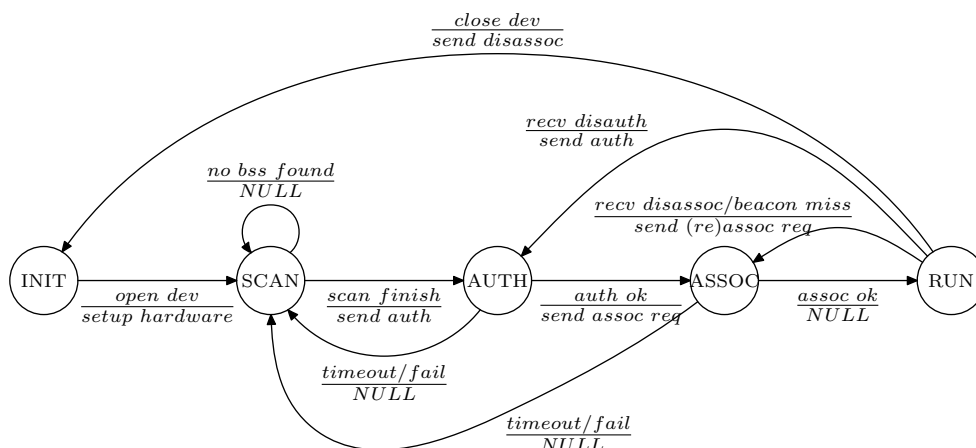
After all channels have been scanned, the AP picks a quiet channel which has the smallest rssi, and then enters RUN state (`ap_end()`). In RUN state, the AP performs normal operations of an access point. It broadcasts beacon messages about every 100 ms (`ath_beacon_send()`), replies probe requests sent by other APs, replies authentication and (re)association messages sent by clients, and transmits data packets.

When the interfacing is closing, the AP sends disassociation messages to every associated client, then it cleans up the resources and enters INIT state.

One thing to note is that the control messages, such as RTS, CTS and ACK, are handled by the HAL part of the driver. The open source part does not process these control messages.

2. STA Operation

The state diagram of STA mode is shown in the following figure.



For STA, the operations in INIT state are the same as those for AP.

In SCAN state, the STA also scans all the supported channels. After the scan is done, the STA selects one bss (AP) that has a desired essid and the highest rssi (`sta_pick_bss()`). If no bss with a matching essid is found, the STA restarts a new scan. If a bss is selected, the STA sets up the parameters required to communicate with the bss, and then enters AUTH state (`ieee80211_sta_join1()`).

On entering AUTH state, the STA starts an authentication procedure by sending an authentication message to the bss. The authentication procedure includes a sequence of messages exchanged between STA and AP. If the authentication succeeds, the STA enters ASSOC state. On the other hand, if the authentication fails or if the STA does not receive any response from the bss within 5 seconds (`ieee80211_tx_timeout()`) the STA goes back to SCAN state.

On entering ASSOC state, the STA sends an association request message to bss and waits for a response. If the STA receives a successful association response, it goes to RUN state. If the association fails (eg. error response or rate negotiation failure), or if the STA does not receive any association response within 5 seconds, the STA goes to SCAN state.

In RUN state, the STA can exchange data packets with the bss. The STA also listens to management messages. If the STA receives a dis-association message from the bss, it sends an association request and goes to ASSOC state. If the STA receives a dis-authentication message, it sends an authentication message and goes to AUTH state.

In RUN state, the STA maintains the connectivity to the bss by listening to beacon messages. If 10 consecutive beacons are missed (`ath_beacon_config()`), the connection to the bss is considered broken. The STA sends a re-association request to the bss and enters ASSOC state (`ieee80211_beacon_miss()`).

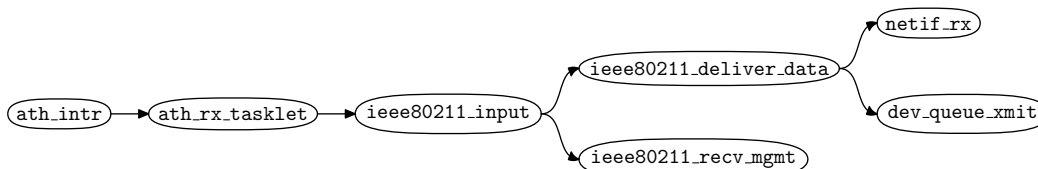
When the interface is closing, the STA informs the bss by sending a disassociation message to the bss and then enters INIT state.

From the above procedure we can identify a handoff procedure of a STA. The handoff procedure starts when the STA is in RUN state and has 10 consecutive beacons missing.

The STA sends a re-association request to the old bss and enters ASSOC state. Without hearing any reply from the bss, the STA enters SCAN state to search for any new bss.

3. Packet Receiving

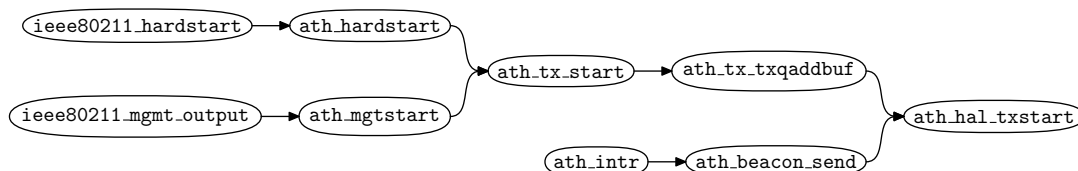
The major functions involved in the packet receiving process is shown in the following figure.



Most of the CSMA/CA mechanism is implemented in HAL or hardware. The open source driver is notified by interrupt when a new packet arrives (`ath_intr()`). The packet is handled by the Linux tasklet `ath_rx_tasklet()`, where the `skb` structure of the packet is retrieved and the receiving node of the packet is identified. The function `ieee80211_input()` receives different types of packets. In this function, management packets are passed to `ieee80211_recv_mgmt()`, while data packets are decapsulated to ethernet format and passed to Linux kernel.

4. Packet Transmitting

The major functions involved in the packet transmitting process is shown in the following figure.



The kernel calls `ieee80211_hardstart()` (`dev->hard_start_xmit` of the virtual interface) to transmit a packet, which in turn calls `ath_hardstart()` (`dev->hard_start_xmit` of the physical interface). The function `ath_hardstart()` encapsulates the ethernet packet into 802.11 packet. The function `ath_tx_start()` encrypts the packets if necessary, maps the `skb` to DMA buffer and selects a transmit queues according the priority of the packet (QoS control). The function `ath_tx_txqaddbuf()` inserts the buffer into the selected transmit queue and calls the HAL functions to start a transmission.

Management packets are generated within the 802.11 layer. They are transmitted from the function `ieee80211_mgmt_output()`.

Beacon messages are triggered by HAL. When it's time to send a beacon, the HAL issues an interrupt, then the function `ath_beacon_send()` is invoked to send the beacon message. The beacon messages are directly passed to the HAL to transmit.

After the HAL successfully transmitted a packet, an interrupt is generated (`ath_intr()`) to report the transmission of the packet as shown in the following figure. The function `ath_tx_tasklet()` updates information for this transmission. If there is a virtual interface working in monitor mode, the packet is passed to the monitor interface in function `ath_tx_capture()`.

