mac80211 overview

Johannes Martin Berg

2009-02-25
mac80211

• is a subsystem to the Linux kernel
• implements shared code for soft-MAC/half-MAC wireless devices
• contains MLME and other code, despite the name
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 2006</td>
<td>John Linville starts as wireless maintainer</td>
</tr>
<tr>
<td>April 2006</td>
<td>First wireless summit (Beaverton)</td>
</tr>
<tr>
<td>May 1, 2006</td>
<td>Devicescape press release (Advanced Datapath Driver as GPLv2)</td>
</tr>
<tr>
<td>May 2006 - May 2007</td>
<td>Lots of work on stack (initially much by Jiri Benc/SuSE)</td>
</tr>
<tr>
<td></td>
<td>including rename from d80211 to mac80211</td>
</tr>
<tr>
<td>May 5, 2007</td>
<td>Merged for 2.6.22</td>
</tr>
<tr>
<td>Oct 23, 2007</td>
<td>I first 'officially' take mac80211 responsibility</td>
</tr>
</tbody>
</table>
Some notable additions to mac80211:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Contributor/Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>HT/aggregation support</td>
<td>Intel</td>
</tr>
<tr>
<td>802.11s draft support</td>
<td>cozybit through o11s.org</td>
</tr>
<tr>
<td>802.11w draft support</td>
<td>Jouni Malinen (Atheros)</td>
</tr>
<tr>
<td>PS (infrastructure mode)</td>
<td>Kalle Valo (Nokia)</td>
</tr>
<tr>
<td>beacon processing offload (WIP)</td>
<td>Vivek Natarajan (Atheros)</td>
</tr>
<tr>
<td></td>
<td>Kalle Valo (Nokia)</td>
</tr>
</tbody>
</table>
beacon processing offload

- beacon processing
  - beacon miss actions
  - signal strength monitoring
  - beacon change monitoring
- offload
  - don’t use software for above tasks
  - have device (firmware) do this
  - results in much fewer CPU wakeups
Architecture

userspace

nl80211

cfg80211

wext

cfg80211_ops

mac80211

ieee80211_ops

iwlwifi

other drivers
Architecture

internally

- TX/RX paths (including software en-/decryption)
- control paths for managed, IBSS, mesh
- some things for AP (e.g. powersave buffering)
- ...
Code structure

Most important for driver authors:

`include/net/mac80211.h`

This file defines the API to `mac80211` from below.
## Code structure

All files except the header file are in **net/mac80211/**.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kconfig, Makefile</td>
<td>build system</td>
</tr>
<tr>
<td>ieee80211_i.h</td>
<td>most internal data structures</td>
</tr>
<tr>
<td>main.c</td>
<td>main module entry points</td>
</tr>
<tr>
<td>iface.c</td>
<td>main entry points for driver calls (reg/dereg)</td>
</tr>
<tr>
<td>key.c, key.h</td>
<td>virtual interface handling</td>
</tr>
<tr>
<td>sta_info.c, sta_info.h</td>
<td>Station (peer) management</td>
</tr>
<tr>
<td>pm.c</td>
<td>power management (suspend/hibernate)</td>
</tr>
<tr>
<td>rate.c, rate.h</td>
<td>internal rate control functions</td>
</tr>
<tr>
<td>rc80211*</td>
<td>rate control algorithms</td>
</tr>
<tr>
<td>rx.c</td>
<td>frame receive path</td>
</tr>
<tr>
<td>tx.c</td>
<td>frame transmit path</td>
</tr>
<tr>
<td>scan.c</td>
<td>software scanning code</td>
</tr>
</tbody>
</table>
Code structure

ht.c, agg-rx.c, agg-tx.c
mesh{,_hwmp,_plink,_pathtbl}.{c,h}
mlme.c
ibss.c
cfg.c, cfg.h, wext.c
event.c
spectmgmt.c
aes*, tkip.*, wep.*, michael.*, wpa.*
wme.c, wme.h
util.c
led.c, led.h
debugfs*

HT/aggregation code
802.11s mesh
Station/managed mode MLME
IBSS MLME
configuration entry points
events to userspace
spectrum management code
WPA/RSN/WEP code
some QoS code
utility functions
LED handling
debugfs code
Data structures

- ieee80211_local/ieee80211_hw
- sta_info/ieee80211_sta
- ieee80211_conf
- ieee80211_bss_conf
- ieee80211_key/ieee80211_key_conf
- ieee80211_tx_info
- ieee80211_rx_status
- ieee80211_sub_if_data/ieee80211_vif
Data structures – ieee80211_local/ieee80211_hw

- each instance of these (hw is embedded into local) represents a wireless device
- ieee80211_hw is the part of ieee80211_local that is visible to drivers
- contains all operating information about a wireless device
Data structures – sta_info/ieee80211_sta

- represents any station (peer)
- could be mesh peer, IBSS peer, AP, WDS peer
- would also be used for DLS peer
- ieee80211_sta is driver-visible part
- ieee80211_find_sta for drivers
- lifetime managed mostly with RCU
Data structures – ieee80211_conf

• hardware configuration
• most importantly - current channel
• intention: hardware specific parameters
Data structures – ieee80211_bss_conf

- BSS configuration
- for all kinds of BSSes (IBSS/AP/managed)
- contains e.g. basic rate bitmap
- intention: per BSS parameters in case hardware supports creating/associating with multiple BSSes
Data structures – ieee80211_keyieee80211_key_conf

- represents an encryption/decryption key
- ieee80211_key_conf given to driver for hardware acceleration
- ieee80211_key contains internal book-keeping and software encryption state
Data structures – ieee80211_tx_info

- most complicated data structure
- lives inside skb’s control buffer (cb)
- goes through three stages (substructure for each)
  - initialisation by mac80211 (control)
  - use by driver (driver_data/rate_driver_data)
  - use for TX status reporting (status)
Data structures – ieee80211_rx_status

- contains status about a received frame
- passed by driver to mac80211 with a received frame
Data structures – ieee80211_sub_if_data/ieee80211_vif

- contains information about each virtual interface
- ieee80211_vif is passed to driver for those virtual interfaces the driver knows about (not monitor, VLAN)
- contains sub-structures depending on mode
  - ieee80211_if_ap
  - ieee80211_if_wds
  - ieee80211_if_vlan
  - ieee80211_if_managed
  - ieee80211_if_ibss
  - ieee80211_if_mesh
Main flows

- configuration
- receive path
- transmit path
- management/MLME
Main flows – configuration

• all initiated from userspace (wext or nl80211)
• for managed and IBSS modes: triggers statemachine (on workqueue)
• some operations passed through to driver more or less directly (e.g. channel setting)
Main flows – receive path

- packet received by driver
- passed to mac80211’s rx function (ieee80211_rx) with rx_status info
- for each interface that the packet might belong to
  - RX handlers are invoked
  - data: converted to 802.3, delivered to networking stack
  - management: delivered to MLME
Main flows – transmit path

- packet handed to virtual interface’s ieee80211_subif_start_xmit
- converted to 802.11 format
- sent to master interface
- packet handed to ieee80211_master_start_xmit
- transmit handlers run, control information created
- packet given to driver
Main flows – transmit path

transmit handlers

- ieee80211_tx_h_check_assoc
- ieee80211_tx_h_ps_buf
- ieee80211_tx_h_select_key
- ieee80211_tx_h_michael_mic_add
- ieee80211_tx_h_rate_ctrl
- ieee80211_tx_h_misc
- ieee80211_tx_h_sequence
- ieee80211_tx_h_fragment
- ieee80211_tx_h_encrypt
- ieee80211_tx_h_calculate_duration
- ieee80211_tx_h_stats
Main flows – management/MLME
Main flows – management/MLME

Ok, so you didn’t want to know that precisely.

- requests from user are translated to internal variables
- state machine is run depending on user request
- normal way looks like this:
  - probe request/response
  - auth request/response
  - assoc request/response
  - notification to userspace
Main flows – management/MLME

For IBSS (wasn’t on the state machine slide) it’s simpler

- try to find IBSS
- join IBSS or create IBSS
- if no peers periodically try to find IBSS to join
Handoff points

Three main points

• configuration (from userspace)
• mac80211/rate control
• mac80211/driver
Handoff points – configuration

- Wireless extensions (wext)
- cfg80211 (which userspace talks to via nl80211)
Handoff points – configuration – wext

Currently still includes

- setting SSID, BSSID and other association parameters
- setting RTS/fragmentation thresholds
- encryption keys in managed/IBSS modes
Handoff points – configuration – cfg80211

Is being extended, already has

• scanning
• station management (AP)
• mesh management
• virtual interface management
• encryption keys in AP mode

(See more in cfg80211/nl80211/userspace talk.)
Handoff points – from mac80211 to rate control

- Rate control is semantically not part of driver
- Per-driver selection of rate control algorithm
- Rate control fills ieee80211_txd_info rate information
- Rate control informed of TX status
Handoff points – from mac80211 to driver

- many driver methods (ieee80211.ops)
- mac80211 also has a lot of exported functions
- refer to include/net/mac80211.h
Execution contexts

- config flows: userspace process context
- state machine flows: workqueue context
- packet processing flows: tasklet context
- some callbacks: interrupt context (_irqsafe functions)
Synchronisation mechanisms

background – RCU

• **read - copy - update**
• think read/write locks without locking reads
• instead, copy structure, and atomically publish
• problem: when to get rid of old copy
Synchronisation mechanisms

background – rtnl

- “big networking lock”, global lock
- used to protect all configuration calls, e.g. interface start/stop
- consequently used by wireless extensions to protect config calls
Synchronisation mechanisms

- config flows: mostly rtnl
- a lot of RCU-based synchronisation (sta_info, key management)
- mutex for interface list management
- spinlocks for various tightly constrained spots like sta list management, sta_info members etc.
- some more specialised locks
Stay up-to-date

- also http://wireless.kernel.org/en/developers/todo-list/
- subscribe to wiki changes on these pages
- follow patches going in: git log -- net/mac80211/
- read the wireless list (http://wireless.kernel.org/en/developers/MailingLists)
Thank you for your attention.

Questions?
virtual interfaces

vif 1

vif 2

vif 3

...

master interface

driver/hardware
virtual interfaces

- allow, in theory, multiple network interfaces on single hardware
- for example WDS and AP interfaces (to be bridged)
- for example multiple AP interfaces (multi-BSS)
- any number of monitor interfaces
- any number of AP_VLAN interfaces (to implement multi-SSID with single BSSID)
virtual interfaces

supported interface types

- ad-hoc (IBSS)
- managed
- AP and AP_VLAN
- WDS
- mesh point
- monitor
virtual interfaces

relevancy to drivers

- drivers need to allow each interface type
- drivers need to support certain operations for certain interface types
- drivers can support multiple virtual interfaces
- **but:** drivers not notified of monitor interfaces
filter flags

- used to configure hardware filters
- best-effort, not all filter flags need to be supported
- best-effort, not all filters need to be supported
- filter flags say which frames to pass to mac80211 – thus a filter flag is supported if that type of frames passed to mac80211
- passing more frames than requested is always permitted but may affect performance
filter flags

monitor interfaces

• handled entirely in mac80211
• may affect filters depending on configuration
• it is possible to create a monitor interface that does not affect filters, can be useful for debugging (iw phy phy0 interface add moni0 type monitor flags none)
Even backup slides end somewhere.