

MADWiFi

Multiband Atheros Driver for WiFi

Irfan Sheriff

Different wireless chipsets

- Prism54 based 802.11a/b/g cards
- ZyDAS 1201 based 802.11a/b/g cards
- TI ACX100/111 based 802.11a/b/g cards
- ..
- ..
- **Atheros based 802.11a/b/g cards**

So what about it ?

- Open Source Linux kernel driver for Wireless LAN chipsets from Atheros
- Great flexibility
- Active development
- ..

Supported cards

Netgear

- WAG511
- WAG311
- WG311
- WG311T
- WG511T
- WG511U
- WPN311
- WPN311GR
- WPN511

D-Link

- DWL-AG660
- DWL-AB650
- DWL-650+
- DWL-G650
- DWL-AG650
- DWL-AG520
- DWL-G520
- DWL-AG530
- DWL-G630

USB is not supported yet
For complete list

<http://madwifi.org/wiki/Compatibility>

madwifi-ng

- Old codebase
 - Code used till a few months back
 - <http://snapshots.madwifi.org/madwifi-old/>
- New codebase
 - Atheros started their own internal branch to add features to the driver
 - New code a lot different from the old codebase
 - No stable release yet
 - <http://snapshots.madwifi.org/madwifi-ng/>

Atheros chipsets

Atheros 5210 - 11a

Atheros 5211 - 11a/b

Atheros 5212 - 11a/b/g

iwpriv ath0 mode 1 lock operation to 11a only

iwpriv ath0 mode 2 lock operation to 11b only

iwpriv ath0 mode 3 lock operation to 11g only

iwpriv ath0 mode 0 autoselect from 11a/b/g (default)

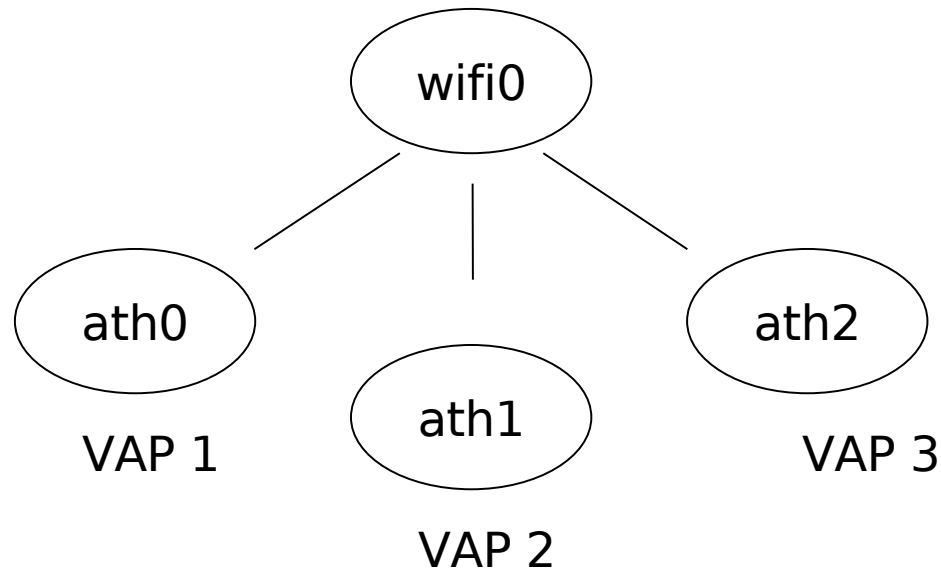
Build

- make; make install;
- modprobe ath_pci;
- Modules
 - wlan: 802.11 state machine, device independent
 - wlan_wep, wlan_tkip, wlan_ccmp, wlan_acl, wlan_auth, wlan_radius
 - ath_hal: atheros chip specific
 - Hardware Access Layer
 - ath_rate_sample
 - Link adaptation modules

Getting the interface up

- wlanconfig

- wlanconfig athX create wlandev wifiX wlanmode [sta|ap|adhoc|monitor|ahdemo]
- wlanconfig athX destroy
- wlanconfig athX list [scan|active|chan|sta]



Creating multiple interfaces

```
wlanconfig ath0 create wlandev wifi0 wlanmode ap
```

```
wlanconfig ath1 create wlandev wifi0 wlanmode ap
```

```
wlanconfig ath2 create wlandev wifi0 wlanmode ap
```

```
iwconfig ath0 essid "lmn"
```

```
iwconfig ath1 essid "XYZ"
```

```
iwconfig ath2 essid "abc"
```

To force the ap's to share the address of the base device, use:

```
wlanconfig ath create wlandev wifi0 wlanmode ap -bssid
```

```
wlanconfig ath create wlandev wifi0 wlanmode ap -bssid
```

Creating multiple interfaces

- One station VAP can exist on a device. If the station VAP is the first, then no other VAPs are allowed.
- When ap VAP and station coexist, the 'nosbeacon' should be disabled, this flag disables the use of hardware beacon timers for the station mode operation.
- If "-bssid" is used, the mac address of the VAP is cloned from the mac address of the base network device

Configuring the interface

- iwconfig

- iwconfig interface [essid X]
[freq X]
[rts X]
[rate X]
[power X]
[sens X]
[retry X]

Scanning the interface

- wlanconfig athX list [scan|active|chan|sta]
 - Show the list of aps around
 - Show the list of connected peers
 - Show the list of channels
 - Show the list of available channels

SSID	BSSID	CHAN	RATE	S:N	INT	CAPS
eddie	00:06:25:e8:3a:05	6	54M	36:0	100	EPs

A few things..

MAC address change

- `ifconfig wifi0 down hw ether xx:xx:xx:xx:xx:xx`
- `ifconfig wifi0 up`

Creating access control list

- `iwpriv ath0 maccmd 1 -- whitelist`
 - 2 -- blacklist
 - 3 -- flushlist
- `iwpriv ath0 addmac 00:01:02:03:04:05`

The /proc

- Under `/proc/sys/dev/wifiX`
 - `acktimeout`
 - `ctstimeout`
 - `slottime`
 - `diversity`
 - `txantenna`
 - `rxantenna`

A few handy tools

- `80211stats -i ath0`

- X rx from wrong bssid

- X rx discard due to its a dup

- X rx discard due to mcast echo

- X rx discard mgmt frames

- X rx element unknown

- X rx frame chan mismatch

- X active scans started

A few handy tools..

- `athstats -i ath0`

- X tx management frames

- X tx failed due to too many retries

- X lon on-chip tx retries

- X tx frames with no ack marked

- X tx frames with short preamble

- X tx frames with alternate rate

- X rx failed due to CRC

- X PHY errors

- X OFDM timing

- X CCK timing

- X CCK restart

- rssr of last ack: X

- rssr of last rcv: X

Signal strength

- Conversion for Atheros

RSSI_Max = 60

Convert % to RSSI

Subtract 95 from RSSI to derive dBm

This gives a dBm range of -35dBm at 100% and -95dBm at 0%

Justification: At a constant temperature, the thermal noise at any frequency looking into a 50Ω load is -174dBm/Hz. For a 20Mhz OFDM channel: $-174 + 10\log_{10}(20 \times 10^6)$, or -101.7dBm thermal noise at the antenna. If an additional 5dBm noise from the amplifier, -96dBm noise floor inside the radio!

- RSSI

10 or less represents a weak signal

20 is decent.

40 or more is very strong and should be able to support both 54MBit/s

A few handy tools..

- `athctrl -i ath0 -d 1000`
 - Sets CTS/retransmission timeouts and few other timing parameters based on distance
 - Prevents unnecessary collisions for long distance links
- `athchans -i ath0 5-9`
 - Restricts channel search in the range

So you want to do..

- Per packet power control
- Per packet channel switch
- Per packet retransmission control
- Per packet RSSI monitor
- Per packet antenna control
- Add a rate adaptation module

The HAL

```
HAL_BOOL __ahdecl(*ah_setupTxDesc)(struct ath_hal *, struct ath_desc *,
    u_int pktLen, u_int hdrLen,
    HAL_PKT_TYPE type, u_int txPower,
    u_int txRate0, u_int txTries0,
    u_int keyIx, u_int antMode, u_int flags,
    u_int rtsctsRate, u_int rtsctsDuration,
    u_int compicvLen, u_int compivLen,
    u_int comp);
```

- Sets-up a transmit descriptor. txPower is 0.5dBm steps, txRate0 and txTries0 are the initial data rate and the number of retries at that data rate if no ack is received. The ack processing and the retries are performed in hardware. antMode specifies which antenna to use for transmission

The HAL

```
HAL_BOOL __ahdecl(*ah_reset)(struct ath_hal *, HAL_OPMODE,  
                            HAL_CHANNEL *, HAL_BOOL bChannelChange,  
                            HAL_STATUS *status);
```

- ah_reset resets the chip and defines the new operation mode and channel. It needs to be called to change the channel.
- The developers say that before calling ah_reset() one has to call ath_hal_init_channels(), or else the reset doesn't occur properly.

OpenHAL

- The OpenHAL ported to MADWiFi at <http://pdos.csail.mit.edu/~jbicket/openhal/>
- What works: 802.11 station mode (receiving packets, monitor mode, transmitting, associating)
- You guys should check out

Using rate adaptation

- In Makefile, `ATH_RATE=ath_rate/sample`

Or call make:

- `make ATH_RATE=ath_rate/sample`
- Add your own rate adaptation module too

A few issues..

- ad hoc mode is broken
 - Long stable experiments is a difficulty
 - 'ahdemo' has been added to use the old mode functionality
- Switching 802.11 authentication mode
 - iwpriv ath0 authmode 1
 - Key gets cleared
- wlanconfig create and destroy results in athN being unusable
 - Use wlanconfig ath ... to let the driver select the appropriate number

A few issues

- WEP results in authorization failed
 - Order of assigning key could matter, try:
 - iwconfig ath0 key XXXXXXXXXXXX
 - iwconfig ath0 ap XX:XX:XX:XX:XX:XX
- In summary, madwifi-ng is yet to have a stable release
 - Bugs.. Bugs..
 - expect unreliable behaviour

Thanks