## NAME

**ath** - Atheros IEEE 802.11 wireless network driver

## SYNOPSIS

To compile this driver into the kernel, place the following lines in your kernel configuration file:

```
device ath
device ath_hal
device ath_rate_sample
device wlan
```

Alternatively, to load the driver as a module at boot time, place the following line in loader.conf(5):

```
if_ath_load="YES"
```

## DESCRIPTION

The **ath** driver provides support for wireless network adapters based on the Atheros AR5210, AR5211, AR5212, AR5416 and AR9300 programming APIs. These APIs are used by a wide variety of chips; most all chips with a PCI, PCIe and/or CardBus interface are supported.

Supported features include 802.11 and 802.3 frames, power management, BSS, IBSS, MBSS, WDS/DWDS TDMA, and host-based access point operation modes. All host/device interaction is via DMA.

The **ath** driver encapsulates all IP and ARP traffic as 802.11 frames, however it can receive either 802.11 or 802.3 frames. Transmit speed and operating mode is selectable and depends on the specific chipset. AR5210-based devices support 802.11a operation with transmit speeds of 6 Mbps, 9 Mbps, 12 Mbps, 18 Mbps, 24 Mbps, 36 Mbps, 48 Mbps, and 54 Mbps. AR5211-based devices support 802.11a and 802.11b operation with transmit speeds as above for 802.11a operation and 1Mbps, 2Mbps, 5.5 Mbps and 11Mbps for 802.11b operation. AR5212-based devices support 802.11a, 802.11b, and 802.11g operation with transmit speeds appropriate to each. AR5416 and later class devices are capable of 802.11n operation. Most chips also support an Atheros Turbo Mode (TM) that operates in the 5GHz frequency range with 2x the transmit speeds. Some chips also support Turbo mode in the 2.4GHz range with 802.11g though this support is not presently available due to regulatory requirements. (Note that Turbo modes are, however, only interoperable with other Atheros-based devices.) AR5212-based and AR5416-based devices also support half- (10MHz) and quarter-width (5MHz) channels. The actual transmit speed used is dependent on signal quality and the "rate control" algorithm employed by the driver. All chips support WEP encryption. AR5212, AR5416 and later parts have hardware support for the AES-CCM, TKIP, and Michael cryptographic operations required for WPA. To enable encryption, use ifconfig(8) as shown below.

The driver supports **station**, **adhoc**, **adhoc-demo**, **hostap**, **mesh**, **wds**, and **monitor** mode operation. Multiple **hostap** virtual interfaces may be configured for simultaneous use on cards that use a 5212 or later part. When multiple interfaces are configured each may have a separate mac address that is formed by setting the U/L bits in the mac address assigned to the underlying device. Any number of **wds** virtual interfaces may be configured together with **hostap** interfaces. Multiple **station** interfaces may be operated together with **hostap** interfaces to construct a wireless repeater device. The driver also support **tdma** operation when compiled with **options IEEE80211_SUPPORT_TDMA** (which also enables the required 802.11 support). For more information on configuring this device, see ifconfig(8).

Devices supported by the **ath** driver come in Cardbus, ExpressCard, Mini-PCI and Mini-PCIe packages. Wireless cards in Cardbus and ExpressCard slots may be inserted and ejected on the fly.

## HARDWARE

The **ath** driver supports all Atheros Cardbus, ExpressCard, PCI and PCIe cards, except those that are based on the AR5005VL chipset.

## EXAMPLES

Join a specific BSS network with WEP encryption:

```
ifconfig wlan0 create wlandev ath0
ifconfig wlan0 inet 192.168.0.20 netmask 0xffffff00 ssid my_net \
        wepmode on wepkey 0x8736639624
```

Join/create an 802.11b IBSS network with network name "my_net":

```
ifconfig wlan0 create wlandev ath0 wlanmode adhoc
ifconfig wlan0 inet 192.168.0.22 netmask 0xffffff00 ssid my_net \
        mode 11b
```

Create an 802.11g host-based access point:

```
ifconfig wlan0 create wlandev ath0 wlanmode hostap
ifconfig wlan0 inet 192.168.0.10 netmask 0xffffff00 ssid my_ap \
        mode 11g
```

Create an 802.11a mesh station:

```
ifconfig wlan0 create wlandev ath0 wlanmode mesh
ifconfig wlan0 meshid my_mesh mode 11a inet 192.168.0.10/24
```

Create two virtual 802.11a host-based access points, one with WEP enabled and one with no security, and bridge them to the fxp0 (wired) device:

```
ifconfig wlan0 create wlandev ath0 wlanmode hostap \
        ssid paying-customers wepmode on wepkey 0x1234567890 \
        mode 11a up
ifconfig wlan1 create wlandev ath0 wlanmode hostap bssid \
        ssid freeloaders up
ifconfig bridge0 create addm wlan0 addm wlan1 addm fxp0 up
```

Create a master node in a two slot TDMA BSS configured to use 2.5 millisecond slots.

```
ifconfig wlan0 create wlandev ath0 wlanmode tdma \
        ssid tdma-test tmdaslot 0 tdmaslotlen 2500 \
        channel 36 up
```

## DIAGNOSTICS

**ath%d: unable to attach hardware; HAL status %u** The Atheros Hardware Access Layer was unable to configure the hardware as requested. The status code is explained in the HAL include file *sys/dev/ath/ath_hal/ah.h*.

**ath%d: failed to allocate descriptors: %d** The driver was unable to allocate contiguous memory for the transmit and receive descriptors. This usually indicates system memory is scarce and/or fragmented.

**ath%d: unable to setup a data xmit queue!** The request to the HAL to set up the transmit queue for normal data frames failed. This should not happen.

**ath%d: unable to setup a beacon xmit queue!** The request to the HAL to set up the transmit queue for 802.11 beacon frames failed. This should not happen.

**ath%d: 802.11 address: %s** The MAC address programmed in the EEPROM is displayed.

**ath%d: hardware error; resetting** An unrecoverable error in the hardware occurred. Errors of this sort include unrecoverable DMA errors. The driver will reset the hardware and continue.

**ath%d: rx FIFO overrun; resetting** The receive FIFO in the hardware overflowed before the data could be transferred to the host. This typically occurs because the hardware ran short of receive descriptors and had no place to transfer received data. The driver will reset the hardware and continue.

**ath%d: unable to reset hardware; hal status %u** The Atheros Hardware Access Layer was unable to reset

the hardware as requested. The status code is explained in the HAL include file *sys/dev/ath/ath_hal/ah.h*. This should not happen.

**ath%d: unable to start recv logic** The driver was unable to restart frame reception. This should not happen.

**ath%d: device timeout** A frame dispatched to the hardware for transmission did not complete in time. The driver will reset the hardware and continue. This should not happen.

**ath%d: bogus xmit rate 0x%x** An invalid transmit rate was specified for an outgoing frame. The frame is discarded. This should not happen.

**ath%d: ath_chan_set: unable to reset channel %u (%u MHz)** The Atheros Hardware Access Layer was unable to reset the hardware when switching channels during scanning. This should not happen.

**ath%d: failed to enable memory mapping** The driver was unable to enable memory-mapped I/O to the PCI device registers. This should not happen.

**ath%d: failed to enable bus mastering** The driver was unable to enable the device as a PCI bus master for doing DMA. This should not happen.

**ath%d: cannot map register space** The driver was unable to map the device registers into the host address space. This should not happen.

**ath%d: could not map interrupt** The driver was unable to allocate an IRQ for the device interrupt. This should not happen.

**ath%d: could not establish interrupt** The driver was unable to install the device interrupt handler. This should not happen.

## SEE ALSO

ath_hal(4), cardbus(4), intro(4), pcic(4), wlan(4), wlan_ccmp(4), wlan_tkip(4), wlan_wep(4), wlan_xauth(4), hostapd(8), ifconfig(8), wpa_supplicant(8)

## HISTORY

The **ath** device driver first appeared in FreeBSD 5.2.

## CAVEATS

Revision A1 of the D-LINK DWL-G520 and DWL-G650 are based on an Intersil PrismGT chip and are not supported by this driver.

## BUGS

The driver does supports optional station mode power-save operation.

The AR5210 can only do WEP in hardware; consequently hardware assisted WEP is disabled in order to allow software implementations of TKIP and CCMP to function. Hardware WEP can be re-enabled by modifying the driver.