About iw

iw is a new nl80211 based CLI configuration utility for wireless devices. It supports all new drivers that have been added to the kernel recently. The old tool iwconfig, which uses Wireless Extensions interface, is deprecated and it's strongly recommended to switch to iw and nl80211.

Like the rest of the Linux kernel, iw is still under development. Features are added 'as we go'. The only documentation for iw is this page and output from 'iw help'. Please help expand this page.

There is a page listing use cases with iwconfig and iw: replacing iwconfig.

Getting iw

Release tarballs of iw are available from http://kernel.org/pub/software/network/iw/.

Alternatively, you can download iw from git: http://git.kernel.org/?p=linux/kernel/git/jberg/iw.git.

Build requirements

- libnl >= libnl1
- libnl-dev >= libnl-dev-1
- pkg-config Using iw requires you to have libnl, the first working version is 1.0 pre8 as this release introduced genl, *Generic Netlink*, which nl80211 relies on. If your distribution's libnl is a wrong version then you'll have to download and compile libnl yourself for now (http://www.infradead.org/~tgr/libnl/).

Help

Just enter

```
iw help
```

on your command line and it will print out the commands it supports.

Getting device capabilities

Use the following to get device capabilities for all devices, such as band information (2.4 GHz, and 5 GHz), and 802.11n information:

```
iw list
```

Scanning

```
iw dev wlan0 scan
```

Listening to events

Just use

```
iw event
```

When debugging, it can be useful to see the auth/assoc/deauth/disassoc frames, use

```
iw event -f
```

and sometimes timing information is also useful:

```
iw event -t
```

Getting link status

To determine if you are connected to an AP or not and if you are the last TX rate used you can use the command below.

Example output when associated to a legacy (non-802.11n) AP:

```
iw dev wlan0 link
Connected to 04:21:b0:e8:c8:8b (on wlan0)
        SSID: attwifi
        freq: 2437
        RX: 2272 bytes (18 packets)
        TX: 232 bytes (3 packets)
        signal: -57 dBm
        tx bitrate: 36.0 MBit/s
```

Example output when associated to an 802.11n AP:

```
iw dev wlan0 link
Connected to 68:7f:74:3b:b0:01 (on wlan0)
        SSID: tesla-5g-bcm
        freq: 5745
        RX: 30206 bytes (201 packets)
        TX: 4084 bytes (23 packets)
        signal: -31 dBm
        tx bitrate: 300.0 MBit/s MCS 15 40Mhz short GI
```

Example output when not connected to an AP:

```
iw dev wlan0 link
Not connected.
```

This would happen if you are not connected to an AP. To connect to an AP you can use *iw connect* if the connection requires:

- No encryption
- Uses WEP for encryption If you need to connect to an AP with WPA or WPA2 encryption requirements then you must use wpa_supplicant.

Establishing a basic connection

You can use *iw* to connect to an AP directly if and only if the AP has:

- No encryption
- Uses WEP for encryption It however should be noted that if you disconnect from the AP, which can happen quite frequently on a busy environment, you will need to reissue the command. If you do not want to do this you can just use wpa_supplicant which will automatically try to reconnect you when you get disconnected.

If you do choose to deal with disconnects yourself you can use *iw connect* as follows.

To connect to an AP that has encryption disabled, where its SSID is *foo*:

```
iw wlan0 connect foo
```

Suppose you have two APs with the SSID *foo*, and you know the one you want to connect to is on the frequency 2432, you can specify the frequency to use:

```
iw wlan0 connect foo 2432
```

To connect to an AP that uses WEP, you can use:

```
iw wlan0 connect foo keys 0:abcde d:1:0011223344
```

Getting station statistics

To get station statistic information such as the amount of tx/rx bytes, the last TX bitrate (including MCS rate) you can do:

```
$ iw dev wlan1 station dump
Station 12:34:56:78:9a:bc (on wlan0)
        inactive time:  304 ms
        rx bytes:       18816
        rx packets:     75
        tx bytes:       5386
        tx packets:     21
        signal:         -29 dBm
        tx bitrate:     54.0 MBit/s
```

Getting station statistics against a peer

If you want to get specific statistics against a peer you station is communicating with you can use the following:

```
sudo iw dev wlan1 station get <peer-MAC-address>
```

In the case of a STA the above <peer-MAC-address> would be the MAC address of your AP.

Modifying transmit bitrates

iw supports modifying TX bitrates, both legacy and HT MCS rates. It does this by masking in the allowed bitrates, and also lets you clear the mask.

Modifying tx legacy bitrates

You can set preference for transmitting using only certain legacy bitrates. For example:

```
iw wlan0 set bitrates legacy-2.4 12 18 24
```

Here's how to enable what some folks call “Purge G” which disables 802.11b associations:

```
iw wlan0 set bitrates legacy-2.4 6 12 24
```

Modifying tx HT MCS bitrates

Setting preference for transmitting using MCS rates is supported by letting you specify the band and MCS rate. Note that whether or not the device actually listens to your petition will vary depending on the device driver and cooperation from the firmware. For example:

```
iw dev wlan0 set bitrates mcs-5 4
```

```
iw dev wlan0 set bitrates mcs-2.4 10
```

To clear all tx bitrates and set things back to normal:

```
iw dev wlan0 set bitrates mcs-2.4
iw dev wlan0 set bitrates mcs-5
```

Setting TX power

You can set the txpower by using either the device interface name of the respective phy.

```
iw dev <devname> set txpower <auto|fixed|limit> [<tx power in mBm>]
iw phy <phyname> set txpower <auto|fixed|limit> [<tx power in mBm>]
```

(Note that the value this commands takes is in *milli*bel-milliwatts (mBm) instead of the commonly used *deci*bel-milliwatts (dBm). <power in mBm> = 100 * <power in dBm>)

Power save

To enable power save by default you can use:

```
sudo iw dev wlan0 set power_save on
```

For mac80211 drivers this means Dynamic Power Save gets enabled.

To query the current power save settings you can use:

```
iw dev wlan0 get power_save
```

Adding interfaces with iw

There are several modes supported. The modes supported are:

- monitor
- managed [also station]
- wds
- mesh [also mp]
- ibss [also adhoc] To see a description of these please read our modes documentation.

For example to add a monitor interface:

```
iw phy phy0 interface add moni0 type monitor
```

where you can replace

```
monitor
```

by anything else and

```
moni0
```

by the interface name, and need to replace

```
phy0
```

by the PHY name for your hardware (usually phy0 will be correct unless you hotplugged or reloaded any modules.) If your udev is configured incorrectly, the newly created virtual interface may be renamed by it right away, use

```
ip link
```

to list all interfaces. To create a new managed mode interface you would use:

```
iw phy phy0 interface add wlan10 type managed
```

Note that the interface is automatically put into AP mode when using hostapd.

Modifying monitor interface flags

You can customize the type of monitor interface you create. This can be very useful for debugging purposes on end user systems. For example, suppose you want to help a user. You can take advantage of the fact that a monitor interface in mac80211 uses radiotap to pass up to userspace additional data. Say we want to help a user fish out data without affecting the device's performance by setting it to a full monitor interface. A monitor interface with no additional monitor flags can be created as follows:

```
iw dev wlan0 interface add fish0 type monitor flags none
```

You can then request the user to use tcpdump on a session:

```
tcpdump -i fish0 -s 65000 -p -U -w  /tmp/fishing.dump
```

The nice thing about these type of alternative monitor interfaces is you can further extend radiotap even with vendor extensions to add more data to radiotap to help debug device specific features.

Keep in mind this requires drivers to honor mac80211's flag requests strictly, so drivers like ath5k and ath9k which still enable flags based on operation mode need to be fixed to take advantage of this.

Monitor flags possible

The following are flags you can specify:

- none
- fcsfail
- plcpfail
- control
- otherbss
- cook
- active

Deleting interfaces with iw

Use

```
iw dev moni0 del
```

Virtual vif support

There is a dedicated section for virtual vif support, see the iw vif page.

Updating your regulatory domain

The command line is:

```
iw reg set alpha2
```

Where “alpha2” is the ISO/IEC 3166 alpha2 country code. The information used and set comes from our regulatory infrastructure.

You can also use the latest wpa_supplicant (as of 0.6.7) now to change your regulatory domain, to do so just add a “country=US” entry into your configuration for example.

Creating and inspecting Mesh Point interfaces with iw

You may add a mesh interface to drivers that support Mesh Point operation. Mesh Point interfaces have a mesh_id parameter which may be up to 32 bytes long. For example, to add an interface “mesh0” to device phy0 with mesh_id “mymesh”,

```
iw phy phy0 interface add mesh0 type mp mesh_id mymesh
```

Mesh Point interfaces, by default, are configured on Channel 1. Mesh Point operation begins when the interface is brought up. In the default configuration, Mesh Point interfaces will automatically detect and attempt to create Peer Links with other Mesh Points (peers) having the same mesh ID. Use the station list and station statistics to see the peer list and Peer Link status.

After sending traffic (ex: pinging another mesh node), you may wish to see a list of Mesh Paths:

```
iw dev mesh0 mpath dump
```

Please see the open80211s.org HOWTO for further details on Mesh Point related commands and their output, as well as more examples. iw also provides commands for advanced Mesh Point configuration. These are documented in the Advanced Tinkering section of the open80211s HOWTO.

Setting up a WDS peer

WDS mode is a non-standard extension to the IEEE 802.11 standard to allow transparent Ethernet bridging on the station and to implement seamingless hand-over for wireless clients roaming between different access points. Due to its non-standard nature, WDS is often implemented differently in wireless drivers and vendor firmwares making them incompatible with each other. In order to use WDS, one should use the same hardware and software on all deployed wireless devices to maintain compatibility.

To create a WDS peer you will first need to create an interface of WDS type, and then set the peer:

```
iw phy phy0 interface add wds0 type wds
iw dev wds0 set peer <MAC address>
```

In order for this to work the driver must implement the cfg80211 callback set_wds_peer(). mac80211 implements this callback, so the respective mac80211 driver would just need to support WDS type interfaces. What WDS will do is replace the first address on the 802.11 header with the peer address when TXing frames. Instead of using WDS though you may want to consider using 4-address mode described below if you have control over the software running on the AP and respective clients/peers connected.

Using 4-address for AP and client mode

In some situations it might be useful to run a network with an Access Point and multiple clients, but with each client bridged to a network behind it. For this to work, both the client and the AP need to transmit 4-address frames, containing both source and destination MAC addresses. 4-address mode is how OpenWrt supports WDS mode for mac80211 drivers, that is if you enable *wds* option on your OpenWrt OpenWrt wireless

WDS mode for mac80211 drivers, that is if you enable *wds* option on your OpenWrt OpenWrt wireless configuration you will end up using 4-address mode. 4-address mode is not compatible with other WDS implementations, ie, you'll need all endpoints using this mode in order for WDS to work appropriately.

Linux wireless has support for 4-address mode for AP and STAs but each driver needs to define this capability explicitly. All mac80211 drivers support 4-address mode if AP or STA modes of operation are supported respectively.

On the AP side you can enable 4-address frames for individual clients by isolating them in separate AP VLANs which are configured in 4-address mode. Such an AP VLAN will be limited to one client only, and this client will be used as the destination for all traffic on its interface, regardless of the destination MAC address in the packet headers. The advantage of this mode compared to regular WDS mode is that it's easier to configure and does not require a static list of peer MAC addresses on any side. 4-address mode is incompatible with WDS.

To enable 4-address mode when creating an interface you should add *4addr on*, for example:

```
iw phy phy0 interface add sta0 type managed 4addr on
```

In this mode, the new interface can be in a bridge – if it is then you need to use the

```
-b
```

flag to wpa_supplicant to make it listen for EAPOL on the bridge instead of the interface itself.

In hostapd you can enable this with the flag on hostapd.conf:

```
wds_sta=1
```

Please note 4-address mode is currently broken on 3.9 because of commit 576eb62598f10c8c7fd75703fe89010cdcfff596 , this topic is currently being addressed on the mailing lists for a resolution.

Creating packet coalesce rules

In most cases, host that receives IPv4 and IPv6 multicast/broadcast packets does not do anything with these packets. Therefore the reception of these unwanted packets causes unnecessary processing and power consumption.

Packet coalesce feature helps to reduce number of receive interrupts to host by buffering these packets in firmware/hardware for some predefined time. Receive interrupt will be generated when one of the following events occur.

- Expiration of hardware timer whose expiration time is set to maximum coalescing delay of matching coalesce rule.
- Coalescing buffer in hardware reaches it's limit.
- Packet doesn't match any of the configured coalesce rules. To view coalesce configuration support information, you can use 'iw phy0 info'. Here is an example output:

```
Coalesce support:
         * Maximum 8 coalesce rules supported
         * Each rule contains upto 4 patterns of 1-4 bytes,
           maximum packet offset 50 bytes
         * Maximum supported coalescing delay 100 msecs
```

You need to configure following parameters for creating a coalesce rule.

- Maximum coalescing delay
- List of packet patterns which needs to be matched
- Condition for coalescence. pattern 'match' or 'no match' Multiple such rules can be provided through a configuration file.

To enable coalesce feature using rules listed in coalesce.conf file, you can use:

```
iw phy phy0 enable coalesce.conf
```

Where coalesce.conf contains:

```
delay=25
condition=0
patterns=8+34:xx:ad:22,10+23:45:67,59:33:xx:25,ff:ff:ff:ff
delay=40
condition=1
patterns=12+00:xx:12,23:45:67,46:61:xx:50
```

To display current coalesce configuration, you can use:

```
$ iw phy phy0 coalesce show
Coalesce is enabled:
Rule - max coalescing delay: 25msec condition:match
 * packet offset: 8 pattern: 34:--:ad:22
 * packet offset: 10 pattern: 23:45:67
 * packet offset: 0 pattern: 59:33:--:25
 * packet offset: 0 pattern: ff:ff:ff:ff
Rule - max coalescing delay: 40msec condition:not match
 * packet offset: 12 pattern: 00:--:12
 * packet offset: 0 pattern: 23:45:67
 * packet offset: 0 pattern: 46:61:--:50
```

To disable coalesce feature, you can use:

To disable coalesce feature, you can use:

```
iw phy phy0 coalesce disable
```

'iw display' output when coalesce is not configured:

```
$ iw phy phy0 coalesce show
Coalesce is disabled.
```