Wireless network configuration

From ArchWiki

Configuring wireless is a two-part process; the first part is to identify and ensure the correct driver for your wireless device is installed (they are available on the installation media, but often have to be installed explicitly), and to configure the interface. The second is choosing a method of managing wireless connections. This article covers both parts, and provides additional links to wireless management tools.

Related articles

Network configuration

Software access point

Ad-hoc networking

Internet sharing

Contents

Device driver

The default Arch Linux kernel is *modular*, meaning many of the drivers for machine hardware reside on the hard drive and are available as modules. At boot, udev takes an inventory of your hardware and loads appropriate modules (drivers) for your corresponding hardware, which will in turn allow creation of a network *interface*.

Some wireless chipsets also require firmware, in addition to a corresponding driver. Many firmware images are provided by the `linux-firmware (https://www.archlinux.org/packages/?name=linux-firmware)` package which is installed by default, however, proprietary firmware images are not included and have to be installed separately. This is described in #Installing driver/firmware.

Note: Udev is not perfect. If the proper module is not loaded by udev on boot, simply load it manually. Note also that udev may occasionally load more than one driver for a device, and the resulting conflict will prevent successful configuration. Make sure to blacklist the unwanted module.

Tip: Though not strictly required, it's a good idea to first install user-space tools

mentioned in #Manual setup, especially when some problem should appear.

Check the driver status

To check if the driver for your card has been loaded, check the output of the `lspci -k` or `lsusb -v` command, depending on if the card is connected by PCI(e) or USB. You should see that some kernel driver is in use, for example:

```
$ lspci -k
```

```
06:00.0 Network controller: Intel Corporation WiFi Link 5100
        Subsystem: Intel Corporation WiFi Link 5100 AGN
        Kernel driver in use: iwlwifi
        Kernel modules: iwlwifi
```

Note: If the card is a USB device, running `dmesg | grep usbcore` should give something like `usbcore: registered new interface driver rtl8187` as output.

Also check the output of `ip link` command to see if a wireless interface (usually it starts with the letter "w", e.g. `wlp2s1`) was created. Then bring the interface up with `ip link set *interface* up`. For example, assuming the interface is `wlan0`:

```
# ip link set wlan0 up
```

If you get this error message: `SIOCSIFFLAGS: No such file or directory`, it most certainly means that your wireless chipset requires a firmware to function.

Check kernel messages for firmware being loaded:

```
$ dmesg | grep firmware
```

```
[   7.148259] iwlwifi 0000:02:00.0: loaded firmware version 39.30.4.1 build 35138 op_mode iwldvm
```

If there is no relevant output, check the messages for the full output for the module you identified earlier (`iwlwifi` in this example) to identify the relevant message or further issues:

```
$ dmesg | grep iwlwifi
```

```
[   12.342694] iwlwifi 0000:02:00.0: irq 44 for MSI/MSI-X
[   12.353466] iwlwifi 0000:02:00.0: loaded firmware version 39.31.5.1 build 35138 op_mode iwldvm
[   12.430317] iwlwifi 0000:02:00.0: CONFIG_IWLWIFI_DEBUG disabled
...
[   12.430341] iwlwifi 0000:02:00.0: Detected Intel(R) Corporation WiFi Link 5100 AGN, REV=0x6B
```

If the kernel module is successfully loaded and the interface is up, you can skip the next section.

Installing driver/firmware

Check the following lists to discover if your card is supported:

- The Ubuntu Wiki (https://help.ubuntu.com/community/WifiDocs/WirelessCardsSupported) has a good list of wireless cards and whether or not they are supported either in the Linux kernel or by a user-space driver (includes driver name).
- Linux Wireless Support (http://linux-wless.passys.nl/) and The Linux Questions' Hardware Compatibility List (http://www.linuxquestions.org/hcl/index.php?cat=10) (HCL) also have a good database of kernel-friendly hardware.
- The kernel page (http://wireless.kernel.org/en/users/Devices) additionally has a matrix of supported hardware.

Note that some vendors ship products that may contain different chip sets, even if the product identifier is the same. Only the usb-id (for USB devices) or pci-id (for PCI devices) is authoritative.

If your wireless card is listed above, follow the #Troubleshooting drivers and firmware subsection of this page, which contains information about installing drivers and firmware of some specific wireless cards. Then check the driver status again.

If your wireless card is not listed above, it is likely supported only under Windows (some Broadcom, 3com, etc). For these, you can try to use #ndiswrapper.

Wireless management

Assuming that your drivers are installed and working properly, you will need to choose a method of managing your wireless connections. The following subsections will help you decide.

Procedure and tools required will depend on several factors:

- The desired nature of configuration management; from a completely manual command line procedure to an automated solution with graphical front-ends.
- The encryption type (or lack thereof) which protects the wireless network.
- The need for network profiles, if the computer will frequently change networks (such as a laptop).

Tip:

- Whatever is your choice, **you should try to connect using the manual method first**. This will help you understand the different steps that are required and troubleshoot possible problems.
- If possible (e.g. if you manage your Wi-Fi access point), try connecting with no encryption, to check that everything works. Then try using encryption, either WEP (simple to configure, but crackable in a matter of seconds), WPA or WPA2.

The following table shows the different methods that can be used to activate and manage a wireless connection, depending on the encryption and management types, and the various tools that are required. Although there may be other possibilities, these are the most frequently used:

Management method	Interface activation	Wireless connection management (/=alternatives)	Assigning IP address (/=alternatives)
Manually managed, with no or WEP encryption	ip	iw (https://www.archlinux.org/packages/?name=iw) / iwconfig (https://www.archlinux.org/packages/?name=wireless_tools)	ip / dhcpcd / dhclient (https://www.archlinux.org/packages/?name=dhclient)
Manually managed, with WPA or WPA2 PSK encryption	ip	iw (https://www.archlinux.org/packages/?name=iw) / iwconfig (https://www.archlinux.org/packages/?name=wireless_tools) + wpa_supplicant	ip / dhcpcd / dhclient (https://www.archlinux.org/packages/?name=dhclient)
Automatically managed, with network profiles support	netctl, Wicd, NetworkManager, etc. These tools pull in the required dependencies from the list of packages in the manual method.		

Manual setup

Just like other network interfaces, the wireless ones are controlled with *ip* from the iproute2 (https://www.archlinux.org/packages/?name=iproute2) package.

You will need to install a basic set of tools for managing the wireless connection. Either:

- iw (https://www.archlinux.org/packages/?name=iw) - only supports the nl80211 (netlink) standard. It does not support the older WEXT (Wireless EXTentions) standard. If *iw* does not see your card, this may be the reason.

or

- wireless_tools (https://www.archlinux.org/packages/?name=wireless_tools) - currently deprecated, but still widely supported. Use this for modules using the WEXT standard.

For WPA/WPA2 encryption, you will also need:

- wpa_supplicant (https://www.archlinux.org/packages/?name=wpa_supplicant) - works with both WEXT and nl80211.

The table below gives an overview of comparable commands for *iw* and *wireless_tools* (see iw replaces iwconfig (http://wireless.kernel.org/en/users/Documentation/iw/replace-iwconfig) for more examples). These user-space tools work extremely well and allow complete manual control of wireless connection.

Note:

- Examples in this section assume that your wireless device is `wlan0` and that you are connecting to *your_essid* wifi access point. Replace both accordingly.
- Note that most of the commands have to be executed with root permissions. Executed with normal user rights, some of the commands (e.g. *iwlist*), will exit without error but not produce the correct output either, which can be confusing.

iw command	*wireless_tools* command	Description
iw dev wlan0 link	iwconfig wlan0	Getting link status.
iw dev wlan0 scan	iwlist wlan0 scan	Scanning for available access points.
iw dev wlan0 set type ibss	iwconfig wlan0 mode ad-hoc	Setting the operation mode to *ad-hoc*.
iw dev wlan0 connect *your_essid*	iwconfig wlan0 essid *your_essid*	Connecting to open network.
iw dev wlan0 connect *your_essid* 2432	iwconfig wlan0 essid *your_essid* freq 2432M	Connecting to open network specifying channel.
iw dev wlan0 connect *your_essid* key 0:*your_key*	iwconfig wlan0 essid *your_essid* key *your_key*	Connecting to WEP encrypted network using hexadecimal key.
iw dev wlan0 connect *your_essid* key 0:*your_key*	iwconfig wlan0 essid *your_essid* key s:*your_key*	Connecting to WEP encrypted network using ASCII key.
iw dev wlan0 set power_save on	iwconfig wlan0 power on	Enabling power save.

Note: Depending on your hardware and encryption type, some of these steps may not be necessary. Some cards are known to require interface activation and/or access point scanning before being associated to an access point and being given an IP address. Some experimentation may be required. For instance, WPA/WPA2 users may try to directly activate their wireless network from step #Association.

Getting some useful information

Tip: See official documentation (http://wireless.kernel.org/en/users/Documentation/iw) of the *iw* tool for more examples.

- First you need to find the name of wireless interface. You can do it with following command:

```
$ iw dev
```

```
phy#0
        Interface wlan0
                ifindex 3
                wdev 0x1
                addr 12:34:56:78:9a:bc
                type managed
                channel 1 (2412 MHz), width: 40 MHz, center1: 2422 MHz
```

- To check link status, use following command. Example output when not connected to an AP:

```
$ iw dev wlan0 link
```

```
Not connected.
```

When connected to an AP, you will see something like:

```
$ iw dev wlan0 link
```

```
Connected to 12:34:56:78:9a:bc (on wlan0)
        SSID: MyESSID
        freq: 2412
        RX: 33016518 bytes (152703 packets)
        TX: 2024638 bytes (11477 packets)
        signal: -53 dBm
        tx bitrate: 150.0 MBit/s MCS 7 40MHz short GI

        bss flags:      short-preamble short-slot-time
        dtim period:    1
        beacon int:     100
```

- You can get statistic information, such as the amount of tx/rx bytes, signal strength etc., with following command:

```
$ iw dev wlan0 station dump
```

```
Station 12:34:56:78:9a:bc (on wlan0)
        inactive time:  1450 ms
        rx bytes:       24668671
        rx packets:     114373
        tx bytes:       1606991
        tx packets:     8557
        tx retries:     623
        tx failed:      1425
        signal:         -52 dBm
        signal avg:     -53 dBm
        tx bitrate:     150.0 MBit/s MCS 7 40MHz short GI
        authorized:     yes
        authenticated:  yes
        preamble:       long
        WMM/WME:        yes
        MFP:            no
        TDLS peer:      no
```

Interface activation

> **Tip:** Usually this step is not required

Some cards require that the kernel interface be activated before you can use *iw* or *wireless_tools*:

```
# ip link set wlan0 up
```

> **Note:** If you get errors like `RTNETLINK answers: Operation not possible due to RF-kill`, make sure that hardware switch is *on*. See #Rfkill caveat for details.

To verify that the interface is up, inspect the output of the following command:

```
# ip link show wlan0
```

```
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state DOWN mode DORMANT group default qlen 1000
```

```
    link/ether 12:34:56:78:9a:bc brd ff:ff:ff:ff:ff:ff
```

The `UP` in `<BROADCAST,MULTICAST,UP,LOWER_UP>` is what indicates the interface is up, not the later `state DOWN`.

Access point discovery

See what access points are available:

```
# iw dev wlan0 scan | less
```

Note: If it displays `Interface doesn't support scanning`, then you probably forgot to install the firmware. In some cases this message is also displayed when not running *iw* as root.

Tip: Depending on your location, you might need to set the correct regulatory domain in order to see all available networks.

The important points to check:

- **SSID:** the name of the network.
- **Signal:** is reported in a wireless power ratio in dbm (e.g. from -100 to 0). The closer the negative value gets to zero, the better the signal. Observing the reported power on a good quality link and a bad one should give an idea about the individual range.
- **Security:** it is not reported directly, check the line starting with `capability`. If there is `Privacy`, for example `capability: ESS Privacy ShortSlotTime (0x0411)`, then the network is protected somehow.
  - If you see an `RSN` information block, then the network is protected by Robust Security Network protocol, also known as WPA2.
  - If you see an `WPA` information block, then the network is protected by Wi-Fi Protected Access protocol.
  - In the `RSN` and `WPA` blocks you may find the following information:
    - **Group cipher:** value in TKIP, CCMP, both, others.
    - **Pairwise ciphers:** value in TKIP, CCMP, both, others. Not necessarily the same value than Group cipher.
    - **Authentication suites:** value in PSK, 802.1x, others. For home router, you'll usually find PSK (*i.e.* passphrase). In universities, you are more likely to find 802.1x suite which requires login and password. Then you will need to know which key management is in use (e.g. EAP), and what encapsulation it uses (e.g. PEAP). Find more details at Wikipedia:Authentication protocol and the sub-articles.
  - If you do not see neither `RSN` nor `WPA` blocks but there is `Privacy`, then WEP is used.

Operating mode

Tip: This step is optional, but may be required

At this step you might need to set the proper operating mode of the wireless card. More specifically, if you are going to connect an ad-hoc network, you need to set the operating mode to `ibss`:

```
# iw dev wlan0 set type ibss
```

Note: Changing the operating mode on some cards might require the wireless interface to be *down* (`ip link set wlan0 down`).

Association

Depending on the encryption, you need to associate your wireless device with the access point to use and pass the encryption key:

- **No encryption**

```
# iw dev wlan0 connect "your_essid"
```

- **WEP**
  - using a hexadecimal or ASCII key (the format is distinguished automatically, because a WEP key has a fixed length):

```
# iw dev wlan0 connect "your_essid" key 0:your_key
```

  - using a hexadecimal or ASCII key, specifying the third set up key as default (keys are counted from zero, four are possible):

```
# iw dev wlan0 connect "your_essid" key d:2:your_key
```

- **WPA/WPA2** According to what you got from #Access point discovery, issue this command:

```
# wpa_supplicant -D nl80211,wext -i wlan0 -c <(wpa_passphrase "your_SSID" "your_key")
```

If this does not work, you may need to adjust the options. If connected successfully, continue in a new terminal (or quit `wpa_supplicant` with `Ctrl+c` and add the `-B` switch to the above command to run it in the background). WPA supplicant contains more information on options and on how to create a permanent configuration file for the wireless access point.

Regardless of the method used, you can check if you have associated successfully:

```
# iw dev wlan0 link
```

Getting an IP address

Note: See Network configuration#Configure the IP address for more examples. This part is identical.

Finally, provide an IP address to the network interface. Simple examples are:

```
# dhcpcd wlan0
```

for DHCP, or

```
# ip addr add 192.168.0.2/24 dev wlan0
# ip route add default via 192.168.0.1
```

for static IP addressing.

> **Tip:** dhcpcd contains a hook (enabled by default) to automatically launch WPA supplicant on wireless interfaces. In most cases, you do not need to create any custom service, just enable `dhcpcd@*interface*.service` .

Custom startup scripts/services

Although the manual configuration method will help troubleshoot wireless problems, you will have to re-type every command each time you reboot. You can also quickly write a *systemd* service to automate the whole process, which is still a quite convenient way of managing network connection while keeping full control over your configuration. You can find some examples in this section. If you prefer to use scripts to handle the commands, see the network configuration article for an example how to source them.

Manual wireless connection at boot using systemd and dhcpcd

This example uses systemd for start up, WPA supplicant for connecting, and `dhcpcd (https://www.archlinux.org/packages/?name=dhcpcd)` for assigning an IP address.

> **Note:** Make sure that `wpa_supplicant (https://www.archlinux.org/packages/?name=wpa_supplicant)` is installed and create `/etc/wpa_supplicant/wpa_supplicant.conf` . See WPA supplicant for details.

Create a *systemd* unit, e.g `/etc/systemd/system/network-wireless@.service` :

/etc/systemd/system/network-wireless@.service

```
[Unit]
Description=Wireless network connectivity (%i)
Wants=network.target
Before=network.target
BindsTo=sys-subsystem-net-devices-%i.device
After=sys-subsystem-net-devices-%i.device

[Service]
Type=oneshot
RemainAfterExit=yes

ExecStart=/usr/bin/ip link set dev %i up
ExecStart=/usr/bin/wpa_supplicant -B -i %i -c /etc/wpa_supplicant/wpa_supplicant.conf
ExecStart=/usr/bin/dhcpcd %i

ExecStop=/usr/bin/ip link set dev %i down

[Install]
WantedBy=multi-user.target
```

Start and/or enable the unit as described in systemd#Using units, remember to pass the name of the interface:

```
# systemctl enable network-wireless@wlan0.service
# systemctl start network-wireless@wlan0.service
```

Systemd with wpa_supplicant and static IP

Note: Make sure that wpa_supplicant (https://www.archlinux.org/packages/?name=wpa_supplicant) is installed and create a custom /etc/wpa_supplicant/wpa_supplicant.conf . See WPA supplicant for details.

First create configuration file for the systemd service, replace *interface* with the proper interface name for reference:

/etc/conf.d/network-wireless-*interface*

```
address=192.168.0.10
netmask=24
broadcast=192.168.0.255
gateway=192.168.0.1
```

Create a *systemd* unit file:

/etc/systemd/system/network-wireless@.service

```
[Unit]
Description=Wireless network connectivity (%i)
Wants=network.target
Before=network.target
BindsTo=sys-subsystem-net-devices-%i.device
After=sys-subsystem-net-devices-%i.device

[Service]
Type=oneshot
RemainAfterExit=yes
EnvironmentFile=/etc/conf.d/network-wireless-%i

ExecStart=/usr/bin/ip link set dev %i up
ExecStart=/usr/bin/wpa_supplicant -B -i %i -c /etc/wpa_supplicant/wpa_supplicant.conf
ExecStart=/usr/bin/ip addr add ${address}/${netmask} broadcast ${broadcast} dev %i
ExecStart=/usr/bin/ip route add default via ${gateway}

ExecStop=/usr/bin/ip addr flush dev %i
ExecStop=/usr/bin/ip link set dev %i down

[Install]
WantedBy=multi-user.target
```

Enable and start the unit network-wireless@*interface* , passing your name of the interface.

Automatic setup

There are many solutions to choose from, but remember that all of them are mutually exclusive; you should not run two daemons simultaneously. The following table compares the different connection managers, additional notes are in subsections below.

Connection manager	Network profiles support	Roaming (auto connect dropped or changed location)	PPP support (e.g. 3G modem)	Official GUI	Console tools
Connman	Yes	Yes	Yes	No	`connmanctl`
netctl	Yes	Yes	Yes	No	`netctl` , `wifi-menu`
NetworkManager	Yes	Yes	Yes	Yes	`nmcli`
Wicd	Yes	Yes	No	Yes	`wicd-curses`

Connman

ConnMan is an alternative to *NetworkManager* and *Wicd*, designed to be light on resources making it ideal for netbooks, and other mobile devices. It is modular in design takes advandage of the dbus API and provides proper abstraction on top of *wpa_supplicant*.

See Connman.

netctl

netctl is a replacement for *netcfg* designed to work with *systemd*. It uses a profile based setup and is capable of detection and connection to a wide range of network types. This is no harder than using graphical tools.

See netctl.

Wicd

Wicd is a network manager that can handle both wireless and wired connections. It is written in Python and Gtk with fewer dependencies than *NetworkManager*, making it an ideal solution for lightweight desktop users.

See Wicd.

> **Note:** wicd may cause excessive dropped connections with some drivers, while NetworkManager might work better.

NetworkManager

NetworkManager is an advanced network management tool that is enabled by default in most popular GNU/Linux distributions. In addition to managing wired connections, *NetworkManager* provides worry-free wireless roaming with an easy-to-use GUI program for selecting your desired network.

See NetworkManager.

> **Note:** GNOME's `network-manager-applet (https://www.archlinux.org/packages/?name=network-manager-applet)` also works under Xfce if you install `xfce4-xfapplet-plugin (https://aur.archlinux.org/packages/xfce4-xfapplet-plugin/)` (available in the AUR) first. Additionally, there are applets available for KDE.

WiFi Radar

WiFi Radar is a Python/PyGTK2 utility for managing wireless (and **only** wireless) profiles. It enables you to scan for available networks and create profiles for your preferred networks.

See Wifi Radar.

Troubleshooting

This section contains general troubleshooting tips, not strictly related to problems with drivers or firmware. For such topics, see next section #Troubleshooting drivers and firmware.

Rfkill caveat

Many laptops have a hardware button (or switch) to turn off wireless card, however, the card can also be blocked by kernel. This can be handled by `rfkill (https://www.archlinux.org/packages/?name=rfkill)`. Use *rfkill* to show the current status:

```
# rfkill list
```

```
0: phy0: Wireless LAN
        Soft blocked: yes
        Hard blocked: yes
```

If the card is *hard-blocked*, use the hardware button (switch) to unblock it. If the card is not *hard-blocked* but *soft-blocked*, use the following command:

```
# rfkill unblock wifi
```

Note: It is possible that the card will go from *hard-blocked* and *soft-unblocked* state into *hard-unblocked* and *soft-blocked* state by pressing the hardware button (i.e. the *soft-blocked* bit is just switched no matter what). This can be adjusted by tuning some options of the `rfkill` kernel module.

More info: http://askubuntu.com/questions/62166/siocsifflags-operation-not-possible-due-to-rf-kill

Respecting the regulatory domain

The regulatory domain (http://en.wikipedia.org/wiki/IEEE_802.11#Regulatory_domains_and_legal_compliance), or "regdomain", is used to reconfigure wireless drivers to make sure that wireless hardware usage complies with local laws set by the FCC, ETSI and other organizations. Regdomains use ISO 3166-1 alpha-2 country codes (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2). For example, the regdomain of the United States would be "US", China would be "CN", etc.

Regdomains affect the availability of wireless channels. In the 2.4GHz band, the allowed channels are 1-11 for the US, 1-14 for Japan, and 1-13 for most of the rest of the world. In the 5GHz band, the rules for allowed channels are much more complex. In either case, consult this list of WLAN channels (https://en.wikipedia.org/wiki/List_of_WLAN_channels) for more detailed information.

Regdomains also affect the limit on the effective isotropic radiated power (EIRP) (https://en.wikipedia.org/wiki/Equivalent_isotropically_radiated_power) from wireless devices. This is derived from transmit power/"tx power", and is measured in dBm/mBm (1dBm=100mBm) or mW (log scale) (https://en.wikipedia.org/wiki/DBm). In the 2.4GHz band, the maximum is 30dBm in the US and Canada, 20dBm in most of Europe, and 20dB-30dBm for the rest of the world. In the 5GHz

band, maximums are usually lower. Consult the wireless-regdb (http://git.kernel.org/cgit/linux/kernel/git/linville/wireless-regdb.git/tree/db.txt) for more detailed information (EIRP dBm values are in the second set of brackets for each line).

Misconfiguring the regdomain can be useful - for example, by allowing use of an unused channel when other channels are crowded, or by allowing an increase in tx power to widen transmitter range. However, **this is not recommended** as it could break local laws and cause interference with other radio devices.

To configure the regdomain, install `crda` (https://www.archlinux.org/packages/?name=crda) and `wireless-regdb` (https://www.archlinux.org/packages/?name=wireless-regdb) and reboot (to reload the `cfg80211` module and all related drivers). Check the boot log to make sure that CRDA is being called by `cfg80211`:

```
$ dmesg | grep cfg80211
```

The current regdomain can be set to the United States with:

```
# iw reg set US
```

And queried with:

```
$ iw reg get
```

> **Note:** Your device may be set to country "00", which is the "world regulatory domain" and contains generic settings. If this cannot be unset, CRDA may be misconfigured.

However, setting the regdomain may not alter your settings. Some devices have a regdomain set in firmware/EEPROM, which dictates the limits of the device, meaning that setting regdomain in software can only increase restrictions (http://wiki.openwrt.org/doc/howto/wireless.utilities#iw), not decrease them. For example, a CN device could be set in software to the US regdomain, but because CN has an EIRP maximum of 20dBm, the device will not be able to transmit at the US maximum of 30dBm.

For example, to see if the regdomain is being set in firmware for an Atheros device:

```
$ dmesg | grep ath:
```

For other chipsets, it may help to search for "EEPROM", "regdomain", or simply the name of the device driver.

To see if your regdomain change has been successful, and to query the number of available channels and their allowed transmit power:

```
$ iw list | grep -A 15 Frequencies:
```

A more permanent configuration of the regdomain can be achieved through editing `/etc/conf.d/wireless-regdom` and uncommenting the appropriate domain. `wpa_supplicant` can also use a regdomain in the `country=` line of `/etc/wpa_supplicant.conf`.

It is also possible to configure the cfg80211 (http://wireless.kernel.org/en/developers/Documentation/cfg80211) kernel module to use a specific regdomain by adding, for example, `options cfg80211 ieee80211_regdom=EU` to `/etc/modprobe.d/modprobe.conf`. However, this is part of the old regulatory implementation (http://wireless.kernel.org/en/developers/Regulatory#The_ieee80211_regdom_module_parameter).

For further information, read the wireless.kernel.org regulatory documentation (http://wireless.kernel.org/en/developers/Regulatory/).

Observing Logs

A good first measure to troubleshoot is to analyze the system's logfiles first. In order not to manually parse through them all, it can help to open a second terminal/console window and watch the kernels messages with

```
$ dmesg -w
```

while performing the action, e.g. the wireless association attempt.

When using a tool for network management, the same can be done for systemd with

```
# journalctl -f
```

Frequently a wireless error is accompanied by a deauthentication with a particular reason code, for example:

```
wlan0: deauthenticating from XX:XX:XX:XX:XX:XX by local choice (reason=3)
```

Looking up the reason code (http://www.aboutcher.co.uk/2012/07/linux-wifi-deauthenticated-reason-codes/) might give a first hint.

The individual tools used in this article further provide options for more detailed debugging output, which can be used in a second step of the analysis, if required.

Failure to Connect

First try to connect to the network with no authentication

Other troubling-shooting in no particular order:

Try disabling n mode within your router/Access Point settings.

Change the the channel of in your Access Point

Use a channel width of 20hz instead of 40hz

Power saving

See Power saving#Network interfaces.

Failed to get IP address

- If getting an IP address repeatedly fails using the default `dhcpcd`

(https://www.archlinux.org/packages/?name=dhcpcd) client, try installing and using dhclient (https://www.archlinux.org/packages/?name=dhclient) instead. Do not forget to select *dhclient* as the primary DHCP client in your connection manager!

- If you can get an IP address for a wired interface and not for a wireless interface, try disabling the wireless card's power saving features:

```
# iw dev wlan0 set power_save off
```

- If you get a timeout error due to a *waiting for carrier* problem, then you might have to set the channel mode to `auto` for the specific device:

```
# iwconfig wlan0 channel auto
```

Before changing the channel to auto, make sure your wireless interface is down. After it has successfully changed it, you can bring the interface up again and continue from there.

Connection always times out

The driver may suffer from a lot of tx excessive retries and invalid misc errors for some unknown reason, resulting in a lot of packet loss and keep disconnecting, sometimes instantly. Following tips might be helpful.

Lowering the rate

Try setting lower rate, for example 5.5M:

```
# iwconfig wlan0 rate 5.5M auto
```

Fixed option should ensure that the driver does not change the rate on its own, thus making the connection a bit more stable:

```
# iwconfig wlan0 rate 5.5M fixed
```

Lowering the txpower

You can try lowering the transmit power as well. This may save power as well:

```
# iwconfig wlan0 txpower 5
```

Valid settings are from `0` to `20`, `auto` and `off`.

Setting rts and fragmentation thresholds

Default iwconfig options have rts and fragmentation thresholds off. These options are particularly useful when there are many adjacent APs or in a noisy environment.

The minimum value for fragmentation value is 256 and maximum is 2346. In many windows drivers the maximum is the default value:

```
# iwconfig wlan0 frag 2346
```

For rts minimum is 0, maximum is 2347. Once again windows drivers often use maximum as the default:

```
# iwconfig wlan0 rts 2347
```

Random disconnections

Cause #1

If dmesg says `wlan0: deauthenticating from MAC by local choice (reason=3)` and you lose your Wi-Fi connection, it is likely that you have a bit too aggressive power-saving on your Wi-Fi card[1] (http://us.generation-nt.com/answer/gentoo-user-wireless-deauthenticating-by-local-choice-help-204640041.html). Try disabling the wireless card's power-saving features:

```
# iwconfig wlan0 power off
```

See Power saving for tips on how to make it permanent (just specify `off` instead of `on`).

If your card does not support `iwconfig wlan0 power off`, check the **BIOS** for power management options. Disabling PCI-Express power management in the BIOS of a Lenovo W520 resolved this issue.

Cause #2

If you are experiencing frequent disconnections and dmesg shows messages such as

```
ieee80211 phy0: wlan0: No probe response from AP xx:xx:xx:xx:xx:xx after 500ms, disconnecting
```

try changing the channel bandwidth to `20MHz` through your router's settings page.

Cause #3

On some laptop models with hardware rfkill switches (e.g., Thinkpad X200 series), due to wear or bad design, the switch (or its connection to the mainboard) might become loose over time resulting in seemingly random hardblocks/disconnects when you accidentally touch the switch or move the laptop. There is no software solution to this, unless your switch is electrical and the BIOS offers the option to disable the switch. If your switch is mechanical (most are), there are lots of possible solutions, most of which aim to disable the switch: Soldering the contact point on the mainboard/wifi-card, glueing or blocking the switch, using a screw nut to tighten the switch or removing it altogether.

Troubleshooting drivers and firmware

This section covers methods and procedures for installing kernel modules and *firmware* for specific chipsets, that differ from generic method.

See Kernel modules for general informations on operations with modules.

Ralink

rt2x00

Unified driver for Ralink chipsets (it replaces `rt2500`, `rt61`, `rt73`, etc). This driver has been in the Linux kernel since 2.6.24, you only need to load the right module for the chip: `rt2400pci`, `rt2500pci`, `rt2500usb`, `rt61pci` or `rt73usb` which will autoload the respective `rt2x00` modules too.

A list of devices supported by the modules is available at the project's homepage (http://rt2x00.serialmonkey.com/wiki/index.php/Hardware).

Additional notes

- Since kernel 3.0, rt2x00 includes also these drivers: `rt2800pci`, `rt2800usb`.
- Since kernel 3.0, the staging drivers `rt2860sta` and `rt2870sta` are replaced by the mainline drivers `rt2800pci` and `rt2800usb` [2] (https://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=fefecc6989b4b24276797270c0e229c07be02ad3).
- Some devices have a wide range of options that can be configured with `iwpriv`. These are documented in the source tarballs (http://web.ralinktech.com/ralink/Home/Support/Linux.html) available from Ralink.

rt3090

For devices which are using the rt3090 chipset it should be possible to use `rt2800pci` driver, however, is not working with this chipset very well (e.g. sometimes it's not possible to use higher rate than 2Mb/s).

The best way is to use the `rt3090-dkms` (https://aur.archlinux.org/packages/rt3090-dkms/) driver from AUR. Make sure to blacklist the `rt2800pci` module and setup the `rt3090sta` module to load at boot.

> **Note:** This driver also works with rt3062 chipsets. Also the `rt3090` (https://aur.archlinux.org/packages/rt3090/) package is not supported by the latest kernel and has been orphaned `rt3090-dkms` (https://aur.archlinux.org/packages/rt3090-dkms/) should be used instead.

rt3290

The rt3290 chipset is recognised by the kernel `rt2800pci` module. However, some users experience problems and reverting to a patched Ralink driver seems to be beneficial in these cases (https://bbs.archlinux.org/viewtopic.php?id=161952).

rt3573

New chipset as of 2012. It may require proprietary drivers from Ralink. Different manufacturers use it, see the Belkin N750 DB wireless usb adapter (https://bbs.archlinux.org/viewtopic.php?pid=1164228#p1164228) forums thread.

rt5572

New chipset as of 2012 with support for 5 Ghz bands. It may require proprietary drivers from Ralink and some effort to compile them. At the time of writing a how-to on compilation is available for a DLINK DWA-160 rev. B2 here (http://bernaerts.dyndns.org/linux/229-ubuntu-precise-dlink-dwa160-revb2).

Realtek

rtl8192cu

The driver is now in the kernel, but many users have reported being unable to make a connection although scanning for networks does work.

Package `8192cu-dkms (https://aur.archlinux.org/packages/8192cu-dkms/)` in the AUR includes many patches, try this if it doesn't work fine with the driver in kernel.

rtl8192e

The driver is part of the current kernel package. The module initialization may fail at boot giving this error message:

```
rtl819xE:ERR in CPUcheck_firmware_ready()
rtl819xE:ERR in init_firmware() step 2
rtl819xE:ERR!!! _rtl8192_up(): initialization is failed!
r8169 0000:03:00.0: eth0: link down
```

A workaround is to simply unload the module:

```
# modprobe -r r8192e_pci
```

and reload the module (after a pause):

```
# modprobe r8192e_pci
```

rtl8188eu

Some dongles, like the TP-Link TL-WN725N v2 (not sure, but it seems that uses the rtl8179 chipset), use chipsets compatible with this driver. In Linux 3.12 the driver has been moved (http://lwn.net/Articles/564798/) to kernel staging source tree. For older kernels use out-of-tree driver sources built with dkms - install the `dkms-8188eu (https://aur.archlinux.org/packages/dkms-8188eu/)` package from AUR. At the times of 3.15 kernel rtl8188eu driver is buggy and has many stability issues.

rtl8723be

The `rtl8723be` module is included in the mainline Linux kernel since version 3.15.

Some users may encounter errors with powersave on this card. This is shown with occasional disconnects that are not recognized by high level network managers (netctl, NetworkManager). This error can be confirmed by running `dmesg -w` or `journalctl -f` and looking for output related to powersave and the `rtl8723be` module. If you are having this issue, use the `fwlps=0` kernel option, which should prevent the WiFi card from automatically sleeping and halting connection.

/etc/modprobe.d/rtl8723be.conf

```
options rtl8723be fwlps=0
```

Atheros

The MadWifi team (http://madwifi-project.org/) currently maintains three different drivers for devices with Atheros chipset:

- `madwifi` is an old, obsolete driver. Not present in Arch kernel since 2.6.39.1[3] (https://mailman.archlinux.org/pipermail/arch-dev-public/2011-June/020669.html).
- `ath5k` is newer driver, which replaces the `madwifi` driver. Currently a better choice for some chipsets, but not all chipsets are supported (see below)
- `ath9k` is the newest of these three drivers, it is intended for newer Atheros chipsets. All of the chips with 802.11n capabilities are supported.

There are some other drivers for some Atheros devices. See Linux Wireless documentation (http://wireless.kernel.org/en/users/Drivers/Atheros#PCI_.2F_PCI-E_.2F_AHB_Drivers) for details.

ath5k

External resources:

- http://wireless.kernel.org/en/users/Drivers/ath5k
- http://wiki.debian.org/ath5k

If you find web pages randomly loading very slow, or if the device is unable to lease an IP address, try to switch from hardware to software encryption by loading the `ath5k` module with `nohwcrypt=1` option. See Kernel Modules#Setting module options for details.

Some laptops may have problems with their wireless LED indicator flickering red and blue. To solve this problem, do:

```
# echo none > /sys/class/leds/ath5k-phy0::tx/trigger
# echo none > /sys/class/leds/ath5k-phy0::rx/trigger
```

For alternatives, see this bug report (https://bugzilla.redhat.com/show_bug.cgi?id=618232).

ath9k

External resources:

- http://wireless.kernel.org/en/users/Drivers/ath9k
- http://wiki.debian.org/ath9k

As of Linux 3.15.1, some users have been experiencing a decrease in bandwidth. In some cases this can fixed by editing `/etc/modprobe.d/ath9k.conf` and adding the line:

```
options ath9k nohwcrypt=1
```

> **Note:** Check with the command lsmod what module(-name) is in use and change it if named otherwise (e.g. ath9k_htc).

In the unlikely event that you have stability issues that trouble you, you could try using the `backports-patched (https://aur.archlinux.org/packages/backports-patched/)` package. An ath9k mailing list (https://lists.ath9k.org/mailman/listinfo/ath9k-devel) exists for support and development related discussions.

Power saving

Although Linux Wireless (http://wireless.kernel.org/en/users/Documentation/dynamic-power-save) says that dynamic power saving is enabled for Atheros ath9k single-chips newer than AR9280, for some devices (e.g. AR9285) powertop (https://www.archlinux.org/packages/?name=powertop) might still report that power saving is disabled. In this case enable it manually.

On some devices (e.g. AR9285), enabling the power saving might result in the following error:

```
# iw wlan0 set power_save on
command failed: Operation not supported (-95)
```

The solution is to set the `ps_enable=1` option for the `ath9k` module:

```
/etc/modprobe.d/ath9k.conf
options ath9k ps_enable=1
```

ASUS

With some ASUS laptops (tested with ASUS U32U series), it could help to add `options asus_nb_wmi wapf=1` to `/etc/modprobe.d/asus_nb_wmi.conf` to fix rfkill-related issues.

You can also try to blacklist the module asus_nb_wmi (tested with ASUSPRO P550C):

```
# echo "blacklist asus_nb_wmi" >> /etc/modprobe.d/blacklist.conf
```

Intel

ipw2100 and ipw2200

These modules are fully supported in the kernel, but they require additional firmware. Depending on which of the chipsets you have, install either ipw2100-fw (https://www.archlinux.org/packages/?name=ipw2100-fw) or ipw2200-fw (https://www.archlinux.org/packages/?name=ipw2200-fw). Then reload the appropriate module.

Tip: You may use the following module options:

- use the `rtap_iface=1` option to enable the radiotap interface
- use the `led=1` option to enable a front LED indicating when the wireless is connected or not

iwlegacy

iwlegacy (http://wireless.kernel.org/en/users/Drivers/iwlegacy) is the wireless driver for Intel's 3945 and 4965 wireless chips. The firmware is included in the linux-firmware (https://www.archlinux.org/packages/?name=linux-firmware) package.

udev should load the driver automatically, otherwise load `iwl3945` or `iwl4965` manually. See Kernel modules#Loading for details.

iwlwifi

iwlwifi (http://wireless.kernel.org/en/users/Drivers/iwlwifi) is the wireless driver for Intel's current wireless chips, such as 5100AGN, 5300AGN, and 5350AGN. See the full list of supported devices (http://wireless.kernel.org/en/users/Drivers/iwlwifi#Supported_Devices). The firmware is included in the linux-firmware (https://www.archlinux.org/packages/?name=linux-firmware) package.

If you have problems connecting to networks in general or your link quality is very poor, try to disable 802.11n, and perhaps also enable software encryption:

/etc/modprobe.d/iwlwifi.conf

```
options iwlwifi 11n_disable=1
options iwlwifi swcrypto=1
```

If you have a problem with slow uplink speed in 802.11n mode, for example 20Mbps, try to enable antenna aggregation:

/etc/modprobe.d/iwlwifi.conf

```
options iwlwifi 11n_disable=8
```

Do not be confused with the option name, when the value is set to `8` it does not disable anything but re-enables transmission antenna aggregation.[4] (http://forums.gentoo.org/viewtopic-t-996692.html?sid=81bdfa435c089360bdfd9368fe0339a9) [5] (https://bugzilla.kernel.org/show_bug.cgi?id=81571)

In case this does not work for you, you may try disabling power saving for your wireless adapter. For a permanent solution, add a new udev rule:

/etc/udev/rules.d/80-iwlwifi.rules

```
ACTION=="add", SUBSYSTEM=="net", ATTR{address}=="your_mac_address", RUN+="/usr/bin/iw dev %k set power_save o
```

Some (http://ubuntuforums.org/showthread.php?t=2183486&p=12845473#post12845473) have never gotten this to work. Others (http://ubuntuforums.org/showthread.php?t=2205733&p=12935783#post12935783) found salvation by disabling N in their router settings after trying everything. This is known to have be the only solution on more than one occasion. The second link there mentions a 5ghz option that might be worth exploring.

Note: The linux-lts (https://www.archlinux.org/packages/?name=linux-lts)-3.14 kernel may take several minutes to load the firmware and make the wireless card ready for use. The issue is reported to be fixed in linux (https://www.archlinux.org/packages/?name=linux)-3.17 kernel.[6] (https://bbs.archlinux.org/viewtopic.php?id=190757)

Disabling LED blink

Note: This works with the `iwlegacy` and `iwlwifi` drivers.

The default settings on the module are to have the LED blink on activity. Some people find this extremely annoying. To have the LED on solid when Wi-Fi is active, you can use the systemd-tmpfiles:

/etc/tmpfiles.d/phy0-led.conf

```
w /sys/class/leds/phy0-led/trigger - - - - phy0radio
```

Run `systemd-tmpfiles --create phy0-led.conf` for the change to take effect, or reboot.

To see all the possible trigger values for this LED:

```
# cat /sys/class/leds/phy0-led/trigger
```

Tip: If you do not have `/sys/class/leds/phy0-led`, you may try to use the `led_mode="1"` module option. It should be valid for both `iwlwifi` and `iwlegacy` drivers.

Broadcom

See Broadcom wireless.

Other drivers/devices

Tenda w322u

Treat this Tenda card as an `rt2870sta` device. See #rt2x00.

orinoco

This should be a part of the kernel package and be installed already.

Some Orinoco chipsets are Hermes II. You can use the `wlags49_h2_cs` driver instead of `orinoco_cs` and gain WPA support. To use the driver, blacklist `orinoco_cs` first.

prism54

The driver `p54` is included in kernel, but you have to download the appropriate firmware for your card from this site (http://linuxwireless.org/en/users/Drivers/p54#firmware) and install it into the `/usr/lib/firmware` directory.

Note: There's also older, deprecated driver `prism54`, which might conflict with the newer driver (`p54pci` or `p54usb`). Make sure to blacklist `prism54`.

ACX100/111

Warning: The drivers for these devices are broken (https://mailman.archlinux.org/pipermail/arch-dev-public/2011-June/020669.html) and do not work with newer kernel versions.

Packages: `tiacx` `tiacx-firmware` (deleted from official repositories and AUR)

See official wiki (http://sourceforge.net/apps/mediawiki/acx100/index.php?title=Main_Page) for details.

zd1211rw

`zd1211rw` (http://zd1211.wiki.sourceforge.net/) is a driver for the ZyDAS ZD1211 802.11b/g USB WLAN chipset, and it is included in recent versions of the Linux kernel. See [7] (http://www.linuxwireless.org/en/users/Drivers/zd1211rw/devices) for a list of supported devices. You only need to install the firmware for the device, provided by the `zd1211-firmware` (https://www.archlinux.org/packages/?name=zd1211-firmware) package.

hostap_cs

Host AP (http://hostap.epitest.fi/) is a Linux driver for wireless LAN cards based on Intersil's Prism2/2.5/3 chipset. The driver is included in Linux kernel.

> **Note:** Make sure to blacklist the `orinico_cs` driver, it may cause problems.

ndiswrapper

Ndiswrapper is a wrapper script that allows you to use some Windows drivers in Linux. You will need the `.inf` and `.sys` files from your Windows driver. Be sure to use drivers appropriate to your architecture (x86 vs. x86_64).

> **Tip:** If you need to extract these files from an `*.exe` file, you can use `cabextract` (https://www.archlinux.org/packages/?name=cabextract).

Follow these steps to configure ndiswrapper.

1. Install `ndiswrapper` (https://aur.archlinux.org/packages/ndiswrapper/) package from the AUR

2. Install the driver to `/etc/ndiswrapper/*`

```
# ndiswrapper -i filename.inf
```

3. List all installed drivers for ndiswrapper

```
$ ndiswrapper -l
```

4. Let ndiswrapper write its configuration in `/etc/modprobe.d/ndiswrapper.conf` :

```
# ndiswrapper -m
# depmod -a
```

Now the ndiswrapper install is almost finished; follow the instructions on Kernel modules#Loading to automatically load the module at boot.

The important part is making sure that ndiswrapper exists on this line, so just add it alongside the other modules. It would be best to test that ndiswrapper will load now, so:

```
# modprobe ndiswrapper
# iwconfig
```

and *wlan0* should now exist. If you have problems, some help is available at: ndiswrapper howto (http://sourceforge.net/p/ndiswrapper/ndiswrapper/HowTos/) and ndiswrapper FAQ (http://sourceforge.net/p/ndiswrapper/ndiswrapper/FAQ/).

compat-drivers-patched

Patched compat wireless drivers correct the "fixed-channel -1" issue, whilst providing better injection. Please install the compat-drivers-patched (https://aur.archlinux.org/packages/compat-drivers-patched/) package from the AUR.

compat-drivers-patched (https://aur.archlinux.org/packages/compat-drivers-patched/) does not conflict with any other package and the modules built reside in /usr/lib/modules/*your_kernel_version*/updates .

These patched drivers come from the Linux Wireless project (http://wireless.kernel.org/) and support many of the above mentioned chips such as:

```
ath5k ath9k_htc carl9170 b43 zd1211rw rt2x00 wl1251 wl12xx ath6kl brcm80211
```

Supported groups:

```
atheros ath iwlagn rtl818x rtlwifi wl12xx atlxx bt
```

It is also possible to build a specific module/driver or a group of drivers by editing the PKGBUILD, particularly uncommenting the **line #46**. Here is an example of building the atheros group:

```
scripts/driver-select atheros
```

Please read the package's PKGBUILD for any other possible modifications prior to compilation and installation.

See also

- The Linux Wireless project (http://wireless.kernel.org/)
- Aircrack-ng guide on installing drivers (http://aircrack-ng.org/doku.php?id=install_drivers)