Wireless network configuration

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.

Contents

Related articles

Network configuration

Software access point

Ad-hoc networking

Internet sharing

Wireless bonding

Network Debugging

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: If the proper module is not loaded by udev on boot, simply **load it manually**. If udev loads more than one driver for a device, the resulting conflict may 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:

- See the table of **existing Linux wireless drivers (https://wireless.wiki.kernel.org/en/users/drivers)** and follow to the specific driver's page, which contains a list of supported devices. There is also a **List of Wi-Fi Device IDs in Linux (https://wikidevi.com/wiki/List_of_Wi-Fi_Device_IDs_in_Linux)**.
- 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.

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

The **#Manual setup** uses command-line tools to manually manage your network interface. The **#Automatic setup** section describes several programs that can be used to automatically manage your wireless interface, some of which include a GUI and all of which include support for network profiles (useful when frequently switching wireless networks, like with laptops).

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. **Install** one of the following:

Note: For WPA/WPA2 encryption, *wpa_supplicant* is required.

- **iw** — *iw* only supports the nl80211 (netlink) standard, the older WEXT (Wireless EXTentions) standard is not supported. If *iw* does not see your card, this may be the reason.

  https://wireless.kernel.org/en/users/Documentation/iw || **iw (https://www.archlinux.org/packages/?name=iw)**

- **wireless_tools** — *wireless_tools* is deprecated but still widely supported. Use this for modules using the WEXT standard.

  http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html || **wireless_tools (https://www.archlinux.org/packages/?name=wireless_tools)**

- **WPA supplicant** — *wpa_supplicant* is a cross-platform supplicant with support for WEP, WPA and WPA2 (IEEE 802.11i / RSN (Robust Secure Network)). It works with both WEXT and nl80211.

  http://hostap.epitest.fi/wpa_supplicant/ || **`wpa_supplicant (https://www.archlinux.org/packages/?name=wpa_supplicant)`**

Tip: For a comparison of commands between *iw* and *wireless_tools*, see **#iw and wireless_tools comparison**.

Note:

- 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.
- 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 **#Connect to an access point**.

Examples in this section assume that your wireless device interface is `interface` and that you are connecting to `your_essid` wifi access point. Replace both accordingly.

Get the name of the interface

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

To get the name of your wireless interface do:

```
$ iw dev
```

The name of the interface will be output after the word "Interface". For example, it is commonly `wlan0`.

Get the status of the interface

To check link status, use following command.

```
$ iw dev interface link
```

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

```
$ iw dev interface station dump
```

Activate the interface

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 interface 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 interface
```

```
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`.

Discover access points

To see what access points are available:

```
# iw dev interface scan | less
```

Note: If it displays `Interface does not 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 will 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). See **WPA2 Enterprise** and **Wikipedia:Authentication protocol** for details.

- If you see neither `RSN` nor `WPA` blocks but there is `Privacy`, then WEP is used.

Set operating mode

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 interface set type ibss
```

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

Connect to an access point

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 interface 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 interface 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 interface connect "your_essid" key d:2:your_key
```

- **WPA/WPA2** - See **WPA supplicant#Connecting with wpa_passphrase**.

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

```
# iw dev interface link
```

Get an IP address

Follow the instructions in **Network configuration#Manual assignment** for more information on the following examples.

Example

Here is a complete example of setting up a wireless network with WPA supplicant and DHCP.

```
# ip link set dev wlan0 up
# wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf
# dhcpcd wlan0
```

And then to close the connection, you can simply disable the interface:

```
# ip link set dev wlan0 down
```

For a static IP, you would replace the *dhcpcd* invocation with

```
# ip addr add 192.168.0.10/24 broadcast 192.168.0.255 dev wlan0
# ip route add default via 192.168.0.1
```

And before disabling the interface you would first flush the IP address and gateway:

```
# ip addr flush dev wlan0
# ip route flush dev wlan0
```

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)	Archiso [1] (https://git.archlinux.org/archiso.git/tree/configs/releng/packages.both)	Official GUI	Console tools	Systemd units
ConnMan	Yes	Yes	Yes	No	No	`connmanctl`	`connman.service`
netctl	Yes	Yes	Yes	Yes (`base` (https://www.archlinux.org/groups/x86_64/base/))	No	`netctl`, `wifi-menu`	`netctl-auto@`*`interface`*`.service`
NetworkManager	Yes	Yes	Yes	No	Yes	`nmcli`, `nmtui`	`NetworkManager.service`
Wicd	Yes	Yes	No	No	Yes	`wicd-curses`	`wicd.service`
Wifi Radar	Yes	Yes	No	No	Yes	`wifi-radar`	

See also **List of applications/Internet#Network managers**.

WPA2 Enterprise

WPA2 Enterprise is a mode of **Wi-Fi Protected Access**. It provides better security and key management than *WPA2 Personal*, and supports other enterprise-type functionality, such as VLANs and **NAP**. However, it requires an external authentication server, called **RADIUS**

server to handle the authentication of users. This is in contrast to Personal mode which does not require anything beyond the wireless router or access points (APs), and uses a single passphrase or password for all users.

The Enterprise mode enables users to log onto the Wi-Fi network with a username and password and/or a digital certificate. Since each user has a dynamic and unique encryption key, it also helps to prevent user-to-user snooping on the wireless network, and improves encryption strength.

This section describes the configuration of **network clients** to connect to a wireless access point with WPA2 Enterprise mode. See **Software access point#RADIUS** for information on setting up an access point itself.

Note: Enterprise mode requires a more complex client configuration, whereas Personal mode only requires entering a passphrase when prompted. Clients likely need to install the server's CA certificate (plus per-user certificates if using EAP-TLS), and then manually configure the wireless security and 802.1X authentication settings.

For a comparison of protocols see the following **table (http://deployingradius.com/documents/protocols/compatibility.html)**.

Warning: It is possible to use WPA2 Enterprise without the client checking the server CA certificate. However, you should always seek to do so, because without authenticating the access point the connection can be subject to a man-in-the-middle attack. This may happen

because while the connection handshake itself may be encrypted, the most widely used setups transmit the password itself either in plain text or the easily breakable **#MS-CHAPv2**. Hence, the client might send the password to a malicious access point which then proxies the connection.

eduroam

eduroam is an international roaming service for users in research, higher education and further education, based on WPA2 Enterprise.

Note:

- Check connection details **first** with your institution before applying any profiles listed in this section. Example profiles are not guaranteed to work or match any security requirements.
- When storing connection profiles unencrypted, it is recommended restrict read access to the root account by specifying `chmod 600 profile` as root.

Tip: Configuration for **NetworkManager** and **#wpa_supplicant** can be generated with the **eduroam Configuration Assistant Tool (https://cat.eduroam.org/)**.

Manual/automatic setup

wpa_supplicant

WPA supplicant can be configured directly and used in combination with a DHCP client or with systemd. See the examples in `/usr/share/doc/wpa_supplicant/wpa_supplicant.conf` for configuring the connection details.

NetworkManager

NetworkManager can generate WPA2 Enterprise profiles with **graphical front ends**. *nmcli* and *nmtui* do not support this, but may use existing profiles.

connman

ConnMan needs a separate configuration file before **connecting** to the network. See **connman-service.config(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/connman-service.config.5)** and **Connman#Connecting to eduroam** for details.

netctl

netctl supports **#wpa_supplicant** configuration through blocks included with `WPAConfigSection=`. See **netctl.profile(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/netctl.profile.5)** for details.

Warning: Special quoting rules apply: see the `SPECIAL QUOTING RULES` section in **netctl.profile(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/netctl.profile.5)**.

Tip: Custom certificates can be specified by adding the line `'ca_cert="/path/to/special/certificate.cer"'` in `WPAConfigSection`.

Troubleshooting

MS-CHAPv2

WPA2-Enterprise wireless networks demanding MSCHAPv2 type-2 authentication with PEAP sometimes require **pptpclient (https://www.archlinux.org/packages/?name=pptpclient)** in addition to the stock **ppp (https://www.archlinux.org/packages/?name=ppp)** package. **netctl** seems to work out of the box without ppp-mppe, however. In either case, usage of MSCHAPv2 is discouraged as it is highly vulnerable, although using another method is usually not an option. See also **[2] (https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/)** and **[3] (http://research.edm.uhasselt.be/~bbonne/docs/robyns14wpa2enterprise.pdf)**.

Tips and tricks

Respecting the regulatory domain

The **regulatory domain**, 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**. 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** for more detailed information.

Regdomains also affect the limit on the **effective isotropic radiated power (EIRP)** from wireless devices. This is derived from transmit power/"tx power", and is measured in **dBm/mBm (1dBm=100mBm) or mW (log scale)**. 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 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/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` as **module options**. 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/)**.

iw and wireless_tools comparison

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.

***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.
	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.

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**.

Temporary internet access

If you have problematic hardware and need internet access to, for example, download some software or get help in forums, you can make use of Android's built-in feature for internet sharing via USB cable. See **Android tethering#USB tethering** for more information.

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*. 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**.

Hardware buttons to toggle wireless cards are handled by a vendor specific **kernel module**, frequently these are **WMI (https://lwn.net/Articles/391230/)** modules. Particularly for very new hardware models, it happens that the model is not fully supported in the latest stable kernel yet. In this case it often helps to search the kernel bug tracker for information and report the model to the maintainer of the respective vendor kernel module, if it has not happened already.

See also **[4] (http://askubuntu.com/questions/62166/siocsifflags-operation-not-possible-due-to-rf-kill)**.

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. Maybe it also helps you to look at the control message **flowchart (https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/auth-assoc-deauth)**, the journal messages will follow it.

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.

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 the **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 (specify `off` instead of `on`).

- 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.

Valid IP address but cannot resolve host

If you are on a public wireless network that may have a **captive portal**, make sure to query an HTTP page (not an HTTPS page) from your web browser, as some captive portals only redirect HTTP. If this is not the issue, it may be necessary to remove any custom DNS servers from **resolv.conf**.

Setting RTS and fragmentation thresholds

Wireless hardware disables RTS and fragmentation by default. These are two different methods of increasing throughput at the expense of bandwidth (i.e. reliability at the expense of speed). These are useful in environments with wireless noise or many adjacent access points, which may create interference leading to timeouts or failing connections.

Packet fragmentation improves throughput by splitting up packets with size exceeding the fragmentation threshold. The maximum value (2346) effectively disables fragmentation since no packet can exceed it. The minimum value (256) maximizes throughput, but may carry a significant bandwidth cost.

```
# iw phy0 set frag 512
```

RTS improves throughput by performing a handshake with the access point before transmitting packets with size exceeding the RTS threshold. The maximum threshold (2347) effectively disables RTS since no packet can exceed it. The minimum threshold (0) enables RTS for all packets, which is probably excessive for most situations.

```
# iw phy0 set rts 500
```

Note: `phy0` is the name of the wireless device as listed by `$ iw phy` .

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**[5] (http://us.generation-nt.com/answer/gentoo-user-wireless-deauthen**

ticating-by-local-choice-help-204640041.html). Try disabling the wireless card's **power saving** features (specify `off` instead of `on`).

If your card does not support enabling/disabling power save mode, 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.

Cause #4

Another cause for frequent disconnects or a complete failure to connect may also be a sub-standard router, incomplete settings of the router, or interference by other wireless devices.

To troubleshoot, first best try to connect to the router with no authentication.

If that works, enable WPA/WPA2 again but choose fixed and/or limited router settings. For example:

- If the router is considerably older than the wireless device you use for the client, test if it works with setting the router to one wireless mode
- Disable mixed-mode authentication (e.g. only WPA2 with AES, or TKIP if the router is old)
- Try a fixed/free channel rather than "auto" channel (maybe the router next door is old and interfering)
- Disable WPS
- Disable `40Mhz` channel bandwidth (lower throughput but less likely collisions)
- If the router has quality of service settings, check completeness of settings (e.g. Wi-Fi Multimedia (WMM) is part of optional QoS flow control. An erroneous router firmware

may advertise its existence although the setting is not enabled)

Wi-Fi networks invisible because of incorrect regulatory domain

If the computer's Wi-Fi channels do not match those of the user's country, that may result in some in-range Wi-Fi networks becoming invisible, because they use wireless channels that aren't allowed by default. The solution is to configure the regulatory domain correctly, see **#Respecting the regulatory domain**.

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/Mediatek

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)**[**dead link** 2016-08-02].

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` **[6] (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 is not possible to use higher rate than 2Mb/s).

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)**.

mt7612u

New chipset as of 2014, released under their new commercial name Mediatek. It is an AC1200 or AC1300 chipset. Manufacturer provides drivers for Linux on their **support page (https://www.mediatek.com/products/broadbandWifi/mt7612u)**

Realtek

See [7] (https://wikidevi.com/wiki/Realtek) for a list of Realtek chipsets and specifications.

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.

`8192cu-dkms (https://aur.archlinux.org/packages/8192cu-dkms/)`[AUR] includes many patches, try this if it does not work fine with the driver in kernel.

rtl8723ae/rtl8723be

The `rtl8723ae` and `rtl8723be` modules are included in the mainline Linux kernel.

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 `rtl8723ae` / `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/rtl8723ae.conf
```

```
options rtl8723ae fwlps=0
```

or

```
/etc/modprobe.d/rtl8723be.conf
```

```
options rtl8723be fwlps=0
```

If you have poor signal, perhaps your device has only one physical antenna connected, and antenna autoselection is broken. You can force the choice of antenna with `ant_sel=1` or `ant_sel=2` kernel option. **[8] (https://bbs.archlinux.org/viewtopic.php?id=208472)**

rtl88xxau

Realtek chipsets rtl8811au/rtl8812au/rtl8814au/rtl8821au designed for various USB adapters ranging from AC600 to AC1900.

Several packages provide the kernel drivers:

Chipset	Driver version	AUR package	Notes
rtl8812au	5.2.9.3	rtl8812au-dkms-git (https://aur.archlinux.org/packages/rtl8812au-dkms-git/)[AUR]	Latest driver version for rtl8812au only
rtl8811au, rtl8812au and rtl8821au	5.1.5	rtl8821au-dkms-git (https://aur.archlinux.org/packages/rtl8821au-dkms-git/)[AUR]	Works, for rtl8812au latest version is recommended instead
rtl8814au	4.3.21	rtl8814au-dkms-git (https://aur.archlinux.org/packages/rtl8814au-dkms-git/)[AUR]	Possibly works for rtl8813au too

These require **DKMS** so make sure you have your proper kernel headers installed.

rtl8822bu

rtl8822bu-dkms-git (https://aur.archlinux.org/packages/rtl8822bu-dkms-git/)[AUR] provides a kernel module for the Realtek 8822bu chipset found in the Edimax EW7822ULC USB3 and Asus AC53 Nano USB 802.11ac adapter.

This requires **DKMS**, so make sure you have your proper kernel headers installed.

rtl8xxxu

Issues with the `rtl8xxxu` mainline kernel module may be solved by compiling a third-party module for the specific chipset. The source code can be found in **GitHub repositories (https://github.com/lwfinger?tab=repositories)**.

Some drivers may be already prepared in the AUR, e.g. **rtl8723bu-git (https://aur.archlinux.org/packages/rtl8723bu-git/)**[AUR] and **rtl8723bu-git-dkms (https://aur.archlinux.org/packages/rtl8723bu-git-dkms/)**[AUR].

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 **[9] (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/)** AUR 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 dev 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
```

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** for details.

If you have problems connecting to networks in general, random failures with your card on bootup or your link quality is very poor, try to disable 802.11n:

```
/etc/modprobe.d/iwl4965.conf
```

```
options iwl4965 11n_disable=1
```

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. The **linux-firmware-iwlwifi-git (https://aur.archlinux.org/packages/linux-firmware-iwlwifi-git/)**[AUR] may contain some updates sooner.

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 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.**[10] (http://forums.gentoo.org/viewtopic-t-996692.html?sid=81bdfa435c089360bdfd9368fe0339a9) [11] (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.

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.

Bluetooth coexistence

If you have difficulty connecting a bluetooth headset and maintaining good downlink speed, try disabling bluetooth coexistence **[12] (https://wireless.wiki.kernel.org/en/users/Drivers/iwlwifi#wifibluetooth_coexistence)**:

```
/etc/modprobe.d/iwlwifi.conf
```

```
options iwlwifi bt_coex_active=0
```

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 is 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 [13] (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.

Warning: 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-dkms (https://www.archlinux.org/packages/?name=ndiswrapper-dkms)**

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#Automatic module handling** 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/)**.

backports-patched

backports-patched (https://aur.archlinux.org/packages/backports-patched/)[AUR] provide drivers released on newer kernels backported for usage on older kernels. The project started since 2007 and was originally known as compat-wireless, evolved to compat-drivers and was recently renamed simply to backports.

If you are using old kernel and have wireless issue, drivers in this package may help.

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)**

Retrieved from "https://wiki.archlinux.org/index.php?title=Wireless_network_configuration&oldid=509963"