

OpenWRT – Kamikaze

Configuration



Document original :

<http://downloads.openwrt.org/kamikaze/docs/openwrt.html>

1 [The Router](#)

1.1 [Getting started](#)

1.1.1 [Installation](#)

1.1.2 [Initial configuration](#)

1.1.3 [Failsafe mode](#)

1.2 [Configuring OpenWrt](#)

1.2.1 [Network](#)

1.2.2 [Wireless](#)

1.3 [Advanced configuration](#)

1.3.1 [Hotplug](#)

1.3.2 [Init scripts](#)

1.3.3 [Network scripts](#)

2 [Development issues](#)

2.1 [The build system](#)

2.1.1 [Building an image](#)

2.1.2 [Creating packages](#)

2.1.3 [Creating kernel modules packages](#)

2.1.4 [Conventions](#)

2.1.5 [Troubleshooting](#)

2.1.6 [Using build environments](#)

2.2 [Extra tools](#)

2.2.1 [Image Builder](#)

2.2.2 [SDK](#)

2.3 [Adding platform support](#)

2.3.1 [Which Operating System does this device run?](#)

2.3.2 [Finding and using the manufacturer SDK](#)

2.4 [Debugging and debricking](#)

2.4.1 [Adding a serial port](#)

2.4.2 [JTAG](#)

2.5 [Reporting bugs](#)

2.5.1 [Using the Trac ticket system](#)

2.6 [Submitting patches](#)

2.6.1 [How to contribute](#)

2.6.2 [Where to listen and talk](#)

2.6.3 [Patch Submission Process](#)

Chapter 1 : The Router

1.2 Configuring OpenWrt

1.2.1 Network

The network configuration in Kamikaze is stored in `/etc/config/network` and is divided into interface configurations. Each interface configuration either refers directly to an ethernet/wifi interface (`eth0`, `wl0`, ..) or to a bridge containing multiple interfaces. It looks like this:

```
config interface "lan"
    option ifname "eth0"
    option proto "static"
    option ipaddr "192.168.1.1"
    option netmask "255.255.255.0"
    option gateway "192.168.1.254"
    option dns "192.168.1.254"
```

`ifname` specifies the Linux interface name. If you want to use bridging on one or more interfaces, set `ifname` to a list of interfaces and add:

```
option type "bridge"
```

It is possible to use VLAN tagging on an interface simply by adding the VLAN IDs to it, e.g. `eth0.1`. These can be nested as well.

This sets up a simple static configuration for `eth0`. `proto` specifies the protocol used for the interface. The default image usually provides `'none'`, `'static'`, `'dhcp'` and `'pppoe'`. Others can be added by installing additional packages.

When using the `'static'` method like in the example, the options `ipaddr` and `netmask` are mandatory, while `gateway` and `dns` are optional. You can specify more than one DNS server, separated with spaces.

DHCP currently only accepts `ipaddr` (IP address to request from the server) and `hostname` (client hostname identify as) - both are optional.

PPP based protocols (`pppoe`, `pptp`, ...) accept these options:

- `username`
The PPP username (usually with PAP authentication)
- `password`
The PPP password
- `keepalive`
Ping the PPP server (using LCP). The value of this option specifies the maximum number of failed pings before reconnecting. The ping interval defaults to 5, but can be changed by appending "`<interval>`" to the `keepalive` value
- `demand`
Use Dial on Demand (value specifies the maximum idle time).
- `server: (pptp)`
The remote pptp server IP

For all protocol types, you can also specify the MTU by using the `mtu` option.

Setting up static routes

You can set up static routes for a specific interface that will be brought up after the interface is configured.

Simply add a config section like this:

```
config route foo
    option interface lan
    option target 1.1.1.0
    option netmask 255.255.255.0
    option gateway 192.168.1.1
```

The name for the route section is optional, the interface, target and gateway options are mandatory. Leaving out the `netmask` option will turn the route into a host route.

Setting up the switch (currently broadcom only)

The switch configuration is set by adding a 'switch' config section. Example:

```
config switch    "eth0"
  option vlan0   "1 2 3 4 5*"
  option vlan1   "0 5"
```

On Broadcom hardware the section name needs to be eth0, as the switch driver does not detect the switch on any other physical device. Every vlan option needs to have the name vlan<n> where <n> is the VLAN number as used in the switch driver. As value it takes a list of ports with these optional suffixes:

- '*' : Set the default VLAN (PVID) of the Port to the current VLAN
- 'u' : Force the port to be untagged
- 't' : Force the port to be tagged

The CPU port defaults to tagged, all other ports to untagged. On Broadcom hardware the CPU port is always 5. The other ports may vary with different hardware.

For instance, if you wish to have 3 vlans, like one 3-port switch, 1 port in a DMZ, and another one as your WAN interface, use the following configuration :

```
config switch    "eth0"
  option vlan0   "1 2 3 5*"
  option vlan1   "0 5"
  option vlan2   "4 5"
```

Three interfaces will be automatically created using this switch layout : eth0.0 (vlan0), eth0.1 (vlan1) and eth0.2 (vlan2). You can then assign those interfaces to a custom network configuration name like lan, wan or dmz for instance.

Setting up IPv6 connectivity

OpenWrt supports IPv6 connectivity using PPP, Tunnel brokers or static assignment.

If you use PPP, IPv6 will be setup using IP6CP and there is nothing to configure.

To setup an IPv6 tunnel to a tunnel broker, you can install the 6scripts package and edit the /etc/config/6tunnel file and change the settings accordingly :

```
config 6tunnel
  option tnlifname  'sixbone'
  option remoteip4  '1.0.0.1'
  option localip4   '1.0.0.2'
  option localip6   '2001::DEAD::BEEF::1'
  option prefix     '/64'
```

- 'tnlifname': Set the interface name of the IPv6 in IPv4 tunnel
- 'remoteip4': IP address of the remote end to establish the 6in4 tunnel. This address is given by the tunnel broker
- 'localip4': IP address of your router to establish the 6in4 tunnel. It will usually match your WAN IP address.
- 'localip6': IPv6 address to setup on your tunnel side This address is given by the tunnel broker
- 'prefix': IPv6 prefix to setup on the LAN.

Using the same package you can also setup an IPv6 bridged connection :

```
config 6bridge
  option bridge 'br6'
```

By default the script bridges the WAN interface with the LAN interface and uses ebtables to filter anything that is not IPv6 on the bridge.

IPv6 static addressing is also supported using a similar setup as IPv4 but with the ip6 prefixing (when applicable).

```
config interface "lan"
  option ifname  "eth0"
  option proto   "static"
  option ip6addr "fe80::200:ff:fe00:0/64"
  option ip6gw   "2001::DEAF:BEE:1"
```

1.2.2 Wireless

The WiFi settings are configured in the file `/etc/config/wireless` (currently supported on Broadcom, Atheros and mac80211). When booting the router for the first time it should detect your card and create a sample configuration file. By default 'option network lan' is commented. This prevents unsecured sharing of the network over the wireless interface.

Each wireless driver has its own configuration script in `/lib/wifi/driver_name.sh` which handles driver specific options and configurations. This script is also calling driver specific binaries like `wlc` for Broadcom, or `hostapd` and `wpa_supplicant` for atheros.

The reason for using such architecture, is that it abstracts the driver configuration.

Generic Broadcom wireless config:

```
config wifi-device "wl0"
    option type "broadcom"
    option channel "5"
```

```
config wifi-iface
    option device "wl0"
# option network lan
    option mode "ap"
    option ssid "OpenWrt"
    option hidden "0"
    option encryption "none"
```

Generic Atheros wireless config:

```
config wifi-device "wifi0"
    option type "atheros"
    option channel "5"
    option hwmode "11g"
```

```
config wifi-iface
    option device "wifi0"
# option network lan
    option mode "ap"
    option ssid "OpenWrt"
    option hidden "0"
    option encryption "none"
```

Generic mac80211 wireless config:

```
config wifi-device "wifi0"
    option type "mac80211"
    option channel "5"
```

```
config wifi-iface
    option device "wlan0"
# option network lan
    option mode "ap"
    option ssid "OpenWrt"
    option hidden "0"
    option encryption "none"
```

Generic multi-radio Atheros wireless config:

```
config wifi-device wifi0
    option type atheros
    option channel 1
```

```
config wifi-iface
    option device wifi0
# option network lan
    option mode ap
    option ssid OpenWrt_private
    option hidden 0
```

```
option encryption none
```

```
config wifi-device wifi1  
option type atheros  
option channel 11
```

```
config wifi-iface  
option device wifi1  
# option network lan  
option mode ap  
option ssid OpenWrt_public  
option hidden 1  
option encryption none
```

There are two types of config sections in this file. The 'wifi-device' refers to the physical wifi interface and 'wifi-iface' configures a virtual interface on top of that (if supported by the driver).

A full outline of the wireless configuration file with description of each field:

```
config wifi-device wifi device name  
option type broadcom, atheros, mac80211  
option country us, uk, fr, de, etc.  
option channel 1-14  
option maxassoc 1-128 (broadcom only)  
option distance 1-n  
option hwmode 11b, 11g, 11a, 11bg (atheros, mac80211)  
option rxantenna 0,1,2 (atheros, broadcom)  
option txantenna 0,1,2 (atheros, broadcom)  
option txpower transmission power in dBm
```

```
config wifi-iface  
option network the interface you want wifi to bridge with  
option device wifi0, wifi1, wifi2, wifiN  
option mode ap, sta, adhoc, monitor, or wds  
option txpower (deprecated) transmission power in dBm  
option ssid ssid name  
option bssid bssid address  
option encryption none, wep, psk, psk2, wpa, wpa2  
option key encryption key  
option key1 key 1  
option key2 key 2  
option key3 key 3  
option key4 key 4  
option server ip address  
option port port  
option hidden 0,1  
option isolate 0,1  
option doth 0,1 (atheros, broadcom)  
option wmm 0,1 (atheros, broadcom)
```

Options for the wifi-device:

- type
The driver to use for this interface.
- country
The country code used to determine the regulatory settings.
- channel
The wifi channel (e.g. 1-14, depending on your country setting).
- maxassoc
Optional: Maximum number of associated clients. This feature is supported only on the broadcom chipset.
- distance
Optional: Distance between the ap and the furthest client in meters. This feature is supported only on the atheros chipset.

- mode
The frequency band (b, g, bg, a). This feature is only supported on the atheros chipset.
- diversity
Optional: Enable diversity for the Wi-Fi device. This feature is supported only on the atheros chipset.
- rxantenna
Optional: Antenna identifier (0, 1 or 2) for reception. This feature is supported by atheros and some broadcom chipsets.
- txantenna
Optional: Antenna identifier (0, 1 or 2) for emission. This feature is supported by atheros and some broadcom chipsets.
- txpower Set the transmission power to be used. The amount is specified in dBm.

Options for the wifi-iface:

- network
Selects the interface section from /etc/config/network to be used with this interface
- device
Set the wifi device name.
- mode
Operating mode:
 - ap
Access point mode
 - sta
Client mode
 - adhoc
Ad-Hoc mode
 - monitor
Monitor mode
 - wds
WDS point-to-point link
- ssid Set the SSID to be used on the wifi device.
- bssid Set the BSSID address to be used for wds to set the mac address of the other wds unit.
- txpower (Deprecated, set in wifi-device) Set the transmission power to be used. The amount is specified in dBm.
- encryption
Encryption setting. Accepts the following values:
 - none
 - wep
 - psk, psk2
WPA(2) Pre-shared Key
 - wpa, wpa2
WPA(2) RADIUS
- key, key1, key2, key3, key4 (wep, wpa and psk)
WEP key, WPA key (PSK mode) or the RADIUS shared secret (WPA RADIUS mode)
- server (wpa)
The RADIUS server ip address
- port (wpa)
The RADIUS server port (defaults to 1812)
- hidden
0 broadcasts the ssid; 1 disables broadcasting of the ssid
- isolate
Optional: Isolation is a mode usually set on hotspots that limits the clients to communicate only with the AP and not with other wireless clients. 0 disables ap isolation (default); 1 enables ap isolation.
- doth
Optional: Toggle 802.11h mode. 0 disables 802.11h (default); 1 enables it.
- wmm
Optional: Toggle 802.11e mode. 0 disables 802.11e (default); 1 enables it.

Wireless Distribution System WDS is a non-standard mode which will be working between two Broadcom devices for instance but not between a Broadcom and Atheros device.

Unencrypted WDS connections This configuration example shows you how to setup unencrypted WDS connections. We assume that the peer configured as below as the BSSID ca:fe:ba:be:00:01 and the remote WDS endpoint ca:fe:ba:be:00:02 (option bssid field).

```
config wifi-device "wl0"  
  option type "broadcom"  
  option channel "5"
```

```
config wifi-iface  
  option device "wl0"  
  option network lan  
  option mode "ap"  
  option ssid "OpenWrt"  
  option hidden "0"  
  option encryption "none"
```

```
config wifi-iface  
  option device "wl0"  
  option network lan  
  option mode wds  
  option ssid "OpenWrt WDS"  
  option bssid "ca:fe:ba:be:00:02"
```

Encrypted WDS connections It is also possible to encrypt WDS connections. psk, psk2 and psk+psk2 modes are supported. Configuration below is an example configuration using Pre-Shared-Keys with AES algorithm.

```
config wifi-device wl0  
  option type broadcom  
  option channel 5
```

```
config wifi-iface  
  option device "wl0"  
  option network lan  
  option mode ap  
  option ssid "OpenWrt"  
  option encryption psk2  
  option key "<key for clients>"
```

```
config wifi-iface  
  option device "wl0"  
  option network lan  
  option mode wds  
  option bssid ca:fe:ba:be:00:02  
  option ssid "OpenWrt WDS"  
  option encryption psk2  
  option key "<psk for WDS>"
```

802.1x configurations OpenWrt supports both 802.1x client and Access Point configurations. 802.1x client is only working with Atheros or mac80211 drivers. Configuration only supports EAP types TLS, TTLS or PEAP.

EAP-TLS

```
config wifi-iface  
  option device "ath0"  
  option network lan  
  option ssid OpenWrt  
  option eap_type tls  
  option ca_cert "/etc/config/certs/ca.crt"  
  option priv_key "/etc/config/certs/priv.crt"  
  option priv_key_pwd "PKCS#12 passphrase"
```

EAP-PEAP

```
config wifi-iface
```

```

option device      "ath0"
option network    lan
option ssid       OpenWrt
option eap_type   peap
option ca_cert    "/etc/config/certs/ca.crt"
option auth       MSCHAPV2
option identity   username
option password   password

```

Limitations: There are certain limitations when combining modes. Only the following mode combinations are supported:

- Broadcom:
 - 1x sta, 0-3x ap
 - 1-4x ap
 - 1x adhoc
 - 1x monitor

WDS links can only be used in pure AP mode and cannot use WEP (except when sharing the settings with the master interface, which is done automatically).

- Atheros:
 - 1x sta, 0-Nx ap
 - 1-Nx ap
 - 1x adhoc

N is the maximum number of VAPs that the module allows, it defaults to 4, but can be changed by loading the module with the maxvaps=N parameter.

Adding a new driver configuration Since we currently only support thread different wireless drivers : Broadcom, Atheros and mac80211, you might be interested in adding support for another driver like Ralink RT2x00, Texas Instruments ACX100/111.

The driver specific script should be placed in /lib/wifi/<driver>.sh and has to include several functions providing :

- detection of the driver presence
- enabling/disabling the wifi interface(s)
- configuration reading and setting
- third-party programs calling (nas, supplicant)

Each driver script should append the driver to a global DRIVERS variable :

```
append DRIVERS "driver name"
```

scan_<driver> This function will parse the /etc/config/wireless and make sure there are no configuration incompatibilities, like enabling hidden SSIDS with ad-hoc mode for instance. This can be more complex if your driver supports a lot of configuration options. It does not change the state of the interface.

Example:

```

scan_dummy() {
  local device="$1"

  config_get vifs "$device" vifs
  for vif in $vifs; do
    # check config consistency for wifi-iface sections
  done
  # check mode combination
}

```

enable_<driver> This function will bring up the wifi device and optionally create application specific configuration files, e.g. for the WPA authenticator or supplicant.

Example:

```

enable_dummy() {
  local device="$1"

  config_get vifs "$device" vifs

```

```

for vif in $vifs; do
# bring up virtual interface belonging to
# the wifi-device "$device"
done
}

```

disable_<driver> This function will bring down the wifi device and all its virtual interfaces (if supported).

Example:

```

disable_dummy() {
local device="$1"

# bring down virtual interfaces belonging to
# "$device" regardless of whether they are
# configured or not. Don't rely on the vifs
# variable at this point
}

```

detect_<driver> This function looks for interfaces that are usable with the driver. Template config sections for new devices should be written to stdout. Must check for already existing config sections belonging to the interfaces before creating new templates.

Example:

```

detect_dummy() {
[ wifi-device = "$(config_get dummydev type)" ] && return 0
cat <<EOF
config wifi-device dummydev
option type dummy
# REMOVE THIS LINE TO ENABLE WIFI:
option disabled 1

config wifi-iface
option device dummydev
option mode ap
option ssid OpenWrt
EOF
}

```

1.3 Advanced configuration

Structure of the configuration files

The config files are divided into sections and options/values.

Every section has a type, but does not necessarily have a name. Every option has a name and a value and is assigned to the section it was written under.

Syntax:

```

config <type> ["<name>"] # Section
option <name> "<value>" # Option

```

Every parameter needs to be a single string and is formatted exactly like a parameter for a shell function. The same rules for Quoting and special characters also apply, as it is parsed by the shell.

Parsing configuration files in custom scripts

To be able to load configuration files, you need to include the common functions with:

```

./etc/functions.sh

```

Then you can use config_load <name> to load config files. The function first checks for <name> as absolute filename and falls back to loading it from /etc/config (which is the most common way of using it).

If you want to use special callbacks for sections and/or options, you need to define the following shell functions before running config_load (after including /etc/functions.sh):

```

config_cb() {
    local type="$1"
    local name="$2"
    # commands to be run for every section
}

```

```

option_cb() {
    # commands to be run for every option
}

```

You can also alter `option_cb` from `config_cb` based on the section type. This allows you to process every single config section based on its type individually.

`config_cb` is run every time a new section starts (before options are being processed). You can access the last section through the `CONFIG_SECTION` variable. Also an extra call to `config_cb` (without a new section) is generated after `config_load` is done. That allows you to process sections both before and after all options were processed.

Another way of iterating on config sections is using the `config_foreach` command.

Syntax:

```

config_foreach <function name> [<sectiontype>] [<arguments...>]

```

This command will run the supplied function for every single config section in the currently loaded config. The section name will be passed to the function as argument 1. If the section type is added to the command line, the function will only be called for sections of the given type.

You can access already processed options with the `config_get` command Syntax:

```

# print the value of the option
config_get <section> <option>

# store the value inside the variable
config_get <variable> <section> <option>

```

In busybox ash the three-option `config_get` is faster, because it does not result in an extra fork, so it is the preferred way.

Additionally you can also modify or add options to sections by using the `config_set` command.

Syntax:

```

config_set <section> <option> <value>

```

If a config section is unnamed, an automatically generated name will be assigned internally, e.g. `cfg1`, `cfg2`, ...

While it is possible, using unnamed sections through these autogenerated names is strongly discouraged. Use callbacks or `config_foreach` instead.

1.3.1 Hotplug

1.3.2 Init scripts

Because OpenWrt uses its own init script system, all init scripts must be installed as `/etc/init.d/name` use `/etc/rc.common` as a wrapper.

Example: `/etc/init.d/httpd`

```

#!/bin/sh /etc/rc.common
# Copyright (C) 2006 OpenWrt.org

START=50
start() {
    [ -d /www ] && httpd -p 80 -h /www -r OpenWrt
}

stop() {
    killall httpd
}

```

as you can see, the script does not actually parse the command line arguments itself. This is done by the wrapper script `/etc/rc.common`.

`start()` and `stop()` are the basic functions, which almost any init script should provide. `start()` is called when the user runs `/etc/init.d/httpd start` or (if the script is enabled and does not override this behavior) at system boot time.

Enabling and disabling init scripts is done by running `/etc/init.d/name enable` or `/etc/init.d/name disable`. This creates or removes symbolic links to the init script in `/etc/rc.d`, which is processed by `/etc/init.d/rcS` at boot time.

The order in which these scripts are run is defined in the variable `START` in the init script. Changing it requires running `/etc/init.d/name enable` again.

You can also override these standard init script functions:

- `boot()`
Commands to be run at boot time. Defaults to `start()`
- `restart()`
Restart your service. Defaults to `stop()`; `start()`
- `reload()`
Reload the configuration files for your service. Defaults to `restart()`

You can also add custom commands by creating the appropriate functions and referencing them in the `EXTRA_COMMANDS` variable. Help text is added in `EXTRA_HELP`.

Example:

```
status() {
    # print the status info
}

EXTRA_COMMANDS="status"
EXTRA_HELP="    status Print the status of the service"
```

1.3.3 Network scripts

Using the network scripts

To be able to access the network functions, you need to include the necessary shell scripts by running:

```
./etc/functions.sh # common functions
include /lib/network # include /lib/network/*.sh
scan_interfaces    # read and parse the network config
```

Some protocols, such as PPP might change the configured interface names at run time (e.g. `eth0 => ppp0` for PPPoE). That's why you have to run `scan_interfaces` instead of reading the values from the config directly. After running `scan_interfaces`, the `'ifname'` option will always contain the effective interface name (which is used for IP traffic) and if the physical device name differs from it, it will be stored in the `'device'` option. That means that running `config_get lan ifname` after `scan_interfaces` might not return the same result as running it before.

After running `scan_interfaces`, the following functions are available:

- `find_config interface`
looks for a network configuration that includes the specified network interface.
- `setup_interface interface [config] [protocol]`
will set up the specified interface, optionally overriding the network configuration name or the protocol that it uses.

Writing protocol handlers

You can add custom protocol handlers by adding shell scripts to `/lib/network`. They provide the following two shell functions:

```
scan_<protocolname>() {
    local config="$1"
    # change the interface names if necessary
}
```

```
setup_interface_<protocolname>() {
```

```
local interface="$1"  
local config="$2"  
# set up the interface  
}
```

scan_protocolname is optional and only necessary if your protocol uses a custom device, e.g. a tunnel or a PPP device.